Data Structure

魏恒峰

hfwei@nju.edu.cn

2018 年 02 月 27 日

Data Structure

(i) What is (and why) data structure?

(ii) Common (simple) data structures:

  (1) Variable, Pointer

  (2) Linear data structures:
  - Array, List (Singly-linked List, Doubly-linked List, Circular List)
  - Stack, Queue (Deque, Priority Queue)

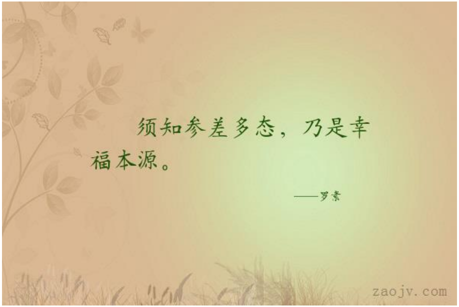
  (3) Trees
  - Binary Search Tree (BST)
  - ...

  (4) Hashes

  (5) Graphs

  (6) ...

Why are there so many data structures?



Data Strcture *vs.* Data Type

Data type: data + operations

Data structure: data type + structure

A data structure is an implementation of an abstract data type (ADT).

Example: Sequence of Data

OP: SEARCH, INSERT, DELETE

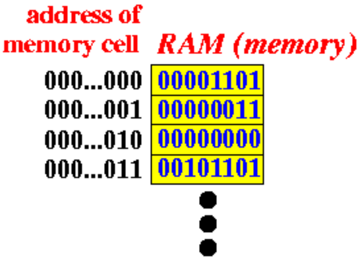
Array *vs.* List

Variable and Pointer

Memory

Definition (Memory (K&R))

The memory is organized as a collection of consecutively addressed cells that may be manipulated individually or in contiguous groups.



Variable

```
int x;
```

Pointer

Definition (Pointer (K&R))

A pointer is a variable that contains the address of a variable.

```
int a = 0;
int *p = &a;
```

Definition (Pointer in Memory (K&R))

A pointer is a group of cells (often two or four) that can hold an address.

```c
swap(a, b);

void swap(int a, int b) {
  int temp = a;
  a = b;
  b = tmp;
}
```

Pointer arguments enable a function to access and change objects in the function that called it. — *K&R*

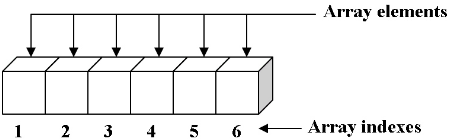
```c
swap(&a, &b);

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = tmp;
}
```

Array

Array

Array: A sequence of contiguously stored elements.

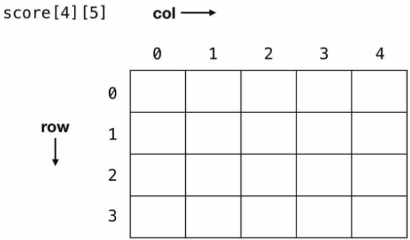


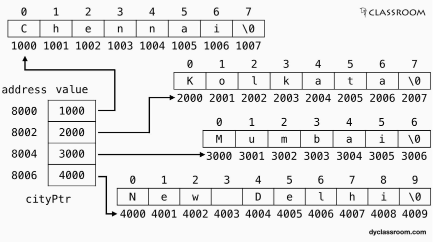
One-dimensional array with six elements

```
vector<int> array {1,5,7,9,10};

array[1] = 3;  // offset
array.insert(pos, val);  // moving elements
array.erase(pos)  // moving elements
```

2D Array



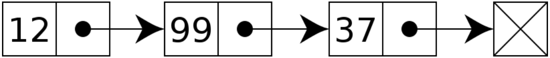



```
vector<int> array {1,5,7,9,10};
matrix<int>

array[1] = 3;  // offset
array.insert(pos, val);  // moving elements
array.erase(pos)  // moving elements
```

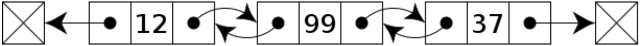
List

Singly-linked List



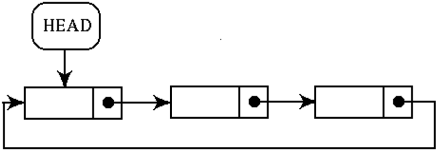
SEARCH, INSERT, DELETE

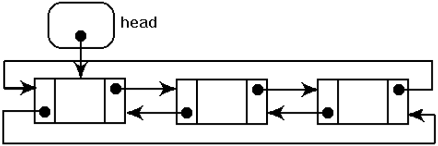
Doubly-linked List



SEARCH, INSERT, DELETE

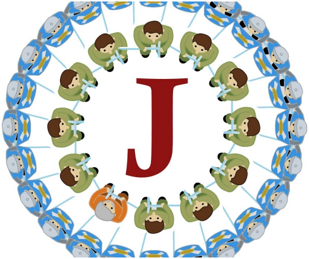
Circular Linked List





Doubly Linked Circular list

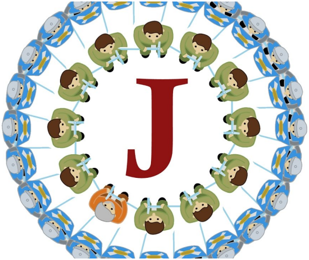
The Josephus Puzzle



The Josephus Puzzle

$$J(n) = ?$$

The Josephus Programming Task



Input: n

Output: $J(n)$

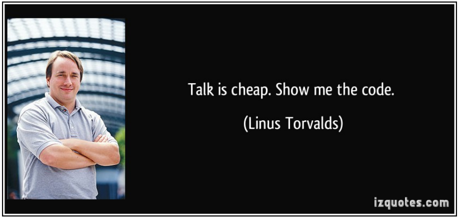
Input: n

Output: $J(1), J(2), \cdots, J(n)$

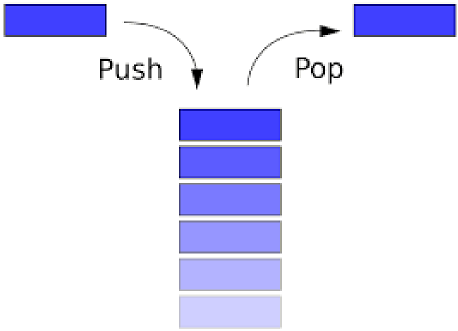
The Josephus Programming Practice

Q: What data structure do you use?

WHY?



Stack



PUSH, POP, EMPTY, PEEK

Brackets Matching

Brackets Matching

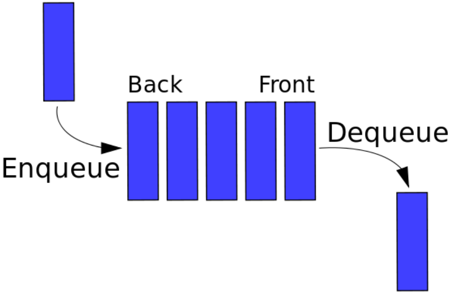
判断给定字符串中的括号是否匹配。

输入格式:

- 首行是一个正整数 (记为 n)。
- 接着是 n 行字符串。
  - 每个字符串最多含有 (, [, {,),], } 六种不同字符。
  - 字符串可以为空。规定空字符串是 “括号匹配的”。

输出格式: 如果某行字符串中的括号是匹配的，则对应行输出 1，否则输出 0。

Queue



Enqueue, Dequeue

Stutter

Stutter

- Given a queue of integers
- To replace every element with two copies of itself

$$\{1, 2, 3\} \to \{1, 1, 2, 2, 3, 3\}$$

Binary Search Tree

Binary Search Tree

Definition (BST)

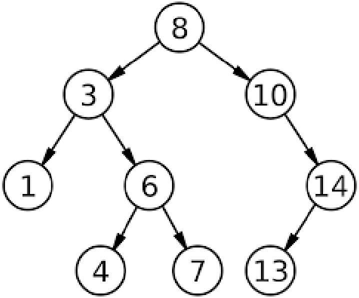
A binary search tree is a rooted binary tree,

1. each internal node stores a key/value
2. each internal node has two distinguished subtrees

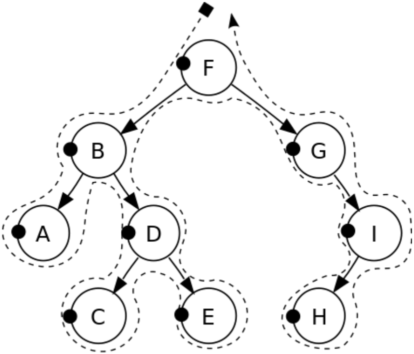
   left subtree the key in each node must be $\geq$ any key stored in the left subtree

   right subtree the key in each node must be $\leq$ any key stored in the right subtree

BST

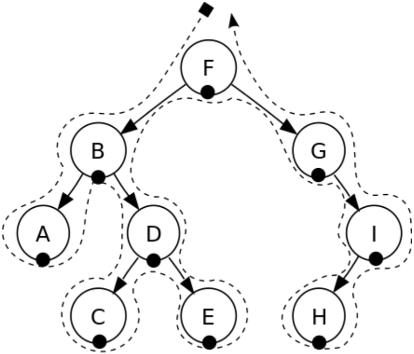


Preorder Traversal



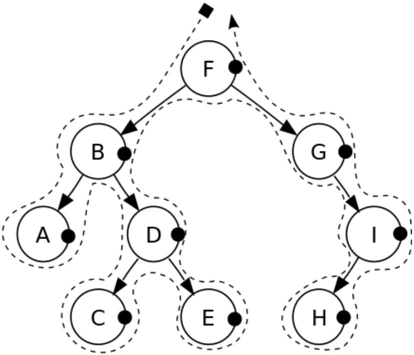
1. Check if the current node is a leaf.
2. Display the data part of the root (or current node).
3. Traverse the left subtree by recursively calling the preorder function.
4. Traverse the right subtree by recursively calling the preorder function.

Inorder Traversal



1. Check if the current node is a leaf.
2. Traverse the left subtree by recursively calling the inorder function.
3. Display the data part of the root (or current node).
4. Traverse the right subtree by recursively calling the inorder function.

Postorder Traversal



1. Check if the current node is a leaf.
2. Traverse the left subtree by recursively calling the postorder function.
3. Traverse the right subtree by recursively calling the postorder function.
4. Display the data part of the root (or current node).

DH 2.16: Treesort

(i) Construct an algorithm that transforms a given list of integers into a binary search tree.

```
procedure put-x-into-BST (t):
  ... call put-x-into-BST (t′s left subtree)
  ... call put-x-into-BST (t′s right subtree)
end procedure
```

DH 2.16: Treesort

(i) Construct an algorithm that transforms a given list of integers into a binary search tree.

```
Node:
  int val = NIL,
  Node left = NULL,
  Node right = NULL
```

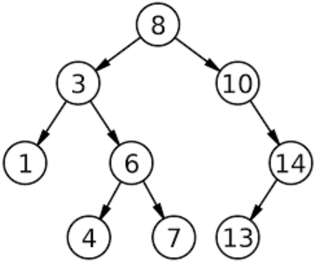
```
buildBST(int eles[]):
  Node root(eles[0])

  foreach e ∈ eles[1..]:
    insert(root, e)
```

```
insert(Node T, int e):
  if (e < T.val)
    if (T.left == NULL)
      T.left = new Node(e)
    else
      insert(T.left, e)
  else  // e >= T.val
    if (T.right == NULL)
      T.right = new Node(e)
    else
      insert(T.right, e)
```

DH 2.16: Treesort

(ii) right;   val;   left




Thank
You!