Project 2 Report

Jingjng Nie 30456417

1. Description of the implementation of the linked list.

When a Map is declared, it does not have any pair yet, so the size *m\_size* is set to 0, and the head and tail pointers are set to *nullptr*. Then, when a pair can be added through the function *insert*, a new node is inserted to the empty linked list, and both the head and tail pointers are set to point at this node. Then, when another node is added, it is inserted to the end of the linked list, and pointer tail now points to the newly added node. Similarly, if a node needs to be removed when function *erase* is called, then the node will be removed from either the start, the middle, or the end. A node consists of values *m\_key*, *m\_value,* pointer *previous,* and *pointer next*. They are not in particular orders. It is not a circular linked list and does not have a dummy node. In the list nodes: keyType m\_key, valueType m\_value, Node\* previous, Node\* next.

m\_key

m\_value

previous

next

“Fred”

200

nullptr

20000000

Struct Pair:

Assume address = 100000

m\_size

2

head

10000010

tail

200000

Typical Map:

m\_key

m\_value

previous

next

“Greg”

100

10000000

nullptr

Struct Pair:

Assume address = 200000

m\_size

0

head

nullptr

tail

nullptr

Empty Map:

1. Pseudocode for non-trivial algorithms

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

Use a temporary Map tempResult

set up pointers to point to the new Map and the Maps passed into this function;

set up key and value types to get size and content in each Map;

Assign m1 to tempResult

For (nodes in m2){

Get the key

If (tempResult contains the key){

Erase the key from tempResult

}

}

Assign tempResult to result

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

Use a temporary Map tempResult

set up pointers to point at the Maps passed in

set up int and values to hold the content in the Maps

if m1, m2, and, m3 are the same Map

(do nothing)

else

//traverse through m1

for (each node in m1){

Get key

If (m2 contains the key){

Set boolean to false if the two values are different

Insert to the tempResult if values are the same

}

Insert to the tempResult if not found in m2

}

//traverse through m2

For (each node in m2){

Insert to tempresult if not found in m1

}

return boolean

}

bool Map::erase(const KeyType& key){

set up a pointer to point at the position found for the key

if key not found

return false

if (the node to be deleted is the only one node)

set head and tail to nullptr

else if it is the first node

link head with the next node

else if it is the last node

link tail to the previous node

else

link tail and head to the previous and next node

delete the node

m\_size decreases by 1

return true;

}

1. Test cases:

//KeyTye = string, ValueType = double

//constructor

Map a;

Map b;

KeyType k[6] = {"andy","beta","ciara","diana","emily","fiona"};

KeyType j[5] = {"greg","howard","inga","jenny","kelly" };

ValueType v = 100;

assert(a.size()==0 && b.size()==0 && c.size()==0); //test size()

assert(a.empty() && b.empty() && c.empty()); //test empty()

assert(!a.erase("Ricky")&&!b.erase("R")&&!c.erase("c")); //test erase

//test insert

for (int n = 0; n < 5; n++)

assert(b.insert(j[n], v));

for (int n = 0; n < 6; n++){

a.insert(k[n], v);

}

a.swap(b); //test swap

assert(!b.insert(k[5], v)); //cannot insert with the same key present

assert(b.insert(j[4], v));

assert(a.size()==5 && b.size()==7);

assert(b.contains("andy")); //test contains

//test get(int i, KeyType& key, ValueType& value)

assert(a.get(1,k1,v1)&&(k1 =="greg"||k1 =="howard"||k1=="inga"||k1== "jenny"||k1 =="kelly"));

//test get(const KeyType& key, ValueType& value)

assert(b.get("ciara", v1) && v1 == 100);

Map m3;

m3.insert("Lucy", 1);

//test insertOrUpdate

assert(m3.insertOrUpdate("Fred", 123) && m3.insertOrUpdate("Ethel", 456) && m3.insertOrUpdate("Lucy", 789));

Map m4;

assert(m4.insertOrUpdate("Lucy", 789) && m4.insertOrUpdate("Ricky", 321));

assert(combine(m3, m4, c)); //test combine

assert(c.size()==4)

assert(combine(m3, m3, c)); //test combine and aliasin

assert(combine(m3, m4, m4)); //test combine and aliasing

assert(combine(m3, m4, m3)); //test combine and aliasing

assert(combine(m3, m3, m3)); //test combine and aliasing

m4.update("Lucy", 35); //test update

assert(!combine(m3, m4, c)); //test combine

subtract(m3, m4, c); //test subtract

assert(c.size()==4)

subtract(m3, m3, c); //test subtract and aliasing

assert(c.size()==4)