582670 Algorithms for Bioinformatics

Lecture 3: Graph Algorithms for Genome Assembly

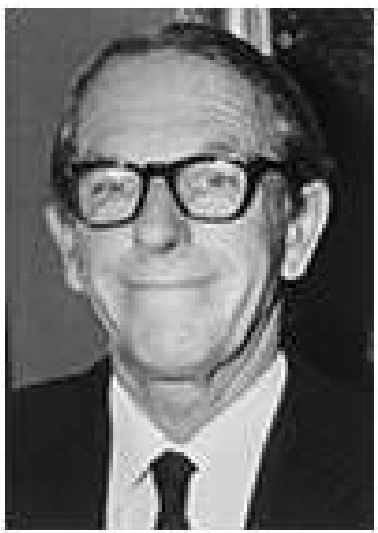
17.09.2015

Adapted from slides by Alexandru Tomescu, Leena Salmela and Veli Mäkinen, which are partly from http://bix.ucsd.edu/bioalgorithms/slides.php

DNA Sequencing: History

Sanger method (1977):

- Labeled ddNTPs terminate DNA copying at random points.



Gilbert method (1977):

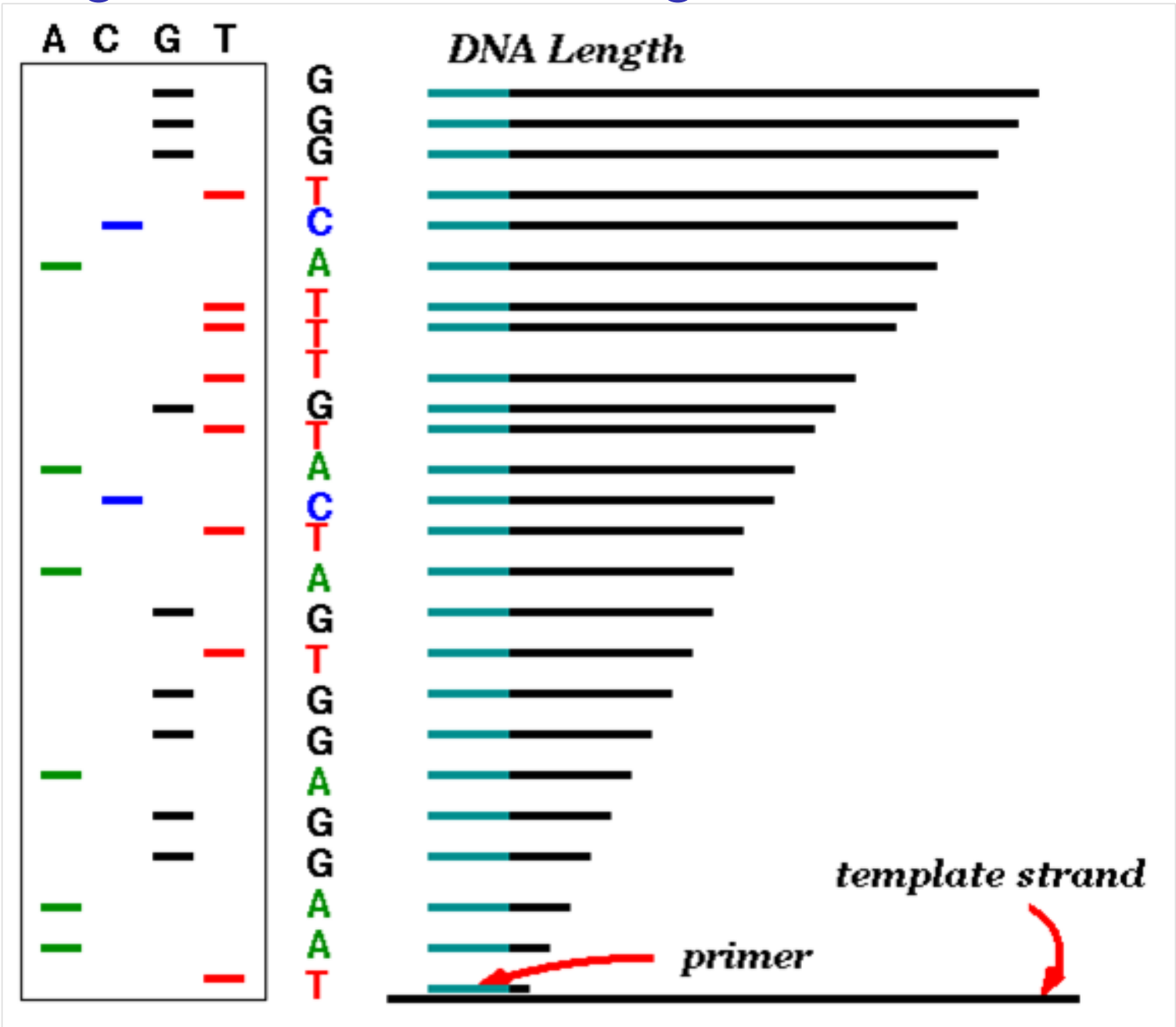
- Chemical method to cleave DNA at specific points (G, G+A, T+C, C).

- Both methods generate labeled fragments of varying lengths that are further measured by electrophoresis.

Sanger Method: Generating a Read

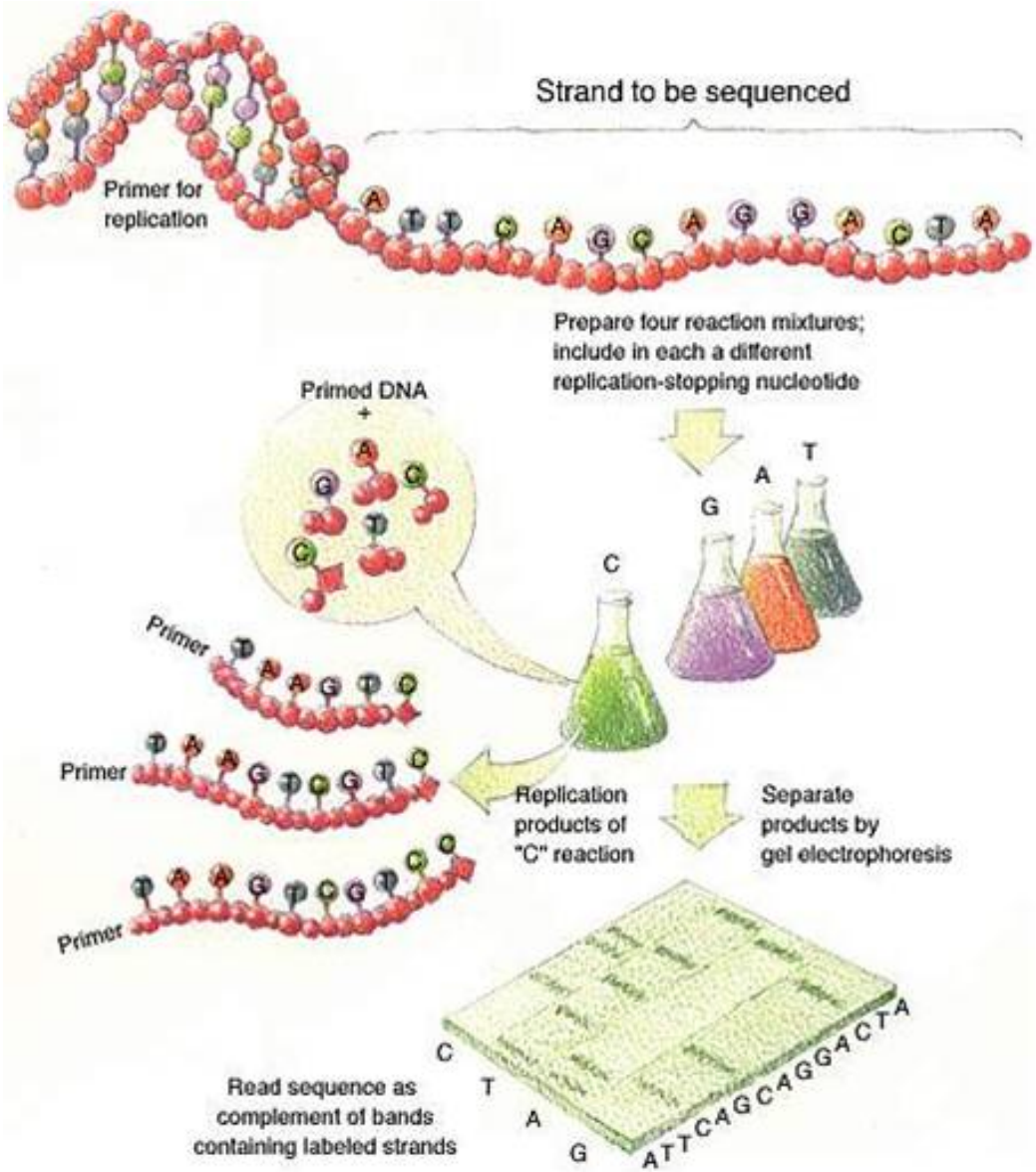
1. Divide DNA sample into four.
2. Each sample will have available all normal nucleotides and modified nucleotides of one type (A, C, G or T) that will terminate DNA strand elongation.
3. Start at primer (restriction site).
4. Grow DNA chain.
5. In each sample the reaction will stop at all points ending with the modified nucleotide.
6. Separate products by length using gel electrophoresis.

Sanger Method: Generating a Read



DNA Sequencing

- Shear DNA into millions of small fragments.
- Read 500-700 nucleotides at a time from the fragments (Sanger method)



Fragment Assembly

- **Computational Challenge:** assemble individual reads into the full genomic sequence
- Until late 1990s the fragment assembly of human genome was viewed as computationally too difficult
- For small and "easy" genomes, such as bacterial genomes, fragment assembly is easy with many software tools
- Remains to be difficult problem for more complex genomes

Shortest Superstring Problem

- Problem: Given a set of strings, find a shortest string that contains all of them
- Input: Strings $S = \{s_1, s_2, \ldots, s_n\}$
- Output: A string s that contains all string $s_1, s_2, \ldots s_n$ as substrings, such that the lenght of s is minimized

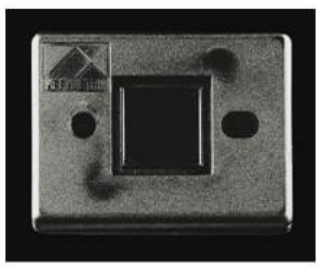
- Models the fragment assembly problem
  - Input strings are the reads, output is the genome
- One of study group topics

Sequencing by Hybridization (SBH): History

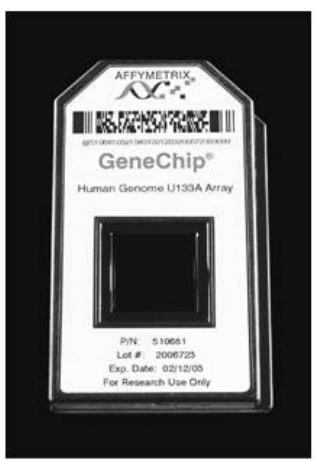
- 1988: SBH suggested as an alternative sequencing method. Nobody believed it will ever work.
- 1991: Light directed polymer synthesis developed by Steve Fodor and colleagues.
- 1994: Affymetrix develops first 64-kb DNA microarray.

First microarray prototype (1989)

First commercial DNA microarray prototype with 16,000 features (1994)



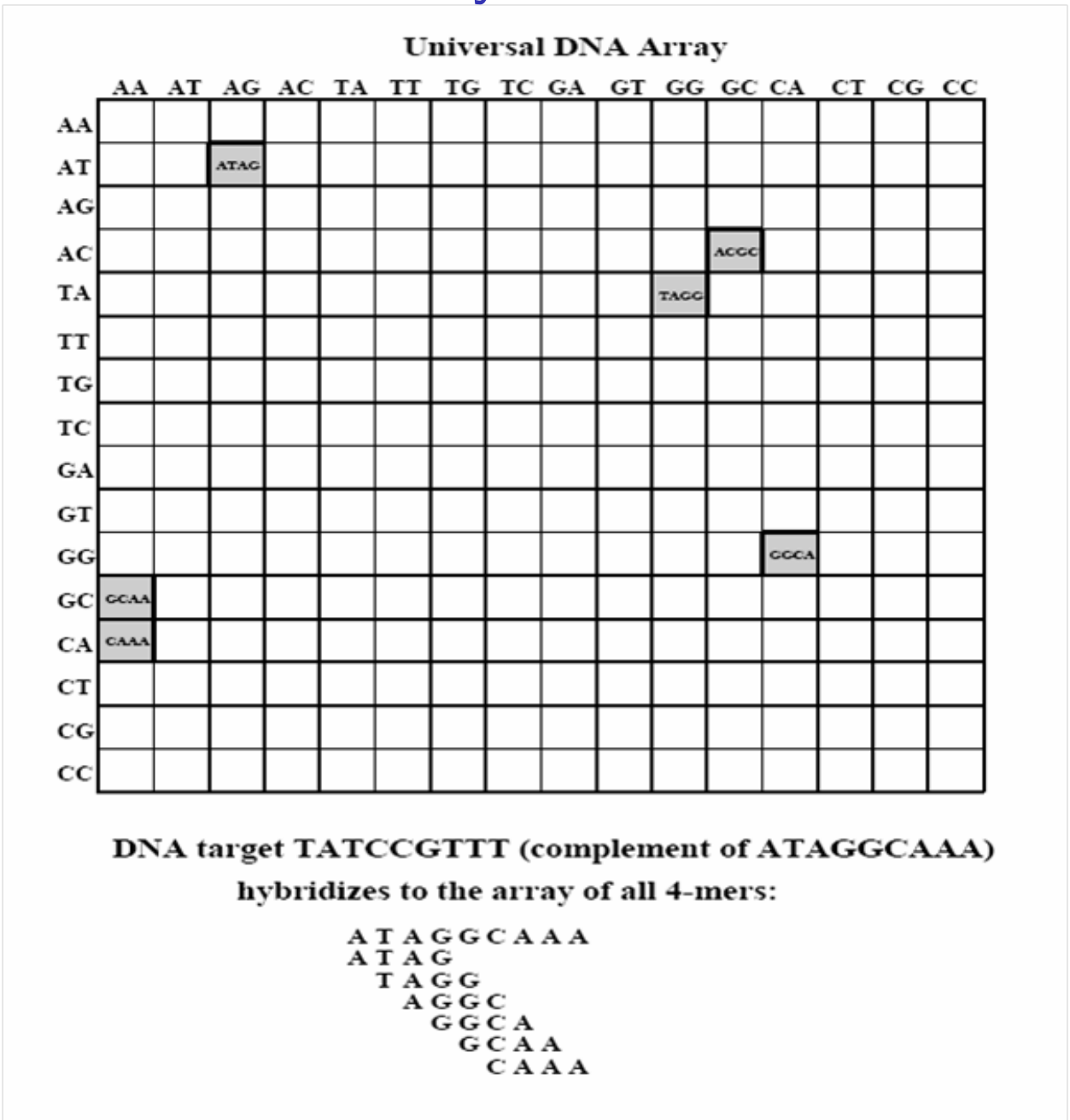
500,000 features per chip (2002)



How SBH works

- Attach all possible DNA probes of length k to a flat surface, each probe at a distinct and known location. This set of probes is called the *DNA microarray*.
- Apply a solution containing fluorescently labeled DNA fragment to the array.
- The DNA fragment hybridizes with those probes that are complementary to substrings of length k of the fragment.
- Using a spectroscopic detector, determine which probes hybridize to the DNA fragment to obtain the k-mer composition of the DNA fragment.
- Reconstruct the sequence of the DNA fragment from the k-mer composition.

Hybridization on DNA Array



k-mer composition

- $\text{Composition}_k(\mathit{Text})$ is a multiset of all $(n-k+1)$ k-mers in a string Text of length n.
- For example

$$\text{Composition}_3(\text{TATGGTGC}) = \{\text{ATG}, \text{GGT}, \text{GTG}, \text{TAT}, \text{TGC}, \text{TGG}\}$$

- Different sequences may have the same composition:

$$\text{Composition}_2(\text{GTATCT}) =$$
$$\text{Composition}_2(\text{GTCTAT}) =$$
$$\{\text{AT}, \text{CT}, \text{GT}, \text{TA}, \text{TC}\}$$

String Composition Problem

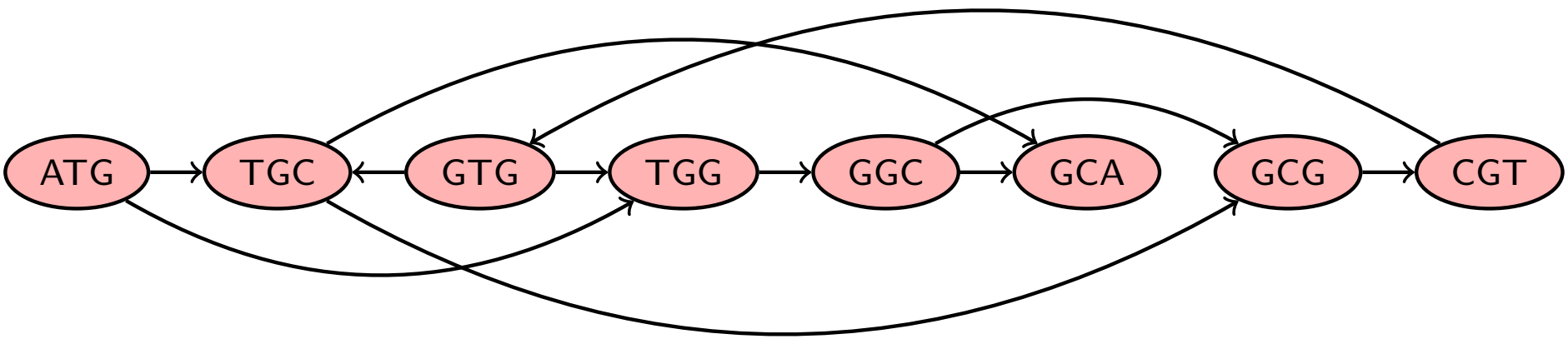
- Goal: Generate the k-mer composition of a string.
- Input: A string *Text* and an integer k
- Output: $\mathrm{Composition}_k(\mathit{Text})$

String Reconstruction Problem

- Goal: Reconstruct a string from its k-mer composition.
- Input: An integer k and a collection *Patterns* of k-mers
- Output: A string *Text* with k-mer composition *Patterns* (if such a string exists)

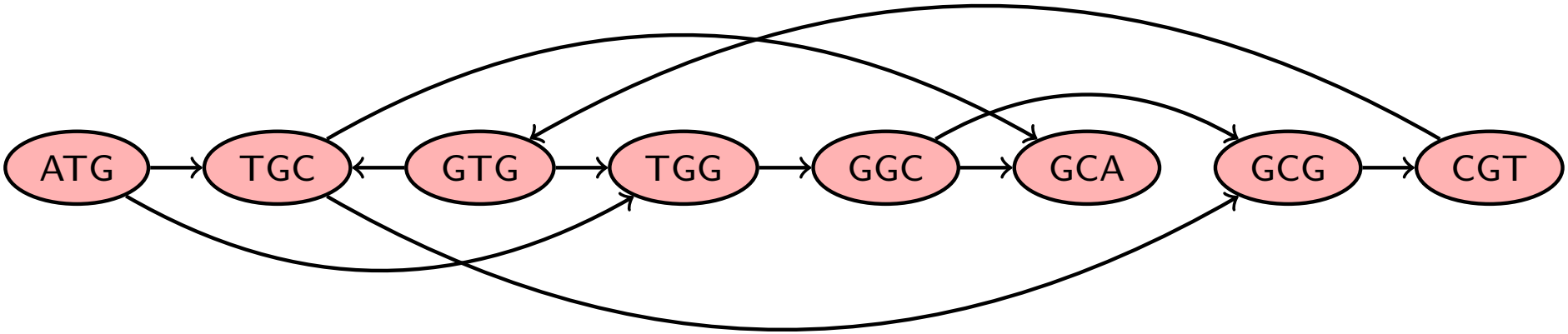
Overlap Graph

- The overlap graph $\mathrm{Overlap}(\mathit{Patterns})$ is a *directed graph* that contains
  - One node for each k-mer in *Patterns*
  - An edge from $\mathit{Pattern}_i$ to $\mathit{Pattern}_j$ whenever the same $(k-1)$-mer is a suffix of $\mathit{Pattern}_i$ and a prefix of $\mathit{Pattern}_j$.
    (E.g., TG is a suffix of ATG and a prefix of TGG)
- Example:
  $\mathit{Patterns} = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$



Adjacency Lists

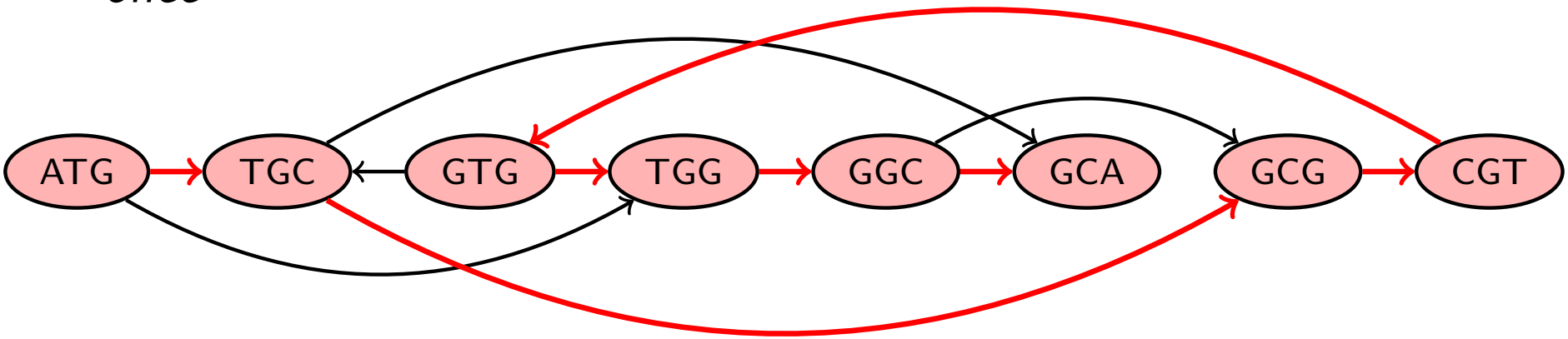
- A common representation of a graph is to list for each node its adjacent nodes



ATG $\rightarrow$ TGC, TGG
TGC $\rightarrow$ GCA, GCG
GTG $\rightarrow$ TGC, TGG
TGG $\rightarrow$ GGC
GGC $\rightarrow$ GCA, GCG
GCA $\rightarrow$
GCG $\rightarrow$ CGT
CGT $\rightarrow$ GTG

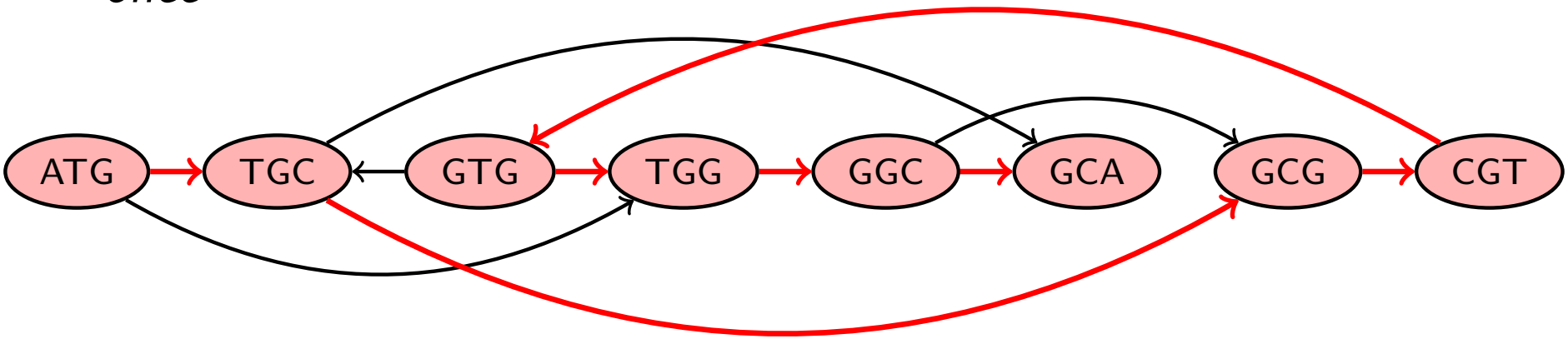
Hamiltonian Path

- A *Hamiltonian* path in a graph is a path that visits every node *exactly once*



Hamiltonian Path

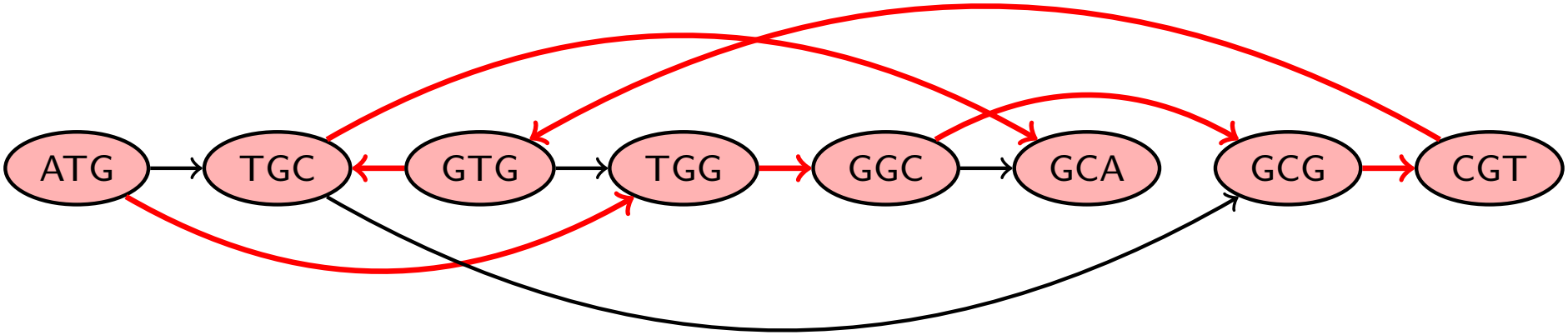
- A *Hamiltonian* path in a graph is a path that visits every node *exactly once*



- A Hamiltonian path in an overlap graph lists the k-mers in an order they appear in some string

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

Another Hamiltonian Path




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

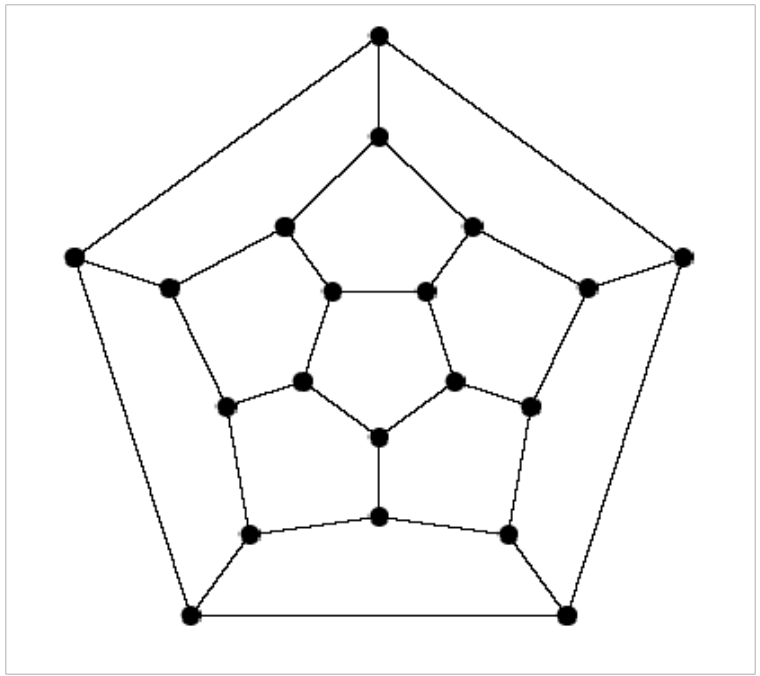
Hamiltonian Path Problem

- Goal: Construct a Hamiltonian path in a graph
- Input: A directed graph
- Output: A path visiting every node in the graph (if such a path exists)

Hamiltonian Path Problem

- Goal: Construct a Hamiltonian path in a graph
- Input: A directed graph
- Output: A path visiting every node in the graph (if such a path exists)

- Hamiltonian Cycle Problem: Find a *cycle* that visits every node exactly once
- A much studied problem invented as a game by Sir William Hamilton in 1857



Solving String Reconstruction Problem

- Goal: Reconstruct a string from its k-mer composition.
- Input: An integer k and a collection *Patterns* of k-mers
- Output: A string *Text* with k-mer composition *Patterns* (if such a string exists)

Algorithm

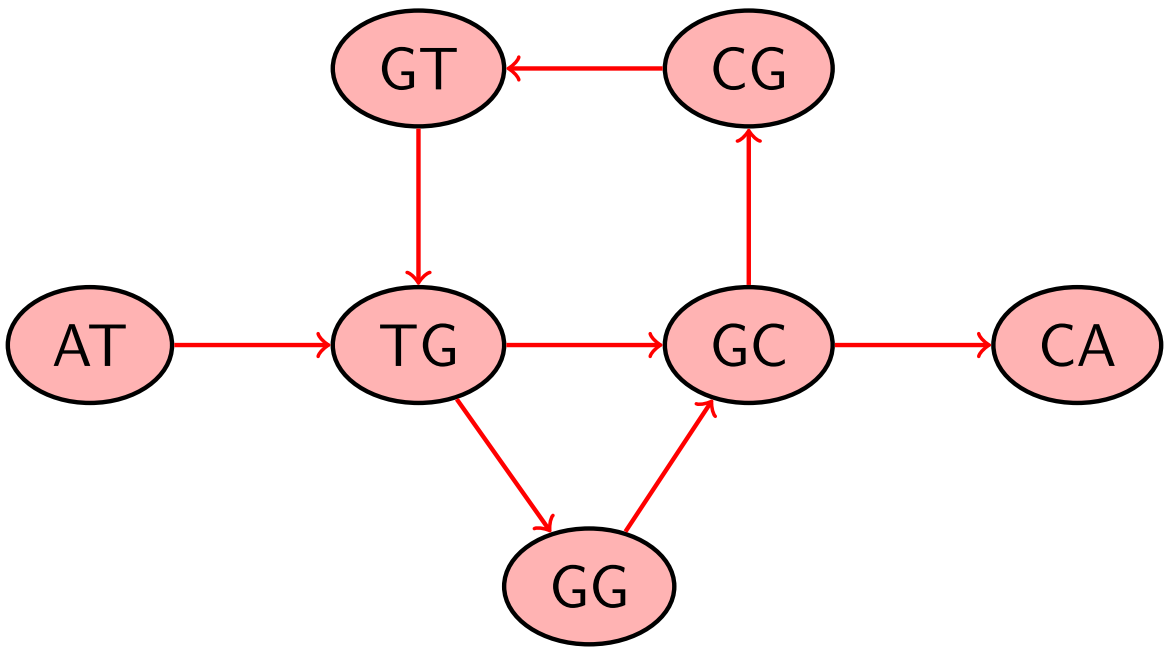
- Construct $\mathrm{Overlap}(\mathit{Patterns})$
- Find a Hamiltonian path in $\mathrm{Overlap}(\mathit{Patterns})$
- Output the string formed by the k-mers on the path
- If there is no Hamiltonian path, there is no solution to the String Reconstruction Problem

Hamiltonian Path Problem Is NP Complete

- *NP complete* problems
  - Thousands of important computational problems
  - No efficient (polynomial time) algorithm known
  - An efficient algorithms for one problem would immediately give an efficient algorithm to all the problems
  - Finding an efficient algorithm or proving that such an algorithm does not exist is the greatest open problem in computer science (P vs. NP problem)
- Hamiltonian Path problem can be solved only for small or easy graphs
  - What would be an easy graph?

De Bruijn Graph

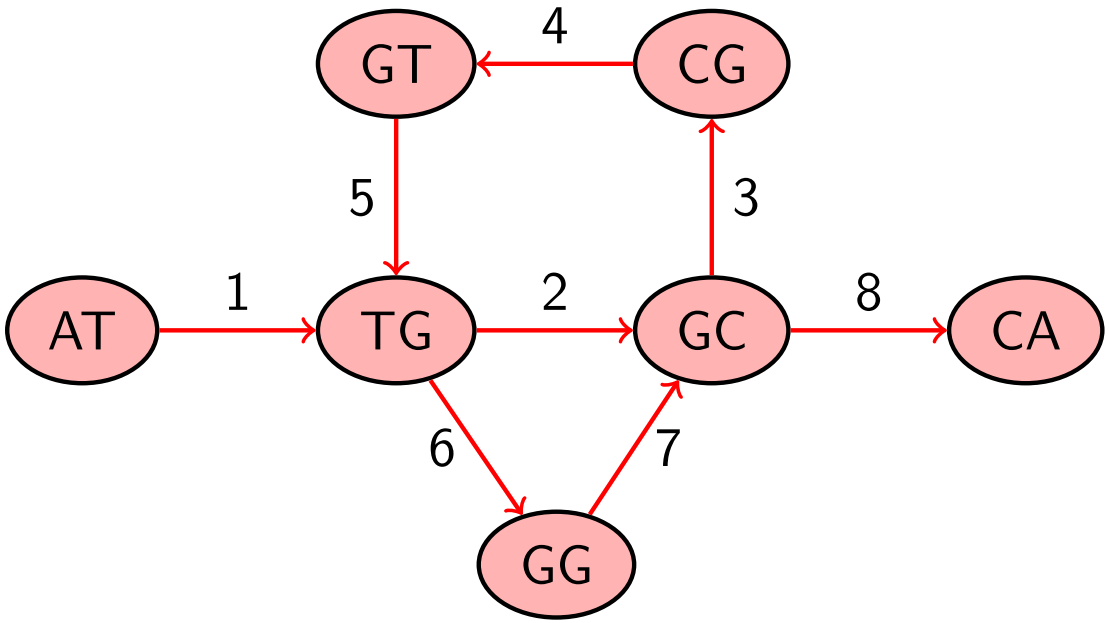
- The de Bruijn graph $\mathrm{DeBruijn}(\mathit{Patterns})$ is a directed graph that contains
  - A node for each $(k-1)$-mer that is a suffix or a prefix of a k-mer in *Patterns*
  - An edge for each k-mer *Pattern* in *Patterns* connecting the prefix of *Pattern* to the suffix of *Pattern*
- Example:
  $\mathit{Patterns} = \{\mathrm{ATG}, \mathrm{TGC}, \mathrm{GTG}, \mathrm{TGG}, \mathrm{GGC}, \mathrm{GCA}, \mathrm{GCG}, \mathrm{CGT}\}$



Eulerian Path

- An *Eulerian Path* in a graph is a path that uses every edge exactly once

- An Eulerian path in a de Bruijn graph lists the k-mers (corresponding to edges) and the $(k-1)$-mers (corresponding to nodes) in an order they appear in some string



ATGCGTGGCA

Eulerian Path Problem

- Goal: Construct an Eulerian path in a graph
- Input: A directed graph
- Output: A path visiting every edge in the graph (if such a path exists)

Solving String Reconstruction Problem

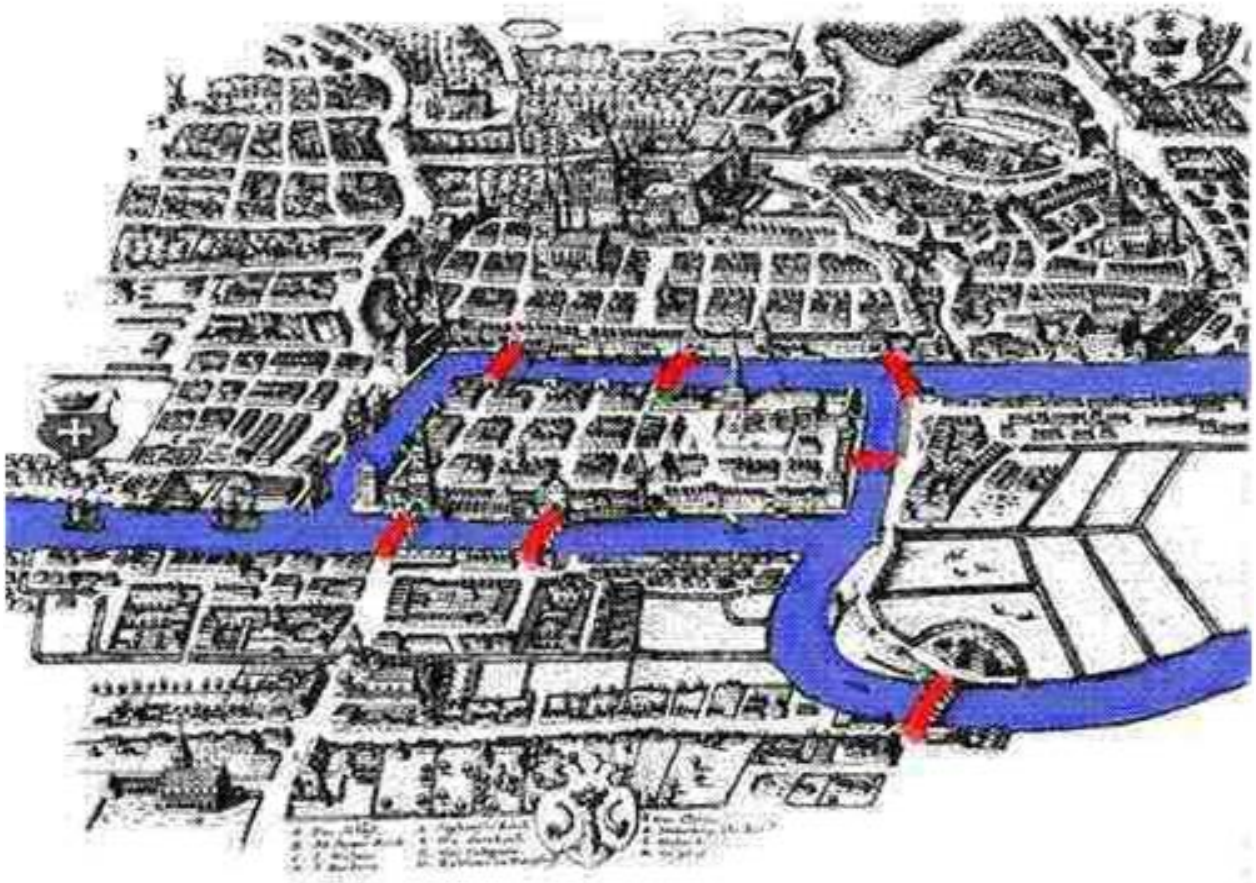
- Goal: Reconstruct a string from its k-mer composition.
- Input: An integer k and a collection *Patterns* of k-mers
- Output: A string *Text* with k-mer composition *Patterns* (if such a string exists)

Algorithm

- Construct $\mathrm{DeBruijn}(\mathit{Patterns})$
- Find an Eulerian path in $\mathrm{DeBruijn}(\mathit{Patterns})$
- Output the string formed by the k-mers on the path
- If there is no Eulerian path, there is no solution to the String Reconstruction Problem

Bridges of Königsberg

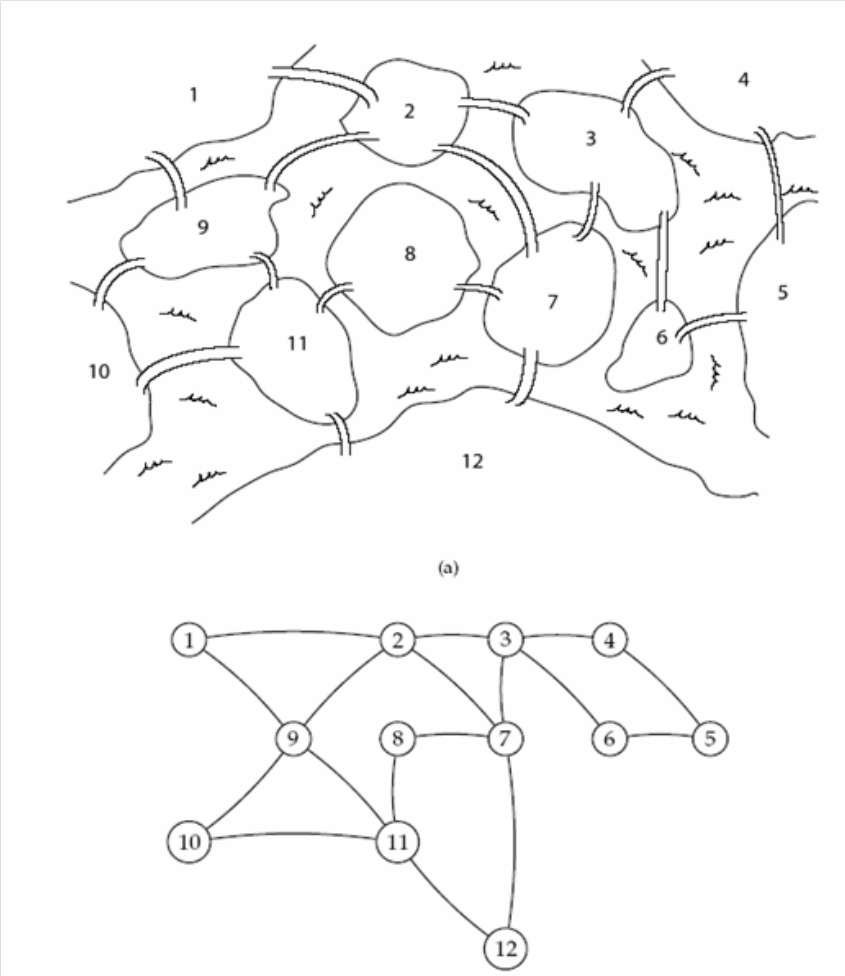
Find a tour crossing every bridge just once
Leonhard Euler, 1735



Bridges of Königsberg

Eulerian Cycle Problem

- Find a cycle that visits every edge exactly once
- Linear time algorithm



More complicated Königsberg

Euler's Theorem

- A node is *balanced* if the number of incoming edges equals the number of outgoing edges:

$$in(v) = out(v)$$

  A graph is balanced if every node is balanced.

- **Theorem**: A connected graph has an *Eulerian cycle* if and only if it is balanced.
- Such a graph is called an *Eulerian graph*.

Euler's Theorem

- A node is *balanced* if the number of incoming edges equals the number of outgoing edges:

$$in(v) = out(v)$$

  A graph is balanced if every node is balanced.

- **Theorem**: A connected graph has an *Eulerian cycle* if and only if it is balanced.
- Such a graph is called an *Eulerian graph*.

- Proof of "only if"
  - Any cycle enters each node the same number of times that it leaves the node, i.e., it uses the same number of incoming and outgoing edges.
  - There is no cycle that uses all edges of an unbalanced node.

Proof of "if"

- We want to show that every graph that is
  - balanced and
  - connected (i.e., there is a path from every node to every other node)

  has an Eulerian cycle.
- We do this by describing an algorithm that constructs an Eulerian cycle and is guaranteed to succeed for every balanced connected graph.

Eulerian Cycle Algorithm: Finding Cycles

Procedure FindCycle:

- Start following an arbitrary path from an arbitrary node
- Mark each travelled edge as used. Marked edges cannot be travelled again.
- Continue as long as possible, i.e., until you end up in a node with no unmarked outgoing edges.

- If the path ends in the starting node, a cycle has been formed.
- If the path ends in a different node, that node must be unbalanced (more incoming than outgoing edges) and the algorithm reports that the graph is not Eulerian.

Eulerian Cycle Algorithm: Combining Cycles

Procedure CombineCycles:

- Input: Two edge-disjoint cycles and a node that belongs to both cycles
  - Edge-disjoint means that no edge belongs to both cycles
- Output: A single cycle that containing all edges of the input cycles
- Algorithm: Starting from the shared node, travel first one cycle and then the other cycle

Eulerian Cycle Algorithm: Full Algorithm

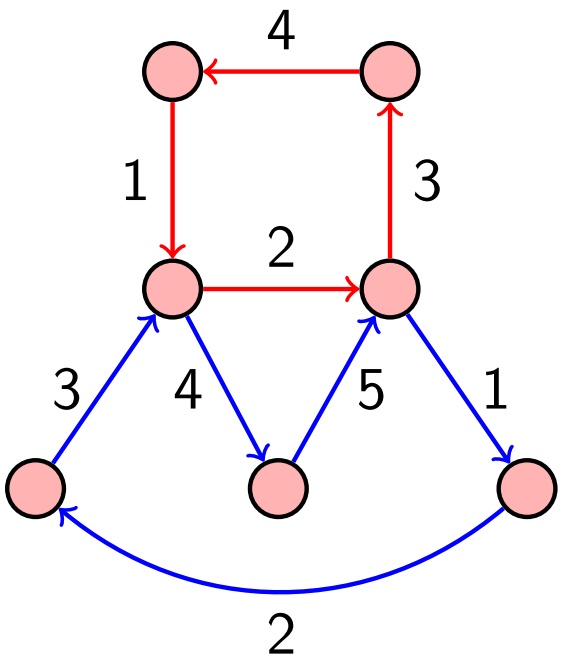
Algorithm EulerianCycle

1. Repeatedly find cycles using procedure FindCycle as long as there are unused edges.
   - In a balanced graph, all edges end up used.
2. Repeatedly combine two cycles into one using procedure CombineCycles until only one long cycle remains.
   - If the algorithm ends with two or more cycles that cannot be combined, the graph must be unconnected.
3. Output the one cycle (containing all edges).

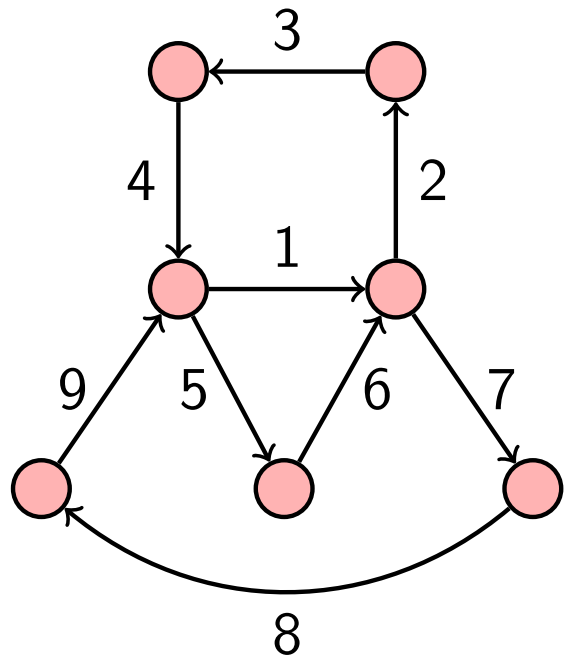
- Since the algorithm never fails for a balanced connected graph, this proves the "if" part.
- Can be implemented to run in linear time.

Eulerian Cycle Algorithm: Example

Find two cycles



Combine the cycles



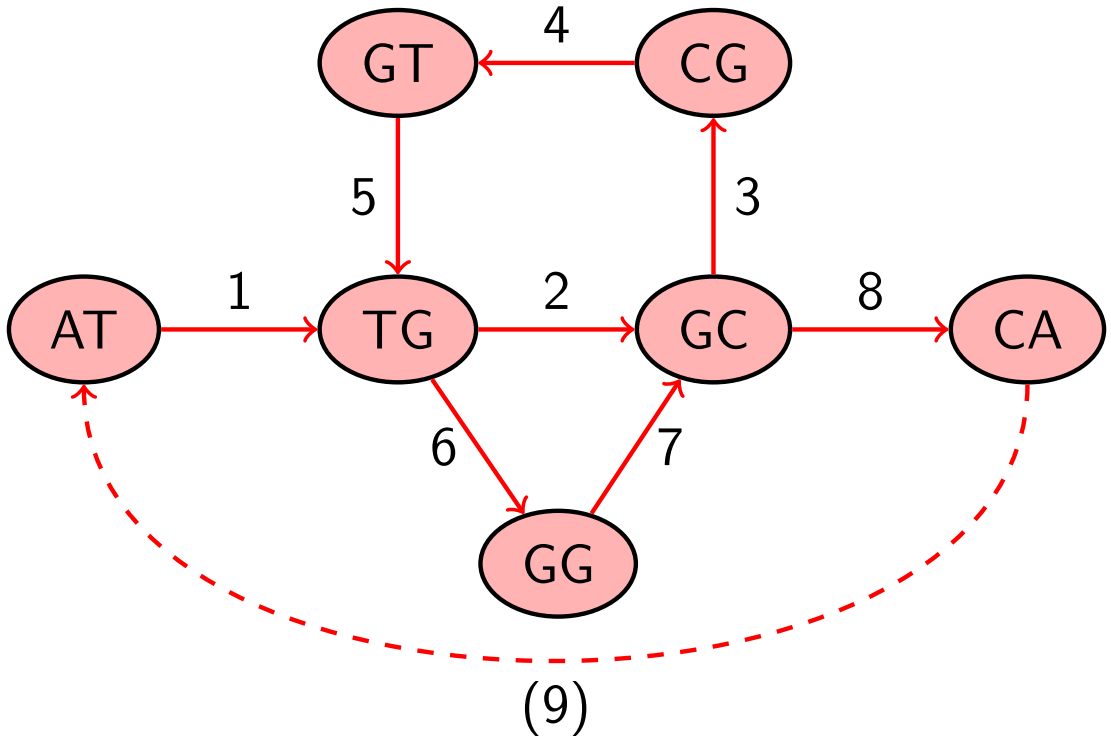
Eulerian Path vs. Cycle

- A node is *semi-balanced* if $in(v) = out(v) + 1$ or $in(v) = out(v) - 1$
- **Theorem**: A connected graph has a *Eulerian path* if and only if it contains at most two semi-balanced nodes and all other nodes are balanced.

Finding Eulerian path

- Find the two semi-balanced nodes, and add an edge between them to make the graph balanced
- Find an Eulerian cycle
- Remove the added edge to obtain an Eulerian path

Eulerian Path vs. Cycle: Example



k-Mers from Reads

- Sequencing by hybridization is not usable for large genomes
  - Too inaccurate data
  - Too small k (needs a microarray for all possible k-mers)
- Other sequencing techniques produce longer, possibly varying length reads (fragments) covering the genome (multiple times)
- Similar algorithmic techniques are used though
  - Form a combined k-mer compositions of all reads and build the DeBruijn graph for it
  - Fairly large k, e.g., $k = 20$ or even larger

k-Mers from Reads: Example

String	A T G G C G T G C A
Reads	A T G G C
	G C G T G
	G T G C A
k-Mers	A T G
	T G G
	G G C
	G C G
	C G T
	G T G
	G T G
	T G C
	G C A

Some Problems and Solutions

- Errors in reads
  - Multiple coverage, multiple sequencing methods
  - Error detection and correction
  - For example, find "bubbles" in de Bruijn graph
- Long repeats in genome (longer than reads)
  - Paired reads (obtained from the two ends of a long fragment)
  - Reconstruct first non-repetitive parts forming contigs
  - For example, find non-branching paths in de Bruijn graph