ECSE-322

25-January-2008

Lecture 10

Linked lists and Trees

CLASS TEST MONDAY

11:35 – 12:25 (30)

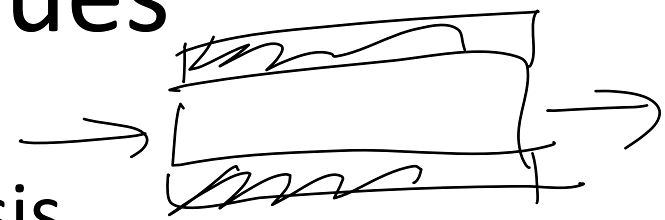
TR 1100 → Aaaa – Fzzz

TR 0100 → Gaaa – Zzzz

PROBLEM SETS 1 TO 4. 40's each 6 marks

Wednesday 31 Jan – No lecture

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.

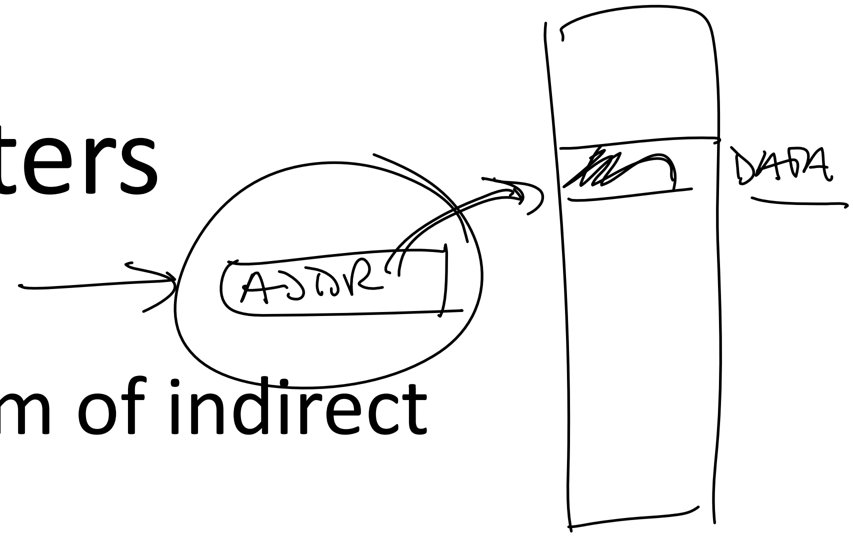
Priority Queues

- How can such a data type be built?
  - Simplest approach:
    - Use 2 arrays

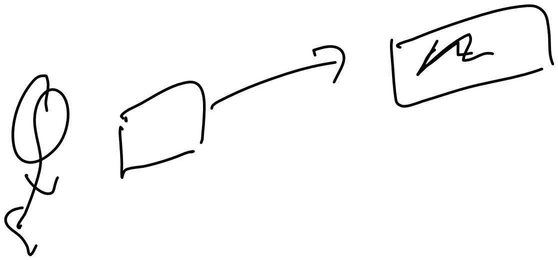
      one for the data

      one for the priority values
    - The priority value then becomes one of the keys for accessing the data.
  - This is somewhat inflexible -- we need some "better" structures for implementation...

Pointers

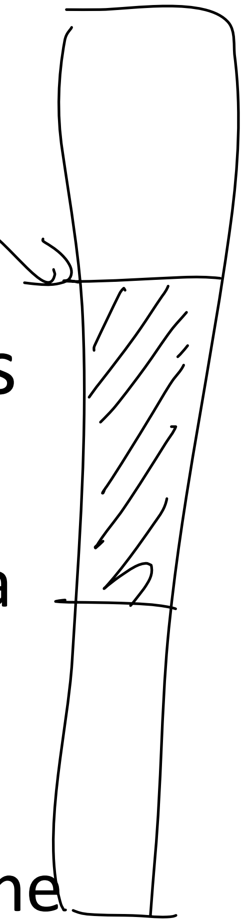


- A **pointer** allows a form of indirect addressing.
  - It holds the **address** of the **data**, not the data itself.
  - We have already seen a hardware implementation in the *Address Register* on an interface and the *Address lines* on a bus (and in one of the frame buffer architectures)
  - A kind of *key*..



Pointers

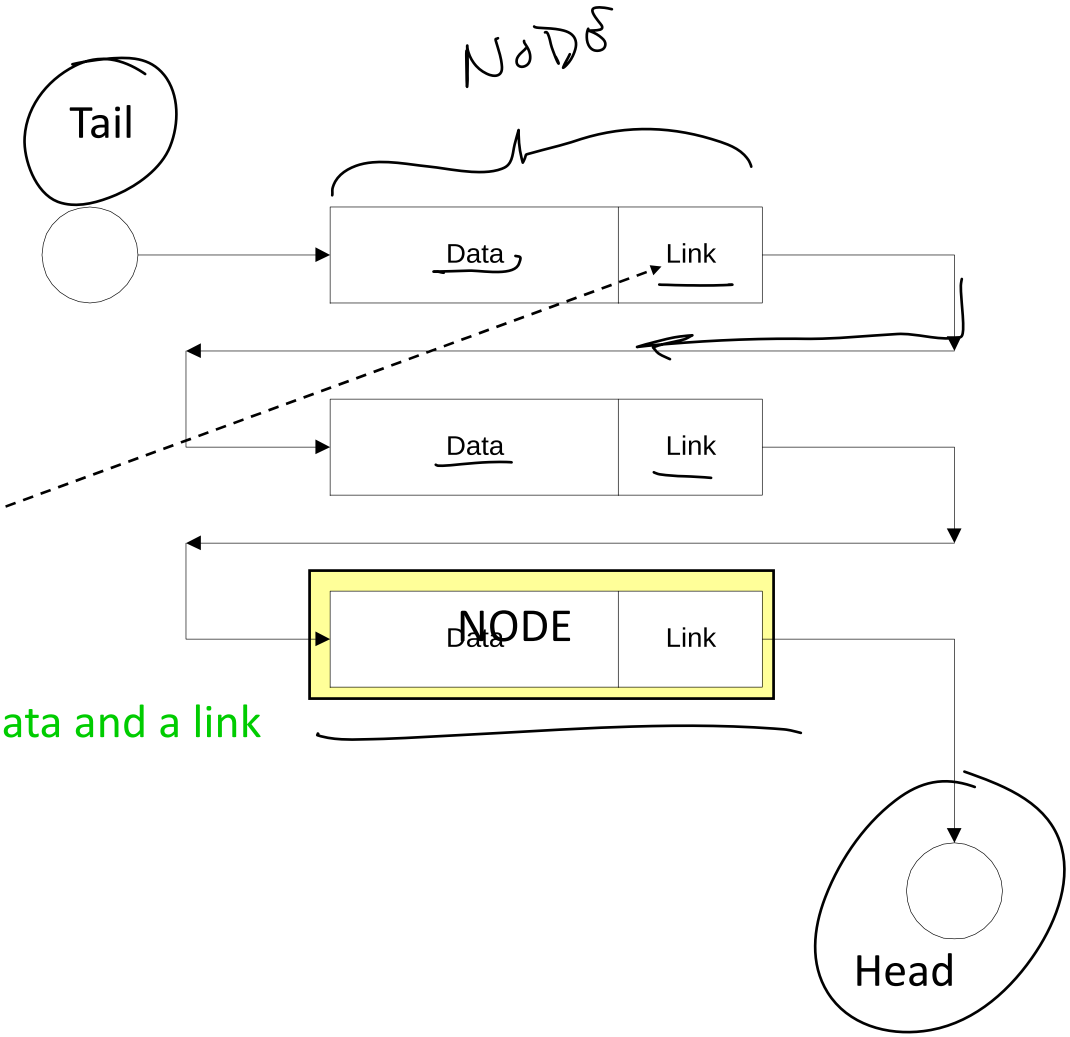
PTR



- Consider the construction of a list of items which has a dynamic structure..
  - ~~Allocate a block of memory to house the data~~
  - Set up an initial pointer to point to the first empty memory location
  - Structure the memory as pairs of locations -the first to hold the data, the second to contain a pointer to the next available location.
- This structure is known as a *linked list*.

Linked Lists

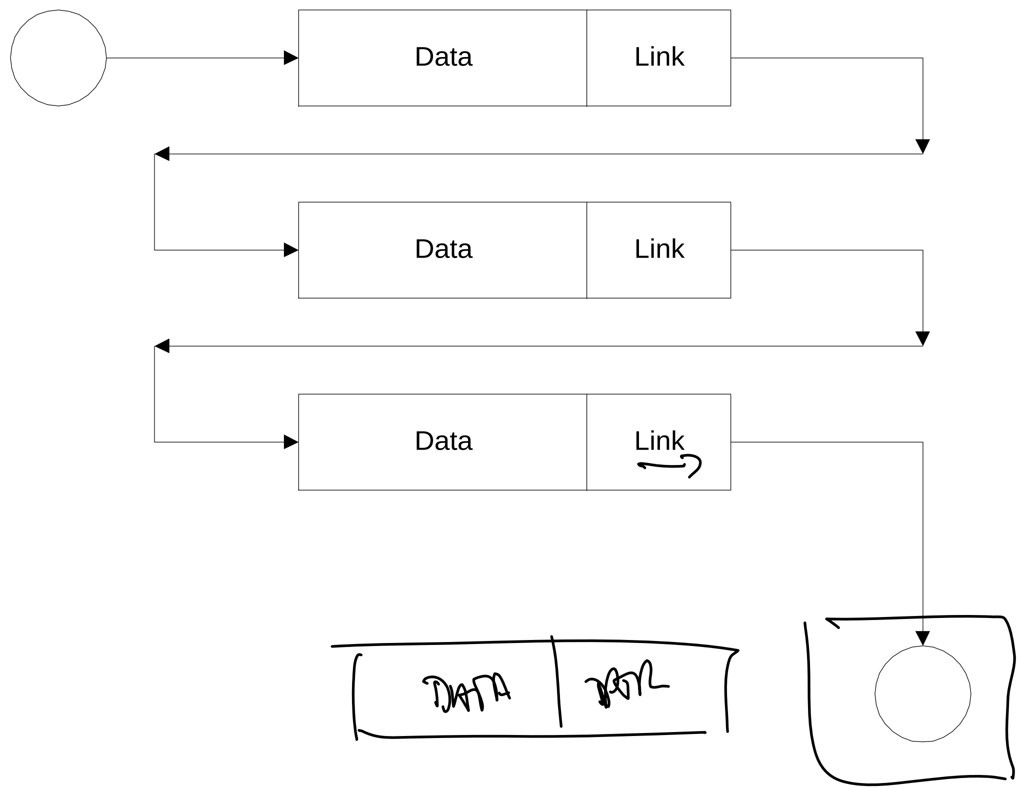
In a linked list, every data item "knows" who is in front..



Each data item has a link to the next data item..

The combination of a piece of data and a link is known as a node.

Linked Lists

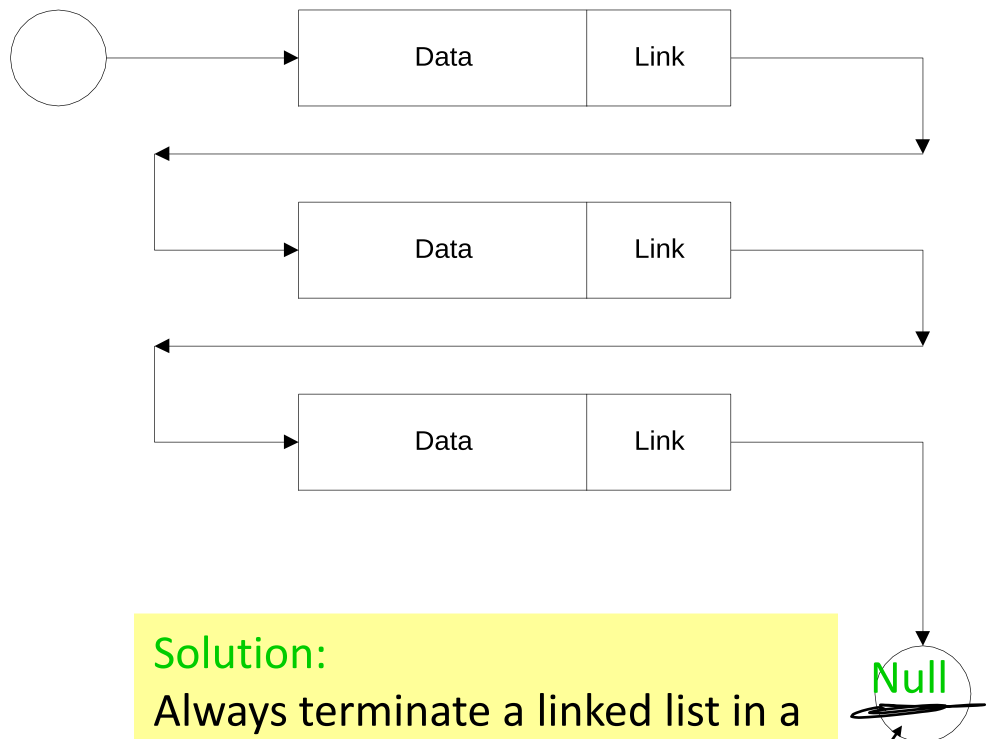


Problem:

- In memory, every location contains a bit pattern.
- Hence every link automatically contains a valid address..

How can the end of a list be found?

Linked Lists



Problem:
•In memory, every location contains a bit pattern.
•Hence every link automatically contains a valid address..
How can the end of a list be found?

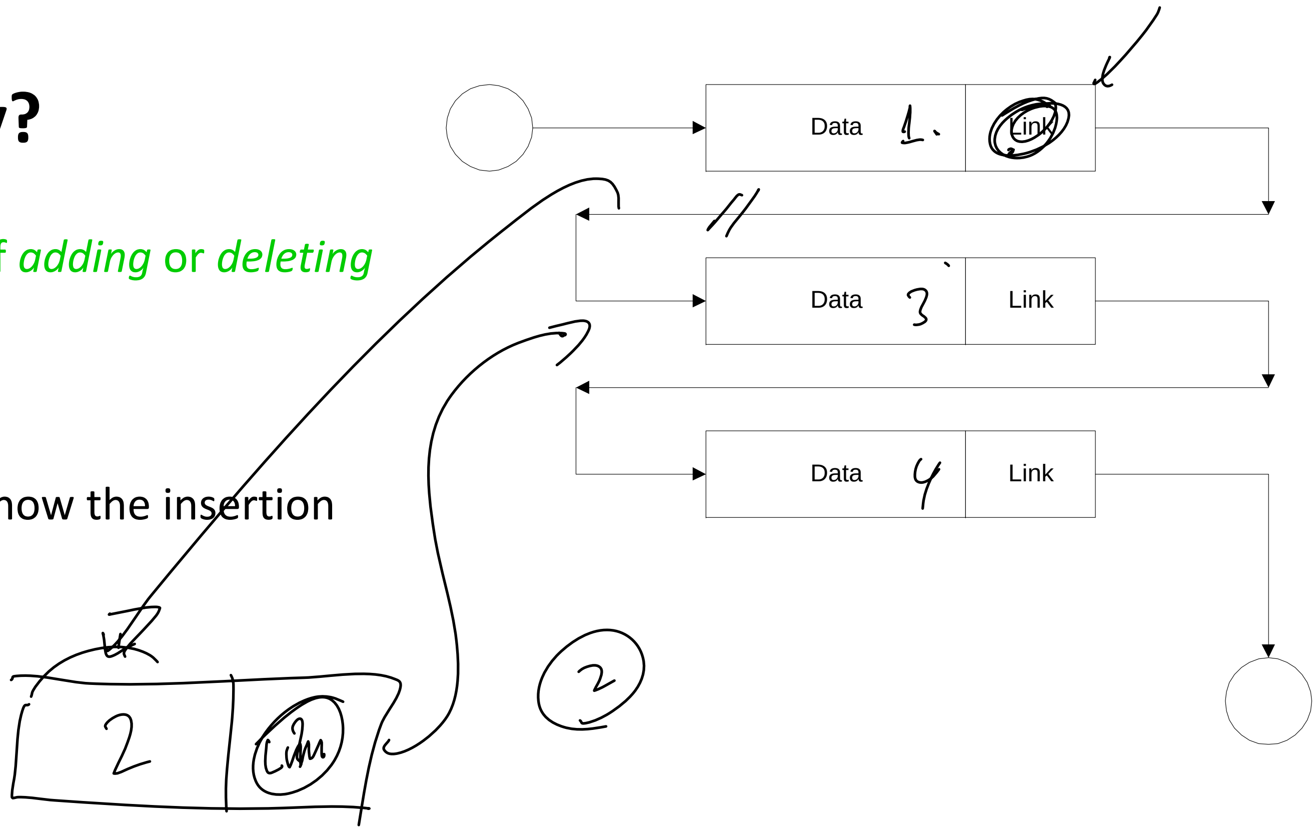
Solution:
Always terminate a linked list in a known value = *no node at all*.
The ***null*** or ***nil*** link value

Linked Lists

Complexity?

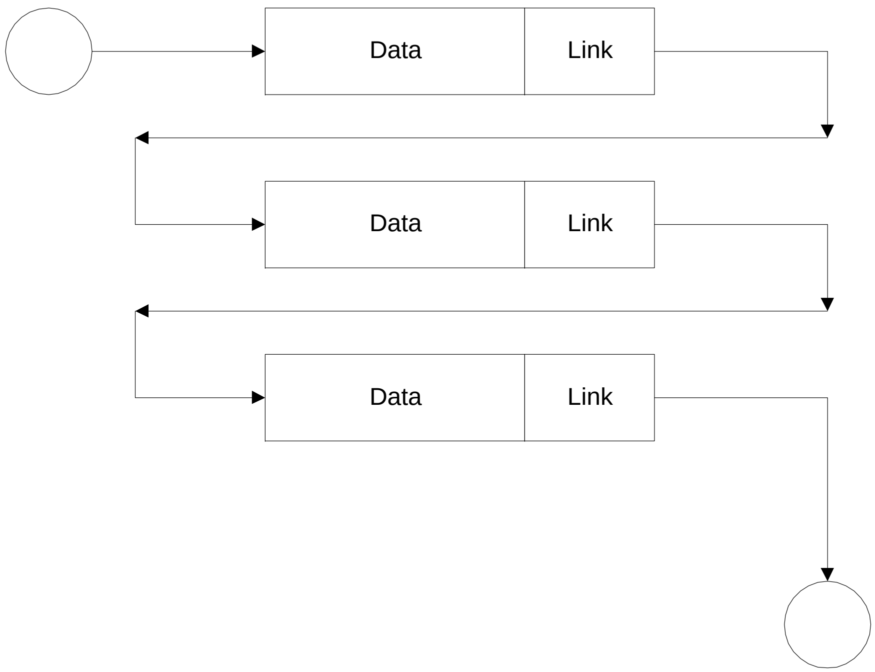
What is the cost of *adding* or *deleting* a data item?

Assume that we know the insertion point.



Linked Lists

Complexity?



What is the cost of *adding* or *deleting* a data item?

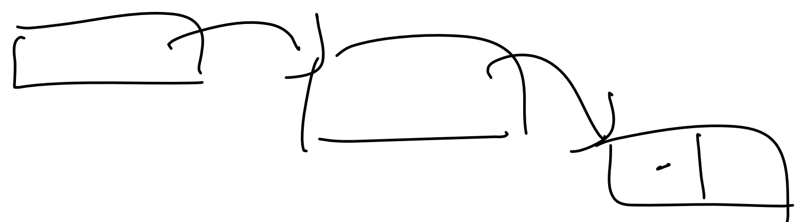
Assume that we know the insertion point.

O(1)

independent of list length!

Linked Lists

- *List Insertion and Deletion*
  - Find where in the list the new item belongs
  - Break the link between the two old members and insert the new node.
  - *Finding the correct place in the list requires a traversal*
- What can go wrong?



Linked Lists

- There is no existing list
- New node fits at the front of the list so no comparison with the preceding node is possible...
- New node fits at the tail of the list so no comparison with the following node is possible..

• What is the complexity?

O(N)

Operations on Lists

- The **Traversal**:
  - in a *traversal*, every element of a structure is visited once.

Start point of list

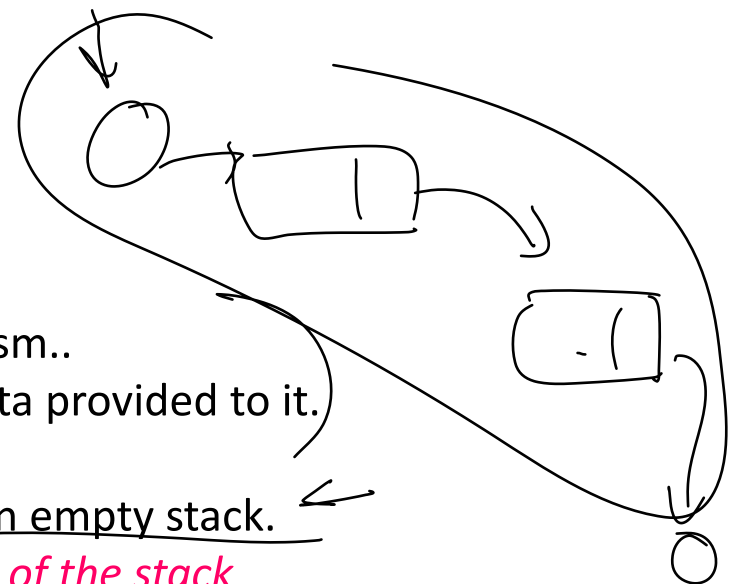
Get data item

Get next link

```
Procedure showlist(list: pointer); var temp: pointer;
begin
        temp := list;
        write (`content: `);
        while temp <> nil do begin
                write(temp^.item, ` `);
                temp := temp^.link;
                end;
        writeln (`END`);
    end;
```

Operations on Lists

- *Inverting a Linked List*



How can a list be traversed backwards?

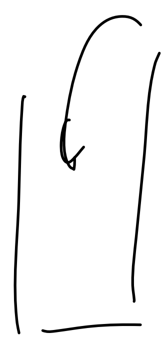
Use the fact that a stack is a LIFO storage mechanism..
Hence a stack naturally inverts the sequence of data provided to it.

1. Take all the items in a list and push them onto an empty stack.

   The head of the list is now at the bottom of the stack

2. Pop all the items off the stack and link them into a new list as they come off the stack.

This is expensive in memory but will work.

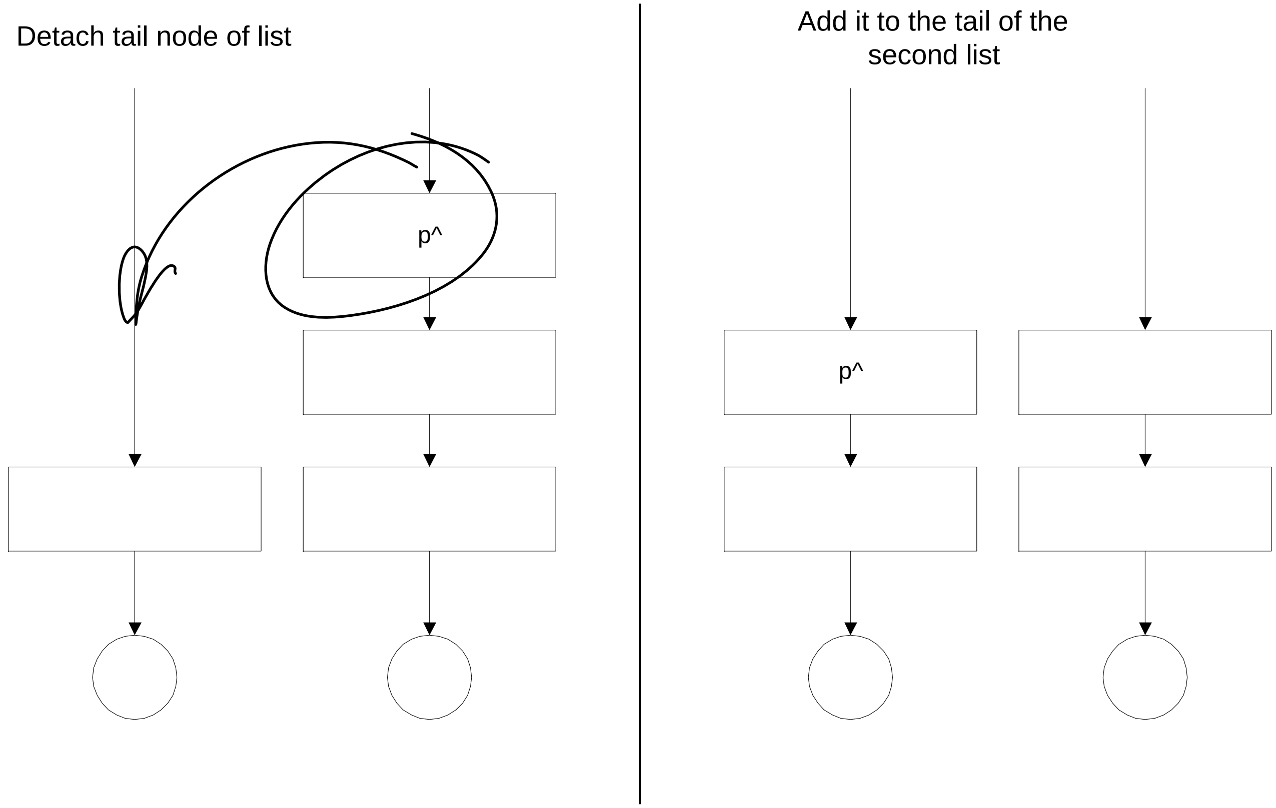


Operations on Lists

- Alternately
  - Recognize that the list is already a sort of stack…
- Process is then:

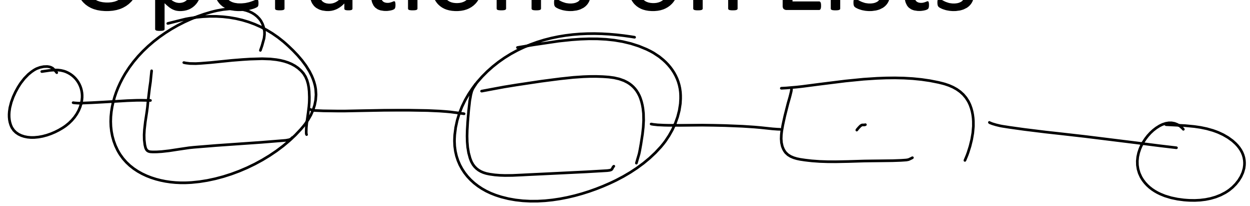
  1. Detach the tail node of the list.
  2. Link it to the tail of a second list
  3. repeat from 1 until the original list is empty

Operations on Lists

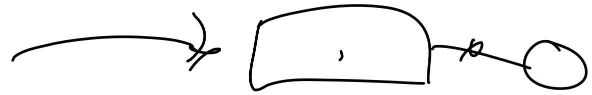


Complexity? O(N)

Operations on Lists



- *Insertion Sorting*



  – An unordered list is sorted by taking items out of an existing list one by one and linking them into a new list.

Complexity?

On average, the position of the k^{th} item is found in $k/2$ comparisons.
So, for N items, the complexity is $O(N^2)$

Multiply Linked Lists

- What happens if a node consists of data plus several links?

data	link 1	link 2	link 3

e.g. a node might have a *forward* and a *backward* pointer - we can now *traverse in either direction!*

Problem:

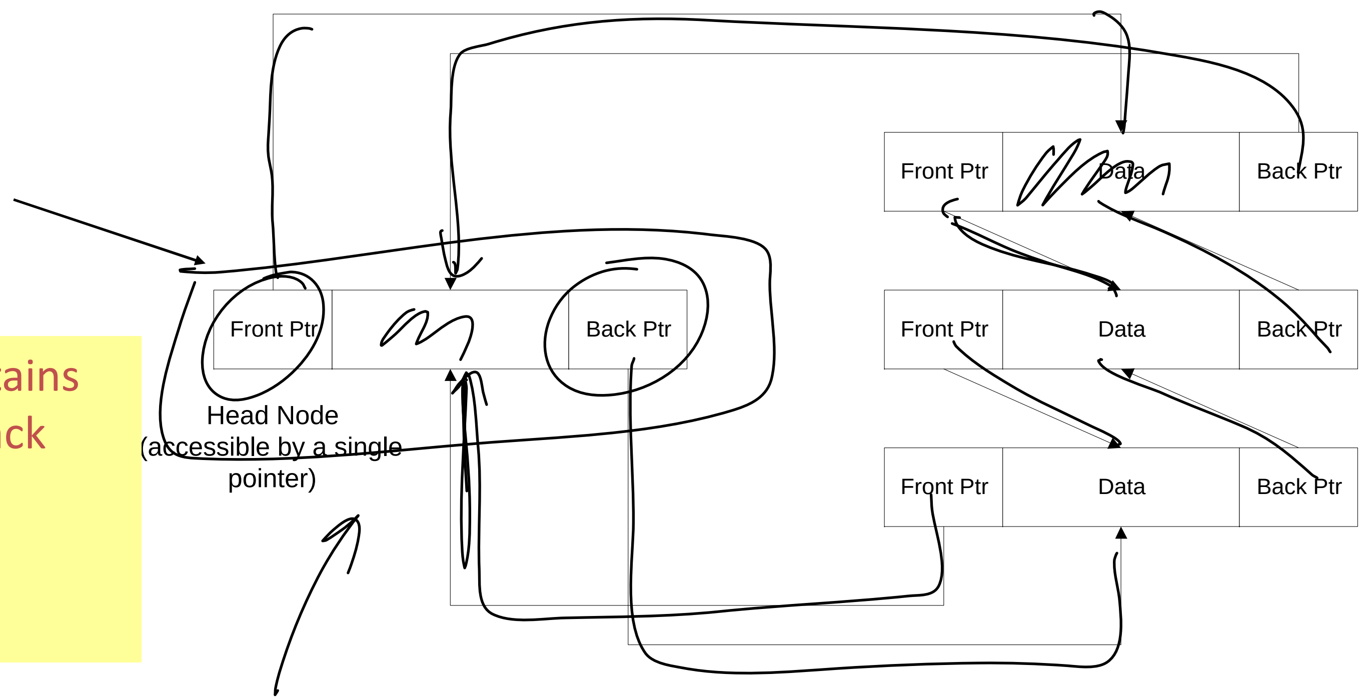
The list now has two entry points - one at either end of the data. The start of a list should be unique (i.e. only one pointer)

Multiply Linked Lists

Solution:

Add an extra node

The Head Node



The head node contains links to front and back *but no data*

Threaded Lists

- A linked list with more than one chain or *thread* of *independent* links.
- The *threads* are the strings of pointers that link the same nodes, but in different sequences.
- Each node in such a list has *multiple entry and exit points*.

Threaded Lists

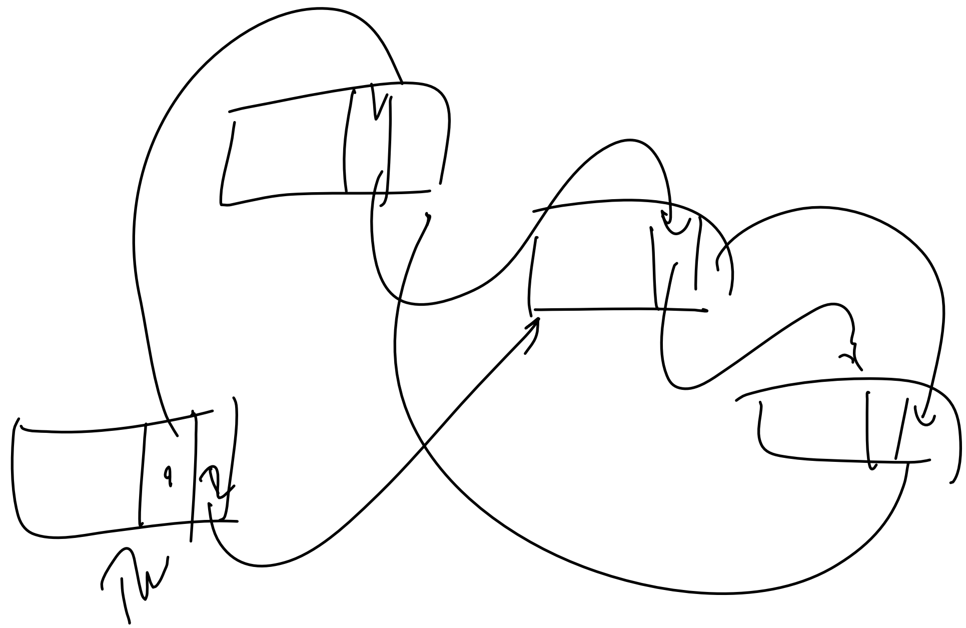
- *e.g.,*
  - A class listing may be in alphabetical order according to last names, or it may be in alphanumeric order according to student ID numbers.
  - A list of numbers may be linked in ascending or descending order.

Threaded Lists

Example:

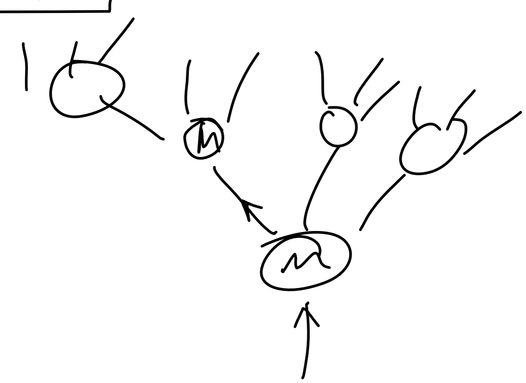
A sparse matrix may be stored as a threaded list, doubly linked by rows and columns:

```
struct element {
float data;
int row;
int col;
struct element *row_ptr;
struct element *col_ptr;
        }
```



Trees

- A hierarchical organization of nodes via linking pointers.
- It resembles a threaded or multiply linked list in that every node carries pointers to several other nodes.
- Each node has *only one entry point*.

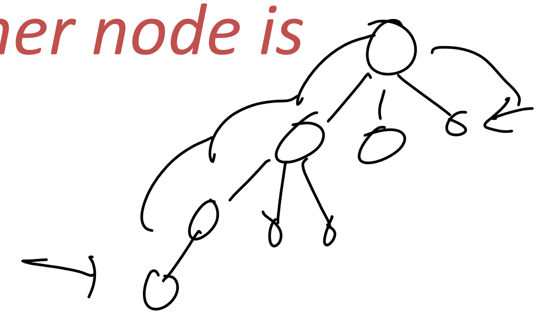


Trees

- Examples of trees:
  - Structure of files and directories in an operating system.
  - Family Trees.
  - Class-based organization of objects in a database

Trees

- Properties of a tree:
  - The data element at each node is of the same type.
  - Each node can be reached via a pointer from *one* parent node.
  - Each node can have several child nodes.
  - *The path from one node to any other node is unique*.

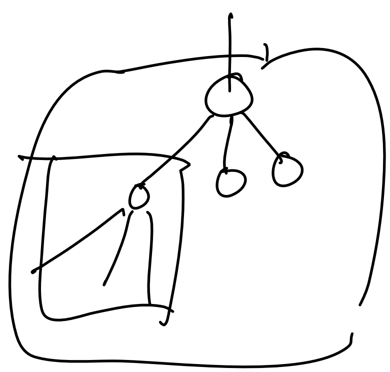


Trees

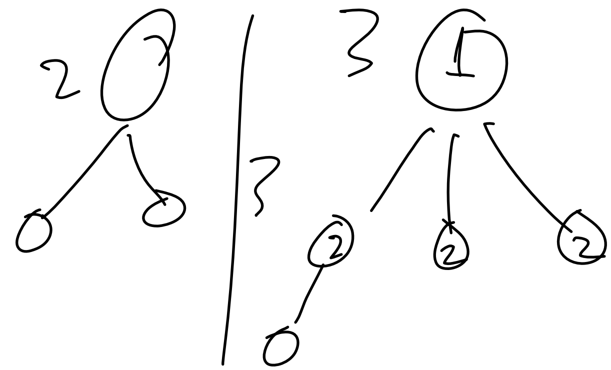
- Definitions:
  - *Singly-linked* trees have links that are uni-directional (typically from the parent node to its child nodes).
  - *Doubly-linked* trees have links that are bi-directional.

Trees

- Definitions (cont'd):
  - The *root* node has no parents.
  - The *leaf or terminal nodes* have no children.
    - In a singly-linked tree, no paths can be drawn from a leaf node to any other nodes in the tree.
  - Any node is also the root node of a *sub-tree*.

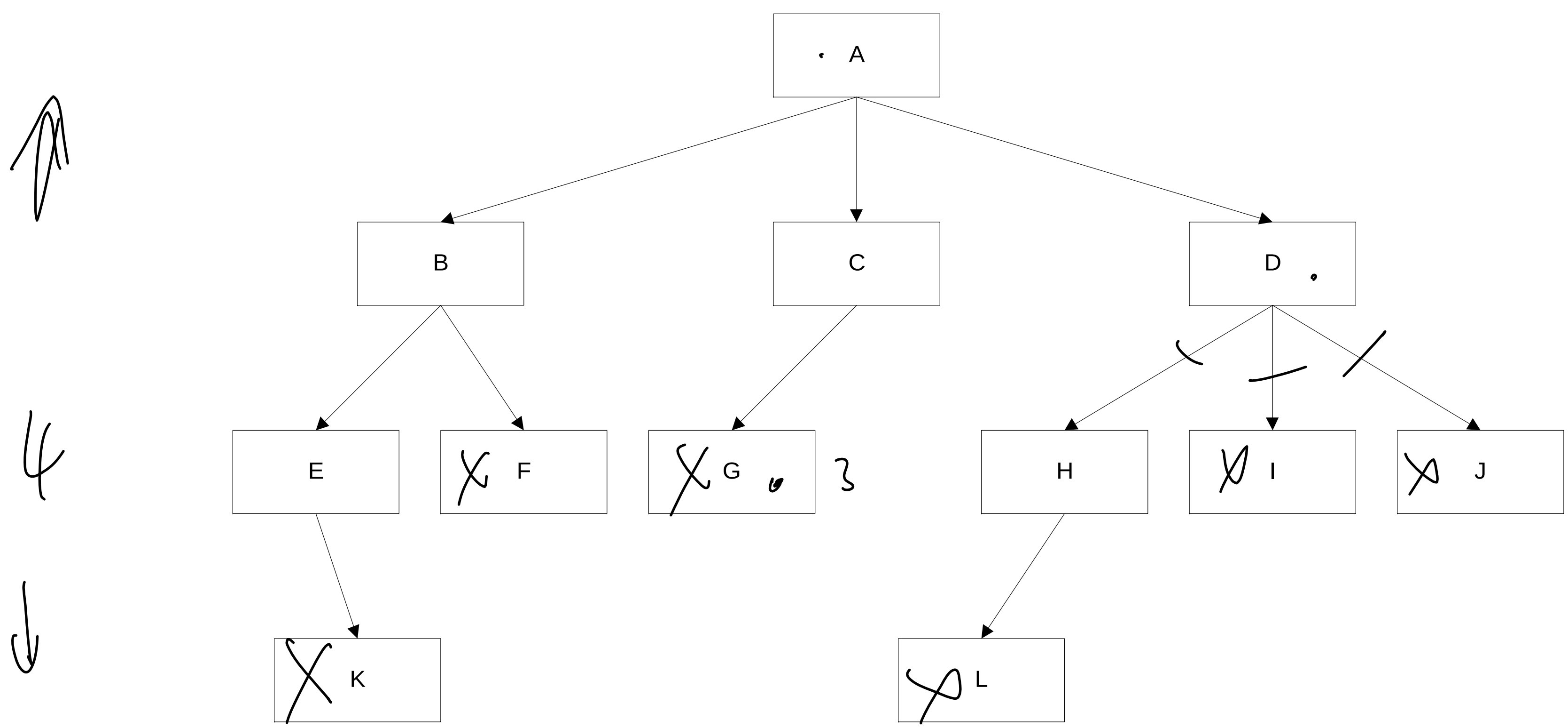


Trees



- Definitions (cont'd):
  - The *degree* of a node is the number of child nodes attached to it.
  - The *level* of a node is the number of nodes traversed in reaching that node from the root, *e.g.,* the root node is at level 1.
  - The *height* of a tree is the total number of levels.

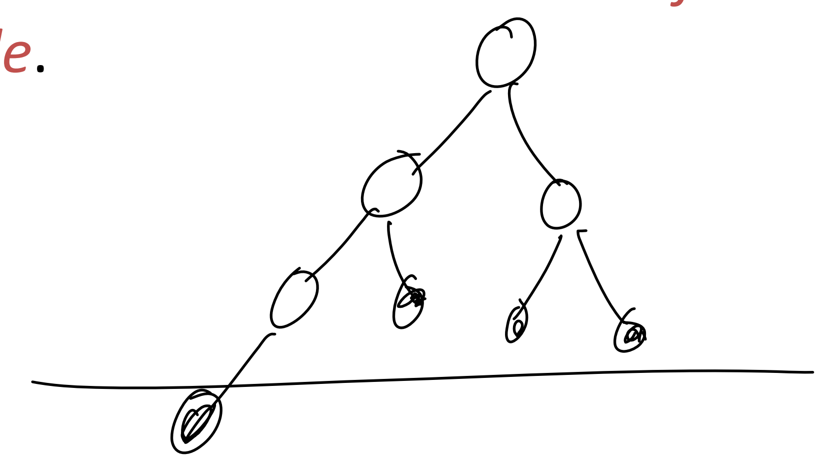
Trees



Number of leaves:	6
Degree of node D:	3
Level of node G:	3
Height of tree:	4

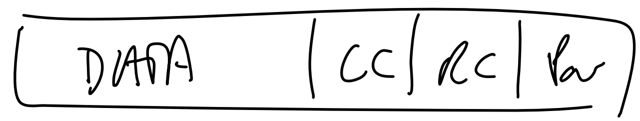
Binary Trees

- Definitions:
  - Every node of the tree is of *degree 2 or less*.
  - In a *complete* binary tree:
    - *Every level, except the last, is fully populated with nodes*.
    - *The nodes in the last level are as far to the left as possible*.



Binary Trees

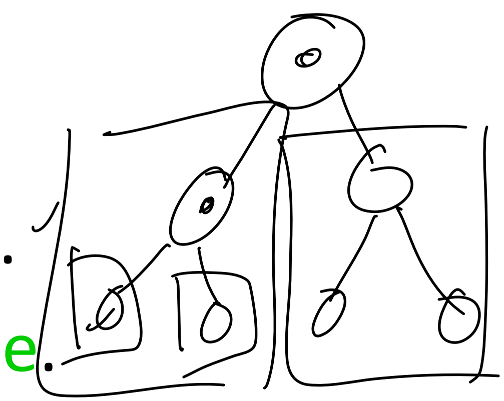
- In conventional *linked storage* a binary tree consists of:
  - A data item.
  - A left child (successor) pointer.
  - A right child (successor) pointer.
  - A parent node pointer (if it is doubly-linked).



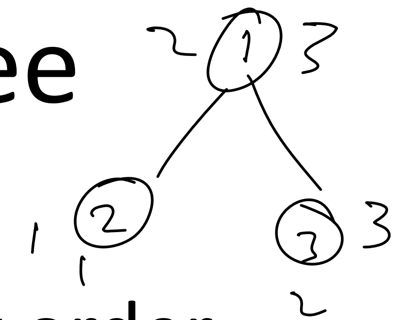
- If a node has *no left (or right) child*, the corresponding pointer is *NULL*.

Traversing a Tree

- The *traversal* of a data structure amounts to visiting each node exactly once in a prescribed order.
- For binary trees, traversal is described recursive terms, *e.g.,:*
  - Visit the root node.
  - Visit *all members* of the left subtree.
  - Visit *all members* of the right subtree.



Traversing a Binary Tree



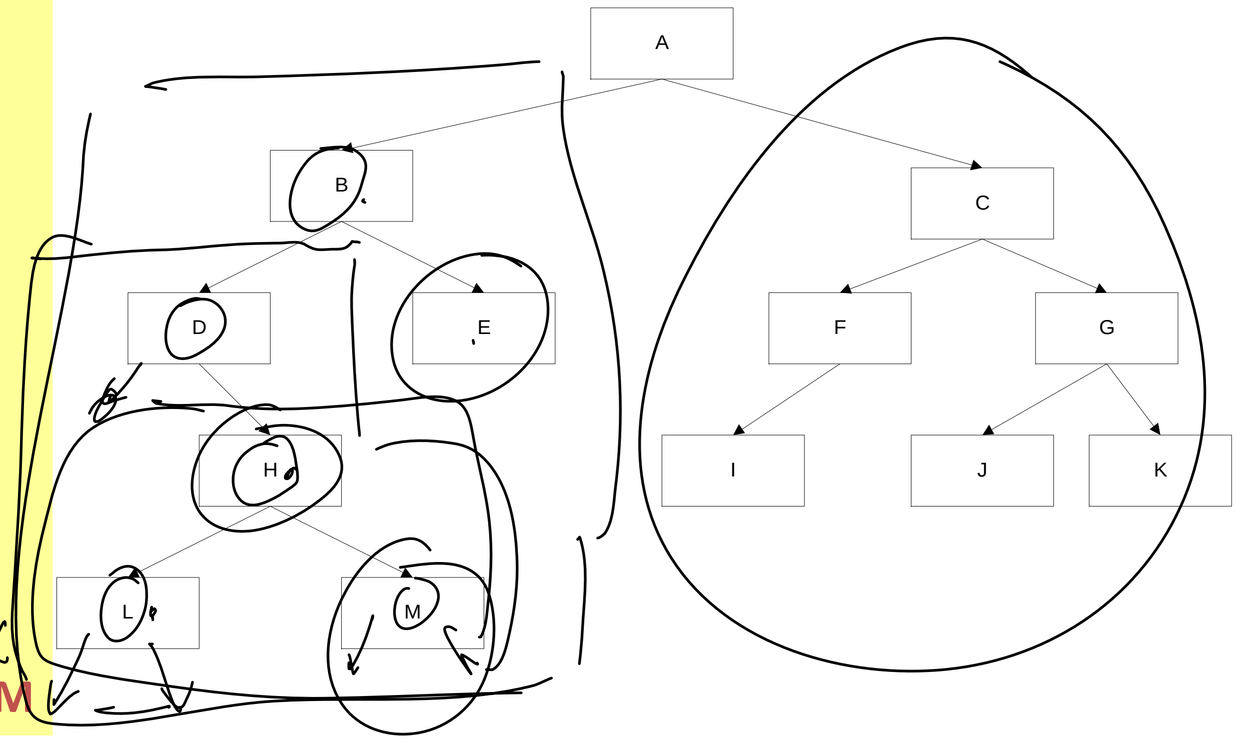
- Three types of traversal (defined by order in which the root is visited with respect to the other nodes):
  - *Preorder Traversal*: Traversals are carried out in the order root, left-subtree, right-subtree.
  - *Inorder Traversal*: Traversals are carried out in the order left-subtree, root, right-subtree.
  - *Postorder Traversal*: Traversals are carried out in the order left-subtree, right-subtree, root.

Traversing a Binary Tree

Example - a Postorder Traversal:

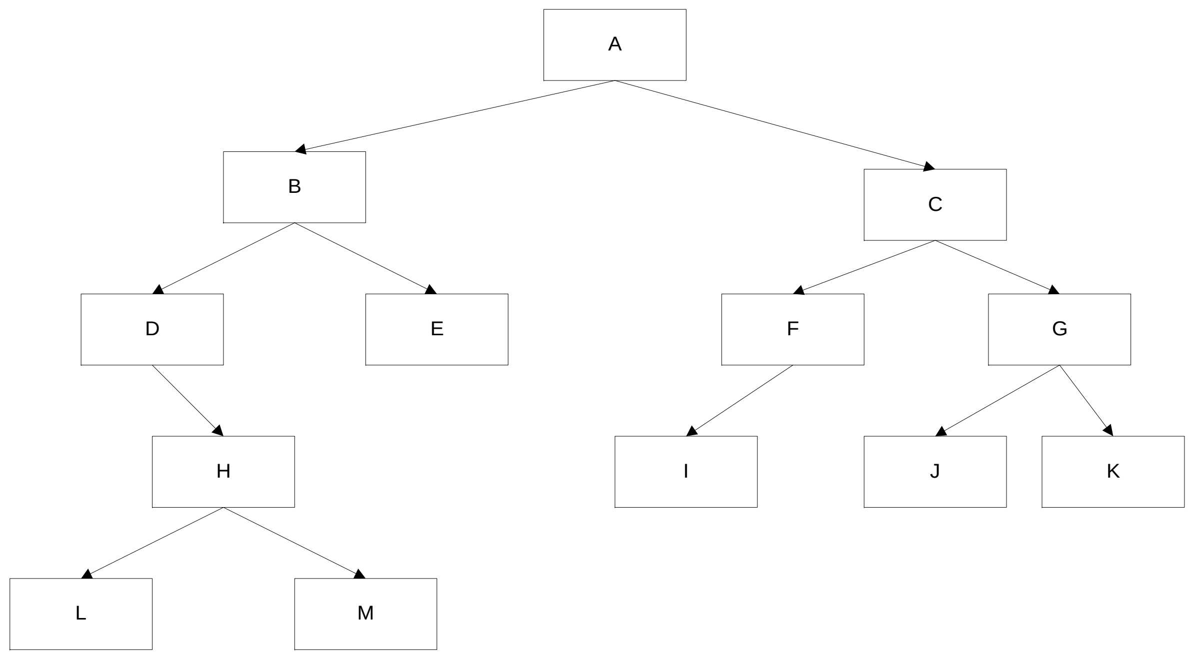
Process - *Left sub, Right sub, Root*

Applying recursively:
Left = B
Left =D
Left – there is none
Right = H
Left = L
Left – there is none
Right – there is none
First item extracted = root = **L**
Recurse back..
Right = M
Left – there is none
Right – there is none
Second item extracted = root = **M**



Traversing a Binary Tree

This recursive algorithm is repeated to produce:
{L M H D E B I F J K G C A}



Applying inorder recursion gives:
{ D L H M B E A I F C J G K}

Binary Tree Insertion and Sorting

- An example Sorting Rule
  - "If an item is smaller than an existing item place it to the left of the existing item"
- Data Insertion:
  - Consider the data elements to be inserted in random order.
  - Compare each element with the root node of a tree, and apply the sorting rule to determine whether it should be placed in the left-subtree or the right-subtree.

Binary Tree Insertion and Sorting

- Data Insertion (cont'd):
  - If that sub-tree exists, repeat the procedure recursively in that sub-tree.
  - If that sub-tree does not exist, insert the data element as the root node of that sub-tree, with *NULL* successor pointers.

Binary Tree Insertion and Sorting

Ascii

- An example:
  - Insert the sequence {K B P A Q E Z I T} in a binary tree that is initially empty, using the following sorting rule:

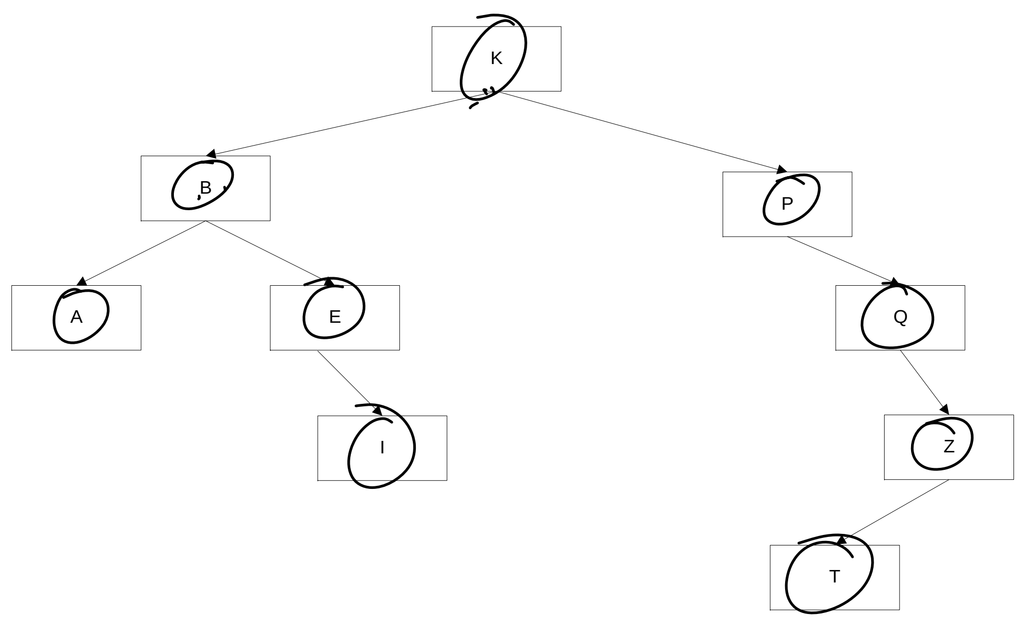
"If the data element to be inserted is before the element at the root node (alphabetically), place it in the left-subtree, otherwise place it in the right-subtree."

Binary Tree Insertion and Sorting

Data: {K B P A Q E Z I T}

Sorting Rule:

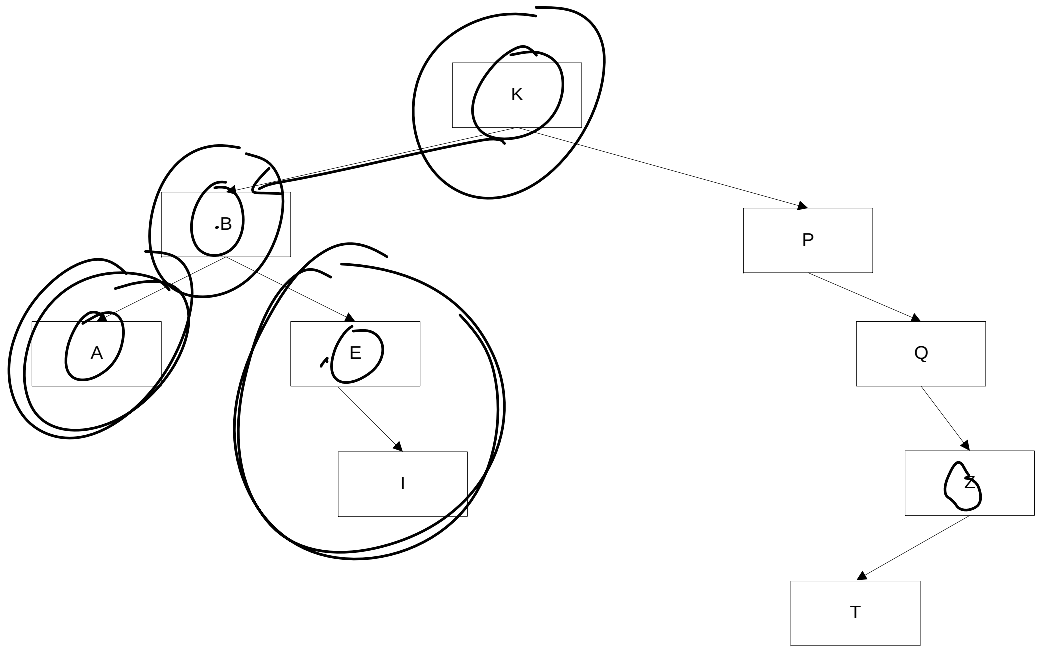
"If the data element to be inserted is before the element at the root node (alphabetically), place it in the left-subtree, otherwise place it in the right-subtree."



Insertion cost? O(N)

Binary Tree Data Retrieval

Retrieve data using an inorder traversal..



Rule:

Left-subtree,
Root,
Right-subtree

Data retrieved:

A, B, E, I, K, P, Q, T, Z

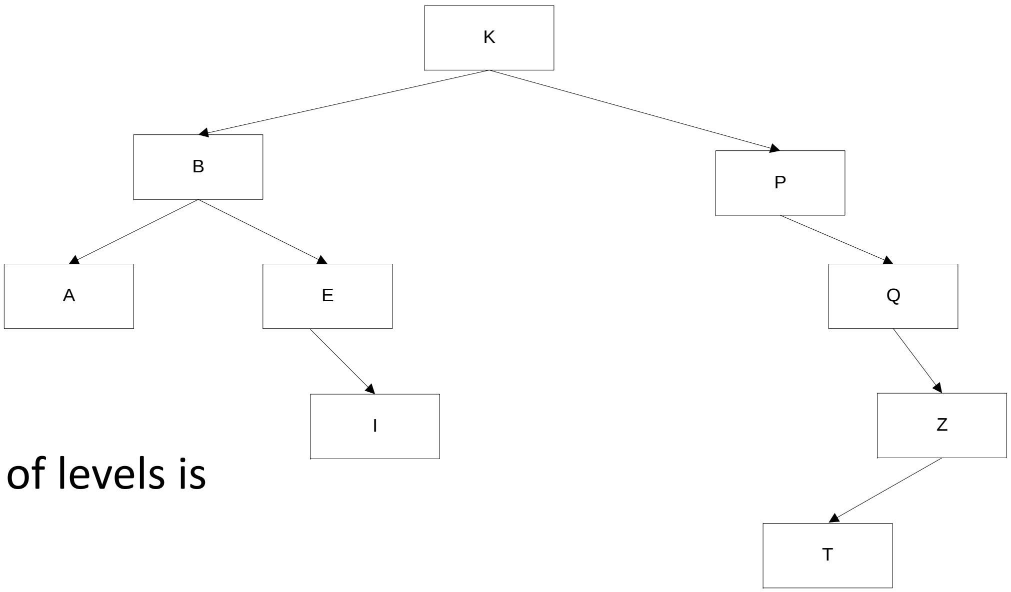
i.e. it is ordered!

Cost of ordering: O(N)

Binary Tree Search

What is the cost of finding a data item?

Maximum number of compares = number of levels



In complete binary tree, the number of levels is $\log_2(N)$

Thus the cost of finding a data item in a complete binary tree is $O(\log_2(N))$