Algorithm Design and Analysis

Homework #5

Due: 1pm, Monday, December 26, 2011

=== Homework submission instructions ===

- Submit the answers for writing problems (including your programming report) through the CEIBA system (electronic copy) or to the TA in R432 (hard copy). Please write down your name and school ID in the header of your documents. You also need to submit your programming assignment (problem 1) to the Judgegirl System(`http://katrina.csie.ntu.edu.tw/judgegirl/`).

- Each student may only choose to submit the homework in one way; either all as hard copies or all through CEIBA except the programming assignment. 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 through CEIBA (the part that the TA chooses).

- If you choose to submit the answers of the writing problems through CEIBA, please combine the answers of all writing problems into only one file in the doc/docx or pdf format, with the file name in the format of "hw5_[student ID].{pdf,docx,doc}" (e.g. "hw5_b99902010.pdf"); otherwise, you might only get the score of one of the files (the one that the TA chooses).

- For each problem, please list your references (they can be the names of the classmates you discussed the problem with, the URL of the information you found on the Internet, or the names of the books you read). The TA can deduct up to 100% of the score assigned to the problems where you don't list your references.

Problem 1. (30%) **Arbitrage** is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 49 Indian rupees, 1 Indian rupee buys 2 Japanese yen, and 1 Japanese yen buys 0.0107 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $49 \times 2 \times 0.0107 = 1.0486$ U.S.

dollars, thus turning a profit of 4.86 percent. Suppose that we are given n currencies $c_1, c_2, ..., c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j. We would like to find a sequence of currencies $\langle c_{i_1}, c_{i_2}, ..., c_{i_k} \rangle$ such that $R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]$ is maximized, where $k \leq n$. If $\max \{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]\} > 1$, we call this set of currencies *profitable*. To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but any starting currency may be considered.

Input:

The first line contain n, $2 \leq n \leq 50$. The next n lines represent the exchange rate table R in the following format:

$R[1,1]\ R[1,2]\ R[1,3] \ldots R[1,n]$

$R[2,1]\ R[2,2]\ R[2,3] \ldots R[2,n]$

...

$R[n,1]\ R[n,2]\ R[n,3] \ldots R[n,n]$

with all numbers ranging from 10^{-5} to 10^5.

Output:

For each intput table you must determine whether a sequence of exchanges exists that results in a profit of more than 1 percent (0.01). If a sequence exists you must print the sequence of exchanges that results in a profit. If there is more than one sequence that results in a profit of more than 1 percent you must print a sequence of minimal length and starting currency index, i.e., one of the sequences that uses the fewest exchanges of currencies to yield a profit. All profiting sequences must consist of n or fewer transactions where n is the dimension of the table giving exchange rates. You should output the profitable sequence in the following format:

$i_1, i_2, ..., i_k, i_1$

where $i_1, i_2, ..., i_k, i_1$ is the sequence of the indices of the currencies in the profitable exchange, where $i_1 < i_j$, for $j = 2, ..., k$. The indices from i_2 to i_k are all distinct. If no profitable sequence can be found, output "Not profitable". All numbers in this output line are separated by comma.

Sample Input 1:

```
3
1 0.6 0.8
1.6 1 1.2
1.2 0.7 1
```

Sample Output 1:

```
Not profitable
```

Sample Input 2:

```
3
1 0.05 0.21
16 1 4
5.1 0.09 1
```

Sample Output 2:

```
1,3,1
```

Please write a program to solve this problem. Please also submit a report in which you give a clear description of your algorithm and analyze the running time of your algorithm.
Sol: To be announced later.

Problem 2. (10%) Read section 17.4 on the textbook. Then solve problem 17.4-3 on page 471.
Sol:

(a) If the ith operation does not trigger a contraction, then

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + |2num_i - size_i| - |2num_{i-1} - size_{i-1}| \\
&= 1 + |2num_i - size_i| - |2(num_i + 1) - size_i| \\
&= 1 + |2num_i - size_i| - |2num_i - size_i + 2| \\
&\leq 1 + 2 = 3.
\end{aligned}$$

(b) If the ith operation does trigger a contraction, then $num_i < size_{i-1}/3 = size_i/2$, and thus $2num_i - size_i < 0$. Therefore,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (num_i + 1) + |2num_i - size_i| - |2num_{i-1} - size_{i-1}| \\
&= (num_i + 1) + |2num_i - size_i| - |2(num_i + 1) - 3size_i/2| \\
&= (num_i + 1) + (size_i - 2num_i) - (size_i - 2num_i + size_i/2) + 2 \\
&= num_i - size_i/2 + 3 \\
&\leq 3.
\end{aligned}$$

For both of the cases, the amortized cost $\hat{c}_i$ is bounded by a constant.

Problem 3. (20%) Solve problem 17-5 on page 476 in the textbook.

Sol:

a. Each time when we encounter a key in the access sequence, the worst case is to check the whole list, taking $O(n)$ time, thus is takes $\Omega(mn)$ time in the worst case.

b. Since it takes $\text{rank}_L(x)$ to find x in L and $\text{rank}_L(x) - 1$ to move x to the front of the list by transpositions, $c_i = 2 \cdot \text{rank}_L(x) - 1$.

c. Since it takes $\text{rank}^*_{L-1}(x)$ to find x in L and t^*_i to move x to the front of the list by transpositions, $c^*_i = \text{rank}_{L^*_{i-1}}(x) + t^*_i$.

d. Consider transposing two adjacent list elements x and y in L_i, so that y precedes x after the transposition. If x precedes y in ${L_i}^*$, the potential would increase by 2; for otherwise, the potential would decrease by 2.

e. Since $\text{rank}_{L_{i-1}}(x)$ counts the number of elements precedes x including itself, and the elements in L_{i-1} that precedes x is either followed or preceded by x in ${L^*}_{i-1}$, $\text{rank}_{L_{i-1}}(x) = |A| + |B| + 1$. With the same argument, we can get $\text{rank}_{{L^*}_{i-1}}(x) = |A| + |C| + 1$.

f. The change in potential of $\Phi(L_i) - \Phi(L_{i-1})$ counts the number of the maximum numbers of inversions occurring in the new round. Moving x to the front of L would cause $|A|$

inversions and eliminate $|B|$ inversions while there is no element precedes x, with no more than t_i^* inversions occurs in the worst case while performing transpositions in L_i^*, therefore $\Phi(L_i) - \Phi(L_{i-1}) \leq 2(|A| - |B| + t_i^*)$.

g.

$$\begin{aligned}
\hat{c_i} &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2 \cdot \text{rank}_{L_{i-1}}(x) - 1 + 2(|A| - |B| + t_i^*) \\
&= 2(|A| + |B| + 1) - 1 + 2(|A| - |B| + t_i^*) \\
&= 4|A| + 1 + 2t_i^* \\
&\leq 4(|A| + |C| + 1 + t_i^*) \\
&= 4(\text{rank}_{L_{i-1}^*}(x) + t_i^*) \\
&= 4c_i^*
\end{aligned}$$

h. Since $c_i \leq \hat{c_i} \leq 4c_i^*$ for $i = 1$ to n
$\Rightarrow C_{\text{MTF}}(\sigma) = \sum_{i=1}^{m} c_i \leq \sum_{i=1}^{m} 4c_i^* = 4\sum_{i=1}^{m} c_i^* = 4(C_H(\sigma))$.

Problem 4. (20%) Solve problem 17-1 on page 472. Key sentence: bit-reversal permutation swaps elements whose indices have binary representations that are the reverse of each other.

Sol:

a. Let $A[0..n-1]$ denote the input array and $S[0..n-1]$ be the array with all elements initialled to be 0, where $S[i]$ records whether $A[i]$ has been swapped or not. Given rev_k that runs in time $\Theta(k)$, the following procedure runs in time $O(nk)$ to perform the bit-reversal permutation on A.

```
BIT-REVERSAL-PERMUTATION(A[0..n − 1])
1  for(i = 1 to n − 1)
2     reverse=rev_k(i)
3     if(S[index] =false)
4        swap A[i] and A[reverse]
5        S[i] =true
6        S[reverse] =true
7     end if
8  end for
```

b. The idea of the procedure BIT-REVERSED-INCREMENT(A) is very similar to INCREMENT(A) as taught in the class (p.454-455), the only difference is that we start the bit check from the highest bit.

```
BIT-REVERSED-INCREMENT(A[0..k − 1])
1  i = k − 1
2  while(A[i] ≠ 0)
3     A[i] = 1
4     i = i − 1
5  end while
```

The amortized analysis of BIT-REVERSED-INCREMENT(A) is therefore also the same as of INCREMENT(A).

c. While we shift only one bit at a time in BIT-REVERSED-INCREMENT(A), the time complexity is $O(n)$ under the restriction.

Problem 5. (20%) **Nesting boxes**

A d-dimensional box with dimensions $(x_1, x_2, ..., x_d)$ **nests** within another box with dimensions $(y_1, y_2, ..., y_d)$ if there exists a permutation π on $1, 2, ..., d$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, ..., x_{\pi(d)} < y_d$.

1. Argue that the nesting relation is transitive.

2. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

3. Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, ..., B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, ..., B_{i_k}\rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, ..., k-1$. Express the running time of your algorithm in terms of n and d. (Hint: construct a directed graph where the vertices represents the boxes.)

Sol:

1. Suppose x nests within y and y nests within z. Let π_1 and π_2 be the permutations on $1, 2, \ldots, d$ such that $x_{\pi_1(1)} < y_1, x_{\pi_1(2)} < y_2, \ldots, x_{\pi_1(d)} < y_d$, and $y_{\pi_2(1)} < z_1, y_{\pi_2(2)} < z_2, \ldots, y_{\pi_2(d)} < z_d$. Let $\pi' = \pi_2 \circ \pi_1$. Then, $x_{\pi'(i)} = x_{\pi_2(\pi_1(i))} < y_{\pi_2(i)} < z_i$, for $i = 1, \ldots, d$. That is, x nests within z. Therefore the nesting relation is transitive.

2. Given two d-dimensional boxes x and y. Sort on $x_1, x_2, \ldots, x_d$ and $y_1, y_2, \ldots, y_d$ respectively tow get two sequences $(x'_1, x'_2, \ldots, x'_d)$ and $(y'_1, y'_2, \ldots, y'_d)$, where $x'_i \leq x'_j$ and $y'_i \leq y'_j$, for all indices$i < j$. Then x nests within y if and only if $x'_i < y'_i$, for $i = 1, \ldots, d$. This method takes time $O(d \log d)$.

3. We transform this problem to a graph problem. First, construct a directed acyclic graph $G = (V, E)$ with $V = \{s, t, v_1, v_2, \ldots, v_n\}$, where every $v_i \in V$ corresponds a box B_i. For the edge set E, add (v_i, v_j) in E if and only if B_i nests within B_j, each with weight equals to 1, and add other $2n$ edges (s, v_i) and (v_i, t) with edge weight equals to 0, for $i = 1, \ldots, n$. It takes $O(dn^2 + nd \log d)$ to construct this graph. Then, to find the longest sequence $\langle B_{i_1}, B_{i_2}, ..., B_{i_k}\rangle$ such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, ..., k-1$, it is sufficient to find the longest path from s to t in G. Since it takes $\Theta(|V| + |E|) = O(n^2)$ to find the longest lath in a directed acyclic graph (Section 24.2 and Exercise 24.2-3 in the textbook), the overall running time of this algorithm is $O(dn^2 + nd \log d)$.