Shubhan Singh

SE-Comps B/Batch C

2022300118

DAA Experiment 6

Greedy Approach - I

Aim – To implement Dijkstra's and Prim's algorithms

Part 1 : Prim's Algorithm

Pseudocode :

Algorithm Prim(G, w, r)

Input: A connected, undirected graph G = (V, E), a weight function w: E -> R, and a root vertex r in V

Output: A minimum spanning tree T for G

```
1. for each u in V
2.     key[u] = infinity
3.     parent[u] = NIL
4. key[r] = 0
5. Q = V[G]    // Initialize priority queue Q to contain all vertices in V
6. while Q is not empty
7.     u = Extract-Min(Q)    // Pull out new vertex u for which key[u] is minimum
8.     for each vertex v in Adj[u]    // Process the edge (u, v)
9.         if v is in Q and w(u, v) < key[v]
10.            parent[v] = u
```

11. key[v] = w(u, v)

12. return parent

Source code(C language):

```c
#include<stdio.h>
#include<stdlib.h>

#define inf 0x7fffffff
int extractmin(int* key,int n){
    int min=inf,minindex=0;
    for(int i=0;i<n;i++){
        if(key[i]<min){
            min=key[i];
            minindex=i;
        }
    }
    key[minindex]=inf;
    return minindex;
}

int* mst_prim(int** graph, char root,int n){
    int *parents=malloc(n*sizeof(int));
    int key[n];
    int visited[n];
    for(int i=0;i<n;i++){
        visited[i]=0;
        parents[i]=-1;
        key[i]=inf;
    }
    key[root-'a']=0;
    int u;
    for(int i=0;i<n;i++){
        u=extractmin(key,n);
        visited[u]=1;
        for(int j=0;j<n;j++){
            if(visited[j]==0 && graph[u][j]!=0){
                if(graph[u][j]<key[j]){
                    key[j]=graph[u][j];
                    parents[j]=u;
                }
            }
```

```
        }
    }

    return parents;
}

int main(){
    int nV,nE,w;
    char v1,v2;
    printf("Enter number of vertices: ");
    scanf("%d", &nV);
    printf("Enter number of edges: ");
    scanf("%d", &nE);
    int **graph = (int **)malloc(nV * sizeof(int *));
    for(int i=0;i<nV;i++){
        graph[i] = (int *)calloc(nV,sizeof(int));
    }

    while((getchar()) != '\n');
    printf("Enter the edges as vortex1 vortex2, where names
of vertices start from 'a': \n");
    for(int i=0;i<nE;i++){
        scanf("%c %c", &v1, &v2);
        int v1i=v1-'a';
        int v2i=v2-'a';
        printf("Enter weight of edge %c <-> %c: ", v1, v2);
        while((getchar()) != '\n');
        scanf("%d", &w);
        graph[v1i][v2i] = w;
        graph[v2i][v1i] = w;
        while((getchar()) != '\n');
    }
    printf("\nEnter root vortex: ");
    scanf("%c",&v1);
    printf("\nThe edges in the Minimum spanning tree as found
by Prim's algorithm are:\n");
    int *parents=mst_prim(graph,v1,nV);
    for(int i=1;i<nV;i++){
        printf("%c<->%c  ",('a'+i),('a'+parents[i]));
    }
    int cost=0;
    for(int i=1;i<nV;i++){
```

```
        cost+=graph[i][parents[i]];
    }
    printf("\n\nThe cost of the minimum spanning tree
is: %d\n",cost);

    for(int i=0;i<nV;i++){
        free(graph[i]);
    }
    free(graph);
    free(parents);
return 0;
}
```

Output:

```
C:\Users\shubh\OneDrive - Bharatiya Vidya Bhavans Sardar Patel Institute Of Technology\DAA>.\prims
Enter number of vertices: 9
Enter number of edges: 14
Enter the edges as vortex1 vortex2, where names of vertices start from 'a':
a b
Enter weight of edge a ↔ b: 4
a h
Enter weight of edge a ↔ h: 8
b h
Enter weight of edge b ↔ h: 11
b c
Enter weight of edge b ↔ c: 8
c i
Enter weight of edge c ↔ i: 2
c d
Enter weight of edge c ↔ d: 7
c f
Enter weight of edge c ↔ f: 4
d e
Enter weight of edge d ↔ e: 9
d f
Enter weight of edge d ↔ f: 14
e f
Enter weight of edge e ↔ f: 10
f g
Enter weight of edge f ↔ g: 2
g i
Enter weight of edge g ↔ i: 6
g h
Enter weight of edge g ↔ h: 1
h i
Enter weight of edge h ↔ i: 7

Enter root vortex: a
```

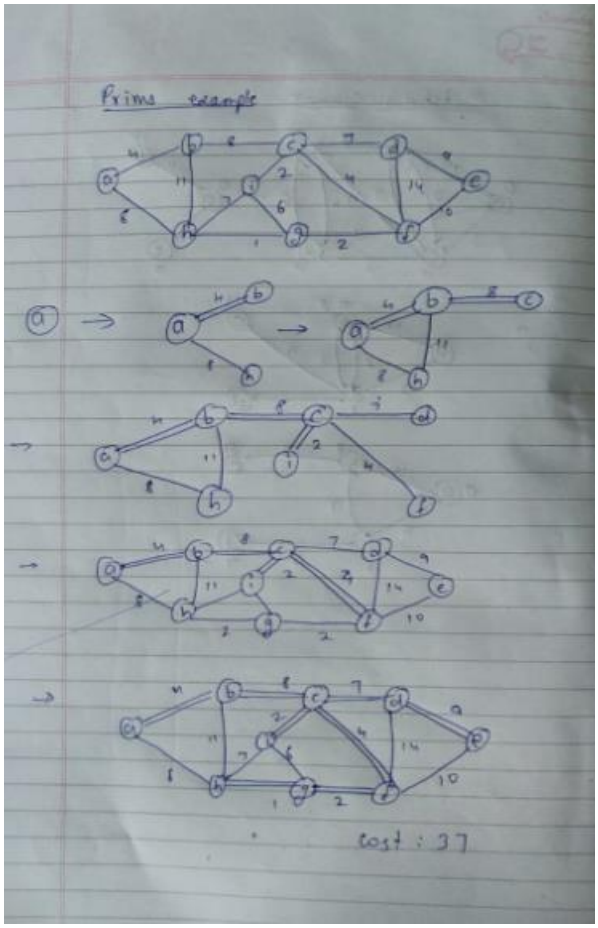
```
Enter root vortex: a

The edges in the Minimum spanning tree as found by Prim's algorithm are:
b↔a  c↔b  d↔c  e↔d  f↔c  g↔f  h↔g  i↔c

The cost of the minimum spanning tree is: 37

C:\Users\shubh\OneDrive - Bharatiya Vidya Bhavans Sardar Patel Institute Of Technology\DAA>
```

Rough Working :



Part 2 : Dijkstra's Algorithm

Pseudocode:

Algorithm Dijkstra(G, w, r)

Input: A connected, undirected graph G = (V, E), a weight function w: E -> R, and a source vertex r in V

Output: The shortest path from the source vertex r to all other vertices in V

1. Initialize-Single-Source(G, r)
2. S = empty set
3. Q = V[G] // Initialize priority queue Q to contain all vertices in V
4. while Q is not empty

5. u = Extract-Min(Q) // Pull out new vertex u for which d[u] is minimum
6. Add u to S
7. for each vertex v in Adj[u] // Process the edge (u, v)
8. Relax(u, v, w)

Function Initialize-Single-Source(G, r)

1. for each vertex v in V[G]
2. d[v] = infinity
3. parent[v] = NIL
4. d[r] = 0

Function Relax(u, v, w)

1. if d[v] > d[u] + w(u, v)
2. d[v] = d[u] + w(u, v)
3. parent[v] = u

Source code (C language) :

```c
#include<stdio.h>
#include<stdlib.h>

#define inf 0x7fffffff

int extractmin(int** distances,int n,int* visited){
    int min=inf,minindex=0;
    for(int i=0;i<n;i++){
        if(distances[0][i]<min && visited[i]==0){
            min=distances[0][i];
            minindex=i;
        }
    }
    visited[minindex]=1;
```

```
    return minindex;
}

void relax(int** graph,int u, int v, int** distances){
    if(graph[u][v]<distances[0][v]){
        distances[0][v]=graph[u][v]+distances[0][u];
        distances[1][v]=u;
    }
}

void printpath(int ** distances, int source){
    if(source==-1){
        return;
    }
    printpath(distances,distances[1][source]);
    printf("%c -> ",(source+'a'));
    return;
}

int** dijkstra(int **graph, int source,int n){
    int** distances=malloc(2*sizeof(int*));
    for(int i=0;i<2;i++){
        distances[i]=malloc(n*sizeof(int));
    }
    int visited[n];
    for(int i=0;i<n;i++){
        distances[0][i]=inf;
        visited[i]=0;
    }
    distances[0][source]=0;
    distances[1][source]=-1;
    int u;
    for(int i=0;i<n;i++){
        u=extractmin(distances,n,visited);
        for(int j=0;j<n;j++){
            if(visited[j]==0 && graph[u][j]!=0){
                relax(graph,u,j,distances);
            }
        }
    }

    return distances;
```

```
}

int main(){
    int nV,nE,w;
    char v1,v2;
    printf("Enter number of vertices: ");
    scanf("%d", &nV);
    printf("Enter number of edges: ");
    scanf("%d", &nE);
    int **graph = (int **)malloc(nV * sizeof(int *));
    for(int i=0;i<nV;i++){
        graph[i] = (int *)calloc(nV,sizeof(int));
    }

    while((getchar()) != '\n');
    printf("Enter the edges as vortex1 vortex2, where names
of vertices start from 'a': \n");
    for(int i=0;i<nE;i++){
        scanf("%c %c", &v1, &v2);
        int v1i=v1-'a';
        int v2i=v2-'a';
        printf("Enter weight of edge %c -> %c: ", v1, v2);
        while((getchar()) != '\n');
        scanf("%d", &w);
        graph[v1i][v2i] = w;
        while((getchar()) != '\n');
    }
    printf("\nEnter source vortex: ");
    scanf("%c",&v1);
    int** distances=dijkstra(graph,v1-'a',nV);
    printf("\nThe shortest paths to the other vertices from
the source as per Dijkstra's algorithm is:\n");
    for(int i=0;i<nV;i++){
        if(i!=v1-'a'){
            printpath(distances,distances[1][i]);
            printf("%c\tcost: %d\n",(i+'a'),distances[0][i]);
        }
        else{
            printf("%c\tcost: 0\n",i+'a');
        }
    }
```

```
    for(int i=0;i<nV;i++){
        free(graph[i]);
    }
    free(graph);
    free(distances);

return 0;
}
```

Output:

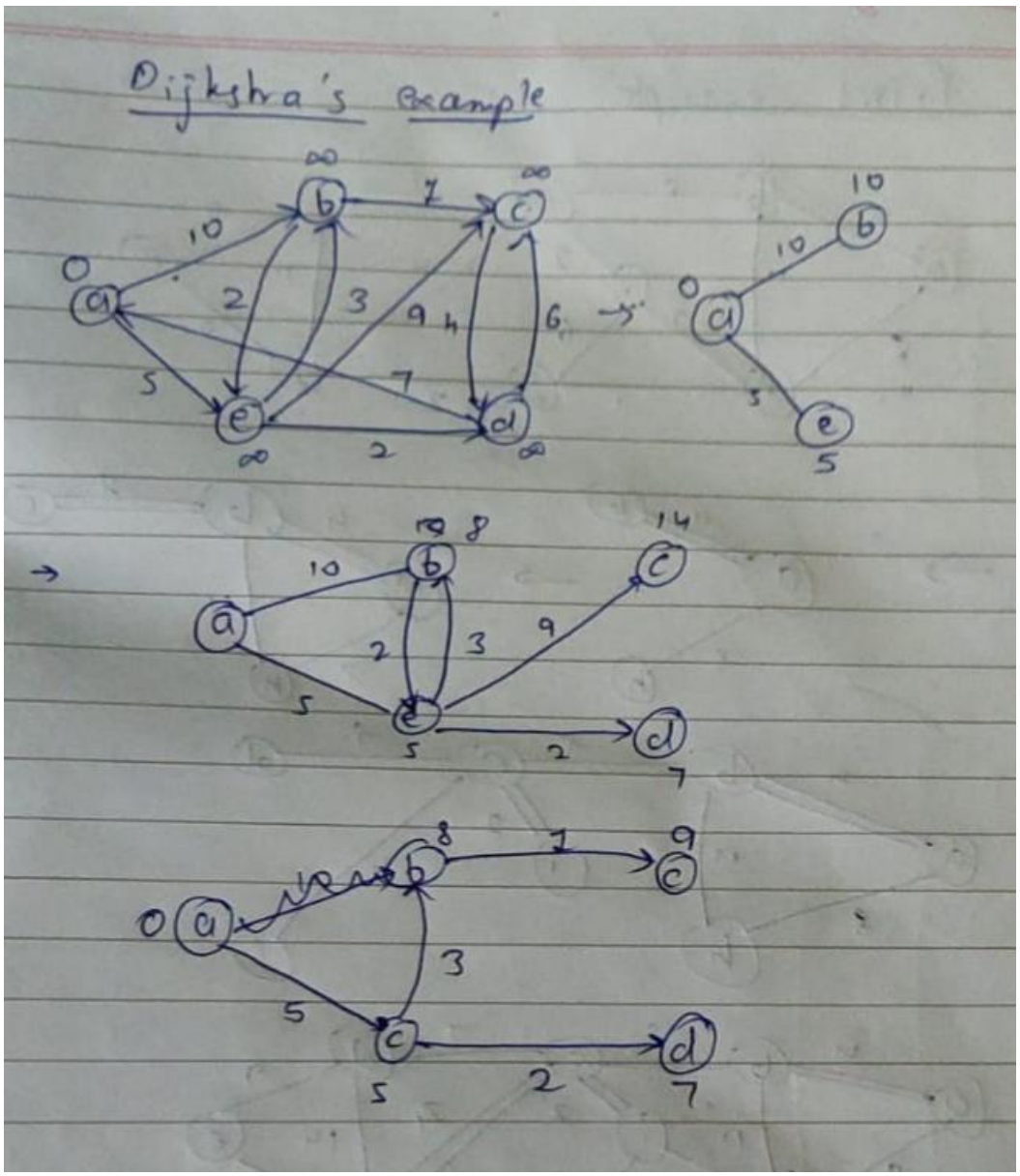
```
C:\Users\shubh\OneDrive - Bharatiya Vidya Bhavans Sardar Patel Institute Of Technology\DAA>.\dijkstra
Enter number of vertices: 5
Enter number of edges: 10
Enter the edges as vortex1 vortex2, where names of vertices start from 'a':
a b
Enter weight of edge a → b: 10
a e
Enter weight of edge a → e: 5
b c
Enter weight of edge b → c: 1
b e
Enter weight of edge b → e: 2
e b
Enter weight of edge e → b: 3
e d
Enter weight of edge e → d: 2
e c
Enter weight of edge e → c: 9
d a
Enter weight of edge d → a: 7
c d
Enter weight of edge c → d: 4
d c
Enter weight of edge d → c: 6

Enter source vortex: a

The shortest paths to the other vertices from the source as per Dijkstra's algorithm is:
a       cost: 0
a → e → b     cost: 8
a → e → b → c        cost: 9
a → e → d     cost: 7
a → e  cost: 5

C:\Users\shubh\OneDrive - Bharatiya Vidya Bhavans Sardar Patel Institute Of Technology\DAA>
```

Rough working:



Theory:

Prim's Algorithm:

Prim's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, undirected graph. The minimum spanning tree of a graph is a subgraph that includes all the vertices of the graph with the minimum possible total edge weight. Prim's algorithm starts with an arbitrary vertex and grows the minimum spanning tree by adding the shortest edge that connects a vertex in the tree to a vertex outside the tree, until all vertices are included.

Time Complexity:

The time complexity of Prim's algorithm depends on the data structure used for implementing the priority queue. With a simple array-based implementation, the time complexity is O(V^2), where V is the number of vertices. With more efficient implementations using Fibonacci heaps or binary heaps, the time complexity can be reduced to O(E + V log V), where E is the number of edges.

Dijkstra's Algorithm:

Dijkstra's algorithm is a greedy algorithm used to find the shortest path between a single source vertex and all other vertices in a weighted graph with non-negative edge weights. It maintains a set of vertices whose shortest distance from the source vertex is already known and continuously expands this set by selecting the vertex with the minimum distance from the source and updating the distances to its neighboring vertices.

Time Complexity:

The time complexity of Dijkstra's algorithm depends on the data structure used for implementing the priority queue. With a simple array-based implementation, the time complexity is O(V^2), where V is the number of vertices. With more efficient implementations using Fibonacci heaps or binary heaps, the time complexity can be reduced to O((E + V) log V), where E is the number of edges.

Both Prim's and Dijkstra's algorithms are widely used in various applications such as network routing, telecommunications, and computer networking for finding optimal paths and minimum spanning trees.