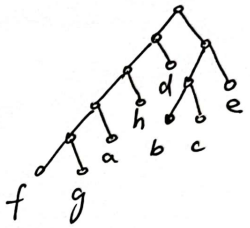
一. ABDAB
DBCBA

二. ×××××
××√×√
×××√

三. 1. 2^{i-1}
$2^k - 1$
2. ~~2h+1~~ 2h-1
~~2^{h+1}~~ $2^h - 1$
3. 5
2和4
4. CEDBGFA
5. 顺序存储
~~链式存储~~
链式存储
6. $\log_2(n+1)$
7. 10
8. O(n)
9. 500

四. 1. a. 0001
b. 100
c. 101
d. 01
e 11
f. 00000
g 0000 1
h. 001



则平均长度为
5×0.03+5×0.04+4×0.05+3×0.1+3×0.14
+3×0.16+2×0.2+2×0.28 = 2.71
~~则存储~~ 则100个字符需 2.71×100 = 271个
二进制位

2.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXSIZE 100
```

```c
typedef struct node
{
    char ele;
    struct node *next;
}LinkNode, *LinkList;
```

```
LinkList InitList()
{   // 初始化链表
    LinkList p = (LinkList)malloc(sizeof(LinkNode));
    p->next = p;
    return p;
}
```

```
void AddEles(LinkList L, char *str)
{   // 向每个结点加入字符串中的字符
    LinkList p = L;
    for(int i = 0; str[i] != '\0'; i ++)
    {   // 字符串的每个值依次进入每一个结点

        // 为新结点赋值
        LinkList q = (LinkList)malloc(sizeof(LinkNode));
        q->ele = str[i];

        // 插入到之前结点的后继
        p->next = q;
        p = q;
    }

    p->next = L;
}
```

```
int IsPalindrome(LinkList p, char *str, int size)
{   // 判断是否中心对称
```

```
    char temp[MAXSIZE];
    for(int j = 0; j < size; j++)
    {
        temp[j] = str[size - j - 1];
    }
    for(int j = 0; j < size; j++)
    {
        if(temp[j] != str[j])
        {
            return 0;
        }
    }
    return 1;
}
```

```
int main()
```

```
{
    LinkList L = InitList();
    char str[MAXSIZE];
    printf("输入字符串:\n");
    scanf("%s", str);

    int size = strlen(str);
    AddEles(L, str);

    if(IsPalindrome(L, str, size))
        printf("中心对称\n");
    else
        printf("不中心对称\n");

    free(L);
    return 0;
}
```

3.1

```
#include <stdio.h>
#include <stdlib.h>

#define MAXSIZE 100
#define ElemType int
```

```
typedef struct BTNode
{
    ElemType data;
    struct BTNode *lchild, *rchild, *next;
}BTNode, *BiTree;
```

```
// 结点个数
int length = 0;
// 后序树的指针数组
BiTree BackPtr[MAXSIZE];
```

```
// 递归找到后序
void BackWard(BiTree T)
{
    if(T == NULL)
        return;
    else if(T->lchild != NULL)
```

```
        BackWard(T->lchild);
    else if(T->rchild != NULL)
        BackWard(T->rchild);
    else
    {
        BackPtr[length] = T;
        length ++;
    }
}
```

```
// 给每一个结点按照后序赋 next
void GetNext(BiTree T)
{
    BackWard(T);
    for(int i = 0; i < length - 1; i ++)
    {
        BackPtr[i]->next = BackPtr[i + 1];
    }
    BackPtr[length - 1]->next = NULL;
}
```

3.2

```
#include <stdio.h>
#include <stdlib.h>

#define ElemType int
```

```
typedef struct BTNode
{
    ElemType data;
    struct BTNode *lchild, *rchild, *next;
}BTNode, *BiTree;
```

```
int flag = 0;
int HaveRelation(BiTree ptrx, BiTree ptry)
{   // 有关系
    if(ptrx == NULL || ptry == NULL)
        return 0;
    if(ptrx == ptry || flag)
        return 1;
    /*左孩子右孩子向下寻找*/
    if(ptry->lchild)
```

```
    {
        flag ++;
        return HaveRelation(ptrx, ptry->lchild);
    }

    if(ptry->rchild)
    {
        flag ++;
        return HaveRelation(ptrx, ptry->rchild);
    }

    if(ptrx->lchild)
    {
        flag ++;
        return HaveRelation(ptrx->lchild, ptry);
    }

    if(ptrx->rchild)
    {
        flag ++;
        return HaveRelation(ptrx->rchild, ptry);
    }
}
```

3.3

```
#include <stdio.h>
#include <stdlib.h>

#define ElemType int
```

```
typedef struct BTNode
{
    ElemType data;
    struct BTNode *lchild, *rchild, *next;
}BTNode, *BiTree;
```

```
// 从根节点开始找
int Get_Level(BiTree root, BiTree ptr, int level)
{
    if(root == NULL)
        return 0;
    if(root == ptr)     // 找到
```

```
        return level;
```

```
    /*向左孩子右孩子寻找*/
    if(Get_Level(root->lchild, ptr, level + 1))
        return (Get_Level(root->lchild, ptr, level + 1));
    else
        return (Get_Level(root->rchild, ptr, level + 1));
}
```