Lemma: Let $T$ be an optimal prefix code over alphabet $C$. Consider any two characters $x$ and $y$ that appear as sibling leaves in $T$ and let $z$ be their parent. Then, the tree $T'=T-\{x,y\}$ is an optimal prefix code over alphabet $C'=C-\{x,y\}+\{z\}$ where the frequency of $z$ is $f[z]=f[x]+f[y]$.

Proof: Let's express $B(T)$ in terms of $B(T')$.

For each $c \in C-\{x,y\}$, $d_T(c) = d_{T'}(c)$, and thus,

$$f[c]d_T(c) = f[c]d_{T'}(c)$$

Since $d_T(x) = d_T(y) = d_{T'}(z) + 1$, we have

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

$$= f[z]d_{T'}(z) + (f[x] + f[y]) .$$

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

If $T'$ represents a nonoptimal code for $C'$, then there exists a tree $T''$ whose leaves are characters in $C'$ such that $B(T'')<B(T')$. Since $z$ is a character in $C'$, it is a leaf in $T''$. So, if we add $x$ and $y$ as children of $z$ in $T''$, then we get a code for $C$ with cost $B(T'')+f[x]+f[y]<B(T)$, a contradiction. Thus, $T'$ must be optimal for $C'$.
Theorem: There is an optimal code where the 2 lowest frequency characters x and y have codewords of same length and differing only in the last bit.

Proof: Suppose not. Let b and c be two characters that are sibling leaves of maximum depth in T. Assume \( f[b] \leq f[c] \) and \( f[x] \leq f[y] \).

\[
B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)
\]
\[
= f[x] d_T(x) + f[b] d_T(b) - f[x] d_{T'}(x) - f[b] d_{T'}(b)
\]
\[
B(T) - B(T') = f[x] d_T(x) + f[b] d_T(b) - f[x] d_{T'}(b) - f[b] d_{T'}(x)
\]
\[
B(T) - B(T') = f[b] (d_T(b) - d_T(x)) - f[x] (d_T(b) - d_T(x))
\]
\[
B(T) - B(T') = (f[b] - f[x]) (d_T(b) - d_T(x))
\]
\[
B(T) - B(T') \geq 0
\]
Huffman(C)
Q=C n=|C|
for i=1 to n-1
    z=allocate-node()
    left[z]=Extract-Min(Q)
    right[z]=Extract-Min(Q)
    f[z]=f[left[z]]+f[right[z]]
    Insert(Q,z)
return Extract-Min(Q)
encoding/decoding with fixed-length code
abbe=000 001 001 100 easy; 001010000101=001 010 000 101=bcaf easy
encoding/decoding with variable-length code
abbe=0 101 101 1101 easy; 10110001100=101 100 0 1100=bcaf careful

Prefix codes:
a code in which no codeword is also a prefix of some other codeword
decoding a prefix code is easy

One convenient representation of a prefix code is a binary tree

![Binary tree diagram]

given a tree T, the number of bits required to encode a file using T is B(T)

\[ B(T) = \sum_{c \in C} f(c) d_T(c) \]
Greedy strategy gives an optimal solution when the problem has
1) greedy choice property: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice
2) optimal substructure property: optimal solution to the problem contains within it optimal solutions to subproblems

Huffman codes (a data compressing technique)
Example: suppose we have a 100K-character data file and want to store it. There are 6 different characters in file, occurring with given frequencies.
We want to design a binary character code -- each character gets a unique binary string to represent it.

<table>
<thead>
<tr>
<th>characters</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency(\text{in K})</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>fixed-length code</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>variable-length code</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
<td>1100</td>
</tr>
</tbody>
</table>

fixed-length storage: 3*100K=300K
variable-length storage: 1*45K+3*13K+3*12K+3*16K+4*9K+4*5K=224K
Greedy-Activity-Selection(S)
A={1}        j=1
for i=2 to n
    if \( s_i \geq f_j \) then  A=A+\{i\} and j=i
return A

Proof that this algorithm selects the largest number of compatible activities:
First, we show that there exists an optimal solution with activity 1 in it.
Take any optimal solution B. If it includes 1, we are done.
If it doesn’t include 1, take the lowest numbered activity in B, say k.
Now define a new set of activities C=B-\{k\}+\{1\}.
Note that C includes activity 1, and |C|=|B|
Also, no activities in C conflict (since 1 completes before k).
Therefore, C is an optimal solution and it includes activity 1.

Once greedy choice of 1 is made, the rest of optimal solution must be
the best solution for the remaining activities that are compatible with 1,
i.e. if A is optimal solution to S, then A’=A-\{1\} must be an optimal
solution to \( S' = \{ i \in S: s_i \geq f_1 \} \). Thus, after greedy choice is made,
we are left with an optimization problem of the same form as the original.
TOPIC 13: Greedy Algorithms

always makes a choice that looks locally (currently) optimal in the hope that this choice will lead to a globally optimal solution.

Activity-selection problem
input: set of $n$ activities $S=\{1,2,\ldots,n\}$ that all want to use a resource
-- only one activity can use this resource at a time
-- want to schedule the largest number of activities
-- activity $i$ has start time $s_i$ and finish time $f_i$

Renumber activities in order of non-decreasing finishing times:

$$f_1 \leq f_2 \leq \ldots \leq f_n$$