CSC265 F18: Assignment 4

Due: November 7, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources. Solutions typed using LaTeX receive 2 bonus marks.
- Your submission should be no more than 6 pages long, in a single-column US Letter or A4 page format, using at least 9 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2018-09`
- This is a *group assignment*. This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (14 marks)

In this problem you are given as input a set P of n *distinct* points in the plane. Each point p is specified by two *integer* coordinates, $p.x$ and $p.y$, which are the x- and the y-coordinates of p. We have $0 \le p.x \le N-1$ and $0 \le p.y \le N-1$, where $N = \Theta(n^{100})$. You can assume that P is given as a linked list of point objects.

Suppose that, together with P, you are also given an integer $t \ge 0$. Design a *randomized* algorithm which returns a linked list of all pairs of points (p_1, p_2) such that $p_1, p_2 \in P$, $p_1 \ne p_2$ and $d(p_1, p_2) \le t$. Here $d(p_1, p_2)$ is the *distance* between p_1 and p_2, given by the formula

$$d(p_1, p_2) = \sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2}.$$

Let s_r be, for any integer $r \ge 0$, the size of the set $\{(p_1, p_2) : p_1, p_2 \in P, p_1 \ne p_2, d(p_1, p_2) \le r\}$. Your algorithm should run in expected time $O(n + s_{2\sqrt{2}t})$ and use $O(n + s_{2\sqrt{2}t})$ words of space for every input P, t.

Describe your algorithm using clear and precise English, and, optionally, using pseudocode. Justify its correctness and why it satisfies the time and space requirements above.

[Solution]

First we describe a way to turn the x- and y-coordinates of a point p into a single number in a one-to-one way. For any point p with coordinates in $[0, N-1]$, we set $g(p) = Np.x + p.y$. Then g takes values in $[0, N^2 - 1]$. Moreover, it is one to one, since from $g(p)$ we can recover $p.x$ as $\lfloor g(p)/N \rfloor$, and $p.y$ as $g(p) \bmod N$.

We are going to divide the square $[0, N-1]^2$ in which all points in P lie into smaller squares of side length t and hash points based on which smaller square they lie in. Formally, define a function $r(p) = (\lfloor p.x/t \rfloor, \lfloor p.y/t \rfloor)$. Thus all points in the square $[0, t-1] \times [0, t-1]$ go to the point $(0, 0)$, all points in $[0, t-1] \times [t, 2t-1]$ go to $(0, 1)$, etc.

Finally, we define the function $f(p) = g(r(p))$. Since g is one to one, we have that $f(p) = f(q)$ if and only if they lie in the same square of side length t, i.e. if and only if $r(p) = r(q)$.

We pick a hash function $h : \{0, \ldots, N^2 - 1\} \to \{0, \ldots, m\}$ from a universal family, where $m = \Theta(n)$. We go over all points p in P, and we insert p into the hash table at the slot $h(f(p))$.

Then we go through the list P once again, and for each $p \in P$, compute the pair $r(p) = (a, b)$, idenfying the square in which p lies. For each of the nine pairs $(a + c, b + d)$, where $c, d \in \{-1, 0, 1\}$, we go through all points q stored in $T[h(g(a + c, b + d))]$ (if $a + c$ and $b + d$ are in $[0, N - 1]$), and, if $d(p, q) \le t$, we add the pair (p, q) to the list L of pairs to be returned. This describes the entire algorithm.

This algorithm will add every pair to be returned twice to the list L. We can make sure every pair appears at most once by adding (p_1, p_2) only if, say, $g(p_1) < g(p_2)$. (This is one way to impose order on the points; of course, there are others, but this is convenient as we already have the function g defined.)

Let p be a point such that $r(p) = (a, b)$. Then p lies in the square $[at, (a+1)t-1] \times [bt, (b+1)t-1]$. If we look at any other point q with $r(q) = (a', b')$, it must lie in the square $[a't, (a'+1)t-1] \times [b't, (b'+1)t-1]$. Then, if $|a' - a| \ge 2$, it must be the case that $|p.x - q.x| > t$, and, similarly, $|b' - b| \ge 2$ implies $|p.y - q.y| > t$. Therefore, any point q such that $d(p, q) \le t$ must have $r(q) = (a + c, b + d)$ where $c, d \in \{-1, 0, 1\}$, and the algorithm returns all pairs of distinct points in P at distance at most t.

It remains to bound the expected running time. It takes time $O(n)$ to insert all points in P into the hash table. Let us fix some $p \in P$ with coordinates x and y, and $r(p) = (a, b)$. Let us also fix some $c, d \in \{-1, 0, 1\}$. Let $P_{p,c,d} = \{q \in P : r(q) = (a+c, b+d)\}$, and let $n_{p,c,d}$ be the number of points stored in $T[h(g(a+c, b+d))]$. We will argue that $\mathbb{E}[n_{p,c,d}] \le |P_{p,c,d}| + \frac{n}{m}$. To see this, define, for any point $q \in P$, the indicator random variable X_q to equal 1 if and only if $h(f(q)) = h(g(r(q))) = h(g(a+c, b+d))$. Then, $X_q = 1$ with probability 1 for any $q \in P_{p,c,d}$. For any $q \notin P_{p,c,d}$, we have that $g(r(q)) \ne g(a + c, b + d)$ because g is one to one, and, because h was sampled from a universal family, $\Pr(X_q = 1) \le \frac{1}{m}$. This implies that

$$\mathbb{E}[n_{p,c,d}] = \mathbb{E}\left[\sum_{q \in P} X_q\right] = \sum_{q \in P} \Pr(X_q = 1) \le |P_{p,c,d}| + \frac{|P \setminus P_{p,c,d}|}{m} \le |P_{p,c,d}| + \frac{n}{m}.$$

The total running time of the algorithm is bounded, up to constant factors, by

$$n + \sum_{p \in P} \sum_{c=-1}^{1} \sum_{d=-1}^{1} n_{p,c,d}.$$

Therefore, by linearity of expectation, the total expected running time is bounded by

$$n + \sum_{p \in P} \sum_{c=-1}^{1} \sum_{d=-1}^{1} \mathbb{E}[n_{p,c,d}] \leq n + \frac{9n^2}{m} + \sum_{p \in P} \sum_{c=-1}^{1} \sum_{d=-1}^{1} |P_{p,c,d}|.$$

The first two terms on the right are $O(n)$ since $m = \Theta(n)$. To bound the sum, observe that for any $q \in P_{p,c,d}$,

$$d(p,q)^2 = |p.x - q.x|^2 + |p.y - q.y|^2 \leq 4t^2 + 4t^2 = 8t^2,$$

i.e. $d(p,q) \leq 2\sqrt{2}t$. Moreover, the number of times pairs (p,q) of points in P such that $d(p,q) \leq 2\sqrt{2}t$ contribute to the sum is at most twice: once to $P_{p,c,d}$ for the appropriate c, d, and once to $P_{q,-c,-d}$. Therefore, the sum is bounded by $2s_{2\sqrt{2}t}$. This completes the analysis of the algorithm.

Question 2. (14 marks)

(The set up is the same as in the last Assignment)

Let T be a complete ternary tree, i.e. a tree in which all leaf nodes are at the same distance from the root, and every tree node except the leaves has exactly three children: a left, a middle, and a right child. Then T has 3^h leaves, where h is its height. Suppose that each leaf of T is given a value 0 or 1. Then we determine values for the internal nodes of T as follows: the value of an internal node u of T equals 0 if at least two of its children have value 0, and its value equals 1 otherwise.

Assume that the values of the leaves, numbered from left to right, are given in an array $A[1\,..\,3^h]$. Give a *randomized* algorithm which computes the value of the root of T so that in the worst case (over the choice of A) the *expected number* of entries of A queried by the algorithm is at most C^h, where $C < 3$ is a constant. Give a precise value for C and justify it, and also justify the correctness of your algorithm.

[Solution]

The algorithm picks two random children of the root uniformly at random, and recursively determines their value. If it is the same, then this is also the value of the root, and we are done. If the two random children have different values, we also recursively determine the value of the remaining third child of the root, and assign its value to the root as well.

Let $N(h)$ be the worst-case expected number of entries of A queried by the algorithm when run on a complete ternary tree of height h. Let $X(u)$ be a random variable equal to the number of leaves under node u (corresponding to entries of A) examined by our algorithm when called on node u of the tree. Let r be the root of the tree, and let U_1 and U_2 be the first two random nodes on which we recursively call our algorithm. Let E be the event that the values of U_1 and U_2 are equal, which happens with probability at least $1/3$ as two out of the three children of r must have the same value. If E happens, then the value of r equals the values of U_1 and U_2 and we do not call the algorithm on the third child. If E does not happen, then we call the algorithm on the third child, which we denote U_3, as well.

So, $N(h)$ satisfies the equalities

$$\begin{aligned} N(h) = \mathbb{E}[X(r)] &= \mathbb{E}[X(r)|E] \Pr(E) + \mathbb{E}[X(r)|\bar{E}](1 - \Pr(E)) \\ &= \mathbb{E}[X(U_1) + X(U_2)|E] \Pr(E) + \mathbb{E}[X(U_1) + X(U_2) + X(U_3)|\bar{E}](1 - \Pr(E)) \end{aligned}$$

Notice that, by the law of total expectation, for any i, $\mathbb{E}[X(U_i)|E] = \mathbb{E}[\mathbb{E}[X(U_i)|U_i]|E]$, and that because the tree under U_i is a complete ternary tree of height $h-1$, by definition $\mathbb{E}[X(U_i)|U_i] \leq N(h-1)$. Therefore,

$\mathbb{E}[X(U_i)|E] \leq N(h-1)$ for any i. Analogously, $\mathbb{E}[X(U_i)|\bar{E}] \leq N(h-1)$ as well. Then, $N(h)$ satisfied the recurrence

$$N(h) \leq 2N(h-1)\Pr(E) + 3N(h-1)(1-\Pr(E)) \leq \frac{8}{3}N(h-1).$$

Moreover, $N(0) = 1$. This recursion is solved by $N(h) = \left(\frac{8}{3}\right)^h$.