算法基础 HW6

PB18111697 王章瀚

2020 年 11 月 24 日

1

我们对钢条切割问题进行一点修改，除了切割下的钢条段具有不同价格p_i外，每次切割还要付出固定的成本c．这样，切割方案的收益就等于钢条段的价格之和减去切割的成本．设计一个动态规划算法解决修改后的钢条切割问题．

Algorithm 1 考虑切割成本的钢条切割问题

Require: 长度为 i 的钢条价格为 p_i, 每次切割有固定成本 c, 钢管长度为 n

```
function BOTTOM-UP-CUT-ROD(p, c)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 → n do
        q = −∞
        for i = 1 → j do
            q = max(q, p[i] + r[j − i] − c)        ▷ 固定成本 c 在这里被考虑
        r[j]=q
    return r[n]
```

2

给定一个有向无环图 $G = (V; E)$，边权重为实数，给定图中两个顶点 s 和 t. 设计动态规划算法，求从 s 到 t 的最长加权简单路径.

这里我们显然有: 如果某结点 v 是该路径上的, 那么沿着这个路径, s 到 v 和 v 到 t 也分别是最长加权简单路径. 否则一旦有更长的, 换上去就行了.

这是因为这个更长$v \rightsquigarrow_{new} t$ (或 $s \rightsquigarrow_{new} v$)的显然不会经过 $s \rightsquigarrow v$ (或$v \rightsquigarrow t$)上的顶点.3

假设经过, 比如是 $s \rightsquigarrow_{new} v$ 经过了 $v \rightsquigarrow t$ 上的顶点ω, 那么就有$\omega \rightsquigarrow_{new} v$ 和 $v \rightsquigarrow \omega$, 这样一来就构成了环, 与题设矛盾. 另一种情况同理.

故确实可以利用上述思想做动态规划.

Algorithm 2 有向无环图的最长加权简单路径

```
Require: 给定有向无环图G = (V, E), 两个顶点s, t
function LONGEST-WEIGHTED-PATH(V, E, s, t)
    for v in V do
        v.val = −∞
        v.last = NULL
    s.val = 0
    let S = {s}
    while S is not ∅ do
        for v in S do
            S = S/{v}
            for w in v.to_list do
                if w.val < edge(v, w).weight + v.val then
                    w.val = edge(v, w).weight + v.val
                    w.from = v
                S = S ∪ {w}
    let path = a list
    v = t
    while visnotNULL do
        path.insert(v, path.begin)                    ▷ 把 v 插入列表首部
        v = v.last
    return path
```

3

在最优二叉搜索树问题中，若给定7个关键字的概率如下所示，求其最优二叉搜索树的结构和代价.(p，q定义和课本相同)

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

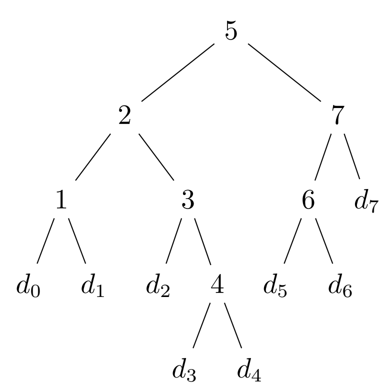
首先可以画出以下三个表:

w \ i / j	1	2	3	4	5	6	7	8
7	1.0	0.90	0.78	0.64	0.56	0.41	0.24	0.05
6	0.81	0.71	0.59	0.45	0.37	0.22	0.05	
5	0.64	0.54	0.42	0.28	0.20	0.05		
4	0.49	0.39	0.27	0.13	0.05			
3	0.42	0.32	0.20	0.06				
2	0.28	0.18	0.06					
1	0.16	0.06						
0	0.06							

e \ i / j	1	2	3	4	5	6	7	8
7	3.12	2.61	2.12	1.55	1.20	0.78	0.34	0.05
6	2.44	1.96	1.48	1.01	0.72	0.32	0.05	
5	1.83	1.41	1.04	0.57	0.30	0.05		
4	1.34	0.93	0.57	0.24	0.05			
3	1.02	0.68	0.32	0.06				
2	0.62	0.30	0.06					
1	0.28	0.06						
0	0.06							

root \ i / j	1	2	3	4	5	6	7
7	5	5	5	6	6	7	7
6	3	5	5	5	6	6	
5	3	3	4	5	5		
4	2	3	3	4			
3	2	3	3				
2	2	2					
1	1						

从而能够得出最优二叉搜索树:



其代价即为 $e[1,7] = 3.12$

4

一位公司主席正在向Stewart教授咨询公司聚会方案．公司的内部结构关系是层次化的，即员工按主管-下属关系构成一棵树，根结点为公司主席．人事部按"宴会交际能力"为每个员工打分，分值为实数．为了使所有参加聚会的员工都感到愉快，主席不希望员工及其直接主管同时出席．

公司主席向Stewart教授提供公司结构树，采用左孩子右兄弟表示法(参见课本10.4节)描述． 每个节点除了保存指针外，还保存员工的名字和宴会交际评分．设计算法，求宴会交际评分之和最大的宾客名单．分析算法复杂度．

采用动态规划的思想, 如果最优方案选中了 x, 那么其所有孙子(因为孩子不会被选)的选择情况将也是最优的, 否则通过替换即可得到更优解.

因此如果令 $r[x]$ 表示以 x 为根的子树能够构成的最佳宴会名单对应的评分和, 那么就有

$$\text{若不邀请x: } r_0[x] = \max\begin{cases}\sum\limits_{\text{y in x.children}} r_0[y] \\ \sum\limits_{\text{y in x.grandchildren}} r_0[y]\end{cases}$$

$$\text{若邀请x: } r_1[x] = x.score + \sum_{\text{y in x.grandchildren}} r_0[y]$$

因此可以写出相应算法[1]:

其通过自上而下递归遍历结点, 且若遇到已经算完的情况就能自动返回(动态规划的关键), 并且最后计算出名单只需要遍历一遍, 因此**复杂度是** $\Theta(n)$.

具体算法见下页

[1]参考自`https://blog.csdn.net/yangtzhou/article/details/84305563`

Algorithm 3 宴会邀请

Require: 员工结构树 T, 返回显示是否被邀请的表 $guests$

function BEST-INVITATION(T) ▷ 此后, 把 $guest$ 中被邀请者提取即为名单
    r_0, r_1 为结点到 r 值的表, 其初始值均为 $-\infty$
    $guest$ 指示是否被邀请, 初值为 $False$
    BEST-INVITATION($T, r_0, r_1, guests$)
    if $r_0[T] > r_1[T]$ **then**
        $guests[T] = False$
        return $r_0[T], guests$
    else
        $guests[T] = True$
        return $r_1[T], guests$

function BEST-INVITATION-SUB($T, \&r_0, \&r_1, \&guests$)
    if T is NULL **then**
        return
    if $r_0[T]! = \infty$ **then** ▷ 已经算过
        return
    children = T.children
    $r_0[T] = 0$
    $r_1[T] = T.score$
    for x in children **do**
        BEST-INVITATION($x, r_0, r_1, guests$)
        if $r_0[x] > r_1[x]$ **then** ▷ 不邀请 T 的情况下, 邀请了 x 能得到更好的结果
            $r_0[T] += r_0[x]$
        else ▷ 不邀请 T 的情况下, 不邀请 x 能得到更好的结果
            $r_0[T] += r_1[x]$
        $r_1[T] += T.score + r_0[x]$ ▷ 邀请 T 的情况下, 只要直接加上不邀请孩子 x 的 r 值即可
    if $r_0[T] > r_1[T]$ **then**
        $guests[T] = False$
    else
        $guests[T] = True$

5

设计一个高效的算法，对实数线上给定的一个点集$\{x_1, x_2, \cdots, x_n\}$，求一个单位长度闭区间的集合，包含所有给定的点，并要求此集合最小．证明你的算法是正确的．

首先给出算法:

1. 将点集排序得到列表 X, 初始化区间集合为 $sections = \varnothing$
2. 取 X 中最小元 x_i, 并取区间 $s = [x_i, x_i + 1]$
   将 X 中落在区间 s 中的点去掉, 并令 $sections = sections \cup s$
3. 如果 $X \neq \varnothing$, 继续执行第 2 步, 否则进入 第4步
4. 此时区间集合 S 即为所求集合

现在证明其正确性:
首先显然如果区间左端点不是点集中的点之一, 那么完全可以使之增大直到达到的第一个点, 这样只可能减小区间集合大小.
其次, 在第一次进入第二步的时候, 不论如何一定要有一个区间包含第一个点x_1, 那么根据前述, 它必须是$[x_1, x_1 + 1]$, 此时显然可以不考虑落在该区间的点了. 然后重复地执行, 每次都是全局最优方案之选, 因此整个算法是正确的. $\square$

6

考虑用最少的硬币找n美分零钱的问题．假定每种硬币的面额都是整数． 设计贪心算法求解找零问题，假定有25美分，10美分，5美分和1美分四种面额的硬币．证明你的算法能找到最优解．

首先给出算法:

1. 先选$a = n/25$个25美分硬币
2. 再选$b = (n - 25 * a)/10$个10美分硬币
3. 再选$c = (n - 25 * a - 10 * b)/5$个5美分硬币
4. 再选$(n - 25 * a - 10 * b - 5 * c) = d$ 个1美分硬币
5. 最终得到结果: a个25美分硬币, b 个10美分硬币, c个5美分硬币, d个1美分硬币

下面证明算法的最优性: 如果不是最优, 假设 (a', b', c', d') 为一更优解,
若$a' < a$, 则$(0, b', c', d')$的价值必然至少比$(0, b, c, d)$的刚好大$25 * (a - a')$, 此时可以从中凑出 $(a - a')$个25, 显然这能够找出比(a', b', c', d')更小的结果.
依此类推, 若$b' < b$, 若$c' < c$, 若$d' < d$均有类似结论. 所以$a' \geq a$, $b' \geq b$, $c' \geq c$, $d' \geq d$, 因此(a, b, c, d)必然是最优解.
□