Homework 3

Walker Bagley

February 22, 2024

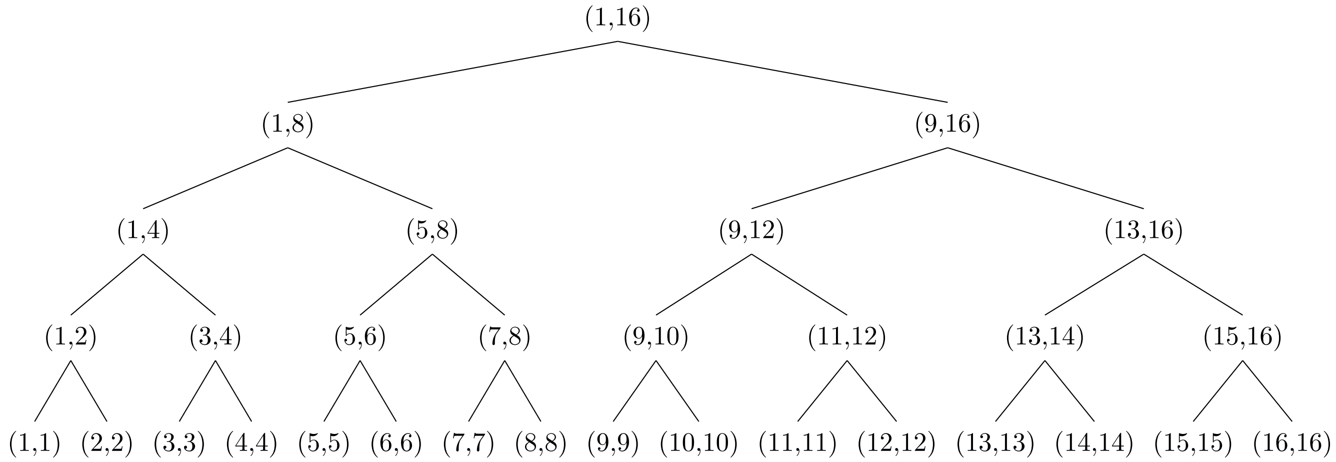
1 Exercise 14.1-3

```
MEMOIZED-CUT-ROD-COST (p, n, c)
    let r[0:n] be a new array
    for i = 0 to n
        r[i] = -infinity
    return MEMOIZED-CUT-ROD-COST-AUX(p, r, n, c)

MEMOIZED-CUT-ROD-COST-AUX (p, n, r, c)
    if r[n] >= 0
        return r[n]
    q = 0
    if n > 0
        for i = 1 to n
            gain = p[i]
            if i != n
                gain = gain - c
            q = max(q, MEMOIZED-CUT-ROD-COST-AUX(p, r, n - i, c) + gain)
    r[n] = q
    return q
```

This algorithm works the same as the original memoized rod cutting algorithm with two simple changes. First, the cost per cut must be passed into each function and second, we must check if we are making a cut before subtracting the cost from the solution. Both of these operate in $\Theta(1)$ time, so the complexity of the algorithm does not change.

2 Exercise 14.3-2



A good divide and conquer problem such as merge sort, has no repeated subproblems. This means that every subproblem is seen once and only once. Thus memoization does nothing since no subproblems need to be referenced after they are solved.

3 Exercise 14.4-3

```
REC-LCS-LENGTH (X, Y, m, n)
    let c[0:m, 0:n] be a new table
    for i = 1 to m
        c[i, 0] = 0
    for j = 0 to n
        c[0, j] = 0
    return REC-LCS-LENGTH-AUX(X, Y, m, n, c)

REC-LCS-LENGTH-AUX (X, Y, i, j, c)
    if i = 0 or j = 0
        return 0
    if X[i] = Y[i]
        c[i, j] = REC-LCS-LENGTH-AUX(X, Y, i - 1, j - 1, c) + 1
    else
        c[i, j] = max(REC-LCS-LENGTH-AUX(X, Y, i - 1, j, c),
                      REC-LCS-LENGTH-AUX(X, Y, i, j - 1, c))
    return c[i, j]
```

This algorithm works by checking if we have reached a base case (the start of one of the strings) and returns 0 if so. Otherwise if the two current symbols are equal, then it sets the current table entry equal to 1 + the recursive call to the two previous symbols in each string. If this is not the case, then it sets the current entry equal to the maximum of the two subproblems (1 previous symbol for either list). As this solution is memoized, any subproblem is computed only once, with any recurrences resolved in $\Theta(1)$ from the lookup. This means there are only mn subproblems and this algorithm runs in $O(mn)$ time.

4 Exercise 14.4-4

In a bottom up or top down solution to the LCR problem, one can reduce the c table to a maximum of $2\min(m, n)$ entries by iterating over the smaller of the two inputs first and nesting the larger iteration inside. By only retaining the last 2 rows/columns, we can reset the first row/column of the c table to the second one after iterating over each row/column. That is, $c[0, i] = c[1, i]$ after each iteration in the inside loop. Thus, the order of the subproblems is maintained and we have everything we need to reference contained in the smaller c table without using any extra space.