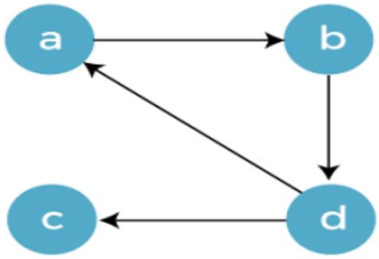
INDEX

Experiment No: 01 **Date:** 18-09-23

Experiment Name: Compute the transitive closure of a given directed graph using Warshall's Algorithm.



Source Code:

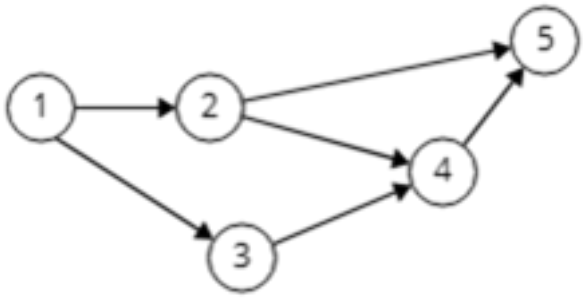
```
#include <stdio.h>
int n,a[10][10],p[10][10];
void path() {
	int i,j,k;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			p[i][j]=a[i][j];
	for(k=0;k<n;k++)
		for(i=0;i<n;i++)
			for(j=0;j<n;j++)
				if(p[i][k]==1&&p[k][j]==1)
					p[i][j]=1;
}
void main() {
	int i,j;
	printf("Enter the number of nodes:");
	scanf("%d",&n);
	printf("\nEnter the adjacency matrix:\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&a[i][j]);
	path();
	printf("\nThe path matrix is shown below\n");
	for(i=0;i<n;i++) {
		for(j=0;j<n;j++)
			printf("%d ",p[i][j]);
		printf("\n");
	}
}
```

Output:

```
Enter the number of nodes:4

Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0

The path matrix is shown below
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1

Process returned 4 (0x4)   execution time : 22.535 s
Press any key to continue.
```

Experiment No: 02 **Date:** 18-09-23

Experiment Name: Obtain the Topological ordering of vertices in a given digraph



Source Code:

```
#include<stdio.h>
int a[10][10],n,indegre[10];
void find_indegre() {
        int j,i,sum;
        for(i=1;i<n;i++) {
                sum=0;
                for(j=0;j<n;j++)
                        sum+=a[j][i];
                indegre[i]=sum;
        }
}
void topology() {
        int i,u,v,stor[10],stk[10],top= -1,k=0;
        find_indegre();
        for(i=0;i<n;i++) {
                if(indegre[i]==0)
                        stk[++top]=i;
        }
        while(top!= -1) {
                u=stk[top--];
                stor[k++]=u;
                for(v=0;v<n;v++) {
                        if(a[u][v]==1) {
                                indegre[v]--;
                                if(indegre[v]==0)
                                        stk[++top]=v;
                        }
                }
        }
        printf ("\nThe topological Sequence is:\n");
        for(i=0;i<n;i++)
                printf ("%d ",stor[i]);
}
```

```
void main() {
        int i,j;
        printf("Enter number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=0;i<n;i++) {
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
        }
        topology();
}
```

Output:

```
Enter number of nodes:5

Enter the adjacency matrix:
0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0

The topological Sequence is:
0 2 1 3 4
Process returned 5 (0x5)   execution time : 6.215 s
Press any key to continue.
```

Experiment No: 03 **Date:** 02-10-23

Experiment Name: Counting inversion using bubble sort.

Source Code:

```
#include<stdio.h>

int InvCnt(int arr[], int n){
	int count=0;
	for(int i=0; i<n-1; i++){
		for(int j=i+1; j<n; j++){
			if(arr[i] > arr[j]){
				count++;
			}
		}
	}
	return count;
}

int main(){
	int n;
	printf("Enter array size: ");
	scanf("%d", &n);
	int arr[n];
	printf("Enter array elements: ");
	for(int i=0; i<n; i++){
		scanf("%d", &arr[i]);
	}
	printf("Number of inversions are: %d", InvCnt(arr, n));

}
```

Output:

```
Enter array size: 7
Enter array elements: 16 7 9 4 13 8 10
Number of inversions are: 11
Process returned 0 (0x0)   execution time : 50.612 s
Press any key to continue.
```

Experiment No: 04 **Date:** 09-10-23

Experiment Name: Implement 0/1 Knapsack problem using Dynamic Programming.

Source Code:

```
#include<stdio.h>

int wght[10], prft[10], tempTable[10][10], obj, i, j, capacity, selected_obj[10] = {0};

int max_f(int a, int b) {
	return ((a > b) ? a : b);
}
int knapsack_f(int obj_indx, int currentCapacity) {
	int value;
	if (tempTable[obj_indx][currentCapacity] < 0) {
		if (currentCapacity < wght[obj_indx]) {
			value = knapsack_f(obj_indx - 1, currentCapacity);
		} else {
			value = max_f(knapsack_f(obj_indx - 1, currentCapacity), prft[obj_indx] +
			knapsack_f(obj_indx - 1, currentCapacity - wght[obj_indx]));
		}
		tempTable[obj_indx][currentCapacity] = value;
	}
	return tempTable[obj_indx][currentCapacity];
}

int main() {
	int t_prft, cnt = 0;

	printf("\nEnter the number of objects ");
	scanf("%d", &obj);

	printf("Enter the profits and weights of the elements\n");
	for (i = 1; i <= obj; i++) {
		printf("\nEnter profit and weight for object no %d: ", i);
		scanf("%d %d", &prft[i], &wght[i]);
	}

	printf("\nEnter the knapsack capacity ");
	scanf("%d", &capacity);

	for (i = 0; i <= obj; i++) {
		for (j = 0; j <= capacity; j++) {
			if ((i == 0) || (j == 0)) {
				tempTable[i][j] = 0;
			} else {
				tempTable[i][j] = -1;
			}
		}
	}
```

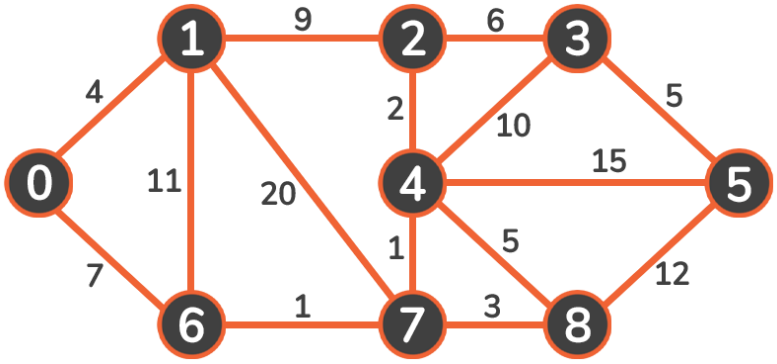
```
        t_prft = knapsack_f(obj, capacity);
        i = obj;
        j = capacity;
        while (j != 0 && i != 0) {
                if (tempTable[i][j] != tempTable[i - 1][j]) {
                        selected_obj[i] = 1;
                        j = j - wght[i];
                        i--;
                } else {
                        i--;
                }
        }
        printf("\nObjects included in the knapsack:\n");
        printf("Sl.no\tWeight\tProfit\n");
        for (i = 1; i <= obj; i++) {
                if (selected_obj[i]) {
                        printf("%d\t%d\t%d\n", ++cnt, wght[i], prft[i]);
                }
        }
        printf("Total profit = %d\n", t_prft);
        return 0;
}
```

Output:

```
Enter the number of objects 4
Enter the profits and weights of the elements

Enter profit and weight for object no 1: 2 3

Enter profit and weight for object no 2: 3 4

Enter profit and weight for object no 3: 1 6

Enter profit and weight for object no 4: 4 5

Enter the knapsack capacity 8

Objects included in the knapsack:
Sl.no   Weight  Profit
1       3       2
2       5       4
Total profit = 6

Process returned 0 (0x0)   execution time : 22.486 s
Press any key to continue.
```

Experiment No: 05 **Date:** 06-11-23

Experiment Name: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.



Source Code:

```
#include <stdio.h>
#define INFINITY 999

void dijkstra_f(int nodes, int src_node, int adj_mat[20][20], int short_dist[]) {
        int i, select_node, count, currentWeight, minWeight;
        int visited[20];
        for (i = 1; i <= nodes; i++) {
                visited[i] = 0;
                short_dist[i] = adj_mat[src_node][i];
        }
        visited[src_node] = 1;
        count = 2;

        while (count <= nodes) {
                minWeight = INFINITY;
                for (i = 1; i <= nodes; i++) {
                        if (short_dist[i] < minWeight && !visited[i]) {
                                minWeight = short_dist[i];
                                select_node = i;
                        }
                }
                visited[select_node] = 1;
                count++;
                for (i = 1; i <= nodes; i++) {
                        if ((short_dist[select_node] + adj_mat[select_node][i] < short_dist[i]) &&
                            !visited[i]) {
                                short_dist[i] = short_dist[select_node] + adj_mat[select_node][i];
                        }
                }
        }
}
```

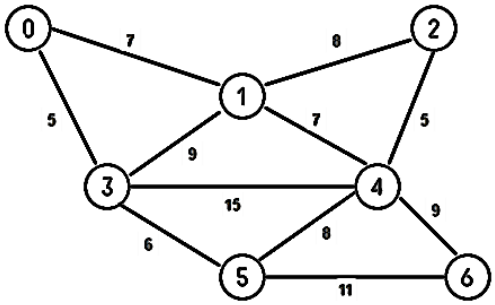
```
int main() {
	int nodes, src_node, i, j, adj_mat[20][20], short_dist[20];
	printf("Enter the number of nodes: ");
	scanf("%d", &nodes);
	printf("\nEnter the adjacency matrix:\n");
	for (i = 1; i <= nodes; i++) {
		for (j = 1; j <= nodes; j++) {
			scanf("%d", &adj_mat[i][j]);
				if (adj_mat[i][j] == 0)
					adj_mat[i][j] = INFINITY;
		}
	}
	printf("\nEnter the source node: ");
	scanf("%d", &src_node);
	dijkstra_f(nodes, src_node, adj_mat, short_dist);
	printf("\nShortest paths:\n");
	for (i = 1; i <= nodes; i++) {
		if (i != src_node) {
			printf("%d -> %d, cost = %d\n", src_node, i, short_dist[i]);
		}
	}
	return 0;
}
```

Output:

```
Enter the number of nodes: 9

Enter the adjacency matrix:
0 4 0 0 0 0 7 0 0
4 0 9 0 0 0 11 20 0
0 9 0 6 2 0 0 0 0
0 0 6 0 10 5 0 0 0
0 0 2 10 0 15 0 1 5
0 0 0 5 15 0 0 0 12
7 11 0 0 0 0 0 1 0
0 20 0 0 1 0 1 0 3
0 0 0 0 5 12 0 30 0

Enter the source node: 4

Shortest paths:
4 -> 1, cost = 17
4 -> 2, cost = 15
4 -> 3, cost = 6
4 -> 5, cost = 8
4 -> 6, cost = 5
4 -> 7, cost = 10
4 -> 8, cost = 9
4 -> 9, cost = 12

Process returned 0 (0x0)   execution time : 10.168 s
Press any key to continue.
```

Experiment No: 06 **Date:** 04-12-23

Experiment Name: Find the Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.



Source Code:

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

int i, j, k, vrtx1, vrtx2, u, v, n_vrtx, edge_cnt = 1;
int min_wght, t_min_cost = 0, adj_mat[9][9], parent[9];

int findSet_f(int);
int unionSet_f(int, int);
void main() {
	printf("\nKruskal's Algorithm Implementation\n\n");
	printf("\nEnter the number of vertices\n");
	scanf("%d", &n_vrtx);
	printf("\nEnter the cost adjacency matrix\n");
	for (i = 1; i <= n_vrtx; i++) {
		for (j = 1; j <= n_vrtx; j++) {
			scanf("%d", &adj_mat[i][j]);
			if (adj_mat[i][j] == 0)
				adj_mat[i][j] = 999;
		}
	}
	printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");
	while (edge_cnt < n_vrtx) {
		for (i = 1, min_wght = 999; i <= n_vrtx; i++) {
			for (j = 1; j <= n_vrtx; j++) {
				if (adj_mat[i][j] < min_wght) {
					min_wght = adj_mat[i][j];
					vrtx1 = u = i;
					vrtx2 = v = j;
				}
			}
		}
		u = findSet_f(u);
		v = findSet_f(v);
```

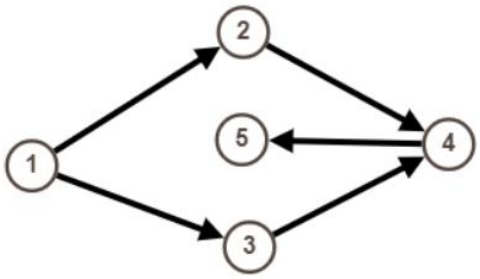
```
                if (unionSet_f(u, v)) {
                        printf("\n%d edge (%d,%d) = %d\n", edge_cnt++, vrtx1, vrtx2, min_wght);
                        t_min_cost += min_wght;
                }
                adj_mat[vrtx1][vrtx2] = adj_mat[vrtx2][vrtx1] = 999;
        }
        printf("\n\tTotal Minimum Cost = %d\n", t_min_cost);
}
int findSet_f(int vertex) {
        while (parent[vertex])
                vertex = parent[vertex];
   return vertex;
}
```

Output:

```
Enter the number of vertices
7

Enter the cost adjacency matrix
0 7 0 5 0 0 0
7 0 8 9 7 0 0
0 8 0 0 5 0 0
5 9 0 0 15 6 0
0 7 5 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0

The edges of Minimum Cost Spanning Tree are


1 edge (1,4) = 5

2 edge (3,5) = 5

3 edge (4,6) = 6

4 edge (1,2) = 7

5 edge (2,5) = 7

6 edge (5,7) = 9

        Total Minimum Cost = 39

Process returned 26 (0x1A)   execution time : 55.065 s
Press any key to continue.
```

Experiment No: 07 **Date:** 11-12-23

Experiment Name: Print all the nodes reachable from a given starting node in a digraph using BFS/DSF method.



Source Code:

```
#include<stdio.h>
#include<conio.h>

int adj_mat[20][20], queue[20], visited[20], n_vrtx, i, j, front = -1, rear = 0;
void BFS_f(int strt_vrtx) {
        queue[++rear] = strt_vrtx;
        visited[strt_vrtx] = 1;
        while (front <= rear) {
                for (i = 1; i <= n_vrtx; i++) {
                        if (adj_mat[strt_vrtx][i] && !visited[i]) {
                                visited[i] = 1;
                                queue[++rear] = i;
                        }
                }
                front++;
                strt_vrtx = queue[front];
        }
}

void main() {
        int strt_vrtx;
        printf("\nEnter the number of vertices: ");
        scanf("%d", &n_vrtx);
        for (i = 1; i <= n_vrtx; i++) {
                queue[i] = 0;
                visited[i] = 0;
        }
        printf("\nEnter graph data in matrix form:\n");
        for (i = 1; i <= n_vrtx; i++)
                for (j = 1; j <= n_vrtx; j++)
                        scanf("%d", &adj_mat[i][j]);
```

```
        printf("\nEnter the starting vertex: ");
        scanf("%d", &strt_vrtx);
        BFS_f(strt_vrtx);
        printf("\nThe nodes which are reachable are:\n");
        for (i = 1; i <= n_vrtx; i++) {
                if (visited[i]) {
                        printf("%d\t", queue[i]);
                } else {
                        printf("\nBFS is not possible");
                }
        }
}
```

Output:

```
Enter the number of vertices: 5

Enter graph data in matrix form:
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0

Enter the starting vertex: 1

The nodes which are reachable are:
1       2       3       4       5
Process returned 5 (0x5)   execution time : 12.349 s
Press any key to continue.
```

Experiment No: 08 **Date:** 18-12-23

Experiment Name: Implement Longest Common subsequence (LCS).

Source Code:

```
#include <stdio.h>
#include <string.h>

int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];

void lcsAlgo(){
	m = strlen(S1);
	n = strlen(S2);
	for (i = 0; i <= m; i++)
		LCS_table[i][0] = 0;
	for (i = 0; i <= n; i++)
		LCS_table[0][i] = 0;
	for (i = 1; i <= m; i++){
		for (j = 1; j <= n; j++){
			if (S1[i - 1] == S2[j - 1]){
				LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
			}
			else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]){
				LCS_table[i][j] = LCS_table[i - 1][j];
			}
			else{
				LCS_table[i][j] = LCS_table[i][j - 1];
			}
		}
	}
	int index = LCS_table[m][n];
	char lcsAlgo[index + 1];
	lcsAlgo[index] = '\0';
	int i = m, j = n;
	while (i > 0 && j > 0){
		if (S1[i - 1] == S2[j - 1]){
			lcsAlgo[index - 1] = S1[i - 1];
			i--;
			j--;
			index--;
		}
		else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
			i--;
		else
			j--;
	}
	printf("S1 : %s \nS2 : %s \n", S1, S2);
	printf("LCS: %s", lcsAlgo);
}
```

```
int main()
{
lcsAlgo();
printf("\n");
}
```

Output:

```
S1 : abaaba
S2 : babbab
LCS: baba

Process returned 0 (0x0)   execution time : 0.121 s
Press any key to continue.
```