**11.) WAP TO IMPLEMENT KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING**

#include <iostream>

using namespace std;

int max(int a, int b)

{

return (a > b) ? a : b;

}

int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int K[n + 1][W + 1];

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

{

for (w = 0; w <= W; w++)

{

if (i == 0 || w == 0)

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w]= max(val[i-1]+K[i - 1][w - wt[i - 1]],K[i-1][w]);

else

K[i][w] = K[i - 1][w];

}

}

return K[n][W];

}

int main()

{

cout << "Enter the number of items in a Knapsack:";

int n, W;

cin >> n;

int val[n], wt[n];

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

{

cout << "Enter value and weight for item " << i << ":";

cin >> val[i];

cin >> wt[i];

}

cout << "Enter the capacity of knapsack";

cin >> W;

cout << knapSack(W, wt, val, n);

return 0;

}

**OUTPUT**

Enter the number of items in a Knapsack:5

Enter value and weight for item 0:11 111

Enter value and weight for item 1:22 121

Enter value and weight for item 2:33 131

Enter value and weight for item 3:44 141

Enter value and weight for item 4:55 151

Enter the capacity of knapsack 300

99

12.)WAP TO SOLVE ALL PAIR SHORTEST PROBLEMUSING DYNAMIC PROGRAMMING

#include<iostream>

#include<conio.h>

using namespace std;

int min(int a,int b);

int cost[10][10],a[10][10],i,j,k,c;

main()

{

int n,m;

cout <<"enter no of vertices";

cin >> n;

cout <<"enter no od edges";

cin >> m;

cout<<"enter the**\n**EDGE Cost**\n**";

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

{

cin>>i>>j>>c;

a[i][j]=cost[i][j]=c;

}

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

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

{

if(a[i][j]== 0 && i !=j)

a[i][j]=31999;

}

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]);

cout <<"Resultant adj matrix**\n**";

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

{

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

{

if(a[i][j] !=31999)

cout << a[i][j] <<" ";

}

cout <<"**\n**";

}

getch();

}

int min(int a,int b)

{

if(a<b)

return a;

else

return b;

}

OUTPUT

enter no of vertices3  
enter no od edges5  
enter the  
EDGE Cost  
1 2 4  
2 1 6  
1 3 11  
3 1 3  
2 3 2  
Resultant adj matrix  
0 4 6  
5 0 2  
3 7 0