Assignment 5

Due: February 17, 2022

Submitted by: - Parth Parashar

Your solutions must be typed (preferably typeset in LaTeX) and submitted as a PDF through Canvas at the beginning of class on the day its due.
When asked to provide an algorithm you need to give well formatted pseudocode, a description of how your code solves the problem, and a brief argument of its correctness.

Problem 1: MST The *cut property* makes it possible to build minimum spanning trees greedily, starting from an empty graph and adding one edge at a time. A different approach is to start with the original graph and remove edges greedily, one at a time, until an MST remains. A scheme of this second type can be justified by the following property.

> Pick any cycle in the graph, and let e be the heaviest edge in that cycle. Then there is a minimum spanning tree that does not contain e.

(a) [5 points] Prove this *cycle property*.

Answer: - In this example we can prove the cycle property that if we remove the heaviest edge e from the cycle, then there is a minimum spanning tree that does not contain e.

Let's assume the graph is G and it has cycle C. Furthermore, let's assume that the heaviest edge in the graph is e. As the graph has cycle C and e is a part of that cycle, so even if we remove the edge e from the graph G, it will still be connected and because of the removal of the edge e, it will not have cycle.

So, let's assume e1 is the edge which is holding together the graph despite the removal of the edge e. Now, as per the greedy choice, the tree which we got after removal of edge e will have weight which is less than or at most equal to the tree with edge e.

So, weight (T with e) > weight (T without e but with e1 to maintain the connectedness of tree).

So, this proves the cycle property i.e., if we remove the heaviest edge e from the cycle, then there is a minimum spanning tree that does not contain e.

(b) [5 points] Use the property to justify the following MST algorithm. The input is an undirected graph $G = (V, E)$ with edge weights $\{w_e\}$.

```
1  sort the edges according to their weights
2  for each edge e ∈ E, in decreasing order of w_e
3      if e is part of a cycle of G
4          G = G − e (that is, remove e from G)
```

5 **return** G

Answer: -

We can prove this by contradiction. There exists e ∈ E. Also, there exists e' ∈ E such that e > e'.

Now assume that we remove e' from the cycle. Then, sum of all weights of edges in MST containing e is greater than sum of all weights of edges in MST containing e'.

However, if we have e such that e is greater than e' in the same cycle, our greedy algorithm will remove e instead of e', and thus sum of weights of all edges will always be minimum in our MST.

(c) [5 points] On each iteration, the algorithm must check whether there is a cycle containing a specific edge *e*. Give a linear-time algorithm for this task, and justify its correctness.

Answer: - If we consider an edge e = (u, v) and there is a path from u to v which does not use the edge e, then we can say that there exists a cycle.

We can either do it by exploring (G', u) where G' = G - e, if it returns the vertex v, then there exists a cycle.

Or by using Depth First Search, by starting from one vertex v on edge e and then marking all the visited vertices (popped from the stack), if we again come across a vertex in the stack that is already visited then it is having a cycle.

And DFS has a linear time of O (V + E).

Algorithm: -
Graph G = createcyclicGraph()
MST = findMST(G)
Iterate over Edges
If cyclePresent(G-e, e.u, e.v)
G=G-e
cyclePresent(G)
initial = 0
Destintaion=0
Iterate over edges such that iterator i<G.E\
If G.i = e.u
Initial = 1
If G.i = e.v
Destination = 1
If initial=1 and destination = 1
Return G
Else
cycleNotFound()

(d) [5 points] What is the overall time complexity of this algorithm, in terms of $|E|$? Explain your answer.

Answer: - As per the given algorithm, we need to sort the edges according to their weights. If we use the quicksort algorithm, it takes O (n log n) time to sort n numbers.
If we apply quicksort to sort the edges, it will lead to time complexity of O (|E| log |E|).

Now if we analyse the given loop in the algorithm:
It will run for that number of times which is equal to number of edges, so number of iterations would be |E|. And it will run for each vertex, so the resultant complexity for the loop will be
O (|V| + |E|).

So the resultant complexity can be given as:
T = O (|E| log |E|) + O (|V | + |E|).
As we know that for a tree, number of edges are always one less than the number of vertices,
so,
E = V - 1 ----- (1)
Using this we get the resultant complexity of O ((|V| + |E|) |E|) = O (|E|^2).

Problem 2: Longest-Probe Bound for Hashing Suppose we use an open-addressed hash table (section 11.4 in CLRS) of size m to store $n \leq \frac{m}{2}$ items.

(a) [10 points] Assuming simple uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability is at most 2^{-k} that the ith insertion requires strictly more than k probes.

Answer: - For an unsuccessful search, every probe except for the last accesses an occupied slot that does not contain the desired key and the last slot probed is empty.

Let us consider a random variable Y_K to be the number of probes that was made in an unsuccessful search

Let E_K be an event such that there is a kth probe and it is to an occupied slot (k=1,2,3..)

This means that the event $\{Y_K > K\}$ is the intersection of events E1.E2.E3,E4.....E_k .

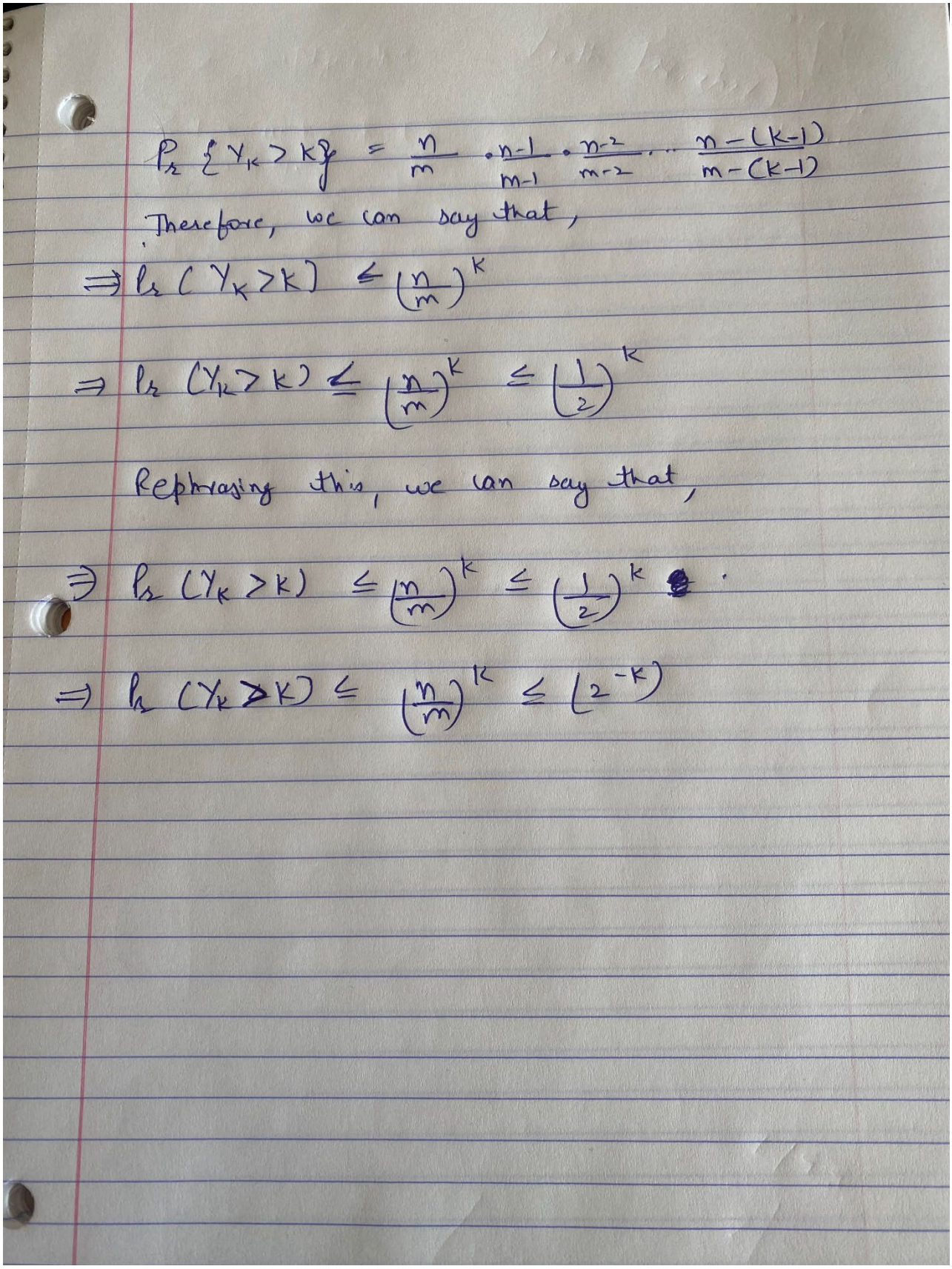
We will bound $\Pr\{Y_K > k\}$ by bounding $\Pr\{E1 \cap E2 \cap E3 \ldots E_k\} = \Pr\{E1\}. \Pr\{E2/E1\}. \Pr\{Ek/E1 \cap E2 \ldots E_K\}$.

Since it is given that there are m slots and n elements $\Pr\{E1\}/m$. For i>1, the probability that there is ith probe and it is to an occupied slot, given that first i-1 probes are to occupied slots is (n-i+1)/(m-i+1)

This probability follows because we would be finding one of the remaining (n-(i-1)) elements in one of the (m-(i-1)) non-examined slots

In uniform hashing we assume that the probability is the ration of these quantities.

Therefore, we can say that n<= n/m for all i such that 0<=i<k



(b) [10 points] Assuming simple uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability

n

is $O(\_^{1})$ that the ith insertion requires more than $2 \lg n$ probes.

Answer: - As we know from the question 2(a), the probability is at most 2^(-k) for the ith insertion requiring strictly more than k probes.
This value could be placed for the specific cases.

The probability that the insertion requires more than 2lgn probes is at most 1/n^2.

$$P_r\{Y_k > k\} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-(k-1)}{m-(k-1)}$$

$$P_r\{Y_{2\lg n} > k\} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-(2\lg n-1)}{m-(2\lg n-1)}$$

From here, we can say that,

$$P_r\{Y_{2\lg n} > k\} \leq \left(\frac{n}{m}\right)^{2\lg n}$$

$$P_r\{Y_{2\lg n} > k\} \leq \left(\frac{1}{2}\right)^{2\lg n}$$

$$= 2^{-2\lg n}$$

$$= \frac{1}{n^2}$$