Design and Analysis of Algorithms

PSet-2

23 September 2025

This is a compilation of solutions for PSet-2 of Design and Analysis of Algorithms class instructed by Prof. Siddharth Pritam. The solutions are of original design.

Table of contents

§0. Problems

1. We are given n points in the unit disk, $p_i = (x_i, y_i)$, such that $0 \leq x^2 + y^2 \leq 1$ for $i \in \{1, 2, ..., n\}$.

   Suppose that the points are uniformly distributed; that is, the probability of finding a point in any region of the circle is proportional to the area of that region. Design an algorithm with an average-case running time of $\Theta(n)$ to sort the n points by their distances $d_i = \sqrt{x_i^2 + y_i^2}$ from the origin. (Hint: Use ideas from the BUCKET-SORT.)

2. Suppose that we insert n keys into a hash table of size m using open addressing and uniform hashing. Let $p(n, m)$ be the probability that no collisions occur. Show that $p(n, m) \leq e^{-\frac{n(n-1)}{2m}}$. Argue that when n exceeds $\sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

3. Let G be a weighted undirected graph and T is a minimum spanning tree of G, suppose that we decrease the weight of one of the edges of G not in T . Give an efficient algorithm for finding the minimum spanning tree in the modified graph.

4. Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique minimum weight crossing edge. Show that the converse is not true by giving a counterexample.

5. Given two positive n digit integers, a and b. Design an algorithm that multiplies a and b in $O(n^{1.58})$ time. (Hint: Use divide and conquer technique.)

6. Given n chords on a circle, each defined by its endpoints. Describe an $O(n \log(n))$ time algorithm to determine the number of pairs of chords that intersect inside the circle. Assume that no two chords share an endpoint.

§1. Problem 1

Exercise

We are given n points in the unit disk, $p_i = (x_i, y_i)$, such that $0 \leq x^2 + y^2 \leq 1$ for $i \in \{1, 2, ..., n\}$.

Suppose that the points are uniformly distributed; that is, the probability of finding a point in any region of the circle is proportional to the area of that region. Design an algorithm with an average-case running time of $\Theta(n)$ to sort the n points by their distances $d_i = \sqrt{x_i^2 + y_i^2}$ from the origin. (Hint: Use ideas from the BUCKET-SORT.)

Solution. Consider $r_1, r_2, ..., r_m$ such that

$$\pi r_1^2 = \frac{\pi}{m}$$

$$\pi(r_i^2 - r_{i-1}^2) = \frac{\pi}{m}$$

$$\Rightarrow r_1 = \frac{1}{\sqrt{m}}$$

$$\Rightarrow r_i = \sqrt{\frac{1}{m} + r_{i-1}^2}$$

$$\Rightarrow r_i = \sqrt{\frac{i}{m}} \text{ by induction}$$

Taking our buckets as $(b_1, b_2, ..., b_m) = \left[0, \frac{1}{\sqrt{m}}\right], \left(\frac{1}{\sqrt{m}}, \sqrt{\frac{2}{m}}\right], ..., \left(\sqrt{\frac{n-1}{m}}, 1\right]$

ALGORITHM TO BUCKET SORT THE POINTS

```
ans = []
for i ∈ 1...n:
    put p_i in it's respective bucket
for i ∈ 1...m:
    sort(b_m) ++ ans
return ans
```

Initializing the buckets takes $O(m)$ and scattering the elements into the buckets takes $O(n)$ time as we can just square and multiply by m and take floor to get the bucket.

Concatenating the sorted buckets also takes $O(m)$ time.

Thus, we now need to talk about the time to sort the buckets.

$$
\begin{aligned}
T(n) &= \mathbb{E}\left(\sum_{i=1}^{m} n_i^2\right) \text{ where} \sum_{i=1}^{m} n_i = n \\
&= \sum_{i=1}^{m} \mathbb{E}(n_i^2) \text{ where} \sum_{i=1}^{m} n_i = n \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \mathbb{E}\big(x_{i,j}\big)\right)^2 \text{ where } x_{i,j} \text{ is the 1 if } p_j \in b_{|i|} \text{ and 0 otherwise.} \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \sum_{k=1}^{n} \mathbb{E}\big(x_{i,j}\big) E\big(x_{i,k}\big)\right) \\
&= \sum_{i=1}^{m} \frac{n^2}{m^2} \text{ as } \mathbb{E}\big(x_{i,j}\big) = \frac{1}{m} \\
&= \frac{n^2}{m}
\end{aligned}
$$

This makes the complexity $\Theta\left(n + \frac{n^2}{m} + m\right)$ which is $\Theta(n)$ if we take $m = n$. □

§2. Problem 2

Exercise

Suppose that we insert n keys into a hash table of size m using open addressing and uniform hashing. Let $p(n, m)$ be the probability that no collisions occur. Show that $p(n, m) \leq e^{-\frac{n(n-1)}{2m}}$. Argue that when n exceeds $\sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

Proof. Notice,

$$\begin{aligned}
p(n,m) &= \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-n+1}{m} \\
&= \prod_{i=1}^{n} \frac{m-i+1}{m} \\
&= \prod_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{(m-i)(m-n+i)}{m^2} \\
&= \prod_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{(m-\frac{n}{2}+\frac{n}{2}-i)(m-\frac{n}{2}-\frac{n}{2}+i)}{m^2} \\
&= \prod_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{(m-\frac{n}{2})^2-(\frac{n}{2}-i)^2}{m^2} \\
&\leq \prod_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} \frac{(m-\frac{n}{2})^2}{m^2} \\
&\leq \frac{(m-\frac{n}{2})^{n-1}}{m^{n-1}} \\
&= \left(1-\frac{n}{2m}\right)^{n-1} \\
&< \left(e^{-\frac{n}{2m}}\right)^{n-1} \text{ using } 1+x<e^x \Rightarrow 1-x<e^{-x} \\
&< e^{-\frac{n(n-1)}{2m}}
\end{aligned}$$

If

$$\begin{aligned}
& n > \sqrt{m} \\
& \Rightarrow p(n,m) \\
& < e^{-\frac{n(n-1)}{2m}} \\
& < e^{-\sqrt{m}\frac{\sqrt{m}-1}{2m}} \\
& < e^{-\frac{\sqrt{m}-1}{2\sqrt{m}}} \\
& < e^{-\frac{1}{2}} e^{\frac{1}{2\sqrt{m}}}
\end{aligned}$$

As m increases, $p(n, m)$ exponentially approaches 0. □

§3. Problem 3

Exercise

Let G be a weighted undirected graph and T is a minimum spanning tree of G, suppose that we decrease the weight of one of the edges of G not in T . Give an efficient algorithm for finding the minimum spanning tree in the modified graph.

Given the input to be u, v which are the vertices the edge connects and w' be it's new weight.

ALGORITHM TO UPDATE MST

```
1  def update(G, T, u, v, w')
2      P = path(u,v, T)
3      w_max = maxWeightedEdge(P)
4      if weight(w_max) < w'
5          return T
6      else:
7          return (T - w_max + (u,v))
```

Proof. Adding any other edges of the graph can't be better as that would imply T was not the minimal spanning tree of the unmodified graph. Thus, we only need to consider the newly updated edge.

as a tree is the maximal acyclic graph. Thus, adding a new edge makes it cyclic.

As it is the minimal connecting graph, removing an edge from the cycle, would make it a tree again (as connecting is maintained).

We can remove the most costly edge in this cycle, and the remaining tree is minimum spanning as no more edges can be removed and removing any other edge is inefficient. □

Time Complexity. We can find the path in atmost $O(n)$ time where n is number of vertices. We find $w_{\max}$ in $O(n)$ time and make the update in $O(1)$ time.

Thus, we have an $O(n)$ algorithm. □

Remark

As the path algorithm is not done in class, although we are yet to touch graph algorithms in general, so I am not sure if I as to suppose that an algorithm exists or not.

Anyways, for the sake of completness, I have included a way to find the path by DFS.

PATH IN TREE

```
1  def path(u, v, T):
2  visited = []
3  parent = []
4  stack = [u]
5  visited.add(u)
6  parent[u] = null
7  while stack not empty:
8      node = stack.pop()
9          if node == v:
10             break
11         for neighbor in T[node]:
12             if neighbor not in visited:
13                 visited.add(neighbor)
14                 parent[neighbor] = node
15                 stack.push(neighbor)
16 path = []
17 current = v
18 while current is not null:
19     path.append(current)
20     current = parent[current]
21 reverse(path)
22 return path
```

In worst case it clearly visits all vertices and all the edges, hence, $O(|V| + |V| - 1) = O(|V|)$ is the time complexity. Proof of correctness is trivial in this case as e literally just check the entire tree.

§4. Problem 4

Exercise

Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique minimum weight crossing edge. Show that the converse is not true by giving a counterexample.

Proof. (B) For a graph of size 1, the vertex alone is the MST.

For a graph of size 2, the only edge is also the MST.

(S) Let an MST exist for graphs of size $n-1$. Consider $|G| = n$.

By the induction hypothesis, We have an unique MST covering $n-1$ vertices. Let the not included vertex be v.

Then, $G - v$ and v form a cut and there is a minimum weight crossing edge. We claim adding this edge forms an MST as adding any other edge would lead to a heavier spanning tree.

This is an unique MST as the induction implies that the MST covering $G - v$ as unique and as there is a unique minimum weight crossing edge. □

We claim $V = \{A, B, C\}$ and $w(A, B) = 1, w(B, C) = 1, w(A, C) = 2$ is a counterexample to the converse. Then the MST is $\{(A, B), (B, C)\}$ and is unique. However, the unique minimum weight crossing edge is violated as V and $V - B$ has 2 distinct edges of same, minimum weight.

§5. Problem 5

Exercise

Given two positive n digit integers, a and b. Design an algorithm that multiplies a and b in $O(n^{1.58})$ time. (Hint: Use divide and conquer technique.)

MULTIPLICATION VIA TOOM-COOK-3

```
1  def multiply(a,b,n):
2      m := n + (3 − n mod 3)
3      append 0's to a till it is of length m
4      append 0's to b till it is of length m
5      (a_1, a_2, a_3) := (first m/3 digits of a, middle m/3 digits of a, last m/3 digits of a)
6      (b_1, b_2, b_3) := (first m/3 digits of b, middle m/3 digits of b, last m/3 digits of b)
7      w_1 = multiply(a_1, b_1, m/3)
8      w_2 = multiply(a_3, b_3, m/3)
9      w_3 = multiply(a_1 + 2a_2 + 4a_3, b_1 + 2b_2 + 4b_3, m/3 + 2)
10     w_4 = multiply(a_1 − a_2 + a_3, b_1 − b_2 + b_3, m/3 + 1)
11     w_5 = multiply(a_1 + a_2 + a_3, b_1 + b_2 + b_3, m/3 + 1)
12     c_1 = w_1
13     c_5 = w_2
14     c_3 = (w_4+w_5)/2 − w_1 − w_2
15     c' = (w_5−w_4)/2
16     c_2 = (w_3−w_1−16w_2−4c_3−8c')/−6
17     c_4 = (w_3−w_1−16w_2−4c_3−2c')/6
18     return c_1 · 10^(4n/3) + c_2 · 10^n + c_3 · 10^(2n/3) + c_4 · 10^(n/3) + c_5
```

Proof of correctness.

$$ab = \underbrace{a_1b_1}_{c_1} \cdot 10^{\frac{4n}{3}} + \underbrace{(a_2b_1 + a_1b_2)}_{c_2} \cdot 10^n + \underbrace{(a_3b_1 + a_2b_2 + a_1b_3)}_{c_3} \cdot 10^{\frac{2n}{3}} + \underbrace{(a_2b_3 + a_3b_2)}_{c_4} \cdot 10^{\frac{n}{3}} + \underbrace{a_3b_3}_{c_5}$$

$$
\begin{aligned}
w_1 &= a_1 b_1 = c_1 \\
w_2 &= a_3 b_3 = c_5 \\
w_3 &= (a_1 + 2a_2 + 4a_3)(b_1 + 2b_2 + 4b_3) \\
&= a_1 b_1 + 2a_1 b_2 + 4a_1 b_3 \\
&\quad 2a_2 b_1 + 4a_2 b_2 + 8a_2 b_3 \\
&\quad 4a_3 b_1 + 8a_3 b_2 + 16a_3 b_3 \\
w_4 &= (a_1 - a_2 + a_3)(b_1 - b_2 + b_3) \\
&= a_1 b_1 - a_1 b_2 + a_1 b_3 \\
&\quad -a_2 b_1 + a_2 b_2 - a_2 b_3 \\
&\quad +a_3 b_1 - a_3 b_2 + a_3 b_3 \\
w_5 &= (a_1 + a_2 + a_3)(b_1 + b_2 + b_3) \\
&= a_1 b_1 + a_1 b_2 + a_1 b_3 \\
&\quad +a_2 b_1 + a_2 b_2 + a_2 b_3 \\
&\quad +a_3 b_1 + a_3 b_2 + a_3 b_3
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow w_4 + w_5 = 2(a_1 b_1 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_3 b_3) \\
&\Rightarrow \frac{w_4 + w_5}{2} - w_1 - w_2 = a_1 b_3 + a_2 b_2 + a_3 b_1 = c_3
\end{aligned}
$$

$$
\begin{aligned}
w_5 - w_4 &= 2(a_1 b_2 + a_2 b_1 + a_2 b_3 + a_3 b_2) \\
\Rightarrow \frac{w_5 - w_4}{2} &= a_1 b_2 + a_2 b_1 + a_2 b_3 + a_3 b_2 = c' \\
\Rightarrow w_3 - w_1 - 16w_2 - 4c_3 - 8c' &= -6(a_2 b_1 + a_1 b_2) \\
\Rightarrow \frac{w_3 - w_1 - 16w_2 - 4c_3 - 8c'}{-6} &= c_2 \\
\Rightarrow w_3 - w_1 - 16w_2 - 4c_3 - 2c' &= 6(a_2 b_3 + a_3 b_2) \\
\Rightarrow \frac{w_3 - w_1 - 16w_2 - 4c_3 - 2c'}{6} &= c_4
\end{aligned}
$$

□

Time Complexity.

$$
\begin{aligned}
T(n) &= 5T\left(\frac{n}{3}\right) + O(1) \\
\Rightarrow T(n) &= O\left(n^{\log_3(5)}\right) \quad \text{by master's theorem} \\
\Rightarrow T(n) &= O(n^{1.46})
\end{aligned}
$$

As $1.46 < 1.58, T(n) = O(n^{1.58})$. □

> **Remark**
>
> I would like to point out that $2^{1.58} = 2.98969849727 < 3$. This implies that any solution to this problem using Karaturba's algorithm should be penalized.
>
> I might also add
>
> $$\begin{aligned}&\left(a_1 10^{\frac{n}{2}} + a_2\right)\left(b_1 10^{\frac{n}{2}} + b_2\right)\\ &= a_1 b_1 10^n + (a_1 b_2 + a_2 b_1) 10^{\frac{n}{2}} + a_2 b_2\\ &= a_1 b_1 10^n + ((a_1 + b_1)(a_2 + b_2) - a_1 b_1 - a_2 b_2) 10^{\frac{n}{2}} + a_2 b_2\end{aligned}$$
>
> is something I can do (and am aware of). I just respect the exact number given in the assignment and have solved accordingly.

§6. Problem 6

Exercise

Given n chords on a circle, each defined by its endpoints. Describe an $O(n \log(n))$ time algorithm to determine the number of pairs of chords that intersect inside the circle. Assume that no two chords share an endpoint.

ALGORITHM TO COUNT POINTS ON CIRCLE

```
1  Choose arbitrary point p on circle
2  lis = list of chords c_i = (x_i, y_i) where x_i is closer to p than y_i moving clockwise (flip otherwise)
3  x' = sort lis based of distance of x_i from p in anti-clockwise direction in ascending order
4  y' = sort lis based of distance of y_i from p in clockwise direction in, ascending order
5  count = 0
6  for i in 1..n:
7      current_chord = x'_i
8      count += index(current_chord, y')
9      remove current_chord from y'
10 return count
```

Proof of Correctness. We are using the fact that wrt to a point p, if let our chords be (x_i, y_i) where x_i is closer to p than y_i moving clockwise; then two chords, c_i, c_j intersect if and only if either of the following are true

- $p \curvearrowleft x_i > p \curvearrowleft x_j$ and $p \curvearrowright y_i < p \curvearrowright y_j$
- $p \curvearrowleft x_i < p \curvearrowleft x_j$ and $p \curvearrowright y_i > p \curvearrowright y_j$

Where $a \curvearrowleft b$ is the anti-clockwise distance from a to b and $a \curvearrowright b$ is the clockwise distance from a to b.

We simply start with a point such that $p \curvearrowleft x_i < p \curvearrowleft x_j$ forall $j \neq i$ and then count the number of points where the clockwise distance to y_j is lesser than that to y_i. This is the count of all the chords that intersect c_i.

We then remove c_i from consideration. □

Time Complexity. All distance calculations and flipping takes $O(n)$ time.

The sorting takes $O(n \log(n))$ time.

The for loop performs $O(1)$ operations n times leading to $O(n)$ time.

Thus, we are done in $O(n) + O(n \log n) + O(n) = O(n \log n)$ time. □