Mitchell Dennen
Shahin Safarov
Sowmya Kodityala

Assignment 2

Prim's algorithm for MCSTs grows a tree in a natural way, starting from an arbitrary root; at each stage it adds a new branch to the already constructed tree. The algorithm stops when all nodes have been reached:

Algorithm 1 Prim

```
1: T ⟵ ∅
2: B ⟵ {v : an arbitrary vertex of V}
3: while B ≠ V do
4:         Find cheapest e = (u1, u2) such that u1 ∈ B, u2 ∈ V − B
5:         T ⟵ T ∪ {e}
6:         B ⟵ B ∪ {u2}
7: end while
```

1. Compare what happens in Kruskal's algorithm versus Prim's algorithm if we run them on a graph that is not connected.

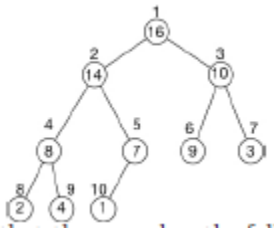
As Kruskal's algorithm initiates with an edge, it selects the edges in a way that the position of the edge is not based on the last step and also kruskal's algorithm is based on edges of the graph. Therefore, kruskal's algorithm works on disconnected graphs and finds the minimum spanning tree for the connected portion of the graph. Whereas, Prim's algorithm starts with a vertex, it starts with one vertex and keeps on adding edges with the least weight till all the vertices are covered. So if the graph is disconnected, it would not be able to traverse all the vertices.

2. Adapt Prim's algorithm to graphs that may include edges of negative costs; give an example of an application where negative costs may occur naturally.

If in the case of finance, negative edges mean spending and positive edges mean gaining, an adjustment to prim's algorithm would simply by finding a maximum spanning tree or priciest path instead of minimum spanning tree or cheapest path. In the case of statistics for looking at a bell curve, a modification could be finding the minimum spanning tree of the absolute value of edge cost.
A really common example of naturally occurring negative weight graphs are in finance.
If you were to make a graph tracking business deals between companies, any amount of money paid to another company would manifest as a negative weight on the graph.

3. A binary heap data structure is an array that we can view naturally as a nearly complete
binary tree. Each node of the tree corresponds to an element in the array, as shown

below:



16	14	10	8	7	9	3	2	4	1

note that the array has the following interesting structure: the parent of i is ⌊i/2⌋ and the left child of i is 2i and the right child of i is 2i + 1.
With the binary heap data structure we can implement line 4. efficiently, that is, find the cheapest e. To this end, implement a min-priority queue B, which, during the execution of Prim's algorithm, keeps track of all the vertices that are not in the tree T. The min-priority queue B uses the following key attribute: minimum weight of any edge connecting the vertex to T (and ∞ if no such edge exists).
Describe the details of the above scheme, and implement it in Python 3.

(problem submitted in separate file)