CS 667 : Homework 2(Due: Feb 28, 2012)

Problems 1-6 are for 200pts. You may replace some of them with Problem 7 or 8 for a total of 200.

Problem 1. (20 POINTS)
Consider a hash table of size $m = 1000$ and a corresponding hash function $h(k) = \lceil m(kA \bmod 1) \rceil$, for $A = (\sqrt{5}-1)/2$. Compute the locations to which the keys 61,62,63,64 are mapped.

Problem 2. (20 POINTS)
You are given three sorted sequences A, B, C of $\lg n$, $\sqrt{n}$, and n keys respectively. A, B and C are given in the form of arrays. Think of A containing documents that contain term A and so on. The output for the problems listed below should be in the same form; it does not need to be sorted however.

(i) We want to find all keys that are in A and B and C.

(ii) We want to find all keys that are in A, or B, or C. (No duplicate listing of keys.)

Give efficient algorithms for solving each one of these problems. Analyze their worst-case running time. Make them as space efficient as possible. Justify your answers.

Problem 3. (40 POINTS)
This continues problem 2. You are given three sorted sequences A, B, C of $\lg n$, $\sqrt{n}$, and $n \lg n$ keys respectively. We want to determine the keys that appear in exactly two of the three sets. Give an efficient (space and time) algorithm that finds sequence D whose keys are in exactly two of A and B and C. For example, if the keys happen to be numbers and say, $n = 16$, if $A = \langle 1, 5, 7, 34 \rangle$, $B = \langle 2, 5, 8, 9 \rangle$, $C = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, \ldots, 64 \rangle$ (C is all numbers between 1 and 64), then $D = \langle 1, 2, 7, 8, 9, 34 \rangle$.

Problem 4. (40 POINTS)
Suppose that we try to insert n keys into a hash table of size m using open addressing and uniform hashing. Let $p(n, m)$ be the probability that no collisions occur. Show that $p(n, m) \le exp(-n(n-1)/(2m))$. Argue that when n exceeds $\sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

Hint: Use induction... Also use $exp(x) \ge 1 + x$ for any real x. Note that $exp(x) = e^x$. If you argue probabilistically (too much) you might get confused.

Problem 5. (40 POINTS)
This is Problem 11-3 of CLRS (page 250-251, 283-284 depending on edition). Suppose that we are given a key k to search for in a hash table with positions $0, \ldots, m-1$, and suppose that we have a hash function h mapping the key space into the set $\{0, \ldots, m-1\}$. The search scheme is as follows.

1. Compute the value $j = h(k)$ and set $i = 0$.
2. Probe in position j for the desired k. If you find it, or if this position is empty, terminate the search.
3. Set $i = i + 1$. If i equals m the table is full, otherwise set $j = (i + j) \bmod m$ and return to step 2. (The mod$m$ operator find the remainder of the division by m.)

Assume that m is a power of 2.

a. Show that this scheme is an instance of the general quadratic probing scheme by exhibiting the appropriate constant c_1, c_2 for $h(k, i) = h(k) + c_1 i + c_2 i^2 \bmod m$.

b. Prove that this algorithm examines every table position in the worst case.

Problem 6. (40 POINTS)
You have a machine with 2000MB of free memory. We want you to use (potentially) all of its memory to store `wordIDs` and corresponding `words`, similarly to the structure shown in class Subject 2 (cases T2∗). An array A will be used to store `words` delimited by nulls (`\0`). A hash table T will store a `wordIDs` along with a reference/pointer p to A. That way a `wordID` will be associated with a specific `word`.
The Operating System consumes 70MB and other user programs including the ones dealing with A and T another 30MB. The average length of a `word` is given as 8 UNICODE characters (1 UNICODE uses two bytes). A `wordID` and p can only be multiples of one byte (i.e. 1B, 2B, 3B but not of bits, and thus 23bits is not a option) for efficiency. Organize the hash table and the array for maximum efficiency.
What is your choice of n, the number of words that can be accommodated? What is your choice of m, the number of slots of the hash table? How many bytes for a `wordID` and how many bytes for p? How much space in MB will your scheme consume for T and for A? Explain. (Round to nearest million multiple for n, m, T, A.) (List the 6 output values as shown in Problem 8.)

Problem 7. (40 POINTS)
Implement hashing by open-addressing consistent with the operations Insert/Delete/Search as defined in the textbook (Chapter 11) or the associated notes and possibly enhanced as follows. A hash table will consist of two entities: one the table T itself as used in open-addressing. The table itself won't store key values but indexes to a dictionary of words separated by null values (`\0`). Call the latter dictionary D. Insertion operations affect both T and D. Deletion operations won't delete strings from D, only indexes from T. Search operations will return not only the location in T but also the corresponding location in D (location of first character of word). Implement (an approximate interface is provided) the following functions. You may deviate from the interface depending on the use of C, C++, Java. However the testing interface must be maintained intact.

```
1.   int HashFunction(string k, probe i) // Hash function takes key k as input returns 0..m-1

   /* For open addressing  implement
    * h(string,i) --->   (h(key(string)) % m + i**2 + i ) % m
    *  where k=key(string) is a numerical conversion of the string.
    *  choice of key is up to you. You may use for example key=MD5
    * and isolate a subset of the 128 bits of it if necessary.
    * Deviation from standard: Location h(string,i) stores not the key
    * but a pointer/reference/index into an array D of characters.
    * Strings there are sequences of characters separated by \0
    * Thus the array of length 20 stores three strings alex, algorithm,data
    * 0       9        19
      alex0algorithm0data0
    */
and
   /* In the remainder T refers to both T and its associated dictionary D */
2.   HashCreate(table T, int m); // Create a hash table with m slots /Initialize
3.   HashEmpty (table T); //Check if Table is empty
4.   HashFull  (table T); // or full; this is different from overflow.
5.   HashInsert(table T, string k); //Insert string to T (and D)
6.   HashDelete(table T, string k); //Delete string from T (but not necessarily from D)
7.   HashSearch(table T, string k, int m); //Search for string in T (and D)

8.   HashTable(table T, operation o, string k); //Generic Interface for operations
                                              // o id 10,11,12,13
9.   HashPrint(table T);                      //Print D not T
10.  ProcessHash(file file-name)
```

The end result is the implementation of HashTable, a function that can implement an open-addressing scheme with quadratic probing as defined above. An operation can be defined in a single line with two arguments. The first being the operation (10 for Insertion, 11 for Deletion, 12 for search, 13 for HashPrint) and the second the key value involved (except for HashPrint) given in the form of a string with no more than 20 characters (if this is important to your implementation). If the operation is 13 (HashPrint) no key value needs to be present and the dictionary D is printed in the following compact form (note what only gets printed!)

```
0  alex
5  algorithm
15 data
```

I will test your code, through the command line, by typing in `% ./ProcessHash my-test-file` . A test file might look like the following one. Your program should support such an interface. The semantics of the other functions are thus not important; the semantics of ProcessHash however need to be maintained intact.

```
10 alex
10 algorithm
10 data
12 alex
11 algorithm
10 structures
12 algorithm
12 structures
13
12 algorithm
```

An operation that cannot be performed correctly should return a `-1`, which is equivalent to string not found. An operation that successfully terminates should return the position of the hash table relevant to the operation and the position in D containing the relevant word.

Problem 8. (40 POINTS)

Read also Problem 6 for background. Design a hash-table organization program that accepts from a file with a fixed name `hw2p6conf` the following input (the values of the parameters can vary and are given based on the statement of Problem 6). Your code understands units `MB, GB, B, M`, where `1GB=1000MB, 1MB=1000000B`. `1M=1000000`. In addition the encoding of the words is defined by CODE which can be ASCII (1B) or UNICODE (2B). Thus, a 7 character word would need 7B or 14B for the former or latter case.

```
RAM=2000MB
OS=70MB
PROGRAMS=30MB
AVERAGE=9
WORDID=YES
CODE=ASCII
```

Your design would output the optimal values of the six unknowns specified in Problem 6. In addition, it would output $A + T$ as a seventh output (and make sure it is no more than the size of `RAM`. A difference between this and Problem 6 is the variability allowed for `WORDID` being a `YES` (as in Problem 6) or `NO` as it was the example done in class. In the latter case you don't store a `WORDID` field into the hash table, just p. Another variability is the encoding of the words in A: `ASCII` (1B per char) or `UNICODE` (2B per char) are possible. (Order of output is also that of Problem 6.)

```
   n        =
   m        =
   wordID   =
   p        =
   T        =
   A        =
 A+T        =
```

Values for T, A are to be reported in `MB` with units properly displayed. `wordID`, `p` should be in `B` with unit information also displayed and following the restrictions of Problem 6. Values `n,m` in `M` with unit symbol also displayed.
Your code in C/C++ should be run with

```
% ./hw2p6 hw2p6conf
or
% ./hw2p6                      //  Reads default hw2p6conf
```

and for Java (name classes appropriately)

```
% java hw2p6 hw2p6conf
or
% java hw2p6                   //  Reads default hw2p6conf
```

For testing, although leading spaces will be avoided in any one of the lines of `hw2p6conf`, you may not assume this to be the case for the trailing white space after the values.