COL351 Assignment - 3

Aditya (2019CS50471)
Ajay (2019CS10323)

27^{th} October, 2021

Contents

1 Convex Hull

Given set P of n input points (x_1,y_1)..........(x_n,y_n)
Algorithm Sketch:
Preprocessing step:
Sort the given set P by x-coordinate
Base case:
if $|P| \leq 3$, solve the problem directly else we perform divide and conquer.
Divide step:
Divide the set of points into sets P_1 and P_2. P_1 contains left $\lfloor n/2 \rfloor$ points, P_2 contains right $\lceil n/2 \rceil$ points.
Conquer step:
Recursively compute convex hull of sets P_1 and P_2.
Merge step:
Here, we need to combine convex hull of P_1 and P_2 to obtain convex hull of P.
let convex hull of P_1 and P_2 be $CH(P_1)$ and $CH(P_2)$ respectively.
let's define upper tangent and lower tangent.

Upper tangent:
The edge $\overline{uv}$, where $u \in CH(P_1)$ and $v \in CH(P_2)$ such that:
All the vertices in $CH(P_1)$ and $CH(P_2)$ are below $\overline{uv}$.
$\Rightarrow$ The two neighbours of u in $CH(P_1)$ and two neighbours of v in $CH(P_2)$ are below $\overline{uv}$.
$\Rightarrow$ Anticlockwise neighbour of u in $CH(P_1)$ and clockwise neighbour of v in $CH(P_2)$ are below $\overline{uv}$
Lower tangent:
Lower tangent is defined similarly as $\overline{uv}$, where $u \in CH(P_1)$ and $v \in CH(P_2)$ such that clockwise

neighbour of u in CH(P_1) and anticlockwise neighbour of v in CH(P_2) are below $\overline{uv}$
An example of upper and lower tangents

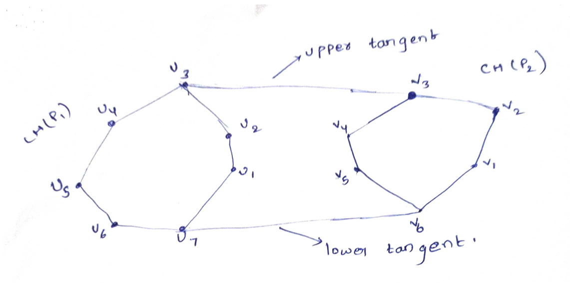

Figure 1: Example of Upper Tangent and Lower Tangent

Once we find lower and upper tangents we can find CH(P) by taking vertices in CH(P_1) from upper tangent to lower tangent in anti clockwise direction and taking vertices in CH(P_2) from lower tangent to upper tangent in anticlockwise direction

In the above example CH(P) = (u_3,u_4,u_5,u_6,u_7,v_6,v_1,v_2,v_3)

Algorithm to find Upper tangent:
We use the definition defined earlier.

```
def findUpperTangent(CH(P1),CH(P2))
     u - rightmost point in CH(P1)
     v - leftmost point in CH(P2)
     while true:
          if anticlockwise neighbour(u) is above uv:
               u = counterclockwiseneighbour(u)
          if clockwise neighbour(v) is above uv:
               v = clockwise neighbour(v)
          else:
               return(u,v)
```

Algorithm to find lower tangent is similar, we use the definition of lower tangent in above algorithm.

Pseudo code:

```
P - sort P by x-cordinate
def ConvexHull(P):
     if (|P|≤3):
          return P sorted in anticlockwise order
     P1 - left n/2 points of P
     P2 - right n/2 points of P
     CH(P1) = ConverxHull(P1)
     CH(P2) = ConverxHull(P2)
     UT = findUpperTangent(CH(P1),CH(P2))
     LT = findUpperTangent(CH(P1),CH(P2))
     CH(P) = {}
     for each vertex u in CH(P1) from UT to LT in anticlockwise order:
          CH(P).append(u)
     for each vertex v in CH(P2) from LT to UT in anticlockwise order:
          CH(P).append(v)
     return CH(P)
```

Time Complexity:
1) Divide step takes O(n) time, because we are splitting left $\lfloor n/2 \rfloor$ points to P_1,right $\lceil n/2 \rceil$ points to P_2
2) Conquer step takes 2T(n/2)
3) Finding tangents takes O(n) because let n_1 be no of points in CH(P_1) and n_2 be no of points in CH(P_2). In each step we move either u or v (u,v as defined in algorithm). So, atmost n_1+n_2 steps and $n_1+n_2 \leq$ n.So, time to find tangents it take O(n) time
4)Finding CH(P) takes O(n) because we look vertices in CH(P_1) and CH(P_1) and append correct vertices to CH(P) which takes O(n_1+n_2) = O(n)
So T(n) = 2T(n/2)+ O(n)
$\Rightarrow$ T(n) = O(nlogn)
We perform sorting once in preprocessing step which takes O(nlogn) time

So, Total time complexity = O(nlogn)

2 Particle Interaction

Given charged particles are placed at 1,2,.....n.
At each point j, particle has charge q_j

$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2}$ - $\sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}$
$\Rightarrow \frac{F_j}{cq_j} = \sum_{i<j} \frac{Cq_i}{(j-i)^2}$ - $\sum_{i>j} \frac{Cq_i}{(j-i)^2}$

let A = (q_1, q_2, q_3,....,q_n)
$\Rightarrow A(x) = q_1 + q_2\ x + q_3\ x^2 + ... + q_n\ x^{n-1}$

let B = ($\frac{-1}{(n-1)^2}$, $\frac{-1}{(n-2)^2}$,.....,-1,0,1,....., $\frac{1}{(n-2)^2}$,$\frac{1}{(n-1)^2}$)
$\Rightarrow B(x) = \frac{-1}{(n-1)^2} + \frac{-1}{(n-2)^2}\ x + ... + 0x^{n-1} + ... + \frac{1}{(n-2)^2}x^{2n-3} + \frac{1}{(n-1)^2}x^{2n-2}$

We can zero pad A(x) to make it same length as B(x) i.e; $q_i = 0$ for n+1 $\leq$ i $\leq$ 2n-1
$\Rightarrow A(x) = q_1 + q_2\ x + q_3\ x^2 + ... + q_n\ x^{n-1} + 0x^n + ... + 0x^{2n-}2$
Let D(x) = A(x).B(x)
Let D_i be the coefficient of x_i in D(x)
Let A_i be the coefficient of x_i in A(x)
$\Rightarrow A_i = q_{i+1}$
Let B_i be the coefficient of x_i in B(x)

$$\Rightarrow B_i = \begin{cases} \frac{-1}{((n-1)-i)^2}, & 0 \leq i \leq n-2 \\ 0, & i = n-1 \\ \frac{1}{(i-(n-1))^2}, & n \leq i \leq 2n-2 \end{cases}$$

Claim: $\frac{F_j}{cq_j} = D_{j+n-2}$
Proof:
$D_{j+n-2} = A_{j+n-2}.B_0 + A_{j+n-3}.B_1 + ... + A_{j-1}.B_{n-1} + ... + A_0.B_{j+n-2}$
From the previous expressions of A_i and B_i
$D_{j+n-2} = q_{j+n-1}\frac{-1}{(n-1)^2} + q_{j+n-2}\frac{-1}{(n-2)^2} + ... + q_j(0) + ... + q_1\frac{1}{(j-1)^2}$
We know that $q_i = 0$, if i $> n$
$\Rightarrow D_{j+n-2} = q_n\frac{-1}{(n-j)^2} + q_{n-1}\frac{-1}{(n-1-j)^2} + ... + q_j(0) + ...\ q_1\frac{1}{(j-1)^2}$
$\Rightarrow D_{j+n-2} = \sum_{i<j} \frac{Cq_i}{(j-i)^2}$ - $\sum_{i>j} \frac{Cq_i}{(j-i)^2}$, $1 \leq i \leq n$ and i $\neq$ j
$\Rightarrow D_{j+n-2} = \frac{F_j}{cq_j}$
Claim is true

Now we need to find D_{j+n-2} for $1 \leq j \leq n$, i.e., we need D_{n-1}, D_n, ..., D_{2n-2}
We can compute coefficients of D using polynomial multiplication.
We compute DFT(A(x)), DFT(B(x)) using FFT Algorithm
let A',B' be at DFT(A(x)), DFT(B(x))
Now we compute D'[i] = A'[i].B'[i]
On D' we apply inverse DFT using FFT Algorithm. So we get Coefficients of D(x)
We can get F_j by multiplying D_{n+j-2} with Cq_j
We can get all F_j's using this algorithm

Algorithm:
```
A = (q1, q2, q3,....,qn)
B = (-1/(n-1)^2 , -1/(n-2)^2 ,.....,-1,0,1,....., 1/(n-2)^2 ,1/(n-1)^2)
A' = DFT(A)
B' = DFT(B)
for i from 0 to length(A'):
      D'[i] = A'[i]*B'[i]
D = InverseDFT(D')
```

```
for j from 1 to n:
        F_j = D_{n+j-2}*C*q_j
        Output F_j
```

Note: Algorithms DFT and InverseDFT are discussed in class
Proof: Proof of FFT is discussed in class and proof of relation between coefficients of D(x) and F_j is given above.

Time Complexity:
1)Forming Polynomials A(x), B(x) takes O(n) time (They are of length 2n - 1 = O(n))
2)Forming Polynomials A'(x), B'(x) takes O(nlogn) time (FFT takes O(nlogn))
3)Finding D' takes O(n) time.
4)Finding D from D' takes O(nlogn) time(InverseDFT uses FFT algorithm which takes O(nlogn))

Total time complexity = O(nlogn)

3 Distance computation using Matrix Multiplication

Given G = (V,E) an unweighted, undirected graph
H = (V,E_H) be an undirected graph such that (x,y) ∈ E_H if and only if (x,y) ∈ E or ∃ w ∈ V such that (x,w), (w,y) ∈ E

3.a Part a

1) If (x,y) ∈ E then (x,y) ∈ E_H This can be done in O(m) = O(n^2) time (m = |E|)

2) For x,y in G such that (x,w) (w,y) ∈ E then (x,y) ∈ E $_H$
So, we need to find x,y in G such that there exists a path of length exactly 2 from x to y

Let A be the adjacency matrix of G.From Transitive closure(Discussed in class) if A^2(i,j) > 0 iff ∃ a path of length 2 from i to j. So we compute A^2 , and then find(i,j) such that A^2(i,j)>0 and add (i,j) to E_H.

Computing A^2 takes O(n^ω) ($\omega > 2$) and finding all (i,j) pairs take O(n^2)
So total **time complexity** = O(n^ω)
So we can construct H in O(n^ω) time

3.b Part b

Claim : For any (x,y), D_H(x,y) = $\lceil \frac{D_G(x,y)}{2} \rceil$
Proof:
If (x,y) are disconnected in G then they are disconnected in H ⇒ D_H(x,y) = D_G(x,y) = 0 .
So claim is true in this case.
let P be the shortest path from x to y in G
let P = xv_1v_2........v_ny

Case 1 - n is odd:
Let n = 2k+1 is odd ⇒ P = xv_1v_2........v_{2k+1}y ⇒ D_G(x,y) = 2k+2
From definition of H, (x,v_2) ∈ E_H as (x, v_1), (v_1, v_2) ∈ E
Similarly, (v_2, v_4) ∈ E_H ..... (v_{2k-2}, v_{2k}) ∈ E_H and (v_{2k}, y) ∈ E_H
So P' = xv_2v_4........v_{2k}y is shortest path from x to y in H
⇒ D_H(x,y) = k + 1
⇒ D_H(x,y) = D_G(x,y)/2
⇒ D_H(x,y) = $\lceil \frac{D_G(x,y)}{2} \rceil$
So, Claim is True in this case

Case 2 - n is even:
let n = 2k is even ⇒ P = xv_1v_2........v_{2k}y ⇒ D_G(x,y) = 2k+1
From the definition of H (x, v_2) ∈ E_H
Similarly, (v_2, v_4) ∈ E_H ..... (v_{2k-2}, v_{2k}) ∈ E_H
So P' = xv_2v_4........v_{2k}y is the shortest path from x to y in H
⇒ D_H(x,y) = k + 1
⇒ D_H(x,y) = (D_G(x,y)+1)/2 = D_G(x,y)/2 + 1/2
⇒ D_H(x,y) = $\lceil \frac{D_G(x,y)}{2} \rceil$
So,claim is True

3.c Part c

M = D_H * A_G
M(i,j) = i^{th} row of D_H * j^{th} column of A_G
M(i,j)= $\sum_{w=1}^{n} D_H(i,w) + A_G(i,w)$ D_H(i,w) gives distance of vertex w from i

$A_{\mathrm{G}}(w,j) = \begin{cases} 1, & \text{if w is neighbour of j in G} \\ 0, & \text{otherwise} \end{cases}$

M(i,j) = $\sum_{\substack{w\in \\ neighbours \\ ofjinG}} D_{\mathrm{H}}(i,w)$

$\Rightarrow$ M(x,y) = $\sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,w)$

From part(b),we can say that

$D_{\mathrm{G}}(x,y) = \begin{cases} 2D_{\mathrm{H}}(x,y), & \text{if } \mathrm{D_G(x,y)} \text{ is even} \\ 2D_{\mathrm{H}}(x,y)-1, & \text{if } \mathrm{D_G(x,y)} \text{ is odd} \end{cases}$

Case 1: $\mathrm{D_G}$(x,y) = $2\mathrm{D_H}$(x,y)
let $\mathrm{D_G}$(x,y) = 2k $\Rightarrow$ $\mathrm{D_H}$(x,y) = k
let w be the neighbours of y in G

Claim: 2k - 1 $\leq$ $\mathrm{D_G}$(x,w) $\leq$ 2k + 1
Reason: If $\mathrm{D_G}$(x,w) < 2k - 1, there exists another shortest path from x to y whose length is less than 2k, which is a contradiction. The maximum distance from x to w is 2k + 1 because there exits a path from x to y of length 2k and w is a neighbour of y.

From part b, k $\leq$ $\mathrm{D_H}$(x,w) $\leq$ k + 1
$\mathrm{D_H}$(x,w) $\geq$ $\mathrm{D_H}$(x,y) for every w neighbour of y in G

$\Rightarrow \sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,w) \geq \sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,y)$

$\Rightarrow$ M(x,y) $\geq \sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,y)$

$\Rightarrow$ M(x,y) $\geq$ $\mathrm{deg_G}$(y).$\mathrm{D_H}$(x,y)
So, $\mathrm{D_G}$(x,y) = $2\mathrm{D_H}$(x,y) if M(x,y) $\geq$ $\mathrm{deg_G}$(y).$\mathrm{D_H}$(x,y)

Case 2: $\mathrm{D_G}$(x,y) = $2\mathrm{D_H}$(x,y) - 1
let $\mathrm{D_H}$(x,y) = k + 1 $\Rightarrow$ $\mathrm{D_G}$(x,y) = 2k+1
let w be neighbour of y in G

Claim: 2k $\leq$ $\mathrm{D_G}$(x,w) $\leq$ 2k + 2
Reason: If $\mathrm{D_G}$(x,w) < 2k, there exists another shortest path from x to y whose length is less than 2k + 1, which is a contradiction. The maximum distance from x to w is 2k + 2 because there exits a path from x to y of length 2k + 1 and w is a neighbour of y.

From part b, k $\leq$ $\mathrm{D_H}$(x,w) $\leq$ k+1
$\mathrm{D_H}$(x,w) $\leq$ $\mathrm{D_H}$(x,y) for every neighbour w of y in G

$\Rightarrow \sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,w) < \sum_{\substack{w\in \\ neighbours \\ ofyinG}} D_{\mathrm{H}}(x,y)$

(for w lying in path from x to y, $\mathrm{D_H}$(x, w) = k. So, $\mathrm{D_H}$(x, w) < $\mathrm{D_H}$(x,y))

$\Rightarrow$ M(x,y) < $\mathrm{deg_G}$(y).$\mathrm{D_H}$(x,y)
So, $\mathrm{D_G}$(x,y) = $2\mathrm{D_H}$(x,y) - 1 if M(x,y) < $\mathrm{deg_G}$(y).$\mathrm{D_H}$(x,y)

3.d Part d

We are given D_H
For all x,y we can compute $deg_G(y)D_H(x,y)$ in $O(n^2)$ time
$M = D_H * A_G$ Can be computed in $O(n^\omega)$(Matrix multiplication)
Now we have M
So For all x,y $D_G(x,y)$ can be computed in $O(n_\omega)$ time using $D_H(x,y)$, $M(x,y)$, $deg_G(y)$
So total time = $O(n_\omega)$

3.e Part e

Algorithm Description:
We are given a graph G.We need to compute D_G.We recursively compute graph H
In each recursive call,diameter of G halves(from b)
let d = diameter of G,so after logd calls, we are left with graph where distances of all pairs of vertices is 1.In this graph we can compute all the distances which is same as adjacency matrix of that graph
Now, we have D_H. So from part-d we can compute D_G from D_H.

Algorithm

```
def all-pair-distances(A):
     if all entries of A = 1 except diagonal then
          return A
     # If (i, j)th entry of (I + A)^2 is greater than 0, then (i, j) are adjacent
in      # squared graph.  Adjacency graph can be computed in this way.
     compute A_H Adjacency matrix of squared graph
     D_H = all-pairs-distances(A_H)
     M = D_H * A
     D_G = distance matrix of G
     for each x,y in V:
          if M(x,y) > deg(y)D_H then:
               D_G = 2D_H
          else:
               D_G = 2D_H-1
     return D_G
```

Time Complexity: let d be the diameter of G, n be the no of vertices
1) Checking entries in A take $O(n^2)$ time
2) Computing A_H take $O(n^\omega)$ time
3) Computing M take $O(n^\omega)$ time
4) Computing D_G take $O(n^2)$ time (given M,D_H)

So each call takes $O(n^\omega)$ time

So, $T(n,d) = T(n,d/2) + O(n^\omega)$(diameter halves in each step) = $O(n^\omega \log d)$
$d = O(n)$
So $T(n,d) = O(n^\omega \log n)$
So we can compute all pair distances in $O(n^\omega \log n)$ time.

4 Universal Hashing

U = [0,M-1] is a universe of M elements.
p is a prime number in range(M,2M)
n < < m
H(x) = (x) mod n
H_r(x) = ((rx) mod p) mod n

4.a Part a

S is a random set
let x ∈ S and as x is selected in random from U.
P(H(x) = i) = P(x mod n = i) = $\frac{i,n+i,2n+i,...,[M/n](n-1)+i}{M}$ = $\frac{1}{n}$
So, P(H(x) = i) = $\frac{1}{n}$
We need to find P(max-chain length in hashtable of S > logn)
let k = logn
Suppose if we take S', a fixed of S containing k elements , then P(H(x) = i , ∀ x ∈ S') = $(\frac{1}{n})^k$
There are $\binom{n}{k}$ subsets of S with k elements.
We need to find probability that any one of these subsets hashes to i

i.e; $P[\bigcup_{\substack{S' in subsets \\ of S of size k}} (H(x) = i, \forall x \in S')]$
Applying union bound
$P[\bigcup_{\substack{S' in subsets \\ of S of size k}} (H(x) = i, \forall x \in S')] \leq \sum_{\substack{S' in subsets \\ of S of size k}} P(H(x) = i, \forall x \in S') = \binom{n}{k} (\frac{1}{n})^k$

$\binom{n}{k} (\frac{1}{n})^k = \frac{n!}{k!(n-k)!} (\frac{1}{n})^k = \frac{n^k}{k!} (\frac{1}{n})^k = \frac{1}{k!} \approx \frac{(e)^k}{(k)^k} = (e^k).(\frac{1}{logn})^{logn} = n^{log(e)}.(\frac{1}{logn})^{logn} = n^{log(e)}.\frac{1}{n^{log(logn)}}$
So, $\binom{n}{k} (\frac{1}{n})_k \approx n^{log(e)}.\frac{1}{n^{log(logn)}}$

So, $P[\bigcup_{\substack{S' in subsets \\ of S of size k}} (H(x) = i, x \in S')] \leq n^{log(e)}.\frac{1}{n^{log(logn)}}$

i can be any value in between 0 to $n-1$.

So, $\sum_{i=0}^{n-1} P[\bigcup_{\substack{S' in subsets \\ of S of size k}} (H(x) = i, x \in S')] = n\, P[\bigcup_{\substack{S' in subsets \\ of S of size k}} (H(x) = i, x \in S')] \leq n^{log(e)+1}.\frac{1}{n^{log(logn)}}$
Lets assume that n ≥ 2^{4e}. So, $n^{log(e)+1}.\frac{1}{n^{log(logn)}} \leq \frac{1}{n}$
Probability that any location gets logn elements is less than $\frac{1}{n}$
So probability that any location gets more than logn elements is less than $\frac{1}{n}$.
So P(any location gets more than logn elements) ≤ $\frac{1}{n}$.

⇒ P(max chain length ≥ logn) ≤ $\frac{1}{n}$ for n ≥ 2^{4e}.

4.b Part b

Not attempted

4.c Part c

Plot of Max-chain-length for hash functions H():

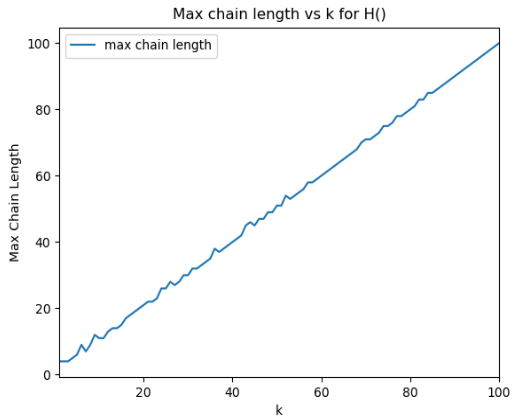

Figure 2: Plot of Max-chain-length for hash functions H()

Justifications:

1. For every set S_k, we are taking k numbers for which x mod n = 0 so as k increases chain length for location 0 increases linearly. And we consider (n - k) elements from random and their probability to get hashed to a location is 1/n. So, expected chain length of all locations except 0 is (n - k)/n and expected chain length of k + (n - k)/n. So, expected max chain length is k + (n - k)/n. This increase linearly with k. This can be observed from the plot.

Plot of Max-chain-length for hash functions Hr():

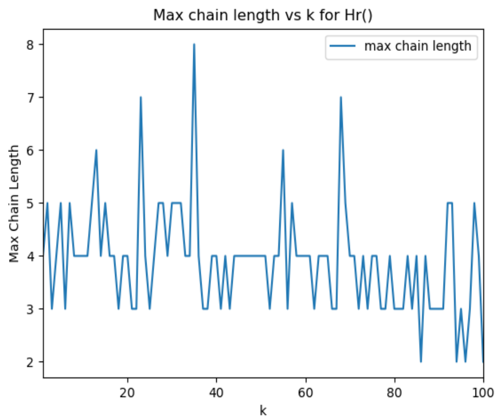

Figure 3: Plot of Max-chain-length for hash functions Hr()

Justifications:

1. We used p = 12497 which lies between [M, 2M]. For each S_k, we are picking a random number r. For k fixed numbers we pick for each set, (rx) mod p randomizes these numbers, so ((rx) mod p) mod n will be random. And of course for n - k random numbers, ((rx) mod p) mod n will be random. So, expected value of chain length of location i is 1 for all i. Maximum chain length depends on hash of random numbers. So, there won't be any pattern in Max-chain-length vs k and this can be observed from graphs.

4.d Code

The code file is uploaded in this drive link. Running file using command "python3 hashing.py" create two plots "plot_h.png" and "plot_hr.png" corresponding to hash functions H() and Hr() respectively.