Report of Data Structure

Graph Theory

Name: Linfeng Liu

No.: 519261910040

Date: 2022.5.21

Score: ______________

SHANGHAI JIAOTONG UNIVERSITY
SJTU-Paris Elite Institute of Technology

Contents

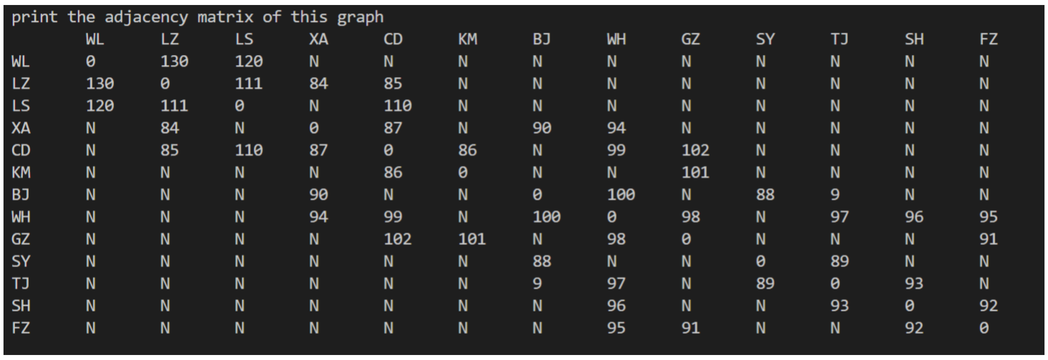
1 Task1: Graph Class 3
1.1 AM 3
1.2 AL 3
2 Task2: Sorting 4
2.1 Heapsort 4
2.2 Mergesort 4
3 Task3: MST 5
3.1 Two algorithms 5
3.2 Result Demo 5
4 Task4: Minimal Distance Path 6
5 Summary 7

1 Task1: Graph Class

As a preparation of all works later, we try to describe a graph in two ways: Adjacency List and Adjacency Matrix. We design two class, named as **Graphmatrix** and **Graphlist** respectively. We demonstrate this two class in this part. Before the design of these two classed, we design a class **Edge** with class members: **depart**, **destin** and **weight**, which represent respectively the start point, end point and the weight of the edge. In order to visit the private members of edge conveniently, we set **Graphmatrix** and **Graphlist** the friend class of **Edge**.

1.1 AM

In the class **Graphmatrix**, we can construct an arbitrary graph and its adjacency matrix with an array of the edges and an array of vertices. (The array of vertices can be generated by function **Edges2Nodes**. The adjacency matrix of the graph is demonstrated in Figure:1. We override the output operation to print this matrix.

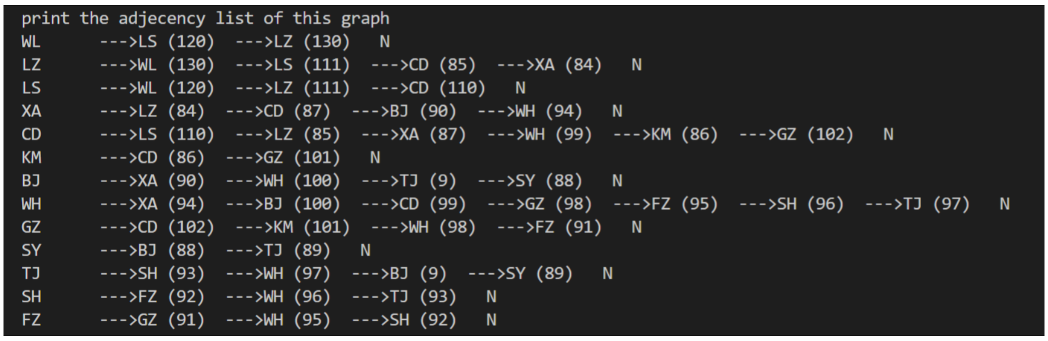


```
print the adjacency matrix of this graph
        WL      LZ      LS      XA      CD      KM      BJ      WH      GZ      SY      TJ      SH      FZ
WL      0       130     120     N       N       N       N       N       N       N       N       N       N
LZ      130     0       111     84      85      N       N       N       N       N       N       N       N
LS      120     111     0       N       110     N       N       N       N       N       N       N       N
XA      N       84      N       0       87      N       90      94      N       N       N       N       N
CD      N       85      110     87      0       86      N       99      102     N       N       N       N
KM      N       N       N       N       86      0       N       N       101     N       N       N       N
BJ      N       N       N       90      N       N       0       100     N       88      9       N       N
WH      N       N       N       94      99      N       100     0       98      N       97      96      95
GZ      N       N       N       N       102     101     N       98      0       N       N       N       91
SY      N       N       N       N       N       N       88      N       N       0       89      N       N
TJ      N       N       N       N       N       N       9       97      N       89      0       93      N
SH      N       N       N       N       N       N       N       96      N       N       93      0       92
FZ      N       N       N       N       N       N       N       95      91      N       N       92      0
```

Figure 1: Adjacency Matrix

1.2 AL

In the class **Graphlist**, we can construct an arbitrary graph and its adjacency list with an array of the edges and an array of vertices.(The array of vertices can be generated by function **Edges2Nodes**. The adjacency list of the graph is demonstrated in Figure:1. We override the output operation to print this list.



```
print the adjecency list of this graph
WL      --->LS (120)  --->LZ (130)   N
LZ      --->WL (130)  --->LS (111)  --->CD (85)  --->XA (84)   N
LS      --->WL (120)  --->LZ (111)  --->CD (110)   N
XA      --->LZ (84)  --->CD (87)  --->BJ (90)  --->WH (94)   N
CD      --->LS (110)  --->LZ (85)  --->XA (87)  --->WH (99)  --->KM (86)  --->GZ (102)   N
KM      --->CD (86)  --->GZ (101)   N
BJ      --->XA (90)  --->WH (100)  --->TJ (9)  --->SY (88)   N
WH      --->XA (94)  --->BJ (100)  --->CD (99)  --->GZ (98)  --->FZ (95)  --->SH (96)  --->TJ (97)   N
GZ      --->CD (102)  --->KM (101)  --->WH (98)  --->FZ (91)   N
SY      --->BJ (88)  --->TJ (89)   N
TJ      --->SH (93)  --->WH (97)  --->BJ (9)  --->SY (89)   N
SH      --->FZ (92)  --->WH (96)  --->TJ (93)   N
FZ      --->GZ (91)  --->WH (95)  --->SH (92)   N
```

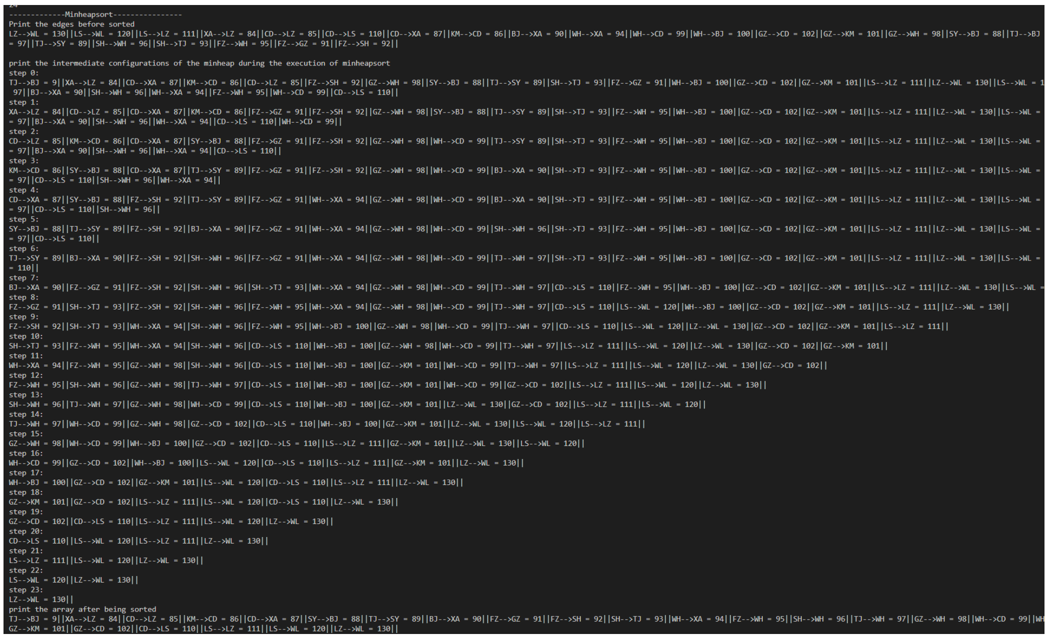
Figure 2: Adjacency List

2 Task2: Sorting

In this part, we study the sorting methods in order to sort the array of edges in the graph. First of all, we design a **GetEdgeAM** to get an array of edges from an instance of **Graphmatrix**.

2.1 Heapsort

We design a class **MinHeap**, which can maintain an array according to the properties of a heap. We remove the minimal element of the heap each time and put it in the target array(used to restore the sorted array). The intermediate configuration of the heap and the result of the sorting are all demonstrated in Figure 3. The complexity of heapsort is $\mathbf{O}(mlog(m))$ where m the number of edges in the array we sort. Because we remove all m element in the array and the complexity of remove the root of a heap is $\mathbf{O}log(m)$. That is why the total complexity of heapsort is $\mathbf{O}(mlog(m))$.



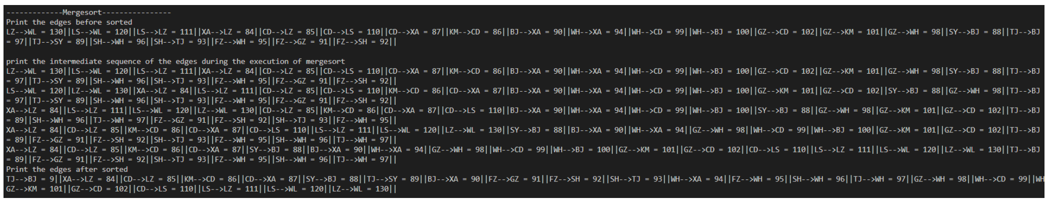
```
24
-------------Minheapsort---------------
Print the edges before sorted
LZ-->WL = 130||LS-->WL = 120||LS-->LZ = 111||XA-->LZ = 84||CD-->LZ = 85||CD-->LS = 110||CD-->XA = 87||KM-->CD = 86||BJ-->XA = 90||WH-->XA = 94||WH-->CD = 99||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||GZ-->WH = 98||SY-->BJ = 88||TJ-->BJ
= 97||TJ-->SY = 89||SH-->WH = 96||SH-->TJ = 93||FZ-->WH = 95||FZ-->GZ = 91||FZ-->SH = 92||

print the intermediate configurations of the minheap during the execution of minheapsort
step 0:
TJ-->BJ = 9||XA-->LZ = 84||CD-->XA = 87||KM-->CD = 86||CD-->LZ = 85||FZ-->SH = 92||GZ-->WH = 98||SY-->BJ = 88||TJ-->SY = 89||SH-->TJ = 93||FZ-->GZ = 91||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL = 1
 97||BJ-->XA = 90||SH-->WH = 96||WH-->XA = 94||FZ-->WH = 95||WH-->CD = 99||CD-->LS = 110||
step 1:
XA-->LZ = 84||CD-->LZ = 85||CD-->XA = 87||KM-->CD = 86||FZ-->GZ = 91||FZ-->SH = 92||GZ-->WH = 98||SY-->BJ = 88||TJ-->SY = 89||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 97||BJ-->XA = 90||SH-->WH = 96||WH-->XA = 94||CD-->LS = 110||WH-->CD = 99||
step 2:
CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||SY-->BJ = 88||FZ-->GZ = 91||FZ-->SH = 92||GZ-->WH = 98||WH-->CD = 99||TJ-->SY = 89||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 97||BJ-->XA = 90||SH-->WH = 96||WH-->XA = 94||CD-->LS = 110||
step 3:
KM-->CD = 86||SY-->BJ = 88||CD-->XA = 87||TJ-->SY = 89||FZ-->GZ = 91||FZ-->SH = 92||GZ-->WH = 98||WH-->CD = 99||BJ-->XA = 90||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 97||CD-->LS = 110||SH-->WH = 96||WH-->XA = 94||
step 4:
CD-->XA = 87||SY-->BJ = 88||FZ-->SH = 92||TJ-->SY = 89||FZ-->GZ = 91||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||BJ-->XA = 90||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 97||CD-->LS = 110||SH-->WH = 96||
step 5:
SY-->BJ = 88||TJ-->SY = 89||FZ-->SH = 92||BJ-->XA = 90||FZ-->GZ = 91||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||SH-->WH = 96||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 97||CD-->LS = 110||
step 6:
TJ-->SY = 89||BJ-->XA = 90||FZ-->SH = 92||SH-->WH = 96||FZ-->GZ = 91||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||TJ-->WH = 97||SH-->TJ = 93||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
= 110||
step 7:
BJ-->XA = 90||FZ-->GZ = 91||FZ-->SH = 92||SH-->WH = 96||SH-->TJ = 93||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||TJ-->WH = 97||CD-->LS = 110||FZ-->WH = 95||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||LS-->WL =
step 8:
FZ-->GZ = 91||SH-->TJ = 93||FZ-->SH = 92||SH-->WH = 96||FZ-->WH = 95||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||TJ-->WH = 97||CD-->LS = 110||LS-->WL = 120||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||LZ-->WL = 130||
step 9:
FZ-->SH = 92||SH-->TJ = 93||WH-->XA = 94||SH-->WH = 96||FZ-->WH = 95||WH-->BJ = 100||GZ-->WH = 98||WH-->CD = 99||TJ-->WH = 97||CD-->LS = 110||LS-->WL = 120||LZ-->WL = 130||GZ-->CD = 102||GZ-->KM = 101||LS-->LZ = 111||
step 10:
SH-->TJ = 93||FZ-->WH = 95||WH-->XA = 94||SH-->WH = 96||CD-->LS = 110||WH-->BJ = 100||GZ-->WH = 98||WH-->CD = 99||TJ-->WH = 97||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||GZ-->CD = 102||GZ-->KM = 101||
step 11:
WH-->XA = 94||FZ-->WH = 95||GZ-->WH = 98||SH-->WH = 96||CD-->LS = 110||WH-->BJ = 100||GZ-->KM = 101||WH-->CD = 99||TJ-->WH = 97||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||GZ-->CD = 102||
step 12:
FZ-->WH = 95||SH-->WH = 96||GZ-->WH = 98||TJ-->WH = 97||CD-->LS = 110||WH-->BJ = 100||GZ-->KM = 101||WH-->CD = 99||GZ-->CD = 102||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||
step 13:
SH-->WH = 96||TJ-->WH = 97||GZ-->WH = 98||WH-->CD = 99||CD-->LS = 110||WH-->BJ = 100||GZ-->KM = 101||LZ-->WL = 130||GZ-->CD = 102||LS-->LZ = 111||LS-->WL = 120||
step 14:
TJ-->WH = 97||WH-->CD = 99||GZ-->WH = 98||GZ-->CD = 102||CD-->LS = 110||WH-->BJ = 100||GZ-->KM = 101||LZ-->WL = 130||LS-->WL = 120||LS-->LZ = 111||
step 15:
GZ-->WH = 98||WH-->CD = 99||WH-->BJ = 100||GZ-->CD = 102||CD-->LS = 110||LS-->LZ = 111||GZ-->KM = 101||LZ-->WL = 130||LS-->WL = 120||
step 16:
WH-->CD = 99||GZ-->CD = 102||WH-->BJ = 100||LS-->WL = 120||CD-->LS = 110||LS-->LZ = 111||GZ-->KM = 101||LZ-->WL = 130||
step 17:
WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||LS-->WL = 120||CD-->LS = 110||LS-->LZ = 111||LZ-->WL = 130||
step 18:
GZ-->KM = 101||GZ-->CD = 102||LS-->LZ = 111||LS-->WL = 120||CD-->LS = 110||LZ-->WL = 130||
step 19:
GZ-->CD = 102||CD-->LS = 110||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||
step 20:
CD-->LS = 110||LS-->WL = 120||LS-->LZ = 111||LZ-->WL = 130||
step 21:
LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||
step 22:
LS-->WL = 120||LZ-->WL = 130||
step 23:
LZ-->WL = 130||
print the array after being sorted
TJ-->BJ = 9||XA-->LZ = 84||CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||SY-->BJ = 88||TJ-->SY = 89||BJ-->XA = 90||FZ-->GZ = 91||FZ-->SH = 92||SH-->TJ = 93||WH-->XA = 94||FZ-->WH = 95||SH-->WH = 96||TJ-->WH = 97||GZ-->WH = 98||WH-->CD = 99||WH
GZ-->KM = 101||GZ-->CD = 102||CD-->LS = 110||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||
```

Figure 3: process of heapsort

2.2 Mergesort

In order to demonstrate the intermediate sequences of edges during mergesort, we abandon the recursive implementation of mergesort and implement it with two loops. The outer loop determine the size of the merging part$(1, 2, 4, 8...)$, and we merge the array partially in the inner loop. The intermediate sequences of edges during mergesort and the result of the sorting are all demonstrated in Figure 4. The complexity of mergesort is $\mathbf{O}(mlog(m))$ where m the number of edges in the array we sort. Because the complexity of the outer loop is $log(m)$(for $2^{len} = m$). The complexity of the inner loop is m, we traverse all the element in the array in order to merge them partially. That is why the total complexity of mergesort is $\mathbf{O}(mlog(m))$.



```
-------------Mergesort----------------
Print the edges before sorted
LZ-->WL = 130||LS-->WL = 120||LS-->LZ = 111||XA-->LZ = 84||CD-->LZ = 85||CD-->LS = 110||CD-->XA = 87||KM-->CD = 86||BJ-->XA = 90||WH-->XA = 94||WH-->CD = 99||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||GZ-->WH = 98||SY-->BJ = 88||TJ-->BJ
= 97||TJ-->SY = 89||SH-->WH = 96||SH-->TJ = 93||FZ-->WH = 95||FZ-->GZ = 91||FZ-->SH = 92||

print the intermediate sequence of the edges during the execution of mergesort
LZ-->WL = 130||LS-->WL = 120||LS-->LZ = 111||XA-->LZ = 84||CD-->LZ = 85||CD-->LS = 110||CD-->XA = 87||KM-->CD = 86||BJ-->XA = 90||WH-->XA = 94||WH-->CD = 99||WH-->BJ = 100||GZ-->CD = 102||GZ-->KM = 101||GZ-->WH = 98||SY-->BJ = 88||TJ-->BJ
= 97||TJ-->SY = 89||SH-->WH = 96||SH-->TJ = 93||FZ-->WH = 95||FZ-->GZ = 91||FZ-->SH = 92||
LS-->WL = 120||LZ-->WL = 130||XA-->LZ = 84||LS-->LZ = 111||CD-->LZ = 85||CD-->LS = 110||KM-->CD = 86||CD-->XA = 87||BJ-->XA = 90||WH-->XA = 94||WH-->CD = 99||WH-->BJ = 100||GZ-->KM = 101||GZ-->CD = 102||SY-->BJ = 88||GZ-->WH = 98||TJ-->BJ
= 97||TJ-->SY = 89||SH-->WH = 96||SH-->TJ = 93||FZ-->WH = 95||FZ-->GZ = 91||FZ-->SH = 92||
XA-->LZ = 84||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||CD-->LS = 110||BJ-->XA = 90||WH-->XA = 94||WH-->CD = 99||WH-->BJ = 100||SY-->BJ = 88||GZ-->WH = 98||GZ-->KM = 101||GZ-->CD = 102||TJ-->BJ
= 89||SH-->WH = 96||TJ-->WH = 97||FZ-->GZ = 91||FZ-->SH = 92||SH-->TJ = 93||FZ-->WH = 95||
XA-->LZ = 84||CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||CD-->LS = 110||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||SY-->BJ = 88||BJ-->XA = 90||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||WH-->BJ = 100||GZ-->KM = 101||GZ-->CD = 102||TJ-->BJ
= 89||FZ-->GZ = 91||FZ-->SH = 92||SH-->TJ = 93||FZ-->WH = 95||SH-->WH = 96||TJ-->WH = 97||
XA-->LZ = 84||CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||SY-->BJ = 88||BJ-->XA = 90||WH-->XA = 94||GZ-->WH = 98||WH-->CD = 99||WH-->BJ = 100||GZ-->KM = 101||GZ-->CD = 102||CD-->LS = 110||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||TJ-->BJ
= 89||FZ-->GZ = 91||FZ-->SH = 92||SH-->TJ = 93||FZ-->WH = 95||SH-->WH = 96||TJ-->WH = 97||
Print the edges after sorted
TJ-->BJ = 9||XA-->LZ = 84||CD-->LZ = 85||KM-->CD = 86||CD-->XA = 87||SY-->BJ = 88||TJ-->SY = 89||BJ-->XA = 90||FZ-->GZ = 91||FZ-->SH = 92||SH-->TJ = 93||WH-->XA = 94||FZ-->WH = 95||SH-->WH = 96||TJ-->WH = 97||GZ-->WH = 98||WH-->CD = 99||WH
GZ-->KM = 101||GZ-->CD = 102||CD-->LS = 110||LS-->LZ = 111||LS-->WL = 120||LZ-->WL = 130||
```

Figure 4: process of mergesort

3 Task3: MST

3.1 Two algorithms

We implement algorithm Prim and algorithm Krustal and get the same MST. The details of the implementation of these two methods are notated in the code. The pseudo code of these two alorithms are as below: the step to check if the two ends of the edge are in the same MST are implemented by

Algorithm 1 MST-Prim

```
Initialization:Heap(heap of edges) of size 0, MST(vector of edges in MST) empty, n the number of
vertexs in the graph
for $i = 1$ to n (n step to build the tree) do
    for vertex in the non-traversed neighbors of the current vertex do
        Push the vertex to Heap
    end for
    pop the root of the heap and put it in MST
    set the destination of the edge as current node
end for
```

Union-Find.

Algorithm 2 MST-Krustal

```
Initialization:Heap(heap of edges) that contain all the edges in the graph, MST(vector of edges in
MST) empty, n the number of vertexs in the graph
for $i = 1$ to n (n step to build the tree) do
    pop the minimal edge of the heap and take it as the current edge
    while the two ends of the current edge are in the same MST do
        pop the minimal edge of the heap and take it as the current edge
    end while
    put the current edge to MST
end for
```

3.2 Result Demo

1. There are 12 routes (i.e. edges) in the MST of the map graph.

2. The minimal total distance of routes that keep all cites connected is **1042**.

3. The MST is demonstrated in Figure 5.

4. Regarding WH as root, the tree is a binary tree which is demonstrated in Figure 5.

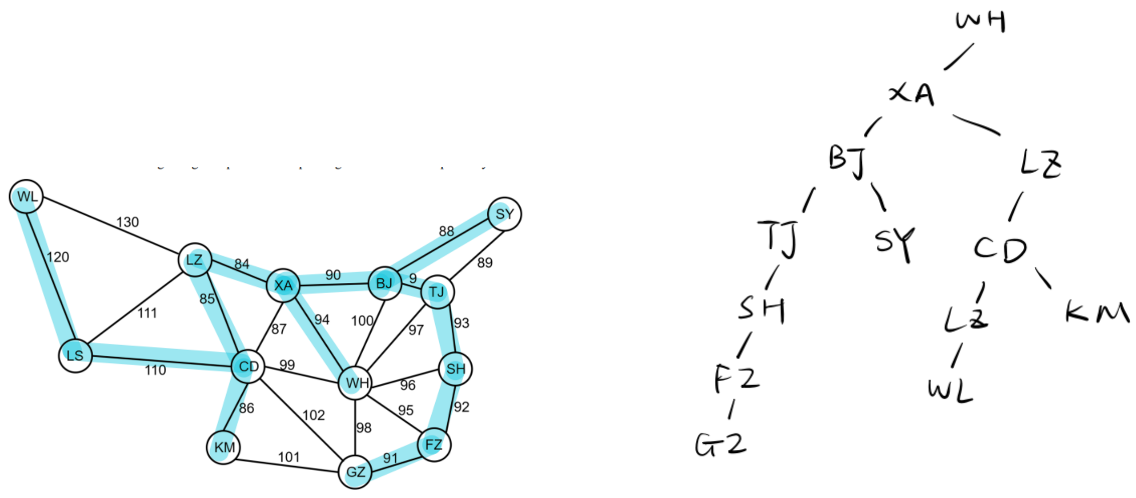


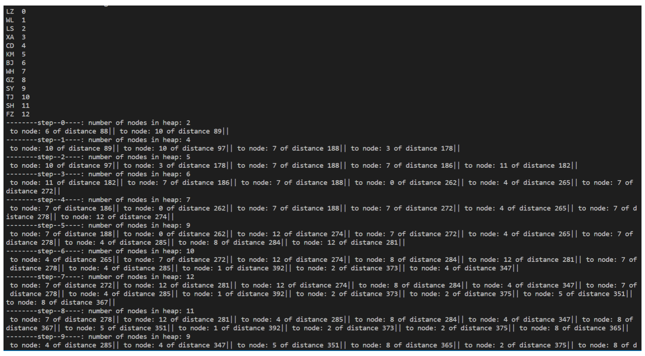
Figure 5: MST

5. The preorder traversal: WH-XA-BJ-TJ-SH-FZ-GZ-SY-LZ-CD-LZ-WL-KM.

6. The postorder traversal: GZ-FZ-SH-TJ-SY-BJ-WL-LZ-KM-CD-LZ-XA-WH.

7. The inorder traversal: GZ-FZ-SH-TJ-BJ-SY-XA-WL-LZ-CD-KM-LZ-WH.

4 Task4: Minimal Distance Path



```
LZ  0
WL  1
LS  2
XA  3
CD  4
KM  5
BJ  6
WH  7
GZ  8
SY  9
TJ  10
SH  11
FZ  12
--------step--0----: number of nodes in heap: 2
 to node: 6 of distance 88|| to node: 10 of distance 89||
--------step--1----: number of nodes in heap: 4
 to node: 10 of distance 89|| to node: 10 of distance 97|| to node: 7 of distance 188|| to node: 3 of distance 178||
--------step--2----: number of nodes in heap: 5
 to node: 10 of distance 97|| to node: 3 of distance 178|| to node: 7 of distance 188|| to node: 7 of distance 186|| to node: 11 of distance 182||
--------step--3----: number of nodes in heap: 6
 to node: 11 of distance 182|| to node: 7 of distance 186|| to node: 7 of distance 188|| to node: 0 of distance 262|| to node: 4 of distance 265|| to node: 7 of
distance 272||
--------step--4----: number of nodes in heap: 7
 to node: 7 of distance 186|| to node: 0 of distance 262|| to node: 7 of distance 188|| to node: 7 of distance 272|| to node: 4 of distance 265|| to node: 7 of d
istance 278|| to node: 12 of distance 274||
--------step--5----: number of nodes in heap: 9
 to node: 7 of distance 188|| to node: 0 of distance 262|| to node: 12 of distance 274|| to node: 7 of distance 272|| to node: 4 of distance 265|| to node: 7 of
distance 278|| to node: 4 of distance 285|| to node: 8 of distance 284|| to node: 12 of distance 281||
--------step--6----: number of nodes in heap: 10
 to node: 4 of distance 265|| to node: 7 of distance 272|| to node: 12 of distance 274|| to node: 8 of distance 284|| to node: 12 of distance 281|| to node: 7 of
 distance 278|| to node: 4 of distance 285|| to node: 1 of distance 392|| to node: 2 of distance 373|| to node: 4 of distance 347||
--------step--7----: number of nodes in heap: 12
 to node: 7 of distance 272|| to node: 12 of distance 281|| to node: 12 of distance 274|| to node: 8 of distance 284|| to node: 4 of distance 347|| to node: 7 of
 distance 278|| to node: 4 of distance 285|| to node: 1 of distance 392|| to node: 2 of distance 373|| to node: 2 of distance 375|| to node: 5 of distance 351||
to node: 8 of distance 367||
--------step--8----: number of nodes in heap: 11
 to node: 7 of distance 278|| to node: 12 of distance 281|| to node: 4 of distance 285|| to node: 8 of distance 284|| to node: 4 of distance 347|| to node: 8 of
distance 367|| to node: 5 of distance 351|| to node: 1 of distance 392|| to node: 2 of distance 373|| to node: 2 of distance 375|| to node: 8 of distance 365||
--------step--9----: number of nodes in heap: 9
 to node: 4 of distance 285|| to node: 4 of distance 347|| to node: 5 of distance 351|| to node: 8 of distance 365|| to node: 2 of distance 375|| to node: 8 of d
```

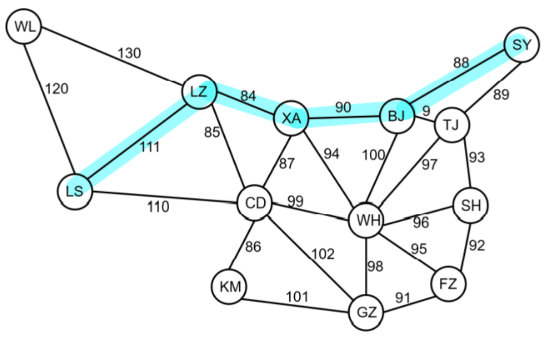


Figure 6: mininal cost path

In this part, we design a class **Node**, which have member variables: **node**, **distance**(the current minimal distance from the starting point to this node.) and **chemin**(the edge lead to this node). We change the pseudo algorithm a little and write it as below: The minimal distance from the city **SY** to city **LS** is **373**. The result is demonstrated in Figure 6

Algorithm 3 Minimal Cost Path

```
Initialization:NH(heap of nodes) of size 1 with the start point in it, end point $p_n$, Path(vector of edges) empty.
while $p_n$ is not visited do
  pop the minimal vertex which have not been visited from NH.
  add the chemin of this vertex to Path.
  for vertex in the non-traversed neighbors of the popped vertex do
    Push the vertex to NH(with its current distance to S).
  end for
end while
```

5 Summary

In this project, we design two main classes **Graphmatrix** and **Graphlist** and study some problems in graph theory: sort the edges by heapsort and mergesort, find the MST(minimal spending tree of the graph), find the minimal cost path to pass the given start point the end point.