CS 667 : Homework 2(Due: Feb 21, 2012)

This Homework is Problems 1-5 and worth 250 points. You can replace some of these points with Problem 6 or 7, but you can only submit 250 points worth of problems.

Problem 1. (50 points)
(a) You are given six polynomials $f_1, \ldots f_6$ of degrees $1, 2, 3, 1, 4, 5$ respectively. We are interested in finding the product $f = f_1 f_2 f_3 f_4 f_5 f_6$. Assume that the cost of multiplying two polynomials of degree a and b is $a \cdot b$. Find a schedule for multiplying the six polynomials that is of the lowest possible total cost.
(b) You are given six polynomials $f_1, \ldots f_6$ of degrees $1, 2, 3, 1, 4, 5$ respectively. We are interested in finding the product $f = f_1 f_2 f_3 f_4 f_5 f_6$. Assume that the cost of multiplying two polynomials of degree a and b is $a + b$ (note the difference from the previous problem) i.e. it is proportional to the space required to store the product which is a polynomial of degree $a + b$. Find a schedule for multiplying the six polynomials that is of the lowest possible total cost for this non-traiditional definition of a cost function.
Example for (b). If you have three polynomials g_1, g_2, g_3 of degrees $1, 2, 3$ respectively and you first compute $g_2 g_3$ and then the multiply the result by g_1, the cost of the first multiplication is 5 $(= 2+3)$ and the cost of the second multiplication is 6 since you multiply the result, a degree 5 polynomial to a degree one polynomial. Total cost is $5 + 6 = 11$. Is this the best one can do for these three polynomials?

Problem 2. (50 points)
(a) Show how to multiply two complex numbers $z_1 = a + bi$ and $z_2 = c + di$ using only three real multiplications. The algorithm should take a, b, c, d as input and produce the real component $Re(z)$ and the imaginary component $Im(z)$ of $z = z_1 z_2$, which are defined as $Re(z) = ac - bd$ and $Im(z) = ad + bc$. Then $z = Re(z) + iIm(z)$.
Note that the naive method requires 4 real multiplications to find ac, bd, ad and bc and two additions/subtractions to subsequently derive $Re(z)$ and $Im(z)$. You must beat this method by a single multiplication. Can you do it with two multiplications?
(b) Show how to multiply two polynomials of degrees/bounds n in time $o(n^2)$. Assume that n or $n - 1$ is a power of two.
Hint. $a + bi$ can be viewed as monomial $a + bx$ where $x = i$ and thus $x^2 = i^2 = -1$, i.e. you never get $x^3, x^4, \ldots$ because $x^3 = -i, x^4 = 1, \ldots$.

Problem 3. (50 points)
Evaluating a polynomial A(x) of degree-bound n at a given point x_0 can also be done by dividing $A(x)$ by the polynomial $(x - x_0)$ to obtain the quotient polynomial $q(x)$ of degree-bound $n-1$ and a remainder r, such that $A(x) = q(x)(x - x_0) + r$. Clearly $A(x_0) = q(x_0)(x_0 - x_0) + r = r$. Show how to compute the remainder r and the coefficients of $q(x)$ in time $\Theta(n)$ from x_0 and the coefficients of A.

Problem 4. (50 points)
Evaluate the following polynomial $f(x)$ of degree n, where $n = 2m + 1$, at $x = c$ using $(n + 1)/2 + O(1) = m + O(1)$ additions/subtractions and $(n + 1)/2 + O(1) = m + O(1)$ multiplications. You are not allowed to use divisions or any operations other than $+, -, *$. (You may give a straight line program if you wish to do so.)

$$f(x) = 1 + x + x^2 + \ldots + x^n$$

Hint. $x^2 + x^3 = (1 + x)x^2$.

Problem 5. (50 points)
Evaluate the following polynomial $f(x)$ of degree n, at $x = c$ using $o(n)$ additions/subtractions and $o(n)$ multiplications (you are not allowed to use divisions or any arithmetic operations other than $+, -, *$). (The $o(n)$ involves a little-Oh.)

$$f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Addendum: Check Appendix C for inspiration. Notation-wise, $\binom{n}{k}$ (i.e. n choose k or choose k out of n) is defined as

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Problem 6. (50 POINTS)
Implement Shamir's secret sharing scheme.

```
ShamirCreate(secret k, parties p, reconstruct  r, file-out file-name)
  // Returns  a file-name that contains one per-line the individual secrets
  // assigned to each of the p parties. file-name thus has p lines one for each party

secret ShamirReconstruct(parties p, reconstruct  r, file-in  file-name)
  // Uses file-name with at least r lines but no more than p to reconstruct
  // the secret s that is returned.

  // Details are left to you for implementation.
```

The interface will be through the command-line. A

```
% ./ShamirCreate k p r out-file
or
% java ShamirCreate k p r out-file
```

will call the corresponding function and generate some output in file out-file. (Note that `ShamirCreate` is not only a function name but also a program name.)

A

```
% ./ShamirReconstruct p r my-file
or
% java ShamirReconstruct p r my-file
```

will use my-file (containing lines of out-file) to return in the standard output the secret.

The catch of this Problem: Numbers can grow very big! The secret is a positive 32-bit integer `int` or `unsigned int`.

Problem 7. (100 POINTS)

You can replace 100 points worth of Problems 1-5 (i.e. two problems) with this Problem.

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.