Course : TIE-2010x Data structures & Algorithm

Student ID : 281249

Name : Phan Vu Thien Quang

- Asymtotic performance:

Function	Asymtotic perf	Short rationale for estimate
stop_count()	O(1)	Const time for size()
clear_all()	O(n)	Linear for clear()
all_stops()	O(n)	Travers through n elements of a map
add_stops()	O(log n)	Map [] operator and multimap::insert
get_stop_name()	O(log n) ~ Θ(1)	Map [] operator
get_stop_coord()	O(log n) ~ Θ(1)	Map [] operator
stops_alphabetically()	O(n)	Travers through n elements of a multimap
stops_coord_order()	O(n log n)	Std::sort complexity
min_coord()	O(n)	Log n for equal_range(), but worst case is n when all stops have same coord.
max_coord()	O(n)	Log n for equal_range(), but worst case is n when all stops have same coord.
find_stops()	O(n) ~ Θ(log n)	Worst case is n when all stops have same name. On average, log n for equal_range().
change_stop_name()	O(n) ~ Θ(1)	Worst case is n when all stops have same name but on average, it is const time.
change_stop_coord()	O(n) ~ Θ(1)	Worst case is n when all stops have same coord but on average, it is const time.
add_region()	O(log n)	Map [] operator
get_region_name()	O(log n)	Map [] operator
all_regions()	O(n)	Travers through n elements of a map
add_stop_to_region()	O(n)	Std::find complexity is at most n
add_subregion_to_region()	O(n)	Std::find complexity is at most n

stop_regions()	O(n) ~ Θ(1)	Worst case is n when going from leaf to root. On average it is const.
creation_finished()	Θ(1)	Left empty
region_bounding_box()	O(n)	Std::min(max)_element complexity
stops_closest_to()	O(n log n)	Std::sort complexity
remove_stop()	O(n) ~ Θ(log n)	Worst case is n when going from leaf to root, but on average it is log n for equal_range().
stops_common_region()	O(n) ~ Θ(log n)	At most n for std::find_first_of and two loops when going from leaf to root of n elements.
existStop()	O(log n)	Map::find operation
existRegion()	O(log n)	Map::find operation
get_stops_fromRegion()	O(n)	Worst case is n for loop and recursive parts
compCoord()	Θ(1)	Const time

- Datastructures explaination:

```
struct Coord {
        int x = NO_VALUE;
        int y = NO_VALUE;
}
```

⇨ Pre-defined struct for storing stop's coordinates.

```
struct Stop {
        Name name;
        Coord coord;
        RegionID parent;
}
```

⇨ Structure to save properties of a stop: name, coord and parent region ID.

```
Struct Region {
        RegionID parent;
        Name name;
        std::vector<RegionID> subregions;
        std::vector<StopID> stops;
}
```

- ⇨ Struct to save properties of a region: paren region ID, name, subregions and direct stops.
  I am using vector as insert to vector is faster than unordered_map and I do not actually have a key for them to store in map.

- ➢ std::unordered_map<StopID, Stop> stops_map_
- ⇨ main container, store Stops struct using StopID (unique) as key. Use unordered_map because I do not need the sorting property of map and also take advantage of its asymtotically performance over map.

- ➢ std::multimap<Name, StopID> names_map_
- ⇨ container for names. Use multimap because it is already sorted based on the key, and different stops can have similar names. This multimap suits the stops_alphabetically() and find_stops() functions.

- ➢ std::multimap<int, StopID> distance_map_
- ⇨ container for distances from stops to the origin. Use multimap as it is sorted and different stops can have the same distance. Thanks to the sorting property, min and max coord can be computed easily.

- ➢ std::unordered_map<RegionID, Region> regions_map_
- ⇨ main container for Regions using RegionID(unique) as key. Same reason as stops_map_.

** Of course other containers can also be implemented, such as std::multimap<Coord, StopID, compFunction> for stops_coord_order(), I only choose the four most general maps to store values because they give a relatively balanced performance for all functions. Even perftest-sorting yields time-out at N = 300000 while other functions result in small run time. If I add such containers to the program, sorting performance will only increase a little bit, but other functions will run slower by about 5 sec.

- Testing:
  1. Testread with “example-all-in.txt” and “Tretest-all-in.txt”. Both result in

     ```
     **No differences in output.**
     Testread-tests have been run, no differences found.
     ```

  2. Performance test: result in file “result.txt”