# Assignment1

Yan Feng 281679

## The reasons for choosing the data structures used in the assignment:

1. Select beaid_beacons, lightbeams and out_lightbeams as the unordered_map to organize the beacons and the lightsources:
   Different from keys in map, keys in unordered_map can be stored in any order, so unordered.
   Therefore, from the first figure below, we can see that time complexity of unordered _map operations is O (1) on average while time complexity of map is O(Log n) as it is implemented as balanced tree structure.
   The unordered_map called as beaid_beacons selects the BeaconID as the key and the minor pointer to Beacon as the corresponding value for better performance and reduced space. In this way, we could get the information of specified beacon through the BeaconID.
   The unordered_map called as lightbeams collecting two BeaconID items as the key and value separately to store the relationship about the lightsources.
   In order to achieve the better time complexity for some functions, the unordered_map called as out_lightbeams collecting BeaconID of the target beacon as the key and storing the corresponding BeaconIDs of source beacons in a vector as the value.

2. Select alphalist and brightness_list using the set which store the pointers to the beacons in order to reorganize the beacons according to alphabetical order of beacon names and increasing brightness of becons' own colors.
   In this way, applying the flags: alphalist_valid and brightness_valid to check the if the alphalist and brightness_list had been constructed or not. And the vectors storing the pointers reduce the space.

3. Select alphaids and brightnessids constructed based on the vector as the return containers for the requirements of the following finctions.

| data-structure | add to end | add elsewhere | remove 1st elem. | remove elem. | nth elem. (index) | search elem. | remove largest |
|---|---|---|---|---|---|---|---|
| array | | | | | $O(1)$ | $O(n)$[2] | |
| vector | $O(1)$ | $O(n)$ | $O(n)$ | $O(n)$[1] | $O(1)$ | $O(n)$[2] | $O(n)$[3] |
| list | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(n)$ | $O(n)$ | $O(n)$[3] |
| deque | $O(1)$ | $O(n)$[4] | $O(1)$ | $O(n)$[1] | $O(1)$ | $O(n)$[2] | $O(n)$[3] |
| stack[9] | $O(1)$ | | | $O(1)$[5] | | | |
| queue[9] | | $O(1)$[6] | | $O(1)$[7] | | | |
| priority queue[9] | | $O(\log n)$ [10] | | | | | $O(\log n)$[8] |
| set (multiset) | | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ |
| map (multimap) | | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ |
| unordered_ (multi)set | | $O(n)$ $\approx \Theta(1)$ | | $O(n)$ $\approx \Theta(1)$ | | $O(n)$ $\approx \Theta(1)$ | $O(n)$ $O(n)$ |
| unordered_ (multi)map | | $O(n)$ $\approx \Theta(1)$ | | $O(n)$ $\approx \Theta(1)$ | | $O(n)$ $\approx \Theta(1)$ | $O(n)$ $O(n)$ |

Figure 1: time complexity

## Operation analysis and performance estimation:

1) Function: int beacon_count();
   Estimate of performance: O(n)
   Short rationale for estimate: The simple counting operation is used to loop the elements in unordered_map structure.

2) Function: void clear_beacons();
   Estimate of performance: O(n)
   Short rationale for estimate: The clear function is based on erase all element. So, the complexity depends on the size of map.

3) Function: std::vector<BeaconID> all_beacons();
   Estimate of performance:O(n)
   Short rationale for estimate: It is based on the simple loop of this beaid_beacons map.

4) Function: bool add_beacon(BeaconID id, std::string const& name, Coord xy, Color color);
   Estimate of performance: O(log(n))
   Short rationale for estimate: Since beaid_beacons is unordered_map, but the types of alphalist and brightness_list are set. The average complexity of adding elemnent is log(n).

| N | cmd(sec) | sec/N | sec/(logN) |
|---|---|---|---|
| 10 | 0.000191 | 1.91E-05 | 5.7406E-05 |
| 30 | 0.00024 | 7.99E-06 | 4.887E-05 |
| 100 | 0.000632 | 6.32E-06 | 9.5125E-05 |
| 300 | 0.002065 | 6.88E-06 | 0.000251 |
| 1000 | 0.007604 | 7.6E-06 | 0.00076298 |
| 3000 | 0.023122 | 7.71E-06 | 0.00200174 |
| 10000 | 0.088434 | 8.84E-06 | 0.00665529 |
| 30000 | 0.281492 | 9.38E-06 | 0.01892679 |
| 100000 | 0.999143 | 9.99E-06 | 0.0601544 |
| 300000 | 3.28122 | 1.09E-05 | 0.18034029 |

Figure 2: add_beacon

5) Function: std::string get_name(BeaconID id);
Estimate of performance:O(n)≈θ(1)
Short rationale for estimate:This operation equals the search operation in unordered map.

6) Function: Coord get_coordinates(BeaconID id);
Estimate of performance:O(n)≈θ(1)
Short rationale for estimate:This operation equals the search operation in unordered map.

7) Function: Color get_color(BeaconID id);
Estimate of performance:O(n)≈θ(1)
Short rationale for estimate:This operation equals the search operation in unordered map.

8) Function: std::vector<BeaconID> beacons_alphabetically();
Estimate of performance: O(n)
Short rationale for estimate: Because the sorting process had been finished in add_beacon function and the alphalist is the ordered set according to alphabetical order of beacon names. So, the left process in this function is to put the elements in alphalist into vector alphaids. Since vector adds new element to the beck is O (1). Hence, collecting all the elements is O(n).

9) Function: std::vector<BeaconID> beacons_brightness_increasing();
Estimate of performance: O(n)
Short rationale for estimate: Because the sorting process had been finished in add_beacon function and the brightness_list is the ordered set according to increasing brightness of becons' own colors. So, the left process in this function is to put the elements in brightness_list into vector brightnessids. Since vector adds new element to the beck is O (1). Therefore, collecting all the elements is O(n).

| N | cmd(sec) | sec/N | sec/(NlogN) | sec/(N^2) |
|---|---|---|---|---|
| 10 | 0.01111 | 0.001111 | 0.00033445 | 0.000111 |
| 30 | 0.01005 | 0.000335 | 6.8274E-05 | 1.12E-05 |
| 100 | 0.015824 | 0.000158 | 2.3818E-05 | 1.58E-06 |
| 300 | 0.024155 | 8.05E-05 | 9.7846E-06 | 2.68E-07 |
| 1000 | 0.074375 | 7.44E-05 | 7.463E-06 | 7.44E-08 |
| 3000 | 0.294467 | 9.82E-05 | 8.4978E-06 | 3.27E-08 |
| 10000 | 1.17461 | 0.000117 | 8.8398E-06 | 1.17E-08 |
| 30000 | 3.62074 | 0.000121 | 8.115E-06 | 4.02E-09 |
| 100000 | 11.0375 | 0.00011 | 6.6452E-06 | 1.1E-09 |

Figure 3: sort_alpha sort_brightness

10) Function: BeaconID min_brightness();
Estimate of performance: O(1)
Short rationale for estimate: If add_beacon function had been applied , the brightness_list had been sorted and organized. The left operation is to take the first element: O (1).

11) Function: BeaconID max_brightness();
Estimate of performance:O(1)
Short rationale for estimate: If add_beacon function had been applied, the brightness_list had been sorted and organized. The left operation is to take the last element: O(1).

| N | cmd(sec) |
|---|---|
| 10 | 0.001839 |
| 30 | 0.002073 |
| 100 | 0.001856 |
| 300 | 0.001894 |
| 1000 | 0.002151 |
| 3000 | 0.002139 |
| 10000 | 0.002439 |
| 30000 | 0.002432 |
| 100000 | 0.002527 |
| 300000 | 0.002345 |
| 1000000 | 0.002492 |

Figure 4: min_max_brightness

12) Function: std::vector<BeaconID> find_beacons(std::string const& name);
Estimate of performance: O(n)
Short rationale for estimate: The key point is to loop the elements in unordered_map and compare: O(n). Pushing the suitable elements into vector is O (1). So, this operation is O(n).

13) Function: bool change_beacon_name(BeaconID id, std::string const& newname);
Estimate of performance:O(log(n))
Short rationale for estimate: The basic idea is to search element with the specified id in

beaid_beacons and to change the corresponding name. This operation is (O(n)≈θ (1)). However, sorted sequence in set alphalist also need to be changed. Since the erase () and insert() in set is O(log(n)). Thus, the total time complexity shall be O(log(n))

14) Function: bool change_beacon_color(BeaconID id, Color newcolor);
Estimate of performance: O(log(n))
Short rationale for estimate: The basic idea is to search element with the specified id in beaid_beacons and to change the corresponding color. This operation is (O(n)≈θ (1)). However, sorted sequence in set brightness_list also need to be changed. Since the erase () and insert() in set is O(log(n)). Thus, the total time complexity shall be O(log(n))

| N | cmd(sec) | sec/N | sec/(logN) |
|---|---|---|---|
| 10 | 0.021757 | 0.002176 | 0.00654957 |
| 30 | 0.021878 | 0.000729 | 0.00445871 |
| 100 | 0.022488 | 0.000225 | 0.00338481 |
| 300 | 0.024814 | 8.27E-05 | 0.00301546 |
| 1000 | 0.026121 | 2.61E-05 | 0.00262111 |
| 3000 | 0.027852 | 9.28E-06 | 0.0024113 |
| 10000 | 0.03214 | 3.21E-06 | 0.00241876 |
| 30000 | 0.035038 | 1.17E-06 | 0.00235583 |
| 100000 | 0.039234 | 3.92E-07 | 0.00236213 |
| 300000 | 0.045539 | 1.52E-07 | 0.00250289 |
| 1000000 | 0.050342 | 5.03E-08 | 0.00252575 |

Figure 5: exchange name and color

15) Function: bool add_lightbeam(BeaconID sourceid, BeaconID targetid);
Estimate of performance:O(n)≈θ(1)
Short rationale for estimate: For unordered_map, adding element is O(n)≈θ(1).

16) Function: std::vector<BeaconID> get_lightsources(BeaconID id);
Estimate of performance: O(n)≈θ(1)
Short rationale for estimate: At first, the find function in unordered map out_lightbeams is O(n)≈θ(1). And the next step is to sort the small vector. Thus, the total time complexity is O(n)≈θ(1).

17) Function: std::vector<BeaconID> path_outbeam(BeaconID id);
Estimate of performance: O(n)≈θ(1)
Short rationale for estimate: The worst case is to use "while" to loop the elements in unordered_map lightbeams :O(n). The average time complexities both for the vector to push element and the unordered_map to find are O (1). Thus. the total average time complexity is O(n)≈θ(1).

| N | cmd(sec) | sec/N | sec/(NlogN) | sec/(N^2) |
|---|---|---|---|---|
| 10 | 0.019899 | 0.00199 | 0.00059901 | 0.000199 |
| 30 | 0.022497 | 0.00075 | 0.00015283 | 2.5E-05 |
| 100 | 0.025728 | 0.000257 | 3.8724E-05 | 2.57E-06 |
| 300 | 0.022315 | 7.44E-05 | 9.0392E-06 | 2.48E-07 |
| 1000 | 0.026388 | 2.64E-05 | 2.6479E-06 | 2.64E-08 |
| 3000 | 0.029988 | 1E-05 | 8.654E-07 | 3.33E-09 |
| 10000 | 0.036851 | 3.69E-06 | 2.7733E-07 | 3.69E-10 |
| 30000 | 0.037611 | 1.25E-06 | 8.4295E-08 | 4.18E-11 |
| 100000 | 0.040434 | 4.04E-07 | 2.4344E-08 | 4.04E-12 |
| 300000 | 0.080573 | 2.69E-07 | 1.4761E-08 | 8.95E-13 |
| 1000000 | 0.042424 | 4.24E-08 | 2.1285E-09 | 4.24E-14 |

Figure 6: path_outbeam

18) Function: bool remove_beacon(BeaconID id);
Estimate of performance:O(n)
Short rationale for estimate: Besides the erasing operation in unordered_map beaid_beacons: O(n)≈θ(1)
We need loop the elements in lightbeams, alphalist, brightness_list, alphaids, brightnessids and out_lightbeams to erase the suitable element: O(n)

19) Function: std::vector<BeaconID> path_inbeam_longest(BeaconID id);
Estimate of performance: O(n)≈θ(1)
Short rationale for estimate: In this place, I applied the DFS idea. In this way, the algorithm would visit beacons before the given beacon and get its lightsources. Because get_lightsources is O(n)≈θ(1). So the total average time complexity is O(n)≈θ (1).

| N | cmd(sec) | sec/N | sec/(NlogN) | sec/(N^2) |
|---|---|---|---|---|
| 10 | 0.061898 | 0.00619 | 0.00186332 | 0.000619 |
| 30 | 0.090854 | 0.003028 | 0.00061719 | 0.000101 |
| 100 | 0.06568 | 0.000657 | 9.8857E-05 | 6.57E-06 |
| 300 | 0.080301 | 0.000268 | 3.2528E-05 | 8.92E-07 |
| 1000 | 0.087966 | 8.8E-05 | 8.8268E-06 | 8.8E-08 |
| 3000 | 0.098361 | 3.28E-05 | 2.8385E-06 | 1.09E-08 |
| 10000 | 0.114776 | 1.15E-05 | 8.6378E-07 | 1.15E-09 |
| 30000 | 0.103177 | 3.44E-06 | 2.3125E-07 | 1.15E-10 |
| 100000 | 0.097073 | 9.71E-07 | 5.8444E-08 | 9.71E-12 |
| 300000 | 0.098147 | 3.27E-07 | 1.7981E-08 | 1.09E-12 |
| 1000000 | 0.102003 | 1.02E-07 | 5.1177E-09 | 1.02E-13 |

Figure 7: path_inbeam_longest

20) Function: Color total_color(BeaconID id);
Estimate of performance: O(n)≈θ(1)
Short rationale for estimate: Similar to the operation "path_inbeam_longest", only visit once for each beacon before the given beacon and the a set of operations applied on this beacon is θ(1).

| N | cmd(sec) | sec/N | sec/(NlogN) | sec/(N^2) |
|---|---|---|---|---|
| 10 | 0.134396 | 0.01344 | 0.00404572 | 0.001344 |
| 30 | 0.141407 | 0.004714 | 0.0009606 | 0.000157 |
| 100 | 0.224066 | 0.002241 | 0.00033725 | 2.24E-05 |
| 300 | 0.235114 | 0.000784 | 9.524E-05 | 2.61E-06 |
| 1000 | 0.198654 | 0.000199 | 1.9934E-05 | 1.99E-07 |
| 3000 | 0.187522 | 6.25E-05 | 5.4115E-06 | 2.08E-08 |
| 10000 | 0.2117 | 2.12E-05 | 1.5932E-06 | 2.12E-09 |
| 30000 | 0.297046 | 9.9E-06 | 6.6575E-07 | 3.3E-10 |
| 100000 | 0.274756 | 2.75E-06 | 1.6542E-07 | 2.75E-11 |
| 300000 | 0.272364 | 9.08E-07 | 4.9898E-08 | 3.03E-12 |
| 1000000 | 0.747002 | 7.47E-07 | 3.7478E-08 | 7.47E-13 |

Figure 8: total_color