Department of Computer Science & Engineering
RCC Institute of Information Technology, Kolkata
Canal South Road, Kolkata - 700015

Paper Name: Design and Analysis of Algorithms Lab **Paper Code: PCC-CS 494**
Semester: 4th B.Tech (CSE) Sec A & B **Academic Session: 2022-23**

Lab Manual for listed Experiments List

Sl No.	Day	Experiment Number	Experiment Name	COs
1).	**Day - 1**	Experiment -1	**Write a C program to implement Quick Sort algorithm by using Divide & Conquer technique as follows:** a) Define main () to store n number of integers in an array. b) Define a function Partition () for partitioning the array/sub-array by using pivot element. c) Define a recursive quicksort () to sort the given integers. d) Define a function to display the sorted integers.	**CO1**
2).		Experiment - 2	**Write a C program to implement Merge Sort algorithm by using Divide & Conquer technique as follows:** a) Define main () to store n number of integers in an array. b) Define a function Merge () for merging the sub- arrays. c) Define a recursive mergesort () to sort the given integers. d) Define a function to display the sorted integers.	**CO1**
3).	**Day - 2**	Experiment - 3	**Write a C program to implement the chain matrix multiplication algorithm by using Dynamic Programming as follows:** a) Define main () to read n (>1) number of matrices and their dimensions as integer. b) Define a function to generate the cost matrix for chain of matrices. c) Define a function to find the minimum number of scalar multiplication for chain of matrix. d) Define a function to print optimal Matrix Multiplication Sequence.	**CO2**
4).	**Day - 3**	Experiment - 4	**Write a C program to implement Traveling Salesman Problem by using Dynamic Programming as follows:** a) Define main () to read number of cities and travelling cost b) Define two functions- mincost () & least() to implement Traveling Salesman Problem using DP c) Define a function to display the minimum travelling cost and routes.	**CO2**
5).	**Day - 4**	Experiment -5	**Write a C program to implement Single source shortest Path for a graph by using Bellman Ford Algorithm (Dynamic Programming) as follows:** a) Define main() to input the number of vertices, number of edges, cost matrix and path matrix of the graph. b) Define a function for creating a graph. c) Define bellmanford() to pass the graph and source vertex. d) Define a function to display the optimal single source paths.	**CO2**
6).	**Day - 5**	Experiment -6	**Write a C program to implement the fractional Knapsack problem using following functions:** a) Define a function to read number of items, profit and weight of items and knapsack capacity. b) Define a function to sort the items based on the ratio of profit and weight c) Define a function to implement Knapsack problem using Greedy d) Define a function to display the maximum profit and the result vector.	**CO3**
7).		Experiment -7	**Write a C program to implement the job scheduling with deadline problem by using following functions:** a) Define a function to read number of jobs, profit and deadline of job. b) Define a function to sort the items based on the ratio of profit and weight c) Define a function to implement job sequencing with deadline using Greedy. d) Define a function to display the maximum profit and the job sequence	**CO3**
8).	**Day - 6**	Experiment -8	**Write a C program to implement the N-Queen problem by using Backtracking method as follows:** a) Define a function to read number of queens b) Define two functions – queen() and place() for implementation of n-queen	**CO4**

			problem using backtracking c) Define a function to display the result vectors and place of queens as a table form.	
9).		Experiment -9	**Write a C program to implement the graph coloring problem by using Backtracking method as follows:** a) Define main () to read number of vertices, edges and assign 0 and/or 1 to all index of adjacency matrix b) Define a function– next-color() to solve the graph coloring problem using backtracking c) Define a function to displaying the color of each vertex.	**CO4**
10).	**Day - 7**	Experiment -10	**Write a C program to implement the Kruskal's Algorithm for undirected graph by using following functions:** a) Define main () to input the cost matrix of a graph. b) Define a function to find Minimum Cost of the Spanning Tree of an undirected graph using Kruskal's algorithm. c) Define a function to display the input cost matrix and minimum cost.	**CO5**
11).		Experiment- 11	**Write a C program to implement the Prim's Algorithm by using following functions:** a) Define main () to input the cost matrix of a graph. b) Define a function to find Minimum Cost of the Spanning Tree of an undirected graph using Prim's algorithm. c) Define a function to display the input cost matrix and minimum cost.	
12).	**Day - 8**	Experiment- 12	**Write a C program to implement the BFS and DFS algorithm for a undirected and directed graph by using following functions:** a) Define main () to read number of vertices, graph data and starting vertex. b) Define function– bfs() and dfs() to implement the BFS and DFS algorithms c) Define a function to displaying the BFS and DFS paths.	**CO6**

Solutions of Experiments

Program

Day-1: Experiment-1

Write a C program to implement Quick Sort algorithm by using Divide & Conquer technique as follows:
a) **Define main () to store n number of integers in an array.**
b) Define a function Partition () for partitioning the array/sub-array by using pivot element.
c) Define a recursive quicksort () to sort the given integers.
d) Define a function to display the sorted integers.

Algorithm:

How to work Quick sort
QuickSort is a Divide and Conquer algorithm. The steps are:
1. Pick an element from the array, this element is called as pivot element.
2. Divide the unsorted array of elements in two arrays with values less than the pivot come in the first sub array, while all elements with values greater than the pivot come in the second sub-array (equal values can go either way). This step is called the partition operation.
3. Recursively repeat the step 2(until the sub-arrays are sorted) to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Program:

```
#include<stdio.h>

void swap (int a[], int left, int right)
{
 int temp;
 temp=a[left];
 a[left]=a[right];
 a[right]=temp;
}                               //end swap

void quicksort( int a[], int low, int high )
{
 int pivot;
                               // Termination condition!
 if ( high > low )
 {
  pivot = partition( a, low, high );
  quicksort( a, low, pivot-1 );
  quicksort( a, pivot+1, high );
 }
}                           //end quicksort

int partition( int a[], int low, int high )
{
 int left, right;
 int pivot_item;
 int pivot = left = low;
 pivot_item = a[low];
 right = high;
 while ( left < right )
 {
                               // Move left while item < pivot
  while( a[left] <= pivot_item )
```

```
   left++;
                          // Move right while item > pivot
  while( a[right] > pivot_item )
   right--;
  if ( left < right )
   swap(a,left,right);
 }
                       // right is final position for the pivot
 a[low] = a[right];
 a[right] = pivot_item;
 return right;
}                     //end partition
                     // void quicksort(int a[], int, int);
void printarray(int a[], int);

int main()
{
 int a[50], i, n;
 printf("\nEnter no. of elements: ");
 scanf("%d", &n);
 printf("\nEnter the elements:");
 for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
 printf("\nUnsorted input elements:");
 printarray(a,n);
 quicksort(a,0,n-1);
 printf("\nSorted output elements:");
 printarray(a,n);
}                   //end main

void printarray(int a[], int n)
{
 int i;
 for (i=0; i<n; i++)
  printf(" %d ", a[i]);
 printf("\n");
}//end printarray

/*
```

Input-Output:

(1)

Enter no. of elements: 7

Enter the elements:7 3 9 11 5 16 8

Unsorted input elements: 7 3 9 11 5 16 8

Sorted output elements: 3 5 7 8 9 11 16

(2)

Enter no. of elements: 12

Enter the elements:81 73 61 54 49 41 38 32 28 22 18 11

Unsorted input elements: 81 73 61 54 49 41 38 32 28 22 18 11

Sorted output elements: 11 18 22 28 32 38 41 49 54 61 73 81

Day-1: Experiment-2

Write a C program to implement Merge Sort algorithm by using Divide & Conquer technique as follows:
a) Define main () to store n number of integers in an array.
b) Define a function Merge () for merging the sub- arrays.
c) Define a recursive mergesort () to sort the given integers.
d) Define a function to display the sorted integers.

Algorithm:

Given an array of length, say n, we perform the following steps to sort the array:

Divide the array into 2 parts of lengths n/2 and n - n/2 respectively (here if n is odd, we round off the value of n/2). Let us call these arrays as left half and right half respectively.
Recursively sort the left half array and the right half array.
Merge the left half array and right half-array to get the full array sorted.

Program

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

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

 /* Copy the remaining elements of R[], if there are any */
```

```
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
 int m;
    if (l < r) {
        m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

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

/* Driver code */
int main()
{
    int arr[100], n, i;
    printf("\nEnter the number of elements:");
    scanf("%d", &n);
    printf("\nEnter the %d elements\n", n);
    for(i=0; i<n; i++)
     scanf("%d", &arr[i]);

    printf("*******Given array is ******** \n");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("\n******Sorted array is ******\n");
    printArray(arr, n);
    return 0;
}
```

Input-Output

Input (1)
Enter the number of elements:7

Enter the 7 elements
9 11 5 15 29 2 33
*******Given array is ********
9 11 5 15 29 2 33

******Sorted array is ********
2 5 9 11 15 29 33

Input (2)
Enter the number of elements:7

Enter the 7 elements
21 19 17 16 13 11 9
*******Given array is ********
21 19 17 16 13 11 9

******Sorted array is ********
9 11 13 16 17 19 21

Day 2: Experiment - 3

Write a C program to implement the chain matrix multiplication algorithm by using Dynamic Programming as follows:
a) Define main () to read n (>1) number of matrices and their dimensions as integer.
b) Define a function to generate the cost matrix for chain of matrices.
c) Define a function to find the minimum number of scalar multiplication for chain of matrix.
d) Define a function to print optimal Matrix Multiplication Sequence.

Algorithm:

Let M[i,j] represent the number of multiplications required for matrix product $A_i \times \cdots \times A_j$
For $1 \le i \le j < n$
M[i,i]=0 since no product is required; i=j
The optimal solution of $A_i \times A_j$ must break at some point, k, with $i \le k < j$
Thus, $M[i,j]=M[i,k]+M[k+1,j]+d_{i-1}d_k d_j$
We are given the sequence {4, 10, 3, 12, 20, and 7}.
The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7.
The dimensions ={p0,p1,p2,p3,p4}={4, 10, 3, 12, 20, 7}
We need to compute M [i,j], $0 \le i, j \le 5$. We know M [i, i] = 0 for all i.
Calculation of Product of 2 matrices:
1. m (1,2) = m1 x m2
= 4 x 10 x 10 x 3
= 4 x 10 x 3 = 120

2. m (2, 3) = m2 x m3
= 10 x 3 x 3 x 12
= 10 x 3 x 12 = 360

3. m (3, 4) = m3 x m4
= 3 x 12 x 12 x 20
= 3 x 12 x 20 = 720

4. m (4,5) = m4 x m5
= 12 x 20 x 20 x 7
= 12 x 20 x 7 = 1680

Calculation of Product of 3 matrices:
1. M [1, 3] = M1 M2 M3
M[1,3] = min{M[1,2] + M[3,3] + p0p2p3, M[1,1] + M[2,3] + p0p1p3}
=min{120 +0 +4*3*12, 0 +360 + 4*10*12}
= min{264, 840} = 264
2. M [2, 4] = M2 M3 M4
N[2, 4] = min{ M[2,3] + M[4,4] + p1p3p4, M[2,2]+M[3,4]+p1p2p4}
M [2, 4] =min{2760, 1320} = 1320
3. M [3, 5] = M3 M4 M5
M [3, 5] = min{M[3,4] + M[5,5] +p2p4p5, M[3,3] + M[4,5]+p2p3p5}
=min{1140, 1932} = 1140

Calculation of Product of 4 matrices:

M [1, 4] = M1 M2 M3 M4
=min{M[1,3]+M[4,4]+p0p3p4, M[1,2]+M[3,4]+p0p2p4, M[1,1]+M[2,4]+p0p1p4}
=min{264 + 0 + 4*12*20, 120+720+4*3*20, 0+1320 +4*10*20}
M [1, 4] =min{1224, 1080, 2120} = 1080

Calculation of Product of 5 matrices:
M[1,5]=min{1544, 2016, 1344, 1630} = 1344
*/

Program :

```
#include <stdio.h>
#include<limits.h>
#define INFY 999999999

long int m[20][20];
int s[20][20];
int p[20];

void print_optimal(int i,int j)
{
if (i == j)
printf(" A%d ",i);
else
   {
      printf("( ");
      print_optimal(i, s[i][j]);
      print_optimal(s[i][j] + 1, j);
      printf(" )");
   }
}

void MatrixChainMultiplication(int n)
{
long int q;
int k, i, j;
for(i=n;i>0;i--)
 {
   for(j=i;j<=n;j++)
    {
     if(i==j)
       m[i][j]=0;
     else
       {
        for(k=i;k<j;k++)
        {
         q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
         if(q<m[i][j])
          {
            m[i][j]=q;
            s[i][j]=k;
          }
         }
        }
      }
 }
}

int MatrixChainOrder(int p[], int i, int j)
{
    if(i == j)
        return 0;
    int k;
    int min = INT_MAX;
```

```
    int count;

    for (k = i; k <j; k++)
    {
        count = MatrixChainOrder(p, i, k) +
                MatrixChainOrder(p, k+1, j) +
                p[i-1]*p[k]*p[j];

        if (count < min)
            min = count;
    }

    // Return minimum count
    return min;
}

int main()
{
int k, n, i, j;
printf("Enter the no. of matrices: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
   for(j=i+1;j<=n;j++)
   {
    m[i][i]=0;
    m[i][j]=INFY;
    s[i][j]=0;
   }
printf("\nEnter the dimensions: \n");
for(k=0;k<=n;k++)
{
 printf("P%d: ",k);
 scanf("%d",&p[k]);
}

MatrixChainMultiplication(n);

printf("\n************Cost Matrix M************\n");
for(i=1;i<=n;i++)
 for(j=i;j<=n;j++)
  printf("m[%d][%d]: %ld\n",i,j,m[i][j]);
printf("\n*************************************\n");

i=1,j=n;

printf("\n************Multiplication Sequence*****\n");
print_optimal(i,j);
printf("\n*************************************\n");

printf("\nMinimum number of multiplications is : %d ",
                          MatrixChainOrder(p, 1, n));
return 0;
}
```

Input-Output:

/*
**********output**********
(1)
Enter the no. of matrix: 4

Enter the dimensions:
P0: 5
P1: 4
P2: 6
P3: 2

```
P4: 7

********Cost Matrix M ************
m[1][1]: 0
m[1][2]: 120
m[1][3]: 88
m[1][4]: 158
m[2][2]: 0
m[2][3]: 48
m[2][4]: 104
m[3][3]: 0
m[3][4]: 84
m[4][4]: 0

************************************

*******Multiplication Sequence*******
( (  A1 (  A2  A3  ) ) A4  )
**************************************

Minimum number of multiplications is : 158
(2)
Enter the no. of matrices: 5

Enter the dimensions:
P0: 4
P1: 10
P2: 3
P3: 12
P4: 20
P5: 7

************Cost Matrix M**************
m[1][1]: 0
m[1][2]: 120
m[1][3]: 264
m[1][4]: 1080
m[1][5]: 1344
m[2][2]: 0
m[2][3]: 360
m[2][4]: 1320
m[2][5]: 1350
m[3][3]: 0
m[3][4]: 720
m[3][5]: 1140
m[4][4]: 0
m[4][5]: 1680
m[5][5]: 0

****************************************

************Multiplication Sequence*****
( (  A1  A2  )( (  A3  A4  ) A5  ) )
****************************************

Minimum number of multiplications is : 1344
```

Day-3: Experiment - 4

Write a C program to implement Traveling Salesman Problem by using Dynamic Programming as follows:
a) Define main () to read number of cities and travelling cost
b) Define two functions- mincost () & least () to implement Traveling Salesman Problem using DP
c) Define a function to display the minimum travelling cost and routes.

Program:

```
#include<stdio.h>
#include<limits.h>
int c_matrix[25][25], v_cities[20], no_city, cost=0;

void display_cost()
{
      printf("\n\nMinimu cost=", cost);
}
int least_tsp(int c)
{
      int i, nearest_city, nd;
      int minimum = INT_MAX, temp;

      for(i=0; i<no_city; i++)
      {
            if((!v_cities[i])&& (c_matrix[c][i]!=0) && (c_matrix[c][i]<minimum))
             {
                  minimum =c_matrix[c][i];
                  temp=c_matrix[c][i];
                  nearest_city=i;
             }
      }
      if(minimum!=INT_MAX)
       cost = cost +temp;

       return nearest_city;
}

 void tsp_mincost(int city)
 {
      int nearest_city;
       v_cities[city]=1;
       printf("%d --->", city+1);

         nearest_city=least_tsp(city);
         if(nearest_city==INT_MAX)
         {
            int v=0;
             cost = cost + c_matrix[nearest_city][v];
             return;
            }
            tsp_mincost(nearest_city);
 }


int main()
{
      int i, j;
      printf("\nEnter total no. of cities:");
      scanf("%d", &no_city);

      printf("\nEnter cost matrix:\n");
      for(i=0;i<no_city; i++ )
       {
            printf("\nEnter %d elements in row [%d]\n", no_city, i+1);
            for(j=0;j<no_city; j++)
            {
                  scanf("%d",&c_matrix[i][j]);
             }
       }
       for(i=0; i<no_city; i++)
          v_cities[i]=0;

      printf("\nTSP Path::");

       tsp_mincost(0);
        display_cost();
}
```

Day-4: Experiment - 5

Write a C program to implement Single source shortest Path for a graph by using Bellman Ford Algorithm (Dynamic Programming) as follows:
a) Define main() to input the number of vertices, number of edges, cost matrix and path matrix of the graph.
b) Define a function for creating a graph.
c) Define bellmanford() to pass the graph and source vertex.
d) Define a function to display the optimal single source paths

Program:

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

struct Edge
{
    int source;
    int destination;
    int weight;
};
struct Graph
{
    int V;
    int E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph *g;

    g = (struct Graph*) malloc( sizeof(struct Graph));
    g->V = V;
    g->E = E;
    g->edge = (struct Edge*) malloc( g->E * sizeof( struct Edge ) );
    return g;
}

void displaySolution(int dist[], int n, int flag)
{
      int i;
      if(flag)
      {
      printf("\nVertex\tDistance from Source Vertex\n");
      for (i = 0; i < n; ++i)
            printf("%d \t\t %d\n", i, dist[i]);
        printf("This graph contains no negative edge cycle\n");
      }
      else
       printf("This graph contains negative edge cycle\n");
}

void BellmanFord(struct Graph *g, int source)
{
      int i,j, u, v, w, flag=1;

    int V = g->V;
```

```
    int E = g->E;

    int StoreDistance[V];

    for (i = 0; i < V; i++)
        StoreDistance[i] = INT_MAX;

    StoreDistance[source] = 0;

    for (i = 1; i <= V-1; i++)
    {
        for (j = 0; j < E; j++)
        {
            u = g->edge[j].source;

            v = g->edge[j].destination;

            w = g->edge[j].weight;

            if (StoreDistance[u] + w < StoreDistance[v])
                StoreDistance[v] = StoreDistance[u] + w;
        }
    }

    for (i = 0; i < E; i++)
    {
         u = g->edge[i].source;

         v = g->edge[i].destination;

         w = g->edge[i].weight;

        if (StoreDistance[u] + w < StoreDistance[v])
            flag=0;
    }

    displaySolution(StoreDistance, V, flag);
}

int main()
{
    int V,E,S, i;

      printf("Enter number of vertices in a graph\n");
    scanf("%d",&V);

      printf("Enter number of edges in a graph\n");
    scanf("%d",&E);

      printf("Enter your source vertex number\n");
      scanf("%d",&S);

    struct Graph *gp = createGraph(V, E);

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

```
      {
        printf("\nEnter edge %d properties Source, destination, weight
respectively\n",i+1);
        scanf("%d",&gp->edge[i].source);
        scanf("%d",&gp->edge[i].destination);
        scanf("%d",&gp->edge[i].weight);
    }
    BellmanFord(gp, S);


    return 0;
}

/* Output (case 1)

Enter number of vertices in a graph
4
Enter number of edges in a graph
4
Enter your source vertex number
0

Enter edge 1 properties Source, destination, weight respectively
0 1 4

Enter edge 2 properties Source, destination, weight respectively
0 3 5

Enter edge 3 properties Source, destination, weight respectively
3 2 3

Enter edge 4 properties Source, destination, weight respectively
2 1 -10

Vertex  Distance from Source Vertex
0               0
1               -2
2               8
3               5
This graph contains no negative edge cycle
----------------------------------------
Output (case 2):

Enter number of vertices in a graph
4
Enter number of edges in a graph
5
Enter your source vertex number
0

Enter edge 1 properties Source, destination, weight respectively
0 1 4

Enter edge 2 properties Source, destination, weight respectively
0 3 5
```

```
Enter edge 3 properties Source, destination, weight respectively
3 2 3

Enter edge 4 properties Source, destination, weight respectively
2 1 -10

Enter edge 5 properties Source, destination, weight respectively
1 3 5
This graph contains negative edge cycle
*/
```

Day-5: Experiment-6

Write a C program to implement the fractional Knapsack problem using following functions:
a) Define a function to read number of items, profit and weight of items and knapsack capacity.
b) Define a function to sort the items based on the ratio of profit and weight
c) Define a function to implement Knapsack problem using Greedy
d) Define a function to display the maximum profit and the result vector.

Program:

```
# include<stdio.h>

float t_profit=0, x[50];

void display( int n)
{
 int i;
   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);

   printf("\nMaximum profit is:- %f", t_profit);

}

void sort_Descending(int n, float weight[], float profit[], float ratio[])
{
 int i, j, temp;
 //sort
   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;
        }
       }
     }
}

void Greedy_fract_knapsack(int n, float weight[], float profit[], float capacity)
 {
   int i, j, u;
   float ratio[20];
   u = capacity;
```

```
   for (i = 0; i < n; i++) // solution vector initialization
      x[i] = 0.0;

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

   sort_Descending(n, weight, profit, ratio);

   for (i = 0; i < n; i++)
   {
      if (weight[i] > u)
         break;
      else
        {
         x[i] = 1.0;
         t_profit = t_profit + profit[i];
         u = u - weight[i];
      }
   }

   if (i < n)
    {
      x[i] = u / weight[i];
      t_profit = t_profit + (x[i] * profit[i]);
    }
   display(n);
}//end Greedy_fract_knapsack()

int main()
 {
   float weight[20], profit[20], capacity;
   int num, i;

   printf("\nEnter the no. of items:- ");
   scanf("%d", &num);

   printf("\nEnter the weights and profits of each item:- ");
   for (i = 0; i < num; i++)
      scanf("%f %f", &weight[i], &profit[i]);

   printf("\nEnter the capacity of knapsack:- ");
   scanf("%f", &capacity);

   Greedy_fract_knapsack(num, weight, profit, capacity);
  return(0);
}//end main
```

Day-5: Experiment-7

Write a C program to implement the job scheduling with deadline problem by using following functions:
a) Define a function to read number of jobs, profit and deadline of job.
b) Define a function to sort the items based on the ratio of profit and weight
c) Define a function to implement job sequencing with deadline using Greedy.
d) Define a function to display the maximum profit and the job sequence

Program:

```
#include <stdio.h>

#define MAX 100

typedef struct Job {
      char id[5];
      int deadline;
      int profit;
} Job;

void displayJobSequence(Job jobs[], int dmax, int timeslot[], int maxprofit)
{
  int i, j;
      //required jobs
      printf("\nRequired Jobs: ");
```

```
      for(i = 1; i <= dmax; i++)
    {
          printf("%s", jobs[timeslot[i]].id);
          if(i < dmax)
        {
              printf(" -->");
         }
        }

      printf("\nMax Profit: %d\n", maxprofit);
}

int minValue(int x, int y)
{
      if(x < y)
         return x;
      return y;
}

 void sort( Job jobs[], int n )
 {
    int i, j;
    Job temp;

      //sort the jobs profit wise in descending order
      for(i = 1; i < n; i++) {
            for(j = 0; j < n - i; j++) {
                  if(jobs[j+1].profit > jobs[j].profit) {
                        temp = jobs[j+1];
                        jobs[j+1] = jobs[j];
                        jobs[j] = temp;
                  }
            }
      }
    printf("Sorted jobs\n");
      printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
      for(i = 0; i < n; i++)
       {
            printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
     }
}


void jobSequencingWithDeadline(Job jobs[], int n)
{
      //variables
      int i, j, k, maxprofit=0;

      //free time slots
      int timeslot[MAX];

      //filled time slots
      int filledTimeSlot = 0;

      //find max deadline value
      int dmax = 0;

        sort(jobs, n); //sort the jobs profit wise in descending order

      for(i = 0; i < n; i++) // find the maximum deadline
    {
          if(jobs[i].deadline > dmax)
       {
              dmax = jobs[i].deadline;
          }
      }

      //free time slots initially set to -1 [-1 denotes EMPTY]
      for(i = 1; i <= dmax; i++)
    {
          timeslot[i] = -1;
       }
```

```
      printf("deadline max: %d\n", dmax);

      for(i = 1; i <= n; i++)
    {
          k = minValue(dmax, jobs[i - 1].deadline);
          while(k >= 1)
      {
            if(timeslot[k] == -1)
            {
                  timeslot[k] = i-1;
                  filledTimeSlot++;
                  break;
            }
            k--;
        }//endwhile

             //if all time slots are filled then stop
      if(filledTimeSlot == dmax)
    {
       break;
       }
      }//endfor
       //required profit
      for(i = 1; i <= dmax; i++)
    {
          maxprofit += jobs[timeslot[i]].profit;
      }
      displayJobSequence(jobs, dmax, timeslot, maxprofit);
}


int main(void)
 {
      //variables
      int i, j, n;
      Job jobs[MAX];

        printf("\n Enter the number of jobs\n");
        scanf("%d", &n);

      printf("\nEnter the job id with deadline and profit\n");
        for(i=0; i<n; i++)
        {
         scanf("%s %d %d", jobs[i].id, &jobs[i].deadline , &jobs[i].profit);
      }

        jobSequencingWithDeadline(jobs, n);

      return 0;
 }
```

Day-6: Experiment-8

Write a C program to implement the N-Queen problem by using Backtracking method as follows:
a) Define a function to read number of queens
b) Define two functions – queen() and place() for implementation of n-queen problem using backtracking
c) Define a function to display the result vectors and place of queens as a table form.

Algorithm:

N-Queens Problem
---
1. N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.
2. It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3.

So first we will consider the 4 queens problem and then generate it to n - queens problem.

Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.
---
1 2 3 4
1

2

3

4

Since, we have to place 4 queens such as q1 q2 q3 and q4 on the chessboard, such that no two queens attack each other.
In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."

Now, we place queen q1 in the very first acceptable position (1, 1).
Next, we put queen q2 so that both these queens do not attack each other.
We find that if we place q2 in column 1 and 2, then the dead end is encountered.

Thus the first acceptable position for q2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q3' safely.

So we backtrack one step and place the queen 'q2' in (2, 4), the next best possible solution.
Then we obtain the position for placing 'q3' which is (3, 2).
But later this position also leads to a dead end, and no place is found where 'q4' can be placed safely.

Then we have to backtrack till 'q1' and place it to (1, 2) and
then all other queens are placed safely by moving q2 to (2, 4), q3 to (3, 1) and q4 to (4, 3).

That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem.
For another possible solution, the whole method is repeated for all partial solutions.

The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

	1	2	3	4
1	-	-	q1	-
2	q2	-	-	-
3				q3
4		q4		

Algorithm:Queen(row, n) // to check for proper positioning of queen, initially row =1 and n is no. of queens
---

Steps:

```
1. for column=1 to n Do
        if(Place(row,column)) Then
                c_board[row]=column      //no conflicts so place queen
                if(row==n) Then          //dead end
                        Display(n);    //printing the board configuration
                else                    //try queen with next position
                        Queen(row+1,n);
2. return
```

Place(row,column) //to check conflicts , If no conflict for desired postion returns 1 otherwise returns 0

Steps:

```
1. for i=1 to row-1 Do
        if (c_board[i]==column) Then //checking column and digonal conflicts
                return 0
        else
                if(abs(c_board[i]-column)==abs(i-row)) Then
    return 0
2. return 1; //no conflicts
```

Display(n) //function for printing the solution

Steps:

```
1. fori=1 to n Do
        print i
        for j=1 to n Do //for nxn board
                if(c_board[i]==j) Then
                        print " Q"     //queen at i,j position
                else
                        print "   -" //empty slot
```

2. return

Program:

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

int c_board[20],count;

//function for printing the solution
void display(int n)
{
      int i,j;
      printf("\n\nSolution %d:\n\n",++count);

      for(i=1;i<=n;++i)
            printf("\t%d",i);

      for(i=1;i<=n;++i)
      {
            printf("\n\n%d",i);
            for(j=1;j<=n;++j) //for n x n board
             {
                  if(c_board[i]==j)
                        printf("\tQ"); //queen at i,j position
                  else
                        printf("\t-"); //empty slot
              }
      }
}

/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/

int place(int row,int column)
{
      int i;
      for(i=1;i<=row-1;++i)
      {
      //checking column and digonal conflicts
            if(c_board[i]==column)
                  return 0;
            else
                  if(abs(c_board[i]-column)==abs(i-row))
      return 0;
   }
   return 1; //no conflicts
}

//function to check for proper positioning of queen

void queen(int row,int n)
{
      int column;

      for(column=1;column<=n;++column)
```

```
	{
		if(place(row,column))
		{
			c_board[row]=column;   //no conflicts so place queen
			if(row==n)            //dead end
				display(n);    //printing the board configuration
			else                    //try queen with next position
				queen(row+1,n);
		}
	}
}


int main()
{
	int n,i,j;

	printf("N-Queens Problem Using Backtracking:");
	printf("\n\nEnter number of Queens:");
	scanf("%d",&n);

	queen(1,n); // 1 is first row and n is no. of queens

	return 0;
}
```

Input-Output:

N-Queens Problem Using Backtracking:

Enter number of Queens:4

Solution 1:

	1	2	3	4
1	-	Q	-	-
2	-	-	-	Q
3	Q	-	-	-
4	-	-	Q	-

Solution 2:

	1	2	3	4
1	-	-	Q	-
2	Q	-	-	-
3	-	-	-	Q
4	-	Q	-	-

Day-6: Experiment-9

Write a C program to implement the graph coloring problem by using Backtracking method as follows:
a) Define main () to read number of vertices, edges and assign 0 and/or 1 to all index of adjacency matrix
b) Define a function– next-color() to solve the graph coloring problem using backtracking
c) Define a function to displaying the color of each vertex.

Algorithm:

Graph-Coloring : In this problem, for any given graph G
we will have to color each of the vertices in G in such a way that
no two adjacent vertices get the same color and the least number of colors are used.

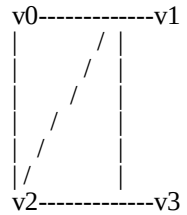
In this problem, an undirected graph is given.
There is also provided m colors. The problem is to find if it is possible to assign nodes with m different colors,
such that no two adjacent vertices of the graph are of the same colors.
If the solution exists, then display which color is assigned on which vertex.

Starting from vertex 0, we will try to assign colors one by one to different nodes.
But before assigning, we have to check whether the color is safe or not.
A color is not safe whether adjacent vertices are containing the same color.

How to solve the problem?????

First take input number of vertices and edges in graph G.
Then input all the indexes of adjacency matrix of G whose value is 1.
Now we will try to color each of the vertex.
A next_color(k) function takes in index of the kth vertex which is to be colored.
First we assign color1 to the kth vertex.Then we check whether it is connected to any of previous (k-1) vertices using backtracking.
If connected then assign a color x[i]+1 where x[i] is the color of ith vertex that is connected with kth vertex.

Enter no. of vertices : 4



Graph G

Input:
The adjacency matrix of a graph G(V, E) and an integer m,
which indicates the maximum number of colors that can be used.
Adjacency matrix:

	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

Enter no. of edges : 5

Enter indexes where value is 1-->
1 0
2 0
2 1
3 1
3 2

Let the maximum color m = 3.
This algorithm will return which node will be assigned with which color. If the solution is not possible, it will return false.
For this input the assigned colors are:
Colors of vertices -->
Vertex[1] : 1
Vertex[2] : 2
Vertex[3] : 3
Vertex[4] : 1

Algorithm:

GraphColour(G, no, eg) // G is the adjacency matrix of a graph whose initial assign 0 to all index of adjacency matrix,
//no is the total number of vertices
//and eg is the number of edges

Steps:
1. for i=0 to eg-1 //Enter indexes where value is 1
read u, v
G[u][v]=1
G[v][u]=1

2. Next_color(0, no); //coloring each vertex, 0 is starting vertex
3. return

Next_color(k, n) // check and find unique colour
Steps:
1. if k==n Then
   display(n)
   return
2. x[k]=1 //coloring vertex with color 1
3. for i=0 to k-1 Do //checking all k-1 vertices-backtracking
   if(G[i][k]!=0 && x[k]==x[i]) Then //if connected and has same color
      x[k]=x[i]+1 //assign higher color than x[i]
4. next_color(k+1, n)
5. return

Display(n) // // display n number of vertices with unique colours of adjacency vertices
Steps:
1. write "Colors of vertices:"
2. for i=0 to n-1 Do //displaying color of each vertex
   write "Vertex", i+1,"Colour", x[i]
3. return

Program:

```
#include<stdio.h>
int G[50][50], x[50];               //G:adjacency matrix,x:colors

void display(int n)
{
   int i;
   printf("Colors of vertices -->\n");
   for(i=0;i<n;i++)                 //displaying color of each vertex
     printf("Vertex[%d] : %d\n",i+1,x[i]);
}

void next_color(int k, int n)
{
   int i,j;
   if(k==n)
    {
     display(n);                        // calling the display()
     return;
    }
   x[k]=1;                          //coloring vertex with color1
   for(i=0;i<k;i++)
   {                              //checking all k-1 vertices-backtracking
     if(G[i][k]!=0 && x[k]==x[i])   //if connected and has same color
       x[k]=x[i]+1;                 //assign higher color than x[i]
   }
   next_color(k+1, n);
}


int main()
{
  int no,eg,i,j,k,l;

  printf("\nEnter no. of vertices : ");
  scanf("%d",&no);                  //total vertices

  printf("\nEnter no. of edges : ");
  scanf("%d",&eg);                  //total edges

  for(i=0;i<no;i++)
    for(j=0;j<no;j++)
      G[i][j]=0;                    //assign 0 to all index of adjacency matrix

  printf("Enter indexes where value is 1-->\n");
  for(i=0;i<eg;i++)
  {
    scanf("%d %d",&k,&l);
```

```
    G[k][l]=1;
    G[l][k]=1;
  }

    next_color(0, no);                    //coloring each vertex

  return 0;
}
```

Input Output:

Enter no. of vertices : 4

Enter no. of edges : 5
Enter indexes where value is 1-->
1 0
2 0
2 1
3 1
3 2
Colors of vertices -->
Vertex[1] : 1
Vertex[2] : 2
Vertex[3] : 3
Vertex[4] : 1

Day-7: Experiment-10

Write a C program to implement the Kruskal's Algorithm for undirected graph by using following functions:
a) Define main () to input the cost matrix of a graph.
b) Define a functions to find Minimum Cost of the Spanning Tree of an undirected graph using Kruskal's algorithm.
c) Define a function to display the input cost matrix and minimum cost.

Algorithm:

Implementation of Kruskal's algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

Input cost adjacency matrix:

 0 3 1 6 0 0
 3 0 5 0 3 0
 1 5 0 5 6 4
 6 0 5 0 0 2
 0 3 6 0 0 6
 0 0 4 2 6 0

The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

Minimum cost = 13

Program:

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

int parent[20]={0}, mincost=0;
int find(int i)
{
      while(parent[i])
            i=parent[i];
      return i;
}

int uni(int i,int j)
{
      if(i!=j)
      {
            parent[j]=i;
            return 1;
      }
      return 0;
}

void kruskalMST(int n, int mcost[20][20])
{
   int i, j, min, a, b, u, v, ne=1;

      printf("\n\nThe edges of Minimum Cost Spanning Tree are\n");
      while(ne < n)
      {
            for(i=1,min=INT_MAX;i<=n;i++)
            {
                  for(j=1;j <= n;j++)
                  {
                        if(mcost[i][j] < min)
                        {
                              min=mcost[i][j];
                              a=u=i;
                              b=v=j;
                        }
                  }
            }
            u=find(u);
            v=find(v);
            if(uni(u,v))
            {
                  printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                  mincost +=min;
            }
            mcost[a][b]=mcost[b][a]=INT_MAX;
      }     //end while
}

void display(int n, int temp[20][20])
{
      int i, j;
      printf("\nInput cost adjacency matrix:\n");
      for(i=1;i<=n;i++)
      {
            printf("\n");
            for(j=1;j<=n;j++)
            {
                  if(temp[i][j]==INT_MAX)
                  printf("  0");
            else
                     printf("%3d",temp[i][j]);
            }
```

```
	}
	printf("\n\tMinimum cost = %d\n",mincost);
}

int main()
{
	int n, i,j, cost[20][20], temp[20][20];
	printf("\n\tImplementation of Kruskal's algorithm\n");

	printf("\nEnter the no. of vertices:");
	scanf("%d",&n);

	printf("\nEnter the cost adjacency 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]=INT_MAX;
		}
	}
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		   temp[i][j]=cost[i][j];

	kruskalMST(n, cost);

	display(n,temp);

return 0;
}
```

Input-Output:

Implementation of Kruskal's algorithm

Enter the no. of vertices:7

Enter the cost adjacency matrix:
0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 18 24 0 0

The edges of Minimum Cost Spanning Tree are
1 edge (1,6) =10
2 edge (3,4) =12
3 edge (2,7) =14
4 edge (2,3) =16
5 edge (4,5) =22
6 edge (5,6) =25

Input cost adjacency matrix:

0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24

```
10 0 0 0 25 0 0
0 14 0 18 24 0 0

 Minimum cost = 99
----------------------------------------------------
 Implementation of Kruskal's algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

Input cost adjacency matrix:

 0  3  1  6  0  0
 3  0  5  0  3  0
 1  5  0  5  6  4
 6  0  5  0  0  2
 0  3  6  0  0  6
 0  0  4  2  6  0

The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

      Minimum cost = 13
```

Day-7: Experiment-11

Write a C program to implement the Prim's Algorithm by using following functions:
a) Define main () to input the cost matrix of a graph.
b) Define a function to find Minimum Cost of the Spanning Tree of an undirected graph using Prim's algorithm.
c) Define a function to display the input cost matrix and minimum cost.

Program:

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

int visited[20]= {0},mincost=0,ne=1;

void primMST(int n, int cost[20][20])
{
 int i, j, a, b, u, v, min;

      visited[1]=1;

      printf("\n");
      while(ne<n)
      {
            for (i=1,min=INT_MAX;i<=n;i++)
               for (j=1;j<=n;j++)
                if(cost[i][j]<min)
                 if(visited[i]!=0)
                  {
                        min=cost[i][j];
```

```
                        a=u=i;
                        b=v=j;
                  }
            if(visited[u]==0 || visited[v]==0)
            {
                  printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                  mincost+=min;
                  visited[b]=1;
            }
            cost[a][b]=cost[b][a]=INT_MAX;
      }

}

void display(int n, int temp[20][20])
{
      int i, j;
      printf("\nInput cost adjacency matrix:\n");
      for(i=1;i<=n;i++)
      {
            printf("\n");
            for(j=1;j<=n;j++)
            {
                  if(temp[i][j]==INT_MAX)
                  printf("  0");
            else
                     printf("%3d",temp[i][j]);
            }
      }
      printf("\n\tMinimum cost = %d\n",mincost);
}

int main()
{
      int n,i,j, cost[20][20], temp[20][20];

      printf("\n Implemetion of the Prim's Algorithm");
      printf("\n Enter the number of nodes:");
      scanf("%d",&n);

      printf("\n Enter the adjacency 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]=INT_MAX;
        }
      }

    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
             temp[i][j]=cost[i][j];


        primMST(n, cost);

        display(n, temp);

      return 0;
}
```

Input-Output

Implementation of the Prim's Algorithm
Enter the number of nodes:5

Enter the adjacency matrix:
0 10 4 0 0
10 0 2 6 3
4 2 0 1 0
0 6 1 0 0
0 3 0 0 0

Edge 1:(1 3) cost:4
Edge 2:(3 4) cost:1
Edge 3:(3 2) cost:2
Edge 4:(2 5) cost:3
Input cost adjacency matrix:

0 10 4 0 0
10 0 2 6 3
4 2 0 1 0
0 6 1 0 0
0 3 0 0 0
Minimum cost = 10

Day-8: Experiment-12

Write a C program to implement the BFS and DFS algorithm for a undirected and directed graph by using following functions:
a) Define main () to read number of vertices, graph data and starting vertex.
b) Define a function– bfs() and dfs() to implement the BFS algorithm
c) Define a function to displaying the BFS and DFS path.

Program:

```
#include<stdio.h>

int a[20][20]={{0,1,0,0,1},{1,0,0,0,1},{0,1,0,1,0},{0,0,1,0,0},{1,1,0,0,0}};
int q[20]={0},visited[20]={0},f=0,r=-1;

void bfs(int v, int n)
{
 int i, j;
      for (i=0;i<n;i++)
        if(a[v][i]==1 && !visited[i])
         q[++r]=i;

      if(f<=r)
    {
            visited[q[f]]=1;
            bfs(q[f++], n);
      }
}

int displayBFS(int n)
{
      int i;
      printf("\n BFS sequence:\n");
      for (i=0;i<n;i++)
        if(visited[i])
            printf("%d\t",i);
        else
            printf("\n Bfs is not possible");
}
int main()
 {
```

```
      int v, n=5, i, j;
      bfs(0, n);
      displayBFS(n);

  return 0;
}
```

DFS Program:

```
#include<stdio.h>

int G[10][10],visited[10];    //n is no of vertices and graph is sorted in array
G[10][10]

void display(int v)
{
      printf("\t%d",v);
}

void DFS(int i, int n)
{
    int j;
      display(i);
    visited[i]=1;

      for(j=0;j<n;j++)
       if(!visited[j]&&G[i][j]==1)
            DFS(j,n);
}

int main()
{
    int i,j, n;
    printf("\nEnter number of vertices:");
      scanf("%d",&n);

    //read the adjecency matrix
      printf("\nEnter adjecency matrix(Transitive closer)) of the graph:\n");

      for(i=0;i<n;i++)
      for(j=0;j<n;j++)
                  scanf("%d",&G[i][j]);

    //visited is initialized to zero
      for(i=0;i<n;i++)
        visited[i]=0;

    printf("\n DFS Sequence:\n") ;
    DFS(0,n);
return 0;
}
```

Input-Output:

Enter number of vertices:8

Enter adjecency matrix of the graph:
0 1 1 1 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 0 0 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0
    0   1   5   2   7   6   3   4

References

1. **Programming with ANSI and Turbo C –** Ashok N.Kamthane
2. **Programming in ANSI C –** E. Balagurusamy
3. **Let Us C -** Yaswanth Kanethkar
4. **C & Data Structures –** Prof. P.S.Desh Pande, Prof. O.G.Kakde, Wiley Dreamtech Pvt.Ltd
5. **Data Structures Using C –** A.S.Tenenbum, PHI/Person Education.
6. **The C Programming Language –** B.W.Kernighan,Dennis M.Richie, PHI/ Person Education.
7. **Foundations of Algorithms Using C++ Pseudocode ,** by Richard Neapolitan
8. **Mastering Algorithms with C,**By Kyle Loudon
9. **Data Structures and Algorithm Analysis in C,** By Weiss
10. **Algorithms in C, By Robert Sedgewick**