104-1 Final, CSIE, NTPU

Advanced Algorithms (高等演算法)

Date: 2016/01/12 Class: ID: Name:

1. (10%) Use dynamic programming to find a Longest Common Subsequence of the two sequences X = (A, B, C, B, D, A, B) and Y = (B, D, C, A, B, A).

2. (15%) Given 4 matrices A_1 (with dimension 30×35), A_2 (35×15), A_3 (15×5), A_4 (5×10), we want to compute the matrix-chain product $A_1A_2A_3A_4$. It is known that different ways to parenthesize the product may need different numbers of scalar multiplications.
   (a) List all possible ways in which we can parenthesize the product $A_1A_2A_3A_4$.
   (b) Find the optimal parenthesization to minimize the number of scalar multiplications.
   (c) Find the minimal number of scalar multiplications needed to compute the product $A_1A_2A_3A_4$.

3. (15%) Answer the following questions.
   (a) What is the difference between worst-case analysis and amortized analysis?
   (b) What is the main idea of **aggregate** method?
   (c) What is the main idea of **accounting** method?

4. (15%) A sequence of n operations is performed on a data structure. The i-th operation costs i if i is an exact power of 2, and 1 otherwise.
   (a) Use worst-case analysis to determine the worst-case cost of an operation. (5%)
   (b) Use **aggregate** method to determine the amortized cost per operation. (5%)
   (c) Use **accounting** method to determine the amortized cost per operation. (5%)

5. (15%) Consider a sequence of n INCREMENT operations on an initially zero counter with k binary bits.
   (a) Use worst-case analysis to determine the worst-case cost of an operation. (5%)
   (b) Use **aggregate** method to determine the amortized cost per operation. (5%)
   (c) Use **potential** method to determine the amortized cost per operation. (5%)

6. (15%) Design an algorithm for the following problem.

A Game on a Hexagram-Shaped Board

Description

Consider the following two-player board game played on a hexagram-shaped board, which consists of 13 cells, each identified by a power of 2, as shown in Figure 1.

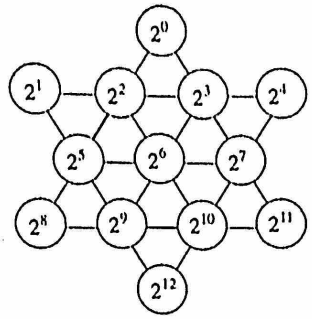


Figure 1: A hexagram-shaped board with 13 cells.

In the beginning of the game, each of some randomly selected M ($1 \leq M \leq 13$) cells are encircled by a ring. We call such a distribution of rings the *initial state* of a game. It appears that every possible state can be described by a distinct integer S, which is the sum of all numbers (i.e., powers of 2) identifying the encircled cells. For example, in Figure 2(a), the initial state of the board is described as $S = 2^3 + 2^5 + 2^{11} = 2088$. As another example, the board in Figure 2(b) is described as $2^7 + 2^9 + 2^{10} = 1664$.

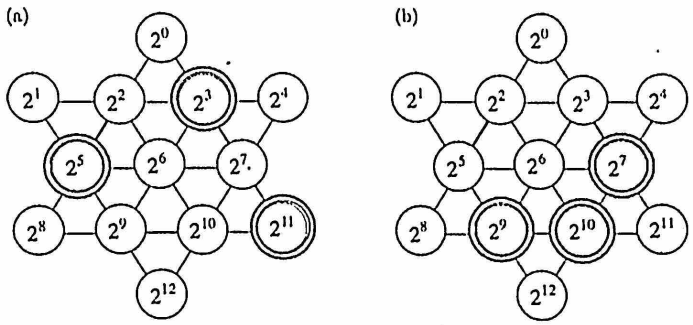


Figure 2: Two initial states of the game: (a) $S = 2^3 + 2^5 + 2^{11} = 2088$ (b) $S = 2^7 + 2^9 + 2^{10} = 1664$

Given the initial state of the board, the game is played with the following rules. The two players take turns removing rings from the board. On each turn, a player must remove one single ring or two *adjacent* rings. (Two rings are said to be adjacent if the cells they encircle are connected by an edge.) The player to remove the last ring *loses*.

It can be proved that either of the two players has a winning strategy in this game. That is, either the player who makes the first removal (hereinafter the *first* player) or the player who makes the second removal (hereinafter the *second* player) can always win, no matter how his/her opponent plays the game.

For example, consider the game started with the initial state in Figure 2(a). None of the rings is adjacent to another, forcing the players to remove a single ring on each turn. Therefore, the second player has a winning strategy since the first player must take the last ring with the other two rings removed.

Similarly, the winning strategy goes to the first player in Figure 2(b), where he/she can always win the game by removing two adjacent rings (e.g., 2^7 and 2^{10}).

Given the initial state of a board, your task is to write a program to determine whether the winning strategy goes to the first or the second player.

Input

The first line of the input contains an integer N, where $1 \leq N \leq 10$, indicating the number of test cases. Each of the following N lines contains a positive integer S, which represents the initial state of a board.

Output

For each case, your program should print either 1 (indicating that the first player has a winning strategy) or 0 (indicating that the second player has a winning strategy) as the answer. Leave a whitespace between cases.

Sampel Input 1

```
3
32
1024
1664
```

Sample Output 1

```
0 0 1
```

Sample Input 2

```
5
6
13
19
2088
2305
```

Sample Output 2

```
1 1 0 0 0
```

7. (15%) Design an algorithm for the following problem.

Squeeze the Cylinders

Input: Standard Input
Time Limit: 1 second

Laid on the flat ground in the stockyard are a number of heavy metal cylinders with (possibly) different diameters but with the same length. Their ends are aligned and their axes are oriented to exactly the same direction.

We'd like to minimize the area occupied. The cylinders are too heavy to lift up, although rolling them is not too difficult. So, we decided to push the cylinders with two high walls from both sides.

Your task is to compute the minimum possible distance between the two walls when cylinders are squeezed as much as possible. Cylinders and walls may touch one another. They cannot be lifted up from the ground, and thus their order cannot be altered.

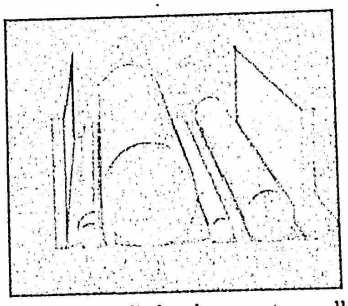


Figure B.1. Cylinders between two walls

Input

The input consists of a single test case. The first line has an integer N ($1 \leq N \leq 500$), which is the number of cylinders. The second line has N positive integers at most 10,000. They are the radii of cylinders from one side to the other.

Output

Print the distance between the two walls when they fully squeeze up the cylinders. The number should not contain an error greater than 0.0001.

Sample Input 1	Sample Output 1
2 10 10	40.00000000

Sample Input 2	Sample Output 2
2 4 12	20.06640046

Sample Input 3	Sample Output 3
5 1 10 1 10 1	40.00000000

Sample Input 4	Sample Output 4
3 1 1 1	0.00000000

Sample Input 5	Sample Output 5
2 5000 10000	29142.13562373

The following figures correspond to the Sample 1, 2, and 3.

The following figures correspond to the Sample 1, 2, and 3.

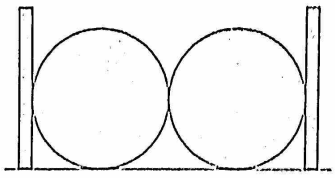


Figure B.2. Sample 1

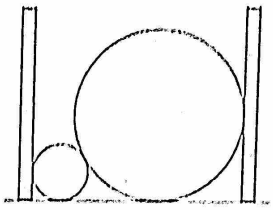


Figure B.3. Sample 2

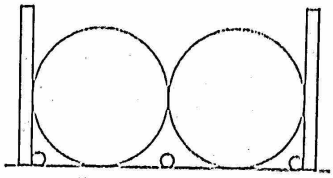
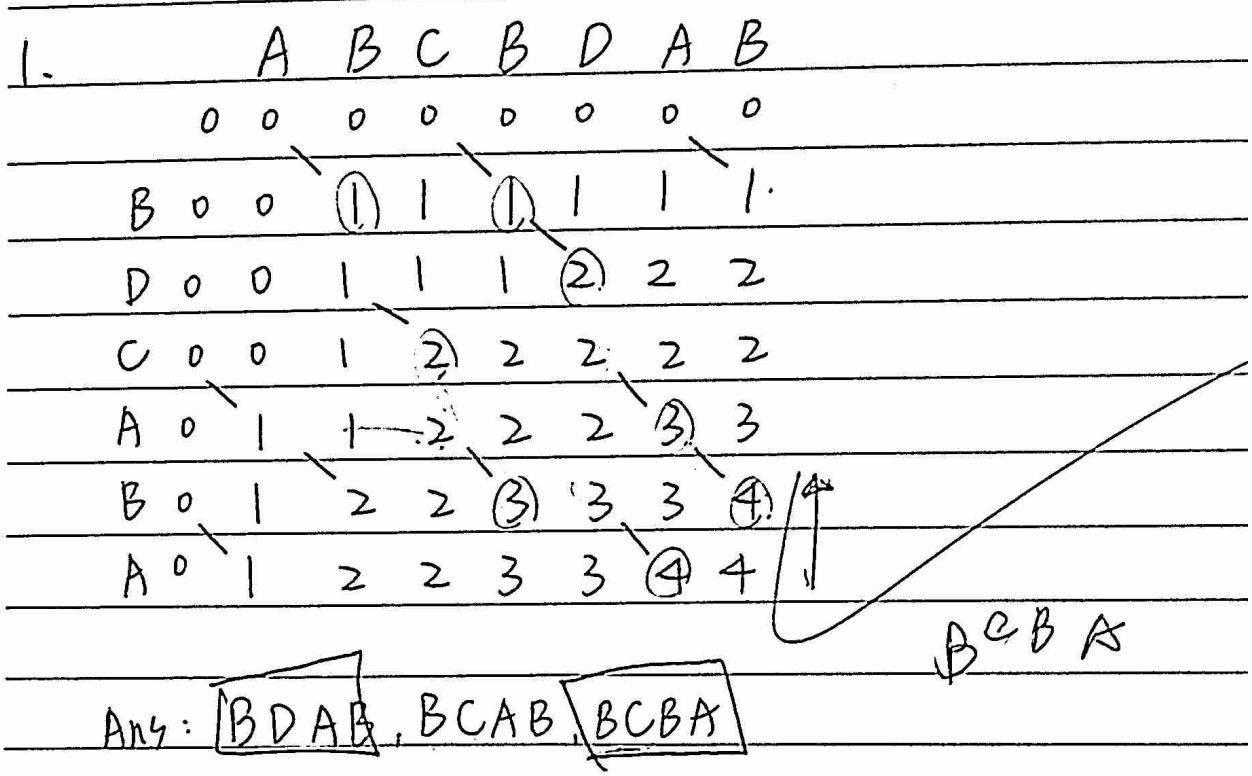


Figure B.4. Sample 3

Please fill out the above information, otherwise the

1.



Ans: BDAB, BCAB, BCBA

2. (a) ① $(((A_1A_2)A_3)A_4)$: $(30\times35\times15)+(30\times15\times5)+(30\times5\times10) = 5^3((6\times7\times3)+(6\times3\times1)+(6\times1\times2))$

② $((A_1A_2)(A_3A_4))$: $(30\times35\times15)+(15\times5\times10)+(30\times15\times10) = 5^3((6\times7\times3)+(3\times1\times2)+(6\times3\times2))$

③ $((A_1(A_2A_3))A_4)$: $(35\times15\times5)+(30\times35\times5)+(30\times5\times10) = 5^3((7\times3\times1)+(6\times7\times1)+(6\times1\times2))$

④ $(A_1((A_2A_3)A_4))$: $(35\times15\times5)+(35\times5\times10)+(30\times35\times10) = 5^3((7\times3\times1)+(7\times1\times2)+(6\times7\times2))$

⑤ $(A_1(A_2(A_3A_4)))$: $(15\times5\times10)+(35\times15\times10)+(30\times35\times10) = 5^3((3\times1\times2)+(7\times3\times2)+(6\times7\times2))$

(b) ① < ②

③ < ④

① > ③

⑤ > ③

→ $((A_1(A_2A_3))A_4)$ #

(c) $(35\times15\times5)+(30\times35\times5)+(30\times5\times10) = 9375$

3. (a) worst-case: A gaurantee but assually its estimation is higher than the reality.

amortized analysis: A gaurantee that has closed estimation to the reality.

(b) According to the change of the state, find the average cost of the worst-case estimation.

(c) Pay first, a credit that maintain ~~[illegible]~~ actually ≤ amortized

4 (a) If nth operation performed n structure, and if $n = 2^x$, $x \in Z^+$

Cost = $O(n)$

(b) If n has $(1+\lfloor \log n \rfloor)$ 1's 的 i 為 2 的指數, → Cost 2

Others cost 1 : $n-(1+\lfloor \log n \rfloor)$

i : 1 2 3 4 5 6 7 8 ... n Cost =

cost : 1 2 1 4 1 1 1 8 ... n or 1 → $2(1+\lfloor \log n \rfloor)+1(n-(1+\lfloor \log n \rfloor))$

$\frac{O(n)}{n}$: Cost = O(1)

(c) Amortized cost per op = 3 = O(1)

5. (a) A sequence n performed k binary bits to do increment.

One bits at most do one time. Cost = O(n)

(b) i : 1 2 3 4 5

cost : 0000 0...01 0...10 0...11 0...100

Cost = O(1)

(c) O(1)

```
int find(sum){
    for(i=1~sum){
        sum_component(i)
        table[i] = extract[i]
    }
}

int extract(sum){
    for(i = 0 ~ BOARDNUM-1){        // BOARDNUM = 13.
        if(component(i) ≠ 0){
            if(!table[sum_component(i)]) return FIRST.
            for(j = 0 ~ neighbor[i][j] ≠ -1){
                if(component(neighbor[i][j]) ≠ 0)
                    if(!table[sum_component(i)] - component(neighbor[i][j]))
                        return FIRST.
            }
        }
    return SECOND
```

```
7.  for (i = 2 ~ N)        // 從第2個開始.
        If (r(i)^2 < 4(r(i)·r(i-1)))        // 如果小球半徑夠大.
            distance = √(4·r(i)·r(i-1))
        else if (r(i-1)^2 < 4(r(i)·r(i-1)))        // 如果小球半徑夠大.
            distance = √(4·r(i)·r(i-1))
        else if (r(i)^2 > 4(r(i)·r(i-1)))        // 如果r(i)過大.
            distance = distance + r(i)
        else if (r(i-1)^2 > 4(r(i)·r(i-1)))        // 如果r(i)過小, r(i-1)過大.
            distance = distance + r(i-1)
    }
    If (r(2)^2 < 4(r(2)·r(1)))        // 如果第1個球夠大要算
        distance = distance + r(1)

    If (r(N)^2 < 4(r(N)·r(N-1)))        // 如果最後1個球夠大要算
        distance = distance + r(N)
```

no wall check

1