Theory and Code Task 4

Russell Cannon, Ian Mooney, Patrick Murphy

May 5, 2025

Abstract

citations
https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/
https://www.geeksforgeeks.org/string-hashing-using-polynomial-rolling-hash-function/

1 Experimenting with occupancy ratios in linear probing

The occupancy ratio to be used in linear probing. This involves experimenting with different values such as 50%, 70%, and 80%, and reporting runtimes in nanoseconds.

2 Optimizing chain length in open hashing

At least three experiments should be conducted, and runtimes in nanoseconds should be reported.

3 Experimentation with different hash functions

3.1 Simple Hash

```cpp
static int hash(const std::string& word, int size) {
    long int hashValue = 0;

    for (char c : word) {
        hashValue += c;
    }

    return hashValue % size;
}
```

Results

Hash Type	Time(ms)	Lambda	Chain Length/ Cluster Size		
			Max	Min	Mean (non-zero)
Open	31	0.705688	33	0	5.74652
Linear	283	0.340332	5481	1	242.435

3.2 Rabin-Karp Hash

```cpp
static int hash(const std::string& word, int size) {
    int n = 0; // Hash value
    int d = 256; // number of characters in the input alphabet
    for (int i = 0; i < (int)word.size(); i++)
        n = (d * n + word[i]) % size;

    return abs(n);
}
```

Results

Hash Type	Time(ms)	Lambda	Chain Length/ Cluster Size		
			Max	Min	Mean (non-zero)
Open	40	0.705688	740	0	20.573
Linear	80	0.340332	4879	1	242.435

3.3 Polynomial Rolling Hash

Our second attempt: (Polynomial Rolling Hash from Geeks4Geeks)

```
static int hash(const std::string& word, int size) {
    int p = 31, m = 1e9 + 7, hashValue = 0, pPow = 1;

    for (char c : word) {
        hashValue = (hashValue + (c - '-' + 1) * pPow) % m;
        pPow = (pPow * p) % m;
    }

    return abs(hashValue % size);
}
```

Results

Hash Type	Time(ms)	Lambda	Chain Length/ Cluster Size		
			Max	Min	Mean (non-zero)
Open	46	0.705688	6	0	1.40044
Linear	24	0.340332	32	1	1.79491

3.4 Polynomial Rolling Hash With Reduced Alphabet

third attempt: (maps only the characters we allow)

```
static int charToIndex(const char c) {
    //-, a, b, ..., z, 0, 1, ... 9
    if (c == '-') return 1;
    if (c == '\'') return 2;
    if (std::isalpha(c)) return c - 'a' + 3;
    if (std::isdigit(c)) return c - '0' + 26 + 4;
    return 0;
}

static int hash(const std::string& word, int size) {
    int p = 31;
    long int m = 1e9 + 7, hashValue = 0, pPow = 1;

    for (char c : word) {
        hashValue = (hashValue + charToIndex(c) * pPow) % m;
        pPow = (pPow * p) % m;
    }

    return hashValue % size;
}
```

Results

Hash Type	Time(ms)	Lambda	Chain Length/ Cluster Size		
			Max	Min	Mean (non-zero)
Open	26	0.705688	5	0	1.389
Linear	21	0.340332	14	1	1.80804

4 Handling collisions in the table for linear probing

Handling collisions in the table for linear probing. The collision resolution method implemented must be described, with research and inclusion of a method described in the lecture.

4.1 Linear

```
int index = hash(pair.word, size);
while (!arr[index].empty && arr[index].word != pair.word) {
    index = (index + 1) % size;
}
```

4.2 Double Hash

```
int index = hash(pair.word, size);
while (!arr[index].empty && arr[index].word != pair.word) {
    index = (index + hash(index, size)) % size;
}
```

5 Rabin-Karp

Writing a function to prompt a user for a word, display the number of occurrences of this word in the text, and the locations of said occurrences in “The Adventure of the Engineer’s Thumb”.

6 80 Least and most frequent words

Implementing a function to output a list of the 80 least frequently occurring words in the text.

Implementing a function to output a list of the 80 most frequently occurring words in the text.