**Greediness**

**Reading:**  *CLRS* 16.1 – 16.3; *CLR* 17.1 – 17.3

**Hill Climbing**

You are trying to climb to the top of a mountain in a dense fog. A *greedy* strategy is to take each step in the direction of the highest gradient.

Whether or not the greedy strategy finds the top of the mountain depends on the terrain. Explain.

**Activity Selection Problem**

Given a set of activities for a given room with start and stop times, schedule maximal number of activities. E.g.

![Activity Selection Diagram](image)

An (optimal) greedy strategy: schedule activities by stop times:

```
Schedule-Activities(acts) \triangleright acts is a list of activities, each with start/finish times
sorted ← sortByFinish(acts) \triangleright sort acts by finish time
Greedily-Schedule(sorted)
```

```
Greedily-Schedule(sorted) \triangleright sorted is a list of activities sorted by finish time.
if Empty?(sorted) then
    return Empty \triangleright Return the empty list
else
    let first ← Head(sorted)
    in sorted ← Tail(sorted)
    while start(Head(sorted)) < finish(first) do
        sorted ← Tail(sorted)
    return Prepend(first, (Greedily-Schedule(sorted)))
```
Greediness Doesn’t Always Pay Off

Show that the following alternative greedy strategies for activity selection are not optimal for our example:

- Schedule activities by choosing those with the earliest start times (if two have the same start time, choose the shorter one).

- Schedule activities by choosing those with the shortest duration (if two have the same duration, choose the earlier one).

Coin Changing

You want to make change for a given amount using a minimal number of coins that range over a given list of denominations (e.g., [25,10,5,1]). A greedy strategy is to choose for the next coin the largest denomination that is less than the remaining amount.

\[
\text{Greedy-Make-Change(amount, denoms)}\\
coins \leftarrow \emptyset \quad \text{This is a bag, not a set.}\\
\text{while amount > 0 do}\\
\quad \text{if Empty?(denoms) then}\\
\quad \quad \text{error "Greedy strategy fails"}\\
\quad \text{let}\ h = \text{Head(denoms)}\ \text{in}\\
\quad \text{if } h \leq \text{amount then}\\
\quad \quad \text{coins} \leftarrow \text{coins} \cup \{h\}\\
\quad \quad \text{amount} \leftarrow \text{amount} - h\\
\quad \text{else}\\
\quad \quad \text{denoms} \leftarrow \text{Tail(denoms)}\\
\text{return coins}
\]

The greedy strategy correctly solves the problem some denomination lists (e.g., [25,5,1] and [25,10,5,1]) but not for others (e.g., [25,10,1]). Show the latter fact for 30 cents.

It turns out that for any integer \(c > 1\), Greedy-Make-Change returns an optimal result for the denomination list \([c^0, c^1, c^2, \ldots, c^k]\). (See CLRS Problem 16-1/CLR Problem 17-1.)
Generic Greedy Algorithm

A greedy algorithm makes the locally optimal choice at every step:

\[
\text{Greedy}(\text{problem}) \\
\text{soln} \leftarrow \emptyset \\
\text{subproblem} \leftarrow \text{problem} \\
\text{while not Solution?(soln, problem) do} \\
\quad \text{choice} \leftarrow \text{Greedy-Choice(subproblem)} \\
\quad \triangleright\text{Greedy-Choice makes locally optimal choice for current subproblem.} \\
\quad \text{soln} \leftarrow \text{choice} \cup \text{soln} \\
\quad \text{subproblem} \leftarrow \text{Simplify(subproblem, choice)} \\
\quad \triangleright\text{Simplify gives remaining subproblem after choice is made.} \\
\text{return soln}
\]

Proving Greedy Algorithms Optimal

As shown above, greedy algorithms are not always optimal.

But when a greedy algorithm is optimal, optimality can be shown via the following two properties:

**Greedy Choice Property**: An optimal solution can begin with the greedy choice.
(Note: there may be many optimal solutions.)

**Optimal Substructure Property**: The optimal solution for the whole problem can be derived from the optimal solution for the parts.

This is usually shown as follows. Assume that you are given an optimal solution $S$ for the whole problem and show that you can extract the optimal solution $S_i$ for subproblem $i$ from the optimal solution to the whole problem. The proof proceeds by contradiction: assume that there is a better solution $S'_i$ to subproblem $i$ and show that you could construct a better whole solution $S'$ using $S'_i$. But that contradicts your original assumption about the optimality of $S$. So it must be the case that your other assumption is incorrect: $S'_i$ cannot be better than $S_i$. That is, each $S_i$ is optimal.
Proving Greedily-Schedule is Optimal

Prove that Greedily-Schedule is optimal by showing that it satisfies the following two properties:

**Greedy Choice Property** : Suppose that $as$ is a list of activities sorted by finish time, and that $gs$ is an optimal schedule for this list, also sorted by finish time. Show that $gs_1$ can be replaced by $as_1$ and still be optimal.

**Optimal Substructure Property** : Suppose that $as$ is a list of activities sorted by finish time, and that $gs$ is an optimal schedule for this list, also sorted by finish time. For any $k$, where $1 \leq k \leq length(gs)$, show the following two facts:
- $[gs_1, \ldots, gs_{k-1}]$ is an optimal solution to activity selection for all elements of $as$ with finish times $\leq start(g_k)$;
- $[gs_{k+1}, \ldots, gs_{length(gs)}]$ is an optimal solution to activity selection for all elements of $as$ with start times $\geq finish(g_k)$.

Knapsack Problems

Given items with integer values and weights, fill a knapsack with weight capacity $W$ so that it carries the most valuable load.

*Greedy strategy:* Take the most valuable item whose weight still fits in the knapsack.

*Example:* Consider a knapsack with weight capacity 50 and items with the following (value,weight) pairs: $[(60,10), (100,20), (120,30)]$

1. *0/1 knapsack problem:* must take entire item. Greedy strategy is not always optimal.

2. *Fractional knapsack problem:* can take fraction of an item. Greedy strategy is optimal.