

贪心法(Greedy Algorithm)

袁征

yzh12282002@gmail.com

What we are to talk about

- Introduction
- An example to review
- What is Greedy Algorithm
- What makes Greedy Algorithm work
- More examples

Introduction

- Programming: data structure + algorithm
- CHOICES on the way to optimization.
- Mathematic tools to help to proof
- Now you learn HOW, learn WHY later.
- A greedy algorithm always makes the choice that looks best at the moment.

An activity-selection problem

- $S = \{a_1, a_2, \dots, a_n\}$ n proposed activities
 - a_i : s_i (start time); f_i (finish time) $[s_i, f_i)$
 - a_i & a_j compatible:
 - $[s_i, f_i) \& [s_j, f_j)$ do not overlap
- | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|----|----|----|----|----|
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| s_i | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| f_i | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

■ What we need?

- A greedy choice (only one subproblem)
- Subproblem: $S_{ij} = \{a_k \mid S: f_i \leq s_k < f_k \leq s_j\}$
- $f_m = \min\{f_k \mid a_k \in S_{ij}\}$
- 1. Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij} .
- 2. The subproblem S_{im} is empty, so that choosing a_m leaves the subproblem S_{mj} as the only one that may be non empty.

Recursive algorithm

- recursive-activity-selector(s, f, i, j)
- 1 $m \leftarrow i + 1$
- 2 while $m < j$ and $s_m < f_i$
- 3 do $m \leftarrow m + 1$
- 4 if $m < j$
- 5 then return $\{a_m\} \cup \text{recursive-activity-selector}(s, f, m, j)$
- 6 else return

Iterative greedy algorithm

- GREEDY-ACTIVITY-SELECTOR(s,f)
- 1 $n \leftarrow \text{length}[s]$
- 2 $A \leftarrow \{a_1\}$
- 3 $i \leftarrow 1$
- 4 for $m \leftarrow 2$ to n
- 5 do if $s_m \leq f_i$
- 6 then $A \leftarrow A \cup \{a_m\}$
- 7 $i \leftarrow m$
- 8 return A

What is Greedy Algorithm

- 1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- 2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
- 3. Demonstrate that what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

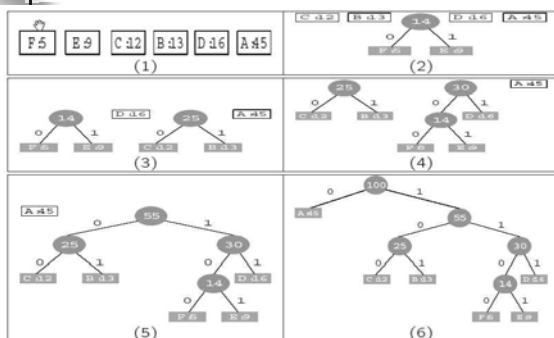
What makes Greedy Algorithm work

- Greedy-choice property(贪心选择性质)
 - 所求问题的最优解可以通过一系列局部最优解的选择, 即贪心选择达到
 - 在当前状态下做出局部最优选择, 然后解做出这个选择之后产生的相应子问题
 - 从全局来看, 运用贪心策略解决的问题在程序的运行过程中无回溯过程
- Optimal substructure(最优子结构性质)
 - 一个问题的最优解包含其子问题的最优解

Proof

- 方法
 - 考察问题的一个整体最优解, 并证明可修改这个最优解, 使其以贪心选择开始
 - 做出贪心选择之后, 原问题简化为规模更小的类似子问题: 利用该问题的最优子结构性质
 - 然后用数学归纳法证明

Huffman编码(3-1)



Huffman编码(3-2)

- HUFFMAN(C)
- 1 $n \leftarrow |C|$
- 2 $Q \leftarrow C$
- 3 for $i \leftarrow 1$ to $n - 1$
- 4 do allocate a new node z
- 5 $\text{left}[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$
- 6 $\text{right}[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 $f[z] \leftarrow f[x] + f[y]$
- 8 INSERT(Q, z)
- 9 return EXTRACT-MIN(Q)

Huffman编码(3-3)

■ 算法的正确性

■ 贪心选择性质

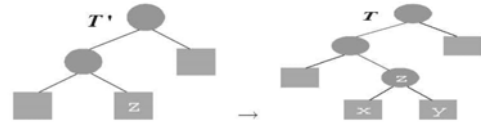
- 证明：设 C 是编码字符集， C 中字符 c 的频率是 $f(c)$ 。设 x, y 是 C 中具有最小频率的两个字符，存在 C 的最优前缀码使 x, y 具有相同码长且最后一位编码不同

■ 最优子结构性

- 证明：设 C 是编码字符集， C 中字符 c 的频率是 $f(c)$ 。设 x, y 是树 T 中的两个叶子结点且为兄弟， z 是它们的父结点。若将 z 看作是具有频率 $f(x)+f(y)$ 的字符，则树 $T' = T - \{x, y\}$ 表示字符集 $C' = C - \{x, y\} \cup \{z\}$ 的一个最优前缀码

附： Huffman编码最优子结构性(2-1)

- 证明：设 C 是编码字符集， C 中字符 c 的频率是 $f(c)$ 。设 x, y 是树 T 中的两个叶子结点且为兄弟， z 是它们的父结点。若将 z 看作是具有频率 $f(x)+f(y)$ 的字符，则树 $T' = T - \{x, y\}$ 表示字符集 $C' = C - \{x, y\} \cup \{z\}$ 的一个最优前缀码



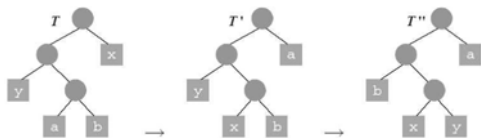
附： Huffman编码最优子结构性(2-2)

- 首先证明 T 的平均码长 $B(T)$ 可用 T' 的平均码长 $B(T')$ 表示。事实上，对 $c \in C - \{x, y\}$ ，有 $d_T(c) = d_{T'}(c)$ ，故 $f(c)d_T(c) = f(c)d_{T'}(c)$ 。另一方面， $d_T(x) = d_T(y) = d_{T'}(z) + 1$ ，故 $f(x)d_T(x) + f(y)d_T(y) = (f(x) + f(y))(d_{T'}(z) + 1) = f(z)d_{T'}(z) + (f(x) + f(y))$ ，由此知 $B(T) = B(T') + (f(x) + f(y))$ 。

若 T' 所表示的 C' 的前缀码不是最优的，则有 T'' 表示的 C' 的前缀码使得 $B(T'') < B(T')$ 。由于 z 被看作是 C' 中的一个字符，故 z 在 T'' 中是一树叶结点。若将 x 和 y 加入树 T'' 中作为 z 的儿子结点，则得到表示字符集 C 的二叉树 T''' ，且有 $B(T''') = B(T'') + f(x) + f(y) < B(T') + f(x) + f(y) = B(T)$ ，这与 T 的最优性矛盾。

附： Huffman编码贪心选择性质(2-1)

- 证明：设 C 是编码字符集， C 中字符 c 的频率是 $f(c)$ 。设 x, y 是 C 中具有最小频率的两个字符，存在 C 的最优前缀码使 x, y 具有相同码长且最后一位编码不同




附： Huffman编码贪心选择性质(2-2)

- 设 a, b 是 T 中最深叶子结点且为兄弟。不失一般性可设 $f[a] \leq f[b]$ 和 $f[x] \leq f[y]$ 。于是 $B(T) - B(T')$ 为

$$f[c]d_T(c) - f[c]d_{T'}(c) = f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) + f[a]d_{T'}(a) = f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) + f[a]d_T(x) = (f[a] - f[x])(d_T(a) - d_T(x)) \geq 0$$
 由于 $f[x] \leq f[a]$ 且 $d_T(x) \leq d_T(a)$ 。类似的，可得 $B(T'') \leq B(T')$ 。因此， T 最优树，且 x 和 y 为最深叶子结点且为兄弟。

Greedy versus dynamic programming

- The 0-1 knapsack problem
 - Dynamic programming
- The fractional knapsack problem
 - Greedy



knapsack problem

The i th item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers.

Select from n items as many as possible to put into the knapsack of W pounds to make the value as high as possible.



Thanks!