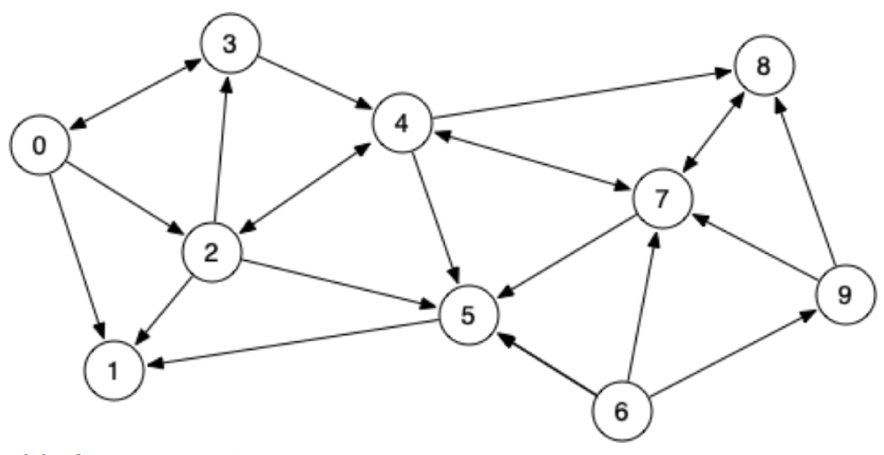
Week 08 Tutorial
Graphs

Week 08 Tutorial
Graphs

1. In the following graph:



a. Which vertices are reachable from vertex 0?
b. Which vertices are reachable from vertex 1?
c. Which vertices are reachable from vertex 5?
d. Which vertices are reachable from vertex 6?

2. Write a C function that takes a Graph and a starting Vertex and returns a set containing all of the vertices that can be reached by following a path from the starting point. Use the function template:

```
Set reachable(Graph g, Vertex src) { ... }
```

You may use any of the standard ADTs from the slides (e.g. Sets, Lists, Stacks, Queues).

What algorithm can we use to do this?

Dijkstra's Algorithm
- Used to find the shortest path in a weighted graph from a starting vertex to another
- From the current nodes we have visited and edges available, take the most optimal path and revise as more nodes are visited and edges are added

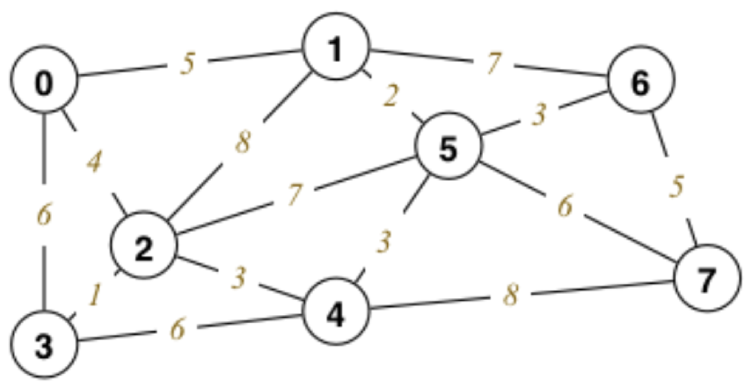
```
dijkstraSSSP(G, src):
    Inputs: graph G, source vertex src

    create dist array, initialised to ∞
    create pred array, initialised to -1
    create vSet containing all vertices of G

    dist[src] = 0
    while vSet is not empty:
        find vertex v in vSet such that dist[v] is minimal
        remove v from vSet
        for each edge (v, w, weight) in G:
            relax along (v, w weight)
```

1. Calculate the distance from source vertex to current vertex's neighbours via the current vertex i.e. d = dist[v] + |v + w|
2. If d < dist[w] = d, pred[w] = v
3. Select vertex from vSet with smallest distance i.e. smallest dist[v] and make it the current vertex
4. Repeat the steps above

3. Trace the execution of Dijkstra's algorithm on the following graph to compute the minimum distances from source node 0 to all other vertices:



Show the values of `vSet`, `dist[]` and `pred[]` after each iteration.

Minimum Spanning Tree
Tree: a connected graph with no cycles
Spanning Tree: a subgraph which is a tree and contains every vertex in a graph
Minimum spanning tree: a spanning tree with the minimum total weight of edges

Note: the number oof edges in a spanning tree is (number of vertices - 1)

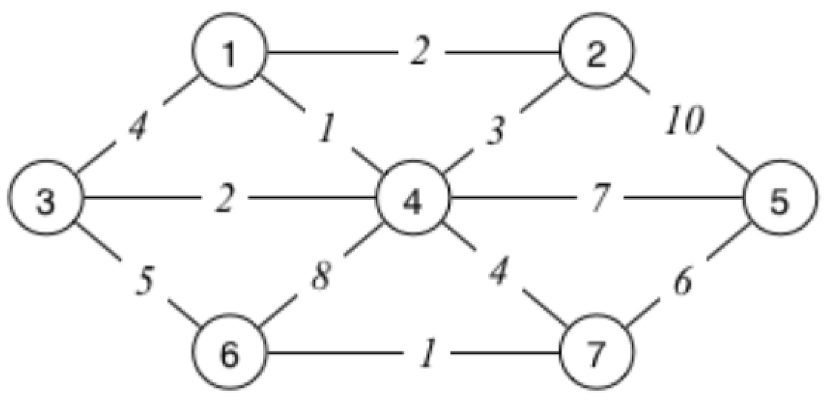
Kruskal's Algorithm

```
typedef Graph MSTree;

MSTree kruskalFindMST(Graph g) {
    MSTree mst = GraphNew(g->nV); // MST initially empty
    Edge eList[g->nE]; // sorted array of edges
    edges(eList, g->nE, g);
    sortEdgeList(eList, g->nE);
    for (int i = 0; mst->nE < g->nV - 1; i++) {
        Edge e = eList[i];
        GraphAddEdge(mst, e);
        if (GraphHasCycle(mst)) GraphRemoveEdge(mst, e);
    }
    return mst;
}
```

1. Have a list or priority queue of all edges in the graph from smallest to highest weight
2. Get the smallest weight edge from the list and add to MST
3. If a cycle forms, remove it from the MST
4. Repeat this until there are (number of vertices - 1) edges

Note: Kruskal's runs faster in sparse graphs

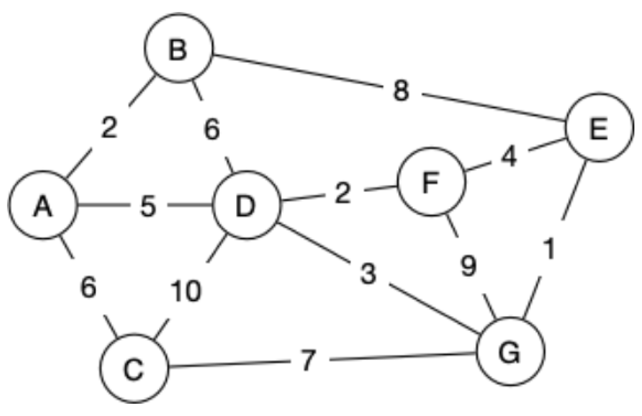


Prim's algorithm

1. Start from any vertex *v* and empty MST
2. Choose edge not already in MST, satisfying
   - incident on a vertex *s* already in MST
   - incident on a vertex *t* not already in MST
   - with minimal weight of all such edges
3. Add chosen edge to MST
4. Repeat until MST covers all vertices

Note: Prim's runs faster in dense graphs

Show how Prim's algorithm produces an MST on the following graph:



Start at vertex A.