Scope for Quiz 5

- Chapter 6 of DPV.
- Sections 7.1

Practice Problems

Problem 1. When you get to the prize phase of the TV game show Take It or Leave It™, they wheel out a table with piles of money, numbered 1 through \( n \), where pile \( i \) has \( P[i] \) many dollars (\( P[i] > 0 \)). You can take any set of piles you want, subject to the rule:

\[
\text{If you take pile } i, \text{ then you cannot take pile } (i - 1) \text{ or pile } (i + 1).
\]

Let's say a clutch is a set of pile numbers that obeys the rule. We want to find optimal clutches: that is, clutches worth as much money as possible. Example: Suppose \( n = 5 \) and \( P[1] = 300 \), \( P[2] = 800 \), \( P[3] = 600 \), \( P[4] = 300 \), and \( P[5] = 600 \). Then \( \{2,5\} \) and \( \{1,3,5\} \) are both clutches. The value of \( \{2,5\} \) is: \( P[2] + P[5] = 1400 \). The value of \( \{1,3,5\} \) is: \( P[1] + P[3] + P[5] = 1500 \). It turns out that \( \{1,3,5\} \) is an optimal clutch for \( P[1..5] \).

a. Give an example to show that the following algorithm sometimes fails to find an optimal clutch for \( P[1..n] \). (Hint: There is an example with \( n = 3 \).)

```
clutch ← ∅;
candidates ← \{1, ..., n\};
while candidates ≠ ∅ do
    pick an \( i \) ∈ candidates such that \( P[i] \) is maximal
    clutch ← (clutch ∪ \{ i \})
    candidates ← (candidates − \{ (i − 1), i, (i + 1) \})
return clutch
```

b. Give an example to show that the following algorithm sometimes fails to find an optimal clutch for \( P[1..n] \). (Hint: There is an example with \( n = 4 \).)

```
clutch_0 ← the even numbers in \{1, ..., n\}
clutch_1 ← the odd numbers in \{1, ..., n\}
if (the value of clutch_0) > (the value of clutch_1)
    then return clutch_0 else return clutch_1
```

Parts c through i below develop a dynamic programming algorithm to find an optimal clutch for a given \( P[1..n] \). The trick is to compute \( V[i, j] \) for each \( i \) and \( j \) with \( 1 \leq i \leq j \leq n \), where

\[
V[i, j] = \text{the value of an optimal clutch for } P[i..j].
\]

c. Why is \( V[i, i] = P[i] \)?

d. Why is \( V[i, i + 1] = \max(P[i], P[i + 1]) \)?

e. Why it is the case that, for \( i + 2 \leq j \):
   If some optimal clutch for \( P[i..j] \) omits \( i \), then \( V[i, j] = V[i + 1, j] \).

f. Why it is the case that, for \( i + 2 \leq j \):
   If every optimal clutch for \( P[i..j] \) includes \( i \), then \( V[i, j] = V[i] + V[i + 2, j] \).

g. Using parts e and f, explain why

\[
V[i, j] = \max(V[i] + V[i + 2, j], V[i + 1, j])
\]

when \( i + 2 \leq j \).

h. Using parts c, d, and g, give a dynamic programming algorithm that, given \( P[1..n] \), finds the value of an optimal clutch set for \( P[1..n] \). A memoized recursive solution is fine.

i. Suppose you are given the correctly filled in \( V \) table. Show how to use it to construct an optimal clutch for \( P[1..n] \). (Hint: Use parts c, d, and g again—particularly g.)

Problem 2. Set the following up as a linear programming problem. Be sure to explain what the variables mean. You do not have to solve the problem. (The problem is from: [http://www.sonoma.edu/users/w/wilsonst/Courses/Math_131/lp/default.html](http://www.sonoma.edu/users/w/wilsonst/Courses/Math_131/lp/default.html))

A farmer has 10 acres to plant in wheat and rye. He has to plant at least 7 acres. However, he has only $1200 to spend and each acre of wheat costs $200 to plant and each acre of rye costs $100 to plant. Moreover, the farmer has to get the planting done in 12 hours and it takes an hour to plant an acre of wheat and 2 hours to plant an acre of rye. If the profit is $500 per acre of wheat and $300 per acre of rye how many acres of each should be planted to maximize profits?


An answer to 1c. As there is just one pile of money—you simply take it.

An answer to 1d. As there are two piles of money and the rule implies that you can take only one, then taking the larger pile is clearly the right thing to do.

An answer to 1e. If $i$ is not used in some optimal clutch for $P[i..j]$, then the value of that optimal clutch must be $V[i + 1, j]$.

An answer to 1f. If every optimal clutch for $P[i..j]$ uses $i$, then no optimal clutch uses $i + 1$. Hence, $V[i, j] = P[i] + V[i + 2, j]$.

An answer to 1g. Either
(i) some optimal clutch for $P[i..j]$ omits $i$, or
(ii) every optimal clutch for $P[i..j]$ uses $i$.
If (i) holds, $V[i, j] = V[i + 1, j]$ by part e.
If (ii) holds, $V[i, j] = P[i] + V[i + 2, j]$ by part f.
Hence, $V[i, j] = \max(P[i] + V[i + 2, j], V[i + 1, j])$.

Some possible answers to 1h. Here is a memoized version.

```plaintext
function findOpt(P[1..n])
    array V[1..n, 1..n]
    for i ← 1 to n do
        for j ← 1 to n do
            V[i, j] ← P[i]
        end for
    end for
    function eval(V, i, j) /* Compute V[i, j] where i ≥ j */
        if V[i, j] = 0 then
            if i = j then
                V[i, j] ← P[i]
            else if i + 1 = j then
                V[i, j] ← max(P[i], P[j])
            else
                V[i, j] ← max(eval(V, i + 1, j), P[i] + eval(V, i + 2, j))
            end if
        end if
        return V[i, j]
end function
```

Here is a straight iterative version.

```plaintext
function findOpt'(P[1..n])
    array V[1..n, 1..n]
    for i ← 1 to n do
        V[i, i] ← P[i]
        for j ← 1 to n − i do
            V[i, j] ← max(V[i + 1, j], P[i] + V[i + 2, j])
        end for
    end for
    function optClutch(V[1..n, 1..n])
        clutch ← ∅;
        i ← 1
        while i ≤ n − 2 do
            if V[i, i] ≠ V[i + 1, i] then
                add i to clutch;
                i ← i + 2
            else
                i ← i + 1
            end if
            if ((n − 2) ∈ clutch or P[n − 1] ≤ P[n]) then
                add n to clutch
            else
                add n − 1 to clutch
            end if
        end while
        return clutch
    end function
```

An answer to 1i. By parts e and f, when $i + 2 ≤ j$, $V[i, j] ≠ V[i + 1, j]$ means that $i$ must be in any optimal clutch for $P[i..j]$; and $V[i, j] = V[i + 1, j]$ means that you can omit $i$ from an optimal clutch.
Some Reference Math Facts

a. $a^n \cdot a^m = a^{n+m}$.
b. $a^{m \cdot n} = (a^m)^n = (a^n)^m$.
c. $a^n \cdot b^n = (a \cdot b)^n$.
d. $\log_a a^n = n$.
e. $a^{\log_a n} = n$.
f. $c \cdot \log_a a = \log_a a^c$.
g. $\log_a (a \cdot b) = (\log_a a) + (\log_a b)$.
h. $(\log_a x) / (\log_a b) = \log_b x$.
i. $a^{\log_a c} = c^{\log_a a}$.
j. $\sum_{k=1}^n k = \frac{1}{2} \cdot n \cdot (n+1)$.
k. $\sum_{k=0}^n a^k = \frac{a^{n+1} - 1}{a-1}$.
l. $\sum_{k=0}^\infty 2^{-k} = 1$.
m. $\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$.
n. $\sum_{k=1}^\infty k \cdot 2^{-k} = 2$.
o. For $a > 1$ and $b, c > 0$:
   \[
   \lim_{n \to \infty} \frac{(\log_a n)^b}{n^c} = 0.
   \]
p. For $a > 1$ and $b, c > 0$:
   \[
   \lim_{n \to \infty} \frac{n^b}{n^c} = 0.
   \]
q. If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$, then:

<table>
<thead>
<tr>
<th>$f(n) \in O(g(n))$</th>
<th>$f(n) \in \Theta(g(n))$</th>
<th>$f(n) \in \Omega(g(n))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = 0$</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>$0 &lt; c &lt; \infty$</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>$c = \infty$</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

r. The Master Recurrence Theorem: Suppose that for all $n > n_0$, we have $T(n) = a \cdot T(n/b) + c \cdot n^k$ for $a, b \geq 1$ and $k \geq 0$, then:
   - If $a < b^k$, then $T(n) \in \Theta(n^k)$.
   - If $a = b^k$, then $T(n) \in \Theta(n^k \log n)$.
   - If $a > b^k$, then $T(n) \in \Theta(n^{\log_b a})$.

Above, $\log$ can be taken as $\lceil \log \rceil$ or $\lfloor \log \rfloor$ ± $c$ for constant $c$. 

---

Page 3 of 3 October 31, 2009