算法设计与分析

Lecture 7: Greedy Algorithms (贪心算法)

曹刘娟

厦门大学信息学院计算机系

caoliujuan@xmu.edu.cn

Greedy Algorithms

- Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step.
- A greedy algorithm (贪心算法) always makes the choice that looks best at the moment. 贪心算法一般只顾眼前
- Greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. 希望通过只顾眼前的这种方式来找到全局最优解
  - Don't think greedy approach is evil due to its name "greedy" with negative meaning. It often lead to very efficient and simple solution. 贪心算法其实很多时候非常高效
- Everyday examples:
  - Playing cards
  - Invest on stocks
  - Choose a university

ACTIVITY SELECTION PROBLEM

任务选择问题

Activity Selection Problem 任务选择问题

- Schedule n activities $S = \{a_1, \ldots, a_n\}$ that require use of a common resource. 假设有很多个预期的活动/任务要举行，但是这些任务要用一个公共的资源，每次只能举办一个任务。比如在厦大建南大会堂举办不同专业学生的毕业典礼。假设不同专业的典礼要独占大会堂
  - We can only do one activity at the same time.
- a_i needs resource during period $[s_i, f_i)$. 不同任务有不同的起止时间
  - s_i = start time and f_i = finish time of activity a_i.
  - $0 \leq s_i < f_i < \infty$
- Activities a_i and a_j are compatible (兼容的) if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap:

$f_i \leq s_j$ or $f_j \leq s_i$

Activity Selection Problem

- Select the largest possible set of mutually compatible (相互兼容) activities. 目的是找到一个任务的最大集合，这个集合中每个活动都是互相兼容的
  - We treat each activity gives same profit.
- For example,

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

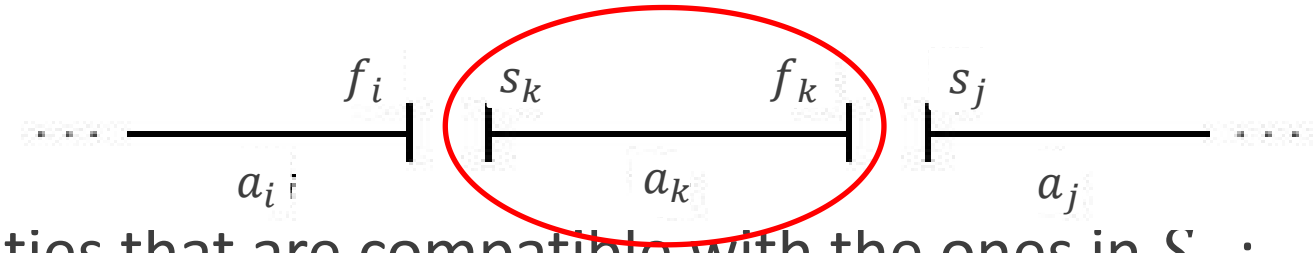
  - Activities are sorted in increasing order of finish times. 任务按照结束时间从小到大排序
  - A subset of mutually compatible activities: $\{a_3, a_9, a_{11}\}$.
  - Maximal set of mutually compatible activities: $\{a_1, a_4, a_8, a_{11}\}$ and $\{a_2, a_4, a_9, a_{11}\}$.

Representing the Problem 表达问题

- Define the subproblems:

$$S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$$

as activities whose periods are after a_i finishes and before a_j starts.
S_{ij}集合中的所有活动的时间都是在ai结束之后，在aj开始之前



- Activities that are compatible with the ones in S_{ij}:

与Sij相容的活动

  - All activities that finish before f_i. 在i结束之前结束的任务
  - All activities that start no earlier than s_j. 在j开始之后开始的任务

Representing the Problem表达问题

- Based on the definition of S_{ij}:

$$S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$$

S_{1n} does not cover the original problem because a_1 and a_n are excluded.

S_{1n}没有包括原问题的定义，因此添加虚拟任务a_0和a_{n+1}

We can add two fictitious (虚拟的) activities:

  - $a_0 = [-\infty, 0);\ a_{n+1} = [\infty, \infty + 1)$.

- Thus, $S_{0(n+1)} = S$ covers the entire space of activities a_1, …, a_n.
- $S_{0(n+1)}$ 包含了原问题的所有任务
- Assume that activities are sorted in increasing order of finish times 假设任务的截止时间是从小到大排序的。

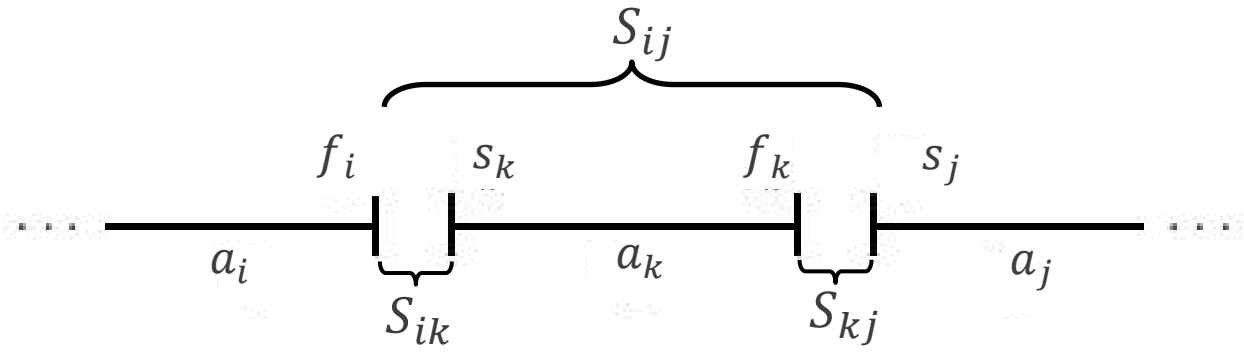
$$f_0 \leq f_1 \leq f_2 \leq \ldots \leq f_n < f_{n+1},$$

We only need to consider sets S_{ij} with $0 \leq i < j \leq n + 1$, because $S_{ij} = \emptyset$ if $i > j$.（可以用反证法证明）

若 存在ak满足$f_i \leq s_k < f_k \leq s_j < f_j$ 矛盾

Dynamic Programming Solution

- 首先我们用动态规划的方法来求解，看看是否有最优子结构
- Recall the optimal substructure of matrix-chain multiplication problem and optimal BST problem, the optimal subproblem has contains a position k.
- In this problem, assume that a solution to the above a subproblem includes activity a_k. 假设子问题的答案中包含a_k
  - Solution to S_{ij} = (Solution to S_{ik}) ∪ $\{a_k\}$ ∪ (Solution to S_{kj}).
  - |Solution to S_{ij}| = |Solution to S_{ik}| + 1 + |Solution to S_{kj}|.

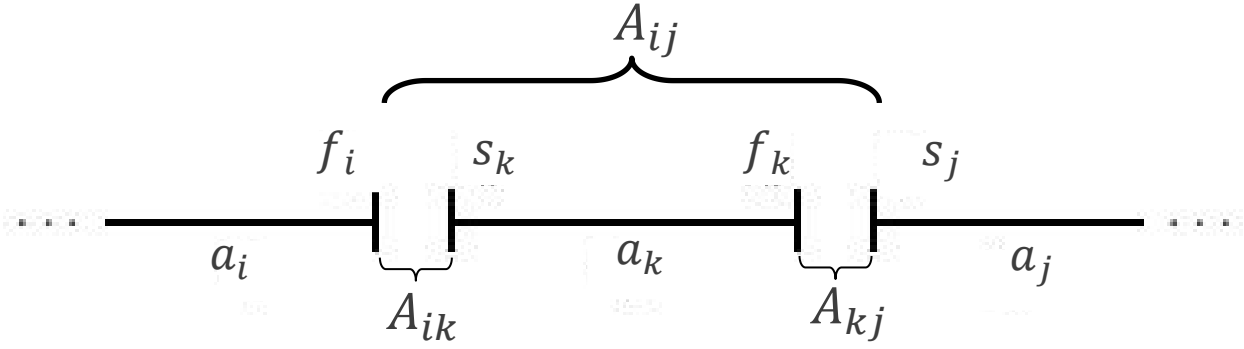


Optimal Substructure

- Assume that A_{ij} is an optimal solution to S_{ij} and includes activity a_k. Sets A_{ik} and A_{kj} must also be optimal solutions.

最优子结构：假设A_{ij}是S_{ij}的最优解，并且A_{ij}包含a_k,那么A_{ik}以及A_{kj}也是最优解

- Prove by contradiction. 通过**反证法**来证明



Recursive Equation

构建递归方程

- Let $c[i,j] = |A_{ij}|$as the size of the maximum subset of mutually compatible activities in S_{ij}.

定义$c[i,j] = |A_{ij}|$是最大相容任务的集合的大小

- If $S_{ij} = \emptyset$, then $c[i,\ j] = 0$.
- If $S_{ij} \neq \emptyset$ and if we consider that a_k is used in an optimal solution, we have: 递归方程就是下面的

$$c[i,j] = c[i,k] + c[k,j] + 1$$

Recursive Equation

- The recursion equation is:

$$c[i,j] = \begin{cases} 0 & S_{ij} = \emptyset \\ \max\limits_{i<k<j, a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & S_{ij} \neq \emptyset \end{cases}$$

- There are $j - i - 1$ possible values for k.
  - $k = i + 1, \ldots, j - 1$.（K不等于i或者j）
  - We check all the values and take the best one. 遍历i和j之间所有的k，找最大的
  - Nothing special, very similar to the matrix-chain multiplication problem and the optimal BST problem.

Simpler Idea 更简单的解法

- Is dynamic programming the most efficient to solve this problem? 动态规划方法在这个例子里面不够高效
- How about just simply select the activity with earliest finish time? 采用贪心法每次选择最早结束的任务，之后选择和这个任务先容的最早结束的任务，不断迭代

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)

- It seems working. But how to prove the correctness?

Simpler Idea

Theorem

Let $S_{ij} \neq \emptyset$ and a_m be the activity in S_{ij} with the earliest finish time:

$$f_m = \min\{f_k : a_k \in S_{ij}\}$$

Then:

(1) Activity a_m must be in some optimal subset A_{ij}.

- i.e. there exist some optimal solutions that contains a_m. a_m肯定是包含在A_{ij}的某个最优解法里

(2) $S_{im} = \emptyset$, so that choosing a_m leaves S_{mj} the only nonempty subproblem.

选择a_m之后，S_{im}是空的子问题，剩下的非空的子问题就是 S_{mj}了

(1) 选最早结束的a_m准没错!
(2) 选最早结束的之后只有一个子问题需要求解

Simpler Idea

(1) 选最早结束的a_m准没错! 证明a_m肯定是包含在A_{ij}的某个最优解法里

Proof of (1) Activity a_m must be in some optimal subset A_{ij}:

- Assume activities in A_{ij} are in increasing order of finish time, and let a_k be the first activity in A_{ij}: $A_{ij} = \{a_k, \ldots\}$.
- If $a_k = a_m$, done!
- Otherwise, replace a_k with a_m (resulting in a set A_{ij}')
  - Because f_m is the earliest finish time, $f_m \leq f_k$. The activities in A_{ij}' will continue to be compatible.
  - A_{ij}' will have the same size with A_{ij}.

如果a_k不等于a_m 那么就通过修改A_{ij}的方式，将A_{ij}修改得是A_{ij}'，他们两个的大小是一样的. 得到一个包含任务a_m的最优解A_{ij}' 得证

Simpler Idea

Proof of (2) $S_{im} = \emptyset$, so that choosing a_m leaves S_{mj} the only nonempty subproblem:

选择a_m之后，会使得 $S_{im} = \emptyset$是一个空集，并且S_{mj} 是剩下的唯一的非空子问题

- Assume $S_{im} \neq \emptyset$, i.e. there exists an activity $a_k \in S_{im}$:

$$f_i \leq s_k < f_k \leq s_m < f_m$$

- $f_k < f_m$ contradicts with the fact that a_m has the earliest finish time.
- Therefore, there is no $a_k \in S_{im}$, which implies $S_{im} = \emptyset$.

Why is the Theorem Useful? 这个理论有用么

- Given the theorem, what can we do and how can we improve?

对比下动态规划方法和采用刚才的这个定理的方法

	Dynamic programming	Using the theorem
Number of subproblems in the optimal solution	2 subproblems: S_{ik}, S_{kj}	1 subproblem: S_{mj} $S_{im} = \emptyset$
Number of choices to consider	$j - i - 1$ choices	1 choice: the activity with the earliest finish time in S_{ij}

- Making the greedy choice (the activity with the earliest finish time in S_{ij})
  - Reduce the number of subproblems and choices.
  - Solve each subproblem in a top-down fashion.

Greedy Approach

- To select a maximum size subset of mutually compatible activities from set S_{ij}:
  - Choose $a_m \in S_{ij}$ with earliest finish time (greedy choice). 贪心选择最早结束的任务
  - Add a_m to the set of activities used in the optimal solution. 将最早结束的任务加入解法中
  - Solve the same problem for the set S_{mj}. 迭代的解决剩下的问题
- From the theorem, it is proved that by choosing a_m we are guaranteed to have used an activity included in an optimal solution
  - We do not need to solve the subproblem S_{mj} before making the choice!
- This problem has the greedy choice property (贪心选择性质). 也就是说必须证明我们贪心的方式取得的结果必须属于最优解的。这一题里面的 a_m 就是符合这个性质

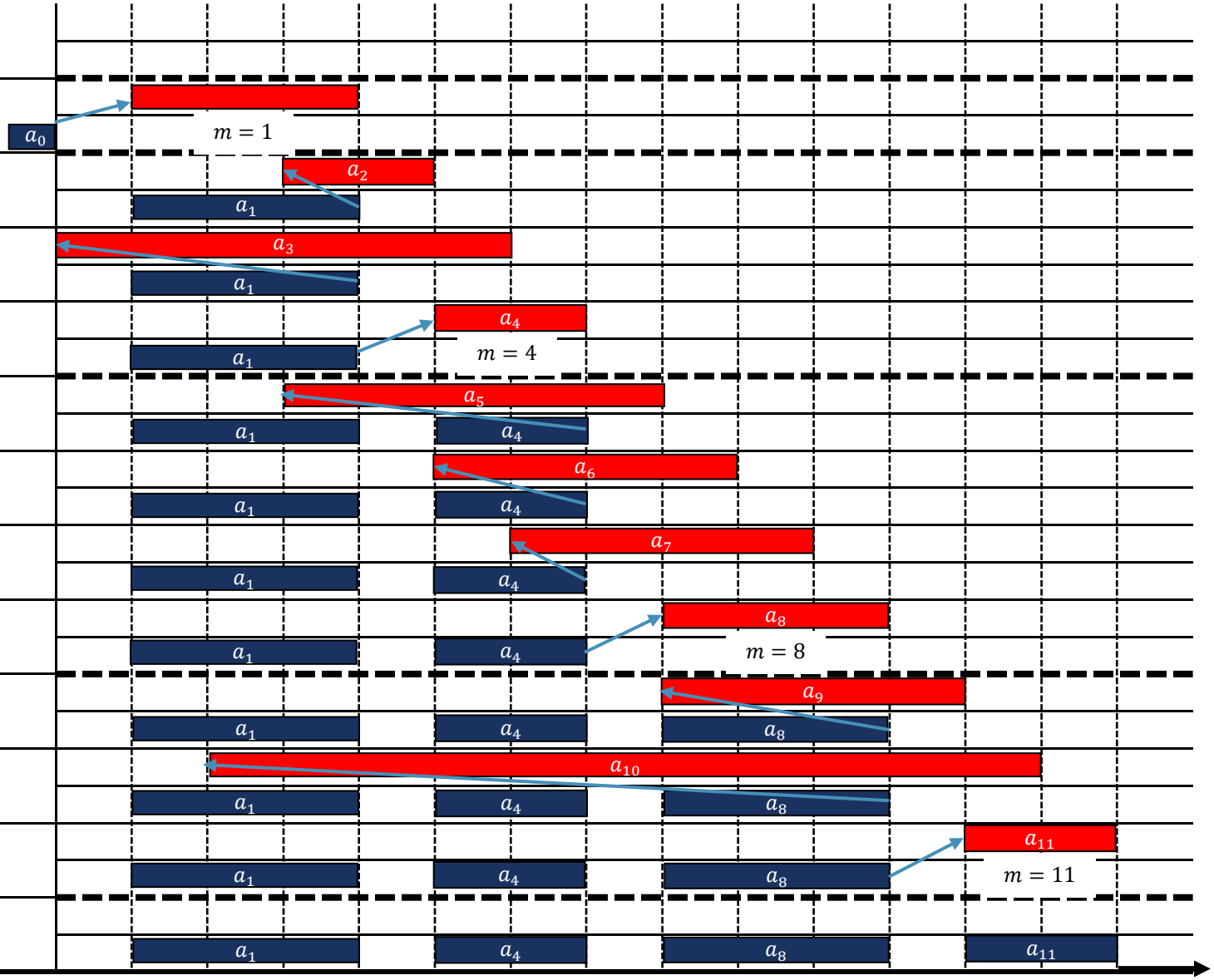
A Recursive Greedy Algorithm 贪心问题递归解

- Activities are ordered in increasing order of finish time.
- Running time: $O(n)$
  - Each activity is examined only once.
- Initial call: RecursiveTaskSelect$(s, f, 0, n+1)$

```
RecursiveTaskSelect(s, f, i, j)
1 m ← i + 1
2 while m < j and s_m < f_i do
3     m ← m + 1
4 if m < j then return {a_m} ∪
                RecursiveTaskSelect(s, f, m, j)
5 else return ∅
```

$s_m < f_i$: 开始仍然在i结束之前
当a_m的开始时间在i的结束之前，那么就要m = m + 1，选后面一个任务

k	s_k	f_k
0	-	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-



An Iterative Greedy Algorithm 迭代

- It totally not necessary to use recursion.
- Activities are ordered in increasing order of finish time.
- Running time: $\boldsymbol{O(n)}$
  - Each activity is examined only once.

```
GreedyTaskSelect(s, f)
1  A ← {a_1}
2  i ← 1
3  for m ← 2 to n do
4      if s_m ≥ f_i then
5          A ← A ∪ {a_m}
6              i ← m
7  return A
```

Designing Greedy Algorithms

贪心算法一般的求解步骤

1. Cast the optimization problem as: we make a choice and are left with only one subproblem to solve. 我们做了一步选择之后，问题变成一个子问题 (而不是多个子问题)
2. Prove that there is always an optimal solution to the original problem when making the greedy choice.
   - Making the greedy choice is always safe. 要证明我们做出的这步贪心决策是对的，也就是这一步贪心做出的选择是在最优解里面的
3. Demonstrate that after making the greedy choice: an optimal solution = the greedy choice + an optimal solution to the resulting subproblem. 证明最优解=贪心这一步+后续子问题的最优解

Correctness of Greedy Algorithms

- Greedy choice property 贪心选择性质
  - A globally optimal solution can be arrived at by making a locally optimal (greedy) choice. 贪心性质，全局最优解可以通过局部最优解一步步达到
- Optimal substructure property 最优子结构性质
  - We know that we have arrived at a subproblem by making a greedy choice.
  - optimal solution for the original problem = optimal solution to subproblem + greedy choice.

当通过贪心到达一个子问题之后，这个子问题的最优解+贪心的选择=原问题最优解

Correctness of Greedy Algorithm for 任务选择问题

全局最优解可以通过
局部最优解一步步达到

- Greedy choice property 贪心选择性质
  - There exists an optimal solution that includes the greedy choice: The activity a_m with the earliest finish time in S_{ij}. 选择am，am是最优解的一部分
- Optimal substructure property 最优子结构
  - An optimal solution to subproblem S_{ij} = selecting activity a_m + optimal solution to subproblem S_{mj}. 选择am之后，和后续的贪心结果拼接起来就是最优解

贪心算法好编，但是不好证明它的正确性

Dynamic Programming vs. Greedy Algorithms

- Dynamic programming
  - Make a choice at each step.
  - The choice depends on solutions to subproblems. 选择依赖于子问题，所以要遍历子问题
  - Bottom up solution, from smaller to larger subproblems.
- Greedy algorithm
  - Make the greedy choice.
  - Solve the subproblem arising after the choice is made.
  - The choice we make may depend on previous choices, but not on solutions to subproblems. 这个贪心的选择不依赖于子问题，因此就很快
  - Top down solution, problems decrease in size.
- Common: Optimal substructure

Experiment for Activity Selection Problem

Table 7.1 Running-time comparison of algorithms for activity selection problem

n	100	200	400	600	800	1000
Dynamic programming(DP)	0.000	0.015	0.189	0.656	1.656	3.250
RecursiveTaskSelect(RTS)	0.000	0.000	0.000	0.000	0.000	0.000
GreeyTaskSelect(GTS)	0.000	0.000	0.000	0.000	0.000	0.000
Maximum task number	19	24	30	40	46	52

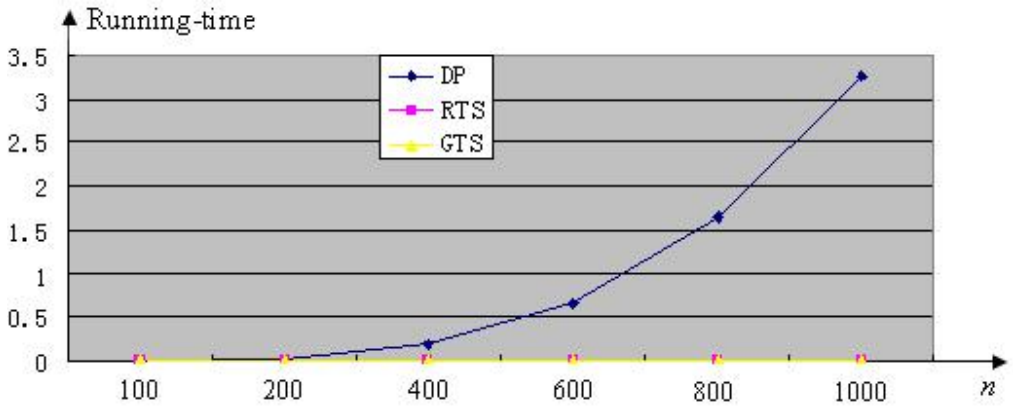


Image source: 图7.10, 张德富, 算法设计与分析, 国防工业出版社, 2009.

Classroom Exercise 课堂练习

Use greedy algorithm to solve the following problem:

- Given n integers, concatenate them in a row to constitute a maximum integer.
- For example:
  - $n = 3$, 34331213 is the maximum integer to concatenate 13, 312, 343.
  - $n = 4$, 7424613 is the maximum integer to concatenate 7, 13, 4, 246.

Classroom Exercise

Solution 1:

- Simply select the integer with largest first bits.
- How about the following cases:
  - 12, 121
  - 12, 123

Solution 2:

- For the numbers with the same first bits, compare $a + b$ with $b + a$ and use the one with maximum value.

FRACTIONAL KNAPSACK PROBLEM

背包问题

Fractional Knapsack Problem

Fractional knapsack (部分背包) problem:

- There are n items: the ith item is worth v_i dollars and weights w_i kg.
- The capacity of knapsack is W kg.
- ~~Items must be taken entirely or left behind.~~
- Items can be taken fractionally. 一个物品可以切分的放到背包里面
- Which item fractions should we take to maximize the total value?

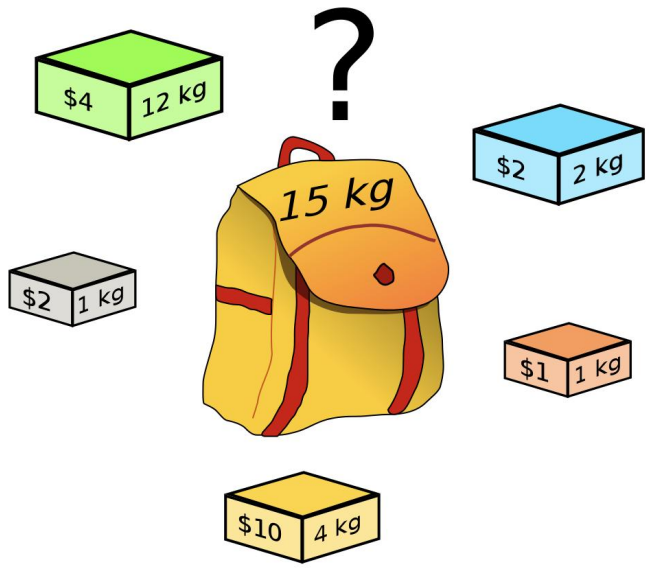


Image source: https://en.wikipedia.org/wiki/Knapsack_problem

Example

- Weight capacity $W = 50\text{kg}$.
- Using the solution of 0/1 knapsack problem, we choose item 2 and 3 with total value $100 + 120 = 220$.
- However, the solution of fractional knapsack problem is to choose item 1 and 2 plus 2/3 of item 3, with total value $60 + 100 + 120 \times 2/3 = 240$.

i	v_i	w_i
1	\$60	10kg
2	\$100	20kg
3	\$120	30kg

Greedy Strategy

- Greedy strategy 1: Pick the item with the maximum value.
- This greedy strategy is obviously not optimal.

$$W = 50\text{kg}$$

i	v_i	w_i
1	\$60	10kg
2	\$100	20kg
3	\$120	30kg

Greedy Strategy

- Greedy strategy 2: Pick the item with the maximum value per kg v_i/w_i. （单位重量价值）
- If the supply of that element is exhausted and we can carry more, take as much as possible from the item with the next greatest value per kg.
- It is good to order items based on their value per kg:

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}.$$

$$W = 50\text{kg}$$

i	v_i	w_i	v_i/w_i
1	\$60	10kg	\$6/kg
2	\$100	20kg	\$5/kg
3	\$120	30kg	\$4/kg

It seems ok! But how to prove?

Pseudocode

Greedy2Knapsack(W, v)

1 order items based on their value per kg: $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$

2 $i \leftarrow 1$; $w \leftarrow W$

3 **while** $w > 0$ and as long as there are items remaining **do**

4 $\quad x_i \leftarrow \min\{1, w/w_i\}$

5 $\quad$ remove item i from list

6 $\quad w \leftarrow w - x_i w_i$

7 $\quad i \leftarrow i + 1$

Running time: $\Theta(n)$ if items already ordered; else $\Theta(nlgn)$

Greedy Choice Property 贪心选择性质

- Now we need to prove that the fractional knapsack problem has greedy choice property.
- Assume the items are ordered based on their value per kg:

$$1 \;\; 2 \;\; 3 \;\; \dots \;\; i \;\; \dots \;\; n$$

- The greedy solution is

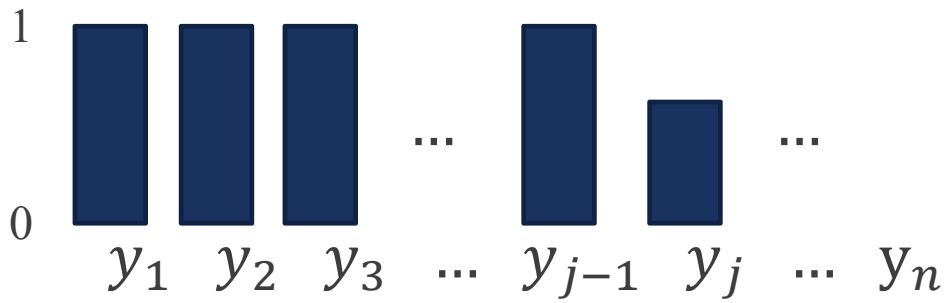
$$x_1 \;\; x_2 \;\; x_3 \;\; \dots \;\; x_i \;\; \dots \;\; x_n$$

where $x_i \in [0,1]$ is the fraction to take item i.

Greedy Choice Property

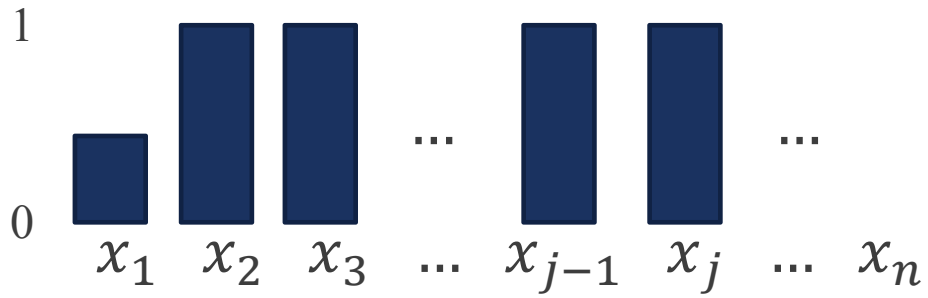
- The greedy solution will always look like this:

假设我们的贪心解法是这样的y序列



- Now, if the greedy solution is not optimal, there must exists an optimal solution that looks like this:

一个最优解是这样的X序列，不妨假设第一个物品没有装满

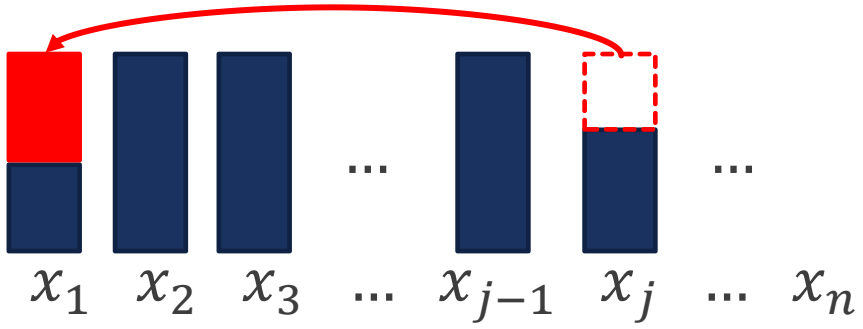


One of the first $j-1$ items is not fully taken, simply assume that it is the first item.

Greedy Choice Property

- Now, we can do some transformation to the optimal solution: Increase the fraction of item 1 and by decreasing item j.
  - Moving weight: $(y_1 - x_1)w_1$.
  - Value increased for item 1: $(y_1 - x_1)w_1 \times v_1/w_1$.
  - Value decreased for item j: $(y_1 - x_1)w_1 \times v_j/w_j$.

Which one is larger?



Greedy Choice Property 贪心选择性质

- Therefore, given any solution, we can transform it to the greedy solution for larger value.

给定任意一个解法，我们都可以通过这种方式，将这个解法改为贪心解法，并且价值更高

- It means the greedy solution has the largest value.

这个就证明了贪心解法有最高的价值

Optimal Substructure

- 证明贪心选择性质还不够，我们还必须证明这个问题具有最优子结构性质
- Consider the most valuable load that weights at most W kg. 假设有一个最多为W kg的最优解法
- 如果我们将重量为w的j物品部分从最优解法中去除，那么剩下的解法肯定是给定容量$W-w$的背包，并且物品为剩余的n-1个物品以及 w_j-w 的j物品的最佳装载方案
- If we remove a weight w of item j from the optimal load, the remaining load must be the most valuable load weighing at most $W-w$ that can be taken from the remaining $n-1$ items plus w_j-w kg of item j.（反证法）

Classroom Exercise 课堂练习

Can greedy algorithm obtain optimal solution for the coin change problem?

对于我们在动态规划里面的找硬币问题，贪心解法能够找到最优解么?

找硬币问题

你有三种硬币，d_1=x元、，d_2 = y元、，d_3 = z元，每种硬币足够多，买一本书需要W元，用最少的硬币数量来买书

Classroom Exercise 课堂练习

贪心算法

1. Select the largest coin.
2. Check if adding the coin makes the change exceed the amount.
   a. No, add the coin.
   b. Yes, set the largest coin as the second largest coin and go back to step 1.
3. Check if the total value of the change equals the amount.
   a. No, go back to step 1.
   b. Yes, problem solved.

Classroom Exercise 课堂练习

- Successful example:
  - For $N = 86$ (cents) and $d_1 = 1,\ d_2 = 2,\ d_3 = 5,\ d_4 = 10,\ d_5 = 25,\ d_6 = 50,\ d_7 = 100.$
  - The greedy approach is optimal: 50, 25, 10, 1.
- Failed example: 失败的例子
  - For $N = 6$ (cents) and $d_1 = 1,\ d_2 = 3,\ d_3 = 4.$
  - The greedy approach is not optimal: 4, 1, 1.
  - The optimal solution: 3, 3.
- For this problem, the success of greedy approach depends on the coin currency.
  - It works for canonical coin systems like US, but not for arbitrary coin system.

HUFFMAN CODE

哈夫曼编码

Data Compression by Binary Code

- The problem of data compression (数据压缩) is to find an efficient method for encoding a data file.
  - Compress string S into S', which can be restored to S, such that $|S'| < |S|$.
- A common way to represent a file is to use a binary code (二进制编码).
- In such a code, each character is represented by a unique binary string, called the codeword (码字).

- 为什么程序员也被成为码农?
  - 编程也是一种编码

Data Compression by Binary Code

There are two ways to use binary code:

- A fixed-length code (固定长度编码) represents each character using the same number of bits.
  - For example, suppose our character set is $\{a,\ b,\ c\}$.
  - Then we could use 2 bits to code each character: a: 00, b: 01, c: 11.
  - Given this code, if our file is $ababcbbbc$, our encoding will be 000100011101010111.
- We can obtain a more efficient coding using a variable-length code (可变长度编码).
  - Such a code can represent different characters using different numbers of bits.

Data Compression by Binary Code

- For example, a data file of 100 characters contains only the characters a-f, with the frequencies:

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- How much do we save in this case?
  - Fixed-length code: $100 \times 3 = 300$ bits.
  - Variable-length code: $(45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4) = 224$ bits

在谍战片里面用的比较多的莫尔斯码，就是一种变长编码，用长短不同的两种信号（不同长度的滴答声音）对英文的字母和常见符号进行编码

Prefix Codes

- One particular type of variable-length code is a prefix code **(前缀编码)**.
- No codeword is also a prefix of some other codeword.

没有一个码字是另外一个码字的前缀

- The advantage of a prefix code is that there is no ambiguity when interpreting the codes. 这种方式在解码时没有歧义
- For example:
  - $abc \rightarrow 0 \cdot 101 \cdot 100 \rightarrow 0101100$
  - $010110000 \rightarrow 0 \cdot 101 \cdot 100 \cdot 0 \cdot 0 \rightarrow abcaa$

a	b	c
0	101	100

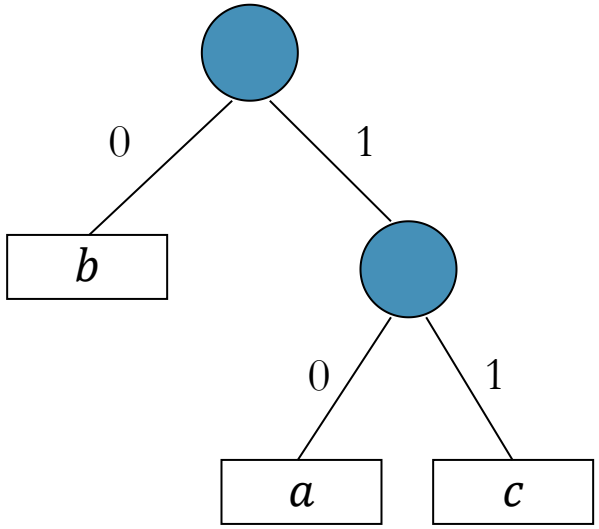
由字符编码获得的最优数据压缩总可用某种前缀编码来获得

Binary Tree Representation

二叉树的形式表示前缀码

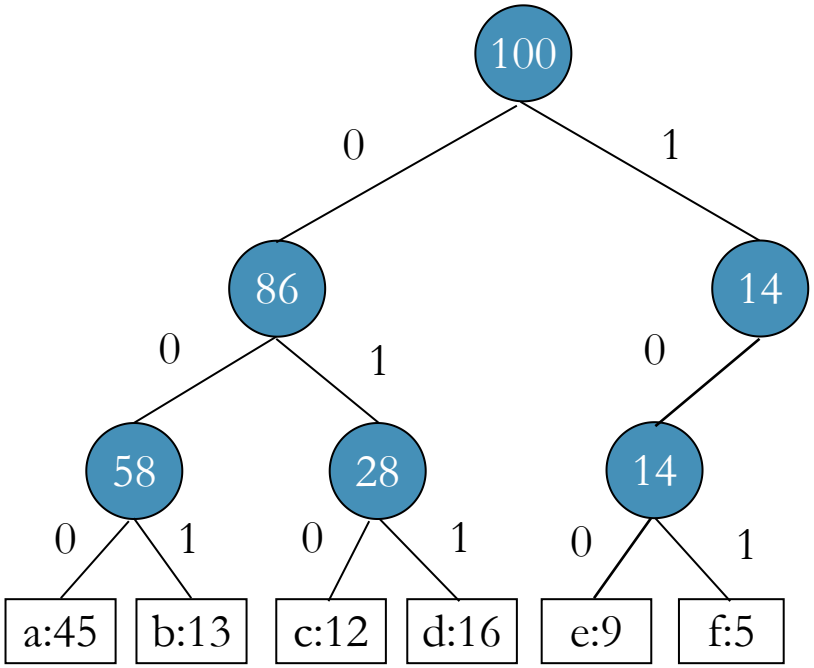
- The prefix coding can be represented by a binary tree, where we put all characters on leaves. 前缀编码方式
- Interpret the binary codeword for a character as **the path** from the root to that character, where 0 means "go to the left child" and 1 means "go to the right child.", then we can construct a binary tree corresponding to the coding schemes.

一个字母的编码是从根到该字母所在叶的的路径的0,1串

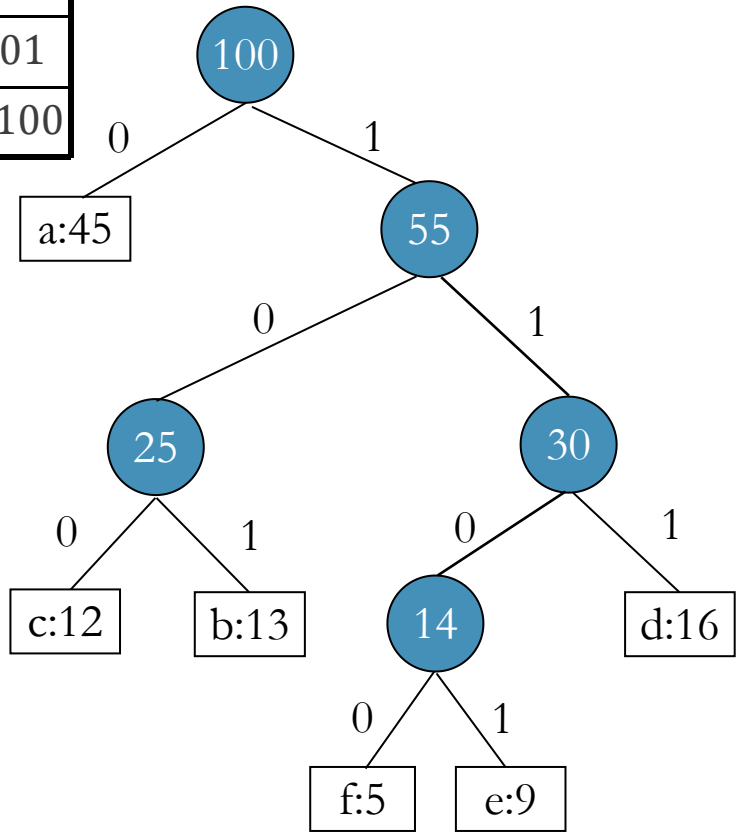


Binary Tree Representation

	a	*b*	*c*	*d*	*e*	*f*
Frequency	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100



Fixed-length code



Variable-length code

Binary Tree Representation

- Given a tree T corresponding to a prefix code, for each character c in the alphabet C, let （一个二叉树对应一个前缀码编码）
  - $\boldsymbol{f(c)}$ denote the frequency of c in the file; 字母c文件中出现的概率
  - $\boldsymbol{d_T(c)}$ denote the depth of c's leaf in the tree, also the length of the codeword for character c. C的深度
- The number of bits required to encode a file is thus 总共需要的编码位数

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

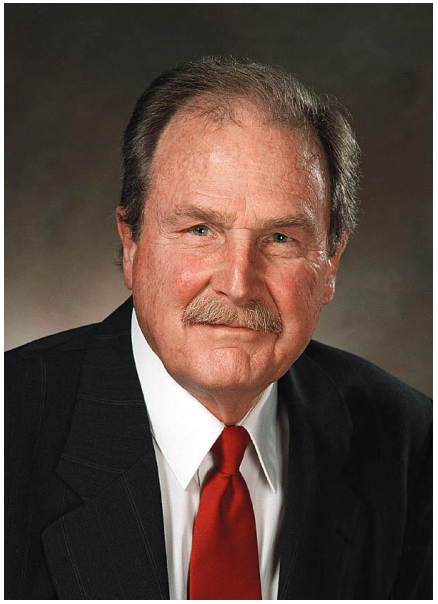
which we define as the cost of the tree T.

- It is similar to the optimal BST problem, but with no constraint of being a search tree ($k_{left} \leq k_{node} \leq k_{right}$). 类似最优二叉搜索树问题，但是没有关键字的限制

编码位数取决于T，关键在于：如何构建T，使得代价B(T)最小?

Huffman Code

- Huffman code (哈夫曼编码/霍夫曼编码) was developed by David A. Huffman while he was a Sc.D. student at MIT in 1952.
- 哈夫曼编码是他上一门课的作业
- Huffman code efficiently builds the optimal prefix code, given the frequency of each character.



David Albert Huffman
(1925 – 1999)

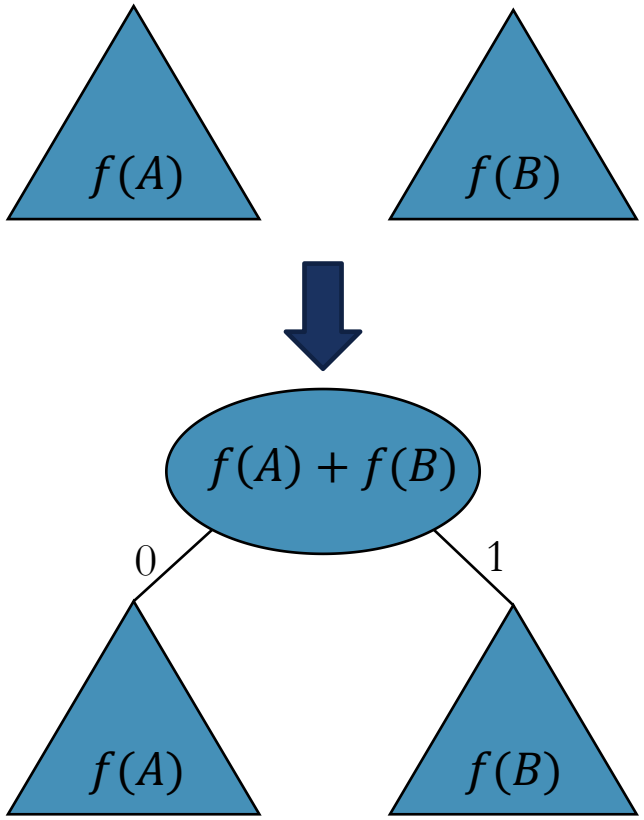
Huffman's Algorithm

- Huffman's algorithm **builds the codeword** bottom up. 自底向上的构建二叉树
- Consider a forest of trees:
  - Initially, one separate node for each character. 最开始时每个字母一个节点，一个节点一棵树
  - In each step, join two trees into a larger tree.

  每一步都选择两颗树进行合并
  - Repeat this until only one big tree remains.

  直到只剩下一颗树
- Which trees to join? Greedy choice the trees with the lowest frequencies!

贪心选择概率最小的两颗树进行合并



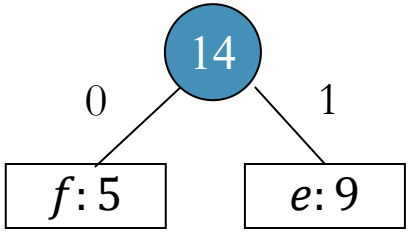
Example

- **Sort** with character frequency.

$f: 5$	$e: 9$	$c: 12$	$b: 13$	$d: 16$	$a: 45$

Example

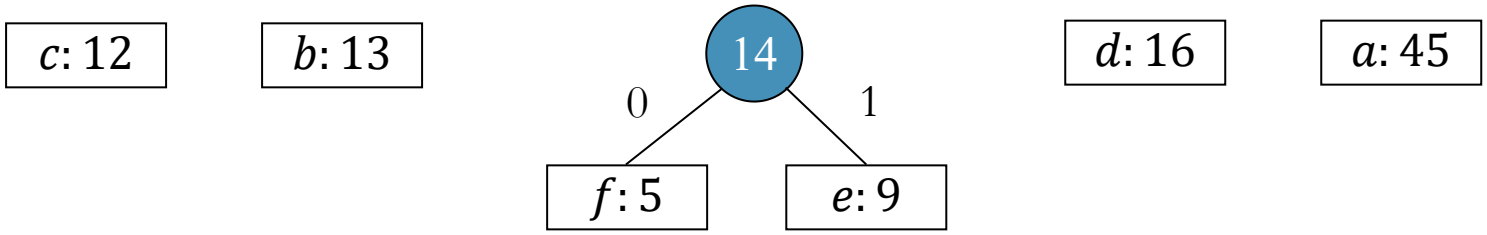
- **Join** two nodes with lowest frequency, and sum up the frequency.



c: 12 b: 13 d: 16 a: 45

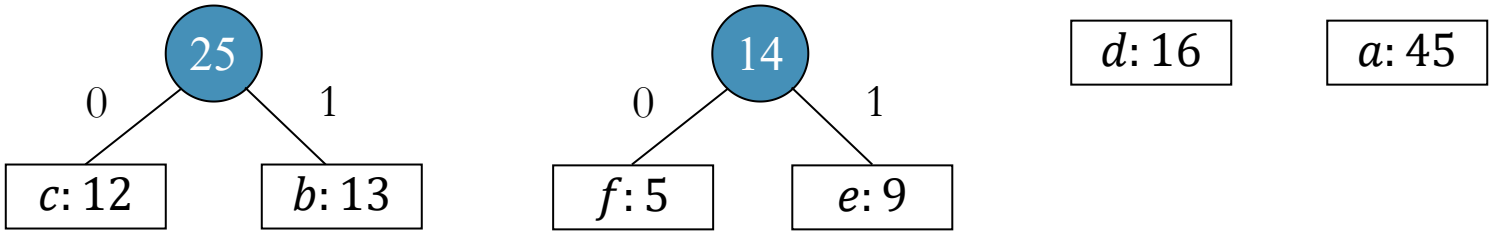
Example

- **Insert** into proper position. 插入到合适的位置



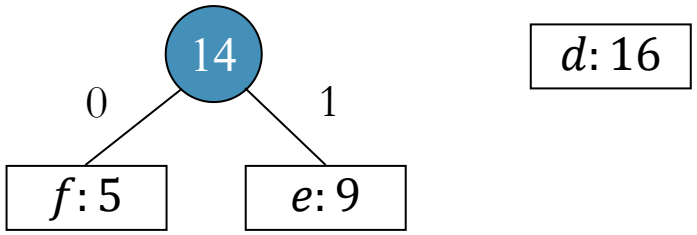
Example

- Join two nodes with lowest frequency, and sum up the frequency.

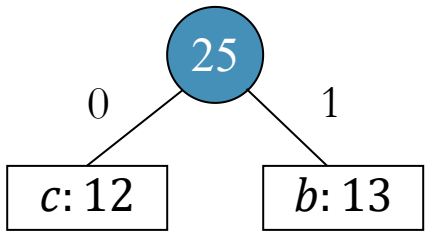


Example

- **Insert** into proper position.



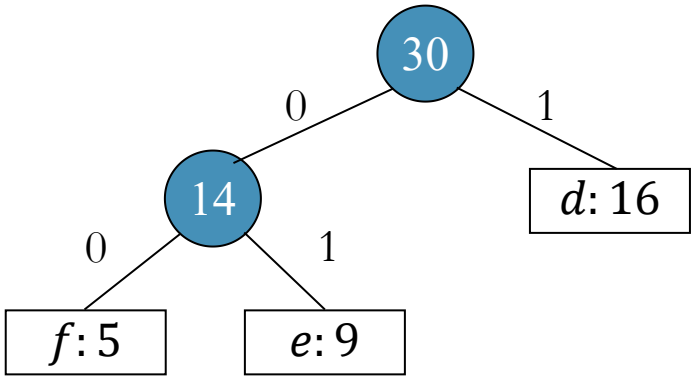
d: 16

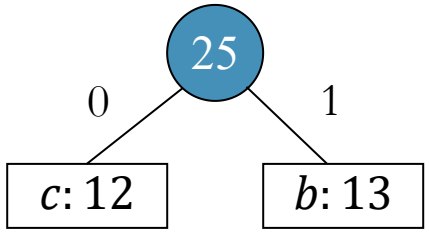


a: 45

Example

- **Join** two nodes with lowest frequency, and **sum up** the frequency.

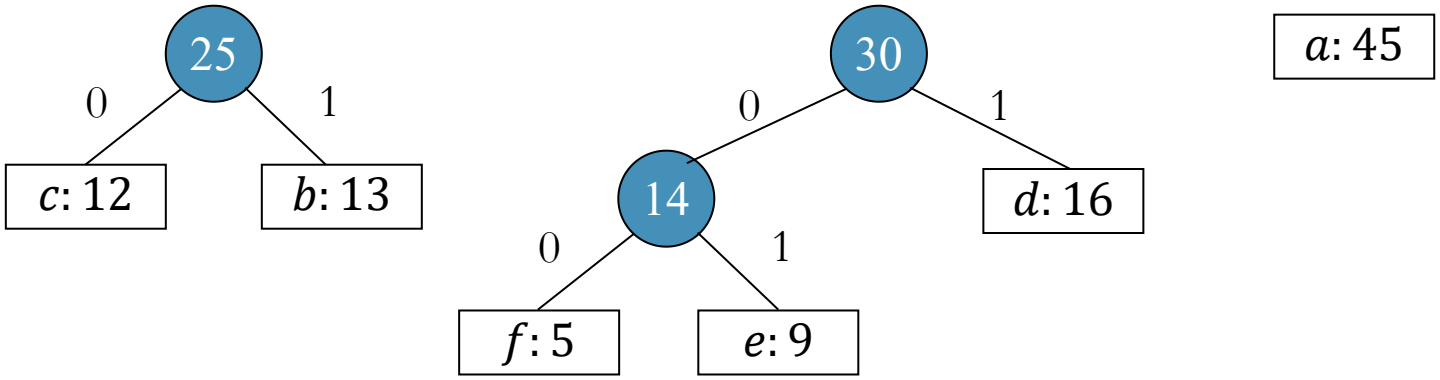




$a: 45$

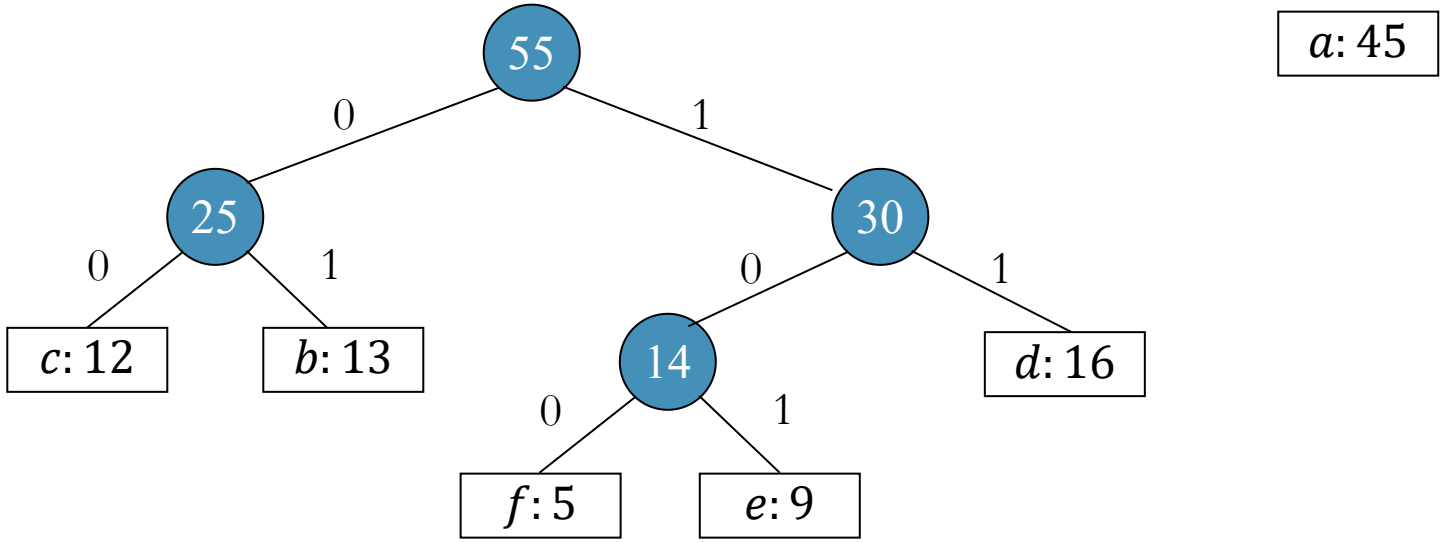
Example

- **Insert** into proper position. 插入到合适的位置



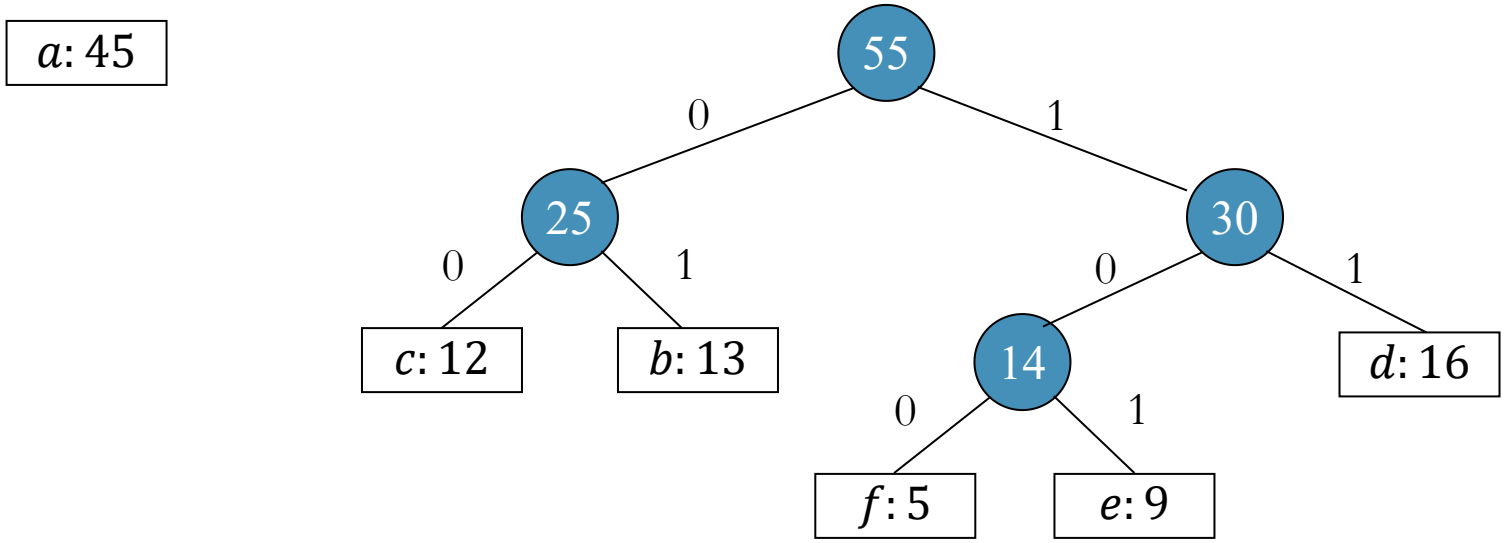
Example

- **Join** two nodes with lowest frequency, and **sum up** the frequency.



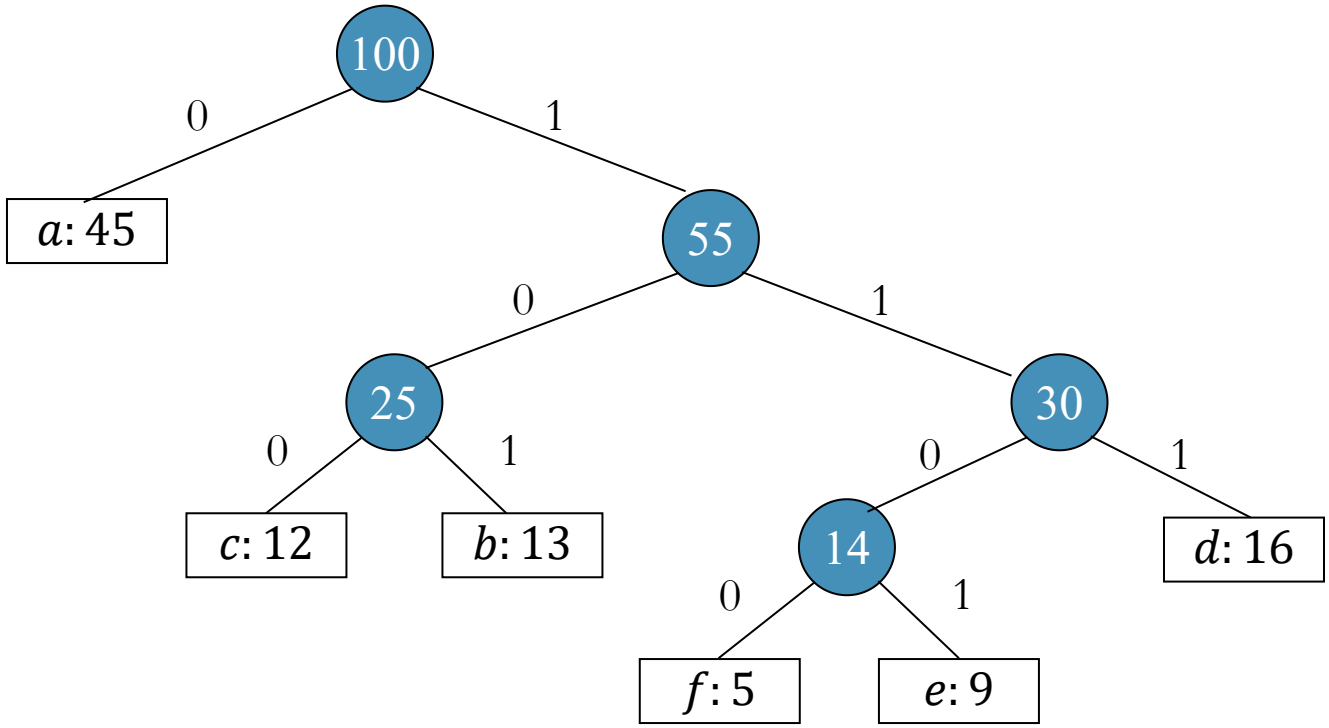
Example

- **Insert** into proper position.



Example

- **Join** two nodes with lowest frequency, and **sum up** the frequency.



Pseudocode 伪代码

Minimum priority queue

```
HuffmanCode(C)
1  Q ← C
2  for i ← 1 to n − 1 do
3       allocate a new node z
4       x ← ExtractMin(Q)
5       y ← ExtractMin(Q)
6       f(z) ← f(x) + f(y)
7       Insert(Q, z)
8  return ExtractMin(Q)
```

ExtractMin and Insert are heap operators, which take $O(\lg n)$. Therefore, the total cost is $O(n \lg n)$.

- Greedy algorithm is always easy. What is the difficult part?

  最小优先队列，可以用堆来实现

Correctness of Huffman's Algorithm

证明哈夫曼编码算法的正确性

We are going to prove two things:

- Greedy choice property: Select the lowest frequency characters is always correct.贪心选择性质，选择最低频率的两棵树是正确的
- Optimal substructure property: Greedy choice + optimal solution of subproblem = optimal solution of original problem.

贪心选择的结果 + 子问题的最优解 = 最优解

Correctness of Huffman's Algorithm

- We first prove the **greedy choice property**. 证明贪心选择性质

Theorem 7.4

Let C be an alphabet in which each character $c \in C$ has frequency $f(c)$.

Let x and y be two characters in C having the lowest frequencies.

Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

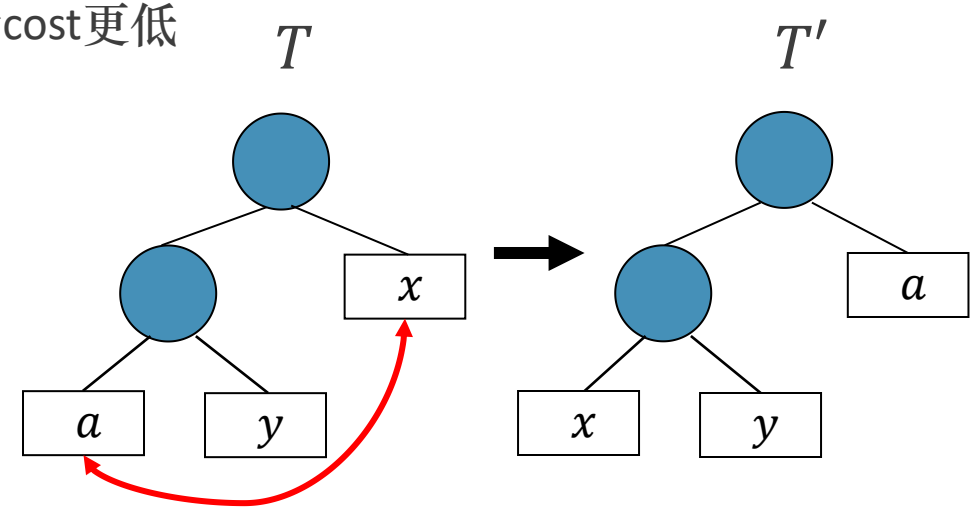
- In one sentence: To join the lowest frequency characters is always correct!

Correctness of Huffman's Algorithm

假设有个二叉树是最优编码，那么将二叉树最深的两个节点，换成概率最小的两个
证明换了之后cost更低

Proof:

- Idea: If x and y are not in the deepest level of an optimal tree, we can always switch them to the deepest level, to obtain a better tree.



$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$

$$= \big(f(x)d_T(x) + f(a)d_T(a)\big) - \big(f(x)d_{T'}(x) + f(a)d_{T'}(a)\big)$$

$$= \big(f(x)d_T(x) + f(a)d_T(a)\big) - \big(f(x)d_T(a) + f(a)d_T(x)\big)$$

$$= \big(f(a) - f(x)\big)\big(d_T(a) - d_T(x)\big) \geq 0$$

Correctness of Huffman's Algorithm

- Then we prove that the problem of constructing optimal prefix codes has the optimal substructure property. 现在要证明最优子结构

Theorem 7.5

The conditions is the same as Theorem 7.4 .

Let C' be the alphabet C with characters x, y removed and a new character z added, namely $C' = \{C - \{x, y\}\} \cup \{z\}$.（删除x,y加入Z得到新表）

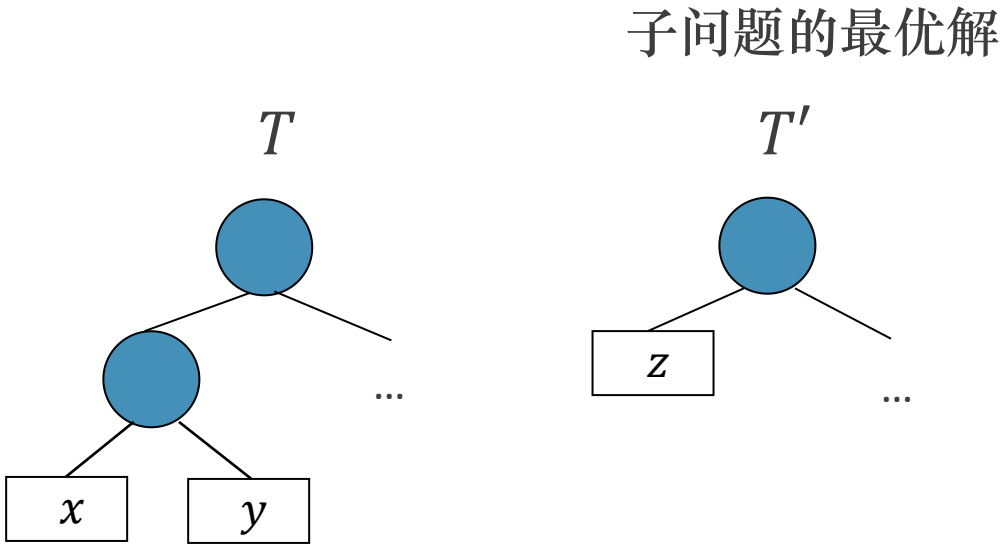
Let $f(z) = f(x) + f(y)$ and other frequencies in C and C' are same.

Let T' be an optimal prefix code for C'.

Then the tree T for an optimal prefix code for C can be obtained from T' by replacing the leaf node for z with an internal node having x and y as children.

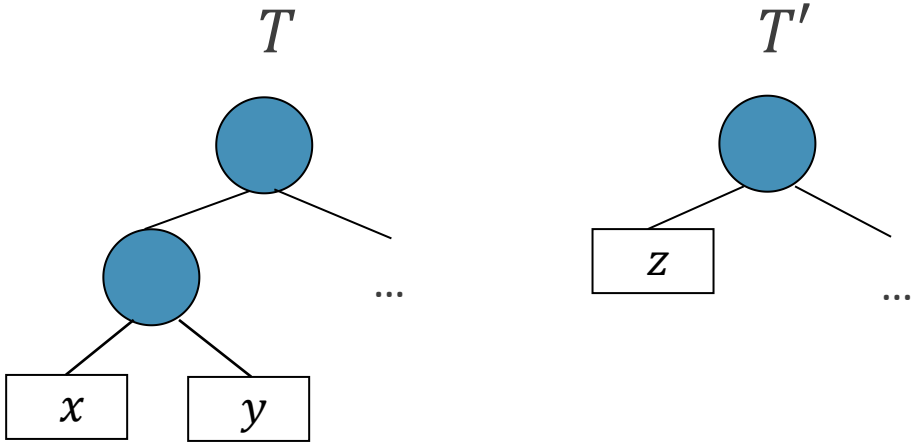
- In one sentence: Merge x and y + optimal solution of C' = optimal solution of C.

Correctness of Huffman's Algorithm



- T' is the optimal solution of the subproblem.
- We need to prove: Given the greedy choice and T', we can obtain the optimal solution T for the original problem.

Correctness of Huffman's Algorithm



Proof:

- Let $B(T)$ and $B(T')$ be the cost of tree T and T', then

$$\begin{aligned} & f(x)d_T(x) + f(y)d_T(y) \\ &= \big(f(x) + f(y)\big)(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + \big(f(x) + f(y)\big) \end{aligned}$$

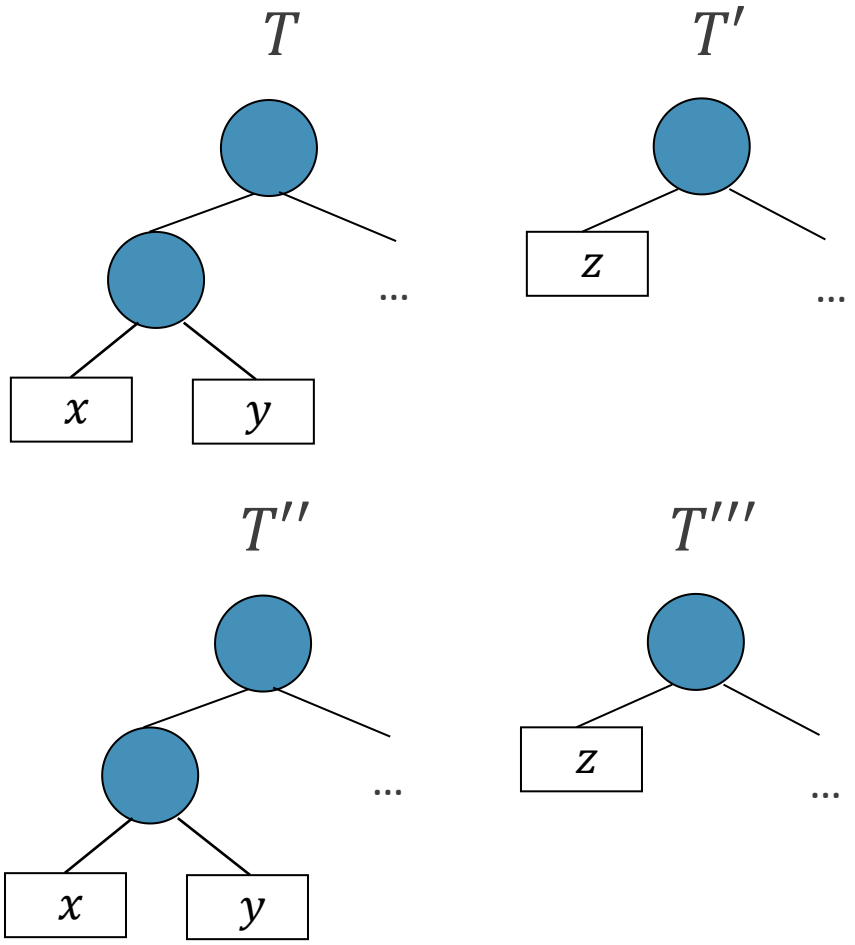
$d_T(x) = d_T(y) = d_{T'}(z) + 1$

- So $B(T) = B(T') + f(x) + f(y)$.

Correctness of Huffman's Algorithm

Proof (cont'd): 反证法证明

- We prove it by contradiction: If T' is optimal but T is not optimal.
- If T is not optimal, there must exist an optimal tree T'' such that $B(T'') < B(T)$, and x and y are also at the deepest level of T'' (Theorem 7.4).
- Then we can construct a tree T''' from T'', by removing x and y and adding z to T''.



Correctness of Huffman's Algorithm

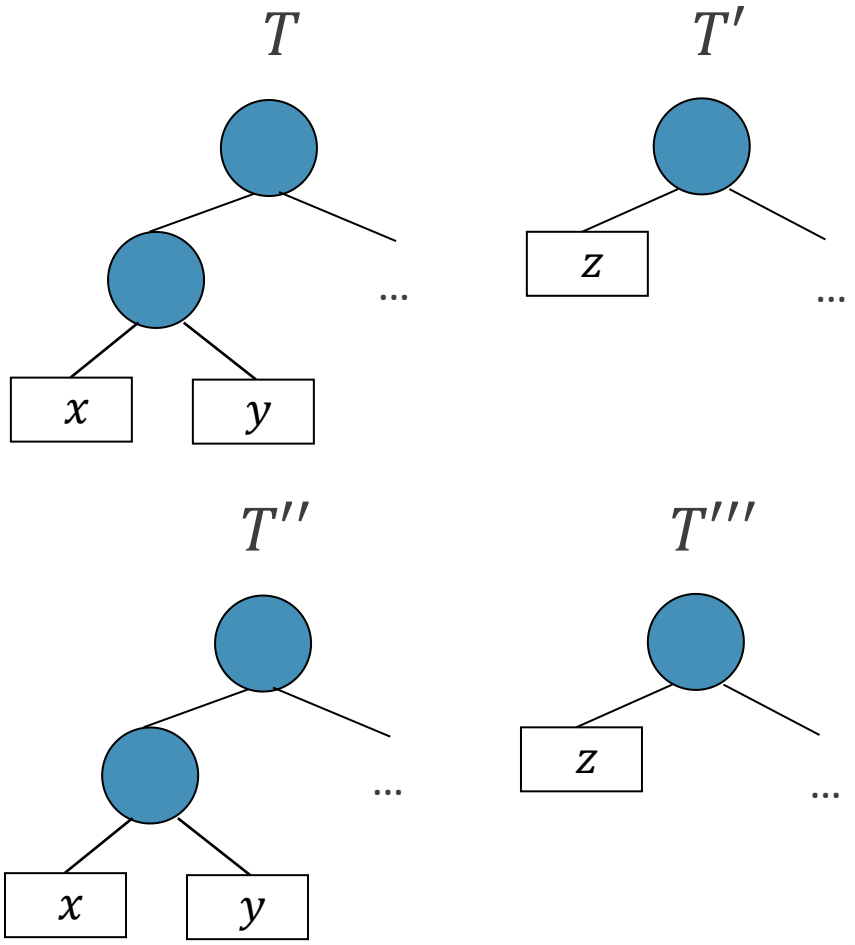
Proof (cont'd):

- By the formula we derive before:

$$B(T'') = B(T''') + f(x) + f(y)$$
$$B(T) = B(T') + f(x) + f(y)$$
$$B(T'') < B(T)$$

we obtain $B(T''') < B(T')$, yielding a contradiction to that T' is optimal.

- Thus T represents an optimal prefix code for C.



Correctness of Huffman's Algorithm

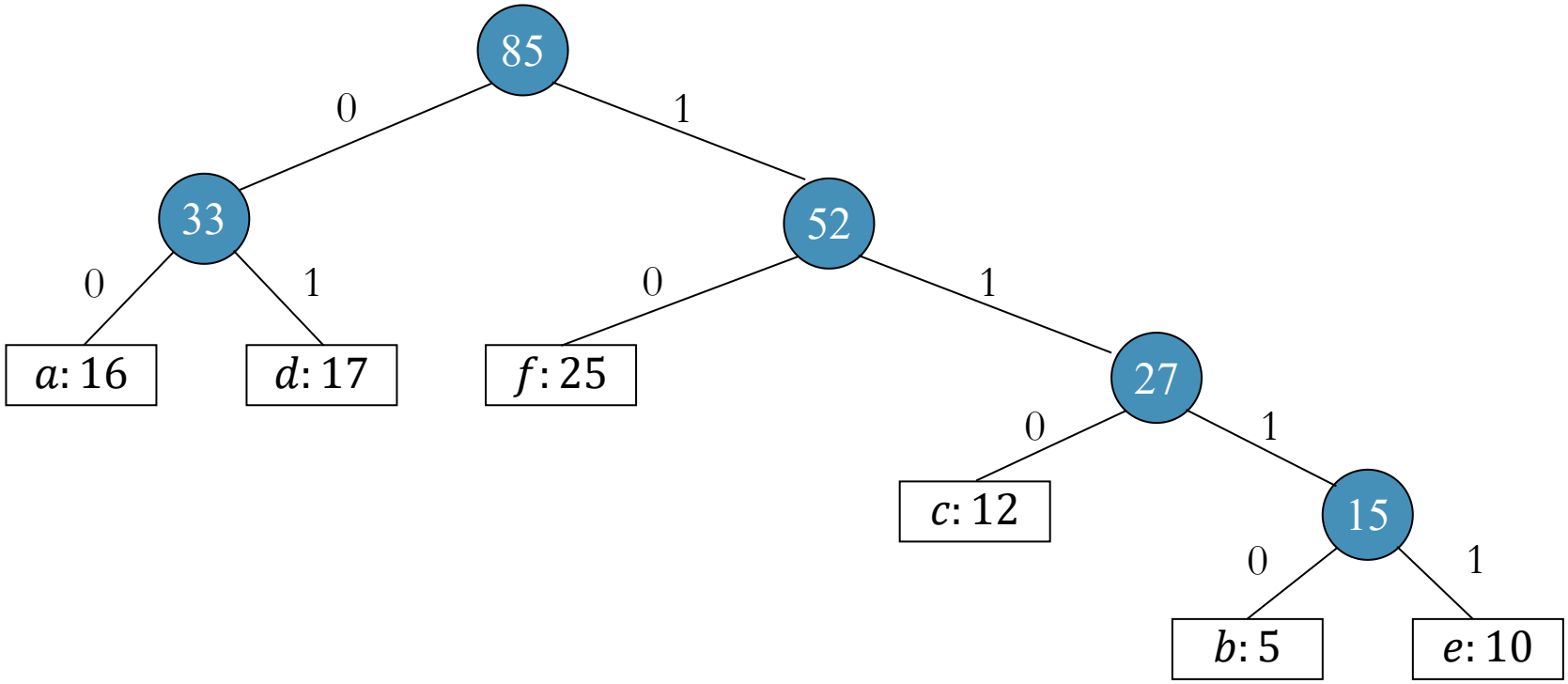
- Theorem 7.4: The greedy choice property.
- Theorem 7.5: The optimal substructure property
- Therefore, the Huffman algorithm produces an optimal prefix code.

Classroom Exercise （课堂练习）

Build the Huffman code for the following character frequencies.

Character	Frequecy
a	16
b	5
c	12
d	17
e	10
f	25

Classroom Exercise



Conclusion

After this lecture, you should know:

- What is greedy approach.
- What is greedy choice property.
- How to prove the correctness of a greedy algorithm.
- What is the difference between dynamic programming and the greedy approach.

Homework 课后习题

- Page 109-111

  7.2

  7.5

  7.8

  7.11

  课后习题和编程作业一起交

Experiment 1

如何使广告的收益最大化?

- Google的AdWords,或者其他搜索引擎的关键定广告, 使用的基本都是“关键字竞价” (或者称“关键字拍卖”) 的机制, 对每个用户搜索的关键定, 挑选为它竞价的广告来显示.
- 用户搜索的关键字到达搜索引擎次序无法预知, 每个竞价者为一个关键字出的价钱也千差万别, 竞价者还会对每天的花费总额有一个封顶的预算, 超过了这个预算, 即使有合适的关键字, 竞价者也不希望为它多花钱了.
- 搜索引擎们是通过什么样的规则来安排哪个广告给哪个关键字, 以最大化当天的收益的呢?

Experiment 1

- 有N个竞价者, 每个竞价者指定了一个最大预算b_i. 一共有M个不重复出现的关键字. 每个竞价者i对第j个关键字指定一个出价C_{ij}. 竞价开始后, 关键字序列 1, 2, ..., M实时到达, 每个关键字j必须实时分配给某个竞价者i的广告以赚取收益C_{ij}.
- 问题的目标是: 在满足竞价者对关键字匹配要求的基础上, 使总收益最大.

$$\max \sum_{i=1}^{N} \sum_{j=1}^{M} q_{ij} C_{ij}$$

$$\text{s.t.} \sum_{j=1}^{M} q_{ij} C_{ij} \leq b_i \quad \text{for } i = 1, \ldots, N$$

  其中$q_{ij} = 1$表示将关键字j实时分配给竞价者i, 否则为0.
- 请设计出一个贪心算法: 任给一个输入实例, 能输出总收益.
  - 测试案例自行设计, 尽可能覆盖各种情形.

Experiment 2

- 用贪心算法求解石材切割问题