CS 261

Skip Lists Practice
by
Tim Budd
Sinisa Todorovic
Skip Lists

List

topSentinel

Sentinel

Sentinel
Skip Lists – Have It All
Pugh 1989

• Fast addition: $O(\log n)$
• Fast search: $O(\log n)$
• Fast removal: $O(\log n)$

• Disadvantage: - *Slightly* more complicated
Contains Skip List

1. Start at topmost sentinel
2. Loop as follows
   1. Slide right, get a link right before
   2. If next element is OK, return true
   3. If no down element, return false
   4. Move down
Skip Link

```c
struct skipLink{
    TYPE value;
    struct skipLink *next;
    struct skipLink *down;
}

struct skipList{
    struct skipLink *topSentinel;
    int size
}
```
struct skipLink * slideRight
   (struct skipLink * current, TYPE d)
{
    while (((current->next != 0) &&
             LT(current->next->value, d) )
          current = current->next;

    return current;
}
Skip List Contains

```c
int skipListContains (struct skipList *slst, TYPE e) {
    struct skipLink *current;
    current = slst->topSentinel;

    while (current) {
        current = slideRight(current, e);

        if ((current->next != 0) &&
            EQ(current->next->value, e)) return 1;

        current = current->down;
    }

    return 0;
}
```
Complexity of Contains

- Makes zig-zag motion to bottom
- Proportional to height
- $O(\log n)$
Remove Skip List

- Start at topmost sentinel
- Loop as follows
  - Slide right, get a link right before
  - If next element is OK, remove it
  - If no down element,
    - reduce size
    - else, move down
void skipListRemove (struct skipList *slist, TYPE e) {
    struct skipLink *current, *temp;
    current = slist->topSentinel;

    while (current) {
        current = slideRight (current, e);
        if ((current->next != 0) &&
            EQ(current->next->value, e)) {
            temp = current->next;
            current->next = current->next->next;
            free(temp);
            if (current->down == NULL) slist->size--;
        }
    }
    ...
}
void skipListRemove (struct skipList *slst, TYPE e) {
    struct skipLink *current, *temp;
    current = slst->topSentinel;

    while (current) {
        current = slideRight (current, e);
        if ((current->next != 0) &&
            EQ(current->next->value, e)) {
            ...
        }
        current = current->down;
    }
}
Note about Remove

- Only decrement size at bottom level
- Makes zig-zag motion to bottom
- Proportional to height
- $O(\log n)$
How to construct a skip list when we do not know the number of elements in advance?
Add Skip List

• Add the element to the bottom list
  – must increment size

• Move up by flipping a coin, and add the element to the next higher list as long as heads (H)

• If H at the top list, and if # of lists < log (size)
  – make a new top list,
  – add that element to the new top list
Add

void add (struct skipList *slst, TYPE e)
{
    /* recursively add the new element */
    downLink = addLink(slideRight(slst, e), e);

    if (!downLink && (coinFlip == heads)){/*if top list and heads*/
        make a new link for e, pointing down to downLink
        and
        make new sentinel, pointing right to new link
        and down to existing sentinel
    }
}
Add

Insert: 9 14 20 21 7
Coin toss: T H T T T H H T
Add

Insert:  9  14  20  21  7
Coin toss:  T  H  T  T  T  H  H  T
Complexity of Add

- Proportional to height, not to the number of nodes in the list
- $O(\log n)$
Useful Properties of Skip Lists

- Insertion is $O(\log n)$
- Keep elements in order
- These can be combined for sorting
Skip List Sorting Algorithm

Problem: Sort an array A

Step 1. Copy elements from A into a skip list

Step 2. Copy elements from the skip list to A
Skip List Sorting Algorithm

Complexity:

Step 1. Copy elements from A into a skip list
  \( O(??) \)

Step 2. Copy elements from the skip list to A
  \( O(??) \)