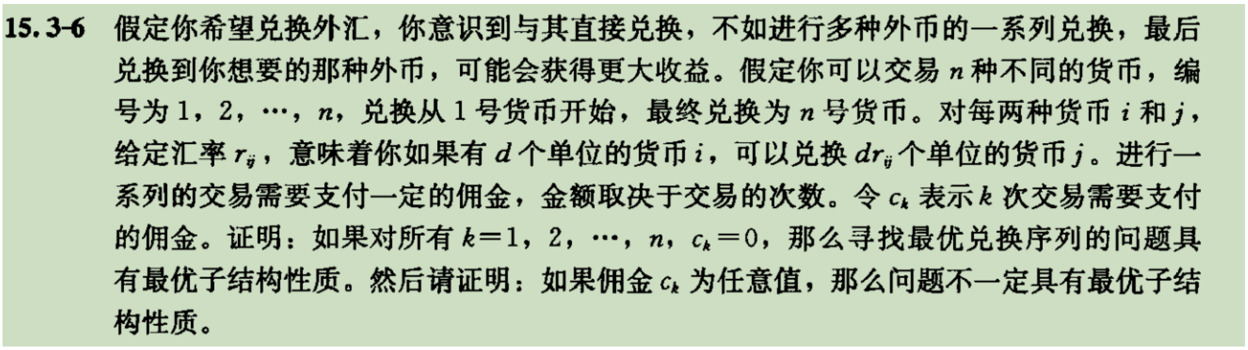
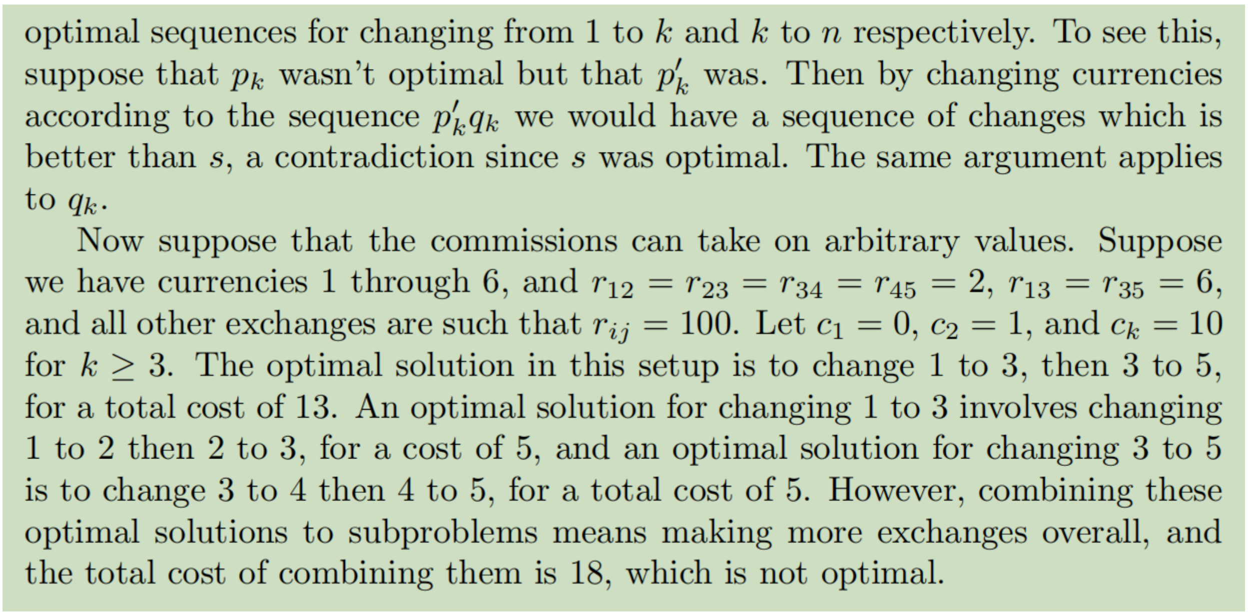
# PS13--动态规划

## 一、无最优子结构

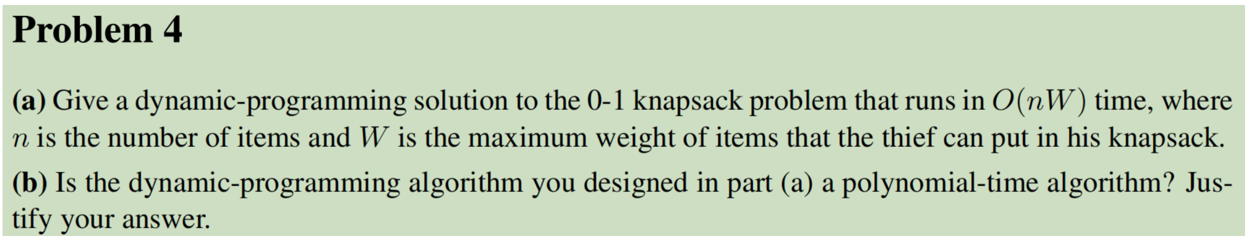

**15.3-6** 假定你希望兑换外汇，你意识到与其直接兑换，不如进行多种外币的一系列兑换，最后兑换到你想要的那种外币，可能会获得更大收益。假定你可以交易 $n$ 种不同的货币，编号为 1，2，…，$n$，兑换从 1 号货币开始，最终兑换为 $n$ 号货币。对每两种货币 $i$ 和 $j$，给定汇率 $r_{ij}$，意味着你如果有 $d$ 个单位的货币 $i$，可以兑换 $dr_{ij}$ 个单位的货币 $j$。进行一系列的交易需要支付一定的佣金，金额取决于交易的次数。令 $c_k$ 表示 $k$ 次交易需要支付的佣金。证明：如果对所有 $k=1$，2，…，$n$，$c_k=0$，那么寻找最优兑换序列的问题具有最优子结构性质。然后请证明：如果佣金 $c_k$ 为任意值，那么问题不一定具有最优子结构性质。

首先把一段最优解分成p和q两个小段


optimal sequences for changing from 1 to $k$ and $k$ to $n$ respectively. To see this, suppose that $p_k$ wasn't optimal but that $p'_k$ was. Then by changing currencies according to the sequence $p'_k q_k$ we would have a sequence of changes which is better than $s$, a contradiction since $s$ was optimal. The same argument applies to $q_k$.

Now suppose that the commissions can take on arbitrary values. Suppose we have currencies 1 through 6, and $r_{12} = r_{23} = r_{34} = r_{45} = 2$, $r_{13} = r_{35} = 6$, and all other exchanges are such that $r_{ij} = 100$. Let $c_1 = 0$, $c_2 = 1$, and $c_k = 10$ for $k \geq 3$. The optimal solution in this setup is to change 1 to 3, then 3 to 5, for a total cost of 13. An optimal solution for changing 1 to 3 involves changing 1 to 2 then 2 to 3, for a cost of 5, and an optimal solution for changing 3 to 5 is to change 3 to 4 then 4 to 5, for a total cost of 5. However, combining these optimal solutions to subproblems means making more exchanges overall, and the total cost of combining them is 18, which is not optimal.

## 二、0-1背包


**Problem 4**

**(a)** Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $O(nW)$ time, where $n$ is the number of items and $W$ is the maximum weight of items that the thief can put in his knapsack.

**(b)** Is the dynamic-programming algorithm you designed in part (a) a polynomial-time algorithm? Justify your answer.

## 4 Problem 4

**(a)** To index the subproblem, we will give the first $k$ items from the item list and the maximum weight of current knapsack. There can be at most $O(nW)$ of these subproblem. For each item, we can decide to put it in or not. If we put the $i$st item in, we need to prepare $w_i$ weight for it and earn $v_i$ value. The value is $v_i$ plus the solution to the subproblem with the first $i-1$ items, and $W - w_i$ capacity. Otherwise, the value is as same as the solution to the subproblem with the first $i-1$ items, and $W$ capacity. We take the maximum value from the two choices for each item.

**(b)** No since $W$ can increase exponential as input length.

# 三、树

**Problem 2**

This problem asks you to devise algorithms to compute optimal subtrees in *unrooted* trees—connected acyclic undirected graphs. In this problem. a *subtree* of an unrooted tree is any connected subgraph.

**(a)** Suppose you are given an unrooted tree $T$ with weights on its *edges*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a *path* in $T$ with maximum total weight. Your algorithm only need to return the weight value. You may assume a path only contains distinct vertices. For full credit, your algorithm should have $O(n)$ runtime assuming $T$ contains $n$ vertices.

**(b)** Suppose you are given an unrooted tree $T$ with weights on its *vertices*, which may be positive, negative, or zero. Describe and analyze an algorithm to find a subtree of $T$ with maximum total weight. Your algorithm only need to return the weight value. For full credit, your algorithm should have $O(n)$ runtime assuming $T$ contains $n$ vertices. (This was a 2016 Google interview question.)

*(We only ask for returning the weight value so as to keep the code cleaner. Nonetheless, you should also think about how to efficiently reconstruct the optimal solution. It's another good exercise!)*

- 任选结点作为根
- 向下DFS

First choose an arbitrary vertex $r \in T$ as the root of $T$ and use $DFS$ to find the parent of u $u.p$ and the children of $u$ for each $u \in T$. Suppose the parent if the root $r.p = -1$. This problem uses recursion on tree. Let the weight of edge $(u, v)$ be $w(u, v)$. Let the weight of vertex $u$ be $w(u)$.

**(a)** Algorithm overview: we let $dp[u]$ for each $u$ in $T$ be the path with the max weight from $u$ down to the leaf (not required to get to one of the leaves). The algorithm first complete $dp[u]$ us recursion on tree and find $\max(dp[v_1] + w(u, v_1), 0) + \max(0, dp[v_2] + w(u, v_2))$ for each $u \in T$ and $v_1, v_2$ are two of $u$'s children with the max value of $dp[v_2] + w(u, v_2)$.

**Algorithm:** Run $DFS(r)$ to compute $dp$. $DFS(u)$ for $u \in T$ is defined as following:

- If $u$ has no children ($u$ is a leaf), then let $dp[u] = 0$.
- Else, for each child $v$ of $u$, call $DFS(v)$. Let $v_m$ be the child of $u$ with the max $dp[v_m]+w(u, v_m)$. Let $dp[u] = \max(0, dp[v_m] + w(u, v_m))$.

For each $u \in T$, compute $dp'[u]$ as following:

- $max_1 = -\infty, max_2 = -\infty$.
- For each child $v$ of $u$,
  - If $dp[v] + w(u, v) > max_1$, then $max_2 \leftarrow max_1$ and $max_1 \leftarrow dp[v] + w(u, v)$.
  - Else, if $dp[v] + w(u, v) > max_2$, then $max_2 \leftarrow dp[v] + w(u, v)$.
- Let $dp'[u]$ be $\max(max_1, 0) + \max(max_2, 0)$.

The final answer is the following value

$$\max_{u \in T} dp'[u]$$

## 四、正反字符串

### Problem 3

Describe and analyze an efficient algorithm to find the length of the longest *contiguous* substring that appears both forward and backward in an input string $T[1 \cdots n]$. The forward and backward substrings must *not* overlap. Here are several examples:

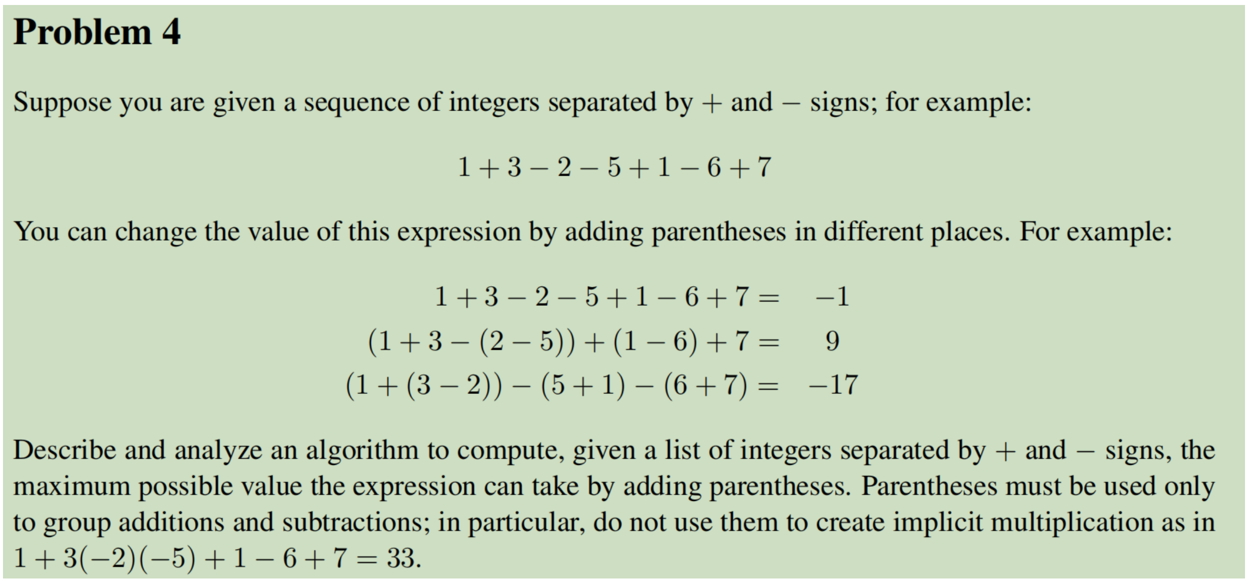
- Given the input string `ALGORITHM`, your algorithm should return 0.
- Given the input string `RECURSION`, your algorithm should return 1, for the substring `R`.

- Given the input string `REDIVIDE`, your algorithm should return 3, for the substring `EDI`. (The forward and backward substrings must not overlap!)
- Given the input string `DYNAMICPROGRAMMINGMANYTIMES`, your algorithm should return 4, for the substring `YNAM`. (In particular, it should not return 6, for the subsequence `YNAMIR`).

- 三维数组A[i][j][len]，表示从第i到第j项里是否存在长度为len的符合要求的正反字符串
- 首先初始化
- 则A[i][j][len]=0，若T[i]!=T[j]
- A[i][j][len]=A[i+1][j-1][len-1]，若T[i]==T[j]
- 时间O（n^3）

> O（n）时间去掉所有只出现一次的字符
> 用max保存最大值，初始为1，t_max存储当前最大值
> 从头开始i=1检索匹配，从后往前寻找，维持下标不重复
> 相同，则匹配，查找下一项，++，更新max，t_max
> 遇到下一项不相同，t_max=1，从i+1再开始，从后往前匹配
> i=n结束，返回max
> O（nnn）

## 五、加括号



**Problem 4**

Suppose you are given a sequence of integers separated by + and − signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$\begin{aligned} 1 + 3 - 2 - 5 + 1 - 6 + 7 &= -1 \\ (1 + 3 - (2 - 5)) + (1 - 6) + 7 &= 9 \\ (1 + (3 - 2)) - (5 + 1) - (6 + 7) &= -17 \end{aligned}$$

Describe and analyze an algorithm to compute, given a list of integers separated by + and − signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$.

- Amax[i][j]与Amin[i][j]分别记录从i到j项所能达到的最大值和最小值

https://courses.engr.illinois.edu/cs374/sp2017/labs/solutions/lab8bis-sol.pdf

### 一个待证明的贪心想法

- 以下操作中，所有被括号括起来的，直接计算，结果当作一个新数字替代原来的式子
- 任意连续的加号可以被分割成两个子问题。也可变成一个加数，无连续加号

> 尽量让每个减号后的括号内值为负

- 多个连续减号-a-b-c，-（a-b-c）。变成一个减号，无连续减号
- 此时变成加减号交错的式子a-b+c-d+e-f+g
- 从后往前，对于每个先加后减的，如+e-f：若f>e，则变成a-b+c-(d+e-f)+g。e>=f，则变成+

(e-f+g)。此时仍为一加一减的形式，且等价的式子减少了两个符号

- 最后一定可以变成一个大括号或A-B的形式，即为最大