CS648A : Randomized Algorithms
Semester II, 2022-23, CSE, IIT Kanpur

Theoretical Assignment 1

Deadline : 11:55 PM, 6th February 2023.

Most Important guidelines

- It is only through the assignments that one learns the most about the algorithms and data structures. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. Remember - **Before cheating the instructor, you are cheating yourself**. The onus of learning from a course lies first on you. So act wisely while working on this assignment.
- Refrain from collaborating with the students of other groups. If any evidence is found that confirms copying, the penalty will be very harsh. Refer to the website at the link: https://cse.iitk.ac.in/pages/AntiCheatingPolicy.html regarding the departmental policy on cheating.

General guidelines

1. This assignment is to be done in groups of 2 students. You have to form groups on your own. You are strongly advised not to work alone.
2. The assignment consists of 3 problems. Each problem carries 50 marks. There is an additional but optional problem given at the end whose purpose is to prepare you for Quiz on 4th February.
3. Make sincere attempt solve the problems. You are welcome to meet me during office hours if you get stuck and want small hints.
4. **Naming the file**:
   The submission file has to be given a name that reflects the information about the type of the assignment, the number of the assignment, and the roll numbers of the 2 students of the group. If you are submitting the solution of Theoretical Assignment x, you should name the file as **Theor_x_Rollnumber1_Rollnumber2.pdf**.
5. **Each student of a group** has to upload the same submission file separately. Be careful during the submission of an assignment. Once submitted, it can not be re-submitted.
6. Deadline is strict. Make sure you upload the assignment well in time to avoid last minute rush.

1 Randomized quick select

Let S be a set of n real numbers. Consider the randomized algorithm *Rand-QSelect*(k, S) described below that finds the k^{th} smallest element from the set S.

Select a pivot element x uniformly randomly from set S.
Find its rank in the set S (by comparing x with every other element of set S). Let r be the rank of x.
If $r = k$, we report x as the output. Otherwise we proceed recursively as follows:
If $r > k$, then *Rand-QSelect*$(k, S_{<x})$
Else *Rand-QSelect*$(k - r, S_{>x})$.

Where $S_{<x}$ and $S_{>x}$ are the sets consisting of all those elements that are respectively smaller and greater than the element x. Observe that the running time of the above algorithm is dominated by the number of comparisons performed. Therefore, in order to get a bound on the expected running time of the algorithm, our aim is essentially to find out the expected number of comparisons performed in *Rand-QSelect*(k, S). Prove the following statements.

1. The expected number of comparisons is at most $3.5n$. (You will loose 20% marks in this question if you are able to get a bound greater than $3.5n$).

2. There are elements in set S which will be compared expected $\Theta(\log n)$ times during the algorithm. Can you characterize these elements (this part of the problem is not to be submitted) ?

2 An interesting application of partition theorem

Recall the partition theorem, which states that if events $\mathcal{E}_1, ..., \mathcal{E}_\ell$ form a partition of a sample space Ω, and A is any event, then

$$\mathbf{P}[A] = \sum_{j=1}^{\ell} \mathbf{P}[A|\mathcal{E}_j] \ \cdot \ \mathbf{P}[\mathcal{E}_j]$$

This theorem can sometimes be used very effectively to calculate probability of event A when any direct method of calculating $\mathbf{P}[A]$ appears difficult. But applying this theorem effectively requires some creative skills and a better insight into the problem. In general, it works when the partition formed is such that calculating $\mathbf{P}[A|\mathcal{E}_j]$ is very easy for each $\mathcal{E}_j$ (in fact usually it turns out to be independent of j). We have already seen many application of partition theorem in the lectures (two problems).

As a warm up, try to solve the following problem that was given in the lecture on elementary probability theory (solution needs not be submitted).

There are n sticks each of different heights. There are n vacant slots arranged along a line and numbered from 1 to n as we move from left to right. The sticks are placed into the slots according to a uniformly random permutation. A stick placed at ith slot is said to be a dominating stick if its height is largest among all sticks placed in slots 1 to $i-1$. Let Let A be the event that the ith slot contains a dominating stick. Find the probability of A.

The first, and perhaps the most natural, approach to solve this problem would be to use the partition defined by the rank of the stick occupying ith slot. Conditioned on the event that jth smallest stick occupied ith slot ($j \geq i$), the probability of event A would be the following.

$$\frac{\binom{j-1}{i-1}(i-1)!(n-i)!}{n!}$$

In order to calculate unconditional probability $\mathbf{P}(A)$, you need to sum the above expression for all $i \leq j \leq n$. Convince yourself that this approach, and hence this partition scheme, does not work. Now think of a better partition (get inspired from some problem solved in the lectures).

With the above problem as a warm-up, now solve the following main problem. Elgoog, a very reputed company, is going to visit IITK to hire the best qualified (excellent knowledge of the fundamentals of computer science, excellent analytical & creative skills) student in his/her final year. There are n applicants and n is obviously really huge since Elgoog is offering a huge package. They will select a person based totally on his/her qualification which can be revealed only through interview. Assume that there is a total order among all n applicants as far as their qualifications are concerned. Since n is huge, it is not possible to interview every applicant. Furthermore, the placement office requires that each applicant should be informed about his/her selection or rejection immediately after the interview. Therefore, the following strategy is followed by Elgoog. They fix a number $k < n$. They interview and reject first k applicants. After that they continue taking interviews and stop as soon as they find an applicant better than the first k applicants. If they don't find any applicant better than the first k applicants, they return without hiring any one.

1. Assuming the applicants appear in a uniformly random order (all permutations are equally likely), what is the probability in terms of k and n that Elgoog will be successful in selecting the best qualified applicant ?

2. For what value of k, is the probability of selecting the best qualified applicant maximum ?

3 2-Dimensional Pattern Matching

marks = (10,20,30)

First we shall discuss a different finger printing technique to solve one-dimensional pattern matching problem. The idea is to map any bit string s into a 2×2 matrix $M(s)$, as follows.

- For the empty string ϵ, $M(\epsilon) = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$
- $M(0) = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$
- $M(1) = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$
- For non-empty strings x and y, $M(xy) = M(x) \times M(y)$.

Convince yourself that this fingerprint function has the following properties.

- $M(x)$ is well defined for all $x \in \{0,1\}^*$.
- $M(x) = M(y) \Rightarrow x = y$.

Starting with the two properties of M listed above, solve the following problems.

1. By using the matrix $M(x)$ for a bit-string x, design an algorithm for the pattern matching problem in 1-dimension (text and pattern are bit-strings of length n and m respectively). The algorithm should take $O(n+m)$ time. You may make use of the following assumption.
   Assumption: Any arithmetic operation involving two numbers of arbitrary length can be executed in $O(1)$ time.
2. It can be shown that for a bit string x of length n, entries of $M(x)$ can be quite long; however, they will be bounded by Fibonacci number F_n. Unfortunately, this fact suggests that the assumption mentioned above is not practical. In order to circumvent this problem, we need to work with small numbers - numbers of $O(\log n)$ bits only. Therefore, taking inspiration from one of the lectures, we keep entries of M modulo a prime number p picked randomly uniformly from a suitable range. This is now a practical but randomized Monte Carlo algorithm for 1-dimensional pattern matching. Do proper analysis to find the range from which you need to pick the prime number randomly to achieve error probability $< 1/n^4$.
   (*Most of you might be finding this pattern matching algorithm more complicated than the one we discussed in the class. However, this algorithm has the merit that it can be extended to solve 2-dimensional pattern matching very easily. You will do this extension as the last part of this problem.*)
3. Consider the two-dimensional version of the pattern matching problem. The text is an $n \times n$ bit-matrix X, and the pattern is an $m \times m$ bit-matrix Y. Of course, $m \leq n$. A pattern match occurs if Y appears as a (contiguous) sub-matrix of X. Get inspired from the randomized algorithm for part (b) above to design a randomized Monte Carlo algorithm for this 2-dimensional pattern matching problem. The running time should be $O(n^2)$ and the error probability should be $< 1/n^4$.

Preparation for Quiz on 4th February

Making an intelligent guess

marks=(40,10)

We have a function $F : \{O, ..., n-1\} \to \{O, ..., m-1\}$. We know that, for $0 \leq x, y \leq n-1$,

$$F((x+y) \bmod n) = (F(x) + F(y)) \bmod m$$

The only way we have for evaluating F is to use a lookup table that stores the values of F. Unfortunately, an Evil Adversary has changed the value of $1/5$ of the table entries when we were not looking. Describe a simple randomized algorithm that given an input z, outputs a value that equals $F(z)$ with probability at least $1/2$. Your algorithm should work for every value of z, regardless of what values the Adversary changed. Your algorithm should use as few lookups and as little computation as possible.

Suppose I allow you to repeat your initial algorithm k times. What should you do in this case, and what is the probability that your *enhanced* algorithm returns the correct answer?