Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Fall '2013-2014

Programming Assignment 4

Due date: 25 January 2014, Saturday, 23:55

1 Objectives

In this assignment you are going to practice on hash table implementation and design a simple basic search engine to get some statistical information for the words in a document using C++.

Keywords: *Hash table, search engine, inverted index*

2 Problem Definition

Assume that you are a worker in a technology department of a press company where books, journals and daily newspapers are published. Some companies which work on statistical analysis make a demand for statistical information for the words in a document from your company. Your boss wants you to generate a basic search engine to get such data.

You are asked to get positions of all the occurrences of a given word in a document. A position will be indexed by the line number and the word number in that line. This structure is called *inverted index* (see `http://en.wikipedia.org/wiki/Inverted_index` for examples).

For this task, you are expected to design a hash table structure that stores inverted index values for each word in a given document. Your hash implementation must be fast and accurate. More details are given below.

- You will store all occurrences of each distinct word in the hash table as a single entry.
- The position information of a word is the line number (starting from 1) and the word number (starting from 1) in that line (E.g. 1^{th} word, 2^{nd} word and so on).
- The words are case-insensitive: all words must be converted to lowercase before hashing.
- You will use two open addressing collision resolution strategy (linear probing and quadratic probing) with two different hash functions. Hence, there will be four different hash tables.
- For the first hash function, use $((\sum_{n=0}^{keySize-1} key[keySize-n-1]*24^n) \bmod tableSize)$ and for the second hash function use $((\sum_{n=0}^{keySize-1} key[keySize-n-1]*37^n) \bmod tableSize)$. In the case of overflow add the value of table size to the result of the hash function.

- You will read the document from a *.txt file. Each word is separated with single white space character. You will ignore all special characters such as: **. , : ; ? ! ’ “ ” ( )**. Hence you can assume that a word is composed of only alpha-numeric characters. The file name is given as the first command line argument.

3 Tasks

To compare different hash table implementations, you are going to calculate the average number of probes and the average access time for a successful search of all the distinct words in the given document. Also you are expected to calculate the probe count and the access time for a given word. To implement the hash tables, please carefully examine the HashTable.h file. Some definitions of this file are described below.

- The **enum hashTableType { LINEAR, QUADRATIC}** is an enumeration for the type of collision resolution strategy. LINEAR is for linear probing, QUADRATIC is for quadratic probing.
- The **enum hashFunctionType { FUNC1, FUNC2 }** is an enumeration for the type of hash functions. FUNC1 is for the first hash function and FUNC2 is for the second hash function.
- The **struct wordData{int lineNumber; int wordNumber;}** data member contains the position information for a word in the document. lineNumber represents the line number of the word in the given document and wordNumber represents the word number of the word in that line.
- The **struct wordInText{string name; vector<wordData> data;}** data member contains the position information of a distinct word along with its all occurrences in the given document. name represents the word itself and vector<wordData> data represents position information of all the occurrences of the word.
- Construct a hash table class by implementing the methods in the interface given in HashTable.h. Note that you are not allowed to make any changes in the interface. The methods and data members are explained below:
  - **vector<wordInText> hashTable:** This is the array that will be used for implementing the hash table structures with given inputs in constructor. The entries of the hashTable array is the wordInText struct for each distinct word in the document. In order to distribute the keys evenly among the cells and in order to guarantee the insertion in quadratic probing, take the table size as the smallest prime number which is greater than two times the number of distinct words in the given document.
  - **hashTableType tableType:** This is the variable that keeps the type of hash table that you use.
  - **hashFunctionType hashFuncType:** This is the variable that keeps the type of hash function that you use.
  - **HashTable constructor:** You will create the hash table with the given inputs. The parameters of the constructor are:
    - **hashTableType tType:** This variable stores the type of hash table that you create.
    - **hashFunctionType hType:** This variable stores the type of hash function that you use in the hash table.
    - **const vector<wordInText>& words :** This variable represents the each distinct word along with its position information of all occurences that must be populated before calling the constructor (after parsing the input file).

- **void getWord method:** Positions of all occurrences of the given word that is stored in the hash table is returned using reference parameters. The probing count and the access time of the given word are also returned using reference parameters. The time should be in terms of seconds. The code of the time calculation will be given later. The parameters of the method are:
  - **const string &name:** This variable is the word name that will be searched.
  - **vector<wordData> &wordResult:** Positions of all occurrences of the given word is returned in this variable.
  - **int &probingCount:** The probe count is returned in this variable.
  - **double &accessTime:** The access time to the given word is returned in this variable.
- **void getHashTablePerformance method:** You will try to access all the words in the hash table for getting the performance of the hash table. As a measurement of the performance, you will return the average probing count and average access time using reference parameters. The time should be in terms of seconds. The code of the time calculation will be given later. The parameters of the method are:
  - **double &averageProbingCount:** The average probe count is returned in this variable.
  - **double &averageAccessTime:** The average access time is returned in this variable.
- **int hashFunction method:** This method returns the hash table index of the given word. For the key value, you will use the ASCII values of the each letter in the word. Remember that all the letters of the word should be lower case. The parameters of the method are:
  - **const string &key:** This variable represents the key for the hash function. In our problem, it is the word name.

4 Regulations

1. **Programming Language:** You should code your program in C++ language. Your submission will be compiled on department's inek machines. You are expected to make sure that your code compiles with g++ correctly.

2. **Late Submission:** You have a total of 7 days for late submission. You can spend this credit for any of the assignments or distribute it for all. If total of your late submissions exceed the limit, a penalty of 100 - $5 \times day^2$ will be applied.

3. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.

4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis. Please ask your questions related to the homework on COW instead of sending email in order to your friends, who may face with same problem, can see your questions and answers.

5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

6. **Not Graded:** Below are some situations that your homework will not be graded (automatically assigned zero):

   (a) Implementing the program by making any change in the interface file HashTable.h.

   (b) Sending the homework files with different names. They should be hw4.cpp and main.cpp.

5 Submission

Submission will be done via COW. You will be given HashTable.h file. You will do hw4.cpp which contains the implementation of the class that is given HashTable.h. Create a tar.gz file named `hw4.tar.gz` that contains hw4.cpp and main.cpp files. We will share an example main in the following days. Use sufficient comment lines in your algorithm in order to explain your solution clearly.

Note: The submitted archive should not contain any directories! The following command sequence is expected to run your program on a Linux system:

```
$ tar -xf hw4.tar.gz
$ g++ main.cpp -o hw4
$ ./hw4 document.txt
```