# Week 9 Notes

*Prof Bill - May 2018*

Week 9… graph algorithms.

- ❏ Graph traversal: DFS, BFS
- ❏ Topological Sort
- ❏ Shortest Path: Dijkstra’s Algorithm
- ❏ Minimum Spanning Tree: Prim, Kruskal

thanks… yow, bill

## A. Graph Traversal

** Online: Read pages 27-53, algs4.cs.princeton.edu/lectures/41UndirectedGraphs.pdf
** Animation:

- DFS, www.cs.usfca.edu/~galles/visualization/DFS.html
- BFS, www.cs.usfca.edu/~galles/visualization/BFS.html

**Graph traversal** - explore graph by visiting all verts
Two flavors: Depth-first search (DFS) and Breadth-first search (BFS)

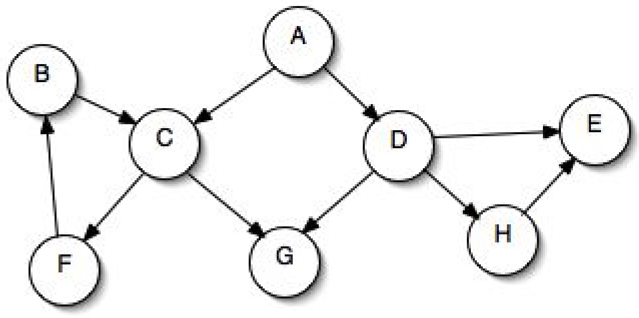
### Depth-first search (DFS)

Key concept - it's recursive!
Pseudocode:

```
// mark vertex v as visited, then recursively visit all connected verts
// prior to first dfs call, mark all verts as not visited
dfs( vertex v)
   mark v as visited
   for each vert w: adjacent to v {
      if w not visited
         dfs( w)
   }
```

Applications of DFS: detect cycle in graph, find path between verts, determine if connected graph, topological sort, determine if bipartite graph, walk thru maze
Source: www.geeksforgeeks.org/?p=11644

Traversal example: Start at vertex A, list verts as you visit them
/* when you have a choice of >1 verts, use sorted order */



DFS traversal example answer: A, C, F, B, G, D, E, H

## Breadth-first search (BFS)

Key concept - use a queue!

Pseudocode:

```
// use queue to do a breadth-first traversal of graph
bfs( vertex v)
   mark all verts not visited
   q = new queue
   q.enqueue( v)
   mark v as visited
   while ! q.isEmpty() {
      v2 = q.dequeue()
      for each vert w: adjacent to v2 {
         if w not visited
            q.enqueue( w)
            mark w as visited
      }
   }
```

Applications of BFS: min spanning tree, shortest path, peer-to-peer networks, social media, search engine crawlers,

Source: www.geeksforgeeks.org/applications-of-breadth-first-traversal/

Try traversal example again, using BFS…

BFS traversal example answer: A, C, D, F, G, E, H, B

## B. Topological Sort

** Online: Read pages 38-50, algs4.cs.princeton.edu/lectures/42DirectedGraphs.pdf

** Animation: www.cs.usfca.edu/~galles/visualization/TopoSortDFS.html

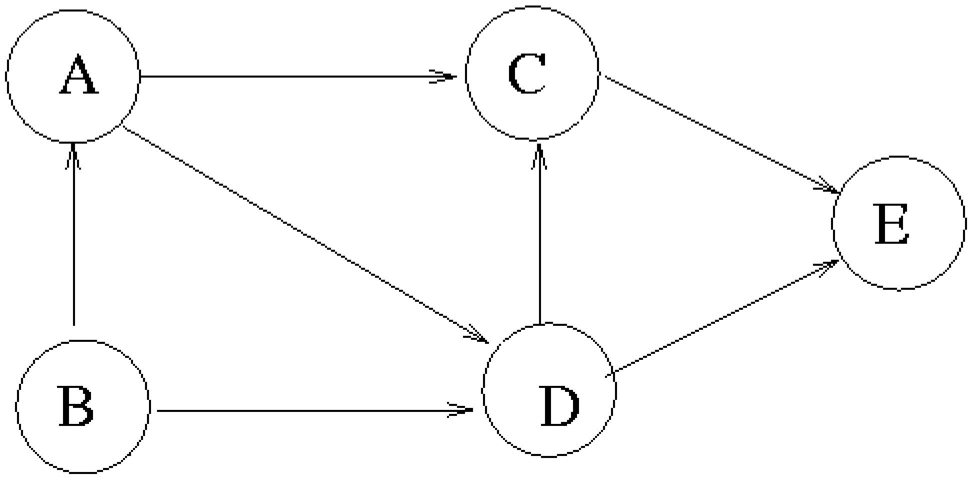
**Topological sort** - a nice recursive definition

An ordering of vertices in a directed acyclic graph, such that:
If there is a path from u to v, then v appears after u in the ordering.

Also...

- For directed acyclic graph (**DAG**) only, please.
- There can be more than one topological sort for any graph.
- A **cycle** in the graph breaks this definition and therefore has no topological sort.
  For example: path from u to v… and also v to u… then who goes first in sort?

Example: What is topological sort?



Source: lcm.csa.iisc.ernet.in/dsa/node170.html

Example answer: B A D C E

## DFS algorithm

Combine DFS with stack. Push vert onto stack after its DFS traversal is done.
Topological sort = reverse postorder of DFS search.

Pseudocode

```
// print verts in topological sort order
toposort( graph g) {
   s = new stack
   for each vert v in g
      dfs2( v, s)

   while not s.isEmpty()
      v = s.pop()
      print v
}

// dfs traversal, vert pushed on stack once all neighbors visited
dfs2( vertex v, stack s) {
   mark v as visited
   for each vert w: adjacent to v {
      if w not visited
         dfs( w)
   }
   s.push(v)
}
```

## C. Shortest Path: Dijkstra's Algorithm

** Online: Princeton section, algs4.cs.princeton.edu/lectures/44ShortestPaths.pdf

** Animation: www.cs.usfca.edu/~galles/visualization/Dijkstra.html

**weighted graph** - each edge has a positive weight (distance, force, etc)

**Dikstra's Algorithm** - single-source shortest path in a graph; greedy + relaxation

- greedy algorithm - choose the shortest (best) edge at each step.
- relaxation - shortest path updated during algorithm with better option, if found

Basis for Dijkstra = edge relaxation:

```
// if new path to v is shorter, then use it!
if D[u] + w( u, v) < D[v] then
    D[v] = D[u] + w( u, v)
```

Etc.

- ➢ Positive weights only, negative weights break some algorithms (like Dijkstra)
- ➢ To use Dijkstra on unweighted graphs, use weight=1 for each edge
- ➢ Dijsktra algo has a single source (vert), but find the shortest path to *all* other verts from the source
- ➢ Edsger W. Dijkstra was a GIANT in computer science, en.wikipedia.org/wiki/Edsger_W._Dijkstra

**QOTD**

The art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos as effectively as possible.

- Dijkstra, en.wikiquote.org/wiki/Edsger_W._Dijkstra

## Pseudocode

Prep notes:

- ➔ d[v] = distance (sum of weights) from source to v; init to d[v] = INF
- ➔ prev[v] = previous edge on shortest path to v; init to path[v] = -1
- ➔ d[v] and path[v] are updated and improved over the iterations (relaxation!)
- ➔ controlling data structure: Priority Queue with verts ordered by distance, d[v]

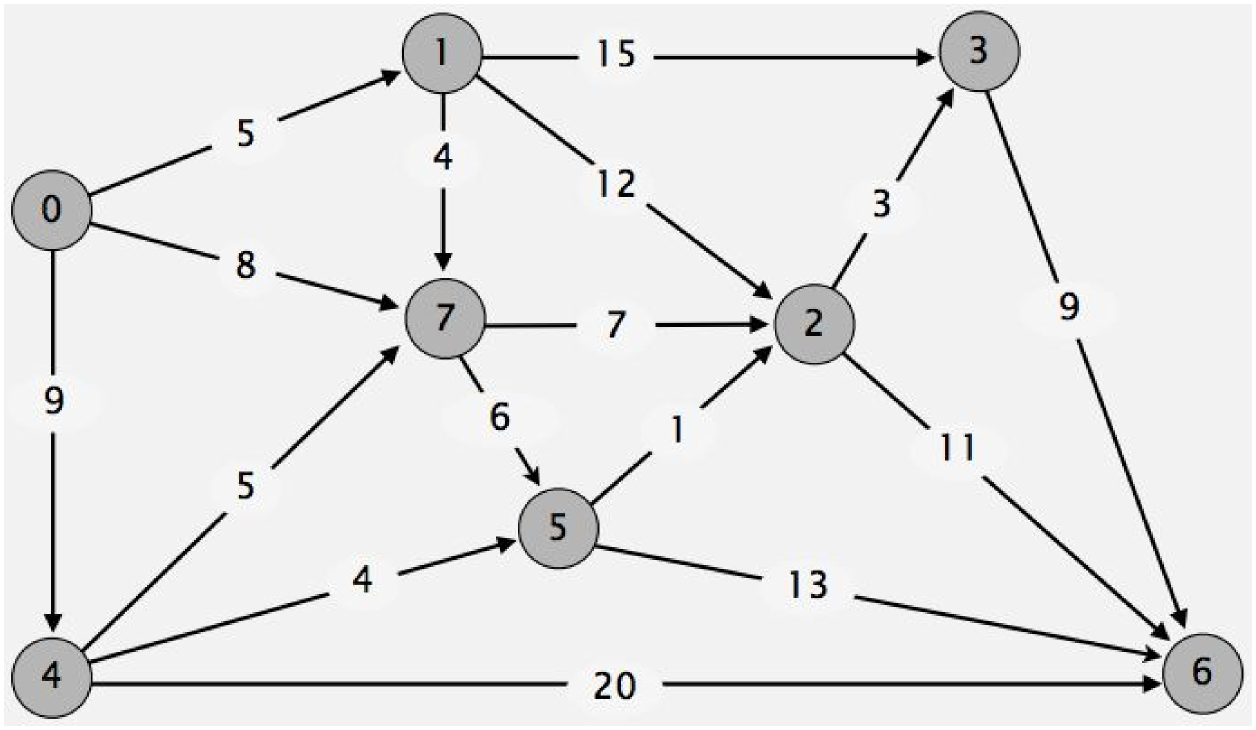
```
// Dijkstra's algorithm, single-source shortest path
shortestPath( graph G, vertex source)
   for all verts v    // init all vert with infinite distance and no prev edge
      d[v]=INF
      prev[v]=-1

   d[source] = 0    // distance to source is zero
   pq.enqueue(source)

   while pq not empty {
      u = pq.removeMin()
      for each edge (u, v) {
         if d[u] + weight( u, v) < d[v]
            d[v] = d[u] + weight(u, v)    // relaxation
            prev[v] = u
            if pq.contains( v) then pq.decreaseKey( v, d[v])
            else pq.enqueue( v)
      }
   }
```

Wrap notes:

- ➔ Shortest path now contained in d[] and prev[] arrays
  - ◆ Shortest path distance from source to any vert v is at d[v]
  - ◆ Reconstruct verts in the path by looping back from prev[v] to the source
- ➔ Performance: With binary heap, O(log V) operations (enqueue, decreaseKey, removeMin) performed for each edge = **O( E log V)**

Question: What are d, prev for an unreachable vertex?

Example (from Princeton): Find shortest paths, starting at vert 0



Solution is Princeton, Slide 38.

Dijkstra’s greedy algo is similar to Prim’s for min spanning tree… coming up!

## D. Min Spanning Tree: Prim, Kruskal

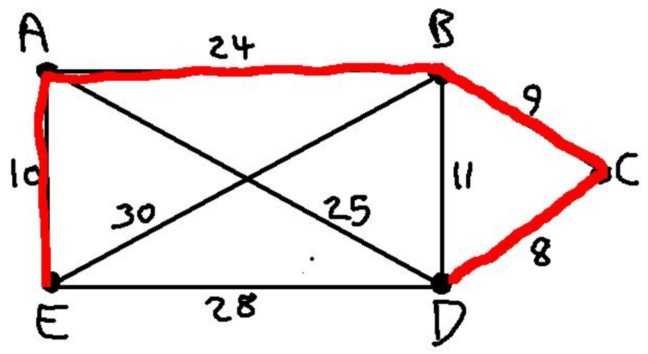
** Online: Princeton section, algs4.cs.princeton.edu/lectures/43MinimumSpanningTrees.pdf

** Animation:

- ➔ Prim's algorithm: www.cs.usfca.edu/~galles/visualization/Prim.html
- ➔ Kruskal's algorithm: www.cs.usfca.edu/~galles/visualization/Kruskal.html

**spanning tree** - a tree that contains every vertex in a connected graph (remember - tree means no cycles!); it's a list of edges

**min spanning tree** - the spanning tree where the sum of edge weights is smallest



**Prim’s Algorithm**

In English: Pick a vertex to be root of the tree. Find the min weight edge connected to the tree. Add that edge’s vertex. Repeat until all vertices are in the tree.
Similar to Dijkstra’s Algorithm for finding shortest path.
Animation: [www.cs.usfca.edu/~galles/visualization/Prim.html](www.cs.usfca.edu/~galles/visualization/Prim.html)

Pseudo-code:

```
PrimsMST( G, startv)
      create distance array, D[#vertices] = inf
      create parent array, parent[#vertices] = -1
      create known array, known[#vertices] = false

      D[startv] = 0   // start vertex is tree root
      add each D[i] to PriorityQueue PQ
      while PQ not empty
            u = PQ.removeMin()
            known[u] = true
            for each edge connected to u, (u, v)
                  if ! known[v]  and weight of edge < D[v]
                        D[v] = weight of edge
                        parent[v] = u
                        change D[v] key in PQ   // remove, and re-add to PQ
      for each vertex, v
            add edge ( v, parent[v]) to MST list
```

**Kruskal's Algorithm**

In English:

```
    sort the edges by weight
    for each edge
        add edge to MST if it doesn't create a cycle
```

Animation: www.cs.usfca.edu/~galles/visualization/Kruskal.html

Pseudo-code:

```
KruskalsMST( G)
      place each vertex in its own disjoint set
      sort all edges in G by weight
      for each edge (u,v)
            ds1 = find disjoint set of u
            ds2 = find disjoint set of v
            if ds1 != ds2
                  add edge to MST list
                  union( ds1, ds2)    // merge 2 disjoint sets into 1
```

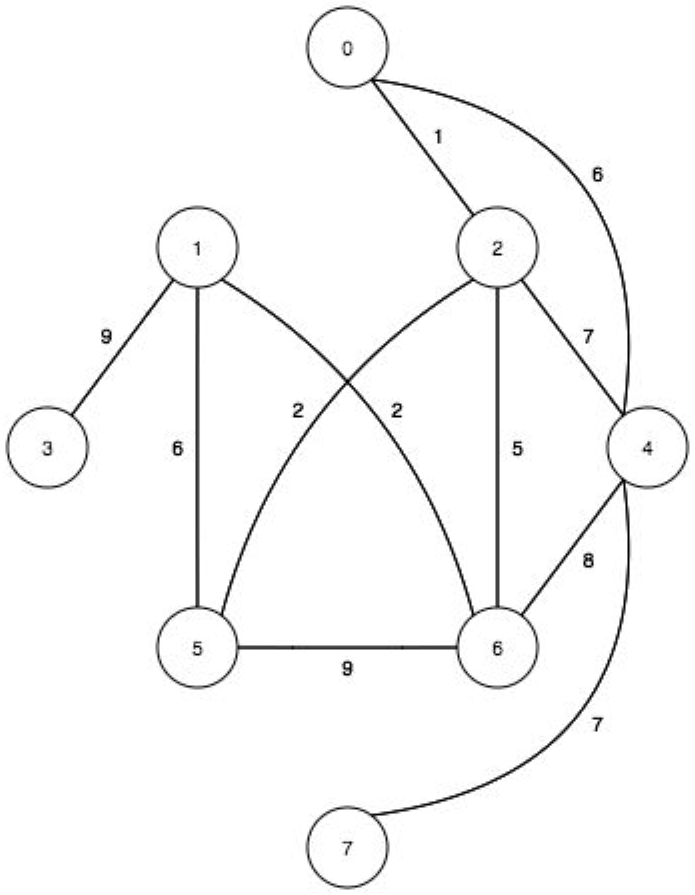
What's are disjoint sets? What is find()? Union?

Answer: Disjoint set is a collection of sets whose members don't intersect.

We use a nifty representation of disjoint sets (an array) to efficiently determine if adding edge would create a cycle. A lot of people use disjoint sets, eh...

    en.wikipedia.org/wiki/Disjoint-set_data_structure

Example:

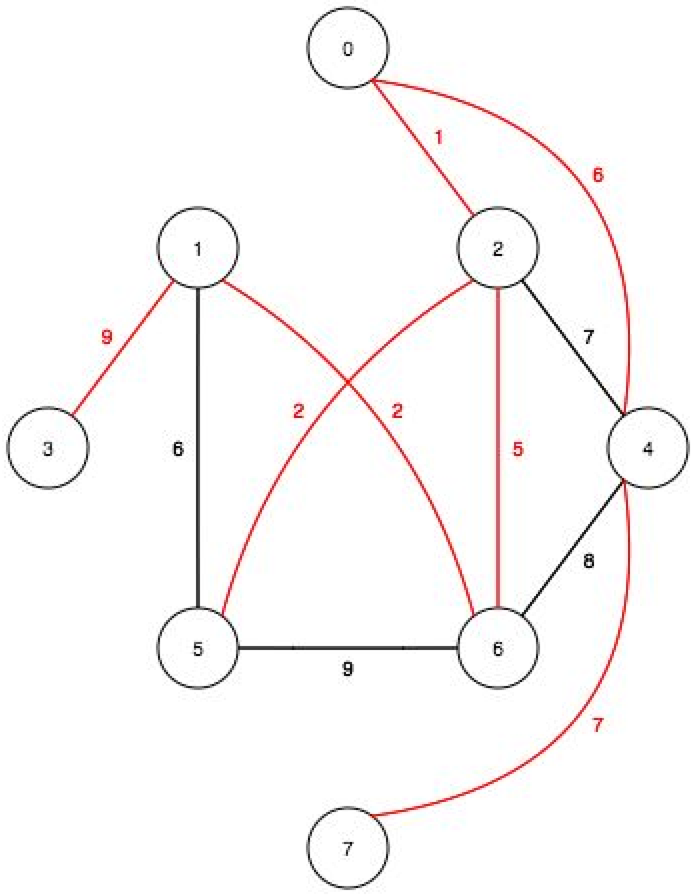


Source: www.cs.usfca.edu/~galles/visualization/Prim.html

1) Run Prim's

2) Run Kruskal's

Answer:


0
1
2
3
4
5
6
7
1
6
9
7
6
2
2
5
8
9
7