SkipList

Alon Efrat
Computer Science Department
University of Arizona

Searching a key in a Sorted linked list

- Searching an element $x$
  - cell *p = head;
  - while (p->next->key < x) p = p->next;
  - return p;

Note: we return the element proceeding either the element containing $x$, or the largest element with a key smaller than $x$ (if $x$ does not exists)

Inserting a key into a Sorted linked list

To insert 35 -
- p = find(35);
- CELL *p1 = (CELL *) malloc(sizeof(CELL));
- p1->key = 35;
- p1->next = p->next;
- p->next = p1;
To delete 37 -
- \( p = \text{find}(37); \)
- \( \text{CELL } *p1 = p->\text{next}; \)
- \( p->\text{next} = p1->\text{next}; \)
- \( \text{free}(p1); \)

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

**Rules:**
- Consists of several levels.
- All keys appear in level \( \infty \).
- 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.
- We also have a counter specifying the number of levels.
An empty SkipList

Level 1

Top

Finding an element with key $x$

$p = \text{top}$;

while(1){
    while ($p->\text{next}\rightarrow\text{key} < x$) $p = p->\text{next}$;
    if ($p->\text{down} == \text{NULL}$) return $p->\text{next}$
    $p = p->\text{down}$;
}

Observe that we return the element in the lowest level containing $x$, (if exists), or $\text{pred}(x)$ if $x$ is not in the SList.

Inserting new element $x$

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

2. Do $\text{find}(x)$, 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 $\text{next-pointer}$ behave like a "standard" linked list
   - The $\text{down-pointer}$ points 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 $k=4$

```
Level 1
Level 2
Level 3
Top
```

```
7
14
21
```

```
71
85
```

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($OurRnd()$) $k++$;

Deleteing a key $x$

- Find $x$ in all the levels it participates, using $\text{find}(x)$.
- 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 $num\_of\_levels$)
Facts about SL

Claim: The expected number of levels is $O(\log n)$

(here $n$ is the number of keys)

"Proof" (a rigorous proof requires the use of random variables)

- The number of elements participate in the lowest level is $n$. Since the probability of an element to participates in level 2 is $\frac{1}{2}$, the expected number of elements in level 2 is $\frac{n}{2}$.
- Since the probability of an element to participates in level 3 is $\frac{1}{4}$, the expected number of elements in level 3 is $\frac{n}{4}$.
- ...
- The probability of an element to participates in level $j$ is $\frac{1}{2^{j-1}}$ so $\frac{n}{2^{j-1}}$.
- So after $\log(n)$ levels, no element is left.

Facts about SL

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 $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8}... \leq 2n$

To reduce the worst case scenario, we verify during insertion that $k$ (the number of levels that an element participates in) is $\leq \log n$.

Facts about SL

Thm: The expected number of elements scanned by a find operation is $O(\log n)$

Proof – we know that there are $O(\log n)$ levels. Will show – 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.

Level $r+1$
Level $r$

None of these 7 elements reached level $r+1$

The probability that none of these 7 elements reached level $r+1$ is $\frac{1}{2^7}$. For larger value of 7 – very slim.
Facts about SL

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

- **Proof** For all 3 operations, the time is bounded by the number of elements need to be scan during find($x$) operation, which is $O(\log n)$