Eric Blasko

CSE 330 Data Structurs

Winter 2018

Lab 9 – Unordered Map

* **Status**

100% complete

* **Time Complexity**

The following methods are O(1) were there time is constant

* unsigned operator()(const Key & k) const
* unsigned operator()(const string & k) const
* unsigned operator()(const unsigned & k) const
* Unordered\_map(unsigned count = 11):
* Unordered\_map(Unordered\_map && rhs): num\_buckets
* Unordered\_map & operator=(const Unordered\_map & rhs)
* Unordered\_map & operator=(Unordered\_map && rhs)
* unsigned bucket\_count()
* unsigned size()
* unsigned bucket\_size(unsigned n)
* Hash\_iterator(Unordered\_map<Key, Value, Hash> \* m = nullptr): theMap{m}
* ~Hash\_iterator() = default;
* Hash\_iterator(const Hash\_iterator & iter):
* Hash\_iterator(Hash\_iterator && iter):
* Hash\_iterator & operator=(const Hash\_iterator & iter)
* Hash\_iterator & operator=(Hash\_iterator && iter)
* pair<Key, Value> & operator\*()
* typename list<pair<Key,Value>>::iterator operator->()
* bool operator==(const Hash\_iterator & rhs)
* bool operator!=(const Hash\_iterator & rhs)

The following methods are O(n) were n is the size of the bucket. Most times many methods could be O(1), however, collisions will cause methods such as find to iterate over the linked list inside the vector

* bool find(const Key & key, unsigned & index, typename list< pair< Key, Value > >::iterator &)
* Value & operator[](const Key & key)
* Value & operator[](Key && key)
* void insert(const pair<Key, Value> & r)
* void insert(pair<Key, Value> && r)
* void print()
* iterator find(const Key & key)
* void erase(const Key & key)
* void rehash(unsigned x)
* Unordered\_map(const Unordered\_map & rhs):
* ~Unordered\_map() = default
* iterator begin()
* iterator end()
* Hash\_iterator & operator++()
* Hash\_iterator operator++(int)
* **Source Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Unordered\_map.h

\* 03/14/2018

\* This class imitates the methods found in the unordered map found in STL. This class utilizes

\* hash functions which is used to index data within a vector of linked list. therefor there

\* are three classes within this code, the class for the hash function, the class for

\* unordered maps, and a class for hash iterator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef UNORDERED\_H

#define UNORDERED\_H

#include <iostream>

#include <vector>

#include <string>

#include <list>

using namespace std;

//function used to override the stream output

template<class Key, class Value>

ostream & operator<<(ostream & out, const pair<Key, Value> & in)

{

out << "(" << in.first << ":" << in.second << ")";

return out;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class Hasher

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename Key>

class Hasher

{

public:

unsigned operator()(const Key & k) const;

};

//Overloads () operator to take in string to be able to hash and return value

template <>

class Hasher<string>

{

public:

unsigned operator()(const string & k) const

{

unsigned hashVal = 0;

for(char ch: k)

hashVal = 37 \* hashVal + ch;

return hashVal;

}

};

//Overloads () operator to take in unsigned value to be used in hash and returns value

template <>

class Hasher<unsigned>

{

public:

unsigned operator()(const unsigned & k) const

{

return k%37;

}

};

//forward prototype

template <typename Key, typename Value, template <typename> class Hash> class Hash\_iterator;

//hash table of "num\_buckets" buckets. eash bucket is an stl list of pairs

template <typename Key, typename Value, template <typename> class Hash = Hasher>

class Unordered\_map

{

private:

unsigned num\_buckets;

vector<list < pair < Key, Value> > > bucket;

Hash<Key> hashFunc;

unsigned theSize;

//set bucket index based on key, if key is found in bucket[index] return true and set i

bool find(const Key & key, unsigned & index, typename list< pair< Key, Value > >::iterator & i)

{

index = hashFunc(key)%num\_buckets;

for(i = bucket[index].begin(); i != bucket[index].end(); i++)

if(i->first == key)

break;

if(i != bucket[index].end())

return true;

return false;

}

public:

typedef Hash\_iterator<Key, Value, Hash> iterator;

//default constructor

Unordered\_map(unsigned count = 11): num\_buckets(count), hashFunc(Hash<Key>()), theSize(0)

{

bucket = vector< list < pair < Key, Value > > >(num\_buckets);

}

//destructor. calls default destructor for vector and list

~Unordered\_map() = default;

//copy constructor

Unordered\_map(const Unordered\_map & rhs): num\_buckets{rhs.num\_buckets}, hashFunc{rhs.hashFunc}, theSize{0}

{

bucket = vector<list<pair<Key,Value>>>(num\_buckets);

typename list<pair<Key,Value>>::const\_iterator j;

for(int i = 0; i < rhs.num\_buckets; i++)

for(j = rhs.bucket[i].begin(); j != rhs.bucket[i].end(); j++)

insert(\*j);

}

//Overloaded assignment operator, move

Unordered\_map(Unordered\_map && rhs): num\_buckets{rhs.num\_buckets}, bucket{move(rhs.bucket)}, hashFunc{rhs.hashFunc}, theSize{rhs.theSize} {}

Unordered\_map & operator=(const Unordered\_map & rhs)

{

Unordered\_map<Key, Value, Hash> copy{rhs};

swap(\*this, copy);

return \*this;

}

//Overloaded assingment operator, copy

Unordered\_map & operator=(Unordered\_map && rhs)

{

swap(num\_buckets, rhs.num\_buckets);

swap(bucket, rhs.bucket);

swap(hashFunc, rhs.hashFunc);

swap(theSize, rhs.theSize);

return \*this;

}

//returns number of buckets in vector

unsigned bucket\_count()

{

return num\_buckets;

}

//returns overall size

unsigned size()

{

return theSize;

}

//returns size of bucket in linked list

unsigned bucket\_size(unsigned n)

{

if(n >= num\_buckets)

return 0;

return bucket[n].size();

}

//Overload [] operator to take in Key values, looks to see if value is already in buckets, if it

//is return its value, if not insert it to front of linked list. copy method

Value & operator[](const Key & key)

{

unsigned index;

typename list<pair<Key, Value>>::iterator i;

if(find(key,index,i))

return i->second;

bucket[index].push\_front(pair<Key, Value>(key, Value()));

theSize++;

return bucket[index].front().second;

}

//Overload [] operator to take in Key value, looks to see if value is already in buckets, if it

//is return its value, if not insert it to front of linked list. move mehtod

Value & operator[](Key && key)

{

unsigned index;

typename list<pair<Key, Value>>::iterator i;

if(find(key,index,i))

return i->second;

bucket[index].push\_front(pair<Key, Value>(move(key), Value()));

theSize++;

return bucket[index].front().second;

}

//looks to see if key is in buckets first and replaces value if found. if not, insert value into

//front of linked list at index. copy method

void insert(const pair<Key, Value> & r)

{

unsigned index;

typename list<pair<Key, Value>>::iterator i;

if(find(r.first, index, i))

\*i = r;

else

{

bucket[index].push\_front(r);

theSize++;

}

}

//looks to see if key is in buckets first and replaces value if found. if not, insert value into

//front of linked list at index. move method

void insert(pair<Key, Value> && r)

{

unsigned index;

typename list<pair<Key, Value>>::iterator i;

if(find(r.first, index, i))

\*i = move(r);

else

{

bucket[index].push\_front(move(r));

theSize++;

}

}

//prints all contents of each bucket

void print()

{

for(int i = 0; i < num\_buckets; i++)

{

cout << "bucket[" << i << "] size=" << bucket\_size(i) << " ";

typename list<pair<Key, Value>>::iterator j;

for(j = bucket[i].begin(); j!= bucket[i].end(); j++)

cout << \*j << " ";

cout << endl;

}

}

//searches for were data first occurs in vector and returns that pos. Else it returns the end

//of the vector

iterator begin()

{

iterator iter(this);

unsigned index = 0;

while(index < num\_buckets and bucket[index].size() == 0)

++index;

if(index < num\_buckets)

{

iter.bucket\_index = index;

iter.current = bucket[index].begin();

}

else

{

iter.bucket\_index = num\_buckets-1;

iter.current = bucket[num\_buckets-1].end();

}

return iter;

}

//returns the last pos of the vector and linked list.

iterator end()

{

iterator iter(this);

iter.bucket\_index = num\_buckets-1;

iter.current = bucket[num\_buckets-1].end();

return iter;

}

//searches for key in entire vector and linked list. if found set iterator and index,

//else return end of list

iterator find(const Key & key)

{

unsigned index;

iterator iter(this);

typename list<pair<Key,Value>>::iterator i;

if(find(key, index, i))

{

iter.bucket\_index = index;

iter.current = i;

}

else

{

iter.bucket\_index = num\_buckets-1;

iter.current = bucket[num\_buckets-1].end();

}

return iter;

}

//uses search to find if key is present, if it is find returns index and i to be able

//to delete data

void erase(const Key & key)

{

unsigned index;

typename list<pair<Key,Value>>::iterator i;

if(find(key,index,i))

{

bucket[index].erase(i);

theSize--;

}

}

//allocate new vector and rehash all previous data into new vector

void rehash(unsigned x)

{

Unordered\_map<Key,Value,Hash>copy(x);

iterator current, i = begin();

while(i != end())

{

current = i++;

copy.insert(\*current);

}

\*this = move(copy);

}

friend class Hash\_iterator<Key,Value,Hash>;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Hash\_iterator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename Key, typename Value, template <typename> class Hash>

class Hash\_iterator

{

Unordered\_map<Key, Value, Hash> \*theMap;

typename list< pair< Key, Value> >::iterator current;

unsigned bucket\_index;

public:

//default constructor

Hash\_iterator(Unordered\_map<Key, Value, Hash> \* m = nullptr): theMap{m} {}

//destructor

~Hash\_iterator() = default;

//copy constructor

Hash\_iterator(const Hash\_iterator & iter): theMap{iter.theMap}, current{iter.current}, bucket\_index{iter.bucket\_index} {}

//move constructor

Hash\_iterator(Hash\_iterator && iter): theMap{iter.theMap}, current{move(iter.current)}, bucket\_index{iter.bucket\_index} {iter.theMap = nullptr;}

//overload assingment operator. copy method

Hash\_iterator & operator=(const Hash\_iterator & iter)

{

theMap = iter.theMap;

current = iter.current;

bucket\_index = iter.bucket\_index;

return \*this;

}

//overload assingment operator. move method

Hash\_iterator & operator=(Hash\_iterator && iter)

{

swap(theMap, iter.theMap);

swap(current, iter.current);

swap(bucket\_index, iter.bucket\_index);

return \*this;

}

//pre increment

Hash\_iterator & operator++()

{

if(current == theMap->bucket[theMap->num\_buckets-1].end())

return \*this;

++current;

if(current == theMap->bucket[bucket\_index].end())

{

++bucket\_index;

while(bucket\_index < theMap->num\_buckets and theMap->bucket[bucket\_index].size() == 0)

++bucket\_index;

if(bucket\_index < theMap->num\_buckets)

current = theMap->bucket[bucket\_index].begin();

else

current = theMap->bucket[theMap->num\_buckets-1].end();

}

return \*this;

}

//post increment which call pre increment

Hash\_iterator operator++(int)

{

Hash\_iterator copy = \*this;

++(\*this);

return copy;

}

//overload \* operator to return current value

pair<Key, Value> & operator\*()

{

return \*current;

}

//Overload -> operator for current to return values at current

typename list<pair<Key,Value>>::iterator operator->()

{

return current;

}

//Overload == operator to return bool if values are equal

bool operator==(const Hash\_iterator & rhs)

{

return current == rhs.current;

}

//Overload != operator to return bool if values are not equal

bool operator!=(const Hash\_iterator & rhs)

{

return !(\*this == rhs);

}

friend class Unordered\_map<Key,Value,Hash>;

};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Unordered\_map\_test1.cpp

\* 03/14/2018

\* This program test the methods of class Unordered\_map.h. Main test will include

\* inserting data into the map, checking the size of the map, and printing its

\* contents

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Unordered\_map.h"

using namespace std;

//Main function to test class Unordered\_map.h

int main()

{

Unordered\_map<string, string> h(7);

cout << "h # count = " << h.bucket\_count() << endl;

cout << "h size = " << h.size() << endl;

cout << "Insert a few pairs\n";

h.insert(pair<string,string>("Red", "123 Main"));

h.insert(pair<string,string>("Brown", "45 First"));

h.insert(make\_pair("Blue", "20 2nd St"));

h["Jack"] = "500 E St";

h["Jill"] = "400 Lakewood";

h["Jake"] = "2 Sierra Madre";

cout << "h # count = " << h.bucket\_count() << endl;

cout << "h size = " << h.size() << endl;

cout << "Lookup a few values\n";

cout << "Brown:" << h["Brown"] << endl;

cout << "Green:" << h["Green"] << endl;

cout << "Jack:" << h["Jack"] << endl;

cout << "One move:" << h["One more"] << endl;

cout << "h # count = " << h.bucket\_count() << endl;

cout << "h size = " << h.size() << endl;

cout << "h Everything\n";

h.print();

cout << "Modify Red's value\n";

h["Red"] = "1234 Main St";

cout << "h Again\n";

h.print();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Unordered\_map\_test2.cpp

\* 03/14/2018

\* This program test the methods found in Unordered\_map.h. The main test for this program

\* will include inserting values into the map, searching for data using the a key, eraseing

\* values using there key, and iterating through the map

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Unordered\_map.h"

#include <string>

using namespace std;

//Main function to test class Unordered\_map.h

int main()

{

Unordered\_map<string,string> h(7);

h.insert(make\_pair("Red", "123 Main"));

h.insert(make\_pair("Brown", "45 First"));

h.insert(make\_pair("Blue", "20 2nd St"));

h["Jack"] = "500 E St";

h["jill"] = "400 Lakewood";

h["Jake"] = "2 Sierra Madre";

h["Green"] = "Mile";

h["One more"] = "Something";

Unordered\_map<string, string>::iterator i;

for(i = h.begin(); i != h.end(); i++)

cout << (\*i).first << ":" << (\*i).second << endl;

cout << "\nErase Jack\n";

h.erase("Jack");

cout << "\nFind Green\n";

i = h.find("Green");

if(i != h.end())

cout << \*i << endl;

else

cout << "Green not found\n";

cout << "\nFind NotThere\n";

i = h.find("NotThere");

if(i != h.end())

cout << \*i << endl;

else

cout << "NotThere not found\n\n";

for(i = h.begin(); i != h.end(); i++)

cout << i->first << ":" << i->second << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Unordered\_map\_test3.cpp

\* 03/14/2018

\* This program test the methods found in Unordered\_map.h. Test for this program

\* will include inserting, itterating, amd rehashing the map. Results will be

\* printed to console to verify its results

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Unordered\_map.h"

using namespace std;

//hash function

template <typename Key>

class Another

{

public:

unsigned operator()(const Key & k) const

{

return k;

}

};

//Main function to test class Unordered\_map.h

int main()

{

Unordered\_map<string, string, Hasher> h(7);

h.insert(make\_pair("Red", "123 Main"));

h.insert(make\_pair("Brown", "45 First"));

h.insert(make\_pair("Blue", "20 2nd St"));

h["Jack"] = "500 E St";

h["Jill"] = "400 Lakewood";

h["Jake"] = "2 Sierra Madre";

h["Green"] = "Mile";

h["One more"] = "Something";

h.print();

Unordered\_map<string, string>::iterator i;

for (i = h.begin(); i != h.end(); i++)

cout << "(" << i->first << ":" << i->second << ") ";

cout << endl;

h.rehash(11);

cout << "\nAfter rehash(11)\n\n";

h.print();

for (i = h.begin(); i != h.end(); i++)

cout << "(" << i->first << ":" << i->second << ") ";

cout << endl;

Unordered\_map<unsigned, string, Another> k(5);

k.insert(make\_pair(13, "cookie"));

k.insert(make\_pair(12, "monster"));

k.insert(make\_pair(17, "eat"));

k.insert(make\_pair(25, "something"));

k.insert(make\_pair(10, "else"));

cout << endl;

k.print();

Unordered\_map<unsigned, string, Another>::iterator j;

for (j = k.begin(); j != k.end(); j++)

cout << "(" << j->first << ":" << j->second << ") ";

cout << endl;

k.rehash(7);

cout << "\nAfter rehash(7)\n\n";

k.print();

for (j = k.begin(); j != k.end(); j++)

cout << "(" << j->first << ":" << j->second << ") ";

cout << endl;

}

* **Sample Runs**

**Test 1**

Script started on 2018-03-12 15:22:21-0700

]777;notify;Command completed;exit]0;005670557@csusb.edu@jb358-26:~/cse330/lab9]7;file://jb358-26.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab9[005670557@csusb.edu@jb358-26 lab9]$ g++ e-c [K[K[K[K-c Unordered\_map\_test1.cpp

]777;notify;Command completed;g++ -c Unordered\_map\_test1.cpp]0;005670557@csusb.edu@jb358-26:~/cse330/lab9]7;file://jb358-26.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab9[005670557@csusb.edu@jb358-26 lab9]$ g++ Unordered\_map\_test1.o

]777;notify;Command completed;g++ Unordered\_map\_test1.o]0;005670557@csusb.edu@jb358-26:~/cse330/lab9]7;file://jb358-26.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab9[005670557@csusb.edu@jb358-26 lab9]$ a[K.a/[K[K/a.op[Kut

h # count = 7

h size = 0

Insert a few pairs

h # count = 7

h size = 6

Lookup a few values

Brown:45 First

Green:

Jack:500 E St

One move:

h # count = 7

h size = 8

h Everything

bucket[0] size=3 (Jake:2 Sierra Madre) (Blue:20 2nd St) (Red:123 Main)

bucket[1] size=1 (One more:)

bucket[2] size=1 (Brown:45 First)

bucket[3] size=0

bucket[4] size=1 (Jack:500 E St)

bucket[5] size=0

bucket[6] size=2 (Green:) (Jill:400 Lakewood)

Modify Red's value

h Again

bucket[0] size=3 (Jake:2 Sierra Madre) (Blue:20 2nd St) (Red:1234 Main St)

bucket[1] size=1 (One more:)

bucket[2] size=1 (Brown:45 First)

bucket[3] size=0

bucket[4] size=1 (Jack:500 E St)

bucket[5] size=0

bucket[6] size=2 (Green:) (Jill:400 Lakewood)

]777;notify;Command completed;./a.out]0;005670557@csusb.edu@jb358-26:~/cse330/lab9]7;file://jb358-26.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab9[005670557@csusb.edu@jb358-26 lab9]$ exit

Script done on 2018-03-12 15:22:54-0700

**Test 2**

Script started on 2018-03-13 07:10:55-0700

]0;005670557@csusb.edu@jb358-0:~/cse330/lab9[005670557@csusb.edu@jb358-0 lab9]$ g++ -c Unordered\_map\_test2.cpp

]0;005670557@csusb.edu@jb358-0:~/cse330/lab9[005670557@csusb.edu@jb358-0 lab9]$ g++ Unore[Kdered [K\_map\_test2.o

]0;005670557@csusb.edu@jb358-0:~/cse330/lab9[005670557@csusb.edu@jb358-0 lab9]$ ./a.out

Jake:2 Sierra Madre

Blue:20 2nd St

Red:123 Main

One more:Something

Brown:45 First

jill:400 Lakewood

Jack:500 E St

Green:Mile

Erase Jack

Find Green

(Green:Mile)

Find NotThere

NotThere not found

Jake:2 Sierra Madre

Blue:20 2nd St

Red:123 Main

One more:Something

Brown:45 First

jill:400 Lakewood

Green:Mile

]0;005670557@csusb.edu@jb358-0:~/cse330/lab9[005670557@csusb.edu@jb358-0 lab9]$ exit

Script done on 2018-03-13 07:11:22-0700

**Test 3**

Script started on 2018-03-13 07:14:14-0700

]0;005670557@csusb.edu@csevnc:~/cse330/lab9[005670557@csusb.edu@csevnc lab9]$ g++ -c Unordered\_map)t[K[K\_test3.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab9[005670557@csusb.edu@csevnc lab9]$ g++ Unordered\_map\_test3.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab9[005670557@csusb.edu@csevnc lab9]$ ./a.out

bucket[0] size=3 (Jake:2 Sierra Madre) (Blue:20 2nd St) (Red:123 Main)

bucket[1] size=1 (One more:Something)

bucket[2] size=1 (Brown:45 First)

bucket[3] size=0

bucket[4] size=1 (Jack:500 E St)

bucket[5] size=0

bucket[6] size=2 (Green:Mile) (Jill:400 Lakewood)

(Jake:2 Sierra Madre) (Blue:20 2nd St) (Red:123 Main) (One more:Something) (Brown:45 First) (Jack:500 E St) (Green:Mile) (Jill:400 Lakewood)

After rehash(11)

bucket[0] size=1 (Brown:45 First)

bucket[1] size=1 (Red:123 Main)

bucket[2] size=0

bucket[3] size=1 (Green:Mile)

bucket[4] size=2 (Jill:400 Lakewood) (Jack:500 E St)

bucket[5] size=0

bucket[6] size=0

bucket[7] size=0

bucket[8] size=1 (Jake:2 Sierra Madre)

bucket[9] size=2 (One more:Something) (Blue:20 2nd St)

bucket[10] size=0

(Brown:45 First) (Red:123 Main) (Green:Mile) (Jill:400 Lakewood) (Jack:500 E St) (Jake:2 Sierra Madre) (One more:Something) (Blue:20 2nd St)

bucket[0] size=2 (10:else) (25:something)

bucket[1] size=0

bucket[2] size=2 (17:eat) (12:monster)

bucket[3] size=1 (13:cookie)

bucket[4] size=0

(10:else) (25:something) (17:eat) (12:monster) (13:cookie)

After rehash(7)

bucket[0] size=0

bucket[1] size=0

bucket[2] size=0

bucket[3] size=2 (17:eat) (10:else)

bucket[4] size=1 (25:something)

bucket[5] size=1 (12:monster)

bucket[6] size=1 (13:cookie)

(17:eat) (10:else) (25:something) (12:monster) (13:cookie)

]0;005670557@csusb.edu@csevnc:~/cse330/lab9[005670557@csusb.edu@csevnc lab9]$ exit

Script done on 2018-03-13 07:14:45-0700