Training Problems #6 Solutions

15.1-4

Modify MEMOIZED-CUT-ROD to return not only the value but the actual solution, too.

Solution:

```
MEMOIZED-CUT-ROD$(p, n)$
  let $r[0..n]$ and $s[0..n]$ be new arrays
  for $i = 0$ to $n$
      $r[i] = -\infty$
  $(val, s) =$ MEMOIZED-CUT-ROD-AUX$(p, n, r, s)$
  print "The optimal value is " $val$ " and the cuts are at "
  $j = n$
  while $j > 0$
      print $s[j]$
      $j = j - s[j]$
```

```
MEMOIZED-CUT-ROD-AUX$(p, n, r, s)$
  if $r[n] \geq 0$
      return $r[n]$
  if $n == 0$
      $q = 0$
  else $q = -\infty$
      for $i = 1$ to $n$
          $(val, s) =$ MEMOIZED-CUT-ROD-AUX$(p, n - i, r, s)$
          if $q < p[i] + val$
              $q = p[i] + val$
              $s[n] = i$
  $r[n] = q$
  return $(q, s)$
```

PRINT-CUT-ROD-SOLUTION constructs the actual lengths where a cut should happen. Array entry $s[i]$ contains the value j indicating that an optimal cut for a rod of length i is j inches. The next cut is given by $s[i - j]$, and so on.

16.1-3

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

Solution:

- For the approach of selecting the activity of least duration from those that are compatible with previously selected activities:

i	1	2	3
s_i	0	2	3
f_i	3	4	6
duration	3	2	3

  This approach selects just $\{a_2\}$, but the optimal solution selects $\{a_1, a_3\}$.

- For the approach of always selecting the compatible activity that overlaps the fewest other remaining activities:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	0	1	1	1	2	3	4	5	5	5	6
f_i	2	3	3	3	4	5	6	7	7	7	8
# of overlapping activities	3	4	4	4	4	2	4	4	4	4	3

  This approach first selects a_6, and after that choice it can select only two other activities (one of a_1, a_2, a_3, a_4 and one of a_8, a_9, a_{10}, a_{11}). An optimal solution is $\{a_1, a_5, a_7, a_{11}\}$.

- For the approach of always selecting the compatible remaining activity with the earliest start time, just add one more activity with the interval $[0, 14)$ to the example in Section 16.1. It will be the first activity selected, and no other activities are compatible with it.

Provide the Fractional Knapsack algorithm. It must return both the resulting load and value in the knapsack.

Solution:

```
Fractional-Knapsack(v,w,W)
    load=0          //weight of items stolen
    i=1
    n=w.length //number of items available to steal
    value = 0       //value of items stolen
    while load < W  and i <= n
            if $w_i$ <= W - load
            {
                    value = value + $v_i$
                    load = load + $w_i$
            }
            else
            {
                    value = value + $v_i$ * (W-load/$w_i$)
                    load = W

            }
            i++
return load, value
```