Homework Assignment #4
Due: June 17, 2020 at 12:00 noon

1. In class, we discussed how various priority queue implementations can be used to implement Prim's algorithm for finding minimum spanning trees (MST).

Now we consider a different greedy MST algorithm. Its input is a connected graph $G = (V, E)$ with a positive edge weight $w(e)$ for each edge e. We assume the vertices of the graph are labelled $1..n$. Assume the input graph is provided in an adjacency list representation (see Section 22.1 of the textbook).

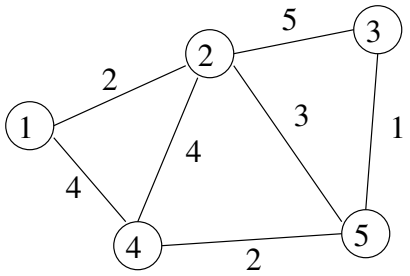
```
1   MST(G)
2       T ← {} // set that will store the edges of the MST
3       for i ← 1..n
4           V_i ← {i}
5           E_i ← {(i, j) : j is a vertex and (i, j) is an edge of G} // set of all edges incident with vertex i
6       end for
7       while there is more than one set V_i
8           choose any V_i
9           extract minimum weight edge (u, v) from E_i
10          one of the endpoints u of this edge is in V_i; let V_j be the set that contains the other endpoint v
11          if i ≠ j then
12              T ← T ∪ {(u, v)}
13              combine V_i and V_j into V_i (destroying V_j)
14              combine E_i and E_j into E_i (destroying E_j)
15          end if
16      end while
17      return T
18  end MST
```

Your goal is to implement this algorithm using Fibonacci heaps to store each of the sets V_i and E_i.

(a) Explain in detail how Fibonacci heaps could store the information needed by the algorithm. Would you have to add any additional fields to nodes in the heaps or use any additional data structures?

(b) Explain how you would use your data structure in part (a) to implement line 5, 10, 11, 13 and 14. In order to achieve good efficiency, when would you run the CONSOLIDATE routine?

(c) Draw a picture of the state of all your data structures after executing two iterations of the while loop in the algorithm on the following input graph. Assume that in each of those two iterations, you chose the V_i that contains vertex 1.



(d) If the input graph has n nodes and m edges, give a good bound on the worst-case running time of the algorithm using your data structure.

Remark: Let's check that this algorithm does indeed produce a MST. The while loop satisfies the following loop invariant.

1. The V_i sets form a partition of the vertices of G. (I.e., each vertex of G is in exactly one V_i.)
2. At least one endpoint of each edge in E_i is in V_i. Moreover, E_i includes every edge that connects a vertex in V_i to a vertex that is not in V_i.
3. Two vertices are in the same set V_i if and only if there is a path connecting them that uses only edges of T.
4. There exists a MST T^* such that $T \subseteq T^* \subseteq T \cup \bigcup_i E_i$

The first three are trivial to prove. We'll prove the fourth one. We use T_k to denote the value of the set T after k iterations of the while loop.
Base case: After 0 iterations, T_0 is empty and the E_i's contain all the edges of G. So any MST of G satisfies the invariant.
Induction step: Let $k > 0$. Suppose there is a MST T^* that satisfies the claim for the set T_{k-1}. We must prove there is a MST $\hat{T}$ that satisfies the claim for the set T_k. We consider several cases.

- Suppose the test on line 11 is true and the edge (u, v) chosen on line 9 is in T^*. Then $\hat{T} = T^*$ satisfies the claim for T_k.
- Suppose the test on line 11 is true and the edge (u, v) chosen on line 9 is not in T^*. Since T^* is a MST, there is a path from u to v in T^*. Let (x, y) be the first edge along this path such that $x \in V_i$ and $y \notin V_i$. Such an edge must exist, since $u \in V_i$ and $v \notin V_i$. By invariant (2), $(x, y) \in E_i$. Since (u, v) was a minimum weight edge in E_i, we have $w(u, v) \leq w(x, y)$. Let $\hat{T} = T^* \cup \{(u, v)\} - \{(x, y)\}$. Note that $\hat{T}$ is still a spanning tree: there is still a path from x to y in $\hat{T}$ since $\hat{T}$ contains a path from x to u, the edge (u, v), and a path from v to y. Moreover, the swap we made to change T^* to $\hat{T}$ cannot increase the total weight of the spanning tree. Thus $\hat{T}$ is still a MST.

  By the induction hypothesis, $T_{k-1} \subseteq T^*$. Moreover, by invariant (2), $(x, y) \notin T_{k-1}$. So, $T_{k-1} \subseteq T^* - \{(x, y)\}$. Hence, $T_k = T_{k-1} \cup \{(u, v)\} \subseteq T^* \cup \{(u, v)\} - \{(x, y)\} = \hat{T}$. Finally, we show that $\hat{T} \subseteq T_k \cup \bigcup_i E_i$. Consider any edge (y, z) in $\hat{T}$. If $(y, z) = (u, v)$ then $(y, z) \in T_k$. If $(y, z) \neq (u, v)$, then $(y, z) \in T^*$. By the induction hypothesis, either (y, z) was in T_{k-1} and therefore also in T_k, or (y, z) was in one of the E_i sets at the beginning of this iteration of the loop and it is still in E_i at the end of the iteration, since $(y, z) \neq (u, v)$.
- Suppose the test on line 11 is false. Then u and v lie in the same set V_i. By invariant (3), there is a path from u to v in T_{k-1}. If $(u, v) \in T_{k-1}$ then $\hat{T} = T^*$ satisfies the claim for T_k. If $(u, v) \notin T_{k-1}$ then (u, v) cannot be in T^*, since then T^* would contain a cycle. So, $\hat{T} = T^*$ satisfies the claim for T_k in this case too.

This completes the proof of invariant (4).

To see that the loop terminates, notice that the total size of all the E_i sets decreases by 1 in each iteration, so eventually all the E_i sets will become empty (if the algorithm does not terminate). When that happens, all vertices of G are in a single set V_i, by invariant (2) and the fact that the input graph is connected. So, the while loop will terminate.

When the loop terminates, all vertices are in a single set V_i. By invariant (3), T is a spanning tree of G. Moreover, it is a subset of a minimum spanning tree T^* by invariant (4). So $T = T^*$ and the set returned by the algorithm is a MST.