5.1 Introduction:

Facility location problems deal with the task of choosing a site subject to some criterion. For example, in determining where to locate an emergency facility such as a hospital or fire station, we would like to minimize the response time between the facility and the location of a possible emergency. In deciding the position for a service facility such as post office, power station, or employment office, we want to minimize the total travel time for all people in the district. When constructing a railroad line, pipe line, or superhighway, we want to minimize the distance from the new structure to each of the communities to be served. Each of these situations deals with the concept of centrality. However, the type of center differs for each of the three examples mentioned. Centrality questions are now examined using graphs and distance concepts are useful in facility location problems.

Let \( u \) and \( v \) be two vertices of a graph \( G \). Suppose that there exists a path from \( u \) to \( v \) in \( G \). The distance from \( u \) to \( v \), written \( d_G(u, v) \) or simply \( d(u, v) \), is the least length among all paths from \( u \) to \( v \). Suppose that there exists no path from \( u \) to \( v \) in \( G \) then \( d(u, v) = \infty \).

A graph is a collection of points and lines connecting some of them. The points of a graph are most commonly known as graph vertices. Similarly the lines connecting the vertices of a graph are most commonly known as graph edges. Formally we can define a graph as a graph \( G \) is a pair of sets \( V \) and \( E \) together with a function \( F: E \rightarrow V \times V \). The elements of \( V \) are vertices and the elements of \( E \) are edges. Connections between the points come in two forms those that are non-directional and those that have an implicit direction are undirected and directed respectively.
In a graph theory a tree is connected acyclic graph stated otherwise trees and graph are undirected. A tree is called a rooted tree if one vertex has been designated the root in which case the edges have a natural orientation towards or away from the root.

5.1.1 Eccentricity:

The concept of eccentricity is fundamental in graph theory. In this chapter we are designing an algorithm for finding eccentricity of a tree. Tree is connected graph with no cycles. In an undirected tree a leaf is a vertex of degree 1. Some basic properties of trees are:

- Every tree with at least one edge has at least two leaves. If the minimum degree of a graph is at least 2, then that graph must contain a cycle.
- Every tree on n vertices has exactly n – 1 edges.

Eccentricity of a tree: the eccentricity of a vertex \( v \) in a graph \( G \). Denoted \( \text{ecc}(v) \), is the distance from \( v \) to a vertex farthest from \( v \), that is

\[
\text{ecc}(v) = \max_{x \in V_G} \{d(v,x)\}.
\]

A central vertex of a graph is a vertex with minimum eccentricity.

We begin with some existing [3] preliminary results concerning the eccentricity of vertices in a tree.

5.1.1.1 *Lemma 1*: Let \( T \) be a tree with at least three vertices

- a) If \( v \) is a leaf of \( T \) and \( w \) is its neighbor, then
  \[
  \text{ecc}(v) = \text{ecc}(w) + 1.
  \]
- b) If \( v \) is a central vertex of \( T \), then
  \[
  \text{deg}(v) \geq 2.
  \]

5.1.1.2 *Lemma 2*: Let \( v \) and \( w \) be two vertices in a tree \( T \) such that \( w \) is of maximum, distance from \( v \) (i.e \( \text{ecc}(v) = d(v, w) \)). then \( w \) is a leaf.

5.1.1.3 *Lemma 3*: Let \( T \) be a tree with at least three vertices, and let \( T^* \) be the subtree of \( T \) obtained by deleting from \( T \) all its leaves. If \( v \) is a vertex of \( T^* \), then

\[
\text{ecc}_T(v) = \text{ecc}(T^*(V)) + 1
\]
5.1.2 Diameter:

Determining the diameter of a graph is a fundamental seemingly quite time consuming operation but we are restricted it to tree. Recall that the eccentricity of vertex \( x \). \( \text{ecc} (x) = \max_{y \in V} d(x, y) \), where \( d(x, y) \) denotes the distance between \( x \) and \( y \); the diameter of \( G \) equals the maximum eccentricity of any vertex in \( V \).

Let \( G \) be a graph and \( v \) be a vertex of \( G \). The diameter of \( G \) is the maximum eccentricity among the vertices of \( G \).

Thus, diameter \( (G) = \max \{\text{ecc}(v) : V \in V(G)\} \).

Let us now consider the existing properties [1] to help us to find the diameter of tree.

5.1.2.1 Fact: Suppose that \( SPT(V_1, V_2) \) is a diameter of \( T \) and \( r \) is a vertex on the diameter. For any vertex \( x \),

\[
d_T(x, r) \leq \max \{d_T(r, V_1), d_T(r, V_2)\}.
\]

5.1.2.2 Lemma: Let \( r \) be any vertex in a tree \( T \). If \( v \) is the farthest vertex to \( r \),

the eccentricity of \( v \) is the diameter of \( T \).

5.1.3 Radius:

The minimum eccentricity of all points in a graph is called the radius \( r(G) \) of the graph. The radius can be obtained from a diameter.

The radius of \( G \) is the minimum eccentricity among the vertices of \( G \).

Therefore radius \( (G) = \min \{\text{ecc}(v) : V \in V(G)\} \).

Suppose that \( p = SPT(V_1, V_2) \) is a diameter. Starting at \( V_1 \) and travelling along the path \( p \). We compute the distance \( d_T(u, v) \) for each vertex \( u \) on the path. Let \( u_1 \) be the last encountered vertex. Such that \( d_T(u_1, v) \leq \frac{1}{2} w(P) \) and \( u_2 \) be the next vertex to \( u_1 \) as shown in the below figure 5.1
5.2 Algorithm: Eccentricity, Diameter and Radius of a tree

**Input:** $n$ — number of nodes

Cost $[50][50]$ - adjacency matrix

**Output:** Eccentricity, Diameter and Radius

5.2.1 Input the number of nodes and adjacency matrix

5.2.2 For $i \leftarrow 0$ to $n - 1$

   For $j \leftarrow 0$ to $n - 1$

   $D[i][j] = cost[i][j]$

   End $j$

   End $i$

5.2.3 To find the shortest distance

   For $k \leftarrow 0$ to $n - 1$

   For $i \leftarrow 0$ to $n - 1$

   For $j \leftarrow 0$ to $n - 1$

   $D[i][j] = min1(D[i][j], D[j][k], D[k][i])$

   End $j$

   End $i$

   End $k$

---

Figure 5.1
Algorithm: Eccentricity

5.2.4 To find eccentricity

for $i \leftarrow 0$ to $n - 1$

Initialize $\text{max} = D[i][0]$

for $j \leftarrow 0$ to $n - 1$

if $D[i][j] > \text{max}$

$\text{max} = D[i][j]$

End j

$e[i] = \text{max}$

End i

Let $T$ be tree

Consider the above tree, and then the eccentricity of each vertex in the tree is given below.

$E(1) = 3$
$E(2) = 3$
$E(3) = 3$
$E(4) = 2$
$E(5) = 2$
$E(6) = 3$
Algorithm: Diameter

5.2.5 To find Diameter

Initialize min and \( max = e[0] \)
for \( i \leftarrow 0 \) to \( n - 1 \)
if \( e[i] > max \)

\[ max = e[i] \]

else if \( e[i] < min \)

\[ min = e[i] \]

End \( i \)

Diameter = \( max \).

\[ E(1) = 3 \]
\[ E(2) = 3 \]
\[ E(3) = 3 \]
\[ E(6) = 3 \]

Diameter is 3.
Algorithm: Radius

5.2.6 To find the radius

Initialize $min$ and $max = e[0]$

for $i ← 0$ to $n - 1$

if $e[i] > max$

$\quad max = e[i]$

else if $e[i] < min$

$\quad min = e[i]$

end i

Radius = $min$

Figure: 5.4

$E(4) = 2$

$E(5) = 2,$

Radius of the graph is 2.
5.2.7 Flow chart of algorithm to find the eccentricity diameter and radii of graphs using DFS.

Figure: 5.5.a
Figure: 5.5.b
Create (n)

print f("enter matrix")

for i = 0 to n

  ub[i] = (table*)malloc(size of (table))

  for j = 0 to n

    Enter item

    if item == 1

      W = 0

      Enter item

    if item

      T

      Create new node

    F

    if e[i] = Radius

      T

      Centre [i]

      Display centre vertex

      Stop

Figure: 5.5.c
```plaintext
dfs()

Initialize cur = tab(u)->nodeptr

while

  if tab[cur->info]->visit == false

    dfs(cur->info)

    e[i]=k cur = cur->link

    max = e[i]

  cur

Stop
```

Figure: 5.5.d
5.2.8 We proposed another program which finds the eccentricity diameter and radii of graphs using DFS. We implemented it through linked list

```c
#include<stdio.h>
#define FALSE 0
#define TRUE 1
#define SIZE 15
typedef struct node
{
    int inf;
    int weight;
    struct node *link;
} node;
typedef struct table
{
    int visit;
    char data;
    node *nodeptr;
} table;
table *tab[SIZE]
int max=0,n,e[50],I, j;
Void dfs (int);
Void create (int)
Void main()
{
    int start, radius, center[20],diameter,min;
    node *cur;
c1rscrt();
    printf("Enter the no of nodes:");
    scanf("%d",&n);
    Create (n);
    for (i=0; i<n; i++)
    {
        e[i]=0;
    }
    for (i=0; i<n; i++)
    {
        e[i]=0;
    }
```
max = 0;
for (j=0;j<n;j++)
tab [i]-> visit = FALSE;
tab [i]-> visit = TRUE;
dfs(i);
e [i]=max;
for (j=0;j<n;j++)
tab [j]->visit=FALSE;
}
for (i=0;i<n;i++)
Printf ("n Eccentricity of %d is %d\n",i,e[i]);
Min=e[0];
Max=e [0];
for(i=1;i<n;i++)
{
if (e[i].max)
max =e[i];
if (e[i]<min)
min =e[i];
}
radius =min;
diameter =max;
printf ("Radius = %d\n", radius);
printf ("Diameter = %d\n", diameter);
}

Void create (int n)
{
Node *new1,*temp;
printf ("Enter the elements of the matrix below: \n");
for (i=0; i<n; i++)
{
    tab [i] =(table*)malloc(size of (table));
}
for (j=0;j<n;j++)
{
    printf("is there is edge from %d to %d\n",i,j);
    scanf("%d",&item)
    if (item== 1)
    {
        printf("Enter the Weight\n");
        scanf("%d",&w);
    }
    else
    
    w = 0;
    if(item)
    {
        newl = (node*) malloc (size of (node));
        newl -> info =j;
        newl -> weight = w;
        newl -> link = NULL;
        if (temp)
            temp -> link =new;
        else
            tab[i] -> nodeptr = newl;
        temp = new;
    }
}
}
for (i=0; i<n;i++)
{
    if (e[i] == radius)
{ 
    Center[i];
    printf("Center Vertex is %d", center[i]);
}

void dfs(int u) 
{
    node *cur;
    int k;
    k = e[i]
    cur = tab[u] -> nodeptr;
    while(cur)
    {
        if (tab[cur -> info] -> visit = = FALSE)
        {
            E[i] +=cur -> weight;
            tab[cur -> info] -> visit = TRUE;
            dfs(cur -> info);
            if (max < e[i];
            {
                (max = e[i]);
            }
        } 
        e[i] = k;
        
        cur = cur -> link;
    }
}  
}
5.3 Conclusion:

Tree is connected graph with no cycles. In an undirected tree a leaf is a vertex of degree 1. In this chapter we are designing an algorithm for finding eccentricity of a tree, an algorithm for finding diameter of a tree and an algorithm for finding a radius of a tree.
Reference: