Genômica Computacional

Montagem de genomas

Professor: Ricardo A. Vialle

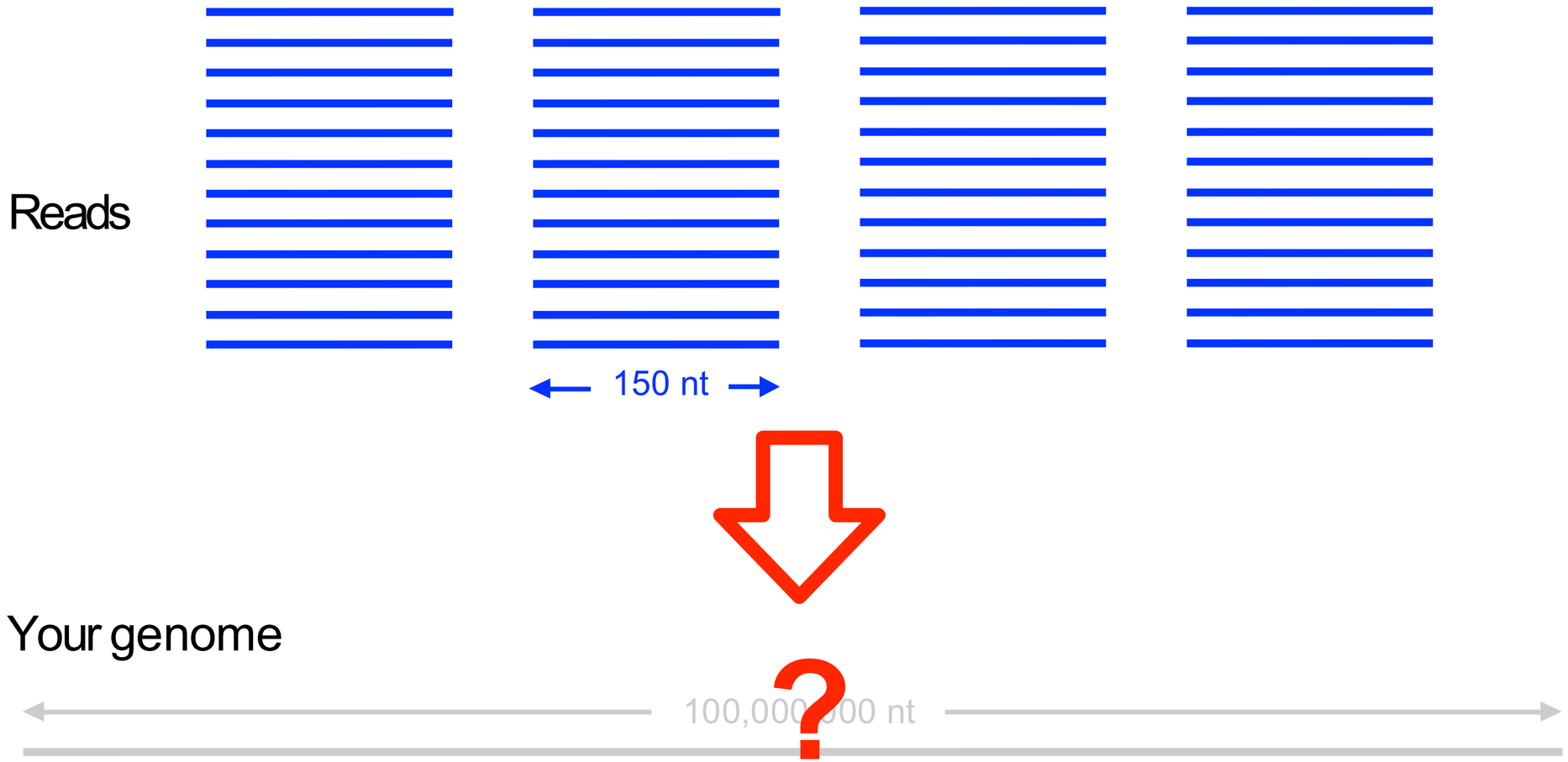
CS31 - Genômica Computacional [14/07-17/07 - 10h00-12h00]

15 de Julho de 2025

Cronograma

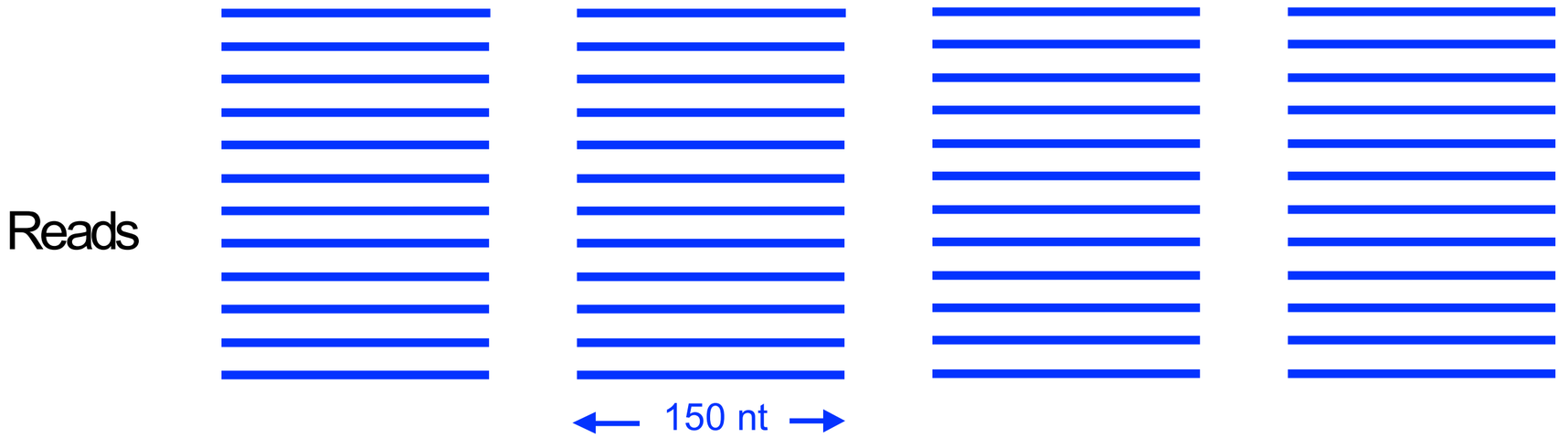
Data	Tema
~~14-Jul~~	~~Introdução e processamento de dados de sequenciamento (teórico-prática)~~
15-Jul	Montagem de genomas (teórico-prática)
16-Jul	Anotação de genomas (teórico-prática)
17-Jul	Analise de variabilidade genética (teórico-prática)

Genome sequenced, what next?



https://www.cs.jhu.edu/~langmea/resources/lecture_notes/01_genomics_comp_genomics_v2.pdf

Genome sequenced, what next?



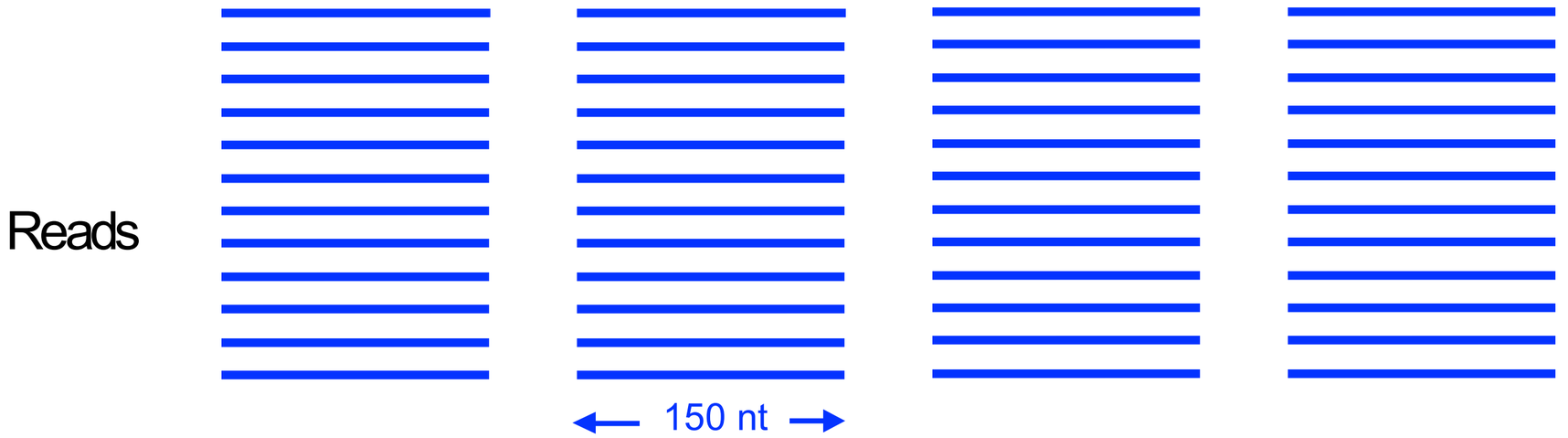
Your genome

Solutions:

1 – Align your reads to a genome of reference

2 – Assembly your genome from scratch (*de novo*)

Genome sequenced, what next?



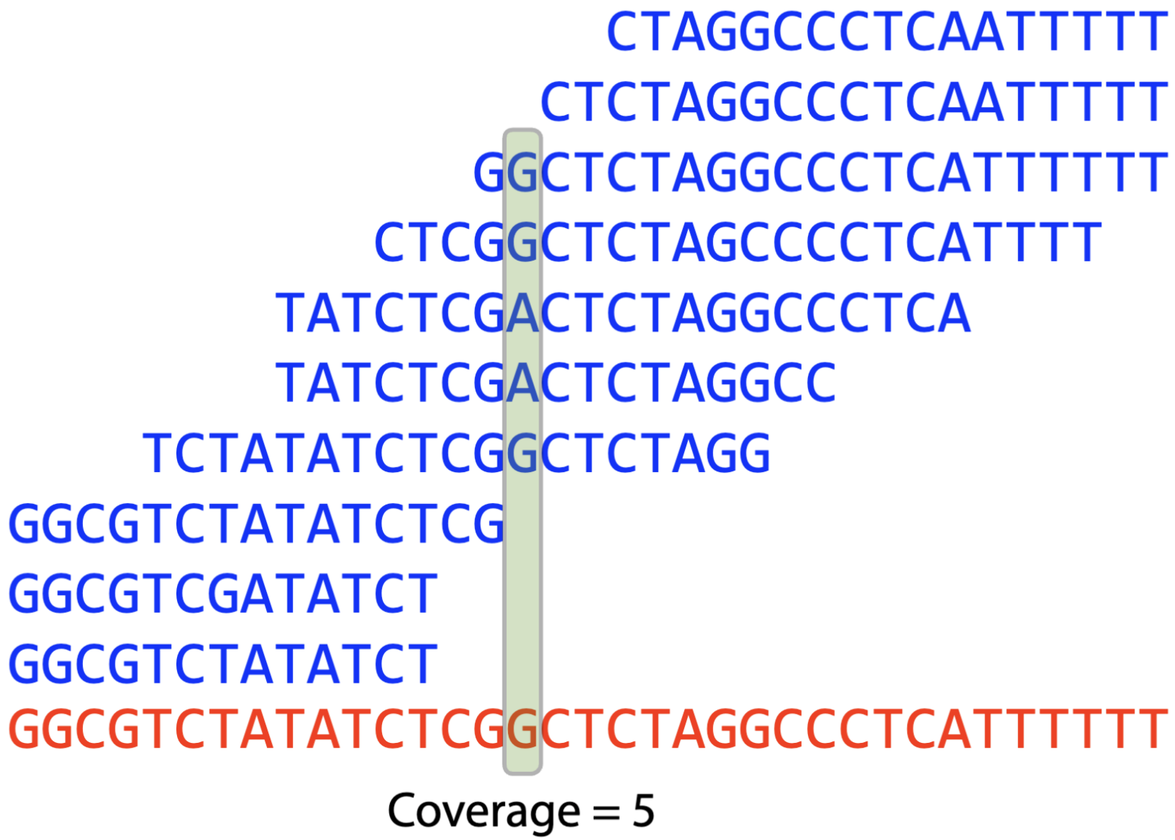
Your genome

Solutions:

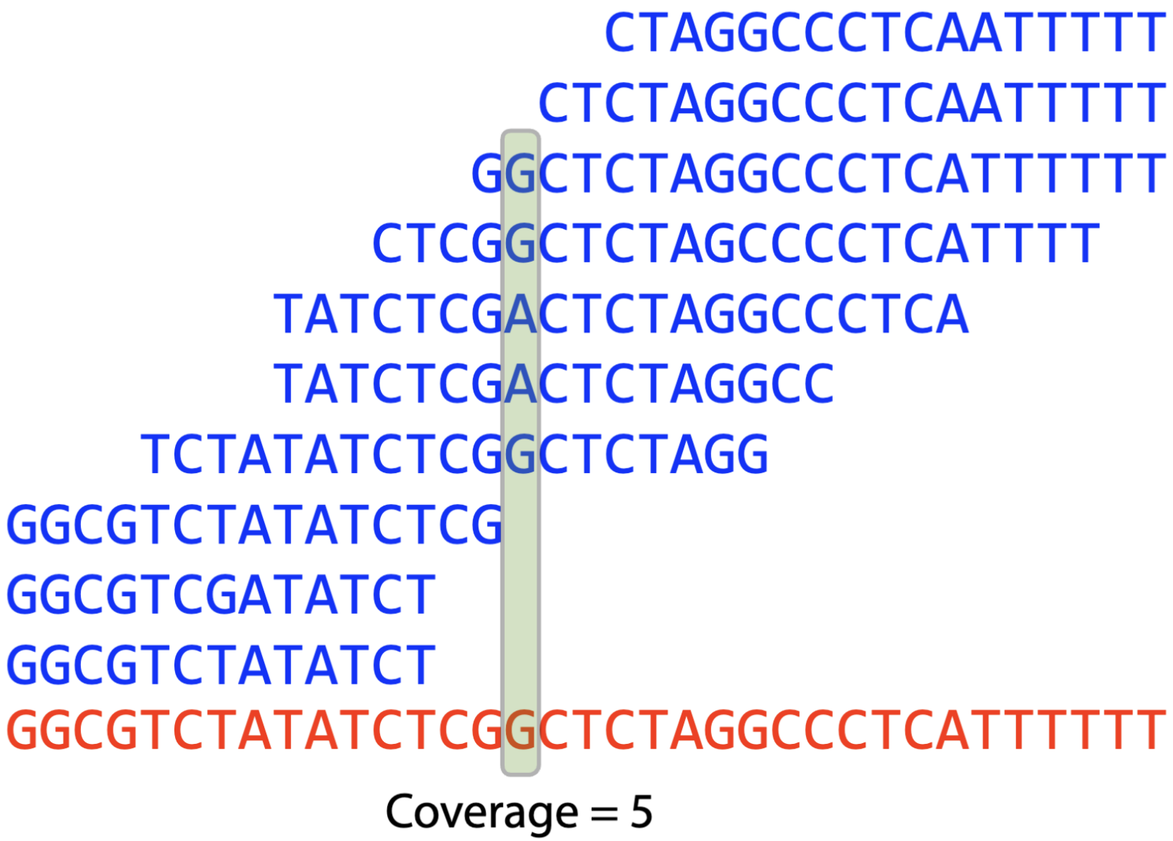
1 – Align your reads to a genome of reference

2 – Assembly your genome from scratch (*de novo*)

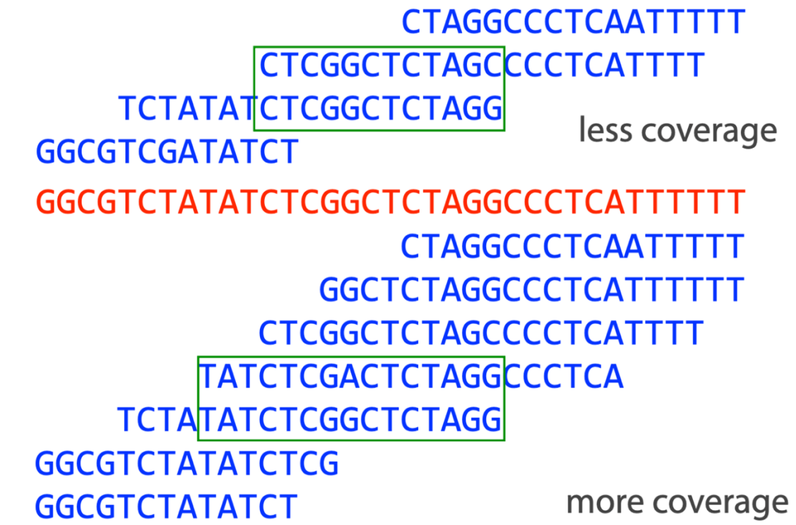
Concepts - Coverage



Coverage



More coverage leads to more and longer overlaps



Overlaps

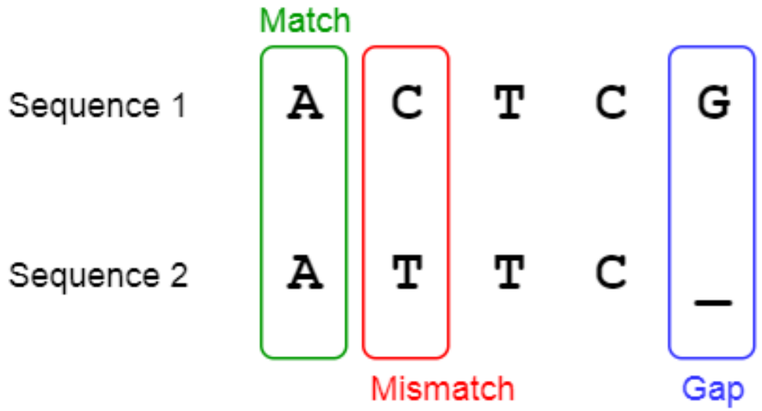
If a suffix of read A is similar to a prefix of read B...

```
TCTATATCTCGGCTCTAGG
      ||||||| |||||||
      TATCTCGACTCTAGGCC
```

...then A and B might *overlap* in the genome

```
      TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
      TATCTCGACTCTAGGCC
```

Sequence alignment



https://medium.com/analytics-vidhya/sequence-alignment-and-the-needleman-wunsch-algorithm-710c7b1a23a4

Sequence alignment

The Needleman-Wunsch Algorithm (1970)

Let’s align 2 sequences
CTCGCAGC and CATTCAC

STEP 1: Initialization of score matrix.

0	0	C	T	C	G	C	A	G	C
0									
C									
A									
T									
T									
C									
A									
C									

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Let’s assume:
Match= 10
Mismatch= -2
Gap Penalty= -5

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up								
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

SequenceA \ SequenceB	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

$$\text{Score}[i, j] = \max \begin{cases} \text{score}[i, j-1] + \text{gap penalty}, & \textbf{Left} \\ \text{score}[i-1, j] + \text{gap penalty}, & \textbf{Up} \\ \text{score}[i-1, j-1] + s(a_i, b_j) & \textbf{Diag} \end{cases}$$

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

SequenceA \ SequenceB	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB							
	0	0	C	T	C	G	C	A	G	C
SequenceA	0	0	-5	-10	-15	-20	-25	-30	-35	-40
	C	-5								
	A	-10								
	T	-15								
	T	-20								
	C	-25								
	A	-30								
	C	-35								

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row
j = 1 # column

```
SequenceA(i) = SequenceA(1) = C
SequenceB(j) = SequenceB(1) = C
```

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB							
	0	0	C	T	C	G	C	A	G	C
SequenceA	0	0	-5	-10	-15	-20	-25	-30	-35	-40
	C	-5								
	A	-10								
	T	-15								
	T	-20								
	C	-25								
	A	-30								
	C	-35								

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 1 # column `SequenceB(j) = SequenceB(1) = C`

Here since Seq1[i]=Seq2[j]: **Match= 10**

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB							
	0	0	C	T	C	G	C	A	G	C
SequenceA	0	0	-5	-10	-15	-20	-25	-30	-35	-40
	C	-5								
	A	-10								
	T	-15								
	T	-20								
	C	-25								
	A	-30								
	C	-35								

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 1 # column `SequenceB(j) = SequenceB(1) = C`

Here since Seq1[i]=Seq2[j]: **Match= 10**
Left = -5 + (-5) = -10

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

SequenceA \ SequenceB	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row
j = 1 # column

```
SequenceA(i) = SequenceA(1) = C
SequenceB(j) = SequenceB(1) = C
```

Here since Seq1[i]=Seq2[j]: **Match= 10**
Left = -5 + (-5) = -10 , Up = -5 + (-5) = -10

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB							
	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

SequenceA

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 1 # column `SequenceB(j) = SequenceB(1) = C`

Here since Seq1[i]=Seq2[j]: **Match= 10**

Left = -5 + (-5) = -10 , Up = -5 + (-5) = -10 , Diag = 0 + 10 = 10

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB							
	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

SequenceA

Score [i, j] = max

- score [i, j-1] + gap penalty, -5 — **Left**
- score [i-1, j] + gap penalty, -5 — **Up**
- score [i-1, j-1] + s(ai, bj) — **Diag**

10 or -2

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 1 # column `SequenceB(j) = SequenceB(1) = C`

Here since Seq1[i]=Seq2[j]: **Match= 10**
Left = -5 + (-5) = -10 , Up = -5 + (-5) = -10 , Diag = 0 + 10 = 10

Maximum of the 3 values obtained = 10.
Therefore, the value at **Score[i][j] = 10** and **Direction [i][j] = Diag**.

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Score Matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10							
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

- Direction matrix

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag							
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

The Needleman-Wunsch Algorithm

- Score Matrix:

SequenceA \ SequenceB	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10							
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 2 # column `SequenceB(j) = SequenceB(2) = T`

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

The Needleman-Wunsch Algorithm

- Score Matrix:

SequenceA \ SequenceB	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10							
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

i = 1 # row `SequenceA(i) = SequenceA(1) = C`

j = 2 # column `SequenceB(j) = SequenceB(2) = T`

Here since Seq1[i] is not equal to Seq2[j]:

Left = 10 + (**-5**) = 5, Up = -10 + (**-5**) = -15 , Diag = -5 + (**-2**) = -7

Maximum of the 3 values obtained = 10.
Therefore, the value at **Score[i][j] = 5** and **Direction [i][j] = Left**.

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Score Matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5						
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

- Direction matrix

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left						
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

Trace backing to find the sequence

start with the last cell of the table.

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- **Direction matrix:**

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

SequenceB

SequenceA	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- **Direction matrix:**

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

This is the path obtained.

Therefore, the reverse sequences obtained are:

SequenceA

C	-	A	C	T	-	T	A	C
C	G	A	C	G	C	T	-	C

SequenceB

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- **Direction matrix:**

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

This is the path obtained.
Therefore, the reverse sequences obtained are:

C	-	A	C	T	-	T	A	C
C	G	A	C	G	C	T	-	C

To obtain final results, we reverse the results.
Final sequences obtained are:

C	A	T	-	T	C	A	-	C
C	-	T	C	G	C	A	G	C

In the above example we considered diag when we get 2 direction in the table

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Needleman-Wunsch Algorithm

Complexity: O(mn). Since we are using 2 for loops to traverse the matrices.
Algorithm warnings:
NW dynamic programming algorithm guarantees us optimal global alignment with the parameter we supply (gap_penalty, match_award, mismatch_penalty)
However:
· Different parameters give different alignments.
· Correct alignment might not be the optimal one.
· Correct alignment might correspond only to part of global alignments.

https://medium.com/@nandiniumbarkar/needleman-wunsch-algorithm-7bba68b510db

Sequence alignment

The Smith-Waterman Algorithm (local alignment)

Initialize the scoring matrix

		T	G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0	0
G	0								
G	0								
T	0								
T	0								
G	0								
A	0								
C	0								
T	0								
A	0								

Substitution matrix: $S(a_i, b_j) = \begin{cases} +3, & a_i = b_j \\ -3, & a_i \neq b_j \end{cases}$

Gap penalty: $W_k = kW_1$
$W_1 = 2$

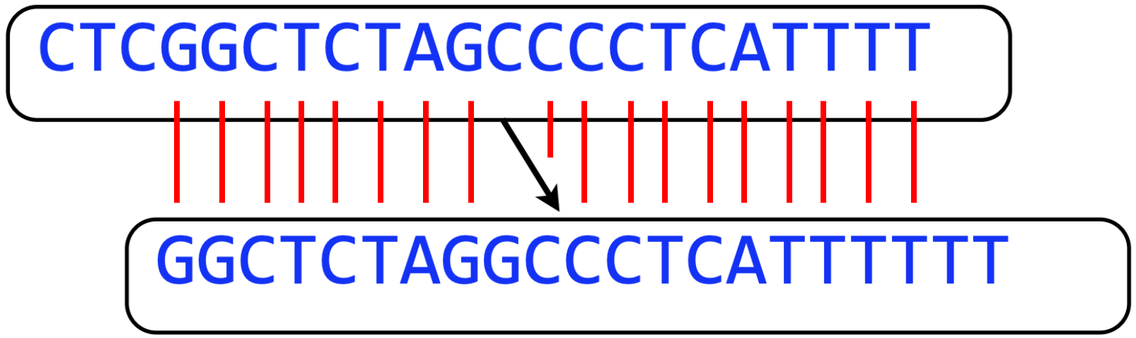
	Smith–Waterman algorithm	Needleman–Wunsch algorithm
Initialization	First row and first column are set to 0	First row and first column are subject to gap penalty
Scoring	Negative score is set to 0	Score can be negative
Traceback	Begin with the highest score, end when 0 is encountered	Begin with the cell at the lower right of the matrix, end at top left cell

https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Overlaps as graphs

Each node is a read

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



Overlaps graphs

"*k*-mer" is a substring of length *k*

S: GGCGATTCATCG

A 4-mer of *S*: ATTC

All 3-mers of *S*:

```
GGC
 GCG
  CGA
   GAT
    ATT
     TTC
      TCA
       CAT
        ATC
         TCG
```

mer: from Greek meaning "part"

Overlaps graphs

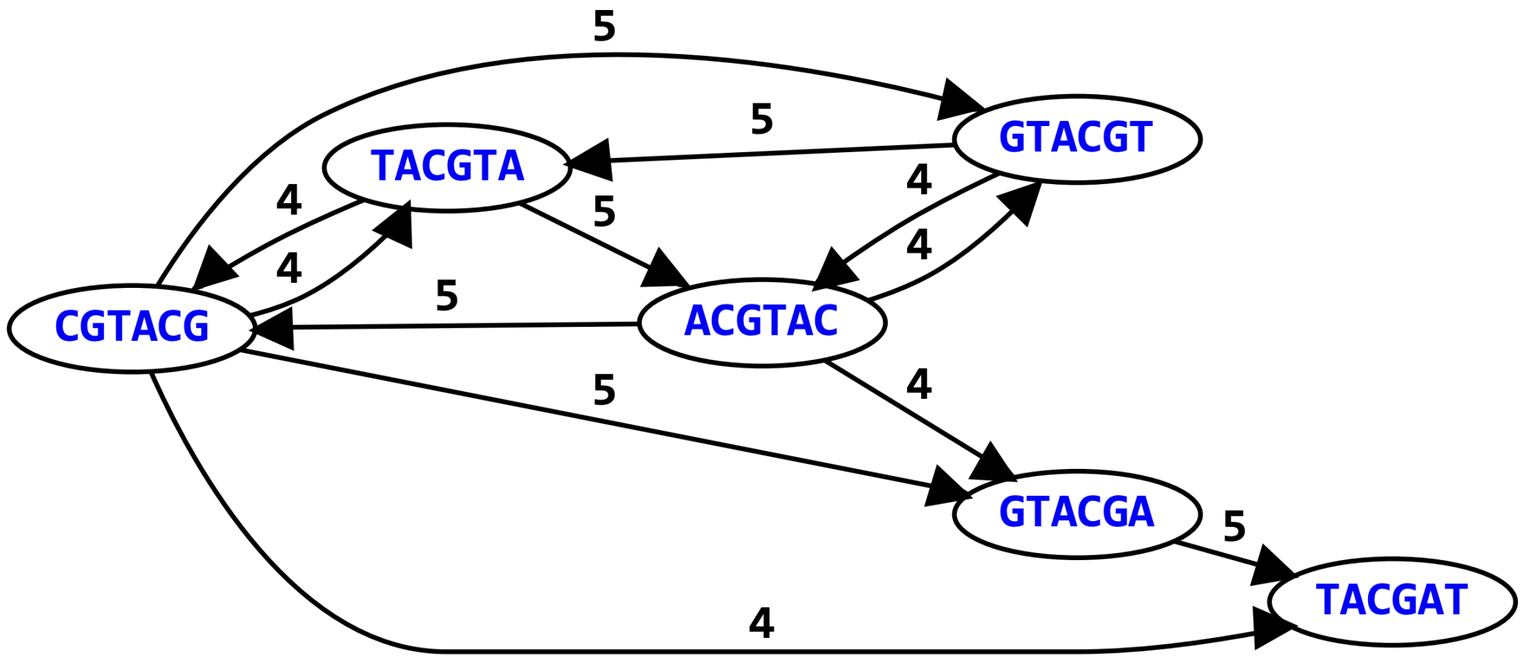
Nodes: all 6-mers from GTACGTACGAT

```
GTACGTACGA
GTACGT
 TACGTA
  ACGTAC
   CGTACG
    GTACGA
```

Overlaps graphs

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4

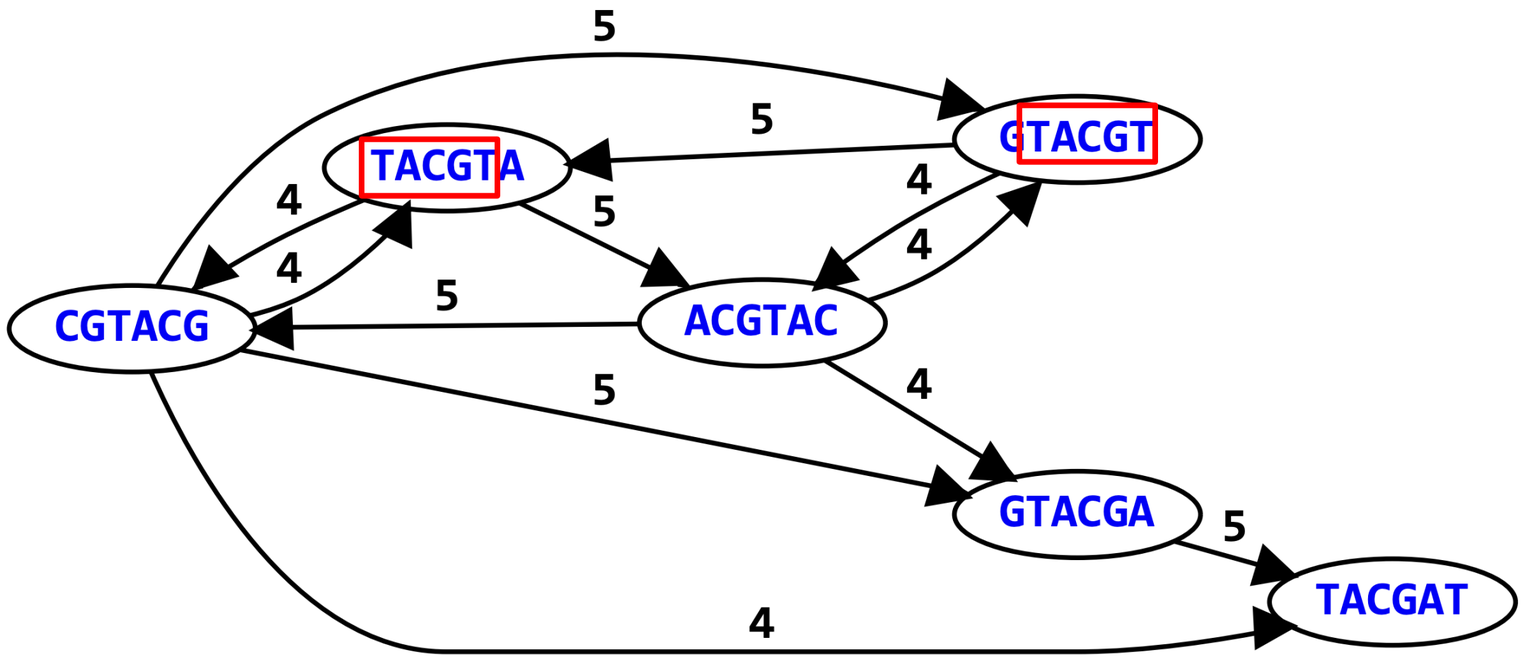


GTACGTACGA
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA

Overlaps graphs

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4

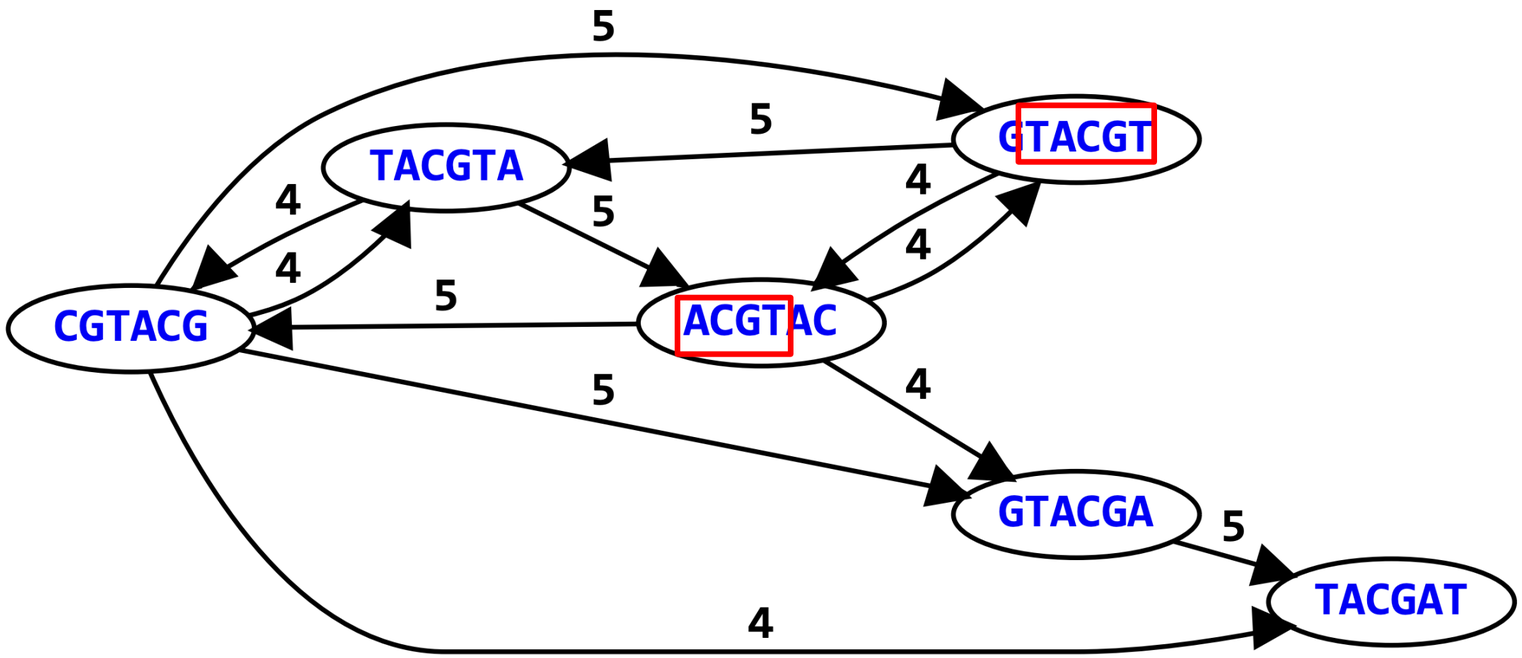


GTACGTACGA
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA

Overlaps graphs

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4

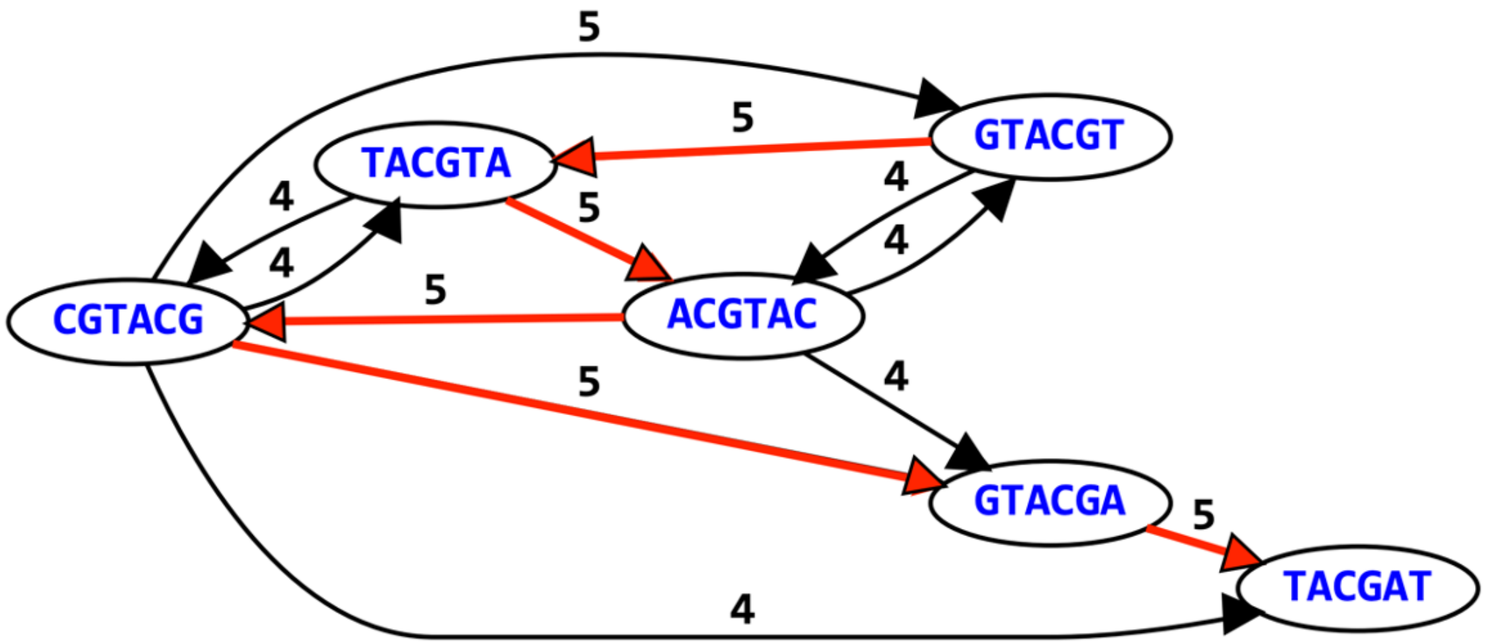


GTACGTACGA
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA

Overlaps graphs

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4



GTACGTACGA

GTACGT

TACGTA

ACGTAC

CGTACG

GTACGA

De Bruijn graph

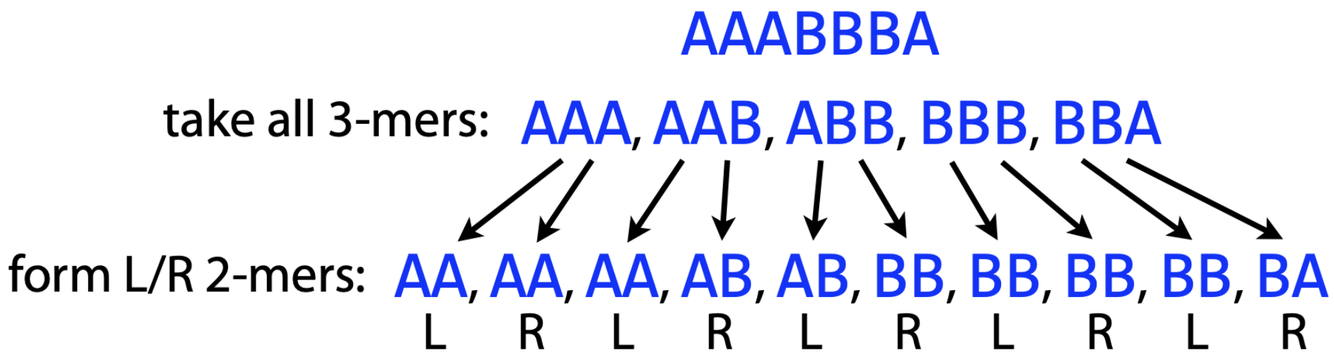
Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

AAABBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

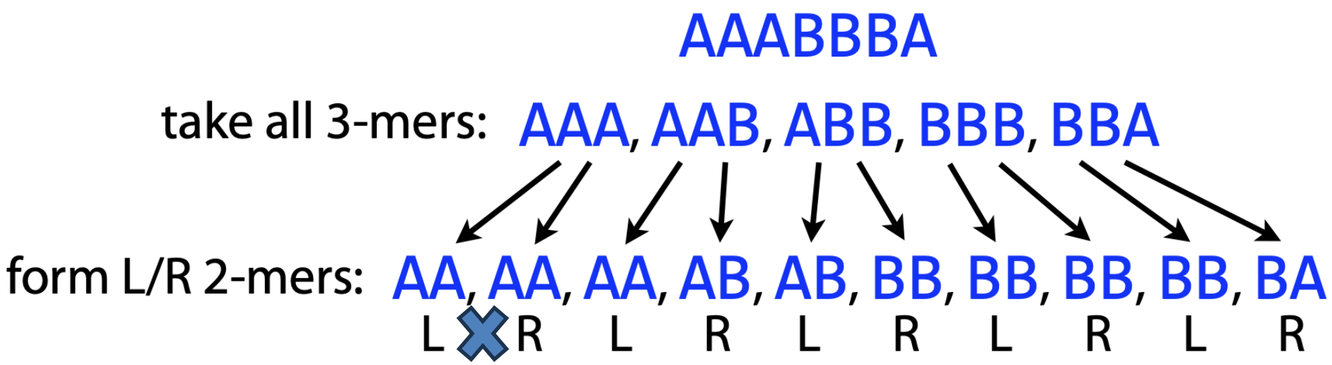
De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.



De Bruijn graph

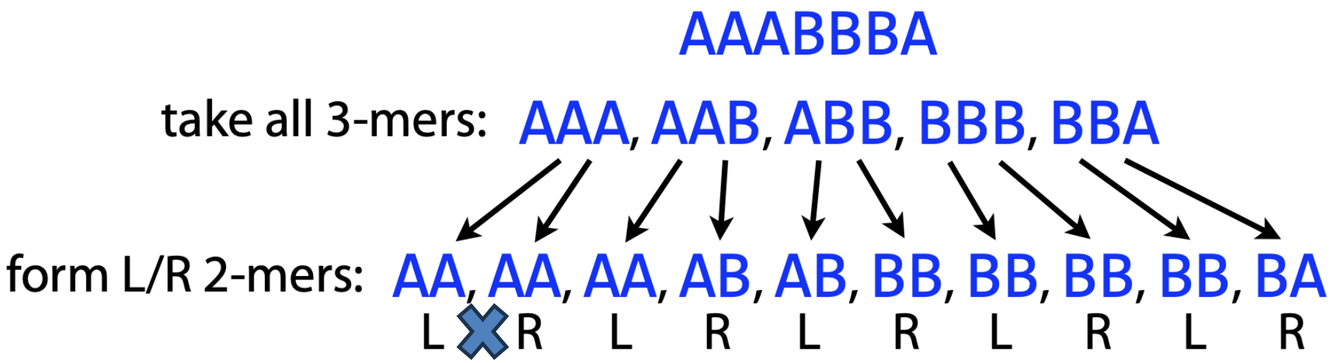
Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.



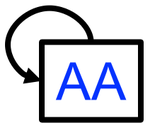
Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

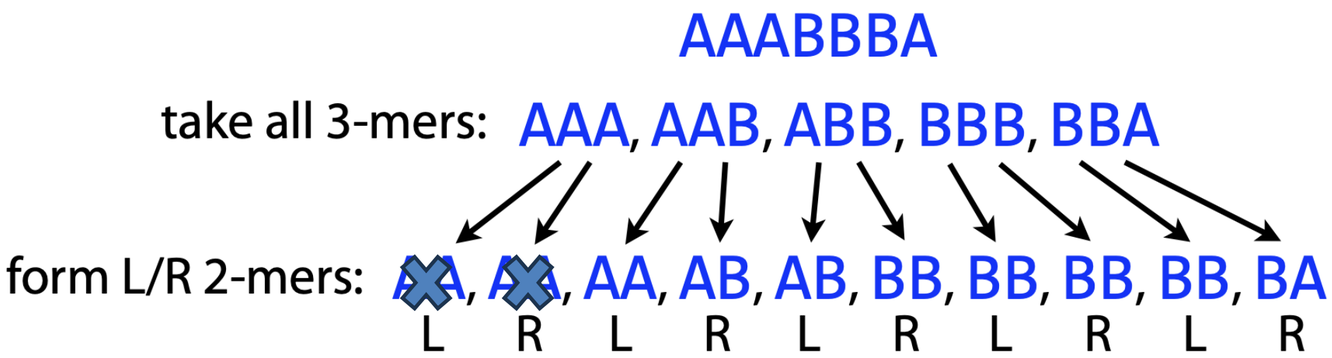


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

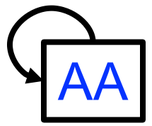


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

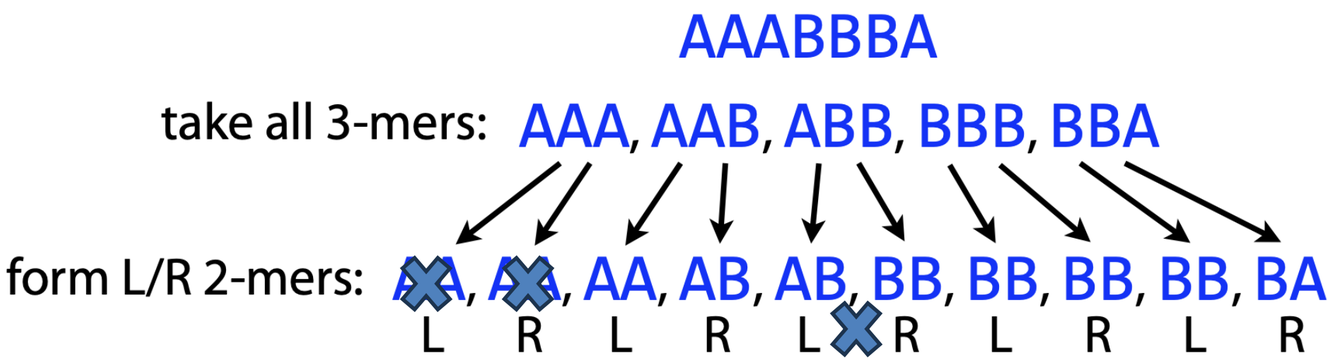


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

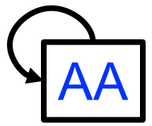


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

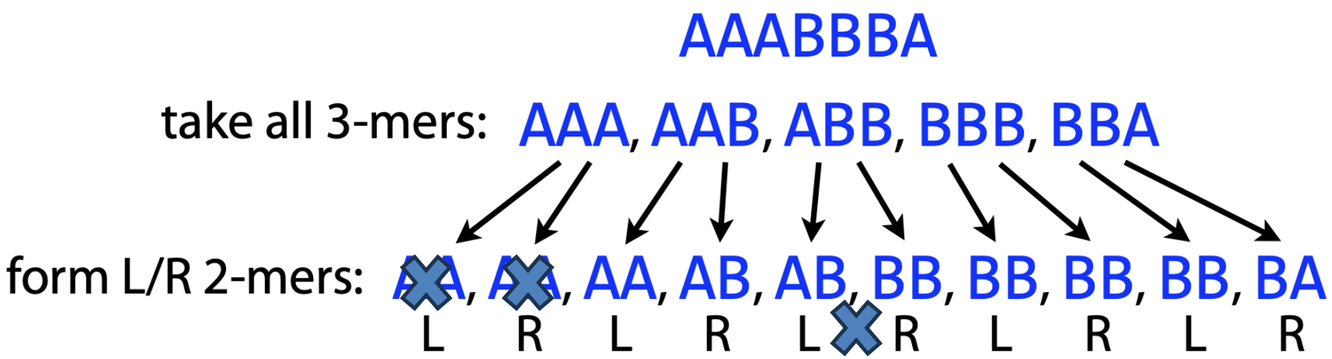


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

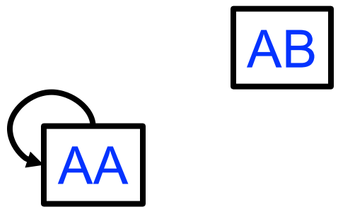


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

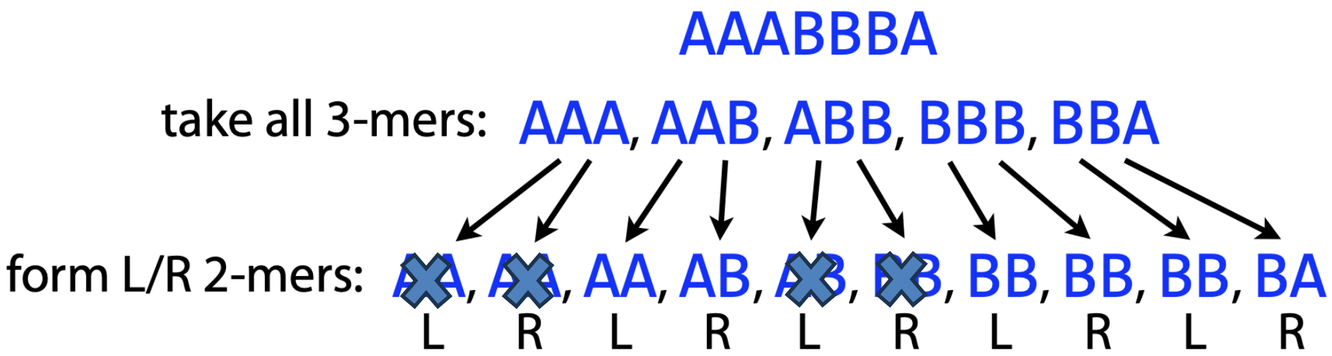


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

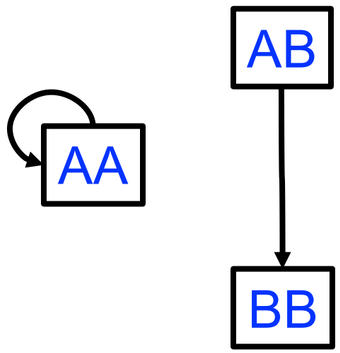


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

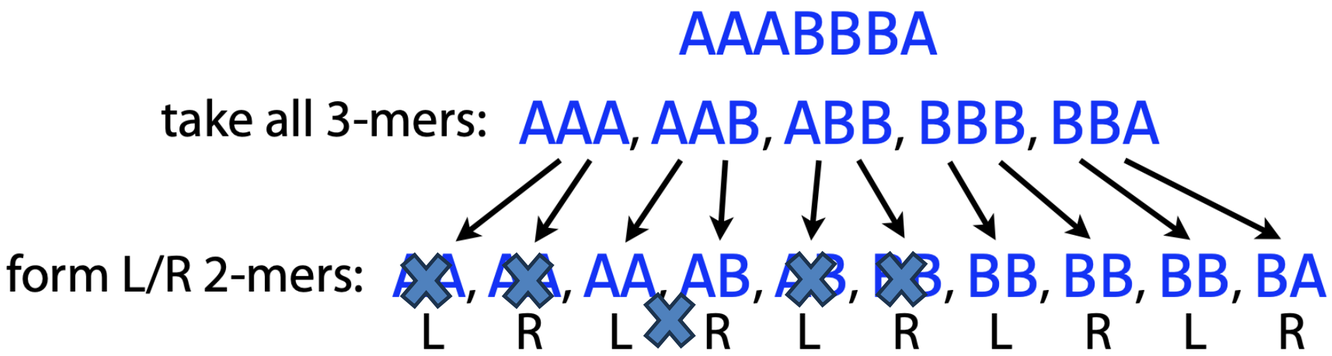


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

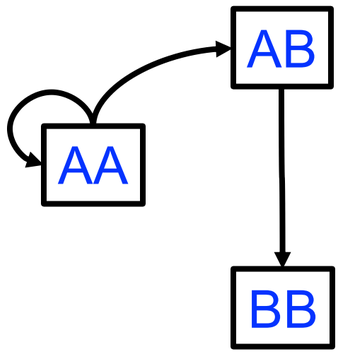


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

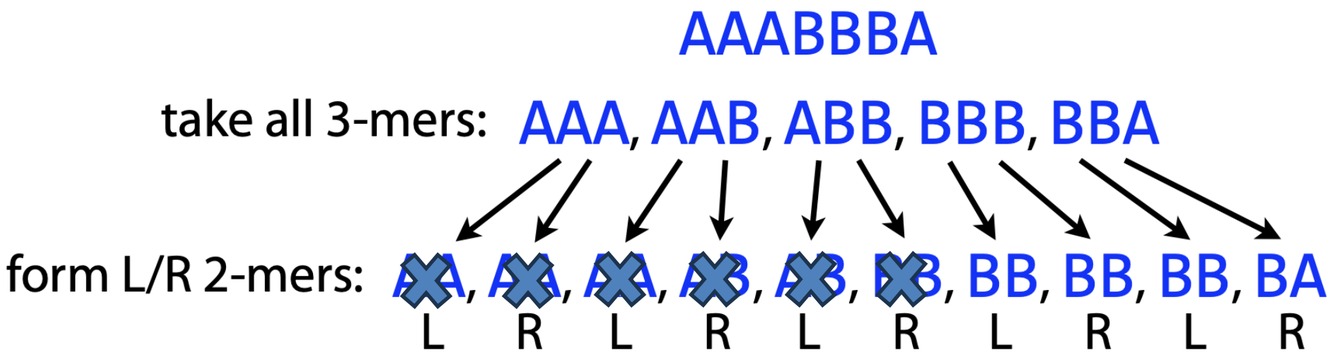


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

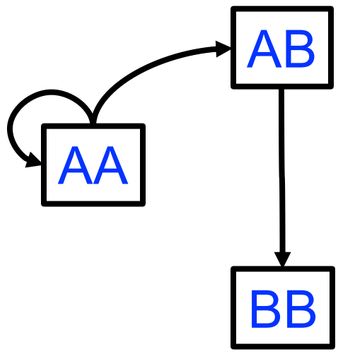


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

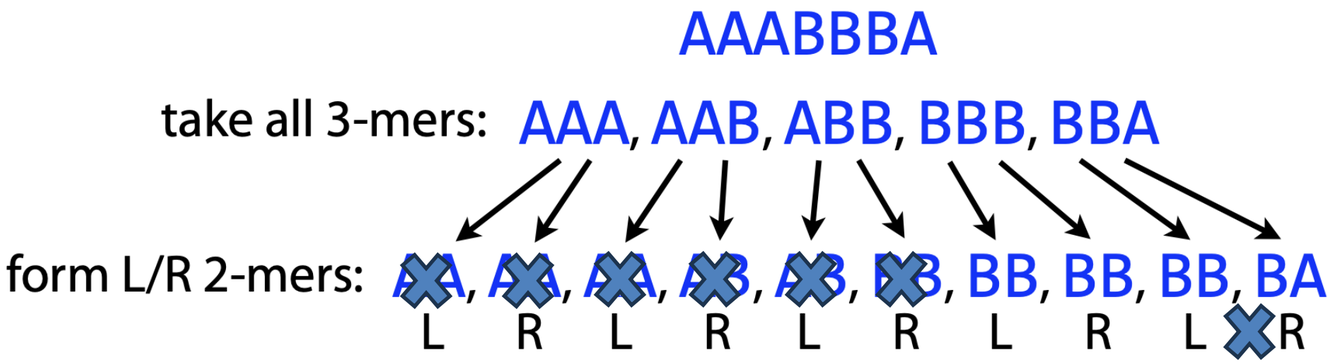


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

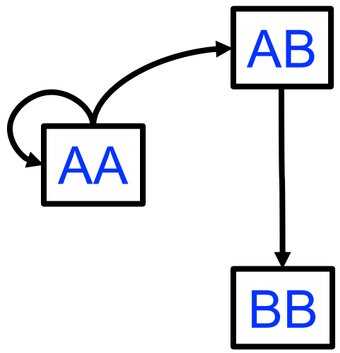


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

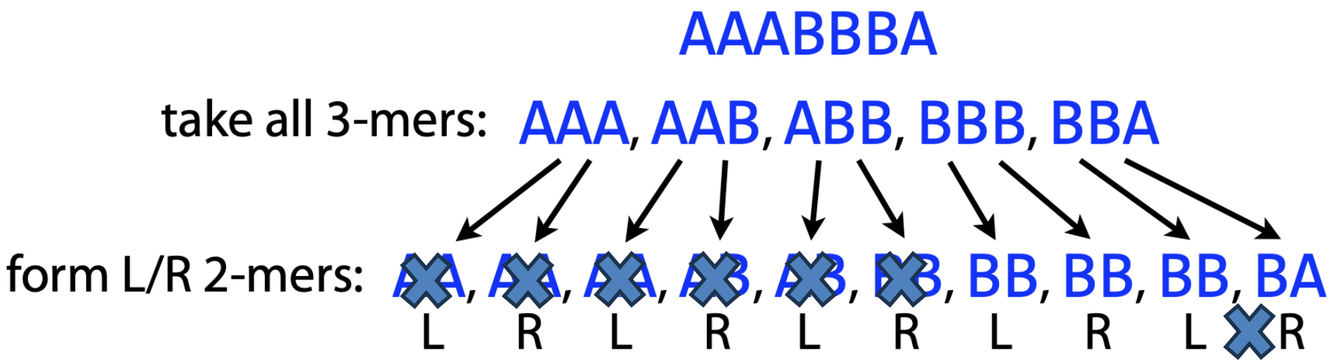


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

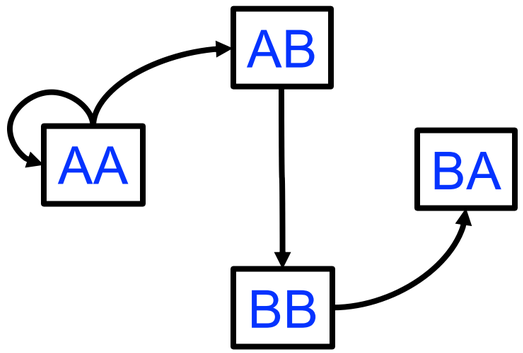


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

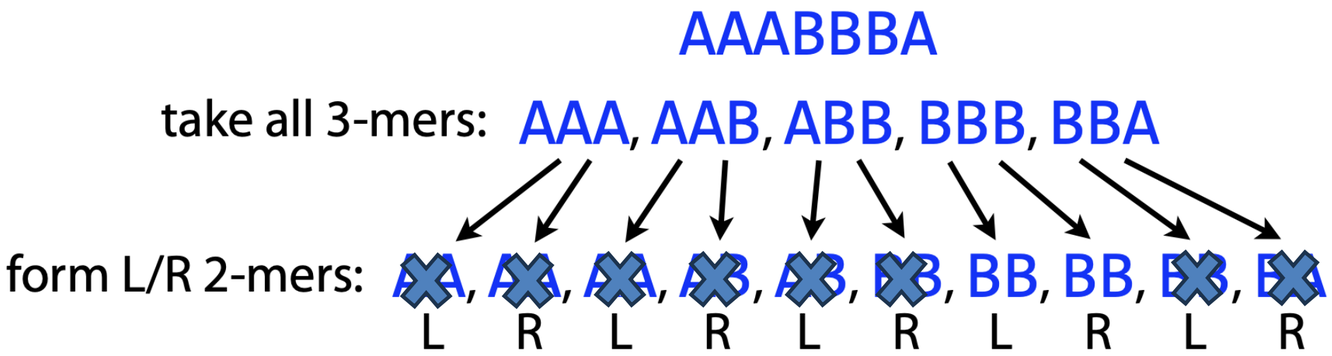


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

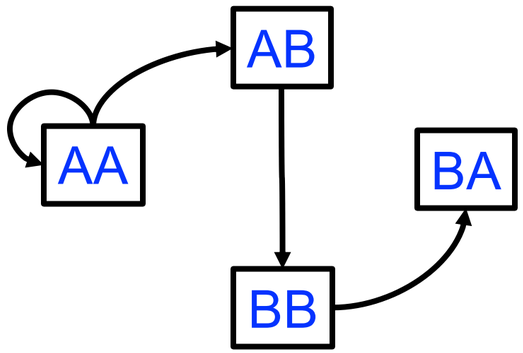


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

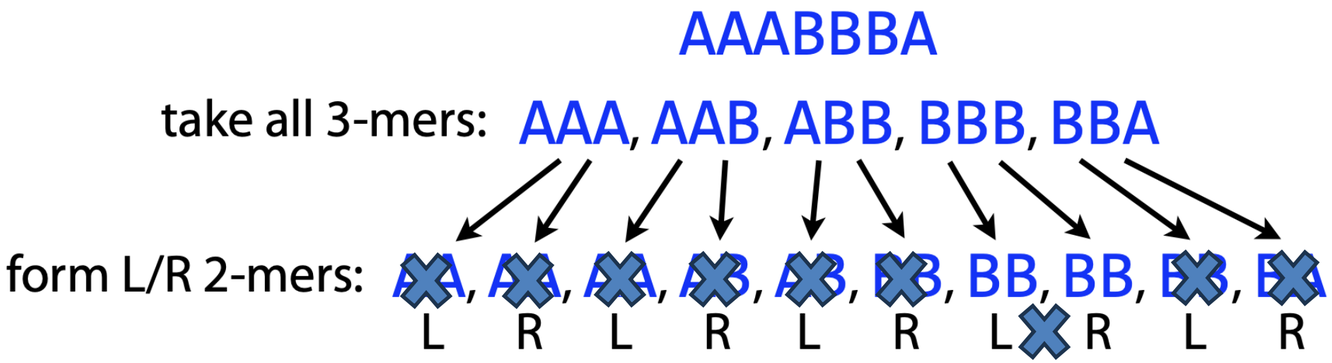


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

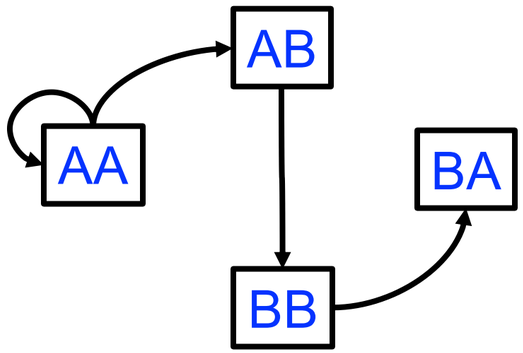


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

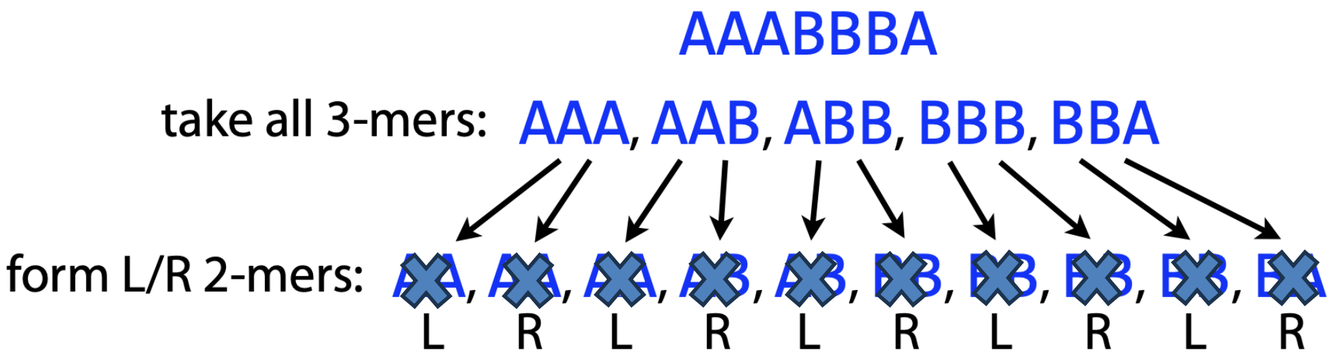


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

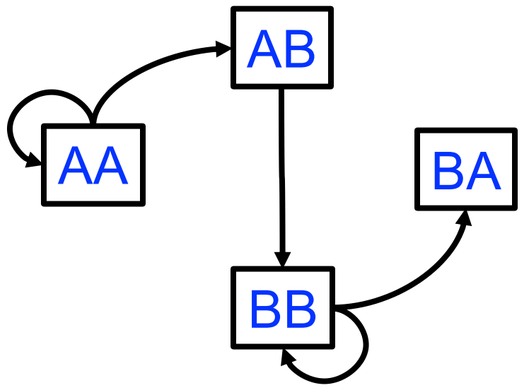


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.



Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:



De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

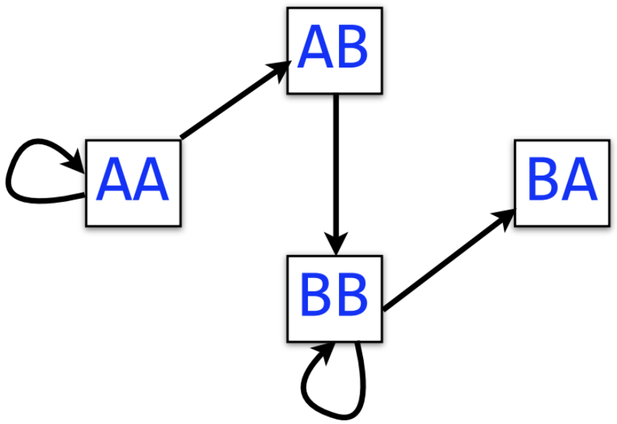
AAABBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

form L/R 2-mers: AA, AA, AA, AB, AB, BB, BB, BB, BB, BA

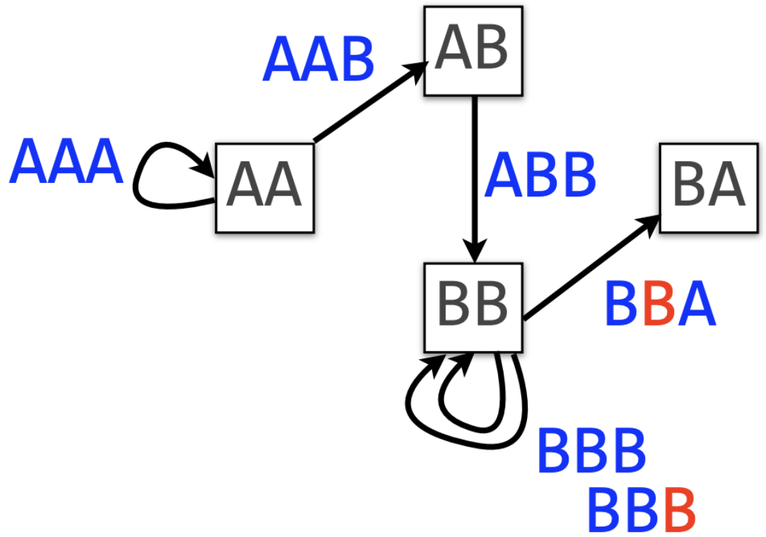
L R L R L R L R L R

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:



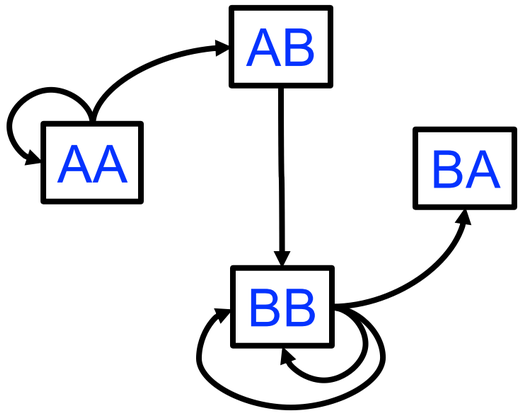
Each *edge* in this graph corresponds to a length-3 input string

De Bruijn graph



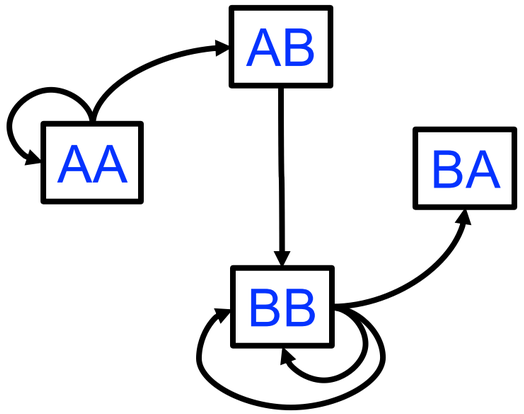
If we add one more B to our input string: AAABBBBA, and rebuild the De Bruijn graph accordingly, we get a *multiedge*.

De Bruijn graph



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

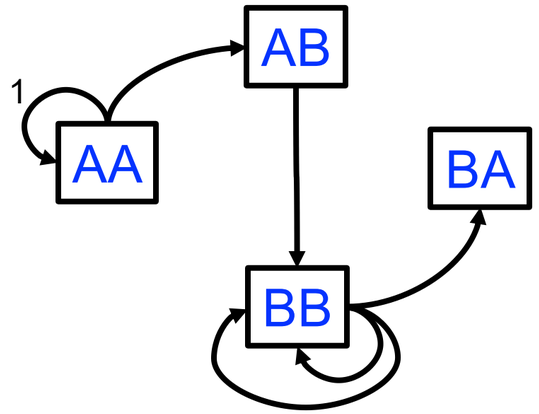
De Bruijn graph



AA

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

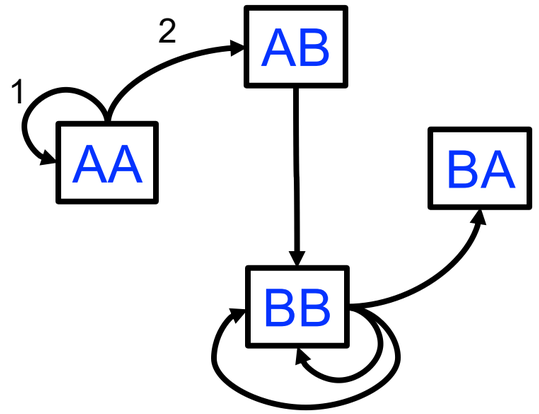
De Bruijn graph



AAA

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

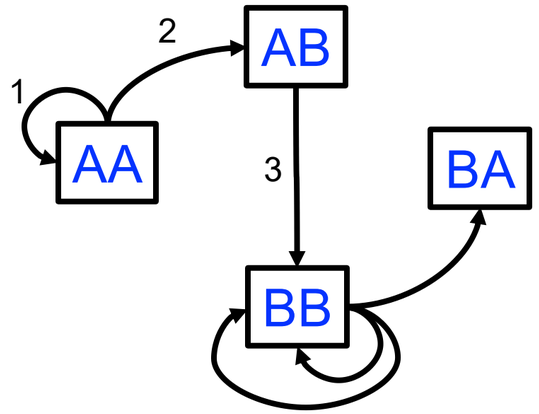
De Bruijn graph



AAAB

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

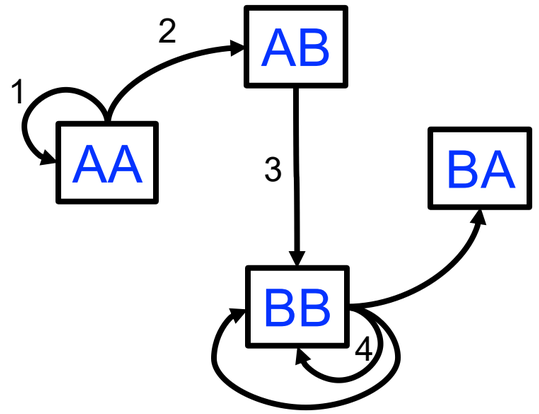
De Bruijn graph



AAABB

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

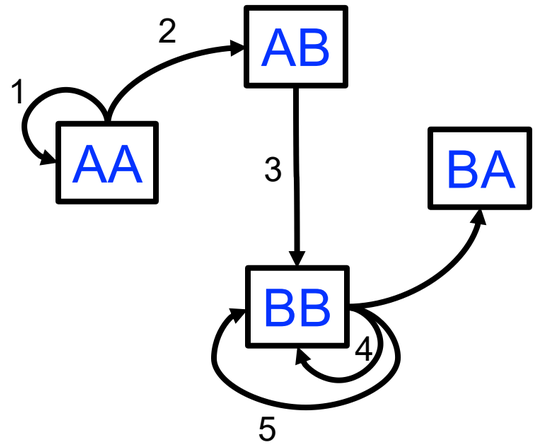
De Bruijn graph



AAABBB

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

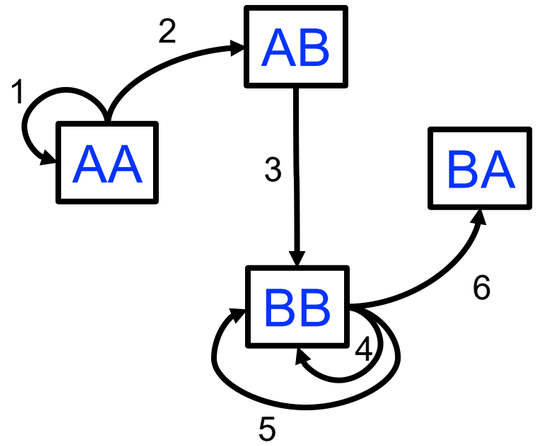
De Bruijn graph



AAABBBB

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

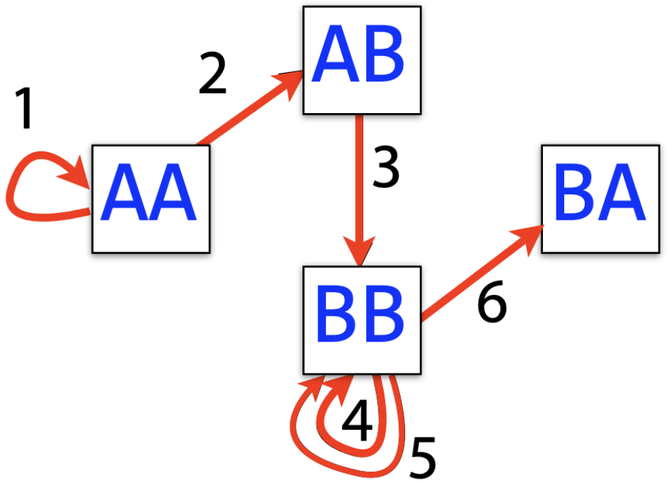
De Bruijn graph



AAABBBBA

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

De Bruijn graph



AAABBBBA

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

De Bruijn graph

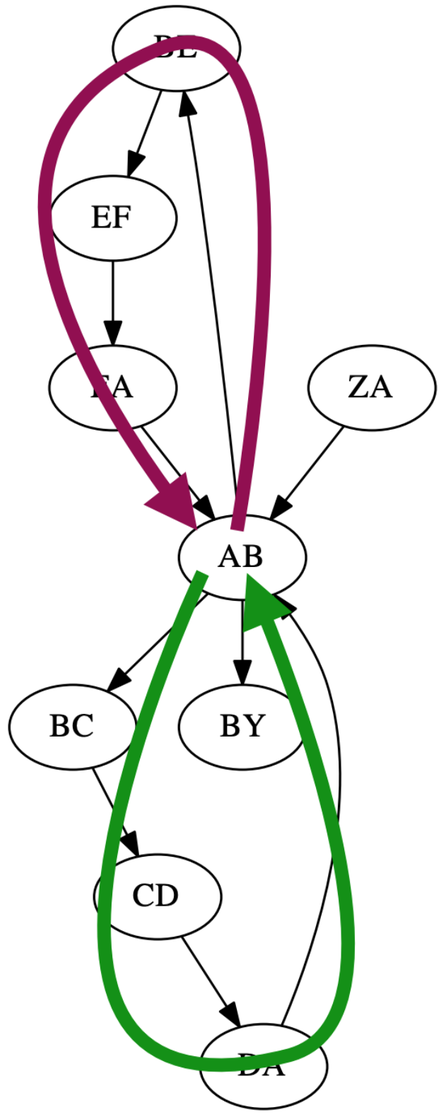
Assuming perfect sequencing, procedure yields graph with Eulerian walk that can be found efficiently.

We saw cases where Eulerian walk corresponds to the original superstring. Is this always the case?

De Bruijn graph

No: graph can have multiple Eulerian walks, only one of which corresponds to original superstring

Right: graph for ZABCDABEFABY, $k = 3$



De Bruijn graph

No: graph can have multiple Eulerian walks, only one of which corresponds to original superstring

Right: graph for ZABCDABEFABY, $k = 3$

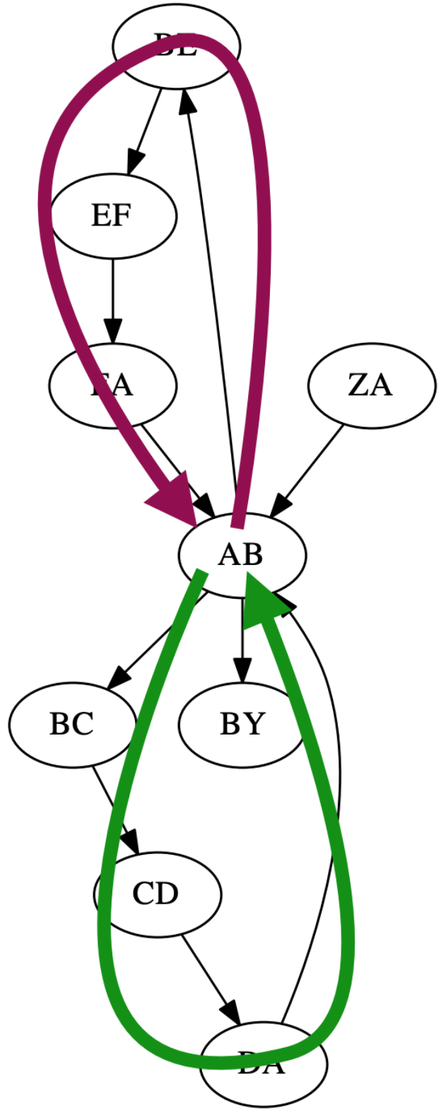
Alternative Eulerian walks:

ZA → AB → BE → EF → FA → AB → BC → CD → DA → AB → BY

ZA → AB → BC → CD → DA → AB → BE → EF → FA → AB → BY

These correspond to two edge-disjoint directed cycles joined by node AB

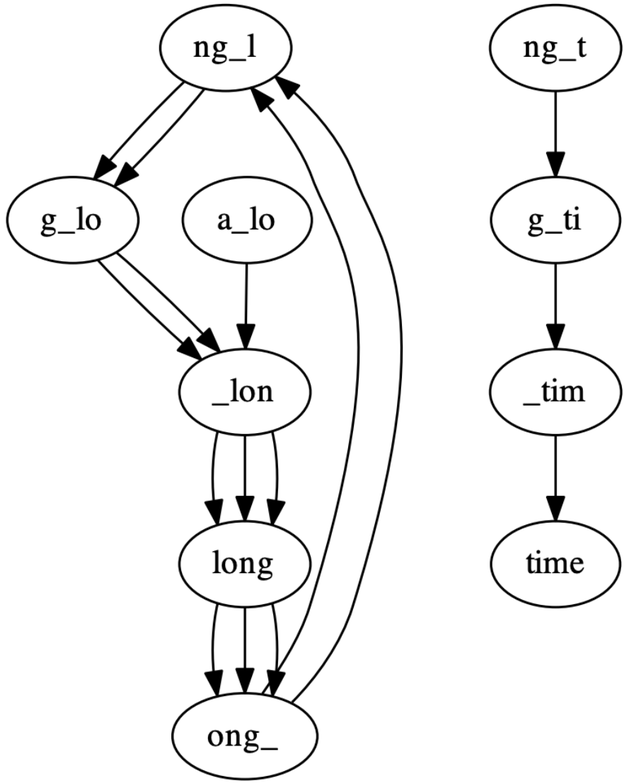
AB is a repeat: ZABCDABEFABY



De Bruijn graph

Gaps in coverage can lead to *disconnected* graph

Graph for `a_long_long_long_time`, $k = 5$ but *omitting* `ong_t` :



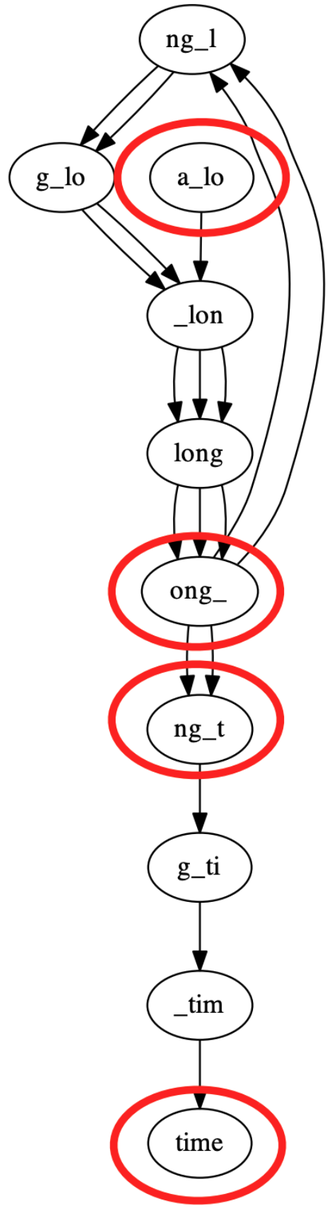
Connected components are individually Eulerian, overall graph is not

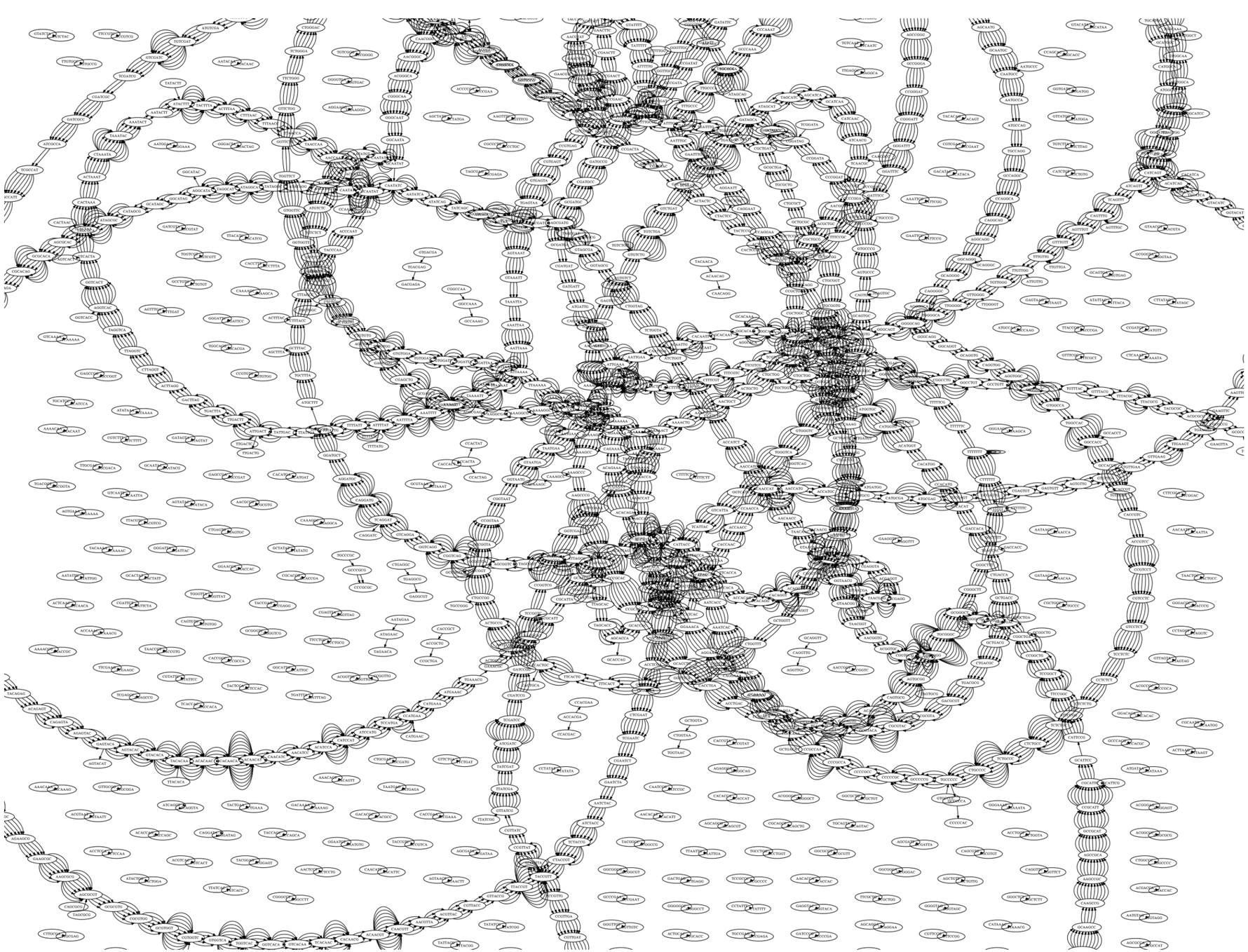
De Bruijn graph

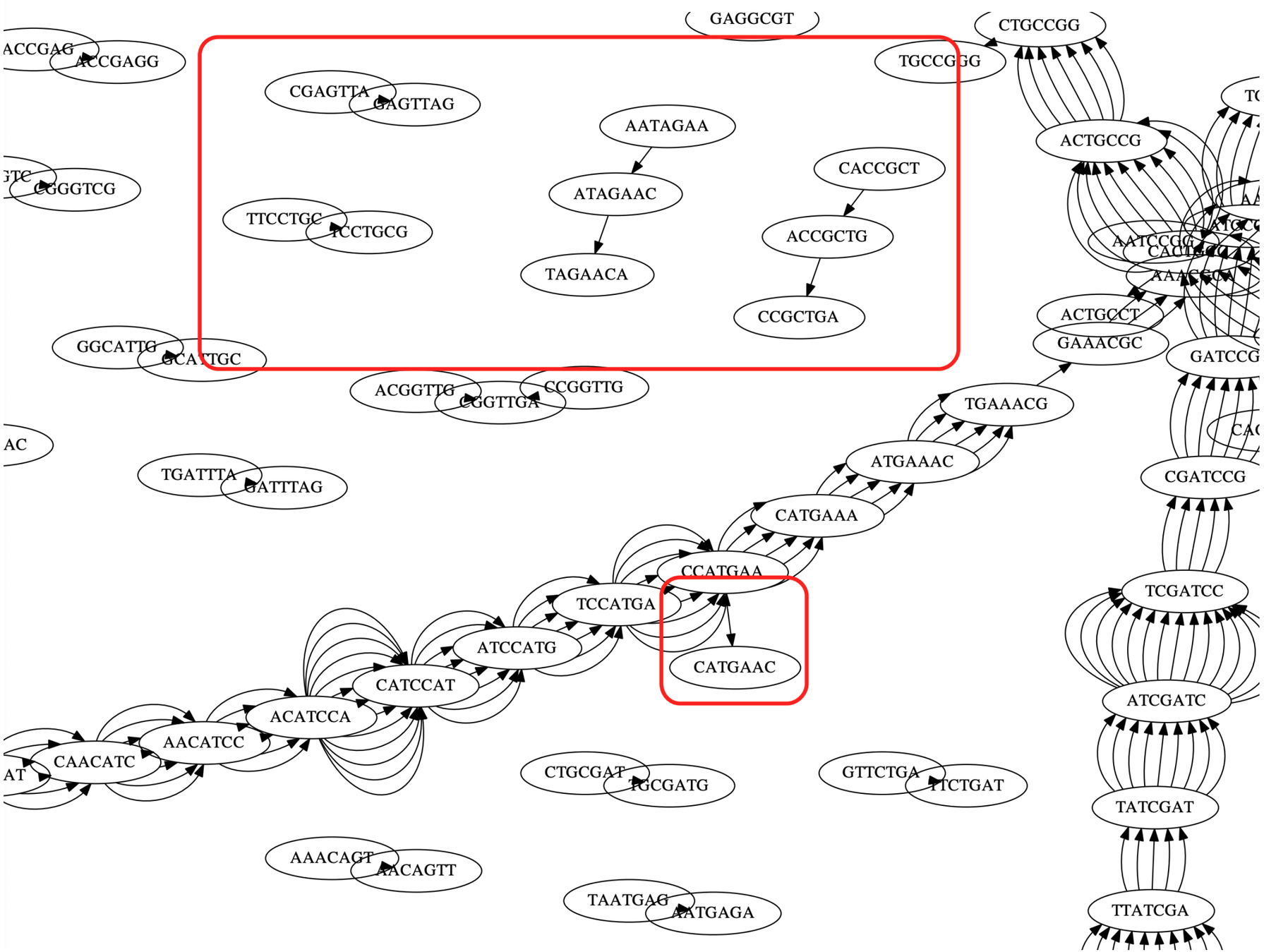
Differences in coverage also lead to non-Eulerian graph

Graph for `a_long_long_long_time`, $k = 5$ but with *extra copy* of `ong_t` :

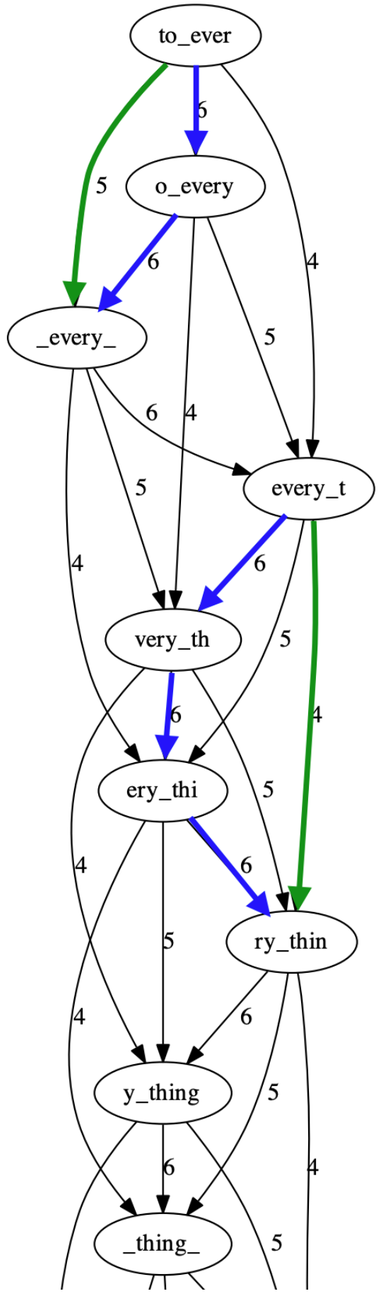
Graph has 4 semi-balanced nodes, isn't Eulerian

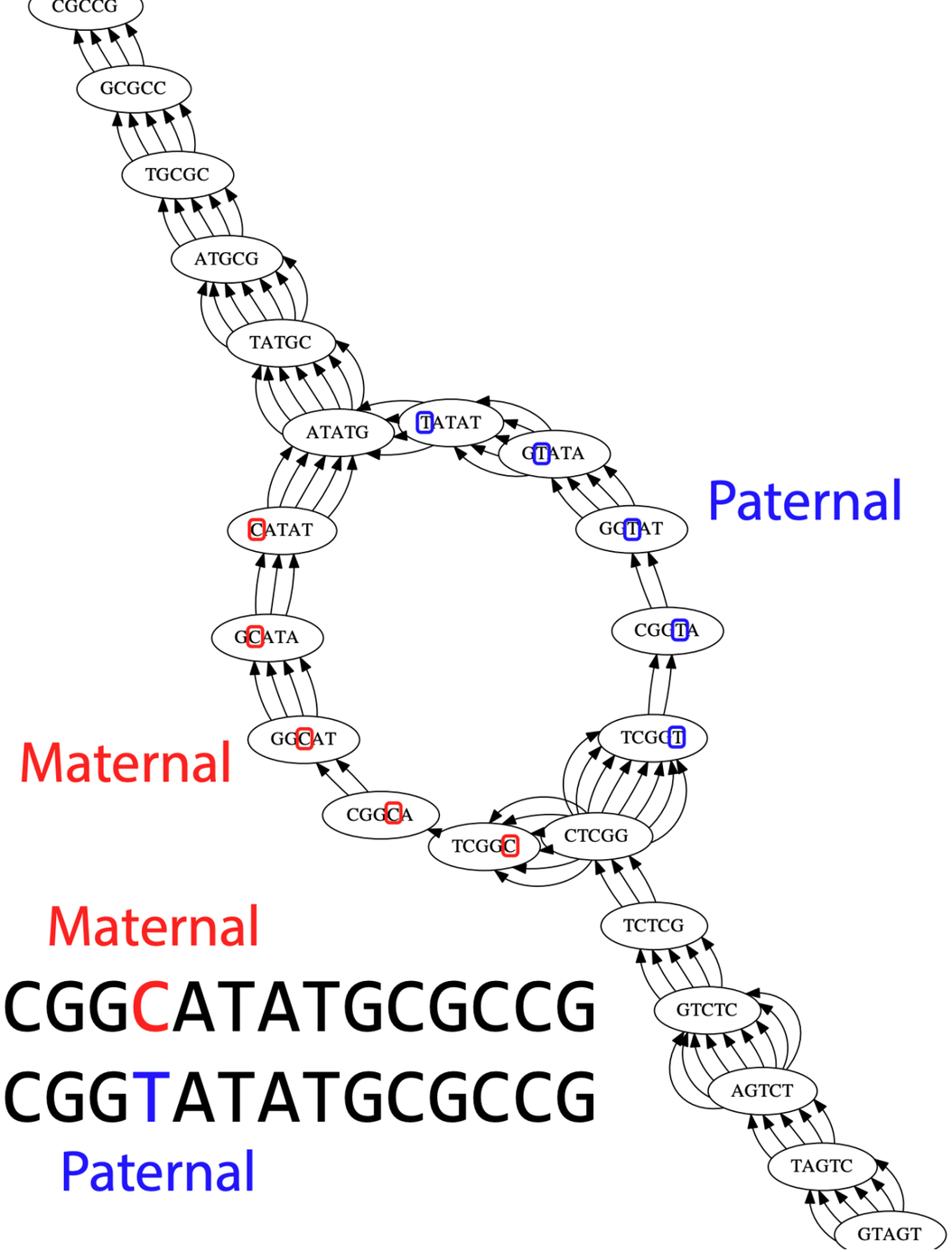


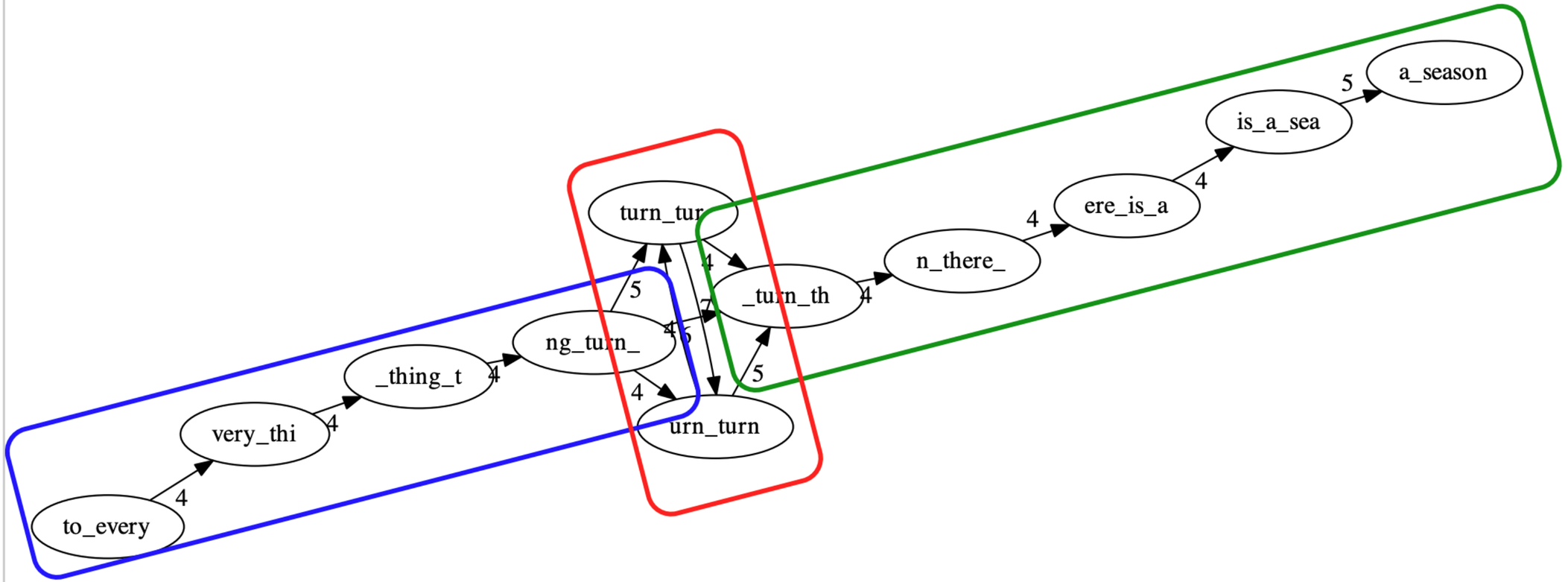




Green edges can be inferred from blue

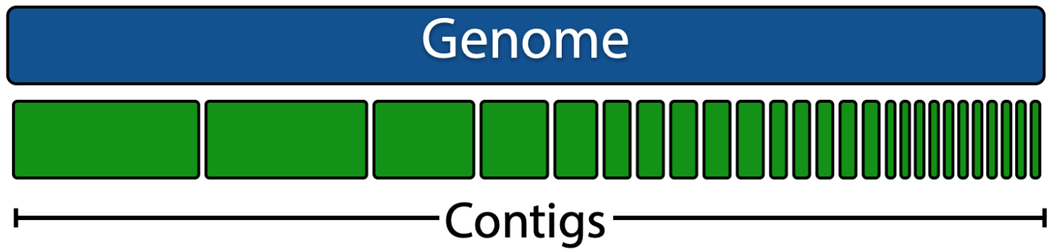


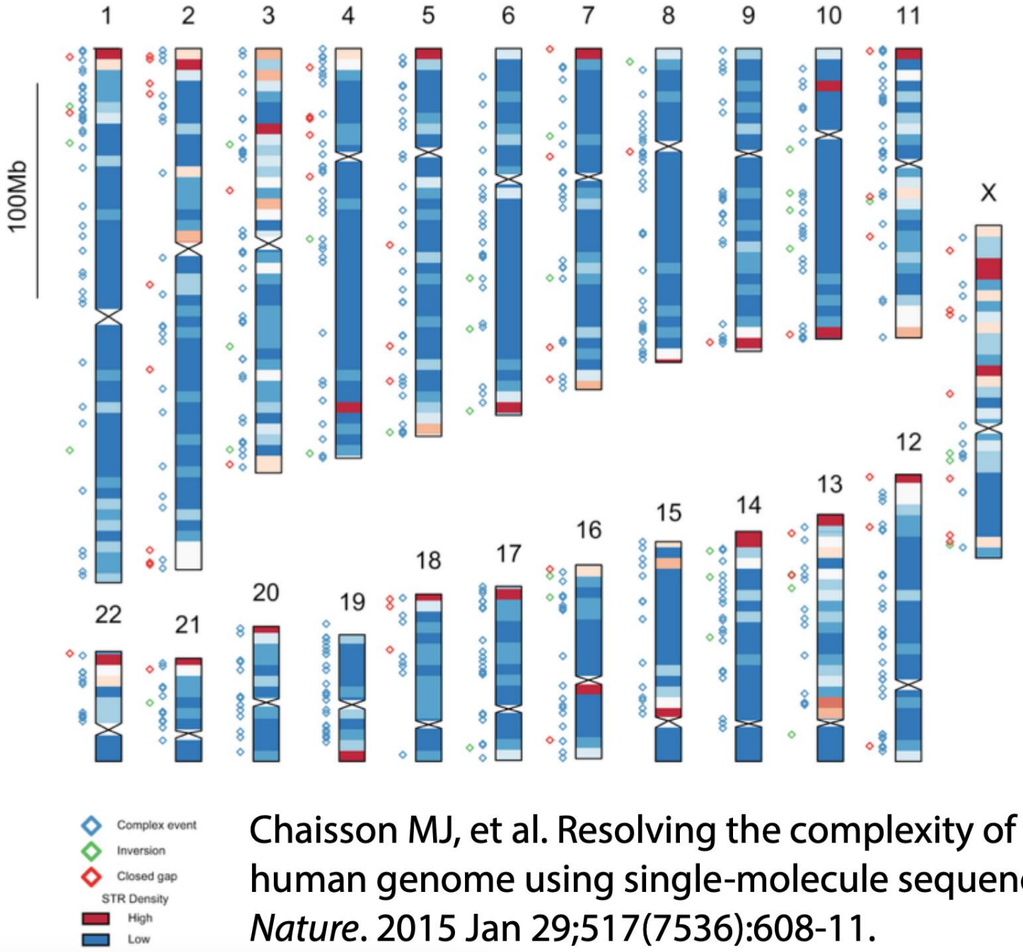

CGCCG
GCGCC
TGCGC
ATGCG
TATGC
ATATG
TATAT
GTATA
CATAT
GCTAT
GCATA
CGCTA
GGCAT
TCGCT
CGGCA
TCGGC
CTCGG
TCTCG
GTCTC
AGTCT
TAGTC
GTAGT
Paternal
Maternal
Maternal
GTAGTCTCGGCATATGCGCCG
GTAGTCTCGGTATATGCGCCG
Paternal



to_every_thing_turn_ _turn_there_is_a_season

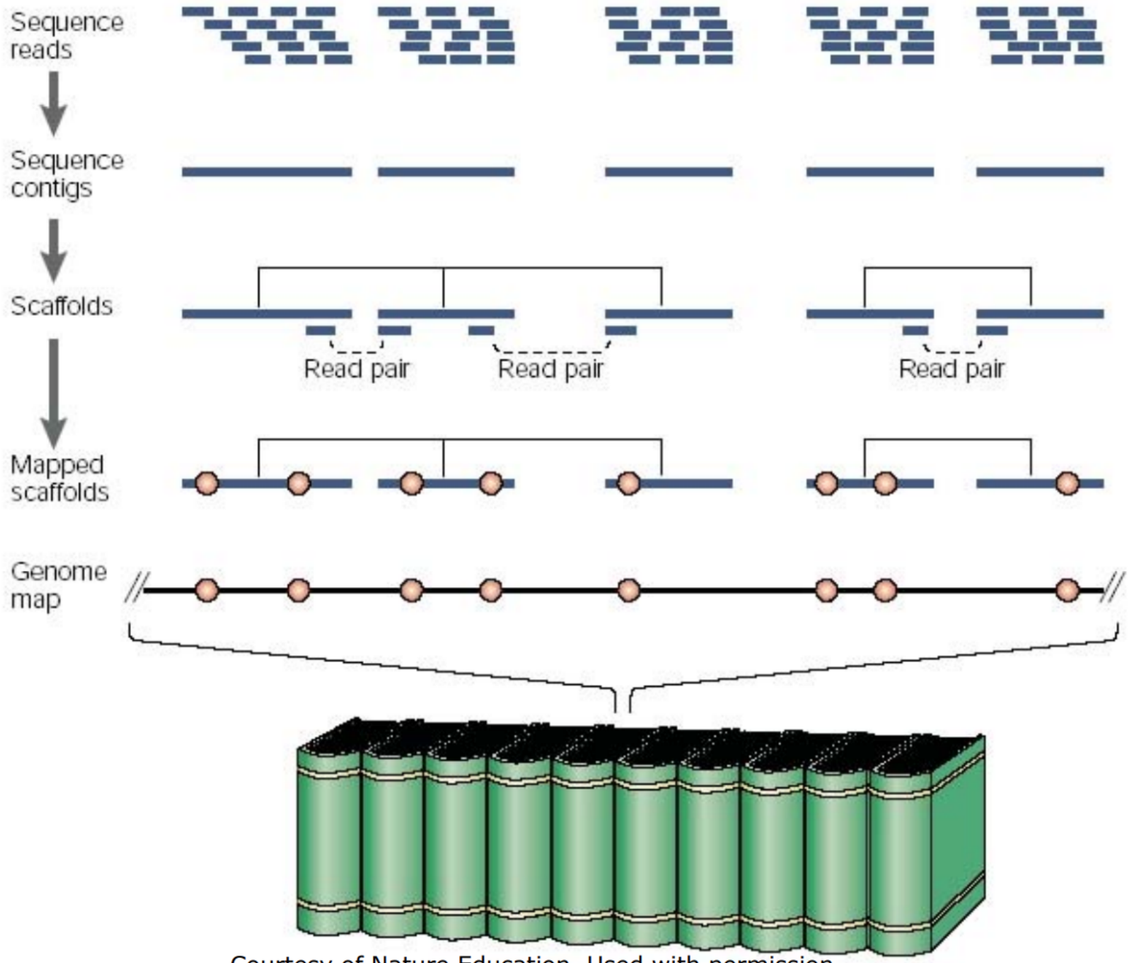
_turn (repeated)





Chaisson MJ, et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature*. 2015 Jan 29;517(7536):608-11.

de novo whole-genome shotgun assembly



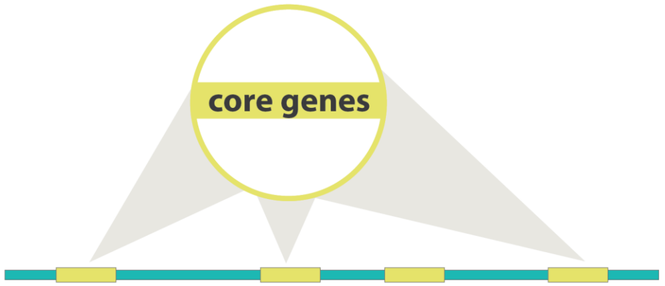
Courtesy of Nature Education. Used with permission.
Source: Green, Eric D. "Strategies for the Systematic Sequencing of Complex Genomes." *Nature Reviews Genetics* 2, no. 8 (2001): 573-83.

Adams, J. (2008) Complex genomes: Shotgun sequencing. Nature Education 1(1)

Genome assembly quality control, or “the 3C”

Contiguity

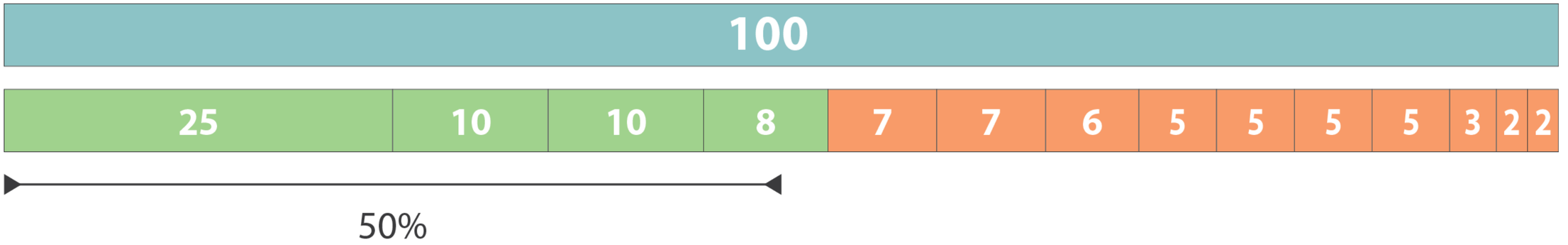
Completeness



Correctness

https://training.galaxyproject.org/training-material/topics/assembly/tutorials/assembly-quality-control/slides-plain.html

N50 & L50

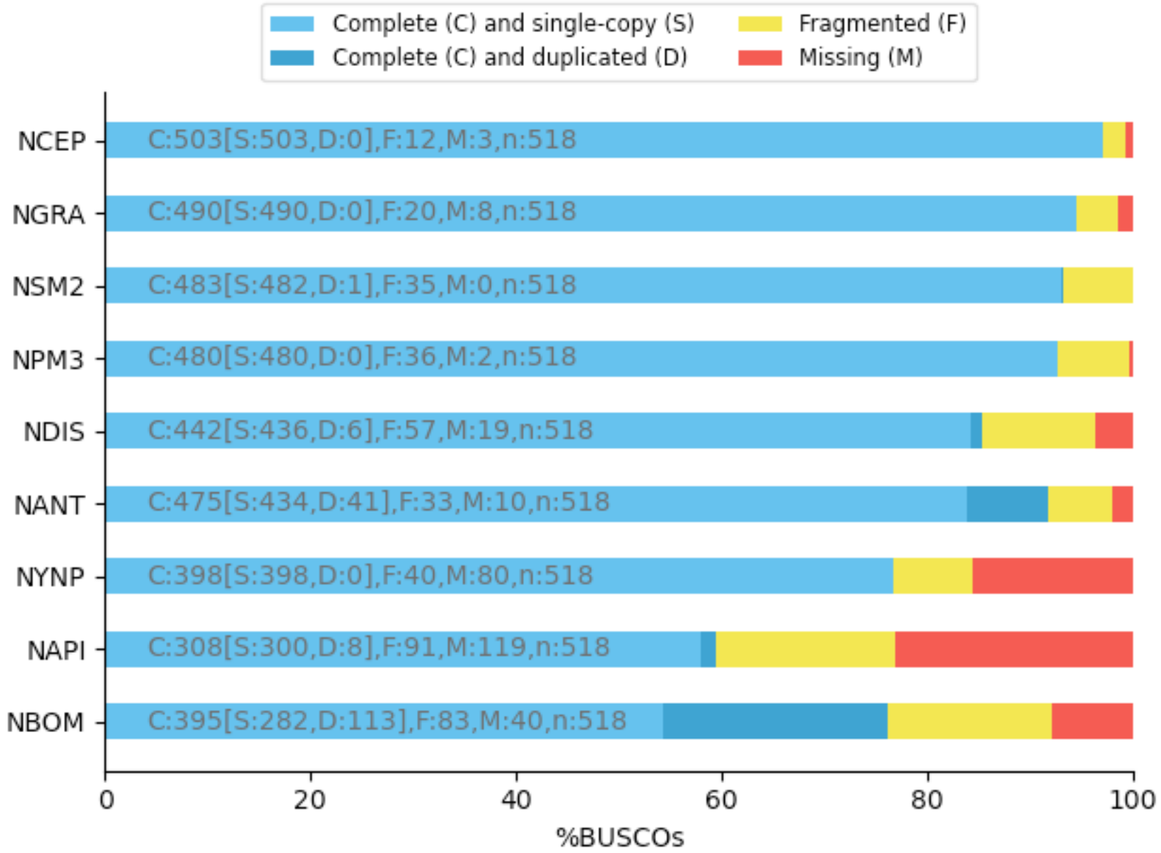


N50: given a set of sequences of varying lengths, the N50 is defined as **the length L of the shortest contig** for which **longer and equal length contigs cover at least 50% of the assembly**.
L50: given a set of sequences of varying lengths, the L50 is defined as **count of smallest number of sequences** whose **length sum makes up 50% of the assembly**.
N50 describes a sequence length whereas L50 describes a number of sequences.

N50 = 8 and **L50 = 4**

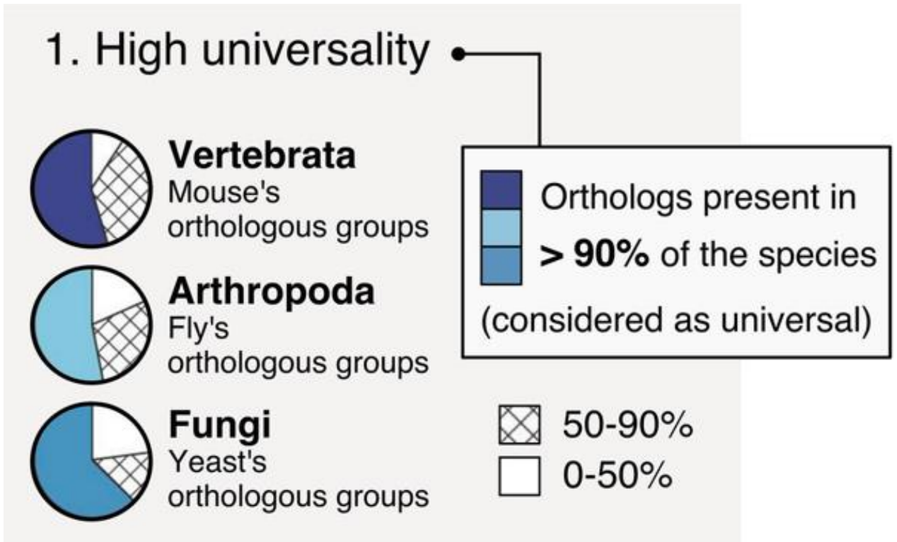
“Core” genes

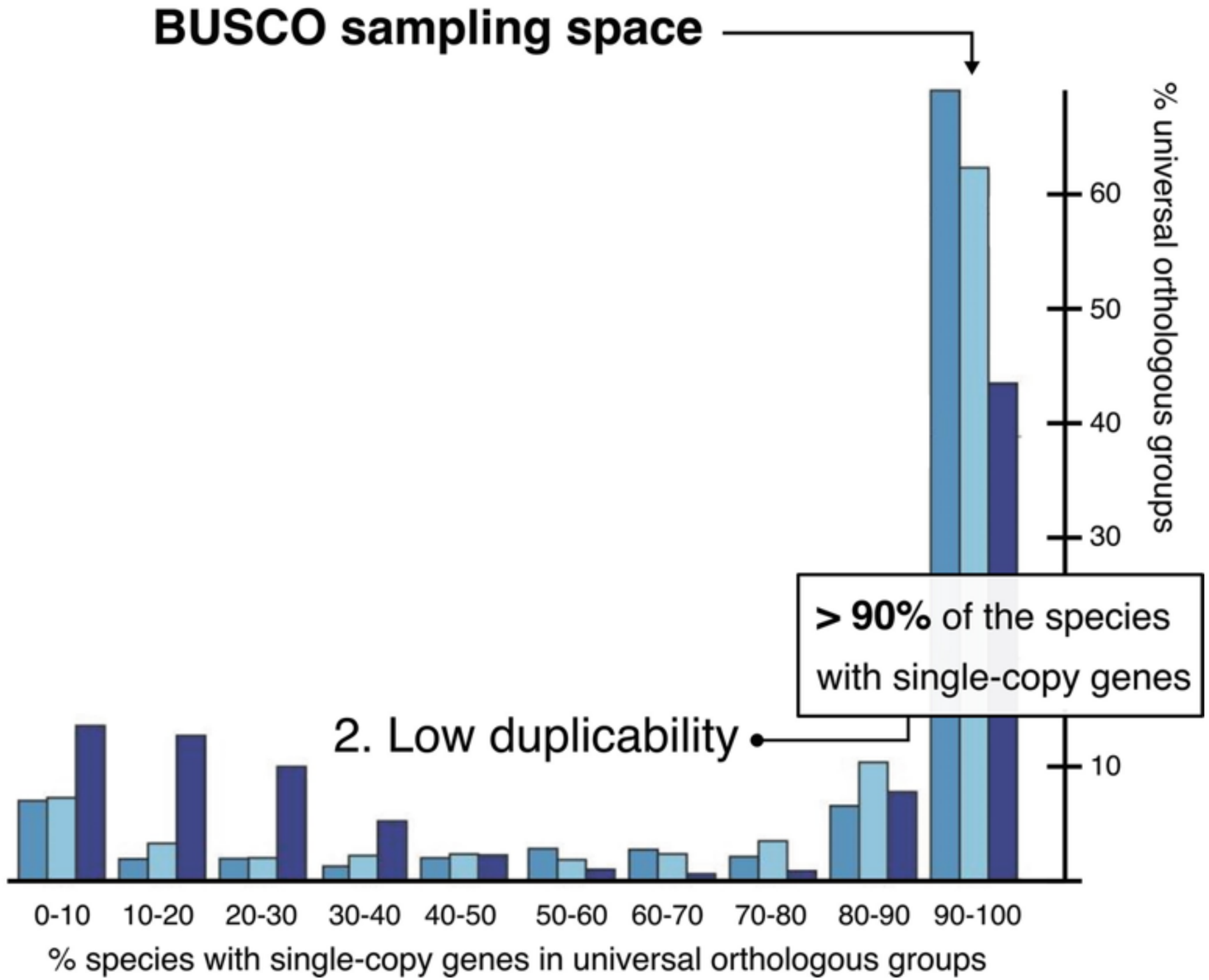
$$\frac{\textit{Core genes in assembly}}{\textit{Core genes in reference database}}$$



https://training.galaxyproject.org/training-material/topics/assembly/tutorials/assembly-quality-control/slides-plain.html

“Core” genes

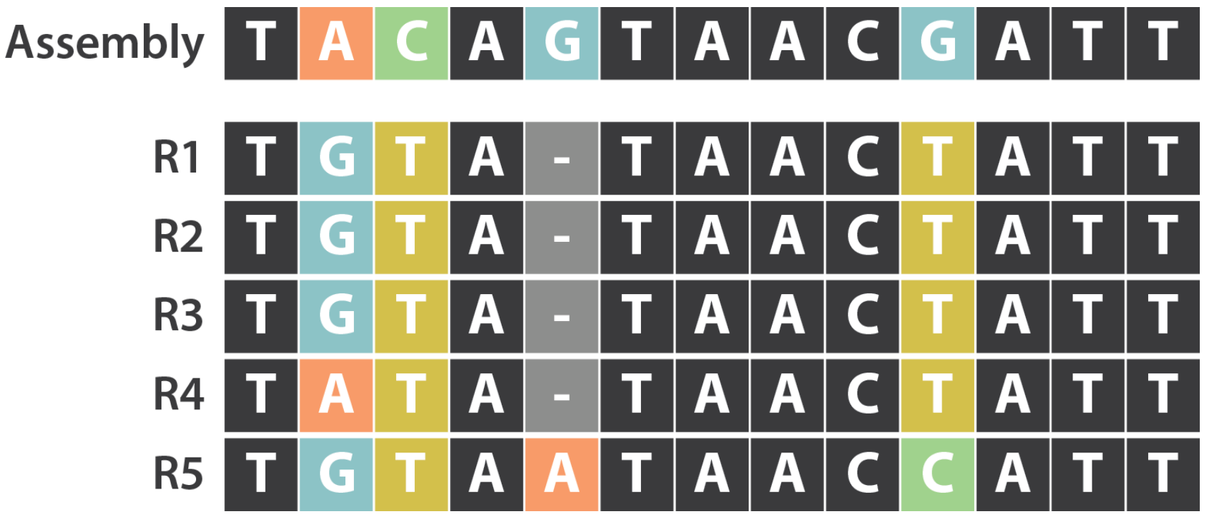


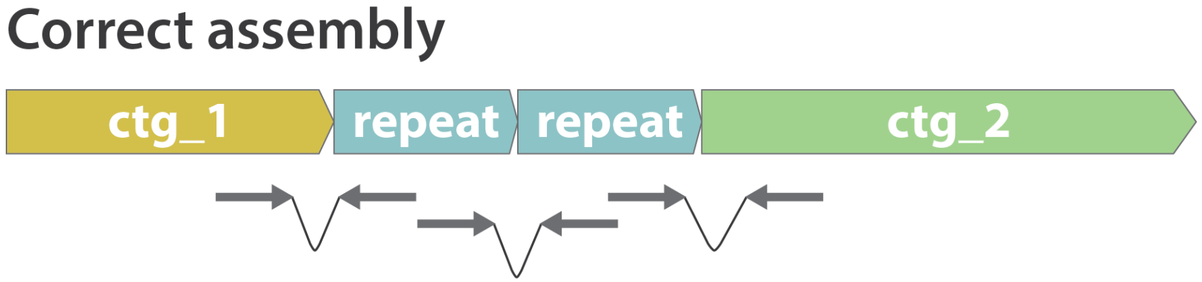


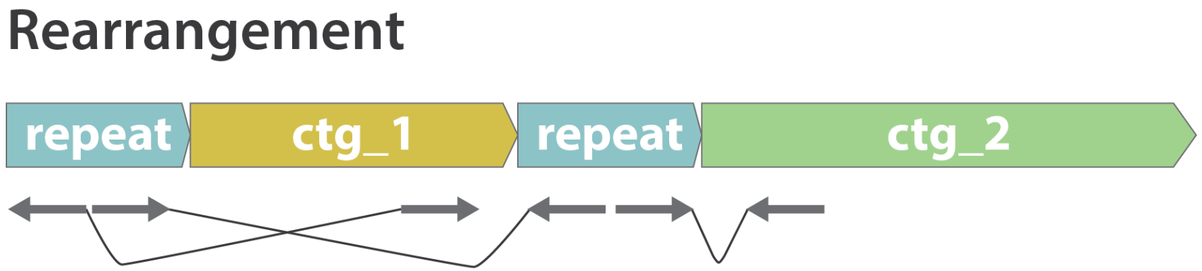
https://training.galaxyproject.org/training-material/topics/assembly/tutorials/assembly-quality-control/slides-plain.html

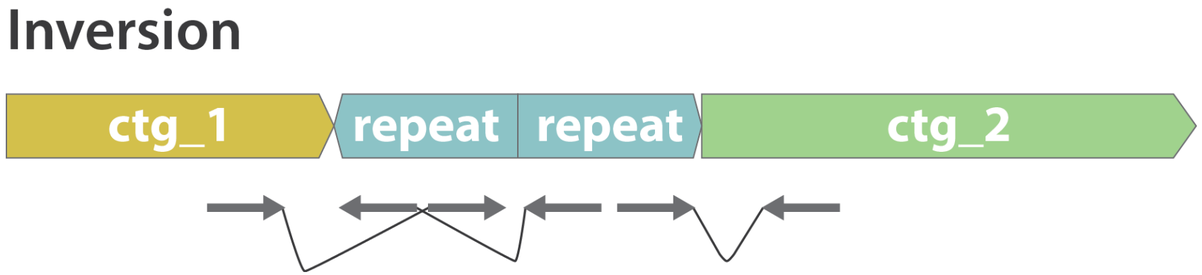
Mistakes into the assembly

- Indels / SNPs
- Mis-joins
- Repeat compressions
- Unnecessary duplications
- Rearrangements





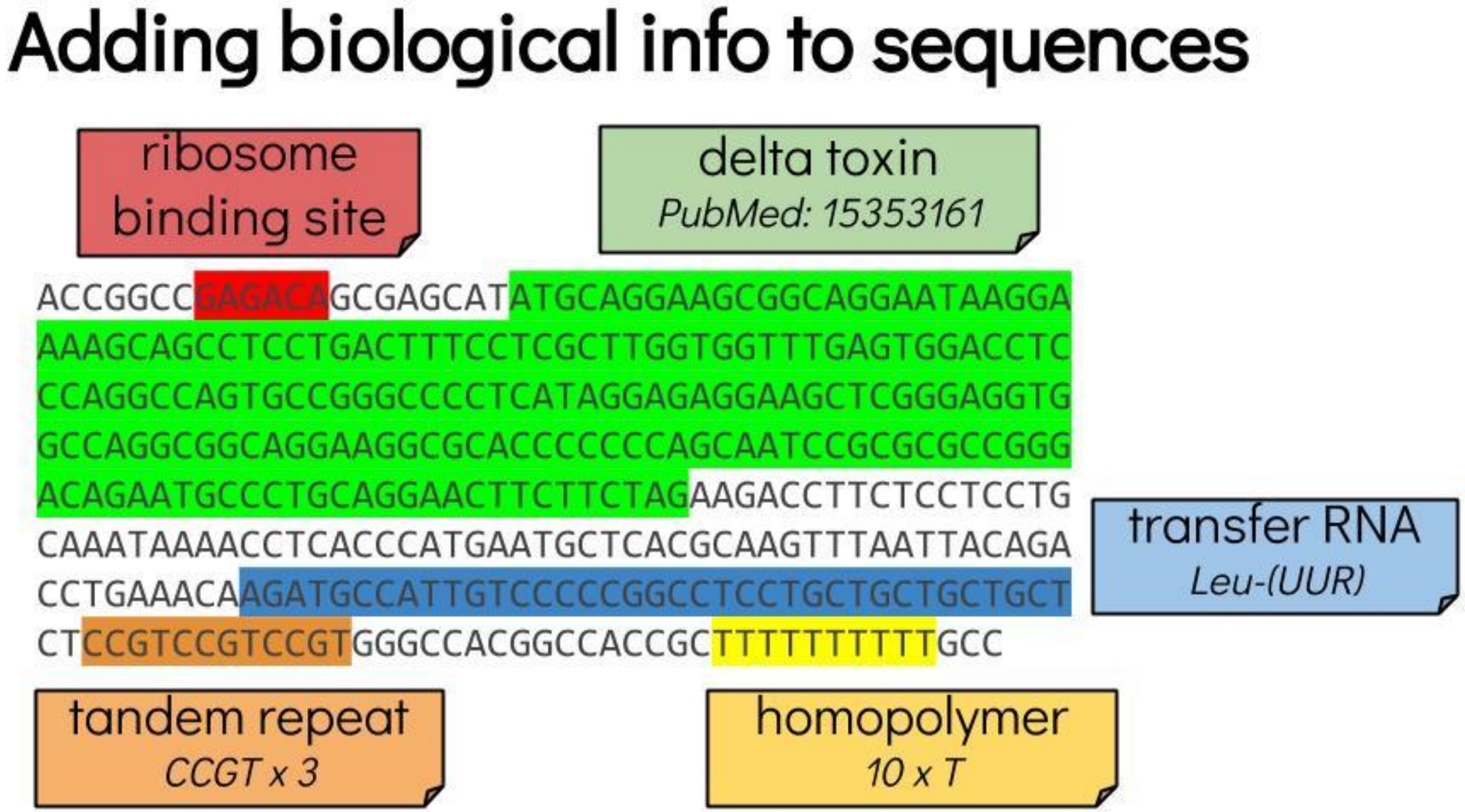




https://training.galaxyproject.org/training-material/topics/assembly/tutorials/assembly-quality-control/slides-plain.html

Proxima aula…

Anotação de genomas (teórico-prática)



https://training.galaxyproject.org/training-material/topics/genome-annotation/tutorials/annotation-with-prokka/slides-plain.html