ALG 04

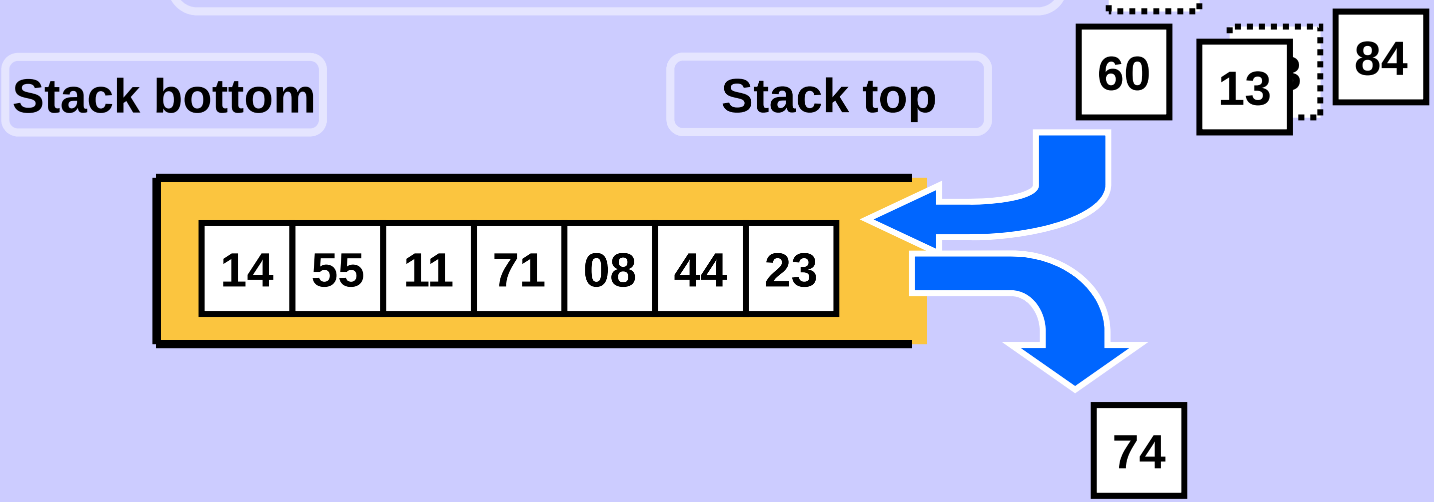
Queue

- Operations Enqueue, Dequeue, Front, Empty....
- Cyklic queue implementation

Breadth-first search (BFS) in a tree

Stack

Elements are stored at the stack top before they are processed.



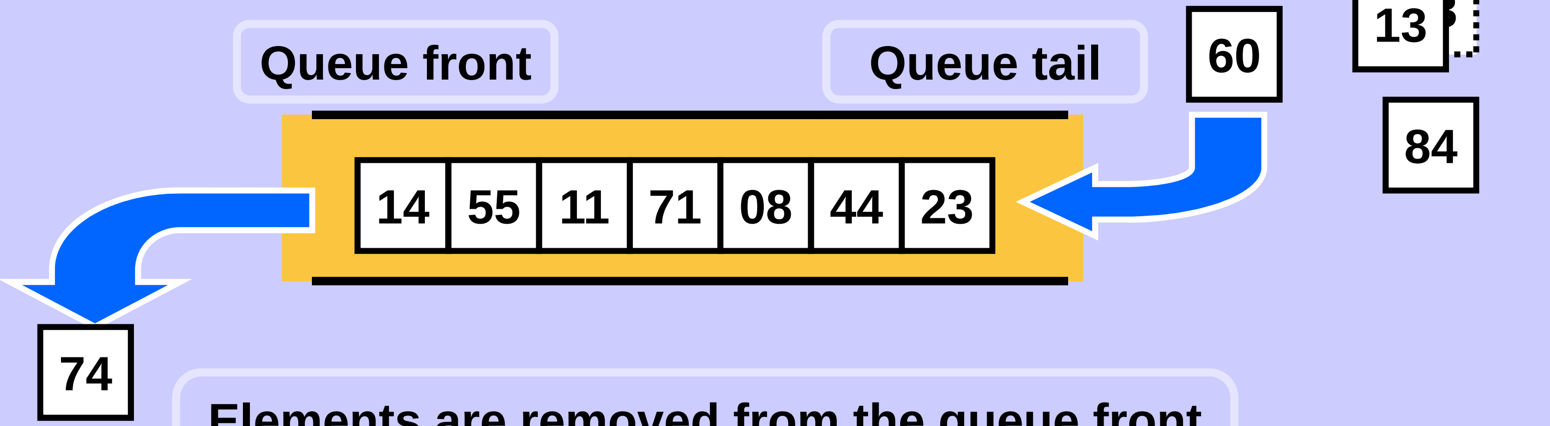
Elements are removed from the stack top and then they are processed.

Operation names

Put at the top	Push
Remove from the top	Pop
Read the top	Top
Is the stack empty?	Empty

Queue

Elements are stored at the queue tail before they are processed.



Elements are removed from the queue front and then they are processed.

Operation names

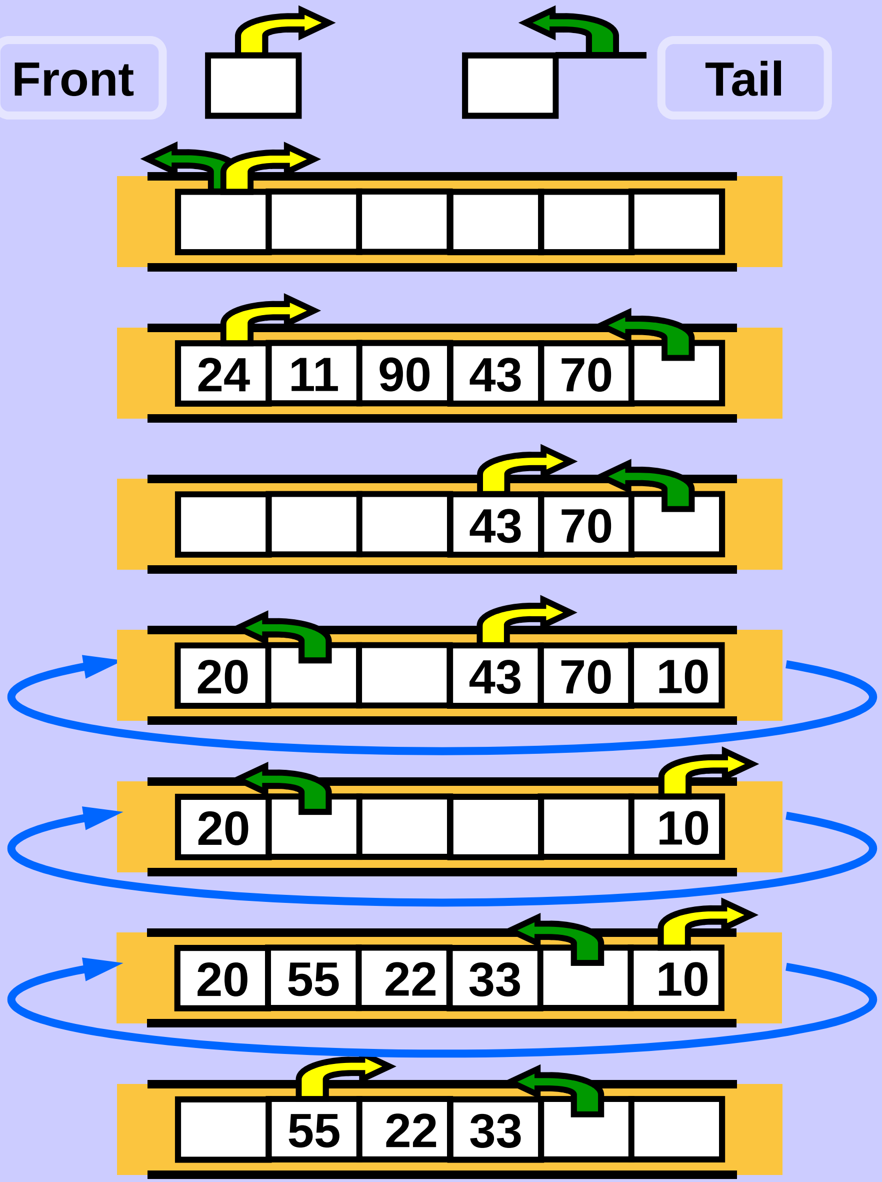
Insert at the tail	Enqueue / InsertLast / Push ...
Remove from the front	Dequeue / delFront / Pop ...
Read the front elem	Front / Peek ...
Is the queue empty?	Empty

Queue

Easy example of aqueue life cycle.

	Front … Tail
Empty	
Insert(24)	24
Insert(11)	24 11
Insert(90)	24 11 90
DelFront()	11 90
Insert(43)	11 90 43
DelFront()	90 43
DelFront()	43
Insert(79)	43 79

Cyclic queue implementation in an array



An empty queue in a fixed length array

Insert 24, 11, 90, 43, 70.

DelFront, DelFront, DelFront .

Insert 10, 20.

DelFront, DelFront .

Insert 55, 22, 33.

DelFront, DelFront .

Cyclic queue implementation in an array

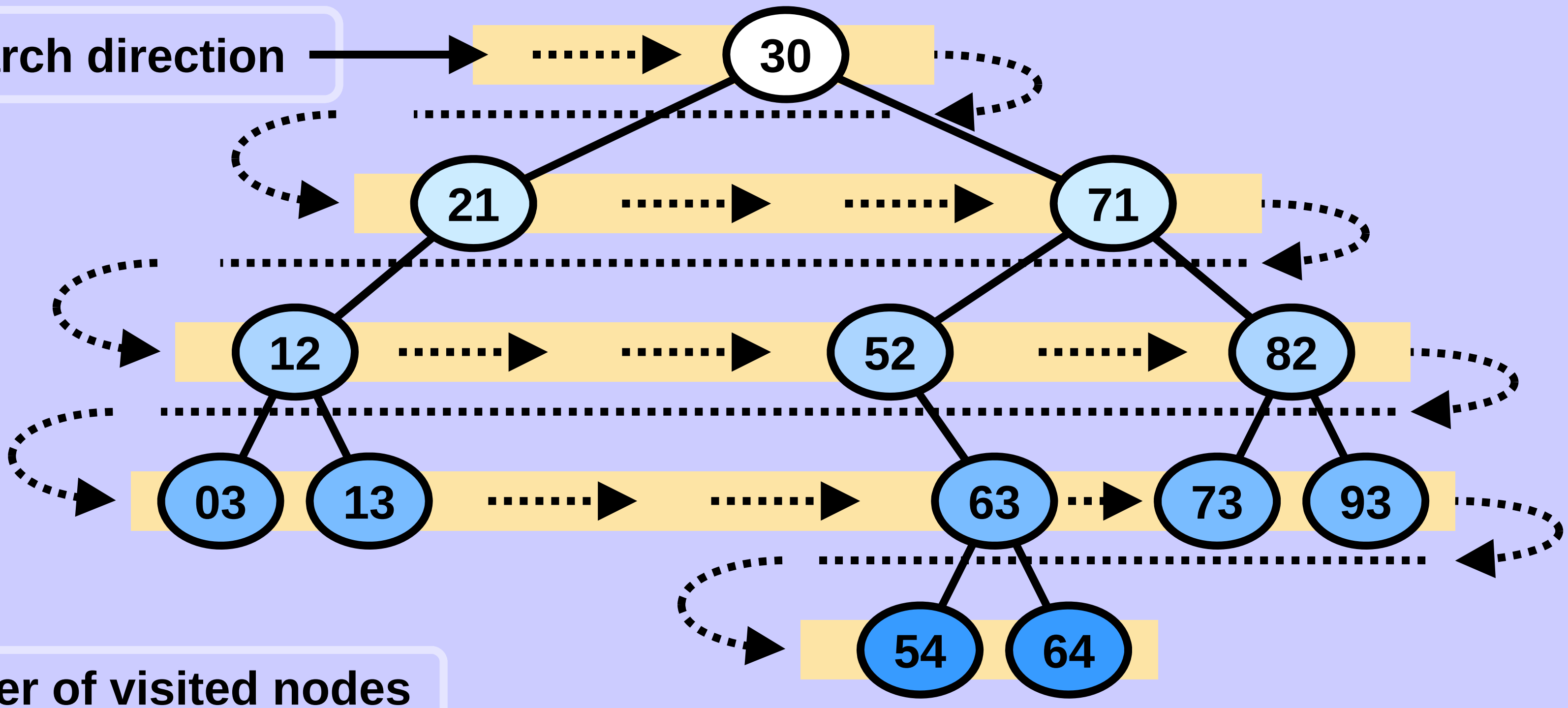
Tail index points to the first free position behind the last queue element.
Front index points to the first position occupied by a queue element.
When both indices point to the same position the queue is empty.

```
class Queue {
  Node q [];
  int size;
  int front;
  int tail;
           //constructor:
 Queue(int qsize) {
   size = qsize;
   q = new Node[size];
   front = 0;
   tail = 0;
   }

 boolean Empty() {
   return (tail==front);
 }
```

```
void Enqueue(Node node) {
  if ((tail+1 == front) ||
      (tail-front == size-1))
      ... // queue full, fix it

  q[tail++] = node;
  if (tail==size) tail=0;
}

Node Dequeue() {
  Node n = q[front++];
  if (front==size) front=0;
  return n;
}

} // end of Queue
```

Breadth-first search (BFS) of a tree

Search direction



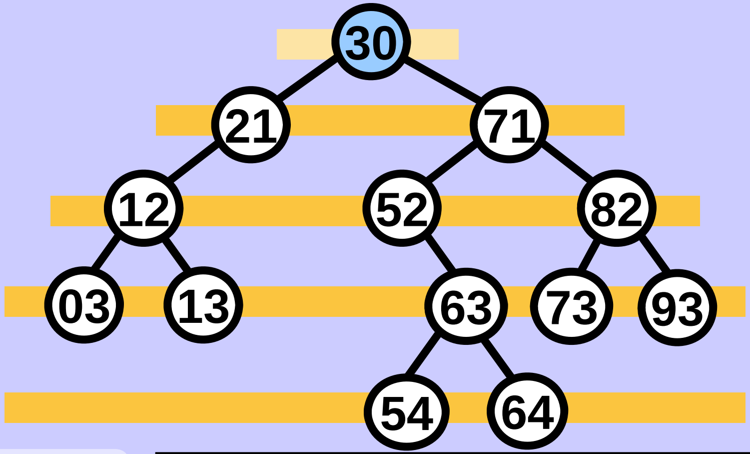
Order of visited nodes

30	21	71	12	52	82	03	13	63	73	93	54	64

Nor the tree structure nor the recursion support this approach directly.

Breadth-first search (BFS) of a tree

Initialization



Output

Create an empty queue.

Enqueue the tree root.

Front

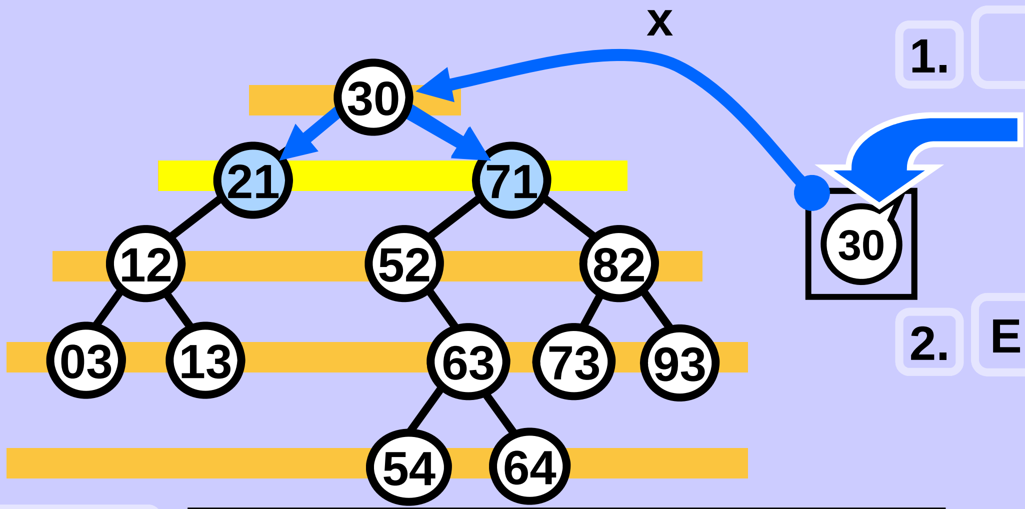
Tail

Main loop

While the queue is not empty do:

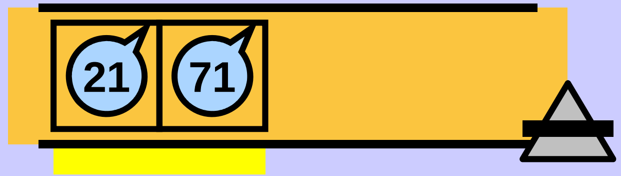
1. Remove the first element from the queue and process it.
2. Enqueue the children of removed element.

Breadth-first search (BFS) of a tree

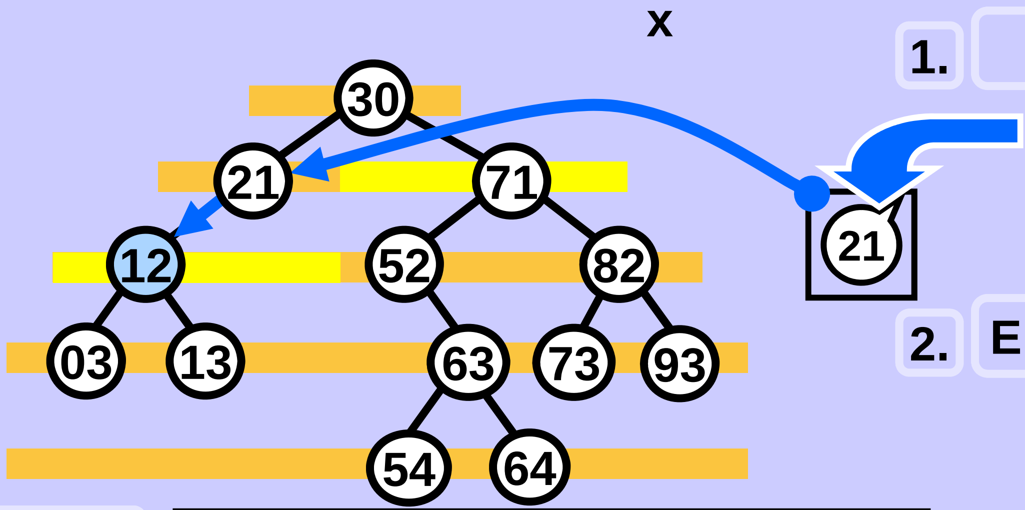


1. x = Dequeue(), print (x.key).

2. Enqeue(x.left), Enqueue(x.right). *)

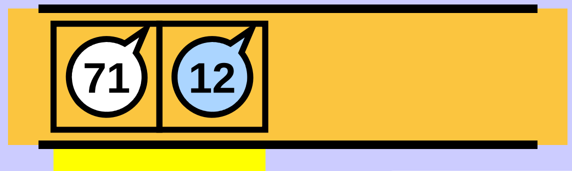


Output | 30



1. x = Dequeue(), print (x.key).

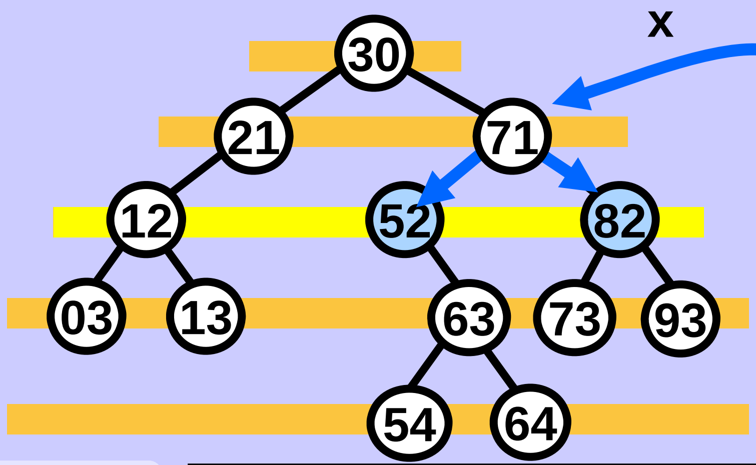
2. Enqeue(x.left), Enqueue(x.right). *)



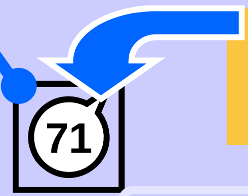
Output | 30 21

*) if exists

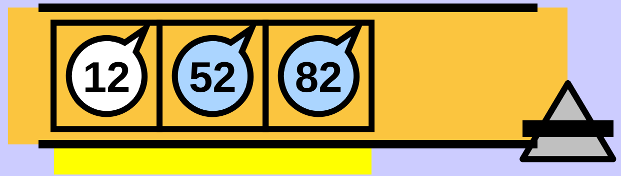
Breadth-first search (BFS) of a tree



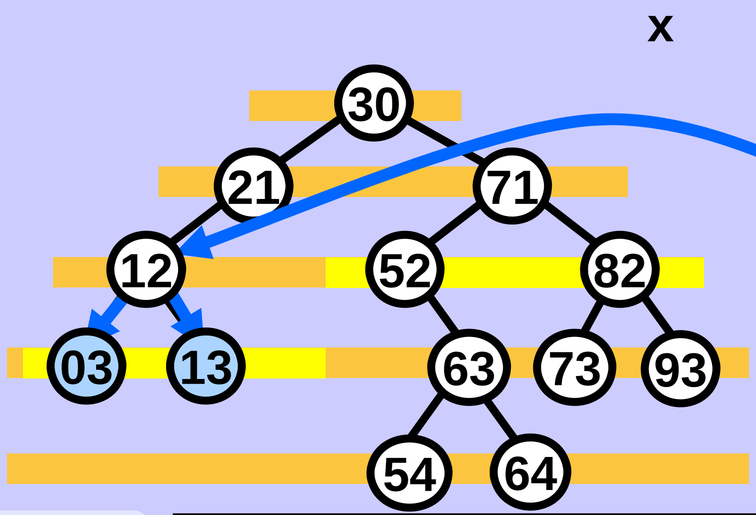
1. x = Dequeue(), print (x.key).



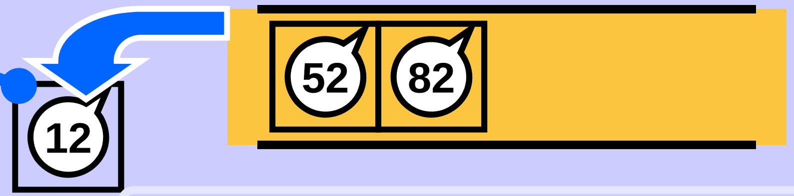
2. Enqeue(x.left), Enqueue(x.right). *)



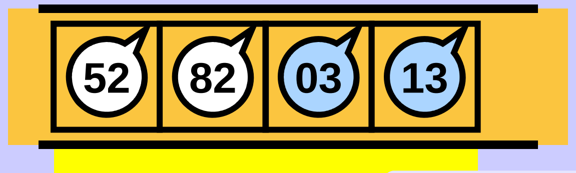
Output 30 21 71



1. x = Dequeue(), print (x.key).



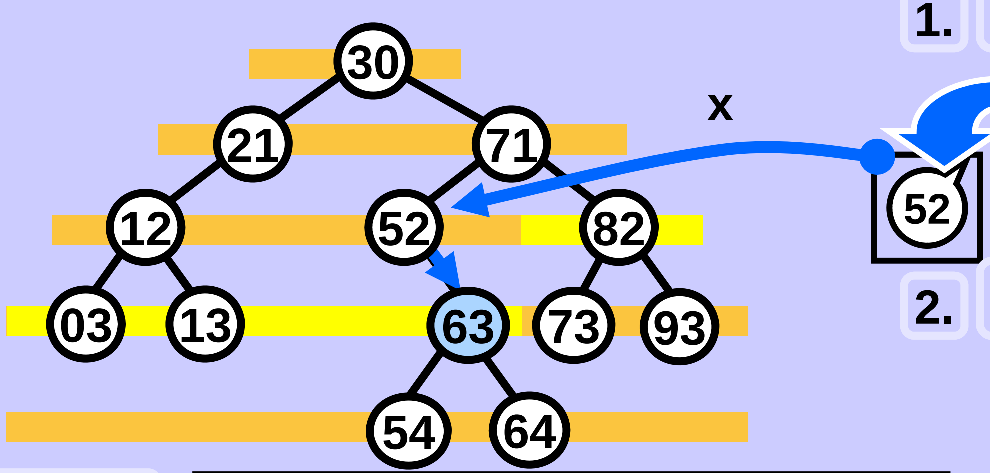
2. Enqeue(x.left), Enqueue(x.right). *)



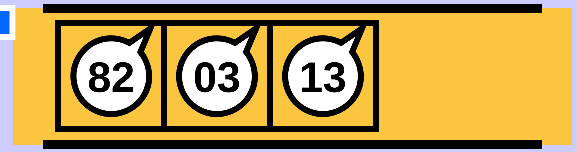
Output 30 21 71 12

*) if exists

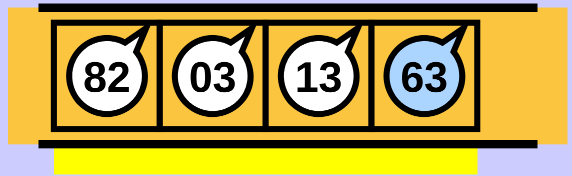
Breadth-first search (BFS) of a tree



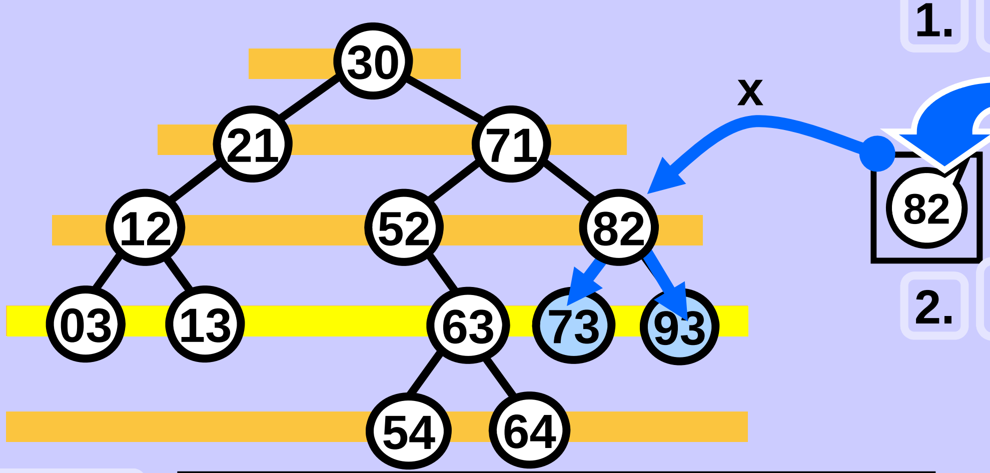
1. x = Dequeue(), print (x.key).



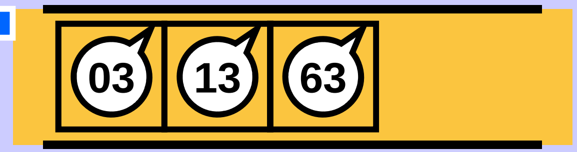
2. Enqeue(x.left), Enqueue(x.right). *)



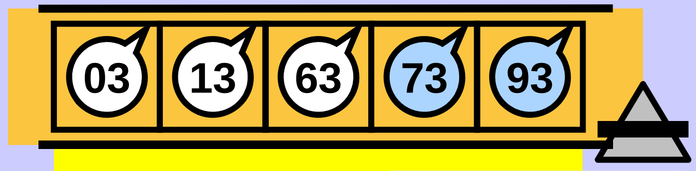
Output: 30 21 71 12 52



1. x = Dequeue(), print (x.key).



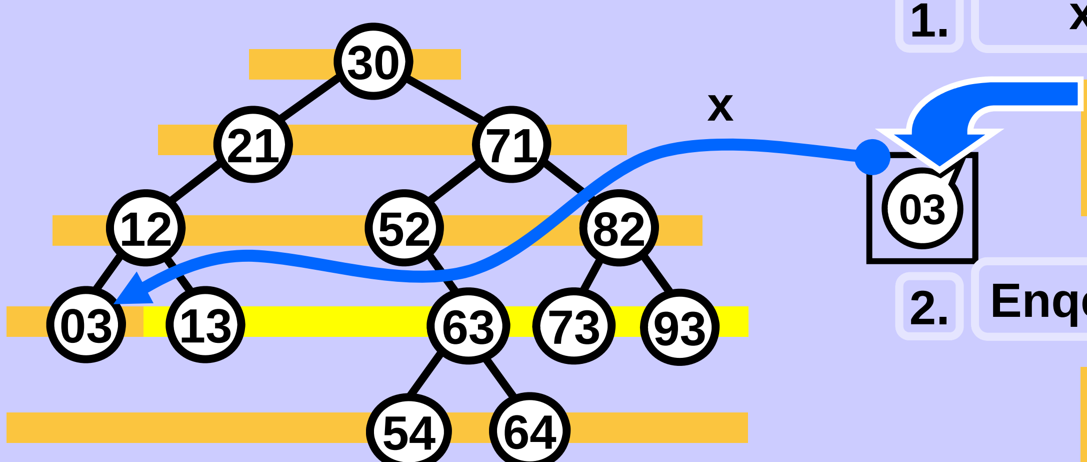
2. Enqeue(x.left), Enqueue(x.right). *)



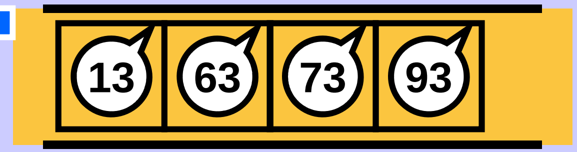
Output: 30 21 71 12 52 82

*) if exists

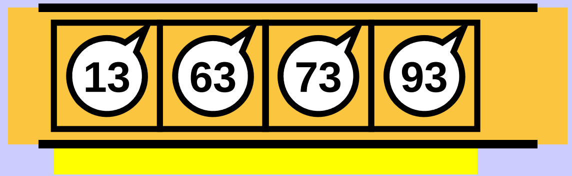
Breadth-first search (BFS) of a tree



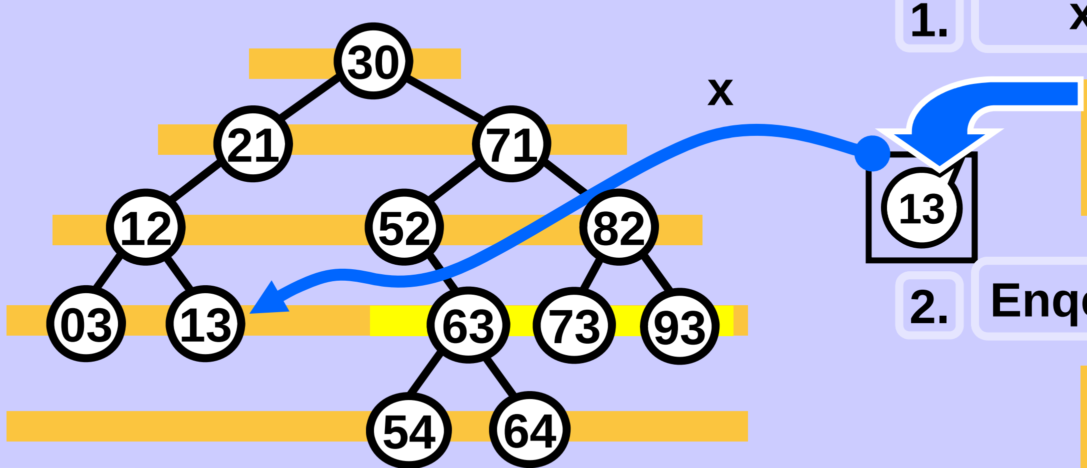
1. x = Dequeue(), print (x.key).



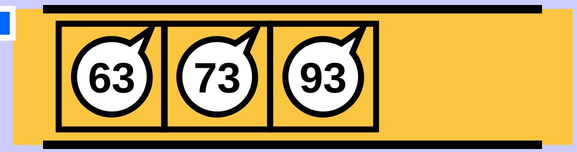
2. Enqeue(x.left), Enqueue(x.right). *)



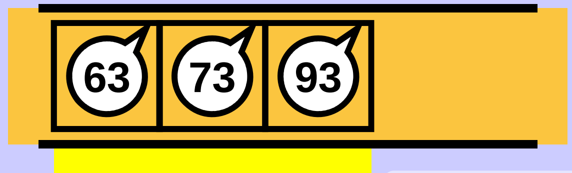
Output 30 21 71 12 52 82 03



1. x = Dequeue(), print (x.key).



2. Enqeue(x.left), Enqueue(x.right). *)



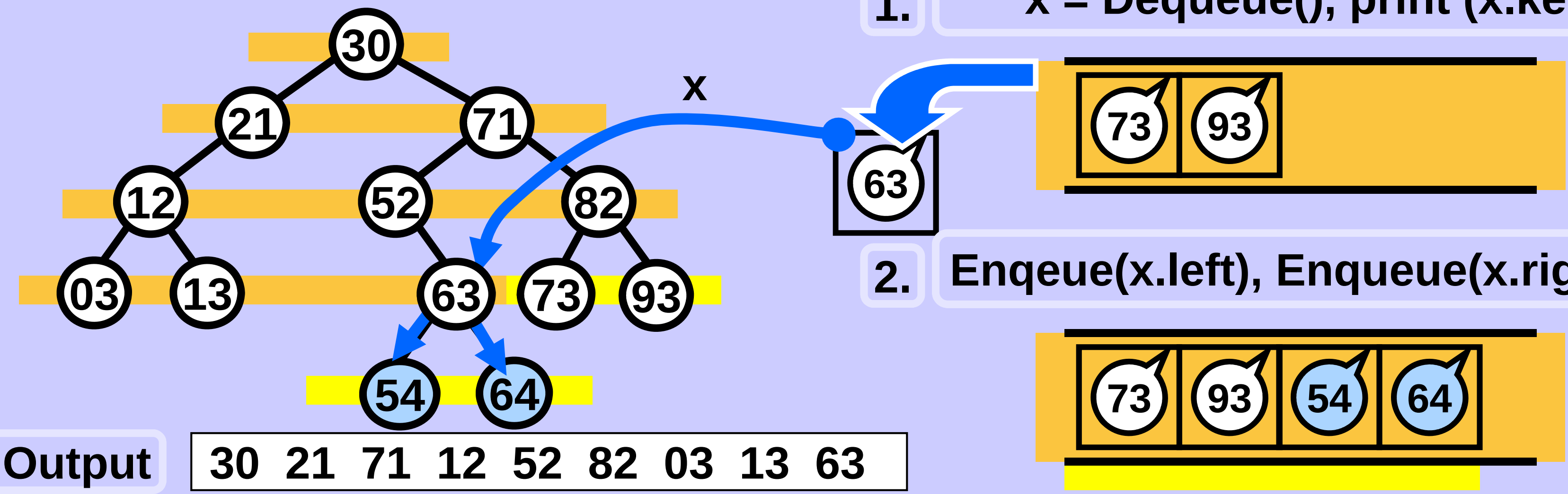
Output 30 21 71 12 52 82 03 13

*) if exists

Breadth-first search (BFS) of a tree

1. x = Dequeue(), print (x.key).

2. Enqeue(x.left), Enqueue(x.right). *)

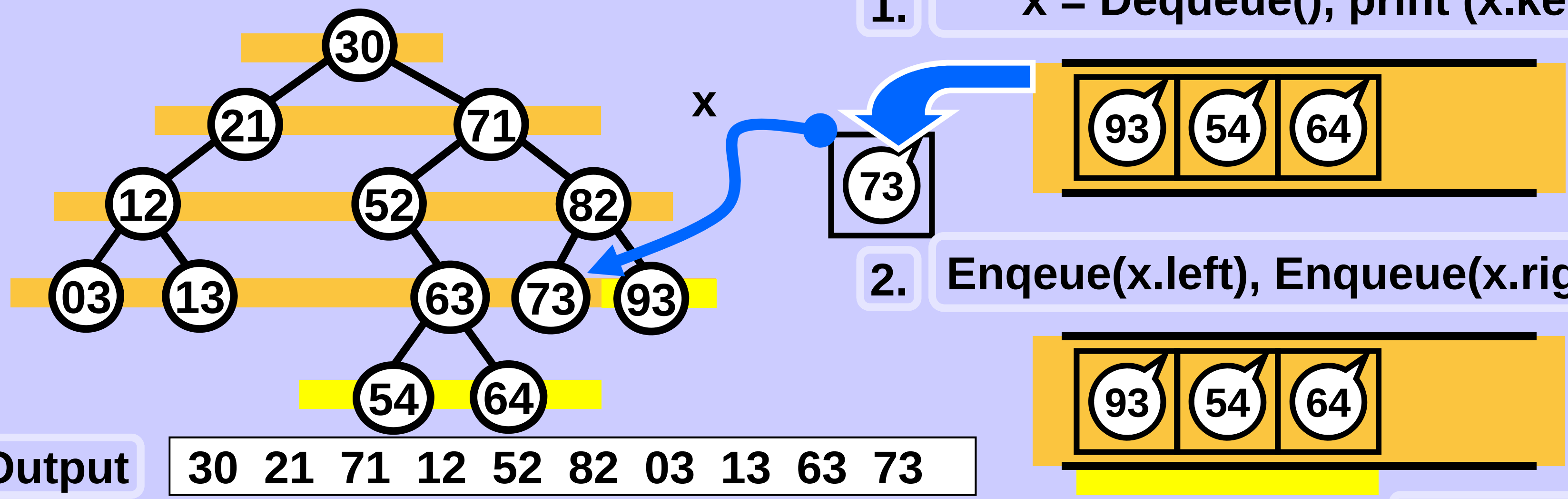


Output | 30 21 71 12 52 82 03 13 63

---

1. x = Dequeue(), print (x.key).

2. Enqeue(x.left), Enqueue(x.right). *)



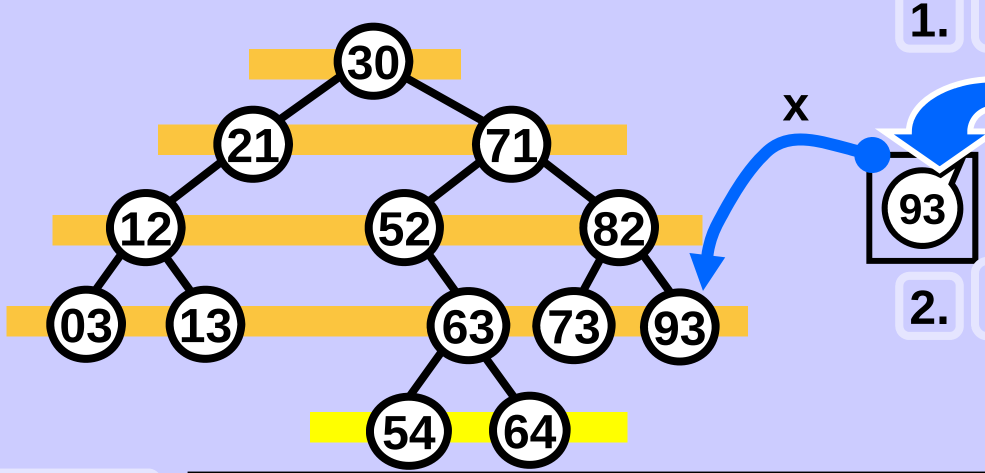
Output | 30 21 71 12 52 82 03 13 63 73

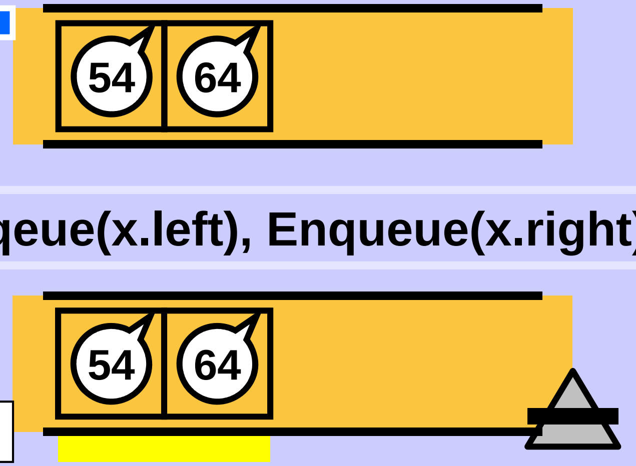
*) if exists

Breadth-first search (BFS) of a tree

1. x = Dequeue(), print (x.key).

2. Enqeue(x.left), Enqueue(x.right). *)

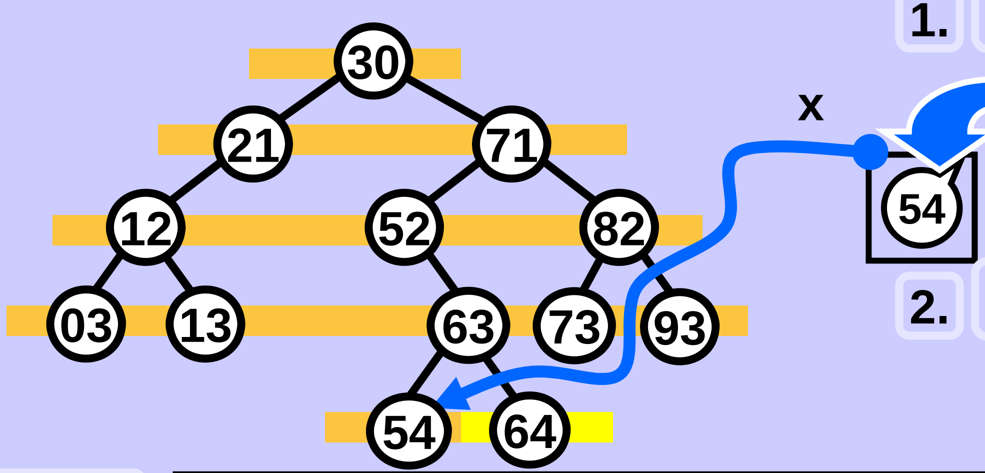




Output 30 21 71 12 52 82 03 13 63 73 93

1. x = Dequeue(), print (x.key).

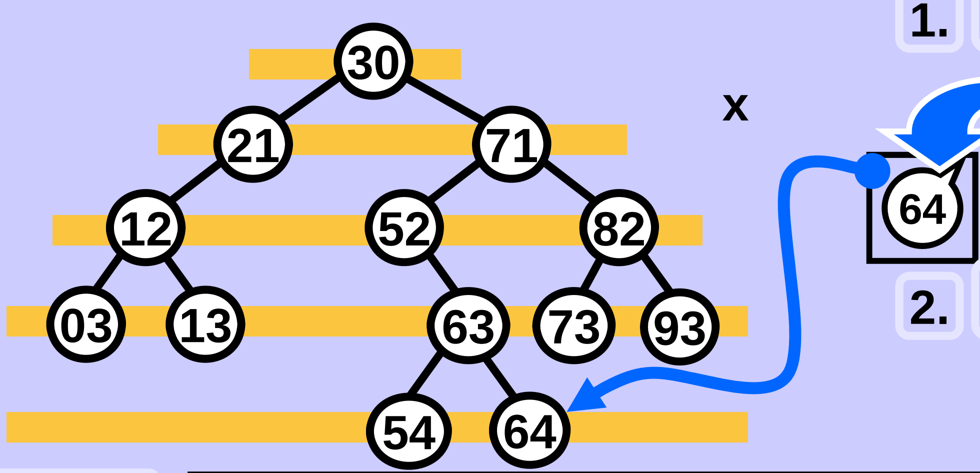
2. Enqeue(x.left), Enqueue(x.right). *)



Output 30 21 71 12 52 82 03 13 63 73 93 54

*) if exists

Breadth-first search (BFS) of a tree



1. x = Dequeue(), print (x.key).

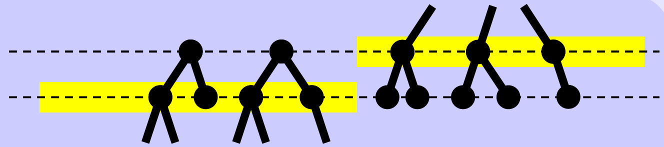
2. Enqeue(x.left), Enqueue(x.right). *)

Output 30 21 71 12 52 82 03 13 63 73 93 54 64

*) if exists.

The queue is empty,
BFS is complete.

An unempty queue always contains exactly
-- some (or all) nodes of one level and
-- all children of those nodes of this level which have already left the queue.



Sometimes the queue contains just nodes of one level. See above:

Breadth-first search (BFS) of a tree

```
void binaryTreeBFS (Node node) {
    if (node == null) return;
    Queue q = new Queue();          // init
    q.Enqueue(node);                // root into queue
    while (!q.Empty()) {
        node = q.Dequeue();
        print(node.key);            // process node
        if (node.left  != null) q.Enqueue(node.left);
        if (node.right != null) q.Enqueue(node.right);
    }
}
```