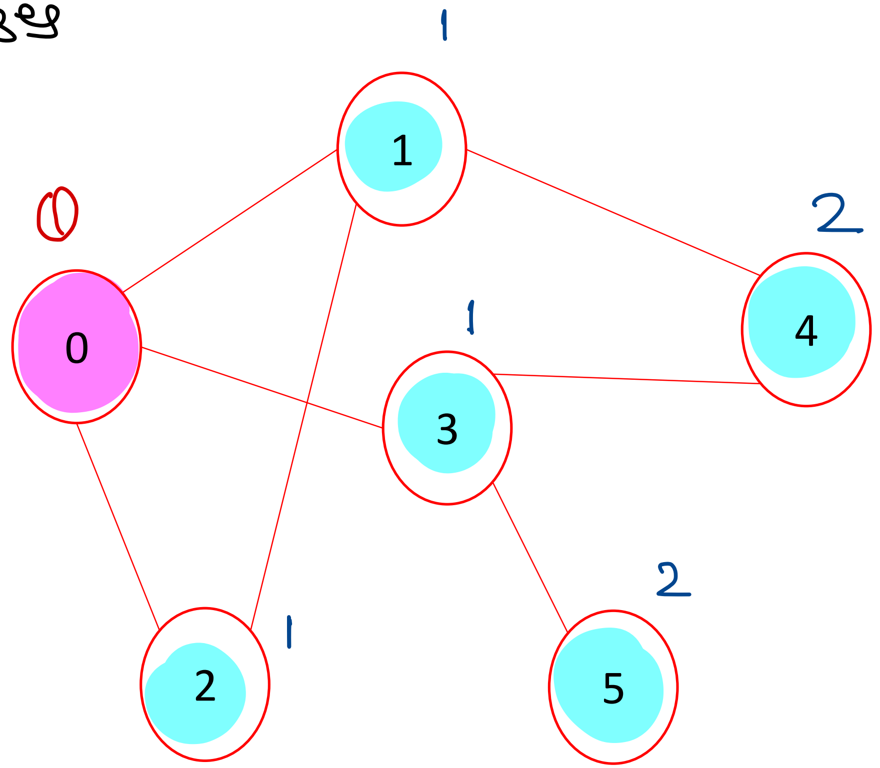
Graph Data Structure & Algorithms

Sunbeam Infotech

Single Source Path Length - non-weighted graph

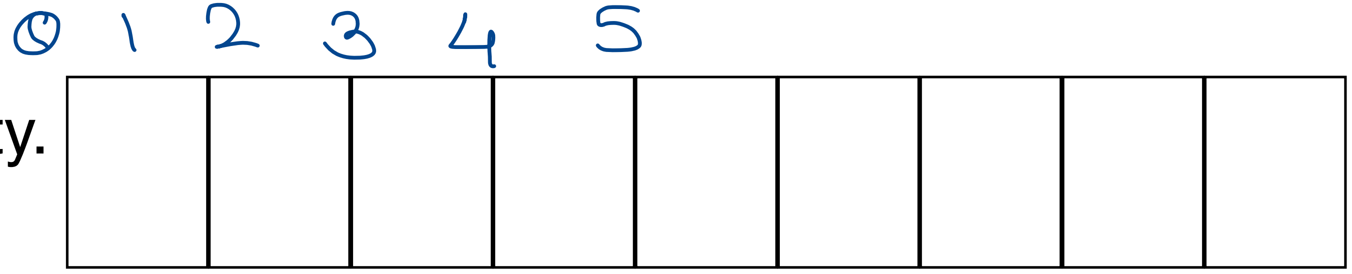
↳ from one vertex ↳ number of edges

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.



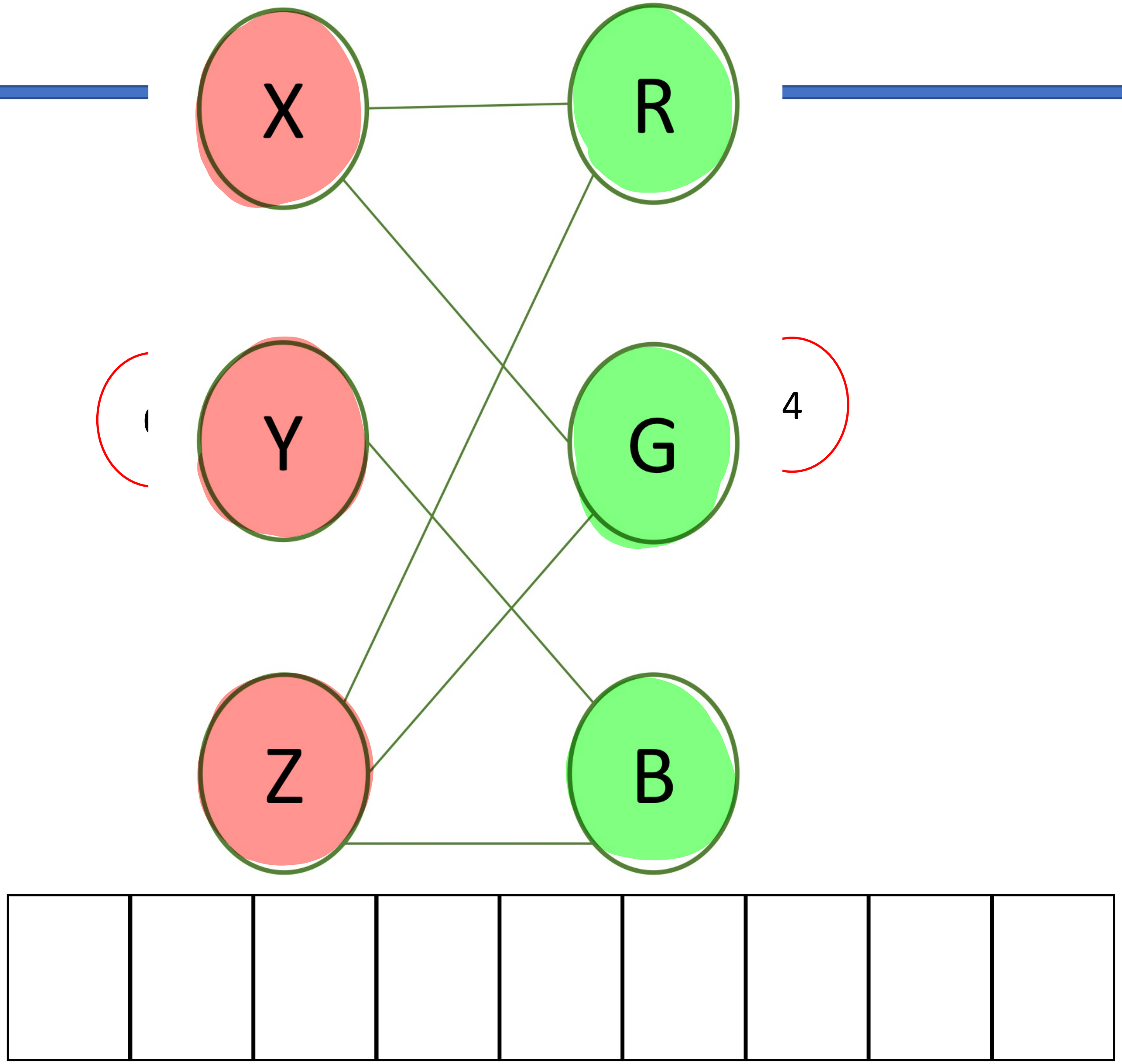
dist

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



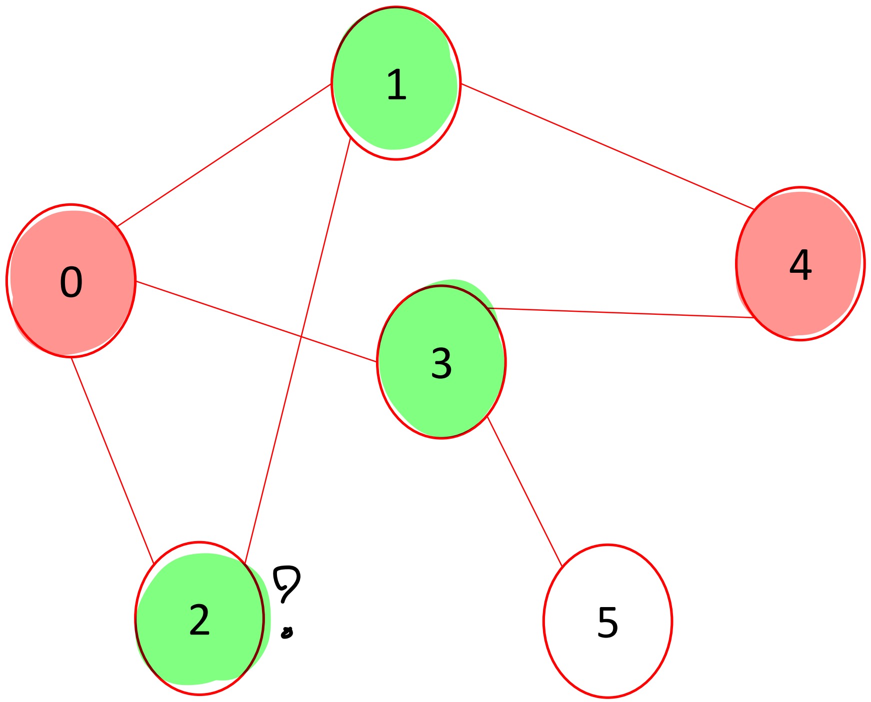
Check Bipartite-ness

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.



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.



Time Complexity - BFS / DFS

Adj Matrix Impl

outer loop → $O(V)$

inner loop → $O(V)$

Time Complexity → $O(V^2)$

Adj List Impl

outer loop → $O(V)$

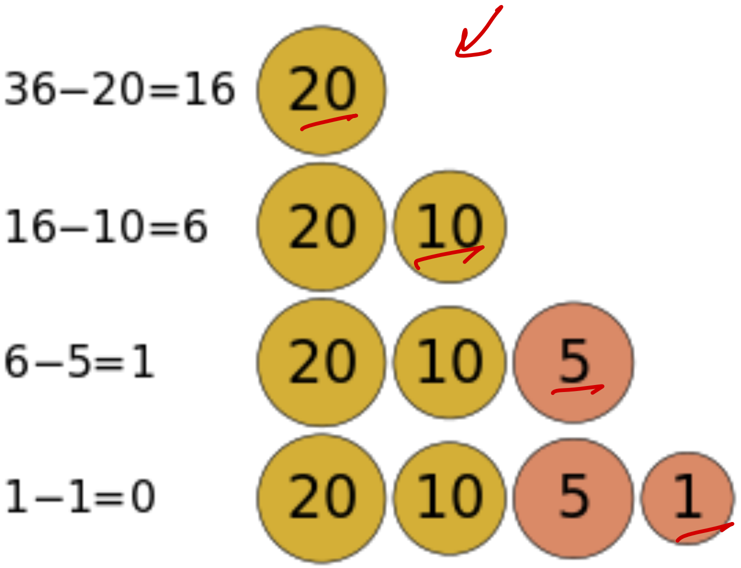
inner loop → for all vertices

$O(E)$

Time Complexity → $O(V+E)$

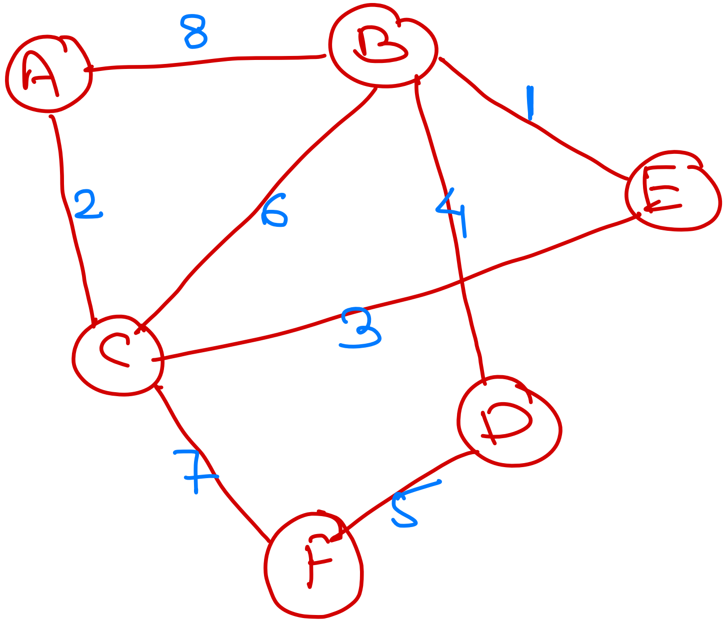
Problem solving technique: Greedy approach

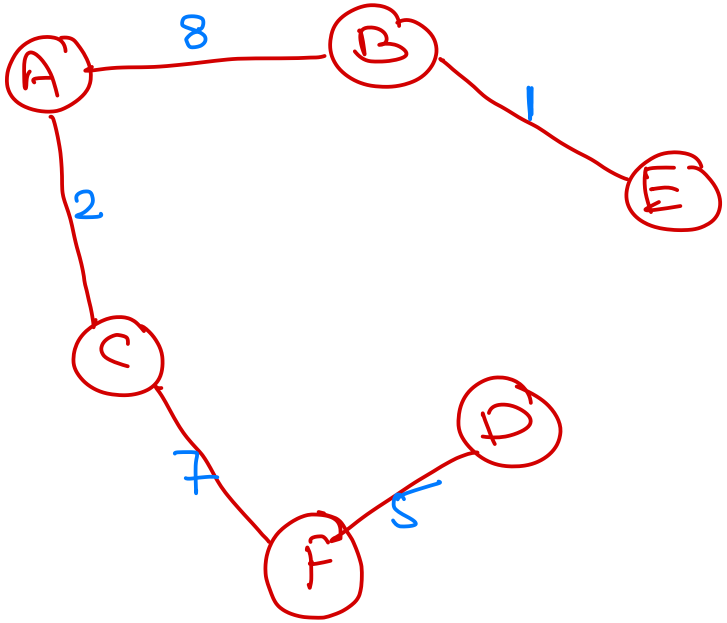
- A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
- We can make choice that seems best at the moment and then solve the sub-problems that arise later.
- The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub-problem.
- It iteratively makes one greedy choice after another, reducing each given problem into a smaller one.
- A greedy algorithm never reconsiders its choices.
- A greedy strategy may not always produce an optimal solution.



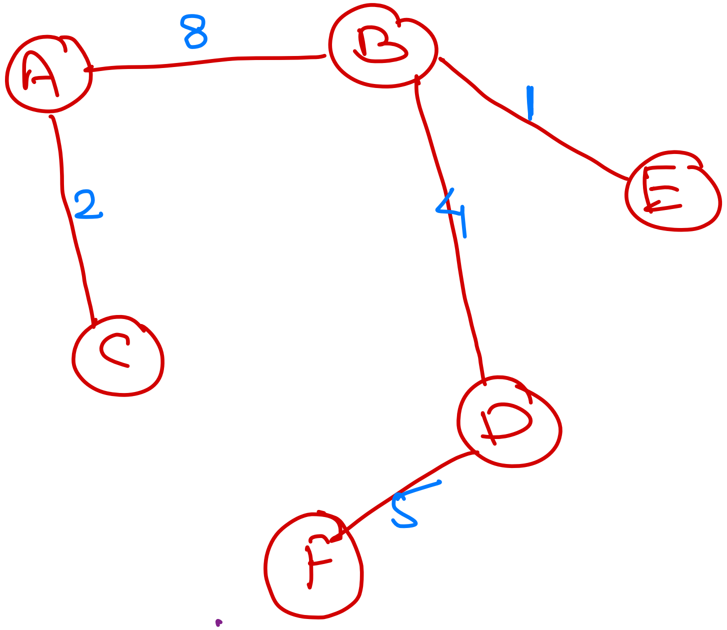
- Greedy algorithm decides minimum number of coins to give while making change.

Min Spanning Tree → spanning tree whose total weight is less than all other spanning trees.





weight = 23



weight = 20

Applications

* Optimal resource planning
* Travelling Salesman Problem
* Road making

Spanning Tree

V vertices
V-1 edges
no cycles

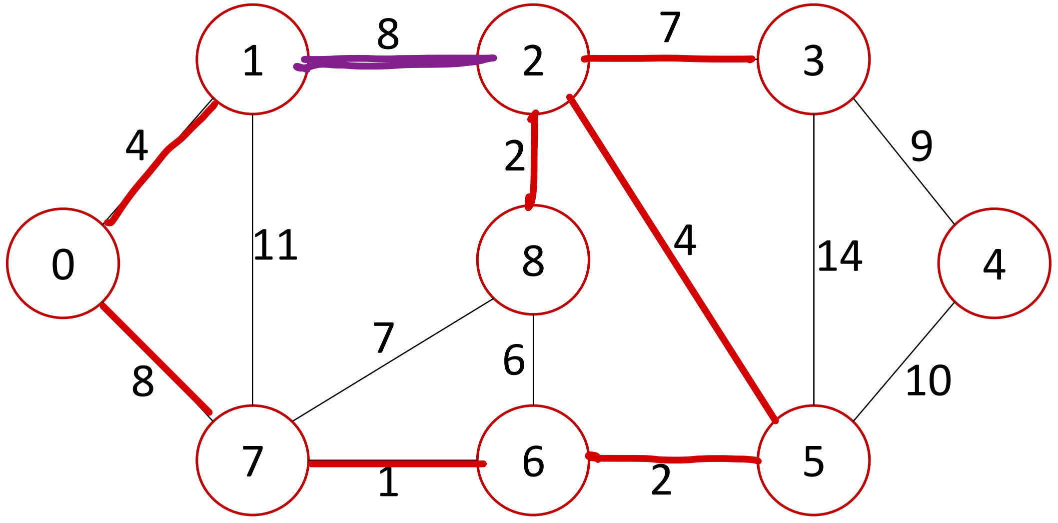
Algorithms

1. Kruskal
2. Prim

Union Find Algorithm

1. Consider all vertices as disjoint sets (parent = -1).
2. For each edge in the graph
   1. Find set of first vertex.
   2. Find set of second vertex.
   3. If both are in same set, cycle is detected.
   4. Otherwise, merge both the sets i.e. add root of first set under second set

parent[sr] = dr;



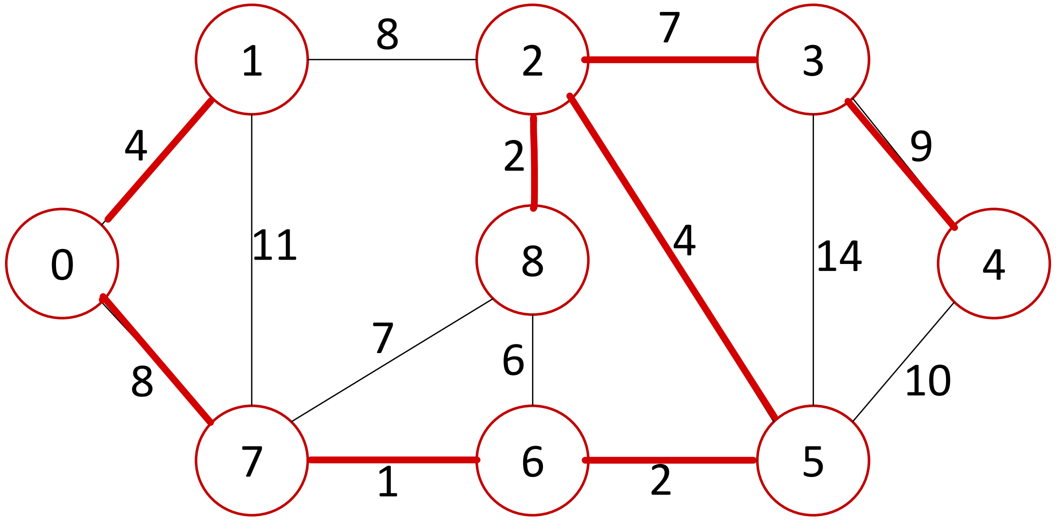
parent

0	1	2	3	4	5	6	7	8
1	3	5			3	5	6	2

src	des	wt
✓ 7	6	1
✓ 8	2	2
✓ 6	5	2
✓ 0	1	4
✓ 2	5	4
~~8~~	~~6~~	6
✓ 2	3	7
~~7~~	~~8~~	7
✓ 0	7	8
✓ 1	2	8
3	4	9
5	4	10
1	7	11
3	5	14

Kruskal’s MST

1. Sort all the edges in ascending order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spanning tree.



	src	des	wt
✓	7	6	1
✓	8	2	2
✓	6	5	2
✓	0	1	4
✓	2	5	4
✗	8	6	6
✓	2	3	7
✗	7	8	7
✓	0	7	8
✗	1	2	8
✓	3	4	9
	5	4	10
	1	7	11
	3	5	14

Thank you!

Nilesh Ghule <Nilesh@sunbeaminfo.com>