Arrays and Lists

1 Arrays

Array - A sequence of elements $a_1, a_2 ... a_n$

An array is usually denoted something like:
`A[1],A[2],...,A[n]` or `A[1...n]`

They're usually in consecutive memory cells, but we (usually) don't care where exactly.

All array elements are of the same time, e.g. integer. Can declare as `integer A[1...n]`

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

1.1 Questions

How do we find the ith element, how long does it take?
Go to address and read the data, a quick operation. Not affected by how large the data structure is

What if we want to erase an element?
Once again, easy to find element. However there would be an empty space left, so all following elements would have to be shifted, making it a slow process overall. Affected by size of dataset and position in the dataset.

The size of an array is fixed when we declare it. How big should we make it?
Could make it very big so you don't run out, but this is a waste of allocated memory. In practice make it correct size, then if size needs to be increased, add a pointer to a different bit of memory, but this adds complexity.

2 Linked lists

- A **list** is made up of **nodes**. Each node stores an element (a piece of data) plus a pointer or "**link**" to another node
- The first node is called the **head**
- The last node, called the **tail** points to null
- The nodes may be scattered all over the memory
- Each node points to the next node in the list

2.1 Implementing a list

Assume for a list `L` we have pointers to the first as well as the last node of list:

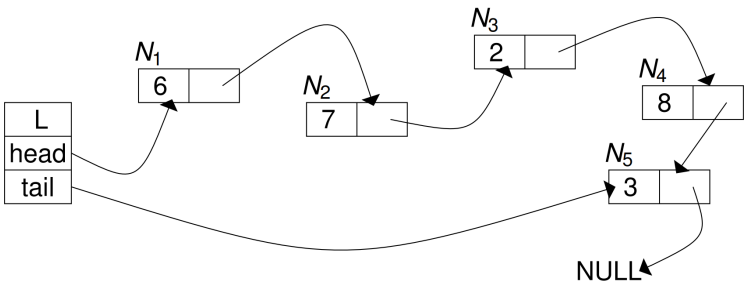
- `L.head`
- `L.tail`
- (and possibly we also have `L.size`) - if the size of the list is often requested, it is easier to set this up at the start, rather than calculating every time

May refer to node `N` using:

- `N.data`, the element
- `N.next`, the link, the next node in the list (may be `NULL`)

`NULL` means "there's nothing there", i.e., last element has no successor

2.2 A linked list



2.3 Finding the ith item

How would we like `L.find(i)` to find the *i*th piece of data in a list? How can we do this? How long will it take?

Input:list L
Output: *i*th item in L

```
#Checks i is in the list
if i>l.size then
    return error
end if

#Stops error you would get if i was 1
if i=1 then
    return l.head.data
end if

#Walk through the list
current_node=l.head
for j = 2 to i do
    current_node=current_node.next
return current_node.data
```

Need to follow all pointers until you come to the correct piece of data, unlike an array.

2.4 Altering the list

How can we alter the list? Anything that inserts or removed must fix all references to predecessors and successors
Deletion of the head
Input: List L
Output: List L with head deleted

```
# If list is empty then return an error
if l.head=NULL then
    return error
end if

# If list only has one element, set the tail as null
if l.head=l.tail then
    l.tail=NULL
end if

# Set the next element as the head
l.head=l.head.next
```

Insertion of v after N
Input: list L, data v, node N
Output: L with v inserted after N

```
node M
M.data=v
# incorrect if N is the last object
if L.tail=N
    L.tail=M

N.next=M
M.next=N.next
```

3 Arrays vs Lists

	Array	Linked List
Data Access	Fast	Slow
Insertion, deletion	Slow	Fast

4 Doubly linked lists

- A node in a **doubly** linked list stores two references:
  - a next link, which points to the next node in the list
  - a prev link, which points to the previous node in the list
- To simplify, we add two **dummy** or sentinel nodes at the ends of the doubly linked list:
  - The header has a valid next reference but a null prev reference
  - The trailer has a valid prev reference but a null next reference
  - These act as the head and tail of the list, contain no data
  - These are useful as they will never be deleted during operations on the list.

A doubly linked list needs to store references to the two sentinel nodes and a size counter keeping track of the number of nodes in the list (not counting sentinels)

An empty list would have the two sentinel nodes pointing to each other

5 Circularly linked list

- A **circularly** linked list has the same kind of nodes as a singly linked list. That is, each node has a next pointer and a reference to an element
- The circularly linked list has no beginning or end. You can think of it as a singly linked list, where the last nodes next pointer, instead of being `NULL`, points back to the first node
- Instead of references to the head and tail, we mark a node of a circularly linked list as the cursor. The cursor is the starting node when we traverse the list

Deletion from a doubly linked list Input: List L, Node N
Output: L with N removed

```
M=N.prev
P=N.next
M.next=P
P.prev=M
N.prev=NULL
N.next=NULL
L.size=L.size-1
```

6 Question about an array

```
Input: n numbers in array A[0],. . .,A[n-1]
Output: ?
    for i=0 to n-2 do
        e = A[i]
        p = i
        for j=i+1 to n-1 do
            if A[j]<e then
                e = A[j]
                p= j
            end if
        end for
        swap A[i] and A[p]
    end for
    return A
```

What is the output and how is obtained. How long does this procedure take (that is, say, how many times do we make the comparison of A[j] and e in the condition of the **if** statement

- First time through the loop, puts the smallest item in the array in A[0]
- Then puts the second smallest item in the array in A[1] etc etc
- This provides a list sorted from smallest to largest
- This is running the sorting algorithm selection sort
- Comparisons (n-1)+(n-2)...
- Complexity is approximately n^2