Bioinformatik für Biologen

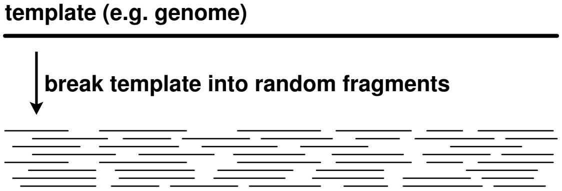
Heiko A. Schmidt

Center for Integrative Bioinformatics Vienna (CIBIV)
Max F. Perutz Laboratories (MFPL)
Vienna, Austria
`http://www.cibiv.at`

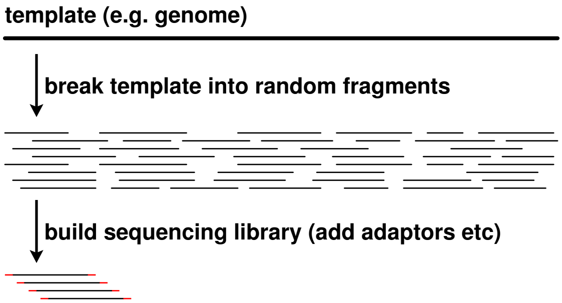
Genome Sequencing, Assembly and NGS Data

Shotgun sequencing

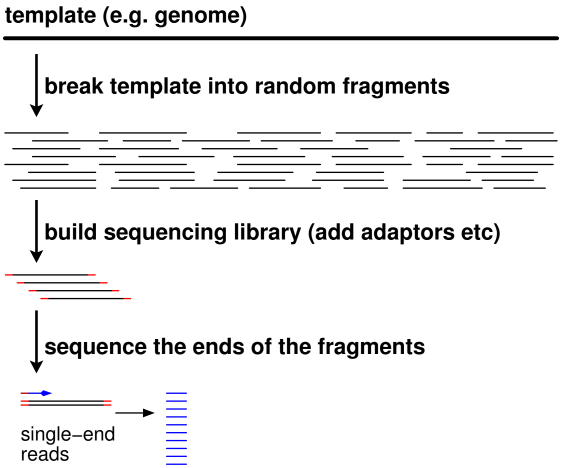
Shotgun sequencing



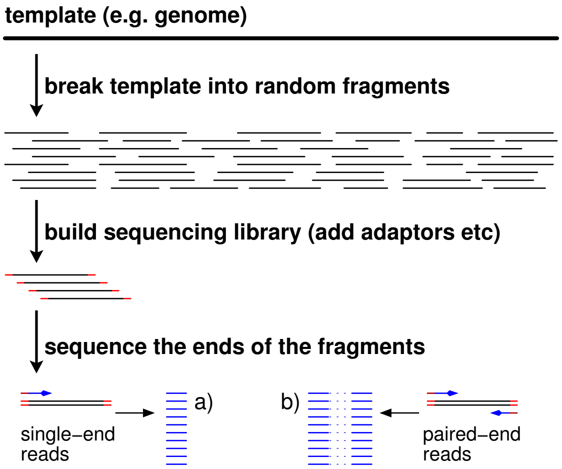
Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)

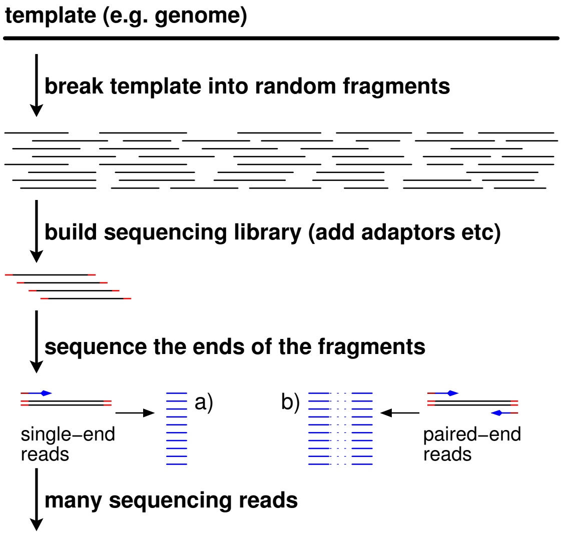
Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
single–end reads

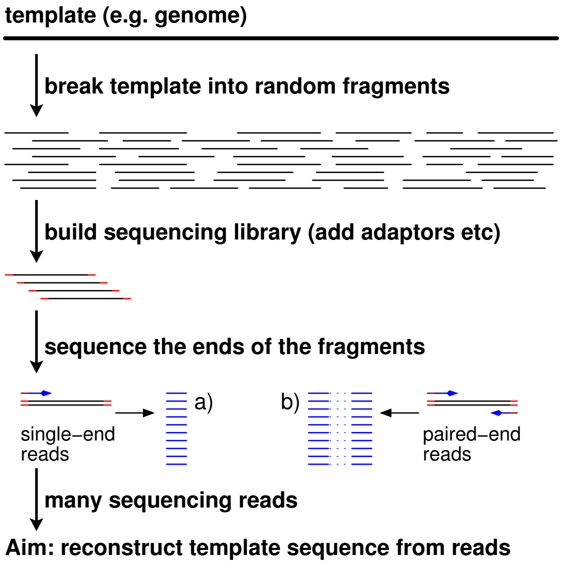
Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
single–end reads
a)
b)
paired–end reads

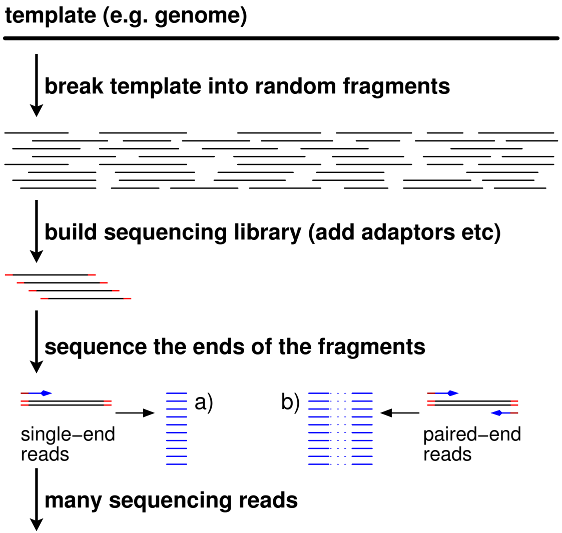
Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
single−end reads
a)
b)
paired−end reads
many sequencing reads

Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
single–end reads
a)
b)
paired–end reads
many sequencing reads
Aim: reconstruct template sequence from reads

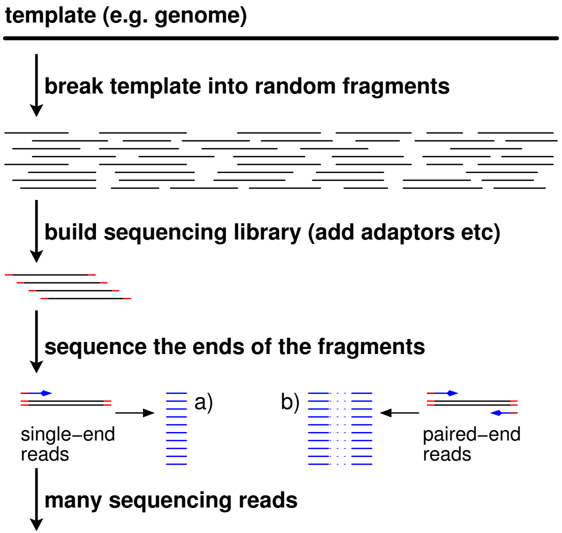
Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
a)
b)
single–end reads
paired–end reads
many sequencing reads

Aim: reconstruct template sequence from reads

– reference–guided assembly (map against reference)

Shotgun sequencing


template (e.g. genome)
break template into random fragments
build sequencing library (add adaptors etc)
sequence the ends of the fragments
single–end reads
a)
b)
paired–end reads
many sequencing reads

Aim: reconstruct template sequence from reads

– reference–guided assembly (map against reference)

– de–novo assembly (reconstruct directly from reads)

Reference-based Genome Assembly

- If a reference genome of the sequenced species exists (or a relatively close taxonomic relative), we can use it to guide the assembly.
- The reads are mapped to the reference genome using approximative search algorithms.
- The closer the reference is to the sequenced genome, the easier is the mapping and assembly.
- From mapped contiguous reads we construct consensus sequences - the contigs.

Why re-sequencing

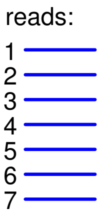
Why re-sequencing if a close reference genome already exists?

- Typically one does not re-sequence exactly the same individual the reference originated from.
- Usually one uses the reference to find
  - the differences in an individual carrying a disease (e.g. personalized medicine),
  - the characteristic changes in a new infectious virus (epidemiology),
  - the abundance of alleles in a population (population genetics),
  - or just to make the assembly of the (yet unassembled) genome of a related species a little bit easier.

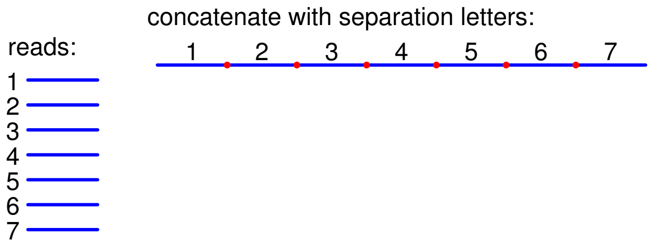
de novo Genome Assembly

- If no reference genome exists, assembling the sequenced genome is much harder.
- We have to find overlapping reads to stitch them together to longer and longer contigs.

Overlap Layout Consensus (as in CAP3): Overlap search

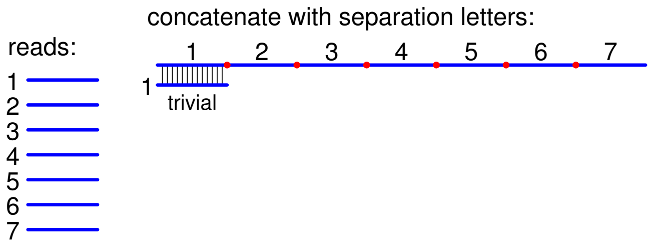


Overlap Layout Consensus (as in CAP3): Overlap search



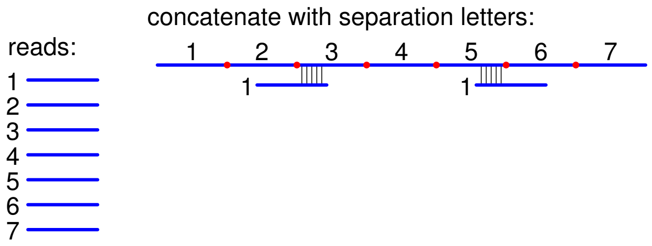
1. concatenate reads (separated by a separation character)

Overlap Layout Consensus (as in CAP3): Overlap search



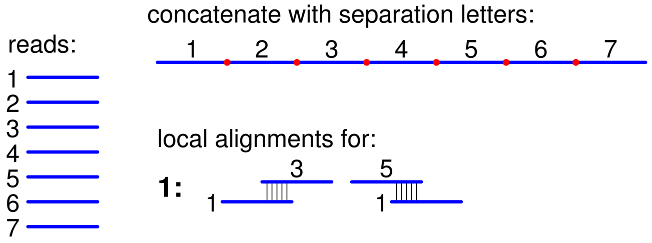
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



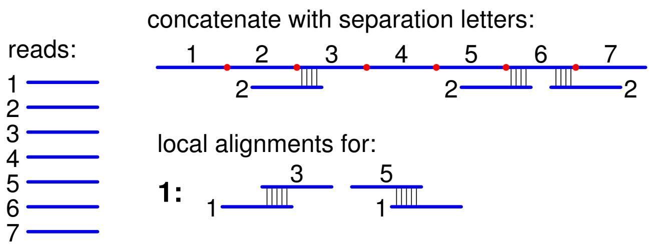
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
   - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



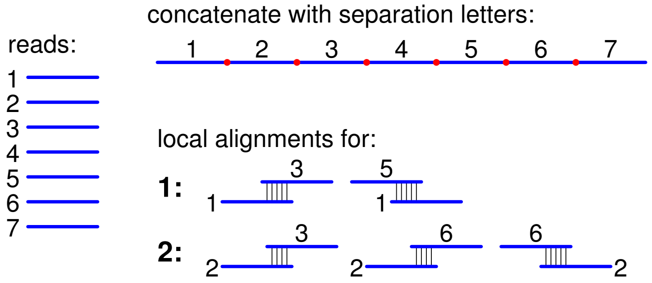
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
   - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



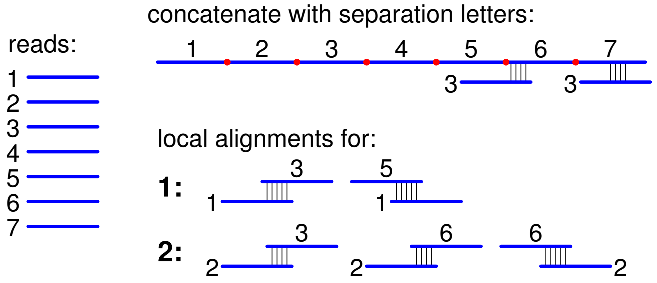
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



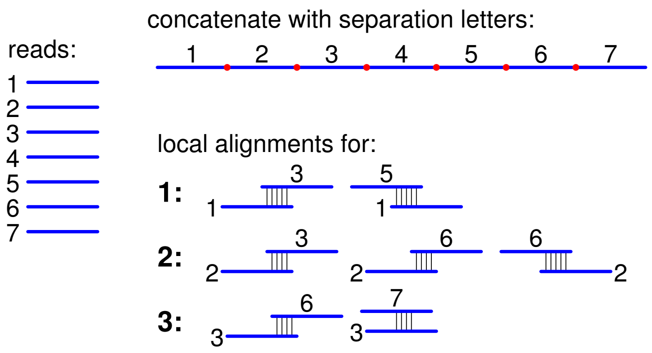
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



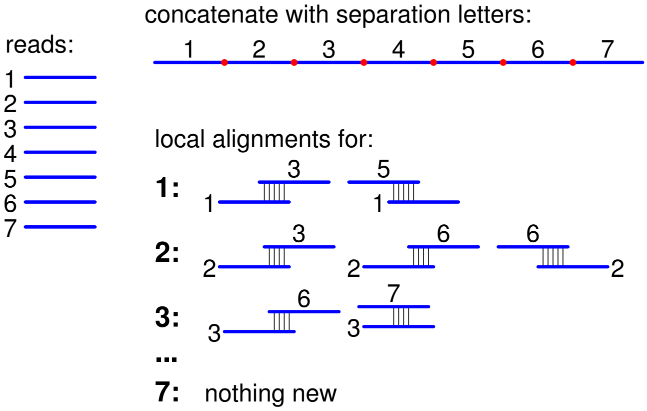
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



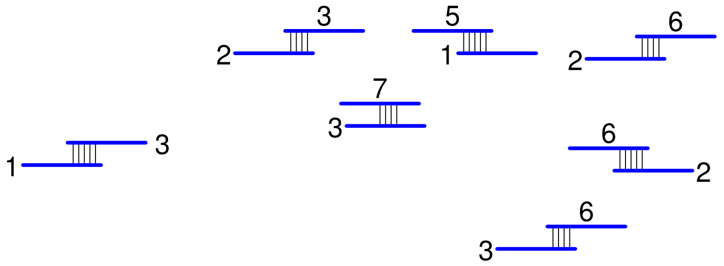
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)

Overlap Layout Consensus (as in CAP3): Overlap search



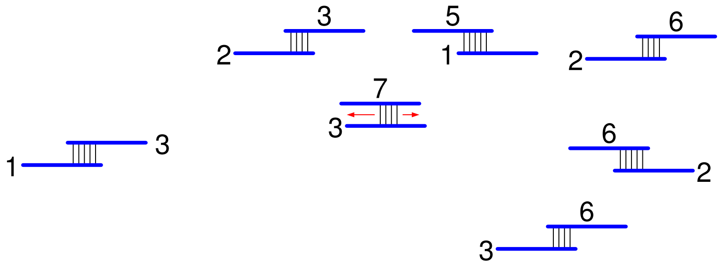
1. concatenate reads (separated by a separation character)
2. identify candidate overlaps (local alignments of reads against the concatenated string)
    - discard the trivial matches (i.e. read *i* matches itself)
    - each pair only once (result of *i* vs *j* should be identical to *j* vs *i*)

Overlap Layout Consensus (as in CAP3): Filtering



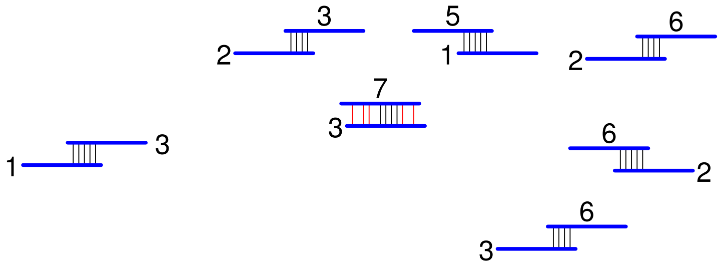
3. remove poor quality reads

Overlap Layout Consensus (as in CAP3): Filtering



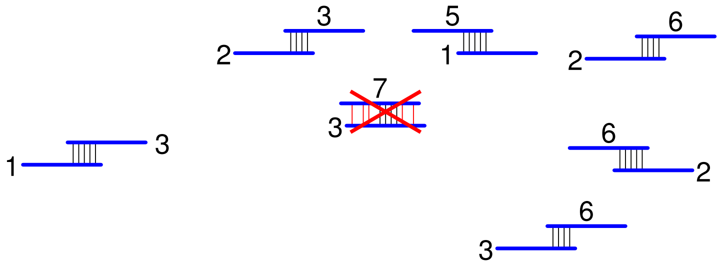
3. remove poor quality reads
4. compute global alignment for high quality pairs.

Overlap Layout Consensus (as in CAP3): Filtering



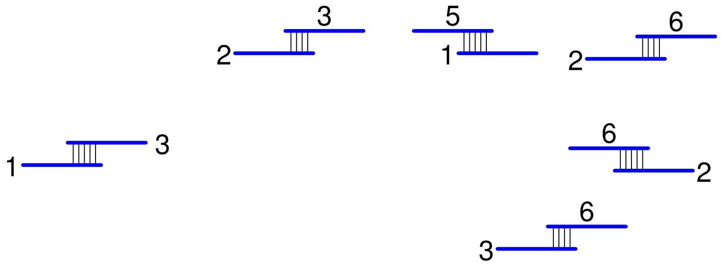
3. remove poor quality reads
4. compute global alignment for high quality pairs.
5. evaluate alignments due to
    1. minimum length
    2. minimum identity
    3. minimum similarity
    4. number of high-quality (true) mismatches

Overlap Layout Consensus (as in CAP3): Filtering

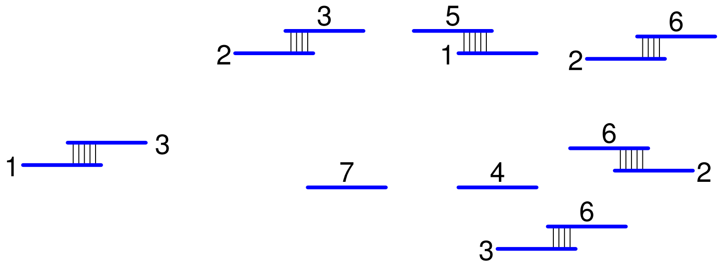


3. remove poor quality reads
4. compute global alignment for high quality pairs.
5. evaluate alignments due to
    1. minimum length
    2. minimum identity
    3. minimum similarity
    4. number of high-quality (true) mismatches
6. remove pairs that do not match thresholds 5.1-5.4.

Overlap Layout Consensus (as in CAP3): Contig building

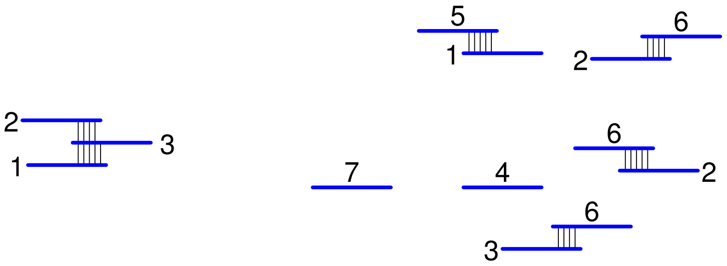


Overlap Layout Consensus (as in CAP3): Contig building



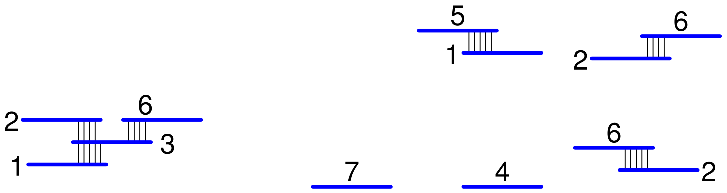
7. add all reads without overlaps

Overlap Layout Consensus (as in CAP3): Contig building



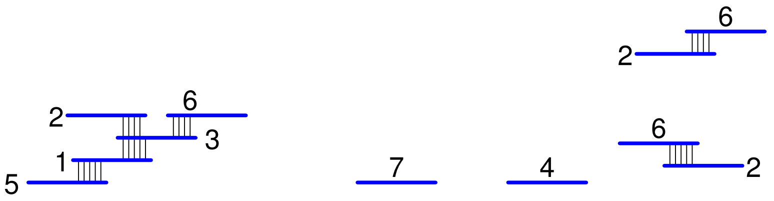
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)

Overlap Layout Consensus (as in CAP3): Contig building



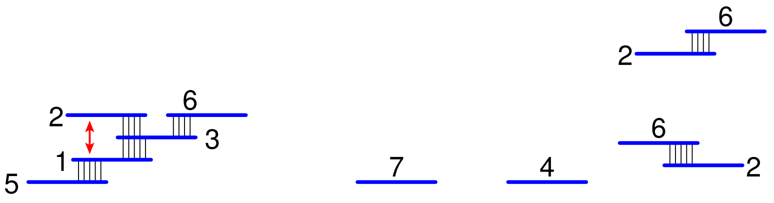
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)

Overlap Layout Consensus (as in CAP3): Contig building



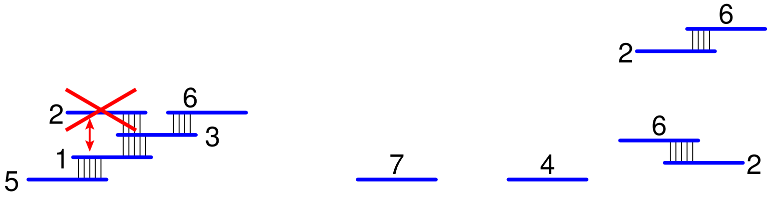
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)

Overlap Layout Consensus (as in CAP3): Contig building



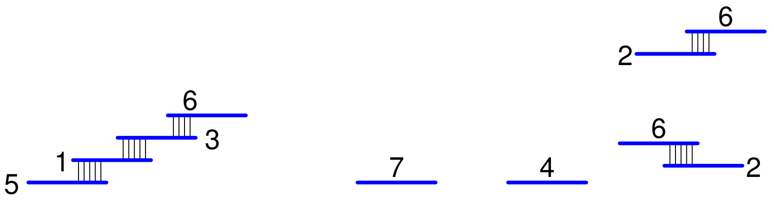
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



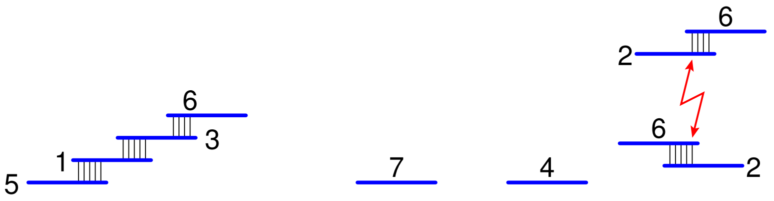
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
   1. in the layout (remove greedily from layout)
   2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



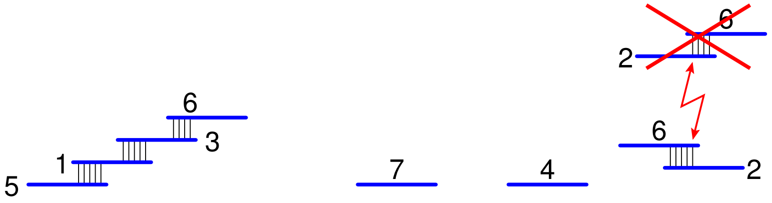
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



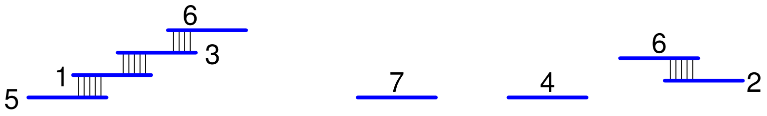
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



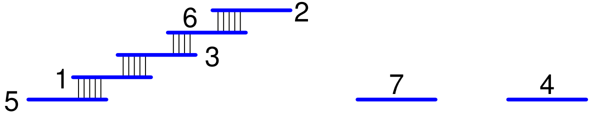
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
   1. in the layout (remove greedily from layout)
   2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



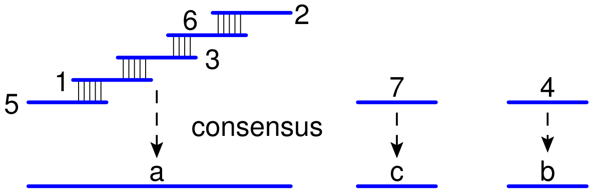
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



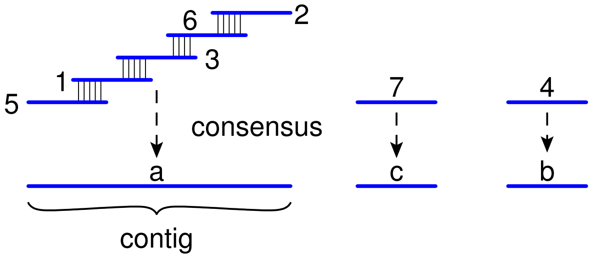
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)

Overlap Layout Consensus (as in CAP3): Contig building



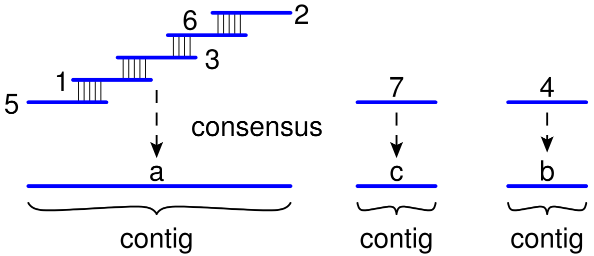
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)
10. construct consensus sequence for each contig

Overlap Layout Consensus (as in CAP3): Contig building



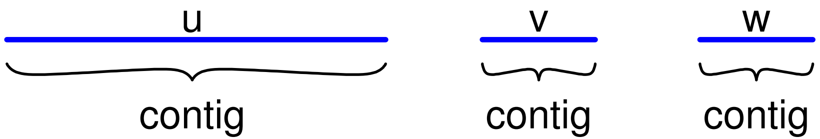
7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
   1. in the layout (remove greedily from layout)
   2. between overlap candidates (remove candidate greedily)
10. construct consensus sequence for each contig

Overlap Layout Consensus (as in CAP3): Contig building



7. add all reads without overlaps
8. generate a general layout using overlapping reads (ordered with decreasing overlap scores)
9. check for incompatibilities
    1. in the layout (remove greedily from layout)
    2. between overlap candidates (remove candidate greedily)
10. construct consensus sequence for each contig

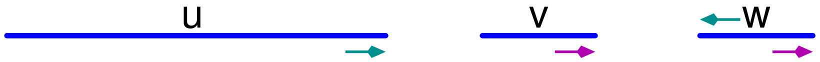
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended

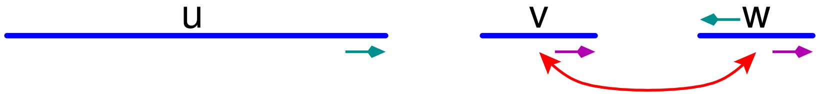
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)

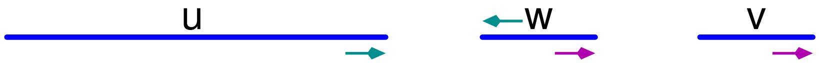
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other

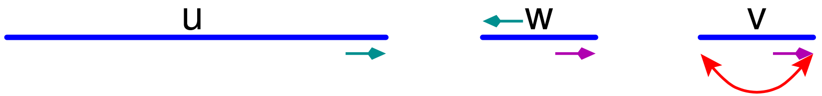
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other

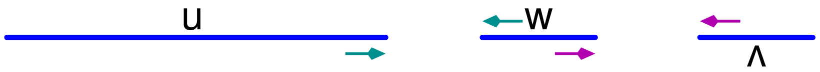
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other
14. orientate contigs according to the paired-ends

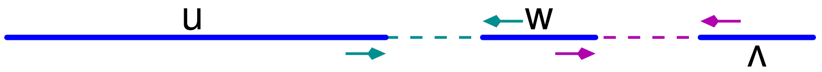
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other
14. orientate contigs according to the paired-ends

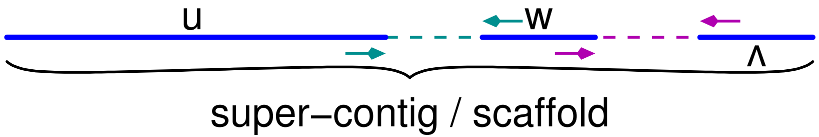
Overlap Layout Consensus (as in CAP3): Scaffolding



(now, assume that all above contigs were constructed from many shorter reads.)

11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other
14. orientate contigs according to the paired-ends
15. fill the gaps with N's according to the insert sizes used when preparing the sequencing library

Overlap Layout Consensus (as in CAP3): Scaffolding

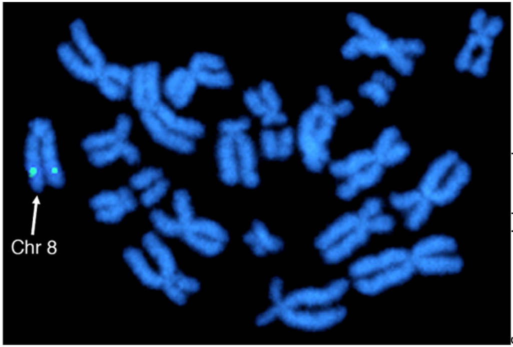


(now, assume that all above contigs were constructed from many shorter reads.)

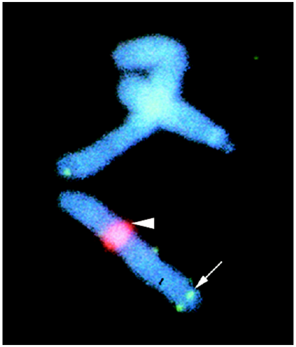
11. the set of contigs can often be extended
12. using additional information like paired-end reads (if available)
13. order contigs to bring matching paired-ends next to each other
14. orientate contigs according to the paired-ends
15. fill the gaps with N's according to the insert sizes used when preparing the sequencing library
16. the joined contigs are called super-contigs or scaffolds

Assembly Completeness and Contig Location

- usually it is not possible to easily assemble each chromosome into a single contig or scaffold
  (e.g. due to repeats, low quality regions, too low read coverage)
- thus, it can be important to locate scaffolds in the genome using, e.g., FISH (fluorescence in-situ hybridization) with genetic markers.



Source: www.stjuderesearch.org



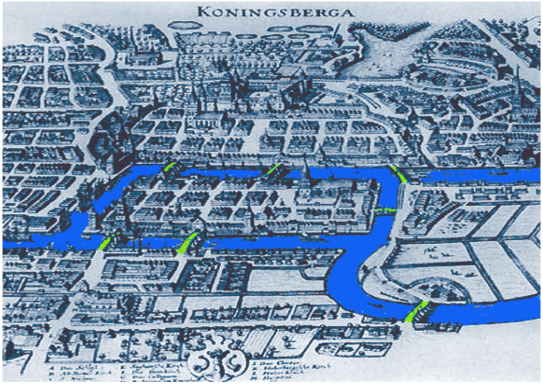
Source: Westbrook et al. (2008)
red chromosome marker, green probe

Assembly from 2nd and 3rd generation sequencing reads

- CAP3 has been developed for Sanger sequencing reads.
- NGS reads are typically shorter and come in huge numbers.
- Thus, also the overlaps are short, producing false positives easily.

- Assembly of NGS data works along the same principles.
- However they have to employ more elaborate methods to deal with the amount of data, the short overlaps and to efficiently detect false positive overlaps.
- A number of such tools apply approaches like *de Bruijn graphs*.

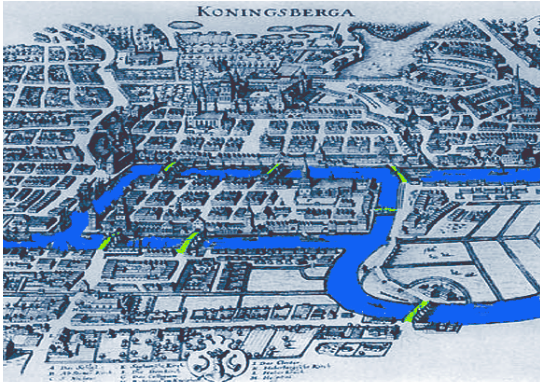
Excursion: Beginning of Graph Theory

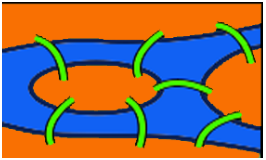


Leonhard Euler (1707-1783)

Popular 18th century problem:
Is there a walk through Königsberg using each bridge exactly once?

Excursion: Beginning of Graph Theory

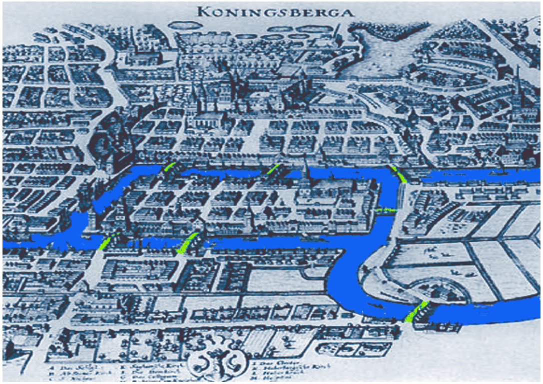


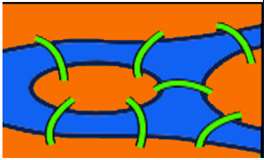


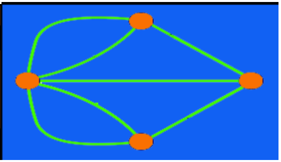
Leonhard Euler (1707-1783)

Popular 18th century problem: *Is there a walk through Königsberg using each bridge exactly once?*

Excursion: Beginning of Graph Theory







Leonhard Euler (1707-1783)

Popular 18th century problem: *Is there a walk through Königsberg using each bridge exactly once?*

Euler Paths and Tours

Given an undirected connected graph $G = (V, E)$ with nodes V and edges E, we define:

Euler Paths and Tours

Given an undirected connected graph $G = (V, E)$ with nodes V and edges E, we define:

Euler Path or Trail

is a path visiting each edge (of a graph) exactly once.

Euler Paths and Tours

Given an undirected connected graph $G = (V, E)$ with nodes V and edges E, we define:

Euler Path or Trail

is a path visiting each edge (of a graph) exactly once.

Euler Tour or Circuit or Cycle

is a path visiting each edge (of a graph) exactly once <u>and</u> ending at the starting point.

Euler Paths and Tours: Solution?

Leonhard Euler (1735) showed that the existence of such paths or tours depend on the degree of the nodes.

Euler Paths and Tours: Solution?

Leonhard Euler (1735) showed that the existence of such paths or tours depend on the degree of the nodes.

Euler Path

exists if there are exactly 0 or 2 nodes with uneven degree (i.e. number of attached edges).

Euler Paths and Tours: Solution?

Leonhard Euler (1735) showed that the existence of such paths or tours depend on the degree of the nodes.

Euler Path

exists if there are exactly 0 or 2 nodes with uneven degree (i.e. number of attached edges).

Euler Tour

exists iff the graph is connected and has no nodes of odd degree.

Euler Paths and Tours: Solution?

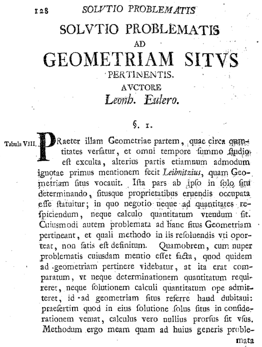
Leonhard Euler (1735) showed that the existence of such paths or tours depend on the degree of the nodes.

Euler Path

exists if there are exactly 0 or 2 nodes with uneven degree (i.e. number of attached edges).

Euler Tour

exists iff the graph is connected and has no nodes of odd degree.



128 *SOLVTIO PROBLEMATIS*

SOLVTIO PROBLEMATIS
AD
GEOMETRIAM SITVS
PERTINENTIS.
AVCTORE
Leonh. Eulero.

§. 1.

Tabula VIII. PRaeter illam Geometriae partem, quae circa quantitates verſatur, et omni tempore ſummo ſtudio eſt exculta, alterius partis etiamnum admodum ignotae primus mentionem fecit *Leibnitzius*, quam Geometriam ſitus vocauit. Iſta pars ab ipſo in ſolo ſitu determinando, ſitusque proprietatibus eruendis occupata eſſe ſtatuitur; in quo negotio neque ad quantitates reſpiciendum, neque calculo quantitatum vtendum ſit. Cuiusmodi autem problemata ad hanc ſitus Geometriam pertineant, et quali methodo in iis reſoluendis vti oporteat, non ſatis eſt definitum. Quamobrem, cum nuper problematis cuiusdam mentio eſſet facta, quod quidem ad geometriam pertinere videbatur, at ita erat comparatum, vt neque determinationem quantitatum requireret, neque ſolutionem calculi quantitatum ope admitteret, id ad geometriam ſitus referre haud dubitaui: praeſertim quod in eius ſolutione ſolus ſitus in conſiderationem veniat, calculus vero nullius prorſus ſit vſus. Methodum ergo meam quam ad huius generis problemata

Euler Paths and Tours on directed graphs

In directed graphs, that means, that edges have only one direction in which they can be crossed...

Euler Tour

exists iff the in-degree of each node is equal to its out-degree and if the graph is a strongly connected component, i.e. every node is reachable from every other node via a directed path.

Euler Paths and Tours on directed graphs

In directed graphs, that means, that edges have only one direction in which they can be crossed...

Euler Tour

exists iff the in-degree of each node is equal to its out-degree and if the graph is a strongly connected component, i.e. every node is reachable from every other node via a directed path.

Euler Path

exists iff there at most one node with *in-degree* - *out-degree* $= 1$ exists at most one node with *out-degree* - *in-degree* $= 1$.

Note, in graph theory the terms *node* and *vertex* (pl. vertices) are used interchangeably, as are *directed edge* and *arc*.

Variation: Hamiltonian Paths and Tours

William Rowan Hamilton
(1805-1865)

Variation: Hamiltonian Paths and Tours

Hamiltonian Path (or traceable path)

is a path that visits every node (of a graph) exactly once.

William Rowan Hamilton (1805-1865)

Variation: Hamiltonian Paths and Tours

Hamiltonian Path (or traceable path)

is a path that visits every node (of a graph) exactly once.

Hamiltonian Tour

is a Hamiltonian path ending at its starting point.

William Rowan Hamilton (1805-1865)

Variation: Hamiltonian Paths and Tours

Hamiltonian Path (or traceable path)

is a path that visits every node (of a graph) exactly once.

Hamiltonian Tour

is a Hamiltonian path ending at its starting point.

Problems:

- determining whether such a path/tour exists is NP-complete.

William Rowan Hamilton (1805-1865)

Variation: Hamiltonian Paths and Tours

Hamiltonian Path (or traceable path)

is a path that visits every node (of a graph) exactly once.

Hamiltonian Tour

is a Hamiltonian path ending at its starting point.

Problems:

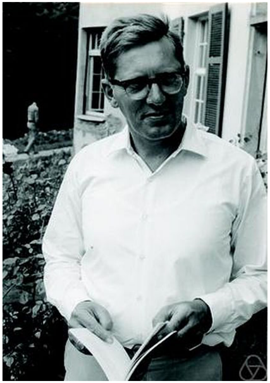
- determining whether such a path/tour exists is NP-complete.
- Hamiltonian Tours are a special case of the Traveling Salesman Problem.

William Rowan Hamilton (1805-1865)

Nicolaas de Bruijn

In 1946 Nicolaas de Bruijn got interested in the superstring problem:

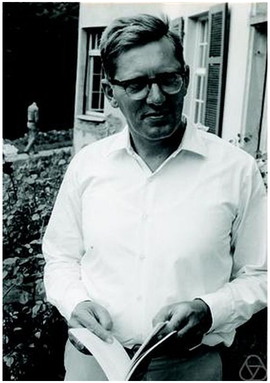
- *Find the shortest circular superstring that contains all possible k-mers as substrings.*


Nicolaas de Bruijn (1918-2012)

Nicolaas de Bruijn

In 1946 Nicolaas de Bruijn got interested in the superstring problem:

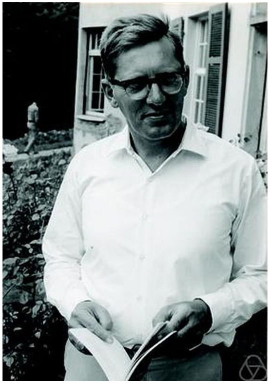
- *Find the shortest circular superstring that contains all possible k-mers as substrings.*
- There are n^k k-mers for an alphabet of size $n = |\Sigma|$.


Nicolaas de Bruijn (1918-2012)

Nicolaas de Bruijn

In 1946 Nicolaas de Bruijn got interested in the superstring problem:

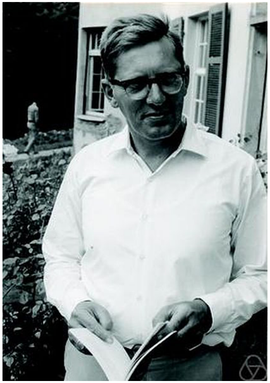
- *Find the shortest circular superstring that contains all possible k-mers as substrings.*
- There are n^k k-mers for an alphabet of size $n = |\Sigma|$.
- For example, the number DNA-triplets


Nicolaas de Bruijn (1918-2012)

Nicolaas de Bruijn

In 1946 Nicolaas de Bruijn got interested in the superstring problem:

- *Find the shortest circular superstring that contains all possible k-mers as substrings.*
- There are n^k k-mers for an alphabet of size $n = |\Sigma|$.
- For example, the number DNA-triplets: $n^k = 4^3 = 64$.



Nicolaas de Bruijn (1918-2012)

De Bruijn Graphs

De Bruijn Graph

- nodes: for all possible $(k-1)$-mers

De Bruijn Graphs

De Bruijn Graph

- nodes: for all possible $(k-1)$-mers
- edges: directed links between nodes **a** and **b** if the $k-2$-long prefix of **b** is the suffix of **a**.

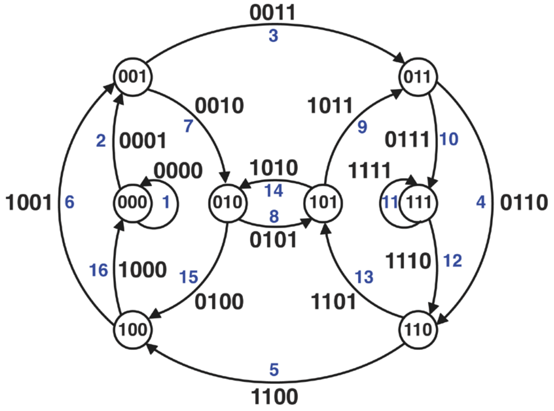
De Bruijn Graphs

De Bruijn Graph

- nodes: for all possible $(k-1)$-mers
- edges: directed links between nodes **a** and **b** if the $k-2$-long prefix of **b** is the suffix of **a**.
- Note: A Eulerian Tour exists because every node have one in-edge and one out-edge for each character in Σ.

De Bruijn Graph Example

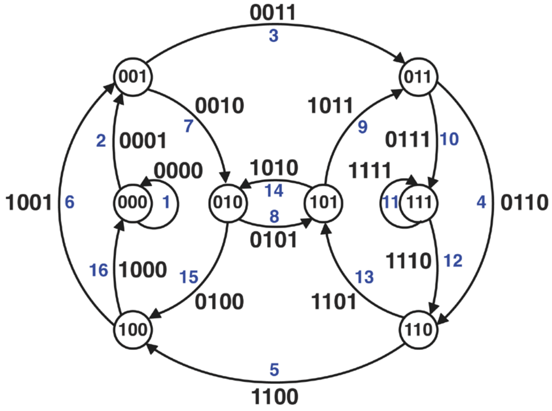
De Bruijn Graph for $k = 4$ and $\Sigma = \{0, 1\}$:



- The nodes are labeled with 000, 001, 010, 011, 100, 101, 110, 111.
- The edges are labeled linking the overlapping node labels, e.g. 100→ 001 with 1001.

De Bruijn Graph Example

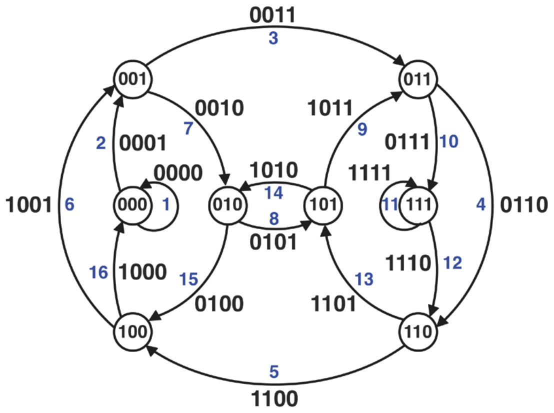
De Bruijn Graph for $k = 4$ and $\Sigma = \{0, 1\}$:



- The nodes are labeled with 000, 001, 010, 011, 100, 101, 110, 111.
- The edges are labeled linking the overlapping node labels, e.g. 100$\rightarrow$ 001 with 1001.
- An Eulerian Tour exists, each node has in-degree and out-degree 2.

De Bruijn Graph Example

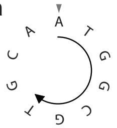
De Bruijn Graph for $k = 4$ and $\Sigma = \{0, 1\}$:



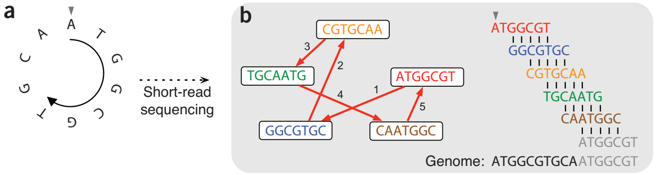
- The nodes are labeled with 000, 001, 010, 011, 100, 101, 110, 111.
- The edges are labeled linking the overlapping node labels, e.g. 100$\rightarrow$ 001 with 1001.
- An Eulerian Tour exists, each node has in-degree and out-degree 2.
- The Eulerian Tour (marked by blue numbers) spells out the circular superstring: 0000110010111101.

De Bruijn Graphs in Sequence Assembly

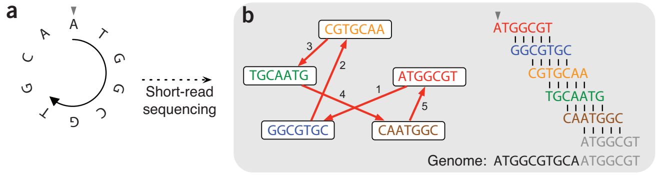
a



De Bruijn Graphs in Sequence Assembly

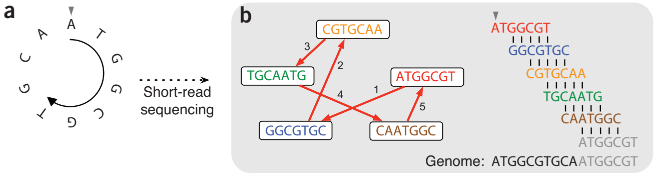


De Bruijn Graphs in Sequence Assembly



split reads into *k*-mers

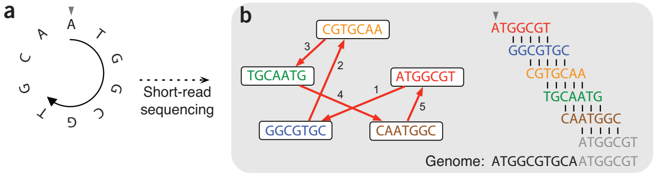
De Bruijn Graphs in Sequence Assembly



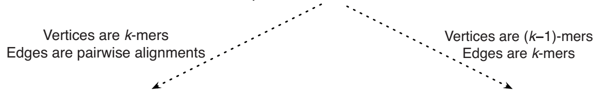
split reads into *k*-mers

Vertices are *k*-mers
Edges are pairwise alignments

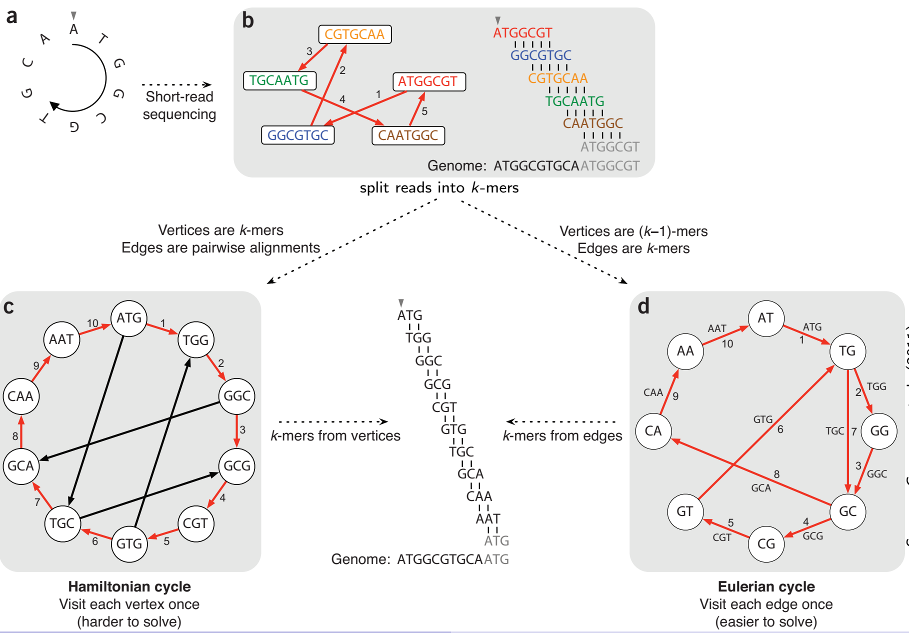
De Bruijn Graphs in Sequence Assembly



split reads into *k*-mers



De Bruijn Graphs in Sequence Assembly



Source: Compeau et al. (2011)

Sequence De Bruijn Graph (from reads to graph)

read 1:
G G A C T A A A T

- construct a de Bruijn graph for each read with $k-1=3$

Sequence De Bruijn Graph (from reads to graph)

read 1:

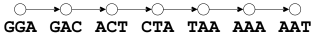
```
G   G   A   C   T   A   A   A   T
G   G   A
    G   A   C
        A   C   T
            C   T   A
                T   A   A
                    A   A   A
                        A   A   T
```

- construct a de Bruijn graph for each read with $k - 1 = 3$
- split the read into overlapping $k - 1$mers

Sequence De Bruijn Graph (from reads to graph)

read 1:

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

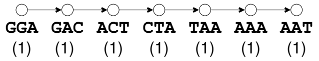


- construct a de Bruijn graph for each read with $k-1=3$
- split the read into overlapping $k-1$mers
- and construct the de Bruijn graph

Sequence De Bruijn Graph (from reads to graph)

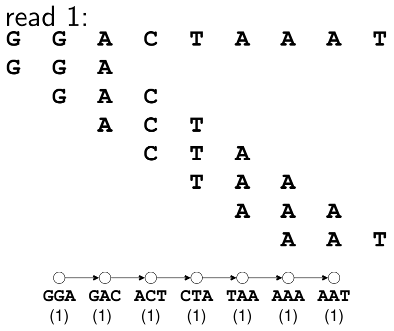
read 1:

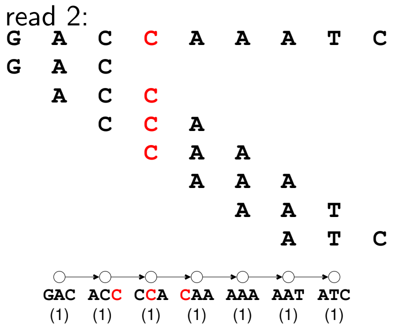
```
G G A C T A A A T
G G A
  G A C
    A C T
      C T A
        T A A
          A A A
            A A T
```



- construct a de Bruijn graph for each read with $k - 1 = 3$
- split the read into overlapping $k - 1$mers
- and construct the de Bruijn graph
- count 1 for the read at each node

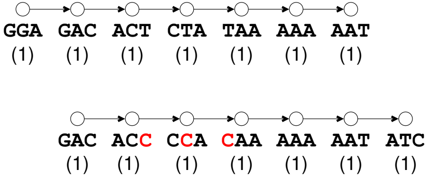
Sequence De Bruijn Graph (from reads to graph)





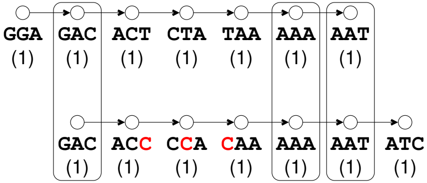
- construct a de Bruijn graph for each read with $k - 1 = 3$
- split the read into overlapping $k - 1$mers
- and construct the de Bruijn graph
- count 1 for the read at each node

Sequence De Bruijn Graph (merging identical nodes)



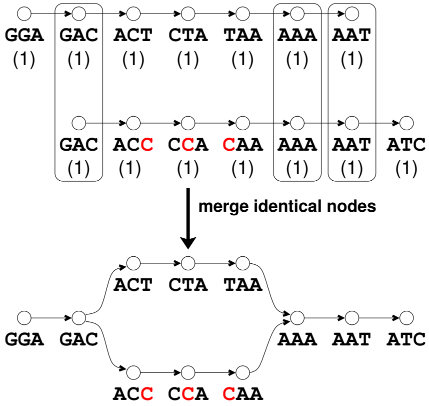
- take all 'read graphs'

Sequence De Bruijn Graph (merging identical nodes)



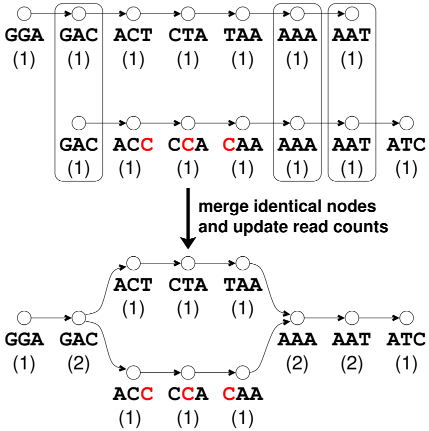
- take all 'read graphs'
- . . . and merge identical nodes

Sequence De Bruijn Graph (merging identical nodes)



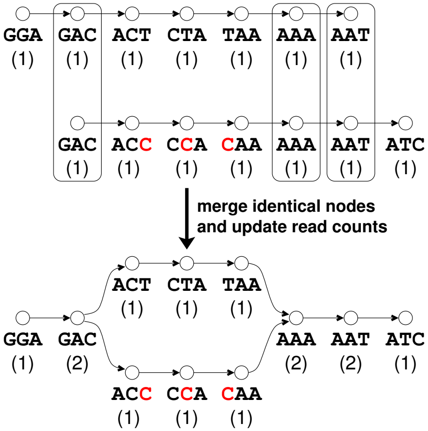
- take all 'read graphs'
- . . . and merge identical nodes
- . . . gaining a large de Bruijn graph

Sequence De Bruijn Graph (merging identical nodes)



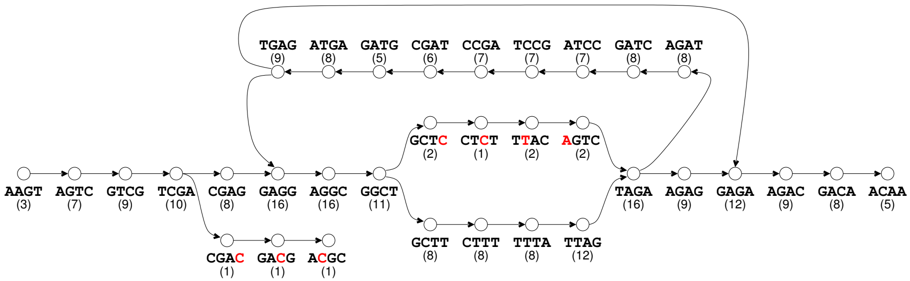
- take all 'read graphs'
- . . . and merge identical nodes
- . . . gaining a large de Bruijn graph
- sum up the read counts at the merged node

Sequence De Bruijn Graph (merging identical nodes)

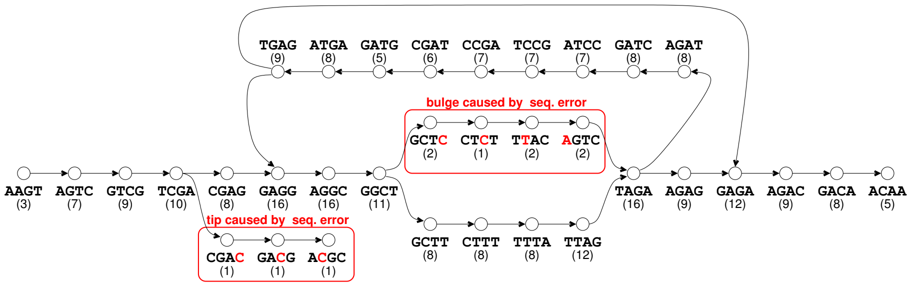


- take all 'read graphs'
- ... and merge identical nodes
- ... gaining a large de Bruijn graph
- sum up the read counts at the merged node
- do repeat this for all other reads

Sequence De Bruijn Graph (reducing the graph to contigs)

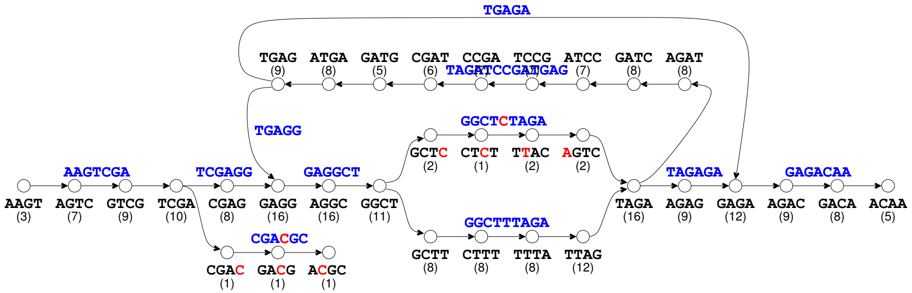


Sequence De Bruijn Graph (reducing the graph to contigs)



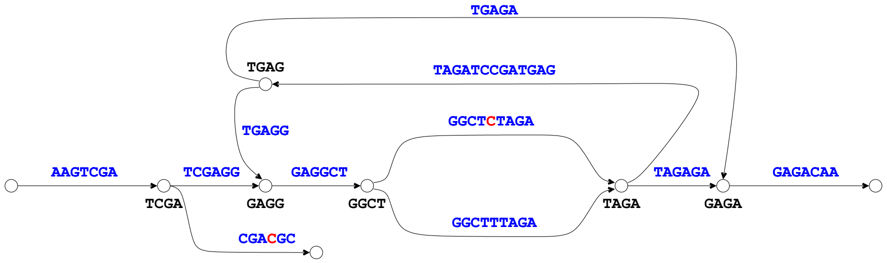
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds

Sequence De Bruijn Graph (reducing the graph to contigs)



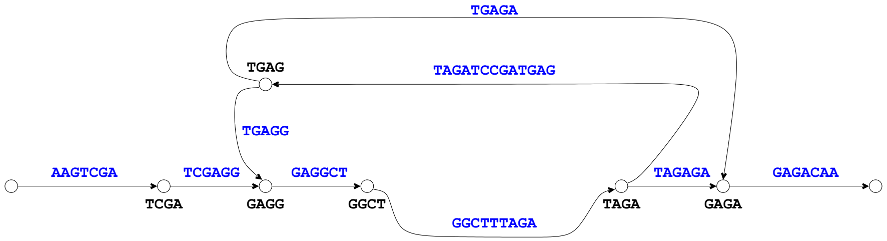
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds
- compactify the graph by merging the branch labels (rf. de Bruijn graph definition) on non-branching paths

Sequence De Bruijn Graph (reducing the graph to contigs)



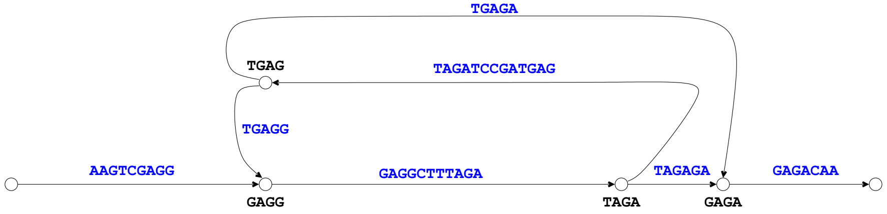
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds
- compactify the graph by merging the branch labels (rf. de Bruijn graph definition) on non-branching paths
- delete the obsolete nodes and re-link the branching nodes

Sequence De Bruijn Graph (reducing the graph to contigs)



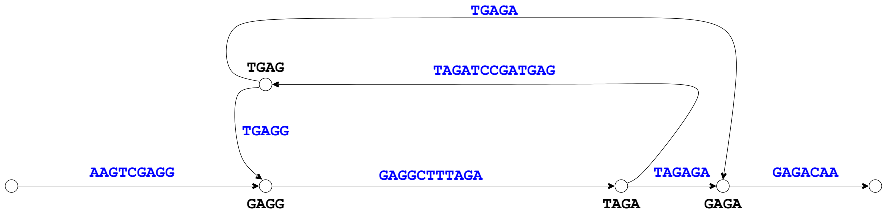
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds
- compactify the graph by merging the branch labels (rf. de Bruijn graph definition) on non-branching paths
- delete the obsolete nodes and re-link the branching nodes
- remove the tips and bulges with low read counts (sequencing errors)

Sequence De Bruijn Graph (reducing the graph to contigs)



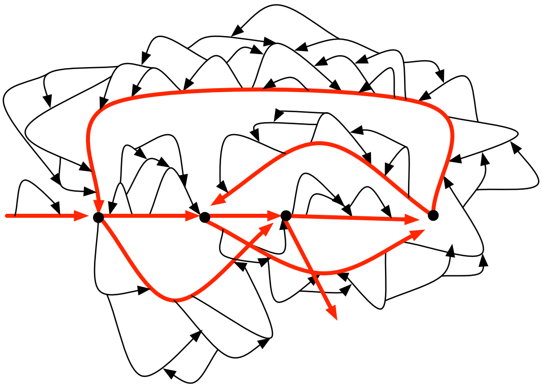
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds
- compactify the graph by merging the branch labels (rf. de Bruijn graph definition) on non-branching paths
- delete the obsolete nodes and re-link the branching nodes
- remove the tips and bulges with low read counts (sequencing errors)
- re-compactify the graph

Sequence De Bruijn Graph (reducing the graph to contigs)



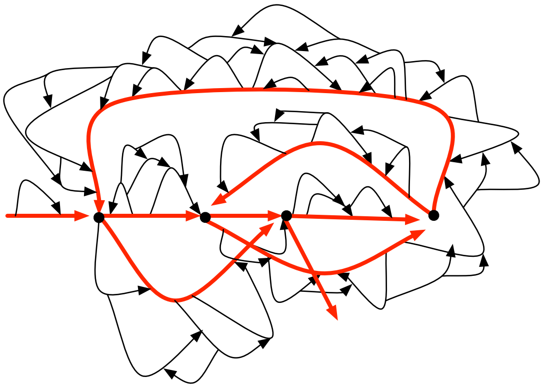
- sequencing errors cause tips and bulges with low read counts and can be identified using thresholds
- compactify the graph by merging the branch labels (rf. de Bruijn graph definition) on non-branching paths
- delete the obsolete nodes and re-link the branching nodes
- remove the tips and bulges with low read counts (sequencing errors)
- re-compactify the graph
- all the labels on the non-branching paths are our reconstructed contigs

De Bruijn Graphs for Assembly with Sequencing Errors



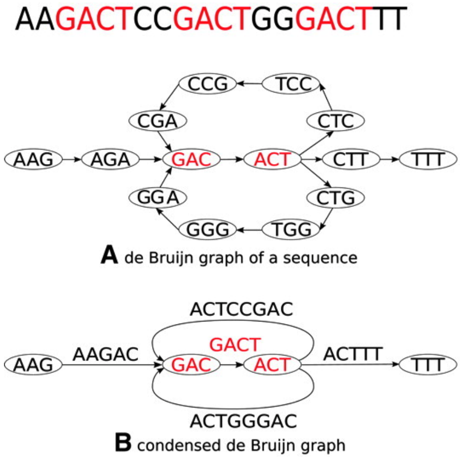
- The number of reads participating in bulges and tips tell us which are the frequent, and thus likely true ones.

De Bruijn Graphs for Assembly with Sequencing Errors



- The number of reads participating in bulges and tips tell us which are the frequent, and thus likely true ones.
- Bulges and tips with few reads are removed.

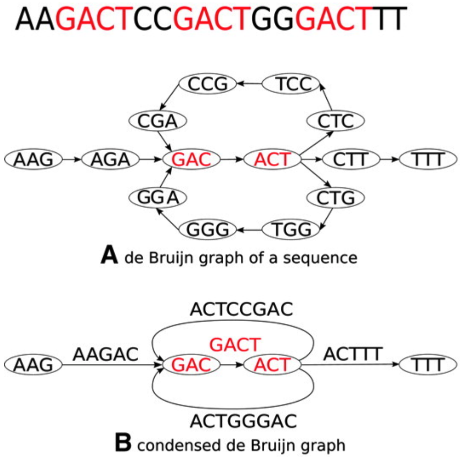
De Bruijn Graphs for Assembly with Repeats



Source: Chaisson et al. (2009)

- In the case of repeats, Euler paths are not possible, because the edges of the repeated region have to be used repeatedly.

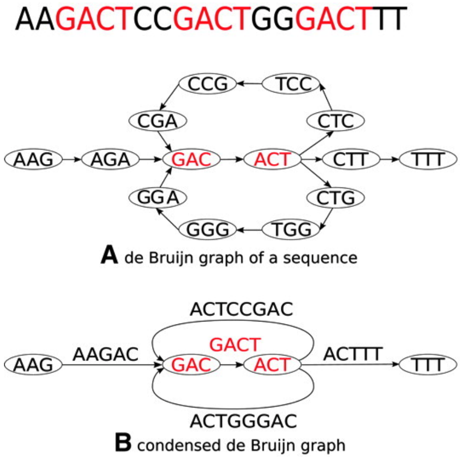
De Bruijn Graphs for Assembly with Repeats



Source: Chaisson et al. (2009)

- In the case of repeats, Euler paths are not possible, because the edges of the repeated region have to be used repeatedly.
- Typically the order in which edges from a repeat have to be followed cannot be determined. Then the paths have to be kept as separate contigs.

De Bruijn Graphs for Assembly with Repeats



Source: Chaisson et al. (2009)

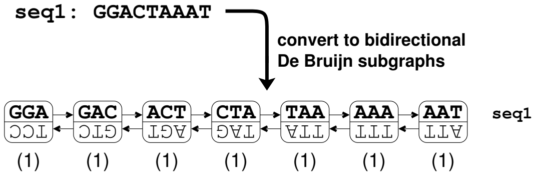
- In the case of repeats, Euler paths are not possible, because the edges of the repeated region have to be used repeatedly.
- Typically the order in which edges from a repeat have to be followed cannot be determined. Then the paths have to be kept as separate contigs.
- Exception: if we have reads or paired-end information, which reach longer than the repeat, this helps to order the contigs.

Bidirected de Bruijn Graphs (Medvedev et al. 2007)

```
seq1: GGACTAAAT
```

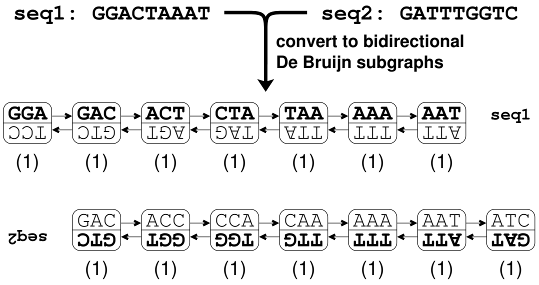
- Problem: How do we know which strand our read is from? We don't!

Bidirected de Bruijn Graphs (Medvedev et al. 2007)



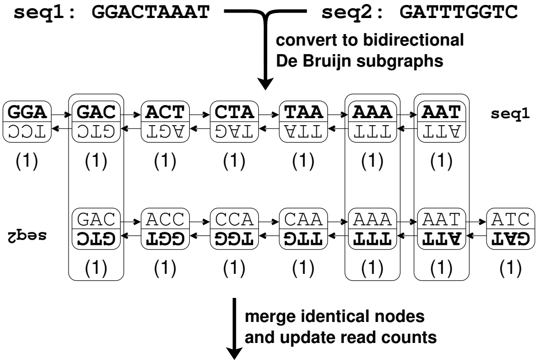
- Problem: How do we know which strand our read is from? We don't!
- With bidirected de Bruijn Graphs one can cover k-mers and their complements.

Bidirected de Bruijn Graphs (Medvedev et al. 2007)



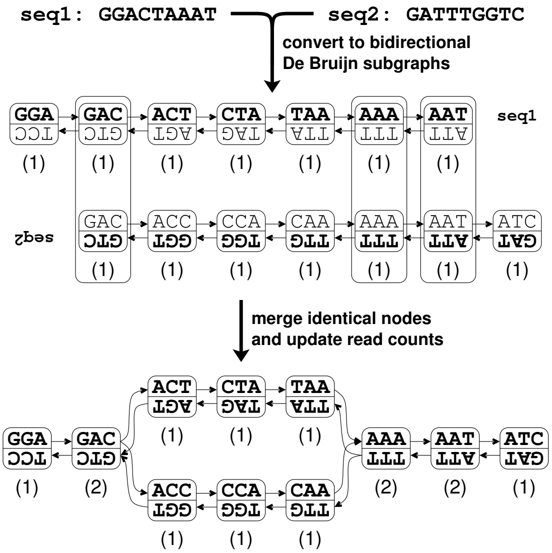
- Problem: How do we know which strand our read is from? We don't!
- With bidirected de Bruijn Graphs one can cover k-mers and their complements.

Bidirected de Bruijn Graphs (Medvedev et al. 2007)


seq1: GGACTAAAT
seq2: GATTTGGTC
convert to bidirectional De Bruijn subgraphs
GGA
GAC
ACT
CTA
TAA
AAA
AAT
seq1
(1)
GAC
ACC
CCA
CAA
AAA
AAT
ATC
seq2
merge identical nodes and update read counts

- Problem: How do we know which strand our read is from? We don't!
- With bidirected de Bruijn Graphs one can cover k-mers and their complements.

Bidirected de Bruijn Graphs (Medvedev et al. 2007)



- Problem: How do we know which strand our read is from? We don't!
- With bidirected de Bruijn Graphs one can cover k-mers and their complements.
- Note:
  At no time two nodes with identical strings can exist in one (sub)graph, and both strings have to be treated equally.

Some measures on genome sequencing and assembly

Coverage

Coverage describes the average number of times a nucleotide in the template DNA has been sequenced which is equivalent to the number of reads that cover each nucleotide on average.

$$coverage = \frac{\sum_{i \in \{all\ \ reads\}} length\ of\ \ read\ \ i}{length\ of\ \ template\ or\ genome}$$

Rule of thumb: the higher the better!

Some measures on genome sequencing and assembly

The quality of an assembly is hard to measure. Typically several values are used like

- maximum contig/scaffold length
- average contig/scaffold length
- combined total length
- the N50 or the NG50 value

N50 and NG50

N50 value

All contigs/scaffold are ordered descending in size. Starting from the largest contig/scaffold add their lengths. The N50 value is the length of the first contig, for which this sum of contig lengths covers $\geq$ 50% of the total length of contigs/scaffolds, i.e. the entire assembly.

Rule of thumb: the longer the better!

NG50 value

All contigs/scaffold are ordered descending in size. Starting from the largest contig/scaffold add their lengths. The NG50 value is the length of the first contig, for which this sum of contig lengths covers $\geq$ 50% of the total length of the sequenced genome.

Rule of thumb: the longer the better!

Sometimes other percentages than 50% are used leading to, e.g. N70 etc.

Reference based assembly

The whole procedure gets much easier if we have a reference genome available.

Reference based assembly

The whole procedure gets much easier if we have a reference genome available.

- mapping using search tools like BLAST or BLAT

Reference based assembly

The whole procedure gets much easier if we have a reference genome available.

- mapping using search tools like BLAST or BLAT
- dynamic programming (e.g. Smith-Waterman) with pre-filtering to keep the candidate regions small, using
  - *hash-based k-mer index*
  - *spaced-seeds index*
- Approaches using the *Burrows-Wheeler-Transform* (BWT) of the reference sequence,

and the mapped reads are then summarized to contigs using consensus approaches.

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
...A C T A T C A T C G T A C A C A T...   reference sequence

   A C T A T C A T T G T A C A C A T      query sequence
```

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
        1 1 1 1 1 1 1 1 1                seed encoding
...A C T A T C A T C G T A C A C A T...  reference sequence
        T C A T C G T A C                seed sequence (len=9)
   A C T A T C A T T G T A C A C A T     query sequence
```

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
   1 1 1 1 1 1 1 1 1                     seed encoding
...A C T A T C A T C G T A C A C A T...  reference sequence
   A C T A T C A T C                     seed sequence (len=9)
   A C T A T C A T T G T A C A C A T     query sequence
```

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
                    1 1 1 1 1 1 1 1 1     seed encoding
...A C T A T C A T C G T A C A C A T...   reference sequence
                    C G T A C A C A T     seed sequence (len=9)
   A C T A T C A T T G T A C A C A T      query sequence
```

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
                  1 1 1 1 1 1 1 1 1      seed encoding
...A C T A T C A T C G T A C A C A T...  reference sequence
                  C G T A C A C A T      seed sequence (len=9)
   A C T A T C A T T G T A C A C A T     query sequence
```

- Another way is to use *spaced seeds*, i.e. that only certain letters in a longer word have to match.

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

```
                    1 1 1 1 1 1 1 1 1         seed encoding
... A C T A T C A T C G T A C A C A T ...     reference sequence
                    C G T A C A C A T         seed sequence (len=9)
    A C T A T C A T T G T A C A C A T         query sequence
```

- Another way is to use *spaced seeds*, i.e. that only certain letters in a longer word have to match.

```
    1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1         seed encoding
... A C T A T C A T C G T A C A C A T ...     reference sequence
    A . . . T C A . . . T A C . . A T         spaced seed (weight=9, len=17)
    A C T A T C A T T G T A C A C A T         query sequence
```

Hash-based approaches

- Hash-based approaches typically require matching seed sequences (one or several) to identify candidate regions to be checked (similar to Baeza-Yeates-Perleberg)
- often contiguous seeds are used (e.g. perfectly matching words of length k)

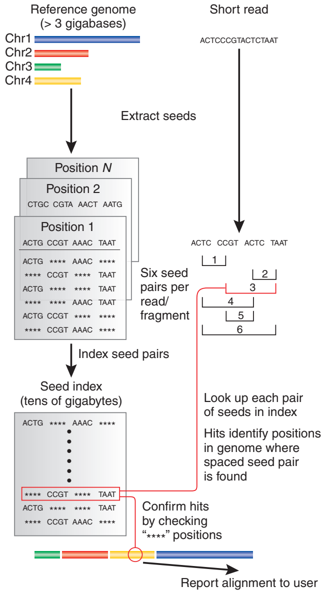
```
                   1 1 1 1 1 1 1 1 1      seed encoding
...A C T A T C A T C G T A C A C A T...   reference sequence
                   C G T A C A C A T      seed sequence (len=9)
   A C T A T C A T T G T A C A C A T      query sequence
```

- Another way is to use *spaced seeds*, i.e. that only certain letters in a longer word have to match.

```
   1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1      seed encoding
...A C T A T C A T C G T A C A C A T...   reference sequence
   A . . . T C A . . . T A C . . A T      spaced seed (weight=9, len=17)
   A C T A T C A T T G T A C A C A T      query sequence
```

- It has been shown that the use of *spaced seeds* is much more sensitive, missing less hits. Especially, when using sets of spaced seeds.

Hash-based mapping



- mapping with candidate filtering based on (spaced) seed matches is easy to implement
- however, to generate a typical seed index is memory-intense (about 50GB for the human genome of 3 Gbp)
- the example uses six spaced seeds (1111111100000000, 0000000011111111, 0000111100001111, 1111000011110000, 0000111111110000, 1111000000001111)
- from all candidates the actual best hit position of the read has to be found by alignment and reported

Source: Trapnell+Salzberg (2009)

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:

```
^ B A N A N A $
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end: **generate all rotations:**

```
                         ^ B A N A N A $

^ B A N A N A $
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:

```
^ B A N A N A $
```

generate all rotations:

```
^ B A N A N A $
$ ^ B A N A N A
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:

```
^ B A N A N A $
```

generate all rotations:

```
^ B A N A N A $
$ ^ B A N A N A
A $ ^ B A N A N
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:

```
^ B A N A N A $
```

generate all rotations:

```
^ B A N A N A $
$ ^ B A N A N A
A $ ^ B A N A N
N A $ ^ B A N A
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:	**generate all rotations:**
^ B A N A N A $	^ B A N A N A $
	$ ^ B A N A N A
	A $ ^ B A N A N
	N A $ ^ B A N A
	A N A $ ^ B A N
	N A N A $ ^ B A
	A N A N A $ ^ B
	B A N A N A $ ^

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:	**generate all rotations:**	**sort lexicographically:**
	^ B A N A N A $	A N A N A $ ^ B
	$ ^ B A N A N A	A N A $ ^ B A N
	A $ ^ B A N A N	A $ ^ B A N A N
^ B A N A N A $	N A $ ^ B A N A	B A N A N A $ ^
	A N A $ ^ B A N	N A N A $ ^ B A
	N A N A $ ^ B A	N A $ ^ B A N A
	A N A N A $ ^ B	^ B A N A N A $
	B A N A N A $ ^	$ ^ B A N A N A

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:	**generate all rotations:**	**sort lexicographically:**	**BWT is the last column:**
	^ B A N A N A $	A N A N A $ ^ B	
	$ ^ B A N A N A	A N A $ ^ B A N	
	A $ ^ B A N A N	A $ ^ B A N A N	
^ B A N A N A $	N A $ ^ B A N A	B A N A N A $ ^	B N N ^ A A $ A
	A N A $ ^ B A N	N A N A $ ^ B A	
	N A N A $ ^ B A	N A $ ^ B A N A	
	A N A N A $ ^ B	^ B A N A N A $	
	B A N A N A $ ^	$ ^ B A N A N A	

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:	**generate all rotations:**	**sort lexicographically:**	**BWT is the last column:**
	^ B A N A N A $	A N A N A $ ^ B	
	$ ^ B A N A N A	A N A $ ^ B A N	
	A $ ^ B A N A N	A $ ^ B A N A N	
^ B A N A N A $	N A $ ^ B A N A	B A N A N A $ ^	B N N ^ A A $ A
	A N A $ ^ B A N	N A N A $ ^ B A	
	N A N A $ ^ B A	N A $ ^ B A N A	
	A N A N A $ ^ B	^ B A N A N A $	
	B A N A N A $ ^	$ ^ B A N A N A	

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)

use text with start+end:

```
^ B A N A N A $
```

generate all rotations:

```
^ B A N A N A $
$ ^ B A N A N A
A $ ^ B A N A N
N A $ ^ B A N A
A N A $ ^ B A N
N A N A $ ^ B A
A N A N A $ ^ B
B A N A N A $ ^
```

sort lexicographically:

```
A N A N A $ ^ B
A N A $ ^ B A N
A $ ^ B A N A N
B A N A N A $ ^
N A N A $ ^ B A
N A $ ^ B A N A
^ B A N A N A $
$ ^ B A N A N A
```

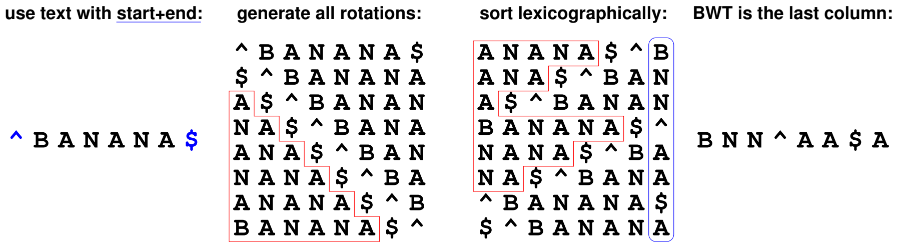
BWT is the last column:

```
B N N ^ A A $ A
```

Burrows-Wheeler-Transform (BWT) - encoding

Methods using the Burrows-Wheeler-Transform (BWT) for mapping generate the BWT from the text, e.g. the genome, adding an start (ˆ) and end character ($):

(Note, in the example we assume that ˆ and $ sort after the letters.)



```
use text with start+end:   generate all rotations:   sort lexicographically:   BWT is the last column:

                           ^ B A N A N A $           A N A N A $ ^ B
                           $ ^ B A N A N A           A N A $ ^ B A N
                           A $ ^ B A N A N           A $ ^ B A N A N
^ B A N A N A $            N A $ ^ B A N A           B A N A N A $ ^           B N N ^ A A $ A
                           A N A $ ^ B A N           N A N A $ ^ B A
                           N A N A $ ^ B A           N A $ ^ B A N A
                           A N A N A $ ^ B           ^ B A N A N A $
                           B A N A N A $ ^           $ ^ B A N A N A
```

Originally, the Burrows-Wheeler-Transform (BWT) has been introduced in the field of data compression, because (a) the BTW compresses better than the original text and (b) one can decode the original text from the BWT.

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW

```
B
N
N
^
A
A
$
A
```

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	**sort**
B	A
N	A
N	A
^	B
A	N
A	N
$	^
A	$

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column
B	A	A . . . B
N	A	A . . . N
N	A	A . . . N
^	B	B . . . ^
A	N	N . . . A
A	N	N . . . A
$	^	^ . . . $
A	$	$ . . . A

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	**sort**	**we know 1st +last column**	**rotate (BTW front)**
B	A	A . . . B	B A . . .
N	A	A . . . N	N A . . .
N	A	A . . . N	N A . . .
^	B	B . . . ^	^ B . . .
A	N	N . . . A	A N . . .
A	N	N . . . A	A N . . .
$	^	^ . . . $	$ ^ . . .
A	$	$ . . . A	A $ . . .

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .
N	A	A . . . N	N A . . .	A N . . .
N	A	A . . . N	N A . . .	A $ . . .
^	B	B . . . ^	^ B . . .	B A . . .
A	N	N . . . A	A N . . .	N A . . .
A	N	N . . . A	A N . . .	N A . . .
$	^	^ . . . $	$ ^ . . .	^ B . . .
A	$	$ . . . A	A $ . . .	$ ^ . . .

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)
B	A	A . . . B	B A . . .	A N . . .	B A N . . .
N	A	A . . . N	N A . . .	A N . . .	N A N . . .
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .
A	N	N . . . A	A N . . .	N A . . .	A N A . . .
A	N	N . . . A	A N . . .	N A . . .	A N A . . .
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort
A N A N
A N A $
A $ ^ B
B A N A
N A N A
N A $ ^
^ B A N
$ ^ B A

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort	add BTW and sort
A N A N	A N A N A
A N A $	A N A $ ^
A $ ^ B	A $ ^ B A
B A N A	B A N A N
N A N A	N A N A $
N A $ ^	N A $ ^ B
^ B A N	^ B A N A
$ ^ B A	$ ^ B A N

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort	add BTW and sort	add BTW and sort
A N A N	A N A N A	A N A N A $
A N A $	A N A $ ^	A N A $ ^ B
A $ ^ B	A $ ^ B A	A $ ^ B A N
B A N A	B A N A N	B A N A N A
N A N A	N A N A $	N A N A $ ^
N A $ ^	N A $ ^ B	N A $ ^ B A
^ B A N	^ B A N A	^ B A N A N
$ ^ B A	$ ^ B A N	$ ^ B A N A

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort	add BTW and sort	add BTW and sort	add BTW and sort
A N A N	A N A N A	A N A N A $	A N A N A $ ^
A N A $	A N A $ ^	A N A $ ^ B	A N A $ ^ B A
A $ ^ B	A $ ^ B A	A $ ^ B A N	A $ ^ B A N A
B A N A	B A N A N	B A N A N A	B A N A N A $
N A N A	N A N A $	N A N A $ ^	N A N A $ ^ B
N A $ ^	N A $ ^ B	N A $ ^ B A	N A $ ^ B A N
^ B A N	^ B A N A	^ B A N A N	^ B A N A N A
$ ^ B A	$ ^ B A N	$ ^ B A N A	$ ^ B A N A N

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort	add BTW and sort	add BTW and sort	add BTW and sort	...until the matrix has its width again
A N A N	A N A N A	A N A N A $	A N A N A $ ^	A N A N A $ ^ B
A N A $	A N A $ ^	A N A $ ^ B	A N A $ ^ B A	A N A $ ^ B A N
A $ ^ B	A $ ^ B A	A $ ^ B A N	A $ ^ B A N A	A $ ^ B A N A N
B A N A	B A N A N	B A N A N A	B A N A N A $	B A N A N A $ ^
N A N A	N A N A $	N A N A $ ^	N A N A $ ^ B	N A N A $ ^ B A
N A $ ^	N A $ ^ B	N A $ ^ B A	N A $ ^ B A N	N A $ ^ B A N A
^ B A N	^ B A N A	^ B A N A N	^ B A N A N A	^ B A N A N A $
$ ^ B A	$ ^ B A N	$ ^ B A N A	$ ^ B A N A N	$ ^ B A N A N A

Burrows-Wheeler-Transform (BWT) - decoding

From the BTW the original text can easily decoded:

start from the BTW	sort	we know 1st +last column	rotate (BTW front)	sort	rotate (BTW front)	sort
B	A	A . . . B	B A . . .	A N . . .	B A N . . .	A N A
N	A	A . . . N	N A . . .	A N . . .	N A N . . .	A N A
N	A	A . . . N	N A . . .	A $ . . .	N A $ . . .	A $ ^
^	B	B . . . ^	^ B . . .	B A . . .	^ B A . . .	B A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A N
A	N	N . . . A	A N . . .	N A . . .	A N A . . .	N A $
$	^	^ . . . $	$ ^ . . .	^ B . . .	$ ^ B . . .	^ B A
A	$	$ . . . A	A $ . . .	$ ^ . . .	A $ ^ . . .	$ ^ B

add BTW and sort	add BTW and sort	add BTW and sort	add BTW and sort	...until the matrix has its width again
A N A N	A N A N A	A N A N A $	A N A N A $ ^	A N A N A $ ^ B
A N A $	A N A $ ^	A N A $ ^ B	A N A $ ^ B A	A N A $ ^ B A N
A $ ^ B	A $ ^ B A	A $ ^ B A N	A $ ^ B A N A	A $ ^ B A N A N
B A N A	B A N A N	B A N A N A	B A N A N A $	B A N A N A $ ^
N A N A	N A N A $	N A N A $ ^	N A N A $ ^ B	N A N A $ ^ B A
N A $ ^	N A $ ^ B	N A $ ^ B A	N A $ ^ B A N	N A $ ^ B A N A
^ B A N	^ B A N A	^ B A N A N	^ B A N A N A	^ B A N A N A $
$ ^ B A	$ ^ B A N	$ ^ B A N A	$ ^ B A N A N	$ ^ B A N A N A

The original text can be found in the line starting with ^ and ending with $.

Burrows-Wheeler-Transform (BWT) - some observations

$\texttt{\textasciicircum}^1\ \texttt{B}^2\texttt{A}^3\ \texttt{N}^4\texttt{A}^5\ \texttt{N}^6\texttt{A}^7\ \texttt{\$}^8$

Burrows-Wheeler-Transform (BWT) - some observations

^^{1} B^{2} A^{3} N^{4} A^{5} N^{6} A^{7} \$^{8}

^{3}A	N	A	N	A	\$	^	B^{2}
^{5}A	N	A	\$	^	B	A	N^{4}
^{7}A	\$	^	B	A	N	A	N^{6}
^{2}B	A	N	A	N	A	\$	^^{1}
^{4}N	A	N	A	\$	^	B	A^{3}
^{6}N	A	\$	^	B	A	N	A^{5}
^{1}^	B	A	N	A	N	A	\$^{8}
^{8}\$	^	B	A	N	A	N	A^{7}

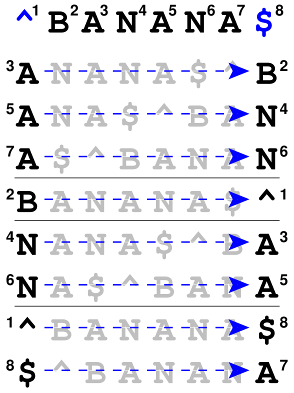
Burrows-Wheeler-Transform (BWT) - some observations

^^{1} B^{2} A^{3} N^{4} A^{5} N^{6} A^{7} \$^{8}

^{3}**A**	N	A	N	A	$	^	**B**^{2}
^{5}**A**	N	A	$	^	B	A	**N**^{4}
^{7}**A**	$	^	B	A	N	A	**N**^{6}
^{2}**B**	A	N	A	N	A	$	**^**^{1}
^{4}**N**	A	N	A	$	^	B	**A**^{3}
^{6}**N**	A	$	^	B	A	N	**A**^{5}
^{1}**^**	B	A	N	A	N	A	**$**^{8}
^{8}**$**	^	B	A	N	A	N	**A**^{7}

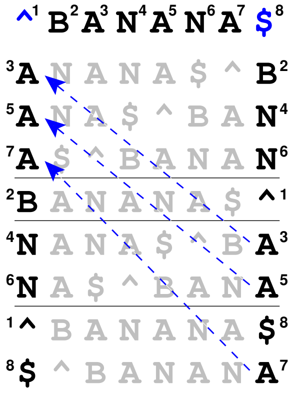
- a letter in the 1st column is easy to find, because they are sorted

Burrows-Wheeler-Transform (BWT) - some observations



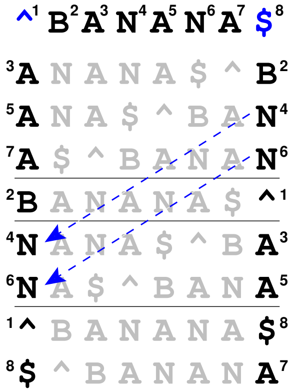
- a letter in the 1st column is easy to find, because they are sorted
- the letter in the last column preceeds the one in the 1st column (thus, searching for words starts at the last letter)

Burrows-Wheeler-Transform (BWT) - some observations



- a letter in the 1st column is easy to find, because they are sorted
- the letter in the last column preceeds the one in the 1st column (thus, searching for words starts at the last letter)
- the order of occurrence of a single letter in the last and the 1st column is the same (the 2nd A in the one is the 2nd A in the other)

Burrows-Wheeler-Transform (BWT) - some observations



- a letter in the 1st column is easy to find, because they are sorted
- the letter in the last column preceeds the one in the 1st column (thus, searching for words starts at the last letter)
- the order of occurrence of a single letter in the last and the 1st column is the same (the 2nd A in the one is the 2nd A in the other)

Burrows-Wheeler-Transform (BWT) - searching

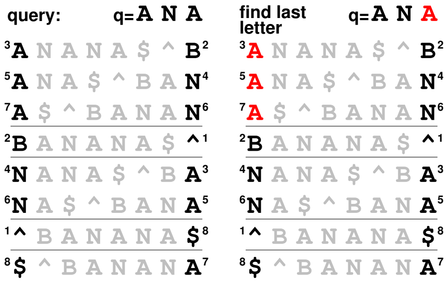
Searching from the last letter to the first of the search string (q=ANA):

query: q=**A N A**

3	**A**	N	A	N	A	$	^	**B**	2
5	**A**	N	A	$	^	B	A	**N**	4
7	**A**	$	^	B	A	N	A	**N**	6
2	**B**	A	N	A	N	A	$	**^**	1
4	**N**	A	N	A	$	^	B	**A**	3
6	**N**	A	$	^	B	A	N	**A**	5
1	**^**	B	A	N	A	N	A	**$**	8
8	**$**	^	B	A	N	A	N	**A**	7

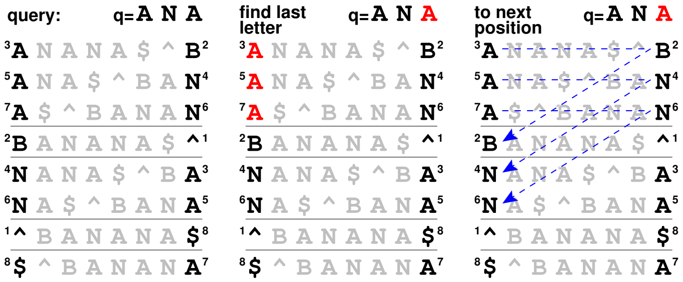
Burrows-Wheeler-Transform (BWT) - searching

Searching from the last letter to the first of the search string (q=ANA):



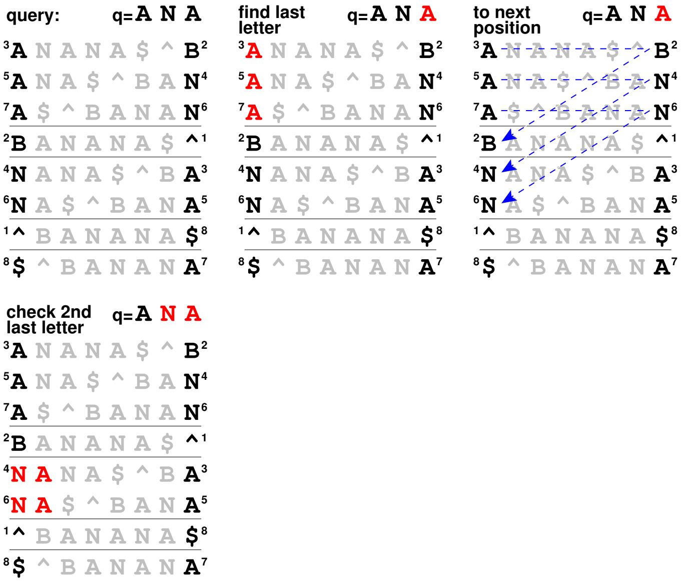
Burrows-Wheeler-Transform (BWT) - searching

Searching from the last letter to the first of the search string (q=ANA):



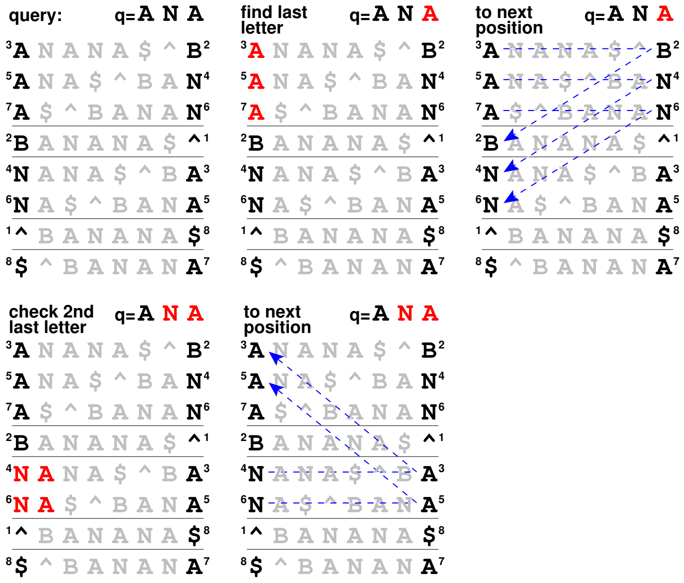
Burrows-Wheeler-Transform (BWT) - searching

Searching from the last letter to the first of the search string (q=ANA):



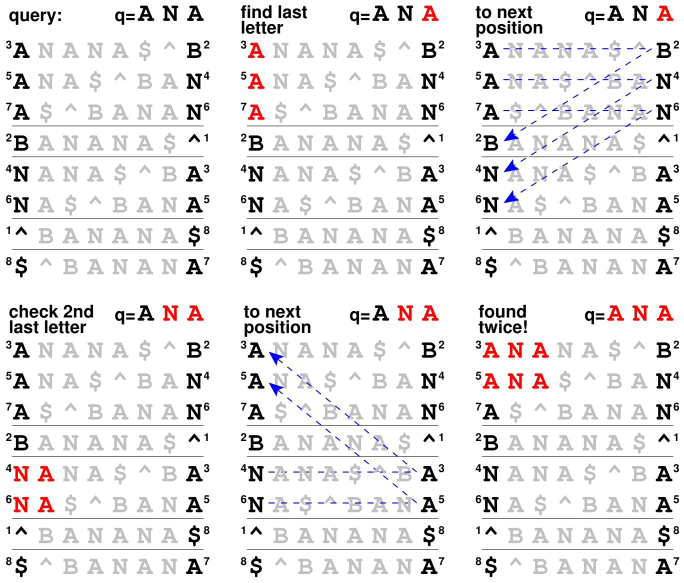
Burrows-Wheeler-Transform (BWT) - searching

Searching from the last letter to the first of the search string (q=ANA):



Burrows-Wheeler-Transform (BWT) - searching

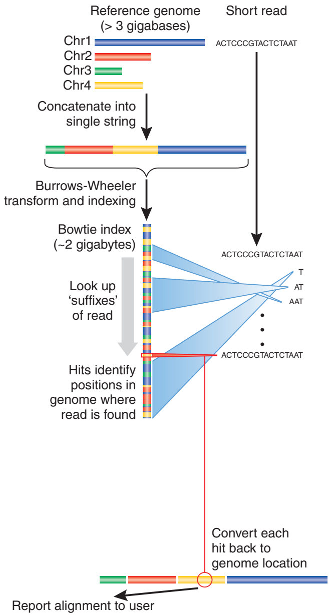
Searching from the last letter to the first of the search string (q=ANA):



Burrows-Wheeler-Transform (BWT) - approximate matches

- this way exact matches can be found easily
- to find approximate matches,
- everytime a mismatch is detected,
- a backtrace is done, introducing changes at any position
- at the beginning only one change and later more if still no match is found

BWT-based mapping


Reference genome
(> 3 gigabases)
Short read
Chr1
Chr2
Chr3
Chr4
ACTCCCGTACTCTAAT
Concatenate into single string
Burrows-Wheeler transform and indexing
Bowtie index (~2 gigabytes)
Look up 'suffixes' of read
ACTCCCGTACTCTAAT
T
AT
AAT
ACTCCCGTACTCTAAT
Hits identify positions in genome where read is found
Convert each hit back to genome location
Report alignment to user

Source: Trapnell+Salzberg (2009)

- BWT-based is harder to implement than seed based approaches
- however, it is less memory intense (only (about 1-2GB for the human genome of 3 Gbp) and much faster
- on the other hand, seed based approaches have been shown to be much more sensitive, and thus able to match more reads correctly