Assembling Genomes

BCH394P/364C Systems Biology / Bioinformatics

Edward Marcotte, Univ of Texas at Austin

1


www.yourgenome.org/facts/timeline-the-human-genome-project


Nature 409, 860-921(2001)

2

3

The NovaSeq in the UT GSAF core generates >1.4 terabases of sequence in a 1-day run

→ **Many millions of 75-150 bp reads**

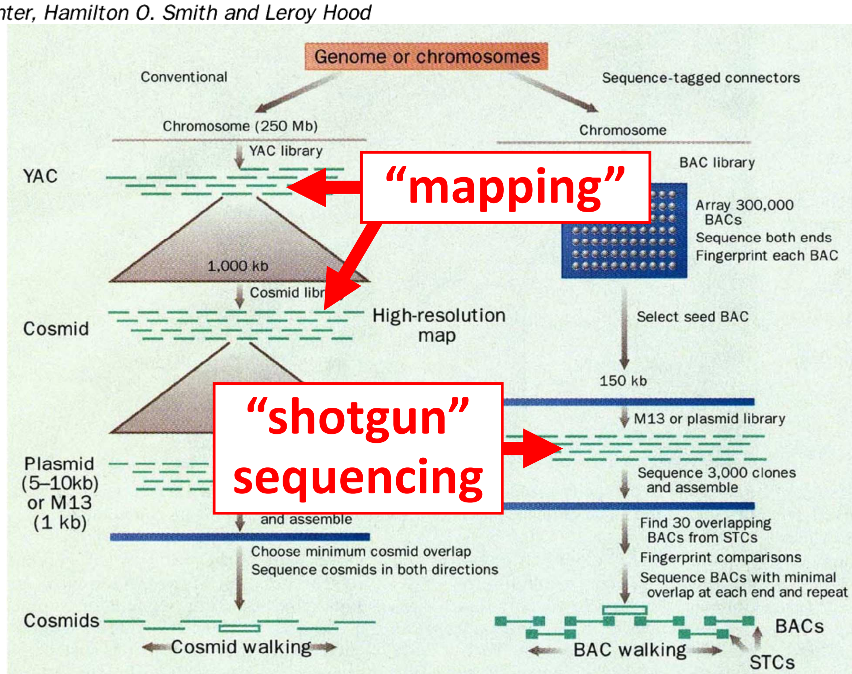
4

→ **Thousands of 1,000 to 1,000,000 (ish) bp reads**

5

A new strategy for genome sequencing

J. Craig Venter, Hamilton O. Smith and Leroy Hood



NATURE · VOL 381 · 30 MAY 1996

6

(Translating the cloning jargon)

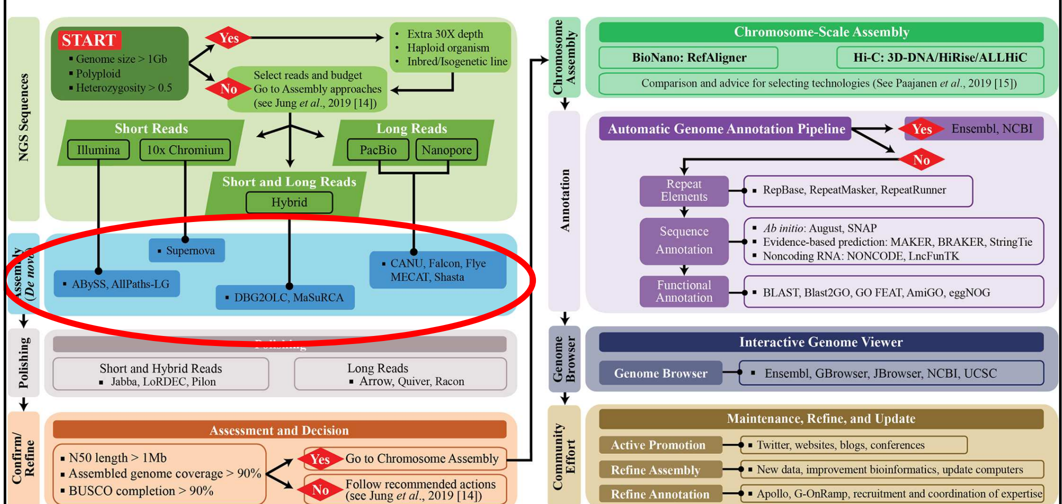
CLONE LIBRARIES USED FOR GENOME MAPPING AND SEQUENCING

Vector	Human-DNA insert size range	Number of clones required to cover the human genome
Yeast artificial chromosome (YAC)	100–2,000 kb	3,000 (1,000 kb)
Bacterial artificial chromosome (BAC)	80–350 kb	20,000 (150 kb)
Cosmid	30–45 kb	75,000 (40 kb)
Plasmid	3–10 kb	600,000 (5 kb)
M13 phage	1 kb	3,000,000 (1 kb)

NATURE · VOL 381 · 30 MAY 1996

7

Contemporary genome assembly is fairly complex, but at its core are assembly algorithms that grew from the shotgun concept



Twelve quick steps for genome assembly and annotation in the classroom
PLoS Comp Biology (2020), doi:10.1371/journal.pcbi.1008325

8

Beverly Micro “Pure White Hell” Jigsaw Puzzle (10,000,000,000 Piece)

9

Thinking about the basic shotgun concept

- Start with a very large set of random sequencing reads
- How might we match up the overlapping sequences?
- How can we assemble the overlapping reads together in order to derive the genome?

10

Thinking about the basic shotgun concept

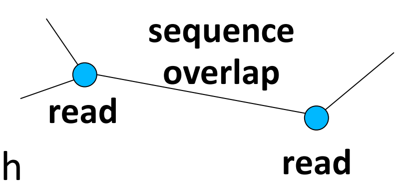
- At a high level, the first genomes were sequenced by comparing pairs of reads to find overlapping reads
- Then, building a graph (*i.e.*, a network) to represent those relationships
- The genome sequence is a "walk" across that graph

11

The "Overlap-Layout-Consensus" method

Overlap: Compare all pairs of reads
(allow some low level of mismatches)

Layout: Construct a graph describing the overlaps

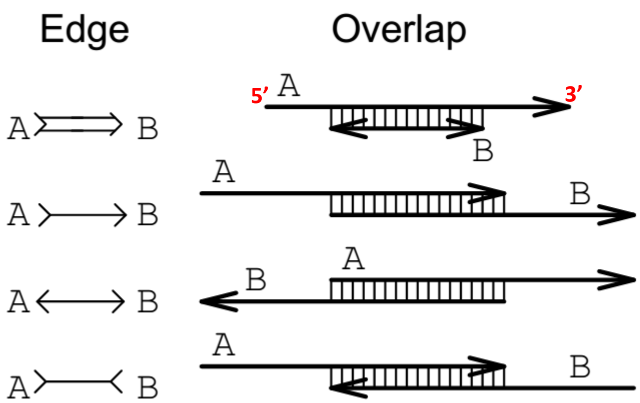


Simplify the graph
Find the simplest path through the graph

Consensus: Reconcile errors among reads along that path to find the consensus sequence

12

Building an overlap graph

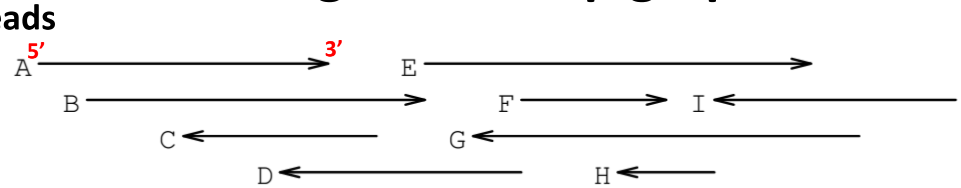


EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290

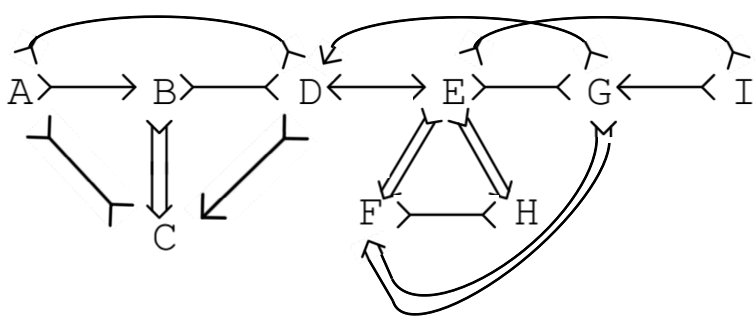
13

Building an overlap graph

Reads



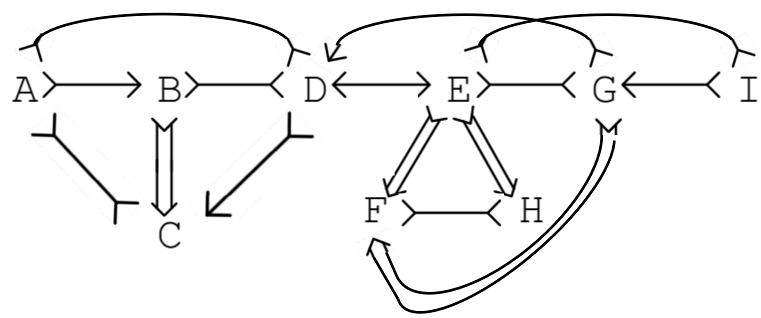
Overlap graph



EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

14

Simplifying an overlap graph

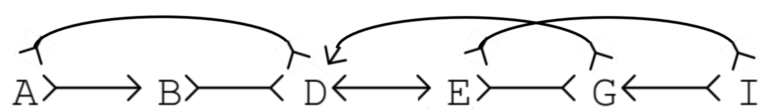


1. Remove all contained nodes & edges going to them

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

15

Simplifying an overlap graph

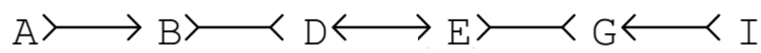


2. Transitive edge removal:
   Given A – B – D   and   A – D , remove A – D

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

16

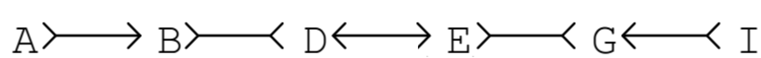
Simplifying an overlap graph

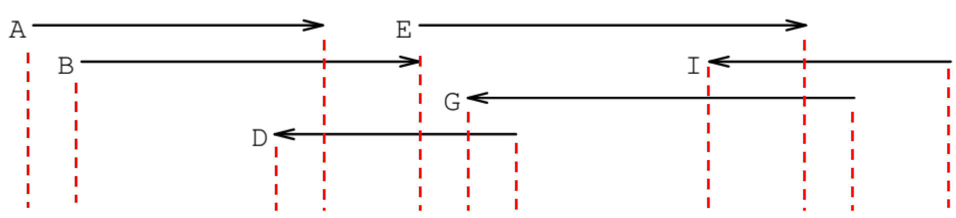


3. If un-branched, calculate consensus sequence
   If branched, assemble un-branched bits and then decide
   how they fit together

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

Simplifying an overlap graph



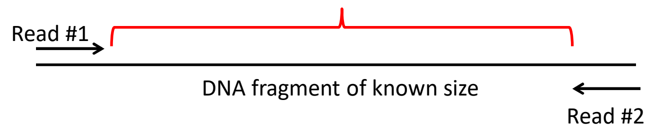


“contig” (assembled contiguous sequence)

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

This basic strategy was used for most of the early genomes. Also useful: “mate pairs”

2 reads separated by a known distance



Contigs can be ordered using these paired reads

to produce “scaffolds”

19

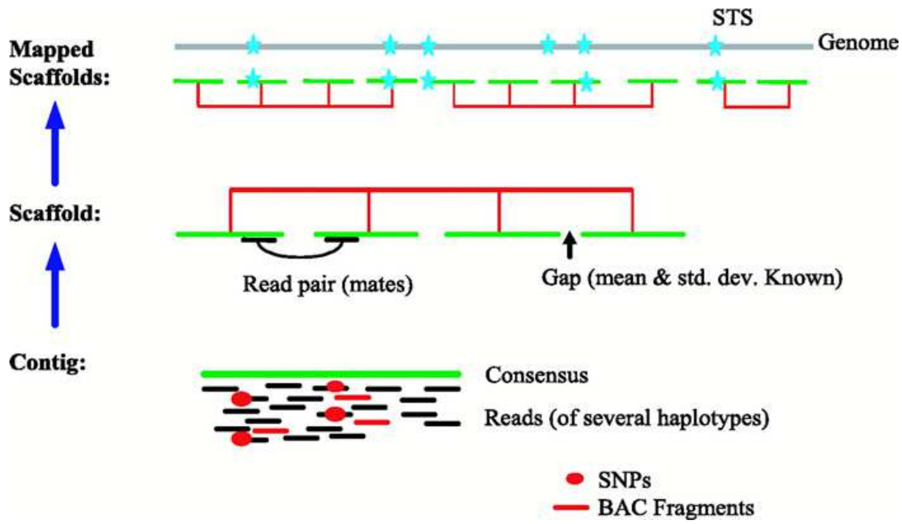
GigAssembler (used to assemble the public human genome project sequence)


Jim Kent David Haussler

Let’s take a little walk through history to see what they did…

20

Whole genome Assembly: big picture



http://www.nature.com/scitable/content/anatomy-of-whole-genome-assembly-20429

21

GigAssembler – Preprocessing

1. Decontaminating & Repeat Masking.
2. Aligning of mRNAs, ESTs, BAC ends & paired reads against initial sequence contigs.
   - psLayout → BLAT
3. Creating an input directory (folder) structure.

```
chr1/
chr1/contig1.e
chr1/contig1.a
chr1/contig1.c
chr1/contig1.b
chr1/contig1.d
chr3/
chr2/
chr2/contig2.d
chr2/contig2.b
chr2/contig2.a
chr2/contig2.c
```

22

RepBase + RepeatMasker

```
taejoon@fourierseq:~/RepBase/RepBase15.05.fasta$ ls -a
.             dcotrep.ref  mamsub.ref   rodsub.ref
..            diarep.ref   mcotrep.ref  simple.ref
angrep.ref    drorep.ref   mousub.ref   spurep.ref
appendix      fngrep.ref   nemrep.ref   synrep.ref
athrep.ref    fugrep.ref   oryrep.ref   tmplanrep.ref
bctrep.ref    grasrep.ref  plnrep.ref   tmpnemrep.ref
cbrrep.ref    humrep.ref   prirep.ref   tmpxenrep.ref
celrep.ref    humsub.ref   prisub.ref   version
chlrep.ref    invrep.ref   pseudo.ref   vrtrep.ref
cinrep.ref    invsub.ref   ratsub.ref   zebrep.ref
cinunc.ref    mamrep.ref   rodrep.ref
```

```
>MER51D ERV1  Homo sapiens
tgaggcaggagaaaatagcagagggaattggaagttggataaagggagaatgagtaaaagcangagagca
gaagcaaggtaaagaggcgggtgagcaagaagcaagataagaagcagaagttgagcagccaaaacaaaag
taagatnanaaagaagtgagtaaggagcccacatggctggctagatccagaccaaaccagtaaggggcag
ctcctcagagatgggcatgtacattagagagaaaaagtatccttaaaatgaccccgtatgataatcagct
cattaaagctcatgcatatggactgcatatcatgcatgtacttaaaattatgggatggaggtgacgcgca
agawgtcacaagcacacaggggccatagkattaagtaactaagcaacccacctatcaatcaaaaggcaga
tgctggctagagattaggcagccttgggaagagaagaaaaaaaaaacacataaaaagacccaaagtacac
caaactgacgctgatctcatttcgcagaggtcagcccactctcccctctctgagagtgtaatactgtgct
taataaacttttgctgctttgctatctgtgtgtgtcttgtccaattctttgtttgggacaccaagagcct
ggaactgcacrgcaccakctggtaaca
>MIRb SINE2/tRNA  Mammalia
cagaggggcagcgtggtgcagtggaaagagcacgggctttggagtcaggcagacctgggttcgaatcctg
gctctgccacttactagctgtgtgaccttgggcaagtcacttaacctctctgagcctcagtttcctcatc
tgtaaaatggggataataatacctacctcgcagggttgttgtgaggattaaatgagataatgcatgtaaa
gcgcttagcacagtgcctggcacacagtaagcgctcaataaatggtagctctattatt
>LTR45  ERV1  Homo sapiens
tgtaaccgcgggaccagcccaaactgggcctactctgttgataacaaaatgtcaagttaccttgtaggta
taacagagcccaaaactgcaagtcatgtagcccgggcatgtgcaatagaaaaagctttgacctctaacaa
cacccagaaccaatgattcctcccctcggaaccaagaagaccgggacatgaccggaacctgaatgccgga
actctttcagaagcaaaggggtccgttggcccggaagatctggggctaaaatctgcctcaacatacctta
ccgtaaatggtcaaatttgaagccctccaatcagaccctgccaagccaacattcctaaatcctttccctt
gccctctgatcccttaaaacttgccccagaccccaaatcggggagacagatttgagcccacctcctgtct
ccttgctggccggttttgcaataaagcctttcttttctcaaaagctggtgccatagttattggcttctgt
gtgcatcaggcagcaagcccatttgctcgataaca
>MER80B hAT Homo sapiens
cagggcttcttaaccagaggtccatggatgggcttcaggaggtctgtgaaccctctgaaattatatacaa
aaatgttgtgtatatgtgcatatatgtatttttctggggagagggttcatagctttcatcagattctcaa
aggggtctatgatctmaaaaaggttaagaagccctg
```

23

GigAssembler: Build merged sequence contigs (“rafts”)

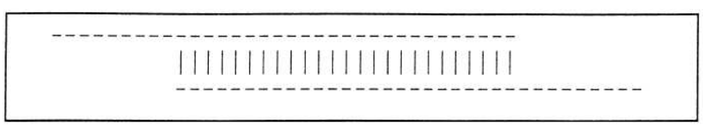
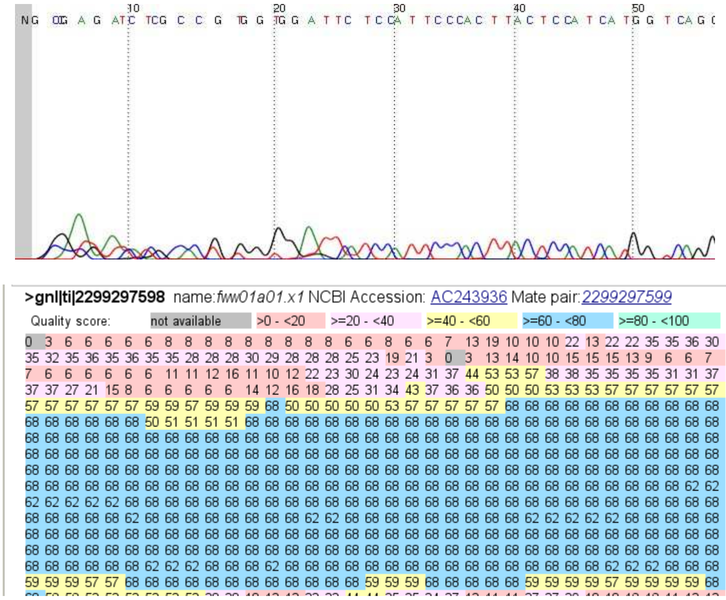


Figure 1 Two sequences overlapping end to end. The sequences are represented as dashes. The aligning regions are joined by vertical bars. End-to-end overlap is an extremely strong indication that two sequences should be joined into a contig.

24

Sequencing quality (Phred Score)



Sequencing quality (Phred Score)

$$Q = -10 \log_{10} P$$ ← *Base-calling Error Probability*

or

$$P = 10^{\frac{-Q}{10}}$$

Phred quality scores are logarithmically linked to error probabilities

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90 %
20	1 in 100	99 %
30	1 in 1000	99.9 %
40	1 in 10000	99.99 %
50	1 in 100000	99.999 %

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

We're going to skip the remaining details of GigAssembler (mainly of historical interest now) to get to the key strategy for assembling all of the various contigs and paired end reads into a genome

27

GigAssembler: Build a "raft-ordering" graph

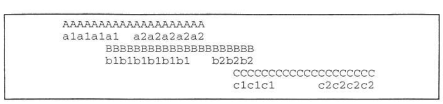


Figure 4 Three overlapping draft clones: A, B, and C. Each clone has two initial sequence contigs. Note that initial sequence contigs a1, b1, and a2 overlap as do b2 and c1.

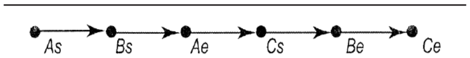


Figure 5 Ordering graph of clone starts and ends. This represents the same clones as in Fig. 4. (As) The start of clone A; (Ae) the end of clone A. Similarly Bs, Be, Cs, and Ce represent the starts and ends of clones B and C.

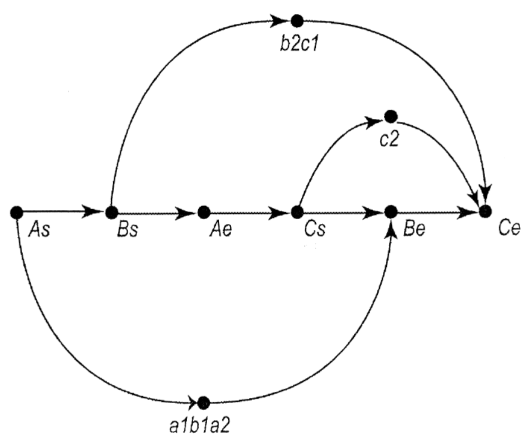


Figure 6 Ordering graph after adding in rafts. The initial sequence contigs shown in Fig. 4 are merged into rafts where they overlap. This forms three rafts: a1b1a2, b2c1, and c2. These rafts are constrained to lie between the relevant clone ends by the addition of additional ordering edges to the graph shown in Fig. 5.

28

GigAssembler: Build a "raft-ordering" graph

- Add information from mRNAs, ESTs, paired plasmid reads, BAC end pairs: building a "bridge"
  - Different weight to different data type: (mRNA ~ highest)
  - Conflicts with the graph as constructed so far are rejected.
- Build a sequence path through each raft.
- Fill the gap with N's.
  - 100: between rafts
  - 50,000: between bridged barges

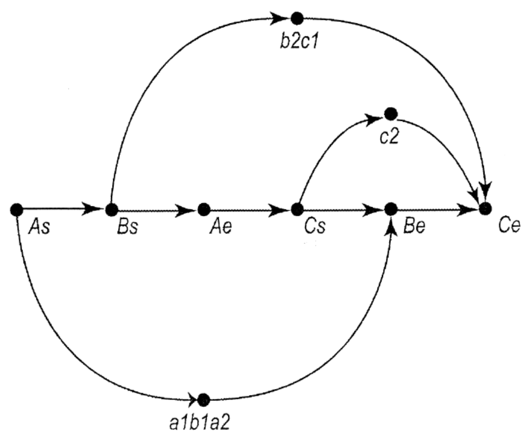


Figure 6 Ordering graph after adding in rafts. The initial sequence contigs shown in Fig. 4 are merged into rafts where they overlap. This forms three rafts: a1b1a2, b2c1, and c2. These rafts are constrained to lie between the relevant clone ends by the addition of additional ordering edges to the graph shown in Fig. 5.

29

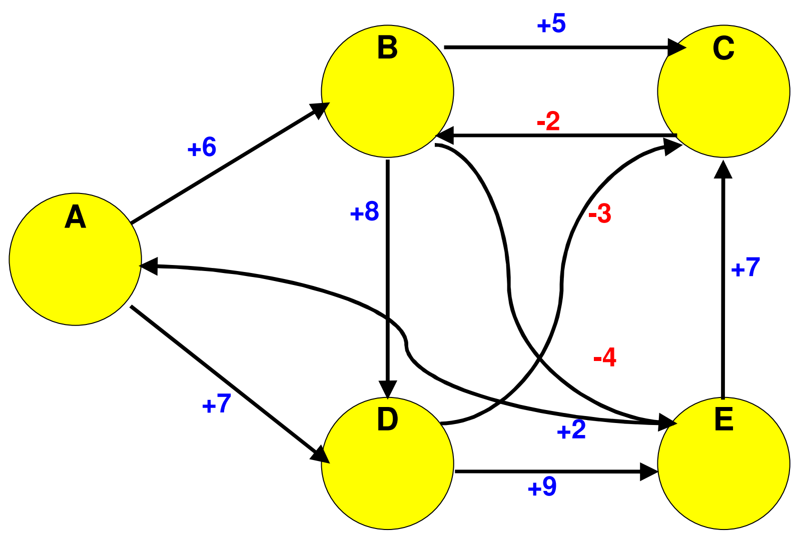
Finding the shortest path across the ordering graph using the Bellman-Ford algorithm

http://compprog.wordpress.com/2007/11/29/one-source-shortest-path-the-bellman-ford-algorithm/

30

Find the shortest path to all nodes.

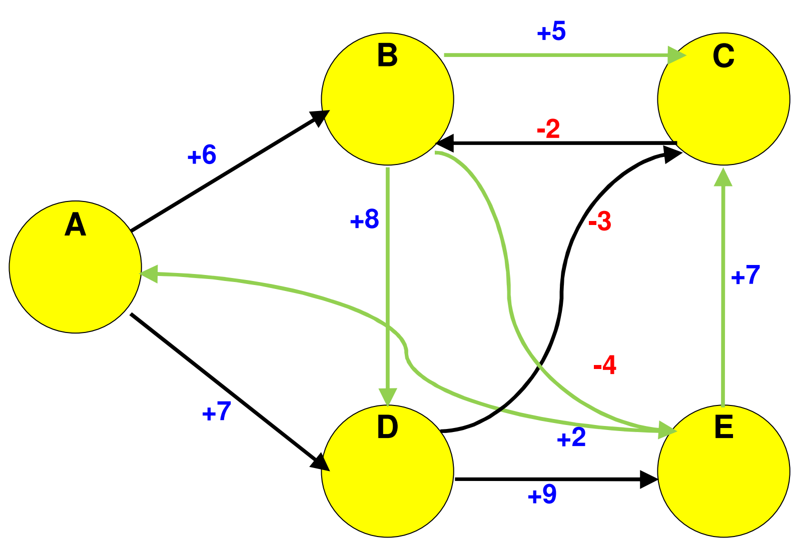
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



31

Find the shortest path to all nodes.

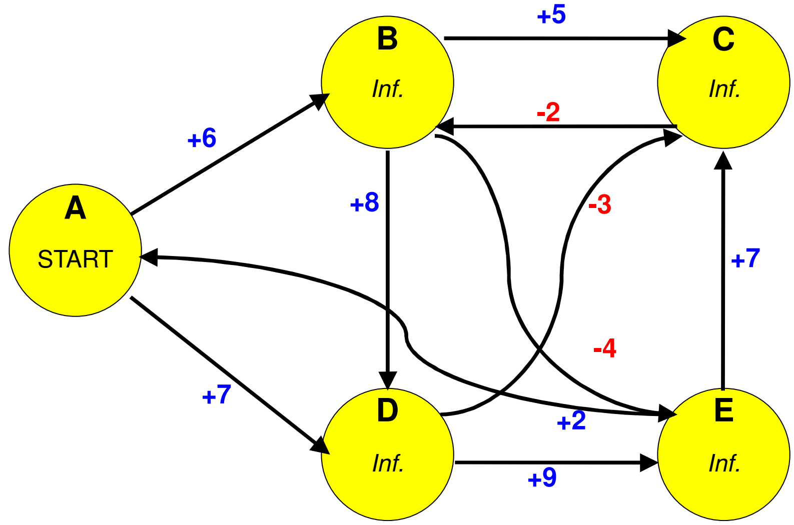
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



32

Find the shortest path to all nodes.

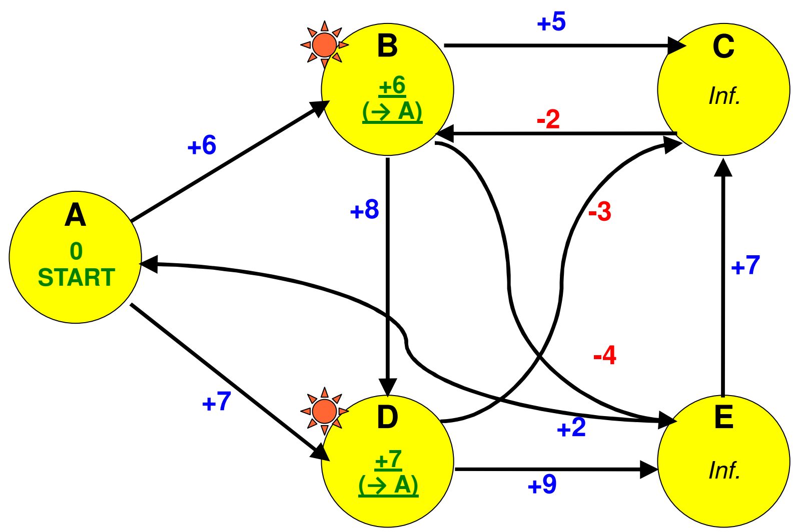
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



33

Find the shortest path to all nodes.

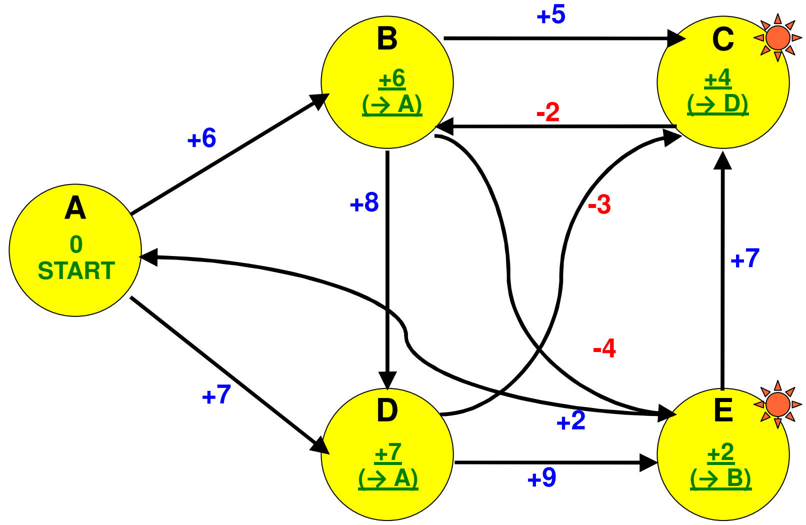
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



34

Find the shortest path to all nodes.

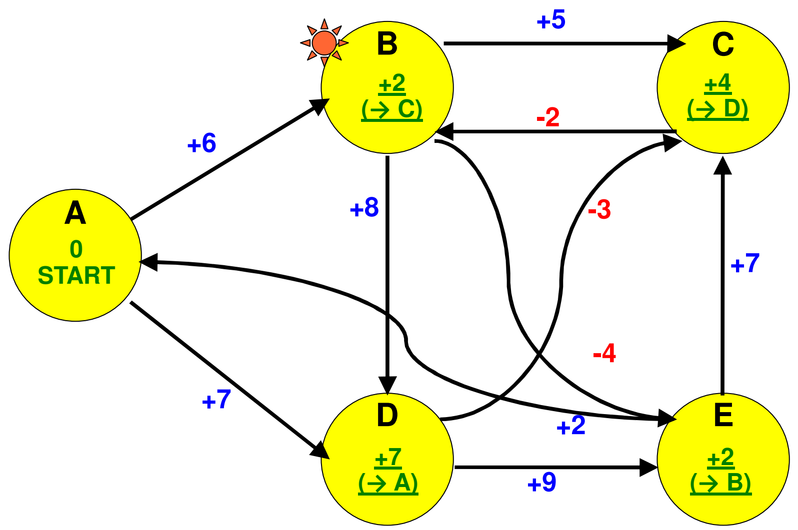
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



35

Find the shortest path to all nodes.

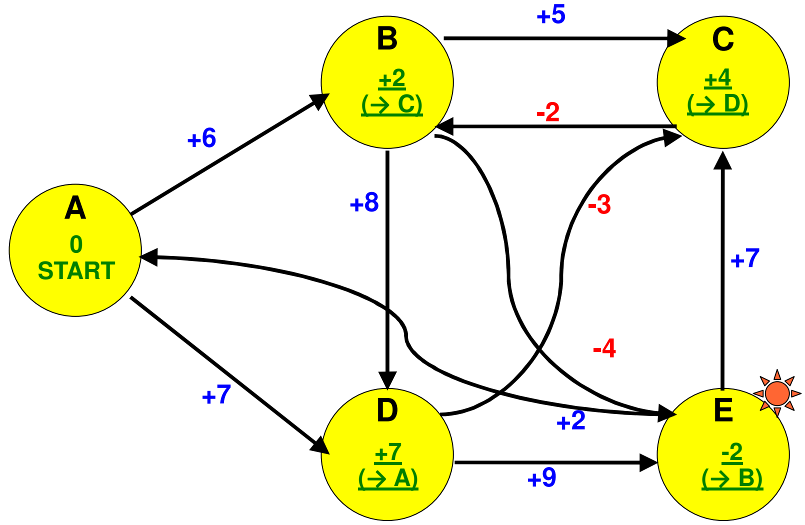
Take every edge and try to relax it (N – 1 times where N is the count of nodes)



36

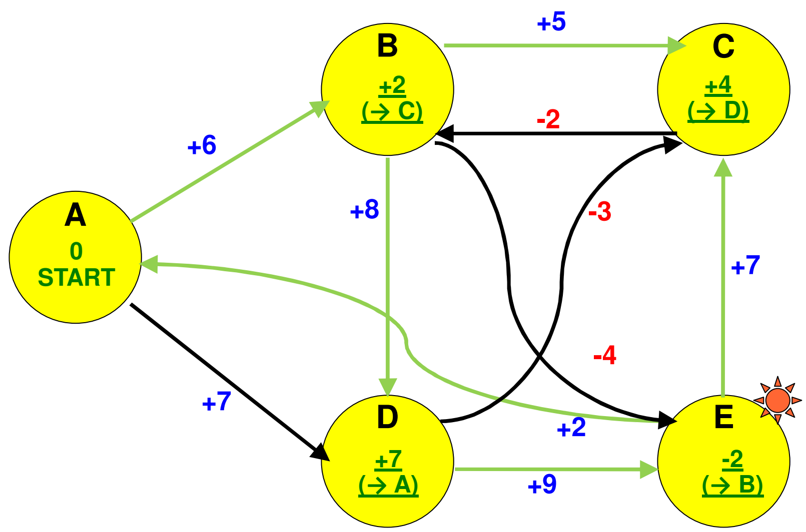
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



37

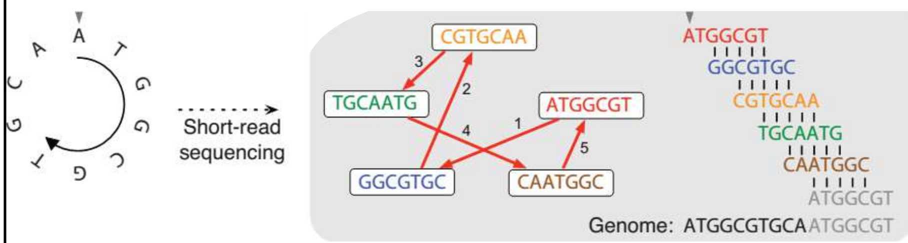
Answer: A-D-C-B-E



38

Modern assemblers now work a bit differently, using so-called DeBruijn graphs:

Here's what we saw before:



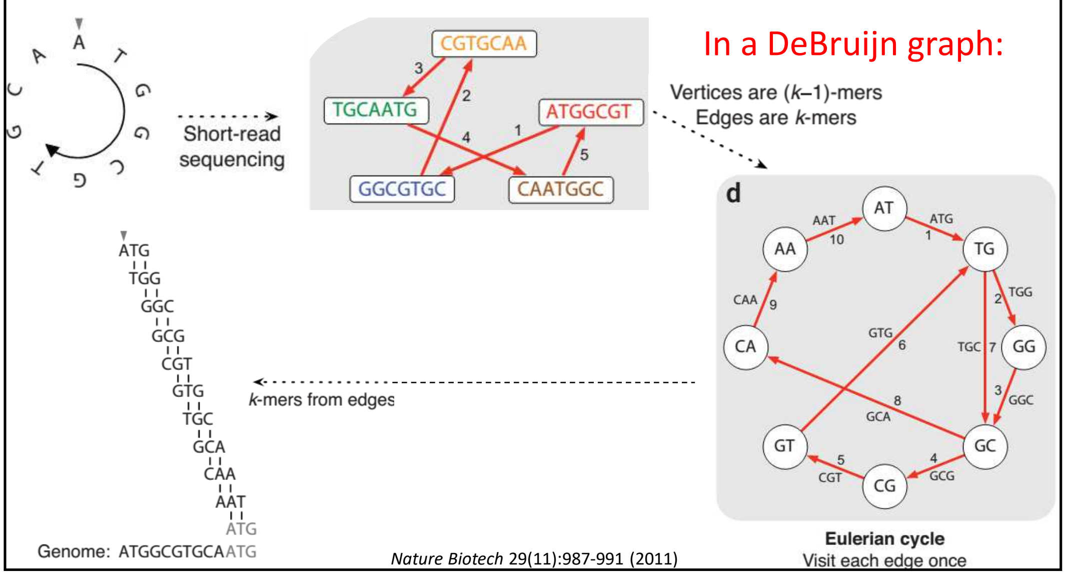
In Overlap-Layout-Consensus:
Nodes are reads
Edges are overlaps

Nature Biotech 29(11):987-991 (2011)

39

Modern assemblers now work a bit differently, using so-called DeBruijn graphs:

In a DeBruijn graph:



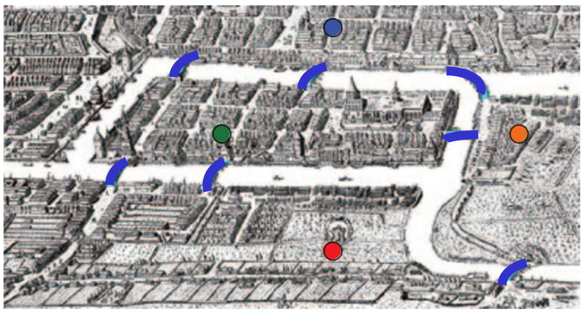
Nature Biotech 29(11):987-991 (2011)

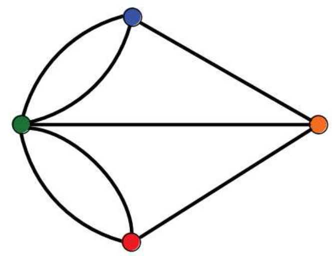
40

Why Eulerian?

From Leonhard Euler's solution in 1735 to the
'Bridges of Königsberg' problem:

Königsberg (now Kaliningrad, Russia) had 7 bridges connecting 4 parts of the city. **Could you visit each part of the city, walking across each bridge only once, & finish back where you started?**



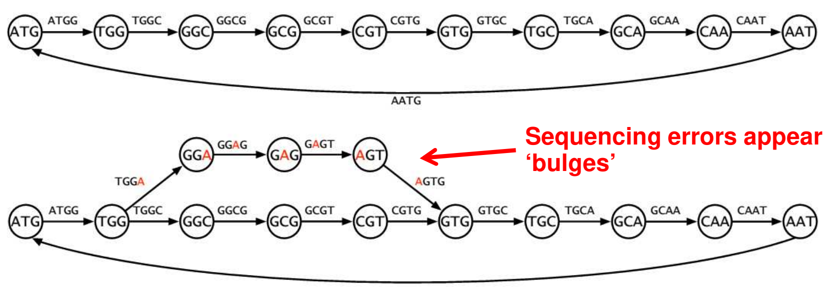


(Visiting every edge once = an *Eulerian* path)

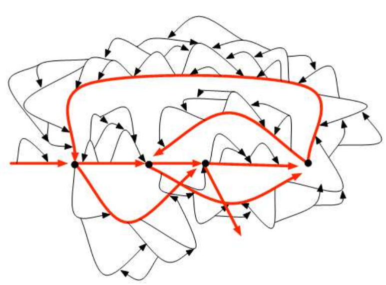
Euler conceptualized it as a graph:
Nodes = parts of city
Edges = bridges

Nature Biotech 29(11):987-991 (2011)

DeBruijn graph assemblers tend to have nice properties, e.g. correcting sequencing errors & handling repeats better



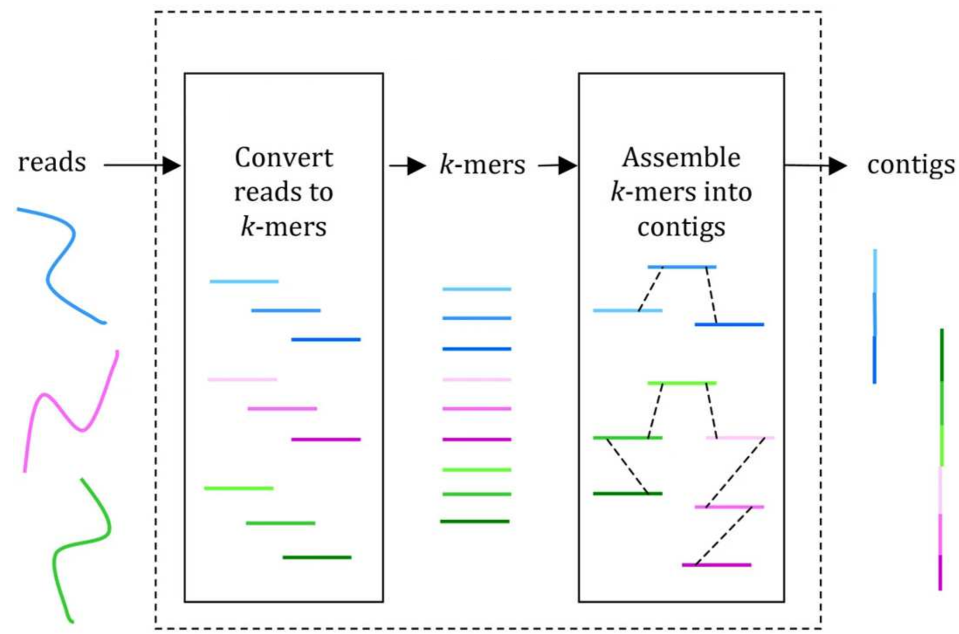
Sequencing errors appear as 'bulges'



Removing the 'bulges' corrects the errors (e.g. leaves the red path)

Nature Biotech 29(11):987-991 (2011)

e.g. Velvet, an example algorithm using DeBruijn graphs



Beginner's guide to comparative bacterial genome analysis using next-generation sequence data
Microb Informatics Exp (2013) doi:10.1186/2042-5783-3-2

43

Once a reference genome is assembled, new sequencing data can 'simply' be mapped to the reference.

44

Mapping reads to assembled genomes

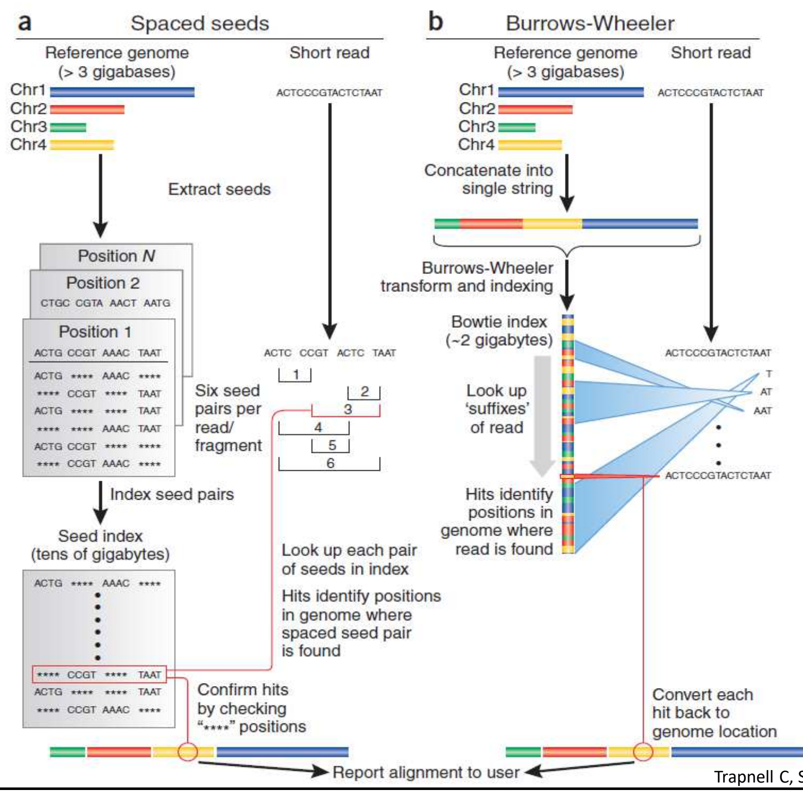
Table 1 A selection of short-read analysis software

Program	Website	Open source?	Handles ABI color space?	Maximum read length
Bowtie	http://bowtie.cbcb.umd.edu	Yes	No	None
BWA	http://maq.sourceforge.net/bwa-man.shtml	Yes	Yes	None
Maq	http://maq.sourceforge.net	Yes	Yes	127
Mosaik	http://bioinformatics.bc.edu/marthlab/Mosaik	No	Yes	None
Novoalign	http://www.novocraft.com	No	No	None
SOAP2	http://soap.genomics.org.cn	No	No	60
ZOOM	http://www.bioinfor.com	No	Yes	240

The list is a little longer now! e.g. see https://en.wikipedia.org/wiki/List_of_sequence_alignment_software#Short-Read_Sequence_Alignment

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

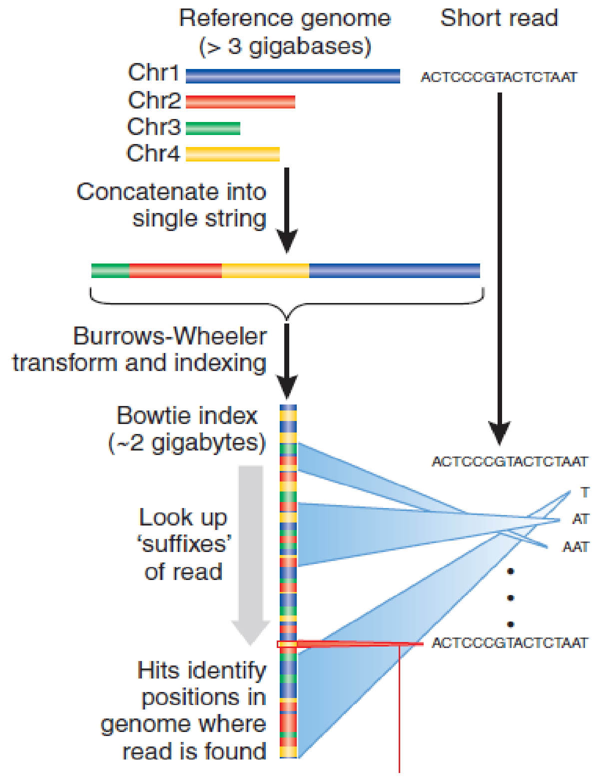
45



Mapping strategies

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

46



Burroughs Wheeler indexing

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

47

Burroughs-Wheeler transform indexing

BWT is often used for file compression (like bzip2),
here used to make a fast 'lookup' index in a genome

BWT = 'reversible block-sorting'

Input SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

↓ **Forward BWT** — This sequence is more compressible

Output TEXYDST.E.IXIXIXXSSMPPS.B..E.S.EUSFXDIIOIIIT

↓ **Reverse BWT**

Recovered input SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

48

Burroughs-Wheeler transform indexing

Input
^BANANA|

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

49

Burroughs-Wheeler transform indexing

All Rotations
^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

50

Burroughs-Wheeler transform indexing

Sorting All Rows in Alphabetical Order

```
ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA
```

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

51

Burroughs-Wheeler transform indexing

Taking Last Column

```
ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA
```

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

52

Burroughs-Wheeler transform indexing

Output Last Column
BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

53

Burroughs-Wheeler transform indexing

Transformation				
Input	All Rotations	Sorting All Rows in Alphabetical Order	Taking Last Column	Output Last Column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **A**NA|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **N**A|^BANA **^**BANANA| **|**^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|**^** NANA|^B**A** NA|^BAN**A** ^BANANA**|** |^BANAN**A**	BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

54

BWT is remarkable because it is ***reversible.***

Any ideas as how you might reverse it?

55

Burroughs-Wheeler transform indexing

Input
BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

56

Burroughs-Wheeler transform indexing

<table>
<tr><th>Add 1</th><th>Sort 1</th><th>Add 2</th><th>Sort 2</th></tr>
<tr><td>B</td><td>A</td><td>BA</td><td>AN</td></tr>
<tr><td>N</td><td>A</td><td>NA</td><td>AN</td></tr>
<tr><td>N</td><td>A</td><td>NA</td><td>A|</td></tr>
<tr><td>^</td><td>B</td><td>^B</td><td>BA</td></tr>
<tr><td>A</td><td>N</td><td>AN</td><td>NA</td></tr>
<tr><td>A</td><td>N</td><td>AN</td><td>NA</td></tr>
<tr><td>|</td><td>^</td><td>|^</td><td>^B</td></tr>
<tr><td>A</td><td>|</td><td>A|</td><td>|^</td></tr>
<tr><td>Write the sequence as the last column</td><td>Sort it...</td><td>Add the columns...</td><td>Sort those...</td></tr>
</table>

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

57

Burroughs-Wheeler transform indexing

<table>
<tr><th>Add 3</th><th>Sort 3</th><th>Add 4</th><th>Sort 4</th></tr>
<tr><td>BAN</td><td>ANA</td><td>BANA</td><td>ANAN</td></tr>
<tr><td>NAN</td><td>ANA</td><td>NANA</td><td>ANA|</td></tr>
<tr><td>NA|</td><td>A|^</td><td>NA|^</td><td>A|^B</td></tr>
<tr><td>^BA</td><td>BAN</td><td>^BAN</td><td>BANA</td></tr>
<tr><td>ANA</td><td>NAN</td><td>ANAN</td><td>NANA</td></tr>
<tr><td>ANA</td><td>NA|</td><td>ANA|</td><td>NA|^</td></tr>
<tr><td>|^B</td><td>^BA</td><td>|^BA</td><td>^BAN</td></tr>
<tr><td>A|^</td><td>|^B</td><td>A|^B</td><td>|^BA</td></tr>
<tr><td>Add the columns...</td><td>Sort those...</td><td>Add the columns...</td><td>Sort those...</td></tr>
</table>

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

58

Burroughs-Wheeler transform indexing

<table>
<tr><th>Add 5</th><th>Sort 5</th><th>Add 6</th><th>Sort 6</th></tr>
<tr><td>BANAN</td><td>ANANA</td><td>BANANA</td><td>ANANA|</td></tr>
<tr><td>NANA|</td><td>ANA|^</td><td>NANA|^</td><td>ANA|^B</td></tr>
<tr><td>NA|^B</td><td>A|^BA</td><td>NA|^BA</td><td>A|^BAN</td></tr>
<tr><td>^BANA</td><td>BANAN</td><td>^BANAN</td><td>BANANA</td></tr>
<tr><td>ANANA</td><td>NANA|</td><td>ANANA|</td><td>NANA|^</td></tr>
<tr><td>ANA|^</td><td>NA|^B</td><td>ANA|^B</td><td>NA|^BA</td></tr>
<tr><td>|^BAN</td><td>^BANA</td><td>|^BANA</td><td>^BANAN</td></tr>
<tr><td>A|^BA</td><td>|^BAN</td><td>A|^BAN</td><td>|^BANA</td></tr>
<tr><td>Add the columns...</td><td>Sort those...</td><td>Add the columns...</td><td>Sort those...</td></tr>
</table>

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

59

Burroughs-Wheeler transform indexing

<table>
<tr><th>Add 7</th><th>Sort 7</th><th>Add 8</th></tr>
<tr><td>BANANA|</td><td>ANANA|^</td><td>BANANA|^</td></tr>
<tr><td>NANA|^B</td><td>ANA|^BA</td><td>NANA|^BA</td></tr>
<tr><td>NA|^BAN</td><td>A|^BANA</td><td>NA|^BANA</td></tr>
<tr><td>^BANANA</td><td>BANANA|</td><td>^BANANA|</td></tr>
<tr><td>ANANA|^</td><td>NANA|^B</td><td>ANANA|^B</td></tr>
<tr><td>ANA|^BA</td><td>NA|^BAN</td><td>ANA|^BAN</td></tr>
<tr><td>|^BANAN</td><td>^BANANA</td><td>|^BANANA</td></tr>
<tr><td>A|^BANA</td><td>|^BANAN</td><td>A|^BANAN</td></tr>
<tr><td>Add the columns...</td><td>Sort those...</td><td>Add the columns...</td></tr>
</table>

The row with the "end of file" character at the end is the original text

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

60

Burroughs-Wheeler transform indexing

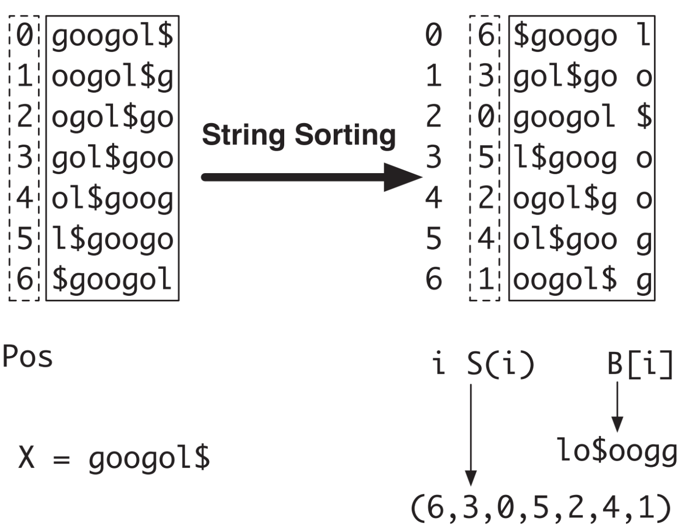
Output
^BANANA|

The row with the "end of file" character at the end is the original text

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

61

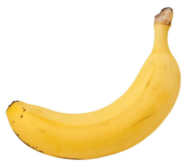
The Burroughs-Wheeler transform leads naturally to a suffix array



Li & Durbin, doi:10.1093bioinformatics/btp324/

62

The Burroughs-Wheeler transform leads naturally to a suffix array



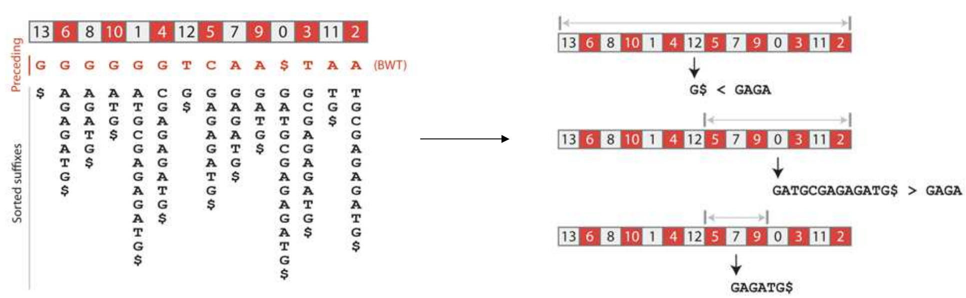
Suffix Array							BWT
1	a	n	a	n	a	$	b
3	a	n	a	$	b	a	n
5	a	$	b	a	n	a	n
0	b	a	n	a	n	a	$
2	n	a	n	a	$	b	a
4	n	a	$	b	a	n	a
6	$	b	a	n	a	n	a

http://blog.avadis-ngs.com/2012/04/elegant-exact-string-match-using-bwt-2/ (& wikipedia)

"If string W is a substring of X, the position of each occurrence of W in X will occur in an interval in the suffix array. This is because all the suffixes that have W as prefix are sorted together."

Li & Durbin, doi:10.1093bioinformatics/btp324/

e.g. applying BWT to construct the suffix array of **GATGCGAGAGATG**



The search can be even more efficient by using compression & various other extensions

http://blog.thegrandlocus.com/2016/07/a-tutorial-on-burrows-wheeler-indexing-methods

Why is this efficient?

Searching a suffix array in this way cuts the search space in half at each step, so…

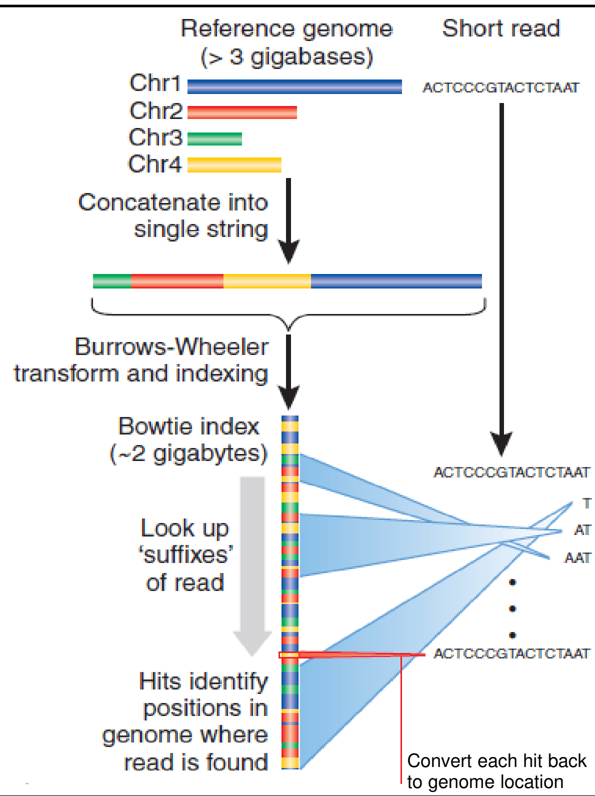
A suffix array of the human genome (3.2 billion bases) takes at most

$$\log_2(3.2 \text{ billion}) + 1 = 32 \text{ steps}$$

to determine if a query sequence is present or not

There are few more steps to find all the occurrences, build an efficient real-world implementation, use compression to reduce memory and storage space, etc., but this still illustrates the massive savings in time and memory from constructing an index

65



Burroughs Wheeler indexing

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

66