Algorithms Homework 9

Andrew Osborne

April 18, 2019

Overhead

Before I begin addressing the problems at hand, it is useful to consider some auxillary facts which clear up the first two problems. When I refer to some set of edges, or to a tree, I refer to the set of edges therein. Moreover, if I refer to the vertices in some set of edges, I am referring to the set of vertices upon which some edge in our set is incedent. I make this distinction to justify some loose language in the following discussion.

Theorem 1. *For some graph G with vertices V and edges E, if T is the set of edges in a MST of G, then $T \cup \{e\}$ has exactly one cycle, where e is an edge in E which is not already in T.*

Proof. Take some edge not in T, say $e = (u, v)$. T is a spanning tree and therefore there is some path of edges in T which connects u to v. By a result in the text, this path is unique. When we add e to T, there is a second path from u to v. Then the union of the edges in T which connect u to v with e forms a cycle containing e. □

Theorem 2. *As before, if T is some MST of a graph G, if we add some edge e to T and then remove the greatest element of the cycle in T containing e, what remains is also a MST.*

Proof. Take the hypothesis of the theorem. Let C be the cycle of edges in $T \cup \{e\}$ which contains e. Then there are two distinct paths between any two vertices. If we were to remove some edge, say (u, v) from C, then we would only remove one of the two paths from u to v, and another would remain, along which all other vertices of C lie. This is to say that, if we add one edge to a MST, and then remove an edge from the created cycle, we will retain the spanning property. On the other hand, if we remove the edge with the greatest weight from C, then we have reduced or unaffected the sum of weights of the edges in C and therefore in T. Then, clearly, add some edge to an MST and then remove the greatest element from the subsequently created cycle, the result will be an MST.

□

23.1-1

With the results of Theorem 2 in mind, let us consider a MST, T, of a graph G. If $e \in G$ is a minimum–weight edge and $e \notin T$, then we may add e to T and then remove the largest edge from the created cycle containing e. However, since e is an edge of least weight, all edges in the cycle of e are greater than or equal to e in weight. If there is an edge with a greater weight that e in the cycle containing e, then we have reduced the cost of T which was assumed to be a MST, which is a contradiction, so there may not exist any such edge. Then, every edge in the cycle containing e must have the same weight as e, and we may remove any element other than e from said cycle to obtain an MST which contains e.

The above supposes that there is some MST which does not contain the element of minimum weight of some graph. This is actually only permitted if there are multiple edges with least weight and we are free to suppose that there is some MST which does not contain an edge of least weight because, otherwise, there is nothing to show.

23.1-5

This argument is quite similar to the last. If m is an edge of maximal weight in a MST, T, of a graph G then we seek the existence of an MST which does not contain m provided that m is in some cycle in G. Suppose C is the set of edges which form some cycle containing m. Then, since T is spanning, there is some path in T between any pair of vertices in C. There is then some edge in $G - T$ which will create a cycle involving m and since m is an element of greatest weight, we are free to remove it. That is, to construct the desired MST, we need only find vertices u and v of G so that $(u, v) \in C \cap G - T$ which is guaranteed by our hypothesis.

We could alternatively argue that m is a light edge for no cut which splits the vertices of C, and therefore will not be a safe edge for any cut, from which our goal follows immediately. Note, however that the first argument is better because it shows that a max–weight edge cannot participate in a MST unless its weight is non–unique.

23.2-2

I will submit this argument as pseudocode. Note that, if we pass our graph as an adjacency matrix `A` , we have already a natural way to index our vertices.

```
// assume A is an adjacency matrix of dimensions V x V
// assume w is a weight function which takes two integers as inputs
// assume Min(Array) returns the index of the element with the minimum
    value in Array
A-PRIM(A,w)
  V,_ = size(A)
  let Af[1..V,1..V] be a new array of zeros
  let val[1..V] be a new array of infinities
  let index[1..V] be a new array of zeros

  for i = 1 to V
    if A[1,i] == 1
      val[i] = w(1,i)
      parent[i] = 1

  numAdded = 0
  while numAdded <= V-1
    ind = min(val)
    par = parent[ind]
    Af[par,ind] = Af[ind,par] = 1
    numAdded += 1

    for i = 1 to V
      if A[ind,i] == 1 and val[i] > w(ind,i)
        parent[i] = ind
        val[i] = w(ind,i)
    numAdded += 1
  return Af
```

This algorithm begins by tabulates the smallest edge from any node outside of the computed tree to some element in the computed tree. At the beginning of each pass, the smallest such edge is added to the computed tree and the tabulated values are updated if there exist edges from outside the tree to the newly added vertex. Note that we check each element of the $V \times V$ matrix `A` exactly once and therefore this algorithm clearly runs in $O(V^2)$ time. Note that this makes the assumption

that `A` is symmetric (undirected graph). And the final MST is returned in the $V \times V$ matrix `Af` which is also an adjacency matrix.

24.2-3

The solution to this problem is wrapped up quite nicely in the definition of m. The maximum over the edges in a graph G of the minimum number of nodes in a shortest path from a single source s is precisely m. This is to say that, when m passes of the Bellman-Ford algorithm have occured, then each edge of G has been relaxed m times, and each shortest path has less than or equal to m constituent edges. Then, at such a point, all shortest paths will be formed! Furthermore, if all shortest paths are formed, no relaxations will result in any change after m passes. On the other hand, if there is a pass in which no relaxations result in change, the the next pass will also result in no change and so on. Then, If we are sure that there are no negative weighted cycles in our graph, we may simply stop our operation when an entire pass is completed without changing any vertex values by relaxation. This will be on the $(m + 1)$th pass.