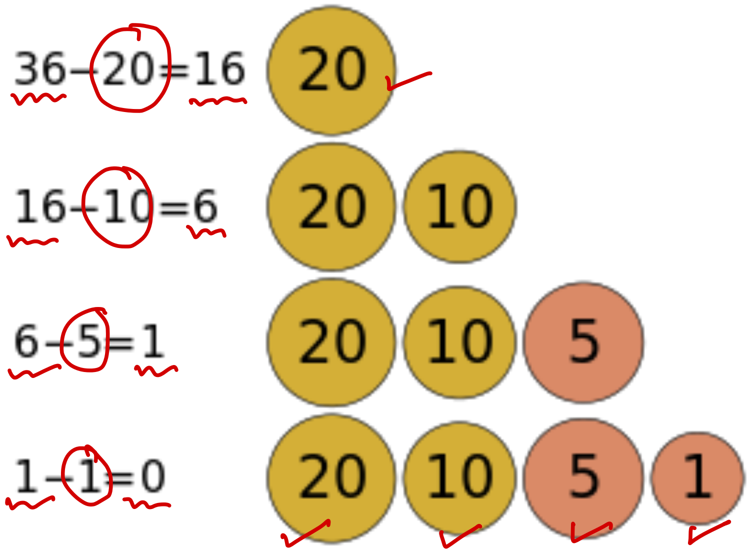
Data Structure & Algorithms

Nilesh Ghule

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.

1, 2, 5, 10, 20, 50, 100

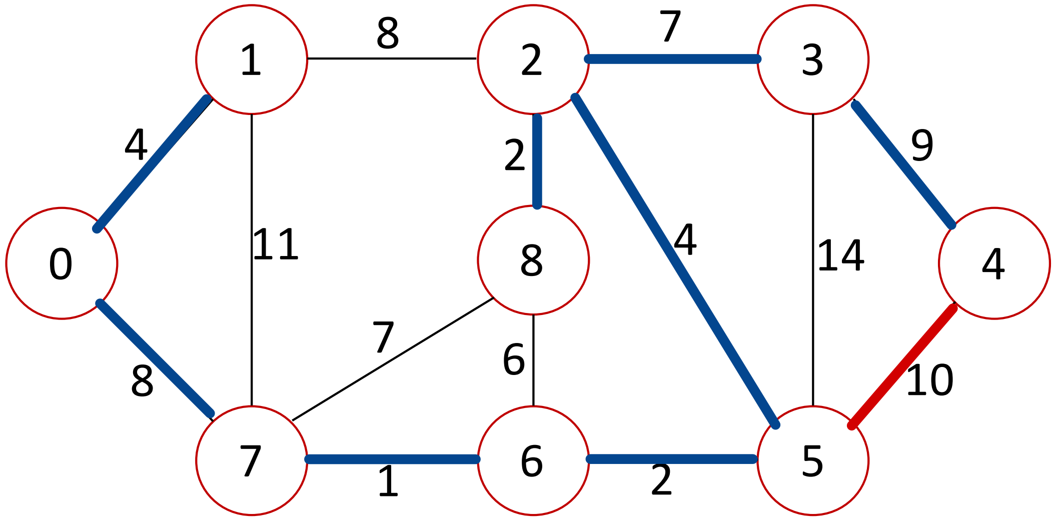


- Greedy algorithm decides minimum number of coins to give while making change.

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 [srcRoot] = destRoot;



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

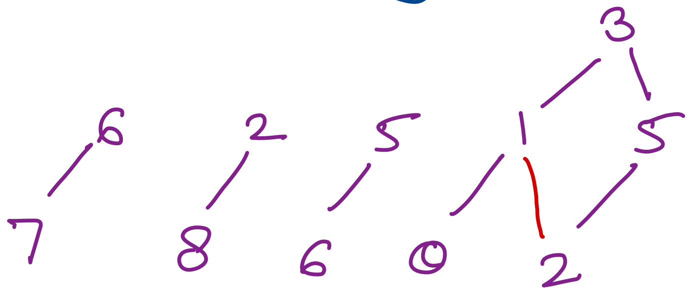
parent []

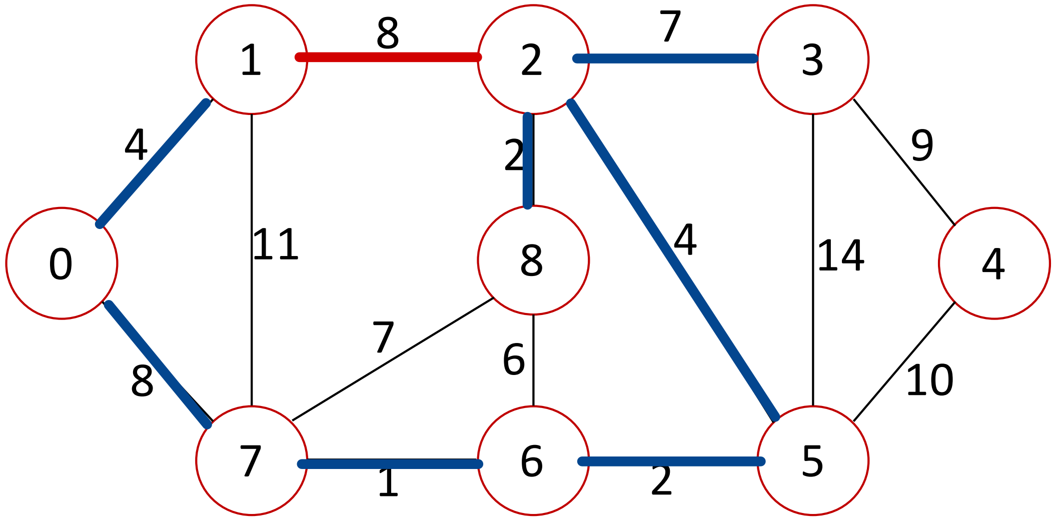
0	1	2	3	4	5	6	7	8
1	3	5	4	-1	3	5	6	2

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. (root src)
   2. Find set of second vertex. (root dest)
   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[srcRoot] = destRoot;





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

Parent[]

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

Union Find Algorithm – Analysis

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

- Time complexity
    - Skewed tree implementation
    - O(V)
- Improved time complexity
    - Rank based tree implementation
    - O(log V)

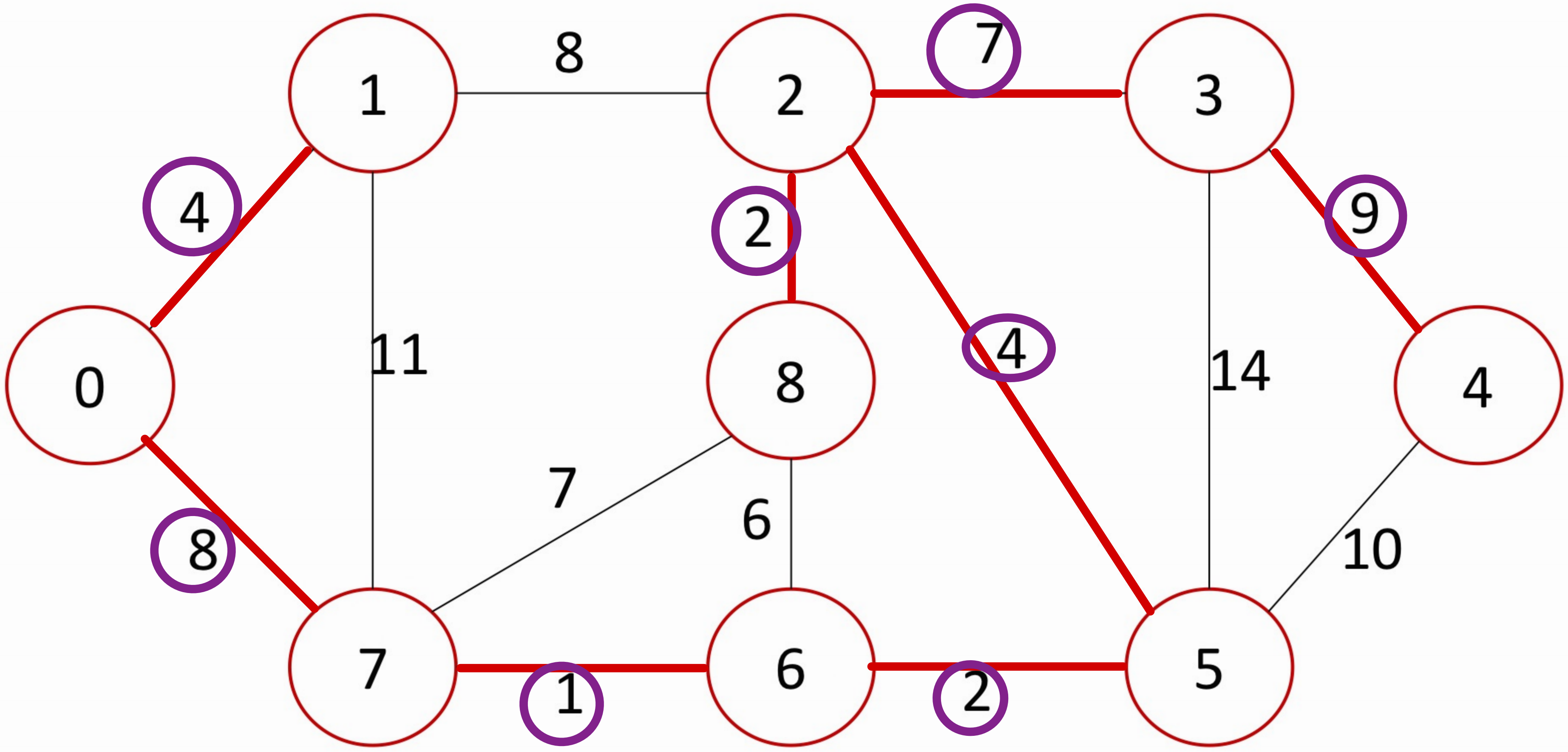
Kruskal's MST – Analysis

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.

- Time complexity
  - Sort edges: O(E log E)
  - Pick edges (E edges): O(E)
  - Union Find: O(log V)

- Time complexity
  - O(E log E + E log V)

  - E can max V^2.
  - So max time complexity: O(E log V).

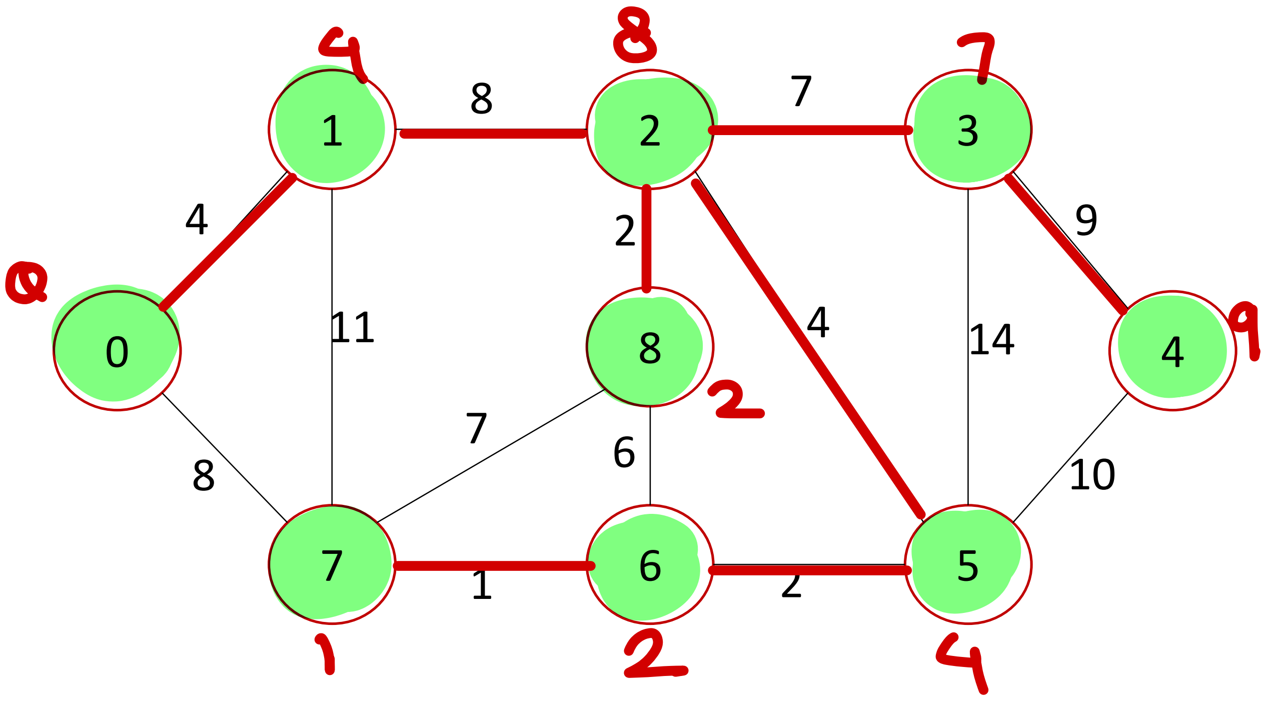


src	des	wt
7	6	1
8	2	2
6	5	2
0	1	4
2	5	4
8	6	6
2	3	7
7	8	7
0	7	8
1	2	8
3	4	9
5	4	10
1	7	11
3	5	14

Min weight Spanning Tree = 37

Prim's MST

1. Create a set *mst* to keep track of vertices included in MST.
2. Also keep track of parent of each vertex. Initialize parent of each vertex -1.
3. Assign a key to all vertices in the input graph. Key for all vertices should be initialized to INF. The start vertex key should be 0.
4. While *mst* doesn't include all vertices
   i. Pick a vertex u which is not there in *mst* and has minimum key.
   ii. Include vertex u to *mst*.
   iii. Update key and parent of all adjacent vertices of u.
      a. For each adjacent vertex v, if weight of edge u-v is less than the current key of v, then update the key as weight of u-v.
      b. Record u as parent of v.



mst total weight = 37

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>