Quinn Roemer

CISP - 430

Assignment 10

4/19/2018

Part 0 - Hashing Implementation

Description:

The goal for this section of the assignment was to create a program that is able to hash an extremely long list of proteins in a hashing table of size 40. In the file, there are only 20 unique proteins that all need to be hashed to a location in the array. However collisions are going to happen. To resolve these collisions I used the technique of double hashing to create a unique step value for each key.

Hash Function

h(key) = (first_letter_of_key + (2 * last_letter_of_key)) % 40

Double Hash Function

h2(key) = h1key % 5

Source Code:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct arrayElement{
    string protein;
    int count = 0;
};

//Global Variables
char alphabet[26];
char lookUp[26];
arrayElement proteins[40];

//Function Prototypes
void readHash();
int createKey(string);
int secondKey(int);
void printProteins();
void search(string);

int main()
{
    char cTemp = 'A';
    string searchTerm;
```

```
    //Fills the alphabet array with the alphabet.
    for (int count = 0; count < 26; count++)
    {
        alphabet[count] = cTemp;
        cTemp++;
    }

    //Fills the lookUp array with the numbers 1-25.
    for (int count = 0; count < 26; count++)
    {
        lookUp[count] = count;
    }

    readHash();

    printProteins();

    //Searching loop.
    while (true)
    {
        cout << "Please enter a sequence: ";
        cin >> searchTerm;

        if (searchTerm == "done" || searchTerm == "DONE")
        {
                break;
        }

        search(searchTerm);
    }
}

//This function reads the list of data and hashes it.
void readHash()
{
    string line;
    int key = 0;

    ifstream myfile;
    myfile.open("proteins.txt", ios::in);

    getline(myfile, line);

    //Debug Line.
    //cout << "First line: " << line << endl;

    while (myfile.peek() != EOF)
    {
        getline(myfile, line);
        key = createKey(line);
```

```
        //If the place is empty this code executes.
        if (proteins[key].count == 0)
        {
                proteins[key].protein = line;
                proteins[key].count++;
        }

        //If the place has the same data this code executes.
        if (line == proteins[key].protein)
        {
                proteins[key].count++;
        }

        //If there is a collision this code executes.
        else
        {
                int doubleKey = secondKey(key);

                while (true)
                {
                        key = key + doubleKey;

                        //If the key is greater than 40 this code executes.
                        if (key >= 40)
                        {
                                key = key - 39;
                        }

                        //If the new location is empty this code executes.
                        if (proteins[key].count == 0)
                        {
                                proteins[key].protein = line;
                                proteins[key].count++;
                                break;
                        }

                        //If the new location is the same as the current protein this code
executes.
                        if (line == proteins[key].protein)
                        {
                                proteins[key].count++;
                                break;
                        }
                }
        }
    }

    //Debug Line.
    //cout << "Last line: " << line << endl;
```

```
}

//This function creates a key for a given string.
int createKey(string protein)
{
    int size = protein.length();
    char firstLetter = protein[0];
    char lastLetter = protein[size - 1];
    int key = 0;

    //Debug Code
    //cout << "First: " << firstLetter << endl;
    //cout << "Last: " << lastLetter << endl;

    //Looking up the value of the first letter.
    for (int count = 0; count < 26; count++)
    {
        if (firstLetter == alphabet[count])
        {
                key = lookUp[count];
                break;
        }
    }

    //Looking up the value of the second letter.
    for (int count = 0; count < 26; count++)
    {
        if (lastLetter == alphabet[count])
        {
                key = key + (2 * lookUp[count]);
                break;
        }
    }

    //Returning the key.
    return key % 40;
}

//This function creates the second key if a collision occurs.
int secondKey(int key)
{
    int iTemp = key % 5;

    //Prevents the return of an even number.
    if (iTemp % 2 == 0)
    {
        iTemp++;
    }
```

```
    //Returning the second key.
    return iTemp;
}

//This function prints the data.
void printProteins()
{
    int indent = 0;
    cout << "Protein\t\t\t\t\t\t\tCount" << endl;
    for (int count = 0; count < 40; count++)
    {
        //Making sure all lines print evenly.
        if (proteins[count].count != 0)
        {
                if (proteins[count].protein.length() <= 23)
                {
                        cout << proteins[count].protein << "\t\t\t\t\t" <<
proteins[count].count << endl;
                }
                else
                {
                        cout << proteins[count].protein << "\t\t\t\t" << proteins[count].count
<< endl;
                }
        }
    }

    cout << endl;
}

//This function searches for a certain piece of data.
void search(string searchTerm)
{
    //Uppercasing the search term.
    for (int count = 0; count < searchTerm.length(); count++)
    {
        searchTerm[count] = toupper(searchTerm[count]);
    }

    //Creating a key for the searchTerm.
    int key = createKey(searchTerm);

    //Looking for the data.
    if (proteins[key].protein == searchTerm)
    {
        cout << proteins[key].count << " FOUND" << endl;
    }

    //If not found this code executes.
    else
```

```
    {
        //Creating a second key to find data.
        int doubleKey = secondKey(key);

        while (true)
        {
                key = key + doubleKey;

                //If the key is greater than 40 this code executes.
                if (key >= 40)
                {
                        key = key - 39;
                }

                //If an empty location is found this code executes.
                if (proteins[key].count == 0)
                {
                        cout << "NOT FOUND" << endl;
                        break;
                }

                //If the searchTerm is found this code executes.
                if (proteins[key].protein == searchTerm)
                {
                        cout << proteins[key].count << " FOUND" <<  endl;
                        break;
                }
        }
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Protein                                          Count
BIKFPLVHANQHVDNSVRWGIKDW                         5930
AWGKKKTKTQFQFPTADANCDCDD                         7866
HAFSRCDTWWDCEGLANCALA                            14279
AECPMHSWTSTLVHANQHVDNMF                          7744
CMWIPTVRIKVAKVEGIFPWVFPDEGIF                     16336
LDCQWVRWVVHLDCQGIFC                              11980
RWGADANCDCDKKKTQFPTCRDDWA                        8713
KKTQFPTCRDDEGTWDCE                               8713
DANVRWGIKDWCDCDEGHI                              5930
MWIPTVRIFPMHGIFVRLFCDTWVF                        5930
PFPMHLMVHLDCQWSWWKDC                             5929
TMFSCCQLRDWALMDVECQWKD                           21660
WSWGFNFFNVRQVVQHAFSRCWC                          10528
CWSWGFNFFNKTCRDVRQVFSIKFK                        8712
KTQFADANGFGIFNFCDEGTWCDPKDK                      5325
SCCQLRDWALMDVECQMVH                              10044
VHANQHVDNSVRWKKKTQFPTCRDDG                       8713
CLANCALADANCDCDEGHIANMFSWFP                      8712
ECPMHSWTKKKTQFPTCRDDEGTVVQ                       10528
KKPTCRDVHLDCQANMGIFFFPDECQAN                     5324

Please enter a sequence: TMFSCCQLRDWALMDVECQWKD
21660 FOUND
Please enter a sequence: AECPMHSWTSTLVHANQHVDNMF
7744 FOUND
Please enter a sequence: testData
NOT FOUND
Please enter a sequence: done
Press any key to continue . . .
```

Part 1 - Perfect Hashing Implementation

Description:

In this section of the assignment my job was to create a perfect hash of a given set of data with 28 unique keywords. To do this, I decided to write a program that would try different look-up table values until it found a good combination. My program was able to generate unique keys for the given data set down to an array size of 30. This left only 2 spots free. After I generated the right numbers for the look-up table I hashed the data from the huge file provided by the professor in a program similar to the one before this one.

Hash Function

h(key) = (g[first_letter_of_key] + g[last_letter_of_key] + length_of_key) % 30

Note: The first source code and output are for the program that generated the correct numbers for the perfect hash. The second is the actual hashing program with the real data.

Source Code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>

using namespace std;

//Global Data
string keywords[28];
char alphabet[26];
int keys[28];
int lookUp[26] = { 0 };


//Function prototypes.
void genKey();
bool duplicates();


int main()
{
    cout << "Perfect Hash started (Mod 30)..." << endl;
    srand(time(NULL));
    string line;
    char cTemp = 'a';

    //Grabbing keywords from file.
```

```
    ifstream myfile;

    myfile.open("keyWordsUnique.txt", ios::in);

    for (int count = 0; count < 28; count++)
    {
        getline(myfile, line);
        keywords[count] = line;
    }

    //Intilizing the alphabet
    for (int count = 0; count < 26; count++)
    {
        alphabet[count] = cTemp;
        cTemp++;
    }

    //Trying random value until a solution is found.
    while (true)
    {
        for (int count = 0; count < 26; count++)
        {
                lookUp[count] = rand() % 30;

        }

        genKey();
        duplicates();
    }
}

//This algorithm generates the keys for an array of strings.
void genKey()
{
    string word;
    int size = 0;
    char firstLetter;
    char lastLetter;
    int placeFirst = 0;
    int placeLast = 0;

    for (int count = 0; count < 28; count++)
    {
        word = keywords[count];

        size = word.length();
        firstLetter = word[0];
        lastLetter = word[size - 1];

        placeFirst = 0;
```

```
        placeLast = 0;

        //This loops grabs the correct value from the look up table for the first letter.
        for (int index = 0; index < 26; index++)
        {
                if (firstLetter == alphabet[index])
                {
                        placeFirst = lookUp[index];
                        break;
                }
        }

        //This loops grabs the correct value from the look up table for the last letter.
        for (int index = 0; index < 26; index++)
        {
                if (lastLetter == alphabet[index])
                {
                        placeLast = lookUp[index];
                        break;
                }
        }

        //Storing the key in the keys array.
        keys[count] = ((placeFirst + placeLast + size) % 30);
    }
}

//This function looks for duplicates in the keys array.
bool duplicates()
{
    bool dupe = false;
    string answer;

    //Looking for duplicates.
    for (int count = 0; count < 28; count++)
    {
        for (int index = 0; index < 28; index++)
        {
                if (index == count)
                {
                        continue;
                }
                if (keys[index] == keys[count])
                {
                        dupe = true;
                        break;
                }
        }

        //If a duplicate is found this code executes.
```

```
        if (dupe == true)
        {
                break;
        }
    }

    //If no duplicates are found this code executes.
    if (dupe == false)
    {
        cout << "Solution found!" << endl;
        cout << "Look Up Table: " << endl;

        //Printing look up table.
        for (int count = 0; count < 26; count++)
        {
                cout << lookUp[count] << " ";

        }

        cout << endl << endl;
        cout << "Keys:" << endl;

        //Printing keys.
        for (int count = 0; count < 28; count++)
        {
                cout << keywords[count] << endl << keys[count] << endl;
        }

        cout << endl;

        //Prompting the user.
        cout << "Enter anything to find another solution." << endl;
        cin >> answer;
    }

    //Returning to caller.
    return dupe;
}
```

Output:

```
C:\Users\compu\Desktop\Mod 30\CISP 430 - Perfect Hasher.exe
Perfect Hash started (Mod 30)...
Solution found!
Look Up Table:
17 16 21 29 7 5 22 15 0 21 14 22 9 23 16 2 10 25 18 28 6 27 29 2 27 19

Keys:
auto
7
break
5
case
2
char
20
class
14
const
24
continue
6
do
17
double
12
float
8
for
3
friend
10
int
1
long
18
new
25
operator
19
private
16
public
29
short
21
signed
23
static
15
struct
22
switch
9
union
4
unsigned
13
virtual
26
void
0
while
11
```

Source Code:

Note: This code is for the actual hashing program that used the generated look-up table values to hash the real set of data.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct arrayElement {
    string keyword;
    int count = 0;
};

//Global Variables
char alphabet[26];
char lookUp[26] = { 17, 16, 21, 29, 7, 5, 22, 15, 0, 21, 14, 22, 9, 23, 16, 2, 10, 25, 18, 28,
6, 27, 29, 2, 27, 19 };
arrayElement keywords[30];

//Function Prototypes
void readHash();
int createKey(string);
void printKeywords();
void search(string);

int main()
{
    char cTemp = 'a';
    string searchTerm;

    //Fills the alphabet array with the alphabet.
    for (int count = 0; count < 26; count++)
    {
        alphabet[count] = cTemp;
        cTemp++;
    }

    readHash();

    printKeywords();

    //Searching loop.
    while (true)
    {
        cout << "Please enter a keyword: ";
```

```
        cin >> searchTerm;

        if (searchTerm == "done" || searchTerm == "DONE")
        {
                break;
        }

        search(searchTerm);
    }
}

//This function reads the list of data and hashes it.
void readHash()
{
    string line;
    int key = 0;

    ifstream myfile;
    myfile.open("keywords.txt", ios::in);

    getline(myfile, line);

    while (myfile.peek() != EOF)
    {
        getline(myfile, line);
        key = createKey(line);

        //If the place is empty this code executes.
        if (keywords[key].count == 0)
        {
                keywords[key].keyword = line;
                keywords[key].count++;
        }

        //If the place has the same data this code executes.
        else
        {
                keywords[key].count++;
        }
    }
}

//This function creates a key for a given string.
int createKey(string keyword)
{
    int size = keyword.length();
    char firstLetter = keyword[0];
    char lastLetter = keyword[size - 1];
    int key = 0;
```

```
    //Debug Code
    //cout << "First: " << firstLetter << endl;
    //cout << "Last: " << lastLetter << endl;

    //Looking up the value of the first letter.
    for (int count = 0; count < 26; count++)
    {
        if (firstLetter == alphabet[count])
        {
                key = lookUp[count];
                break;
        }
    }

    //Looking up the value of the second letter.
    for (int count = 0; count < 26; count++)
    {
        if (lastLetter == alphabet[count])
        {
                key = key + (lookUp[count]);
                break;
        }
    }

    //Returning the key.
    return ((key + size) % 30);
}

//This function prints the data.
void printKeywords()
{
    cout << "Keyword\t\t\t\tCount" << endl;

    //Printing data evenly.
    for (int count = 0; count < 30; count++)
    {
        if (keywords[count].count == 0)
        {
                continue;
        }

        if (keywords[count].keyword.length() >= 8)
        {
                cout << keywords[count].keyword << "\t\t\t" << keywords[count].count << endl;
                continue;
        }

        else
        {
                cout << keywords[count].keyword << "\t\t\t\t" << keywords[count].count << endl;
```

```
        }
    }

    cout << endl;
}

//This function searches for a certain piece of data.
void search(string searchTerm)
{
    //Lowercasing the search term.
    for (int count = 0; count < searchTerm.length(); count++)
    {
        searchTerm[count] = tolower(searchTerm[count]);
    }

    //Creating a key for the searchTerm.
    int key = createKey(searchTerm);

    //Looking for the data.
    if (keywords[key].keyword == searchTerm)
    {
        cout << keywords[key].count << " FOUND" << endl;
    }

    else
    {
        cout << "NOT FOUND" << endl;
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Keyword                         Count
void                            15360
int                             20736
case                            15360
for                             22272
union                           16128
break                           20736
continue                        19968
auto                            19200
float                           22272
switch                          21504
friend                          23040
while                           16128
double                          18432
unsigned                        16128
class                           23808
static                          28416
private                         17664
do                              20736
long                            20736
operator                        18432
char                            21504
short                           17664
struct                          49920
signed                          18432
const                           39936
new                             19967
virtual                         24576
public                          24576

Please enter a keyword: long
20736 FOUND
Please enter a keyword: continue
19968 FOUND
Please enter a keyword: testData
NOT FOUND
Please enter a keyword: done
Press any key to continue . . .
```

Conclusion

I had a lot of fun on this assignment! I especially enjoyed programming the code that would generate the values I needed in my look-up table to create a perfect hash. The code that I created to generate those values only took around 2-5 seconds to develop a solution when I had the possible open spaces in the array set to either 12 or 7. However, as soon as I set the possible open spaces to 2 and 0 the program took forever to complete. To get the data I used in this assignment I let the code run overnight to get a better solution. The code that was trying to generate the values to have 0 open spaces left never completed. Without a doubt, my code is not all that efficient. Yet, I don’t believe there is a much better way to accomplish the task then the way I tackled it. Looking forward to the next assignment!