ECE250-Project 2

Shihang Wang
S652wang
Apr 13rd,2021

1. **Overview of Classes**

   class adjacency_matrix:
   This is a class to create the graph by the adjacency matrix. An int type n represents row and column, and another int type edge_count represents the number of edges. Two vectors are used to initialize the matrix and the degree of the graph. The size of the matrix is n*n. The value of an entry matrix[i][j] is either 0 or 1. It depends on whether the edge is existed between the vertex i and vertex j.
   class MST:
   This is a class to calculate the mst. It returns the minimum weight edge from the edges for two sets of vertices.
   class illegal_exception:
   This is a class to handle invalid input using exception handling.
   class operator:
   This is a class handle all the commond which including n, I, e, adjacent, degree, edge_count, clear, mst, and exit.
   A change to the Operator class needs a change to the illegal_exception calss, adjacenecy_matrix class, and MST class. A change to the adjacency_matrix needs a change to the MST. Edge is a struct inside private MST class.

2. **UML class diagram (see Last Page)**

3. **Details on design decision**

   class adjacency_matrix:
   The constructor of this class initializes the adjacency matrix, degree, and edge count. The destructor uses default. addEdge() and deleteEdge()add and delete edges. And getEdge return the matrix.
   minimumEdge() updates the minimum edge connect with vertex i. Geters for Row, Column, EdgeCount, Degree simply return these values. And clear() is used to clear the graph, which clear the matrix and set degree and edge count to 0. printMatrix() is a debug function.

   class MST:
   Default Constructor and destructor all called in this class. MST() is to construct and calculate MST for a graph by using adjacency matrix representation. setU and setT are vectors. (it is a very bad implementation for the run time, it will be explained in the later section) sn represents start node. Use push back to update the setT. Another vector it is used to update the vertex in the next two for loop. For the size is less than the setU and setT, update the u, v, and w. “not connected” is returned when u==-1 or v==-1.

Otherwise, accumulating the weight and erase the setT. Finally, mst return the weight after the operation.
printEdge() is also a debug function. And getWeight() function return the weight.

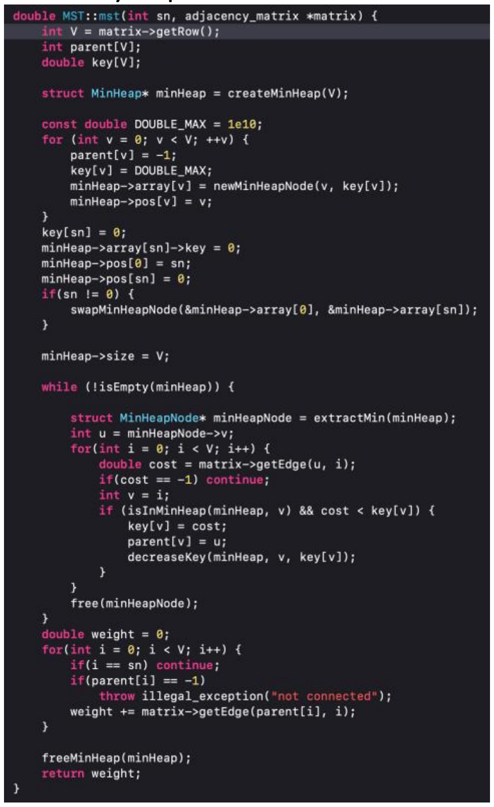
class illegal_exception:
Exception Handling to handle invalid characters in the word.

class Operator:
The constructor of this class initializes the matrix, mst, and node. The destructor of this class deletes the matrix and mst, then set them to nullptr. Comd function for ‘n’, ‘i’, ‘e’, ‘adjacent’, and ‘clear’ are all bool type. They manage the illegal input and call the functions in class matrix to do the command. And command function for ‘degree’, ‘edge_count’ are int type, they return the value by call the functions in class matrix. So as comd ‘mst’, however it returns a type of double. run() runs all the operations mentioned above and print the desired result.

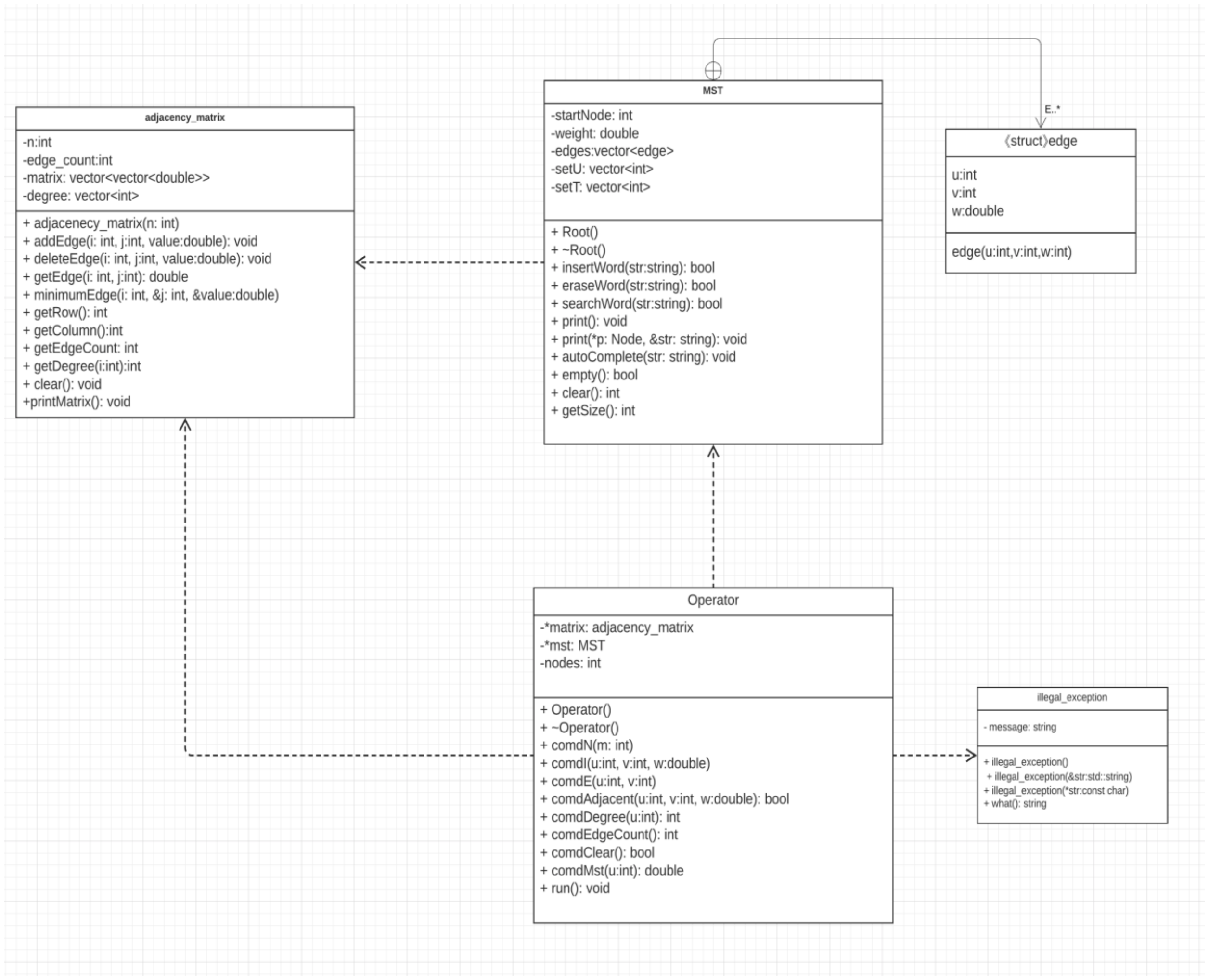
4. **Test cases**
Basic cases:
Check the decimal weight and test each command.
Test the illegal_exception cases. (10;10;3.4 should be illegal argument) (Outside the valid range cases should be illegal argument)
Test “not connected” “no adjacent” cases. (10 nodes, 5 nodes connected, print “not connected”)
Use min spanning tree graph generator to test returned mst value.
Corner cases:
The large decimal number should be considered.
Infinity should be implemented in order to get the max value of a type.
The automated random test cases are used. In addition, memory should be considered when clean the graph. As well as Running time will also be considered to achieve the minimum running time.

5. **Performance considerations**
class adjacency_matrix:
Constructor of this class takes O(V^2) to initialize the matrix.
addEdge() ,deleteEdge(), and getEdge() take constant time O(1).
minimunEdge() takes O(n) time to traverse by for loop and n is the number of row and colume for the adjacenecy matrix.
clear() also takes O(n) time, and n is the number of row and colume for the adjacenecy matrix.
printMatrix() takes O(n^2) times to print the matrix.
The other getter functions all take constant time O(1) to return the desired value.
The space adjacency Matrix used are O(v^2).

class Operator:
It takes constant time to do commonds.
class MST:
It is very bad implementation for MST. The beginning size of setU is 1, and setT is V. Let's assume the V is 4. After first time traversal, setU becomes 2, and setT becomes 3. Based on that, the total times needed is 1*4+2*3+3*2+4*1 = 20. The beginning size of set is 5-1=4. The number of nodes is 5. It seems less than 5^2. It should be O((n-1)n!). However, if the n is very large. It is even worse that O(n^2). It really depends on the size of setU and setT, since the two for loop needs to traverse based on these two vectors. In addition, the while loop which check the empty of setT will also make the time complexity larger. It is a really bad idea to use vector.
I have also tried to implement the min heap method for mst. As you can see in the picture. This one takesO(ElogV). extractMin() is using min heap, it take O(logV). And running v times will take Vlog(V). And the time complexity for the decreaseKey() is O(logV), it will running E times to find minimum value. The total running time will be O(ElogV) as expected.
However, my min heap works badly. And I do not have enough time to debug. I will definitely implement this after this term.



```
double MST::mst(int sn, adjacency_matrix *matrix) {
    int V = matrix->getRow();
    int parent[V];
    double key[V];

    struct MinHeap* minHeap = createMinHeap(V);

    const double DOUBLE_MAX = 1e10;
    for (int v = 0; v < V; ++v) {
        parent[v] = -1;
        key[v] = DOUBLE_MAX;
        minHeap->array[v] = newMinHeapNode(v, key[v]);
        minHeap->pos[v] = v;
    }
    key[sn] = 0;
    minHeap->array[sn]->key = 0;
    minHeap->pos[0] = sn;
    minHeap->pos[sn] = 0;
    if(sn != 0) {
        swapMinHeapNode(&minHeap->array[0], &minHeap->array[sn]);
    }

    minHeap->size = V;

    while (!isEmpty(minHeap)) {

        struct MinHeapNode* minHeapNode = extractMin(minHeap);
        int u = minHeapNode->v;
        for(int i = 0; i < V; i++) {
            double cost = matrix->getEdge(u, i);
            if(cost == -1) continue;
            int v = i;
            if (isInMinHeap(minHeap, v) && cost < key[v]) {
                key[v] = cost;
                parent[v] = u;
                decreaseKey(minHeap, v, key[v]);
            }
        }
        free(minHeapNode);
    }
    double weight = 0;
    for(int i = 0; i < V; i++) {
        if(i == sn) continue;
        if(parent[i] == -1)
            throw illegal_exception("not connected");
        weight += matrix->getEdge(parent[i], i);
    }

    freeMinHeap(minHeap);
    return weight;
}
```


adjacency_matrix
-n:int
-edge_count:int
-matrix: vector<vector<double>>
-degree: vector<int>
+ adjacenecy_matrix(n: int)
+ addEdge(i: int, j:int, value:double): void
+ deleteEdge(i: int, j:int, value:double): void
+ getEdge(i: int, j:int): double
+ minimumEdge(i: int, &j: int, &value:double)
+ getRow(): int
+ getColumn():int
+ getEdgeCount: int
+ getDegree(i:int):int
+ clear(): void
+printMatrix(): void
MST
-startNode: int
-weight: double
-edges:vector<edge>
-setU: vector<int>
-setT: vector<int>
+ Root()
+ ~Root()
+ insertWord(str:string): bool
+ eraseWord(str:string): bool
+ searchWord(str:string): bool
+ print(): void
+ print(*p: Node, &str: string): void
+ autoComplete(str: string): void
+ empty(): bool
+ clear(): int
+ getSize(): int
E..*
《struct》edge
u:int
v:int
w:double
edge(u:int,v:int,w:int)
Operator
-*matrix: adjacency_matrix
-*mst: MST
-nodes: int
+ Operator()
+ ~Operator()
+ comdN(m: int)
+ comdI(u:int, v:int, w:double)
+ comdE(u:int, v:int)
+ comdAdjacent(u:int, v:int, w:double): bool
+ comdDegree(u:int): int
+ comdEdgeCount(): int
+ comdClear(): bool
+ comdMst(u:int): double
+ run(): void
illegal_exception
- message: string
+ illegal_exception()
+ illegal_exception(&str:std::string)
+ illegal_exception(*str:const char)
+ what(): string