CSC 421: Applied Algorithms and Structures Week 9 (Winter)

“When you come to a fork in the road, take it.” – Yogi Berra

“The shortest distance between two points is under construction.” – Leo Aikman

Graph facts

- Connected graph: There is a path between every pair of vertices.
- Tree: A connected graph with no cycles; if a tree has n vertices, it must have $n-1$ edges.
- Empty graph: A graph with no edges
- Complete graph: A graph with all possible edges; a complete graph with n vertices will have $\binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$ edges.
- Component: A connected subgraph of a disconnected graph.

Depth first search (DFS)

Starting at some vertex, visit a neighbor and continue the search from there. Once the search is completed from that neighbor, continue the search from the next neighbor.

In other words, go as far as you can before having to *backtrack*.

Depth first search

- See DFS and DFSall algorithms in the textbook.
- Note pre-visit and post-visit method calls.
- Note pre-processing method call.
- Add clock to DFS (figure 6.3).
- Vertices are *new*, *active*, or *finished.*
- Edges are *tree*, *forward*, *back*, or *cross.*

DFS algorithm

```
DFS(v):
    mark v
    PreVisit(v)
    for each edge $v \to w$
        if w is unmarked
            parent(w) ← v
            DFS(w)
    PostVisit(v)
```

DFS applications

- Connectivity
- Connected component labeling
- Directed cycle finding
- Topological sort in a DAG (reverse postorder)
- Strongly connected component finding (Kosaraju-Sharir)

Breadth-first search (BFS)

Starting at some vertex, visit all of its neighbors. Then continue from each of the neighbors, visiting each of their as-yet unvisited neighbors.

You need to keep track of when a vertex is first visited. If you also track the edge traversed for this first visit, you can construct a breadth-first traversal tree.

BFS algorithm

```
BFS($s$):
    InitSSSP($s$)
    Push($s$)
    while the queue is not empty
        $u \leftarrow$ Pull( )
        for all edges $u \rightarrow v$
            if $dist(v) > dist(u) + 1$
                $dist(v) \leftarrow dist(u) + 1$
                $pred(v) \leftarrow u$
                Push($v$)
```

Queue operations Push and Pull

BFS: applications

- Find shortest unweighted paths
- Find cycles (encountering a visited vertex)
- Find connected components (a vertex with $d = \infty$ is in a different component from the start vertex)
- Is the graph 2-colorable? Equivalently, does it contain any odd-length cycles?

BFS: analysis

The initial for-loop executes once for each vertex.

The while-loop depends on the entries in the queue. Each iteration dequeues a vertex but the for-loop starting at line 12 iterates once for each edge.

Running time: $\Theta(n + m)$

BFS and DFS

Breadth-first search relies on a queue. It discovers a vertex when it is dequeued from the queue. It does not return to the vertex.

Depth-first search relies on a stack (run-time stack or explicit stack). It discovers a vertex v from a previously discovered vertex u, continuing the search from there. But at some point, it might resume the search from u.

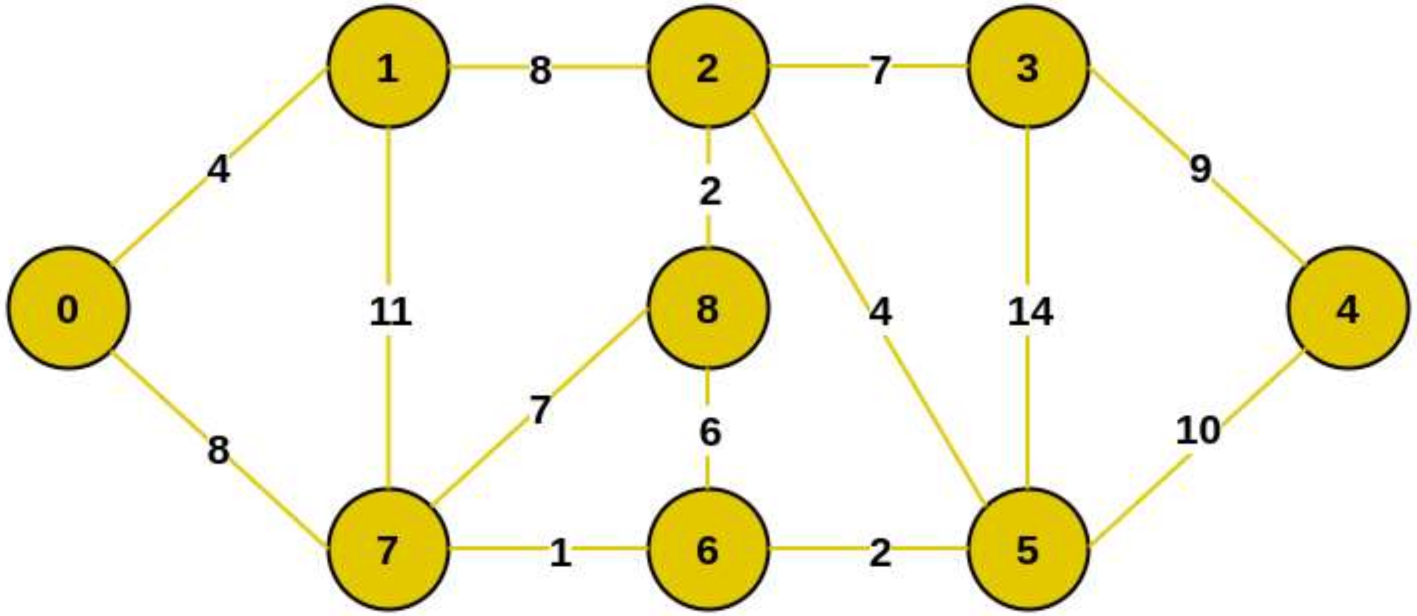
Minimum weight spanning tree

- Weights on the edges
- Definition of minimum weight spanning tree (MST)
- Build an MST edge by edge. At each step, the (perhaps disconnected) graph formed is called F (for forest).
- Add an edge if it's ***safe***. An edge is safe if it is the minimum weight edge with exactly one vertex in a component of F. That is, both of the edge's vertices cannot be in the same component of F.
- Prim's algorithm: Add safe edge from tree T built so far
- Kruskal's algorithm: Add least cost safe edge.

Shortest paths

- Bellman-Ford (relies on **INITSSSP** and **RELAX**)
- Dijskstra’s algorithm
- Priority queues and heaps

Jarník's/Prim's MST, Kruskal's MST

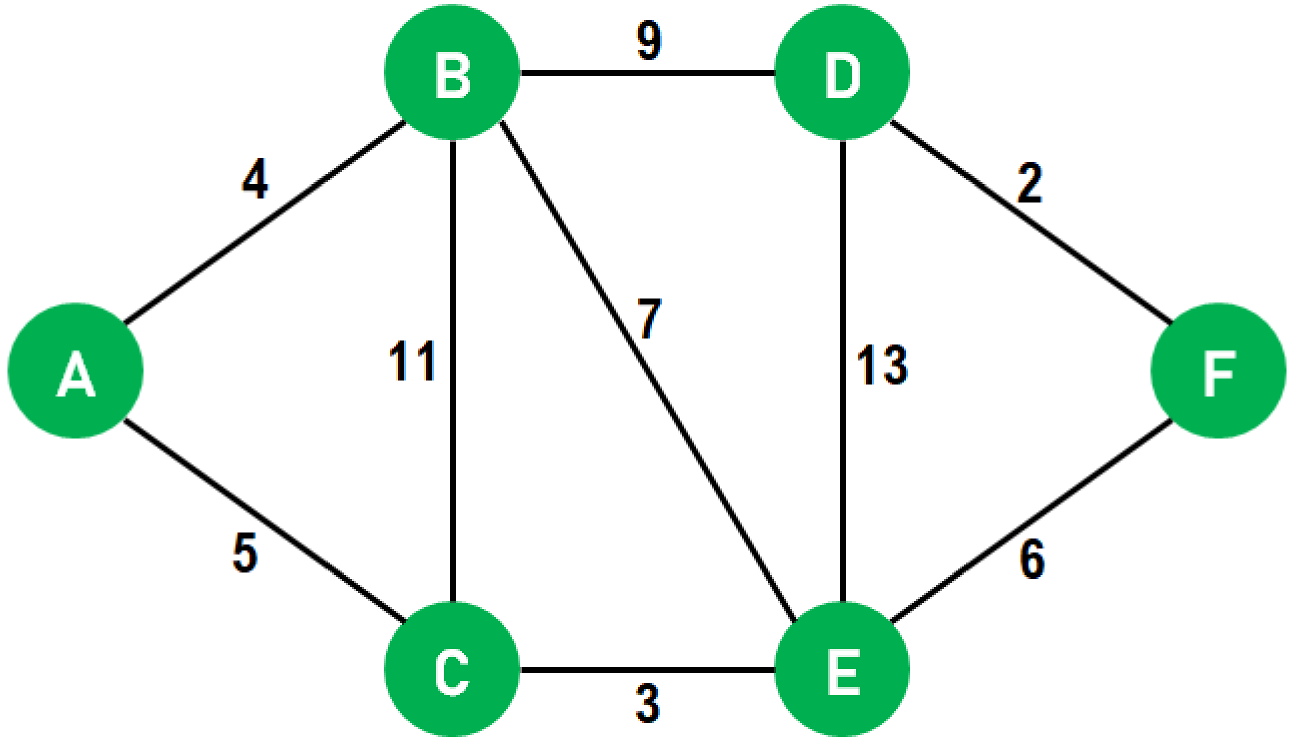


Courtesy of Geeks For Geeks website (http://geeksforgeeks.org)

Abstract data types, data structures

- ADT: Minimum priority queue
- Data structure: Minimum heap (binary tree), implemented as an array
- ADT: Disjoint set union-find
- Data structure: Tree, implemented as an array

Bellman-Ford SSSP, Dijkstra's SSSP


A
B
C
D
E
F
4
5
9
11
7
3
13
2
6