CS 261 - Spring 2011

Skip Lists
by
Tim Budd
Sinisa Todorovic
Complexity – Lists and Arrays

• Ordinary linked lists and arrays have:
  – fast addition $O(1)$
  – slow search $O(n)$

• Sorted Arrays have:
  – fast search $O(\log n)$
  – slow insertion $O(n)$
Complexity – Lists and Arrays

• Ordinary linked lists and arrays have:
  – fast addition \( O(1) \)
  – slow search \( O(n) \)

• Sorted Arrays have:
  – fast search \( O(\log n) \)
  – slow insertion \( O(n) \)

What about sorted lists?
Sorted Linked List
Operations for Sorted Linked Lists

• Add:
  – find correct location,
  – add the element

• Contains:
  – find correct location,
  – check if the element is in the list

• Remove:
  – find correct location,
  – remove the element if found in the list
Sorted List Structure Definition

struct list {
    struct link * frontSentinel;
    struct link * backSentinel;
    int size;
};
Find an Element in a Sorted List

```c
struct link * slideRightSortedList
    (struct link *current, TYPE e)
{
    while ((current->next != 0)
        && LESS_THAN(current->next->value, e))
        current = current->next;
    return current;
}
```

Finds the link RIGHT BEFORE

![Diagram of a sorted list with pointers indicating the links before and after the search target.]
Add Sorted List

```c
void addSortedList (struct list* lst, TYPE e) {
    struct link * current =
        slideRightSortedList(lst->frontSentinel, e);
    struct link * newLink =
        (struct link *) malloc(sizeof(struct link));
    assert (newLink != 0);
    newLink->value = newValue;
    ....
}
```
Add Sorted List

```c
void addSortedList (struct list* lst, TYPE e) {
    ... (see previous page)
    newLink->value = newValue;
    newLink->next = current->next;
    newLink->previous = current;
    current->next->previous = newLink;
    current->next = newLink;
    lst->size++;
}
```
void removeSortedList (struct list *lst, TYPE e) { 
    struct link * current = 
        slideRightSortedList(lst->frontSentinel, e);
    if ((current->next != 0) && 
        (EQ(current->next->value, e))){
        ....
    }
}
Remove Sorted List

```c
void removeSortedList (struct list *lst, TYPE e) { 
    ... (see previous page)
    if ( ... ){
        struct link * temp = current->next;
        current->next = current->next->next;
        current->next->previous = current;
        free(temp);
        lst->size--;
    }
}
```
Sorted Linked List

• What is complexity of:
  – search $O(??)$
  – insertion $O(??)$
  – removal $O(??)$

• Because we do not have random access
Problem with Sorted List

• What’s the use?
• Add, contains, remove $\Rightarrow$ $O(n)$
• No better than an ordinary list
• Major problem: sequential access
Sorted Linked List

- How to make a sorted linked list have faster operations?
Sorted Linked List

• Should I add more pointers?
  – E.g., we could add log n pointers
Adding more pointers…

- In theory this would work
- Would give us $O(\log n)$ search
- Hard to maintain insertion and removal
Sorted Linked List

- Or, we could use two sorted linked lists, with pointers between them
Two Sorted Linked Lists

- Constructed from the same elements
- Establish pointers between equal links
Motivation

• Regular Trains vs. Express Trains
Two Sorted Linked Lists

- List 2: stores all elements
- List 1: stores only a subset of elements
How to Search for an Element?

- We start from the 1\textsuperscript{st} element of List 1
- Stay on the "express line" as long as you can
- Then, take the "local line"
How to Choose Elements for List 1?

- Goal: Maximize fast access to all elements
- List 1 picks uniformly a subset of elements
What is Complexity of Search?

List 1

List 2
What is Complexity of Search?

\[ |L_1| + \frac{|L_2|}{|L_1|} \]

num. of elements in List 1

num. of elements in a segment of List 2
What is Complexity of Search?

\[ |L_1| + \frac{|L_2|}{|L_1|} = n \]

\[ = x \]
What is Complexity of Search?

minimize $x + \frac{n}{x}$
What is Complexity of Search?

minimum of the function

\[ d \left( x + \frac{n}{x} \right) \quad \frac{dx}{dx} = 0 \]
What is Complexity of Search?

\[
d\left(\frac{x + \frac{n}{x}}{x}\right) = 1 - \frac{n}{x^2} = 0 \quad \Rightarrow \quad x = \sqrt{n}
\]
What is Complexity of Search?

\[ \sqrt{n} = |L_1| + \frac{|L_2|}{|L_1|} = n \]
What is Complexity of Search?

\[ 2\sqrt{n} \]
What is Complexity of Search?

- two sorted lists
  \[ O(2\sqrt{n}) < O(n) \]  
- one sorted list

- two sorted lists

- one sorted list
What is Complexity for 3 Lists?
What is Complexity for 3 Lists?

\[3 \sqrt[3]{n}\]
What is Complexity for $k$ Lists?

for $k$ linked lists: $k \sqrt[3]{n}$
How Many Lists?

minimize \( k \sqrt{k} n \)
What is Complexity for $k$ Lists?

$\log n \log \sqrt{n}$
\[ n = 2^{\log n} \]

\[ \log n^{\sqrt{n}} = n \frac{1}{\log n} = 2^{\frac{\log n}{\log n}} = 2 \]
Complexity for log n Lists

\( O(\log n) \)
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
Complexity of Contains

• Makes zig-zag motion to bottom
• Proportional to height
• $O(\log n)$
Remove 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, remove it
   3. If no down element, reduce size
   4. Move down
Note about Remove

- Only decrement size at bottom level
- Makes zig-zag motion to bottom
- Proportional to height
- $O(\log n)$
Add Skip List

• Add to bottom row (must increment size)
• Move upwards, flipping a coin
• As long as heads, add to next higher list
• At top, if the number of lists < log (size)
  – Flip a coin, if heads make new list
Add Example

Insert the following: 9 14 7 3 20

Coin toss: T H T H H T H T (move up if heads)
Add Example:

Insert : 9 14 7 20

Coin toss: T HT HHH T  (H = move up)
Add Example:

Insert : 9 14 7 20

Coin toss: T HT HHH T (H = move up)
Complexity of Add

• Proportional to height, not to the number of nodes in the list

• $O(\log n)$