Algorithm Design and Analysis

Homework #4

Due: 14:20, December 6, 2012

Homework submission instructions

- Submit your programming assignment (problem 1) to the Judgegirl System (http://katrina.csie.ntu.edu.tw/judgegirl/). Also, use SVN to commit your report (named "report.pdf") in the same folder of your code.
- Submit the answer of the writing problems by one of the following ways:
  1. Submit an electronic copy to the CEIBA system before the deadline; or
  2. Submit a hard copy right before the class on the day of the deadline.

  If you submit by both ways, the TA will randomly select only one of them for grading.
- Please make sure that you write down your name and school ID in the header of your documents. Otherwise, the TA will not grade them.
- If you choose to submit the answers of the writing problems via CEIBA, please combine the answers of all writing problems into only one file in the pdf format, with the file name in the format of "hw4 [ID].pdf" (e.g. "hw4 b99902010.pdf"). Otherwise, the TA will not grade them.
- What should be included in the report:
  1. Explain how your program works in detail.
  2. Derive the time complexity of your program and briefly explain why. No need for formal proof.
  3. **The reference of your report.**
- If you are stuck with any problem, you are welcome to ask the TAs via e-mails or in person during the office hour. The TA may discuss with you and give you some useful hints. :D

Problem 1 – Money

Description

A few days ago, you received a few Thin-Dog coupons that are worth M dollars in total. You decided to find a way to make money out of these coupons. However, as you cannot exchange them for cash directly, you would have to exchange for a few items in Thin-Dog with these coupons and then sell the items for cash. Note that you can purchase neither items nor additional coupons with cash. In this problem, please write a program to calculate the maximum amount of money that you can get out of these coupons.

Input

The first line contains an integer *M*, the total value of the coupons. (1 <= *M* <= 100,000)
The second line contains another integer *N*, the number of kinds of items in Thin-Dog. (1 <= *N* <= 100)
There will be three integers, n_i, b_i, s_i, in each of the following *N* lines and in the order given, separated by space characters, ' ';
n_i represents the number of item *i* in Thin-Dog. (1 <= n_i <= 100,000) Obviously, you can only exchange at most n_i item *i* with coupons.
b_i represents the value of item *i* when the item is exchanged with coupons. (1 <= b_i<= 1,000,000,000)
s_i represents the value of item *i* when the item is sold for cash. (1 <= s_i <= 1,000,000,000)

Output

Please output a single line containing the maximum amount of money you can get out of the coupons.

Sample Input

```
100
2
4 6 5
2 40 33
```

Sample Output

81

Hint

1. 81 = 3 * 5 + 2 * 33
2. Remember to use **long long** data type.
3. Is this a DP problem? Yes, and you will need an "amortized algorithm".
4. DP[i][j] = the maximum money you can get with *$j* coupons when only the first *i* kinds of items could be exchanged with coupons.
5. Try to calculate DP[i][0..M] in O(M). Think about a moving window problem; in this problem you are given $\{a_1, a_2, a_3, \dots, a_L\}$ and you have to find the following (L-w+1) numbers:
   $\{\max\{a_1, a_2, \dots, a_{1+w}\}, \max\{a_2, a_3, \dots, a_{2+w}\}, \dots, \max\{a_{L-w+1}, a_{L-w+2}, \dots, a_L\}\}$.
   This is a moving window problem since you try to obtain the maximums of windows of *(w+1)* numbers while you move the window gradually to the right until the right side of the window reaches the end. Try to find a O(L) algorithm to solve the problem. Then, try to find a way to relate the moving window problem to a dynamic programming algorithm which can calculate DP[i][0..M] in O(M).

Grade

Judge result (10%)
Report & code (10%)

Problem 2 – Queue

A queue can be implemented with two ordinary stacks. The two main operations of the queue, ENQUEUE(x) and DEQUEUE(), perform the functions described as follows.

ENQUEUE(x): Push x into stack 1.

DEQUEUE(): If stack 2 is not empty, then we POP an element from stack 2 and return this element. If stack 2 is empty, then we POP an element of stack 1, PUSH the element into stack 2, and then continue the POP and PUSH operations until all elements in stack 1 have been transferred to stack 2. Then, we POP an element from stack 2 and return this element.

1) (3%) Write down the pseudo code of ENQUEUE(x) and DEQUEUE()
2) (2%) Show the content of the two stacks after executing each of the following sequence of operations: ENQUEUE(a), ENQUEUE(d), ENQUEUE(a), DEQUEUE(), DEQUEUE().
3) Prove that

A) (1%) Enqueuing an element in this queue always takes exactly one PUSH.
B) (2%) Dequeuing an element from this queue takes at most *n+1* POPs and *n* PUSHes.
C) (2%) By A) and B), without an amortized analysis, show that if we start with an empty queue and perform *n* ENQUEUE(x) and then *n* DEQUEUE(), it will take $O(n^2)$ PUSHes and $O(n^2)$ POPs.

4) However, the $O(n^2)$ bound is not tight enough for a sequence of *n* operations on this data structure. Show that $O(n)$ is a worst case bound on the cost of *n* operations with each of the following amortized analysis methods.
   A) (10%) The accounting method
   B) (10%) The potential method

Problem 3 – Counter

We have seen in the lecture that the amortized cost of each INCREMENT operation of a k-bit binary counter is $O(1)$.

1) (5%) Show that if a DECREMENT operation is also implemented for a k-bit binary counter, the $O(1)$ bound on the amortized cost of each operation, either an INCREMENT or a DECREMENT, will not hold anymore.
2) (15%) Now, consider a ternary number system, in which a number is represented by a sequence of digits $A[0..k-1]$, $A[i] \in \{-1, 0, 1\}$. The value of the number is given by $\sum_{i=0}^{k-1} A[i]2^i$, where k is the number of digits. For example, $A[] = \{-1,0,-1,-1,1\}$ is a representation of $2^4 - 2^3 + 2^2 - 2^0 = 11$. Consider the following procedures of INCREMENT and DECREMENT operations.

```
INCREMENT(A)                    DECREMENT(A)
    i = 0                           i = 0
    while A[i] = 1 do               while A[i] = -1 do
        A[i] = 0                        A[i] = 0
        i = i + 1                       i = i + 1
    A[i] = A[i] + 1                 A[i] = A[i] - 1
```

Assume that n operations, consisting of INCREMENT and DECREMENT operations, are performed on an initially zero counter. Show that the amortized cost of each operation is $O(1)$.

Problem 4 – TABLE-DELETE (Please read Section 17.4.2 of the textbook before you work on this problem.)

(10%) Show that if $\alpha^{i-1} \geq 1/2$ and the *i*-th operation on a dynamic table is TABLE-DELETE, then the amortized cost of the operation with respect to the potential

function (17.6 in the textbook) is bounded above by a constant.

Problem 5 – Making Binary Search Dynamic

Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays.

Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \lg(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We have k sorted arrays $A_0, A_1, \ldots, A_{k-1}$ where for $i = 0, 1, \ldots, k-1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, elements in different arrays have no particular relationship to each other.

1) SEARCH
   A) (5%) Describe how to perform the SEARCH operation for this data structure
   B) (5%) Derive its worst-case running time.
2) INSERT
   A) (5%) Describe how to perform the INSERT operation for this data structure and show that your algorithm has an amortized cost of $O(\lg n)$ per operation.
   B) (5%) Derive its worst-case running time.
3) DELETE
   A) (Bonus 5%) Describe how to perform the DELETE operation for this data structure, and derive its worst-case running time.