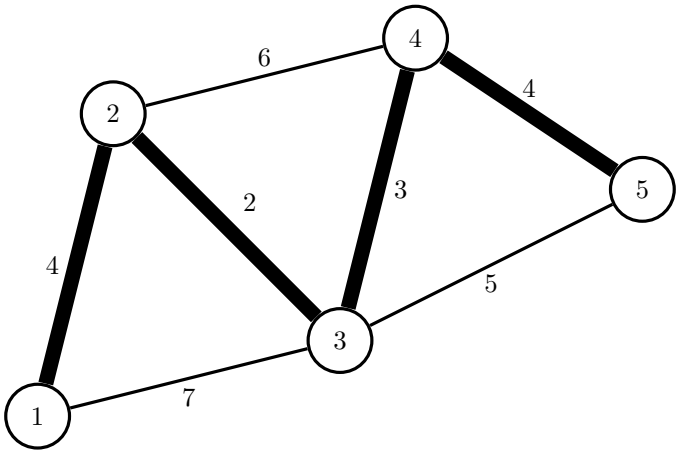
Advanced Graph Theory

Daniel Wisdom

September 29, 2017

1 Minimum Spanning Tree

Consider a connected, undirected graph. A *spanning tree* is a subgraph that is a tree and contains every vertex in the original graph. A *minimum spanning tree* (MST) is a spanning tree such that the sum of the edge weights of the tree is minimized. If we had a graph of cities and the distances between them, the minimum spanning tree would be the shortest set of roads to build that made every city reachable from every other city. Finding the minimum spanning tree uses union-find, which was discussed earlier.



1.1 Kruskal's Algorithm

Kruskal's algorithm constructs an MST by greedily adding edges. We initialize every node as being in its own tree. Then, loop over all edges in increasing order of weight. If the edge connects two nodes in different trees, merge those trees and add that edge to the MST. If an edge connects two vertices from the same tree, those two nodes are already connected and we just discard that edge. Conveniently, we have a union-find algorithm that can merge components and find the parent of a node in basically linear time.

The MST of n nodes always contains exactly $n-1$ edges, so we can stop looking through more edges after adding $n-1$ edges to the MST.

Algorithm 1 Kruskal

```
for all edges (u, v) in sorted order do
    if FIND(u) ≠ FIND(v) then
        add (u, v) to spanning tree
        UNION(u, v)
```

This algorithm requires a sort of the edges and thus has complexity $O(E \log E) = O(E \log V)$.

2 Shortest Paths

2.1 Floyd-Warshall

Floyd-Warshall solves the shortest path problem for all pairs of vertices in $O(V^3)$ time, which is faster than V single-source Dijkstra runs on a dense graph. Floyd-Warshall is very simple and quick to code, making it useful in contests with high execution time limits.

The algorithm uses a table of the minimum distance between every pair of nodes. Every entry in the table in initialized to $+\infty$. The distance from each node to itself is 0. The distance from the start to the end of each edge is also set to that edge's weight.

We then loop over every triple of nodes k, i, and j. Node k is the "bridge" that connects pairs of i and j. Specifically, if i can reach k and k can reach j, then the distance from i to j may be shortened to the sum of the distance from i to k and k to j. The order of these loops is important; The "bridge" node must always be the outermost loop.

Floyd-Warshall works on graphs with negative edge weights, unlike Dijkstra. In graphs which contains cycles with net negative path weights the shortest path becomes $-\infty$. Floyd Warshall will not report a length of $-\infty$, but it can easily be modified to detect negative cycles. After the algorithm finishes we can check the distances from each node to itself. If the $dist(i, i) < 0$, node i must be part of a negative cycle.

Algorithm 2 Floyd-Warshall

```
for all vertices v do
    dist(v, v) = 0
for all edges (u, v) do
    dist(u, v) = weight(u, v)
for all vertices k do
    for all vertices i do
        for all vertices j do
            if dist(i, j) > dist(i, k) + dist(k, j) then
                dist(i, j) ← dist(i, k) + dist(k, j)
```

2.2 Bellman-Ford

Bellman-Ford is a single-source $O(VE)$ shortest path algorithm that works when edge weights can be negative. It is preferable to Floyd-Warshall when the graph is sparse and we only need the answer for one source. Like Floyd-Warshall, the algorithm fails if the graph contains a negative cycle, but the algorithm is still useful for detecting negative cycles.

The key observation here is that the shortest path, assuming no negative cycles, has length at most $V - 1$. If we loop over all edges and consider if they form a new shortest path to their endpoint we can find all shortest paths of length 1. If we repeat this loop $V - 1$ times we discover all paths of length up to $V - 1$.

To detect a negative cycle we simply run the loop one last time. If any shorter paths are discovered, then the optimal path is longer than $V - 1$ nodes. The only way this is possible is if there is a negative cycle.

Algorithm 3 Bellman-Ford

```
for all vertices v do
    dist(v) ← ∞
    prev(v) ← −1
dist(src) ← 0
for i = 1 to V − 1 do
    for all edges (u, v) do
        if dist(u) + weight(u, v) < dist(v) then
            dist(v) ← dist(u) + weight(u, v)
            prev(v) ← u
for all edges (u, v) do                          ▷ check for negative cycles
    if dist(u) + weight(u, v) < dist(v) then
        negative cycle detected
```

3 Problems

1. Silver, February 2015 USACO Contest Problem 3, Superbull:

Given $N \leq 2000$ teams and the "excitement" caused by a match between every pair of teams, find the most excitement possible in a tournament. The loser is eliminated after every game, but you get to pick who loses.

2. Give an algorithm for finding the spanning tree with the smallest product of edge weights, assuming all edge weights are positive.

3. Bessie the cow is walking through some very corrupt cities in Zimbabwe. The cities are connected by a set of roads, and each road has some bandit associated with it. Some bandits are stronger than her, and will require a specific fee -X_i for one safe passage. Other bandits are weaker than Bessie, so she can force them to give her up to X_i amount of money each time she traverses the road. Please help her compute the maximum amount of money she can obtain on her journey from Harare to Kwekwe.