Sander VanWilligen

Zackery Lovisa

Com S 311 – Project 1 Report

buildDataStructure() Algorithm:

for our building data structures method, the algorithm looks like the pseudocode below:

```
table = new hashTable(list.size);
for (i = 0 to list.size){
    int key = floor(list[i]);
    tuple t = new tuple(key, lists[i]);
    int hashValue = hash(key);
    if(table[hash] == null) table[hash] = t;
    else{
        temp = contents[hash];
        while(temp.getNext()!=null){
            temp = temp.getNext();
        }
        Temp.setNext(t);
    }
    numElements++;
    if(loadFactor > .7){
        //re-factor p to be the first prime greater than 2p
        p = getFirstPrime(2 * p);
        //clone the contents of the table into a new table
        Tuple[] newTable = table.clone();
        h = new HashFunction(p);
        table= new Tuple[p];
        numElements=0;
        //for each index in newTable
        Tuple temp2;
        for (int i = 0 to newTable.length){
            Tuple temp = newTable[i];
            if(temp!= null){
                temp2 = new Tuple(temp.getKey(), temp.getValue());
                this.add(temp2);
                //iterate through the elements in the linked list in newTable
                //and re-add them to this hashTable
                while (temp.getNext() != null) {
                    temp = temp.getNext();
                    temp2 = new Tuple(temp.getKey(), temp.getValue());
                    this.add(temp2);
                }
            }
```

```
                }
        }
}
```

npHashNearestPoints() Algorithm:

input = floating point p;

```
int key = (int)Math.floor(p);
//all points nearby must be in the same bucket as the point, or in the two nearby buckets
//so search 3 buckets
ArrayList<Tuple> tempResults1 = null;
if(key>=1) tempResults1 = table.search(key-1);
ArrayList<Tuple> tempResults2 = table.search(key);
ArrayList<Tuple> tempResults3 = table.search(key+1);
ArrayList<Float> results = new ArrayList<Float>();
//now check results returned by search in all three buckets, and append any nearby points to the
//result list
if(key>=1){
        for(int i = 0; i < tempResults1.size(); i++){
                if (Math.abs(p-tempResults1.get(i).getValue()) <=1)
                        results.add(tempResults1.get(i).getValue());
        }
}
for(int i = 0; i < tempResults2.size(); i++){
        if (Math.abs(p-tempResults2.get(i).getValue()) <=1)
                results.add(tempResults2.get(i).getValue());
}
for(int i = 0; i < tempResults3.size(); i++){
        if (Math.abs(p-tempResults3.get(i).getValue()) <=1)
                results.add(tempResults3.get(i).getValue());
}
return results;


public ArrayList<Tuple> search(int k) {
        //hash the value of k to get the index of the hashtable
        int hash = h.hash(k);
        ArrayList<Tuple> result = new ArrayList<Tuple>();
        //if contents[hash] is empty, return an empty arraylists
        if (contents[hash] == null) return result;
        else {
                Tuple temp = contents[hash];
                //for each element in the linked list, add it to the arraylist
                while (temp != null) {
                        if (temp.getKey() == k) result.add(temp);
                        temp = temp.getNext();
                }
        }
        return result;
}
```

Runtime:

For the allNearestPointsNaive() method, our program runs in 42,618 milliseconds, or 42.618 seconds.

For the allNearestPointsHash() method, our program runs in 190 milliseconds, or .190 seconds.

In addition the buildDataStructure() algorithm runs in 40 milliseconds, or .040 seconds.