Quinn Roemer

CISP - 430

Assignment 9

4/12/2018

Part 0 - Hashing Hand Execution - Chaining

Description:
The goal for this assignment was to create several detailed diagrams that outlined how a number of different hashing methods worked. The first method that I diagrammed used chaining to avoid collisions. In other words, the colliding node is linked onto the element where it should be placed if there is already data there.

Data to be inserted
Hash: (22, 3, 12, 1, 18, 36, 33, 9);

Hash Function
h(key) = key % 10;

Note: The following diagram will be placed on the next two pages.

Part 1 - Hashing Hand Execution - Linear

Description:

This method of hashing uses linear steps to avoid collisions. For example, if a collision is detected then it will try a different node by incrementing one. It will do this until it finds an available location for the data to be stored.

Data to be inserted

Hash: (22, 3, 12, 1, 18, 36, 33, 9);

Hash Function

h(key) = key % 10;

Note: The following diagram will be placed on the next two pages.

Part 2 - Hashing Hand Execution - Generalized Linear

Description:
This method of hashing uses generalized linear steps to avoid collisions. For example, if a collision is detected then it will try a different node by incrementing by some value. It will do this until it finds an available location for the data to be stored. The only difference between this method and regular linear hashing is the step value. In this case I will be stepping by a value of four.

<u>Data to be inserted</u>
Hash: (22, 3, 12, 1, 18, 36, 33, 9);

<u>Hash Function</u>
h(key) = key % 10;

Note: The following diagram will be placed on the next two pages.

Part 3 - Hashing Hand Execution - Double Hashing

Description:
This method of hashing uses a second key generated from the first key to determine the step value to avoid collisions. This method creates a more evenly dispersed hash that is usually much easier for the computer to work with.

<u>Data to be inserted</u>
Hash: (22, 3, 12, 1, 18, 36, 33, 9);

<u>Hash Functions</u>
h(key) = key % 10;

h2(key) = (key % 3) + 1;

Note: The following diagram will be placed on the next two pages.

Part 4 - Perfect Hashing

Description:

The goal for this section of the assignment was to create a perfect hash for a sequence of strings. We were to generate by hand a 26 element look-up table called g that would hash each element to a unique key. To do this, I created several tables in Excel that listed my keys and look-up values. In addition, I wrote a program that could recalculate the keys whenever I changed a look-up value.

<u>Data to be inserted</u>

Hash: (“auto”, “break”, “case”, “const”, “for”, “switch”, “struct”, “while”, “static”, “continue”);

<u>Hash Function</u>

h(key) = (g[first_letter_of_key] + g[last_letter_of_key] + length_of_key) % 10

Excel Tables before changing values:

Here is my look-up table and generated keys before I changed any values.

Look Up Table G		
Index	**Char**	**Value**
1	a	1
2	b	2
3	c	3
4	d	4
5	e	5
6	f	6
7	g	7
8	h	8
9	i	9
10	j	10
11	k	11
12	l	12
13	m	13

14	n	14
15	o	15
16	p	16
17	q	17
18	r	18
19	s	19
20	t	20
21	u	21
22	v	22
23	w	23
24	x	24
25	y	25
26	z	26

Key Table:

Data Table		
Index	**Key**	**h(key)**
0	auto	0
1	break	8
2	case	2
3	const	8
4	for	7
5	switch	3
6	struct	5
7	while	3
8	static	8
9	continue	6

Excel Tables after changing values:

Here are the steps taken to modify the table to produce a perfect hash. Also, posted are my look-up table after the changes and my final generated keys.

1. Change C to a value of 5.
2. Change K to a value of 12.
3. Change H to a value of 7.
4. Change A to a value of 6.
5. Change T to a value of 6.

Look Up Table G after changes		
Index	**Char**	**Value**
1	a	6
2	b	2
3	c	5
4	d	4
5	e	5
6	f	6
7	g	7
8	h	7
9	i	9
10	j	10
11	k	12
12	l	12
13	m	13
14	n	14
15	o	15
16	p	16
17	q	17
18	r	18
19	s	19
20	t	6
21	u	21
22	v	22
23	w	23

24	x	24
25	y	25
26	z	26

Keys after changes:

Data Table		
Index	**Key**	**h(key)**
0	auto	5
1	break	9
2	case	4
3	const	6
4	for	7
5	switch	2
6	struct	1
7	while	3
8	static	0
9	continue	8

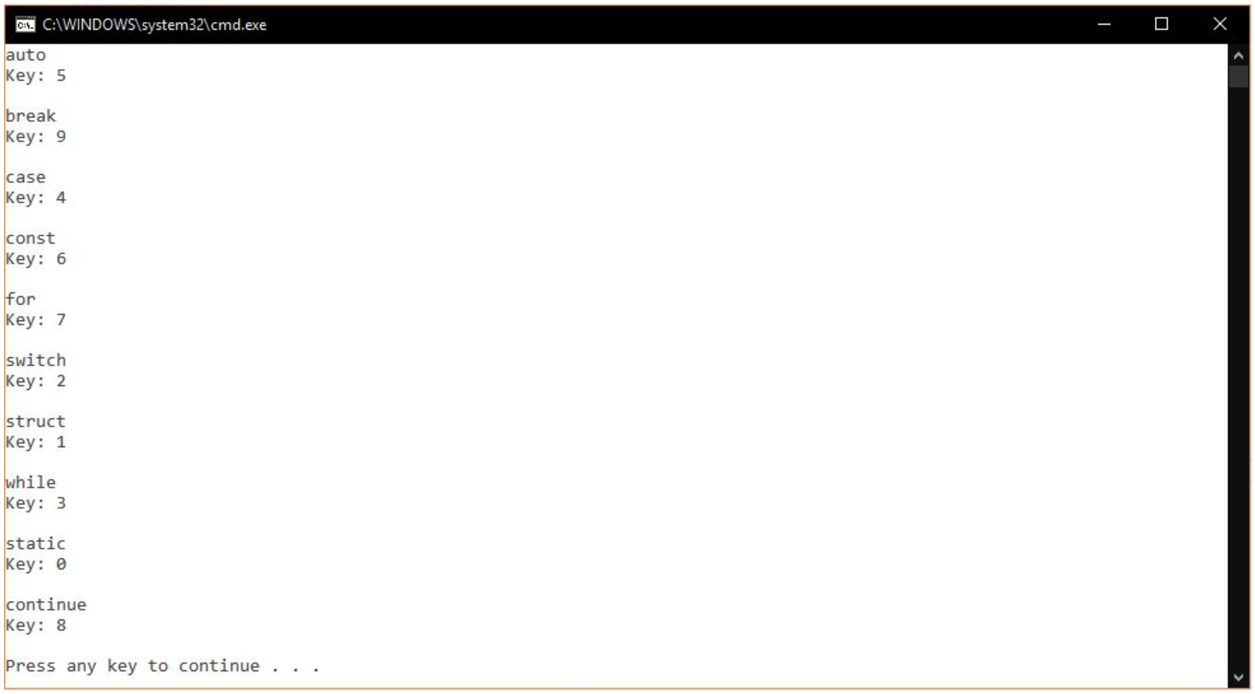
Code used to generate keys:

Here is the code that I wrote to help me create a perfect hash.

Source Code:

```
//Code written by Quinn Roemer 4/12/2018.
#include <iostream>
#include <string>

using namespace std;

//Global variables.
int value[26] = { 6, 2, 5, 4, 5, 6, 7, 7, 9, 10, 12, 12, 13, 14, 15, 16, 17, 18, 19, 6, 21,
22, 23, 24, 25, 26 };
char alphabet[26];

int keygen(string);

//Main function to execute.
int main()
{
    string words[10] = { "auto","break", "case", "const", "for", "switch", "struct", "while",
"static", "continue" };

    int keys[10];

    char test = 'a';

    //Loop used to insert a - z into the alphabet array.
    for (int count = 0; count < 26; count++)
    {
        alphabet[count] = test;
        test++;
    }

    //Loop used to call the keygen function for every data string.
    for (int count = 0; count < 10; count++)
    {
        keys[count] = keygen(words[count]);
    }

    //Loop used to output keys.
    for (int count = 0; count < 10; count++)
    {
        cout << words[count] << endl;
        cout << "Key: " << keys[count];
        cout << endl << endl;
    }
}

//This function looks at a string and creates the hashing key.
int keygen(string word)
{
    int size = word.length();
    char firstLetter = word[0];
    char lastLetter = word[size - 1];

    int placeFirst = 0;
    int placeLast = 0;

    //Debugging code.

    /*
    cout << word << endl;
    cout << "First letter: " << firstLetter << endl;
    cout << "Last letter: " << lastLetter << endl;
    cout << "Size: " << size << endl;

    cout << endl << endl;
    */
    //This loop grabs the correct value from the look up table for the first letter.
    for (int count = 0; count < 26; count++)
    {
        if (firstLetter == alphabet[count])
        {
                placeFirst = value[count];
        }
    }

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

    //Returning the key.
    return ((placeFirst + placeLast + size) % 10);
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
auto
Key: 5

break
Key: 9

case
Key: 4

const
Key: 6

for
Key: 7

switch
Key: 2

struct
Key: 1

while
Key: 3

static
Key: 0

continue
Key: 8

Press any key to continue . . .
```

Conclusion

This assignment was an awesome introduction to hashing. Being able to see and visualize several different hashing techniques was not only valuable but interesting. I especially liked it when I had to try and create the perfect hash. It started off almost like a guessing game but as I progressed I was able to see the patterns emerging and quickly solve the puzzle. Looking forward to the next assignment!