Further Discussions on Linked Lists

Yufei Tao's Teaching Team

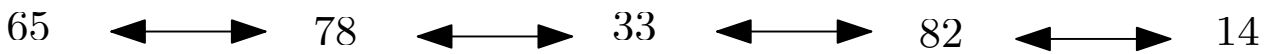
Inserting an element in the middle of a linked list

Review:

A linked list storing a set of integers $\{14, 65, 78, 33, 82\}$:

	b				a				d				e			c			
	78	a	c		65	$\perp$	b		82	c	e		14	d	$\perp$	33	b	d	

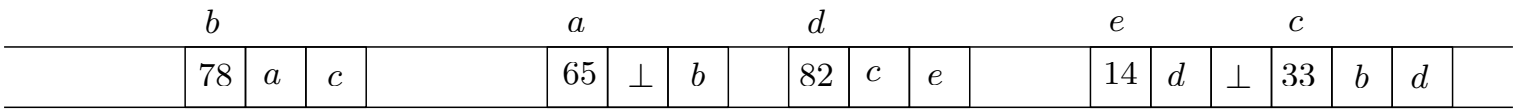
Conceptually, we can think of the sequence $(65, 78, 33, 82, 14)$ in the linked list as:



Remember that a linked list stores a sequence Σ of elements.

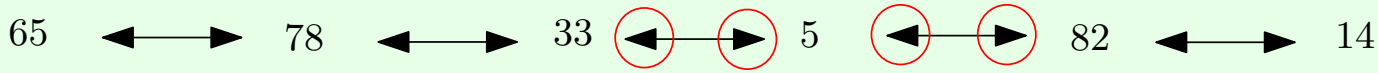
In general, to insert a new element e at the i-th position of Σ, we need $O(i)$ time because we need to find that position by traversing from the head of the linked list.

However, if we already know the address of the element p before e, then the insertion can be done in $O(1)$ time.



Suppose that we want to insert number $e = 5$ after 33, assuming that we already have the address of node $p = 33$ (the address is c). The insertion can be completed as follows:

- Get the element s after p.
- Setting the next pointer of e to the address of s.
- Setting the previous pointer of s to the address of e.
- Setting the next pointer of p to the address of e.
- Setting the previous pointer of e to the address of p.



The pointers modified are circled in red.

An application of the stack

While the usefulness of "queues" is easy to appreciate, it would be more difficult to imagine why we need "stacks". This data structure plays an important role in algorithm design, but we will not see this until we discuss graph algorithms (in particular, depth first search). To satisfy your curiosity, next we will describe a representative application of stacks.

Review:

Consider the following sequence of operations on an empty stack:

- `Push`(35): $S = \{35\}$.
- `Push`(23): $S = \{35, 23\}$.
- `Push`(79): $S = \{35, 23, 79\}$.
- `Pop`: return 79 after removing it from S. Now $S = \{35, 23\}$.
- `Pop`: return 23 after removing it from S. Now $S = \{35\}$.
- `Push`(47): $S = \{35, 47\}$.
- `Pop`: return 47 after removing it from S. Now $S = \{35\}$.

You are given a sentence stored in a sequence of n cells. Each cell contains a word or one of the following pairing characters:

$$“,”, (,), \{, \}, <, >$$

Design an algorithm to determine whether the paring characters have been matched correctly (in the way we are used to in English). The following input is a correct sentence:

I	say	“	I	like	(	red	)	apple	”

And the following input is not a correct sentence:

I	say	“	I	like	(	red	”	)	apple

Your algorithm should finish in $O(n)$ time.

The key idea is to use a stack to manage the opening characters.

Algorithm: Sequentially scan the input sentences.

- At reading a “, (, <, or {, push it into the stack.
- At reading a ”,), >, or }, pop and check whether the symbol removed from top of the stack matches the character just read. If not, report ”incorrect”.
- After reading all the cells, check whether the stack is empty. If so, report “correct”; otherwise, report “incorrect”.

The time complexity is $O(n)$.

We will demonstrate the algorithm using the following two examples: