

## Greedy Algorithms

Greedy algorithms is a technique for solving some optimization problems.

A greedy algorithm solution to a problem usually involves:

1. Repeatedly identify a decision to be made.
2. Make a locally optimal choice for each decision.

---

To reach a globally optimal solution, the problem must have an appropriate recursive structure, i.e., an optimal solution equals a locally optimal choice plus an optimal solution for the remainder of the problem.

## Activity Selection

A set of activities are to be scheduled in a room. An activity  $a$  has start time  $start[a]$  and a finish time  $finish[a]$ . Maximize the number of scheduled activities.

Locally optimal choice: Among activities with no conflicts with previously scheduled activities, choose the activity with the earliest finish time.

Recursive structure: This choice maximizes the amount of time afterwards, so no other choice can allow more activities to be scheduled.

$A$  is an array of activities.  $S$  is a schedule.

**GREEDY-ACTIVITY-SELECTOR**( $A$ )

  sort  $A$  by  $finish[A]$

$n \leftarrow length[A]$

$S \leftarrow \{A[1]\}$

$j \leftarrow 1$

**for**  $i \leftarrow 2$  **to**  $n$

**do if**  $start[A[i]] \geq finish[A[j]]$

**then**  $S \leftarrow S \cup \{A[i]\}$

$j \leftarrow i$

**return**  $S$

## Huffman Codes

A *binary character code* assigns a bit string to each character.

In a *prefix code*, no codeword is a prefix of another codeword.

A *Huffman code* is an optimal prefix code for a sequence of characters.

An example application is pack.

---

Example prefix codes:

|           | a  | b    | c   | d   | e  | f    |
|-----------|----|------|-----|-----|----|------|
| Frequency | 11 | 3    | 7   | 5   | 13 | 2    |
| Code 1    | 00 | 100  | 101 | 110 | 01 | 111  |
| Code 2    | 00 | 1110 | 10  | 110 | 01 | 1111 |

Locally optimal choice:

Combine the two least frequent characters into a single pseudo-character, i.e., they will have the same prefix, followed by a 0 for one character and 1 for the other.

Recursive structure:

An optimal prefix code corresponds to a full binary tree with characters as leaves.

Less frequent characters should have equal or greater depth in the tree.

Less frequent sets of characters should have equal or greater depth in the tree.

Let  $C$  be a set of characters with a *freq* field.

HUFFMAN( $C$ )

$n \leftarrow \text{length}[C]$

create a priority queue  $Q$  from  $C$  using  $\text{freq}[c]$

**for**  $i \leftarrow 1$  **to**  $n - 1$

**do**  $z \leftarrow \text{ALLOCATE-NODE}()$

$\text{left}[z] \leftarrow \text{EXTRACT-MIN}(Q)$

$\text{right}[z] \leftarrow \text{EXTRACT-MIN}(Q)$

$\text{freq}[z] \leftarrow \text{freq}[\text{left}[z]] + \text{freq}[\text{right}[z]]$

        INSERT( $Q, z$ )

**return** EXTRACT-MIN( $Q$ )