**Experiment No. 6**

Implement in Java, the **0/1 Knapsack** problem using

(a) Dynamic Programming method

(b)Greedy method.

(a) Dynamic Programming method

**import** java.util.Scanner;

**public class** knapsackDP {

/\*\*

\* **@param** args

\*/

**public void** solve(**int**[] wt, **int**[] val, **int** W, **int** N)

{

**int** i,j;

**int**[][] sol = **new int**[N + 1][W + 1];

**for** ( i = 0; i <= N; i++)

{

**for** ( j = 0; j <= W; j++)

{

**if**(i==0||j==0)

sol[i][j]=0;

**else if**(wt[i]>j)

sol[i][j]=sol[i-1][j];

**else**

sol[i][j]=Math.*max*((sol[i-

1][j]), (sol[i - 1][j - wt[i]] + val[i]));

}

}

System.*out*.println("The optimal solution

is"+sol[N][W]);

**int**[] selected = **new int**[N + 1];

**for**(i=0;i<N+1;i++)

selected[i]=0;

i=N;

j=W;

**while** (i>0&&j>0)

{

**if** (sol[i][j] !=sol[i-1][j])

{

selected[i] = 1;

j = j - wt[i];

}

i--;

}

System.*out*.println("\nItems selected : ");

**for** ( i = 1; i < N + 1; i++)

**if** (selected[i] == 1)

System.*out*.print(i +" ");

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.21**

System.*out*.println();

}

**public static void** main(String[] args) {

Scanner scan = **new** Scanner(System.*in*);

knapsackDP ks = **new** knapsackDP();

System.*out*.println("Enter number of elements ");

**int** n = scan.nextInt();

**int**[] wt = **new int**[n + 1];

**int**[] val = **new int**[n + 1];

System.*out*.println("\nEnter weight for "+ n +"

elements");

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

wt[i] = scan.nextInt();

System.*out*.println("\nEnter value for "+ n +"

elements");

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

val[i] = scan.nextInt();

System.*out*.println("\nEnter knapsack weight ");

**int** W = scan.nextInt();

ks.solve(wt, val, W, n);

}

}

**Output:**

Enter number of elements

4

Enter weight for 4 elements

2132

Enter value for 4 elements

12

10

20

15

Enter knapsack weight

5

The optimal solution is37

Items selected :

1 2 4

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.22**

(b)Greedy method.

**import** java.util.Scanner;

**public class** knapsacgreedy {

/\*\*

\* **@param** args

\*/

**public static void** main(String[] args) {

**int** i,j=0,max\_qty,m,n;

**float** sum=0,max;

Scanner sc = **new** Scanner(System.*in*);

**int** array[][]=**new int**[2][20];

System.*out*.println("Enter no of items");

n=sc.nextInt();

System.*out*.println("Enter the weights of each

items");

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

array[0][i]=sc.nextInt();

System.*out*.println("Enter the values of each

items");

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

array[1][i]=sc.nextInt();

System.*out*.println("Enter maximum volume of

knapsack :");

max\_qty=sc.nextInt();

m=max\_qty;

**while**(m>=0)

{

max=0;

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

{

**if**(((**float**)array[1][i])/((**float**)array[0][i])>max)

{

max=((**float**)array[1][i])/((**float**)array[0][i]);

j=i;

}

}

**if**(array[0][j]>m)

{

System.*out*.println("Quantity of item number: "

+ (j+1) + " added is " +m);

sum+=m\*max;

m=-1;

}

**else**

{

System.*out*.println("Quantity of item

number: " + (j+1) + " added is " + array[0][j]);

m-=array[0][j];

sum+=(**float**)array[1][j];

array[1][j]=0;

}

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.23**

}

System.*out*.println("The total profit is " + sum);

sc.close();

}

}

Output:

Enter no of items

4

Enter the weights of each items

2132

Enter the values of each items

12

10

20

15

Enter maximum volume of knapsack :

5

Quantity of item number: 2 added is 1

Quantity of item number: 4 added is 2

Quantity of item number: 3 added is 2

The total profit is 38.333332

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.24**

**Experiment No. 7**

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

**Dijkstra's algorithm**. Write the program in Java.

**import** java.util.Scanner;

**public class** Dijkstra {

/\*\*

\* **@param** args

\*/

**int** d[]=**new int**[10];

**int** p[]=**new int**[10];

**int** visited[]=**new int**[10];

**public void** dijk(**int**[][]a, **int** s, **int** n)

{

**int** u=-1,v,i,j,min;

**for**(v=0;v<n;v++)

{

d[v]=99;

p[v]=-1;

} d[s]=0;

**for**(i=0;i<n;i++){

min=99;

**for**(j=0;j<n;j++){

**if**(d[j]<min&& visited[j]==0)

{

min=d[j];

u=j;

}}

visited[u]=1;

**for**(v=0;v<n;v++){

**if**((d[u]+a[u][v]<d[v])&&(u!=v)&&visited[v]==0)

{

d[v]=d[u]+a[u][v];

p[v]=u;

}

}

}

}

**void** path(**int** v,**int** s)

{

**if**(p[v]!=-1)

path(p[v],s);

**if**(v!=s)

System.*out*.print("->"+v+" ");

}

**void** display(**int** s,**int** n){

**int** i;

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

{

**if**(i!=s){

System.*out*.print(s+" ");

path(i,s);

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.25**

}

**if**(i!=s)

System.*out*.print("="+d[i]+" ");

System.*out*.println();

}

}

**public static void** main(String[] args) {

**int** a[][]=**new int**[10][10];

**int** i,j,n,s;

System.*out*.println("enter the number of vertices");

Scanner sc = **new** Scanner(System.*in*);

n=sc.nextInt();

System.*out*.println("enter the weighted matrix");

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

**for**(j=0;j<n;j++)

a[i][j]=sc.nextInt();

System.*out*.println("enter the source vertex");

s=sc.nextInt();

Dijkstra tr=**new** Dijkstra();

tr.dijk(a,s,n);

System.*out*.println("the shortest path between source"+s+"to remaining

vertices are");

tr.display(s,n);

sc.close();

}

}

**Output:**

enter the number of vertices

5

enter the weighted matrix

0 3 99 7 99

3 0 4 2 99

99 4 0 5 6

5 2 5 0 4

99 99 6 4 0

enter the source vertex

0

the shortest path between source0to remaining vertices are

0 ->1 =3

0 ->1 ->2 =7

0 ->1 ->3 =5

0 ->1 ->3 ->4 =9

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.26**

**Experiment No. 8**

. Find Minimum Cost Spanning Tree of a given undirected graph using

(a) **Kruskal's algorithm**

(b) **Prim's algorithm**. Implement the program in Java language.

(a) **Kruskal's algorithm**

**import** java.util.Scanner;

**public class** kruskal {

**int** parent[]=**new int**[10];

**int** find(**int** m)

{

**int** p=m;

**while**(parent[p]!=0)

p=parent[p];

**return** p;

}

**void** union(**int** i,**int** j)

{

**if**(i<j)

parent[i]=j;

**else**

parent[j]=i;

}

**void** krkl(**int**[][]a, **int** n)

{

**int** u=0,v=0,min,k=0,i,j,sum=0;

**while**(k<n-1)

{

min=99;

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

**if**(a[i][j]<min&&i!=j)

{

min=a[i][j];

u=i;

v=j;

}

i=find(u);

j=find(v);

**if**(i!=j)

{

union(i,j);

System.*out*.println("("+u+","+v+")"+"="+a[u][v]);

sum=sum+a[u][v];

k++;

}

a[u][v]=a[v][u]=99;

}

System.*out*.println("The cost of minimum spanning tree = "+sum);

}

**public static void** main(String[] args) {

**int** a[][]=**new int**[10][10];

**int** i,j;

System.*out*.println("Enter the number of vertices of the graph");

Scanner sc=**new** Scanner(System.*in*);

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.27**

**int** n;

n=sc.nextInt();

System.*out*.println("Enter the wieghted matrix");

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

a[i][j]=sc.nextInt();

kruskal k=**new** kruskal();

k.krkl(a,n);

sc.close();

}

}

**Output:**

Enter the number of vertices of the graph

6

Enter the wieghted matrix

0 3 99 99 6 5

3 0 1 99 99 4

99 1 0 6 99 4

99 99 6 0 8 5

6 99 99 8 0 2

5 4 4 5 2 0

(2,3)=1

(5,6)=2

(1,2)=3

(2,6)=4

(4,6)=5

The cost of minimum spanning tree = 15

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.28**

(b) **Prim's algorithm**. Implement the program in Java language.

**import** java.util.Scanner;

**public class** prims {

**public static void** main(String[] args) {

**int** w[][]=**new int**[10][10];

**int** n,i,j,s,k=0;

**int** min;

**int** sum=0;

**int** u=0,v=0;

**int** flag=0;

**int** sol[]=**new int**[10];

System.*out*.println("Enter the number of vertices");

Scanner sc=**new** Scanner(System.*in*);

n=sc.nextInt();

**for**(i=1;i<=n;i++)

sol[i]=0;

System.*out*.println("Enter the weighted graph");

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

w[i][j]=sc.nextInt();

System.*out*.println("Enter the source vertex");

s=sc.nextInt();

sol[s]=1;

k=1;

**while** (k<=n-1)

{

min=99;

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

**if**(sol[i]==1&&sol[j]==0)

**if**(i!=j&&min>w[i][j])

{

min=w[i][j];

u=i;

v=j;

}

sol[v]=1;

sum=sum+min;

k++;

System.*out*.println(u+"->"+v+"="+min);

}

**for**(i=1;i<=n;i++)

**if**(sol[i]==0)

flag=1;

**if**(flag==1)

System.*out*.println("No spanning tree");

**else**

System.*out*.println("The cost of minimum spanning tree is"+sum);

sc.close();

}

}

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.29**

**Output:**

Enter the number of vertices

6

Enter the weighted graph

0 3 99 99 6 5

3 0 1 99 99 4

99 1 0 6 99 4

99 99 6 0 8 5

6 99 99 8 0 2

5 4 4 5 2 0

Enter the source vertex

1

1->2=3

2->3=1

2->6=4

6->5=2

6->4=5

The cost of minimum spanning tree is15

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.30**

**Experiment No. 9**

Write Java programs to

(a) Implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.

(b) Implement **Travelling Sales Person problem** using Dynamic programming.

**Floyd's algorithm**:

**import** java.util.Scanner;

**public class** floyd {

**void** flyd(**int**[][] w,**int** n)

{

**int** i,j,k;

**for**(k=1;k<=n;k++)

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

w[i][j]=Math.*min*(w[i][j], w[i][k]+w[k][j]);

}

**public static void** main(String[] args) {

**int** a[][]=**new int**[10][10];

**int** n,i,j;

System.*out*.println("enter the number of vertices");

Scanner sc=**new** Scanner(System.*in*);

n=sc.nextInt();

System.*out*.println("Enter the weighted matrix");

**for**(i=1;i<=n;i++)

**for**(j=1;j<=n;j++)

a[i][j]=sc.nextInt();

floyd f=**new** floyd();

f.flyd(a, n);

System.*out*.println("The shortest path matrix is");

**for**(i=1;i<=n;i++)

{

**for**(j=1;j<=n;j++)

{

System.*out*.print(a[i][j]+" ");

}

System.*out*.println();

}

sc.close();

}

}

**Output:**

enter the number of vertices

4

Enter the weighted matrix

0 99 3 99

2 0 99 99

99 7 0 1

6 99 99 0

The shortest path matrix is

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.31**

**Travelling Sales Person problem** using Dynamic programming:

**import** java.util.Scanner;

**class** TSPExp {

**int** weight[][],n,tour[],finalCost;

**final int** INF=1000;

TSPExp()

{

Scanner s=**new** Scanner(System.*in*);

System.*out*.println("Enter no. of nodes:=>");

n=s.nextInt();

weight=**new int**[n][n];

tour=**new int**[n-1];

**for**(**int** i=0;i<n;i++)

{

**for**(**int** j=0;j<n;j++)

{

**if**(i!=j)

{

System.*out*.print("Enter weight of

"+(i+1)+" to "+(j+1)+":=>");

weight[i][j]=s.nextInt();

}

}

}

System.*out*.println();

System.*out*.println("Starting node assumed to be node 1.");

eval();

}

**public int** COST(**int** currentNode,**int** inputSet[],**int** setSize)

{

**if**(setSize==0)

**return** weight[currentNode][0];

**int** min=INF;

**int** setToBePassedOnToNextCallOfCOST[]=**new int**[n-1];

**for**(**int** i=0;i<setSize;i++)

{

**int** k=0;//initialise new set

**for**(**int** j=0;j<setSize;j++)

{

**if**(inputSet[i]!=inputSet[j])

setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];

}

**int**

temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);

**if**((weight[currentNode][inputSet[i]]+temp) <

min)

{

min=weight[currentNode][inputSet[i]]+temp;

}

}

**return** min;

}

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.32**

**public int** MIN(**int** currentNode,**int** inputSet[],**int** setSize)

{

**if**(setSize==0)

**return** weight[currentNode][0];

**int** min=INF,minindex=0;

**int** setToBePassedOnToNextCallOfCOST[]=**new int**[n-1];

**for**(**int** i=0;i<setSize;i++)//considers each node of inputSet

{

**int** k=0;

**for**(**int** j=0;j<setSize;j++)

{

**if**(inputSet[i]!=inputSet[j])

setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];

}

**int**

temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);

**if**((weight[currentNode][inputSet[i]]+temp) < min)

{

min=weight[currentNode][inputSet[i]]+temp;

minindex=inputSet[i];

}

}

**return** minindex;

}

**public void** eval()

{

**int** dummySet[]=**new int**[n-1];

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

dummySet[i-1]=i;

finalCost=COST(0,dummySet,n-1);

constructTour();

}

**public void** constructTour()

{

**int** previousSet[]=**new int**[n-1];

**int** nextSet[]=**new int**[n-2]; **for**(**int** i=1;i<n;i++)

previousSet[i-1]=i;

**int** setSize=n-1;

tour[0]=MIN(0,previousSet,setSize);

**for**(**int** i=1;i<n-1;i++)

{

**int** k=0;

**for**(**int** j=0;j<setSize;j++)

{

**if**(tour[i-1]!=previousSet[j])

nextSet[k++]=previousSet[j];

}

--setSize;

tour[i]=MIN(tour[i-1],nextSet,setSize);

**for**(**int** j=0;j<setSize;j++)

previousSet[j]=nextSet[j];

}

display();

}

**public void** display()

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.33**

{

System.*out*.println();

System.*out*.print("The tour is 1-");

**for**(**int** i=0;i<n-1;i++)

System.*out*.print((tour[i]+1)+"-");

System.*out*.print("1");

System.*out*.println();

System.*out*.println("The final cost is "+finalCost);

}

}

**class** TSP

{

**public static void** main(String args[])

{

TSPExp obj=**new** TSPExp();

}

}

Output:

Enter no. of nodes:=>

4

Enter weight of 1 to 2:=>2

Enter weight of 1 to 3:=>5

Enter weight of 1 to 4:=>7

Enter weight of 2 to 1:=>2

Enter weight of 2 to 3:=>8

Enter weight of 2 to 4:=>3

Enter weight of 3 to 1:=>5

Enter weight of 3 to 2:=>8

Enter weight of 3 to 4:=>1

Enter weight of 4 to 1:=>7

Enter weight of 4 to 2:=>3

Enter weight of 4 to 3:=>1

Starting node assumed to be node 1.

The tour is 1-2-4-3-1

The final cost is 11

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.34**

**Experiment No. 10a**

Design and implement in Java to find a **subset** of a given set **S** = {Sl, S2,.....,Sn} of ***n***

positive integers whose SUM is equal to a given positive integer ***d***. For example, if S ={1, 2,

5,6, 8} and ***d***= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the

given problem instance doesn't have a solution.

import java.util.Scanner;

import static java.lang.Math.pow;

public class subSet {

/\*\*

\* @param args

\*/

void subset(int num,int n, int x[])

{

int i;

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

x[i]=0;

for(i=n;num!=0;i--)

{

x[i]=num%2;

num=num/2;

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

int a[]=new int[10];

int x[]=new int[10];

int n,d,sum,present=0;

int j;

System.out.println("enter the number of elements of set");

Scanner sc=new Scanner(System.in);

n=sc.nextInt();

System.out.println("enter the elements of set");

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

a[i]=sc.nextInt();

System.out.println("enter the positive integer sum");

d=sc.nextInt();

if(d>0)

{

for(int i=1;i<=Math.pow(2,n)-1;i++)

{

subSet s=new subSet();

s.subset(i,n,x);

sum=0;

for(j=1;j<=n;j++)

if(x[j]==1)

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.35**

sum=sum+a[j];

if(d==sum)

{

System.out.print("Subset={");

present=1;

for(j=1;j<=n;j++)

if(x[j]==1)

System.out.print(a[j]+",");

System.out.print("}="+d);

System.out.println();

}

}

}

if(present==0)

System.out.println("Solution does not exists");

}

}

**Output:**

enter the number of elements of set

5

enter the elements of set

1 2 5 6 8

enter the positive integer sum

9

Subset={1,8,}=9

Subset={1,2,6,}=9

**Experiment No. 10b**

Design and implement the presence of **Hamiltonian Cycle** in an undirected Graph **G** of ***n***

vertices.

**import** java.util.\*;

**class** Hamiltoniancycle

{

**private int** adj[][],x[],n;

**public** Hamiltoniancycle()

{

Scanner src = **new** Scanner(System.*in*);

System.*out*.println("Enter the number of nodes");

n=src.nextInt();

x=**new int**[n];

x[0]=0;

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

x[i]=-1;

adj=**new int**[n][n];

System.*out*.println("Enter the adjacency matrix");

**for** (**int** i=0;i<n; i++)

**for** (**int** j=0; j<n; j++)

adj[i][j]=src.nextInt();

}

**public void** nextValue (**int** k)

{

**int** i=0;

**while**(**true**)

{ x[k]=x[k]+1;

**if** (x[k]==n)

x[k]=-1;

**if** (x[k]==-1)

**return**;

**if** (adj[x[k-1]][x[k]]==1)

**for** (i=0; i<k; i++)

**if** (x[i]==x[k])

**break**;

**if** (i==k)

**if** (k<n-1 || k==n-1 && adj[x[n-1]][0]==1)

**return**;

}}

**public void** getHCycle(**int** k)

{

**while**(**true**)

{

nextValue(k);

**if** (x[k]==-1)

**return**;

**if** (k==n-1)

{

System.*out*.println("\nSolution : ");

**for** (**int** i=0; i<n; i++)

System.*out*.print((x[i]+1)+" ");

System.*out*.println(1);

}

**else** getHCycle(k+1);

**15CSL47-Algorithms Lab IV Sem CSE**

**Dept. of CSE, CIT, Gubbi- 572 216 Page No.37**

}}

}

**class** HamiltoniancycleExp

{

**public static void** main(String args[])

{

Hamiltoniancycle obj=**new** Hamiltoniancycle();

obj.getHCycle(1);

}

}

**Output:**

Enter the number of nodes

6

Enter the adjacency matrix

0 1 1 1 0 0

1 0 1 0 0 1

1 1 0 1 1 0

1 0 1 0 1 0

0 0 1 1 0 1

0 1 0 0 1 0

Solution :

1 2 6 5 3 4 1

Solution :

1 2 6 5 4 3 1

Solution :

1 3 2 6 5 4 1

Solution :

1 3 4 5 6 2 1

Solution :

1 4 3 5 6 2 1

Solution :

1 4 5 6 2 3 1