Junhao Zhang “Freddie”

USC ID:

CSCI 570 – HW2

07/13/2021

jzhang49@usc.edu

**1. Graded Problems**

* **[Question 1]**

Sort set *A* and *B* in the decreasing order, and then let be the *i-th* element of set *A* so that be the greatest integer in the set *A*, and let be the *i-th* element of set *B* so that be the greatest integer in the set *B.* Then can receive the payoff on . This algorithm has runtime complexity since we need to sort the sets. However, if these two sets are sorted already, then then runtime complexity can be .

*Proof.*

Let’s prove its correctness by showing that if there is an optimal solution, then we can transform the optimal solution to our solution by making swaps and making such swaps won’t negatively impact the optimality.

Let and and > for some , , in the optimal solution:

\*

If we make inversion to transform the optimal solution to:

\*

Then we can observe:

, which is greater than 1, which means > .

Now we have proven that each time we make swap to transform the optimal solution to our solution will not have negative impact to the optimality. Thus, we have proven that our solution is also optimal.

* **[Question 2]**

We have a min-heap which has first values of all lists, and after each “pop”, if the list of the popped value is not empty then we push the next value in that list to our min-heap. We repeat *n* times, and we will finish merging these *k* sorted lists into one sorted list if the list saves the popped values in the order of being popped.

Since the heap has length *k*, thus each pop would cost *log* *k*, and we need to do that *n* times. So, the time complexity is *O* (*n log k*).

* **[Question 3]**

Let us denote the edge *e* has the minimal weight. We will prove by contradiction; thus, we first assume that there exists a minimal spanning tree *T* that doesn’t have the edge *e*. Because *T* doesn’t have edge *e*, thus the nodes that were connected by edge *e* must be connected by one or more other edges. However, since *e* is the edge that has the minimal weight and no edges have the same weight, if all weights are positive, then there doesn’t exist an edge or a path that can connect the two nodes with less weights than *e*’s. If the edge or the path that connects the two nodes is not minimal weight, then *T* is not a minimal spanning tree, but we already assumed that *T* is a minimal spanning tree. Contradiction!

* **[Question 4]**

First, all these spanning trees they must have the same number of edges, and let’s denote *N* be the number. The reason is, less than *N* edges are not enough to keep the graph connected, and more than *N* edges will cause cycles.

If we add one new edge to the minimal spanning tree, then there will be a cycle, but after we removed an edge of the cycle that is not the one that we just added, then a new spanning tree will be formed. Among all the spanning trees that can be formed this way, the one with the minimal total weight is the second-best minimal spanning tree.

* **[Question 5]**

Let the cow with the strongest strength take the most weight, which means the stronger strength a cow has, the lower position it takes in the cow tower.

*Proof.*

We are going to prove it by showing if there exists an optimal solution, then we can transform the optimal solution to our solution by swapping positions of cows without making negative impacts.

Let *i, j* ∈ {*1, 2, 3, … N*} and *i < j* (which means cow *i* is above the cow *j*). Moreover, let . Then, we can see that cow *i* has risk value and cow *j* has risk value and in this case cow *j* has the maximum risk value. But after swap, the cow *i* has risk value and cow *j* has risk value , and we can see both of these two values are less than the previous maximum risk value .

Thus, we have proven that making swap won’t negatively impact the solution, thus an optimal solution can be swapped to our solution without being negatively impacted, which makes our solution also an optimal solution.

**2. Practice Problems**

* **[Question 1]**

Let’s denote *G* to be the given graph, and we repeat the following: each time, we find a vertex with no incoming edges, and remove it from *G*. We repeat until either *G* is empty, or G is non-empty but has no more such vertices. If *G* is empty, then the original *G* is DAG. But if *G* is still non-empty and all its vertices have incoming edges, then the original *G* is not DAG and has cycle in it. The time complexity will be *O* (*m + n*), because the algorithm needs to iterate all edges in order to visit all vertices.

* **[Question 2]**

We will prove it by contradiction. First, we assume there is an edge *e* such that *e* exists in *G* but doesn’t exist in *T*. Since *T* has all nodes that *G* has, it must also have the two nodes that *e* connects, and because *T* is a DFS tree, thus one of the two nodes must be the ancestor of another. Let’s denote *x* be the ancestor, and *y* be the descendent. Moreover, since *e* connects *x* and *y*, thus the distance between these must be 1. However, there is a contradiction: since *x* is the ancestor of *y* and their distance is 1, thus when running BFS from *x*, *y* must be added to *T* as the direct child of *x*, which means *e* exists in T, but we already assumed *e* doesn’t exist in T. Contradiction! Therefore, we proved *G* cannot contain any edges that do not belong to *T*.

* **[Question 3]**

~~A tree must have~~ *~~n – 1~~* ~~edge, and the problem states that the~~ *~~near-tree~~* ~~can have up to~~ *~~n + 8~~* ~~edges. Therefore, we may need to remove up to 9 edges in order to make the minimum spanning tree.~~

~~Thus, we keep a max-heap with size 9, and when we iterate an edge, we push the edge to the heap and~~

* **[Question 4]**

We will prove it by

* **[Question 5]**

1. We use the coin with the greatest value (i.e.: quarter) to add up until that is going to exceed the amount (*n* cents), then we use the coin with the second greatest value (i.e.: dime) to add up until that is going to exceed the rest value, then we use the coin with the third greatest value and do the same thing. Repeat until is equal to *n* cents or exceeded the amount. If our sum is equal to the amount, then great we have the solution. But if exceeded the amount, then our sum
2. We will prove it by