CS648 : Randomized Algorithms
CSE, IIT Kanpur

Assignment 2

Deadline : 6:00 PM, 14 September, venue: RM404

Important Guidelines:

1. It is only through the assignments that one learns the most in any course on algorithms. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. **If you are cheating the instructor, you are cheating yourself first**. The onus of learning from a course lies first on you and then on the quality of teaching of the instructor. So act wisely while working on this assignment.

2. Handwritten submissions will not be accepted. You must type your solution using any word processor (For example : Latex, Microsoft Word, Google Doc). You will have to upload the soft copy on moodle before the deadline. You will also have to submit its printed copy before the deadline.

3. The answer of each questions must be formal, complete, and to the point. Do not waste your time writing intuition or idea. There will be penalty if you provide any such unnecessary details.

4. This assignment carries 150 marks.

1 Using prime numbers for 2-D pattern matching

Prime numbers are used very effectively in designing many randomized algorithms. We discussed such an algorithm for 1-dimension pattern matching. It could be seen as an application of finger-printing technique. You have to now design a randomized algorithm for 2-dimensional pattern matching.

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 $< 1/n^4$.

2 Perfect hashing with just 2 hash-tables

(*marks = 50*)

The aim of this exercise is to make you realize that simple ideas work quite well in randomized algorithms.

Let H be a universal hash family. In one of the lectures, we showed that we can use H to design a perfect hashing in $O(s^2)$ space. Interestingly, we can design a perfect hashing using much smaller space by using two instead of just one hash table as follows. We pick two hash functions h_1, h_2 randomly uniformly and independently from H. We create two hash tables T_1 and T_2 of size n each and insert the elements of S into these tables using the following algorithm.

Algorithm 1: Insert(S, T_1, T_2, h_1, h_2): Inserting elements of S into two hash tables.

```
1 for each a ∈ S do
2 |   i ← h_1(a);
3 |   if T_1(i) is empty then
4 |   |   Insert a in the list T(i);
5 |   else
6 |   |   j ← h_2(a);
7 |   |   Insert a in the list T_2(j)
8 |   end
9 end
```

It can be observed that each element of S is present in exactly one of the two tables T_1 and T_2. Searching an element $y \in U$ is also quite easy: first we search for y in $T_1[h_1(y)]$, and if y is not found there, we search for y in $T_2[h_2(y)]$.

Notice that at most one element is stored in each index of T_1. So collisions, if any, are contributed solely by elements present in T_2. We thus introduce the following terminology:

Any two elements of S are said to collide in the above hashing scheme if both of them are stored at the same index in T_2.

1. (*marks = 12*) Let a_k denote the element of S inserted during kth iteration in the above algorithm. Let $\mathcal{E}_k$ be the event that a_k is inserted in table T_2. Calculate $\mathbf{P}(\mathcal{E}_k)$.

2. (*marks = 12*) Let X_k denote the random variable defined as follows. If a_k is stored in T_1, then $X_k = 0$; otherwise, it is the number of elements among $\{a_1, \ldots, a_{k-1}\}$ that collide with it in T_2. Provide a suitable upper bound on $\mathbf{E}[X_k \mid \mathcal{E}_k]$.

3. (*marks = 26*) Use (b) to show that we can design a perfect hashing (at most one element in any location of T_1 and T_2) using n which is asymptotically much smaller than s^2.

Note: You must not assume any thing more that what is guaranteed by a universal hash family. It is quite common to make extra (though wrong) assumption in order to derive a better bound on the space. Please refrain from making such wrong assumptions.

3 How well did you internalize the proof of Chernoff bound ?

(*marks=50*)

Consider a collection $X_1, \cdots X_n$ of n independent geometrically distributed random variables with expected value 2. Let $X = \Sigma_{i=1}^{n} X_i$ and $\delta > 0$.

1. Derive a bound on $\mathbf{P}(X \geq (1+\delta)(2n))$ by appyling the Chernoff bound to a sequence of $(1+\delta)(2n)$ fair coin tosses.
2. Directly derive a Chernoff like bound on $\mathbf{P}(X \geq (1+\delta)(2n))$ from scratch.
3. Which bound is better?