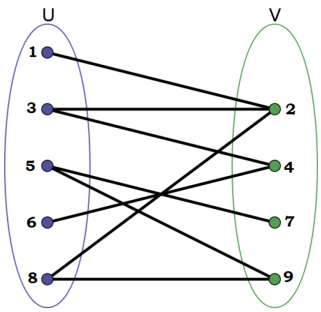
Graphs-1

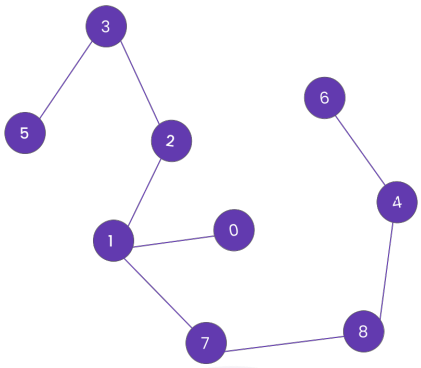
Assignment Solutions

PW SKILLS

Q1. Given an undirected graph, check if it is bipartite or not. A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V The following graph is bipartite as we can divide it into two sets, U and V, with every edge having one endpoint in set U and the other in set V:



Input:



Output:

```
Graph is bipartite
```

Ans: [ASS_Code1.java]

If a graph contains an odd cycle, we cannot divide the graph such that every adjacent vertex has a different color.

- To check if a given graph contains an odd-cycle or not, do a breadth–first search starting from an arbitrary vertex v.
- If in the BFS, we find an edge, both of whose endpoints are at the same level, then the graph is not Bipartite, and an odd-cycle is found.
- Here, the vertex level is its minimum distance from the starting vertex v.
- So, the odd-level vertices will form one set, and the even-level vertices will form another.

Q2. Given a list of departure and arrival airports, find the itinerary in order. It may be assumed that departure is scheduled from every airport except the final destination, and each airport is visited only once, i.e., there are no cycles in the route.

For example,

Input:

HKG —> DXB
FRA —> HKG
DEL —> FRA

Output: DEL —> FRA —> HKG —> DXB

Input:

LAX —> DXB
DFW —> JFK
LHR —> DFW
JFK —> LAX

Output: LHR —> DFW —> JFK —> LAX —> DXB

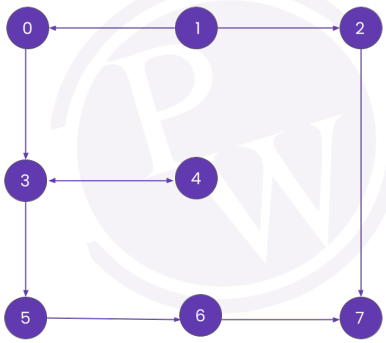
Ans: ASS_Code2.java

We can easily solve this problem in linear time using hashing.

- The idea is to find the source airport and print the itinerary starting from the source airport.
- To find the source airport, construct a set of destination airports from a given list of tickets.
- A source airport would be one that is not present in the list of destination airports.
- Once the source airport is found, traverse the list of tickets to the print itinerary in order using recursion.

Q3. Given a directed graph and two vertices (say source and destination vertex), determine if the destination vertex is reachable from the source vertex or not. If a path exists from the source vertex to the destination vertex, print it.

For example, there exist two paths [0—3—4—6—7] and [0—3—5—6—7] from vertex 0 to vertex 7 in the following graph. In contrast, there is no path from vertex 7 to any other vertex.

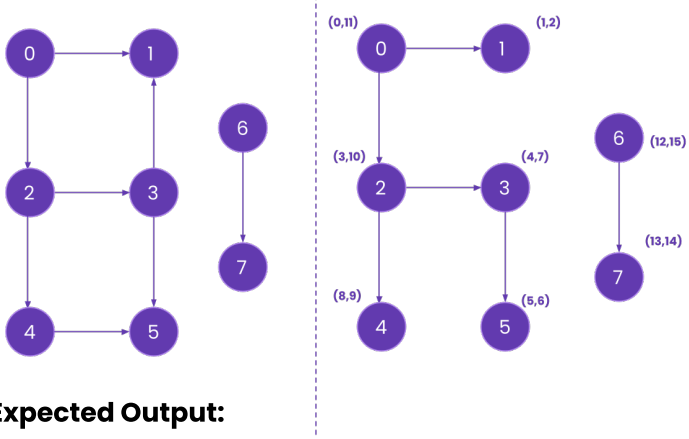


Ans: ASS_Code2.java

We can easily solve this problem in linear time using hashing.

- The idea is to find the source airport and print the itinerary starting from the source airport.
- To find the source airport, construct a set of destination airports from a given list of tickets.
- A source airport would be one that is not present in the list of destination airports.
- Once the source airport is found, traverse the list of tickets to the print itinerary in order using recursion.

Q4. Given a graph, find the arrival and departure time of its vertices in DFS. The arrival time is the time at which the vertex was explored for the first time in the DFS, and departure time is the time at which we have explored all the neighbors of the vertex, and we are ready to backtrack.
The following directed graph has two connected components. The right-hand side shows the arrival and departure time of vertices when DFS starts from vertex 0.



Expected Output:

```
Vertex 0 (0, 11)
Vertex 1 (1, 2)
Vertex 2 (3, 10)
Vertex 3 (4, 7)
Vertex 4 (8, 9)
Vertex 5 (5, 6)
Vertex 6 (12, 15)
Vertex 7 (13, 14)
```

Ans: ASS_Code4.java

The idea is to run Depth–first search (DFS).

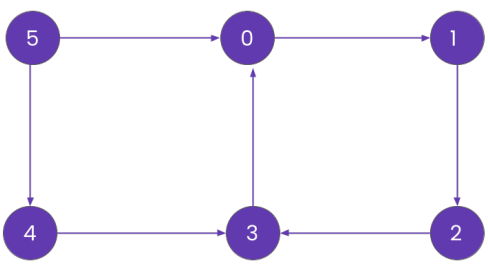
- Before exploring any adjacent nodes of any vertex in DFS, note the vertex's arrival time.
- After exploring all adjacent nodes of the vertex, note its departure time.
- After the DFS call is over (i.e., all the graph vertices are discovered), print the vertices' arrival and departure time.

Please note that the arrival and departure time of vertices may vary depending upon the insertion order of edges in the graph and starting node of DFS.

Q5. A root vertex of a directed graph is a vertex u with a directed path from u to v for every pair of vertices (u, v) in the graph. In other words, all other vertices in the graph can be reached from the root vertex.

A graph can have multiple root vertices. For example, each vertex in a strongly connected component is a root vertex. In such cases, the solution should return anyone of them. If the graph has no root vertices, the solution should return -1.

The root vertex is 4 since it has a path to every other vertex in the following graph:



Expected output:

```
The root vertex is 4
```

Ans: ASS_Code5.java

We can easily find the root vertex in O(n + m) time using a DFS.

- The idea is to start a DFS procedure from any node of the graph and mark the visited vertices.
- If there are any unvisited vertices, start the DFS again until all vertices are visited.
- Finally, the vertex having the maximum departure time in DFS is a candidate for the root vertex. It's because if let's say there are unreachable nodes or separate components in a graph then those components or nodes would definitely have higher departure time and nodes with less departure time would never be able to reach them . So it becomes higher possibility that node with highest departure time can be a possible candidate for solution
- We can easily check whether that vertex is a root vertex or not by doing a DFS (or BFS) on it.