A. V. Gerbessiotis — CS 667-101 — Fall 2007

PS 1 — Sep 12, 2007 — 250 points

CS 667 : Homework 1(Due: Sep 26, 2005)

Problem 1. (50 points)

Professor I.M. Nuts proposes the following algorithm `Permute` to generate (uniformly at) random permutations on n elements of array A. Does his method work? Explain. **Note.** If the algorithm works fine, you need to show that all possible $n!$ permutations can be generated, and also, that each one of these permutations is equally likely to occur.

```
Permute(A,n) // A is an array A[1..n]
  for(i=1;i<=n;i++)
     swap(A[i], A[random(1,n)]); //random(1,n) returns a uniformly at random integer between 1 and  n
```

Problem 2. (50 points)

We are interested in determining whether n is a perfect power i.e. whether there exist integer x, y such that $n = x^y$ and if such x, y exist also determining them. Answer the following questions.

(a) If n is indeed a perfect power, how large can y be? Express your answer as a function of n only.

(b) Suppose you have a black box `Broot (N,Y)` that given N and Y it finds the integer Y-th root of N if such exists or returns `-1` if such root does not exist. For example `Broot(8,3)` would return 2 since $2^3 = 8$, whereas `Broot(8,2)` would return -1. Suppose that the time it takes for `Broot` to return an answer is $O(1)$. How could you use `Broot` to determine whether n is a perfect power or not? How many times do you need to call `Broot` in an efficient determination of whether n is a perfect power or not?

(c) `Broot` in $O(1)$ time is rather unrealistic. How fast could you implement it? Does a solution that is $O(\log^2 n)$ (or $o(\log^2 n)$) exist? Explain.

Problem 3. (50 points)

Consider two sets A and B each having n integers in the range from 0 to $25n$. We wish to compute the Cartesian sum of A and B defined by

$$C = \{x + y : x \in A \ \ and \ \ y \in B\}$$

Note that the integers in C are in the range from 0 to 50n. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Suppose that we have a black-box function `BProduct` that takes as input two polynomials of degree m and returns its product in time $O(m \lg m)$. Show that you could use `BProduct` to find C in time $O(n \lg n)$. **Hint.** Represent A and B as polynomials of degree at most $25n$.

Problem 4. (50 points)

(a) You are given six polynomials $f_1, \ldots f_6$ of degrees $1, 2, 3, 2, 4, 5$ 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 $1, 2, 3, 2, 4, 5$ 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-traditional definition of a cost function.

Example. 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 you can do for these three polynomials?

Problem 5. (50 points)

This is Problem 11-3 of CLRS (page 250-251). 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 $i = h(k)$ and set $j = 0$.
2. Probe in position i for the desired k. If you find it, or if this position is empty, terminate the search.
3. Set $j = (j + 1) \bmod m$ and $i = (i + j) \bmod m$ and return to step 2.

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 P1. (80 points)

Implement hashing by chaining and hashing by open-addressing. Implement (approximate interface) the following functions.

```
int HashFunction(key k)
or
int HashFunction(key k, probe i) // Hash function takes key k as input returns 0..m-1
/* For open addressing (Oa) implement
 * h(k,i) as  (h(k) % m + i**2 + i ) % m
 */


HashChainCreate(table T, int m); // Create a hash table/Initialize
HashChainEmpty (table T, int m); //Check if Table is empty
HashChainFull  (table T, int m); // or full
HashChainInsert(table T, key k, int m);
HashChainDelete(table T, key k, int m);
HashChainSearch(table T, key k, int m);
```

```
and
   HashOaCreate(table T, int m); // Create a hash table/Initialize
   HashOaEmpty (table T, int m); //Check if Table is empty
   HashOaFull  (table T, int m); // or full; this is different from overflow.
   HashOaInsert(table T, key k, int m);
   HashOaDelete(table T, key k, int m);
   HashOaSearch(table T, key k, int m);

   HashTable(type  t, table T, operation o, key k);
   ProcessHash(file file-name)
```

The end result is the implementation of HashTable, a function that can implement both types hash tables (eg. if t is equal to 0 then it means chaining, and 1 open-addressing with quadratic probing as defined above). An operation o can be defined in a single line with two arguments. The first being the operation (10 for Insertion, 11 for Deletion, and 12 for search) and the second the key value involved (assume integers keys.

I will test your code, through the command line, by typing in `Your program should support such an interface.`

```
 1      <<<<mean open-addressing
10 1
10 10
10 20
10 8
10 7
12 10
11 20
11 8
10 30
12 25
```

`12 10 returns the index of the hash table containing key 10, but 12 25 returns -1 (key not found) .`

Problem P2. (120 points)

Huffman coding. Says all. Arguments ($N \leq 10$) in the command-line won't exceed 10 in the file option case, there is only one in the directory argument case. File inputs can be arbitrary files.

```
% ./huffman-encode file1 file2 ... fileN
% ./huffman-encode dir1
% ./huffman-decode file1 file2 ... fileN
% ./huffman-decode dir1
 // Encode : converts file1 ---> file1.huf
 // Decode : From command-line file1 reads (if it exists) file1.huf and converts it into file1
 // file1 ... fileN may have suffixes: eg myfile.pdf ---> myfile.pdf.huf --> myfile.pdf
 // Operation is destructive! myfile.pdf would be erased after creating myfile.pdf.huf

 Example
% ./huffman-encode myfile.tex
  // encode myfile.tex into myfile.tex.huf using Huffman coding. myfile.tex still exists.
  // If a previous myfile.tex.huf exists, it will be overwritten. No warning required.
% ./huffman-decode myfile.tex
  // It locates a myfile.tex.huf and decodes it by overwriting a myfile.tex if it still
  // exists.
  // If a previous myfile.tex exists, it will be  overwritten. No warning required.
```