DATASTRUCTURE AND ALGORITHM – PHASE 1

PROGRAMMING ASSIGNMENT DOCUMENTATION

Name: Thuy Phuong Nhi Tran

Student no: H291937

Email: thuyphuongnhi.tran@tuni.fi

I. Data structure explanations:

- Struct for storing place information including distinguishing id, name, place type and coordinates of place:

```
struct Place {
        PlaceID place_id;
          Name place_name;
        PlaceType place_type;
         Coord place_coord;
    };
```

- Struct for storing area information including distinguishing id, name, vector of multi coordinates and parent area id (one sub area can only have one parent but parent can have many subareas):

```
struct Area {
        AreaID area_id;
        Name area_name;
           std::vector<Coord> area_coords;
         AreaID parent_area_id;
          };
```

- Container for storing all Place struct is unorder_map with the unique key is the Place id and value corresponding to is Place struct. As the key of unordered_map needs to be unique and all method using Place ID to find a specific information, so Place ID is suitable to be the key. Additionally, we do not need to the map to be in ascending order according to the Place ID and to reduce the time accessing to one element using Place ID, so unorder_map is suitable:

```
std::unordered_map<PlaceID, Place> place_container;
```

- Container for storing all Area struct is unorder_map with the unique key is the Area ID and value corresponding to is Area struct. As the key of unordered_map needs to be unique and all method using Area ID to find a specific information, so Area ID is suitable to be the key.

Additionally, we do not need to the map to be in ascending order according to the Place ID and to reduce the time accessing to one element using Place ID, so unorder_map is suitable:

std::unordered_map<AreaID, Area> area_container;

- Container in order to reduce the time for sorting Place alphabetically and find all places with specific name. As we need to sort alphabetically, so the key will be the name of the place and the corresponding value is Place ID. As the name can be the same for many Places which are different in Place ID, so multimap is chosen.

  std::multimap<Name, PlaceID> place_name_container;

Additionally to these data structures, there can be two multimaps similar to place_name_container that PlaceType will be key with Place Id be value and another one that distance from the origin to be key with Place ID be value. It will increases the performance of some functions but also worse the other functions performance and memory for storing. Therefore, these data structures is not the necessary

II. Asymptotic Performance of implementation

Function	Asymptotic performance	Brief explanation
place_count()	O(1)	Equal to map.size()
clear_all()	O(n)	Equal to map.clear()
all_place()	O(n)	loop through the map to retrieve all keys.
add_place()	O(log(n))	Equal to map.insert()
get_place_name_type()	O(1)	Find the value of the given id and return the name and place type (equal to operation[] of map)
get_place_coord()	O(1)	Find the value of the given id and return the coordinate (equal to operation[] of map)
place_alphatetically()	O(n)	Loop through the multimap having Place name as key.
place_coord_order()	O(nlog(n))	Using sort build-in function with lambda function (O(nlogn)). Initialize the result vector containing the PlaceID according to the name (O(n))
find_places_name()	O(log(n))	find range of element in multimap having given name (equal to performance multimap.equal_range()).
find_places_type()	O(n)	loop through the map (linear depending on the size of the map)
change_place_name()	O(log(n))	Remove element in multimap (find range and remove, equal to multimap.equal_range()) and insert new item to multimap (map.insert())
change_place_coord()	O(1)	find the value of the given id and change the coord (equal to operation[] of map)

add_area()	O(1)	add element to the map (equal to map.insert())
get_area_name()	O(1)	find the value of the given id and return name (equal to operation[] of map)
get_area_coords()	O(1)	find the value of the given id and return vector of coords (equal to operation[] of map)
all_areas()	O(n)	Loop through the map to retrieve all keys
add_subarea_to_area()	O(1)	Check if the child is belong to another area (equal to operation[] of map). Access to the child area to change its propertie (equal to operation[] of map)
subarea_in_areas()	O(n)	Loop to find all parent are of given area. In the loop push the AreaID to result vector (equal to vector.push_back()) and find the parent of the present parens (equal to operation[] of map).
creation_finished()	O(1)	Do nothing
all_subareas_in_area()	O(n^2)	loop through the area map and find the parents of each area (O(n)) and then find if the given id is in that vector (O(n))
places_closest_to()	O(nlogn)	Initialize a vector containing only Place (O(n)) (loop through the map). Using sort build-in function with lambda function (equal to std::sort). Initialize the result vector containing the PlaceID according to the name (loop through the sorted one). If size of result is larger than 3, it will only take the three last element.
remove_place()	O(log(n))	Remove element in multimap (equal to multimap.equal_range). Erase element by position/iterator (equal to map.erase() with iterator)
common_area_of_subareas()	O(n^2)	Get all parents of given ids (linear). For loop to loop through one of the parents vector and find if the other vector contains (n^2)

3. Result

- Simple test result:

There is no differences in the result

- Performance result: Attachment in the perf-result.txt