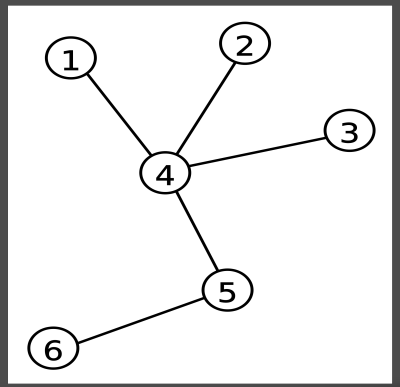
Advanced Computer Contest Preparation

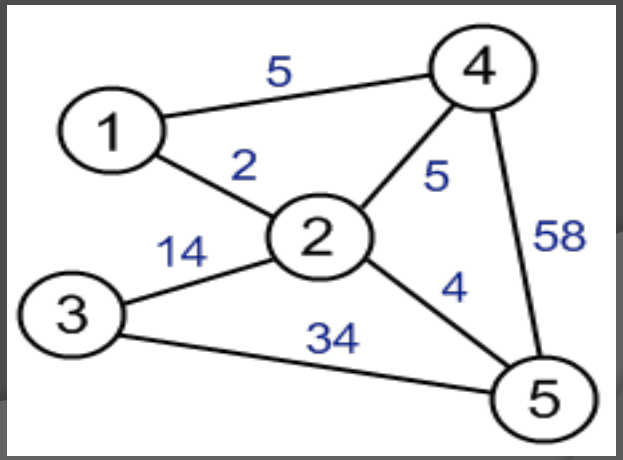
Lecture 7

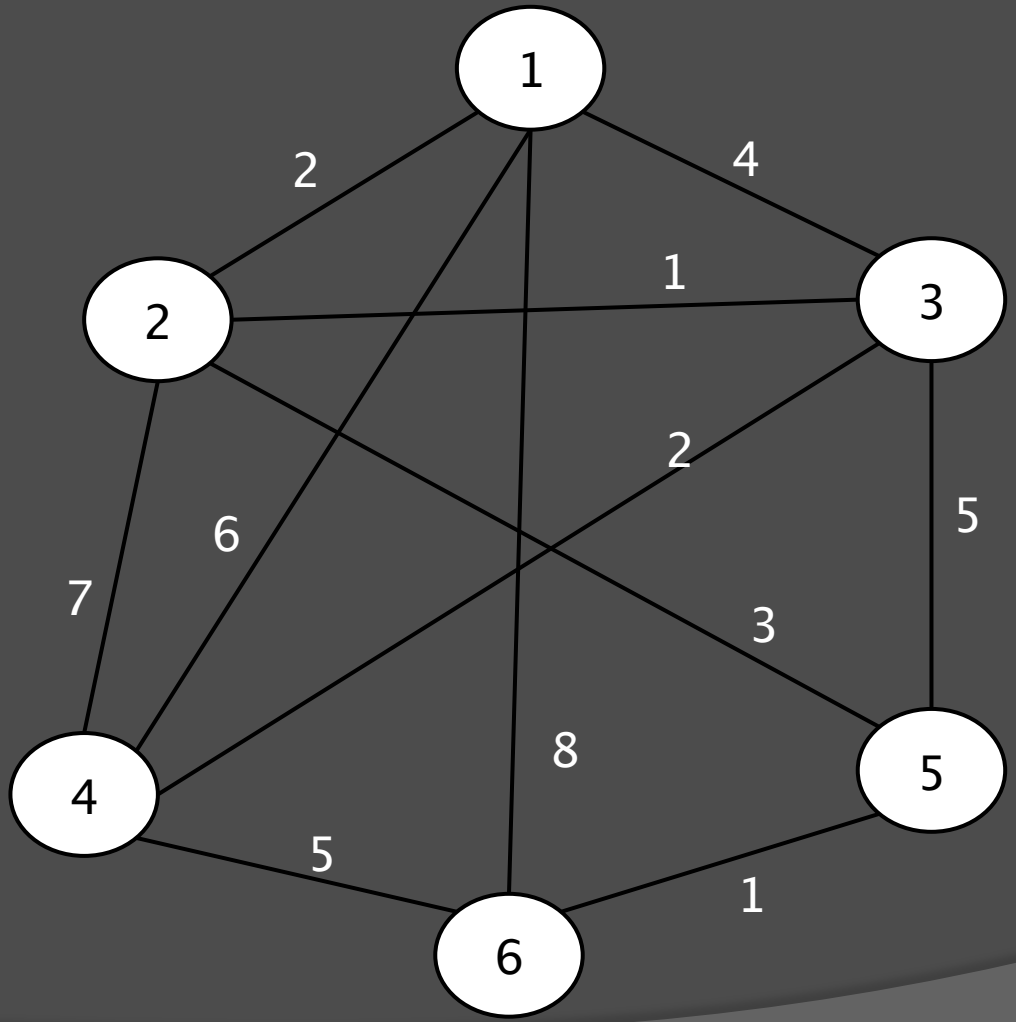
DIJKSTRA'S ALGORITHM

Distance Between Nodes?

- We can only BFS to find distance only if the graph is **unweighted**
- How can the shortest path from one node to all the others be computed on a weighted graph?



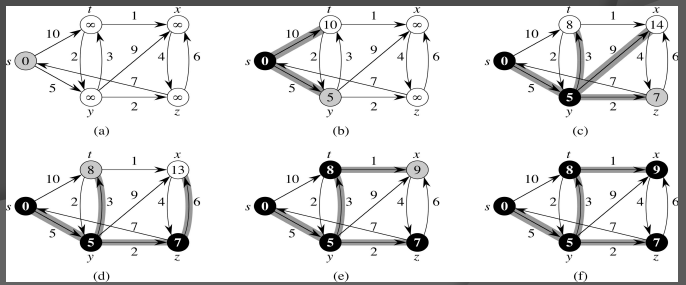




How do we find the shortest distance to each node from node 1?

Dijkstra's Algorithm

- Named after Esdger W. Dijkstra
- Computes distance to every node in a graph from **a single, source node**
  - Solves the **single source shortest path** problem
- Can be used on a graph with **non-negative** edge weights



Dijkstra's Algorithm

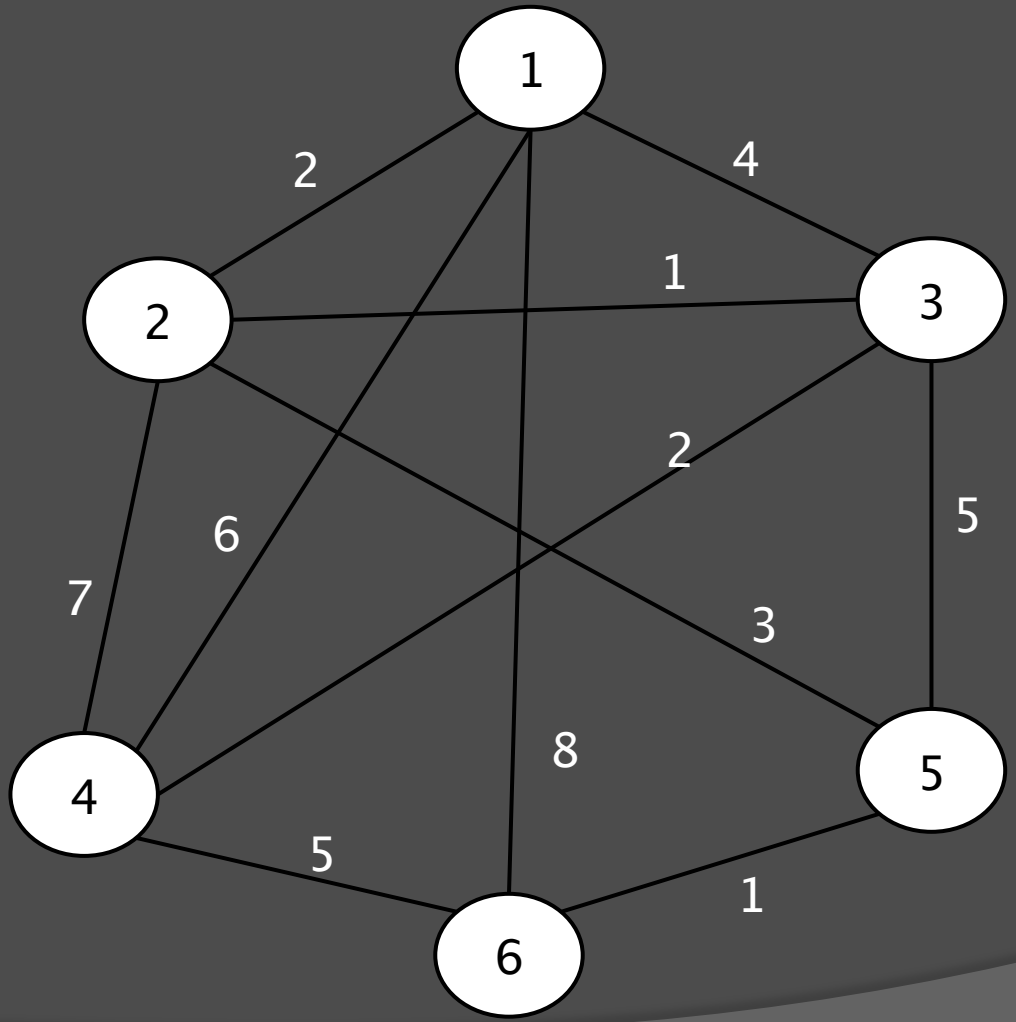
- Like BFS, there is a boolean array to mark if a node has been visited or not
- There is also a distance array, used to compute the shortest distance

Vis

1	2	3	4	5	6
0	0	0	0	0	0

Dist

1	2	3	4	5	6
∞	∞	∞	∞	∞	∞



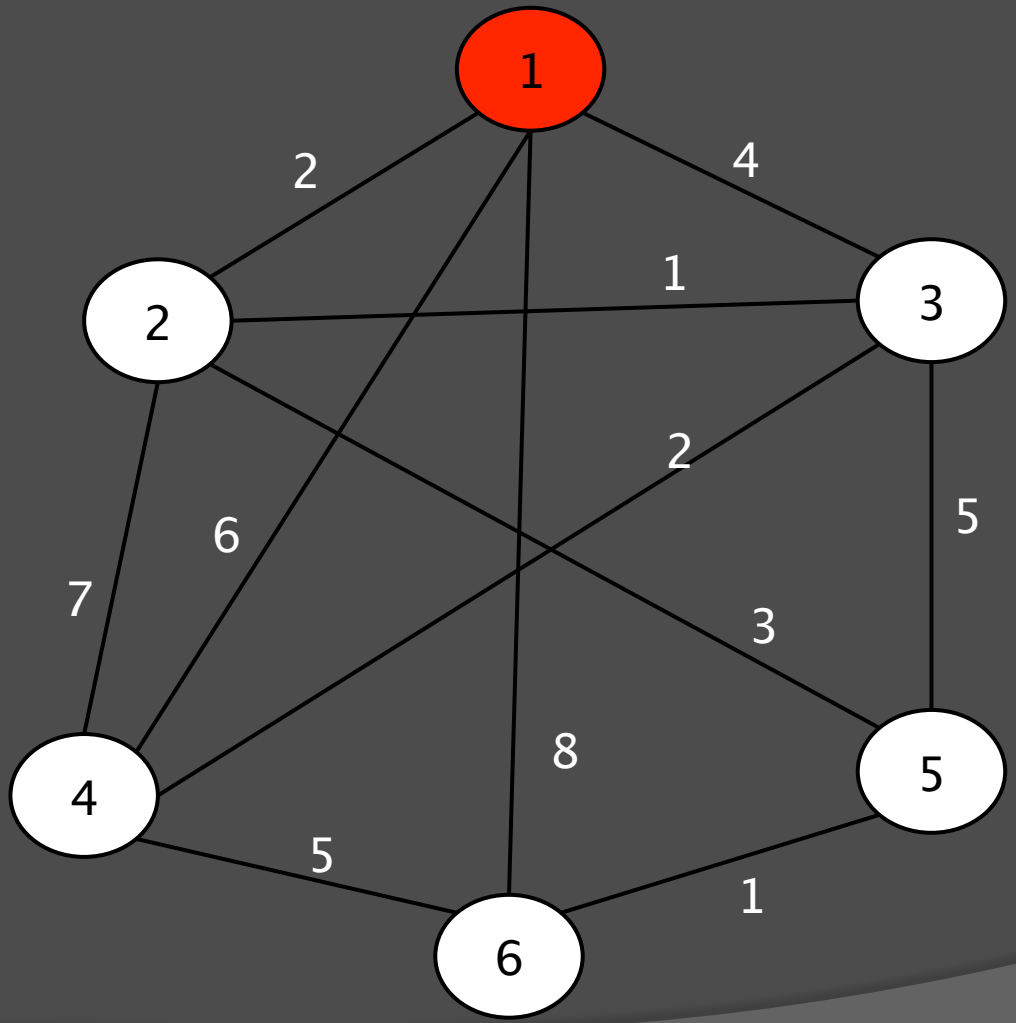
Initialize Everything:
Set vis array to false
Set dist array to infinity

Vis

1	2	3	4	5	6
0	0	0	0	0	0

Dist

1	2	3	4	5	6
∞	∞	∞	∞	∞	∞



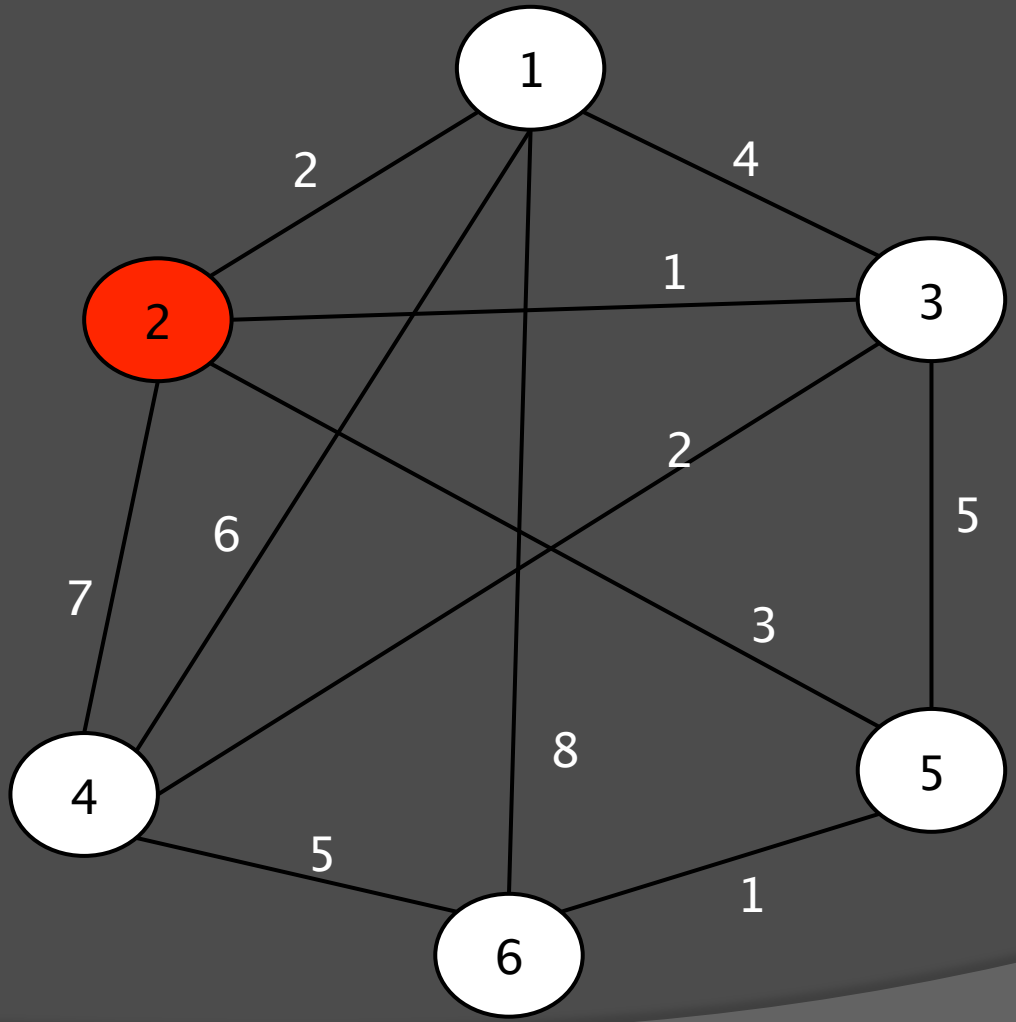
Start from node 1, set distance to 0, mark as visited, update distance to neighbors

Vis

1	2	3	4	5	6
1	0	0	0	0	0

Dist

1	2	3	4	5	6
0	2	4	6	∞	8



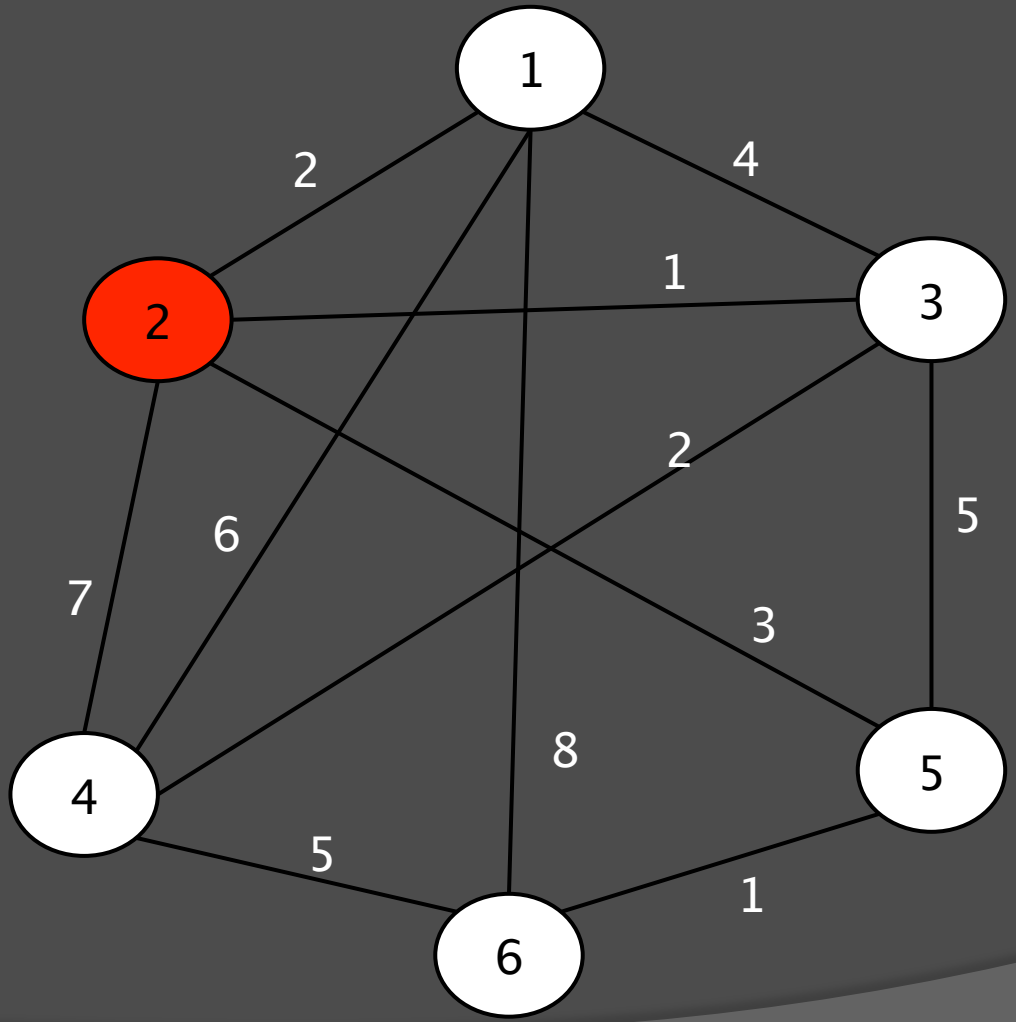
Choose unvisited node with lowest distance (node 2)

Vis

1	2	3	4	5	6
1	0	0	0	0	0

Dist

1	2	3	4	5	6
0	2	4	6	∞	8



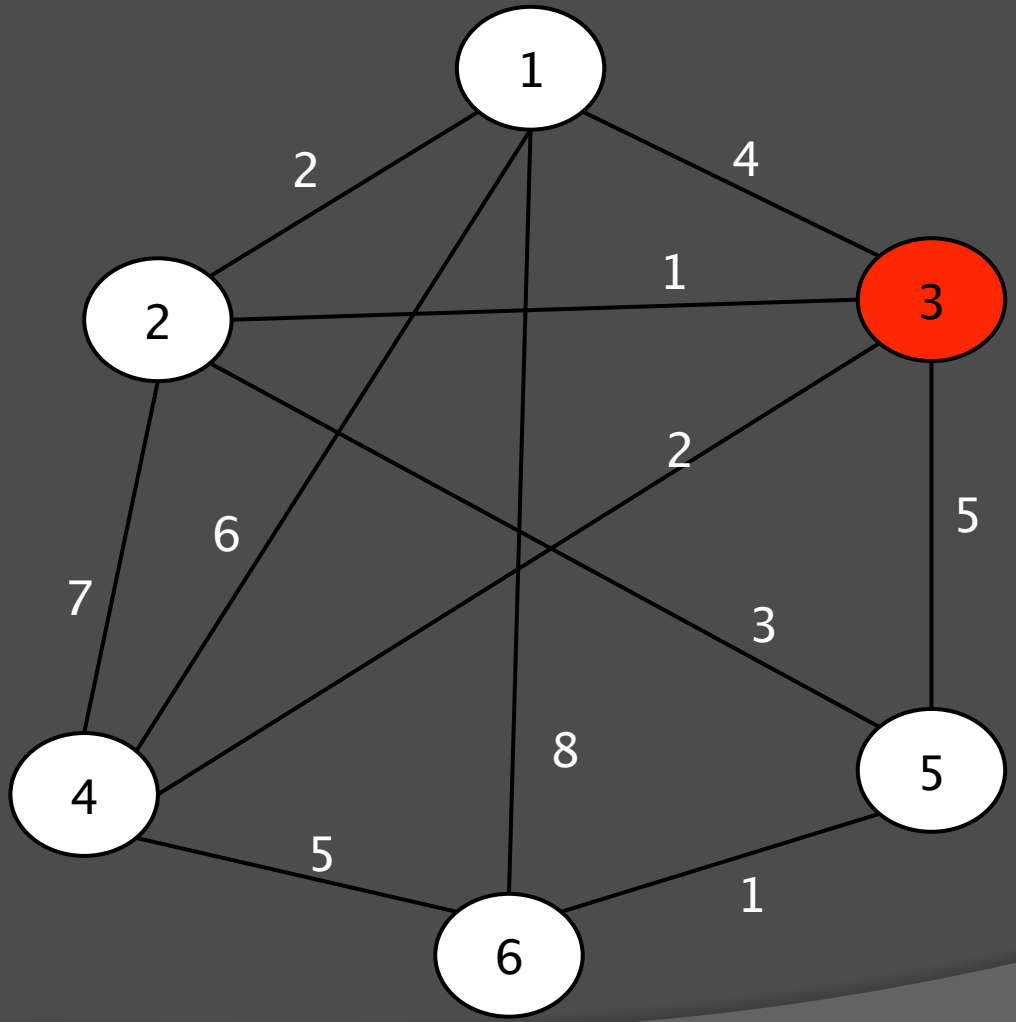
Mark node 2 as visited, update distance to neighbors (distance to node 2 plus edge cost). Change distance if it is **less.**

Vis

1	2	3	4	5	6
1	1	0	0	0	0

Dist

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



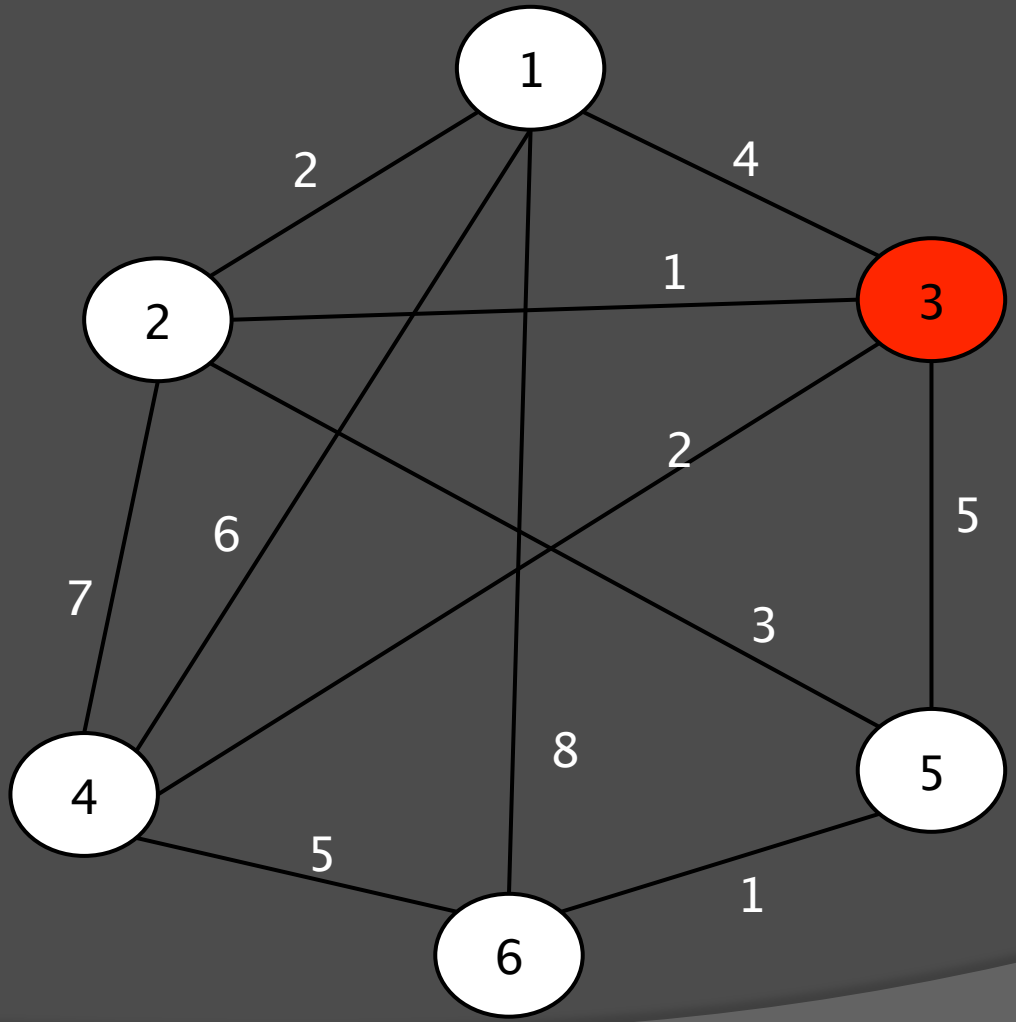
Choose unvisited node with lowest distance (node 3)

Vis

1	2	3	4	5	6
1	1	0	0	0	0

Dist

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



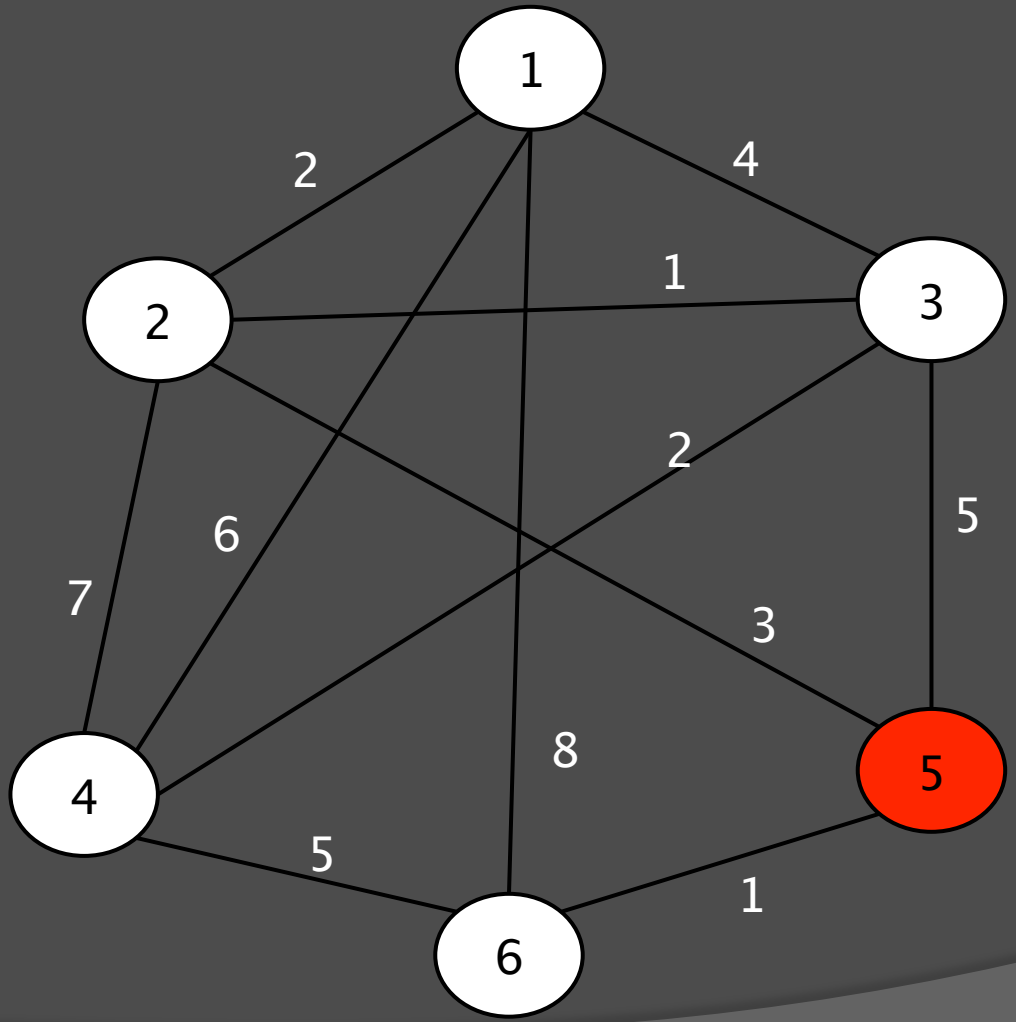
Mark node 3 as visited, update distance to neighbors

Vis

1	2	3	4	5	6
1	1	1	0	0	0

Dist

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



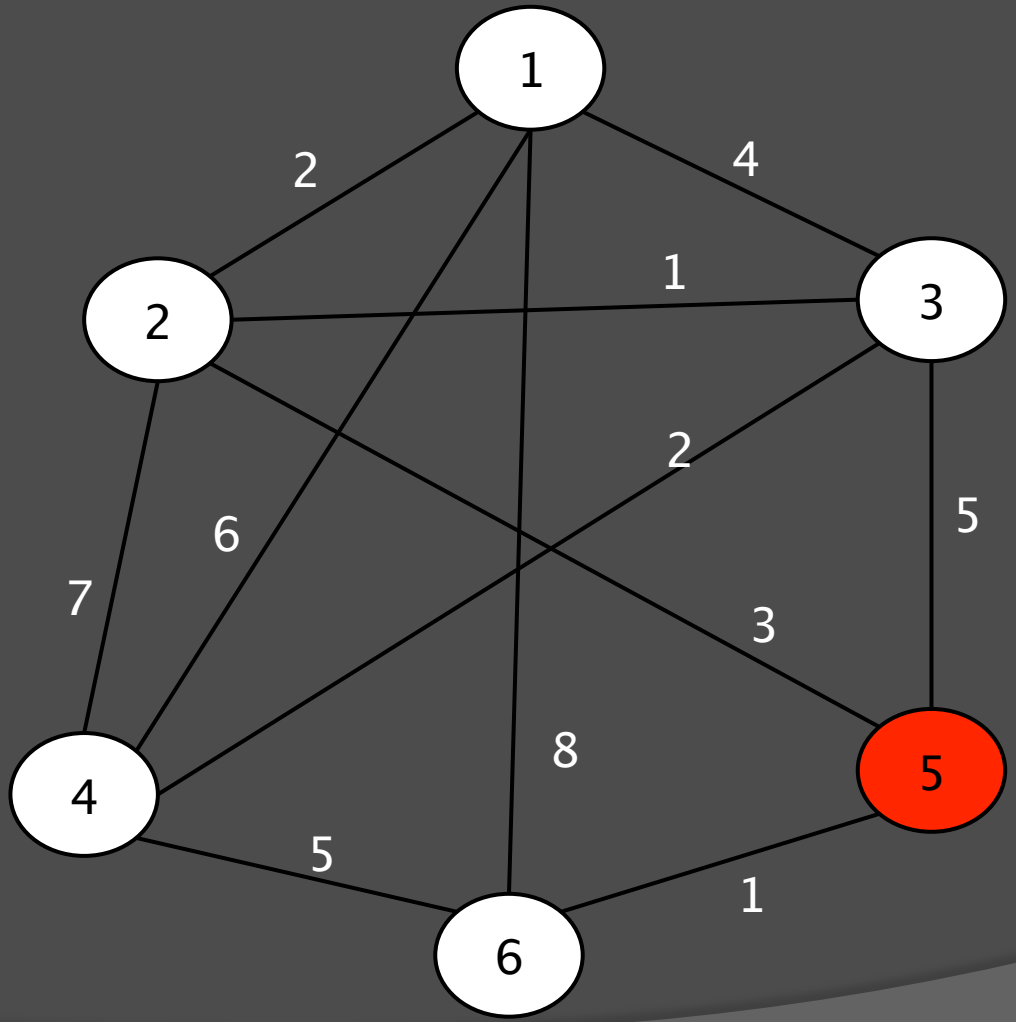
Choose unvisited node with lowest distance (node 5)

Vis

1	2	3	4	5	6
1	1	1	0	0	0

Dist

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



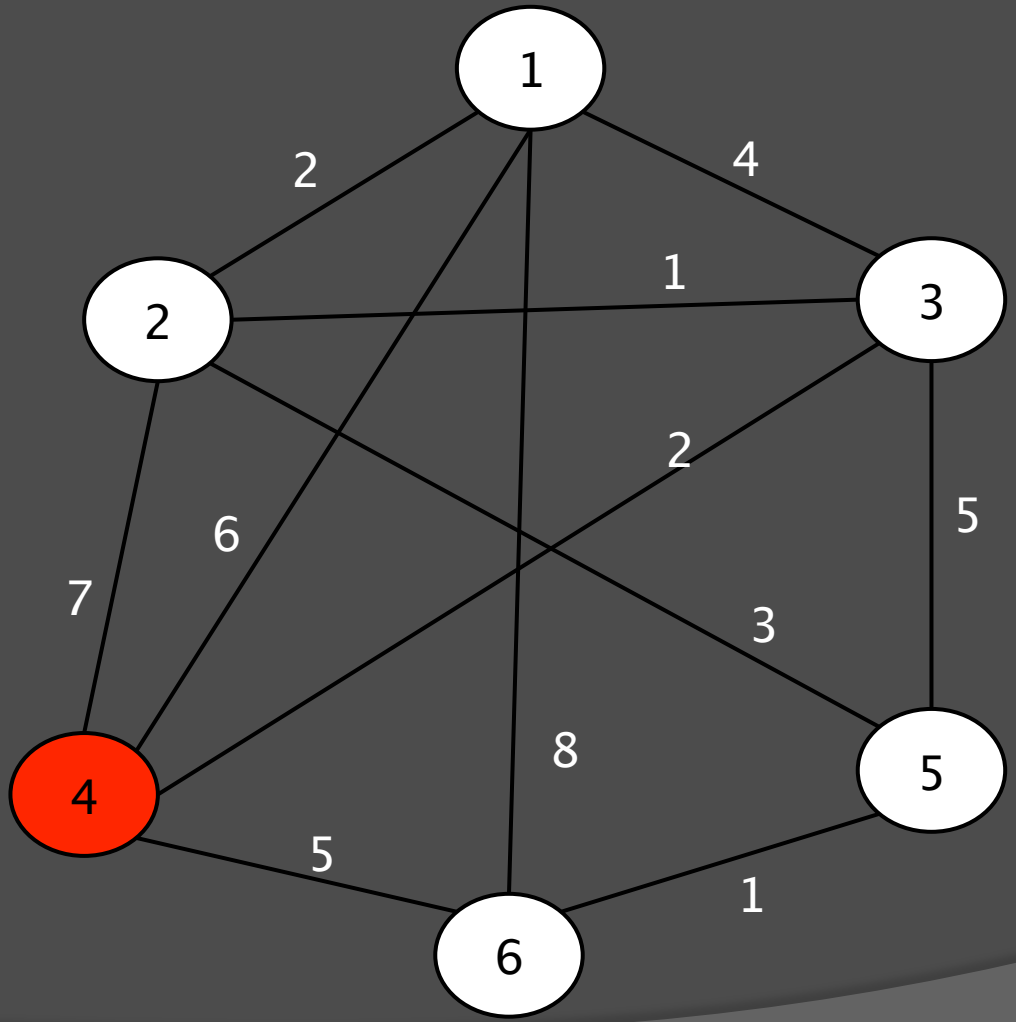
Mark node 5 as visited, update distance to neighbors

Vis

1	2	3	4	5	6
1	1	1	0	1	0

Dist

1	2	3	4	5	6
0	2	3	5	5	6



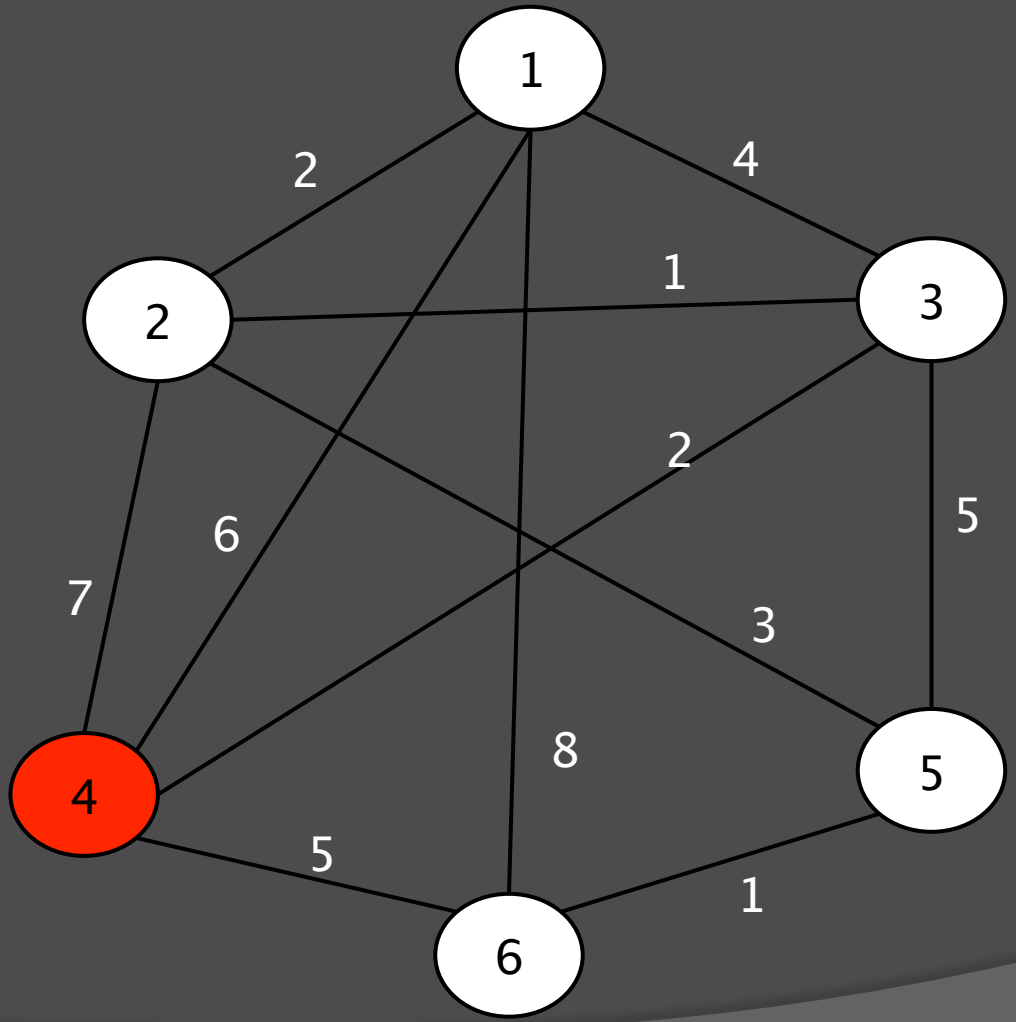
Choose unvisited node with lowest distance (node 4)

Vis

1	2	3	4	5	6
1	1	1	0	1	0

Dist

1	2	3	4	5	6
0	2	3	5	5	6



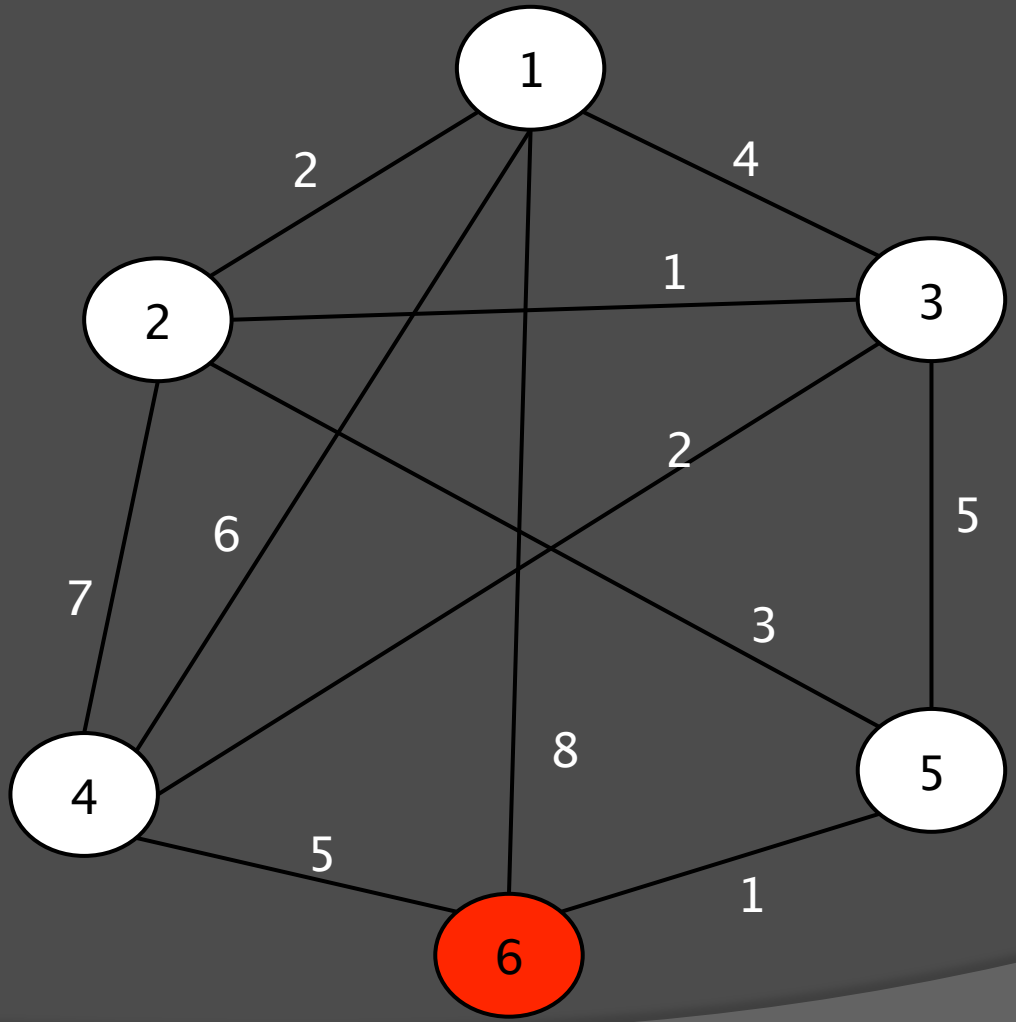
Mark node 4 as visited, update distance to neighbors

Vis

1	2	3	4	5	6
1	1	1	1	1	0

Dist

1	2	3	4	5	6
0	2	3	5	5	6



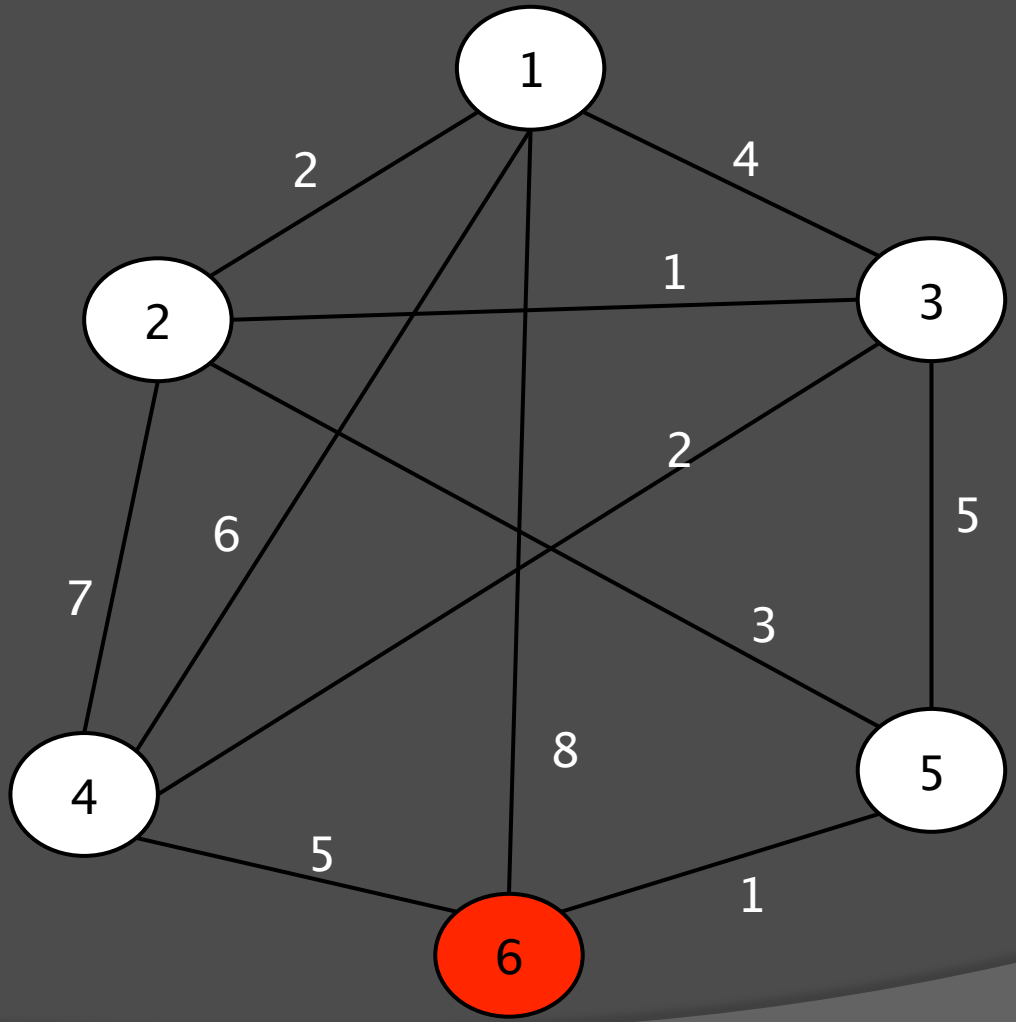
Choose unvisited node with lowest distance (node 6)

Vis

1	2	3	4	5	6
1	1	1	1	1	0

Dist

1	2	3	4	5	6
0	2	3	5	5	6



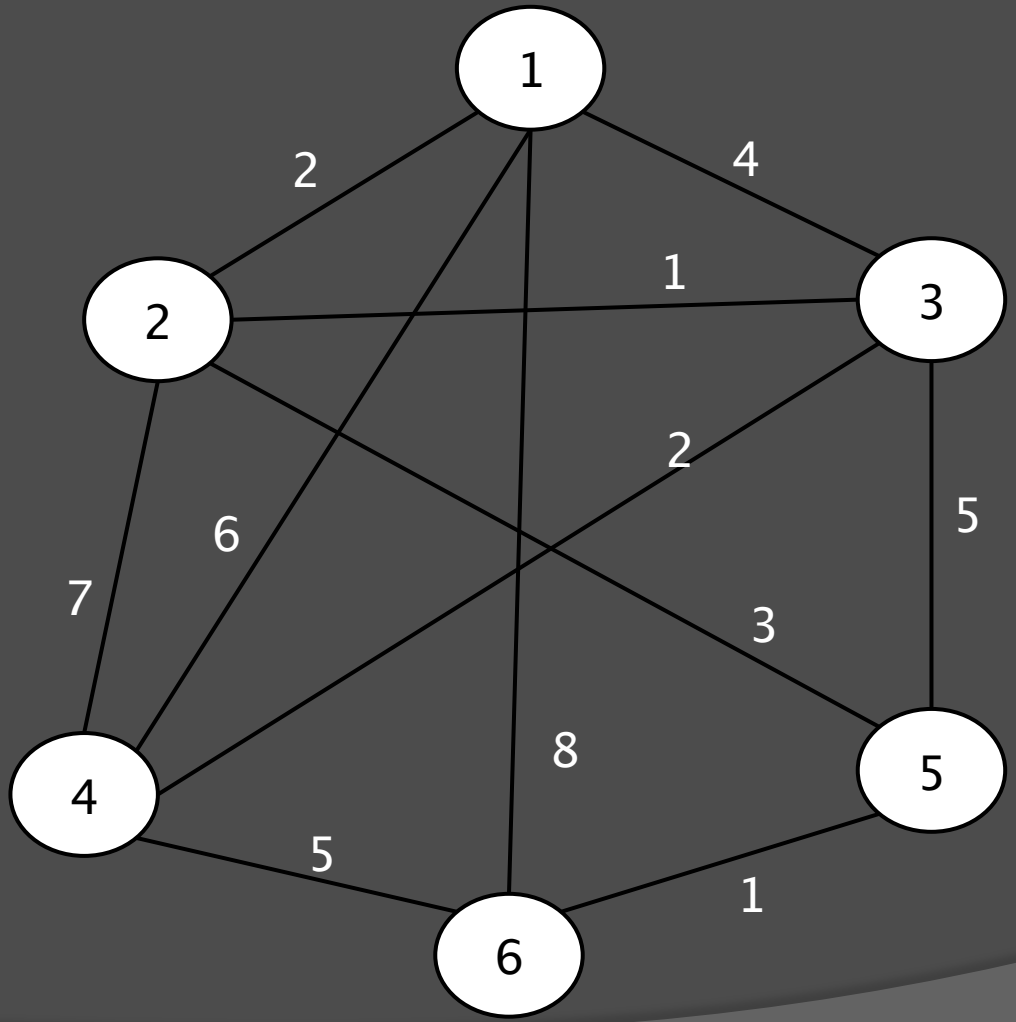
Mark node 6 as visited, update distance to neighbors

Vis

1	2	3	4	5	6
1	1	1	1	1	1

Dist

1	2	3	4	5	6
0	2	3	5	5	6



Done!
The dist array represents least distance to corresponding node from node 1
(e.g. minimum distance from 1 to 4 is 5)

Vis

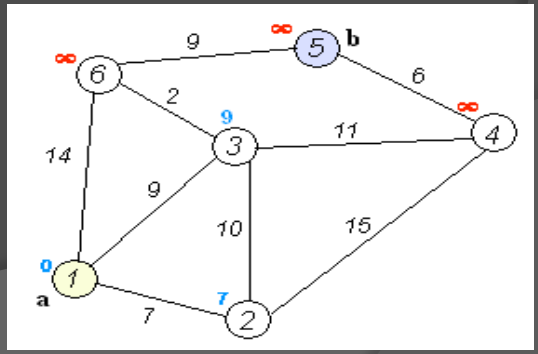
1	2	3	4	5	6
1	1	1	1	1	1

Dist

1	2	3	4	5	6
0	2	3	5	5	6

Dijkstra's Algorithm - Summary

- Start at a node (initialize)
- Until all nodes are processed:
  - Choose an unvisited node with the lowest distance
  - Mark the node as visited
  - Update distance to neighbors



Why Does it Work?

- The unvisited node with the lowest distance computed is chosen next
- All distances to other nodes are equal or greater
- Therefore, choosing any other path to the same node **must** be longer, since the graph contains non-negative edge weights

Dijkstra's Algorithm - Pseudocode

```
bool vis[]; int dist[];
void dijkstra(node root){
        set all dist to infinity;
        dist[root] = 0;
        while (there are nodes left){
                node cur = unvisited node with lowest distance;
                mark cur as visited;
                for (every node u adjacent to cur){
                        if (u is not visited){
                                dist[u] = min(dist[u],dist[cur] + cost[cur][u]);
                        }
                }
        }
}
```

Runtime

- What is the runtime for Dijkstra's algorithm?
- Needs to be run on every node: $O(V)$
- On every node, every node must be searched to find lowest distance: $O(V)$

Runtime

- On an **adjacency matrix**, all nodes must be checked for adjacency: $O(V)$
- On an **adjacency list**, all edges must be checked for adjacency: $O(E)$

```
[[a, [b],
 [b, [a, c, g],
 [c, [b, d],
 [d, [c, e],
 [e, [d, f],
 [f, [e, g],
 [g, [b, f]]
```

Adjacency List

```
   a  b  c  d  e  f  g
a [ , x,  ,  ,  ,  ,  ]
b [x,  , x,  ,  ,  , x]
c [ , x,  , x,  ,  ,  ]
d [ ,  , x,  , x,  ,  ]
e [ ,  ,  , x,  , x,  ]
f [ ,  ,  ,  , x,  , x]
g [ , x,  ,  ,  , x,  ]
```

Adjacency Matrix

Runtime

- Therefore, the total runtime of Dijkstra's algorithm is:
  - $O(2V^2)$ (adjacency matrix)
  - $O(V^2 + E)$ (adjacency list)
- Look at the limits of V and E to decide whether to use adjacency matrix or list

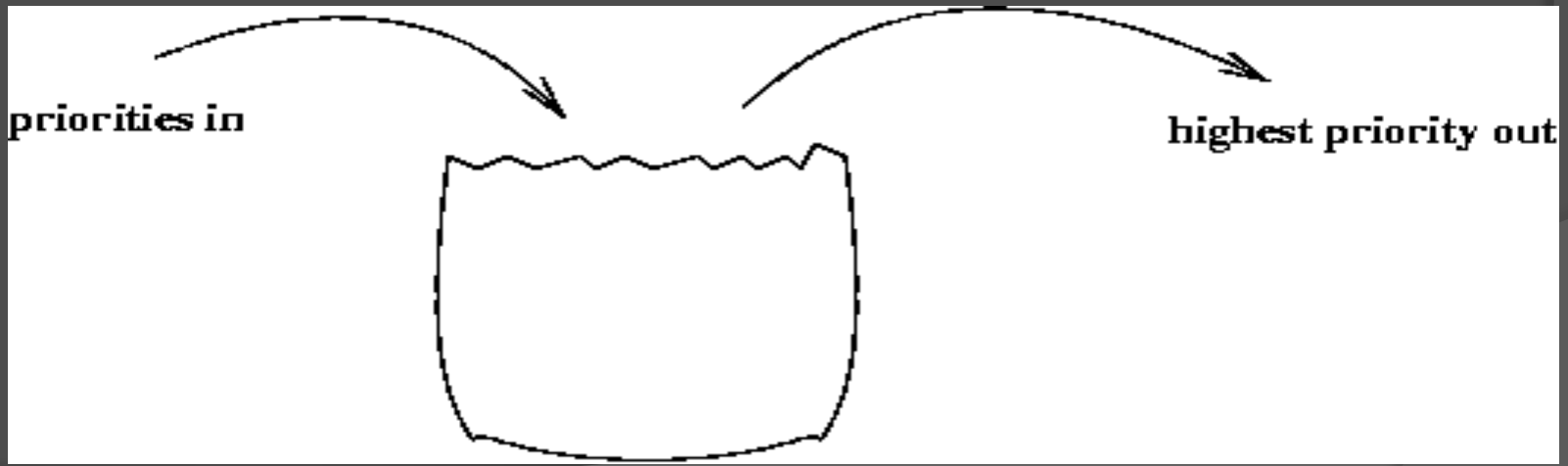
Optimization

- Which step in Dijkstra's algorithm could be optimized?

- Needs to be run on every node
- Every time the algorithm is run, every node must be searched to find lowest distance
- All nodes or edges must be checked for adjacency

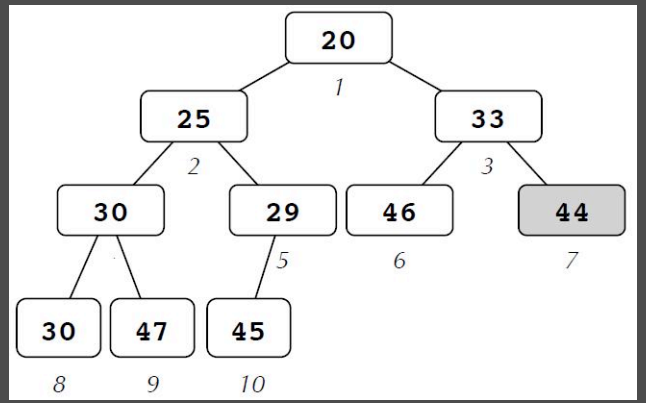
Optimization

- How can we optimize finding the next node to visit?
- Instead of searching through every node...
- Use a **priority queue**



Recall: Priority Queue

- A container where the first element is always the greatest or least of all of its elements
- Functions similar to priority lines at airports
- Compare function can be custom



Dijkstra - Optimization

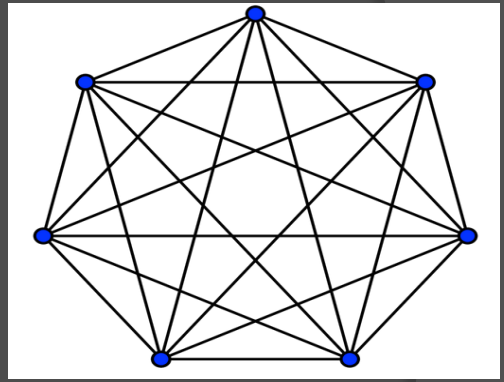
- Instead of checking all unvisited nodes to find the node with lowest distance...
- **If distance to a node is updated**, push the distance and node (order matters!) into the priority queue
- What is the runtime of using priority queue?
  - $O(E \log E)$

PQ Dijkstra - Summary

- Start at a node and push it into the priority queue (min distance priority)
- While the priority queue is not empty
  - Keep popping until node on top has not been visited
    - Can check if distance in priority queue is equal to best distance to node instead, visited array is not necessary if this is done
  - Mark that node as visited
    - Unnecessary if distance is used instead
  - Update distance to neighbors
    - If distance is updated, push that neighbor into the priority queue

PQ Dijkstra

- Final runtime:
- Adjacency Matrix: $O(V^2 + E \log E)$
- Adjacency List: $O(V + E + E \log E)$
- Compare unoptimized Dijkstra:
  - $O(2V^2)$ (adjacency matrix)
  - $O(V^2 + E)$ (adjacency list)
- Unoptimized Dijkstra is better for graphs with a lot of edges and fewer nodes (dense)
  - $O(E)$ can be up to $O(V^2)$

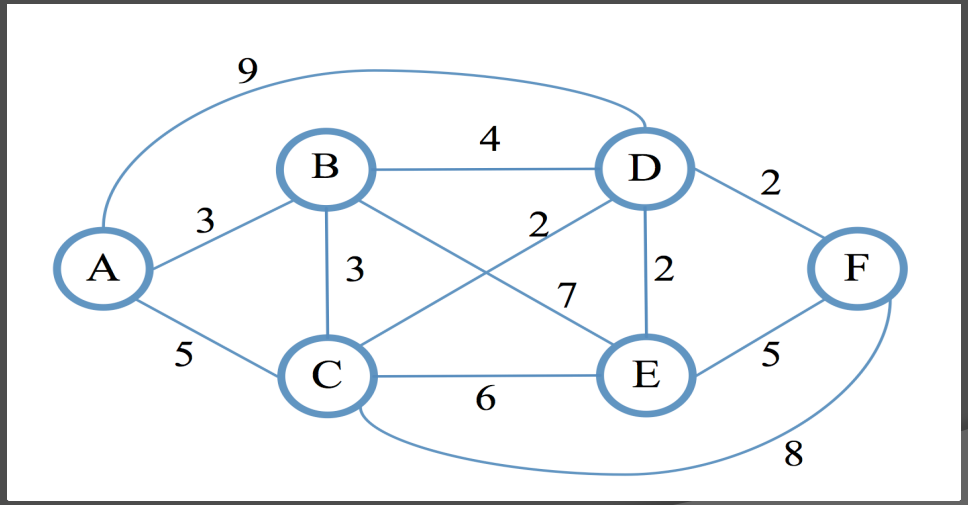


PQ Dijkstra - Pseudocode

```
int dist[],vis[];
priority_queue<pair<int,node> > q;
void dijkstra(node root){
          set all dist to infinity;
          dist[root] = 0;
          q.push(make_pair(0,root));
          while (q is not empty){
                    pair cur_pair = q.top(); q.pop();
                    int cur_dist = cur_pair.first; node cur = cur_pair.second;
                    if (vis[cur]) continue;
                    mark cur as visited;
                    for (every node u adjacent to cur){
                              if (u is not visited){
                                        int newDist = dist[cur] + cost[cur][u];
                                        if (newDist < dist[u]){
                                                  dist[u] = newDist;
                                                  q.push(make_pair(newDist,u));
                                        }
                              }
                    }
          }
}
```

Try it Out!

- Given a **connected, undirected, edge-weighted** graph, output the distance to every node from node 1.



THANK YOU!