ADA-2024: Homework 2

Rachit Arora, 2022384
Mahi Mann, 2022272

February 10, 2024

Note: throughout this doc we'll be assuming that the array A is indexed from 1 to n.

1 Subproblem Definition

$$\begin{aligned}\text{MC(i, op, streak)} = &\text{ maximum score upto the i}^{\text{th}}\text{ index when } \textit{op} \text{ (op} \in \{\text{Ring, Ding}\}) \\ &\text{was said at the i}^{\text{th}} \text{ index, and the current streak of saying } \textit{op} \text{ is } \textit{streak}.\end{aligned}$$

2 Recurrence of the subproblem

Let's consider the base cases first.

$$\begin{aligned}\text{MC(i, op, streak)} &= (-\infty) && \text{(if i < streak)} \\ \text{MC(1, Ring, 1)} &= \text{ A[1]} \\ \text{MC(1, Ding, 1)} &= \text{(-A[1])}\end{aligned}$$

Note that we're not working with the absolute values of A[i] here. These base cases handle either sign, and so will the recurrence relations below.

Now we come to the actual recurrences:

$$\begin{aligned}&\text{MC(i, Ring, 1)} = \text{A[i]} + \max(\{ \text{ MC(i-1, Ding, s)} \mid \text{s } \in \{1, 2, 3\}\}) \\ &\text{MC(i, Ding, 1)} = \text{(-A[i])} + \max(\{ \text{ MC(i-1, Ring, s)} \mid \text{s } \in \{1, 2, 3\}\}) \\ &\text{MC(i, Ring, s)} = \text{A[i]} + \text{MC(i-1, Ring, s-1) (s} \in \{2, 3\}) \\ &\text{MC(i, Ding, s)} = \text{(-A[i])} + \text{MC(i-1, Ding, s-1) (s} \in \{2, 3\})\end{aligned}$$

Essentially, if the streak is 1, then the streak of the opposite saying ended at the last index. And if the streak is 2 or 3, the last index was forced to have the same saying.

An case-wise analysis for the border cases has been omitted for the sake of sanity. The actual implementation in code is much cleaner, please refer to pseudocode or C++ implementation.

3 Subproblem that solves the actual problem

Because of how we've defined the subproblem, there are 6 subproblems at index n. We simply take the maximum as follows:

$$\max(\{\ \text{MC(n, op, streak)}\ |\text{op} \in \{\text{Ring, Ding}\}, \text{streak} \in \{1, 2, 3\}\ \})$$

4 Algorithm Description

4.1 Idea

Pretty much described above. We'll have a 3-dimensional array of size 6n with all the answers. We'll start with the base cases, and index 3 onwards, compute all subproblems in index order.

4.2 Pseudocode

Here, we are assuming that $(-\infty)$ is an arbritrarily large negative integer bound

(Next page)

Algorithm 1 MaxChickens

```
function MaxChickens(A,n)
    MC ← array of dimension (n × 2 × 3)
    Ring ← 1
    Ding ← 2
    MC[1][Ring][2] ← (−∞)
    MC[1][Ring][3] ← (−∞)
    MC[1][Ding][2] ← (−∞)
    MC[1][Ding][3] ← (−∞)
    MC[1][Ring][1] ← A[1]
    MC[1][Ding][1] ← (-A[1])
    for i = 2, 3, ··· , n do
        for streak = 1, 2, 3 do
            MC[i][Ring][streak] ← (−∞)
            MC[i][Ding][streak] ← (−∞)
            if streak = 0 then
                for j = 1, 2, 3 do
                    if MC[i-1][Ding][j] ≠ (−∞) then
                        if MC[i-1][Ding][j] + A[i] > MC[i][Ring][1] then
                            MC[i][Ring][1] ← MC[i-1][Ding][j] + A[i]
                        end if
                    end if
                    if MC[i-1][Ring][j] ≠ (−∞) then
                        if MC[i-1][Ring][j] - A[i] > MC[i][Ding][1] then
                            MC[i][Ding][1] ← MC[i-1][Ring][j] - A[i]
                        end if
                    end if
                end for
            else
                if MC[i-1][Ring][streak-1] ≠ (−∞) then
                    MC[i][Ring][streak] ← MC[i-1][Ring][streak-1] + A[i]
                end if
                if MC[i-1][Ding][streak-1] ≠ (−∞) then
                    MC[i][Ding][streak] ← MC[i-1][Ding][streak-1] - A[i]
                end if
            end if
        end for
    end for
    ans ← (−∞)
    for i ∈ {Ring, Ding} do
        for s ∈ {1, 2, 3} do ans ← max(ans, MC[n][i][s])
        end for
    end for
    return ans
end function
```

5 Running time

For every index i, there are at most $c \geq 6$ constant-time addition and comparison operations (for some natural number c), so

$$T(i) \leq T(i-1) + c$$

We will now prove that $T(n) = O(n)$ by induction on n.

Base case: at $n = 1$, two values were initialised, so $2 \leq c$ addition and comparison operations.

Inductive Hypothesis: $T(k) \leq ck$ for some $k \in \mathbb{Z}$.

Inductive Step:

$$\begin{aligned} T(k+1) &\leq T(k) + c \\ &\leq ck + c && \text{(by Inductive Hypothesis)} \\ &= O(k+1) \end{aligned}$$

Therefore, our algorithm runs in $O(n)$ time (linear in n).

6 Addendum 1: C++ implementation

```cpp
#include <bits/stdc++.h>

using namespace std;

#define RING 0
#define DING 1

int main(){

    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

    int A[n];
    for(int i = 0; i < n; ++i){
        cin >> A[i];
    }
```

```
int MC[n][2][3];

MC[0][RING][1] = MC[0][DING][1] = MC[0][RING][2] = MC[0][DING][2] = INT_MIN;

MC[0][RING][0] = A[0];
MC[0][DING][0] = (-A[0]);

for(int i = 1; i < n; ++i){

    for(int streak = 0; streak < 3; ++streak){

        MC[i][RING][streak] = MC[i][DING][streak] = INT_MIN;

        if(streak == 0){

            for(int j = 0; j < 3; ++j){

                if(MC[i-1][DING][j] != INT_MIN)
                    MC[i][RING][0] = max(MC[i][RING][0], A[i] + MC[i-1][DING][j]);

                if(MC[i-1][RING][j] != INT_MIN)
                    MC[i][DING][0] = max(MC[i][DING][0], MC[i-1][RING][j] - A[i]);
            }

        }else{

            if(MC[i-1][RING][streak-1] != INT_MIN)
                MC[i][RING][streak] = MC[i-1][RING][streak-1] + A[i];

            if(MC[i-1][DING][streak-1] != INT_MIN)
                MC[i][DING][streak] = MC[i-1][DING][streak-1] - A[i];

        }
    }

}

int ans = INT_MIN;

for(int i = RING; i <= DING; ++i){
    for(int s = 0; s < 3; ++s){
        ans = max(ans, MC[n-1][i][s]);
    }
}
```

```
    cout << ans << '\n';
}
```

7 Addendum 2: Tests

I ran these tests on my C++ implementation:

```
Input:
8
1 2 3 4 -1 -2 -3 -4
Output:
14
Comments:
DING RING RING RING DING RING DING DING

Input:
8
-1 -2 -3 -4 1 2 3 4
Output:
14
Comments:
Negative of an array gives the same answer as expected (invert the sayings)

Input:
100000
1 1 1 1 .... (100000 ones)
Output:
50000
Comments:
RING RING RING DING (repeat this pattern)
```

Also, here's a timed test of size 200000 to demonstrate how fast this algorithm is:

```
$ g++ a.cpp
$ time python -c "t=200000;print(str(t)+' 1'*t)" | ./a.out
100000
python -c "t=200000;print(str(t)+' 1'*t)"  0.01s user 0.01s system 70% cpu 0.022 total
./a.out  0.02s user 0.00s system 69% cpu 0.034 total
```