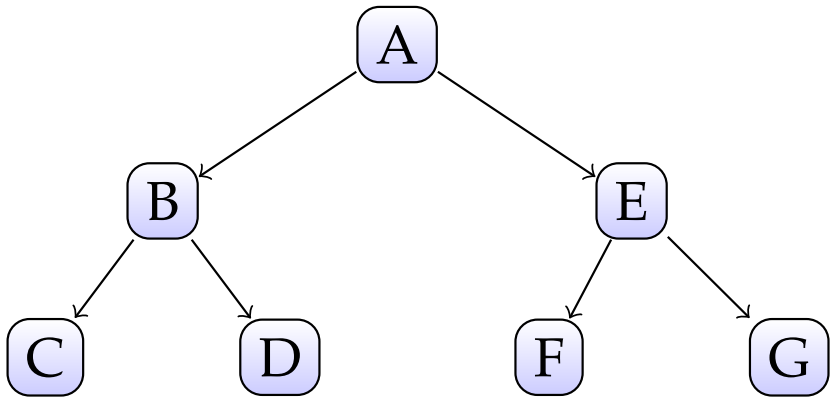
CSCI 2270: Data Structures

Lecture 14: Trees (introduction)

Ashutosh Trivedi



Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Stacks and Queues: Recap

Trees

Stacks: Recap

1. Lists with LIFO (last-in first-out structure)
2. Insertions and deletions happen from the same side, called the *top* of the stack.
3. An *insertion* is called a push to the stack
4. A *deletion* is called a pop from the stack
5. You can implement them using array or linked lists
6. When implementing as an array, keep the top of the stack as the highest occupied index
7. When implementing as a linked list, keep the top of the stack as the head of the list.

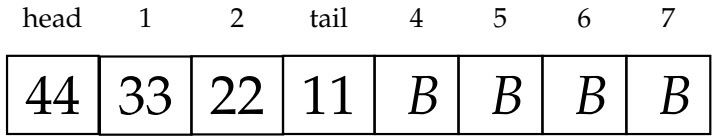
Some applications of stacks

- Reverse a string — How?
- Check for balanced parenthesis:
  - $(5 + 3) * ((8 * 2)/7)$
  - $if(x < 4)\{if(y < 3)\{cout <<" hello"; \}\}\{\}$
  - ((()()())) and ((((()(()))))))) versus))(((or ((())((()))))
- Check if a string is palindrome (Able was I ere I saw Elba). — How?
- Infix, prefix, and postfix expressions:
  - (infix): A + B * C + D (A + B) * (C + D) A * B + C * D
  - (prefix): + + A * B C D * + A B + C D + * A B * C D
  - (postfix): A B C * + D + A B + C D + * A B * C D * +
- Tree and Graph Traversal. — Later in this course

Queues: Recap

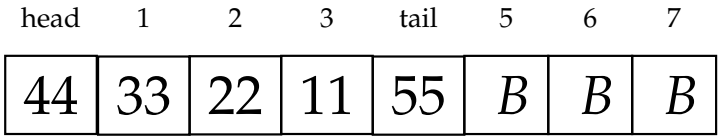
1. Lists with FIFO (first-in first-out structure)
2. Insertions and deletions happen from the opposite side of the queue, called the head and tail of the queue, respectively.
3. An *insertion* is called a enqueue to the "tail" end of the queue
4. A *deletion* is called a dequeue from the "head" end of the queue.
5. You can implement them using array or linked lists
6. When implementing as a linked list, keep the "head" of the queue as the first node of the list and "tail" as the "end" of the list. Why?
7. When implementing as an array, both options were equally bad!
8. Solution: Circular array based queue implementation.

Queue implemented as a Circular Array



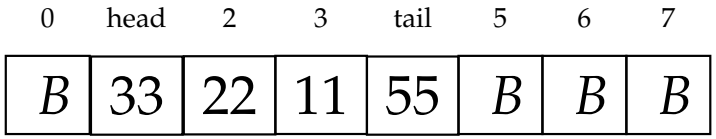
- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array



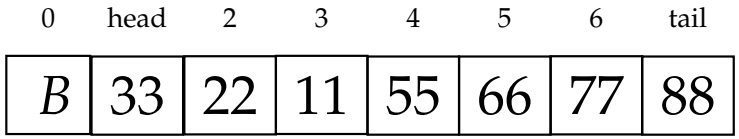
- **enqueue(55);**
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array



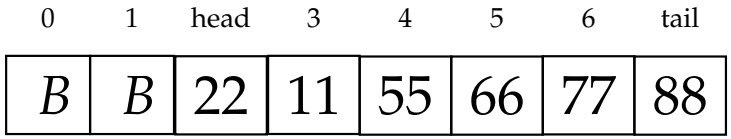
- enqueue(55);
- **dequeue();**
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array



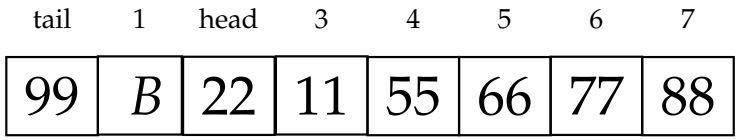
- enqueue(55);
- dequeue();
- **enqueue(66); enqueue(77); enqueue(88);**
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array



- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- **dequeue();**
- enqueue(99);

Queue implemented as a Circular Array



- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- **enqueue(99);**

CircularArrayQueue: Enqueue

```
void CircularArrayQueue::enqueue(int element) {
  if (isFull()) {
    std::cerr << "Queue Overflow!! Enqueue failed" << std::endl;
  }
  else {
    if (head == -1) {
      //first element to insert
      head = 0;
      tail = 0;
      items[tail] = element;
    }
    else {
      if (tail == capacity-1) {
  items[0] = element;
  tail = 0;
      }
      else {
  tail = tail + 1;
  items[tail] = element;
      }
    }
  }
}
```

CircularArrayQueue: Dequeue

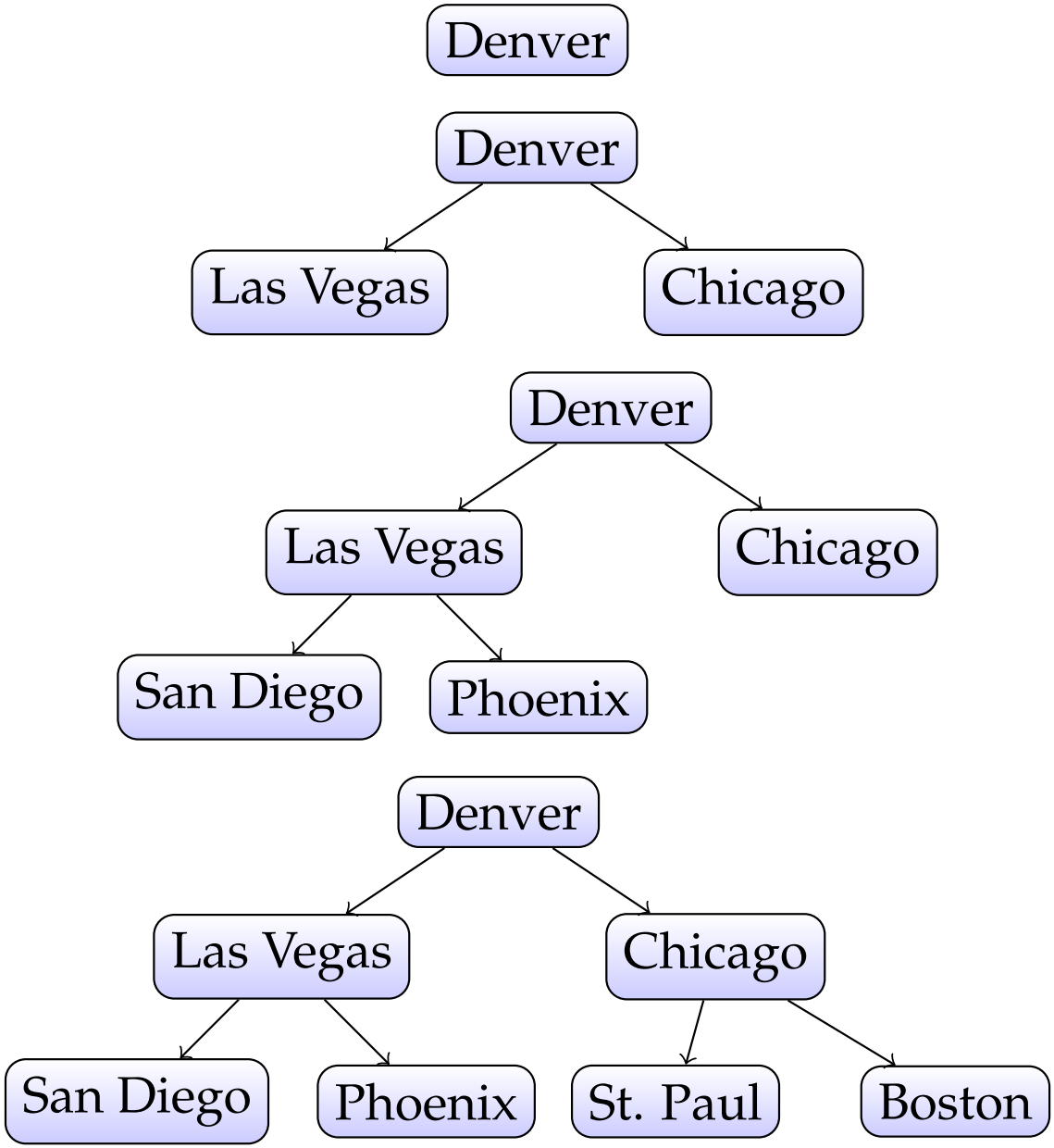
```
int CircularArrayQueue::dequeue() {
  if (isEmpty()) {
    std::cerr << "Queue Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[head];

    if (head == tail) {
      // Only one element in the queue
      head = -1;
      tail = -1;
    }
    else {
      if (head == capacity -1) {
  head = 0;
      }
      else {
  head = head + 1;
      }
    }
    return result;
  }
}
```

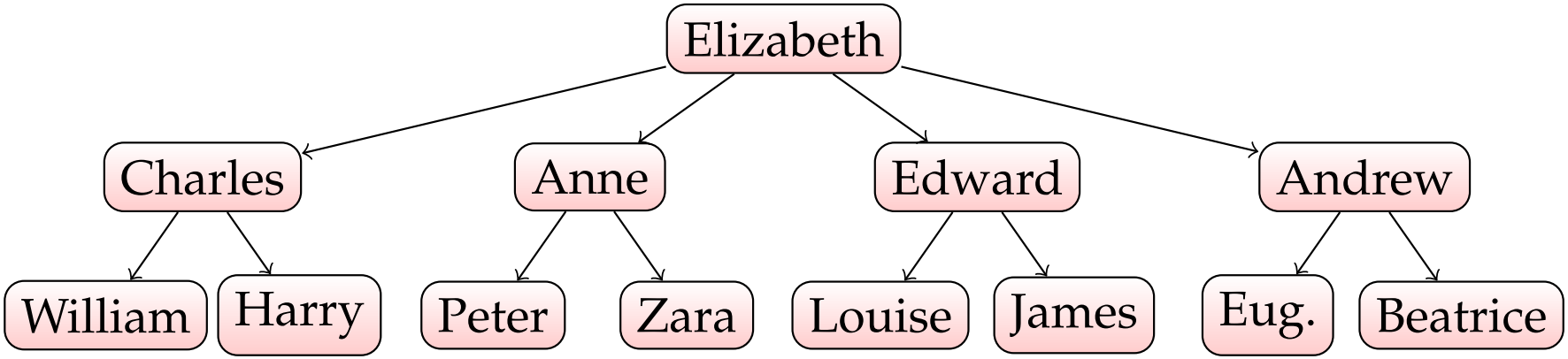
Stacks and Queues: Recap

Trees

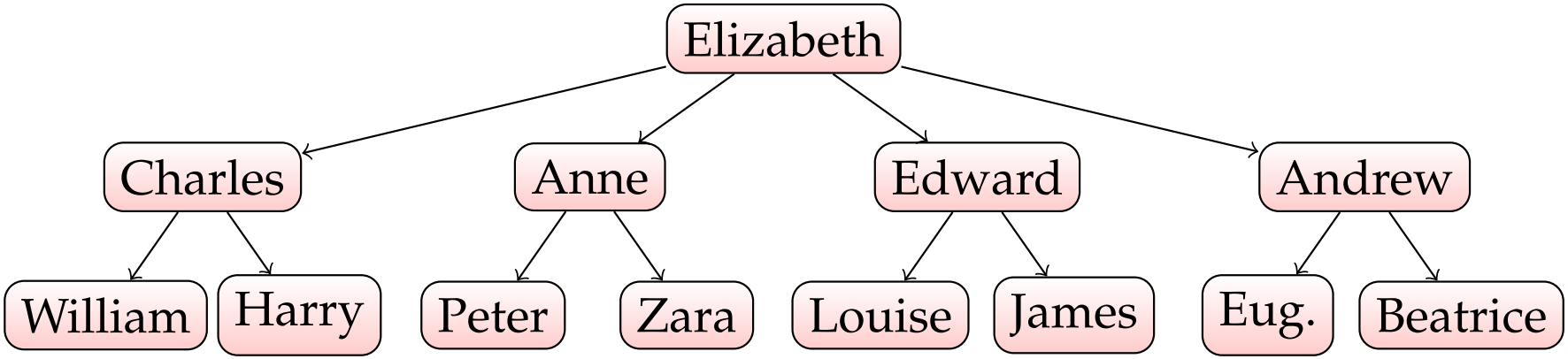
Trees: Terminology



Trees: Terminology

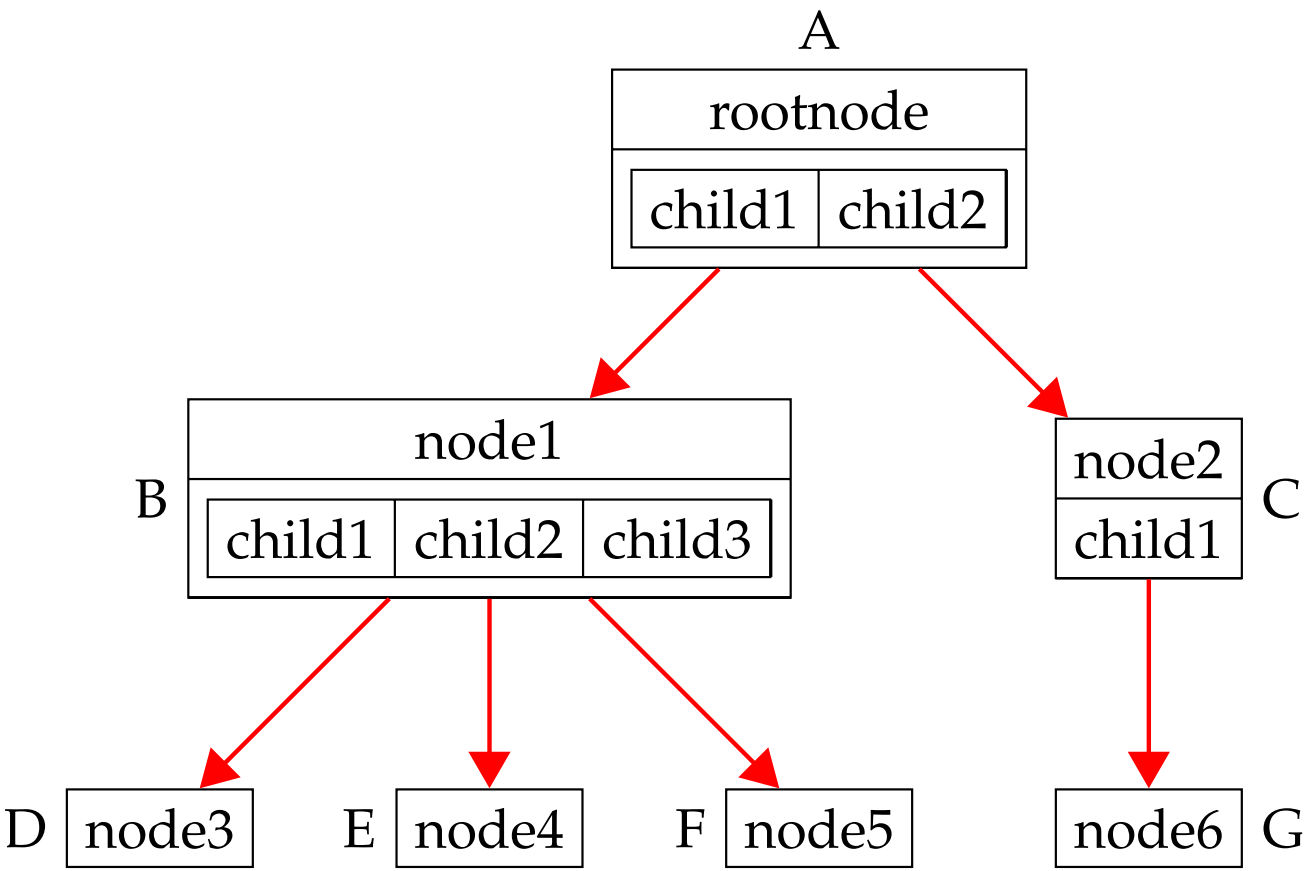


Trees: Terminology

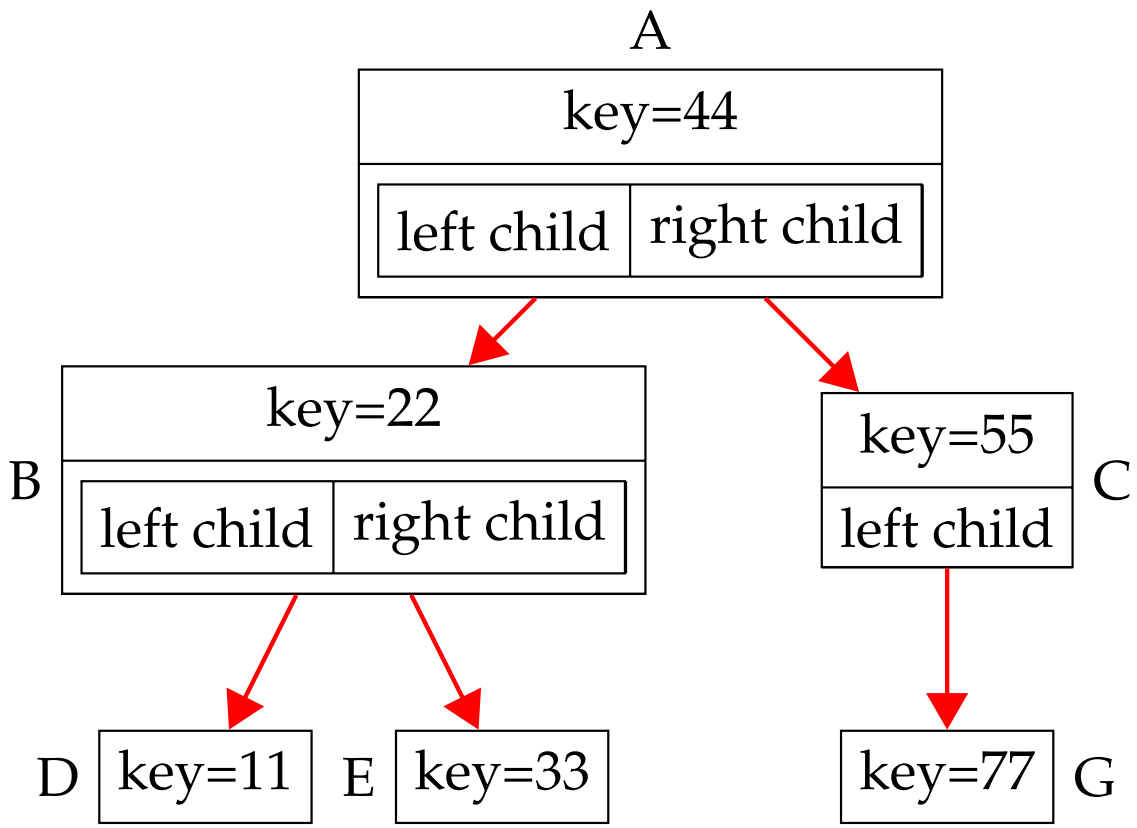


- *arity* of a tree: binary trees, ternary trees, k-ary trees, etc.
- *root* of a tree
- *leaf* of a tree
- *child* of a node
- *parent* of a node
- *ancestor* of a node
- *descendant* of a node
- *sibling* of a node

Trees as a linked structure

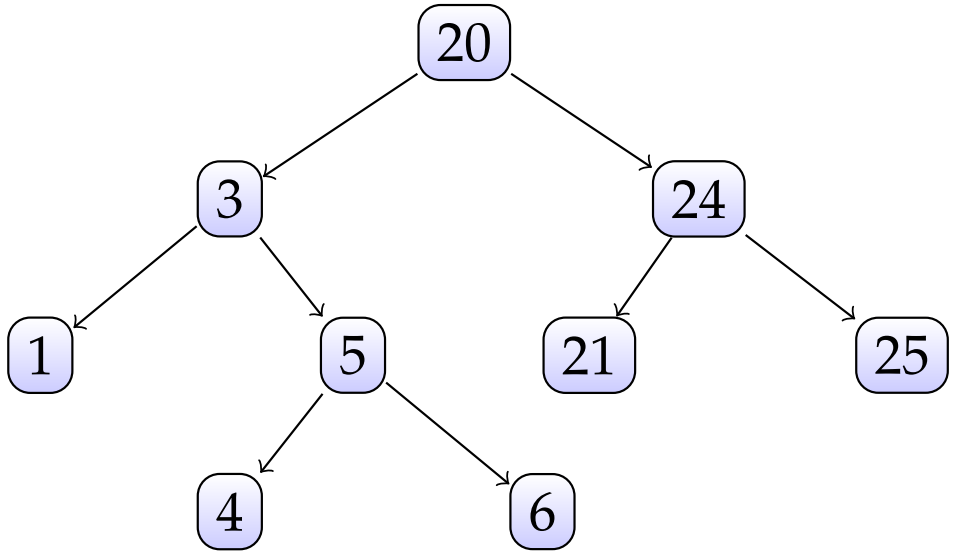


Binary Trees as a linked structure



- *left sub-tree* of a node
- *right sub-tree* of a node

Binary Trees



Properties: If x and y are nodes, and

1. y is in the left sub-tree of x, then

$$y.key < x.key$$

2. y is in the right sub-tree of x, then

$$y.key \geq x.key$$