Data Structure and Algorithm
Homework #1

Due: 2:20pm, Tuesday, March 12, 2013

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1, submit your source code, a Makefile to compile the source, and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "`hw1`" and put these three files in it.
- The filenames of the source code, Makefile, and the documentation file should be "`main.c`", "`Makefile`", and "`report.txt`", respectively; you will get some penalties in your grade if your submission does not follow the naming rule. The documentation file should be in plain text format (`.txt` file). In the documentation file you should explain how your code works, and anything you would like to convey to the TAs.
- For Problem 2 through 5, submit the answers via the SVN server (electronic copy) or to the TA at the beginning of class on the due date (hard copy).
- Except the programming assignment, each student may only choose to submit the homework in only one way; either all in hard copies or all via SVN. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted via SVN (the part that the grading TA chooses).
- If you choose to submit the answers of the writing problems through SVN, please combine the answers of all writing problems into only ONE file in the pdf format, with the file name in the format of "`hw1_[student ID].pdf`" (e.g. "`hw1_b01902888.pdf`"); otherwise, you might only get the score of one of the files (the one that the grading TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem.
- **NO LATE SUBMISSION IS ALLOWED** for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - DelayTime/86400)$$

***Problem* 1.** Basic Matrix Operation (30%)

You have implemented an algorithm to calculate the determinant of a matrix in Homework #0. In this problem, we ask you to write a program to utilize the determinant calculation to navigate in a maze.

In the input of this problem, the first line includes the entry coordinates (x, y) of the maze. The dimensions of the maze, (m, n), are presented in the second line. Two matrices are used to determine whether a move in the maze is valid. Matrix A, used as the "value matrix", starts from line 3. Matrix B, used as the "hint matrix", is presented after the lines containing matrix A. Both matrices have m rows and n columns. Numbers in the same row are separated by single space characters ' '.

The rules of the maze are as follows:

1. You will enter the maze from the entry coordinates (x_1, y_1). Your goal is to move in the maze according to the rules until you reach a dead end.
2. From coordinates (x, y), it is only possible to move to 4 adjacent coordinates, $(x \pm 1, y)$ or $(x, y \pm 1)$. To determine whether the move to one of these coordinates, denoted as (x', y'), is valid, calculate the determinant of the sub-matrix $B[x-1 : x+1][y-1 : y+1]$, i.e., the 3x3 sub-matrix centred at $B[x][y]$. If $\det(B[x-1 : x+1][y-1 : y+1]) = A[x'][y']$, then the move is valid (representing an empty space that you can move into). Otherwise, the move is invalid (representing a wall).
3. Moving to the edge of the maze, the first row/column and the last row/column is not valid. (Therefore, it is always possible to find the sub-matrix $B[x'-1 : x'+1][y'-1 : y'+1]$).
4. When it is possible to move to more than one adjacent coordinates from the current coordinates, you should move according to the following priorities: right $(y+1) >$ down $(x+1) >$ left $(y-1) >$ up $(x-1)$.
5. Moving to the coordinates where you just came from (where you were before the last move) is invalid, i.e., turning back is not allowed.
6. When you reach a dead end, i.e., there is no valid move to adjacent coordinates, end the navigation in the maze and output all coordinates you have been to in the maze, including the entry coordinates and the coordinates of the dead end.

The input has the following format: (download a sample of the input, "hw1_input_example", from the course website).

$x_1 \quad y_1 \leftarrow$ Maze entry X and Y coordinates
$m \quad n \leftarrow$ Numbers of rows (m) and columns (n) of the matrices
$a_{11}\ a_{12}\ \ldots\ \ldots\ a_{1n} \leftarrow$ the n numbers of the first row of matrix A
$a_{21}\ a_{22}\ \ldots\ \ldots\ a_{2n} \leftarrow$ the n numbers of the second row of matrix A
$\vdots$
$a_{m1}\ a_{m2}\ \ldots\ \ldots\ a_{mn} \leftarrow$ the n numbers of the m-th row of matrix A
$b_{11}\ b_{12}\ \ldots\ \ldots\ b_{1n} \leftarrow$ the n numbers of the first row of matrix B
$b_{21}\ b_{22}\ \ldots\ \ldots\ b_{2n} \leftarrow$ the n numbers of the second row of matrix B
$\vdots$
$b_{m1}\ b_{m2}\ \ldots\ \ldots\ b_{mn} \leftarrow$ the n numbers of the m-th row of matrix B

Output the coordinates that you have been to in the maze in the order that you visited them in the following format:

$x_1 \quad y_1 \leftarrow$ Maze entry coordinates
$x_2 \quad y_2 \leftarrow$ second visited coordinates
$x_3 \quad y_3 \leftarrow$ third visited coordinates
$\vdots$
$x_n \quad y_n \leftarrow$ last visited coordinates (dead end)

For this problem, you can utilize the following assumptions:

1. $m \leq 1000$ and $n \leq 1000$.
2. Every entry of the matrices in the input can be stored in a 16-bit integer.
3. The empty space in the maze will not form any cycle.
4. You can assume that there will be no overflow in the process of determinant calculation of a 3x3 matrix if 64-bit (*long long int*) integers are used to store the matrix elements. Since we only need to calculate the determinant of a 3x3 matrix, feel free to use a formula (such as the one shown here: `http://en.wikipedia.org/wiki/Determinant`) to calculate the determinant without worrying about the overflow problem.
5. Take the input from the standard input device (*stdin*) and output your results to the standard output device (*stdout*).

Problem 2. Asymptotic Notation (20%)

2.1. (10%) Let $p(n) = \sum_{i=0}^{d} a_i n^i$, where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties. (please refer to p.43-52 of Cormen et al. for the notations which are not covered in the lecture.)

(a) If $k \geq d$, then $p(n) = \mathrm{O}(n^k)$.

(b) If $k \leq d$, then $p(n) = \Omega(n^k)$.

(c) If $k = d$, then $p(n) = \Theta(n^k)$.

(d) If $k > d$, then $p(n) = \mathrm{o}(n^k)$.

(e) If $k < d$, then $p(n) = \omega(n^k)$.

2.2. (10%) Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following statements. Note that if it is a "if and only if" statements, then you need to prove both directions.

(a) $f(n) = \mathrm{O}(g(n))$ if and only if $g(n) = \Omega(f(n))$

(b) $f(n) = \Theta(g(n))$ if and only if $f(n) = \mathrm{O}(g(n))$ and $f(n) = \Omega(g(n))$

(c) $f(n) = \mathrm{O}(g(n))$ implies $f(n) \cdot g(n) = \mathrm{O}(g(n)^2)$

(d) $f(n) = \mathrm{O}(g(n))$ implies $(f(n))^2 = \mathrm{O}(g(n)^2)$

(e) $f(n) = \mathrm{O}(g(n))$ implies $2^{f(n)} = \mathrm{O}(2^{g(n)})$
(Hint: Let $f(n) = 2g(n)$, then $2^{f(n)} = 2^{2g(n)} = 2^{g(n)} \times 2^{g(n)}$)

Problem 3. Time and Space Complexities (15%)

In a search problem, you are given a sorted sequence of numbers S and a sequence of query numbers Q. For each query number in Q, you are required to output whether the number exists in S or not. The following two pseudo code segments present two different algorithms to solve this problem.

Assume that the lengths of S and Q are n and m, respectively. The values in S and Q are not greater than K. For the time and space complexity analysis in this problem, you do not need to consider the time and space costs of the input data (n, S, and Q). You should give the complexities in big-Oh notation in terms of n, m, and K.

```
Function binary_search(n, S, Q) {
    while(Q.nonEmpty()){
        searchnum = Q.getNextQuery();
        left = 0, right = n-1;
        while(left <= right) {
            middle = (left+right) / 2;
            if(S[middle] == searchnum) print("Yes"); break;
            if(S[middle] < searchnum)  left = middle+1;
            if(S[middle] > searchnum)  right = middle-1;
        }
        if(left > right) print("No");
    }
}

Function count_search(n, S, Q) {
    C = malloc(K);
    // Assume that all elements of C are initialized as false.
    for(i=0;i<n;i++)
        C[S[i]] = true;
    while(Q.nonEmpty()){
        searchnum = Q.getNextQuery();
        if(C[searchnum] == true) print("Yes");
        else                     print("No");
    }
    free(C);
}
```

Example. Since *binary_search* uses only four additional variables, the space complexity (without considering the space cost of the input) is O(1).

3.1. (4%) Give the time complexity of *binary_search*.

3.2. (4%) Give the time complexity of *count_search*.

3.3. (3%) Give the space complexity of *count_search*.

3.4. (4%) Which method is better if $m = \mathrm{O}(1)$? Explain your reason briefly.

Problem 4. Stack and Queue (20%)

4.1. (5%) You are given an initially empty stack. Perform the following stack operations in the given order. Please write down the output of each POP operation.

```
PUSH 3
PUSH 5
POP
PUSH 4
POP
POP
PUSH 2
PUSH 1
POP
POP
```

4.2. (5%) You are given an initially empty queue. Perform the following queue operations in the given order. Please write down the output of each DEQUEUE operation.

```
ENQUEUE 1
DEQUEUE
ENQUEUE 5
ENQUEUE 4
DEQUEUE
DEQUEUE
ENQUEUE 2
ENQUEUE 3
DEQUEUE
DEQUEUE
```

4.3. (6%) The following 3 sequences of operations could be operations for a stack, a queue, or neither. The IN operation means either ENQUEUE or PUSH, while the OUT operation means either DEQUEUE or POP. `OUT` x means that the output of the operation is x. For each of the 3 sequences, determine the data structure that the sequence is for.

Set 1	*Set 2*	*Set 3*
`IN   4`	`IN   4`	`IN   3`
`IN   3`	`OUT  4`	`IN   1`
`IN   1`	`IN   3`	`OUT  1`
`OUT  1`	`IN   1`	`OUT  3`
`OUT  3`	`IN   3`	`IN   3`
`IN   3`	`OUT  3`	`IN   2`
`IN   2`	`IN   4`	`OUT  3`
`OUT  2`	`OUT  1`	`OUT  2`
`OUT  3`	`OUT  3`	

4.4. (4%) In problem **4.3**, how do you check if a sequence of operations is for a stack, a queue, or neither? Briefly describe your method.

Problem 5. Scorched Toast (15%)

Twilight was trying to use magic to bake bread. But she failed and some parts of bread are scorched. The shape of the bread is a rectangle, and it can be represented by a $M \times N$ 0-1 matrix B composed by $M \times N$ small pieces. The *0*s pieces ($B_{ij} = 0$) represent the scorched area, and the *1*s pieces ($B_{ij} = 1$) represent the other successful area. For example, Figure 1 is a 4×5 bread, the *1*s pieces and *0*s pieces represent the successful pieces and scorched pieces. However, she wants to show off her "great" baking technique to her friends, so she decides to pick a successful sub-rectangle of her bread with the maximum area. For example, the maximum successful sub-rectangle in Figure 1 is the sub-rectangle in orange color, which has an area of 6. Can you help Twilight to solve this problem?

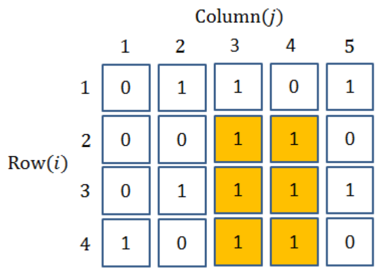

Figure 1: The example of problem 5.1.

5.1. (5%) Brute-force is the simplest method. We can enumerate four boundaries of the sub-rectangle and check whether all pieces of the sub-rectangle are successful. Please design an algorithm using the described brute-force method, write down its pseudo code and show that the time complexity of the algorithm is $O(M^3N^3)$.

5.2. (5%) Define $S(i,j) = \begin{cases} 1 + i - \min\{k \mid k \text{ s.t. } B_{pj} = 1, \forall k \le p \le i\} & \text{if} \quad B_{ij} = 1 \\ 0 & \text{if} \quad B_{ij} = 0 \end{cases}$, which means that the number of successful pieces of B_{ij} and above connected pieces. For the bread in Figure 1, $S(4,2) = 0$, $S(3,3) = 3$ and $S(4,3) = 4$. Please design an algorithm to calculate $S(i,j)$ in $O(MN)$.

5.3. (5%) We can define the maximum potential height $H_i(a,b)$ as the maximum possible height of the sub-rectangles between the left boundary a and the right boundary b and with the bottom in row i. For example, $H_i(k,k)$ is the maximum potential height of the sub-rectangles on column k (with width 1), since both boundaries, a and b, are on column k, and with the bottom in row i. It is obvious that $H_i(k,k) = S(i,k)$.

It is also not hard to see that the maximum potential height $H_i(a,b)$ is the minimum of $S(i,k)$ of all k in $\{k : a \le k \le b\}$. The area of the maximum sub-rectangle with the bottom in row i and between the left and right boundaries a and b is $(b - a + 1) \cdot H_i(a,b)$.

We can then enumerate the left and right boundaries and calculate $H_i(a,b)$ to find out the maximum sub-rectangle with the bottom in row i. Finally, we can enumerate the bottom row

i and use the same method to determine the maximum sub-rectangle with different bottom row to find out the final solution.

Please design an algorithm using the described method, write down its pseudo code, and show that its time complexity is $\mathrm{O}(MN^3)$.