ALGORITHM LAB

ASSIGNMENT 2

CS-2271

This report is made by:

Tanis Ahamed – 2021CSB012

Subham Ghosh – 2021CSB022

Ananya Satrudhar – 2021CSB110

Code can be found in:

Here

Question 1–A:

Prepare a dataset for the convex polygon with increasing number of arbitrary vertices.

What is it?

We use a **rand()** function to generate random vertices and, by entering such vertices, we check whether we can form a convex polygon by using the function **isConvex()**.

1. A polygon is strictly convex if in addition, no three vertices lie on the same line. Equivalently, all its interior angles are less than 180 degrees.
2. Hence, it is checked by doing cross product of three consecutive lines.

Overall we use the function **generatePolygon()** to generate n number of polygons each having a defined number of vertices.

This polygon is used for further processing down below.

Code Snippets:

```c
typedef struct
{
    int x;
    int y;
} Vertex;

typedef struct
{
    int total_vertices;
    Vertex vertices[MAX];
} Polygon;

double min(double a, double b)
{
    return (a < b) ? a : b;
}

// Generating the vertices randomly
Vertex generateVertex()
{
    Vertex v;
    v.x = rand() % MAX_VAL;
    v.y = rand() % MAX_VAL;
    return v;
}
```

Code Snippets:

generator.cpp

```cpp
int crossProduct(Vertex v1, Vertex v2) // vertex direction vector
{
    return v1.x * v2.y - v1.y * v2.x;
}

// Function to determine if a polygon is convex
int isConvex(Polygon poly)
{
    int n = poly.total_vertices;
    int crossproduct = 0;
    for (int i = 0; i < n; i++)
    {
        Vertex v1 = poly.vertices[i];
        Vertex v2 = poly.vertices[(i + 1) % n];
        Vertex v3 = poly.vertices[(i + 2) % n];
        int cp = crossProduct((Vertex){v2.x - v1.x, v2.y - v1.y}, (Vertex){v3.x - v2.x, v3.y - v2.y});
        if (i == 0)
        {
            crossproduct = cp;
        }
        else if (crossproduct * cp < 0)
        {
            return 0;
        }
    }
    return 1;
}
```

Code Snippets:

generator.cpp

```
Polygon generatePolygon(int n)
{
    Polygon P;
    P.total_vertices = n;
    Vertex* arr = new Vertex[n];
    arr[0] = generateVertex();
    for (int i = 1; i < n; i++)
    {
        Vertex v;
        do
        {
            v = generateVertex();
        } while (!isConvex((Polygon){i, *arr}));
        arr[i] = v;
    }
    for (int i = 0; i < n; i++)
    {
        P.vertices[i] = arr[i];
    }
    return P;
}
```

Question 1-B:

Consider a brute force method where you do an exhaustive search for finding the optimal result.

Procedure:-

- We define two functions:

1. **minTriangulationCost**: This function recursively calculates the minimum cost of triangulating a given polygon. It takes as input an array of vertices **points** and two indices **i** and **j** representing the range of vertices to consider. It first checks the base case where **j** is less than or equal to **i+1**, which means there are no triangles to form. It then iterates through all possible indices **k** in the range **(i+1, j-1)** and calculates the cost of triangulating the polygon by forming a triangle between **points[i]**, **points[j]**, and **points[k]**. The minimum cost among all possible triangles is returned. This function calls itself recursively for the left and right sub-polygons formed by splitting at the minimum cost triangle. The overall minimum cost is the sum of the minimum costs of each sub-polygon and the cost of the minimum cost triangle.

Procedure:-

2. **cost**: This function calculates the cost of a given triangle formed by three vertices. It takes as input an array of vertices **points** and the indices **i**, **j**, and **k** representing the indices of the vertices forming the triangle. It calculates the sum of the distances between each pair of vertices in the triangle and returns the result.

By recursively calling the **minTriangulation()** function, we find a brute force solution of the Polygon Triangulation problem. This will be used to formulate a DP solution later.

Code Snippets:

bruteForce.cpp

```cpp
double cost(Vertex *points, int i, int j, int k)
{
    Vertex v1 = points[i];
    Vertex v2 = points[j];
    Vertex v3 = points[k];
    return dist(v1, v2) + dist(v2, v3) + dist(v3, v1);
}

double minTriangulationCost(Vertex *points, int i, int j)
{
    // Base Case
    if (j <= i + 1)
    {
        return 0;
    }
    double ans = 1e7;
    for (int k = i + 1; k < j; k++)
    {
        ans = min(ans, (minTriangulationCost(points, i, k) + minTriangulationCost(points, k, j) + cost(points, i, k, j)));
    }
    return ans;
}
```

Results:

We then tally the results in a file, from 3 vertices to 15 vertices.
e.g.

Number of Vertices: 3
57,158
20,46
121,192
Cost of Triangulation is 367.954343

Number of Vertices: 15
80,150 173,141 28,84 2,116 169,115 150,7
21,62 129,164 144,53 56,63 119,110 95,69
58,58 106,155 75,25
Cost of Triangulation is 3012.170091

Question 1-C:

Next apply dynamic programming, in line with the matrix chain multiplication problem.

Procedure:-

All the functions are almost similar except we use dynamic programming approach where we store the values of smaller triangles in a 2D array rather than calculating them over and over again.

DP.cpp

```
double dp[MAX][MAX];
```

The minTriangulatioCostDP() function is also similar to the normal function.

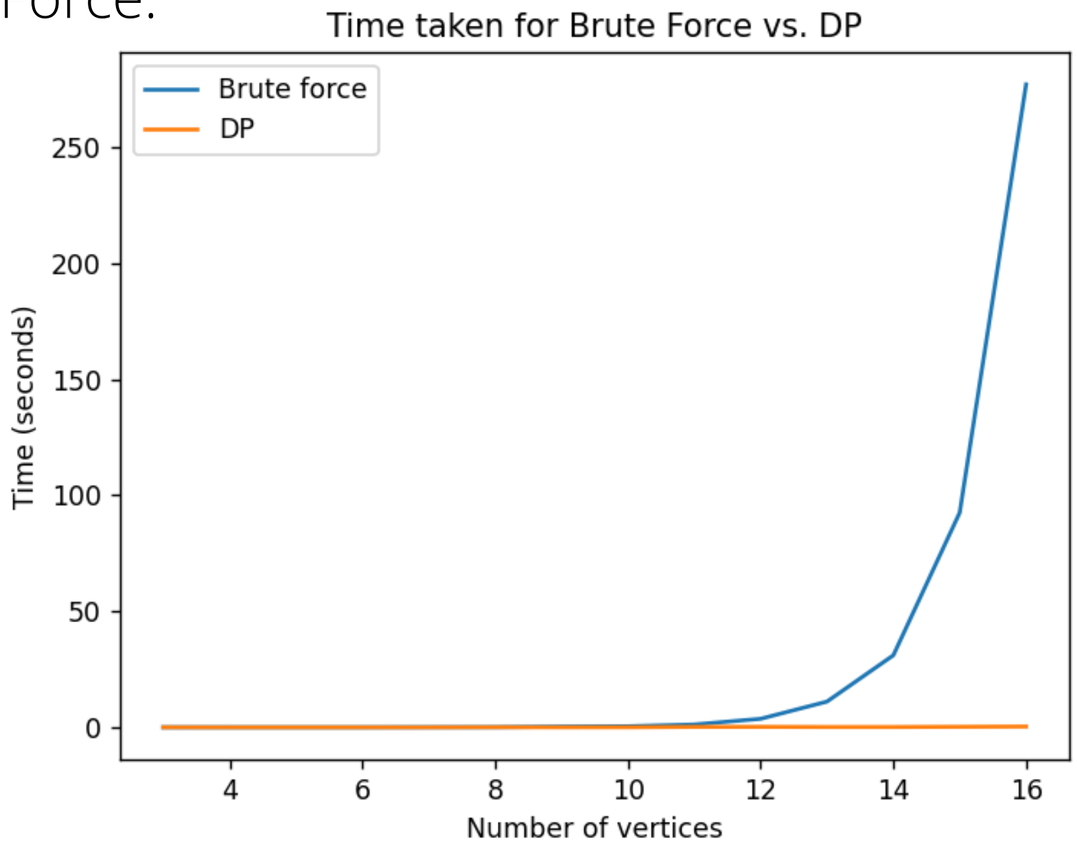
Code Snippets:

DP.cpp

```cpp
double minTriangulationCostDP(Vertex *points, int n)
{
  double Max = 1e7;
  if (n <= 2)
  {
    return 0;
  }
  /* table[i][j] will store the min triangulation cost for vertices from points
    i to j. Final ans will table[0][n-1]
  */
  for (int m = 0; m < n; m++)
  {
    for (int i = 0, j = m; i < n; i++, j++)
    {
      if (j <= i + 1)
      {
        dp[i][j] = 0.0;
      }
      else
      {
        dp[i][j] = Max;
        for (int k = i + 1; k < j; k++)
        {
          double ans = dp[i][k] + dp[k][j] + cost(points, i, j, k);
          dp[i][j] = min(dp[i][j], ans);
        }
      }
    }
  }
  return dp[0][n - 1];
}
```

Results:

We take the time taken of average 50 cases for each number of sides and also match the DP solution and the Brute Force solution, we find that both the solutions are the same for every case. Hence, we can prove that DP solution is optimal and with better time complexity than Brute Force.



Question 1-D:

Then implement a greedy strategy to solve the same problem by choosing non-intersecting diagonals in sorted order.

Procedure:-

The function **minTriangulationCostGreedy()** calculates the minimum cost of triangulating a given polygon using a greedy algorithm.

The function takes as input an array of n vertices represented by the points parameter, where each vertex is a Vertex structure with an x and a y coordinate.

The function first generates all the possible non-adjacent diagonals of the polygon and sorts them in ascending order of length. It then iterates through the diagonals in this sorted order and selects each diagonal that does not *intersect* any previously selected diagonals or the edges of the polygon. The function keeps track of the total *cost* of the selected diagonals and the count of selected diagonals, which should be n-3 in a convex polygon.

Procedure:-

After selecting all the diagonals, the function calculates the total *cost* of the sides of the polygon using the dist function, which computes the Euclidean distance between two points. Finally, the function returns the sum of the cost of the selected diagonals and the cost of the sides of the polygon.

Overall, this function uses a greedy algorithm to quickly find a close-to-optimal triangulation of a given polygon. However, it may not always produce the optimal triangulation, and the optimal solution may depend on the order in which diagonals are considered.

Code Snippets:

```
double minTriangulationCostGreedy(Vertex *points, int n)
{
  Diagonal arr[MAX * MAX];
  int taken[MAX][MAX] = {0};
  int count_diag = 0;
  for (int i = 0; i < n; i++)
  {
    for (int j = i + 2; j <= n - 2; j++) // non-adjacent
    {
      if (!taken[i][j] && !taken[j][i])
      {
        taken[i][j] = 1;
        taken[j][i] = 1;
        double len = dist(points[i], points[j]);
        arr[count_diag].i = i;
        arr[count_diag].j = j;
        arr[count_diag].length = len;
        count_diag++;
      }
    }
  }
  qsort(arr, count_diag, sizeof(Diagonal), cmp_diag);
```

Code Snippets:

```
int visited[MAX][MAX] = {0};
  double diag_cost = 0;
  int count = 0;
  for (int k = 0; k < count_diag; k++)
  {
    Diagonal diag = arr[k];
    if (!visited[diag.i][diag.j] && !intersects(points[diag.i], points[diag.j], points[(diag.i + 1) % n], points[(diag.j + 1) % n]) && count <= n - 3)
    {
      visited[diag.i][diag.j] = 1;
      diag_cost += diag.length;
      count++;
      // used diag.i and diag.j which can be printed later
    }
  }
  double side_cost = 0;
  for (int i = 0; i < n; i++)
  {
    int j = (i + 1) % n;
    side_cost += dist(points[i], points[j]);
  }
  return diag_cost + side_cost;
}
```

Code Snippets:

```
int cmp_diag(const void *u, const void *v)
{
  Diagonal *du = (Diagonal *)u;
  Diagonal *dv = (Diagonal *)v;
  return (du->length < dv->length) ? 1 : -1;
}
```

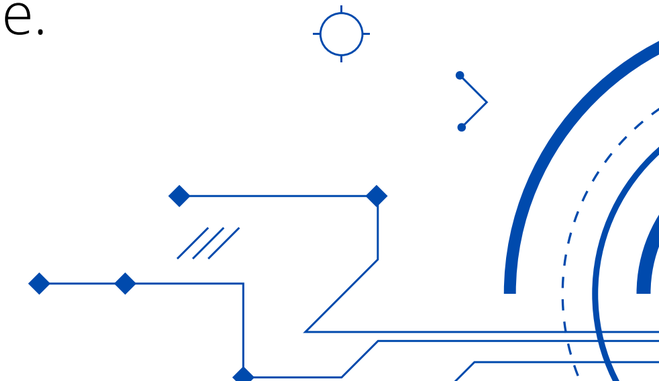
```
int isConvex(Polygon poly)
{
  int n = poly.total_vertices;
  int crossproduct = 0;
  for (int i = 0; i < n; i++)
  {
    Vertex v1 = poly.vertices[i];
    Vertex v2 = poly.vertices[(i + 1) % n];
    Vertex v3 = poly.vertices[(i + 2) % n];
    int cp = crossProduct((Vertex){v2.x - v1.x, v2.y - v1.y}, (Vertex){v3.x - v2.x, v3.y - v2.y});
    if (i == 0)
    {
      crossproduct = cp;
    }
    else if (crossproduct * cp < 0)
    {
      return 0;
    }
  }
  return 1;
}
```

Procedure:-

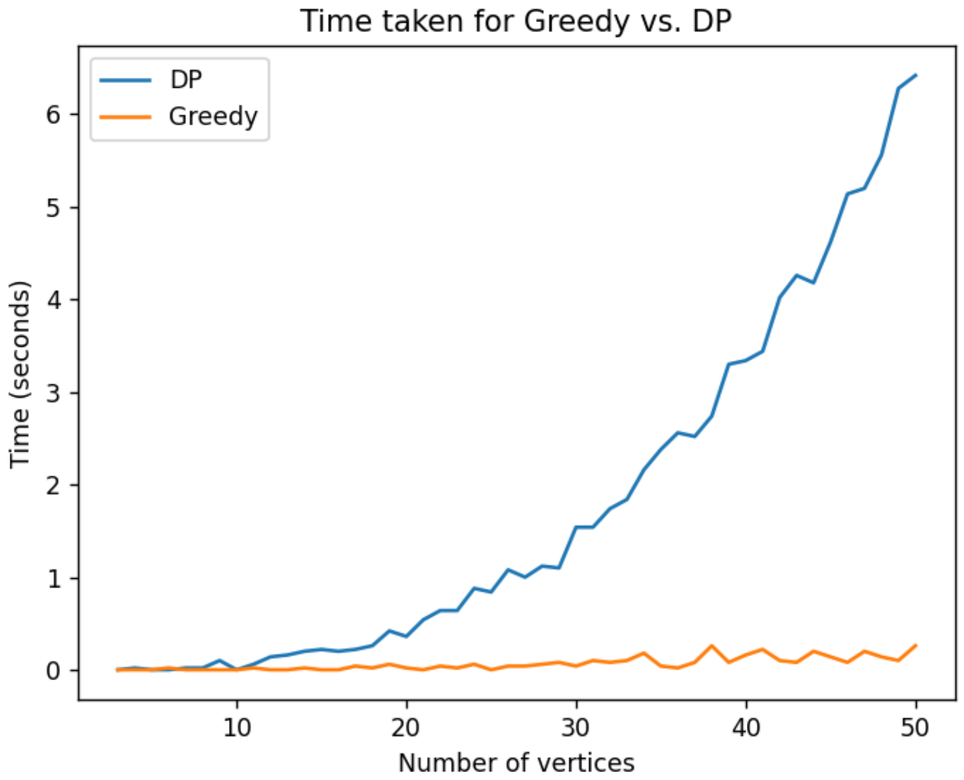
The **intersects** function takes four Vertex parameters representing two line segments AB and CD, and returns an integer value. It checks whether these two line segments intersect or not.

To check the intersection of two line segments, the function first calculates the direction of each line segment. It does so by calculating the cross products of the vectors formed by the two endpoints of each segment. Then, it checks whether the endpoints of one segment lie on opposite sides of the other segment. If this is true for both segments, then the two line segments intersect.

The function returns 1 if the line segments intersect, and 0 otherwise.



Results:



The greedy approach involves choosing the shortest diagonal at each step, which can be done in O(n) time, where n is the number of vertices in the polygon. Since we need to choose n-3 diagonals to triangulate an n-sided polygon, the overall time complexity of the greedy approach is O(n^2).

Greedy algorithm takes even less time than DP approach but we should check whether this gives optimal solution or not.

Question 1-E:

Finally compare the results and suggest whether the DP and Greedy approach results show mismatch.

Procedure:-

In the greedy algorithm, diagonals are added in a specific order, and at each step, the diagonal that has the shortest length and does not intersect any previously added diagonals is selected. However, this may not necessarily result in the minimum cost triangulation.

We produce the results of both DP and Greedy algorithms and check the answers between them, we calculate how many of them are same.

Results:

We then tally the results in a file, from 3 vertices to 15 vertices.
e.g.

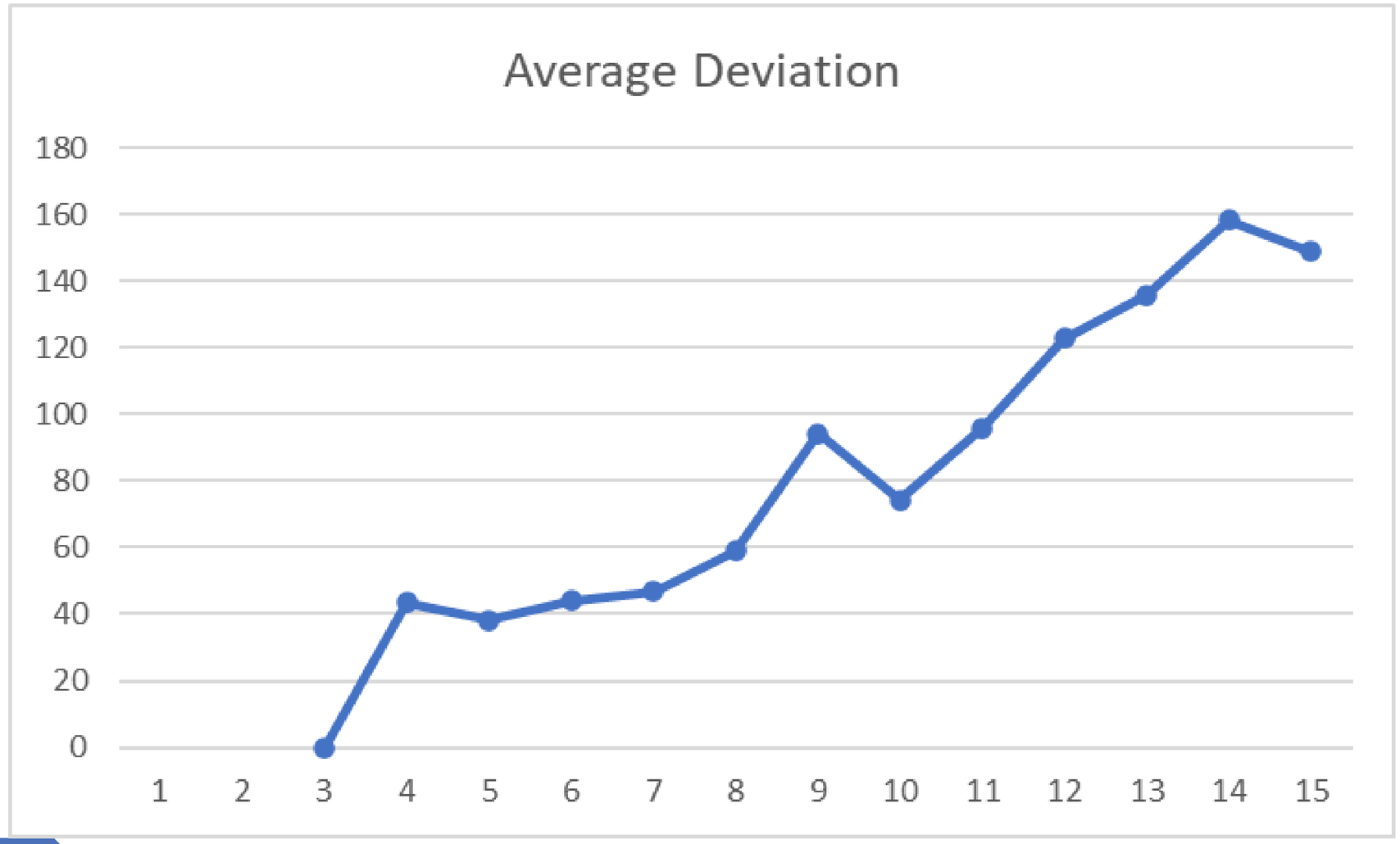
Number of Vertices:9
22,59
41,12
45,27
47,26
63,0
74,93
81,86
82,65
86,19
Cost of Triangulation (Greedy)is 885.657578
Cost of Triangulation (DP) is 770.213056

Results:

We then tally the deviation of the results in a file.
e.g.

Number of Vertices:3
0.000000
Number of Vertices:4
55.324880
Number of Vertices:5
19.638909
Number of Vertices:6
93.204800
Number of Vertices:7
108.169016

Results:


Average Deviation
180
160
140
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Results:

Greedy algorithms make locally optimal choices at each step, without considering the overall optimal solution. In some cases, these locally optimal choices can lead to the overall optimal solution. However, in general, dynamic programming algorithms are more reliable and are guaranteed to find the optimal solution. They consider all possible choices and make decisions based on the optimal substructure of the problem.

The reason for this is that the cost of a triangulation depends not only on the length of the diagonals but also on the angles between them. In a greedy approach, we choose the shortest diagonal at each step, which may not necessarily be the diagonal that leads to the minimum cost triangulation.

But the performance of an algorithm can also depend on the specific input data. It’s possible that the input data used favors the greedy approach.

Question 2-A:

Implement data compression strategies: -

- Write encoding algorithms for the Shannon-Fano coding scheme.

What is it?

Shannon Fano Algorithm is an entropy encoding technique for lossless data compression of multimedia. Named after Claude Shannon and Robert Fano, it assigns a code to each symbol based on their probabilities of occurrence. It is a variable-length encoding scheme, that is, the codes assigned to the symbols will be of varying lengths.

How it works?

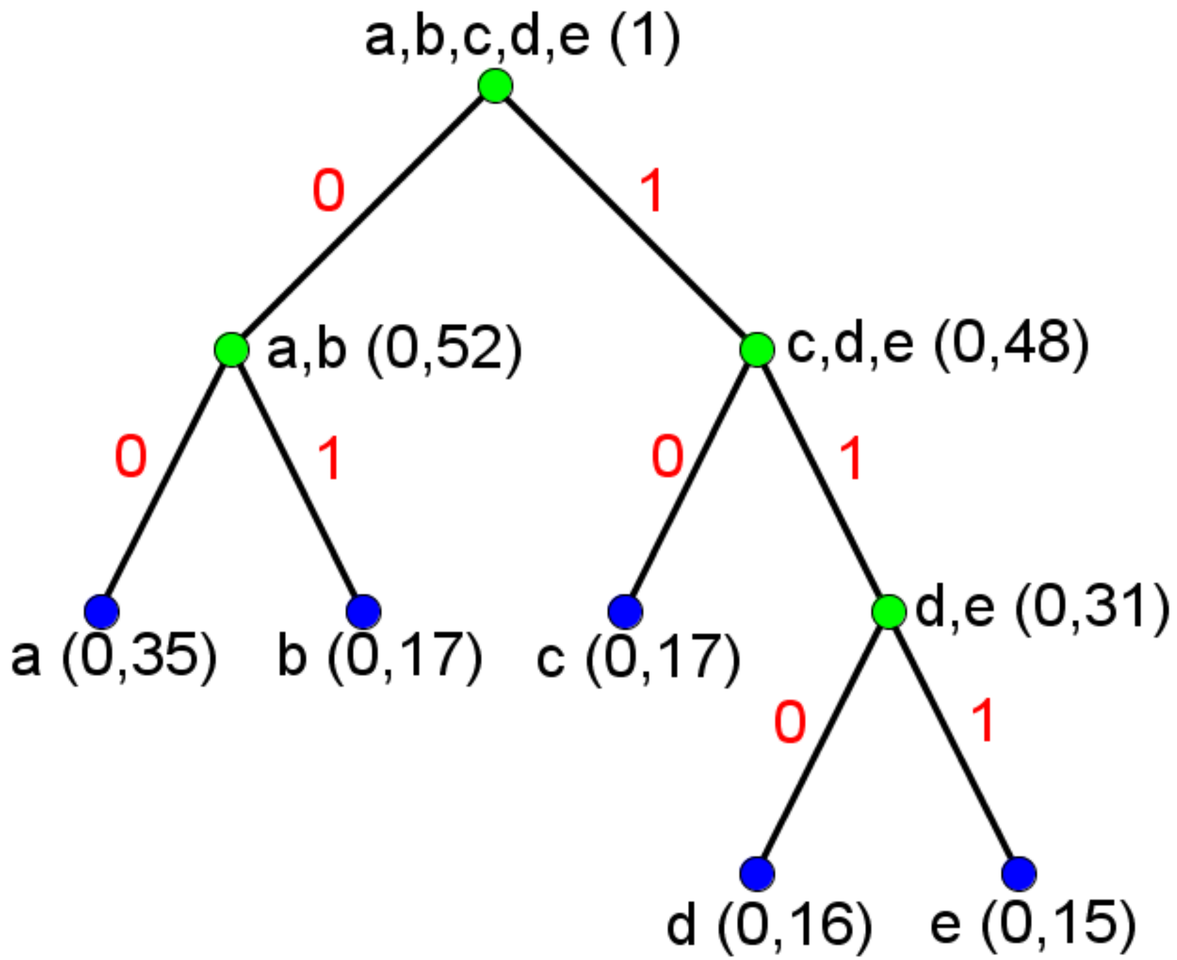
The steps of the algorithm are as follows:-

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.


a,b,c,d,e (1)
0
1
a,b (0,52)
c,d,e (0,48)
0
1
0
1
a (0,35)
b (0,17)
c (0,17)
d,e (0,31)
0
1
d (0,16)
e (0,15)

shannonFano.cpp

```cpp
void shannonFano(vector<Symbol>& symbols, int start, int end) {
    if (end - start <= 1) {
        return;
    }
    int mid = start;
    double totalProb = 0.0;
    for (int i = start; i < end; i++) {
        totalProb += symbols[i].probability;
    }
    totalProb /= 2.0;
    while (abs(int(totalProb - symbols[mid].probability)) > abs(int(totalProb - symbols[mid+1].probability))) {
        mid++;
        totalProb -= symbols[mid].probability;
    }
    for (int i = start; i <= mid; i++) {
        symbols[i].code += "0";
    }
    for (int i = mid+1; i < end; i++) {
        symbols[i].code += "1";
    }
    shannonFano(symbols, start, mid+1);
    shannonFano(symbols, mid+1, end);
}
```

- The function shannonFano takes in a vector of Symbol objects, which contain the symbol name and its probability, as well as the start and end indices of the range of symbols to be processed.
- If there are 1 or 0 symbols in the range, there is no need to generate codes, so the function simply returns.
- Otherwise, the function calculates the total probability of the symbols in the range and divides it by 2 to get the target probability for the first half of the range.
- The function then iterates through the symbols in the range, starting from the beginning, and adds up their probabilities until the sum is greater than or equal to the target probability. The index of the last symbol added is the midpoint of the range.

- The function then assigns “0” as the code prefix for all symbols from the start of the range up to (and including) the midpoint, and “1” as the code prefix for all symbols from the midpoint+1 up to the end of the range.
- Finally, the function recursively calls itself on the first half of the range (from the start to the midpoint+1) and the second half of the range (from the midpoint+1 to the end).

```
unordered_map<char, string> shannonFanoEncode(string input) {
    unordered_map<char, double> symbolProbs;
    for (char c : input) {
        if (symbolProbs.find(c) == symbolProbs.end()) {
            symbolProbs[c] = 0.0;
        }
        symbolProbs[c] += 1.0;
    }
    vector<Symbol> symbols;
    for (auto it : symbolProbs) {
        double prob = it.second / input.length();
        symbols.push_back(Symbol(it.first, prob));
    }
    sort(symbols.begin(), symbols.end(), symbolCompare);
    shannonFano(symbols, 0, symbols.size());
    unordered_map<char, string> codewords;
    for (Symbol s : symbols) {
        codewords[s.symbol] = s.code;
    }
    return codewords;
}
```

- The function first creates an unordered map **symbolProbs** to store the probability of each character in the input string. It iterates through each character in the string and increments the count for that character in the map.
- The function then creates a vector of **Symbol** objects, where each object contains the character and its probability. The probability is calculated by dividing the count of the character by the length of the input string.
- The vector of Symbol objects is sorted in descending order of probability using the **symbolCompare** function.
- The **shannonFano** function is called on the vector of Symbol objects, with the start index as 0 and the end index as the size of the vector.

- The **shannonFano** function generates variable-length codes for each symbol in the vector using the Shannon-Fano algorithm, and stores the codes in the code field of each Symbol object.
- Finally, the function creates an unordered map **codewords** to store the code for each character. It iterates through the vector of Symbol objects and adds each character and its code to the map.

Question 2-B:

Implement data compression strategies: -

- Implement Huffman encoding scheme using heap as data structure..

What is it?

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

How Does it work?

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

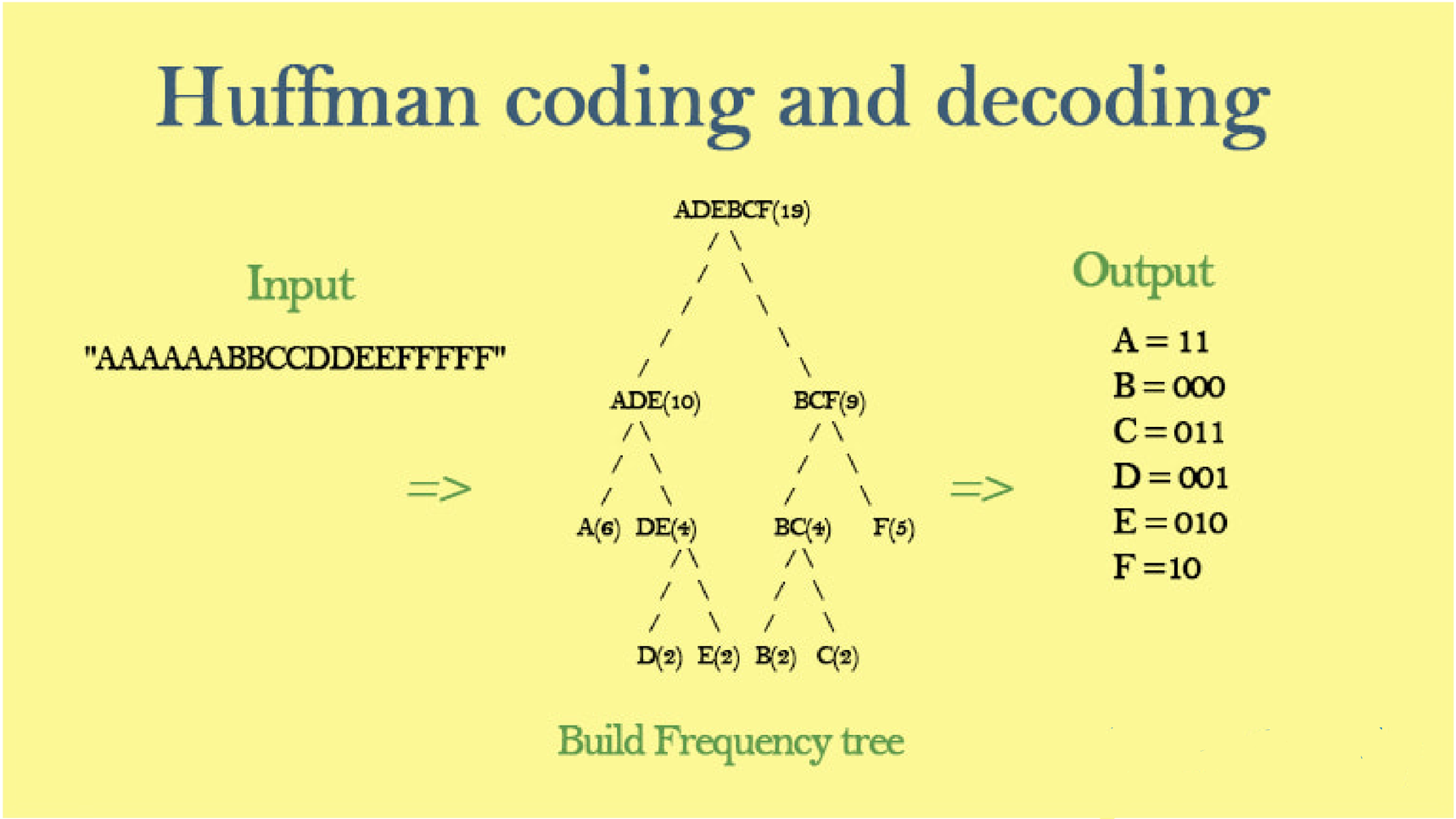
1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Algorithm:-

The method which is used to construct optimal prefix code is called Huffman coding. This algorithm builds a tree in bottom up manner. We can denote this tree by T.Let, |c| be number of leaves.|c| -1 are number of operations required to merge the nodes. Q be the priority queue which can be used while constructing binary heap.

```
Algorithm Huffman (c)
{
  n= |c|
  Q = c
  for i<-1 to n-1
  do
  {
     temp <- get node ()
    left (temp] Get_min (Q) right [temp] Get Min (Q)
    a = left [templ b = right [temp]
    F [temp]<- f[a] + [b]
    insert (Q, temp)
   }
return Get_min (0)
}
```

Algorithm:-


Huffman coding and decoding
Input
"AAAAAABBCCDDEEFFFFF"
=>
ADEBCF(19)
ADE(10)
BCF(9)
A(6)
DE(4)
BC(4)
F(5)
D(2)
E(2)
B(2)
C(2)
Build Frequency tree
=>
Output
A = 11
B = 000
C = 011
D = 001
E = 010
F = 10

```
string encode(string text) {
    priority_queue<Node*, vector<Node*>, Compare> pq;

    unordered_map<char, int> freq;

    for (char c : text) {
        freq[c]++;
    }
    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }
    while (pq.size() > 1) {
        Node* left = pq.top();
        pq.pop();
        Node* right = pq.top();
        pq.pop();
        int sum = left->frequency + right->frequency;
        pq.push(new Node(sum, left, right));
    }
    Node* root = pq.top();
    pq.pop();
    generateCodes(root, "");
    string encoded = "";
    for (char c : text) {
        encoded += codes[c];
        encoded += " ";
    }
    return encoded;
}
```

- The function **encode** takes in a string **text** and first creates a priority queue **pq** to store Node objects. It also creates an unordered map **freq** to store the frequency of each character in the string.
- The function iterates through each character in the string and increments the count for that character in the **freq** map.
- The function then creates a Node object for each character in the **freq** map and adds it to the priority queue **pq**. Each Node object contains the character, its frequency, and pointers to its left and right child nodes (initially set to nullptr).

- The function then repeatedly extracts the two nodes with the lowest frequency from the priority queue, creates a new Node object with their sum as the frequency and the two nodes as its left and right children, and adds the new Node object back to the priority queue. This process continues until there is only one node left in the priority queue, which becomes the root of the Huffman tree.
- The function then calls the **generateCodes** function on the root node to generate the variable-length codes for each character in the tree. The codes are stored in the global **codes** map, where the key is the character and the value is its code.
- Finally, the function iterates through each character in the input string and appends its code (retrieved from the codes map) to the encoded string, separated by a space.

Question 2-C:

Implement data compression strategies: -

- Implement an instantaneous decoding algorithm for Huffman encoding scheme.

huffman_decode.cpp

```cpp
string decode(Node* root, string encoded) {
    cout<<"inside decoded";
    Node* curr = root;
    string decoded = "";
    for (char c : encoded) {
        if (c == '0') {
            curr = curr->left;
        } else {
            curr = curr->right;
        }
        if (is_leaf_node(curr)) {
            decoded += curr->data;
            curr = root;
        }
    }
    return decoded;
}
```

What is it?

The decode function takes two arguments: a pointer to the **root** node of the Huffman tree, and a **string** representing the encoded message. It returns a string representing the decoded message.
The function first initializes a pointer **curr** to the root node of the Huffman tree, and an empty string **decoded** to store the decoded message.
The function then iterates through each character in the encoded message. For each character, the function checks whether it is a ‘0’ or a ‘1’. If it is a ‘0’, the function moves curr to its left child. If it is a ‘1’, the function moves curr to its right child.
The function then checks whether curr is a leaf node (i.e., it has no children). If curr is a leaf node, it means that we have reached a character in the original message, so we append that character to the decoded string and reset curr to the root node of the Huffman tree.
Finally, the function returns the decoded string.

Question 2-D:

Choose a large text document and get results of Huffman coding with this document.

Document_1:

Summer vacation is a much-awaited time for students of all ages. For college students, it is a much-needed break from the hustle and bustle of academic life. One particular day of my summer vacation as a college student stands out in my memory as a day full of fun and adventure. I woke up early in the morning, feeling rejuvenated after a good night's sleep. I decided to start the day with some exercise, so I went for a jog around my neighborhood. The warm morning breeze and the lush green surroundings made it an enjoyable experience. After the jog, I came back home and had a healthy breakfast. After breakfast, I decided to catch up on some reading. I had been wanting to read a book for a while, but academic pressure had kept me busy. So, I settled down on my balcony with a cup of coffee and started reading. The peaceful surroundings, the gentle breeze, and the chirping of birds made it a perfect reading environment. Around noon, I decided to meet up with some of my college friends. We planned to go to the beach and spend the day there. We packed some snacks, sunscreen, and towels and headed out. The beach was crowded, but we managed to find a spot and set up our things. We spent the day swimming, sunbathing, and playing beach volleyball. We even tried parasailing, which was an exhilarating experience. In the evening, we decided to grab some food from a nearby food truck. We tried some delicious seafood and watched the sunset while sitting on the beach. It was a magical experience, and I felt grateful for being able to spend such a wonderful day with my friends. After returning home, I decided to catch up on some movies that I had been meaning to watch. I curled up in my bed with some popcorn and started watching. The movie was engaging, and I lost track of time.

1100000101 01100 01101 01101 010 11010 111 0000001 1011 01111 1011 1000 11011 0011 1001 111 11011 11001 111 1011 111 01101 01100 01111 10101 1100000110 1011 101000 1011 11011 1000 010 0010 111 1000 11011 01101 010 111 110001 0011 11010 111 11001 1000 01100 0010 010 1001 1000 11001 111 0011 110001 111 1011 00010 00010 111 1011 00001 010 11001 011100 111 1100001111 0011 11010 111 01111 0011 00010 00010 010 00001 010 111 11001 1000 01100 0010 010 1001 1000 11001 000001 111 11011 1000 111 11011 11001 111 1011 111 01101 01100 01111 10101 1100000110 1001 010 010 0010 010 0010 111 101001 11010 010 1011 0001101 111 110001 11010 0011 01101 111 1000 10101 010 111 10101 01100 11001 1000 00010 010 111 1011 1001 0010 111 101001 01100 11001 1000 00010 010 111 0011 110001 111 1011 01111 1011 0010 010 01101 11011 01111 111 00010 11011 110001 010 011100 111 1100000000 1001 010 111 011101 1011 11010 1000 11011 01111 01100 00010 1011 11010 111 0010 1011 000111 111 0011 110001 111 01101 000111 111 11001 01100 01101 01101 010 11010 111 0000001 1011 01111 1011 1000 11011 0011 1001 111 1011 11001 111 1011 111 01111 0011 00010 00010 010 00001 010 111 11001 1000 01100 0010 010 1001 1000 111 11001 1000 1011 1001 0010 11001 111 0011 01100 1000 111 11011 1001 111 01101 000111 111 01101 010 01101 0011 11010 000111 111 1011 11001 111 1011 111 0010 1011 000111 111 110001 01100 00010 00010 111 0011 110001 111 110001 01100 1001 111 1011 1001 0010 111 1011 0010 0000001 010 1001 1000 01100 11010 010 011100 1100001 111 101000 0011 0001101 010 111 01100 011101 111 010 1011 11010 00010 000111 111 11011 1001 111 1000 10101 010 111 01101 0011 11010 1001 11011 1001 00001 000001 111 110001 010 010 00010 11011 1001 00001 111 11010 010 00000000 01100 0000001 010 1001 1011 1000 010 0010 111 1011 110001 1000 010 11010 111 1011 111 00001 0011 0011 0010 111 1001 11011 00001 10101 1000 1100001110 11001 111 11001 00010 010 010 011101 011100 111 1100001 111 0010 010 01111 11011 0010 010 0010 111 1000 0011 111 11001 1000 1011 11010 1000 111 1000 10101 010 111 0010 1011 000111 111 101000 11011 1000 10101 111 11001 0011 01101 010 111 010 00011000 010 11010 01111 11011 11001 010 000001 111 11001 0011 111 1100001 111 101000 010 1001 1000 111 110001 0011 11010 111 1011 111 00000000 0011 00001 111 1011 11010 0011 01100 1001 0010 111 01101 000111 111 1001 010 11011 00001 10101 101001 0011 11010 10101 0011 0011 0010 011100 111 110000001 10101 010 111 101000 1011 11010 01101 111 01101 0011 11010 1001 11011 1001 00001 111 101001 11010 010 010 1100000100 010 111 1011 1001 0010 111 1000 10101 010 111 00010 01100 11001 10101 111 00001 11010 010 010 1001 111 11001 01100 11010 11010 0011 01100 1001 0010 11011 1001 00001 11001 111 01101 1011 0010 010 111 11011 1000 111 1011 1001 111 010 1001 00000000 0011 000111 1011 101001 00010 010 111 010 00011000 011101 010 11010 11011 010 1001 01111 010 011100 111 0000001 110001 1000 010 11010 111 1000 10101 010 111 00000000 0011 00001 000001 111 1100001 111 01111 1011 01101 010 111 101001 1011 01111 0001101 111 10101 0011 01101 010 111 1011 1001 0010 111 10101 1011 0010 111 1011 111 10101 010 1011 00010 1000 10101 000111 111 101001 11010 010 1011 0001101 110001 1011 11001 1000 011100 00000001 110001 1000 010 11010 111 101001 11010 010 1011 0001101 110001 1011 11001 1000 000001 111 1100001 111 0010 010 01111 11011 0010 010 0010 111 1000 0011 111 01111 1011 1000 01111 10101 111 01100 011101 111 0011 1001 111 11001 0011 01101 010 111 11010 010 1011 0010 11011 1001 00001 011100 111 1100001 111 10101 1011 0010 111 101001 010 010 1001 111 101000 1011 1001 1000 11011 1001 00001 111 1000 0011 111 11010 010 1011 0010 111 1011 111 101001 0011 0011 0001101 111 110001 0011 11010 111 1011 111 101000 10101 11011 00010 010 000001 111 101001 01100 1000 111 1011 01111 1011 0010 010 01101 11011 01111 111 011101 11010 010 11001 11001 01100 11010 010 111 10101 1011 0010 111 0001101 010 011101 1000 111 01101 010 111 101001 01100 11001 000111 011100 111 1100000101 0011 000001 111 1100001 111 11001 010 1000 1000 00010 010 0010 111 0010 0011 101000 1001 111 0011 1001 111 01101 000111 111 101001 1011 00010 01111 0011 1001 000111 111 101000 11011 1000 10101 111 1011 111 01111 01100 011101 111 0011 110001 111 01111 0011 110001 110001 010 010 111 1011 1001 0010 111 11001 1000 1011 11010 1000 010 0010 111 11010 010 1011 0010 11011 1001 00001 011100 111 110000001 10101 010 111 011101 010 1011 01111 010 110001 01100 00010 111 11001 01100 11010 11010 0011 01100 1001 0010 11011 1001 00001 11001 000001 111 1000 10101 010 111 00001 010 1001 1000 00010 010 111 101001 11010 010 010 1100000100 010 000001 111 1011 1001 0010 111 1000 10101 010 111 01111 10101 11011 11010 011101 11011 1001 00001 111 0011 110001 111 101001 11011 11010 0010 11001 111 01101 1011 0010 010 111 11011 1000 111 1011 111 011101 010 11010 110001 010 01111 1000 111 11010 010 1011 0010 11011 1001 00001 111 010 1001 0000001 11011 11010 0011 1001 01101 010 1001 1000 011100 00000001 11010 0011 01100 1001 0010 111 1001 0011 0011 1001 000001 111 1100001 111 0010 010 01111 11011 0010 010 0010 111 1000 0011 111 01101 010 010 1000 111 01100 011101 111 101000 11011 1000 10101 111 11001 0011 01101 010 111 0011 110001 111 01101 000111 111 01111 0011 00010 00010 010 00001 010 111 110001 11010 11011 010 1001 0010 11001 011100 111 0011001 010 111 011101 00010 1011 1001 1001 010 0010 111 1000 0011 111 00001 0011 111 1000 0011 111 1000 10101 010 111 101001 010 1011 01111 10101 111 1011 1001 0010 111 11001 011101 010 1001 0010 111 1000 10101 010 111 0010 1011 000111 111 1000 10101 010 11010 010 011100 111 0011001 010 111 011101 1011 01111 0001101 010 0010 111 11001 0011 01101 010 111 11001 1001 1011 01111 0001101 11001 000001 111 11001 01100 1001 11001 01111 11010 010 010 1001 000001 111 1011 1001 0010 111 1000 0011 101000 010 00010 11001 111 1011 1001 0010 111 10101 010 1011 0010 010 0010 111 0011 01100 1000 011100 111 110000001 10101 010 111 101001 010 1011 01111 10101 111 101000 1011 11001 111 01111 11010 0011 101000 0010 010 0010 000001 111 101001 01100 1000 111 101000 010 111 01101 1011 1001 1011 00001 010 0010 111 1000 0011 111 110001 11011 1001 0010 111 1011 111 11001 011101 0011 1000 111 1011 1001 0010 111 11001 010 1000 111 01100 011101 111 0011 01100 11010 111 1000 10101 11011 1001 00001 11001 011100 111 0011001 010 111 11001 011101 010 1001 1000 111 1000 10101 010 111 0010 1011 000111 111 11001 101000 11011 01101 01101 11011 1001 00001 000001 111 11001 01100 1001 101001 1011 1000 10101 11011 1001 00001 000001 111 1011 1001 0010 111 011101 00010 1011 000111 11011 1001 00001 111 101001 010 1011 01111 10101 111 0000001 0011 00010 00010 010 000111 101001 1011 00010 00010 011100 111 0011001 010 111 010 0000001 010 1001 111 1000 11010 11011 010 0010 111 011101 1011 11010 1011 11001 1011 11011 00010 11011 1001 00001 000001 111 101000 10101 11011 01111 10101 111 101000 1011 11001 111 1011 1001 111 010 00011000 10101 11011 00010 1011 11010 1011 1000 11011 1001 00001 111 010 00011000 011101 010 11010 11011 010 1001 01111 010 011100 1100001 1001 111 1000 10101 010 111 010 0000001 010 1001 11011 1001 00001 000001 111 101000 010 111 0010 010 01111 11011 0010 010 0010 111 1000 0011 111 00001 11010 1011 101001 111 11001 0011 01101 010 111 110001 0011 0011 0010 111 110001 11010 0011 01101 111 1011 111 1001 010 1011 11010 101001 000111 111 110001 0011 0011 0010 111 1000 11010 01100 01111 0001101 011100 111 0011001 010 111 1000 11010 11011 010 0010 111 11001 0011 01101 010 111 0010 010 00010 11011 01111 11011 0011 01100 11001 111 11001 010 1011 110001 0011 0011 0010 111 1011 1001 0010 111 101000 1011 1000 01111 10101 010 0010 111 1000 10101 010 111 11001 01100 1001 11001 010 1000 111 101000 10101 11011 00010 010 111 11001 11011 1000 1000 11011 1001 00001 111 0011 1001 111 1000 10101 010 111 101001 010 1011 01111 10101 011100 111 1100001 1000 111 101000 1011 11001 111 1011 111 01101 1011 00001 11011 01111 1011 00010 111 010 00011000 011101 010 11010 11011 010 1001 01111 010 000001 111 1011 1001 0010 111 1100001 111 110001 010 00010 1000 111 00001 11010 1011 1000 010 110001 01100 00010 111 110001 0011 11010 111 101001 010 11011 1001 00001 111 1011 101001 00010 010 111 1000 0011 111 11001 011101 010 1001 0010 111 11001 01100 01111 10101 111 1011 111 101000 0011 1001 0010 010 11010 110001 01100 00010 111 0010 1011 000111 111 101000 11011 1000 10101 111 01101 000111 111 110001 11010 11011 010 1001 0010 11001 011100 00000001 110001 1000 010 11010 111 11010 010 1000 01100 11010 1001 11011 1001 00001 111 10101 0011 01101 010 000001 111 1100001 111 0010 010 01111 11011 0010 010 0010 111 1000 0011 111 01111 1011 1000 01111 10101 111 01100 011101 111 0011 1001 111 11001 0011 01101 010 111 01101 0011 0000001 11011 010 11001 111 1000 10101 1011 1000 111 1100001 111 10101 1011 0010 111 101001 010 010 1001 111 01101 010 1011 1001 11011 1001 00001 111 1000 0011 111 101000 1011 1000 01111 10101 011100 111 1100001 111 01111 01100 11010 00010 010 0010 111 01100 011101 111 11011 1001 111 01101 000111 111 101001 010 0010 111 101000 11011 1000 10101 111 11001 0011 01101 010 111 011101 0011 011101 01111 0011 11010 1001 111 1011 1001 0010 111 11001 1000 1011 11010 1000 010 0010 111 101000 1011 1000 01111 10101 11011 1001 00001 011100 111 110000001 10101 010 111 01101 0011 0000001 11011 010 111 101000 1011 11001 111 010 1001 00001 1011 00001 11011 1001 00001 000001 111 1011 1001 0010 111 1100001 111 00010 0011 11001 1000 111 1000 11010 1011 01111 0001101 111 0011 110001 111 1000 11011 01101 010 011100 1100000001

Code Table_1:

code_table1.txt

```
  --> 111
f --> 110111
, --> 1101101
x --> 110110011
j --> 110110010
z --> 1101100011
- --> 1101100010
' --> 1101100000
m --> 01010
i --> 0011
o --> 0100
d --> 0010
u --> 00011
l --> 00010
g --> 00000
v --> 0000100
```

code_table1.txt

```
k --> 0000101
y --> 000011
b --> 110000
. --> 010110
s --> 11010
p --> 010111
e --> 011
n --> 1000
t --> 1001
c --> 10100

 --> 1101100001
h --> 10101
a --> 1011
w --> 110001
r --> 11001

```

Question 2-E:

Examine the optimality of Huffman codes as data compression strategy by comparing with another document.

Document_2:

Cows are domesticated mammals that belong to the Bovidae family, which also includes goats, sheep, and buffalo. They are known for their gentle nature and are commonly raised for their milk, meat, and leather. Cows are herbivorous animals and graze on grass, hay, and other vegetation. They have a four-chambered stomach that allows them to digest fibrous foods that other animals cannot. Cows can weigh up to 1500 pounds and can stand over six feet tall at the shoulder. They have a distinctive black and white or brown and white pattern on their skin, with each cow's pattern being unique. Cows are social animals and form strong bonds with their herd members. They communicate with each other through a range of vocalizations and body language, including mooing, bellowing, and grooming. Cow's milk is a nutritious food that contains vitamins, minerals, and proteins that are essential for human health. It is commonly used to make cheese, butter, and other dairy products. Cows are milked using a variety of methods, including by hand or with machines. In addition to providing milk and meat, cows are also used for plowing fields and pulling carts in many rural areas. They are known for their strength and stamina, making them valuable working animals. In some cultures, cows are considered sacred and are revered as symbols of wealth and prosperity. Cows have been domesticated for thousands of years and continue to play an important role in many societies around the world. They have been selectively bred for traits such as high milk production and meat quality. However, the intensive farming practices used to raise cows for meat and dairy have come under scrutiny in recent years, with concerns about animal welfare and the environmental impact of large-scale farming operations. Despite these challenges, cows remain an important part of human society and continue to be cherished for their contributions to agriculture, food production, and culture.

Encoded Document_2(Using direct huffman encoding)

00111011 1001 011101 0010 110 1110 0100 000 110 10111 1001 01101 000 0010 1000 0101 01100 1110 1000 000 10111 110 01101 1110 01101 01101 1110 01111 0010 110 1000 11110 1110 1000 110 1111110 000 01111 1001 1010 101100 110 1000 1001 110 1000 11110 000 110 00111010111 1001 1111100 0101 10111 1110 000 110 011100 1110 01101 0101 01111 001111 1111111 110 011101 11110 0101 01100 11110 110 1110 01111 0010 1001 110 0101 1010 01100 01111 00110 10111 000 0010 110 101100 1001 1110 1000 0010 1111111 110 0010 11110 000 000 1111101 1111111 110 1110 1010 10111 110 1111110 00110 011100 011100 1110 01111 1001 1011011 110 101101011 11110 000 001111 110 1110 0100 000 110 0011100 1010 1001 011101 1010 110 011100 1001 0100 110 1000 11110 000 0101 0100 110 101100 000 1010 1000 01111 000 110 1010 1110 1000 00110 0100 000 110 1110 1010 10111 110 1110 0100 000 110 01100 1001 01101 01101 1001 1010 01111 001111 110 0100 1110 0101 0010 000 10111 110 011100 1001 0100 110 1000 11110 000 0101 0100 110 01101 0101 01111 0011100 1111111 110 0010 11110 000 000 1111101 1111111 110 1110 1010 10111 110 1001 1000 11110 000 0100 1011011 110 00111011 1001 011101 0010 110 1110 0100 000 110 11110 000 0100 1111110 0101 1111100 1001 0100 1001 00110 0010 110 1110 1010 0101 01101 1110 01111 0010 110 1110 1010 10111 110 01100 1110 1010 110 0010 1000 1110 1010 10111 110 1001 1111100 000 0100 110 0010 0101 10110100110 110 011100 000 000 1000 110 1000 1110 01111 01111 110 1110 1000 110 1000 11110 000 110 0010 11110 1001 00110 01111 10111 000 0100 1011011 110 101101011 11110 000 001111 110 11110 1110 1111100 000 110 1110 110 10111 0101 0010 1000 0101 1010 01100 1000 0101 1111100 000 110 1111110 01111 1110 01100 0011100 110 1110 1010 10111 110 011101 11110 0101 1000 000 110 1001 0100 110 1111110 0100 1001 011101 1010 110 1110 1010 10111 110 011101 11110 0101 1000 000 110 1111101 1110 1000 1000 000 0100 1010 110 1001 1010 110 1000 11110 000 0101 0100 110 0010 0011100 0101 1010 1111111 110 011101 0101 1000 11110 110 000 1110 01100 11110 110 01100 1001 011101 1011010001 0010 110 1111101 1110 1000 1000 000 0100 1010 110 1111110 000 0101 1010 101100 110 00110 1010 0101 1011010100 00110 000 1011011 110 00111011 1001 011101 0010 110 1110 0100 000 110 0010 1001 01100 0101 1110 01111 110 1110 1010 0101 01101 1110 01111 0010 110 1110 1010 10111 110 011100 1001 0100 01101 110 0010 1000 0100 1001 1010 101100 110 1111110 1001 1010 10111 0010 110 011101 0101 1000 11110 110 1000 11110 000 0101 0100 110 11110 000 0100 10111 110 01101 000 01101 1111110 000 0100 0010 1011011 110 101101011 11110 000 001111 110 01100 1001 01101 01101 00110 1010 0101 01100 1110 1000 000 110 011101 0101 1000 11110 110 000 1110 01100 11110 110 1001 1000 11110 000 0100 110 1000 11110 0100 1001 00110 101100 11110 110 1110 110 0100 1110 1010 101100 000 110 1001 011100 110 1111100 1001 01100 1110 01111 0101 1011010000 1110 1000 0101 1001 1010 0010 110 1110 1010 10111 110 1111110 1001 10111 001111 110 01111 1110 1010 101100 00110 1110 101100 000 1111111 110 0101 1010 01100 01111 00110 10111 0101 1010 101100 110 01101 1001 1001 0101 1010 101100 1111111 110 1111110 000 01111 01111 1001 011101 0101 1010 101100 1111111 110 1110 1010 10111 110 101100 0100 1001 1001 01101 0101 1010 101100 1011011 110 00111011 1001 011101 1011010001 0010 110 01101 0101 01111 0011100 110 0101 0010 110 1110 110 1010 00110 1000 0100 0101 1000 0101 1001 00110 0010 110 011100 1001 1001 10111 110 1000 11110 1110 1000 110 01100 1001 1010 1000 1110 0101 1010 0010 110 1111100 0101 1000 1110 01101 0101 1010 0010 1111111 110 01101 0101 1010 000 0100 1110 01111 0010 1111111 110 1110 1010 10111 110 1111101 0100 1001 1000 000 0101 1010 0010 110 1000 11110 1110 1000 110 1110 0100 000 110 000 0010 0010 000 1010 1000 0101 1110 01111 110 011100 1001 0100 110 11110 00110 01101 1110 1010 110 11110 000 1110 01111 1000 11110 1011011 110 001110100 1000 110 0101 0010 110 01100 1001 01101 01101 1001 1010 01111 001111 110 00110 0010 000 10111 110 1000 1001 110 01101 1110 0011100 000 110 01100 11110 000 000 0010 000 1111111 110 1111110 00110 1000 1000 000 0100 1111111 110 1110 1010 10111 110 1001 1000 11110 000 0100 110 10111 1110 0101 0100 001111 110 1111101 0100 1001 10111 00110 01100 1000 0010 1011011 110 00111011 1001 011101 0010 110 1110 0100 000 110 01101 0101 01111 0011100 000 10111 110 00110 0010 0101 1010 101100 110 1110 110 1111100 1110 0100 0101 000 1000 001111 110 1001 011100 110 01101 000 1000 11110 1001 10111 0010 1111111 110 0101 1010 01100 01111 00110 10111 0101 1010 101100 110 1111110 001111 110 11110 1110 1010 10111 110 1001 0100 110 011101 0101 1000 11110 110 01101 1110 01100 11110 0101 1010 000 0010 1011011 110 001110100 1010 110 1110 10111 10111 0101 1000 0101 1001 1010 110 1000 1001 110 1111101 0100 1001 1111100 0101 10111 0101 1010 101100 110 01101 0101 01111 0011100 110 1110 1010 10111 110 01101 000 1110 1000 1111111 110 01100 1001 011101 0010 110 1110 0100 000 110 1110 01111 0010 1001 110 00110 0010 000 10111 110 011100 1001 0100 110 1111101 01111 1001 011101 0101 1010 101100 110 011100 0101 000 01111 10111 0010 110 1110 1010 10111 110 1111101 00110 01111 01111 0101 1010 101100 110 01100 1110 0100 1000 0010 110 0101 1010 110 01101 1110 1010 001111 110 0100 00110 0100 1110 01111 110 1110 0100 000 1110 0010 1011011 110 101101011 11110 000 001111 110 01111 0101 1010 101100 110 1000 11110 000 0101 0100 110 0010 1000 0100 000 1010 101100 1000 11110 110 1110 1010 10111 110 0010 1000 1110 01101 0101 1010 1110 1111111 110 01101 1110 0011100 0101 1010 101100 110 1000 11110 000 01101 110 1111100 1110 01111 00110 1110 1111110 01111 000 110 011101 1001 0100 0011100 0101 1010 101100 110 1110 1010 0101 01101 1110 01111 0010 1011011 110 001110100 1010 110 0010 1001 01101 000 110 01100 00110 01111 1000 00110 0100 000 0010 1111111 110 01100 1001 011101 0010 110 1110 0100 000 110 01100 1001 1010 0010 0101 10111 000 0100 000 10111 110 0010 1110 01100 0100 000 10111 110 1110 1010 10111 110 1110 0100 000 110 0100 000 1111100 000 0100 000 10111 110 1110 0010 110 0010 001111 01101 1111110 1001 01111 0010 110 1001 011100 110 011101 000 1110 01111 1000 11110 110 1110 1010 10111 110 1111101 0100 1001 0010 1111101 000 0100 0101 1000 001111 1011011 110 00111011 1001 011101 0010 110 11110 1110 1111100 000 110 1111110 000 000 1010 110 10111 1001 01101 000 0010 1000 0101 01100 1110 1000 000 10111 110 011100 1001 0100 110 1000 11110 1001 00110 0010 1110 1010 10111 0010 110 1001 011100 110 001111 000 1110 0100 0010 110 1110 1010 10111 110 01100 1001 1010 1000 0101 1010 00110 000 110 1000 1001 110 1111101 01111 1110 001111 110 1110 1010 110 0101 01101 1111101 1001 0100 1000 1110 1010 1000 110 0100 1001 01111 000 110 0101 1010 110 01101 1110 1010 001111 110 0010 1001 01100 0101 000 1000 0101 000 0010 110 1110 0100 1001 00110 1010 10111 110 1000 11110 000 110 011101 1001 0100 01111 10111 1011011 110 101101011 11110 000 001111 110 11110 1110 1111100 000 110 1111110 000 000 1010 110 0010 000 01111 000 01100 1000 0101 1111100 000 01111 001111 110 1111110 0100 000 10111 110 011100 1001 0100 110 1000 0100 1110 0101 1000 0010 110 0010 00110 01100 11110 110 1110 0010 110 11110 0101 101100 11110 110 01101 0101 01111 0011100 110 1111101 0100 1001 10111 00110 01100 1000 0101 1001 1010 110 1110 1010 10111 110 01101 000 1110 1000 110 1011010100 00110 1110 01111 0101 1000 001111 1011011 110 10110101011 1001 011101 000 1111100 000 0100 1111111 110 1000 11110 000 110 0101 1010 1000 000 1010 0010 0101 1111100 000 110 011100 1110 0100 01101 0101 1010 101100 110 1111101 0100 1110 01100 1000 0101 1001 1010 110 1111101 0100 1110 01100 1000 0101 1001 1010 110 0010 000 10111 110 1000 1001 110 0100 1110 0101 0010 000 110 01100 1001 011101 0010 110 011100 1001 0100 110 1000 11110 1001 00110 0010 1110 1010 10111 0010 110 1001 011100 110 001111 000 1110 0100 0010 110 1110 1010 10111 110 01100 1001 1010 1000 0101 1010 00110 000 110 1000 1001 110 1111101 01111 1110 001111 110 1110 1010 110 0101 01101 1111101 1001 0100 1000 1110 1010 1000 110 0100 1001 01111 000 110 0101 1010 110 01101 1110 1010 001111 110 0010 1001 01100 0101 000 1000 0101 000 0010 110 1110 0100 1001 00110 1010 10111 110 1000 11110 000 110 011101 1001 0100 01111 10111 1011011 110 101101011 11110 000 001111 110 11110 1110 1111100 000 110 1111110 000 000 1010 110 0010 000 01111 000 01100 1000 0101 1111100 000 01111 001111 110 1111110 0100 000 10111 110 011100 1001 0100 110 1000 0100 1110 0101 1000 0010 110 0010 00110 01100 11110 110 1110 0010 110 11110 0101 101100 11110 110 01101 0101 01111 0011100 110 1111101 0100 1001 10111 00110 01100 1000 0101 1001 1010 110 1110 1010 10111 110 01101 000 1110 1000 110 1011010100 00110 1110 01111 0101 1000 001111 1011011 110 10110101011 1001 011101 000 1111100 000 0100 1111111 110 1000 11110 000 110 0101 1010 1000 000 1010 0010 0101 1111100 000 110 011100 1110 0100 01101 0101 1010 101100 110 1111101 0100 1110 01100 1000 0101 1001 1010 110 1111101 0100 1110 01100 1000 0101 1001 1010 110 0010 000 10111 110 1000 1001 110 0100 1110 0101 0010 000 110 01100 1001 011101 0010 110 011100 1001 0100 110 1000 11110 000 0101 0100 110 01100 1001 1010 1000 0100 0101 1111110 00110 1000 0101 1001 1010 0010 110 1000 1001 110 1110 101100 0100 0101 01100 00110 01111 1000 00110 0100 000 1111111 110 011100 1001 1001 10111 110 1111101 0100 1001 10111 00110 01100 1000 0101 1001 1010 1111111 110 1110 1010 10111 110 01100 00110 01111 1000 00110 0100 000 1011011

Code Table_2:

code_table3.txt

```
 1    --> 111
 2  b --> 1101111
 3  p --> 1101110
 4  v --> 1101101
 5  . --> 1101100
 6  h --> 11010
 7  a --> 1100
 8  g --> 101101
 9  w --> 101100
10  t --> 1010
11  - --> 0010111101
12  r --> 0100
13  z --> 0010111100
14  d --> 10111
15  5 --> 0010111010
16  c --> 01111
17  q --> 0010111011
```

code_table3.txt

```
18  , --> 001010
19  1 --> 0010111001
20  x --> 0010111000
21  e --> 000
22  u --> 00100
23  0 --> 0010111111
24  k --> 0010110
25  ' --> 0010111110
26  l --> 01110
27  s --> 0011
28  y --> 010100
29  f --> 010101
30  m --> 01011
31  i --> 0110
32  o --> 1000
33  n --> 1001
```

Encoded Document_2(Using code table of Document_1)

0011 101000 11001 111 1011 11010 010 111 0010 0011 01101 010 11001 1000 11011 01111 1011 1000 010 0010 111 01101 1011 01101 01101 1011 00010 11001 111 1000 10101 1011 1000 111 101001 010 00010 0011 1001 00001 111 1000 0011 111 1000 10101 010 111 0011 0000001 11011 0010 1011 010 111 110001 1011 01101 11011 00010 000111 000001 111 101000 10101 11011 01111 10101 111 1011 00010 11001 0011 111 11011 1001 01111 00010 01100 0010 010 11001 111 00001 0011 1011 1000 11001 000001 111 11001 10101 010 010 011101 000001 111 1011 1001 0010 111 101001 01100 110001 110001 1011 00010 0011 011100 111 110000001 10101 010 000111 111 1011 11010 010 111 0001101 1001 0011 101000 1001 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 00001 010 1001 1000 00010 010 111 1001 1011 1000 01100 11010 010 111 1011 1001 0010 111 1011 11010 010 111 01111 0011 01101 01101 0011 1001 00010 000111 111 11010 1011 11011 11001 010 0010 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 01101 11011 00010 0001101 000001 111 01101 1011 00010 11001 111 1011 1001 0010 111 0011 1000 10101 010 11010 111 0000001 010 00001 010 1000 1011 1000 11011 0011 1001 011100 111 110000001 10101 010 000111 111 10101 1011 0000001 010 111 1011 111 110001 0011 01100 11010 1100000110 01111 10101 1011 01101 101001 010 11010 010 0010 111 11001 1000 0011 01101 1011 01111 10101 111 1000 10101 1011 1000 111 1011 00010 00010 0011 101000 11001 111 1000 10101 010 01101 111 1000 0011 111 0010 11011 00001 010 11001 1000 111 110001 11011 101001 11010 0011 01100 11001 111 110001 0011 0011 0010 11001 111 1000 10101 1011 1000 111 0011 1000 10101 010 11010 111 1011 1001 11011 01101 1011 00010 11001 111 01111 1011 1001 1001 0011 1000 011100 111 0011 101000 11001 111 01111 1011 1001 111 101000 010 11011 00001 10101 111 01100 011101 111 1000 0011 111 111 011101 0011 01100 1001 0010 11001 111 1011 1001 0010 111 01111 1011 1001 111 11001 1000 1011 1001 0010 111 0011 0000001 010 11010 111 11001 11011 00011000 111 110001 010 010 1000 111 1000 1011 00010 00010 111 1011 1000 111 1000 10101 010 111 11001 10101 0011 01100 00010 0010 010 11010 011100 111 110000001 10101 010 000111 111 10101 1011 0000001 010 111 1011 111 0010 11011 11001 1000 11011 1001 01111 1000 11011 0000001 010 111 101001 00010 1011 01111 0001101 111 1011 1001 0010 111 101000 10101 11011 1000 010 111 0011 11010 111 101001 11010 0011 101000 1001 111 1011 1001 0010 111 00001 11010 1011 1100000100 010 111 0011 1001 111 00001 11010 1011 11001 11001 000001 111 10101 1011 000111 000001 111 1011 1001 0010 111 0011 1000 10101 010 11010 111 0000001 010 00001 010 1000 1011 1000 11011 0011 1001 011100 111 110000001 10101 010 000111 111 10101 1011 0000001 010 111 1011 111 110001 0011 11010 01101 111 11001 1000 11010 0011 1001 00001 111 101001 0011 1001 0010 11001 111 101000 10101 11011 1000 010 111 0011 11010 111 101001 11010 0011 101000 1001 111 1011 1001 0010 111 101000 10101 11011 1000 010 111 011101 1011 1000 1000 010 11010 1001 111 0011 1001 111 1000 10101 010 11011 11010 111 11001 0001101 11011 1001 000001 111 101000 11011 1000 10101 111 010 1011 01111 10101 111 01111 0011 101000 11000001110 11001 111 011101 1011 1000 1000 010 11010 1001 111 101001 010 11011 1001 00001 111 01100 1001 11011 01100 010 011100 111 0011 101000 11001 111 1011 11010 010 111 11001 0011 01111 11011 1011 00010 111 1011 1001 1011 01101 11011 1001 00001 111 101001 0011 1001 0010 11001 111 101000 11011 1000 10101 111 1000 10101 010 11011 11010 111 1000 10101 11010 0011 01100 00001 10101 111 1011 111 11010 1011 1001 00001 010 111 0011 110001 111 0000001 0011 01111 1011 00010 11011 1100000100 1011 1000 11011 0011 1001 11001 111 1011 1001 0010 111 101001 0011 0010 000111 111 00010 1011 1001 00001 01100 1011 00001 010 000001 111 11011 1001 01111 00010 01100 0010 11011 1001 00001 111 01101 0011 0011 11011 1001 00001 000001 111 101001 010 00010 00010 0011 101000 11011 1001 00001 000001 111 1011 1001 0010 111 00001 11010 0011 0011 01101 11011 1001 00001 011100 111 0011 101000 11000001110 11001 111 01101 11011 00010 0001101 111 11011 11001 111 1011 111 1001 01100 1000 11010 11011 1000 11011 0011 01100 11001 111 110001 0011 0011 0010 111 1000 10101 1011 1000 111 01111 0011 1001 1000 1011 11011 1001 11001 111 0000001 11011 1000 1011 01101 11011 1001 11001 000001 111 01101 11011 1001 010 11010 1011 00010 11001 000001 111 1011 1001 0010 111 011101 11010 0011 1000 010 11011 1001 11001 111 1000 10101 1011 1000 111 1011 11010 010 111 010 11001 11001 010 1001 1000 11011 1011 00010 111 110001 0011 11010 111 10101 01100 01101 1011 1001 111 10101 010 1011 00010 1000 10101 011100 111 1100001 1000 111 11011 11001 111 01111 0011 01101 01101 0011 1001 00010 000111 111 01100 11001 010 0010 111 1000 0011 111 01101 1011 0001101 010 111 01111 10101 010 010 11001 010 000001 111 101001 01100 1000 1000 010 11010 000001 111 1011 1001 0010 111 0011 1000 10101 010 11010 111 0010 1011 11011 11010 000111 111 011101 11010 0011 0010 01100 01111 1000 11001 011100 111 0011 101000 11001 111 1011 11010 010 111 01101 11011 00010 0001101 010 0010 111 01100 11001 11011 1001 00001 111 1011 111 0000001 1011 11010 11011 010 1000 000111 111 0011 110001 111 01101 010 1000 10101 0011 0010 11001 000001 111 11011 1001 01111 00010 01100 0010 11011 1001 00001 111 101001 000111 111 10101 1011 1001 0010 111 0011 11010 111 101000 11011 1000 10101 111 01101 1011 01111 10101 11011 1001 010 11001 011100 111 1100001 1001 111 1011 0010 0010 11011 1000 11011 0011 1001 111 1000 0011 111 011101 11010 0011 0000001 11011 0010 11011 1001 00001 111 01101 11011 00010 0001101 111 1011 1001 0010 111 01101 010 1011 1000 000001 111 01111 0011 101000 11001 111 1011 11010 010 111 1011 00010 11001 0011 111 01100 11001 010 0010 111 110001 0011 11010 111 011101 00010 0011 101000 11011 1001 00001 111 110001 11011 010 00010 0010 11001 111 1011 1001 0010 111 011101 01100 00010 00010 11011 1001 00001 111 01111 1011 11010 1000 11001 111 11011 1001 111 01101 1011 1001 000111 111 11010 01100 11010 1011 00010 111 1011 11010 010 111 1011 11001 011100 111 110000001 10101 010 000111 111 1011 11010 010 111 0001101 1001 0011 101000 1001 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 11001 1000 11010 010 1001 00001 1000 10101 111 1011 1001 0010 111 11001 1000 1011 01101 11011 1001 1011 000001 111 01101 1011 0001101 11011 1001 00001 111 1000 10101 010 01101 111 0000001 1011 00010 01100 1011 101001 00010 010 111 101000 0011 11010 0001101 11011 1001 00001 111 1011 1001 11011 01101 1011 00010 11001 011100 111 1100001 1001 111 11001 0011 01101 010 111 01111 01100 00010 1000 01100 11010 010 11001 000001 111 01111 0011 101000 11001 111 1011 11010 010 111 01111 0011 01101 010 11001 1000 11011 01111 1011 1000 010 0010 111 110001 0011 11010 111 1000 10101 0011 01100 11001 1011 1001 0010 11001 111 0011 110001 111 000111 010 1011 11010 11001 111 1011 1001 0010 111 01111 0011 1001 1000 11011 1001 01100 010 111 1000 0011 111 011101 00010 1011 000111 111 1011 1001 111 11011 01101 011101 0011 11010 1000 1011 1001 1000 111 11010 0011 00010 010 111 11011 1001 111 01101 1011 1001 000111 111 11001 0011 01111 11011 010 1000 11011 010 11001 111 1011 1001 0010 111 1011 11010 010 111 01111 0011 101000 11001 111 10101 1011 0000001 010 111 101001 010 010 1001 111 0010 0011 01101 010 11001 1000 11011 01111 1011 1000 010 0010 111 110001 0011 11010 111 1000 10101 0011 01100 11001 1011 1001 0010 11001 111 0011 110001 111 000111 010 1011 11010 11001 111 1011 1001 0010 111 01111 0011 1001 1000 11011 1001 01100 010 111 1000 0011 111 011101 00010 1011 000111 111 1011 1001 111 11011 01101 011101 0011 11010 1000 1011 1001 1000 111 11010 0011 00010 010 111 11011 1001 111 01101 1011 1001 000111 111 11001 0011 01111 11011 010 1000 11011 010 11001 111 1011 11010 0011 01100 1001 0010 111 1000 10101 010 111 01111 0011 1001 1000 11011 1001 01100 010 111 1000 0011 111 101001 010 111 01111 10101 010 11010 11011 11001 10101 010 0010 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 01111 0011 1001 1000 11010 11011 101001 01100 1000 11011 0011 1001 11001 111 1000 0011 111 1011 00001 11010 11011 01111 01100 00010 1000 01100 11010 010 011100 111 0011 101000 11001 111 1011 11010 010 111 01101 11011 00010 0001101 010 0010 111 01100 11001 11011 1001 00001 111 1011 111 0000001 1011 11010 11011 010 1000 000111 111 0011 110001 111 01101 010 1000 10101 0011 0010 11001 000001 111 11011 1001 01111 00010 01100 0010 11011 1001 00001 111 101001 000111 111 10101 1011 1001 0010 111 0011 11010 111 101000 11011 1000 10101 111 01101 1011 01111 10101 11011 1001 010 11001 011100 111 1100001 1001 111 1011 0010 0010 11011 1000 11011 0011 1001 111 1000 0011 111 011101 11010 0011 0000001 11011 0010 11011 1001 00001 111 01101 11011 00010 0001101 111 1011 1001 0010 111 01101 010 1011 1000 000001 111 01111 0011 101000 11001 111 1011 11010 010 111 1011 00010 11001 0011 111 01100 11001 010 0010 111 110001 0011 11010 111 011101 00010 0011 101000 11011 1001 00001 111 110001 11011 010 00010 0010 11001 111 1011 1001 0010 111 011101 01100 00010 00010 11011 1001 00001 111 01111 1011 11010 1000 11001 111 11011 1001 111 01101 1011 1001 000111 111 11010 01100 11010 1011 00010 111 1011 11010 010 111 1011 11001 011100 111 110000001 10101 010 000111 111 1011 11010 010 111 0001101 1001 0011 101000 1001 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 11001 1000 11010 010 1001 00001 1000 10101 111 1011 1001 0010 111 11001 1000 1011 01101 11011 1001 1011 000001 111 01101 1011 0001101 11011 1001 00001 111 1000 10101 010 01101 111 0000001 1011 00010 01100 1011 101001 00010 010 111 101000 0011 11010 0001101 11011 1001 00001 111 1011 1001 11011 01101 1011 00010 11001 011100 111 1100001 1001 111 11001 0011 01101 010 111 01111 01100 00010 1000 01100 11010 010 11001 000001 111 01111 0011 101000 11001 111 1011 11010 010 111 01111 0011 01101 010 11001 1000 11011 01111 1011 1000 010 0010 111 110001 0011 11010 111 1000 10101 0011 01100 11001 1011 1001 0010 11001 111 0011 110001 111 000111 010 1011 11010 11001 111 1011 1001 0010 111 01111 0011 1001 1000 11011 1001 01100 010 111 1000 0011 111 011101 00010 1011 000111 111 1011 1001 111 11011 01101 011101 0011 11010 1000 1011 1001 1000 111 11010 0011 00010 010 111 11011 1001 111 01101 1011 1001 000111 111 11001 0011 01111 11011 010 1000 11011 010 11001 111 1011 11010 0011 01100 1001 0010 111 1000 10101 010 111 010 1001 0000001 11011 11010 0011 1001 01101 010 1001 1000 1011 00010 111 11011 01101 011101 1011 01111 1000 111 0011 110001 111 00010 1011 11010 00001 010 1100000110 11001 01111 1011 00010 010 111 110001 1011 11010 01101 11011 1001 00001 111 0011 011101 010 11010 1011 1000 11011 0011 1001 11001 111 01111 1000 10101 010 11001 010 111 01111 10101 1011 00010 00010 010 1001 00001 010 11001 000001 111 01111 0011 101000 11001 111 11010 010 01101 1011 11011 1001 111 1011 1001 111 11011 01101 011101 0011 11010 1000 1011 1001 1000 111 011101 1011 11010 1000 111 0011 110001 111 10101 01100 01101 1011 1001 111 11001 0011 01111 11011 010 1000 000111 111 1011 1001 0010 111 01111 0011 1001 1000 11011 1001 01100 010 111 1000 0011 111 101001 010 111 01111 10101 010 11010 11011 11001 10101 010 0010 111 110001 0011 11010 111 1000 10101 010 11011 11010 111 01111 0011 1001 1000 11010 11011 101001 01100 1000 11011 0011 1001 11001 111 1000 0011 111 1011 00001 11010 11011 01111 01100 00010 1000 01100 11010 010 011100 111 110001 0011 0011 0010 111 011101 11010 0011 0010 01100 01111 1000 11011 0011 1001 000001 111 1011 1001 0010 111 01111 01100 00010 1000 01100 11010 010 011100

Optimality:

Encoding Document_2 using Code table of Document_1

```
Length from direct huffman coding = 8479
Length from using code table of another doc = 8385
Optimality incresed by 1.10862%
```

Encoding Document_1 using Code table of Document_2

```
Length from direct huffman coding = 8313
Length from using code table of another doc = 10363
Optimality decresed by 19.7819%
```

Question 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.

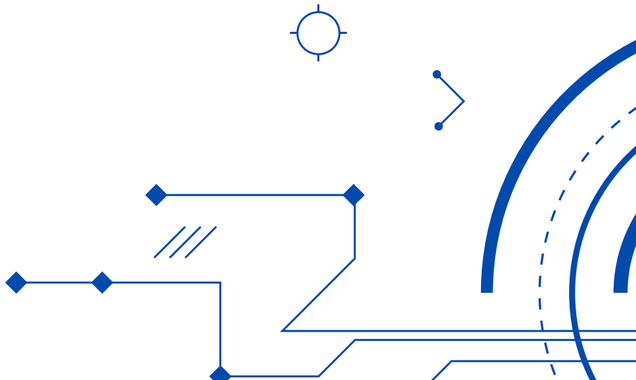
(b) KONECT – Koblenz Network Collection – datasets compiled by Koblenz University, Deutschland (Germany).

What is it?

The dataset has given us n vertices and m predefined edges in a graph. All the edges are listed as a b;
It means there is a edge from a to b.

We are going to use put these data in a disjoint set method by using two methods, those are:

1. Connected Components using Linked List
2. Tree implementation



Datasets:

We are using 4 datasets:

Two from SNAP:

1. ego-Facebook : https://snap.stanford.edu/data/ego-Facebook.html
2. ego-Twitter : https://snap.stanford.edu/data/ego-Twitter.html

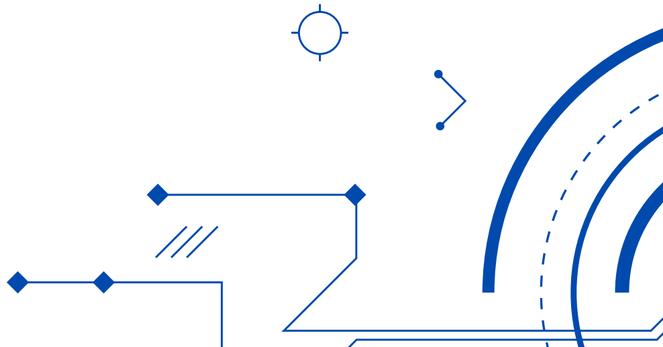
Two from KONECT:

1. Linux Sources : http://konect.cc/networks/linux/
2. Bitcoin Alpha : http://konect.cc/networks/soc-sign-bitcoinalpha/

Connected Component:

1. Create a linked list of nodes with the size of the number of vertices in the graph.
2. Initialize each node as a separate component using MAKE_SET() function.
3. Read the edges of the graph from the dataset.
4. For each edge, call the UNION() function to combine the sets of the two nodes that the edge connects.
5. After processing all the edges, each connected component will be represented by a set of nodes whose parents are the same.
6. To find the connected component of a node, use FIND_SET() to find the root of the tree and return all the nodes whose roots are the same.

Note that the algorithm assumes that the input graph is undirected. If the graph is directed, it is needed to process each edge in both directions.



Code Snippets:

```
class Node {
public:
    Node* next;
    CC* rep;
    unsigned long data;

    Node(unsigned long data) {
        this->data = data;
        next = nullptr;
        rep = nullptr;
    }

    friend class UnionFind;
};
```

```
class Edge {
private:
    unsigned long value1;
    unsigned long value2;
public:
    Edge() {
        value1 = 0;
        value2 = 0;
    }

    friend class UnionFind;
};
```

Code Snippets:

```
class CC {
public:
    Node* head;
    Node* tail;
    CC* next;
    CC* prev;
    unsigned long len;

    CC() {
        head = nullptr;
        tail = nullptr;
        next = nullptr;
        prev = nullptr;
        len = 0;
    }

    friend class UnionFind;
};
```

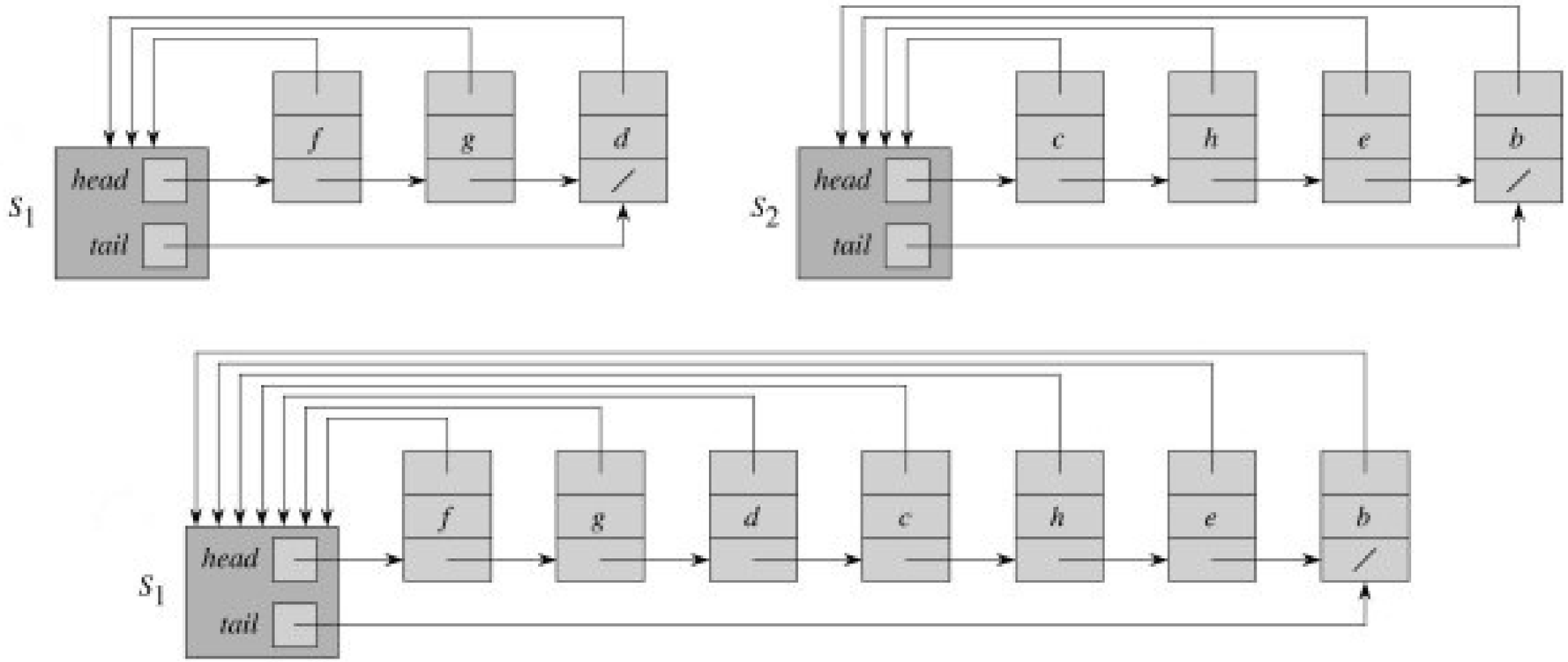
```
class UnionFind {
private:
    CC start;

public:

    ~UnionFind() {
        CC* temp = &start;
        while (temp != nullptr) {
            CC* next = temp->next;
            Node* node_temp = temp->head;
            while (node_temp != nullptr) {
                Node* next = node_temp->next;
                delete node_temp;
                node_temp = next;
            }
            delete temp;
            temp = next;
        }
    }
```

Code Snippets:

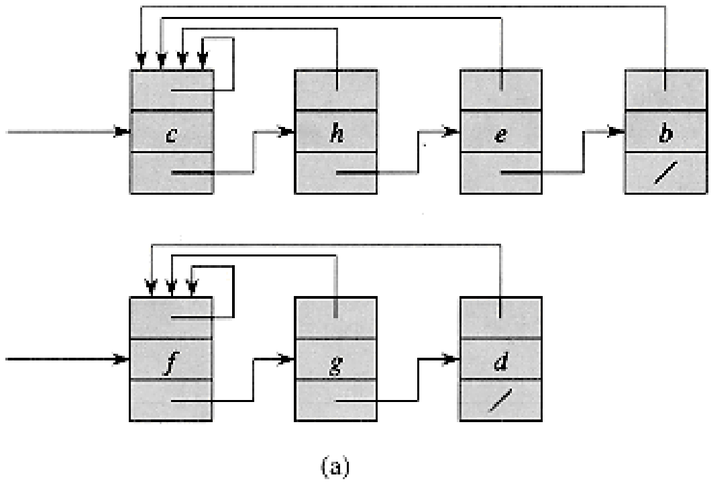
```
void make_set(unsigned long x) {
    CC* temp = start.next;

    CC* new_CC = new CC();
    new_CC->len = 1;
    new_CC->next = temp;
    start.next = new_CC;
    new_CC->prev = &start;

    if (temp != nullptr)
        temp->prev = new_CC;

    new_CC->head = new Node(x);
    new_CC->tail = new_CC->head;
    new_CC->head->rep = new_CC;
}
```

```
void union_op(Edge temp) {
        if (temp.value1 != temp.value2)
        link(find_set(temp.value1), find_set(temp.value2));
    }

    void update_edge(Edge& given_edge, std::ifstream& fin) {
        if (!fin.eof()){
            char line[10];
            fin.getline(line, 10);

            stringstream str_strm;
            str_strm << line; //convert the string s into stringstream

            char num[5];

            str_strm >> num;
            given_edge.value1 = atoi(num);

            str_strm >> num;
            given_edge.value2 = atoi(num);

            // cout << given_edge.value1 << " " << given_edge.value2 << endl;
        }
    }
```

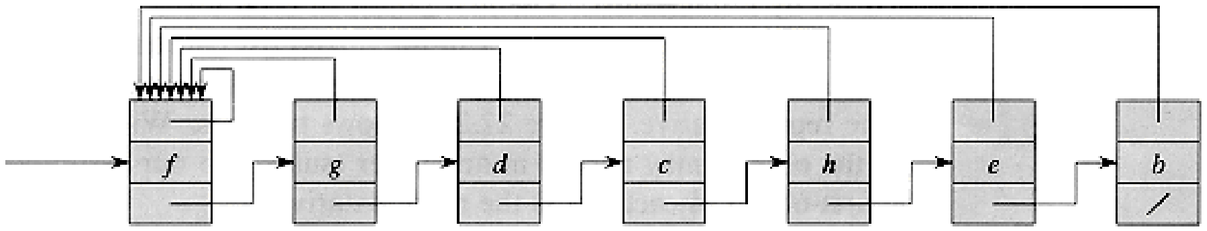
What is it?



What is it?



(a)



(b)

Union function of linked list implementation of disjoint sets.

Tree Implementation

1. Start by creating an empty tree. Make vertices for number of vertices.
2. Iterate through each edge in the dataset.
3. If the two vertices of the edge are in different subtrees of the tree, connect the two subtrees by making one of the vertices the child of the other.
4. If the two vertices of the edge are already in the same subtree, do not connect them to avoid creating a cycle.
5. Continue iterating through the edges until all have been processed.
6. Once all edges have been processed, the resulting tree represents the connected components of the graph.
7. To find the connected components, iterate through the nodes of the tree and find the root of each subtree.
8. Nodes with the same root belong to the same connected component.

Code Snippets:

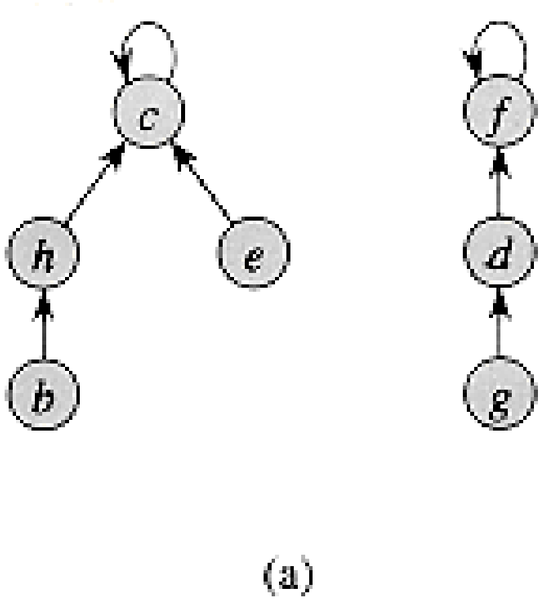
```
class disjointSetUnion
{
    vector<int> parent;
    vector<int> size;

public:
    disjointSetUnion();
    int find_set(int x);
    void setUnion(int x, int y);
};

disjointSetUnion::disjointSetUnion()
{
    parent = vector<int>(numVertices);

    for (int i = 0; i < parent.size(); i++)
        parent[i] = i;

    size = vector<int>(numVertices, 1);
}
```

Code Snippets:

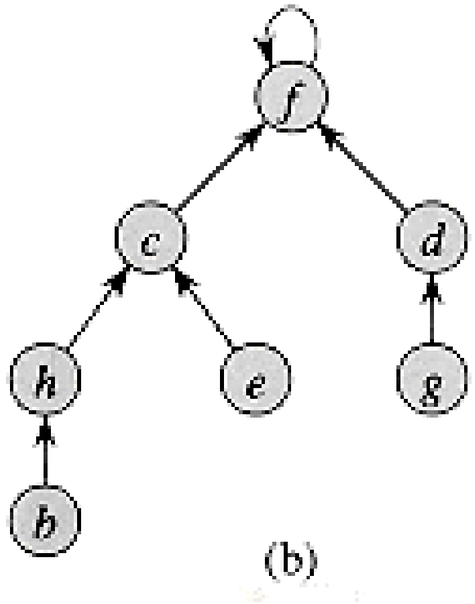
```
int disjointSetUnion::find_set(int x)
{
    if (parent[x] != x)
        parent[x] = find_set(parent[x]);

    return parent[x];
}

void disjointSetUnion::setUnion(int x, int y)
{
    int parX = find_set(x);
    int parY = find_set(y);

    if (parX != parY)
    {
        if (size[x] < size[y])
            swap(x, y);

        // now confirmed x is the bigger tree

        parent[y] = x;
        size[x] += size[y];
    }
}
```

```
pair<int, int> getNextEdge(ifstream &fin)
{
    if (!fin.eof())
    {
        char line[100];
        fin.getline(line, 100);

        if (line[0] == '#')
        {
            while (line[0] == '#')
                fin.getline(line, 100);

            stringstream str_strm;
            str_strm << line; //convert the string s into stringstream

            char num[10];

            str_strm >> num;
            int val1 = atoi(num);
            str_strm >> num;
            int val2 = atoi(num);

            return make_pair(val1, val2);
        }
    }
}
```

What is it?



Disjoint sets in tree



Union of disjoint set

Results:

Bitcoin Dataset:
num_vertex = 3783;
num_edges = 24186;

Bitcoin List Approach:
Average Time Taken: 6575ms
Bitcoin Tree Approach:
Average Time Taken: 6351ms

Facebook Dataset:
num_vertex = 4039;
num_edges = 88234;

Facebook List Approach:
Average Time Taken: 22823ms
Facebook Tree Approach:
Average Time Taken: 21632ms

Linux Dataset:
num_vertex = 30837;
num_edges = 213954;

Linux List Approach:
Average Time Taken: 89370.4ms
Linux Tree Approach:
Average Time Taken: 88637ms

Twitter Dataset:
num_vertex =81306
num_edges = 1768149;

Twitter List Approach:
Average Time Taken: 617960ms
Twitter Tree Approach:
Average Time Taken: 564233ms