CSE620: Assignment2

Sima Shafaei

Due: November 6, 2020

1. For the symmetrical TSP problem with the distance matrix given in Table 1 (where the i, j entry represents the distance form city i to city j) find the optimal tour using exhaustive search. Show all steps and all the suboptimal paths.

	City 1	City 2	City 3	City 4
City 1	∞	10	20	5
City 2	10	∞	2	10
City 3	20	2	∞	7
City 4	5	10	7	∞

Table 1: Distance matrix for Problem 1.

Solution:

We wrote a Python program for this question:

```python
import math
import itertools as iter

distance=[[math.inf,10,20,5],
         [10,math.inf,2,10],
         [20,2,math.inf,7],
         [5,10,7,math.inf]]

cities=[1,2,3,4]
n=4

tour=list(iter.permutations(cities))
num_of_sol=len(possible_solutions)
tourList=[""]*num_of_sol
tourLen=[0]*num_of_sol
for i in range(num_of_sol):
    pathLen=0
    path="city" +str(tour[i][0])+"-->"
    for j in range(n):
        pathLen+=distance[tour[i][j]-1][tour[i][(j+1)%n]-1]
        path += "city" +str(tour[i][(j+1)%n])
        if (j+1)!=n:
            path+="-->"

    tourList[i]=path
    tourLen[i]=pathLen
    print("tour: "+path +"     length:"+str(pathLen))

minPath=min(tourLen)
res = [i for i, j in enumerate(tourLen) if j == minPath]

print("==========================================")
print("the length of minimum tour is:"+str(minPath))
print("and the list of optimal tours is as follows:")
for i in range(len(res)):
    print(tourList[res[i]])
```

The result is as follows:

tour: city1-->city2-->city3-->city4-->city1 length:24
tour: city1-->city2-->city4-->city3-->city1 length:47
tour: city1-->city3-->city2-->city4-->city1 length:37
tour: city1-->city3-->city4-->city2-->city1 length:47
tour: city1-->city4-->city2-->city3-->city1 length:37
tour: city1-->city4-->city3-->city2-->city1 length:24
tour: city2-->city1-->city3-->city4-->city2 length:47
tour: city2-->city1-->city4-->city3-->city2 length:24
tour: city2-->city3-->city1-->city4-->city2 length:37
tour: city2-->city3-->city4-->city1-->city2 length:24
tour: city2-->city4-->city1-->city3-->city2 length:37
tour: city2-->city4-->city3-->city1-->city2 length:47
tour: city3-->city1-->city2-->city4-->city3 length:47
tour: city3-->city1-->city4-->city2-->city3 length:37
tour: city3-->city2-->city1-->city4-->city3 length:24
tour: city3-->city2-->city4-->city1-->city3 length:37
tour: city3-->city4-->city1-->city2-->city3 length:24
tour: city3-->city4-->city2-->city1-->city3 length:47
tour: city4-->city1-->city2-->city3-->city4 length:24
tour: city4-->city1-->city3-->city2-->city4 length:37
tour: city4-->city2-->city1-->city3-->city4 length:47
tour: city4-->city2-->city3-->city1-->city4 length:37
tour: city4-->city3-->city1-->city2-->city4 length:47
tour: city4-->city3-->city2-->city1-->city4 length:24

*Based on above result the length of minimum tour is **24** and the list of all optimal tours is:*

city1-->city2-->city3-->city4-->city1
city1-->city4-->city3-->city2-->city1
city2-->city1-->city4-->city3-->city2
city2-->city3-->city4-->city1-->city2
city3-->city2-->city1-->city4-->city3
city3-->city4-->city1-->city2-->city3
city4-->city1-->city2-->city3-->city4
city4-->city3-->city2-->city1-->city4

2. For the symmetrical TSP problem with the distance matrix given in Table 2 (where the i, j entry represents the distance form city i to city j) find the optimum tour using hill climbing, with a 2-change neighborhood (i.e. swap 2 non-adjacent edges), starting at path 1-3-2-4-5-1. Show all steps

	City 1	City 2	City 3	City 4	City 5
City 1	∞	10	20	5	10
City 2	10	∞	2	10	6
City 3	20	2	∞	7	1
City 4	5	10	7	∞	20
City 5	10	6	1	20	∞

Table 2: Distance matrix for Problem 2.

Solution:

To solve this problem again we create a python program:

```python
import math
def cost(currentPath,distance,n):
    cost=0
    for i in range(n):
        cost+=distance[currentPath[i]-1][currentPath[(i+1)%n]-1]
    return cost



distance=[[math.inf,10,20,5,10],
         [10,math.inf,2,10,6],
         [20,2,math.inf,7,1],
         [5,10,7,math.inf,20],
         [10,6,1,7,20,math.inf]]

cities=[1,2,3,4,5]
n=5
currentPath=[1,3,2,4,5]
```

```python
while(True):
    iteration+=1
    print(currentPath,end=" ")
    costCurrent=cost(currentPath,distance,n)
    print(costCurrent)
    k=0
    neighbor=[None]*5
    c=[0]*5
    has_swaped=[[0]*(n+1)]*(n+1)
    for i in range(n):
        for j in range(n-3):
            t=(j+i+2)%n
            if has_swaped[currentPath[i]][currentPath[t]]==0:
                has_swaped[currentPath[i]][currentPath[t]]=1
                neighbor[k]=currentPath.copy()
                temp=neighbor[k][(i+1)%n]
                neighbor[k][(i+1)%n]=neighbor[k][t]
                neighbor[k][t]=temp
                print("swap:"+str(currentPath[i])+"->"+str(currentPath[(i+1)%n])+
                      " and "+str(currentPath[t])+"->"+str(currentPath[(t+1)%n]),end=" ")
                print(neighbor[k], end="  ")
                c[k]=cost(neighbor[k],distance,n)
                print("cost="+str(c[k]) )
                k+=1
    minIndex=c.index(min(c))
    if c[minIndex]<costCurrent:
        currentPath=neighbor[minIndex].copy()
    else:
        break
```

The initial path is :[1,3,2,4,5,1], with Cost = 62.

The first group of neighbors in first iteration, and their cost are:

swap:1->3 and 2->4: path= [1, 2, 3, 4, 5,1] cost=49
swap:1->3 and 4->5: path= [1, 4, 2, 3, 5,1] cost=28
swap:3->2 and 5->1: path= [1, 3, 5, 4, 2,1] cost=48
swap:2->4 and 1->3: path= [4, 3, 2, 1, 5,4] cost=36
swap:4->5 and 3->2: path= [1, 5, 2, 4, 3,1] cost=53

The path=[1, 4, 2, 3, 5,1] *with cost=28 has the minimum weight and will be selected for the next iteration. The neighbors of selected path in second iteration are:*

swap:1->4 and 2->3 [1, 2, 4, 3, 5] cost=38
swap:1->4 and 3->5 [1, 3, 2, 4, 5] cost=62
swap:4->2 and 5->1 [1, 4, 5, 3, 2] cost=38
swap:2->3 and 1->4 [3, 4, 2, 1, 5] cost=38
swap:3->5 and 4->2 [1, 5, 2, 3, 4] cost=30

As we can see there is no path with cost less than 28, so we can stop the search and the optimum solution would be : path=[1, 4, 2, 3, 5,1] and Cost=28

Problems 3, 4, and 5 refer to the asymmetrical TSP with the distance matrix given in Table 3.

	City 1	City 2	City 3	City 4
City 1	∞	20	10	5
City 2	15	∞	2	10
City 3	5	7	∞	7
City 4	10	1	4	∞

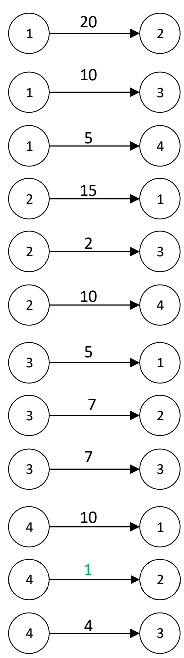
Table 3: Asymmetric Distance matrix for Problems 3, 4, and 5.

Problems 3,4,5: Find the best path using incremental arc addition and the following approaches:

3. Greedy: show cost for all neighbors and all steps.

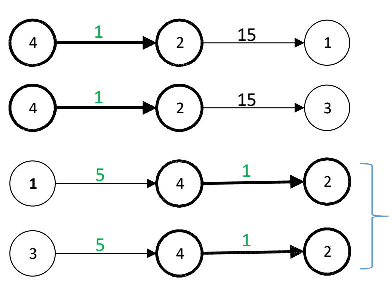
Solution:

in first iteration we have to select the first edge and select the one with minimum weight:



Among all possible selection 4->2 has the minimum weight and will be selected in the first iteration.

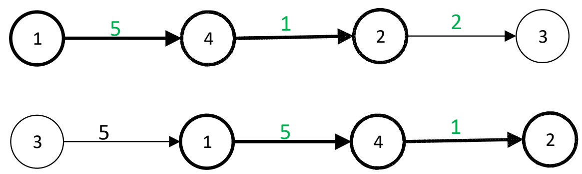
.Iteration 2:



In next steps, in each iteration, we have to add one node to our connected path that would not create a loop and select the minimum weight among all possible paths.

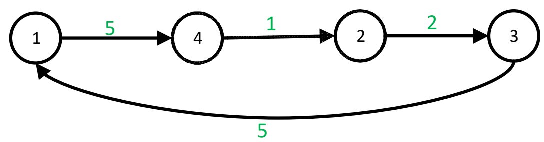
In this iteration we have two winners (1-4 and 3-4). We selected the first one in the list.

Iteration 3:



In this iteration the winner is 2->3

In this iteration winner is 2->3. Finally, when we added all the nodes we have to return to the origin therefore we will add 3->1 to complete the tour:



The total length of the final tour is 13.

4. Branch and bound using methodology presented in class.

Solution:

- *Estimating Lower Bound*

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	20	10	5
City 2	15	∞	2	10
City 3	5	7	∞	7
City 4	10	1	4	∞

Find min for each row

Min
5
2
5
1

LB=$\sum Min\ of\ each\ row = 13$

Subtract each row from min

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	15	5	0
City 2	13	∞	0	8
City 3	0	2	∞	2
City 4	9	0	3	∞

Find min for each column

Min	0	0	0	0

LB=$\sum Min\ of\ each\ Col + 13 = 13$

Subtract each con from its min

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	15	5	0
City 2	13	∞	0	8
City 3	0	2	∞	2
City 4	9	0	3	∞

==

Most influential arc:

1->4 : LB+(5+2)

2->3: LB+(8+3) Max

3->1: LB+(2+9) Max

4->2: LB+(3+2)

We can choose either 3->1 or 2->3 for branching. We will select 3->1

LB=13+11=24

==

include 3->1

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	15	∞	0
City 2	∞	∞	0	8
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

Find min for each row

Min
0
0
0

$\sum Min\ of\ each\ row = 0$

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	15	∞	0
City 2	∞	∞	0	8
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

Find min for each column

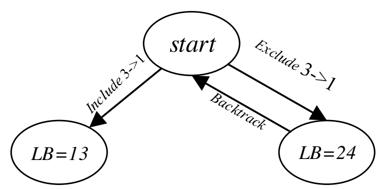
Min	∞	0	0	0

$\sum Min\ of\ each\ col + 0 = 0$

Subtract each con from its min

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	15	∞	0
City 2	∞	∞	0	8
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

LB=13+0=13



==

Most influential arc:

1->4 : LB+(15+8) Max

2->3: LB+(8+3)

4->2: LB+(3+15)

1->4 will be selected

LB=13+23=36

==

include 1->4

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

Find min for each row

Min
0
0

$\sum Min\ of\ each\ row = 0$

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

Find min for each column

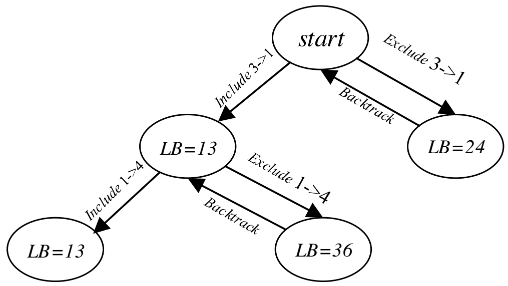
Min	∞	0	0	0

$\sum Min\ of\ each\ col + 0 = 0$

Subtract each con from its min

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	0	3	∞

LB=13+0=13



==

Most influential arc:

2->3: LB+(3)

4->2: LB+(3)

We can choose either 4->2 or 2->3 for branching. We will select 4->2

LB=13+3=16

==

include 4->2

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	∞	∞	∞

Find min for each row

Min
0

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	∞	∞	∞

$\sum Min\ of\ each\ row = 0$

Find min for each column

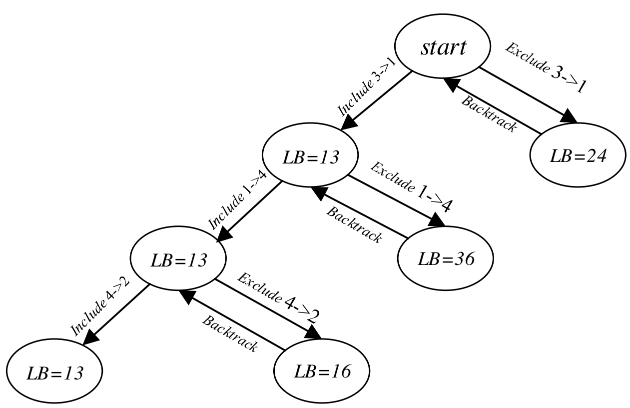
Min			0	

$\sum Min\ of\ each\ col + 0 = 0$

Subtract each con from its min

	City 1	*City 2*	*City 3*	*City 4*
City 1	∞	∞	∞	∞
City 2	∞	∞	0	∞
City 3	∞	∞	∞	∞
City 4	∞	∞	∞	∞

LB=13+0=13



==

We need only one more arc to back to the first city and it would be 2->3. Therefore, the optimal path is: **3->1->4->2->3**

5. Dynamic programming.

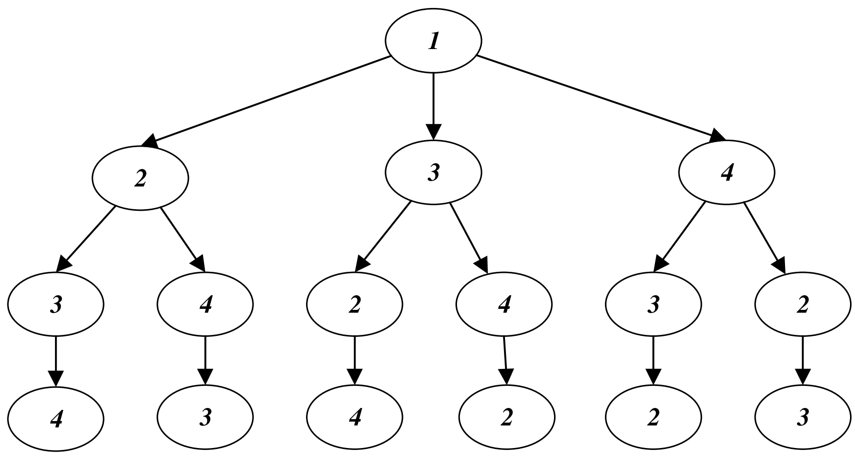
Solution:

The general formula that followed by dynamic programming for TSP is:

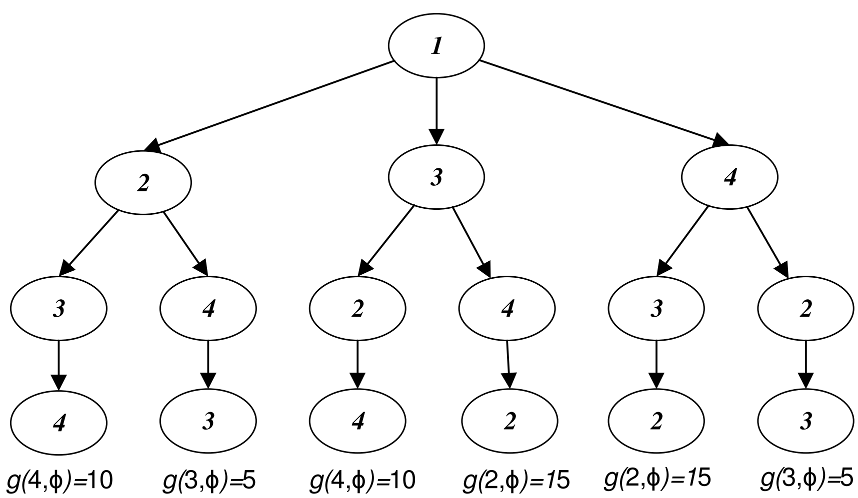
$$g(i, S) = \min_{k \in S}\{d_{ik} + g(k, S - \{k\})\}$$

In this problem we start with city 1 therefore, i=1 and S={2,3,4} and $g(j, \phi) = d(j, i)$

We start with creating tree and compute it from bottom to top, following tree shows all possible tour in four cities:



First level in tree:



Second level in the tree:

$g(3, \{4\}) = min\{d_{34} + g(4, \phi)\} = 7 + 10 = 17$

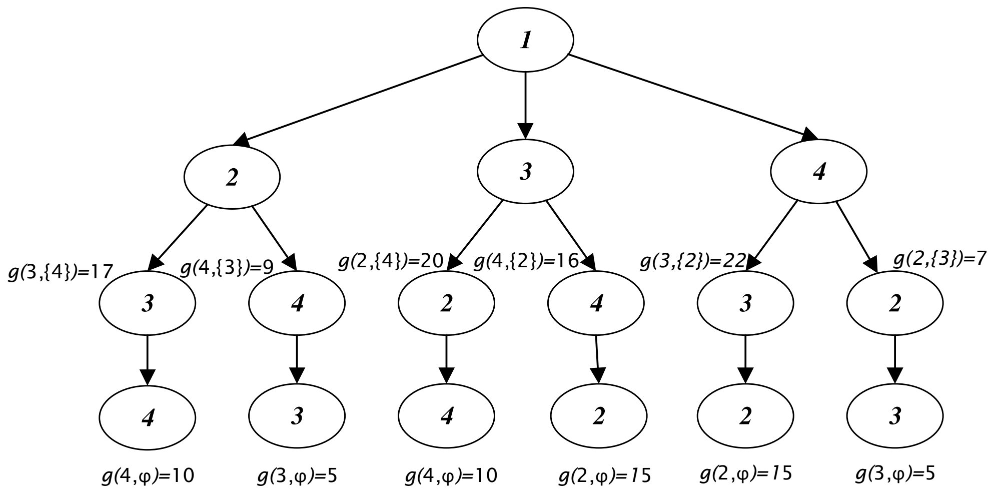
$g(4, \{3\}) = min\{d_{43} + g(3, \phi)\} = 4 + 5 = 9$

$g(2, \{4\}) = min\{d_{24} + g(4, \phi)\} = 10 + 10 = 20$

$g(4, \{2\}) = min\{d_{42} + g(2, \phi)\} = 1 + 15 = 16$

$g(3, \{2\}) = min\{d_{32} + g(2, \phi)\} = 7 + 15 = 22$

$g(2,\{3\}) = min\{d_{23} + g(3,\phi)\} = 2 + 5 = 7$



third level in the tree:

$g(2,\{3,4\}) = min\{d_{23} + g(3,\{4\}), d_{24} + g(4,\{3\})\} = min\{2 + 17,10 + 9\} = min\{19,19\} = 19$

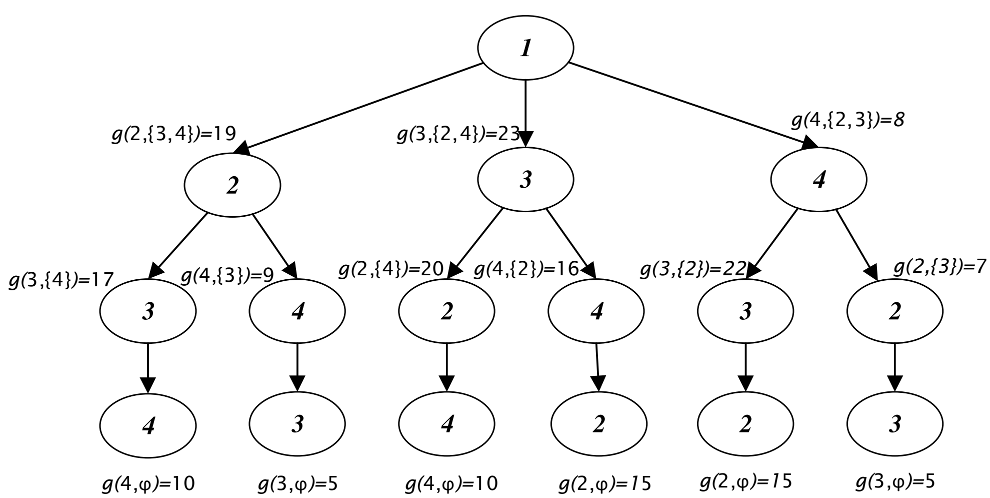
(minimum come from k=3 or 4)

$g(3,\{2,4\}) = min\{d_{32} + g(2,\{4\}), d_{34} + g(4,\{2\})\} = min\{7 + 20,7 + 16\} = min\{27,23\} = 23$

(minimum come from k=4)

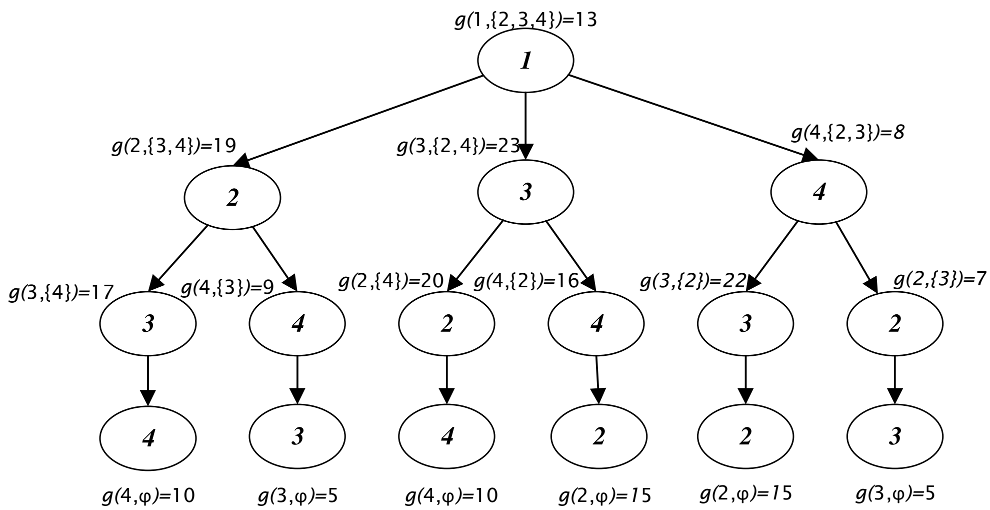
$g(4,\{3,2\}) = min\{d_{43} + g(3,\{2\}), d_{42} + g(2,\{3\})\} = min\{4 + 22,1 + 7\} = min\{26,8\} = 8$

(minimum come from k=2)



The root of tree:

$g(1,\{2,3,4\}) = min\{d_{12} + g(2,\{3,4\}), d_{13} + g(3,\{2,4\}), d_{14} + g(4,\{2,3\})\} = min\{20 + 19,10 + 23,5 + 8\} = min\{39,33,13\} = 13$ *(minimum come from k=4)*



Based on where minimum came from in each step the shortest tour is: 1->4->2->3->1