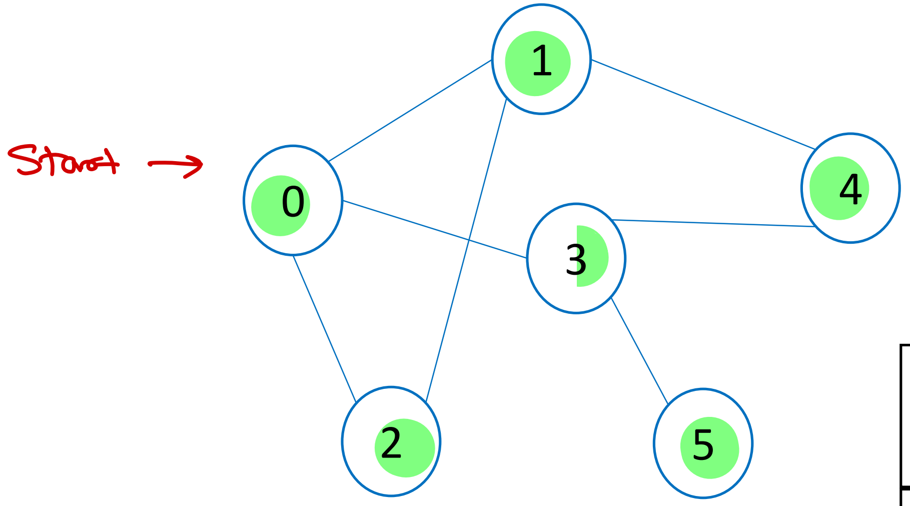
Data Structure & Algorithms

Nilesh Ghule

Graph Traversal – DFS Algorithm

1. Choose a vertex as start vertex.
2. Push start vertex on stack & mark it.
3. Pop vertex from stack.
4. Visit (Print) the vertex.
5. Put all non-visited neighbours of the vertex on the stack and mark them.
6. Repeat 3-5 until stack is empty.



	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	1	0	1	0
2	1	1	0	0	0	0
trav → 3	1	0	0	0	1	1
4	0	1	0	1	0	0
5	0	0	0	1	0	0

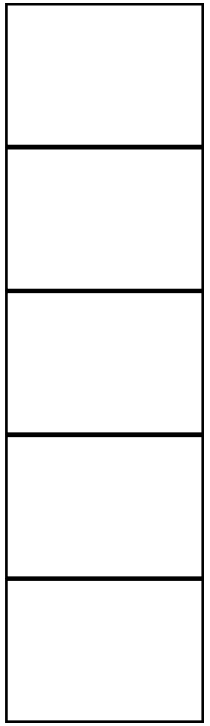
```
for(v=0; v< vCount; v++) {
    if (adjmat[trav][v]==1)
        // v is neighbour of trav.
```

3

boolean marked[]

0	1	2	3	4	5
✓	✓	✓	✓	✓	✓

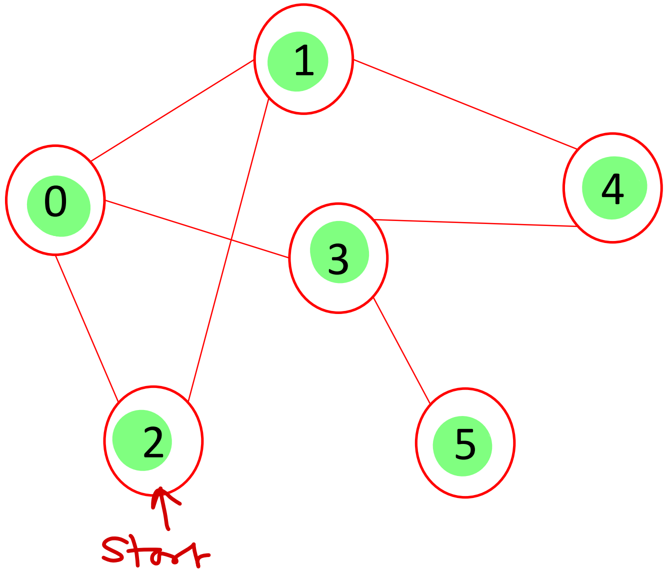
0 3 5 4 2 1

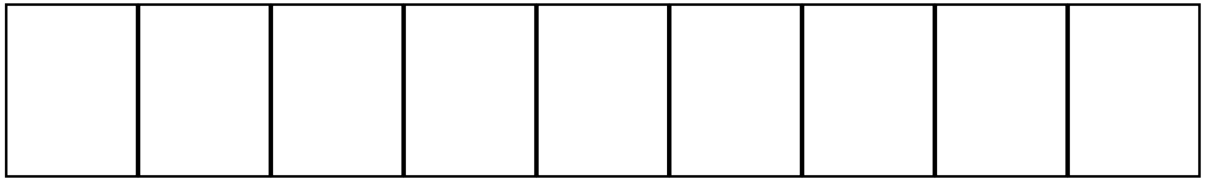


Graph Traversal – BFS Algorithm

1. Choose a vertex as start vertex.
2. Push start vertex on queue & mark it.
3. Pop vertex from queue.
4. Visit (Print) the vertex.
5. Put all non-visited neighbours of the vertex on the queue and mark them.
6. Repeat 3-5 until queue is empty.

- BFS is also referred as level-wise search algorithm.



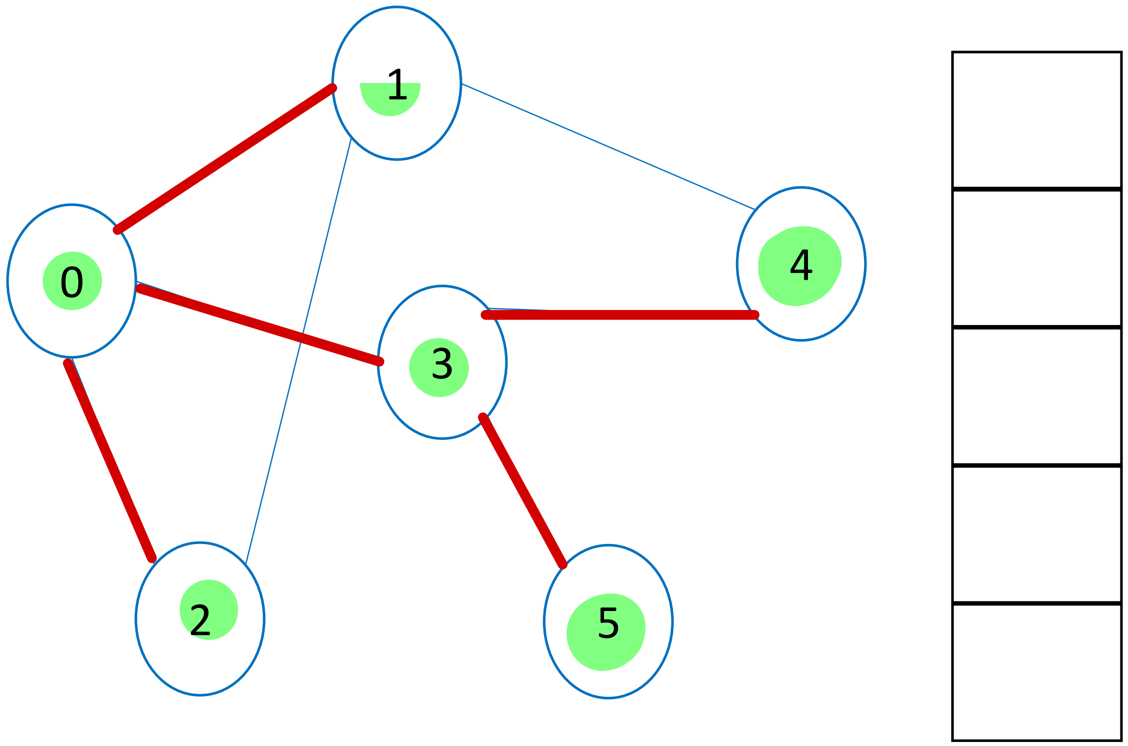


2 0 1 3 4 5

DFS Spanning Tree

1. push starting vertex on stack & mark it.
2. pop the vertex.
3. push all its non-marked neighbors on the stack, mark them. Also print the vertex to neighboring vertex edges.
4. repeat steps 2-3 until stack is empty.

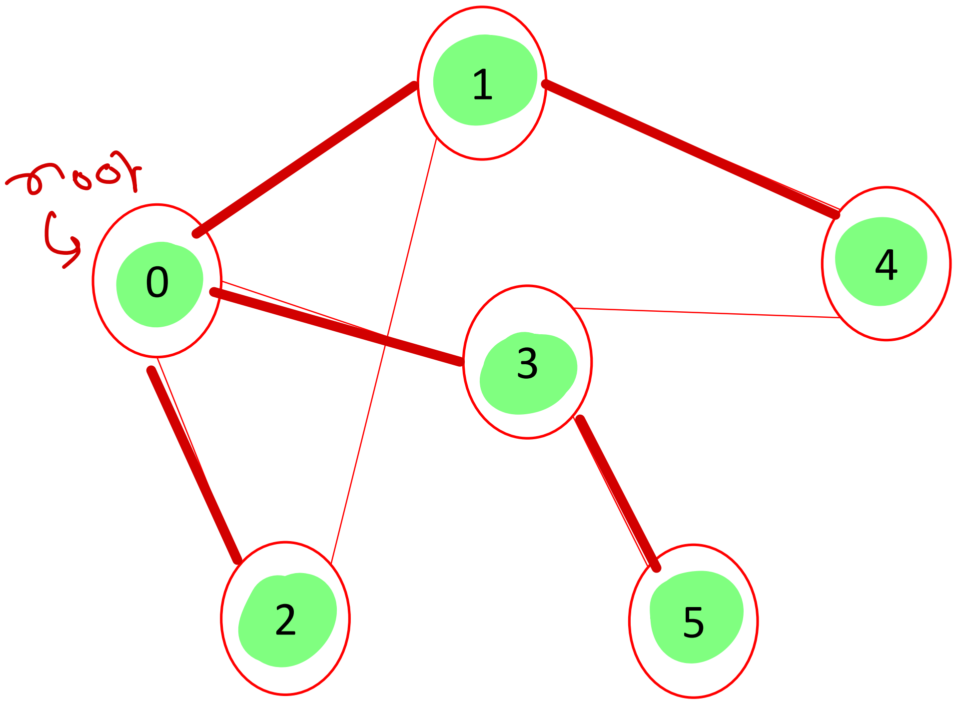
0-1
0-2
0-3
3-4
3-5



0 3 5 4 2 1

BFS Spanning Tree

1. push starting vertex on queue & mark it.
2. pop the vertex.
3. push all its non-marked neighbors on the queue, mark them. Also print the vertex to neighboring vertex edges.
4. repeat steps 2-3 until queue is empty.

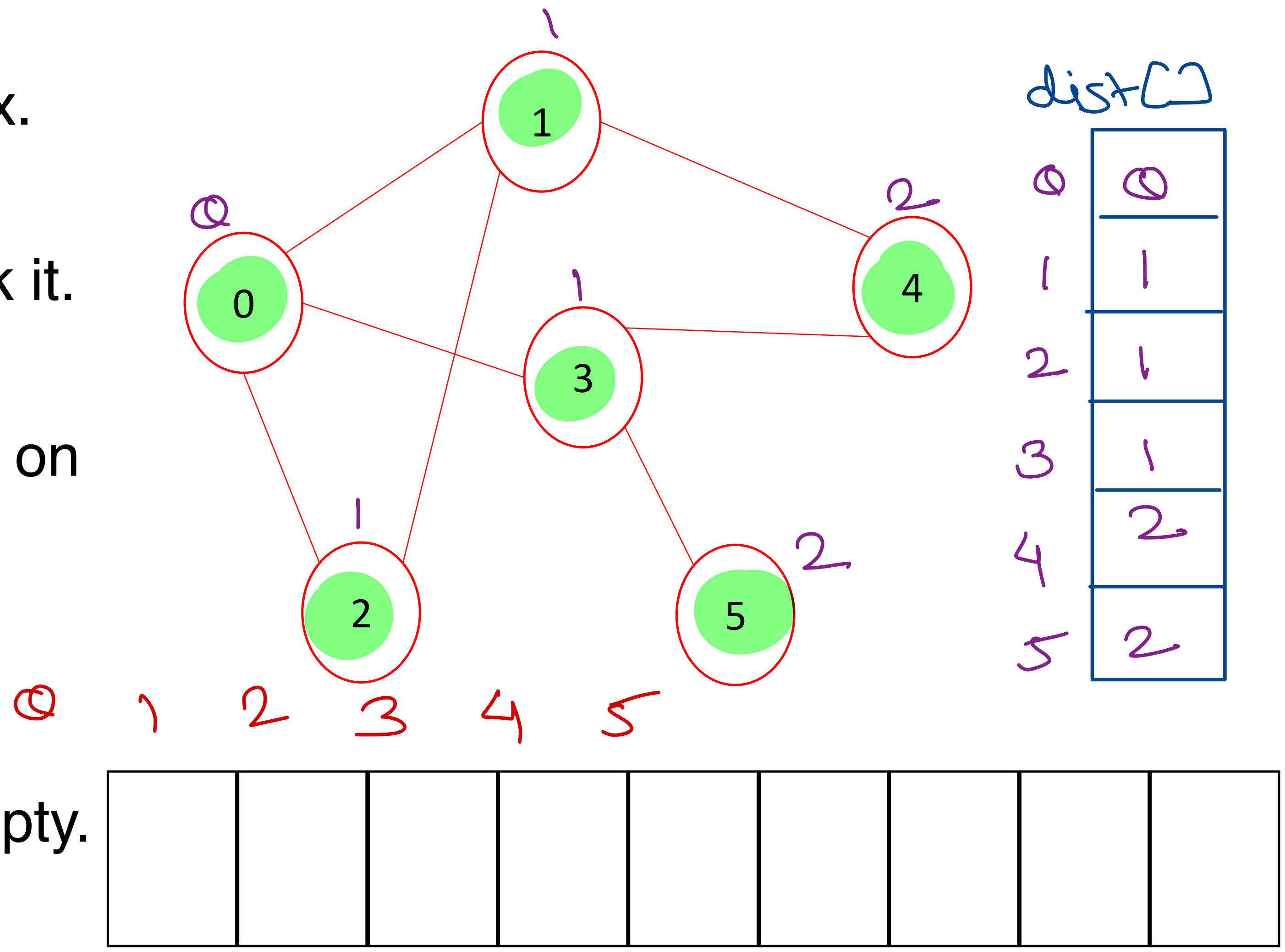


0 - 1
0 - 2
0 - 3
1 - 4
3 - 5

0 1 2 3 4 5

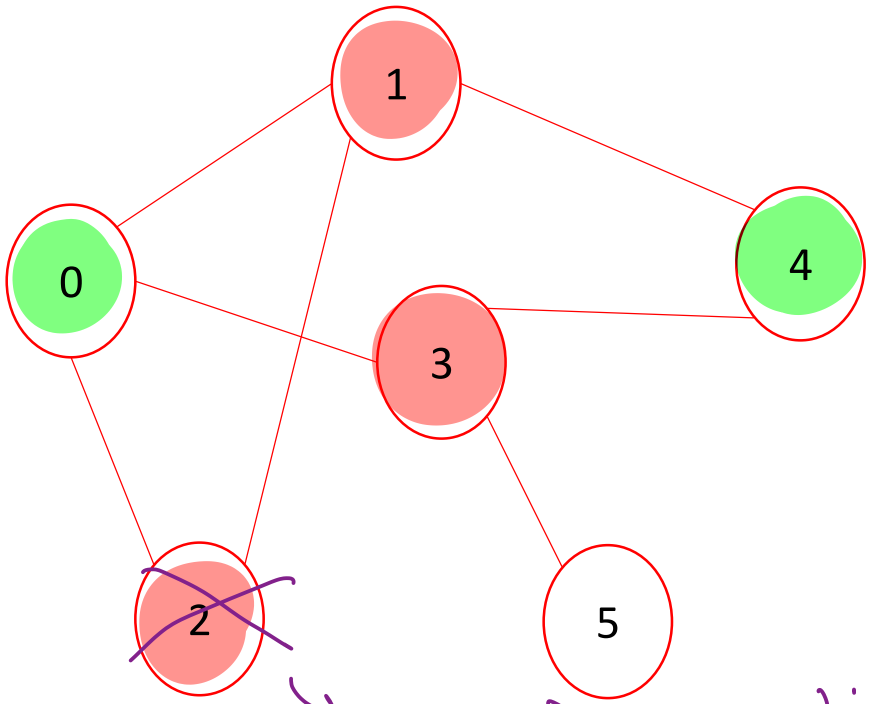
Single Source Path Length

1. Create path length array to keep distance of vertex from start vertex.
2. Consider dist of start vertex as 0.
3. push start vertex on queue & mark it.
4. pop the vertex.
5. push all its non-marked neighbors on the queue, mark them.
6. For each such vertex calculate its distance as dist[neighbor] = dist[current] + 1
7. repeat steps 3-6 until queue is empty.
8. Print path length array.



Check Bipartite-ness

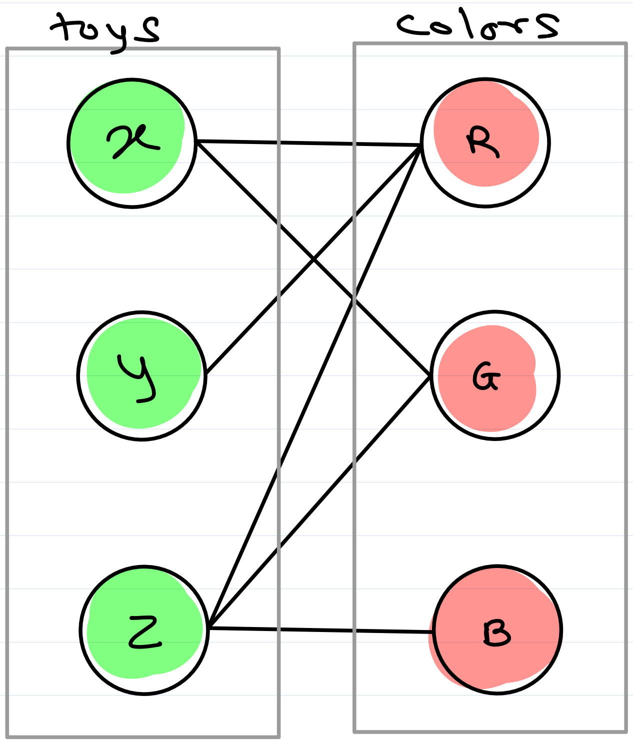
1. keep colors of all vertices in an array. Initially vertices have no color.
2. push start on queue & mark it. Assign it color1.
3. pop the vertex.
4. push all its non-marked neighbors on the queue, mark them.
5. For each such vertex if no color is assigned yet, assign opposite color of current vertex (c1-c2, c2-c1).
6. If vertex is already colored with same of current vertex, graph is not bipartite (return).
7. repeat steps 3-6 until queue is empty.



→ graph is not bipartite

0 1

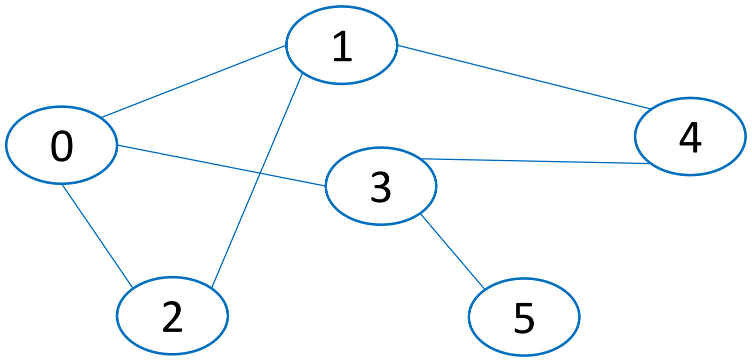
	2	3	4					

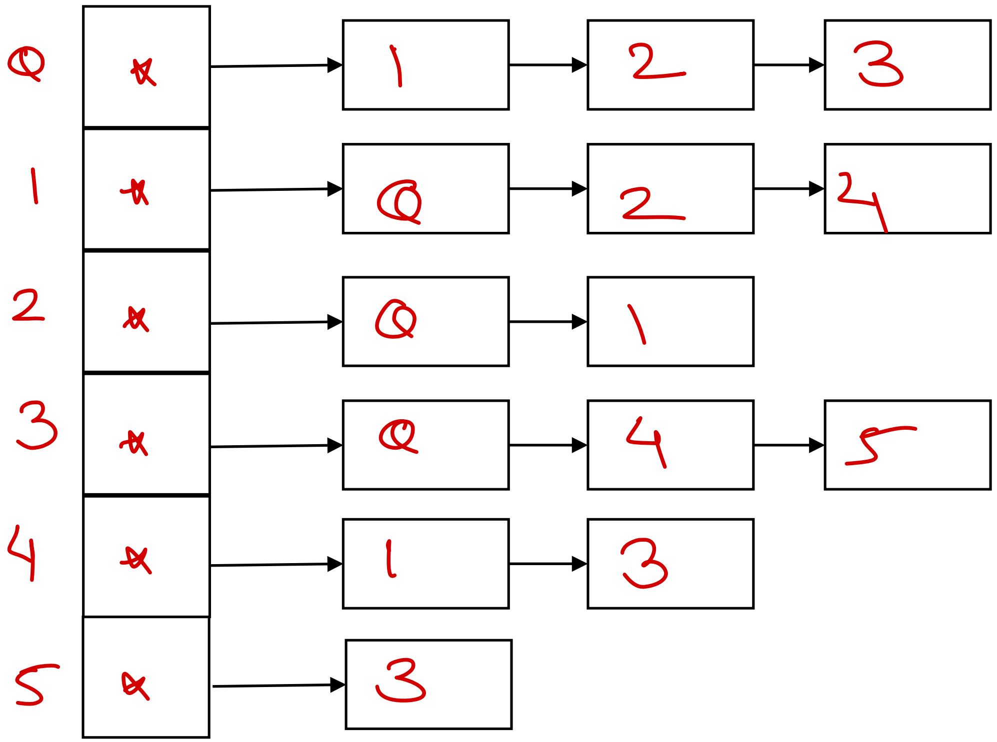


x R G y z B

Graph Implementation – Adjacency List

- Each vertex holds list of its adjacent vertices.
- For non-weighted graphs only, neighbour vertices are stored.
- For weighted graph, neighbour vertices and weights of connecting edges are stored.
- Space complexity of this implementation is O(V*E).
- If graph is sparse graph (with fewer number of edges), this implementation is more efficient (as compared to adjacency matrix method).





Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>