Searching a key $x$ in a sorted linked list

1. cell *$p$ = head ;
2. while ($p$->key < $x$) $p$ = $p$->next ;
3. return $p$ ; // (which is either equal or larger than $x$)

Note:
- The $-\infty$ and $\infty$ elements are not "real" keys.
- They are in the list to prevent checking special cases.
- Sometimes we prefer to return the element preceding the one containing $x$. Then line 2 is replaced with:
  while ($p$->next->key < $x$) $p$ = $p$->next;

Inserting a key into a Sorted linked list

To insert 35 -
- $p$= find(35); // find the proceeding element – the next one is > 35
- CELL *$p1$ = (CELL *) malloc(sizeof(CELL));
- $p1$->key=35;
- $p1$->next = $p$->next ;
- $p$->next = $p1$;
To delete 37 -
- \( p = \text{find}(37); \) // Again find proceeding element
- \( \text{CELL} \ast p1 = p->\text{next}; \)
- \( p->\text{next} = p1->\text{next}; \)
- \( \text{free}(p1); \)

**SKIP LIST - A data structure for maintaining keys in a sorted order**

**Rules:**
- Consists of several levels.
- All keys appear in level 1
- Each level is a sorted list.
- If key \( x \) appears in level \( i \), then it also appears in all levels below level \( i \)

- First element in each level has key \( -\infty \).
- Last element has key \( +\infty \)
- First element in upper level is pointed to by variable \( \text{top} \).

**More rules**

- An element in level \( i > 1 \) points (via down pointer) to the element with the same key in the level below.
- Elements in the lowest level have down-pointer=NULL
- Also maintain a counter specifying the number of levels.
Finding an element with key x

```c
while(1){
    while (p→next→key < x) p=p→next;
    if (p→down == NULL) return p;
    p=p→down;
}
```

Observe that we return \( \text{pred}(x) \) (the key preceding \( x \)).

Inserting new element \( x \)

Determine \( k \), defined as the number of levels in which \( x \) participates (explained later how).

Perform \( \text{find}(x) \), but once the search path is in one of the lowest \( k \) levels:
- \( x \) is inserted after the elements at which the search path branches down or terminates.
- The next pointer behave like a “standard” linked list.
- The down pointer(s) point between themselves.

Example - inserting 119, \( k=2 \)
Inserting an element - cont.

- If $k$ is larger than the current number of levels, add new levels (and update $top$, and $num\_of\_levels$ counter)
- Example - insert(119) when $k=4$
- Heuristic: Add at most one new level (not needed for the analysis)

```
Example - insert(119) when k=4
```

Determining $k$

- $k$ - the number of levels at which an element $x$ participate.
- Use a random function $OurRnd()$ --- returns 1 or 0 (True/False) with equal probability.
  - $k=1$;
  - While ($OurRnd()==1$) $k++$;

Deleteing a key $x$

- Find $x$ in all the levels it participates, using find($x$).
- During the “find”, delete $x$ from each level it participates using the standard “delete from a linked list” method.
- If one or more of the upper levels become empty, remove them (and update $top$ and $num\_of\_levels$)
"expected" on what?

- **Claim:** The expected number of elements is $\mathcal{O}(n)$.
- The term "expected" here refers to the experiments we do while tossing the coin (or calling `OurRnd()`). No assumption about input distribution.
- So imagine a given set, given set of operations insert/ del/find, but we repeat many times the experiments of constructing the SL, and count the #elements.

Facts about SL

- **Def:** The height of the SL is the number of levels
- **Claim:** The expected number of levels is $\mathcal{O}(\log n)$
- (here $n$ is the number of keys)
- "**Proof**"
  - The number of elements participate in the lowest level is $n$.
  - Since the probability of an element to participates in level 2 is 1/2, the expected number of elements in level 2 is $n/2$.
  - Since the probability of an element to participates in level 3 is 1/4, the expected number of elements in level 3 is $n/4$.
  - ... 
  - The probability of an element to participate in level $j$ is $(1/2)^{j-1}$ so number of elements in this level is $n/(2^j)$
  - So after $\log(n)$ levels, no element is left.

Facts about SL

- **Claim:** The expected number of elements is $\mathcal{O}(n)$.
- (here $n$ is the number of keys)
- "**Proof**" (a rigorous proof requires the use of random variables)
  - The total number of elements is $n+n/2+n/4+n/8... \leq n(1+1/2+1/4+1/8...) = 2n$
  - To reduce the worst case scenario, we verify during insertion that $k$ (the number of levels that an element participates) in) is $\leq \log n$.
- **Conclusion:** The expected storage is $\mathcal{O}(n)$
More facts

**Thm:** The expected time for find/insert/delete is $O(\log n)$

**Proof**

For all Insert and Delete, the time is $s$

expected #elements scanned during find($x$) operation.

Will show: Need to scan expected $O(\log n)$ elements.

---

**Thm:** Expected time for `find` operation is $O(\log n)$

**Proof**

– we know that there are $O(\log n)$ levels. Will show that we spend $O(1)$ time in each level.

Assume during find($x$), we scanned $t$ elements, (for $t>8$) in level $r$. Assume first that $r$ is not the upper level.

*(the search visited $b_1$, branched down to $b_2$ and then visited $b_3...b_8$ (not sure what happened before or after)*

Level $r+1$

Level $r$

$\leq x$

All smaller than $x$

None of these 7 elements reached level $r+1$ *(why?)*

The probability that none of these 7 elements reached level $r+1$ is $1/2^7$. For larger value of 7 – very slim.

---

**Bounding time for insert/delete/find**

Putting it together The expected number of elements scanned in each level is $O(1)$

There are $O(\log n)$ levels

Total time is $O(\log n)$

As stated, getting bounds for time for insert/delete are similar
How likely is that the SL is too tall?

Let's ask how likely it is that the \#levels is $2 \log_2 n$, where $Z=1,2,3...$

That is, we estimate the probability that the height of the SL is

- $\log_2 n$
- $2 \log_2 n$
- $3 \log_2 n$
- $4 \log_2 n$
- ...

Reminder from probability

- Assume that $A, B$ are two events. Let
  - $\Pr(A)$ be the probability that $A$ happens,
  - $\Pr(B)$ be the probability that $B$ happens
  - $\Pr(A \cup B)$ is the probability that either event $A$ happens or event $B$ happens (or both).
- So probably that at least one of them happened is $\Pr(A) + \Pr(B) - \Pr(A \cap B) \leq \Pr(A) + \Pr(B)$

Similarly, for 3 Events $A_1, A_2, A_3$. The probability that at least one of them happens $\Pr(A_1 \cup A_2 \cup A_3) \leq \Pr(A_1) + \Pr(A_2) + \Pr(A_3)$

Example: In a roulette, we pick a number $k$ between 1..38

- Event $A$: $k$ is even. $\Pr(A)=Pr(k \text{ is even}) = 19/38 = 0.5$
- Event $B$: $k$ is divided by 3. $\Pr(B)= 12/38=0.315$
- $\Pr(A \text{ or } B) = \Pr((k \text{ is divided by 2}) \text{ or } (k \text{ is divided by 3})) = 0.5 + 0.3 = 0.8$

But how likely is that the SL is too tall?

- Assume the keys in the SL are $\{x_1, x_2, ..., x_n\}$
- The probability that $x_i$ participates in at least $k$ levels is $2^{-k}$
  (same probability for all $x_i$).
- Define: $A_i$ is the event that $x_i$ participates in $\geq k$ levels.
  $\Pr(A_i) \leq 2^{-k}$
- Define: $A_j$ is the event that $x_j$ participates in $\geq k$ levels
  $\Pr(A_j) \leq 2^{-k}$
- If the height of SL $\geq k$ then at least one of the $x_i$ participate in $\geq k$ levels.
- The probability that any $x_i$ participates in $\geq k$ levels is $\leq \Pr(A_1) + \Pr(A_2) + ... + \Pr(A_n) = n \cdot 2^{-k}$
- This is the probability that the height of the SL is $\geq k$
But how likely is that the SL is tall?  

The probability that any $x$ participates in at least $k$ levels is $\leq n 2^k$. Then the height of the SL $\geq k$.

Recall $y(ab) = (y^a)^b$.

Write $k = z \log n$, and recall that $2^{\log n} = n$.

Want to find: The probability that the height is $z$ times $\log n$.

Twice, 3 time, 4 times...

$2^k = 2^{z \log n} = (2^{\log n})^z = n^z = 1/n^z$

So $n 2^k \leq n / n^z = 1/n^z$

This is the probability that the height of SL $\geq z \log n$

Example: $n = 1000$.

The probability that the height $\geq 7 \log n$ is $\leq 1/1000^6 = 1/10^{18}$

The prob. that the height $\geq 10 \log n$ is $\leq 1/1000^9 = 1/10^{27}$

In other words (and with some hand-waving)

Assume we have a set of $n > 1000$ keys, and we keep rebuilding Skiplists for them.

Call a SL bad if its height $> 7 \log n$.

First build SL$_1$.

Then build SL$_2$ (for the same keys)

Then ...

Then SL$_M$ where $M = 10^{20}$.

Then less than 100 of them are bad.

Using similar techniques we can also bound the probability that the search takes more than $z \log n$. 