Algorithms
Chapter 16
Greedy Algorithms

貪婪演算法

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **An activity-selection problem**
- Elements of the greedy strategy 贪婪演算法的組成
- Huffman codes 资料压縮最佳方法

Greedy Algorithms

- Similar to dynamic programming. 與动態規劃法相似
- Used for optimization problems.
  用來解決最佳化問題,使成本最小化,利益最大化
- **Idea:** When we have a choice to make, make the one that looks best right now. 每一次都選擇目前看起來最好的選擇
  - Make a locally optimal choice in hope of getting a globally optimal solution. 希望區域的最佳選擇會產生全域的最佳解
- Greedy algorithms don’t always yield an optimal solution. But sometimes they do. 並不會總是產生最佳解,但有時候會
  - We’ll see problems for which they do. 先看一個例子
  - Also, we’ll look at some general characteristics of when greedy algorithms give optimal solutions.

  再看當貪婪演算法產生最佳解時,所具備的一些性質

An activity-selection problem

- **Input:** A set $A = \{a_1, a_2, \ldots, a_n\}$ of n proposed activities.
  - Each activity a_i has a start time s_i and a finish time f_i, where $0 \leq s_i < f_i < \infty$. 包含起始時間與結束時間
- **Output:** A maximum set of compatible activities.
  - Activities a_i and a_j are **compatible** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. 這些活動必須相容(時間不衝突)
- For example: Consider the following set A, sorted by finish time.

	i	1	2	3	4	5	6	7	8	9	10	11
起始時間	s_i	1	3	0	5	3	5	6	8	8	2	12
結束時間	f_i	4	5	6	7	8	9	10	11	12	13	14

以結束時間排序(小→大)

- $\{a_3, a_9, a_{11}\}$ is a set of compatible activities. 沒有衝突,可放入同一集合
- $\{a_1, a_4, a_8, a_{11}\}$ is a maximum set of compatible activities.

Greedy templates

- **Earliest start time:**
  - Consider jobs in ascending order of s_i.
- **Earliest finish time:**
  - Consider jobs in ascending order of f_i.
- **Shortest interval:**
  - Consider jobs in ascending order of $f_i - s_i$.

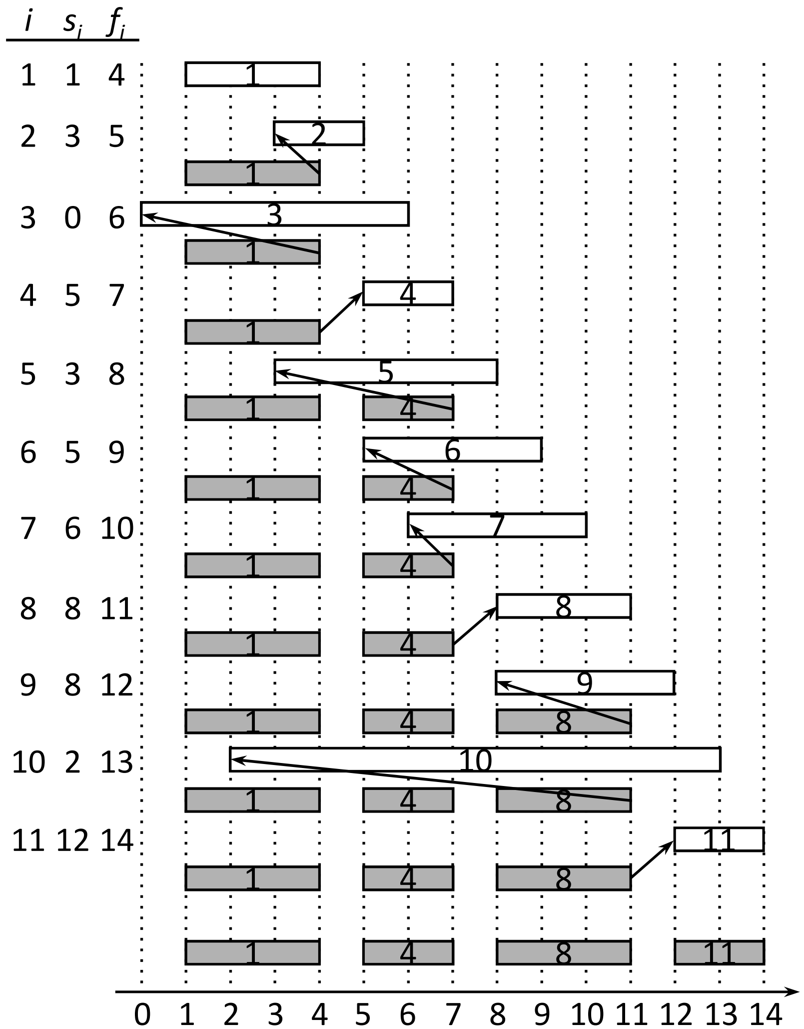
贪婪：用最少時間排入一個活動

GREEDY-ACTIVITY-SELECTOR pseudocode

```
GREEDY-ACTIVITY-SELECTOR(s, f)
    n ← length[s]
    A ← {a_1}
    i ← 1
    for m ← 2 to n
        do if s_m ≥ f_i
            then A ← A ∪ {a_m}
                i ← m
    return A
```

一個一個檢查

- s: array of start times.
- f: array of finish times.
- The input is sorted by f_i.



- Time: $O(n \lg n)$ to sort, $O(n)$ thereafter.

檢查：起始時間 > 結束時間，放入

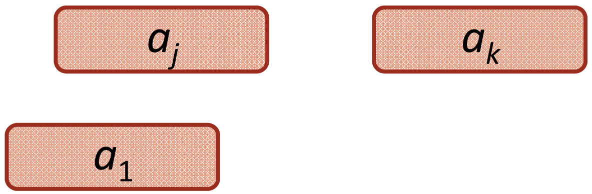
Correctness$_{1/3}$

- **Lemma 1** There exists an optimal activity selection contains a_1.

存在一個最佳解,此最佳解包含a_1

proof.

- Consider an optimal activity selection S. 設S為最佳的
- If $a_1 \in S$, then S is the desired selection. 若a_1在S中,得證
- Otherwise, let a_j be the activity in S with the smallest finish time.
  否則,設a_j是S中完成時間最早
- Every $a_k \in S - \{a_j\}$ has $s_k \geq f_j$.
  S中其他人的起始時間 > a_j完成時間
- So, $S - \{a_j\} \cup \{a_1\}$ is also a set of compatible activities.
  把a_j移掉,a_1放入也符合
- Thus, $S - \{a_j\} \cup \{a_1\}$ is an optimal selection.
  交換後個數不變,也為最佳解



Correctness$_{2/3}$

- **Theorem 2** Algorithm GREEDY-ACTIVITY-SELECTOR produces solutions of maximum size for the activity-selection problem.
- **proof.** 目標：|S| ≥ |T|
  - Induction on the number of $|A|$.
  - S = an activity selection by our algorithm. S是我們產生的
  - T = an optimal activity selection of A containing a_1.

    T是一組包含a_1的最佳解
- **The basis:**
  - $|A| = 1$. 只有一個活动
  - Clearly, $S = T = A$.

Correctness$_{3/3}$

- **Induction step:** 假設個數少於|A|都成立
  - Suppose our selection algorithm works for all sets of activities with less than $|A|$ activities (strong induction).
  - $A' = \{a_i \in S \mid s_i \geq f_1\}$. 將A中与$a_1$不相容者刪除，$a_1$也刪除
  - S' = our algorithm's selection for A'. 演算法对A'產生的解
  - By inductive hypothesis, S' is an optimal selection of A'.
  - By greedy method, $S = \{a_1\} \cup S'$.

歸納法告訴我們
S'是A'的最佳解

  - Let $T' = T - \{a_1\}$. Then $T' \subseteq A'$.
  - Therefore, $|T'| \leq |S'|$ by optimality of S'.
  - Hence, $|T| = |T'| + 1 \leq |S'| + 1 = |S|$. $\Rightarrow |T| \leq |S|$
  - Thus, S is an optimal selection of A.

T'也是A'的一組解，但不一定是最佳

Outline

- An activity-selection problem
- **Elements of the greedy strategy**
- Huffman codes

Elements of the greedy strategy

- **Greedy-choice property**
  - A globally optimal solution can be arrived at by making a locally optimal (greedy) choice. 全域的最佳解可藉由区域的最佳解達成
  - Typically show the greedy-choice property by what we did for activity selection.
    - Look at a globally optimal solution. 選擇一個最佳解
    - If it includes the greedy choice, done. 若包含区域最佳選擇,得證
    - Else, modify it to include the greedy choice, yielding another solution that's just as good.
    否則,用置換的方法由最佳解產生一個包含区域最佳選擇的最佳解
- **Optimal substructure**
  - An optimal solution to the problem contains within it optimal solutions to subproblems. 問題的最佳解包含子問題的最佳解

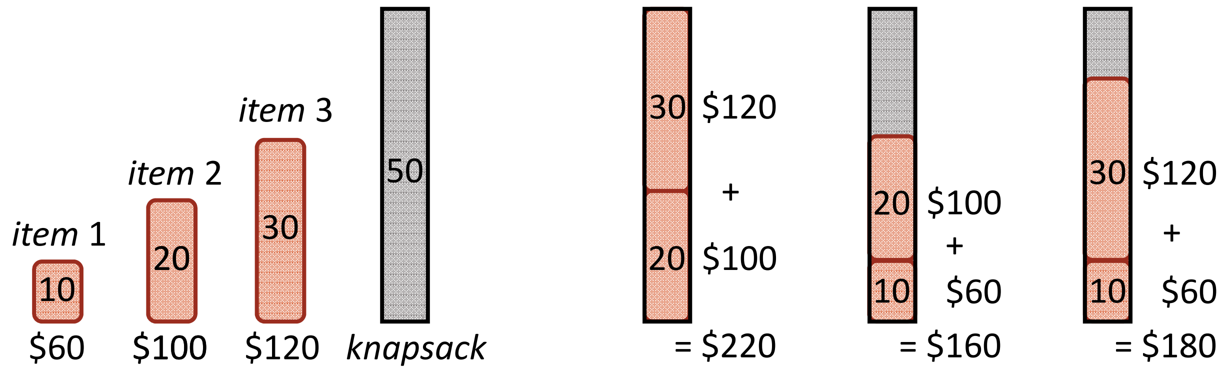
Greedy versus dynamic programming

- **Dynamic programming:**
  - Make a choice at each step.
  - Choice depends on knowing optimal solutions to subproblems.
  - Solve subproblems **first**. 先解決子問題
  - Solve **bottom-up**. 由下往上解
- **Greedy:**
  - Make a choice at each step.
  - Make the choice **before** solving the subproblems. 不管子問題，直接解
  - Solve **top-down**. 由上往下解

动態規劃法 = 填表法

0-1 knapsack problem-- using DP 背包問題

- **Input:** A set $A = \{a_1, a_2, \ldots, a_n\}$ of n items and a knapsack of capacity C.
  背包容量
  - Each item a_i is worth v_i dollars and weighs w_i pounds.
    各物品有其重量及價值
- **Output:** A subset of items whose total size is bounded by C and whose profit is maximized. 如何取有最大價值
  - **Each item must either be taken or left behind.**
    每個item只能選擇取或不取
- For example:



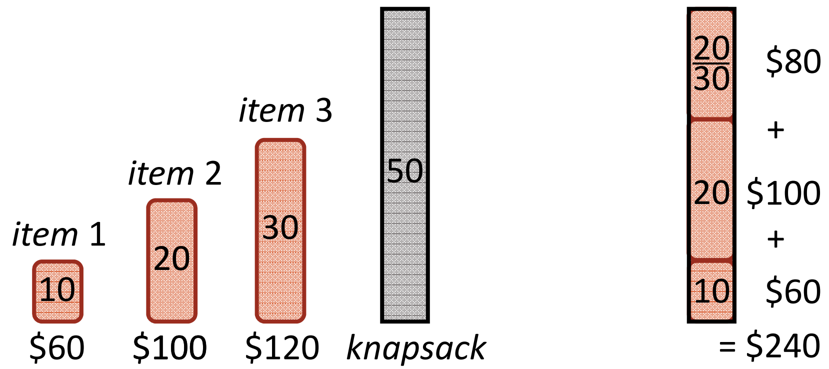
貪婪演算法

Fractional knapsack problem-- using greedy

- **Input:** A set $A = \{a_1, a_2, \ldots, a_n\}$ of n items and a knapsack of capacity C.
  - Each item a_i is worth v_i dollars and weighs w_i pounds.
- **Output:** A subset of items whose total size is bounded by C and whose profit is maximized.
  - **The thief can take fractions of items.**

可只拿 item 某部分
先取單位價值最高者

- For example:



FRACTIONAL-KNAPSACK pseudocode

```
FRACTIONAL-KNAPSACK(v,w,C)
    load ← 0
    i ← 1
    while load < C and i ≤ n
        do if w_i ≤ C − load
            then take all of item i
            else take (C − load)/w_i of item i
        add what was taken to load
    i ← i + 1
```

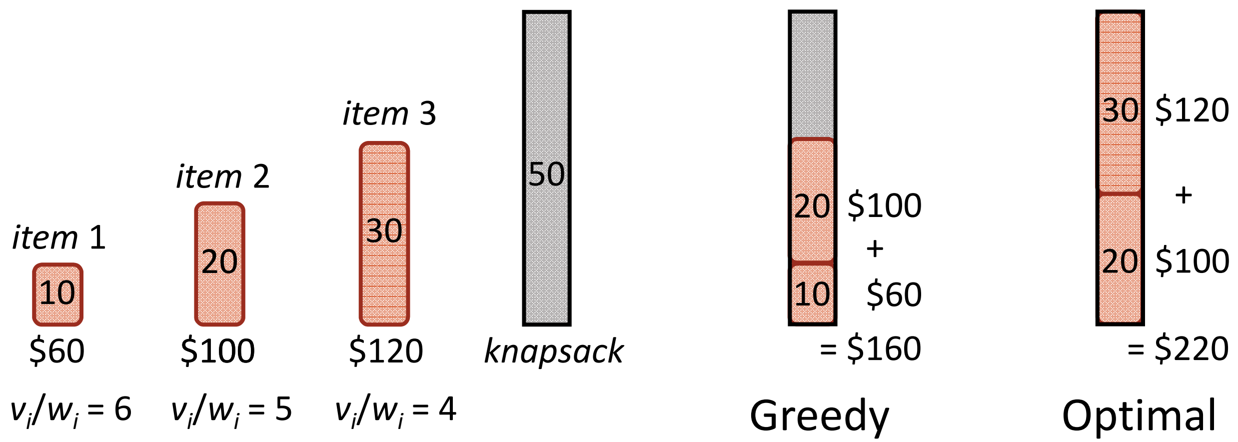
每次都拿單位價值最高者
直到滿為止

- v: array of values.
- w: array of weights.
- C: capacity
- The input is sorted by v_i / w_i.

- Time: $O(n \lg n)$ to sort, $O(n)$ thereafter.

Does greedy algorithm work for 0-1knapsack?

- Greedy doesn't work for the 0-1 knapsack problem.
- For example: 区域最佳選擇不会達到最佳



Outline

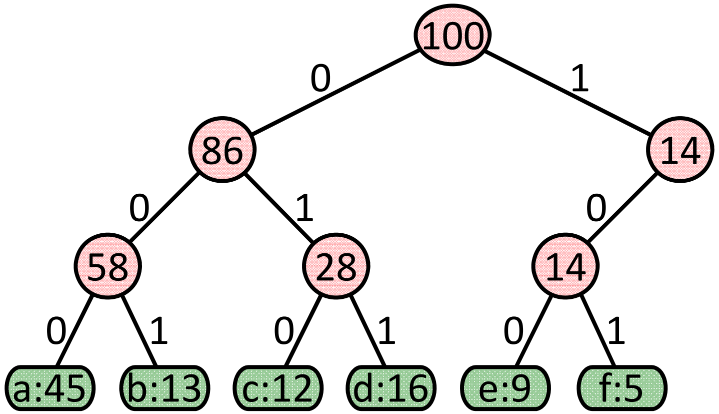
- An activity-selection problem
- Elements of the greedy strategy
- **Huffman codes**

Huffman codes 霍夫曼編碼：一種有效率的壓縮法

- A very effective technique for compressing data.
- A **prefix code** in which no codeword is also a prefix of some other codeword. 沒有人的編碼是別人的前半部
- An optimal prefix binary code.
- **Huffman coding problem**
  - **Input:** A alphabet $C = \{c_1, c_2, \ldots, c_n\}$ of n characters.
    - Each character c_i has a frequency $f_i > 0$.
  - **Output:** A prefix binary code for C with minimum cost.
    - The code is represented by a full binary tree. 用 leaves 表示字母
    - The leaves of the code tree represent the given characters.
    - $d_T(c)$ is the length of the codeword for character c.
    - The number of bits required to encode a file is $B(T) = \sum_{c \in C} f(c) d_T(c)$.

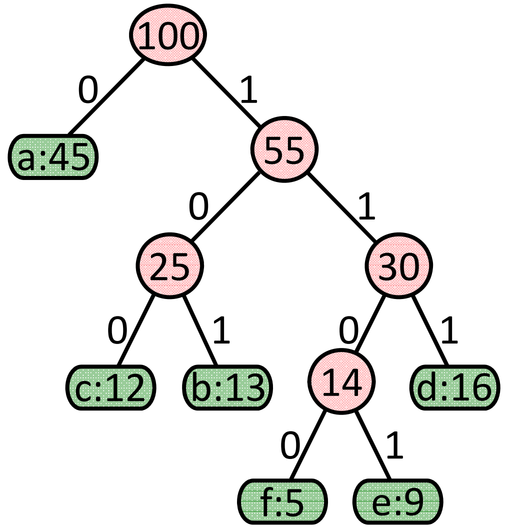
編碼長度 = 出現頻率 × 字母長度

An example



編碼長度固定

The tree corresponding to the fixed-length code a = 000, ..., f = 101.
The code is not optimal

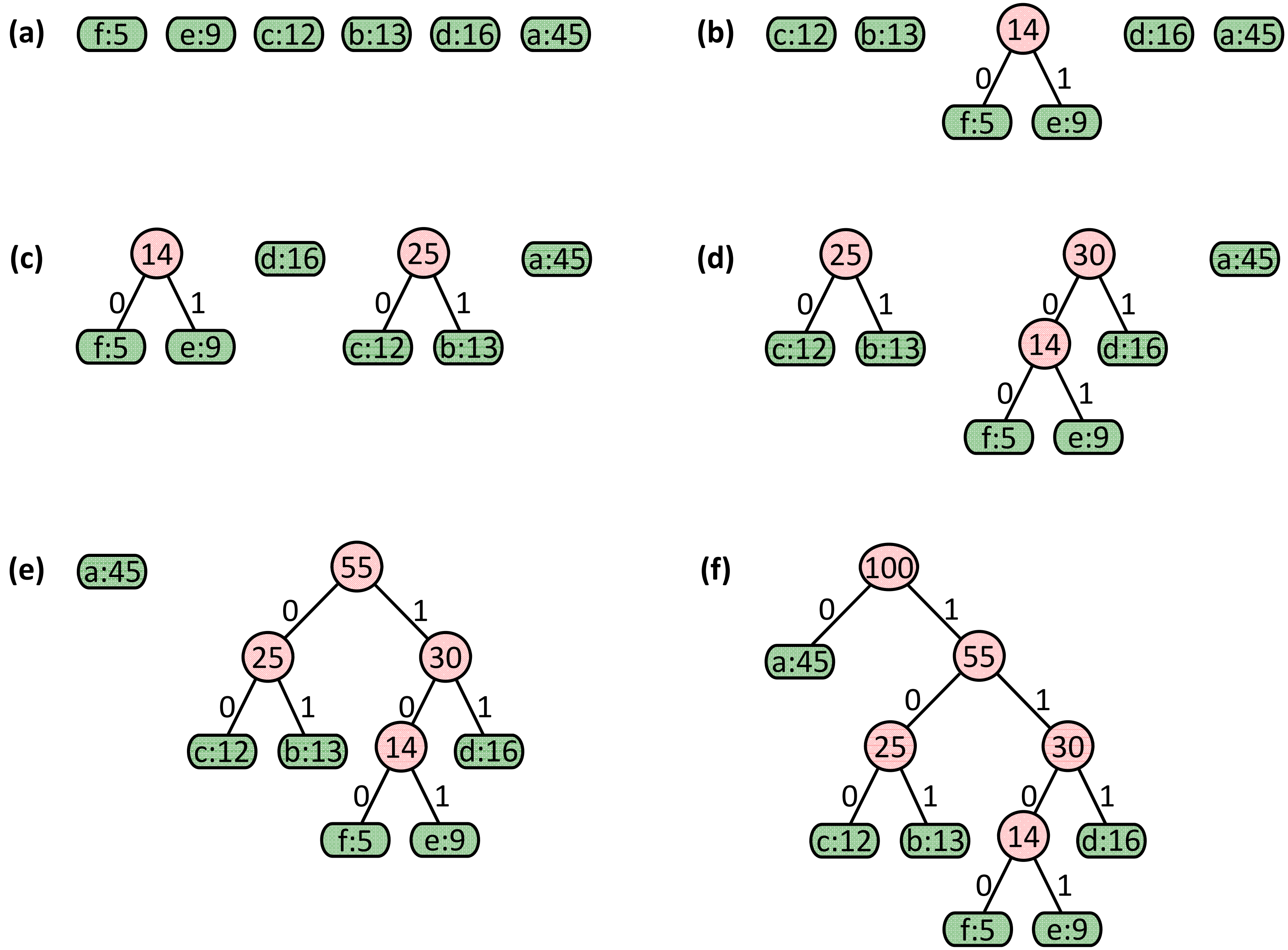


編碼長度不固定

The tree corresponding to the optimal prefix code a = 0, b = 101, ..., f = 1100.

永遠可以用 full binary tree 表示一個最佳編碼

(I)取出次數最少的兩個 ⇒ (II)合併 ⇒ (III)放入



HUFFMAN pseudocode

```
HUFFMAN(C)
1.        n ← |C|   } O(1)  c有n個
2.        Q ← C     } O(n)  建立 min queue
3.        for i ← 1 to n − 1
4.              do allocate a new node z
(I)   5.              left[z] ← x ← EXTRACT-MIN(Q)
      6.              right[z] ← y ← EXTRACT-MIN(Q)
(II)  7.              f [z] ← f [x] + f [y]
(III) 8.              INSERT(Q, z)
9.        return EXTRACT-MIN(Q)   /* Return the root of the tree. */  } O(1)
```

Lines 3–8: 調整 min queue 需花 O(lgn)

$(n-1) \cdot O(\lg n)$

- Line 2 initializes the min-priority queue Q with the characters in C.
- Time: $O(n \lg n)$.
- Correctness: omitted.