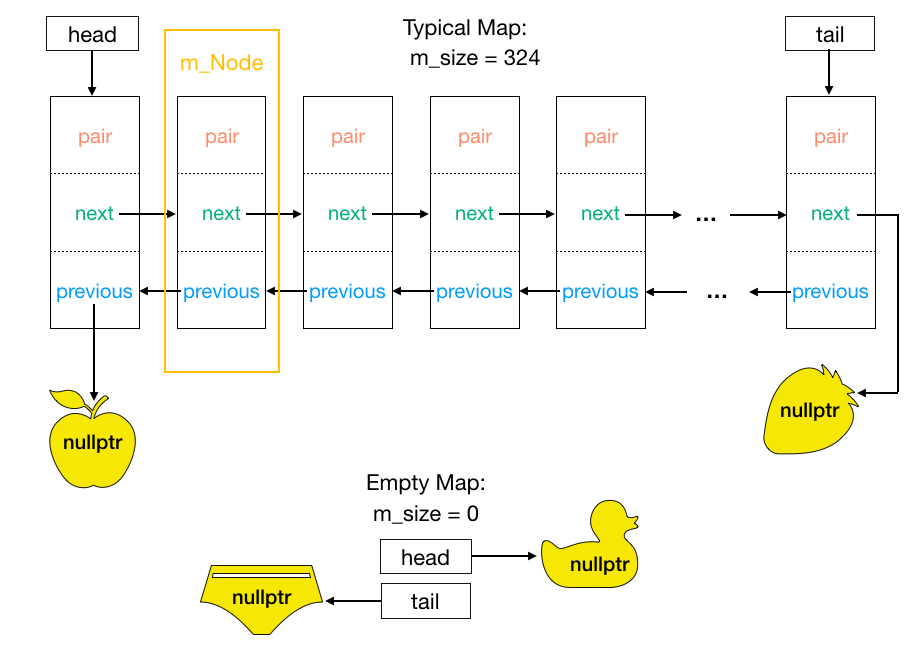
CS 32 Project 2 Report

1. My doubly-linked list is linear with each m\_Node comprised of a pair of m\_Pair type, a next pointer, and a previous pointer. Each pair holds a variable of KeyType and one of ValueType. New nodes are added at the end of the linked list. My design is not circular and does not have a dummy node. Null pointers are shown in random shapes to indicate random memory address in the following graph:
2. Pseudocode for non-trivial algorithms
   * Default Constructor

Set size to 0

Set head/tail to nullptr

* + Destructor

For every node

Set a temporary node to node after target

Delete target node

Continue traversing with temporary node

* + Copy Constructor

Initialize size and head/tail pointers

For every node of parameter Map

Get and insert node values to current Map

* + Assignment Operator

If parameter map is not equal to current map

construct a temporary Map temp that copies rhs

swap the value of temp with current object

Otherwise, do nothing

Return itself

* + bool insert(const KeyType& key, const ValueType& value)

If already contains key

Return false

Create new node and assign values

If empty list assign head/tail to node

Add node to beginning of map

Return true

* + bool update(const KeyType& key, const ValueType& value)

For every node

If a node’s key equals the key parameter

Change parameter value to node value

Return true

Return false

* + bool insertOrUpdate(const KeyType& key, const ValueType& value)

If Map can be updated

Update and return true

Insert new node and return true

* + bool erase(const KeyType& key)

If key is not in Map or list is empty

Return false

If only one node

Delete node, decrement size, return true

If target is head or tail

Delete node and repoint previous/next pointers

If target is in middle

Set next pointer of previous node to that of after

Set previous pointer of following node to that of before

Return false

* + bool contains(const KeyType& key) const

For every node

If node key matches parameter

Return true

Return false

* + bool get(const KeyType& key, ValueType& value) const

For every node

If node key matches parameter

Change value parameter to that of node

Return true

Return false

* + bool get(int i, KeyType& key, ValueType& value) const

If i is out of bound

Return false

Traverse linked list to i

Change key and value

Return true

* + void swap(Map& other)

Make temp variables for head, tail and size

Swap head and tail pointers

Swap linked list sizes

* + void combine(const Map& m1, const Map& m2, Map& result)

Make a temp Map of parameters m1

Make a temp bool set to true

Swap the temp Map with result

Go through every value of other temp2 map

Get values of temp2

If temp does not have that node

Add node to result

If temp has node but different values

Erase node from result

Set bool to false

Return bool

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

Make temp Maps of parameters m1 and m2

Construct empty map

Swap empty map with result

For every value of temp map

Get values of temp map

If other temp map does not have node

Insert node to result map

1. Test Cases

//These tests were performed on maps from strings to doubles

//These tested the main functions from Homework 1 plus the two new functions combine() and subtract()

//Default constructor, empty(), size(), insert(), update(), insertOrUpdate(), erase(), contains(), get(), swap(), combine(), and subtract()

Map m; //Default constructor

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

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

m.insert("A", 1.23); //test inserting

m.insert("B", 2.34);

m.insert("C", 3.45);

assert(m.size() == 3); //test size

KeyType k = "B"; //Setting values and keys

ValueType v = 9.87;

KeyType k1 = "D";

ValueType v1 = 2.36;

KeyType k2 = "A";

ValueType v2 = 5.67;

KeyType k3 = "B";

ValueType v4 = 5.89;

ValueType v3;

KeyType key;

ValueType val;

assert(m.get(0, key, val) && key == "A" && val == 1.23); //testing get

assert(!m.get(-1, key, val) && key == "A" && val == 1.23); //testing invalid i in get

assert(m.contains("A") && m.contains("B") && !m.contains("D"));

assert(m.update(k, v) && m.get(k, v3) && v3 == 9.87); //testing update

assert(!m.insert(k3, v4) && m.get(k3, v3) && v3 == 9.87); //testing invalid insert

assert(m.insertOrUpdate(k2, v2) && m.get(k2, v3) && v3 == 5.67); //testing insertOrUpdate

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

assert(m.insertOrUpdate(k1, v1) && m.get(k1, v3) && v3 == 2.36);

assert(m.size() == 4);

assert(m.erase("C") && m.size() == 3); //testing erase

assert(!m.get("C", v3)); //testing erase/invalid get

assert(v3 == 2.36 && !m.contains("C"));

//Testing swap

Map m1;

m1.insert("Z", 3.21);

m1.insert("Y", 5.43);

m.swap(m1); //testing swap

assert(m.contains("Z") && m.contains("Y"));

assert(m1.contains("A") && m1.size() == 3);

//Testing assignment operator and copy constructor

ValueType vv;

Map m2(m1); //testing copy constructor

assert(m2.contains("A") && m2.contains("B") && m2.get("B", vv) && vv == 9.87 && m2.size() == 3);

Map m3;

m3.insert("K", 1.23);

m3 = m1; //testing assignment operator

assert(m3.contains("D") && m3.contains("B") && m3.get("D", vv) && vv == 2.36 && m3.size() == 3);

assert(m3.erase("A") && m3.size() == 2); //testing erasing head

m3 = m2;

assert(m3.erase("B") && !m3.contains("B")); //testing erasing middle

m3 = m1;

assert(m3.erase("D") && m3.size() == 2); //testing erasing tail

//testing combine and subtract and aliasing

Map map;

map.insert("a", 0);

map.insert("b", 1);

map.insert("c", 2);

map.insert("d", 3);

map.insert("e", 4);

Map map4(map);

Map q(map);

Map r(map);

Map map1;

map1.insert("d", 3);

map1.insert("e", 6);

map1.insert("f", 7);

Map w(map1);

Map result;

assert(!combine(map, map1, result) && result.size() == 5 && !result.contains("e") && result.contains("d")); //testing combine

assert(!combine(map, map1, map) && map.size() == 5 && !map.contains("e") && map.contains("d")); //testing aliasing

Map map2;

map2.insert("h", 7);

map2.insert("g", 8);

assert(combine(map4, map2, result) && result.size() == 7 && !result.contains("f")); //testing combine with true

Map y;

y.insert("a", 1);

y.insert("b", 2);

y.insert("c", 3);

y.insert("d", 4);

y.insert("e", 5);

Map u;

u.insert("a", 5);

u.insert("b", 4);

u.insert("c", 8);

u.insert("d", 2);

u.insert("e", 1);

Map i;

assert(!combine(y, u, i)); //checks if none are combined

assert(i.empty());

Map results;

subtract(q, w, results); //testing subtract

assert(results.size() == 3 && results.contains("a") && !results.contains("d") && !results.contains("f"));

subtract(q, w, q); //testing aliasing

assert(q.size() == 3 && q.contains("a") && !q.contains("d") && !q.contains("f"));

Map g;

g.insert("a", 1);

g.insert("b", 2);

g.insert("c", 3);

g.insert("d", 4);

g.insert("e", 5);

Map h;

h.insert("a", 5);

h.insert("b", 4);

h.insert("c", 8);

h.insert("d", 2);

h.insert("e", 1);

Map j;

subtract(g, h, j); //checks if all are subtracted

assert(j.empty());