Point location and Persistent Data structures

Alon Efrat
Computer Science Department
University of Arizona

First problem

- Given a subdivision of the plane into triangles, create a data structure, such that given a query point, we can find in which triangle it lies.

Too hard ;-)

A question that we would solve...

- A simpler problem

Given a set of horizontal segments, create a data structure, so that given a query point q, we can answer quickly which segment is vertically above the query point.

A really really really easy question...

- Same question, but for a set of horizontal segments, all having the same x-coordinates.

For simplicity, we use the y-coordinate of a segment as the "name" of the segment

Solution: Skip list

Solution: Store the y-coordinates of the segments in a SkipList. Once a query point (x,y) is given, perform succ(y)

Different question

- In a city, people are born, and die.
- Each person is recognized by its height (no two people have the same height). We denote height by the letter y, and (birth/death) date by x.
- Need a DS that would find Find(x,y) - who was the person that was alive at date x, and her/his height is y (or if not exists), larger and as close as possible to y.
**Different problem**

How to delete an element from the skipList, without destroying it?

Idea #1: Copy the whole SkipList, and delete - too much memory

Idea #2: Copy the path that changes during the deletion, then modify this path.

---

**Virtually copying SL**

To create the new virtual copy:
- Start from TOP of the old SL, create a new top, named Top2
- Do find(x) /* x is the key to be deleted */
- Copy and connect every element along the search path.
- Delete x from the SL pointed to by Top2 /*it does not affect the SL pointed to by Top1 - only blue pointers change*/
- Need to be a bit careful in the deletion (next slide)

---

**Need a little new function**

```c
void Follower(SL* sl, int x, int d) {
    // Returns a pointer to the smallest cell at level d, with key strictly greater than x
    P = sl->top; int d1 = sl->l;
    while(1) {
        while(p->key < x) p = p->nxt;
        if(d1 == d) return p;
        assert(p->down != NULL); //add #include<assert.h>
        p = p->down;
        d1--;
    }
}
```

---

**Virtually copying SL**

To delete 37 -
- We copy as before the search path (brown path)
- In each level d at which appear, we delete it using the command p->nxt = follower(sl, 37, 3)

---

**Something fishy**

But it should not be.

The new SL (let's call it SL(2)) is a perfectly legal SL. It has the same keys of SL(1), excluding 71 that was deleted.

---

**Inserting a key**

How to insert a key into the skipList, without destroying it?

Same idea: Assume we want to insert(75)
- Do search(75).
- Copy every element that the search goes through.
- Let Top2 point to the top of the list.
- Insert(72) into the SL pointed to by Top2 - only blue elements change
Again - a brandnew SL

- Note - again we obtained a perfectly legal SL.
- We have two SkipLists - one contains 73, the other one does not contain 73.
- We can now insert/delete elements into/from SL(2)
- **Remember**: to access a SL, one only need the root - the top.

How much space

- We saw that the average length of a path is $O(\log n)$, so each insert/delete takes $O(\log n)$ time and space.

Back to the birth/death question

- In a city, people are born, and die.
- Each person is recognized by its height (no two people have the same height). We denote height by the letter $y$, and (birth/death) date by $x$.
- Need a DS that would find $\text{Find}(x,y)$ - who was the person that was alive at date $x$, and her/his height is $y$ (or if not exist), larger and as close as possible to $y$.

We can solve this one by...

- We split the time axis into time-intervals.
  - We split the time at each birth or death.
  - During the same interval, no birth or death occurs.
  - We create the SL of one interval by inserting (birth) or deleting a segment (person) from the SL of the previous interval.

And remembering that this one is easy...

- All births/deaths start at the same date
- Call this problem the same-population problem (no births no deaths)
- Easily solved via standard skip list

How to access the different SL?

- So we obtain $2n+1$ SkipLists, (one for time intervals).
- The roots of all different SL are stored in a new SL (call it dates SL), sorted by date.
- The SkipLists of the people are called People SL, sorted by height.
How to access the different SL?

Alg: Create an empty people-SL
Create the dates-SL, and insert each birth/death event.
Scan these events (in increasing dates), and for each event (date) do:
Insert/delete (virtually) into/from the people SL.
Link the TOP of the new people-SL with the current key of the dates-SL.

Answering a query

A query point \((x,y)\) - which segment is vertically above the point \((x,y)\)

Use the people SL to find the time interval containing \(x\), and the corresponding people SL (use `successor(x)` operation in Dates SL).
Perform `successor(y)` operation in this people SL to find the segment above \((x,y)\).

Overall data structure

- Dates SL (only lower level shown)
- Each cell in the lower level of the dates SL points to the People SL of the time interval ending at this cell.