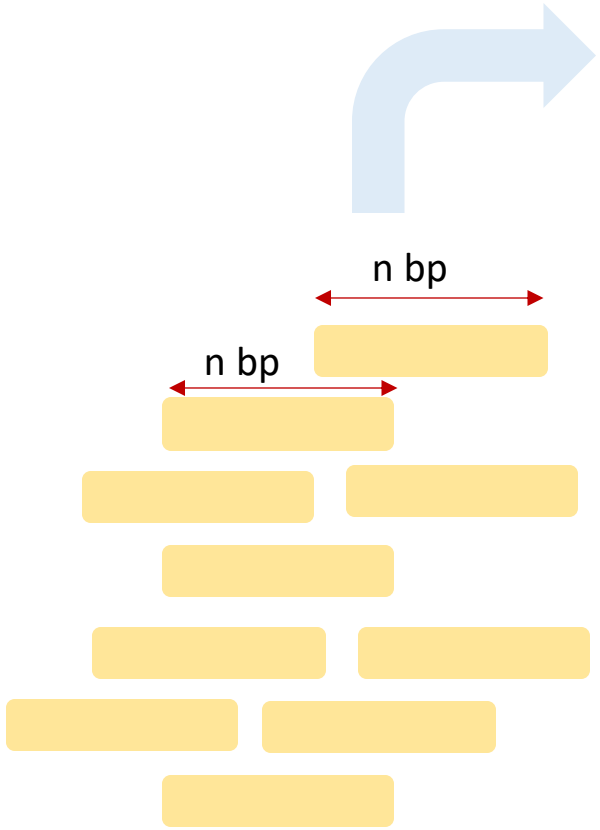
Genome Assembly

Reconstruct the sequence by combining the reads

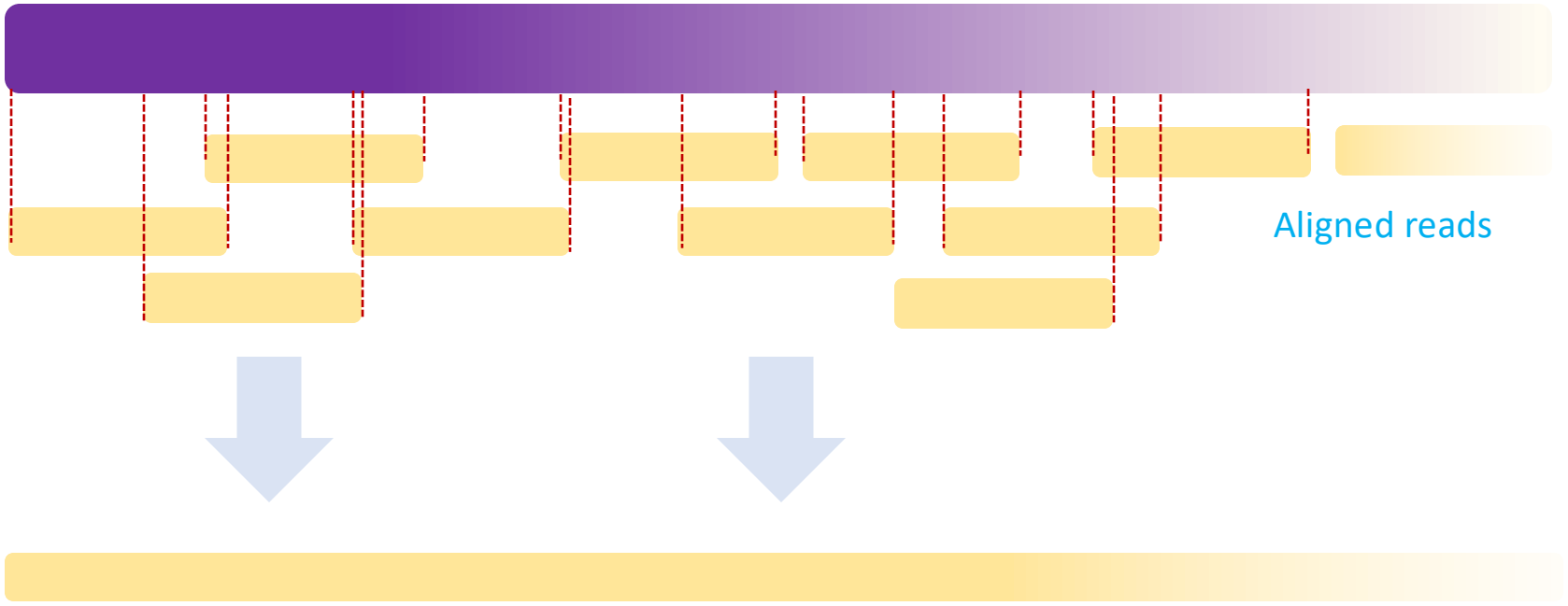
Assembly?

matching to reference!



short sequencing reads
(many of them)

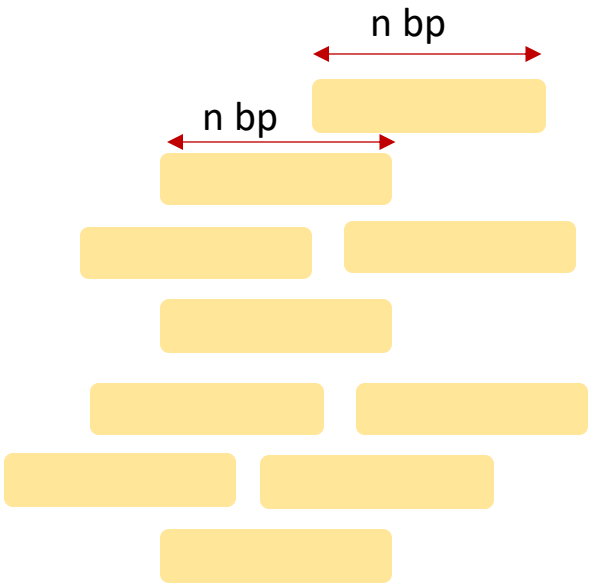
reference sequence (very long!)



Assembled genome based on
the reference

But what if there is no reference???

Assembly (*de novo assembly*)



short sequencing reads
(many of them)

reconstruct

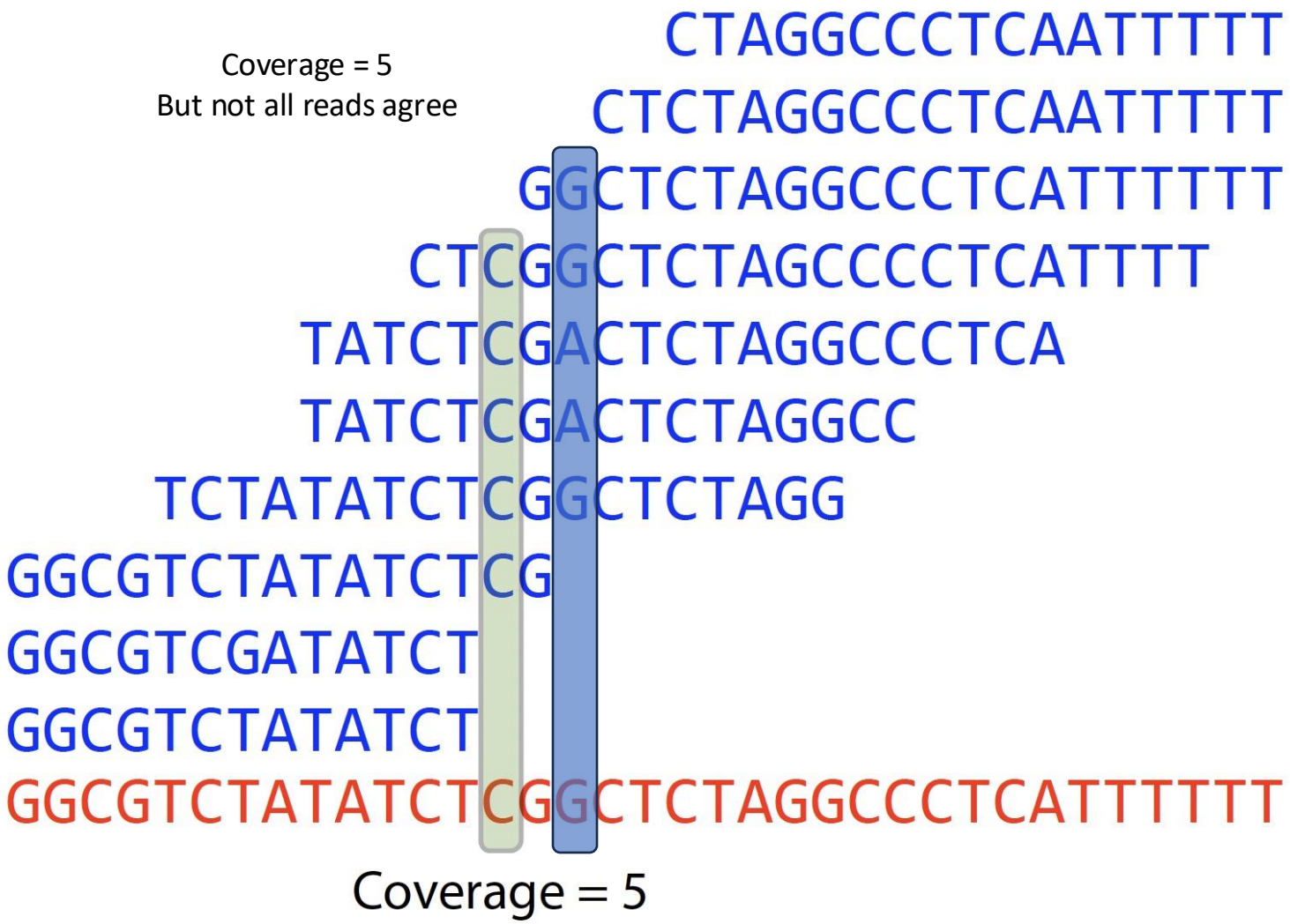
No reference genome to refer

How??

Assembly

source: John Hopkins University

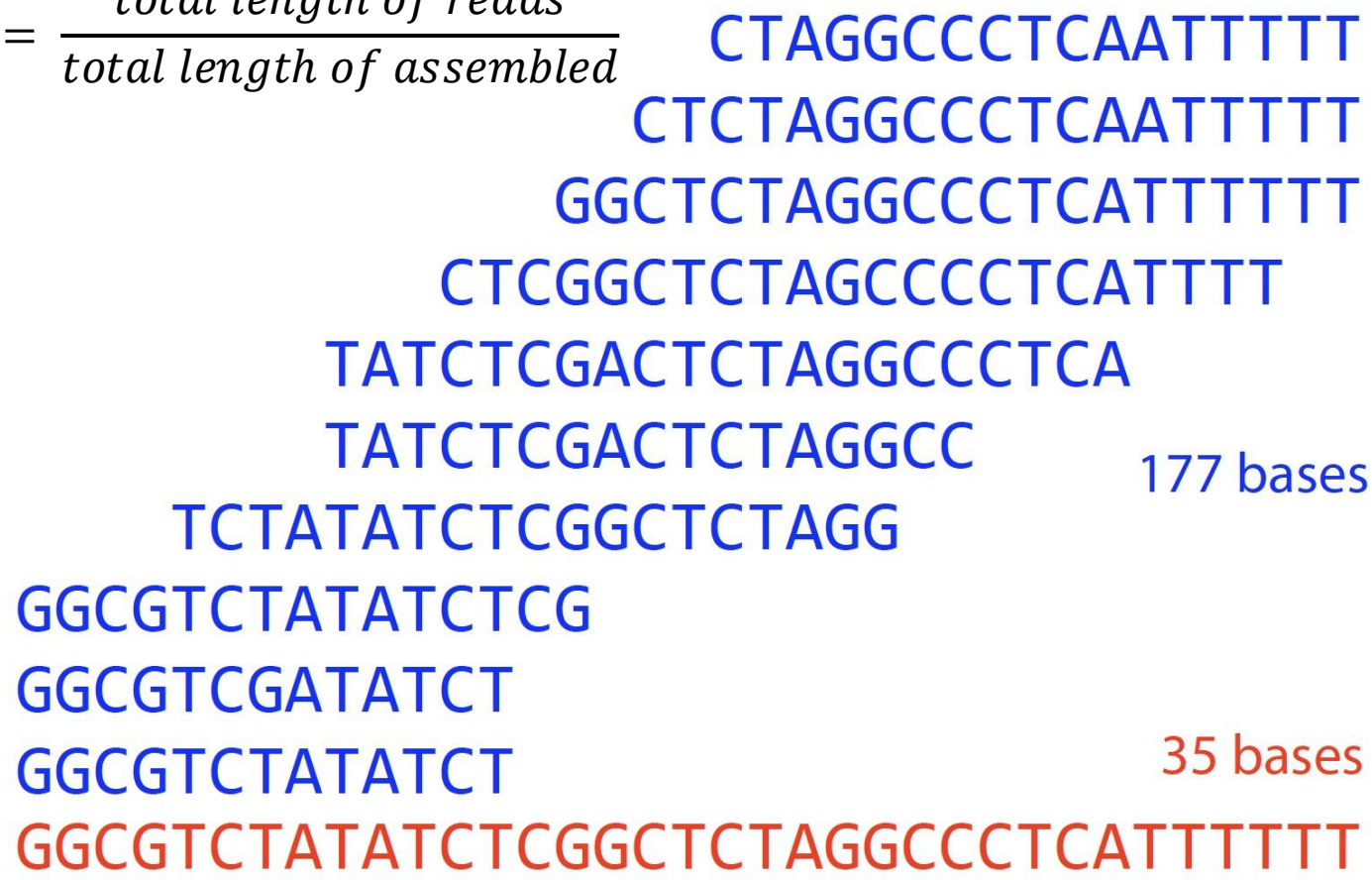
Coverage



source: John Hopkins University

Average coverage

$$Average\ coverage = \frac{total\ length\ of\ reads}{total\ length\ of\ assembled}$$



Average coverage = 177 / 35 $\approx 5\ fold$

source: John Hopkins University

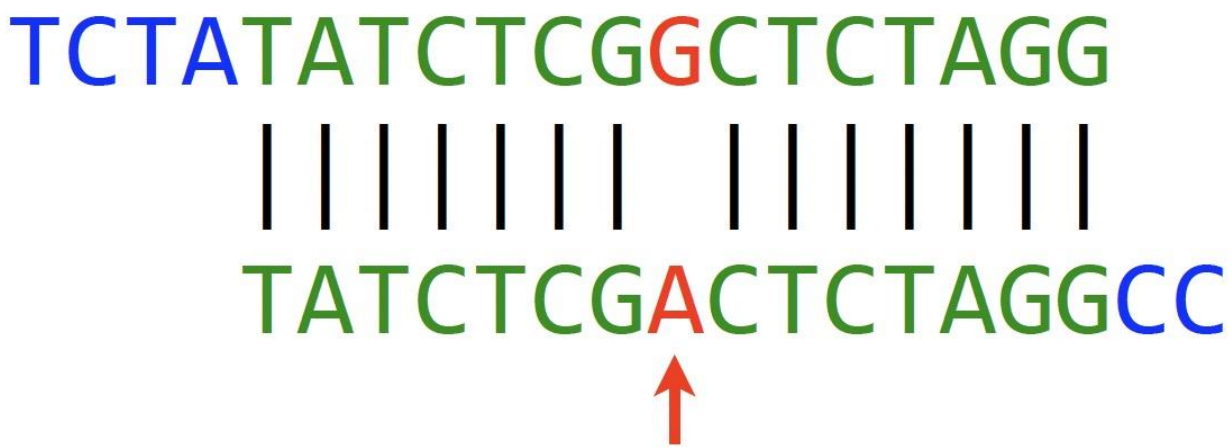
Principles of assembly

- First law of assembly:
  - If a suffix of read A is similar to a prefix of read B…

```
A:  TCTATATCTCGGCTCTAGG
        ||||||| |||||||
B:      TATCTCGACTCTAGGCC
```

…then A and B might overlap in the genome

```
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
   TATCTCGACTCTAGGCC
```



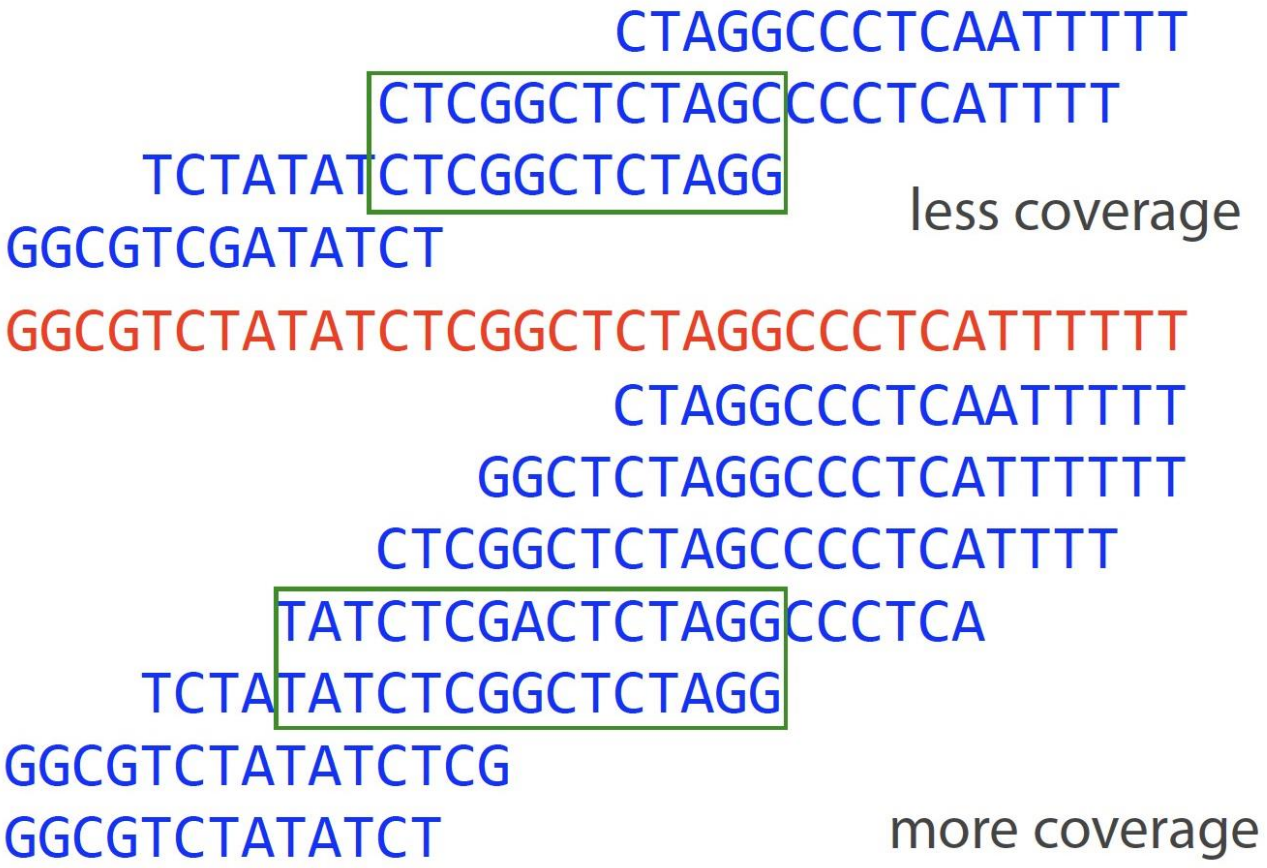
Why the differences?

1. Sequencing errors
2. Polyploidy: e.g. humans have 2 copies of each chromosome, and copies can differ

source: John Hopkins University

Principles of assembly

- Second law of assembly
  - More coverage leads to more and longer overlaps

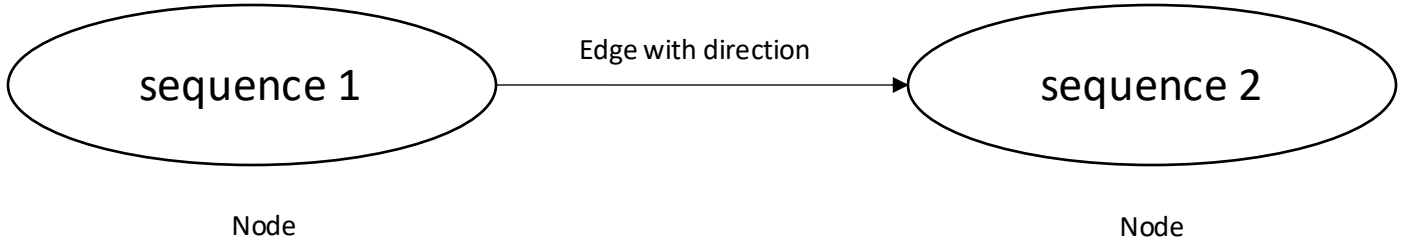


source: John Hopkins University

Representation of overlaps??

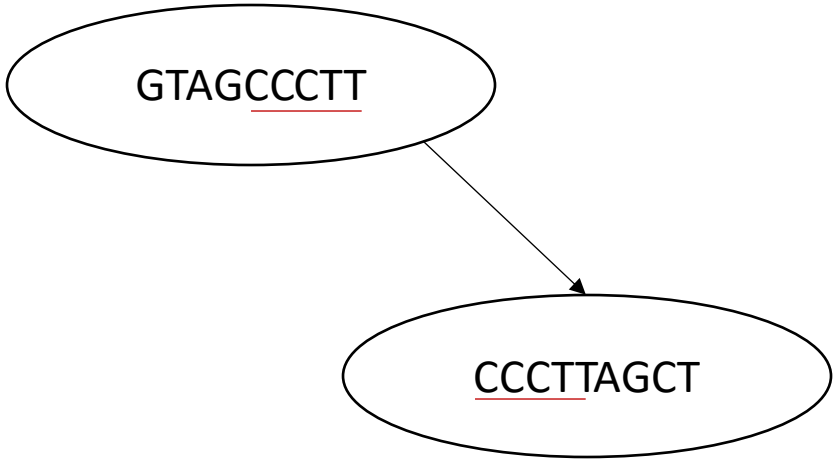
with directed graphs!

Directed graph

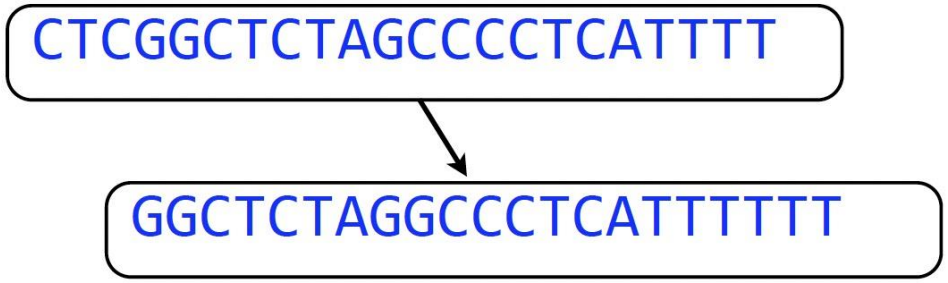


Each node is a read

CTCGGCTCTAGCCCCTCATTTT



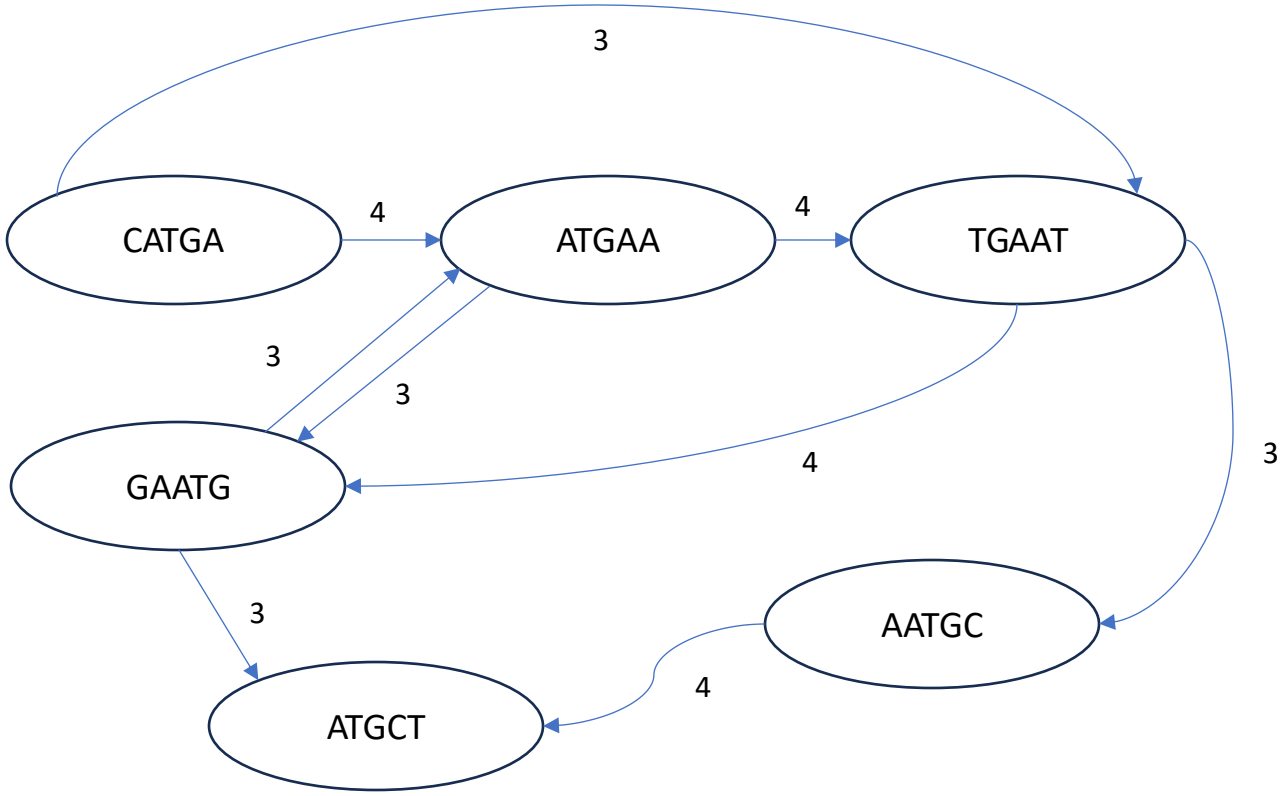
Draw edge A -> B when suffix of A overlaps prefix of B



Example

- Sequence: CATGAATGCT
- Overlap graph (5-mers, and min 3 overlaps)

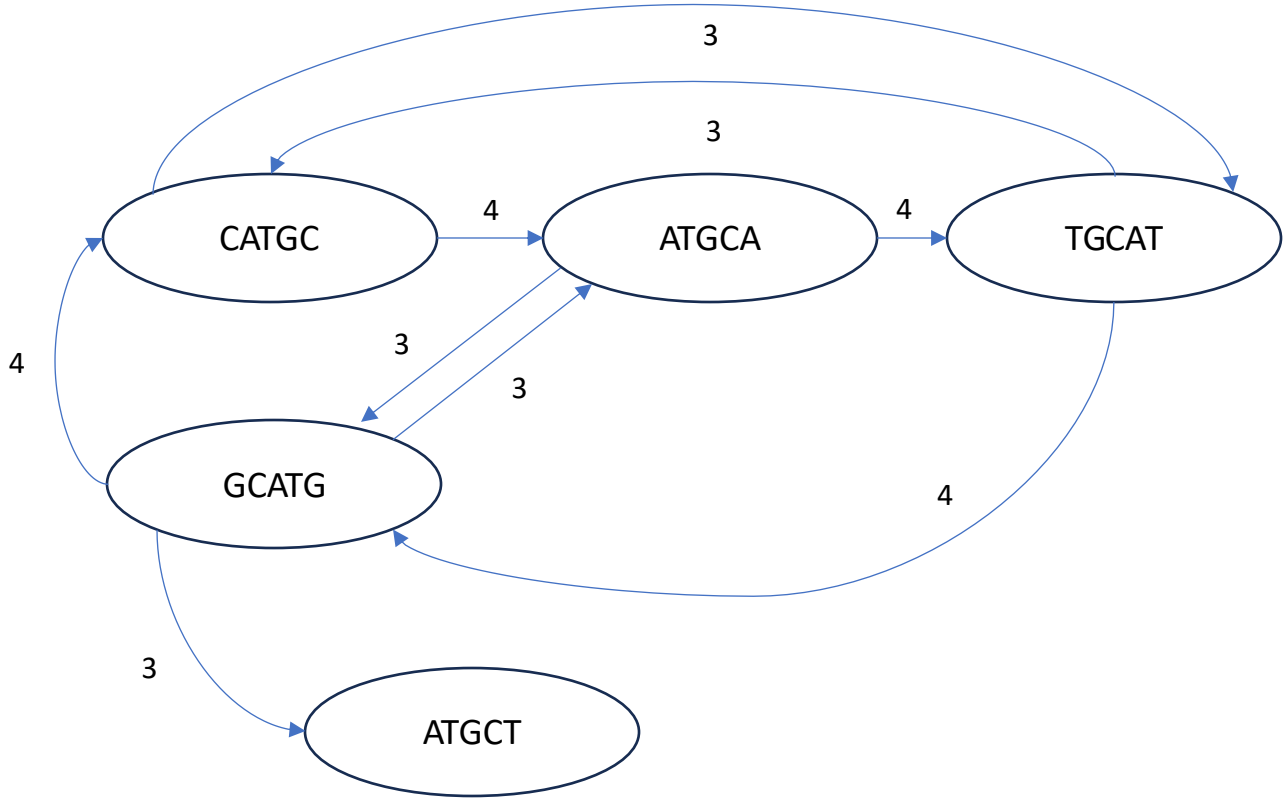
- 5-mers:
  - CATGA
  - ATGAA
  - TGAAT
  - GAATG
  - AATGC
  - ATGCT



Example

- Sequence: CATGCATGCT
- Overlap graph (5-mers, and min 3 overlaps)

- 5-mers:
  - CATGC
  - ATGCA
  - TGCAT
  - GCATG
  - CATGC
  - ATGCT



Let’s try

- Generate an overlap graph with the given reads, and edges with >=4 overlaps :

  GTACGT

  TACGTA

  ACGTAC

  CGTACG

  GTACGA

  TACGAT

Shortest common superstring

- Given a set of strings, *X,* find the **shortest common superstring (SCS)** from this set of strings.
- SCS(X) → shortest string containing the strings in *X* as substrings

Example:

X: BAA AAB BBA ABA ABB BBB AAA BAB

concatenation: BAAAABBBAABAABBBBBAAABAB (length: 24)

SCS(X): AAABBBABAA (length: 10)

How??

Getting shortest common superstring

- Idea: using different order for strings in X

order 1: AAA AAB ABA ABB BAA BAB BBA BBB

AAABABBAABABBABBB ⟵ superstring 1

order 2: AAA AAB ABA BAB ABB BBB BAA BBA

AAABABBBBAABBA ⟵ superstring 2

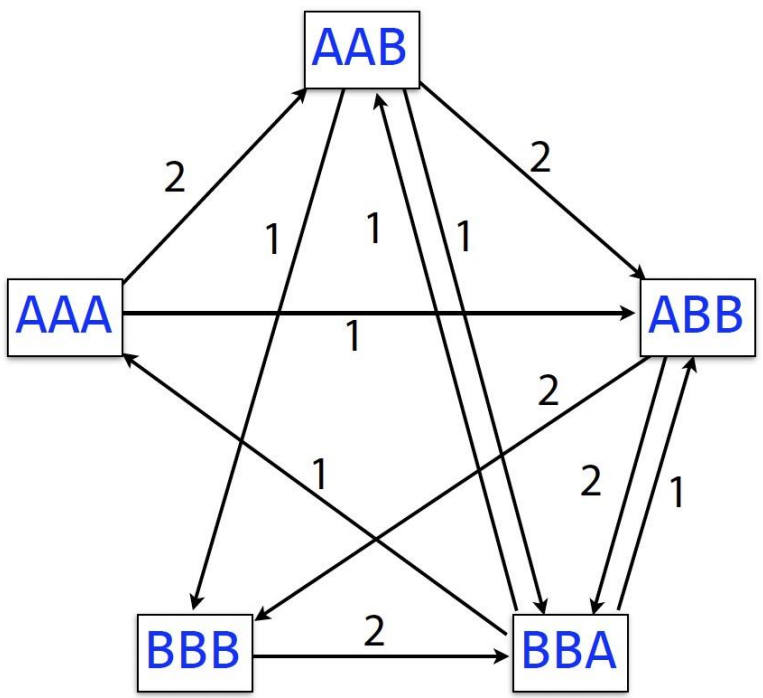
Try all possible orderings and pick shortest superstring

If x contains n strings, $n!$ (n factorial) orderings possible

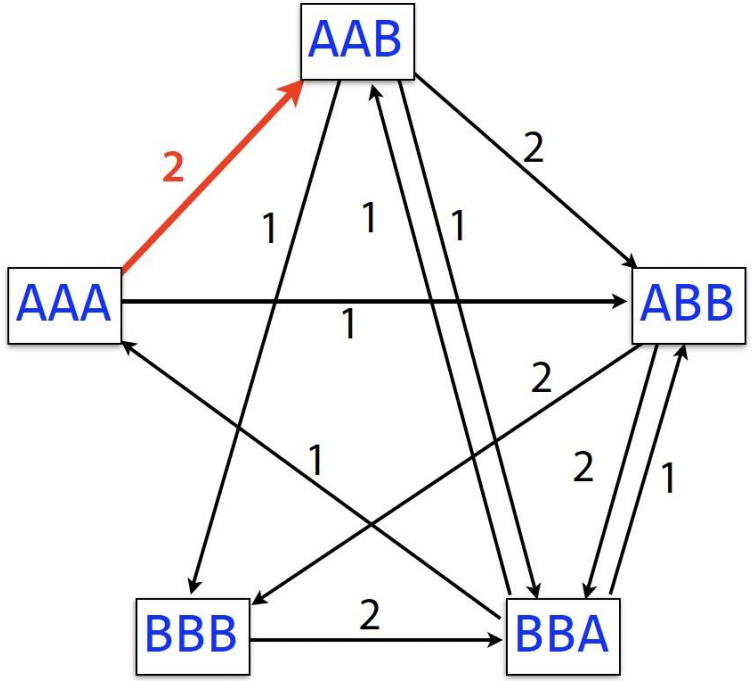
source: John Hopkins University

Greedy shortest common superstring

1

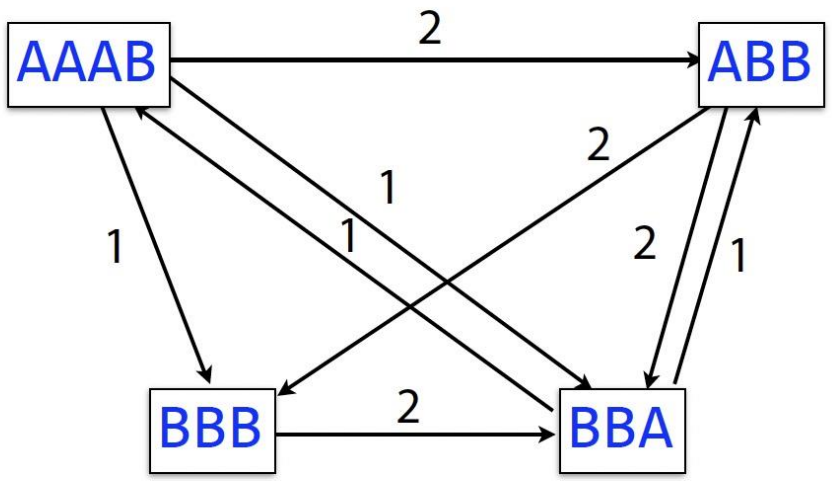


2

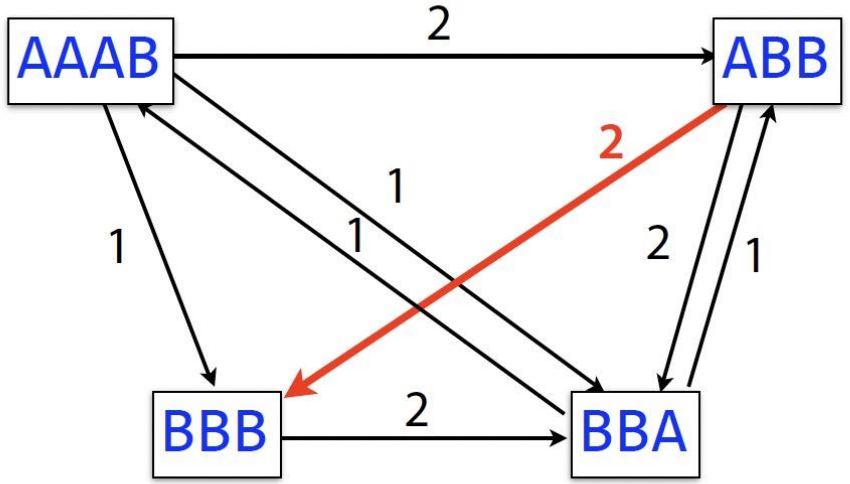


Greedy shortest common superstring

3

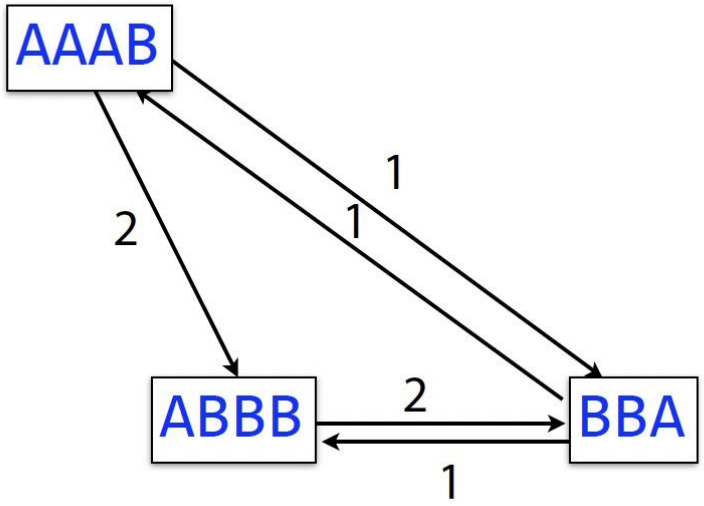


4

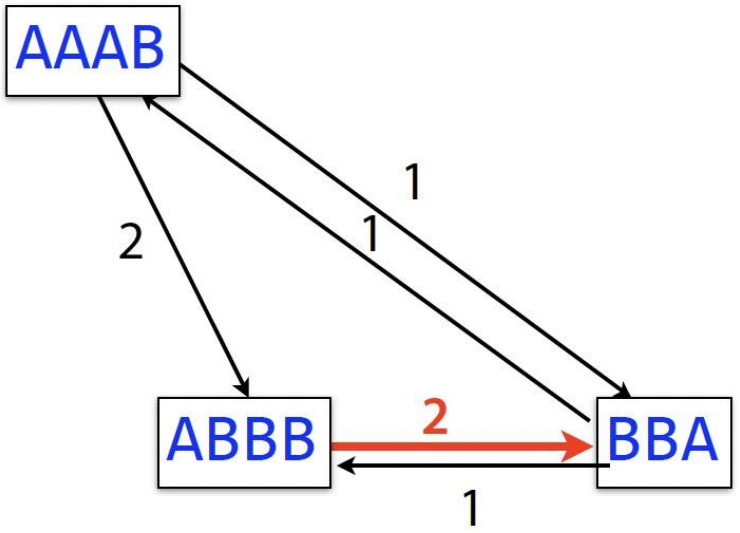


Greedy shortest common superstring

5

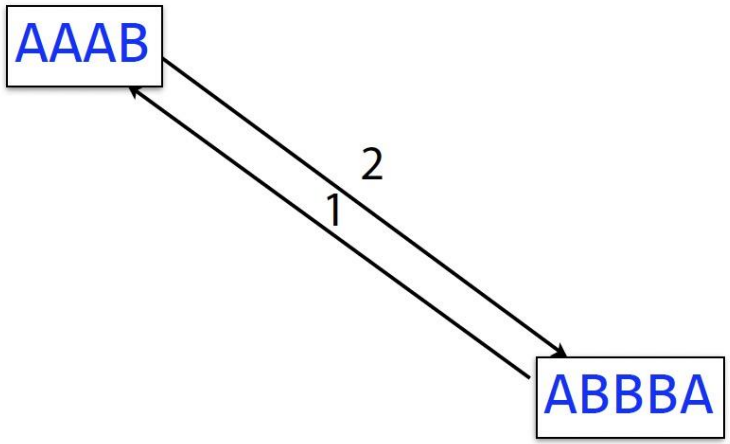


6

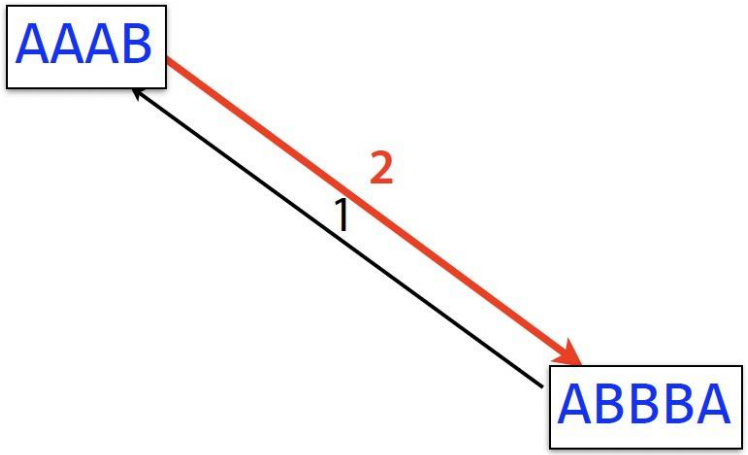


Greedy shortest common superstring

7



8



AAABBBA ⟵ superstring, length=7

Let’s try

- Given the sequence: GATCAACAACAT
- Task:
  - Extract the 3-mers from the sequence
  - From the 3-mers, construct the SCS using greedy approach.

Greedy SCS on 6-mers of a_long_long_long_time

```
ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long g_time ng_tim
ng_time ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long
ng_time g_long_ ng_lon a_long long_l ong_ti ong_lo long_t
ng_time long_ti g_long_ ng_lon a_long long_l ong_lo
ng_time ong_lon long_ti g_long_ a_long long_l
ong_lon long_time g_long_ a_long long_l
long_lon long_time g_long_ a_long
long_lon g_long_time a_long
long_long_time a_long
a_long_long_time
```

superstring

Repeated sequence could be collapsed during assembly

a_long_long_long_time

↓

a_long_long_time

source: John Hopkins University

Principles of assembly

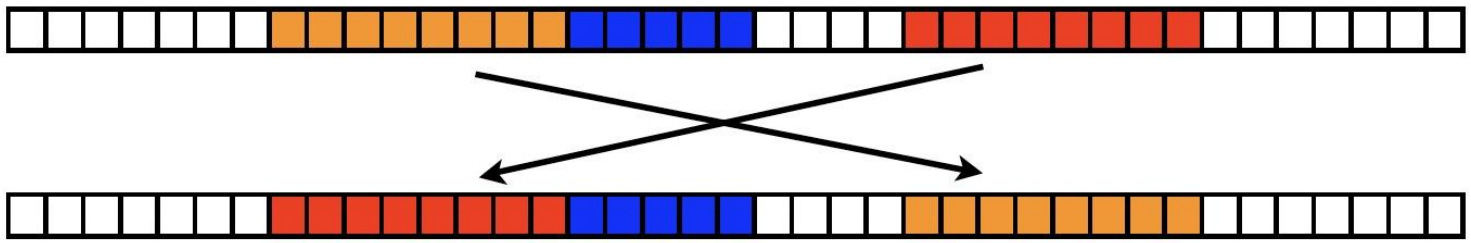
- Third law of assembly: repeats makes assembly difficult

Repeated sequence could be collapsed during assembly

a_long_long_long_time

↓

a_long_long_time

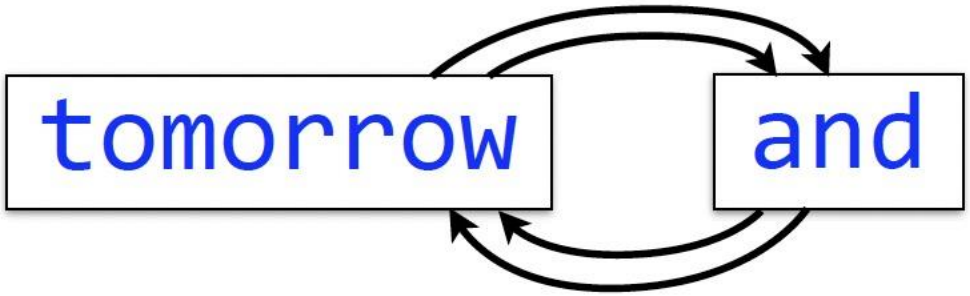


Possible switch of region in the genome that contains repetitive sequence

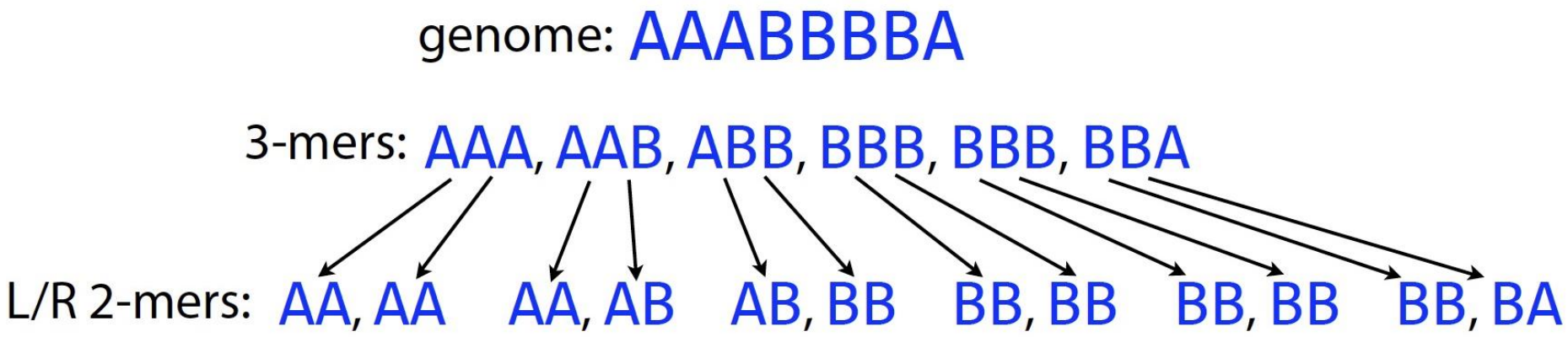
De Bruijn graph

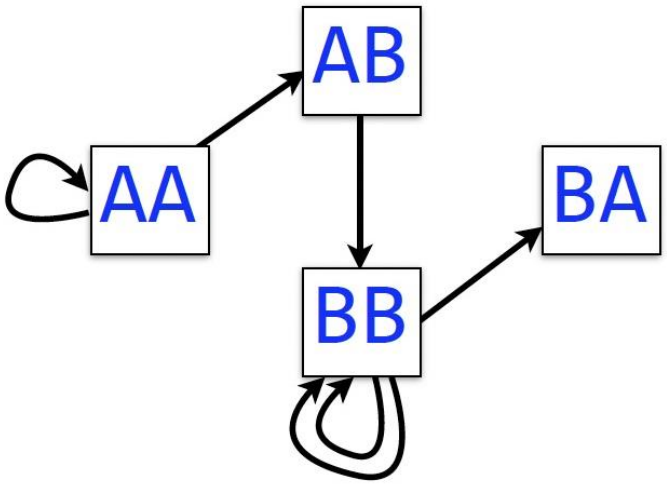
- A multi-graph that represent every unique words as node, and each existence of the word will be represented as the edge.

"tomorrow and tomorrow and tomorrow"



De Bruijn graph

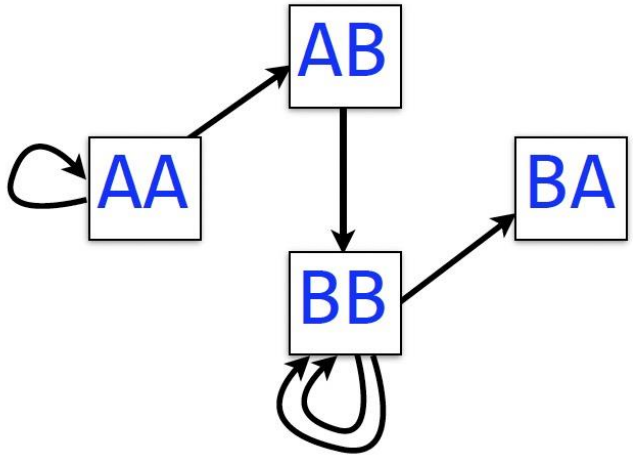




One edge per *k*-mer

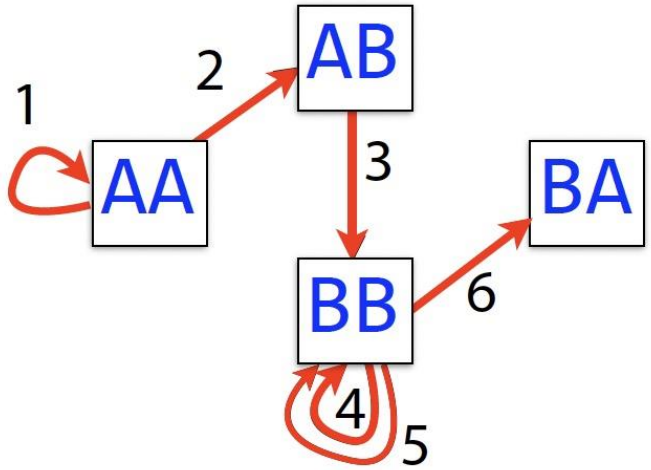
One node per distinct *k*-1-mer

How to interpret?



Walk crossing each edge exactly once gives a reconstruction of the genome

How to interpret?



AAABBBBA

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

Let’s try

- Generate the De Bruijn graph using the following 3-mer reads

  TCG,
  ATC,
  CGG
  GGC
  GGG
  GCC
  CAA
  CCA

- sequence: ATCGGGCCAA