CSCI 2300 HW #3

by Joshua M. Zone

Question 1 (6.4):

a) The Subproblems: define an array of subproblems S(i) for $0 \leqslant i \leqslant n$ Where S(i) is 1 if s[1..i] is a sequence of words. Otherwise it will be 0.

   Algorithm: initialize S(0) = 1 and update the values S(i) in ascending order. Recursion is below:

$$S(i) = \max\left(0 \leqslant j < i\right) \{S(j) \colon \mathrm{dict}(s[j+1...i])\,\mathrm{is\,true}\}$$

   Then the string s can be reconstructed as a sequence of valid words if and only if S(n) = 1.

   If S[1...i] is a sequence of valid words then there is a last word S[j...i] that is valid and where S(j) = 1 which will cause S(i) to be set to 1. Otherwise, for any valid word S[j...i], S(j) must be 0 and S(i) will also be set to 0. Runs in $O\left(n^2\right)$ since there are n subproblems that are solvable in $O(n)$ time using the previous subproblem.

b) When S(i) is updated, keep track of the previous item, S(j) which caused the update of S(i), since S[j+1..i] was a valid word. When the program ends, if S(n) = 1, then to recover the partition in words, trace the series of updates. This runs in constant time to each subproblem which is O(n) time to pass over the full array. So the runtime remains $O\left(n^2\right)$.

Question 2 (6.15):

Define the subproblems A(i, j) for $1 \leqslant i, j \leqslant n$ to represent the probability that A is te first to win n games, given that after $i+j$ games, A has won i.

Initialize A by setting $A(n, j) = 1$ for $j! = n$ and $A(i, n) = 0$ for all i. Other subproblems can be solved incrementally in decreasing order of $i + j$ using the recursion:

$$A(i, j) = \frac{1}{2}(A(i, j+1) + A(i+1, j))$$

Using the recursion we can compute A(i, j) by conditioning on the outcome of the (i + j + 1)th game therefore we know that the recursion is correct. These outcomes take place with the probability of $\frac{1}{2}$. If A wins, then the probability of A winning the game is A(i + 1, j) also if B wins, then A has a probability of A(i, j+1) of winning. To find the solutions of all the subproblems we would have to solve $O(n^2)$ subproblems that would each take O(1) time to solve which leaves a total runtime of $O(n^2)$. If we only want to know A(i,j) subproblem then stop once $O((n-(i+j))^2)$ subproblems.

Question 3 (6.17):

Define D(v) as a predicate which evaluates to true if it can make change for v using the available denominations $x_1, x_2, \ldots, x_n$. If it is possible to make change for v using given denominations, then it is possible to make change for $v - x_i$ using the same denomiations with one coin of x_i being chosen. Since we don't know which i will finally give us true value we will do a *logical or* over all i.

$$D(v) = V_{1 \leqslant i \leqslant n} \left\{ \frac{D(v - x_i) \text{ if } x_i \leqslant v}{\text{false otherwise}} \right.$$

```
precedure make-change(x_1,...x_n, V)
Input: Denominations of available coins x_1,...,x_n, Value V for which we are seeking denominations
Output: true if making change with available denomination is feasible
      false otherwise
Declare an array D of size V + 1
D[0] = true
for i = 1 to V
   D[i] = false
for v = 1 to V:
   for j = 1 to n:
      if x_j ⩽ v:
         D[v] ∨ D[v − x_j]
      else:
         D[v] = false
return D[V]
```

Complexity: $O(|n||V|)$

Question 4 (WordWrap):

```
Print-Neatly(M,n,l)
for i := (n...1)
do length := l_i
   j := i
   OPT[i] := ∞
   last[i]

   while length <= M
   do if j =
      then OPT[i] := 0
         last[i] := n
         exit while

      else cost := OPT[j] + (M - length)^2
         if cost < OPT[i]
            then OPT[i] := cost
               last[i] := l

         j := j + 1
         length := length + l_j + 1

i := 1
while i < n
do Print words i through next[i]
   Print new line
```

```
i := next[i] + 1
```

The outerloop will perform n iterations, and the inner loop will perform at most $\frac{M}{2}$ with an overall time complexity of O(nM). The OPT and $next$ arrays both have n elements, therefore they require *O(n)* space.