**Programming Assignment 1**

**Name: Rui Zuo**

**x500: zuoxx067**

**ID: 5323641**

**Class Number: CSCI 4041**

**Professor Name: Nikolaos Papanikolopoulos**

1. Location and Name of Files

Source codes of the first problem is separated into two parts namead as partA1.py and partA2.py. The other two problems’ source code are partB.py and partC.py.And the locations of them and corresponding txt files is in a file folder named as Rui\_pa2

2）MST

2.1 Program Implementation Description

(1)Prim Algo

My program is encapsulated in a class named Prim\_execution . A function named read\_file will be called when input file is loaded into the program. This function will produce a t-dimension list which records a matrix-representation graph read from input file. And another class named vertice is defined to record the data of every vertice of the graph including index, edge and key. Under this class’s definition, the function make\_vertices will be called to produce the instances of class vertices corresponding to the input graph. After that, function Prim will be called to build the MST following the Prim-Algo. Every instance of vertice will be used and their attribute key and pi will be determined. (pi means the parent of this vertice in MST) . Finally ,function print\_MST will be printed out under the instruction of assignment which prints out a matrix-representation graph of the MST.

(2)Kruskal Algo

In this program, the function read\_file overally is the same as the Prim Algo. In the process of read\_file function executed, the instance of class edge will be initialized for every edge of the input graph. The corresponding function of union set is built up for the use of Kruskal Algo which could distribute instance of vertice into a set. And function Union could unite two sets by setting the root of a set as the node of another set’s node. Under such preparation, function Kruskal will be called following the Kruskal Algo and finally return a list of instances of edge which built up the MST.

2.2 Known Bugs

None to the best of my k

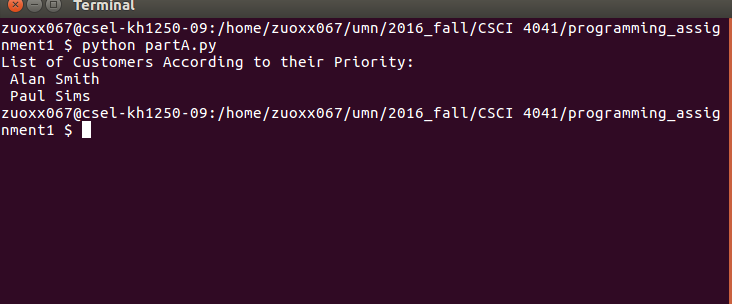
2.3 Style Guidelines

My program follows the general indentation style of python 2.7. Every function within the class have been followed with comments. My variables were named by its definition and usage separated by a underline.

2.4 Innovations

I made use of two classes name as vertice and edge to implement two algo smoothly. Two classes contain the necessary attribute for these two algorithms like key, pi, weight. Under this definition, the implementation of algorithms is very naturally and easy.

2.5 Terminal Session and Source Code



3）

3.1 Program Implementation Description

In this program I still make use of the function read\_file to read the input file and produce a global list in the class Floydw\_execution to represent the graph. In the input file, Because essentially the Floydw algorithm is a dynamic programming problem. In every step of this algorithm the comparison of the shortest path will be implemented between last step and this step. So the weight of edge will be initialized as 10000 to represent INF in the input file. The implementation of the Algo is clean and short following the description of algo in the textbooks.

3.2 Known Bugs

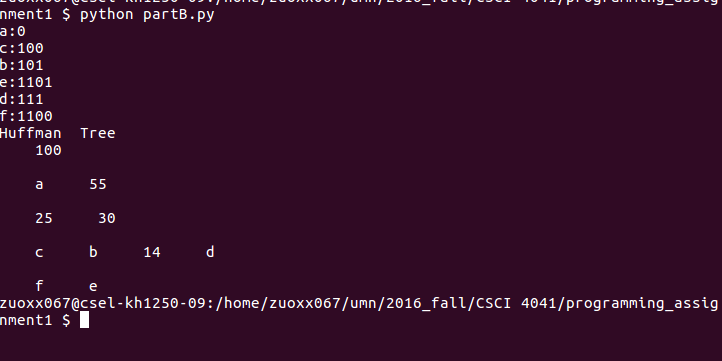
3.3 Style Guidelines

My program follows the general indentation style of python 2.7. Every function within the class have been followed with comments. My variables were named by its definition and usage separated by a underline.

3.4 Innovations

I make use of two two-dimensional list to implement the algo. One list record the weight of shortest path and the other one is to record the parent vertice to finally print out every step of shortest path. The use of memory will be saved a lot under this design.

3.5 Terminal Session Script and Source Code



4)Biconnected Component

4.1 Program Implementation Description

In this program I make use of a list which contains two elements to represent the edge rather than meke use of class edge. In the implementation of Algorithm , Seven global lists is initialized firstly including stack(), parent(), d(), low() , bridge() and AP() to record the corresponding data of the algorithm, And then the in the loop of every vertice in the graph, DFS function will be called to explore the structure of the graph and determine the bridge, AP and biconnected-component. In the DFS function once a new tree-edge is found it will be pushed into the stack. And following the algorithm posted in the moodle, once the requirement that low[v]>=d[u], the component is found and will be printed out. And then in the bottom of DFS function the AP will be determined and add into the AP list. In the function OutputComp, if the number of edge of this component is only one. It is a bridge and will be added into bridge list. Finally the bridge list and AP list will also be printed out following the instruction.

4.2 Known Bugs

For the convenience, the string list in my program which contains patter and matching string name W is actually started from index1 which means the 2nd position of the list. It is the same for the next table. All the position 0 of these lists are character ‘!’

4.3 Style Guidelines

My program follows the general indentation style of python 2.7. Every function within the class have been followed with comments. My variables were named by its definition and usage separated by a underline.

4.4 Innovations

I don’t think there exists some innovative points because I just design it following the general kmp algorithm

4.5 Terminal Session Script and Source Code

