Greedy Algorithms (Ch 16, L #7, HW #4)

We come to the next paradigm for algorithm creation.

We illustrate the notion of a “greedy algorithm” with an example (for which it does not guarantee correct results — however, it illustrates aspects of greedy algorithm much better than most.)

Imagine you are at the cafeteria at UTD (I realize during Covid, this is out-of-place).

And you order something, pay cashier, with a large bill and the cashier owes you $2.89 back. Here is what a normal cashier does (without thinking) automatically.

For this consider denominations of US coins

Arranged in decreasing order of value:

<table>
<thead>
<tr>
<th>Coin Value</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1.00</td>
<td>2</td>
</tr>
<tr>
<td>$0.50</td>
<td>1</td>
</tr>
<tr>
<td>$0.25</td>
<td>1</td>
</tr>
<tr>
<td>$0.10</td>
<td>1</td>
</tr>
<tr>
<td>$0.05</td>
<td>1</td>
</tr>
<tr>
<td>$0.01</td>
<td>4</td>
</tr>
</tbody>
</table>

Total 9

[Decisions made at previous steps are irrevocable]
An example to show this does not always use minimum number of coins (the "goal" of the algorithm).

Consider denomination shown below & need 0.40

- 0.25 x 1
- 0.20 x 2
- 0.10 x 1
- 0.05 x 1
- 0.01 x 3

GA

Optimal solution.

So in this section, proof of correctness is very important since we are interested in valid algorithms. Now we go over some example, when this paradigm works. When it works, it produces algorithm with low W.C. Complexity.

We begin with Activity Selection (16.1).

Input: given an n activities with a start time \( s_i \) and a finish time \( f_i \): \( s_i < f_i \) \( i = 1 \ldots n \)

Def: \( i \) and \( j \) overlapping if \( \exists t \in [s_i, f_i) \cap [s_j, f_j) \) (a common interior point)
Desired output

A subset \( S \subseteq \{1, 2, \ldots, n\} \), such that no two in \( S \) are overlapping, and \( \max |S| \).

Max. non-overlapping set of activities

(Ex: Max # of classes that can be assigned to the same classroom given the Schedule)

For those who have Graph Theory background, this is a special case of finding longest path in a directed acyclic graph.

A numerical Example [Page 414 CLRS]

\[ n = 11 \]

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
31 & 30 & 5 & 3 & 5 & 6 & 8 & 8 & 2 & 12 \\
4 & 5 & 6 & 7 & 9 & 9 & 10 & 11 & 12 & 14 & 16 \\
\end{array}
\]

\( 3, 9, 11 \) are non-overlapping

\( 0, 2 \) are overlapping.

There are several possible candidates for greedy algorithms: Some valid; Some invalid.

We will explore a few and show examples of each kind.
Each is based on a greedy principle (in red)

a) Select the job with least duration ($f_i - \delta_i$) and remove those that overlap with it & repeat the process with remaining jobs at each step.

b) Select the job with $\min \delta_i$, remove that overlap with it, repeat the process at each step with remaining jobs.

c) Select the job with $\max \delta_i$.

d) Select the job with $\min f_i$.

e) Select the job with $\max f_i$.

f) Select the job that overlaps with fewest number of jobs, remove overlapping jobs and repeat the process at each step with remaining jobs.

We have only listed a few possibilities; some of these are valid and others are not.

---

(a) is invalid: example when it does not work:

\[
\begin{array}{c}
\text{GA (a): \{1\}} \\
\text{Opt: \{1,3\}} \\
\end{array}
\]

---

We now show (d) is valid. For first we show how this is executed in 2 different ways. Then prove correctness, check complexity etc.
Algorithm I: Activities Sorted: $f_1 < \cdots < f_n$, $O(n \log n)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$k_i$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

$x: \exists i < f_i$

$x: \exists i < f_4$

$x: \exists i < f_8$

$A = \{0, 4, 8, 11\}$

a max size non-overlapping set

Work after initial Sorkiy $O(n^2)$

Two nested loops

Overall: $O(n^2)$

There is a slightly better way of doing the work

After Sorkiy [see page 421]

1. $n \leftarrow \text{length}[S]$
2. $A \leftarrow \{0\}$
3. $k \leftarrow 1$
4. for $m = 2$ to $n$
5. if $k_m \geq f_k$
6. then $A \leftarrow A \cup \{k\}$
7. $k \leftarrow m$
8. Return $A$

We show evolution (trace) of Alg II on next page for same example
Algorithm II

activities sorted so $f_1 \leq f_2 \ldots \leq f_n$ \(\Theta(n \log n)\)

\[
\begin{array}{c|c|c|c}
 k & i & \beta_i & f_i \\
\hline
 & & & \\
1 & 4 & 1 & 4 \\
2 & 5 & 3 & 5 \\
3 & 6 & 0 & 6 \\
4 & 7 & 5 & 7 \\
5 & 8 & 3 & 9 \\
6 & 9 & 5 & 9 \\
7 & 10 & 6 & 10 \\
8 & 11 & 8 & 11 \\
9 & 12 & 8 & 12 \\
10 & 13 & 2 & 14 \\
11 & 14 & 12 & 16 \\
\end{array}
\]

\[A = \{0\}, \quad A = \{0, 4\}, \quad A = \{0, 4, 8\}, \quad A = \{0, 4, 8, 11\}\]

The two algorithms result in same output.

\[\text{Need to prove this}\]

May

However, complexity of AlgII (after sorting) is \(O(n)\).

Our loop

Overall complexity \(O(n \log n)\)

Now proof of correctness: We will do it for AlgI, [One way to prove AlgII, is after proof AlgI, show AlgI & AlgII have same output]
Proof of Correctness of Alg. I

Assume activities are sorted in increasing order of \( f_i \) and renumbered so that \( f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n \)

Algorithm I (and II) include activity 1 at the first step. Our goal is to find a subset \( S \) of \((\max \text{ Size})\) of nonoverlapping activities. We want to claim that including activity 1 is not an incorrect decision.

To this end:

**Lemma 1.** There exists a max. size non-overlapping set that includes activity 1.

**Proof:** By Contradiction.

Suppose 1 is not any max. size non-overlapping set; let \( A \) be such a set, and \( 1 \not\in A \).

Let \( k = \min j : j \neq 1 \) by supposition \( j \not\in A \).

Several possibilities:

1. \( 1 \not\in A \):

   \[ A \cup \{1\} \text{ is non-overlapping} \]

   \[ \therefore A \text{ is max. size} \]

2. \( 1 \in A \):

   \[ A' = A - \{1\} \]

   \[ |A'| = |A| - 1 \leq \text{max. size} \]

   \[ 1 \in A', 1 \not\in A \text{ contradicts} \]

In each, \( f_1 \leq f_k \)
So our first step will not lead us astray.

Now we focus on only those max size nonoverlapping set that contain 1 [We know by lemma 1, such a set is also overall max size set]

Any such set can contain any activity that overlaps with 1. So removing such activities does not affect the problem.

At this stage we only have 1 and those that do not overlap with 1 remaining

\[
\begin{align*}
\{ & 8, f, 1 \\
\end{align*}
\]

activities that do not overlap with 1

Smaller set than initial one

Largest non overlapping set is 1 or largest non overlapping among them

Now the proof of the main algorithm is completed using induction.

Alg II is valid since output is same as Alg I

(Exercise for you)