Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel David Steurer Peter Widmayer
Chih-Hung Liu
Stefano Leucci

Datenstrukturen & Algorithmen Blatt P13 HS 17

Solution for Exercise P13.1 *Frequency Assignment.*

In addition to the m companies, indexed by $j = 1, \dots, m$, we consider and additional fictitious company, indexed by $j = 0$, that has $c_{i,0} = h_{i,0} = 0 \forall i$. Assigning a channel i to company 0 in this modified instance corresponds to leaving channel i unallocated in the original instance. We can hence consider the problem in which all channels must be assigned to some company.[1]

Let $\mathrm{OPT}[i, j]$, for $i = 0, \dots, N$ and $j = 0, \dots, m$, be the maximum amount of Flops the city of Algo can earn by a lawful assignment of the first i channels, with the additional constraint that the i-th channel must be assigned to company j. Moreover, let $\mathrm{BEST}[i]$ be the index of a company j that maximizes $\mathrm{OPT}[i][j]$, i.e., $\mathrm{BEST}[i] \in \arg\max_j \mathrm{OPT}[i][j]$. Similarly, we let $\mathrm{SBEST}[i]$ be the index of the "second best" company, namely a company $j \neq \mathrm{BEST}[i]$ that maximizes $\mathrm{OPT}[i][j]$ (formally $\mathrm{SBEST}[i] \in \arg\max_{j \neq \mathrm{BEST}[i]} \mathrm{OPT}[i][j]$). It is clear that $\mathrm{BEST}[i]$ and $\mathrm{SBEST}[i]$ can be computed in $O(m)$ time once all the values $\mathrm{OPT}[i][j], j = 0, \dots, m$ are known. Therefore we only focus on computing the values of $\mathrm{OPT}[i, j]$.

From our definitions we know that $\mathrm{OPT}[0, j] = 0$ for every $j = 0, \dots, m$, therefore we focus on $i > 0$. In particular, we now discuss the case $i \geq 3$ (the formulas for $i = 1$ and $i = 2$ will follow by similar arguments). Consider an optimal solution for $\mathrm{OPT}[i, j]$ and notice that, since it assigns channel i to company j, it must also assign at least one channel $i^* \in \{i-1, i-2, i-3\}$ to a company $k \neq j$. Moreover, if $\mathrm{BEST}[i^*] \neq j$ then k is exactly $\mathrm{BEST}[i^*]$, otherwise $\mathrm{SBEST}[i^*] \neq \mathrm{BEST}[i^*] = j$ and $k = \mathrm{SBEST}[i^*]$. For any i^* let us define[2]

$$\mathrm{prev}(i^*, j) = \begin{cases} \mathrm{BEST}[i^*] & \text{if } \mathrm{BEST}[i^*] \neq j; \\ \mathrm{SBEST}[i^*] & \text{otherwise.} \end{cases}$$

We can now derive the following recursive formula for $\mathrm{OPT}[i, j]$:

$$\mathrm{OPT}[i, j] = \max \begin{cases} v_{i,j} + \mathrm{OPT}[i-1, \mathrm{prev}(i-1, j)] \\ v_{i,j} + v_{i-1,j} + \mathrm{OPT}[i-2, \mathrm{prev}(i-2, j)] \\ v_{i,j} + v_{i-1,j} + v_{i-2,j} + h_{i-2,j} + \mathrm{OPT}[i-3, \mathrm{prev}(i-3, j)] \end{cases}$$

If $i = 1$ (resp. $i = 2$) then the previous maximum will only include the first term (resp. first and second terms). The computation of each $\mathrm{OPT}[i, j]$ requires constant time. It follows that the computation of $\mathrm{OPT}[i, j]$, $\mathrm{BEST}[i]$, and $\mathrm{SBEST}[i]$ for a fixed i and all $j = 0, \dots, m$ requires $O(m)$ time. Since there are exactly $n + 1$ values of i, the overall time complexity is $O(N \cdot m)$.

The value of an optimal solution can be found in $\mathrm{OPT}[N, \mathrm{BEST}[N]] = \max_j \mathrm{OPT}[N, j]$.

[1] It is easy to show that there always exists an optimal solution that does not assign 3 or more contiguous channels to company 0.

[2] Notice that $\mathrm{prev}(i^*, j)$ can be computed in constant time once $\mathrm{BEST}[i^*]$ and $\mathrm{SBEST}[i^*]$ are known.

Solution for Exercise P13.2 *Dynamic Order Statistic.*

The key idea is to store the elements of A in a max-heap H^-, and a min-heap H^+ such that, after each operation, the following invariant holds: *H^- contains the smallest $\left\lceil \frac{|A|}{\alpha} \right\rceil$ elements of A, H^+ contains the remaining elements (i.e., the largest $|A| - \left\lceil \frac{|A|}{\alpha} \right\rceil$ elements of A).*

We will use $|H^-|$ to denote the number of elements in H^-. The three operations can be implemented as follows:

`Reset(`n`):` Initializes the two heaps H^- and H^+ to can contain up to n elements each. This requires $O(n)$ time (e.g., if the elements of the heap are stored in an array, this operation corresponds to allocating two arrays of length n). After a `Reset(`n`)` operation the invariant is trivially satisfied as $A = \emptyset$ and the heaps are empty.

`Query():` Since H^- contains the $k = \left\lceil \frac{|A|}{\alpha} \right\rceil$ smallest elements of A, the k-th order statistic of A is exactly the largest element in H^-. This is the element associated with the root vertex of H^- and can therefore be returned in $O(1)$ time.

`Add(`x`):` If $A = \emptyset$ then H^- is empty, $\left\lceil \frac{|A \cup \{x\}|}{\alpha} \right\rceil = \left\lceil \frac{1}{\alpha} \right\rceil = 1$, and the invariant can be restored by inserting x into H^- (this requires constant time). Otherwise, if $A \neq \emptyset$, we have that $\frac{|A|}{\alpha} > 0$ which implies that H^- is not empty. We compare the largest element y of H^- with x: if $x < y$ then we insert x into H^-, otherwise we insert x into H^+. We are now in one of the following three cases:

- If $|H^-| = \left\lceil \frac{|A|+1}{\alpha} \right\rceil$, then the invariant is already satisfied and we are done.
- If $|H^-| = \left\lceil \frac{|A|+1}{\alpha} \right\rceil + 1$, then H^- contains one extra element. We can restore the invariant by extracting the maximum from H^- and inserting it into H^+. This requires 2 heap operations that can be performed in $O(\log n)$ time each.
- If $|H^-| = \left\lceil \frac{|A|+1}{\alpha} \right\rceil - 1$, then we need to add one element to H^- in order to restore the invariant. We can do so by extracting the minimum from H^+ and inserting it into H^-. This requires $O(\log n)$ time.