

1.树

1.1树的概念

1.2树的度

1.3树的边数和高度

1.4树概念相关的图

1.5树的逻辑结构

2.二叉树

2.1二叉树的概念

2.2二叉树的性质

2.3完全二叉树特性

2.4二叉树的顺序存储

2.5二叉树的链式存储

2.6完全二叉树的创建

2.7借助队列完成二叉树的遍历(BFS)

2.8二叉树的前中后序遍历（DFS）

2.9普通二叉树创建

3.递归

3.1使用递归实现n!阶乘

3.2使用递归实现单链表逆序

1.树

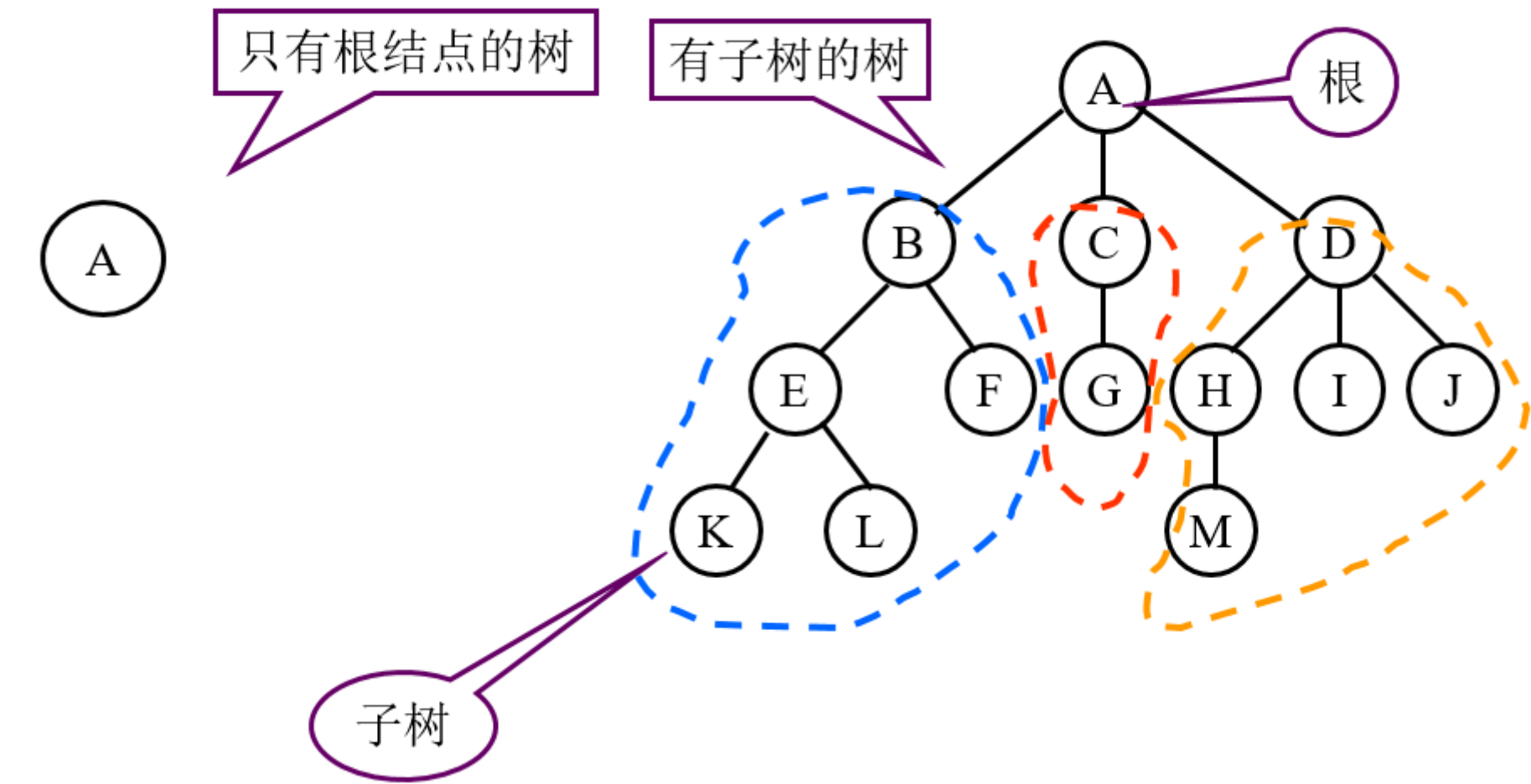
1.1树的概念

树（Tree）是n（n≥0）个节点的有限集合T，它满足两个条件：

1. 有且仅有一个特定的称为根（Root）的节点；

2. 其余的节点可以分为m（m≥0）个互不相交的有
限集合T1、T2、.....、Tm，其中每一个集合又是一棵树，
并称为其根的子树（Subtree）。

表示方法： 树形表示法、目录表示法。



1.2树的度

一个节点的子树的个数称为该节点的**度数**，一棵树的度数是指数该树中节点的最大度数。

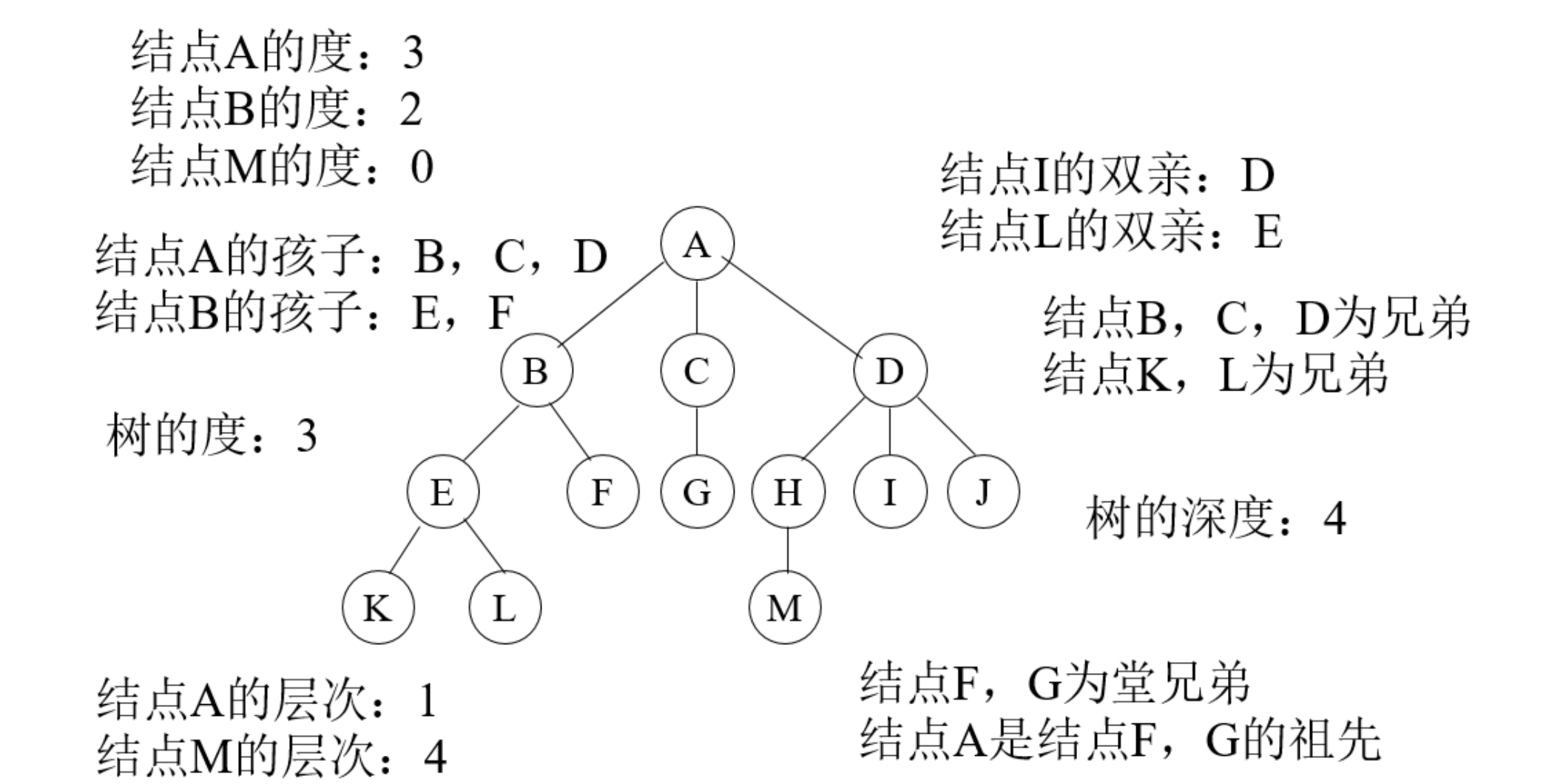
度数为零的节点称为**树叶**或**终端节点**，度数不为零的节点称为分支节点，除根节点外的分支节点称为内部节点。一个节点的子树之根节点称为该节点的**子节点**，该节点称为它们的**父节点**，同一节点的各个子节点之间称为兄弟节点。一棵树的根节点没有父节点，叶节点没有子节点。

1.3树的边数和高度

一个节点系列k1,k2, ,ki,ki+1, ,kj,并满足ki是ki+1的父节点，就称为一条从k1到kj的**路径**，路径的长度为j-1,即路径中的**边数**。路径中前面的节点是后面节点的祖先，后面节点是前面节点的子孙。节点的层数等于父节点的层数加一，根节点的层数定义为一。树中节点层数的最大值称为该树的**高度**或**深度**。若树中每个节点的各个子树的排列为从左到右，不能交换，即兄弟之间是有序的，则该树称为**有序树**。一般的树是有序树。

m（m≥0）棵互不相交的树的集合称为**森林**。树去掉根节点就成为森林，森林加上一个新的根节点就成为树。

1.4树概念相关的图



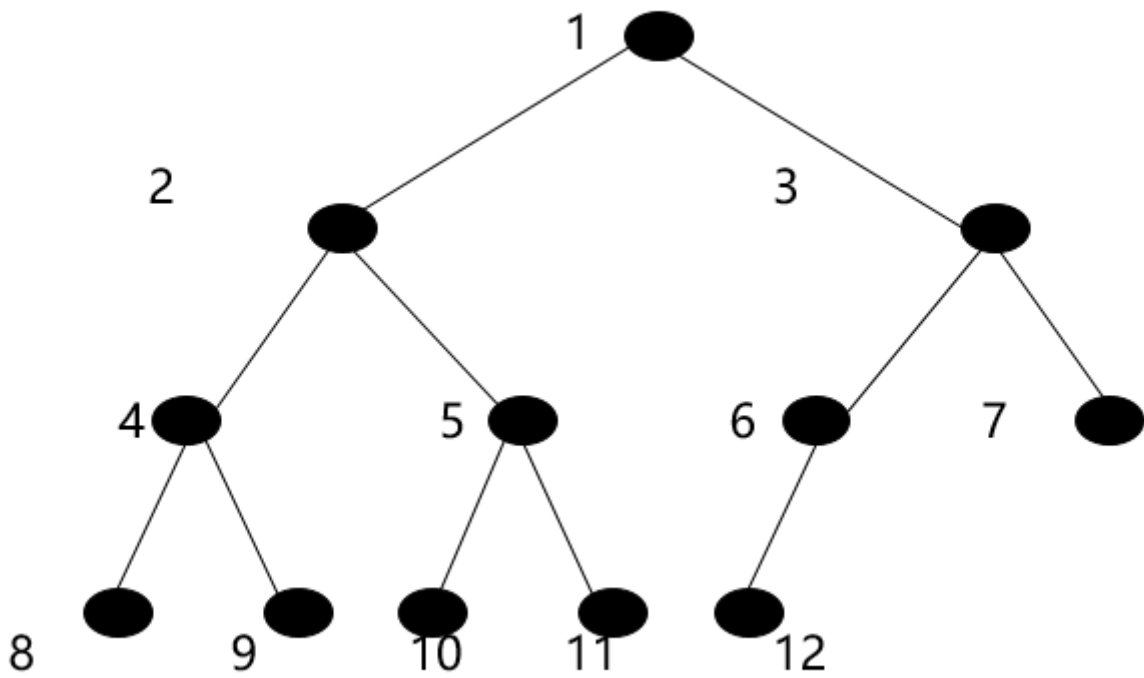
1.5树的逻辑结构

树的逻辑结构： 树中任何节点都可以有零个或多个直接后继节点（子节点），但至多只有一个直接前趋节点（父节点），根节点没有前趋节点，叶节点没有后继节点。

2.二叉树

2.1二叉树的概念

二叉树的定义： 二叉树（BinaryTree）是n（n≥0）个节点的有限集合，它或者是空集（n = 0），或者是由一个根节点以及两棵互不相交的、分别称为左子树和右子树的二叉树组成。二叉树与普通有序树不同，二叉树严格区分左孩子和右孩子，即使只有一个子节点也要区分左右。



2.2 二叉树的性质

- 1. 二叉树第 i ($i \geq 1$) 层上的节点最多为 2^{i-1} 个。
- 2. 深度为 k ($k \geq 1$) 的二叉树最多有 $2^k - 1$ 个节点。
- 3. 在任意一棵二叉树中，树叶的数目比度数为2的节点的数目多一。
总节点数为各类节点之和： $n = n_0 + n_1 + n_2$
总节点数为所有子节点数加一： $n = n_1 + 2 * n_2 + 1$
故得： $n_0 = n_2 + 1$ ；
- 4. **满二叉树**：深度为 k ($k \geq 1$) 时有 $2^k - 1$ 个节点的二叉树。
- 5. **完全二叉树**：只有最下面两层有度数小于2的节点，且最下面一层的叶节点集中在最左边的若干位置上。具有 n 个节点的完全二叉树的深度为 $(\log_2 n) + 1$ 或 $\log_2(n+1)$ 。

2.3 完全二叉树特性

完全二叉树节点的编号方法是从上到下，从左到右，根节点为1号节点。

设完全二叉树的节点数为 n ，某节点编号为 i

- 1. 当 $i > 1$ （不是根节点）时，有父节点，父节点编号为 $i/2$ ；
- 2. 当 $2*i \leq n$ 时，有左孩子，其编号为 $2*i$ ，否则没有左孩子，本身是叶节点；
- 3. 当 $2*i + 1 \leq n$ 时，有右孩子，其编号为 $2*i + 1$ ，否则没有右孩子；
- 4. 当 i 为奇数且不为1时，有左兄弟，其编号为 $i - 1$ ，否则没有左兄弟；
- 5. 当 i 为偶数且小于 n 时，有右兄弟，其编号为 $i - 1$ ，否则没有右兄弟；

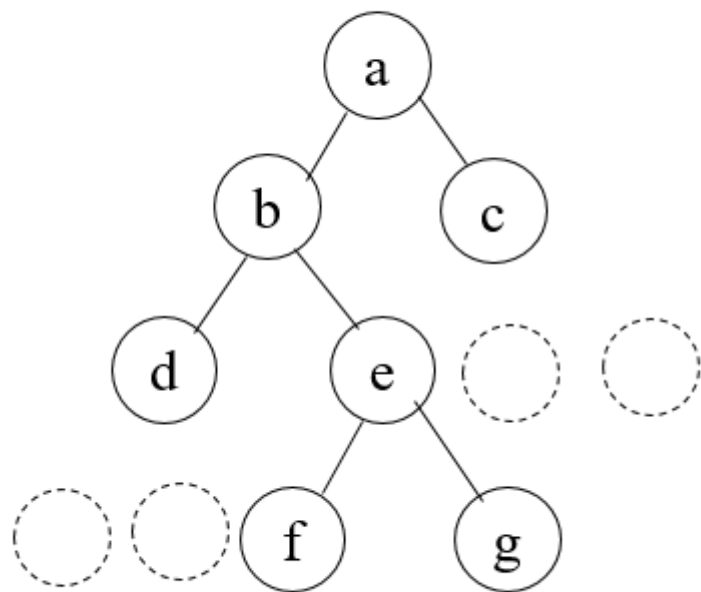
2.4 二叉树的顺序存储

有 n 个节点的完全二叉树可以用有 $n + 1$ 个元素的数组进行顺序存储，节点号

和数组下标——对应，下标为零的元素不用。

利用以上特性，可以从下标获得节点的逻辑关系。不完全二叉树通过添加虚

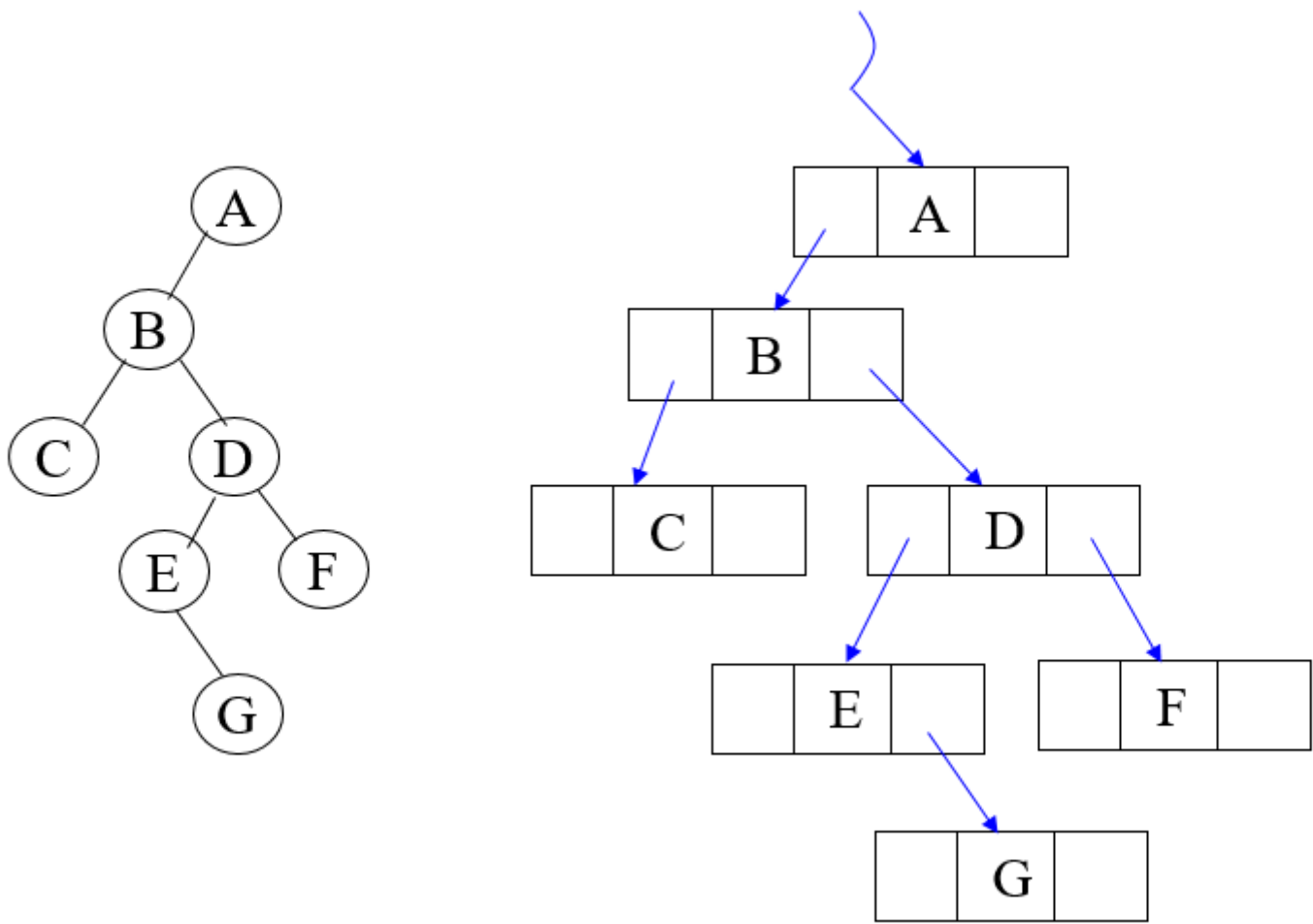
节点构成完全二叉树，然后用数组存储，这要浪费一些存储空间。



a	b	c	d	e	0	0	0	0	f	g
---	---	---	---	---	---	---	---	---	---	---

浪费空间

2.5 二叉树的链式存储



2.6 完全二叉树的创建

bitree.h

```
1  #ifndef __BITREE_H__
2  #define __BITREE_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define datatype int
8
9  typedef struct bitree{
10     datatype data;
11     struct bitree *lchild,*rchild;
12 }bitree_t;
13
14 // 创建一颗完全二叉树
15 // n是完全二叉树的节点个数
16 // i是节点的编号, i>=1
17 bitree_t * BiTreeCompleteCreate(int n,int i);
18 #endif
```

bitree.c

```

bitree_t* BiTreeCompleteCreate(int n, int i)
{
    bitree_t* root;
    root = (bitree_t*)malloc(sizeof(*root));
    root->data = i;

    2*1<=3
    if ((2 * i) <= n)
        root->lchild = BiTreeCompleteCreate(n, 2 * i);
    else
        root->lchild = NULL;

    2*1+1<=3
    if ((2 * i + 1) <= n)
        root->rchild = BiTreeCompleteCreate(n, 2 * i+1);
    else
        root->rchild = NULL;
    return root;
}

```

```

bitree_t* BiTreeCompleteCreate(int n, int i)
{
    bitree_t* root;
    root = (bitree_t*)malloc(sizeof(*root));
    root->data = i;

    2*2<=3
    if ((2 * i) <= n)
        root->lchild = BiTreeCompleteCreate(n, 2 * i);
    else
        root->lchild = NULL;

    2*2+1<=3
    if ((2 * i + 1) <= n)
        root->rchild = BiTreeCompleteCreate(n, 2 * i+1);
    else
        root->rchild = NULL;
    return root;
}

```

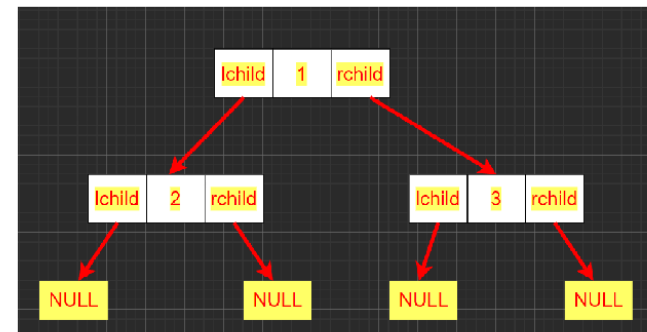
```

bitree_t* BiTreeCompleteCreate(int n, int i)
{
    bitree_t* root;
    root = (bitree_t*)malloc(sizeof(*root));
    root->data = i;

    2*3<=3
    if ((2 * i) <= n)
        root->lchild = BiTreeCompleteCreate(n, 2 * i);
    else
        root->lchild = NULL;

    2*3+1<=3
    if ((2 * i + 1) <= n)
        root->rchild = BiTreeCompleteCreate(n, 2 * i+1);
    else
        root->rchild = NULL;
    return root;
}

```



```

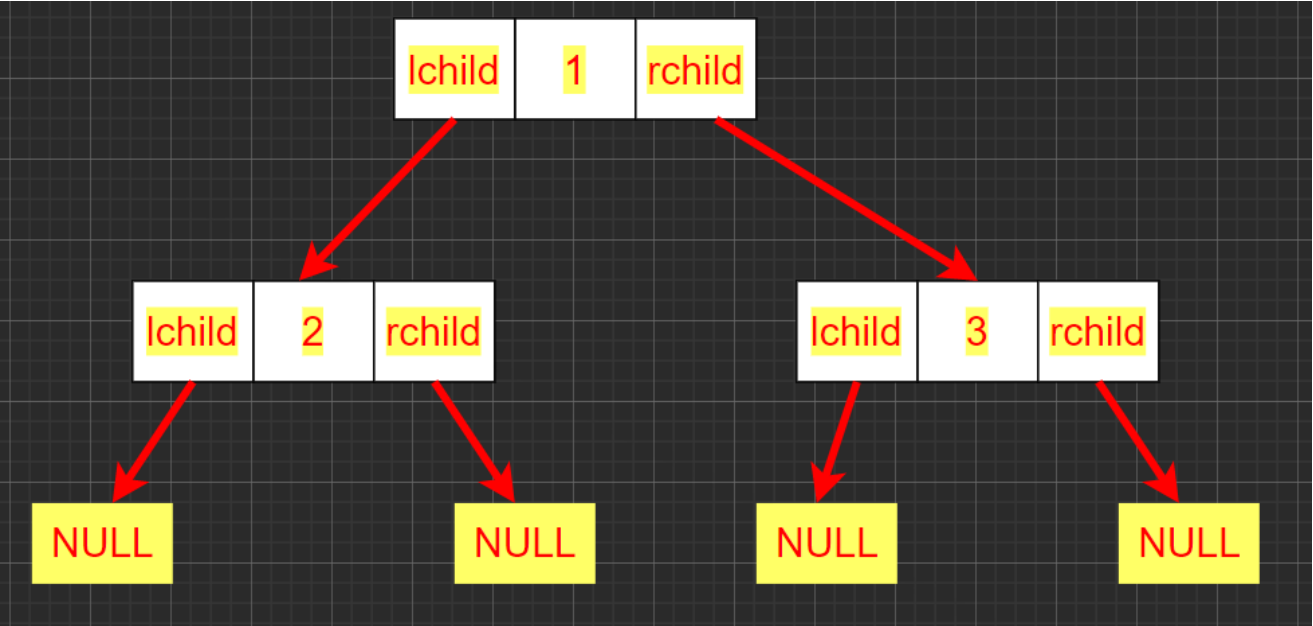
1 #include "bitree.h"
2
3 bitree_t* BiTreeCompleteCreate(int n, int i)
4 {
5     bitree_t* root;
6
7     root = (bitree_t*)malloc(sizeof(*root));
8     if (root == NULL) {
9         printf("%s malloc memory error\n", __func__);
10        return NULL;
11    }
12    root->data = i; // 将根节点的编号存入
13    // 当2*i≤n时, 有左孩子, 其编号为2*i , 否则没有左孩子, 本身是叶节点;
14    if ((2 * i) <= n) {
15        // 有左孩子
16        root->lchild = BiTreeCompleteCreate(n, 2 * i);
17    } else {
18        // 没有左孩子
19        root->lchild = NULL;
20    }
21    // 当2*i+1≤n时, 有右孩子, 其编号为2*i+1 , 否则没有右孩子;
22    if ((2 * i + 1) <= n) {
23        // 有右孩子
24        root->rchild = BiTreeCompleteCreate(n, 2 * i+1);
25    } else {
26        // 没有右孩子
27        root->rchild = NULL;
28    }
29
30    return root;
31 }

```

main.c

```
1  #include "bitree.h"
2
3  int main(int argc, const char* argv[])
4  {
5      bitree_t* root;
6
7      root = BiTreeCompleteCreate(7, 1);
8      if (root == NULL)
9          return -1;
10
11     return 0;
12 }
```

2.7借助队列完成二叉树的遍历(BFS)



遍历的思想：先让根节点入队，让根节点出队，打印根节点的值，判断根节点是否有左右孩子，如果有左右孩子让他们入队。先让根节点的左孩子出队，打印左孩子的值，在判断它是否有左右孩子，如果没有就不入队了。先让根节点的右孩子出队，打印右孩子的值，判断它是否有左右孩子，如果没有就不入队了。此时队列为空，循环退出即可。

bitree.h

```
1  #ifndef __BITREE_H__
2  #define __BITREE_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define datatype int
8
9  typedef struct bitree{
10     datatype data;
11     struct bitree *lchild,*rchild;
12 }bitree_t;
13
14 // 创建一颗完全二叉树
15 // n是完全二叉树的节点个数
16 // i是节点的编号，i>=1
17 bitree_t * BiTreeCompleteCreate(int n,int i);
18 #endif
```

bitree.c

```
1  #include "bitree.h"
2  // BiTreeCompleteCreate(3, 1)
3  //      1
4  //     / \
5  //    2   3
6  bitree_t* BiTreeCompleteCreate(int n, int i)
7  {
8      bitree_t* root;
9
10     root = (bitree_t*)malloc(sizeof(*root));
11     if (root == NULL) {
12         printf("%s malloc memory error\n", __func__);
13         return NULL;
14     }
```

```
15     root->data = i; // 将根节点的编号存入
16     // 当2*i≤n时，有左孩子，其编号为2*i，否则没有左孩子，本身是叶节点；
17     if ((2 * i) <= n) {
18         // 有左孩子
19         root->lchild = BiTreeCompleteCreate(n, 2 * i);
20     } else {
21         // 没有左孩子
22         root->lchild = NULL;
23     }
24     // 当2*i+1≤n时，有右孩子，其编号为2*i+1，否则没有右孩子；
25     if ((2 * i + 1) <= n) {
26         // 有右孩子
27         root->rchild = BiTreeCompleteCreate(n, 2 * i+1);
28     } else {
29         // 没有右孩子
30         root->rchild = NULL;
31     }
32
33     return root;
34 }
35
```

linkqueue.h

```
1  #ifndef __LINKQUEUE_H__
2  #define __LINKQUEUE_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define datatype int
8  typedef struct node {
9      datatype data;
10     struct node* next;
11 } node_t;
12
13 typedef struct {
14     node_t *front, *rear;
15 } linkqueue_t;
16
17 linkqueue_t* LinkQueueCreate(void);
18 int LinkQueueEnQueue(linkqueue_t* q, datatype data);
19 int LinkQueueIsEmpty(linkqueue_t* q);
20 datatype LinkQueueDeQueue(linkqueue_t* q);
21 void LinkQueueShow(linkqueue_t* q);
22 #endif
```

linkqueue.c

```
1  #include "linkqueue.h"
2
3  linkqueue_t* LinkQueueCreate(void)
4  {
5      node_t* h;
6      linkqueue_t* q;
7      h = (node_t*)malloc(sizeof(*h));
8      if (h == NULL) {
9          printf("%s malloc node_t error\n", __func__);
10         return NULL;
11     }
12     h->data = (datatype)0;
13     h->next = NULL;
14
15     q = (linkqueue_t*)malloc(sizeof(*q));
16     if (q == NULL) {
17         printf("%s malloc linkqueue_t error\n", __func__);
18         return NULL;
19     }
20     q->front = q->rear = h;
21
22     return q;
23 }
24
25 int LinkQueueEnQueue(linkqueue_t* q, datatype data)
26 {
27     node_t* tmp;
28     // 1.分配tmp(node_t*)
29     tmp = (node_t*)malloc(sizeof(*tmp));
30     if (tmp == NULL) {
31         printf("%s malloc node_t memory error\n", __func__);
32         return -1;
33     }
34     tmp->data = data;
```

```

35 // 2.将tmp入队
36 tmp->next = q->rear->next;
37 q->rear->next = tmp;
38 // 3.让rear记录当前队列队尾
39 q->rear = tmp;
40
41 return 0;
42 }
43 int LinkQueueIsEmpty(linkqueue_t* q)
44 {
45     return q->front == q->rear ? 1 : 0;
46 }
47
48 datatype LinkQueueDeQueue(linkqueue_t *q)
49 {
50     node_t *tmp;
51     datatype data;
52     if(LinkQueueIsEmpty(q)){
53         printf("%s linkqueue empty\n",__func__);
54         return (datatype)-1;
55     }
56
57     tmp = q->front->next;
58     q->front->next = tmp->next;
59     if(tmp->next == NULL){
60         q->rear = q->front;
61     }
62
63     data = tmp->data;
64     if(tmp!=NULL){
65         free(tmp);
66         tmp = NULL;
67     }
68
69     return data;
70 }
71
72 void LinkQueueShow(linkqueue_t* q)
73 {
74     node_t *h = q->front;
75
76     while(h->next){
77         printf("%d",h->next->data);
78         h = h->next;
79     }
80     printf("-\n");
81 }

```

main.c

```

1  #include "bitree.h"
2  #include "linkqueue.h"
3
4  int main(int argc, const char* argv[])
5  {
6      bitree_t *root, *tmp;
7      linkqueue_t* q;
8      // 1.创建完全二叉树
9      root = BiTreeCompleteCreate(9, 1);
10     if (root == NULL)
11         return -1;
12     // 2.创建队列
13     q = LinkQueueCreate();
14     if (q == NULL)
15         return -1;
16     // 3.将根节点入队
17     LinkQueueEnQueue(q, (datatype)root);
18
19     // 4.循环遍历,队列为空退出,不为空循环
20     while (!LinkQueueIsEmpty(q)) {
21         tmp = (bitree_t*)LinkQueueDeQueue(q); // 出队
22         printf(" %d", tmp->data); // 访问数据
23
24         if (tmp->lchild != NULL) { //如果有左孩子让左孩子入队
25             LinkQueueEnQueue(q, (datatype)tmp->lchild);
26         }
27         if (tmp->rchild != NULL) { //如果有右孩子让右孩子入队
28             LinkQueueEnQueue(q, (datatype)tmp->rchild);
29         }
30     }
31     printf("\n");
32     return 0;
33 }

```


2.8二叉树的前中后序遍历（DFS）

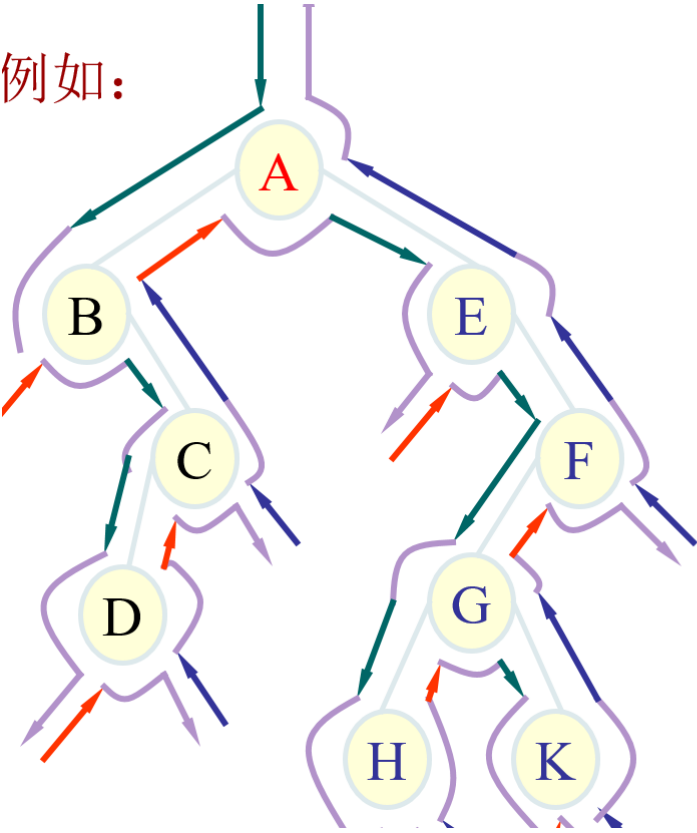
2.8.1先中后序遍历的思想

先序遍历（根左右）：先访问树根，再访问左子树，最后访问右子树；

中序遍历（左根右）：先访问左子树，再访问树根，最后访问右子树；

后序遍历（左右根）：先访问左子树，再访问右子树，最后访问树根；

2.8.2先中后序遍历的实例1



先序序列：

A B C D E F G H K

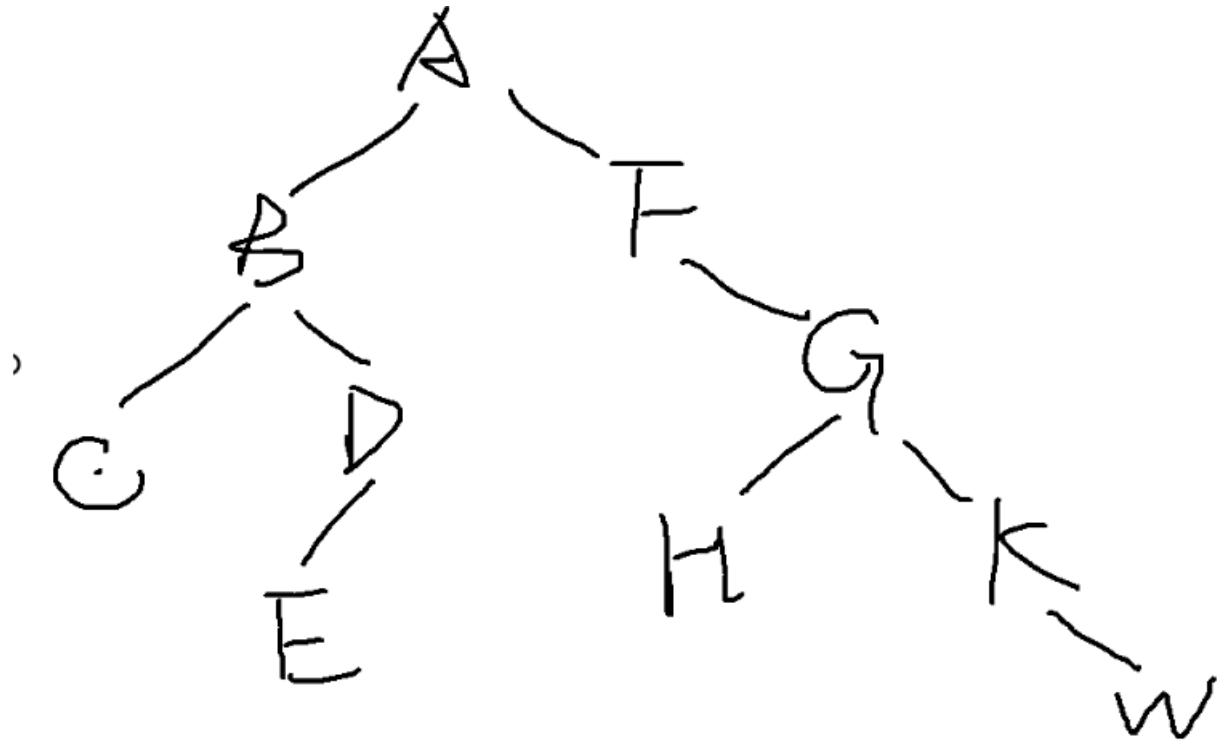
中序序列：

B D C A E H G K F

后序序列：

D C B H K G F E A

2.8.3先中后序遍历实例2

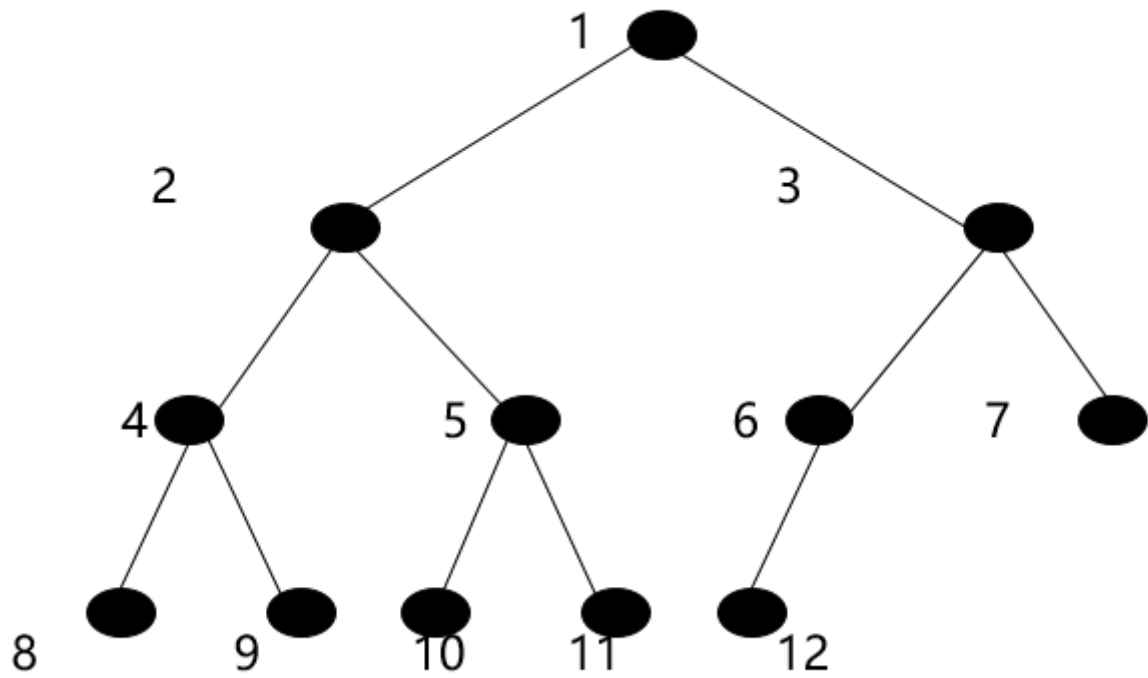


先序（根左右）：A B C D E F G H K W

中序（左根右）：C B E D A F H G K W

后序（左右根）：C E D B H W K G F A

2.8.4先中后序遍历的实例3



先: 1 2 4 8 9 5 10 11 3 6 12 7

中: 8 4 9 2 10 5 11 1 12 6 3 7

后: 8 9 4 10 11 5 2 12 6 7 3 1

2.8.5先中后序遍历的实例4

给出先和中序遍历，推导后序遍历

给出中和后序遍历，推导先序遍历

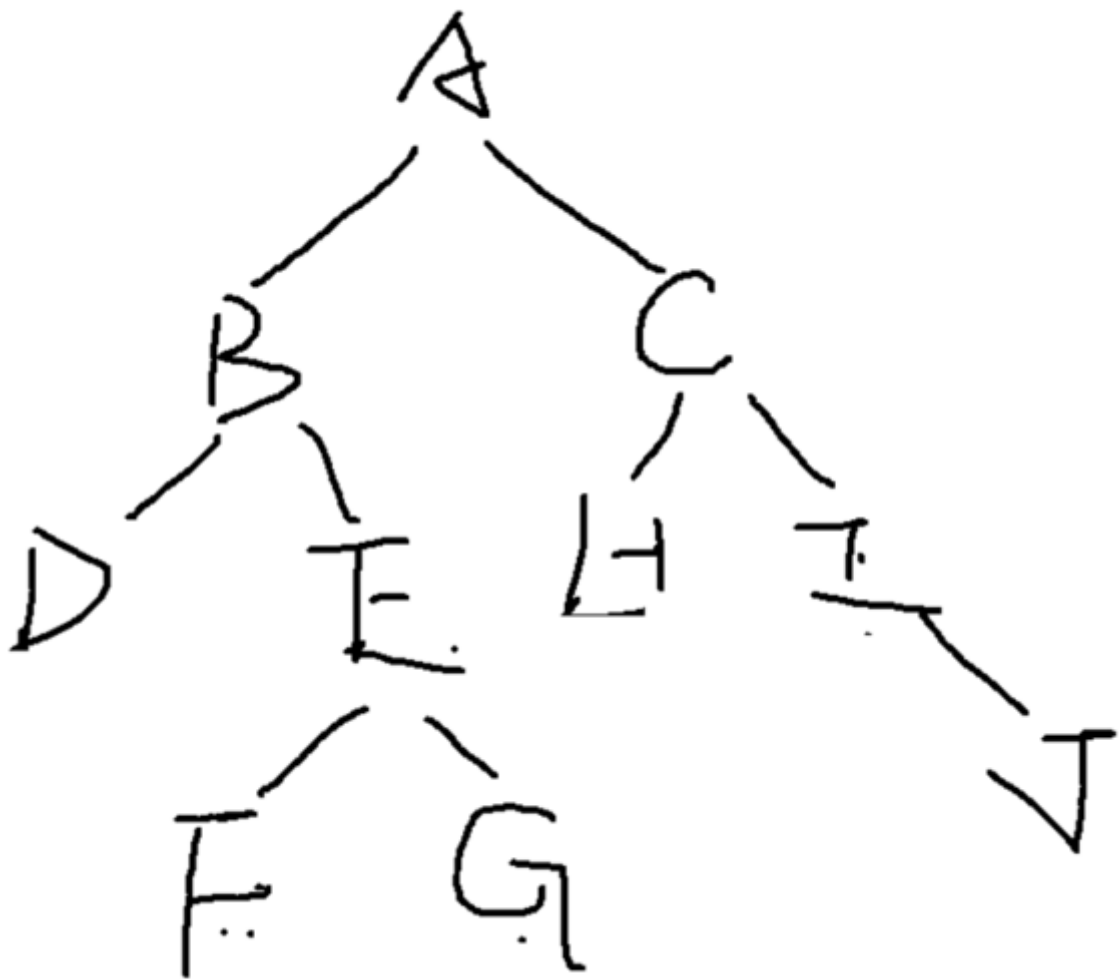
注：如果让你推导二叉树的遍历方式，一定会给出中序遍历。

先序序列：

A B D E F G C H I J

中序序列：

D B F E G A H C I J



后序遍历的结果：D F G E B H J I C A

2.8.6先中后序遍历代码实现

bitree.h

```
1  #ifndef __BITREE_H__
2  #define __BITREE_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define datatype int
8
9  typedef struct bitree{
10     datatype data;
11     struct bitree *lchild,*rchild;
12 }bitree_t;
13
14 // 创建一颗完全二叉树
15 // n是完全二叉树的节点个数
16 // i是节点的编号, i>=1
17 bitree_t * BiTreeCompleteCreate(int n,int i);
18 void BiTreeFrontOrder(bitree_t *root);
19 void BiTreeMiddleOrder(bitree_t *root);
20 void BiTreeRearOrder(bitree_t *root);
21 #endif
```

bitree.c

```
1  #include "bitree.h"
2  // BiTreeCompleteCreate(3, 1)
3  //      1
4  //    /  \
5  //   2    3
6  bitree_t* BiTreeCompleteCreate(int n, int i)
7  {
8     bitree_t* root;
9
10     root = (bitree_t*)malloc(sizeof(*root));
11     if (root == NULL) {
12         printf("%s malloc memory error\n", __func__);
13         return NULL;
14     }
15     root->data = i; // 将根节点的编号存入
16     // 当2*i≤n时, 有左孩子, 其编号为2*i ,否则没有左孩子, 本身是叶节点;
17     if ((2 * i) <= n) {
18         // 有左孩子
19         root->lchild = BiTreeCompleteCreate(n, 2 * i);
20     } else {
21         // 没有左孩子
22         root->lchild = NULL;
23     }
24     // 当2*i+1≤n时, 有右孩子, 其编号为2*i+1 ,否则没有右孩子;
25     if ((2 * i + 1) <= n) {
26         // 有右孩子
27         root->rchild = BiTreeCompleteCreate(n, 2 * i + 1);
28     } else {
29         // 没有右孩子
30         root->rchild = NULL;
31     }
32
33     return root;
34 }
35
36 // 根左右
37 void BiTreeFrontOrder(bitree_t* root)
38 {
39     if (root == NULL)
40         return;
41     printf(" %d", root->data);
42     BiTreeFrontOrder(root->lchild);
43     BiTreeFrontOrder(root->rchild);
44 }
45 void BiTreeMiddleOrder(bitree_t* root)
46 {
47     if (root == NULL)
48         return;
49     BiTreeMiddleOrder(root->lchild);
50     printf(" %d", root->data);
51     BiTreeMiddleOrder(root->rchild);
52 }
53 void BiTreeRearOrder(bitree_t* root)
54 {
55     if (root == NULL)
```

```

56     return;
57     BiTreeRearOrder(root->lchild);
58     BiTreeRearOrder(root->rchild);
59     printf(" %d", root->data);
60 }

```

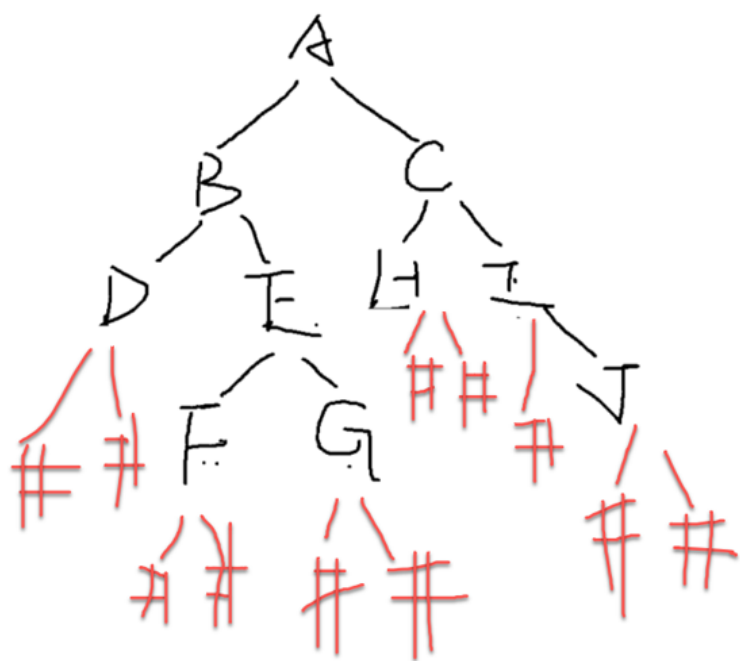
main.c

```

1  #include "bitree.h"
2
3  int main(int argc, const char* argv[])
4  {
5      bitree_t *root;
6      // 1. 创建完全二叉树
7      root = BiTreeCompleteCreate(12, 1);
8      if (root == NULL)
9          return -1;
10
11     printf("先序: ");
12     BiTreeFrontOrder(root);
13     putchar(10);
14
15     printf("中序: ");
16     BiTreeMiddleOrder(root);
17     putchar(10);
18
19     printf("后序: ");
20     BiTreeRearOrder(root);
21     putchar(10);
22
23     return 0;
24 }

```

2.9 普通二叉树创建



输入顺序

ABD##EF##G##CH##I#J##

ABD##EF##G##CH##I#J##

bitree.h

```

1  #ifndef __BITREE_H__
2  #define __BITREE_H__
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #define datatype int
8
9  typedef struct bitree {
10     datatype data;
11     struct bitree *lchild, *rchild;
12 } bitree_t;
13
14 bitree_t* BiTreeNormalCreate(void);
15 void BiTreeFrontOrder(bitree_t* root);
16 void BiTreeMiddleOrder(bitree_t* root);

```

先序序列:

ABDEF GCHIJ

中序序列:

DBFEG AHCIJ

后序序列:

DFGEB HJICA

```
17 void BiTreeRearOrder(bitree_t* root);
18 #endif
```

bitree.c

```
1  #include "bitree.h"
2
3  bitree_t* BiTreeNormalCreate(void)
4  {
5      bitree_t* root;
6      char ch;
7      ch = getchar();
8      if (ch == '#') {
9          return NULL;
10     }
11
12     root = (bitree_t*)malloc(sizeof(*root));
13     if (root == NULL) {
14         printf("%s malloc memory error\n", __func__);
15         return NULL;
16     }
17     root->data = ch;
18     root->lchild = BiTreeNormalCreate();
19     root->rchild = BiTreeNormalCreate();
20
21     return root;
22 }
23
24 // 根左右
25 void BiTreeFrontOrder(bitree_t* root)
26 {
27     if (root == NULL)
28         return;
29     printf(" %c", root->data);
30     BiTreeFrontOrder(root->lchild);
31     BiTreeFrontOrder(root->rchild);
32 }
33 void BiTreeMiddleOrder(bitree_t* root)
34 {
35     if (root == NULL)
36         return;
37     BiTreeMiddleOrder(root->lchild);
38     printf(" %c", root->data);
39     BiTreeMiddleOrder(root->rchild);
40 }
41 void BiTreeRearOrder(bitree_t* root)
42 {
43     if (root == NULL)
44         return;
45     BiTreeRearOrder(root->lchild);
46     BiTreeRearOrder(root->rchild);
47     printf(" %c", root->data);
48 }
```

main.c

```
1  #include "bitree.h"
2
3  int main(int argc, const char* argv[])
4  {
5      bitree_t *root;
6      // 1.创建普通二叉树
7      root = BiTreeNormalCreate();
8      if (root == NULL)
9          return -1;
10
11     printf("先序: ");
12     BiTreeFrontOrder(root);
13     putchar(10);
14
15     printf("中序: ");
16     BiTreeMiddleOrder(root);
17     putchar(10);
18
19     printf("后序: ");
20     BiTreeRearOrder(root);
21     putchar(10);
22
23     return 0;
24 }
```

3.递归

在c语言中函数可以调用本身，那这样的函数就叫做递归函数。

递归函数在写的时候比较简单，但是递归函数在空间复杂度和

时间复杂度上都会比较高。有可能会出现栈溢出的现象。在设计

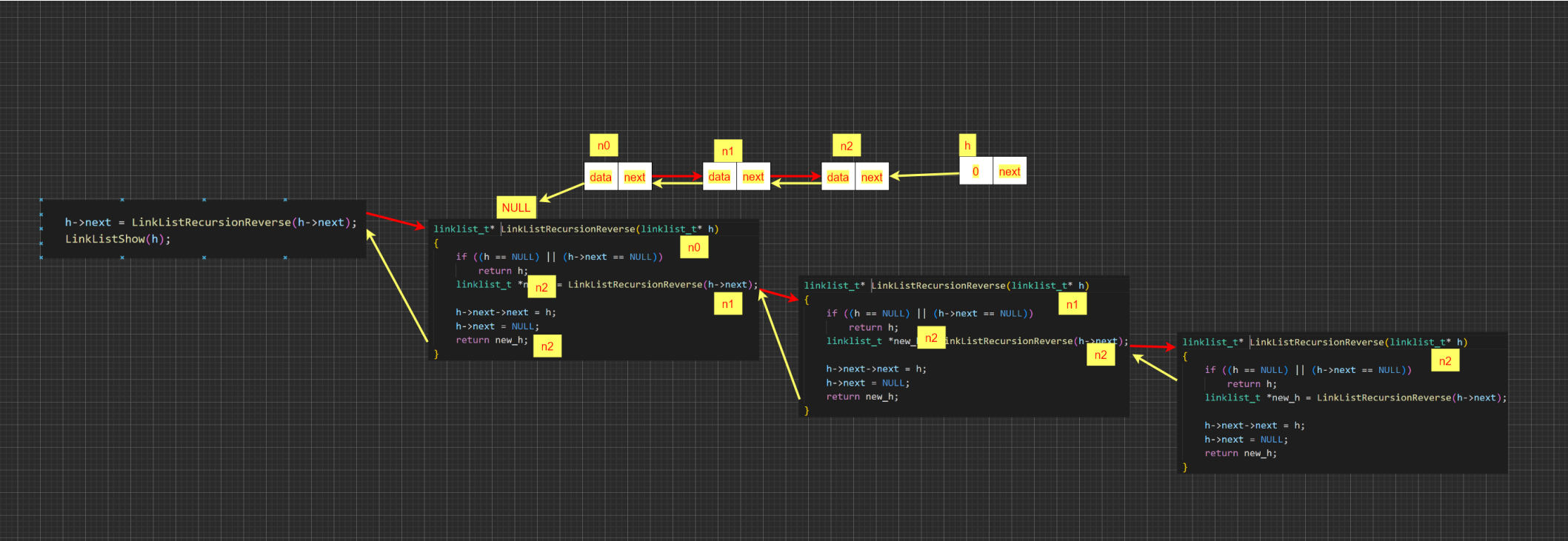
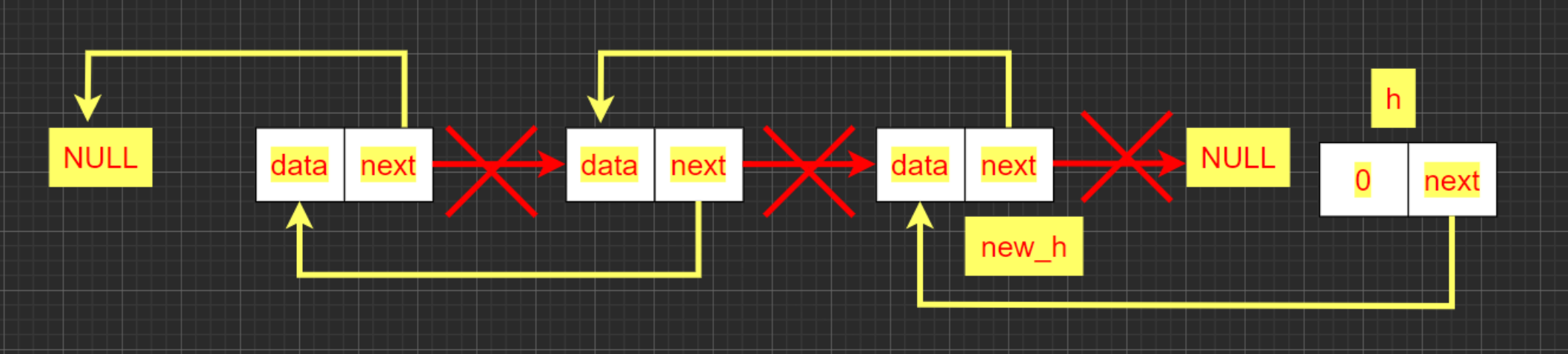
递归调用的时候一定要找到结束条件，否则就是死循环。

3.1使用递归实现n!阶乘

5! =5 * 4 * 3 * 2 * 1;

```
1  #include <stdio.h>
2  // jicheng(5) :5*jicheng(4) : 5*4*3*2*1
3  // jicheng(4) :4*jicheng(3) : 4*3*2*1
4  // jicheng(3) :3*jicheng(2) : 3*2*1
5  // jicheng(2) :2*jicheng(1) : 2*1
6  // jicheng(1) :1
7  int jicheng(int n)
8  {
9      if (n == 1)
10         return 1;
11     return n*jicheng(n-1);
12 }
13 int main(int argc, const char* argv[])
14 {
15     int n = 5;
16     printf("%d\n",jicheng(n));
17
18     return 0;
19 }
```

3.2使用递归实现单链表逆序



```
1  linklist_t* LinkListRecursionReverse(linklist_t* h)
2  {
3      if ((h == NULL) || (h->next == NULL))
4          return h;
5      linklist_t *new_h = LinkListRecursionReverse(h->next);
6
7      h->next->next = h;
8      h->next = NULL;
9      return new_h;
10 }
11 //在main.c中的调用方式
12 //h->next = LinkListRecursionReverse(h->next);
```

