Kuanysh Kairbek

256285

Readme.pdf

Homework assignment 2.

As my basic data structure, I have used unordered map because of constant average time complexity on average access to elements, that is called all the time during execution of the program. In addition, I use 2 vectors of keys of map sorted in alphabetical and salary related orders.

void add_person(string name, PersonID id, string title, Salary salary);

I used insert member function of unordered map. Single element insertions: Average case: constant. Worst case: linear in container size.

void remove_person(PersonID id);

To find element I have used find member function which is constant time complexity and I have used erase member function of unordered map to remove element which is constant with iterators and keys.

string get_name(PersonID id), string get_title(PersonID id) and Salary get_salary(PersonID id);

For all these functions I have used find member function with const time complexity.

vector<PersonID> find_persons(string name);, vector<PersonID> personnel_with_title(string title);

Iterator through all elements of the map. Push found elements into the vector. Sort function from <algorithm> with N*log(N) time complexity.

void change_name(PersonID id, string new_name), void change_salary(PersonID id, Salary new_salary), void add_boss(PersonID id, PersonID bossid);

Find member function with iterator through the map. Constant complexity.

unsigned int size();

size member function. Constant complexity.

void clear();

clear member function. Linear complexity.

vector<PersonID> underlings(PersonID id);

Iterator through all elements of the map. Push found elements into the vector. Sort function from <algorithm> with N*log(N) time complexity.

vector<PersonID> personnel_alphabetically(), vector<PersonID> personnel_salary_order();

Copy PersonID into the vector and sort it. Sort function from <algorithm> with N*log(N) time complexity.

PersonID find_ceo();

Iterator through the map. Constant time complexity.

PersonID nearest_common_boss(PersonID id1, PersonID id2);

Traverse through data structure and mark visited to check common ‘nodes’. Complexity approx. linear.

PersonID min_salary();

Min_element fuction from <algorithm>. Linear complexity.

PersonID max_salary();

Max_element fuction from <algorithm>. Linear complexity.

PersonID median_salary(), PersonID first_quartile_salary(), PersonID third_quartile_salary();

Calculated with size member function. Constant complexity.