Shiqi Wang

CS 32

Project 2

Report

Design of the doubly-linked list

The list is circular with a dummy node pointed by the head pointer. The dummy node doesn’t contain any particular value. In a typical Map, the first node starts with head->next, and the last nodes ends with head->prev. When the list is empty, both head->next and head->prev point to the dummy node.

Pseudocode for non-trivial algorithms

1. Map::Map(const Map& other)

Initialize this Map;

Create a pointer p points to the last node of other;

While p doesn’t point to the dummy node of other

Insert the p->m\_key and p->m\_value to this Map;

Move p to the previous node;

1. Map& Map::operator= (const Map& other)

If not aliasing

Using copy constructor to create a temp Map and initialize it to other Map;

Swap temp and this Map;

Return this Map;

1. bool Map::insert(const KeyType& key, const ValueType& value)

Create a pointer temp points to the first node in the list;

While temp doesn’t point to the dummy node

Check whether the key exists in the list, if so, return false;

Move temp to the next node;

Create a new node pointed by p;

Initialize its key and value;

Add it to the front of the list, i.e. right behind the dummy node;

Increment the size of the list;

1. bool Map::update(const KeyType& key, const ValueType& value)

Create a pointer p points to the first node in the list;

While p doesn’t point to the dummy node

Check whether the key exists in the list, if so, update its value and return true;

Move p to the next node;

If no such key exists in the list, return false;

1. bool Map::erase(const KeyType& key)

Create a pointer p points to the first node in the list;

While p doesn’t point to the dummy node

Check whether the key exists in the list, if so, delete it and decrease the size of the list by 1, return true;

Move p to the next node;

If no such key exists in the list, return false;

1. bool Map::get(int i, KeyType& key, ValueType& value) const

if i is undefined

return false;

Create a pointer p points to the first node in the list;

Move p to the i th node, where the first node is denoted by 0th node;

Put the m\_key and m\_value of the i th node into key and value;

return true;

1. void Map::swap(Map& other)

Exchange the m\_size of this Map and other Map;

Exchange the head pointer of this Map and other Map;

1. bool combine(const Map& m1, const Map& m2, Map& result)

Initialize a Boolean t to true;

Use copy constructor to create a temp Map and initialize it to m1;

Repeatedly:

Get a key and its value from m2;

If if temp doesn’t contain the key

Insert the key and its value from m2 into temp;

Else if there is a same key in temp but with different value

Erase the key in the temp and convert t to false;

Use assignment operator= to make result Map equals to temp Map;

Return t;

1. void subtract(const Map& m1, const Map& m2, Map& result)

Initialize an empty map temp;

Repeatedly:

Get a key and its value from m1;

If m2 doesn’t contain the key from m1;

Insert the key and value from m1 into temp;

Use assignment operator= to make result Map equals to temp Map;

Test Cases

////default constructor

Map m;

///for an empty map

assert(m.empty()); //test empty function

assert(m.size() == 0); //test size function

assert(!m.update("Fred", 123)); //nothing to update

assert(!m.erase("Fred")); //nothing to erase

assert(!m.contains("Fred")); //doesn't contain anything

string s1 = "xyz";

double v1 = -9999;

//test get function

assert(!m.get(s1,v1) && v1 == -9999); //doesn't have a key with the same name as s1, do not update v1

assert(!m.get(0,s1,v1) && s1 == "xyz" && v1 == -9999); //doesn't have a key with the same name as s1, do not update s1 and v1

//test insert function

assert(m.insert("Fred", 123) && m.size() == 1);

assert(!m.insert("Fred", 456) && m.size() == 1); //already contains key name "Fred", fail to insert

assert(m.insert("Ethel", 456) && m.size() == 2);

///for non-empty map

assert(!m.empty()); //test empty

assert(m.size() == 2); //test size

assert(m.contains("Fred") && m.contains("Ethel")); //test contains function

//tese get function

assert(!m.get(s1, v1) && v1 == -9999); //v1 should not be updated

assert(m.get("Fred", v1) && v1 == 123); //v1 should be properly updated

v1 = -9999;

assert(m.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "Ethel" && v1 == 456))); //s1 and v1 should be properly updated

string s2;

double v2;

assert(m.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "Ethel" && v2 == 456))); //s2 and v2 should be properly updated, s1 and s2 should not be the same

//test update function

double temp;

assert(m.update("Fred", 789));

assert(m.get("Fred", temp) && temp == 789); //The value of Fred should be properly updated

assert(!m.update("fRED", 123));

assert(!m.contains("fRED")); //doesn't contain "fRED"

//test insertOrUpdate function

assert(m.insertOrUpdate("Fred", 123));

assert(m.get("Fred", temp) && temp == 123); //The value of Fred should be updated

assert(m.insertOrUpdate("", 100));

assert(m.get("", temp) && temp == 100 && m.size() == 3); //The key empty string and its value should be inserted into Map m

//test erase function

assert(!m.erase("xyz")); //nothing to erase

assert(m.erase("Ethel") && !m.contains("Ethel") && m.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "" && v2 == 100))); //erase one node without interfere other nodes

//test swap function

Map m1;

m1.insert("Ethel", 456);

m1.swap(m);

//now m1 contains two nodes , which are "" - 100 and "Fred" - 123; m contains one node, "Ethel" - 456

assert(m.size() == 1 && m1.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m1.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m1.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "" && v2 == 100)));

s1 = "xyz"; v1 = -9999;

assert(m.get(0, s1, v1) && s1 == "Ethel" && v1 == 456);

//aliasing

m1.swap(m1);

assert(m1.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m1.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m1.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "" && v2 == 100)));

////copy constructor

Map m2(m1);

//now m2 contains two nodes , which are "" - 100 and "Fred" - 123

assert(m2.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m2.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m2.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "" && v2 == 100)));

////assignment operator

m2 = m;

//now m2 contains one node, "Ethel" - 456

s1 = "xyz"; v1 = -9999;

assert(m2.size() == 1);

assert(m.get(0, s1, v1) && s1 == "Ethel" && v1 == 456);

//aliasing

m2 = m2;

s1 = "xyz"; v1 = -9999;

assert(m2.size() == 1);

assert(m.get(0, s1, v1) && s1 == "Ethel" && v1 == 456);

////test for combine function

//aliasing && same key has different values in two maps

m.insert("Fred", 456);

//m contains two nodes, "Ethel" - 456 and "Fred" - 456

assert(!combine(m, m1, m)); //function returns false

//now m contains two nodes, "Ethel" - 456 and "" - 100

assert(m.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m.get(0, s1, v1) &&

((s1 == "Ethel" && v1 == 456) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m.get(1, s2, v2) && s1 != s2 &&

((s2 == "Ethel" && v2 == 456) || (s2 == "" && v2 == 100)));

//function returns true && two maps doen't contain any identical key name

Map m3;

m.erase("");

//now m contains one node, "Ethel" - 456

assert(combine(m,m1,m3));

//now m3 contains three nodes, "Fred" - 123, "Ethel" - 456 and "" - 100

assert(m3.size() == 3);

s1 = "xyz"; v1 = -9999;

assert(m3.get(0, s1, v1) &&

((s1 == "Ethel" && v1 == 456) || (s1 == "" && v1 == 100) || (s1 == "Fred" && v1 == 123)));

s2 = "xyz"; v2 = -9999;

assert(m3.get(1, s2, v2) && s1 != s2 &&

((s2 == "Ethel" && v2 == 456) || (s2 == "" && v2 == 100) || (s2 == "Fred" && v2 == 123)));

string s3 = "xyz"; double v3 = -9999;

assert(m3.get(2, s3, v3) && s1 != s3 && s2 != s3 &&

((s3 == "Ethel" && v3 == 456) || (s3 == "" && v3 == 100) || (s3 == "Fred" && v3 == 123)));

//function returns true && same key name has same value in two maps

assert(combine(m,m2,m3));

//now m3 contains one node, "Ethel" - 456

assert(m.size() == 1);

s1 = "xyz"; v1 = -9999;

assert(m.get(0, s1, v1) && (s1 == "Ethel" && v1 == 456));

/\*

m: "Ethel" - 456

m1: "" - 100 && "Fred" - 123

m2: "Ethel" - 456

m3: "Ethel" - 456

\*/

//test for subtract

subtract(m1, m2, m3);

//now m3 contains two nodes, "" - 100 && "Fred" - 123

assert(m3.size() == 2);

s1 = "xyz"; v1 = -9999;

assert(m3.get(0, s1, v1) &&

((s1 == "Fred" && v1 == 123) || (s1 == "" && v1 == 100)));

s2 = "xyz"; v2 = -9999;

assert(m3.get(1, s2, v2) && s1 != s2 &&

((s2 == "Fred" && v2 == 123) || (s2 == "" && v2 == 100)));

//aliasing

subtract(m, m, m);

assert(m.size() == 0);