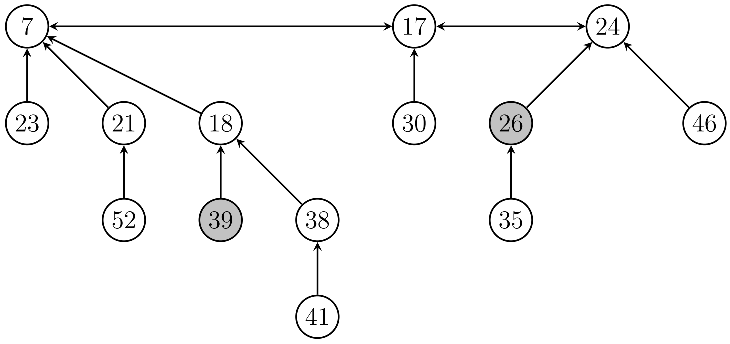
Exercise 7

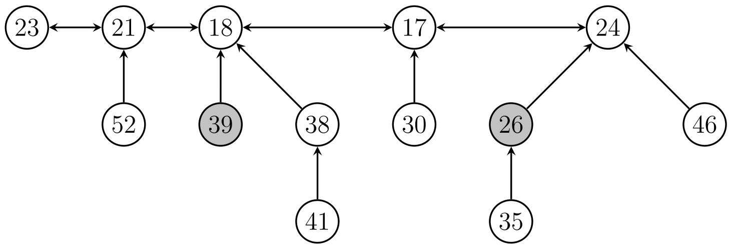
Task 1
Given the following Fibonacci heap:



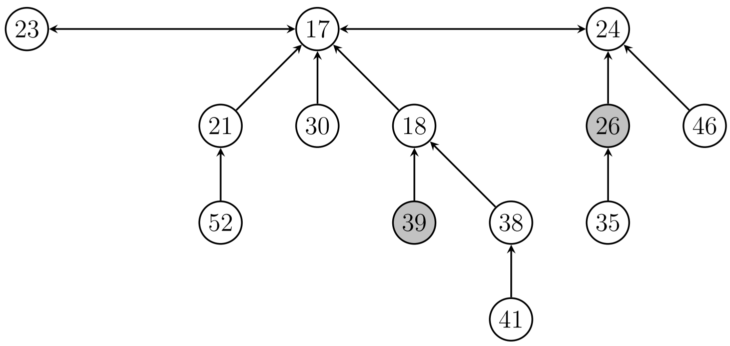
Perform the following operations in that order:
delete-min, **decrease-key**(“52”, 9), **decrease-key**(“46”, 3), **insert**(42), **delete-min**, **decrease-key**(“35”, 7)

Solution

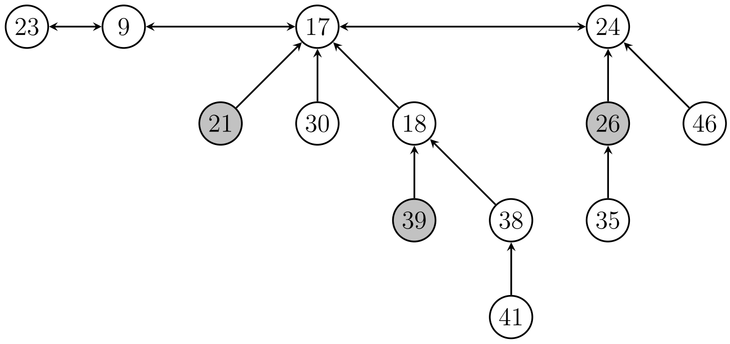
1. **delete-min**: The node with key 7 gets deleted.



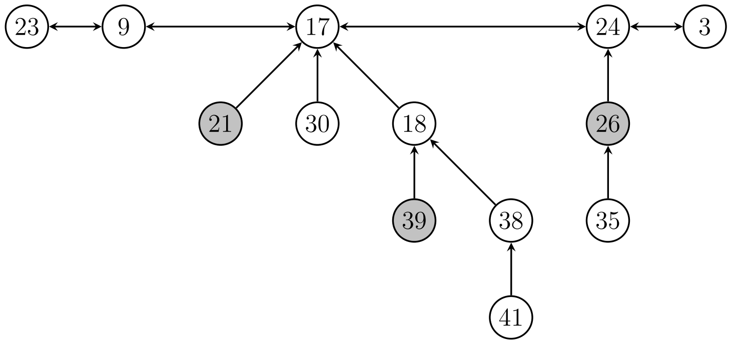
And we tidy the forest a bit.



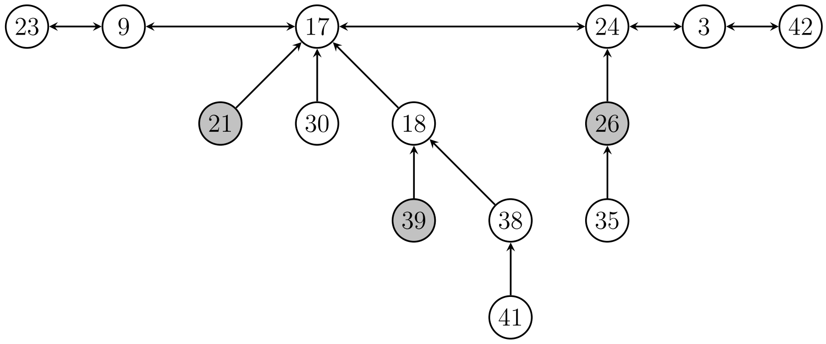
2. **decrease-key**(“52”, 9): 9 moves up, 21 gets marked.



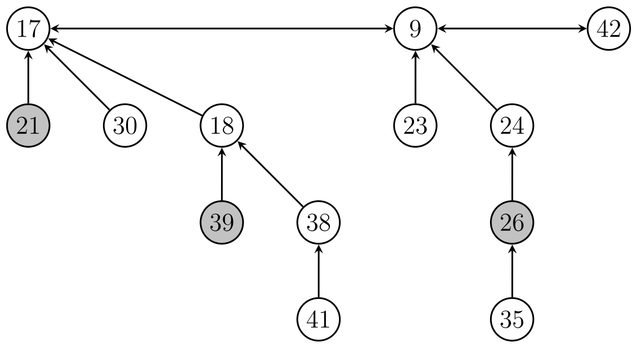
3. **decrease-key**(“46”, 3): 3 moves up, 24 cannot be marked.



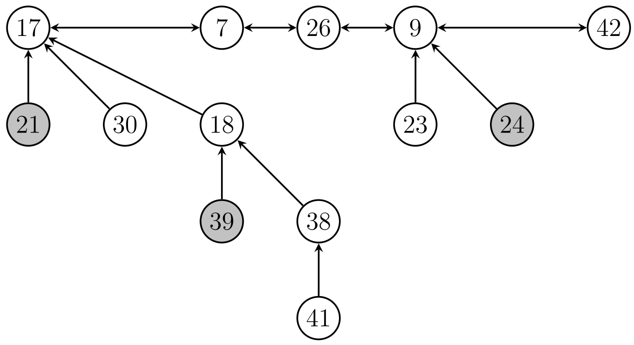
4. **insert**(42): Inserting 42 as a new tree.



5. **delete-min**: Node with key 3 gets deleted and we tidy the forest.



6. **decrease-key**("35", 7): 7 moves up and 26 as well, since it is marked (but loses its mark). 24 gets marked.



Task 2

Find the optimal order to compute the following product (only the dimensions of the matrices are given):

$$(2 \times 4) \cdot (4 \times 6) \cdot (6 \times 1) \cdot (1 \times 10) \cdot (10 \times 10)$$

Solution

We compute the number of multiplications of the product $A_1A_2A_3A_4A_5$ by dynamic programming.

Matrix products of length 2: $2 \cdot 4 \cdot 6 = 48 \mid 4 \cdot 6 = 24 \mid 6 \cdot 10 = 60 \mid 10 \cdot 10 = 100$

Matrix products of length 3 ($2 + 1$ or $1 + 2$):
$\min(48 + 12, 24 + 8) = 32$, hence $A_1(A_2A_3)$ is optimal | $\min(24 + 40, 60 + 240) = 64$, hence $(A_2A_3)A_4$ is optimal | $\min(60 + 600, 100 + 60) = 160$, hence $A_3(A_4A_5)$ is optimal

Matrix products of length 4 ($3 + 1$ or $2 + 2$ or $1 + 3$):
$\min(32 + 20, 48 + 60 + 120, 64 + 80) = 52$, hence $(A_1(A_2A_3))A_4$ is optimal |
$\min(64 + 400, 24 + 100 + 40, 160 + 240) = 164$, hence $(A_2A_3)(A_4A_5)$ is optimal

Matrix product of length 5 ($4 + 1$ or $3 + 2$ or $2 + 3$ or $1 + 4$):
$\min(52 + 200, 32 + 100 + 20, 48 + 160 + 120, 164 + 80) = 152$

Hence, to compute the product $A_1A_2A_3A_4A_5$, it is the best to compute it via the bracketing $(A_1(A_2A_3))(A_4A_5)$, which takes 152 multiplications.

We can also encode these values in two tables. The table cost consists of the optimal number of multiplications for each subproblem, where entry (i, j) is representing the cost of the product $A_i \cdots A_j$ and best$[i, j]$ is the optimal cut-point for the bracketing:

$i\backslash j$	1	2	3	4	5
1	0	48	32	52	152
2	-	0	24	64	164
3	-	-	0	60	160
4	-	-	-	0	100
5	-	-	-	-	0

cost$[i, j]$

$i\backslash j$	1	2	3	4	5
1	-	1	1	3	3
2	-	-	2	3	3
3	-	-	-	3	3
4	-	-	-	-	4
5	-	-	-	-	-

best$[i, j]$

Task 3

Let $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ be two sequences. We say X is a *subsequence* of Y if there are indices $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that for all $1 \leq j \leq m$ it holds that $x_j = y_{i_j}$.

Use dynamic programming to implement an algorithm that runs in polynomial time which, given two sequences X and Y, computes the length of the longest common subsequence of X and Y.

Solution

Let $c[i, j]$ be the length of a LCS of $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$. We have

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i-1, j], c[i, j-1]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

We iniciate the table with 0 at position $c[i, j]$, where i or j is 0. Wlog. let $n < m$. In the first step, we compute $c[i, 1]$ for $i = 1, \ldots, n$ and $c[1, j]$ for $j = 1, \ldots, m$. In step k we compute $c[i, k]$ for $i = k, \ldots, n$ and $c[k, j]$ for $j = k, \ldots, m$. After $\min(n, m) = n$ steps we filled in exactly the whole table and we know the value $c[n, m]$. The algorithm works in time $\mathcal{O}(n \cdot m) \subseteq \mathcal{O}(m^2)$.

Example: $X = (1, 2, 4), Y = (2, 3, 4, 6)$. The goal is the value $c[3, 4]$.

$i\backslash j$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	1	1	1	1
3	0	1	1	2	2

Since $3 < 4$, we can also just fill in the table row by row.

Task 4
Construct an optimal binary search tree for the following elements v with probability (weight) $\gamma(v)$.

v	1	2	3	4	5	6
$\gamma(v)$	0.25	0.1	0.2	0.15	0.25	0.05

Solution
To compute a BST with smallest weighted inner path length, we use dynamic programming. Let cost$[i, j]$ be the weighted inner path length of the optimal BST for the node set $\{i, \ldots, j\}$ with root r$[i, j]$.

$i \backslash j$	1	2	3	4	5	6
1	0.25	0.45	0.95	1.3	1.95	2.1
2	0	0.1	0.4	0.7	1.35	1.5
3	-	0	0.2	0.5	1.05	1.2
4	-	-	0	0.15	0.55	0.65
5	-	-	-	0	0.25	0.35
6	-	-	-	-	0	0.05

cost$[i, j]$

$i \backslash j$	1	2	3	4	5	6
1	1	1	1	3	3	3
2	-	2	3	3	4	5
3	-	-	3	3	4	5
4	-	-	-	4	5	5
5	-	-	-	-	5	5
6	-	-	-	-	-	6

r$[i, j]$

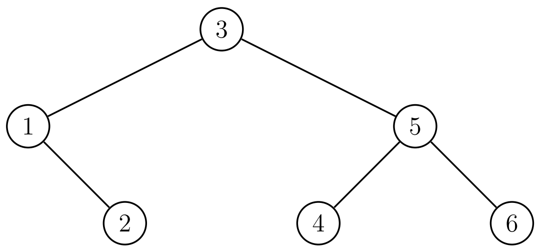
Initialize cost$[i, i-1] = 0$, cost$[i, i] = \gamma(i)$ and r$[i, i] = i$. In the next step, the node with the highest weight (probability) has to be the root node, hence we can fill in the tables at position $(i, i+1)$. The trick in the following steps is now to pick a root, where the sum of the optimal costs of the BST for the left and the right subtree plus the total weight $\Gamma[i, j] = \gamma(i) + \cdots + \gamma(j)$ is minimal (for the minimization we can ignore Γ of course). Among the optimal roots, we pick the one with the largest key (by convention).

BST of size 3: For instance $(3, 5)$; $0.55 + 0.6$ (3) vs. $0.2 + 0.25 + 0.6$ (4) vs. $0.5 + 0.6$ (5). Hence, node 4 is at the top. For the other 3 values ($(1, 3)$, $(3, 5)$ and $(4, 6)$), we do the same. In the following examples we ignore $\Gamma[i, j]$.

BST of size 4: For instance $(2, 5)$; 1.15 (2) vs. $0.1 + 0.55$ (3) vs. $0.4 + 0.25$ (4) vs. 0.7 (5). Hence, node 4 is at the top (root 3 and 4 are equally good, but we choose 4 as convention tells us). The values $(1, 4)$ and $(3, 6)$ are obtained similarly.

We skip the example for BSTs of size 5 and jump directly to size 6.

We have 1.5 (1) vs. 1.45 (2) vs. 1.1 (3) vs. 1.3 (4) vs. 1.35 (5) vs. 1.95 (6). Clearly node 3 wins. The optimal BST has weighted inner path length of $1.1 + 1 = 2.1$ and looks like this:



Task 5
Assume we want to construct an optimal binary search tree using the following greedy algorithm: Choose an element v for which $\gamma(v)$ is maximal as the root node and then continue recursively. Show that this approach does not always yield an optimal binary search tree.

Solution
Choose $\gamma_1 = \frac{1}{3} - \varepsilon$, $\gamma_2 = \frac{1}{3}$ and $\gamma_3 = \frac{1}{3} + \varepsilon$. The greedy algorithm yields a chain $(3 - 2 - 1)$ with a weighted inner path length of

$$\left(\frac{1}{3} + \varepsilon\right) \cdot 1 + \frac{1}{3} \cdot 2 + \left(\frac{1}{3} - \varepsilon\right) \cdot 3 = 2 - 2\varepsilon.$$

It is better to take the tree with root $v = 2$ (and left child 1, right child 3). We obtain a weighted inner path length of

$$\frac{1}{3} \cdot 1 + \left(\frac{1}{3} - \varepsilon + \frac{1}{3} + \varepsilon\right) \cdot 2 = \frac{5}{3} < 2 - 2\varepsilon$$

for all $0 < \varepsilon < \frac{1}{6}$.