PRJ

GRAPH THEORY

N26114976 王梓帆

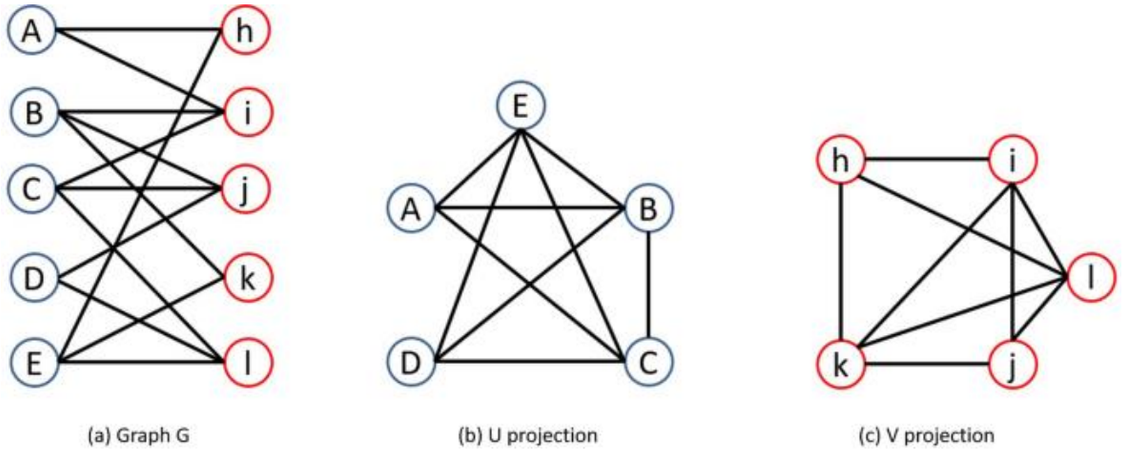
Date: 2022/03/18 Fri.

Outline

- Introduction
- Algorithm
- Code Explanation
- Result and Discussion
- Conclusion

01 章節 PART

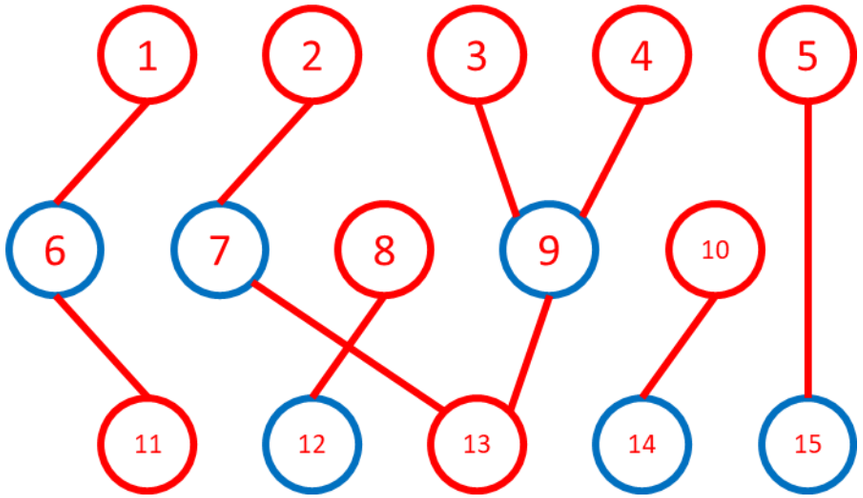
Introduction



(a) Graph G (b) U projection (c) V projection

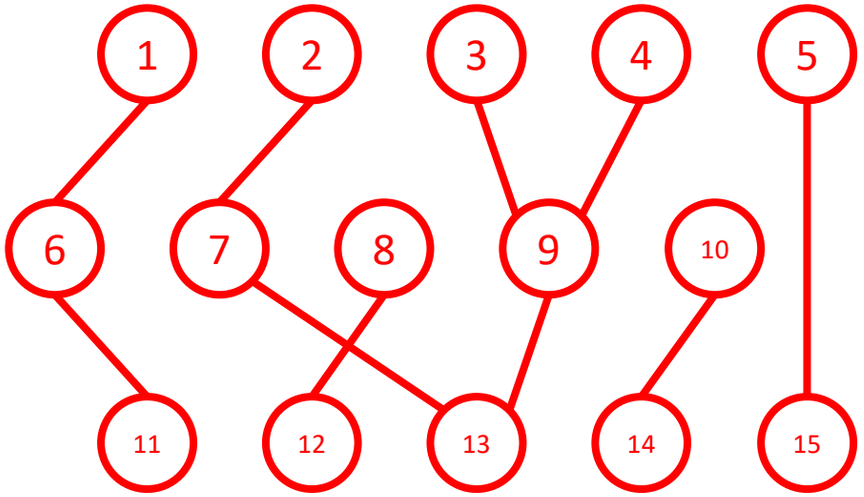
Bipartite Graph

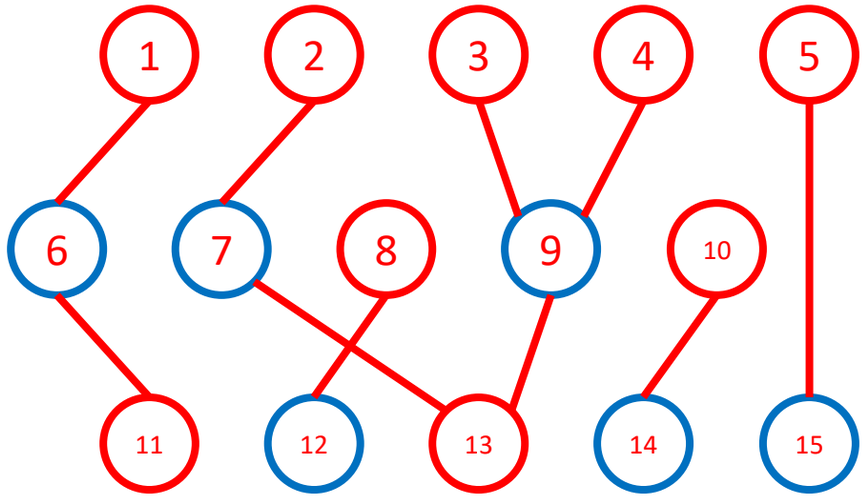
- Chromatic number 2
- $\mathbf{G} = (\mathbf{U}, \mathbf{V}, \mathbf{E})$
- U,V :Independent Sets
- E : Set of Graph Edges



Example: Benchmark_1

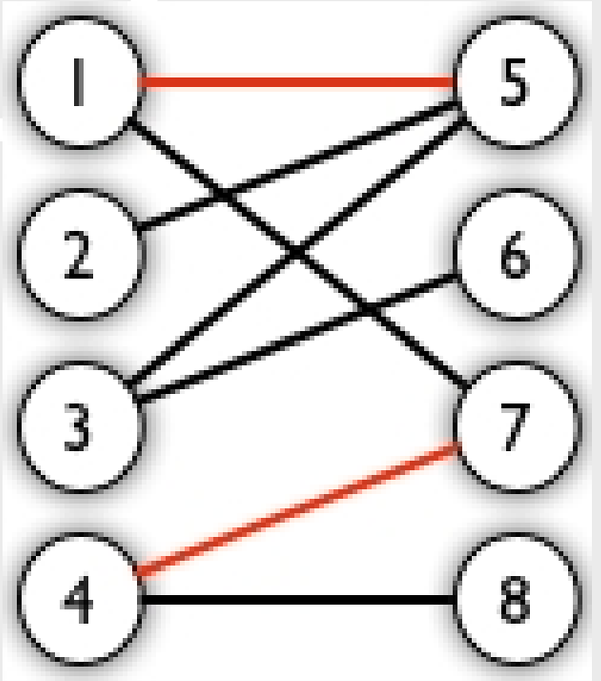
- Adjacent vertices should have different colors.
- Divide a graph into two independent sets.





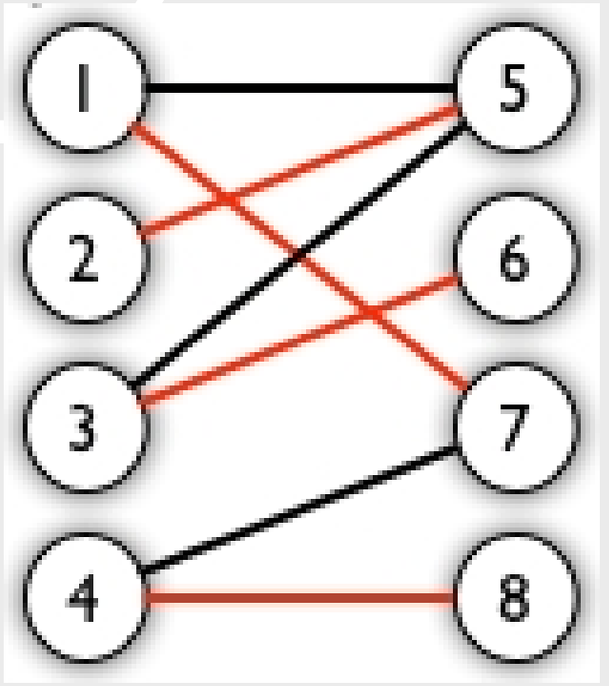
Matching

- Given graph G
- with set of edges E
- One vertex can only be matched once



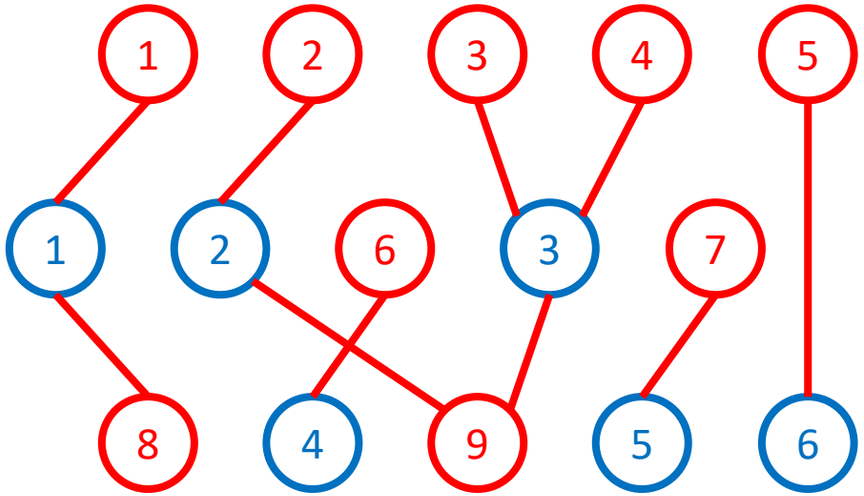
Maximum Matching

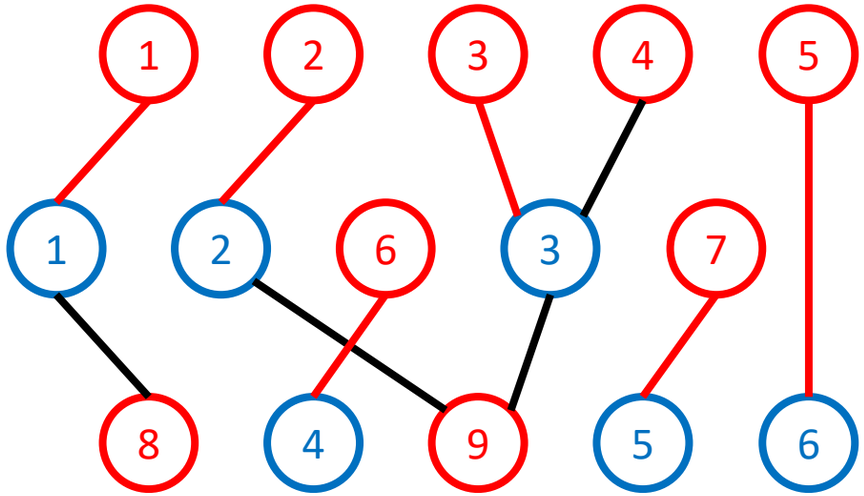
- Given graph G
- with set of edges E
- One vertex can only be matched once



Example: Benchmark_1

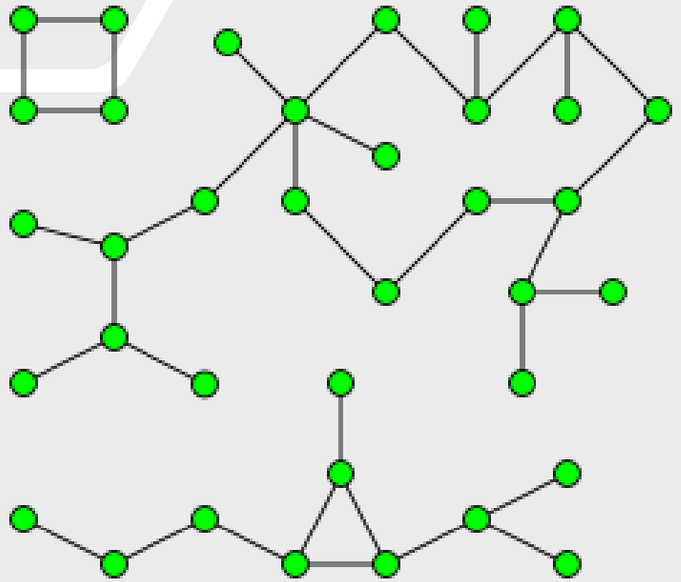
- Adjacent vertices can only be matched by nonmatched vertices.
- The maximum matching of the graph is 6.





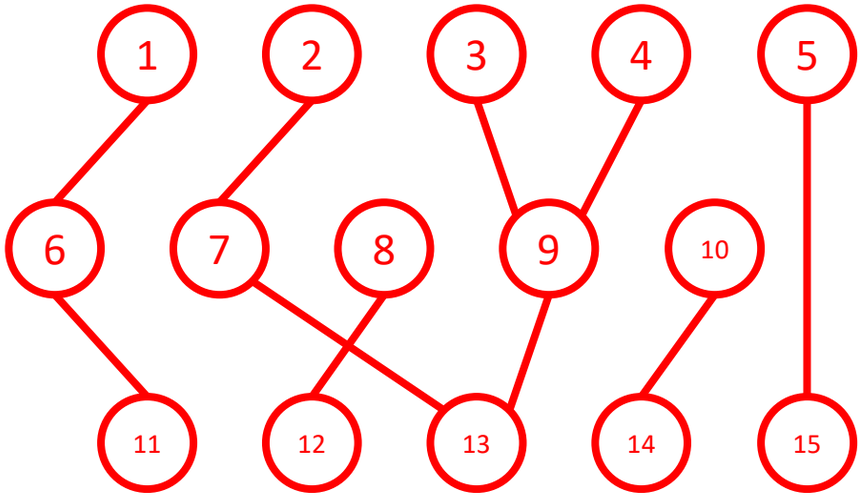
Component

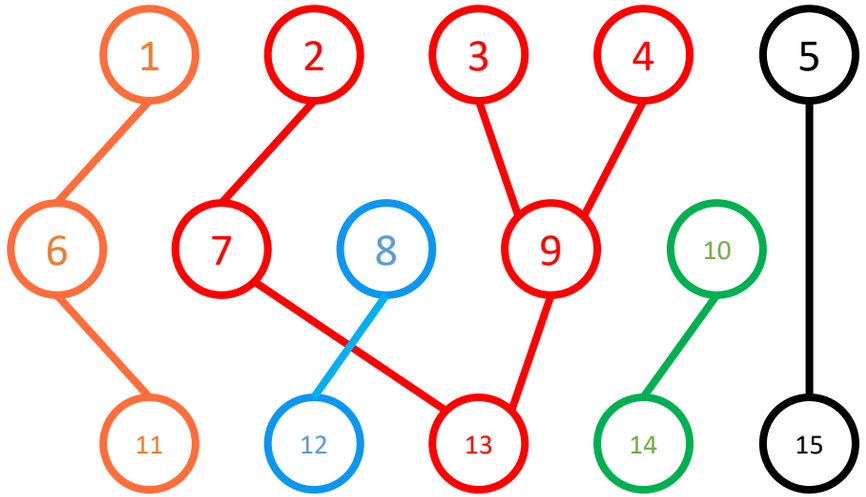
- A connected graph that is not part of any larger connected subgraph.
- The components of any graph partition its vertices into disjoint sets.



Example: Benchmark_1

- Connected vertices should be in one subgraph.
- Divide a graph into 6 disjointed sets.



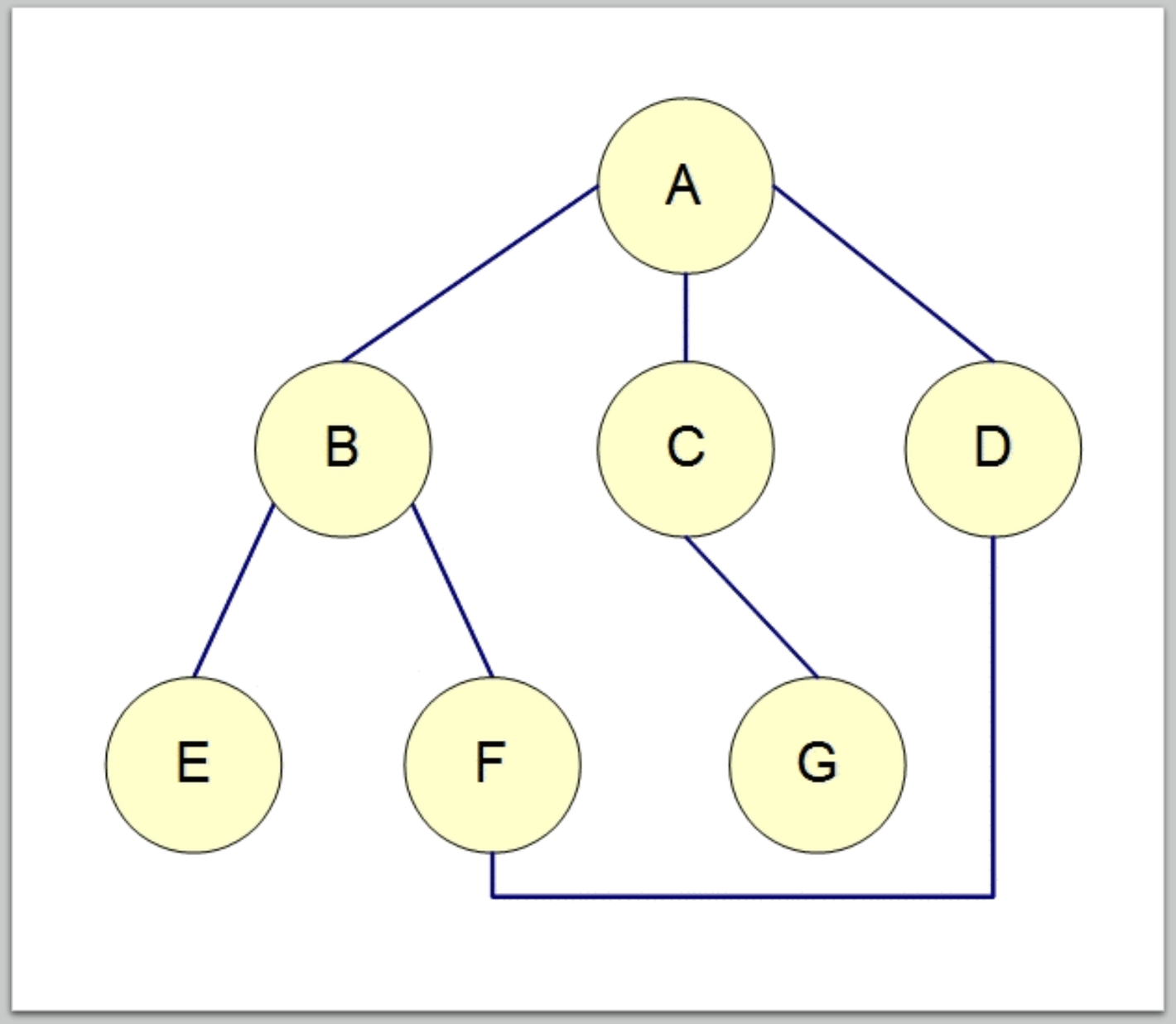


Algorithm

02 章節 PART

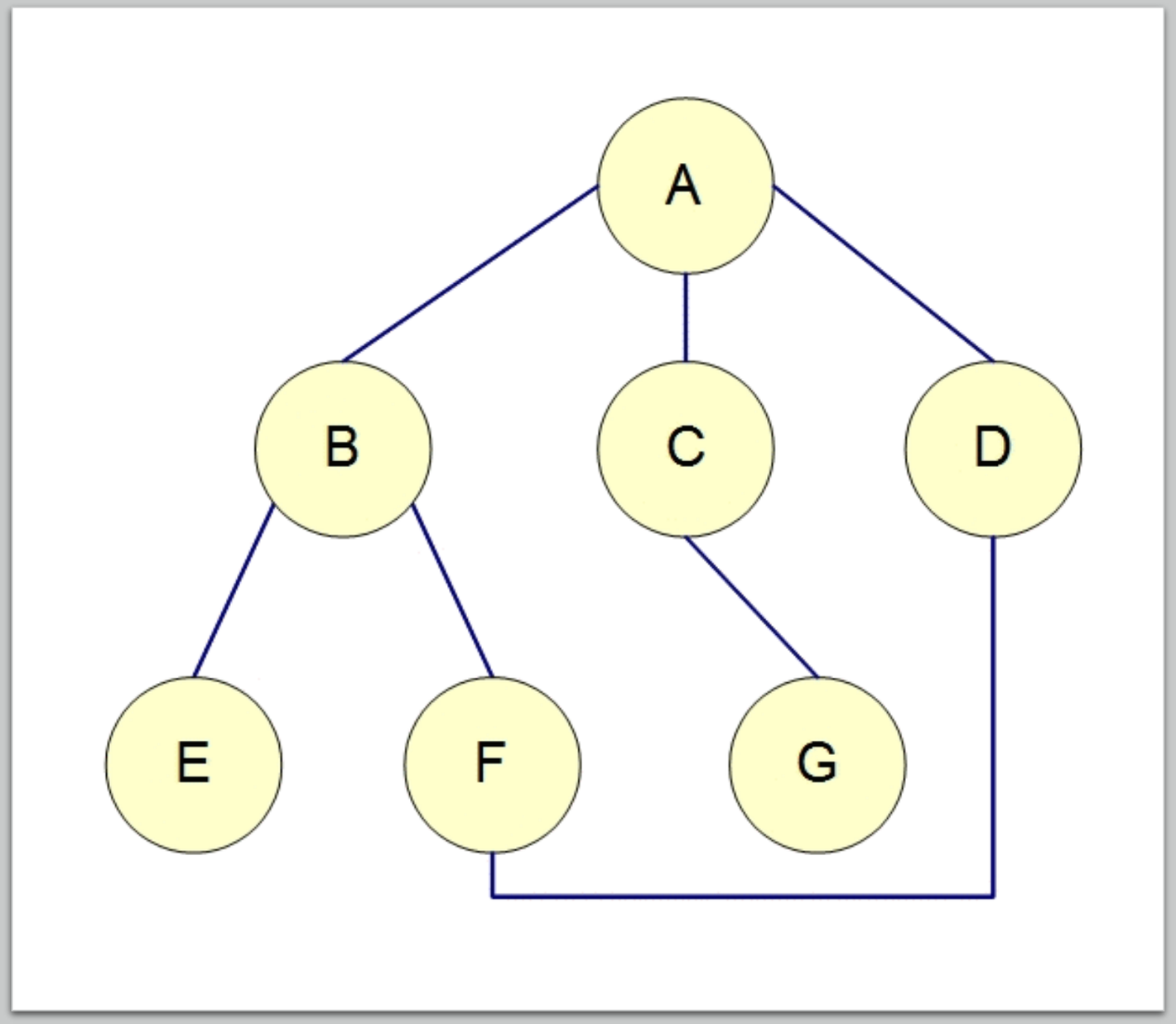
Depth-First Search

- Searching method of a graph.
- Selecting some node as the root in the graph.
- Explores as far as possible along each branch before backtracking.



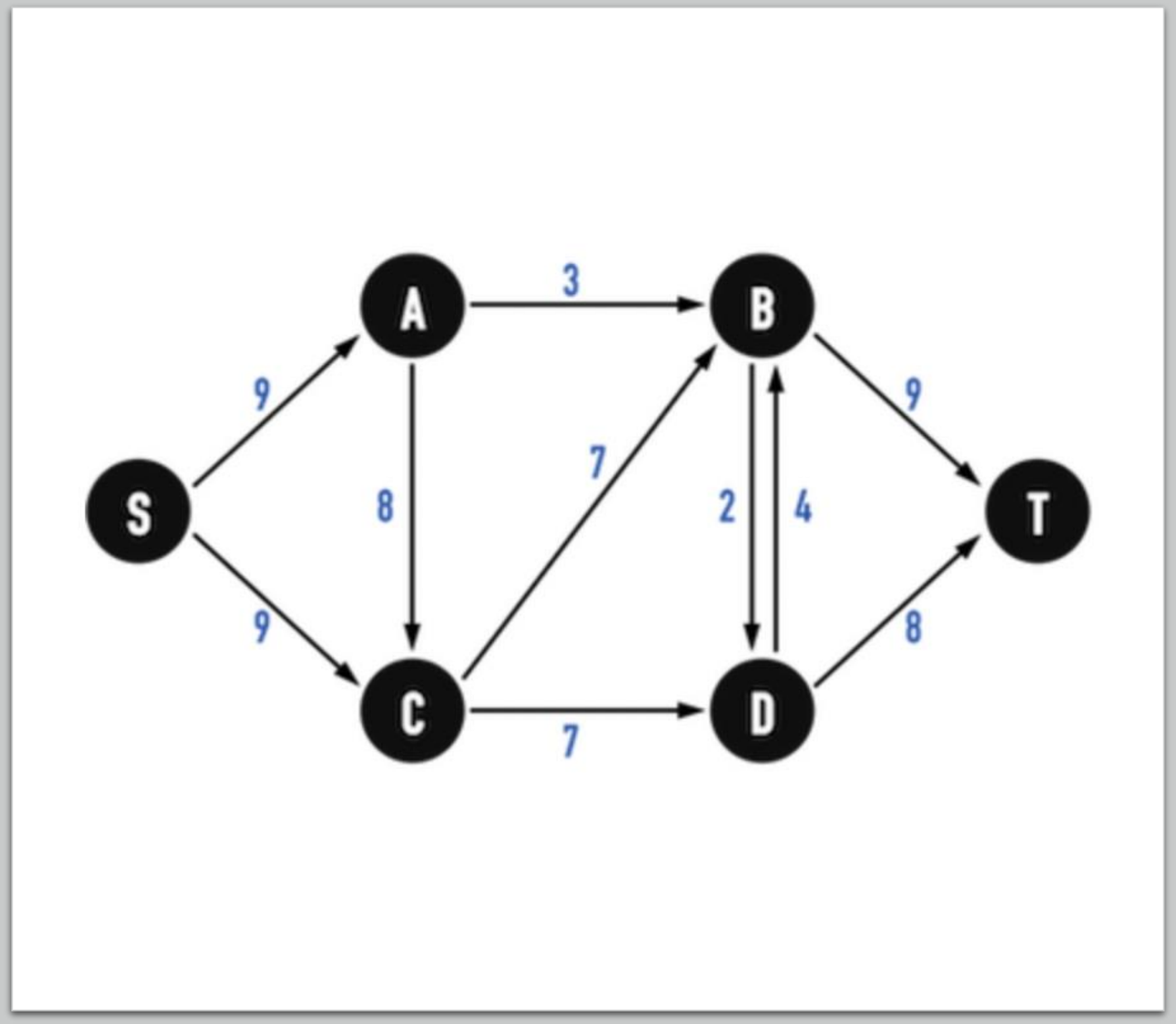
Breadth-First Search

- Searching method of a graph.
- The BFS begins at a root node and inspects all the neighboring nodes.
- Inspects neighbor nodes which were unvisited.



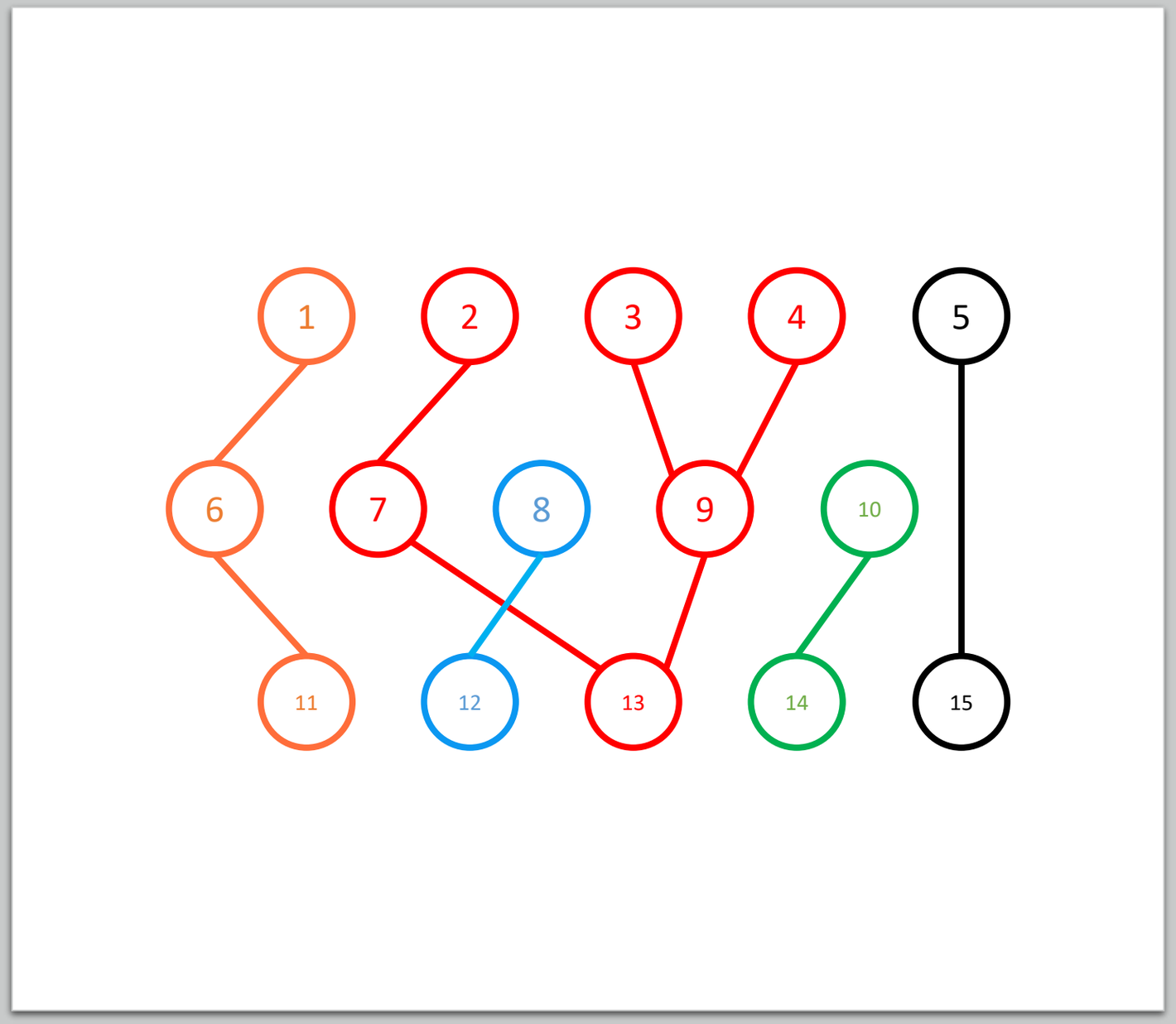
Maximum Flow Algorithm

- Solving flow network problems
- Two special vertices Source and Sink (Termination)
- **Capacity constrain**
- **Skew symmetry**
- **Flow conservation**



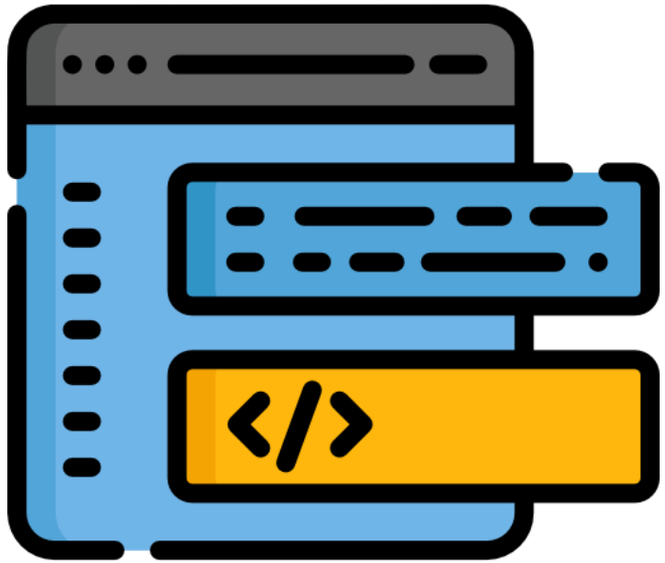
Union Find Algorithm

- Base on Disjoint-set data structure
- Two operations:
  - Union
  - Find



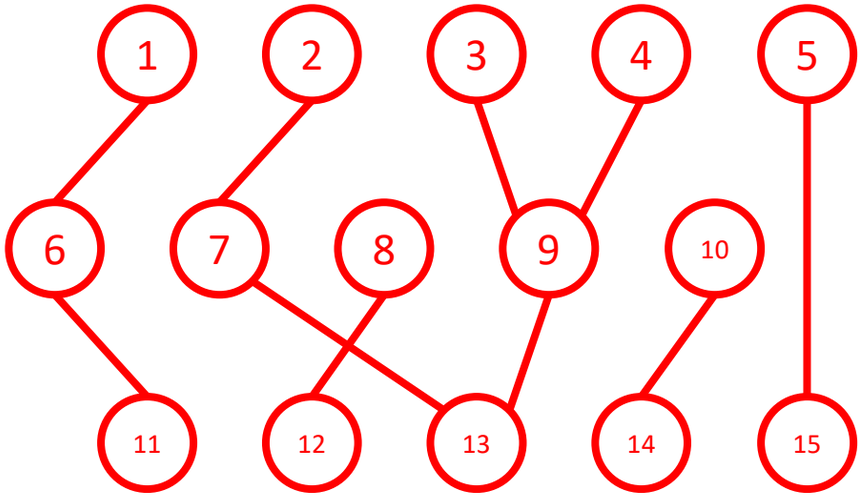
03 章節 PART

Code Explanation



Input Form of Graph

- 2-D Vector with $V * V$
- V: Vertex Number
- V[A][B]: edge between A and B

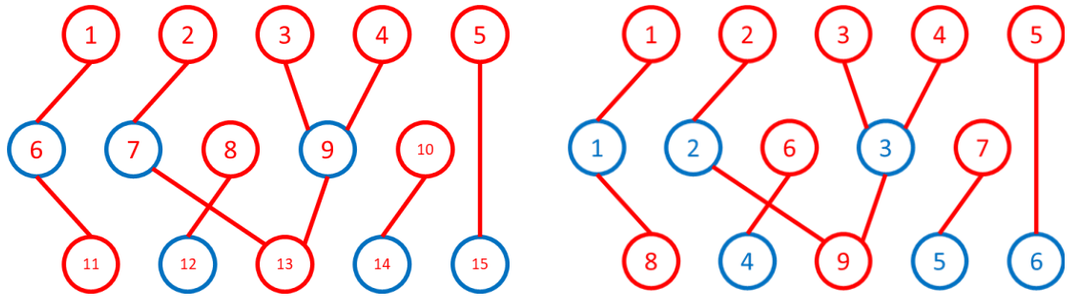



```
vector<vector<bool>> G = { {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
                           {0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
                           {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
```

Input Form of Bipartite Graph

- 2-D Vector with M * N
- M: Red Vertices Number
- N: Blue Vertices Number
- bpGraph[M][N]:

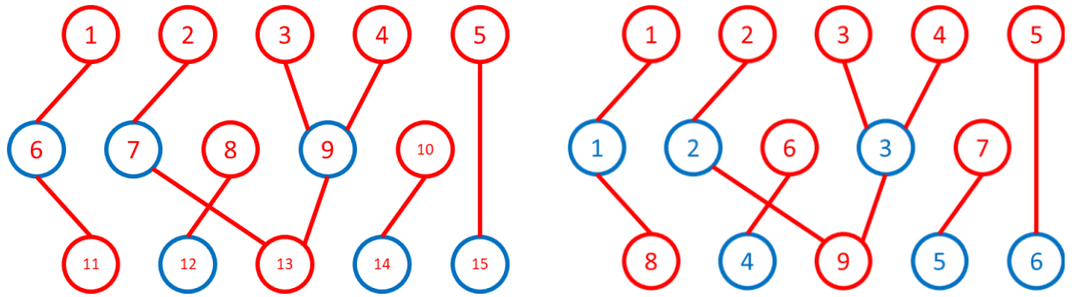
edge between M and N




```
tor<vector<bool>> bpGraph = {{1, 0, 0, 0, 0, 0},
                             {0, 1, 0, 0, 0, 0},
                             {0, 0, 1, 0, 0, 0},
                             {0, 0, 1, 0, 0, 0},
                             {0, 0, 0, 0, 0, 1},
                             {0, 0, 0, 1, 0, 0},
                             {0, 0, 0, 0, 1, 0},
                             {1, 0, 0, 0, 0, 0},
                             {0, 1, 1, 0, 0, 0}};
```

Input Form of edges

- 2-D Vector with N * 2
- N: Edges Number
- 2: Vertices of the edge
- From 0 to V - 1




```
vector<vector<int>> edges = {{0, 5},
                             {1, 6},
                             {2, 8},
                             {3, 8},
                             {4, 14},
                             {5, 10},
                             {6, 12},
                             {7, 11},
                             {8, 12},
                             {9, 13}};
```

Unified Modeling Language Chart

Solution
+: bool isBipartite_BFS(vector<vector<bool>>, int)
+: isBipartite_DFS(vector<vector<bool>>)
+: bool bpm(vector<vector<bool>>, int, bool, int)
+: int maxBPM(vector<vector<bool>>)
+: int merge(vector<int>, int)
+: int connectedcomponents(int, vector<vector<int> >&, vector<vector<bool> >)
+: static vector<int> V_color;

Depth-First Search

將起點 source 著成紅色

當前的點著色為 c，與該點相鄰的下一點著色為 1 - c

如果已經著色，且被著成同一個顏色，就判斷非二部圖

如果成功便利所有資料，都沒有產生錯誤，則判斷為二部圖

Breadth-First Search

將起點 source 著成紅色

創造一個存放頂點的 queue 用來進行 BFS 演算法的遍歷

只要 queue 裡還有頂點就持續判斷

先把 queue 裡的頂點拿出來另存，並把該點從 queue 中移除

找出所有尚未被塗色的相鄰頂點

- 給定的圖 G edge 存在，而且 v 點尚未被著色
- 相鄰兩點不同色
- 把新被著色的點放到 queue 裡
- 判斷 v 點是否已被著上與 u 點同樣的顏色

Maximum Matching Number

Bipartite Match：如果和點 u 可能匹配（和點 u 相連），則 bpm 判斷為 True

• 將所有著藍色的點拿來做測試
• 如果紅色的點 u，和藍色的點 v 相連，且 v 尚未被其他點相連
  • 先將 v 標示為已配對
  • 如果藍色的點 v 沒有被其他點相連，或者先前與藍色的點相連的紅點（即 matchR[v]）有其他可以相連的點。
  • 由於 v 在上一行中被標記為已配對，因此 matchR[v] 將不會再次遞迴點 v

Maximum Match：回傳最大匹配數

• 儲存與藍色的點相連的紅點 matchR[i] 是與藍色的點 i 相連的紅點數若為 -1 則該點沒有被相連
• 初始條件：所有的 v 點都可以被連線
  • 對下一個紅點 u 初始化為尚未連接
  • 判斷紅點 u 是否匹配，有的話匹配數加一。

Number of Connected Components

定義一個大小為 G.size() 的數組 parent，其中 G.size() 是節點的總數。

對於數組 parent 的每個索引 i，該值表示第 i 個頂點的父節點是誰。 比如 parent[1] = 3，那麼我們可以說頂點 1 的 parent 節點是 3

將每個節點初始化為自身的 parent 節點，然後在將它們相加的同時，相應地更改它們的 parent 節點。

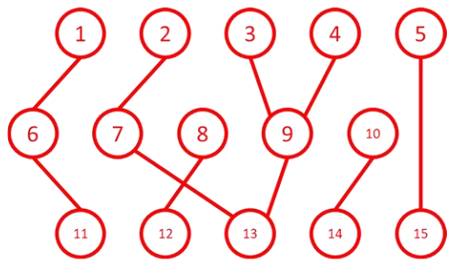
Result and Discussion

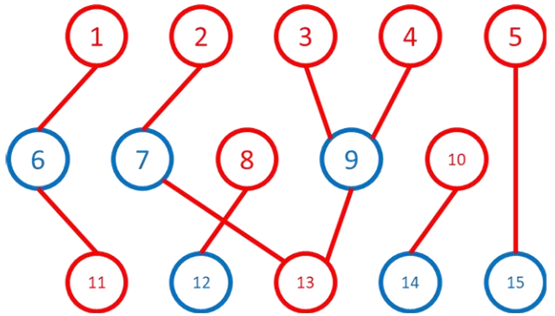
04 章節 PART

Output Form

- Determine whether graph is BIPARTITE
  - BFS
  - DFS
- Two Subgraph
- Maximum Matching
- Component Numbers and Members

INPUT

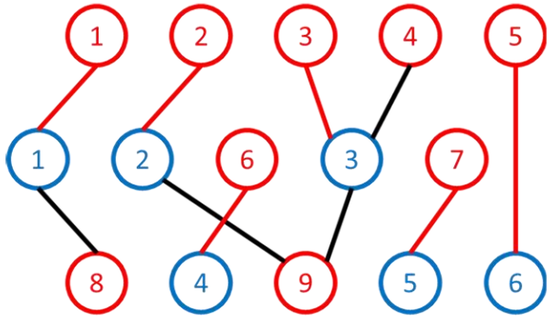


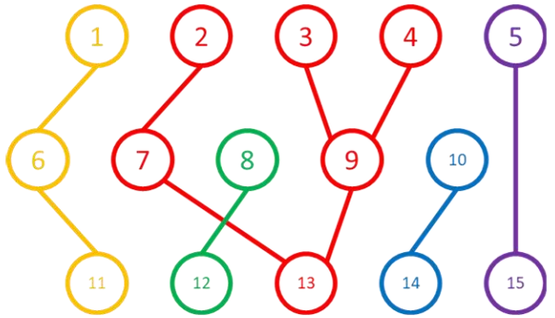


IS BIPARTITE!

Shortest Time: 37.2 us

```
$ ./GT_Project_1.exe
Which graph would you like to input:
1
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0373ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.2529ms
----------------------------
Red Group Members: 1,2,3,4,5,8,10,11,13
Blue Group Members: 6,7,9,12,14,15
----------------------------
Maximum Matching is 6
----------------------------
Components => 1 6 11
Components => 8 12
Components => 2 3 4 7 9 13
Components => 10 14
Components => 5 15
----------------------------
There are 5 components.
============================
```





LAB

CONCLUSION

王梓帆

Date: 2021/12/16 Thu.