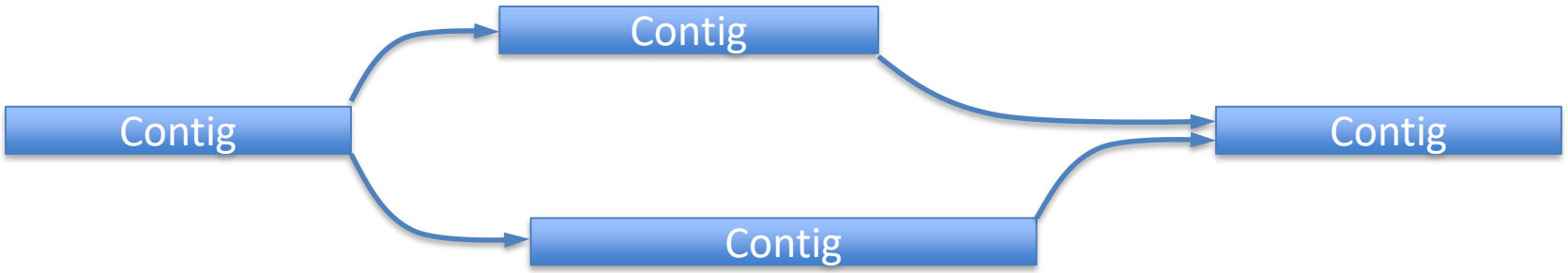
# CS342: Bioinformatics Assembling a Genome

# Overlap Layout Consensus

1. **Overlap** – Build the overlap graph
2. **Layout** – Bundle stretches of the overlap graph into ***contigs*.**
3. **Consensus** – Pick the most likely nucleotide sequence for each contig.



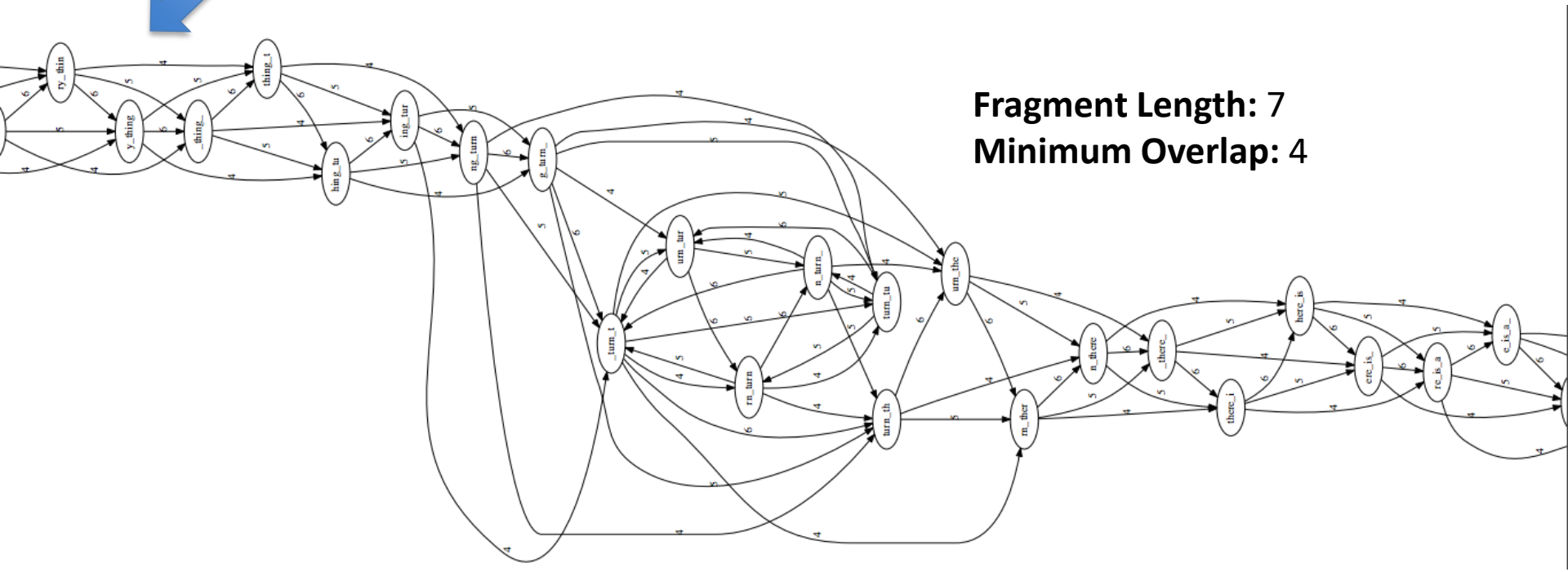

Sometimes additional information can be used to begin to ***scaffold*** or order together contigs.

# **Overlap** Layout Consensus

Overlap graph is big and messy!!!  Contigs don't just pop out.

Part of the overlap graph for:

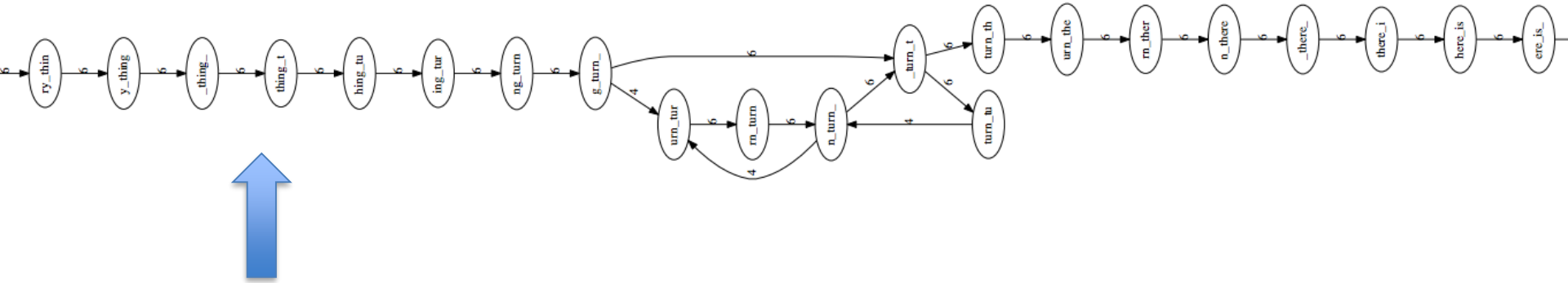
`To_every_thing_turn_turn_turn_there_is_a_season`



**Fragment Length:** 7
**Minimum Overlap:** 4

# Overlap **Layout** Consensus

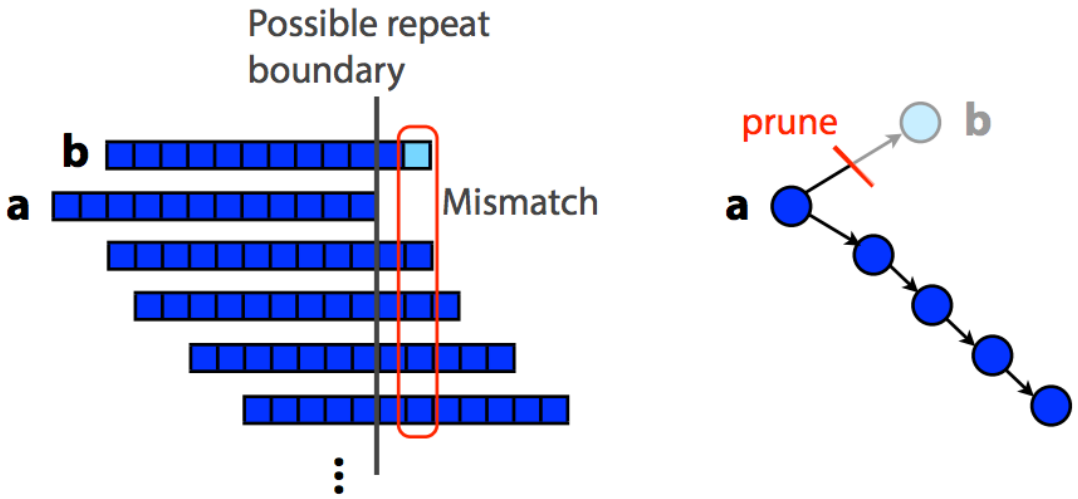
You can remove **redundant** information (edges) from the graph.




Same graph after simplification

# Overlap **Layout** Consensus

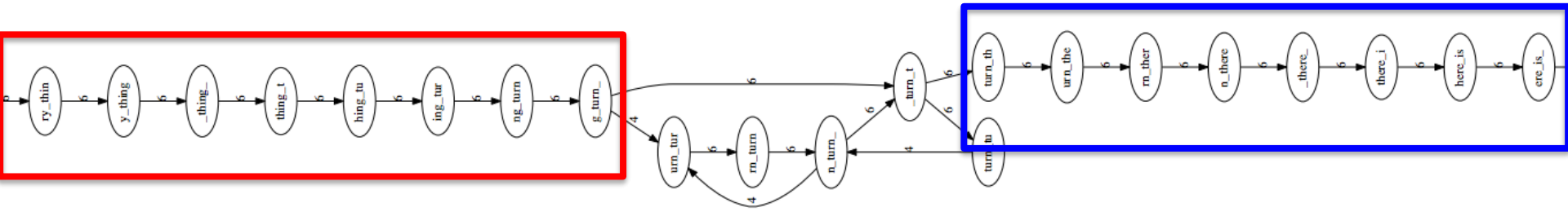
The layout step can also deal with problematic subgraphs, likely the result of errors.




Mismatch could be due to sequencing error or repeat. Since the path through **b** ends abruptly we might conclude it's an error and prune **b**.

# Overlap **Layout** Consensus

**Contigs** correspond to non-branching stretches in the simplified graph




Contig 1:
`to_every_thing_turn_`

Unresolvable repeat!

Contig 2:
`turn_there_is_a_season`

# Overlap Layout **Consensus**

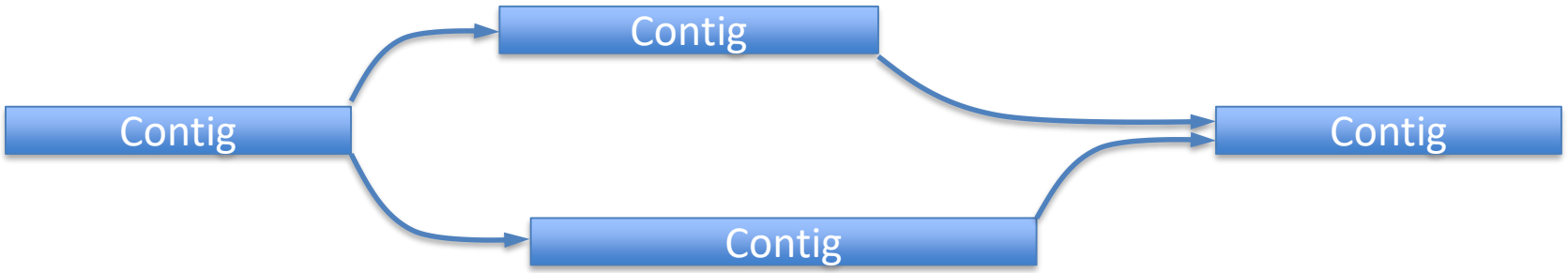

At each position, ask: what nucleotide (and/or gap) is here?

Complications: (a) sequencing error, (b) ploidy

Say the true genotype is AG, but we have a high sequencing error rate and only about 6 reads covering the position.

# Overlap Layout Consensus

1. **Overlap** – Build the overlap graph
2. **Layout** – Bundle stretches of the overlap graph into ***contigs*.**
3. **Consensus** – Pick the most likely nucleotide sequence for each contig.




Sometimes additional information can be used to begin to ***scaffold*** or order together contigs.

# De Bruijn Graph Assembly

A formulation conceptually similar to overlapping/SCS, but has some potentially helpful properties not shared by SCS.

# The graph of a sequence

For the moment let's imagine that reads are like $k$-mers from a sequence, as they do tend to be uniform in length.

```
GACGGCGGCGCACGGCGCAA    - Our toy sequence
GACGG
 ACGGC
  CGGCG
   GGCGG
    GCGGC
     CGGCG
      GGCGC    - The complete set of 16 5-mers
       GCGCA
        CGCAC
         GCACG
          CACGG
           ACGGC
            CGGCG
             GGCGC
             GCGCA
               CGCAA
```

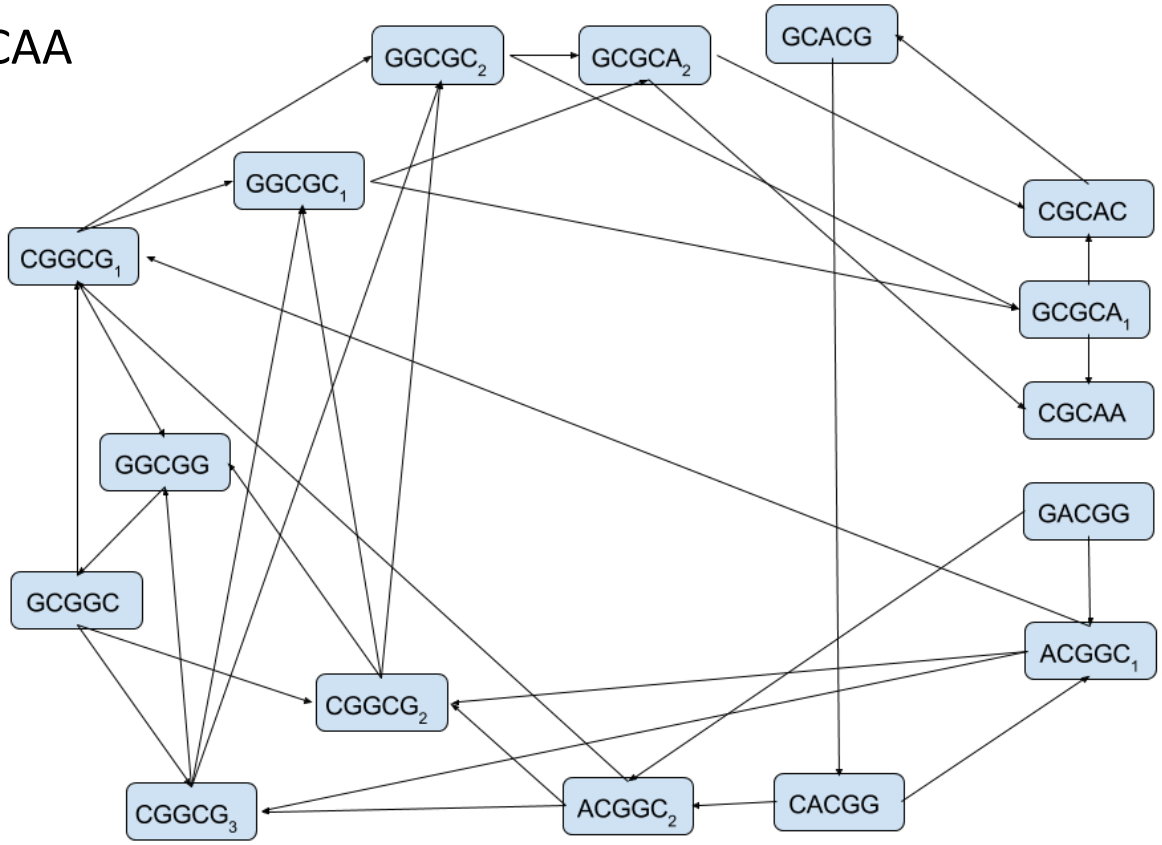
Now we can construct a graph where:

- Each $5$-mer is a node
- There is a directed edge from a $k$-mer that shares its $(k-1)$-base suffix with the $(k-1)$-base prefix of another $k$-mer

# A read-overlap graph

The read-overlap graph for the 5-mers from:

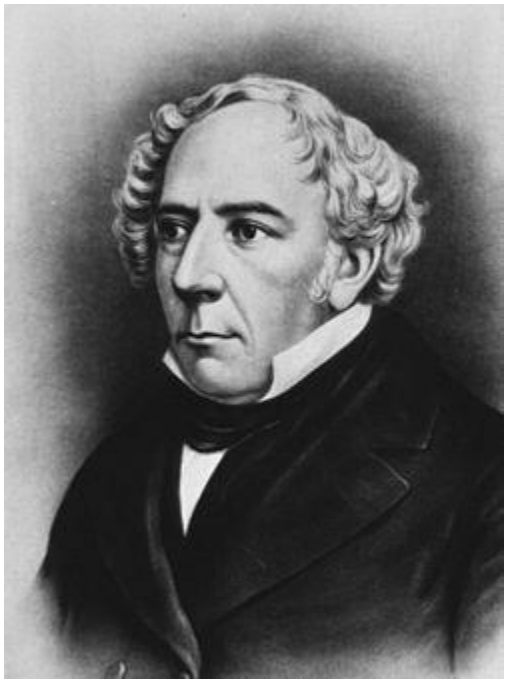
GACGGCGGCGCACGGCGCAA



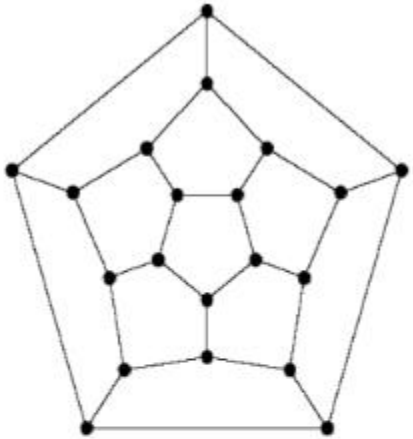

The problem is ***How to infer the original sequence from this graph?***

# The rules of our game

- Every node, $k$-mer, can be used exactly once
- The object is to find a path along edges that visits every **node (vertex)** one time
- This game was invented in the mid 1800's by a mathematician called Sir William Hamilton



A version of Hamilton's game:

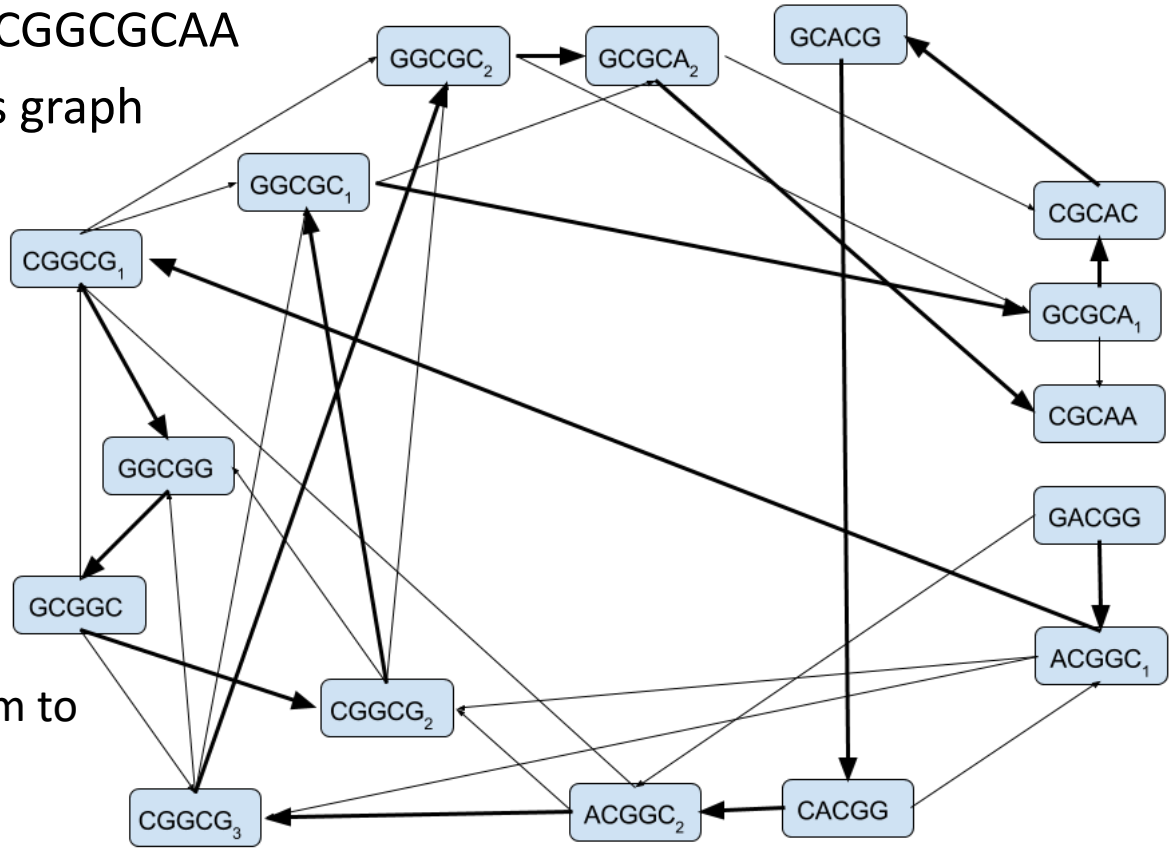


# Finding a Hamiltonian Path in a graph

Our desired sequence:

GACGGCGGCGCACGGCGCAA

is indeed a path in this graph



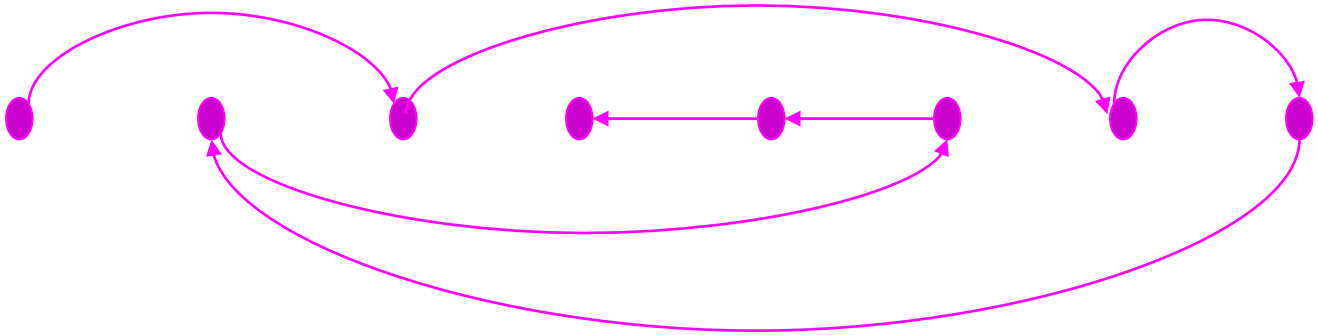

How can we write a program to solve Hamilton's puzzle?

Is the solution unique?

# Example Problem: Hamiltonian Path Approach

$S$ = { ATG  AGG  TGC  TCC  GTC  GGT  GCA  CAG }

ATG  AGG  TGC  TCC  GTC  GGT  GCA  CAG



ATG CAG G TCC

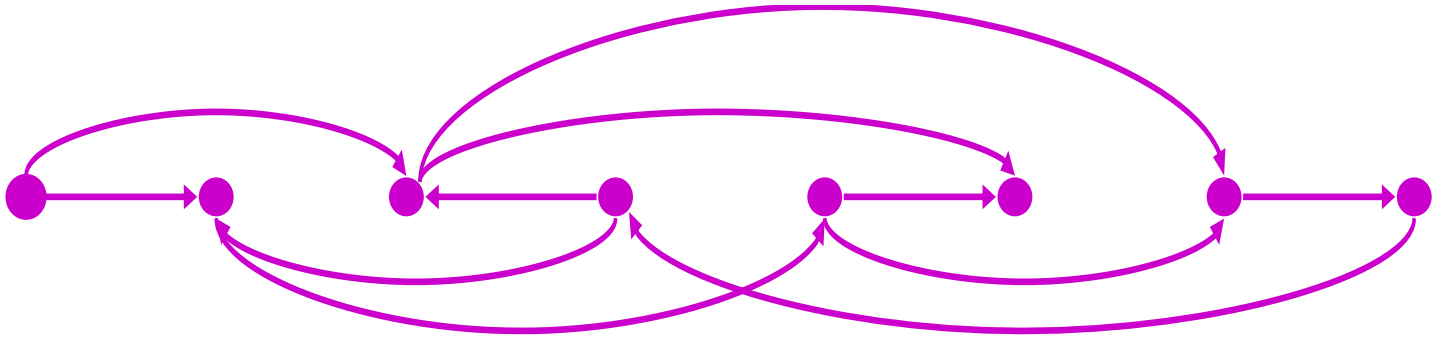
Path visited every VERTEX once

# More Complicated Example: Hamiltonian Path Approach

A more complicated graph:

*S* = { ATG TGG TGC GTG GGC GCA GCG CGT }

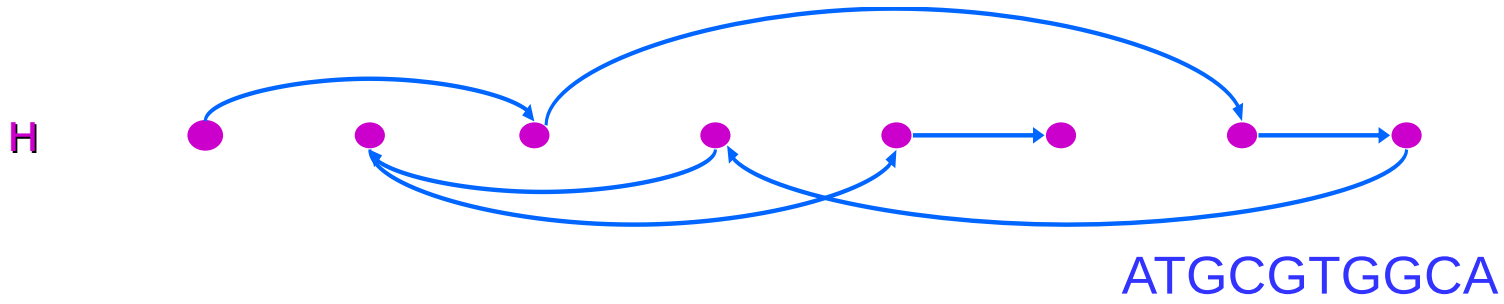
H



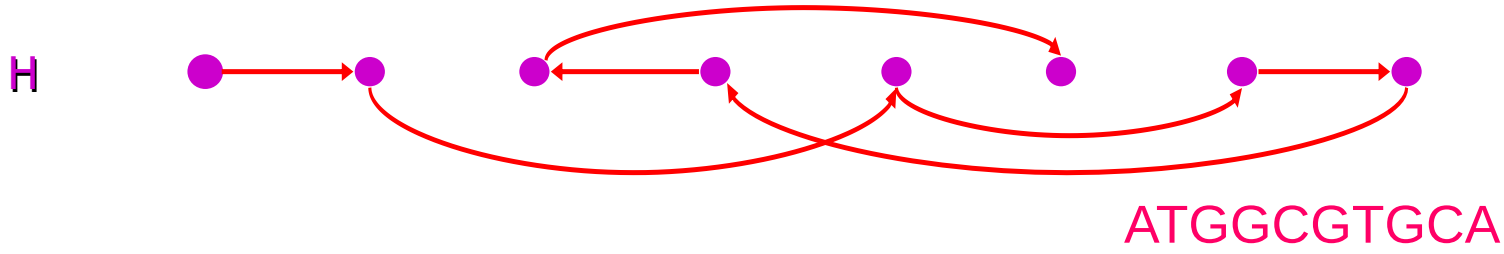
# More Complicated Example: Hamiltonian Path Approach

*S* = { ATG  TGG  TGC  GTG  GGC  GCA  GCG  CGT }

Path 1:




Path 2:



# Another way to represent our $k$-mers in a graph

- Rather than making each $k$-mer a node, let's try making them an edge
- That seems odd, but it is related to the overlap idea

The 5-mer

GACGG

has a prefix

GACG

and a suffix

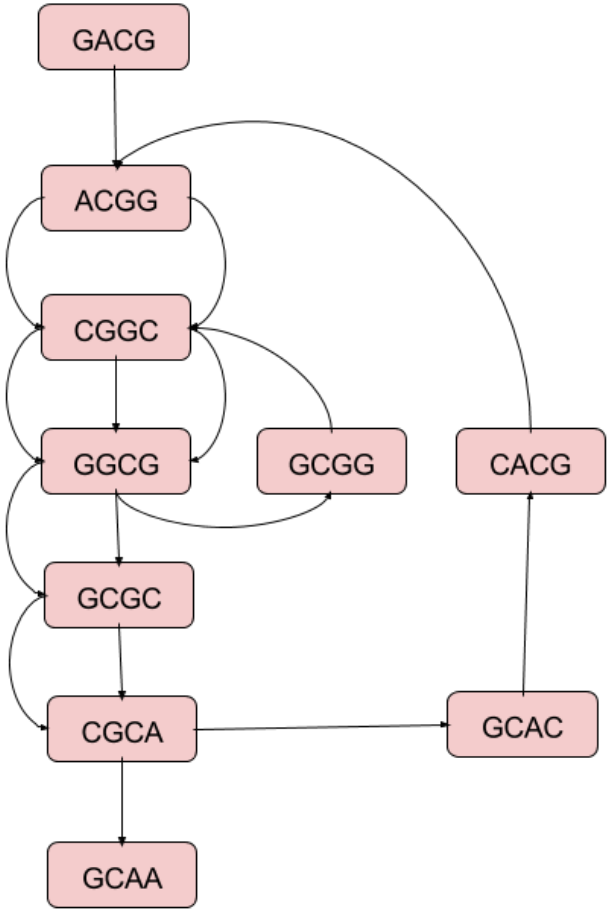
ACGG


- Think of the k-mer as the edge connecting a prefix to a suffix
- This leads to a series of simple graphs
- Then combine all nodes with the same label

# A De Bruijn Graph

This rather odd graph is called the "De Bruijn" graph; was named after a famous mathematician.

The problem is ***How to infer the original sequence from this graph?***

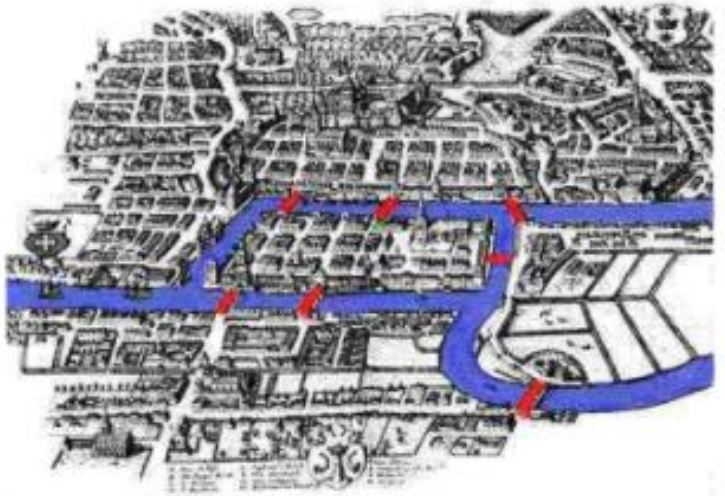


# The rules of our new game

- Every edge, $k$-mer, can be used exactly once
- The object is to find a path in the graph that uses each edge only one time
- This game was invented in the late 1700's by a mathematician called Leonhard Euler

Leonhard Euler

A version of Euler's game:



Bridges of Königsberg

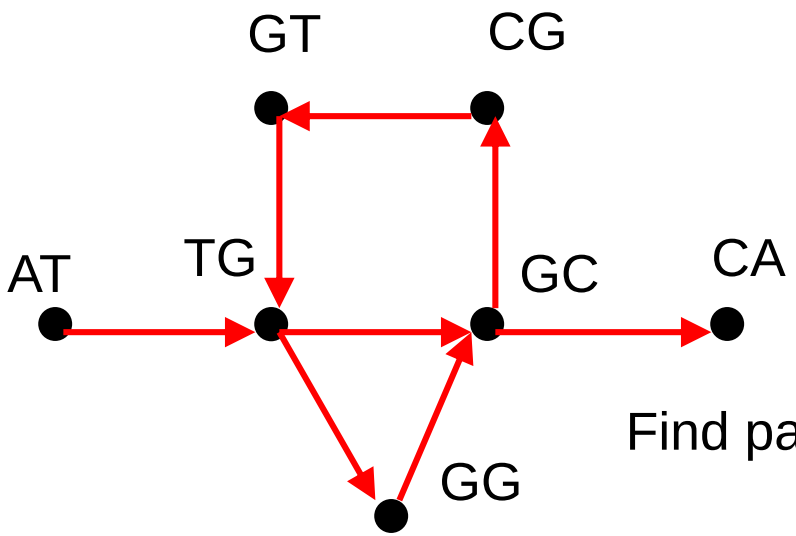
Find a city tour that crosses every bridge just once

# Example Problem: Eulerian Path Approach

*S* = { ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT }

Vertices correspond to $( k - 1 )$ – mers : { AT, TG, GC, GG, GT, CA, CG }

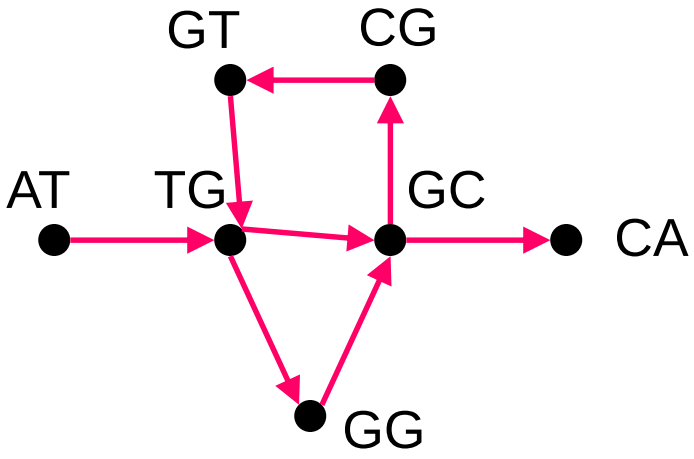
Edges correspond to *k* – mers from *S*



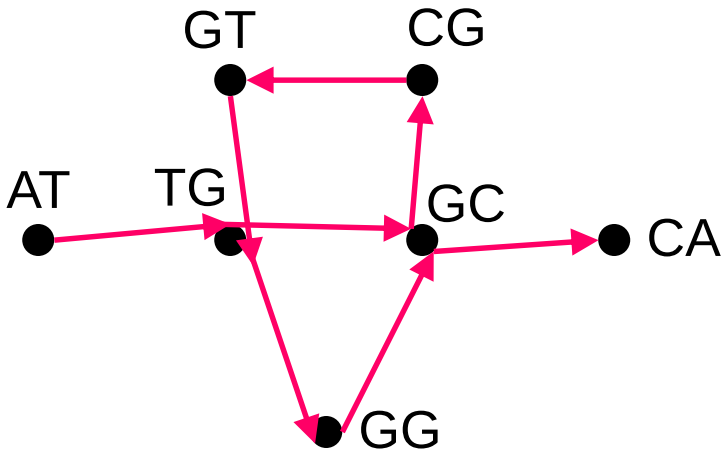

Find path that visits every **EDGE** once

# Example Problem: Eulerian Path Approach

*S* = { AT, TG, GC, GG, GT, CA, CG } corresponds to two different paths:




ATGGCGTGCA




ATGCGTGGCA