VAISHNO COLLEGE OF ENGINEERING

Affliated to HPTU, Hamirpur and approved by AICTE

DAA

Lab Manual

CSPC-413P (CPCS Syllabus)

Department of Computer Science Engineering

Vill Thapkour, PO Bhardoya, Tehsil Indora,Distt. Kangra (HP)-176403

Contact: 094183-18394, Web: www.vaishno.edu.in

Vision of Institute

To emerge as an institute of eminence in the fields of engineering, technology and management in serving the industry and the nation by empowering students with a high degree of technical managerial and practical competence.

Mission of Institute

M1 To strengthen the theoretical, practical and ethical dimensions of the learning process by fostering a cultural of research and innovation among faculty members and students.

M2 To encourage long term interaction between academia and industry through the involvement of industry for hands on implementation of the curriculum.

M3 To strengthen and molding students in professional ethical, social and environmental dimensions by encouraging participation in co-curricular extracurricular and CSR activities.

Vision of the Department

To emerge as a department of eminence in computer science and engineering in serving the industry and the nation by empowering students with high degree of technical and practical competence.

Mission of the department

M1 To strengthen the theoretical and practical aspects of learning process by strongly encouraging a computer cultural of research, innovation and hands on learning in computer science and engineering

M2 To encourage long term interaction between the department and IT industry, through the involvement of IT industry for hands on implementation of course curriculum.

M3 To widen the awareness of students in professional, ethical, social and environmental dimensions by encouraging their participation in co-curricular extracurricular and CSR activities.

Program Educational Objectives (PEOs) of the department

PEO 1: Engage in successful careers in industry, academia, and public service, by applying the acquired knowledge of Science, Mathematics and Engineering, providing technical leadership for their business, profession and community

PEO 2: Establish themselves as entrepreneur, work in research and development organization and pursue higher education

PEO 3: Exhibit commitment and engage in lifelong learning for enhancing their professional and personal capabilities.

PROGRAM OUTCOMES

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able tocomprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcome (PSOs)

PSO1: Apply knowledge of mathematics, engineering sciences and multidisciplinary knowledge to the solution of computer science engineering problems.

PSO2: Apply research-based knowledge, appropriate techniques, IT tools to complex computer science engineering problems including design, analysis, interpretation of data, and synthesis of the information to provide valid conclusions.

PSO3: Apply ethical principles engineering profession and recognize the need of independent and lifelong learning for professional development and personnel growth.

CSPC-413PDAALab							
Teaching Scheme			Credit	Marks Distribution			Duration of End Semester Examination
L	T	P	C	Internal Assessment	End Semester Examination	Total	
0	0	2	1	MaximumMarks:30	MaximumMarks:20	S0	2Hours
				MinimumMarks:12	MinimumMarks:08	20	

Following is the list of experiments out of which minimum 08 experiments must be performed in the lab. The additional experiments may be performed by the respective in situation depending on the in restructure available.

List of experiments:

1. Write a program to perform Insetion sort for any given list of numbers.
2. Write a program to perform Quick Sort for the given list of integer values.
3. Write a program to find Maximum and Minimum of the given set of integer values.
4. Write a Program to perform Merge Sort on the given two lists of integer values.
5. Write a Program to perform Binary Search for a given set integer values recursively and non-recursively.
6. Write a program to find solution for knapsack problem using greedy method.
7. Write a program to find minimum cost spanning tree using Prim's Algorithm.
8. Write a program to find minimum cost spanning tree using Kruskal's Algo1ithm.
9. Write a program to perform Single source shortest path problem for a given graph.
10. Write a program to find solution for job sequencing with deadlines problem.
11. Write a program for all pairs shortest path problem.
12. Write a program to solve N-QUEENS problem.

<u>**GENERAL GUIDELINES AND SAFETY INSTRUCTIONS**</u>

1. You may use the computers in the lab only when a teacher is present.
2. Please place your bags at the front of the lab.
3. Do not eat or drink in the lab.
4. Keep the lab clean and neat at all times.
5. Use only the computer you are assigned to.
6. Report any hardware fault immediately to your teacher. Never attempt to dismantle the different parts of the computer.
7. Each student must log in to his/her account. No sharing of accounts is permitted.
8. The computers are for your academic use. Playing computer games for entertainment is strictly not allowed.
9. Shut down the computer properly after use.
10. Do not charge your personal mobile devices in the lab.

Cleanliness

- Keep your workspace clean and free of clutter
- Don't eat or drink in the lab
- Don't litter
- Don't remove cables or items from the lab

<u>Fire safety</u>

- Have a fire extinguisher and first-aid kit available
- Follow fire safety guidelines
- Be aware of the possibility of an accidental fire
- Know how to react to a fire
- Have a planned fire escape route

<u>Eye and body safety</u>

- Avoid eye fatigue by blinking often or closing your eyes for a few minutes
- Sit straight and in a comfortable posture
- Spread your fingers apart or rotate your wrists at regular intervals
- Wear proper lab attire
- Practice good hygiene

<u>**Other safety guidelines**</u>

- Don't spill liquids on the computer
- Don't touch hot or high voltage areas of printers

- Don't open a power supply or CRT monitor
- Don't tamper with wires or network cables
- Don't use illegal software
- Don't attempt to compromise network security

Experiment No: 1

AIM: Write a program to perform Insertion sort for any given list of numbers.

SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:
```
#include <math.h>
#include <stdio.h>

void insertionSort(int arr[], int N) {

    // Starting from the second element
    for (int i = 1; i < N; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1], that are
          // greater than key, to one position to
          // the right of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }

        // Move the key to its correct position
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6 };
    int N = sizeof(arr) / sizeof(arr[0]);

    printf("Unsorted array: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    // Calling insertion sort on array arr
    insertionSort(arr, N);

    printf("Sorted array: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Out Put:

```
Unsorted array: 12 11 13 5 6
Sorted array: 5 6 11 12 13
```

Experiment No: 2

AIM: Write a program to perform Quick Sort for the given list of integer values
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
// C Program to sort an array using qsort() function in C
#include <stdio.h>
#include <stdlib.h>

// If a should be placed before b, compare function should
// return positive value, if it should be placed after b,
// it should return negative value. Returns 0 otherwise
int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int arr[] = { 4, 2, 5, 3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

                                              // Sorting arr using inbuilt quicksort method
    qsort(arr, n, sizeof(int), compare);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Out Put:

```
1 2 3 4 5
```

<u>Experiment No: 3</u>

AIM: Write a program to find Maximum and Minimum of the given set of integer values.
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
// Function to find maximum and minimum in an array
void findMinMax(int arr[], int n, int *max, int *min) {

   // Assuming first element as minimum and maximum
   *max = arr[0];
   *min = arr[0];

   for (int i = 1; i < n; i++) {

      // Update max if arr[i] is larger
      if (arr[i] > *max)
         *max = arr[i];

      // Update min if arr[i] is smaller
      if (arr[i] < *min)
         *min = arr[i];
   }
}

int main() {
   int arr[] = {5, 2, 7, 6};
   int n = sizeof(arr) / sizeof(arr[0]);
   int max, min;

    // Finding minimum and maximum values in arr
   findMinMax(arr, n, &max, &min);

   printf("%d\n", max);
   printf("%d\n", min);
   return 0;
}
```

Out Put :

```
7
2
```

Experiment No: 4

AIM: Write a Program to perform Merge Sort on the given two lists of integer values.
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
// C program for the implementation of merge sort
#include <stdio.h>
#include <stdlib.h>

// Merges two subarrays of arr[].
// First subarray is arr[left..mid]
// Second subarray is arr[mid+1..right]
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    int leftArr[n1], rightArr[n2];

    // Copy data to temporary arrays
    for (i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        rightArr[j] = arr[mid + 1 + j];

    // Merge the temporary arrays back into arr[left..right]
    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        }
        else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftArr[], if any
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }

    // Copy the remaining elements of rightArr[], if any
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

// The subarray to be sorted is in the index range [left-right]
void mergeSort(int arr[], int left, int right) {
    if (left < right) {

        // Calculate the midpoint
        int mid = left + (right - left) / 2;

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

      // Sorting arr using mergesort
    mergeSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Out Put:

```
Given array is
12 11 13 5 6 7


Sorted array is
5 6 7 11 12 13
```

Experiment No: 5

AIM: Write a Program to perform Binary Search for a given set integer values recursively and non-recursively.
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
#include <stdio.h>
#define MAX_LEN 10

/* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
{
   int l1,i,j, flag = 0;
   l1 = 0;
   i = num-1;
   while(l1 <= i)
   {
      j = (l1+i)/2;
      if( l[j] == ele)
      {
      printf("\nThe element %d is present at position %d in list\n",ele,j);
             flag =1;
             break;
      }
      else
          if(l[j] < ele)
              l1 = j+1;
          else
               i = j-1;
   }
   if( flag == 0)
   printf("\nThe element %d is not present in the list\n",ele);
}

/* Recursive function*/
int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
  int m,pos;
  if (arrayStart<=arrayEnd)
  {
    m=(arrayStart+arrayEnd)/2;
    if (l[m]==a)
      return m;
    else if (a<l[m])
      return b_search_recursive(l,arrayStart,m-1,a);
    else
      return b_search_recursive(l,m+1,arrayEnd,a);
   }
   return -1;
}

void read_list(int l[],int n)
{
   int i;
   printf("\nEnter the elements:\n");
   for(i=0;i<n;i++)
       scanf("%d",&l[i]);
}

void print_list(int l[],int n)
{
    int i;
   for(i=0;i<n;i++)
       printf("%d\t",l[i]);
}

/*main function*/
main()
{
   int l[MAX_LEN], num, ele,f,l1,a;
   int ch,pos;

   //clrscr();

   printf("======================================================");
   printf("\n\t\t\tMENU");
   printf("\n=====================================================");
```

```
    printf("\n[1] Binary Search using Recursion method");
    printf("\n[2] Binary Search using Non-Recursion method");
    printf("\n\nEnter your Choice:");
    scanf("%d",&ch);

    if(ch<=2 & ch>0)
    {
      printf("\nEnter the number of elements : ");
      scanf("%d",&num);
      read_list(l,num);
      printf("\nElements present in the list are:\n\n");
      print_list(l,num);
      printf("\n\nEnter the element you want to search:\n\n");
      scanf("%d",&ele);


    switch(ch)
    {
      case 1:printf("\nRecursive method:\n");
             pos=b_search_recursive(l,0,num,ele);
             if(pos==-1)
             {
               printf("Element is not found");
             }
             else
             {
               printf("Element is found at %d position",pos);
             }
             //getch();
             break;

      case 2:printf("\nNon-Recursive method:\n");
             b_search_nonrecursive(l,num,ele);
             //getch();
             break;
      }
    }
//getch();
}
```

Out Put:

```
[1] Binary Search using Recursion method
[2] Binary Search using Non-Recursion method

Enter your Choice:1
Enter the number of elements : 5

Enter the elements:
12
22
32
42
52

Elements present in the list are:

12      22      32      42      52

Enter the element you want to search:

42

Recursive method:
Element is found at 3 position
```

Experiment No:6

AIM: Write a program to find solution for knapsack problem using greedy method
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
#include<stdio.h>
int main()
{
     float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
     int n,i,j;
     printf("Enter the number of items :");
     scanf("%d",&n);
    for (i = 0; i < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n");
    scanf("%f",&capacity);

     for(i=0;i<n;i++)
         ratio[i]=profit[i]/weight[i];

    for (i = 0; i < n; i++)
      for (j = i + 1; j < n; j++)
         if (ratio[i] < ratio[j])
        {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;

            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;

            temp = profit[j];
            profit[j] = profit[i];
            profit[i] = temp;
         }

     printf("Knapsack problems using Greedy Algorithm:\n");
     for (i = 0; i < n; i++)
     {
      if (weight[i] > capacity)
          break;
       else
      {
          Totalvalue = Totalvalue + profit[i];
          capacity = capacity - weight[i];
       }
     }
       if (i < n)
       Totalvalue = Totalvalue + (ratio[i]*capacity);
     printf("\nThe maximum value is :%f\n",Totalvalue);
     return 0;
}
```

Out Put:

```
Enter the number of items :4
Enter Weight and Profit for item[0] :
2
12
Enter Weight and Profit for item[1] :
1
10
Enter Weight and Profit for item[2] :
3
20
Enter Weight and Profit for item[3] :
2
15
Enter the capacity of knapsack :
5
Knapsack problems using Greedy Algorithm:

The maximum value is :38.333332
```

Experiment No: 7

AIM: Write a program to find minimum cost spanning tree using Prim's Algorithm
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:
```
// A C++ program for Prim's Minimum
// Spanning Tree (MST) algorithm. The program is
// for adjacency matrix representation of the graph

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

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(vector<int> &key, vector<bool> &mstSet) {

    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(vector<int> &parent, vector<vector<int>> &graph) {
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t"
             << graph[parent[i]][i] << " \n";
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
```

```
void primMST(vector<vector<int>> &graph) {

    // Array to store constructed MST
    vector<int> parent(V);

    // Key values used to pick minimum weight edge in cut
    vector<int> key(V);

    // To represent set of vertices included in MST
    vector<bool> mstSet(V);

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first
    // vertex.
    key[0] = 0;

    // First node is always root of MST
    parent[0] = -1;

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {

        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent
            // vertices of m mstSet[v] is false for vertices
            // not yet included in MST Update the key only
            // if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // Print the constructed MST
    printMST(parent, graph);
}

// Driver's code
int main() {
      vector<vector<int>> graph = { { 0, 2, 0, 6, 0 },
                                    { 2, 0, 3, 8, 5 },
                                    { 0, 3, 0, 0, 7 },
                                    { 6, 8, 0, 0, 9 },
                                    { 0, 5, 7, 9, 0 } };

    // Print the solution
    primMST(graph);

    return 0;
}
```

Out Put :

```
Edge     Weight
0 - 1     2
1 - 2     3
0 - 3     6
1 - 4     5
```

Experiment No: 8

AIM: Write a program to find minimum cost spanning tree using Kruskal's Algorithm
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
#include <stdio.h>
#include <stdlib.h>
const int inf = 999999;
int k, a, b, u, v, n, ne = 1;
int mincost = 0;
int cost[3][3] = {{0, 10, 20},{12, 0,15},{16, 18, 0}};
int  p[9] = {0};
int applyfind(int i)
{
   while(p[i] != 0)
      i=p[i];
   return i;
}
int applyunion(int i,int j)
{
   if(i!=j) {
      p[j]=i;
      return 1;
   }
   return 0;
}
int main()
{
   n = 3;
   int i, j;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         if (cost[i][j] == 0) {
            cost[i][j] = inf;
         }
      }
   }
   printf("Minimum Cost Spanning Tree: \n");
   while(ne < n) {
      int min_val = inf;
      for(i=0; i<n; i++) {
         for(j=0; j <n; j++) {
                if(cost[i][j] < min_val) {
                    min_val = cost[i][j];
                    a = u = i;
                    b = v = j;
                }
            }
        }
        u = applyfind(u);
        v = applyfind(v);
        if(applyunion(u, v) != 0) {
            printf("%d -> %d\n", a, b);
            mincost +=min_val;
        }
        cost[a][b] = cost[b][a] = 999;
        ne++;
    }
    printf("Minimum cost = %d",mincost);
    return 0;
}
```

Out Put :

```
Minimum Cost Spanning Tree:
0 -> 1
1 -> 2
Minimum cost = 25
```

Experiment No: 9

AIM: Write a program to perform Single source shortest path problem for a given

SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
// C++ program for Dijkstra's single source shortest path
// algorithm. The program is for adjacency matrix
// representation of the graph
#include <iostream>
using namespace std;
#include <limits.h>

// Number of vertices in the graph
#define V 9

// A utility function to 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;

    return min_index;
}

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

// 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

    // Initialize all distances as INFINITE and stpSet[] 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 = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[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 sptSet,
            // 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's code
int main()
{

    /* Let us create the example graph discussed above */
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 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 } };

    // Function call
    dijkstra(graph, 0);

    return 0;
}

// This code is contributed by shivanisinghss2110
```

Out Put:

```
Vertex      Distance from Source
0                 0
1                 4
2                 12
3                 19
4                 21
5                 11
```

6	9
7	8
8	14

Experiment No: 10

AIM: Write a program to find solution for job sequencing with deadlines problem.

SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
 #include <algorithm> // For sort function
#include <iostream>  // For input-output operations
using namespace std;

// Structure to represent a job with its id, deadline, and profit
struct Job {
   char id;    // Job Id
   int dead;   // Deadline of the job
   int profit; // Profit if the job is done before or on the deadline
};

// Comparison function to sort jobs based on descending order of profit
bool comparison(Job a, Job b) {
   return (a.profit > b.profit); // Return true if profit of job 'a' is greater than that of 'b'
}

// Function to schedule jobs for maximum profit
void printJobScheduling(Job arr[], int n) {
   // Step 1: Sort all jobs in descending order of profit
   sort(arr, arr + n, comparison); // Sort jobs using the comparison function

   // Step 2: Create an array to keep track of free time slots
   bool slot[n] = {false};  // Initially, all time slots are free (false)

   // Step 3: Create an array to store result of the job sequence
   char result[n] = {0};    // Result array will store the job ids scheduled

   // Step 4: Iterate through all given jobs
   for (int i = 0; i < n; i++) {
      // Check if the job can be scheduled before or on its deadline
      // We start checking from the last possible free slot before the deadline
      if(slot[arr[i].dead] == false) {  // If the slot for this job's deadline is free
         result[i] = arr[i].id;  // Assign the job id to the result
         slot[arr[i].dead] = true; // Mark this slot as filled
      }
```

```
    }

    // Step 5: Output the scheduled jobs (only those in valid slots)
    for (int i = 0; i < n; i++) {
        if (slot[arr[i].dead]) {   // If the slot was filled
            if(result[i] != 0)     // Check if the result has a valid job id
                cout << result[i] << " ";  // Print the job id
        }
    }
    cout << endl;
}

int main() {
    // Array of jobs with id, deadline, and profit
    Job arr[] = { {'a', 2, 100},
                  {'b', 1, 19},
                  {'c', 2, 27},
                  {'d', 1, 25},
                  {'e', 3, 15} };

    // Number of jobs
    int n = sizeof(arr) / sizeof(arr[0]);

    // Print the result of the job scheduling
    cout << "Following is the maximum profit sequence of jobs:\n";
    printJobScheduling(arr, n);

    return 0;
}
// Code is contributed by Pratham Lashkari
```

Out Put:

```
Following is the maximum profit sequence of jobs:
a d e
```

Experiment No: 11

AIM: Write a program for all pairs shortest path problem
SOFTWARE/APPARATUSREQUIRED:- Turbo C**,** Personal Computer

Program:

```
#include<iostream>
#include<iomanip>
#define NODE 7
#define INF 999
using namespace std;
//Cost matrix of the graph
int costMat[NODE][NODE] = {
   {0, 3, 6, INF, INF, INF, INF},
   {3, 0, 2, 1, INF, INF, INF},
   {6, 2, 0, 1, 4, 2, INF},
   {INF, 1, 1, 0, 2, INF, 4},
   {INF, INF, 4, 2, 0, 2, 1},
   {INF, INF, 2, INF, 2, 0, 1},
   {INF, INF, INF, 4, 1, 1, 0}
};
void floydWarshal(){
   int cost[NODE][NODE]; //defind to store shortest distance from any node to
any node
   for(int i = 0; i<NODE; i++)
      for(int j = 0; j<NODE; j++)
         cost[i][j] = costMat[i][j]; //copy costMatrix to new matrix
         for(int k = 0; k<NODE; k++){
            for(int i = 0; i<NODE; i++)
               for(int j = 0; j<NODE; j++)
                  if(cost[i][k]+cost[k][j] < cost[i][j])
                     cost[i][j] = cost[i][k]+cost[k][j];
   }
   cout << "The matrix:" << endl;
   for(int i = 0; i<NODE; i++){
      for(int j = 0; j<NODE; j++)
         cout << setw(3) << cost[i][j];
      cout << endl;
   }
}
int main(){
   floydWarshal();
}
```

Out Put:

```
The matrix:
0 3 5 4 6 7 7
3 0 2 1 3 4 4
5 2 0 1 3 2 3
4 1 1 0 2 3 3
6 3 3 2 0 2 1
7 4 2 3 2 0 1
7 4 3 3 1 1 0
```

Basic Questions

1. What is **Algorithm Analysis**, and why is it important?
2. What are the characteristics of a good algorithm?
3. What is **Asymptotic Notation**? Explain **Big O, Big Theta (Θ), and Big Omega (Ω)**.
4. What are **Time Complexity** and **Space Complexity**?
5. What is the difference between **Worst-case, Best-case, and Average-case** complexities?

Divide and Conquer

6. Explain the **Divide and Conquer** approach with examples.
7. How does **Merge Sort** work? What is its time complexity?
8. Explain the working of **Quick Sort**. Why is it preferred over Merge Sort in some cases?
9. What is the worst-case complexity of Quick Sort? How can it be avoided?
10. How does **Binary Search** work? What is its time complexity?

Greedy Algorithms

11. What is a **Greedy Algorithm**? Give an example.
12. Explain **Huffman Coding** and its application.
13. How does **Kruskal's Algorithm** work for finding the Minimum Spanning Tree (MST)?
14. Explain **Prim's Algorithm** and compare it with Kruskal's Algorithm.
15. What is **Dijkstra's Algorithm**, and where is it used?

Dynamic Programming

16. What is **Dynamic Programming (DP)**? How is it different from Divide and Conquer?
17. Explain the **0/1 Knapsack Problem** using DP.
18. What is **Floyd-Warshall Algorithm**, and what problem does it solve?
19. How does the **Longest Common Subsequence (LCS)** problem work?
20. What is **Memoization**, and how does it improve efficiency?

Backtracking and Branch & Bound

21. What is **Backtracking**? Give an example.
22. Explain the **N-Queens Problem** and how it is solved using Backtracking.
23. What is **Branch and Bound**, and how does it differ from Backtracking?
24. How is **Travelling Salesman Problem (TSP)** solved using Branch and Bound?
25. What is the difference between **Backtracking and Dynamic Programming**?

Graph Algorithms

26. What are different ways to represent a **Graph**?
27. What is **Topological Sorting**, and where is it used?
28. How does **Bellman-Ford Algorithm** work, and how is it different from Dijkstra's Algorithm?
29. What is the **Strongly Connected Component (SCC)** in a graph?
30. Explain **Floyd-Warshall Algorithm** for finding shortest paths.

NP-Completeness and Computational Complexity

31. What is **P, NP, NP-Hard, and NP-Complete**?
32. Explain why **Travelling Salesman Problem (TSP)** is NP-Complete.
33. What is **Cook's Theorem**, and why is it important?
34. Can NP problems be solved in polynomial time?
35. What is an **Approximation Algorithm**, and why is it used?

Laboratory Experiment Evaluation Rubric

Category	**Outstanding (Up to 100%)**	**Accomplished (Up to 75%)**	**Developing (Up to 50%)**	**Beginner (Up to 25%)**
Written/Presentation/Demonstration	The write-up is clear, well-organized, and follows the prescribed format. All required sections (aim, apparatus, theory, procedure, diagram, etc.) are present and well-written. Demonstration is clear and thorough.	The report follows the specified format, but some sections (like the diagram or theory) are missing or incomplete. The demonstration is understandable but lacks depth.	The report includes most sections but lacks clarity, coherence, or completeness in some parts (e.g., diagram missing, unclear theoretical explanation). The demonstration is incomplete or unclear.	The report is poorly written and organized. Many sections are missing or incorrect (e.g., no diagram, incomplete procedure). The demonstration lacks clarity or is missing.
Viva-Voice	Demonstrates a deep understanding of the experiment, underlying principles, and outcomes. Answers questions confidently and accurately.	Demonstrates a general understanding of the experiment and principles but struggles with some aspects. Provides correct answers to most questions.	Struggles with some fundamental concepts and principles. Answering questions requires additional prompts, with a few errors in understanding.	Lacks a basic understanding of the experiment. Unable to answer most questions accurately. Demonstrates significant gaps in knowledge.

Category	Outstanding (Up to 100%)	Accomplished (Up to 75%)	Developing (Up to 50%)	Beginner (Up to 25%)
Performance/Report/File Work	Performs the experiment accurately and efficiently. The report is thorough, with correct observations, calculations, and analysis. Data is recorded neatly and with appropriate units. All relevant calculations and interpretations are included.	Performs the experiment well with minor errors or delays. The report is complete but may contain some inaccuracies or missing components in calculations or observations.	Completes the experiment but with notable mistakes, either in the setup or the data. The report has several missing or inaccurate components, including incorrect or incomplete calculations.	Struggles to perform the experiment correctly. Significant errors in setup, data collection, and analysis. The report is poorly structured with major inaccuracies or missing sections.
Attendance	Consistently attends all lab sessions, actively participates, and engages with the experiment and group discussions.	Attends most lab sessions with occasional absences. Participation is generally good but lacks consistency or depth.	Attends some lab sessions but has frequent absences or minimal participation.	Misses several lab sessions and shows minimal to no participation in class or group activities.