Data Structure & Algorithms

Sunbeam Infotech

Agenda

- Union-find algorithm
- Kruskal's algorithm
- Dynamic Programming
- Bellaman Ford algorithm
- Warshall Floyd algorithm

Dijkstra's Algo: → $O(V \log V)$

1. Shortest Path tree.
2. If start vertex is changed, dist to all other vertices will change.
3. Min time/distance/resources to reach from given point to any other point.

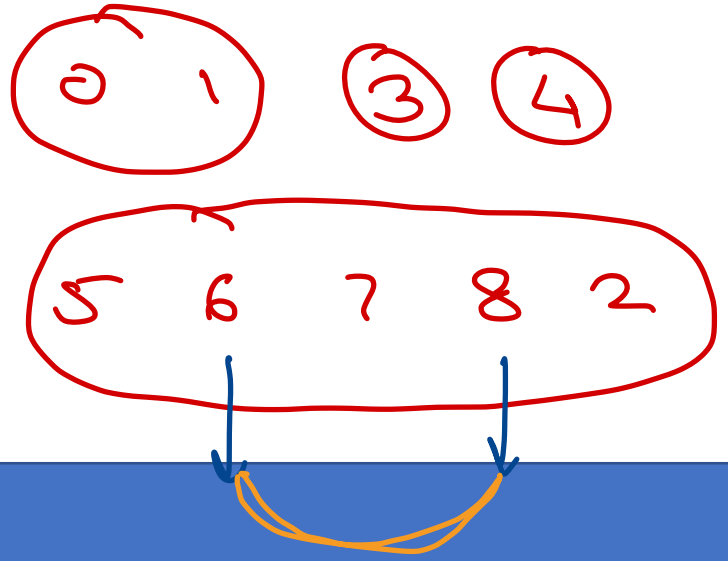
Prim's Algo: → $(V \log V)$

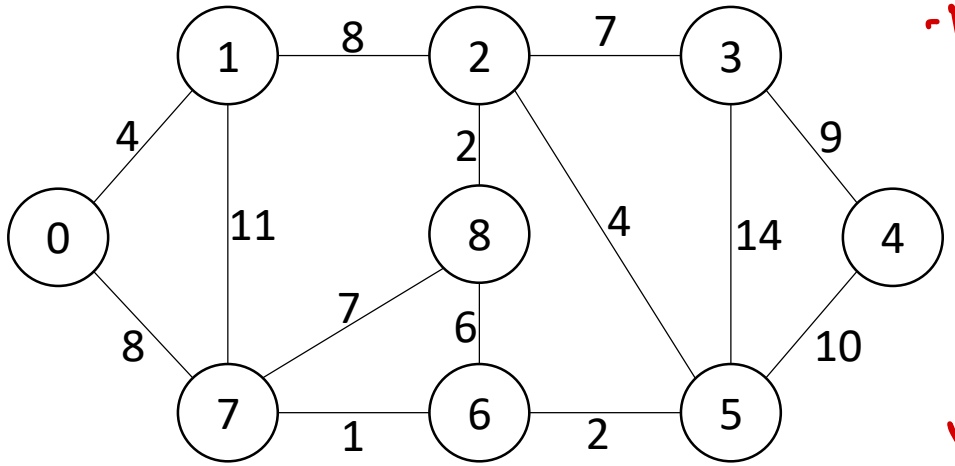
1. Min Spanning tree.
2. If root (start) is changed, MST may differ, but total weight remains same.
3. Min resources required to connect all the points.

Union Find Algorithm – to detect cycle in graph

union → parent[s] = d;

1. Consider all vertices as disjoint sets (parent = -1).
2. For each edge in the graph
   1. Find set of first vertex. (root)
   2. Find set of second vertex. (root)
   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 (union)






```
int findRoot(int v) {
    while(parent[v] != -1)
        v = parent[v];
    return v;
}
```

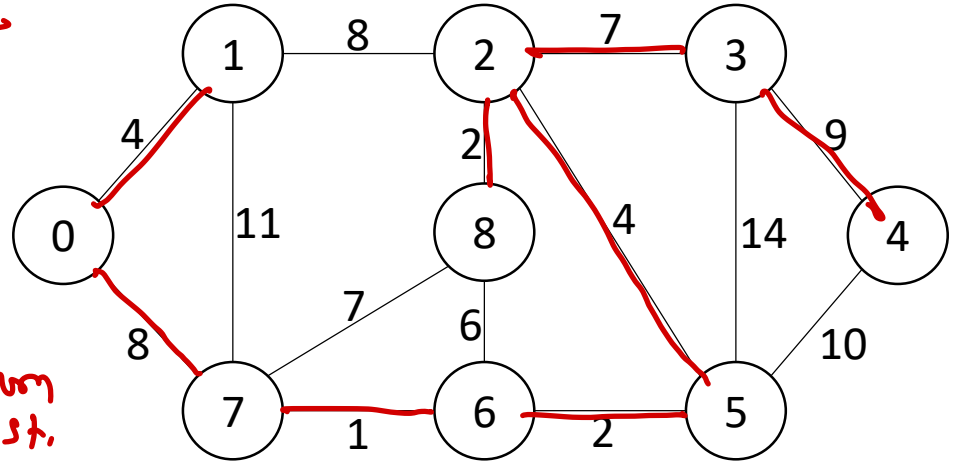
Parent

0	1	2	3	4	5	6	7	8
1	-1	5	-1	-1	-1	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 – min spanning tree also (like Prim's).

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. → from mst.
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	✓

Time Complexity = O(E V)

① get V-1 edges → O(E)

② find cycle → O(V)

Time Complexity = O(E log V)

using rank method. → V(log V)

Dynamic Programming

merge sort, quick sort, BST
x x x

- Dynamic programming is optimization over recursion.
- Typical DP problem give choices (to select from) and ask for optimal result (maximum or minimum).
- Technically it can be used for the problems having two properties
  - Optimal sub-structure
  - Overlapping sub-problems
- Optimal sub-structure usually represented by recursion.
- Overlapping sub-problem refers to recursive calls for the same values repeatedly.

- DP problems involve more than one recursive calls that too with overlapping.
- Alternative solution to DP is memoizing the recursive calls. This solution needs more stack space, but similar time complexity.

- Greedy algorithms pick optimal solution to local problem and never reconsider the choice done.
- DP algorithms solve the sub-problem in a iteration and improves upon it in subsequent iterations.

Dynamic Programming - fibonacci

$T_n = T_{n-1} + T_{n-2}$

f(5) = f(4) + f(3)

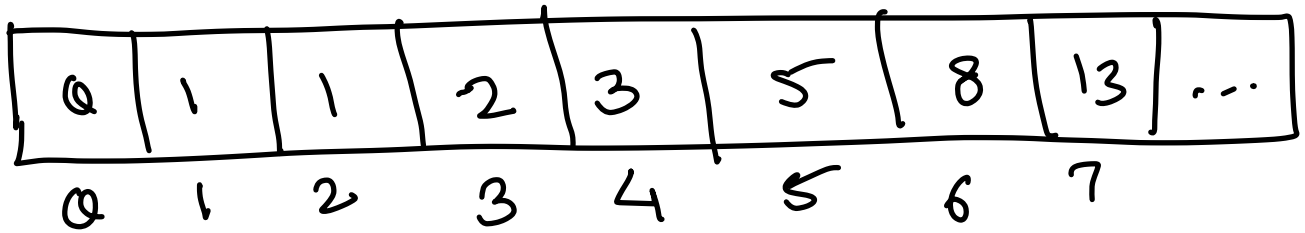
f(4) = f(3) + f(2)

f(3) = f(2) + f(1)

f(2) = 1 -

f(1) = 1 -

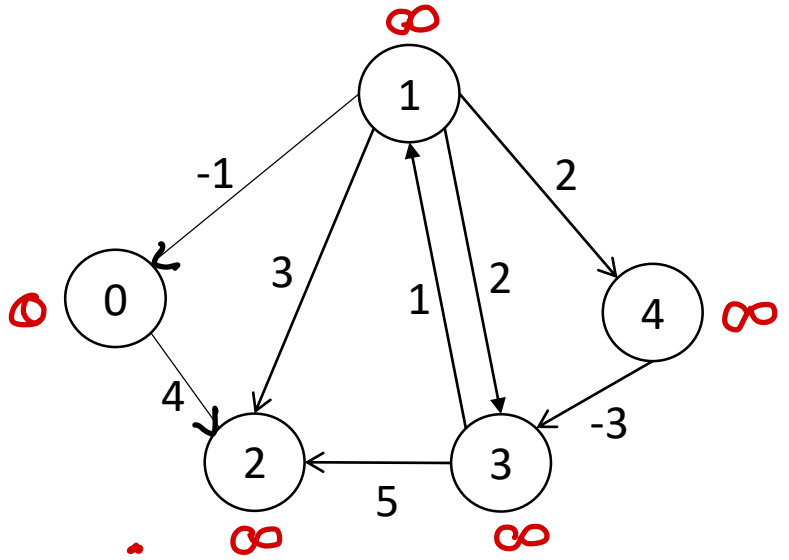
recursion memoization



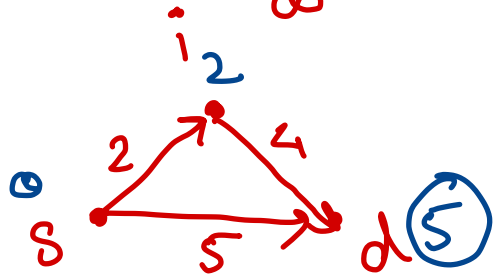

```
int fibo_rec(int n){
    if(n==1 || n==2)
        return 1;
    else
        return
            fibo_rec(n-1)
          + fibo_rec(n-2);
}
```

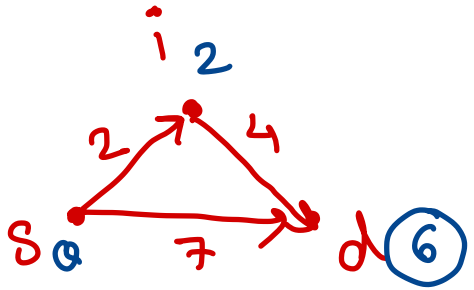
Bellman Ford Algorithm – shortest path algorithm. O(V*E)

- Initializes distances from the source to all vertices as infinite and distance to the source itself as 0.
- Calculates shortest distance V-1 times: For each edge u-v, if dist[v] > dist[u] + weight of edge u-v, then update dist[v], so that dist[v] = dist[u] + weight of edge u-v.
- Check if negative edge in the graph: For each edge u-v, if dist[v] > dist[u] + weight of edge uv, then graph has –ve weight cycle.



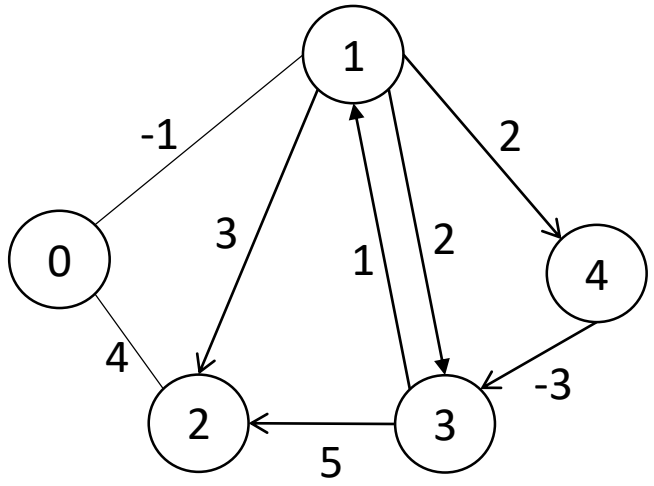
Src	Des	Wt
1	4	2
3	1	1
1	3	2
0	1	-1
0	2	4
3	2	5
1	2	3
4	3	-3





Dijkstra limitation: Can't handle negative edges
Dijkstra advantage: faster O(V log V)

Bellman Ford Algorithm



Src	Des	Wt
1	4	2
3	1	1
1	3	2
0	1	-1
0	2	4
3	2	5
1	2	3
4	3	-3

Warshall Floyd Algorithm

- Create distance matrix to keep distance of every vertex from each vertex. Initially assign it with weights of all edges among vertices (i.e. adjacency matrix).
- Consider each vertex (i) in between pair of any two vertices (s, d) and find the optimal distance between s & d considering intermediate vertex i.e. dist(s,d) = dist(s,i) + dist(i,d), if dist(s,i) + dist(i,d) < dist(s,d);

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>