Project 4 Design report

Pan Liu 20830383

Overview of the Classes

In the project 4, we are implement a graph data structure to stored a weighted, directive graph. In the project, we store a research ID a collaborate with b and contain the “influence weight” 0 to 1. Thus, we will have 4 classes of this implementation which is node, edge, graph, and illegal_argument.

In the node class, we have private variable of vector neighbour which represent all neighbour of the vertex. Then edge_index vector stored the index of adjacency edge of that vertex. We have a boolean value in_the_graph to see if the vertex is insert to the graph or not. The variable mst_k is for see the collaborate weight between the node when we doing the maximum spanning tree. Besides, boolean value is_visit to check if the node is visited in mst, it will reset to false when the mst is over.

In the edge class, it will store a edge of the graph, which contains a start point, a end point (destination), and weight. If the node need to be deleted, we set the boolean value i_graph to false.

In the illegal_exception class, we do nothing in the class, but, in the main function if we find any illegal argument, we will catch and throw that illegal argument.

In the graph class, it contains a dynamically allocated fixed array of size 23133 represent it will contains maximum 23133 vertices. Vector g_edge store ever edge in the graph. Vector mst store the maximum spanning tree nodes and heap store the node when doing the prim algorithm. Besides, we have following functions.

bool find(int a, int b, double w): find if the edge is exist in the graph, if true, we cannot insert, if false, we can insert.

int num_of_v(): return the value nov which represent the number of vertices.

void insert(int a, int b, int w): insert the edge in the g_edge vector, then insert the node/vertex into the array.

void delete(int a): go to the loop of vector of neighbour, and we have correspond edge_index of that edge either point to the vertex or the adjacency. Then we set the i_graph of those edge to false, call the clear function to clear the vector of that node. Then, set the boolean value in_the_graph to false represent it is not in the graph now.

void print(int a): in the node, we have vector of neighbour, doing the loop when i is smaller than the neighbour size. If the edge is the adjacency of the node, we print it. If the edge is deleted or is not adjacency node, we do not print that.

void build_heap(): in the heap vector, half of the heap has child, so we do a loop from half of heap size down to 0, and we doing the heapify on the current index.

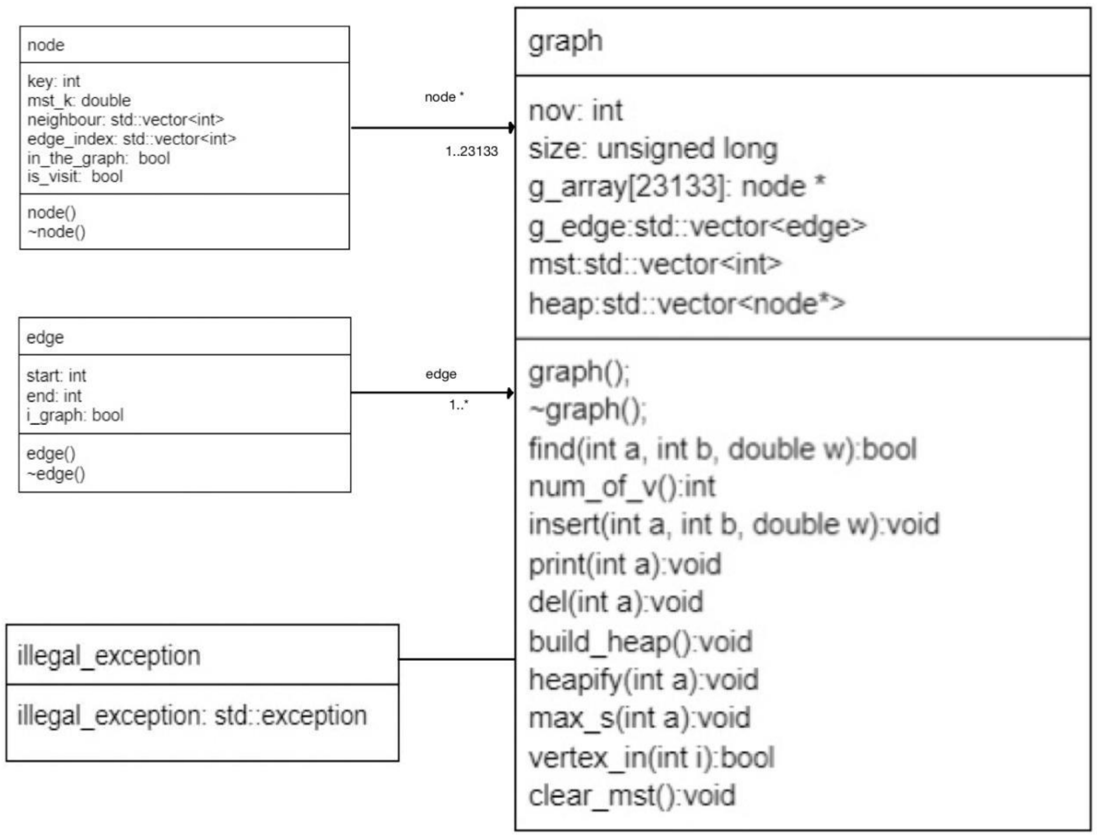
void heapify(int i): the left child is 2*i+1 and right child is 2*i+2, and if the left child is greater than the parent, we set the max to the left child, then if the max

void max_s(int a): we apply the prim algorithm push back the first node to the heap and set mst_k = 0, the other nodes in the graph will push back to the heap and set the mst_k to -1. Then while the heap size is not 0, we call build_heap() function and extract the first(largest mst_k) node of the heap. If the extracted node’s is_visit is false, push_back the key of node into the vector mst then set is_visit to true. Then we loop find the adjacency edge of that node and reset the mst_k(if it is new weight is greater than the current mst_k, then set the mst_k to new weight). After those, we build_heap() to seek next largest mst_k node.

bool vertex_in(int i): if the boolean value in_the_graph is true, then return true. Otherwise, return false

void clear_mst(): after doing mst, we have a mst vector, for each element in the mst, the boolean value of is_visit is true, we need to reset those to false after we doing the max_s(int a) function.

The UML



Design decisions

It is a fixed array which contains 23133 element we dynamically allocate the array, when we modify the value in the node, it will change in the heap.

Node Class

Constructor: initial all values to default. Node of key = 0, mst_k = -1 and both in_the_graph and is_visit are false.

Destructor: clear all the vector

Edge class

Constructor: initial the value to default(start = 0 end = 0 i_graph = false).

Destructor: Do nothing

Graph class

Constructor: new every node in the array and set the size and nov to zero.

Destructor: Deallocate all the node, delete and set to null pointer.

Test Case

Test1: test print, insert. Insert several edges then see if it is insert correct or is failure.

Test2: test load, print, insert, and delete. Load the text into the program then delete some vertex and reassign the vertex to see the program works fine.

Test3: test load, print, insert, delete and illegal exception to see if throw a correct illegal argument

Performance consideration

Command i: Insert the edges is a constant time. However, we need to search the neighbour of the node to see if already in the graph. The best case is O(1) the node is not in the graph, and worst case is O(n) which n is the number of neighbour of node a.

Command **p**: Best case is when there is no adjacency vertex to print which is O(1) and worst case is print every adjacency vertex which will be O(degree(a)).

Command **d**: For delete a vertex, the best case is the vertex is not in the graph(already deleted), the running time will be O(1). The worst case will be delete every edge associate with the node. The running time will be O(n) which n is the number of neighbour of node.

Command **mst**: By applying the prim algorithm. In this project, we are implement a heap data structure. The best case is when the vertex is not in the graph or the graph is empty which O(1). The worst case of running time will be V* T(extract) + O(E)T(modify key). For binary heap, the extract time and modify key time will be O(lgV). Thus the running time will be O(V lgV + E lgV) = O(E lgV)

Command **size**: A constant time O(1) which print the variable nov.