DAA LAB MANUAL

Lab Manual for the Academic Year 2024-2025

BCA V Sem

Vidhyaashram First Grade College

Mysore

In-charge

EVELIN JACOB

DAA LAB MANUAL

Vidhyaashram First Grade College

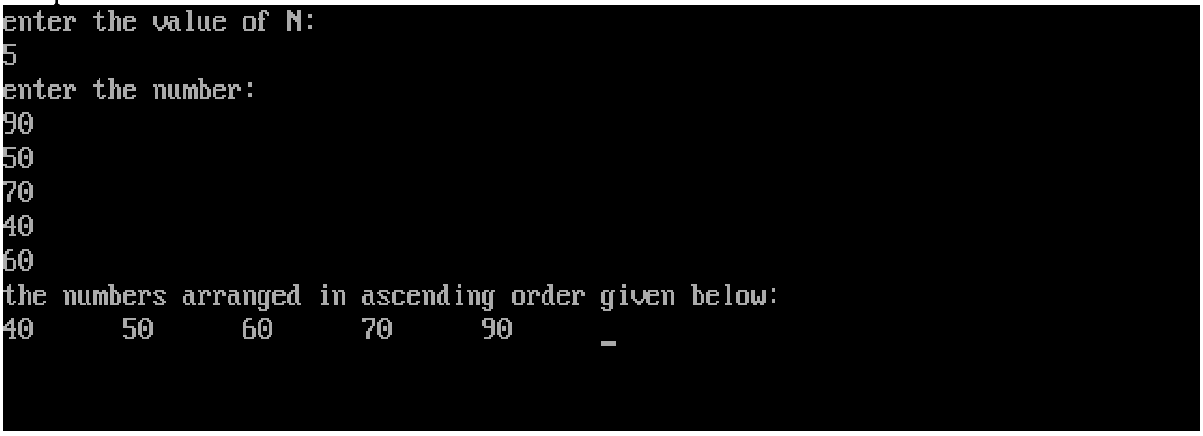
BCA (Bachelor of Computer Applications)

Course Title Design and Analysis of Algorithms Laboratory (Practical)
Practical Credits 02
Course Code DSC13-Lab
Contact Hours 4 Hours/wk
Formative Assessment 25 Marks
Summative Assessment 25 Marks

Program-1: Write a program to sort a list of n elements using selection sort technique

```
#include<stdio.h>

void main() {
  int i, j, a, n, number[20];
  clrscr();
  printf("enter the value of N:\n");
  scanf("%d", & n);
  printf("enter the number:\n");
  for (i = 0; i < n; i++)
    scanf("%d", & number[i]);
  for (i = 0; i < n; i++) {
    for (j = i + 1; j < n; j++) {
      if (number[i] > number[j]) {
        a = number[i];
        number[i] = number[j];
        number[j] = a;
      }
    }
  }
  printf("the numbers arranged in ascending order given below:\n");
  for (i = 0; i < n; i++)
    printf("%d\t", number[i]);
  getch();
}
```

Output:



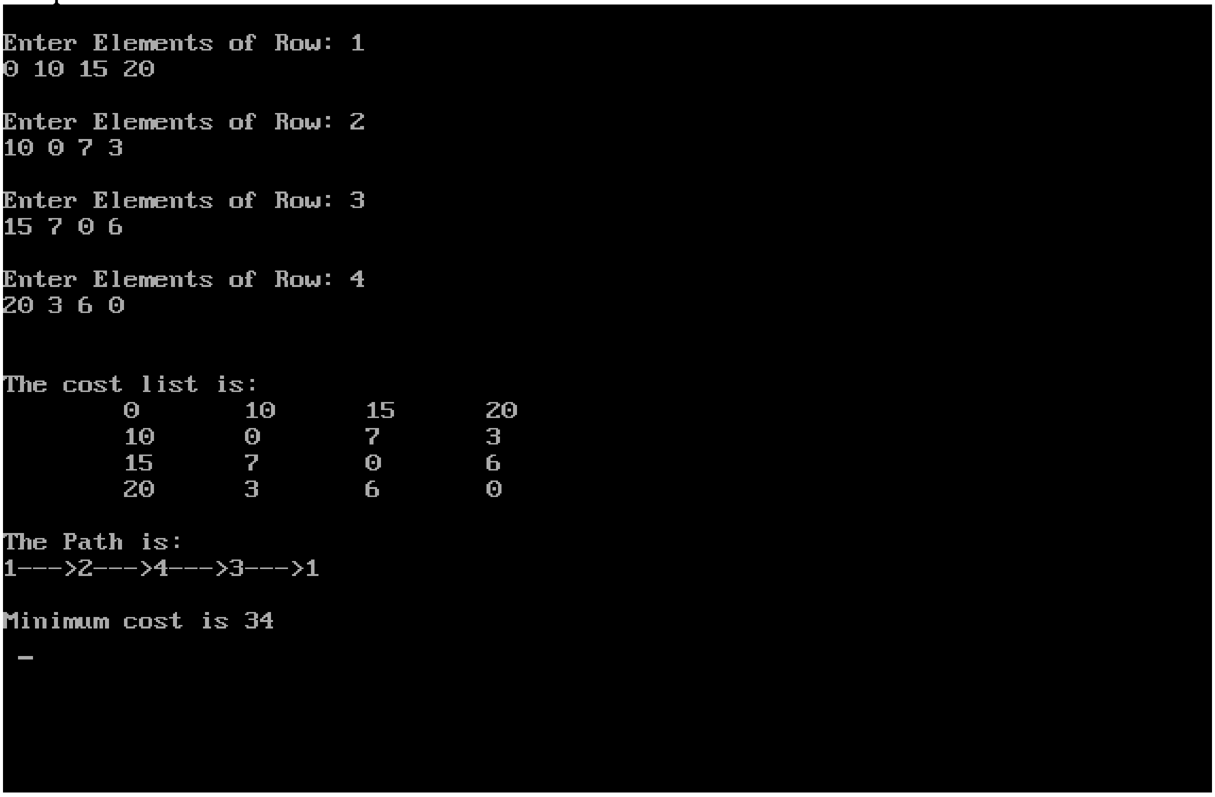
```
enter the value of N:
5
enter the number:
90
50
70
40
60
the numbers arranged in ascending order given below:
40      50      60      70      90      _
```

Program-2: Write a program to perform travelling salesman problem

```
#include<stdio.h>

#include<conio.h>

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

void takeInput() {
  int i, j;
  printf("Enter the number of villages: ");
  scanf("%d", & n);
  printf("\nEnter the Cost Matrix\n");
  for (i = 0; i < n; i++) {
    printf("\nEnter Elements of Row: %d\n", i + 1);
    for (j = 0; j < n; j++)
    scanf("%d", & ary[i][j]);
    completed[i] = 0;
  }
  printf("\n\nThe cost list is:");
  for (i = 0; i < n; i++) {
    printf("\n");
    for (j = 0; j < n; j++)
      printf("\t%d", ary[i][j]);
  }
}

void mincost(int city) {
  int i, ncity;
  completed[city] = 1;
  printf("%d--->", city + 1);
  ncity = least(city);
  if (ncity == 999) {
    ncity = 0;
    printf("%d", ncity + 1);
    cost += ary[city][ncity];
    return;
  }
  mincost(ncity);
}
```

```
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];
       kmin = ary[c][i];
       nc = i;
     }
 }
 if (min != 999) cost += kmin;
 return nc;
}

void main() {
 takeInput();
 printf("\n\nThe Path is:\n");
 mincost(0); //passing 0 because starting vertex
 printf("\n\nMinimum cost is %d\n ", cost);
 getch();
}
```

Output:

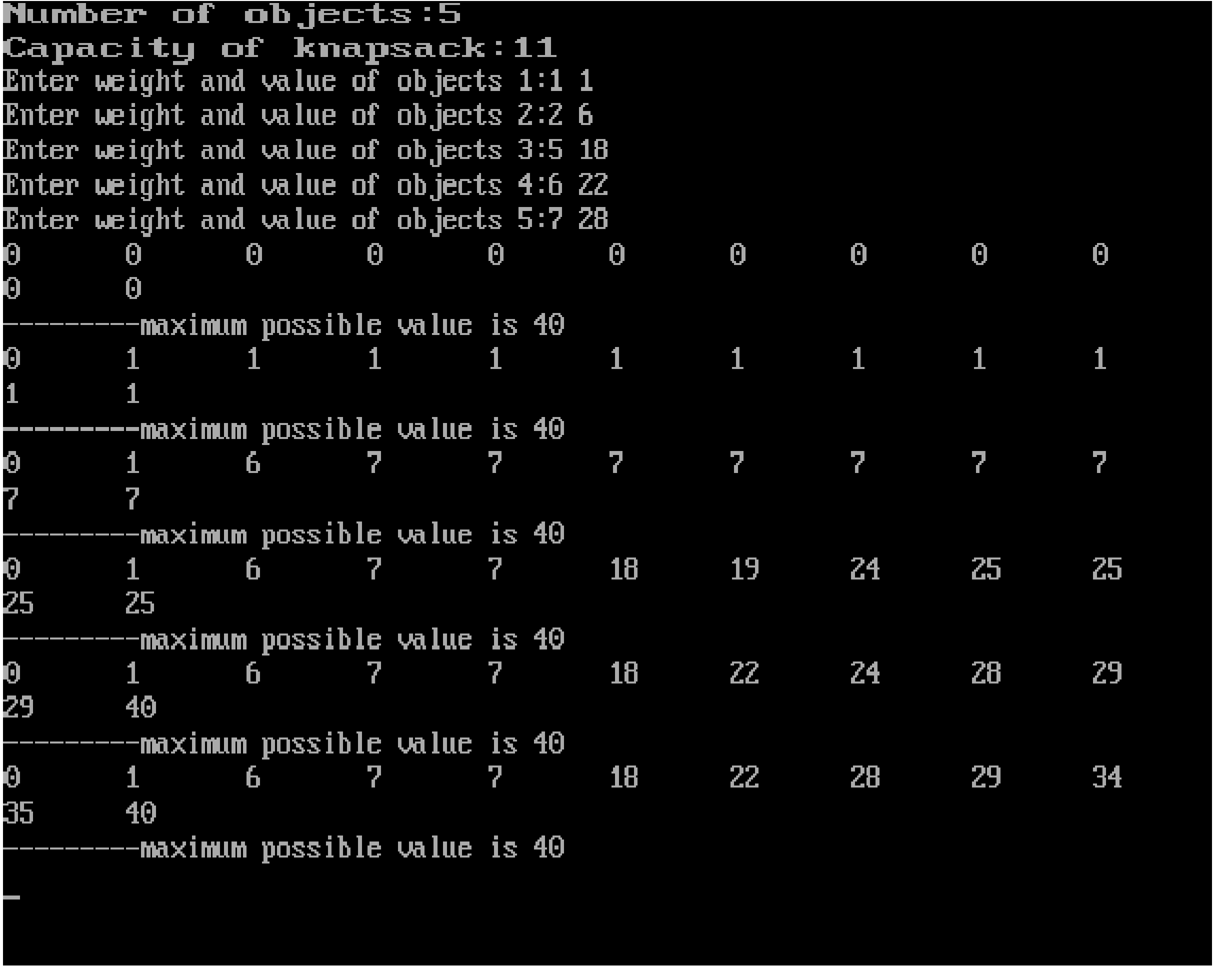


```
Enter Elements of Row: 1
0 10 15 20

Enter Elements of Row: 2
10 0 7 3

Enter Elements of Row: 3
15 7 0 6

Enter Elements of Row: 4
20 3 6 0


The cost list is:
        0       10      15      20
        10      0       7       3
        15      7       0       6
        20      3       6       0

The Path is:
1--->2--->4--->3--->1

Minimum cost is 34
 _
```

Program-3: Write a program to implement dynamic programming algorithm for the 0/1 knapsack problem

```
#include<stdio.h>

#include<conio.h>

void main() {
  int v[20], w[20], i, j, n, W;
  void knapsack(int[], int[], int, int);
  clrscr();
  printf("Number of objects:");
  scanf("%d", & n);

  printf("Capacity of knapsack:");
  scanf("%d", & W);
  for (i = 1; i <= n; i++) {
    printf("Enter weight and value of objects %d:", i);
    scanf("%d", & w[i]);
    scanf("%d", & v[i]);
  }
  knapsack(v, w, n, W);

  getch();
}

void knapsack(int v[], int w[], int n, int W) {
  int k[20][20], i, j;

  for (i = 0; i <= W; i++) {
    for (j = 0; j <= W; j++) {

      if (i == 0 || j == 0) {
        k[i][j] = 0;
      } else if (j < w[i]) {
        k[i][j] = k[i - 1][j];
      } else {
        if (k[i - 1][j] > k[i - 1][j - w[i]] + v[i]) {
          k[i][j] = k[i - 1][j];
        } else {
          k[i][j] = k[i - 1][j - w[i]] + v[i];
        }
      }
    }
  }
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= W; j++) {
      printf("%d\t", k[i][j]);
    }
    printf("\n");
    printf("         ");
    printf("maximum possible value is %d\n", k[n][W]);
    getch();
  }
}
```

Output:



```
Number of objects:5
Capacity of knapsack:11
Enter weight and value of objects 1:1 1
Enter weight and value of objects 2:2 6
Enter weight and value of objects 3:5 18
Enter weight and value of objects 4:6 22
Enter weight and value of objects 5:7 28
0       0       0       0       0       0       0       0       0       0
0       0
---------maximum possible value is 40
0       1       1       1       1       1       1       1       1       1
1       1
---------maximum possible value is 40
0       1       6       7       7       7       7       7       7       7
7       7
---------maximum possible value is 40
0       1       6       7       7       18      19      24      25      25
25      25
---------maximum possible value is 40
0       1       6       7       7       18      22      24      28      29
29      40
---------maximum possible value is 40
0       1       6       7       7       18      22      28      29      34
35      40
---------maximum possible value is 40
_
```

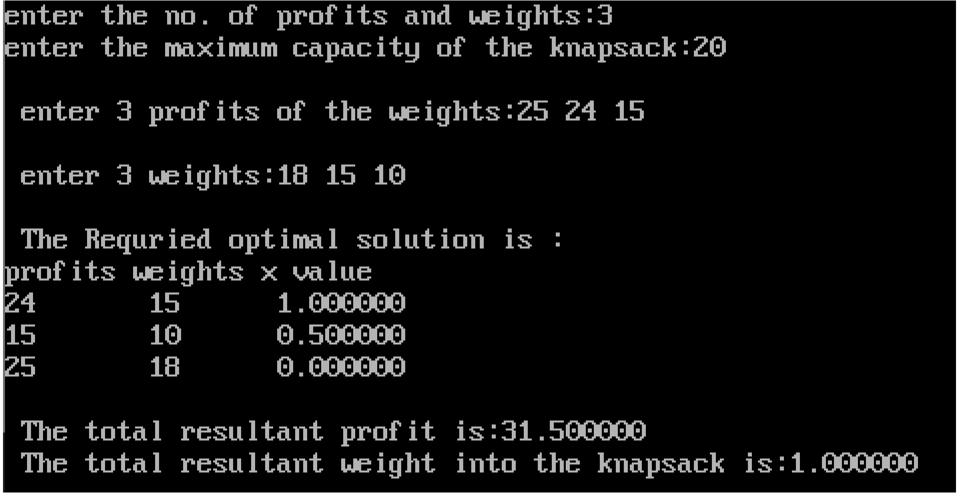
Program-4: Write a program to perform knapsack problem using Greedy solution

```
#include<stdio.h>

#include<conio.h>

void readf();
void knapsack(int, int);
void dsort(int n);
void display(int);

int p[20], w[20], n, m;
double x[20], d[20], temp, res = 0.0, sum = 0.0;

void readf() {
  int i, m, n, j;
  printf("enter the no. of profits and weights:");
  scanf("%d", & n);
  printf("enter the maximum capacity of the knapsack:");
  scanf("%d", & m);
  printf("\n enter %d profits of the weights:", n);
  for (i = 0; i < n; i++)
    scanf("%d", & p[i]);
  printf("\n enter %d weights:", n);
  for (i = 0; i < n; i++)
    scanf("%d", & w[i]);
  for (i = 0; i < n; i++)
    d[i] = (double) p[i] / w[i];
  dsort(n);
  knapsack(m, n);
  display(n);
}
```

```
void dsort(int n) {
  int i, j, t;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n - 1; j++) {
      if (d[j] < d[j + 1]) {
        temp = d[j];
        d[j] = d[j + 1];
        d[j + 1] = temp;
        t = p[j];

        p[j] = p[j + 1];
        p[j + 1] = t;
        t = w[j];
        w[j] = w[j + 1];
        w[j + 1] = t;
      }
    }
  }

}

void display(int n) {
  int i, m;
  printf("\n The Requried optimal solution is :\n");
  printf("profits weights x value\n");
  for (i = 0; i < n; i++) {
    printf("%d\t %d\t %f\n", p[i], w[i], x[i]);
    sum = sum + (p[i] * x[i]);
    res = res + (w[i] = x[i]);
  }
  printf("\n The total resultant profit is:%f\n", sum);
  printf("\ The total resultant weight into the knapsack is:%f\n", res);
}
```

```
void knapsack(int m, int n) {
  int i, cu = m;
  for (i = 0; i < n; i++) {
    x[i] = 0.0;
  }
  for (i = 0; i < n; i++) {
    if (w[i] < cu) {
      x[i] = 1.0;
      cu = cu - w[i];
    } else break;
  }
  if (i <= n) {
    x[i] = (double) cu / w[i];
  }
}

int main() {
  clrscr();
  readf();
  getch();
  return 0;
}
```

Output:



```
enter the no. of profits and weights:3
enter the maximum capacity of the knapsack:20

 enter 3 profits of the weights:25 24 15

 enter 3 weights:18 15 10

 The Requried optimal solution is :
profits weights x value
24       15      1.000000
15       10      0.500000
25       18      0.000000

 The total resultant profit is:31.500000
 The total resultant weight into the knapsack is:1.000000
```

Program-5: Write a program to implement the DFS and BFS algorithm for a graph

```
#include<stdio.h>

int q[20], top = -1, front = -1, rear = -1, vis[20], a[20][20], stack[20];
int delete();
void add(int item);
void bfs(int s, int n);
void dfs(int s, int n);
void push(int item);
int pop();

void main() {
  int n, i, s, ch, j;
  char c, dummy;
  clrscr();
  printf("ENTER THE NUMBER VERTICES:");

  scanf("%d", & n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0: ", i, j);
      scanf("%d", & a[i][j]);
    }
  }

  printf("THE ADJACENT MATRIX IS\n");
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= n; j++) {
      printf("%d", a[i][j]);
    }
    printf("\n");
  }
```

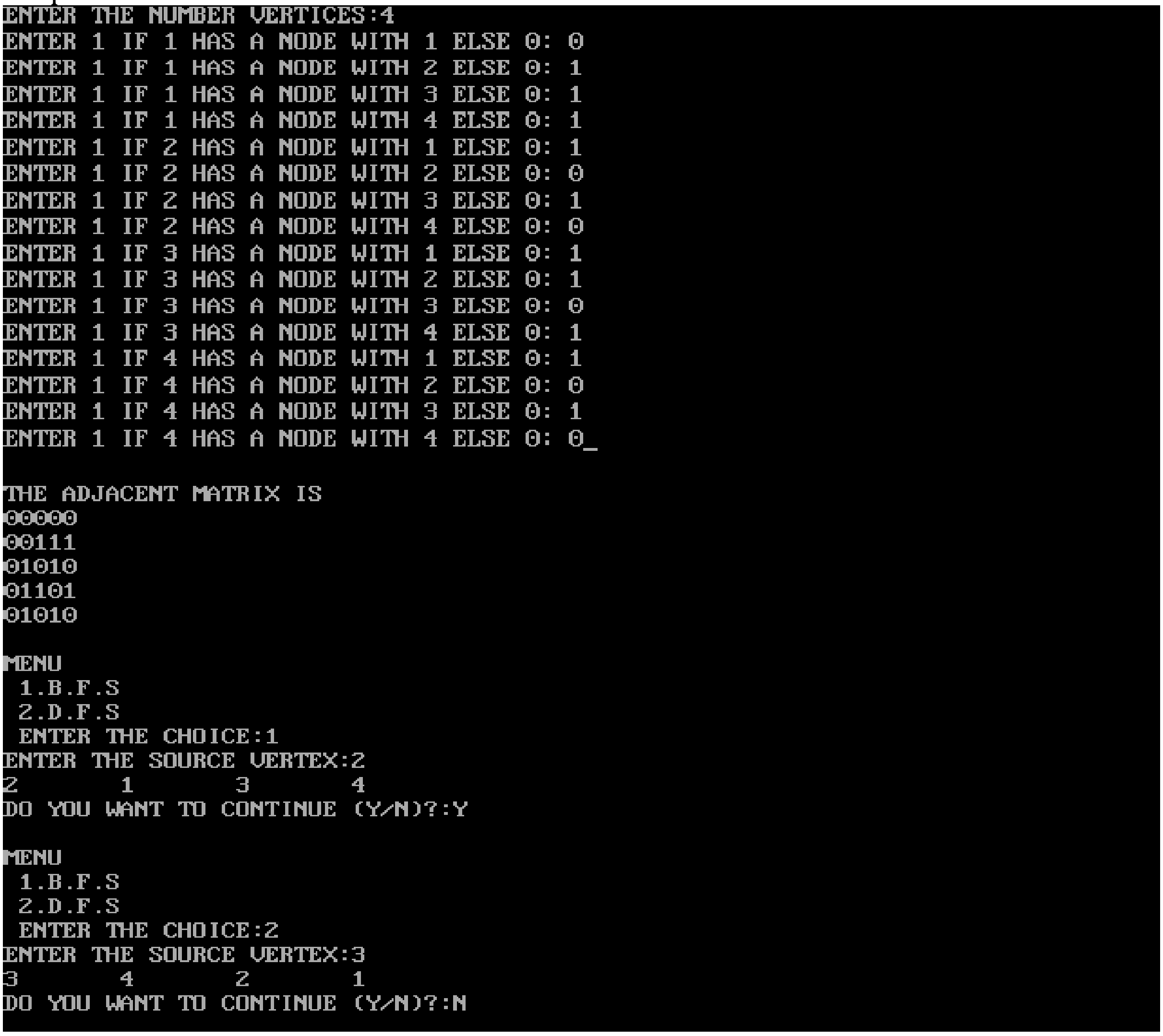
```
 do {
   for (i = 1; i <= n; i++) vis[i] = 0;
   printf("\nMENU");
   printf("\n 1.B.F.S");
   printf("\n 2.D.F.S");
   printf("\n ENTER THE CHOICE:");
   scanf("%d", & ch);
   printf("ENTER THE SOURCE VERTEX:");
   scanf("%d", & s);
   switch (ch) {
   case 1:
     bfs(s, n);
     break;
   case 2:
     dfs(s, n);
     break;
   }

   printf("\n DO YOU WANT TO CONTINUE (Y\N)?");
   scanf("%c", & dummy);
   scanf("%c", & c);
 }

 while ((c == 'y') || (c == 'Y'));
}
```

```
void bfs(int s, int n) {
  int p, i;
  add(s);
  vis[s] = 1;
  p = delete();
  if (p != 0)
    printf("%d\t", p);
  while (p != 0) {
    for (i = 1; i <= n; i++)
      if ((a[p][i] != 0) && (vis[i] == 0)) {
        add(i);
        vis[i] = 1;
      }
    p = delete();
    if (p != 0) printf("%d\t", p);
  }
  for (i = 1; i <= n; i++)
    if (vis[i] == 0)
      bfs(i, n);
}

void add(int item) {
  if (rear == 19) printf("QUEUE FULL");
  else {
    if (rear == -1) {
      q[++rear] = item;
      front++;
    } else
      q[++rear] = item;
  }

}
```

```
int delete() {
 int k;
 if ((front > rear) || (front == -1)) return (0);
 else {
  k = q[front++];
  return (k);
 }
}

void dfs(int s, int n) {
 int i, k;
 push(s);
 vis[s] = 1;
 k = pop();
 if (k != 0) printf("%d\t", k);
 while (k != 0) {
  for (i = 1; i <= n; i++)
   if ((a[k][i] != 0) && (vis[i] == 0)) {
    push(i);
    vis[i] = 1;
   }
  k = pop();
  if (k != 0) printf("%d\t", k);
 }
 for (i = 1; i <= n; i++)
  if (vis[i] == 0)
   dfs(i, n);
}

void push(int item) {
 if (top == 19)
  printf("stack overflow");

 else
  stack[++top] = item;
}
```

```
int pop() {
  int k;
  if (top == -1) return (0);
  else {
    k = stack[top--];
    return (k);
  }
}
```

Output:



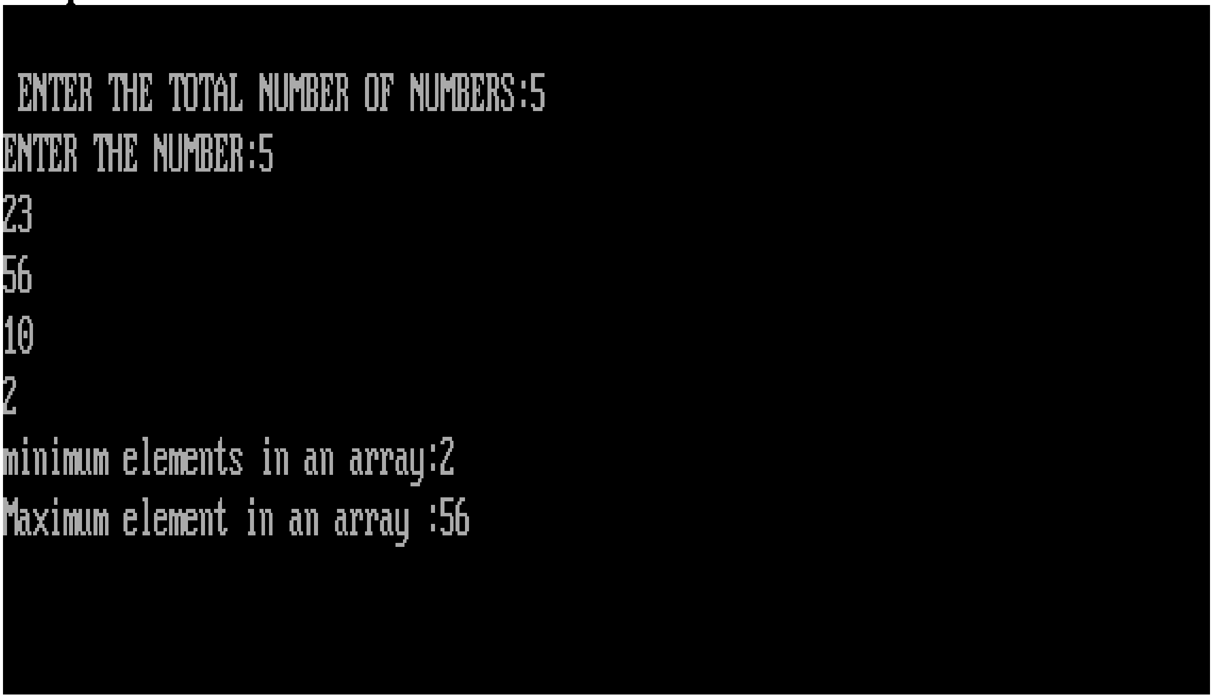
```
ENTER THE NUMBER VERTICES:4
ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0: 0
ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0: 1
ENTER 1 IF 1 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 1 HAS A NODE WITH 4 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0: 0
ENTER 1 IF 2 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 4 ELSE 0: 0
ENTER 1 IF 3 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 3 HAS A NODE WITH 2 ELSE 0: 1
ENTER 1 IF 3 HAS A NODE WITH 3 ELSE 0: 0
ENTER 1 IF 3 HAS A NODE WITH 4 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 2 ELSE 0: 0
ENTER 1 IF 4 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 4 ELSE 0: 0_

THE ADJACENT MATRIX IS
00000
00111
01010
01101
01010

MENU
 1.B.F.S
 2.D.F.S
 ENTER THE CHOICE:1
ENTER THE SOURCE VERTEX:2
2       1       3       4
DO YOU WANT TO CONTINUE (Y/N)?:Y

MENU
 1.B.F.S
 2.D.F.S
 ENTER THE CHOICE:2
ENTER THE SOURCE VERTEX:3
3       4       2       1
DO YOU WANT TO CONTINUE (Y/N)?:N
```

Program-6: Write a program to find minimum and maximum value in an array using divide and conquer

```
#include<stdio.h>

#include<conio.h>

int max, min;
int a[100];

void maxmin(int i, int j) {
  int max1, min1, mid;

  if (i == j) {
    max = min = a[i];
  } else {
    if (i == j - 1) {
      if (a[i] < a[j]) {
        max = a[j];
        min = a[i];
      } else {
        max = a[i];
        min = a[j];
      }
    } else {
      mid = (i + j) / 2;
      maxmin(i, mid);
      max1 = max;
      min1 = min;
      maxmin(mid + 1, j);
      if (max < max1)
        max = max1;
      if (min > min1)
        min = min1;
    }
  }
}
```

```
int main() {
  int i, num;
  clrscr();
  printf("\n ENTER THE TOTAL NUMBER OF NUMBERS:");
  scanf("%d", & num);
  printf("ENTER THE NUMBER:");
  for (i = 1; i <= num; i++) scanf("%d", & a[i]);
  max = a[0];
  min = a[0];
  maxmin(1, num);
  printf("minimum elements in an array:%d\n", min);
  printf("Maximum element in an array :%d\n", max);
  getch();
  return 0;
}
```

Output:



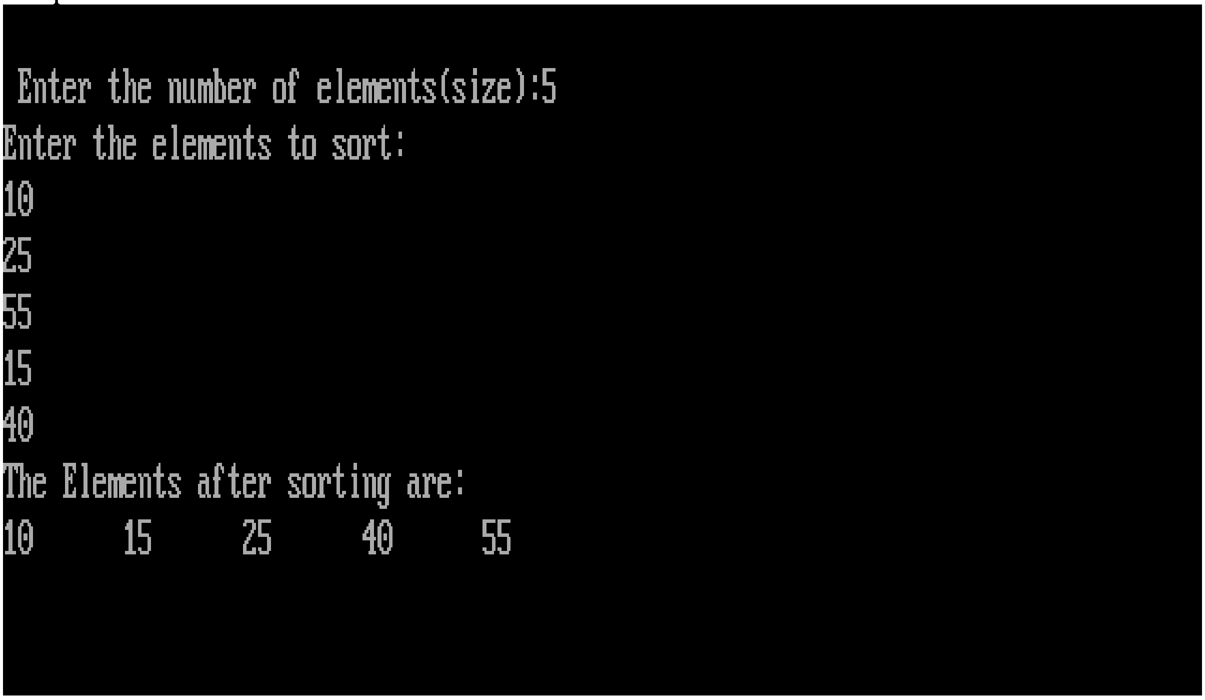
```

 ENTER THE TOTAL NUMBER OF NUMBERS:5
ENTER THE NUMBER:5
23
56
10
2
minimum elements in an array:2
Maximum element in an array :56
```

Program-7: Write a program to implement divide and conquer strategy. Eg: Quick sort algorithm for sorting list of integers in ascending order

```
#include<stdio.h>

#include<conio.h>

void qsort(int[], int, int);
int partition(int[], int, int);

void qsort(int a[], int first, int last) {
  int j;
  if (first < last) {
    j = partition(a, first, last + 1);
    qsort(a, first, j - 1);
    qsort(a, j + 1, last);
  }
}

int partition(int a[], int first, int last) {
  int v = a[first];
  int i = first;
  int j = last;
  int temp = 0;
  do {
    do {
      i++;
    } while (a[i] < v);

    do {
      j--;
    }

    while (a[j] > v);
    if (i < j) {
      temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }
  }
  while (i < j);
  a[first] = a[j];
  a[j] = v;

  return j;
}
```

```
int main() {
  int a[40], i, n;
  clrscr();
  printf("\n Enter the number of elements(size):");
  scanf("%d", & n);
  printf("Enter the elements to sort:\n");
  for (i = 0; i < n; i++)
    scanf("%d", & a[i]);
  qsort(a, 0, n - 1);
  printf("The Elements after sorting are:\n");
  for (i = 0; i < n; i++) {
    printf("%d\t", a[i]);
  }
  getch();
  return 0;
}
```

Output:



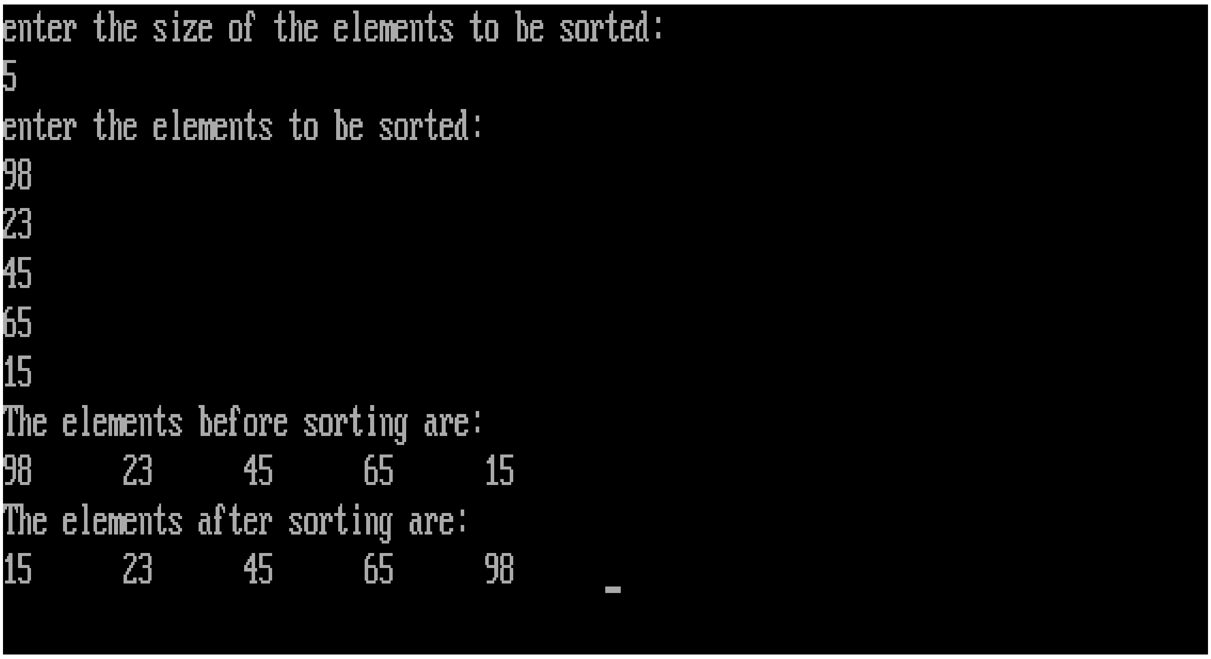
```
 Enter the number of elements(size):5
Enter the elements to sort:
10
25
55
15
40
The Elements after sorting are:
10      15      25      40      55
```

Program-8:Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order

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

void merge(int[], int, int, int);
void mergesort(int[], int, int);

void merge(int a[25], int low, int mid, int high) {
 int b[25], h, i, j, k;
 h = low;
 i = low;
 j = mid + 1;
 while ((h <= mid) && (j <= high)) {
  if (a[h] < a[j]) {
   b[i] = a[h];
   h++;
  } else {
   b[i] = a[j];
   j++;
  }
  i++;
 }
 if (h > mid) {
  for (k = j; k <= high; k++) {
   b[i] = a[k];
   i++;
  }
 } else {
  for (k = h; k <= mid; k++) {
   b[i] = a[k];

   i++;
  }
 }
 for (k = low; k <= high; k++) {
  a[k] = b[k];
 }
}

void mergesort(int a[25], int low, int high) {
 int mid;
 if (low < high) {
  mid = (low + high) / 2;
  mergesort(a, low, mid);
  mergesort(a, mid + 1, high);
  merge(a, low, mid, high);
 }
}
```

```
void main() {
  int a[25], i, n;
  clrscr();
  printf("enter the size of the elements to be sorted:\n");
  scanf("%d", & n);
  printf("enter the elements to be sorted:\n");
  for (i = 0; i < n; i++)
    scanf("%d", & a[i]);
  printf("The elements before sorting are:\n");
  for (i = 0; i < n; i++)
    printf("%d\t", a[i]);
  mergesort(a, 0, n - 1);
  printf("\nThe elements after sorting are:\n");
  for (i = 0; i < n; i++)
    printf("%d\t", a[i]);
  getch();
}
```

Output:



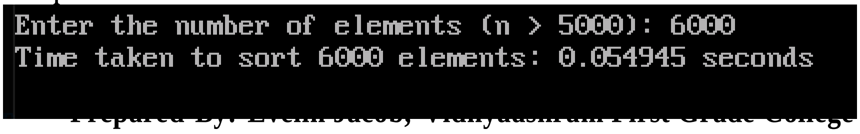
```
enter the size of the elements to be sorted:
5
enter the elements to be sorted:
98
23
45
65
15
The elements before sorting are:
98      23      45      65      15
The elements after sorting are:
15      23      45      65      98
```

Program-9: Sort a given set of n integer element using merge sort method and compute its time complexity .Run the program for varied values of n>5000 and record the time taken to sort

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void merge(int arr[], int l, int m, int r) {
  int n1 = m - l + 1;
  int n2 = r - m;

  int * L = (int * ) malloc(n1 * sizeof(int));
  int * R = (int * ) malloc(n2 * sizeof(int));
  int i,j,k;
  for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
  for (j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];

  i = 0, j = 0, k = l;
  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      arr[k++] = L[i++];
    } else {
      arr[k++] = R[j++];
    }
  }

  while (i < n1)
    arr[k++] = L[i++];
  while (j < n2)
    arr[k++] = R[j++];

  free(L);
  free(R);
}
```

```
void mergeSort(int arr[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);

    merge(arr, l, m, r);
  }
}

int main() {
  int n,i;
  int *arr;
  double time_taken;
  clock_t start,end;
  printf("Enter the number of elements (n > 5000): ");
  scanf("%d", & n);

  if (n <= 5000) {
    printf("The number of elements must be greater than 5000.\n");
    return 1;
  }

  arr = (int * ) malloc(n * sizeof(int));

  // Generate random elements
  srand(time(NULL));
  for (i = 0; i < n; i++) {
    arr[i] = rand() % 10000; // Random integers between 0 and 9999
  }

  start = clock();

  mergeSort(arr, 0, n - 1);

  end = clock();

  time_taken = (double)(end - start) / CLOCKS_PER_SEC;

  printf("Time taken to sort %d elements: %f seconds\n", n, time_taken);

  free(arr);
  getch();
  return 0;
}
```

Output:



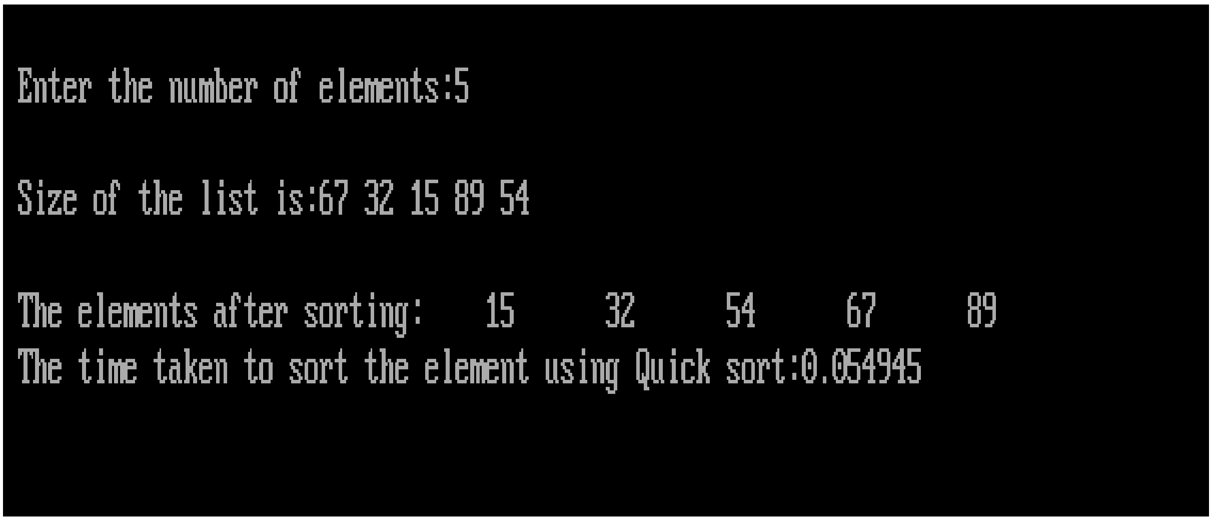
```
Enter the number of elements (n > 5000): 6000
Time taken to sort 6000 elements: 0.054945 seconds
```

Program-10: Sort a given set of n integer element using quick sort method and compute its time complexity .Run the program for varied values of n>5000 and record the time taken to sort

```
#include<stdio.h>

#include<conio.h>

#include<time.h>

#include<dos.h>

#include<stdlib.h>

void quick(int a[], int low, int high);
int partition(int a[], int low, int high);

void main() {
  int i, n, * a = NULL;
  clock_t s, e;
  clrscr();
  printf("\n Enter the number of elements:");
  scanf("%d", & n);

  for (i = 0; i < n; i++) {
    a = (int * ) malloc(sizeof(int));
    if (a == NULL)
      printf("Error\n");
    else {
      printf("\n Size of the list is:");
      for (i = 0; i < n; i++) {
        scanf("%d", & a[i]);
      }
      s = clock();
      delay(30);
      quick(a, 0, n - 1);
      delay(50);
      e = clock();
      printf("\n The elements after sorting:");
      for (i = 0; i < n; i++)
        printf("\t%d", a[i]);
      printf("\n The time taken to sort the element using Quick sort:%f", (e - s) / CLK_TCK);
    }
  }
  getch();
}
```

```
int partition(int a[], int low, int high) {
  int pivot = a[low];
  int temp, i = low;
  int j = high + 1;
  while (i <= j) {
    do {
      i = i + 1;
    } while (pivot >= a[i]);

    do {
      j = j - 1;
    } while (pivot < a[j]);

    if (i < j) {
      temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }
  }
  temp = a[j];
  a[j] = a[low];
  a[low] = temp;
  return j;
}

void quick(int a[], int low, int high) {
  if (low < high) {
    int k = partition(a, low, high);
    quick(a, low, k - 1);
    quick(a, k + 1, high);
  }
}
```

Output:

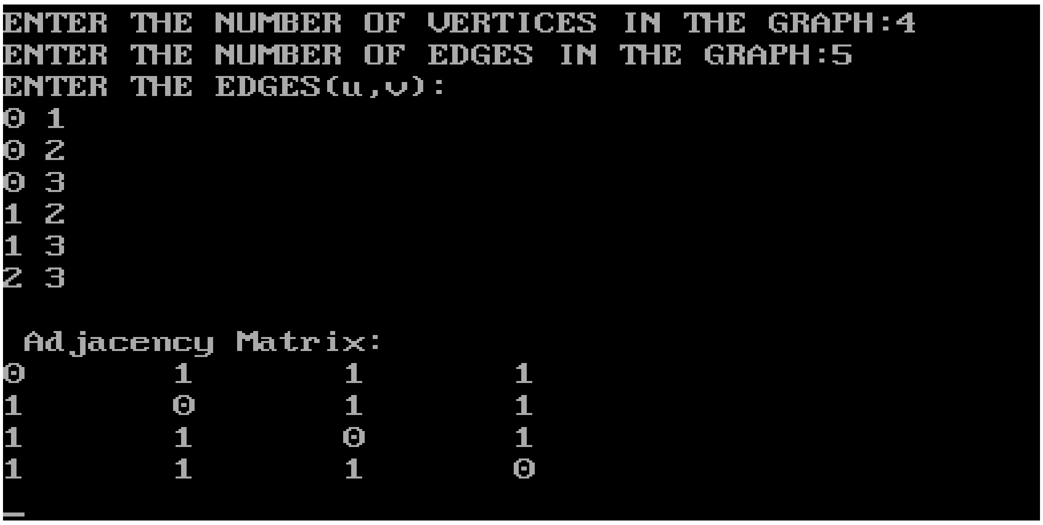


```
Enter the number of elements:5


Size of the list is:67 32 15 89 54


The elements after sorting:    15      32      54      67      89
The time taken to sort the element using Quick sort:0.054945
```

Program-11: Write a c program that excepts the vertices and edges for a graph and stores adjacency matrix

```
#include<stdio.h>

#define MAX_VERTICES 100

int main() {
  int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {
    0
  };
  int numVertices, numEdges;
  int i, j, u, v;
  printf("ENTER THE NUMBER OF VERTICES IN THE GRAPH:");
  scanf("%d", & numVertices);
  printf("ENTER THE NUMBER OF EDGES IN THE GRAPH:");
  scanf("%d", & numEdges);
  printf("ENTER THE EDGES(u,v):\n");

  for (i = 0; i <= numEdges; i++) {
    scanf("%d%d", & u, & v);
    adjMatrix[u][v] = 1;
    adjMatrix[v][u] = 1;
  }
  printf("\n Adjacency Matrix:\n");
  for (i = 0; i < numVertices; i++) {
    for (j = 0; j < numVertices; j++) {
      printf("%d\t", adjMatrix[i][j]);
    }
    printf("\n");
  }
  getch();
  return 0;
}
```

Output:



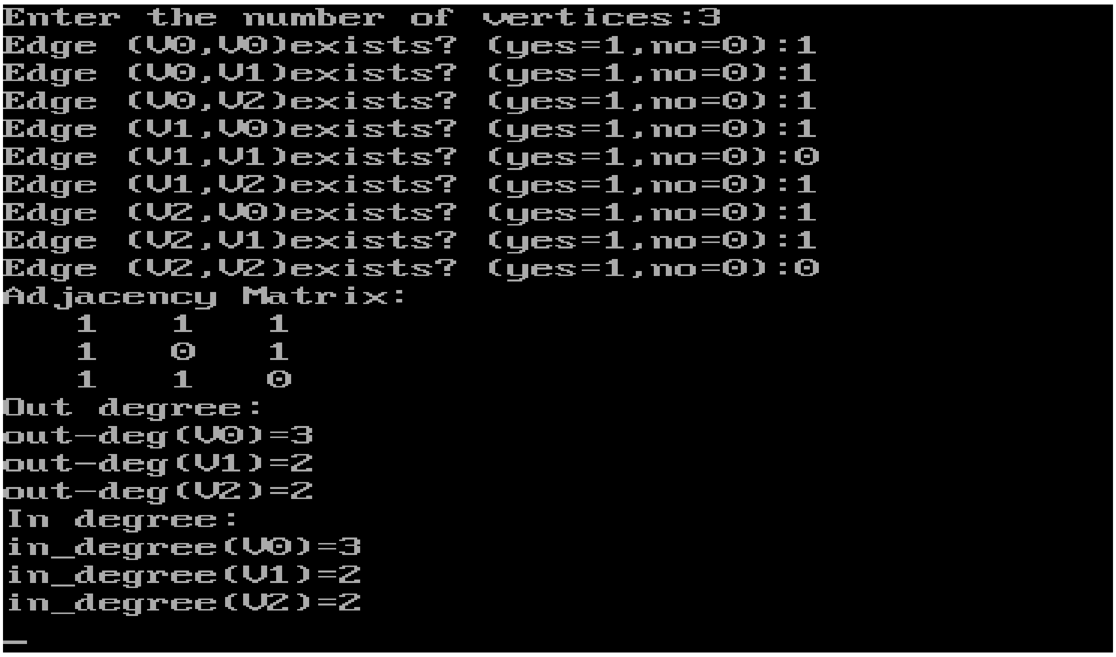
```
ENTER THE NUMBER OF VERTICES IN THE GRAPH:4
ENTER THE NUMBER OF EDGES IN THE GRAPH:5
ENTER THE EDGES(u,v):
0 1
0 2
0 3
1 2
1 3
2 3

 Adjacency Matrix:
0       1       1       1
1       0       1       1
1       1       0       1
1       1       1       0
_
```

Program-12: Implement function to print indegree, outdegree and display that adjacency matrix

```
#include<stdio.h>

#include<conio.h>

#define MAX 10

void accept_graph(int G[][MAX], int n) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      printf("Edge (V%d,V%d)exists? (yes=1,no=0):", i, j);
      scanf("%d", & G[i][j]);
    }
  }
}

void disp_adj_mat(int G[][MAX], int n) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      printf("%4d", G[i][j]);
    }
    printf("\n");
  }
}

void calc_out_degree(int G[][MAX], int n) {
  int i, j, sum;
  for (i = 0; i < n; i++) {

    sum = 0;
    for (j = 0; j < n; j++) {
      sum += G[i][j];
    }
    printf("out-deg(V%d)=%d\n", i, sum);
  }
}

void calc_in_degree(int G[][MAX], int n) {
  int i, j, sum;
  for (i = 0; i < n; i++) {
    sum = 0;
    for (j = 0; j < n; j++) {
      sum += G[j][i];
    }
    printf("in_degree(V%d)=%d\n", i, sum);
  }
}
```

```
void main() {
 int G[MAX][MAX], n;
 clrscr();
 printf("Enter the number of vertices:");
 scanf("%d", & n);
 accept_graph(G, n);
 printf("Adjacency Matrix:\n");
 disp_adj_mat(G, n);
 printf("Out degree:\n");
 calc_out_degree(G, n);
 printf("In degree:\n");
 calc_in_degree(G, n);
 getch();
      }
```

Output:



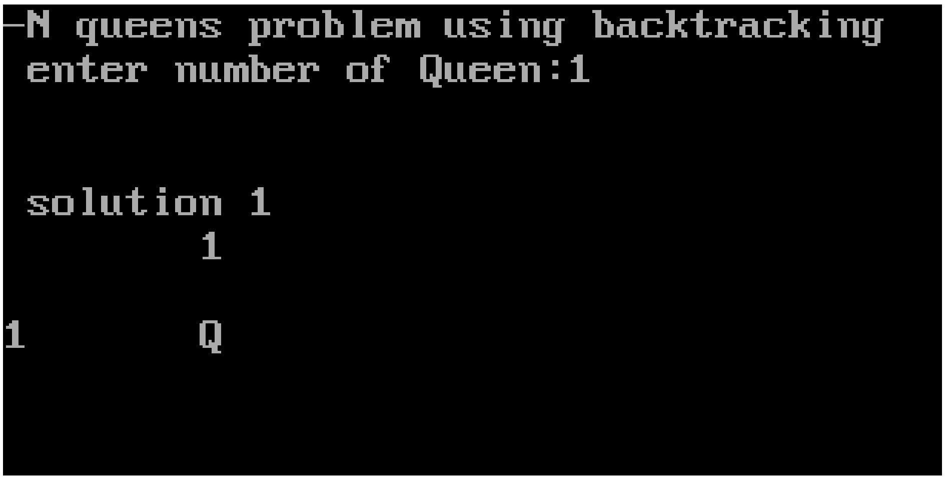
```
Enter the number of vertices:3
Edge (V0,V0)exists? (yes=1,no=0):1
Edge (V0,V1)exists? (yes=1,no=0):1
Edge (V0,V2)exists? (yes=1,no=0):1
Edge (V1,V0)exists? (yes=1,no=0):1
Edge (V1,V1)exists? (yes=1,no=0):0
Edge (V1,V2)exists? (yes=1,no=0):1
Edge (V2,V0)exists? (yes=1,no=0):1
Edge (V2,V1)exists? (yes=1,no=0):1
Edge (V2,V2)exists? (yes=1,no=0):0
Adjacency Matrix:
   1   1   1
   1   0   1
   1   1   0
Out degree:
out-deg(V0)=3
out-deg(V1)=2
out-deg(V2)=2
In degree:
in_degree(V0)=3
in_degree(V1)=2
in_degree(V2)=2
_
```

Program-13: Write a program to implement back tracking algorithm for solving problems like NQueen

```
#include<stdio.h>

#include<conio.h>

int board[20], count;

int main() {
  int n, i, j;
  void queen(int row, int n);
  printf("-N queens problem using backtracking");
  printf("\n enter number of Queen:");
  scanf("%d", & n);
  queen(1, n);
  getch();
  return 0;
}

void print(int n) {
  int i, j;
  printf("\n\n solution %d\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) {
      if (board[i] == j)
        printf("\t Q");
      else
        printf("\t-");
    }
  }
}

int place(int row, int column) {
  int i, n;
  for (i = 1; i <= n; ++i) {

    if (board[i] == column)
      return 0;
    else
    if (abs(board[i] = column) == abs(i - row))
      return 0;
  }
  return 1;
}
```

```
void queen(int row, int n) {
  int column;
  for (column = 1; column <= n; ++column) {
    if (place(row, column)) {
      board[row] = column;
      if (row == n)
        print(n);
      else
        queen(row + 1, n);
    }
  }
}
```

Output:

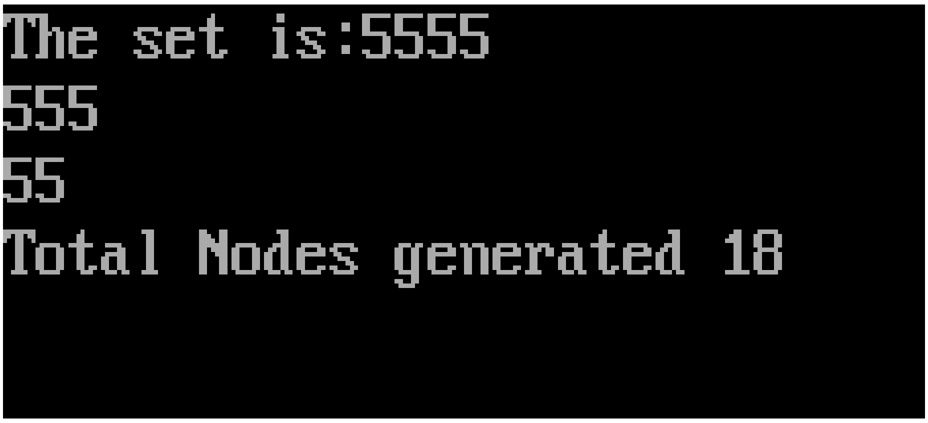


```
-N queens problem using backtracking
 enter number of Queen:1


 solution 1
        1

1       Q
```

Program-14: Write a program to implement back tracking algorithm for the sum of subset problem

```
#include<stdio.h>

#include<conio.h>

static int total;

void print(int a[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    printf("%d", 5, a[i]);
  }
  printf("\n");
}

void subset_sum(int s[], int t[], int s_size, int t_size, int sum, int item, int
  const target_sum) {
  int i;
  total++;
  if (target_sum == sum) {
    print(t, t_size);
    subset_sum(s, t, s_size, t_size - 1, sum - s[item], item + 1, target_sum);
    return;
  } else {
    for (i = item; i < s_size; i++) {
      t[t_size] = s[i];
      subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
    }
  }
}

void generateSubsets(int s[], int size, int target_sum) {
  int * tuplet_vector = (int * ) malloc(size * sizeof(int));
  subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
  free(tuplet_vector);
}

int main() {
  int set[] = {
    2,
    3,
    4,
    5
  };
  int size = sizeof(set) / sizeof(set[0]);
  clrscr();
  printf("The set is:");
  print(set, size);
  generateSubsets(set, size, 11);
  printf("Total Nodes generated %d\n", total);
  getch();
  return 0;
}
```

Output:



```
The set is:5555
555
55
Total Nodes generated 18
```

Program-15: Write a program to implement Greedy algorithm for job sequence with deadlines

```
#include<stdio.h>


#define MAX 100


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


void jobSequencingWithDeadline(Job jobs[], int n);


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

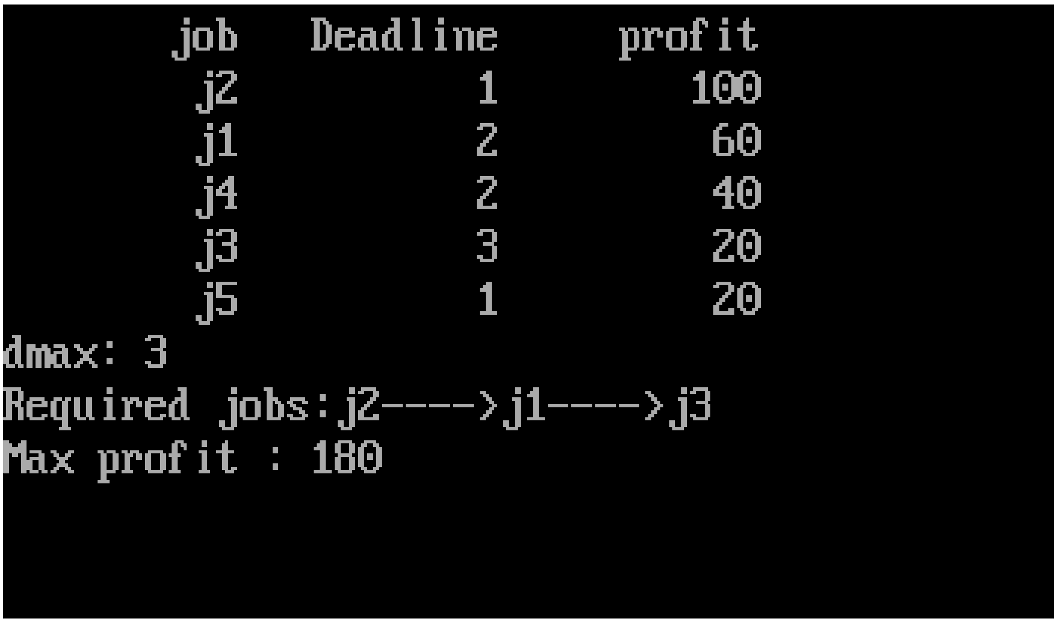
```
int main(void) {
 int i, j;

 Job jobs[5] = {
  {
   "j1",
   2,
   60
  },
  {
   "j2",
   1,
   100
  },
  {
   "j3",
   3,
   20
  },
{
   "j4",
   2,
   40
  },
  {
   "j5",
   1,
   20
  },
 };
 Job temp;
 int n = 5;
 for (i = 1; i < n; i++) {
  for (j = 0; j < n - 1; j++) {
   if (jobs[j + 1].profit > jobs[j].profit) {
    temp = jobs[j + 1];
    jobs[j + 1] = jobs[j];
    jobs[j] = temp;
   }
  }
 }
 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);
  }


  jobSequencingWithDeadline(jobs, n);
  return 0;
}

void jobSequencingWithDeadline(Job jobs[], int n) {
  int i, j, k, maxprofit;
  int timeslot[MAX];
  int filledTimeSlot = 0;
  int dmax = 0;
  for (i = 0; i < n; i++) {
    if (jobs[i].deadline > dmax) {
      dmax = jobs[i].deadline;
    }
  }
  for (i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
  }
```

```
 printf("dmax: %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--;
  }
  if (filledTimeSlot == dmax) {


    break;
  }
 }
```

```
  printf("Required jobs:");

  for (i = 1; i <= dmax; i++) {

    printf("%s", jobs[timeslot[i]].id);

    if (i < dmax) {

      printf("          >");

    }

  }

  maxprofit = 0;

  for (i = 1; i <= dmax; i++) {

    maxprofit += jobs[timeslot[i]].profit;

  }

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

getch();

}
```

Output:



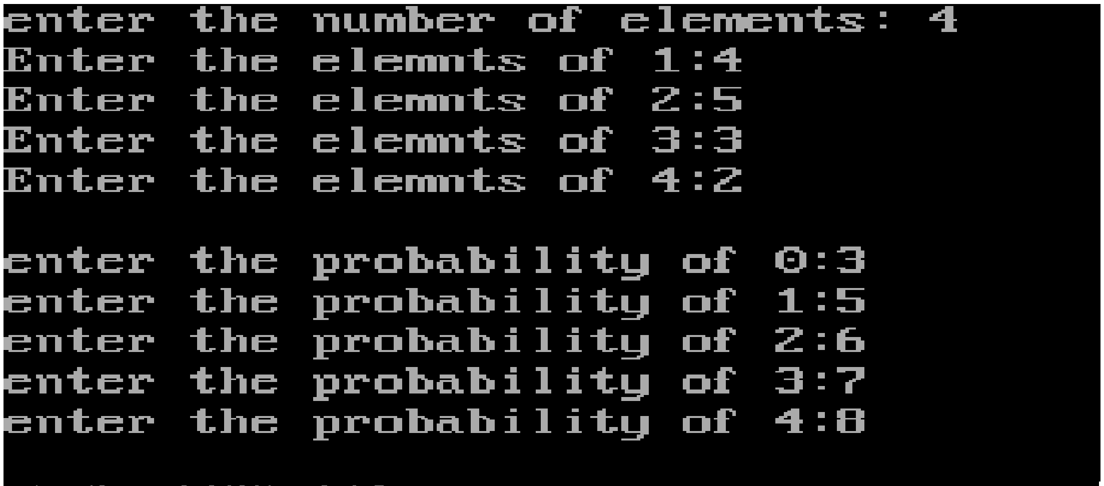
```
        job   Deadline     profit
         j2          1        100
         j1          2         60
         j4          2         40
         j3          3         20
         j5          1         20
dmax: 3
Required jobs:j2---->j1---->j3
Max profit : 180
```

Program-16: Write a program to implement dynamic programming algorithm for the optimial binary search tree problem

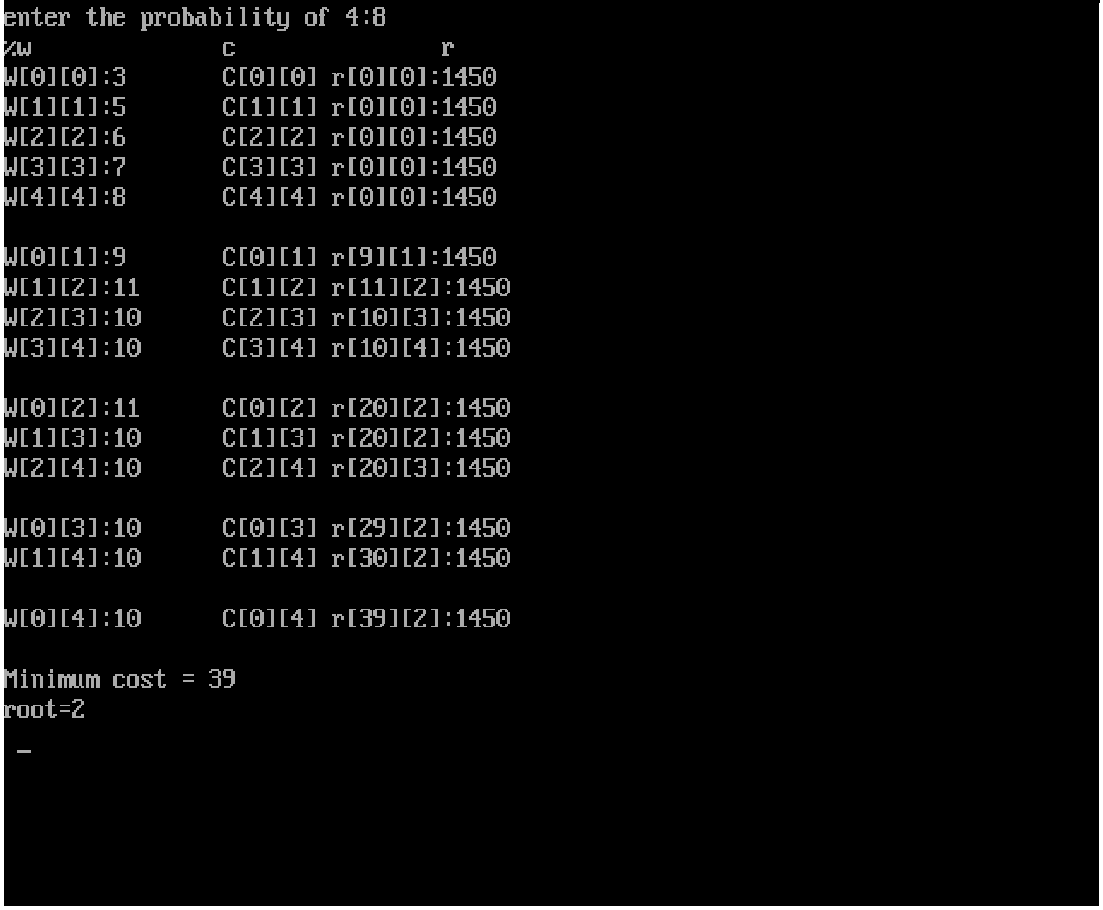
```
#include<stdio.h>
#include<conio.h>
#define MAX 10

void main() {
 char ele[MAX][MAX];
 int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX], p[MAX], q[MAX];
 int temp = 0, root, min, min1, n;
 int i, j, k, b;
 clrscr();
 printf("enter the numberof elemsnts: ");
 scanf("%d", & n);
 for (i = 1; i <= n; i++) {
  printf("Enter the elemnts of %d:", i);
  scanf("%d", & p[i]);
 }
 printf("\n");
 for (i = 0; i <= n; i++) {
  printf("enter the probability of %d:", i);
  scanf("%d", & q[i]);
 }
 printf("%w\t\tc\t\tr\n");
 for (i = 0; i <= n; i++) {
  for (j = 0; j <= n; j++) {
   if (i == j) {
    w[i][j] = q[i];
    c[i][j] = 0;
    r[i][j] = 0;
    printf("W[%d][%d]:%d\tC[%d][%d]\tr[%d][%d]:%d\n", i, j, w[i][j], i, j, c[i][j], r[i][j]);
   }
  }
 }
 printf("\n");
```

```
  for (b = 0; b < n; b++) {
    for (i = 0, j = b + 1; j < n + 1 && i < n + 1; j++, i++) {
      if (i != j && i < j) {
        w[i][j] = p[j] + q[j] + w[i][i - 1];
        min = 30000;
        for (k = i + 1; k <= j; k++) {
          min1 = c[i][k - 1] + c[k][j] + w[i][j];
          if (min > min1) {
            min = min1;
            temp = k;
          }
        }
        c[i][j] = min;
        r[i][j] = temp;
      }
      printf("W[%d][%d]:%d\tC[%d][%d]\tr[%d][%d]:%d\n", i, j, w[i][j], i, j, c[i][j], r[i][j]);
    }
    printf("\n");
  }
  printf("Minimum cost = %d\n", c[0][n]);
  root = r[0][n];
  printf("root=%d\n ", root);
  getch();
}
```

Output:



```
enter the number of elements: 4
Enter the elemnts of 1:4
Enter the elemnts of 2:5
Enter the elemnts of 3:3
Enter the elemnts of 4:2

enter the probability of 0:3
enter the probability of 1:5
enter the probability of 2:6
enter the probability of 3:7
enter the probability of 4:8
```

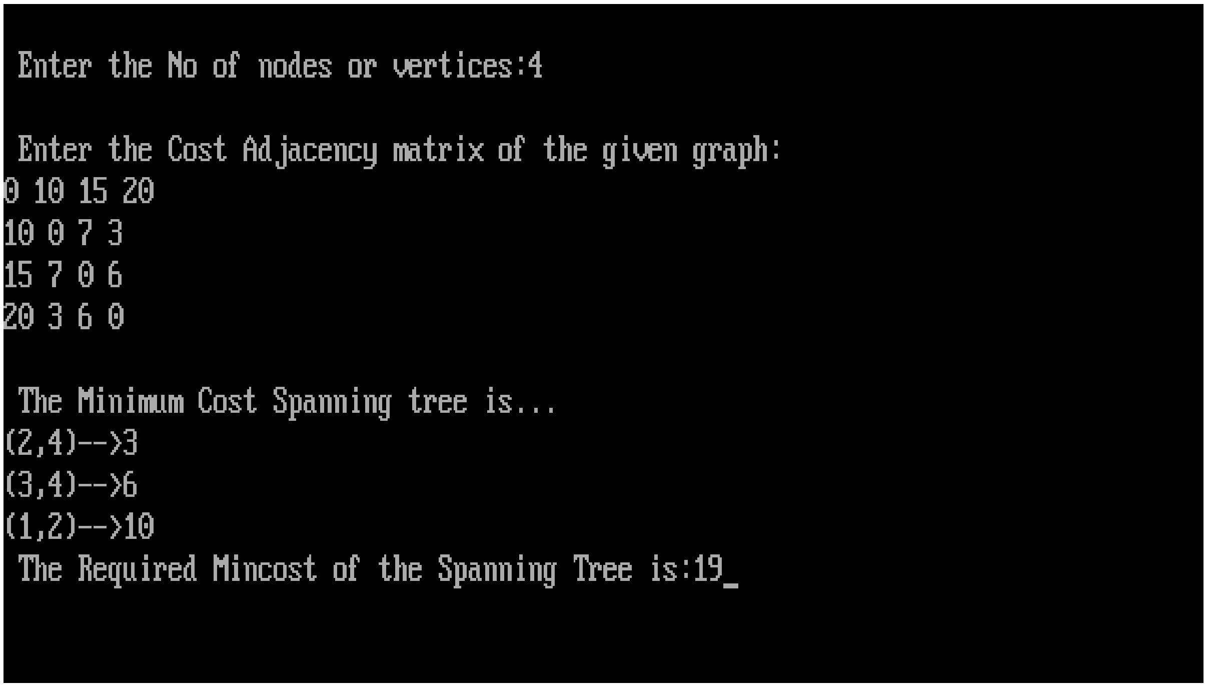


```
enter the probability of 4:8
%w              c               r
W[0][0]:3       C[0][0] r[0][0]:1450
W[1][1]:5       C[1][1] r[0][0]:1450
W[2][2]:6       C[2][2] r[0][0]:1450
W[3][3]:7       C[3][3] r[0][0]:1450
W[4][4]:8       C[4][4] r[0][0]:1450

W[0][1]:9       C[0][1] r[9][1]:1450
W[1][2]:11      C[1][2] r[11][2]:1450
W[2][3]:10      C[2][3] r[10][3]:1450
W[3][4]:10      C[3][4] r[10][4]:1450

W[0][2]:11      C[0][2] r[20][2]:1450
W[1][3]:10      C[1][3] r[20][2]:1450
W[2][4]:10      C[2][4] r[20][3]:1450

W[0][3]:10      C[0][3] r[29][2]:1450
W[1][4]:10      C[1][4] r[30][2]:1450

W[0][4]:10      C[0][4] r[39][2]:1450

Minimum cost = 39
root=2

 _
```

Program-17: Write a program to implement prim’s algorithm to generate minimum cost spanning tree

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

int n, cost[10][10], temp, nears[10];
void readv();
void primsalg();

void readv()
{
  int i, j;
  printf("\n Enter the No of nodes or vertices:");
  scanf("%d", & n);
  printf("\n Enter the Cost Adjacency matrix of the given graph:");
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      scanf("%d", & cost[i][j]);
      if ((cost[i][j] == 0) && (i != j)) {
        cost[i][j] = 999;
      }
    }
  }
}

void primsalg() {
  int k, l, min, a, t[10][10], u, i, j, mincost = 0;
  min = 999;
  for (i = 1; i <= n; i++) //To Find the Minimum Edge E(k,l)
  {
    for (u = 1; u <= n; u++) {
      if (i != u) {
        if (cost[i][u] < min) {
          min = cost[i][u];
          k = i;
          l = u;
        }
      }
    }
  }
  t[1][1] = k;
  t[1][2] = l;
  printf("\n The Minimum Cost Spanning tree is...");
  printf("\n(%d,%d)-->%d", k, l, min);

  for (i = 1; i <= n; i++) {
    if (i != k) {
      if (cost[i][l] < cost[i][k]) {
        nears[i] = l;
      } else {
        nears[i] = k;
      }
    }
  }
  nears[k] = nears[l] = 0;
  mincost = min;
  for (i = 2; i <= n - 1; i++) {
    j = findnextindex(cost, nears);
    t[i][1] = j;
    t[i][2] = nears[j];
    printf("\n(%d,%d)-->%d", t[i][1], t[i][2], cost[j][nears[j]]);
    mincost = mincost + cost[j][nears[j]];
    nears[j] = 0;
    for (k = 1; k <= n; k++) {
      if (nears[k] != 0 && cost[k][nears[k]] > cost[k][j]) {
        nears[k] = j;
      }
    }
  }
  printf("\n The Required Mincost of the Spanning Tree is:%d", mincost);
}

int findnextindex(int cost[10][10], int nears[10]) {
  int min = 999, a, k, p;
  for (a = 1; a <= n; a++) {
    p = nears[a];

    if (p != 0) {
      if (cost[a][p] < min) {
        min = cost[a][p];
        k = a;
      }
    }
  }
  return k;
}

void main() {
  clrscr();
  readv();
  primsalg();
  getch();
}
```

Output:



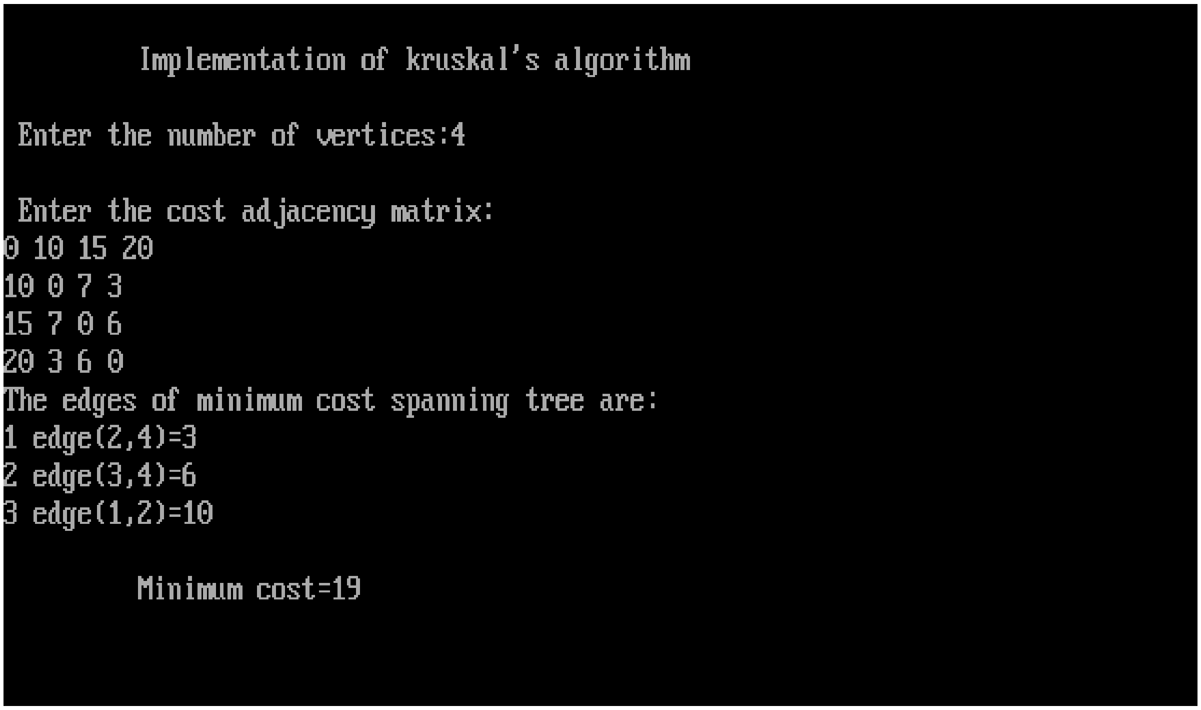
```
 Enter the No of nodes or vertices:4

 Enter the Cost Adjacency matrix of the given graph:
0 10 15 20
10 0 7 3
15 7 0 6
20 3 6 0

 The Minimum Cost Spanning tree is...
(2,4)-->3
(3,4)-->6
(1,2)-->10
 The Required Mincost of the Spanning Tree is:19_
```

Program-18: Write a program to implement kruskal’s algorithm to generate minimum cost spanning tree

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[9][9], parent[9];
int find(int);
int uni(int, int);

void main() {
  clrscr();
  printf("\n\t Implementation of kruskal's algorithm\n");
  printf("\n Enter the number of vertices:");
  scanf("%d", & n);
  printf("\n Enter 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] = 999;
    }
  }
  printf("The edges of minimum cost spanning tree are:\n");
  while (ne < n) {
    for (i = 1, min = 999; i <= n; i++) {
      for (j = 1; j <= n; j++) {
        if (cost[i][j] < min) {
          min = cost[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;
    }
    cost[a][b] = cost[b][a] = 999;
  }
  printf("\n\t Minimum cost=%d\n", mincost);
  getch();
}
```

```
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;
}
```

Output:



```
        Implementation of kruskal's algorithm

 Enter the number of vertices:4

 Enter the cost adjacency matrix:
0 10 15 20
10 0 7 3
15 7 0 6
20 3 6 0
The edges of minimum cost spanning tree are:
1 edge(2,4)=3
2 edge(3,4)=6
3 edge(1,2)=10

        Minimum cost=19
```