Greedy Algorithm

Optimization Problem

1. Dynamic programming: A sequence of choices, carefully make choice.
2. Greedy approach: Make choice that looks the best at the moment, and it leads to optimal solution.

Activity-Selection problem

- Problem of scheduling a resource among several competing activities.
- Given a set $S = \{1, 2, \ldots, n\}$ of n proposed activities that wish to use a resource.
- Each activitiy i has a start time s_i and a finish time f_i, $s_i < f_i$ ($[s_i, f_i)$).

Activity-Selection problem

- Activities i and j are *compactable* if interval $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. (Or i and j are compactable if $s_i \geq f_j$ or $s_j \geq f_i$).
- The activity-Selection problem is to select a maximum-size set of mutually compactable activities.

Try to solve the problem
using the dynamic programming technique

- Define $S_{i,j} = \{a_k \in S | f_i < s_k < f_k \leq s_j\}$,
- $S_{i,j}$ a subset of activities in S,
  - start after activity a_i finishes,
  - finish before activity a_j starts.
- $S_{i,j}$ consists of all activities that are compatible with a_i, a_j.

To represent the entire problem

- Add activities a_0 and a_{n+1}, $f_0 = 0$, $s_{n+1} = \infty$
- Suppose that the activities are sorted in monotonically increasing order of finish time. $f_0 \leq f_1 \leq, \ldots, \leq f_n \leq f_{n+1}$, then $S_{i,j} = \emptyset$ when $i \geq j$.
- Thus we are looking for a maximum-size subset of mutually compatible activities from $S_{i,j}$, $0 \leq i < j \leq n+1$. (since all other $S_{i,j}$ are $\emptyset$)

The substructure of the optimal solution

- For a non-empty subproblem $S_{i,j}$
- suppose a solution to $S_{i,j}$ includes a_k, $f_i \leq s_k < f_k \leq s_j$
- using a_k generate 2 subproblems $S_{i,k}$ and $S_{k,j}$.
- Solution to $S_{i,j}$ = solution to $S_{i,k}$ + solution to $S_{k,j}$ + 1 (for a_k).
- Furthermore solutions to $S_{i,k}$ and $S_{k,j}$ must be optimal (otherwise cut and paste obtains a better solution.)

A recursive solution

- $c[i,j]$: the number of activities in the maximum-size subset of mutually compatible activities in $S_{i,j}$.
- $c[i,j] = 0$ whenever $S_{i,j}$ is $\emptyset$ (for $i \geq j$).
- $c[i,j] = c[i,k] + c[k,j] + 1$, if $S_{i,j}$ is not empty.
- Since we don't know k,
  $c[i,j] = \max_{i<k<j}\{c[i,k] + c[k,j] + 1\}$.

- Now build the table and the problem can be solved.
- But the theorem simplifies the solution

Theorem Consider a nonempty subproblem $S_{i,j}$ with earliest finish time $f_m = \min\{f_k | a_k \in S_{i,j}\}$. Then

1. Activity a_m is used in some maximum-size subset of mutually compatible activities of $S_{i,j}$.
2. The subproblem $S_{i,m}$ is empty, so that choosing a_m leaves the subproblem $S_{m,j}$ as the only one that may not be empty.

Proof Prove the first part.
Suppose $A_{i,j}$ is a maximum-size subset of mutually compactable activities of $S_{i,j}$.
Now we order the activities in no-decreasing finish time order, and let a_k be the first one.
If $a_k = a_m$, we are done
If $a_k \neq a_m$, we can construct $A'_{i,j} = A_{i,j} - \{a_k\} \cup \{a_m\}$ because $f_m \leq f_k$. Note that $A'_{i,j}$ has the same number of activities as $A_{i,j}$.
We have an optimal solution that contains a_m, the first one has the earliest finish time.

Prove the 2nd part.
If $S_{i,m}$ is not empty, we can find a_k s.t. $f_i \leq s_k < f_k < s_m < f_m$.
That means $a_k \in S_{i,j}$ and has an ealier finish time than a_m, contradict to our choice of a_m.
Based on the theorem, now we know the k.

Greedy Approach

- Assume that input activities are ordered by increasing finishing time $f_1 \leq f_2 \leq \ldots \leq f_n$, if not, sort them in $O(n \log n)$ time.
- iteratively select the next compactable one, i.e., (select the first one, then look for the first compactable one and so on.)
- Easy, scheduling is done in $\Theta(n)$ time.
- Is it optimal?

To show greedy approach solve the optimal solution

Theorem: Greedy approach produces solutions of maximum size of the activity-selection problem.
Proof: We first show there is one optimal solution that contains activity 1.
Let $S = \{1, 2, \ldots, n\}$ be the set of activities to schedule, f_i are in increasing order. Suppose that $A \subseteq S$ is an optimal solution.
Suppose that 1 is not in A. k is the activity that has the first finishing time. Since $f_k \geq f_1$, $B = A - \{k\} \cup \{1\}$ is also an optimal solution.

Once greedy approach choice of activity is made, the problem reduces to finding an optimal solution of those activities in S that are compactable with activity 1. That is if A is an optimal solution to the original problem S, then $A' = A - \{1\}$ is an optimal solution to $S' = \{i \in S : s_i > f_1\}$. (If we could find a solution B' in S' with more activities, than A', then adding activity 1 to B' get a better solution than A, a contradiction.)
Thus after greedy choice is made, we have an optimization problem of the same form as the original problem.
By induction on the number of choices made, making the greedy choice at every step produces an optimal solution.

0-1 Knapsack problem, fractional knapscak problem

A thief robbing a store, finds n items. ith items worth v_i dollars and weight w_i pounds. He can carry at most W pounds in his knapscak.

- 0-1 (whole or nothing) knapscak can not be solved using greedy approach.
- fractional knapsack (can take a fraction), compute the "value per pound" $\frac{v_i}{w_i}$, greedy approach calculates the optimal solution.

Huffman Code

- For Data Compression, saving of 20% to 90%.
- Use frequency of occurence to build up an optimal way to represent each character in binary string.

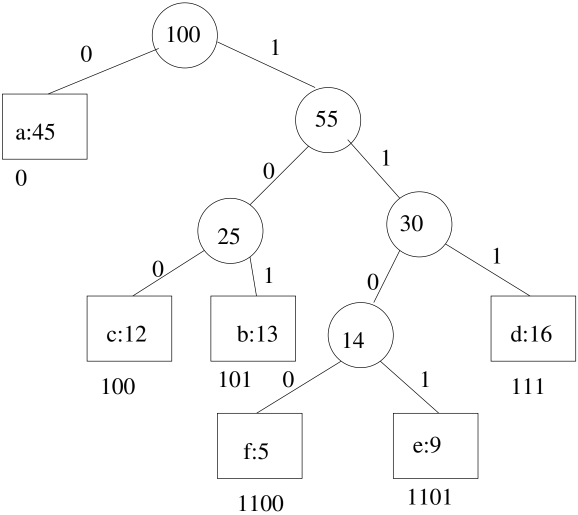
- 6 characters, `a`, `b`, `c`, `d`, `e`, `f`
- 3 bits are sufficient to present 6 characters.

	a	b	c	d	e	f
Freq. (k)	45	13	12	16	9	5
Fixed length	000	001	010	011	100	101
Variable length	0	101	100	111	1101	1100

- Fixed length, 3 bits for each char. $3\sum f_i = 300{,}000$.
- Variable length, $\sum l_i \cdot f_i = 224000$.

Prefix Code

- No codeword is a prefix of some other codewords.
- It can be shown that in optimal data compression, code is prefix code.
- $0 \cdot 101 \cdot 101 = 0101101$. If a code word is not prefix code, $01 \cdot 010 = 01010$, ambigious.


100
0
1
a:45
0
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
100
101
0
1
111
f:5
e:9
1100
1101

Huffman Tree

- leaves: chars.
- if there is a set of C chars, there are $|C|$ leaves, $|C| - 1$ internal nodes.
- 0 branching to left and 1 branching to right.
- to decode 00101111

Huffman Tree

- An optimal code for a file: represented by a *full binary tree* (every nonleaf node has two children).
- Given a tree T corresponding to a prefix code, the number of bits required to encode the file = the cost of the tree.
- Cost of the tree: $B(T) = \sum_{c \in C} f(c) d_T(c)$, where $d_T(c)$ the depth of the leaf representing c, $f(c)$ the frequency of occuring of c.
- It is optimal iff $B(T)$ is the least.

Constructing a Huffman Tree

- Invented by Huffman, a greedy approach
- Need a priority queue Q that stores the frequencies,
- delete the two smallest from Q,
- merge them and insert the merged back to Q
- iterate until the Q is empty.
- an example f:5, e:9, c:12, b:13, d:16, a:15

Lemma, Greedy Choice Property

Let C be an alphabet in which each char $c \in C$ has frequency $f(c)$. Let x and y be the two chars in C having the lowest frequencies. Then there exist 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. (x and y are sibling leaves)

Proof of the Greedy Choice Property

- T is an arbitrary optimal prefix code. Let b and c be two chars that are sibling leaves of maximum depth in T.
- Assume without loss of generality that $f[b] \leq f[c]$ and $f[x] \leq f[y]$.
- Since $f[x]$ and $f[y]$ are the smallest two, $f[x] \leq f[b]$ and $f[y] \leq f[c]$.

- In T, exchange b and x to produce T',
- then exchange c and y in T' to produce T''.

Exchange b and x, the difference between the costs of T and T',

$$\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x] d_T(x) + f[b] d_T(b) - f[x] d_{T'}(x) - f[b] d_{T'}(b) \\
&= f[x] d_T(x) + f[b] d_T(b) - f[x] d_T(b) - f[b] d_T(x) \\
&= (f[b] - f[x])(d_T(b) - d_T(x)) \\
&\geq 0
\end{aligned}$$

Lemma, Optimal substructure property

Let T be a full binary tree representing an optimal prefix code over an alphabet C, where freq $f[c]$ is defined for each character $c \in C$. Consider any two chars x and y that appear as sibling in T and let z be their parent. Then considering z as a char with freq $f[z] = f[x] + f[y]$.
The tree $T' = T$-$\{x, y\}$ representing an optimal prefix code for the alphabat $C' = C - \{x, y\} \cup \{z\}$.

Proof of the Optimal substructure property

- For each $c \in C - \{x, y\}$, $d_T(c) = d_{T'}(c)$, we have $f[c]d_T(c) = f[c]d_{T'}(c)$.
- Since $d_T(x) = d_T(y) = d_{T'}(z) + 1$ we have $f[x]d_T(x) + f[y]d_T(y) = (f[x] + f[y])(d_T[z] + 1) = f[z]d_{T'}[z] + (f[x] + f[y])$, or $B(T) = B(T') + (f[x] + f[y])$.
- If T' represents a non-optimal prefix code in C', there exists T'' whose leaves are chars in C' s.t. $B(T'') < B(T')$. Note that z is also a leaf in T''.
- If we add x and y as children of z in T'', then obtain a prefix code for C with cost $B(T'') + f[x] + f[y] < B(T)$, a contradiction.