Ex.No-1	TOPOLOGICAL ORDERING	Date: 6-3-24

Aim:

To obtain the topological ordering of Vertices in a Given Digraph.

Description:

- Topological sort is an ordering of the vertices in a directed acyclic graph, such that, if there is a path from u to v, then v appears after u in the ordering.
- **The graphs should be directed:** otherwise for any edge (u, v) there would be a path from u to v and also from v to u, and hence they cannot be ordered.
- **The graphs should be acyclic:** otherwise for any two vertices u and v on a cycle u would precede v and v would precede u.

Using Source Removal algorithm:

1. Compute the in degrees of all vertices
2. Find a vertex U with in degree 0 and print it (store it in the ordering)
3. If there is no such vertex then there is a cycle and the vertices cannot be ordered. Stop.
4. Remove U and all its edges (U, V) from the graph.
5. Update the indegrees of the remaining vertices.
6. Repeat steps 2 through 4 while there are vertices to be processed.

Source Code:

```
#include<stdio.h>
int temp[10],k=0;
voidtopo(intn,intindegree[10],int a[10][10])
{
inti,j;
for(i=1;i<=n;i++)
{
if(indegree[i]==0)
{
indegree[i]=1;
temp[++k]=i;
for(j=1;j<=n;j++)
{
if(a[i][j]==1&&indegree[j]!=-1)
indegree[j]--;
}
i=0;
}
}
}
void main()
{
```

```
inti,j,n,indegree[10],a[10][10];
printf("enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
indegree[i]=0;
printf("\n enter the adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
if(a[i][j]==1)
indegree[j]++;
}
topo(n,indegree,a);
if(k!=n)
printf("topological ordering is not possible\n");
else
{
printf("\n topological ordering is :\n");
for(i=1;i<=k;i++)
printf("v%d\t",temp[i]);
}
}
```

Using DFS algorithm:

1. Run DFS(G), computing finish time for each vertex
2. As each vertex is finished, insert it onto the front of a list

Source Code:

```
#include<stdio.h>
inti,visit[20],n,adj[20][20],s,topo_order[10];
voiddfs(int v)
{
int w;
visit[v]=1;
for(w=1;w<=n;w++)
if((adj[v][w]==1) && (visit[w]==0))
dfs(w);
topo_order[i--]=v;
}
voidmain ()
{
intv,w;
printf("Enter the number of vertices:\n");
```

```
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(v=1;v<=n;v++)
for(w=1;w<=n;w++)
scanf("%d",&adj[v][w]);
for(v=1;v<=n;v++)
visit[v]=0;
i=n;
for(v=1;v<=n;v++)
{
if(visit[v]==0)
dfs(v);
}
printf("\nTopological sorting is:");
for(v=1;v<=n;v++)
printf("v%d ",topo_order[v]);
}
```

OUTPUT:

Enter the no. of vertices : 4
Enter the adjacency matrix :

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Topological string : V_4 V_3 V_2 V_1

RESULT: The above Topological Ordering program is executed succesfully.

Ex.No-2	QUICK SORT	Date: 13-3-2024

Aim:

To Sort a given set of elements using the Quick sort method and determine the time required to sort the elements.

Description:

1. Quicksort is a very efficient sorting algorithm invented by C.A.R. Hoare. It has two phases:
2. The partition phase and the sort phase.
3. As we will see, most of the work is done in the partition phase. The sort phase simply sorts the two smaller problems that are generated in the partition phase.
4. This makes Quicksort a good example of the divide and conquers strategy for solving problems. In quicksort, we divide the array of items to be sorted into two partitions and then call the quicksort procedure recursively to sort the two partitions, ie we divide the problem into two smaller ones and conquer by solving the smaller ones.

Pseudo code For partition(a, left, right, pivotIndex):

```
pivotValue := a[pivotIndex]
swap(a[pivotIndex], a[right]) // Move pivot to end
storeIndex := left
for i from left to right-1
if a[i] ≤ pivotValue
swap(a[storeIndex], a[i])
storeIndex := storeIndex + 1
swap(a[right], a[storeIndex]) // Move pivot to its final place
returnstoreIndex
```

Pseudo code For quicksort(a, left, right):

```
if right > left
select a pivot value a[pivotIndex]
pivotNewIndex := partition(a, left, right, pivotIndex)
quicksort(a, left, pivotNewIndex-1)
quicksort (a, pivotNewIndex+1, right)
```

ANALYSIS:

The partition routine examines every item in the array at most once, so complexity is clearly O (n).Usually, the partition routine will divide the problem into two roughly equal sized partitions. We know that we can divide n items in half log2n times.

Source code:

```
#include<stdio.h>
#include<conio.h>
void quicksort(int[],int, int);
intpartition (int[],int,int);
```

```
void main()
{
inti,n,a[20],ch=1;
clrscr();
while(ch)
{
printf("\n enter the number of elements\n");
scanf("%d",&n);
printf("\n enter the array elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
quicksort(a,0,n-1);
printf("\n\nthe sorted array elements are\n\n");
for(i=0;i<n;i++)
printf("\n%d",a[i]);
printf("\n\n do u wish to continue (0/1)\n");
scanf("%d",&ch);
}
getch();
}
void quicksort(int a[],intlow,int high)
{
int mid;
if(low<high)
{
mid=partition(a,low,high);
quicksort(a,low,mid-1);
quicksort(a,mid+1,high);
}
}
int partition(int a[],intlow,int high)
{
intkey,i,j,temp,k;
key=a[low];
i=low+1;
j=high;
while(i<=j)
{
while(i<=high && key>=a[i])
i=i+1;
while(key<a[j])
j=j-1;
if(i<j)
{
temp=a[i];
a[i]=a[j];
```

f"Time Taken = %.1f")

```
a[j]=temp;
}
else
{
k=a[j];
a[j]=a[low];
a[low]=k;
}
}
return j;
}
```

OUTPUT:

```
Enter the number of elements : 8
Enter the array elements :
5
3
6
7
8
6
4
2
the sorted array elements are :
   -54 -52 -50 -48 -46 -44 -42 -40
do you wish to Continue (0/1)
0
```

RESULT: The above given set of elements are sorted using quicksort method and determination of the sort elements have been executed and verified successfully.

Ex.No-3 20-3-24	MERGE SORT	Date: 20-3-24

Aim:

To Sort a given set of elements using the Merge sort method and determine the time required to sort the elements.

Objective:

Sort a given set of elements using Merge sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

Algorithm:

MergeSort (arr [], l, r)
If r > l
1. Find the middle point to divide the array into two halves:
   Middle m = (l+r)/2
2. Call merges Sort for first half:
   Call merges Sort (arr, l, m)
3. Call merge Sort for second half:
   Call merge Sort (arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
   Calls merge (arr, l, m, r)

Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#define max 20
voidmergesort(int a[],intlow,int high);
void merge(int a[],intlow,intmid,int high);
void main()
{
intn,i,a[max],ch=1;
clock_tstart,end;
clrscr();
while(ch)
{
printf("\n\t enter the number of elements\n");
scanf("%d",&n);
printf("\n\t enter the elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
start= clock();
mergesort(a,0,n-1);
```

clock_t start, end

```
end=clock();
printf("\nthe sorted array is\n");
for(i=0;i<n;i++)
printf("%d\n",a[i]);
printf("\n\ntime taken=%lf",(end-start)/CLK_TCK);
printf("\n\ndo u wish to continue(0/1) \n");
scanf("%d",&ch);
}
getch();
} voidmergesort(int a[],intlow,int high) {
int mid;
delay(100);
if(low<high) {
mid=(low+high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
merge(a,low,mid,high);
}
}
void merge(int a[],intlow,intmid,int high) {
inti,j,k,t[max];
i=low;
j=mid+1;
k=low;
while((i<=mid) && (j<=high))
if(a[i]<=a[j])
t[k++]=a[i++];
else
t[k++]=a[j++];
while(i<=mid)
t[k++]=a[i++];
while(j<=high)
t[k++]=a[j++];
for(i=low;i<=high;i++)
a[i]=t[i];
}
```

OUTPUT:

Enter the number of elements : 11
Enter the elements : 9 2 4 6 7 10 8 1 0 11 14
The sorted array
0 6 11
1 7 14
2 8
3 9
4 10
5
Time taken : 2881F

RESULT: Hence the set of elements using the m
sort method and determination of time required
to sort the elements are executed successfully.

Ex.No-4	DEPTH FIRST SEARCH (DFS) ALGORITHM	Date: 27/3/24

Aim:

To check whether a given graph is connected or not using DFS method.

Description:

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

Algorithm:

DFS(G,v) (v is the vertex where the search starts)
Stack S := {}; (start with an empty stack)
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
u := pop S;
if (not visited[u]) then
visited[u] := true;
for each unvisited neighbour w of u
push S, w;
end if
end while
END DFS()

Source code:

```
#include<stdio.h>
int visit[20],n,adj[20][20],s,count=0;
voiddfs(int v)
{
int w;
visit[v]=1;
count++;
for(w=1;w<=n;w++)
if((adj[v][w]==1) && (visit[w]==0))
dfs(w);
}
void main()
{
intv,w;
printf("Enter the no.of vertices:");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(v=1;v<=n;v++)
```

```
for(w=1;w<=n;w++)
scanf("%d",&adj[v][w]);
for(v=1;v<=n;v++)
visit[v]=0;
dfs(1);
if(count==n)
printf("\nThe graph is connected");
else
printf("The graph is not connected");
}
```

OUTPUT:

Enter the no. of vertices : 3
Enter the adjacency matrix :
' ' '
' ' '
' ' '
The graph is Connected.

Enter the no. of vertices : 5
Enter the adjacency matrix :

5	7	9	10	3
12	4	0	5	6
20	5	9	8	2
5	9	0	3	0
2	4	7	9	5

The graph is not Connected.

RESULT: Hence, By using DFS I can check whether a given is Connected or not program executed Successfully.

Ex.No-5	BREADTH FIRST SEARCH (BFS) ALGORITHM	Date: 3-4-24

Aim:

To print all the nodes reachable from a given starting node in a directed graph using BFS method.

Description:

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors.

Algorithm:

1. Start from node s.
2. Visit all neighbors of node s.
3. Then visit all of their neighbors, if not already visited
4. Continue until all nodes visited

Source code:

```
#include<stdio.h>
#define size 20
#define true 1
#define false 0
int queue[size],visit[20],rear=-1,front=0;
intn,s,adj[20][20],flag=0;
voidinsertq(int v)
{
queue[++rear]=v;
}
intdeleteq()
{
return(queue[front++]);
}
intqempty()
{
if(rear<front)
return 1;
else
return 0;
}
voidbfs(int v)
{
int w;
visit[v]=1;
insertq(v);
```

```
while(!qempty())
{
v=deleteq();
for(w=1;w<=n;w++)

if((adj[v][w]==1) && (visit[w]==0))
{
visit[w]=1;
flag=1;
printf("v%d\t",w);
insertq(w);
}
}
}
void main()
{
int v,w;
printf("Enter the no.of vertices:\n");
scanf("%d",&n);
printf("Enter adjacency matrix:");
for(v=1;v<=n;v++)
{
for(w=1;w<=n;w++)
scanf("%d",&adj[v][w]);
}
printf("Enter the start vertex:");
scanf("%d",&s);
printf("Reachability of vertex %d\n",s);
for(v=1;v<=n;v++)
visit[v]=0;
bfs(s);
if(flag==0)
{
printf("No path found!!\n");
}
}
```

OUTPUT:

```
Enter the no.of vertices : 3
Enter adjacency matrix :
1 3 2
3 2 1
2 3 1
Enter the start vertex : 2
Reachability of vertex : 2
```

RESULT: Hence, the above Breadth first Search (BFS) is executed Successfully.

```
while(!qempty())
{
v=deleteq();
for(w=1;w<=n;w++)

if((adj[v][w]==1) && (visit[w]==0))
{
visit[w]=1;
flag=1;
printf("v%d\t",w);
insertq(w);
}
}
}
void main()
{
intv,w;
printf("Enter the no.of vertices:\n");
scanf("%d",&n);
printf("Enter adjacency matrix:");
for(v=1;v<=n;v++)
{
for(w=1;w<=n;w++)
scanf("%d",&adj[v][w]);
}
printf("Enter the start vertex:");
scanf("%d",&s);
printf("Reachability of vertex %d\n",s);
for(v=1;v<=n;v++)
visit[v]=0;
bfs(s);
if(flag==0)
{
printf("No path found!!\n");
}
}
```

OUTPUT:

Enter the no.of vertices : 3
Enter adjacency matrix :
1 3 2
3 2 1
2 3 1
Enter the start vertex : 2

RESULT: Reachability of vertex : 2

Hence, the above Breadth First Sea

Ex.No-6	PRIM'S ALGORITHM.	Date: 10/4/24

Aim:

To Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Description:

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Algorithm:

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

Source code:

```
# include <stdio.h>
int Prim (int g[20][20], int n, int t[20][20])
{
intu,v, min, mincost;
int visited[20];
inti,j,k;
visited[1] = 1;
for(k=2; k<=n; k++)
visited[k] = 0 ;
mincost = 0;
for(k=1; k<=n-1; k++)
{
min= 99;
u=1;
v=1;
for(i=1; i<=n; i++)
if(visited[i]==1)
for(j=1; j<=n; j++)
if( g[i][j] < min )
{
min = g[i][j];
u = i;   v = j;
}
t[u][v] = t[v][u] = g[u][v] ;
```

```
mincost = mincost + g[u][v] ;
visited[v] = 1;
printf("\n (%d, %d) = %d", u, v, t[u][v]);
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
if( visited[i] && visited[j] )
g[i][j] = g[j][i] = 99;
}
return(mincost);
}
void main()
{
int n, cost[20][20], t[20][20];
intmincost,i,j;
printf("\nEnter the no of nodes: ");
scanf("%d",&n);
printf("Enter the cost matrix:\n");
for(i=1; i<=n; i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=99;
}
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
t[i][j] = 99;
printf("\nThe order of Insertion of edges:");
mincost = Prim (cost,n,t);
printf("\nMinimum cost = %d\n\n", mincost);
}
```

OUTPUT:

Enter the no of nodes : 4
Enter the cost matrix :

1	2	4	6
2	4	5	6
2	1	7	4
2	4	5	7

The order of insertion of edges :
(1,1) = 1
(1,2) = 2
(1,3) = 4
Minimum cost = 7

RESULT: Hence, The minimal cost spanning t
of a given undirected graph using Prim's
algorithm is Sucessfully executed.