topic 5 - Dynamic Programming   March 27.

- Subproblems - allows recursive constructure ...
  - efficiency: store some solved sub in table, reuse it.

① Activity Selection

- Instance: A list of activities a_i, $1 \le i \le n$ with starting times s_i, and finishing time f_i. No two activities can take place simultaneously.

- task: find a subset of compatible (兼容的) activitie of "Maximal total duration". 最长的活动时间

  - we start from sorting their finishing time into a non-decreasing (increasing) sequence. so that we assume $f_1 \le f_2 \le \dots \le f_n$.

    consider the activities up to ⓘ

defining subp:

  - for every $i \le n$, we solve the following sub-problems:

    sub-problem $P(i)$: find a subsequence ϑ_i of the sequence of the activity $S_i = \langle a_1, a_2 \dots a_i \rangle$.

    在这个列表里的任何元素

    Subsequence 是列表里的元素 ($\in$)

    ① ϑ_i consisting of non-overlaping activities 不重复

    ② ϑ_i ends with activity a_i.

    ③ ϑ_i is of maximal total duration among all subsequences of S_i which satisfies ① and ②.

    note: the role of ② is to simplify recursion.

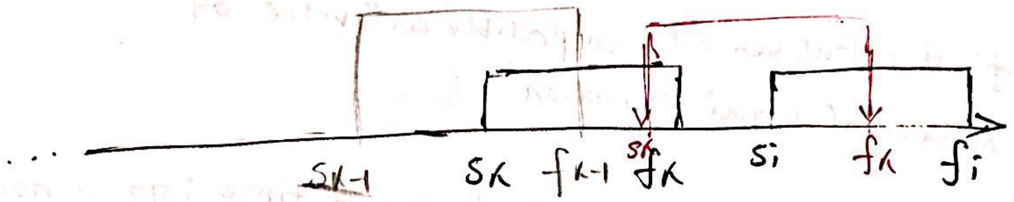
    the very last activity ⓘ must be included.

- Let $T(i)$ be the total duration of the optimal solution $S(i)$ of the subproblem $P(i)$. End with ⓘ + non-overlaping + longest duration.
- For $S(1)$ we choose a_1, then $T(1) = f_1 - s_1$.
- Recursion: Assuming that we have solved the subproblems for all $j < i$ and stored them in a table.

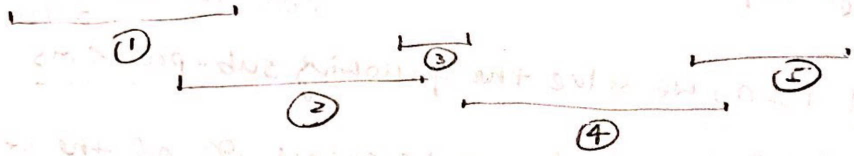
最后一个j的完成时间

$$T(i) = \max\{T(j) + (f_i - s_i) : j < i \ \& \ f_j < s_i\}$$ 在Si (i) 开始之前

j个活动的duration.　　compatible.



→ 假设已经将结束时间排好序(从小到大)



对于②，最好并不重复的结果是②. } 比较

③, ... ①③

④, ... ②④

⑤, ... ①③. ⇒ 对于⑤来说，上一个在⑤之前结束的是③.
所以选择包括③或之前的最好结果.

检测是否重复(compatible)
→ 如果是就是candidate
→ 在所有候选中选择最长duration的那个

- continuing with the solution of the problem, let

$$T_{max} = \max\{T(i),\ i \le n\}$$

- time complexity: · having sorted activities by their finishing time $O(n\log n)$

$n \times n$ { · examin n intervals. in roles of the last activity is an optimal sub-sequence.
· for each interval, need to find their optimal compatible solution.

$\therefore (n^2)$

② Longest Increasing Subsequence.

· given a sequence of n real numbers A[1 ... n], determine a subsequence (not necessary contiguous) of maximum length in which the values in the subsequence is strictly increasing.

? ⇒ For every real number in the given sequence.
- always contain the last real number ✓
- the increasing order
- End before i (maybe)
- choose the optimal from the ones smaller than i ✓

Eg. A[1 ... n] = 1, 17, 4, 31, 5, 11, 2, 6, 4, 12.

1 → 1
17 → 1, 17
4 → 1, 4
31 → (1, 17) or (1, 4), 31
5 → (1, 4), 5
11 → (1, 4, 5), 11
2 → (1, 2)
6 → 1, 4, 5, 6.
4 → 1, 4
12 → 1, 4, 5, 6, 12
(longest)

· Solution: For each i ≤ n we solve the following ~~subsequence~~ subproblems:

subproblem, P(i): find subsequence of sequence A[1 ... i] of maximum length in which the values are strictly increasing and which ends with A[i].

recursion: Assume we solved all the subproblems for j < i;

- we now look for all A[m] such that m < i and A[m] < A[i]

在数据比当前数据小的情况下选择最优解. 再加上当前数据.

· Among thoes, we pick m which produced the longest subsequence ending with A[m], and extend it with A[i] to obtain the longest increasing subsequence which ends with A[i].

optimal one. A[m].

· We store in the i^{th} slot of the table of length of the longest increasing subsequence ending with A[i]. and m such that the optimal solution for P(i) extends to optimal solution for P(m).

· so, we have found for every i ≤ n the longest increasing subsequence of the sequence A[1 ... i] which ends with A[i].

· time complexity : · for n element, find the longest increasing sub-seq and plus it on, takes O(n) time · x O(n)

· ~~For the last one, search~~

$\Rightarrow O(n^2)$: run "search ntimes" (n times).

③ Making change.

· you are given n types of coin ~~deter~~ denominations of values $v(1) < v(2) < \cdots < v(n)$, all integers.

· Assume $v(1) = 1$, so that you can always make change for any integer amount. Give an algorithm which makes change for any given integer amount C, with as few coins as possible. assume that you having an ultimated supply of coins of each denomination.

?· 1 7 9 6 10 12 given 30.

1 → 30 × 1 = 30 coins

7 → 4 × 7 + 3 × 1 = 7 coins.

9 → 3 × 9 + 3 × 1 = 6 coins.

6 → 5 × 6 = 5 coins.

10 → 3 × 10 = 3 coins.

12 → 2 × 12 + 1 × 6 = 3 coins.

(maybe) use the greedy method / calculate everyone ntimes.

replace the second best one by the best one ?

⇒ Search all between 1 and including S.

having i < S.

the solution for (i-1)

get / solve for (i+1).

sorted eg. 1 6 7 9 10 12 30 given.

6 → contains (1 × 6) or (6).

7 → contains (1 × 7) or (1 + 6) or (7)

30 → contains. (1 × 30) (5 × 6). ⋮ get the fewest one.

... (...) or (9).

· Solution: DP recursion on amount C, we will fill a table containg C many slots, so the optimal solution for amount i is stored in slot i

· if C = 1, the solution is trivial: just one coin of denomination $v(1) = 1$.

· Assume we have found optimal solution for every amount $j < i$, and now want to find the optimal solution for amount i.

· We consider optimal solution opt $(i - v(K))$ for every amount of the form $i - v(K)$, where K ranges from 1 to n.

· Among all of these optimal solutions (which we find in table we are constructing recursively) we pick one which uses fewest number of coins, say this is $opt(i - v(m))$ for some m.

$1 \le m \le n$

· We obtain an optimal solution $opt(i)$ for amount i by adding to $opt(i - v(m))$ one coin of denomination $v(m)$.

1, 3, 5

Eg: $d(0) = 0$

to get 0 we need 0 coins.

$d(i) = \min\{d(i - v_j) + 1\}$

$(i - v_j) \geq 0$

the denomination of the j^{th} coin.

$d(1) = d(1-1) + 1 = d(0) + 1 = 0 + 1 = 1$

$d(2) = d(2-1) + 1 = d(1) + 1 = 1 + 1 = 2$.

$d(3) = d(3-1) + 1 = d(2) + 1 = 2 + 1 = 3$.

$d(3) = d(3-3) + 1 = d(0) + 1 = 1$

$d(3) = \min\{d(3-1) + 1, d(3-3) + 1\} = d(3-3) + 1$.

$d(4) = d(4-1) + 1 = d(3) + 1 = \cancel{3}\ 1 + 1 = 2$

always find the optimal solution.

$d(5) = d(5-1) + 1 = d(4) + 1 = 2 + 1 = 3$.

$d(5) = d(5-3) + 1 = d(2) + 1 = 2 + 1 = 3$.

$d(5) = d(5-5) + 1 = d(0) + 1 = 1$.

$d(5) = \min\{d(5-1) + 1, d(5-3) + 1, d(5-5) + 1\} = d(5-5) + 1$

- It is enough to store in the i^{th} slot of the table such m and opt (i) because this allows us to reconstruct the optimal solution by looking at m_1 stores in i^{th} slot, then look at m_2 stored in the slot $i - v(m_1)$ at the slot $i - v(m_1) - v(m_2)$, then then look at m_2 stored in the slot $i - v(m_1) - v(m_2)$, then

$$opt(i) = \min \{ opt(i - V_1), opt(i - V_2), opt(i - V_3) \cdots opt(i - V_K) \} + 1$$

```
Eg. int coin[3] = {1, 3, 5}
    int finding[12]
        find_coin (int num) .
            finding[0] = 0;
            for (int i=1 ; i ≤ num ; i++) {.
                finding[i] = 9999 ;
                for( int j=0 ; coin[j] ≤ i && j ≤ 3 ; j++ }.
                    if ( finding[i - coin[j] + 1 < finding [i] ).
                        finding[i] = finding[i - coin[j]] + 1
        ...
    }
    }
```

→ 硬币的面值小于等于输入值.