**ALGORITHM:** floyds(a[1….n,1….n])

//Implements Floyd’s algorithm for all-pairs shortest path problem

//Input: cost matrix a[1….n,1….n] of size nXn

//Output: Shortest distance matrix a[1….n,1….n] of size nXn

**for** k🡨1 to n **do**

**for** i🡨1 to n **do**

**for** j🡨1 to n **do**

a[i,j]🡨min(a[i,j],a[i,k]+a[k,j])

**end for**

**end for**

**end for**

write ‘all pair shortest path matrix is’

**for** i🡨1 to n **do**

**for** j🡨1 to n **do**

write a[i,j]

**end for**

**end for**

**Program:**

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void floyds();

int min(int,int);

void main()

{

int i,j;

clrscr();

printf("\nenter the no. of vertices:\t");

scanf("%d",&n);

printf("\nenter the cost matrix:\n");

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

{

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

{

scanf("%d",&a[i][j]);

}

}

floyds();

getch();

}

void floyds()

{

int i,j,k;

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

{

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

{

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

{

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

printf("\nall pair shortest path matrix is:\n");

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

{

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

{

printf("%d\t",a[i][j]);

}

printf("\n\n");

}

}

int min(int x,int y)

{

if(x<y)

{

return x;

}

else

{

return y;

}

}

**Output**

Enter the no. of vertices: 4

Enter the cost matrix:

9999 9999 3 9999

2 9999 9999 9999

9999 7 9999 1

6 9999 9999 9999

All pair shortest path matrix is:

10 10 3 4

2 12 5 6

7 7 10 1

6 16 9 10

**AIM: Implement O/I Knapsack problem using dynamic programming.**

**ALGORITHM :** knapsack(w[1…n],p[1…n],n,m)

**//**To find the optimal solution for the Knapsack problem using dynamic programming

**//** Input: n-number of objects to be selected

**//** m-maximum capacity of the Knapsack

**//**  An array w[1….n] contains weights of all objects

**//**  An array p[1….n] contains profits of all objects

**//** Output :A matrix v[0….n,0….m] contains the optimal solution for the number of objects selected with

// specified remaining capacity

**for** i🡨0 to n **do**

**for** j🡨0 to m **do**

**if** i=0 **or** j=0

v[i,j]=0

**else if** j-w[i]<0

v[i,j]=v[i-1,j]

**else**

v[i,j]=max(v[i-1,j],v[i-1,j-w[i]+p[i])

**end if**

**end for**

**end for**

write ‘the output is’

**for** i🡨0 to n **do**

**for** j🡨0 to m **do**

write v[i,j]

**end for**

**end for**

write ‘the optimal solution is’,v[n,m]

write ‘solution vector is’

**for** i🡨n downto 1 **do**

**if**  v[i,m]**!=**v[i-1,m]

x[i]🡨1

m🡨m-w[i]

**else**

x[i]🡨0

**end if**

**end for**

**for** i🡨1 to n **do**

write x[i]

**end for**

**return**

**Program:**

#include<stdio.h>

#include<conio.h>

void knapsack();

int max(int,int);

int i,j,n,m,p[10],w[10],v[10][10];

void main()

{

clrscr();

printf("\nenter the no. of items:\t");

scanf("%d",&n);

printf("\nenter the weight of the each item:\n");

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

{

scanf("%d",&w[i]);

}

printf("\nenter the profit of each item:\n");

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

{

scanf("%d",&p[i]);

}

printf("\nenter the knapsack's capacity:\t");

scanf("%d",&m);

knapsack();

getch();

}

void knapsack()

{

int x[10];

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

{

for(j=0;j<=m;j++)

{

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

{

v[i][j]=0;

}

else if(j-w[i]<0)

{

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

}

else

{

v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);

}

}

}

printf("\nthe output is:\n");

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

{

for(j=0;j<=m;j++)

{

printf("%d\t",v[i][j]);

}

printf("\n\n");

}

printf("\nthe optimal solution is %d",v[n][m]);

printf("\nthe solution vector is:\n");

for(i=n;i>=1;i--)

{

if(v[i][m]!=v[i-1][m])

{

x[i]=1;

m=m-w[i];

}

else

{

x[i]=0;

}

}

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

{

printf("%d\t",x[i]);

}

}

int max(int x,int y)

{

if(x>y)

{

return x;

}

else

{

return y;

}

}

**Output:**

Enter the no. of items: 4

Enter the weight of each item:

2 1 3 2

Enter the profit of the each item:

12 10 20 15

Enter the Knapsack’s capacity: 5

The output is:

0 0 0 0 0 0

0 0 12 12 12 12

0 10 12 22 22 22

0 10 12 22 30 32

0 10 15 25 30 37

The optimal solution is: 37

The solution vector is:

1 1 0 1