ECSE-322

23 January 2008
Lecture 9
Multidimensional Arrays
Dynamic Structures

Class Test Monday 28 Jan

Problem Set

40's

A - 65

No crib Sheet

Multidimensional Arrays!

Sparse Multidimensional Arrays

Can we invent a form of vectoring?

For each column, store a value which indicates where, in the linear array, the column begins..

Consider:

11	0	0	0	0	12
0	22	0	15	0	6
0	0	0	5	7	0
15	0	0	2	0	13
14	9	17	0	0	0
0	10	1	31	0	0

1 1 1 2 2 2 3 3 4 4 4 4 5 6 6 6

Array of Non-zeros: 11, 15, 14, 22, 9, 10, 17, 1, 15, 5, 2, 31, 7, 12, 6, 13

Array of row indices: 1, 4, 5, 2, 5, 6, 5, 6, 2, 3, 4, 6, 3, 1, 2, 4

"Vector" Array: 1, 4, 7, 9, 13, 14

What is the overhead cost?

Sparse Multidimensional Arrays

- This form of vectoring does not result in an indexing polynomial…
- An *indexing function* is developed…
  - This function accesses the row and column vectors.
  - On a query about a particular entry, it returns a value or zero..
    - Function S(r,c).. Val := S(r,c)

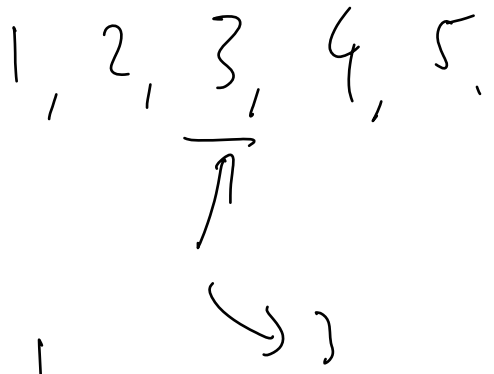
Linear Dynamic Structures

- All the structures considered so far have been *static* - i.e. of fixed size..
  - This allows memory to be pre-allocated and storage and retrieval to be fast.
  - The system is limited because the size of the problem has to be estimated ahead of time..
  - This can lead to a waste of memory space (or an overflow)..

Linear Dynamic Structures

- If size cannot be determined ahead of time (e.g. how big is a file?) then a *dynamic* structure is needed..
  - The counterpart to a vector or linear array is the *linear list*
  - These structures are used to implement the abstract data types *stacks* and *queues*

Stacks



- A dynamic object
  - it grows as more data is received
  - it shrinks as data is removed
  - when a data element is removed from the stack - *it no longer exists in the stack*

    (compare this with an array - when an element is retrieved it stays in the array and a copy is made)

Stacks

- Stacks are:
  - Single ended data structures
  - All operations are performed in one place
  - An example is a stack of cards, or a pile of plates.
  - Data can only be placed on the top of a stack.
  - Data can only be retrieved from the top of a stack.
  - A stack is a *Last In, First Out (LIFO)* structure

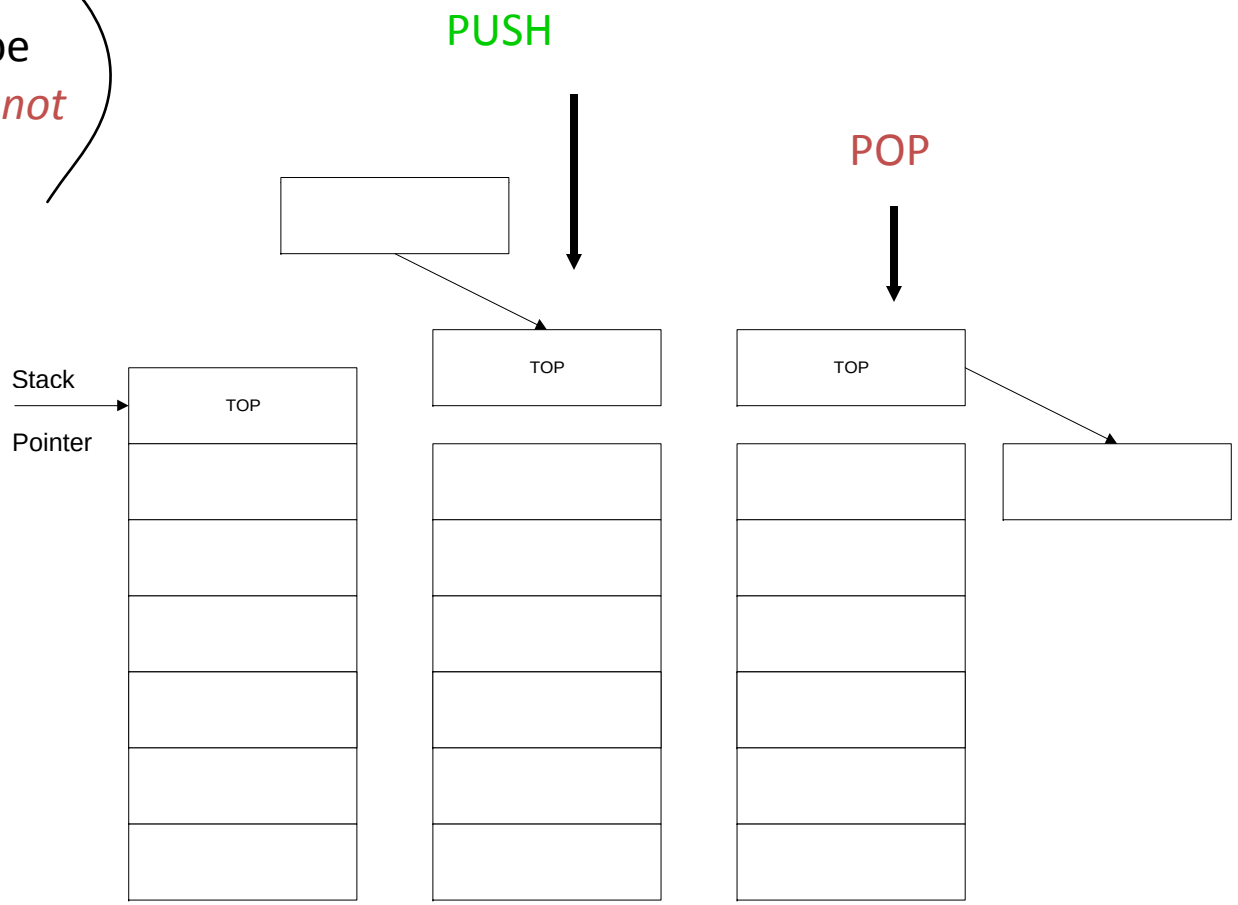
Stacks

- Operations:
  - PUSH - *place an item on the stack*
  - POP - *remove an item from the stack*
    - Each push operation increases the size of a stack
    - Each pop operation decreases the size of a stack.
  - In principle a stack can keep growing for ever so a push can always be executed.
  - In practice, a stack can run out of memory so this state must be signaled.

Stacks

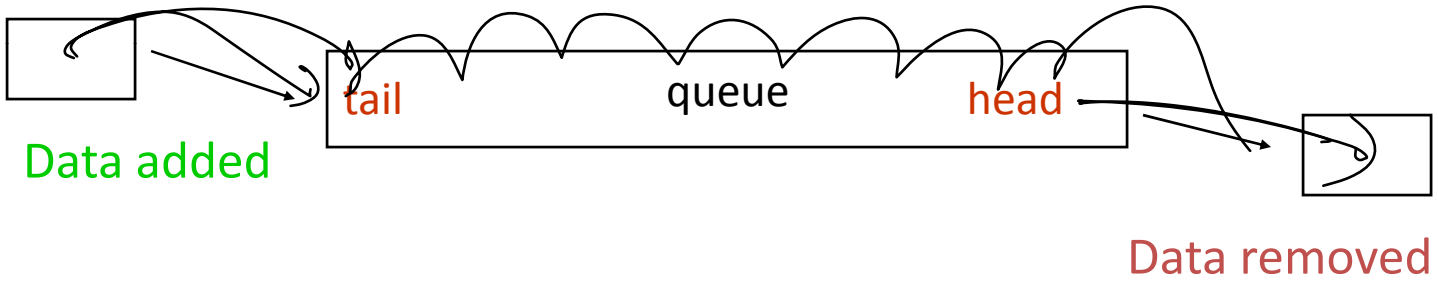
Note - A pop can only be executed if the stack is *not empty*

A stack might be implemented within an array...



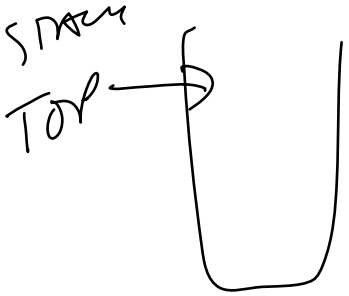
Queues

- Queues are *double-ended*



A First In, First Out (FIFO) structure

Additions at the tail, removals at the head..

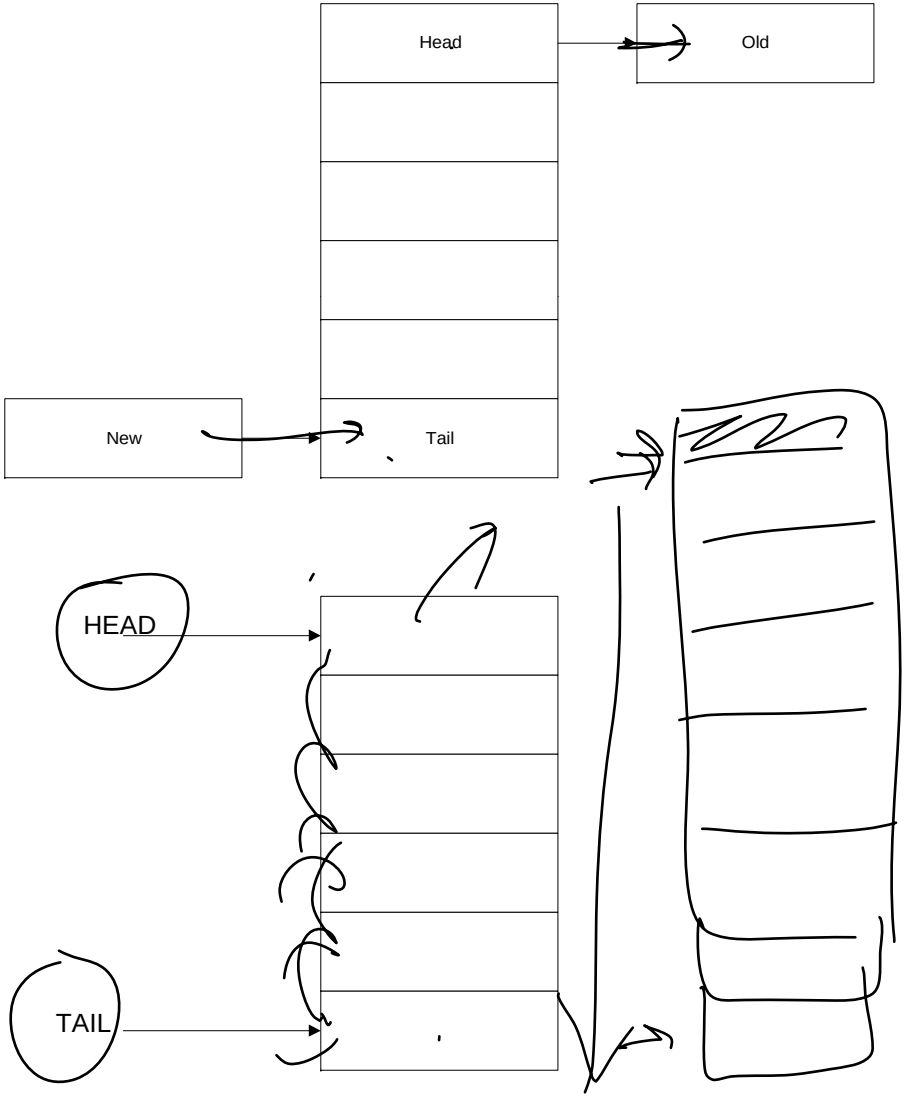


Queues

Note - the double ended nature requires two pointers.

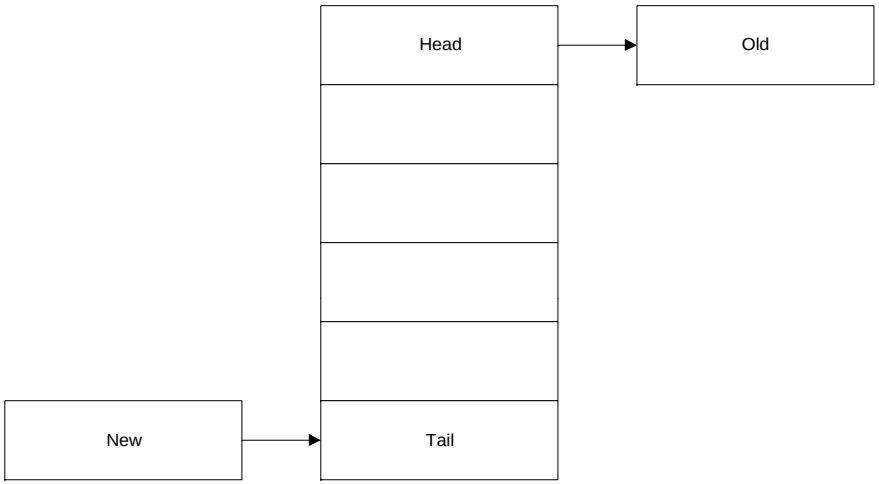
Queues have a problem which does not occur with a stack…

What is it?



Queues

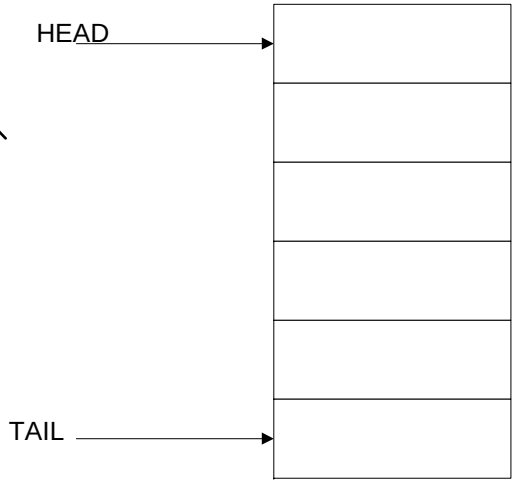
Note - the double ended nature requires two pointers.



Queues have a problem which does not occur with a stack…

What is it?

Adding and removing items means that the queue migrates through memory!



Queues

If an array of size N is allocated to the queue, then, after N add-and-remove operation pairs, there is no space to add items, even though the length of the queue may be zero!

How can this problem be solved?

1. All items can be moved towards the head whenever an item is removed.

Why is this not a good idea?

Queues

If an array of size N is allocated to the queue, then, after N add-and-remove operation pairs, there is no space to add items, even though the length of the queue may be zero!

How can this problem be solved?

1. All items can be moved towards the head whenever an item is removed.

   Why is this not a good idea?

Because it requires O(N) work every time something is removed

Queues

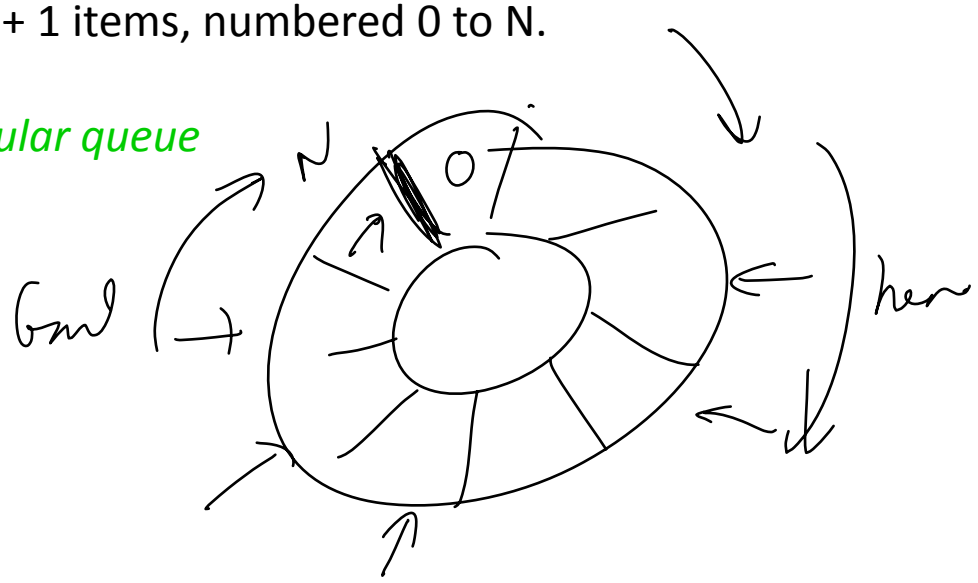
2. Use *wrap-around indexing*

In Pascal this can be achieved by the use of the mod operator...

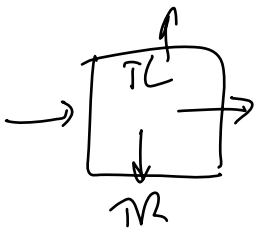
```
cind := ind mod (N+1);
```

where the array contains N+ 1 items, numbered 0 to N.

Such a structure is often called a *circular queue*

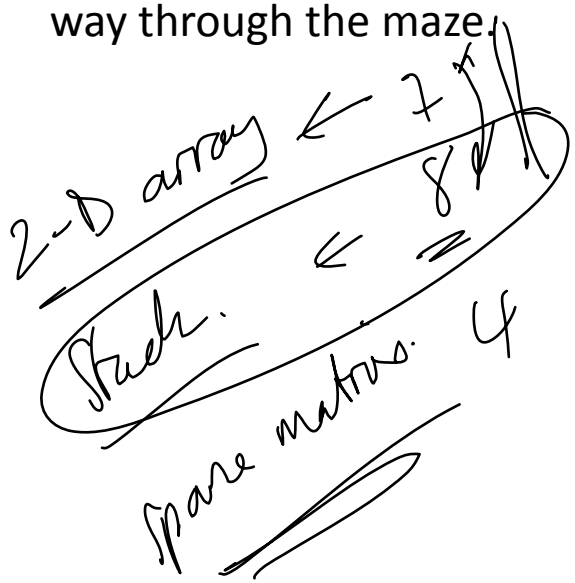


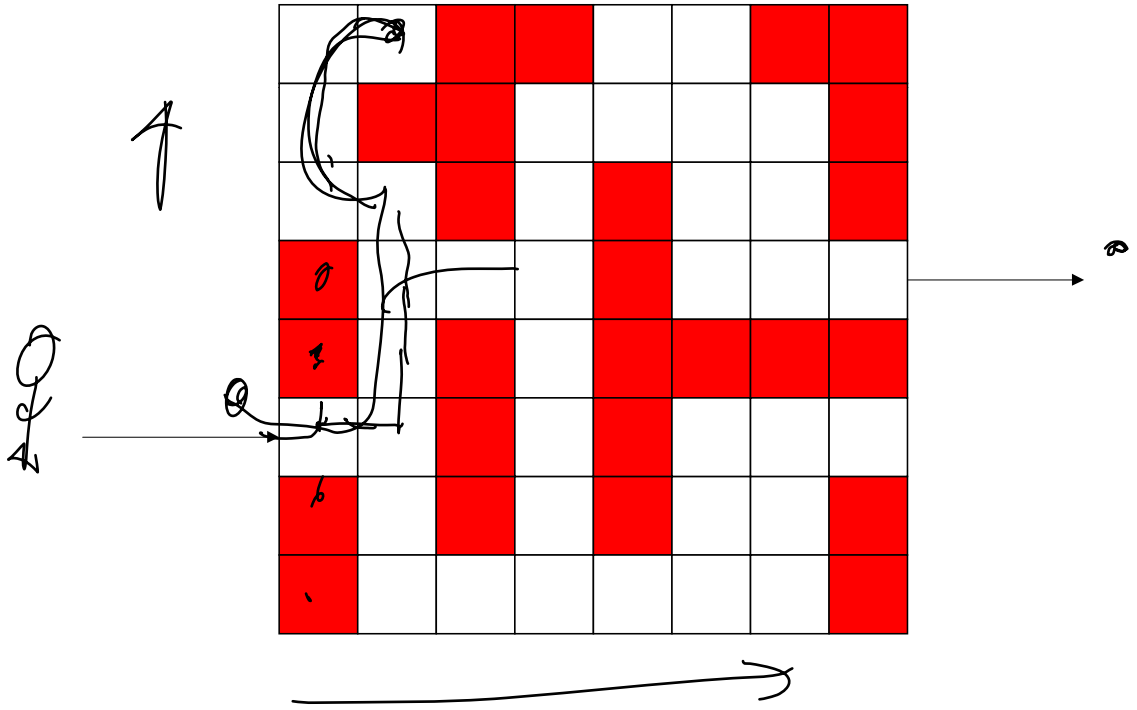
An Example - The Maze



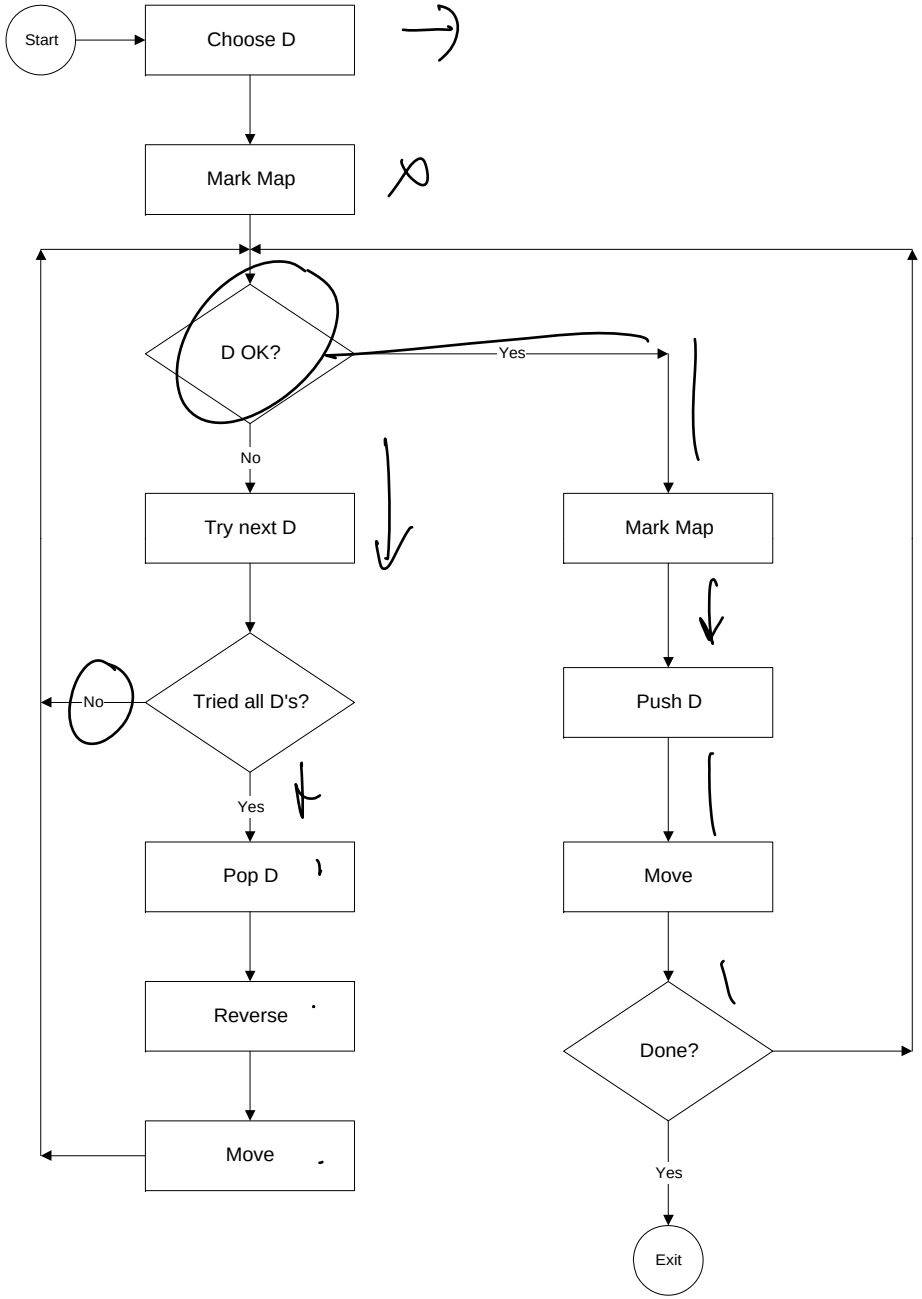
Problem:

Design a data structure to help solve the problem of finding a way through the maze.





Use a stack to keep track of the path through the maze.



Lists

- Stacks and Queues imply an ordering to the data..
  - Data can only be accessed through the ends
  - They are *linear structures*
  - *Accesses are destructive*
- A List is an alternate linear structure which allows access to any item in the structure

Lists

- Operations:

Given an ordered sequence of items – q_1, q_2, q_3,...qn-1, q_n - the operations possible on lists are:

Insert (X,n,Q) store item X as the nth item of list Q.

Retrieve (X,n,Q) copies the n^{th} item of list Q into X.

Delete (n,Q) deletes the n^{th} item of the list Q

Lists are dynamic structures i.e. the length changes with time.

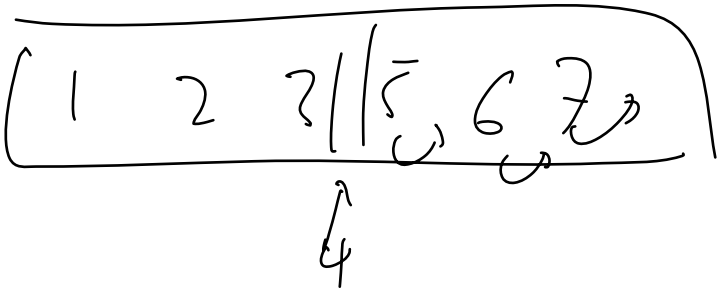
Lists

One more operation is optional but convenient…

Length (n,Q)

determines the length n of the list Q.

The implementation?



An array is possible (easy access) - but

The insertion of an element requires moving the existing data to create a hole..

this operation is O(N).

Ordered Lists

- A list ordered on some **key** or **criterion**, e.g. the entry in a telephone directory is alphabetic..
- The *key* may be implicit, as in the alphabetic ordering..(*keys* were described earlier in the discussion on abstract data types).
  - An ordered list may be implemented as an array -.
  - Because the data is ordered, binary searches may be used.
- However, insertions and deletions can be slow..

Hence, *hashed storage is often used*

Priority Queues

- Access data on a *priority* basis
  - A *Most Important First Out* structure
  - Requirements:
    - Data is retrieved based on a concept of priority
  - Examples:
    - A conventional queue is a special case of the priority queue - priority is based on insertion time.