CS345 : Algorithms II
Semester I, 2017-18, CSE, IIT Kanpur

Assignment 2

Deadline : 6:00 PM, 29 August 2017

Important note:

- 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. **Before cheating the instructor, you are cheating yourself**. 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.
- There are three problems in this assignment. Each problem has two choices, one easy and one hard, solve **exactly one** of the choices.
- Each assignment has to be done in groups of 2. You need to find your partner on your own. Please do not contact the instructor for this purpose.
- Handwritten submissions will not be accepted. You must type your solution using any word pro- cessor (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.
- 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.

1 Power of Data Structures

1. **An amazing geometric data structure** (*marks=25*)
   A line in a plane can be seen as dividing the plane into two regions. For any non-vertical line L, we can define these two regions as *upper* half-plane and *lower* half-plane. See Figure 1(i) for these concepts. Now we describe the problem.

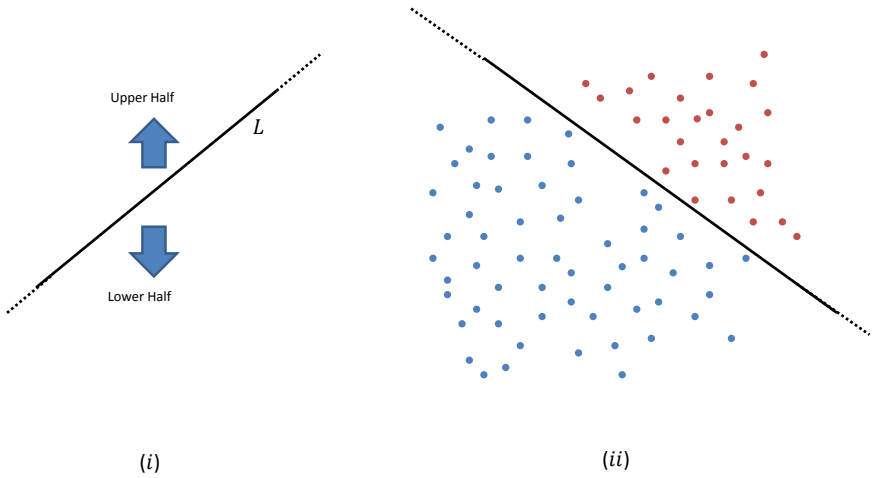

Figure 1: (i) The upper and lower half planes defined by a line. (ii) We need to report all red points to answer the query.

There is a set P of n points in a plane. You need to build an $O(n)$ size data structure storing these points such that given any non-vertical line L, all points of P that lie in the *upper* half-plane defined by L can be reported efficiently. Refer to Figure 1 (ii) for a better understanding. Your aim should be to achieve query time $O(k \log n)$, where k is the output size, i.e., the number of points lying in the corresponding upper half-plane of L. Isn't this amazing ? The line representing the half-plane could have *any* orientation, and still we may report all points lying in the upper half-plane by spending just $O(\log n)$ time per point. In the orthogonal range searching problem we had some structure on the query - it consists of a rectangle with sides parallel to the two axis. But for this problem, there is no such structure on the query, but still we are able to answer the query so efficiently. Hopefully this exercise will make you realize the power of data structures.

Hint: You will have to use some geometric structure discussed in the first week of this course. By the way, it is even possible to achieve $O(k+\log n)$ query time. However, that data structure is sophisticated and is beyond the scope of this course.

2. **Orthogonal Range Counting** (*marks=15*)

   In the class, we discussed the problem of orthogonal range searching in detail and gave following guarantees : size $=O(n \log n)$, query time $= O(k + \log^2 n)$ and pre-processing time $O(n \log n)$. Now in this problem your task is to design a data structure that supports query, $RangeCount(T, x_1, x_2, y_1, y_2)$, to report the number of points inside the rectangular region $[(x_1, y_1), (x_2, y_2)]$ with following guarantees : query time $= O(\log^2 n)$, size $= O(n \log n)$, pre-processing time $= O(n \log n)$.

2 Dynamic sequences

1. **Multi-add and min together** (*marks=20*)

   In the class, we discussed in detail an augmented BST for dynamic sequeces that supports Multi-add operation. We also mentioned the problem of Min operation on dynamic sequences as a homework. Now you need to have a single data structure which can support both of these operations.

   Design a data structure which supports the following operations on a dynamic sequence of real numbers. Each operation must take $O(\log n)$ time and the size of the data structure must be $O(n)$, where n is the length of the sequence at the time of the operation.

   - Insert(D, i, x): Insert an element x at ith place in the sequence.
   - Delete(D, i): Delete ith element from the sequence.
   - Multi-add(D, i, j, Δ): Add Δ to each element of the sequence starting from ith place to jth place.
   - Min(D, i, j): Report the smallest element in the sequence starting from ith place to jth place.

2. **Flip bits** (*marks=15*)

   Design a data structure which supports the following operations on a dynamic sequence of bits $\langle b_1, b_2, \cdots, b_n \rangle$. Each operation must take $O(\log n)$ time and the size of the data structure must be $O(n)$, where n is the length of the sequence at the time of the operation.

   (a) Insert(i,b): Insert a bit of value b at position i.

   (b) Report(i): Return the value of ith bit.

   (c) Delete(i): Delete ith bit.

   (d) Multi-flip(i, j): Flip all bits starting from ith location to jth location in the sequence.

3 Generic way to analyse a Greedy strategy

1. **Hierarchical metric** (*marks=20*)

   There is a complete graph G on n vertices $V = \{v_1, \ldots, v_n\}$. There is a mapping $d : V \times V \to R^+$ such that $d(v_i, v_j) \geq 0$ for each $i \neq j$. Further $d(v_i, v_i) = 0$ for all i.

   Consider a rooted tree T whose leaves are V. Each node x in T has a label $h(x)$ such that $h(x) = 0$ if x is a leaf node, and $h(u) \geq h(v)$ if u is parent of v in T. For any two vertices $v_i, v_j \in V$, their distance in T is defined as $h(x)$ where x is the lowest common ancestor of v_i and v_j in T. T is said to be consistent with G if distance between each $v_i, v_j \in V$ in tree T is less than or equal to $d(v_i, v_j)$. Aim is to compute a tree T^* for a given graph G satisfying the following conditions.

   - T^* is consistent with G
   - For any other tree T' consistent with G, the following condition must hold for each $v_i, v_j \in V$:
     "distance between v_i and v_j in T' is less than or equal to their distance in T^*."

   You need to design a greedy algorithm to compute tree T^* using a generic strategy for greedy algorithms that we discussed in the class .

   (a) Provide complete details of the greedy step and the smaller instance G' obtained by applying the greedy step.

   (b) State the theorem that relates $T^*(G)$ with $T^*(G')$ and give a short proof for it.

   (c) Describe the fastest possible implementation of the algorithm based on the greedy step described above.

2. **Synchronizing the circuit** (*marks=15*)

   This is the problem we discussed in the class. However, we shall now design and analyse an algorithm along the generic framework of the Greedy algorithms discussed in the class. Let us restate the problem for the sake of completeness.

   There is a complete binary tree representing a circuit. There are n leaves. Each edge has a positive no. which represents the delay along that edge. An electric pulse originates at the root and travels all the way to the leaves. The delay incurred along a path is sum of the delay on its edges. You need to ensure that the pulse reaches all leaves at the same time. As a result the entire circuit becomes synchronized. To achieve this objective, you are allowed to increase the delay along some edges. The delay enhancement is the sum of increase in the delay of all the edges. The aim is to synchronize the given circuit such that total delay enhancement in the circuit is minimum.

   (a) You have to design a greedy step that transforms the given instance to a smaller instance of the same problem. You might like to *generalize* the problem for this purpose.

   (b) Then you have to establish a relation (and prove it formally) between the optimal solution of the given instance and optimal solution of the smaller instance.

   (c) Describe the fastest possible implementation of the algorithm based on the greedy step and the relation derived above.