Exercises

Group 2

13 November 2023

Exercise 1

We have two distinct trees T, T'. Let H be the graph formed by adding the e from T into T'. Since a tree is an edge-minimal acyclical connected graph the only possible outcome of this (in order to not contradict edge-minimality) is that H contains a cycle. Since T is a tree this cycle can not be contained in it, and as such the cycle must contain an edge f that is not in T. Lastly we construct T'' by removing this edge f from H. Prove that T'' is a spanning tree.

Answer:

Since all vertices of H were connected to the vertices of the cycle and removing an edge from a cycle keeps it connected, we have that T'' is connected through the remaining vertices of the cycle. We also have that $E(T'') = E(T') = n - 1$, hence T'' is a tree. T'' contains all vertices of G since we didn't remove any vertices from T' in the construction of T'', therefore T'' is a spanning tree.

Exercises 2

Prove that this really is a tree. You can equivalently think of it as the union of all shortest paths between the root v and some other vertex. (This tree is called a shortest-path tree.)

Answer:

Let $G = (V, E)$ be a positively weighted connected graph with $|V| = n$. Dijkstra's algorithm creates a parent function p such that all vertices in the resulting graph except r is assigned a neigbouring parent vertex. Equivalently one could view this as every vertex except r being assigned the edge from it to its assigned parent, and as such the algorithm adds one edge per vertex except r. Notice that such an edge may not be the same throughout all of the algorithm and that there may be some ambiguity in choice in the choice of it due to paths of equal length. Therefore upon termination we have a subgraph of G, G', such that $E(G') \leq n - 1$. Since the algorithm always adds an incident edge of a vertex in question such that the vertex becomes connected to a path to the root r we have that all vertices of G' are connected to r and as such that all vertices are connected to each other, and hence G' is connected. However we also know that trees are edge-minimal connected graphs and that a graph is a tree if and only if it is connected and has $n - 1$ edges. Hence $E(G') \geq n - 1$, and as such $E(G') = n - 1$. Therefore G' is a connected n-vertex graph of $n - 1$ edges which means its a tree.

Exercise 3

We have shown in an exercise during our previous exercise session that a weighted graph with all weights distinct has a unique minimum spanning tree.

Is the stronger statement that in fact no two spanning trees have the same weight also true?

Answer:

No, it is not true.

For a graph $G = (V, E, w)$, where all weights w are distinct, we have a tree T. If there is a way to modify T into tree T', which has the same weight as that of T, then the statement is false.

As a tree T always has $(|V| - 1)$ edges, every edge added must result in an edge removed and vice versa. And as all weights are distinct no single edge can be switched out in a way that results in a tree with the same weight. So, for weight to be conserved, at least two edges must be removed and switched out for other ones. The condition is that the increase in weight from one edge switch must be counteracted by a decrease in weight from another switch. So, as long as there are unused edges that can facilitate a switch like that, it is indeed possible, which makes the statement false. This can easily be seen in Figure 1

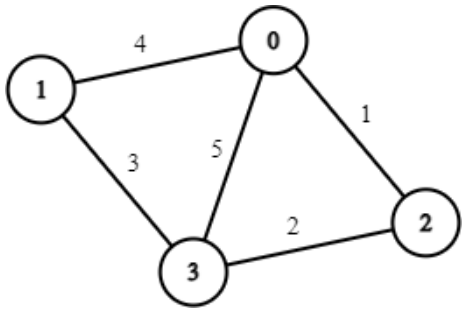

Figure 1: Graph $G = (V, E, w)$, with tree T in red and tree T' in green. Notice that both trees have a weight of 10.

As this can be done for at least some graphs, the statement is proven false.

Exercise 4

One way to find a path between two vertices is to first find a minimum spanning tree for the graph and then find the unique path between them in this tree.

Answer:

1. Will this always be the shortest path between them?

   No, there is no guarantee that this would be the shortest path between this two vertices. To show this, one can look at a specific example as a counterexample. Given a graph shown in the Figure 2 (left):

   - it should be obvious for the reader that Figure 2 (middle) is an MST (If not, one can try to use Kruskal's or Prim's algorithm to find it).
   - Now, given the path to go from C to D via the path in the MST, one would lead to a path with total weight 7.
   - However, in Figure 2 (right), we could have use a shorter path (with weight 5) which the edge that was removed to form the MST.

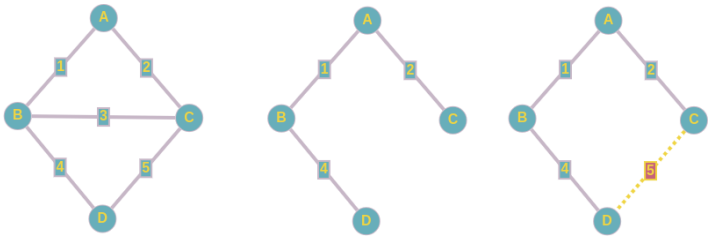

Figure 2: Left: Example weighted graph. Middle: MST. Right: Shorter Path from C to D

2. If not, will there always be *some* pair of vertices such that this is the shortest path between them?

   Yes, there will always be some pair of vertices such that this is the shortest path between them. This is true if edge[1], e_{min} with minimum weight w_{min} always exist in the MST. The shortest path between any two vertices that are connected by an e_{min} should be the path with only e_{min}.

   Since e_{min} has minimum weight, any other shorter path between its vertices that is not this, would imply that either:

   - there is an edge with a lower weight (which contradict with the definition of e_{min}) or
   - another edge with minimum weight connected to the same pairs of vertices (which would just lead to another e_{min} in another MST and also contradict the definition of a simple graph)

[1] Note that you can have more than one edge in a graph with the minimum weight. Also, one can have multiple shortest path between two vertices

Now, one just need to prove that at least one e_{min} would always exist in any MST, then we will have proved our statement. To do this, we will prove by contradiction.

Given a graph, G(V,E) with $w(e_{min}) = w_{min}$ where $e_{min} = \{v_1, v_2\}$. We first consider the existence of path P that connects v_1 to v_2 that is not $P_{min} = v_1 e_{min} v_2$. Here, we would also have $w(P) = w$ and $w > w_{min}$:

(a) if P does not exist, then e_{min} must be in the MST, since its the only way to connect v_1 and v_2.

(b) if P exist, then it implies the existence of a cycle in the graph since there are two ways of going from v_1 to v_2. Hence, there are also two ways of having a spanning tree.

Now, in the scenario where P exist, assume that P_{min} is not in the MST but P is.

(a) First, we add e_{min} into the MST to form a graph with cycle.

(b) Now if we remove the maximal edge in the cycle , the removed edge can't be e_{min}.

(c) This would again form a tree but with the weight of the tree less than the weight of the MST, contradicting the definition of an MST.

Hence, e_{min} must always exist in an MST and thus, there will always be some pair of vertices such that the path in the MST is the shortest path between them.

Exercise 5

Are any of the things we did in this lecture useful for computing the diameter of a graph? Think about how one might do it, and what runtimes your suggested algorithms will have.

Answers:

It's pretty useful to know how to compute the distance between any two nodes in a weighted graph if you want to find the maximal distance.

The most straightforward way to compute the diameter is just determining the distance function $d(v, \cdot), \forall v \in V$. This can be done via an algorithm which takes the current maximum distance D as an argument . Start algorithm with $D = 0$. Then for every vertex $v \in V$, use Dijkstras algorithm to find $d(v, \cdot)$. If $\max(d(v, \cdot)) > D$, then set $D = \max(d(v, \cdot))$. When you have iterated through all vertices, return D. This will obviously give the diameter. Dijkstras algorithm has runtime $\mathcal{O}(|E| + |V| \log(|V|))$. So this algorithm has runtime $\mathcal{O}(|V|(|E| + |V| \log(|V|)))$.

One could definitely utilise the fact that intialising Dijkstras algorithm from a node, one also gets the distance between other nodes "for free". Not only is $d(v, v') = d(v', v)$, but also the distances between any intermediate nodes from v' to v is acquired. However, it seems somewhat complicated to determine a Θ complexity for such an algorithm with our current knowledge.

Searching the internet for a more effective algorithm I found the Floyd-Warshall algorithm which has a time complexity of $\mathcal{O}(|V|^3)$. This algorithm uses a modified weighted adjacency matrix which finds the distance between all nodes simultaneously by considering each node as an intermediate node. For example, we might know that there is a path of length d from v to v', now consider if the path via v'' is shorter: $d(v, v'') + d(v'', v') < d(v, v')$? The complexity of the Floyd-Warshall algorithm is preferred if the amount of edges greatly exceeds the amount of vertices, meaning that the graph is dense.

Exercise 6

If there are multiple edges of the same weight there may be some element of choice in the ordering of the edges in Prim's and Kruskal's algorithms. Using this show that given an arbitrary MST the algorithms can always find it.

Answers:

Kruskal's Reaches all MST:s

Let T be an MST of $G = (V, E)$. We are going to construct a weight-ordered sequence, L, of all edges in E as input to Kruskal's algorithm. This sequence will be constructed in such a manner that for any weight $w \in W$ we have that all edges in $E(T)$ of weight w appear first in the list followed by any residual edges in $E \setminus E(T)$ also of weight w.

We will begin by letting W be a $\leq$-ordered sequence of all the weights which appear in G. Then for $w \in W$ let L_w^T be a sequence which orders the elements of $\{e \in E(T) \mid w(e) = w\}$, i.e. L_w is an ordering of the edges in T with weight w (might be empty if there are no edges in T of weight w), and Let $L_w^{T^c}$ be a sequence which orders $\{e \in E \setminus E(T) \mid w(e) = w\}$.

(Below we will use $\oplus$ for sequences and this will be interpreted as $(a_1, \ldots, a_m) \oplus (b_1, \ldots, b_n) = (a_1, \ldots, a_m, b_1, \ldots, b_n)$, and for an ordered sequence $I = (i_1, \ldots, i_k)$ we will interpret $\bigoplus_{i \in I} A_i$ as $A_{i_1} \oplus \ldots \oplus A_{i_k}$.)

Now let $L_w = L_w^T \oplus L_w^{T^c}$ and finally:

$$L = \bigoplus_{w \in W} L_w$$

We now want to show that Kruskal's algorithm indeed returns T upon termination when given L as input.

We will suppose for contradiction that there is some first timestep i at which Kruskal's deviates from T. This deviation could occur in two different ways:

- The algorithm adds an edge e_i which is not in T.
- The algorithm does not add e_i even though it is in T.

In fact the deviation must be of the former type since the second type would mean that adding e_i creates a cycle with the already added edges all of which are in T. However this is not possible since that would mean there is a cycle in T.

Now if we add e_i to T we create a cycle. This cycle must contain some edge of greater weight than e_i since all edges in T of weight $\leq w(e_i)$ have been added before e_i by the algorithm. This is since

e_i is in $L_w(e_i)^{T^c}$ which comes after both $L^T_{w(e_i)}$ and all the edges with weight strictly less than $w(e_i)$ in L. Now if we remove this edge from the cycle we get a connected graph of $n-1$ edges. Hence this graph is a tree. Call it T'. We now have that $w(T') < w(T)$ which contradicts the fact that T is an MST. Hence the algorithm does not deviate at any point, and as such the algorithm produces T.

Prims Algorithm Reaches all MST:s

Let $G = V, E$ be a weighted graph with weight w. Assume there is a MST T which Prims algorithm can't find. Define w' by removing a small amount ϵ from the weight of each edge in T such that $0 < \epsilon < m$ where

$$m = \min_{e_i, e_j \in E} |w(e_i) - w(e_j)|$$

This is now the unique MST of G with weights w'. Prims algorithm can find this MST due to it being unique. In every step of the algorithm we are going to consider adding edges between the vertices $V(T)$ of our yet generated tree and $V(T)^C$. We want to add the minimal edge so there is going to be an ordering such that all $w_i - \epsilon$ comes before all w_i but after $w_j < w_i$ due to our choice of ϵ. However, this means that the ordering is still proper with our original weight w. This means that the run on $\{G, w'\}$ to find the MST T is a valid run of Prims algorithm on $\{G, W\}$. So Prims algorithm can find T .