CS217 Data Structures and Algorithm Analysis
Fall term 2025

Exercise Sheet 11
Handout: November 18th | **Deadline:** November 25th

Question 11.1 (0.25 marks)

Consider a modification of the rod-cutting problem where in addition to a price p_i for each rod, each cut incurs a fixed cost c.

The revenue for a solution is now the sum of the prices of the individual pieces minus the costs of making the cuts.

Give a **Dynamic Programming Bottom-Up** algorithm to solve this problem.

Answer:

```
BOTTOM-UP-CUT-ROD-COST(p, n, c)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 to n
        q = -∞
        for i = 1 to j
            if i == j
                q = max(q, p[i])  // No cut cost if no cut is made
            else
                q = max(q, p[i] + r[j - i] - c)  // Subtract cut cost
        r[j] = q
    return r[n]
```

Question 11.2 (0.25 marks)

Modify **MEMOIZED-CUT-ROD** so that it also returns the actual solution rather than just its value.

Answer:

```
MODIFIED-MEMOIZED-CUT-ROD-AUX(p, n, r, s)
    if r[n] >= 0
        return r[n]
    if n == 0
        q = 0
    else
        q = -∞
        for i = 1 to n
            temp = p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r, s)
            if temp > q
                q = temp
                s[n] = i  // Store the best cut position
    r[n] = q
    return q

MEMOIZED-CUT-ROD(p, n)
    let r[0..n] be a new array
    let s[0..n] be a new array  // Array to store cut positions
    for i = 0 to n
        r[i] = -∞
    MEMOIZED-CUT-ROD-AUX(p, n, r, s)
    return s

// To reconstruct the solution
PRINT-CUT-ROD-SOLUTION(s, n)
    while n > 0
        print s[n]
        n = n - s[n]
```

Question 11.3 (0.25 marks)

Provide the pseudo-code of an $O(n)$ time Dynamic Programming algorithm for calculating the n_{th} Fibonacci number. Draw the subproblem graph.

How many vertices and edges does the subproblem graph contain?

Answer:

```
FIBONACCI(n)
    let f[0..n] be a new array
    f[0] = 0
    f[1] = 1
    for i = 2 to n
        f[i] = f[i - 1] + f[i - 2]
    return f[n]
```

There are $n + 1$ vertices in the subproblem graph (one for each Fibonacci number from $F(0)$ to $F(n)$).
There are $2n - 2$ edges in the subproblem graph, as each Fibonacci number $F(i)$ (for $i \geq 2$) has two incoming edges from $F(i - 1)$ and $F(i - 2)$.

Question 11.4 (1 mark)

After retiring from a large company, Mr Mortimer wonders how much money he could have made if he had invested his life savings in shares of his company.

He looks back at the share prices over the n days of his employment and asks himself how much his investment could have returned if he had somehow known the best time to buy and sell his shares.

Mortimer doesn't like trading shares, so he would only ever buy once and sell once and assume that his life savings is a fixed amount.

Let $a_1, ..., a_n$ describe the difference of share prices over time: a_i is the amount of money Mr Mortimer would have gained if he had held on to shares on day i.

Note that a_i can be negative, in which case day i would have been a loss.

Assume $a_i \neq 0$. Let $f(i, j) = \sum_{k=i}^{j} a_k$ be the money earned if Mortimer had bought shares at the start of day i of his employment and sold them at the end of day j.

Mortimer wants to find the maximum return $\max\{f(i, j) \mid 1 \leq i \leq j \leq n\}$.

Example: for $a_1, ..., a_n = +5, -3, -4, +8, -1, +12, -6, +4, +4, -14, +2, +8$, Mortimer's maximum return would have been $f(4, 9) = 8 + (-1) + 12 + (-6) + 4 + 4 = 21$ pounds.

(a) Design an algorithm in pseudocode that computes the maximum return using dynamic programming in time $O(n)$, following the hints below. Explain your solution.

(b) Show that your algorithm runs in time $O(n)$.

Hints: Use a bottom-up approach, tabulating for each day k:

- the maximum return A_k for an investment up to day k (buying and selling up to day k, formally $\max\{f(i,j) \mid 1 \leq i \leq j \leq k\}$) and
- the maximum return B_k up to day k for an ongoing investment: still holding on to shares on day k (formally $\max\{f(i,k) \mid 1 \leq i \leq k\}$).

Work out how A_1 and B_1 can be initialised, and work out Bellman equations showing how, for $k \geq 2$, A_k and B_k can be computed based on the input and values of A and B that you have tabulated earlier.

Pay attention to the order in which to tabulate values. Don't forget to state how to compute the final output from the tabulated values.

Answer:

(a)

```
MAXIMUM-RETURN(a, n)
    let A[1..n] be a new array
    let B[1..n] be a new array
    A[1] = a[1]
    B[1] = a[1]
    for k = 2 to n
        B[k] = max(B[k - 1] + a[k], a[k])  // Ongoing investment
        A[k] = max(A[k - 1], B[k])         // Maximum return up to day k
    return A[n]
```

Explanation:

To find out the maximum return, we only focus on the whether Mortimer sells his share on day `k`. If on day k the investment is still ongoing, the maximum return is the maximum ongoing return up to day `k`, i.e., `B[k]`. If the investment is sold on day `k`, the maximum return is the maximum return up to day `k-1`, i.e., `A[k-1]`. Hence we first maintain an array `B` to keep track of the maximum ongoing return up to day `k`, and then use it to compute the maximum return `A[k]` up to day `k`. For the ongoing return `B[k]`, we have two choices: either we continue the ongoing investment from

day `k-1` to day `k`, or we start a new investment on day `k`. We take the maximum of these two choices to get `B[k]`.

(b)

The algorithm runs in time `O(n)` because it consists of a single loop that iterates from `2` to `n`, performing a constant amount of work in each iteration. Thus, the overall time complexity is linear with respect to the number of days `n`.