CPSC 335 Project 4: Hashing

Fall 2015

Prof. Doina Bein, CSU Fullerton

dbein@fullerton.edu

Introduction

In this project you will design, implement, and analyze one algorithm for the hashing problem. The algorithm is called Cuckoo Hashing, presented in the class.

For this problem, you will design one algorithm, describe the algorithm using clear pseudocode, implement the algorithm in C/C++/Java/Python, and test it on various inputs.

The Cuckoo Hashing Algorithm

There are several versions of cuckoo hashing. The version we learned in class is probably the simplest, where there are two hash functions, and thus only two places where any given item could be stored in the table. Let us consider the set of keys to be printable ASCII strings of length no more than 80. Let us consider the hash table size to be 17 .

Let us consider two hash functions, f_1 and f_2. If key is the string representing the key, then let keysize be the size of the string and key[i] be the ASCII code of the $(i+1)^{th}$ character in the string key read from left to right: key[0] is the leftmost character, key[1] is the second leftmost character, etc..

Function f_1 computes first a large number then it brings the result into the proper range using the formulas below:

$$val = \sum_{i=0}^{keysize-1} key[i] \cdot 37^{i}$$

$$f_1 = val \ \% \ tablesize$$

if $f_1 < 0$ then $f_1 = f_1 +$ tablesize

Function f_2 computes first a large number then it brings the result into the proper range using the formulas below:

$$val = \sum_{i=0}^{keysize-1} key[keysize - i - 1] \cdot 37^{i}$$

$f_2 = val \ \% \ tablesize$

if $f_2 < 0$ then $f_2 = f_2 +$ tablesize

a. Insert the strings (input file in4.txt)

Algorithm Engineering
California
State University
Fullerton
College of Engineering and Computer Science
Department of Computer Science
Dynamic Programming
Monge Properties
String Matching
Matrix Searching
Optimal Tree Construction
Online algorithms
emphasis on
Server Problem
Some related problem
Self-Stabilization
One of the greatest
mysteries in science
Quantum Nature of Universe
macroscopic quantum objects
Greatest mysteries
In physics and
astronomy
are known
to scientists
Dynamic decision making

into the hash table using f_1 for the first table and f_2 for the second table. Show the result of the insertion in the table below.

	Table T1	Table T2
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		
[10]		
[11]		
[12]		
[13]		
[14]		
[15]		
[16]		

Hint: consider a two-dimensional table of strings t, where t[0] is T1 and t[1] is T2. Consider a variable index that oscillates between 0 and 1as it would have oscillated between T1 and T2. In C++, the value of index could be changed using the tertiary operator: index = index? 0:1. Depending on the value of index, either apply hash function f_1 (index == 0) or f_2 (index == 1).

What to do

1. Write clear pseudocode for the algorithm.

2. Type these notes (electronically or on paper) and submit it as a PDF report.

3. Implement your algorithm in C/C++/Java/Python. You may use the templates provided at the end of this file.

4. Compile and execute the program.

5. Complete the table and insert it into the PDF report.

6. Create a file with the output of the program for an input value and submit it together with the program. Note, the output can be redirected to a file (for easy printing). For example, the following command line will create an output file in Linux-based operating system called a1out.txt by re-directing the output from the screen (display) to the file a1out.txt:

K:\cpscs335> a.out > a4out.txt

Sample outputs Cuckoo Hashing Algorithm:

Example #1:
K:\202> ast4
CPSC 335-x – Programming Assignment #4: Cuckoo Hashing algorithm

Input the file name (no spaces)!
in4.txt

String <Algorithm Engineering> will be placed at t[3][0]
String <California> will be placed at t[2][0]
String <State University> will be placed at t[13][0]
String <Fullerton> will be placed at t[16][0]
String <College of Engineering and Computer Science> will be placed at t[11][0]
String <Department of Computer Science> will be placed at t[10][0]
String <Dynamic Programming> will be placed at t[8][0]
String <Monge Properties> will be placed at t[0][0]
String <String Matching> will be placed at t[11][0] replacing <College of Engineering and Computer Science>

String <College of Engineering and Computer Science> will be placed at t[14][1]
String <Matrix Searching> will be placed at t[15][0]
String <Optimal Tree Construction> will be placed at t[8][0] replacing <Dynamic Programming>
String <Dynamic Programming> will be placed at t[13][1]
String <Online algorithms> will be placed at t[9][0]
String <emphasis on> will be placed at t[0][0] replacing <Monge Properties>
String <Monge Properties> will be placed at t[15][1]
String <Server Problem> will be placed at t[4][0]
String <Some related problem> will be placed at t[12][0]
String <Self-Stabilization> will be placed at t[7][0]
String <One of the greatest > will be placed at t[0][0] replacing <emphasis on>
String <emphasis on> will be placed at t[6][1]
String <mysteries in science> will be placed at t[9][0] replacing <Online algorithms>
String <Online algorithms> will be placed at t[1][1]
String <Quantum Nature of Universe> will be placed at t[4][0] replacing <Server Problem>
String <Server Problem> will be placed at t[1][1] replacing <Online algorithms>
String <Online algorithms> will be placed at t[9][0] replacing <mysteries in science>
String <mysteries in science> will be placed at t[5][1]
String <macroscopic quantum objects> will be placed at t[6][0]
String <Greatest mysteries> will be placed at t[4][0] replacing <Quantum Nature of Universe>
String <Quantum Nature of Universe> will be placed at t[7][1]
String <In physics and> will be placed at t[7][0] replacing <Self-Stabilization>
String <Self-Stabilization> will be placed at t[7][1] replacing <Quantum Nature of Universe>
String <Quantum Nature of Universe> will be placed at t[4][0] replacing <Greatest mysteries>
String <Greatest mysteries> will be placed at t[3][1]
String <astronomy > will be placed at t[7][0] replacing <In physics and>
String <In physics and> will be placed at t[7][1] replacing <Self-Stabilization>
String <Self-Stabilization> will be placed at t[7][0] replacing <astronomy >
String <astronomy > will be placed at t[13][1] replacing <Dynamic Programming>
String <Dynamic Programming> will be placed at t[8][0] replacing <Optimal Tree Construction>
String <Optimal Tree Construction> will be placed at t[8][1]
String <are known> will be placed at t[13][0] replacing <State University>
String <State University> will be placed at t[11][1]
String <to scientists> will be placed at t[8][0] replacing <Dynamic Programming>
String <Dynamic Programming> will be placed at t[13][1] replacing <astronomy >
String <astronomy > will be placed at t[7][0] replacing <Self-Stabilization>
String <Self-Stabilization> will be placed at t[7][1] replacing <In physics and>
String <In physics and> will be placed at t[7][0] replacing <astronomy >
String <astronomy > will be placed at t[13][1] replacing <Dynamic Programming>
String <Dynamic Programming> will be placed at t[8][0] replacing <to scientists>
String <to scientists> will be placed at t[16][1]
String <Dynamic decision making> will be placed at t[1][0]

Template for a C/C++ program doing Cuckoo Hashing algorithm:

```
// Assignment 4: Cuckoo Hashing algorithm
// XX YY ( YOU NEED TO COMPLETE YOUR NAME )
// An open addressing method called Cuckoo Hashing
// INPUT: an input file containing strings of characters, one string per line
// OUTPUT: a detailed list of where the strings are inserted.

#include <iostream>
#include <cstring>
#include <stdio.h>

using namespace std;

// cuckoo tables' size
const int tablesize = 17;
// combine the two 1-dimensional table into one 2-dimensional table
char  t[tablesize][2][255];

// compute the hash functions
size_t f(char*, size_t);

// place a string in one of the hash tables
bool place_in_hash_tables (char*);

int main() {

  // the strings to be stored in the hash tables
  char s[255]="";
  char null_st[] ="";
  size_t i, len;
  bool placed;

  // clear the tables
  for(i=0; i< tablesize; i++) {
    strcpy(t[i][0], null_st);
    strcpy(t[i][1], null_st);
  }

  char filename[255] = "";

   // display the header
  cout << endl << "CPSC 335-x - Programming Assignment #4: ";
  cout << "Cuckoo Hashing algorithm" << endl;

  // read the strings from a file
  cout << "Input the file name (no spaces)!" << endl;
  cin >> filename;
```

```
  // open the file for reading
  FILE *file = fopen ( filename, "r" );
  if ( file != NULL )
    {
     /* read line by line from the file */
     while ( fgets ( s, 255, file ) != NULL ) {
           // place null character at the end of the line instead of <return>
           len = strlen(s);
           s[len-2]='\0';
           // insert the string in the cuckoo table
           placed = place_in_hash_tables(s);
           // check whether the placement was successful
           if (!placed) {
             cout << "Placement has failed" << endl;
             return -1;
           }
     }
     fclose ( file );
    }
  else
    {
     perror ( filename ); /* why didn't the file open? */
    }
  return 0;

}


bool place_in_hash_tables (char *s) {

  bool placed;
  size_t pos;
  int index;
  char temp_s[255], temp[255];

  strcpy(temp_s, s);

  // use a counter to detect loops
  int counter = 0;

  // start with table T1
  index = 0;

  placed = false;

  pos = f(temp_s, index);
```

```
  while((!placed ) && (counter < 2*tablesize)) {

    if (strcmp(t[pos][index], "") == 0 ) {
      // the entry at index <pos> in the <index> hash table is available so store the string <temp_s> there
      cout << "String <" << temp_s << "> will be placed at";
      cout  << " t[" << pos <<"][" << index << "]" << endl;
      strcpy(t[pos][index], temp_s);
      placed = true;
      return placed;
    }
    else {
      // the entry at index <pos> in the <index> hash table is not available so
      // obtain the string stored over there in variable <temp> and store the string <temp_s> there
      // now the string <temp> needs to be placed in the other table
      cout << "String <" << temp_s << "> will be placed at" << " t[" << pos;
      cout <<"][" << index << "]" << " replacing <" << t[pos][index] << ">";
      cout << endl;
      // YOU NEED TO WRITE THE CODE TO STORE IN temp THE STRING STORED AT
      // t[pos][index] AND STORE IN t[pos][index] THE STRING temp_s
      strcpy(temp_s, temp);
      // NOW temp_s CONTAINING THE EVICTED STRING NEEDS TO BE STORED
      // IN THE OTHER TABLE
      // WRITE THE CODE TO SET index TO INDICATE THE OTHER TABLE
      // WRITE THE CODE TO CALCULATE IN pos THE HASH VALUE FOR temp_s
      counter ++;
    }
  }
  return placed;
};


// compute the hash functions
size_t f(char *s, size_t index) {
// s is the string (the key) to which we apply the hash function
// index indicates which hash function will be used
// index == 0 means the first hash function
// index == 1 means the second hash function
  size_t po, len;
  int i, val, temp;
  po = 1;

  len = strlen(s);

  if (index == 0) {

    val = s[0];
    val = val % tablesize;
    if (val < 0) val += tablesize;
```

```
    if (len == 1)
      return val;

    for (i = 1; i < len; i++)
    {
      temp = s[i];
      po *= 37;

      po = po % tablesize;
      if (po < 0) po += tablesize;

      val += temp * po;
      val = val % tablesize;

      if (val < 0) val += tablesize;
    }
    return val;
}
  else {
    // YOU NEED TO IMPLEMENT THE STEPS TO CALCULATE THE SECOND
    // HASH FUNCTION
    val = val % tablesize;
    return val;
 }
}
```