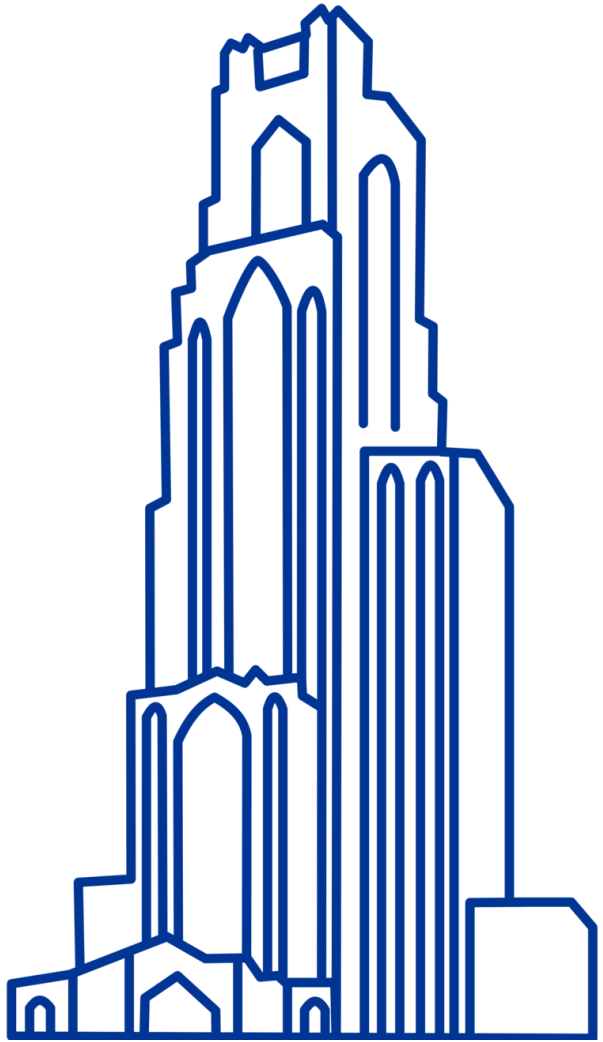


Computational Biology
(BIOSC 1540)

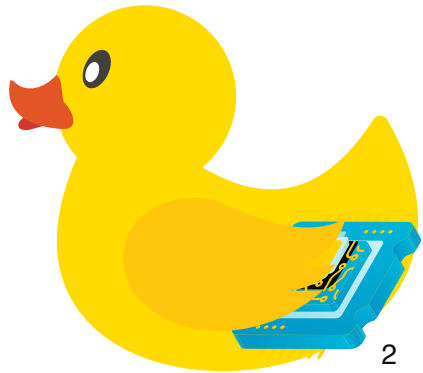
Lecture 04:
De novo assembly

Sep 5, 2024

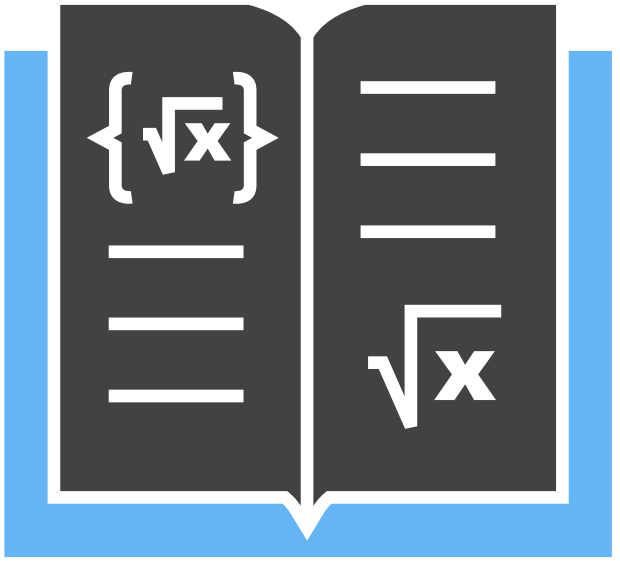
University of Pittsburgh

Announcements

- A01 is due tonight at 11:59 pm
- A02 will be released tomorrow and due next Thursday

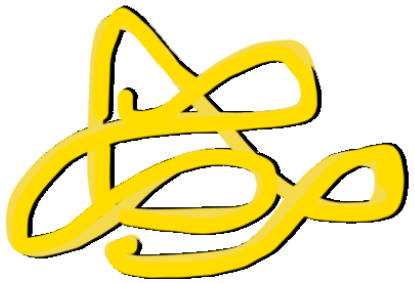


We are putting our computer algorithm hats now



After today, you should be able to

1. **Explain the fundamental challenge of reconstructing a complete genome.**
2. Describe and apply the principles of the greedy algorithm.
3. Understand and construct de Bruijn graphs.



Genomic DNA

Next-generation
DNA sequencing

...CATTCAGTAG...
...AGCCATTAG...
...GGTAGTTAG...
...GGTAGTTAG...
...AGCCATTAG...
...GGTAAACTAG...

Millions-billions of ***reads***
~30-1,000 nucleotides

RESEQUENCING

Align reads to ***reference genome***
and identify variants

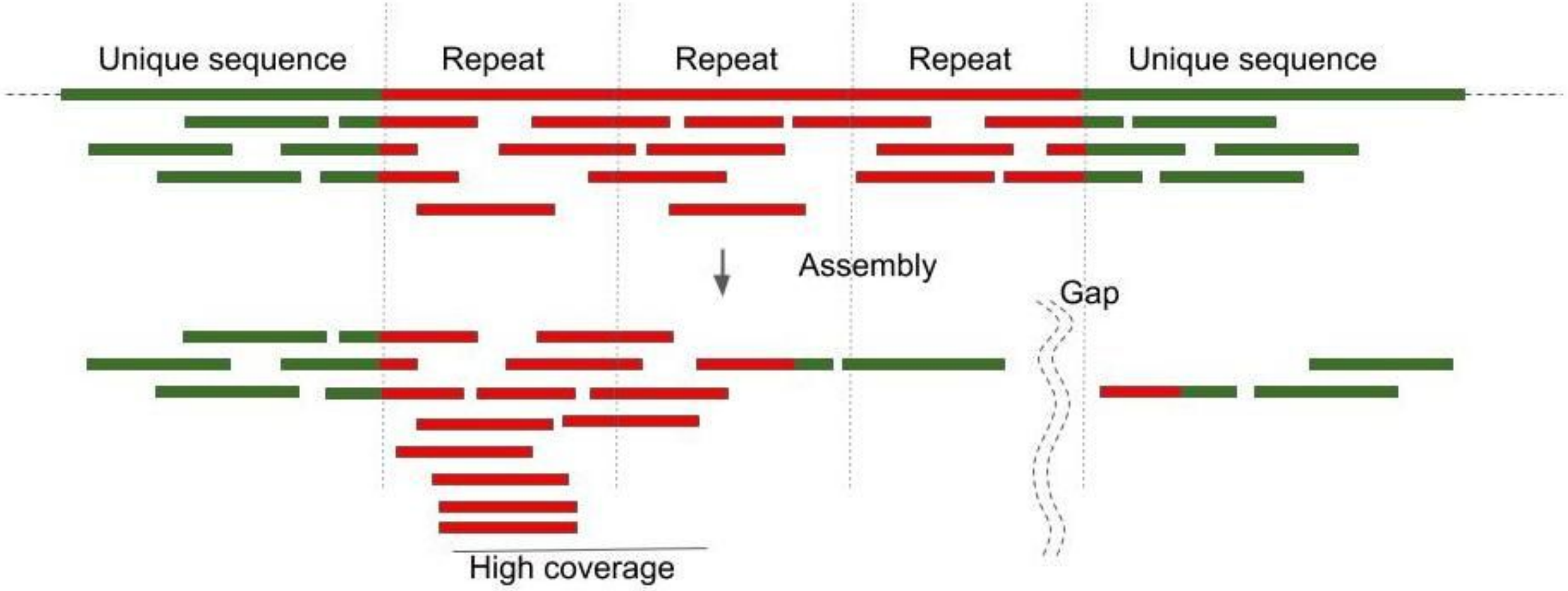
De Novo ASSEMBLY

Construct ***genome sequence***
from overlaps between reads

This is just alignment with extra steps (our topic for next Thursday)

What is done 99% of the time

Repeats and high coverage are the main challenges



After today, you should be able to

1. Explain the fundamental challenge of reconstructing a complete genome.
2. **Describe and apply the principles of the greedy algorithm.**
3. Understand and construct de Bruijn graphs.

Let's formulate our problem

Suppose we have a **collection of strings** (i.e., reads)

(In CS, we call a sequence of characters a **string**)

BAA **AAB** **BBA** **ABA**

ABB **BBB** **AAA** **BAB**

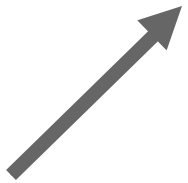
We want to assemble these strings into a **single, continuous string** (i.e., contig)

What's the easiest way?

Concatenate

BAAAABBBAABAABBBBBAAABAB

Done! Right?



This is called a "superstring"

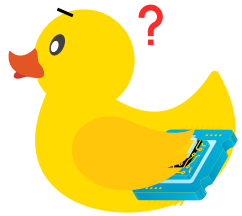
Well, no.

Suppose we want the **shortest** superstring

BAAAABBBAABAABBBBBBAAABAB

This is a valid superstring, but why would we want the shortest?

Talk with your neighbors



Overlap maximization

- Reduces redundancy
- Maximizes confidence with highest overlaps

Repeat resolution

Resolves repeats by favoring collapsed arrangements

Evolutionary pressure

Most genomes have selective pressure to be efficient

Merge strings by highest overlap

Concatenated: BAAAABBBAABAABBBBBAAABAB

Overlapped: AAABBBABAA

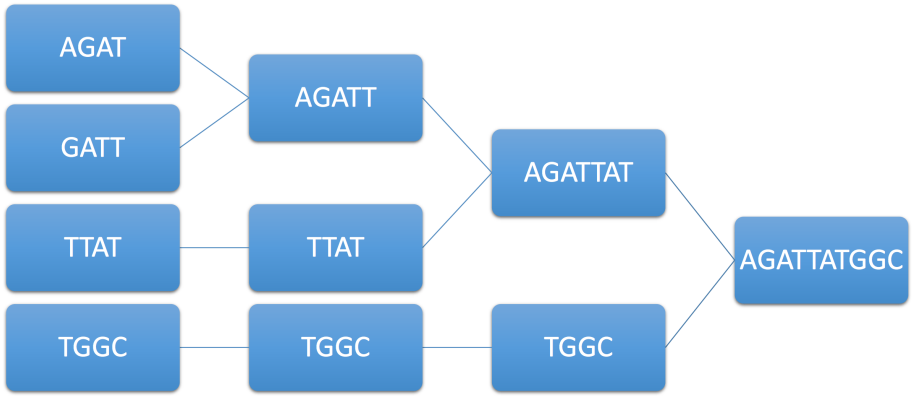
We can arrange the strings with overlaps of two

```
AAA
 AAB
  ABB
   BBB
    BBA
     BAB
      ABA
       BAA
```

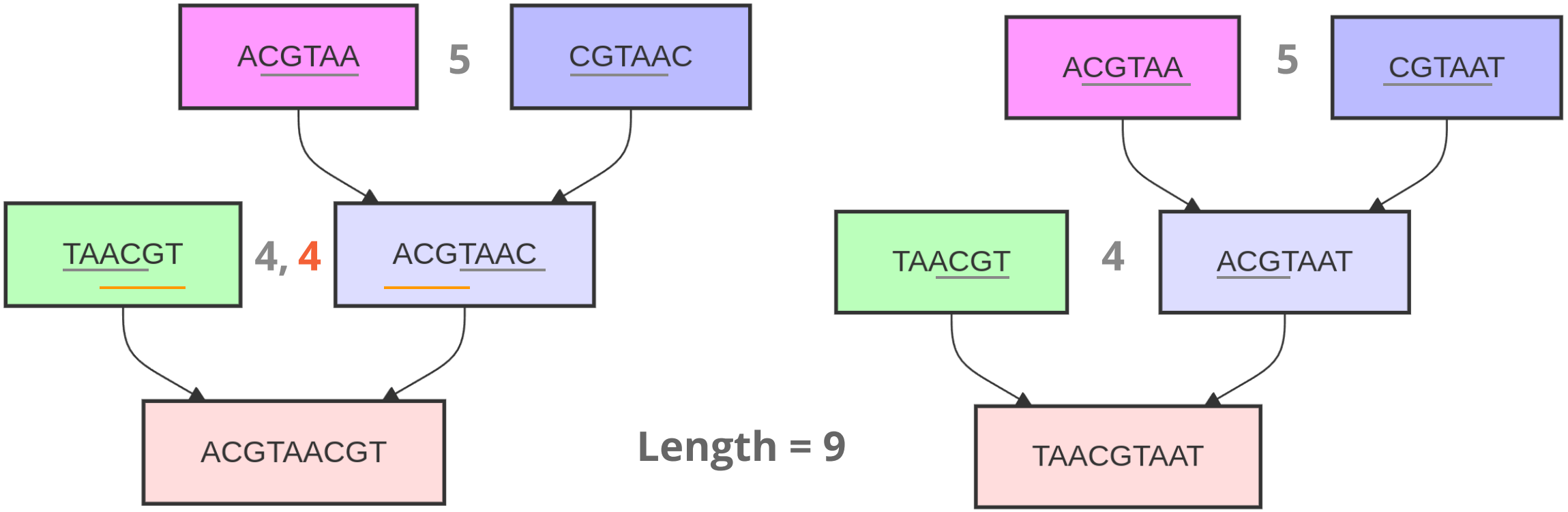
Great! That was easy

Procedure

1. Merge strings one at a time keeping a consistent 5' and 3'
2. Always merge the largest overlap
3. Repeat

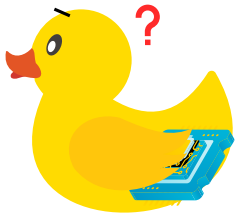


What happens if we have a tie?



Length = 9

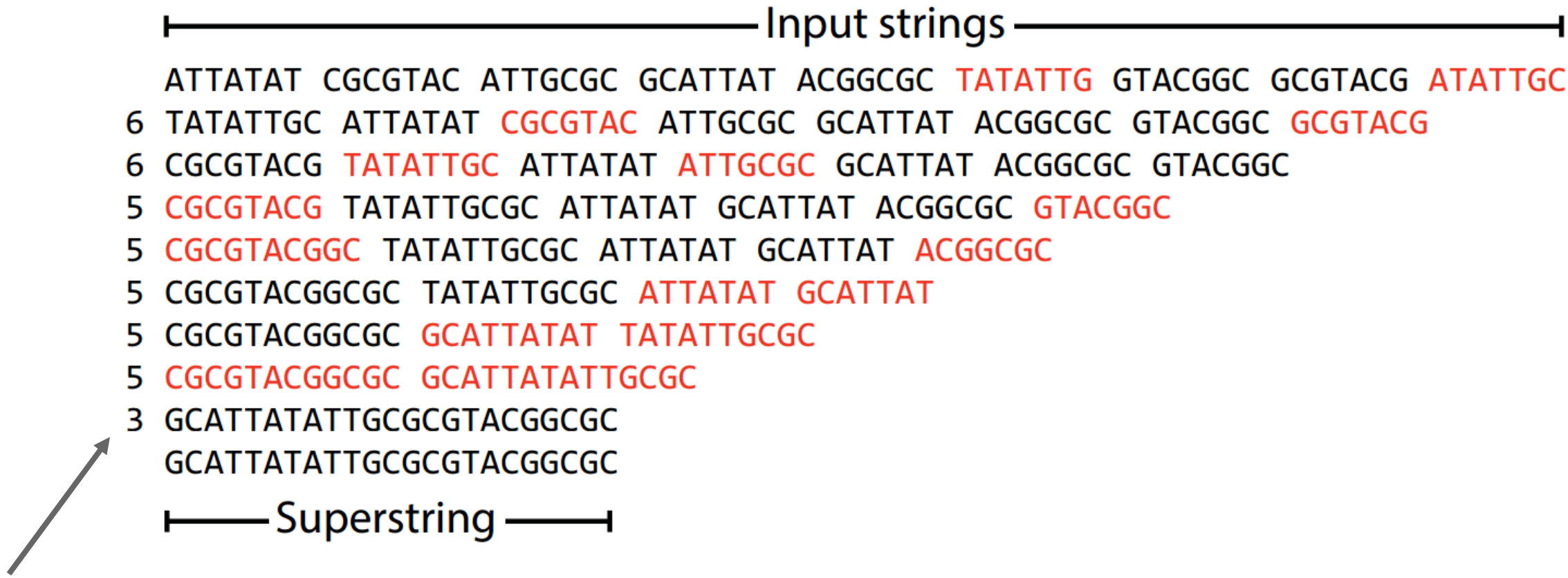
Talk with your neighbors



Tie breakers are a personal preference

First encountered, first merged	The one you found first
Highest quality base calls	Use sequence with highest quality
Highest coverage	Whichever results in more coverage
Look ahead	Do both and evaluate consequences
Exclude	Be petty and don't merge them (separate contigs)

Being greedy makes genome assembly tractable

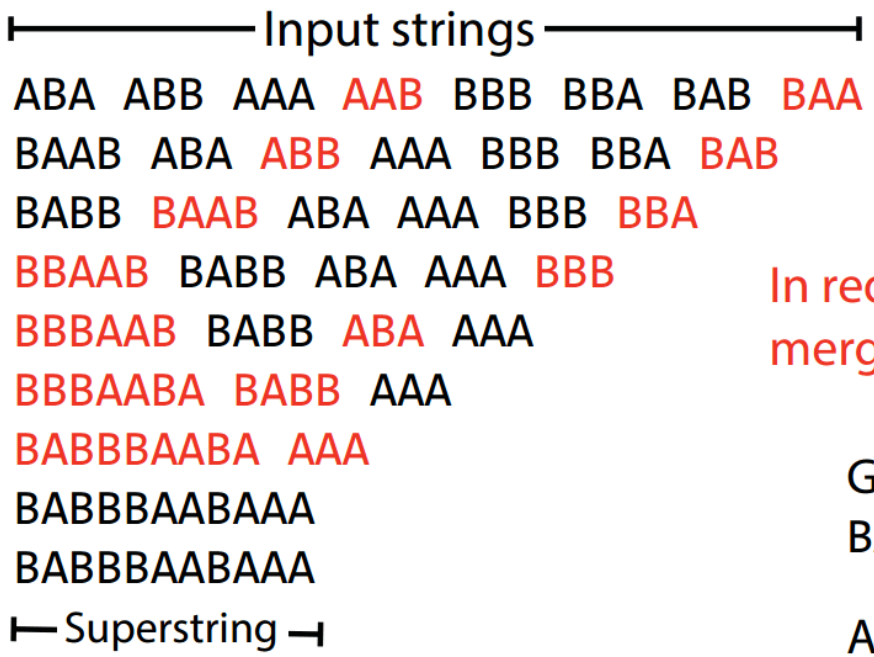


Rounds of merging, one merge per line.
Number in first column = length of overlap merged before that round.

Let's get some practice being greedy

ABA ABB AAA AAB BBB BBA BAB BAA

For ties, choose the one you found first



In red are strings that get merged before the next round

Greedy answer:
BABBBAABAAA

Actual SCS:
AAABBBABAA

Repeats ruin our assembly

Let's take a string, and cyclically permute it with k = 6

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
a_long_long_time
```

We are missing a "_long". **Why?**

Longer reads and genome assembly

k = 8

a_long_long_long_time

```
long_lon ng_long_ _long_lo g_long_t ong_long g_long_l ong_time a_long_l _long_ti long_tim
long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l _long_ti
_long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l
_long_time a_long_lo long_lon ng_long_ g_long_t ong_long g_long_l
_long_time ong_long_ a_long_lo long_lon g_long_t g_long_l
g_long_time ong_long_ a_long_lo long_lon g_long_l
g_long_time ong_long_ a_long_lon g_long_l
g_long_time ong_long_l a_long_lon
g_long_time a_long_long_l
a_long_long_long_time
a_long_long_long_time
```

We get the correct string back, but how did increasing our k fix this?

By having one read span all three "long"s, we prevented a collapse

a_long_long_long_time

g_long_l

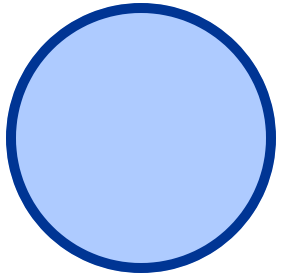
Greedy assembly is not used in practice

It just helps us understand our problem

After today, you should be able to

1. Explain the fundamental challenge of reconstructing a complete genome.
2. Describe and apply the principles of the greedy algorithm.
3. **Understand and construct de Bruijn graphs.**

Graphs is a data structure for drawing relationships between items



Node

Represents a single entity

- Person
- Location
- Protein
- Sequencing read

Edge

Represents a connection (possibly with a direction)

- Instagram follower
- Flights
- Protein-protein interaction
- Sequence overlap

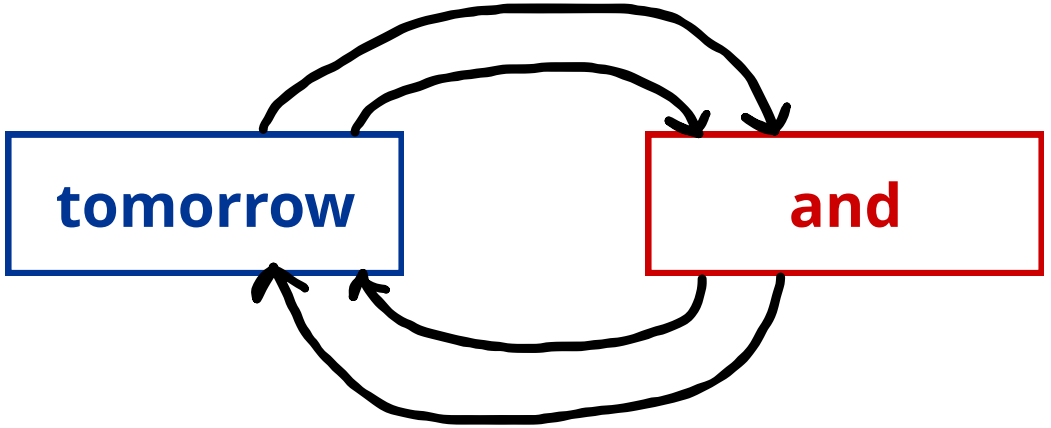
Genome assembly uses direct edges to specify overlap and concatenation

Let's build a **directed multigraph:**

"tomorrow and tomorrow and tomorrow"

1. Each unique k-mer is a node
2. Add directed edges for each overlap and concatenation

K-mer is a substring of length k



(We will cheat here and write down just unique words)

Building k-mers from a string

Spectrum with k = 3

1. Slice first k characters
2. Shift right one character
3. Repeat

```
GGCGATTCATCG

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

All 3-mers

Build a De Bruijn graph with k-1 nodes

5' **AATGGCGTA** **3'**

Step 1: Build k-mers

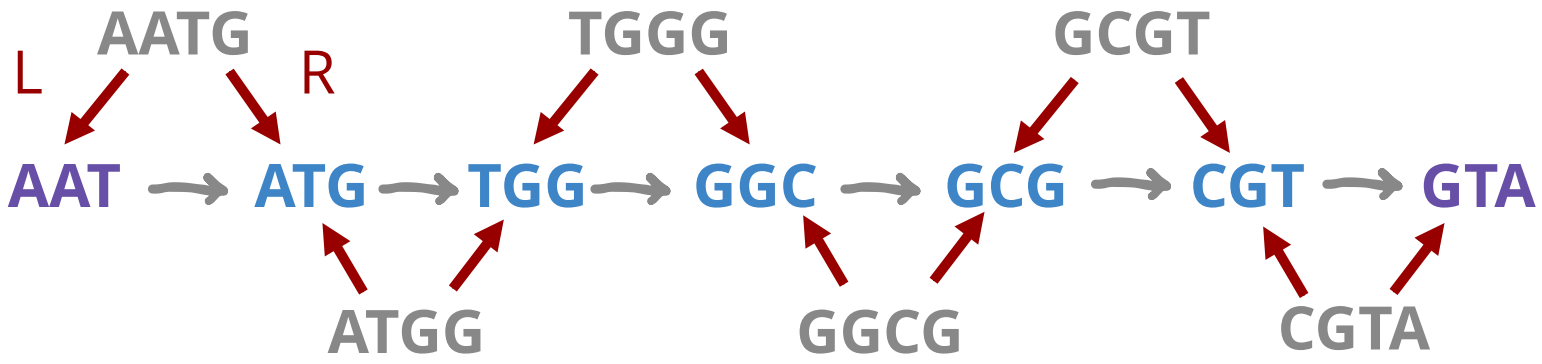
Let's use k = 4

AATG ATGG TGGG

GGCG GCGT CGTA

Step 2: Take left and right k-1 mer and make two connected nodes

Repeat



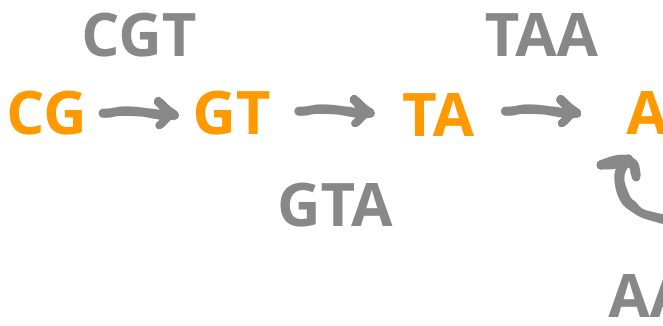
Semi-balanced has difference of 1 ← **Graph is Eulerian if it contains <= 2 of these**

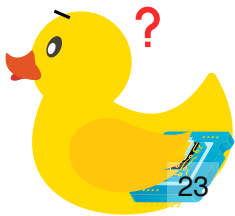
A node is balanced if indegree equals outdegree

De Bruijn practice

Build a De Bruijn graph with k = 3

CGTAAAT



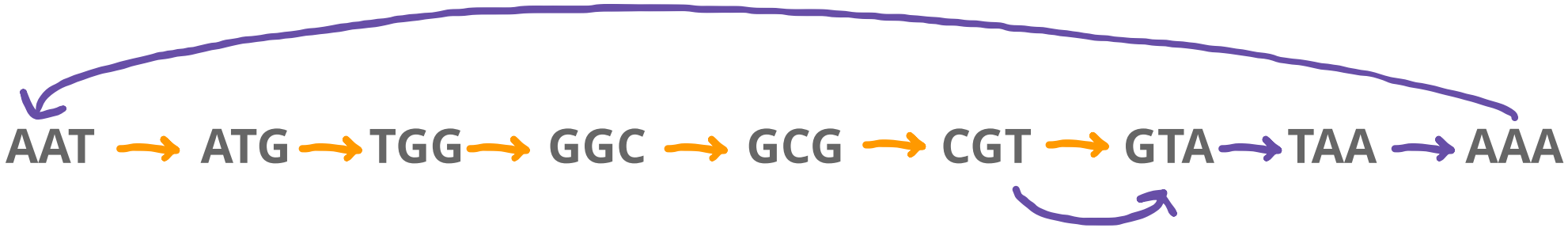


De Bruijn graphs with multiple reads

Read 1
5' AATGGCGTA 3'

Let's use k = 4

Read 2
5' CGTAAAT 3'



Wait, what happend? This is not Eulerian

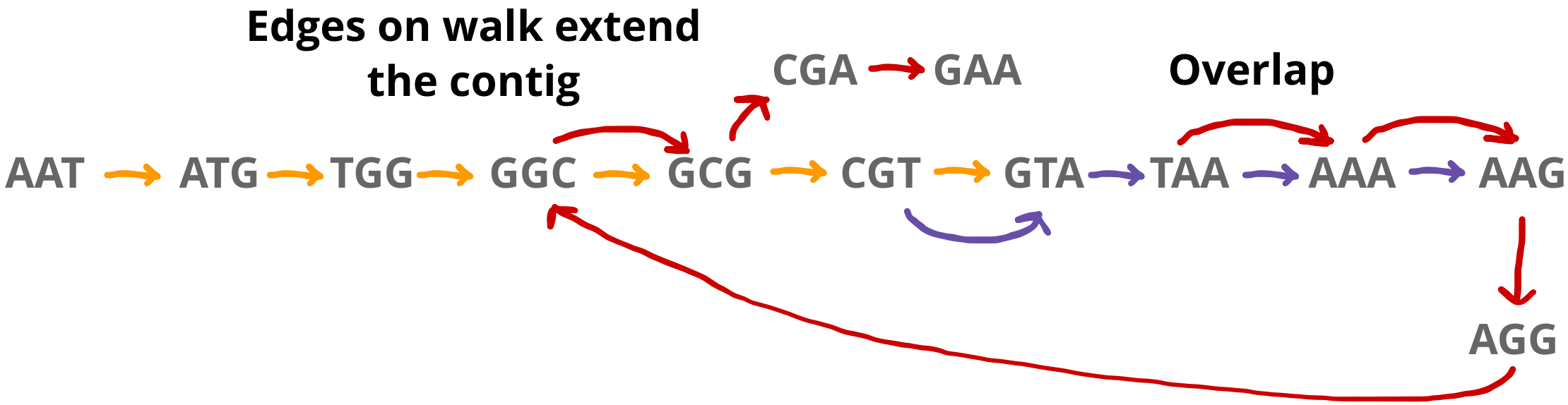
Circular genomes are not Eulerian

Redo, but make it not circular

Read 1
5' AATGGCGTA 3'

Read 2
5' CGTAAAG 3'

Read 3
5' TAAAGGCGAA3'



Why is this not Eulerian?

More than two semi-balanced nodes

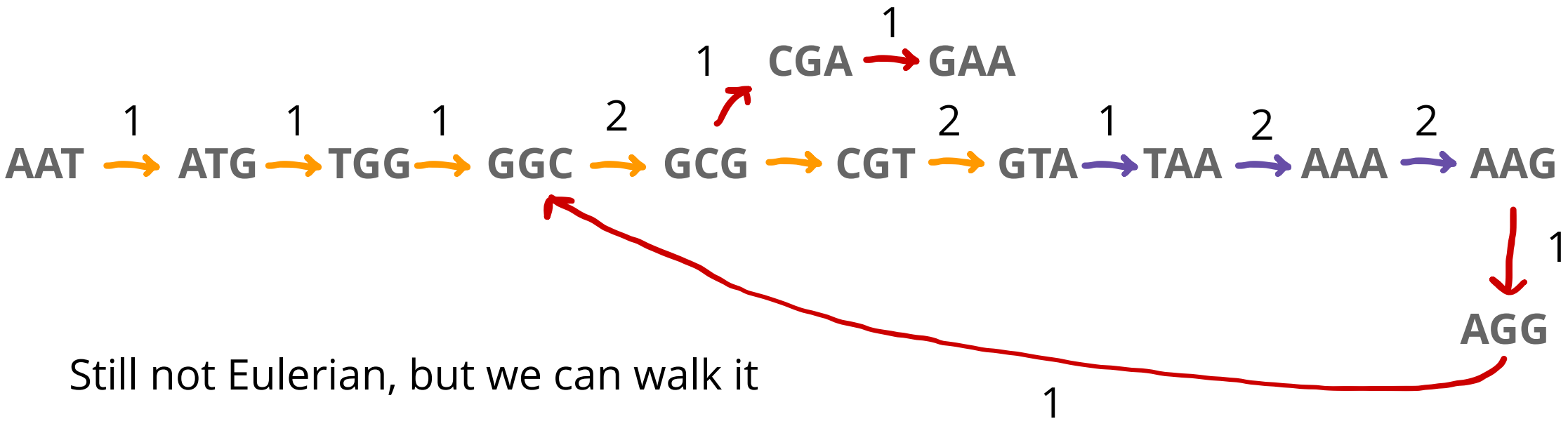
Cannot walk along each edge once

We can add weights to edges

Read 1
5' AATGGCGTA 3'

Read 2
5' CGTAAAG 3'

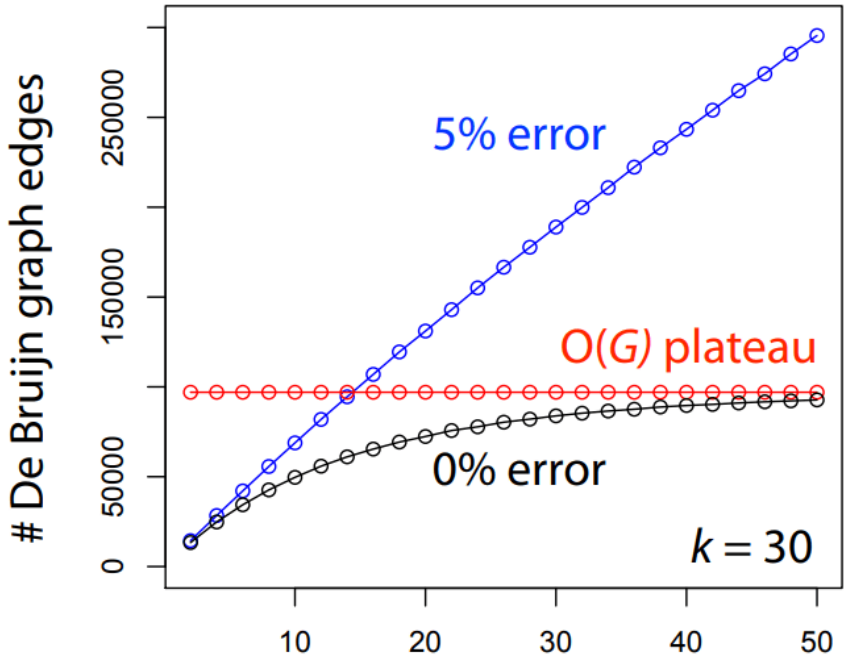
Read 3
5' TAAAGGCGAA3'

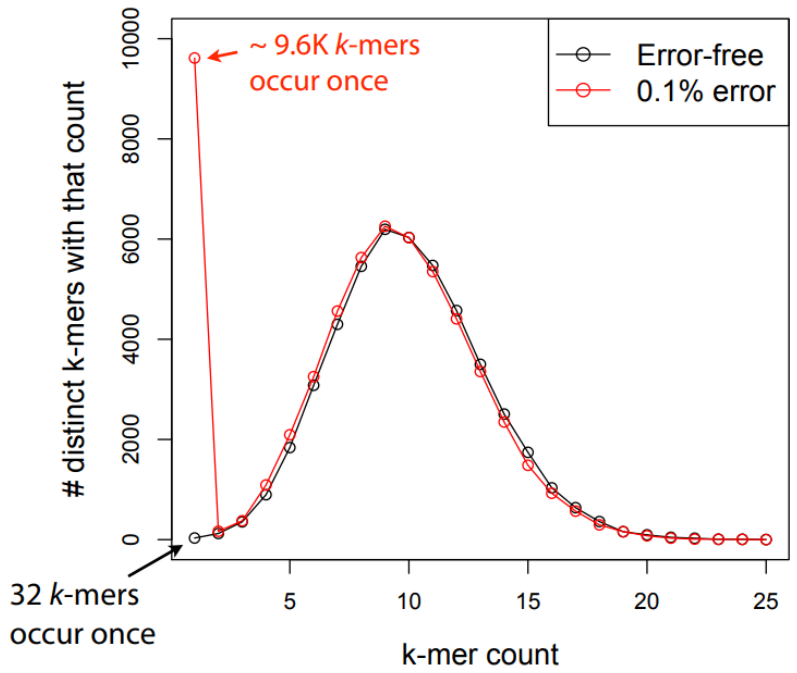


Still not Eulerian, but we can walk it

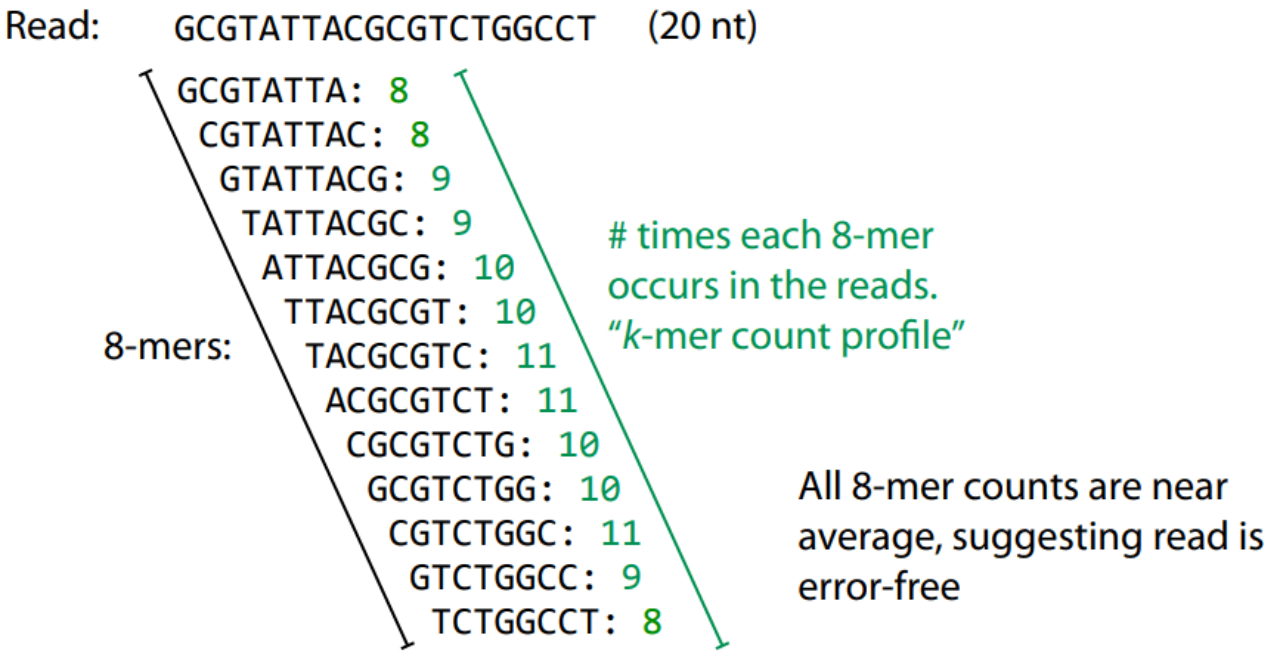
If there was no overlap, then we would have some unconnected graphs

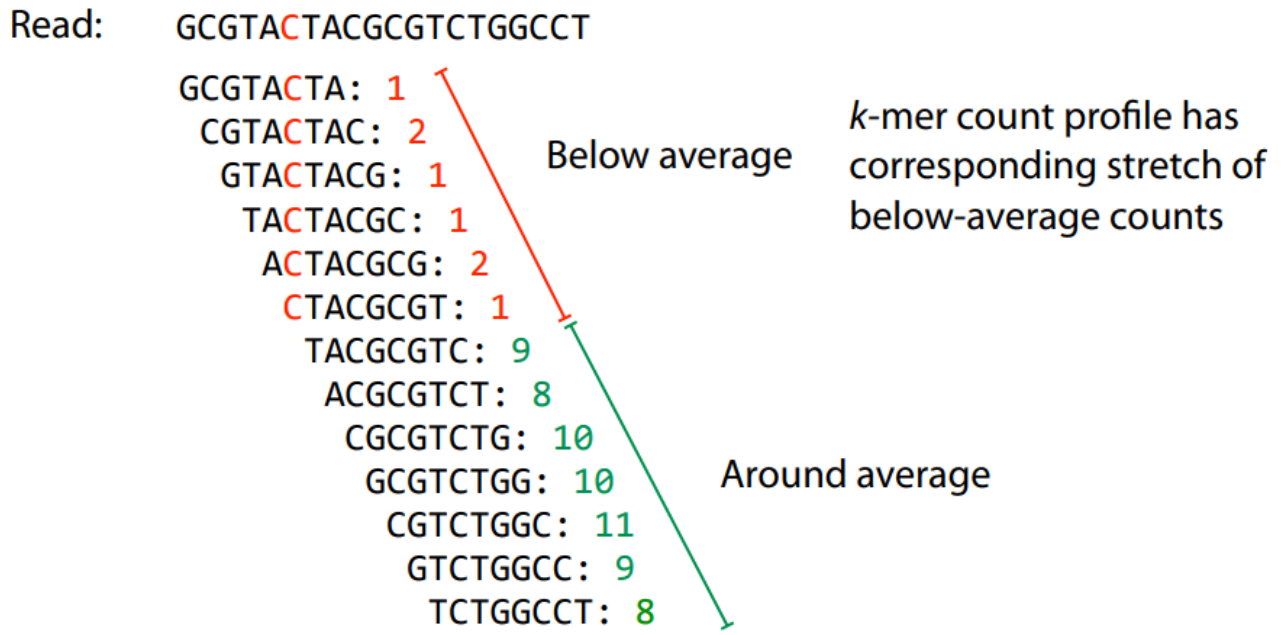
Errors dramatically increase the number of edges and unconnected graphs





Errors affect k-mer counts





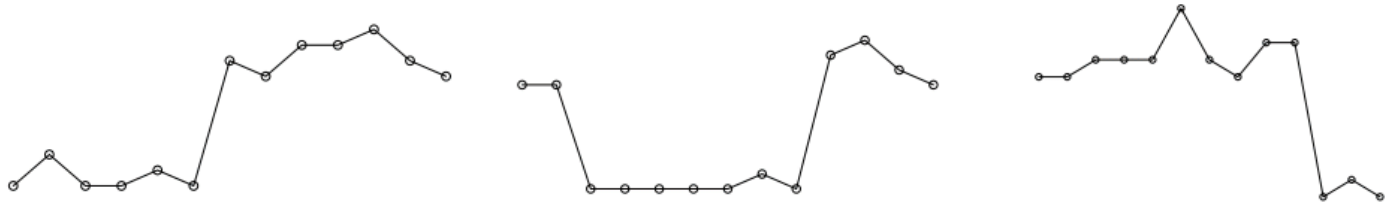
Error correction

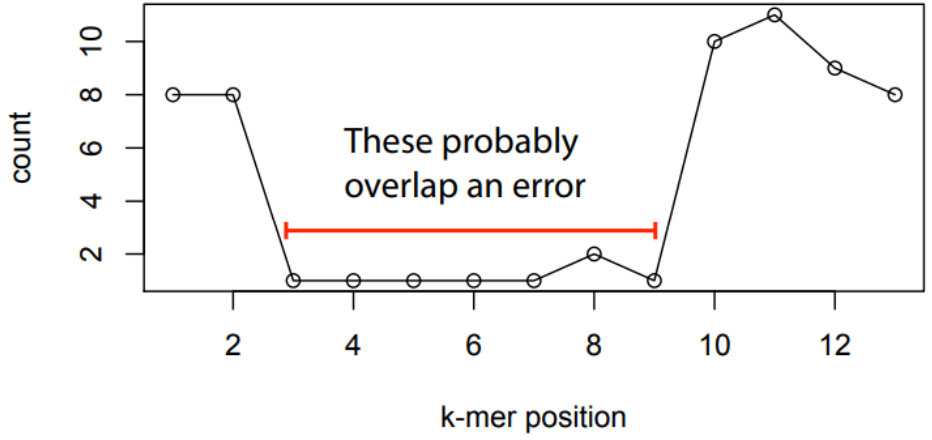
k-mer counts when errors are in different parts of the read:

```
GCGTACTACGCGTCTGGCCT
GCGTACTA: 1
 CGTACTAC: 3
  GTACTACG: 1
   TACTACGC: 1
    ACTACGCG: 2
     CTACGCGT: 1
      TACGCGTC: 9
       ACGCGTCT: 8
        CGCGTCTG: 10
         GCGTCTGG: 10
          CGTCTGGC: 11
           GTCTGGCC: 9
            TCTGGCCT: 8
```

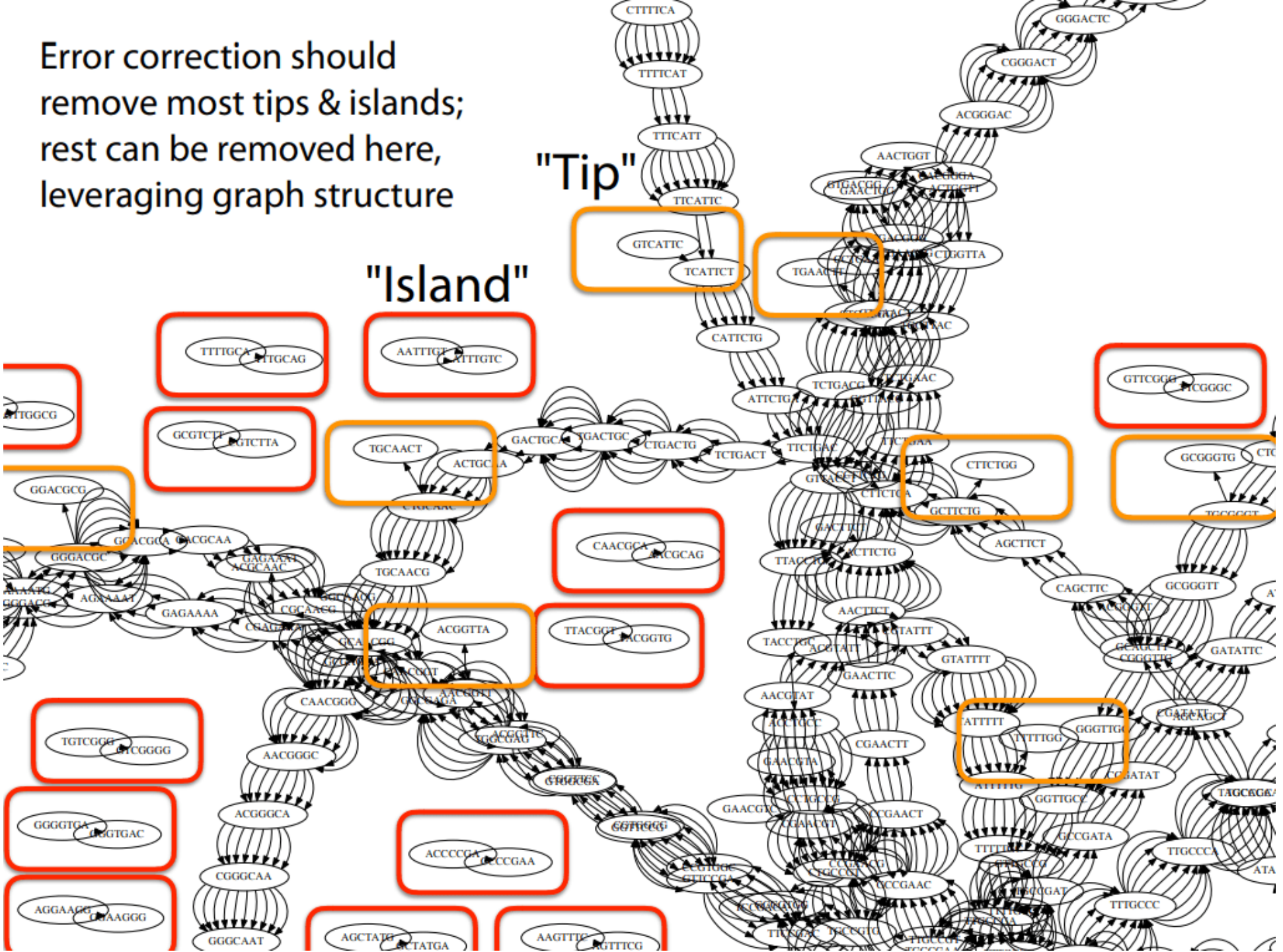
```
GCGTATTACACGTCTGGCCT
GCGTATTA: 8
 CGTATTAC: 8
  GTATTACA: 1
   TATTACAC: 1
    ATTACACG: 1
     TTACACGT: 1
      TACACGTC: 1
       ACACGTCT: 2
        CACGTCTG: 1
         ACGTCTGG: 1
          CGTCTGGC: 11
           GTCTGGCC: 9
            TCTGGCCT: 8
```

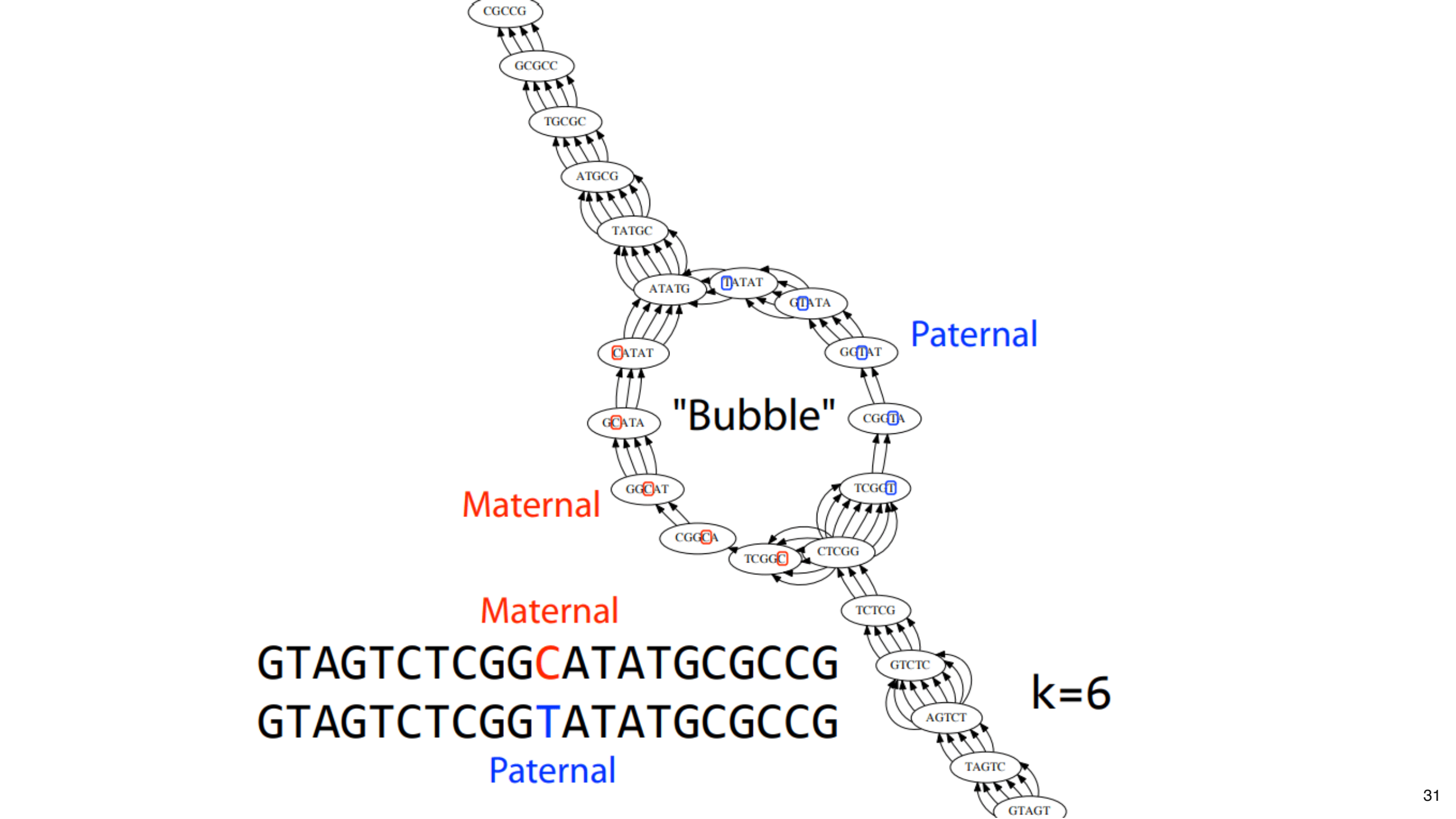
```
GCGTATTACGCGTCTGGTCT
GCGTATTA: 8
 CGTATTAC: 8
  GTATTACG: 9
   TATTACGC: 9
    ATTACGCG: 9
     TTACGCGT: 12
      TACGCGTC: 9
       ACGCGTCT: 8
        CGCGTCTG: 10
         GCGTCTGG: 10
          CGTCTGGT: 1
           GTCTGGTC: 2
            TCTGGTCT: 1
```





Error correction should remove most tips & islands; rest can be removed here, leveraging graph structure




CGCCG
GCGCC
TGCGC
ATGCG
TATGC
ATATG
TATAT
CATAT
GCATA
GGCAT
CGGCA
TCGGC
CTCGG
TCGCT
CGCTA
GCTAT
"Bubble"
Paternal
Maternal
Maternal
GTAGTCTCGGCATATGCGCCG
GTAGTCTCGGTATATGCGCCG
Paternal
TCTCG
GTCTC
AGTCT
TAGTC
GTAGT
k=6

Before the next class, you should

Lecture 04:
De novo assembly

Lecture 05:
Gene annotation

Today

Tuesday

- Start Assignment 02, which is due Thursday at 11:59 pm.