Lab 08

I. El-Shaarawy & M. Touny

December 9, 2023

Contents

Exercises

8.1.3

Homework.

8.1.5

Homework.

8.1.6

Hints

- Explain why is the formulation $F(n) = F(n-1)+p_1$ is wrong. Derive a counter example.
- The optimal solution may be $F(n) = p_n$. Modify it so that it is in terms of $F(k)$ for some $k < n$.
- Generalize.

Solution

Recursive formulation.

$$\begin{aligned} F(0) &= 0 \\ F(n) &= \max_{1 \le j \le n} \{p_j + F(n-j)\} \end{aligned}$$

Algorithm.

```
# input: Length n, and values of pieces of length i, P[i]
# output: Maximum value of all possible cuts on a rod of length n
def dynamicRodCut(n, P[0..n])

    # a rod of length zero contributes nothing to revenue
    P[0] = 0

    # Initialize an array of size n
    F = [] * n

    # Set the base case
    F[0] = 0

    # Compute bottom-up F[i]
    for i in 1..n

        maxVal = 0

        # Compute the maximum among all js
```

```
        for j in 0..i
            # call memoized subinstances
            # update if found a greater value
            maxVal = max( maxVal, P[j] + F(n-j) )

        # memoize
        F[i] = maxVal

    # return max value of cuts, on given length n
    return F[n]
```

Complexity. Time is $1 + \cdots + n = n(n+1)/2$. Additional space is $n + 1$.

8.2.2

Homework.

8.2.3

Homework.

8.2.5

Hints

- Recall for the algorithm given in the book, at each step, either we take or leave the ith item.
- For our case what if we at each step, either leave all items, or take 1st item, or take 2nd item, .., or nth item. Modify the formulation.

Solution

Recursive Formulation.

$$\begin{aligned} F(W) &= 0 && \text{if } W < w_j,\ 1 \le j \le n \\ F(W) &= \max_{j: W \ge w_j} F(W - w_j) + v_j && \text{otherwise} \end{aligned}$$

Algorithm

```
# memory function
# input: i indicating selecting a multiset of size at most i, from all n items
         j capacity
# output: optimal value of feasible multiset of size at most i
def MFKnapsack(i, j, weight, value, F)

    # only if not memoized, compute and cache it
    if F[i,j] < 0
```

```
        # find the maximum value among all cases

        # case, no additional item is taken
        # this value sustains only if there is no capacity for any item
        # recall values are positive integers
        maxVal = MFKnapsack(i-1, j)

        # for each item out of total n items,
        # if there is a capacity for it,
        # compute total value, and update if greater.
        for i in 1..n
            maxVal = max( maxVal, value[i] + MFKnapsack(i-1, j-weight[i]) )

        # memoize
        F[i,j] = maxVal

    return F[i,j]

# input: weight of ith item, value of ith item, total capacity W
# output: max value of a multiset of size at most n,
#         from all n items, constrained by capacity W
def dynamicKnapsack(weight[1..n], value[1..n], W)

    # memoization table
    # all cells -1, indicating no value is computed
    F[0..n, 0..W] = -1
    # except row 0 and column 0, values are 0, by definition of base case
    for i in 0..n, F[i,0] = 0
    for i in 0..W, F[0,i] = 0

    # compute memoization table F, and read F[n, W]
    sol = MFKnapsack(n, W, weight, value, F)

    # problem solution is F[n, W]
    return sol
```

8.3.4

Homework.

8.3.8

Hints.

- Recall you have table R, where $R[i,j]$ contains the root of the tree of nodes $i,\ldots,j$.
- Recall how the optimal solution of knapsack was constructed.

Solution.

```
# global variables: table R of roots indices
                    keys A
# input: root node of a tree, and indices i and j of keys covered
# output: None. Tree is modified so the root points to its children
# initialize with i = 1 and j = n
# def optimalBST(root, i, j)

    # base case
    if root = NULL
        return

    # index of the root of subtree of keys A_i, .., A_j
    k = R[i,j]

    # left child
    root.left = A[ R[i, k-1] ]
    # right child
    root.right = A[ R[k+1, j] ]

    # Recursively, Call the child
    optimalBST(root.left, i, k-1)
    optimalBST(root.right, k+1, j)
```

P.S. Anything by Donald Knuth is worthwhile studying, however for our pragmatic purposes we omit the analysis bounding $\mathcal{O}(n^2)$.