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 $\text{key}[y] \leq k$ stored in the skip list.

A skip list $S$ for a set $S$ consists of a number of sorted linked lists $L_0, L_1, \ldots, L_h$. Each list $L_i$ stores a subset $S_i \subseteq S$, such that $S_0 = S$, and $S_i \subseteq S_{i-1}$ for all $0 < i \leq h$, 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 $S$ (or list $L_i$) 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 $L_i$ to its occurrence at the lower level in $L_{i-1}$ — 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 $\text{key}[x_4] \leq k < \text{key}[x_5]$) is indicated.](image)

We denote the pointer from an element $x \in L_i$ to the next element in $L_i$ by $\text{next}[x]$, and the pointer from $x$ in $L_i$ to the copy of $x$ in $L_{i-1}$ by $\text{down}[x]$. 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 $\text{Predecessor}(S, k)$ in pseudo-code. Prove that it outputs the predecessor of $k$, i.e. the element $y$ with the largest $\text{key}[y] \leq k$ in the set $S$ encoded by the skip list $S$. 

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.
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.

**Answer.** Algorithm ?? presents the pseudo-code for the Predecessor($S$, $k$) function together with the invariants that allow to prove its correctness.

**Algorithm 1 Predecessor($S$, $k$)**

1: Let $y$ point to the dummy element $-\infty$ of $L_h$.
2: \[\text{condition} \rightarrow \text{y points to the predecessor of k in } L_i \]
3: \text{for i := h down to 1 do}
4: \[\text{condition} \rightarrow \text{y points to the predecessor of k in } L_i \]
5: \text{end for}
6: \[\text{condition} \rightarrow \text{y is to the left of k in } L_{i-1} \]
7: \text{while key[next[$y$]]} $\leq$ $k$ do
8: \text{y := next[$y$]}
9: \[\text{condition} \rightarrow \text{y points to the predecessor of k in } L_{i-1} \]
10: \text{end while}
11: \text{return y}

The correctness of the invariants is straightforward and follows from the facts that 1) every level consists in a sorted linked list, 2) level $L_i$ is a sublist of level $L_{i-1}$, 3) for each $y$ in $L_i$, next[$y$] points to the occurrence of $y$ in $L_{i-1}$, and 4) $L_h$ consists only in the list $-\infty$, $+\infty$ whereas $L_0$ contains all the elements in $S$. \H

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 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 (we assume that the keys of all the elements are distinct, in particular that key[$x$] is not already in $S$). 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 occurred before Tail was obtained. Then, $x$ is inserted at every level from $L_i$ to $L_0$.

**Question 1.2** Write the procedure Insert($S$, $x$) 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 $S$ is a skip list, the resulting data structure remains a skip list after the insertion.

**Answer.** The insertion algorithm is an adaption of the Predecessor procedure. Algorithm ?? presents the pseudo-code together with the invariants that demonstrates its correctness.

Again the invariants are straightforward to prove using that the properties defining a skip list. Note that we insert $x$ from top to bottom at the same time as we pursue the descent to the level $L_0$ in order to match the orientation of the pointers. \H

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: $\Pr\{\text{height}(x) \geq s\} = 1/2^s$ for all $s \in \mathbb{N}$. 


Let $\ell := 0$

while newRandomBit() == 0 do

$\ell := \ell + 1$

----------- $\checkmark$ Pr$\{\ell \geq i\} = 2^{-i}$ for all $i \in \mathbb{N}$ -----------

for $i := h + 1$ to $\ell + 1$ do

$L_i$ := new list containing two dummy elements $-\infty$ and $+\infty$

Set the pointers down of $-\infty$ and $+\infty$ in $L_i$ to their copies in $L_{i-1}$.

$h := \max(h, \ell + 1)$

----------- $\checkmark$ The height of $S$ has been extended (if needed) to $h \geq \ell + 1$ -----------

Let $y$ point to the dummy element $-\infty$ of $L_h$.

Let $z := \text{null}$.

----------- $\checkmark$ $y$ points to the predecessor of key$[x]$ in $L_h$ -----------

for $i := h$ down to 1 do

----------- $\checkmark$ $L_i$ is unchanged if $i > \ell$, and -----------

----------- $\checkmark$ $L_i$ now contains a copy of $x$ at its rightful position if $i \leq \ell$ -----------

----------- $\checkmark$ $y$ points to the predecessor of key$[x]$ in $L_i$ if $i > \ell$, -----------

----------- $\checkmark$ $y$ points to the predecessor of the copy of key$[x]$ in $L_i$ if $i \leq \ell$, -----------

----------- $\checkmark$ $y$ points to the element to the left of the copy of $x$ in level $L_i$ if $i \leq \ell$, -----------

----------- $\checkmark$ $y$ points to the copy of $x$ in level $L_i$ if $i \leq \ell$, and -----------

----------- $\checkmark$ $y$ points to the correct position in $L_i$ -----------

$y := \text{down}[y]$

----------- $\checkmark$ $y$ is to the left of key$[x]$ in $L_{i-1}$ -----------

while key$[\text{next}[y]] < \text{key}[x]$ do

$y := \text{next}[y]$

----------- $\checkmark$ $y$ is the predecessor of key$[x]$ in $L_{i-1}$ -----------

if $i - 1 \leq \ell$ then

// Insert a copy of $x$ just after $y$ in $L_{i-1}$


Set $z := t$.

----------- $\checkmark$ $x$ has been inserted at every level from $L_\ell$ down to $L_0$ at the correct positions -----------

return $y$

---

Answer. $\triangleright$ The probability that height$[x] \geq s$ is the probability that the outcomes of a fair coin are Head $s$ times in a row, which is $1/2^s$. $\triangleright$

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

$\triangleright$ Question 1.4 $\triangleright$ 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 $1/n^{t-1}$.

$\triangleright$ Hint. Use the union bound other the height of all elements ever inserted in the skip list.

Answer. $\triangleright$ In a skip list, $h = 1 + \max_{i=1, \ldots, n}$ height$[x_i]$. Thus, using the union bound:

$$
\text{Pr}\{h \geq 1 + t \log_2 n\} = \text{Pr}\left\{\bigvee_{i=1}^{n} (\text{height}[x_i] \geq t \log_2 n)\right\} \leq \sum_{i=1}^{n} \text{Pr}\{\text{height}[x_i] \geq t \log_2 n\} \\
\leq n \cdot 2^{-t \log_2 n} = 1/n^{t-1}.
$$

$\triangleright$

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 $L_i$ when answering the query.
**Question 1.5)** Show that $\mathbb{E}[X_i] \leq 1$ for all $i$.

**Answer.** The key to a simple analysis is to consider the path followed in reverse: starting from a node at level $L_i$ and going backward (to the left), how long does it take to encounter on level $L_k$ a node whose height is at least $i + 1$? $X_i$ is the random variable corresponding to this quantity. Since every node in a given level belongs to an higher level independently with probability $1/2$, $\Pr\{X_i \geq s\} \leq 1/2^s \leq 1$ and not $= 1$ because we might run out of elements in the set if we hit $\omega$. It follows that $\mathbb{E}[X_i] \leq \sum_{k=i}^{\infty} 1/2^k = 1$. (Recall that for any non-negative integer-valued random variable $Y$, $\mathbb{E}[Y] = \sum_{y=1}^{\infty} \Pr\{Y \geq y\}$. Indeed, $\mathbb{E}[Y] = \sum_{t=1}^{\infty} t \cdot \Pr\{Y = t\} = \sum_{t=1}^{\infty} \sum_{y=1}^{t} \Pr\{Y = t\} = \sum_{y=1}^{\infty} \Pr\{Y \geq y\}$ by Fubini's theorem.)

**Question 1.6** Conclude that the expected time to answer a query is at most $O(\log n)$.

**Answer.** Use question 1.4 to bound the expected time spent on levels possibly higher than $3 \log n$.

Let us now consider deletions. The procedure is very similar to the insertion. Algorithm 2 presents the pseudo-code together with the invariants that demonstrate its correctness.

Again the invariants are straightforward to prove using that the properties defining a skip list. Note that, as before, we delete $x$ from top to bottom at the same time as we pursue the descent to the level $L_0$ in order to match the orientation of the pointers. Remark that the procedure does nothing if $\text{key}[x]$ is not present in the skip list.
Algorithm 3 Delete($S,x$)

1: Let $y$ point to the dummy element $-\infty$ of $\mathcal{L}_h$.

2: $\cdots$ $\checkmark$ $y$ points to the predecessor of $\text{key}[x]$ in $\mathcal{L}_h$ $\cdots$

3: for $i := h$ down to 1 do

4: $\cdots$ $\checkmark$ $y$ points to the predecessor of $\text{key}[x]$ in $\mathcal{L}_i$ $\cdots$

5: $\cdots$ $\checkmark$ key[x] does not appear anymore at levels $\mathcal{L}_j$ for $j \geq i$ $\cdots$

6: $y := \text{down}[y]$

7: $\cdots$ $\checkmark$ $y$ is to the left of $\text{key}[x]$ in $\mathcal{L}_{i-1}$ $\cdots$

8: while $\text{key[next}[y]] < \text{key}[x]$ do

9: $\cdots$ $\checkmark$ $y$ points to the predecessor of $\text{key}[x]$ in $\mathcal{L}_{i-1}$ $\cdots$

10: $\cdots$ $\checkmark$ next[y] is the copy of x in level $\mathcal{L}_{i-1}$ $\cdots$

11: if $\text{key[next}[y]] = \text{key}[x]$ then

12: $\cdots$ $\checkmark$ next[y] is the copy of x in level $\mathcal{L}_{i-1}$ $\cdots$

13: // delete next[y] from $\mathcal{L}_{i-1}$


15: $\cdots$ $\checkmark$ key[x] has been deleted from level $\mathcal{L}_{i-1}$ if it was present $\cdots$

16: $\cdots$ $\checkmark$ x has been deleted from every level. $\cdots$

$\triangleright$ **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).

**Answer.** $\triangleright$ Note that a priori deletions should only speed up the searching process because it shortens the length of the paths at each level. This is true if deletions are performed regardless of the actual random heights of the elements. As we will see in the next question, if deletions are performed by an adversary who is aware of the actual heights, all the benefits of the skip list data structure can be lost. $\triangleright$

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

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

**Answer.** $\triangleright$ Just insert $n$ elements in the skip list. Then delete all the elements with positive height. There remain on expectation $m = n/2$ elements, all of height 0. The skip list is thus reduced to the simple linked list $\mathcal{L}_0$. Every operation takes then $\Theta(n) \gg \log_2 n$ on average! Deletions must then be handled more carefully against an adversary if one wants to preserve the benefits of the skip list in the worst case. $\triangleright$