University of New South Wales (UNSW)

Algorithm Design & Analysis

Formatif Submission

(R) Pocket Money

Submitted By:
Agam Singh
z5479697
2024/11/09 04:20

Tutor:
Jeremy Le

November 9, 2024

Pocket Money

Agam Singh z5479697

November 8, 2024

Pocket Money

(a)

We can determine the sub-problem by letting $OPT(i, p)$ represent the maximum pocket money obtainable from days 1 through i.

For $1 \leq i \leq n$, the recurrence relation is:

$$OPT(i) = \max(OPT(i-1), c_i + OPT(i-2))$$

This recurrence is correct because:

- At each day i, we have two choices:
  - Skip today, keeping the best solution up to yesterday $(OPT(i-1))$
  - Take today's money c_i and add it to the best solution that ends at least two days ago $(OPT(i-2))$
- This naturally enforces the consecutive day constraint, as taking money on day i means we must look back two days for our previous valid solution

The base case is:

- $OPT(0) = 0$, as no days means no money taken
- $OPT(1) = c_1$ as on the first day, we can only take that day's money

The **final answer** is found in $OPT(n)$, which represents the maximum money obtainable through all n days.

To compute this efficiently:

(a) Initialize OPT array of size $(n + 1)$

(b) Fill from day 1 up to n using the recurrence relation to fill each cell

(c) The final answer is at $OPT(n)$

Time Complexity: The time complexity is $O(n)$ as we compute one value for each of the n days, with $O(1)$ work per value. This is optimal as we must examine each day's value at least once.

(b)

The subproblem $OPT(i)$ is insufficient because it cannot track whether we've used our one-time consecutive day opportunity. To resolve this, we need an additional parameter r to track our remaining consecutive day opportunities, making our new subproblem $OPT(i, r)$.

(c)

For $1 \leq i \leq n$, the recurrence relation is:

$$OPT(i, r) = \max \begin{cases} OPT(i-1, r) & \text{skip day } i \\ c_i + OPT(i-2, r) & \text{take money normally} \\ c_i + OPT(i-1, r-1) & \text{take money consecutively if } r > 0 \end{cases}$$

This recurrence is correct because:

- At each day i, we have three choices:
  - Skip today, keeping both r opportunities ($OPT(i-1, r)$)
  - Take today normally, looking back two days and keeping r ($c_i + OPT(i-2, r)$)
  - Take today using a consecutive opportunity if available, looking back one day and using one opportunity ($c_i + OPT(i-1, r-1)$)
- When $r = 0$, the third option becomes invalid, naturally handling the case where we've used our opportunity

The base cases are:

- $OPT(0, r) = 0$ for any r
- $OPT(1, r) = c_1$ for any r

The **final answer** is found in $OPT(n, 1)$, representing the maximum money obtainable through all n days starting with one consecutive opportunity.

To compute this efficiently:

(a) Initialize OPT array of size $(n+1) \times 2$

(b) Fill from day 1 up to n, computing both r states for each day

(c) The final answer is at $OPT(n, 1)$

Time Complexity: The time complexity is $O(n)$ as we compute 2 states for each of the n days, with $O(1)$ work per state.

(d)

We can adapt our solution to handle k consecutive opportunities by simply extending r to range from 0 to k in our subproblem $OPT(i, r)$. The recurrence relation remains identical:

$$OPT(i,r) = \max \begin{cases} OPT(i-1,r) & \text{skip day i} \\ c_i + OPT(i-2,r) & \text{take money normally} \\ c_i + OPT(i-1,r-1) & \text{take money consecutively if } r > 0 \end{cases}$$

The base cases and computation method remain the same, but now we compute $k+1$ states for each day instead of just 2. The **final answer** is found in $OPT(n, k)$.

Time Complexity: The time complexity is $O(nk)$ as we compute $k+1$ states for each of the n days, with $O(1)$ work per state. This is optimal as we must examine each combination of day and remaining opportunities at least once.