Project 3 Design

Ji Wu

For this project, I designed four classes: 1. graph 2. node 3. pri_queue 4. illegal_exception.

- The graph class contains two matrix representing the relationship and edge between each vertex and an array representing the degree of each vertex. It also holds the size and number of edges in the graph. Member functions can be used to access and modify the private variables.
- The node class represent each element of the priority queue. Each node represents a vertex in the priority queue. It holds the index and key of the vertex. Member functions can be used to access and modify the private variables.
- The pri_queue class contains an array of node as a priority queue. The queue contains heapify, build, extract_min functinos which can be used to modify the priority queue.

(UML is included in the last page)

Detail on Design Decision:

graph:

- Constructor: the constructor initializes the private variables. The graph class represent relationship and edge matrix using vectors, and they will be initialized to be arrays of vectors where the arrays and each vector in array have a size of the input value. The degree array will be initialized to have a size of the input value as well to store the degree of each vertex. Number of edges will be initialized to be 0.
- Destructor will be used to deallocate memory if needed. In this case, p_vertex, p_edge and p_degree requested memory allocation and should be deleted and set to nullptr.
- No operator was overridden.
- Since we do not want the parameters to change during the program, all parameters should be passed using const and reference.

node:

- Constructor: initializes the private variables. q_ind, g-ind and key will be set to input value, parent will be set to nullptr.
- Destructor: will be used to deallocate memory if needed. In this case, no memory needs to be deallocated.
- No operator was overridden.
- Since we do not want the parameters to change during the program, all parameter will be passed using cons and reference except set_parent.

pri_queue:

- Constructor: initializes the private variables. It takes in a root and a size value and initialize p_queue to be an array of nodes. The node with the index of root will be set to key = 0 while others will be set to infinite.

- Destructor: will be used to deallocate memory if needed. In this case, p_queue requested memory and will be deallocated.
- No operator was overridden.
- Since we do not want the parameters to change during the program, all parameters will be passed using const and reference.

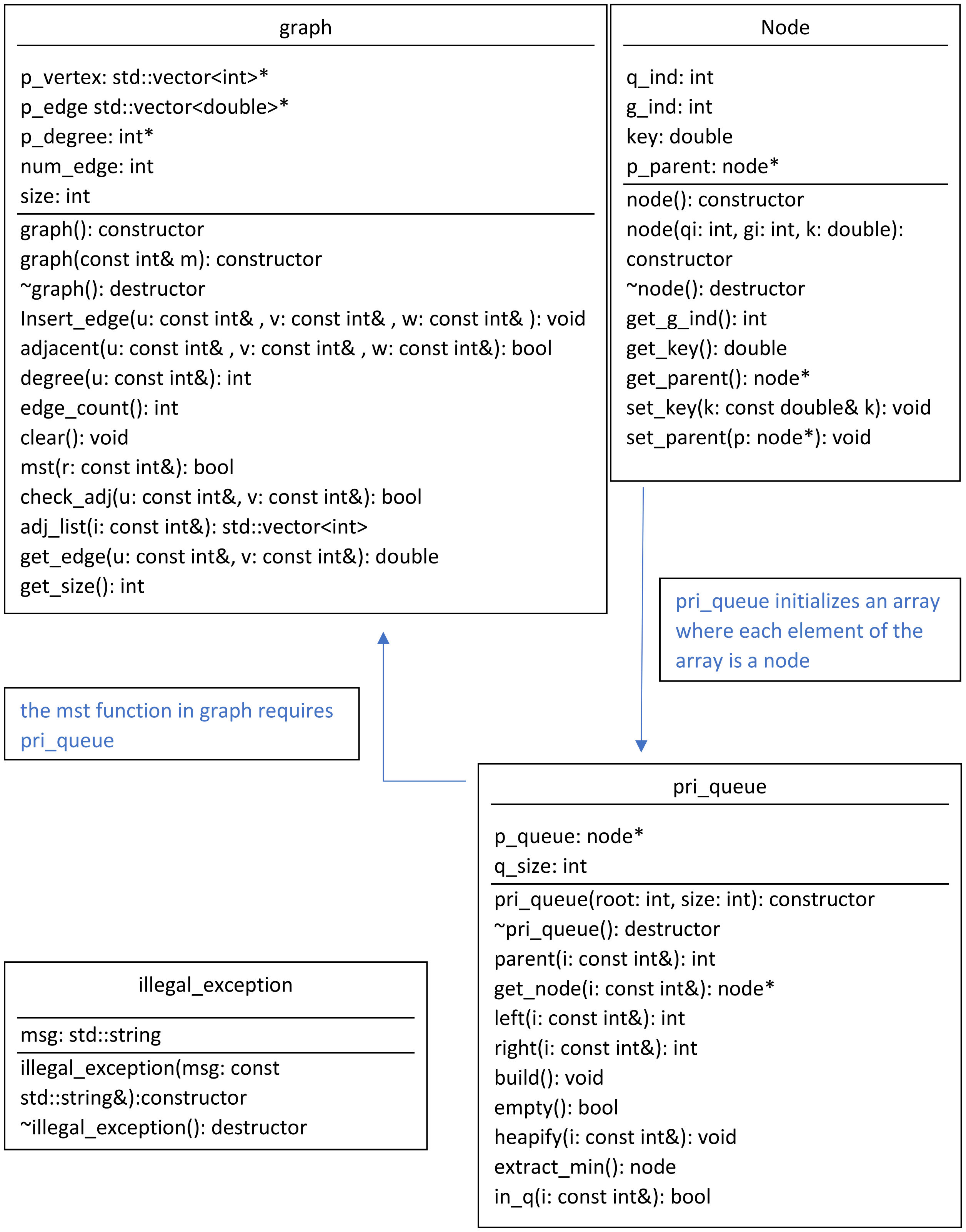
Test cases:

I tested each function separately and then did comprehensive tests. I also tested for illegal arguments.

- Repeatedly insert different edge value to the two same vertexes, use adjacent to check if their edge value. The edge value should be updated every time.
- Repeatedly insert, check adjacent, erase edge and check adjacent between two nodes. The two nodes should not be adjacent after erase.
- Insert multiple times to one vertex between every other vertexes, and check if the degree and number of edges changes correctly. Erase multiple times and check again. Clear and check again.
- Insert to create a connected graph, do mst on every node to see if they always give the same value which is the MST.
- Insert to create a unconnected graph, do mst on every node to see if it always returns “not connected”.

Performance:

- I, e, adjacent, degree, edge_count all require a runtime of O(1). For i, e, and adjacent, we are only modifying or check the entries of arrays. For degree and edge_count, we are only returning the private variables.
- clear requires a runtime of O(m) where m is the number of vertexes in the graph. Clear traverse through the arrays and clear or integers all vectors in it.
- Mst requires O(ElgV) runtime. Extract_min and modify_key requires O(lgV). They will be called in loops, giving the function O(ElgV) runtime.



graph

p_vertex: std::vector<int>*
p_edge std::vector<double>*
p_degree: int*
num_edge: int
size: int

graph(): constructor
graph(const int& m): constructor
~graph(): destructor
Insert_edge(u: const int& , v: const int& , w: const int&): void
adjacent(u: const int& , v: const int& , w: const int&): bool
degree(u: const int&): int
edge_count(): int
clear(): void
mst(r: const int&): bool
check_adj(u: const int&, v: const int&): bool
adj_list(i: const int&): std::vector<int>
get_edge(u: const int&, v: const int&): double
get_size(): int

Node

q_ind: int
g_ind: int
key: double
p_parent: node*

node(): constructor
node(qi: int, gi: int, k: double): constructor
~node(): destructor
get_g_ind(): int
get_key(): double
get_parent(): node*
set_key(k: const double& k): void
set_parent(p: node*): void

pri_queue initializes an array where each element of the array is a node

the mst function in graph requires pri_queue

pri_queue

p_queue: node*
q_size: int

pri_queue(root: int, size: int): constructor
~pri_queue(): destructor
parent(i: const int&): int
get_node(i: const int&): node*
left(i: const int&): int
right(i: const int&): int
build(): void
empty(): bool
heapify(i: const int&): void
extract_min(): node
in_q(i: const int&): bool

illegal_exception

msg: std::string

illegal_exception(msg: const std::string&):constructor
~illegal_exception(): destructor