HW2 BAsic Algorithms

Algo L3

28.09.2016 - **Due on Thursday 5.10 before 8:30**

Please send me your solutions as a PDF file named « `HW2-BOTH_YOUR_NAMES.pdf` » at:
`Cours.AlgoL3@ens.fr`
with Subject « `[HW2]` »
(or return it at the next lecture) on **Thursday 5.10 before 8:30**.

■ **Exercise 1 (Skip List).** A *skip list* is a light-weight data structure that allows to insert, delete and search for keys. Additionally, it can answer efficiently to *predecessor queries* for any key k, that is for the element y with the largest $\texttt{key}[y] \leqslant k$ stored in the skip list.

A skip list $\mathcal{S}$ for a set S consists of a number of *sorted* linked lists $\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_h$. Each list $\mathcal{L}_i$ stores a subset $S_i \subseteq S$, such that $S_0 = S$, and $S_i \subseteq S_{i-1}$ for all $0 < i \leqslant h$, and $S_h = \varnothing$. Each sorted list also stores two dummy elements, one with key $-\infty$ at the beginning of the list and one with key $+\infty$ at the end of the list. For a set S_i (or list $\mathcal{L}_i$) we call i the *level* of that set (or list), and we call h the *height* of the skip list. We also have pointers between consecutive lists. More precisely, for every element $x \in S_i$ (with $i > 0$) we have a pointer from its occurrence in $\mathcal{L}_i$ to its occurrence at the lower level in $\mathcal{L}_{i-1}$ — see Fig. 1 for an example.

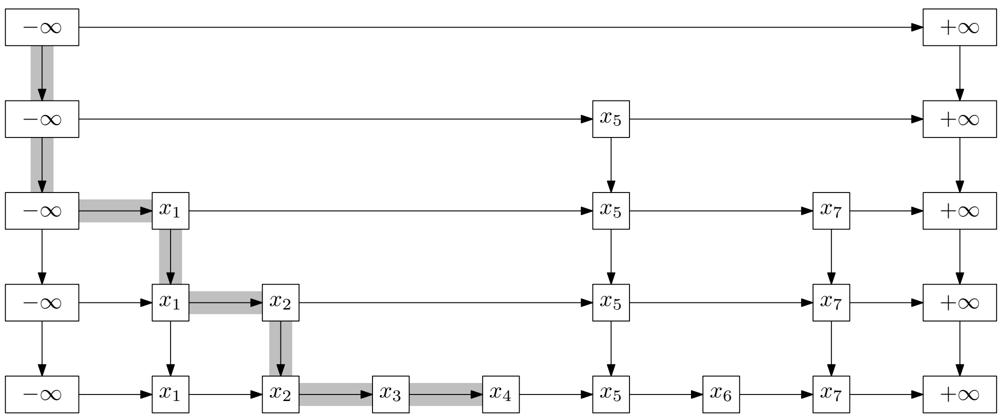

Figure 1: A skip list on a set of seven elements. The search path taken by a predecessor query with query value k (for some k with $\texttt{key}[x_4] \leqslant k < \texttt{key}[x_5]$) is indicated.

We denote the pointer from an element $x \in \mathcal{L}_i$ to the next element in $\mathcal{L}_i$ by $\texttt{next}[x]$, and the pointer from x in $\mathcal{L}_i$ to the copy of x in $\mathcal{L}_{i-1}$ by $\texttt{down}[x]$. Answering a predecessor query for key k works as follows: Start from the dummy element $-\infty$ at the left of the top level and Repeat until you cannot go down anymore: Go down and Move to the next element at the current level as long as its key is at most k; when we cannot go down anymore, Output the element at which we stopped.

► ***Question 1.1)*** *Write the procedure* $\texttt{Predecessor}(\mathcal{S},k)$ *in pseudo-code. Prove that it outputs the predecessor of* k*, i.e. the element* y *with the largest* $\texttt{key}[y] \leqslant k$ *in the set* S *encoded by the skip list* $\mathcal{S}$.

▷ Hint. *In order to prove the correctness of your algorithm, use the three properties defining a skip list: each level consists in a sorted linked list; each level contains a subset of the elements in the level just bellow; the top level is empty (i.e., contains only the two dummy elements $-\infty$ and $+\infty$) and the bottom level contains all the elements.*

In order to answer queries efficiently, we need at the same time the height to be low, $O(\log n)$, and the elements on a given level to be well spread with respect to elements at the level just bellow. It would be very costly to maintain such a structure efficiently deterministically. The key to the efficiency of skip list is to use randomness in order to decide at which level to insert a new element. Here is how the insertion of a new element x proceeds. First, choose the level i at which x is inserted: for this purpose, toss a fair coin repeatedly until **Tail** is obtained and set i to the number of times **Head** occured before **Tail** was obtained. Then, x is inserted at every level from $\mathcal{L}_i$ to $\mathcal{L}_0$.

► ***Question 1.2)*** *Write the procedure* Insert($\mathcal{S}$,x) *in pseudo-code. Be extra-careful on how you update the fields* next *and* down *(Remember that the lists are not double-linked). Prove that if $\mathcal{S}$ is a skip list, the resulting data structure remains a skip list after the insertion.*

Let n denote the *total* number of elements *ever* inserted in the skip list. The insertion procedure ensures that every level contains about half the elements in the level just bellow on expectation. This will guarantee with high probability that the height of the data structure is at most $O(\log n)$ and that the number of elements scanned at each level is $O(1)$ for each query. Let us denote by height(x) the height at which element x is inserted.

► ***Question 1.3)*** *Show that:* $\Pr\{\text{height}(x) \geqslant s\} = 1/2^s$ *for all* $s \in \mathbb{N}$.

Recall the *union bound* which states that for *any* set of events $A_1, \ldots, A_n$ (interdependent or not), $\Pr\{A_1 \vee A_2 \vee \cdots \vee A_n\} \leqslant \Pr A_1 + \Pr A_2 + \cdots + \Pr A_n$.

► ***Question 1.4)*** *Show that for all $t > 1$, the probability that the height h of the skip list is at least $1 + t \log_2 n$ is at most $1/n^{t-1}$.*
▷ Hint. *Use the union bound other the height of all elements ever inserted in the skip list.*

We are now ready to prove a bound on the query time in a skip list. Let X_i denote the random variable for the number of next-pointers followed in $\mathcal{L}_i$ when answering the query.

► ***Question 1.5)*** *Show that* $\mathbb{E}[X_i] \leqslant 1$ *for all* i.
▷ Hint. *Note that the elements in level $\mathcal{L}_i$can be seen as sampled from level $\mathcal{L}_{i-1}$ independently with probability $1/2$. How long does it take to meet a element in $\mathcal{L}_{i-1}$ that belongs to $\mathcal{L}_i$ as well?*

► ***Question 1.6)*** *Conclude that the expected time to answer a query is at most $O(\log n)$.*
▷ Hint. *Use question 1.4 to bound the expected time spent on levels possibly higher than $3 \log n$.*

Let us now consider deletion.

► ***Question 1.7)*** *Write the procedure* Delete($\mathcal{S}$,x) *in pseudo-code. Prove that if $\mathcal{S}$ is a skip list, the resulting data structure remains a skip list after the deletion.*

► ***Question 1.8)*** *Does the analysis in the questions above still hold when deletions are performed? (Recall that n is the total number of elements* ever *inserted in the skip list).*

Let m be the number of elements *presently* in the skip list ($m \leqslant n$ and m might be much lower than n if many deletions did occur).

► ***Question 1.9)*** *Is it true that the expected query time is $O(\log m)$? Prove it or disprove it with an example.*