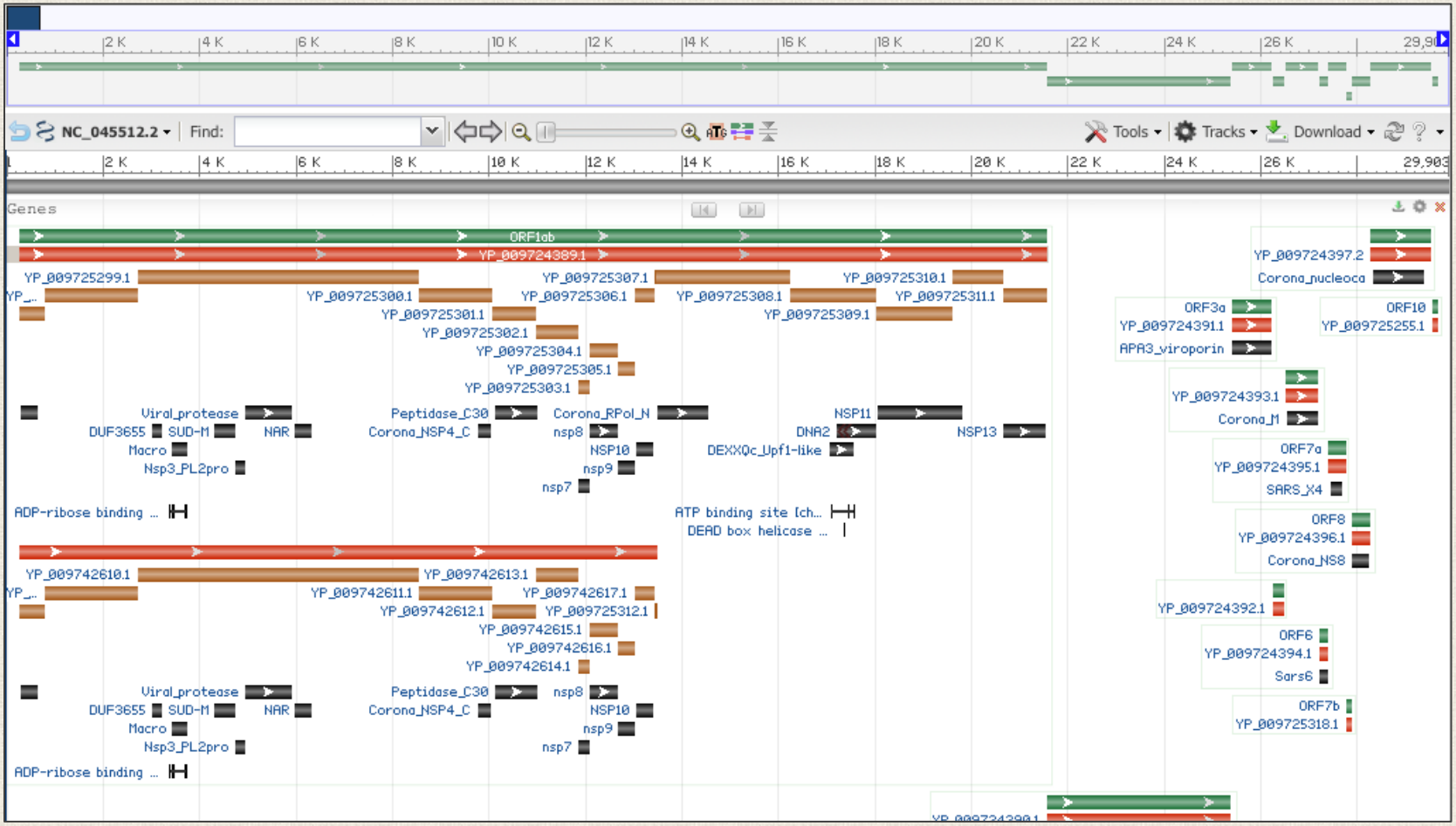
Comparing Genes



INTRODUCTION TO SEQUENCE ALIGNMENT

Comparing Genes is a Fundamental Problem in Biology

Comparing Genes Problem:

- **Input:** Two genes.
- **Output:** How "similar" these genes are.

Goal: Convert this important biological question into a well-defined computational problem.

Try 1: Hamming Distance

Hamming Distance Problem:

- **Input:** Two strings.
- **Output:** The number of "mismatched" symbols in the two strings.

Try 1: Hamming Distance

Hamming Distance Problem:

- **Input:** Two strings.
- **Output:** The number of "mismatched" symbols in the two strings.

```
ATGCATGC
TGCATGCA
```

Hamming distance = 8

Try 1: Hamming Distance

Hamming Distance Problem:

- **Input:** Two strings.
- **Output:** The number of "mismatched" symbols in the two strings.

```
ATGCATGC
TGCATGCA
```

Hamming distance = 8

STOP: What are the issues with this approach?

Try 1: Hamming Distance

Hamming Distance Problem:

- **Input:** Two strings.
- **Output:** The number of "mismatched" symbols in the two strings.

```
ATGCATGC
TGCATGCA
```

Hamming distance = 8

Note: these strings have a long shared substring, it just doesn't line up perfectly.

Try 2: Longest Substring

Longest Shared Substring Problem:

- **Input:** Two strings.
- **Output:** The longest substring shared by both strings.

STOP: What are the weaknesses of using the length of a longest shared substring to represent the similarity between two strings?

Try 2: Longest Substring

Longest Shared Substring Problem:

- **Input:** Two strings.
- **Output:** The longest substring shared by both strings.

Consider the strings AAACAAACAAACAAACAAACAAA and AAAGAAAGAAAGAAAGAAAGAAAGAAA. These strings are very similar, but they don't have a long shared substring in common.

Try 3: Counting Shared *k*-Mers

Instead of finding a longest shared substring of two strings, we will count the number of shared substrings.

For simplicity, we restrict to substrings of the same length; recall that a ***k*-mer** is the term we use in comp bio for a string of length *k*.

Try 3: Counting Shared *k*-Mers

`s1 = ACGTATACACGTAT`

String	Count
ACA	1
ACG	2
ATA	1
CAC	1
CGT	2
GTA	2
TAC	1
TAT	2

`s2 = TATCGGTATATCCTAC`

String	Count
ATA	1
ATC	2
CCT	1
CGG	1
CTA	1
GGT	1
GTA	1
TAC	1
TAT	3
TCC	1
TCG	1

STOP: How should we count the # of shared 3-mers of two strings?

Try 3: Counting Shared *k*-Mers

`s1 = ACGTATACACGTAT`

String	Count
ACA	1
ACG	2
ATA	1
CAC	1
CGT	2
GTA	2
TAC	1
TAT	2

`s2 = TATCGGTATATCCTAC`

String	Count
ATA	1
ATC	2
CCT	1
CGG	1
CTA	1
GGT	1
GTA	1
TAC	1
TAT	3
TCC	1
TCG	1

Take minimum counts for each shared *k*-mer:

1 + 1 + 1 + 2 = 5

Toward a Better Approach

STOP: What similarities do you see in these strings?

```
ATGCTTA
TGCATTAA
```

Toward a Better Approach

STOP: What similarities do you see in these strings?

```
ATGCTTA
TGCATTAA
```

Key Point: we can find similarities if we "slide" the strings, letting symbols shift (but stay in same order).

```
ATGC-TTA-
-TGCATTAA
```

Toward a More Accurate Problem

Symbol Matching Problem:

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any "alignment" of the two strings.

```
ATGC-TTA-
-TGCATTAA
```

Toward a More Accurate Problem

Symbol Matching Problem:

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any "alignment" of the two strings.

Exercise: How many matches can you find if the strings are `ATGTTATA` and `ATCGTCC`? What algorithm did you use?

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A **A**	T G T T A T A T C G T C C	+1

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A **A**	T G T T A T A T C G T C C	+1
A T **A T**	G T T A T A C G T C C	+1

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T – A T C	G T T A T A G T C C	

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T – A T C	G T T A T A G T C C	
A T – G A T C G	T T A T A T C C	+1

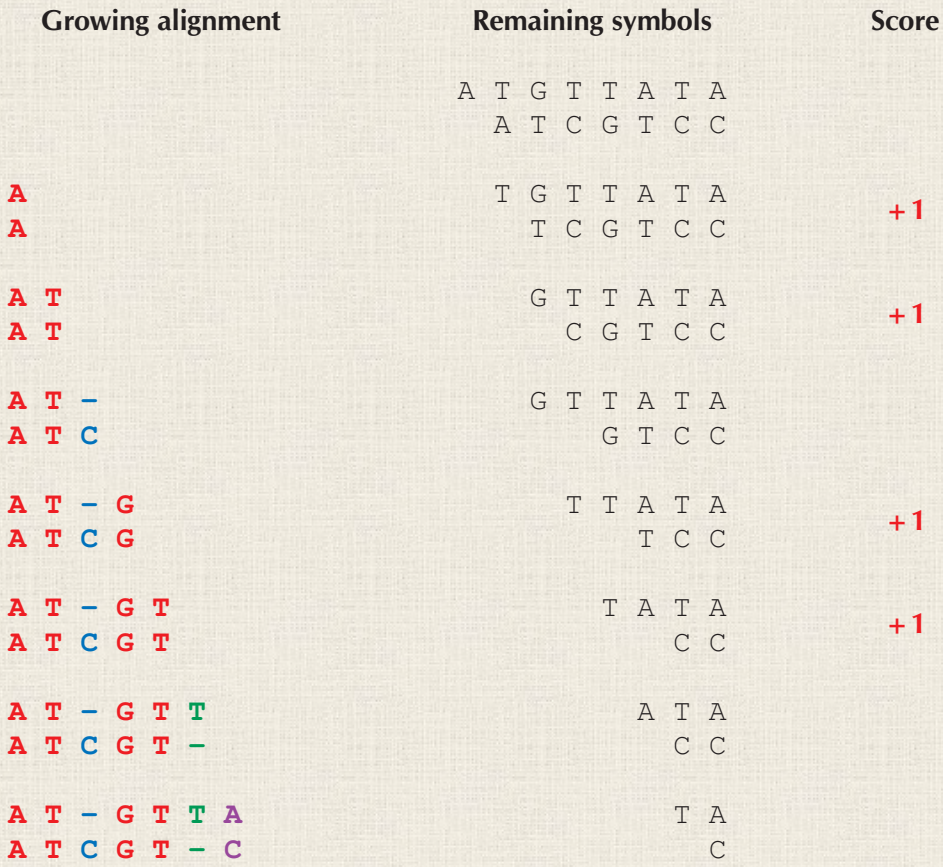
Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T – A T C	G T T A T A G T C C	
A T – G A T C G	T T A T A T C C	+1
A T – G T A T C G T	T A T A C C	+1

Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T – A T C	G T T A T A G T C C	
A T – G A T C G	T T A T A T C C	+1
A T – G T A T C G T	T A T A C C	+1
A T – G T T A T C G T –	A T A C C	

Matching Symbols as a Game



Matching Symbols as a Game

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T – A T C	G T T A T A G T C C	
A T – G A T C G	T T A T A T C C	+1
A T – G T A T C G T	T A T A C C	+1
A T – G T T A T C G T –	A T A C C	
A T – G T T A A T C G T – C	T A C	
A T – G T T A T A T C G T – C –	A C	

Matching Symbols as a Game

From a Game to a Definition

Given two strings *v* and *w*, an **alignment** of *v* and *w* is a two-row matrix such that:

- the first row contains symbols of *v* in order
- the second row contains symbols of *w* in order
- each row may also contain **gap symbols** ("-")
- no column has two gap symbols

```
A T - G T T A T A
A T C G T - C - C
```

From a Game to a Definition

Given two strings *v* and *w*, an **alignment** of *v* and *w* is a two-row matrix such that:

- the first row contains symbols of *v* in order
- the second row contains symbols of *w* in order
- each row may also contain **gap symbols** ("-")
- no column has two gap symbols

```
A T - G T T A T A
A T C G T - C - C
```

Matches

From a Game to a Definition

Given two strings *v* and *w*, an **alignment** of *v* and *w* is a two-row matrix such that:

- the first row contains symbols of *v* in order
- the second row contains symbols of *w* in order
- each row may also contain **gap symbols** ("-")
- no column has two gap symbols

```
A T - G T T A T A
A T C G T - C - C
```

Mismatches

From a Game to a Definition

Given two strings *v* and *w*, an **alignment** of *v* and *w* is a two-row matrix such that:

- the first row contains symbols of *v* in order
- the second row contains symbols of *w* in order
- each row may also contain **gap symbols** ("-")
- no column has two gap symbols

```
A T - G T T A T A
A T C G T - C - C
```

Insertions

From a Game to a Definition

Given two strings *v* and *w*, an **alignment** of *v* and *w* is a two-row matrix such that:

- the first row contains symbols of *v* in order
- the second row contains symbols of *w* in order
- each row may also contain **gap symbols** ("-")
- no column has two gap symbols

```
A T - G T T A T A
A T C G T - C - C
```

Deletions

Finding a Longest Common Subsequence

A **common subsequence** of *v* and *w* is a sequence of symbols occurring (not necessarily contiguously) in both *v* and *w*.

Finding a Longest Common Subsequence

A **common subsequence** of *v* and *w* is a sequence of symbols occurring (not necessarily contiguously) in both *v* and *w*.

The **matches** in an alignment of *v* and *w* form a common subsequence of *v* and *w*.

```
AT-GTTATA
ATCGT-C-C
```

The Problems are the Same!

Longest Common Subsequence Length Problem:.

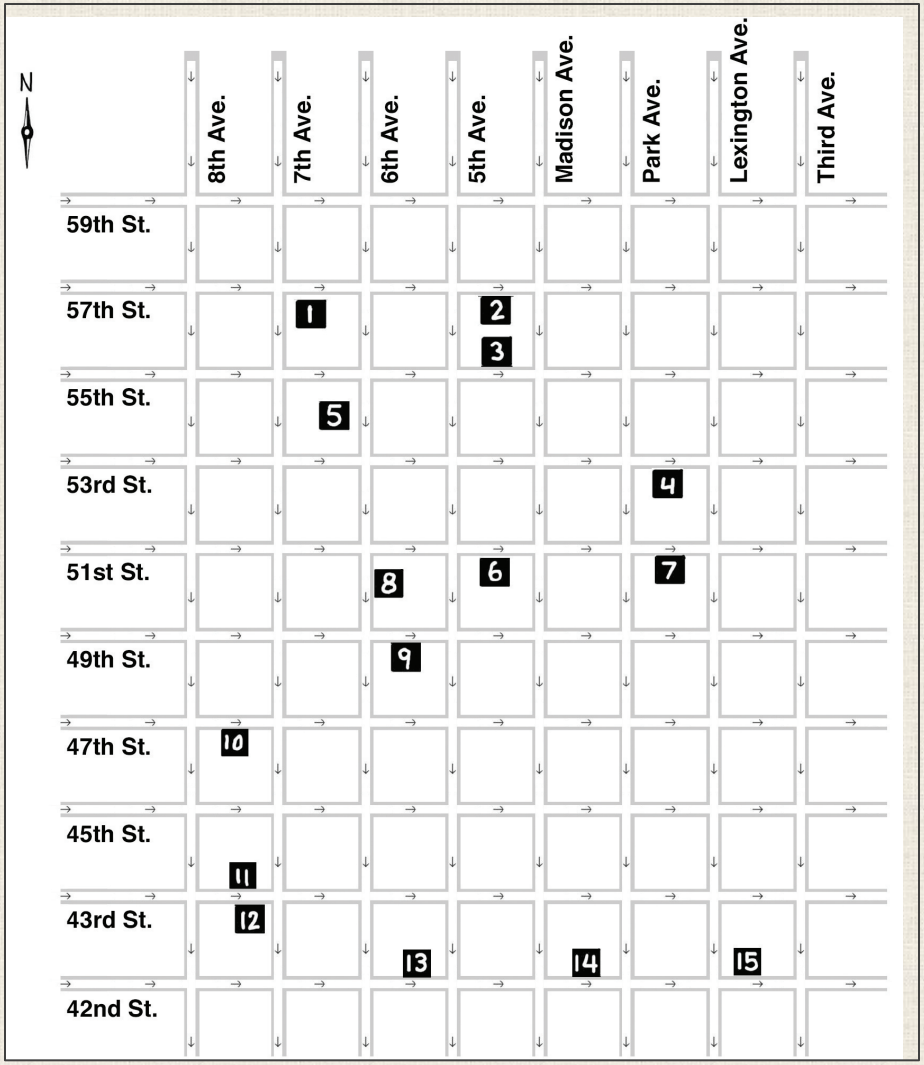
- **Input:** Two strings.
- **Output:** The length of a longest common subsequence of these strings.

Symbol Matching Problem:

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any "alignment" of the two strings.

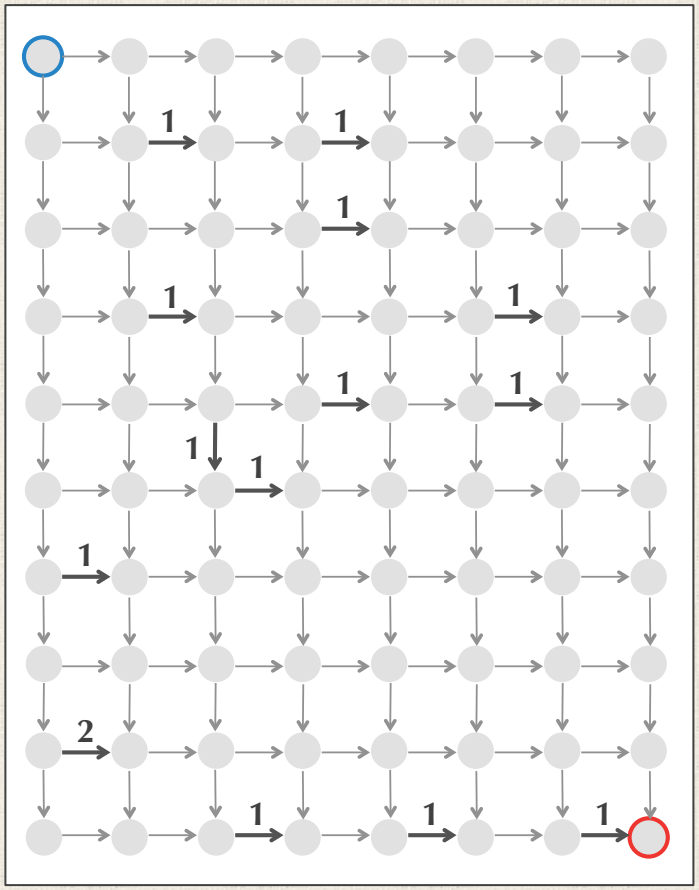
THE MANHATTAN TOURIST PROBLEM

Manhattan Tourist Problem



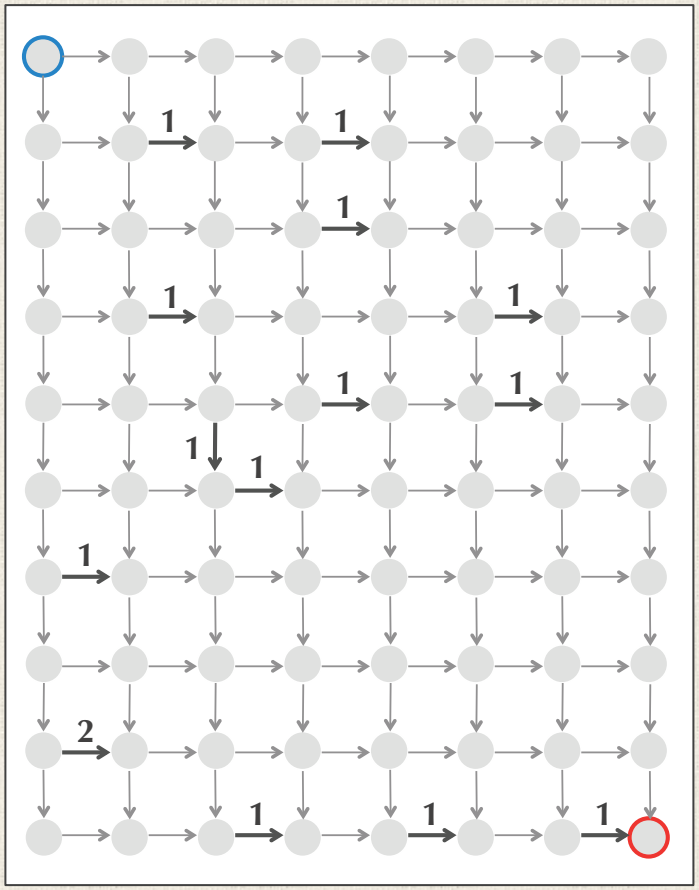
STOP: How can we see the most sites if we move from 59th and 8th to 42nd and 3rd, moving south or east at each step? (And what algorithm did you use?)

Manhattan Tourist as a Network



Weight of edge: number of attractions along the edge.

Manhattan Tourist as a Network



Weight of edge: number of attractions along the edge.

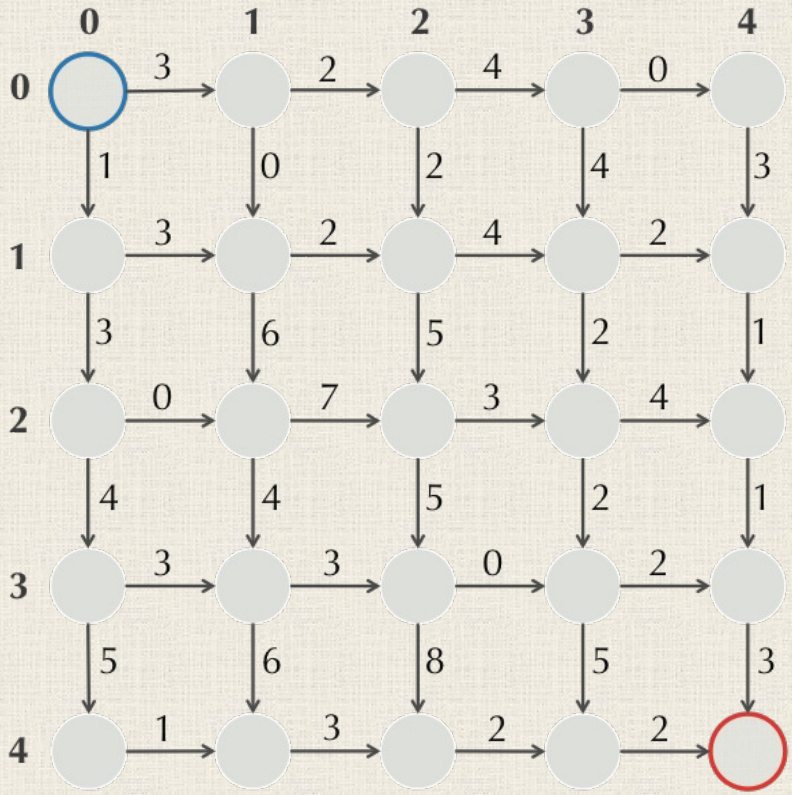
Goal: Find a *longest* path from ***source*** (top left) to ***sink*** (bottom right).

Toward a Computational Problem

Manhattan Tourist Problem:

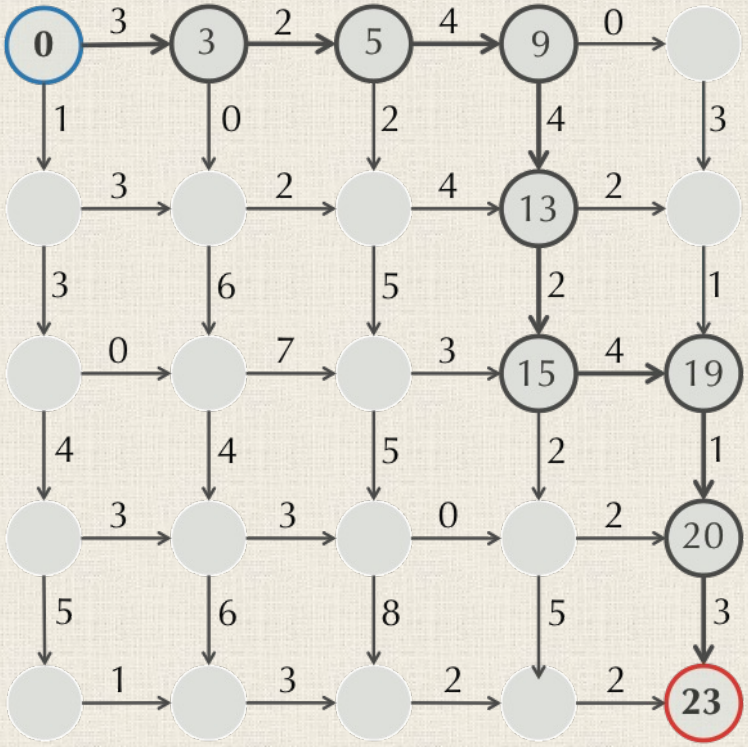
- **Input:** A weighted n x m rectangular grid (n + 1 rows and m + 1 columns).
- **Output:** A longest path from source (0, 0) to sink (n, m) in the grid.

Designing a Manhattan Algorithm



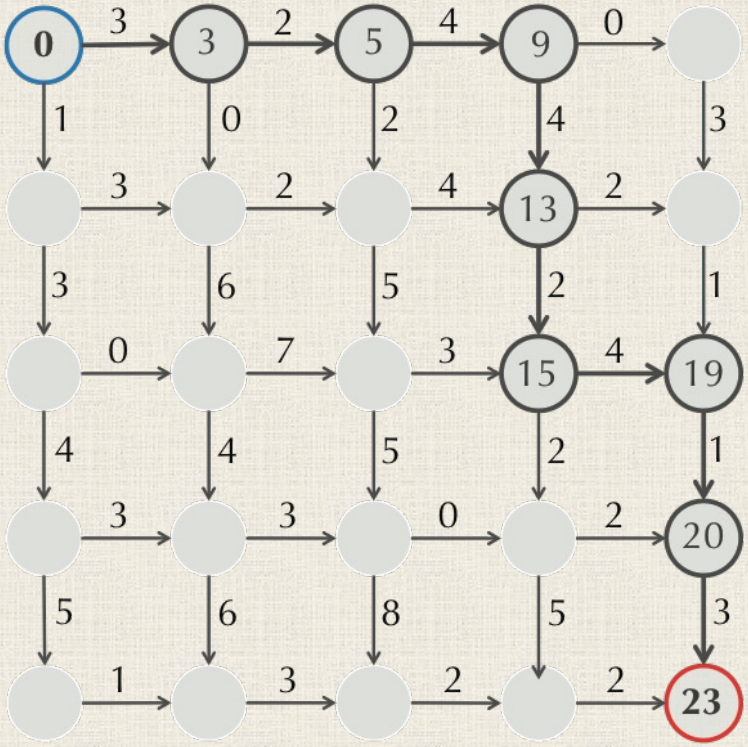
Exercise: What is the longest path in this city? What algorithm did you use?

A "Greedy" Manhattan Algorithm Taking the Best Choice in Each Node



STOP: Does the greedy algorithm solve the problem?

A "Greedy" Manhattan Algorithm Taking the Best Choice in Each Node



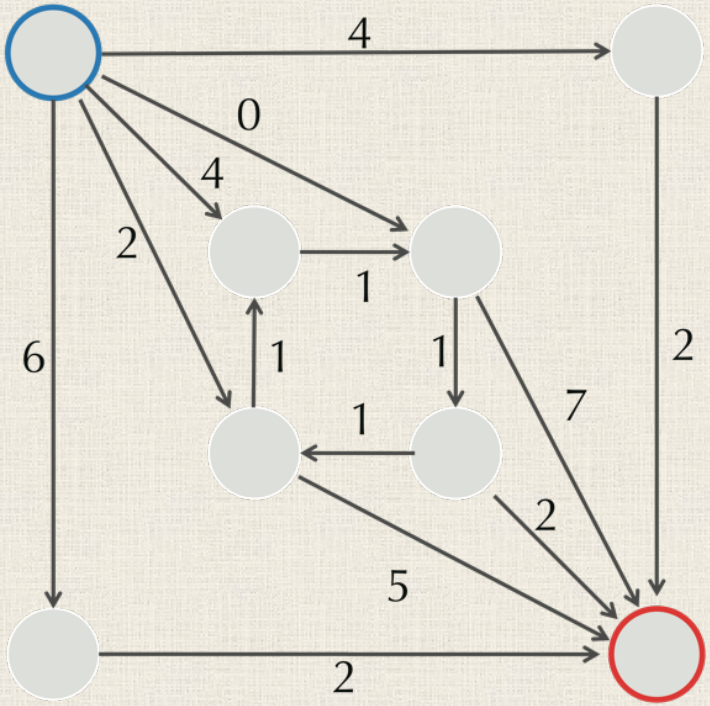
Answer: No! Much like with genome assembly, we need a more clever approach.

Manhattan Tourist as a Network Problem

Longest Path in a Directed Graph:

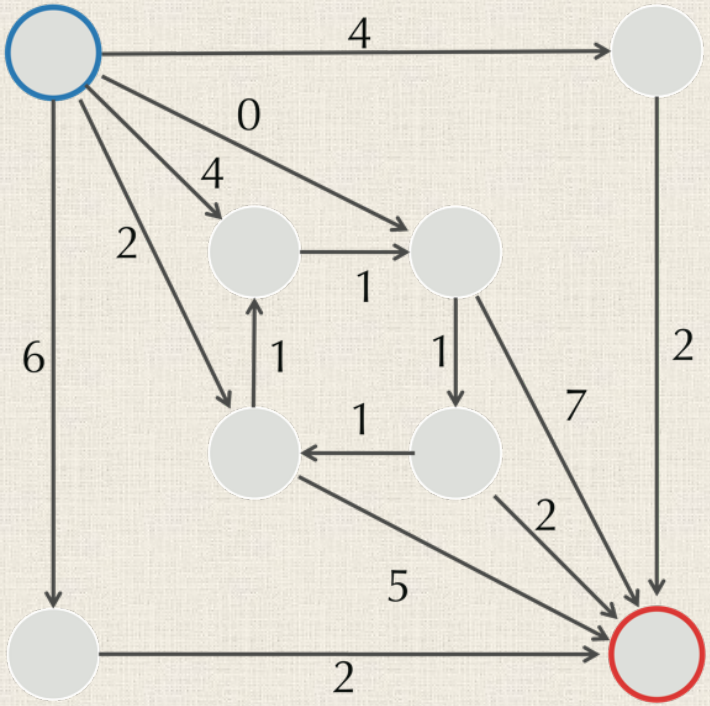
- **Input:** An edge-weighted directed graph with source and sink nodes.
- **Output:** A longest path from source to sink in the graph.

Manhattan Tourist as a Network Problem



STOP: What is the longest path in this graph?

Manhattan Tourist as a Network Problem



Answer: Cycles in graphs cause infinite paths ...

Generalizing Manhattan Tourist

Directed acyclic graph (DAG): A directed graph that contains *no* cycles.

Generalizing Manhattan Tourist

Directed acyclic graph (DAG): A directed graph that contains *no* cycles.

Longest Path in a DAG Problem:

- **Input:** An edge-weighted DAG with source and sink nodes.
- **Output:** A longest path from source to sink in the DAG.

Generalizing Manhattan Tourist

Directed acyclic graph (DAG): A directed graph that contains *no* cycles.

Longest Path in a DAG Problem:

- **Input:** An edge-weighted DAG with source and sink nodes.
- **Output:** A longest path from source to sink in the DAG.

... but what does finding a longest path in a DAG have to do with sequence comparison?

SEQUENCE ALIGNMENT AS A PATH IN A NETWORK

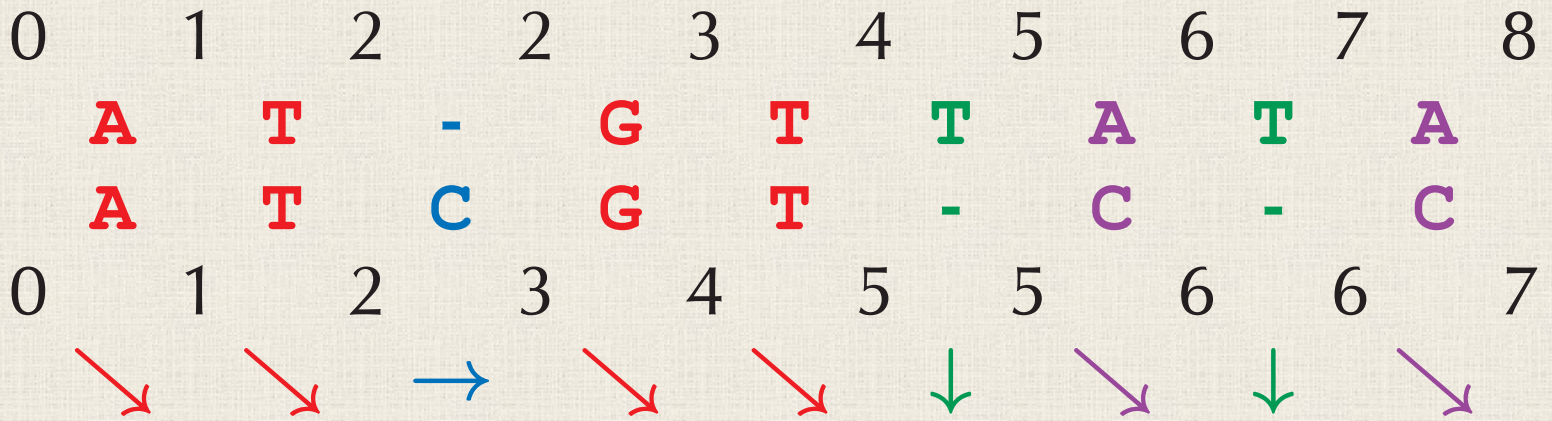
Returning to Sequence Alignment ...

```
A   T   -   G   T   T   A   T   A
A   T   C   G   T   -   C   -   C
```

Returning to Sequence Alignment ...

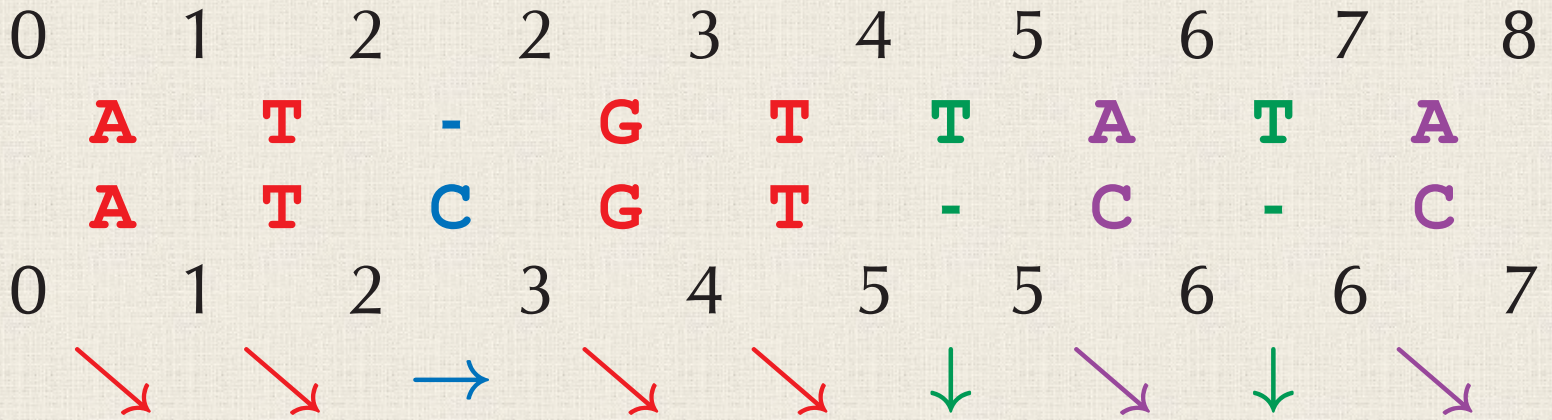
0	1	2	2	3	4	5	6	7	8
	A	T	-	G	T	T	A	T	A
	A	T	C	G	T	-	C	-	C
0	1	2	3	4	5	5	6	6	7

Returning to Sequence Alignment ...



(0, 0) ↘ (1, 1) ↘ (2, 2) → (2, 3) ↘ (3, 4)
↘ (4, 5) ↓ (5, 5) ↘ (6, 6) ↓ (7, 6) ↘ (8, 7)

Returning to Sequence Alignment ...



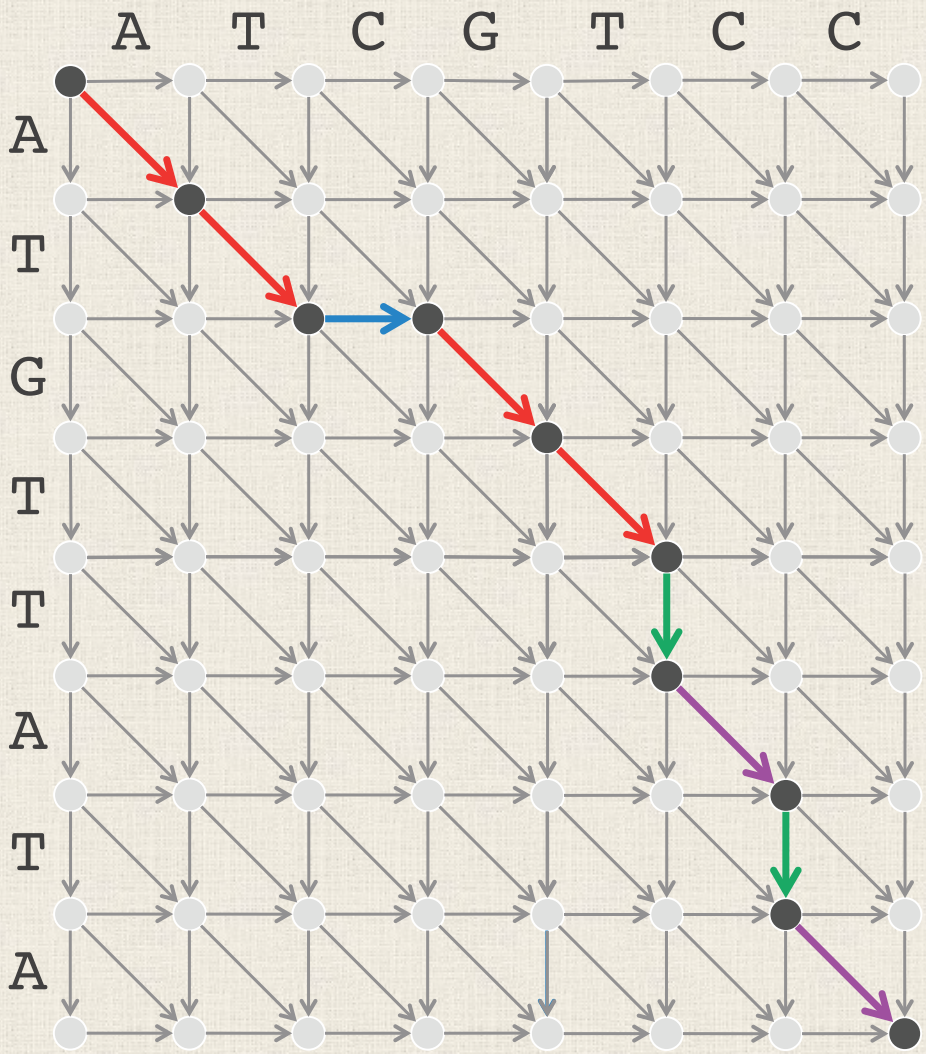
(0, 0) ↘ (1, 1) ↘ (2, 2) → (2, 3) ↘ (3, 4)
↘ (4, 5) ↓ (5, 5) ↘ (6, 6) ↓ (7, 6) ↘ (8, 7)

This is a path in a 2-D network!

Representing an Alignment as a Path in a Manhattan-like DAG

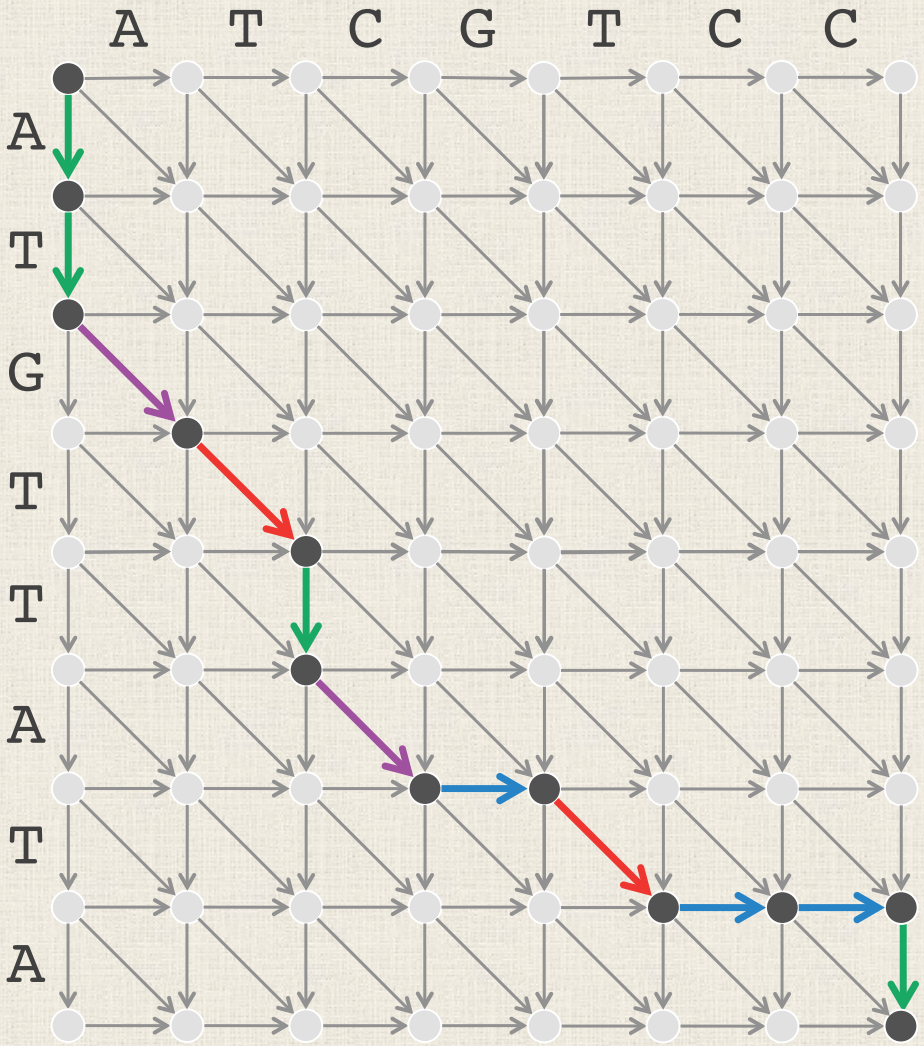
```
A T - G T T A T A
A T C G T - C - C
```

This network is called the **alignment network** of the strings ATGTTATA and ATCGTCC.



We can also construct an alignment from a path

Exercise: What alignment does this path correspond to?



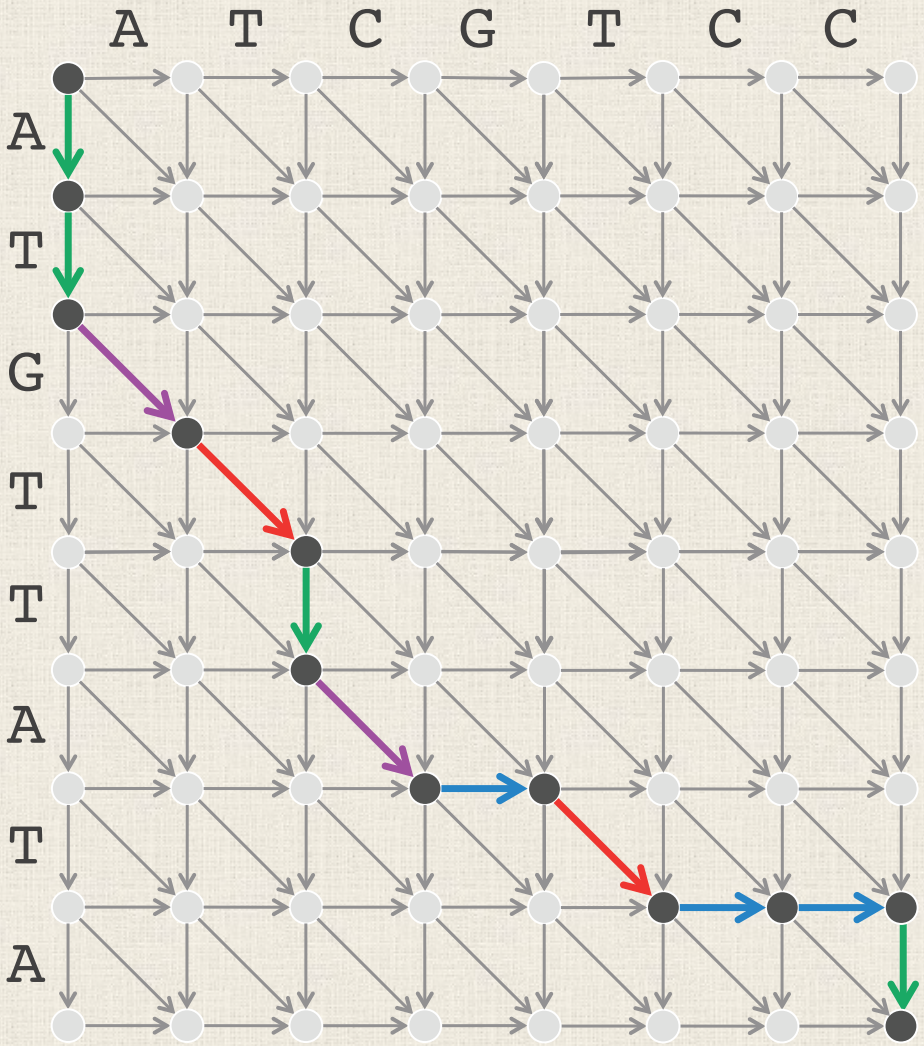
We can also construct an alignment from a path

Exercise: What alignment does this path correspond to?

Answer:

A T G T T A - T - - A

- - A T - C G T C C -



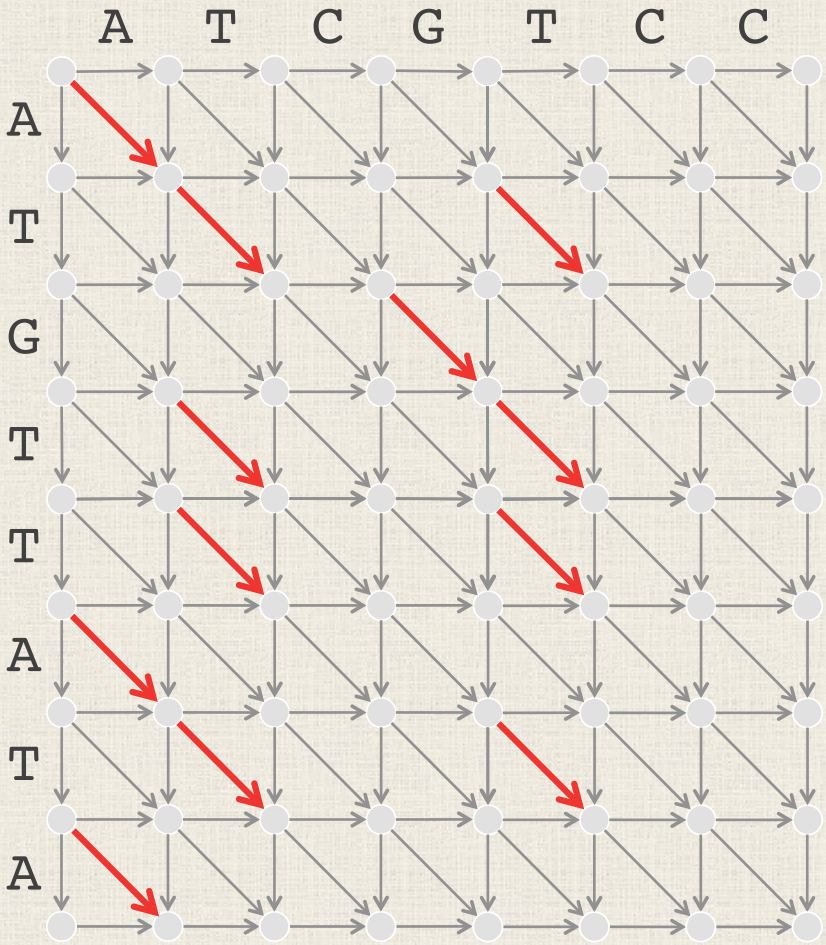
Solving the Symbol Matching Problem

Symbol Matching Problem:

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any alignment of the two strings.

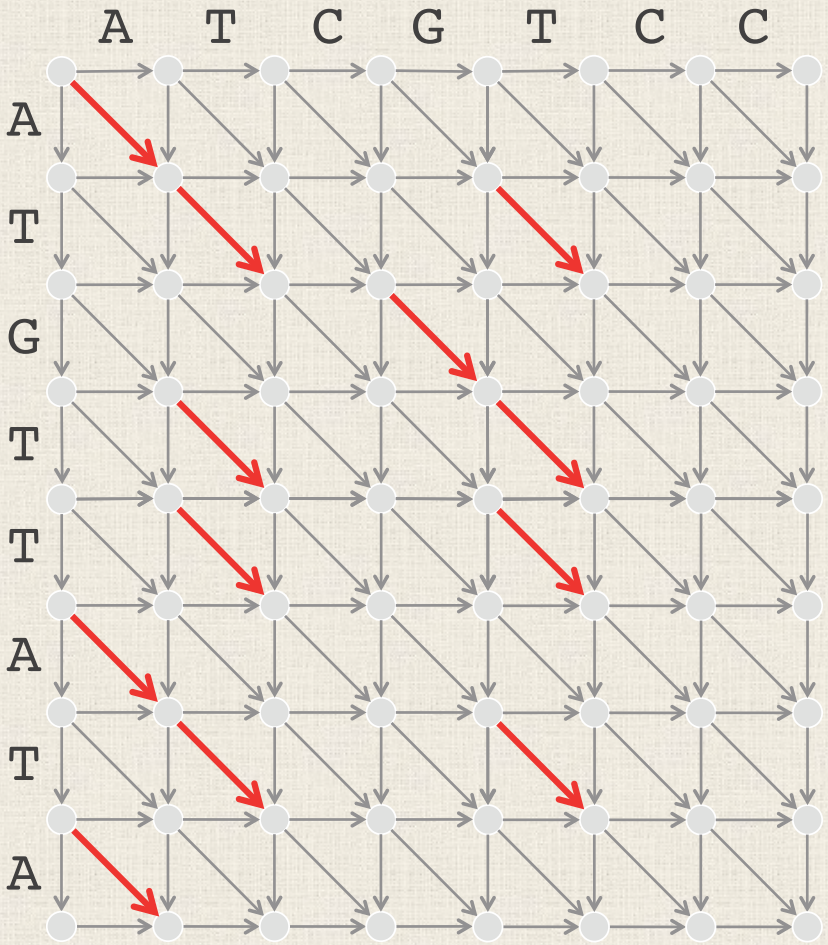
STOP: How can we use the alignment network to solve this problem?

Counting Matches Only



Answer: If we weight the red edges as 1 and the other edges as 0, then a maximum-weight path from source to sink solves the Symbol Matching Problem!

Counting Matches Only



Answer: If we weight the red edges as 1 and the other edges as 0, then a maximum-weight path from source to sink solves the Symbol Matching Problem!

But we haven't said how to *find* the maximum length of a path.

AN INTRO TO DYNAMIC PROGRAMMING

Recursive Fibonacci Numbers

Exercise: Write pseudocode for a recursive function that takes an integer n as an argument and returns the n-th Fibonacci number. Assume 0-based indexing.

If $Fib(n)$ is the n-th Fibonacci number, then

$$Fib(n) = Fib(n - 1) + Fib(n - 2)$$

Recurrence relation: An expression for a function f(x) in terms of values of f(y) where $y < x$.

Recursive Fibonacci Numbers

Exercise: Write pseudocode for a recursive function that takes an integer *n* as an argument and returns the *n*-th Fibonacci number. Assume 0-based indexing.

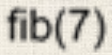
```
RecFib(n)
        if n = 0 or n = 1
                return 1
        else
                return RecFib(n-1) + RecFib(n-2)
```

Recursive Fibonacci Numbers

STOP: Is this a good algorithm? Why or why not?

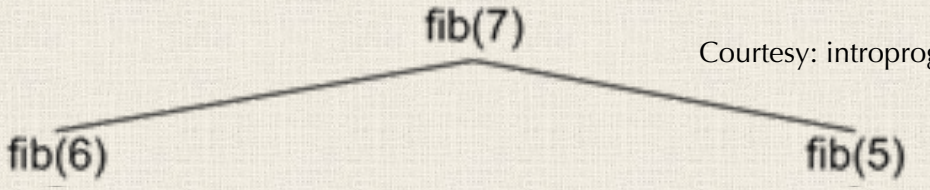
```
RecFib(n)
        if n = 0 or n = 1
                return 1
        else
                return RecFib(n-1) + RecFib(n-2)
```

Calling Fib(7) Shows the Problem with Using Recursion



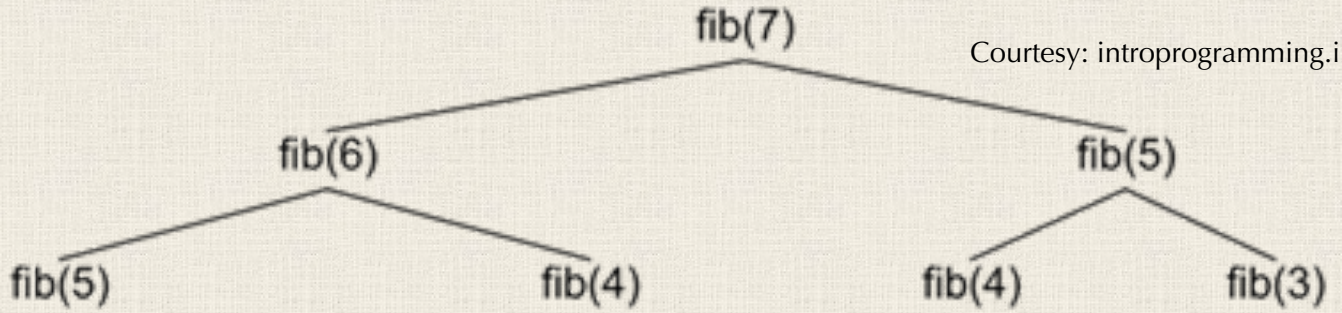
Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



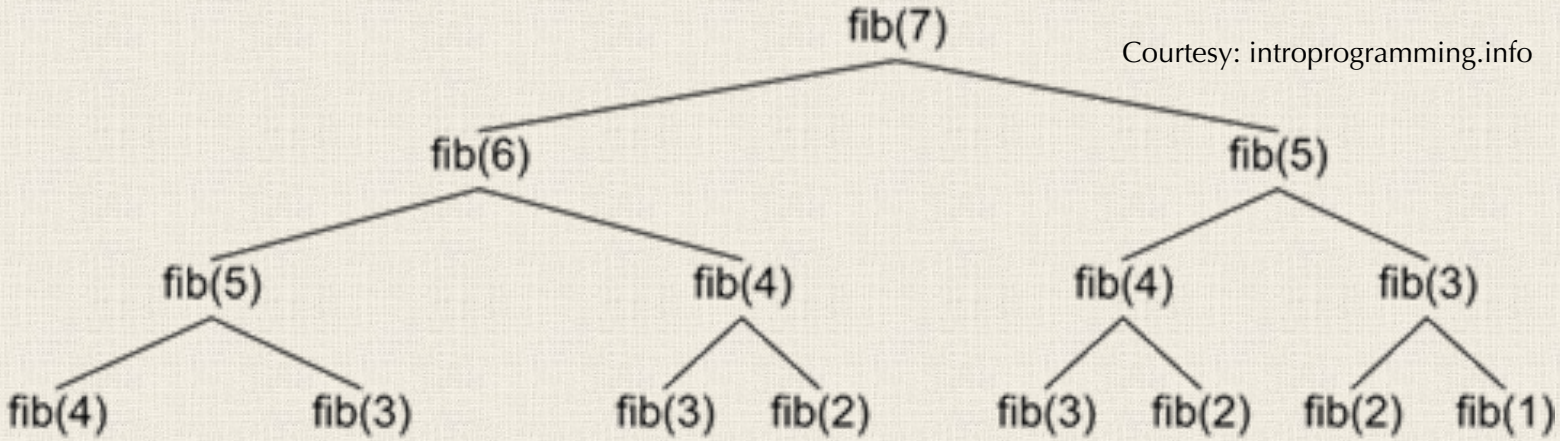
Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



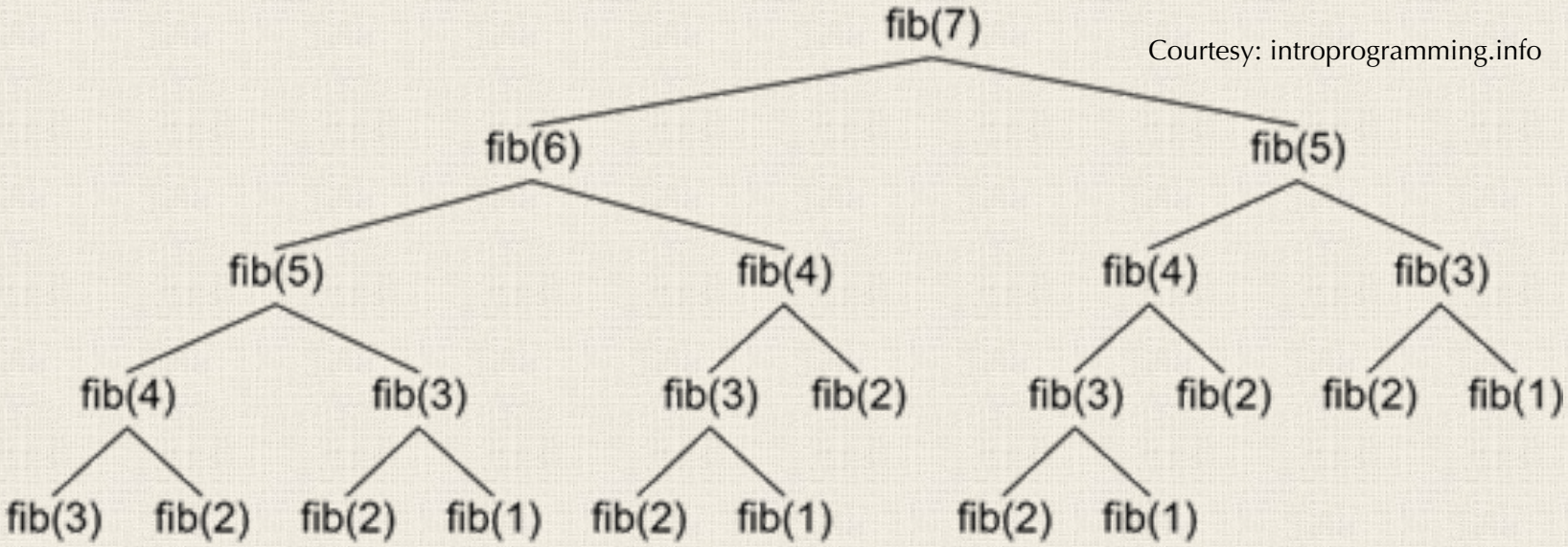
Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



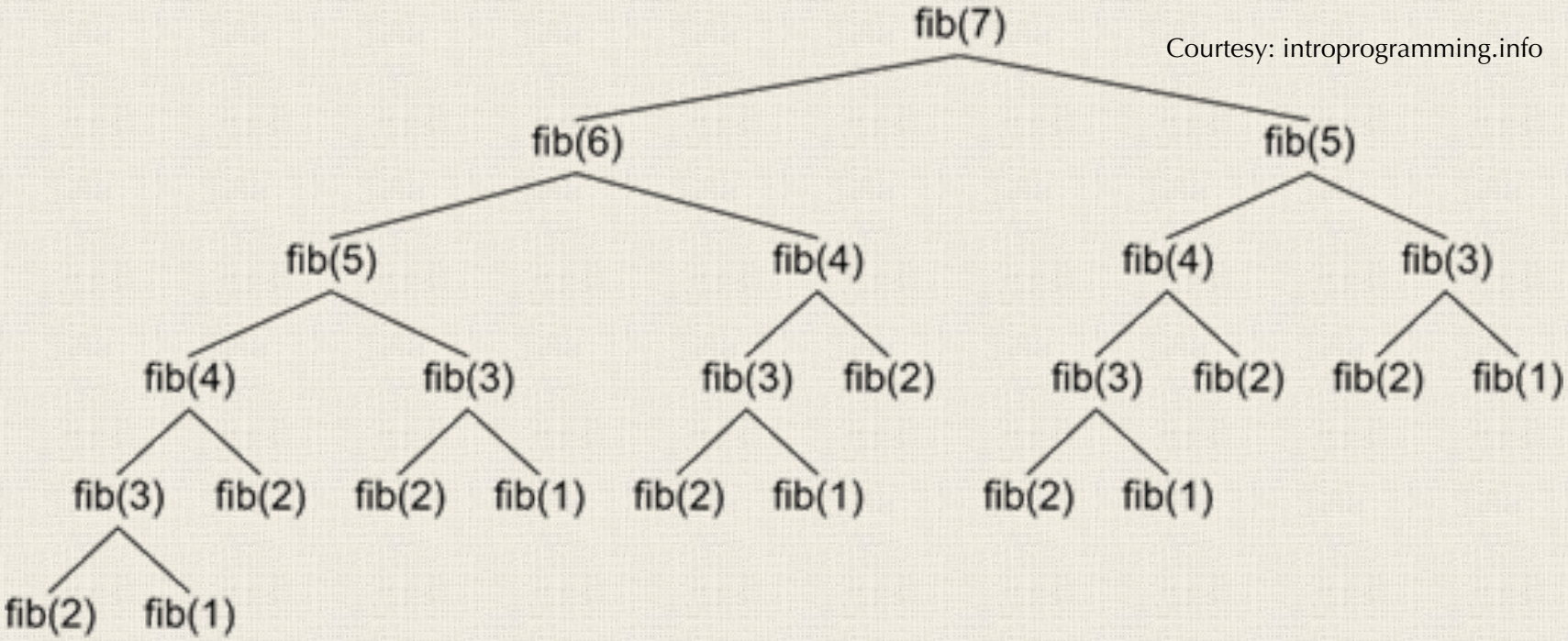
Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



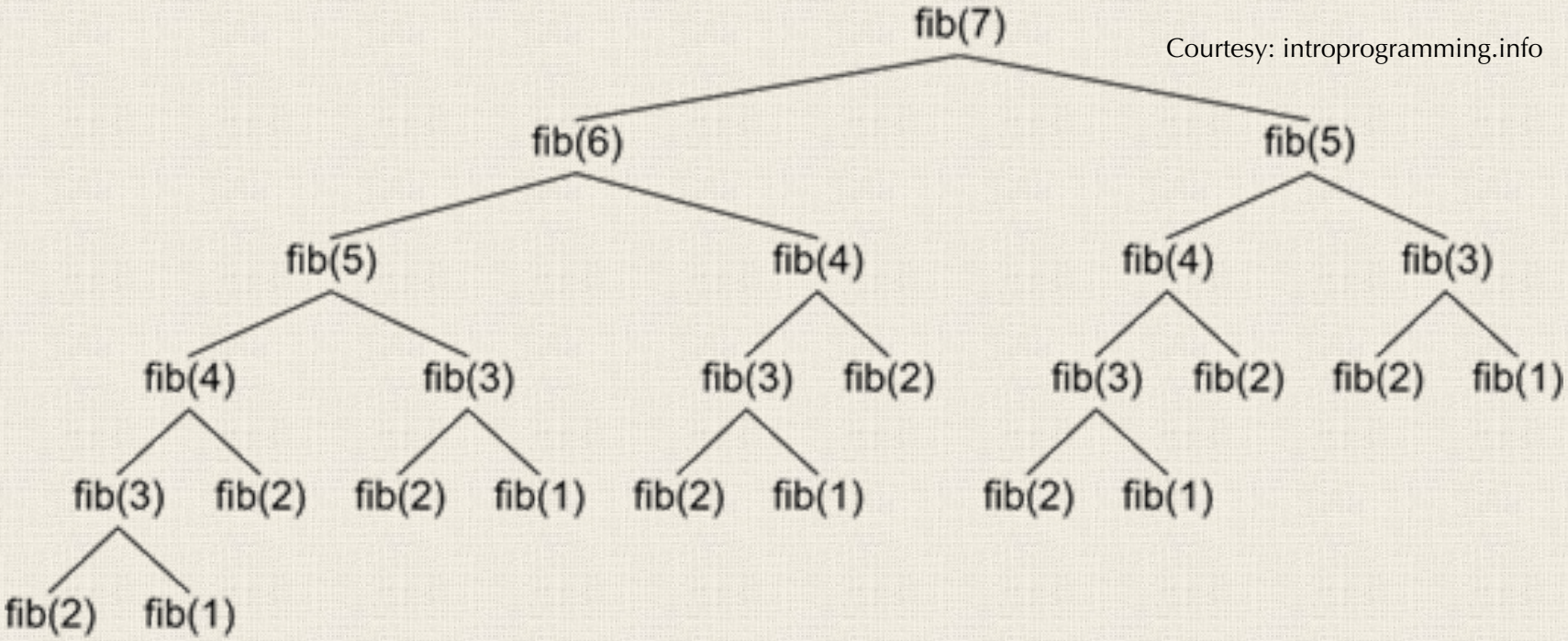
Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



Courtesy: introprogramming.info

Calling Fib(7) Shows the Problem with Using Recursion



Courtesy: introprogramming.info

STOP: Approximately how many calls do you think are made for **RecFib**(20)? What about **RecFib**(45)?

The Issue with Fibonacci Recursion

When we call RecFib(n), there are ~ 2^n calls on the stack. For most values of n, this will exhaust the memory allocated to the stack and produce what is called **stack overflow**, crashing the program.

The Issue with Fibonacci Recursion

When we call RecFib(n), there are ~ 2^n calls on the stack. For most values of n, this will exhaust the memory allocated to the stack and produce what is called **stack overflow**, crashing the program.

Key Point: We should evaluate whether recursion is a good approach for solving a problem based on whether we have many repeated calls with a chance of stack overflow.

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.


```
Fibonacci(n)
        a ← array of length n
        a[0] ← 1
        a[1] ← 1
        for i ← 2 to n
                a[i] ← a[i-1] + a[i-2]
        return a
```

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

1	1	2	3	5	8	13			

a

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

1	1	2	3	5	8	13	21		

a

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

1	1	2	3	5	8	13	21	34	

a

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Instead of computing Fibonacci numbers top-down recursively, we compute them bottom-up.

1	1	2	3	5	8	13	21	34	55

a

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Values "Bottom-Up" Avoids Many Recursive Calls

Computing a recurrence relation bottom-up using an array is called **dynamic programming**.

1	1	2	3	5	8	13	21	34	55

a

Fibonacci(*n*)
    a ← array of length *n*
    a[0] ← 1
    a[1] ← 1
    for *i* ← 2 to *n*
        a[*i*] ← *a*[*i*-1] + *a*[*i*-2]
    return *a*

Computing Fibonacci Numbers

Computing a recurrence relation bottom-up using an array is called **dynamic programming**.

STOP: Wait … why would such a simple idea be called "dynamic programming"?

Richard Bellman

Richard Bellman, a Wise Man

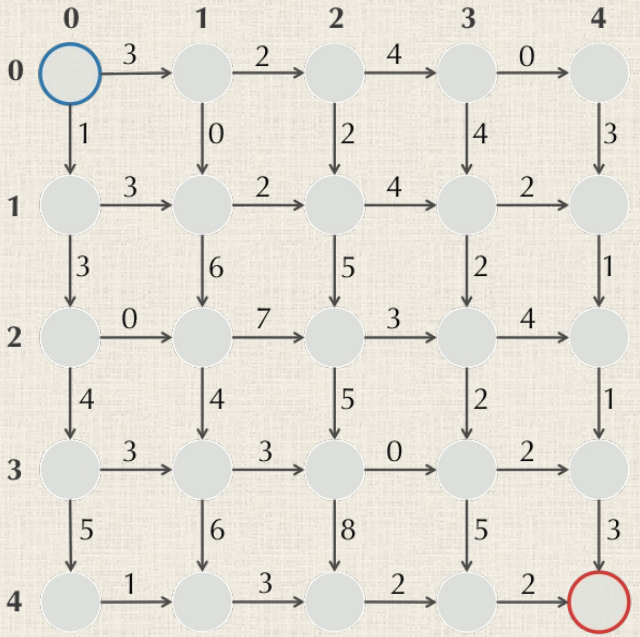
“We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word "research". I’m not using the term lightly; I’m using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.”

FINDING THE LENGTH OF A LONGEST PATH IN A DAG

Returning to Manhattan

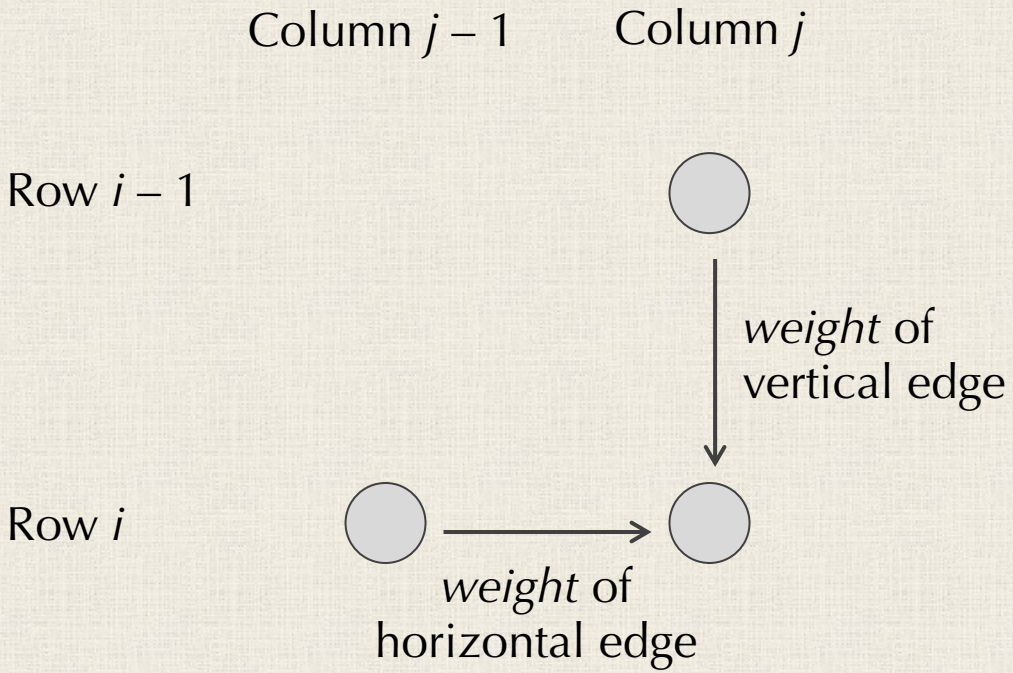
Manhattan Tourist Problem:

- **Input:** A weighted n x m rectangular grid (n + 1 rows and m + 1 columns).
- **Output:** A longest path from source (0, 0) to sink (n, m) in the grid.



Exercise: Find a recurrence relation for the length of a longest path from (0,0) to node (i, j), which we will call *length*(i,j).

Returning to Manhattan

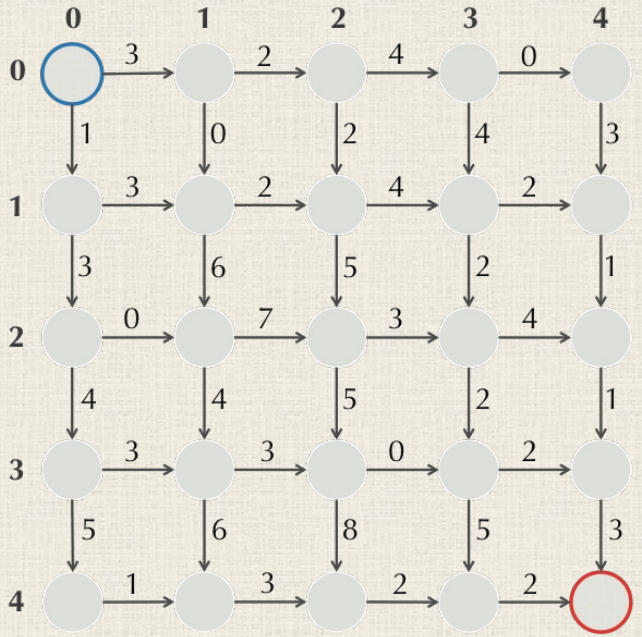


Answer: $length(i,j) = \max\{$
$length(i-1,j) + \text{weight(vertical edge into } i,j),$
$length(i, j-1) + \text{weight(horizontal edge into } i,j)\}.$

Returning to Manhattan

Manhattan Tourist Problem:

- **Input:** A weighted n x m rectangular grid (n + 1 rows and m + 1 columns).
- **Output:** A longest path from source (0, 0) to sink (n, m) in the grid.

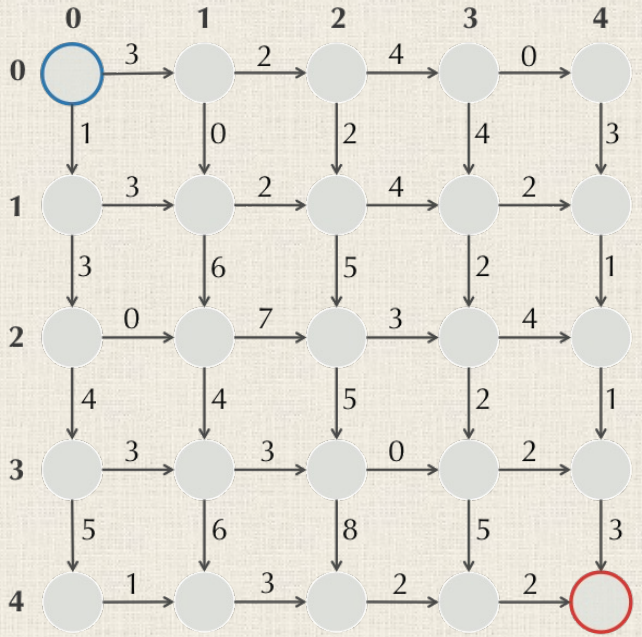


STOP: Will a recursive algorithm for Manhattan Tourist have the same problem that the recursive change-making function encountered?

Returning to Manhattan

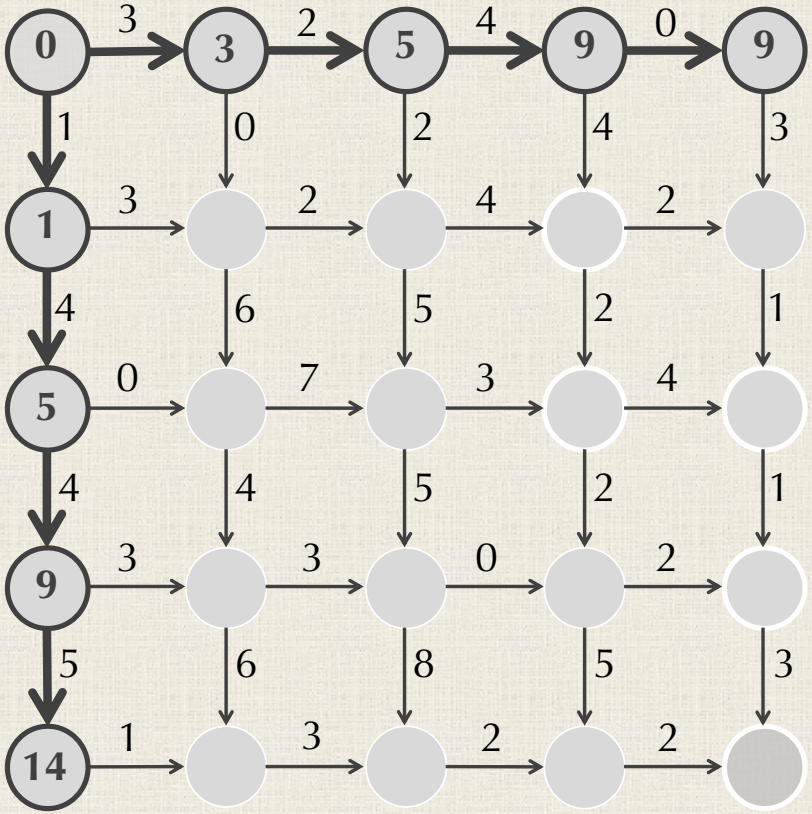
Manhattan Tourist Problem:

- **Input:** A weighted n x m rectangular grid (n + 1 rows and m + 1 columns).
- **Output:** A longest path from source (0, 0) to sink (n, m) in the grid.



Answer: Yes! Because the same *length*(i, j) can get re-computed many times…

Let's Use Dynamic Programming Instead

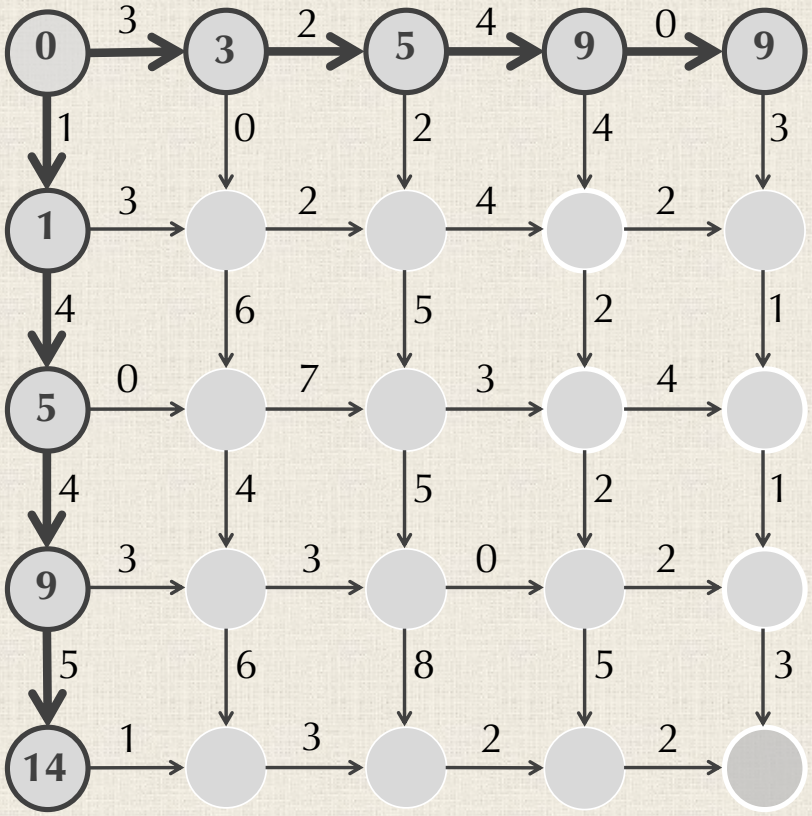


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

Let's Use Dynamic Programming Instead

STOP: Which element of the table should we fill in next and what should its value be?

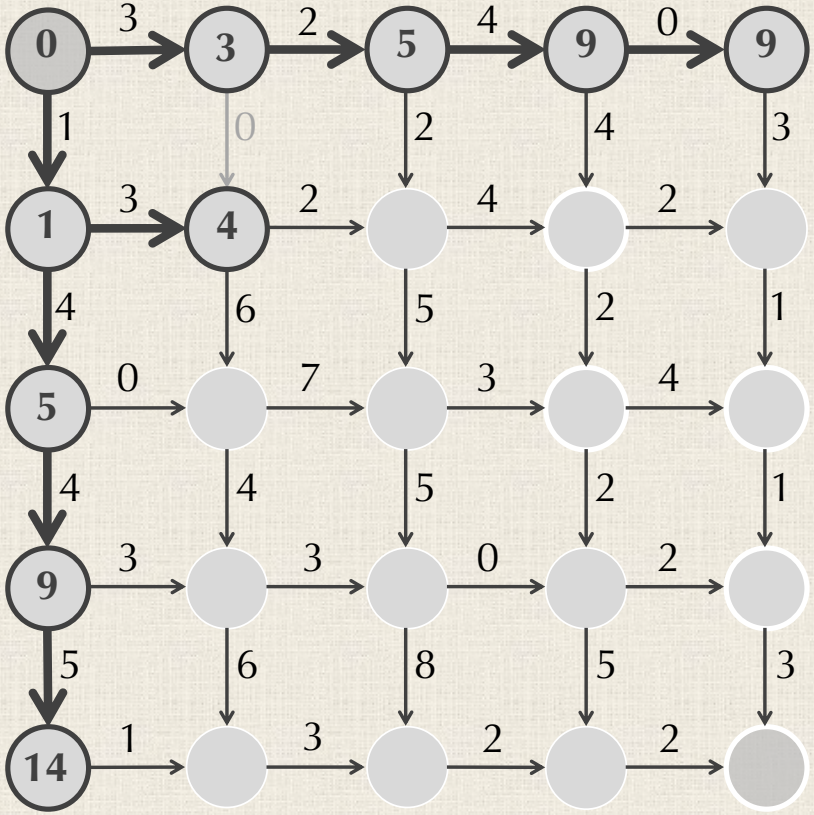


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

Let's Use Dynamic Programming Instead

Answer: We only know the values of MaxWeight for the two nodes adjacent to the node (1, 1); it gets the value max(3+0, 1+3) = 4.

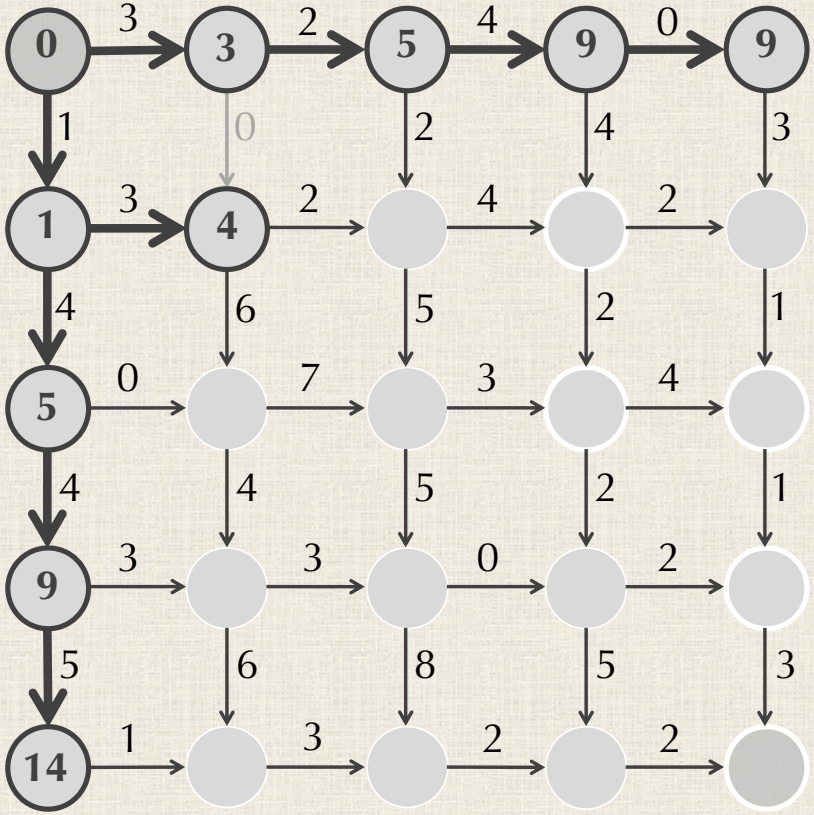


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

Let's Use Dynamic Programming Instead

STOP: Which elements should we fill in next and what should their values be?

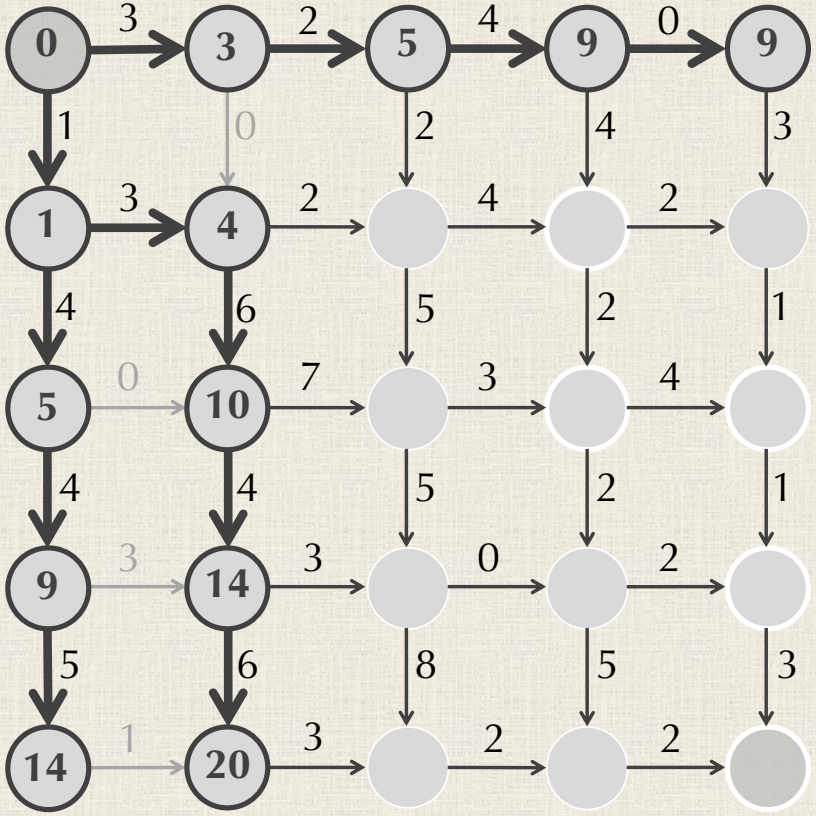


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

Let's Use Dynamic Programming Instead

Answer: We can fill in all of row 1 or all of column 1 (it doesn't matter which).

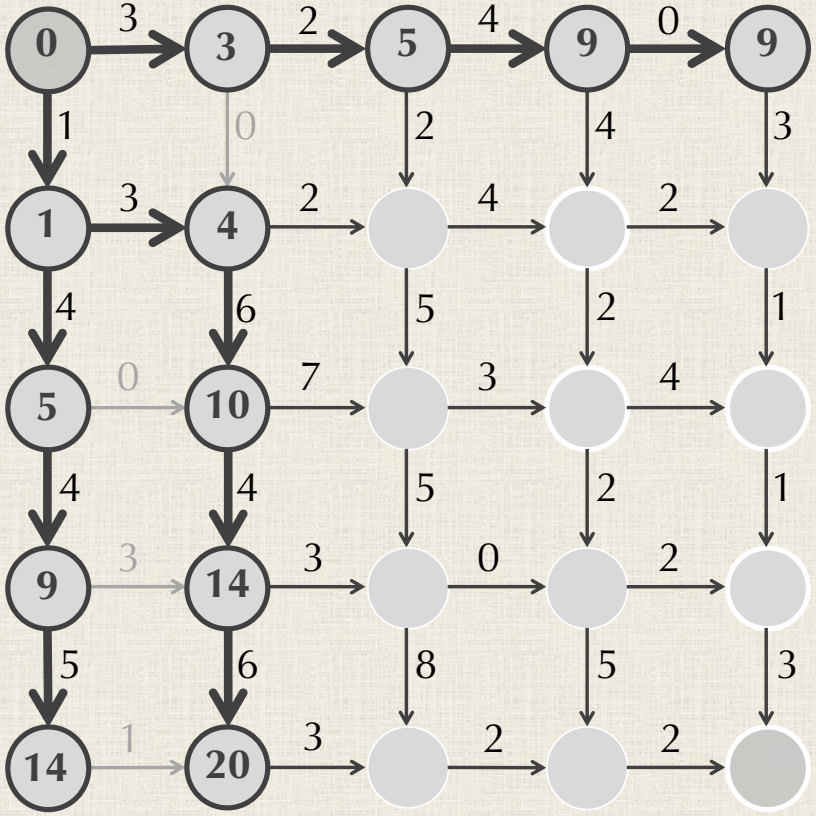


Recurrence relation

$$\text{MaxWeight}(i, j) = \max\{\text{MaxWeight}(i\text{-}1, j) + \textit{down}(i, j), \text{MaxWeight}(i, j\text{-}1) + \textit{right}(i, j)\}$$

Let's Use Dynamic Programming Instead

Exercise: Fill in the remaining values of *length* for this network.

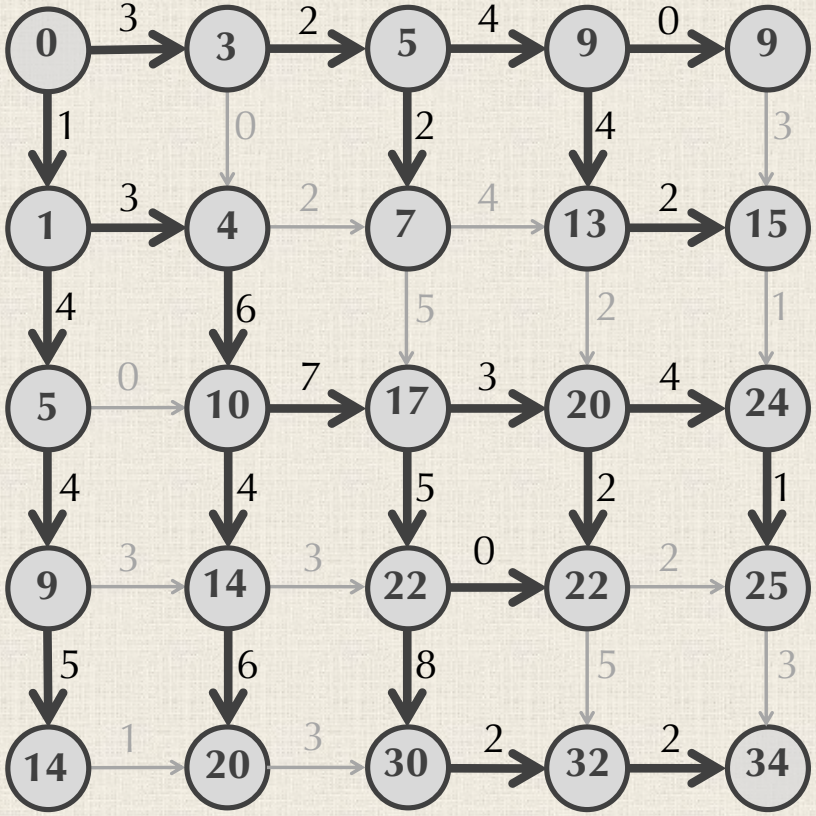


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

Let's Use Dynamic Programming Instead

STOP: Now do you see a longest path in this grid? How might we find one in general?

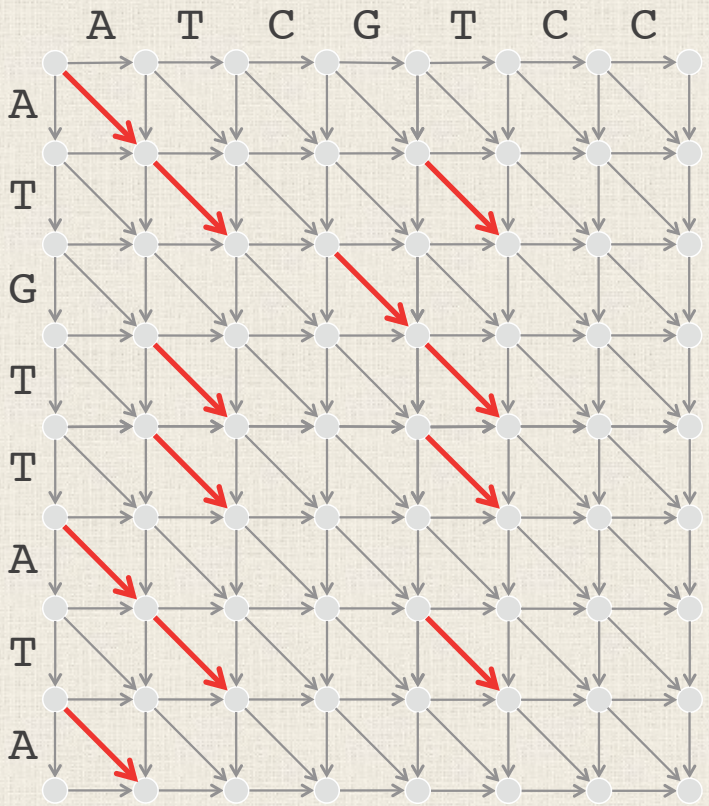


Recurrence relation

$$length(i, j) = \max\{length(i\text{-}1, j) + down(i, j),\ length(i, j\text{-}1) + right(i, j)\}$$

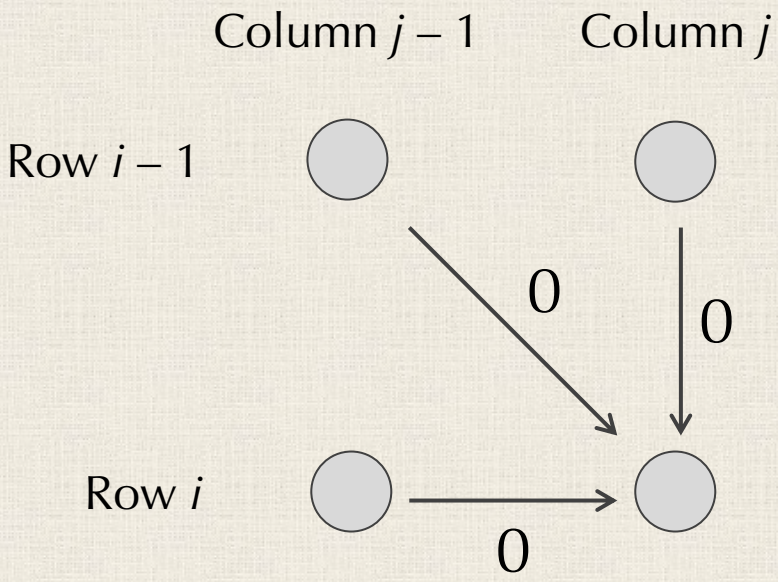
Finding an LCS

Exercise: What is the recurrence relation for finding a longest common subsequence?



Our Recurrence Has Two Cases

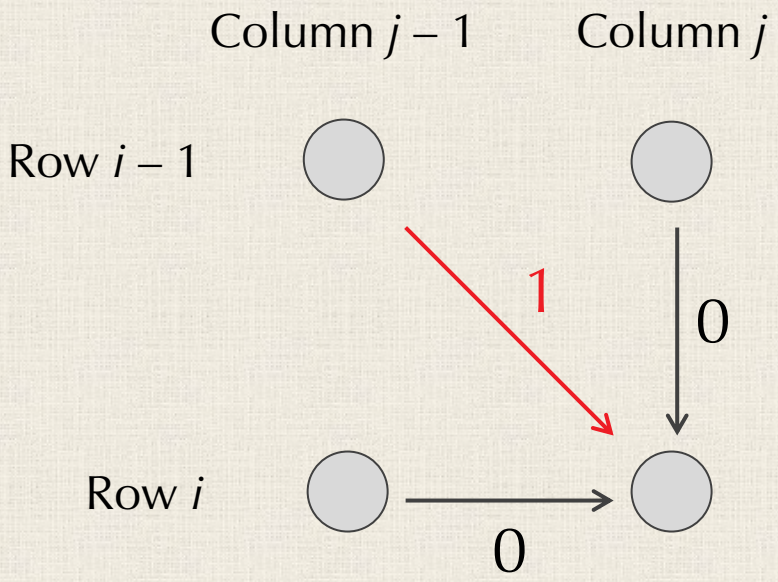
Case 1



$length(i, j)$ = maximum of:

- $length(i-1, j) + 0$
- $length(i, j-1) + 0$
- $length(i-1, j-1) + 0$

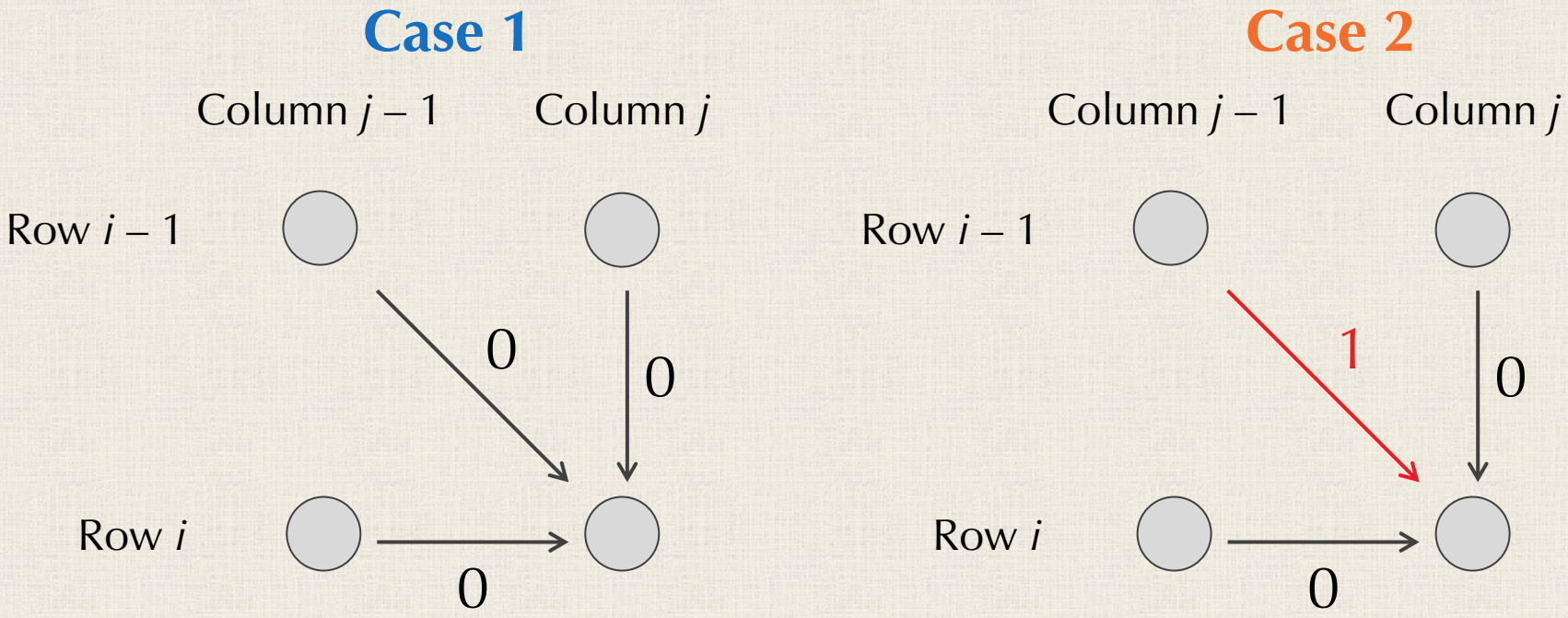
Case 2



$length(i, j)$ = maximum of:

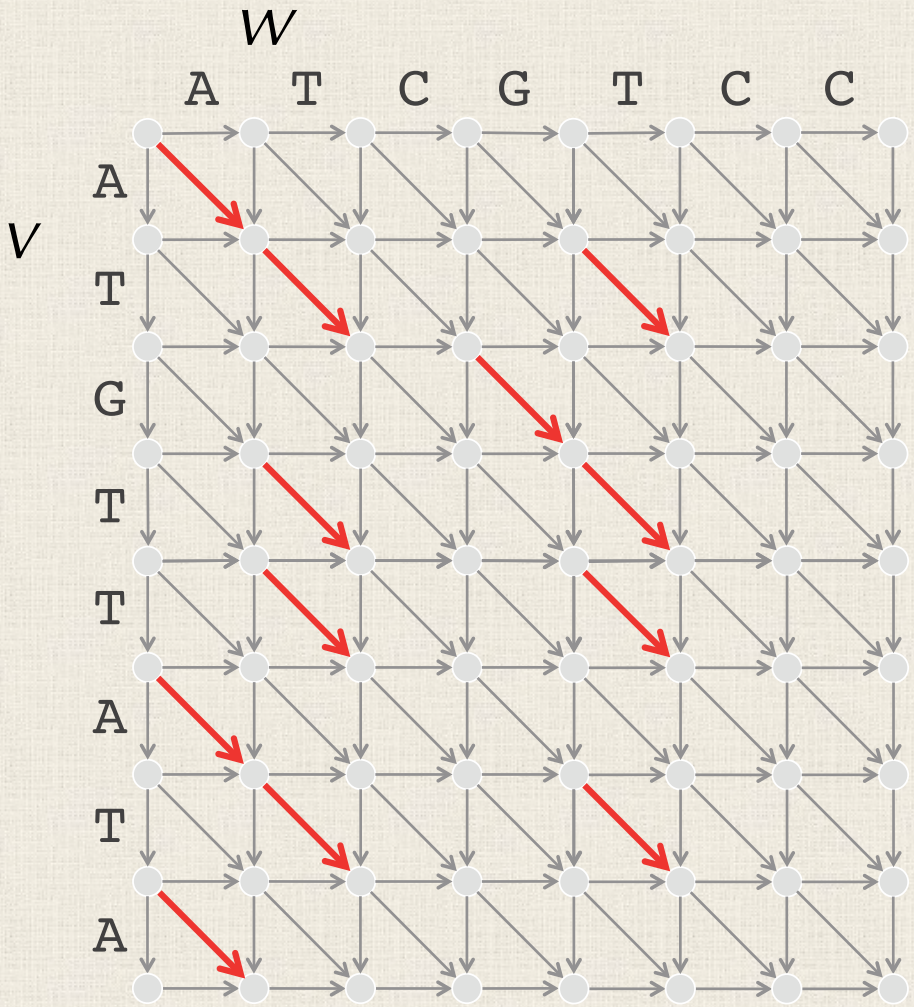
- $length(i-1, j) + 0$
- $length(i, j-1) + 0$
- $length(i-1, j-1) + 1$

Our Recurrence Has Two Cases



STOP: when will the diagonal edge weight be equal to 1?

Counting Matches Only

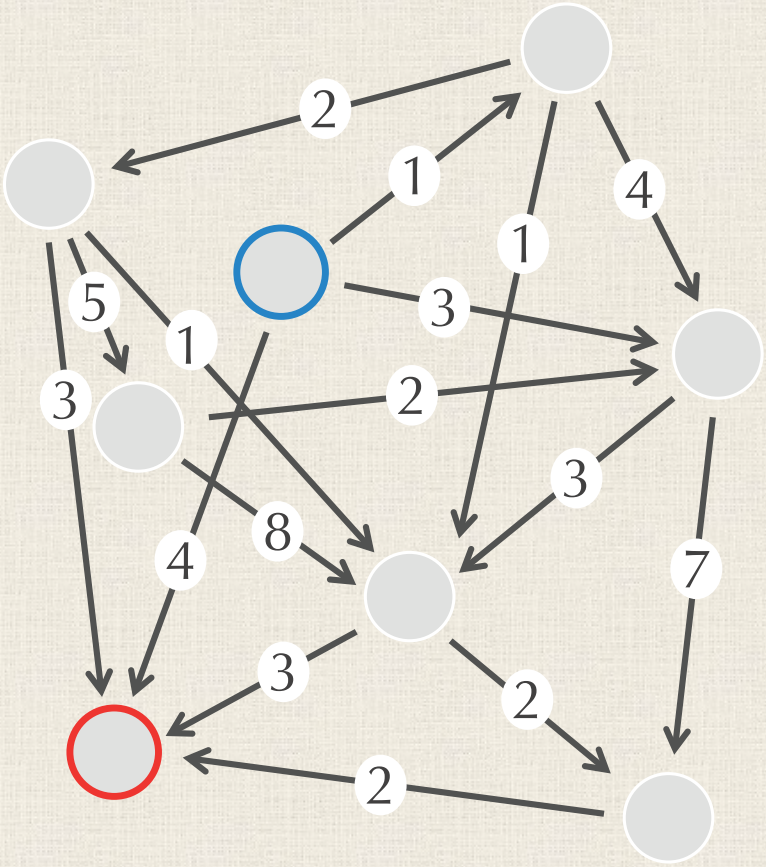


Answer: a diagonal edge connecting $(i - 1, j - 1)$ to (i, j) is 1 when the corresponding symbols $v[i - 1]$ and $w[j - 1]$ of the two strings *match*.

A Recurrence for an Arbitrary DAG?

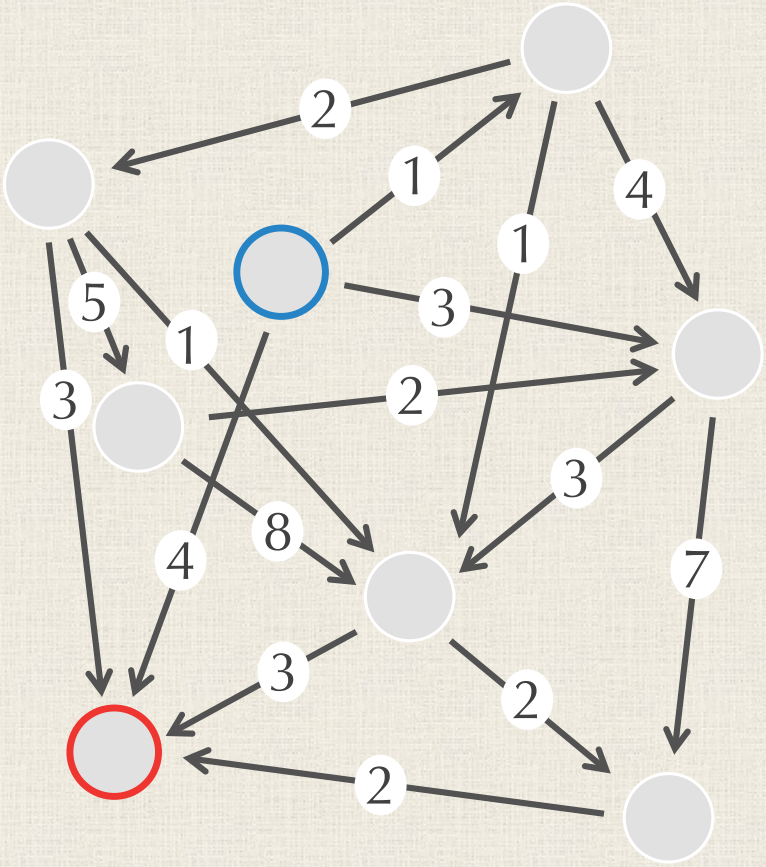
Longest Path in a DAG Problem:

- **Input:** An edge-weighted DAG with source and sink nodes.
- **Output:** A longest path from source to sink in the DAG.



A Recurrence for an Arbitrary DAG?

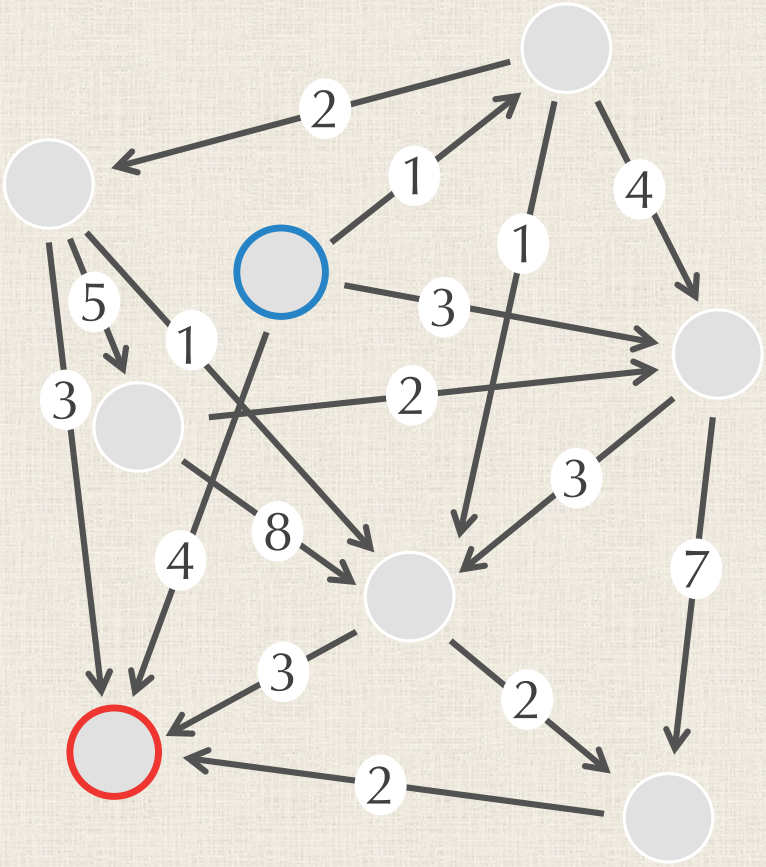
Exercise: Try finding a longest path from *source* to *sink* in this DAG. Can you find a recurrence relation for an arbitrary DAG?



A Recurrence for an Arbitrary DAG?

Let $s(b)$ be the length of a longest path from *source* to b.

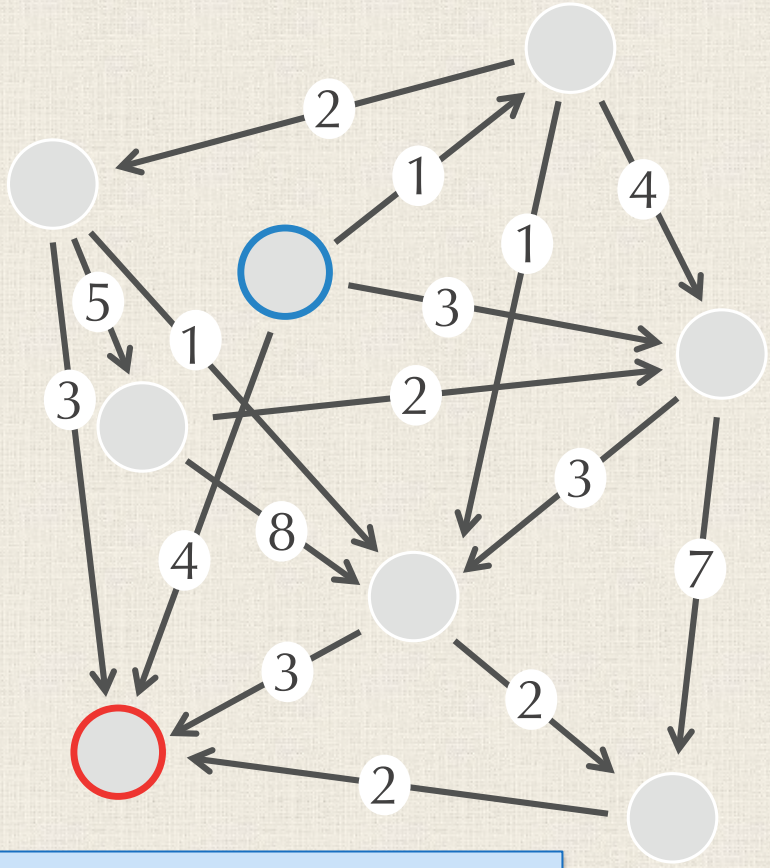
If there is an edge connecting a to b, we call a a **predecessor** of b.



A Recurrence for an Arbitrary DAG?

Let $s(b)$ be the length of a longest path from *source* to *sink*.

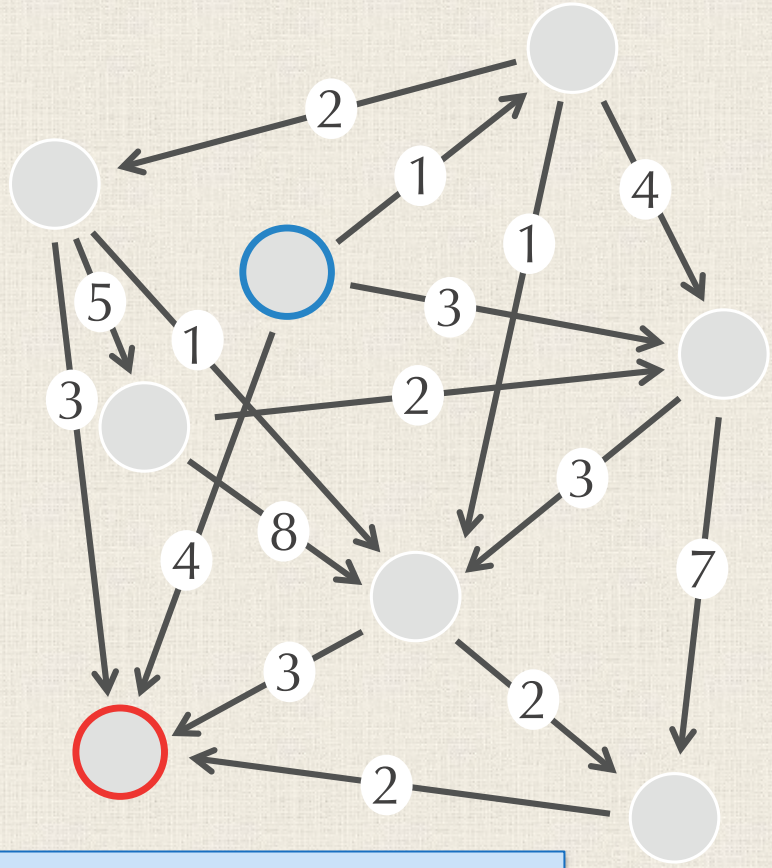
If there is an edge connecting a to b, we call a a **predecessor** of b.



$$s_b = \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge from } a \text{ to } b\}$$

A Recurrence for an Arbitrary DAG?

STOP: What makes computing this recurrence difficult?

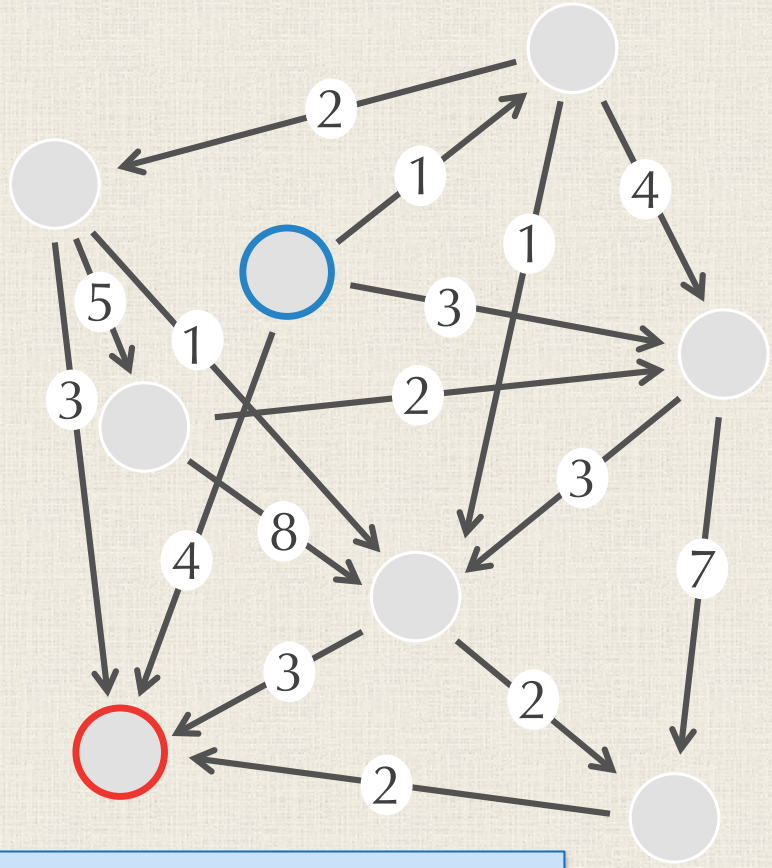


$$s_b = \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge from } a \text{ to } b\}$$

A Recurrence for an Arbitrary DAG?

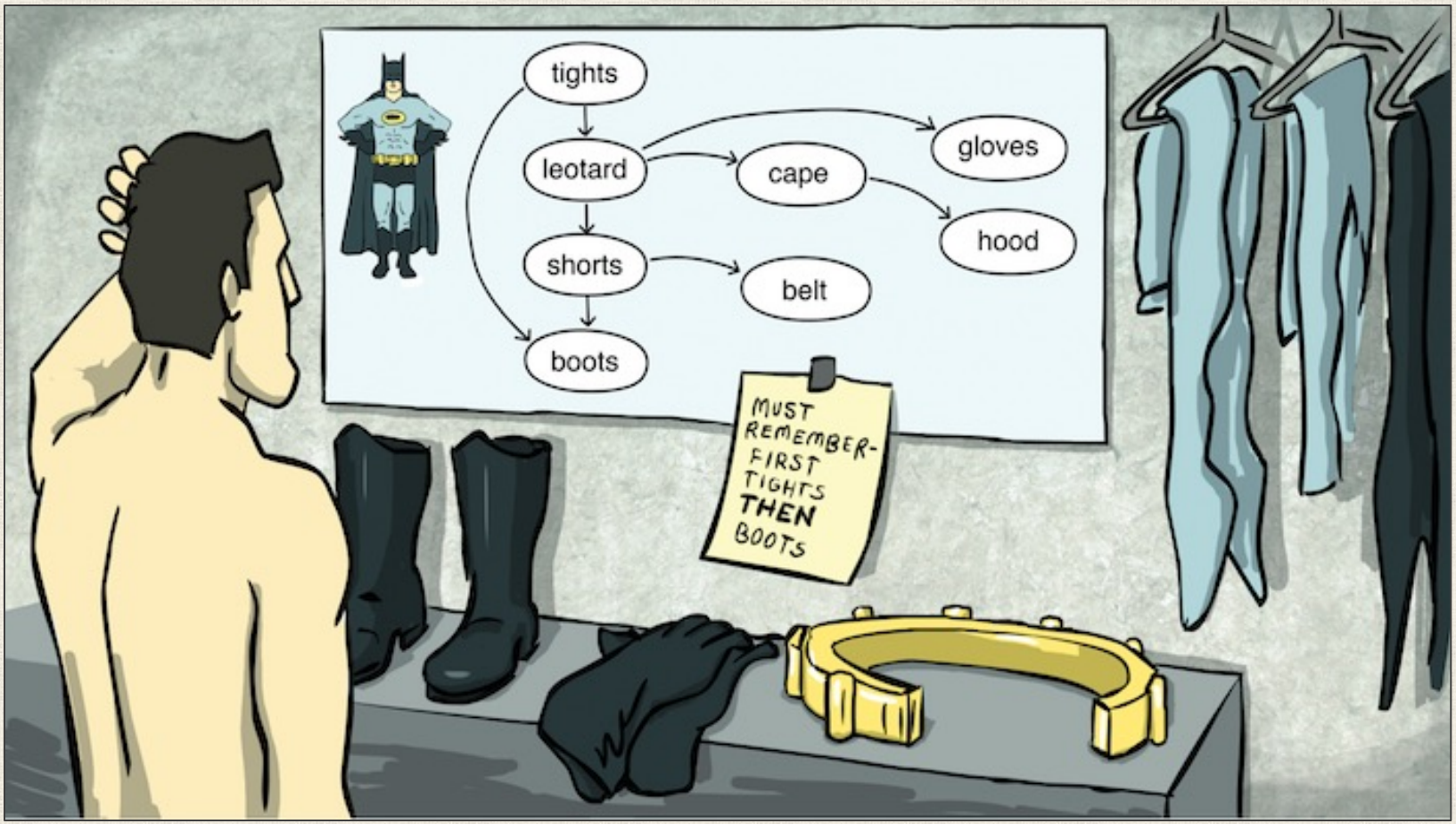
STOP: What makes computing this recurrence difficult?

Answer: We need to know the *order* to consider the nodes.



$$s_b = \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge from } a \text{ to } b\}$$

“Dressing Challenge”: Ordering Nodes in a DAG



Topological Orderings

The critical part of computing $s(b)$ is ensuring that $s(a)$ has already been computed for all predecessors.

That is, we need to have an *ordering* of the nodes in a DAG so that no node is considered before its predecessor.

Topological Orderings

The critical part of computing $s(b)$ is ensuring that $s(a)$ has already been computed for all predecessors.

That is, we need to have an *ordering* of the nodes in a DAG so that no node is considered before its predecessor.

An ordering of nodes $(a_1, \ldots, a_k)$ of nodes in a DAG is a **topological ordering** if every edge $a_i \rightarrow a_j$ is such that $i < j$.

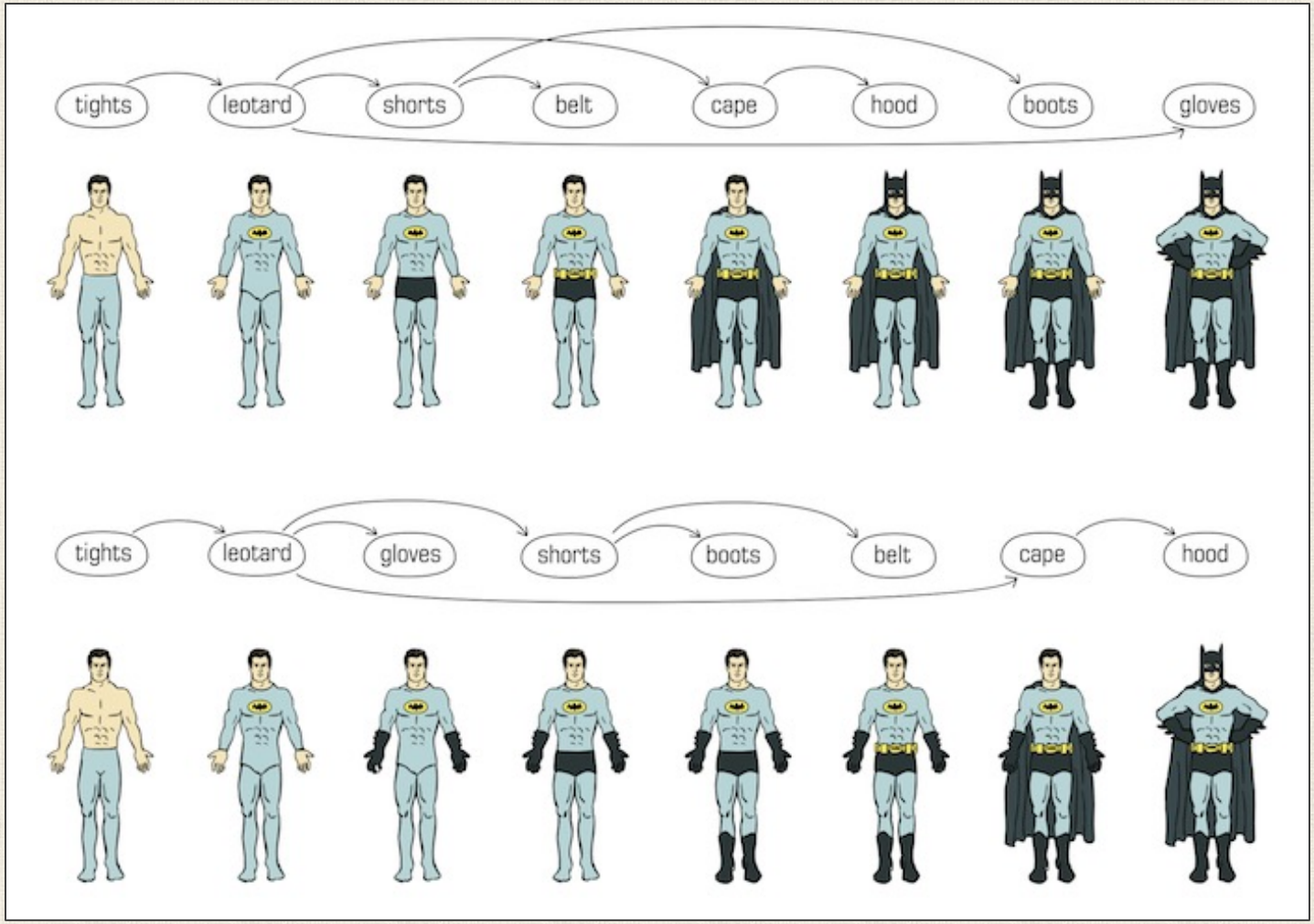
Topological Orderings

The critical part of computing $s(b)$ is ensuring that $s(a)$ has already been computed for all predecessors.

Theorem: Every DAG must have at least one topological ordering (and there is an algorithm for finding it).

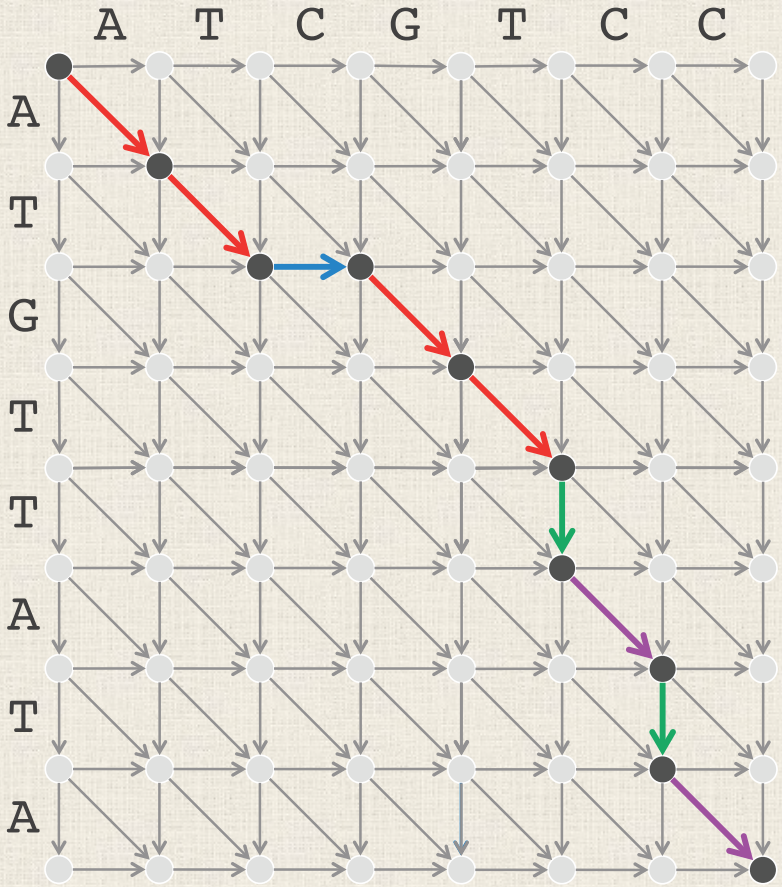
An ordering of nodes $(a_1, \ldots, a_k)$ of nodes in a DAG is a **topological ordering** if every edge $a_i \rightarrow a_j$ is such that $i < j$.

Two Topological Orderings for Dressing DAG

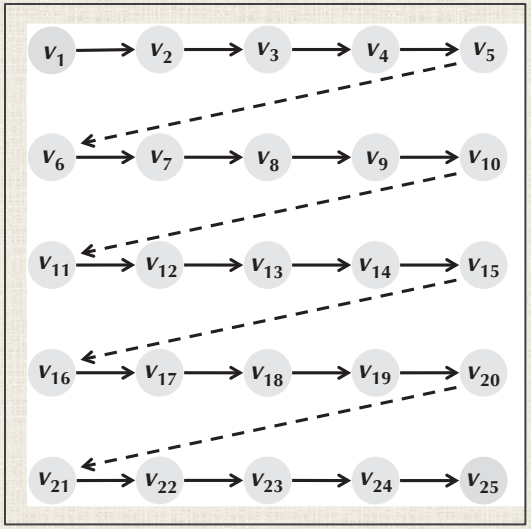


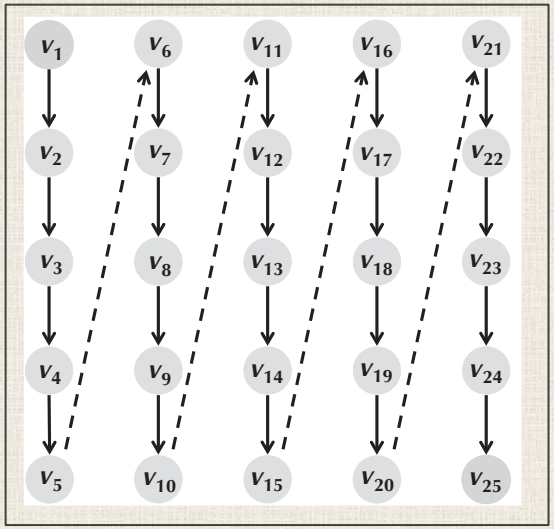
Topological Orderings for the Alignment Graph

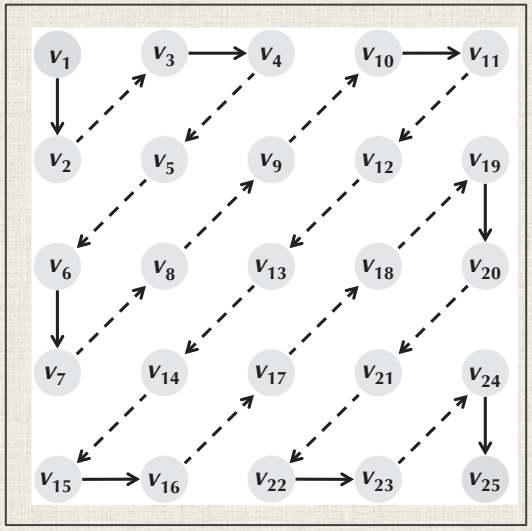
STOP: What topological order(s) do you see for the alignment graph?



Three Topological Orderings for the Alignment Graph







Pseudocode for Finding Length of Longest Path

LongestPath(*Graph*, *source*, *sink*)
  for each node *b* in *Graph*
    $s_b \leftarrow -\infty$
  $s_{source} \leftarrow 0$
  topologically order *Graph*
  for each node *b* in *Graph* (following the topological order)
    $s_b \leftarrow \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge } a \rightarrow b\}$
  return s_{sink}

STOP: What is the approximate ("big O" for the initiated) runtime of **LongestPath**?

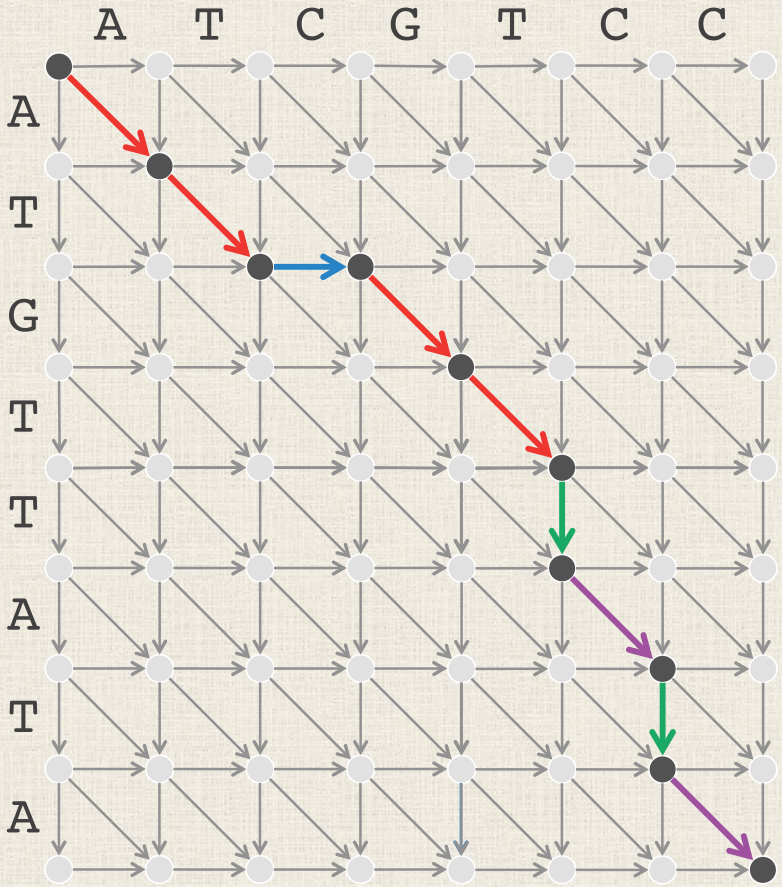
Pseudocode for Finding Length of Longest Path

LongestPath(*Graph*, *source*, *sink*)
  for each node *b* in *Graph*
    $s_b \leftarrow -\infty$
  $s_{source} \leftarrow 0$
  topologically order *Graph*
  for each node *b* in *Graph* (following the topological order)
    $s_b \leftarrow \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge } a \rightarrow b\}$
  return s_{sink}

Answer: We consider each edge exactly once, so (if we know a topological order) the runtime is proportional to the number of *edges*.

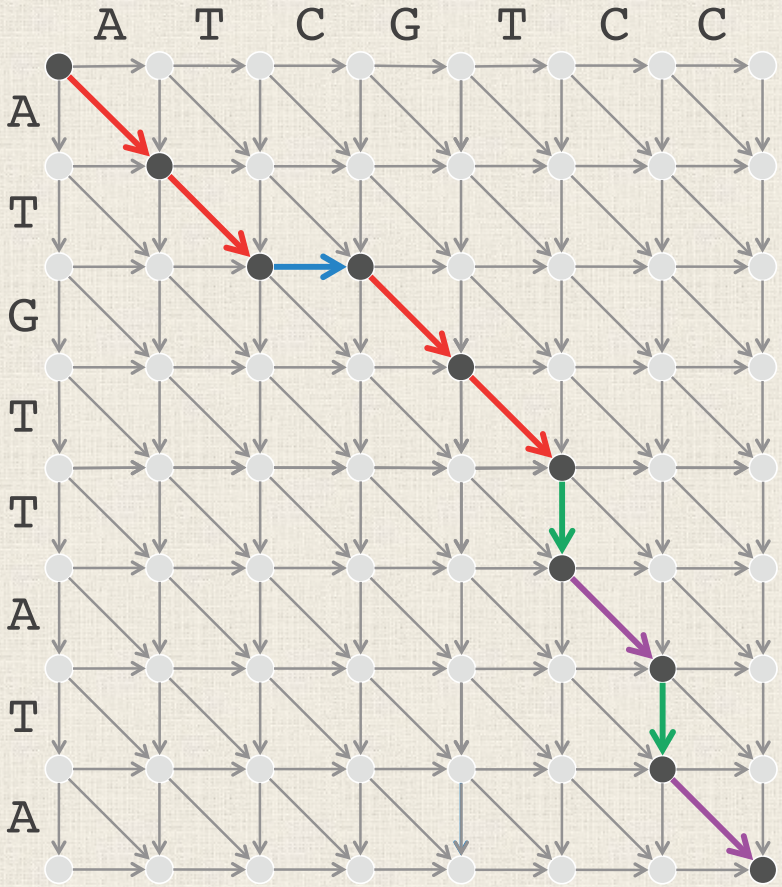
Topological Orderings for the Alignment Graph

STOP: How many edges does the alignment graph of strings *v* and *w* have?



Topological Orderings for the Alignment Graph

STOP: How many edges does the alignment graph of strings *v* and *w* have?



Answer: Each node has 0, 1, or 3 predecessors. So, the number of edges is proportional to $|v| \cdot |w|$.

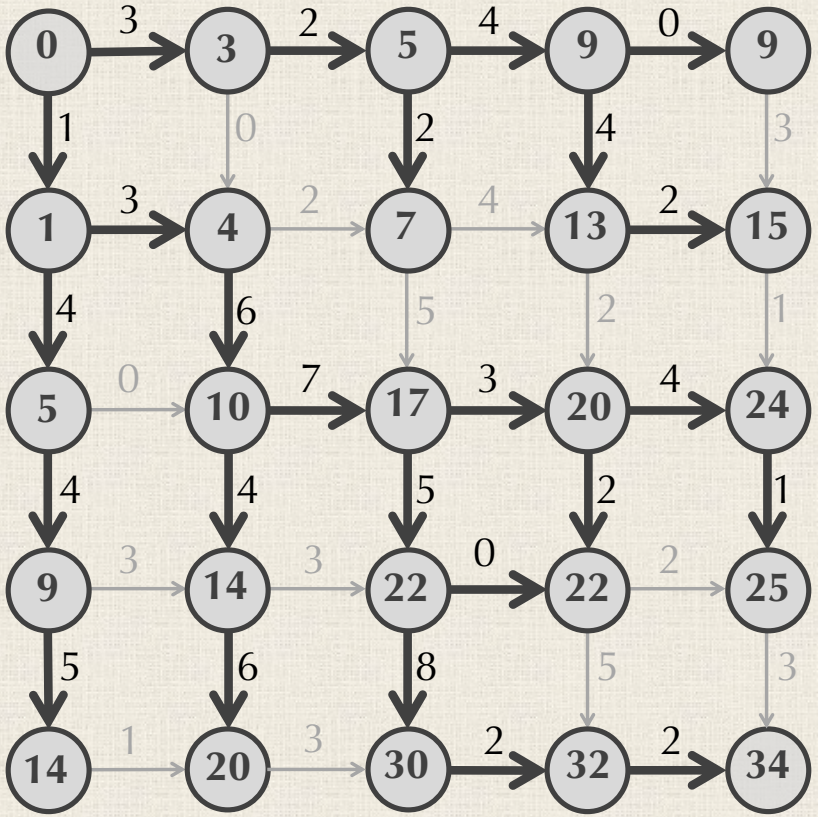
From Finding the Maximum Length to Finding a Path

LongestPath(*Graph*, *source*, *sink*)
  for each node *b* in *Graph*
    $s_b \leftarrow -\infty$
  $s_{source} \leftarrow 0$
  topologically order *Graph*
  for each node *b* in *Graph* (following the topological order)
    $s_b \leftarrow \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge } a \rightarrow b\}$
  return s_{sink}

Note: We can find the length of a longest path, but we still don't know how to *construct* a longest path.

Finding a Longest Path

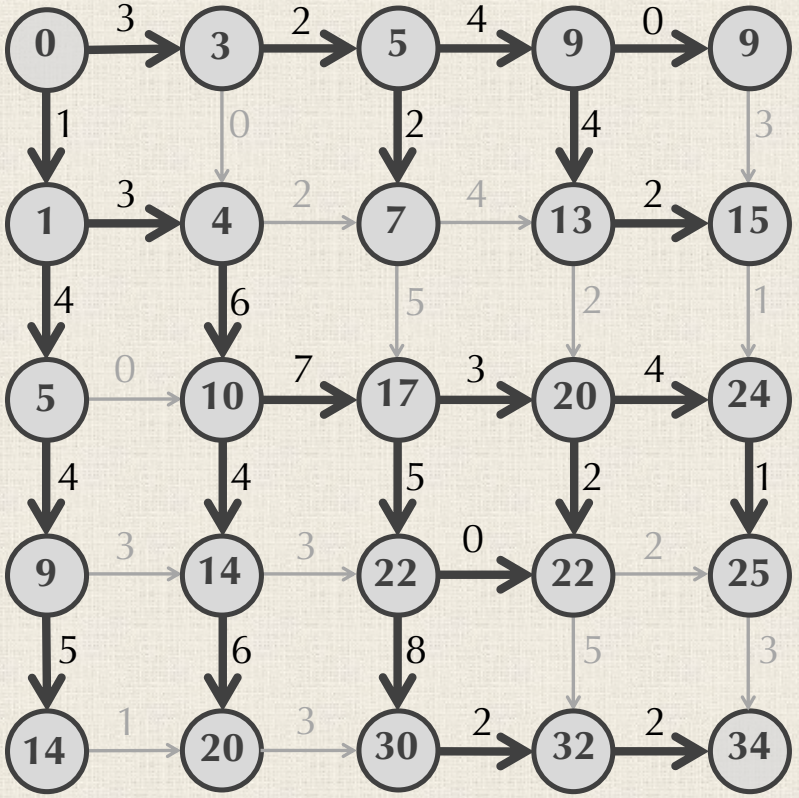
STOP: Take a moment to look at our solution from before when we found the maximum weight of a path. How might we have reconstructed the longest path?



BACKTRACKING IN THE ALIGNMENT GRAPH

From a Recurrence to a Longest Path

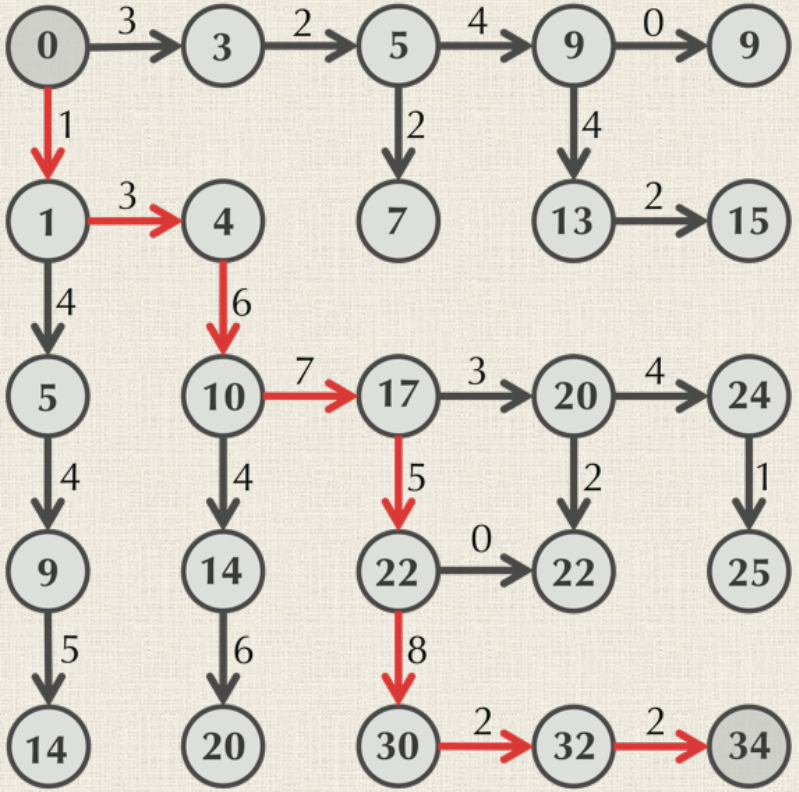
Note: we highlighted the edge used at each node when computing length of longest path.



From a Recurrence to a Longest Path

Note: we highlighted the edge used at each node when computing length of longest path.

We remember *one* predecessor at each node, so following predecessors backward from sink yields longest path!



Recall that these Problems are the Same

Longest Common Subsequence Length Problem:.

- **Input:** Two strings.
- **Output:** The length of a longest common subsequence of these strings.

Symbol Matching Problem:

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any "alignment" of the two strings.

Putting it All Together

Longest Common Subsequence ~~Length~~ Problem:.

- **Input:** Two strings.
- **Output:** A ~~length of a~~ longest common subsequence of these strings.

STOP: How can we find an LCS of two strings?

Putting it All Together

Longest Common Subsequence ~~Length~~ Problem:.

- **Input:** Two strings.
- **Output:** A ~~length of a~~ longest common subsequence of these strings.

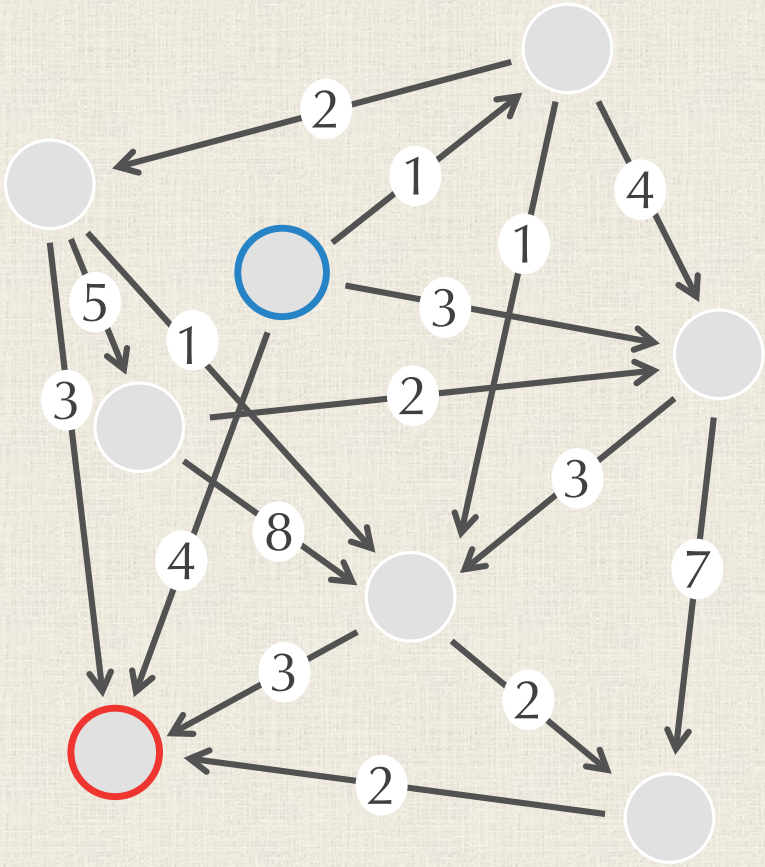
Answer:

1. Build the alignment graph, with ″match″ edges weighted 1.
2. Find the length of an LCS using recurrence relation.
3. Backtrack to find longest path.

Backtracking in an Arbitrary DAG

$$s_b = \max_{\text{all predecessors } a \text{ of node } b} \{s_a + \text{weight of edge from } a \text{ to } b\}$$

When computing the recurrence, we store a "pointer" to the predecessor node *a* that achieved the maximum.



GLOBAL ALIGNMENT

Strengthening Alignment Scoring

```
A   T   -   G   T   T   A   T   A
A   T   C   G   T   -   C   -   C
```

Alignment score: Divided into three components:

- **match** reward (+1)
- **mismatch** penalty (-μ)
- **insertion**/**deletion** penalty (-σ)

STOP: What were μ and σ when finding a longest common subsequence?

Strengthening Alignment Scoring

```
A   T   -   G   T   T   A   T   A
A   T   C   G   T   -   C   -   C
```

Alignment score: Divided into three components:

- **match** reward (+1)
- **mismatch** penalty (-μ)
- **insertion**/**deletion** penalty (-σ)

Answer: They were both equal to zero…

Strengthening Alignment Scoring

Global Alignment Problem: *Find a highest-scoring alignment of two strings*.

- **Input:** Two strings and numbers μ and σ .
- **Output:** An alignment of the strings with maximum alignment score using these parameters.

Strengthening Alignment Scoring

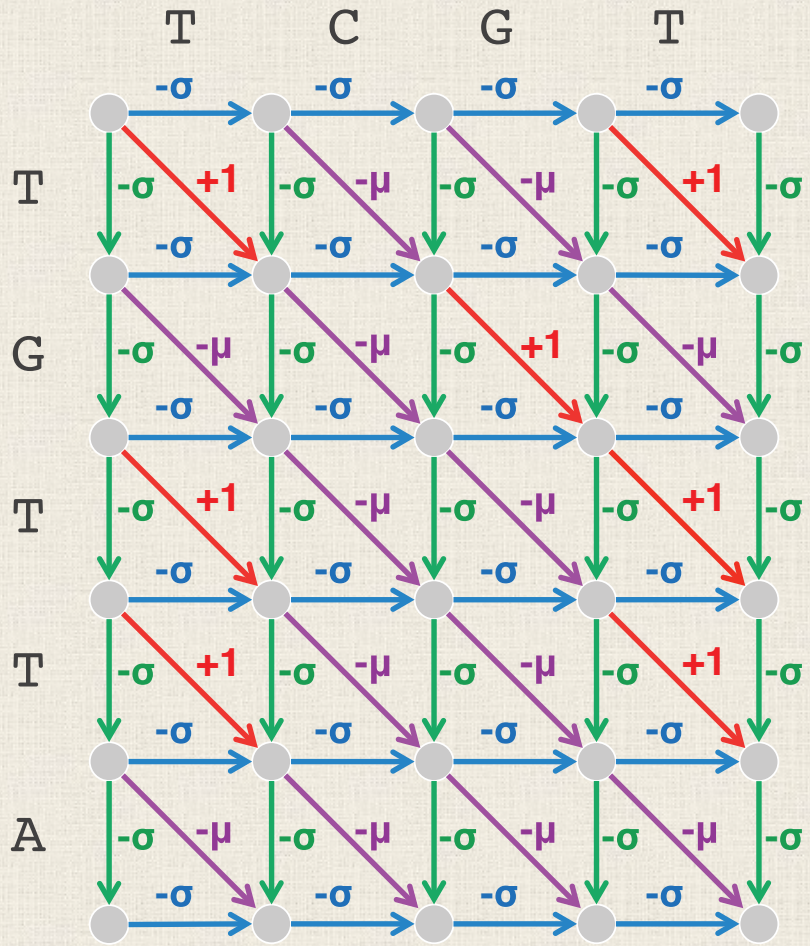
Global Alignment Problem: *Find a highest-scoring alignment of two strings.*

- **Input:** Two strings and numbers μ and σ .
- **Output:** An alignment of the strings with maximum alignment score using these parameters.

STOP: How can we modify the alignment network to solve this problem?

Strengthening Alignment Scoring

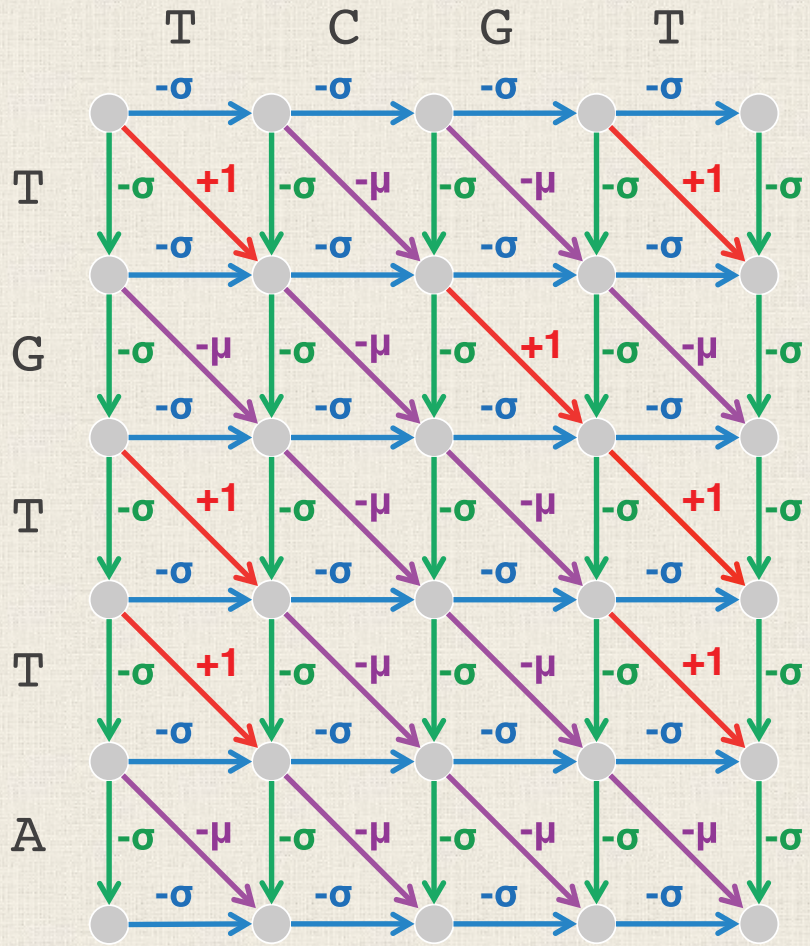
Answer: Slight modification to alignment network ... a longest path will yield an alignment of maximum score!



Strengthening Alignment Scoring

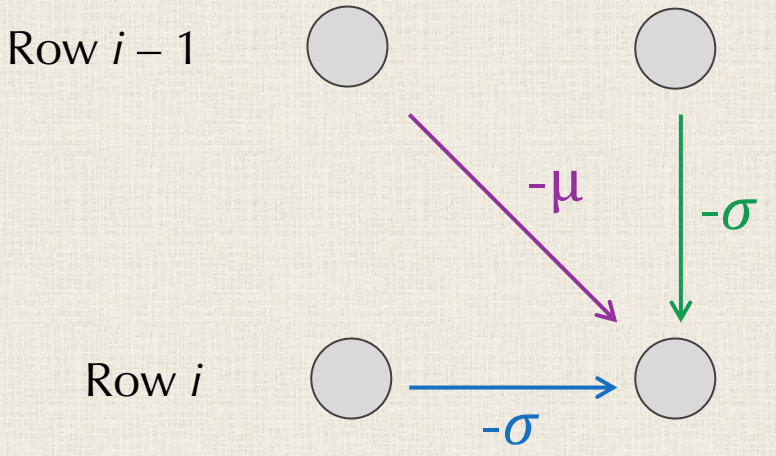
Answer: Slight modification to alignment network ... a longest path will yield an alignment of maximum score!

Exercise: What is the recurrence relation?



Two Cases: Mismatch vs. Match

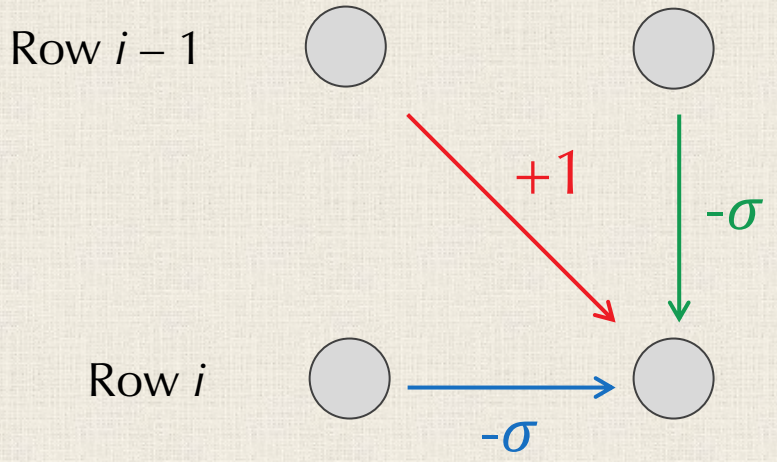
Case 1



length(i, j) = maximum of:

- *length*$(i-1, j) - \sigma$
- *length*$(i, j-1) - \sigma$
- *length*$(i-1, j-1) - \mu$

Case 2



length(i, j) = maximum of:

- *length*$(i-1, j) - \sigma$
- *length*$(i, j-1) - \sigma$
- *length*$(i-1, j-1) + 1$

Further Strengthening Scoring with a Scoring Matrix

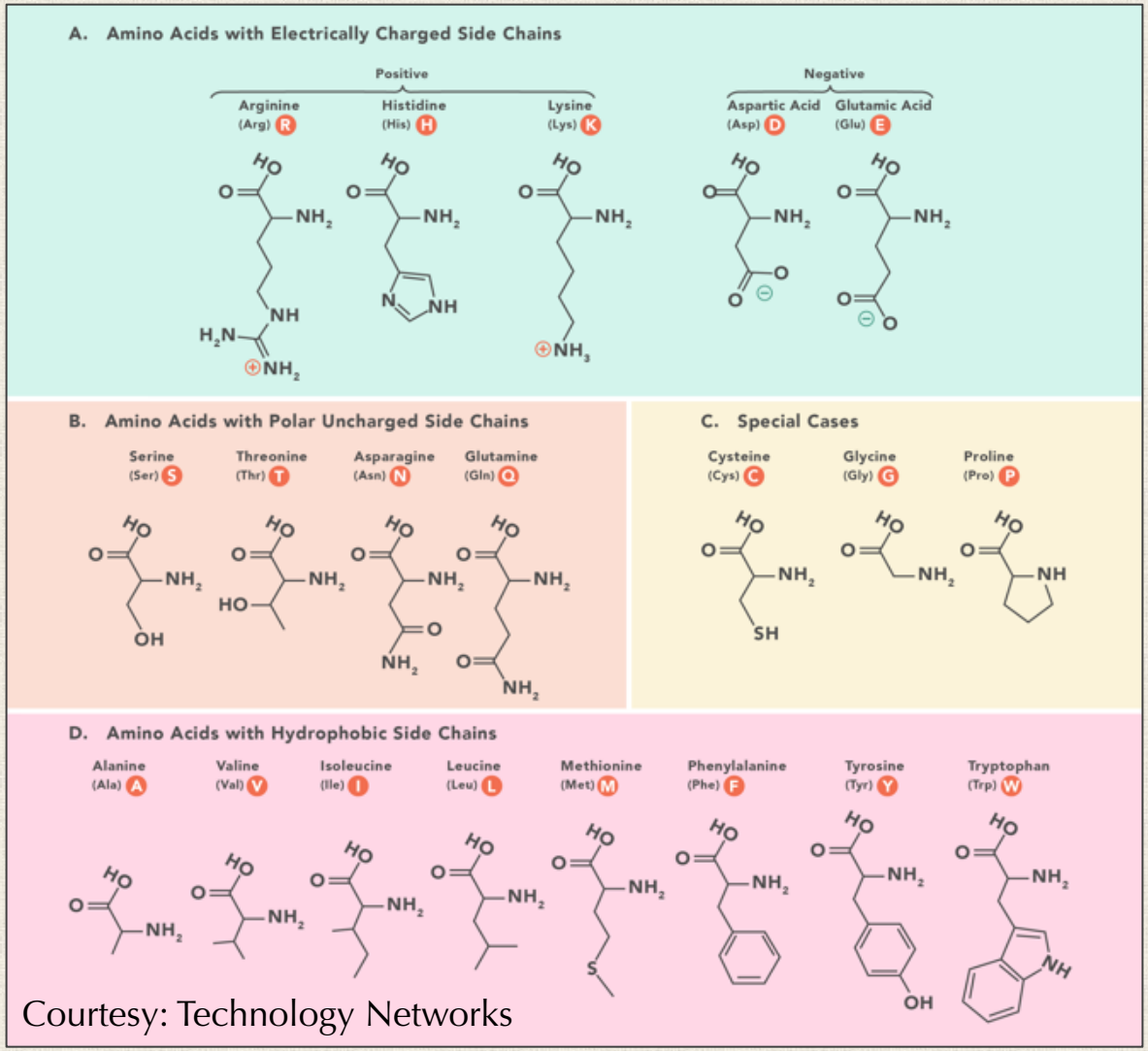
Scoring matrix: Penalizes indels and matches/mismatches differently depending on individual symbols.

STOP: How do you think this matrix was computed?

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y	–
A	2	-2	0	0	-3	1	-1	-1	-1	-2	-1	0	1	0	-2	1	1	0	-6	-3	-8
C	-2	12	-5	-5	-4	-3	-3	-2	-5	-6	-5	-4	-3	-5	-4	0	-2	-2	-8	0	-8
D	0	-5	4	3	-6	1	1	-2	0	-4	-3	2	-1	2	-1	0	0	-2	-7	-4	-8
E	0	-5	3	4	-5	0	1	-2	0	-3	-2	1	-1	2	-1	0	0	-2	-7	-4	-8
F	-3	-4	-6	-5	9	-5	-2	1	-5	2	0	-3	-5	-5	-4	-3	-3	-1	0	7	-8
G	1	-3	1	0	-5	5	-2	-3	-2	-4	-3	0	0	-1	-3	1	0	-1	-7	-5	-8
H	-1	-3	1	1	-2	-2	6	-2	0	-2	-2	2	0	3	2	-1	-1	-2	-3	0	-8
I	-1	-2	-2	-2	1	-3	-2	5	-2	2	2	-2	-2	-2	-2	-1	0	4	-5	-1	-8
K	-1	-5	0	0	-5	-2	0	-2	5	-3	0	1	-1	1	3	0	0	-2	-3	-4	-8
L	-2	-6	-4	-3	2	-4	-2	2	-3	6	4	-3	-3	-2	-3	-3	-2	2	-2	-1	-8
M	-1	-5	-3	-2	0	-3	-2	2	0	4	6	-2	-2	-1	0	-2	-1	2	-4	-2	-8
N	0	-4	2	1	-3	0	2	-2	1	-3	-2	2	0	1	0	1	0	-2	-4	-2	-8
P	1	-3	-1	-1	-5	0	0	-2	-1	-3	-2	0	6	0	0	1	0	-1	-6	-5	-8
Q	0	-5	2	2	-5	-1	3	-2	1	-2	-1	1	0	4	1	-1	-1	-2	-5	-4	-8
R	-2	-4	-1	-1	-4	-3	2	-2	3	-3	0	0	0	1	6	0	-1	-2	2	-4	-8
S	1	0	0	0	-3	1	-1	-1	0	-3	-2	1	1	-1	0	2	1	-1	-2	-3	-8
T	1	-2	0	0	-3	0	-1	0	0	-2	-1	0	0	-1	-1	1	3	0	-5	-3	-8
V	0	-2	-2	-2	-1	-1	-2	4	-2	2	2	-2	-1	-2	-2	-1	0	4	-6	-2	-8
W	-6	-8	-7	-7	0	-7	-3	-5	-3	-2	-4	-4	-6	-5	2	-2	-5	-6	17	0	-8
Y	-3	0	-4	-4	7	-5	0	-1	-4	-1	-2	-2	-5	-4	-4	-3	-3	-2	0	10	-8
–	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	

PAM250 matrix

Amino acids' side chain variety produces different chemical properties



Courtesy: Technology Networks

A Quick Aside About the BLOSUM Scoring Matrices

BLOSUM62 miscalculations improve search performance

Mark P Styczynski, Kyle L Jensen, Isidore Rigoutsos & Gregory Stephanopoulos

Nature Biotechnology **26**, 274–275(2008) | Cite this article

To the editor:

The BLOSUM[1] family of substitution matrices, and particularly BLOSUM62, is the *de facto* standard in protein database searches and sequence alignments. In the course of analyzing the evolution of the Blocks database[2], we noticed errors in the software source code used to create the initial BLOSUM family of matrices (available online at ftp://ftp.ncbi.nih.gov/repository/blocks/unix/blosum/blosum.tar.Z). The result of these errors is that the BLOSUM matrices—BLOSUM62, BLOSUM50, etc.—are quite different from the matrices that should have been calculated using the algorithm described by Henikoff and Henikoff[1]. Obviously, minor errors in research, and particularly in software source code, are quite common. This case is noteworthy for three reasons: first, the BLOSUM matrices are ubiquitous in computational biology; second, these errors have gone unnoticed for 15 years; and third, the 'incorrect' matrices perform better than the 'intended' matrices.

Strengthening Global Alignment

Global Alignment Problem: *Find a highest-scoring alignment of two strings.*

- **Input:** Two strings and a scoring matrix.
- **Output:** An alignment of the strings with maximum alignment score according to the scoring matrix.

STOP: How does this change the alignment graph?

Strengthening Global Alignment

Global Alignment Problem: *Find a highest-scoring alignment of two strings.*

- **Input:** Two strings and a scoring matrix.
- **Output:** An alignment of the strings with maximum alignment score according to the scoring matrix.

Answer: Every edge simply gets weighted with the cost of the corresponding scoring matrix value.

Summarizing our Global Alignment Algorithm

1. Form a 2-D array using the recurrence relation for dynamic programming.
2. Create array containing "backtracking pointers".
3. After reaching the sink, backtrack to source to produce a maximum-weight path.
4. Infer the alignment corresponding to this path.


www.sciencedirect.com › science › article › pii ⋮

A general method applicable to the search for similarities in ...

by SB Needleman · 1970 · Cited by 14225 · Related articles

A computer adaptable **method** for **finding similarities in the amino acid sequences of two proteins** has been developed. From these findings it is possible to determine whether significant homology exists between the **proteins**. This information is used to trace their possible evolutionary development.

Summarizing our Global Alignment Algorithm

STOP (biologists): Would you rather align two genes as DNA strings (nucleotides) or as proteins (amino acids)?

Summarizing our Global Alignment Algorithm

Answer: *If* we know that the genes wind up as protein, then a protein-level function will be more informative since there is a larger alphabet and the amino acids determine function of the protein.


www.sciencedirect.com › science › article › pii

A general method applicable to the search for similarities in ...

by SB Needleman · 1970 · Cited by 14225 · Related articles

A computer adaptable **method** for **finding similarities in the amino acid sequences of two proteins** has been developed. From these findings it is possible to determine whether significant homology exists between the **proteins**. This information is used to trace their possible evolutionary development.

Applying to Real Data

STOP: Let's apply this to the same protein (say, hemoglobin subunit alpha) in a few different species. What do you think we will see?

- *Homo sapiens* vs. *Gorilla gorilla gorilla*
- *Homo sapiens* vs. *Bos Taurus* (cow)
- *Homo sapiens* vs. *Danio rerio* (zebrafish)

Homo sapiens:	https://www.uniprot.org/uniprot/P69905
Gorilla gorilla gorilla:	https://www.uniprot.org/uniprot/P01923
Bos taurus:	https://www.uniprot.org/uniprot/P01966
Danio rerio:	https://www.uniprot.org/uniprot/Q90487

EMBOSS "Needle" server: https://www.ebi.ac.uk/Tools/psa/emboss_needle/

Results of Hemoglobin Alignments

Note: "|" means exact similarity, ":" means strong similarity, and "." means weak similarity.

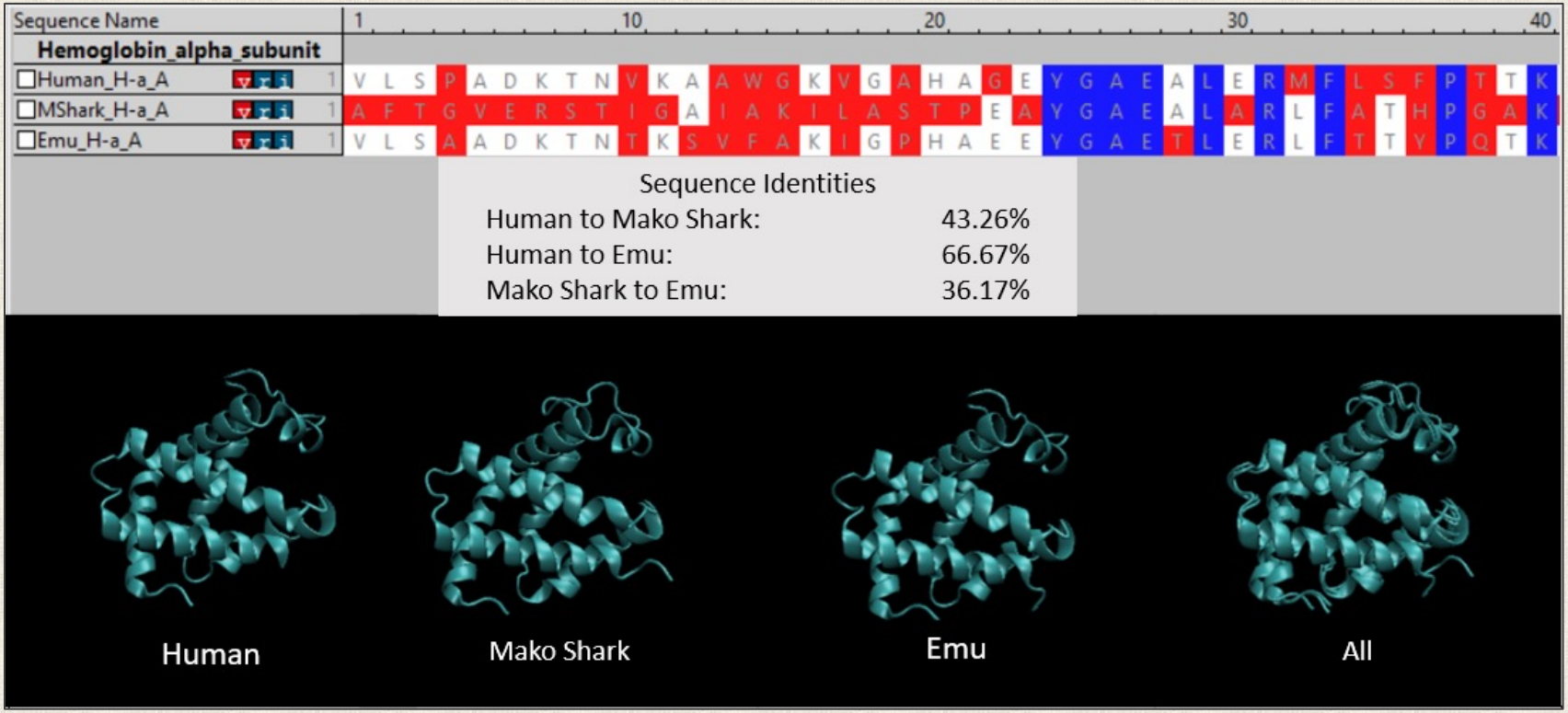
STOP: What is our hypothesis?

```
HBA_HUMAN          1 MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLS     50
                     |||||||||||||||||||||||:|||||||||||||||||||||||||||
HBA_GORGO          1 -VLSPADKTNVKAAWGKVGAHAGDYGAEALERMFLSFPTTKTYFPHFDLS     49

HBA_HUMAN         51 HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFK    100
                     ||||||||||||||||||||||||||||||||||||||||||||||||||||
HBA_GORGO         50 HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFK     99

HBA_HUMAN        101 LLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    142
                     |||||||||||||||||||||||||||||||||||||||||||
HBA_GORGO        100 LLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    141
```

```
HBA_HUMAN          1 MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLS     50
                     ||||.|||.||||||||||.||.|||||||||||||||||||||||||||||
HBA_BOVIN          1 MVLSAADKGNVKAAWGKVGGHAAEYGAEALERMFLSFPTTKTYFPHFDLS     50

HBA_HUMAN         51 HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFK    100
                     ||||||||||.|||.|||.||.|:||:|.|||.||||||||||||||||||
HBA_BOVIN         51 HGSAQVKGHGAKVAAALTKAVEHLDDLPGALSELSDLHAHKLRVDPVNFK    100

HBA_HUMAN        101 LLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    142
                     ||||.||||||:|||::|||||||||||||:||||||||||
HBA_BOVIN        101 LLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR    142
```

```
HBA_HUMAN          1 MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DL     49
                     |.||..||..|||.|.|:...|.|.|||||.||...:|.|||||.|: ||
HBA_DANRE          1 MSLSDTDKAVVKAIWAKISPKADEIGAEALARMLTVYPQTKTYFSHWADL     50

HBA_HUMAN         50 SHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNF     99
                     |.||..||.|||.:..|:..|::.:||:...|:|||:|||.||||||.||
HBA_DANRE         51 SPGSGPVKKHGKTIMGAVGEAISKIDDLVGGLAALSELHAFKLRVDPANF    100

HBA_HUMAN        100 KLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    142
                     |:|||.::|.:|...||:|||.||.|:|||..:::..|:.|||
HBA_DANRE        101 KILSHNVIVVIAMLFPADFTPEVHVSVDKFFNNLALALSEKYR    143
```

Homologous proteins may have different sequences but similar structures



Cold Takes Exposed: Biology c. 1963

From the point of view of hemoglobin structure, it appears that gorilla is just an abnormal human.

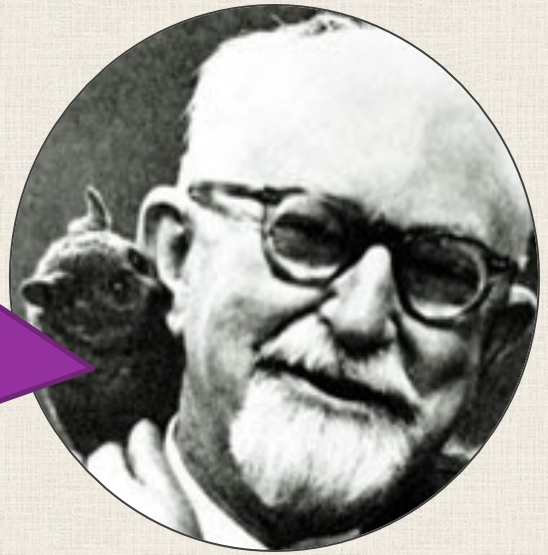
Émile Zuckerkandl

Cold Takes Exposed: Biology c. 1963

From the point of view of hemoglobin structure, it appears that gorilla is just an abnormal human.

Émile Zuckerkandl

...that is of course nonsense. What the comparison really indicates is that hemoglobin is a bad choice and has nothing to tell us about attributes.

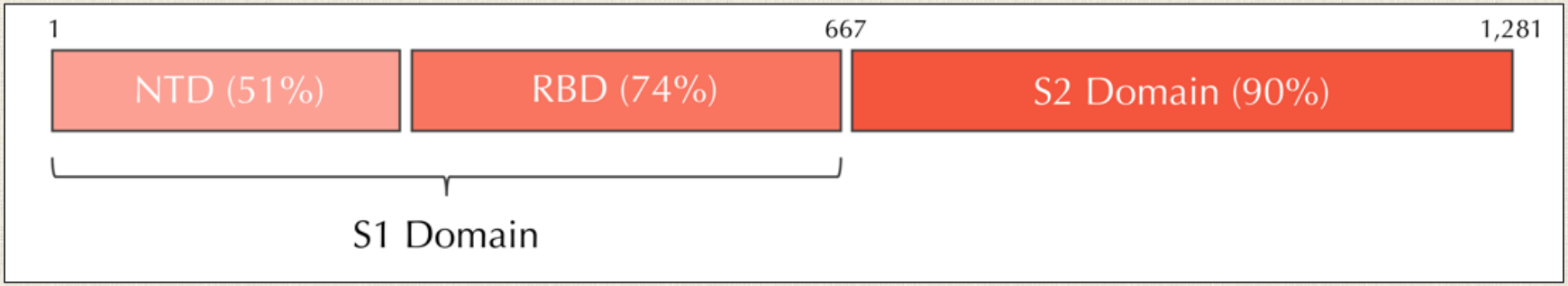


Gaylord Simpson

FROM GLOBAL TO LOCAL ALIGNMENT

Finding "Local" Similarities

Real genes have *variable* and *conserved* regions; the figure below shows the sequence similarity of the spike protein between SARS-CoV and SARS-CoV-2.



We also will need "local" alignment to compare genes against a database

Database Comparison Problem:

- **Input:** A string *query* and a (much longer) string *database*.
- **Output:** One or more "high-scoring" similarities between *query* (or a substring of *query*) and some substring of *database*.

This (poorly defined) problem is probably the most frequent application in computational biology.

Finding "Local" Similarities

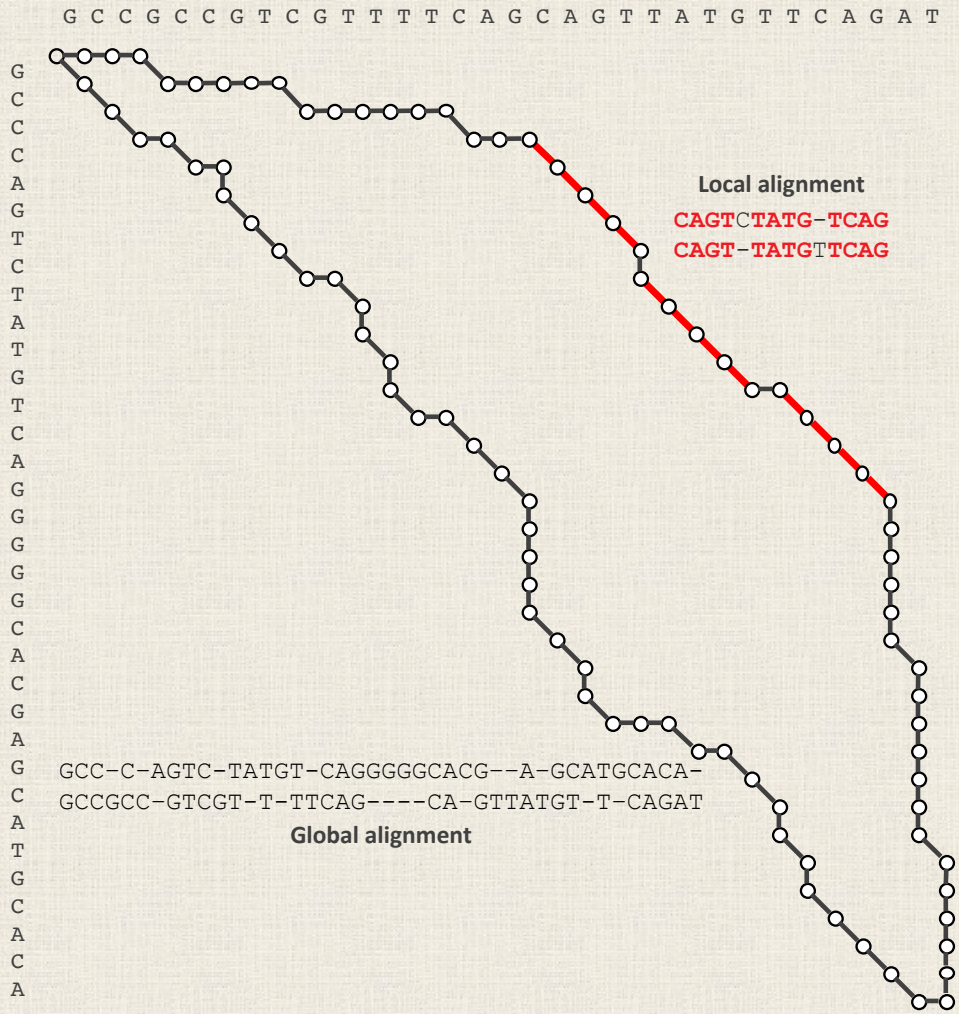
```
GCC-C-AGTC-TATGT-CAGGGGGCACG--A-GCATGCACA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATGT-T-CAGAT
```

Exercise: Score these alignments ($\sigma = \mu = 1$). Which alignment is "better"? Which gets the higher score?

```
---G----C------C--CAGTCTATG-TCAGGGGGCACGAGCATGCACA
GCCGCCGTCGTTTTCAGCAGT-TATGTTCAG------A-------T-----
```

Visualizing Local Alignments

Local alignments may be well away from "main diagonal" because they have a lot of indels on ends of the alignment.



Revisiting Global Alignment

Global Alignment Problem:

- **Input:** Two strings and a scoring matrix.
- **Output:** An alignment of the strings with maximum alignment score according to the scoring matrix.

Revisiting Global Alignment

Global Alignment Problem:

- **Input:** Two strings and a scoring matrix.
- **Output:** An alignment of the strings with maximum alignment score according to the scoring matrix.

STOP: How can we reformulate the problem statement to find areas of "local" similarity?

Revisiting Global Alignment

Local Alignment Problem:

- **Input:** Two strings *v* and *w* and a scoring matrix.
- **Output:** Substrings of *v* and *w* whose best global alignment score is maximized over all substrings.

Revisiting Global Alignment

Local Alignment Problem:

- **Input:** Two strings *v* and *w* and a scoring matrix.
- **Output:** Substrings of *v* and *w* whose best global alignment score is maximized over all substrings.

STOP: One idea for solving this is to solve the Global Alignment Problem for every pair of substrings of *v* and *w*. Why is this an issue?

Revisiting Global Alignment

Local Alignment Problem:

- **Input:** Two strings v and w and a scoring matrix.
- **Output:** Substrings of v and w whose best global alignment score is maximized over all substrings.

Answer: There are $C(|v|, 2)$ substrings of v and $C(|w|, 2)$ substrings of w. As a result we have about $|v|^2|w|^2$ alignments to construct!

This was understood in 1970, and yet the problem remained open …

Ten Years Go By …

A general method applicable to the search for similarities in the amino ...

https://www.sciencedirect.com/science/article/pii/0022283670900574

by SB Needleman - 1970 - Cited by 12553 - Related articles

A computer adaptable **method** for **finding similarities in the amino acid sequences of two proteins** has been developed. From these findings it is possible to determine whether significant homology exists between the **proteins**. ... The maximum match is a number dependent upon the **similarity** of the **sequences**.

Identification of common molecular subsequences. - NCBI

https://www.ncbi.nlm.nih.gov/pubmed/7265238 ▼

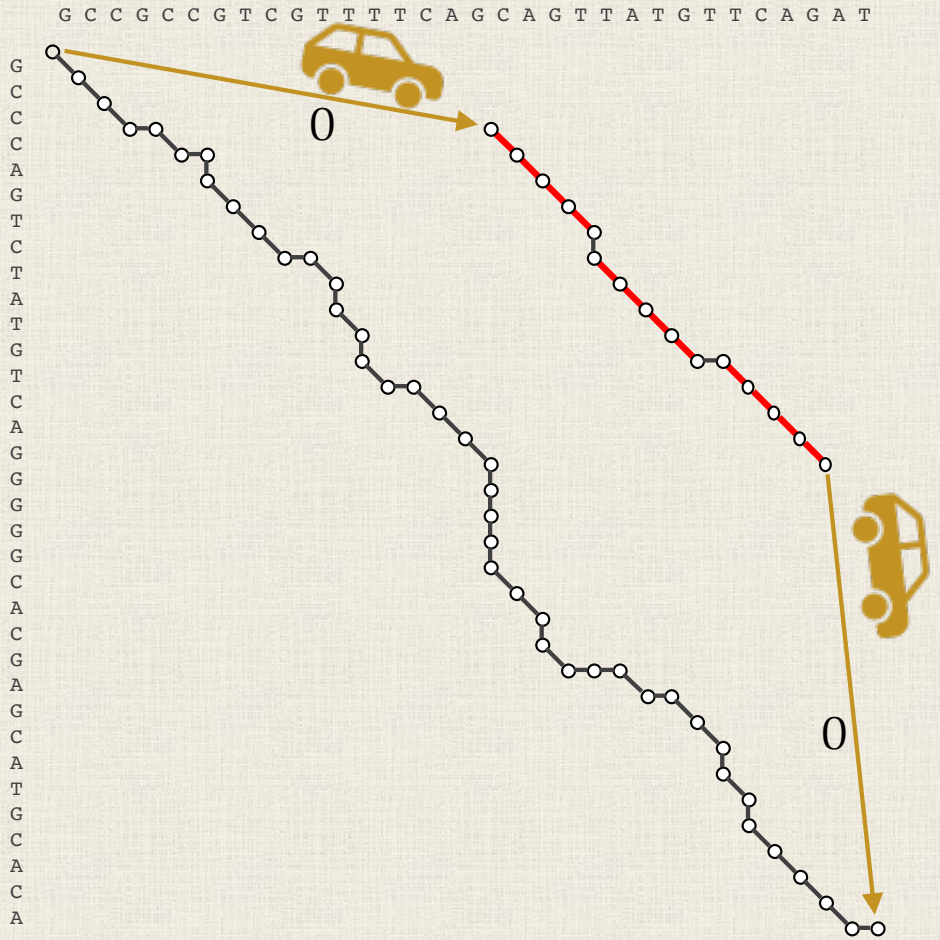
by TF Smith - 1981 - Cited by 11181 - Related articles

Identification of common molecular subsequences. Smith TF, Waterman MS. PMID: 7265238; [Indexed for ... MeSH terms. Base **Sequence***; Models, Chemical *

“Free Rides” for Local Alignment

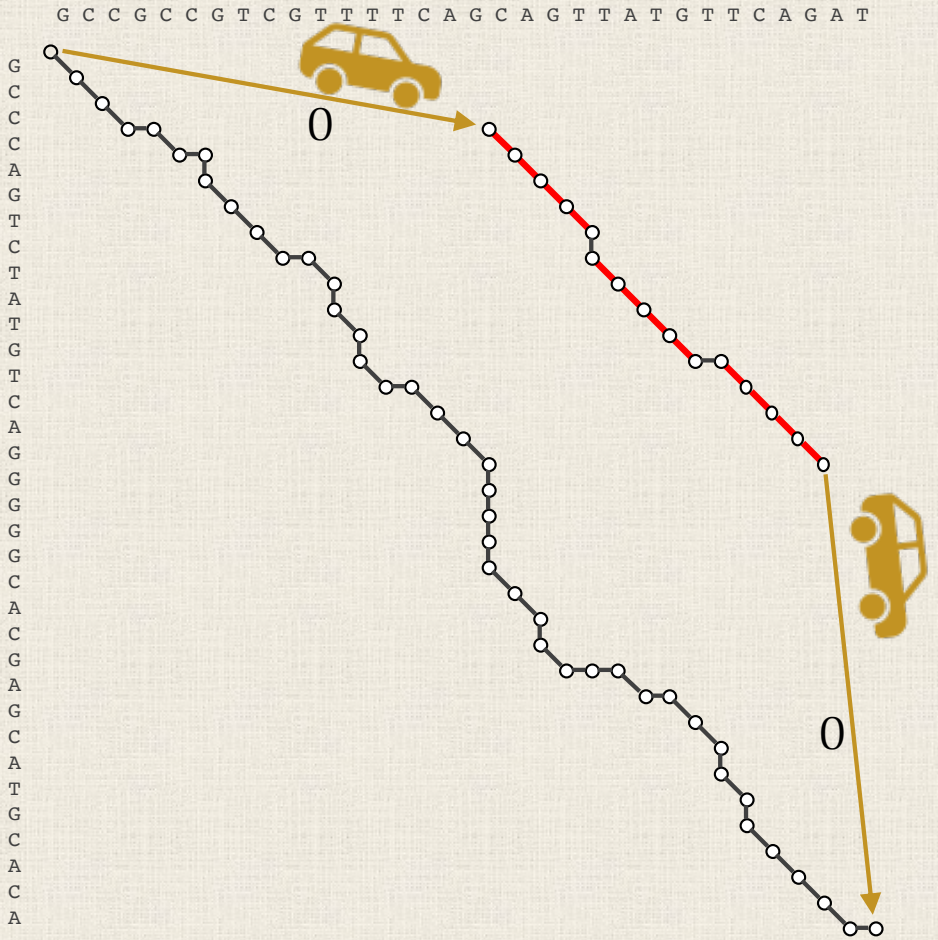
Add a zero-weight edge from the source to every node and the sink to every node.

This will allow a local alignment to start and end anywhere with no penalty.



"Free Rides" for Local Alignment

Exercise: What is the recurrence relation for the local alignment problem?

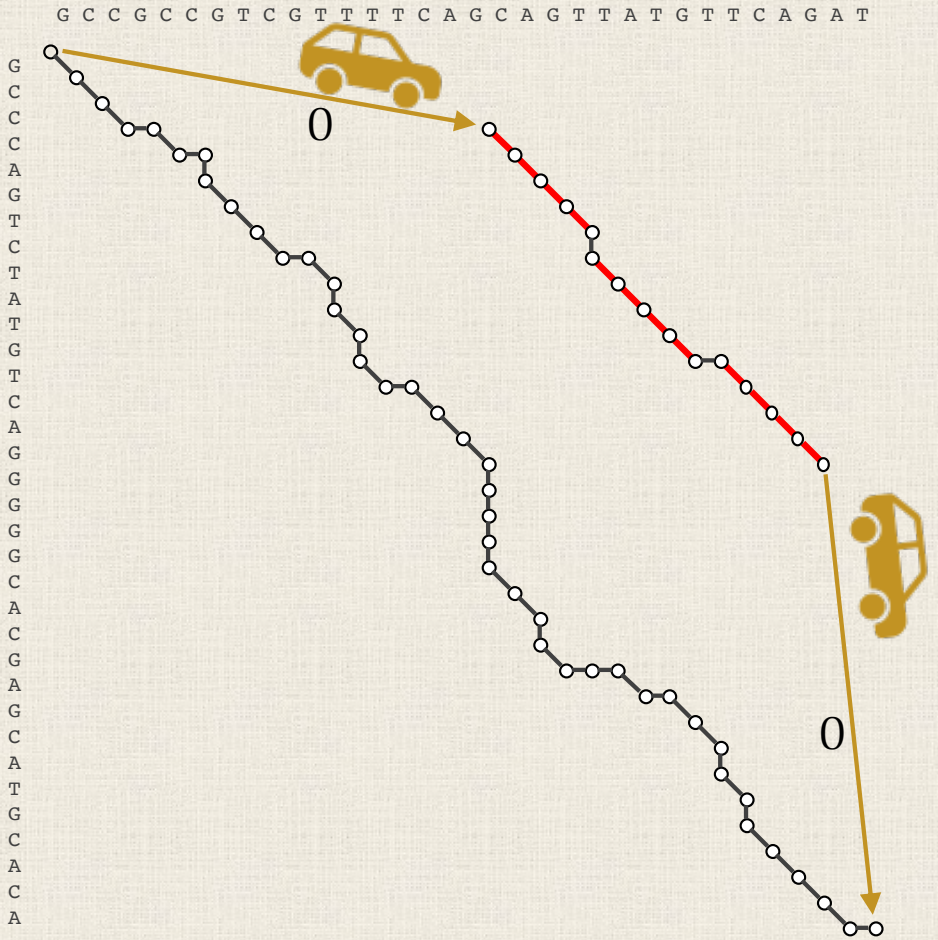


"Free Rides" for Local Alignment

Answer: It is given by

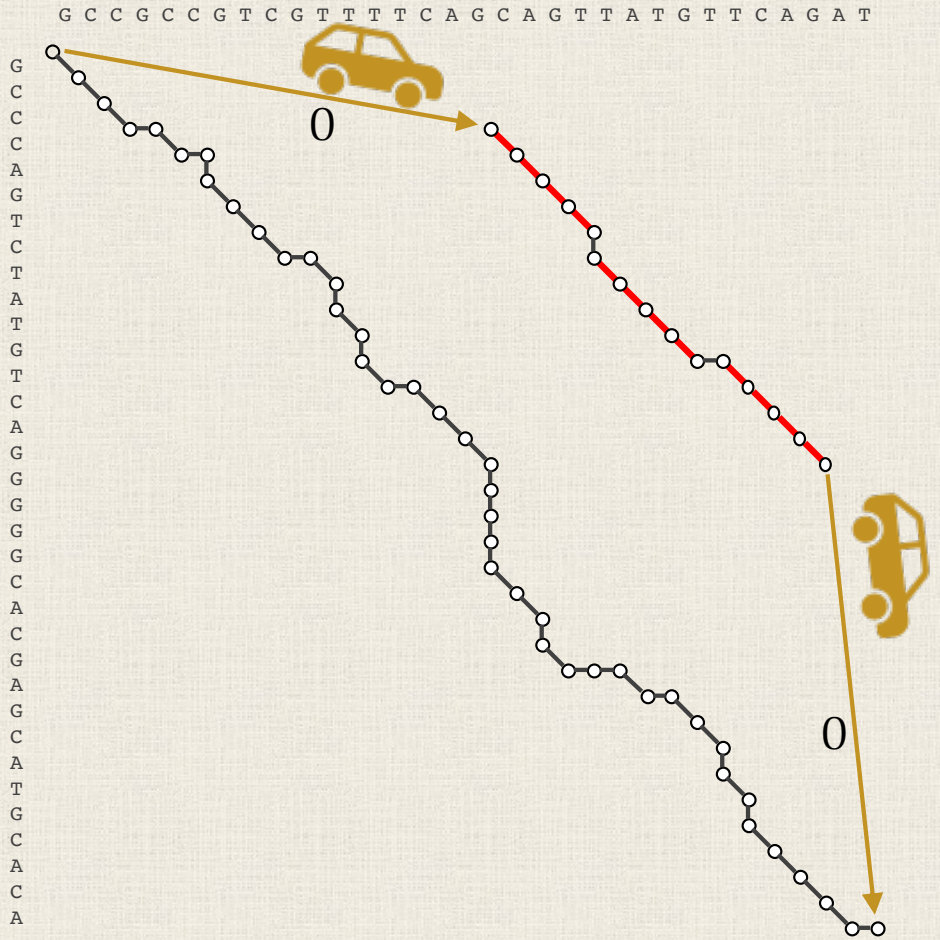
$$s_{i,j} = \max \begin{cases} 0 & \\ s_{i-1,j} & + Score(v_i, -) \\ s_{i,j-1} & + Score(-, w_j) \\ s_{i-1,j-1} & + Score(v_i, w_j) \end{cases}$$

where the scores here are $-\sigma$, $-\sigma$, and either $+1$ or $-\mu$ (depending on a match vs. a mismatch).



“Free Rides” for Local Alignment

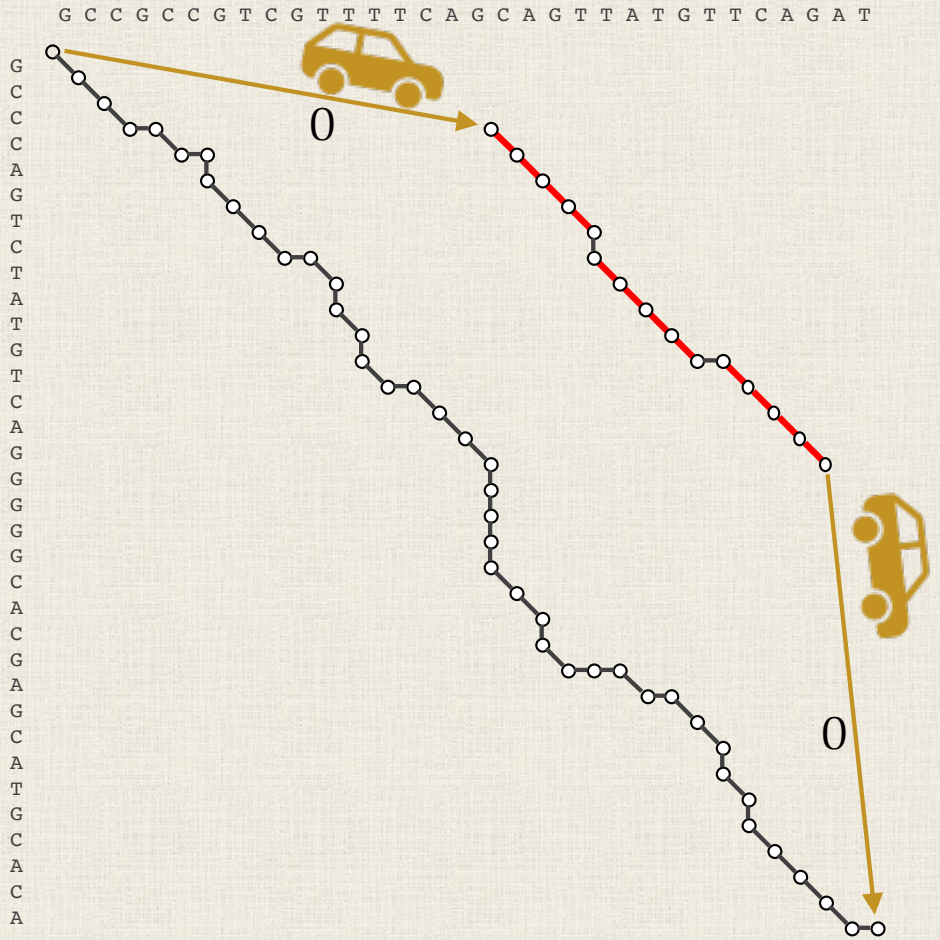
Exercise: After we apply the recurrence, where should we start backtracking? (That is, where does the best local alignment end?)



“Free Rides” for Local Alignment

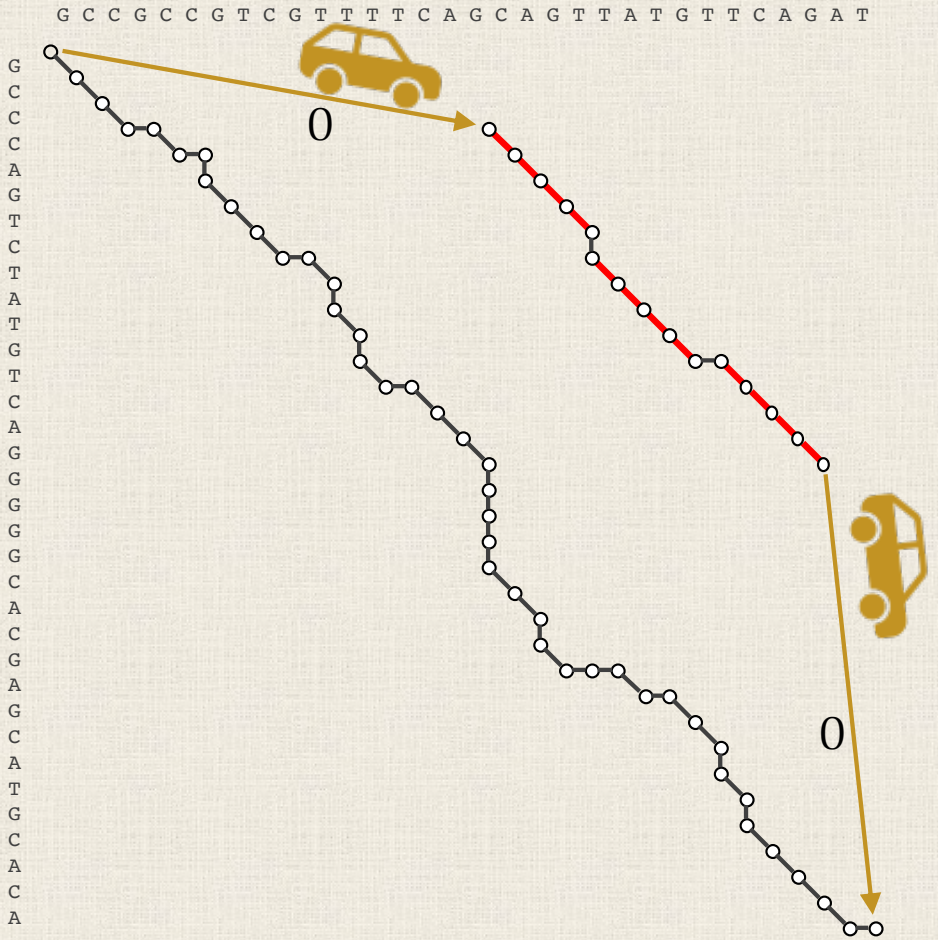
Exercise: After we apply the recurrence, where should we start backtracking? (That is, where does the best local alignment end?)

Answer: Wherever the *maximum* value of the scoring table is.



"Free Rides" for Local Alignment

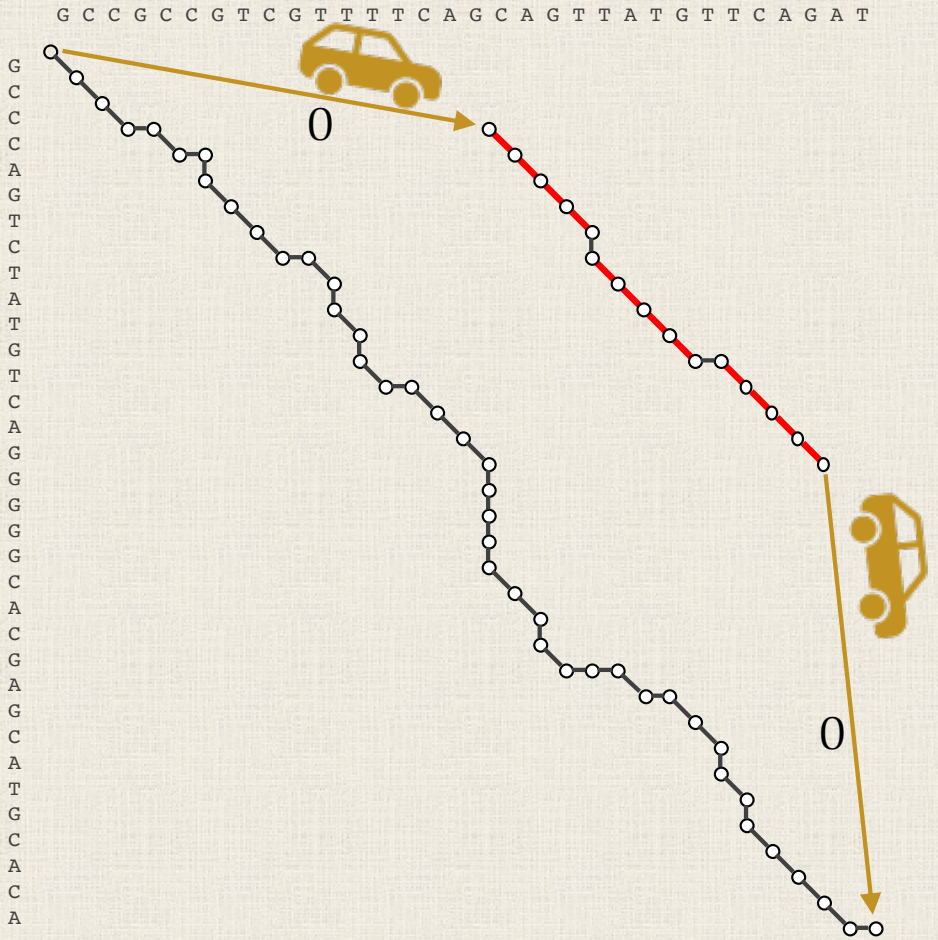
STOP: Recall that the dynamic programming algorithm has runtime proportional to the number of *edges* in the network. How many zero-weight edges did we add?



"Free Rides" for Local Alignment

STOP: Recall that the dynamic programming algorithm has runtime proportional to the number of *edges* in the network. How many zero-weight edges did we add?

Answer: Just ~2*nm*. ☺



The Solution to a Problem Unsolved for Ten Years Nearly Fits on One Slide

J. Mol. Biol. (1981), **147**, 195–197

Identification of Common Molecular Subsequences

The identification of maximally homologous subsequences among sets of long sequences is an important problem in molecular sequence analysis. The problem is straightforward only if one restricts consideration to contiguous subsequences (segments) containing no internal deletions or insertions. The more general problem has its solution in an extension of sequence metrics (Sellers 1974; Waterman *et al.*, 1976) developed to measure the minimum number of "events" required to convert one sequence into another.

These developments in the modern sequence analysis began with the heuristic homology algorithm of Needleman & Wunsch (1970) which first introduced an iterative matrix method of calculation. Numerous other heuristic algorithms have been suggested including those of Fitch (1966) and Dayhoff (1969). More mathematically rigorous algorithms were suggested by Sankoff (1972), Reichert *et al.* (1973) and Beyer *et al.* (1979), but these were generally not biologically satisfying or interpretable. Success came with Sellers (1974) development of a true metric measure of the distance between sequences. This metric was later generalized by Waterman *et al.* (1976) to include deletions/insertions of arbitrary length. This metric represents the minimum number of "mutational events" required to convert one sequence into another. It is of interest to note that Smith *et al.* (1980) have recently shown that under some conditions the generalized Sellers metric is equivalent to the original homology algorithm of Needleman & Wunsch (1970).

In this letter we extend the above ideas to find a pair of segments, one from each of two long sequences, such that there is no other pair of segments with greater similarity (homology). The similarity measure used here allows for arbitrary length deletions and insertions.

Algorithm

The two molecular sequences will be $\underset{\sim}{A}=a_1a_2\ldots a_n$ and $\underset{\sim}{B}=b_1b_2\ldots b_m$. A similarity $s(a,b)$ is given between sequence elements a and b. Deletions of length k are given weight W_k. To find pairs of segments with high degrees of similarity, we set up a matrix H. First set

$$H_{k0}=H_{0l}=0 \text{ for } 0\leq k\leq n \text{ and } 0\leq l\leq m.$$

Preliminary values of H have the interpretation that H_{ij} is the maximum similarity of two segments *ending* in a_i and b_j, respectively. These values are obtained from the relationship

$$H_{ij}=\max\{H_{i-1,j-1}+s(a_i,b_j), \max_{k\geq 1}\{H_{i-k,j}-W_k\}, \max_{l\geq 1}\{H_{i,j-l}-W_l\}, 0\}, \quad (1)$$

$1\leq i\leq n$ and $1\leq j\leq m$.

0022–2836/80/090195–03 $02.00/0

The formula for H_{ij} follows by considering the possibilities for ending the segments at any a_i and b_j.

(1) If a_i and b_j are associated, the similarity is

$$H_{i-1,j-1}+s(a_i,b_j).$$

(2) If a_i is at the end of a deletion of length k, the similarity is

$$H_{i-k,j}-W_k.$$

(3) If b_j is at the end of a deletion of length l, the similarity is

$$H_{i-k,j}-W_l.$$

(4) Finally, a zero is included to prevent calculated negative similarity, indicating no similarity up to a_i and b_j.†

The pair of segments with maximum similarity is found by first locating the maximum element of H. The other matrix elements leading to this maximum value are than sequentially determined with a traceback procedure ending with an element of H equal to zero. This procedure identifies the segments as well as produces the corresponding alignment. The pair of segments with the next best similarity is found by applying the traceback procedure to the second largest element of H not associated with the first traceback.

A simple example is given in Figure 1. In this example the parameters $s(a_ib_j)$ and W_k required were chosen on an *a priori* statistical basis. A match, $a_i=b_j$, produced an $s(a_ib_j)$ value of unity while a mismatch produced a minus one-third. These values have an average for long, random sequences over an equally probable four letter set of zero. The deletion weight must be chosen to be at least equal to the difference between a match and a mismatch. The value used here was $W_k=1{\cdot}0+1/3{*}k$.

	Δ	C	A	G	C	C	U	C	G	C	U	U	A	G
Δ	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0
A	0·0	0·0	1·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	1·0	0·0
A	0·0	0·0	1·0	0·7	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	1·0	0·7
U	0·0	0·0	0·0	0·7	0·3	0·0	1·0	0·0	0·0	0·0	1·0	1·0	0·0	0·7
G	0·0	0·0	0·0	1·0	0·3	0·0	0·0	0·7	1·0	0·0	0·0	0·7	0·7	1·0
C	0·0	1·0	0·0	0·0	2·0	1·3	0·3	1·0	0·3	2·0	0·7	0·3	0·3	0·3
C	0·0	1·0	0·7	0·0	1·0	3·0	1·7	1·3	1·0	1·3	1·7	0·3	0·0	0·0
A	0·0	0·0	2·0	0·7	0·3	1·7	2·7	1·3	1·0	0·7	1·0	1·3	1·3	0·0
U	0·0	0·0	0·7	1·7	0·3	1·3	2·7	2·3	1·0	0·7	1·7	2·0	1·0	1·0
U	0·0	0·0	0·3	0·3	1·3	1·0	2·3	2·3	2·0	0·7	1·7	2·7	1·7	1·0
G	0·0	0·0	0·0	1·3	0·0	1·0	1·0	2·0	3·3	2·0	1·7	1·3	2·3	2·7
A	0·0	0·0	1·0	0·0	1·0	0·3	0·7	0·7	2·0	3·0	1·7	1·3	2·3	2·0
C	0·0	1·0	0·0	0·7	1·0	2·0	0·7	1·7	1·7	3·0	2·7	1·3	1·0	2·0
G	0·0	0·0	0·7	1·0	0·3	0·7	1·7	0·3	2·7	1·7	2·7	2·3	1·0	2·0
G	0·0	0·0	0·0	1·7	0·7	0·3	0·3	1·3	1·3	2·3	1·3	2·3	2·0	2·0

FIG. 1. H_{ij} matrix generated from the application of eqn (1) to the sequences A-A-U-G-C-C-A-U-U-G-A-C-G-G and C-A-G-C-C-U-C-G-C-U-U-A-G. The underlined elements indicate the trackback path from the maximal element 3·30.

† Zero need not be included unless there are negative values of $s(a,b)$.

The Solution to a Problem Unsolved for Ten Years Nearly Fits on One Slide

Note, in this simple example, that the alignment obtained:

-G-C-C-A-U-U-G-
-G-C-C—U-C-G-

contains both a mismatch and an internal deletion. It is the identification of the latter which has not been previously possible in any rigorous manner.

This algorithm not only puts the search for pairs of maximally similar segments on a mathematically rigorous basis but it can be efficiently and simply programmed on a computer.

Northern Michigan University T. F. SMITH

Los Alamos Scientific Laboratory M. S. WATERMAN
P.O. Box 1663, Los Alamos
N. Mex. 87545, U.S.A.

Received 14 July 1980

REFERENCES

Beyer, W. A., Smith, T. F., Stein, M. L. & Ulam, S. M. (1979). *Math. Biosci.* **19**, 9–25.
Dayhoff, M. O. (1969). *Atlas of Protein Sequence and Structure*, National Biomedical Research Foundation, Silver Springs, Maryland.
Fitch, W. M. (1966). *J. Mol. Biol.* **16**, 9–13.
Needleman, S. B. & Wunsch, C. D. (1970). *J. Mol. Biol.* **48**, 443–453.
Reichert, T. A., Cohen, D. N. & Wong, A. K. C. (1973). *J. Theoret. Biol.* **42**, 245–261.
Sankoff, D. (1972). *Proc. Nat. Acad. Sci., U.S.A.* **61**, 4–6.
Sellers, P. H. (1974). *J. Appl. Math. (Siam)*, **26**, 787–793.
Smith, T. F., Waterman, M. S. & Fitch, W. M. (1981). *J. Mol. Evol.* In the press.
Waterman, M. S., Smith, T. F. & Beyer, W. A. (1976). *Advan. Math.* **20**, 367–387.

Note added in proof: A weighting similar to that given above was independently developed by Walter Goad of Los Alamos Scientific Laboratory.

Smith and Waterman's Scoring Table

	Δ	C	A	G	C	C	U	C	G	C	U	U	A	G
Δ	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0
A	0·0	0·0	1·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	1·0	0·0
A	0·0	0·0	1·0	0·7	0·0	0·0	0·0	0·0	0·0	0·0	0·0	0·0	1·0	0·7
U	0·0	0·0	0·0	0·7	0·3	0·0	1·0	0·0	0·0	0·0	1·0	1·0	0·0	0·7
G	0·0	0·0	0·0	1·0	0·3	0·0	0·0	0·7	1·0	0·0	0·0	0·7	0·7	1·0
C	0·0	1·0	0·0	0·0	2·0	1·3	0·3	1·0	0·3	2·0	0·7	0·3	0·3	0·3
C	0·0	1·0	0·7	0·0	1·0	3·0	1·7	1·3	1·0	1·3	1·7	0·3	0·0	0·0
A	0·0	0·0	2·0	0·7	0·3	1·7	2·7	1·3	1·0	0·7	1·0	1·3	1·3	0·0
U	0·0	0·0	0·7	1·7	0·3	1·3	2·7	2·3	1·0	0·7	1·7	2·0	1·0	1·0
U	0·0	0·0	0·3	0·3	1·3	1·0	2·3	2·3	2·0	0·7	1·7	2·7	1·7	1·0
G	0·0	0·0	0·0	1·3	0·0	1·0	1·0	2·0	3·3	2·0	1·7	1·3	2·3	2·7
A	0·0	0·0	1·0	0·0	1·0	0·3	0·7	0·7	2·0	3·0	1·7	1·3	2·3	2·0
C	0·0	1·0	0·0	0·7	1·0	2·0	0·7	1·7	1·7	3·0	2·7	1·3	1·0	2·0
G	0·0	0·0	0·7	1·0	0·3	0·7	1·7	0·3	2·7	1·7	2·7	2·3	1·0	2·0
G	0·0	0·0	0·0	1·7	0·7	0·3	0·3	1·3	1·3	2·3	1·3	2·3	2·0	2·0

FIG. 1. H_{ij} matrix generated from the application of eqn (1) to the sequences A-A-U-G-C-C-A-U-U-G-A-C-G-G and C-A-G-C-C-U-C-G-C-U-U-A-G. The underlined elements indicate the trackback path from the maximal element 3·30.

ONE MORE INNOVATION: AFFINE ALIGNMENT

Comparing Same-Score Alignments

```
GATCCAG
GA-C-AG
```

```
GATCCAG
GA--CAG
```

STOP: Which of these two alignments (which have the same score) is "better"? Why?

Comparing Same-Score Alignments

```
GATCCAG
GA-C-AG
```

```
GATCCAG
GA--CAG
```

Affine penalty: a way of scoring contiguous gaps higher than discontiguous gaps.

- **gap opening penalty (σ):** given to first symbol.
- **gap extension penalty (ε):** given to extra symbols.

Comparing Same-Score Alignments

```
GATCCAG
GA-C-AG
```

```
GATCCAG
GA--CAG
```

Affine penalty: a way of scoring contiguous gaps higher than discontiguous gaps.

- **gap opening penalty (σ):** given to first symbol.
- **gap extension penalty (ε):** given to extra symbols.

If $\sigma = 5$ and $\varepsilon = 1$, then the alignment on the left is penalized by $2\sigma = 10$, whereas the alignment on the right is only penalized by $\sigma + \varepsilon = 6$.

Adding Affine Gap Penalties

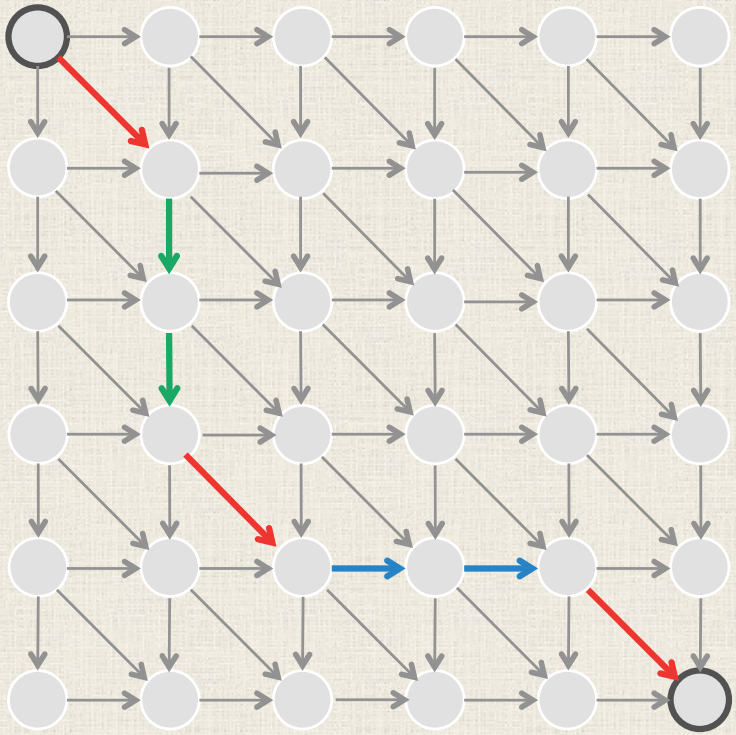
Alignment with Affine Gap Penalties Problem:

- **Input:** Two strings along with numbers σ and ε and a scoring matrix.
- **Output:** A highest scoring global alignment between these strings, as defined by the gap opening and extension penalties σ and ε.

STOP: How can we modify the alignment graph to solve this problem?

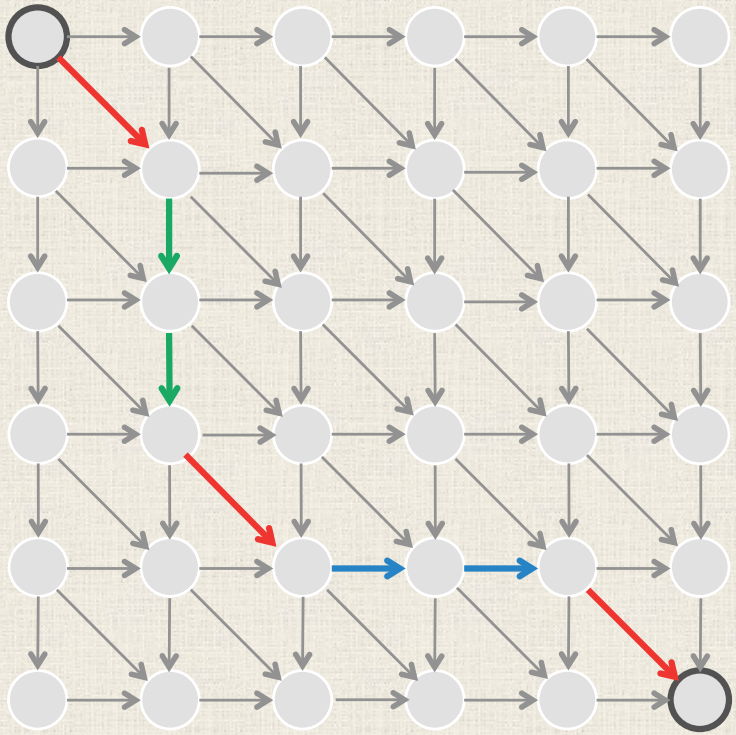
Adding "Long" Edges to Graph

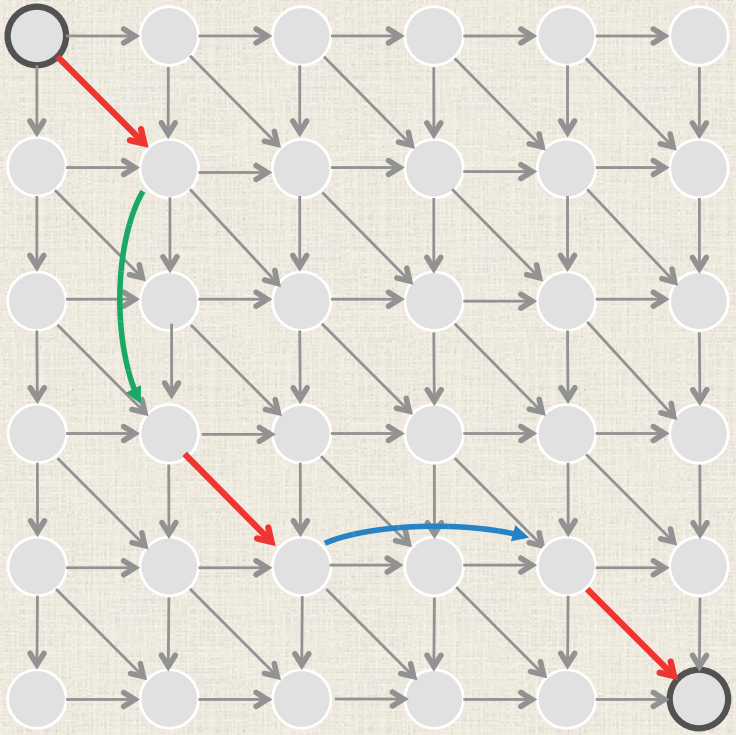
One solution: Add (huge number of) new edges to alignment graph to facilitate longer gaps.



Adding "Long" Edges to Graph

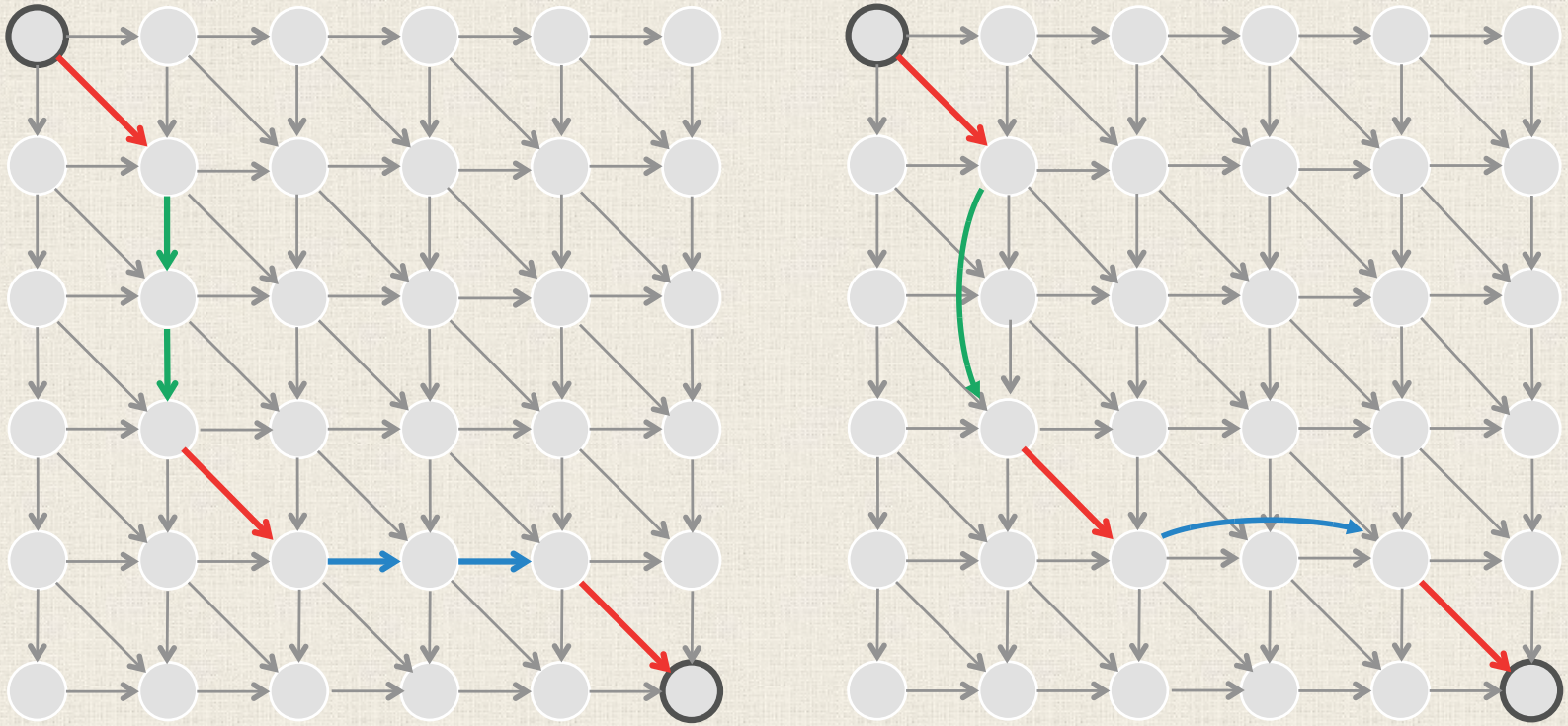
One solution: Add (huge number of) new edges to alignment graph to facilitate longer gaps.





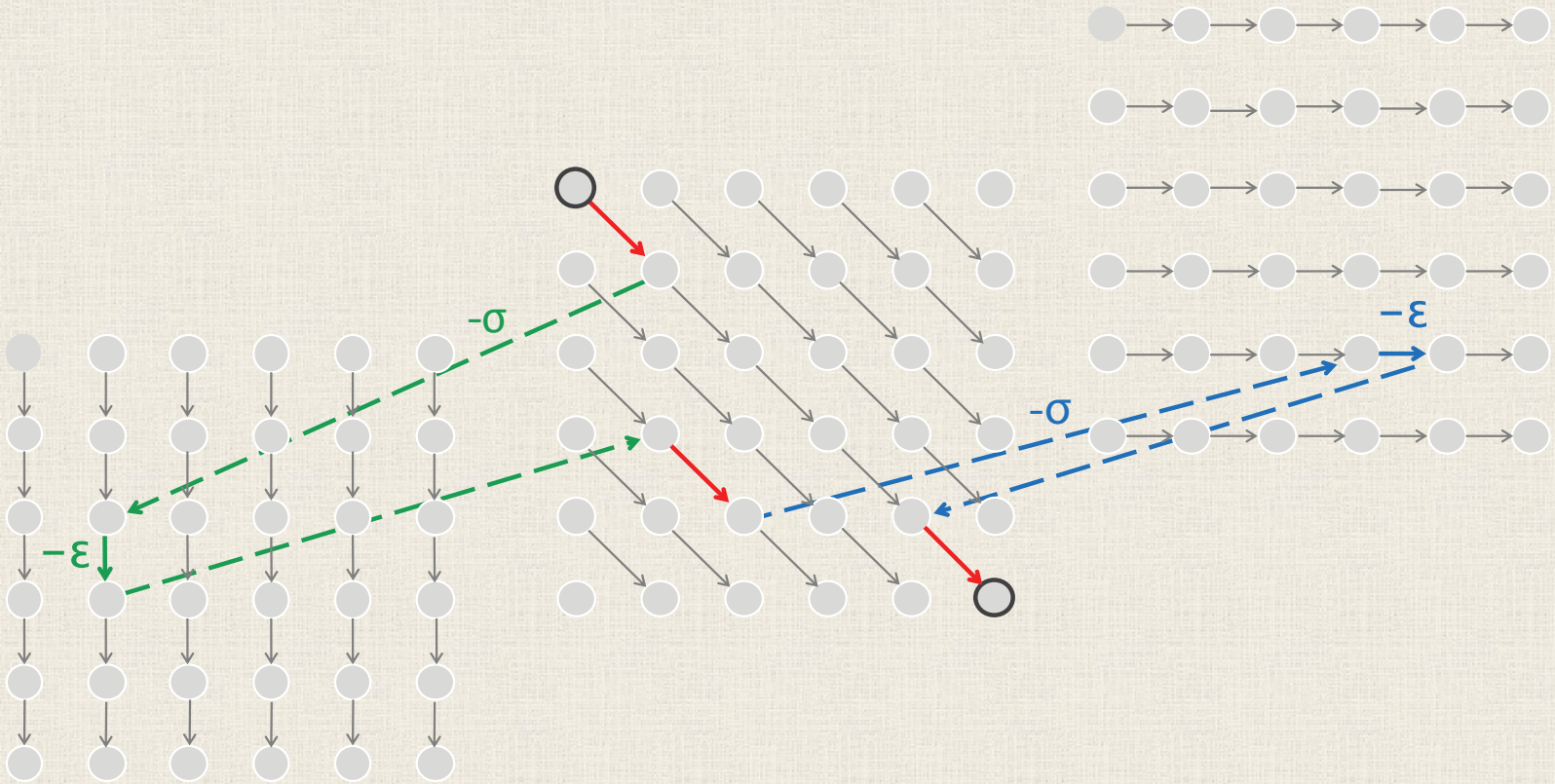
Adding "Long" Edges to Graph

The runtime of our algorithm is proportional to the number of edges, so maybe we can use *fewer* edges.



Three-Level Manhattan for Affine Alignment

This is the same path in a "three-level" Manhattan.

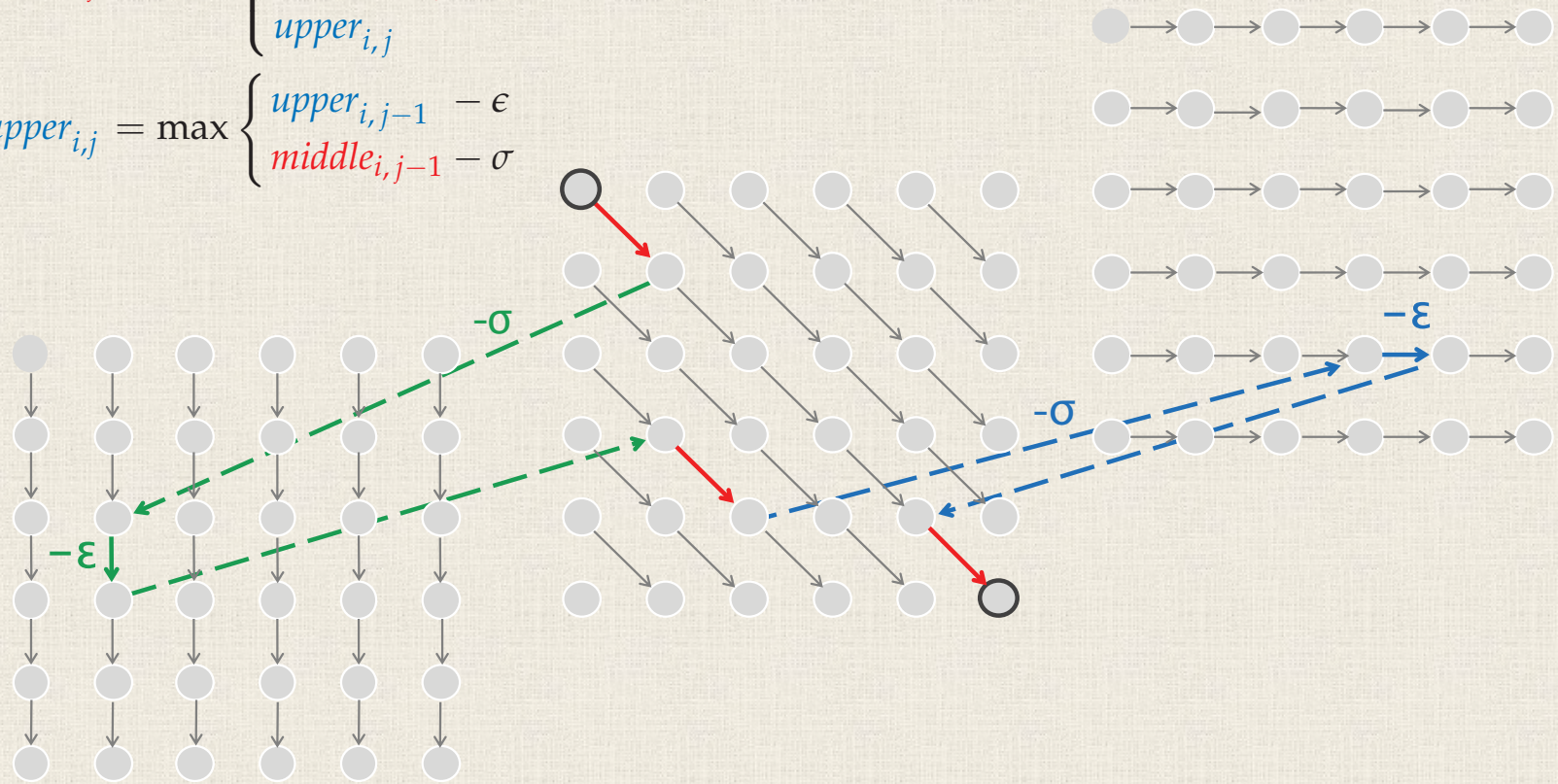


We Still are Finding a Longest Path and Have a Recurrence Relation

$$lower_{i,j} = \max \begin{cases} lower_{i-1,j} - \epsilon \\ middle_{i-1,j} - \sigma \end{cases}$$

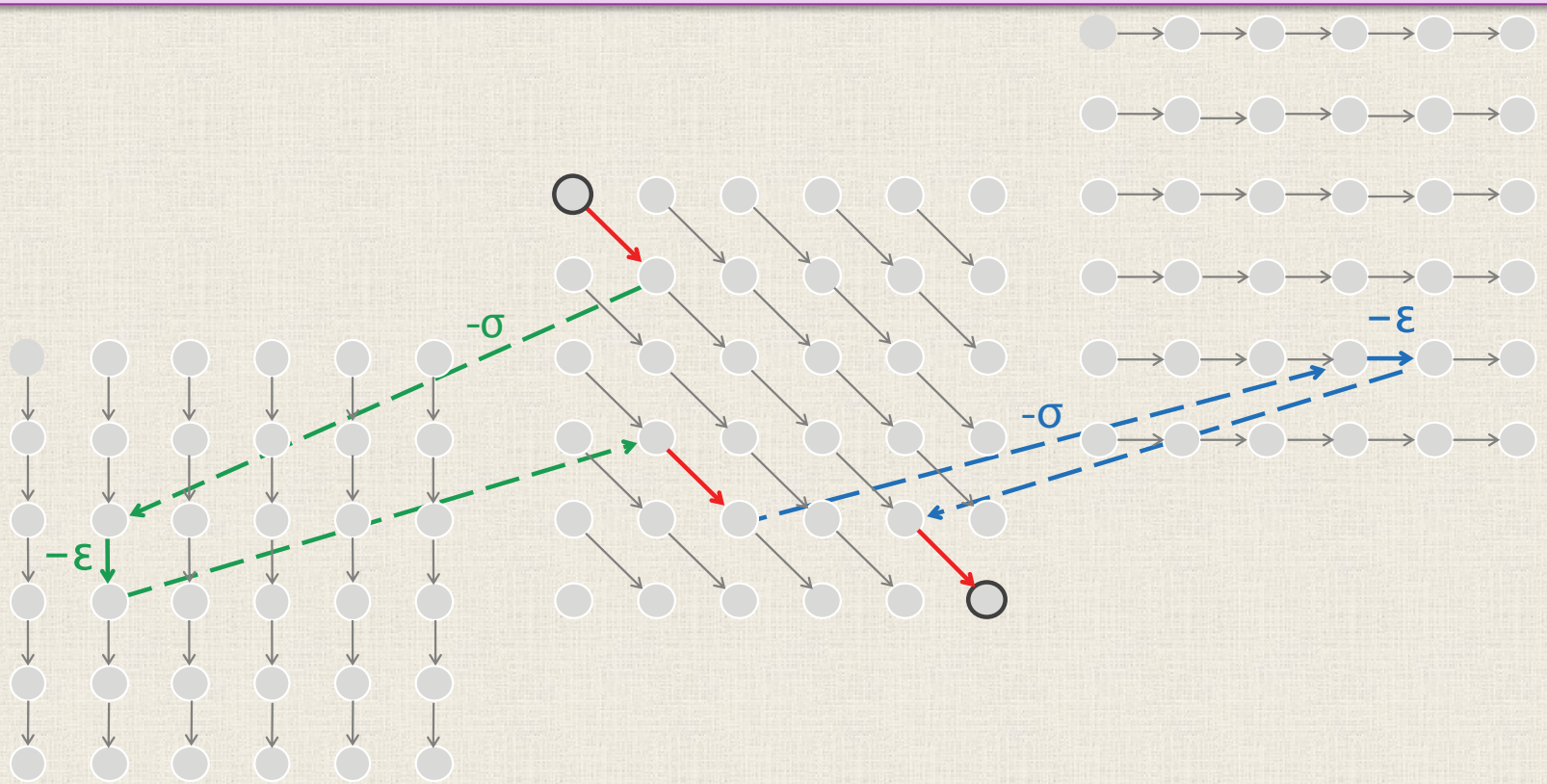
$$middle_{i,j} = \max \begin{cases} lower_{i,j} \\ middle_{i-1,j-1} + Score(v_i, w_j) \\ upper_{i,j} \end{cases}$$

$$upper_{i,j} = \max \begin{cases} upper_{i,j-1} - \epsilon \\ middle_{i,j-1} - \sigma \end{cases}$$



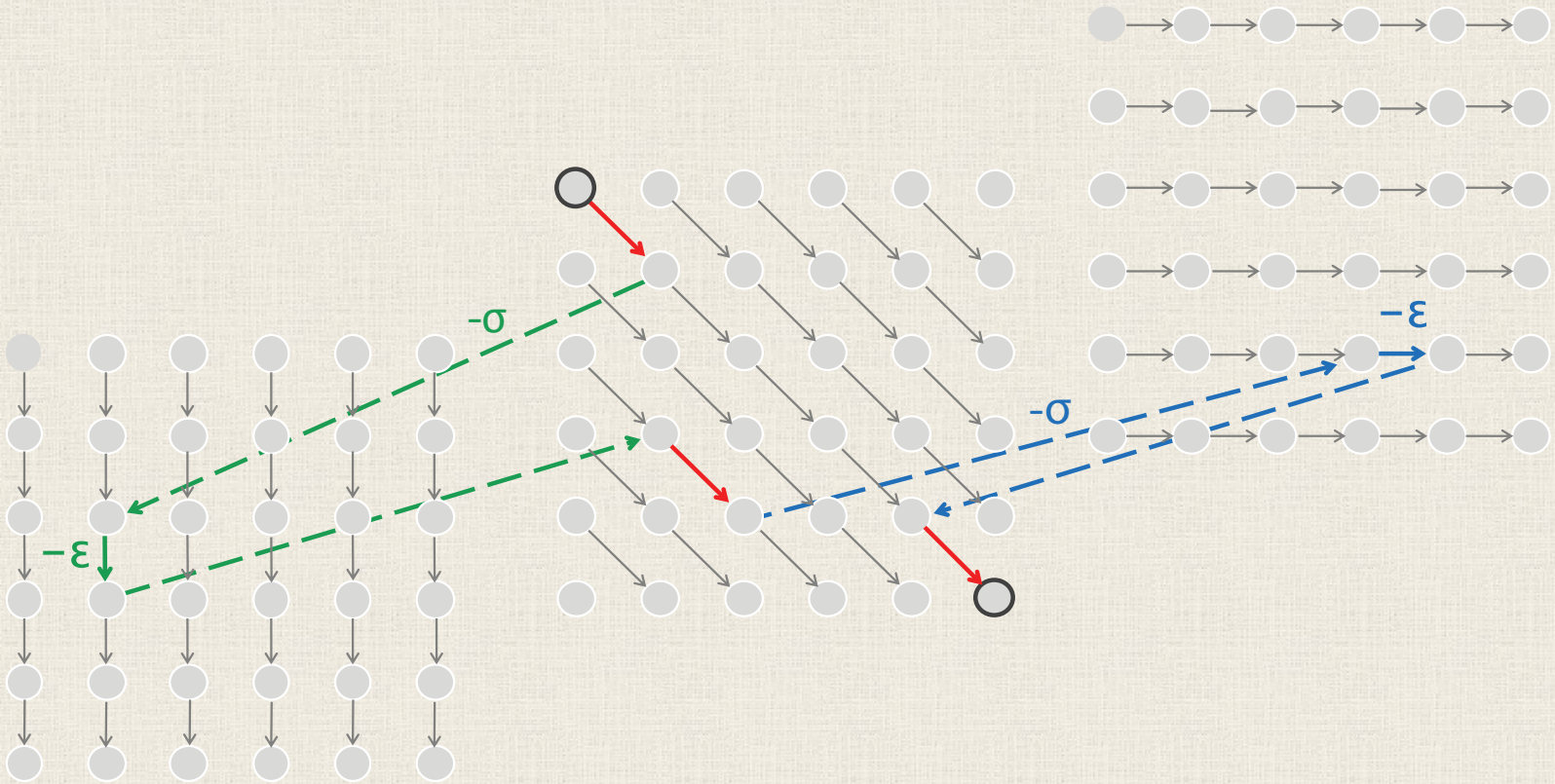
The Number of Edges is Still Manageable

Exercise: What is the approximate number of edges in this graph?



The Number of Edges is Still Manageable

Answer: Approximately (in fact, at most) $7 \cdot |v| \cdot |w|$.



ALIGNING MULTIPLE STRINGS

Moving to Multiple Sequences

Multiple Alignment Problem: *Find the highest-scoring alignment between multiple strings*.

- **Input:** A collection of *t* strings (and some way of scoring columns of a multiple alignment).
- **Output:** A multiple alignment of these strings having maximum score.

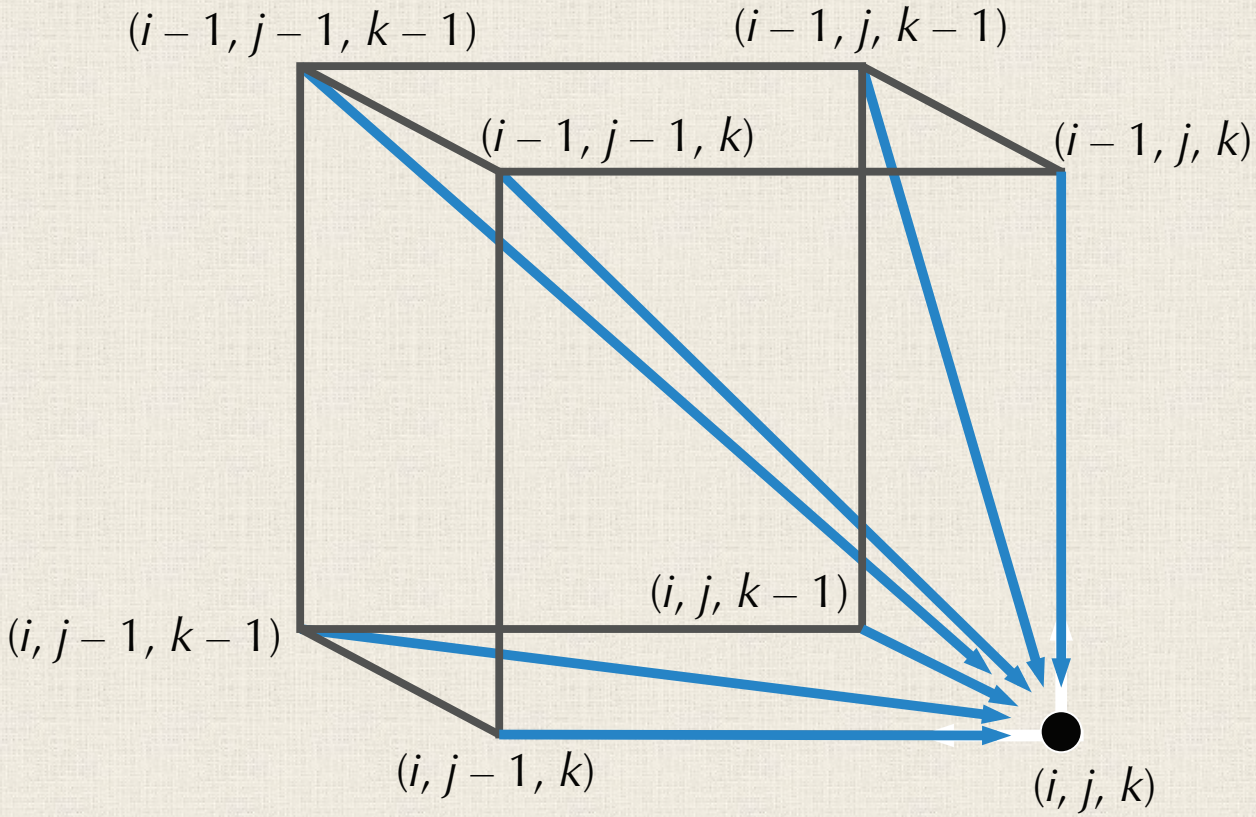
Moving to Multiple Sequences

Multiple Alignment Problem: *Find the highest-scoring alignment between multiple strings.*

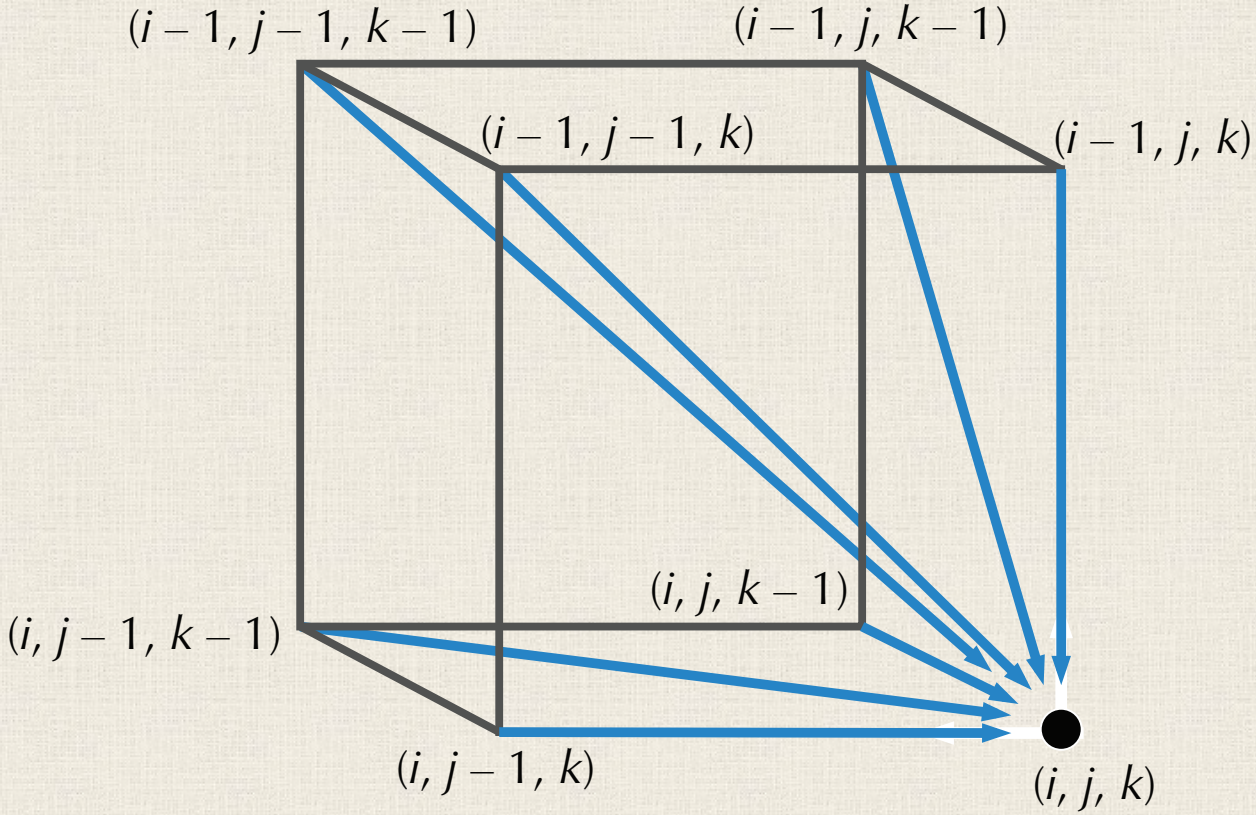
- **Input:** A collection of *t* strings (and some way of scoring columns of a multiple alignment).
- **Output:** A multiple alignment of these strings having maximum score.

STOP: What algorithm would you propose to solve this problem?

Moving to Multiple *Dimensions*

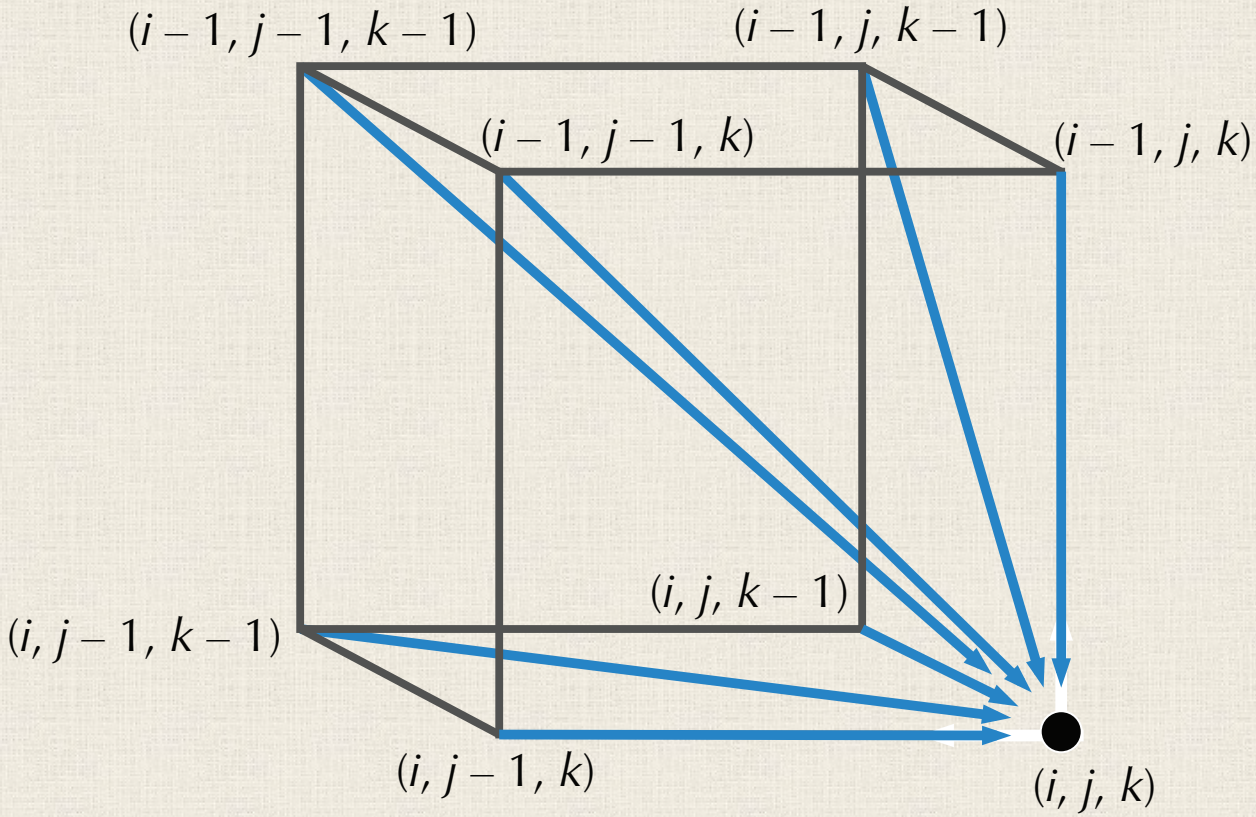


Moving to Multiple *Dimensions*



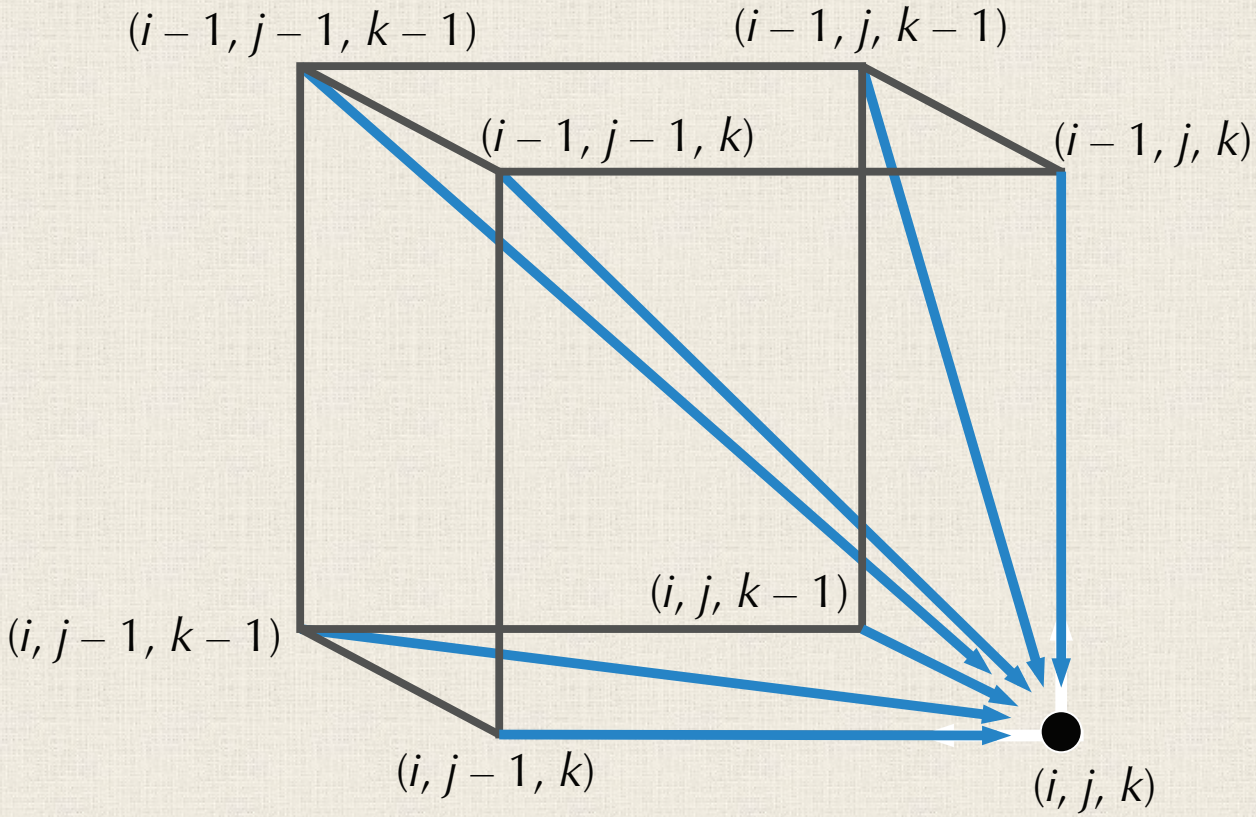
STOP: What is the issue with the dynamic programming approach in multiple dimensions?

Moving to Multiple *Dimensions*



Answer: The number of edges in a single block grows like $2^t - 1$...

Moving to Multiple *Dimensions*



STOP: What heuristic might you propose to align multiple sequences?

Greedy Heuristic for Multiple Alignment

1. Find an optimal pairwise alignment of each pair of strings.
2. Combine the set of optimal pairwise alignments into a multiple alignment.

Greedy Heuristic for Multiple Alignment

1. Find an optimal pairwise alignment of each pair of strings.
2. Combine the set of optimal pairwise alignments into a multiple alignment.

STOP: Try this approach on the strings `CCCCTTTT`, `TTTTGGGG`, and `GGGGCCCC`.

Greedy Heuristic for Multiple Alignment

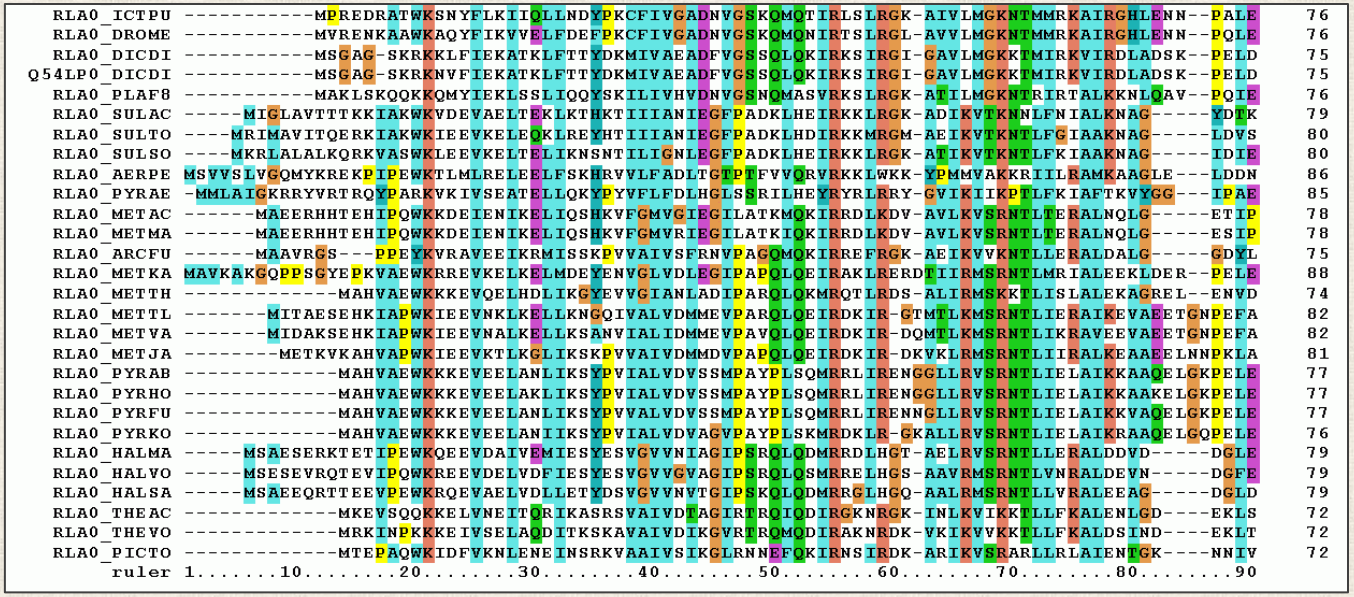
1. Find an optimal pairwise alignment of each pair of strings.
2. Combine the set of optimal pairwise alignments into a multiple alignment.

There is no way to combine these optimal pairwise alignment into a meaningful multiple alignment!

```
CCCCTTTT----        ----CCCCTTTT        TTTTGGGG----
----TTTTGGGG        GGGGCCCC----        ----GGGGCCCC
```

Pairwise Alignment Whispers, Multiple Alignment Shouts

Fortunately, strings that we are aligning will often be so similar that even simple heuristics will find correct alignments. *But not always*...



INTERLUDE: WHY DON'T WE HAVE AN HIV VACCINE?

Waiting for an HIV Vaccine …

Margaret Heckler
1984

Yet another terrible disease is about to yield to patience, persistence and outright genius.

Waiting for an HIV Vaccine …

Margaret Heckler
1984

Yet another terrible disease is about to yield to patience, persistence and outright genius.

It is no longer a question of *whether* we can develop an AIDS vaccine, it is simply a question of *when*.

Bill Clinton
1997

Waiting for an HIV Vaccine …

The failed HIV Merck vaccine study: a step back or a launching point for future vaccine development?

Rafick-Pierre Sekaly

‣ Author information ‣ Copyright and License information Disclaimer

See the article "An HIV-1 clade C DNA prime, NYVAC boost vaccine regimen induces reliable, polyfunctional, and long-lasting T cell responses" on page 63.

This article has been cited by other articles in PMC.

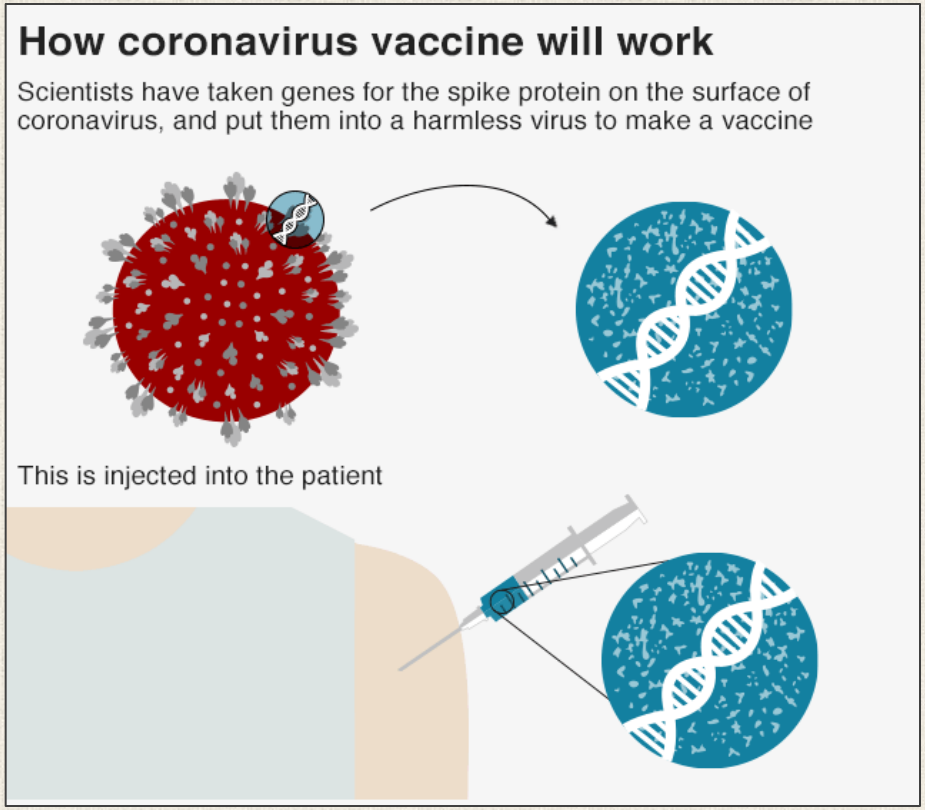
Abstract

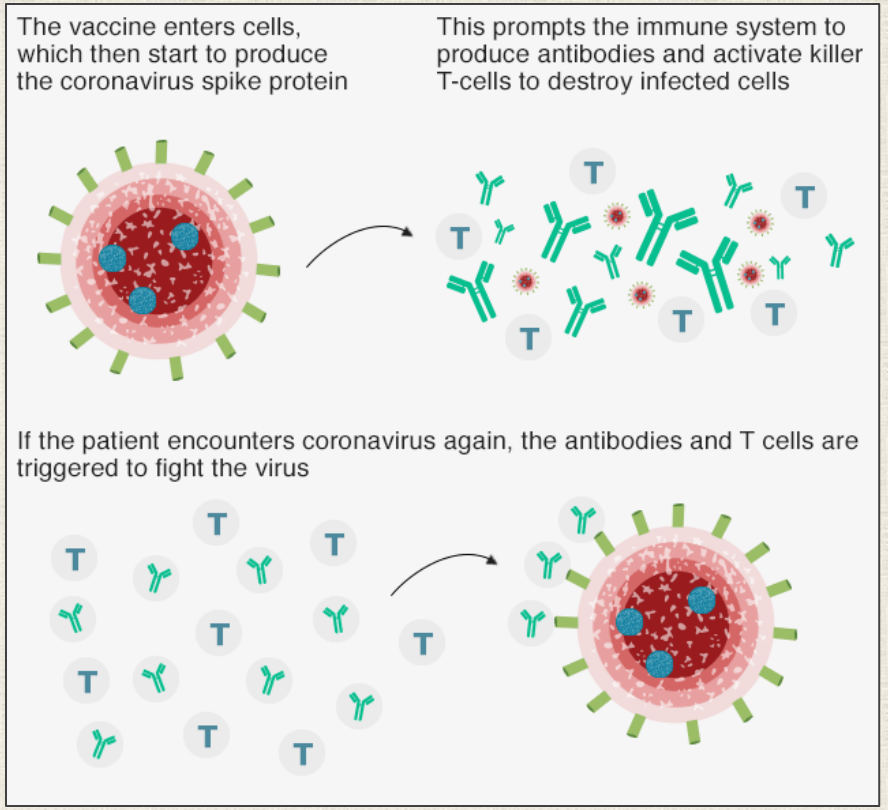
Go to:

The world of human immunodeficiency virus (HIV) vaccines has suffered a baffling setback. The first trial of a vaccine designed to elicit strong cellular immunity has shown no protection against infection. More alarmingly, the vaccine appeared to increase the rate of HIV infection in individuals with prior immunity against the adenovirus vector used in the vaccine. A new study in this issue suggests that a different vaccine approach—using a DNA prime/poxvirus boost strategy—induces polyfunctional immune responses to an HIV immunogen. The disappointing results of the recent vaccine trial suggest that a more thorough assessment of vaccine-induced immune responses is urgently needed, and that more emphasis should be placed on primate models before efficacy trials are undertaken.

… and yet we got a SARS-CoV-2 vaccine in under a year #ThanksPfizer

Many Vaccines Target Viral Surface Proteins

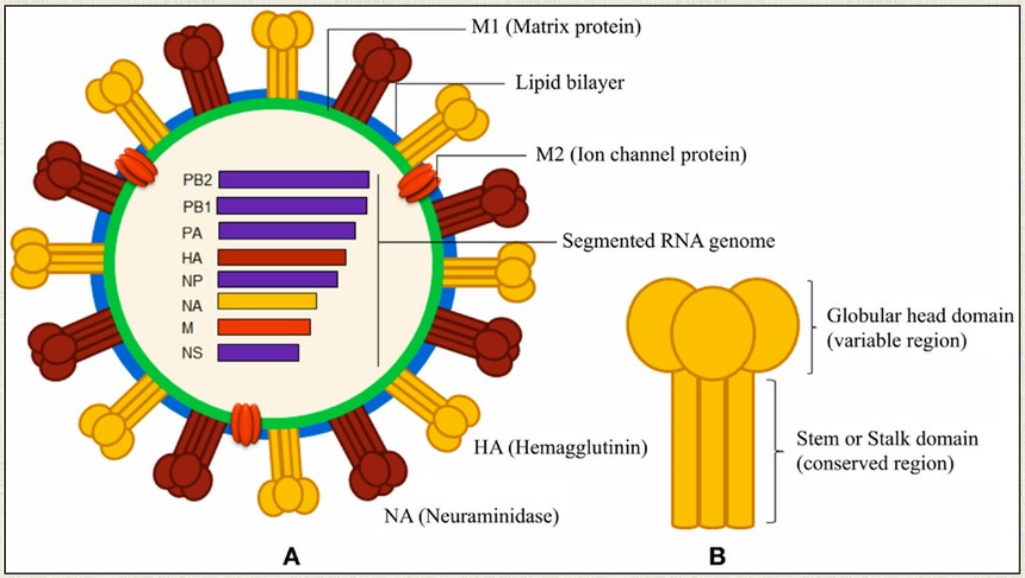




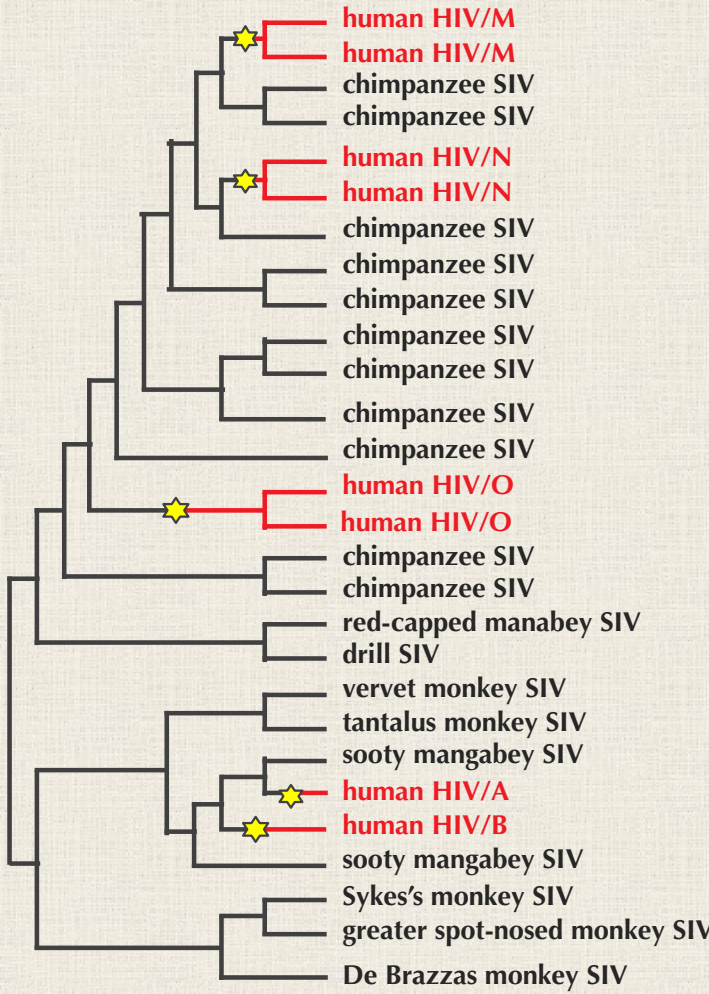
Source: https://www.bbc.com/news/health-52394485

Many Vaccines Target Viral Surface Proteins

Vaccines training the immune system to recognize HIV's surface proteins fail because HIV strains are so *variable*.



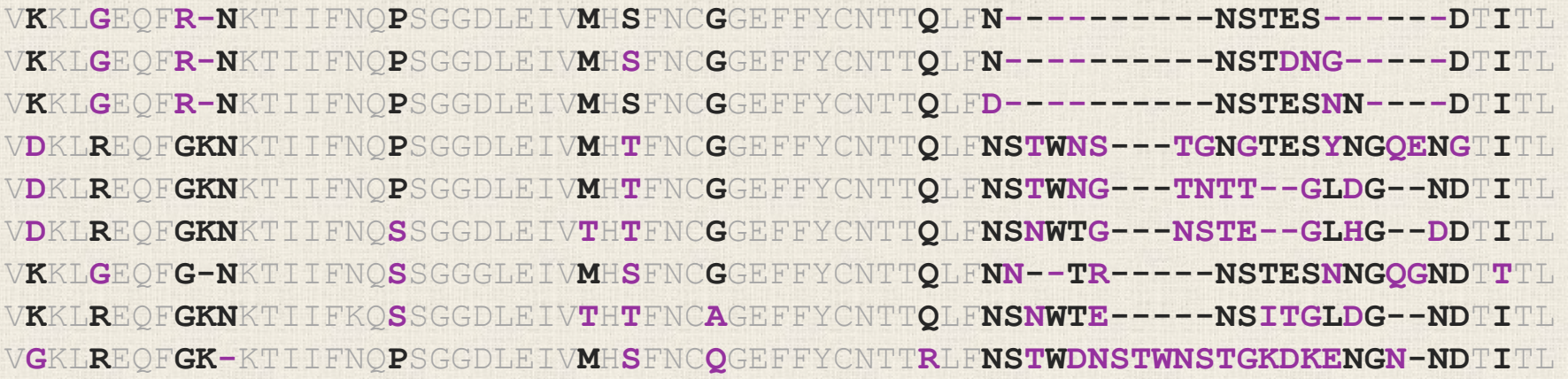
https://www.frontiersin.org/articles/10.3389/fimmu.2015.00336/full



HIV Drug "Cocktails" Have to Deal with Variability

The HIV population in a ***single*** infected individual rapidly evolves to evade the immune system.

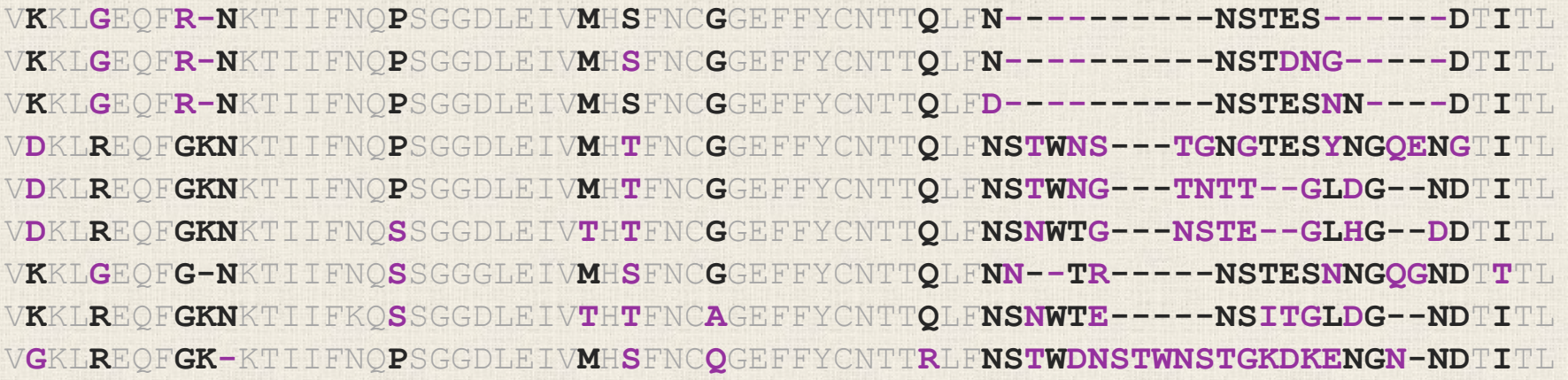
envelope glycoprotein gp120



HIV Drug "Cocktails" Have to Deal with Variability

The HIV population in a ***single*** infected individual rapidly evolves to evade the immune system.

envelope glycoprotein gp120



HIV strains from ***different*** patients are diverged phenotypes requiring different drug cocktails.

Returning to Multiple Alignment

Multiple Alignment Problem: *Find the highest-scoring alignment between multiple strings.*

- **Input:** A collection of *t* strings (and some way of scoring columns of a multiple alignment).
- **Output:** A multiple alignment of these strings having maximum score.

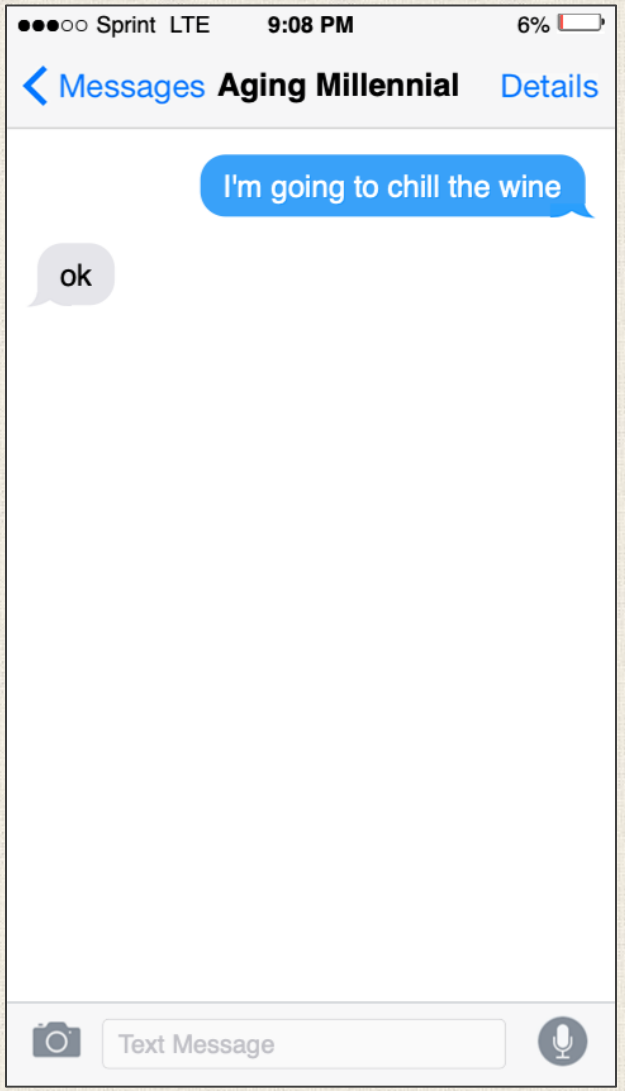
A single misalignment could lead to an error, so we have to be accurate. And so we need a *problem formulation* that scores different columns differently.

Another Problem

Once we have a collection of *known* protein alignments ("families"), we need to be able to identify which family a new protein belongs to. That is, add a new string into an existing alignment.

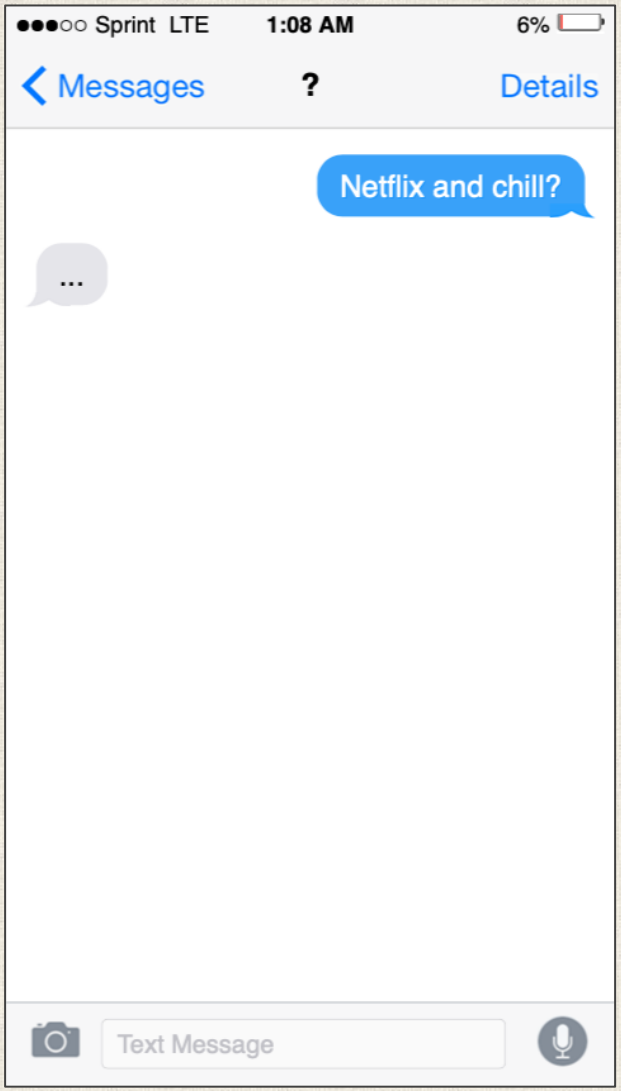
Trying to give you a deep understanding of alignment using an idiotic analogy

STOP: If we replace "chill" with "refrigerate", does it change the meaning of the sentence?



Trying to give you a deep understanding of alignment using an idiotic analogy

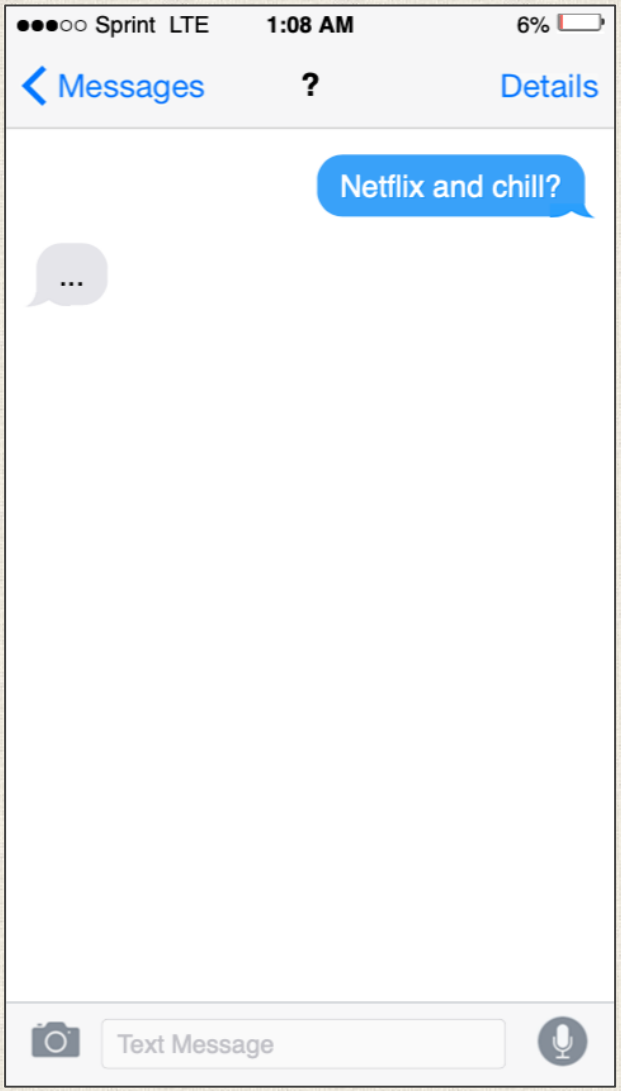
STOP: What about now? More importantly, what do you think I am getting at here?



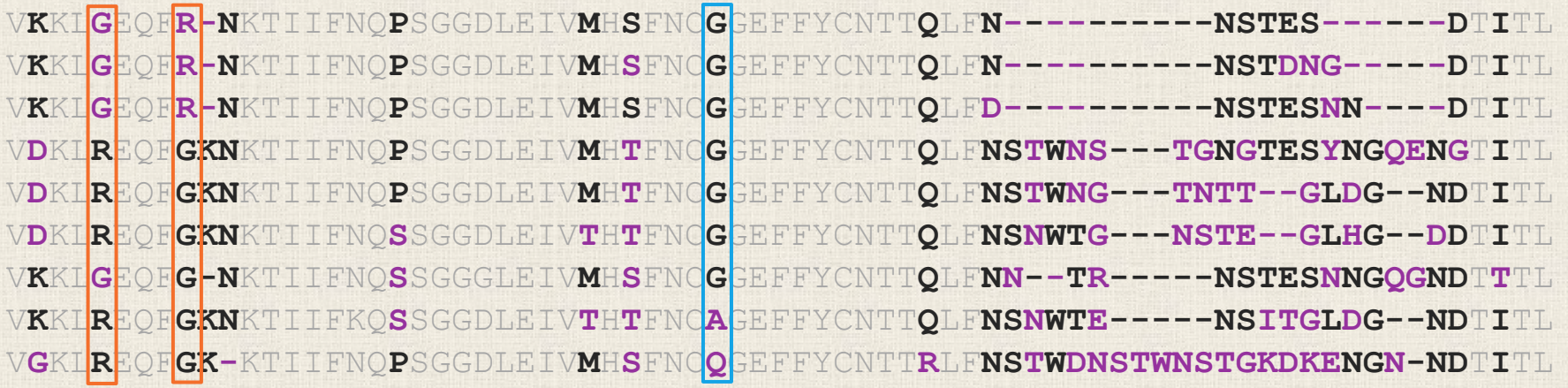
Trying to give you a deep understanding of alignment using an idiotic analogy

STOP: What about now? More importantly, what do you think I am getting at here?

Key point: Proteins have a "language", so why would we treat every replacement of two symbols the same?



I am making a good point, I promise



The cell is somehow OK with a G-R substitution in these two columns.

The cell is not OK with a G-R substitution in this column.

Can we introduce a model that has different scoring parameters in different columns?

GAMBLING WITH YAKUZA

Chō-Han and "Heads or Tails"

Chō-Han: A game played in 18th Century Japanese casinos in which players wager that the sum will be even ("chō") or odd ("han").

We will think about an equivalent game called "Heads or Tails" in which we bet on a coin toss.

Chō-Han and "Heads or Tails"

Chō-Han: A game played in 18th Century Japanese casinos in which players wager that the sum will be even ("chō") or odd ("han").

We will think about an equivalent game called "Heads or Tails" in which we bet on a coin toss.

	Sunday, Feb 02, 2020	
	Specials	
	Coin Toss of Super Bowl LIV, 2020	
05:58 PM	95251 Heads	-105
	95252 Tails	-105

Identifying a Biased Coin

A crooked dealer may use one of two coins:

- The fair coin (F) gives heads with probability ½:

  $\Pr_F(\text{“Head”}) = 1/2$ $\Pr_F(\text{“Tail”}) = 1/2$

- The biased coin (B) gives heads with probability ¾:

  $\Pr_B(\text{“Head”}) = 3/4$ $\Pr_B(\text{“Tail”}) = 1/4$

Identifying a Biased Coin

A crooked dealer may use one of two coins:

- The fair coin (F) gives heads with probability ½:

  $\Pr_F(\text{“Head”}) = 1/2$ $\Pr_F(\text{“Tail”}) = 1/2$

- The biased coin (B) gives heads with probability ¾:

  $\Pr_B(\text{“Head”}) = 3/4$ $\Pr_B(\text{“Tail”}) = 1/4$

STOP: Say that you play Heads or Tails 100 times, and the coin produces heads 63 times. Is the dealer cheating? Was the coin fair or biased?

Identifying a Biased Coin

A crooked dealer may use one of two coins:

- The fair coin (F) gives heads with probability ½:

  $\Pr_F(\text{“Head”}) = 1/2$ $\Pr_F(\text{“Tail”}) = 1/2$

- The biased coin (B) gives heads with probability ¾:

  $\Pr_B(\text{“Head”}) = 3/4$ $\Pr_B(\text{“Tail”}) = 1/4$

STOP: A better question would be, “Which coin is more likely to have been used if we see heads 63 times?”

Identifying a Biased Coin

A crooked dealer may use one of two coins:

- The fair coin (F) gives heads with probability ½:

  $\Pr_F(\text{“Head”}) = 1/2$ $\Pr_F(\text{“Tail”}) = 1/2$

- The biased coin (B) gives heads with probability ¾:

  $\Pr_B(\text{“Head”}) = 3/4$ $\Pr_B(\text{“Tail”}) = 1/4$

Answer: 63 is closer to 75 than 50, but there must be a more quantitative answer …

Identifying a Biased Coin

Given a sequence of n flips with k "Heads":

$$x = x_1\, x_2\, \ldots\, x_n$$

Identifying a Biased Coin

Given a sequence of n flips with k "Heads":

$$x = x_1\, x_2\, \ldots\, x_n$$

The probability this sequence was generated by the fair coin:

$$\Pr(x|F) = \Pr_F(x_1) \cdot \ldots \cdot \Pr_F(x_n) = (1/2)^n$$

Identifying a Biased Coin

Given a sequence of n flips with k "Heads":

$$x = x_1\, x_2\, \ldots\, x_n$$

The probability this sequence was generated by the fair coin:

$$\Pr(x|F) = \Pr_F(x_1) \cdot \ldots \cdot \Pr_F(x_n) = (1/2)^n$$

The probability that it was generated by the biased coin:

$$\Pr(x|B) = \Pr_B(x_1) \cdot \ldots \cdot \Pr_B(x_n) = (3/4)^k \cdot (1/4)^{n-k}$$

Identifying a Biased Coin

$\Pr(x|F) > \Pr(x|B) \rightarrow$ fair is more likely

$\Pr(x|F) < \Pr(x|B) \rightarrow$ biased is more likely

The probability this sequence was generated by the fair coin:

$$\Pr(x|F) = \Pr_F(x_1) \cdot \ldots \cdot \Pr_F(x_n) = (1/2)^n$$

The probability that it was generated by the biased coin:

$$\Pr(x|B) = \Pr_B(x_1) \cdot \ldots \cdot \Pr_B(x_n) = (3/4)^k \cdot (1/4)^{n-k}$$

Identifying a Biased Coin

Exercise: For a sequence of 100 flips with 63 heads, which coin is more likely?

The probability this sequence was generated by the fair coin:

$$\Pr(x|F) = \Pr_F(x_1) \cdot \ldots \cdot \Pr_F(x_n) = (1/2)^n$$

The probability that it was generated by the biased coin:

$$\Pr(x|B) = \Pr_B(x_1) \cdot \ldots \cdot \Pr_B(x_n) = (3/4)^k \cdot (1/4)^{n-k}$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

The probability this sequence was generated by the fair coin:

$$\Pr(x|F) = \Pr_F(x_1) \cdot \ldots \cdot \Pr_F(x_n) = (1/2)^n$$

The probability that it was generated by the biased coin:

$$\Pr(x|B) = \Pr_B(x_1) \cdot \ldots \cdot \Pr_B(x_n) = (3/4)^k \cdot (1/4)^{n-k}$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

$$2^n = 3^k$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

$$2^n = 3^k$$

$$n = k \cdot \log_2(3)$$

Identifying a Biased Coin

Both $(1/2)^{100}$ and $(3/4)^{63} \cdot (1/4)^{37}$ are so close to zero that this question is harder than it seems!

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

$$2^n = 3^k$$

$$n = k \cdot \log_2(3)$$

$$k = n / \log_2(3) \approx 0.632\ n$$

Identifying a Biased Coin

STOP: So … which coin was more likely?

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

$$2^n = 3^k$$

$$n = k \cdot \log_2(3)$$

$$k = n / \log_2(3) \approx 0.632\ n$$

Identifying a Biased Coin

Answer: The fair coin (!) because $k < 0.632\ n$.

Equilibrium occurs when

$$\Pr(x|F) = \Pr(x|B)$$

$$(1/2)^n = (3/4)^k \cdot (1/4)^{n-k}$$

$$(1/2)^n = 3^k/4^n$$

$$2^n = 3^k$$

$$n = k \cdot \log_2(3)$$

$$k = \mathrm{n} \,/\, \log_2(3) \approx 0.632\ n$$

Identifying a Biased Coin

Log-odds ratio: The logarithm of the ratio of $\Pr(x|F)$ and $\Pr(x|B)$:

$$\log_2(\Pr(x|F) \,/\, \Pr(x|B)) = \log_2(2^n/3^k) = n - k \cdot \log_2(3)$$

The log-odds ratio is positive when $\Pr(x|F) > \Pr(x|B)$ and negative when $\Pr(x|B) > \Pr(x|F)$.

Bakuto Dealers Were Shirtless for a Reason…

Now let's assume that the dealer has both a fair and biased coin and can switch back and forth.

Bakuto Dealers Were Shirtless for a Reason…

Now let's assume that the dealer has both a fair and biased coin and can switch back and forth.

Casino Problem: *Given a sequence of coin flips, determine when the dealer used a fair coin and a biased coin.*

- **Input:** A sequence $x = x_1\, x_2\, \ldots\, x_n$ of flips made by coins *F* and *B*.
- **Output:** A sequence $\pi = \pi_1\, \pi_2 \cdots \pi_n$, with each π_i being equal to either *F* or *B*.

Bakuto Dealers Were Shirtless for a Reason…

This is not a computational problem! Any of the 2^n sequences π can generate any x.

Casino Problem: *Given a sequence of coin flips, determine when the dealer used a fair coin and a biased coin.*

- **Input:** A sequence $x = x_1\, x_2\, \ldots\, x_n$ of flips made by coins *F* and *B*.
- **Output:** A sequence $\pi = \pi_1\, \pi_2 \cdots \pi_n$, with each π_i being equal to either *F* or *B*.

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

$\Pr(x|F) < \Pr(x|B)$

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

FFFFF

$\Pr(x|F) < \Pr(x|B)$

$\Pr(x|F) > \Pr(x|B)$

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

FFFFF

$\Pr(x|F)/\Pr(x|B) < 1$

$\Pr(x|F)/\Pr(x|B) > 1$

Grading π Using Log-Odds Ratio

HHHTHTHHHT

BBBBB $\Pr(x|F)/\Pr(x|B) < 1$

FFFFF $\Pr(x|F)/\Pr(x|B) > 1$

If n = # tosses and k = # heads, use log-odds ratio:

$$\log_2(\Pr(x|F) \,/\, \Pr(x|B)) = \log_2(2^n/3^k) = n - k \cdot \log_2(3)\ .$$

Grading π Using Log-Odds Ratio

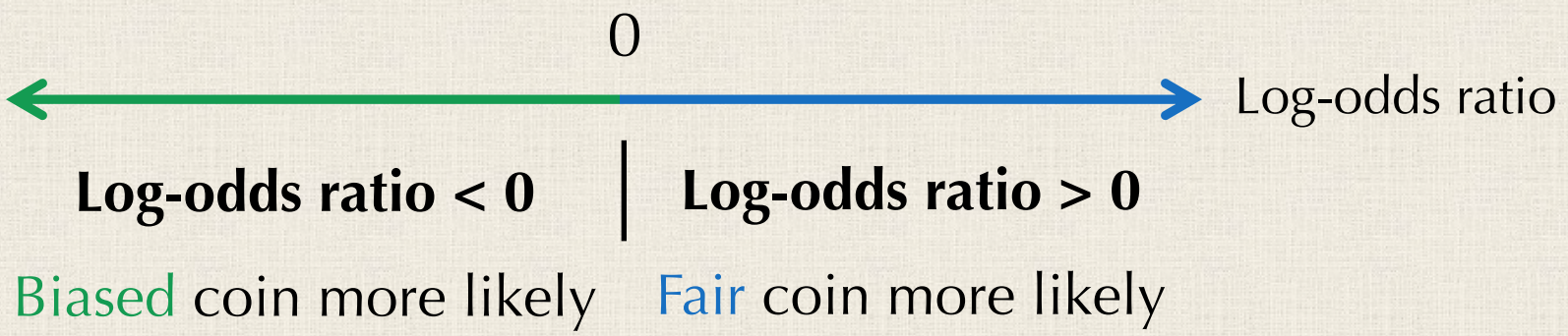
HHHT**H**T**HHH**T

BBBBB Log-odds < 0

FFFFF Log-odds > 0

If n = # tosses and k = # heads, use log-odds ratio:

$$\log_2(\Pr(x|F) \,/\, \Pr(x|B)) = \log_2(2^n/3^k) = n - k \cdot \log_2(3)\ .$$



Grading π Using Log-Odds Ratio

HHHT**H**T**H**H**H**T

BBBBB

FFFFF

FFFFF

Grading π Using Log-Odds Ratio

HHHT**HT**T**HH**T**H**T

BBBBB

FFFFF

FFFFF

FFFFF

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

FFFFF

FFFFF

FFFFF

BBBBB

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

FFFFF

FFFFF

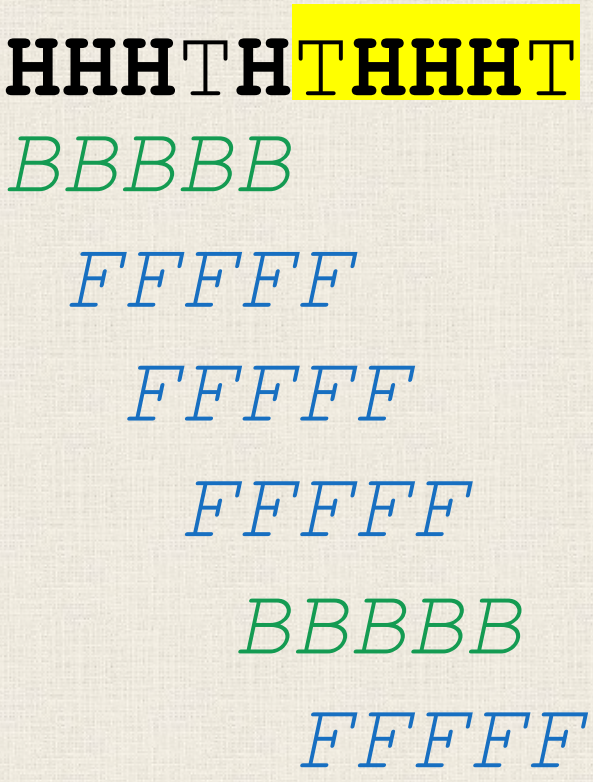
FFFFF

BBBBB

FFFFF

STOP: What are the disadvantages of this approach?

Grading π Using Log-Odds Ratio



Answer: Overlapping windows may make different prediction for the same flip.

Grading π Using Log-Odds Ratio

HHHT**H**T**HHH**T

BBBBB

FFFFF

FFFFF

FFFFF

BBBBB

FFFFF

(Also, there is no clear choice for window length.)

HIDDEN MARKOV MODELS

Turning the Dealer into a Machine

Think of the dealer as a machine with k **hidden states** (F and B) that proceeds in a sequence of steps.

Turning the Dealer into a Machine

Think of the dealer as a machine with k **hidden states** (F and B) that proceeds in a sequence of steps.

In each step, it emits a symbol (H or T) with certain probability based on its current state.

Turning the Dealer into a Machine

Think of the dealer as a machine with k **hidden states** (F and B) that proceeds in a sequence of steps.

In each step, it emits a symbol (H or T) with certain probability based on its current state.

While in a certain state, the machine makes two decisions:

1. Which *symbol* will I emit?
2. Which *hidden state* will I move to next?

Why are the States "Hidden"?

An observer can see the emitted symbols of an HMM but *does not* know which state the HMM is currently in.

Why are the States "Hidden"?

An observer can see the emitted symbols of an HMM but *does not* know which state the HMM is currently in.

Goal: infer the most likely sequence of hidden states of an HMM based on the sequence of emitted symbols.

Why are the States "Hidden"?

An observer can see the emitted symbols of an HMM but *does not* know which state the HMM is currently in.

Goal: infer the most likely sequence of hidden states of an HMM based on the sequence of emitted symbols.

If we also have a collection of probabilities for the likelihood of changing states, we have a **hidden Markov model (HMM)**.

An HMM Consists of Four Objects

Σ: an **alphabet** of emitted symbols H and T

An HMM Consists of Four Objects

Σ: an **alphabet** of emitted symbols — H and T

States : a set of **hidden states** — *F* and *B*

An HMM Consists of Four Objects

Σ: an **alphabet** of emitted symbols — H and T

States : a set of **hidden states** — *F* and *B*

Transition = ($transition_{l,k}$): a |*States*| ×|*States*| matrix of **transition probabilities** (of changing from state *l* to state *k*)

	F	*B*
F	0.9	0.1
B	0.1	0.9

An HMM Consists of Four Objects

Σ: an **alphabet** of emitted symbols — H and T

States : a set of **hidden states** — *F* and *B*

Transition = ($transition_{l,k}$): a $|States| \times |States|$ matrix of **transition probabilities** (of changing from state *l* to state *k*)

	F	*B*
F	0.9	0.1
B	0.1	0.9

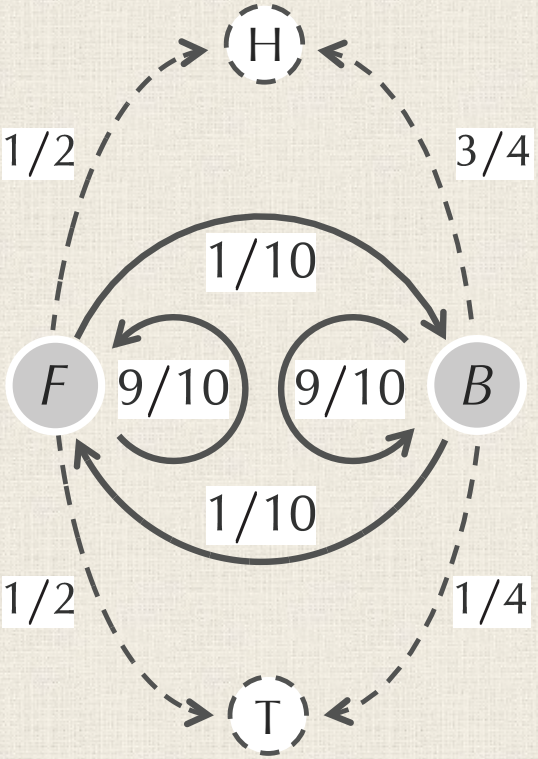
Emission= ($emission_k(b)$): a $|States| \times |\sum|$ matrix of **emission probabilities** (emitting symbol *b* when HMM is in state *k*)

	H	T
F	0.50	0.50
B	0.75	0.25

The HMM Diagram Visualizes an HMM

HMM Diagram:

- solid nodes are hidden states
- dashed nodes are emitted symbols
- solid directed edges: connect states and are labeled by transition probabilities
- dashed directed edges: connect state to symbol and labeled by emission probabilities.



Hidden Paths

Hidden path: a sequence $\pi = \pi_1 \ldots \pi_n$ of states that an HMM passes through.

Pr(x, π): the probability that an HMM follows the hidden path π *and* emits the string $x = x_1 x_2 \ldots x_n$.

x:	T	H	T	H	H	H	T	H	T	T	H
π:	*F*	*F*	*F*	*B*	*B*	*B*	*B*	*B*	*F*	*F*	*F*

Representing $\Pr(x, \pi)$ as a Product

HMM follows π *and* emits x when two events occur.

1. The HMM follows the path π. The probability of this event is $\Pr(\pi)$.
2. Given that HMM follows path π, it emits x. This is the *conditional* probability $\Pr(x|\pi)$.

Representing $\Pr(x, \pi)$ as a Product

HMM follows π *and* emits x when two events occur.

1. The HMM follows the path π. The probability of this event is $\Pr(\pi)$.
2. Given that HMM follows path π, it emits x. This is the *conditional* probability $\Pr(x|\pi)$.

This is a more general result in probability:

$$\Pr(x, \pi) = \Pr(\pi) \cdot \Pr(x|\pi).$$

Let's compute each of the terms on the right.

First: Computing $\Pr(\pi)$

$\Pr(\pi)$ is just the *product* of the probabilities $\Pr(\pi_i \rightarrow \pi_{i+1})$, where each $\Pr(\pi_i \rightarrow \pi_{i+1})$ is the probability of transitioning from state π_i to state π_{i+1}.

i	1	2	3	4	5	6	7	8	9	10	11
x	T	H	T	H	H	H	T	H	T	T	H
π	*F*	*F*	*F*	*B*	*B*	*B*	*B*	*B*	*F*	*F*	*F*
$\Pr(\pi_i \rightarrow \pi_{i+1})$	$\frac{1}{2}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$

First: Computing $\Pr(\pi)$

$\Pr(\pi)$ is just the *product* of the probabilities $\Pr(\pi_i \rightarrow \pi_{i+1})$, where each $\Pr(\pi_i \rightarrow \pi_{i+1})$ is the probability of transitioning from state π_i to state π_{i+1}.

Below: $\Pr(\pi_0 \rightarrow \pi_1)$ is ½ since we assume there is a 50-50 chance of starting in state π_1.

i	1	2	3	4	5	6	7	8	9	10	11
x	T	H	T	H	H	H	T	H	T	T	H
π	*F*	*F*	*F*	*B*	*B*	*B*	*B*	*B*	*F*	*F*	*F*
$\Pr(\pi_i \rightarrow \pi_{i+1})$	$\frac{1}{2}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$

Next: Computing $\Pr(x|\pi)$

If we know the hidden path, then the probability of emitting a string $x = x_1 \ldots x_n$ is just the product of the emission probabilities of each symbol x_i.

$$\Pr(x|\pi) = \prod_{i=1}^{n} \Pr(x_i|\pi_i) = \prod_{i=1}^{n} emission_{\pi_i}(x_i)$$

i	1	2	3	4	5	6	7	8	9	10	11
x	T	H	T	H	H	H	T	H	T	T	H
π	F	F	F	B	B	B	B	B	F	F	F
$\Pr(\pi_i \to \pi_{i+1})$	$\frac{1}{2}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$
$\Pr(x_i \mid \pi_i)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

Putting it All Together

Exercise: Compute $\Pr(x, \pi) = \Pr(\pi) \cdot \Pr(x|\pi)$ for the x and π below. Can you find a better explanation for x = "THTHHHTHTTH" than π = *FFFBBBBBFFF*?

$$\Pr(x|\pi) = \prod_{i=1}^{n} \Pr(x_i|\pi_i) = \prod_{i=1}^{n} emission_{\pi_i}(x_i)$$

i	1	2	3	4	5	6	7	8	9	10	11
x	T	H	T	H	H	H	T	H	T	T	H
π	*F*	*F*	*F*	*B*	*B*	*B*	*B*	*B*	*F*	*F*	*F*
$\Pr(\pi_i \to \pi_{i+1})$	$\frac{1}{2}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{9}{10}$	$\frac{1}{10}$	$\frac{9}{10}$	$\frac{9}{10}$
$\Pr(x_i \mid \pi_i)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

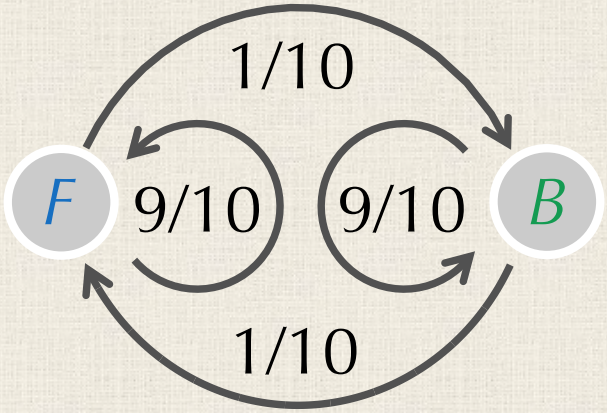
THE DECODING PROBLEM

Finding the Best Path for a String

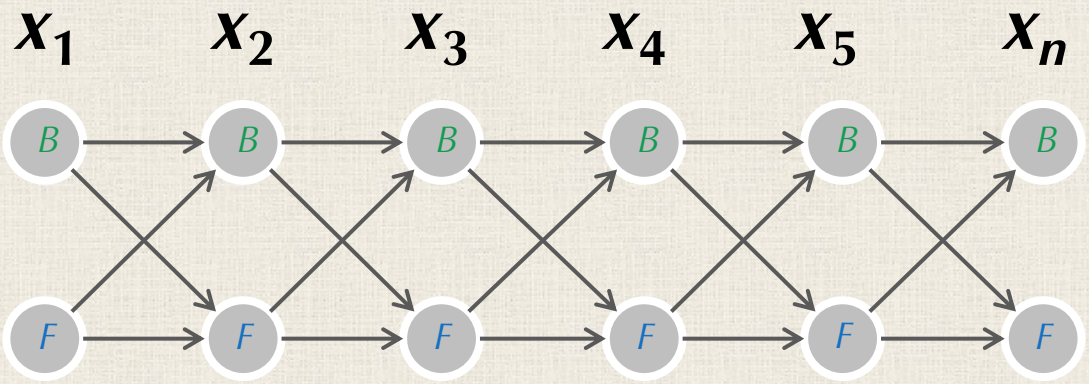
Decoding Problem: *Find an optimal hidden path in an HMM given its emitted string.*

- **Input:** A string $x = x_1 \ldots x_n$ emitted by an HMM ($\sum$, *States, Transition, Emission*).
- **Output:** A path π that maximizes the probability $\Pr(x,\pi)$ over all possible paths through this HMM.

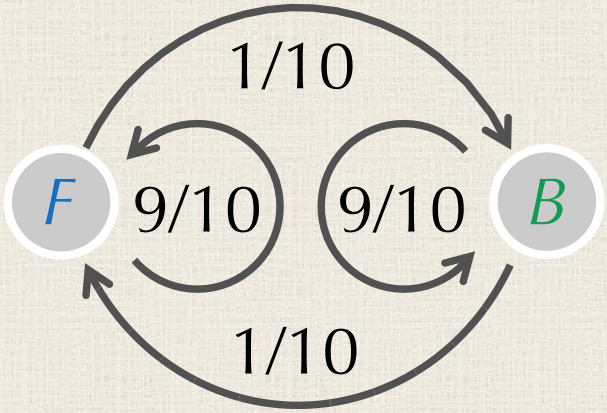
Building a DAG for the Crooked Casino



HMM diagram

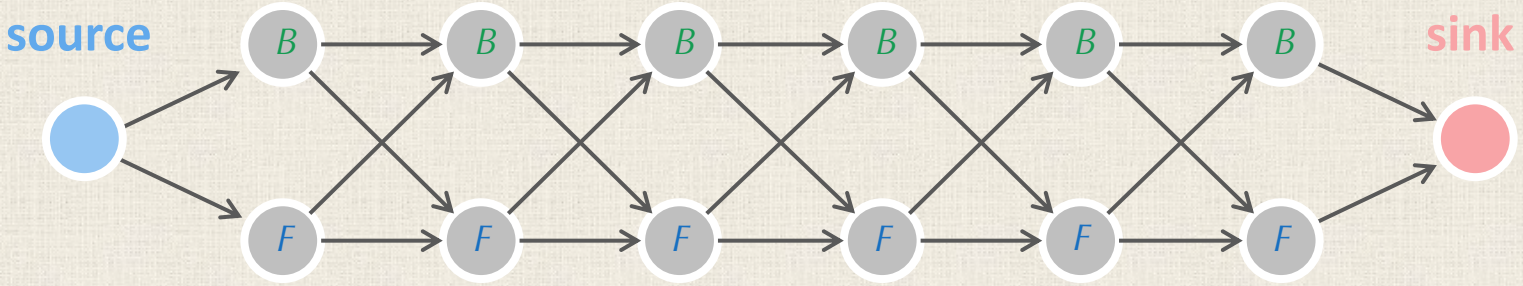


Building a DAG for the Crooked Casino

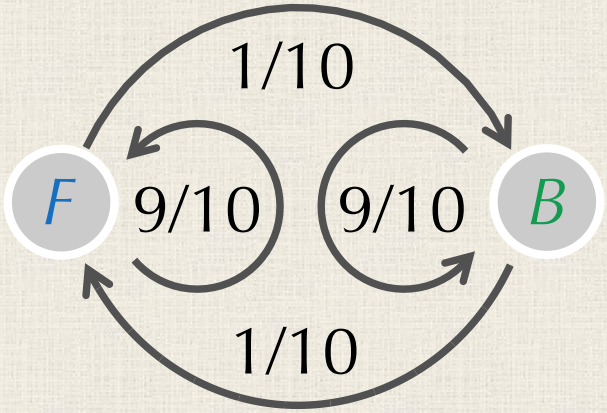


HMM diagram

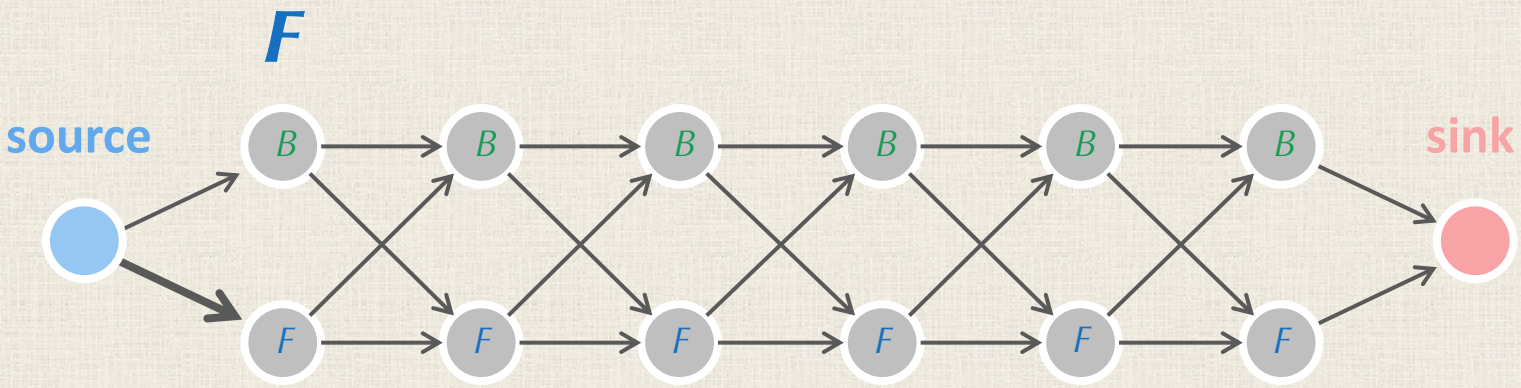
The source and sink are "**silent states**" (don't emit a symbol).



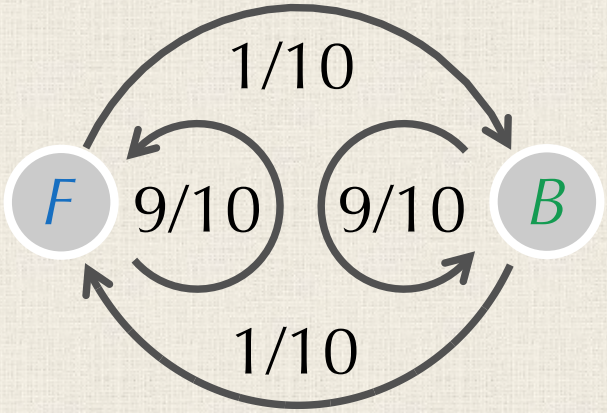
Building a DAG for the Crooked Casino



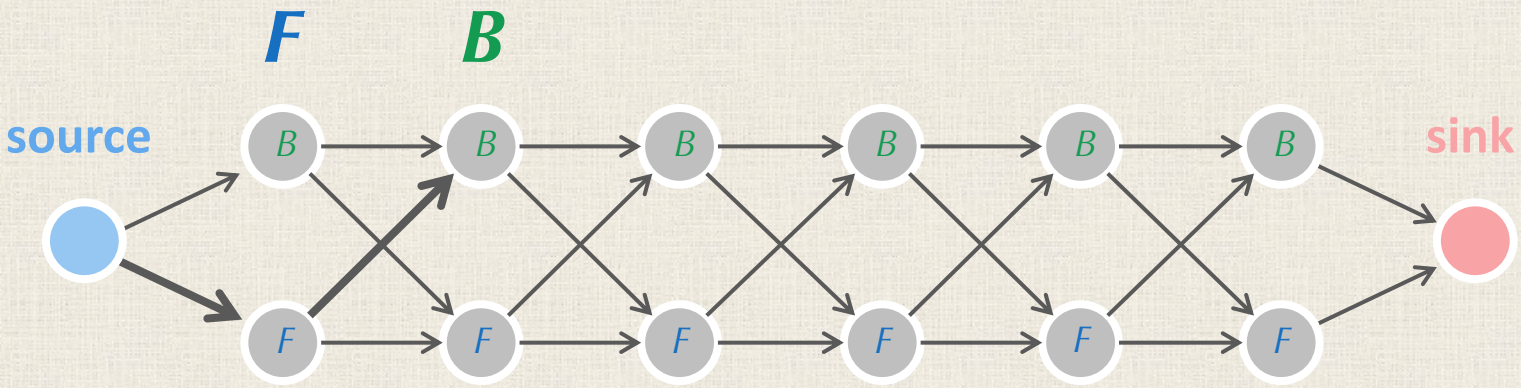
HMM diagram



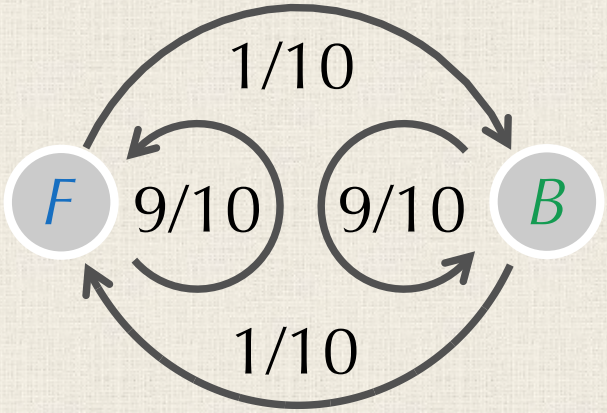
Building a DAG for the Crooked Casino



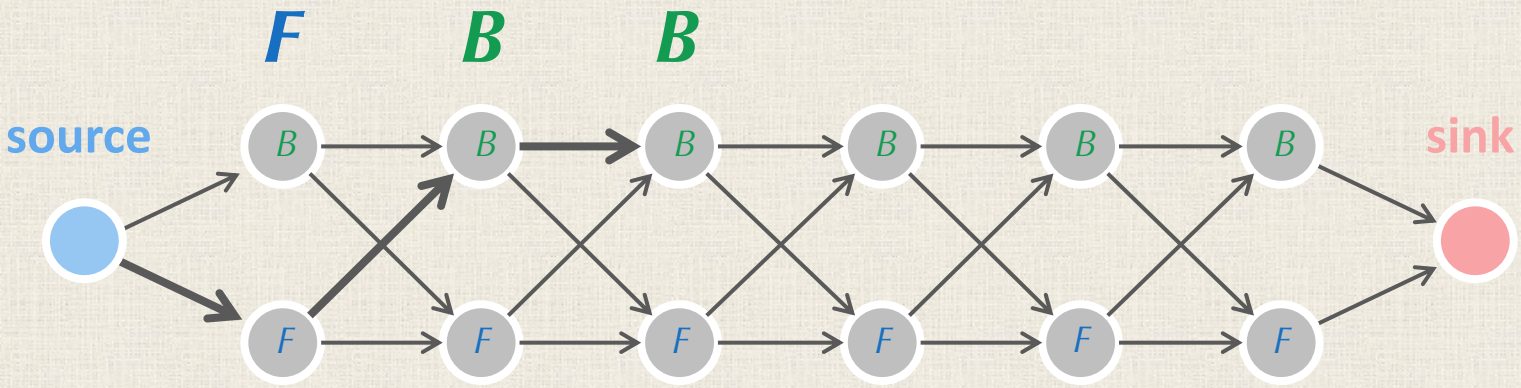
HMM diagram



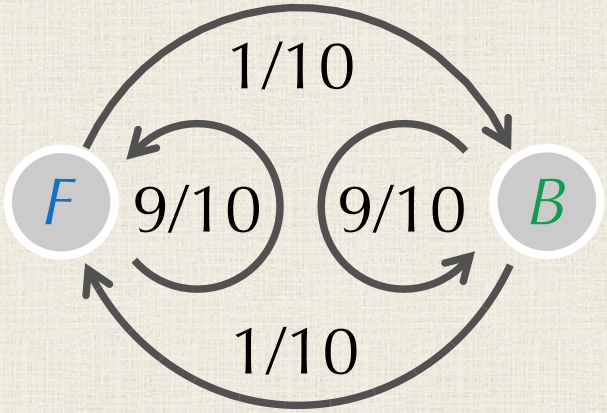
Building a DAG for the Crooked Casino



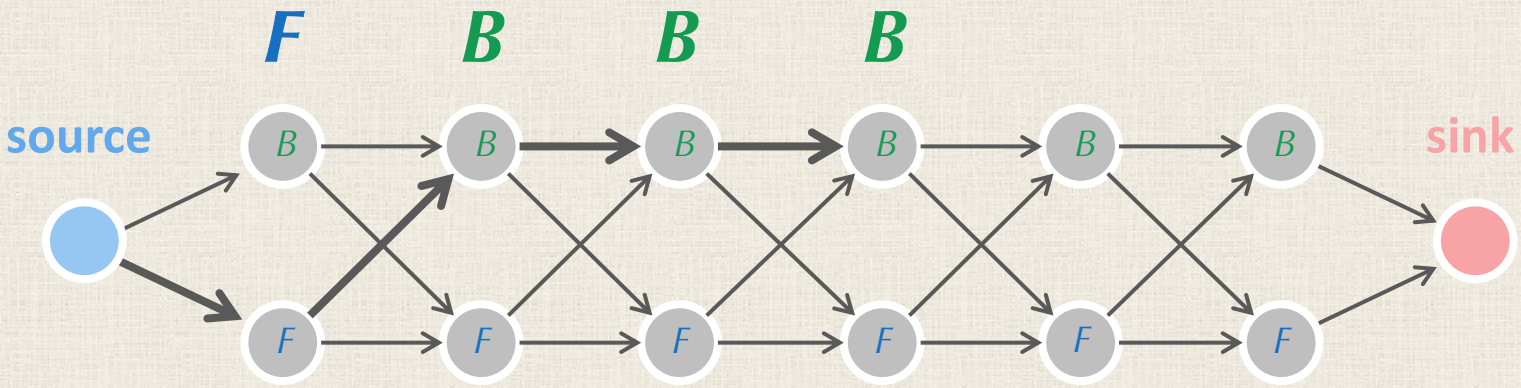
HMM diagram



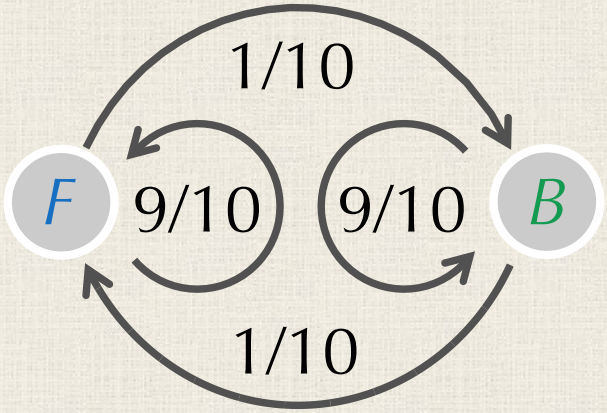
Building a DAG for the Crooked Casino



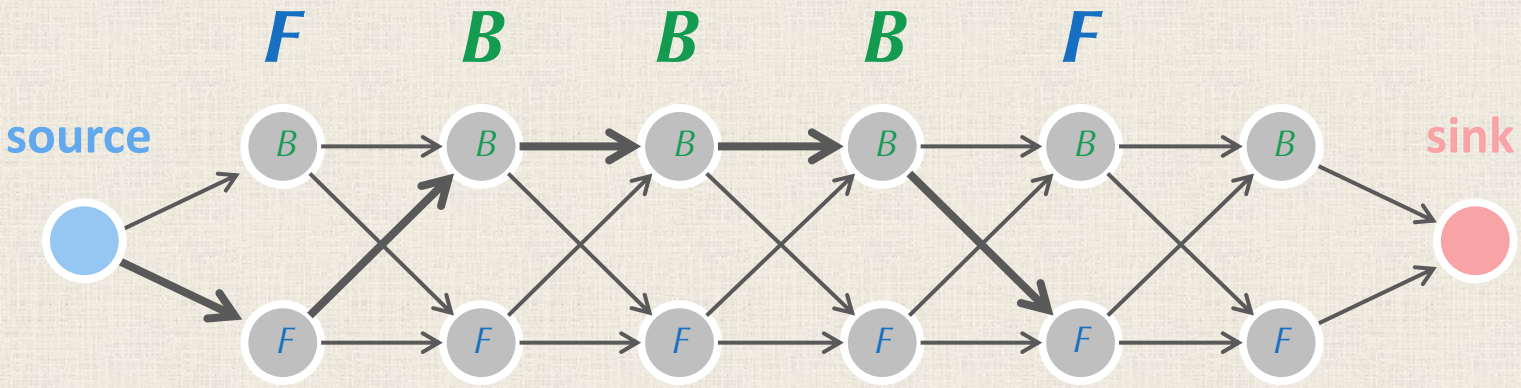
HMM diagram



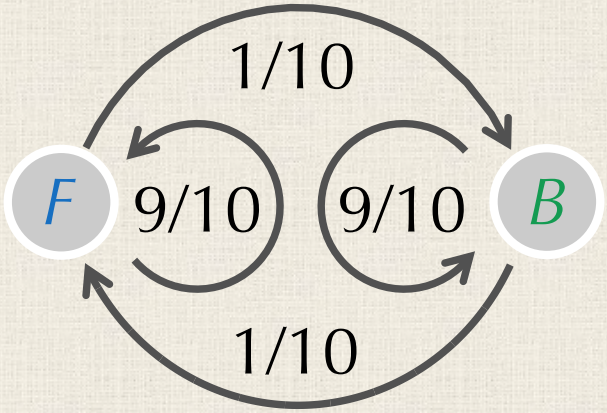
Building a DAG for the Crooked Casino



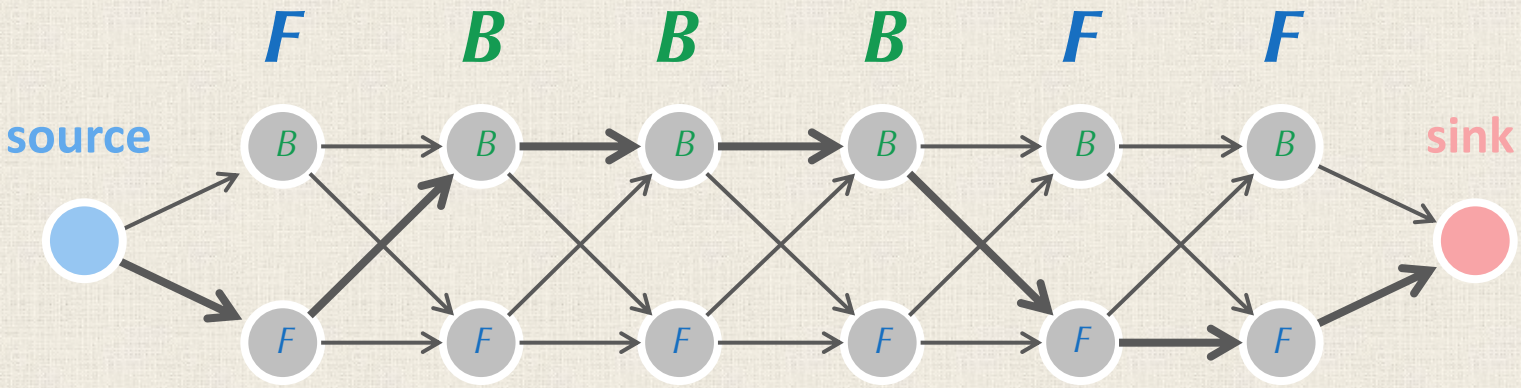
HMM diagram



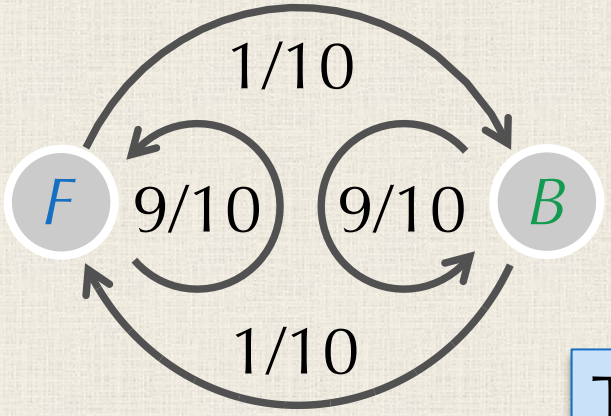
Building a DAG for the Crooked Casino



HMM diagram

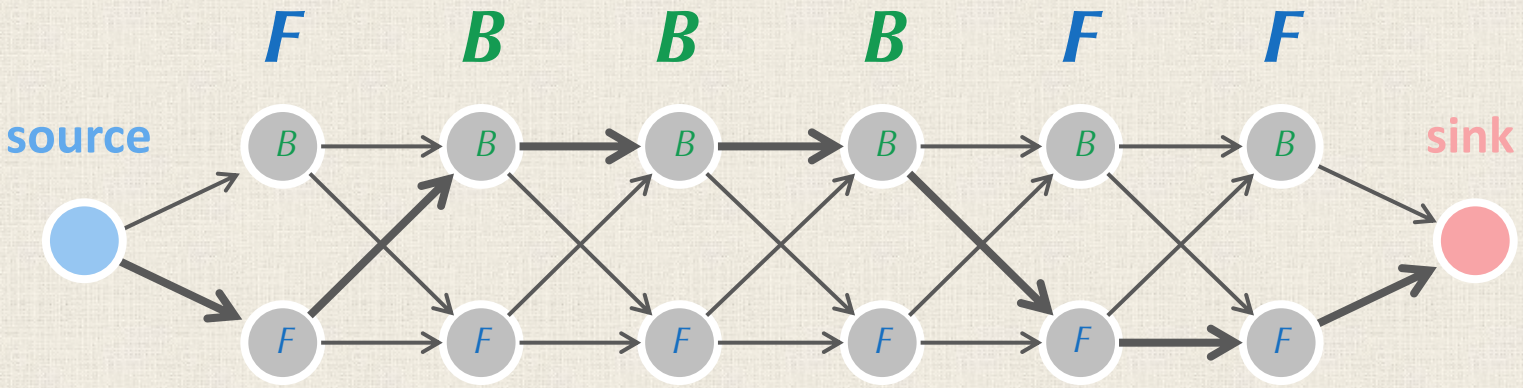


Building a DAG for the Crooked Casino

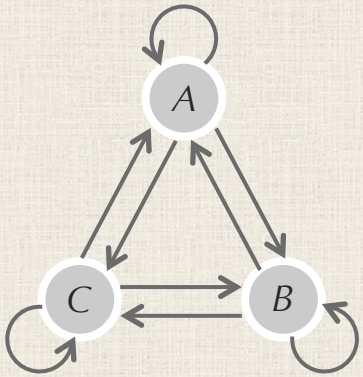


HMM diagram

This is the **Viterbi graph** of this HMM.



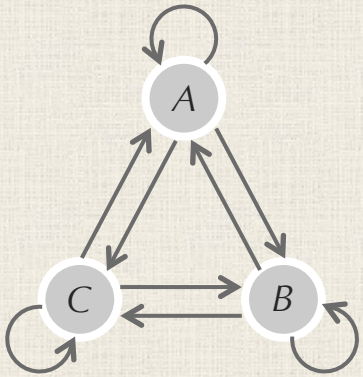
A DAG for an Arbitrary HMM



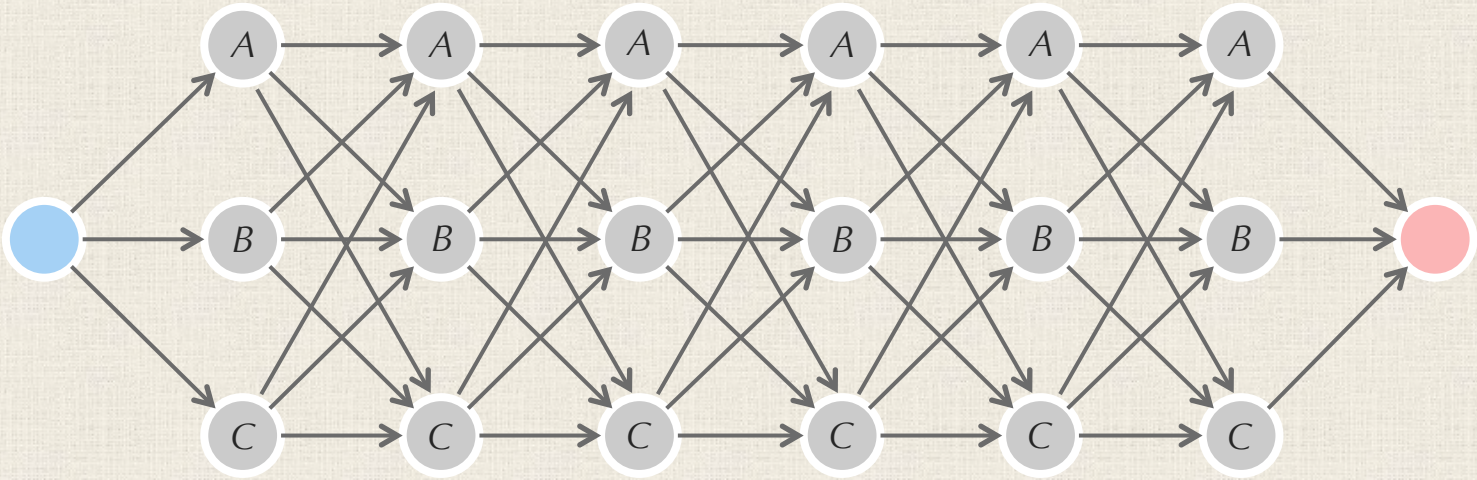
HMM diagram

Exercise: What is the Viterbi graph of this HMM diagram?

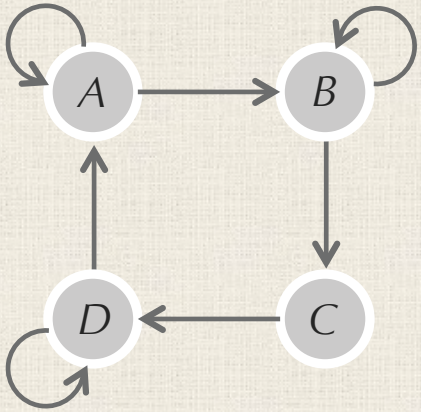
A DAG for an Arbitrary HMM



HMM diagram



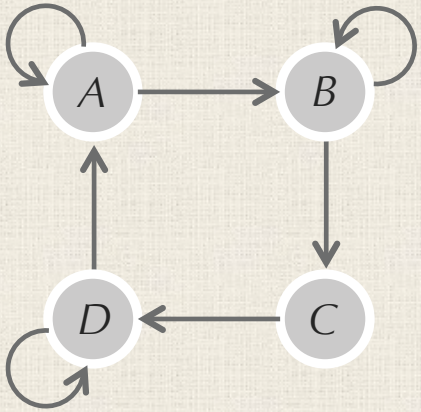
A DAG for an Arbitrary HMM



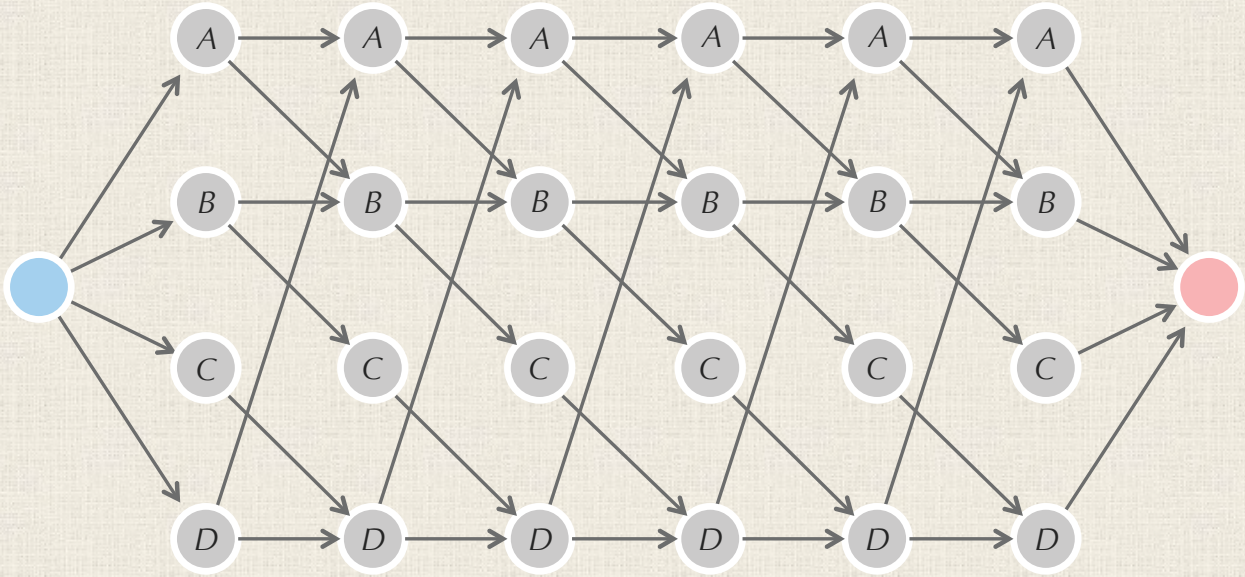
HMM diagram

Exercise: What about this HMM diagram? It has "forbidden transitions" between states.

A DAG for an Arbitrary HMM



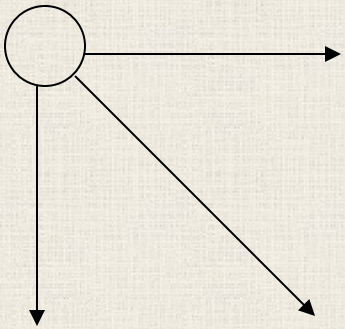
HMM diagram



Alignment Manhattan vs. Decoding Manhattan

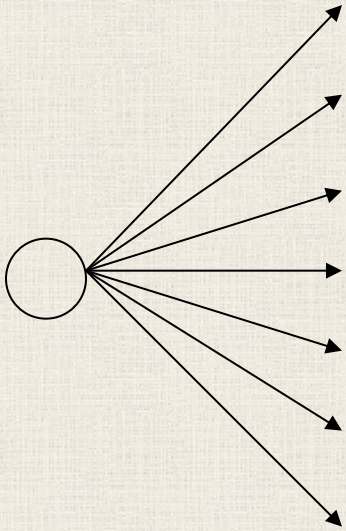
Alignment

three valid directions

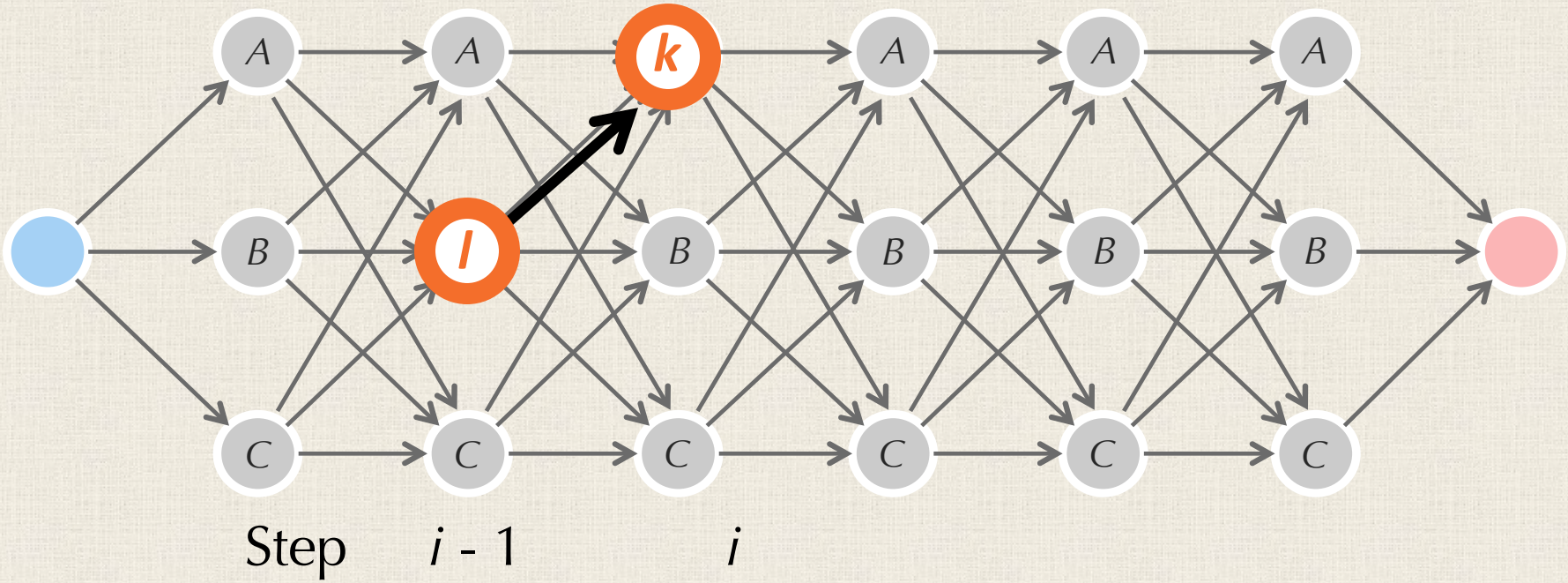


Decoding

many valid directions



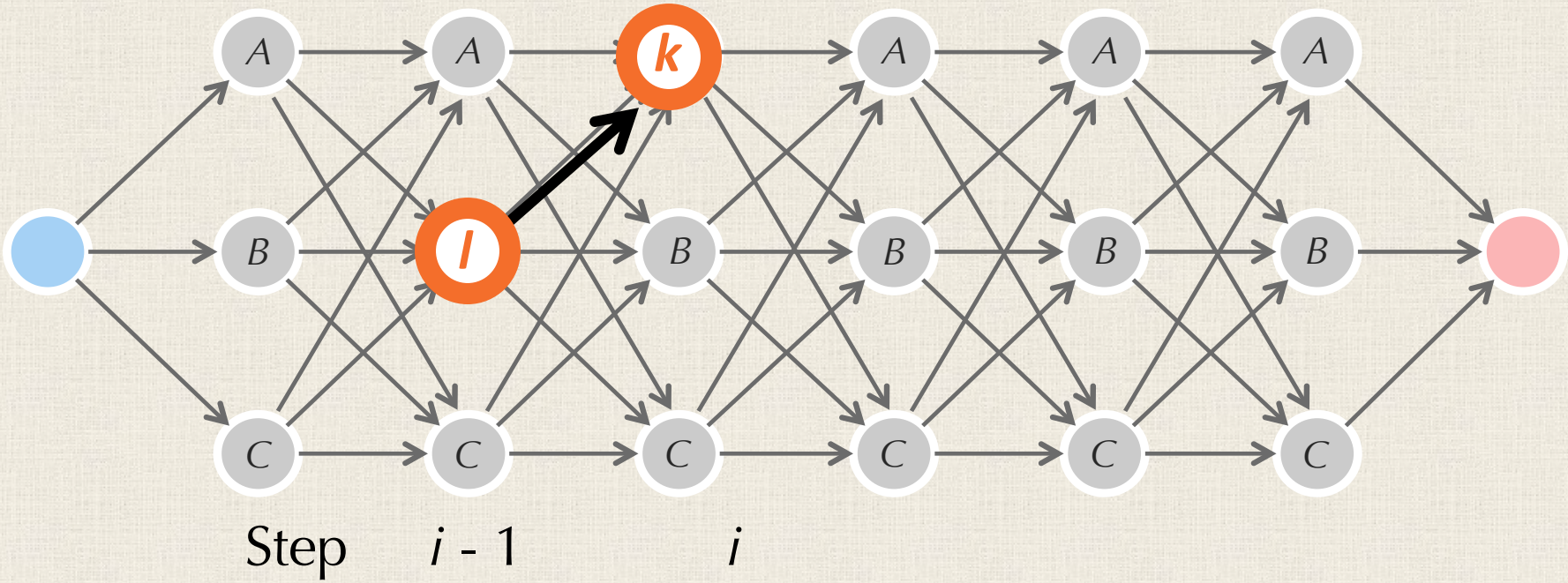
Edge-Weighting the Viterbi Graph



The edge from (l, i-1) to (k, i) corresponds to:

- transitioning from state l to state k (with probability $transition_{l,k}$)
- emitting symbol x_i (with probability $emission_k(x_i)$)

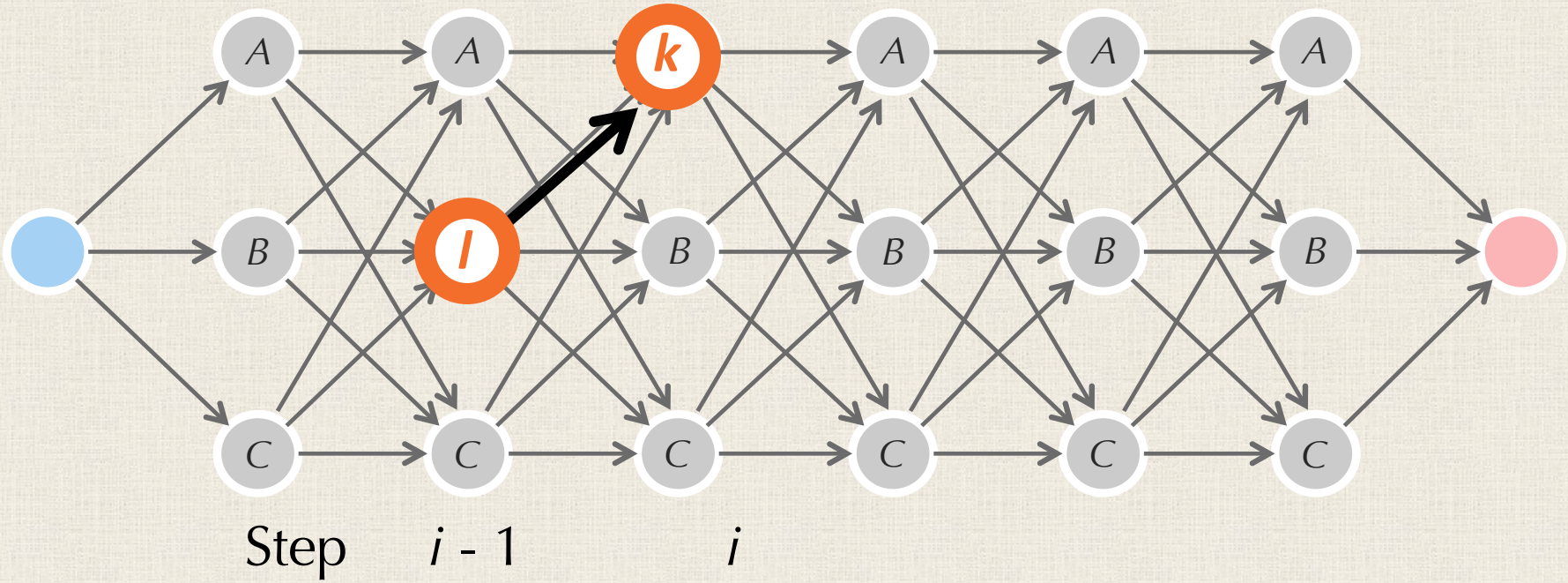
Edge-Weighting the Viterbi Graph



We weight this edge with $transition_{l,k}\ emission_k(x_i)$.
The **product weight** of a path π through the Viterbi graph is the product of its edge weights:

$$\prod_{i=1}^{n} transition_{\pi_{i-1},\pi_i} \cdot emission_{\pi_i}(x_i)$$

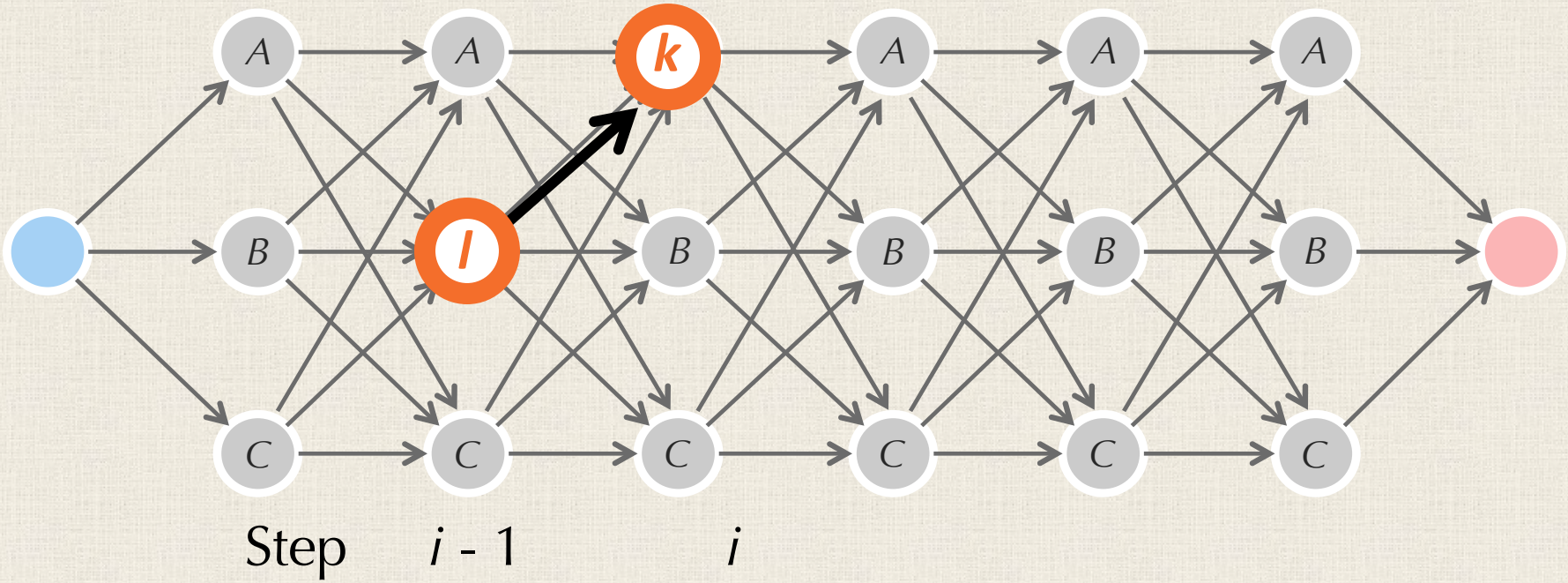
Edge-Weighting the Viterbi Graph



STOP: How does the product weight differ from Pr(x, π)?

$$\prod_{i=1}^{n} transition_{\pi_{i-1},\pi_i} \cdot emission_{\pi_i}(x_i)$$

Edge-Weighting the Viterbi Graph



Answer: It is the same ... so to maximize Pr(x, π), we are looking for a path of maximum product-weight!

$$\prod_{i=1}^{n} transition_{\pi_{i-1},\pi_i} \cdot emission_{\pi_i}(x_i)$$

Finding a "Longest" Path

Maximum Product-Weight Path in a DAG Problem:
Find a path in a DAG of maximum product weight.

- **Input:** A DAG with positive edge weights, along with source and sink nodes.
- **Output:** A path from source to sink of maximum product weight.

Finding a "Longest" Path

Maximum Product-Weight Path in a DAG Problem:
Find a path in a DAG of maximum product weight.

- **Input:** A DAG with positive edge weights, along with source and sink nodes.
- **Output:** A path from source to sink of maximum product weight.

STOP: How do we use what we have learned to solve this problem?

Answer 1: Dynamic Programming with a Recurrence Relation

Define $s_{k,i}$ as the weight of an optimal path from *source* to the node (*k, i*).

Answer 1: Dynamic Programming with a Recurrence Relation

Define $s_{k,i}$ as the weight of an optimal path from *source* to the node (*k, i*).

We have "optimal substructure" because an optimal path from *source* to (*k, i*) must be an optimal path from *source* to (*l, i*-1) for some node in column *i*-1.

Answer 1: Dynamic Programming with a Recurrence Relation

Define $s_{k,i}$ as the weight of an optimal path from *source* to the node (*k*, *i*).

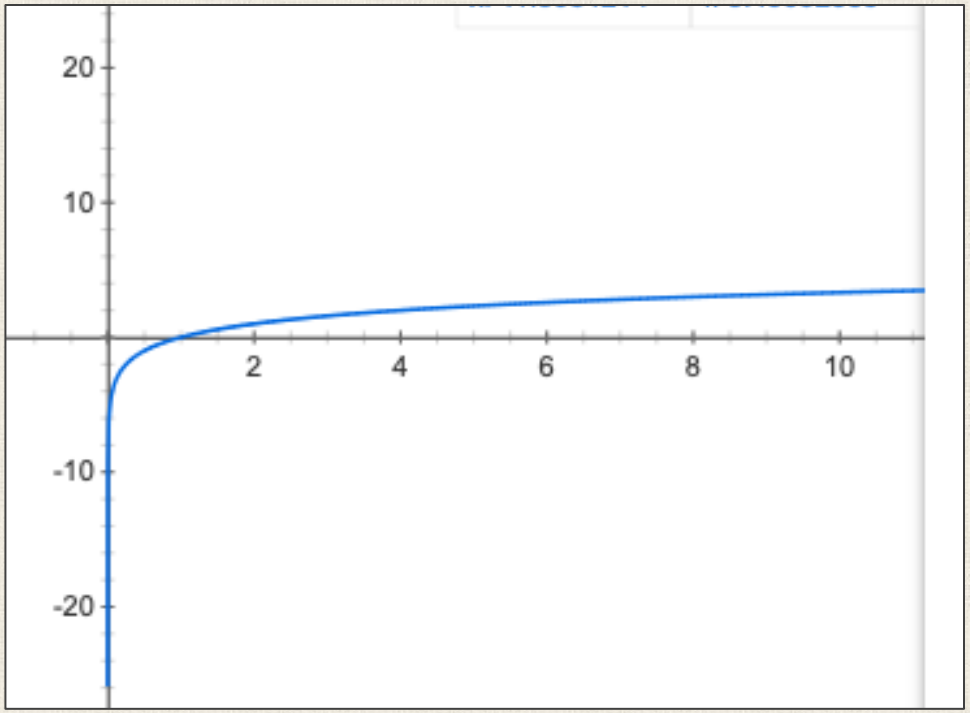
We have "optimal substructure" because an optimal path from *source* to (*k*, *i*) must be an optimal path from *source* to (*l*, *i*-1) for some node in column *i*-1.

$$
\begin{aligned}
s_{k,i} &= \max_{\text{all states } l} \left\{ s_{l,i-1} \cdot (\text{weight of edge between nodes} (l, i-1) \text{ and } (k,i)) \right\} \\
&= \max_{\text{all states } l} \left\{ s_{l,i-1} \cdot \textsc{Weight}_i(l,k) \right\} \\
&= \max_{\text{all states } l} \left\{ s_{l,i-1} \cdot \mathit{transition}_{\pi_{i-1},\pi_i} \cdot \mathit{emission}_{\pi_i}(x_i) \right\}
\end{aligned}
$$

Answer 2: You Never Thought Logarithms Would be Useful ...

Two logarithm properties:

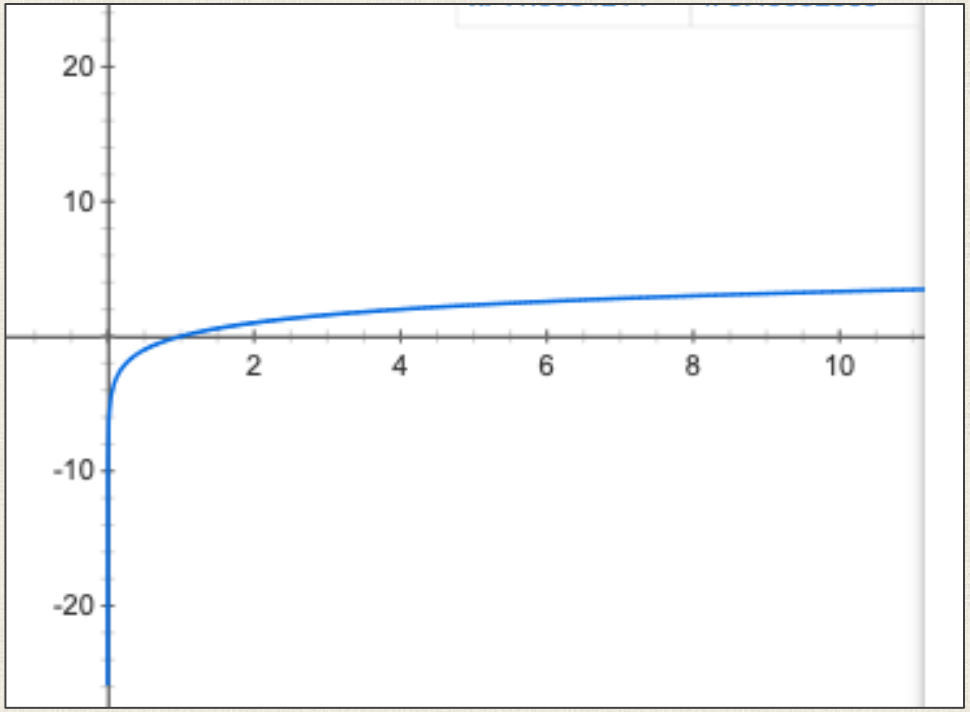
1. $\log(x_1 \cdot x_2) = \log(x_1) + \log(x_2)$.
2. It's increasing; that is, if $x_1 < x_2$, then $\log(x_1) < \log(x_2)$.



Answer 2: You Never Thought Logarithms Would be Useful ...

Two logarithm properties:

1. $\log(x_1 \cdot x_2) = \log(x_1) + \log(x_2)$.
2. It's increasing; that is, if $x_1 < x_2$, then $\log(x_1) < \log(x_2)$.



STOP: How are these properties useful for our purposes?

Answer 2: You Never Thought Logarithms Would be Useful ...

Two logarithm properties:

1. $\log(x_1 \cdot x_2) = \log(x_1) + \log(x_2)$.
2. It's increasing; that is, if $x_1 < x_2$, then $\log(x_1) < \log(x_2)$.

If we take the logarithm of a product of edge weights w_1 ... w_n , thenby property 1, we obtain a sum of edge weights $\log(w_1) + \ldots + \log(w_n)$.

Answer 2: You Never Thought Logarithms Would be Useful ...

Two logarithm properties:

1. $\log(x_1 \cdot x_2) = \log(x_1) + \log(x_2)$.
2. It's increasing; that is, if $x_1 < x_2$, then $\log(x_1) < \log(x_2)$.

If we take the logarithm of a product of edge weights w_1 ... w_n , thenby property 1, we obtain a sum of edge weights $\log(w_1) + \ldots + \log(w_n)$.

And if the weights correspond to a maximum weight path, this optimality will be preserved by property 2.

Our Problem is "Longest Path in a DAG" in Disguise!

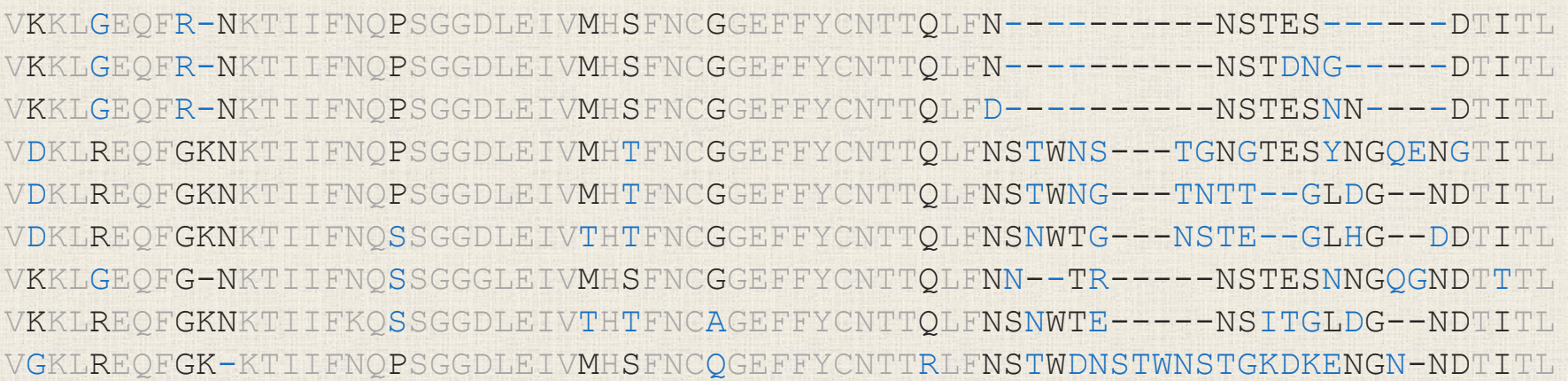
Maximum Product-Weight Path in a DAG Problem:
Find a path in a DAG of maximum product weight.

- **Input:** A DAG with positive edge weights, along with source and sink nodes.
- **Output:** A path from source to sink of maximum product weight.

PROFILE HMMS FOR SEQUENCE ALIGNMENT

Remember Our Problem

Once we have a collection of *known* protein alignments ("families"), we need to be able to identify which family a new protein belongs to. That is, add a new string into an existing alignment.



Remember Our Problem

Once we have a collection of *known* protein alignments (″families″), we need to be able to identify which family a new protein belongs to. That is, add a new string into an existing alignment.

This sets up as an HMM problem, since when adding a *new* string to an alignment, we have:

- a decision to make at each step (align? Gap symbol?)
- We're looking for a ″path″ (decisions) of sorts that ″makes the most sense″.

From an Alignment to a Profile

	1	2	3	4	5		6	7	8
	A	C	D	E	F	A	C	A	D	F
	A	F	D	A	-	-	-	C	C	F
Alignment	A	-	-	E	F	D	-	F	D	C
	A	C	A	E	F	-	-	A	-	C
	A	D	D	E	F	A	A	A	D	F

Seed alignment: remove columns if the fraction of space symbols ("-") exceeds a threshold θ.

From an Alignment to a Profile

	1	2	3	4	5		6	7	8
Alignment	A	C	D	E	F	A C	A	D	F
	A	F	D	A	–	– –	C	C	F
	A	–	–	E	F	D –	F	D	C
	A	C	A	E	F	– –	A	–	C
	A	D	D	E	F	A A	A	D	F

	1	2	3	4	5	6	7	8
*Alignment**	A	C	D	E	F	A	D	F
	A	F	D	A	–	C	C	F
	A	–	–	E	F	F	D	C
	A	C	A	E	F	A	–	C
	A	D	D	E	F	A	D	F

From an Alignment to a Profile

	1	2	3	4	5	6	7	8
Alignment	A	C	D	E	F	A C	A	D	F
	A	F	D	A	–	– –	C	C	F
	A	–	–	E	F	D –	F	D	C
	A	C	A	E	F	– –	A	–	C
	A	D	D	E	F	A A	A	D	F

	1	2	3	4	5	6	7	8
*Alignment**	A	C	D	E	F	A	D	F
	A	F	D	A	–	C	C	F
	A	–	–	E	F	F	D	C
	A	C	A	E	F	A	–	C
	A	D	D	E	F	A	D	F

PROFILE(*Alignment**)								
A	1	0	1/4	1/5	0	3/5	0	0
C	0	2/4	0	0	0	1/5	1/4	2/5
D	0	1/4	3/4	0	0	0	3/4	0
E	0	0	0	4/5	0	0	0	0
F	0	1/4	0	0	1	1/5	0	3/5

From an Alignment to a Profile

	1	2	3	4	5	6	7	8
Alignment	A	C	D	E	F	A C	A	D	F
	A	F	D	A	–	– –	C	C	F
	A	–	–	E	F	D –	F	D	C
	A	C	A	E	F	– –	A	–	C
	A	D	D	E	F	A A	A	D	F

	1	2	3	4	5	6	7	8
*Alignment**	A	C	D	E	F	A	D	F
	A	F	D	A	–	C	C	F
	A	–	–	E	F	F	D	C
	A	C	A	E	F	A	–	C
	A	D	D	E	F	A	D	F

PROFILE(*Alignment**)

	1	2	3	4	5	6	7	8
A	1	0	1/4	1/5	0	3/5	0	0
C	0	2/4	0	0	0	1/5	1/4	2/5
D	0	1/4	3/4	0	0	0	3/4	0
E	0	0	0	4/5	0	0	0	0
F	0	1/4	0	0	1	1/5	0	3/5

HMM diagram

A D D A F F D F

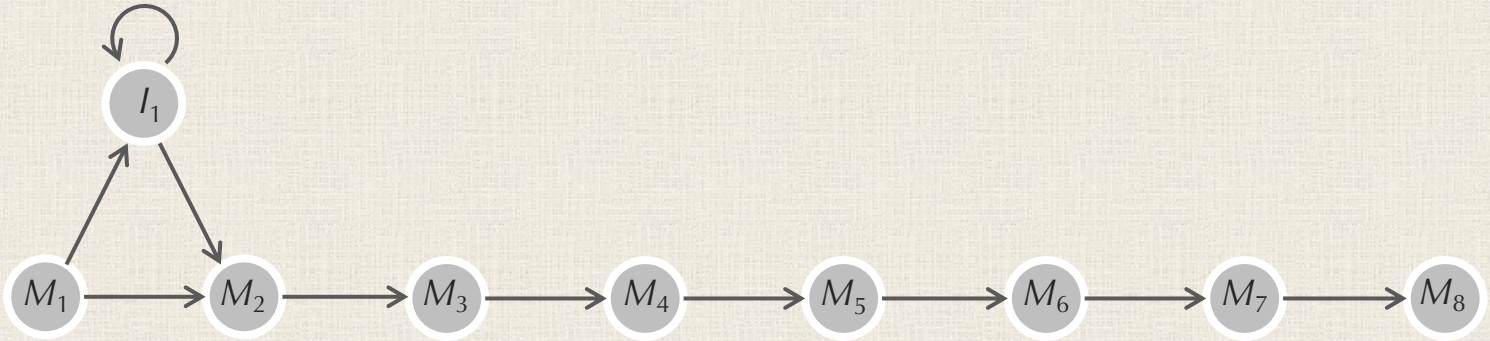
1 * .25 * .75 * .20 * 1 * .20 * .75 * .60

Toward a Profile HMM

A **F** D D A F F D F

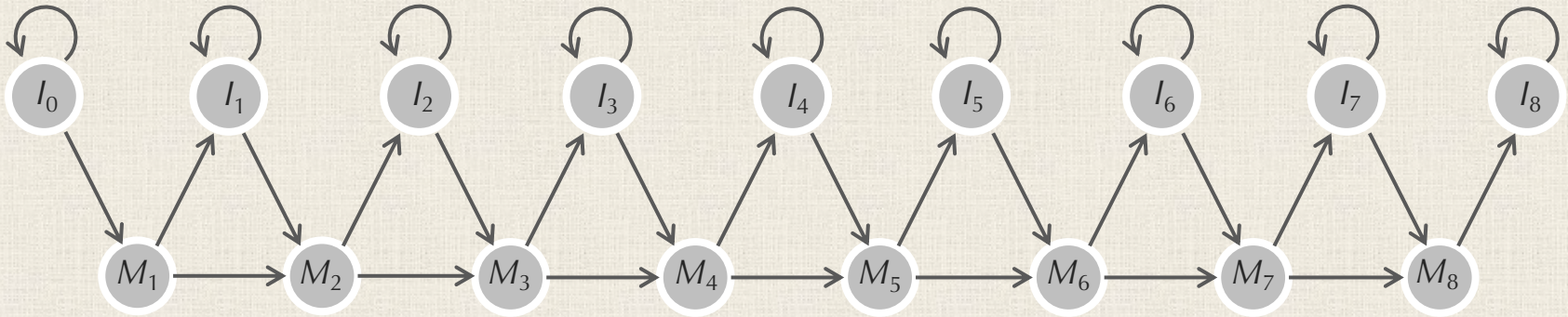
STOP: How do we model insertions?

Toward a Profile HMM: Insertions



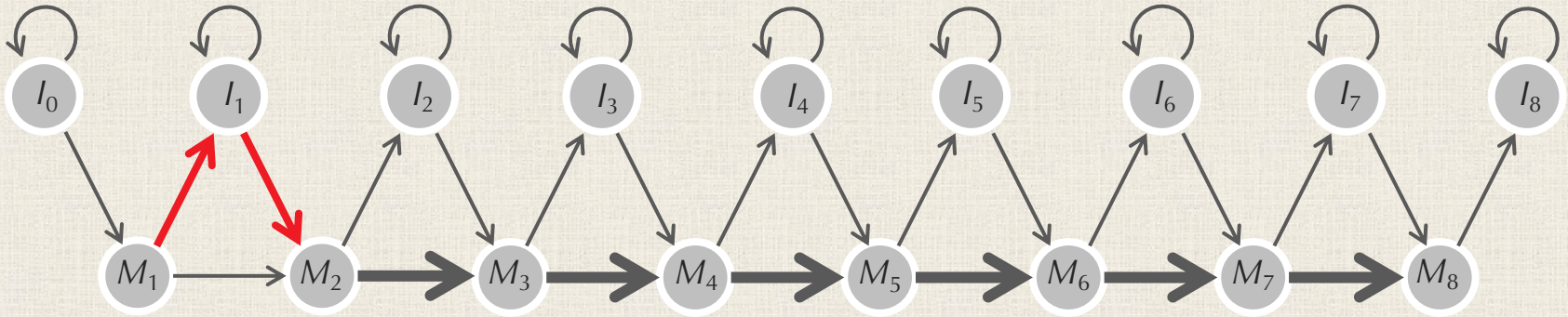
A **F** D D A F F D F

Toward a Profile HMM: Insertions



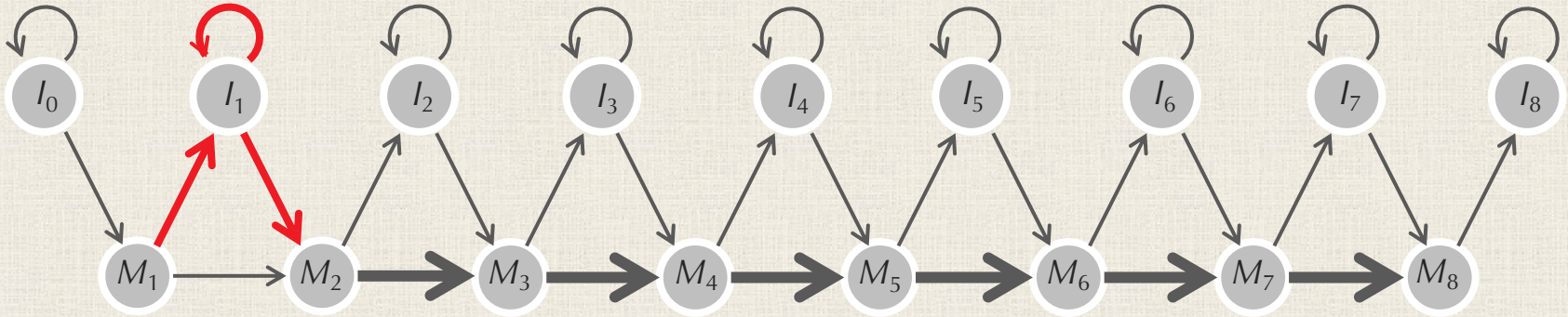
A **F** D D A F F D F

Toward a Profile HMM: Insertions



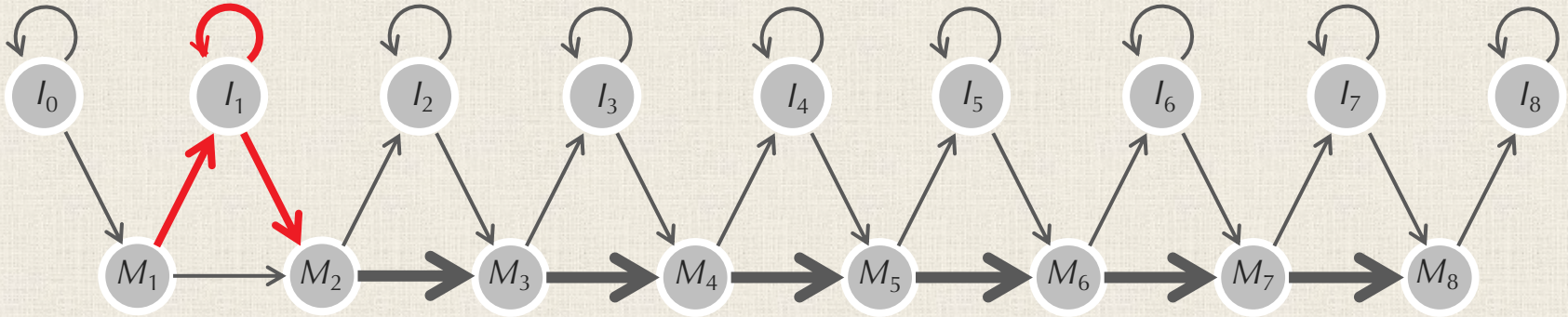
A **F** D D A F F D F

Toward a Profile HMM: Insertions



A **F** D D A F F D F

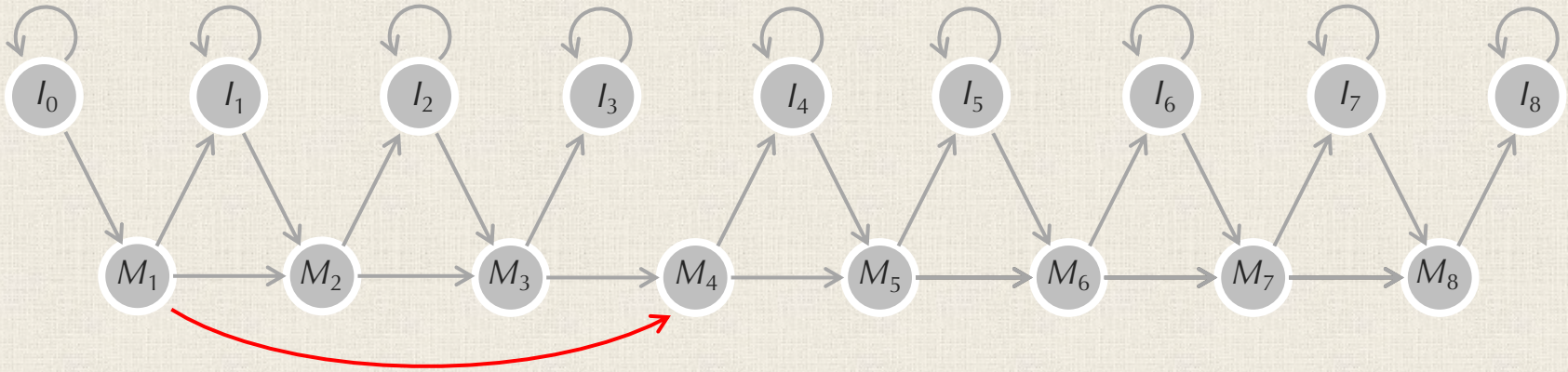
Toward a Profile HMM: Insertions



A **F** D D A F F D F

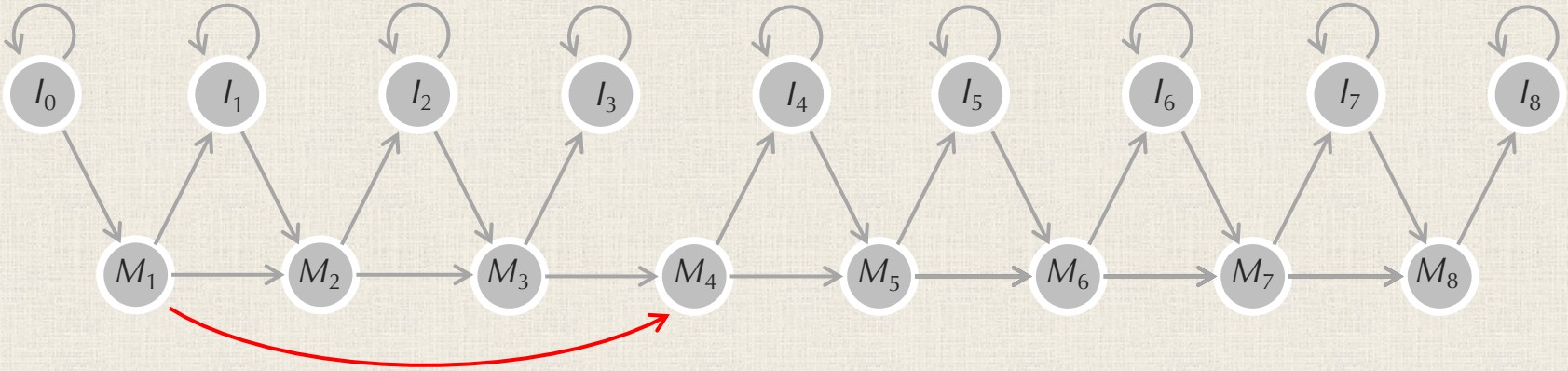
STOP: How do we model deletions?

Toward a Profile HMM: Deletions



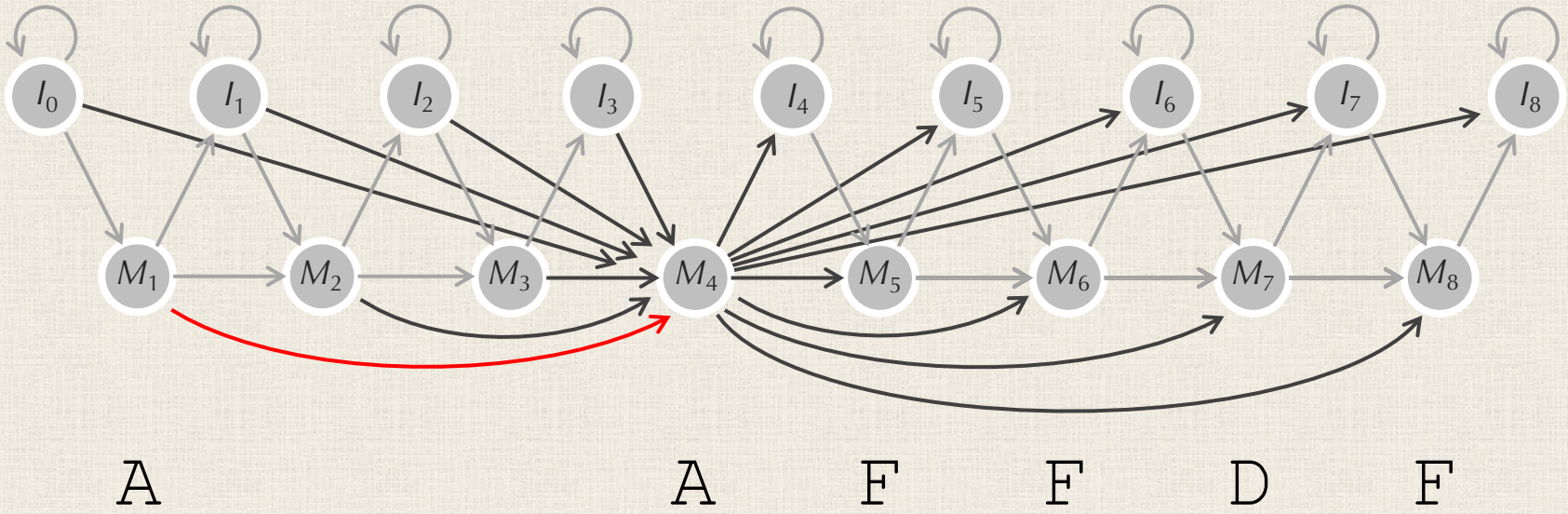
A A F F D F

Toward a Profile HMM: Deletions



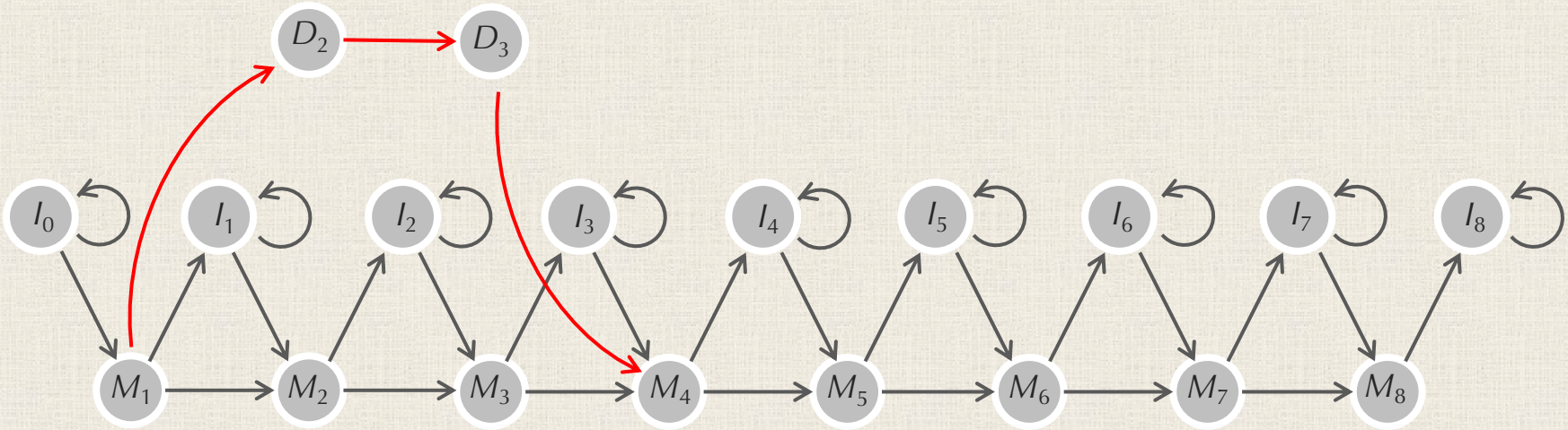
A A F F D F

Toward a Profile HMM: Deletions



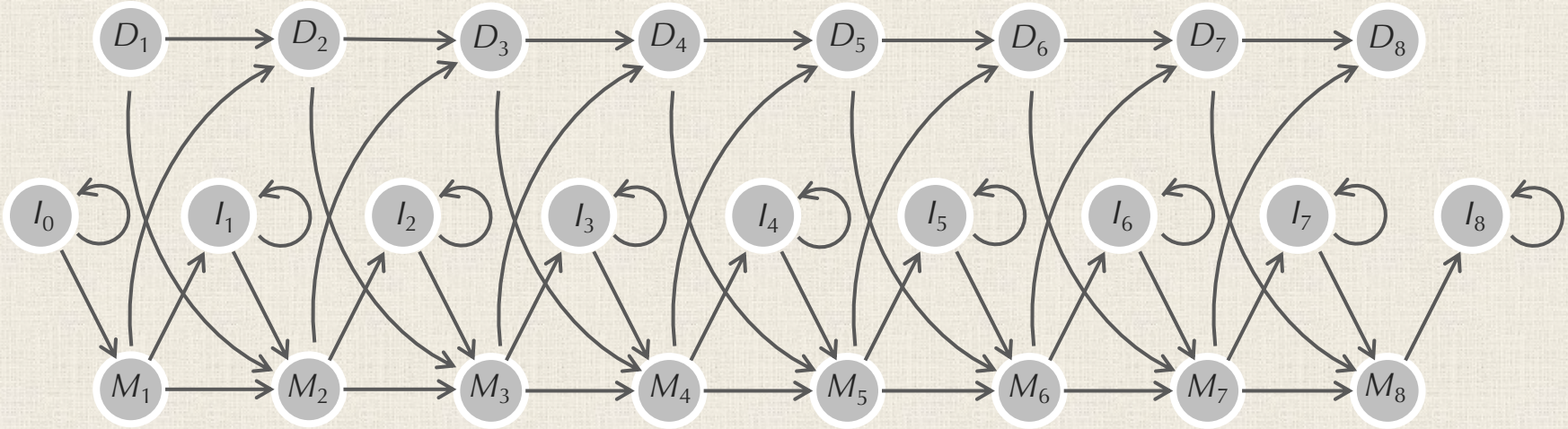
STOP: What issues do you see with this approach?

Toward a Profile HMM: Deletions



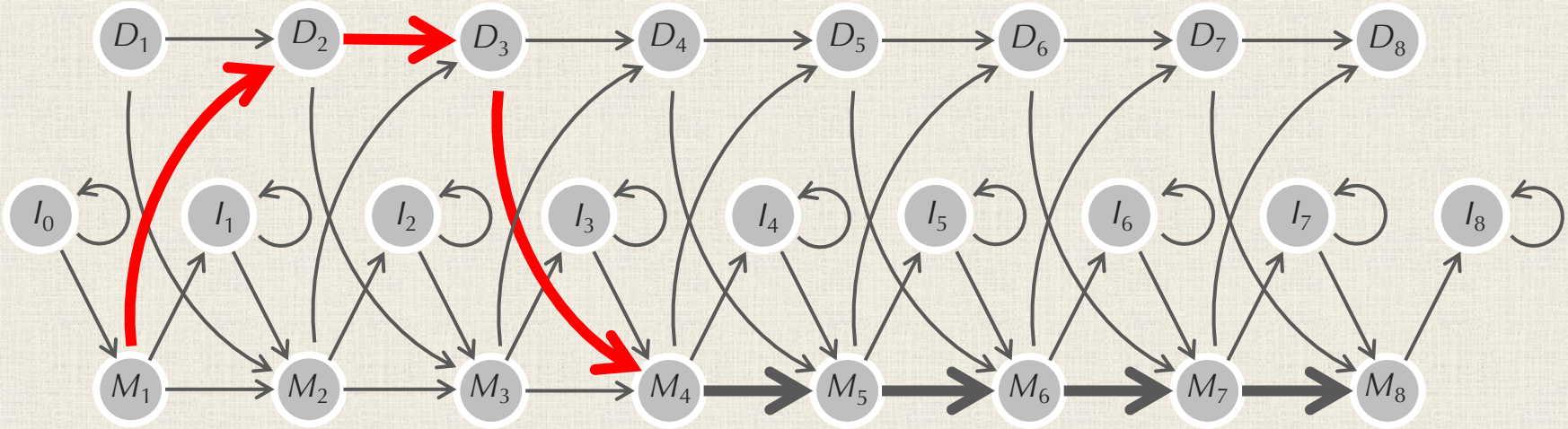
Answer: Just like with affine alignment, we can have fewer edges if we create separate "deletion states".

Toward a Profile HMM: Deletions



A A F F D F

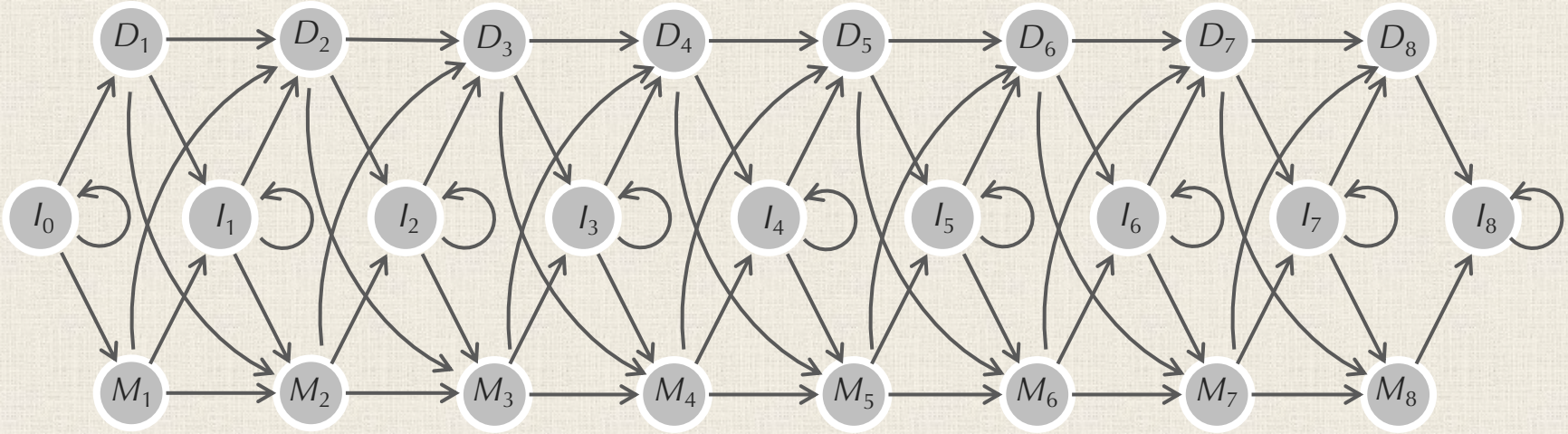
Toward a Profile HMM: Deletions



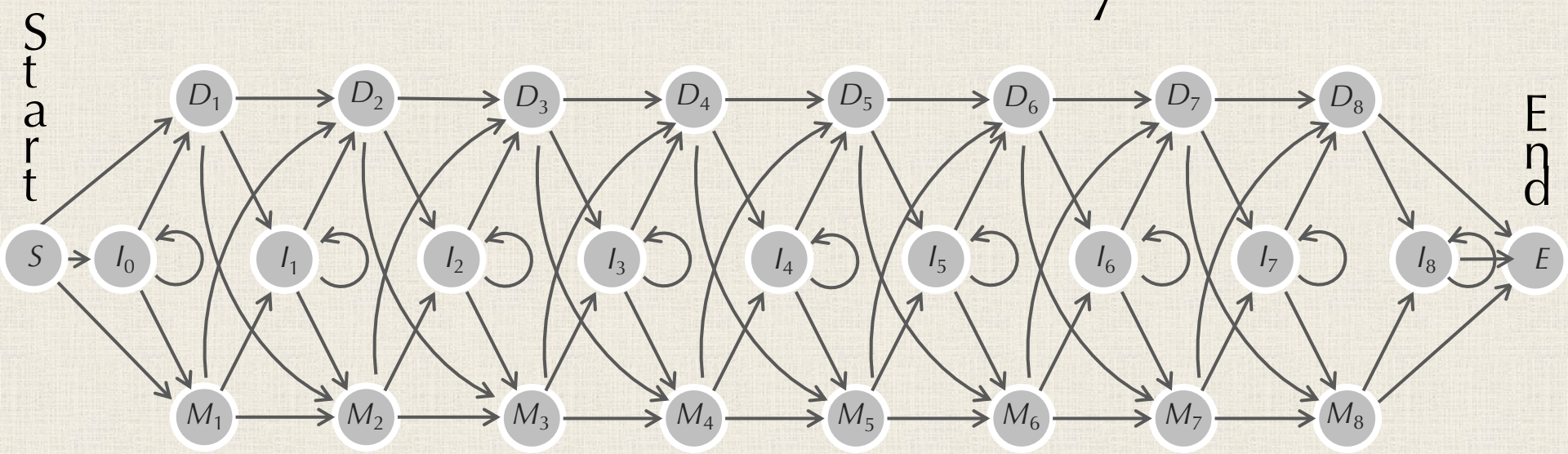
A A F F D F

STOP: Are any edges still missing in this HMM diagram?

Adding Edges Between Insertion/Deletion States



The Profile HMM is Ready to Use!



This is the HMM diagram of the **profile HMM** of a seed alignment.

Summarizing a Profile HMM

Σ: an **alphabet** of emitted symbols

Amino acids

States : a set of **hidden states**

Start, end, match, insertion, and deletion states

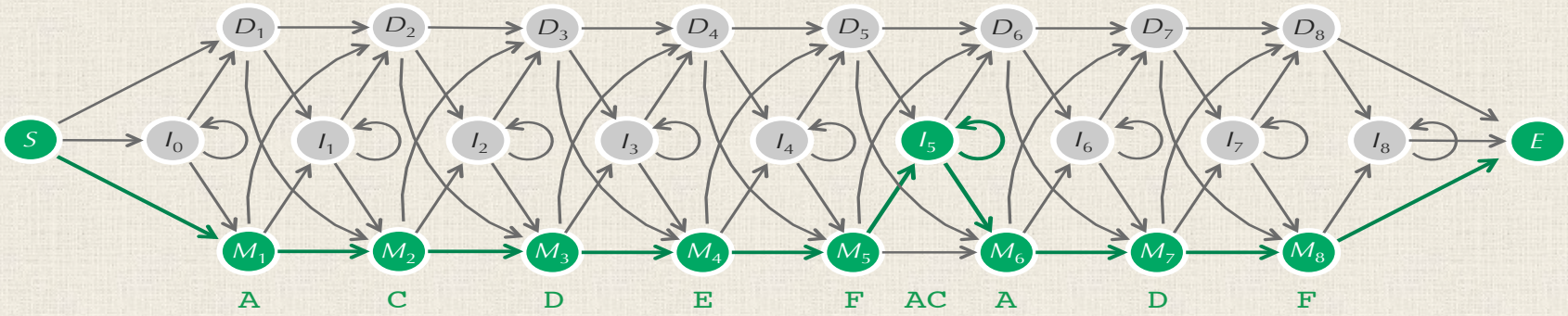
Transition = ($transition_{l,k}$): a $|States| \times |States|$ matrix of **transition probabilities** (of changing from state *l* to state *k*)

Emission= ($emission_k(b)$): a $|States| \times |\sum|$ matrix of **emission probabilities** (emitting symbol *b* when HMM is in state *k*)

It is not yet clear what the transition and emission probabilities should be!

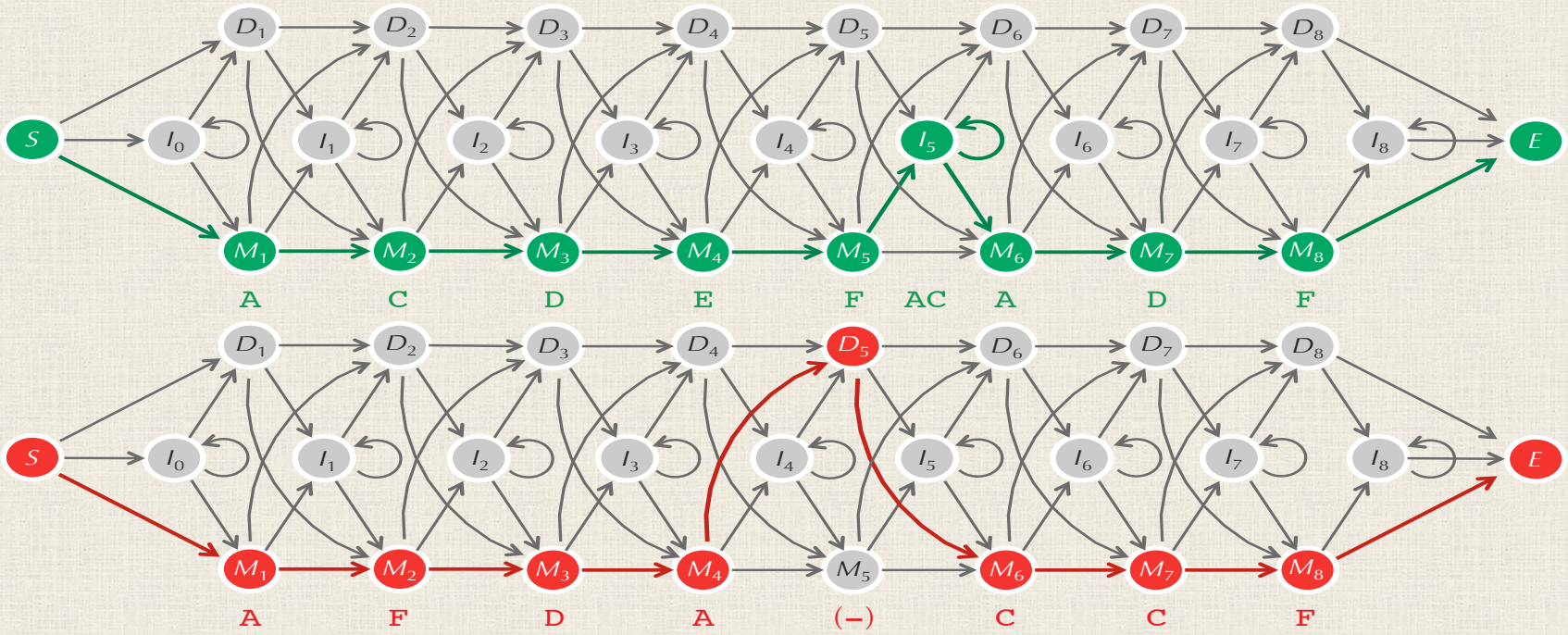
Hidden Paths Through Profile HMM

A	C	D	E	F	A C	A	D	F
A	F	D	A	–	– –	C	C	F
A	–	–	E	F	D –	F	D	C
A	C	A	E	F	– –	A	–	C
A	D	D	E	F	A A	A	D	F



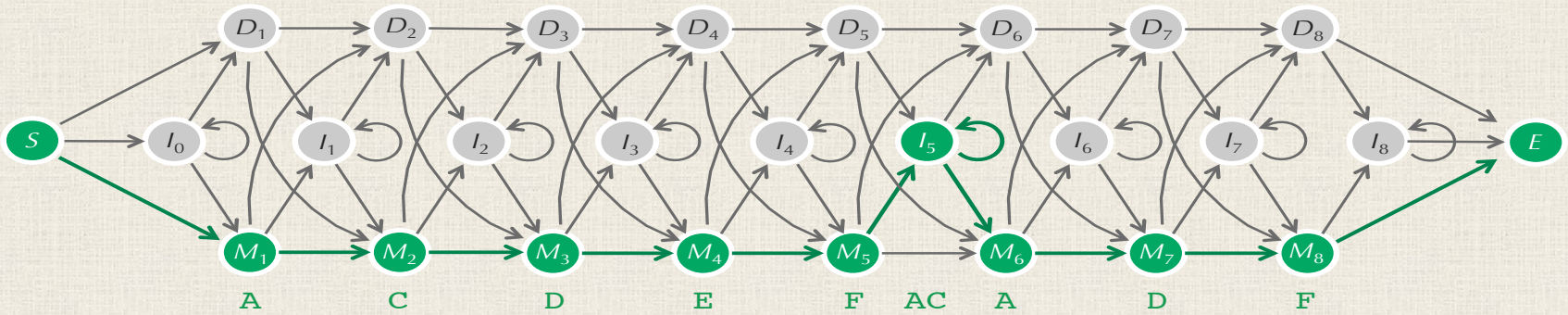
Hidden Paths Through Profile HMM

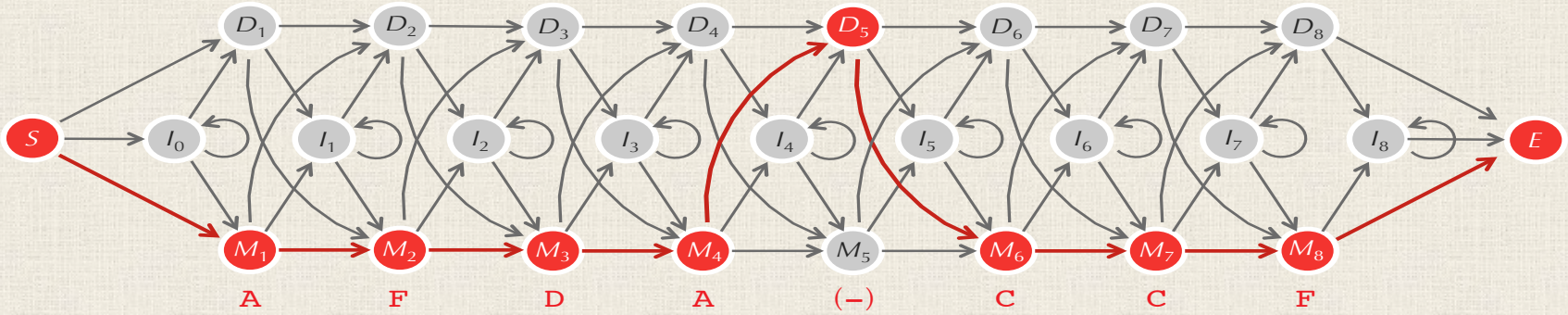
A	C	D	E	F	A C	A	D	F
A	F	D	A	–	– –	C	C	F
A	–	–	E	F	D –	F	D	C
A	C	A	E	F	– –	A	–	C
A	D	D	E	F	A A	A	D	F

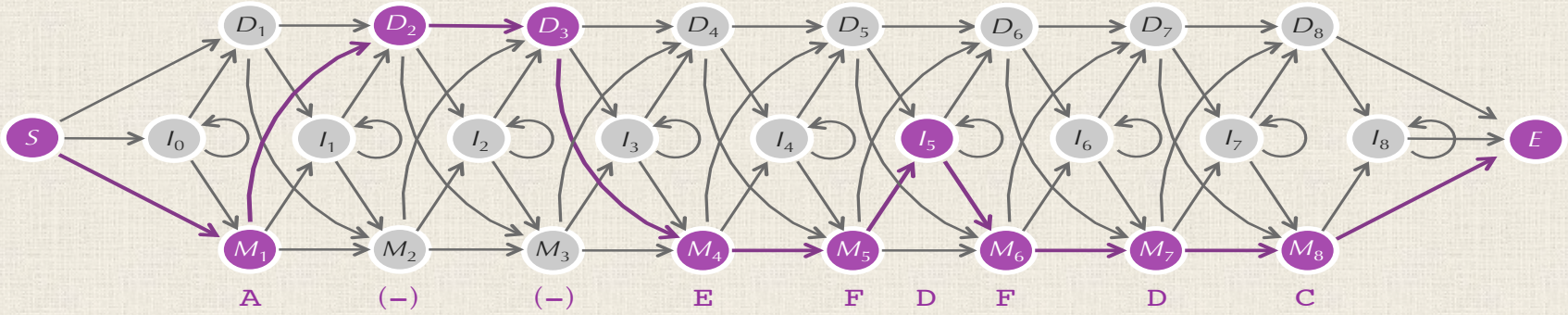


Hidden Paths Through Profile HMM

A	C	D	E	F	A C	A	D	F
A	F	D	A	–	– –	C	C	F
A	–	–	E	F	D –	F	D	C
A	C	A	E	F	– –	A	–	C
A	D	D	E	F	A A	A	D	F

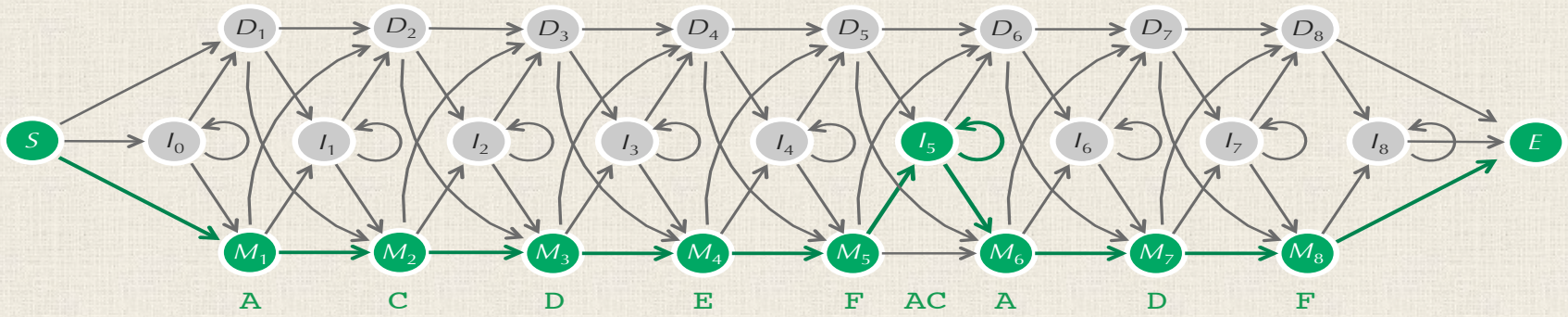


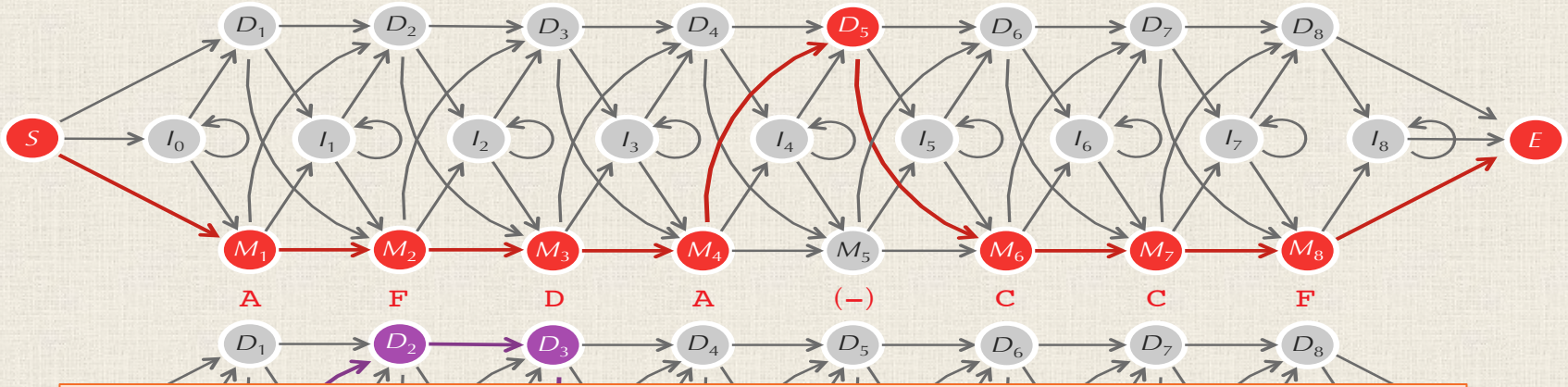


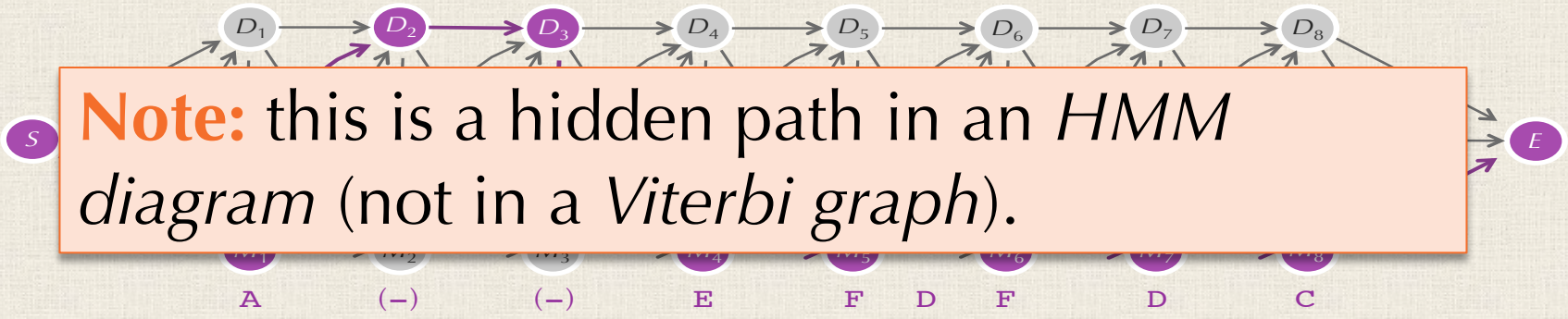


Hidden Paths Through Profile HMM

A	C	D	E	F	A	C	A	D	F
A	F	D	A	–	–	–	C	C	F
A	–	–	E	F	D	–	F	D	C
A	C	A	E	F	–	–	A	–	C
A	D	D	E	F	A	A	A	D	F

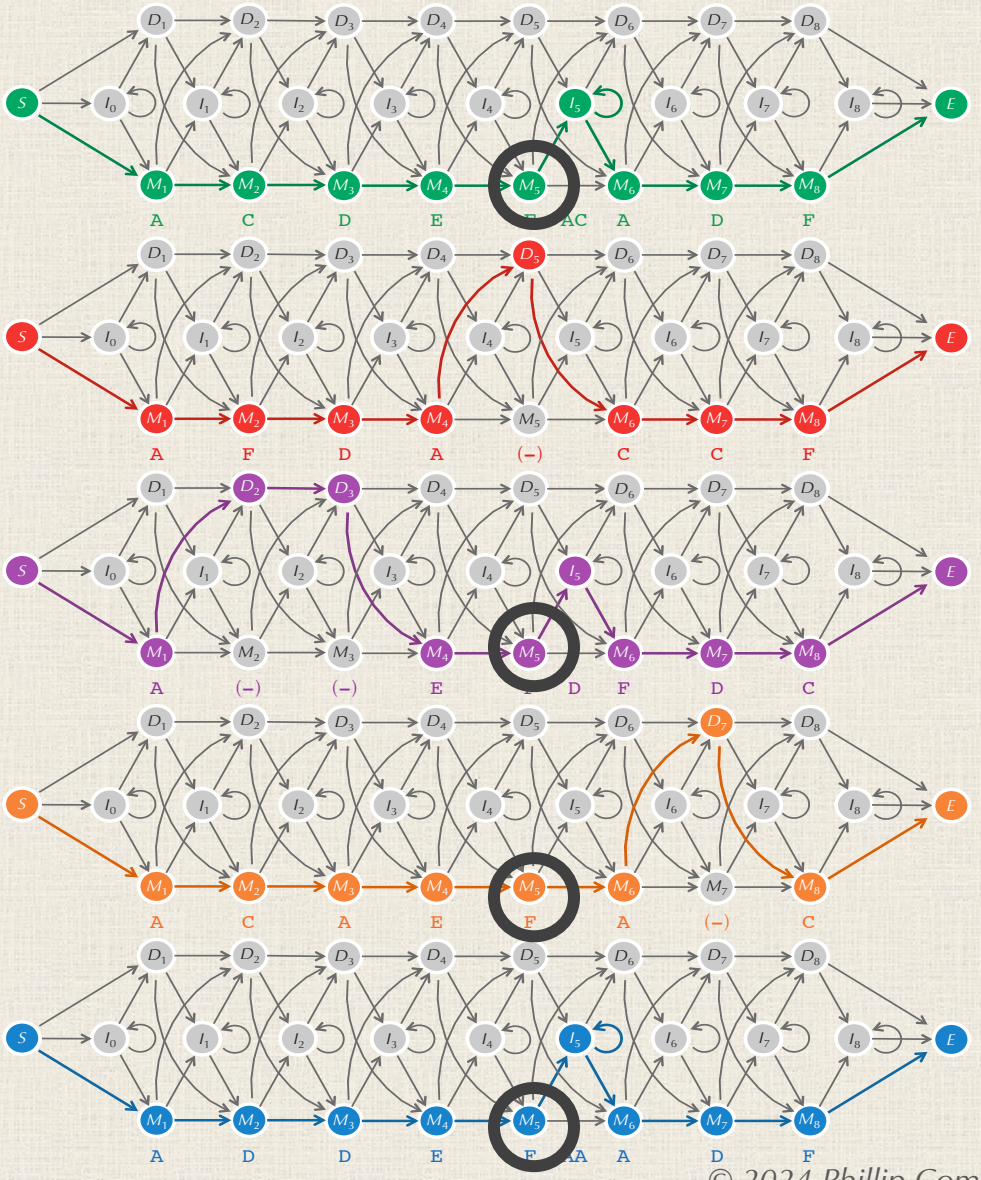






Note: this is a hidden path in an *HMM diagram* (not in a *Viterbi graph*).

Transition Probabilities of Profile HMM



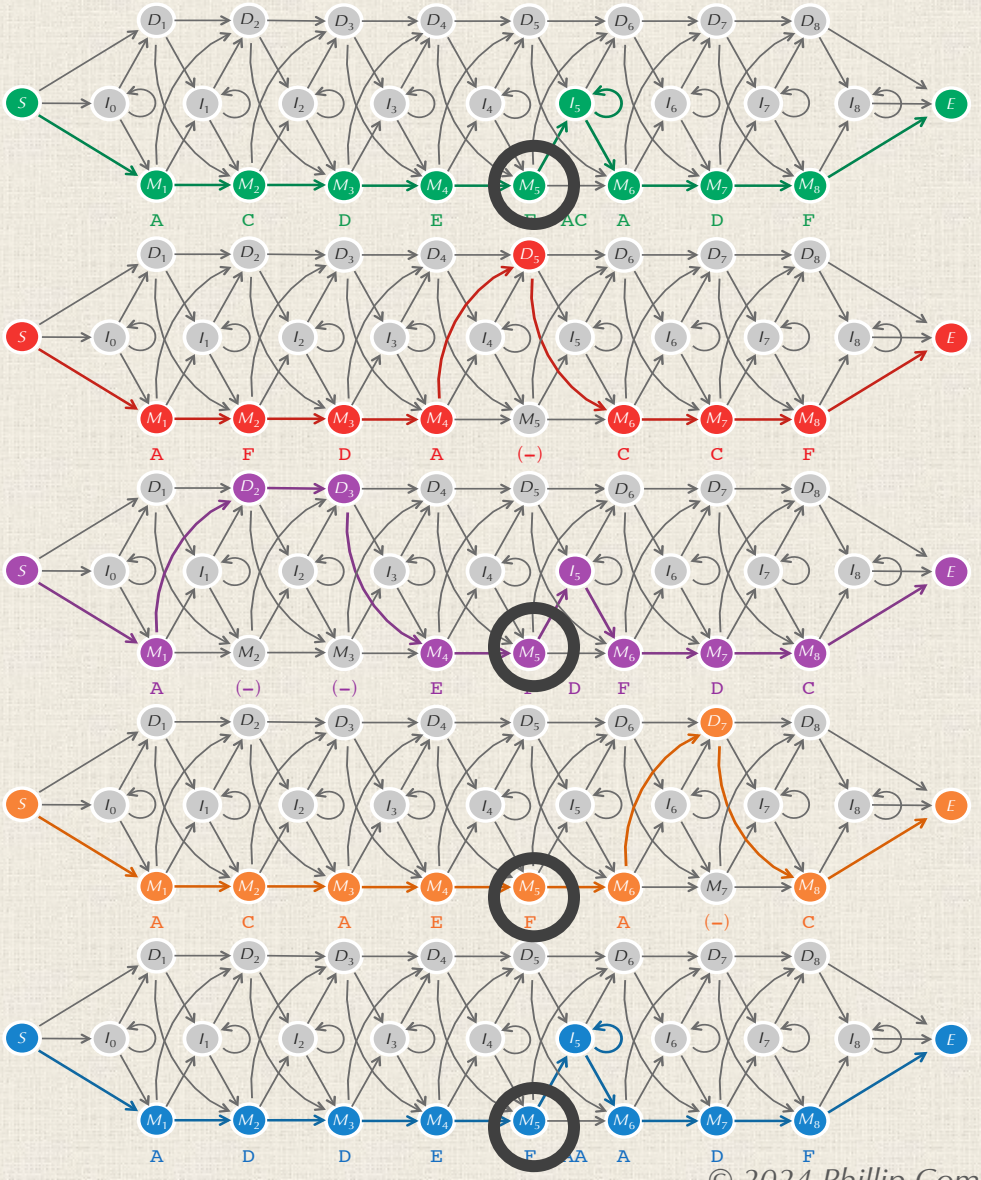
4 transitions from M_5 :

1 + 1 + 1 = 3 into I_5

1 into M_6

0 into D_6

Transition Probabilities of Profile HMM



4 transitions from M_5 :

1 + 1 + 1 = 3 into I_5

1 into M_6

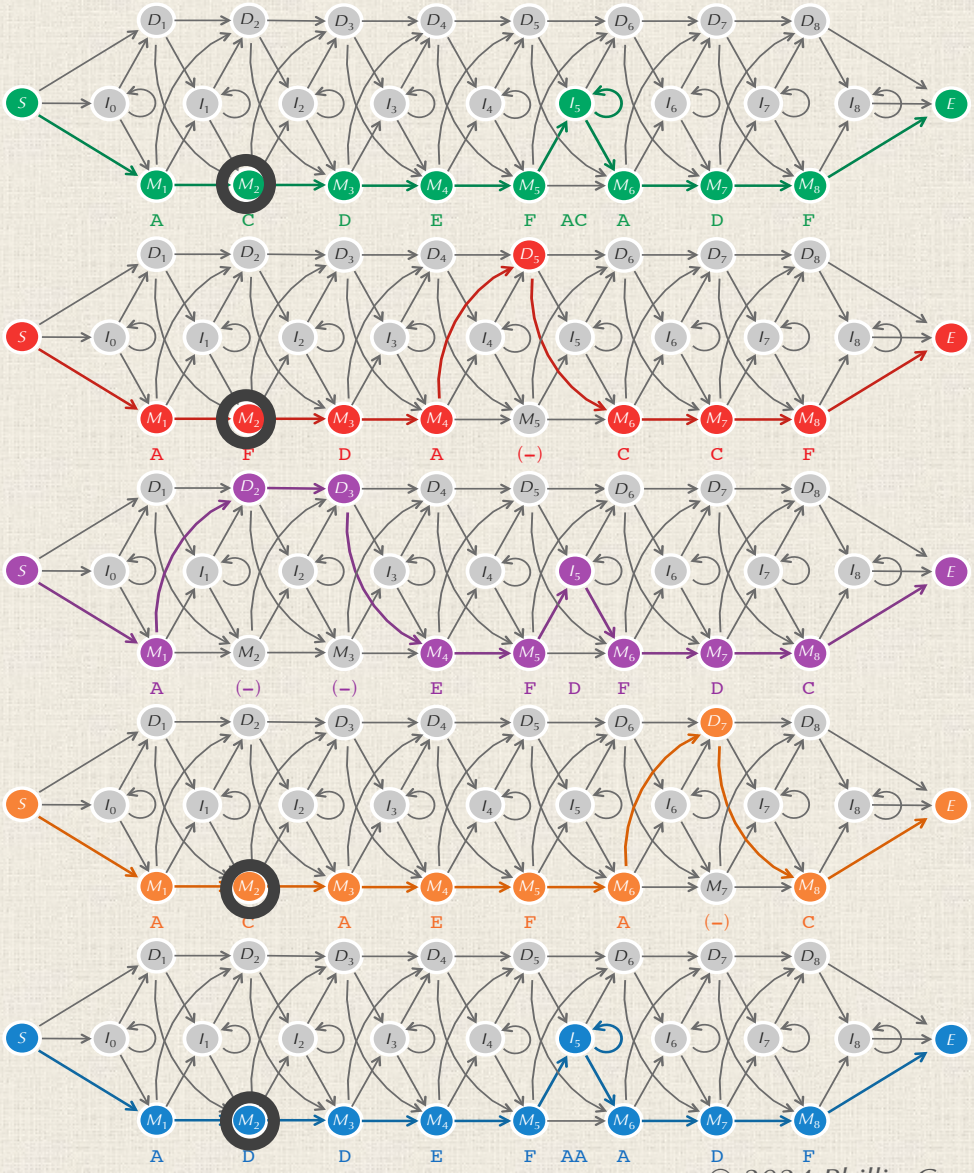
0 into D_6

$transition_{Match(5),Insertion(5)} = 3/4$

$transition_{Match(5),Match(6)} = 1/4$

$transition_{Match(5),Deletion(6)} = 0$

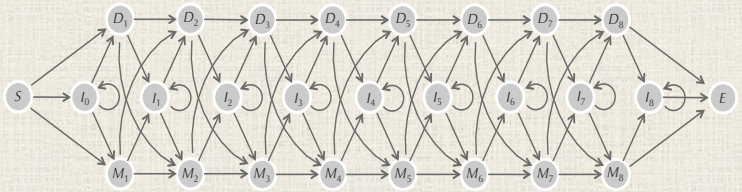
Transition Probabilities of Profile HMM



symbols emitted from M_2:
C, F, C, D

$emission_{Match(2)}(A) = 0$
$emission_{Match(2)}(C) = 2/4$
$emission_{Match(2)}(D) = 1/4$
$emission_{Match(2)}(E) = 0$
$emission_{Match(2)}(F) = 1/4$

Forbidden Transitions



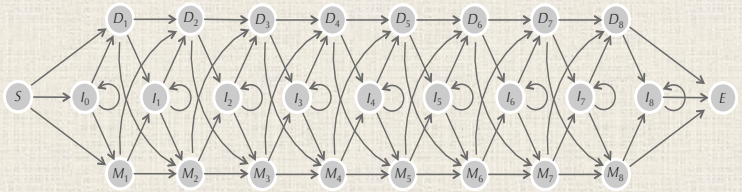
	S	I_0	M_1	D_1	I_1	M_2	D_2	I_2	M_3	D_3	I_3	M_4	D_4	I_4	M_5	D_5	I_5	M_6	D_6	I_6	M_7	D_7	I_7	M_8	D_8	I_8	E
S			1																								
I_0																											
M_1						.8	.2																				
D_1																											
I_1																											
M_2									1																		
D_2																											
I_2																											
M_3												1															
D_3																											
I_3												1															
M_4															.8	.2											
D_4																											
I_4																											
M_5																	.25	.75									
D_5																	.33	.67									
I_5																		1									
M_6																					.8	.2					
D_6																											
I_6																											
M_7																								1			
D_7																											
I_7																								1			
M_8																											1
D_8																											
I_8																											
E																											

Gray cells: edges in the HMM diagram.

Clear cells: *forbidden* transitions.

Empty gray cells: equal to zero.

Forbidden Transitions

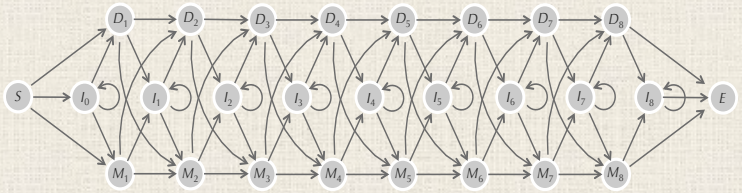


	S	I_0	M_1	D_1	I_1	M_2	D_2	I_2	M_3	D_3	I_3	M_4	D_4	I_4	M_5	D_5	I_5	M_6	D_6	I_6	M_7	D_7	I_7	M_8	D_8	I_8	E
S			1																								
I_0																											
M_1						.8	.2																				
D_1																											
I_1																											
M_2									1																		
D_2																											
I_2																											
M_3												1															
D_3																											
I_3												1															
M_4															.8	.2											
D_4																											
I_4																											
M_5																	.25	.75									
D_5																	.33	.67									
I_5																		1									
M_6																				.8	.2						
D_6																											
I_6																											
M_7																							1				
D_7																											
I_7																							1				
M_8																											1
D_8																											
I_8																											
E																											

Having zero weights will cause issues for two reasons:

1. log(0) is undefined.
2. One weight being zero shouldn't disqualify a path.

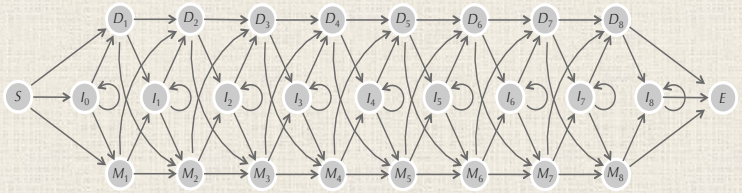
Forbidden Transitions



	S	I_0	M_1	D_1	I_1	M_2	D_2	I_2	M_3	D_3	I_3	M_4	D_4	I_4	M_5	D_5	I_5	M_6	D_6	I_6	M_7	D_7	I_7	M_8	D_8	I_8	E
S			1																								
I_0																											
M_1						.8	.2																				
D_1																											
I_1																											
M_2									1																		
D_2																											
I_2																											
M_3												1															
D_3																											
I_3												1															
M_4															.8	.2											
D_4																											
I_4																											
M_5																	.25	.75									
D_5																	.33	.67									
I_5																		1									
M_6																					.8	.2					
D_6																											
I_6																											
M_7																								1			
D_7																											
I_7																								1			
M_8																											1
D_8																											
I_8																											
E																											

STOP: What should we do?

Forbidden Transitions



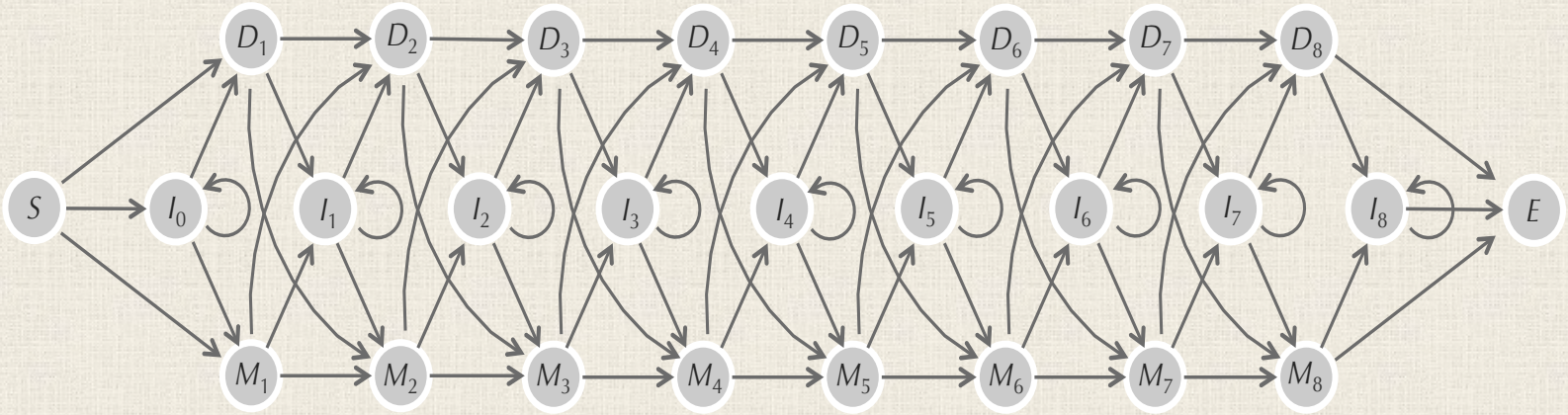
	S	I_0	M_1	D_1	I_1	M_2	D_2	I_2	M_3	D_3	I_3	M_4	D_4	I_4	M_5	D_5	I_5	M_6	D_6	I_6	M_7	D_7	I_7	M_8	D_8	I_8	E
S			1																								
I_0																											
M_1						.8	.2																				
D_1																											
I_1																											
M_2									1																		
D_2																											
I_2																											
M_3												1															
D_3																											
I_3												1															
M_4															.8	.2											
D_4																											
I_4																											
M_5																	.25	.75									
D_5																	.33	.67									
I_5																		1									
M_6																				.8	.2						
D_6																											
I_6																											
M_7																							1				
D_7																											
I_7																							1				
M_8																											1
D_8																											
I_8																											
E																											

STOP: What should we do?

Answer: Add pseudocounts (!) to the zero values and normalize.

CLASSIFYING PROTEINS WITH PROFILE HMMS

Aligning a Protein Against a Profile HMM

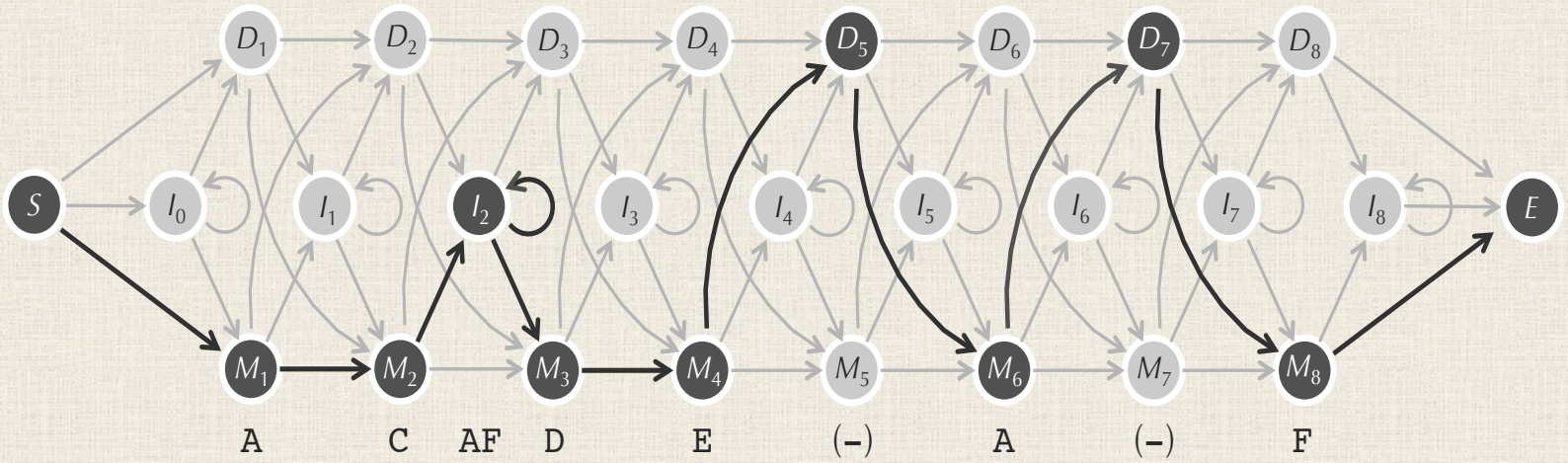


Alignment

A	C	–	–	D	E	F	AC	A	D	F
A	F	–	–	D	A	–	–	C	C	F
A	–	–	–	–	E	F	D	–	F	D	C
A	C	–	–	A	E	F	–	–	A	–	C
A	D	–	–	D	E	F	AA	A	D	F

Protein **ACAFDEAF**

Aligning a Protein Against a Profile HMM



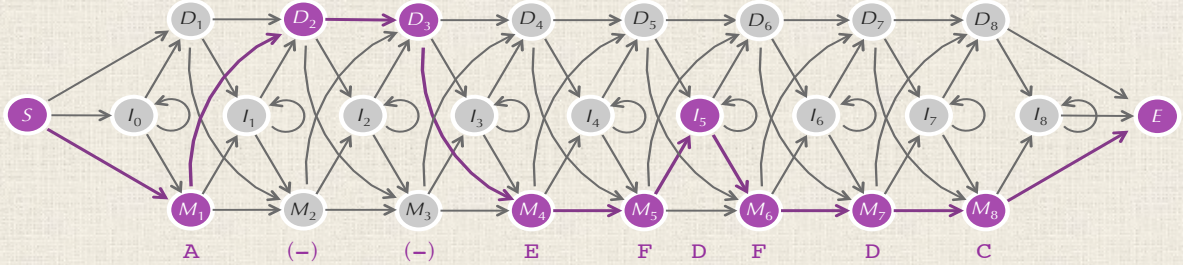
Alignment

A	C	– –	D	E	F	AC	A	D	F
A	F	– –	D	A	–	– –	C	C	F
A	–	– –	–	E	F	D–	F	D	C
A	C	– –	A	E	F	– –	A	–	C
A	D	– –	D	E	F	AA	A	D	F

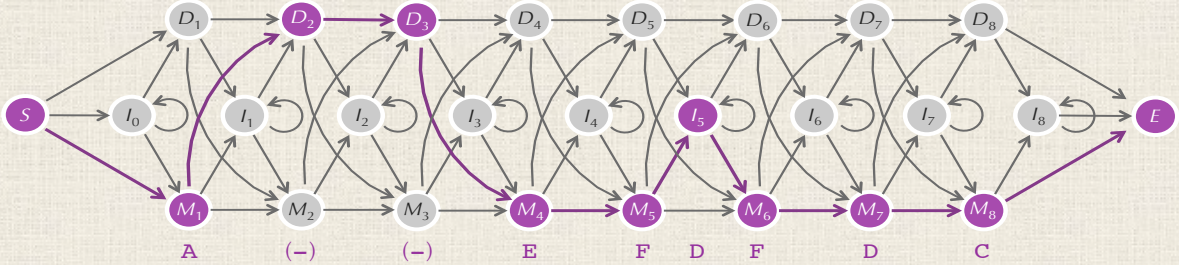
Protein **ACAFDEAF**

Apply Viterbi algorithm to find optimal hidden path.

Profile HMM diagram

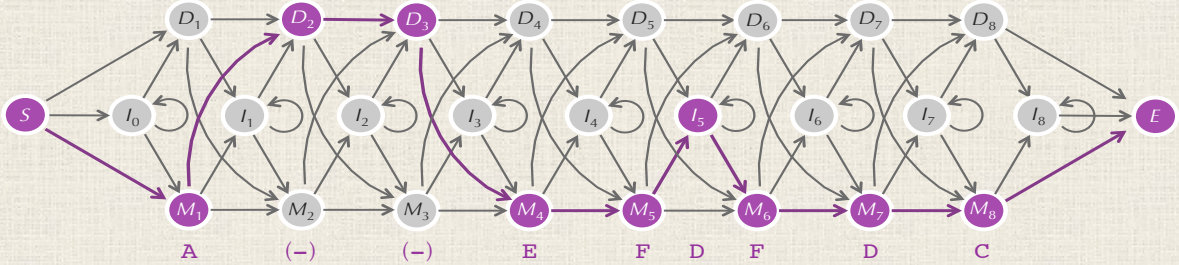


Profile HMM diagram



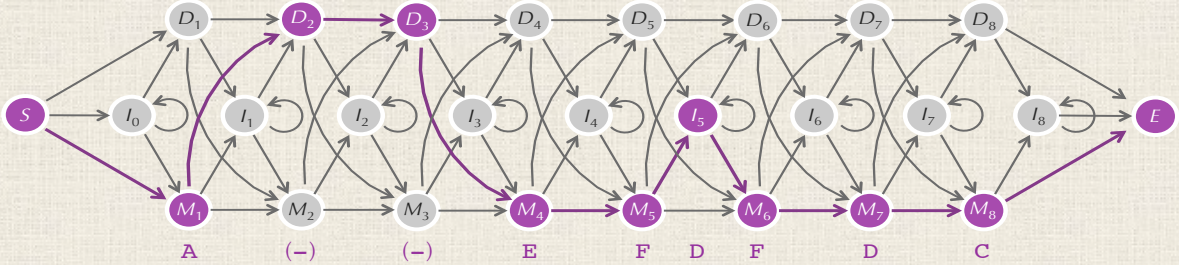
STOP: How many rows and columns does the Viterbi graph of this profile HMM have?

Profile HMM diagram

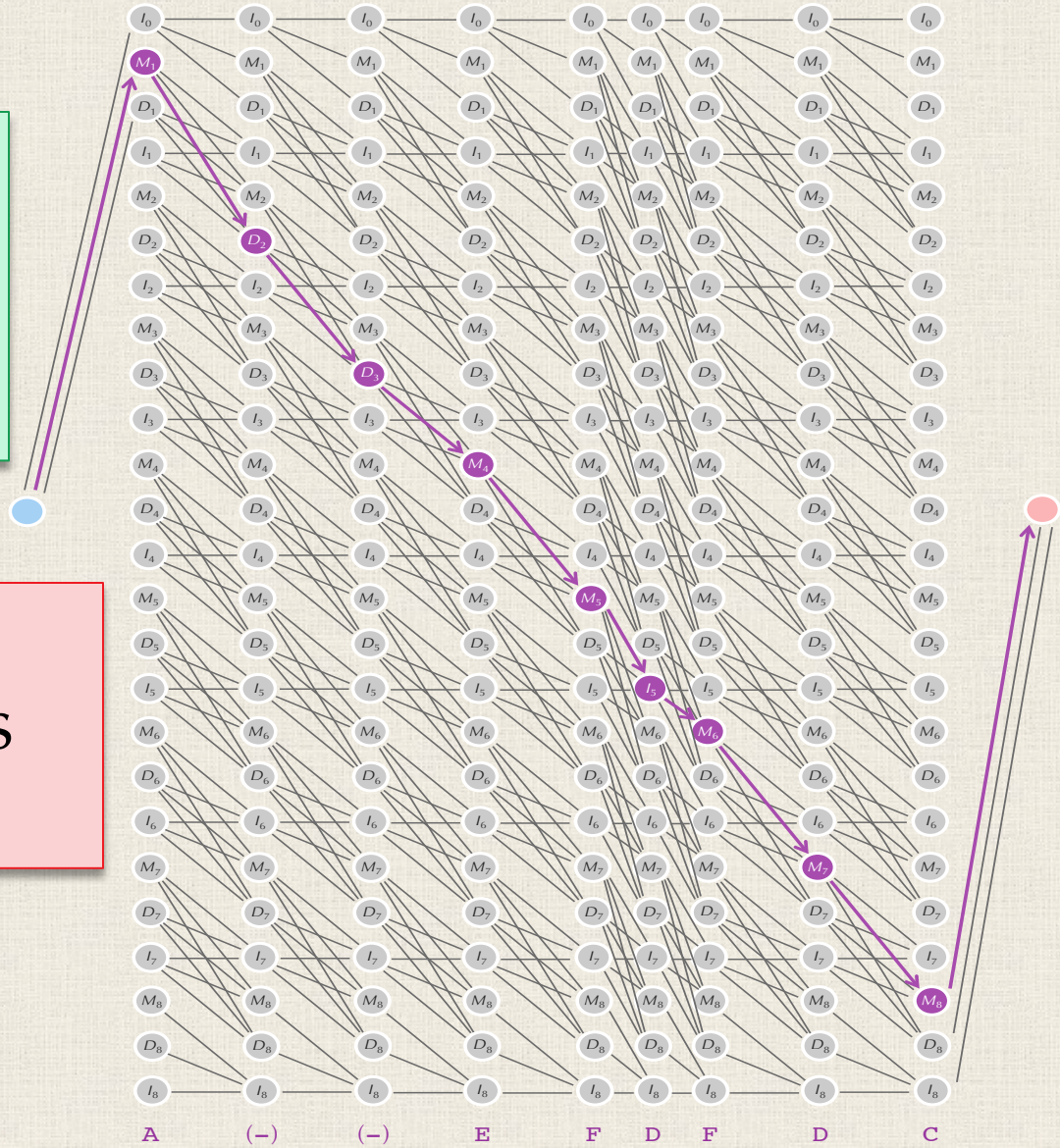


Viterbi graph of profile HMM:
#columns=
#visited states

Profile HMM diagram

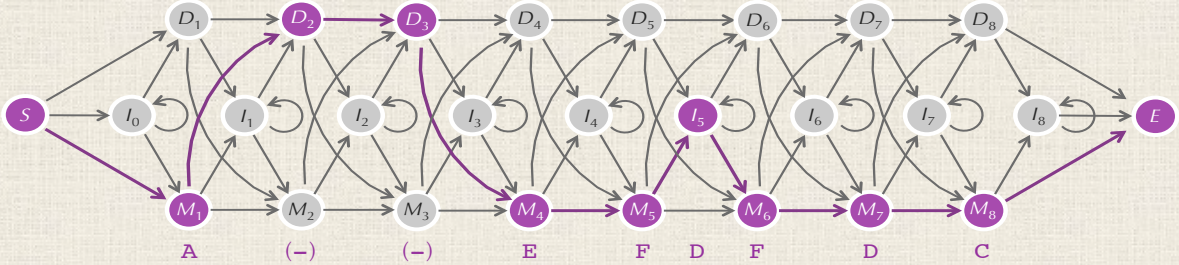


Viterbi graph of profile HMM:
#columns=
#visited states



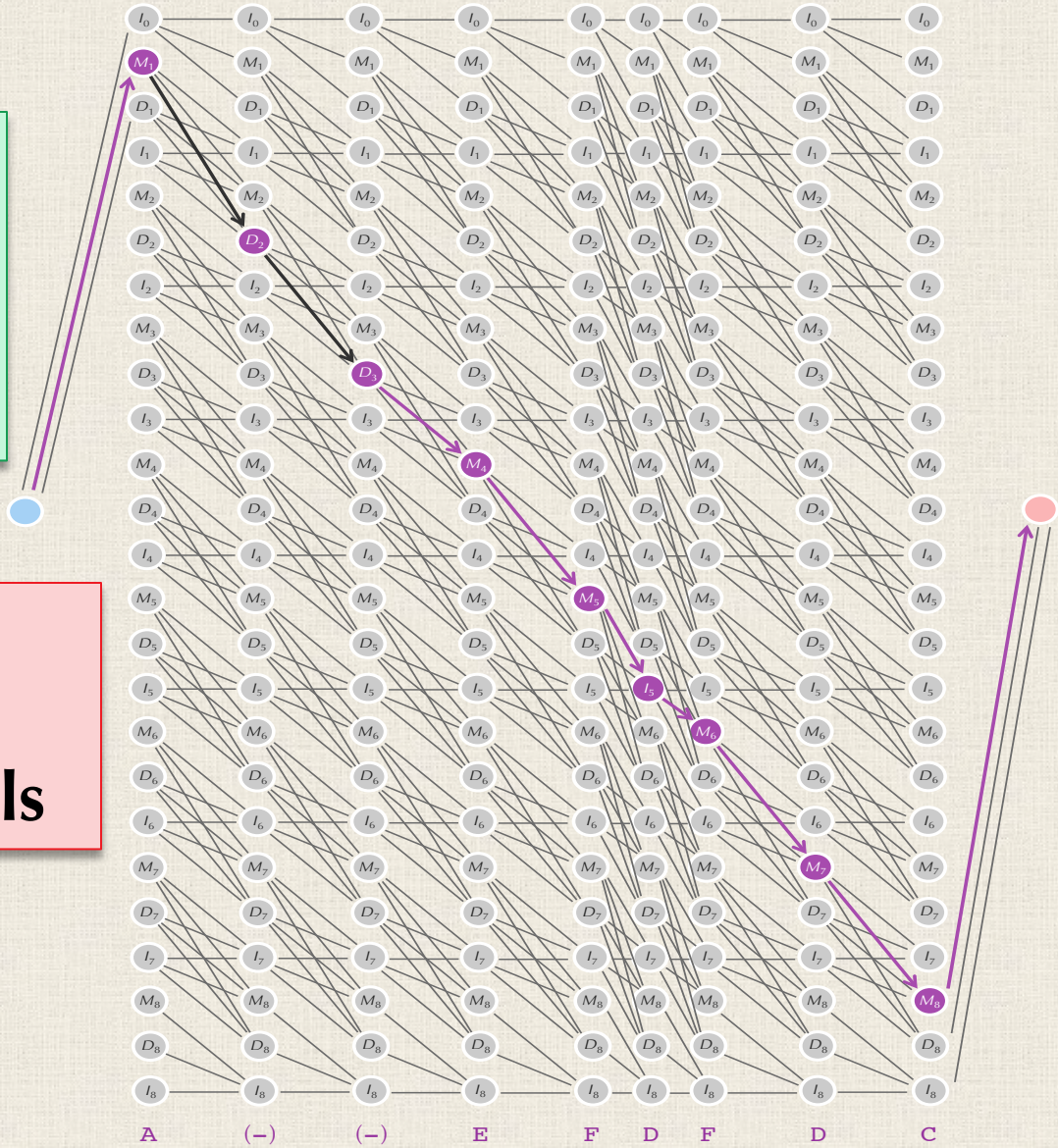
STOP: What is wrong with this Viterbi graph?

Profile HMM diagram

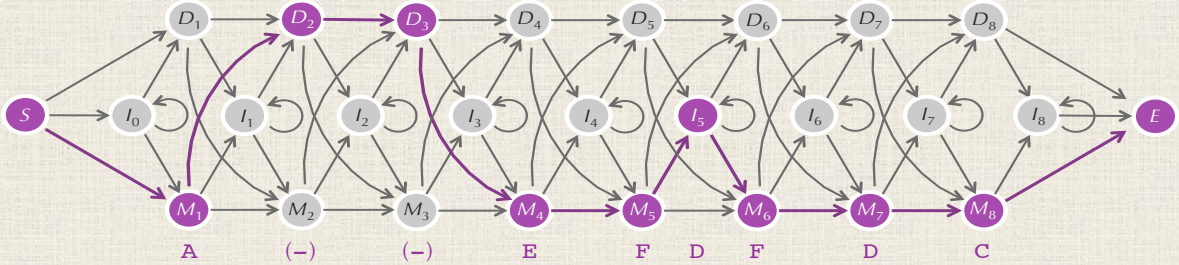


Viterbi graph of profile HMM:
#columns=
#visited states

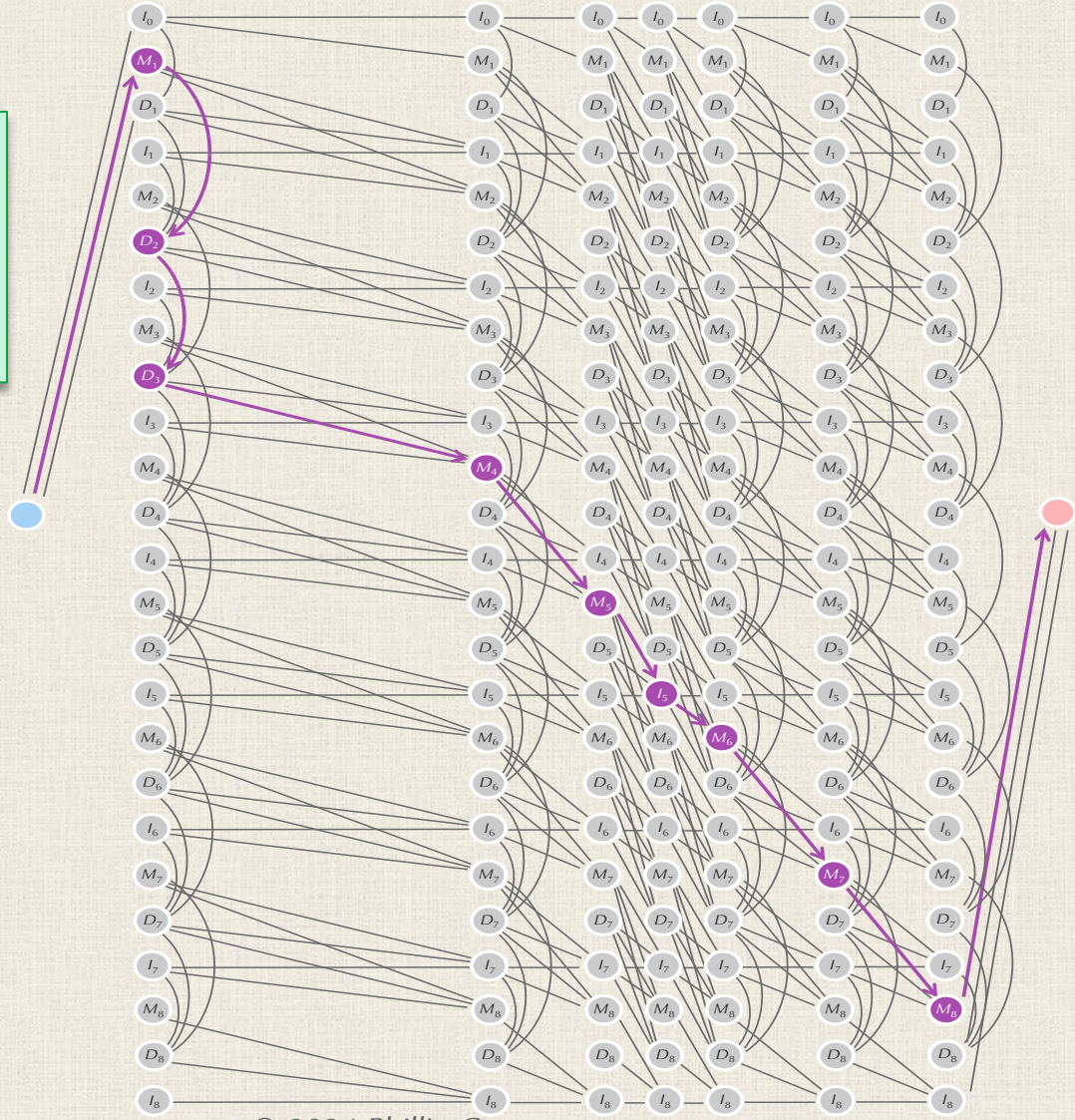
By definition,
#columns =
#emitted symbols



Profile HMM diagram

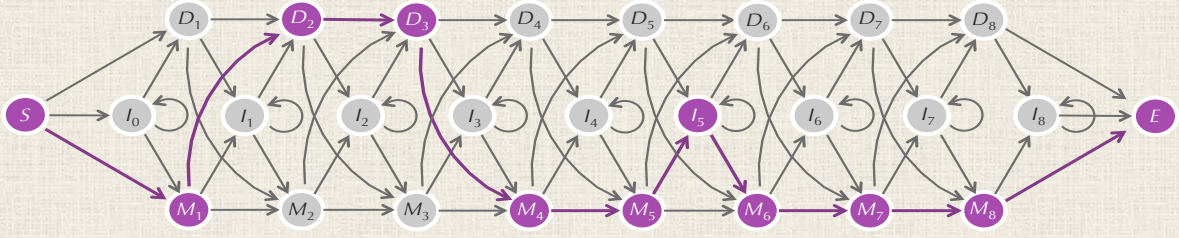


Nearly correct Viterbi graph of profile HMM:

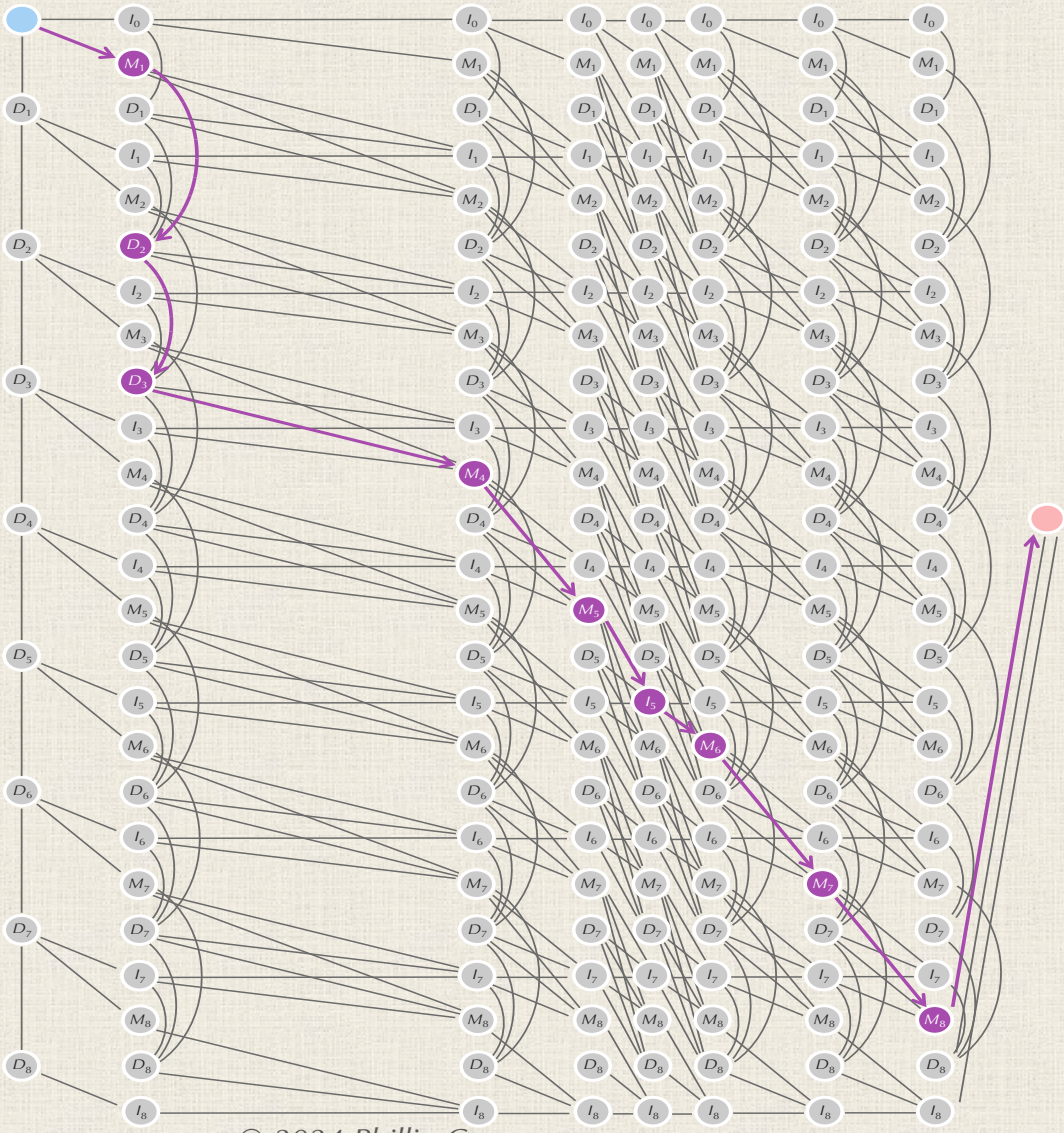


Vertical edges enter "silent" deletion states

Profile HMM diagram

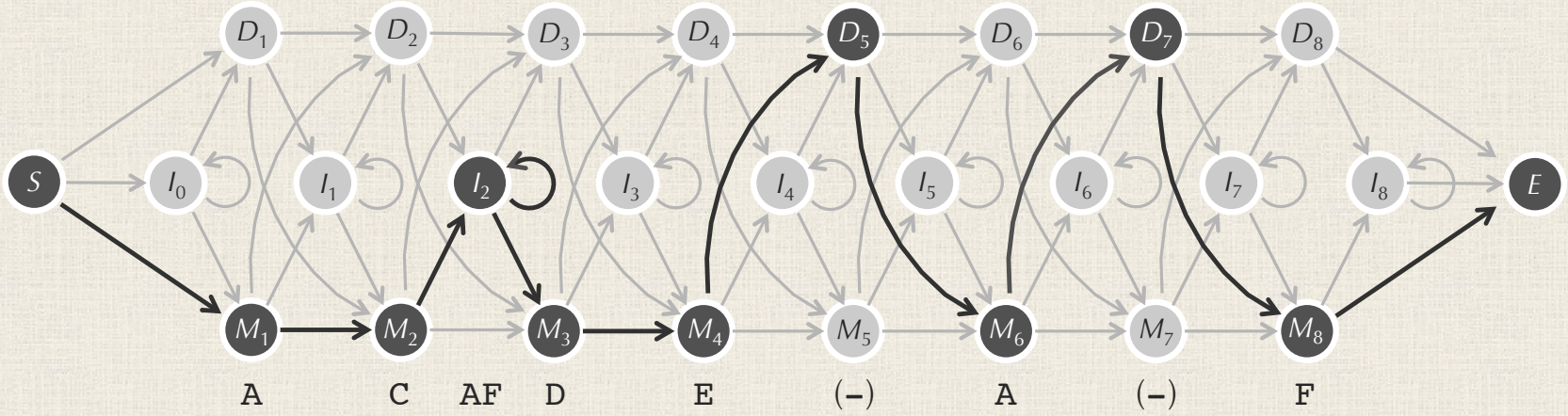


Correct Viterbi graph of profile HMM:



Adding 0-th column that contains only silent states

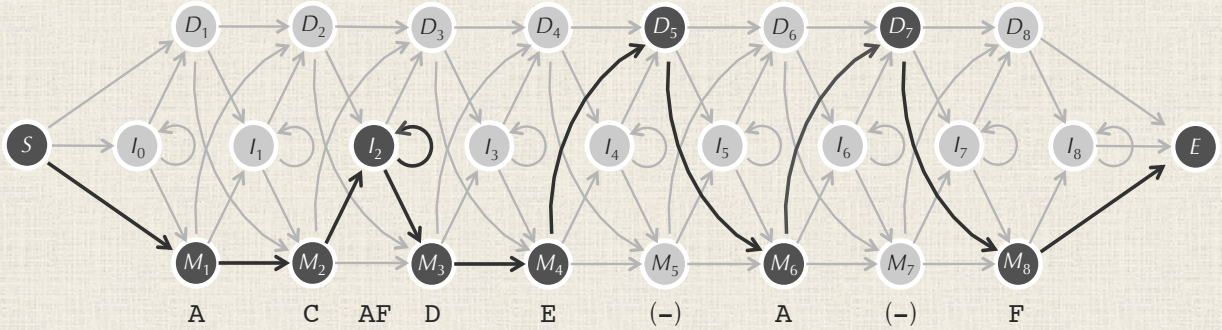
Alignment with a Profile HMM

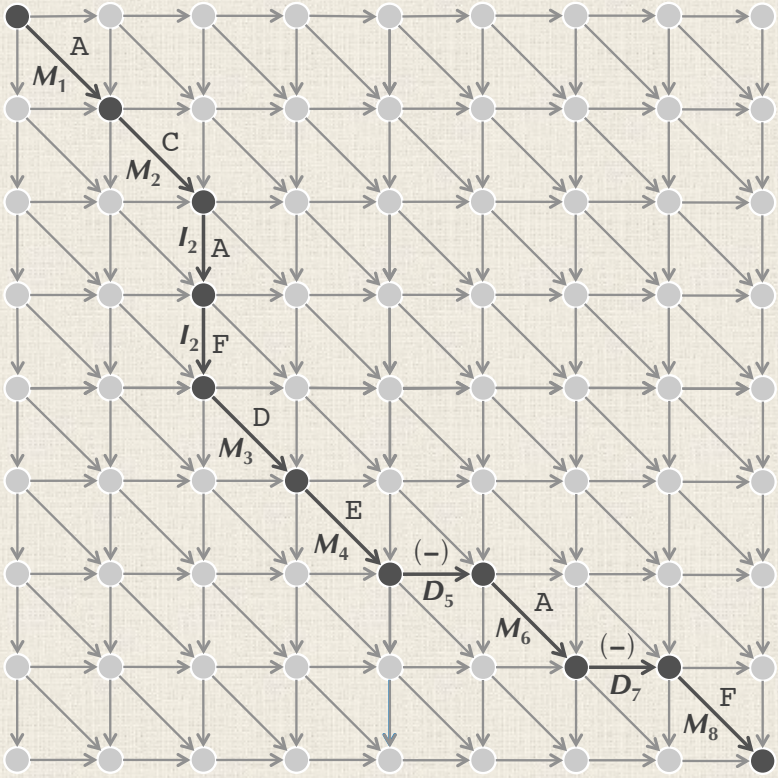


Sequence Alignment with Profile HMM Problem: *Align a new sequence to a family of aligned sequences using a profile HMM.*

- **Input:** A multiple alignment *Alignment*, a string *Text*, a threshold θ (maximum fraction of insertions per column), and a pseudocount σ.
- **Output:** An optimal hidden path emitting *Text* in the profile HMM *HMM*(*Alignment*, θ, σ).

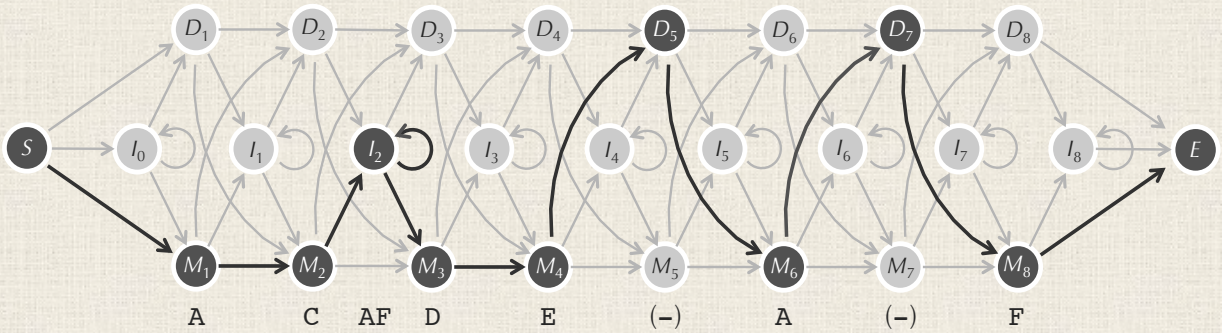
Have I Wasted Your Time?

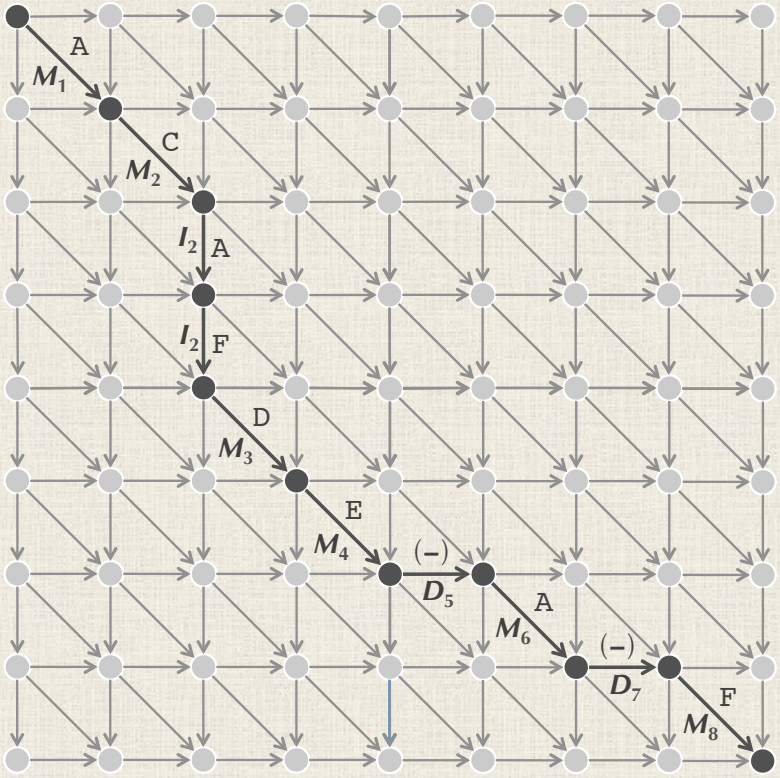




STOP: A path through the profile HMM diagram looks like a lot like a path through an alignment graph! So what is different?

I Hope Not! ☺





Key point: The choice of alignment path is now based on transition and emission probabilities that *vary* from one column to the next.

$$s_{M(j),i} = \max \begin{cases} s_{I(j\text{-}1),i\text{-}1} * weight(I(j\text{-}1),M(j),i\text{-}1) \\ s_{D(j\text{-}1),i\text{-}1} * weight(D(j\text{-}1),M(j),i\text{-}1) \\ s_{M(j\text{-}1),i\text{-}1} * weight(M(j\text{-}1),M(j),i\text{-}1) \end{cases}$$

Three levels of language understanding

Level 1: substitution of one word for another is always treated the same.

Three levels of language understanding

Level 1: substitution of one word for another is always treated the same.

Level 2: word substitutions are treated differently depending on context.

Three levels of language understanding

Level 1: substitution of one word for another is always treated the same.

Level 2: word substitutions are treated differently depending on context.

Level 3: a complete understanding of the language, allowing us to form new sentences with custom meanings.

Three levels of ***protein*** understanding

Level 1: substitution of one ***amino acid*** for another is always treated the same.

Scoring Matrices

Level 2: word substitutions are treated differently depending on context.

Level 3: a complete understanding of the language, allowing us to form new sentences with custom meanings.

Three levels of ***protein*** understanding

Level 1: substitution of one ***amino acid*** for another is always treated the same.

Scoring Matrices

Level 2: ***amino acid*** substitutions are treated differently depending on context.

HMMs

Level 3: a complete understanding of the language, allowing us to form new sentences with custom meanings.

Three levels of ***protein*** understanding

Level 1: substitution of one ***amino acid*** for another is always treated the same.

Scoring Matrices

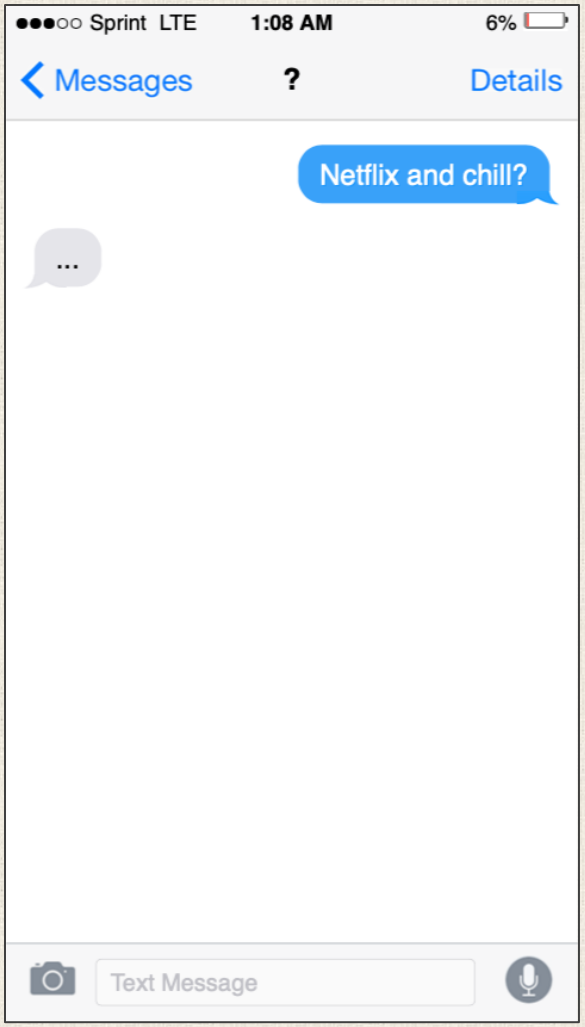
Level 2: ***amino acid*** substitutions are treated differently depending on context.

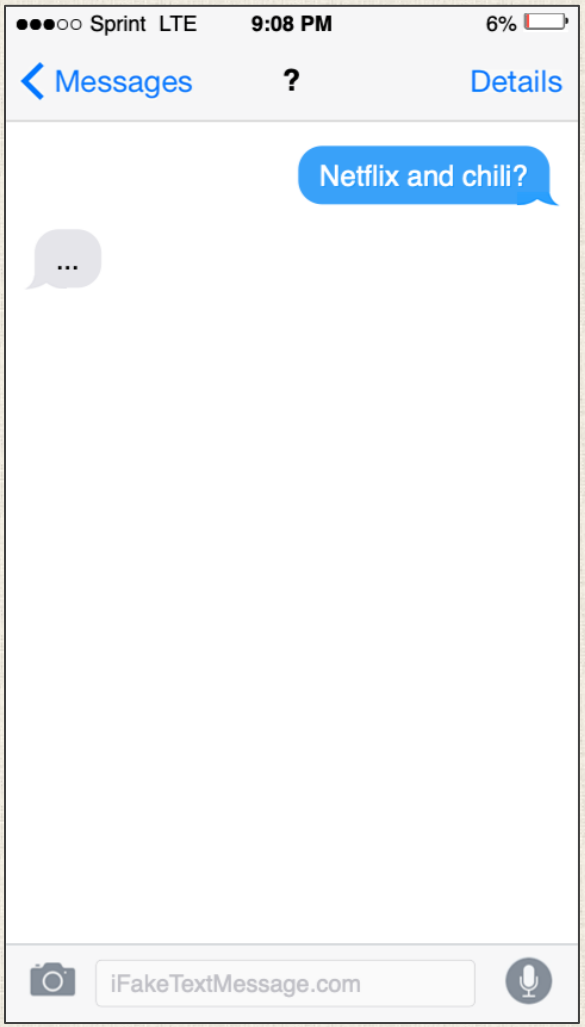
HMMs

Level 3: a complete understanding of the language, allowing us to form new ***proteins*** with custom meanings.

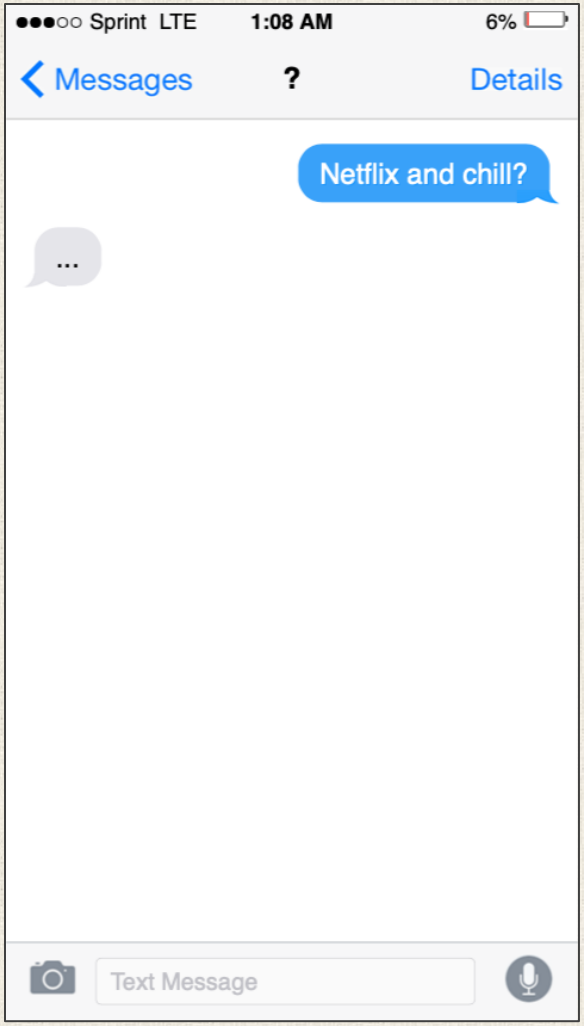
???

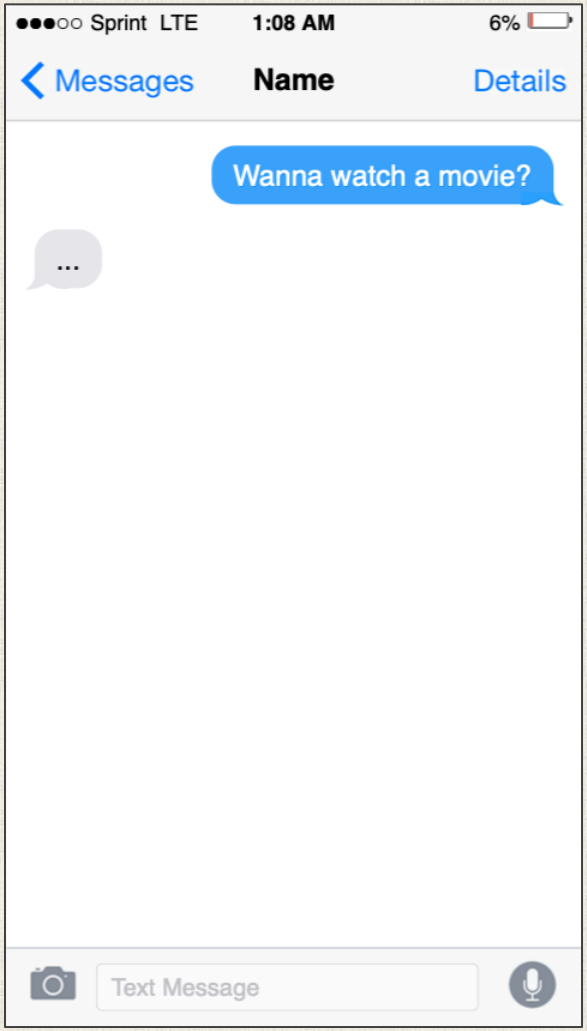
Changing just one letter can produce a huge change in meaning …





... yet sentences can have the *same* meaning but completely *different* words!





In proteins, a single mutation can cause enormous structural changes …

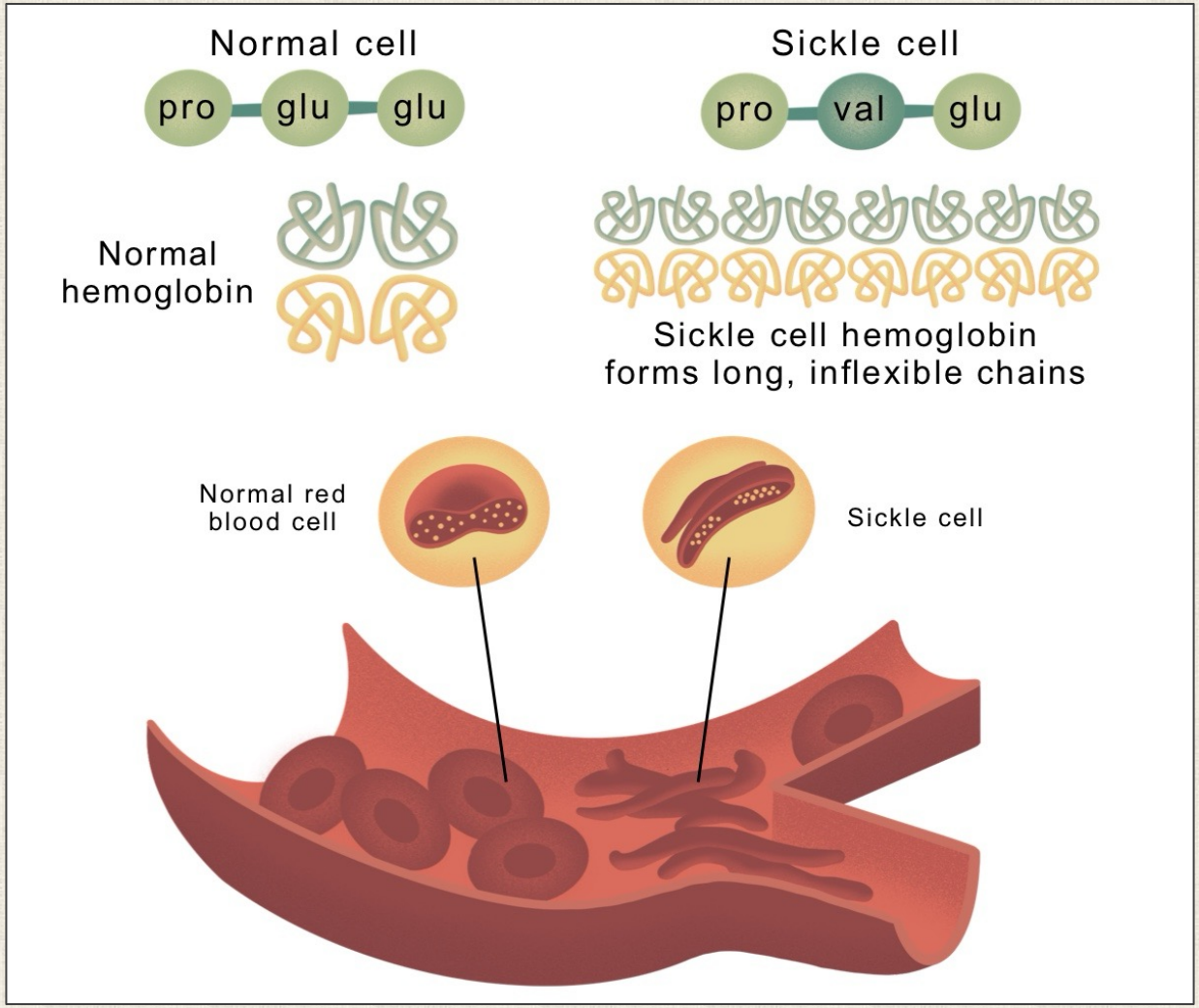
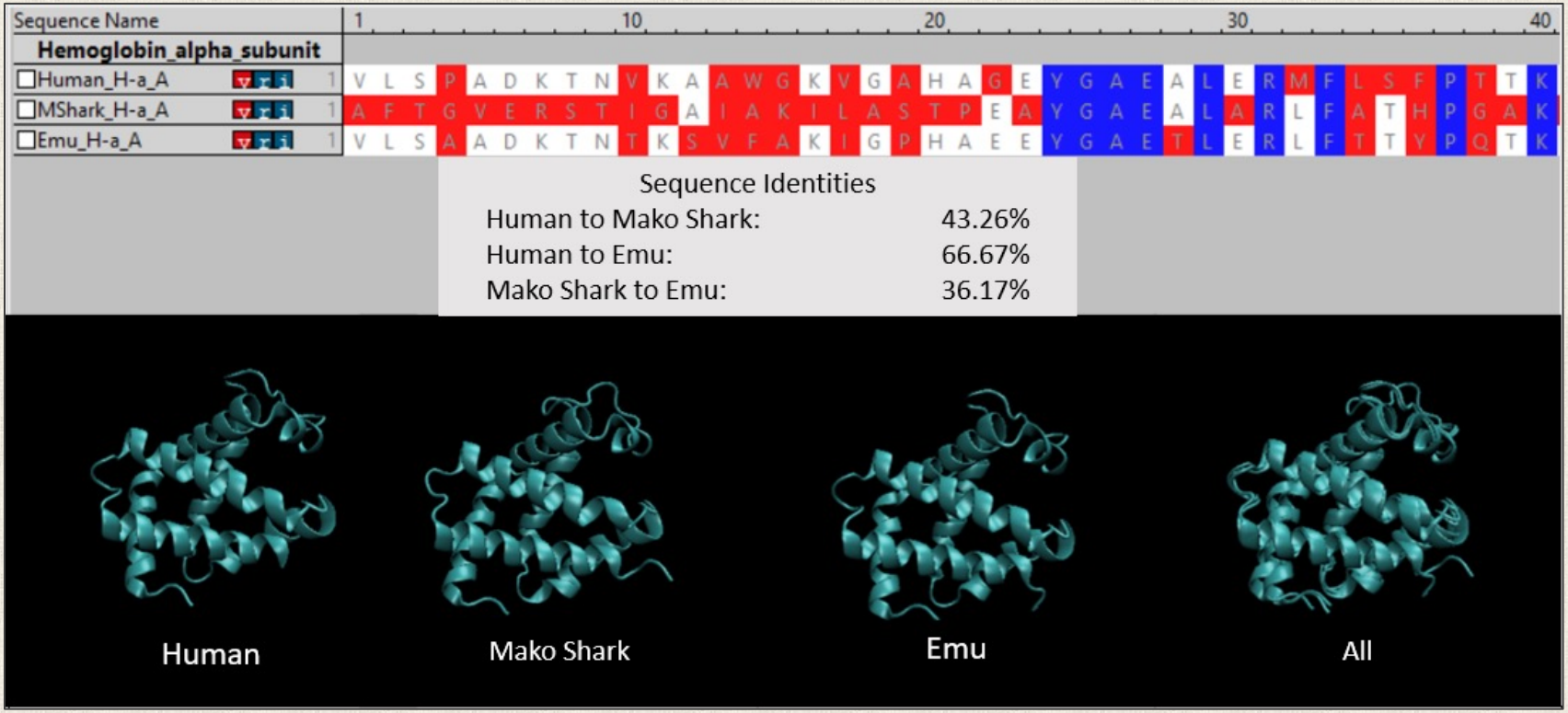


Image courtesy: Sickle-Cell.com

… and yet we already know that similar structures have very different sequences!



So how can we improve on HMMs?

Idea 1 (later in this course): compare proteins not at the level of sequence, but as three-dimensional *structures*.

So how can we improve on HMMs?

Idea 1 (later in this course): compare proteins not at the level of sequence, but as three-dimensional *structures*.

Idea 2 (unsolved problem in biology): train AI (e.g., LLMs) to understand the "language" of proteins and how sequence → structure → protein function

Citations

Biological sequence analysis

by R Durbin · Cited by 7614 — **Biological sequence analysis**. **Probabilistic models of proteins and nucleic acids**. Richard Durbin. Sean R. Eddy. Anders Krogh. Graeme Mitchison…
366 pages

HMMER web server: interactive sequence similarity searching

RD Finn, J Clements, SR Eddy - Nucleic acids research, 2011 - academic.oup.com

… **HMMER** is a software suite for protein sequence similarity searches using probabilistic methods. Previously, **HMMER** has mainly been … A **HMMER** web server (http://**hmmer**.janelia.org) has been designed and implemented such that most protein database searches return …

☆ Save Cite Cited by 3707 Related articles All 17 versions