3-2 Greedy

Jun Ma

majun@nju.edu.cn

2020 年 9 月 24 日

TC 16.1-2

Suppose that instead of always selecting the **first** activity to finish, we instead select the **last** activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

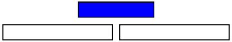
证明.

It is a dual problem. □

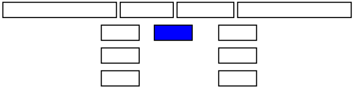
TC 16.1-3

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities.

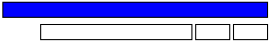
- selecting the activity of least duration from among those that are compatible with previously selected activities does not work.



- selecting the compatible activity that overlaps the fewest other remaining activities



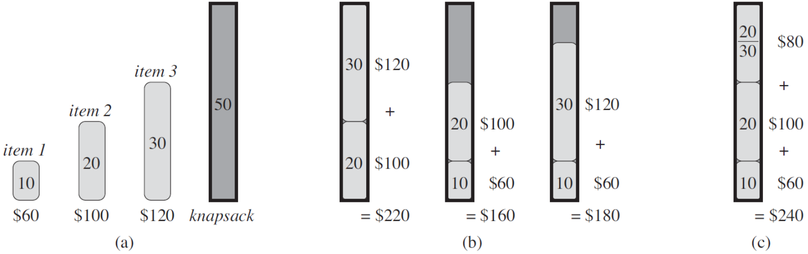
- selecting the compatible remaining activity with the earliest start time.



TC 16.2-1

Prove that the fractional knapsack problem has the greedy-choice property.

In the **fractional knapsack problem**, the thief can take fractions of items, rather than having to make a binary (0-1) choice for each item.



TC 16.2-1

证明.

- Let item $item_i$ be one with greatest v_i/w_i
- Given an optimal solution *opt*
  - if *opt* takes all $item_i$ or it only takes $item_i$, ✓
  - otherwise, let $W_{other} = W - u_i$, $w'_i = w_i - u_i$. Here, u_i indicates the units of $item_i$ in *opt*
    - replace $\min(W_{other}, w'_i)$ units of other items in *opt* with $item_i$.
    - after replacement, we should also obtain an optimal solution.

□

TC 16.2-2

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.

Answer.

Let $dp[i, w]$ record the maximum value can be obtained by taking only items from 1 to i without exceeding the weight limit w.

$$dp[i, w] = \max\left(dp[i-1, w], \max_{k<i}\left(dp[k, w - w_i] + p_i\right)\right)$$

□

背包问题九讲

背包问题九讲 2.0 beta1.2

崔添翼 (Tianyi Cui)*

2012-05-08†

本文题为《背包问题九讲》，从属于《动态规划的思考艺术》系列。

这系列文章的第一版于 2007 年下半年使用 EmacsMuse 制作，以 HTML 格式发布到网上，转载众多，有一定影响力。

2011 年 9 月，本系列文章由原作者用 LaTeX 重新制作并全面修订，您现在看到的是 2.0 beta 版本，修订历史及最新版本请访问 https://github.com/tianyicui/pack 查阅。

Contents

*a.k.a. dd_engi

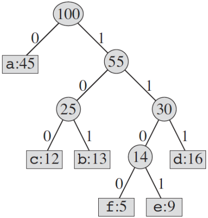
†Build 20120508120900

1

https://raw.githubusercontent.com/tianyicui/pack/master/V2.pdf

TC 16.3-2

Prove that a binary tree that is not full cannot correspond to an optimal prefix code.
Full Binary Tree: every nonleaf node has two children



(b)

证明.

Replace the node (that has only one child) with its subtree would yield a better solution. □

TC 16.3-5

Prove that if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing.

Hint

Monotonicity

A function $f(n)$ is ***monotonically increasing*** if $m \leq n$ implies $f(m) \leq f(n)$. Similarly, it is ***monotonically decreasing*** if $m \leq n$ implies $f(m) \geq f(n)$. A function $f(n)$ is ***strictly increasing*** if $m < n$ implies $f(m) < f(n)$ and ***strictly decreasing*** if $m < n$ implies $f(m) > f(n)$.

- $c_1.f \geq c_2.f \geq \cdots \geq c_n.f$
- T: an optimal code for $c_1, c_2, \cdots, c_n$
- Try to show $\forall i < j, d_T(c_i).l \leq d_T(c_j).l$
  - Can be easily proved by contradiction.

TC 16.3-8

Suppose that a data file contains a sequence of 8-bit characters such that all 256 characters are about **equally common**: *the maximum character frequency is less than twice the minimum character frequency.* Prove that Huffman coding in this case is no more efficient than using an ordinary 8-bit fixed-length code.

证明.

- $\forall i \forall j \forall k (c_i.freq + c_j.freq > c_k.freq)$
- So, there must be 128 internal nodes with 2 leaves
- $\forall i \forall j \forall k \forall l (1/2 < \frac{c_i.freq + c_j.freq}{c_k.freq + c_l.freq} < 2)$
- So, the maximum combined character frequency is less than twice the minimum combined character frequency.
- Recursively, we could show that the tree is a **perfect** binary tree.

□

TC Problem 16-1 (Coin changing)

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

Question-A

Describe a greedy algorithm to make change consisting of (**Q**)uarters, (**D**)imes, (**N**)ickels, and (**P**)ennies. Prove that your algorithm yields an optimal solution.

The Algorithm

- Always give the highest denomination coin that you can without going over.
- Then, repeat this process until the amount of remaining change drops to 0.

Optimal substructure

Naive

Greedy-choice property

An optimal solution to make change for n cents includes one coin of value c, where c is the largest coin value such that $c \leq n$.

Proof of greedy-choice.

- **Case 1**: If this optimal solution includes a coin of value c, then we are done.
- **Case 2** Otherwise, this optimal solution does not include a coin of value c. We have four cases:
  1. $n < 5$: A solution consists only of **P**s. (should be **Case 1**)
  2. $5 \leq n < 10$: Replace five **P**s by one **N**.
  3. $10 \leq n < 25$: Some subset of the **N**s and **P**s in this solution adds up to 10 cents, and so we can replace these **N**s and **P**s by a **D**.
  4. $25 \leq n$:
     - (a) **#dime**≥ 3: 3 dims$\rightarrow$ 1 quarter+1 nickel
     - (b) **#dim**≤ 2: some subset of the dimes, nickels, and pennies adds up to 25 cents, and so we can replace these coins by one **Q**.

□

Question-B

Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, \cdots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

Optimal substructure

- Naive!

Greedy-choice property

There is an optimal solution to make change for n cents includes one coin of value q, where $q = c^k$ is the largest coin value such that $q \leq n$.

引理 (1)

For any coin set T, if $c^k \leq \sum_{x \in T} = n < c^{k+1}$ and $\forall x \in T, x < c^k$, then, there is a subset S of T s.t. $\sum_{x \in S} = c^k$.

Proof of Lemma1.

Induction on k

- **B:** $k = 0$, then $T = \underbrace{\{1, \cdots, 1\}}_{n}$, which is obviously true
- **H:** for all $k < i$ the lemma holds
- **I:** $k = i + 1$
  - **partition** T into c subsets $T_1, \cdots T_c$ s.t. $c^i \leq \sum_{x \in T_j} < c^{i+1}$.
  - By **H**, for each T_j we could find a subset S_j s.t. $\sum_{x \in S_j} = c^i$.
  - We find a subset S of T s.t. $S = \bigcup_{j=1,\dots,c} S_j$ and $\sum_{x \in S} = c^{i+1}$

□

Greedy-choice property

There is an optimal solution to make change for n cents includes one coin of value q, where $q = c^k$ is the largest coin value such that $q \leq n$.

Proof by contradiction.

- Assume there is a n and all optimal solutions to make change for n cents do not include any coin of value q, where $q = c^k$ is the largest coin value such that $q \leq n$.
- We could find a subset S of T s.t. $\sum_{x \in S} = c^k$. (by Lemma(1))
- Replace S with a coin c^k would yield a better solution. (Conflict!)

□

Another proof

引理 (2)

Given an optimal solution $(x_0, x_1, \cdots, x_k)$ *where* x_i *indicates the number of coins of denomination* c^i*. Then we must have* $x_i < c$ *for every* $i < k$*.*

Proof of Lemma2.

- Suppose that we had some $x_i \geq c$, then, we could decrease x_i by c and increase x_{i+1} by 1.
- This collection of coins has the same value and has $c1$ fewer coins, so the original solution must of been non-optimal.

□

Greedy-choice property

There is an optimal solution to make change for n cents includes one coin of value q, where $q = c^k$ is the largest coin value such that $q \leq n$.

Proof by contradiction.

- Assume there is a n and all optimal solutions to make change for n cents do not include any coin of value q, where $q = c^k$ is the largest coin value such that $q \leq n$.
- However, as $x_i < c$ for every $i < k$ (by Lemma2), $\sum_{0 \leq i < k} x_i c^i < c^k \leq n$, Conflict!

$\square$

Question-C

Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

Answer

Try to make change for 8 cents with coins in $\{1, 4, 5\}$

Question-D

Give an $O(nk)$-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

Answer

- $c[i]$: the minimum number of coins needed to make change for i cents.

$$c[i] = \min_{1 \leq j \leq k} (c[i - v_j]) + 1$$


Thank You!