CS 170 DIS 04

Released on 2018-09-24

1 Minimum Spanning Trees (short answer)

(a) Given an undirected graph $G = (V, E)$ and a set $E' \subset E$ briefly describe how to update Kruskal's algorithm to find the minimum spanning tree that includes all edges from E'.

(b) Assume you are given a graph $G = (V, E)$ with positive and negative edge weights and an algorithm that can return a minimum spanning tree when given a graph with only positive edges. Describe a way to transform G into a new graph G' containing only positive edge weights so that the minimum spanning tree of G can be easily found from the minimum spanning tree of G'.

(c) Describe an algorithm to find a maximum spanning tree of a given graph.

Solution:

(a) Assuming E' doesn't have a cycle, add all edges from E' to the MST first, then sort $E \backslash E'$ and run Kruskal's as normal.

(b) We can use the attempted fix to Dijkstra's Algorithm described in problem 3 of discussion 3. We create G' by adding a large positive integer M to all the edge weights of G so that each edge weight is positive. The minimum spanning tree of G' is then the same as the minimum spanning tree of G.
Unlike Dijkstra's algorithm, which is finding minimum paths which may have different numbers of edges, all spanning trees of G must have precisely $|V| - 1$ edges, conserving the MST. So, if the minimum spanning tree of G has weight w, the minimum spanning tree of G' has weight $w + (|V| - 1)M$.

(c) Negate all edge weights and apply the algorithm from the previous part.

2 Picking a Favorite MST

Consider an undirected, weighted graph for which multiple MSTs are possible (we know this means the edge weights cannot be unique). You have a favorite MST, F. Are you guaranteed that F is a possible output of Kruskal's algorithm on this graph? How about Prim's? In other words, is it always possible to "force" the MST algorithms to output F? Justify your answer.

Solution: Yes; for both MST algorithms, it's possible to ensure they output F, provided it is indeed an MST.

First, consider Kruskal's algorithm. Make sure that the edges of F are always before any other equally-weighted edges after the sort. Now, it will add all such edges as early as possible. Consider towards a contradiction the case where at any point Kruskal's declines to add an edge e in F to the MST. This means that e would have created a cycle with other equally-weighted edges in F, and/or lighter edges (possibly in F). (Before this point, Kruskal's may have added edges not in F to the MST.) But then it's impossible that both e and all of the other equally-weighted edges in F in this cycle can be in any MST, as one of them is the heaviest edge in some cycle of the graph, and such edges cannot be in any MST.

Since we assumed that F is an MST, this is a contradiction. Therefore we conclude that Kruskal's algorithm will add all edges in F to the MST. And since all MSTs have the same number of edges, this means it cannot add any edges not in F.

Now, consider Prim's algorithm. As it expands the fringe, have it only choose edges in F (so when there are multiple lightest edges to choose, choose a lightest edge in F). If this strategy fails, there must have been some cut across which none of the lightest edges were in F. But if this is the case, F cannot have been an MST (one of the lightest edges across any cut must be in any MST). Given that F must be an MST, this strategy will work.

3 MST Variant

Give an undirected graph $G = (V, E \cup S)$ with edge weight $c(e)$. Note that S is disjoint with E. Design an algorithm to find a minimum one among all spanning trees having at most one edge from S and others from E.

Input: A graph $G = (V, E)$, set of potential superhighways S, and a cost function $c(e)$ defined for every $e \in E \cup S$.

Output: A tree $T = (V, E')$ such that T is connected (there is a path in T between any two vertices in V), $E' \subseteq E \cup S$, $\sum_{e \in E'} c(e)$ is minimized, and $|E' \cap S| \leq 1$.

Solution:

Main Idea: Use Prim's (or Kruskal's) algorithm to find the MST T' of G. Then for each potential $s \in S$, try adding s to T' and keep track of the difference between $c(s)$ and $c(e)$ where e is the edge of highest weight along that cycle. If $c(s) < c(e)$, then adding s to T' and removing e from T' creates a new spanning tree with lower total weight than T'. Doing this for each $s \in S$ and taking the smallest resulting spanning tree gives us T.

Pseudocode:

```
procedure RoadMST(graph $G$, additional edges $S$, cost function $c$)
    Use Prim's to find $T' = (V, E')$, the MST of $G$
    $bestDifference = 0$
    $bestEdges = E'$
    for $s \in S$ do
        Add $s$ to $T'$ and use DFS to find the cycle containig $s$. Let $e$ be the most costly edge in that cycle.
        if $c(e) - c(s) > bestDifference$ then
            $bestDifference = c(e) - c(s)$
            $bestEdges = E' \cup \{s\} - \{e\}$
    return $(V, bestEdges)$
```

Running time analysis: Prim's takes $O(|E| \log |E|)$ to find the MST. For each edge in S, we run a DFS on T', which has $|V|$ vertices and $O(|V|)$ edges. Therefore, this takes $O(|S||V|)$ time. Thus, the overall running time is $O(|E| \log |E| + |S||V|)$.

Proof of correctness: The optimal solution containing exactly one edge from S cannot be better than the optimal solution containing exactly one edge from S and exactly $|V| - 2$

edges from T', the MST of G. To see this, assume towards contradiction that the optimal solution T contains some $e \in E$ such that e is not part of T'. Consider the cut in T defined by e, i.e. if $e = (u, v)$, then divide the vertices into those reachable from u and those reachable from v without using e. Let e' be the edge of lowest weight in T' that spans that cut. Adding e' to T creates a cycle with e. We know that $c(e') \leq c(e)$ because otherwise e would be in T' instead of e'. Therefore, removing e from T and adding e' yields a spanning tree that costs no more than T. Since we can do this for any edge in T but not in T', we can effectively transform T into a solution that is at least as good and contains no edges from E that are not in T'.

4 Service scheduling

A server has n customers waiting to be served. Customer i requires t_i minutes to be served. If, for example, the customers were served in the order $t_1, t_2, t_3, \ldots$, then the ith customer would wait for $t_1 + t_2 + \cdots + t_i$ minutes.

We want to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)$$

Given the list of t_i, give an efficient algorithm for computing the optimal order in which to process the customers.

Solution: We simply proceed by a greedy strategy, by sorting the customers in the increasing order of service times and servicing them in this order. The running time is $O(n \log n)$.

To prove the correctness, for any ordering of the customers, let $s(j)$ denote the jth customer in the ordering. Then

$$T = \sum_{i=1}^{n} \sum_{j=1}^{i-1} t_{s(j)} = \sum_{i=1}^{n} (n-i) t_{s(i)}$$

For any ordering, if $t_{s(i)} > t_{s(j)}$ for $i < j$, then swapping the positions of the two customers gives a better ordering. Since, we can generate all possible orderings by swaps, an ordering which has the property that $t_{s(1)} \leq \ldots \leq t_{s(n)}$ must be the global optimum. However, this is exactly the ordering we output.