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



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