CS 600.226: Data Structures

Michael Schatz

Nov 14, 2016

Lecture 32: BWT

HW8

Assignment 8: Competitive Spelling Bee

Out on: November 2, 2018

Due by: November 9, 2018 before 10:00 pm

Collaboration: None

Grading:

- Packaging 10%,
- Style 10% (where applicable),
- Testing 10% (where applicable),
- Performance 30% (where applicable),
- Functionality 40% (where applicable)

Overview

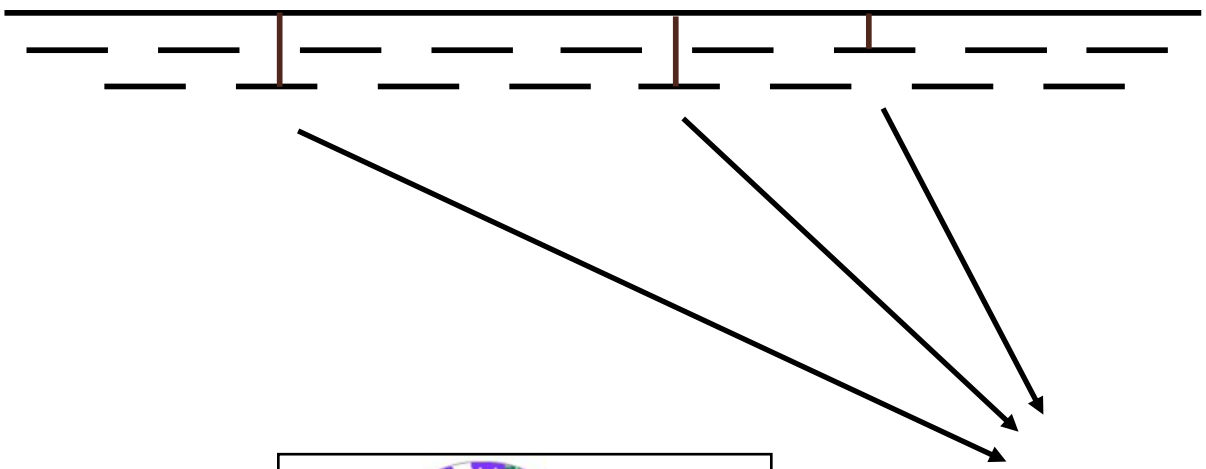
Your "one" task for this assignment is to take the simple spell checker we give you and to turn it into the fastest, most memory-efficient spell checker in the course, subject to the constraints detailed below. You are expected to do this by (once again) implementing the Map interface, this time using one of several hash table techniques (your choice, see below).

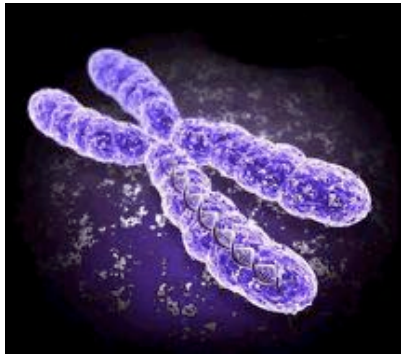
Remember: javac –Xlint:all & checkstyle *.java & Junit & Jaybee BenchMarks

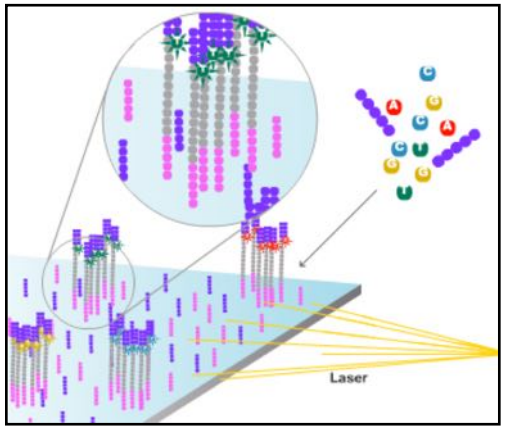
Part 1: Suffix Arrays

Personal Genomics

How does your genome compare to the reference?







Heart Disease

Cancer

Presidential smile

Brute Force Analysis

- Brute Force:
  - At every possible offset in the genome:
    - Do all of the characters of the query match?

- Analysis
  - Simple, easy to understand
  - Genome length = n [3B]
  - Query length = m [7]
  - Comparisons: (n-m+1) * m [21B]

- Overall runtime: O(nm)

[How long would it take if we double the genome size, read length?]

[How long would it take if we double both?]

Brute Force Reflections

Why check every position?

– GATTACA can't possibly start at position 15 [WHY?]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	T	A	C	A	

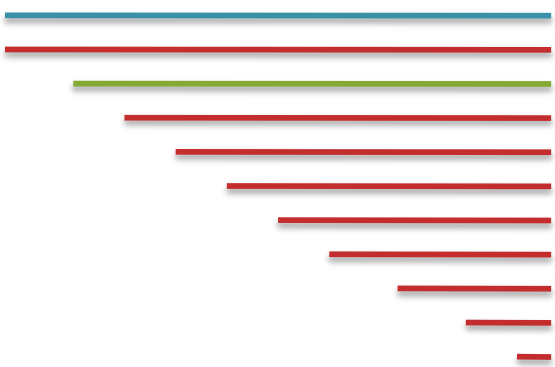
– Improve runtime to O(n + m) [3B + 7]
  - If we double both, it just takes twice as long
  - Knuth-Morris-Pratt, 1977
  - Boyer-Moyer, 1977, 1991

– For one-off scans, this is the best we can do (optimal performance)
  - We have to read every character of the genome, and every character of the query
  - For short queries, runtime is dominated by the length of the genome

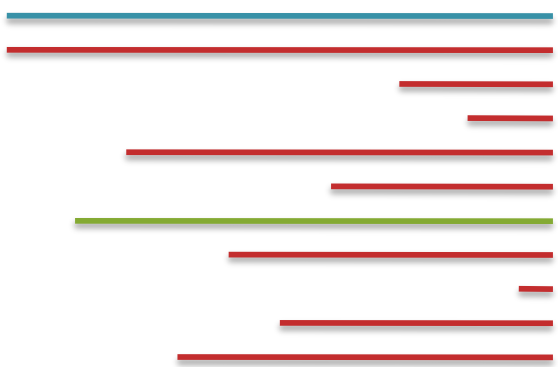
Suffix Arrays: Searching the Phone Book

- What if we need to check many queries?
  - We don't need to check every page of the phone book to find 'Schatz'
  - Sorting alphabetically lets us immediately skip 96% (25/26) of the book *without any loss in accuracy*

- Sorting the genome: Suffix Array (Manber & Myers, 1991)
  - Sort every suffix of the genome



Split into n suffixes



Sort suffixes alphabetically

[Challenge Question: How else could we split the genome?]

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15;

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

Lo →

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15;

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Lo →

Hi →

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
Hi →	15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11;

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
Hi →	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
	15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
Hi →	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
	15	TTACC…	11

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 9;

#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Lo
Hi

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15; Mid = (1+15)/2 = 8
  - Middle = Suffix[8] = CC
    => Higher: Lo = Mid + 1

  - Lo = 9; Hi = 15; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 9; Mid = (9+9)/2 = 9
  - Middle = Suffix[9] = GATTACA…
    => Match at position 2!

	#	Sequence	Pos
	1	ACAGATTACC…	6
	2	ACC…	13
	3	AGATTACC…	8
	4	ATTACAGATTACC…	3
	5	ATTACC…	10
	6	C…	15
	7	CAGATTACC…	7
	8	CC…	14
Lo → Hi →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
	15	TTACC…	11

Binary Search Analysis

- Binary Search

  Initialize search range to entire list

  mid = (hi+lo)/2; middle = suffix[mid]

  if query matches middle: done

  else if query < middle: pick low range

  else if query > middle: pick hi range

  Repeat until done or empty range [WHEN?]

- Analysis
  - More complicated method
  - How many times do we repeat?
    - How many times can it cut the range in half?
    - Find smallest x such that: $n/(2^x) \leq 1$; $x = \lg_2(n)$ [32]

- Total Runtime: O(m lg n)
  - More complicated, but much faster!
  - Looking up a query loops 32 times instead of 3B

[How long does it take to search 6B or 24B nucleotides?]

Binary Search Analysis

- Binary Search

  Initialize search range to entire list

  mid = (hi+lo)/2; middle = suffix[mid]

  if query matches middle: done

  else if query < middle: pick low range

  else if query > middle: pick hi range

  Repeat until done or empty range [WHEN?]

- Analysis
  - More complicated method
  - How many times do we repeat?
    - How many times can it cut the range in half?
    - Find smallest x such that: $n/(2^x) \leq 1$; $x = \lg_2(n)$ [32]

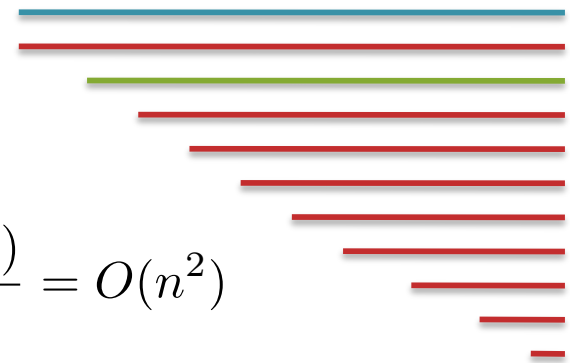
- Total Runtime: O(m lg n)
  - More complicated, but much faster!
  - Looking up a query loops 32 times instead of 3B

Can be reduced to O(m + lg n)
using an auxiliary data structure called the LCP array

Suffix Array Construction

- How can we store the suffix array?

  [How many characters are in all suffixes combined?]



$$S = 1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

  - Hopeless to explicitly store 4.5 billion billion characters
  - Instead use implicit representation
    - Keep 1 copy of the genome, and a list of sorted offsets
    - Storing 3 billion offsets fits on a server (12GB)

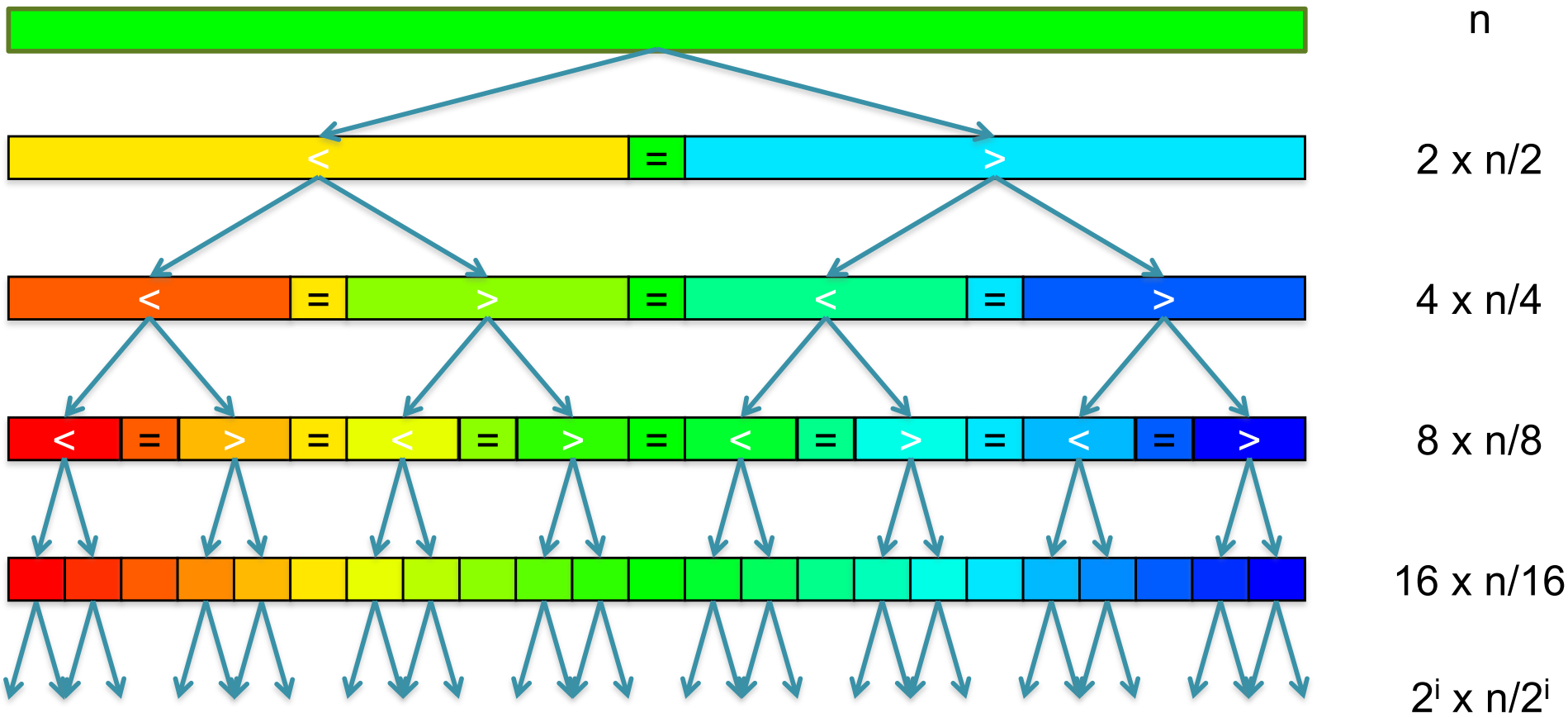
- Searching the array is very fast, but it takes time to construct
  - This time will be amortized over many, many searches
  - Run it once "overnight" and save it away for all future queries

Pos
6
13
8
3
10
15
7
14
2
9
5
12
1
4
11

TGATTACAGATTACC

Divide and Conquer

- Selection sort is slow because it rescans the entire list for each element
  - How can we split up the unsorted list into independent ranges?
  - Hint 1: Binary search splits up the problem into 2 independent ranges (hi/lo)
  - Hint 2: Assume we know the median value of a list



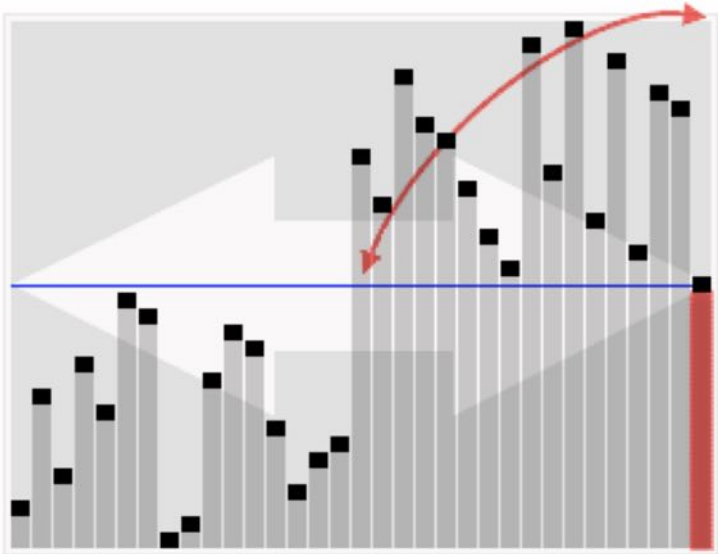
[How many times can we split a list in half?]

QuickSort Analysis

```
QuickSort(Input: list of n numbers)
// see if we can quit
if (length(list)) <= 1): return list

// split list into lo & hi
pivot = median(list)
lo = {}; hi = {};
for (i = 1 to length(list))
if (list[i] < pivot): append(lo, list[i])
else:                 append(hi, list[i])

// recurse on sublists
return (append(QuickSort(lo), QuickSort(hi))
```



http://en.wikipedia.org/wiki/Quicksort

Analysis (Assume we can find the median in O(n))

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases}$$

$$T(n) = n + 2(\frac{n}{2}) + 4(\frac{n}{4}) + \cdots + n(\frac{n}{n}) = \sum_{i=0}^{lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{lg(n)} n = O(n \lg n)$$

Implicit Suffix Comparison (1)

```
public class CompareSuffixes {
  // Compare two suffixes of text starting at s1 and s2
  // Return -1 if s1 is smaller, +1 if s2 is smaller
  public static int cmpSuffixes(String text, int s1, int s2) {
    if (s1 == s2) { return 0; }

    int d = 0;
    while ((s1 + d < text.length()) &&
           (s2 + d < text.length())) {
      char c1 = text.charAt(s1+d);
      char c2 = text.charAt(s2+d);

      if      (c1 < c2) { return -1; }
      else if (c2 < c1) { return +1; }

      // else they are the same, keep going
      d++;
    }

    // no differences through end of string
    // return shorter one, meaning having a bigger offset
    if (s1 < s2) { return +1; }
    return -1;
  }
```

Implicit Suffix Comparison (2)

```
public static void main(String [] args) {
    if (args.length < 3) {
      System.err.println("usage: CompareSuffixes text s1 s2\n");
      return;
    }

    String text = args[0];
    int s1 = Integer.parseInt(args[1]);
    int s2 = Integer.parseInt(args[2]);

    // Show the suffixes for display purposes
    System.out.format("Comparing the suffixes of \"%s\"\n", text);
    System.out.format("s1 [%d]: \"%s\"\n", s1, text.substring(s1));
    System.out.format("s2 [%d]: \"%s\"\n", s2, text.substring(s2));

    int cmp = cmpSuffixes(text, s1, s2);
    System.out.println("Returned: " + cmp);

    if (cmp < 0) {
      System.out.format("s1 \"%s\" < s2 \"%s\"\n",
                        text.substring(s1),text.substring(s2));
    } else if (cmp > 0) {
      System.out.format("s2 \"%s\" < s1 \"%s\"\n",
                        text.substring(s2), text.substring(s1));
    } else {
      System.out.format("s1 \"%s\" == s2 \"%s\"\n",
                        text.substring(s1), text.substring(s2));
    }
}
```

CompareSuffixes

```
$ java CompareSuffixes MichaelSchatz 0 1
Comparing the suffixes of "MichaelSchatz"
s1 [0]: "MichaelSchatz"
s2 [1]: "ichaelSchatz"
Returned: -1
s1 "MichaelSchatz" < s2 "ichaelSchatz”

$ java CompareSuffixes MichaelSchatz 10 4
Comparing the suffixes of "MichaelSchatz"
s1 [10]: "atz"
s2 [4]: "aelSchatz"
Returned: 1
s2 "aelSchatz" < s1 "atz”

$ java CompareSuffixes AAAAAAAAAAAA 0 9
Comparing the suffixes of "AAAAAAAAAAAA"
s1 [0]: "AAAAAAAAAAAA"
s2 [9]: "AAA"
Returned: 1
s2 "AAA" < s1 "AAAAAAAAAAAA"
```

Exact Matching Review & Overview

Where is GATTACA in the human genome?

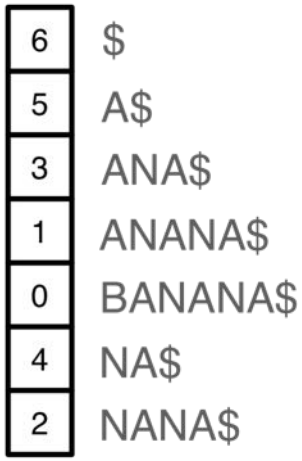
Brute Force (3 GB)

```
BANANA
BAN
  ANA
    NAN
      ANA
```

O(m * n)

Slow & Easy

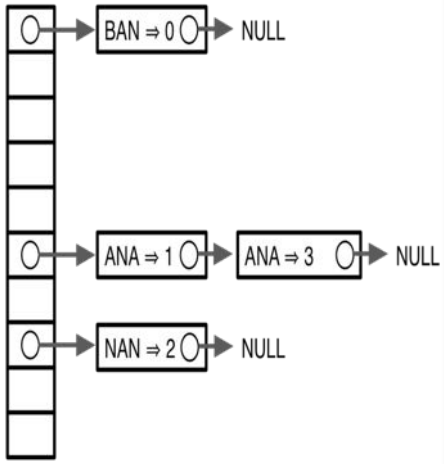
Suffix Array (>15 GB)



O(m + lg n)

Full-text index

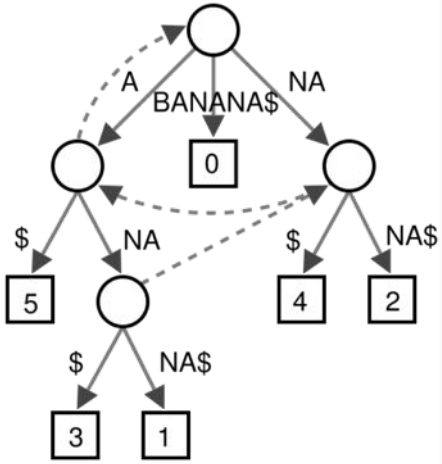
Hash Table (>15 GB)



O(1)

Fixed-length lookup

Suffix Tree (>51 GB)



O(m)

Full-text, but bulky

*** These are general techniques applicable to any text search problem ***

Part 2: Burrows Wheeler Transform

Algorithmic challenge

How can we combine the speed of a suffix array O(m + lg(n)) (or even O(m)) with the size of a brute force analysis (n bytes)?

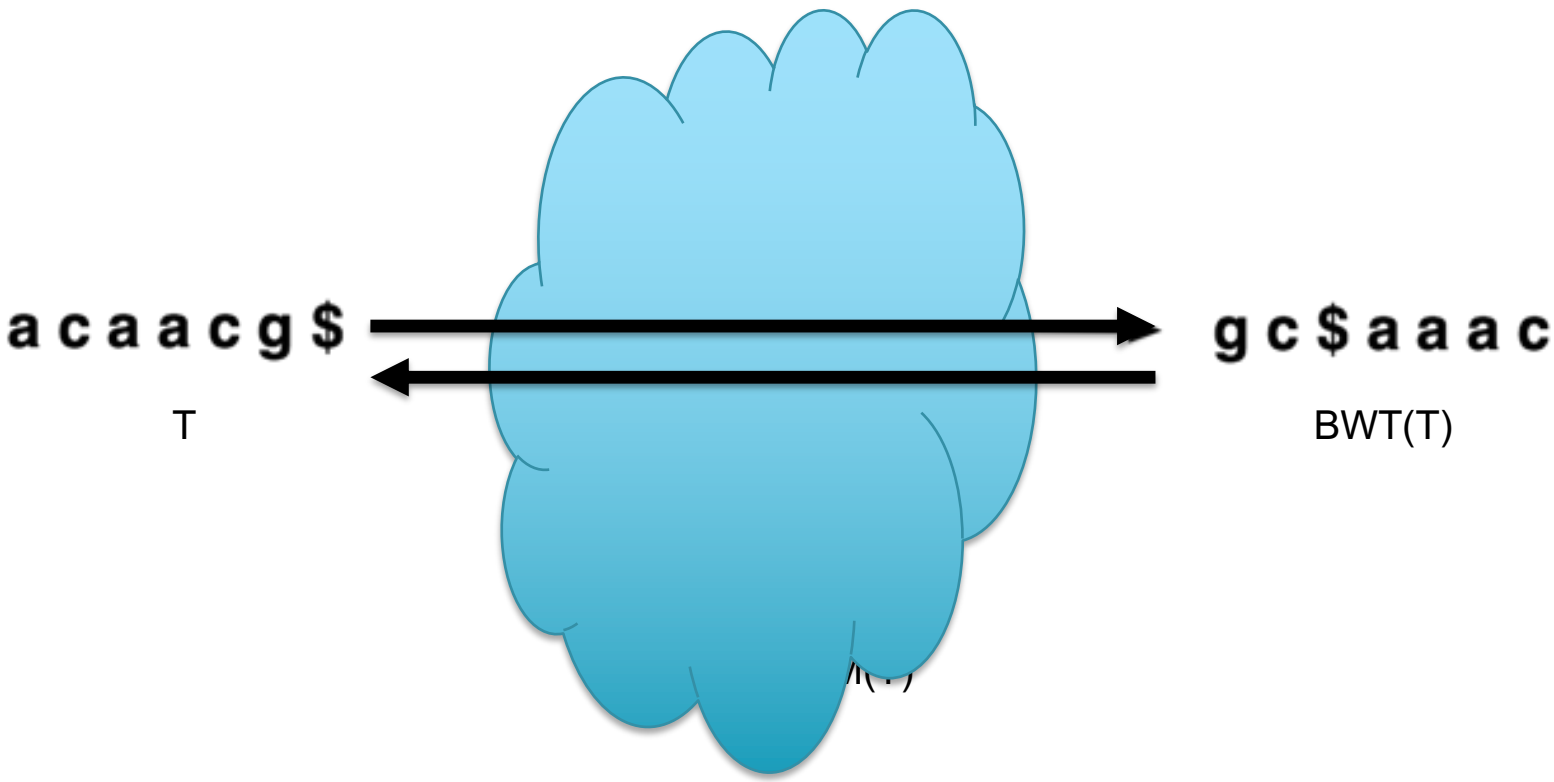
What would such an index look like?

Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

Slides Courtesy of Ben Langmead

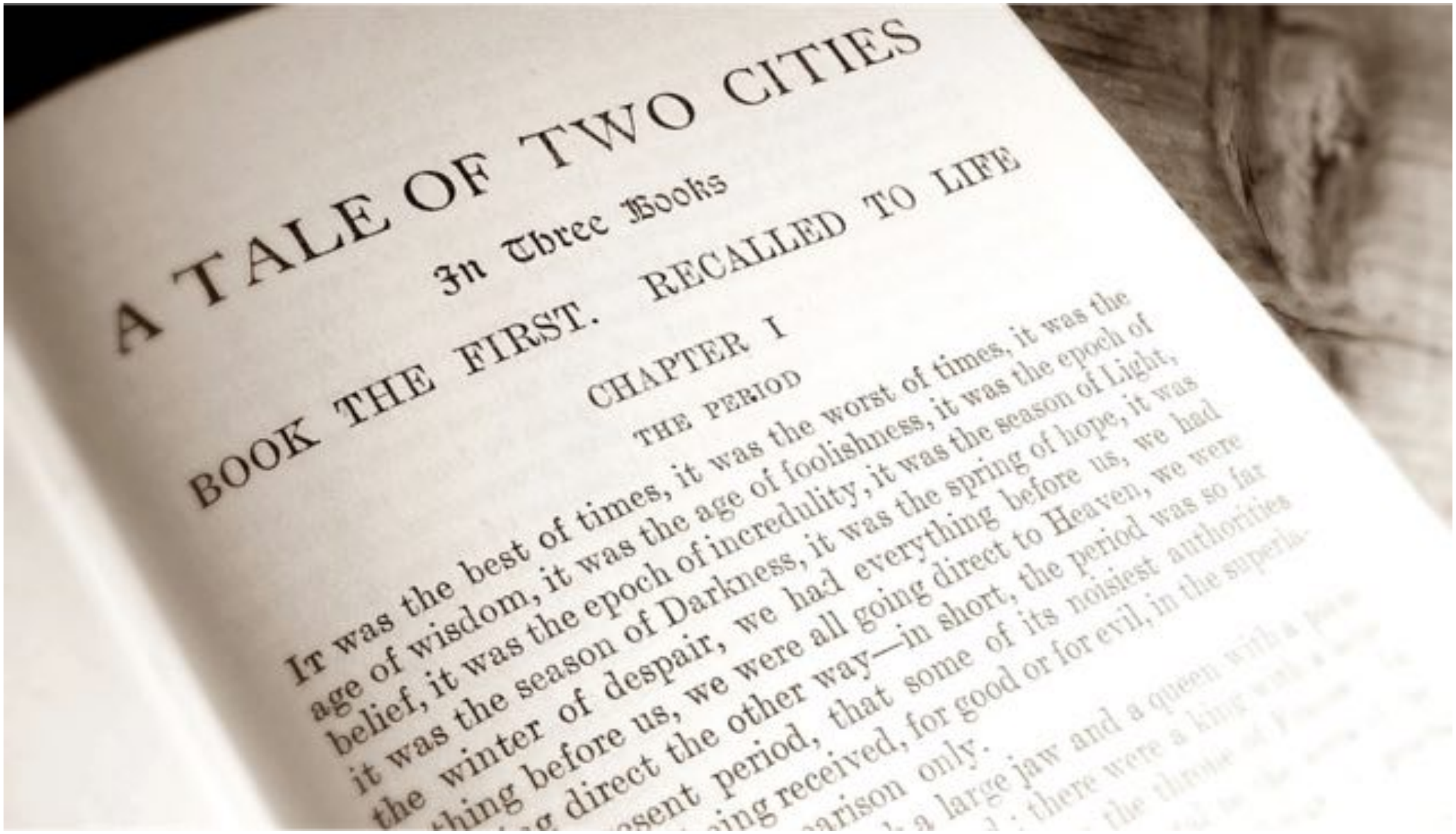
Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

Run Length Encoding



Run Length Encoding

```
ref[614]:
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

Run Length Encoding:

- Replace a "run" of a character X with a single X followed by the length of the run
- GAAAAAAAATTACA => GA8T2ACA (reverse is also easy to implement)
- If your text contains numbers, then you will need to use a (slightly) more sophisticated encoding

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

rle(ref)[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_fo2lishnes2,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darknes2,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_al2_going_direct_to_Heaven,_we_were_al2_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_go2d_or_for_evil,_in_the_superlative_degre2_of_comparison_only.$
```

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooooiiihkiiiiiio__iei
tsppioi____________ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_______________oyoui__a_iiids__aiiaee_____________________tlar
```

Run Length Encoding

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooiiihkiiiiiio__iei
tsppioi____________ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_______________oyoui__a_iiids__aiiaee_____________________tlar
```

rle(bwt)[464]:

```
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

Run Length Encoding

bwt[614]:

```
.dlmssftysesdtrsns_y__$_yfofeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,
,,nfrsdnnhereghettedndeteegeenstee,ssssst,esssnssffteedtttttttttr,,
,,eeefehh__p__fpDwwwwwwwwwweehl_ew_____eoo_neeeoaaeoo____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo
oooooo____eennnnnnaai___ecc__tttttttttttttttttts_tsgltsLlvtt___hhoor
e_wrraddwlors_________r__lteirillre_ouaanooiioeoooiiihkiiiiiio__iei
tsppioi____________ggnodsc_sss_gfhf_fffhwh_nsmo__uee_siooaeeeeoo_ii
cgppeeaoaeooeesseuutetaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciiIi
iiiiiisn_______________oyoui__a_iiids__aiiaee_____________________tlar
```

rle(bwt)[464]:

```
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

rle(bwt)[464]:

```
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

Run Length Encoding

ref[614]:

```
It_was_the_best_of_times,_it_was_the_worst_of_times,_it_was_the_age_
of_wisdom,_it_was_the_age_of_foolishness,_it_was_the_epoch_of_belief
,_it_was_the_epoch_of_incredulity,_it_was_the_season_of_Light,_it_wa
s_the_season_of_Darkness,_it_was_the_spring_of_hope,_it_was_the_wint
er_of_despair,_we_had_everything_before_us,_we_had_nothing_before_us
,_we_were_all_going_direct_to_Heaven,_we_were_all_going_direct_the_o
ther_way_-_in_short,_the_period_was_so_far_like_the_present_period,_
that_some_of_its_noisiest_authorities_insisted_on_its_being_received
,_for_good_or_for_evil,_in_the_superlative_degree_of_comparison_only.$
```

rle(bwt)[464]:

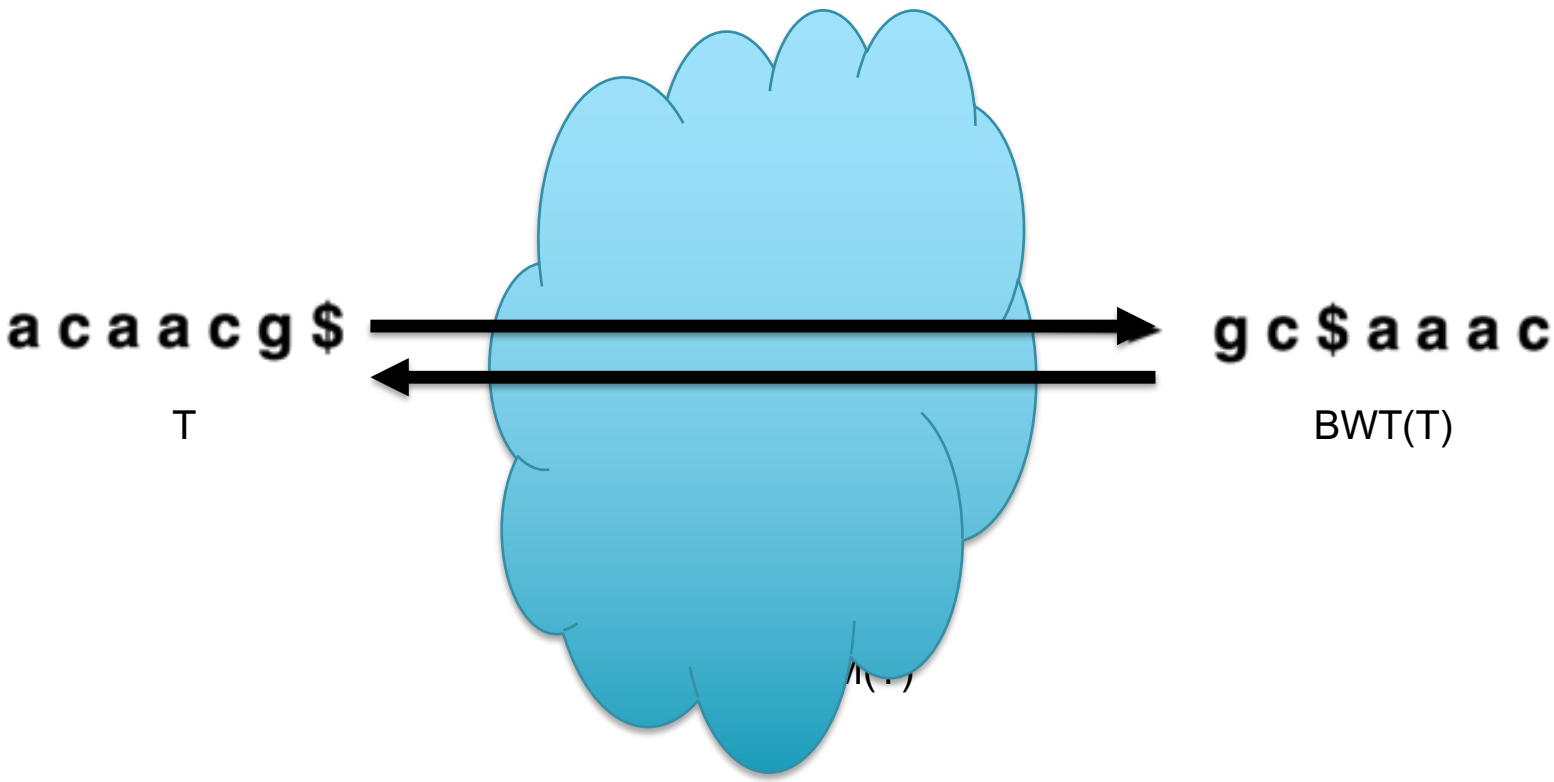
```
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a
2ie_e3rei
```

Saved 614-464 = 150 bytes (24%) with zero loss of information!

Common to save 50% to 90% on real world files with bzip2

Burrows-Wheeler Transform

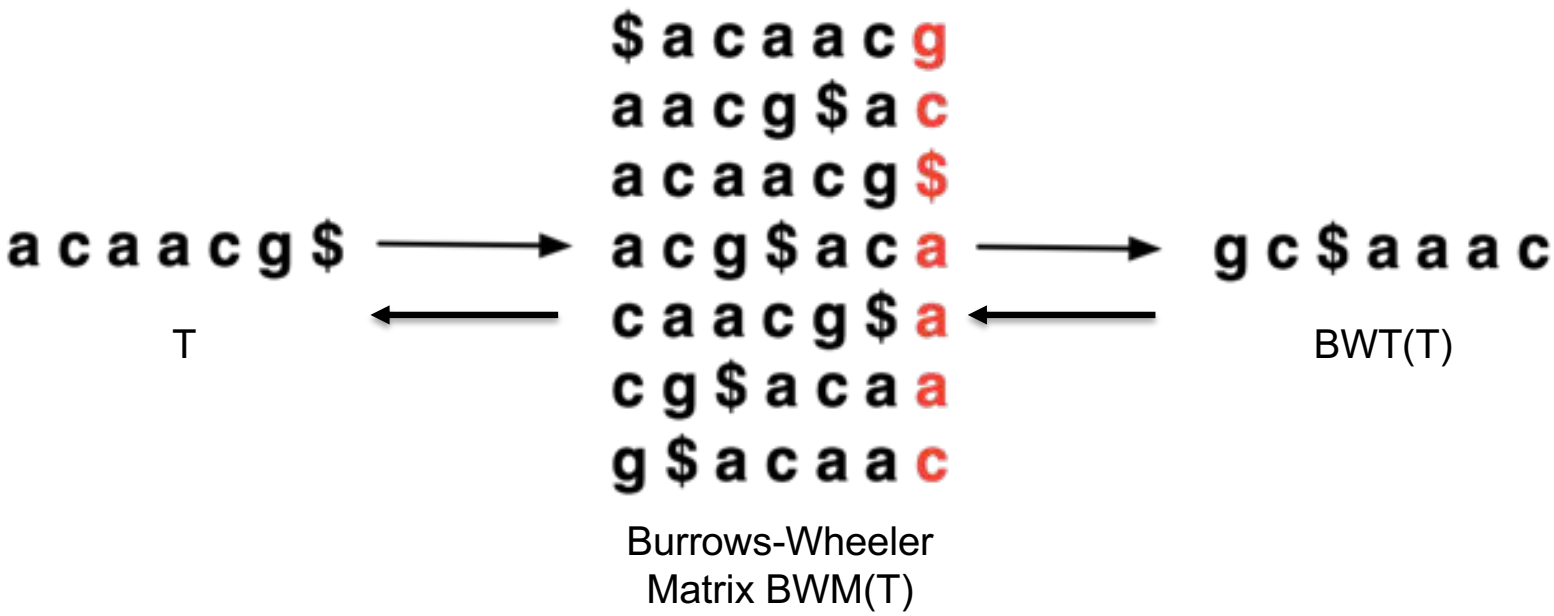
- Reversible permutation of the characters in a text



A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

Burrows-Wheeler Transform

- Reversible permutation of the characters in a text

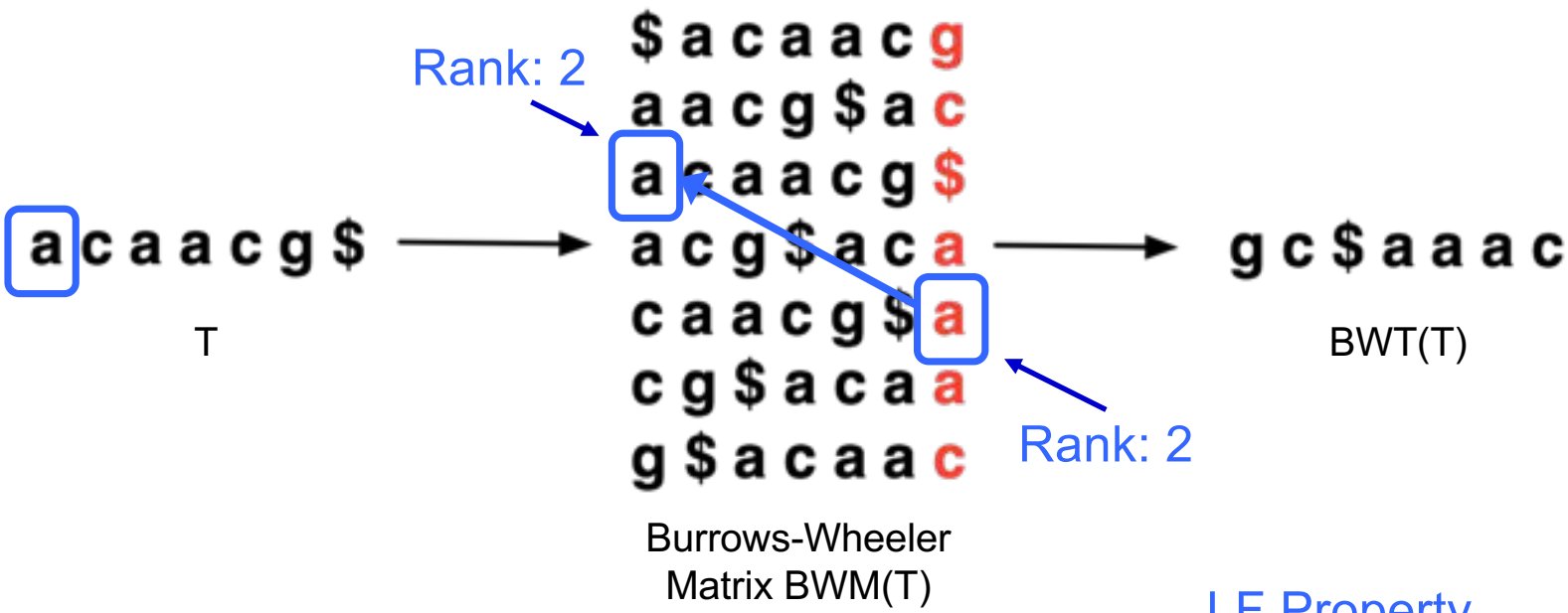


- BWT(T) is the index for T

A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



LF Property
implicitly encodes
Suffix Array

- BWT(T) is the index for T

A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

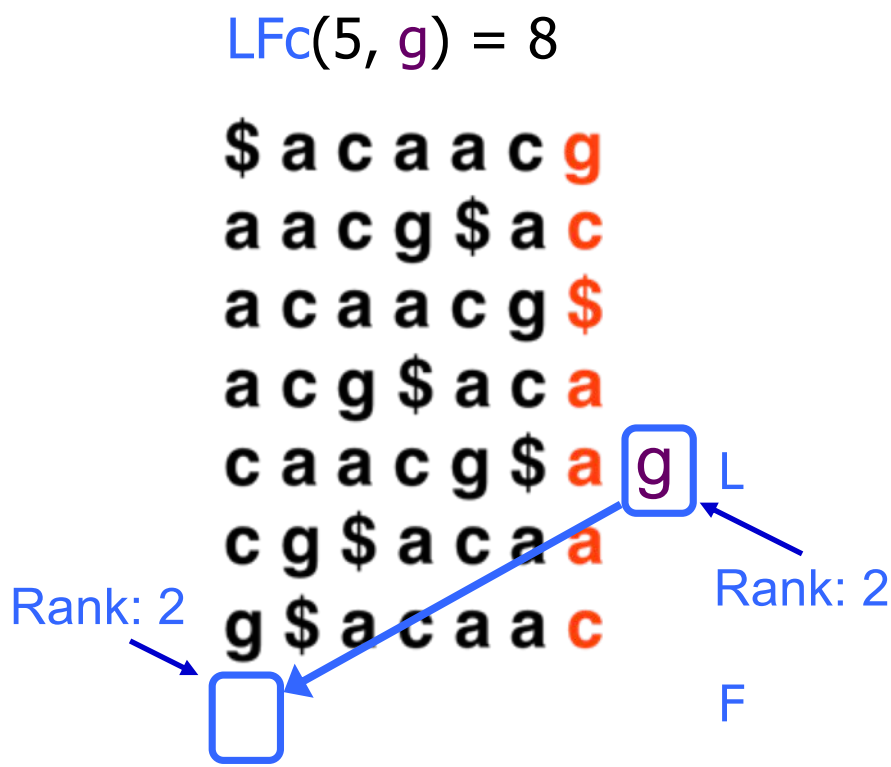
Burrows-Wheeler Transform

- Recreating T from BWT(T)
  - Start in the first row and apply **LF** repeatedly, accumulating predecessors along the way

[Decode this BWT string: ACTGA$TTA]

BWT Exact Matching

- **LFc**(r, c) does the same thing as **LF**(r) but it ignores r's actual final character and "pretends" it's c:



BWT Exact Matching

- Start with a range, (**top**, **bot**) encompassing all rows and repeatedly apply **LFc**:

  top = **LFc**(**top**, **qc**); **bot** = **LFc**(**bot**, **qc**)

  qc = the next character to the left in the query

Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

[Search for TTA this BWT string: ACTGA$TTA]

Algorithm Overview

1. Split read into segments

Read

CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA

Read (reverse complement)

TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG

Policy: extract 16 nt seed every 10 nt

Seeds

+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG

2. Lookup each segment and prioritize

Seeds

+, 0: CCAGTAGCTCTCAGCC
+, 10: TCAGCCTTATTTTACC
+, 20: TTTACCCAGGCCTGTA
-, 0: TACAGGCCTGGGTAAA
-, 10: GGTAAAATAAGGCTGA
-, 20: GGCTGAGAGCTACTGG

→ Ungapped alignment with FM Index

→ Seed alignments (as B ranges)

{ [211, 212], [212, 214] }
{ [653, 654], [651, 653] }
{ [684, 685] }
{ }
{ }
{ [624, 625] }

3. Evaluate end-to-end match

Extension candidates

SA:684, chr12:1955
SA:624, chr2:462
SA:211: chr4:762
SA:213: chr12:1935
SA:652: chr12:1945

→ SIMD dynamic programming aligner

→ SAM alignments

```
r1   0    chr12    1936     0
     36M  *    0    0
     CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
     IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
     AS:i:0    XS:i:-2   XN:i:0
     XM:i:0    XO:i:0    XG:i:0
     NM:i:0    MD:Z:36   YT:Z:UU
     YM:i:0
...
```

(Langmead & Salzberg, 2012)

Next Steps

1. Reflect on the magic and power of Suffix Arrays and the BWT!

1. Assignment 8 due Friday November 16 @ 10pm