CS32 Project2 Report

Luna Piao

UID:304959335

I am using a circularly doubly-linked list to implement my Map. Each node holds a pair of key/value data and two pointers to the node—a next pointer, which points to the next node in the list, and a prev pointer, which points to the previous node in the list. There is a dummy header node before the head of the list. This node does not store any elements, but it can ensure that the next or previous nodes exist for every element.

**Empty list**

|  |  |  |
| --- | --- | --- |
| **prev** |  | **next** |
|  |

**Pseudo code for non-trivial functions:**

//copy constructor

Map::Map(const Map& other)

: m\_size(other.m\_size)

{

Initializes new head.

Repeatedly

loop through the other map

Copy key/value from other map and Store it to the adder.

Link the adder to the current list

Rearrange surrounding Nodes

}

//destructor

Map::~Map()

{

Repeatedly

Create a deleter pointing to the end of the list

Connect the second last node to the head

Delete deleter

delete the head

}

int Map::find(const KeyType& key) const

{

if the map is non-empty

{

Create a node p to traverse through the list

Fist, let p points to head

Repeatedly,

Point p to the next item if key is not found

if key is found

return the value of the iterator i

}

If key is not found return -1

}

bool Map::erase(const KeyType& key)

{

Find the position of the key in the existing map

if key is found in the list

{

Create a new node p to traverse through the linked list

When p points to the position of the key

Create a deleter to point to the item to be deleted

rearrange the surrounding items

delete the target item

reduce m\_size by 1

return true;

}

If key is not found, return false;

}

void Map::swap(Map& other)

{

Create a temporary header pointing to the other head

Then other head points to current head

Current head points to temporary head

Swap the two lists’sizes using the same method

}

bool Map::doInsertOrUpdate(const KeyType& key, const ValueType& value,

bool mayInsert, bool mayUpdate)

{

Find the position of the key in the list

if found and allowed to update

{

Create a node p to point to the head

Repeatedly

P points to the next node until reaches to the position of the key

When p points to the node containing key

update the corresponding value

return true

}

If not found, and allowed to insert

{

//insert the new node at the end

Create and point p to the last item in the map

create a new item to store the key and value

link the new item with head and the last item

increase m\_size by 1;

return true;

}

return false;

}

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

{

If result is non-empty

Make result empty

Make result equal to m1

repeatedly

get each key/value pair from the m2 list

if m1 does not have the key already

insert to m1

if m1 already has the key but the corresponding values are different from the value of m2

erase the key from m1, return false

all other situations return true

}

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

{

If result is non-empty

Make result empty

Make result equal to m1

Going through each key from m1

If m2 contains the same key

Erase the key from result

}

**3)Test cases**

The tests were performed on a map from strings to doubles

Map m; //default constructor

assert(m.empty()); //check if it is empty

assert(m.insert("Fred", 123)); //insert data

assert(m.insert("Ethel", 456)); //insert data

assert(m.size() == 2); //check the size

ValueType v = 42;

assert(!m.get("Lucy", v) && v == 42); //test to make sure the get(k,v)function returns false if the key is not in the list

assert(m.get("Fred", v) && v == 123); //test to make sure if the key can be found in the list, the value get reset to the value in the map

v = 42;

KeyType x = "Lucy";

assert(m.get(0, x, v) &&

((x == "Fred" && v == 123) || (x == "Ethel" && v == 456))); //to make sure the key and value parameters are reset to the key and value of one of the key/value pairs in the map

KeyType x2 = "Ricky";

assert(m.get(1, x2, v) &&

((x2 == "Fred" && v == 123) || (x2 == "Ethel" && v == 456)) && x != x2); //to make sure the key and value parameters are reset to the key and value of one of the key/value pairs in the map

Map m0(m); //test copy constructor

assert(m0.size() == 2); //test size gets copied correctly

Map m00;

m00.swap(m0); //test swap function

assert(m00.size() == 2); // the size is correct after the swap

assert(m0.size() == 0); // the size is correct after the swap

assert(m00.insert("Lucy", 777));//insert

assert(!m00.erase("XX")); //make sure we can't erase something that's not in the list

assert(m00.erase("Fred")&&m00.size()==2); //erase the first item in m00 and check the size

assert(m00.erase("Lucy")&&m00.size()==1); //erase the latest item in m00 and check the size

assert(m00.erase("Ethel")&&m00.size()==0); //erase the last item in m00 and check the size

//test the combind functionality

Map m1,m2,m3,result;

m1.insert("Fred", 123);

m1.insert("Ethel", 456);

m1.insert("Lucy", 789);

m2.insert("Lucy", 654);

m2.insert("Ricky", 321);

combine(m1,m2,result);

assert(!result.contains("Lucy")&&result.size()==3);//the result doesn't contain "Lucy" since the value of "Lucy" in m1 and m2 are different

m2.update("Lucy",789); //now update the value of "Lucy" in m2 to 789 to make the value the same as in m1

combine(m1,m2,result);

assert(result.contains("Lucy")&&result.size()==4);//now the result contains "Lucy" and the size is 4

//test the subtract functionality

m3.insert("Lucy", 789);

m3.insert("Ricky", 321);

m3.insert("Ethel", 654);

subtract(m1,m3,result);

assert(!result.contains("Lucy")&&result.contains("Fred")&&result.size()==1);// the result contains "Fred" only