Algorithms in Bioinformatics
Spring 2023
Lecture 3

Eleazar Eskin

University of California, Los Angeles

Online Textbook

- Need to be part of our course instance of online textbook at Stepik website (see Bruin Learn for link). Should say "Bioinformatics Algorithms @UCLA 2023"
- If have access to online book but not to our course instance fill out form (see Bruin Learn link) by end of today Tue 4/11
- If you have purchased print book upload receipt to form (see Bruin Learn link) by end of today Tue 4/11

Assignments

- HW1 due 12pm 4/13
- Posting of Question on paper 1 due 12pm 4/13
- HW2 Chapter 9 due 12pm 4/18
- All Homeworks are posted

- See course website for full set of announcements

Abstract Data Types

Lecture 3.

April 11th, 2023

Abstract Data Type (ADT)

- Mathematical model for data types
- Point of view from the “user”

- Example: Sequence Index
- Operations:
  - **CreateIndex(Sequence) -> returns sequenceIndex**
  - **IdentifyPositions(Substring) -> returns list of positions**
- Implementations:
  - **Trivial Algorithms (loops through sequence)**
  - **Hash Table / Dictionary**
  - **Burrough Wheeler Transform**

IMPORTANT ADVICE!!!!!

- Separate ADT implementation from problem

1. Solve problems using abstract data types
2. Initially implement trivial ADT and test on small data
3. Later, implement more efficient ADT

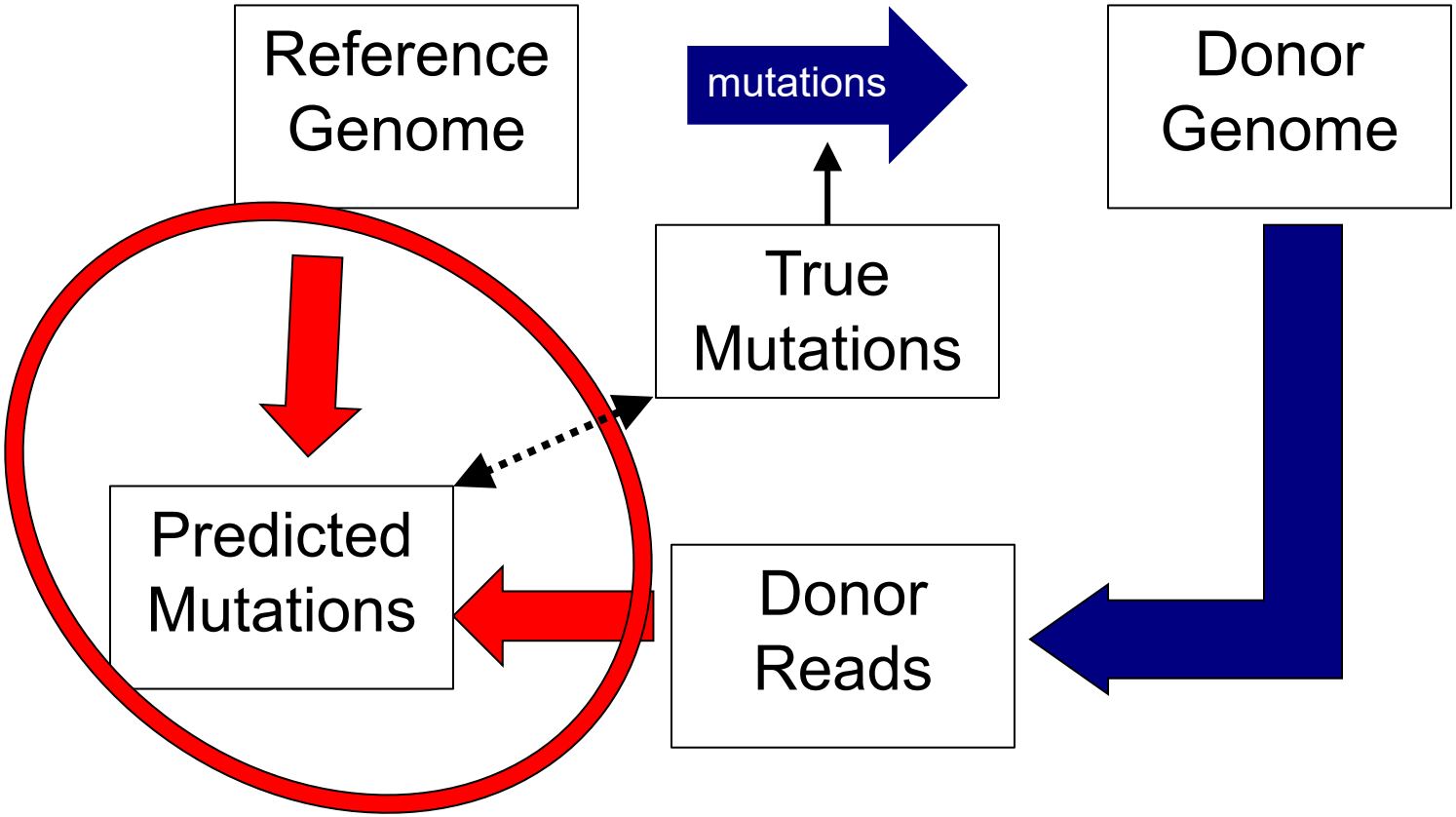
- Changes in ADT implementation should not affect performace

Read Mapping Project

Lecture 3.

April 11th, 2023

Read Mapping Project



- Project predicts mutations using donor reads and reference genome
- Evaluated using true mutations

Mutations Format

- “>” then reference position
- S – Substitution (original new)
- I – Insertion (new)
- D – Deletion (deleted original)

- Also format for prediction

>S125 A A
>S369 C T
>S625 C C
>S630 G A
>S812 T A
>S841 T T
>S845 G C
>S880 A T
>S937 A A
>I447 G
>D633 G

Reference/Read Format

- Reference:

>genome_1000
TCCCTACACTTGGCGATTGAACGGAGACACTGTTCATGCCACCCCGTGCCCTAGCCTGCTTTACCTTGCTGGCGCCCCCT
CACTTAAATTATAATCTTAGCCCCTTCTCCTCTCCCAGTCGTGTCAGCGTTTTCGGTGAGGACCCGGGGTAAGTTCACGT
GCGTTGTCTAACTTGAAACAACTTTCTTTCTTGCCGCATCCGTACTCATTCGCAGTCGCTGTGTCTCTAACCCAACTTCC
TAAGTGCTTGCAGCTAAATCTGAACAGCATTGCCTATTTCTCAGTTAATCTAGCAGTTTAGGTAAGTTAGTACCACTTCC

- Reads:

>read_0
GGTAAGTTAGTACCACTTCCAATAACAAGCTGATAGACATGGACTTGAAC
>read_1
CAAGATTCGGTATCTTACAAACCTTTCCCGAATTGTTCGATTTGGGACAC
>read_2
CGAAGTTCTTGCCGCGCATTTGCGAGAAGCAGATAGAGCGACTCCTGGCT
>read_3
GAATCTGTCACTCCGATCGATAGCTTATTGCCCAACACGATCCCGGTCGG

Paired Reads

- 2 reads separated by ”insert”
- ”insert” size is drawn from a distribution
- Paired Reads:

```
>read_0/1
TTCCTAAGTGCTTGCAGCTAAATCTGAACAGCATTGCCTATTTCTCAGTT
>read_0/2
ATAGACATGGACTTGAACGATTTTAGACAGATATACCTCGATGTAAGGGA
>read_1/1
TTCCCAGCTGCAAACGATTTGATTCGCTGTCAGGTTACGTCAACGCGGGA
>read_1/2
GCGGCCGTAATTGTCATAGCAACGTATGTTTGCCGGCAGTCGTAATCTCT
```

Simulator Details

- Mutation Process
  - **Substitution Rate**
  - **Insert Rate**
  - **Deletion Rate**
- Read sampled uniformly from genome
- Insert sizes drawn uniformly
  - **Minimum Insert**
  - **Maximum Insert**
- Error rates for reads
  - **Substitution Rate**
  - **Insert Rate**
  - **Deletion Rate**

Read Mapping Sample Genome

- Small genome is available with correct answers
- Reads are available in 4 formats:
  - **Single no errors**
  - **Single with errors**
  - **Paired no errors**
  - **Paired with errors**

Read Mapping Assignment

- Starter – 10,000 length genome
- Assignment
  - **1,000,000 length genome for undergrad**
  - **100,000,000 length genome for grad (or extra credit)**
  - **10,000,000,000 length genome (extra credit)**
- Scoreboard will evaluate F-score for substitutions, insertions, deletions
- Other features are extra credit

Extra Credit Read Mapping

- Longer Insertions/Deletions
  - **Longer than 1 position but smaller than read**
  - **Longer than read – paired end are important**
- Inversion
  - **Paired read signature**
- Copy Number Variation / Repeats
  - **Coverage signal**
  - **Paired ends give clues to where the copies are**
- Alu Repeats
- Short Tandem Repeats

Insertions and Deletions

Lecture 3.

April 11th, 2023

"Re"-Sequencing: Insertions

My Genome:

TACATGAGATCCACAT**A**GAGATCTGTAGAGCTGTGAGATC

A Sequence Read:

CCACATAGAGATCTGTAGAGCTGT

The Human Genome:

TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC

CCACATAGAGATCTGTAGAGCTGT

TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC

CCACATAGAGATCTGTAGAGCTGT

How do we deal with this case?

"Re"-Sequencing: Insertions

My Genome:

TACATGAGATCCACATAGAGATCTGTAGAGCTGTGAGATC

A Sequence Read:

CCACATAGAGATCTGTAGAGCTGT

The Human Genome:

TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC
CCACATAGAGATCTGTAGAGCTGT

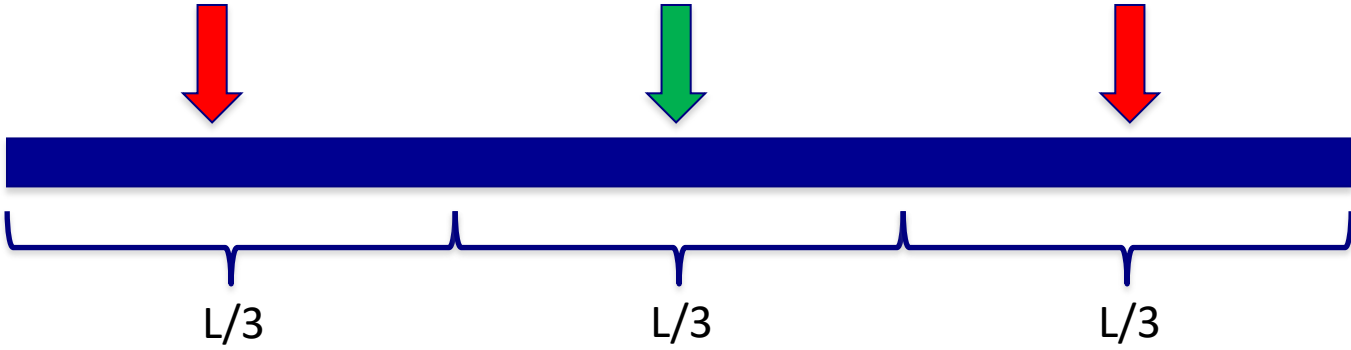
TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC
CCACATAGAGATCTGTAGAGCTGT

Solution: Add Insertion to the Human Genome

TACATGAGATCCACAT-GAGATCTGTAGAGCTGTGAGATC
CCACATAGAGATCTGTAGAGCTGT

Indel in Middle of Read

If indel is in the middle of read.



- Both outside regions of size L/3 will match perfectly.
- Because of coverage, indel will be in middle at least for one read.
- Important: Middle distance will be L/3+1 or L/3-1

"Re"-Sequencing: Insertions

My Genome:

TACATGAGATCCACATAGAGATCTGTAGAGCTGTGAGATC

A Sequence Read:

CCACATAGAGATCTGTAGAGCTGT

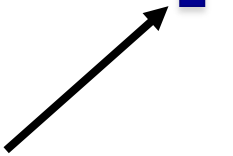
The Human Genome:

TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC

CCACATAGAGATCTGTAGAGCTGT

TACATGAGATCCACATGAGATCTGTAGAGCTGTGAGATC

CCACATAGAGATCTGTAGAGCTGT



Insertion Point

Indel Algorithms

- Trivial Algorithm
  - **Try all inerstion points for a read**
  - **If read matches (with insertion) below number of mismatches, then we desclare a match and identify and indel**
- More Efficient Algorithm
  - **Look for perfect match in first part of read**
  - **Try insertion point at point of first mismatch**
  - **More complicated but faster**
- More accurate Algorithm
  - **Perform alignment between read and reference**
- Extremely Accurate Algorithm
  - **Align all reads with indel together.**
  - **Multiple Sequence Alignment!**

“Re”-Sequencing + Burroughs Wheeler Transform

Lecture 3.

April 11th, 2023

(Some slides from Ben Langmead)

Index for L/3 (is BIG!)

- Intuition: Create an index (or phone book) for the genome.
- We can look up an entry quickly.

Sequence	Positions
AAAAAAAAAA	32453, 64543, 76335
AAAAAAAAAC	64534, 84323, 96536
AAAAAAAAAG	12352, 32534, 56346
AAAAAAAAAT	23245, 54333, 75464
AAAAAAAACA	
AAAAAAAACC	43523, 67543
…	
CAAAAAAAAA	32345, 65442
CAAAAAAAAC	34653, 67323, 76354
…	
TCGACATGAG	54234, 67344, 75423
TCGACATGAT	11213, 22323
…	
TTTTTTTTTG	64252
TTTTTTTTTT	64246, 77355, 78453

If L=30, each entry will have a key of length 10. Each entry will contain on average $N/4^{10}$ positions. (Approximately 3,000).

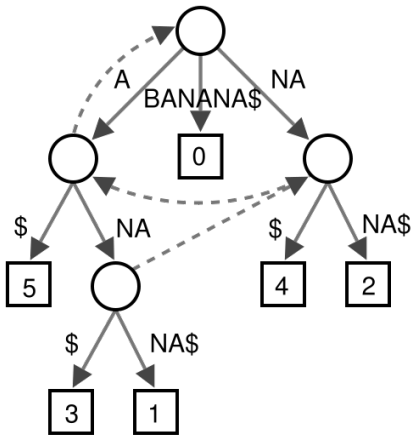
If L=45, each entry will have a key of length 15. Each entry will contain on average 3 positions.

Indexing a genome

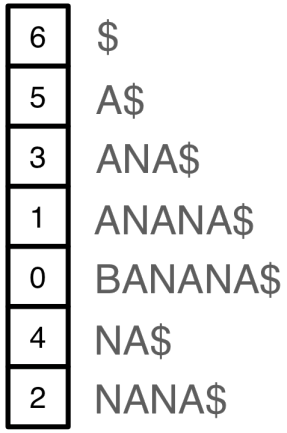
- To find exactly matching substrings, we need to build an index for the whole genome.
- Problem: The genome is BIG!

Indexing

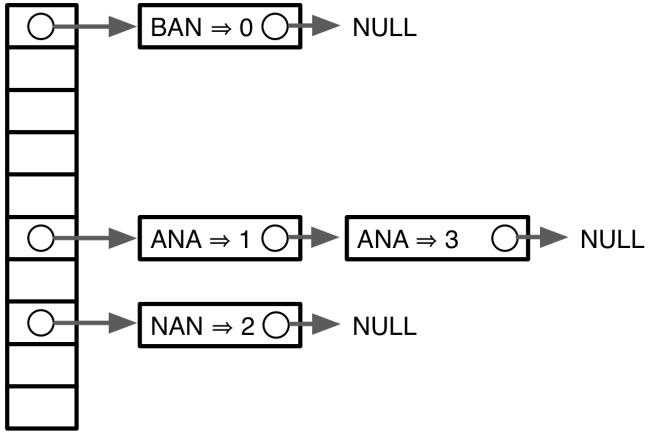
- Genome indices can be big. For human:



> 35 GBs



> 12 GBs



> 12 GBs

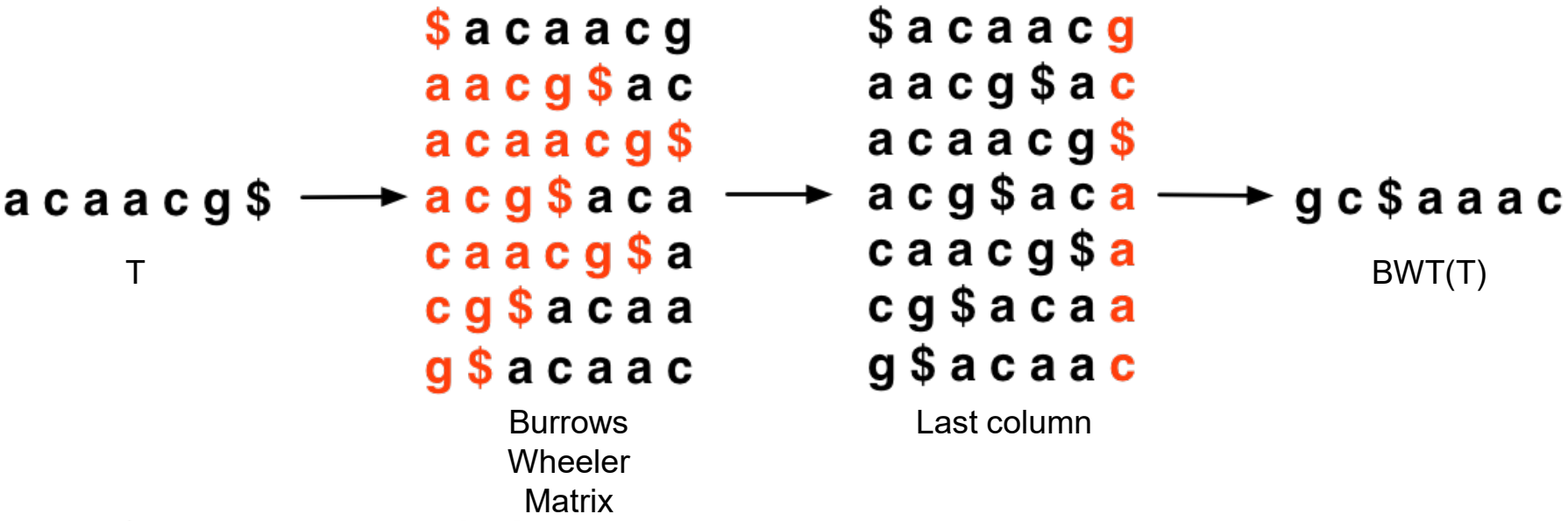
- Large memory requirement implications
  - **Requires large memory machine (expensive)**
  - **Partition genome and index each part (slow)**

Memory Efficient but Slow Algorithm

- Store just the sequence
  - **4 DNA bases per byte (2 bits each)**
  - **3,000,000,000 / 4 ~ 750 MB**
- When looking up string, just loop through the sequence.
- Very slow, but very memory efficient!

Burrows-Wheeler Transform

- http://en.wikipedia.org/wiki/Burrows-Wheeler_transform
- Reversible permutation used originally in compression



- Once BWT(T) is built, *all else shown here is discarded*
  - **Matrix will be shown for illustration only**

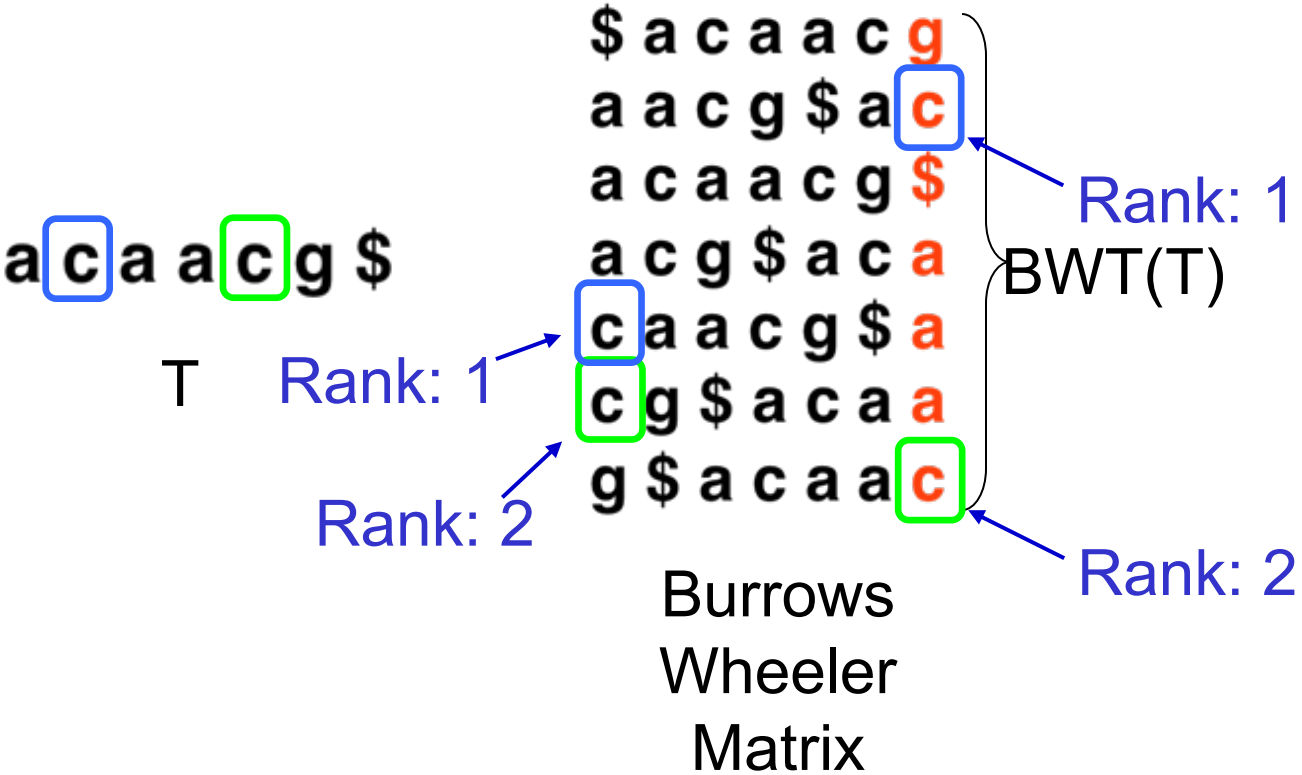
Burrows M, Wheeler DJ: A block sorting lossless data compression algorithm. *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994

Burrows-Wheeler Transform

- Store only last column
- First column can be recovered by counting symbols in last column because it is sorted

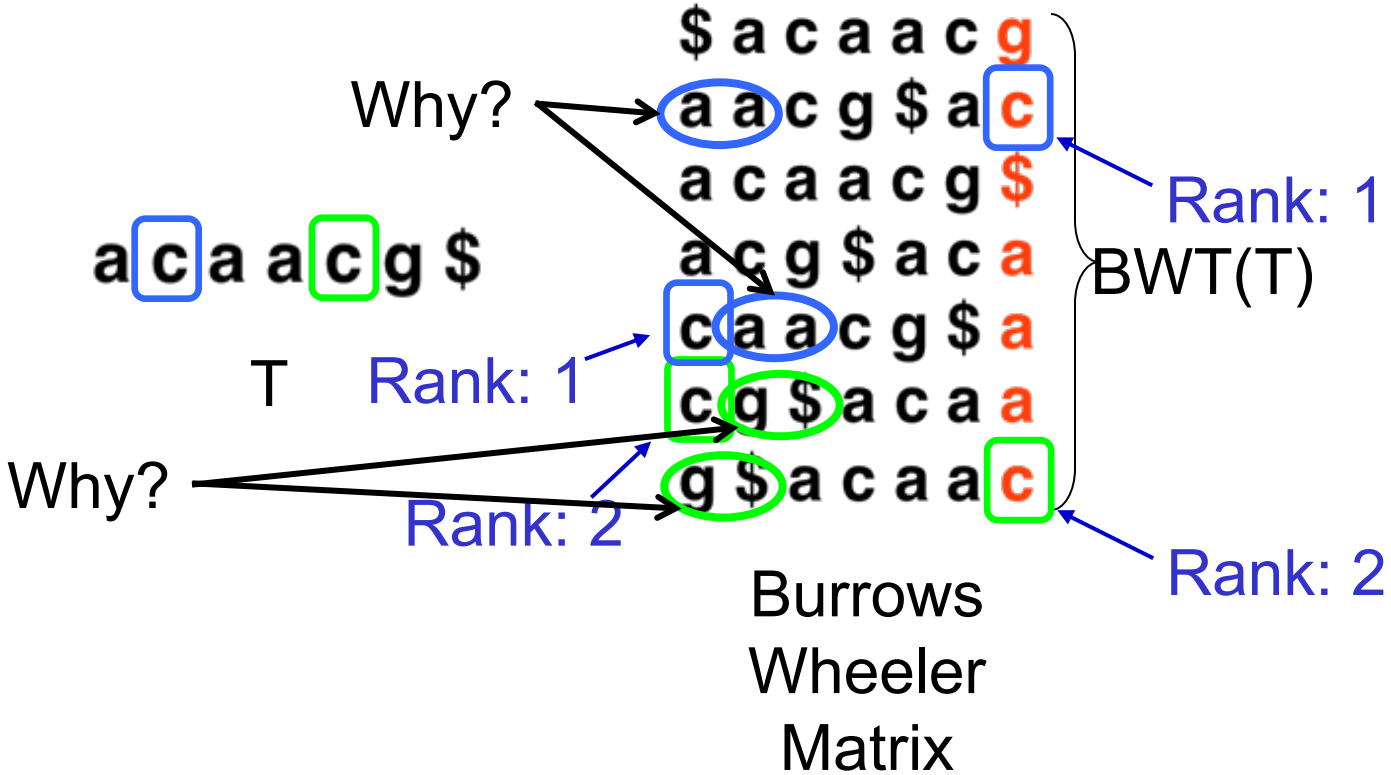
Burrows-Wheeler Transform

- Property that makes BWT(T) reversible is "LF Mapping"
  - **i^{th} occurrence of a character in Last column is same *text* occurrence as the i^{th} occurrence in First column**



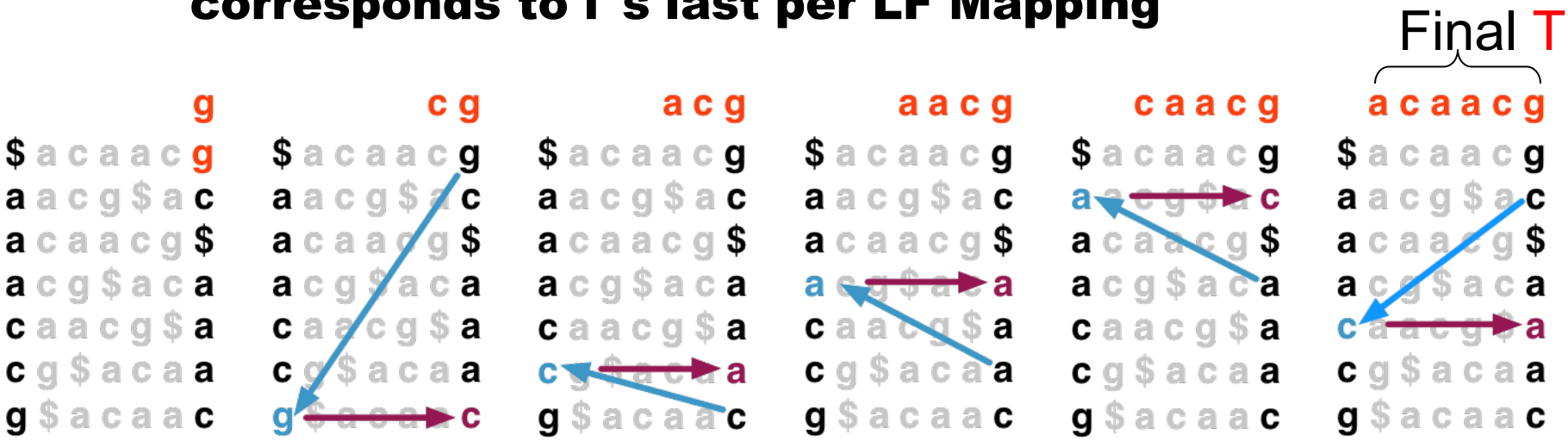
Burrows-Wheeler Transform

- Property that makes BWT(T) reversible is "LF Mapping"
  - **i^{th} occurrence of a character in Last column is same *text* occurrence as the i^{th} occurrence in First column**



Burrows-Wheeler Transform

- To recreate T from BWT(T), repeatedly apply rule:
  - **T = BWT[LF(i)] + T; i = LF(i)**
    - **Where LF(i) maps row i to row whose first character corresponds to i's last per LF Mapping**



- Could be called "unpermute" or "walk-left" algorithm

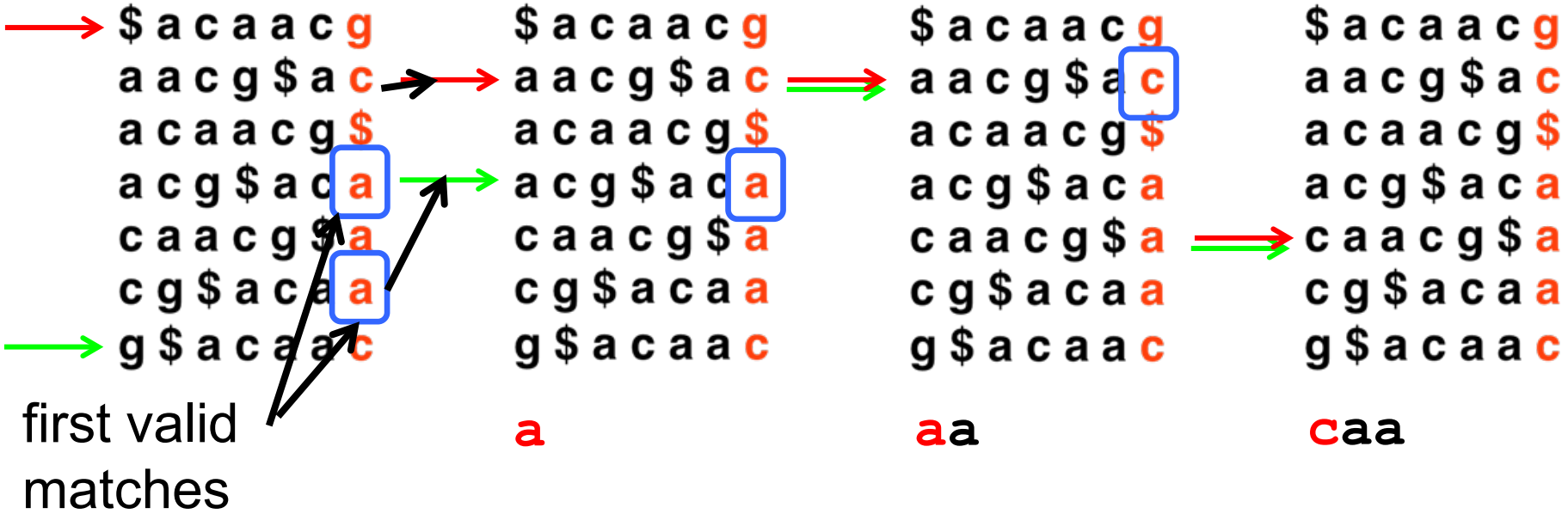
FM Index

- Ferragina & Manzini propose "FM Index" based on BWT
- Observed:
  - **LF Mapping also allows *exact matching* within T**
  - **LF(i) can be made fast with *checkpointing***
  - **...and more (see FOCS paper)**

- Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*
- Ferragina P, Manzini G: An experimental study of an opportunistic index. *SIAM symposium on Discrete algorithms*. Washington, D.C.; 2001.

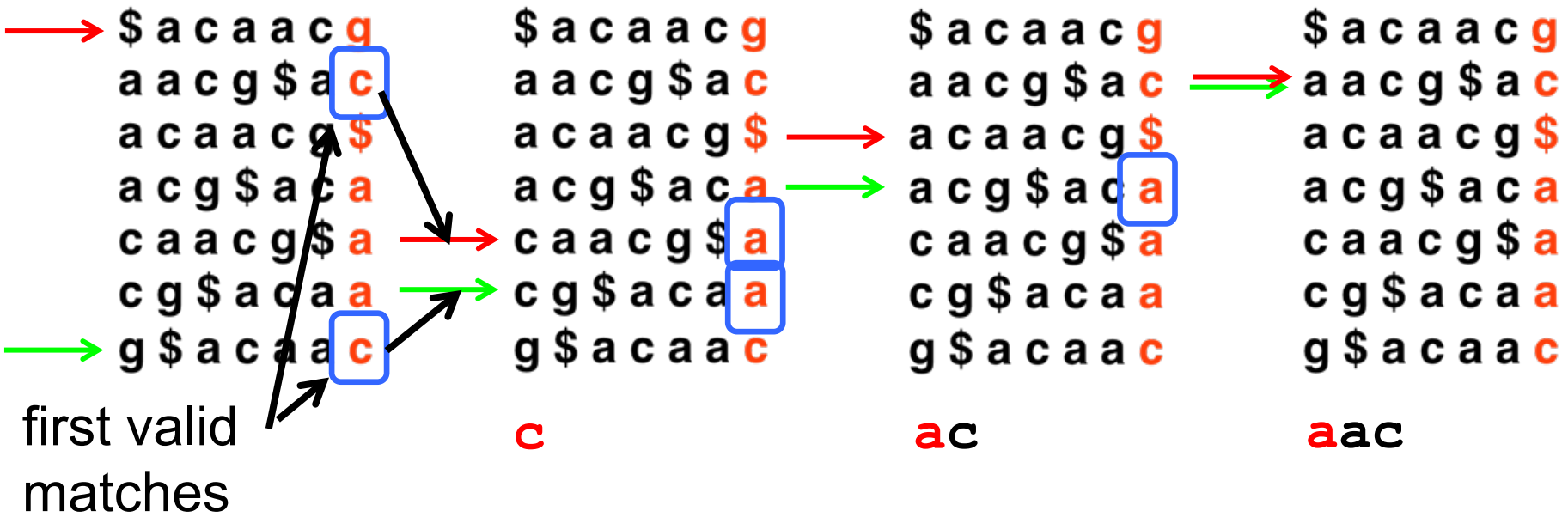
Exact Matching with FM Index

- Look up pattern in reverse.
- Use 2 pointers to represent range of matches.
- Find first valid match for next symbol in range.
  - **Example: searching for "caa"**



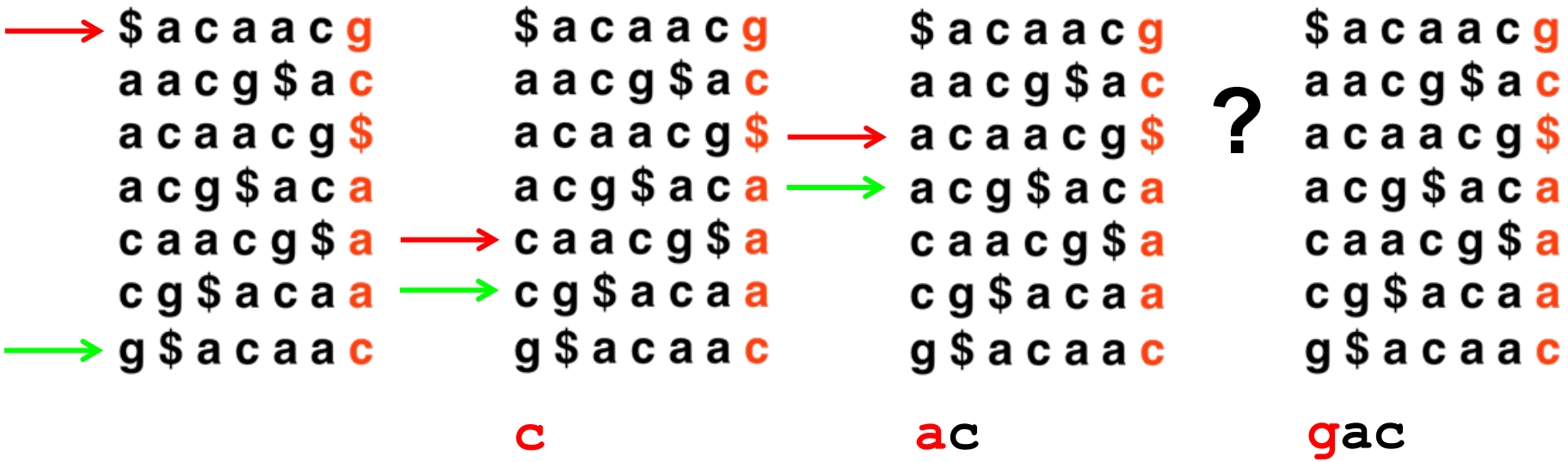
Exact Matching with FM Index

- Look up pattern in reverse.
- Use 2 pointers to represent range of matches.
- Find first valid match for next symbol in range.
  - **Example: searching for "aac"**



Exact Matching with FM Index

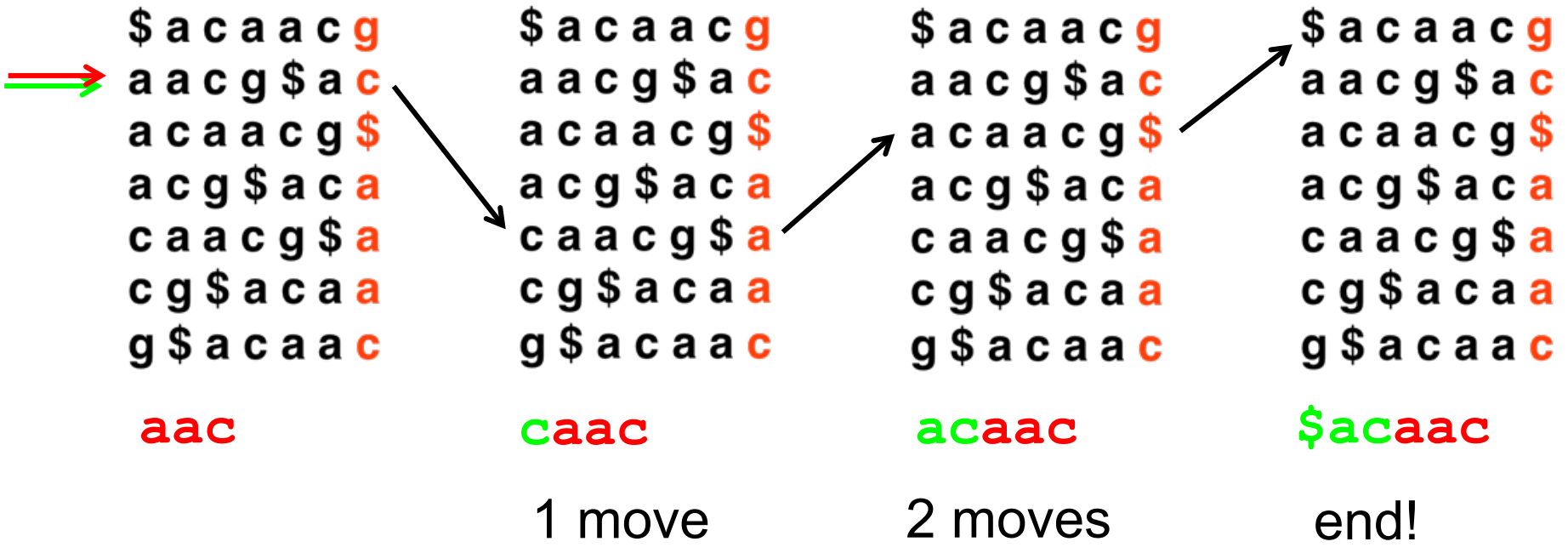
- If no match…
  - **Example: searching for "gac"**



- Pointers will get lost.
- FM index can quickly check for a match.

Where in sequence is the match?

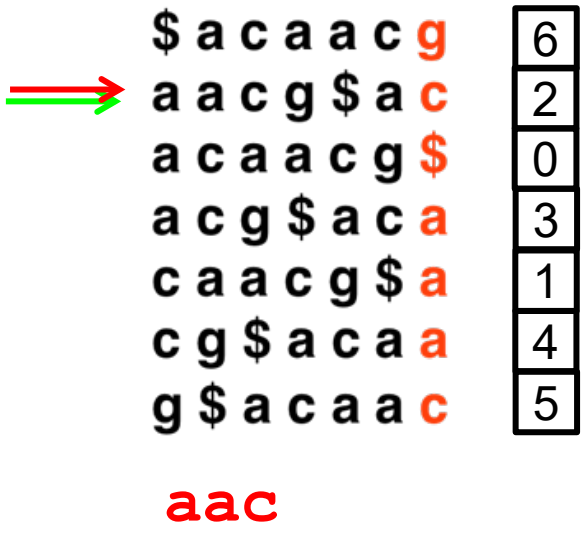
- Use “walk-left” to build sequence to start
- Count number of sequences
  - **Example: searching for “aac”**



- Number of moves back is start position of match
  - **Example: “aac” is in position 2.**

Where in sequence is match?

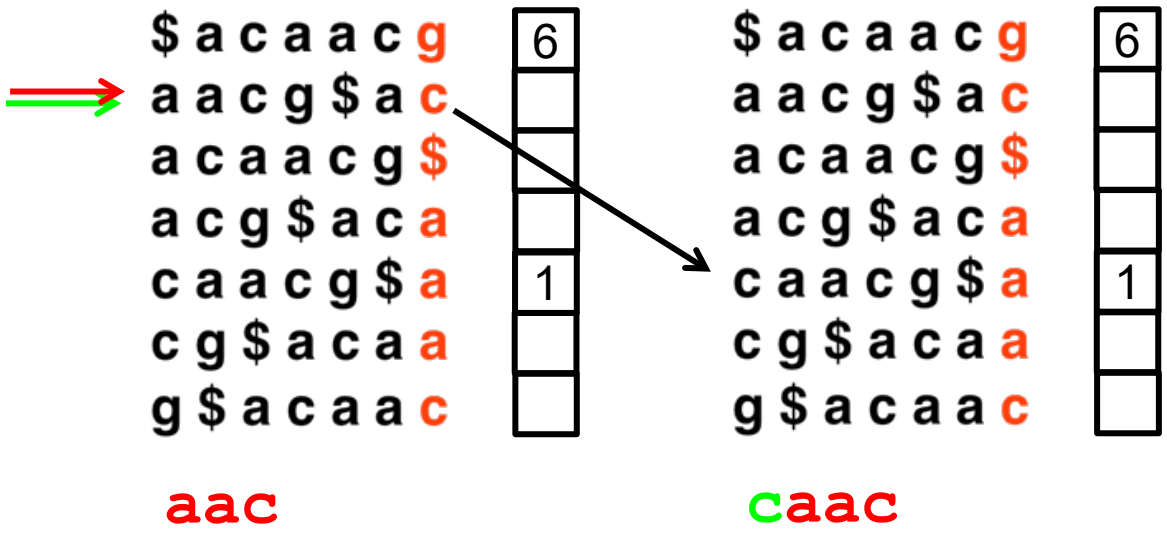
- "walk-left" to start of sequence is slow
- Requires on average N/2 steps to reach start.
- Alternate strategy: keep index of positions.
  - **Example: searching for "aac"**



- Problem: requires as much storage as hashtable!

Where in sequence is match?

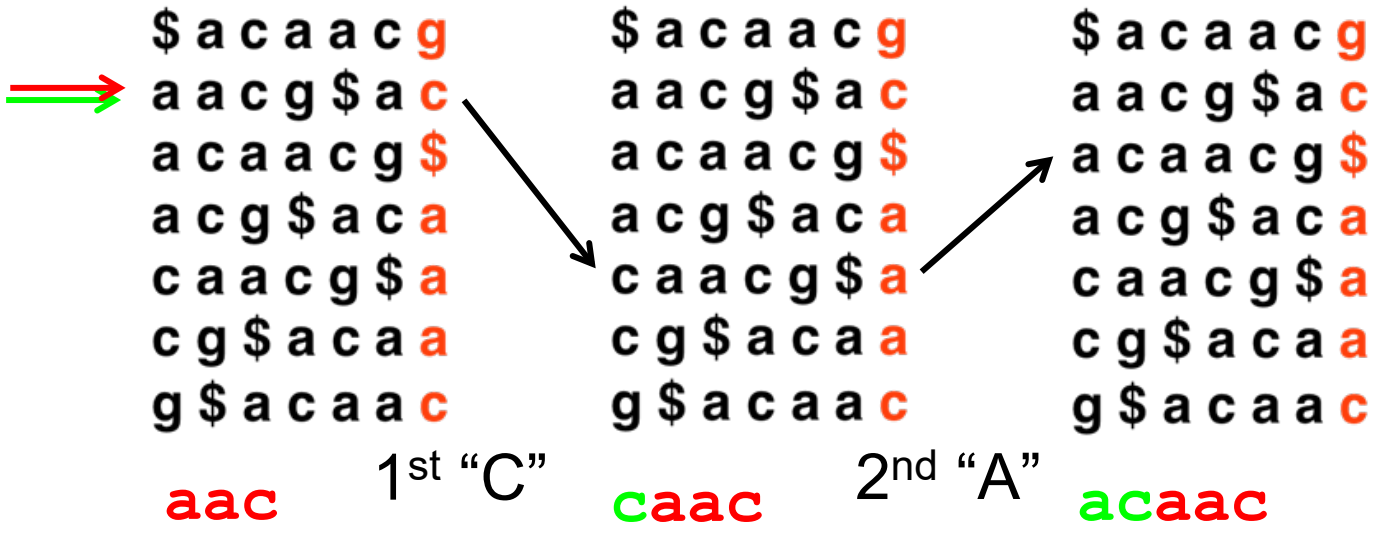
- Key Idea: Store fraction of array (sampling)
- Only store some positions and “walk-left”
- Combines two previous strategies
  - **Example: searching for “aac”**



- How many values to store provides defines time/space tradeoff.

“walk-left” optimization

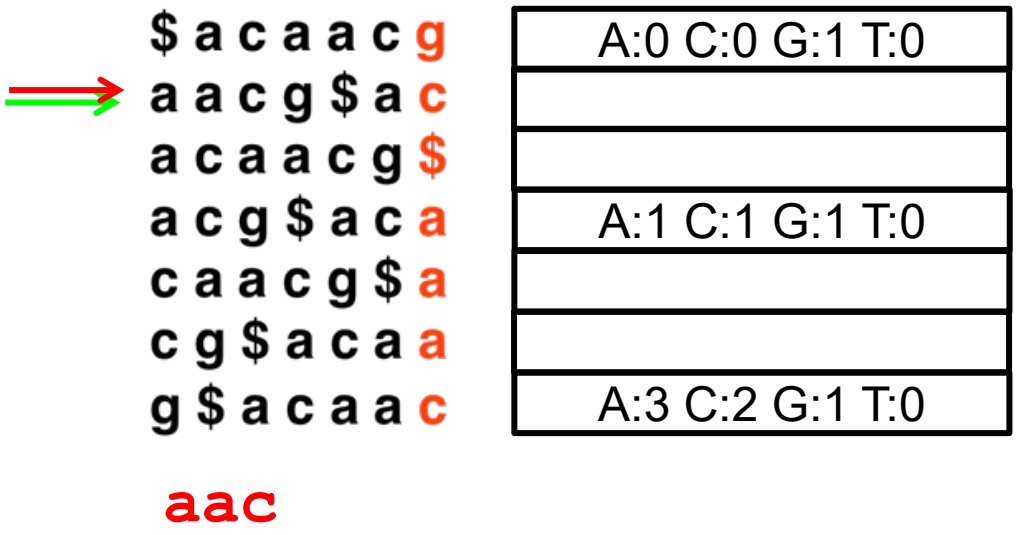
- Each “walk-left” requires counting previous occurrences of symbol in BWT
  - **Example: searching for “aac”**



- Requires counting occurrences in N/2 length string
- Really slow!

“walk-left” optimization

- Idea: use checkpoints to store previous counts
  - **Example: searching for “aac”**



- Requires counting occurrences only until checkpoint.
- Really fast!

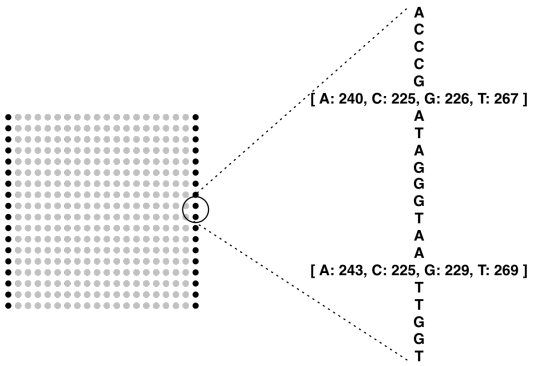
FM Index is Small (Bowtie)

- Entire FM Index on DNA reference consists of:
  - **BWT (same size as T)**
  - **Checkpoints (~15% size of T)**
  - **SA sample (~50% size of T)**
- Total: ~1.65x the size of T

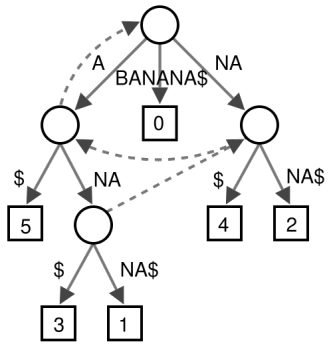
Assuming 2-bit-per-base encoding and no compression, as in Bowtie

Assuming a 16-byte checkpoint every 448 characters, as in Bowtie

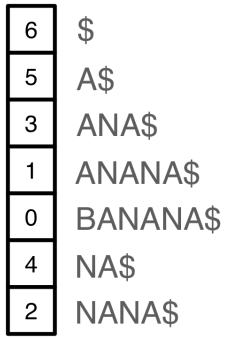
Assuming Bowtie defaults for suffix-array sampling rate, etc



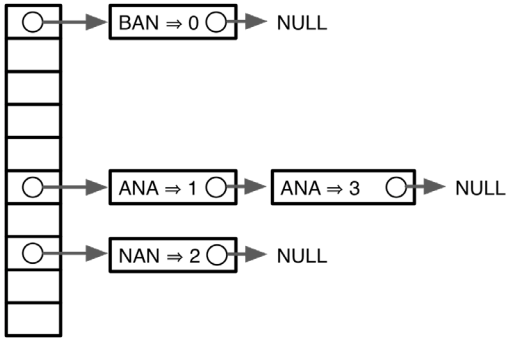
~1.65x



>45x



>15x



>15x

Reference Paper

- Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology 10:R25.
  - (Some slides from paper)