Design – P4

Kehan Xing
UW id: k5xing Student#:20783902

Classes:

I designed two classes to represent linked list and two structs for nodes in this project.

(I)

Struct Citynode:

The struct Citynode contains four variables including name (string), distance(double), citynext (Citynode*) used for connecting the linked list, visitedcity (bool) used in finding shortest distance.

Class Citylist:

This class contains two variables: degree(int) which represents the degree of a node in the graph, cityhead (Citynode*) which is the head of the linked list. In this linked list, the cityhead represents the specific city, and the remaining of the linked list represents the adjacent cities. There are two functions, a constructor and a destructor in the class.

Class Citylist Member functions:

a. insertcity function: this is a function which can insert a city into the linked list, it requires string parameter which is the name of the city and a double parameter which is the direct distance between cityhead and the city we are inserting, and there's no output (void)

b. searchcity function: this is a function which can check whether there is a city in the linked list has the input name. It takes a string variable and returns a CityNode* variable which represents the node with the input name; if there's no such node, it will return NULL.

Class Citylist Constructor and Destructor:

There's one constructor and one destructor. The constructor is a default constructor which takes no input, and it set the variable degree to 0 and the variable cityhead to NULL. The destructor takes no input as well, and it destructs the tree by setting degree to -1 and cityhead to NULL.

(II)

Struct Graphnode:

The struct Graphnode contains six variables including citylist (Citylist), shortd(double), graphnext (Graphnode*) used for connecting the linked list, visitedcity (bool) used for finding shortest distance, parent and child (Graphnode*) used for printing the path with shortest distance.

Class Graph:

This class contains three variables: node_num (int), edge_num (int), graphhead (Graphnode*) which is the head of the linked list. In this linked list, every node contains a citylist. There are ten functions, a constructor and a destructor in the class.

Class Graph Member functions:

a. searchgraph function: this function can find a specific city in the linked list has the input name. It requires a string parameter that represents the name. The return type of this function is Graphnode* which represents the node with the input name; if there's no such node, it will return NULL.

b. insertgraph function: this function takes a string variable as the name of a city, and it will insert the city into the linked list if it is not already existed. The return type of this function is bool which means whether the insertion is successful.

c. setedge function: this function takes two string variables as the name of two cities and a double variable which is the distance of the edge. The return type of this function is bool which represents whether the distance has been set successfully.

d. degree function: this function takes a string variable as the name of a city, and it will return an int variable as the degree of the node with the input name.

e. adjdistance function: this function can return the adjacent distance between two nodes. It requires two string parameters which are the names of the cities, and the return type is double for distance. If either city is not found or the cities are not connected directly, it will return -1.

f. clear function: this function will delete the linked list and set node_num, edge_num to 0 and graphhead to NULL.

g. min function: this function is used in shortdistance function to find the extractMin. It takes no input and returns the name of the city with minimum variable shortd.

h. relax function: this function is used in shortdistance function to relax edges. It takes two Graphnode* parameters which represent two cities in the graph. There's no output (void).

i. shortdistance function: this function will find the shortest distance between two cities. It takes two string variables as the names of two cities, and it will return the shortest distance as a double variable.

j. print function: this function will print the path with shortest distance from one city to another. It takes two string variables as the names of two cities, and it will call the shortdistance function to find the path. There's no output for this function (void).

Class Graph Constructor and Destructor:

There's one constructor and one destructor. The constructor is a default constructor which takes no input, and it set the variables node_num and edge_num to 0 and the variable graphhead to NULL. The destructor takes no input as well, and it destruct the graph by setting node_num and edge_num to to -1 and graphhead to NULL.

Analysis for the time complexity

The ideal running time for Dijkstra's algorithm without using heap is V^2. I did not achieve this time complexity in my project. The formula is Time = |V| T(extractMin) + |E| T(modifyKey).
Because I used unsorted linked list, I need to search through the linked list. Therefore, the time to modify key becomes |V|. The overall complexity T=O (|V|*|V| +|V|*|E|) =O(|V||E|)

The time complexities for all the commands are:

a. i (insert): same as search => T = Θ (|V|)
b. setd: same as search => T = Θ (|V|)
c. s (search): need to go through the graph linked list => T = Θ (|V|)
d. degree: same as search => T = Θ (|V|)
e. graph_nodes: T = Θ (1)
f: graph_edges: T = Θ (1)
g: d(distance): same as search => T = Θ (|V|)
h: shortest_d: T= |V|*(|V|+degree1+degree2+…+degree V)
=|V|*|V| +|V|*|E|
= Θ (|V||E|)
i: print_path: same as shortest_d => T = Θ (|V||E|)
j: clear: need to go through the graph linked list => T = Θ (|V|)

Reflection:
It would be better to use some algorithm with priority, such as binary heap. If using ordinary linked list, I should avoid using search function since the running time is |V| which is relatively big.