

Solutions for Tutorial Exercise 2: MST and Dynamic Programming

1. **Jot's Non-deterministic MST Algorithm.** Professor Jot has proposed the following algorithm to solve for the Minimum Spanning Tree of an undirected, connected, weighted graph $G = (V, E, w)$:

```
[T] = MinSpanningTree(V, E, w)
% Return the set of edges T in a minimum spanning tree of the
% input graph G = (V, E, w) with edge weights w(e).
% Precondition: G is a connected undirected graph.

Initialize T to be the edges of some spanning tree of G (say by using
depth first search from a starting vertex  $v \in V$ , details omitted).
That is, suppose  $(V, T)$  has been initialized to be a spanning tree of G.

Loop:
  Pick any edge  $e = (u, v) \in E \setminus T$  such that the simple path
  from  $u$  to  $v$  in  $T$  contains an edge  $f = (x, y)$  such that  $w(f) > w(e)$ .
  If there is no such edge  $e$  in  $E \setminus T$ :
    Break out of this loop.
  Else:
    Set  $T = (T \setminus \{f\}) \cup \{e\}$ 
  End if
End loop
return T
```

Note that the Prof. Jot's algorithm doesn't specify which set of edges should be used for the initial spanning tree (V, T) . He is claiming any set will do, so long as the graph (V, T) is a tree. Moreover, the algorithm doesn't specify exactly how to select the edge e to be used at each iteration, nor which of several possible edges f to use. Multiple choices are possible, and Jot is simply assuming that some such choice is made. This is an example of a non-deterministic algorithm. If you don't want to think about non-determinism, then whenever the algorithm is faced with a choice, imagine randomly shuffling over the set of possible choices and then selecting whatever choice comes first after shuffling. (Although, note that you are potentially shuffling over a huge number of options.)

- Assume that the initialization is correct, so the original subgraph (V, T) of G is indeed a spanning tree. Show that, at the end of each pass through the loop, the updated graph (V, T') is also a spanning tree, where T' is the updated set of edges $T' = (T \setminus \{f\}) \cup \{e\}$. (**Spoiler-level Hint:** See the proof of Prim's algorithm at <http://www.cs.toronto.edu/~jepson/csc373/lectures/Prim.txt>.)
- Suppose the "else" clause is executed, and let $T' = (T \setminus \{f\}) \cup \{e\}$. That is, let T be the set of edges in the previous tree, and T' be the set for the updated tree. Define $w(T) = \sum_{e \in T} w(e)$ to be the total weight of the edges in tree T . Then show $w(T') < w(T)$.
- Given (a) and (b) above, show the algorithm terminates after a finite number of iterations through the loop. (Hint: **Terminates**, but perhaps not with the correct solution.)
- For the case for which the input graph $G = (V, E, w)$ has edges with distinct weights, prove the algorithm is correct. That is, assuming $w(e) \neq w(f)$ for all edges $e, f \in E$ with $e \neq f$ then, when this algorithm terminates, it outputs the edges T of the MST (V, T) .
- [OPTIONAL]** Is the algorithm correct in the general case of weights, that is, including the cases for which the weights $w(e)$ need not be distinct? If so, how would you prove it?

Solution for Q1:

1a The following argument closely follows that in

<http://www.cs.toronto.edu/~jepson/csc373/lectures/Prim.txt>.

Consider the loop invariant $L(k)$ which states that, after k iterations of the loop, the set of edges T is such that the graph (V, T) in the algorithm is a spanning tree of the original graph $G = (V, E, w)$. We will use induction to prove $L(k)$ is true for all integers $k \geq 0$, or until the algorithm breaks out of the loop.

For $k = 0$, $L(0)$ is true by the initialization step.

Suppose $k \geq 0$ is an integer, and assume $L(k)$ is true.

To prove $L(k + 1)$ we need to show the graph (V, T') where $T' = (T \setminus \{f\}) \cup \{e\}$ is a spanning tree of G , where we know from $L(k)$ that (V, T) is a spanning tree of G .

If the loop stops on the $(k + 1)^{st}$ iteration then T is unchanged and $L(k + 1)$ follows.

Otherwise, it must be the case that the algorithm has found an edge $e = (u, v) \in E \setminus T$ and an edge $f \in T$ with the properties specified in the algorithm. By the loop invariant $L(k)$, the graph (V, T) is a spanning tree and therefore $|T| = |V| - 1$. The new set of edges $T' = (T \setminus \{f\}) \cup \{e\}$ has the same number of edges as T does (since $f \in T$ and $e \notin T$). And therefore $|T'| = |V| - 1$. If we can show that (V, T') is also a connected graph then it will follow (from Property 2 of trees in the MST lecture slides) that (V, T') is a spanning tree.

To show (V, T') is connected we will first build two simple paths, namely P_1 and P_2 , which are in the graph $(V, T \setminus \{f\})$, with one path connecting u with one endpoint of f and while the other connects v with the other endpoint of f . To construct these paths note that, since T is a spanning tree, there is a unique simple path P from u to v (which cannot contain e since $e \notin T$). By the statement of the algorithm, the edge $f = (x, y)$ is on this path P . Since the path is simple, the edge f can only appear once on P . Without loss of generality, we will assume that the vertex x comes before y in the path P from u to v (otherwise, swap the names of x and y).

Thus P has the form, $P = (a(1), a(2), \dots, a(k))$, $a(1) = u$, $a(k) = v$. Note this path must traverse more than one edge (i.e., $k > 2$) since otherwise it would consist of the single edge (u, v) , which in turn would contradict $e \notin T$. Since $f = (x, y)$ appears exactly once on this path and, by assumption, in that order, there must be a j , with $1 \leq j < k$ such that $(a(j), a(j + 1)) = (x, y)$.

Define two sub-paths of P to be $P_1 = (u, a(2), \dots, x)$ and $P_2 = (y, a(j + 2), \dots, v)$ (this may abuse the definition of “path” to include a single vertex, but that does not raise any serious issues). Moreover, by construction, neither P_1 nor P_2 contain the edge $f = (x, y)$ (or its reverse). Therefore, $P_1 = (u, \dots, x)$ and $P_2 = (y, \dots, v)$ are simple paths in the graph $(V, T \setminus \{f\})$, as desired.

With these two sub-paths we can now show that the graph (V, T') is connected. Let $q, r \in V$ be any two distinct vertices. Since the current sub-graph (V, T) is connected (by the loop invariant $L(k)$) there exists a simple path $Q = (s(1), s(2), \dots, s(m))$ in (V, T) with $q = s(1)$ and $r = s(m)$. If Q does not contain the edge $f = (x, y)$, or its reverse, then Q is also a simple path in the updated graph, (V, T') .

Otherwise, since Q is a simple path, the edge $f = (x, y)$ must appear exactly once in Q . As for P above, it follows that Q must consist of more than one edge, so $m > 2$. Without loss of generality (i.e., perhaps by swapping the names q and r and the traversal direction of Q) we can assume $s(i) = x$ and $s(i + 1) = y$ for a unique i with $1 \leq i < m$. We now have two sub-paths (again with the same slight abuse of terminology here), $Q_1 = (q, \dots, x)$ and $Q_2 = (y, \dots, r)$, neither of which contain the edge f (or e for that matter).

Finally, consider the following path Q' from q to r , which is formed by concatenating these sub-paths

(denoted by operator “ $|$ ”) or perhaps their reverses (denoted by $\text{rev}(P_k)$):

$$Q' = Q_1 | \text{rev}(P_1) | (u, v) | \text{rev}(P_2) | Q_2.$$

Specifically, the sub-paths above go from q to x to u to v to y to r , respectively. Moreover, Q' must be a path in the graph (V, T') (since all the sub-paths here are in $(V, T \setminus \{f\})$ except for the single edge e , which is in T'). Therefore the vertices q and r are connected in (V, T') . (Note Q' need not be a simple path, and we only need to find a path.) Since q and r were any two distinct vertices in V , the above shows the graph (V, T') is connected. Together with $|T'| = |V| - 1$, it therefore follows that T' is a spanning tree. Thus $L(k+1)$ is true.

Therefore, by induction, $L(k)$ is true after each iteration of the loop.

- 1b By definition of $w(\cdot)$, e and f , we have $w(T') = w(T) - w(f) + w(e)$ (the relevant details are in part (a)). Since the algorithm ensures $w(f) > w(e)$, it follows that $w(T') < w(T)$.
- 1c Define $\Delta = \min\{|w(e) - w(f)| \mid e, f \in E \text{ and } w(e) \neq w(f)\}$. This Δ is the smallest, non-zero, absolute difference between weights. (For integer valued weights we can take $\Delta = 1$.) Since there are only finitely many edges and weights, it follows that $\Delta > 0$.

Suppose the algorithm does not terminate. Let (V, T_0) be the initial spanning tree, and let (V, T_k) be the result after the k^{th} iteration. By (a) and (b) we have that $w(T_{k+1}) \leq w(T_k) - \Delta$ for all $k \geq 0$. That is, $w(T_k) \leq w(T_0) - k\Delta$. This upper bound on $w(T_k)$ therefore becomes arbitrarily large in absolute value and increasingly negative as $k \rightarrow \infty$. But this is a contradiction since the minimum possible weight of a spanning tree of G is bounded below by, say, the number of edges $|V| - 1$ times the smallest edge weight $\min\{w(e) \mid e \in E\}$. Therefore the loop must terminate.

- 1d Suppose the algorithm terminates with the graph (V, T) . By (a) above, (V, T) is a spanning tree of $G = (V, E)$. We will prove (V, T) is a minimum spanning tree by contradiction. That is, suppose that there exists another spanning tree (V, T^*) , with $T^* \neq T$, such that $w(T^*) < w(T)$.

Then there must exist an edge $f = (u, v) \in T^* \setminus T$. By the statement of the algorithm it must be the case that $w(f) \geq w(e)$ for all edges e on the path P from u to v in (V, T) (otherwise the algorithm would not have stopped).

By the distinctness of the weights on E , it must then be the case that $w(f) > w(e)$ for all edges e on P . Pick one such e . Then a similar exchange argument to the argument in (b) shows that $(V, T^* \setminus \{f\} \cup \{e\})$ is another spanning tree of G . Moreover, $w(T^* \setminus \{f\} \cup \{e\}) = w(T^*) - (w(f) - w(e)) < w(T^*)$. This contradicts T^* being a minimal spanning tree.

- 1e The issue here is that, if $w(e) = w(f)$, then the argument in (d) does not lead to a contradiction. So that proof does not carry forward to this case.

The algorithm is still correct in this case.

OPTIONAL. One way to prove this is to first directly compute the number of edges e , with any particular weight $w(e)$, in any MST of (V, E, w) , and then to compare this with the output of Prof. Jot's algorithm.

To do this, let $\{w_k\}_{k=1}^K$ be the sorted list of edge weights $\{w(e) \mid e \in E\}$, without repeats. So $w_1 < w_2 < \dots < w_K$ and, since there are some repeats in this case, $K < |E|$.

Two important definitions are as follows. First, for any set of edges F , $F \subseteq E$ and some real number τ , define $F(\tau) = \{e \in F \mid w(e) \leq \tau\}$. With this definition, given any subgraph (V, S) of (V, E) , the number of edges in S with weights no larger than w_k can be denoted by $|S(w_k)|$. Secondly, define $\Gamma(V, S)$ to be the number of maximally connected components in the unweighted sub-graph (V, S) . The following two Claims show the relationship between these two quantities for acyclic sub-graphs (V, S) .

Claim 1: For any acyclic subgraph (V, A) , i.e., with $A \subseteq E$,

$$|A(w_k)| \leq |V| - \Gamma(V, E(w_k)). \quad (1)$$

for each $k = 1, \dots, K$.

The intuition here is similar to that of Kruskal’s algorithm. Namely by sorting the edges by weight, each successive edge in $A(w_k)$ can be considered to connect a pair of components in the forest constructed so far, and each additional edge reduces the number of connected components in the resulting forest by one. The number of such edges must be bounded by the maximum change in the number of connected components. The latter starts at $|V|$, before any edge is added, and must be no larger than $\Gamma(V, E(w_k))$ after considering all edges of weights no larger than w_k . (The proof of Claim 1 is left to the reader.)

We next sharpen Claim 1 to give necessary and sufficient conditions for a spanning tree to be a MST. These conditions will be specified in terms of the quantities $M(k) = |V| - \Gamma(V, E(w_k))$ with $M(0) = 0$, and $N(k) = M(k) - M(k - 1)$, for $1 \leq k \leq K$.

Claim 2: Given the graph (V, E, w) and an acyclic subgraph (V, T) , then T is an MST if and only if $|T(w_k)| = M(k)$ for $1 \leq k \leq K$. The latter condition is also equivalent stating that the number of edges with each particular weight w_k must satisfy $|\{e \in T \mid w(e) = w_k\}| = N(k)$ for $1 \leq k \leq K$.

That is, for any weighted graph (V, E, w) , the above claim provide necessary and sufficient conditions for a spanning tree (V, T) to be an MST, expressed simply as the number of edges in T having each distinct weight w_k . The details of the proof of Claim 2 are left to the reader.

The remainder of the proof goes as follows. Let T_J be the output of Prof. Jot’s algorithm. Then (V, T_J) is a spanning tree (by part (a) above). Let $N_J(k)$ equal the number of edges in T_J with weight equal to w_k . If $N_J(j) = N(j)$ for all $j \in \{1, 2, \dots, K\}$ then it follows from the claims above that T_J is a MST. Otherwise there must be a smallest index, say k , at which $N_J(k) \neq N(k)$. However, it follows from Claim 1 that $N(k)$ is actually an upper-bound for $N_J(k)$, that is, $N_J(k) < N(k)$.

It can now be argued that there must exist an MST T such that T and T_J agree on all the edges with weights $w(e) < w_k$, and the MST T includes all the $N_J(k)$ edges with weights $w(e) = w_k$ that appear in T_J . But this means there must be another edge e in this MST T with weight $w(e) = w_k$ but which is not also in T_J .

For this edge e it follows that there must be an edge f in T_J with $w(f) > w(e)$. (Specifically, the cycle formed by adding $e = (u, v)$ to T_J cannot be entirely across edges with weights less than or equal to w_k , since all these edges are in T and T is acyclic.) Finally, this arrangement of e and f contradicts the fact that the algorithm stopped with T_J .

Note that although Prof. Jot’s algorithm is correct, it is much too slow in practice. However the correctness of Prof. Jot’s algorithm says something quite powerful about the MST problem. Namely:

- if we consider only the simplest possible moves from one spanning tree of G to another (i.e., the simple discrete exchange of a suitable pair of edges),
- if we start with any spanning tree, and
- if we consistently move in these discrete, simple steps towards lower cost trees,

then we converge to a MST. In Prof. Jot’s words, “Finding an MST is about as conceptually hard as finding the bottom of a hill while riding a toboggan.” (Although, he probably never rode a toboggan down a steep hill.) This is similar to saying that a continuous optimization problem is convex and therefore a simple descent procedure will not get stuck in a “local minimum”. We will have more to say about local minima later in this course.

2. Consider question 4 from a 2006 final examination:

http://www.cs.toronto.edu/~jepson/csc373/tutNotes/Q4_dynProg_csc373h-d06.pdf.

Note there are some typos in that question. You should replace all instances of the word “week” with “month”. That is, you are to choose exactly one project per month. Also, assume a different internet project is available for you to choose each month.

- (a) Use dynamic programming to solve for the maximum possible profit at the end of each month. Specifically, construct a $2 \times (n + 1)$ array P where $P(k, j)$ stores maximum possible profit at the end month $j \geq 0$, given that you are working in city $k \in \{0, 1\}$ during month $j = 1, \dots, n$. Assume $k = 0$ denotes Toronto and $= 1$ Vancouver. Assume travel is done at the beginning of a new month. Since you have to start in Toronto, take $P(0, 0) = 0$ and $P(1, 0) = -\infty$. Explain how the remaining entries in the table P can be iteratively computed.
- (b) Explain how to compute an optimal travel schedule and job selection given the table P of optimal profits.
- (c) How would you change (a) above if you were allowed to only do at most one internet project over all n months of work?

Solution for Q2:

2a Let $k = 0$ denote Vancouver and $k = 1$ Toronto.

We will build table $P(k, j)$ to consist of the maximum possible profit given we work only during the first j months, and end up working in city k on month j .

Let $T(k, j)$ denote the table of profits from individual jobs given in the question. And let C denote the cost of travel (assumed the same each way).

Then it follows that for $j > 0$

$$P(k, j) = \max\{P(k, j - 1) + T(k, j), \\ P(k, j - 1) + I(j), \\ P((\text{not } k), j - 1) - C + T(k, j)\} \quad (2)$$

This table $P(k, j)$ can be computed by iterating $j = 1, 2, \dots, n$ using the initial conditions $P(0, 1) = 0$ and $P(1, 0) = -\infty$, since we start in Toronto.

In an exam I would expect a short description of what exactly the three choices in (2) represent. For example, the first line above represents the choice of working in the current city k for month j and selecting the optimal profit schedule for the first $j - 1$ jobs.

Note that travelling at the beginning of month j and then doing an internet job, for a monthly profit of $-C + I(j)$, is not considered since it will never be selected in the above max (assuming the cost of travel $C > 0$, which is a safe assumption).

2b Starting at $j = n$, set $k(n) = \operatorname{argmax}(P(:, n))$ and set the cumulative earnings $\Phi(n) = P(k(n), n)$. For $j = n, \dots, 1$ find the first row on the right hand side of (2) that gives the same value as $\Phi(j)$. Record the corresponding action for the month j in terms of the three rows, respectively; (1) do the job during month j in the city $k(j)$ without travel at the beginning of the month, so $k(j - 1) = k(j)$; (2) do the internet job, without travel at the beginning of the month, so $k(j - 1) = k(j)$; or (3) do the job in city $k(j)$ along with travelling from the other city at the beginning of the month, so $k(j - 1) = (\text{not } k(j))$. Set $\Phi(j - 1)$ to be $\Phi(j)$ minus the profit for that month, as determined by corresponding row on the right hand side of (2).

2c Double the number of rows in the table. Let $P(n, j)$ with $n = 0, 1$ denoting the state of being in the two cities without having done the internet job by the end of month j , and use $n = 2, 3$ to denote being in the city $k = n - 2$ having done the internet job during or before month j .

The recurrence relation is then, for $k = 0, 1$,

$$\begin{aligned} P(k, j) &= \max\{P(k, j-1) + T(k, j), \\ &\quad P(\text{not } k), j-1) - C + T(k, j)\} \\ P(k+2, j) &= \max\{P(k+2, j-1) + T(k, j), \\ &\quad P(k, j-1) + I(j), \\ &\quad P(\text{not } k) + 2, j-1) - C + T(k, j)\}. \end{aligned}$$

The initial values at $j = 0$ are the same as above, along with $P(2, 0) = P(3, 0) = \infty$.