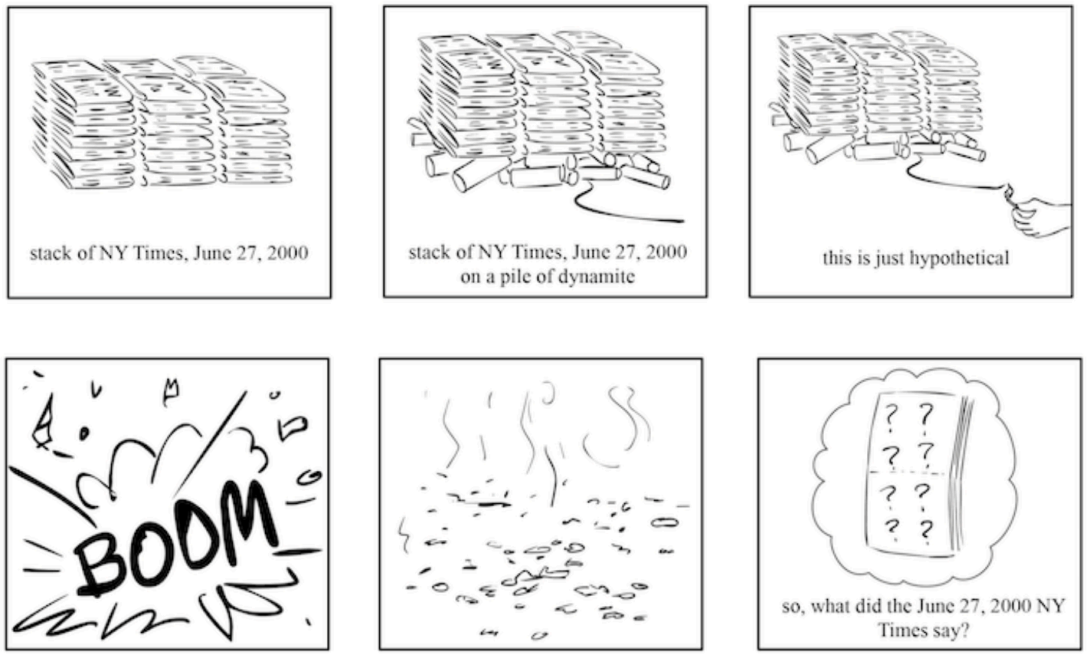
CS4049
Bioinformatics

Spring 2025

Rushda Muneer

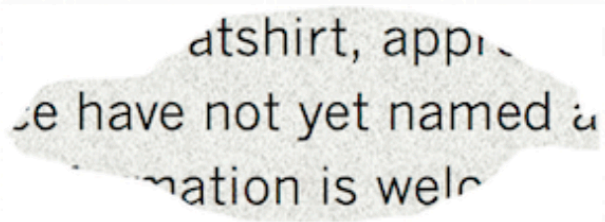
The Newspaper Problem

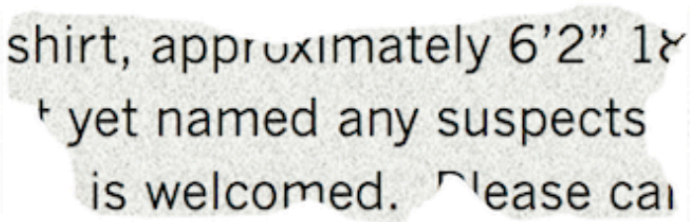
- Imagine stacking 100 copies of the **June 27, 2000** edition of *The New York Times* on a pile of dynamite.
- The explosion scatters **tiny, smoldering snippets** of newspaper.
- Instead of being completely destroyed, the fragments **contain partial information**.
- **Challenge:** Can we **reconstruct the news** from these scattered pieces?
- This problem illustrates how **incomplete data** can still be used to infer information.



From Exploding Newspapers to Genome Sequencing

- The Newspaper Problem is harder than a **jigsaw puzzle** because:
  - Multiple copies exist.
  - Some fragments are lost.
  - We must rely on **overlapping snippets** to reconstruct the whole.
- **Connection to Biology:**
  - Genome sequencing works similarly.
  - A genome is made up of **billions of nucleotides**.
  - Scientists must piece together **overlapping DNA fragments** to reconstruct an organism's genome.

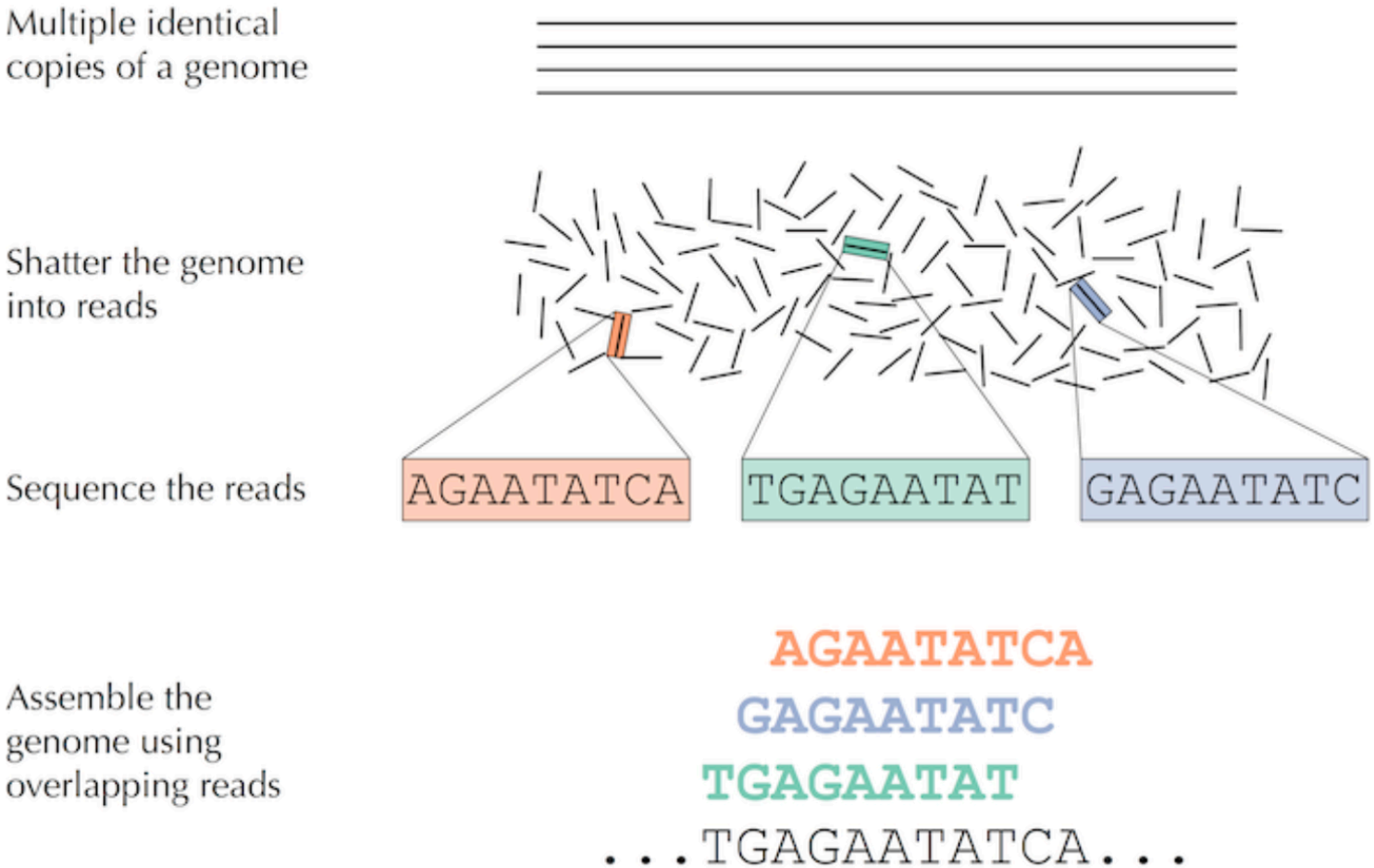




Genome Sequencing: A Puzzle of DNA Reads

- Traditional genome sequencing involves:
  - Extracting DNA from a **tissue or blood sample**.
  - **Breaking** the DNA into fragments.
  - **Sequencing** these fragments to produce **reads**.
- **The Challenge:**
  - Scientists don't know **where each read belongs** in the genome.
  - Like the **Newspaper Problem**, they must use **overlapping reads** to reconstruct the original sequence.
- **Genome assembly** is the process of piecing these reads together, just like reconstructing a shredded newspaper.

Genome Assembly



In DNA sequencing, many identical copies of a genome are broken in random locations to generate short reads, which are then sequenced and assembled into the nucleotide sequence of the genome.

Genome Assembly: Harder Than You Think

1. **Double-stranded DNA:** Reads could come from **either strand**, requiring reverse complements.
2. **Sequencing errors:** Errors in reads make **overlapping** fragments harder to identify.
3. **Gaps in coverage:** Some genome regions may **not be covered** by any read.

Reconstructing Strings from k-mers

- **Genome Assembly as a Computational Problem:**
  - A string **Text** is broken into **k-mers** (substrings of length k).
- The **k-mer composition**, denoted **Composition_k(Text)**, is the set of all k-mers, including repeats.
- **Example:**
  - Given **Text = "TATGGGGTGC"**
  - **Composition_3(Text) =** { "ATG", "GGG", "GGG", "GGT", "GTG", "TAT", "TGC", "TGG" }
- The **correct order of k-mers is unknown** when reads are generated.

String Reconstruction Problem

- **Connecting k-mers to reconstruct a string:**
  - A **k-mer overlaps** with another if they share **k-1 symbols**.
- The goal is to **order k-mers** correctly to reconstruct the original string.
- **Example (3-mers):**Given: **"AAT", "ATG", "GTT", "TAA", "TGT"**
- Identify the **starting k-mer** ("TAA") → No k-mer ends in "TA".
- Build the sequence step by step:
  - **TAA → AAT → ATG → TGT → GTT**
- Final reconstructed string: **"TAATGTT"**

TAA

AAT

ATG

TGT

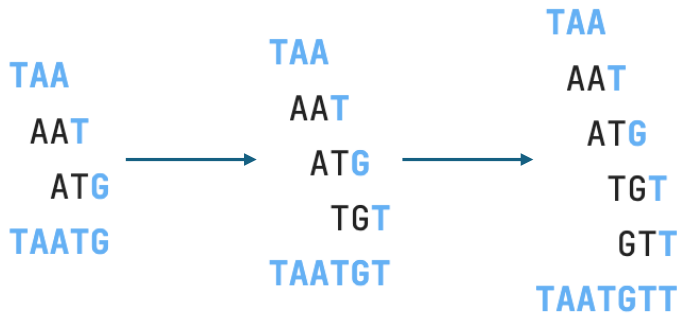
GTT

TAATGTT

String Reconstruction Problem

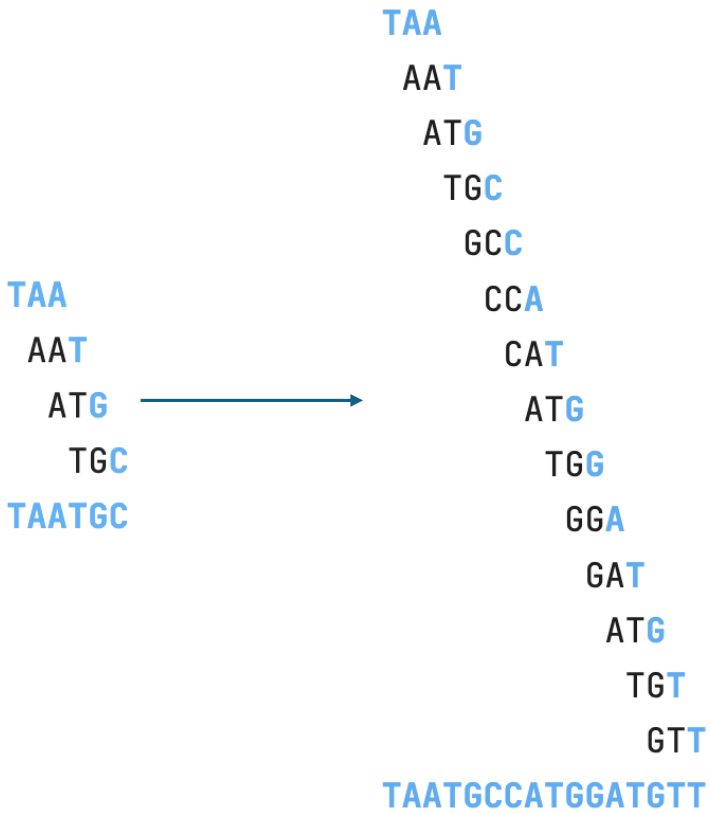
"AAT" "ATG" "ATG" "ATG" "CAT" "CCA" "GAT" "GCC" "GGA" "GGG" "GTT" "TAA" "TGC" "TGG" "TGT"

- **Building a sequence step by step:**
  - Start with **TAA** (no k-mer ends in "TA").
  - Extend **TAA → AAT → ATG**.
- **Choice Point: ATG** can be followed by **TGC, TGG, or TGT**.
  - Select **TGT**, then **GTT**.
- **Problem Encountered:**
  - **No k-mer starts with "TT"** → The sequence **cannot be completed**!



String Reconstruction Problem

- Like a good chess player, **think a few steps ahead** before making a move.
- **Avoiding the Dead End:**
  - Instead of extending **ATG → TGT**, we try **ATG → TGC**.
- We used **14 out of 15** k-mers.**Missing k-mer:** "GGG" → Genome is **one nucleotide too short**.



Repeats Complicate Genome Assembly

- **The Problem with Repeats:**
  - The k-mer **"ATG" appears three times**, creating **multiple choices** for extension:
    - **"TGG", "TGC", or "TGT"**
  - With only **15 reads**, this is manageable, but with **millions of reads**, repeats create **massive challenges**.
- **Why Repeats Matter in Real Genomes:**
  - Random sequences rarely contain **long repeats**, but **real genomes are not random**.
  - **50% of the human genome** consists of **repeated sequences**.
  - Example:
    - **Alu sequences (~300 nucleotides long)** appear **over a million times**, often with **minor variations** (insertions, deletions, or substitutions).
- **Impact on Genome Assembly:**
  - Repeated sequences make it **hard to determine the correct order** of k-mers.
  - Requires **advanced computational methods** to resolve ambiguous overlaps.

From a String to a Graph

- To resolve repeats, we need a strategy that allows us to look ahead and decide the correct assembly.
- The string **"TAATGCCATGGGATGTT"** is a possible solution for the previous set of 3-mers.
- **Different colors represent** the intervals between occurrences of "ATG".
- **Graph Representation:**
- **A graph** can be used to represent the sequence assembly, where **nodes** represent **k-mers**, and **edges** represent overlaps between them.
- **This allows us to "look ahead"** to make the correct sequencing decisions.

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG
         GGG
          GGA
           GAT
            ATG
             TGT
              GTT
TAATGCCATGGGATGTT
```

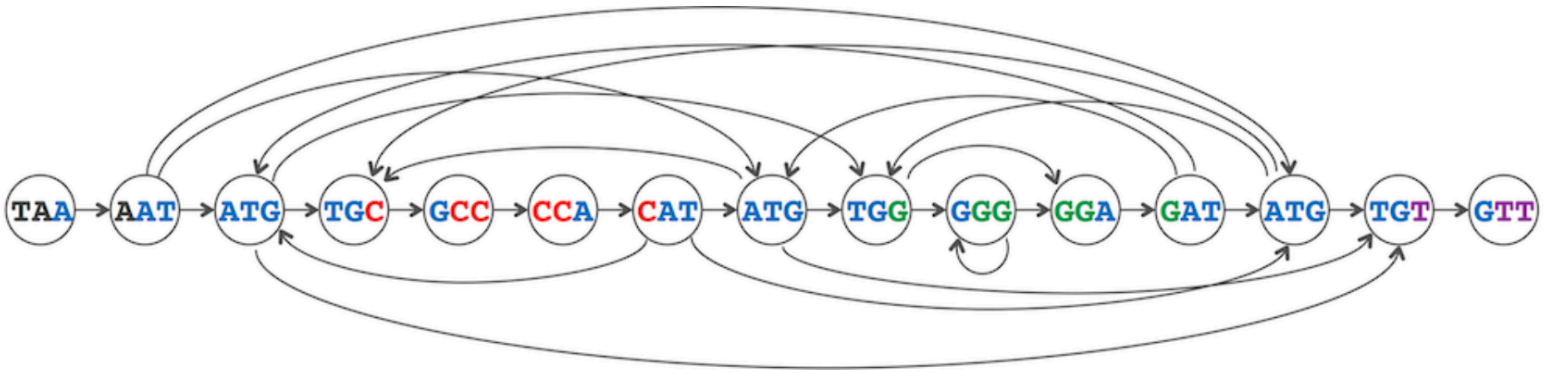
String Reconstruction as a Walk in the Overlap Graph

- Consecutive 3-mers in "**TA**ATGCCATGGGATGTT" are linked together to form this string's **genome path**.

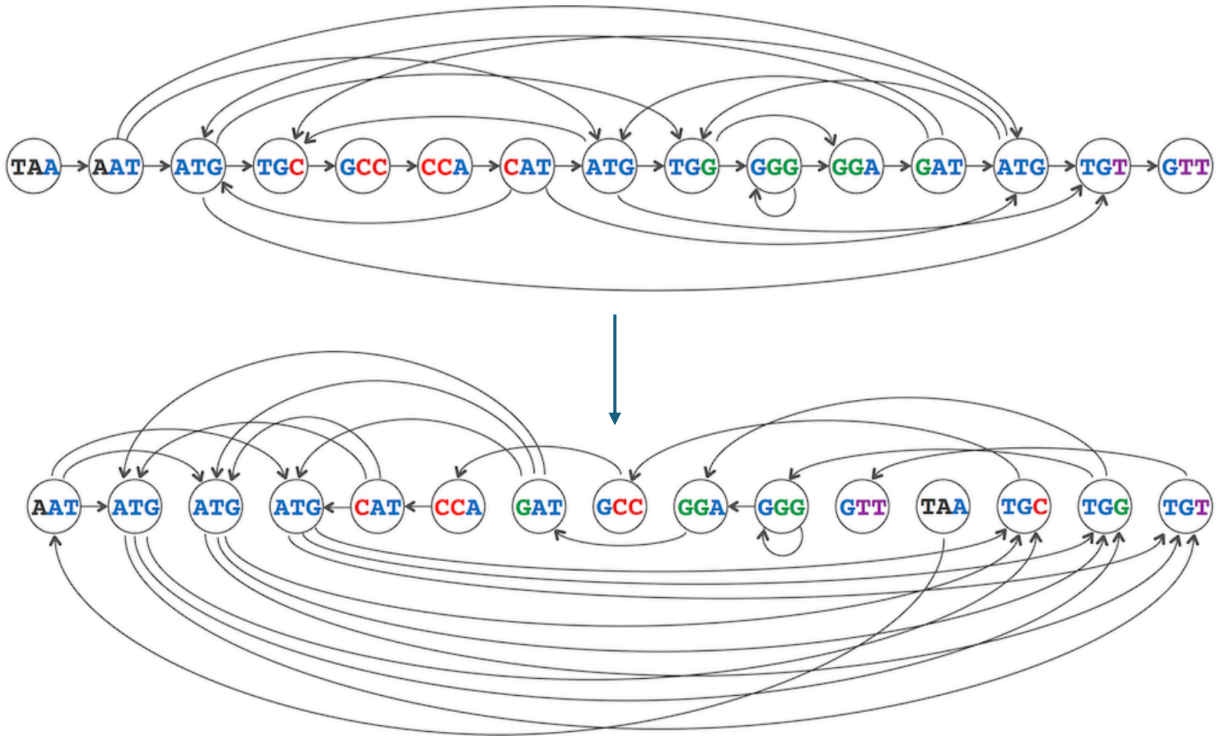
- **Prefix and Suffix of k-mers:**
- **Prefix** of a k-mer is the first **k-1** nucleotides.
- **Suffix** of a k-mer is the last **k-1** nucleotides.
- Example:
  - **Prefix("TAA") = "TA"**
  - **Suffix("TAA") = "AA"**
- In the genome path, the **suffix** of one 3-mer is the **prefix** of the next.
- Example:
  - **Suffix("TAA") = Prefix("AAT") = "AA"** in the genome path for "TAATGCCATGGGATGTT".
- **Constructing the Genome Path:**
- **Connect k-mers** using an arrow when the **suffix** of one k-mer matches the **prefix** of the next.
- This method allows us to reconstruct the **genome path**.

String Reconstruction as a Walk in the Overlap Graph

- **The Problem with Connecting k-mers:**
- Strictly following the rule of connecting k-mers based on their suffix and prefix leads to **many additional connections**.
- For example, the three occurrences of **"ATG"** will be connected to **"TGC", "TGG", and "TGT"**.
- **Directed Graph Representation:**
- The genome path can be represented as a **directed graph**, where:
  - **Nodes** are **k-mers**.
  - **Edges (arrows)** connect **nodes** when the suffix of one k-mer matches the prefix of another.
- This graph shows the possible **connections between k-mers** in the genome path.



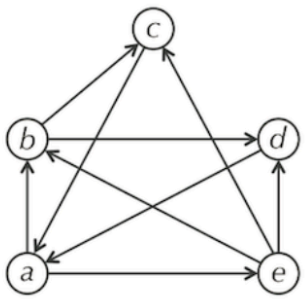
- In genome sequencing, we do not know the correct **order of k-mers** in advance.
- To manage this, we can **arrange k-mers lexicographically** (in dictionary order) to form an overlap graph.
- **Lexicographic Arrangement:**
  - **Rearranging the 3-mers** in lexicographic order leads to a **new overlap graph**.
- This arrangement results in the genome path being **hidden from the naked eye**, but it is still encoded in the graph structure.
- Even though the genome path may not be immediately obvious, it still **exists** in the graph.



Two Graph Representations for Genome Assembly

- When working with graphs in genome assembly, we **only need to store the pairs of nodes** connected by edges.
- **Adjacency Matrix (n × n):**
  - For a **directed graph** with **n nodes**, an **adjacency matrix** stores the information in an **n × n grid**.
  - The matrix element **Ai,j** is set to:
    - **1** if there's a directed edge from node **i** to node **j**.
    - **0** if no directed edge exists between them.
  - This representation is easy to use but can be memory-heavy for large graphs.
- **Adjacency List:**
  - A **memory-efficient alternative** is to use an **adjacency list**, which lists all **nodes connected** to each node.
  - This saves space by only storing the edges that actually exist.

Graph



Adjacency Matrix

	a	b	c	d	e
a	0	1	0	0	1
b	0	0	1	1	0
c	1	0	0	0	0
d	1	0	0	0	0
e	0	1	1	1	0

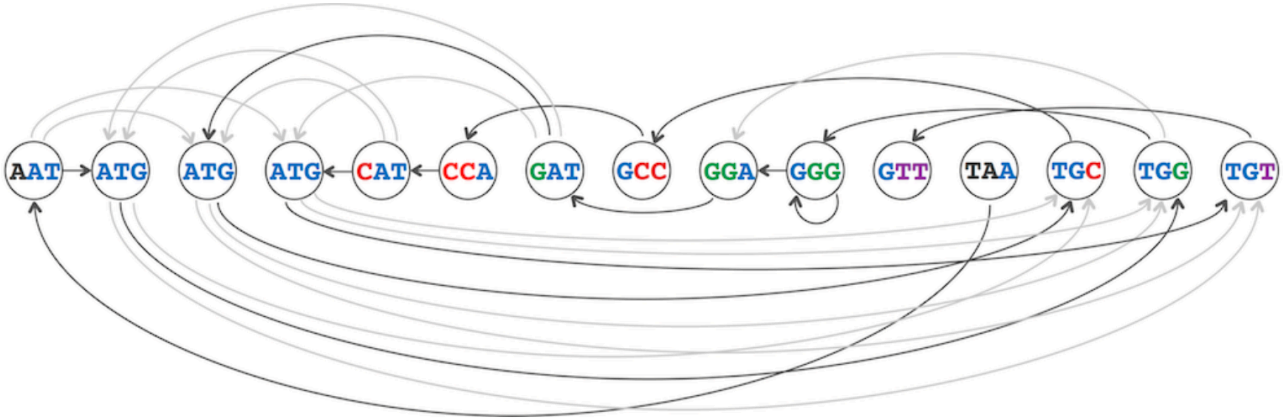
Adjacency List

a is adjacent to *b* and *e*
b is adjacent to *c* and *d*
c is adjacent to *a*
d is adjacent to *a*
e is adjacent to *b*, *c*, and *d*

- **When to Use Each Representation:**
  - **Adjacency Matrix:** Useful for dense graphs where many nodes are interconnected.
  - **Adjacency List:** More efficient for sparse graphs with fewer connections

Hamiltonian Paths

- To solve the **String Reconstruction Problem**, we need to find a **Hamiltonian path** in the overlap graph.
- A **Hamiltonian path** is a path that **visits every node exactly once**.
- This path will give us the correct order of k-mers to reconstruct the genome.
- The overlap graph can have **multiple Hamiltonian paths**.
- For example, in addition to the Hamiltonian path that reconstructs " **TA**ATGCCATGGGATGTT ", we can also find another Hamiltonian path that spells the genome " **TA**ATGGGATGCCATGTT ".
- These two genomes differ in the order of "CC" and "GG", but they have the same 3-mer composition.



•**Universal Strings:**

- A **universal string** is one that contains every possible k-mer as a substring.
- Finding such a string is closely related to finding Hamiltonian paths in overlap graphs.