MATH 333 Discrete Mathematics
Chapter 7 Recursion and Sum

Jianan Lin

linj21@rpi.edu

May 24, 2022

Recursion

We can define a number sequence in different ways.

Method 1: Write the formula

$$a_n = n^3 + n^2 + n + 1$$

Method 2: Write the base case and recursion

$$a_n = \begin{cases} 1, & n = 1 \\ 2 \cdot a_{n-1}, & n > 1 \end{cases}$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ f(n-2) + 2, & n > 2 \end{cases}$$

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ f(n-2) + 2, & n > 2 \end{cases}$$

Answer: $f(n) = n$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ 10 \cdot f(n-1) + 1, & n > 1 \end{cases}$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ 10 \cdot f(n-1) + 9, & n > 1 \end{cases}$$

Answer: We have

$$\begin{aligned} f(n) + 1 &= 10 \cdot (f(n-1) + 1) \\ &= 100 \cdot (f(n-2) + 1) \\ &= ... \\ &= 10^{n-1} \cdot (f(1) + 1) \\ &= 2 \cdot 10^{n-1} \end{aligned}$$

Therefore

$$f(n) = 2 \cdot 10^{n-1} - 1$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ -f(n-1) + 1, & n > 1 \end{cases}$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ -f(n-1)+1, & n > 1 \end{cases}$$

Answer: $f(n) = 0.5 - 0.5 \cdot (-1)^n$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ \dfrac{n}{n-1} f(n-1) + n, & n > 1 \end{cases}$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ \dfrac{n}{n-1} f(n-1) + n, & n > 1 \end{cases}$$

Answer: We have

$$\begin{aligned} \frac{f(n)}{n} &= \frac{f(n-1)}{n-1} + 1 = \frac{f(n-2)}{n-2} + 2 \\ &= \frac{f(1)}{1} + n - 1 = n \end{aligned}$$

Therefore

$$f(n) = n^2$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ \dfrac{10n}{n-1} f(n-1) + n, & n > 1 \end{cases}$$

Recursion

We write $f(n)$ as follows and what is the formula of $f(n)$?

$$f(n) = \begin{cases} 1, & n = 1 \\ \dfrac{10n}{n-1} f(n-1) + n, & n > 1 \end{cases}$$

Answer: Let $g(n) = f(n)/n$, we have

$$\frac{f(n)}{n} = 10 \cdot \frac{f(n-1)}{n-1} + 1 \Longrightarrow g(n) = 10g(n-1) + 1$$

So we have

$$g(n) + \frac{1}{9} = 10\left(g(n-1) + \frac{1}{9}\right) = \ldots = 10^{n-1}\left(g(1) + \frac{1}{9}\right) = \frac{10^n}{9}$$

Recursion

Let $x_1 = 1$ and $x_{n+1} = \sqrt{1 + 2x_n}$ for any $n \geq 1$. Prove $x_n \leq 4$.

Hint: Induction proof

Recursion

Let $x_1 = 1$ and $x_{n+1} = \sqrt{1 + 2x_n}$ for any $n \geq 1$. Prove $x_n \leq 4$.

Proof.

Base Case: $x_1 = 1 \leq 4$.

Suppose $x_k \leq 4$, then $x_{n+1} = \sqrt{1 + 2x_n} \leq \sqrt{9} = 3 \leq 4$.

Given $T_{n+2} = 3 \cdot T_{n+1} - 2 \cdot T_n$, calculate T_n.

Recursion

Given $T_{n+2} = 3 \cdot T_{n+1} - 2 \cdot T_n$ and $T_1 = 1, T_2 = 2$, calculate T_n.

Answer. We have

$$\begin{aligned} T_{n+2} - T_{n+1} &= 2 \cdot (T_{n+1} - T_n) = ... \\ &= 2^n \cdot (T_2 - T_1) = 2^n \end{aligned}$$

We can write the sequence: $1, 2, 4, 8...$ and guess $T_n = 2^{n-1}$.

This is easy to prove.

Recursion

Fibonacci Sequence: $0, 1, 1, 2, 3, 5, \ldots$. We have $F_n = F_{n-1} + F_{n-2}$. What is the formula?

This is very hard to get, so we introduce the method in linear algebra.

First we assume matrix is 2×2 so that

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = M \cdot \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix}$$

So we have

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = \begin{bmatrix} AF_{n-1} + CF_n \\ BF_{n-1} + DF_n \end{bmatrix}$$

According to the definition of Fibonacci sequence, we should let $A = 0, B = C = D = 1$.

Therefore we have

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = M \cdot \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix} = ... = M^n \cdot \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}$$

We use eigen-decomposition. And write it as

$$\begin{bmatrix} F_n \\ F_{n+1} \end{bmatrix} = P \cdot D^n \cdot P^{-1} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}$$

Next we calculate the eigen value of M:

$$|M - \lambda I| = \begin{bmatrix} -\lambda & 1 \\ 1 & 1-\lambda \end{bmatrix} = \lambda^2 - \lambda - 1 = 0$$

Solve it and we have

$$\lambda = \frac{1 \pm \sqrt{5}}{2}$$

The corresponding vectors are

$$\vec{v} = \begin{bmatrix} 1 \\ \frac{1 \pm \sqrt{5}}{2} \end{bmatrix}$$

Therefore we have

$$P = \begin{bmatrix} 1 & 1 \\ \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{1-\sqrt{5}}{2} & -\frac{1+\sqrt{5}}{2} \\ -1 & 1 \end{bmatrix}^{\top} = \begin{bmatrix} \frac{\sqrt{5}-1}{2\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{\sqrt{5}+1}{2\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}^{\top}$$

Last we have

$$F_n = (1,0) \cdot \begin{bmatrix} 1 & 1 \\ \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \end{bmatrix} \cdot \begin{bmatrix} \left(\frac{1+\sqrt{5}}{2}\right)^n & 0 \\ 0 & \left(\frac{1-\sqrt{5}}{2}\right)^n \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

But this is too hard. Do we have some easier way? The answer is yes!

Recursion

Suppose $T_n = AT_{n-1} + BT_{n-2}$, then we have the equation (without proof):

$$r^n = Ar^{n-1} + Br^{n-2} \Longrightarrow r^2 - Ar - B = 0$$

Solve it, we have

$$r = \frac{A \pm \sqrt{A^2 + 4B}}{2}$$

If $A^2 + 4B > 0$, then assuming the solution is λ_1, λ_2, we have

$$T_n = C \cdot \lambda_1^n + D \cdot \lambda_2^2$$

Here, C and D can be solved by considering $n = 1, n = 2$.

If $A^2 + 4B = 0$, then assuming the solution is λ, then

$$T_n = (C + Dn) \cdot \lambda^n$$

Recursion

Let $T_n = T_{n-1} + 2T_{n-2}$ and $T_1 = 0, T_2 = 1$. What is T_n?

Let $T_n = T_{n-1} + 2T_{n-2}$ and $T_1 = 0, T_2 = 1$. What is T_n?

According to the theorem, we have

$$r = \frac{A \pm \sqrt{A^2 + 4B}}{2} = \frac{1 \pm 3}{2} = 2 \text{ or } -1$$

Therefore $T_n = C \cdot 2^n + D \cdot (-1)^n$.

Consider $T_0 = C + D = 0$ and $T_1 = 2C - D = 1$, we have $C = 1/3, D = -1/3$. Therefore

$$T_n = \frac{1}{3} \cdot 2^n - \frac{1}{3} \cdot (-1)^n$$

Recursion

Let $T_n = 2T_{n-1} - T_{n-2}$ and $T_1 = 0, T_2 = 1$. What is T_n?

Recursion

Let $T_n = 2T_{n-1} - T_{n-2}$ and $T_1 = 0, T_2 = 1$. What is T_n?

According to the theorem, we have

$$r = \frac{A \pm \sqrt{A^2 + 4B}}{2} = \frac{2}{2} = 1$$

Therefore $T_n = (C + Dn) \cdot 1^n = C + Dn$.

Consider $T_0 = C + D = 0$ and $T_1 = C + 2D = 1$, we have $C = -1, D = 1$. Therefore

$$T_n = n - 1$$

Sum

Compute

$$\sum_{i=0}^{n} \left(2^i\right)^2$$

Sum

Compute

$$\sum_{i=0}^{n} \left(2^i\right)^2$$

Answer:

$$\begin{aligned}
\sum_{i=0}^{n} \left(2^i\right)^2 &= \sum_{i=0}^{n} 4^i \\
&= 1 + 4 + 4^2 + 4^3 + ... + 4^n \\
&= \frac{4^{n+1} - 1}{4 - 1} \\
&= \frac{4^{n+1} - 1}{3}
\end{aligned}$$

Sum

Compute

$$\sum_{i=0}^{n} i \cdot 2^i$$

Sum

Compute

$$\sum_{i=0}^{n} i \cdot 2^i$$

Answer:

$$\sum_{i=0}^{n} i \cdot 2^i = 0 + 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + ... + n \cdot 2^n$$

$$2\sum_{i=0}^{n} i \cdot 2^i = 0 \cdot 2^1 + 1 \cdot 2^2 + 2 \cdot 2^3 + 3 \cdot 2^4 + ... + n \cdot 2^{n+1}$$

Sum

Therefore

$$\begin{aligned}
\sum_{i=0}^{n} i \cdot 2^i &= 2\sum_{i=0}^{n} i \cdot 2^i - \sum_{i=0}^{n} i \cdot 2^i \\
&= n \cdot 2^{n+1} - (2 + 4 + ... + 2^n) \\
&= n \cdot 2^{n+1} - 2^{n+1} + 2 \\
&= (n-1) \cdot 2^{n+1} + 2
\end{aligned}$$

Sum

Compute

$$\sum_{i=0}^{n} i^2 \cdot 2^i$$

Sum

Compute

$$\sum_{i=0}^{n} i \cdot 2^i$$

Answer:

$$\sum_{i=0}^{n} i^2 \cdot 2^i = 0 + 1^2 \cdot 2^1 + 2^2 \cdot 2^2 + 3^2 \cdot 2^3 + ... + n^2 \cdot 2^n$$

$$2\sum_{i=0}^{n} i^2 \cdot 2^i = 0 \cdot 2^1 + 1^2 \cdot 2^2 + 2^2 \cdot 2^3 + 3^2 \cdot 2^4 + ... + n^2 \cdot 2^{n+1}$$

Sum

Therefore

$$
\begin{aligned}
\sum_{i=0}^{n} i^2 \cdot 2^i &= 2\sum_{i=0}^{n} i^2 \cdot 2^i - \sum_{i=0}^{n} i^2 \cdot 2^i \\
&= n^2 \cdot 2^{n+1} - \sum_{i=1}^{n} (i^2 - (i-1)^2) \cdot 2^i \\
&= n^2 \cdot 2^{n+1} - \sum_{i=1}^{n} (2i-1) \cdot 2^i \\
&= n^2 \cdot 2^{n+1} - 2\sum_{i=1}^{n} i \cdot 2^i + \sum_{i=1}^{n} 2^i \\
&= n^2 \cdot 2^{n+1} - 2 \cdot \left((n-1) \cdot 2^{n+1} + 2\right) + (2^{n+1} - 2) \\
&= (n^2 - 2n + 3) \cdot 2^{n+1} - 6
\end{aligned}
$$

Sum

In fact, we can have the following formula:

$$\sum_{i=0}^{n} \left(A + Bi + Ci^2 + Di^3 + ...\right) k^i$$
$$=(A' + B'n + C'n^2 + D'n^3 + ...)k^{n+1} + c$$

We can calculate A', B'... according to A, B...

Sum

Compute

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(i+j)$$

Sum

Compute

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(i+j)$$

Answer: we have

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{m}(i+j) &= \sum_{i=1}^{n}\sum_{j=1}^{m} i + \sum_{i=1}^{n}\sum_{j=1}^{m} j \\
&= m \cdot \sum_{i=1}^{n} i + n \cdot \sum_{j=1}^{m} j \\
&= \frac{mn(n+1)}{2} + \frac{nm(m+1)}{2} \\
&= \frac{mn(m+n+2)}{2}
\end{aligned}$$

Sum

Compute

$$\sum_{i=1}^{n}\sum_{j=1}^{i}(i+j)$$

Sum

Compute

$$\sum_{i=1}^{n}\sum_{j=1}^{i}(i+j)$$

Answer: we have

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{i}(i+j) &= \sum_{i=1}^{n}\sum_{j=1}^{i} i + \sum_{i=1}^{n}\sum_{j=1}^{i} j \\
&= \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} \frac{i(i+1)}{2} \\
&= \frac{3}{2}\cdot\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{4} \\
&= \frac{n(n+1)^2}{2}
\end{aligned}$$

Sum

Compute

$$\sum_{i=0}^{n} \sum_{j=0}^{i} 2^{i+j}$$

Sum

Compute

$$\sum_{i=0}^{n}\sum_{j=0}^{i} 2^{i+j}$$

Answer:

$$\begin{aligned}\sum_{i=0}^{n}\sum_{j=0}^{i} 2^{i+j} &= \sum_{i=0}^{n}\sum_{j=0}^{i} 2^{i}\cdot 2^{j}\\ &= \sum_{i=0}^{n}\left(2^{i}\cdot\sum_{j=0}^{i} 2^{j}\right)\\ &= \sum_{i=0}^{n}\left(2^{i}\cdot(2^{i+1}-1)\right)\end{aligned}$$

Sum

$$
\begin{aligned}
\sum_{i=0}^{n}\sum_{j=0}^{i} 2^{i+j} &= \sum_{i=0}^{n}\left(2^i \cdot (2^{i+1} - 1)\right) \\
&= 2 \cdot \sum_{i=0}^{n} 4^i - \sum_{i=0}^{n} 2^i \\
&= \frac{2(4^{n+1} - 1)}{4 - 1} - 2^{n+1} + 1 \\
&= \frac{2}{3} \cdot 4^{n+1} - 2^{n+1} + \frac{1}{3}
\end{aligned}
$$

Dynamic Programming

Dynamic Programming is similar to recursion and it is used in algorithm design.

In dynamic programming, we have known some value, and we use these values to get new values.

The simplest example: Fibonacci sequence. What is the $n-$th number?

We can calculate one by one: 0, 1, 1, 2, 3, 5, 8, ...

What if we design the algorithm in recursion? Let $F(n) = F(n-1) + F(n-2)$ and until $n = 1$?

We will meet a huge time complexity because $F(i)$ will be calculated many times!

Dynamic Programming

Consider this problem: I can choose city A or B to work each week. For week i, I will earn A_i, B_i in these two cities respectively. But if I choose another city some week, I have to pay for the transportation fee c which is fixed. How much money can I have for the first $n > 1$ weeks?

Can we use the simple idea: Choose the city with higher salary each week?

The answer is no! Try to give a counter-example.

Dynamic Programming

Problem: I can choose city A or B to work each week. For week i, I will earn A_i, B_i in these two cities respectively. But if I choose another city some week, I have to pay for the transportation fee c which is fixed. How much money can I have for the first $n > 1$ weeks?

Answer. Use x_i to denote if I work in city A in week i, the most money I can have. And use y_i to denote if I work in city B in week i, the most money I can have.

Obviously we have $X_1 = A_1, Y_1 = B_1$. Here we suppose we do not pay the transportation for the first week (it does not matter). And we have the relation as follows:

$$X_i = A_i + \max\left(X_{i-1}, Y_{i-1} - c\right)$$
$$Y_i = B_i + \max\left(Y_{i-1}, X_{i-1} - c\right)$$

For the last week supposed n, we return $\max(X_n, Y_n)$.

Problem: We have some vertices in order: $v_1 \to v_{\to} ... \to v_n$. Each vertex is worth c_i money. We want to pick the vertices with the most money. However, we cannot pick the adjacent vertices. This means that: if we pick v_i, then we cannot pick v_{i-1} and v_{i+1} anymore. Write an algorithm to calculate the largest money we can get.

Hint: Dynamic Programming

Dynamic Programming

Problem: We have some vertices in order: $v_1 \to v_{\to} ... \to v_n$. Each vertex is worth c_i money. We want to pick the vertices with the most money. However, we cannot pick the adjacent vertices. This means that: if we pick v_i, then we cannot pick v_{i-1} and v_{i+1} anymore. Write an algorithm to calculate the largest money we can get.

Answer: Let X_i denote the largest amount of money if I pick vertex v_i as the last vertex in the sequence.

Obviously, $X_1 = c_1$ because we can only pick one. In the same way, we have $X_2 = \max(c_1, c_2)$ and $X_3 = \max(c_1 + c_3, c_2)$. For X_i with $i \geq 4$, we have

$$X_i = c_i + \max\left(X_{i-2}, X_{i-3}\right)$$

We only need to return $\max(X_n, X_{n-1})$ at last.

Dynamic Programming

We have finite coins worth $1, 2, 4, 5$. Given an arbitrary number (positive integer), what is the minimum number of coins we need to denote it? Write an algorithm.

Can we use the following idea? First use 5, then 4, then 2, last 1.

Dynamic Programming

We have finite coins worth $1, 2, 4, 5$. Given an arbitrary number (positive integer), what is the minimum number of coins we need to denote it? Write an $O(n)$ algorithm.

Answer: Use X_i to denote the least number of coins we need to denote number i.

Obviously, we have $X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1, X_5 = 1$. Then

$$X_i = 1 + \min(X_{i-1}, X_{i-2}, X_{i-4}, X_{i-5})$$

In this way we only need to return X_n for a given n.

We have a sequence of number $a_1, ..., a_n$. All the numbers are real number and it can be positive, negative or 0. We want find a consecutive sequence $a_i, a_{i+1}, ..., a_{i+j}$ so that it has the largest sum. Design an $O(n)$ algorithm.

Dynamic Programming

We have a sequence of number $a_1, ..., a_n$. All the numbers are real number and it can be positive, negative or 0. We want find a consecutive sequence $a_i, a_{i+1}, ..., a_{i+j}$ so that it has the largest sum. Design an $O(n)$ algorithm.

Answer: Let X_i be the largest sum of consecutive sequences ending in a_i. Obviously, $X_1 = a_1$. Then we have

$$X_i = \max\left(X_{i-1}, 0\right) + a_i$$

Finally, we only need to return

$$\max_{i \in [1,n]} \left(X_i\right)$$

Dynamic Programming

We have a sequence of number $a_1, ..., a_n$. All the numbers are real number and it can be positive, negative or 0. We want find the longest sequence (not necessarily consecutive) $a_i, a_j, ..., a_k$ so that it is increasing. Design an $O(n^2)$ algorithm.

Dynamic Programming

We have a sequence of number $a_1, ..., a_n$. All the numbers are real number and it can be positive, negative or 0. We want find the longest sequence (not necessarily consecutive) $a_i, a_j, ..., a_k$ so that it is increasing. Design an $O(n^2)$ algorithm.

Answer: Let X_i be the longest length of sequence ending with a_i. Obviously we have $X_1 = 1$. Then we have

$$X_i = 1 + \max_{1 \le j < i, a_i > a_j} (X_j)$$

Finally we only need to return

$$\max_{i \in [1,n]} (X_i)$$

Dynamic Programming

We want to move n steps. Each time we can either move 1 or 2 steps. How many different ways we can choose to move n steps?

Dynamic Programming

We want to move n steps. Each time we can either move 1 or 2 steps. How many different ways we can choose to move n steps?

Answer: Let X_i be the number of different ways to move to i. Obviously $X_1 = 1, X_2 = 2$. Then we have

$$X_i = X_{i-1} + X_{i-2}$$

In fact, this is Fibonacci sequence. Finally we only need to return X_n.

Dynamic Programming

We have a matrix with coordinate $(0,0),(0,n),(m,0),(m,n)$. Here $m,n \geq 1$. A robot start from $(0,0)$ and wants to move to (m,n). It can only stays in integer point (e.g. $(1,2)$), and it can only move towards right or top. How many different ways are there? Design an $O(mn)$ algorithm.

Dynamic Programming

We have a matrix with coordinate $(0,0),(0,n),(m,0),(m,n)$. Here $m,n \geq 1$. A robot start from $(0,0)$ and wants to move to (m,n). It can only stays in integer point (e.g. $(1,2)$), and it can only move towards right or top. How many different ways are there? Design an $O(mn)$ algorithm.

Answer: Let $X_{i,j}$ be the number of ways from $(0,0)$ to (i,j). Obviously we have $X_{0,j} = X_{i,0} = 1$ for any i,j. Then we have

$$X_{i,j} = X_{i-1,j} + X_{i,j-1}$$

Finally we return $X_{m,n}$

Dynamic Programming

We have a matrix with coordinate $(0, 0), (0, n), (n, 0), (n, n)$. Here $m, n \geq 1$. A robot start from $(0, 0)$ and wants to move to (m, n). It can only stays in integer point (i, j) ** with $i \leq j$ ** (e.g. $(1, 2)$), and it can only move towards right or top. How many different ways are there? Design an $O(n^2)$ algorithm.

Dynamic Programming

We have a matrix with coordinate $(0, 0), (0, n), (n, 0), (n, n)$. Here $m, n \geq 1$. A robot start from $(0, 0)$ and wants to move to (m, n). It can only stays in integer point (i, j) ** with $i \leq j$ ** (e.g. $(1, 2)$), and it can only move towards right or top. How many different ways are there? Design an $O(n^2)$ algorithm.

Answer: Let $X_{i,j}$ be the number of ways from $(0, 0)$ to (i, j). Obviously we have $X_{0,j} = 1$ for any j and $X_{i,0} = 0$ for any i. Then we have

$$X_{i,j} = \begin{cases} 0, & \text{if } i > j \\ X_{i-1,j} + X_{i,j-1} & \text{if } i \leq j \end{cases}$$

Finally we return $X_{n,n}$

Dynamic Programming

I can watch TV drama or go to play basketball per evening in n days. I will have c fun (fixed) while playing basketball. TV drama is different per day, so it will give me a_i fun for day i. But if I play basketball, I will be tired tomorrow and can only watch TV. What is the largest fun I can get after day n? Write an $O(n)$ algorithm.

Dynamic Programming

I can watch TV drama or go to play basketball per evening in n days. I will have c fun (fixed) while playing basketball. TV drama is different per day, so it will give me a_i fun for day i. But if I play basketball, I will be tired tomorrow and can only watch TV. What is the largest fun I can get after day n? Write an $O(n)$ algorithm.

Answer: Let A_i denote the largest fun from day 1 to i and I watch TV in day i. Let B_i denote the largest fun from day 1 to i and I play basketball in day i. Obviously, $A_1 = a_1$ and $B_1 = c$. Then we have

$$A_i = a_i + \max\left(A_{i-1}, B_{i-1}\right)$$
$$B_i = b_i + A_{i-1}$$

Finally we only need to return $\max(A_n, B_n)$.