Design Document

- **Overview of class**
  - *class Node:* The Node class contains a pointer in order to make an adjacent list for the graph, and stores the researcher's number and the weight. The node class is initialized by using the constructor Node(), and destruct using ~Node() to avoid memory leak.

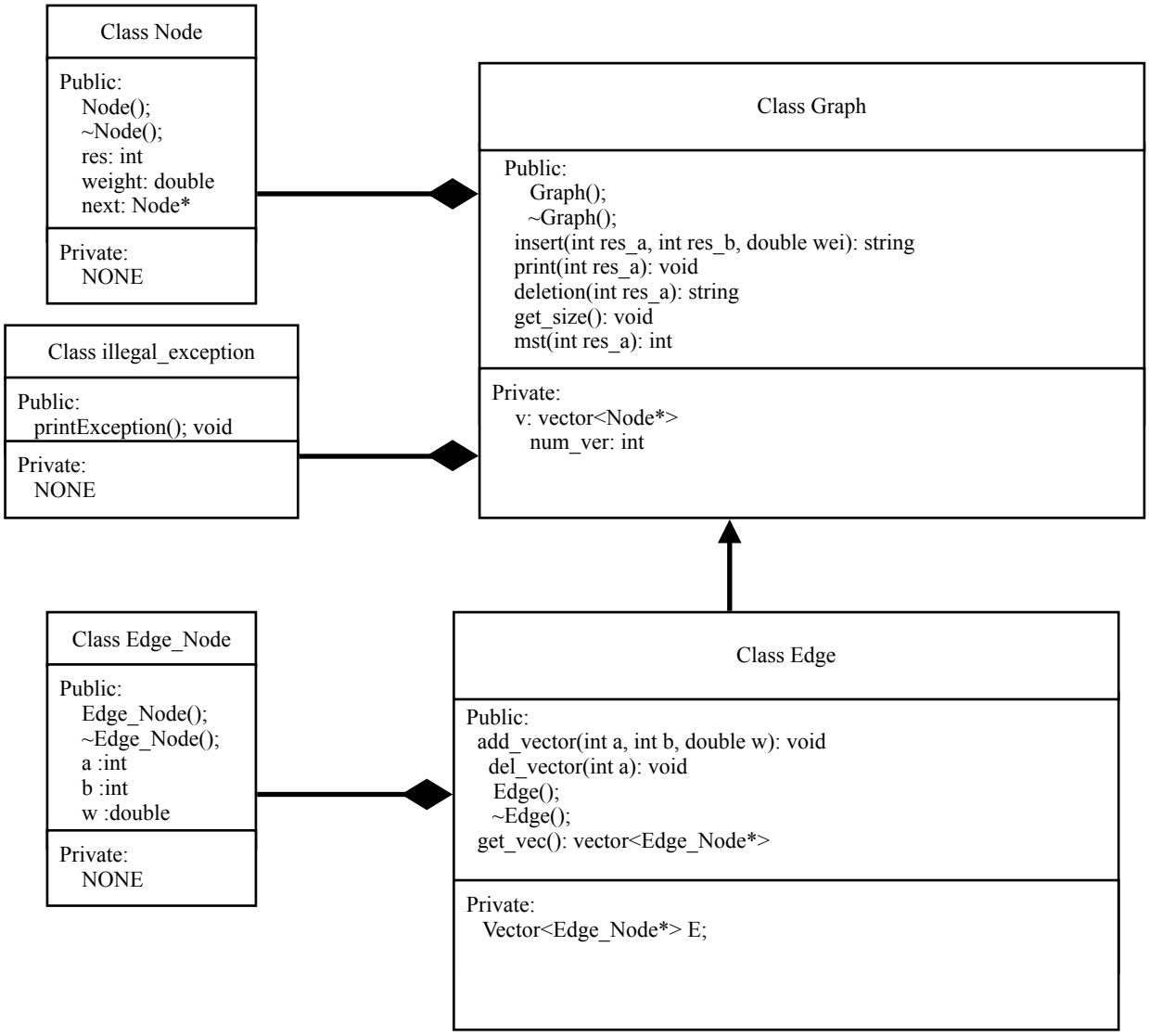
  - *class Graph:* The Graph class contains seven public functions: a constructor, a destructor, insert, print, deletion, get_size, and mst, and two private variable vector v consists of pointers, and the num_ver showing the number of vertex. The constructor initialize the private variable vector to NULL, and num_ver to 0. The destructor free the pointers in the vector v. Function insert inserts researcher numbers and weights within the valid range (research #: [1, 23133]; weight:[0, 1]) using adjacent list. We first check if both researchers' numbers are in the adjacent list, then we insert the ones that are not in the vector v into vector v. Then, we traverse the first researcher's number by using pointer next. Insert the second researcher's number, and the weight into the list if it is not in the list. We increase the value of num_ver each time a vertex is added to the vector. Function print find the input researcher's number using for loop, and then print out the linked list using next pointer. It also throws an error when the input number is not in the valid range. When the vertex is empty, it prints a new line. Function deletion first checks if the input number is in the valid range, and throw an error when the number is out of range. Return failure when the vector is empty. Then, it checks if the number is in the vector v. If not, return failure. If this number is in the vector v, we first delete the pointer, and then erase this value from the vector v. Then, we traverse the whole vector, and each element's linked list to delete the input number if it exists. Decrease the value of num_ver each time a vertex is successfully deleted. Finally, we return "success." Function get_size returns the number of vertex(num_ver). Function mst gives the number of vertices that constructs a maximum spanning tree. We first check if the input num is in the vector, and whether the vector is empty. Return failure for the previous situations. Then, we use three vectors to store the value of the vertices that has already stored in the tree, vertices that has not been stored in the tree, and the weight. Until the vertices inserted into the tree equals to the vertices connected to input node, we print out the number of nodes in MST.

  - class illegal_exception: This class throws an error message: "illegal exception" when the research number is smaller than 1 or larger than 23133, or the weight is not in the range[0,1].

  - class Edge_Node: This class contains three elements, researcher's number a and b, and the weight between them. It stores these information.

  - class Edge: This class add all edges in the graph into the vector while insertion function is called, and delete the edges when user wish to delete the edges, and vertices.

- **UML Diagram**



- **Detail on design decision**
  - class Node:
    - One constructor and one destructor is created.
    - Reason: initialize the pointer, and the values stored in the pointer, and free the pointer to avoid leakage.
  - class Graph:
    - One constructor and one destructor is created.
    - Reason: initialize the vector of pointers to null, and set the number to vertex to 0. Free the vector of pointers to avoid memory leakage.
  - class Edge:
    - One constructor and one destructor is created.
    - Reason: initialize the vector of pointers, and delete it to avoid memory leak.
  - class Edge_Node:
    - One constructor and one destructor is created.
    - Reason: initialize the reaseachers' numbers and the weight
  - class illegal_exception:
    - No constructor and one destructor is created.

    - Reason:this class only needs to print out error message, and there is no need to have a constructor and destructor.
  - No operators being override here.
  - Function Parameters:
    - Node:
      - Function without parameter: Node(); ~Node();
    - Graph:
      - Function without parameter: Graph(); ~Graph(); get_size();
      - Function with parameter:
        - string insert(int res_a, int res_b, double wei);
          - The parameter res_a, res_b, and wei are passed by reference to insert the vertices, and the weight into the graph.
        - void print(int res_a);
          - The parameter res_a is passed by reference to print out the specific vertex res_a with its adjacent nodes.
        - string deletion(int res_a);
          - The parameter res_a is passed by reference to delete the specific vertex, and its connection to its adjacent vertices.
        - int mst(int res_a);
          - The parameter res_a is passed by reference to find the maximum spanning tree of the specific root res_a.
- **Test Cases**
  - Graph:
    - Insert: (1) insert the vertex that are already in the list. (2) successfully insert the vertex that are not in the vector. (3) successfully store the weight into the list. (4) invalid input
    - Print: (1) print a blank line when vertex has no edges is empty. (2) invalid input (3) output the adjacent vertex successfully.
    - Deletion: (1) invalid input. (2) return failure when the graph is empty or res_a not in the graph. (3) delete a specific vertex successfully.
    - MST: return the number of vertices correctly.
    - Load: load the dataset successfully.
    - Exit: successfully

```
i 15 2 0.5
i 15 2 0.3
size
p 15
i 17 3 0.1
i 20 30 1
i 4000 299 0.8
i 1111111 3333 0.9
p 1111111
p 16
p 20
d 20
p 20
mst 15
mst 17
load
p 1
p 5
p 15
size
```

- **Performance Consideration**
  - Insert: O(|V| + |E|)
    - Implementing an adjacent list to store the graph.
    - conditional branch to check invalid input: O(1)
    - For loop to check if the numbers are in the graph: 2T(v)
    - Insert the number: O(1)
    - Store the second number into the linked list: O(degree)
  - Print: O(degree(a))
    - conditional branch to check invalid input: O(1)
    - For loop to find the location of the input number a: O(v)
    - Print the adjacent nodes of a: O(degree(a))
  - Deletion: O(v*degree)
    - conditional branch to check invalid input: O(1)
    - For loop to find the location of the input number a: O(v)
    - Delete the vertex from the vector O(1)
    - For loop to delete the node that is the specific vertex in the vector O(v*degree)
  - MST: O(|E||log(|v|))
    - This function traverse all the adjacent vertices of input A, and then find the non duplicate adjacent vertices of A's vertices. Until all the vertices connected to A have been visited.
  - Size: O(1)
    - Return the value of num_vertex
  - add_vec: O(1)
    - Add one element into the vector each time
  - del_vec: O(1)
    - Delete one element each time