Plummeting Sequencing Costs Mean Millions of Human Genomes

Cost per Raw Megabase of DNA Sequence

Moore’s Law

NIH National Human Genome Research Institute

gene.gov/sequencingcosts

© 2024 Phillip Compeau
Recall: Overview of Genome Sequencing

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads

(Computational)

Assemble the genome using overlapping reads

(Computational)

AGAATATCA

TGAGAATAT

GAGAATATC

AGAATATCA

GAGAATATC

TGAGAATAT

...TGAGAATATCA...
Recall: Overview of Genome Sequencing

But why would we do this if we already have a good sense of the human genome?

Assemble the genome using overlapping reads

(Computational)

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...
Recall: Overview of Genome Sequencing

But why would we do this if we already have a good sense of the human genome?

Analogy: Would you assemble a jigsaw puzzle without looking at the photo on the box?

Assemble the genome using overlapping reads
(Computational)
As different as we may seem, we are genetically very similar

Two humans share ~99.9% of their genomes. The rest is typically single-nucleotide variations.
As different as we may seem, we are genetically very similar.

**Single-nucleotide polymorphism (SNP):** One-nucleotide mutation present in at least 1% of humans.
A First Attempt at Read Mapping

**Read-mapping:** Comparison of sequencing reads' patterns from an individual against a reference human genome's text stored in a database.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT
GAGGA  C CACG  TGA-A
A First Attempt at Read Mapping

**Read-mapping**: Comparison of sequencing reads Patterns from an individual against a reference human genome Text stored in a database.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTTGAGGACCCACGTGA-A

**STOP**: How could we map reads against a reference genome?
A First Attempt at Read Mapping

**Read-mapping:** Comparison of sequencing reads *Patterns* from an individual against a **reference human genome** *Text* stored in a database.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT
GAGGA CCACG TGA-A

**Idea:** We could **align** each read against the reference genome. But what problem are we solving?
Recall these Two Problems from Text

A **fitting alignment** of \( v \) and \( w \) is an alignment of a substring of \( v \) against all of \( w \).

**Fitting Alignment Problem:**
- **Input:** Two strings and a scoring matrix.
- **Output:** A fitting alignment of the strings with maximum alignment score according to the scoring matrix.

<table>
<thead>
<tr>
<th>Global</th>
<th>Local</th>
<th>Fitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGTAGGCTTAAAGGTTA</td>
<td>CGTAGGCTTAAAGGTTA</td>
<td>CGTAGGCTTAAAGGTTA</td>
</tr>
<tr>
<td>A−TAG-----A-----T−A</td>
<td>ATAGATA</td>
<td>ATAGA--TA</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
A First Attempt at Read Mapping

**Read-mapping:** Comparison of sequencing reads *Patterns* from an individual against a reference human genome *Text* stored in a database.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT
GAGGA   CCACG   TGA-A

**Problem:** Fitting alignment has runtime $O(|Pattern| \times |Text|)$ for each string *Pattern* in our reads – too long!
Remember: The First *Full* Human Genome Wasn’t Sequenced Until 2020!
Reference Genomes are Not Diverse

When we “map” reads against a reference human genome, the most commonly used reference is 70% made up of RP11, or “some guy from Buffalo”.

The first complete human reference is taken from a single sperm. This reference is quickly being replaced by “CHM13”, the first ever complete human genome.

CHM13 is a cell line grown from a “molar pregnancy”, an empty egg fertilized by a sperm. Both copies of chromosomes come from this sperm.
Long, Cheap Reads Give Hope of a "Pan Genome" Era

STOP: Why might mapping reads from one ethnicity against a genome from a different ethnicity lead to biased conclusions?
INTRODUCTION TO MULTIPLE PATTERN MATCHING
Simplifying Assumption: No Mutations

We will first find all the exact matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.
We will first find all the exact matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.

**Key Point:** This lecture will assume that we are working with highly accurate reads rather than messy reads.
We will first find all the exact matches of reads in the reference. If we are looking for variants, then we can eliminate from consideration areas of the reference genome where exact matches occur.

**Multiple Pattern Matching Problem:** Find all occurrences of a collection of patterns in a text.

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.
STOP: What is the probability of a randomly selected 250-mer matching in two genomes with 99.9% similarity? (Assume mutations are uniform.)

**Multiple Pattern Matching Problem:** Find all occurrences of a collection of patterns in a text.

- **Input:** A string `Text` and a collection `Patterns` containing (shorter) strings.
- **Output:** All starting positions in `Text` where a string from `Patterns` appears as a substring.
Wait: Isn’t This Too Simple?

**Answer:** We need all 250 positions to match, so the probability is \((1-0.001)^{250} = 0.779\). So nearly 80% of 250-mers will match in two 99.9% similar genomes.

**Multiple Pattern Matching Problem:** Find all occurrences of a collection of patterns in a text.

- **Input:** A string \(Text\) and a collection \(Patterns\) containing (shorter) strings.
- **Output:** All starting positions in \(Text\) where a string from \(Patterns\) appears as a substring.
Wait: Isn’t This Too Simple?

**BruteForcePatternMatching:** slide each string down `Text`, one at a time.

**Multiple Pattern Matching Problem:** *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string `Text` and a collection `Patterns` containing (shorter) strings.
- **Output:** All starting positions in `Text` where a string from `Patterns` appears as a substring.
Wait: Isn’t This Too Simple?

Multiple Pattern Matching Problem: Find all occurrences of a collection of patterns in a text.

- **Input:** A string Text and a collection Patterns containing (shorter) strings.
- **Output:** All starting positions in Text where a string from Patterns appears as a substring.

STOP: What is the Big-O runtime of BruteForcePatternMatching?
Multiple Pattern Matching Problem: Find all occurrences of a collection of patterns in a text.

• Input: A string Text and a collection Patterns containing (shorter) strings.

• Output: All starting positions in Text where a string from Patterns appears as a substring.

Answer: For one read, it is $O(|Text| \times |Pattern|)$. For multiple reads, it is $O(|Text| \times |Patterns|)$, where $|Patterns|$ is the total length of Patterns.
HERDING PATTERNS INTO A TRIE
In Brute Force, Patterns Travel One at a Time

Text

CTGATGATGGACTACGCTACTACTGCTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC

Pattern_1

Pattern_2

Pattern_3

Pattern_4

Pattern_5

© 2024 Phillip Compeau
Idea: Organize Patterns and Make a Single Pass Down the Reference Genome

Text

CTGATGATGGACTACGCTACTACTGCTAGCTGTATTACGATCAGCTACCACATCGTAGCTAC

© 2024 Phillip Compeau
**Trie:** Graph constructed from *Patterns*.
- There is a single root node with indegree 0, denoted *root*; all other nodes have indegree 1.
- Each edge of *Trie*(*Patterns*) is labeled with a letter.
- Edges leading out of a given node have distinct labels.
- Every string in *Patterns* is spelled out by concatenating the letters along some path from the root downward.
- Every path from the root to a **leaf** (node with outdegree 0) spells a string from *Patterns*. 
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
**Patterns**

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana

pan

and

antenna

bandana

ananas

nana
Patterns

banana

pan

and

nab

antenna

bandana

ananas

nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana pan and nab antenna bandana ananas nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
**bandana**
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
anananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
**ananas**
nana
Patterns
banana
pan
and
nab
antenna
bandana
**ananas**
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana

© 2024 Phillip Compeau
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
Patterns
banana
pan
and
nab
antenna
bandana
ananas
nana

© 2024 Phillip Compeau
Patterns

banana
pan
and
nab
antenna
bandana
ananas
nana
STOP: How can we use the trie for pattern matching? (e.g., say Text = “panamabanananas”).
Answer: We can test whether any string in Patterns matches a prefix of Text by ”sliding it” down the Text.
panamabananas

Pattern Positions
panamabananas

Pattern    Positions
panmabanananas

Pattern  Positions

© 2024 Phillip Compeau
We have found a pattern match – now we just need to “slide down” and start the search at the next symbol of Text.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
</tbody>
</table>
pan am a b a na na n a s

Pattern    Positions

pan       0
panamabananas
Pan amabananas

Pattern  Positions
pan      0
panamabanananas

Pattern | Positions
---|---
pan | 0
By how the trie is organized, we can be sure that there are no patterns matching this position of *Text*.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
</tbody>
</table>
panamabanananas

Pattern Positions
pan 0
Panama bananas

Pattern Positions
pan 0
pan amabanananas
panamabanananas

Pattern    Positions
pan          0
The diagram illustrates the pattern matching algorithm with the pattern "panamabananas". The root node is connected to nodes labeled with characters from the pattern. The positions of the pattern within the string are highlighted. The table to the right shows the pattern and its positions, with "pan" occurring at position 0.
panamabanananas

Pattern  Positions
pan      0
The text in the image reads "panama bananas".
panama bananas

Pattern Positions
pan 0
panama bananas

Pattern   Positions
pan        0

© 2024 Phillip Compeau
panama bananas

Pattern Positions

pan 0
panama bananas

Root

Pattern | Positions
--- | ---
pan | 0
panama bananas

Pattern   Positions
pan       0
panama bananas

Pattern Positions

pan 0
panama bananas

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
</tbody>
</table>
panamabananas

Pattern  Positions
pan      0
banana  6
panamabanananas

Pattern     Positions
pan          0
banana       6
pan amabananas

Pattern    Positions
pan         0
banana      6

© 2024 Phillip Compeau
panamabanananas

Pattern    Positions
pan        0
banana     6
anananas   7
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
</tbody>
</table>

Diagram of a suffix tree with the patterns "pan", "banana", and "ananas".
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
</tbody>
</table>
panama bananas

Pattern    Positions
pan        0
banana     6
anananas   7
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
</tbody>
</table>

panamabananas
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
</tbody>
</table>

```
panamabananas
```

```
Pattern    Positions
pan        0
banana     6
ananas     7
nana       8
```
pan am ab an an as

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
</tbody>
</table>
Pattern | Positions
---|---
pan  | 0
banana | 6
ananas | 7
nana | 8

© 2024 Phillip Compeau
**Pattern**      **Positions**

pan          0
banana      6
ananas      7
nana        8
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
</tbody>
</table>
**Pattern** | **Positions**
---|---
pan | 0
banana | 6
anananas | 7
nana | 8
pananabanananas

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
<tr>
<td>Pattern</td>
<td>Positions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>pan</td>
<td>0</td>
</tr>
<tr>
<td>banana</td>
<td>6</td>
</tr>
<tr>
<td>ananas</td>
<td>7</td>
</tr>
<tr>
<td>nana</td>
<td>8</td>
</tr>
</tbody>
</table>
What if one pattern is a prefix of another?

STOP: If we add the pattern “band” to our dataset, why would it cause problems?
What if one pattern is a prefix of another?

STOP: If we add the pattern “band” to our dataset, why would it cause problems?

Answer: “band” is a prefix of “bandana” and would not end at a leaf node …
Adding an “End” Symbol Fixes the Issue

Patterns
ban$
banana$
band$
bandana$
Success?

Runtime of Brute Force: $O(|\text{Text}| \times |\text{Patterns}|)$
Success?

Runtime of Brute Force: $O(|Text| \times |Patterns|)$

**STOP:** What is the running time of multiple pattern matching with a trie?
Runtime of Brute Force: $O(|Text| \times |Patterns|)$

STOP: What is the running time of multiple pattern matching with a trie?

Runtime of Pattern Matching with Trie:
- Trie Construction: $O(|Patterns|)$
- Pattern Matching: $O(|Text| \times |LongestPattern|)$
But We Neglected Memory ...

Our trie has 30 edges (38 when we add the \$ symbol), and \(|Patterns| = 39\).

Worst case: Trie takes up \(O(|Patterns|)\), which for a human sequencing project may be \(~1\, \text{TB}\)!
PREPROCESSING THE GENOME INSTEAD
If $|Patterns| > |Text|$, then why not try to construct a data structure from $Text$?

This would take less memory and allow us to “teleport” each string $Pattern$ to its occurrences in $Text$. 
**Suffix trie:** Trie formed from the *suffixes* of *Text*.

**Note:** We augment *Text* with a $ to mark the end of the string (why?)
panamabananas
panamabananas
banana
banana
Root

nana
STOP: How can we know where this pattern occurs in Text?
Answer: At each leaf, we add the starting position in Text of suffix ending at the leaf.
panamabananas$
panamabanas$

© 2024 Phillip Compeau
panamabananas
pan am a b a n a n a s $
The image contains a tree diagram with the text "panamabanananas$" at the top. The tree is constructed with characters branching out from the root node, each character connected to the next in a sequence. The numbers 0 to 8 are placed at the bottom nodes, possibly indicating levels or paths in the tree.
panamabananas$
panamabana\text{n\text{as}}$

© 2024 Phillip Compeau
$ \text{panamabanananas} $
Once we find a match, we walk down to the leaves to find where it occurs in Text.
STOP: How much runtime and memory does the suffix trie take to construct?
Constructing a trie for a collection of *Patterns* takes \(O(|Patterns|)\) runtime and memory.

Constructing a suffix trie takes \(O(Suffixes(Text)) = O(|Text|^2)\) runtime and memory.

### Suffixes
- panamabanananas$
- anamabanananas$
- namabanananas$
- amabanananas$
- mabanananas$
- abanananas$
- bananas$
- ananas$
- nanas$
- anas$
- nas$
- as$
- s$
- $
SUFFIX TREES
Compressing the Suffix Trie

The suffix trie will only offer a memory improvement if $|Text|^2 < |Patterns|$ (i.e., for very short genomes and many reads). But we can compress the suffix trie into a smaller structure.
Compressing the Suffix Trie

The suffix trie will only offer a memory improvement if $|Text|^2 < |Patterns|$ (i.e., for very short genomes and many reads). But we can compress the suffix trie into a smaller structure.

**Suffix Tree:** Formed by combining the edges on any maximal non-branching path of the suffix trie into a single edge.
bananas
mabanas
mabanas
nas

© 2024 Phillip Compeau
Linear pattern matching algorithms - IEEE Conference ...

by P Weiner · 1973 · Cited by 2270 · Related articles

A linear time algorithm for obtaining a compacted version of a bi-tree associated with a given string is presented. ... With this construction as the basic tool, we indicate how to solve several pattern matching problems, including some from [4] in linear time.
No Kitten!

Algorithm of the Year! (1973)

**Theorem:** The suffix tree of $Text$ has:

1. exactly $|Text| + 1$ leaves (degree 1)
2. at most $|Text|$ internal nodes (larger degree).
The suffix tree of Text has:
1. exactly $|Text| + 1$ leaves (degree 1)
2. at most $|Text|$ internal nodes (larger degree).

Thus, the suffix tree only takes $O(|Text|)$ space. (It is also possible to construct the suffix tree in just $O(|Text|)$ time; that’s a story for another day.)
STOP: But wait ... we are still storing all the suffixes! So why are suffix trees more memory efficient than suffix tries?
STOP: But wait ... we are still storing all the suffixes! So why are suffix trees more memory efficient than suffix tries?

Answer: We can replace each edge with two integers: the starting position of the string, and its length (constant total memory).
Runtime and Memory Analysis

Runtime:
- $O(|Text|)$ to construct the suffix tree directly.
- $O(|Patterns|)$ to find pattern matches.
- Total: $O(|Text| + |Patterns|)$

Memory:
- $O(|Text|)$ to construct the suffix tree directly.
- $O(|Text|)$ to store the suffix tree.
- Total: $O(|Text|)$
We are trained to see an algorithm with $O(|Text|)$ runtime and memory and celebrate.
We are trained to see an algorithm with $O(|\text{Text}|)$ runtime and memory and celebrate.

But a data structure with $2^*|\text{Text}|$ memory is very different from one with $1000^*|\text{Text}|$ memory.
We are trained to see an algorithm with $O(|Text|)$ runtime and memory and celebrate.

But a data structure with $2*|Text|$ memory is very different from one with $1000*|Text|$ memory.

It takes ~60 GB to store a suffix tree for a 3 GB human genome (4 bytes for each edge of tree). Can we do better?
SUFFIX ARRAYS
Fun Time! *The Clock Game*

https://www.youtube.com/watch?v=oc9H8bo8yg0
Suffix Array of “panamabananas$”

20 years go by...

**Suffix array:** Starting positions of sorted suffixes of *Text.*

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
20 years go by...

**Suffix array:** Starting positions of sorted suffixes of *Text*.

Can be constructed in linear time, and takes only about 12 GB for human genome.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
STOP: How can we use the suffix array for pattern matching?

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
STOP: How can we use the suffix array for pattern matching?

Answer: Like the Clock Game, we can pick a value, obtain “higher” or “lower”, and move the next value to the middle of the unsearched suffixes.
Suffix Array of “panamabananas$”

This approach, shown for Pattern = “ana”, is called **binary search**.

**Answer:** Like the Clock Game, we can pick a value, obtain “higher” or “lower”, and move the next value to the middle of the unsearched suffixes.
Suffix Array of “panamabananas$”

This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abanananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamanabananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>alas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamanabananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

### Sorted Suffixes

<table>
<thead>
<tr>
<th>Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

---

**Sorted Suffixes**                  **Suffix Array**

$               13$

abananas$   5

amabananas$  3

**anamabananas$**  1

ananas$   7

anas$   9

as$  11

bananas$   6

mabananas$  4

namabananas$  2

nanas$   8

nas$   10

panamabananas$  0

s$   12

© 2024 Phillip Compeau
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td><strong>anamabanananas$</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
This approach, shown for Pattern = “ana”, is called binary search.

First, we find the first occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.

Next, we find the *last* occurrence of “ana”.

### Sorted Suffixes

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abanananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$</td>
<td>2</td>
</tr>
<tr>
<td>nas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the **first** occurrence of “ana”.

Next, we find the **last** occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
This approach, shown for Pattern = “ana”, is called **binary search**.

First, we find the *first* occurrence of “ana”.  

Next, we find the *last* occurrence of “ana”.

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abanananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td><strong>anas$</strong></td>
<td><strong>9</strong></td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
This approach, shown for \textit{Pattern} = “ana”, is called \textbf{binary search}.

First, we find the \textit{first} occurrence of “ana”.

Next, we find the \textit{last} occurrence of “ana”.

\begin{table}
\begin{tabular}{|l|}
\hline
Sorted Suffixes & Suffix Array \\
\hline
$ & 13 \\
abananas$ & 5 \\
amabanananas$ & 3 \\
anamabanananas$ & 1 \\
ananas$ & 7 \\
anas$ & 9 \\
as$ & 11 \\
bananas$ & 6 \\
mabanananas$ & 4 \\
namabanananas$ & 2 \\
nanas$ & 8 \\
nas$ & 10 \\
panamabanananas$ & 0 \\
s$ & 12 \\
\hline
\end{tabular}
\end{table}
This approach, shown for Pattern = “ana”, is called **binary search**.

**STOP:** How do we know the starting locations of these three occurrences?

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
This approach, shown for Pattern = “ana”, is called **binary search**.

**Answer:** The suffix array tells us!

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
This approach, shown for Pattern = “ana”, is called **binary search**.

**STOP:** What is the runtime of the binary search approach using the suffix array?

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
This approach, shown for Pattern = “ana”, is called **binary search**.

**Answer:** Each pattern takes $O(\log(|Text|))$. 

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>13</td>
</tr>
<tr>
<td>abananas$</td>
<td>5</td>
</tr>
<tr>
<td>amamabanananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>
The Binary Search Pseudocode that 90% of Programmers Get Wrong

```
PatternMatchingSuffixArray(Text, Pattern)
    s ← SuffixArray(Text)
    minIndex ← 0
    maxIndex ← |Text| - 1
    while minIndex <= maxIndex
        midIndex ← floor((minIndex + maxIndex)/2)
        if Pattern > suffix of Text starting at s(midIndex)
            minIndex ← midIndex + 1
        else
            maxIndex ← midIndex - 1
        if Pattern matches suffix of Text starting at s(midIndex)
            first ← minIndex
        else
            return no occurrences
    minIndex ← first
    maxIndex ← |Text| - 1
    while minIndex <= maxIndex
        midIndex ← floor((minIndex + maxIndex)/2)
        if Pattern matches suffix of Text starting at s(midIndex)
            minIndex ← midIndex + 1
        else
            maxIndex ← midIndex - 1
    last ← maxIndex
    return (first, last)
```

© 2024 Phillip Compeau
Suffix Arrays are Another Example of Computational Biologists Influencing CS

**Suffix arrays: a new method for on-line string searches**

U Manber, G Myers - siam Journal on Computing, 1993 - SIAM

A new and conceptually simple data structure, called a suffix array, for on-line string searches is introduced in this paper. Constructing and querying suffix arrays is reduced to a sort and search paradigm that employs novel algorithms. The main advantage of suffix arrays over suffix trees is that, in practice, they use three to five times less space. From a complexity standpoint, suffix arrays permit on-line string searches of the type, “Is \( W \) a substring of \( A \)?” to be answered in time \( O(P + \log N) \), where \( P \) is the length of \( W \) and \( N \) is the ...

Note: This is a paper produced by a computational biologist (and future head of search for Google) in a non-biological context seven years before the publication of the first draft human genome.
THE BURROWS-WHEELER TRANSFORM
Idea to Reduce Memory: Compress the Genome

**Run-length encoding:** compresses every run of $n$ identical symbols $X$ into the substring “$nX$”.

```
Text
GGGGGGGGGGGCCCCCCCCCcAAAAAAAATT TT TT TT TT TT TT TT TT TT T T T T T T T T C CCC C G
```

```
10G11C7A15T5C1G
```

Run-length encoding
Idea to Reduce Memory: Compress the Genome

**Run-length encoding**: compresses every run of $n$ identical symbols $X$ into the substring “$nX$”.

Text

GGGGGGGGGGGCCCCCCCCCCCCCAAAAAAAAA

↓

10G11C7A15T5C1G

Run-length encoding

**Problem**: Genomes don’t have lots of runs…
Idea to Reduce Memory: Compress the Genome

**Run-length encoding:** compresses every run of $n$ identical symbols X into the substring “$nX$”.

Text

$\text{Text}$

$\text{GGGGGGGGGGGCCCCCCCCCAAAAAAAAAATTTTTTTTTTTTTTTTTCCCCCG}$

$\downarrow$

$10G11C7A15T5C1G$

Run-length encoding

...but they do have a lot of repeats!
Converting Repeats to Runs?

Text

→ Convert repeats to runs

\( f(\text{Text}) \)

→ Run-length encoding

\( \text{RLE}(f(\text{Text})) \)
Converting Repeats to Runs?

\[
\text{Text} \quad \rightarrow \quad \text{Convert repeats to runs?} \quad \rightarrow \quad f(\text{Text}) \quad \rightarrow \quad \text{Run-length encoding} \quad \rightarrow \quad RLE(f(\text{Text}))
\]
Converting Repeats to Runs?

One way we could convert repeats into runs would be to simply sort Text lexicographically.
Converting Repeats to Runs?

STOP: What is wrong with sorting the string?
Converting Repeats to Runs?

STOP: What is wrong with sorting the string?

Answer: There is no way to “invert” the compression to recover $Text$. That is, $f$ should be an injection!
Formally, define $S_{A,n}$ as the set of all strings of length $n+1$, with a “$” and $n$ symbols taken from an alphabet $A$. We want a function $f: S_{A,n} \rightarrow S_{A,n}$ that is **invertible**: if $f(x) = f(y)$, then $x = y$. 
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”
Form all cyclic rotations of “panamabanananas$”
Form all cyclic rotations of “panamabanananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of "panamabanananas$"
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”
Form all cyclic rotations of “panamabanananas$”
Form all cyclic rotations of “panamabanananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”
Form all cyclic rotations of “panamabanananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabananas$”
The Burrows-Wheeler Transform

Form all cyclic rotations of "panamabanananas$"
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”

Sort the strings lexicographically ($ comes first)
The Burrows-Wheeler Transform

Form all cyclic rotations of “panamabanananas$”

We call this matrix of symbols $M(\text{Text})$
The Burrows-Wheeler Transform

Form all cyclic rotations of 
$\text{Text} \, = \, ”panamabanananas$”

$BWT(\text{Text}) \, = \, \text{last column of } \, M(\text{text}) \, = \, ”smnpbnnaaaaa$ a”.
Let’s Examine BWT(Text) for Text = Watson & Crick, 1953

nd Corey (1). They kindly made their manuscript availa ........ a
nd criticism, especially on interatomic distances. We ........ a
nd cytosine. The sequence of bases on a single chain d ........ a
nd experimentally (3,4) that the ratio of the amounts o ........ u
nd for this reason we shall not comment on it. We wish ........ a
nd guanine (purine) with cytosine (pyrimidine). In oth ........ a
nd ideas of Dr. M. H. F. Wilkins, Dr. R. E. Franklin ........ a
nd its water content is rather high. At lower water co ........ a
nd pyrimidine bases. The planes of the bases are perpe ........ a
nd stereochemical arguments. It has not escaped our no ........ a
nd that only specific pairs of bases can bond together ........ u
nd the atoms near it is close to Furberg’s 'standard co ........ a
nd the bases on the inside, linked together by hydrogen ........ a
nd the bases on the outside. In our opinion, this stru ........ a
nd the other a pyrimidine for bonding to occur. The hy ........ a
nd the phosphates on the outside. The configuration of ........ a
nd the ration of guanine to cytosine, are always very c ........ a
nd the same axis (see diagram). We have made the usual ........ u
nd their co-workers at King’s College, London. One of ........ a

© 2024 Phillip Compeau
Where do you think the “u”s come from?

nd Corey (1). They kindly made their manuscript availa
nd criticism, especially on interatomic distances. We
nd cytosine. The sequence of bases on a single chain d
nd experimentally (3,4) that the ratio of the amounts o
nd for this reason we shall not comment on it. We wish
nd guanine (purine) with cytosine (pyrimidine). In oth
nd ideas of Dr. M. H. F. Wilkins, Dr. R. E. Franklin
nd its water content is rather high. At lower water co
nd pyrimidine bases. The planes of the bases are perpe
nd stereochemical arguments. It has not escaped our no
nd that only specific pairs of bases can bond together
nd the atoms near it is close to Furberg’s ’standard co
nd the bases on the inside, linked together by hydrogen
nd the bases on the outside. In our opinion, this stru
nd the other a pyrimidine for bonding to occur. The hy
nd the phosphates on the outside. The configuration of
nd the ration of guanine to cytosine, are always very c
nd the same axis (see diagram). We have made the usual
nd their co-workers at King’s College, London. One of
BWT Converts Repeats to Runs

**Burrows-Wheeler Transform!** Convert repeats to runs

```
Text
```

```
BWT(Text)
```

Run-length encoding

```
RLE(BWT(Text))
```
INVERTING THE BURROWS-WHEELER TRANSFORM
How Can We Decompress?

1. Text
2. Convert repeats to runs
3. $BWT(Text)$
4. Run-length encoding
5. $RLE(BWT(Text))$
How Can We Decompress?

IS IT POSSIBLE?

Convert repeats to runs

BWT(Text)

EASY

Run-length encoding

RLE(BWT(Text))
# A Strange Observation

<table>
<thead>
<tr>
<th>$panamabananas$</th>
<th>$p$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>abananas$panam$</td>
<td>$m$</td>
<td></td>
</tr>
<tr>
<td>amabananas$pan$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>anamabananas$p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ananas$panamab$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>anas$panamaban$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>as$panamaban$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bananas$panama$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mabananas$pana$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>namabananas$p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nanas$panamaba$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nas$panamabana$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>panamabananas$s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s$panamabanana$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Strange Observation

\[
\text{panama\text{\textsuperscript{banana}}} \rightarrow \text{panama\text{\textsuperscript{banana}}} \rightarrow \text{panama\text{\textsuperscript{banana}}} \rightarrow \ldots
\]
A Strange Observation

$panamabananas$
$abananass$panam
amanabananas$pan
anamanabananas$p
ananas$panamaban
anas$panamabanana
bananas$panama
mabananas$pana
namabananas$pa
nanas$panamaba
nas$panamabana
panamabananas$
$s$panamabanana

© 2024 Phillip Compeau
A Strange Observation
A Strange Observation

$panamabananas$
abananaspaman
amabananas$pan
anamabananas$pa
ananaspamanab
anas$panamaban
as$panamabanana
bananas$panama
mabananas$pana
namabananas$pa
ananaspamanaba
nas$panamabanana
panamabananas$
s$panamabanana

© 2024 Phillip Compeau
A Strange Observation

© 2024 Phillip Compeau
A Strange Observation

abananas$panamabananas
abanananas$panam
anamabananas$pan
pananas$panamaban
anas$panamaban
as$panamabananas
bananas$panamaban
mabananas$panama
namabananas$pana
nanas$panamabana
nas$panamabanana
panamabananas$s
s$panamabanana

© 2024 Phillip Compeau
A Strange Observation

panamabananas
abananas$panam
amabananas$pan
amamabananas$pan
ananas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas$panamas$panamabananas

© 2024 Phillip Compeau
A Strange Observation

$panamabananas$
$abananasa$pam$
$amanabananas$p$n$
$anamabananas$p$
$ananas$panamab$
$anas$panamaban$n$
$as$panamabanann$
$abananas$panama$
$amabananas$pana$
$namabananas$p$a$
$nanas$panamab$a$
$nas$panamabana$
$panamabananas$s$
$s$panamabananaa

© 2024 Phillip Compeau
Is It True in General?

$\text{panamabanananas}$

1. $\text{abanananas PAN}$
2. $\text{amabanananas PAN}$
3. $\text{anamabanananas P}$
4. $\text{ananas PANAMAB}$
5. $\text{anas PANAMABAN}$
6. $\text{as PANAMABAN}$
   $\text{bananas PANAMA}$
   $\text{mabanananas PAN}$
   $\text{namabanananas PA}$
   $\text{nanas PANAMABA}$
   $\text{nas PANAMABA}$
   $\text{panamabanananas S}$
   $\text{SPANAMABANANA}$
Is It True in General?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$p$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>m</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>m</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p$</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p$</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>s</td>
<td>$p$</td>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>a</td>
<td>m</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p$</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p$</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>$p$</td>
<td>a</td>
<td>n</td>
<td>a</td>
</tr>
</tbody>
</table>

These strings are sorted
Is It True in General?

These strings are sorted

1. $panamabanananas$
2. a$banananas$panam
3. amabananas$pan
4. anamabananas$pan
5. anas$panamabana
6. as$panamabana

Chop off a

banananas$panamabanananas$pan
namabanananas$pan
ananas$panamabana
nas$panamabana
s$panamabana

© 2024 Phillip Compeau
Is It True in General?

These strings are sorted

1. $panamabananas$bananas
2. $panamabananas$amabanas
3. $panamabananas$anamabanas
4. $panamabananas$ananas
5. $panamabananas$anas$panamaban
6. $panamabananas$as$panamaban
   bananas$panamabananas$panamabananas$panananas$panamabanas$panamabananas$panamabanana

Chop off a

Still sorted
Is It True in General?

These strings are sorted

1. $panamabananas$
2. abananas$panam$
3. amabananas$pan$
4. anananas$panamab$
5. anas$panamaban$
6. as$panamaban$

Chop off a

Still sorted

Add a to end
Is It True in General?

These strings are sorted:

1. $panamabananas$
2. abananasa$panam$
3. amabananas$pan$
4. ananas$panamaban$
5. anas$panamaban$
6. as$panamaban$

Chop off a

bananas$panamabananas$pan
namabananas$pananas$panamaban
s$panamabanana

Still sorted

Add a to end

Still sorted
Is It True in General?

These strings are sorted:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$panamabananas$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>m</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>m</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td>$p</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>s</td>
<td>$p</td>
<td>a</td>
<td>n</td>
</tr>
</tbody>
</table>

These strings are sorted:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bananas$</td>
<td>pan</td>
<td>a</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mabananas$</td>
<td>p</td>
<td>a</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>namabananas$</td>
<td>p</td>
<td>a</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>nanas$</td>
<td>panamaban</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>nas$</td>
<td>panamaban</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>panamaban</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Still sorted

Chop off $a$

Still sorted

Add $a$ to end

Ordering doesn’t change!

Still sorted

© 2024 Phillip Compeau
Is It True in General?

First-Last Property: The $k$-th occurrence of symbol in FirstColumn and the $k$-th occurrence of symbol in LastColumn correspond to the same position of symbol in Text.
Efficient BWT Decompression

$s_1$panamabananas$_1$
a$_1$bananas$panam$_1
a$_2$mabananas$pan$_1
a$_3$namabananas$pana$_1
a$_4$nanas$panamab$_1
a$_5$nas$panamaban$_2
a$_6$s$panamabanan$_3
b$_1$nanas$panamana$_1
m$_1$mabananas$pana$_2
n$_1$amabananas$pana$_3
n$_2$anas$panamaba$_4
n$_3$as$panamabana$_5
p$_1$anamabananas$_1$
s$_1$s$panamabanana$_6$

© 2024 Phillip Compeau
Efficient BWT Decompression

$1\text{panamabananas}$
$a1\text{bananas}\$\text{panam}_1$
$a2\text{mabanananas}\$\text{pan}_1$
$a3\text{namabanananas}\$\text{p}_1$
$a4\text{nananas}\$\text{panamab}_1$
$a5\text{nas}\$\text{panamaban}_2$
$a6\text{s}\$\text{panamaban}_3$
$b1\text{ananas}\$\text{panama}_1$
$m1\text{abananas}\$\text{pana}_2$
$n1\text{amabanananas}\$\text{pa}_3$
$n2\text{anas}\$\text{panamaba}_4$
$n3\text{as}\$\text{panamabana}_5$
$p1\text{anamabanananas}\$\text{1}$
$s1\$\text{panamabananana}_6$
Efficient BWT Decompression

$\text{panamabananas}_s$

$\text{bananas}_p$

$\text{mabananana}_a$

$\text{namabananas}_n$

$\text{nanas}_p$

$\text{nas}_p$

$\text{nas}_p$

$\text{nas}_p$

$\text{nas}_p$

$\text{nas}_p$

© 2024 Phillip Compeau
Efficient BWT Decompression

$s_1$panamabananas$a_1$
$a_1$bananas$panam_1$
$a_2$mabananas$pan_1$
$a_3$namabananas$p_1$
$a_4$nanas$panamab_1$
$a_5$nas$panamaban_2$
$a_6$s$panamabanana_3$
$b_1$ananaspbanama_1
$m_1$abananas$panama_2$
$n_1$amabananas$pa_3$
$n_2$anas$panamaba_4$
$n_3$as$panamabanana_5$
$p_1$anamabananas$1$
$s_1$panamabanana$a_6$
Efficient BWT Decompression

$1\text{panamabanananas}_1$
$a_1\text{bananas}_1\$\text{panam}_1$
$a_2\text{mabanananas}_1\$\text{pan}_1$
$a_3\text{namabanananas}_1\$p_1$
$a_4\text{nanas}_1\$\text{panamab}_1$
$a_5\text{natas}_1\$\text{panamaban}_2$
$a_6\text{s}_1\$\text{panamaban}_3$
$b_1\text{ananas}_1\$\text{panama}_1$
$m_1\text{bananas}_1\$\text{pana}_2$
$n_1\text{amabanananas}_1\$\text{pa}_3$
$n_2\text{ananas}_1\$\text{panamaba}_4$
$n_3\text{as}_1\$\text{panamabana}_5$
$p_1\text{anamabanananas}_1$
$s_1\$\text{panamabanana}_a_6$

© 2024 Phillip Compeau
Efficient BWT Decompression

$s_1$panamabananas$a_1$
$a_1$bananas$panam_1$
$a_2$mabananas$pan_1$
$a_3$namabananas$p_1$
$a_4$nanas$panamab_1$
$a_5$nas$panamaban_2$
$a_6$s$panamabana$n_3$
$b_1$ananas$panama_1$
$m_1$abananas$pana_2$
$n_1$amabananas$p_3$
$n_2$anas$panamaba_4$
$n_3$as$panamabana_5$
$p_1$anamabananas$1$
$s_1$panamabananana$a_6$
Efficient BWT Decompression

\$a_1\text{bananas}\$panam$1$
a_2mabananas$pana_1$
a_3namabanananas$pa_1$
a_4nanas$panamab_1$
a_5nas$panamaban_2$
a_6s$panamaban_3$
b_1anananas$panama_1$
mabananas$panam_2$
anananas$panama_3$
abananas$panamab_4$
n$panamabana_5$
p_1anamabananas$pa_1$
s$panamabanana_6$
Efficient BWT Decompression

$s_1panamabananas_1$
$a_1bananas$pana$m_1$
$a_2mabanana$span$a_1$
$a_3namabananas$pana$p_1$
$a_4nanasanapanama$b_1$
$a_5nas$panamaban$a_2$
$a_6s$panamabanana$n_1$
$b_1ananas$panama$n_3$
$m_1abanan$panama$n_1$
$n_1amabananas$pana$p_3$
$n_2anasanapanama$a_4$
$n_3asanpanamabanana$a_5$
$p_1anamabananas$pana$p_1$
$s_1$panamabanana$a_6$
Efficient BWT Decompression

\[
\begin{align*}
\$ & \text{panamabananas}_1 \\
a_1 & \text{bananas}\$\text{panam}_1 \\
a_2 & \text{mabanananas}\$\text{pan}_1 \\
a_3 & \text{namabanananas}\$\text{p}_1 \\
a_4 & \text{nanas}\$\text{panamab}_1 \\
a_5 & \text{nas}\$\text{panamaban}_2 \\
a_6 & \text{s}\$\text{panamaban}_3 \\
b_1 & \text{ananas}\$\text{panama}_1 \\
m_1 & \text{abanananas}\$\text{pana}_2 \\
n_1 & \text{amabanananas}\$\text{pa}_3 \\
n_2 & \text{anas}\$\text{panamaba}_4 \\
n_3 & \text{as}\$\text{panamaban}_5 \\
p_1 & \text{anamabanananas}\$\_1 \\
s_1 & \$\text{panamabanana}_6
\end{align*}
\]
Efficient BWT Decompression

$s_1 panamabananas_1$
a_1 bananas$pana_1$
a_2 mabananas$pan_1$
a_3 namabananas$pana_1$
a_4 nanas$panamab_1$
a_5 nas$panamaban_2$
a_6 s$panamaban_3$
b_1 ananas$panaman_1$
m_1 abananas$pana_2$
n_1 amabananas$pana_3$
n_2 anas$panamab_4$
n_3 as$panamaban_5$
p_1 nanabananas$pana_1$
s_1$panamabanana_6$

© 2024 Phillip Compeau
Efficient BWT Decompression

$s_1panamabanana\text{a}_1$
$a_1\text{bananas}$panam$_1$
$a_2mabananas$pan$_1$
$a_3namabananas$p_1$
$a_4nanas$panamab$_1$
$a_5nas$panamaban$\text{a}_2$
$a_6s$panamabanana$_3$
$b_1ananas$panama$_1$
$m_1abananas$pana$_2$
$n_1amabananas$pa$_3$
$n_2ananas$panamaba$_4$
$n_3as$panamaban$_5$
$p_1anamabananas$$_1$
$s_1$panamabananana$_6$
Efficient BWT Decompression

\[
\begin{align*}
\$ & \text{panamabananas}_1 \\
a_1 & \text{bananas}\$\text{panam}_1 \\
a_2 & \text{mabananas}\$\text{pan}_1 \\
a_3 & \text{namanabananas}\$p_1 \\
a_4 & \text{nanas}\$\text{panamab}_1 \\
a_5 & \text{nasa}\$\text{panamaban}_2 \\
a_6 & \text{s}\$\text{panamaban}_3 \\
b_1 & \text{ananasan}\$\text{panama}_1 \\
m_1 & \text{abananas}\$\text{pana}_2 \\
m_1 & \text{amabananas}\$\text{pa}_3 \\
a_4 & \text{ananas}\$\text{panamaban}_3 \\
_2 & \text{ananas}\$\text{panamaban}_4 \\
_3 & \text{nas}\$\text{panamaban}_5 \\
p_1 & \text{anamabananas}\$1 \\
s_1 & \$\text{panamabanana}_6
\end{align*}
\]
Efficient BWT Decompression

$s_1$panamabananas$_1$
a$_1$bananas$\$panam$_1$
a$_2$manabananas$\$pan$_1$
a$_3$namabananas$\$p$_1$
a$_4$nanas$\$panamaban$_1$
a$_5$nas$\$panamaban$$_2$
a$_6$ans$\$panamaban$$_3$
b$_1$ananas$\$panama$_1$
m$_1$abananas$\$pana$_2$
n$_1$amabananas$\$pa$_3$
n$_2$anas$\$panamabanaa$_4$
n$_3$as$\$panamabanaa$_5$
p$_1$anamabanananas$\$$_1$
s$_1$$\$panamabananaa$_6$
Efficient BWT Decompression

$1\text{panamabananas}1$
$a1\text{bananas}1\text{panam}1$
$a2\text{mabanananas}1\text{pan}1$
$a3\text{nababanananas}1\text{p}1$
$a4\text{nana}1\text{panama}b1$
$a5\text{nas}1\text{panamaban}2$
$a6\text{as}1\text{panamaban}2$
$b1\text{an}1\text{panama}1$
$m1\text{abanananas}1\text{pana}2$
$n1\text{amabanananas}1\text{pan}3$
$n2\text{ananas}1\text{panamaba}4$
$n3\text{as}1\text{panamaban}a5$
$p1\text{anamabanananas}1$
$s1\text{$panamabanana}a6$

© 2024 Phillip Compeau
Efficient BWT Decompression

$1panamabananas1$
a1bananas$panam1$
a2mabananas$pan1$
a3namabananas$1p1$
a4nanas$panama1$
b1ananas$panama1$
m1abananas$pana2$
n1amabananas$pana3$
n2anas$panamas4$
n3as$panamas5$
p1anamabananas$1$
s1$panamabanana6"
Efficient BWT Decompression

$1_{panamabananas}s_{1}$
$a_{1}_{bananas}$s_{1}_{panam}$1$
$a_{2}_{mabanananas}$s_{1}_{pana}$1$
$a_{3}_{namabanananas}$s_{1}_{pana}$1$
$a_{4}_{nananas}$s_{1}_{panama}$b_{1}$
$a_{5}_{nas}$s_{1}_{panamabana}$n_{2}$
$a_{6}_{s}$s_{1}_{panamabanan}$a_{3}$
$b_{1}_{ananas}$s_{1}_{panamat}$a_{1}$
m_{1}_{abanananas}$s_{1}_{panam}$a_{2}$
n_{1}_{amabananas}$s_{1}_{pana}$a_{3}$
n_{2}_{anas}$s_{1}_{panamabana}$a_{4}$
n_{3}_{as}$s_{1}_{panamabanana}$a_{5}$
p_{1}_{anamabananas}$s_{1}_{panamabanana}$a_{6}$

© 2024 Phillip Compeau
Efficient BWT Decompression

$s_1 p a n a m a b a n a n a s_1$
a_1 b a n a n a s $p a n a m_1$
a_2 m a b a n a n a s $p a n_1$
a_3 n a m a b a n a n a s $p_1$
a_4 n a n a s $p a n a m a b_1$
a_5 n a s $p a n a m a b a n_2$
a_6 a n a n a s $p a n a m a b a n n_3$
b_1 a n a n a s $p a n a m a b_1$
m_1 a b a n a n a s $p a n a_2$
n_1 a m a b a n a n a s $p a_3$
n_2 a n a s $p a n a m a b a_4$
n_3 a s $p a n a m a b a_5$
p_1 a n a m a b a n a n a s $s_1$
s_1 $p a n a m a b a n a n a a_6$
Efficient BWT Decompression

$s_1$panamabananas
$a_1$bananas$panam_1$
a_2$bananas$pana_1$
a_3$bananas$panam
$a_4$bananas$panamab_1$
a_5$bananas$panamaban_2$
a_6$bananas$panamaban_3$
b_1$panamabananas$pana_1$
m_1$bananas$panamab_1$
n_1$bananas$panamab_2$
n_2$bananas$panamab_3$
n_3$bananas$panamab_4$
p_1$bananas$panamab_5$
s_1$panamabananas

© 2024 Phillip Compeau
Efficient BWT Decompression

\$1 \text{panama} \text{banana} \text{nas}_1\\a_1 \text{bananas} \text{pana} \text{m}_1\\a_2 \text{mabana} \text{nas} \text{pan}_1\\a_3 \text{namaba} \text{nan} \text{as}_1\\a_4 \text{nana} \text{as} \text{panama} \text{a}_1\\a_5 \text{nas} \text{panama} \text{ban}_1\\a_6 \text{s} \text{panama} \text{banan}_1\\b_1 \text{anana} \text{sa} \text{panama} \text{a}_1\\m_1 \text{abanana} \text{a} \text{pana} \text{a}_2\\n_1 \text{amaba} \text{na} \text{na} \text{sa} \text{pana} \text{a}_3\\n_2 \text{ana} \text{sa} \text{panama} \text{ba} \text{a}_4\\n_3 \text{na} \text{sa} \text{panama} \text{a} \text{a}_5\\p_1 \text{anama} \text{ba} \text{nana} \text{sa} \text{pana} \text{a}_6\\s_1 \text{sa} \text{panama} \text{banana} \text{a}_6

© 2024 Phillip Compeau
Efficient BWT Decompression

$1\text{panamabananas}_1$
$a_1\text{bananas}$\text{panam}_1$
a_2\text{mabananas}$\text{pan}_1$
a_3\text{namabananas}$\text{p}_1$
a_4\text{anananas}$\text{panamab}_1$
a_5\text{nas}$\text{panamaban}_2$
a_6\text{s}$\text{panamabanan}_3$
b_1\text{ananas}$\text{panama}_1$
m_1\text{abananas}$\text{pana}_2$
n_1\text{amabananas}$\text{pan}_3$
n_2\text{ananas}$\text{panamaba}_4$
n_3\text{as}$\text{panamabana}_5$
p_1\text{anamabananas}$\text{s}_1$
s_1\text{$\ddagger$panamabanan}_a_6$

© 2024 Phillip Compeau
Efficient BWT Decompression

\[ \$1\text{panamabananas}_1 \]
\[ a_1\text{bananas}\$\text{panam}_1 \]
\[ a_2\text{mabanananas}\$\text{pan}_1 \]
\[ a_3\text{namabanananas}\$\text{p}_1 \]
\[ a_4\text{nanaas}\$\text{panamab}_1 \]
\[ a_5\text{nas}\$\text{panamaban}_2 \]
\[ a_6\text{s}\$\text{panamabanana}_3 \]
\[ b_1\text{ananaas}\$\text{panama}_1 \]
\[ m_1\text{abanaas}\$\text{pana}_2 \]
\[ n_1\text{amabanaas}\$\text{pa}_3 \]
\[ n_2\text{anaas}\$\text{panamaba}_4 \]
\[ n_3\text{as}\$\text{panamabana}_5 \]
\[ p_1\text{anamabanananas}\$1 \]
\[ s_1\$\text{panamabanana}_a6 \]
Efficient BWT Decompression

\[
\begin{align*}
$1panamabananas & $s_1 \\
a_1bananas & $p_{a_1} \\
a_2mabananas & $p_{a_2} \\
a_3namabananas & $p_{a_3} \\
a_4nanas & $p_{a_4} \\
a_5nas & $p_{a_5} \\
a_6s & $p_{a_6} \\
b_1ananas & $p_{b_1} \\
b_2anan & $p_{b_2} \\
b_3anas & $p_{b_3} \\
b_4anas & $p_{b_4} \\
b_5nas & $p_{b_5} \\
b_6s & $p_{b_6} \\
a_1panamabananas & $s_1 \\
a_2bananas & $s_1 \\
a_3namabananas & $s_1 \\
a_4nanas & $s_1 \\
a_5nas & $s_1 \\
a_6s & $s_1 \\
\end{align*}
\]
Efficient BWT Decompression

$s_1$panamabananas$_1$
a$_1$bananasp$panam_1$
a$_2$mbananasp$panan_1$
a$_3$namabananas$_p_1$
a$_4$nanasp$panamab_1$
a$_5$nas$_p$panamaban$_2$
a$_6$s$_p$panamaban$_3$
b$_1$anasp$panam$_1$
m$_1$abanasp$pana_2$
m$_1$aminabanasp$pana_3$
n$_1$amabanasp$pana_4$
n$_2$anasp$panamaba_4$
n$_3$as$_p$panamaban$_5$
p$_1$amabanasp$_1$
s$_1$p$panamabanana$$_6$
Efficient BWT Decompression

\$a_1\text{bananas}\$\$\text{pana}_1\$
\$a_2\text{mabananas}\$\$\text{pan}_1\$
\$a_3\text{namabanananas}\$\$p_1\$
\$a_4\text{nanas}\$\$\text{panama}_1\$
\$a_5\text{nas}\$\$\text{panamaban}_2\$
\$a_6\text{s}\$\$\text{panamaban}_3\$
\$b_1\text{nanas}\$\$\text{panama}_1\$
\$m_1\text{abananas}\$\$\text{pana}_2\$
\$n_1\text{amabanananas}\$\$\text{pana}_3\$
\$n_2\text{anas}\$\$\text{panama}_4\$
\$n_3\text{as}\$\$\text{panama}_5\$
\$p_1\text{ananabana}\$\$\text{pana}_6\$
\$s_1\$\$\text{panamabanana}_6\$

© 2024 Phillip Compeau
Efficient BWT Decompression

$1$p$anamabananas$a1
a1$bananas$panam1
a2$mabananas$pan1
a3$namabananas$pana1
a4$nanas$panamaban1
a5$nas$panamaban2
a6$s$panamabanan3
b1$lananas$panamaban4
m1$abanananas$pana2
n1$amabananas$pana3
n2$anas$panamaban4
n3$as$panamabanana5
p1$anamabananas$pana6
s1$s$panamabananana6
Efficient BWT Decompression

\$s_1\text{panamabanananas}_1\text{a}_1\text{bananas}_1\text{pana}_1\text{a}_2\text{mabananas}_1\text{pan}_1\text{a}_3\text{namabanananas}_1\text{p}_1\text{a}_4\text{nananas}_1\text{panamab}_1\text{a}_5\text{nas}_1\text{panamaban}_2\text{a}_6\text{s}_1\text{panamabananas}_3\text{b}_1\text{anananas}_1\text{panama}_1\text{m}_1\text{abanananas}_1\text{pana}_2\text{n}_1\text{amabananas}_1\text{pana}_3\text{n}_2\text{anas}_1\text{panamabana}_4\text{n}_3\text{as}_1\text{panamabana}_5\text{p}_1\text{anamabanananas}_1\text{panamabanananas}_6\$
Efficient BWT Decompression
Efficient BWT Decompression

$p_1$ panamabanananas$1$
$a_1$ bananas$panam_1$
$a_2$ mabanananas$pan_1$
$a_3$ namabanananas$p_1$
$a_4$ nanas$panama_1$
$a_5$ nas$panamaban_2$
$a_6$ s$panamaban_3$
$b_1$ ananas$panama_1$
$m_1$ abanananas$pana_2$
$n_1$ amabananas$pa_3$
$n_2$ anas$panamaba_4$
$n_3$ as$panamabana_5$
$p_1$ anamabananas$1$
$s_1$ $panamabanana_6$
Efficient BWT Decompression

Memory: $2|Text| = O(|Text|)$. 
So We Can Decompress BWT ...

It’s possible!

Convert repeats to runs

Run-length encoding

EASY

BWT(Text)

Text

RLE(BWT(Text))

© 2024 Phillip Compeau
So We Can Decompress BWT ... But What About Pattern Matching?

It’s possible! Convert repeats to runs

\[ \text{Text} \]

\[ \text{BWT}(\text{Text}) \]

EASY Run-length encoding

\[ \text{RLE}(\text{BWT}(\text{Text})) \]
PATTERN MATCHING WITH BURROWS-WHEELER
Recalling Our Goal

Pattern Matching with Suffix Array:

• Runtime: $O(|Text| + |Patterns|)$

• Memory: $O(|Text|)$

• Problem: suffix tree takes $\sim 20 \times |Text|$ space, but suffix array takes $\sim 4 \times |Text|$ space.
Recalling Our Goal

Pattern Matching with Suffix Array:
- Runtime: $O(|Text| + |Patterns|)$
- Memory: $O(|Text|)$
- Problem: suffix tree takes $\sim 20 \times |Text|$ space, but suffix array takes $\sim 4 \times |Text|$ space.

Can we use BWT($Text$) as our data structure instead?
Pattern Matches “Clump” at Start of $M(\text{Text})$

\begin{align*}
&\text{$s_1\text{panamabanananas}_1$} \\
&\text{a$_1$bananas$\text{panam}_1$} \\
&\text{a$_2$mabanananas$\text{pan}_1$} \\
&\text{a$_3$namabanananas$\text{p}_1$} \\
&\text{a$_4$nanas$\text{panamab}_1$} \\
&\text{a$_5$nas$\text{panamaban}_2$} \\
&\text{a$_6$s$\text{panamaban}_3$} \\
&\text{b$_1$anananas$\text{panama}_1$} \\
&\text{m$_1$abanananas$\text{pana}_2$} \\
&\text{n$_1$amabanananas$\text{pa}_3$} \\
&\text{n$_2$anas$\text{panamaba}_4$} \\
&\text{n$_3$as$\text{panamabana}_5$} \\
&\text{p$_1$anamabanananas$\text{s}_1$} \\
&\text{s$_1$s$\text{panamabana}_6$}
\end{align*}
Connecting $M(Text)$ to Suffix Array

<table>
<thead>
<tr>
<th>Sorted Suffixes</th>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$$</td>
<td>13</td>
</tr>
<tr>
<td>abananas</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$</td>
<td>1</td>
</tr>
<tr>
<td>ananas$</td>
<td>7</td>
</tr>
<tr>
<td>anas$</td>
<td>9</td>
</tr>
<tr>
<td>as$</td>
<td>11</td>
</tr>
<tr>
<td>bananas$</td>
<td>6</td>
</tr>
<tr>
<td>mabanananas$</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$</td>
<td>2</td>
</tr>
<tr>
<td>nanas$</td>
<td>8</td>
</tr>
<tr>
<td>nas$</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$</td>
<td>12</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
Connecting $M(\text{Text})$ to Suffix Array

We could find pattern matches easily if we had all the suffixes, but we would need $O(\lvert \text{Text} \rvert^2)$ space...
Connecting $M(\text{Text})$ to Suffix Array

We could find pattern matches easily if we had all the suffixes, but we would need $O(|\text{Text}|^2)$ space…

We are going to pattern match using just two columns of $M(\text{Text})$: $\text{FirstColumn}$ and $\text{LastColumn}$
We Match Patterns Backward

Searching for ana in Text = panamabananass
We Match Patterns Backward

Searching for ana in Text = panamabanananas
We Match Patterns *Backward*

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$m_1$</th>
<th>$n_1$</th>
<th>$p_1$</th>
<th>$b_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
</tr>
</thead>
</table>

Searching for **ana** in *Text = panamabanananas*
We Match Patterns *Backward*

$\text{Searching for } \text{ana} \text{ in } Text = \text{panamabanananas}$
We Match Patterns \textit{Backward}

Searching for \texttt{ana} in \texttt{Text = panamabanananas}

Now we can apply the First-Last Property and find where these three “n” are hiding in \textit{FirstColumn}. 

\begin{verbatim}
$1 \texttt{?????????????? s}_1  \\
a_1 \texttt{?????????????? m}_1  \\
a_2 \texttt{?????????????? n}_1  \\
a_3 \texttt{?????????????? p}_1  \\
a_4 \texttt{?????????????? b}_1  \\
a_5 \texttt{?????????????? n}_2  \\
a_6 \texttt{?????????????? n}_3  \\
b_1 \texttt{?????????????? a}_1  \\
m_1 \texttt{?????????????? a}_2  \\
n_1 \texttt{?????????????? a}_3  \\
n_2 \texttt{?????????????? a}_3  \\
n_3 \texttt{?????????????? a}_4  \\
p_1 \texttt{?????????????? a}_5  \\
s_1 \texttt{?????????????? a}_6  \\
\end{verbatim}
We Match Patterns \textit{Backward}

Searching for \texttt{ana} in \texttt{Text = panamabanananas}

Now we can apply the First-Last Property and find where these three “n” are hiding in \textit{FirstColumn}.

We can infer that these three rows start with “na”.

© 2024 Phillip Compeau
We Match Patterns Backward

Searching for **ana** in Text = panamabananans
We Match Patterns *Backward*

Searching for **ana** in *Text = panamabanananas*

All three match, and again we apply the First-Last Property.
We Match Patterns *Backward*

Searching for **ana** in *Text* = panamabanananas

All three match, and again we apply the First-Last Property.

We have found the occurrences of “ana”!
Problem: For a DNA alphabet, it will take \((1/4) |\text{Text}|\) runtime just to cycle through all these!
Matching “ana” backward with count arrays

“Count” arrays allow us to know what we’ve encountered at every point, so the subscripts become unnecessary.
Matching “ana” backward with count arrays

Here: we know that $n_1$ through $n_3$ are here by checking the two green values.

Of course, these arrays are a memory waste…
Matching “ana” backward with count arrays

We repeat this process and see that we must be looking for $a_3$ through $a_5$. 
Matching “ana” backward with count arrays

Once we find $a_3$ through $a_5$, we have now found that there are three occurrences of “ana”.

But where are they in text?

© 2024 Phillip Compeau
WHERE ARE THE MATCHED PATTERNS?
Multiple Pattern Matching Problem: Find all occurrences of a collection of patterns in a text.

- **Input:** A string Text and a collection Patterns containing (shorter) strings.
- **Output:** All starting positions in Text where a string from Patterns appears as a substring.
Where are the Matches?

Example: We know that \textbf{ana} occurs 3 times, but where?
The Suffix Array Holds the Key

$panamabanananas$
$abananas$panam$
$amabanananas$pan$
$anamabanananas$p$
$ananas$panamab$
$anas$panamaban$
as$panamabanana
$bananas$panama
$abanananas$pana
$namabanananas$pa
$nanas$panamaba
$nas$panamabana
$panamabanananas$
s$panamabana
The Suffix Array Holds the Key

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$panamabanananas$</td>
<td>13</td>
</tr>
<tr>
<td>abanananas$panam</td>
<td>5</td>
</tr>
<tr>
<td>amabanananas$pan</td>
<td>3</td>
</tr>
<tr>
<td>anamabanananas$p</td>
<td>1</td>
</tr>
<tr>
<td>ananas$panamab</td>
<td>7</td>
</tr>
<tr>
<td>anas$panamaban</td>
<td>9</td>
</tr>
<tr>
<td>as$panamabanan</td>
<td>11</td>
</tr>
<tr>
<td>bananas$panama</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$pana</td>
<td>4</td>
</tr>
<tr>
<td>namabanananas$pa</td>
<td>2</td>
</tr>
<tr>
<td>nanas$panamaba</td>
<td>8</td>
</tr>
<tr>
<td>nas$panamabana</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$panamabana</td>
<td>12</td>
</tr>
</tbody>
</table>
The Suffix Array Holds the Key

“But we’ve already used the suffix array for pattern matching! How is this useful?”

Suffix Array

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$panamabanananas</td>
<td>13</td>
</tr>
<tr>
<td>abananas$panam</td>
<td>5</td>
</tr>
<tr>
<td>amabananas$pan</td>
<td>3</td>
</tr>
<tr>
<td>amabanananas$p</td>
<td>1</td>
</tr>
<tr>
<td>ananas$panamab</td>
<td>7</td>
</tr>
<tr>
<td>anas$panamaban</td>
<td>9</td>
</tr>
<tr>
<td>as$panamabananan</td>
<td>11</td>
</tr>
<tr>
<td>bananas$panama</td>
<td>6</td>
</tr>
<tr>
<td>mabananas$pana</td>
<td>4</td>
</tr>
<tr>
<td>namabananas$pa</td>
<td>2</td>
</tr>
<tr>
<td>nanas$panamaba</td>
<td>8</td>
</tr>
<tr>
<td>nas$panamabana</td>
<td>10</td>
</tr>
<tr>
<td>panamabanananas$</td>
<td>0</td>
</tr>
<tr>
<td>s$panamabananana</td>
<td>12</td>
</tr>
</tbody>
</table>
How Can BWT Possibly Be Useful if we need all this information?

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th>$</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
BURROWS AND WHEELER SET UP CHECKPOINTS
Key Point: We Don’t Need to Store All This Data

Suffix Array

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th>$</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

© 2024 Phillip Compeau
First: Dealing with the First Column

STOP: Is there any way of representing the information in the first column using less data?
First: Dealing with the First Column

STOP: Is there any way of representing the information in the first column using less data?

Answer: We only need to store the first occurrence of each symbol.
First: Dealing with the First Column

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** we don’t even need to store that the first symbol occurs at position 1 of the first column.
First: Dealing with the First Column

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
</table>

Note: we don’t even need to store that the first symbol occurs at position 1 of the first column.

STOP: For a genome, how many integers will we need to store to hold the first occurrence of each symbol?
First: Dealing with the First Column

Note: we don’t even need to store that the first symbol occurs at position 1 of the first column.

Answer: Only three! The position of the first occurrence of C, G, and T. ($ occurs at position 0; A occurs at position 1.)
Second: A “Partial” Suffix Array

Partial suffix array: only stores values that are divisible by $K$ for some integer $K$.

$K = 5$ at left.
Second: A “Partial” Suffix Array

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.
Second: A “Partial” Suffix Array

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.
Second: A “Partial” Suffix Array

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.
Second: A “Partial” Suffix Array

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.
Second: A “Partial” Suffix Array

We can make at most $K - 1$ additional backtrack steps to determine where our pattern match is.
We have a match! We know “abana” occurs at position 5, and we took two steps back, so this “ana” occurs at position 7.
Third: “Checkpoint” Count Arrays

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

“Count” arrays help us know what we’ve encountered up to given row.
Third: “Checkpoint” Count Arrays

“Count” arrays help us know what we’ve encountered up to given row.

Checkpoint arrays: only store every $C$ arrays.

<table>
<thead>
<tr>
<th>$s$</th>
<th>$a$</th>
<th>$b$</th>
<th>$m$</th>
<th>$n$</th>
<th>$p$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Third: “Checkpoint” Count Arrays

<table>
<thead>
<tr>
<th>$</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

STOP: Say we are at position 13 and see “a”. How can we know how many “a”s have occurred so far?
Third: “Checkpoint” Count Arrays

<table>
<thead>
<tr>
<th>$</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Answer:** Go to nearest checkpoint array, which has counted 3 “a”’s up to position 10, and then count three more (total of 6).
“How are $K$ and $C$ chosen in practice?”
So, We Need Just Four Items ... Total Memory ($K = C = 100$): $\sim 1.5|Text|!

<table>
<thead>
<tr>
<th>First Occurrence</th>
<th>BWT</th>
<th>Checkpoint Arrays</th>
<th>(Partial) Suffix Array</th>
</tr>
</thead>
</table>
| 1                | s   | $\begin{array}{cccccccc}
$ & a & b & m & n & p & s \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}$ | 13 |
| 2                | m   | $\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$ | 5 |
| 3                | n   | $\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 1 & 1 \\
0 & 0 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 3 |
| 4                | p   | $\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 3 & 1 & 1 \\
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 1 |
| 5                | b   | $\begin{array}{cccccccc}
0 & 3 & 1 & 1 & 3 & 1 & 1 \\
0 & 4 & 1 & 1 & 3 & 1 & 1 \\
0 & 5 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 7 |
| 6                | n   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 4 & 1 & 1 & 3 & 1 & 1 \\
0 & 5 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 9 |
| 7                | n   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 11 |
| 8                | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 6 |
| 9                | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 4 |
| 10               | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 2 |
| 11               | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 8 |
| 12               | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 10 |
| 13               | $\#$| $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 0 |
| 14               | a   | $\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 3 & 1 & 1 \\
0 & 2 & 1 & 1 & 3 & 1 & 1 \\
0 & 3 & 1 & 1 & 3 & 1 & 1
\end{array}$ | 12 |
EPILOGUE: MISMATCH-TOLERANT READ MAPPING
Returning to Our Original Problem

**Multiple Pattern Matching Problem:** *Find all occurrences of a collection of patterns in a text.*

- **Input:** A string *Text* and a collection *Patterns* containing (shorter) strings.
- **Output:** All starting positions in *Text* where a string from *Patterns* appears as a substring.
Multiple **Approximate Pattern Matching Problem:** Find all occurrences of a collection of patterns in a text.

- **Input:** A string $\text{Text}$, a collection $\text{Patterns}$ containing (shorter) strings, and an integer $d$.
- **Output:** All starting positions in $\text{Text}$ where a string from $\text{Patterns}$ appears as a substring with at most $d$ mismatches.
Method 1: Seeding

Say that \textit{Pattern} appears in \textit{Text} with just one mismatch.

\textbf{Pattern} \quad \texttt{acttggct}

\textbf{Text} \quad \ldots \texttt{ggcacaactaggctcc} \ldots
Method 1: Seeding

Say that *Pattern* appears in *Text* with just one mismatch.

```
Pattern:    actttggct
Text:       ggccacactaggctcc ...
```
Method 1: Seeding

Say that *Pattern* appears in *Text* with just one mismatch.

If we divide the strings in half, then one must match exactly!
Method 1: Seeding

Let’s take another example of strings that have 3 mismatches.

Pattern: `acttaggctcggggataatcc`

Text: `...ggcacaactaagtctcgggataaagccccccc...`
Method 1: Seeding

Let’s take another example of strings that have 3 mismatches.

Pattern: \texttt{acttaaggctcgggatatcc}

Text: \texttt{...ggcacactaagttctcgggataaggcccccc...}

Now we can divide strings into four equal pieces and find at least one that matches.
Method 1: Seeding

**Theorem:** If $Pattern$ occurs in $Text$ with $d$ mismatches, then we can divide $Pattern$ into $d + 1$ “equal” pieces and find at least one exact match.

Method 1: Seeding

**Theorem:** If Pattern occurs in Text with $d$ mismatches, then we can divide Pattern into $d + 1$ “equal” pieces and find at least one exact match.
Method 1: Seeding

**Theorem:** If Pattern occurs in Text with $d$ mismatches, then we can divide Pattern into $d + 1$ “equal” pieces and find at least one exact match.
Method 1: Seeding

**Theorem:** If Pattern occurs in Text with $d$ mismatches, then we can divide Pattern into $d + 1$ “equal” pieces and find at least one exact match.

```
X X X X X X X  X X X X X X X X X X X X X X X X X X
X X X X X X X  X X X X X X X X X X X X X X X X X X
X X X X X X X  X X X X X X X X X X X X X X X X X X
```
Method 1: Seeding

**Theorem:** If Pattern occurs in Text with $d$ mismatches, then we can divide Pattern into $d + 1$ “equal” pieces and find at least one exact match.
Method 1: Seeding

**Theorem (Formally):** If two strings of length $n$ match with at most $d$ mismatches, then they must share a $k$-mer of length $k = \lfloor n/(d+1) \rfloor$. 

```
XXXXXXX
XXXXXXX
XXXXXXX
```
Method 1: Seeding

**Seed-and-extend** for inexact pattern matching:

1. Divide *Pattern* into \(d+1\) ”equal” segments (called *seeds*).
2. Find which seeds match *Text* exactly, using an approach like BWT (**seed detection**).
3. Extend all seeds in both directions and align *Pattern* against this region of *Text* (**seed extension**) – can be generalized to include gaps.
Method 2: BWT Saves the Day Again

Recall: searching for **ana** in **panamabanananas**
Method 2: BWT Saves the Day Again

Recall: searching for ana in panamabanananas

```
$1panamabanananas
a1bananas$panam1
a2mabanananas$pan1
a3namabanananas$p1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamabanan3
b1anananas$panama1
m1abananas$pana2
n1amabanananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanana6
```
Method 2: BWT Saves the Day Again

Recall: searching for ana in panamabanananas

\[ \text{panabanananas}_1 \]
\[ \text{bananas}_1 \text{panam}_1 \]
\[ \text{mabanananas}_2 \text{pan}_1 \]
\[ \text{namabanananas}_3 \text{p}_1 \]
\[ \text{nanas}_4 \text{panama}_1 \]
\[ \text{nas}_5 \text{panamaban}_2 \]
\[ \text{s}_6 \text{panamabana}_3 \]
\[ \text{bananas}_1 \text{panam}_1 \]
\[ \text{mabanananas}_2 \text{pan}_1 \]
\[ \text{namabanananas}_3 \text{p}_1 \]
\[ \text{nanas}_4 \text{panama}_1 \]
\[ \text{nas}_5 \text{panamaban}_2 \]
\[ \text{as}_6 \text{panamabana}_3 \]
\[ \text{p}_1 \text{panamabanananas}_5 \]
\[ \text{s}_1 \text{panamabana}_6 \]
Method 2: BWT Saves the Day Again

Recall: searching for *ana* in *panamabananas*

If we allow 1 mismatch, then we need to keep the red letters around.
Method 2: BWT Saves the Day Again

Recall: searching for **ana** in **panamabanananas**

If we allow 1 mismatch, then we need to keep the red letters around.

<table>
<thead>
<tr>
<th>$s_1\text{panamabanananas}_1$</th>
<th>$a_1\text{bananas}_1\text{panam}_1$</th>
<th>$a_2\text{mabananas}_1\text{pan}_1$</th>
<th>$a_3\text{namabananas}_1\text{p}_1$</th>
<th>$a_4\text{nanas}_1\text{panamab}_1$</th>
<th>$a_5\text{nas}_1\text{panamaban}_2$</th>
<th>$a_6\text{s}_1\text{panamabananas}_3$</th>
<th>$b_1\text{ananas}_1\text{panama}_1$</th>
<th>$m_1\text{abananas}_1\text{pana}_2$</th>
<th>$n_1\text{amabananas}_1\text{pa}_3$</th>
<th>$n_2\text{ananas}_1\text{panamab}_4$</th>
<th>$n_3\text{as}_1\text{panamaban}_5$</th>
<th>$p_1\text{anamabananas}_1\text{s}_1$</th>
<th>$s_1\text{spanamabanan}_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td># Mismatches</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Method 2: BWT Saves the Day Again

Recall: searching for **ana** in **panamabanananas**

Now we extend only strings that have at most 1 mismatch.

```
<table>
<thead>
<tr>
<th>string</th>
<th># Mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p1 pana bana nasa s1</td>
<td>1</td>
</tr>
<tr>
<td>a1 bananas $pam m1</td>
<td>0</td>
</tr>
<tr>
<td>a2 mabananas $pan n1</td>
<td>1</td>
</tr>
<tr>
<td>a3 namabananas $p1</td>
<td>1</td>
</tr>
<tr>
<td>a4 nanas $panama b1</td>
<td>0</td>
</tr>
<tr>
<td>a5 nas $panamaban n2</td>
<td>0</td>
</tr>
<tr>
<td>a6 s $panamabanana n3</td>
<td>0</td>
</tr>
<tr>
<td>b1 ana $panama1</td>
<td>1</td>
</tr>
<tr>
<td>m1 abananas $pana2</td>
<td>1</td>
</tr>
<tr>
<td>n1 amabananas $pa3</td>
<td>0</td>
</tr>
<tr>
<td>n2 anas $panamaba4</td>
<td>0</td>
</tr>
<tr>
<td>n3 as $panamabana5</td>
<td>0</td>
</tr>
<tr>
<td>p1 anabanananas $1</td>
<td>0</td>
</tr>
<tr>
<td>s1 $panamabanana a6</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Method 2: BWT Saves the Day Again

Recall: searching for \textit{ana} in \textit{panamabanananas}

Now we extend only strings that have at most 1 mismatch.
Method 2: BWT Saves the Day Again

Recall: searching for **ana** in panamabanananas

One string produces a second mismatch (the $), so we discard it.
Method 2: BWT Saves the Day Again

Recall: searching for **ana** in **panamabanananas**

In the end, we have five 3-mers with at most 1 mismatch.
Method 2: BWT Saves the Day Again

**Note:** this approach faces complications in practice. We will have to consider a huge number of strings at the beginning, most of which may be frivolous.

In practice, we may require that a suffix of length $m$ matches the string exactly to reduce the number of strings we have to consider initially.
Citations

---

**[PDF] A block-sorting lossless data compression algorithm**


by AB Lossless - 1994 - Related articles


---

**Opportunistic data structures with applications - IEEE Conference**


by P Ferragina - 2000 - Cited by 1016 - Related articles

Opportunistic data structures with applications. Abstract: We address the issue of compressing and indexing data. We devise a data structure whose space occupancy is a function of the entropy of the underlying data set.

---

**Fast and accurate short read alignment with Burrows–Wheeler transform**

[https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2705234/](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2705234/)

by H Li - 2009 - Cited by 17996 - Related articles

May 18, 2009 - Results: We implemented Burrows-Wheeler Alignment tool (BWA), a new read alignment package that is based on backward search with ...

---

**Ultrafast and memory-efficient alignment of short DNA sequences to**


by B Langmead - 2009 - Cited by 13797 - Related articles

Mar 4, 2009 - *Bowtie* is an ultrafast, memory-efficient alignment program for aligning short DNA sequence reads to large genomes. For the human genome, ...