Optimization of Water Adduction of Mains path

Table of Contents

Abstract

The efficient design of water adduction main involves several optimization processes among which an important place is held by their path optimization. In this project, we developed two deterministic mathematical models for optimization of water adduction main path. Using these optimization models could be obtained an optimal solution for selection of source location and of water adduction main path based on graph theory and dynamic programming.

Introduction

The water adduction mains have a significant importance in water supply systems due to their large investment amount and important energy consumption. As a consequence, the efficient design of water adduction main involves several optimization processes among which an important place is held by their path optimization.

In current design practice, the choice of the optimal solution is made usually through analytical study of two or three versions selected from the possible set by predicted decisions. The errors of these decisions are inverse proportional to the designer experience.

The modern mathematical disciplines as operational research gives to the designer a vast apparatus of scientific analysis in optimal decisions establishing. The mathematical theory and planning of multistage decision processes, the term was introduced by Richard Bellman in 1957. It may be regarded as a branch of mathematical programming or optimization problems formulated as a sequence of decision.

Traditional optimization algorithms have been applied to the minimum cost optimal design problem, such as linear programming, first introduced by Labye for open networks.

Base Paper

This paper develops two deterministic mathematical models for optimization of water adduction main path, based on techniques of sequential operational calculus, implemented in a computer program. Using these optimization models could be obtained an optimal solution for selection of source location and of water adduction main path based on graph theory and dynamic programming. The results of few numerical applications show the effectiveness and efficiency of the proposed optimization models.

The following two optimization models are:

1. Dynamic Programming

   Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. Dynamic programming is applied to the general class of water-resources problems to permit optimum development with respect to all possible benefits.

2. Graph Theory

   Mathematical model of dynamic processes of discrete and, determinist type can be simplified using graph theory. A Graph is a non-linear data structure consisting of nodes and edges. The

nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.
Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like LinkedIn, Face book. For example, in Face book, each person is represented with a vertex (or node). Each node is a structure and contains information like person id, name, gender, locale etc.

Requirements

Software specification

- **Code blocks**
- **Windows 7 or above**
- **ChartGo (for graph creation)**

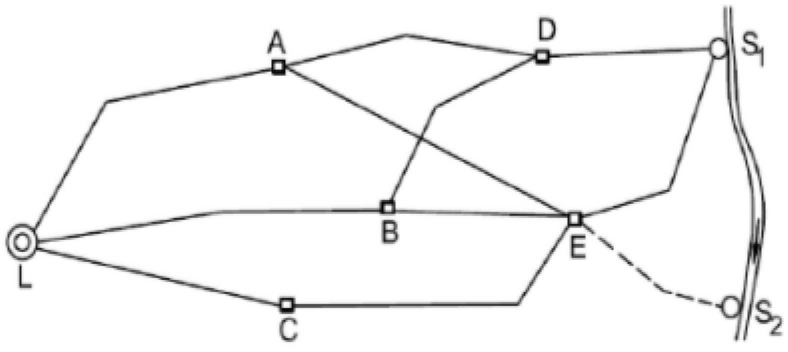
Current Scenario

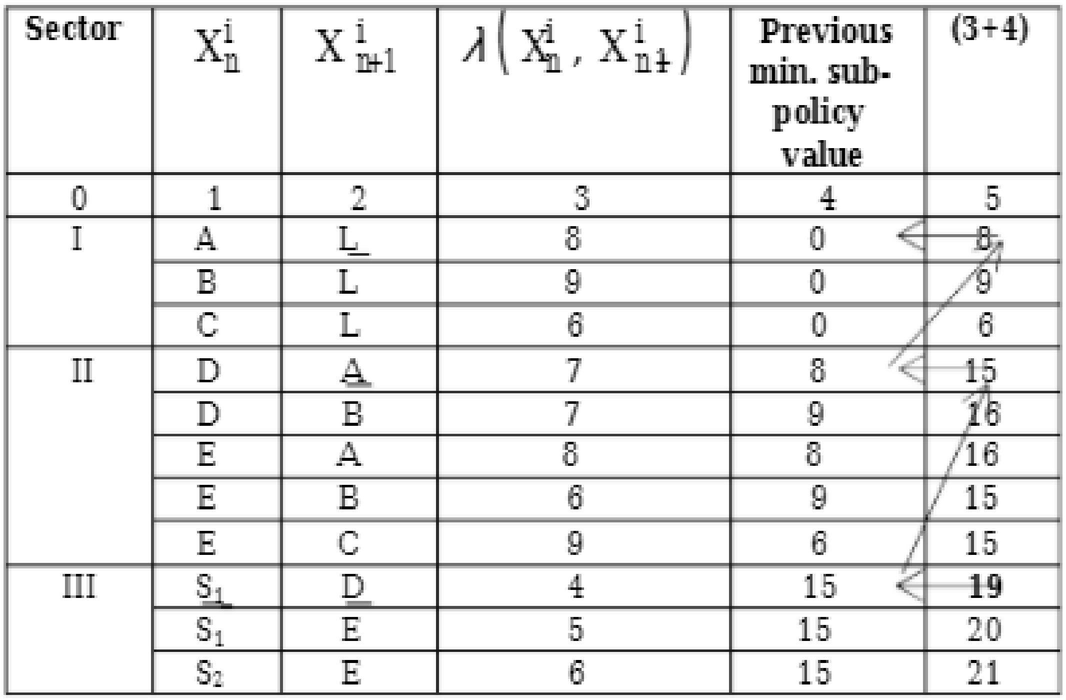
We have a main village L.

S1 and S2 are two water sources connected with a pipeline.

There are several other villages A,B,C,D,E.

We have to build a optimized path to send water from S1 or S2 to L.





Sector	X_n^i	X_{n+1}^i	$\lambda\left(X_n^i, X_{n+1}^i\right)$	Previous min. sub-policy value	(3+4)
0	1	2	3	4	5
I	A	L	8	0	8
	B	L	9	0	9
	C	L	6	0	6
II	D	A	7	8	15
	D	B	7	9	16
	E	A	8	8	16
	E	B	6	9	15
	E	C	9	6	15
III	S_1	D	4	15	**19**
	S_1	E	5	15	20
	S_2	E	6	15	21

Optimization model based on Dynamic Programming

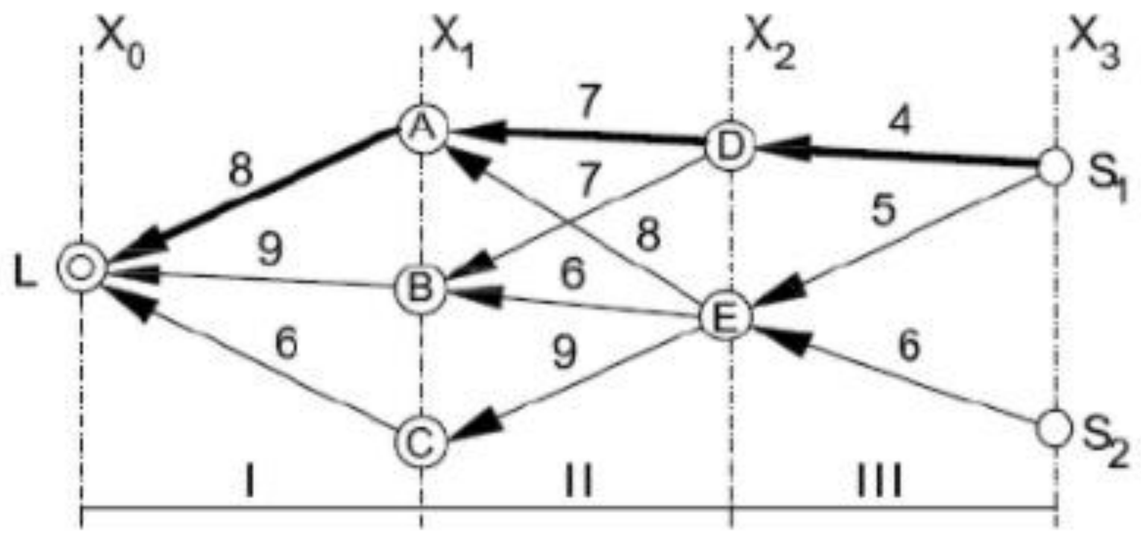
Code:

```
#include<iostream>
using namespace std;
int d1[100],d2[100],p[100];
int min(int a[],int n,int c[],int d[],int q)
{int m=a[0],t=0;
for(int i=0;i<n;i++)
{if(a[i]<m)
{m=a[i]; t=i;}
}
d[q]=c[t];
return m;
}
int main()
{ int n,f1[100][2],f12[100][2],f123[2][2];
f123[0][0]=6;
f123[1][0]=7;
f123[0][1]=f123[1][1]=0
; cout<<"\nvalue of n: ";
cin>>n;
int j=0;
int
k=0;
int x;
cout<<"\nLevel 1: Critcial Points connected to Target\n";
for(int i=1;i<=n;i++)
{
   cout<<"\nDistance between "<<i<<" and L:";
   cin>>x;
if(x!=0)
```

```
{f1[j][0]=i;
f1[j][1]=x;
j++;
}
else if(x==0)
{f12[k][0]=i;
f12[k][1]=0;
k++;
}
}
int c[100];
if(k!=0)
{cout<<"\nLevel 2: Critcial Points connected to Source(s)\n";
for(int i=0;i<k;i++)
{int a=0,f12m[100];
for(int l=0;l<j;l++)
{ cout<<"\nDistance between "<<f12[i][0]<<" and "<<f1[l][0]<<":";
cin>>x;
if(x!=0)
{f12m[a]=x+f1[l][1];
c[a]=f1[l][0];
a++;
}
}
f12[i][1]=min(f12m,a,c,d1,i);
}
}
else{
{f12[k][0]=f1[0][0];
f12[k][1]=f1[0][1];
k++;
}}
cout<<"\nLevel 3: Distance between Level 2 points and Source(s)\n";
for(int i=0;i<2;i++)
{int b=0,f123m[100];
```

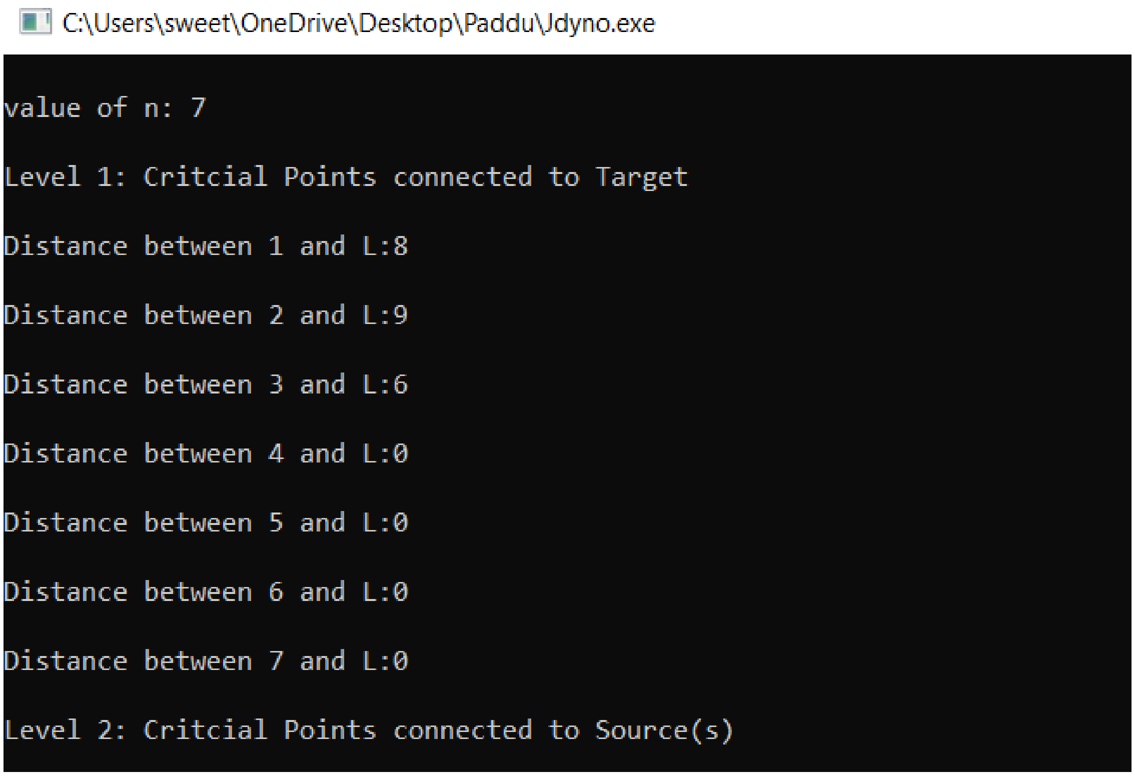
```
for(int l=0;l<k;l++)
{ cout<<"\nDistance between "<<f123[i][0]<<" and "<<f12[l][0]<<":";
cin>>x;
if(x!=0)
{f123m[b]=x+f12[l][1];
c[b]=f12[l][0];
b++;
}
}
f123[i][1]=min(f123m,b,c,d2,i);
}
int i;
if(f123[0][1]<f123[1][1])
{i=0;}
else
i=1;
p[0]=d1[i]; p[1]=d2[i];
int m;
if(f123[0][1]>f123[1][1])
{m=f123[1][1]; p[2]=f123[1][0];}
else
{m=f123[0][1]; p[2]=f123[0][0];}
cout<<"-------------------------------\nPath: "<<p[2]<<"->"<<p[1]<<"-
>"<<p[0]<<"->L"<<"\n\nTotal Minimum Distance: "<<m;
return 0;
}
```

Input:



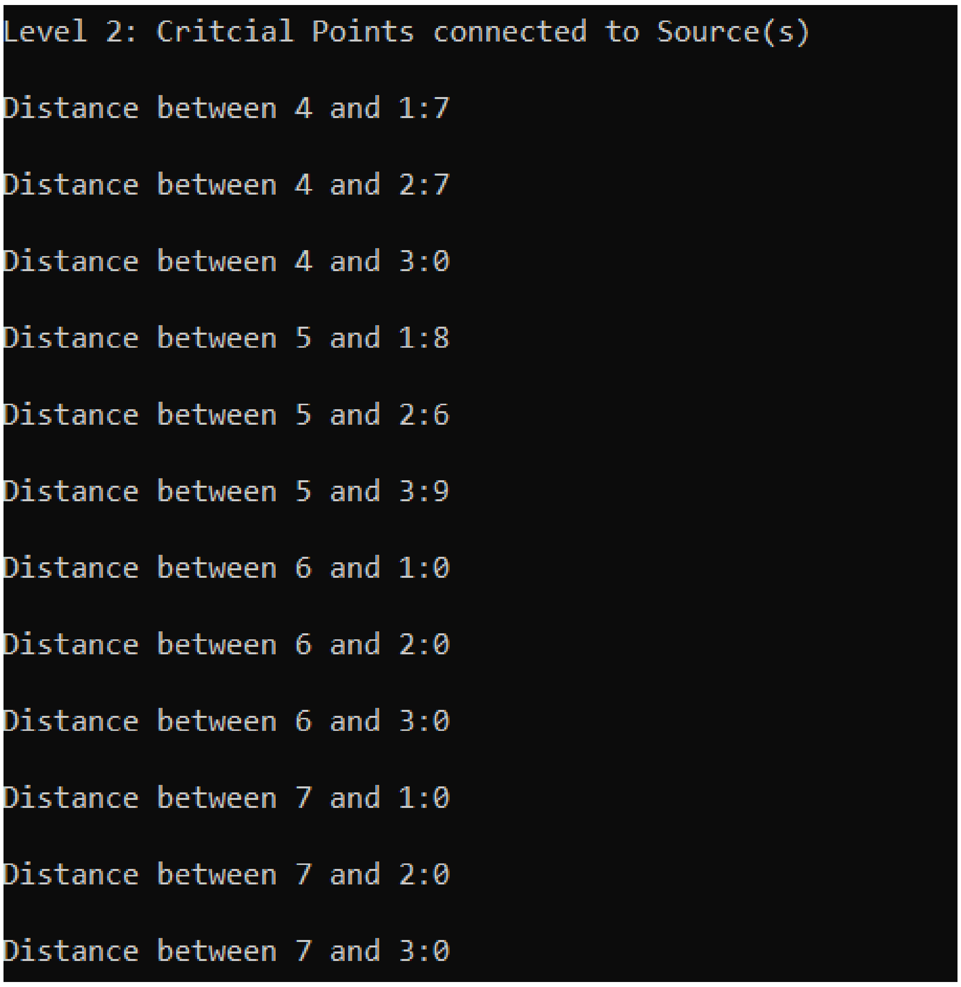
Here A=1, B=2, C=3, D=4, E=5, S1=6, S2=7

Output:

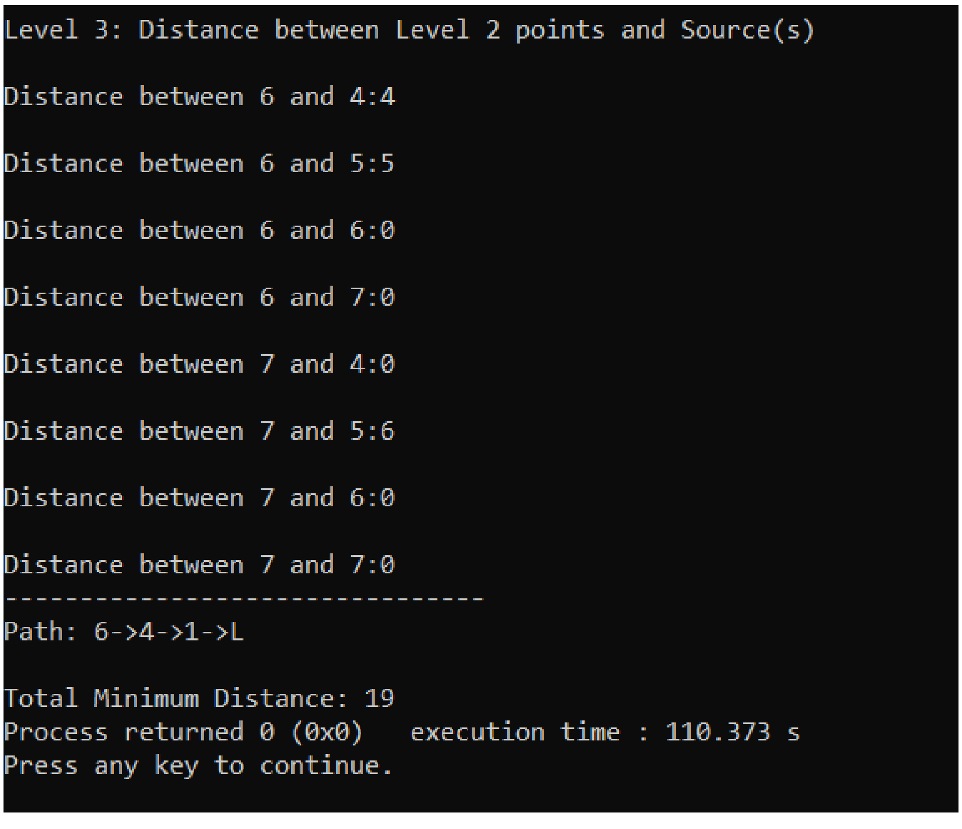


C:\Users\sweet\OneDrive\Desktop\Paddu\Jdyno.exe

```
value of n: 7

Level 1: Critcial Points connected to Target

Distance between 1 and L:8

Distance between 2 and L:9

Distance between 3 and L:6

Distance between 4 and L:0

Distance between 5 and L:0

Distance between 6 and L:0

Distance between 7 and L:0

Level 2: Critcial Points connected to Source(s)
```



```
Level 2: Critcial Points connected to Source(s)

Distance between 4 and 1:7

Distance between 4 and 2:7

Distance between 4 and 3:0

Distance between 5 and 1:8

Distance between 5 and 2:6

Distance between 5 and 3:9

Distance between 6 and 1:0

Distance between 6 and 2:0

Distance between 6 and 3:0

Distance between 7 and 1:0

Distance between 7 and 2:0

Distance between 7 and 3:0
```



```
Level 3: Distance between Level 2 points and Source(s)

Distance between 6 and 4:4

Distance between 6 and 5:5

Distance between 6 and 6:0

Distance between 6 and 7:0

Distance between 7 and 4:0

Distance between 7 and 5:6

Distance between 7 and 6:0

Distance between 7 and 7:0
--------------------------------
Path: 6->4->1->L

Total Minimum Distance: 19
Process returned 0 (0x0)   execution time : 110.373 s
Press any key to continue.
```

Optimization using Graph Theory

Code:

```
#include<stdio.h>
#define MAX 100
#define TEMP 0
#define PERM 1
#define infinity 9999
#define NIL -1
void findpath(int s,int v);
void Dijkstra(int s);
int min_temp();
void create_graph();
int n; /*Denotes number of vertices in the graph*/
int adj[MAX][MAX];
int predecessor[MAX];
int pathLength[MAX];
int status[MAX];
main()
{
int s,v;
create_graph();
while(1)
{
```

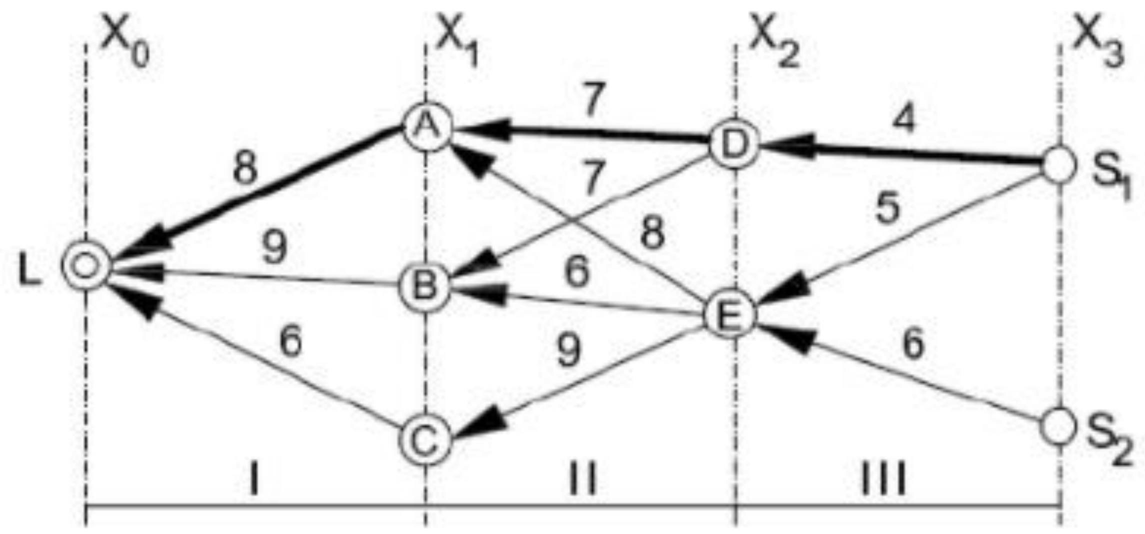
```
printf("Enter source vertex(-1 to quit) : ");
scanf("%d",&s);
if(s==-1)
break;
Dijkstra(s);
printf("Enter destination vertex: ");
scanf("%d",&v);
if(v<0||v>=n)
printf("The vertex does not exist: ");
else if(v==s)
printf("Source and destination vertices are same");
else if(pathLength[v]==infinity)
printf("there is no path from source to destination");
else
findpath(s,v);
}
}/*End of main()*/
void Dijkstra(int s)
{
int i,current;
/*Make all vertices temporary*/
for(i=0;i<n;i++)
{
predecessor[i]= NIL;
```

```
pathLength[i] = infinity;
status[i]=TEMP;
}
/*Make pathLength of source vertex equql to zero*/
pathLength[s]=0;
while(1)
{
/*Search for minimum pathlength and make it current vertex*/
current=min_temp();
if(current==NIL)
return;
status[current]=PERM;
for(i=0;i<n;i++)
{
/*checks for adjacent temporary vertices*/
if(adj[current][i]!=0 && status[i]==TEMP)
if(pathLength[current]+adj[current][i]<pathLength[i])
{
predecessor[i]=current;
pathLength[i]=pathLength[current] +adj[current][i];
}
}
}
}
```

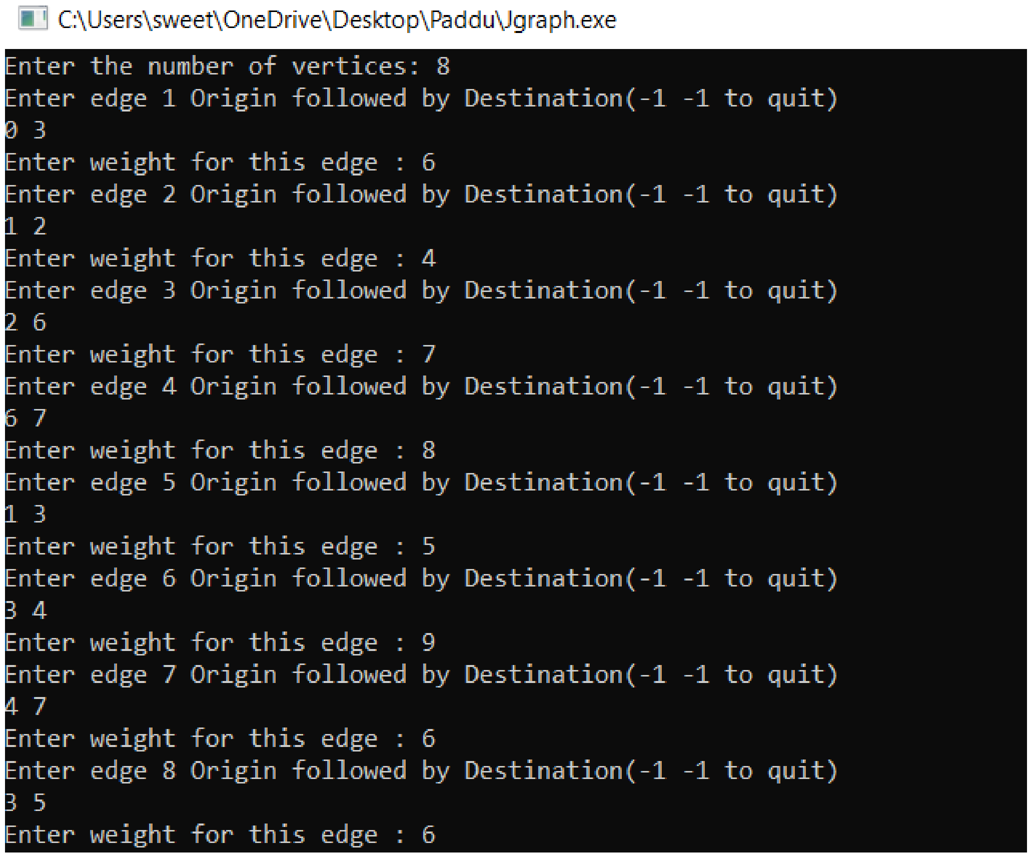
```
/*End of Dijkstra*/
/*returns the temporary vertex with minimum value of pathlength,
Returns NIL if no
temporary
 vertex left or all temporary vertices left have path length infinity*/
int min_temp()
{
int i;
int min = infinity;
int k = NIL;
for(i=0;i<n;i++)
{
if(status[i]==TEMP && pathLength[i]<min)
{
min=pathLength[i];
k=i;
}
}
return k;
}
/*End of min_temp()*/
void findpath(int s,int v)
{
int i,u;
int path[MAX];/*Stores the shortest path*/
int shortdist=0;/*length of shortest path*/
int count =0;/*number of vertices in the shortest path*/
/*Stores the full path in the array path*/
while(v!=s)
{
count++;
path[count]=v;
u=predecessor[v];
shortdist += adj[u][v];
v=u;
}
count++;
path[count]=s;
printf("Shortest Path is: ");
for(i=count; i>=1; i--)
printf("%d ",path[i]);
printf("\n Shortest distance is : %d\n",shortdist);
}/*End of findpath()*/
void create_graph()
{
int i ,max_edges,origin,destin,wt;
printf("Enter the number of vertices: ");
scanf("%d",&n);
```

```
max_edges=n*(n-1);
for(i=1;i<=max_edges; i++)
{
printf("Enter edge %d Origin followed by Destination(-1 -1 to
quit)\n", i);
scanf("%d%d",&origin,&destin);
if((origin==-1)&&(destin==-1))
break;
printf("Enter weight for this edge : ");
scanf("%d",&wt);
if(origin>=n || destin>=n || origin<0 || destin<0)
{
printf("invalid edge!\n");
i--;
}
else
adj[origin][destin]=wt;
}
}
```

Input:

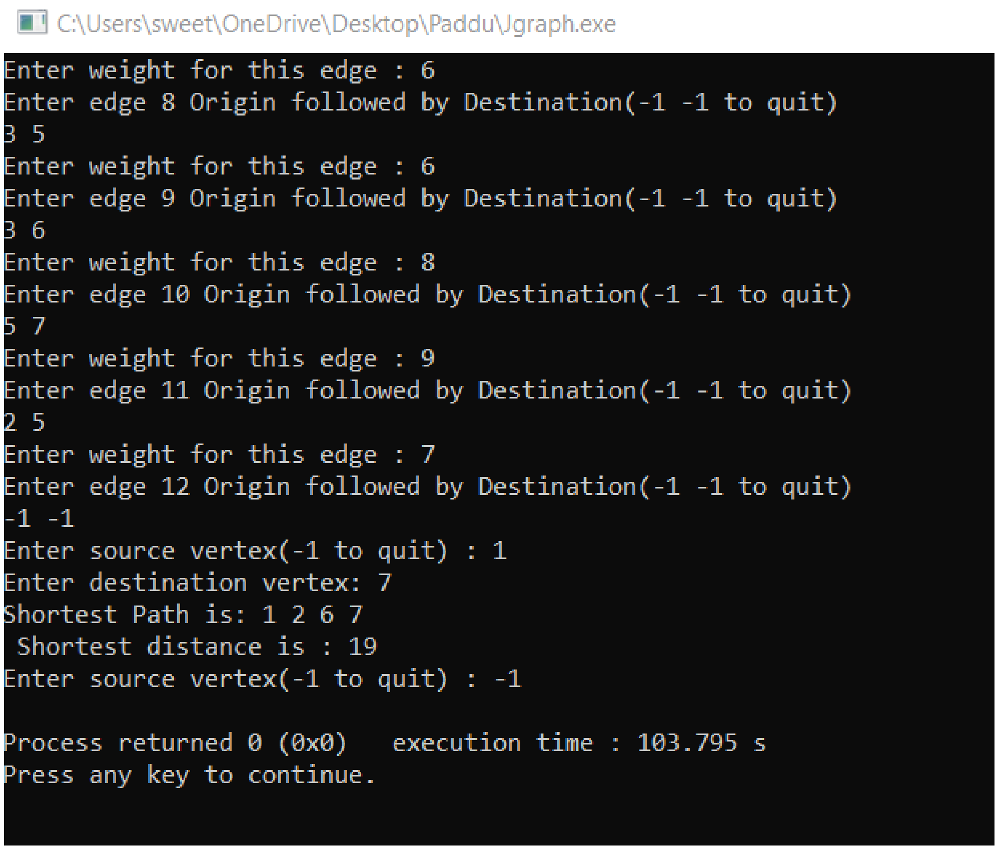


Output:



C:\Users\sweet\OneDrive\Desktop\Paddu\Jgraph.exe

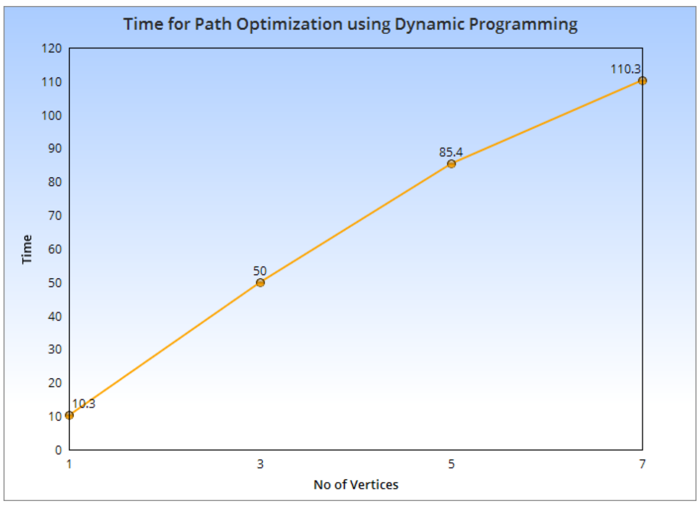
```
Enter the number of vertices: 8
Enter edge 1 Origin followed by Destination(-1 -1 to quit)
0 3
Enter weight for this edge : 6
Enter edge 2 Origin followed by Destination(-1 -1 to quit)
1 2
Enter weight for this edge : 4
Enter edge 3 Origin followed by Destination(-1 -1 to quit)
2 6
Enter weight for this edge : 7
Enter edge 4 Origin followed by Destination(-1 -1 to quit)
6 7
Enter weight for this edge : 8
Enter edge 5 Origin followed by Destination(-1 -1 to quit)
1 3
Enter weight for this edge : 5
Enter edge 6 Origin followed by Destination(-1 -1 to quit)
3 4
Enter weight for this edge : 9
Enter edge 7 Origin followed by Destination(-1 -1 to quit)
4 7
Enter weight for this edge : 6
Enter edge 8 Origin followed by Destination(-1 -1 to quit)
3 5
Enter weight for this edge : 6
```

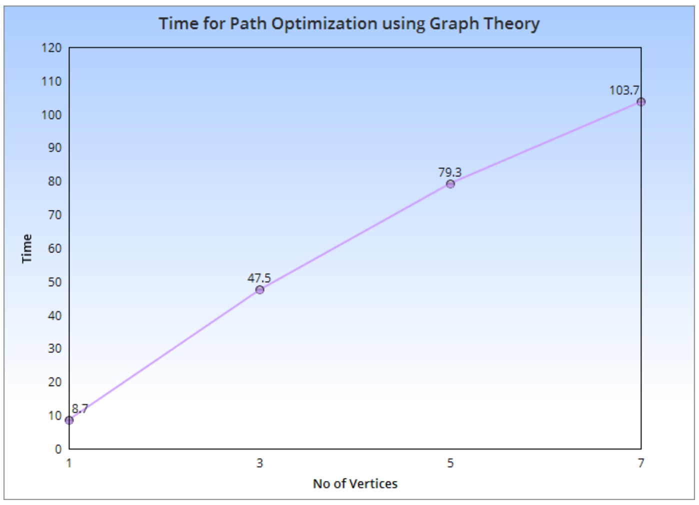

C:\Users\sweet\OneDrive\Desktop\Paddu\Jgraph.exe

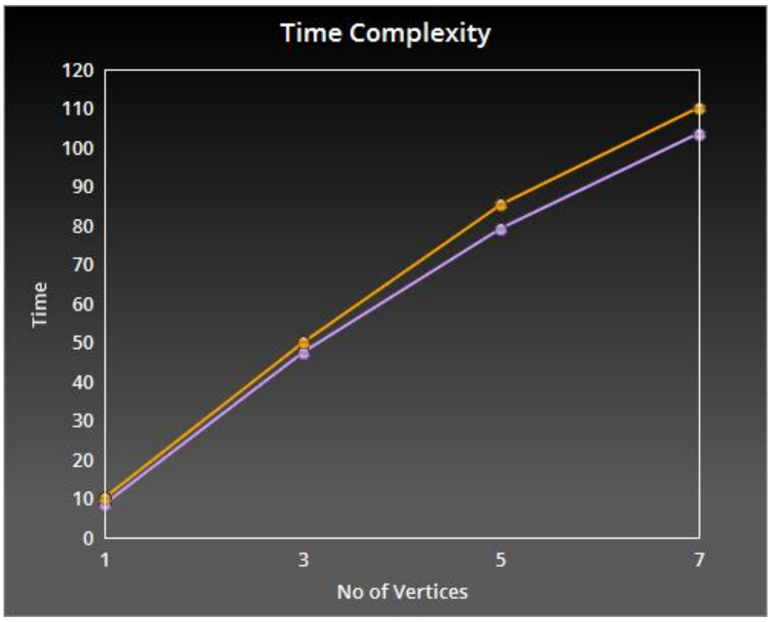
```
Enter weight for this edge : 6
Enter edge 8 Origin followed by Destination(-1 -1 to quit)
3 5
Enter weight for this edge : 6
Enter edge 9 Origin followed by Destination(-1 -1 to quit)
3 6
Enter weight for this edge : 8
Enter edge 10 Origin followed by Destination(-1 -1 to quit)
5 7
Enter weight for this edge : 9
Enter edge 11 Origin followed by Destination(-1 -1 to quit)
2 5
Enter weight for this edge : 7
Enter edge 12 Origin followed by Destination(-1 -1 to quit)
-1 -1
Enter source vertex(-1 to quit) : 1
Enter destination vertex: 7
Shortest Path is: 1 2 6 7
 Shortest distance is : 19
Enter source vertex(-1 to quit) : -1

Process returned 0 (0x0)   execution time : 103.795 s
Press any key to continue.
```

Graph for Time Complexity

No of Vertices	Time for path optimization using Dynamic Programming	Time for path optimization using Graph Theory
1	10.3 s	8.7 s
3	50.0 s	47.5 s
5	85.4 s	79.3 s
7	110.3 s	103.7 s







Conclusion

In the presented optimization models were described two techniques of operational research for solving sequential programs. The dynamic programming approach was successful in determining an optimal path for the water network. However, the path determined by using graph theory was not only successful in determining the path, but also was more time-efficient, based on the time-complexity analysis.

Reference

- https://www.researchgate.net/publication/259739123_Optimization_of_water_distribution_networks_path
- https://www.chartgo.com/home.html