CS 271 Project 6

Tung Luu, Wilson Le

November 20, 2021

1 Hash function

Our hash function hashes string by first interpreting the all characters in the string using radix notation. Since the char type has 8-bits, which can represent 256 characters, I choose 257 as a radix, because 257 is a prime that is close to 256. Let's call the number of slots in our hash table m. With a string s, the formula to calculate the hash of s is:

$$h(s) = (\sum_{i=0}^{s.length()} s[i] * 257^i) \mod m$$

Here's the implementation of my program using the above hash function to hash all words in /usr/share/dict/words into 1000 slots:

```
#include <iostream>
#include <climits>
#include <fstream>
#include <vector>
#include <string>
using namespace std;

/*
A hash functions that map strings to integer in {0,1,...,numSlots - 1}
*/
int hashString(string s, int numSlots) {
    // The maximum value of numSlots should be INT_MAX
    numSlots = min(numSlots, INT_MAX);

    // Since the char type has 8-bits, which can represents 256 characters,
    // we will choose 257 as our base because 257 is a prime which is close to 256.
    int base = 257;
    int powOfBase = 1;
    int mod = numSlots;

    int hashRes = 0; // The hash result
```

```
    for(int i = 0; i < s.length(); i++) {
        unsigned char c = s[i];
        // Using the property (a+b) mod n=(a mod n + b mod n) mod n
        // and the property (ab) mod p = ( (a mod p) (b mod p) ) mod p,
        // we can use the following formulas to calculate the same result as
        // (hashing a string by expressing it as radix-257 and then mod by a number)
        hashRes = (hashRes + (c * powOfBase)) % mod;
        powOfBase = (powOfBase * base) % mod;
    }

    return hashRes;
}

int main() {
    vector<int> freq(1000, 0);
    ifstream dict("/usr/share/dict/words");
    string line;

    if(dict.is_open()) {
        while(getline(dict, line)) {
            int hashCode = hashString(line, 1000);
            freq[hashCode] += 1;
        }
        dict.close();
    }

    ofstream output("./data.csv");
    if(output.is_open()) {
        output << "slot,length" << "\n";
        for(int i = 0; i < 1000; i++) {
            output << i << "," << freq[i] << "\n";
        }
    }
    output.flush();
    output.close();
}
```

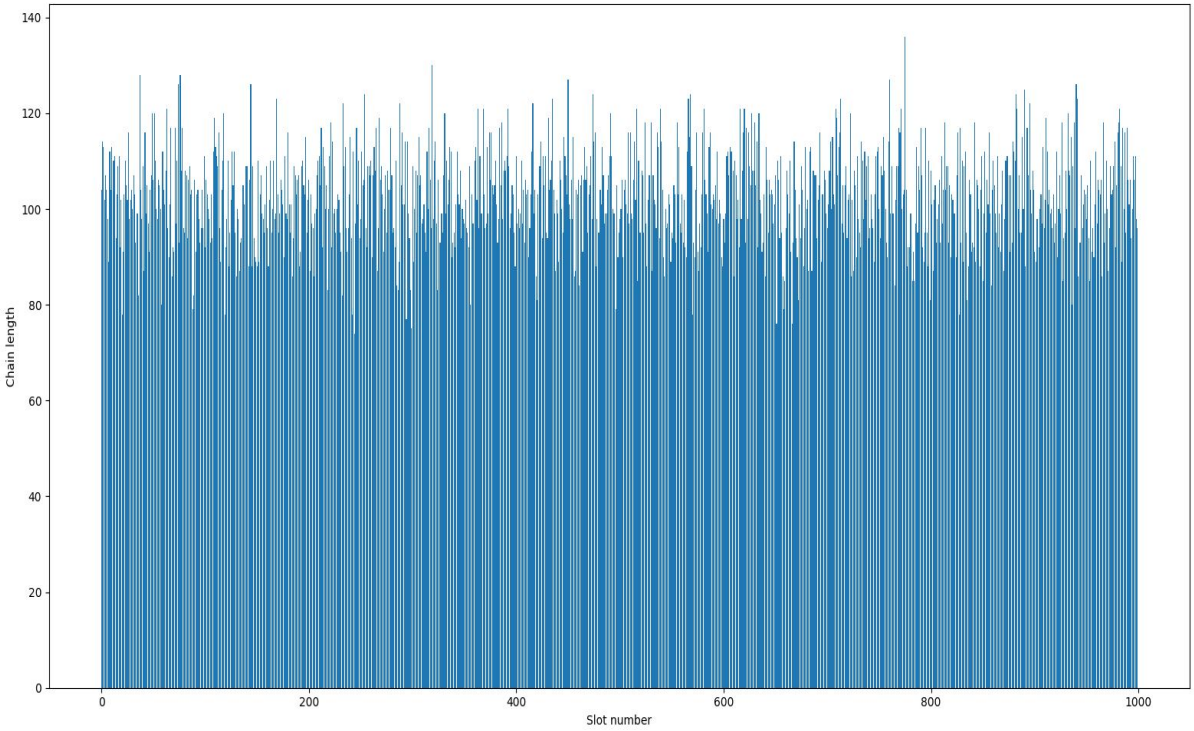
In the above implementation, we use some property of modulo operation to calculate the hash result of a string without getting integer overflow, achieving the same result as our original formula for hashing.

2 Analysis

We also plot a bar graph that displays how evenly our hash function assigns words to 1000 slots, in which the x-axis is slot number and y-axis is chain

length. We write the a short python snippet to create this graph and calculate the standard deviation:

```
import matplotlib.pyplot as plt
import pandas as pd
import math

data = pd.read_csv("./data2.csv")

def stdDev(data):
    n = len(data)
    mean = sum(data) / n
    total = 0
    for x in data:
        total += (x - mean) ** 2
    return math.sqrt(total / n)

def plot():
    plt.bar(data["slot"], data["length"])

    plt.xlabel("Slot number")
    plt.ylabel("Chain length")
    plt.show()

print(stdDev(data["length"]))
plot()
```



The standard deviation of our hash function when apply to the dictionary is 9.86286971423632.

3 Improvement

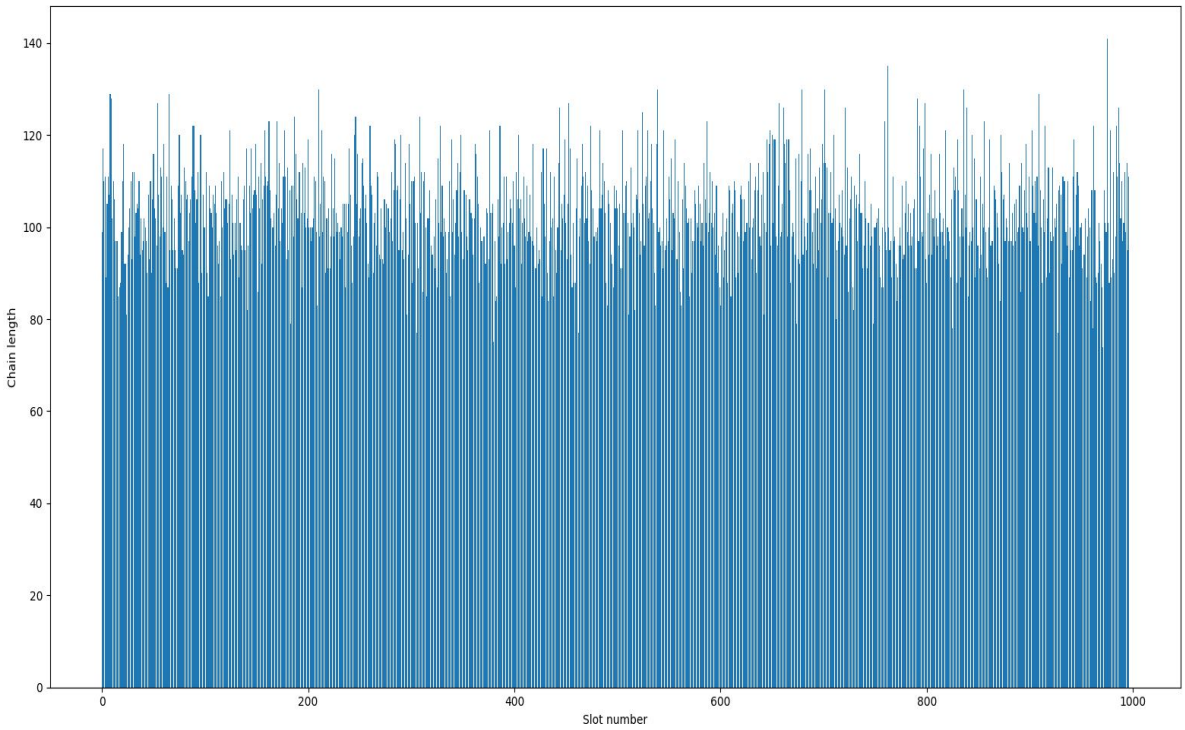
Our choice of 1000 as our number of slots might not be the best choice. I made a slightly different version of my hash function, where the number of slots is a maximum prime number that is smaller or equal to the numSlots parameter passed into the function. I think choosing a prime number as the number of slots can increase uniform of our hash function.

```
#include<iostream>
#include<climits>
#include<fstream>
#include<vector>
#include<string>
using namespace std;

/*
isPrime[i] = true if i is a prime number, false otherwise. This function
fills the isPrime vector from 0 to maxNum;
```

```
*/
void sieveOfEratosthenes(int maxNum, vector<int>& isPrime) {
    for(int num = 2; num * num <= maxNum; num++) {
        if(isPrime[num]) {
            for(int multiples = num * num; multiples <= maxNum; multiples += num) {
                isPrime[multiples] = false;
            }
        }
    }
}

/*
let's call m the biggest prime number less or equal than numSlots.
This hash functions that map strings to integer in {0,1,...,m-1}
*/
int hashString(string s, int numSlots, const vector<int>& isPrime) {
    // The maximum value of numSlots should be INT_MAX
    numSlots = min(numSlots, INT_MAX);

    int base = 257; // Since the char type has 8-bits, which can represents 256 characters,
    int powOfBase = 1;
    int mod;
    // mod is the largest prime number which is less than numSlots
    for(int num = numSlots - 1; num >=0; num--) {
        if(isPrime[num]) {
            mod = num;
            break;
        }
    }

    int hashRes = 0; // The hash result
    for(int i = 0; i < s.length(); i++) {
        unsigned char c = s[i];
        // Using the property (a+b) mod n=(a mod n + b mod n) mod n
        // and the property (ab) mod p = ( (a mod p) (b mod p) ) mod p,
        // we can use the following formulas to calculate the same result as (hashing a str
        hashRes = (hashRes + (c * powOfBase)) % mod;
        powOfBase = (powOfBase * base) % mod;
    }

    return hashRes;
}

int main() {
    vector<int> freq(1000, 0);
    ifstream dict("/usr/share/dict/words");
```

```
    string line;

    if(dict.is_open()) {
        vector<int> isPrime(1000, true);
        sieveOfEratosthenes(999, isPrime);

        while(getline(dict, line)) {
            int hashCode = hashString(line, 1000, isPrime);
            freq[hashCode] += 1;
        }
        dict.close();
    }

    ofstream output("./data2.csv");
    if(output.is_open()) {
        output << "slot,length" << "\n";
        for(int i = 0; i < 1000; i++) {
            output << i << "," << freq[i] << "\n";
        }
    }
    output.flush();
    output.close();
}
```

We also use the above python code to plot a bar graph and standard deviation that correspond to our new hash function.



The standard deviation of our new hash function when apply to the dictionary is 10.09869287254665

4 Part 4

1. Hash table time to create Dictionary from movie file: 0.201735 (s)
2. Binary search tree time to create Dictionary from movie file: 5.32387 (s)