'Transfer Cost Edge to Vertex Graph' Search Algorithm

Ahmad Karami Bukani [1]

Abstract

This algorithm calculates the shortest and longest path with a new method and although the edges are not omitted. Instead of costs on edges, costs are placed on vertices and help in routing in the graph.

Keywords: path finding – graph-vertex-edge -cost transfer - new method – `permutations - random` - algorithm

1- Introduction

At universities and schools around the world, they draw graphs by connecting headlines by lines, while graphs at the edges and the costs of the edges at the top can, part of the complexity of the calculations Remove the shortest and longest route. The edges comprise the movement from point to point in the algorithm, if the costs of the edges are at the top and the costs in the edges are eliminated, the complexity of the algorithm will be reduced.

2- Overview

I have used the Python 3.12 programming language to implement the graphic search algorithm Transfer Vertex Cost to Edge and this algorithm contains two parts:

A. Based on parametric selection, random graff production or admission graph

B- Solve the shortest and longest graphs of graph

I use random libraries and mathematical jacks as well as a number of array variables that I have used the list data type, and I have elaborated on the duplicate lines, and I will explain the important parts of the codes.

Ahmad Karami Bukani
Ahmad.dg84@gmail.com
Department of Education – Bukan City – West Azarbaijan (Mukriyan Kurdistan) Province – Iran

```
import random
from itertools import permutations
```

Using the word Import and From, a random library that helps produce a random or random graph based on the selection of one of the two.

```
List=[]
```

The letters of graphs are assigned to the List variable based on the ASCII table.

```
List2=[]
```

The cost of graphs of graphs is allocated in the List2 variable.

```
List3=[]
```

All graphs produced are placed in the List3 using a random or random place.

```
List4=[]
```

Using the List4 variable, part of the residual graphs are stored.

```
List5=[]
```

A copy of the paths is placed in the List4 variable in the List5 variable.

```
ListFinal=[]
```

In the ListFinal variable, different paths are considered.

```
MyPathList=[]
```

In the graph path, if it has the starting point or the end point, the path is added to the MyPathList variable.

```
MyPathList2=[]
```

Different paths are only paths in MyPathList2 variable that start from the starting point based on the Start variable and terminate based on the end variable.

```
PathSum=""
```

Saves different directions of graphs as a string.

Sum=0

The cost saves different graphs of graphs.

AllListCost=[]

The AllListCost variable helps find the shortest and longest graph.

PathListCost=[]

The PathListCost variable helps to find the cost of the shortest and longest graph.

3-Generation of random graph or permutation graph

Welcome to the search algorithm "Transfer Vertex Cost to Edge Graph"

print("Welcome To \"Transfer Vertex Cost To Edge Graph Search\" Algorithm")

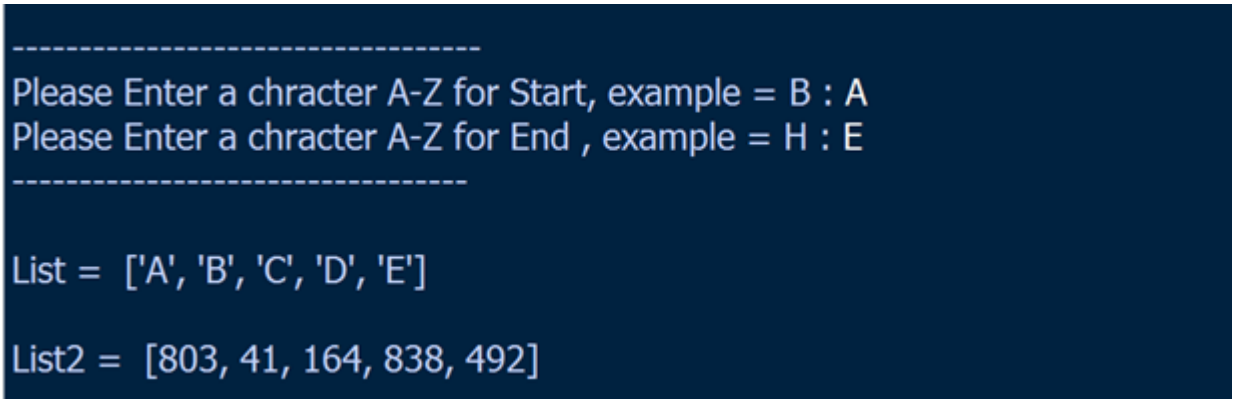

Figure 1

According to Figure 1, two messages are displayed that the number of graphs is specified and allocated by the names. The first message is specified and allocated by variable S variable and the second message with variable E variable is identified and allocated.

```
s=ord(input("Please Enter a character A-Z for Start, example = B : "))
```

The var variable determines the starting point of the graph movement, and here you need to enter a character in the range A to Z using the input function. The default can enter the letter B and return the ORD function of the letter ASCII entered.

```
e=ord(input("Please Enter a character A-Z for End , example = H : "))+1
```

The e variable determines the end point of the graph of the graph, and here you need to enter a character in the range A to Z using the input function. The default can enter the letter H and return the Ord ASCII number. Here a unit is added to the e variable, the reason is in the For loop, which is considered a less unit when we use the Range function. For this reason, in the e variable, we have considered a more unique unit so that the ranges are accurately entered.

According to Figure 2, it is determined how the combination of graphs is created, is the graph or random? Produce the number one, the graph, and produce the number two, the graph, by random.

```
Please Select For Make Graph via Permutations or Random, Enter Press Key 1 or Key 2 : 1

----------  Processing  ----------

List3 =
 [('A', 'B', 'C', 'D', 'E'), ('A', 'B', 'C', 'E', 'D'), ('A', 'B', 'D', 'C', 'E'), ('A', 'B', 'D', 'E', 'C'), ('A', 'B', 'E', 'C', 'D'), ('A',
 'B', 'E', 'D', 'C'), ('A', 'C', 'B', 'D', 'E'), ('A', 'C', 'B', 'E', 'D'), ('A', 'C', 'D', 'B', 'E'), ('A', 'C', 'D', 'E', 'B'), ('A', 'C', '
E', 'B', 'D'), ('A', 'C', 'E', 'D', 'B'), ('A', 'D', 'B', 'C', 'E'), ('A', 'D', 'B', 'E', 'C'), ('A', 'D', 'C', 'B', 'E'), ('A', 'D', 'C', 'E
', 'B'), ('A', 'D', 'E', 'B', 'C'), ('A', 'D', 'E', 'C', 'B'), ('A', 'E', 'B', 'C', 'D'), ('A', 'E', 'B', 'D', 'C'), ('A', 'E', 'C', 'B', 'D')
, ('A', 'E', 'C', 'D', 'B'), ('A', 'E', 'D', 'B', 'C'), ('A', 'E', 'D', 'C', 'B'), ('B', 'A', 'C', 'D', 'E'), ('B', 'A', 'C', 'E', 'D'), ('B',
'A', 'D', 'C', 'E'), ('B', 'A', 'D', 'E', 'C'), ('B', 'A', 'E', 'C', 'D'), ('B', 'A', 'E', 'D', 'C'), ('B', 'C', 'A', 'D', 'E'), ('B', 'C', '
A', 'E', 'D'), ('B', 'C', 'D', 'A', 'E'), ('B', 'C', 'D', 'E', 'A'), ('B', 'C', 'E', 'A', 'D'), ('B', 'C', 'E', 'D', 'A'), ('B', 'D', 'A', 'C
', 'E'), ('B', 'D', 'A', 'E', 'C'), ('B', 'D', 'C', 'A', 'E'), ('B', 'D', 'C', 'E', 'A'), ('B', 'D', 'E', 'A', 'C'), ('B', 'D', 'E', 'C', 'A')
, ('B', 'E', 'A', 'C', 'D'), ('B', 'E', 'A', 'D', 'C'), ('B', 'E', 'C', 'A', 'D'), ('B', 'E', 'C', 'D', 'A'), ('B', 'E', 'D', 'A', 'C'), ('B',
'E', 'D', 'C', 'A'), ('C', 'A', 'B', 'D', 'E'), ('C', 'A', 'B', 'E', 'D'), ('C', 'A', 'D', 'B', 'E'), ('C', 'A', 'D', 'E', 'B'), ('C', 'A', '
E', 'B', 'D'), ('C', 'A', 'E', 'D', 'B'), ('C', 'B', 'A', 'D', 'E'), ('C', 'B', 'A', 'E', 'D'), ('C', 'B', 'D', 'A', 'E'), ('C', 'B', 'D', 'E
', 'A'), ('C', 'B', 'E', 'A', 'D'), ('C', 'B', 'E', 'D', 'A'), ('C', 'D', 'A', 'B', 'E'), ('C', 'D', 'A', 'E', 'B'), ('C', 'D', 'B', 'A', 'E')
, ('C', 'D', 'B', 'E', 'A'), ('C', 'D', 'E', 'A', 'B'), ('C', 'D', 'E', 'B', 'A'), ('C', 'E', 'A', 'B', 'D'), ('C', 'E', 'A', 'D', 'B'), ('C',
'E', 'B', 'A', 'D'), ('C', 'E', 'B', 'D', 'A'), ('C', 'E', 'D', 'A', 'B'), ('C', 'E', 'D', 'B', 'A'), ('D', 'A', 'B', 'C', 'E'), ('D', 'A', '
```

Figure 2

```
for i in range(s,e):
```

The For loop does three tasks using the range of e and s variables.

```
List.append(chr(i))
```

Using the CHR function, the English letter of the ASCII table is added to the List variable.

```
List2.append(random.randint(1,20))
```

Using the Randint function, the cost of 1 to 1000 randomly in the List2 variable is equivalent to the List variable, and each vertex is assigned to the English letter of a random cost.

```
print(“List = “,List)
print(“List2 = “,List2)
```

According to Figure 1, the results of the for loop codes for List, List2 are displayed at the output.

```
Select=input(“Please Select For Make Graph via Permutations or Random, Enter Press Key 1 or Key 2 : “)
```

Using the Select variable, we determine if the number 1 or number 2 is entered, the number one will create a graph of the graph and the number two will produce random graphs.

```
print(‘--------------   Processing   ---------------')
```

According to Figure 2, it is produced by displaying the message and performing the calculations of the graph or the random graph.

```
if Select==”1” :
   Tuple=()
   Tuple = permutations(List)
   List3=list(Tuple)
```

If the number one is entered into the Select variable, using the Topple Data type, the grooming graph is produced using the Permutations function and since the output data type of the Permutations function is of the top type, then it must be used using the Tuple LIST function. Convert the List3 variable with the list data type and the graph is produced.

```
if Select==”2”:
     d=e-s
```

If the number two is entered in the Select variable, the distance of the variable e, s is calculated and based on the distance that its value is stored in the variable d. So the random graph production ring is produced.

```
     for v in range(0,2**d*d**2):
```

Formula 2 ** d*d ** 2 is merely to generate different graphs of graph paths in random mode and for loop for, random graph random, power and multiplication based on Formula 2 to power d, Crossing d to 2 Graph Production Calculations Randomly, based on the existence of the exec function inside, the ss variable has some empty. Then in the next for for zero to the point of origin to the destination point, the variable value a using the character generation function based on the randint random number function, the amount of the source of the source point to the destination point, one The unit will be less. If we do not consider a less unit, e-1 in the randint function and write as randint (s, e), it calculates a higher than the specified limit and causes more computational error and graph production.

```
       exec(“ss=’’”)
       for r in range(0,d):
            exec(“a = chr(random.randint(s,e-1))”)
            exec(“ss=ss+a”)
```

Then the ss variable value, which is the first minute, will be new to the content of the variable a, and the exec function will be repeated at each stage of the for loop execution to produce a random path from the source point to the destination based on the vertices.

```
List3.append(list(ss))
```

Each time the random path production is added, the content of the ss variable is added to the List3.

```
print("List3 = \n",List3)
```

The List3 variable is displayed and this variable is actually a list of all random routes from the source point to the target point.

```
print('--------------   Done   ---------------')
```

```
Please Enter a number for Splite List (1-length(List)) , example = 3 : 3

--------------  Processing  ---------------

List4 =
 [('A', 'B', 'C', 'D', 'E'), ('A', 'B', 'C', 'E', 'D'), ('A', 'B', 'D', 'C', 'E'), ('A', 'B', 'D', 'E', 'C'), ('A', 'B', 'E', 'C', 'D'), ('A',
 'B', 'E', 'D', 'C'), ('A', 'C', 'B', 'D', 'E'), ('A', 'C', 'B', 'E', 'D'), ('A', 'C', 'D', 'B', 'E'), ('A', 'C', 'D', 'E', 'B'), ('A', 'C', '
E', 'B', 'D'), ('A', 'C', 'E', 'D', 'B'), ('A', 'D', 'B', 'C', 'E'), ('A', 'D', 'B', 'E', 'C'), ('A', 'D', 'C', 'B', 'E'), ('A', 'D', 'C', 'E
', 'B'), ('A', 'D', 'E', 'B', 'C'), ('A', 'D', 'E', 'C', 'B'), ('A', 'E', 'B', 'C', 'D'), ('A', 'E', 'B', 'D', 'C'), ('A', 'E', 'C', 'B', 'D')
, ('A', 'E', 'C', 'D', 'B'), ('A', 'E', 'D', 'B', 'C'), ('A', 'E', 'D', 'C', 'B'), ('B', 'A', 'C', 'D', 'E'), ('B', 'A', 'C', 'E', 'D'), ('B',
'A', 'D', 'C', 'E'), ('B', 'A', 'D', 'E', 'C'), ('B', 'A', 'E', 'C', 'D'), ('B', 'A', 'E', 'D', 'C'), ('B', 'C', 'A', 'D', 'E'), ('B', 'C', '
A', 'E', 'D'), ('B', 'C', 'D', 'A', 'E'), ('B', 'C', 'D', 'E', 'A'), ('B', 'C', 'E', 'A', 'D'), ('B', 'C', 'E', 'D', 'A'), ('B', 'D', 'A', 'C
', 'E'), ('B', 'D', 'A', 'E', 'C'), ('B', 'D', 'C', 'A', 'E'), ('B', 'D', 'C', 'E', 'A')]
--------------  Done  ---------------
```

Figure 3

Very good, so far, a random or random graph has been produced, and in Figure 3, it is now time to do separation for the graph produced. In fact, this is done to faster the calculations, eliminate many of the tops, less tensioning of the interconnected vertices, and faster the result.

```
Split=int(input("Please Enter a number for Splite List (1-length(List)) , example
= 3 : "))
```

Using a function of a number, we can enter the number 1, that is, the entire position or random graph is to be checked and processed, but we can enter the number 2 to halve the entire lane or the default number 3 so that the whole Graph paths from three sections review and process only one section. Of course, the entered number must be proportional to the ratio, if the number is large, the result to the set of very low and non -calculating lifts and ultimately to the result.

```
print('--------------   Processing   ---------------')
```

According to Figure 4, the process of graph paths begins and is divided into the Split variable using the LIS3 variable LEN function, which includes the entire direction of the graph. This helps to calculate the shortest and longest graphs of the graph faster. On the other hand, some of the graphics directions are removed in the review and processing cycle and are ignored.

```
for i in range(0,int(len(List3)//split)):
     List4.append(List3[i])
```

Using the List4 variable, the residual graphs are stored and processed the shortest and longest graphs based on the List4 variable.

```
print("List4 = \n",List4)
```

The filtering is successfully performed and the List4 variable contains different graphs from the source point to the destination point.

The next and final stage of graph production is the two nesting rings for the outer loop of the graph of the graph and a copy of the paths in the List4 variable is placed using the List function to the List5_2 variable. In the inner loop this time a copy of the paths is placed in the List4 variable using the LIST function to the List5 variable. Then, using the Reverse function, the List5 variable content is reversed and then put a copy of the List5 variable content in the List6 variable.

```
for j in range(0,len(List4)):
     List5_2=list(List4[j])
     for i in range(0,len(List4)):
         List5=list(List4[i])
         List5.reverse()
         List6=List5.copy()
```

The reason for this copy of the variables is determined by the if condition, which is to re-process the graphs of the graphs and place it in the ListFinal variable, so that the algorithm is subsequently calculated.

```
if List5_2==List6 :
   ListFinal.append(List4[i])
```

This line adds direct graphs to the ListFinal variable.

```
   ListFinal.append(List6)
```

This line adds the opposite directions to the ListFinal variable.

In fact, the inner ring and the exterior ring help help produce direct and reverse directions in the graph. According to Figure 4, it is clear that there is a set of paths in the form of a list and a list, the reason for the Taple is the Permutations function to produce different graphs.

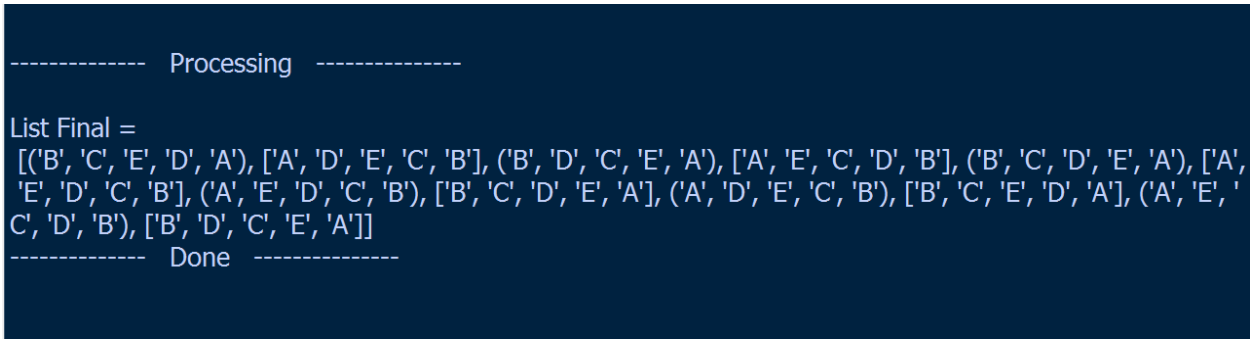

Figure 4

To filter the tops and to remain just a list of graphs, consider a few lines below.

```
ListFinal_help=list()
```

The listfinal_help variable is the empty list type.

```
EmptyTuple=tuple()
```

Emptytuple variable of type 1 is empty.

```
for i in ListFinal:
    if type(i) != type(EmptyTuple):
       ListFinal_help.append(i)
```

In the For loop, each of the top -of -the -type graphs is reviewed, and if the graphs are opposed to the type of top are after the list type, and the list of the list is added to the ListFinal_help variable.

```
ListFinal.clear()
```

After finishing the for loop, this line erases the entire content of the ListFinal variable using the Clear function.

```
ListFinal=ListFinal_help.copy()
```

Using the copy function of the ListFinal_help variable, a copy of the contents of the ListFinal_help variable, which has graph vertices of the list type, is poured into the ListFinal variable, and the filtering of tuple graph vertices is done.

```
print(“List Final =\n”, ListFinal)
```

The final list of non-repetitive graph paths is generated in the ListFinal variable as a separate list.

4-Solve the shortest and longest path of the graph

According to Figure 5, we must determine the starting point and end point of the graph routing, and by generating a random graph or permutation graph, the starting point and the end point of the graph must be assigned in str1 and str2 variables based on the input function. Default points can be start and end, default B value and H value.

```
str1=”Please enter start point (example : B) : “
Start=input(str1)
str2=”Please enter end point (example : H) : “
End=input(str2)

print(‘--------------   Processing   ---------------')
```

```
Please enter start point (example : B) : A
Please enter end point (example : H) : D

--------------  Processing  --------------

My Path List =  [['A', 'C', 'D', 'E', 'B'], ['A', 'C', 'E', 'D', 'B'], ['A', 'D', 'C', 'E', 'B'], ['A', 'D', 'E', 'B', 'C'], ['A', 'D', '
E', 'C', 'B'], ['A', 'E', 'C', 'D', 'B'], ['A', 'E', 'D', 'B', 'C'], ['A', 'E', 'D', 'C', 'B'], ['A', 'D', 'C', 'E', 'B'], ['A', 'D', 'E', 'B
', 'C'], ['A', 'D', 'E', 'C', 'B'], ['B', 'D', 'C', 'E', 'A'], ['B', 'D', 'E', 'A', 'C'], ['B', 'D', 'E', 'C', 'A'], ['C', 'A', 'D', 'E', 'B']
, ['C', 'A', 'E', 'D', 'B'], ['A', 'C', 'D', 'E', 'B'], ['A', 'E', 'D', 'B', 'C'], ['A', 'E', 'D', 'C', 'B'], ['B', 'C', 'D', 'E', 'A'], ['B',
'E', 'D', 'A', 'C'], ['B', 'E', 'D', 'C', 'A'], ['C', 'A', 'D', 'E', 'B'], ['C', 'B', 'D', 'E', 'A'], ['A', 'C', 'E', 'D', 'B'], ['A', 'E', '
C', 'D', 'B'], ['B', 'C', 'E', 'D', 'A'], ['B', 'D', 'E', 'A', 'C'], ['B', 'E', 'C', 'D', 'A'], ['B', 'E', 'D', 'A', 'C'], ['C', 'A', 'E', 'D
', 'B'], ['C', 'B', 'E', 'D', 'A'], ['B', 'C', 'D', 'E', 'A'], ['B', 'C', 'E', 'D', 'A'], ['B', 'D', 'C', 'E', 'A'], ['B', 'D', 'E', 'C', 'A']
, ['B', 'E', 'C', 'D', 'A'], ['B', 'E', 'D', 'C', 'A'], ['C', 'B', 'D', 'E', 'A'], ['C', 'B', 'E', 'D', 'A']]
--------------  Done  --------------
```

Figure 5

According to Figure 5, the calculation process begins, and the outer for loop considers the length of the graph based on the available points, and the inner for loop considers the number of unique paths of the graph.

```
For j in range(0,e-s):
    for i in range(0,len(ListFinal)):
         if Start==ListFinal[i][j] or End==ListFinal[i][j]:
             MyPathList.append(ListFinal[i])
```

Based on the if statement, when the Start variable is equal to the starting unique graph path point or when the End variable is equal to the ending unique graph path point. So, the path of the unique graph has the start point and the end point of the path we entered, and this path is added in the MyPathList variable.

```
print(“My Path List = “,MyPathList)
```

The MyPathList variable has graphic paths that we entered the start point and the end point using the input function, and it is assigned in the Start variable and the End variable.

The outer for loop with variable i traverses the entire list of route graphs with a specific start and end point. The internal for loop with the variable j navigates the length of the start point and the end point, and if the start point in the Start variable is equal to the top point of the list variable i with the index of the graph path length, then the internal for loop with the variable k starts executing and The path length is checked from variable index j to the end of path length. If the list variable i with the index variable k is equal to the end point of the End variable, the point i is placed in the variable MyPathList2 and all the inner loops are broken using the break command.

```
for i in MyPathList:
    for j in range(0,e-s):
         if Start==i[j]:
            for k in range(j,e-s):
                 if i[k]==End:
                     MyPathList2.append(i)
                     break
            break
```

```
print(“My Path List 2 = “,MyPathList2)
```

According to Figure 6, among the different paths, there are only paths in the MyPathList2 variable that start from the starting point based on the Start variable and end based on the End variable, although the end point can be based on the End variable in the unending path.

```
---------- Processing ----------
My Path List 2 =  [['A', 'C', 'D', 'E', 'B'], ['A', 'C', 'E', 'D', 'B'], ['A', 'D', 'C', 'E', 'B'], ['A', 'D', 'E', 'B', 'C'], ['A', 'D'
, 'E', 'C', 'B'], ['A', 'E', 'C', 'D', 'B'], ['A', 'E', 'D', 'B', 'C'], ['A', 'E', 'D', 'C', 'B'], ['A', 'D', 'C', 'E', 'B'], ['A', 'D', 'E',
'B', 'C'], ['A', 'D', 'E', 'C', 'B'], ['C', 'A', 'D', 'E', 'B'], ['C', 'A', 'E', 'D', 'B'], ['A', 'C', 'D', 'E', 'B'], ['A', 'E', 'D', 'B', '
C'], ['A', 'E', 'D', 'C', 'B'], ['C', 'A', 'D', 'E', 'B'], ['A', 'C', 'E', 'D', 'B'], ['A', 'E', 'C', 'D', 'B'], ['C', 'A', 'E', 'D', 'B']]
---------- Done ----------
```

Figure 6

The heart of the algorithm is here, and according to Figure 7, the calculation process begins.

Run=False

The Run variable is initially False.

for i in MyPathList2:
In the outer loop of MyPathList2, it resets the value of the local variable Sum to zero each time. Then the PathSum string variable takes a null value and the Run variable takes a False value. The Run variable helps to calculate the cost and path of the graph once at a time by hitting the starting point.

```
Sum=0
PathSum=””
Run=False
```

The inner loop with variable j traverses the length of the graph path, and whenever the vertex of the graph path is equal to the point of the Start variable and the Run variable is False, the Run variable takes the value True. Another inner loop starts running with the variable k based on the length of the graph path, and this time based on the vertex of the graph, if the vertex is equal to one of the list of the vertex of the graph, the cost of the vertex of the graph is calculated using the Sum variable. Using the PathSum variable, the path of the

graph vertices is considered from the starting point, and then the calculations are ended with the break command.

```
     For j in range(0,e-s):
        if i[j]==Start and Run==False :
             Run=True
             for k in range(0,e-s):
                 if i[j]==List[k]:

                        """Sum=Sum+List2[k]"""
```

This line, which is explained, causes the cost of the first vertex of the graph to be calculated. But because it is explained, it will not be calculated, and according to this algorithm, the amount of cost that is at the top of the starting point of the graph should not be calculated.

```
                        PathSum=PathSum+i[j]+” -> “
                        break
```

Next, if the vertex of the graph is not yet finished and of course the value of the Run variable is still True, it means that the starting point of the graph has been found. Another inner loop is executed with the variable k and based on the length of the graph path, it is checked that if the vertex of the graph is equal to one of the list of graph points, the cost of the vertex of the graph is calculated using the Sum variable. By using the variable PathSum, the path of the vertices of the graph is considered from the starting point, and then the calculations end with the break command, the string " -> " means the continuation of the path from the graph.

```
        elif i[j]!=End and Run :
              for k in range(0,e-s):
                 if i[j]==List[k]:
                        Sum=Sum+List2[k]
                        PathSum=PathSum+i[j]+” -> “
                        break
```

In the following, if the vertex of the graph is finished and of course the value of Run variable is still True, it means that the starting point of the vertex of the graph and the end point of the vertex of the graph have also been found. Another inner loop is executed with the variable k and based on the length of the graph path, it is checked that if the vertex of the graph is equal to one of the list of graph points, the cost of the vertex of the graph is also calculated using the Sum variable. Using the PathSum variable, the previously saved path along with the graph vertices from the start point to the end point is considered, and then the calculations are ended with the break command.

```
        elif i[j]==End and Run:
               for k in range(0,e-s):
                  if i[j]==List[k]:
                        Sum=Sum+List2[k]
                        PathSum=PathSum+i[j]+” . “
                        break
               break
```

Next, based on the existence of the external loop using the j variable, the cost value from the starting point to the end of the graph is placed in the AllListCost variable. Of course, the path traveled by the graph from the starting point to the end is also placed in the PathListCost variable. Then the list of all paths and costs of the graph from the starting point to the end is placed in the variable AllListCost and PathListCost, the string "." means the end of the traveled path of the graph.

```
     AllListCost.append(Sum)
     PathListCost.append(PathSum)
```

The first cost value of the graph path is placed in the AllListCost variable in the Min variable.

```
Min=AllListCost[0]
```

The value of zero is assigned in the Pos variable, this variable stores the position of the lowest cost path of the graph, which is the shortest path.

```
Pos=0
```

The for loop processes the graph based on the list of traversed paths and calculates the lowest cost from the start point to the end point of the graph, the Min variable considers the lowest cost and the Pos variable considers the position of the graph path.

```
for i in range(1,len(AllListCost)):
    if Min>AllListCost[i]:
        Min=AllListCost[i]
        Pos=i
```

The first cost value of the graph path is placed in the AllListCost variable in the Max variable.

```
Max=AllListCost[0]
Pos2=0
```

The value of zero is assigned to the Pos variable, this variable stores the position of the highest cost of the graph path, which is the longest path.

The for loop processes the graph based on the list of traversed paths and calculates the highest cost from the start point to the end point of the graph, the Max variable considers the highest cost and the Pos2 variable considers the position of the graph path.

```
for i in range(1,len(AllListCost)):
    if Max<AllListCost[i]:
        Max=AllListCost[i]
        Pos2=i
```

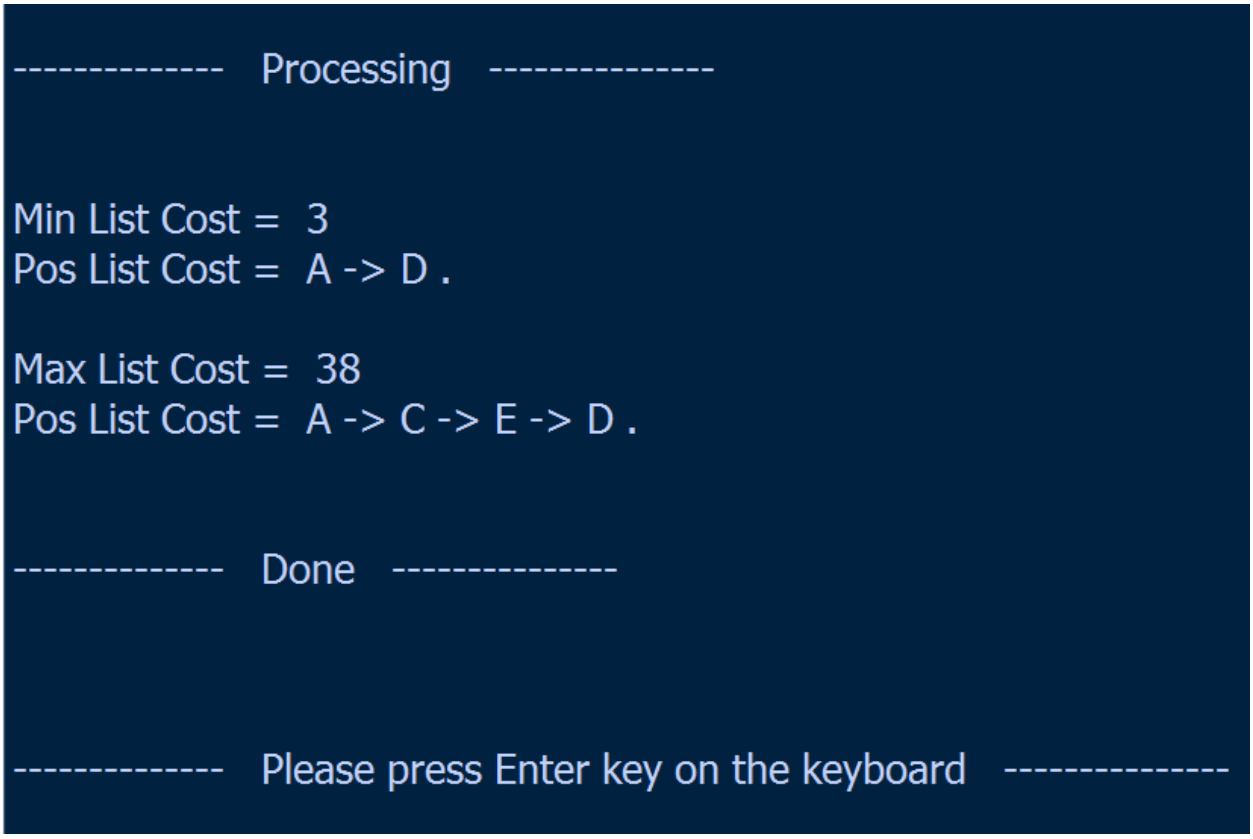

Figure 7

```
print("Min List Cost = ",Min)
```

This line shows the lowest cost path of the graph.

```
print("Pos List Cost = ",PathListCost[Pos])
```

This path line shows the least cost path of the graph, which is the shortest path of the graph.

```
print(“Max List Cost = “,Max)
```

This line shows the highest cost of the graph path.

```
print(“Pos List Cost = “,PathListCost[Pos2])
```

This path line displays the highest cost path of the graph, which is the longest path of the graph.

5- Conclusions

This algorithm has a cubic computing time $O(n^3)$ and linear memory consumption O(n), and the cost of the path is transferred from the vertex to the edge, and the transfer method is performed from the last vertex to the last edge, but the first vertex and the first edge are not transferred and are not calculated. According to figure 8, the explanation is given in the form of several examples. This algorithm can be improved by others, and algorithm codes are fully attached to this article in Python 3.12.

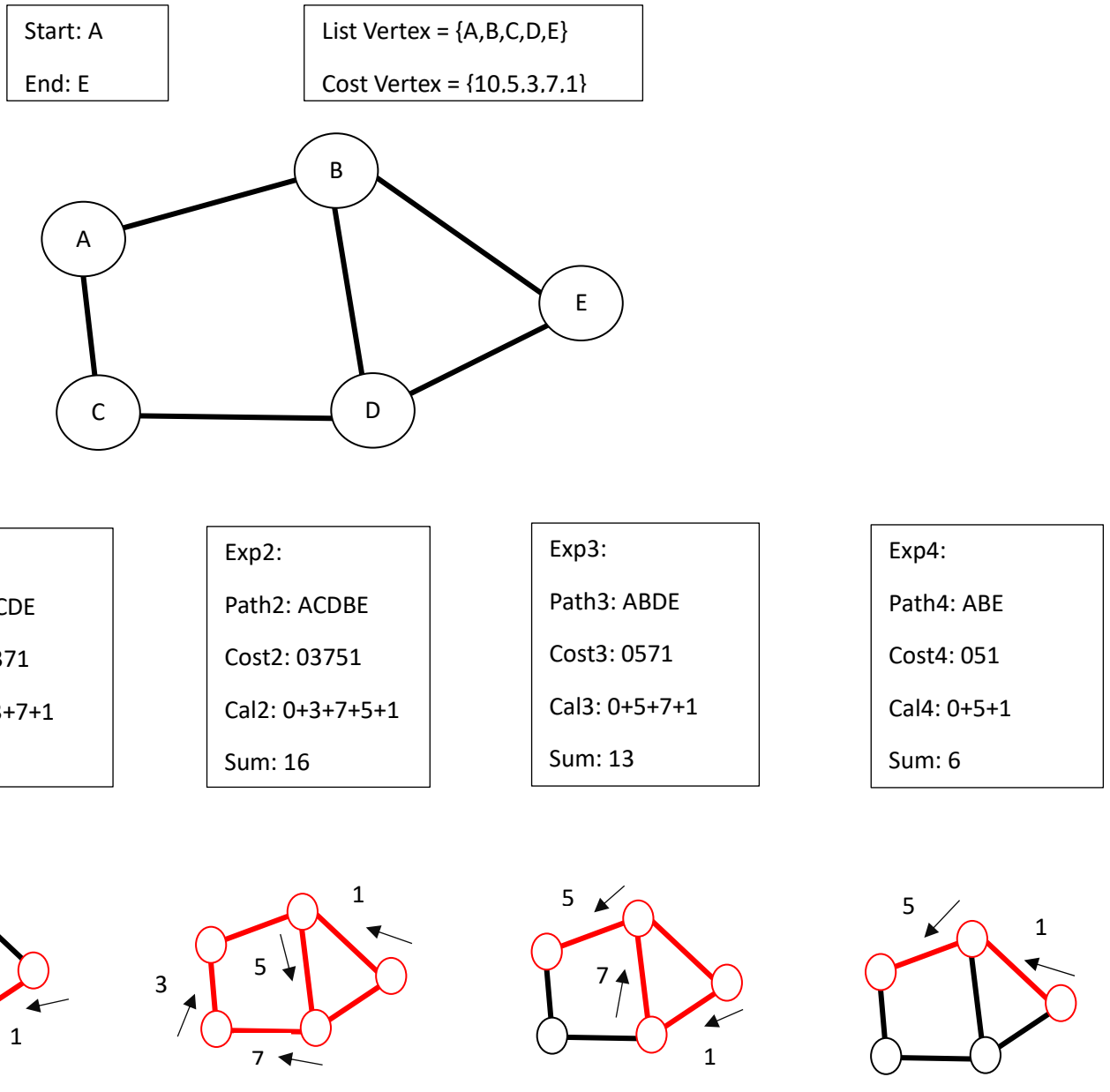

Figure 8

Declarations

I the undersigned declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere, and I'm the only author of this article.

Availability of data and materials: I declare that the collected data are original and all the data are collected by author in this article.

Conflicts of interest Competing interest: The author declare that there is no competing Interest associated with this publication.

Funding: I declare that this research is not supported or funded by any organization or individual Author bear all the expenses of this study by themselves So, this research is self-funded research of the author.

Author's contribution: author contributed to the study and conception completely.

Acknowledgements: -

Consent for publication: This article, plus statement and data is published by the author with full consent.

References

[1] Computer Networks, Andrew S. Tanenbaum, Fourth Edition, Prentice-Hall, 2003.

[2] Richard Neapolitan, Design Algorithms Book, July 29, 2003 -Foundations of Algorithms Using C++ Pseudocode.

[3] https://www.python.org/