REVA UNIVERSITY

Bengaluru, India

SCHOOLOFCOMPUTINGANDINFORMATIONTECHNOLOGY

Design and Analysis of Algorithms Lab (B20CI0404)

For

Fourth Semester

B.TechinCSIT,AIML,ISE, CSSE

INDEX

1.LabObjectives:

Theobjectivesofthiscourseareto:

1. Explain the mathematical foundation for the analysis of algorithms.
2. Illustrate the algorithms using brute force and divide and conquer design technique.
3. Make use of greedy and dynamic algorithmic design techniques for a given problem.
4. Discuss the problems based on backtracking and branch and bound techniques.

2.LabOutcomes:

Onsuccessfulcompletionofthiscourse;studentshallbeableto:

1. Apply the knowledge of mathematical foundation for the analysis of algorithms.
2. Develop a program to solve the given real world problems using brute force and divide and conquer design paradigm.
3. Make use of greedy and dynamic programming techniques for solving the given real world problem.
4. Utilize backtracking and branch and bound techniques to solve real world problems.
5. Learn new tools and technologies in the Desiging of algorithms and apply for suitable application development.
6. Develop solution to the complex problems, either individually or as a part of the team and report the results with proper analysis and interpretation.

3.LabRequirements:
Followingaretherequiredhardwareandsoftwareforthislab,whichisavailableinthelaboratory. **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in theLab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktopsystemsare dual boothaving Windows aswell as LinuxOSinstalled onthem. **Software:** C-compiler. Many C-compilers are available. As we have dual boot, we have *gcc*compilerinLinuxenvironment(fedora8tofedora20),andTurboC-IDEonWindowsEnvironment.

ThisDesign and Analysis of Algorithmlabmanualisdesigned toincreasethe practicalknowledge,thinkingability and creativity of students. Here, students are expected to read, think, design and implement theprogramsforgivenproblemsin"C". TheLinuxandwindowsenvironmentsareusedasexplainedbelow.

Linux environment: All the programs have been written using "C" and tested using the *cc* compiler inLinuxenvironment.

Stepsforthesuccessfulprogramcompletionare:

1. Openvieditorusingtheshell command

$vi<filename.c>

2. WritetheC-codeaccordingtotheprogramlogic.Savetheprogramandexit(Esc+:wq)toget shellcommandprompt.
3. Compiletheprogramusinggcccompilerusingthefollowingshellcommand

$cc<filename.c>

Where,*cc*istheCcompilerand*filename.c*istheinputfilename.

4. Executetheprogramusingthefollowingshellcommand

$./exeFileName

Where,*exeFileName*istheinputfilenam e.

Windows environment: Programs can be edited, compiled and executed using Turbo C/C++ IDE. Alloperations like opening a file, saving a file, compiling the program and executing the program are listedunder different menus inside the IDE. Keyboard shortcuts are also available for the above mentionedoperations.

Note: Every problem in the Lab manual includes problem statement, learning outcomes, theoreticaldescription,algorithm,program,programdescription,expectedresults,implementat ionphase,simulationofsyntax and logical errors,final programwith results.

Recommendation: It is advised to use Linux environment for executing the programs given as part ofthismanual.

4. GuidelinestoStudents

Equipment in the lab for the use of student community. Students need to maintain aproperdecoruminthecomputerlab.Studentsmustusetheequipmentwithcare.Anydamage is causedispunishable.

Students are required to carry their observation / programs book with completedexerciseswhile enteringthe lab.

Students are supposed to occupy the machines allotted to them and are not supposedtotalk ormake noiseinthe lab. Theallocationisput up onthe lab noticeboard.

Lab can be used in free time / lunch hours by the students who need to use thesystemsshould take prior permissionfromthe labin-charge.

Labrecordsneed tobesubmitted onorbefore dateofsubmission.

Studentsarenot supposedtouse flashdrives.

ListofLabExercises

Sl. No	PROBLEM STATEMENT
1	Search for a given pattern in a text string using Brute Force String Matching.
2	Sort a set of elements in ascending order using Quick Sort algorithm.
3	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithms.
4	Find Minimum Cost Spanning Tree of a given undirected graph using Prim'salgorithms.
5	From a given vertex in a weighted connected graph, find shortest paths to other verticesusing Dijikstra's algorithm
6	Design and Implement 0/1 Knapsack problem using Dynamic Programming.
7	Implement All-Pairs Shortest Paths Problem using Floyd's algorithm
8	Obtain the DFS ordering of vertices in a given digraph.
9	Implement Horspool's algorithm for String Matching and find the number ofkeycomparisons in successful search and unsuccessful search
10	Sort a given set of elements in ascending order which has duplicate entries. Use the sortingby counting algorithm
11	Implement N Queen's problem using Back Tracking.

Program 1

Search for a given pattern in a text string using Brute Force String Matching.

```
#include <stdio.h>
#include <string.h>
intmain()
{

  char a[100], b[100];
inti,n,m,j;
printf("Enter some text\n");
  gets(a);
printf("Enter a string to find\n");
  gets(b);
  m=strlen(b);
  n=strlen(a);
  for(i=0;i<=n-m;i++)
  {
    j=0;
while(j<m && b[j]==a[i+j])
    {
    j=j+1;

    }
    if(j==m)
    {
printf("SUBSTRING FOUND AT LOCATION %d\n",i+1);
     return 0;
     }


      }
```

```
    printf("SUBSTRING NOT FOUND\n");
    return 0;
}
```

Program 2
Sort a set of elements in ascending order using Quick Sort algorithm.

```
#include<stdio.h>
void quicksort(int number[25],intfirst,int last){
inti, j, pivot, temp;

  if(first<last){
    pivot=first;
i=first;
    j=last;

    while(i<j){
      while(number[i]<=number[pivot]&&i<last)
i++;
      while(number[j]>number[pivot])
        j--;
      if(i<j){
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
```

```
    }
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);
  }
}
void main()
{
inti, count, number[25];
printf("How many elements are u going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
  for(i=0;i<count;i++)
scanf("%d",&number[i]);
  quicksort(number,0,count-1);
printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
printf(" %d",number[i]);

}
```

OUTPUT:

Enter the number of elements: 5

Enter the elements
45 8 21 3 4

Sorted elements are
3 4 8 21 45

Program 3
Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal' s algorithms.

```
#include <stdio.h>

#define MAX 30

typedef struct edge {
  int u, v, w;
} edge;

typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;

edge_list elist;
```

```
int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();

// Applying Krushkal Algo
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;

  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
      }
    }

  sort();

  for (i = 0; i < n; i++)
    belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
```

```
    }
  }
}

int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}

// Sorting algo
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}

// Printing the result
void print() {
  int i, cost = 0;

  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
  }

  printf("\nSpanning tree cost: %d", cost);
```

```
}

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

  n = 6;

  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0;

  Graph[4][0] = 0;
```

```
Graph[4][1] = 0;
Graph[4][2] = 4;
Graph[4][3] = 3;
Graph[4][4] = 0;
Graph[4][5] = 0;
Graph[4][6] = 0;

Graph[5][0] = 0;
Graph[5][1] = 0;
Graph[5][2] = 2;
Graph[5][3] = 0;
Graph[5][4] = 3;
 Graph[5][5] = 0;
 Graph[5][6] = 0;

 kruskalAlgo();
 print();
}
```

Output:

```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

Program 4

Find Minimum Cost Spanning Tree of a given undirected graph using Prim’ s algorithms

```
#include<stdio.h>
#define INFINITY 999
intprim(int cost[10][10],intsource,int n)
{
        inti,j,sum=0,visited[10];
        int distance[10],vertex[10];
        intmin,u,v;
        for(i=1;i<=n;i++)
       {
        vertex[i]=source;
         visited[i]=0;
         distance[i]=cost[source][i];
       }
       visited[source]=1;
       for(i=1;i<n;i++)
      {
        min=INFINITY;
         for(j=1;j<=n;j++)
         {
```

```
                    if(!visited[j]&&distance[j]<min)
                    {
                            min=distance[j];
                            u=j;
                    }
        }
  visited[u]=1;
  sum=sum+distance[u];
printf("\n%d->%d",vertex[u],u);
  for(v=1;v<=n;v++)
                {
                        if(!visited[v]&&cost[u][v]<distance[v])
                        {
                                distance[v]=cost[u][v];
                                vertex[v]=u;
                        }
          }
    }
    return sum;
}
void main()
{
inta[10][10],n,i,j,m,source;
printf("\n enter the  number of vertices:\n");
scanf("%d",&n);
printf("\nenter the cost matrix:\n 0-self loop and 999-no edge:\n");
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n enter the source:\n");
scanf("%d",&source);
```

```
  m=prim(a,source,n);
printf("\n the cost of spanning tree=%d",m);
  }
```

OUTPUT:

Enter the number of vertices: 5

Enter the cost matrix 0-for self edge and 999-if no edge
0 3 4 999 5
3 0 999 6 1
4 999 0 9 7
999 6 9 0 2
5 1 7 2 0

Enter the source
2

2->5
5->4
2->1
1->3

Cost = 10

Program 5

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

```
#include<stdio.h>
#define INFINITY 999
void dijkstra(int cost[10][10],intn,intsource,int distance[10])
{
    intvisited[10],min,u;
    inti,j;
    for(i=1;i<=n;i++)
```

```
{
        distance[i]=cost[source][i];
        visited[i]=0;
}
visited[source]=1;
for(i=1;i<=n;i++)
{
        min=INFINITY;
        for(j=1;j<=n;j++)
        if(visited[j]==0 && distance[j]<min)
        {
                min=distance[j];
                u=j;
        }
        visited[u]=1;
        for(j=1;j<=n;j++)
if(visited[j]==0 && (distance[u]+cost[u][j])<distance[j])
        {
                distance[j]=distance[u]+cost[u][j];
        }
    }
}
void main()
{
    intn,cost[10][10],distance[10];
    inti,j,source,sum;
    printf("\nEnter how many nodes :  ");
    scanf("%d",&n);
    printf("\nCost Matrix\nEnter 999 for no edge\n");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
```

```
        scanf("%d",&cost[i][j]);
        printf("Enter the source node\n");
        scanf("%d",&source);
        dijkstra(cost,n,source,distance);
        for(i=1;i<=n;i++)
        printf("\n\nShortest Distance from  %d to %d is %d",source,i,distance[i]);
    }
```

OUTPUT:

```
Enter  how many nodes:4

Cost Matrix
 Enter 999 for no edge

999    999    3      999
999    999    4      7
999    4      999    15
999    7      15     999

Enter the source node
1

Shortest Distance for 1 to 1 is 999

Shortest Distance for 1 to 2 is 7

Shortest Distance for 1 to 3 is 3

Shortest Distance for 1 to 4 is 14
```

Program 6

Implement 0 / 1 Knapsack problem using dynamic programming.

```
#include<stdio.h>
int w[10],p[10],n;
intmax(inta,int b)
{
        return a>b?a:b;
}
intknap(inti,int m)
{
        if(i==n)     return w[i]>m?0:p[i];
        if (w[i]>m)   return knap(i+1,m);
        return     max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}
void main()
{
        intm,i,max_profit;
        printf("\nEnter the number of objects: ");
        scanf("%d",&n);
        printf("\nEnter the knapsack capacity: ");
        scanf("%d",&m);
        printf("\nEnter profit followed by weight: ");
        for(i=1;i<=n;i++)
        scanf("%d%d",&p[i],&w[i]);
        max_profit=knap(1,m);
        printf("\nMax profit = %d",max_profit);
        }
```

OUTPUT:

```
Enter the number of objects: 3

Enter the knapsack capacity:116
```

Enter the profit followed by weight:
100 12
12 15
20 30
Max profit=132

Program 7

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

```
#include<stdio.h>
#include<omp.h>
#define INFINITY 999
intmin(inti,int j)
{
	if(i<j)
		return i;
	else
		return j;
}
void floyd(intn,int p[10][10])
{
	inti,j,k;
	#pragma omp parallel for private(i,j,k) shared(p)
	 for(k=1;k<=n;k++)
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
intmain()
{
	inti,j,n,a[10][10],d[10][10],source;
	double starttime,end time;
```

```
printf("Enter the no.of nodes: ");
scanf("%d",&n);
printf("\nEnter the adjacency matrix\n");
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                scanf("%d",&a[i][j]);
starttime=omp_get_wtime();
floyd(n,a);
endtime=omp_get_wtime();
printf("\n\nThe distance matrix is \n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf("%d\t",a[i][j]);
                printf("\n");
        }
        printf("\n\nThe time taken is %l0.9f\n",(double)(starttime-endtime));
  return 0;
  }
```

OUTPUT:

```
Enter the no of nodes: 5
Enter the adjancy matrix
0       15      8       10      999
15      0       4       999     999
8       4       0       999     12
10      999     999     0       7
999 999         12      7        0

The distance matrix is
0       12      8       10      17
12      0       4       22      16
8       4       0       18      12
```

10 22 18 0 7
17 16 12 7 0

The time taken is: 0.001107683

Program 8
Obtain the DFS ordering of vertices in a given digraph.

```
#include<stdio.h>
inti,visit[20],n,adj[20][20],s,topo_order[10];

void dfs(int v)
{
int w;
 visit[v]=1;
 for(w=1;w<=n;w++)
   if((adj[v][w]==1) && (visit[w]==0))
dfs(w);
topo_order[i--]=v;
}

void main()
```

```
{
intv,w;
printf("Enter the number of vertices:\n");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
  for(v=1;v<=n;v++)
    for(w=1;w<=n;w++)
scanf("%d",&adj[v][w]);
  for(v=1;v<=n;v++)
      visit[v]=0;
    i=n;
     for(v=1;v<=n;v++)
     {
      if(visit[v]==0)
    dfs(v);
     }
    printf("\nTopological sorting is:");
     for(v=1;v<=n;v++)
    printf("v%d ",topo_order[v]);
     }
```

OUTPUT 1 :

```
Enter the number of vertices:
5
Enter the adjacency matrix
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
```

Topological ordering is v1 v2 v3 v4 v5

OUTPUT 2 :

enter the number of vertices:

3

Enter the adjacency matrix

0 1 0

0 0 1

1 0 0

Topological ordering is v1 v2 v3

Program 9:

Implement Horspool' s algorithm for String Matching and find the number ofkey comparisons in successful search and unsuccessful search

```
#include<stdio.h>
void main()
{
```

```
inttable[126];
char t[100],p[25];
intn,i,k,j,m,flag=0;
printf(“Enter the text : “);
gets(t);
n=strlen(t);
printf(“Enter the pattern : “);
gets(p);
m=strlen(p);
for(i=0;i<126;i++)
table[i]=m;
for(j=0;j<m-2;j++)
table[p[j]]=m-1-j;
i=m-1;
while(i<=n-1)
{
k=0;
while(k<=m-1 && p[m-1-k]==t[i-k])
k++;
if(k==m)
{
printf(“The position of the pattern is %dn”,i-m+2);
flag=1;
break;
}
else
i=i+table[t[i]];
}
if(!flag)
printf(“Pattern is not found in the given text “);
}
```

OUTPUT:

Enter the text in which pattern is to be searched:
god is great
Enter the pattern to be searched:
great

Length of text=12 Length of

pattern=5

The desired pattern was found starting from position 8

Enter the text in which pattern is to be searched:
god is great
Enter the pattern to be searched:
king

Length of text=12 Length of

pattern=4

The pattern was not found in the given text

Program 10.

Sort a given set of elements in ascending order which has duplicate entries. Use the sorting by counting algorithm

```
#include  <stdio.h>
  /*  Counting sort function  */
void  counting_sort(int a[], int k, int n)
  {
inti, j;
intb[15], c[100];
  for (i = 0; i<= k; i++)
  c[i] = 0;
  for (j = 1; j <= n; j++)
  c[a[j]] = c[a[j]] + 1;
  for (i = 1; i<= k; i++)
  c[i] = c[i] + c[i-1];
  for (j = n; j >= 1; j--)
  {
  b[c[a[j]]] = a[j];
  c[a[j]] = c[a[j]] - 1;
  }
printf("The Sorted array is : ");
  for (i = 1; i<= n; i++)
printf("%d,", b[i]);
  }

void  main()
```

```
  {
int n, k = 0, a[15], i;
printf("Input number of elements:  ");
scanf("%d", &n);
printf("Input the array elements one  by one: \n");
  for (i = 1; i<= n; i++)
  {
scanf("%d", &a[i]);
  if (a[i] > k) {
  k = a[i];
     }
     }
   counting_sort(a, k, n);
   printf("\n");
     }
```

OUTPUT:

Input number of elemets

5

Input the array elements one by one:
15
12
01
13
11

Output:

```
Input number of elements:  Input the array elements one  by one:
The Sorted array is :

15,
12
```

01
13
11

Program 11
Implement N Queen’ s problem using back tracking.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50
intcan_place(int c[],int r)
{
        inti;
        for(i=0;i<r;i++)
                if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))
                return 0;
        return 1;
}
void display(int c[],int n)
{
        inti,j;
        char cb[10][10];
        for(i=0;i<n;i++)
```

```
for(j=0;j<n;j++)
cb[i][j]='-';
for(i=0;i<n;i++)
cb[i][c[i]]='Q';
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        printf("%c",cb[i][j]);
        printf("\n");
 }
}
void n_queens(int n)
{
int r;
int c[MAX];
c[0]= -1;
r=0;
while(r>=0)
{
                c[r]++;
                 while(c[r]<n && !can_place(c,r))
                        c[r]++;
if(c[r]<n)
{
        if(r==n-1)
        {
                display(c,n);
                printf("\n");
         }
        else
        {
```

```
                        r++;
                        c[r]=-1;
                    }
                }
                else
                    r--;
        }
}
void main()
{
        int n;
        printf("\nEnter the number of queens : ");
        scanf("%d",&n);
        n_queens(n);
        }
```

OUTPUT:

Enter the number of queens;
4

```
- Q - -
- - - Q
Q - - -
- - Q -
```

```
- - Q -
Q - - -
- - - Q
- Q - -
```

```
for (i = 0; i < elist.n; i++) {
  cno1 = find(belongs, elist.data[i].u);
  cno2 = find(belongs, elist.data[i].v);

  if (cno1 != cno2) {
```

```
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}

int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}

// Sorting algo
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}

// Printing the result
void print() {
  int i, cost = 0;

  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
```

```
  }

  printf("\nSpanning tree cost: %d", cost);
}

int main()
{
  int i, j, total_cost;
  n = 6;
  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0;
```

```
Graph[4][0] = 0;
Graph[4][1] = 0;
Graph[4][2] = 4;
Graph[4][3] = 3;
Graph[4][4] = 0;
Graph[4][5] = 0;
Graph[4][6] = 0;

Graph[5][0] = 0;
Graph[5][1] = 0;
Graph[5][2] = 2;
Graph[5][3] = 0;
Graph[5][4] = 3;
Graph[5][5] = 0;
Graph[5][6] = 0;

kruskalAlgo();
print();
}
```