COL351 Fall 2021 Assignment 3

Saptarshi Dasgupta, Tamajit Banerjee

TOTAL POINTS

60 / 60

QUESTION 1

1 Convex Hull 15 / 15

- **1 pts** Observation for qualification of a line to be a boundary of convex hull.
- **1 pts** Algorithm for Convex Hull
- **0.5 pts** Lower and Upper Tangents, Algo to find them
- **0.5 pts** Time Complexity
- **3 pts** Proof of Correctness

✓ **- 0 pts** **All Correct**

- **1 pts** How to find all points lie above or below/ CW,ACW., crosses polygon.

Click here to replace this description.

- **0 pts** Click here to replace this description.

QUESTION 2

2 Particle Interaction 15 / 15

+ **6 pts** Polynomial Equation and Algorithm
+ **7 pts** Proof of correctness
+ **2 pts** Time complexity
+ **0 pts** not attempted

+ 15 Point adjustment

QUESTION 3

3 Distances 15 / 15

✓ **+ 2 pts** **Part a**

✓ **+ 2 pts** **Part b**

Part C

+ **4 pts** Partially correct
+ **6 pts** Proved converse of statement

✓ **+ 8 pts** **Correct**

✓ **+ 1 pts** **Part d**

✓ **+ 2 pts** **Part e**

+ **0 pts** Unattempted/Incorrect

QUESTION 4

4 Hashing 15 / 15

Part a

✓ **+ 4 pts** **Correct analysis**

+ **3 pts** Mistakes in the analysis

✓ **+ 3 pts** **Correct Conclusion**

+ **2 pts** Mistakes in the observation
+ **0 pts** Not attempted

Part b

✓ **+ 3 pts** **Correct Proof**

+ **0 pts** Not attempted

Part c

✓ **+ 3 pts** **Plot**

✓ **+ 2 pts** **Justification**

+ **0 pts** Not attempted

+ **0 pts** Click here to replace this description.
+ **0 pts** Not attempted

COL351-A3

Saptarshi Dasgupta (2019CS50447)
Tamajit Bannerjee (2019CS10408)

October 2021

1 Problem 1: Convex Hull

Q. The Convex Hull of a set P of n points in X-Y plane is a minimum subset Q of points in P such that all points in P can be generated by a convex combination of points in Q. In other words, the points in Q are corners of the convex-polygon of smallest area that encloses all the points in P. Design an $\mathcal{O}(n\ logn)$ time Divide-and-Conquer algorithm to compute the convex hull of a set P of n input points - $\{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$.

Ans.

Assumptions : We assume that no points are repeated if they are then we can remove them initially.

We also assume that no 3 points are collinear, if they are collinear then can collect the points which are collinear among themselves and keep only the ones with the highest x coordinate and lowest x coordinate or else break tries using y coordinate. This guarantees the correct convex Hull because points that are in the interior of a collinear segment must lie within the convex Hull.

In the algorithm , we have always maintained the points in the convex hull in counter-clockwise order. So , if we are in point a , then a + 1 means the next point in counter-clockwise order and a - 1 means the next point in clockwise order.

We prove the correctness of the above algorithm with the folllowing lemmas. We first prove that the function UpperTangent returns the upper tangent to H_1 and H_2. The correctness of the LowerTangent function is proved analogously.

The algorithm is presented in the next page.

We tell a line (say ab) is the **upper tangent** at a point (a) or not : by computing the line by taking lines ((a+1)b and (a-1)b) and showing that they are below the line (ab).

We tell a line (say ab) is the **lower tangent** at a point (a) or not : by computing the line by taking lines ((a+1), b and (a-1), b) and showing that they are above the line (ab).

Algorithm 1: Convex Hull Finding Algorithm

Data: List n points $\{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$. in increasing order of X-coordinates, if not we sort them before feeding to the algorithm.
Result: List of points in the boundary of the convex Hull in order
Function `UpperTangent(`H_1, H_2`):`
  $x \leftarrow$ Rightmost point of H_1;
  $y \leftarrow$ Leftmost point H_2;
  $T \leftarrow$ Line xy;
  while T *is not Upper Tangent to both* H_1 *and* H_2 **do**
    $T \leftarrow$ *Line* xy;
    while T *is not upper tangent to* H_1 **do**
      $x \leftarrow x + 1$;
    end
    while T *is not upper tangent to* H_2 **do**
      $y \leftarrow y - 1$;
    end
  end
  return T
Function `LowerTangent(`H_1, H_2`):`
  $x \leftarrow$ Rightmost point of H_1;
  $y \leftarrow$ Leftmost point H_2;
  $T \leftarrow$ Line xy;
  while T *is not Lower Tangent to both* H_1 *and* H_2 **do**
    $T \leftarrow$ *Line* xy;
    while T *is not Lower tangent to* H_1 **do**
      $x \leftarrow x - 1$;
    end
    while T *is not Lower tangent to* H_2 **do**
      $y \leftarrow y + 1$;
    end
  end
  return T
Function `ConvexHull(`P`):`
  if $P.size() \leq 2$ **then**
    return P;
  end
  if $P.size() == 3$ **then**
    if $P[1].y > P[0].y$ **then**
      return P;
    end
    else
      swap($P[1], P[2]$);
      return P;
    end
  end
  $P_1 \leftarrow$ first $\lceil \frac{n}{2} \rceil$ of P;
  $P_2 \leftarrow$ remaining $\lfloor \frac{n}{2} \rfloor$ of P;
  if $P_1[last\_vertex].x == P_2[first\_vertex].x$ **then**
    P_2.add(P_1[last_vertex]);
    P_1.remove(P_1[last_vertex]);
  end
  $H_1 \leftarrow$ ConvexHull(P_1);
  $H_2 \leftarrow$ ConvexHull(P_2);
  $H \leftarrow$ Merge(H_1, H_2);
  return H;
Function `Merge(`H_1, H_2`):`
  a_1, $b_1 \leftarrow$ UpperTangent(H_1, H_2);
  a_2, $b_2 \leftarrow$ LowerTangent(H_1, H_2);
  $H_1 \leftarrow$ Vertices of H_1 from a_1 to a_2 in Counter Clockwise order.;
  $H_2 \leftarrow$ Vertices of H_2 from b_2 to b_1 in Counter Clockwise order.;
  return H_1 *CONCAT* H_2;

Claim 1. *The upper tangent $T = xy$ touches both H_1 and H_2 on their upper halves.*

Proof. P_1 and P_2 are divided from P according to their X coordinates. This implies that there will be a line L which will be the vertical separator of A and B.

There are 2 extremes. B is located very high above A which will imply that x will approach the leftmost point of A whereas if B is located very low below A which will imply that x will approach the rightmost point of A. Between these extremes, x lies on the upper half.

Similarly, for B.

□

Claim 2. *For all iterations of the algorithm, $T = xy$ is does not pass through the interior of P_1 or P_2.*

Proof. The proof is by induction on the interations of the algorithm.

Base Case : At the start of the algorithm, we have x to be the rightmost point of P_1 and y to be the leftmost point of P_2, so the line xy lies outside of both P_1 and P_2.

Induction Hypothesis : Let at iteration i, the statement be true. Thus, line T_i lies ourside of both P_1 and P_2.

Induction Step : Assume by the method of contradiction that for the next iteration, line T_{i+1} does pass through the interior of P_1 (Without loss of generality we can assume P_1.).

Since T_{i+1} passes through P_1, it x must have been incremented to $(x+1)$, in this iteration, which can only happen if xy is not the upper tangent of P_1. The new tangent $T_{i+1} = (x+1, y)$ could only intersects the interior of P_1 if and only if y is to the right of the line $(x+1, x)$ but P_2 could also not be to the right of the line $(x, x-1)$ for then xy would have intersected P_1 which is not possible due to the induction hypothesis. So if xy intersects P_1 in the next step then it will definitely be the upper tangent and the next step will not be taken.

Hence we arrive at a contadiction. This implies that line T_{i+1} does not pass through the interior of P_1. Following a similar logic, we can also arrive at the same conclusion corresponding to P_2. Thus we have proved the Induction Step. □

Claim 3. *The algorithm **Upper Tangent** must terminate and yield a upper tangent as T*

Proof. By **Claim 3**, we know , that xy does not meet the interior of P_1 and P_2 and as y is in the right of L we can conclude that x will never move counterclockwise beyond the leftmost point. as that would imply that T intersects the interior of P_1 which is not allowed. Similarly, y can never move clockwise beyond the rightmost point.

But there are only a finite number of vertices in each set and we only visit each vertices once as we either always increment or always decrement x and y depending on whether we want to find the lower or upper tangent. Furthermore as we just showed, x cannot reach beyond the leftmost point of P_1 and y the rightmost point of P_2. So, our algorithm must terminate.

In addition, from the termination condition of our outer loop in function Upper Tangent, we have that both x and y must be Upper tangent to P_1, P_2.

□

Now we have proved that the correctness of UpperTangent and LowerTangent functions. Now we will prove the correctness of the Merge function.

Claim 4. *If sets P_1 and P_2 have $H_1 = \{x_1, x_2, \cdots x_{k_1}\}$ and $H_2 = \{y_1, y_2, \cdots y_{k_2}\}$ as their convex Hulls with points in counter clockwise order and if the upper tangent for the 2 convex hulls is at points (x_{p_1}, y_{q_1}) and lower tangent is at points (x_{p_2}, y_{q_2}), then the Convex Hull H of P in counter clockwise order is given by*

$$H = \{x_{p_1}, x_{p_1+1}, \cdots, x_{p_2}\ , y_{q_2}, y_{q_2+1}, \cdots y_{q_1}\}$$

Proof. From definition of Upper tangent we have that all points of P_1 and P_2 must lie below the upper tangent, similarly all points lie above the lower tangent. So, for each point x of P_1, it was in the interior of H_1. Since the tangents never intersect the interior of H_1, x must also be in the interior of H. Similarly, for points b of P_2 we prove that they must lie in the interior of P_2. Thus H contains all the points of P in its interior.

To prove that H is convex, we show that all its internal angles are less that 180 degrees. We only need to check for those angles that have either of the common tangents as one of their sides, as the other angles are present in either H_1 or in H_2 and are therefore all less than 180 degrees. For the angles involving the tangents, consider the angle $(x_{p_1+1}, x_{p_1}, y_{q_1})$. Since, x_{p_1}, y_{q_1} is the upper tangent, x_{p_1+1} is below this tangent and hence this angle is less that 180 degrees. Similarly for the other, angles we can show that they are less than 180 degrees each.

Thus we have proved that H is a convex Hull of P □

Claim 5. *The time complexity of the algorithm is $\mathcal{O}(n\ logn)$*

Proof. The complexity of sorting the points into 2 parts takes $\mathcal{O}(n\ logn)$ time which is done only once before the recursive call to Convex Hull.

For the merging operation, we have that the complexity of finding upper tangent is $\mathcal{O}(n)$ as in each iteration we either increment x or decrement y, and they cannot overshoot the leftmost point of H_1 and the rightmost point of H_2 respectively. This ensures that we have atmost $\mathcal{O}(n)$ increment/decrements before we end up at the tangent. The same is true for the upper tangent. Now, constructing the convex hull also takes linear time, as we just need to iterate over the convex hulls of the subsets. So, overall merging has conplexity $\mathcal{O}(n)$.

So, assuming that the complexity of finding the convex Hull is $T(n)$ for n points, we get the recursion

$$T(n) = 2 * T(\left\lceil \frac{n}{2} \right\rceil) + \mathcal{O}(n)$$

.

So, on applying the Master theorem, we get the complexity $T(n) = \mathcal{O}(n\ logn)$. □

The space complexity of the algorithm is $\mathcal{O}(n\ logn)$

1 Convex Hull **15 / 15**

- **1 pts** Observation for qualification of a line to be a boundary of convex hull.
- **1 pts** Algorithm for Convex Hull
- **0.5 pts** Lower and Upper Tangents, Algo to find them
- **0.5 pts** Time Complexity
- **3 pts** Proof of Correctness

✓ **- 0 pts All Correct**

- **1 pts** How to find all points lie above or below/ CW,ACW., crosses polygon.

Click here to replace this description.

- **0 pts** Click here to replace this description.

2 Problem 2 : Particle Intaraction

Q. We can model their structure of a number of charged particles on the X-axis as consisting of the points $\{1, 2, 3, \cdots, n\}$ on the real line; and at each of these points j, they have a particle with charge q_j . (Each charge can be either positive or negative.) They want to study the total force on each particle, by measuring it and then comparing it to a computational prediction. This computational part is where they need your help. The total net force on particle j, by Coulomb's Law, is equal to

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

. Write a program to compute the values of F_j for all j in $\mathcal{O}(n\ logn)$ time.

Ans. Consider the 3 polynomials (where $|x|$ denotes the absolute value of x)

$$A(x) = \sum_{i=0}^{n-1} q_{i+1}x^i,$$

$$B(x) = \sum_{i=0, i\neq n-1}^{2n-2} \frac{|i-(n-1)|}{(i-(n-1))^3}x^i,$$

$$R(x) = C * A(x) * B(x) = \sum_{i=0}^{3n-3} R_i x^i$$

.

Claim 1. $\forall\ i = 1\ to\ n,\ F_i x^{i+n-2} = C\ *\ q_i\ *\ R_{i+n-2}$

Proof ::
We know that as $R(x) = A(x) * B(x)$, coeff of x^i in $R(x)$

$$R[i]x^i = C\sum_{j=0}^{n} A_j x^j * B_{n-j}x^{i-j},\ \text{if degree of B is } n$$

$$\implies R[i+n-2]x^{i+n-2} = C \sum_{j=0, j\neq i-1}^{n-1} \left(q_{j+1}x^j * \frac{|i+n-2-j-(n-1)|}{(i+n-2-j-(n-1))^3}x^{i+n-2-j} \right)$$

$$R[i+n-2]x^{i+n-2} = C \sum_{j=0, j\neq i-1}^{n-1} \left(q_{j+1} * \frac{|i-j-1|}{(i-j-1)^3}x^{i+n-2} \right)$$

$$\implies F_i/q_i x^{i+n-2} = C \sum_{j=1, j\neq i}^{n} \left(q_j * \frac{|i-j|}{(i-j)^3}x^{i+n-2} \right)$$

$$\implies F_j/q_j x^{i+n-2} = C \sum_{i=1, i\neq j}^{n} \left(q_i * \frac{|j-i|}{(j-i)^3}x^{j+n-2} \right)\ \text{(interchanging i and j)}$$

$$\implies F_j/q_j x^{j+n-2} = C\sum_{i<j}^{n} \left(q_i * \frac{|j-i|}{(j-i)^3}x^{j+n-2} \right) + C\sum_{i>j}^{n} \left(q_i * \frac{|j-i|}{(j-i)^3}x^{j+n-2} \right)$$

$$\implies F_j/q_j x^{j+n-2} = C\sum_{i<j} \left(\frac{q_i}{(j-i)^2} \right) x^{j+n-2} - C\sum_{i>j} \left(\frac{q_i}{(j-i)^2} \right) x^{j+n-2}$$

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

Algorithm 2: Force Computation Algorithm

Data: List Q of charges of n particles on the X-axis
Result: List of forces F_j on charge j for $1 \leq j \leq n$
Function `PolynomialMultiplication(`*List A, List B*`):`
  ; `/* Standard Implementation using FFT taught in class */`
Function `CalculateForces(`*List Q*`):`
  List $A \leftarrow []$;
  List $B \leftarrow []$;
  for i *in* $[0, n-1]$ **do**
    $A[i] \leftarrow Q[i+1]$;
  end
  for i *in* $[0, 2n-2]$ **do**
    if $i \neq n-1$ **then**
      $B[i] \leftarrow \frac{|i-(n-1)|}{(i-(n-1))^3}$;
    end
  end
  $B[n-1] \leftarrow 0$;
  $PolynomialR \leftarrow$ C * PolynomialMultiplication(A,B);
  List $F \leftarrow []$;
  for i *in* $[1, n]$ **do**
    $F[i] \leftarrow Q[i] * R[i+n-2]$;
  end
  return F

Claim 2. *Both the time and space complexity of the above algorithm is O(n logn)*

Proof. Computing coeffecients of polynomials $A(x)$ and $B(x)$ take $\mathcal{O}(n)$ time as they are both a single for loop that runs n times. Computing the product of the 2 polynomials using FFT and inverse FFT takes $\mathcal{O}(n\ logn)$ time as taught in class. Finally computing the forces again takes only $\mathcal{O}(n)$ time as it is again a for loop with $n-1$ runs. So, the final time complexity is O(n logn)

For the space complexity, we create 4 new arrays which result in a space complexity of $\mathcal{O}(n)$. But the FFT algorithm has a space complexity of $\mathcal{O}(n\ logn)$. Thus the total space complexity is $\mathcal{O}(n)$. □

2 Particle Interaction **15 / 15**

+ **6 pts** Polynomial Equation and Algorithm
+ **7 pts** Proof of correctness
+ **2 pts** Time complexity
+ **0 pts** not attempted

+ 15 Point adjustment

3 Problem 3 : Distance computation using Matrix Multiplication

Q. Let G = (V, E) be an unweighted undirected graph, and $H = (V, E_H)$ be a undirected graph obtained from G that satisfy: $(x, y) \in E_H$ if and only if $(x, y) \in E$ or there exists a $w \in V$ such that $(x, w), (w, y) \in E$. Further, let D_G denote the distance-matrix of G, and D_H be the distance-matrix of H.

a) Prove that the graph $H = (V, E_H)$ can be computed from G in $O(n^\omega)$ time, where ω is the exponent of matrix-multiplication.

Ans. Let the adjacency matrix of graph H be A_H. adjacency matrix of G be A

We will prove that $(i, j) \in E_H$ if and only if $(A + A^2)_{i,j} > 0$

First assume $(i, j) \in E_H$
$\implies$ $(i, j) \in E$ or $\exists k$, $(i, k) \in E$ and $(k, j) \in E$. (by definition of $A_{Hi,j}$)
$\implies$ $A_{i,j} > 0$ or $\exists k$, $A_{i,k} = 1$ and $A_{Hk,j} = 1$.
$\implies$ $A_{i,j} > 0$ or $\exists k$, $A_{i,k} * A_{k,j} = 1$, (as A is adjacency matrix and its values are either 0 or 1)
$\implies$ $A_{i,j} > 0$ or $\exists k$, $A_{i,k} * A_{k,j} > 0$, ()
$\implies$ $A_{i,j} > 0$ or $\sum_{k=0}^{n} A_{i,k} * A_{k,j} > 0$ (each term of sum ≥ 0, and atleast one is $= 1$)
$\implies$ $A_{i,j} + \sum_{k=0}^{n} A_{i,k} * A_{k,j} > 0$
$\implies$ $(A + A^2)_{i,j} > 0$
Thus we have proved one side of the if and only if claim.

Next assume $(A + A^2)_{i,j} > 0$
$\implies$ $(A + A^2)_{i,j} > 0$
$\implies$ $A_{i,j} + \sum_{k=0}^{n} A_{Hi,k} * A_{Hk,j} > 0$
$\implies$ $A_{i,j} > 0$ or $\sum_{k=0}^{n} A_{i,k} * A_{k,j} > 0$ (sum of 2 non-negative terms > 0, so atleast one > 0)
$\implies$ $A_{i,j} > 0$ or $\exists k$, $A_{i,k} = 1$ and $A_{k,j} = 1$, (sum is positive, so atleast one term must be positive, thus equal to 1))
$\implies$ $(i, j) \in E$ or $\exists k$, $(i, k) \in E$ and $(k, j) \in E$. (by definition of $A_{i,j}$)
$\implies$ $(i, j) \in E_H$. (by definition of graph H)
Thus we have proved both sides of the claim.

Now to compute $A + A^2$ we need to multiply 2 matrices of size $n \times n$ which takes $\mathcal{O}(n^\omega)$ time and then add the result to A which takes $\mathcal{O}(n^2)$ time. So total time taken to compute the matrix is $\mathcal{O}(n^\omega)$ since $\omega \geq 2$.

b) Argue that for any $x, y \in V$, $D_H(x,y) = \left\lceil \frac{D_G(x,y)}{2} \right\rceil$.

Ans.

Claim 1. $D_H(x,y) \leq \left\lceil \frac{D_G(x,y)}{2} \right\rceil$

Proof. Consider the shortest path from x to y of length $D_G(x,y)$ in graph G.
Let that path be $v_1, v_2, v_3, ..., v_{D_G+1}$, where $v_1 = x$ and $v_{D_G+1} = y$. Now, for graph H, we can construct a path from x to y as follows

$$path(x,y) = \begin{cases} \{v_1, v_3, ...y\} \text{ of length } \frac{D_G(x,y)+1}{2}, & \text{when } D_G(x,y) \bmod 2 = 1, \\ \{v_1, v_4, ...y\} \text{ of length } \frac{D_G(x,y)}{2}, & \text{when } D_G(x,y) \bmod 2 = 0. \end{cases}$$

Thus in both cases, a path from x to y of length $\left\lceil \frac{D_G(x,y)}{2} \right\rceil$ exists in H. So, the distance between must be atmost that length. Thus,

$$D_H(x,y) \leq \left\lceil \frac{D_G(x,y)}{2} \right\rceil$$

□

Claim 2. $D_H(x,y) \geq \frac{D_G(x,y)}{2}$

Proof. Consider the path between x and y in graph H of length $l = D_H(x,y)$.
Let the path be $\{v_1, v_2, v_3, ..., v_{l+1}\}$. where $v_1 = x$ and $v_{l+1} = y$. We will contruct a path between x and y in G. Consider any pair of vertices v_i, v_{i+1} on the path for $i > 1$. Since, the edge e connecting these vertices lies in H, we have 2 cases -
Case 1 - $e \in E$:
Add vertex v_{i+1} to path after vertex v_i.
Case 2 - $\exists\ c_i$, such that $(v_i, c_i) \in E$ and $(c_i, v_{i+1}) \in E$
Add vertices c_i, v_{i+1} to path in that order after vertex v_i.

Thus for each v_i in original path after x we are adding atmost 2 new vertices to our path. So the length of the path is atmost twice the number of vertices in the original path other that x, which is $2*D_H(x,y)$. So, distance from x to y in G must be less than this length. So, $D_G(x,y) \leq 2*D_H(x,y)$. Thus

$$D_H(x,y) \geq \frac{D_G(x,y)}{2}$$

. □

We have when $\frac{p}{2}$ is even, $\frac{p}{2} = \left\lceil \frac{p}{2} \right\rceil$ and when $\frac{p}{2}$ is odd, the integer just above $\frac{p}{2}$ is $\left\lceil \frac{p}{2} \right\rceil$.

Now combining the 2 claims and this observation, we get that $D_H(x,y) = \left\lceil \frac{D_G(x,y)}{2} \right\rceil$.

c) Let A_G be adjacency matrix of G, and $M = D_H * A_G$. Prove that for any $x, y \in V$, the following holds.

$$D_G(x,y) = \begin{cases} 2D_H(x,y), & \text{when } M(x,y) \geq deg_G(y) \cdot D_H(x,y), \\ 2D_H(x,y) - 1, & \text{when } M(x,y) < deg_G(y) \cdot D_H(x,y). \end{cases}$$

Ans. In the previous part we have proved that $D_H(x,y) = \left\lceil \frac{D_G(x,y)}{2} \right\rceil$.

Claim 3. *$M_{x,y}$ is the sum of distances (from x wrt graph H) of neighbours of y in G.*

Proof. From definition of M, we have $M_{x,y} = \sum_{k=1}^{n} D_H(x,k) * A_G(k,y)$. Now the non-zero terms of the sum must have $A_G(k,y) = 1$, which means that k is a neighbour of y in G. Thus the sum reduces to $\sum_{k \in N(y)} D_H(x,k)$ □

Claim 4. *The condition on M is equivalent to $D_G(x,y)$ being odd or even.*

Proof. Assume that $D_G(x,y)$ is even $(= 2k)$. Now Consider the neighbour v_1 of y in a shortest path from x to y. $D_G(x,v_1) = 2k-1$ as it is neighbour of y and on a shortest path from x to y. Thus, $D_H(x,v_1) = \left\lceil \frac{2k-1}{2} \right\rceil = k$.

Thus $M(x,y) - deg_G(y) \cdot D_H(x,y) = \{\sum_{k \in N(y)} D_H(x,k)\} - d \cdot k$

Now, for each $k \in N(y)$, $D_G(x,k) \geq D_G(x,y) - D_G(y,k) = 2k-1$ (By Triangle Inequality.)
So, $D_H(x,y) \geq \lceil 2k-1 \rceil = k$.
So, $\{\sum_{k \in N(y)} D_H(x,k)\} \geq d \cdot k$.
So, $M(x,y) - deg_G(y) \cdot D_H(x,y) \geq 0$.

$$\implies M(x,y) \geq deg_G(y) \cdot D_H(x,y), \text{ when } D_G(x,y) \text{ is even.}$$

Again assume that $D_G(x,y)$ is odd $(= 2k+1)$. Thus $D_H(x,y) = k+1$. Now Consider the neighbour v_1 of y in a shortest path from x to y. $D_G(x,v_1) = 2k$ as it is neighbour of y and on a shortest path from x to y. Thus, $D_H(x,v_1) = \left\lceil \frac{2k}{2} \right\rceil = k$.

Thus $M(x,y) - deg_G(y) \cdot D_H(x,y)$
$= \{\sum_{k \in N(y)} D_H(x,k)\} - d \cdot (k+1)$
$= \{\sum_{k \in N(y) \setminus v_1} D_H(x,k)\} + \{D_H(x,v_1) - (k+1)\} - (d-1) \cdot (k+1)$
$< \{\sum_{k \in N(y) \setminus v_1} D_H(x,k)\} - (d-1) \cdot (k+1) \qquad (D_H(x,v_1) - (k+1) = -1 < 0)$

Now, for each $k \in N(y)$, $D_G(x,k) \geq D_G(x,y) + D_G(y,k) = 2k+1+1 = 2k+2$ (By Triangle Inequality.) So, $D_H(x,y) \leq \left\lceil \frac{2k+2}{2} \right\rceil = k+1$.

Thus, $\{\sum_{k \in N(y) \setminus v_1} D_H(x,k)\} - (d-1) \cdot (k+1)$
$\leq \{\sum_{k \in N(y) \setminus v_1} (k+1)\} - (d-1) \cdot k.$
$\leq 0 \qquad (|N(y) \setminus v_1| = d-1).$

So, $M(x,y) - deg_G(y) \cdot D_H(x,y) < 0$.

$$\implies M(x,y) < deg_G(y) \cdot D_H(x,y), \text{ when } D_G(x,y) \text{ is odd.}$$

. □

We know that $D_G(x, y)$ is either $2 * D_H(x, y)$ or $2 * D_H(x, y) - 1$ according to whether $D_G(x, y)$ is odd or even. Thus we get,

$$D_G(x,y) = \begin{cases} 2D_H(x,y), & \text{when } M(x,y) \geq deg_G(y) \cdot D_H(x,y), \\ 2D_H(x,y) - 1, & \text{when } M(x,y) < deg_G(y) \cdot D_H(x,y). \end{cases}$$

d) Use (c) to argue that D_G is computable from D_H in $\mathcal{O}(n^\omega)$ time.

Ans. We can compute $M = D_H * A_G$ in $\mathcal{O}(n^\omega)$ time, as it is the multiplication of 2 matrices. Thus for each pair of vertices (x, y), we can compute $D_G(x, y)$ from $D_H(x, y)$ using part (c). which now takes $\mathcal{O}(n^2)$ time. So, the total time is $\mathcal{O}(n^\omega)$ as $\omega \geq 2$.

e) Prove that all-pairs-distances in n-vertex unweighted undirected graph can be computed in $\mathcal{O}(n^\omega logn)$ time, if ω is larger than two.

Ans.

Let the smallest power of 2 more than or equal to $n-1$ be k. Consider the sequence of graphs, $G_1, G_2, G_3, \ldots G_k$, defined on the vertices of G, where $G_1 = G$, and

$$(x, y) \in G_i \text{ iff } (x, y) \in G_{i-1} \text{ or } \exists v, (x, v), (v, y) \in G_{i-1}$$

.

Claim 5. *In graph G_i, 2 different vertices x, y are connected if $D_G(x, y) \leq 2^i$*

Proof. If $D_G(x, y) \leq 2^i$ for x, y in G, then we have by repeated application of **Part b**, $D_{G_p}(x, y) \leq 2^{i-p}$, which implies that $D_{G_i}(x, y) \leq 1$.So these vertices are connected in G_i.

From the above claim, we have that vertices of G_k are connected iff $D_{G_k}(x, y) \leq 2^k$. But $2^k > n-1$, so all vertices of G_k are connected. Thus we can obtain adjacency and distance matrix of G_k.

Now we can recursively obtain adjacency matrices of G_i from that of G_{i+1} for $i < k$, by repeated application of **Part C**., where $M_i = D_{G_{i+1}} \cdot A_{G_i}$

$$D_{G_i}(x, y) = \begin{cases} 2D_{G_{i+1}}(x, y), & \text{when } M_i(x, y) \geq deg_{G_i}(y) \cdot D_H(x, y), \\ 2D_{G_{i+1}}(x, y) - 1, & \text{when } M_i(x, y) < deg_{G_i}(y) \cdot D_H(x, y). \end{cases}$$

, By **Part d**, we have that computing each G_i takes $\mathcal{O}(n^\omega)$. Now, $2^k \geq n-1 \geq 2^{k+1}$, so, $k = \mathcal{O}(logn)$. So, total time of computing $D_{G_1}(x, y)$ is $\mathcal{O}(n^\omega logn)$

□

3 Distances 15 / 15

✓ + 2 pts **Part a**

✓ + 2 pts **Part b**

Part C

+ 4 pts Partially correct

+ 6 pts Proved converse of statement

✓ + 8 pts **Correct**

✓ + 1 pts **Part d**

✓ + 2 pts **Part e**

+ 0 pts Unattempted/Incorrect

4 Question 4 : Universal Hashing

(a) Prove $Prob$(max-chain-length in hash table of S under hash-function $H(x) > logn) < 1/n$.

Ans. Let A_i be the Event that atleast k numbers have been hashed to i.

so,

$$P[A_i] = \sum_{i=k}^{n} \binom{n}{i} \left(\frac{1}{n}\right)^i \left(1 - \frac{i}{n}\right)^{n-k}$$

Thus using the Union Bound, we have,

$$P[max\ chain\ length \geq k] \leq \sum_{i=1}^{n} P[i^{th}\ chain\ length \geq k] = \sum_{i=1}^{n} P[A_i]$$

$$\text{now, } P[A_i] \leq \sum_{i=k}^{n} ((ne/i)^i)((1/n)^i)$$

Using Stirling approximation, which is

$$\binom{n}{i} \leq (\frac{ne}{i})^i$$

$$\implies P[A_i] \leq (e/k)^k (1/(1 - e/k))$$

$$\implies P[A_i] \leq (e/k)^k$$

putting $k = log_2 n$ we get ,

$$P[A_i] \leq (e/log_2 n)^{log_2 n}$$

using $e = 2^{log_2 e}$

$$P[A_i] \leq 2^{log_2 n (log_2 e - log_2 (log_2 n))}$$

the above value depends on n, if we take sufficiently large value of n (approx $> 2^{32}$) , we can say , the above value is < -2, that is

$$P[A_i] < n^{-2}$$

P[i^{th} chain length $\geq k$] $< (1/n)^2$

Thus ,

$$P[max\ chain\ length \geq k] < 1/n^2 * n = 1/n$$

Thus ,

$$P[max\ chain\ length > k] < 1/n$$

Hence the above is proved for large enough n.

(b) Prove that for any given $r \in [1, p-1]$, there exists at least $\binom{M/n}{n}$ subsets of U of size n in which maximum chain length in hash-table corresponding to $H_r(x)$ is $\Theta(n)$.

Ans. The size of the Universe is M and all these M elements will get mapped to a number from 0 to $n-1$.

By pigeon hole principle, at least M/n of the elements will be mapped to one of the numbers from 0 to $n-1$ as if that is not the case then there will be less than M elements in total.

Let the number of elements that will be mapped to one of the numbers from 0 to $n-1$ with the condition that it is greater than M/n be x.

We can choose n elements among these x elements and all of them will be mapped to the same number between 0 and $n-1$ and hence the chain length will be exactly n which is $\theta(n)$. Since, $x \geq M/n$, we have $\binom{X}{n} \geq \binom{M/n}{n}$, so there are at least $\binom{M/n}{n}$ subsets of size n for which maximum chain length in hash-table corresponding to $H_r(x)$ is $\theta(n)$.

(c) Implement $H()$ and $Hr()$ in Python/Java for $M = 10^4$ and the following different choices of sets of size $n = 100$: For $k \in [1, n]$, S_k is union of $0, n, 2n, 3n, ..., (k-1)n$ and $n-k$ random elements in U. Obtain a plot of Max-chain-length for hash functions $H()$, $Hr()$ over different choices of sets S_k defined above. Note that you must choose a different random r for each choice of S_k. Provide a justification for your plots.

4.0.1 Code

```python
from random import *
from math import *
from matplotlib import pyplot as plt

def Hr(r, p, n, M):
  assert M <= p <= 2*M
  def hash(x):
    assert 0 <= x < M
    return (r*x % p) % n
  return hash

def S(n, k, M):
  assert 1 <= k <= n
  s = set([i*n for i in range(k)])
  for i in range(n-k):
    s.add(randint(0,M-1))
  return s

def max_chain_length(s, h, n):
  out = [0]*n
  for x in s:
    out[h(x)] += 1
  return max(out)

M = 10**4
n,p = 100, 10313
x_pts = list(range(1,n+1))
y_pts, yr_pts = [], []

for k in range(1,n+1):
  r = randint(1,p-1)
  hr = Hr(r, p, n, M)
  h = Hr(1, p, n, M)
  s = S(n, k, M)
  lr = max_chain_length(s, hr, n)
  l = max_chain_length(s, h, n)
  y_pts.append(l)
  yr_pts.append(lr)

plt.xlabel('k')
plt.ylabel('Max chain Length')
plt.grid(axis='y')
plt.grid(axis='x')

plt.plot(x_pts, y_pts, color='b', label='H(x)')
plt.plot(x_pts, yr_pts, color='r', label='Hr(x)')
plt.legend()

plt.savefig(f'{p}.png')
```

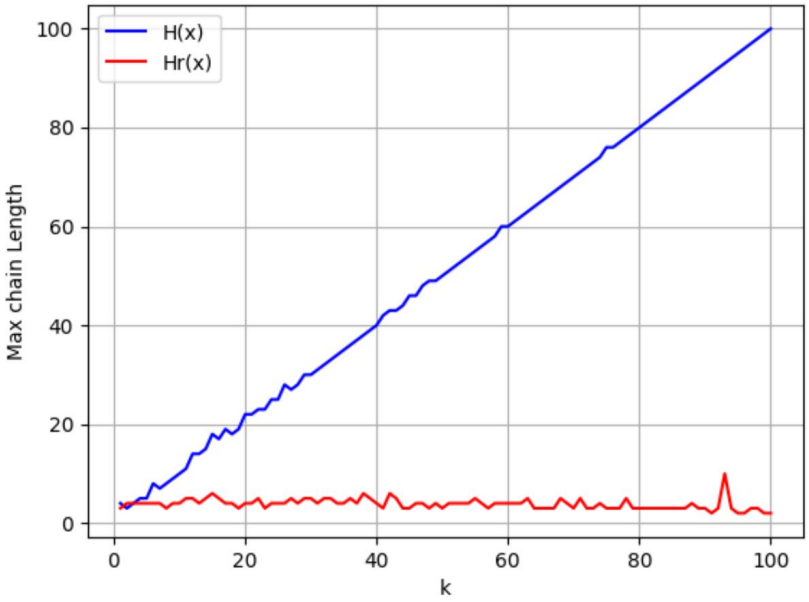

Figure 1: Plot of $H(x)$ and $Hr(x)$ for $p = 15173$

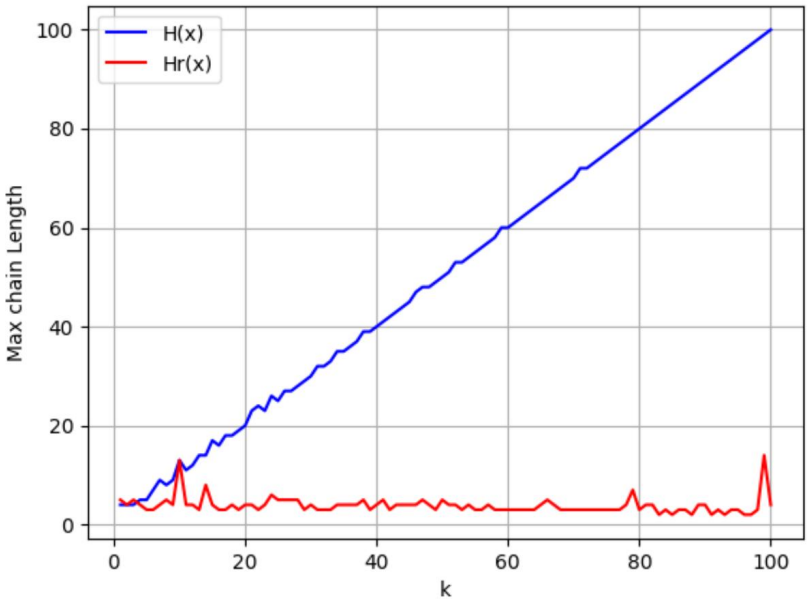

Figure 2: Plot of $H(x)$ and $Hr(x)$ for $p = 11093$

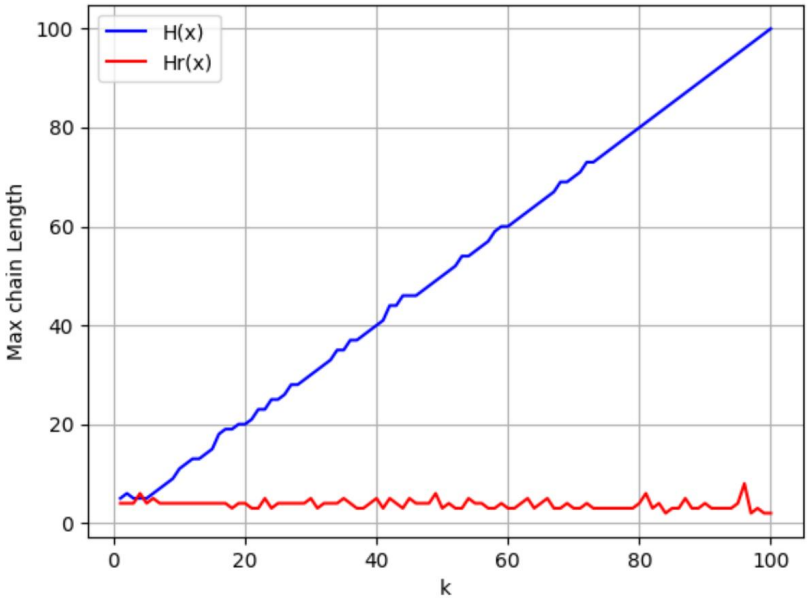

Figure 3: Plot of $H(x)$ and $Hr(x)$ for $p = 13177$

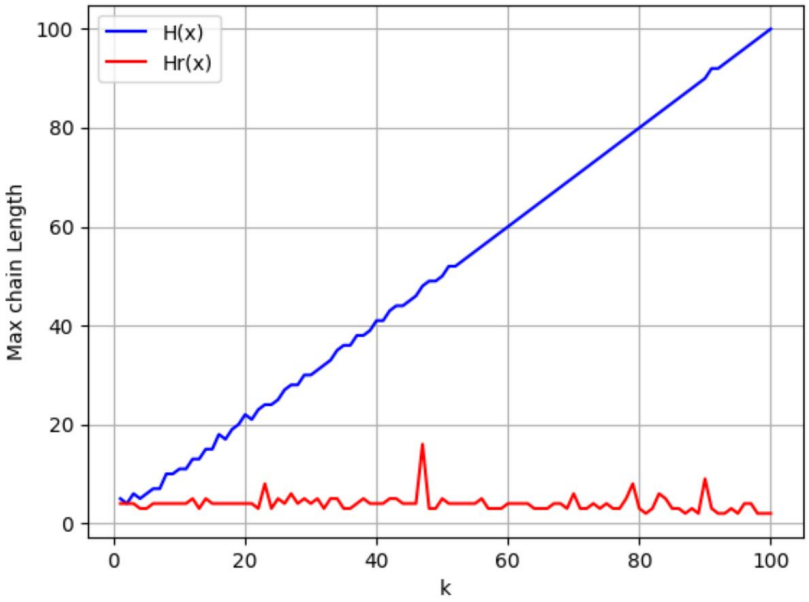

Figure 4: Plot of $H(x)$ and $Hr(x)$ for $p = 18493$

4.0.2 Explanation of plots

- For plot of $H(x)$ we observe a linear graph of S_k vs k. This can be explained using the fact that $(i * n)\%n = 0 \forall i$. So, for S_k contains atleast k elements that on hashing with $H(x)$ result

in a value of 0. Since the rest of the $n-k$ elements are randomly picked from the Universe, we can expect their contribution to the maximum chain length to be around the order of $(n-k)/n$ since each random element has an equal chance to he hashed into any value between 0 and $n-1$. So, S_k has a maximum chain length around k and hence we observe a linear graph.

- For plot of $Hr(x)$ we observe an almost constant graph with a low value around 3 to 5 mostly with some fluctuations. Since we choose a random r for each S_k, thus $(rx\%p)\%n$ is a random mapping of the set S_k to $[0, 1, \cdots n-1]$. Hence irrespective of the choice of k we end up with a random distribution of n integers in the range $0, 1, \cdots n-1$, which leads to low uniform values of maximum chain length. Thus we obtain an almost constant graph.

4 Hashing 15 / 15

Part a

✓ **+ 4 pts** **Correct analysis**

+ 3 pts Mistakes in the analysis

✓ **+ 3 pts** **Correct Conclusion**

+ 2 pts Mistakes in the observation

+ 0 pts Not attempted

Part b

✓ **+ 3 pts** **Correct Proof**

+ 0 pts Not attempted

Part c

✓ **+ 3 pts** **Plot**

✓ **+ 2 pts** **Justification**

+ 0 pts Not attempted

+ 0 pts Click here to replace this description.

+ 0 pts Not attempted