Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
01.	Find the Number of edge in the graph ?	1-2			
02.	C++ program for finding the vertices, even & odd vertices in a graph ?	3-4			
03.	C++ program for find the union, intersection, ring sum, Cartesian product of two graph ?	5-8			31/07/25
04.	C++ program for finding solutions of Travelling Salesman problem.	9-13			
05.	C++ program for finding shortest path between two vertices using Dijkstra Algorithm ?	14-18			

Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
06.	C++ Program for find spanning tree using Kruskal's Algorithm?	19-27			
07.	C++ program for find minimum spanning tree using prim's Algorithm?	28-31			Comp

Expt. No. 01 Page No. 1

Object :-

Find the Number of edges in the graph ?

Source Code :-

C++ Program to find number of edges in the graph given as

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

// function to insert vertices
void insert ( list < int > graph - list [ ], int u, int v )
    } graph - list [u] . push - back (v) ;
    graph - list [v] . push - back (u) ; ,
{

// function to count the total number of edges void count
- edges ( list < int > graph - list [ ], int(v) } int count = 0 ;

// traverse the loop till the vertice is found for ( int i = 0 ;
{ i < v, i ++ ) } count + = graph - list [ i ] . size ( ) ; }

# count = count / 2 ;
count << " count of edges are :" << count ; }
}

int main ( int argc, char* argv [ ] ) }
// creating 5 vertices in a graph
```

Teacher's Signature

```
int vertices = 5;

// declare list to creat a graph and pass the vertices
list <int> graph = list [vertices];

// Call insert function passing the list variable,
    vertice, linked vertice

insert (graph - list, 0, 1);
insert (graph - list, 0, 2);
insert (graph - list, 1, 2);
insert (graph - list, 1, 4);
insert (graph - list, 2, 4);
insert (graph - list, 2, 3);
insert (graph - list 3, 4);

// Calling count function that will count the edges
count - edges (graph - list, vertices); return 0;
}
```

out put :-

Count of edges are : 7

Expt. No. 02 Date

Page No. 3

object :-

C++ program for finding the vertices, even vertices, odd vertices in a graph ?

source code :-

```
// program to find the Number of vertices, odd vertices and
// Number of Edges in a graph

# include < bits/ stdc ++.h >
Using namespace std;
int main ( ) {
        int n;
        cin >> n;
        int a [n] [n];
        for (int i=0, j<n ; i++ ] {
            for (int j =0 ; j <n ; j++ ) {
                cin >> a [i] [j] ;
        }
    }
    int degree [n] = } } ;
    for ( int i = 0 ; i <n ; i++ ) }
        for (int j =0 j j<n } ; j++ ) }
            degree [i] + = a [i] [j] ;
    }
}
```

Teacher's Signature

```
int even = 0, odd = 0, edges = 0;
for (int i = 0; i < n; i++) {

    even += (degree[i] % 2 == 0);
    odd += (degree[i] % 2 != 0);
    edges += degree[i];
}
cout << "number of even vertices" << " = " << even << '\n';

cout << "number of odd vertices" << " = " << odd << '\n';

cout << "number of edges" << " = " << edges/2 << '\n';
}
```

Expt. No. 03 Date

Page No. 5

object :-

C++ program for find the union, intersection, ring sum, Cartesian product of two graph?

Source Code :-

```
#include <iostream>
#include <vector>

using namespace std;

bool isvalid (vector < vector < int >> & adj ) {
    int n = adj . size ( ) ;
    for (int i = 0 ; i < n ; i++ ) {
     for (int j = 0 ; j < n ; j++ ) {
      if (adj [i] [j] ! = adj [j] [i] ) {
          return false ;
      }
     }
    }
    return true ;
}

void union (vector < vector < int >> & adj 1, vector < vector <
                                        int >> & adj 2 ) {
    cout << " union ... " << endl ( ) ;
    int n1 = adj 1 . size ( ) ;
    int n 2 = adj 2 . size ( ) ;
```

Teacher's Signature

```
    int n = max (n1, n2);
vector < vector < int >> adj (n, vector < int > (n));
 for (int i=0 , i<n , i++) {
  for (int j=0, j<n , j++) {
   adj [i] [j] = (i < n1 && j<n2) ? adj 1 [i] [j] : 0)
   || (i<n2 && ( j<n2) ? adj 2 [i] [j] :0);
      Cout << adj [i] [j] << " ";
  {
    Cout << end
  }
    Cout << end
}
 void intersection (vector < vector < int >> & adj 1, vector <
          vector < int >> & adj 2) {
  Cout << " intersection ... " << end ( ;
    int n1 = adj 1. size ( ) ;
    int n2 = adj 2. size ( ) ;
    int n = min (n1, n2)
vector < vector < int >> adj (n, vector < int > (n) ) ;
  for (int i=0 ; i<n , i++ ) {
   for (int j =0 ; j<n ; j++ ) {
     adj [i] [j] = adj 1 [i] [j] && adj 2 [i] [j];
   Cout << adj [i] [j] << " " ";
  }
    Cout << end l ;
 }
   Cout << end l ;
}
```

Expt. No. Date

Page No. 7

```
void Ring sum (vector < vector < int >> & adj 1, vector < vector <
                    int >> & adj 2) {
  Cout << " Ring Sum ..." << endl ;
   int n1 = adj 1. size ( ) ;
   int n2 = adj 2. size ( ) ;
   int n = max (n1, n2) ;
vector < vector < int >> adj (n, vector < int > (n) ) ;
   for (int i=0 , i<n , i++ ) {
      for (int j=0 , j<n , j++) {
   adj [i] [j] = ((i<n2 && j<n1) ? adj 1 [i] [j] : 0
                  ^ ((i<n2 && (j<n2) ? adj 2 [i] [j] : 0) ;
      Cout << adj [i] [j] << "  " ;
   }  Cout << endl ( ;
}

   Cout << endl ( ;
}


int main ( ) {
#  ifndef  ONLINE - JUDGE
   freopen (" input . txt", "r", stdin) ;
   freopen (" output . txt", "w", stdout ) ;
#  endif
        int n1 ;
        Cin >> n1 ;
vector < vector < int >> adj 1 (n1, vector < int > (n2) ) ;
 for (int i=0, i<n, i++) {
  for (int j=0, j<n, j++ ) {
    Cin >> adj 1 [i] [j] ;
```

Teacher's Signature

```
{
    int n2;
    cin >> n2;
    vector < vector < int >> adj2 (n2, vector < int > (n2));
    for (int i=0 , i<n2, i++) {
    for (int j=0, j<n2 , j++) {
        cin >> adj2 [i] [j];
    }
}
if ( is valid (adj1) && is valid (adj2)) {
    union (adj1, adj2);
intersection (adj1, adj2);
    Ring Sum (adj1, adj2);
{
    else {
        Cout << " Entered Graph is valid... " << endl;
    }
    return 0;
}
```

Expt. No. 04 Date

Page No. 9

Object :-

C++ program for finding solutions of Travelling Salesman problem.

Source Code :-

```
#include <iostream>

using namespace std;

int ary[10][10], completed[10], n, cost = 0;

void takeInput()
{
    int i, j;

    cout << "Enter the number of villages : ";
    cin >> n;

    cout << "\n Enter Elements of Row : " << i+1 << "\n";

    for (j=0; j<n; j++)
        cin >> ary[i][j];

        completed[i] = 0;
}

    cout << "\n\n The cost list is :";

for (i=0; i<n; i++)
```

Teacher's Signature

```
{
    cout << "\n" ;

for ( j=0 ; j<n ; j++)
 cout << " \t" << ary [i] [j] ;
}
}

int least (int c)
{
int i, nc = 999 ;
int min = 999 , kmin ;

for (i=0 ; i<n ; i++)
{
 if ( (ary [c] [i] != 0) && (completed [i] == 0))
 if ( ary [c] [i] + ary [i] [c] < min)
{
 min = ary [i] [0] + ary [c] [i] ;
    nc = i ;
 }
 }

 if (min != 999)
 cost + = kmin ;

 return nc ;
 }
```

Expt. No. _____ Date _____

Page No. 11

```
void min cost (int city)
{
  int i, n city ;

  Completed [ city ] = 1 ;

  cout << city + 1 << " ---> " ;
  n city = least (city) ;

  if (n city == 999)
{
  n city = 0 ;
  cout << n city + 1 ;
  cost + = arr [city] [n city] ;
  return ;
}

  min cost (n city) ;
}

int main ( )
{
  take input ( ) ;

cout << " \n\n The Path is : \n " ;
min cost (0) ; // passing 0 because starting vertex

cout << " \n\n minimum Cost is " << cost ;
```

Teacher's Signature _____

```
    return 0;
}
```

output :-

```
Enter the number of village : 4
Enter the Cost matrix
Enter Element of Row : 1
 0  4  1  3
Enter Element of Row : 2
 4  0  2  1
Enter Element of Row : 3
 1  2  0  5
Enter Element of Row : 4
 3  1  5  0
The Cost List is :
 0  4  1  3
 4  0  2  1
 1  2  0  5
 3  1  5  0
```

The path is :

1 ---> 3 ---> 2 ---> 4 ---> 1

minimum cost is : 7

Expt. No. 05 Date

Page No. 14

Object :-

C++ program for finding shortest path between two vertices using Dijkstra Algorithm?

Source Code :-

```
// A C++ program for Dijkstra's single source path algorithm.

// The program is for adjacency matrix representation of
                the graph.

#include <limits.h>
#include <stdio.h>

// Number of vertices in the graph
#define V9

// A utility function of find the vertex with minimum
        distance value, from
// the set of vertices not yet included in shortest
        path tree
int minDistance (int dist[], bool sptSet[])
{
    // initialize min value
    int min = INT_MAX, min_index;

    for (int v=0; v<V; v++)
    if (sptSet[v] == false && dist[v] <= min
        min = dist[v], min_index = v;
```

Teacher's Signature

```
    return min_index;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[7])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i=0; i<V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's Single Source
// Shortest path algorithm
// For a graph represented using adjacency matrix
// representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array. dist[i] will
                 // hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be
                    // true if vertex i is included in shortest

    // path tree or shortest distance from src to i is
    // finalized
```

// Initialized all distance as INFINITE and spt set [ ] as false

```
for (int i=0 ; i< V ; i++)
    dist [i] = INT - MAX , sptset [i] = false ;
```

// Distance of source vertex from itself is always 0

```
dist [src] = 0 ;
```

// Find shortest path for all vertices

```
for (int count = 0 ; count < V-1 ; count ++ ) {
```

// Pick the minimum distance vertex from the set of vertices not

// yet processed. u is always equal to src in the first iteration

```
int u = min Distance (dist , spt set) ;
```

// marked the picked vertex as processed

```
spt set [u] = true ;
```

// update dist value of the adjacent vertices of the picked vertex.

```
for (int v=0 ; v<V ; v++)
```

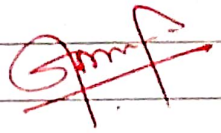
// update dist [v] only if is not in spt set, there is an edge from

// u to v , and total weight of path from src to v through u is

```
// smaller than current value of dist[v]
if (!sptSet[v] && graph[u][v] && dist[u] !=
                                    INT_MAX
    && dist[u] + graph[u][v] < dist[v])
dist[v] = dist[u] + graph[u][v];
}

// Print the constructed distance array printSolution(dist);
}

// driver program to test above function int main() {

/* let us create the example graph discussed above */
int graph[V][V] = {{0,4,0,0,0,0,8,0},
                   {4,0,8,0,0,0,0,1,1,0},
                   {0,8,0,7,0,4,0,0,2},
                   {0,0,7,0,9,14,0,0,0},
                   {0,0,0,9,0,10,0,0,0},
                   {0,0,4,14,10,0,2,0,0},
                   {0,0,0,0,0,2,0,1,6},
                   {8,11,0,0,0,0,1,0,7},
                   {0,0,2,0,0,0,6,7,0}};

dijkstra(graph,0);
return 0;
}
```

Output :-

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14



Expt. No. 06 Date

Page No. 19

object :-

C++ program for find spanning tree using Kruskal's Algorithm ?

Source Code :-

```
#include <bits/stdc++.h>

using namespace std;

// a structure to represent a
// weighted edge in graph
class Edge {
public:
    int src, dest, weight;
};

// a structure to represent a Connected,
// undirected and weighted graph
class Graph {
public:
    // V-> Number of vertices, E-> Number of edges
    int V, E;
```

```
// graph is represented as an array of edges.
// Since the graph is undirected, the edge
// from src to dest is also edge from dest
// to src. Both are counted as 1 edge here.
  Edge * edge;

};

// Creates a graph with V vertices and E edges
  Graph * CreateGraph (int V, int E)

{
    Graph* graph = new Graph;
     graph -> V = V;
     graph -> E = E;

graph -> edge = new Edge [E];

    return graph;
}

// A structure to represent a subset for union-find
  class subset {
    public:
      int parent;
      int rank;
};
```

Expt. No.

Date

Page No. 21

```
// A utility function to find set of an element i
// (uses path compression technique)
int find (subset subset [], int i)
{
    // find root and make root as parent of i
    // (path compression)
    if (subsets [i].parent != i)

        subsets [i].parent
            = find (subsets, subset [i].parent);

    return subsets [i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void union (subset subsets [], int x, int y)
{

    int xroot = find (subsets, x);
    int yroot = find (subsets, y);

// Attach smaller rank tree under root of high
// rank tree (union by Rank)
    if (subsets [xroot].rank < subsets [yroot].rank)
```

Teacher's Signature

```
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;

    // if ranks are same, then make one as root and
    // increment its rank by one
    else {
            subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Compare two edge according to their weights.
// used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}

// The main function to construct MST using Kruskal's
// algorithm
void KruskalMST(Graph* graph)
{

    int V = graph->V;
    Edge result[V]; // This will store the result and MST
```

```
int e = 0 ; // An index variable, used for result[]
int i = 0 ; // An index variable, used for sorted edges

// Step 1 : Sort all the edges in non-decreasing
// order of their weight. If we are not allowed to
// change the given graph, we can creat a copy of
// array of edges

qsort ( graph->edges, graph->E, sizeof (graph->edge
                                        [0]),
        myComp );

// Allocate memory for creating V subsets
Subset* subsets = new subset [(V* sizeof (subset))];

// Create V subsets with single elements
for (int v=0 ; v<V ; ++v)
{

    subset [v]. parent = v;
    subsets [v]. rank = 0 ;
}

// Number of edges to be taken is equal to V,
while (e<V-1 && i < graph->E)
{
// Step 2 : Pick the smallest edge. And increment
```

Teacher's Signature

```
// the index for the next iteration
    Edge next_edge = graph -> edge [i++];

    int x = find (subsets, next_edge.src);
    int y = find (subsets, next_edge.dest);

// if including this edge doesn't cause cycle,
// include it in result and increment the index
// of result for next edge
    if (x != y) {

            result [e++] = next_edge;
            Union (subsets, x, y);
    }

    // Else discard the next_edge
}

// print the contents of result[] to display the
// built MST

Cout << " Following are the edges in the Constructed "
                "MST\n" ;

int minimum_cost = 0;
for ( i=0; i<e; ++i)
{
```

Teacher's Signature _____

```
        cout << result[i].src << " -- " << result[i].dest
             << " == " result[i].weight << endl;

        minimum_cost = minimum_cost + result[i].weight;
    }

    // return;
    cout << " minimum cost spanning Tree: " << minimum
                cost << endl;
}

// Driver Code
int main()
{

/* let us create following weighted graph

                10
        0 ---------- 1

        |\ |
        6| 5\ |15

        |\ |
        2 ---------- 3

        4*/
```

Expt. No. _____ Date _____

Page No. 26

```
int V = 4; // Number of vertices in graph
int E = 5; // Number of edges in graph

Graph* graph = Create Graph (V, E);

// add edge 0-1
graph -> edge [0]. src = 0;
graph -> edge [0]. dest = 1;
graph -> edge [0]. weight = 10;

// add edge 0-2
graph -> edge [1]. src = 0;
graph -> edge [1]. dest = 2;
graph -> edge [1]. weight = 6;

// add edge 0-3
graph -> edge [2]. src = 0;
graph -> edge [2]. dest = 3;
graph -> edge [2]. weight = 5;

// add edge 1-3
graph -> edge [3]. src = 1;
graph -> edge [3]. dest = 3;
graph -> edge [3]. weight = 15;
```

Teacher's Signature _____

```
// add edge 2-3
graph->edge[7].src = 2;
graph->edge[7].dest = 3;
graph->edge[7].weight = 4;

// Function Call
kruskalMST(graph);

return;
}
```

output :-

Following are the edges in the Constructed MST

2 --- 3 == 4
0 --- 3 == 5
0 --- 1 == 10

minimum cost spanning Tree : 19

Expt. No. 07 Date

Page No. 28

object :-

C++ program for find minimum spanning tree using prim's Algorithm ?

Source Code :-

```
# include < bits/stdc++.h >
using name space std ;

# define INF 1e9

typedef pair < int , int > pil ;

vector < vector < pil >> adj ; // adjancency list of the graph

vector < bool > visited , // visited array to keep track of
                          visited vertices

int prim ( int start ) {
        int n = adj.size() ;
        int mst weight = 0 ;
    priority - queue < pil , vector <pii> ,
    greater < pii >> pq ; // priority queue to
        select the ~~mst~~ next edge to add to
        the MST
```

Teacher's Signature

```
pq.push(make_pair(0, start)); // start from the
        given vertex with weight 0

while (!pq.empty()) {
    int u = pq.top().second;
    int w = pq.top().first;
        pq.pop();

if (visited[u]) continue; // skip visited vertices
    visited[u] = true;
    mst_weight += w;

    for (auto& v : adj[u]) {
        if (!visited[v.first]) {
            pq.push(make_pair(v.second, v.first));
        // add edges connected to the current vertex
        }
    }
}

    return mst_weight;
}

int main() {
    // define the graph
    adj.resize(5); // 5 vertices

// add edges to the graph
```

adj[0].push_back(make_pair(1,2)); // edge from vertex 0 to 1 with weight 2
adj[1].push_back(make_pair(0,2)); // edge from vertex 1 to 0 with weight 2

adj[0].push_back(make_pair(3,6)); // edge from vertex 0 to 3 with weight 6
adj[3].push_back(make_pair(0,6)); // edge from vertex 3 to 0 with weight 6

adj[1].push_back(make_pair(2,3)); // edge from vertex 1 to 2 with weight 3
adj[2].push_back(make_pair(1,3)); // edge from vertex 2 to 1 with weight 3

adj[1].push_back(make_pair(3,8)); // edge from vertex 1 to 3 with weight 8
adj[3].push_back(make_pair(1,8)); // edge from vertex 3 to 1 with weight 8

adj[1].push_back(make_pair(4,5)); // edge from vertex 1 to 4 with weight 5
adj[4].push_back(make_pair(1,5)); // edge from vertex 4 to 1 with weight 5

```
adj[2].push_back(make_pair(4,7)); // edge from
        vertex 2 to 4 with weight 7
adj[4].push_back(make_pair(2,7)); // edge from
        vertex 4 to 2 with weight 7

adj[3].push_back(make_pair(4,9)); // edge from
        vertex 3 to 4 with weight 9
adj[4].push_back(make_pair(3,9)); // edge from
        vertex 4 to 3 with weight 9

visited.resize(5, false); // set all vertices as
                          unvisited
int mst_weight = prim(0); // find the MST
                  starting from vertex 0
cout << "minimum spanning tree weight".
        << mst_weight << endl;
return 0;
{
```

output :-

Minimum spanning tree weight : 16

31/07/25