Lecture 4

Gidon Rosalki

2024-11-28

1 Fibonacci numbers

The series $1, 1, 2, 3, 5, 8, \ldots$, defined recursively as $\begin{cases} a_1 = a_2 = 1 \\ a_n = a_{n-1} + a_{n-2} \end{cases}$

If instead of calculating in a tree, we calculate down the table, we only do $O(n)$ calculations, instead of $O(2^n)$

1.0.1 Schema for a solution - dynamic programming

1. Collections of sub problems
2. Writing a recursion formula
3. Define the table, and how to extract the solution
4. Analyse runtime
5. Proof of correctness of the recursion formula

1.0.2 Examples

Example 1 (Programming task board)**.** *Choose the programming problem for the n upcoming weeks. For every week, there are two choices, a hard task, and an easy task. For a difficult task, in the ith week, we are compensated with $h_i > 0$ money, and $l_i > 0$ for an easy task. For a hard task, we must first rest for a week, for which we receive no money. We shall assume that before beginning, we have rested for a week.*
Input: $\{(h_i, l_i)\}_{i=1}^n$
Output: $\{s_i\}_{i=1}^n$ *such that*

1. $\forall i \in [n]\ s_i \in \{0, l_i, h_i\}$
2. $\forall i \in [n] | s_i = h_i \implies s_{i-1} = 0$
3. *We have maximised* $\sum s_i$

	0	l	h
1	0	2	3
2	0	1	10
3	0	2	3
4	0	1	10

Table 1:

Solution. So the greedy algorithm will return 7 money, but the optimal solution returns 10.

1. **Sub problems:** We find the optimal return that can be made in i weeks where $1 \leq i \leq n$
2. **Recursion algorithm:** $M_i = M[i] = \begin{cases} 0, & \text{if } i = 0 \\ \max\{l_1, h_1\}, & \text{if } i = 1 \\ \max\{M[i-1] + l_i, M[i-2] + h_i\}, & \text{if } i > 1 \end{cases}$

3. **Define a table:** Table M of size $(n+1) \times 1$, we will fill from the 0th cell until the nth cell according to the recursion formula.
See table 2
At the time of filling the table, we will save for every cell the choice that was made, we will pass over the table from the end to the beginning, for the ith cell, if we chose an easy task, we will assign "easy" in the week i and continue the loop from $i-1$, if we chose a hard task, we will assigned "hard" for the week i, and "rest" for the week $i-1$, and will continue the loop from $i-2$.

4. **Runtime:** $O(n) \cdot O(1) = O(n)$

5. **Correctness proof:**

   (a) **Correctness:** done

   (b) **Optimality:** For every $0 \leq i \leq n$, $M[i]$ holds the optimal choice that we may make for the ith week. We shall prove this with induction on the order of filling the table.
   Basis: 0: 0 is the maximal element available in the week 0. 1: The maximal element is $\max(0, l_1, h_1)$, which is in fact $M[1]$
   We will assume correctness until the ith cell, and prove it for the ith cell.
   In the ith week, there are 3 options:

   i. Easy task: We will select l_i, there is no requirement on $i-1$, so we take the optimal choice until $i-1$, and this is $M[i-1]$, from the inductive assumption, and so have $M[i-1]+l_i$
   ii. Hard task: We will select h_1, which has a requirement on $i-1$ - rest. We will select 0 in $i-1$, and take the optimal choice to $i-2$, and this is $M[i-2]$, by the inductive assumption. So $M[i-2]+h_i$
   iii. Rest: This is never preferable to easy.

   so we get $\max(1,2,3) = M[i]$

0	0
1	3
2	$\max(M[1]+l_2=4, M[0]+h_2=10)=10$
3	$\max(M[2]+l_3=12, M[1]+h_3=6)=12$
4	$\max(M[3]+l_4=13, M[2]+h_3=20)=20$

Table 2:

□

Example 2 (Maximal substring)**.** *Finding the longest shared substring between two strings.*
Input: 2 strings, $X = x_1 dots x_n \wedge Y = y_1 \dots y_m$
Output: The maximal substring of X, Y*. Continuity is not required, but order is.*
Example problem: $X = ABCD \wedge Y = BDC$*, Output* $= BC$

Solution. 1. **Sub problems:** Find the maximal substring between $X[..i]$ and $Y[..j]$ for $i \in [n] \wedge j \in [m]$

2. **Recursion formula:** $f(i,j)$ 0 the length of the maximal substring of $X[..i] \wedge Y[..j]$.

$$f(i,j) = \begin{cases} 0, & \text{if } i = 0 \vee j = 0 \\ 1 + f(i-1, j-1), & \text{if } X_i = Y_j \\ \max\{f(i-1,j), f(i,j-1)\}, & \text{if } X_i \neq Y_j \end{cases}$$

□