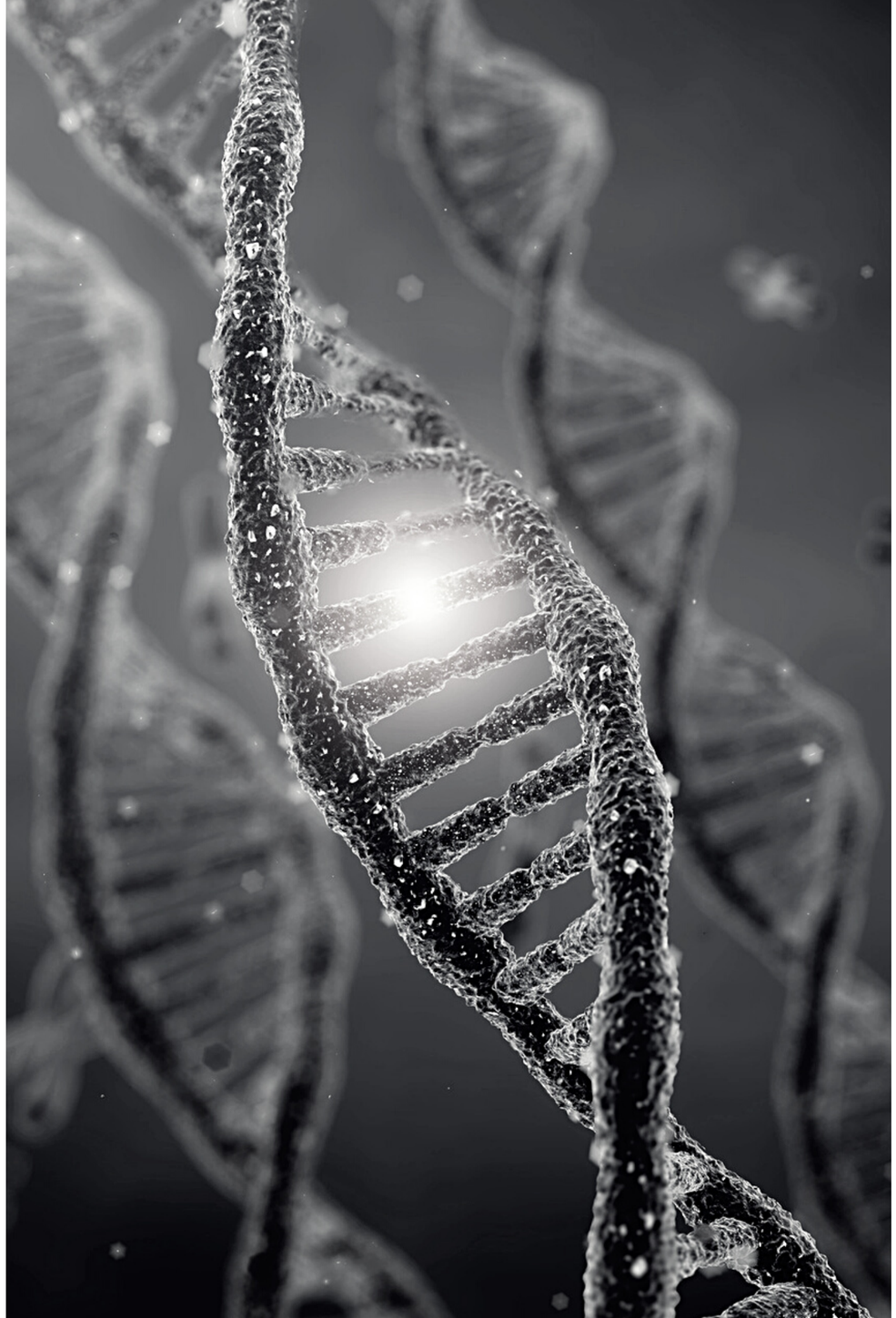
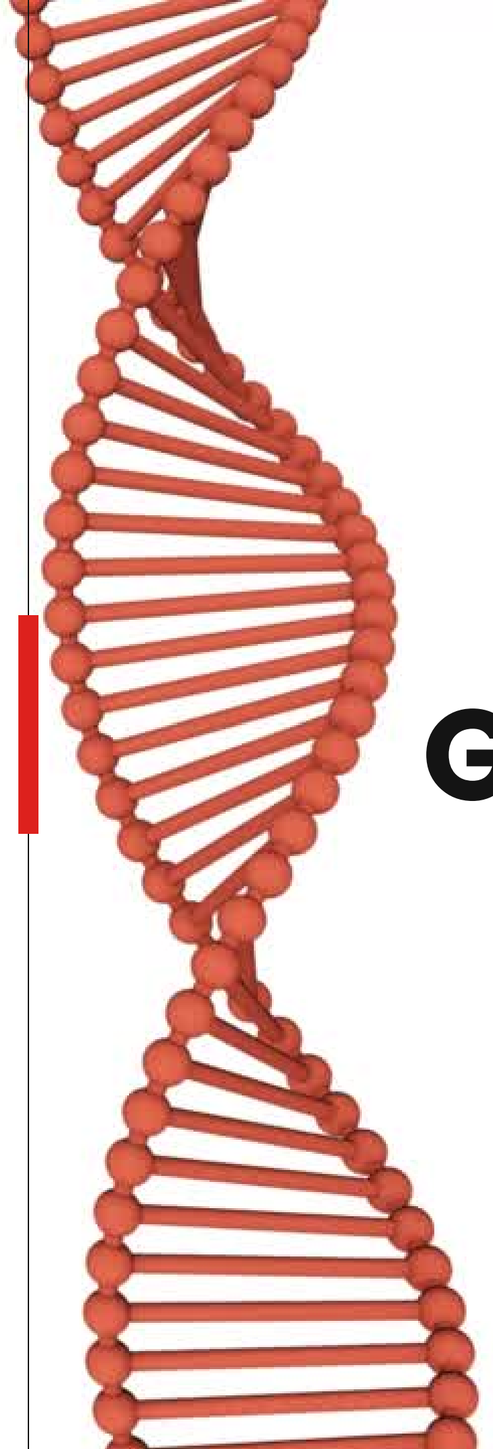


21BIO201

# Hamiltonian path

APPROACH FOR GENOME ASSEMBLY AND ITS CHALLENGES



# Group 1

## Members:


MUHAMMED SHAJAHAN | AM.EN.U4AIE21144

R. JAGATH SURYA | AM.EN.U4AIE21152

AKSHAY KRISHNAN T | AM.EN.U4AIE21109

NIRANJAN PRASANTH | AM.EN.U4AIE21148

ADITYA VARDHAN REDDY | AM.EN.U4AIE21171

KIRAN P | AM.EN.U4AIE21175

# Objectives and Scope

**String Reconstruction**

Reconstructing the genome by finding the hamiltonian path.

**Overlap Graphs**

Overlap graphs of the k-mers are made to find the hamiltonian path.

**Determing if hamiltonian path exists**

Analyzing graphs to determine the existence of hamiltonian path.

**Simple implementation of Hamiltonian Path**

# Introduction

Genome assembly refers to the process of putting nucleotide sequence into the correct order.

A Hamiltonian path is defined as the path in a directed or undirected graph which visits each and every vertex of the graph exactly once.

William Rowan Hamilton

Every Hamiltonian cycle in an overlap graph corresponds to a single genome reconstruction where all the repeats (long sequences that appear more than once) are completely resolved (i.e., their place in the genome determined)

# Previous Approach

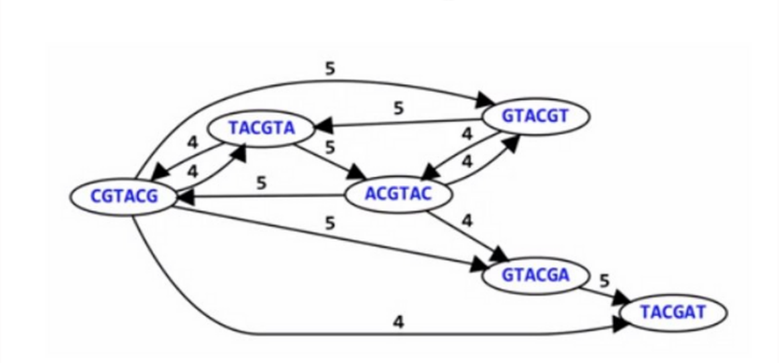
**Genome Assembly before Hamiltonian Paths**

String reconstruction from k-mer composition. K-mer composition is the collection of k-mer substrings of the text.

K-mers are connected if they overlap in k-1 symbols. There might be cases when some k-mers are omitted, this can be solved by backtracking. But backtracking is not possible for a large number of reads.

# Graph Approach

- Since, we have multiple overlaps among various reads, it is better to approach this problem using graph.
- A Graph is a non-linear data structure consisting of vertices and edges.
- Here, we denote reads as vertices and overlaps between reads as edges between two vertices.
- We use Directed acyclic graph
- Our challenge: Find a path through the overlap graph such that we can reconstruct the genome.

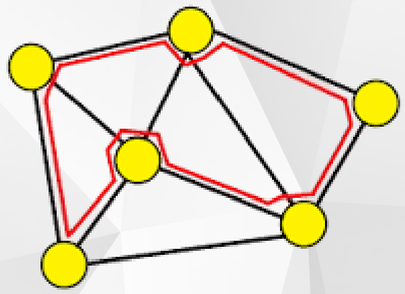


# Hamiltonian Path Cycles

Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once.

Hamiltonian cycle is a Hamiltonian path that is a cycle

- that visits each vertex exactly once
- will return to starting vertex



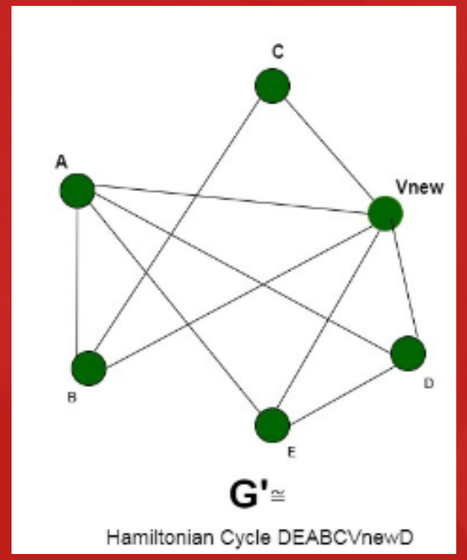
Any Hamiltonian cycle can be converted to a Hamiltonian path by removing one of its edges.

A graph is Hamiltonian if it contains a Hamiltonian cycle.

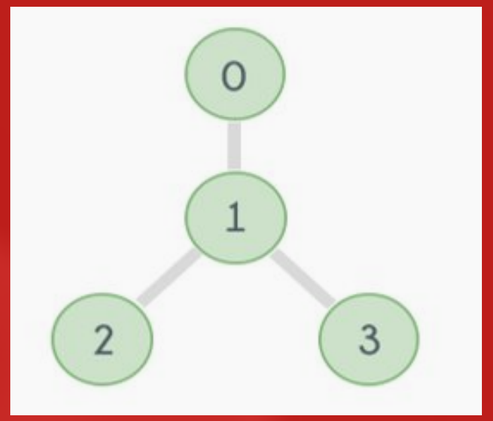
In general, the problem of finding a Hamiltonian path is NP-complete so the only known way to determine whether a given general graph has a Hamiltonian path is to undertake an exhaustive search.

→

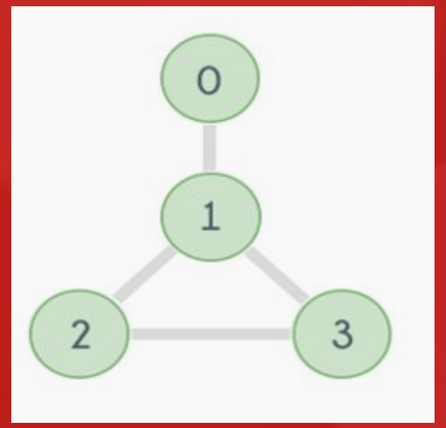
# Visualization of Hamiltonian Graphs




Here a hamiltonian cycle exists.




Here a hamiltonian path does not exist.




Here a hamiltonian path exist but cycle does not exist .

# Check Whether Graph is Hamiltonian

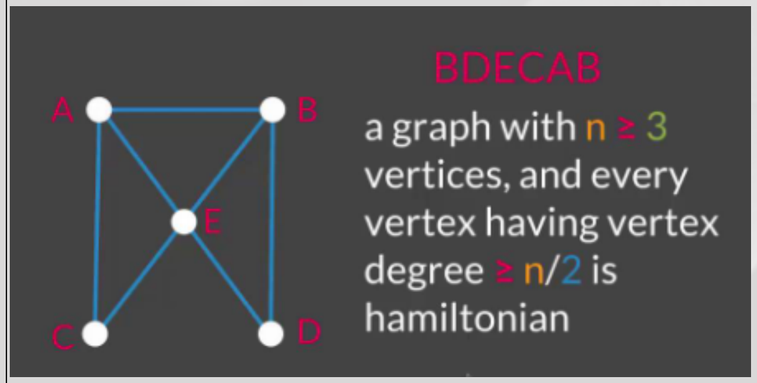
There are certain criteria that rule out the existence of a Hamiltonian circuit in a graph. Such as :

If there is a vertex of degree one in a graph then it is impossible for it to have a Hamiltonian circuit.

There are certain theorems which give sufficient but not necessary conditions for the existence of Hamiltonian graphs. They are as follows:

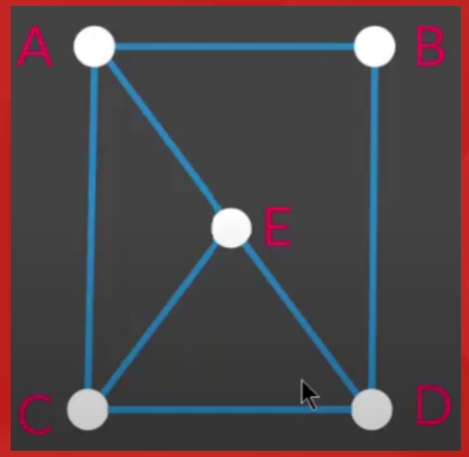
<u>Dirac's Theorem</u>

A Graph G with n vertices (n >= 3) is  Hamiltonian if every vertex has degree n/2 or greater.



## Ore's Theorem

- If for any pair of non-adjacent vertices u and v, deg(u) + deg(v) >= n, then G contains a Hamilton circuit.




For directed graphs there are other theorems. They are:
**Ghouila–Houiri -** A strongly connected simple directed graph with n vertices is Hamiltonian if every vertex has a full degree greater than or equal to n.

**Meyniel** - A strongly connected simple directed graph with n vertices is Hamiltonian if the sum of full degrees of every pair of distinct non-adjacent vertices is greater than or equal to {\displaystyle 2n-1}

# LITERATURE REVIEW

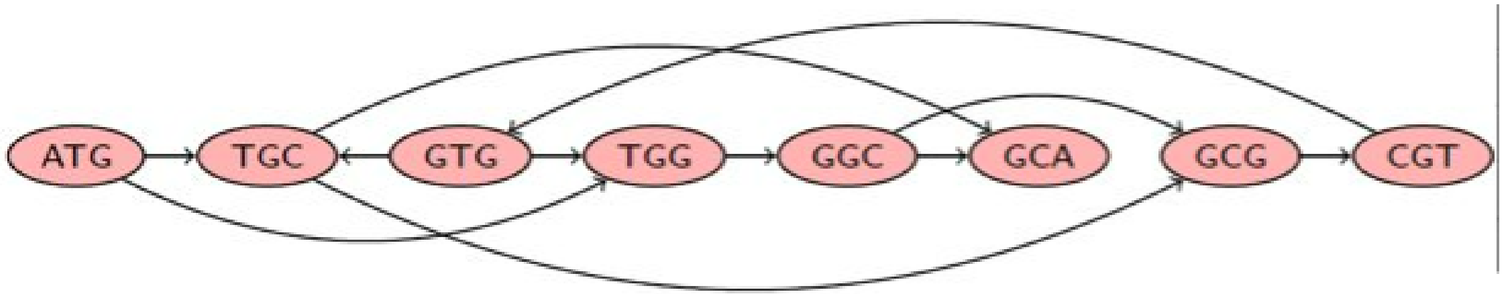
- The Hamiltonian path problem is an NP-complete problem, meaning that there is no known efficient algorithm for solving it. Despite this, there have been a number of approaches proposed for finding Hamiltonian paths in graphs.

- Overall, the research on the Hamiltonian path problem has focused on developing efficient algorithms for finding Hamiltonian paths in various types of graphs, as well as on understanding the computational complexity of the problem and the inherent difficulty of finding Hamiltonian paths in certain types of graphs.

# Overlap graph and construction

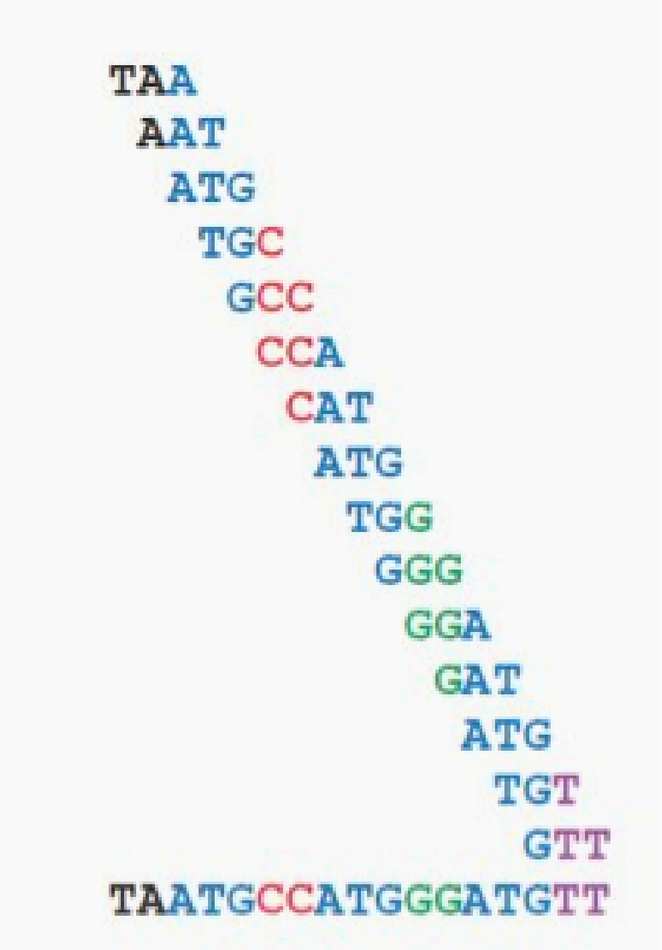
- Overlap graphs highlight situations in which there are interaction effects.
- Overlaps between reads can be determined by constructing an overlap graph.

| Node | Suffix | Prefix | Connected Nodes Suffix= Prefix |
|---|---|---|---|
| ATG | TG | AT | TGC,TGG |
| TGC | GC | TG | GCA,GCG |
| GTG | TG | GT | TGC,TGG |
| TGG | GG | TG | GGC |
| GGC | GC | GG | GCA,GCG |
| GCA | CA | GC | |
| GCG | CG | GC | CGT |
| CGT | GT | CG | GTG |



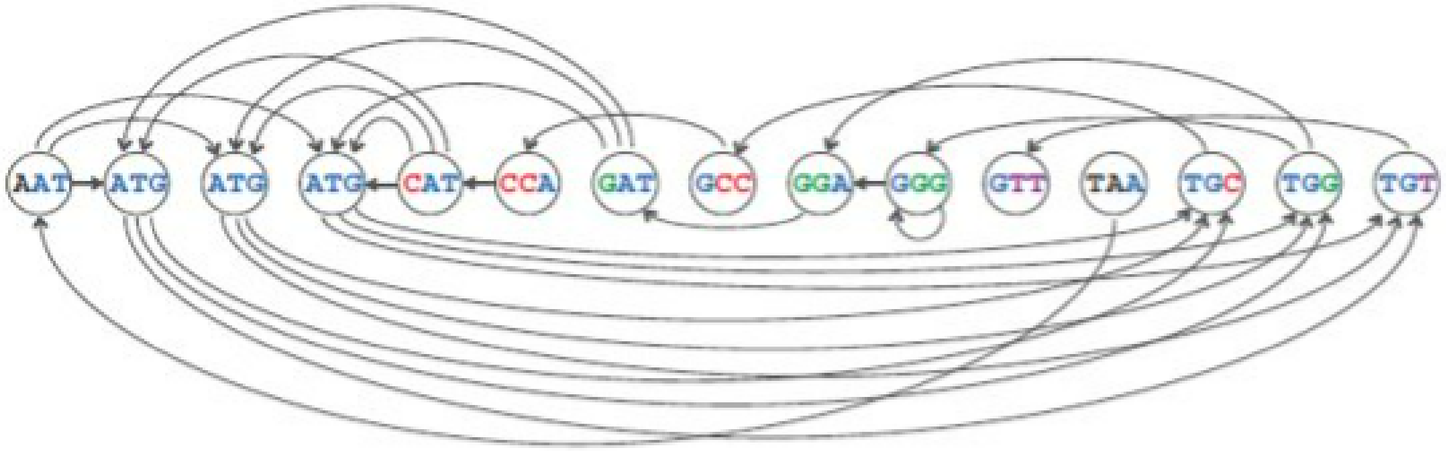
# Hamiltonian path through overlap graph

- To Solve string reconstruction problem, we are looking for a path in overlap graph that visits every node exactly once
- Consecutive k-mers in a genome, are linked together to form genome path
- Consecutive k-mers has relations



# Hamiltonian path through overlap graph

Overlap Graph




Hamiltonian Path 1

Genome path

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

Reconstructed Genome

TAATGGGATGCCATGTT

# SIMPLE IMPLEMENTATION

# Problem Definition

**Problem**

A Hamiltonian path is a path in a graph that visits each vertex exactly once. Checking whether a graph contains a Hamiltonian path is a well-known hard problem. At the same time it is easy to perform such a check if a given graph is a DAG.

**Given:** A positive integer $k \leq 20$ and $k$ simple directed acyclic graphs in the edge list format with at most $10^3$ vertices each.

**Return:** For each graph, if it contains a Hamiltonian path output "1" followed by a Hamiltonian path (i.e., a list of vertices), otherwise output "-1".

- Simple implemention
- Nodes are numbered 1, 2, 3, etc.
- Aim: To check Hamitonian Path exists
- Hamiltonian path of the graph

# Datasets & Outcomes

**Sample Dataset**

```
2

3 3
1 2
2 3
1 3

4 3
4 3
3 2
4 1
```

no. of samples

(V,E)

Edge Information

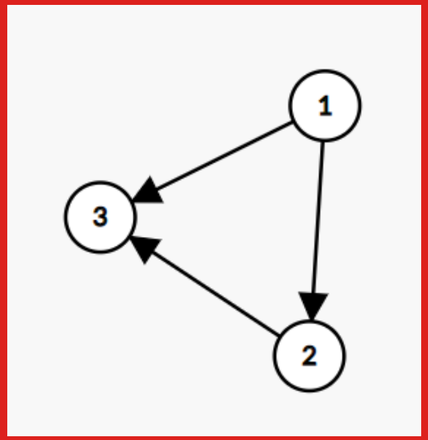
**Sample Output**

```
1 1 2 3
-1
```

- Return 1 if Hamiltonian exists
- followed by Hamiltonian path through the graph
- Otherwise, return -1

# Visualize Graph

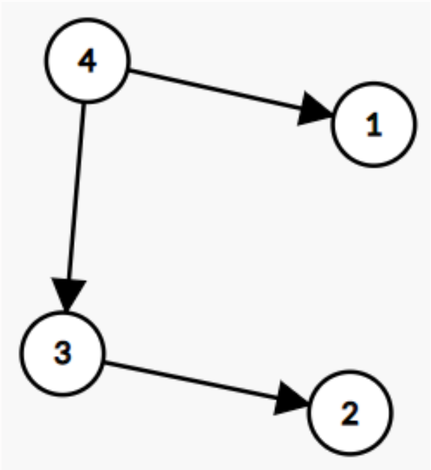
## Graph 1




```
{
  1 : [2,3],
  2 : [3],
  3 : []
}
```

Hamiltonian path: 1->2->3

## Graph 2




```
{
  1 : [],
  2 : [],
  3 : [2],
  4 : [1,3]
}
```

Hamiltonian path does not exist

# Get Information from dataset

```python
data = "file.txt"
graphs = []
vertices = []
edges = []
with open(data, "r") as f:
    k = int(f.readline().strip())
    for line in f:
        if line.strip():
            vertice1, vertice2 = list(map(int, line.strip().split(" ")))
            graph[vertice1].append(vertice2)
        else:
            vertice, edge = map(int, f.readline().strip().split(" "))
            print(edge)
            graph = {v: [] for v in range(1, vertice + 1)}
            graphs.append(graph)
            vertices.append(vertice)
            edges.append(edge)
```

- Read dataset
- k = no. of graph samples (k = 2)
- vertices = list that stores total no. of vertices in the sample graphs.
- edges = list that stores no. of edges in the sample graphs
- graphs = list that stores all the sample graphs in adjacency list format.

# Topological sorting

- Inputs: graph, V
- stack : list which provides the path through the graph.
- Iterate through each vertices and check whether it is already visited
- if not , add to the stack,
- and  iterate through it's neighbouring node(which shares an edge) and check whether it is already visited
- if not add its neighbour to the stack
- return stack

```python
def topologicalsortUtil(v, visited, stack):
    visited[v] = True
    for i in graph[v]:
        if visited[i] == False:
            topologicalsortUtil(i, visited, stack)
    stack.insert(0, v)


def topologicalsort(graph, vertice):
    visited = [False] * (vertice + 1)
    stack = []

    for v in range(1, vertice + 1):
        if visited[v] == False:
            topologicalsortUtil(v, visited, stack)
    return (stack)
```

Note: a node is visited (and added to stack) only once.

- After calling topologicalsort function, we will get a stack containing nodes of the graph
- In check_consequtive function, we are checking if there is a Hamiltonian path in graph
- if there is a hamiltonian path present in the graph, Return 1 ,followed by Hamiltonian path through the graph
- If there is no hamiltonian path present in the graph, Return -1

```python
def check_consecutive(stack, graph):
    for i in range(len(stack) - 1):
        if stack[i + 1] not in graph[stack[i]]:
            return False
    return True


for n in range(k):
    graph = graphs[n]
    vertice = vertices[n]
    stack = topologicalsort(graph, vertice)
    if check_consecutive(stack, graph):
        print(1, end=" ")
        for i in stack:
            print(i, end=" ")
        print("")
    else:
        print(-1)
```

# OUTPUT

```
1 1 2 3

-1
```

# LIMITATIONS OF HAMILTONIAN PATH APPROACH

1) NP HARD - NP-hardness (non-deterministic polynomial-time hardness) is a property of a class of problems that are informally at least as hard as the hardest problems in NP.
NP - A problem is class NP (non-deterministic polynomial time) if a solution can be checked in polynomial time, but no known algorithm can generate the solution in polynomial time

2) Limited applicability: The Hamiltonian path problem is only applicable to graphs that are connected, meaning that there is a path between every pair of vertices.

3) Multiple solutions: In some cases, a graph may have multiple Hamiltonian paths. This can make it difficult to determine the "best" solution

4) Difficulty in finding a starting point: In some cases, it may be difficult to determine where to start the Hamiltonian path.

# Possible Solutions

1) Using Heuristic Algorithms that are designed to find good, but not necessarily optimal, solutions to the problem

2) Exact algorithms: These are methods that guarantee to find the optimal solution to the problem

3) Hybrid algorithms: These are approaches that combine heuristic and exact algorithms to find good solutions that are close to optimal.

4) Another approach is to use de Bruijn graph and find Eulerian Path for the String Reconstruction Problem.

# CONCLUSIONS

1) We saw the previous approaches and its difficulties

2) Overlap graphs and its construction

3) Hamiltonian Cycle and its Visualization

4) Checking whether Hamiltonian Path exists or not

5) Hamiltonian Path through overlap graph demonstration

6) Implementation of the solution to the problem

7) Limitations and Possible solutions to Hamiltonian path problem

# References


1. https://rosalind.info/problems/hdag/
2. https://towardsdatascience.com/russian-bridges-eulerian-circuits-and-genome-assembly-9fd1d84832e9
3. https://www.researchgate.net/publication/228732222_On_Hamiltonian_cycles_and_Hamiltonian_paths
4. https://en.wikipedia.org/wiki/Hamiltonian_path
5. https://en.wikipedia.org/wiki/Hamiltonian_path_problem
6. https://amritauniv.sharepoint.com/sites/IntelligenceofBiologicalSystems

Thank you!