Project 1 - Hashing
CSCI 230 T Th 11:10 am
Compiler: g++
OS: Linux

Michael Morikawa

March 30, 2020

Notes

Status

I have completed all parts for separate chaining, linear probring, and double hashing.

Extra Credit

The second option for extra credit is pretty much fully implemented however I was unable to take into account wrap around for open addressing.

Design Decisions

I was unable to fully make use of inheritance and have all three hash tables be derrived from the same class. This is because of the different structure for separate chainging and open addressing. The iterators for the two types are different and thus can't be inherited. I was able to derive the double hashing scheme from the base class which implemented linear probing.

I also did not throw an exception for when trying to delete an element that was not in the table. It simple counts probes and moves on. There is also nothing to stop someone from using too high of a load factor. I haven't tested it but I think it would go in an infinite loop for open addressing.

Data

Tests all done with p1large.txt

Hash Table Size(All Schemes)				
Load Factor	0.25	0.5	0.75	0.9
Table Size	12799	6397	4271	3557

Separate Chaining

Probe and Cluster Data				
Load Factor	0.25	0.5	0.75	0.9
Avg Probes	1.011	1.043	1.221	1.249
Max Probes	2	3	3	4
Total Clusters	35	126	659	659
Avg Cluster Size	2	2.048	2.036	2.102
Max Cluster Size	2	3	3	4

Linear Probing

Probe and Cluster Data				
Load Factor	0.25	0.5	0.75	0.9
Avg Probes	1.408	2.469	11.198	76.423
Max Probes	2	3	3	4
Total Clusters	74	78	53	10
Avg Cluster Size	5.770	23.346	47.604	298.8
Max Cluster Size	164	346	572	2887

Double Hashing

Probe and Cluster Data				
Load Factor	0.25	0.5	0.75	0.9
Avg Probes	1.024	1.120	2.073	3.138
Max Probes	5	9	15	185
Total Clusters	74	111	395	183
Avg Cluster Size	5.068	14.973	6.615	16.568
Max Cluster Size	124	193	154	423

Conclusions

From the data we see that separate chaining has the lowest average probe count, followed by double hashing with linear probing coming in last. This supports the assumption that it would perform the worst due to the clustering linear probing causes. For low load factors double hashing is similar to separate chaining for probes but as the load factor increases past 0.5 it becomes noticibly worse than separate chaining. The average and max clusters for linear probing is also the highest which is what is expected. The total clusters for separate chaining is higher than double hashing for a load factor greater than 0.5. However despite the high cluster count the cluster sizes remain small so the performace doesn't take as much of a hit as double hashing which has much larger cluster sizes.

Source Code

main.cpp

```cpp
#include "HashCode.hpp"
#include "ChainHashMap.hpp"
#include "OpenAddressMap.hpp"
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

struct RecordData
{
    int population;
    std::string name;
    RecordData(int n = 0, std::string s = "") : population(n), name(s) {}
    friend std::ostream &operator<<(std::ostream &os, const RecordData &data)
    {
        os << data.population << ' ' << data.name;
        return os;
    }
};

//returns number of the choice
int menu();

//Create vector of entries from a properly formatted input file
std::vector<Entry<int, RecordData>> processFile(std::string filePath);

int main()
{
    std::string filepath;
    std::vector<Entry<int, RecordData>> entryList;
    std::cout << "Please enter the file path for the input file you wish to use:\n";
    std::cin >> filepath;
    std::cout << filepath << std::endl; //!Alert: Delete later
    entryList = processFile(filepath);
    if (entryList.empty())
    {
        std::cout << "File not found or file is empty.";
        return -1;
    }

    int code, population;
    std::string name;
    int choice;
    double loadFactor;
    std::cout << "\n1. Chain Hashing\n"
              << "2. Linear Probe Hashing\n"
              << "3. Double Hashing\n"
              << "Please choose a hashing scheme: ";
    std::cin >> choice;
    std::cout << choice << std::endl; //!Alert: Delete later
    std::cout << "\nPlease enter a load factor between 0-1: ";
    std::cin >> loadFactor;
    std::cout << loadFactor << std::endl; //!Alert: Deletle later
    int operation = 0;
    if (choice == 1)
    {
```

```
    ChainHashMap<int, RecordData, HashCode> table(entryList, loadFactor);
    table.printClusterData();
    while (operation != 5)
    {
        operation = menu();
        switch (operation)
        {
        case 1:
        {
            std::cout << "Please enter the state code: ";
            std::cin >> code;
            std::cout << code << std::endl; //!ALERT: For file redirection delete later
            auto it = table.find(code);
            if (it == table.end())
            {
                std::cout << "Could not find the record\n";
            }
            else
            {
                std::cout << "Record Found\n";
                std::cout << (*it).value() << '\n';
            }
        }
        break;
        case 2:
        {
            std::cout << "Please enter the state code, population and name separated by spaces:\n";
            std::cin >> code >> population;
            std::getline(std::cin, name);
            std::cout << code << " " << population << " " << name << std::endl; //!Alert: Delete later
            table.put(code, RecordData(population, name));
        }
        break;
        case 3:
        {
            std::cout << "Please enter the state code: ";
            std::cin >> code;
            std::cout << code << std::endl; //!Alert: Delete later
            table.erase(code);
        }
        break;
        case 4:
        {
            table.printAll();
        }
        break;
        default:
            break;
        }
        if (operation != 4 && operation != 5)
        {
            std::cout << "Last operation took " << table.getLastProbe() << " probe(s).\n";
        }
    }
}
else
{
    OpenAddressMap<int, RecordData, HashCode> *table2;
    if (choice == 2)
    {
```

```
        table2 = new OpenAddressMap<int, RecordData, HashCode>;
    }
    else
    {
        table2 = new DoubleHashMap<int, RecordData, HashCode>;
    }
    table2->createFromEntryList(entryList, loadFactor);
    table2->printClusterData();
    while (operation != 5)
    {
        operation = menu();
        switch (operation)
        {
        case 1:
        {
            std::cout << "Please enter the state code: ";
            std::cin >> code;
            std::cout << code << std::endl; //!Alert: Delete later
            auto it = table2->find(code);
            if (it == table2->end())
            {
                std::cout << "Could not find the record\n";
            }
            else
            {
                std::cout << (*it).value() << '\n';
            }
        }
        break;
        case 2:
        {
            std::cout << "Please enter the state code, population and name separated by spaces:\n";
            std::cin >> code >> population;
            std::getline(std::cin, name);
            std::cout << code << " " << population << " " << name << std::endl; //!Alert: Delete later
            table2->put(code, RecordData(population, name));
        }
        break;
        case 3:
        {
            std::cout << "Please enter the state code: ";
            std::cin >> code;
            std::cout << code << std::endl; //!Alert: Delete later
            table2->erase(code);
        }
        break;
        case 4:
        {
            table2->printAll();
        }
        break;
        default:
            break;
        }
        if (operation != 4 && operation != 5)
        {
            std::cout << "Last operation took " << table2->getLastProbe() << " probe(s).\n";
        }
    }
}
```

```
}

int menu()
{
    std::cout << "\n1. Search for a record\n"
              << "2. Insert a record\n"
              << "3. Delete a record\n"
              << "4. List all records\n"
              << "5. Exit\n\n"
              << "Please Select an option: ";
    int choice;
    std::cin >> choice;
    std::cout << choice << std::endl; //!Alert delete later
    if (std::cin.fail())
    {
        std::cin.clear();
        std::cin.ignore();
        return 0;
    }
    return choice;
}

std::vector<Entry<int, RecordData>> processFile(std::string filePath)
{
    std::vector<Entry<int, RecordData>> entryList;
    Entry<int, RecordData> inputEntry;
    std::ifstream inputFile;
    inputFile.open(filePath);
    if (inputFile.fail())
    {
        return entryList;
    }
    int size;
    inputFile >> size;
    std::string stateCode, population, name;
    for (int i = 0; i < size; i++)
    {
        std::getline(inputFile, stateCode, ',');
        std::getline(inputFile, population, ',');
        std::getline(inputFile, name);
        inputEntry.setKey(std::stoi(stateCode));
        inputEntry.setValue(RecordData(std::stoi(population), name));
        entryList.push_back(inputEntry);
    }
    return entryList;
}
```

HashCode.hpp

```
#pragma once
class HashCode
{
public:
    int operator()(int key)
    {
        return key;
    }
};
```

Entry.hpp

```
#pragma once
template <typename K, typename V>
class Entry
{
public:
    Entry(const K& k = K(), const V& v = V())
        : _key(k), _value(v) {}
    const K& key() const
    {
        return _key;
    }
    const V& value() const
    {
        return _value;
    }
    void setKey(const K& k)
    {
        _key = k;
    }
    void setValue(const V& v)
    {
        _value = v;
    }

private:
    K _key;
    V _value;
};
```

ChainHashMap.hpp

```
#pragma once
#include "Entry.hpp"
#include <cmath>
#include <iostream>
#include <list>
#include <vector>

//Templated on key, value and hash function
template <typename K, typename V, typename H>
class ChainHashMap
{

public:
    class Iterator;

public:
    ChainHashMap(int capacity = 100)
        : n(0), table(capacity), probes(0) {}
    ChainHashMap(const std::vector<Entry<K, V>> &eList, float loadFactor);
    int size() const
    {
        return n;
    }
    bool empty() const
    {
        return n == 0;
    }
    Iterator find(const K &k);
```

```
    Iterator put(const K &k, const V &v);
    Iterator put(const Entry<K, V> &e);
    void erase(const K &k);
    void erase(const Iterator &p);
    void printAll();
    //returns pair of total clusters and average
    //cluster size
    void printClusterData();
    Iterator begin();

    //return probe count for the last operation
    //zero if no operation has been done yet
    int getLastProbe()
    {
        return probes;
    }
    Iterator end();

protected:
    typedef std::list<Entry<K, V>> Bucket;
    typedef std::vector<Bucket> BktArray;
    typedef typename BktArray::iterator BktIter;
    typedef typename Bucket::iterator ListIter;
    Iterator finder(const K &k);
    Iterator inserter(const Iterator &p, const Entry<K, V> &e);
    void eraser(const Iterator &p);
    static void nextEntry(Iterator &p)
    {
        ++p.ent;
    }
    static bool endOfBkt(Iterator &p)
    {
        return p.ent == p.bkt->end();
    }
    static bool isPrime(int n)
    {
        if (n < 2 || n % 2 == 0)
        {
            return false;
        }
        if (n == 2)
        {
            return true;
        }
        for (int div = 3; div <= sqrt(n); div += 2)
        {
            if (n % div == 0)
            {
                return false;
            }
        }
        return true;
    }

private:
    int n;
    H hash;
    BktArray table;
    int probes; //keeps track of probes during operations
```

```
public:
    class Iterator
    {
    private:
        BktIter bkt;
        ListIter ent;
        const BktArray *ba;

    public:
        Iterator(const BktArray &ba, const BktIter &bIt,
                 const ListIter &entIt = ListIter())
            : ba(&ba), bkt(bIt), ent(entIt) {}
        Entry<K, V> &operator*() const;
        bool operator==(const Iterator &p) const;
        Iterator &operator++();
        friend class ChainHashMap;
    };
};

//Iterator Class Definitions

template <typename K, typename V, typename H>
Entry<K, V> &ChainHashMap<K, V, H>::Iterator::operator*() const
{
    return *ent;
}

template <typename K, typename V, typename H>
bool ChainHashMap<K, V, H>::Iterator::operator==(const Iterator &p) const
{
    if (ba != p.ba || bkt != p.bkt)
    {
        return false;
    }
    else if (bkt == ba->end())
    {
        return true;
    }
    else
    {
        return (ent == p.ent);
    }
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator &ChainHashMap<K, V, H>::
    Iterator::operator++()
{
    ++ent;
    if (endOfBkt(*this))
    {
        ++bkt; //check next bucket
        while (bkt != ba->end() && bkt->empty())
        {
            ++bkt;
        }
        if (bkt == ba->end())
        {
            return *this;
        }
```

```
        ent = bkt->begin();
    }
    return *this;
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
    end()
{
    return Iterator(table, table.end());
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
    begin()
{
    if (empty())
    {
        return end();
    }
    BktIter bkt = table.begin();
    while (bkt->empty())
    {
        ++bkt;
    }
    return Iterator(table, bkt, bkt->begin());
}

template <typename K, typename V, typename H>
ChainHashMap<K, V, H>::ChainHashMap(const std::vector<Entry<K, V>> &eList,
                                    float loadFactor)
    : n(0), probes(0)
{

    int capacity = eList.size() / loadFactor;
    //finding size for table
    if (capacity % 2 == 0)
    {
        capacity++; //make it odd for easier prime checking
    }
    while (!isPrime(capacity))
    {
        capacity += 2;
    }
    table.resize(capacity);
    int probeSum = 0;
    int probeMax = 0;
    for (auto e : eList)
    {
        put(e);
        probeSum += probes;
        probeMax = std::max(probes, probeMax);
    }
    std::cout << "Table Size: " << table.size()
              << "\nAverage number of probes: " << (float)probeSum / eList.size()
              << "\nMax Probes: " << probeMax << std::endl;
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
```

```
    finder(const K &k)
{
    probes = 1; //set to 1 because the initial index counts as a probe
    int i = hash(k) % table.size();
    BktIter bkt = table.begin() + i;
    Iterator p(table, bkt, bkt->begin());
    while (!endOfBkt(p) && (*p).key() != k)
    {
        probes++;
        nextEntry(p);
    }
    return p;
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
    find(const K &k)
{
    Iterator p = finder(k);

    if (endOfBkt(p))
        return end();
    else
        return p;
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
    inserter(const Iterator &p, const Entry<K, V> &e)
{
    ListIter ins = p.bkt->insert(p.ent, e);
    n++;
    return Iterator(table, p.bkt, ins);
}

template <typename K, typename V, typename H> // insert/replace (v,k)
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::
    put(const K &k, const V &v)
{
    Iterator p = finder(k);

    if (endOfBkt(p))
    {
        return inserter(p, Entry<K, V>(k, v));
    }
    else
    {
        p.ent->setValue(v);
        return p;
    }
}

template <typename K, typename V, typename H>
typename ChainHashMap<K, V, H>::Iterator ChainHashMap<K, V, H>::

    put(const Entry<K, V> &e)
{

    return put(e.key(), e.value());
}
```

```
template <typename K, typename V, typename H>
void ChainHashMap<K, V, H>::erase(const Iterator &p)
{
    eraser(p);
}

template <typename K, typename V, typename H>
void ChainHashMap<K, V, H>::eraser(const Iterator &p)
{
    p.bkt->erase(p.ent);
    n--;
}

template <typename K, typename V, typename H>
void ChainHashMap<K, V, H>::erase(const K &k)
{
    Iterator p = finder(k);

    if (endOfBkt(p))
    {
        return;
    }
    eraser(p);
}

template <typename K, typename V, typename H>
void ChainHashMap<K, V, H>::printAll()
{
    Iterator p = begin();
    Iterator stop = end();
    while (!(p == stop))
    {
        std::cout << (*p).key() << ' ' << (*p).value() << '\n';
        ++p;
    }
}

template <typename K, typename V, typename H>
void ChainHashMap<K, V, H>::printClusterData()
{
    int clusters = 0;
    int elementsInClusters = 0;
    int maxSize = 0;
    for (auto it : table)
    {
        if (it.size() > 1)
        {
            clusters++;
            elementsInClusters += it.size();
            maxSize = std::max(maxSize, (int)it.size());
        }
    }
    float avgSize = (float)elementsInClusters / clusters;
    std::cout << "Clusters: " << clusters
              << "\nAverage Size: " << avgSize
              << "\nMax Size: " << maxSize << std::endl;
}
```

OpenAddressMap.hpp

```
#include "Entry.hpp"
#include <vector>

template <typename K, typename V>
class VisitEntry : public Entry<K, V>
{
public:
    VisitEntry()
        : Entry<K, V>(), empty(true), available(true) {}

    VisitEntry(const K &k, const V &v)
        : Entry<K, V>(k, v), empty(false), available(false) {}

    bool available;
    bool empty;
};

template <typename K, typename V, typename H>
class OpenAddressMap
{
public:
    class Iterator;

    //data
protected:
    std::vector<VisitEntry<K, V>> table;
    int n;
    int probes;
    H hash;

    //helper functions
protected:
    //skips over erased entries
    virtual Iterator finder(const K &k);

    //different function used to find where to insert
    //will return first open spot
    virtual Iterator insertionFinder(const K &k);
    virtual void eraser(const Iterator &p);

    //defined here because I had issues defining outside of class
    static bool isPrime(int n)
    {
        if (n < 2 || n % 2 == 0)
        {
            return false;
        }
        if (n == 2)
        {
            return true;
        }
        for (int div = 3; div <= sqrt(n); div += 2)
        {
            if (n % div == 0)
            {
                return false;
            }
        }
```

```
        return true;
    }

public:
    OpenAddressMap(int size = 11) : n(0), probes(0), table(size){};

    //Can't be a constructor since it calls a virutal function
    //Also outputs average probes and max probes for insertion from list
    // to console for project
    virtual void createFromEntryList(const std::vector<Entry<K, V>> &eList,
                                     float loadFactor);
    Iterator find(const K &k);
    Iterator put(const Entry<K, V> &e);
    Iterator put(const K &k, const V &v);
    void erase(const Iterator &p);
    void erase(const K &k);
    Iterator end();
    Iterator begin();
    void printAll();
    void printClusterData();

    int getLastProbe()
    {
        return probes;
    }
    bool empty() const
    {
        return n == 0;
    }
    int size()
    {
        return n;
    }

public:
    class Iterator
    {
    protected:
        typedef typename std::vector<VisitEntry<K, V>>::iterator vecItor;

    private:
        const std::vector<VisitEntry<K, V>> *tableRef;
        vecItor bktIt;

    public:
        Iterator(const std::vector<VisitEntry<K, V>> &table, const vecItor &it)
            : tableRef(&table), bktIt(it) {}
        Iterator &
        operator++();
        VisitEntry<K, V> &operator*() const;
        bool operator==(const Iterator &p) const;
        friend class OpenAddressMap;
    };
};

template <typename K, typename V, typename H>
VisitEntry<K, V> &OpenAddressMap<K, V, H>::
    Iterator::operator*() const
{
    return *bktIt;
```

```
}

template <typename K, typename V, typename H>
bool OpenAddressMap<K, V, H>::
    Iterator::operator==(const Iterator &p) const
{
    return (tableRef == p.tableRef && bktIt == p.bktIt);
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator &OpenAddressMap<K, V, H>::
    Iterator::operator++()
{
    ++bktIt;
    if (bktIt->available)
    {
        while (bktIt != tableRef->end() && bktIt->available)
        {
            ++bktIt;
        }
    }
    return *this;
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::
    createFromEntryList(const std::vector<Entry<K, V>> &eList, float loadFactor)
{
    int capacity = eList.size() / loadFactor;
    //finding size for table
    if (capacity % 2 == 0)
    {
        capacity++; //make it odd for easier prime checking
    }
    while (!isPrime(capacity))
    {
        capacity += 2;
    }
    table.resize(capacity);
    int probeSum = 0;
    int probeMax = 0;

    for (auto e : eList)
    {
        put(e);
        probeSum += probes;
        probeMax = std::max(probes, probeMax);
    }
    std::cout << "Table Size: " << table.size()
              << "\nAverage number of probes: " << (float)probeSum / eList.size()
              << "\nMax Probes: " << probeMax << std::endl;
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    end()
{
    return Iterator(table, table.end());
}
```

```
template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    begin()
{
    if (empty())
    {
        return end();
    }
    auto it = table.begin();
    while (it->empty)
    {
        ++it;
    }
    return Iterator(table, it);
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    finder(const K &k)
{
    probes = 1;
    int i = hash(k) % table.size();
    auto it = table.begin() + i;
    while (!it->empty && it->key() != k)
    {
        probes++;
        it++;
        if (it == table.end())
        {
            it = table.begin();
        }
    }
    return Iterator(table, it);
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    insertionFinder(const K &k)
{
    probes = 1;
    int i = hash(k) % table.size();
    auto it = table.begin() + i;
    while (!it->available && it->key() != k)
    {
        probes++;
        it++;
        if (it == table.end())
        {
            it = table.begin();
        }
    }
    return Iterator(table, it);
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    find(const K &k)
{
    Iterator p = finder(k);
```

```
    if ((*p).empty)
    {
        return end();
    }
    else
    {
        return p;
    }
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    put(const K &k, const V &v)
{
    n++;
    Iterator p = insertionFinder(k);
    (*p) = VisitEntry<K, V>(k, v);
    p.bktIt->empty = false;
    p.bktIt->available = false;
    return p;
}

template <typename K, typename V, typename H>
typename OpenAddressMap<K, V, H>::Iterator OpenAddressMap<K, V, H>::
    put(const Entry<K, V> &e)
{
    return put(e.key(), e.value());
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::eraser(const Iterator &p)
{
    //make it usable again
    (*p).available = true;

    n--;

    //reset entry to default
    (*p).setKey(K());
    (*p).setValue(V());
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::erase(const Iterator &p)
{
    eraser(p);
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::erase(const K &k)
{
    Iterator p = finder(k);
    eraser(p);
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::printAll()
{
    Iterator p = begin();
    Iterator stop = end();
```

```
    while (!(p == stop))
    {
        std::cout << (*p).key() << ' ' << (*p).value() << '\n';
        ++p;
    }
}

template <typename K, typename V, typename H>
void OpenAddressMap<K, V, H>::printClusterData()
{
    int clusters = 0;
    int size = 0;
    int sizeSum = 0;
    int maxSize = 0;
    for (auto ent : table)
    {

        if (ent.available)
        {
            if (size > 1)
            {
                sizeSum += size;
                maxSize = std::max(size, maxSize);
                clusters++;
            }
            size = 0;
        }
        else
        {
            size++;
        }
    }
    float avgSize = sizeSum / (float)clusters;
    std::cout << "Clusters: " << clusters
              << "\nAverage Size: " << avgSize
              << "\nMax Size: " << maxSize << std::endl;
}

//Implements double hashing for resolving collisions
template <typename K, typename V, typename H>
class DoubleHashMap : public OpenAddressMap<K, V, H>
{
public:
    void createFromEntryList(const std::vector<Entry<K, V>> &eList,
                             float loadFactor) override
    {
        int capacity = eList.size() / loadFactor;
        //finding size for table
        if (capacity % 2 == 0)
        {
            capacity++; //make it odd for easier prime checking
        }
        while (!this->isPrime(capacity))
        {
            capacity += 2;
        }
        this->table.resize(capacity);
        int probeSum = 0;
        int probeMax = 0;
```

```
        collisionPrime = ((this->table.size()) - 1);
        if (collisionPrime % 2 == 0)
        {
            collisionPrime--;
        }
        while (!this->isPrime(collisionPrime))
        {
            collisionPrime -= 2;
        }
        for (auto e : eList)
        {
            this->put(e);
            probeSum += this->probes;
            probeMax = std::max(this->probes, probeMax);
        }
        std::cout << "Table Size: " << this->table.size()
                  << "\nAverage number of probes: " << (float)probeSum / eList.size()
                  << "\nMax Probes: " << probeMax << std::endl;
    }

protected:
    typedef typename OpenAddressMap<K, V, H>::Iterator Iterator;
    Iterator finder(const K &k) override
    {
        this->probes = 1;
        int counter = 1;
        int index = this->hash(k) % this->table.size();
        auto currentEntry = this->table[index];
        while (!currentEntry.empty && currentEntry.key() != k)
        {
            this->probes++;
            index = (this->hash(k) + counter * (collisionPrime - (k % collisionPrime))) % this->table.size();
            currentEntry = this->table[index];
            counter++;
        }
        return Iterator(this->table, this->table.begin() + index);
    }

    Iterator insertionFinder(const K &k) override
    {
        this->probes = 1;
        int counter = 1;
        int index = this->hash(k) % this->table.size();
        auto currentEntry = this->table[index];
        while (!currentEntry.available && currentEntry.key() != k)
        {
            this->probes++;
            index = (this->hash(k) + counter * (collisionPrime - (k % collisionPrime))) % this->table.size();
            currentEntry = this->table[index];
            counter++;
        }
        return Iterator(this->table, this->table.begin() + index);
    }

private:
    int collisionPrime;
};
```

Output

Small Input File Tests

Separate Chaining

```
Please enter the file path for the input file you wish to use:
docs/p1small.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 1

Please enter a load factor between 0-1: 0.25
Table Size: 59
Average number of probes: 1
Max Probes: 1

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6055 225 "Napa, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 1784 "Riverside, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"
6075 2039 "San Francisco, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6037
Record Found
22851 "Los Angeles, CA"
Last operation took 1 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6000
Could not find the record
Last operation took 2 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6066 1  "New County, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6065 2000  "Riverside, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6999
Last operation took 2 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6075
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6055
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 2000  "Riverside, CA"
6066 1  "New County, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 5
```

Linear Probing

```
Please enter the file path for the input file you wish to use:
docs/p1small.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 2

Please enter a load factor between 0-1: 0.25
Table Size: 59
Average number of probes: 1
Max Probes: 1

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6055 225 "Napa, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 1784 "Riverside, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"
6075 2039 "San Francisco, CA"

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6037
22851 "Los Angeles, CA"
Last operation took 1 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6000
Could not find the record
Last operation took 3 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6066 1  "New County, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6065 2000  "Riverside, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6999
Last operation took 2 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6075
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6055
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 2000  "Riverside, CA"
6066 1  "New County, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 5
```

Double Hashing

```
Please enter the file path for the input file you wish to
docs/p1small.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 3

Please enter a load factor between 0-1: 0.25
Table Size: 59
Average number of probes: 1
Max Probes: 1

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6055 225 "Napa, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 1784 "Riverside, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"
6075 2039 "San Francisco, CA"

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6037
22851 "Los Angeles, CA"
Last operation took 1 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 1
Please enter the state code: 6000
Could not find the record
Last operation took 2 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6066 1  "New County, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 2
Please enter the state code, population and name separated by spaces:
6065 2000  "Riverside, CA"
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6999
Last operation took 2 probe(s).
```

```
1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6075
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 3
Please enter the state code: 6055
Last operation took 1 probe(s).

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 4
6019 1242 "Fresno, CA"
6083 721 "Santa Barbara, CA"
6037 22851 "Los Angeles, CA"
6097 655 "Sonoma, CA"
6047 341 "Merced, CA"
6111 1130 "Ventura, CA"
6059 6214 "Orange, CA"
6001 3648 "Alameda, CA"
6065 2000  "Riverside, CA"
6066 1  "New County, CA"
6067 1809 "Sacramento, CA"
6071 1920 "San Bernardino, CA"
6073 5351 "San Diego, CA"

1. Search for a record
2. Insert a record
3. Delete a record
4. List all records
5. Exit

Please Select an option: 5
```

Large File Probe Testing

Separate Chaining

Loadfactor = 0.25

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 1

Please enter a load factor between 0-1: 0.25
Table Size: 12799
Average number of probes: 1.01094
Max Probes: 2
Clusters: 35
Average Size: 2
Max Size: 2
```

Loadfactor = 0.50

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 1

Please enter a load factor between 0-1: 0.5
Table Size: 6397
Average number of probes: 1.04315
Max Probes: 3
Clusters: 126
Average Size: 2.04762
Max Size: 3
```

Loadfactor = 0.75

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 1

Please enter a load factor between 0-1: 0.75
Table Size: 4271
Average number of probes: 1.22108
Max Probes: 3
Clusters: 659
Average Size: 2.03642
Max Size: 3
```

Loadfactor = 0.90

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 1

Please enter a load factor between 0-1: 0.9
Table Size: 3557
Average number of probes: 1.24922
Max Probes: 4
Clusters: 659
Average Size: 2.10167
Max Size: 4
```

Linaer Probing

Loadfactor = 0.25

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 2

Please enter a load factor between 0-1: 0.25
Table Size: 12799
Average number of probes: 1.40838
Max Probes: 121
Clusters: 74
Average Size: 5.77027
Max Size: 164
```

Loadfactor = 0.50

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 2

Please enter a load factor between 0-1: 0.5
Table Size: 6397
Average number of probes: 2.46904
Max Probes: 119
Clusters: 78
Average Size: 23.3462
Max Size: 346
```

Loadfactor = 0.75

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 2

Please enter a load factor between 0-1: 0.75
Table Size: 4271
Average number of probes: 11.1979
Max Probes: 151
Clusters: 53
Average Size: 47.6038
Max Size: 572
```

Loadfactor = 0.90

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 2

Please enter a load factor between 0-1: 0.9
Table Size: 3557
Average number of probes: 76.4228
Max Probes: 1423
Clusters: 10
Average Size: 298.8
Max Size: 2887
```

Double Hashing

Loadfactor = 0.25

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 3

Please enter a load factor between 0-1: 0.25
12791
Table Size: 12799
Average number of probes: 1.02376
Max Probes: 5
Clusters: 74
Average Size: 5.06757
Max Size: 124
```

Loadfactor = 0.50

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 3

Please enter a load factor between 0-1: 0.5
6389
Table Size: 6397
Average number of probes: 1.12007
Max Probes: 9
Clusters: 111
Average Size: 14.973
Max Size: 193
```

Loadfactor = 0.75

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 3

Please enter a load factor between 0-1: 0.75
4261
Table Size: 4271
Average number of probes: 2.07348
Max Probes: 15
Clusters: 395
Average Size: 6.61519
Max Size: 154
```

Loadfactor = 0.90

```
Please enter the file path for the input file you wish to use:
docs/p1large.txt

1. Chain Hashing
2. Linear Probe Hashing
3. Double Hashing
Please choose a hashing scheme: 3

Please enter a load factor between 0-1: 0.9
3547
Table Size: 3557
Average number of probes: 3.1379
Max Probes: 185
Clusters: 183
Average Size: 16.5683
Max Size: 423
```