

CS150

初赛专题 集训



CS150 初赛专题集训公布资料的固定网站
请每次课前自行将资料下载到电脑

<https://pan.baidu.com/s/12ZsJgSE-p17Vxl-ObMe7Mg>

快快编程地址

<http://120.132.18.213:9062>

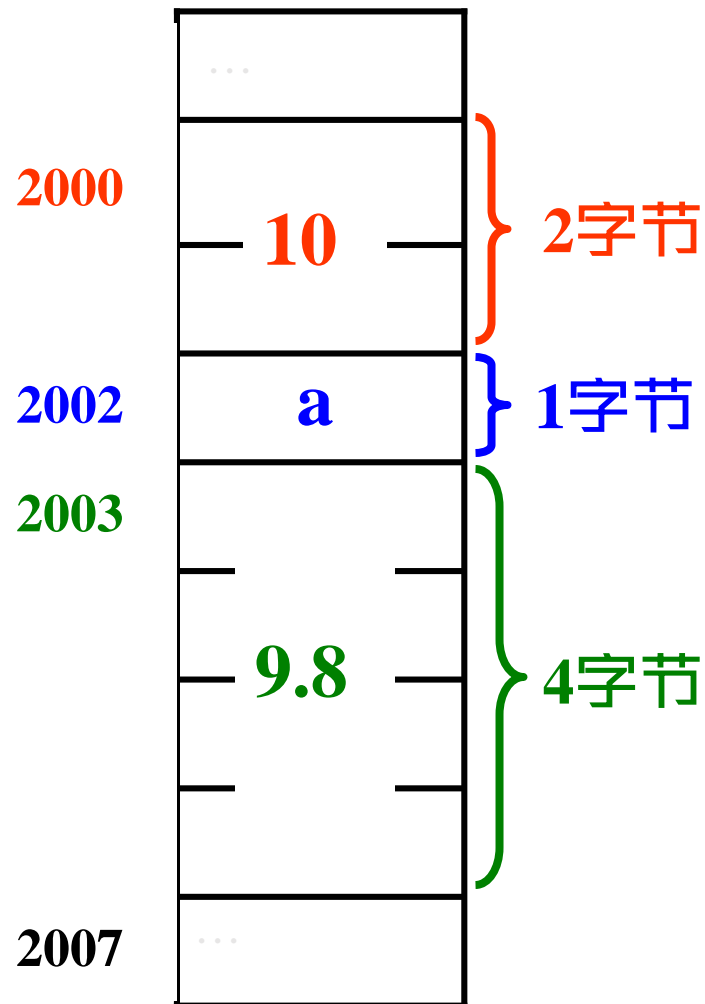
请登陆网站提交作业

线性表、栈、队列

内存与地址

- 存储单元：存放一个字节数据的存储器。
- 存储单元的内容：存储单元内的数据。
- 内存地址：存储单元的编号。

```
short    x ;  
char     y ;  
float    z ;  
  
x = 10 ;  
y = 'a' ;  
z = 9.8 ;
```

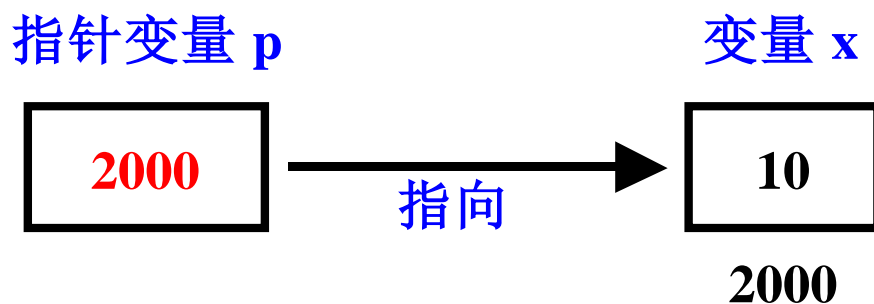


指针

定义：指针就是变量的内存地址

指针变量：存放变量内存地址的变量

变量的指针：变量的地址



当把某个变量x的地址存入指针变量p后，称为指针变量p的指针指向变量x。

指针

指针变量的定义

类型说明符 *变量名1,;

int *p1, *p2;

取地址运算符 &

取变量的地址

```
int i, *p1 ;  
p1 = &i ;
```

指针运算符 *

取指针所指向的变量的值

```
int i=100, *p1;  
p1 = &i;
```

p1 = &i; //将i的地址赋给指针变量p1

p2 = &chr; //将chr的地址赋给指针变量p2

*p1 = 10; //把10存入pi所指的地址 (&i) 中, 等价于i = 10

结构体的引用

1. 结构变量名.成员名
2. (*指向结构的指针).成员名
3. 指向结构的指针->成员名

```
struct man {  
    string name;  
    int age;  
    struct {  
        int year;  
        int month;  
        int day;  
    } birthday;  
};  
  
man m,*p=&m;  
m.name="San Zhang";  
p->birthday.month = 8;  
(*p).age = 22;
```

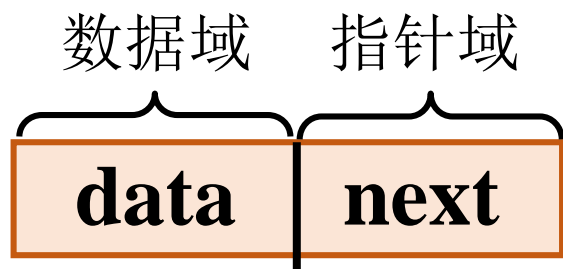
线性表

顺序表	链表
一个表必须用一组连续的内存地址存储	内存地址可以是连续的也可以是不连续的
插入和删除元素难度大	插入和删除元素简单（不需移动元素，只需修改头尾指针即可）
存取数据快（只要确定了起始位置，线性表中任一数据元素可随机存取）	存取数据慢

单向链表

单链表是由若干结点构成的，单链表的结点只有一个指针域

结点结构：

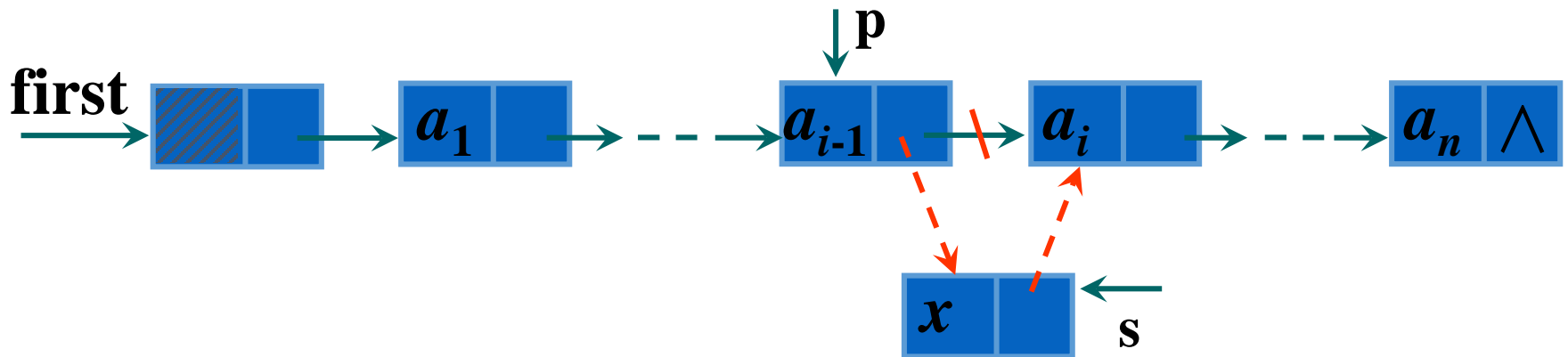


```
struct Node {  
    int data;  
    Node *next;  
};
```

data: 存储数据元素

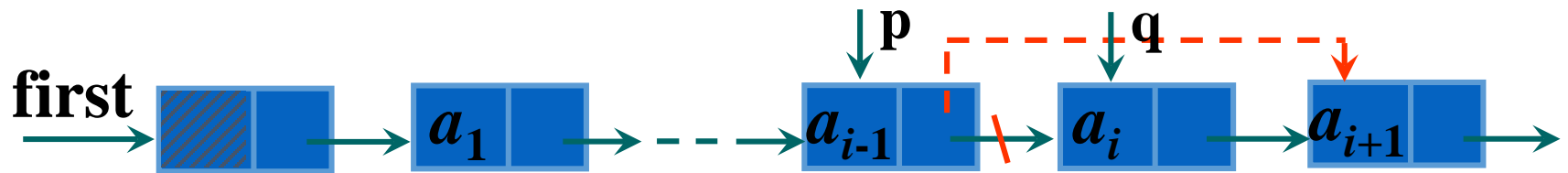
next: 存储指向后继结点的地址

单向链表插入



```
s->data=x;  
s->next=p->next;  
p->next=s;
```

单向链表删除



```
q=p->next;  
x=q->data;  
p->next=q->next;  
delete q;
```

双向链表

双链表：在单链表的每个结点中再设置一个指向其前驱结点的指针域。

结点结构：

prior

data

next

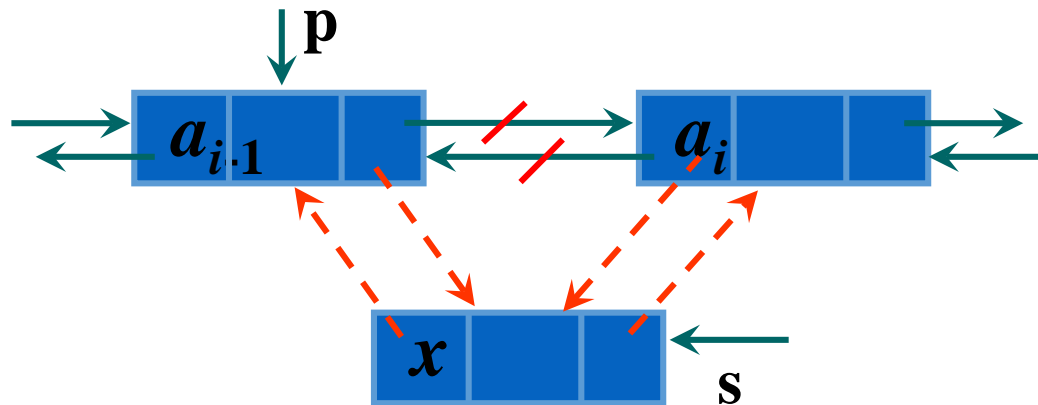
```
struct Du1Node
{
    int data;
    Du1Node *prior, *next;
};
```

data: 数据域，存储数据元素；

prior: 指针域，存储该结点的前趋结点地址；

next: 指针域，存储该结点的后继结点地址。

双向链表插入



```
s->prior=p;
```

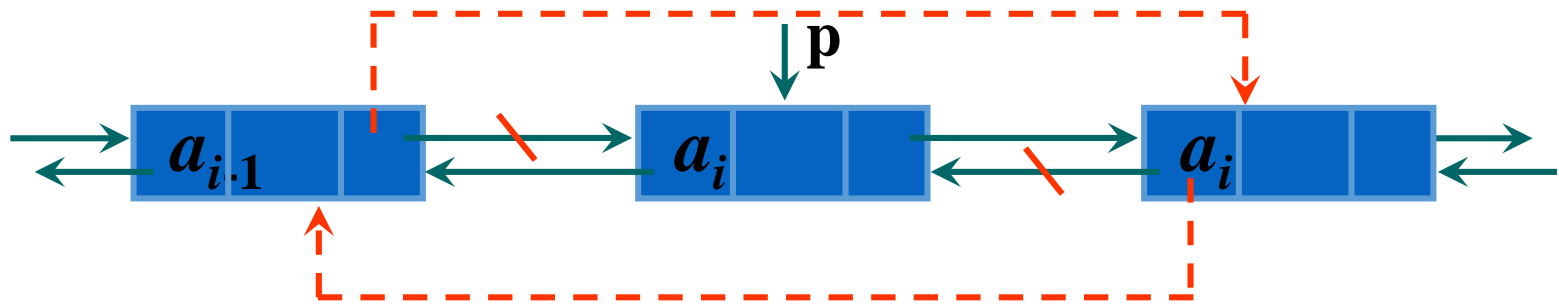
```
s->next=p->next;
```

```
p->next->prior=s;
```

```
p->next=s;
```

注意指针修改的相对顺序

双向链表删除



$(p \rightarrow \text{prior}) \rightarrow \text{next} = p \rightarrow \text{next};$

$(p \rightarrow \text{next}) \rightarrow \text{prior} = p \rightarrow \text{prior};$

栈

栈是限制在表的一端进行插入和删除操作的数据结构

所有操作都在栈顶位置

出栈

入栈

术语

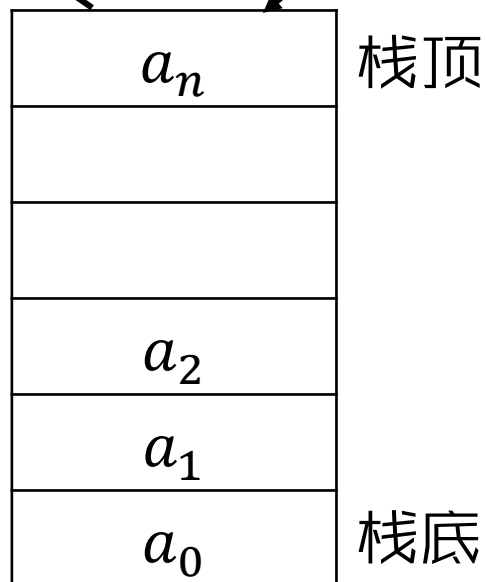
栈顶：允许进行插入、删除操作的一端，又称为表尾

栈底：是固定端

空栈：没有元素时栈为空

入栈：向栈顶压入元素

出栈：从栈顶弹出元素

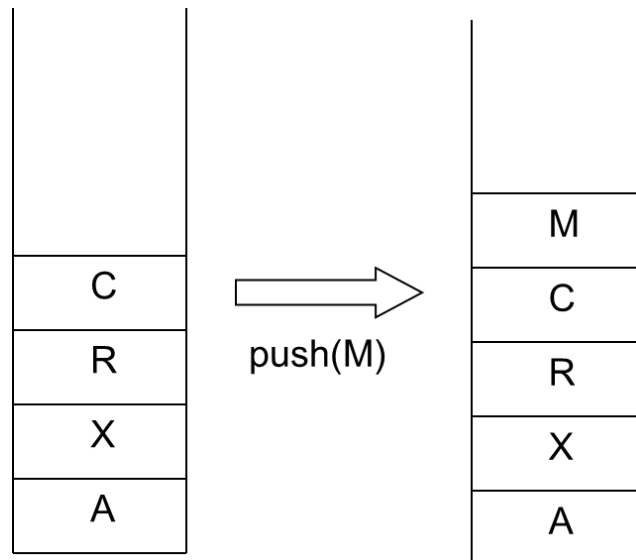


后进先出

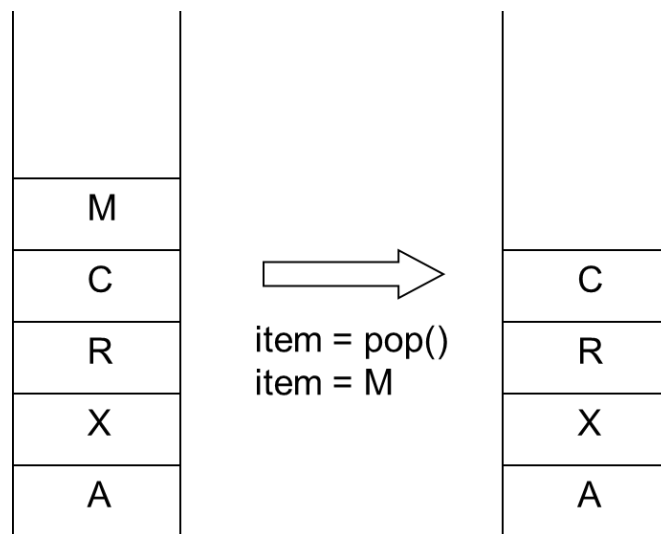
LIFO

栈的基本操作

入栈(push), 向栈顶压入元素



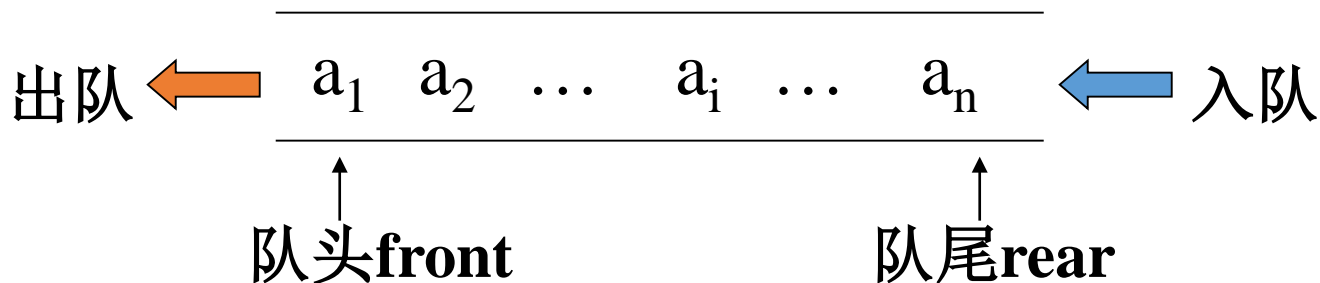
出栈(pop), 从栈顶弹出元素



队列

队列(Queue)的定义

队列是仅限定在表尾进行插入和表头进行删除操作的线性表。



队头——队列的表头，即只允许删除的一端。

队尾——队列的表尾，即只允许插入的一端。

入队——向队尾插入元素。

出队——从队头删除元素。

特点：先进先出（FIFO）

练习

线性表若采用链表存贮结构,要求内存中可用存贮单元地址
()

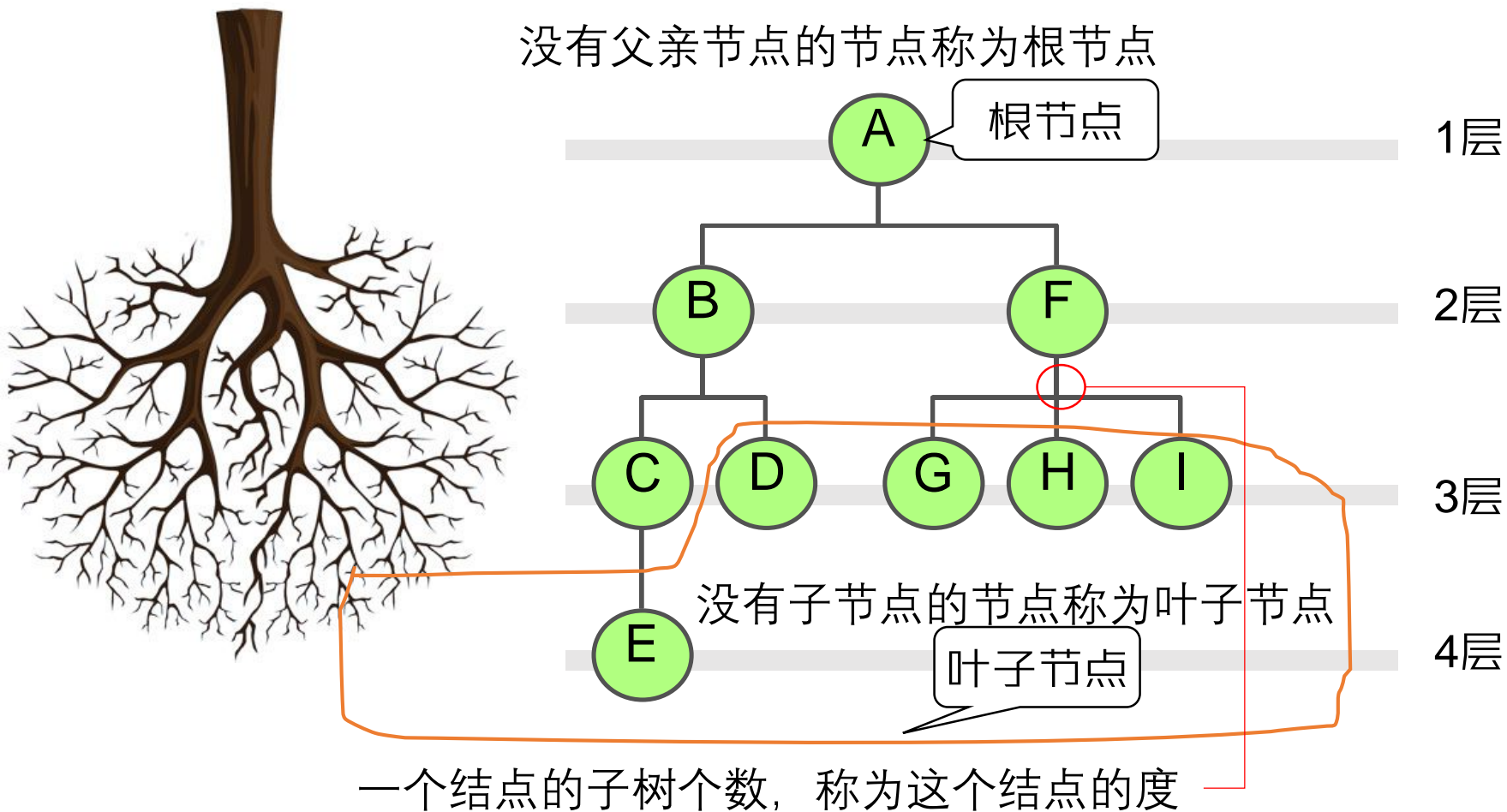
- A.必须连续 B.部分地址必须连续
- C.一定不连续 D.连续不连续均可

下列叙述中,正确的是 ()

- A.线性表的线性存贮结构优于链表存贮结构
- B.队列的操作方式是先进后出
- C.栈的操作方式是先进先出
- D.二维数组是指它的每个数据元素为一个线性表的线性表

二叉树

树

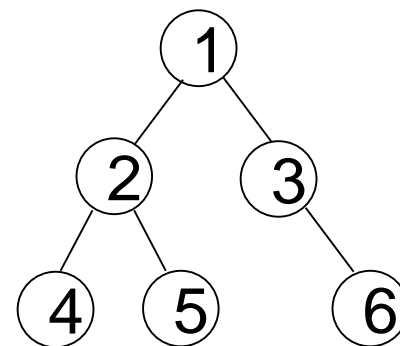
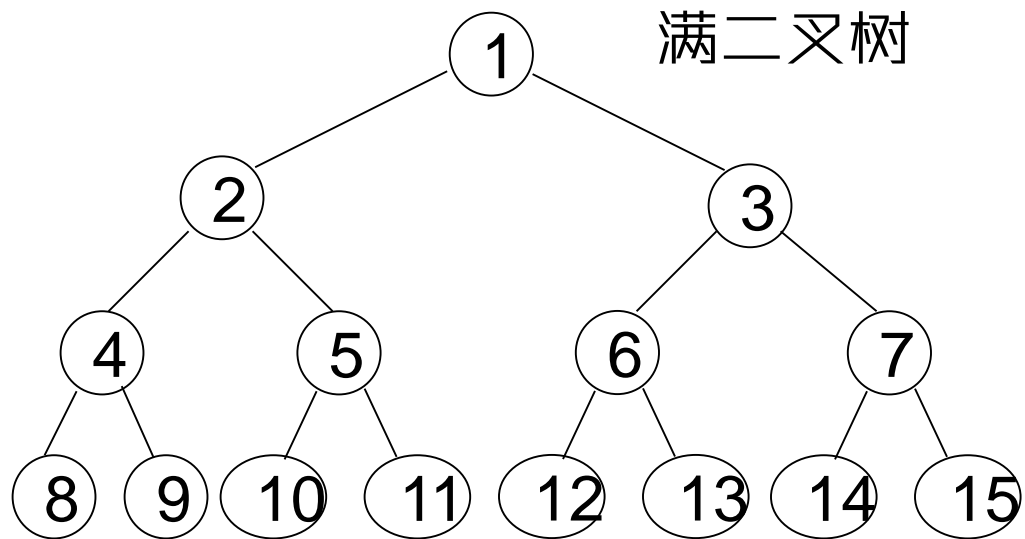


概念

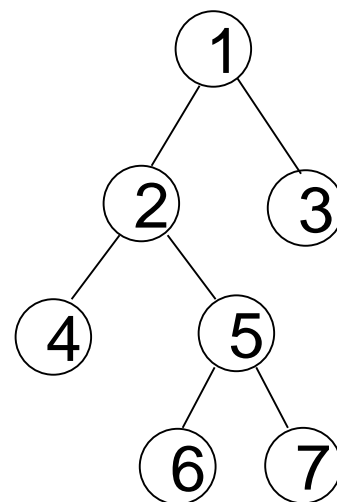
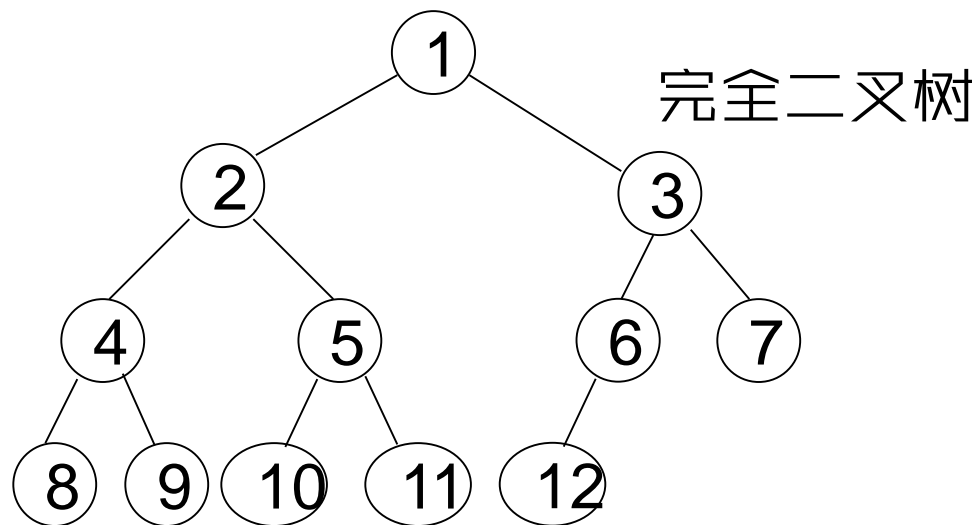
- 节点、层、深度、根、子树、叶子、度

二叉树

二叉树类型

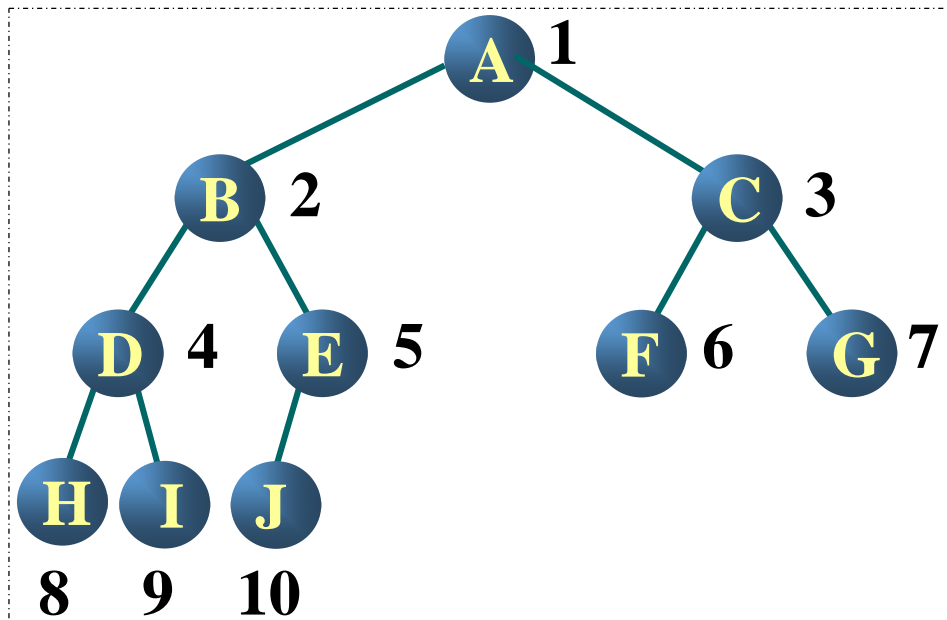


非完全二叉树



完全二叉树特点

1. 叶子结点只能出现在最下两层且最下层的叶子结点都集中在二叉树的左面;
2. 完全二叉树中如果有度为1的结点, 只可能有一个, 且该结点只有左孩子。
3. 深度为 k 的完全二叉树在 $k-1$ 层上一定是满二叉树。
4. 在同样结点个数的二叉树中, 完全二叉树的深度最小。



二叉树性质

性质1： 在非空二叉树中，第 i 层上至多有 2^{i-1} 个结点 ($i \geq 1$)。

二叉树第10层的结点数的最大数目为 ()。

- A. 10 B. 100 C. 512 D. 1024

性质2： 深度为 d 的二叉树至多有 $2^d - 1$ 个结点 ($d \geq 1$)，最少有 d 个结点。

一棵深度为 K 的满二叉树有 () 个结点。

- A. $2^K - 1$ B. 2^K C. $2 * K$ D. $2 * K - 1$

二叉树性质

性质3: 对任何一棵二叉树, 若其叶子结点数为 n_0 , 度为2的结点数为 n_2 , 则 $n_0=n_2+1$ 。

对任何一棵二叉树T, 设 n_0 、 n_1 、 n_2 分别是度数为0、1、2的顶点数, 则下列判断中正确的是()。

A. $n_0=n_2+1$ B. $n_1=n_0+1$ C. $n_2=n_1+1$ D. $n_2=n_0+1$

一棵完全二叉树上有1001个结点, 其中叶子结点的个数是()。

A. 250 B. 500 C. 254 D. 501

性质4: n 个结点的完全二叉树深度为: $\lfloor \log_2 n \rfloor + 1$ 。

其中: $\lfloor x \rfloor$ 表示不大于 x 的最大整数。

一棵 n 个节点的完全二叉树, 则该二叉树的高度 h 为()。

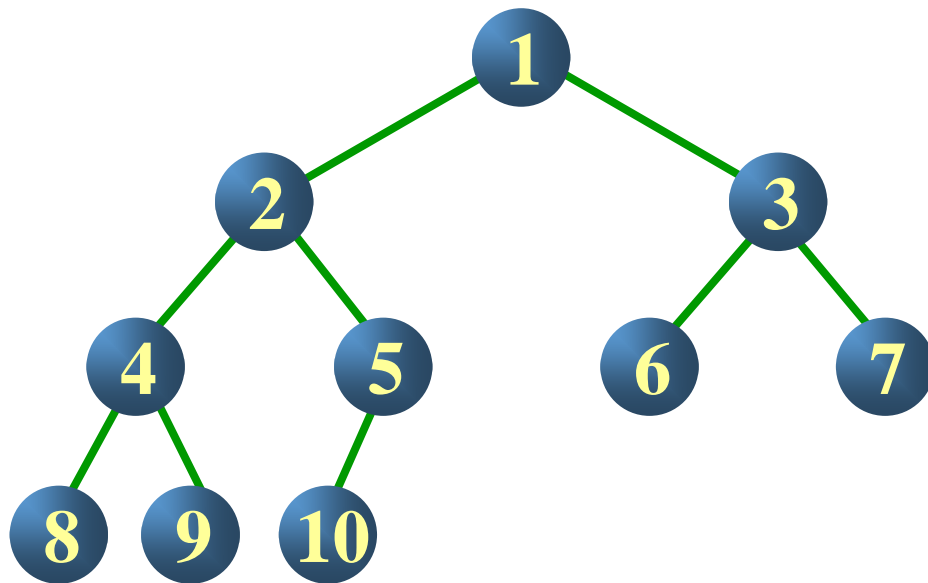
A. $n/2$ B. $\log(n)$ C. $\log(n)/2$ D. $\lfloor \log(n) \rfloor + 1$

二叉树性质

性质5:

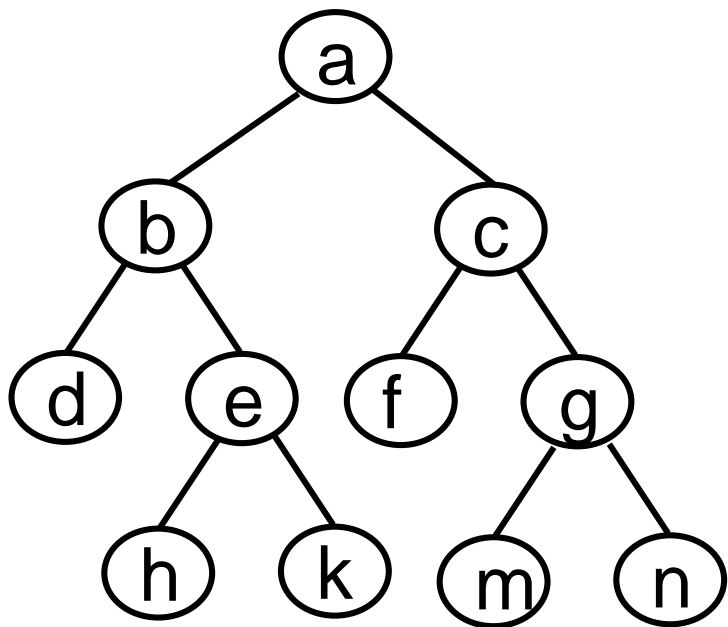
完全二叉树中的某结点编号为 i , 则

1. 若有左孩子, 则左孩编号为 $2i$
2. 若有右孩子, 则右孩子结点编号为 $2i+1$
3. 若有双亲, 则双亲结点编号为 $[i/2]$



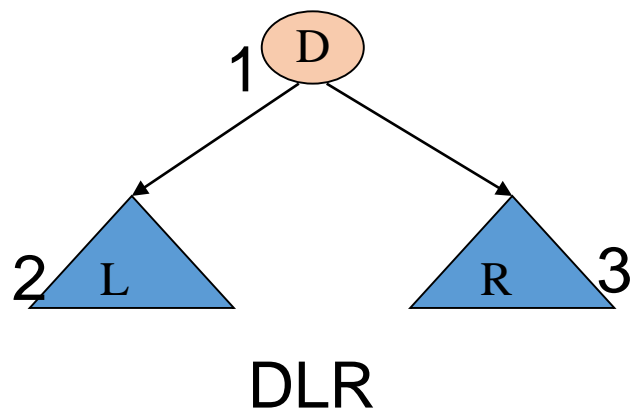
练习

- 一棵完全二叉树的结点总数为18，其叶结点数为()。
A. 7个 B. 8个 C. 9个 D. 10个
- 如果一棵二叉树有N个度为2的节点，M个度为1的节点，则该树的叶子个数为()。
A. $N+1$ B. $2 * N-1$ C. $N-1$ D. $M+N-1$



- 哪个是根结点？哪些是叶子结点？哪个是g的父亲节点？哪些是g的孩子？哪些是e的兄弟？哪些是f的兄弟？
- b和n的层次各是多少？树的深度是多少？以结点c为根的子树的深度是多少？

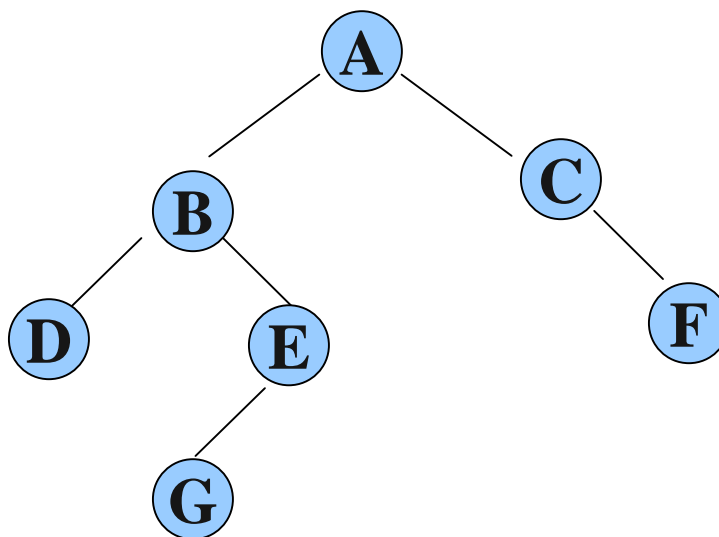
前序遍历



二叉树前序遍历算法

若二叉树为空，则遍历结束；否则

- (1) 访问根结点；
- (2) 前序遍历左子树；
- (3) 前序遍历右子树。



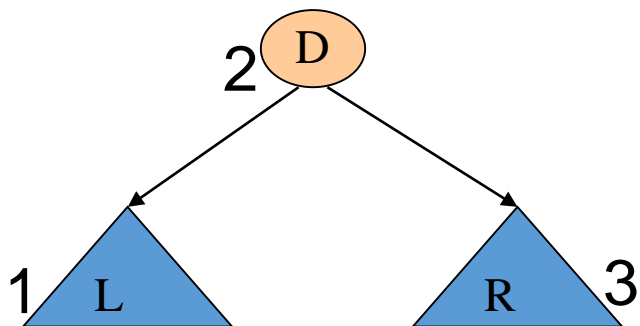
先序遍历序列: A B D E G C F

中序遍历

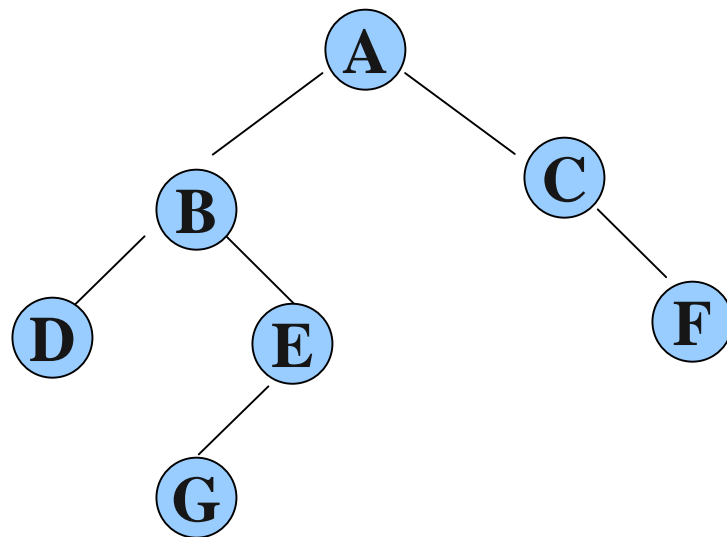
二叉树中序遍历算法

若二叉树为空，则遍历结束；否则

- (1) 中序遍历左子树；
- (2) 访问根结点；
- (3) 中序遍历右子树



LDR



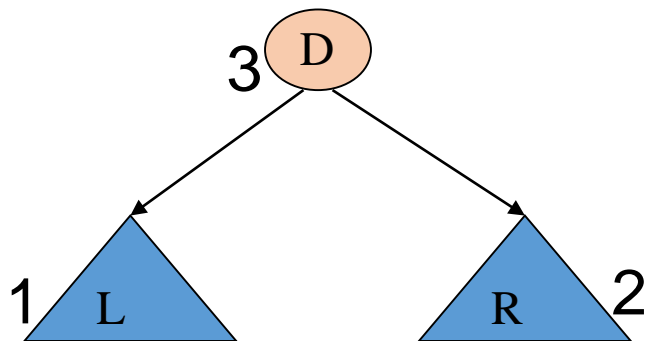
中序遍历序列: D B G E A C F

后序遍历

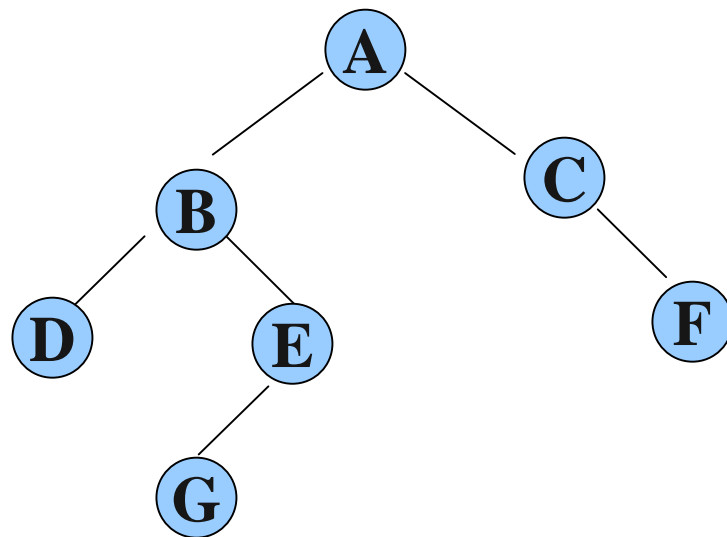
二叉树后序遍历算法

若二叉树为空，则遍历结束；否则

- (1) 后序遍历左子树；
- (2) 后序遍历右子树；
- (3) 访问根结点。



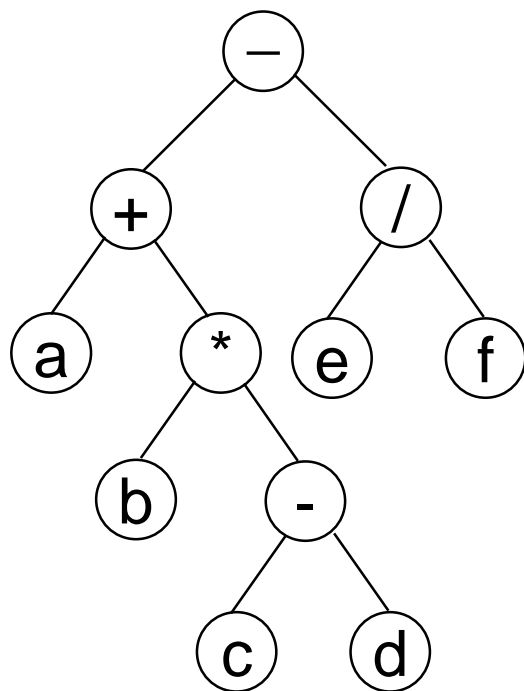
LRD



后序遍历序列: D G E B F C A

练习

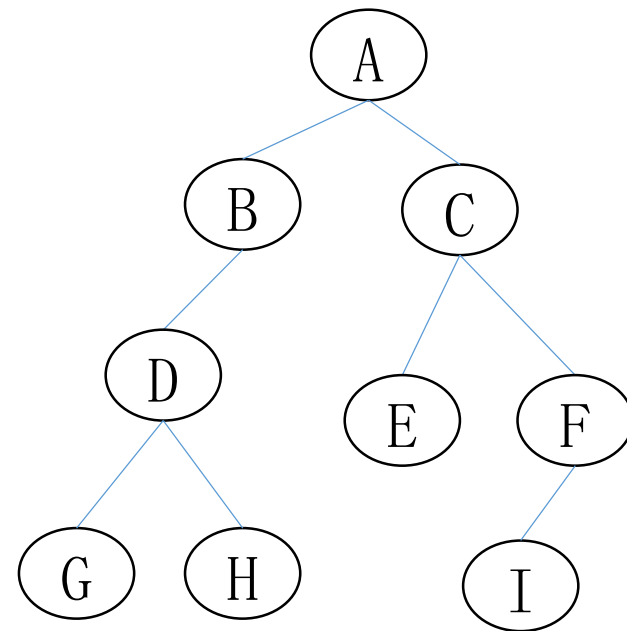
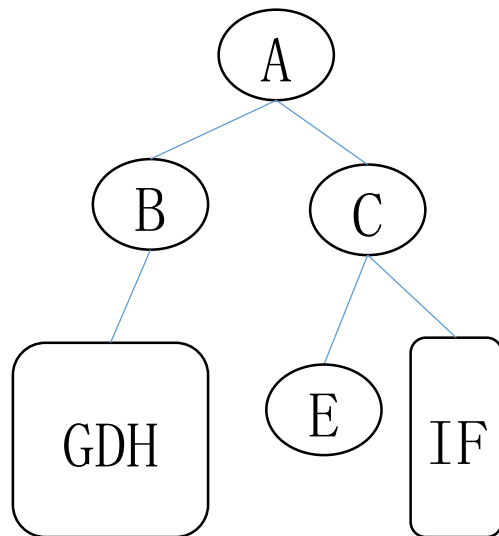
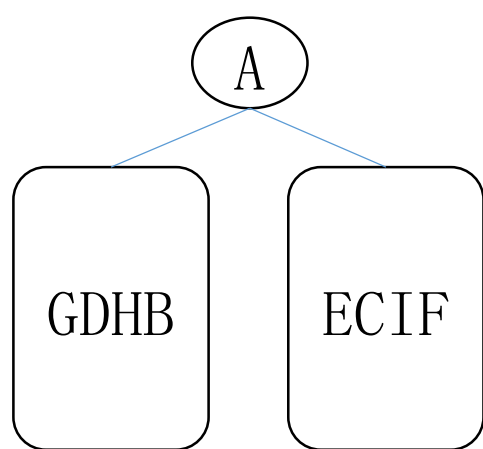
写出下列二叉树的先序、中序、后序遍历序列



表达式 $(a+b*(c-d)-e/f)$ 二叉树

求原二叉树

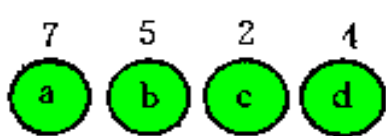
已知一棵二叉树的先序遍历序列和中序遍历序列分别为
ABDGHCEFI和**GDHBAECIF**，请画出这棵二叉树，然后给出该
树的后序遍历序列。



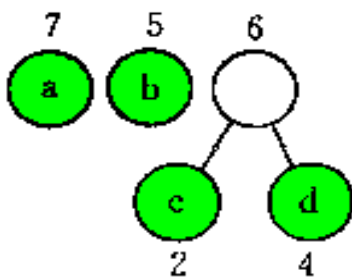
后序遍历: **G H D B E I F C A**

哈夫曼树

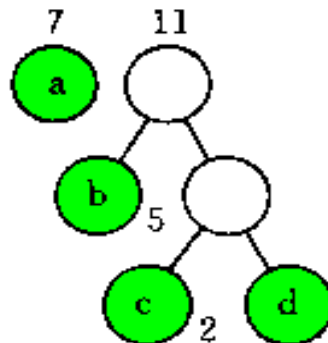
构造哈夫曼树： 给 n 个点，每个点都有权值，构造一棵哈夫曼树。每次选剩下的两棵根权值最小的树合并成一棵新树，新树的根权值等于两棵合并前树的根权值和。



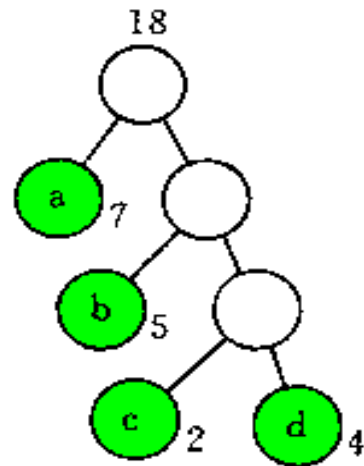
(a)



(b)



(c)

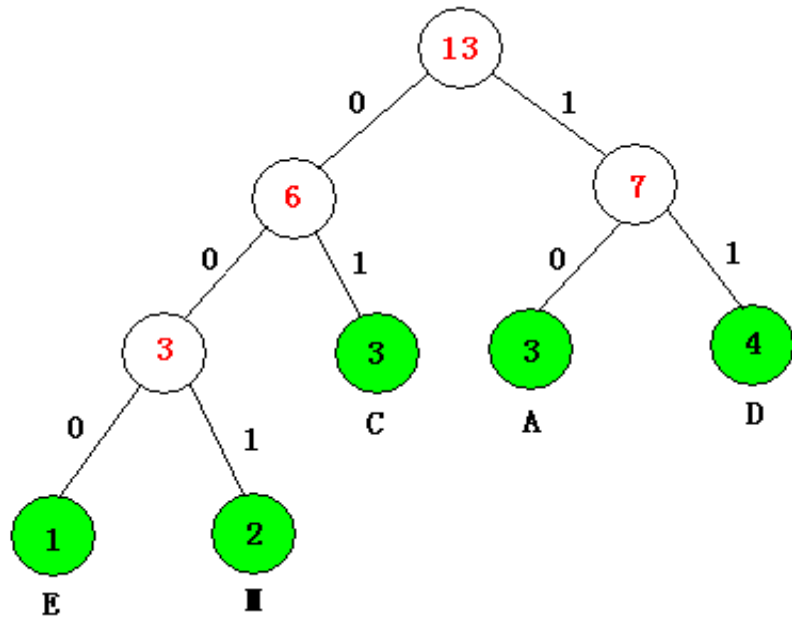


(d)

4个点合并3次 (n 个点，合并 $n-1$ 次)

- ① 选根权值最小的两棵树2 (c) 和4 (d) 合并，新树的根节点为6，如图(b);
- ② 选根权值最小的两棵树5 (b) 和6合并，新树的根节点为11，如图(c);
- ③ 选根权值最小的两棵树7 (a) 和11合并，新树的根节点为18，如图(d);

哈夫曼编码



由哈夫曼编码树获得每个字母编码：

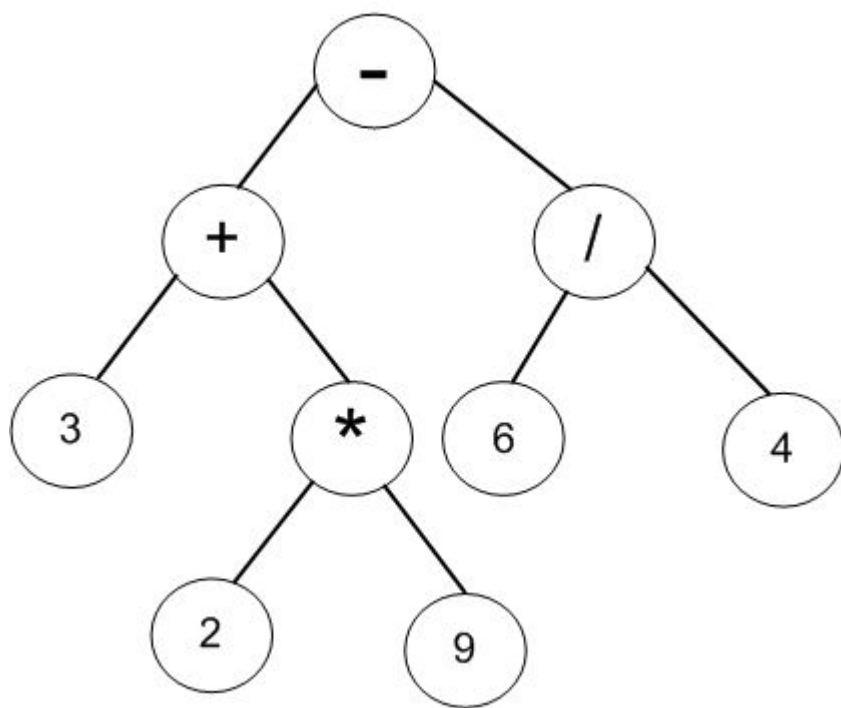
E	M	C	A	D
000	001	01	10	11

5个字母的使用频度分别为
 $\{E, M, C, A, D\} = \{1, 2, 3, 3, 4\}$
用频度为权值生成哈夫曼树，
并在叶子上标注对应的字母，
在树枝上标注分配码“0”或“1”

从哈夫曼树根节点开始，对左子树分配码“0”，右子树分配码“1”，一直到达叶子节点为止，然后将从树根沿每条路径到达叶子结点的代码排列起来，便得到了哈夫曼编码。

表达式树

以基本运算对象作为叶节点中的数据；以运算符作为非叶节点中的数据，其两棵子树是它的运算对象，子树可以是基本运算对象，也可以是复杂表达式。

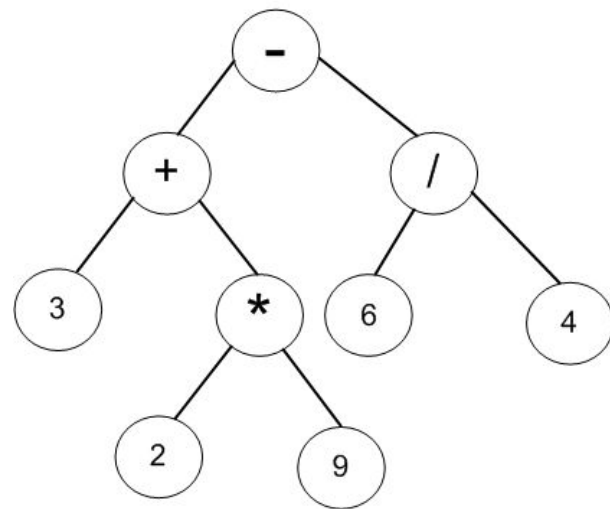


$3 + 2 * 9 - 6 / 4$ 表达式树

构造表达式树

给定一个表达式的中缀形式，例如， $3+2*9-6/4$

- ① 将每个运算加上括号，区分优先级，得到 $(3+(2*9))-(6/4)$ ，括号外的 $-$ 优先级最低，作为根节点， $(3+(2*9))$ 作为左子树， $(6/4)$ 作为右子树；
- ② 递归转换 $(3+(2*9))$ ， $+$ 是根节点， 3 是左子树， $(2*9)$ 是右子树。 $*$ 是右子树的根节点， 2 是左子树， 9 是右子树。
- ③ 递归转换 $(6/4)$ ， $/$ 是根节点， 6 是左子树， 4 是右子树。



构造好表达式树后，前缀表达式和后缀表达式可根据先序遍历和后序遍历得到
前缀表达式： $- + 3 * 2 9 / 6 4$
后缀表达式： $3 2 9 * + 6 4 / -$

后缀表达式

常见的数学公式或表达式称为中缀表达式，运算符在运算数中间

$$3+2*9-6/4 \quad \Rightarrow \quad 3 \ 2 \ 9 \ * \ + \ 6 \ 4 \ / \ -$$

后缀表达式不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右扫描表达式。

3	2	9
---	---	---

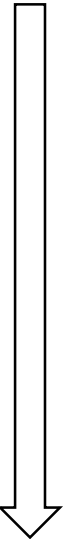
3	18
---	----

21	6	4
----	---	---

21	1
----	---

20

* 2*9
+ 3+18
/ 6/4
- 21-1



后缀表达式计算规则：

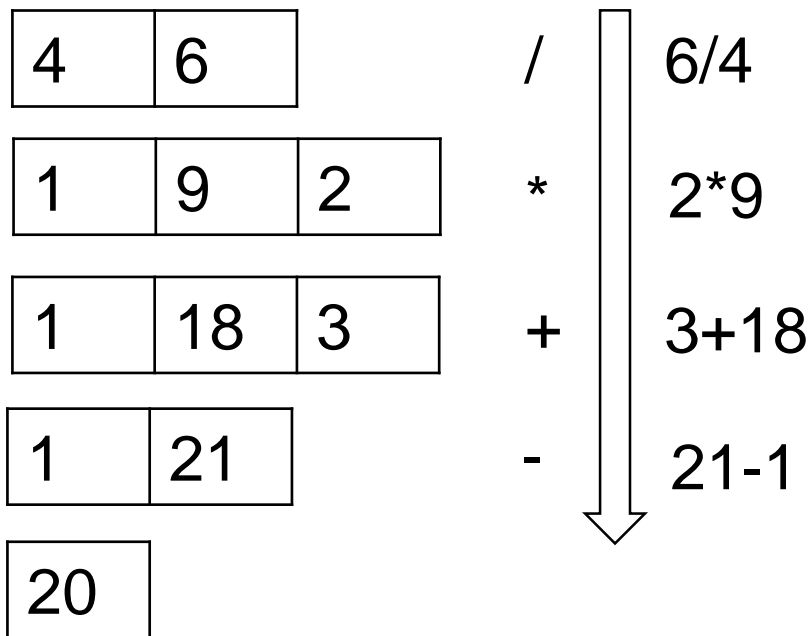
- (1)如果是操作数，则放入栈中；
- (2)如果是操作符，则取出栈中两个操作数，进行运算后，将结果放入栈中；
- (3)直到最后栈中只有一个元素，此元素就是计算结果。

前缀表达式

常见的数学公式或表达式称为中缀表达式，运算符在运算数中间

$$3+2*9-6/4 \quad \Rightarrow \quad - + 3 * 2 9 / 6 4$$

前缀表达式又称波兰式，前缀表达式的运算符位于操作数之前。
严格从右至左扫描表达式。



前缀表达式计算规则：

- (1)如果是操作数，则放入栈中；
- (2)如果是操作符，则取出栈中两个操作数，进行运算后，将结果放入栈中；
- (3)直到最后栈中只有一个元素，此元素就是计算结果。

练习

前序遍历序列与中序遍历序列相同的二叉树为（ ）。

- A.根结点无左子树的二叉树
- B.根结点无右子树的二叉树
- C.只有根结点的二叉树或非叶子结点只有左子树的二叉树
- D.只有根结点的二叉树或非叶子结点只有右子树的二叉树

如果根的高度为1，具有61个结点的完全二叉树的高度为（ ）。

- A.5
- B.6
- C.7
- D.8

完全二叉树共有 2^N-1 个结点，则它的叶节点数是（ ）。

- A. $N-1$
- B. N
- C. 2^N
- D. 2^{N-1}

一个包含 n 个分支结点（非叶结点）的非空二叉树，它的叶结点数最多为（ ）

- A) $2n + 1$
- B) $2n-1$
- C) $n-1$
- D) $n+1$

练习

二叉树T，已知其先根遍历是1 2 4 3 5 7 6（数字为结点的编号，以下同），中根遍历是2 4 1 5 7 3 6，则该二叉树的后根遍历是（ ）。

- | | |
|------------------|------------------|
| A. 4 2 5 7 6 3 1 | B. 4 2 7 5 6 3 1 |
| C. 7 4 2 5 6 3 1 | D. 4 2 7 6 5 3 1 |

已知 7 个结点的二叉树的先根遍历是 1 2 4 5 6 3 7（数字为结点的编号，以下同），中根遍历是 4 2 6 5 1 7 3，则该二叉树的后根遍历是（ ）

- | | |
|------------------|------------------|
| A. 4 6 5 2 7 3 1 | B. 4 6 5 2 1 3 7 |
| C. 4 2 3 1 5 4 7 | D. 4 6 5 3 1 7 2 |

练习

完全二叉树的顺序存储方案，是指将完全二叉树的结点从上至下、从左至右依次存放到一个顺序结构的数组中。假定根结点存放在数组的1号位置，则第K号结点的父结点如果存在的话，应当存放在数组的（ ）号位置。

- A. $2k$ B. $2k+1$ C. $k/2$ 下取整 D. $(k+1)/2$ 下取整

前缀表达式 “+3*2+5 12”的值是（ ）。

- A. 23 B. 25 C. 37 D. 65