DSC 40B - Super Homework
Due: Wed (March 18) 11:59pm

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 p.m.

Note: slip days *are* permitted for the Super Homework.

Problem 1.

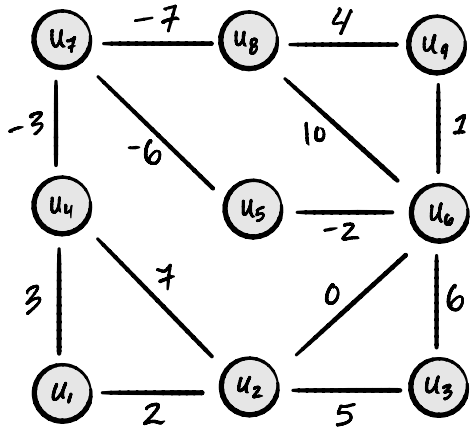
For a given graph G with positive weights containing a node v_0, define $T_{\text{shortest}}(G, v_0)$ as the shortest path tree starting from v_0 (as given by Dijkstra's algorithm), and $T_{\text{MST}}(G, v_0)$ as the minimum spanning tree given by Prim's algorithm starting at v_0.

Find a small example graph G *with at most 4 nodes* where $T_{\text{shortest}}(G, v_0) \neq T_{\text{MST}}(G, v_0)$.

Problem 2.

In this problem you will be asked to list the edges in the minimum spanning tree of the graph below in the order that they are added by either Prim's algorithm or Kruskal's algorithm.

In order to simplify grading, please write an edge with the *smaller* node first. For example: (u_3, u_7) instead of (u_7, u_3). Also, when writing an edge, make sure to write the edge as a pair of nodes, not the weight of the edge. Thanks!

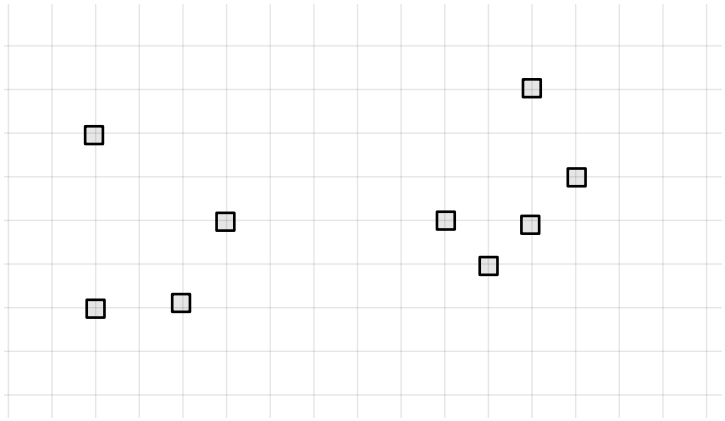


a) Suppose Prim's algorithm is run on the above graph, using node u_8 as the starting node. List the edges of the resulting minimum spanning tree computed in the order that they are added by the algorithm.

b) Suppose Kruskal's algorithm is run on the graph above. List the edges of the resulting minimum spanning tree in the order that they are added by the algorithm.

Problem 3.

The picture below shows a set of points in 2-dimensional space. A grid is provided so that you can compute the distance between points; each grid cell is 1 unit wide and 1 unit tall. You may assume that each data point is placed on a grid intersection.



Suppose a weighted distance graph G is constructed from this data set (recall that a distance graph is a complete graph whose nodes represent points in space, and whose edges are weighted by the distance between its endpoints). Then suppose that a minimum spanning tree is computed for G. What will be the weight of the largest edge in this minimum spanning tree?

Problem 4.

Suppose we are given a undirected weighted graph $G = (V, E; w)$, where $w : E \to \mathbb{R}$ is the map to assign a weight $w(e)$ to each edge$e \in E$. Let T^* be a minimum spanning tree of G.

a) Now let $G_1 = (V, E; w_1)$ be a modified version of G: In particular, G_1 shares the same vertex set and edge set as G. The only difference lies in the edge weights, where for any edge $e \in E$, the new weight doubles the previous weight $w(e)$; that is, $w_1(e) = 2w(e)$.

(True or False): The tree T^* is still a minimum spanning tree for G_1 as well. You do not need to provide a justification.

b) Now let $G_2 = (V, E; w_2)$ be another modified version of G: In particular, G_2 shares the same vertex set and edge set as G. The only difference lies in the edge weights, where for any edge $e \in E$, the new weight equals the previous weight $w(e) + 2$; that is, $w_2(e) = w(e) + 2$.

(True or False): The tree T^* is still a minimum spanning tree for G_2 as well. You do not need to provide a justification.

c) Let $s \in V$ be a source node. Let $\widehat{T}$ be a shortest path tree of G from the source s. Consider the modified graph G_2 defined above.

(True or False): The tree $\widehat{T}$ is a shortest path tree for G_2 from the source s as well. Briefly justify your answer.

Problem 5.

(True or False) Consider the following statements, and in each case, decide whether it is true or false. You do not need to justify your answer.

(a) Let G be an undirected weighted graph. Let e be an edge of G with largest edge weights. Then it must be that e cannot be in any minimum spanning tree of G.

(b) Let G be an undirected weighted graph. Let e be an edge of G with largest edge weights. Then it must be that e cannot be in any shortest path tree of G starting at a source node s.

(c) Let G be an undirected weighted graph. Let e be an edge of G with smallest edge weights. Then it must be that given any minimum spanning tree of G, it has to include e.

(d) Let G be an undirected weighted graph. Let e be the unique edge of G with smallest edge weight – in other words, there is only one edge of G with smallest edge weight. Then it must be that any shortest path tree of G starting at a source node s must include e.

Problem 6.

Describe an algorithm that does the following: Given any undirected weighted graph $G = (V, E; w)$, where $w : E \to \mathbb{R}$ is the map that assigns a weight $w(e)$ to each edge $e \in E$, and given a real value $\rho \in \mathbb{R}$, your algorithm aims to compute a spanning tree T^* with smallest total weights such that any edge in your spanning tree needs to have weight at least ρ. If no such tree exists, your algorithm should detect that.

Describe your algorithm in 8 sentences or less, and you can use any existing code from class. You also need to provide the asymptotic time complexity of your algorithm in terms of $|V|$ and $|E|$. Slower but correct algorithm receives fewer points.

Programming Problem 1.

In lecture, we saw that Kruskal's algorithm can be used to cluster a weighted graph. The name for this approach is *single linkage clustering*.

In a file named `slc.py`, write a function `slc(graph, d, k)` which accepts the following arguments:

- `graph`: An instance of `dsc40graph.UndirectedGraph`.
- `d`: A function of one argument: a tuple containing a pair of nodes. It returns the distance (or dissimilarity) between them. See the example below.
- `k`: A positive integer describing the number of clusters which should be found.

The function should perform single linkage clustering using Kruskal's algorithm and it should return a `frozenset` of k `frozenset`s, each representing a cluster of the graph.

Example:

```
>>> g = dsc40graph.UndirectedGraph()
>>> edges = [('a', 'b'), ('a', 'c'), ('c', 'd'), ('b', 'd')]
>>> for edge in edges: g.add_edge(*edge)
>>> def d(edge):
...     u, v = sorted(edge)
...     return {
...         ('a', 'b'): 1,
...         ('a', 'c'): 4,
...         ('b', 'd'): 3,
...         ('c', 'd'): 2,
...     }[(u, v)]
>>> slc(g, d, 2)
frozenset({frozenset({'a', 'b'}), frozenset({'c', 'd'})})
```

Note: to implement Kruskal's algorithm, you'll need an implementation of a Disjoint Set Forest data structure. We've uploaded a simple one here: `https://gist.github.com/eldridgejm/983d6ce03a82bf295599e9880ef02bab`

You can copy and paste this into `slc.py`, or put it in a separate file that is imported; if you do this, make sure to upload that file alongside `slc.py`.