Homework 3
CEng303- Design and Analysis of Algorithms
(Due Date: 17 December 2017, 23:59)
(100 points)

According to **your student id** and **the first letter of your name**, your implementation differs; so, please **read** the homework **carefully**.

Introduction

In this homework, you are going to implement a binary search tree whose nodes are graphs each of which is composed of at most 10 cities with corresponding highway distances in kilometers (0 km means no direct highway between the two cities). The city names will not contain blank characters. The data structure of a sample node is given below. Note that the following classes are samples and you are free to define and extend your own classes.

```
Class Node{
    …
    int zip_key;
    Graph cities;
    Node * left;
    Node * right;
    …
    }
Class Graph{
    …
    String city_names[10];
    int distances [10][10];
    …
    }
```

Your program will read an input.txt file in which the first line will represent the first two digits of the zip codes belonging to the following 10 cities (**at most**). Then, an adjacency matrix (not necessarily symmetric) will be given in the input file. There is no limit in the number of nodes for the BST. The following text is a sample input:

90
San Francisco
LosAngeles
SantaBarbara
SanDiego
Encino
San Jose
Sacramento
Alameda
Anaheim
Covina

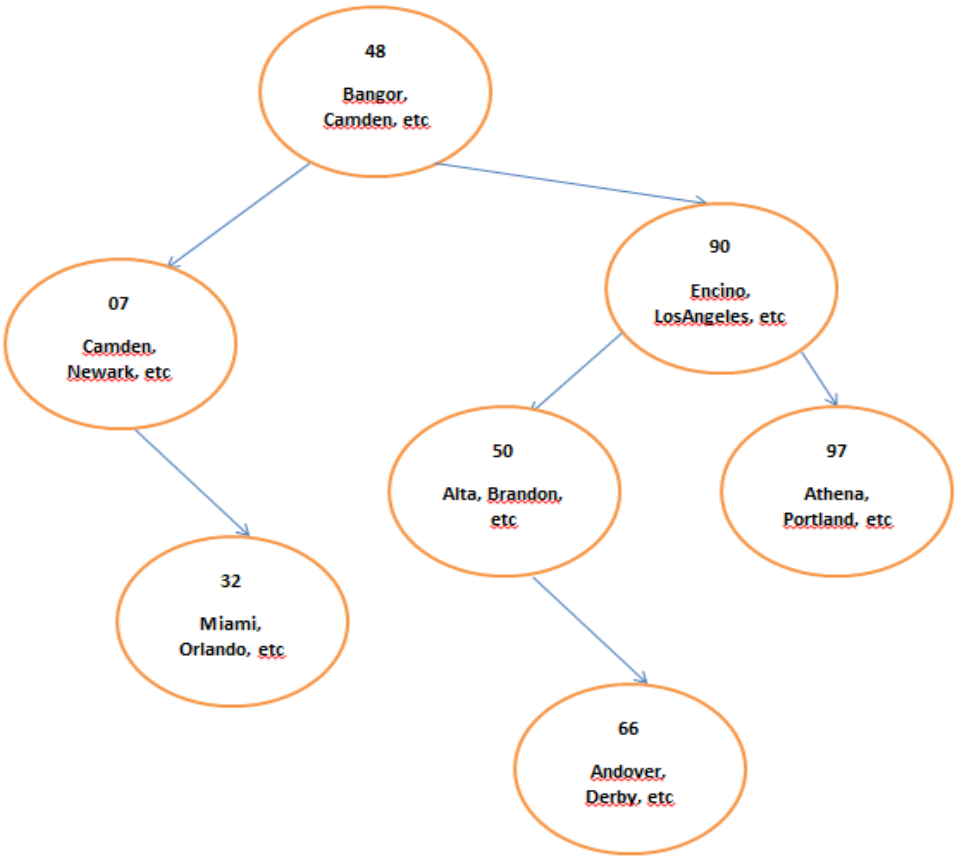
0	450	400	0	470	100	0	520	0	165
450	0	0	133	125	99	0	15	79	0
400	0	0	246	0	0	320	0	65	0
0	133	246	0	460	78	112	0	0	63
470	125	0	460	0	89	490	45	12	440
100	99	0	78	89	0	780	90	0	0
0	0	320	112	490	780	0	89	58	0
520	15	0	0	45	90	89	0	90	530
0	79	65	0	12	0	58	90	0	70
165	0	0	63	440	0	0	530	70	0

7
Newark
Trenton
Edison
Paramus
Camden

0	120	58	0	12
0	0	47	0	97
0	0	0	24	0
0	0	24	0	46
0	12	25	55	0

…

You must create your BST according to the order in **input.txt**. When you create your BST from the input.txt in which each node is a graph of cities, the BST must look like this:



Implementation Steps

- Read the **input.txt** file. Construct a BST by using the two digit zip codes as keys. Your nodes will be graphs of corresponding cities. Then wait for a user input from the command-line. If you need a source node for any algorithm, pick up the city which lexicographically precedes the others. If you need to pick a city among the set of cities adjacent to a visited city or enqueue /push the other cities, again use the lexicographical ordering as we did for BFS and DFS in the class.

1. If the last digit of your student ID is an **odd number** and the first letter of your name is a **vowel,** you will implement **Dijkstra** (D), **Depth-First Search** (DI) and **Prim's** (P) algorithms. Here are the user commands your program must support:
    a. User Input: D 90 SanFransico Sacramento
    Output*: Find your node whose key value is 90. Print out the shortest path and the total length of the distance between San Francisco and Sacramento by Dijsktra's algorithm.*
    b. User Input: DI
    Output: Print out the inorder traversal of the BST tree. Whenever you visit a node (i.e., graph of cities), print out the cities by using Depth-First Search.
    c. User Input: P 7
    Output: Find the node whose key value is 7. Implement Prim's algorithm to print the minimum spanning tree of the cities in that node.

2. If the last digit of your student ID is an **odd number** and the first letter of your name is a **consonant,** you will implement **Dijkstra** (D), **Breadth-First Search** (BPR) and **Kruskal's** (K) algorithms. Here are the user commands your program must support:
    a. User Input: D 90 SanFransico Sacramento
    Output*: Find your node whose key value is 90. Print out the shortest path and the total length of the distance between San Francisco and Sacramento by Dijsktra's algorithm.*
    b. User Input: BPR
    Output: Print out the preorder traversal of the BST tree. Whenever you visit a node (i.e., graph of cities), print out the cities in the node by using Breadth-First Search
    c. User Input: K 7
    Output: Find the node whose key value is 7. Implement Kruskal's algorithm to print the minimum spanning tree of the cities in this node.
    d. User Input: Q
    Output: Bye! (Program terminates)

3. If the last digit your student id is an **even number** and the first letter of your name is a **vowel,** you will implement **Dijkstra** (D), **Depth-First Search** (DPO) and **Topological Sorting** (T) algorithms. Here are the user commands your program must support:
    a. User Input: D 90 SanFransico Sacramento
    Output*: Find your node whose key value is 90. Print out the shortest path and the average of the distance (i.e. total distance / number of edges) between San Francisco and Sacramento by Dijsktra's algorithm.*
    b. User Input: DPO
    Output: Print out the postorder traversal of the BST tree. Whenever you visit a node (i.e., graph of cities), print out the cities in the node by using Depth-First Search
    c. User Input: T 7
    Output: Find the node whose key value is 7. Implement topological sorting algorithm to that node.
    d. User Input: Q
    Output: Bye! (Program terminates)

4. If the last digit your student id is an **even number** and the first letter of your name is a **consonant,** you will implement **Dijkstra** (D), **SCC** (DI) and **Prim's** (P) algorithms. Here are the user commands your program must support:
    a. User Input: D 90 SanFransico Sacramento
    Output*: Find your node whose key value is 90. Print out the shortest path and the average of the distance (i.e. total distance / number of edges) between San Francisco and Sacramento by Dijsktra's algorithm.*
    b. User Input: SCC 7

Output: Find the node whose key value is 7. Implement Kosaraju's algorithm to find the strongly connected components in that node. Print dashed between two strongly connected component. Print comma otherwise.

c. User Input: K 7
   Output: Find the node whose key value is 7. Implement Kruskal's algorithm to print the minimum spanning tree of the cities in that node.
d. User Input: Q
   Output: Bye! (Program terminates)

Sample Runs

DIJKSTRA (Some students will print total distance. Some will print avg distance)
User Input: D 90 Encino SanJose
Output: Encino-SanBernardinio-SanJose
Total Distance: 36 Km
Average: 15 km per stop

PRIM
User Input: P 97
Output: Athena-Bend, Athena-Canby, Carlton-Tigard, Canby-Durham, Canby-Portland, Portland-Salem, Durham-Tigard

KRUSKAL
User Input: K 66
Output: Newton-Salina, Newton-Wichita, Derby-Salina, Parsons-Wichita, Andover-Lenexa, Lansing-Salina, Derby-Lenexa

TOPOLOGICAL SORTING:
User Input: T 32
Output: Miami, Largo, Bell, Naples, Deltona, Aventura, Esto, Orlando

SRONGLY CONNECTED COMPONENTS
User Input: SCC 48
Output: Bangor-Detroit-Gibraltar, Grant, Ithaca

Deliverables

- A ReadMe.txt: Your student ID, name and last name in the first line. Put "Even" in the second line if your student ID is even. Put "Odd" otherwise. Put "Vowel" in the third line if your name starts with a vowel. Put "Consonant" otherwise. Give a brief definition (1-2 sentences) for every function you implemented.
- Your **own** input.txt file if you have one.
- Your source code.
- Follow the instructions given in the homework submission rules on Uzem. Otherwise, we may not read your homework.

BONUS 1 (10 pts)

- User Input: Tarjan 7
- Output: Find the node whose key value is 7. Implement Tarjan's algorithm to print the strongly connected components (https://activities.tjhsst.edu/sct/lectures/1516/SCT_Tarjans_Algorithm.pdf).

BONUS 2 (10 pts)

- User Input: KNUTH 07
- Output: Find the node whose key value is 07. Find ALL possible topological sorts of the cities in that node (http://www.cs.ncl.ac.uk/publications/trs/papers/61.pdf).

BONUS 3 (5 pts)

- If you implement your own queue and stack and use them in a working function, you will get 5 points bonus.

Notes

- You are not allowed to use build-in libraries for trees and graphs. You have to implement your own. You can use stack and queue libraries. You do not have to implement them but you are welcome to do so if you want to practice your data structures knowledge. You will get 5 points bonus if you implement your own stack and queue but they **must** be used in a function producing some output successfully.
- You are free to design your own data structures and you can convert the adjacency matrices to linked-lists if you want to.
- Note that the numbers and cities in this homework are random and do not represent the real life.
- Use lexicographical ordering of the cities to decide on the source city if you need to.
- Push/Enqueue cities preserving lexicographical ordering if you need to.
- Bonus parts were not explained in the class. You have to research, learn and implement by yourselves.

- This is an individual homework. You are welcome to get together and discuss your ideas. However, you are not allowed to share your source codes, which is cheating. In case of cheating, both the source(s) and the receiver(s) will get zero from all homework and the disciplinary actions will be taken.
- If you have any questions, send an email to the instructor or the teaching assistant.
- **You have to follow the instructions carefully. This homework differs with respect to your student id and the first letter of your name.**