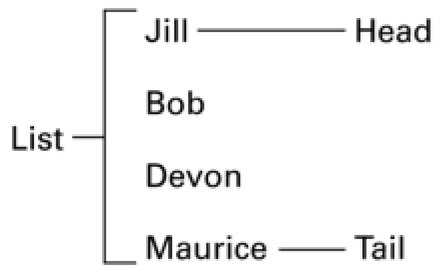
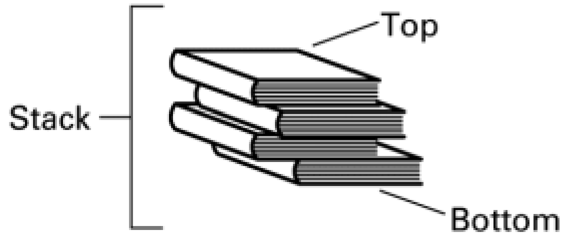
Data structure

Data abstraction

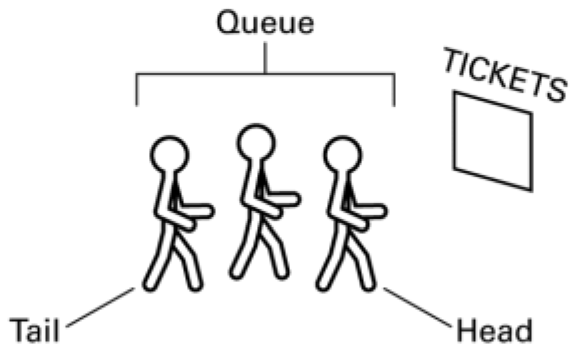
- Main memory is organized as a sequence of addressable cells, but the data we want to model is usually not.



a. A list of names



b. A stack of books



c. A queue of people

- Good data structures help the design of efficient algorithms

Example: insertion sort

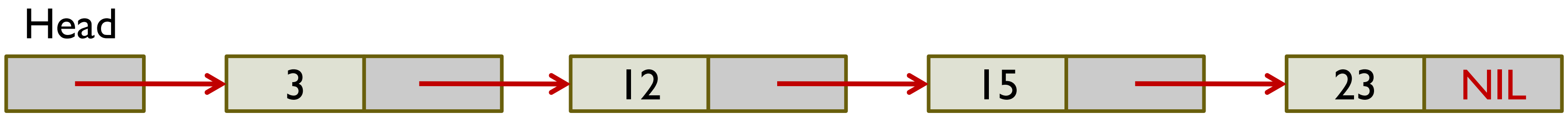
- Suppose the data is coming one by one. You do not have entire dataset in the beginning. But you need to maintain a sorted list of received data.
- Algorithm:
  - Let 'sorted' be the list containing sorted data, N be the size of sorted, and x be the incoming data.
  - If x>sorted[N]

```
If x>sorted[N]
    j = N+1
else
    Find j such that sorted[j-1]<x<=sorted[j]
Insert x to sorted[j]
```

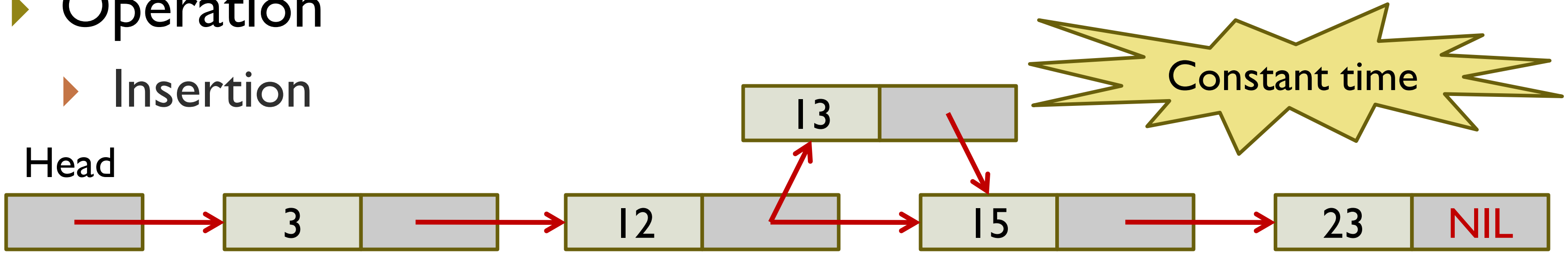
When describing an algorithm, you need to specify what data structure is used

If "sorted" is a linked list

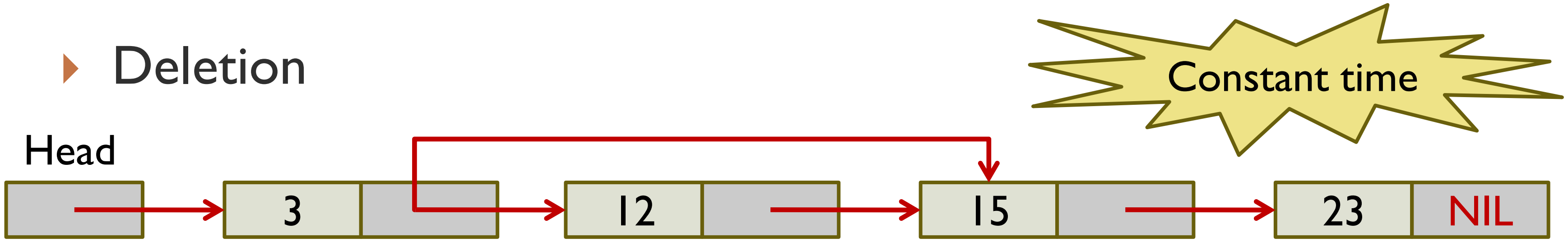
- A list in which each entries are linked by pointers
  - **Head pointer:** Pointer to the first entry in a list
  - **NIL pointer:** A value indicating the end of a list



- Operation
  - Insertion



  - Deletion

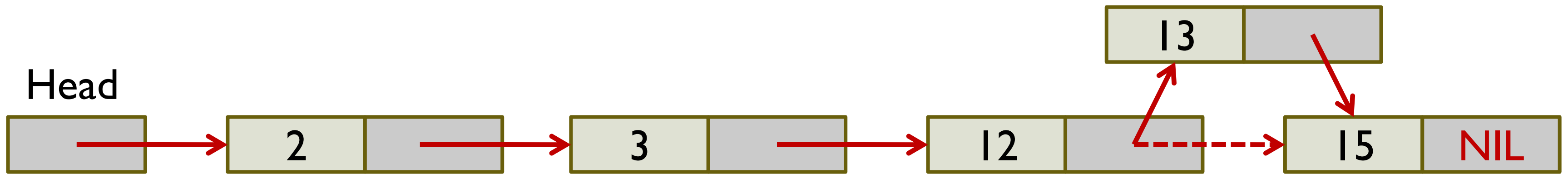


Implement linked list by arrays

	0	1	2	3	4	5	6	7	8	9
data	12	3	15	2	13					
next	4	0	-1	1	2	-1	-1	-1	-1	-1

Number_of_data = 5

Head = 3



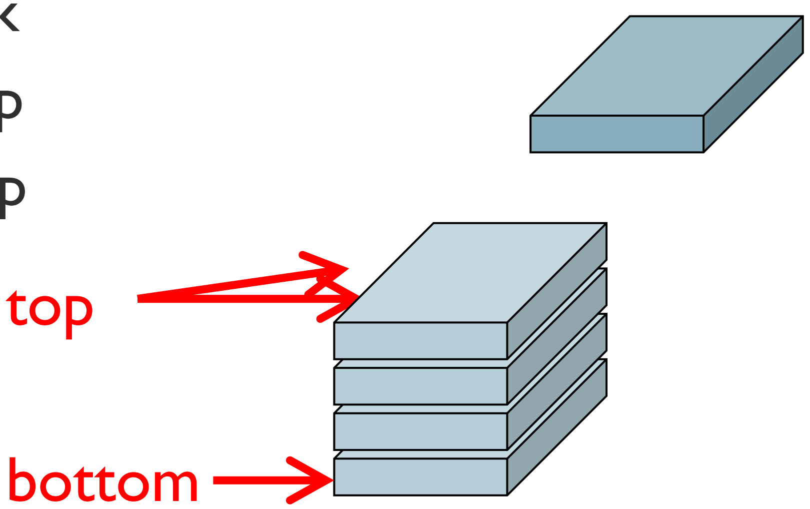
Summary

- Array
  - Sorted data can be searched by binary search in O(logN) time
  - Insertion/deletion takes O(N) time for data movement
  - The size is fixed.
- Linked list
  - Sorted data need be searched in O(N) time
  - Insertion/deletion takes O(1) time
  - The size is flexible.
- For the sorting problem
  - Better algorithms that use data structure, like tree or heap, can achieve O(NlogN) time for sorting.

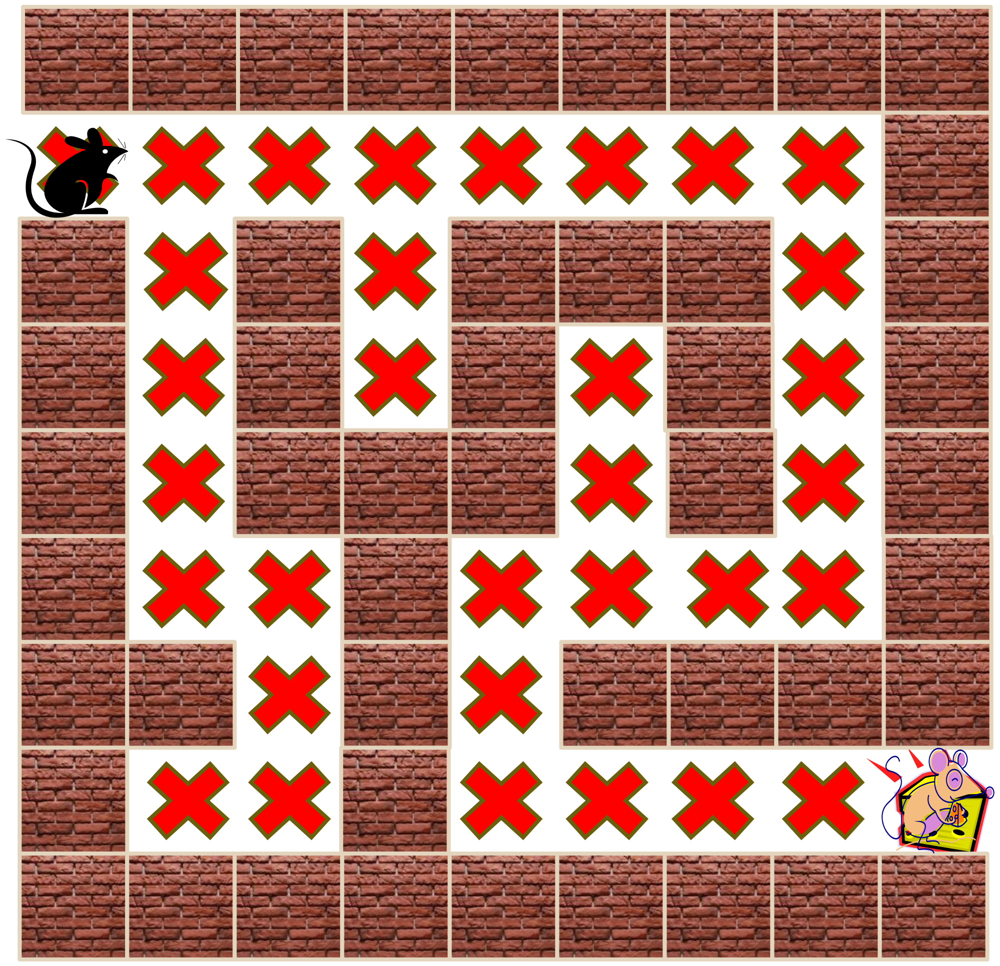
- More in the course algorithm(演算法) and data structure(資料結構)

Stack

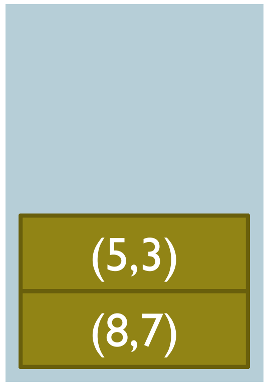
- A list in which entries are inserted/deleted only at head
  - **Top:** The head of stack
  - **Bottom** or **base:** The tail of stack
  - **Push:** To insert an entry at the top
  - **Pop:** To delete the entry at the top
- **LIFO:** Last-in-first-out



Example: mouse maze

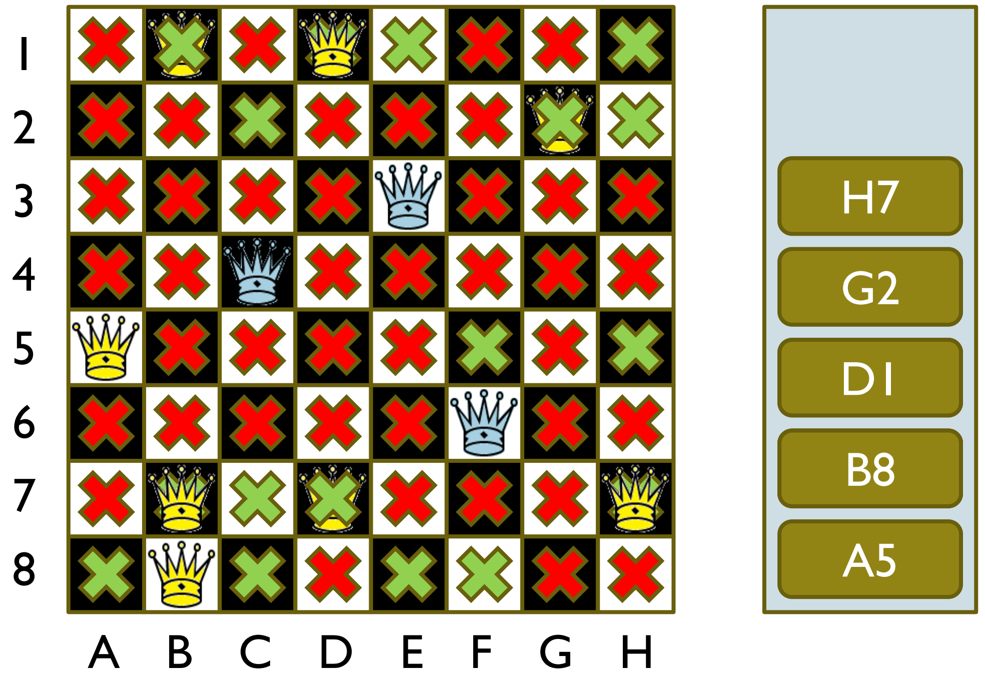


- The order of trial is RIGHT, LEFT, DOWN, and UP



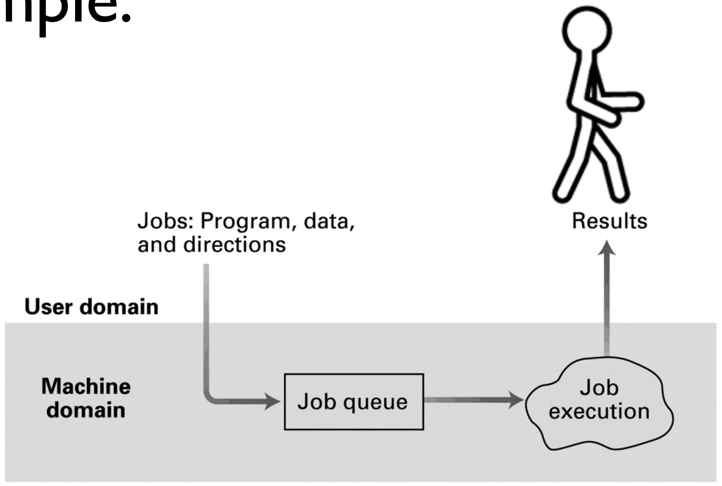
Example: Eight Queens Problem

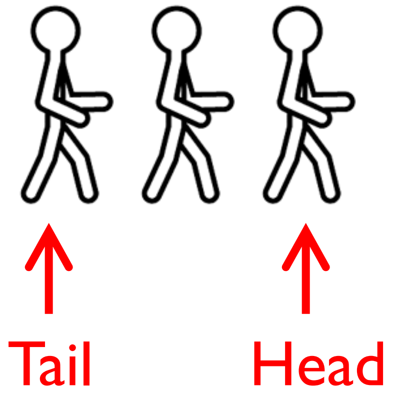
- To place eight chess queens on an 8×8 chessboard so that none of them can capture any other using the standard chess queen's moves.
  - proposed in 1848 by the chess player Max Bezzel



Queue

- A list in which entries are removed at the **head** and are inserted at the **tail**.
  - **Enqueue:** insert an entry at the **tail**
  - **Dequeue:** remove an entry at the **head**
  - **FIFO:** First-in-first-out
- Example:

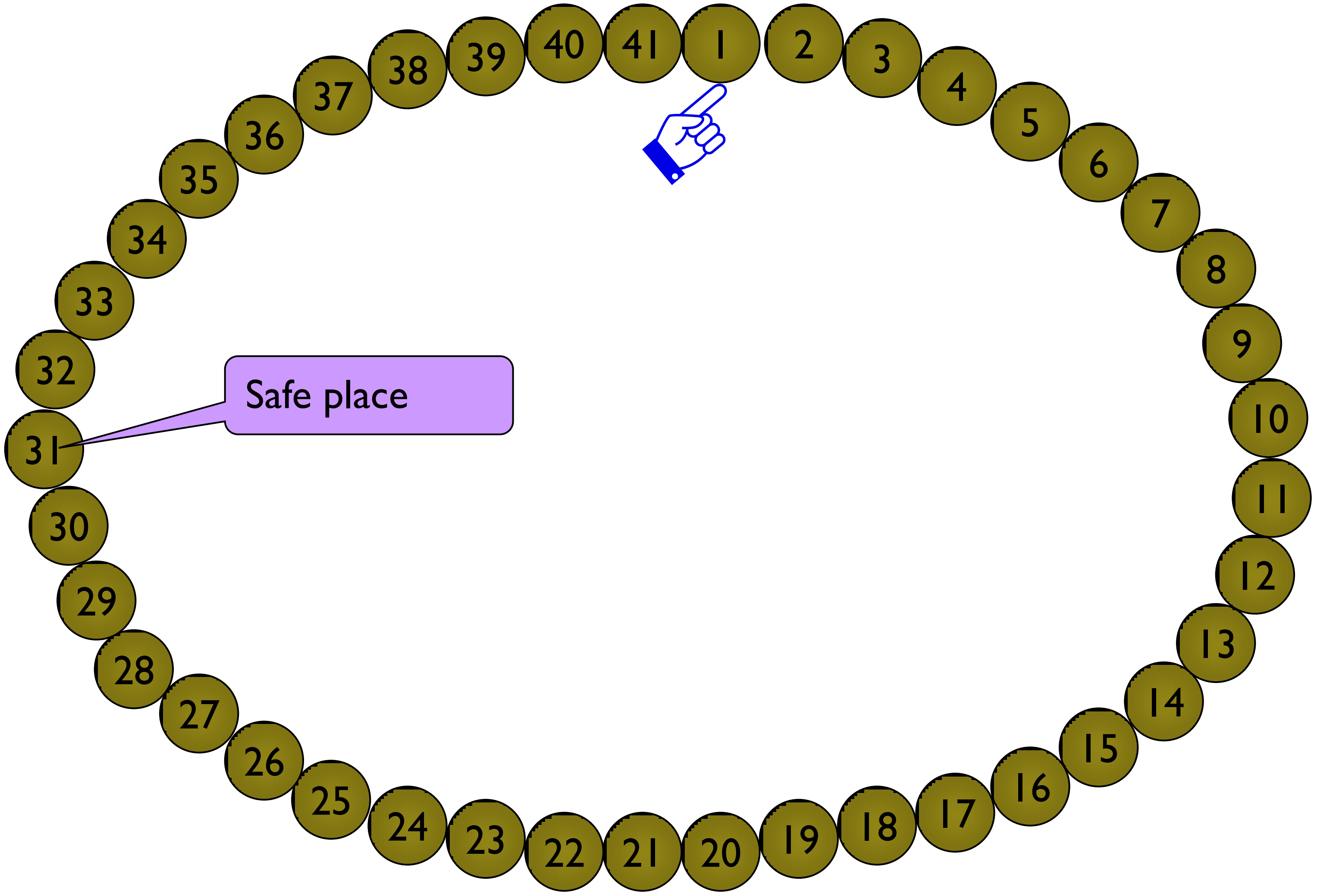




Example: Josephus problem

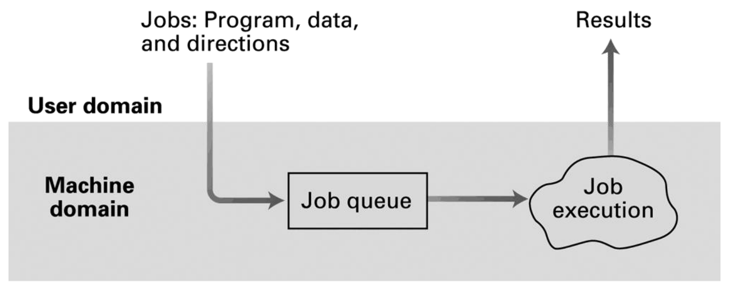
Flavius Josephus is a Jewish historian living in the 1st century. According to his account, he and his 40 comrade soldiers were trapped in a cave, surrounded by Romans. They chose suicide over capture and decided that they would form a circle and start killing themselves using a step of three. As Josephus did not want to die, he was able to find the safe place, and stayed alive with his comrade, later joining the Romans who captured them.

Can you find the safe place?

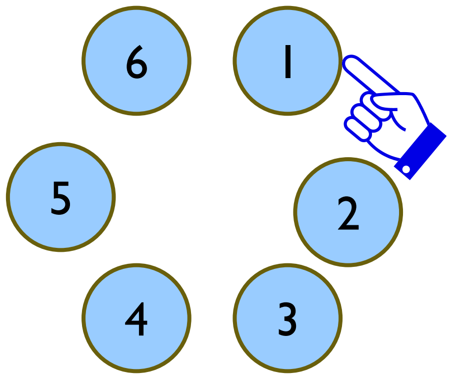


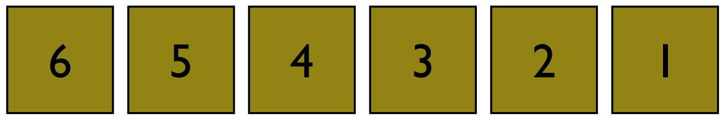
Examples of using queues

- Ex1: the job queues in operating system



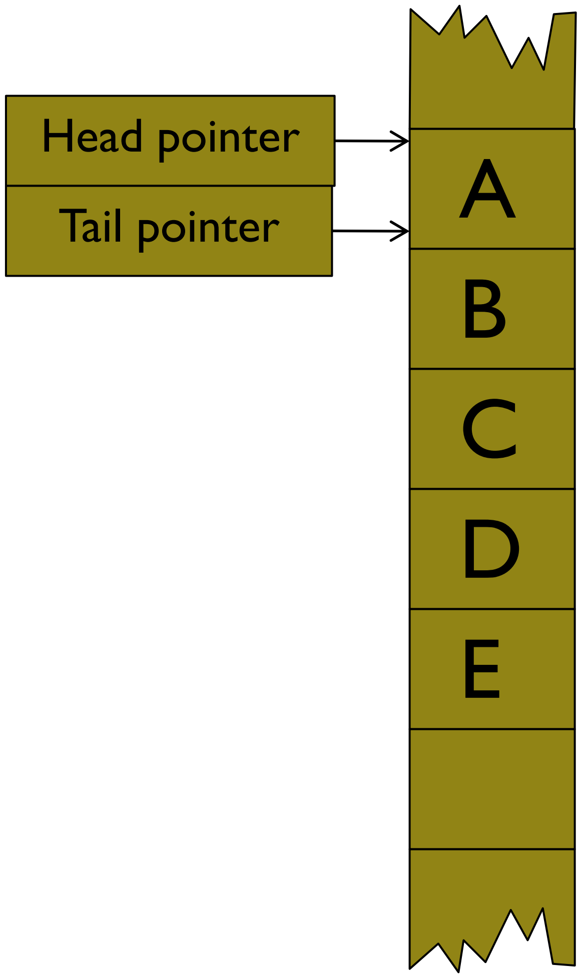
- Ex2: simulation of the Josephus problem
  - Dequeue 1
  - Enqueue 1
  - Dequeue 2
  - Dequeue 3
  - Enqueue 3





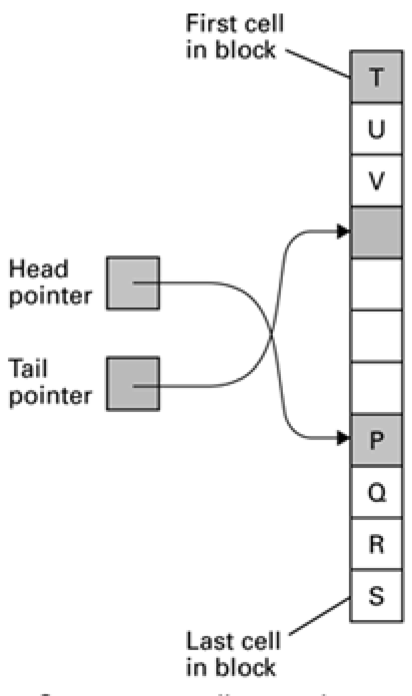
Queue implementation

- A list + 2 pointers (head+tail)
  - Enqueue A, B, C
  - Dequeue A, enqueue D
  - Dequeue B, enqueue E
- If using a static list, the queue crawls through memory as entities are inserted and removed.

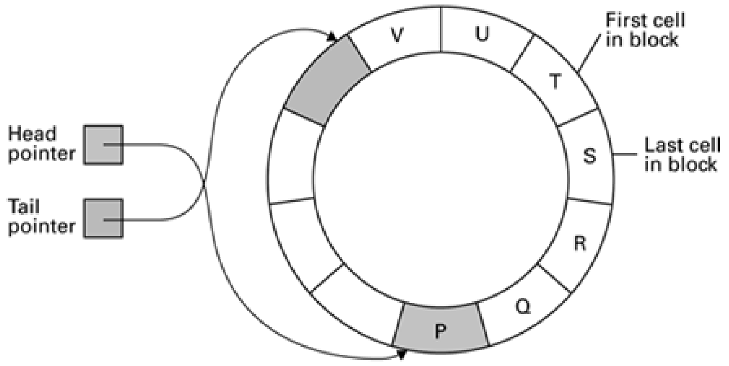


Circular queue

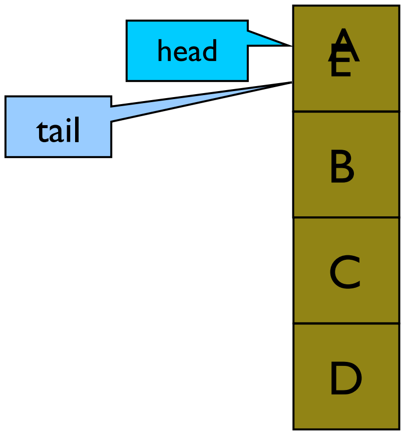
- A technique that uses a fixed region of memory space to implement queue.



a. Queue as actually stored



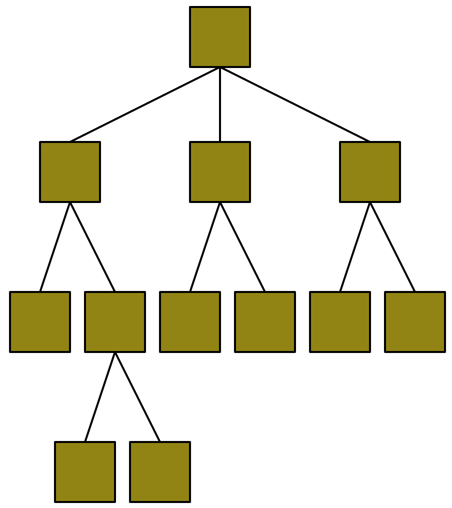
b. Conceptual storage with last cell "adjacent" to first cell



Enqueue A, B, C
Dequeue A, Enqueue D
Dequeue B, Enqueue E

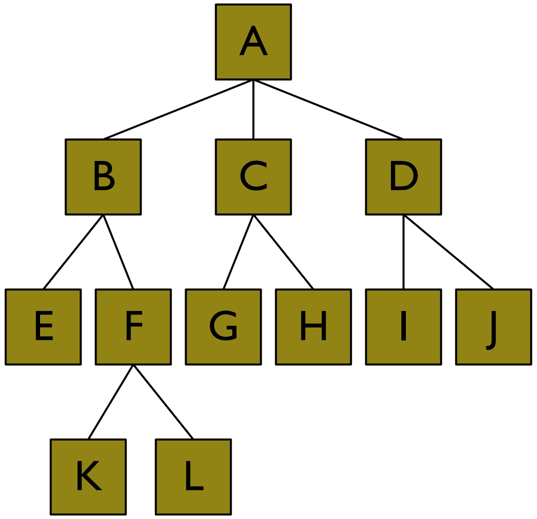
What is a tree?

- A collection of **nodes** that are linked in a hierarchical structure, in which every **node** is linked by one **parent**, except the **root**.
  - **Node:** An entry in a tree
  - **Parent:** The node immediately above a specified node
  - **Root:** The node at the top
  - **Terminal** or **leaf node:** A node at the bottom



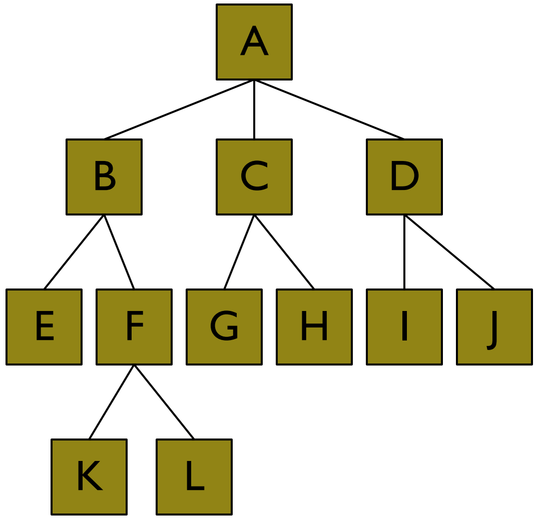
Hierarchical relations

- **Parent:** The node immediately above a node
  - The parent of **F** is **B**
- **Child:** A node immediately below a node
  - The children of **C** are **G** and **H**.
- **Ancestor:** Parent, parent of parent, etc.
  - The ancestor of **K** are **F**, **B**, and **A**.
- **Descendent:** Child, child of child, etc.
  - The descendent of **B** are **E**, **F**, **K**, and **L**.
- **Siblings:** Nodes sharing a common parent
  - The siblings of **C** are **B** and **D**.



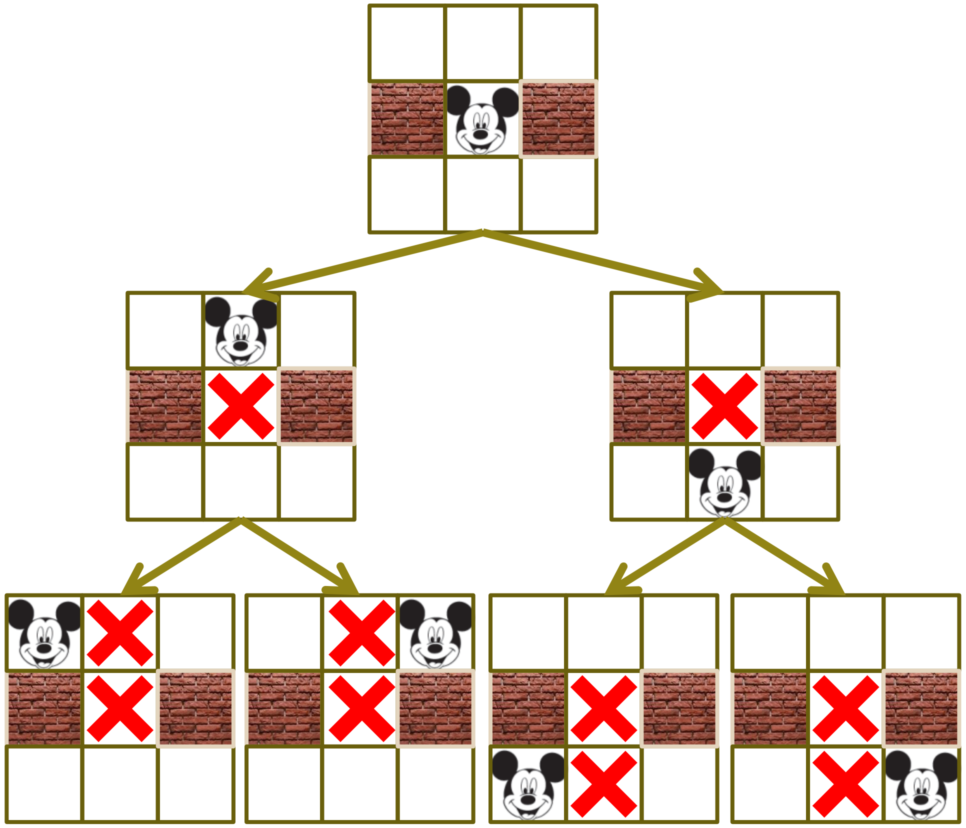
Depth and height

- Textbook's definition
  - The **depth** of a tree is the longest path from the root to a leaf node
    - The length of a path is the number of nodes on the path
    - Ex: the depth of the tree is 4
- Conventional definition
    - Use the word "height" instead of depth
    - The length of a path is the number of links on the path
    - Ex: The height of the tree is 3 (= 4 – 1)



What are trees used for?

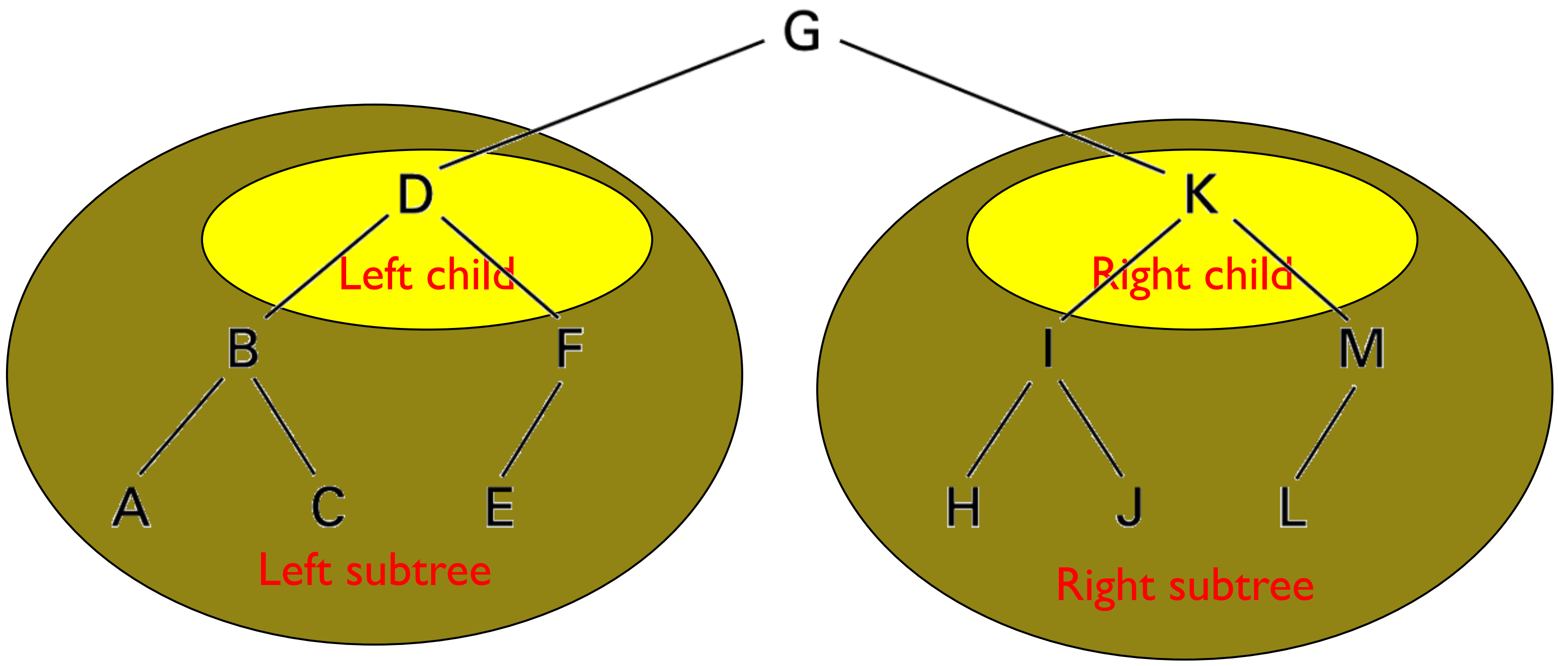
- Example: game tree for mouse maze
- Example: game tree for eight queens



$$\begin{tikzcd}
 & & & \text{C4, E3, F6} \arrow[dl] \arrow[dr] & \\
 & & \text{A5} \arrow[dll] \arrow[dl] \arrow[d] & & \text{A8} \arrow[d] \\
\text{B1} \arrow[d] & \text{B7} \arrow[d] & \text{B8} \arrow[d] \arrow[dr] & & \text{B1} \arrow[d] \\
\text{D7} \arrow[d] & \text{D1} & \text{D1} \arrow[d] & \text{D7} \arrow[d] & \text{D7} \\
\text{G2} & & \text{G2} \arrow[d] & \text{G2} & \\
 & & \text{H7} & &
\end{tikzcd}$$

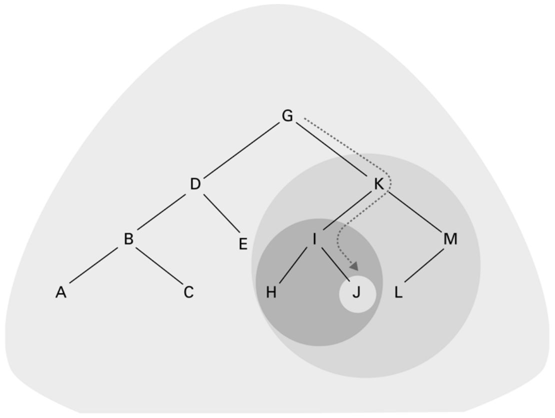
Binary tree

- A tree in which each parent has at most two children



Recursive structure

- Tree is a recursive structure
  - The subtrees of a tree are trees
- The recursive algorithms for a binary tree may look like this

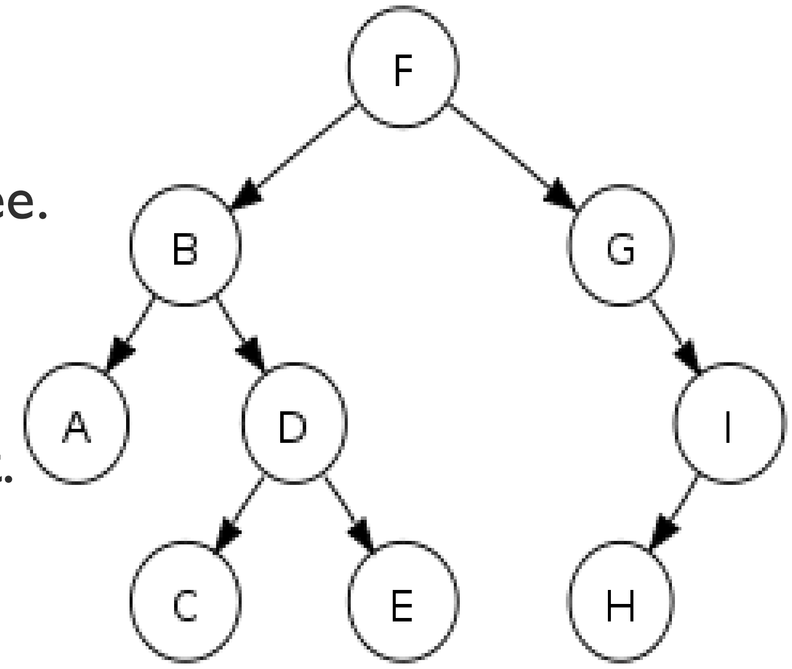


```
procedure some_operation (root)
   if (root is not NULL) then
      ( call some_operation(root.left_child)
        do some operations on root
        call some_operation(root.right_child))
```

  - It is a depth first, in order algorithm for tree

Depth first search (DFS)

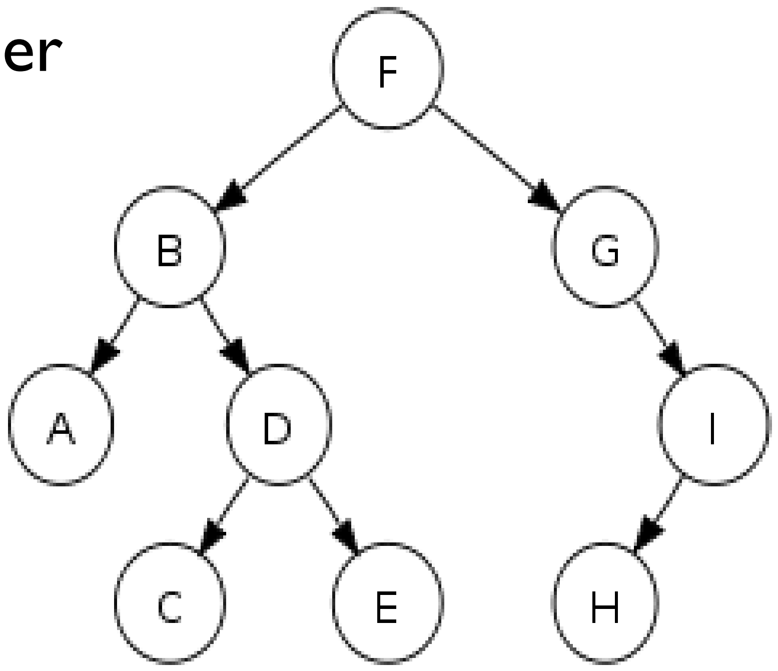
- Both mouse maze and eight queens problem use DFS
- Preorder
  - Root.➔ left subtree.➔right subtree.
  - F, B, A, D, C, E, G, I, H
- Inorder
  - Left subtree.➔ root.➔ right subtree.
  - A, B, C, D, E, F, G, H, I
- Postorder
  - Left subtree.➔right subtree.➔root.
  - A, C, E, D, B, H, I, G, F



Tree figure from Wikipedia: http://en.wikipedia.org/wiki/Tree_traversal

DFS uses stack

- DFS and pre-order



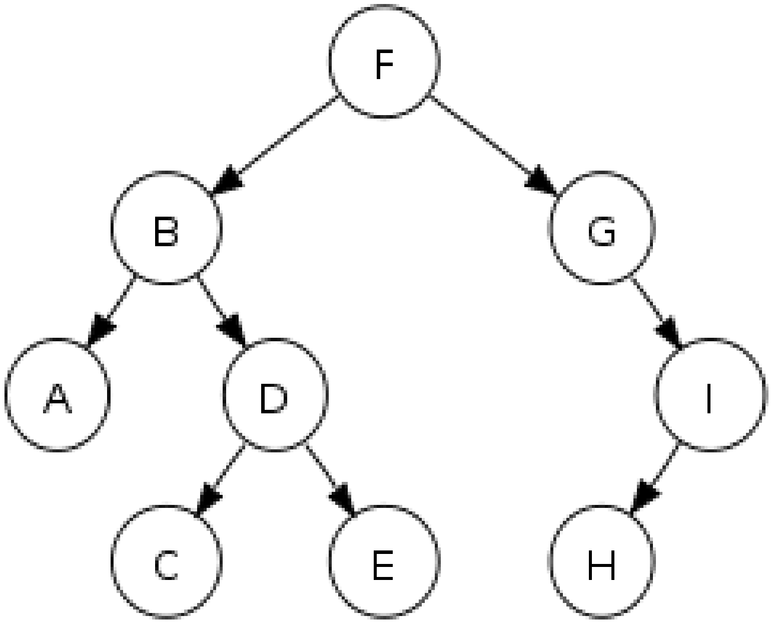
F, B, A, D, C, E, G, I, H

push F
pop F
push G B
pop B
push D A
pop A D
push E C
pop C E G
push I
pop I
push H
pop H

- Inorder and postorder can be done similarly

Breadth first search (BFS)

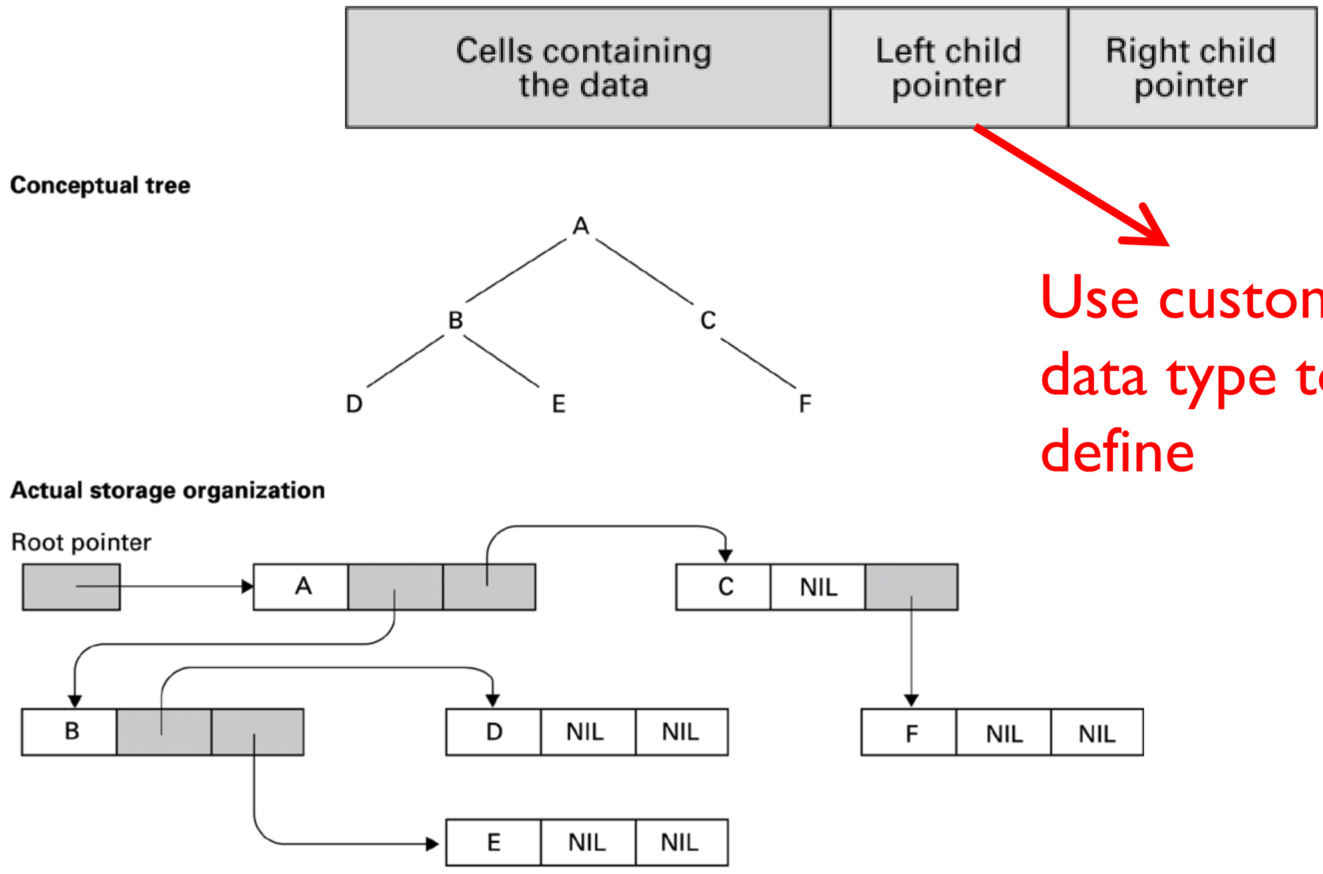
- BFS visits every node on a level before going to a lower level
  - F, B, G, A, D, I, C, E, H
- Uses queue to implement the BFS



Enqueue F
Dequeue F
Enqueue B G
Dequeue B
Enqueue A D
Dequeu G
Enqueue I
Dequeue A
Dequeue D
Enqueue C E
Dequeue I
Enqueue H
Dequeue H

Storing a binary tree using pointers

- Each node

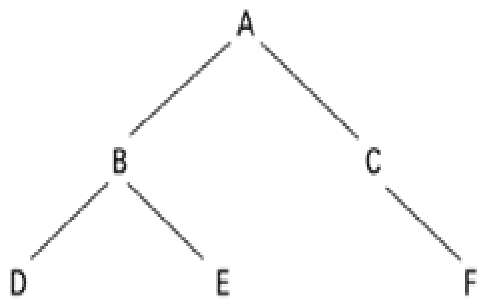


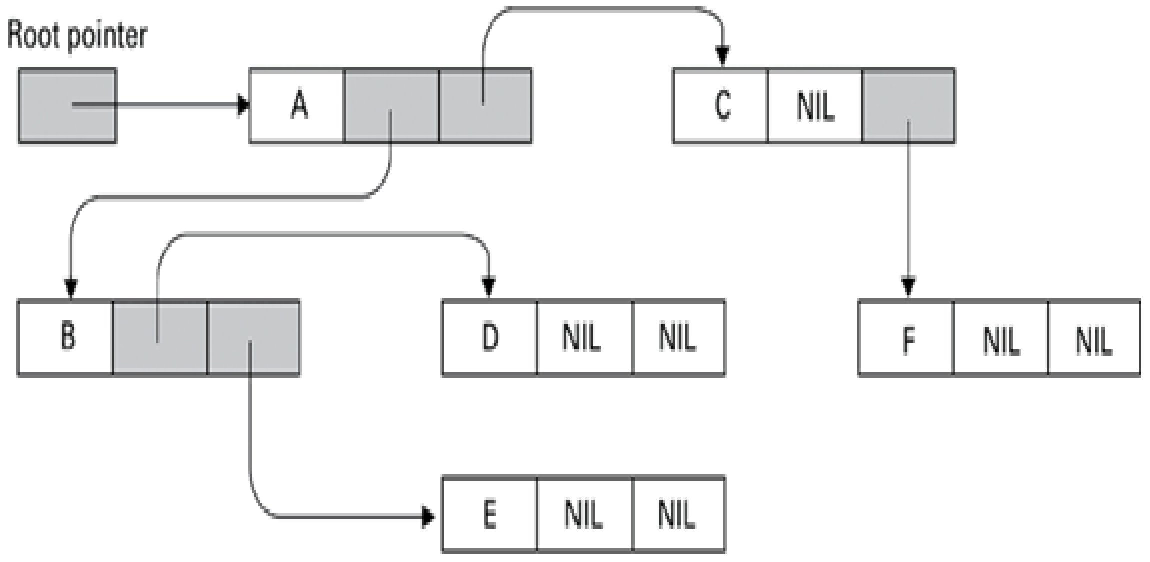
Use customized data type to define

Simulate pointers using array

Root = 2

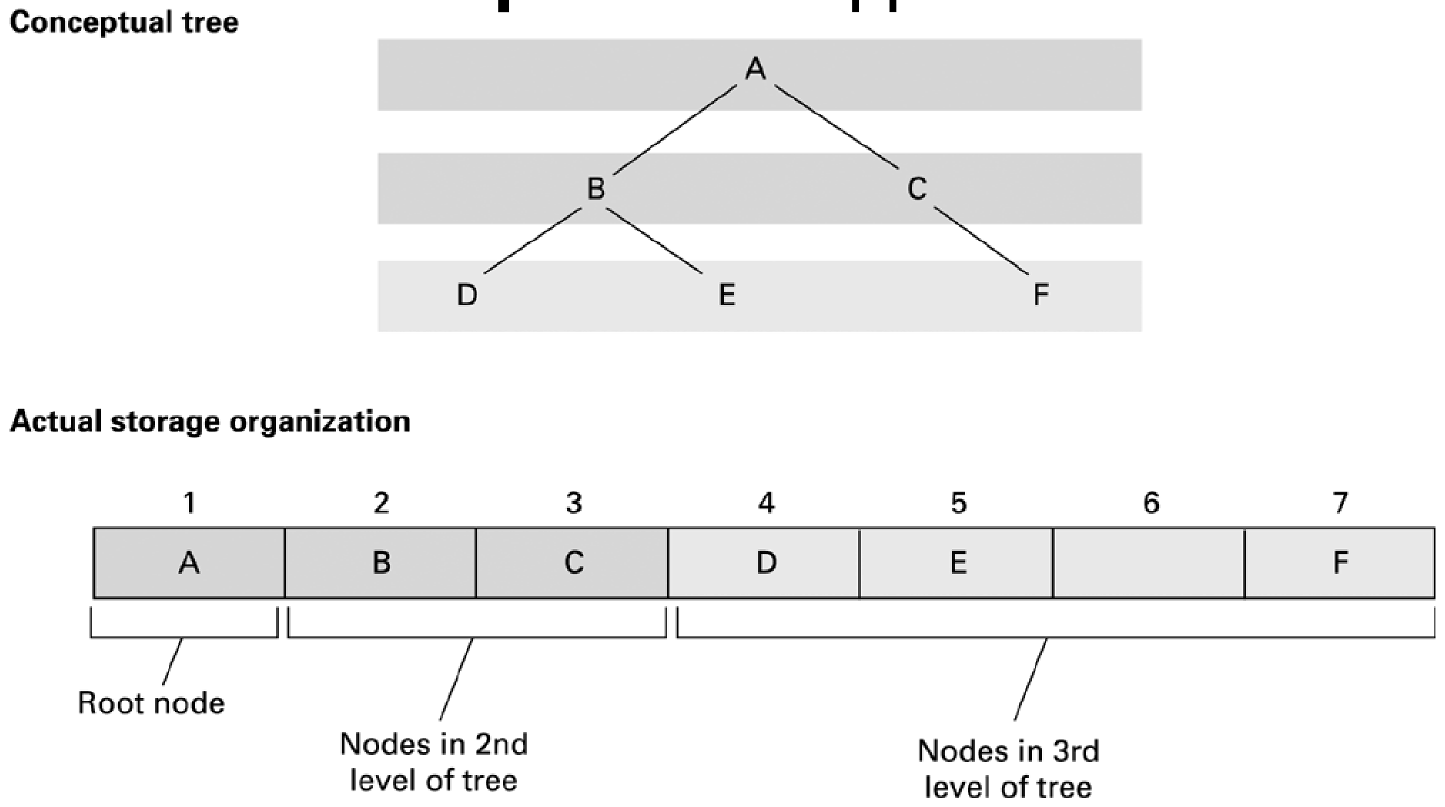
	0	1	2	3	4	5
Data	**C**	**F**	**A**	**D**	**B**	**E**
Left child	-1	-1	4	-1	3	-1
Right child	1	-1	0	-1	5	-1





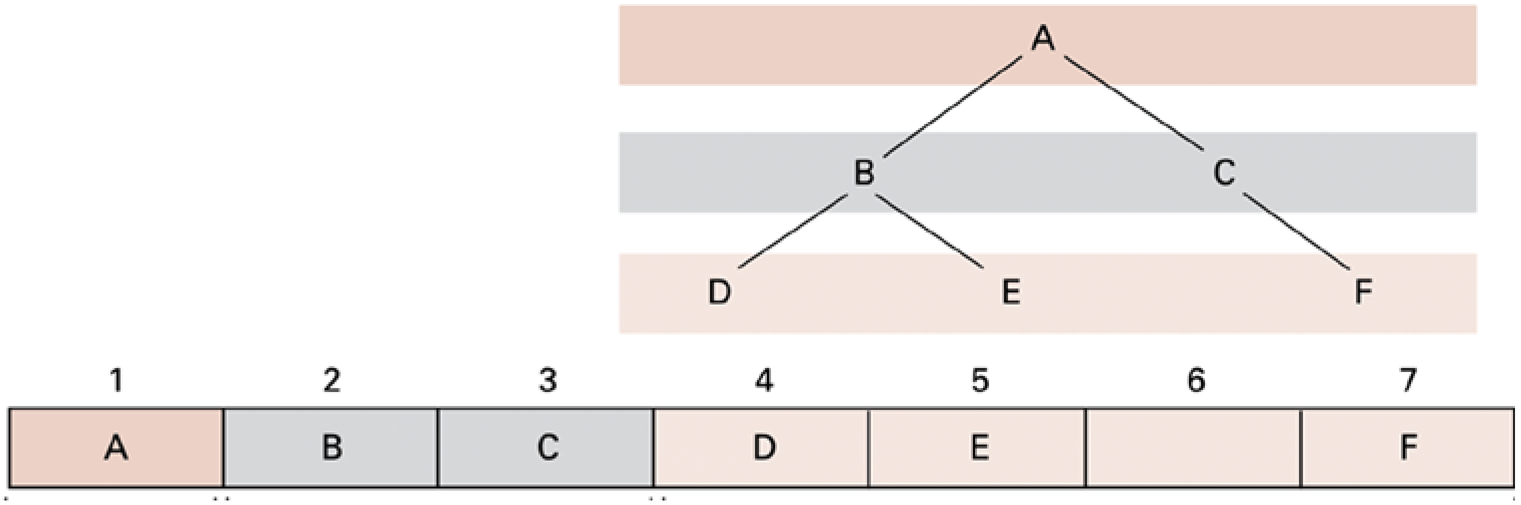
Storing a binary tree in a list

- This is called a **heap** in some applications.


Conceptual tree
A
B
C
D
E
F
Actual storage organization
1
2
3
4
5
6
7
A
B
C
D
E
F
Root node
Nodes in 2nd level of tree
Nodes in 3rd level of tree

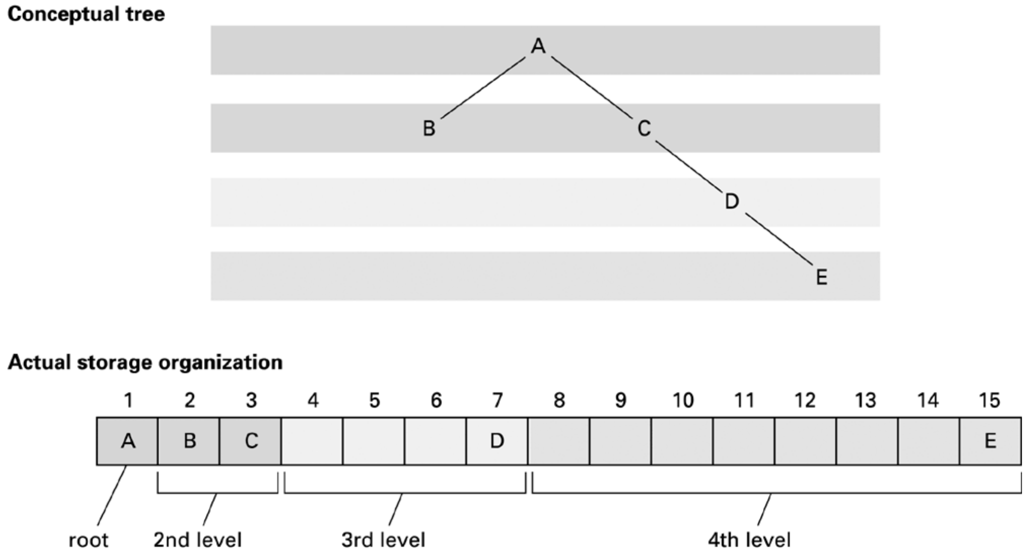
Advantages of using heap

- Easily to find the index of parent & children
  - Parent(B) = [index of B] / 2 = 1
  - LeftChild(B) = [index of B]*2 = 4
  - RightChild(B) = [index of B]*2 + 1= 5



Problems of heap

- Heap is inefficient for storing the binary tree that is sparse and unbalanced
  - Sparse: most node has one or zero child
  - Unbalanced: the right subtree is much larger than the left subtree, or vice versa



Conceptual tree
A
B
C
D
E
Actual storage organization
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A B C D E
root
2nd level
3rd level
4th level