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

\[ p = \text{find}(37); \] // Again find proceeding element

\[ \text{CELL} \ast p1 = p->next; \]

\[ p->next = p1->next; \]

\[ \text{free}(p1); \]
An empty SkipList

```
Finding an element with key \( x \)

```

```
Finding an element with key \( x \)
```

If the key \( x \) is in SL, we return a pointer to the lowest element contain \( x \).
If \( x \) is not in SL, return pointer to lowest predecessor.

```
A "perfect" SkipList
```

```
A "perfect" SkipList
```

```
A "perfect" SkipList
```

Scheme for creation a well-performing SL

• Start from Level 1 (lowers level)
• For \( i=2,3 \).

Generating of Level \( i \), we scan the keys in level \( i-1 \).
Each second key is "promoted" to participate in level \( i \) as well.
A "perfect" SkipList

A SL is Perfect if between every two consecutive keys of level \( i \) there is exactly one key of level \( i-1 \).

Scheme for creation a well-performing SL

- Start from Level 1 (towers level)
- For \( i \geq 2 \)
- Generation of Level \( j \):
  - we scan the keys in level \( i-1 \)
  - Each second key is "promoted" to participate in level \( i \) as well.

Most SL are not perfect.

Another example

```c
p=top;
while(1){
  while (p->next.key ≤ x) p=p->next;
  if (p->down == NULL) return p;
  p=p->down;
}
```

Inserting new element \( x \)

- Determine \( k \geq 1 \) defined as the number of levels in which \( x \) participates (explained later how)
- Perform 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.
**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)

![Diagram of insertion](image)

**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 $num\_of\_levels$)

![Diagram of deletion](image)
**“expected” space requirement**

- **Claim**: The expected number of elements is $O(n)$.

- 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 $O(\log n)$
  (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.

- **Claim**: The expected number of elements is $O(n)$.
  (here $n$ is the number of keys)
- **Proof** (a rigorous proof requires the use of random variables)
  - The total number of elements is
    
    $n + n/2 + n/4 + n/8 + ... \approx n(1 + 1/2 + 1/4 + 1/8 + ...)=2n$

  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 $O(n)$
More facts

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

- **Proof** For all Insert and Delete, the time is ≤ 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 $t \geq 8$) 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 $r+1$ > x
- Level $r$ ≤ 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?

- Lets ask how likely it is that the \#levels is \(Z \log_2 n\), where \(Z = 1, 2, 3, \ldots\)

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

\[ \log_2 n \]
\[ 2 \log_2 n \]
\[ 3 \log_2 n \]
\[ 4 \log_2 n \]
\[ \ldots \]

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 \(\{x_0, x_1, \ldots, x_n\}\)
- The probability that \(x_j\) participates in at least \(k\) levels is \(2^{-k}\)
  (same probability for all \(x_j\)).
- Define: \(A_j\) is the event that \(x_j\) participates in \(\geq k\) levels.
  \[ \Pr(A_j) \leq 2^k \]
- 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 \(x_j\) participate in \(\geq k\) levels.
- The probability that any \(x_j\) participates in \(\geq k\) levels is \(\leq \Pr(A_j) + \Pr(A_j) + \ldots + \Pr(A_j) = n \cdot 2^k\)
- 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 \( \leq n2^{-k} \). 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...
- Then \( 2^k = 2^{(Z \log n)} = (2^{\log n})^Z = n^Z = 1/n^Z \)
- So \( n2^k \leq n / n^Z = 1/n^Z \)
- 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 \( \leq 1/1000 \).
- The probability that the height \( \leq 10 \log n \) 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 \textit{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 \textit{bad}.

Using similar techniques we can also bound the probability that the search takes more than \( Z \log n \).