CS 261

Sorted Linked List

by

Tim Budd

Sinisa Todorovic
Complexity – Lists and Arrays

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

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

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

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

What about sorted lists?
Sorted Linked List

list

sentinel

val: 2
next:

val: 5
next:

val: 7
next:

val: 10
next: null
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

```c
struct list {
    struct link * Sentinel;
    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;  /* Returns the link RIGHT BEFORE */
}
```

e = 7
void addSortedList (struct list* lst, TYPE e) {
    struct link * current =
        slideRightSortedList(lst->Sentinel, e);
    ...
void addSortedList (struct list* lst, TYPE e) {
    struct link * current = slideRightSortedList(lst->Sentinel, e);
    struct link * newLink = (struct link *) malloc(sizeof(struct link));
    assert (newLink != 0);
    newLink->value = e;

    ....
}

Add Sorted List

<table>
<thead>
<tr>
<th>e = 7</th>
<th>val: 2</th>
<th>val: 5</th>
<th>val: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>next:</td>
<td>next:</td>
<td>next: null</td>
</tr>
</tbody>
</table>
Add Sorted List

```c
void addSortedList (struct list* lst, TYPE e) {

    ...

    newLink->next = current->next;
    /* For doubly linked lists */
    /* newLink->previous = current; */
    /* current->next->previous = newLink; */
    current->next = newLink;
    lst->size++;

}
```

**Diagram:**
- **sentinel**
- **current**
- **newLink**

```plaintext
  val: 2
  next:  

  val: 5
  next: 

  val: 7
  next: 

  val: 10
  next: null
```
Remove Sorted List

```c
void removeSortedList (struct list *lst, TYPE e) {
    struct link * temp;
    struct link * current =
        slideRightSortedList(lst->Sentinel, e);
    if ((current->next != 0) &&
        (EQ(current->next->value, e))){
        ....
    }
}
```

- `e = 7`
void removeSortedList (struct list *lst, TYPE e) {
    ... if ( ... ){
        temp = current->next
        current->next = current->next->next;
        /* For doubly linked lists */
        /* 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 ➔ 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., 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