

|  |
| --- |
| **Title: Implementation of Single source shortest path algorithm using Dynamic Programming** |



**Objective** To learn the Dynamic Programming using Single source shortest path algorithm 

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |



**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **en.wikipedia.org/wiki/Knapsack\_problem**
4. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
5. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.ppt**
6. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
7. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.ppt**



**Topic: DYNAMIC PROGRAMMING**

**Theory:** Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions. For some of the problems that may be viewed in this way ,an optimal sequence of decisions can be found by making the decisions one at a time and never making an erroneous decision .This is true for all problem solvable by the greedy method. For many other problems, it is not possible to make stepwise decisions in such a manner that the sequence of decisions made is optimal.

**Principle of Optimality:** It states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis



**Historical Profile:**

Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

**Principle of Optimality:** It states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

The solution to the knapsack problem can be viewed as the result of a sequence of decisions. We have to decide the values of xi, 1<= i<+n. First we make a decision on xi , then on x2 . then on x3 ,etc. An optimal sequence of decisions maximizes the objective function ∑pixi.



**Problem Definition:**

Given directed graph *G* = (*V*, *E*), a weight for each edge in *G*, a source node *v*0, we have to determine the shortest paths from *v*0 to all the remaining vertices in *G*. Length of the path is sum of the weight of the edges



**Algorithm**

Algorithm Bellman Ford(v,cost,dist,n)

{

for i=1 to n do

dist[i]=cost[v,i]

for k=2 to n-1 do

for each u such that u≠v and u has

at least one incoming edge do

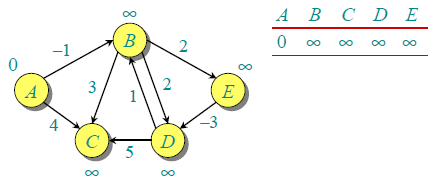
for each<I,u> in the graph do

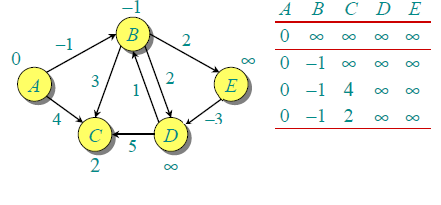
if dist[u]>dist[i]+cost[I,u] then

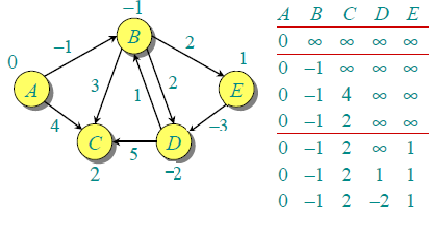
dist[u]=dist[i]+cost[i,u]

**}**

**Example:**







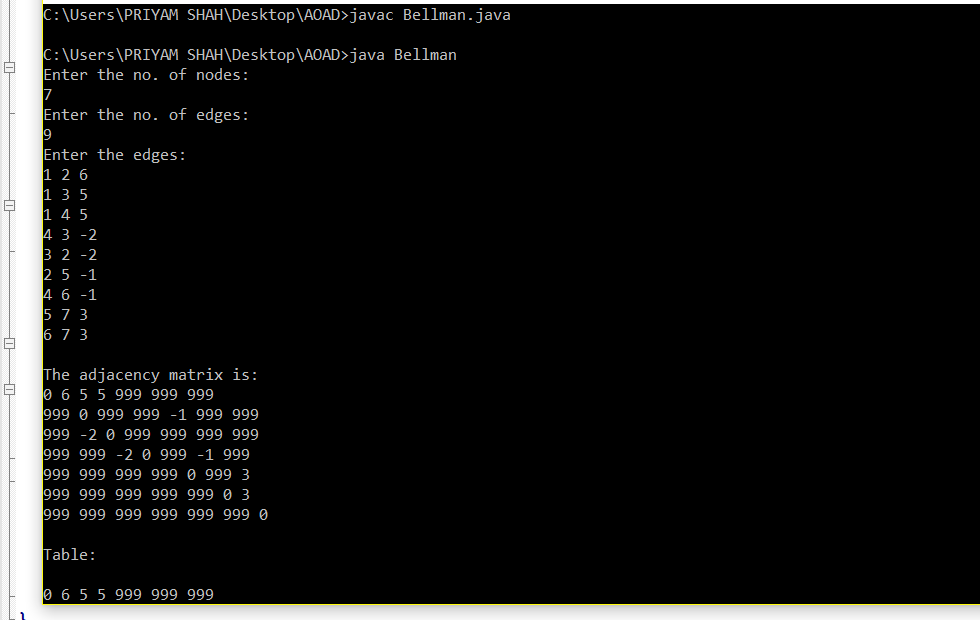
**Analysis of Bellman Ford Algorithm:**

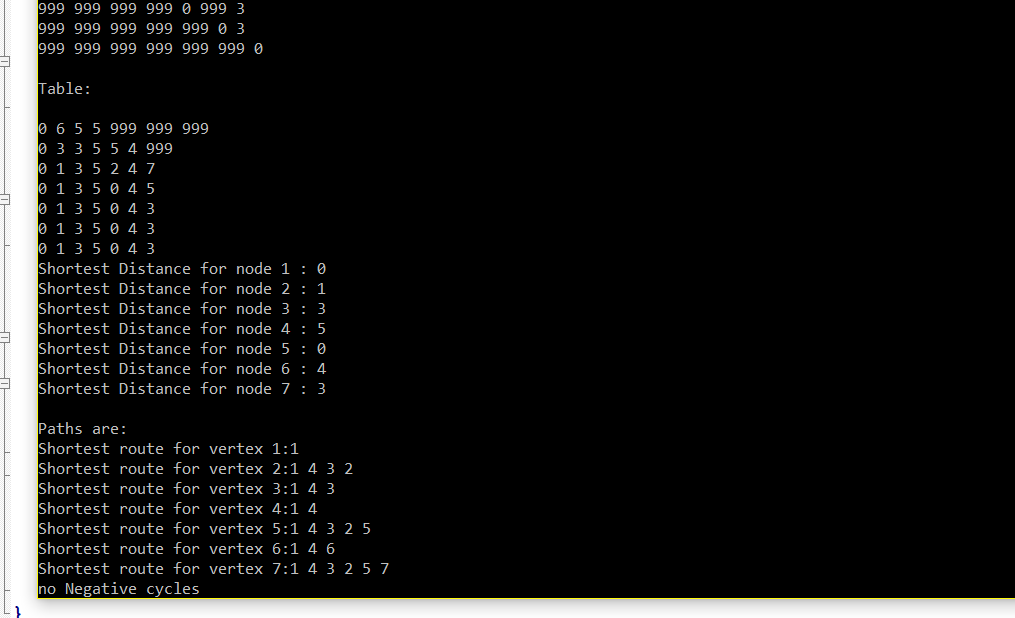
***Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.***

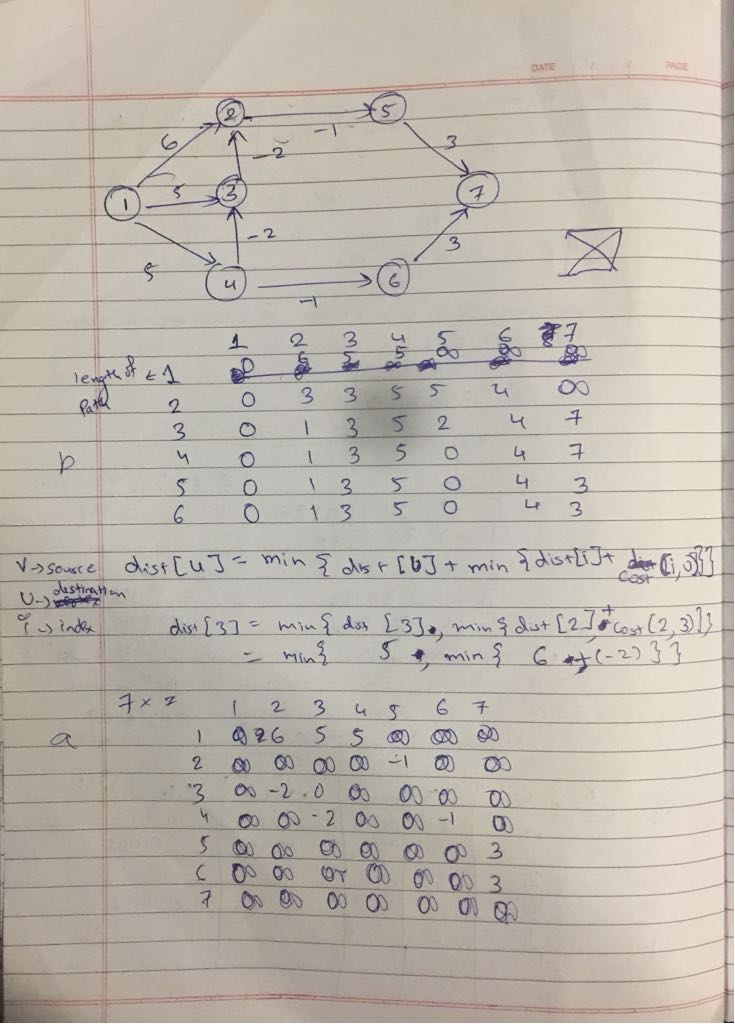
**Code:**

**import java.util.\*;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 Scanner sc=new Scanner (System.in);  
 System.out.println("Enter the no. of nodes: ");  
 int n=sc.nextInt();  
 System.out.println("Enter the no. of edges: ");  
 int e=sc.nextInt();  
 int a[][]=new int[n][n];  
 int b[][]=new int[n-1][n];  
 int i,j,k,l;  
 String path[]=new String[n];  
  
 System.out.println("Enter the edges: ");  
 for(i=0;i<e;i++)  
 {  
 a[sc.nextInt()-1][sc.nextInt()-1]=sc.nextInt();  
 }  
 for(i=0;i<n;i++)  
 {  
 for(j=0;j<n;j++)  
 {  
 if(a[i][j]==0&&i!=j)  
 {  
 a[i][j]=999;  
 }  
 }  
 }  
  
 for(i=0;i<n;i++)  
 {  
 path[i]="1";  
 if(a[0][i]!=0 && a[0][i]!=999)  
 {  
 path[i]=path[i]+" "+(i+1);  
 }  
  
 }  
  
 System.out.println("\nThe adjacency matrix is: ");  
 for(i=0;i<n;i++)  
 {  
 for(j=0;j<n;j++)  
 {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.print("\n");  
 }  
 System.out.println("\nTable: \n");  
 for(i=0;i<n;i++)  
 {  
 b[0][i]=a[0][i];  
 }  
   
 for(l=0;l<n;l++)  
 {  
 System.out.print(b[0][l]+" ");  
 }  
  
 int temp[]=new int[n];  
 int t,t2=0,f,z=0;  
 for(i=1;i<n-1;i++)  
 {  
 for(j=0;j<n;j++)  
 {  
 t=b[i-1][j];  
 for(k=0;k<n;k++)  
 {  
 temp[k]=b[i-1][k]+a[k][j];  
 if(k==0)  
 {  
 t2=temp[k];  
 }  
 else if(k>0&&temp[k]<t2)  
 {  
 t2=temp[k];  
 z=k;  
 }  
 }  
 if(t2<t)  
 {  
 b[i][j]=t2;  
 path[j]=path[z]+" "+(j+1);  
 }  
 else  
 b[i][j]=t;  
 }  
 System.out.println();  
 for(l=0;l<n;l++)  
 {  
 System.out.print(b[i][l]+" ");  
 }  
  
 }  
  
 System.out.println();  
 for(i=0;i<n;i++)  
 {  
 System.out.println("Shortest Distance for node "+(i+1)+" : "+b[n-2][i]);  
 }  
  
 System.out.println("\nPaths are:");  
 for(i=0;i<n;i++)  
 {  
 System.out.println("Shortest route for vertex "+(i+1)+":"+path[i]);   
 }  
   
   
 }  
}**

**Output:**

****

****

****

**CONCLUSION:**

**Implementation of Single source shortest path algorithm using Dynamic Programming has been studied.**