16.Greedy algorithms

Hsu, Lih-Hsing

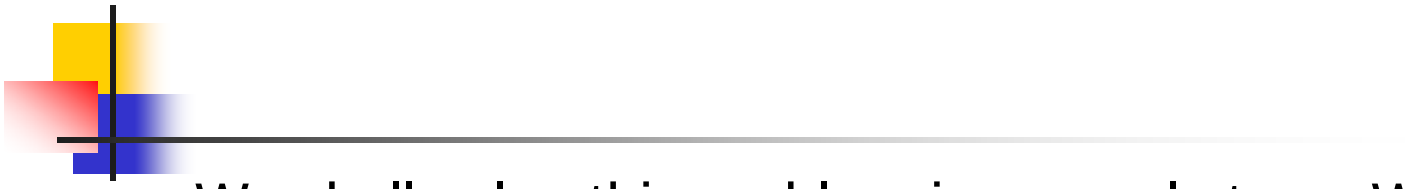
16.1 An activity-selection problem

- Suppose we have a set $S = \{a_1, a_2, ..., a_n\}$ of n proposed **activities** that with to use a resource. Each activity a_i has a **start time** s_i and a **finish time** f_i, where $0 \le s_i < f_i < \infty$.
- If selected, activity a_i take place during the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are **compatible** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap (i.e., a_i and a_j are compatible if $s_i \ge f_j$ or $s_j \ge f_i$).

- The activity-selection problem is to select a maximum-size subset of mutually compatible activities.

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



- We shall solve this problem in several steps. We start by formulating a dynamic programming solution to this program in which we combine optimal solutions to two subproblems to form an optimal solution to the original problem.
- We shall then observe that we need only consider one choice – the greedy choice – and that when we make the greedy choice, one of the subproblems is guaranteed to be empty, so that only one nonempty subproblem remains.

The optimal substructure of the activity-selection problem

- $S_{ij}=\{a_k \in S : f_i \le s_k < f_k \le s_j\}$,
  So that S_{ij} is the subset of activities in S that can start after activity a_i finishes and finish before activity a_j starts.
- $f_0=0$ and $s_{n+1} = \infty$. Then $S = S_{0,n+1}$, and the ranges for i and j are given by $0 \le i, j \le n+1$.
- $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

A recursive solution

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

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = 0 \\ \max_{i<k<j} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq 0 \end{cases}$$

Converting a dynamic-programming solution to a greedy solution

- ***Theorem 16.1***

  Consider any nonempty subproblem S_{ij}, and let a_m be the activity in S_{ij} with the earliest finish time:

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

  then

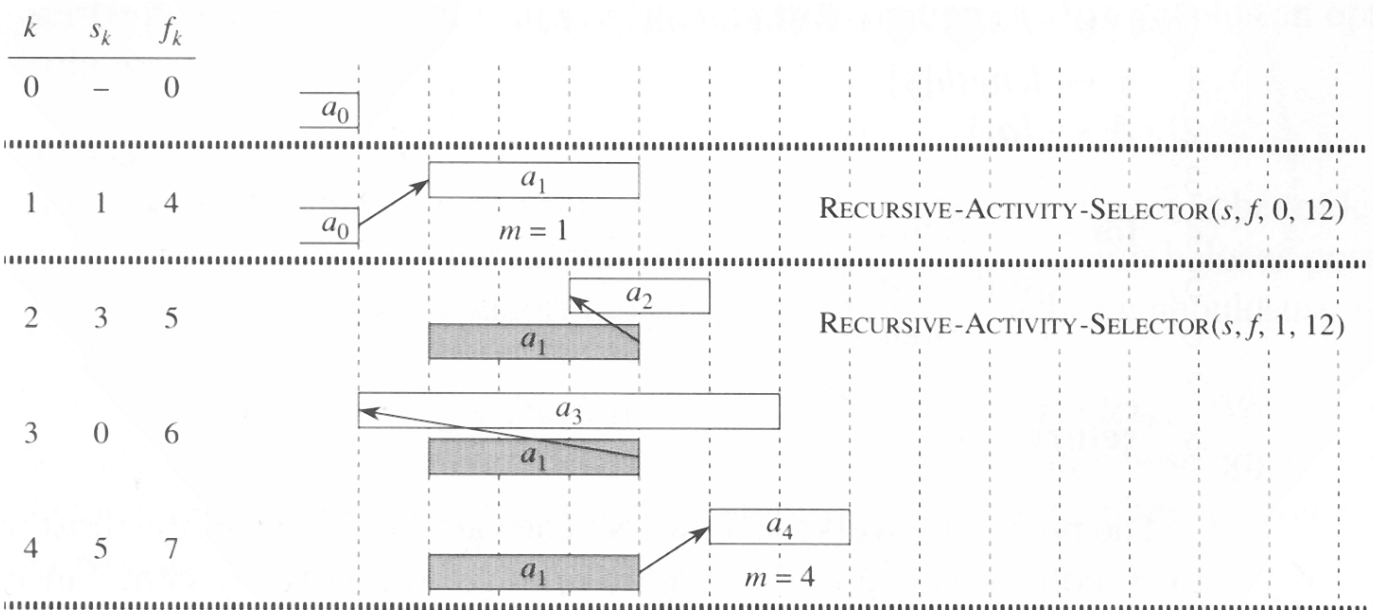
  1. Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.
  2. The subproblem S_{im} is empty, so that choosing a_m leaves the subproblem S_{mj} as the only one that may be nonempty.

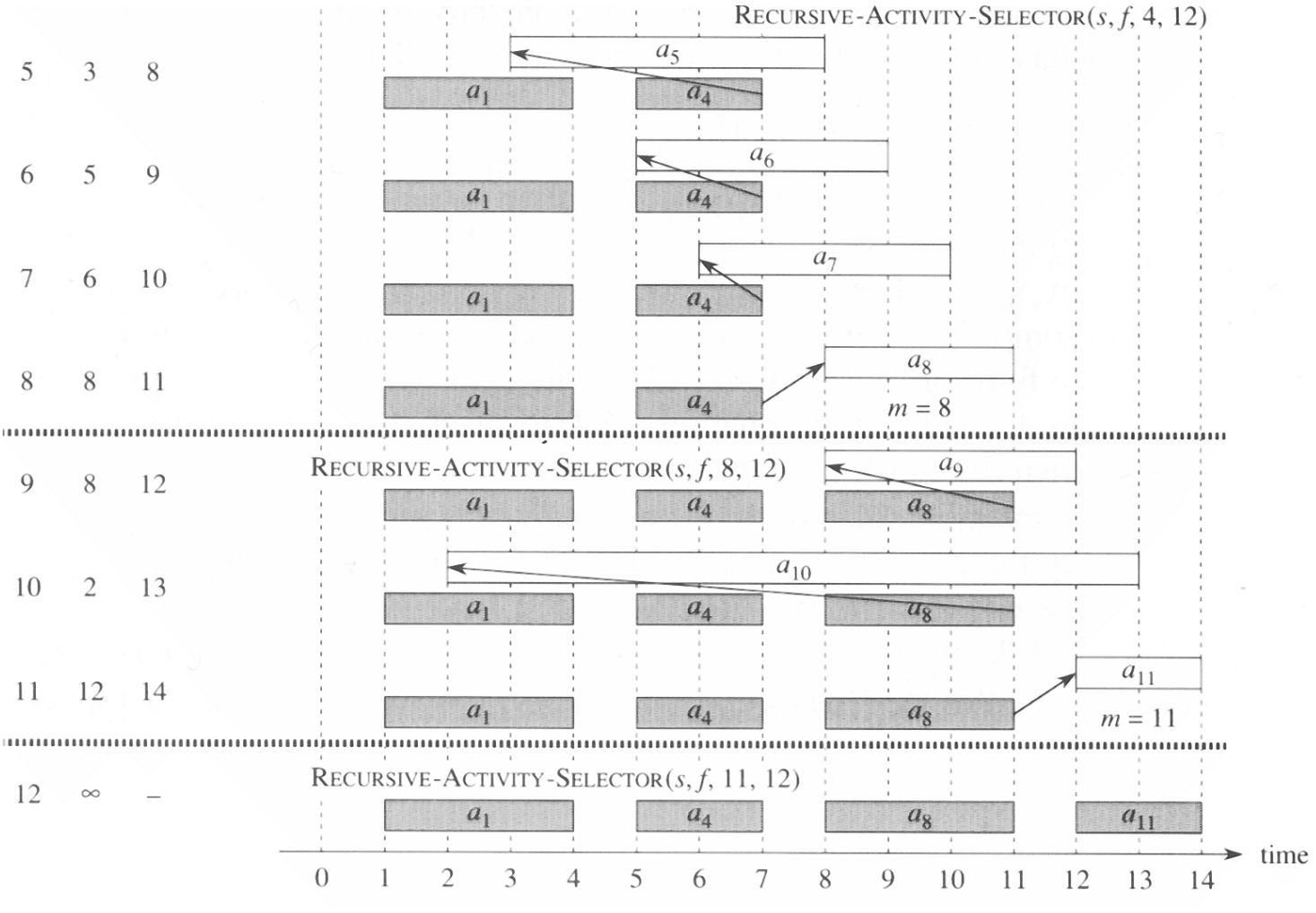
A recursive greedy algorithm

RECURSIVE-ACTIVITY-SELECTOR(s, f, i, j)

```
1  m ← i +1
2  while m < j and s_m < f_i        ▷Find the first activity in S_ij
3        do m ← m + 1
4  if m < j
5        then return {a_m}∪RAS(s, f, m, j)
6        else return 0
```

The operation of RECURSIVE-ACTIVITY-SELECTOR on the 11 activities given earlier





An iterative greedy algorithm

GREEDY-ACTIVITY-SELECTOR(s, f)

1 $n \leftarrow length[s]$
2 $a \leftarrow \{a_1\}$
3 $i \leftarrow 1$
4 **for** $m \leftarrow 2$ to n
5 **do if** $s_m \geq f_i$
6 **then** $A \leftarrow A \cup \{a_m\}$
7 $i \leftarrow m$
8 **return** A

16.2 Elements of the greedy strategy

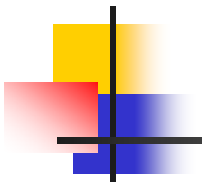
1. Determine the optimal substructure of the problem.
2. Develop a recursive solution.
3. Prove that at any stage of the recursion, one of the optimal choices is the greedy choice. Thus, it is always safe to make the greedy choice.

4. Show that all but one of the subproblems induced by having make the greedy choice are empty.
5. Develop a recursive algorithm that implements the greedy strategy.
6. Convert the recursive algorithm to an iterative algorithm.

- Alternatively, we could have fashioned our optimal substructure with a greedy choice in mind.

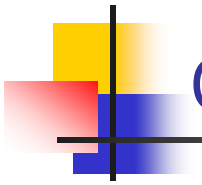
Designing a greedy algorithm

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, having made the greedy choice , 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.



- **Greedy-choice property:** A global optimal solution can be achieved by making a local optimal (optimal) choice.

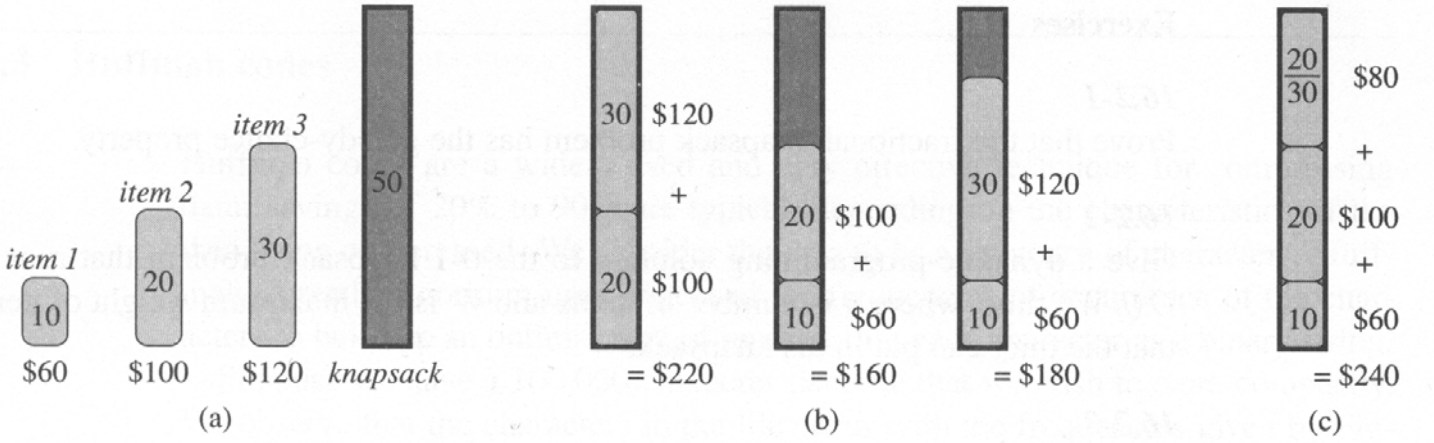
- **Optimal substructure:** An optimal solution to the problem within its optimal solution to subproblem.



Greedy versus dynamic programming

- 0-1 knapsack problem
- fractional knapsack problem
- (Exercise 16.2.2)

The greedy strategy does not work for the 0-1 knapsack



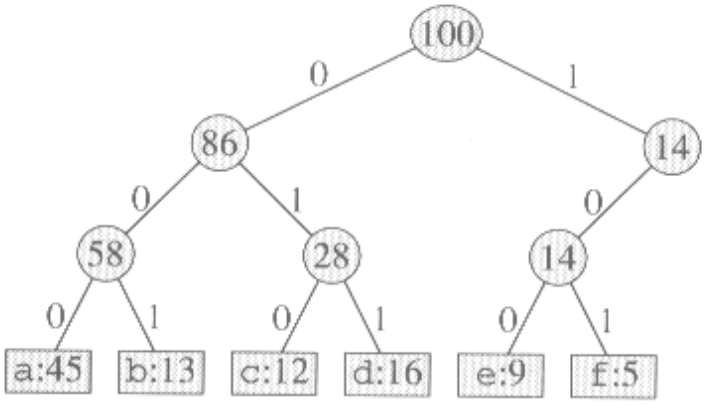
16.3 Huffman codes

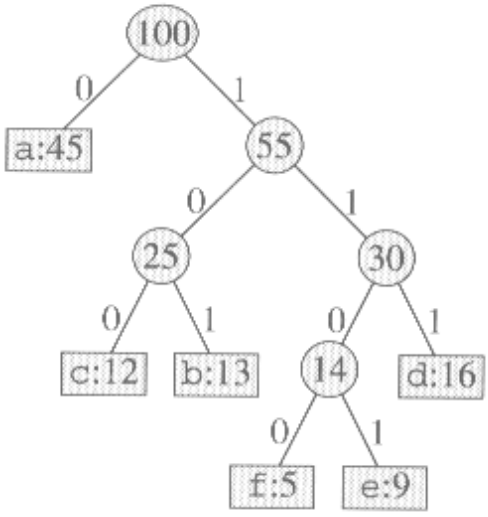
	a	b	c	d	e	f
Frequency (in hundred)	45	13	12	16	9	5
Fixed length codeword	000	001	010	011	100	101
Variable length codeword	0	101	100	111	1101	1100

- **Prefix code:** no codeword is also a prefix of some other codeword.

- Can be shown that the optimal data compression achievable by a character code can always be achieved with prefix codes.
- Simple encoding and decoding.
- An optimal code for a file is always represented by a binary tree.

Tree correspond to the coding schemes





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

cost of tree T

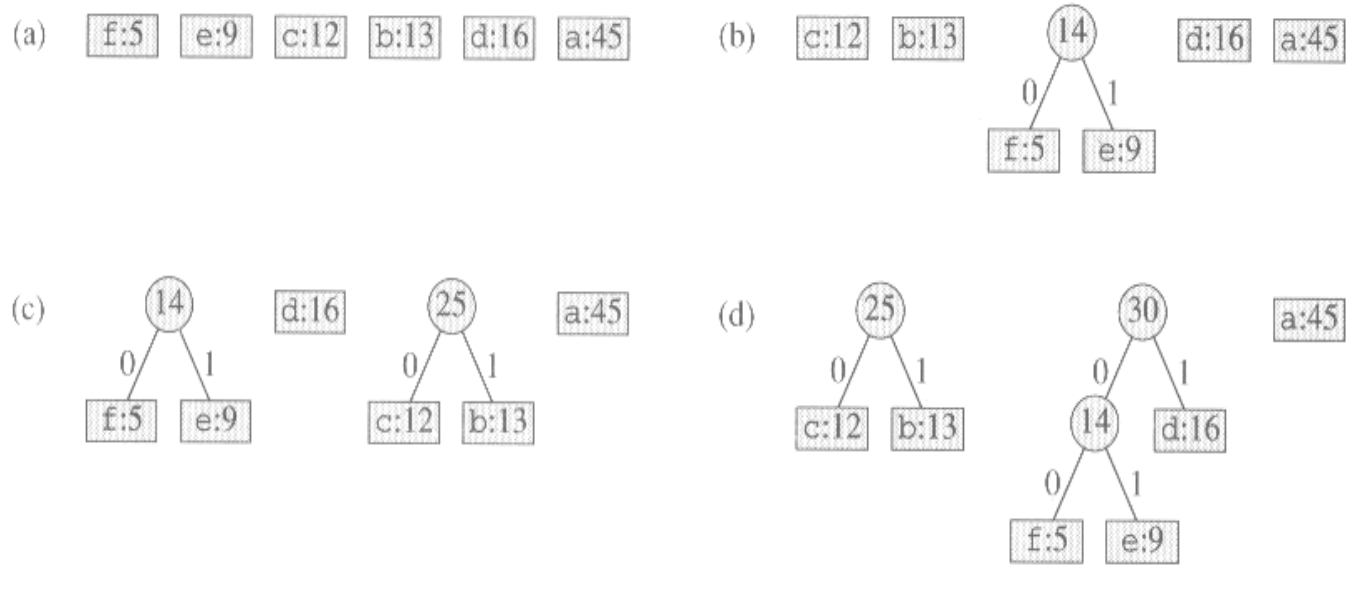
Constructing a Huffman code

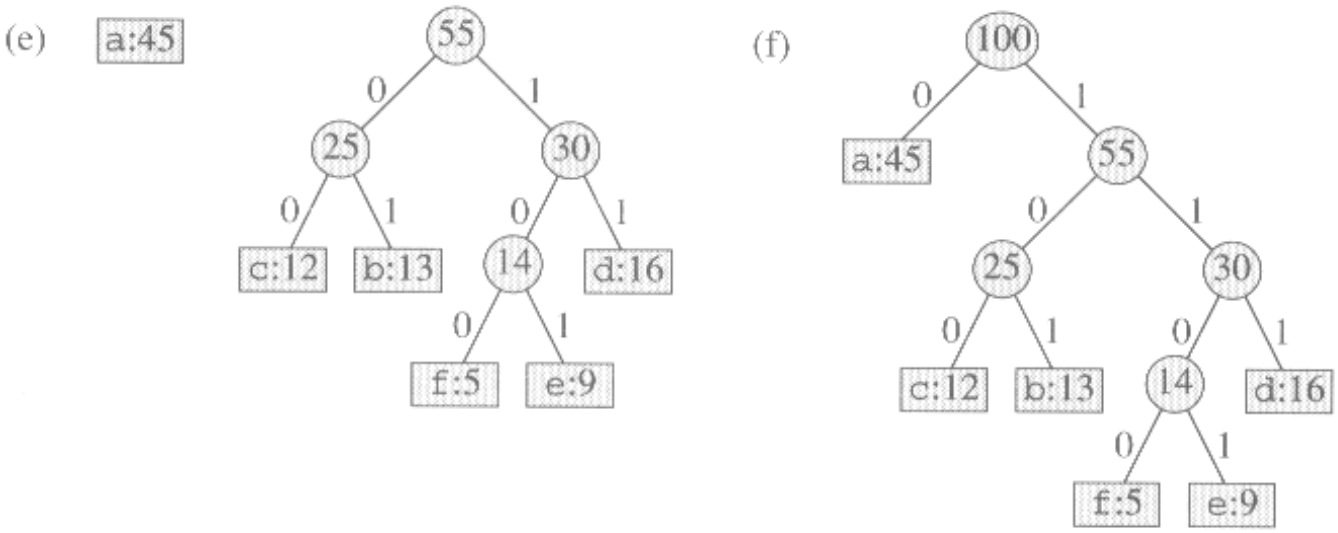
HUFFMAN(C)

```
1  n ← |C|
2  Q ← C
3  for i ← 1 to n – 1
4        do allocate a new node z
5           left[z] ← x ← EXTRACT-MIN(Q)
6           right[z] ← y ← EXTRACT-MIN(Q)
7           f[z] ← f[x] + f[y]
8           INSERT(Q,Z)
9  return EXTRACT-MIN(Q)
```

Complexity: $O(n \log n)$

The steps of Huffman's algorithm



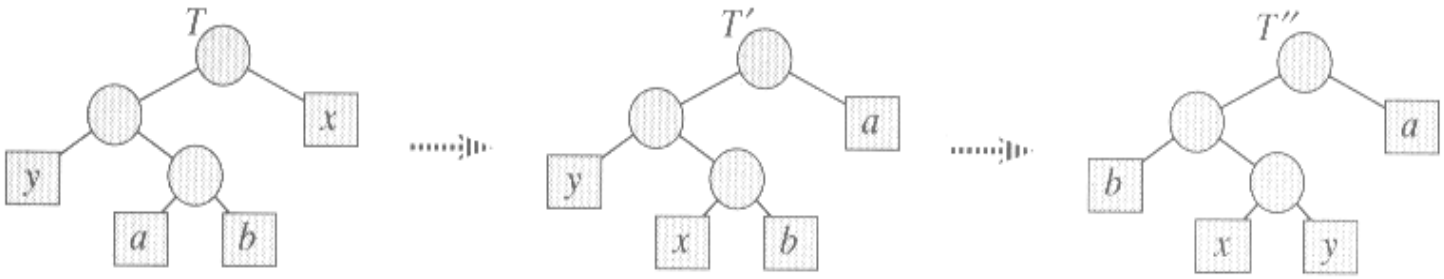


Correction of Huffman's algorithm

- **Lemma 16.2.**

  Let C be an alphabet in which $c \in C$ has frequency $f[c]$. Let x and y be the two characters in C having the lowest frequencies. Then there exists an optimal prefix code in C in which the codeword for x and y having the same length and differ only in the last bit.

Proof.



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

$$\ldots\geq 0$$

$$B(T)-B(T'')\geq 0$$

Lemma 16.3

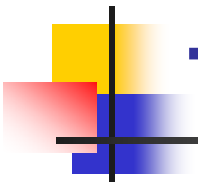
- Let C be a given alphabet with frequency $f[c]$ defined for each character $c \in C$. Let x and y be two characters in C with minimum frequency. Let C' be the alphabet C with characters x, y removed and character z added, so that $C' = C - \{x, y\} \cup \{z\}$; define f for C' as for C, except that $f[z] = f[x] + f[y]$.

Lemma 16.3 (cont)

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

- *Proof*

$$B(T) = B(T') + f[x] + f[y]$$



Theorem 16.4

- Procedure HUFFMAN produces an optimal prefix code.