Greedy Algorithms

Another Optimization Technique

Greedy Algorithms

Used for some types of optimization.

- Question: you have to choose several things in a sequence so that the result is, in some sense, optimal. How do you make your choices?
- Solution: choose one that comes closest to meeting your goal (locally optimal choice). If you are still short of the goal, choose the next best (next locally optimal choice), etc.

Paradigm: Time, activities and resources.

1. We are given a set of activities that require exclusive use of a common resource (e.g., classroom?).
   \[ S = \{ a_1, ..., a_n \} \]
2. Each activity \( a_i \) is associated with a time period \([s_i, f_i)\), where \( s_i \) is the start time and \( f_i \) is the finish time. Note the half-open character of the time periods: we don't want any overlap, even by one point.
3. GOAL: select the largest possible set of non-overlapping (= mutually compatible) activities. Other possible goals based on time-in-use, fees-per-use, etc…

Example

How to proceed: Sort \( S \) by finish time.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>( f_i )</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>
**Greedy Algorithms**

**Question:** what are the maximum size mutually compatible sets of activities?

......

**Optimal substructure activity selection.**

**Def.:**

\[ S_{ij} = \{ a_k \in S \mid f_i \leq s_k < f_j \leq s_j \} \]

= activities that start after \( a_i \) finishes and finish before \( a_j \) starts.

The activities in \( S_{ij} \) are compatible with

- All activities that finish by \( f_i \)
- All activities that start no earlier than \( s_j \)

Add two "bookend" activities: \( a_0 = (-\infty, 0) \), \( a_{n+1} = (\infty, \infty+1) \).

**Greedy Algorithms**

We have to construct an algorithm to solve the problem, and we need to set up a few steps.

**Define** \( S = S_{0,n+1} \).

**Range** for \( S_{ij} \) is \( 0 \leq i, j \leq n+1 \).

Assume activities sorted monotonically by increasing finishing time: \( f_0 \leq f_1 \leq \ldots \leq f_n \leq f_{n+1} \).

**Claim:** \( i \geq j \Rightarrow S_{ij} = \emptyset \).

**Pf.:** assume not and get a contradiction.

\[ \Rightarrow \text{We only need to look at } S_{ij} \text{ with } 0 \leq i < j \leq n+1. \]

**Greedy Algorithms**

**Setting up the recursion.**

Suppose a solution to \( S_{ij} \) involves \( a_k \). We have two subproblems:

- \( S_{ik} \) - start after \( a_i \) finishes, finish before \( a_k \) starts.
- \( S_{kj} \) - start after \( a_k \) finishes, finish before \( a_j \) starts.

A full solution to \( S_{ij} \) must consist of

\( \text{(solution to } S_{ik}) \cup \{ a_k \} \cup \text{(solution to } S_{kj}) \)

Since \( a_k \) is in neither subproblem and the new subproblems are disjoint

\[ |\text{soln. to } S_{ij}| = |\text{soln. to } S_{ik}| + 1 + |\text{soln. to } S_{kj}| \]
Greedy Algorithms

Recursive Solution

Let \( c[i, j] \) = size of max-size subset of mutually compatible activities in \( S_{i, j} \).

Recall: \( i \geq j \Rightarrow S_{i, j} = \emptyset \Rightarrow c[i, j] = 0. \)

If \( S_{i, j} \neq \emptyset \) and \( a_k \in S_{i, j} \),

\[
c[i, j] = c[i, k] + 1 + c[k, j].
\]

Since we don’t know which \( k \) to use:

\[
c[i, j] = \begin{cases} 0 & \text{if } S_{i, j} = \emptyset, \\ \max_{i \leq k < j, a_k \in S_{i, j}} \{ c[i, k] + c[k, j] + 1 \} & \text{if } S_{i, j} \neq \emptyset. \end{cases}
\]

\[
\text{Theorem: Let } S_{i, j} \neq \emptyset, \text{ and let } a_m \in S_{i, j} \text{ have the earliest finish time:}
\]

\[
f_m = \min\{ f_k \mid a_k \in S_{i, j} \}. \text{ Then}
\]

1. \( a_m \) is used in some maximum-size subset of mutually compatible activities of \( S_{i, j} \).
2. \( S_m = \emptyset \), so that choosing \( a_m \) leaves \( S_m \) as the only non-empty subproblem.

\text{Pf.}

2.) Assume \( \exists a_k \in S_{i, j} \Rightarrow f_i \leq s_k < s_m < f_k \leq f_m \Rightarrow f_k < f_m. \)

Then \( a_k \in S_{i, j} \) and it has an earlier finish time than \( f_m \), which contradicts our choice of \( a_m \), so \( S_m = \emptyset \).

1.) Let \( A_{i, j} \) be a max-size subset of mutually compatible activities in \( S_{i, j} \). Order the activities in \( A_{i, j} \) in monotonically increasing order of finish times. Let \( a_i \) be the first activity in \( A_{i, j} \). If \( a_i = a_m \), we are done, since \( a_m \) is used in a max-size subset. If not, let \( A'_{i, j} = (A_{i, j} \setminus \{a_i\}) \cup \{a_m\} \).

Claim: activities in \( A'_{i, j} \) are disjoint.

\text{Pf.} Activities in \( A_{i, j} \) are disjoint and \( a_i \) is the first activity in \( A_{i, j} \) to finish. But then \( f_m \leq f_{a_m} \), and \( a_m \) does not overlap anything else in \( A'_{i, j} \). QED Claim.

Since \( |A'_{i, j}| = |A_{i, j}| \) and \( A_{i, j} \) is maximal, so is \( A'_{i, j} \).

What have we done:

Before the theorem:

# of subproblems in optimal solution: 2
# of choices to consider: \( j - i - 1 \).

After the theorem:

# of subproblems in optimal solution: 1
# of choices to consider: 1.
Greedy Algorithms

A top-down solution:
To solve a problem $S_{i,j}$:
- Choose $a_m \in S_{i,j}$ with earliest finish time.
- Then solve $S_{i',j'}$.

The subproblems are:
- Original problem is $S_{0,n+1}$.
- Suppose first choice is $a_{m_1}$.
- Next subproblem is $S_{m_1,n+1}$.
- And so on…

Each subproblem is of the form $S_{i,n+1}$, the last (remaining) activities to finish.
The subproblems chosen have increasing finishing times, so we consider each activity only once, in increasing order of finishing time.

Recursive Algorithm: time $\Theta(n)$.

```
REC-ACTIVITY-SELECTOR(s, f, i, n)
    m ← i + 1
    while m ≤ n and $s_m < f_i$ do
        do m ← m + 1
    if m ≤ n
        then return $a_m \cup$ REC-ACTIVITY-SELECTOR($s$, $f$, $m$, $n$)
        else return $\emptyset$
```

Initial call: REC-ACTIVITY-SELECTOR($s$, $f$, 0, $n$).

Iterative Algorithm: time $\Theta(n)$.

```
GREEDY-ACTIVITY-SELECTOR($s$, $f$, $n$)
    A ← [$a_1$]
    i ← 1
    for $m$ ← 2 to $n$
        do if $s_m \geq f_i$
            then $A \leftarrow A \cup \{a_m\}$
            i ← m
        return A
```

Where $a_i$ is the most recent addition to $A$. 

$a_i$ is most recent addition to $A$. 

return $A$
Greedy Algorithms

- **NOTE**: we need to sort the activities by finishing time: $\Theta(n \log n)$, if done via comparisons. So this is the most time consuming part….
- **Question**: how many maximal solutions does the original problem have?
- **Question**: if we want to maximize the time-in-use rather than the number of activities, does the greedy strategy work?