树与二叉树

胡船长

初航我带你，远航靠自己

本章题目

1-应试. Leetcode-589：N叉树的前序遍历

2-应试. Leetcode-105：从前序遍历与中序遍历构造二叉树

3-应试. Leetcode-102：二叉树的层序遍历

4-应试. Leetcode-226：翻转二叉树

5-校招. Leetcode-107：二叉树的层序遍历 II

6-校招. Leetcode-103：二叉树的锯齿形层序遍历

7-校招. 剑指 Offer 26：树的子结构

8-竞赛. HZOJ-287：合并果子

9-竞赛. HZOJ-245：货仓选址

本期内容

一. 计算机中的树形结构

二. 二叉树：结构讲解

三. 二叉树：遍历与线索化

四. 二叉树的广义表表示法

五. 最优变长编码：哈夫曼编码

一. 计算机中的树形结构

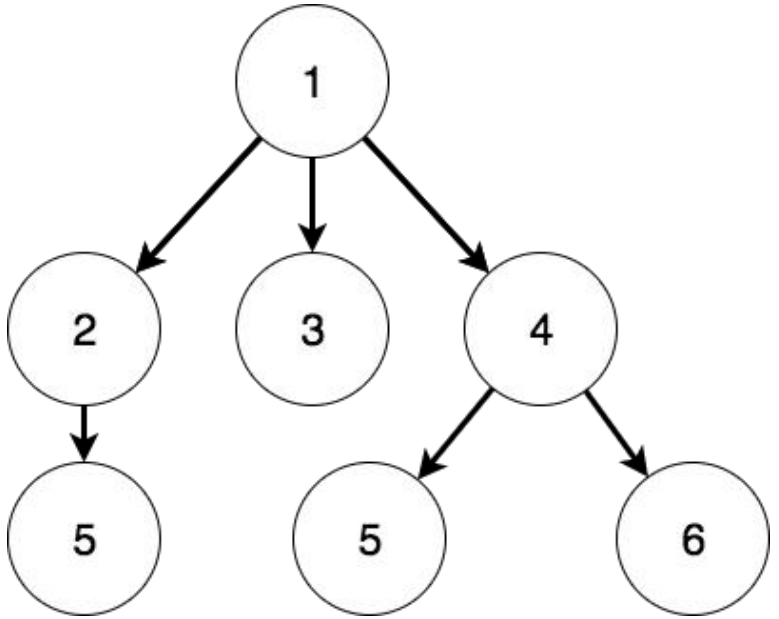
计算机中的树形结构

现实中的树

计算机中的树

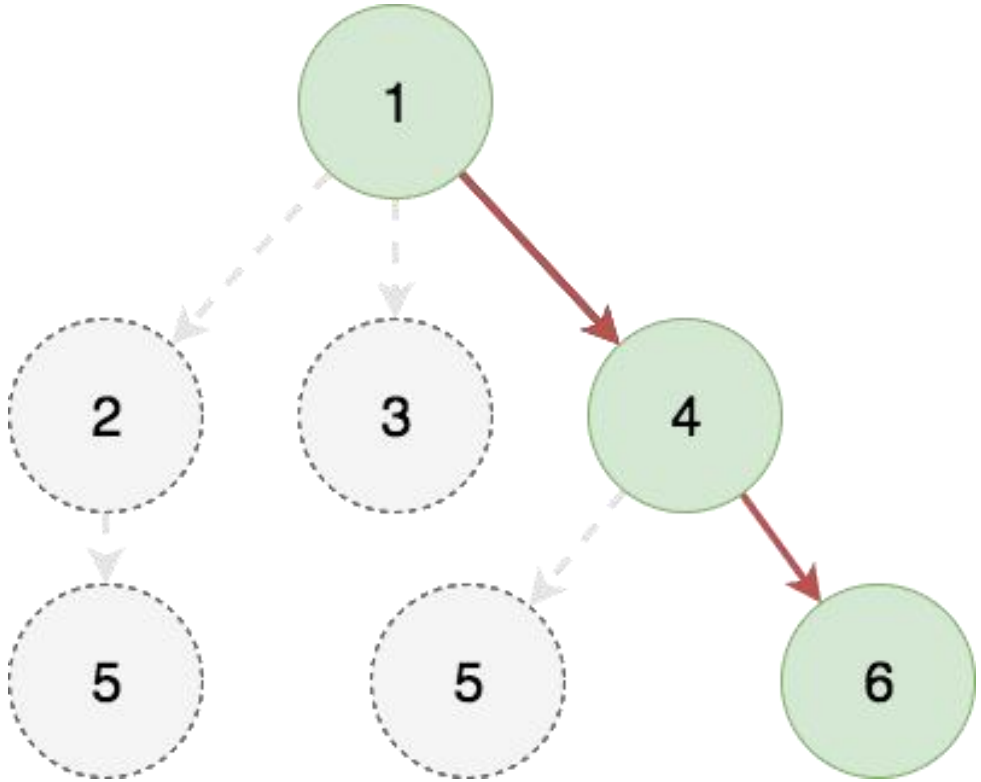
计算机中的树形结构

现实中的树



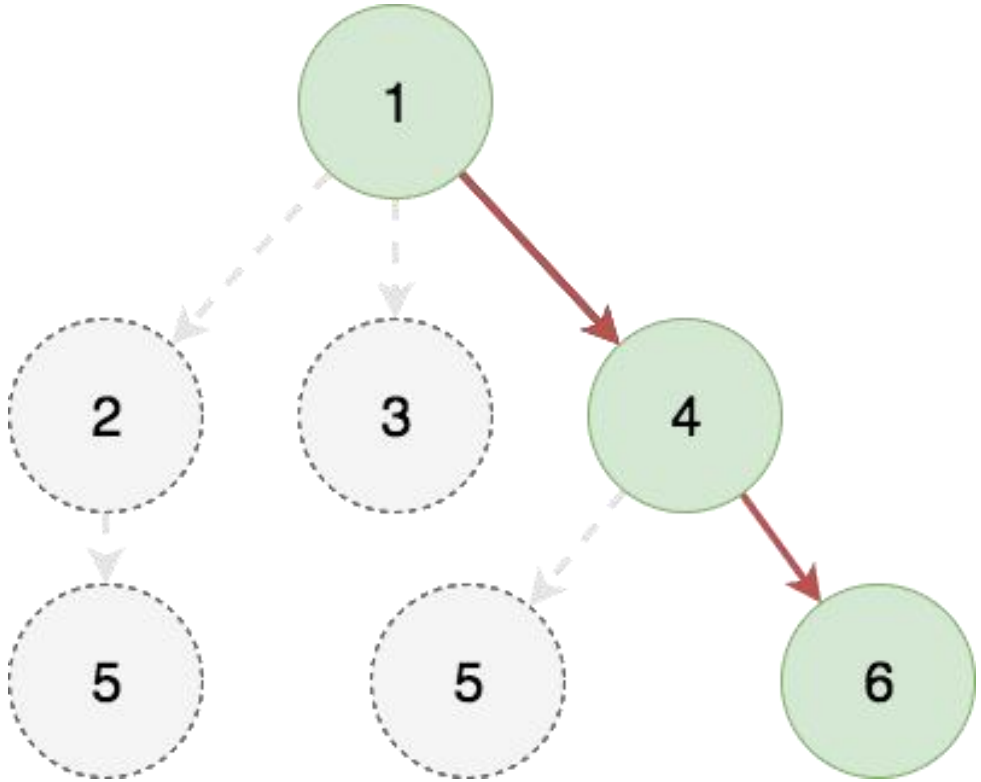
计算机中的树

树-结构定义




```
typedef struct Node{
    int data;
    struct Node *next;
}Node, *LinkedList;
```

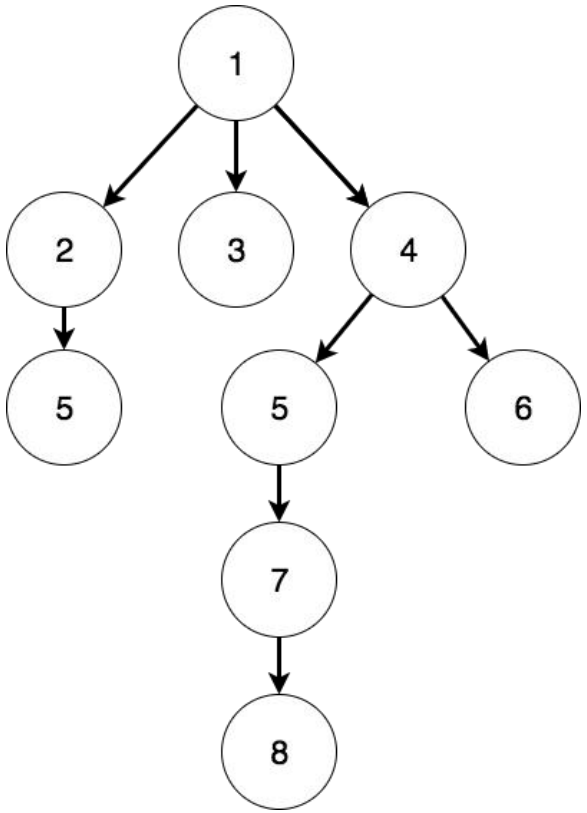
树-结构定义




```
typedef struct Node{
    int data;
    struct Node *next;
}Node, *LinkedList;
```

```
typedef struct Node{
    int data;
    struct Node *next[3];
} Node, *Tree;
```

树-深度、高度和度

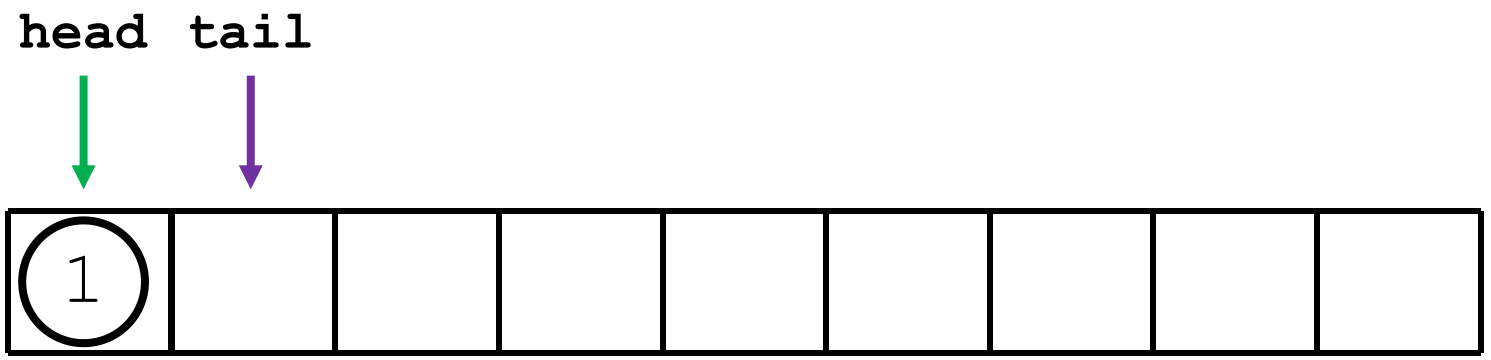


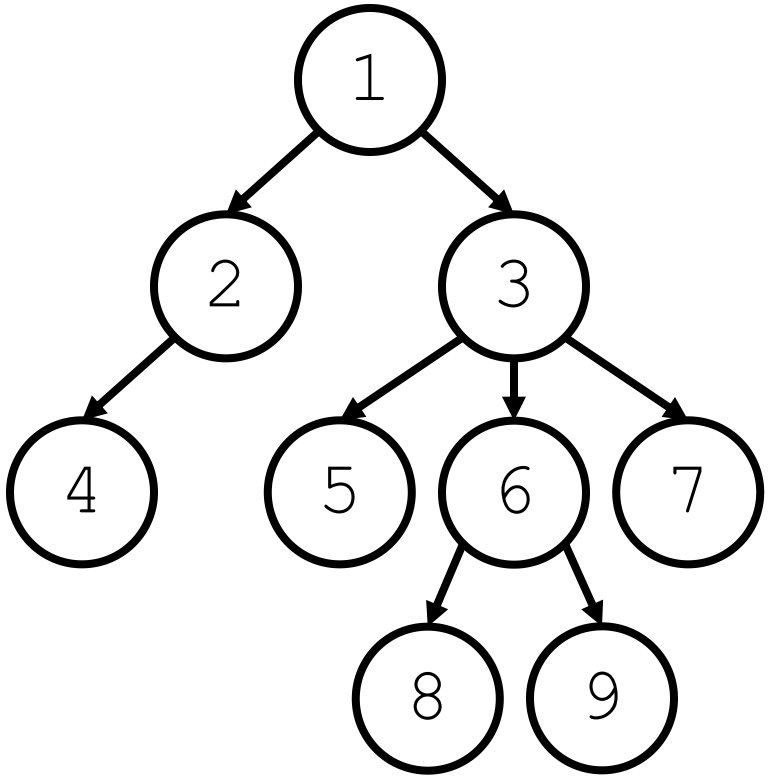
1、树的深度(高度)为 5
2、节点4的深度为1，高度为3
3、节点2的度为1，节点1的度为3
4、节点数量等于边数+1

树结构：深入理解

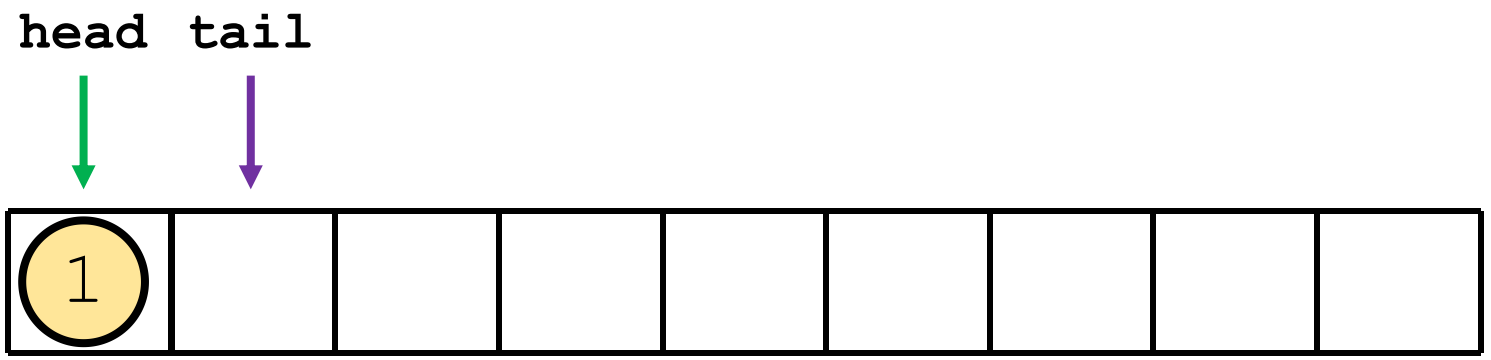
树的节点代表【集合】，树的边代表【关系】

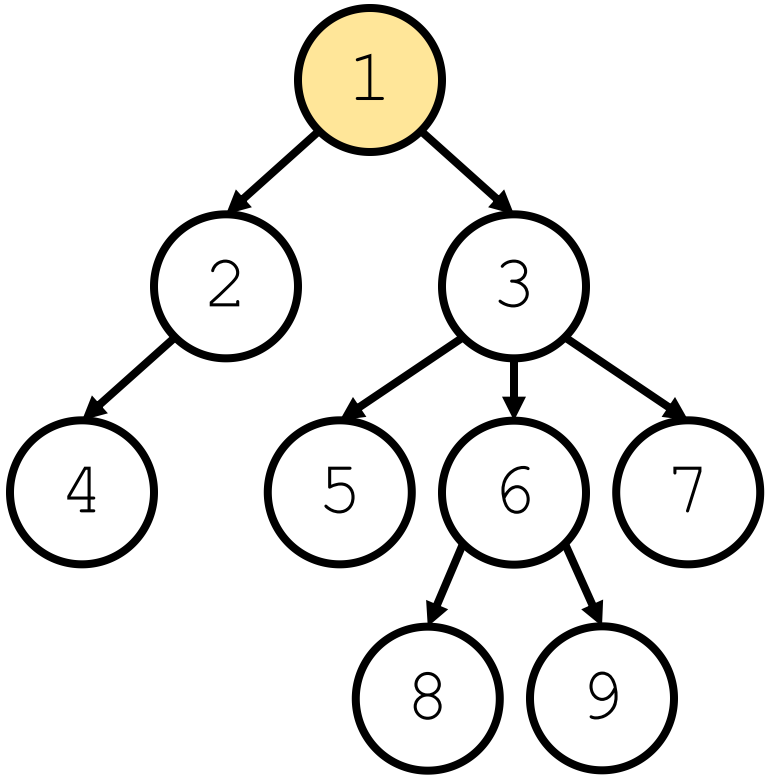
树-广度优先遍历



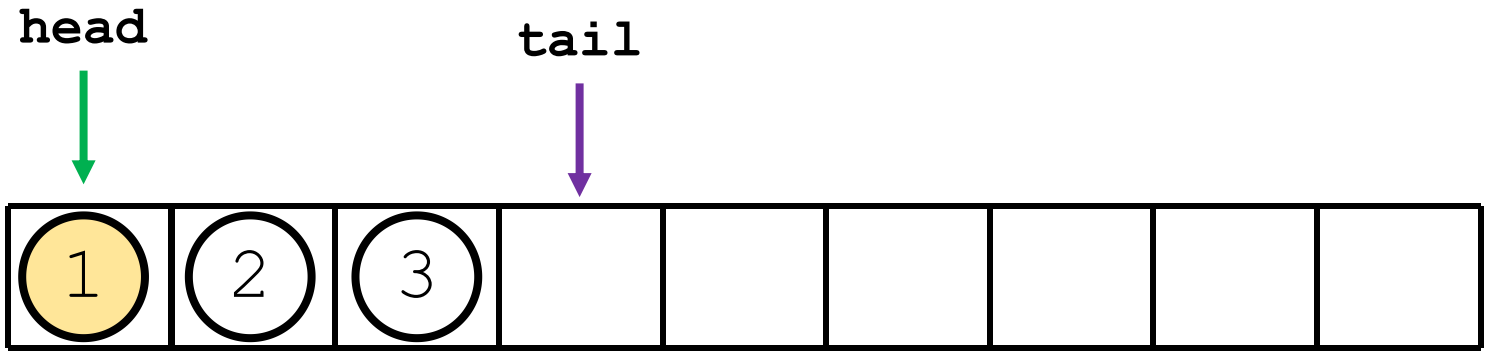


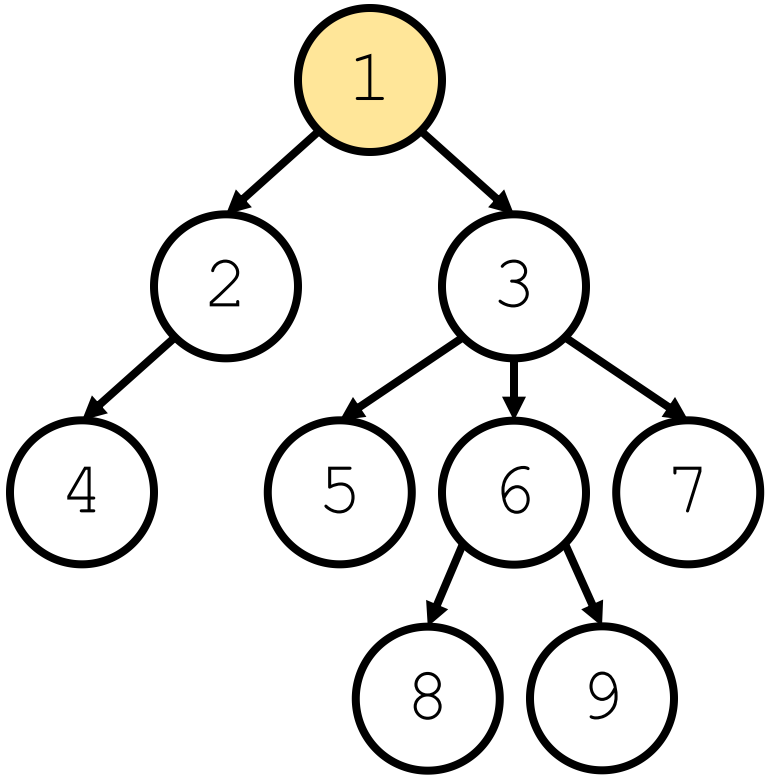
树-广度优先遍历



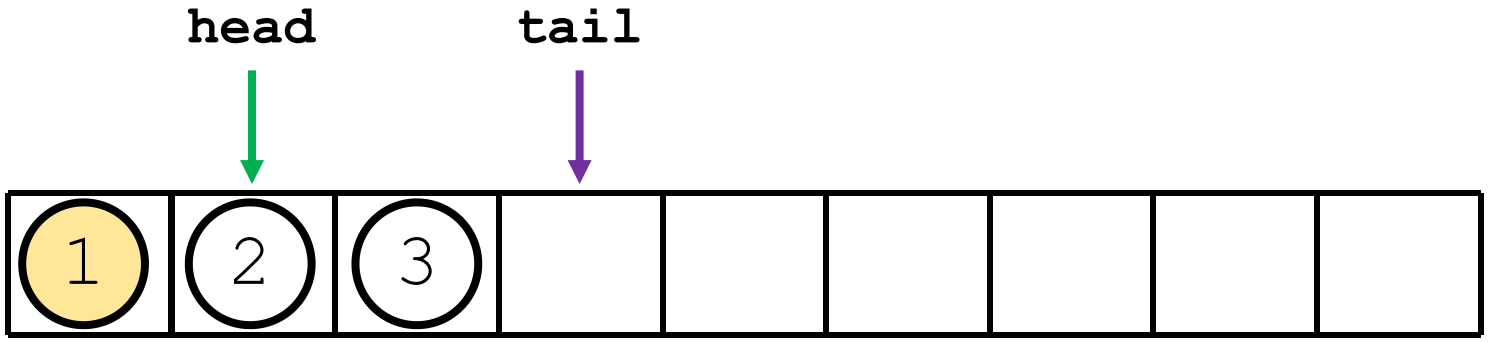


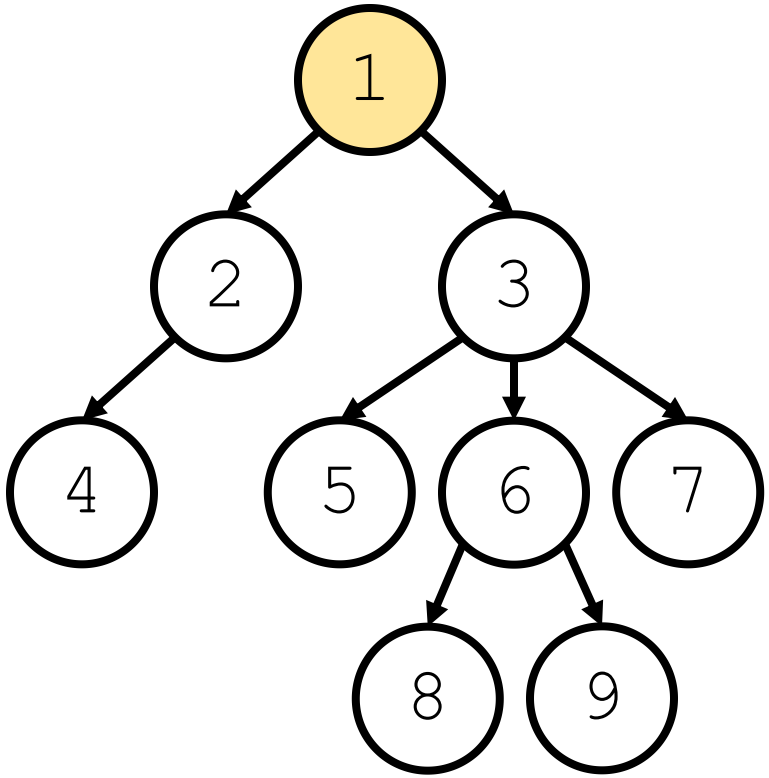
树-广度优先遍历



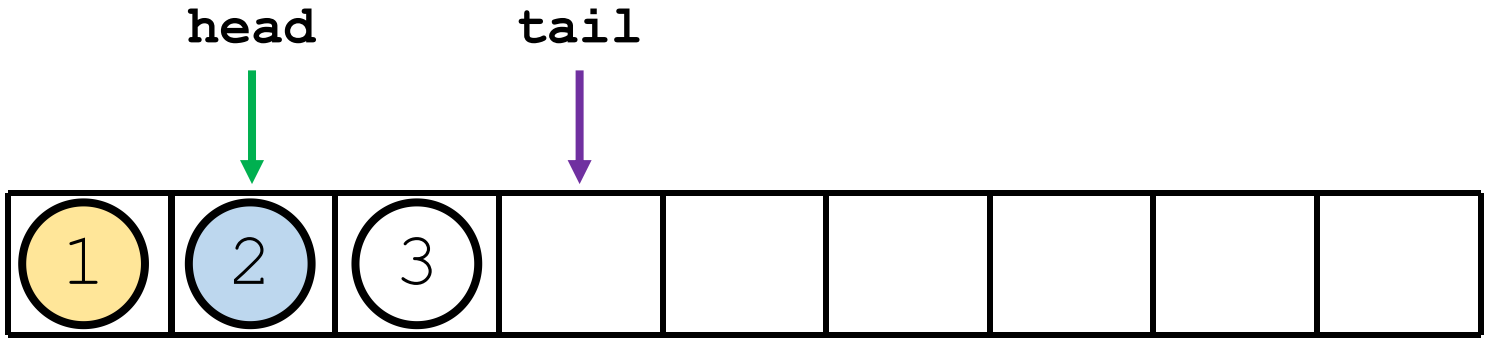


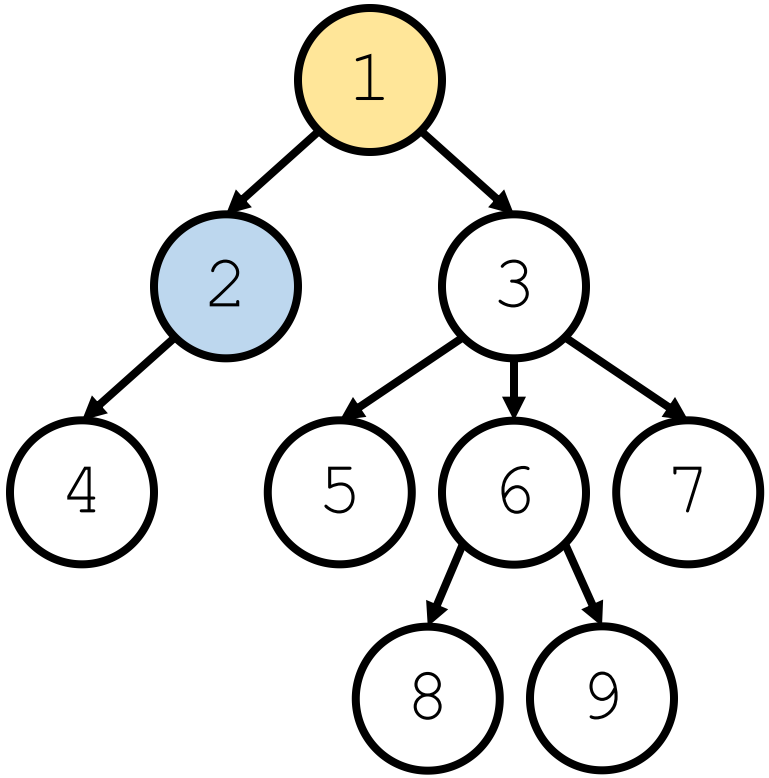
树-广度优先遍历



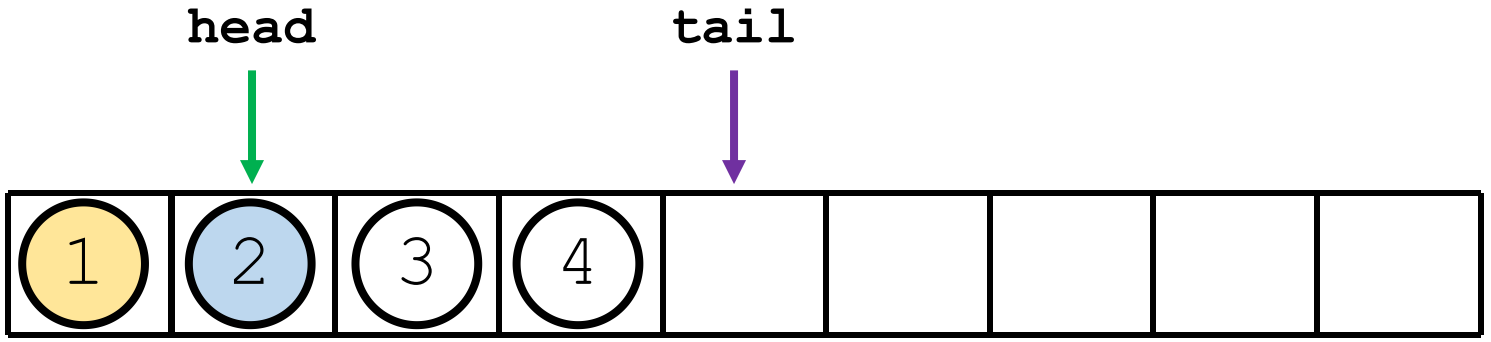


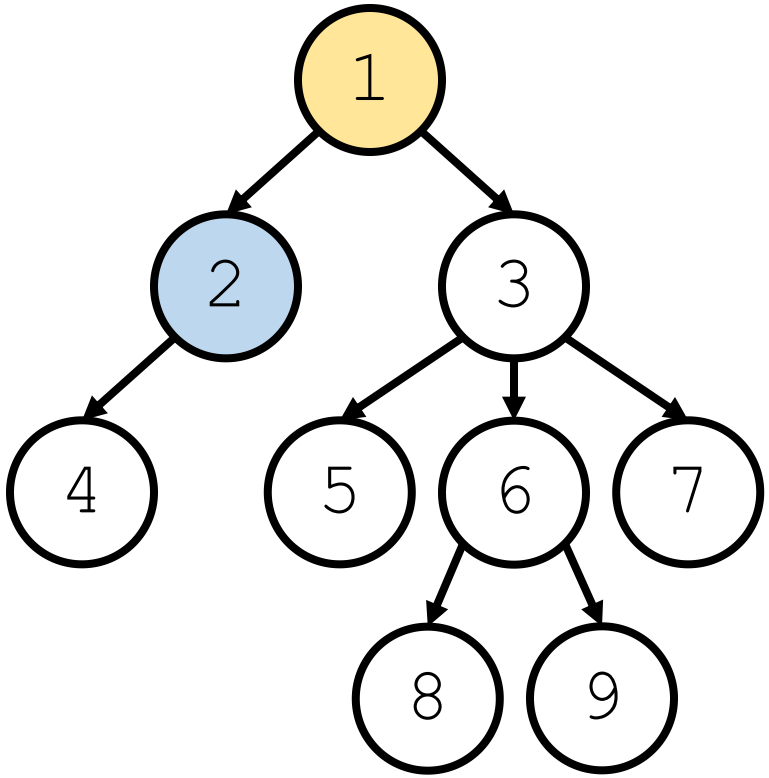
树-广度优先遍历



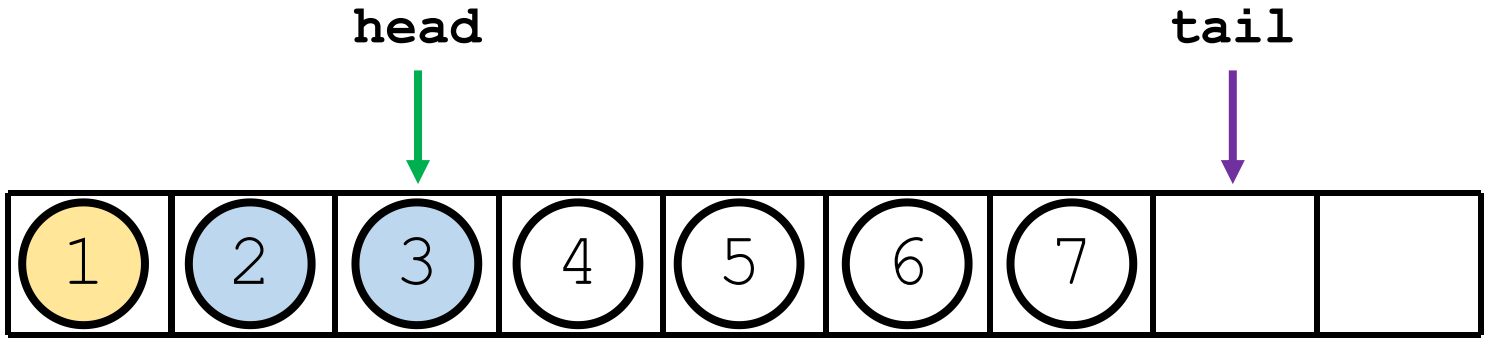


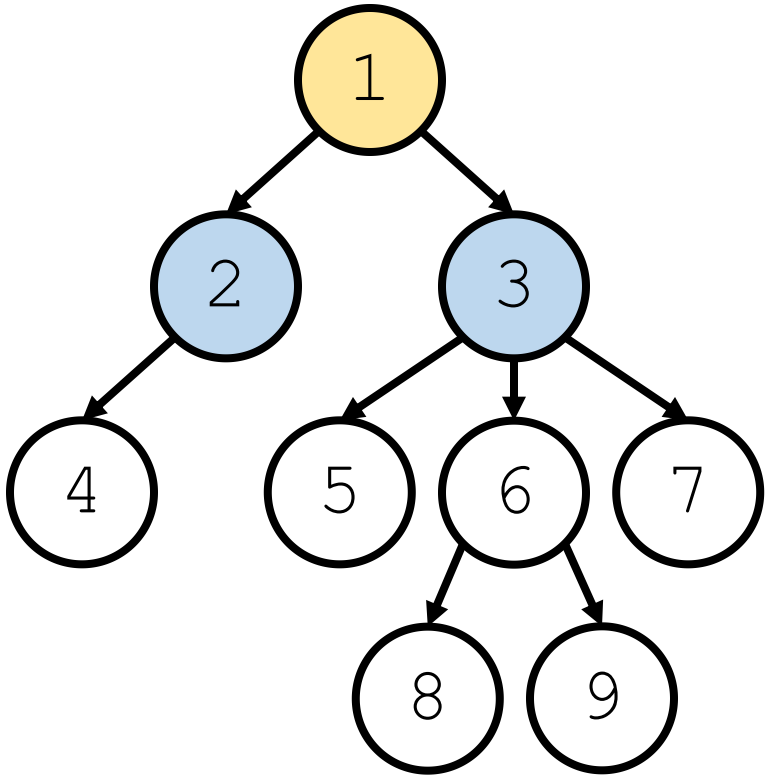
树-广度优先遍历



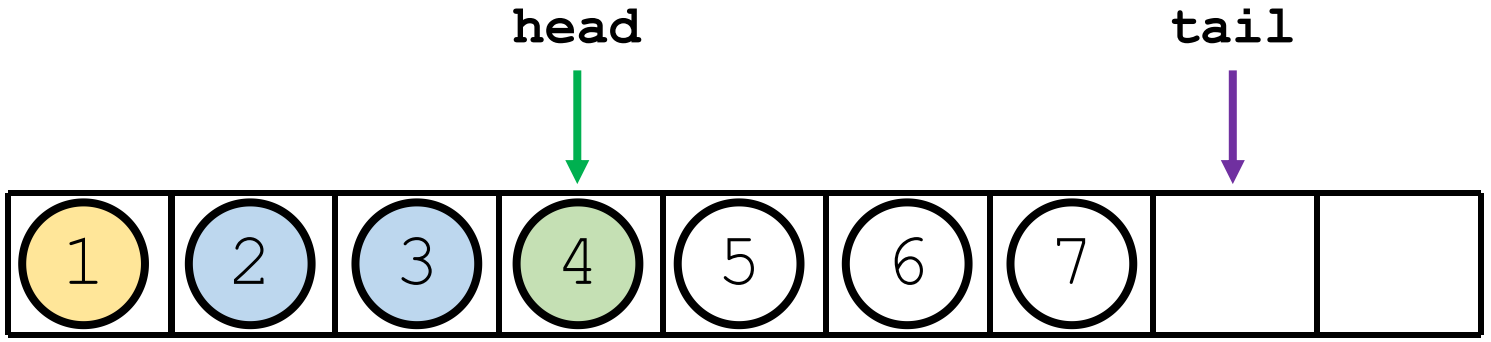


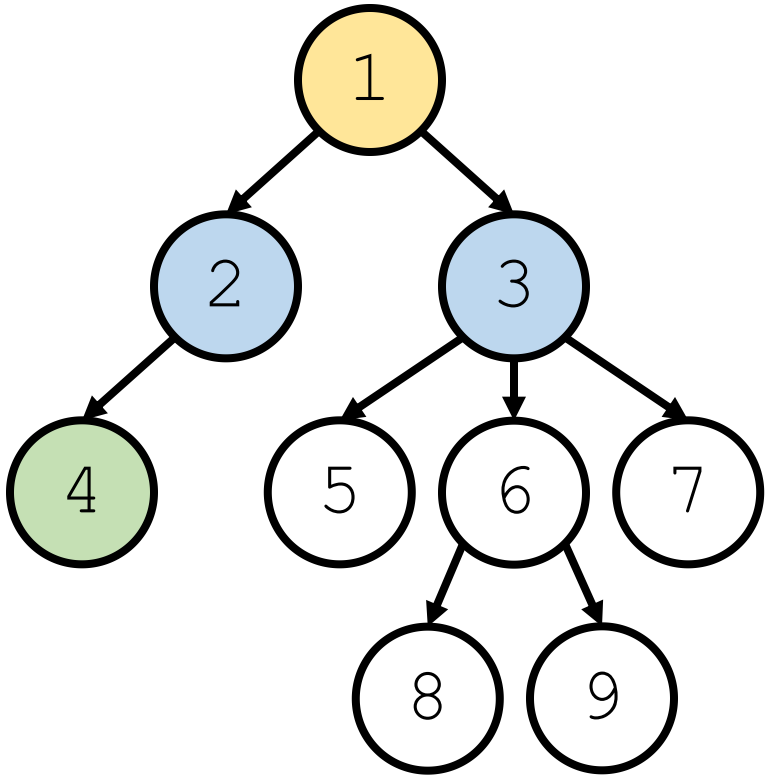
树-广度优先遍历



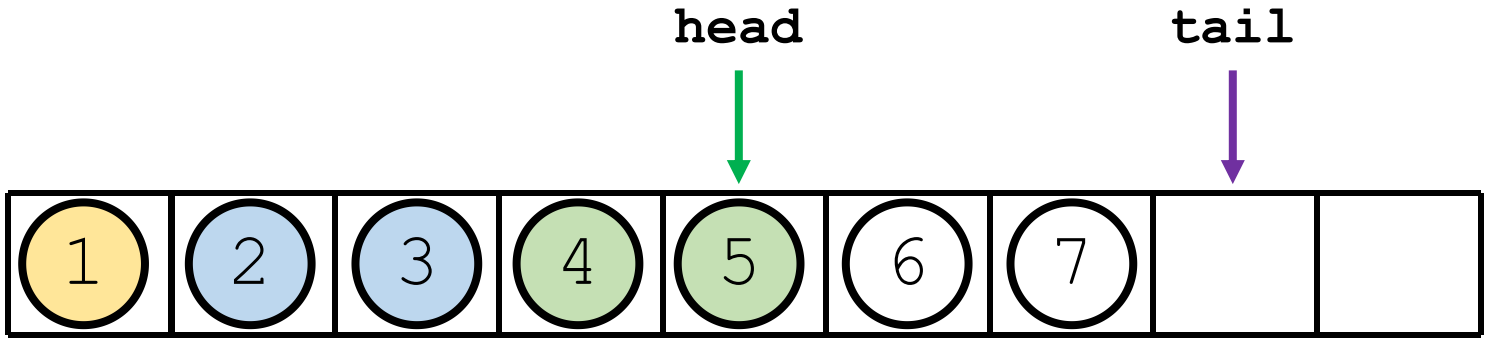


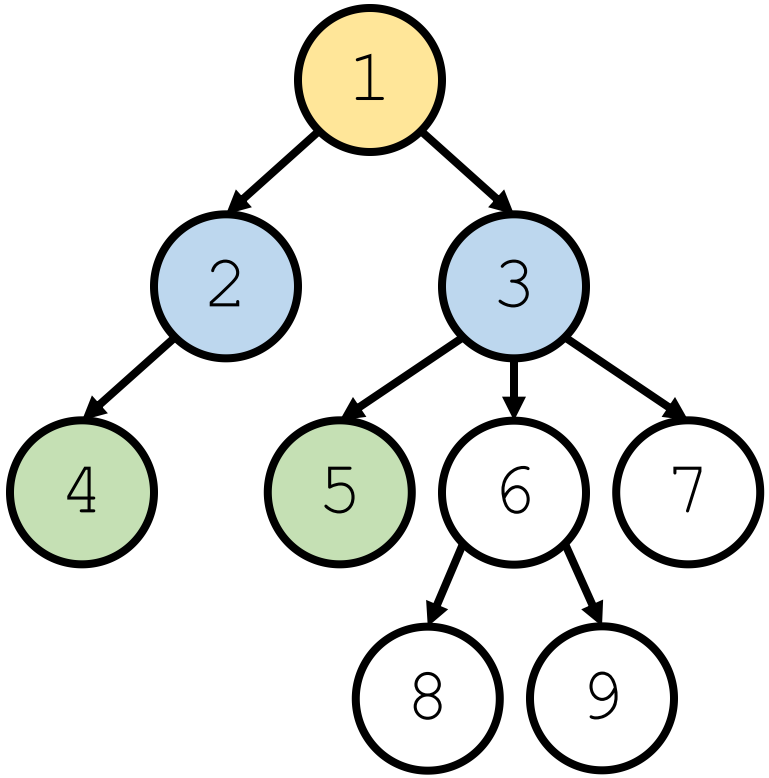
树-广度优先遍历



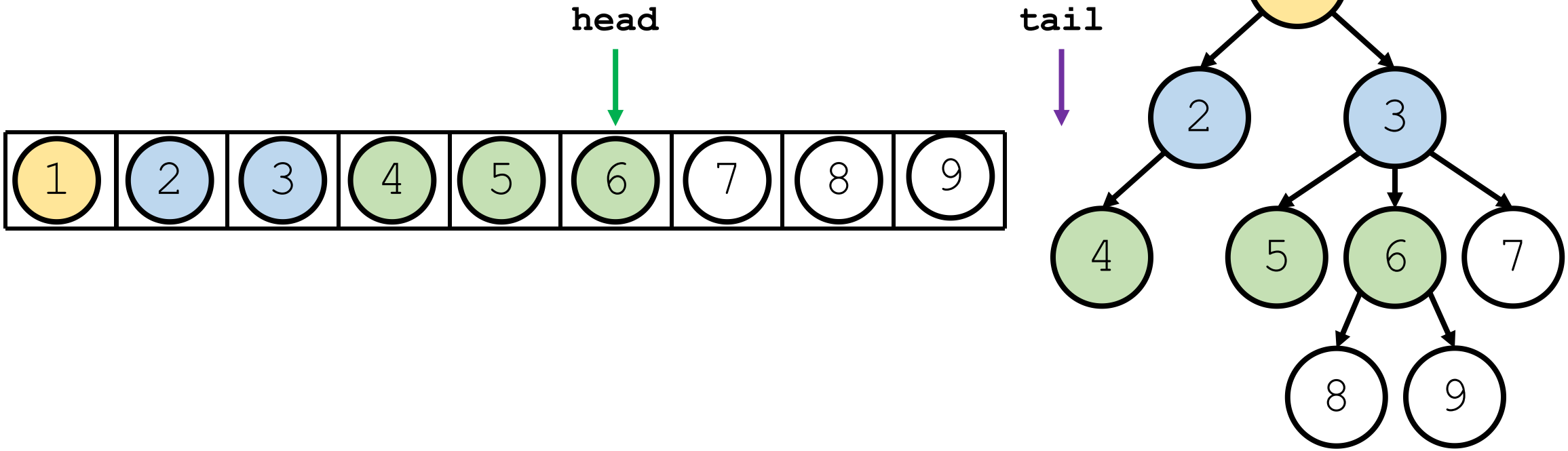


树-广度优先遍历

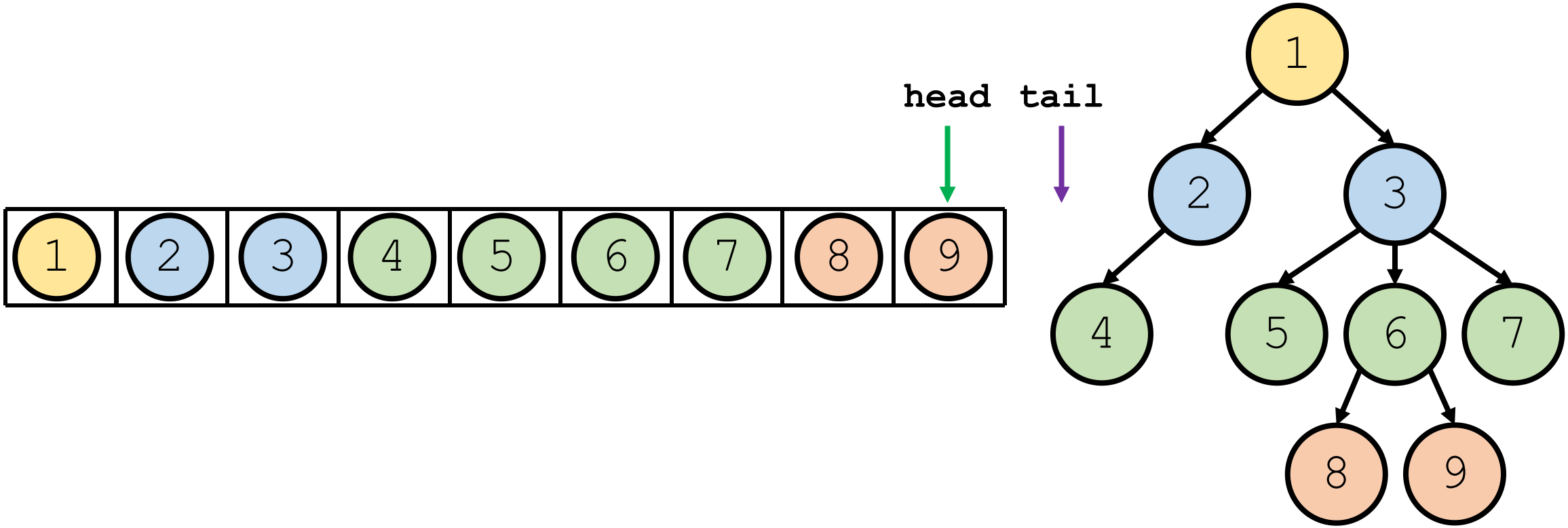




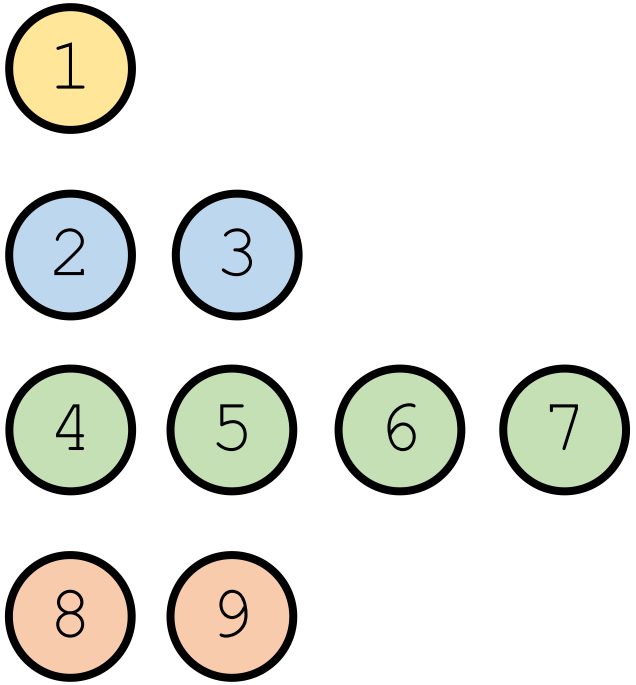
树-广度优先遍历

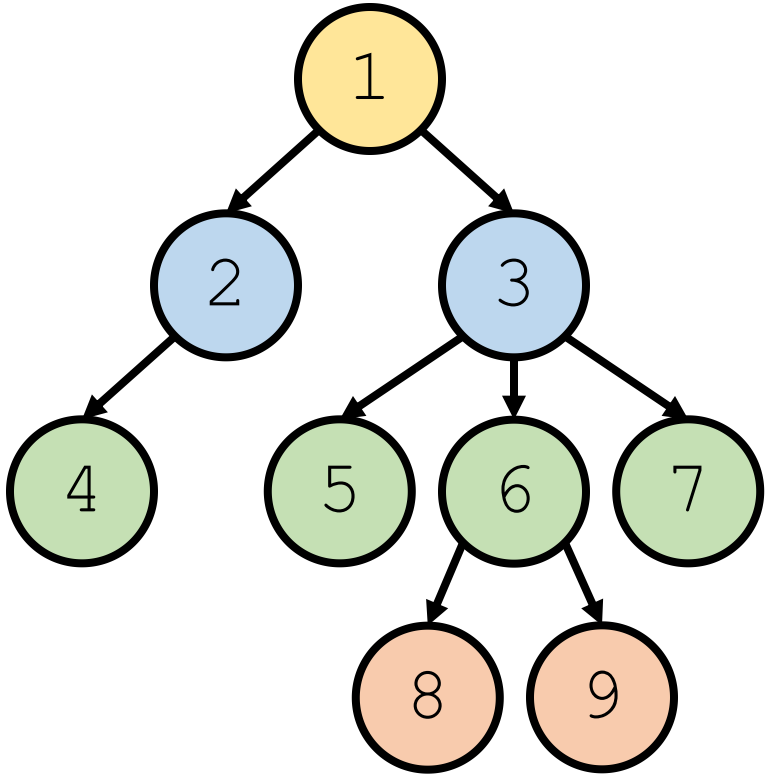


树-广度优先遍历



树-广度优先遍历

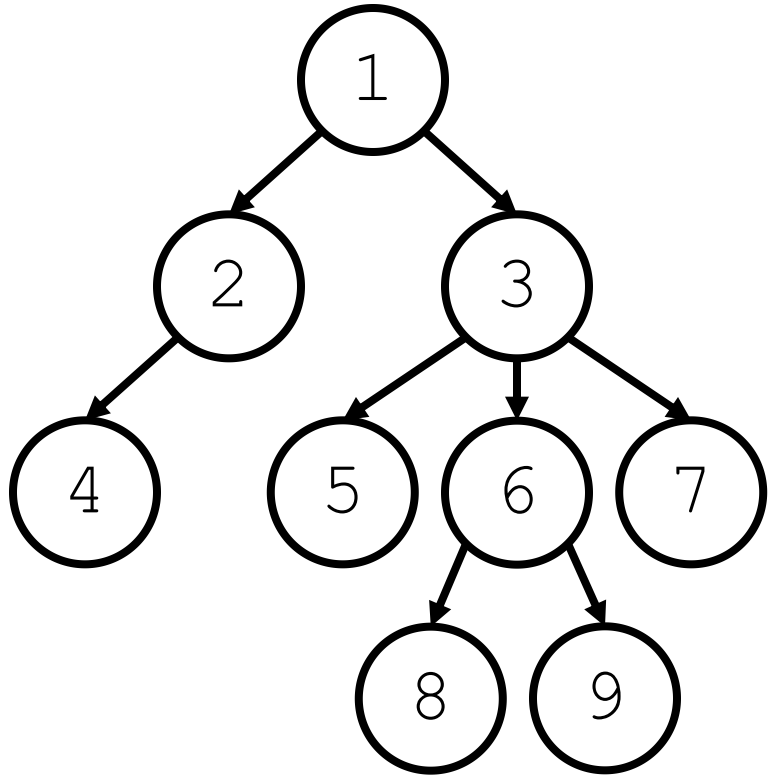




树-深度优先遍历

进入序列

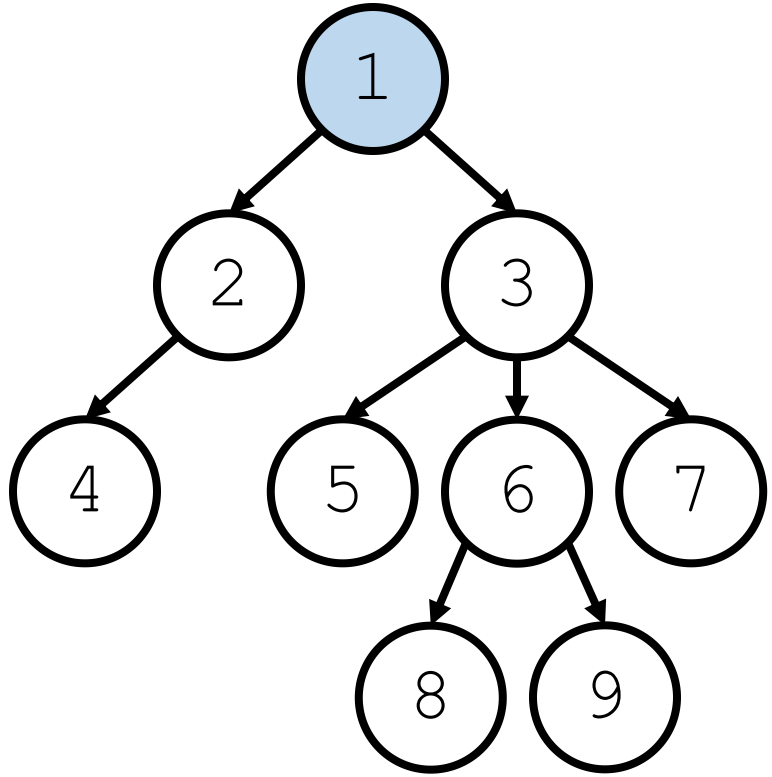
退出序列



树-深度优先遍历

进入序列

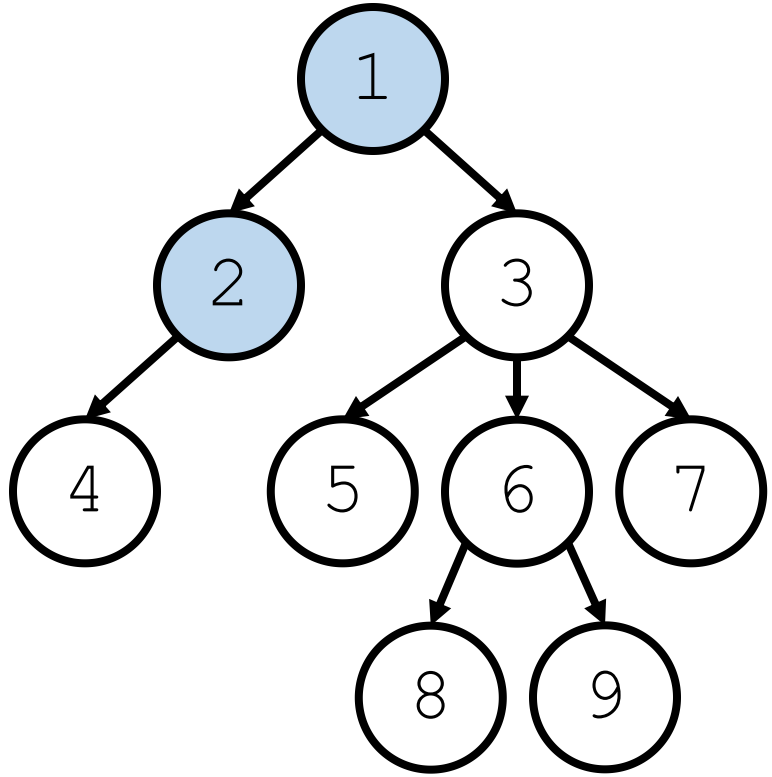
退出序列



树-深度优先遍历

进入序列

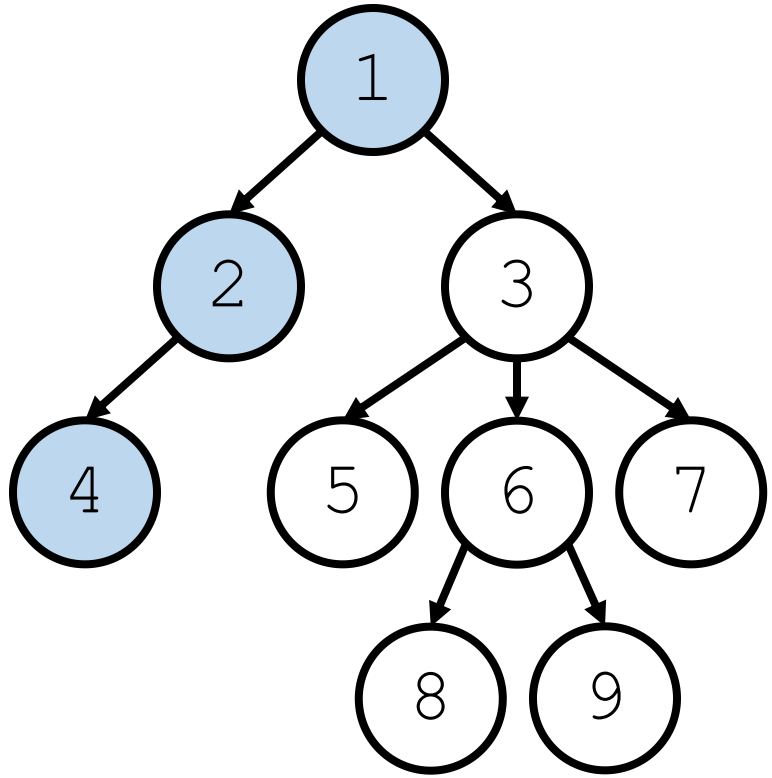
退出序列



树-深度优先遍历

进入序列

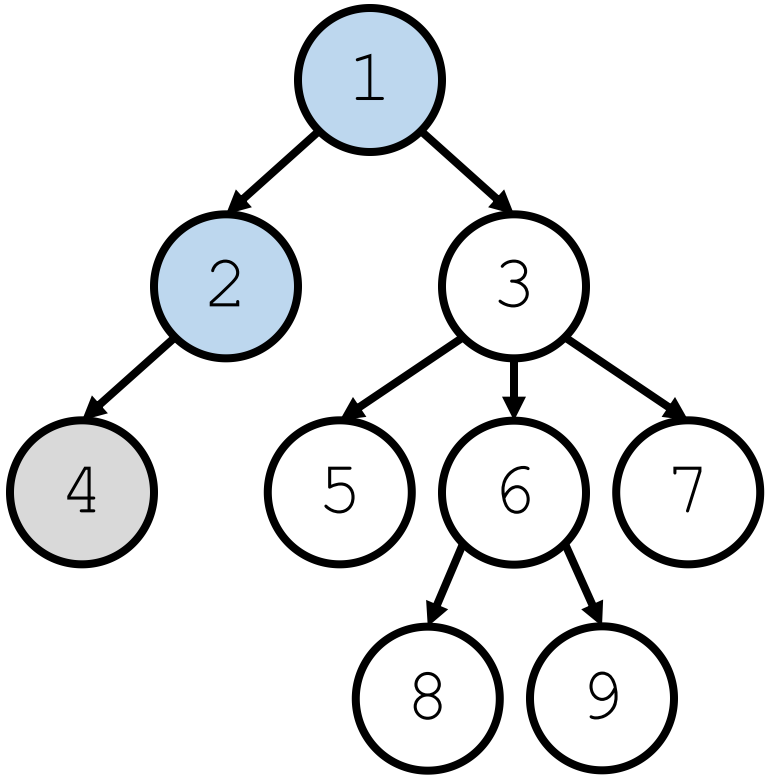
退出序列



树-深度优先遍历

进入序列

退出序列



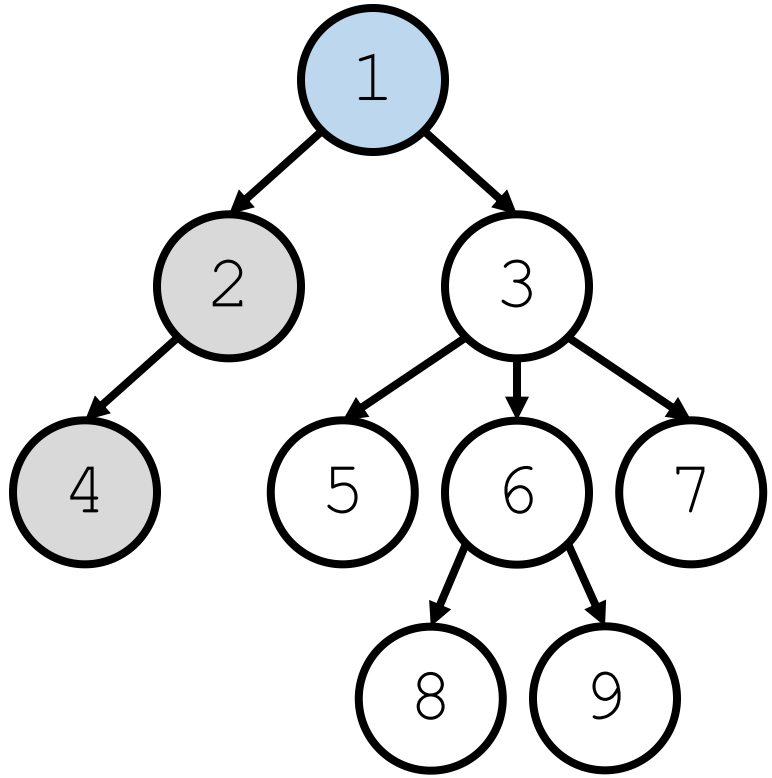
树-深度优先遍历

进入序列

1	2	4						

退出序列

4	2							



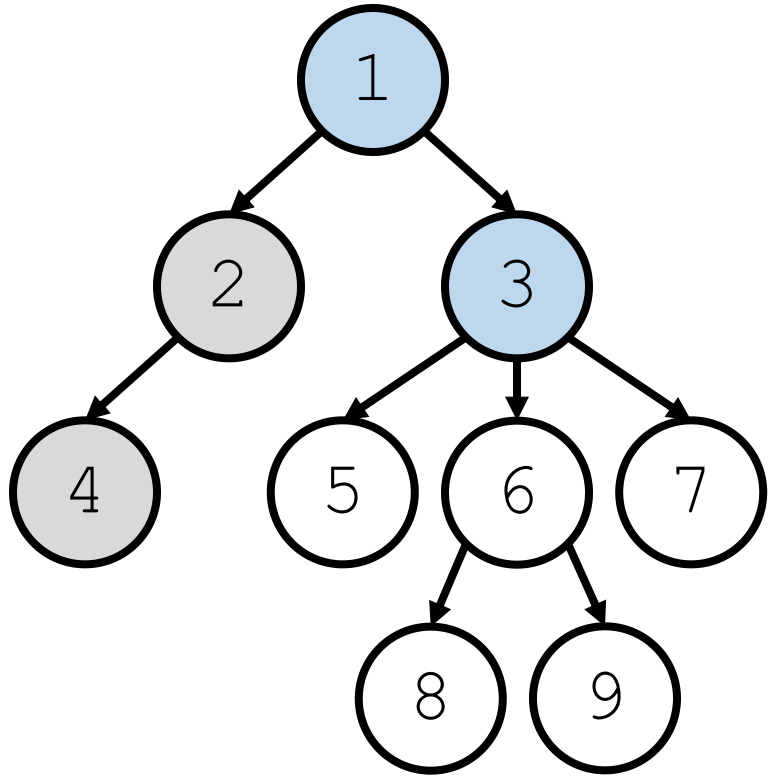
树-深度优先遍历

进入序列

1	2	4	3					

退出序列

4	2							



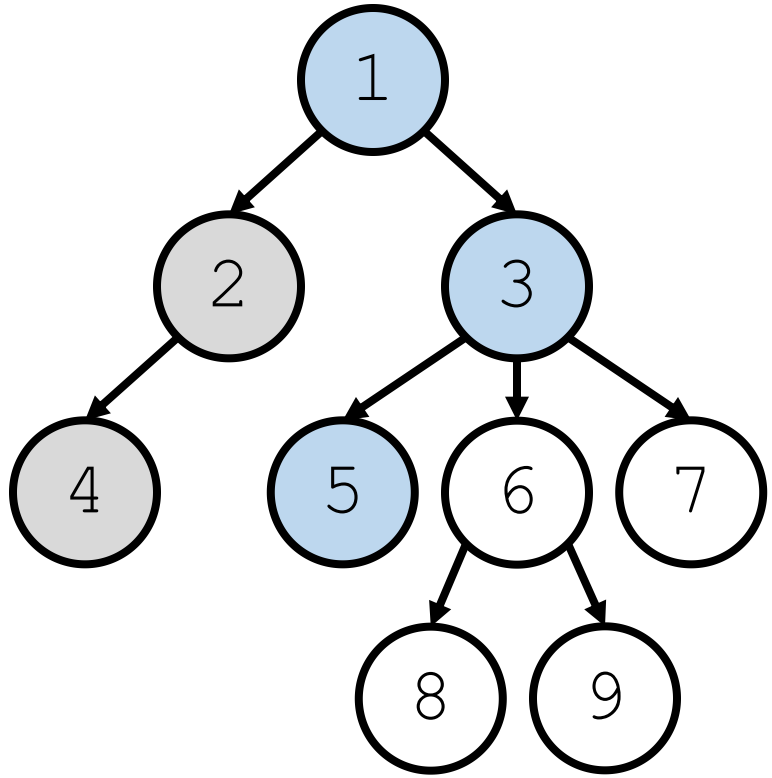
树-深度优先遍历

进入序列

1	2	4	3	5				

退出序列

4	2							



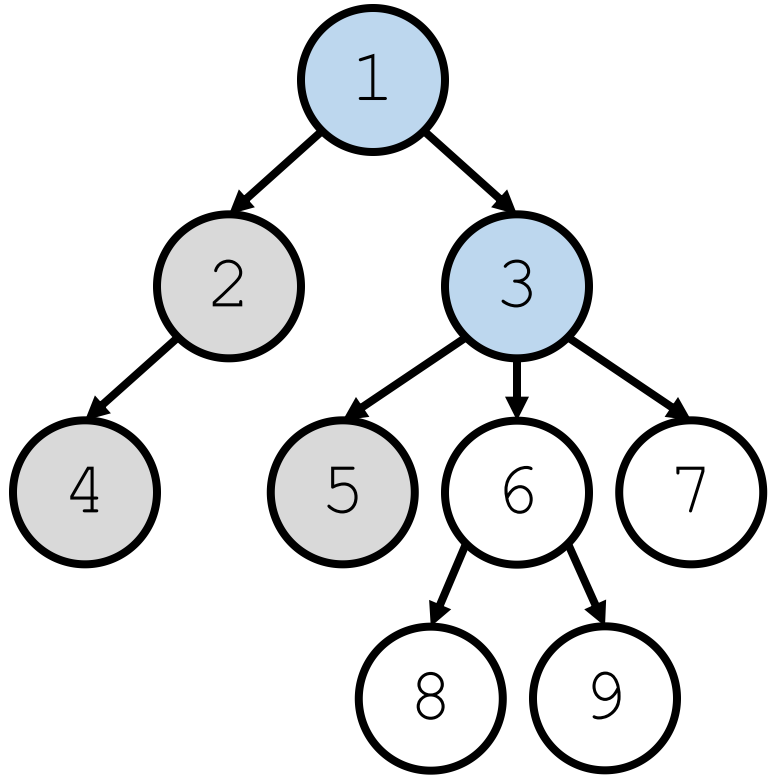
树-深度优先遍历

进入序列

1	2	4	3	5				

退出序列

4	2	5						



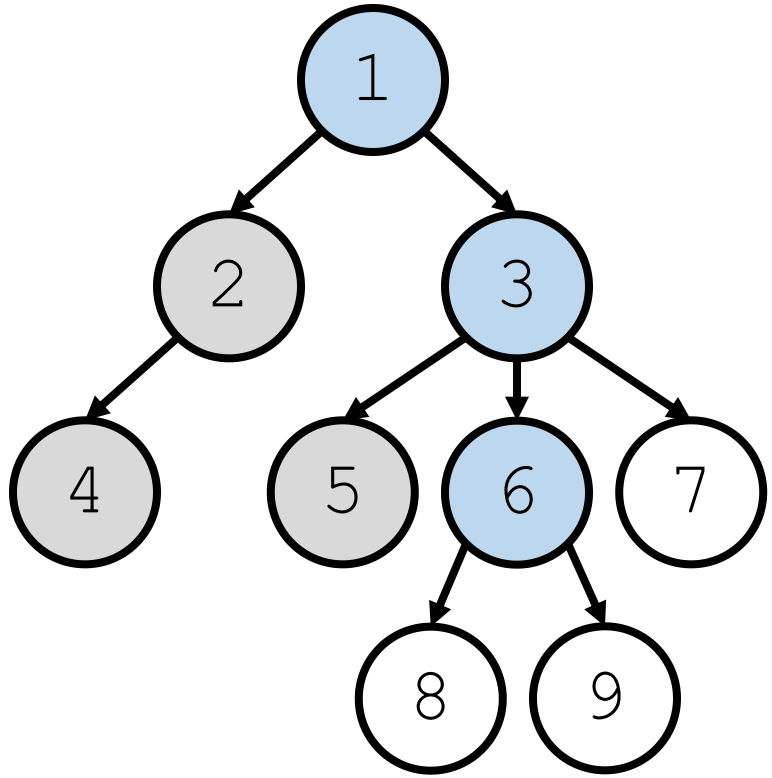
树-深度优先遍历

进入序列

1	2	4	3	5	6			

退出序列

4	2	5						



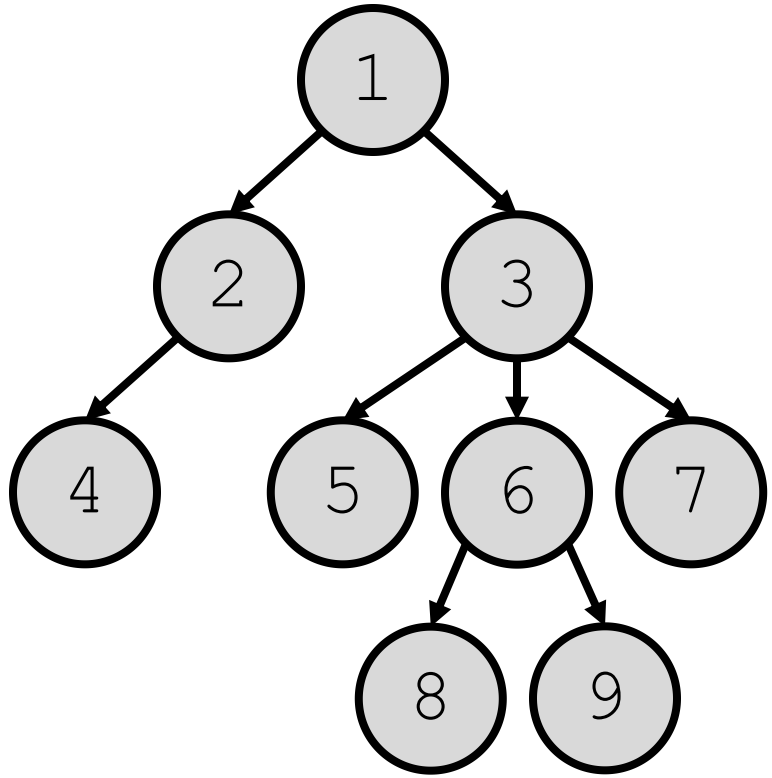
树-深度优先遍历

进入序列

1	2	4	3	5	6	8	9	7

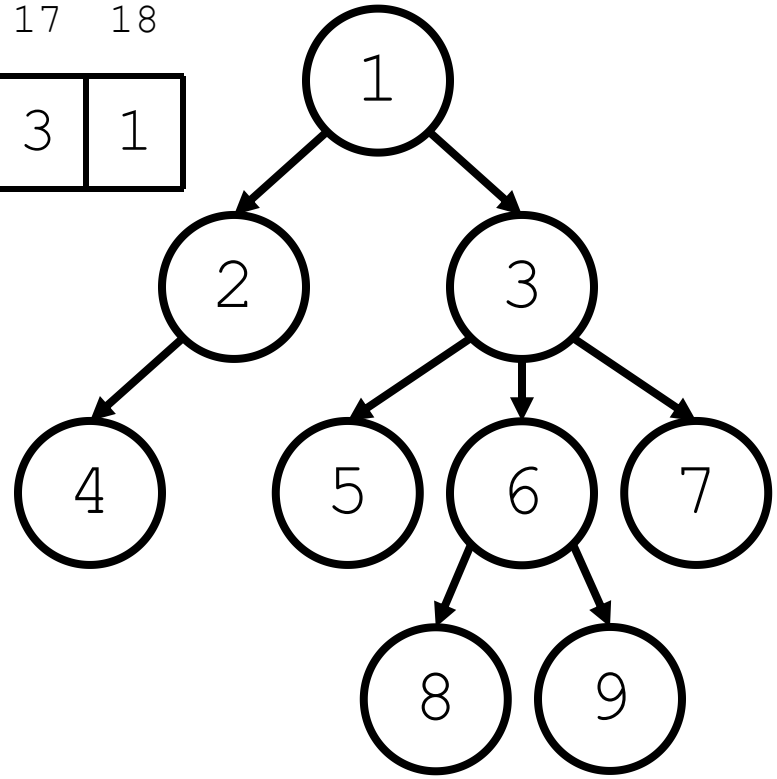
退出序列

4	2	5	8	9	6	7	3	1



树-深度优先遍历

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	2	4	4	2	3	5	5	6	8	8	9	9	6	7	7	3	1

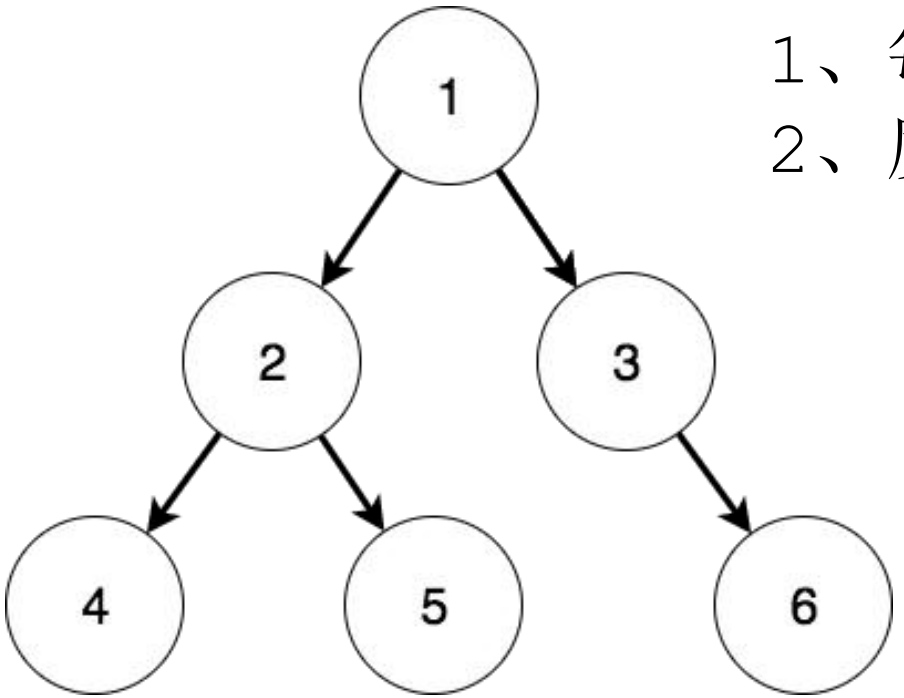


树-遍历总结

栈	树的深度遍历、深度优先搜索(图算法基础)
队列	树的层序遍历，广度优先搜索(图算法基础)

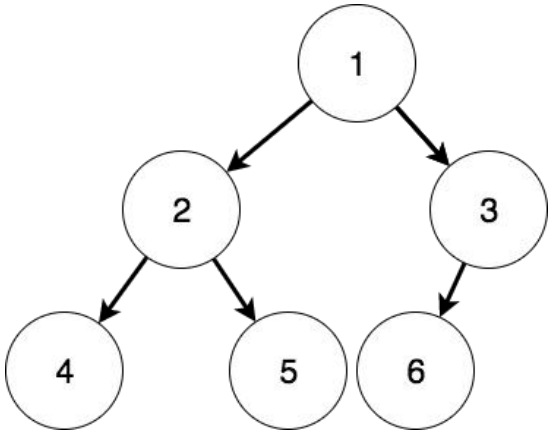
二. 二叉树：结构讲解

二叉树：结构讲解

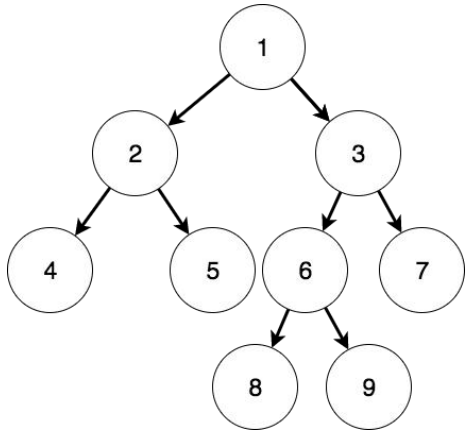


1、每个节点度最多为 2
2、度为0的节点比度为2的节点多1个

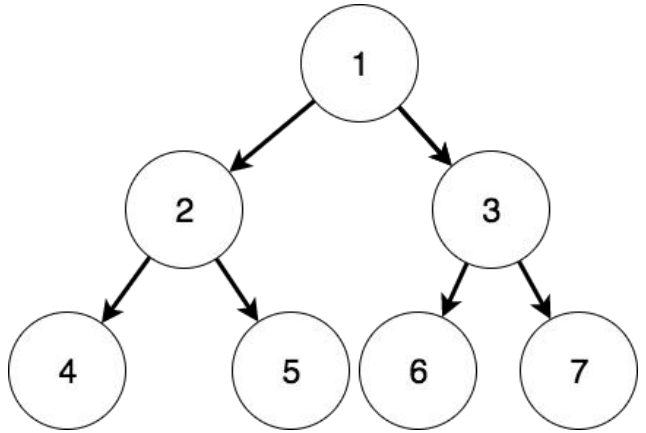
二叉树：特殊种类



完全二叉树
(complete binary tree)



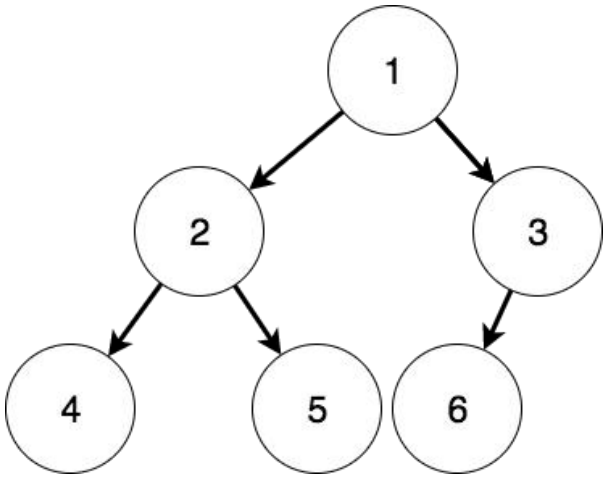
满二叉树
(full binary tree)



完美二叉树
(perfect binary tree)

二叉树：完全二叉树

完全二叉树
(complete binary tree)



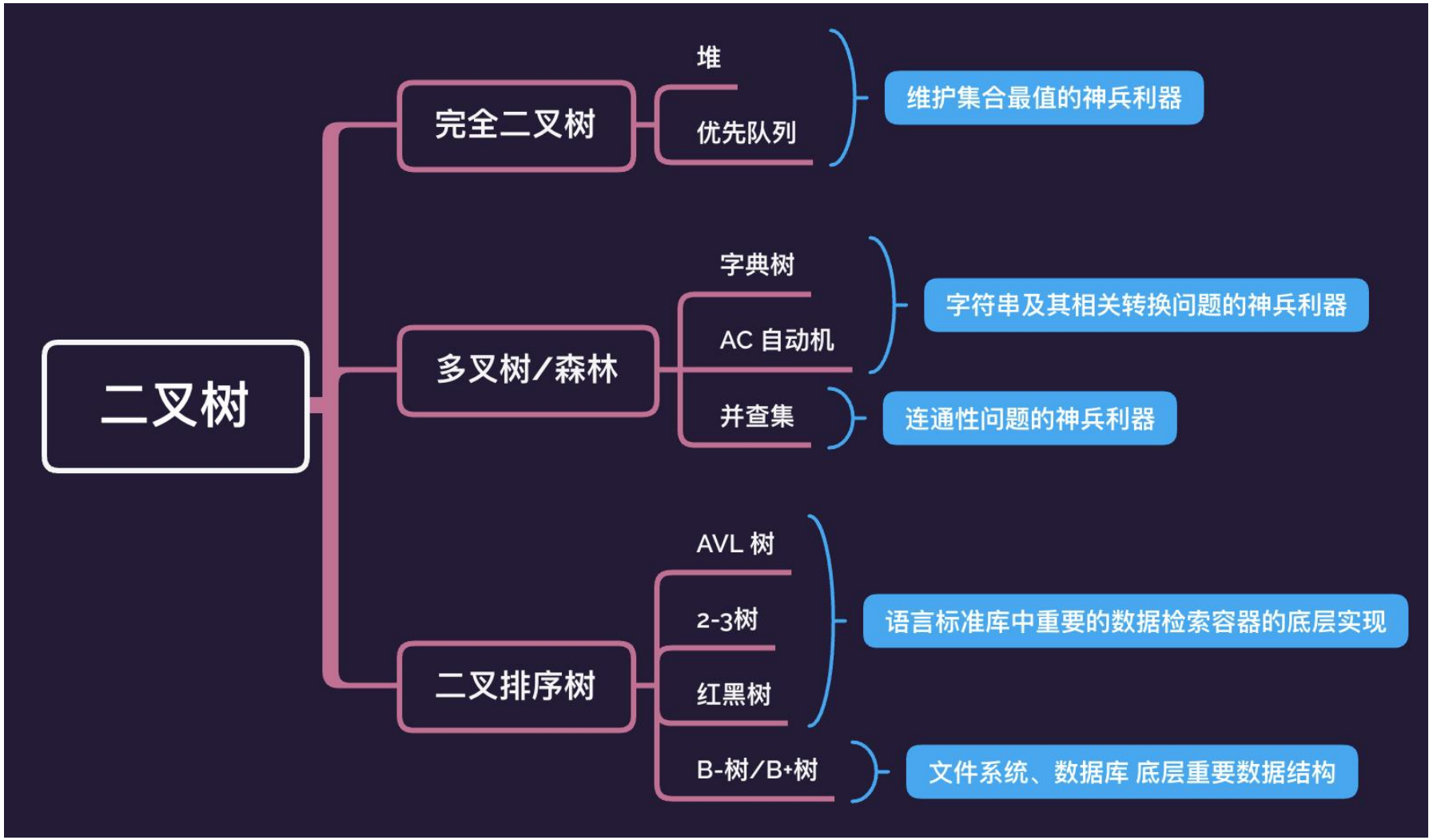
1、编号为 i 的子节点：

左孩子编号：2 * i

右孩子编号：2 * i + 1

2、可以用连续空间存储（数组）

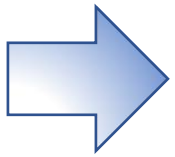
作用一：理解高级数据结构的基础



作用二：练习递归技巧的最佳选择

设计/理解递归程序：

1. 数学归纳法 → 结构归纳法
2. 赋予递归函数一个明确的意义
3. 思考边界条件
4. 实现递归过程

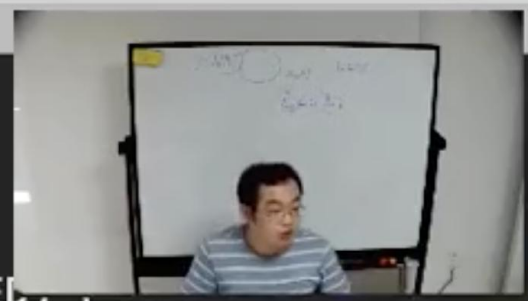


二叉树的前序遍历：

1. **函数意义：**前序遍历以 root 为根节点的二叉树
2. **边界条件：**root 为空时不需要遍历
3. **递归过程：**前序遍历左子树，前序遍历右子树

作用三：左孩子右兄弟表示法节省空间

```
37 /*
38  *            +------+
39  *            | node |
40  *            +------+
41  *          / <- parent
42  * +------+   v- sibling +------+
43  * | node | ------------ | node |
44  * +------+              +------+
45  *    | <- children          |
46  * +------+   +------+   +------+   +------+
47  * | node | - | node |   | node | - | node |
48  * +------+   +------+   +------+   +------+
49  */
50
51 /* TODO: Performance would benefit from a reorganization:
52  * (i) Allocate all children of a node within a single block.
53  * (ii) Keep all u stats together, and all amaf stats together.
54  * Currently, rave_update is top source of cache misses, and
55  * there is large memory overhead for having all nodes separate. */
```

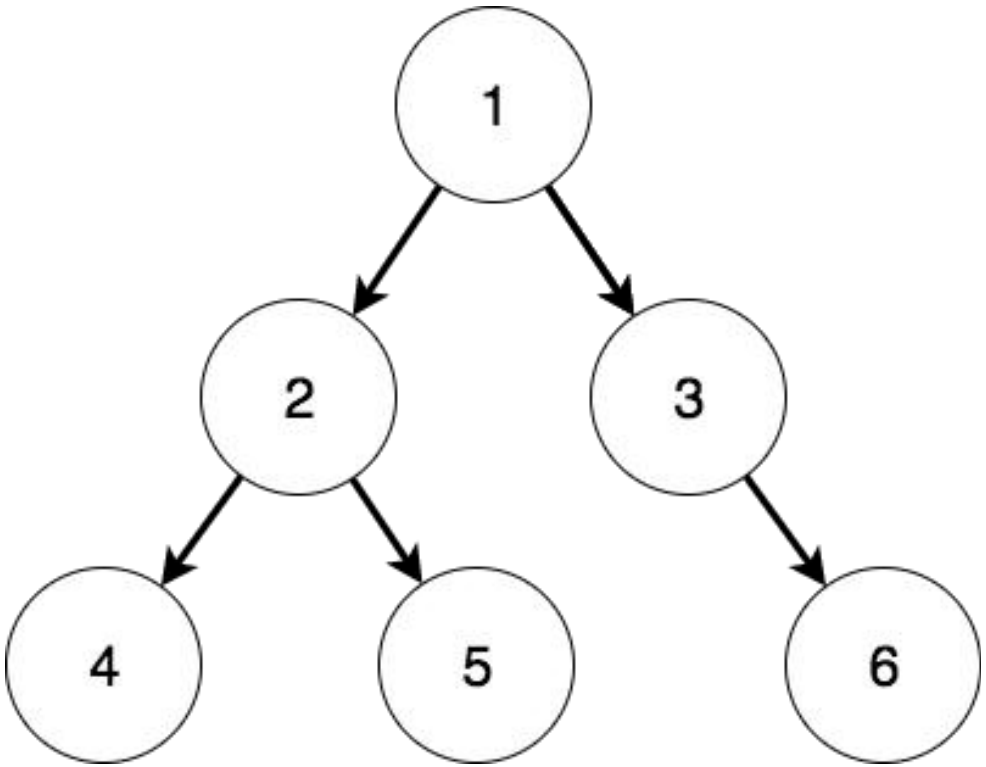


```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED) {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```


二叉树：代码演示

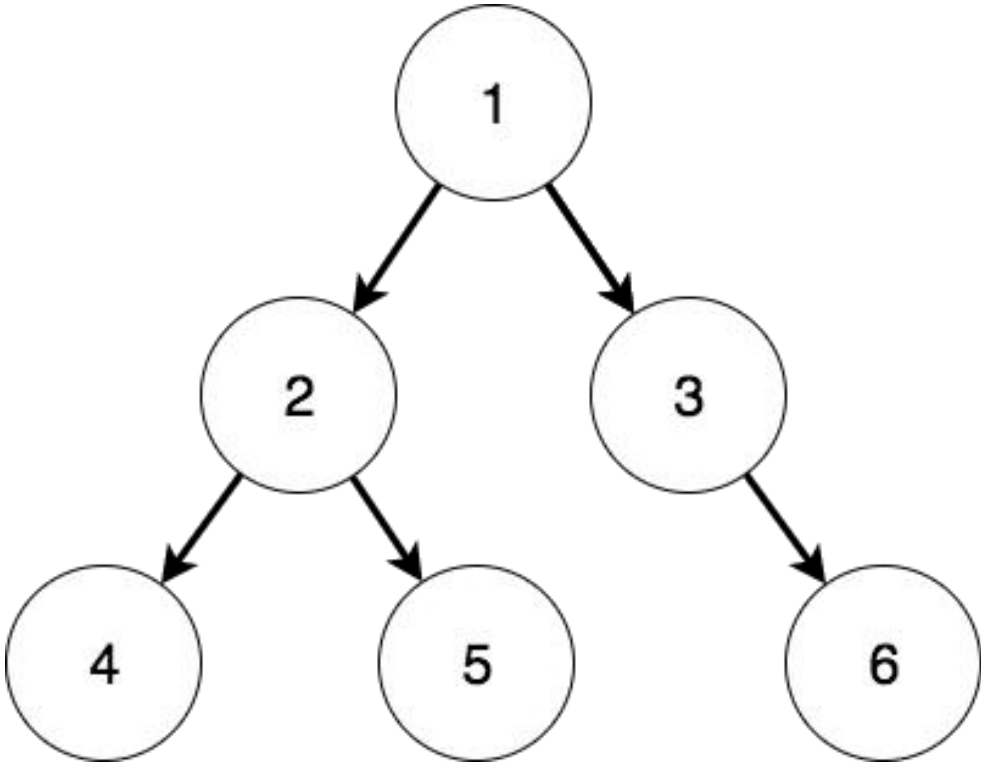
三. 二叉树：遍历与线索化

二叉树：遍历



前序遍历	根　左　右
中序遍历	左　根　右
后序遍历	左　右　根

二叉树：遍历



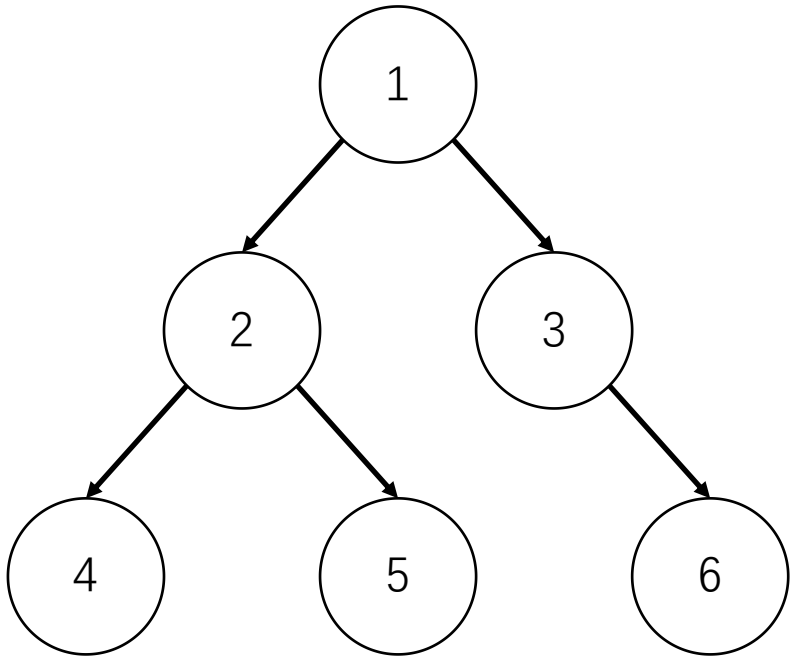
前序遍历	1 2 4 5 3 6
中序遍历	4 2 5 1 3 6
后序遍历	4 5 2 6 3 1

二叉树：线索化

左边空指针 → 前驱

右边空指针 → 后继

前序遍历	1 2 4 5 3 6
中序遍历	4 2 5 1 3 6
后序遍历	4 5 2 6 3 1

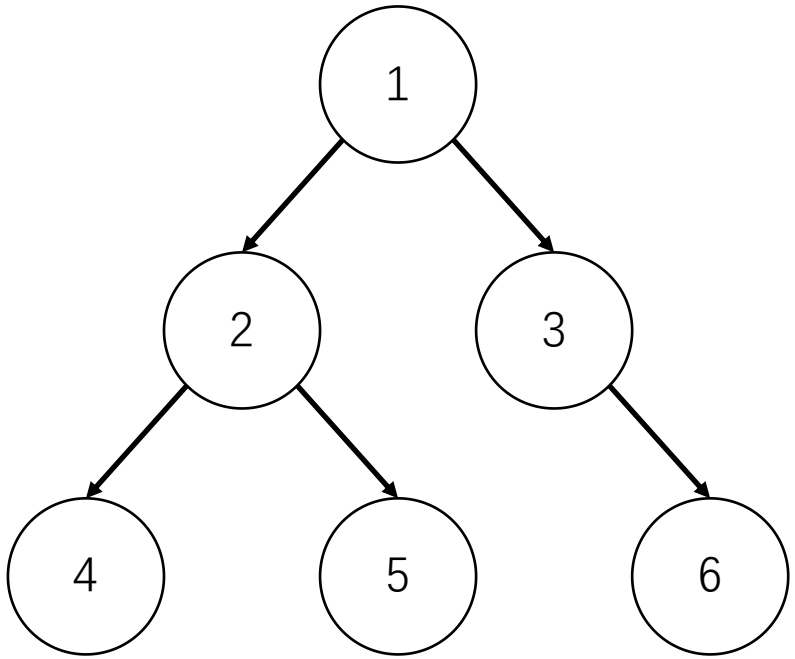


二叉树：线索化

左边空指针 → 前驱

右边空指针 → 后继

中序遍历	4 2 5 1 3 6

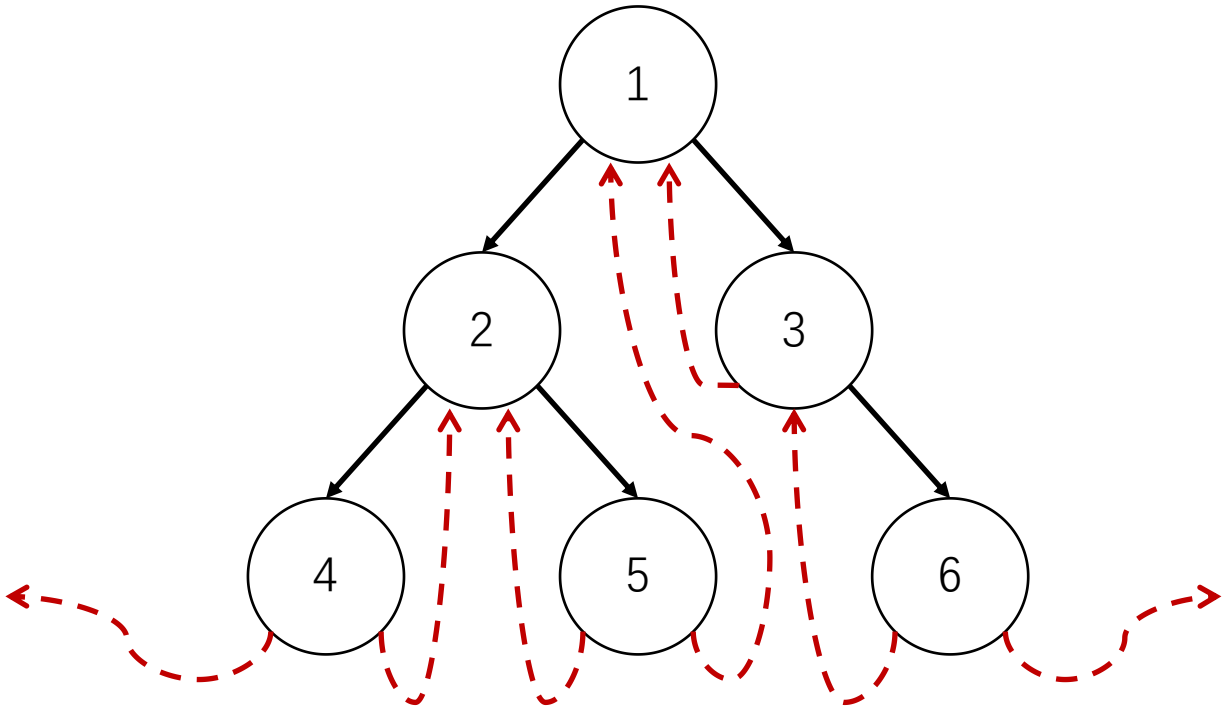


二叉树：线索化

左边空指针 → 前驱

右边空指针 → 后继

中序遍历	4 2 5 1 3 6




1. vim
vim ⌘1
bash ⌘2
bash ⌘3
39 }
40
41 Node *insert_maintain(Node *root) {
42 if (!hasRedChild(root)) return root;
43 if (root->lchild->color == RED && root->rchild->color == RED) {
44 if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45 root->color = RED;
46 root->lchild->color = root->rchild->color = BLACK;
47 return root;
48 }
49 if (root->lchild->color == RED) {
50 if (!hasRedChild(root->lchild)) return root;
51
52
53 } else {
54 if (!hasRedChild(root->rchild)) return root;
55
56 }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62 if (root == NIL) return getNewNode(key);
<-6班 资 料 /X.现 场 撸 代 码 /15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

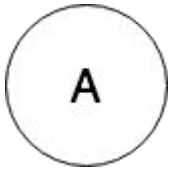
线索化：代码演示

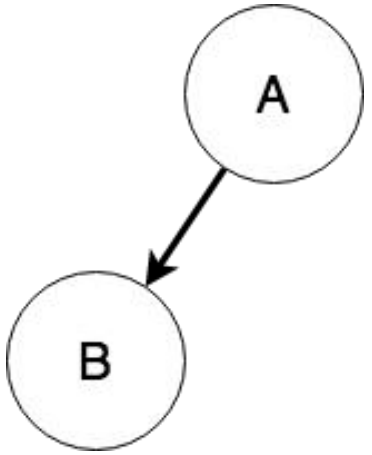
四. 二叉树的广义表表示法

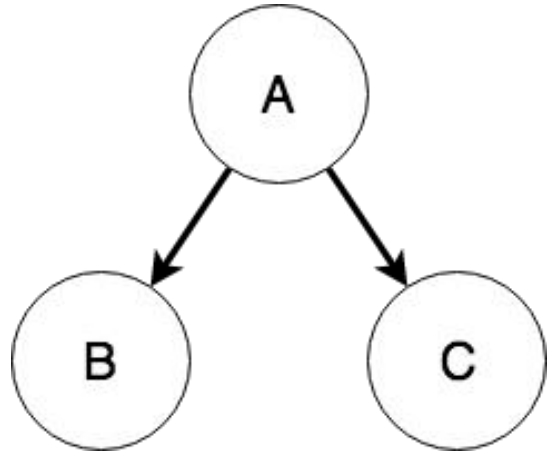
二叉树：广义表表示法

`()`　　`A / A()`　　`A(B,) / A(B)`　　`A(B,C)`

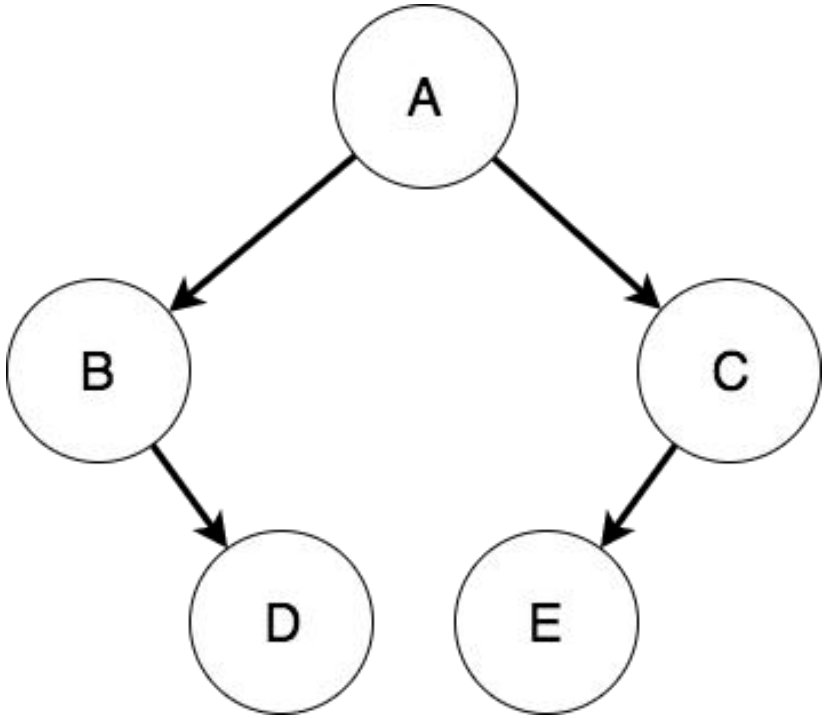
空树







二叉树：广义表表示法




```
A(B(,D),C(E))
A(B(,D),C(E,))
A(B(,D),C(E,()))
A(B(,D(())),C(E,))
```

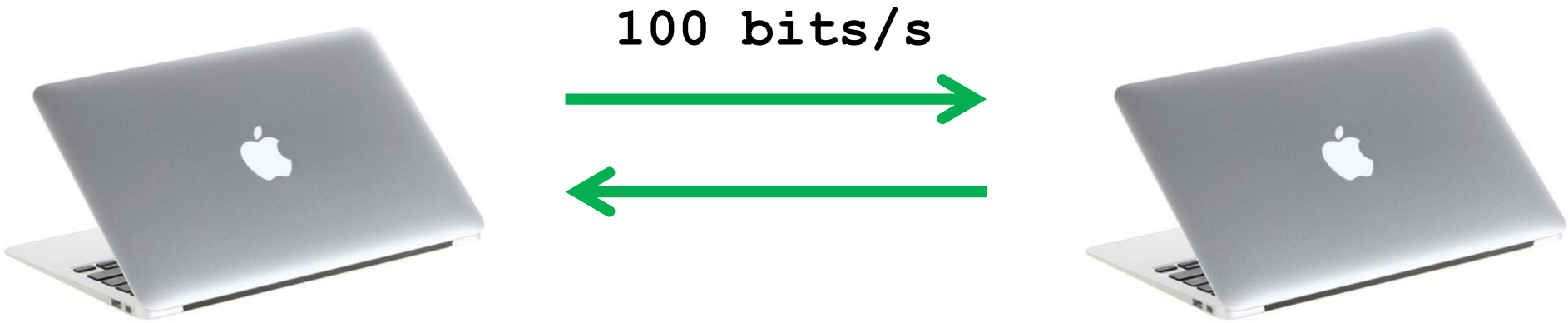
……


1. vim
vim ⌘1
bash ⌘2
bash ⌘3
39 }
40
41 Node *insert_maintain(Node *root) {
42 if (!hasRedChild(root)) return root;
43 if (root->lchild->color == RED && root->rchild->color == RED) {
44 if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45 root->color = RED;
46 root->lchild->color = root->rchild->color = BLACK;
47 return root;
48 }
49 if (root->lchild->color == RED) {
50 if (!hasRedChild(root->lchild)) return root;
51
52
53 } else {
54 if (!hasRedChild(root->rchild)) return root;
55
56 }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62 if (root == NIL) return getNewNode(key);
<-6班 资 料 /X.现 场 撸 代 码 /15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

广义表：代码演示

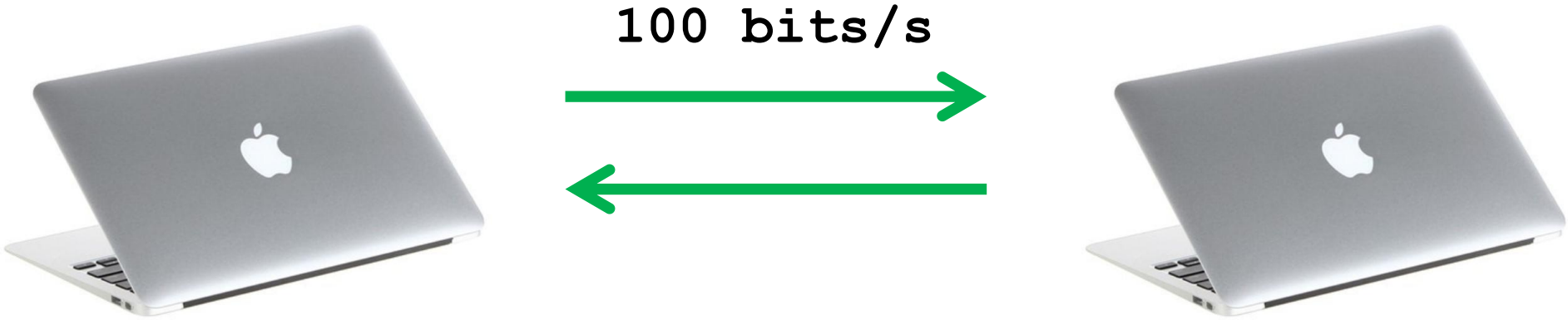
五. 最优变长编码：哈夫曼编码

前置知识：什么影响了网速的体感？



传输：a-z 范围的100个字符

前置知识：什么影响了网速的体感？



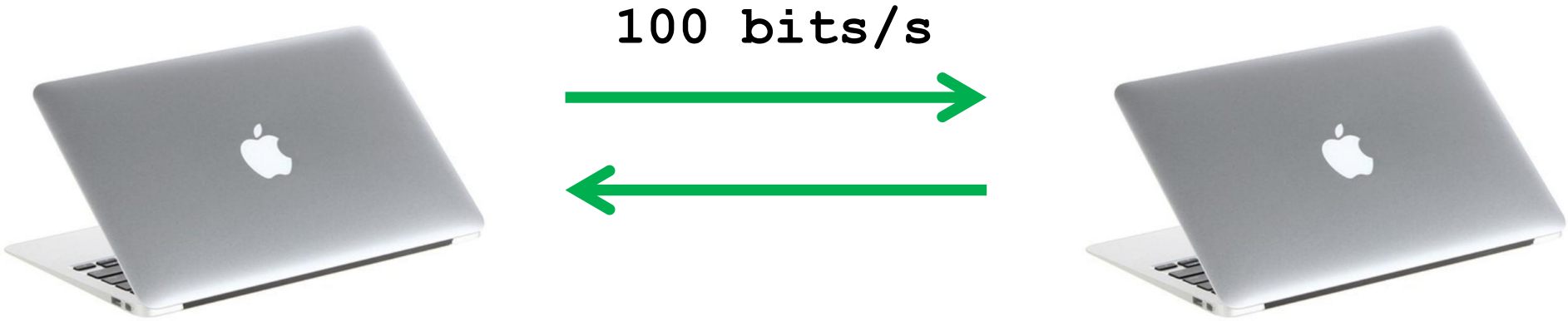
传输：a-z 范围的100个字符

编码方式：ASCII 编码

数据传输量：800 bits

传输时间：8 s

前置知识：什么影响了网速的体感？



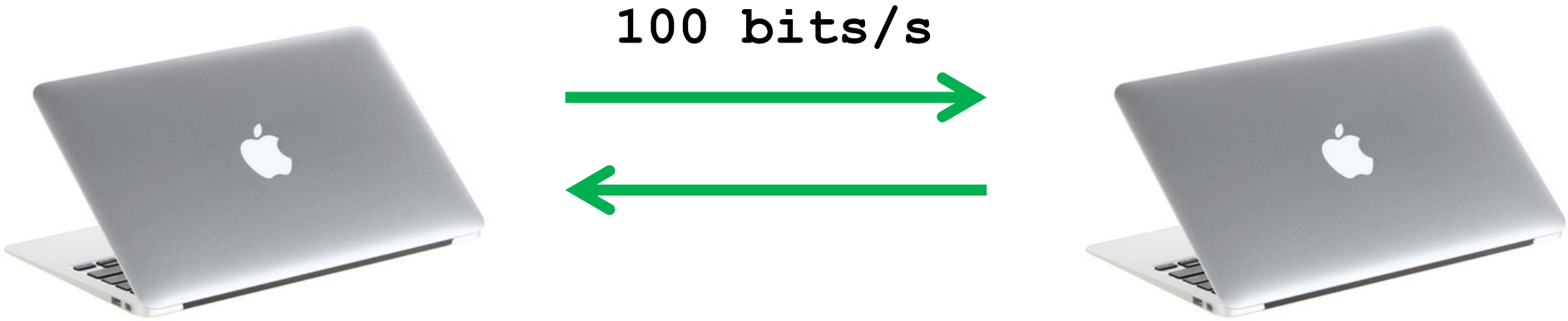
传输：a-z 范围的100个字符

编码方式：自定义

数据传输量：500 bits

传输时间：5 s

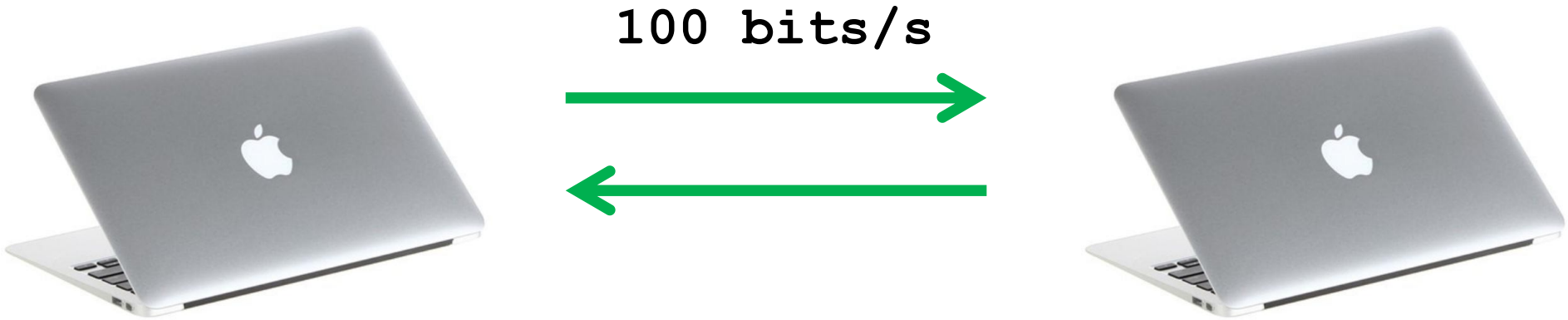
前置知识：什么影响了网速的体感？



传输：a-z 范围的100个字符

编码方式：ASCII 编码	编码方式：自定义
数据传输量：800 bits	数据传输量：500 bits
传输时间：8 s	传输时间：5 s

前置知识：为什么会有变长编码？



传输：a-d 范围的100个字符

其中 (a,0.5),(b,0.2),(c,0.1),(d,0.2)

前置知识：如何衡量两套编码的优劣？

平均编码长度：

l_i：第 i 种字符，编码长度

p_i：第 i 种字符，出现概率

$$avg(l) = \sum l_i \times p_i$$

(a,0.5),(b,0.2)

(c,0.1),(d,0.2)

哈夫曼编码

哈夫曼编码生成过程 ：

1.首先，统计得到每一种字符的概率
2.每次将最低频率的两个节点合并成一棵子树
3.经过了 n-1 轮合并，就得到了一棵哈夫曼树
4.按照左0，右1的形式，将编码读取出来

(a,0.5),(b,0.2)

(c,0.1),(d,0.2)

哈夫曼编码

哈夫曼编码生成过程 ：

1.首先，统计得到每一种字符的概率
2.每次将最低频率的两个节点合并成一棵子树
3.经过了 n-1 轮合并，就得到了一棵哈夫曼树
4.按照左0，右1的形式，将编码读取出来

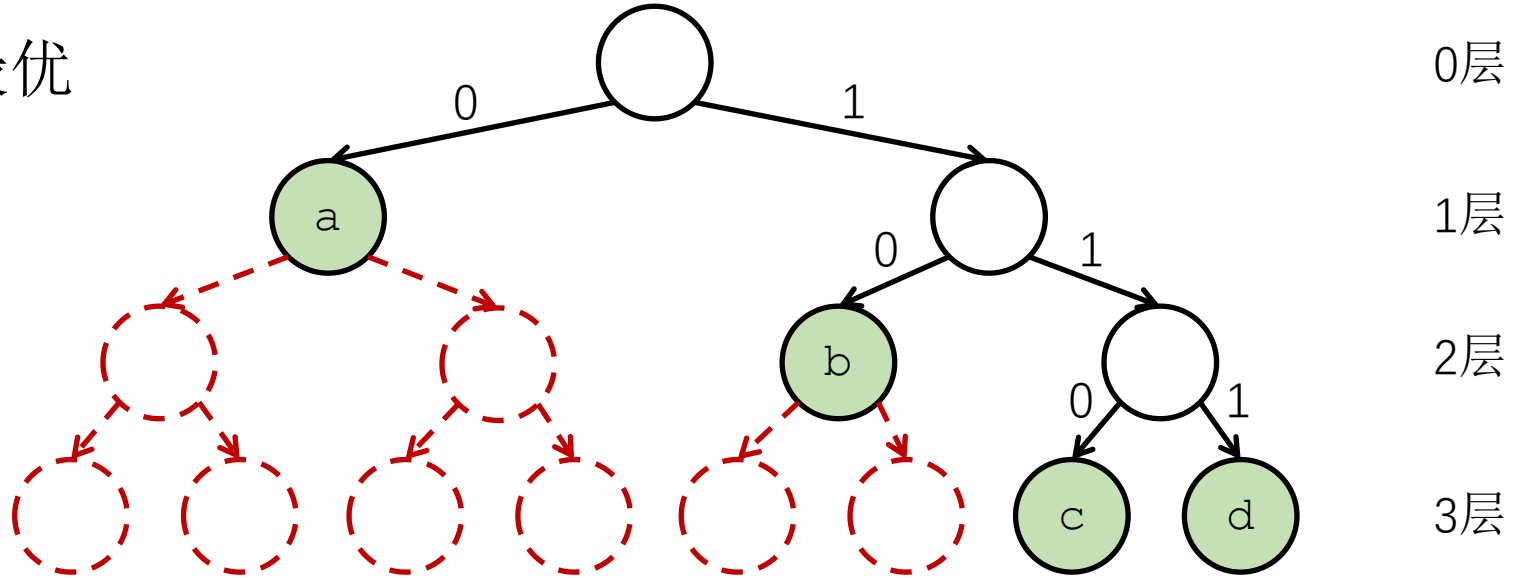
(a,0.5),(b,0.2)

(c,0.1),(d,0.2)

结论：哈弗曼编码，是最优的变长编码

哈夫曼编码：证明

证明 ： 平均编码长度最优



$$2^{H-L_1} + 2^{H-L_2} + 2^{H-L_3} + \ldots + 2^{H-L_n} <= 2^H$$

哈夫曼编码：证明

$2^{H-L_1} + 2^{H-L_2} + 2^{H-L_3} + \ldots + 2^{H-L_n} <= 2^H$

同时除2^H

$\frac{1}{2^{l_1}} + \frac{1}{2^{l_2}} + \frac{1}{2^{l_3}} + \ldots + \frac{1}{2^{l_n}}$ <= 1

设 $l'_i = -l_i$, 得到 : $2^{l'_1} + 2^{l'_2} + \ldots + 2^{l'_n}$ <= 1

所以得证明公式的约束条件 : $\sum 2^{l'_i}$ <= 1

证明 : $\sum p_i * l_i$ 最小

又因为 $l'_i = -l_i$ 所以等价于$-\sum p_i * l'_i$

在设$I_i = 2^{l'_i}$ => $l'_i = log_2 I_i$ => $-\sum p_i * log_2 I_i$

让这个式子达到最小值

同时得到 : 约束 $\sum I_i$ <= 1

哈夫曼编码：证明

目标：$-\sum p_i * log_2 I_i$ 达到最小值

目标函数展开：$-(P_1 log_2 I_1 + P_2 log_2 I_2 + \ldots + P_n log_2 I_n)$

约束条件为：$\sum I_i$ <= 1

证明当目标函数达到最小值的时候，想想看看什么情况下目标函数能达到最优解，让他最小

是不是需要让括号里面尽可能大，需要让$\sum I_i$ = 1

哈夫曼编码：证明

目标：$-\sum p_i * log_2 I_i$ 达到最小值

目标函数展开：$-(P_1 log_2 I_1 + P_2 log_2 I_2 + \ldots + P_n log_2 I_n)$

约束条件为：$\sum I_i$ <= 1

证明当目标函数达到最小值的时候，想想看看什么情况下目标函数能达到最优解，让他最小

是不是需要让括号里面尽可能大，需要让$\sum I_i$ = 1

反证：如果$\sum I_i < 1$的时候目标函数有最小的解

那么$I_1 + I_2 + I_3 + \ldots + I_n$ < 1，但是我们可以让他变成=1

$I_1 + I_2 + I_3 + \ldots + I_n + I'_x$ = 1 整个式子多了一个I'_x，并且是大于0的值

将I'_x加到目标函数$-(P_1 log_2 I_1 + P_2 log_2 I_2 + \ldots + P_n log_2 I_n)$里面

括号里面的内容变的更大了，那么整体就变的更小了

所以得到 $\sum I_i = 1$

哈夫曼编码：证明

继续证明

目标函数 : $-(P_1 log_2 I_1 + P_2 log_2 I_2 + \ldots + P_n log_2 I_n)$

当目标函数达到条件 : $I_1 + I_2 + I_3 + \ldots + I_n$ = 1

设 $I_n = 1 - II$

目标函数变成 :

$-(P_1 log_2 I_1 + P_2 log_2 I_2 + \ldots + P_n log_2 (1 - II))$

想让这个式子达到最小值, 对每一项求偏导, 让每一项偏导等于0

对I_1求偏导:

$$\frac{P_1}{I_1 ln2} - \frac{P_n}{(1-II) ln2} = 0$$

对I_2求偏导:

$$\frac{P_2}{I_2 ln2} - \frac{P_n}{(1-II) ln2} = 0$$

对I_3求偏导:

$$\frac{P_3}{I_3 ln2} - \frac{P_n}{(1-II) ln2} = 0$$

哈夫曼编码：证明

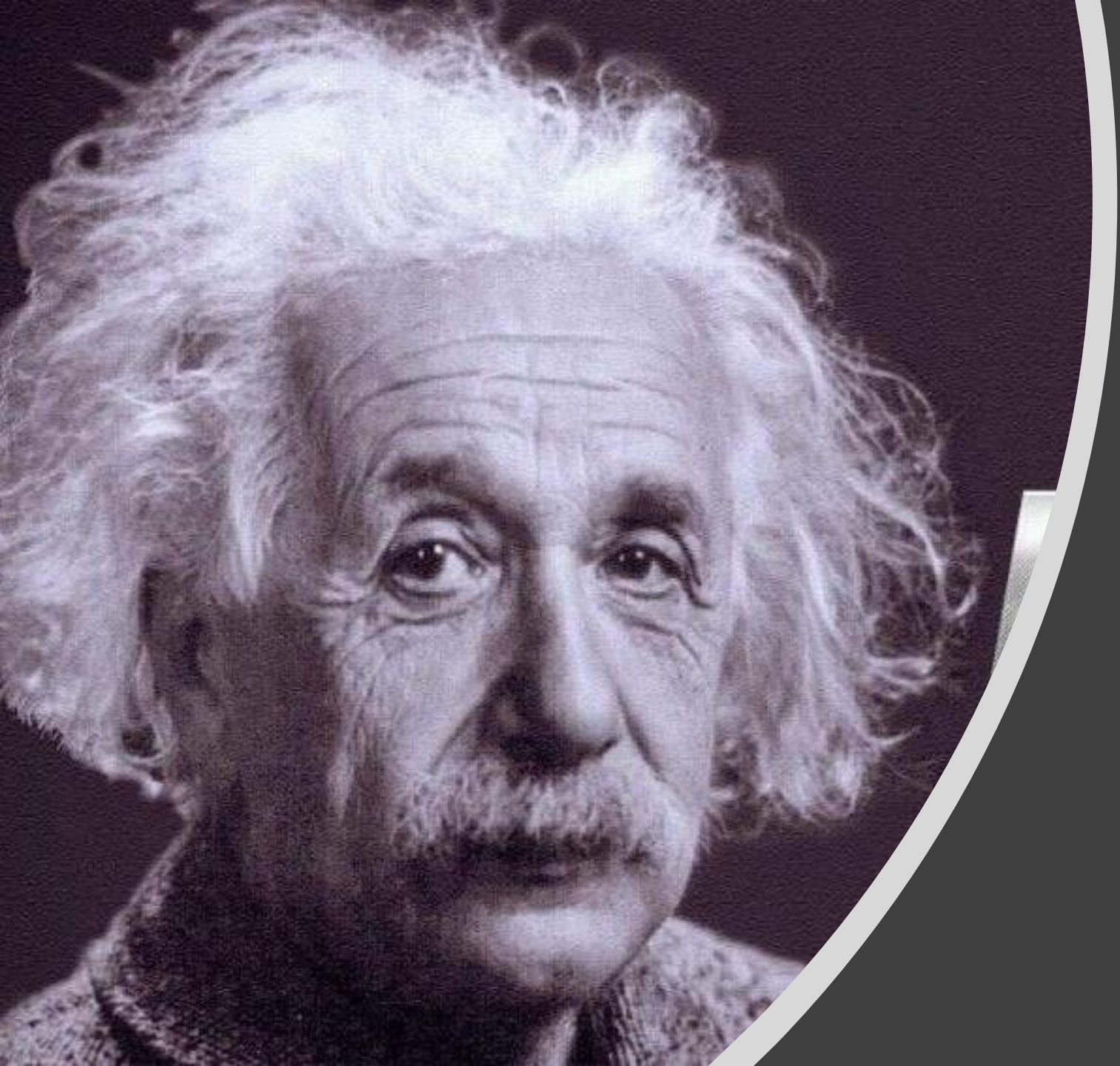
整理后得到：

$$\frac{P_1}{I_1} = \frac{P_2}{I_2} = \frac{P_3}{I_3} = \ldots = \frac{P_n}{I_n}$$

$$P_i = I_i = 2^{l'_i} = \frac{1}{2^{l_i}}$$

li是编码长度, P_i是字符概率

得到结论, 编码越大, 概率越小



为什么
会出一样的题目？