ALGORITHM ASSIGNMENT 2

CS2271

GROUP 2C

MEMBERS :

2021CSB071 PRAYAS MAZUMDER

2021CSB072 SOURIK SAHA

2021CSB074 AISHIKA DAS

2021CSB078 SAYAK SAHA

QUESTION 1:

- *1. You need to implement the polygon triangulation problem.*

  (a) Prepare a dataset for the convex polygon with increasing number of arbitrary vertices.
  (b) Consider a brute force method where you do an exhaustive search for finding the optimal result.
  (c) Next apply dynamic programming, in line with the matrix chain multiplication problem.
  (d) Then implement a greedy strategy to solve the same problem by choosing non-intersecting diagonals in sorted order.
  (e) Finally compare the results and suggest whether the DP and Greedy approach results show mismatch.

POLYGON TRIANGULATION PROBLEM ...

- Polygon triangulation Problem
- **Cost of triangulation** is considered to be the sum of side lengths of the triangles
- **Substructures** - The polygon can be split into two sub polygons by joining two vertices
- **Overlapping** - A pentagon can be split into triangle and quadrilateral and the resulting quadrilateral splits further into two triangles , which two can as well be the starting point

GREEDY VS DP APPROACH

The main difference between greedy and dynamic programming (DP) approaches is that greedy algorithms make locally optimal choices at each step, hoping to reach a global optimum, while DP solves a problem by breaking it down into smaller subproblems and finding the optimal solution to each subproblem.

Greedy algorithms typically work well when the problem has the greedy choice property, which means that the optimal solution to the problem can be obtained by making locally optimal choices. However, greedy algorithms do not always guarantee the optimal solution and can get stuck in local optima.

On the other hand, DP algorithms guarantee to find the optimal solution by exploring all possible subproblems and finding the optimal solution for each subproblem. However, DP can be computationally expensive, especially when the problem has a large number of subproblems.

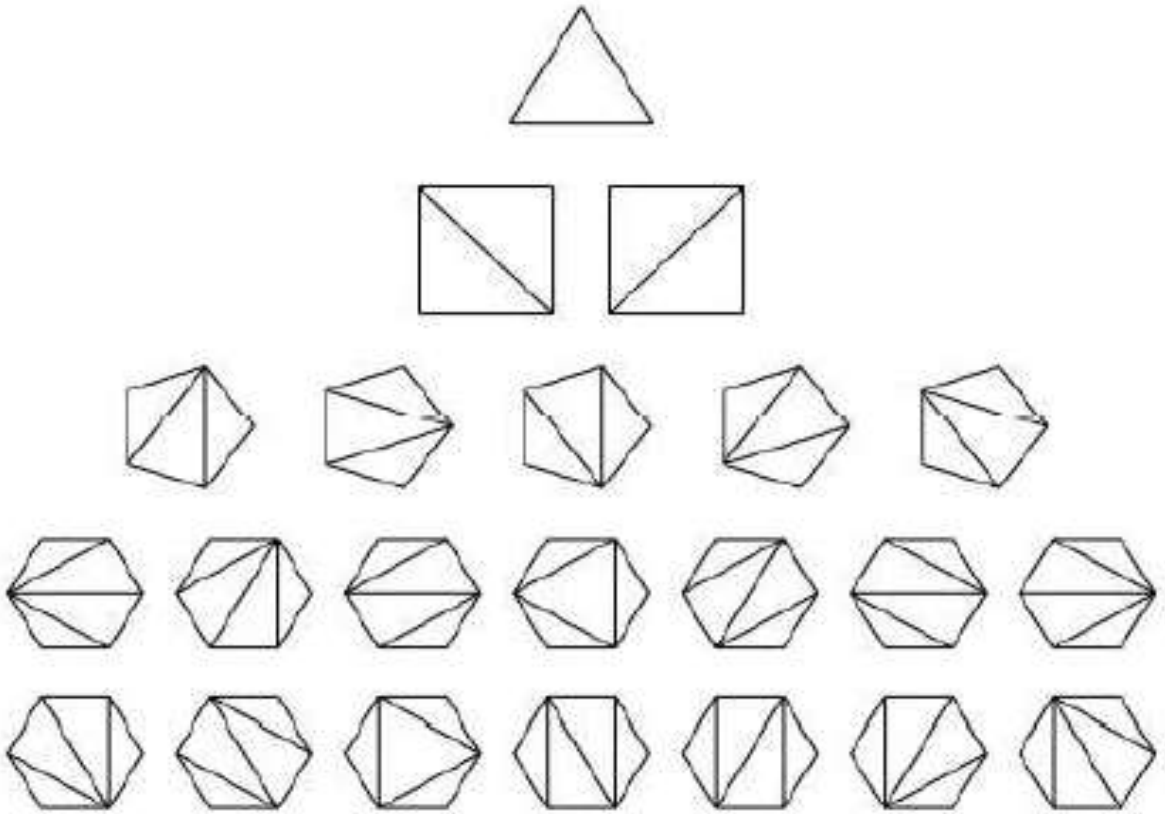
Therefore, the choice between greedy and DP approaches depends on the specific problem instance and the trade-off between finding the optimal solution and computational efficiency.

GREEDY VS DP FOR POLYGON TRIANGULATION PROBLEM

- The greedy approach involves choosing the next edge to triangulate based on some heuristics, such as selecting the shortest edge or the edge that creates the smallest angle. However, the greedy approach does not always guarantee the optimal solution.
- The dynamic programming approach involves breaking down the problem into smaller subproblems and solving them recursively. This approach can find the optimal solution but can be computationally expensive.

Therefore, the choice between the two approaches depends on the specific problem instance and the trade-off between finding the optimal solution and computational efficiency.

Polygon triangulation

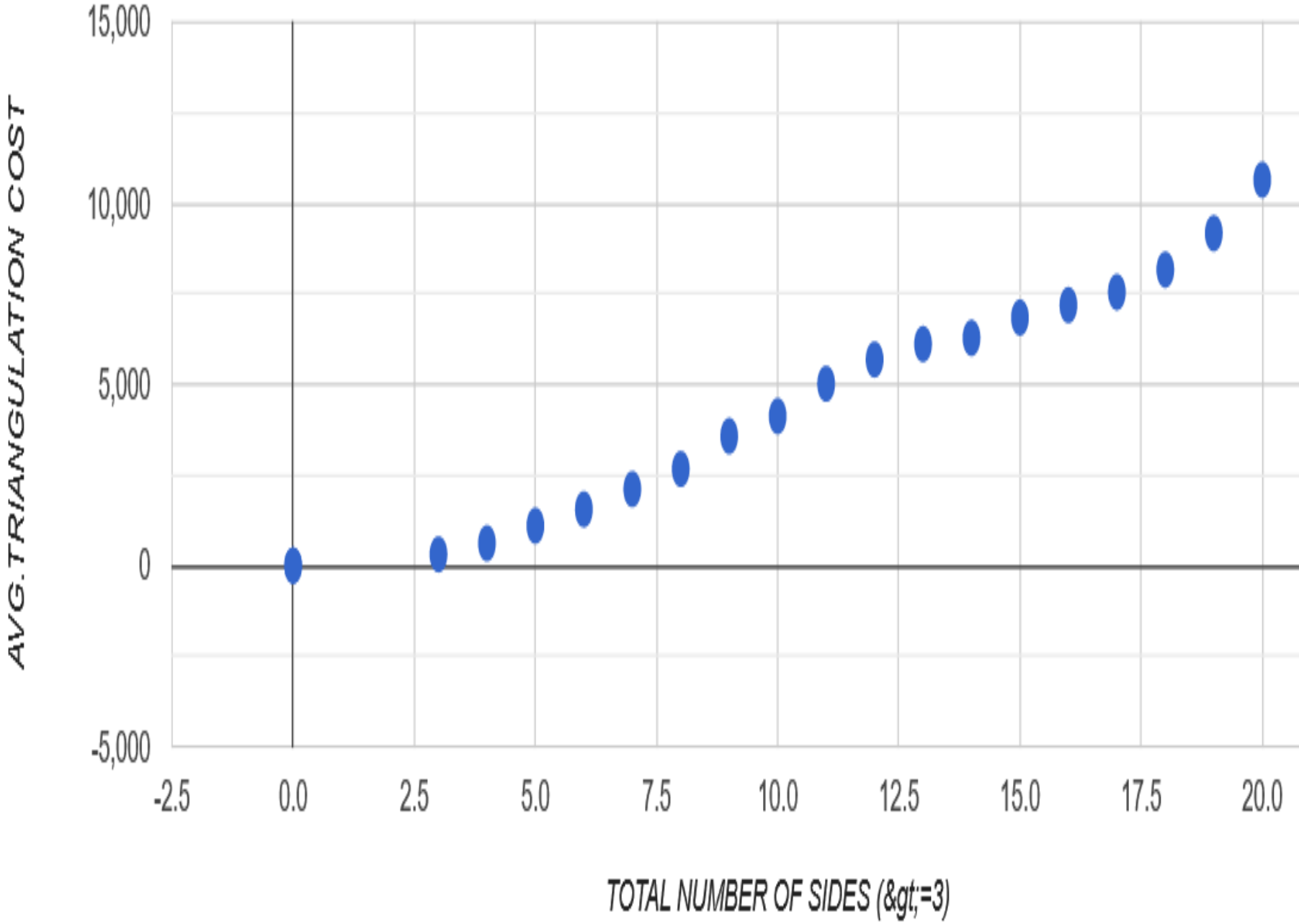


BRUTE FORCE METHOD DATASET

	abscissa x	ordinate y or f(x)
1	3	310
2	4	616
3	5	1101
4	6	1552
5	7	2110
6	8	2666
7	9	3576
8	10	4125
9	11	5018
10	12	5687
11	13	6114
12	14	6284

BRUTE FORCE METHOD

HERE IS THE PICTURE GRAPH DEPICTING THE AVG TRIANGULATION COST (AVEREGED FROM 10 DIFFERENT READINGS) WITH VARYING INCREASING NUMBER OF POLYGON VERTICES … FROM 3 TO 20 …



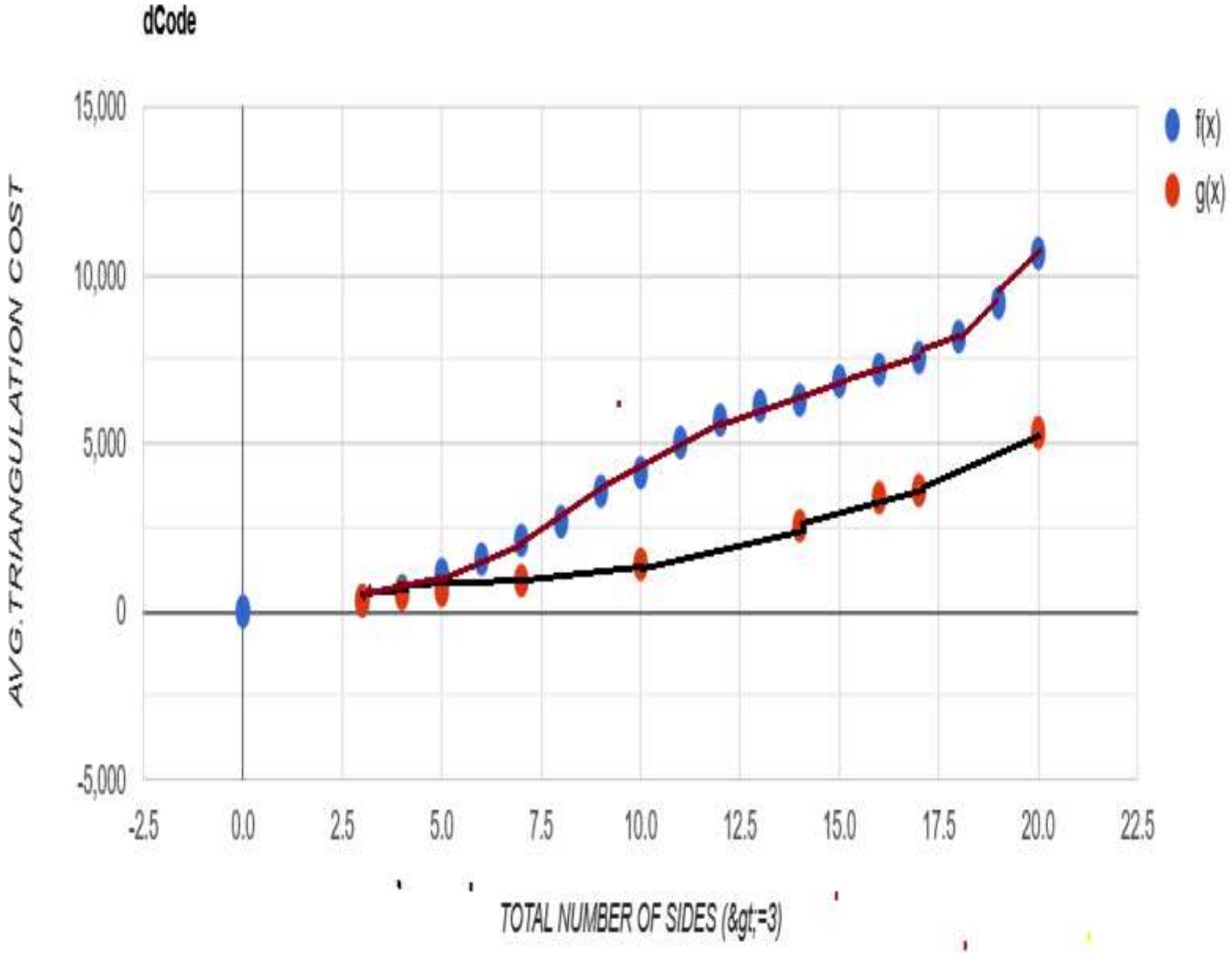
GREEDY APPROACH RESULT

	abscissa x	ordinate y or f(x)
1	3	332
2	4	523
3	5	618
4	7	914
5	14	2544
6	16	3381
7	17	3597
8	20	5315
9	10	1400

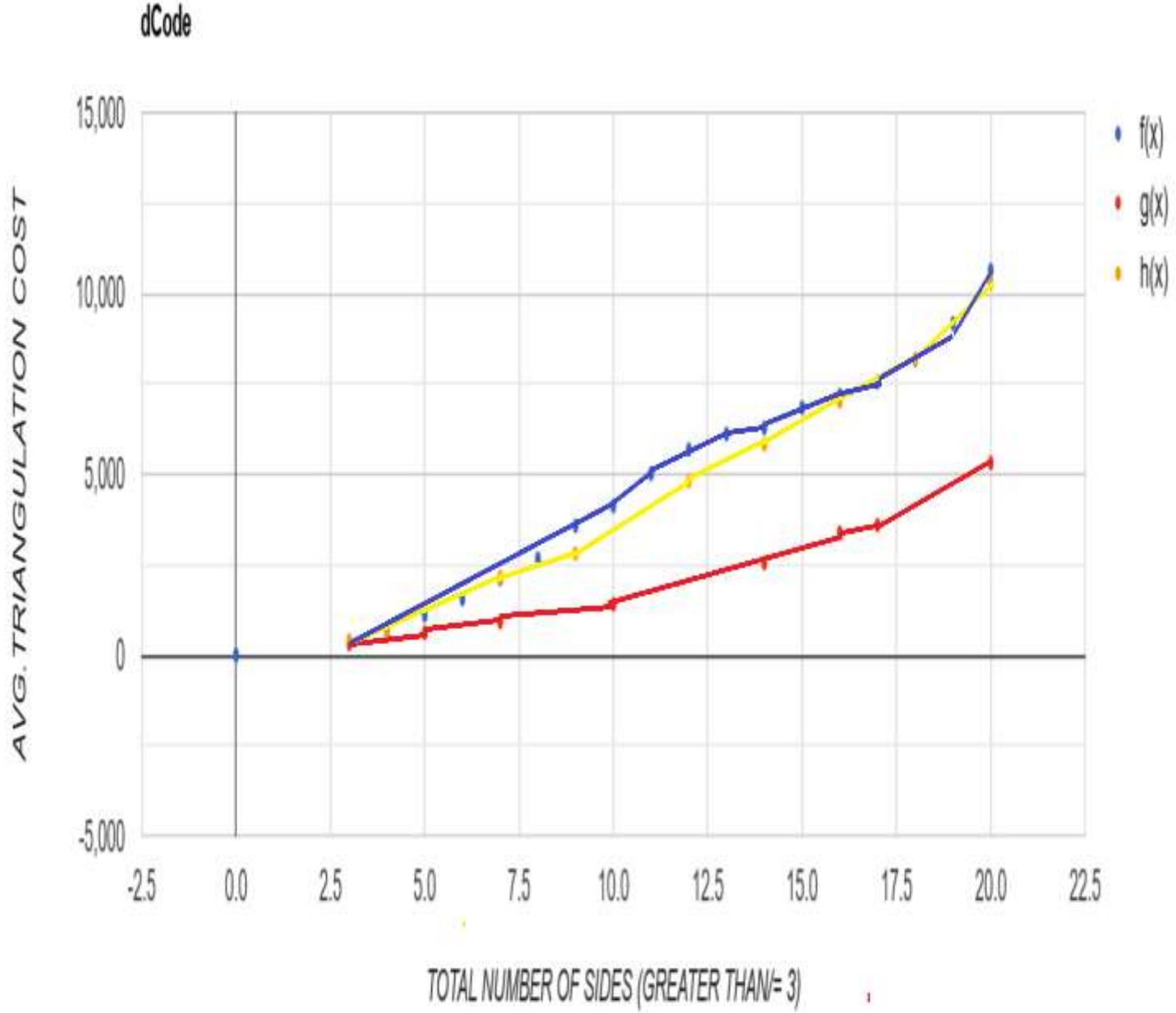
DYNAMIC PROGRAMMING (DP) APPROACH RESULT

	abscissa x	ordinate y or f(x)
1	3	409
2	4	642
3	7	2163
4	9	2811
5	12	4816
6	14	5830
7	16	7017
8	18	8191
9	20	10284

GRAPH SHOWING BRUTE FORCE VS GREEDY ALGORITHM APPROACH



BRUTE FORCE (BLUE) VS GREEDY (RED) VS DP APPROACH (YELLOW)



EXPLANATION

QUESTION 2 :

- *2. Implement data compression strategies:*
- *(a) Write encoding algorithms for the Shannon-Fano coding scheme.*
- *(b) Implement Huffman encoding scheme using heap as data structure.*
- *(c) Implement an instantaneous decoding algorithm*
- *(d) Choose a large text document and get results of Huffman coding with this document.*
- *(e) Examine the optimality of Huffman codes as data compression strategy by comparing with another document.*

DATA COMPRESSION

HUFFMAN CODING VARIABLE LENGTH ENCODING BASED ON FREQUENCY OF OCCURRENCE OF THE SYMBOLS COLLAPSE TWO LEAST OCCURRING SYMBOLS INTO COMPOUND SYMBOL CONTINUE THE PROCESS UNTIL TWO SYMBOLS ARE LEFT HEAP BASED CONSTRUCTION YIELDS THE CODING TREE MINIMUM OVERHEAD ON AVERAGE CODE WORD LENGTH ENSURED BY COLLAPSING OF LEAST PROBABLE SYMBOLS NO OTHER CODE THAT USES ANY OTHER STRATEGY IS CAPABLE OF BETTER COMPRESSION.

Prefix Rule

Huffman Coding implements a rule known as a prefix rule .This is to prevent the ambiguities while decoding . It ensures that the code assigned to any character is not a prefix of the code assigned to any other character.

SHANNON FANO CODING SCHEME

In the field of data compression, Shannon–Fano coding, named after Claude Shannon and Robert Fano, is a name given to two different but related techniques for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured).

HOW DOES IT WORK?

1.Create a list of probabilities or frequency counts for the given set of symbols so that the relative frequency of occurrence of each symbol is known.

2.Sort the list of symbols in decreasing order of probability, the most probable ones to the left and the least probable ones to the right.

3.Split the list into two parts, with the total probability of both parts being as close to each other as possible.

4.Assign the value 0 to the left part and 1 to the right part.

5.Repeat steps 3 and 4 for each part until all the symbols are split into individual subgroups.

The Shannon codes are considered accurate if the code of each symbol is unique

SHANNON FANO CODE FUNCTION

```
// function to find shannon code
void shannon_fano(int l, int h, node p[])
{
    float pack1 = 0, pack2 = 0, diff1 = 0, diff2 = 0;
    int i, d, k, j;
    if ((l + 1) == h || l == h || l > h)
    {
        if (l == h || l > h)
            return;
        p[h].arr[++(p[h].top)] = 0;
        p[l].arr[++(p[l].top)] = 1;
        return;
    }
    else
    {
        for (i = l; i <= h - 1; i++)
            pack1 = pack1 + p[i].pro;
        pack2 = pack2 + p[h].pro;
        diff1 = pack1 - pack2;
        if (diff1 < 0)
            diff1 = diff1 * -1;
        j = 2;
        while (j != h - l + 1)
        {
            k = h - j;
            pack1 = pack2 = 0;
            for (i = l; i <= k; i++)
                pack1 = pack1 + p[i].pro;
            for (i = h; i > k; i--)
                pack2 = pack2 + p[i].pro;
            diff2 = pack1 - pack2;
            if (diff2 < 0)
                diff2 = diff2 * -1;
            if (diff2 >= diff1)
                break;
            diff1 = diff2;
            j++;
        }
        k++;
        for (i = l; i <= k; i++)
            p[i].arr[++(p[i].top)] = 1;
        for (i = k + 1; i <= h; i++)
            p[i].arr[++(p[i].top)] = 0;

        // Invoke shannon function
        shannon_fano(l, k, p);
        shannon_fano(k + 1, h, p);
    }
}
```

EXAMPLE CODE RUN FOR SHANNON FANO ALGORITHM

```
PS C:\Users\ASUS> cd "e:\SAVED CODES\4TH SEM CODES\algorithm lab\assignment2\question_2_HUFFMAN_SHANNON\" ; if ($
  { g++ shannon_fano_algo.cpp -o shannon_fano_algo } ; if ($?) { .\shannon_fano_algo }
 Enter number of symbols : 8
 Enter symbol 1 : A
 Enter symbol 2 : B
 Enter symbol 3 : C
 Enter symbol 4 : D
 Enter symbol 5 : E
 Enter symbol 6 : F
 Enter symbol 7 : G
 Enter symbol 8 : H

  Enter probability of A : 0.22

  Enter probability of B : 0.26

  Enter probability of C : 0.15

  Enter probability of D : 0.2

  Enter probability of E : 0.05

  Enter probability of F : 0.02

  Enter probability of G : 0.06

  Enter probability of H : 0.04



        Symbol  Probability     Code
        B               0.26    00
        A               0.22    01
        D               0.2     10
        C               0.15    110
        G               0.06    1110
        E               0.05    11110
        H               0.04    111110
        F               0.02    111111
```

HUFFMAN DATA COMPRESSION

HUFFMAN CODING IS BASED ON THE IDEA THAT TO REPRESENT THE SYMBOLS WITH THE MOST FREQUENCY OF OCCURRENCE WITH THE LEAST NUMBER OF BITS AND THE SYMBOLS WITH THE MOST NUMBER OF OCCURRENCES BE REPRESENTED WITH THE MOST NUMBER OF BITS.

WHILE DAVID A. HUFFMAN WAS A PH.D. STUDENT AT MIT, THIS METHOD OF CODING WAS INTRODUCED TO THE WORLD IN 1951. HE WAS GIVEN THE TASK TO FIND AN EFFICIENT WAY OF CODING AND CAME UP WITH THE IDEA OF USING A FREQUENCY-SORTED BINARY TREE AND PROVED THIS METHOD TO BE THE MOST EFFICIENT. HE PUBLISHED THIS IN A PAPER IN 1952 TITLED “A METHOD FOR THE CONSTRUCTION OF MINIMUM REDUNDANCY CODES”.

HUFFMAN DATA COMPRESSION(2)

- **Data Compression**
- In communication; data compression is source coding where reduction of bits used is done. The overall aim is to use fewer bits to encode the data than the original number of bits. This coding is usually done at the source of the data before it is transmitted.
  Consider five symbols **A, B, C, D** and **E** to be transmitted which have a **probability of occurring 0.5, 0.2, 0.1, 0.1,** and **0.1** respectively. Since there are 5 symbols or letters, we have to use at least **three(3) bits** to represent one symbol. But if we use the Huffman coding method this can be reduced as follows.
- A — 000 can be represented as A — 0
- B — 001 can be represented as B — 10
- C — 010 can be represented as C — 010
- D — 011 can be represented as D — 011
- E — 100 can be represented as E — 100

- Thus reducing the overall number of bits required to represent the text. For example, to send the message “ABCDEF” across will be reduced from
- “000 0001 010 011 100” to “0 10 010 011 100” which is 3 bits lesser.
- The idea is to replace the *most frequently* occurring symbols with a low number of bits and the *least frequently* occurring symbols with a higher number of bits. If you clearly examine, it can be found that the symbols generated using Huffman coding that are representing the above letters are **never repeated**.

STEPS TO BUILD HUFFMAN TREE

Huffman tree is a technique that is used to generate codes that are distinct from each other. Using this method, most occurring symbols will get the least number of bits and others accordingly.

1.Sort all the different symbols and their particular frequency or probability.

2.Take the two lowest probability symbols and create a new common node with the probability equal to the sum of the two probabilities. Always make sure to add the lowest summing up nodes.

3.Add the new node to the table instead of the lowest two symbols or nodes.

4.Repeat steps two and three until one symbol or node is remaining.

5.Then allocate ‘0’ to all the right branches and ‘1’ to all right branches

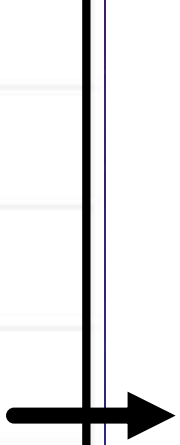
6.Read the bits from the top of the tree to the bottom of each symbol and record their particular bit pattern

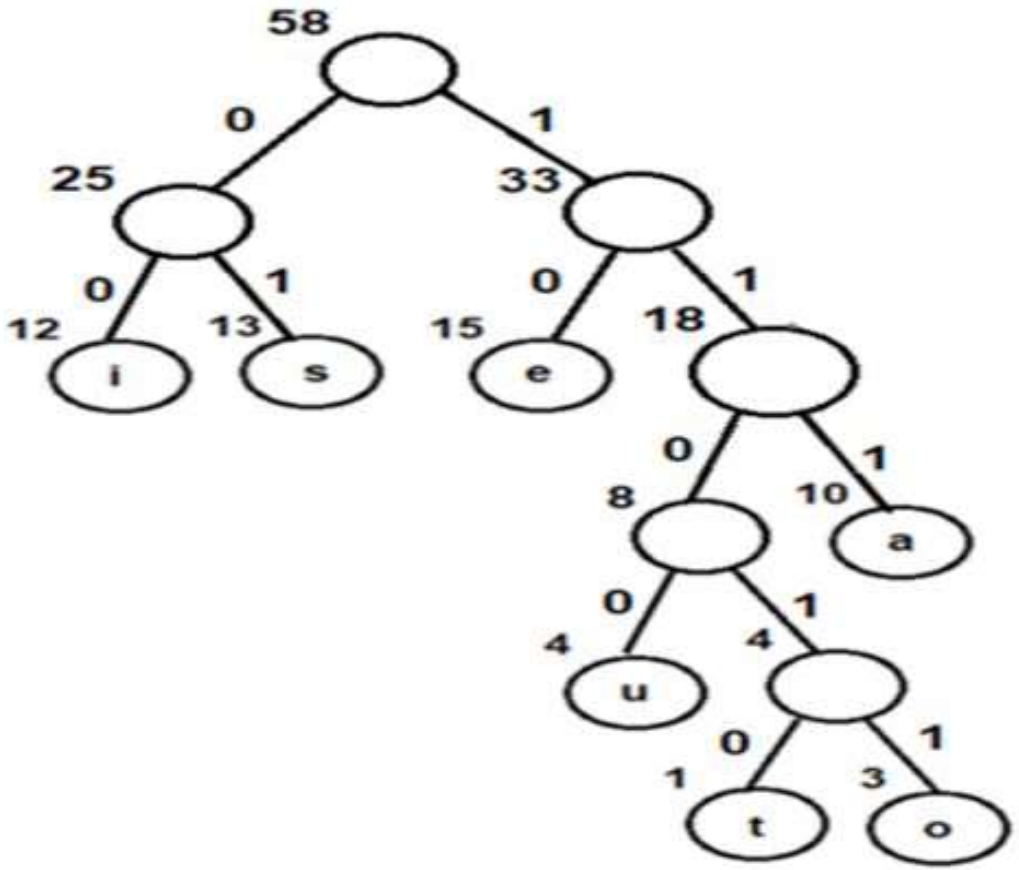
SAMPLE .TEXT FILE USED HERE FOR HUFFMAN ENCODING PROBLEM

- ChatGPT (Chat Generative Pre-trained Transformer) is an artificial intelligence chatbot developed by OpenAI and launched in November 2022. It is built on top of OpenAI's GPT-3.5 and GPT-4 families of large language models (LLMs) and has been fine-tuned (an approach to transfer learning) using both supervised and reinforcement learning techniques .ChatGPT was launched as a prototype on November 30, 2022, and quickly garnered attention for its detailed responses and articulate answers across many domains of knowledge. Its uneven factual accuracy, however, has been identified as a significant drawback. Following the release of ChatGPT , OpenAI's valuation was estimated at US$29 billion
- in 2023.ChatGPT was originally released in November 2022 based on GPT-3.5; a version based on GPT-4, the newest OpenAI model, was released on March 14, 2023 and is available for paid subscribers on a limited basis . ChatGPT is a member of the generative pre-trained transformer (GPT) family of language models. It was fine-tuned (an approach to transfer learning) over an improved version of OpenAI's GPT-3 known as "GPT 3.5". The fine-tuning process leveraged both supervised learning as well as reinforcement learning in a process called reinforcement learning from human feedback (RLHF). Both approaches used human trainers to improve the model's performance. In the case of supervised learning, the model was provided with conversations in which the trainers played both sides: the user and the AI assistant. In the reinforcement learning step, human trainers first ranked responses that the model had created in a previous conversation . These rankings were used to create 'reward models' that the model was further fine-tuned on using several iterations of Proximal Policy Optimization (PPO). Proximal Policy Optimization algorithms present a cost-effective benefit to trust region policy optimization algorithms; they negate many of the computationally expensive operations with faster performance. The models were trained in collaboration with Microsoft on their Azure supercomputing infrastructure, using Nvidia GPUs, "supercomputer developed for OpenAI is a single system with more than 285,000 CPU cores, 10,000 GPUs and 400 gigabits per second of network connectivity for each GPU server". OpenAI collects data from ChatGPT users to further train and fine-tune the service. Users can upvote or downvote responses they receive from ChatGPT and fill out a text field with additional feedback.

EXAMPLE OF HUFFMAN TREE BUILDING

Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1





HUFFMAN ENCODING FREQUENCY FOR SAMPLE.TXT FILE

---------CODE TABLE---------

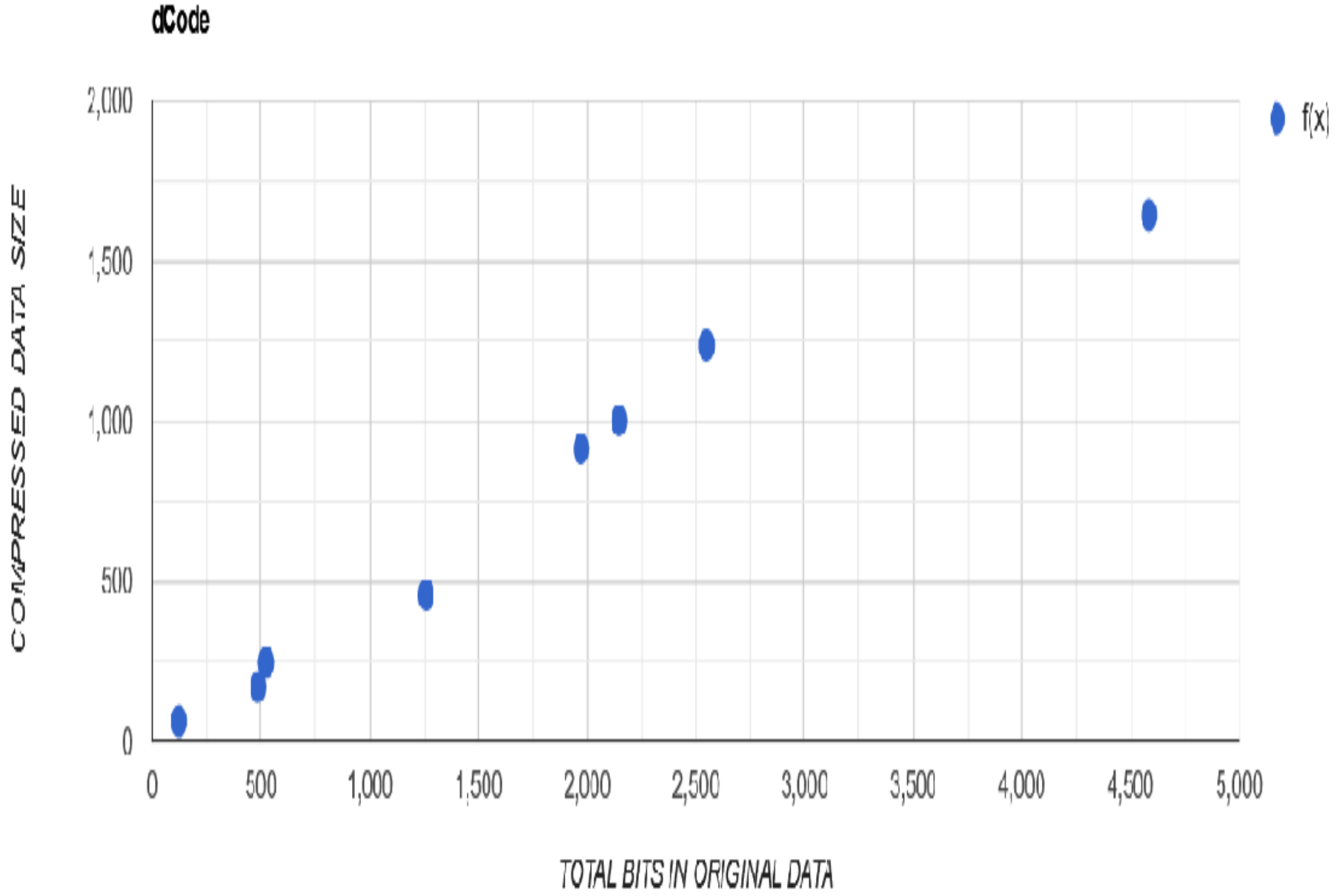
CHAR	FREQ	CODE
' '	364	001
"	4	1010111000
$	1	000100111011
'	6	000011101
(	7	000011100
)	7	101011101
,	15	1010110
-	13	01100101
.	20	0110101
0	15	1010100
1	2	00010011111
2	15	01100110
3	7	011010001
4	4	000011111
5	4	0001001101
8	1	000100111010
9	1	000100111001
:	1	1010101110
;	2	00010011110
A	9	000011110
B	1	011001111
C	9	00010001
F	2	0110100001
G	18	0110110
H	1	011001110
I	13	10101111
L	3	101010110
M	3	1010111001
N	4	0110001
O	10	01101001
P	25	0001010
R	1	000100111000
S	1	1010101111
T	18	0110111
U	6	10101010
a	168	0100
b	25	0001011
c	58	000001
d	72	10100
e	249	111
f	52	000010
g	34	100000
h	62	000000
i	140	1011
k	9	00010000
l	73	10001
m	49	000110
n	160	0101
o	129	1100
p	50	000111
q	2	0110100000
r	140	1001
s	128	1101
t	144	0111
u	45	0110000
v	34	100001
w	26	0000110
x	4	01100100
y	17	00010010
z	4	0001001100

SOME DATA SET FOR ORIGINAL DATA FILE SIZE VS COMPRESSED DATA FILE SIZE

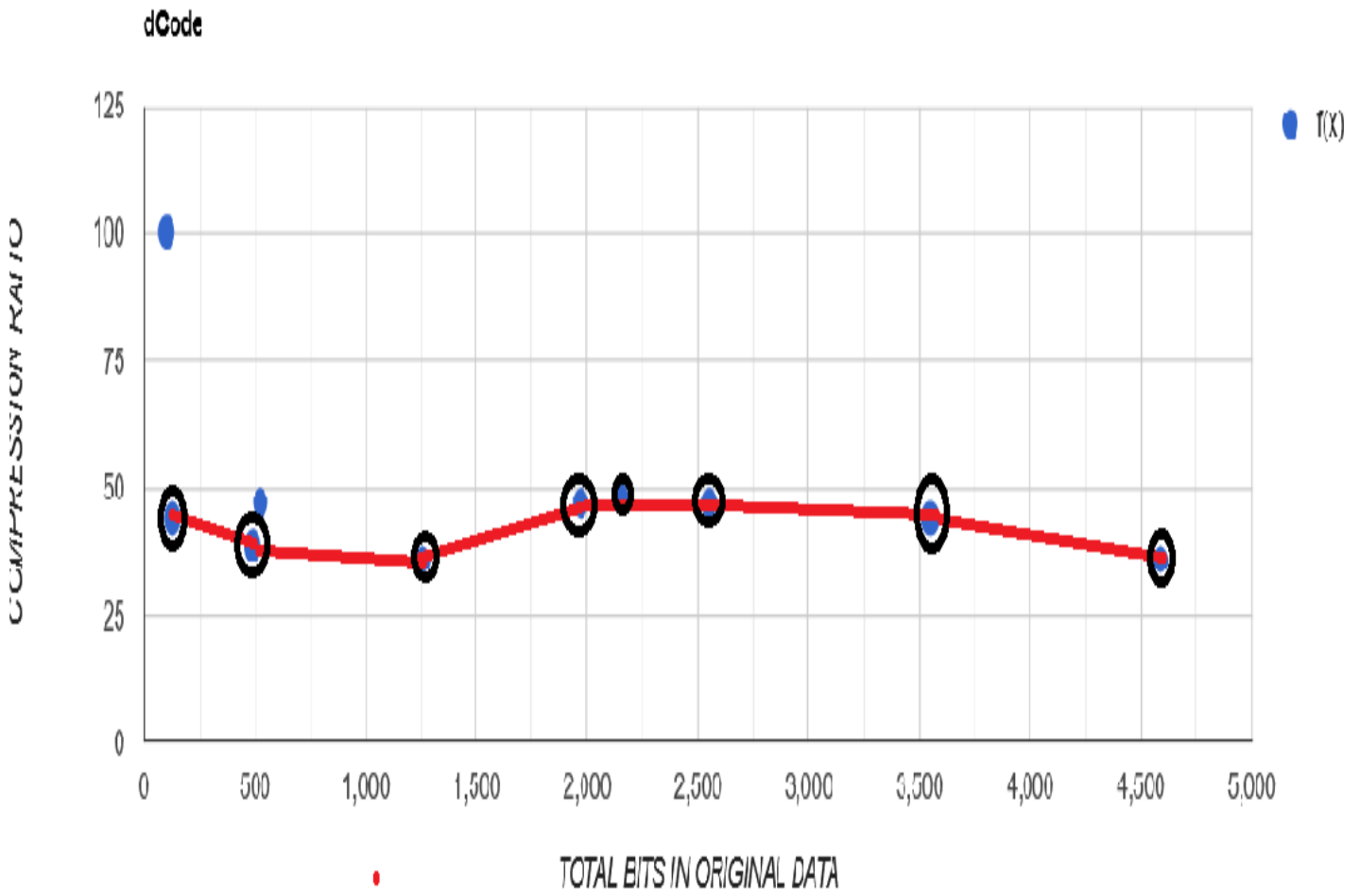
	abscissa x →	ordinate y or f(x) ↑
1	1972	915
2	4587	1645
3	125	64
4	1258	458
5	2548	1238
6	524	247
7	487	171
8	2147	1003

SHOWS A STEADY COMPRESSION RATIO (COMPRESSED SIZE/ORIGINAL SIZE) OF ABOUT 44% IN AVERAGE …

PLOT OF ORIGINAL DATASET VS COMPRESSED DATASET



COMPRESSION RATIO VS TOTAL NUMBER OF BITS OF THE ORIGINAL DOCUMENT ---> GRAPH



QUESTION 3 :

- *3. We need to scale up the things - from toy problems to real life problems. For this, you need to study some large datasets provided by reputed Universities, like*
-
  (a) SNAP - Stanford Network Analysis Project - datasets collected by Stanford University.
-
  (b) KONECT - Koblenz Network Collection - datasets compiled by Koblenz University, Deutschland (Germany).

First study and feel the vastness of these networks from the dynamic and disjoint set operations point of view. Then try to implement graph algorithms like Connected Components and Minimum Spanning Tree using appropriate data structure for these datasets.

WHAT IS SNAP ?

SNAP is **a collection of large network datasets**. It includes graphs representing social networks, citation networks, web graphs, online communities, online reviews and more. Social networks : online social networks, edges represent interactions between people.

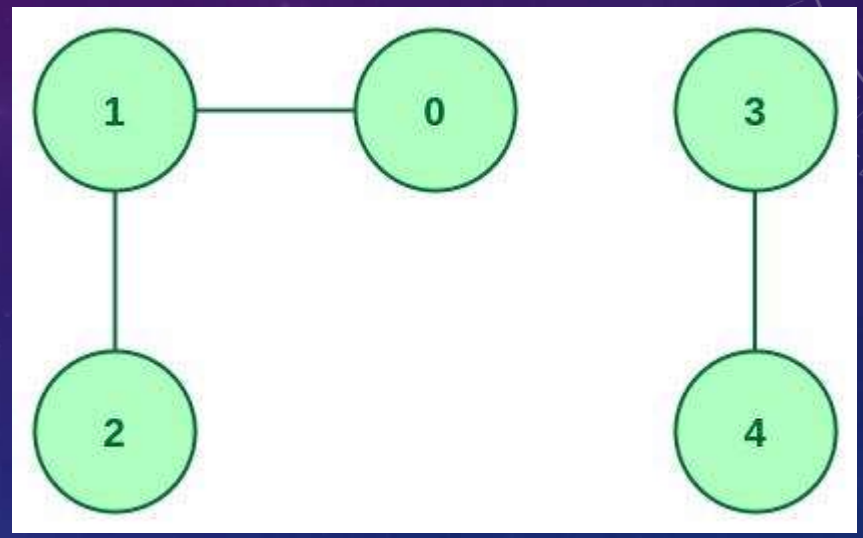
A social network dataset is **a dataset containing the structural information of a social network**. In the general case, a social network dataset consists of persons connected by edges. Social network datasets can represent friendship relationships or may be extracted from a social networking Web site

- FULL FORM : STANFORD LARGE NETWORK DATASET COLLCETION
- WEBSITE : https://snap.stanford.edu/data/

- IT CONTAINS -→
- Social networks ,Networks with ground-truth communities :
- Communication networks :Citation networks
- Collaboration networks, Web graphs Amazon networks :
- Internet networks : Road networks : Autonomous systems : Signed networks : Location-based online social networks :
- Wikipedia networks, articles, and metadata
- Temporal networks ,Twitter and Meme-tracker ,
- Online communities :Online reviews ,
- User actions :Face-to-face communication networks :Graph classification datasets :
- Computer communication networks :Cryptocurrency transactions : Telecom networks :

CONNECTED COMPONENT

In graph theory, a component of an undirected graph is a connected subgraph that is not part of any larger connected subgraph. The components of any graph partition its vertices into disjoint sets, and are the induced subgraphs of those sets



UNDIRECTED GRAPH

Connected Component for undirected graph using *Disjoint Set Union*

The idea to solve the problem using DSU (Disjoint Set Union) is

Initially declare all the nodes as individual subsets and then visit them. When a new unvisited node is encountered, unite it with the under. In this manner, a single component will be visited in each traversal.

Declare an array A[] of size V where V is the total number of nodes.

For every index ‘i’ of array A[], the value denotes who the parent of i’th vertex is.

Initialize every node as the parent of itself and then while adding them together, change their parents accordingly.

Traverse the nodes from 0 to V :

For each node that is the parent of itself start the DSU.

Print the nodes of that disjoint set as they belong to one component.

MINIMUM SPANNING TREE

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components.

KRUSKAL'S MINIMUM SPANNING TREE ALGORITHM

A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. To learn more about Minimum Spanning Tree, refer to this article.

Introduction to Kruskal's Algorithm:

Here we will discuss Kruskal's algorithm to find the MST of a given weighted graph.

In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first at the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.

How to find MST using Kruskal's algorithm?

Sort all the edges in non-decreasing order of their weight.

Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

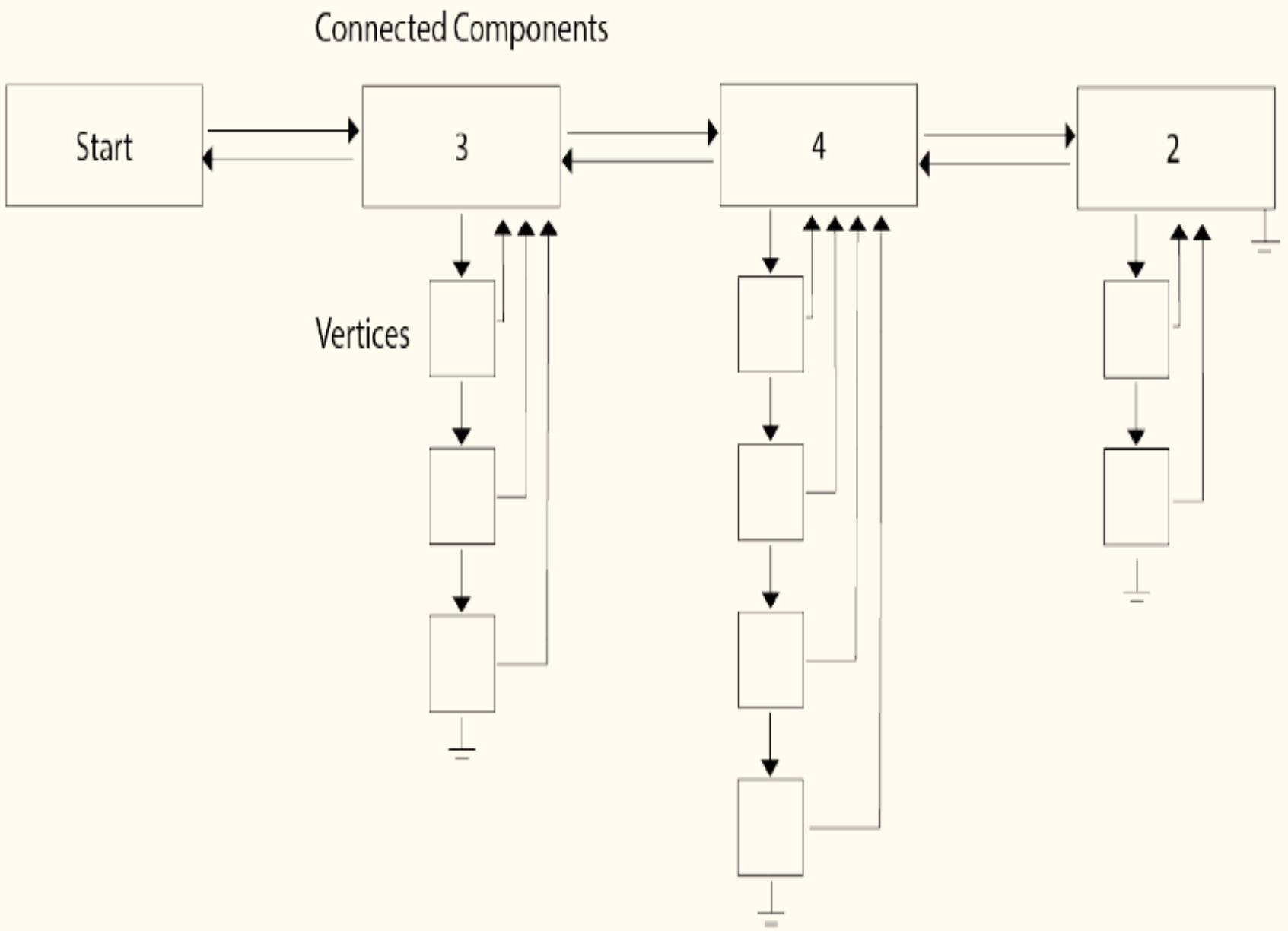
Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far

LINKED LIST IMPLEMENTATION OF DISJOINT SET

Here we used Two Linked Lists, One for dynamically managing number of Connected Components of the graph (also called Representative Element), and other to maintain the list of vertices inside that connected component
The Connected Component list is doubly linked while the vertices list in singly linked.
Every Representative Node also has a length parameter, which stores the length of vertex linked list inside it.
Every vertex also has a representative pointer which points to its Connected Component Node

LINKED LIST IMPLEMENTATION OF DISJOINT SET (2)



Implementation of Make_Set, Find_Set and Union

Make_Set makes a new Representative node with a given vertex (an int) and adds this node to the existing Double Linked List

Find_Set goes through all the Representative Nodes, checking all the node's value and if found, returns it's representative element.

Union takes two representative nodes as input, looks at their length, and shifts the lower length's vertex list to the higher length's vertex list, while updating the length and the representative elements of the higher length vertex list, then it deletes the lower length representative vertex.

Expected Time Complexity: O(lgn) where n is the number of Unions required, i.e more or less equal to number of edges.


ARIGATO