# DSC 40B - Super Homework

Due: Wednesday, December 11

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.

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!
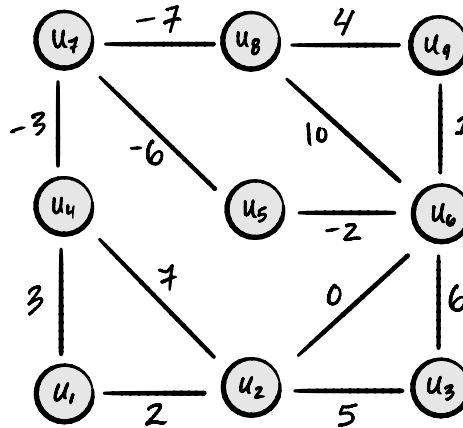
a) Suppose Prim's algorithm is run on the above graph, using node u_1 as the starting node. List the edges of the resulting minimum spanning tree computed in the order that they are added by the algorithm.

> **Solution:** $(u_1, u_2), (u_1, u_4), (u_2, u_3), (u_2, u_6), (u_5, u_7), (u_5, u_6), (u_6, u_9), (u_7, u_8)$

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.

> **Solution:** $(u_1, u_2), (u_2, u_3), (u_2, u_6), (u_4, u_7)(u_5, u_6), (u_5, u_7), (u_6, u_9), (u_7, u_8)$

Problem 2.

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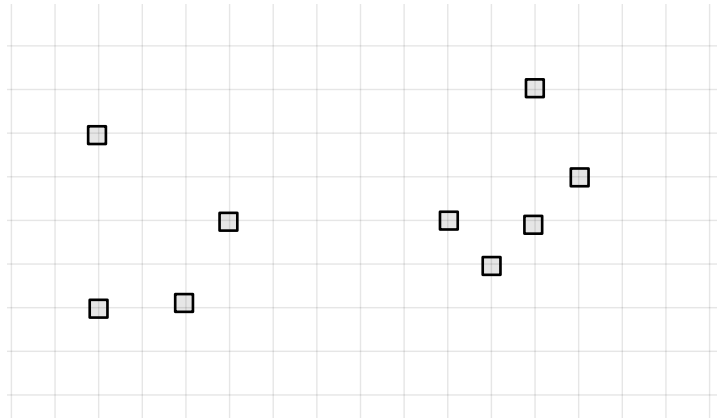
Suppose a weighted distance graph G is constructed from this data set (recall that a distance graph is a complete graph whose nodes represent points in space, and whose edges are weighted by the distance between its endpoints). Then suppose that a minimum spanning tree is computed for G. What will be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

Solution: The weight of the largest edge in the MST will be 5 units. First, we recognize that the graph is a complete graph, meaning every node is connected to every other node by an edge weighted by the Euclidean distance between them. To construct the MST from this complete, distance-weighted graph, we use an algorithm like Prim's Algorithm. This algorithm works by adding the smallest edge that connects a new node to the existing MST, ensuring all nodes are connected with the minimum possible total weight.

In this case, the points in the graph naturally form two clusters. Prim's Algorithm would connect the nodes within each cluster using the smallest available edges. These edge weights would not be the largest edge weights because in both clusters, the largest edge added to the MST would have a weight less than 4. The smallest edge that connects the two clusters has a weight of 5, and this edge is added to the MST to make the MST connected, which is a key requirement. Since all other edges within the clusters have smaller weights, the edge with weight 5 becomes the largest edge in the MST

MST visualized:

$\in$
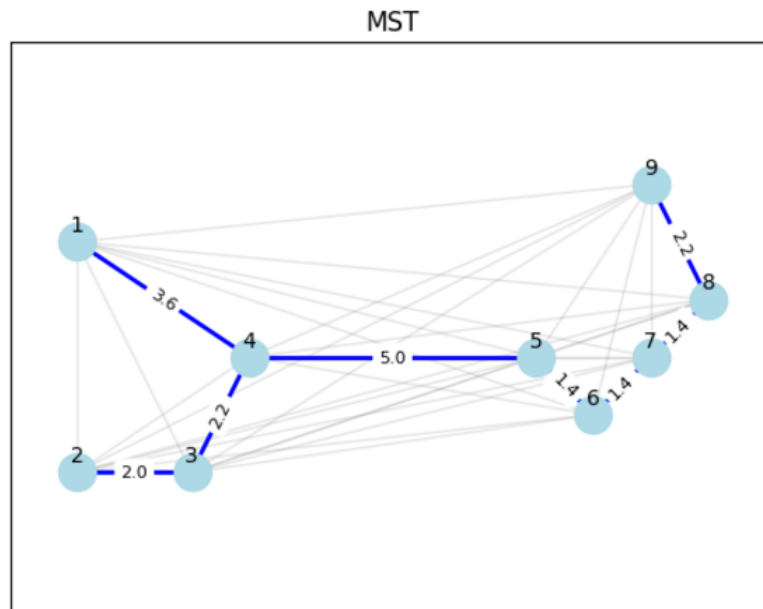
Problem 3.

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 share 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.

> **Solution:** True

b) Now let $G_2 = (V, E; w_2)$ be another modified version of G: In particular, G_1 share 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.

> **Solution:** True

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.

> **Solution:** False, the tree $\widehat{T}$ is not guaranteed to be a shortest path tree for G_2 from the source s. This is because adding a constant to all edge weights can change the shortest paths because it affects paths differently based on how many nodes are in the path. For example, take the nodes s, a, b and their original edges and weights:
>
> - $s \to a$ with $w(s, a) = 1$
> - $a \to b$ with $w(a, b) = 1$
> - $s \to b$ with $w(s, b) = 2$
>
> The possible paths in G_1 are :
>
> - $s \to a \to b$ weight $= 1 + 1 = 2$
> - $s \to b$ weight $= 2$
> - Both possible paths have equal weights, so the shortest path tree $\widehat{T}$ might include either path.
>
> However, in G_2 the weights are:
>
> - $w_2(s, a) = 1 + 2 = 3$
> - $w_2(a, b) = 1 + 2 = 3$
> - $w_2(s, b) = 2 + 2 = 4$
> - $s \to a \to b$ with total weight $3 + 3 = 6$

– $s \to b$ with total weight 4

   – Now, the direct path $s \to b$ is the shortest path and adding $s \to a \to b$ to $\widehat{T}$ would not result in a MST for G_2 even though it did result in a MST For G$_1$

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`.

**Solution:**

Programming Problem 2.

Let `data` be a list of n unique real numbers. Furthermore, suppose that each number in `data` is assigned a color – it is either `'red'` or `'blue'`. Let `colors` be a list of n strings, such that `colors[i]` gives the color of `data[i]`. In all parts of this problem, you may assume for simplicity that there is at least one data point of each color.

a) To begin, suppose that **all of the blue points are less than all of the red points**. In a file named `min_ell_theta.py`, write an efficient function called `learn_theta(data, colors)` which takes in two arguments – the lists `data` and `colors` as described above – and returns a single number θ such that all of the blue points are $\leq \theta$ and all of the red points are $> \theta$, as is depicted in the picture below. You may *not* assume that `data` is sorted. The time complexity of your algorithm should be optimal.
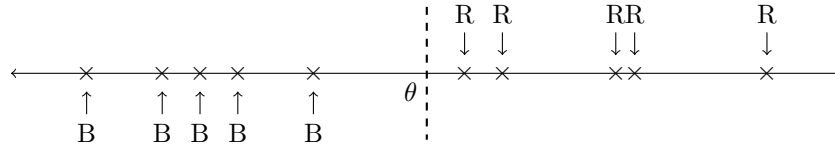
Solution:

Now suppose that a small number of the red points are less than some blue points – that is, there is some overlap, as shown below. Assume for simplicity that the largest data point is red.
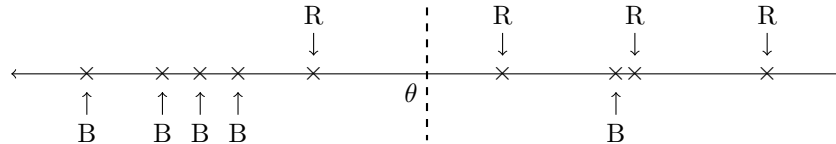
We wish to find a real number θ which "best" separates the blue points and red points. Clearly the points cannot be separated perfectly. Instead, we define a loss function $L(\theta)$ which counts the number of points which are on the wrong side of θ. More precisely:

$$L(\theta) = (\# \text{ of red points } \leq \theta) + (\# \text{ of blue points } > \theta)$$

The loss of the θ shown above is 2, since one red point is to the left of θ and one blue point is to the right. Our goal is to design an algorithm for finding a minimizer of $L(\theta)$. This is a simple instance of the machine learning task of *classification*.

b) Also in `min_ell_theta.py`, write a function named `compute_ell(data, colors, theta)` which takes in lists `data` and `colors` as described above, as well as a floating-point number, `theta`. It should return the loss at `theta` as a floating-point number. Your algorithm should have the best possible time complexity.

Solution:

c) Also in `min_ell_theta.py`, write a function named `minimize_ell(data, colors)` which takes in `data` and `colors` and returns a floating-point number which minimizes the loss L for that particular data set. Your algorithm should have quadratic time complexity. You may assume for simplicity that the smallest data point is blue[1].

Solution:

d) Now assume that `data` is sorted (and `colors[i]` is the color of `data[i]`). In the file called `min_ell_theta.py`, write a function `minimize_ell_sorted(data, colors)` which returns a minimizer θ in linear time. Your code should satisfy the loop invariant: "After the αth iteration, `blue_gt_theta` is the number of blue points which are greater than `data[`$\alpha - 1$`]`."

For simplicity, suppose that exactly $n/2$ of the data points are `'red'`, and $n/2$ are `'blue'`.

Solution:

---

[1]Otherwise it is possible (given a special data set) for the loss to be minimized at some θ to the left of all of the data.