National Institute of Technology, Hamirpur

Department of Computer Science and Engineering.

www.nith.ac.in

DataStructure Project

Title: Encryption Algorithm Using Graph Theory

Made by:

185058 Avinash Yadav

Email: ay22082000@gmail.com

Phone No: 8840517052

185070 Sachin Wattamwar

Email: sachin.w2000@gmail.com

Phone No: 7767065555

185088 Wable Sanket

Email: sanketwable123@gmail.com

Phone No: 9370173013

Under the guidance of :

Prof. Nitin Gupta

1. Aim :

- To generate an Algorithm for data encryption and decryption using Graph Theory.
- To implement knowledge of how new symmetric encryption algorithm uses the concepts of cycle graph, complete graph and Kruskal's minimum spanning tree to generate a complex cipher text using a shared key

2. Objectives and learning Outcomes :

- At the end of project :
  - We will get acquainted with different concepts of Graphs such as complete Graph, Minimum Spanning Tree, Depth first Search, and more Graphs concepts.
  - We will be familiar with programming graphs in C++, eg: Adding Vertices, Adding Edges, Adding Weights.
  - Representation of Graphs using Adjacency Matrix and Adjacency list.

3. Proposed Algorithm :

The first step in this algorithm is to represent data as vertices in a graph, each character represented by a vertex while all adjacent characters in the data will be represented as adjacent vertices in the graph, we keep adding vertices until we form a cycle graph.

Every edge in the graph has its own weight representing the distance of these two characters in the encoding table.

Encoding Table

A	B	C	D	…….	Y	Z
1	2	3	4	…….	25	26

Graph will join with edges to make the graph a complete graph, while every new added edge has a weight starting from the 27 and adding 1 to it . Adjacency
Matrix is constructed for the complete graph. After that Minimum Spanning Tree (MST) is computed from the complete graph and represented as an adjacency matrix that keeps data characters ordered in its diagonal. The resultant matrix multiplied to the key matrix. The final matrix is the encryption data to be sent to the recipient.

Distance = code(A) - code(W)
= 1 – 23 = -22

Encryption Algorithm:

- Add a special character to indicate the starting character (Let A).
- Add vertex for each character in the plain text to the graph.
- Link vertices together by adding an edge between each sequential character in the plain text until we form a cycle graph.
- Weight each edge using the encoding table. Each edge's weight
  represents the distance between the connected two vertices from the encoding table.
- Adding more edges to form a complete graph M1, each new added edge has a sequential weight starting from the maximum weight in the encoding table.
- Then find the Minimum Spanning Tree M2.
- Then store the vertices order in the M2 matrix in the diagonal places.
- Then we multiply matrices M1 by M2 to get M3.
- After that we multiply M3 by a predefined Shared-Key K to form C.
- Then the cipher text contains Matrix C and Matrix M1 line-by-line in a linear format.

Decryption Algorithm:

- The receiver computes M3 by using the inverse form of the Shared-Key K-1.
- Then compute M2 by using the inverse form of M1.
- Then compute the original text by decoding M1 using the encoding table.

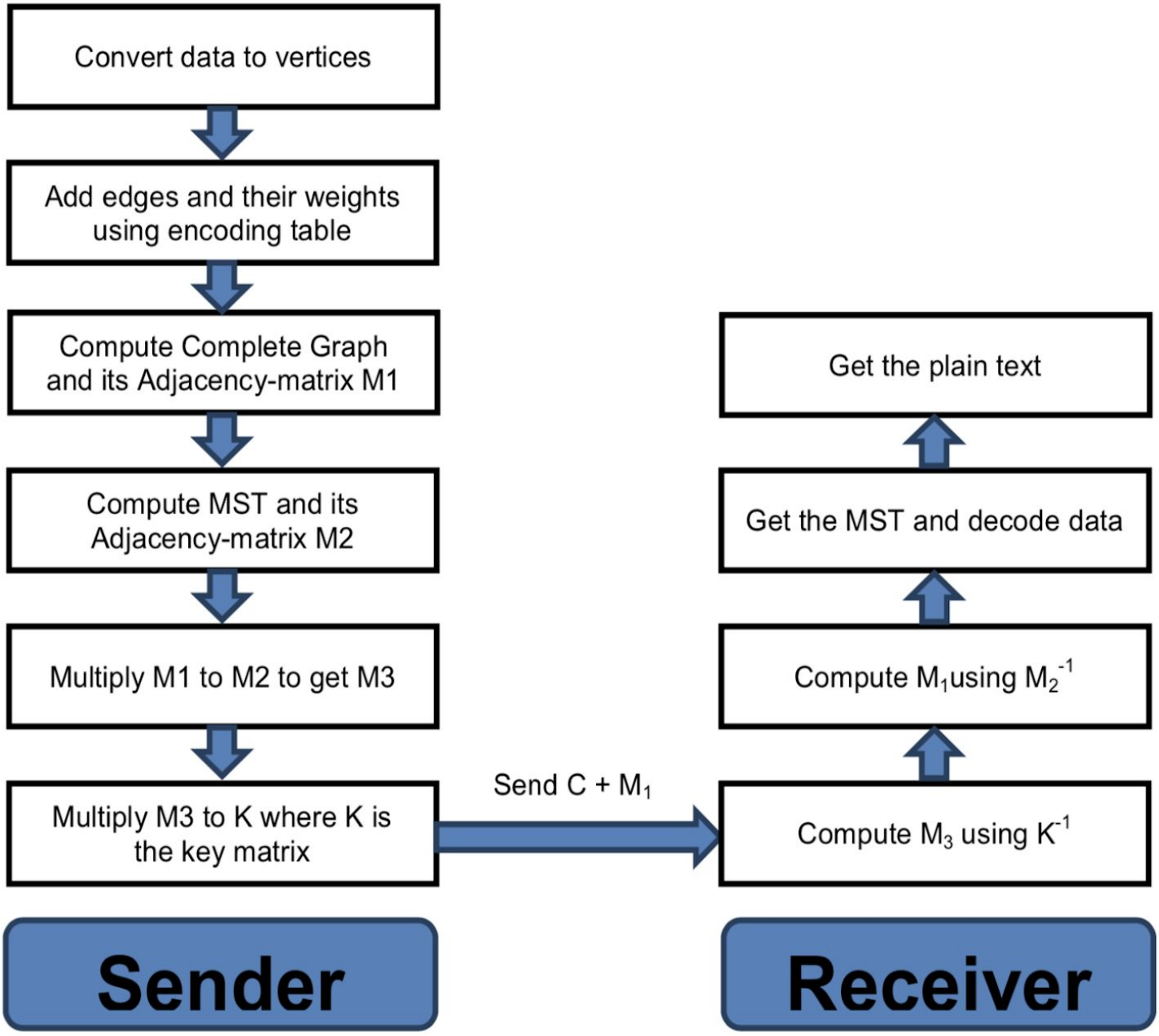
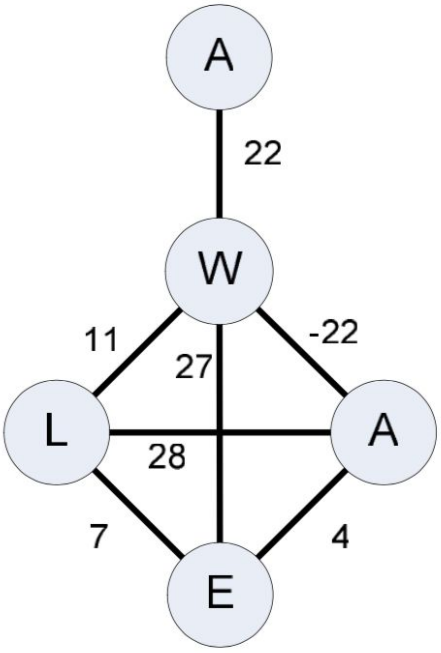


Fig. 1. Algorithm summary

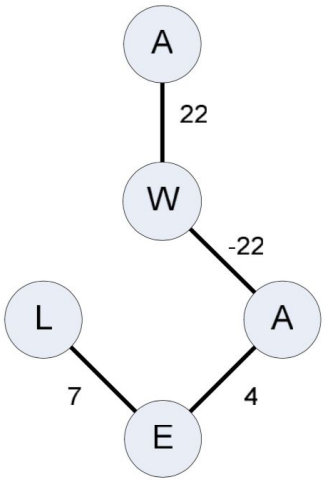
Encryption:

Encrypting Word is **WAEL**



$$M_1 = \begin{bmatrix} 0 & 22 & 0 & 0 & 0 \\ 22 & 0 & -22 & 27 & 11 \\ 0 & -22 & 0 & 4 & 28 \\ 0 & 27 & 4 & 0 & 7 \\ 0 & 11 & 28 & 7 & 0 \end{bmatrix}$$

After MST graph will look like this.



So the Matrix will be modified to :

$$M_2 = \begin{bmatrix} 0 & 22 & 0 & 0 & 0 \\ 22 & 0 & -22 & 0 & 0 \\ 0 & -22 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 7 \\ 0 & 0 & 0 & 7 & 0 \end{bmatrix}$$

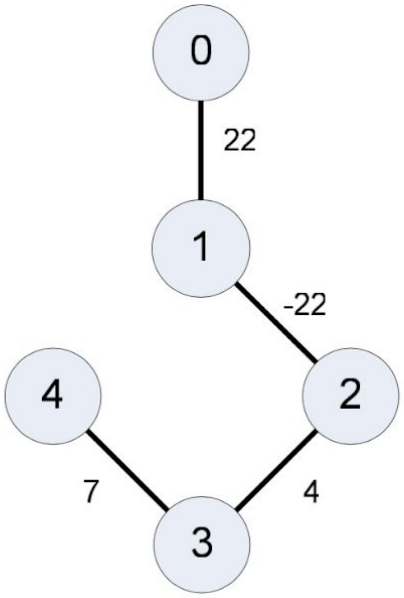
So finally step is to multiply M1*M2 to get M3 and Again Multiply M3 with KEY to get Cipher Text.

Decryption:

Then calculate M_2 by multiplying M_3 by M_1^{-1}:

$$M_2 = M_3 M_1^{-1} = \begin{bmatrix} 0 & 22 & 0 & 0 & 0 \\ 22 & 1 & -22 & 0 & 0 \\ 0 & -22 & 2 & 4 & 0 \\ 0 & 0 & 4 & 3 & 7 \\ 0 & 0 & 0 & 7 & 4 \end{bmatrix}$$

So, M_2 represents the following final graph (Fig. 8) (regardless the diagonal) that we use to retrieve the original message:



Both of these Encryption and Decryption algorithms are implemented in C++ by us.

Both the files are attached to this Project report.

Link to project: *Encryption With Graphs*

Reference: https://www.journaljsrr.com/index.php/JSRR/article/view/21430/39777

Code Screenshot:

Encryption:

```cpp
#include<bits/stdc++.h>
using namespace std;

void multiply(int *mat1, int *mat2, int *res, int N) {

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            *((res+i*N) + j) = 0;
            for (int k = 0; k < N; k++)
                *((res+i*N) + j) += *((mat1+i*N) + k) * *((mat2+k*N) + j);
        }
    }
}

struct DisjointSets {
    int *parent, *rnk;
    int n;

    DisjointSets(int n) {

        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];

        for (int i = 0; i <= n; i++)
        {
            rnk[i] = 0;
            parent[i] = i;
        }
    }
    int find(int u) {
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    void merge(int x, int y) {
        x = find(x), y = find(y);
        if (rnk[x] > rnk[y])
            parent[y] = x;
```

```
        else
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};


class Graph {
   private:
    int** adjMatrix;
    int numVertices;
    int E;
    vector< pair<int, pair<int, int> > > edges;

   public:
  Graph(int numVertices) {
    this->numVertices = numVertices;
    adjMatrix = new int*[numVertices];
    for (int i = 0; i < numVertices; i++) {
      adjMatrix[i] = new int[numVertices];
      for (int j = 0; j < numVertices; j++)
        adjMatrix[i][j] = 0;
    }
  }

  void addEdge(int i, int j, int w) {
    adjMatrix[i][j] = w;
    adjMatrix[j][i] = w;
    edges.push_back(make_pair(w, make_pair(i,j)));
  }

  void removeEdge(int i, int j) {
    adjMatrix[i][j] = 0;
    adjMatrix[j][i] = 0;
  }

  int **GetAdjMatrix() {
    return adjMatrix;
```

```
81    }
82    std::vector<pair<int, pair<int, int> > > kruskalMST() {
83      vector<pair <int, pair<int, int> > > v1;
84
85      sort(edges.begin(), edges.end());
86      DisjointSets ds(numVertices);
87
88      vector< pair<int, pair<int, int> > >::iterator it;
89      for (it=edges.begin(); it!=edges.end(); it++) {
90          int w = it->first;
91          int u = it->second.first;
92          int v = it->second.second;
93
94          int set_u = ds.find(u);
95          int set_v = ds.find(v);
96          if (set_u != set_v) {
97              //cout << u << " - " << v <<"-("<<w<<")"<< endl;
98              v1.push_back(make_pair(w, make_pair(u, v)));
99              ds.merge(set_u, set_v);
100         }
101     }
102     return v1;
103   }
104   ~Graph() {
105     for (int i = 0; i < numVertices; i++)
106       delete[] adjMatrix[i];
107     delete[] adjMatrix;
108   }
109
110 };
111
112
113 int main() {
114
115     string str = "WAEL";
116     int key[5][5] = {
117         {1,1,1,1,1},
118         {0,1,1,1,1},
119         {0,0,1,1,1},
120         {0,0,0,1,1},
```

```
        {0,0,0,0,1}
    };
    cout<<"The string you want to encrypt is "<<str<<endl;

    transform(str.begin(), str.end(), str.begin(), ::tolower);

    Graph g(str.length()+1);
    int weight = 26;

    g.addEdge(0, 1, int(str[0] - 'a'));

    g.addEdge(1, 2, int(str[1] - str[0]));
    g.addEdge(2, 3, int(str[2] - str[1]));
    g.addEdge(3, 4, int(str[3] - str[2]));
    g.addEdge(4, 1, int(str[0] - str[3]));

    g.addEdge(1, 3 , ++weight);
    g.addEdge(2, 4 , ++weight);

    int **adjMatrix = g.GetAdjMatrix();
    int M1[str.length()+1][str.length()+1];
    for (int i=0;i<str.length()+1;i++) {
        for (int j=0;j<str.length()+1;j++) {
            M1[i][j] = adjMatrix[i][j];
        }
    }

    vector<pair<int, pair<int, int> > > edgesWeight = g.kruskalMST();
    vector< pair<int, pair<int, int> > >::iterator itr;

    Graph g2(sizeof(edgesWeight));

    for (itr=edgesWeight.begin(); itr!=edgesWeight.end(); itr++) {
        int w = itr->first;
        int a = itr->second.first;
        int b = itr->second.second;

        g2.addEdge(a, b, w);
    }

```

```
    int **adjMatrix2 = g2.GetAdjMatrix();
    int M2[str.length()+1][str.length()+1];
    int M3[str.length()+1][str.length()+1];
    int C[str.length()+1][str.length()+1];

    for (int i=0;i<str.length()+1;i++) {
        for (int j=0;j<str.length()+1;j++) {
          M2[i][j] = adjMatrix2[i][j];
        }
    }
    for (int i=0;i<str.length()+1;i++) {
        M2[i][i] = i;
    }
    multiply(*M1, *M2, *M3, int(str.length()+1));
    multiply(*key, *M3, *C, int(str.length()+1));
    cout<<"\nUse Key, CipherText and M1 Matrices for Decryption "<<endl;
    cout<<"\nKey:\n";

    for (int i=0;i<str.length()+1;i++) {
        for (int j=0;j<str.length()+1;j++) {
            cout<<key[i][j]<<"\t";
        }
        cout<<endl;
    }
    cout<<"\nCipherText:\n";
    for (int i=0;i<str.length()+1;i++) {
        for (int j=0;j<str.length()+1;j++) {
            cout<<C[i][j]<<"\t";
        }
        cout<<endl;
    }
    cout<<"\nM1:\n";
    for (int i=0;i<str.length()+1;i++) {
        for (int j=0;j<str.length()+1;j++) {
            cout<<M1[i][j]<<"\t";
        }
        cout<<endl;
    }
}
```

Encryption.cpp Output:

```
$ ./encryption
    The string you want to encrypt is WAEL

    Use Key, CipherText and M1 Matrices for
 Decryption

    Key:
    1       1       1       1       1
    0       1       1       1       1
    0       0       1       1       1
    0       0       0       1       1
    0       0       0       0       1

    CipherText:
    836     302     -664    476     450
    352     280     -180    476     450
    352     -688    -244    406     217
    836     -666    -744    198     77
    242     -605    -158    133     49

    M1:
    0       22      0       0       0
    22      0       -22     27      11
    0       -22     0       4       28
    0       27      4       0       7
    0       11      28      7       0
```

Decryption:

```cpp
#include<bits/stdc++.h>
using namespace std;
#define N 5


class Graph {
    int V;
    list<int> *adj;
    void DFSUtil(int v, bool visited[]);

public:
    vector <int> vFinal;
    Graph(int V);
    void addEdge(int v, int w);
    void DFS(int v);
    std::vector<int> getFinalVector() {
        return vFinal;
    }
};

Graph::Graph(int V) {
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);
}

void Graph::DFSUtil(int v, bool visited[]) {
    visited[v] = true;
    //cout << v << " ";
    vFinal.push_back(v);
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}

void Graph::DFS(int v) {
```

```
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;
    DFSUtil(v, visited);
}

void multiply(float *mat1, float *mat2, float *res) {

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            *((res+i*N) + j) = 0;
            for (int k = 0; k < N; k++)
                *((res+i*N) + j) += *((mat1+i*N) + k) * *((mat2+k*N) + j);
        }
    }
}

void getCofactor(int A[N][N], int temp[N][N], int p, int q, int n) {
    int i = 0, j = 0;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            if (row != p && col != q) {
                temp[i][j++] = A[row][col];
                if (j == n - 1) {
                    j = 0;
                    i++;
                }
            }
        }
    }
}

int determinant(int A[N][N], int n) {
    int D = 0;
    if (n == 1)
        return A[0][0];

    int temp[N][N];
    int sign = 1;
    for (int f = 0; f < n; f++) {
```

```
        getCofactor(A, temp, 0, f, n);
        D += sign * A[0][f] * determinant(temp, n - 1);
        sign = -sign;
    }

    return D;
}

void adjoint(int A[N][N],int adj[N][N]) {
    if (N == 1) {
        adj[0][0] = 1;
        return;
    }
    int sign = 1, temp[N][N];

    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            getCofactor(A, temp, i, j, N);
            sign = ((i+j)%2==0)? 1: -1;
            adj[j][i] = (sign)*(determinant(temp, N-1));
        }
    }
}

void inverse(int A[N][N], float inverse[N][N]) {
    int det = determinant(A, N);
    int adj[N][N];
    adjoint(A, adj);
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            inverse[i][j] = adj[i][j]/float(det);
}

int main() {
    int key[5][5] = {
       {1,1,1,1,1},
       {0,1,1,1,1},
       {0,0,1,1,1},
       {0,0,0,1,1},
       {0,0,0,0,1}
```

```
    };
    int M1[5][5] = {
        {0, 22, 0, 0, 0},
        {22, 0, -22, 27, 11},
        {0, -22, 0, 4, 28},
        {0, 27, 4, 0, 7},
        {0, 11, 28, 7, 0}
    };
    float C[5][5] = {
        {836, 302, -664, 476, 450},
        {352, 280, -180, 476, 450},
        {352, -688, -244, 406, 217},
        {836, -666, -744, 198, 77},
        {242, -605, -158, 133, 49}
    };

    float M1Inverse[5][5];
    float keyInverse[5][5];
    float M2[5][5];
    float M3[5][5];

    inverse(key, keyInverse);
    inverse(M1, M1Inverse);
    multiply(*keyInverse,*C,  *M3);

    multiply(*M1Inverse, *M3, *M2);

    Graph g(5);
    for (int i=0;i<5;i++) {
        for (int j=0;j<5;j++) {
           if (!(round(M1[i][j]) == 0  && round(M1[i][j] == -0))) {
               g.addEdge(i, j);
           }
        }
    }
    g.DFS(0);
    vector<int> v = g.getFinalVector();
    int prev = 0;
    char ltr = 'a';
    char ltrnow;
```

```
161     cout<<"The Encrypted word is :"<<endl;
162     for (auto itr = v.begin()+1; itr != v.end(); itr++) {
163
164         ltrnow = char(int(ltr) + M1[prev][*itr]);
165         cout<<ltrnow;
166         ltr = ltrnow;
167         prev = *itr;
168     }
169     cout<<endl;
170
171 }
```

Decryption.cpp Output:

```
$ ./decryption
    The Encrypted word is
:   wael
```

Thank you