CS5050 Advanced Algorithms

Philip Nelson

Spring Semester, 2018

Assignment 7: Graph Algorithms II

Worked with: Ammon H, Raul R, Hailee M, and Jack K

Note: In this assignment, we assume all input graphs are represented by adjacency lists.

1. **(20 points)** Given a directed graph G of n vertices and m edges, each edge (u, v) has a weight $w(u, v)$, which can be positive, zero, or negative. The *bottleneck-weight* of any path in G is defined to be the **largest** weight of all edges in the path. Let s and t be two vertices of G. A *minimum bottleneck-weight path* from s to t is a path with the smallest bottleneck-weight among all paths from s to t in G.

   Modify Dijkstra's algorithm to compute a minimum bottleneck-weight path from s to t. Your algorithm should have the same time complexity as Dijkstra's algorithm.

Solution

In order to find a path from s to t with the lowest bottle-neck, we will modify Dijkstra's algorithm. To begin, initialize the bottle-neck, $v.b$, for all $v \in G$ to ∞ and $s.b$ to $-\infty$. Then following Dijkstra's algorithm, we will prioritize the path with the smallest bottle-neck.

```
lowestBottleNeck(G, s, t) {
  for each v in graph {
    v.pre = null
    v.b = inf
  }
  s.b = -inf
  Q : add all vertices

  while(!Q.empty()) {
    u = Q.extractMin()
    for each v in adj(u) {
      if(v.b == inf) {
        v.b = max(u.b, w(u, v))
        v.pre = u
      }
      else {
        min = min(v.b, max(w(u, v), u.b))
        if (min != v.b) {
```

```
            v.pre = u
          }
          v.b = min
        }
        Q.decreaseKey(v, v.b)
      }
    }
}
```

This algorithm has the same running time as Dijkstra's algorithm, $O(m + n \log n)$, because each vertex is removed once from the priority queue, and each edge is followed once. Then, the extract min operation is performed n times and the decrease key operation is performed m times which takes $O(\log n)$ and $O(1)$ time respectively. This gives the function lowestBottleNeck the time complexity of $O(m + n \log n)$

In order to retrieve the path, simply push and follow the predecessors onto a stack then read the sack.

2. Let $G = (V, E)$ be an undirected connected graph, and each edge (u, v) has a positive weight $w(u, v) > 0$. Let s and t be two vertices of G. Let $\pi(s, t)$ denote a shortest path from s to t in G. Let T be a minimum spanning tree of G. Please answer the following questions and explain why you obtain your answers.

   (a) Suppose we increase the weight of every edge of G by a positive value $\delta > 0$. Then, is $\pi(s, t)$ still a shortest path from s to t? **(10 points)**

   Solution

   No, the following is a counter example: the two paths from s to t,

   $$s \xrightarrow{1} a \xrightarrow{1} b \xrightarrow{1} c \xrightarrow{1} s \text{ and } c \xrightarrow{5} s$$

   With the current edge weights, the first path is lighter, however; if 1 were to be added to each edge, the better path would become the second path. Longer paths gain more weight when a fixed amount is added to each vertex.

   (b) Suppose we increase the weight of every edge of G by a positive value $\delta > 0$. Then, is T still a minimum spanning tree of G? **(10 points)**

   Solution

   Yes, because of to the cut property. If the vertices are partitioned into two sets, and the edge with the lowest weight is selected to be part of the minimal spanning tree, the same edge would still be selected even if some $\delta > 0$ were added to each edge.

3. **(20 points)** Let $G = (V, E)$ be an undirected connected graph of n vertices and m edges. Suppose each edge of G has a color of either *blue* or *red*. Design an algorithm to find a spanning tree T of G such that T has as few red edges as possible. Your algorithm should run in $O((n + m) \log n)$ time.

Solution

To solve this problem we can reduce the problem to a different problem that can be solved with Prim's algorithm. Begin by weighting red edges with 1 and blue edges with 0. This reduction can be done in $O(m)$ time. Then we will use Prim's algorithm for finding minimum spanning trees. This method is correct because you will always select the less weighted edge which, given the option, will be a blue edge.

The time complexity can be $O((n + m) \log n)$ or $O(m + n \log n)$ depending on which priority queue is used, min heap or Fibonacci heap.

Total Points: 60