几个算法及问题

数学科学学院：汪小平

wxiaoping325@163.com

蓝墨云班课号：276547

一、寻找最小元素

若一个数组是以循环顺序排列的,即数组中∃i,有关系:

$$x_0 < x_1 < x_2 < \cdots < x_{i-1}, x_i < x_{i+1} < \cdots < x_n < x_0$$

例如, 8,10,14,15, 2, 6这6个元素就是循环顺序排列的.

编写一个程序,从外部读入一个以循环顺序排列的数组,然后把它的最小元素找出来(用自定义函数实现).

```c
#include <stdio.h>
#define N 100000
int findmin(int* a, int n);//n为元素个数
int main() {
    int i, n, k, a[N];
    scanf("%d", &n);
    for(i = 0; i <= n; i++)
        scanf("%d", &a[i]);
    k = findmin(a, n+1);
    printf("%d\n", k);
    return 0;
}
int findmin(int* a, int n) {
    int i, left = 0, right = n; //区间[left, right)
    for(i = left; i < right - 1; i++) {
        if(a[i] > a[i+1])
            break;
    }
    return a[i+1];
}
```

有没有更快的方法？

$x_0 < x_1 < x_2 < \cdots < x_{i-1}, x_i < x_{i+1} < \cdots < x_n < x_0$

8,10,14,15,2,6

思想: 二分

当处理区间$[L,R)$的元素时,取$M=\dfrac{L+R}{2}$,则会出现两种情况:

(1)$x_L \le x_M$. 由数列性质,则最小元素一定在区间$[M,R)$中;

(2)$x_L > x_M$. 由数列性质,则最小元素一定在区间$[L,M+1)$中.

```
int findmin(int* a, int n) {
    int mid, left = 0, right = n; //区间[left, right)
    while(left < right) {
        mid = (left + right) / 2;
        if(a[left] < a[mid])     left = mid;
        else {
            if(mid - left == 1)  return a[mid];
            else                 right = mid + 1;
        }
    }
}
```

问题:

1. 在哪儿用过二分的思想?

利用零点定理求函数零点

```
double f(double x); //求函数f在x处的函数值
double root(double a, double b, double eps)
{
        double mid = (a + b) / 2.0;
        while((b - a) > eps)
        {
                if(f(mid) == 0)
                        break;
                else if(f(a) * f(mid) < 0)
                        b = mid;
                else
                        a = mid;
                mid = (a + b) / 2;
        }
        return mid;
}
```

问题:

2. 怎样利用二分实现数组元素的查找?

数组要有序

```
int binary_search(int* array, int n, int value) //返回下标
{
        int left = 0, mid, right = n; //区间[left, right)
        while(left < right)
        {
                mid = (left + right) / 2;
                if(array[mid] == value)
                        return mid;
                else if(array[mid] > value)
                        right = mid;
                else
                        left = mid + 1;
        }
        return -1;
}
```

考虑数组:8 21 56 56 56 97 102 978, value = 56

1. 怎样找到下标最小的value (lowerbound)?

2. 怎样找到下标最大的value (upperbound)?

考虑下面代码及返回值的含义:

```
int lower_bound(int* array, int n, int value)
{
      int left = 0, mid, right = n; //区间[left, right)
      while(left < right)
      {
            mid = (left + right) / 2;
            else if(array[mid] >= value) right = mid;
            else   left = mid + 1;
      }
      return mid;
}
```

二、3个数组的共同元素

有3个数组x[ ]、y[ ]和z[ ]，各有nx、ny和nz个元素，而且三者都已经从小到大依序排列．请写一个程序，找出值最小的共同元素，若没有共同元素，请显示合适的信息．

考虑从头开始比较x[i]、y[j]和z[k]时的不同情况分类:

```
1. x[i] < y[j]
   则只须 i++ 即可;
2. x[i] >= y[j]
   (1) y[j] < z[k]
       则只须 j++;
   (2) y[j] >= z[k]
       (a) x[i] > z[k]
           则只须 k++;
       (b) z[k] >= x[i]
           则 x[i] == y[j] == z[k].
```

7 5 8 32 63 85 98 100

5 7 45 62 85 98

14 10 14 16 20 24 26 28 30 32 34 38 85 90 98

```
#include <stdio.h>
#define N 10000
int findsame(int* x, int* y, int* z, int nx, int ny,
                                     int nz, int* value);
int main()
{
    int i, nx, ny, nz, x[N], y[N], z[N], value;
    scanf("%d", &nx);
    for(i = 0; i < nx; i++)
        scanf("%d", &x[i]);
    scanf("%d", &ny);
    for(i = 0; i < ny; i++)
        scanf("%d", &y[i]);
    scanf("%d", &nz);
    for(i = 0; i < nz; i++)
        scanf("%d", &z[i]);
    if(findsame(x, y, z, nx, ny, nz, &value))
        printf("The same value is %d.\n", value);
    else
        printf("No same value!\n");
    return 0;
}
```

```
int findsame(int* x, int* y, int* z, int nx, int ny,
                                     int nz, int* value)
{
    int i = 0, j = 0, k = 0;
    while((i < nx) && (j < ny) && (k < nz)) {
        if(x[i] < y[j])       i++;
        else if(y[j] < z[k])  j++;
        else if(x[i] > z[k])  k++;
        else {
            *value = x[i];
            return 1;
        }
    }
    return 0;
}
/*
7 5 8 32 63 85 98 100
5 7 45 62 85 98
14 10 14 16 20 24 26 28 30 32 34 38 85 90 98
*/
```

```
1.x[i] < y[j]
   则只须 i++ 即可;
2.x[i] >= y[j]
  (1)y[j] < z[k]
     则只须 j++;
  (2)y[j] >= z[k]
     (a)x[i] > z[k]
        则只须 k++;
     (b)z[k] >= x[i]
        则 x[i] == y[j] == z[k].
```

三、查找矩阵

已知n行n列的矩阵M,其任一元素$M_{i,j}$都小于它右边和下方的元素(若存在). 即：$M_{i,j} < M_{i,j+1}$和$M_{i,j} < M_{i+1,j}$.现在给出值K,请写一个程序,检查矩阵M中是否存在K.

$$M = \begin{bmatrix} 1 & 3 & 7 & 9 & 13 \\ 2 & 4 & 8 & 10 & 14 \\ 5 & 6 & 9 & 12 & 15 \\ 7 & 12 & 13 & 15 & 16 \\ 11 & 13 & 16 & 18 & 25 \end{bmatrix}, \quad K = 11$$

1. 查找算法

```
#define N 100
int value, matrix[N][N];
int find() {
    int flag = 0;
    int i = 0; j = n - 1;
    while(flag == 0) {
        if(matrix[i][j] == value) {
            flag = 1;
            break;
        }
        else if(matrix[i][j] > a) {
            if(j == 0) break;
            else       j--;
        }
        else {
            if(i == n - 1) break;
            else           i++;
        }
    }
    return flag;
}
```

2. 设计一个算法, 生成满足上述要求的矩阵.

深度优先搜索

```
#include <stdio.h>
#include <stdlib.h>
#define N 100
int n, matrix[N][N];
void create_matrix(int i, int j);
int main() {
    freopen("data.in", "w", stdout);
    int T = 100;
    printf("%d\n", T);
    while(T--) {
        int i, j;
        memset(matrix, 0, sizeof(matrix));
        switch(T / 10) {
            case 9: n = rand() % 4 + 8; break;
            case 8: n = rand() % 4 + 16; break;
            case 7: n = rand() % 4 + 26; break;
            case 6: n = rand() % 4 + 36; break;
            case 5: n = rand() % 4 + 46; break;
            case 4: n = rand() % 4 + 56; break;
            case 3: n = rand() % 4 + 66; break;
            case 2: n = rand() % 4 + 76; break;
            case 1: n = rand() % 4 + 86; break;
            default: n = rand() % 4 + 96; break;
        }
        printf("%d\n", n);
        matrix[0][0] = rand() % 17 + 2;
        create_matrix(0, 0);
        for(i = 0; i < n; i++) {
            for(j = 0; j < n; j++)
                printf("%d ", matrix[i][j]);
            printf("\n");
        }
    }
    return 0;
}
void create_matrix(int i, int j) {
    if((i < n - 1) && ((matrix[i+1][j] == 0) ||(matrix[i+1][j] <= matrix[i][j]))) {
        matrix[i+1][j] = matrix[i][j] + (rand() % 8 + 1);
        create_matrix(i + 1, j);
    }
    if((j < n - 1) && ((matrix[i][j+1] == 0) ||(matrix[i][j+1] <= matrix[i][j]))) {
        matrix[i][j+1] = matrix[i][j] + (rand() % 8 + 1);
        create_matrix(i, j + 1);
    }
}
```

3.给出一个正整数R,请写一个程序,找出所有满足$X^2+Y^2=R$的正整数对X和Y.

解法:

- 构造矩阵M,其元素$M_{i,j}=i^2+j^2$,则该矩阵满足题目的条件.
- 显然,$i,j<\sqrt{R}$,所以矩阵M为$\sqrt{R}\times\sqrt{R}$就行.
- M其实是多余的,只须在用$M_{i,j}$时计算i^2+j^2即可.
- 由于M矩阵是对称的,只须考虑上三角或下三角即可.

样例输入

```
7
2
38
68
96
338
545
194545
```

样例输出

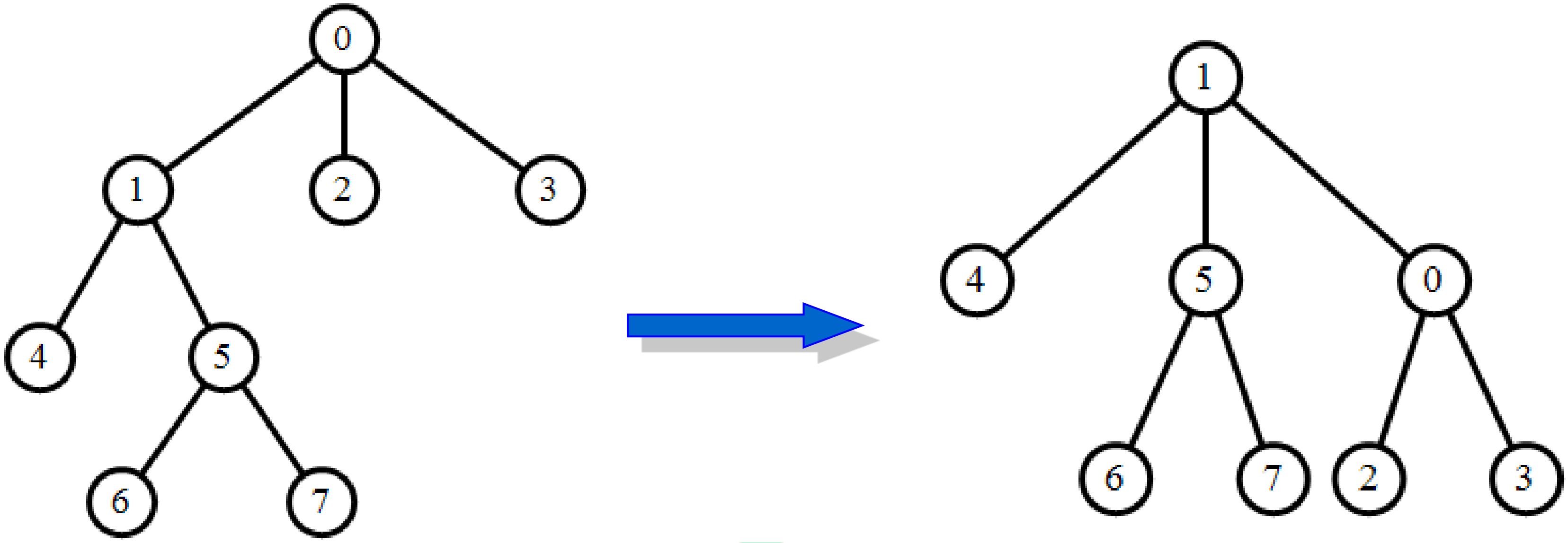
```
2 = 1^2 + 1^2
38 = No found!
68 = 2^2 + 8^2
96 = No found!
338 = 7^2 + 17^2
338 = 13^2 + 13^2
545 = 4^2 + 23^2
545 = 16^2 + 17^2
194545 = 8^2 + 441^2
194545 = 84^2 + 433^2
194545 = 89^2 + 432^2
194545 = 177^2 + 404^2
194545 = 188^2 + 399^2
194545 = 217^2 + 384^2
194545 = 271^2 + 348^2
194545 = 296^2 + 327^2
```

```c
#include <stdio.h>
#include <math.h>
int square_sum(int i, int j);
int find(int r, int* x, int* y, int first);
int main() {
    int T;
    scanf("%d", &T);
    while(T--) {
        int r, x, y, cnt = 0;
        scanf("%d", &r);
        while(find(r, &x, &y, cnt == 0) == 1) {
            printf("%d = %d^2 + %d^2\n", r, x, y);
            cnt++;
        }
        if(cnt == 0) printf("%d = No found!\n", r);
    }
    return 0;
}
int square_sum(int i, int j) {
    return i * i + j * j;
}
```

```
int find(int r, int* x, int* y, int first) {
    int n = (int)sqrt(r), flag = 0;
    static int i, j;
    if(first) {
        i = 0; j = n;
    }
    i++;
    while(flag == 0) {
        if(square_sum(i, j) == r) {
            *x = i; *y = j; flag = 1; break;
        }
        else if(square_sum(i, j) > r) {
            if(j <= i) break;
            else       j--;
        }
        else {
            if(i >= n) break;
            else       i++;
        }
    }
    return flag;
}
```

四、无根树转有根树

输入一个n个结点的无根树的各条边，并指定一个根结点，要求把该树转化为有根树，输出各个结点的父结点编号。$n<=10^6$。



输入样例	输出样例

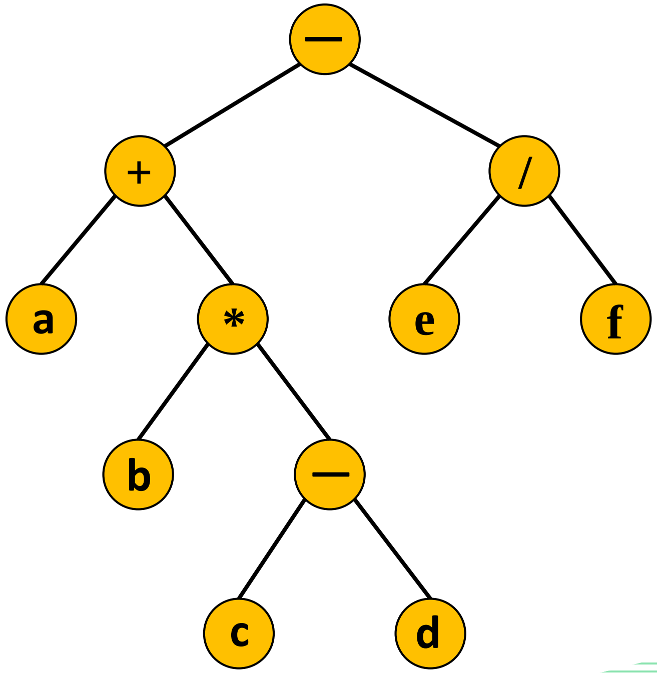
8 1
0 1
0 2
0 3
1 4
1 5
5 6
5 7
	1 -1 0 0 1 1 5 5

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 1000002
struct edge
{
    int node;
    struct edge* next;
} mynode[N], myedge[2 * N];
int n, root, father[N];
void dfs(int u);
int main()
{
    while(scanf("%d%d", &n, &root) == 2)
    {
        int i, k = 0, s, t;
        memset(father, -1, sizeof(father));
        memset(mynode, 0, sizeof(mynode));
        for(i = 0; i < n - 1; i++) //建立邻接表
        {
            scanf("%d%d", &s, &t);
            myedge[k].node = t;
            myedge[k].next = mynode[s].next;
            mynode[s].next = &myedge[k];
            k++;
```

```
                myedge[k].node = s;
                myedge[k].next = mynode[t].next;
                mynode[t].next = &myedge[k];
                k++;
            }
            dfs(root);
            for(i = 0; i < n; i++)
                printf("%d ", father[i]);
            printf("\n");
        }
        return 0;
}
void dfs(int u)
{
    struct edge *p = mynode[u].next;
    while(p != NULL)
    {
        int v = p->node;
        if(v != father[u])
        {
            father[v] = u;
            dfs(v);
        }
        p = p->next;
    }
}
```

五、表达式树

假设一个表达式只含单字符、四则运算符和圆括号(除开圆括号, 表达式字符数不超过1000), 试建立一个表达式树, 使得中序遍历的结果恰好是该表达式. 下面是a + b * (c - d) - e / f对应的表达式树.



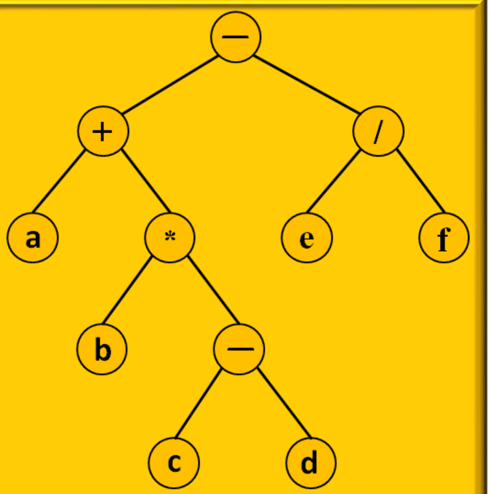
分析: 找到最后运算的运算符, 把表达式分成两半, 通过递归的方式建立整棵树.

表达式二叉树建立操作

```
build_tree(s, 0, strlen(s));
```

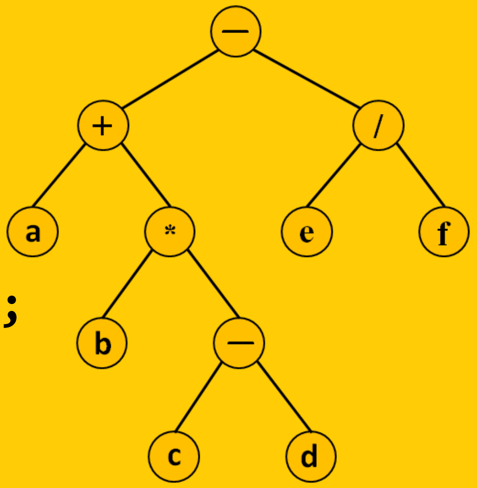
“a+b*(c-d)-e/f ”

```
#define N 1000
int lch[N], rch[N];//左右儿子结点编号
char op[N];//表达式字符
int nc;//结点数计数器
int build_tree(char* s, int x, int y)
{
    int i, c1 = -1, c2 = -1, p = 0;
    int u;
    if(y - x == 1)
    {
        u = nc++;
        lch[u] = rch[u] = -1;//表示是叶子
        op[u] = s[x];
        return u;
    }
```



```
    for(i = x; i < y; i++)
    {
        switch(s[i])
        {
            case '(': p++; break;
            case ')': p--; break;
            case '+':
            case '-':  if(p == 0) c1 = i; break;
            case '*':
            case '/': if(p == 0) c2 = i; break;
        }
    }
    if(c1 == -1) //括号外没有加减号
        c1 = c2;
    if(c1 == -1) //整个表达式外有一对括号
        return build_tree(s, x + 1, y - 1);
    u = nc++;
    op[u] = s[c1];
    lch[u] = build_tree(s, x, c1);
    rch[u] = build_tree(s, c1 + 1, y);
    return u;
}
```

a + b * (c - d) - e / f

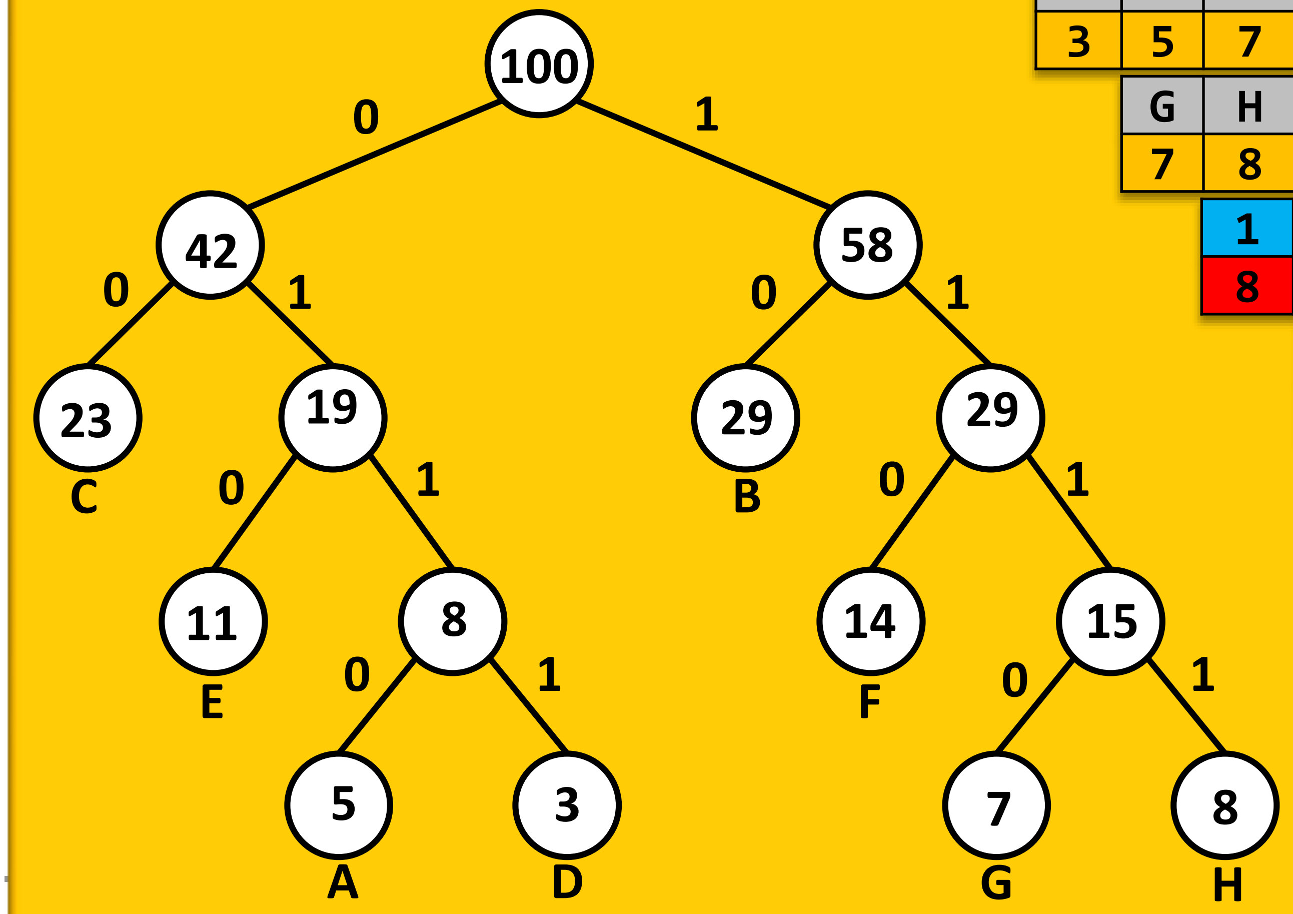


(a + b * (c - d) - e / f)

```c
void middfs(int u)
{
    if(u == -1)
        return;
    printf("(");
    middfs(lch[u]);
    printf("%c", op[u]);
    middfs(rch[u]);
    printf(")");
}
int main()
{
    char s[10 * N];
    while(scanf("%s", s) == 1)
    {
        nc = 0;
        build_tree(s, 0, strlen(s));
        middfs(0);
    }
    return 0;
}
```

六、利用最优二叉树进行文本文件的编码与解码操作

字符	A	B	C	D	E	F	G	H
频数	5	29	23	3	11	14	7	8
编码	0110	10	00	0111	010	110	1110	1111



D	A	G	H	E	F	C	B
3	5	7	8	11	14	23	29

G	H	1	E	F	C	B
7	8	8	11	14	23	29

1	E	F	2	C	B
8	11	14	15	23	29

F	2	3	C	B
14	15	19	23	29

3	C	B	4
19	23	29	29

B	4	5
29	29	42

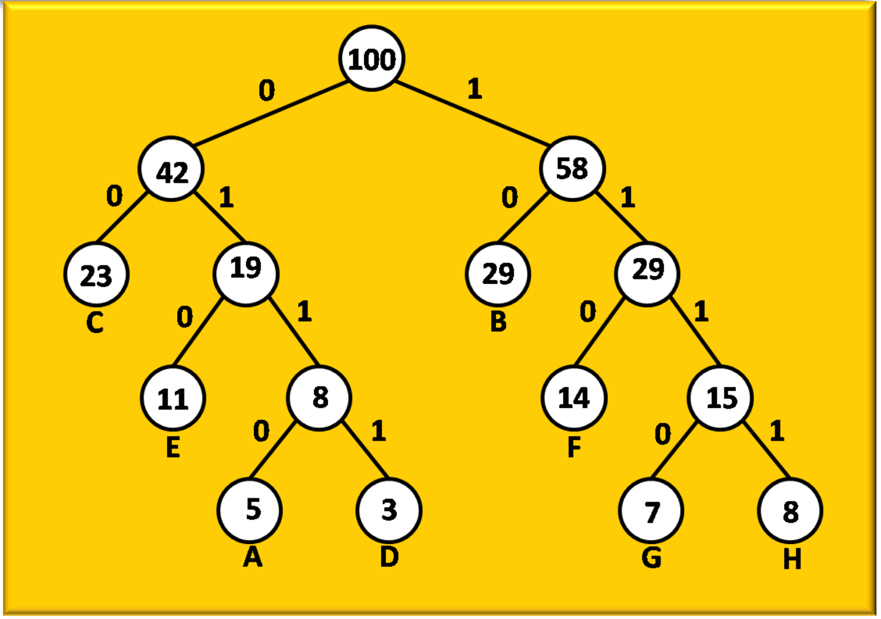
5	6
42	58

7
100

字符	A	B	C	D	E	F	G	H
频数	5	29	23	3	11	14	7	8
编码	0110	10	00	0111	010	110	1110	1111

如果文本文件中的字符序列为: ABCDFF……

则该文本文件的编码文件的二进制序列为:

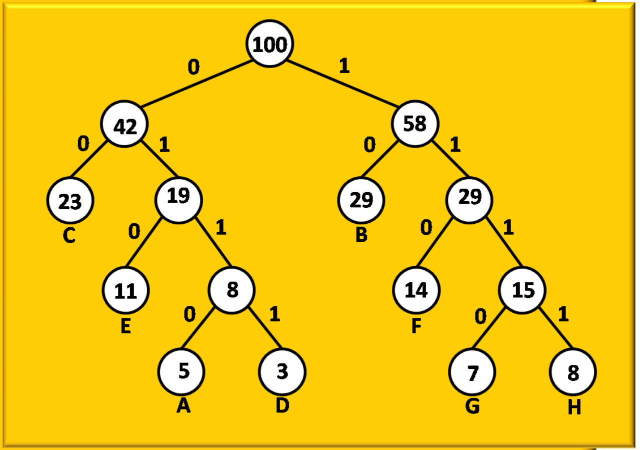


0110100001111110110……

反过来, 由此二进制序列, 结合上面的哈夫曼树, 可以解码得到对应的字符序列. 本例只实现编码过程, 解码作为课后作业由大家完成.

(1)定义类型和声明函数模块

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXLEN 256
typedef struct HuffCode{  //存储每个字符的哈夫曼编码
    int code[MAXLEN];
    int length;
} HuffCode;
typedef struct HTNode{ //哈夫曼树的结点
    int weight;
    int parent, lchild, rchild;
} HTNode;
int CountFreq(FILE* fp, int* freq);
//根据字节出现频数找出每个字节的Huffman编码
void HuffmanCoding(int* freq, HuffCode* hCode, int n);
void FileCoding(FILE* SFile, FILE* DFile,
             HuffCode* hCode, int* freq, int bytesCount);
```



(2)主函数的实现

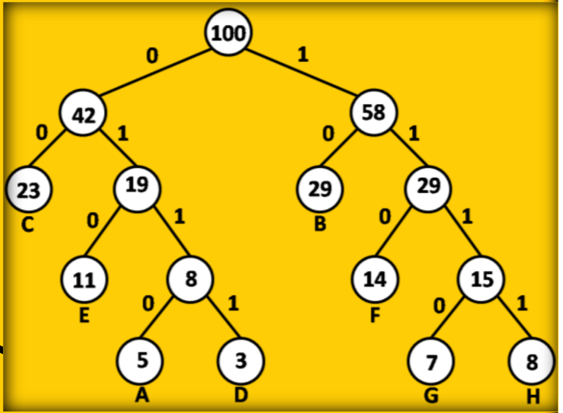
```
int main(){
    char sFile[200], dFile[200];
    scanf("%s%s", sFile, dFile);
    FILE* sfp = (FILE*)fopen(sFile, "rb");
    if(sfp == NULL){
        printf("源文件%s打不开!\n", sFile);
        exit(0);
    }
    FILE* dfp = (FILE*)fopen(dFile, "wb");
    if(dfp == NULL){
        printf("目标文件%s打不开!\n", dFile);
        fclose(sfp);
        exit(0);
    }
    int i, typeCount, freq[MAXLEN], tFreq[MAXLEN];
    int types[MAXLEN];
    int bytesCount = CountFreq(sfp, freq);
    for(i = 0, typeCount = 0; i < MAXLEN; i++){
        if(freq[i] != 0){//去掉没有出现的字节频数
            tFreq[typeCount] = freq[i];
            types[typeCount] = i;
            typeCount++;
        }
    }
    if(typeCount == 0){
        printf("文件为空文件!\n");
        exit(0);
    }
    HuffCode hCode[MAXLEN];
    HuffmanCoding(tFreq, hCode, typeCount);//得到字节编码
    for(i = typeCount - 1; i >= 0; i--){//使得下标即是字节
        hCode[types[i]] = hCode[i];
        memset(&hCode[i], -1, sizeof(HuffCode));
    }
    FileCoding(sfp, dfp, hCode, freq, bytesCount);
    fclose(sfp);  fclose(dfp);
    printf("文件编码完成!\n");
    return 0;
}
```

(3)统计文本文件中每个字节出现的频数

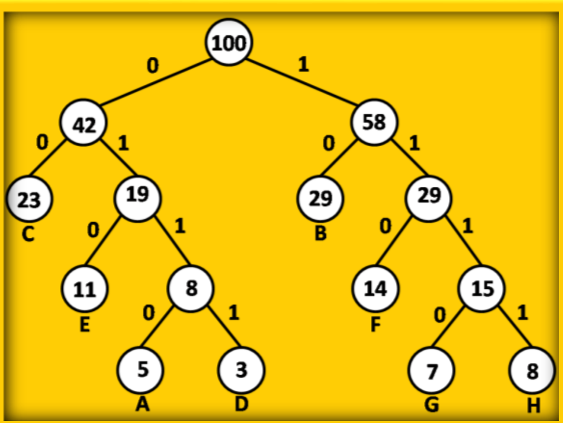
```
int CountFreq(FILE* fp, int* freq)
{
    int ch, cnt = 0;
    memset(freq, 0, MAXLEN * sizeof(int));//清零
    while((ch = fgetc(fp)) != EOF)
    {
        freq[ch]++;//统计每个字节的频数
        cnt++;
    }
    return cnt;
}
```

(4)利用频数实现哈夫曼树，得到字符的哈夫曼编码

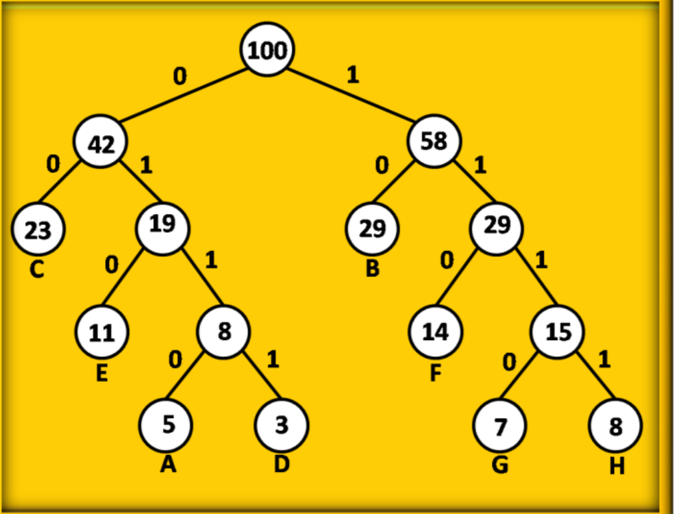
```
void HuffmanCoding(int* freq, HuffCode* hCode, int n){
    if(n == 1){//只出现了一种字符的情形
        hCode[0].code[0] = 0;
        hCode[0].length = 1;
        return;
    }
    HTNode* ht = (HTNode*)malloc((2*n-1) * sizeof(HTNode));
    memset(ht, -1, (2*n-1) * sizeof(HTNode));
    int i, j, s, t;
    for(i = 0; i < n; i++)  ht[i].weight = freq[i];
    for(i = n; i < 2*n-1; i++){//每次生成一个分支结点
        for(j = 0; j < i; j++){//找到第一个没有父亲的结点
            if(ht[j].parent == -1){
                s = j;
                break;//一定会从此退出
            }
        }
        for(j++; j < i; j++){//找到第二个
            if(ht[j].parent == -1){
                if(ht[s].weight > ht[j].weight){
```



```
                t = s;
                s = j;
            }
            else
                t = j;
            break;//一定会从此退出
        }
    }
    for(j++; j < i; j++) {//和剩余的比较,找两个最小的
        if(ht[j].parent == -1) {
            if(ht[j].weight < ht[s].weight) {
                t = s;
                s = j;
            }
            else if(ht[j].weight < ht[t].weight)
                t = j;
        }
    }
    ht[s].parent = ht[t].parent = i;
    ht[i].lchild = s;
    ht[i].rchild = t;
```



```
         ht[i].weight = ht[s].weight + ht[t].weight;
    }//最后一个结点一定是根
    int node;
    int a[MAXLEN];
    for(i = 0; i < n; i++){//逆向查找每个叶子的编码
        for(j = 0, node = i; ht[node].parent >= 0; j++){
            if(ht[ht[node].parent].lchild == node)
                a[j] = 0; //左取0
            else
                a[j] = 1; //右取1
            node = ht[node].parent;
        }
        s = 0;
        hCode[i].length = j;
        while(j--)
            hCode[i].code[s++] = a[j];
    }
}
```



(5)利用字符的哈夫曼编码实现文本文件的重新编码

```
void FileCoding(FILE* SFile, FILE* DFile, HuffCode* hCode,
int* freq, int bytesCount){
    fwrite(&bytesCount, sizeof(bytesCount), 1, DFile);
    fwrite(freq, sizeof(int) * MAXLEN, 1, DFile);
    rewind(SFile);
    int i, a[8] = {1, 2, 4, 8, 16, 32, 64, 128};
    int byte = 0, ch, ibyte = 0;
    while((ch = fgetc(SFile)) != EOF){
        for(i = 0; i < hCode[ch].length; i++){
            if(hCode[ch].code[i] == 1)  byte |= a[ibyte];
            ibyte++;
            if(ibyte == 8) {
                fputc(byte, DFile);
                byte = 0; ibyte = 0;
            }
        }
    }
    if(ibyte > 0) fputc(byte, DFile);
}
```

A	B	C	D	E	F
5	29	23	3	11	14
0110	10	00	0111	010	110

ABCDFF……

0110100001111110110……