Solutions to Final Practice Problems 1-15

1. Given a sequence of n real numbers $A_1 \dots A_n$, determine *in linear time* a contiguous subsequence $A_i \dots A_j \quad (i \leq j)$ for which the sum of elements in the subsequence is maximized.

   Solution:

   • **Subproblem(k):** "find $m \leq k$ such that suffix $A_m A_{m+1} \dots A_k$ of the subsequence $A_1 A_2 \dots A_k$ has a maximal possible sum."
   Note that such a suffix might consist of a single element A_k.

   • **Recursion:** let OptSuffix(k) denote $m \leq k$ such that suffix $A_m A_{m+1} \dots A_k$ of the subsequence $A_1 A_2 \dots A_k$ has the largest possible sum and let this sum be denoted by OptSum(k). Then OptSuffix(1) = 1 and OptSum(1) = A_1 and for $k > 1$

$$\text{OptSuffix}(k) = \begin{cases} \text{OptSuffix}(k-1) & \text{if } \text{OptSum}(k-1) > 0; \\ k & \text{otherwise} \end{cases}$$
$$\text{OptSum}(k) = \begin{cases} \text{OptSum}(k-1) + A_k & \text{if } \text{OptSum}(k-1) > 0; \\ A_k & \text{otherwise} \end{cases}$$

   Thus, if the sum OptSum($k-1$) of the optimal suffix $A_m A_{m+1} \dots A_{k-1}$ (where $m = \text{OptSuffix}(k-1)$) for the subsequence $A_1 A_2 \dots A_{k-1}$ is positive, then we extend such a suffix with A_k to obtain optimal suffix $A_m A_{m+1} \dots A_{k-1} A_k$ for the subsequence $A_1 A_2 \dots A_k$ and consequently OptSuffix(k) is still equal to m and OptSum(k) = OptSum($k-1$) + A_k; otherwise, if OptSum($k-1$) is negative we discard such optimal suffix and start a new one consisting of a single number A_k; thus, in this case OptSuffix(k) = k and OptSum(k) = A_k.

   • **Solution of the original problem:** After we obtain OptSum(k) and OptSuffix(k) for all $1 \leq k \leq n$ we find $1 \leq q \leq n$ such that the corresponding OptSum(q) is the largest among all OptSum(k) for $1 \leq k \leq n$; let the corresponding OptSuffix(q) be equal to p; then the solution for the initial problem is the subsequence $A_p \dots A_q$.

2. You are traveling by a canoe down a river and there are n trading posts along the way. Before starting your journey, you are given for each $1 \leq i < j \leq n$ the fee $F(i, j)$ for renting a canoe from post i to post j. These fees are arbitrary. For example it is possible that $F(1, 3) = 10$ and $F(1, 4) = 5$. You begin at trading post 1 and must end at trading post n (using rented canoes). Your goal is to design an

efficient algorithms which produces the sequence of trading posts where you change your canoe which minimizes the total rental cost.

Solution:

• **Subproblem(k):** "Find the sequence of trading posts which provides the cheapest way of getting from post 1 to post k".

• **Recursion:** Let OptCost(k) denote the minimal possible cost of getting from post 1 to post k, and let OptSeq(k) denote post m, ($m \leq k$) which is the last intermediate post in the corresponding optimal sequence $i_1, i_2, \ldots, i_{p-1}, i_p$ i.e., such that $i_1 = 1$, $i_{p-1} = m$ and $i_p = k$. Then OptCost(1) = 0 and for $k > 1$

$$\text{OptCost}(k) = \min_{1 \leq m \leq k-1} \{\text{OptCost}(m) + F(m,k)\}$$

and let OptSeq(k) be equal to m for which such a minimum is achieved (if there are several such m, pick one arbitrarily, say the smallest one), usually denoted by

$$\text{OptSeq}(k) = \arg\left(\min_{1 \leq m \leq k-1} \{\text{OptCost}(m) + F(m,k)\}\right)$$

• **Solution of the original problem:** After all of these subproblems are solved for all $1 \leq k \leq n$, we can backtrack from n to 1 using OptSeq(n) = p, OptSeq(p) = $q, \ldots$ to obtain a solution of the problem with an overall minimal cost equal to OptCost(n).

3. You are given a set of n types of rectangular boxes, where the i^{th} box has height h_i, width w_i and depth d_i. You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

   Solution:

   Let us take 6 instances of each box, and rotate them in all possible ways to obtain the 6 possible bases. Note that each box has (in general) three different rectangular faces and each rectangular face can be rotated in two possible ways, depending which of the two different edges of its base is "horizontal". For simplicity, after that, we no longer allow boxes to be rotated. Note that now we have in total $6n$ many boxes.

   We now note that if a box B_2 (with a fixed orientation) can be placed on top of a box B_1 then, since both sides of the base of B_2 must be smaller than the corresponding sides of the base of B_1, the surface area of the base of B_2 must be strictly smaller than the surface area of the base of B_1.

• **Producing the right ordering for recursion:** We order all boxes in a decreasing order of the surface area of their bases (putting boxes with the same surface area of their bases in an arbitrary order); then in every possible legitimate stack of boxes if a box $B_{i_{k+1}}$ is placed on top of the box B_{i_k}, then $i_{k+1} > i_k$. From now on we will assume that all the boxes are ordered in such a way.

• **Subproblem(k):** "Find a stack built from boxes $B_1, B_2, \ldots B_k$, $1 \leq k \leq 6n$, *which must end with box B_k* and which is of the largest possible total height".

• **Recursion:** Let us denote by MaxHeight(k) the height of the stack which solves Subproblem(k). Let also length(m), width(m) and height(m) denote the length, width and height of box B_m, respectively. Then

$$\text{MaxHeight}(k) =$$
$$\max_{1\leq m\leq k-1} \{\text{MaxHeight}(m) + \text{height}(k) \ : \ \text{length}(k) < \text{length}(m) \ \&\& \ \text{width}(k) < \text{width}(m)\}$$

We also define PreviousBox(k) to be the argument m for which the above maximum is achieved.

• **Solution of the original problem:** After all of these subproblems are solved for all $1 \leq k \leq n$, we pick the largest of MaxHeight(k) to obtain the height of the tallest stack of boxes and can reconstruct such a stack using pointers PreviousBox(k).

4. Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates $a_1 \ldots a_n$ and n cities on the northern bank with x-coordinates $b_1 \ldots b_n$. You want to connect as many north-south pairs of cities as possible, with bridges such that no two bridges cross. When connecting cities, you are only allowed to connect the the i^{th} city on the northern bank to the i^{th} city on the southern bank.

   Solution:

   We sort in an increasing order the cities on the south bank according to their x-coordinates $a_1 \ldots a_n$; thus we can now assume that indexing of the cities is such that $a_{i+1} > a_i$. Let the corresponding cities on the north bank will have x-coordinates $b_1 \ldots b_n$. Let $i < j$ be arbitrary; now observe that the bridge between a_i and b_i does not cross the bridge between a_j and b_j just in case $b_i < b_j$. Thus, to solve the problem, we just have to find the longest monotonically increasing subsequence of the sequence $b_1 b_2 \ldots b_n$, which is a problem solved in class, see the slides on Dynamic Programming.

5. Some people think that the bigger an elephant is, the smarter it is. To disprove this you want to analyze a collection of elephants and place as large a subset of elephants as possible into a sequence whose weights are increasing but their IQs are

decreasing. Design an algorithm which given the weights and IQs of n elephants, will find a longest sequence of elephants such that their weights are increasing but IQs are decreasing.

Solution:

We again reduce this problem to the problem of finding the longest increasing subsequence, Thus, we sort all the elephants in a decreasing order according to their IQ; we can now assume that indexing of the elephants is chosen so that the IQ(i) of elephant i is lover than the IQ($i-1$) of elephant $i-1$, for all i such that $2 \leq i \leq n$. Let the weights of elephant i be denoted by W(i). Clearly now the problem reduces to finding the longest increasing subsequence of the sequence W(1),. . . ,W(n).

6. You have an amount of money M and you are in a candy store. There are n kinds of candies and for each candy you know how much pleasure you get by eating it, which is a number between 1 and 100, as well as the price of each candy. Your task is to chose which candies you are going to buy to maximise the total pleasure you will get by gobbling them all.

   Solution: All it takes is to notice that this is the classical knapsack problem, with duplicates allowed: the capacity of the knapsack is the amount of money M you got, the size of each item (candy) is its price and the value of each item is its pleasure score.

7. There are N lily pads in a row. A frog starts on the leftmost lily pad and wishes to get to the rightmost one. The frog can only jump to the right. There are two kinds of jump the frog can make:

   - The frog can jump 3 lily pads to the right (skipping over two of them)
   - The frog can jump 5 lily pads to the right (skipping over four of them)

   Each lily pad has some number of flies on it. Design an algorithm that maximizes the total number of flies on lily pads the frog lands on getting to the rightmost lily pad.

   Solution:

   Let us denote the number of flies on lily pad k by $f(k)$.

   • **Subproblem(k):** "Find the largest total number of flies $N(k)$ on all lily pads the frog lands on while getting to the lily pad k."

   If a lily pad k cannot be accessed from the first lily pad by jumps satisfying the prescribed constraints, let us define $N(k) = -\infty$.

   • **Recursion:** Clearly, $N(1) = f(1)$; the immediately accessible lily pads (with a single jump) are lily pad 4 and lily pad 6. Thus $N(2) = N(3) = N(5) = -\infty$ and

$N(4) = f(1) + f(4)$ and $N(6) = f(1) + f(6)$. For $k > 6$ the recursion formula is

$$N(k) = max\{N(k-5), N(k-3)\} + f(k)$$

Note that if lily pad $k-5$ and lily pad $k-3$ are both inaccessible from lily pad 1, this recursion will correctly set the value of $N(k)$ to $-\infty$.

• **Solution of the original problem:** It is just the last obtained value $N(n)$.

8. You are given a rooted tree. Each edge of the tree has a cost for removing it. Devise an algorithm to compute the minimum total cost of removing edges to disconnect the root from all the leaves of the tree.

   Solution 1: Make all the edges directed pointing towards the leaves, and then treat this tree as a flow network with the root of the tree as the source and with the leaves as sinks and with the capacity of each edge equal to the cost of removing that edge. Add a super-sink and connect it with all leaves with edges with infinite capacity. Now run your favorite max flow algorithm until it converged. Take the minimal cut defined as all the vertices in the last residual flow network which are accessible from the source. The edges to be removed are now the edges crossing the minimal cut.

   Note that the fastest max flow algorithm to date runs in time $O(|V|^3)$. We now present a faster dynamic programming solution to the above problem.

   Solution 2:

   Let v be any vertex of the given tree T and let T_v denote the subtree of T with root at v. Let also the cost of removing an edge (u, v) be denoted by $\text{cost}(u, v)$.

   • **Subproblem(v):** "Find the minimum total cost of removing some of the edges of T_v to disconnect the root v from all the leaves of T_v."

   • **Ordering of Subproblems:** Subproblem(v) precedes Subproblem(u) if v belongs to subtree T_u.

   • **Recursion:** Let MinCost(v) be the minimal cost solution for Subproblem(v) and the children of v be $w_1, w_2, \ldots, w_k$; then

$$\text{MinCost}(v) = \sum_{i=1}^{k} \min\{\text{cost}(v, w_i), \text{MinCost}(w_i)\}$$

   Clearly, the above algorithm runs in time $O(|V|)$, because during the recursion steps each entry MinCost(w_i) is computed only once and looked at only once (when computing MinCost(v) for its parent v).

9. You have to cut a wood stick into several pieces at the marks on the stick. The most affordable company, Analog Cutting Machinery (ACM), charges money according to the length of the stick being cut. Their cutting saw allows them to make only one cut at a time. It is easy to see that different cutting orders can lead to different prices.

For example, consider a stick of length 10 m that has to be cut at 2, 4, and 7 m from one end. There are several choices. One can cut first at 2, then at 4, then at 7. This leads to a price of 10 + 8 + 6 = 24 because the first stick was of 10 m, the resulting stick of 8 m, and the last one of 6 m. Another choice could cut at 4, then at 2, then at 7. This would lead to a price of 10 + 4 + 6 = 20, which is better for us. Your boss demands that you design an algorithm to find the minimum possible cutting cost for any given stick.

Solution:

Let us index left edge of the stick by 0, then consecutive marks where the stick is to be cut by 1 to n and finally the right edge by $n + 1$. For every i such that $0 \leq i \leq n$ and for every j such that $i < j \leq n + 1$ we consider the following subproblems:

• **Subproblem(i,j):** "Find the minimum total cost of cutting into pieces a stick with the left end at point i and the right end at point j."

• **Ordering of Subproblems:** Even though the subproblems are indexed with two variables it is enough to order subproblems by their corresponding values of $j - i$. Thus, we first solve subproblems with smaller values of $j - i$, solving the problems with the same value of $j - i$ in an arbitrary order.

Let also $l(i, j)$ denote the length of the part of the stick between marks i and j; we can then take that the cost of making a single cut on the piece of the stick with ends at marks i and j is equal to $l(i, j)$.

• **Recursion:** Let MinCost(i, j) be the minimal cost solution for Subproblem(i, j); then

$$\text{MinCost}(i,j) = \begin{cases} \min_{i<k<j}\{\text{MinCost}(i,k) + \text{MinCost}(k,j) + l(i,j)\} & \text{if} \quad j \geq i+2 \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the above algorithm runs in time $O(|V|)$, because during the recursion steps each entry MinCost(w_i) is computed only once and looked at only once (when computing MinCost(v) for its parent v).

10. You are given a boolean expression consisting of a string of the symbols *true* and *false* and with exactly one operation *and, or, xor* between any two consecutive truth values. Count the number of ways to place brackets in the expression such that it will evaluate to true. For example, there is only 1 way to place parentheses in the expression *true* and *false* xor *true* such that it evaluates to true.

    Solution:

    Let us enumerate all *true* and *false* symbols by indices $1 \ldots n$.

    • **Subproblem(i,j):** "For every substring $S(i, j)$ of the symbols $\sigma_k \in \{true, false\}$, $(i \leq k \leq j)$, between the i^{th} and j^{th} symbol inclusively compute the number $T(i, j)$ of ways to place brackets in the corresponding expression such that it will evaluate to true **and** the number $F(i,j)$ of ways to place brackets in the corresponding expression such that it will evaluate to false.”

• **Ordering of subproblems:** Subproblems are solved in the order of increasing values of $j-i$, breaking evens arbitrarily.

• **Recursion:** Let us denote the k^{th} operand by $\otimes_k$; thus $\otimes_k \in \{and, or, xor\}$. If $j-i \leq 2$ then the truth value of the substring $S(i,j)$ is uniquely determined; for $j-i \geq 3$ we have

$$T(i,j) = \sum_{i<k<j} \begin{cases} T(i,k)T(k+1,j) & \text{if } \otimes_k = and \\ T(i,k)T(k+1,j) + T(i,k)F(k+1,j) + F(i,k)T(k+1,j) & \text{if } \otimes_k = or \\ T(i,k)F(k+1,j) + F(i,k)T(k+1,j) & \text{if } \otimes_k = xor \end{cases}$$

$$F(i,j) = \sum_{i<k<j} \begin{cases} T(i,k)F(k+1,j) + F(i,k)T(k+1,j) + F(i,k)F(k+1,j) & \text{if } \otimes_k = and \\ F(i,k)F(k+1,j) & \text{if } \otimes_k = or \\ T(i,k)T(k+1,j) + F(i,k)F(k+1,j) & \text{if } \otimes_k = xor \end{cases}$$

• **Recursion:** Let us denote the k^{th} operand by $\otimes_k$; thus $\otimes_k \in \{and, or, xor\}$. If $j-i \leq 2$ then the truth value of the substring $S(i,j)$ is uniquely determined; for $j-i \geq 3$ we have

11. You are given a sequence of numbers with operations $+, -, \times$ in between, for example

$$1 + 2 - 3 \times 6 - 1 - 2 \times 3 - 5 \times 7 + 2 - 8 \times 9$$

Your task is to place brackets in a way that the resulting expression has the largest possible value.

Solution:

Similar to the previous one; this time for every substring consisting of the numbers between the i^{th} and the j^{th} number (inclusively) we compute the minimal value $\mathrm{MIN}(i,j)$ of the corresponding subexpression as well as the maximal such value $\mathrm{MAX}(i,j)$.

• **Recursion:** Let us denote the k^{th} operand by $\otimes_k$; thus $\otimes_k \in \{+, -, \times\}$. If $j-i \leq 2$ then the truth value of the substring $S(i,j)$ is uniquely determined; for $j-i \geq 3$ we have

$$\mathrm{MIN}(i,j) = \begin{cases} \mathrm{MIN}(i,k) + \mathrm{MIN}(k+1,j) & \text{if } \otimes_k = + \\ \mathrm{MIN}(i,k) - \mathrm{MAX}(k+1,j) & \text{if } \otimes_k = - \\ \min \begin{cases} \mathrm{MIN}(i,k) \times \mathrm{MIN}(k+1,j), \\ \mathrm{MIN}(i,k) \times \mathrm{MAX}(k+1,j) \\ \mathrm{MAX}(i,k) \times \mathrm{MIN}(k+1,j) \\ \mathrm{MAX}(i,k) \times \mathrm{MAX}(k+1,j) \end{cases} & \text{if } \otimes_k = \times \end{cases}$$

$$\mathrm{MAX}(i,j) = \begin{cases} \mathrm{MAX}(i,k) + \mathrm{MAX}(k+1,j) & \text{if } \otimes_k = + \\ \mathrm{MAX}(i,k) - \mathrm{MIN}(k+1,j) & \text{if } \otimes_k = - \\ \max \begin{cases} \mathrm{MIN}(i,k) \times \mathrm{MIN}(k+1,j), \\ \mathrm{MIN}(i,k) \times \mathrm{MAX}(k+1,j) \\ \mathrm{MAX}(i,k) \times \mathrm{MIN}(k+1,j) \\ \mathrm{MAX}(i,k) \times \mathrm{MAX}(k+1,j) \end{cases} & \text{if } \otimes_k = \times \end{cases}$$

Note that in case $\otimes_k = \times$ the computation of the values of $\mathrm{MIN}(i,j)$ and $\mathrm{MAX}(i,j)$ depend on the signs of the values of $\mathrm{MIN}(i,k)$, $\mathrm{MAX}(i,k)$, $\mathrm{MIN}(k+1,j)$, $\mathrm{MAX}(k+1,j)$; for example if $\mathrm{MIN}(i,k)$ and $\mathrm{MIN}(k+1,j)$ are both negative numbers whose absolute values are larger than $\mathrm{MAX}(i,k)$ and $\mathrm{MAX}(k+1,j)$ respectively, then $\mathrm{MAX}(i,j)$ is equal to the product of these two negative minima (rather than the products of the corresponding two maxima).

12. A company is organizing a party for its employees. The organizers of the party want it to be a fun party, and so have assigned a fun rating to every employee. The employees are organized into a strict hierarchy, i.e. a tree rooted at the president. There is one restriction, though, on the guest list to the party: an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all). Give an algorithm that makes a guest list for the party that maximizes the sum of the fun ratings of the guests.

    Solution: Let the fun rating of an employee v be denoted by $\rho(v)$ We again do recursion on rooted subtrees, solving the following two subproblems by a simultaneous recursion:

    • **Subproblem(v):** "Find the maximal rating score $\mathrm{INCLUDE}(v)$ of guests belonging to the subtree T_v rooted at a node v which must include v and the maximal rating score $\mathrm{EXCLUDE}(v)$ of guests belonging to the subtree T_v rooted at a node v which must exclude v ."

    • **Recursion:**

$$\mathrm{INCLUDE}(v) = \rho(v) + \sum_{w} \{\mathrm{EXCLUDE}(w) \ : \ w \text{ is a child of } \ v\}$$

$$\text{EXCLUDE}(v) = \sum_w \{\max\left(\text{INCLUDE}(w), \text{EXCLUDE}(w)\right) \ : \ w \text{ is a child of } \ v\}$$

13. Consider a sequence of n coins of values $v_1 \dots v_n$, where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the sequence and removes it from the sequence permanently, and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first.

    Solution:

    Let v_i denote the value of the coin c_i. For every i and j such that $i < j$ and such that $j - (i - 1)$ is even we consider the following subproblems:

    • **Subproblem(i,j):** "For the subsequence of coins $c_i, c_{i+1}, \dots c_j$ determine the maximum possible amount of money $\text{OPT}(i, j)$ we can definitely win if we move first."

    • **Recursion:**

$$\text{OPT}(i, j) = \max \begin{cases} v_i + \min\{\text{OPT}(i + 1, j - 1), \text{OPT}(i + 2, j)\} \\ v_j + \min\{\text{OPT}(i + 1, j - 1), \text{OPT}(i, j - 2)\} \end{cases}$$

    In the first case we remove coin c_i and our opponent can then remove either coin c_j or coin c_{i+1} and we have to assume that he will leave us with the worse of the two possible outcomes and thus we take the min of the two options. In the second case we remove the last coin c_j and our opponent can then remove either coin c_i or coin c_{j-1}.

14. You have n_1 items of size s_1, n_2 items of size s_2, and n_3 items of size s_3. You would like to pack all of these items into bins, each of capacity C, using as few bins as possible.

    Solution:

    For every $i \leq n_1$, $j \leq n_2$ and $k \leq n_3$ we solve the following subproblems:

    • **Subproblem(i,j,k):** "Pack i items of size s_1, j items of size s_2, and k items of size s_3 into the smallest possible number of bins, each of capacity C."

    Denote such minimal number by $\text{OPT}(i, j, k)$.

    • **Recursion:**

$$\text{OPT}(i, j, k) = 1 + \min\{\text{OPT}(i - \alpha, j - \beta, k - \gamma) \ : \ \alpha s_1 + \beta s_2 + \gamma s_3 \leq C \ \& \\ \& \ (\alpha + 1)s_1 + \beta s_2 + \gamma s_3 > C \ \& \ \alpha s_1 + (\beta + 1)s_2 + \gamma s_3 > C \ \& \ \alpha s_1 + \beta s_2 + (\gamma + 1)s_3 > C\}$$

    Thus, we fill in all possible ways a single box up to its capacity C and then look for the optimal solutions for the remaining items.

15. Find the number of partitions of a positive integer n, i.e. the number of sets of positive integers $\{p_1, p_2, \ldots, p_k\}$ such that such that $p_1 + p_2 + \ldots + p_k = n$. *(Hint: try a relaxation with respect to the size of the largest number in such a sum.)*

    Solution:

    For every i and j such that $1 \leq j \leq i \leq n$ we solve the following subproblems:

    • **Subproblem(i,j):** "Find the number $N(i, j)$ of partitions $\{p_1, p_2, \ldots, p_k\}$ of i such that $p_m \leq j$ for all m such that $1 \leq m \leq k$."

    • **Recursion:** $N(i, j) = N(i, j - 1) + N(i - j, j)$.

    Thus, the number of partitions of integer i where each part does not exceed j is equal to the number of partitions of i where each part does not exceed $j - 1$ plus the number of partitions which have at least element equal to j. But, taking out one element of size j, the number of partitions which have at least element equal to j is equal to the number of partitions of integer $i - j$ (to account for the number we took out) where each element of such partitions does not exceed j (because there might be more than one element of size j).