《算法竞赛入门到进阶》：勘误与改进

出版社：清华大学出版社
作者：罗勇军 郭卫斌
联系方式：QQ 15512356，QQ 群：567554289

最近更新：2020.3，第 5 次印刷。
在扉页中可以查到是第几次印刷。

本书在多次重印过程中，进行了持续勘误和改进。本文记录了这些细节，包括两方面的内容：

（1）勘误。本书的印刷或内容错误，每次新印刷时，会修改新发现的问题。
（2）新内容。增加的新内容。

本文下载地址：

（1）https://github.com/luoyongjun999/code
（2）QQ 群：567554289 群文件

1、第 2 次印刷（2019.8）

以下已经改正：

（1）55 页，把(c,r)改成(r,c)

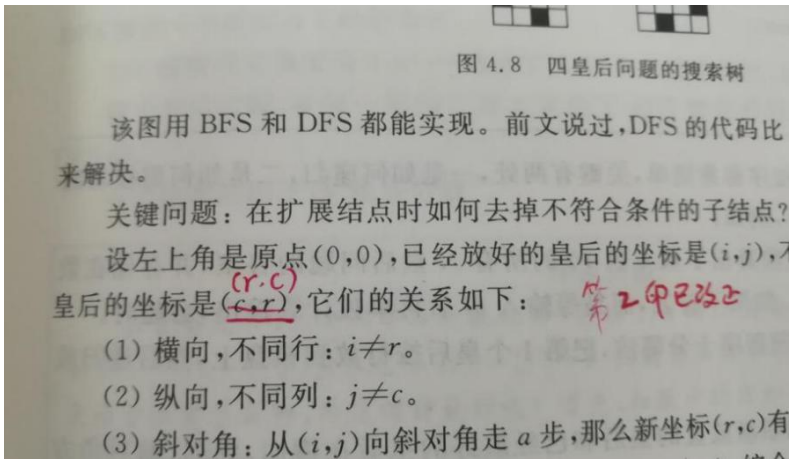


图 4.8 四皇后问题的搜索树

该图用 BFS 和 DFS 都能实现。前文说过，DFS 的代码比来解决。

关键问题：在扩展结点时如何去掉不符合条件的子结点？

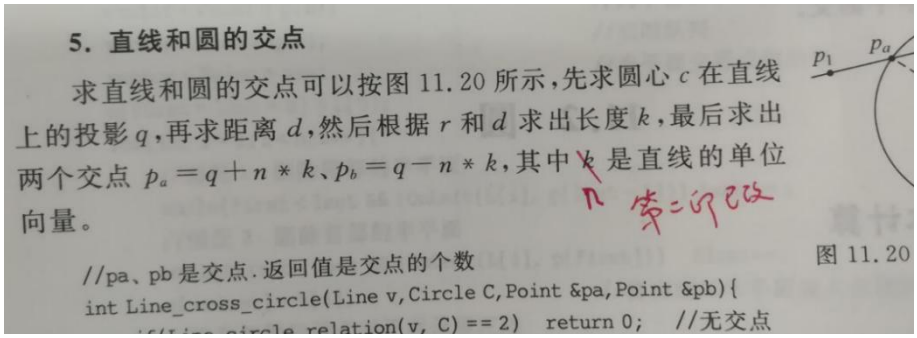
设左上角是原点(0,0)，已经放好的皇后的坐标是(i,j)，皇后的坐标是~~(c,r)~~(r,c)，它们的关系如下：第2印已改正

(1) 横向，不同行：$i \neq r$。

(2) 纵向，不同列：$j \neq c$。

(3) 斜对角：从(i,j)向斜对角走 a 步，那么新坐标(r,c)有

（2）294 页，把 k 改成 n。



5. 直线和圆的交点

求直线和圆的交点可以按图 11.20 所示，先求圆心 c 在直线上的投影 q，再求距离 d，然后根据 r 和 d 求出长度 k，最后求出两个交点 $p_a = q + n * k$、$p_b = q - n * k$，其中 ~~k~~ n 是直线的单位向量。第2印已改

图 11.20

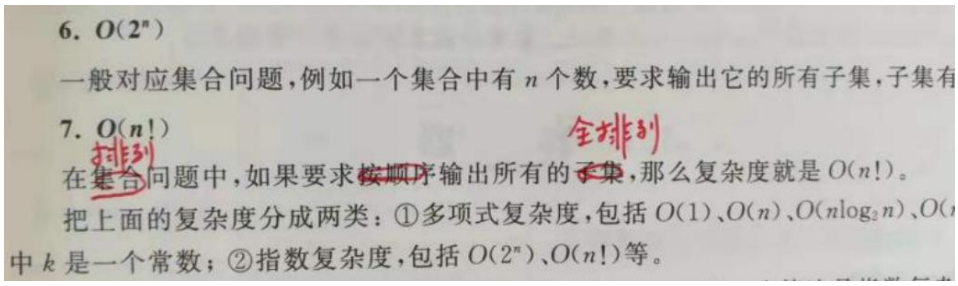
```
//pa、pb 是交点.返回值是交点的个数
int Line_cross_circle(Line v,Circle C,Point &pa,Point &pb){
    if(Line_circle_relation(v, C) == 2)  return 0;   //无交点
```

2、第 3 次印刷 （2019.9）

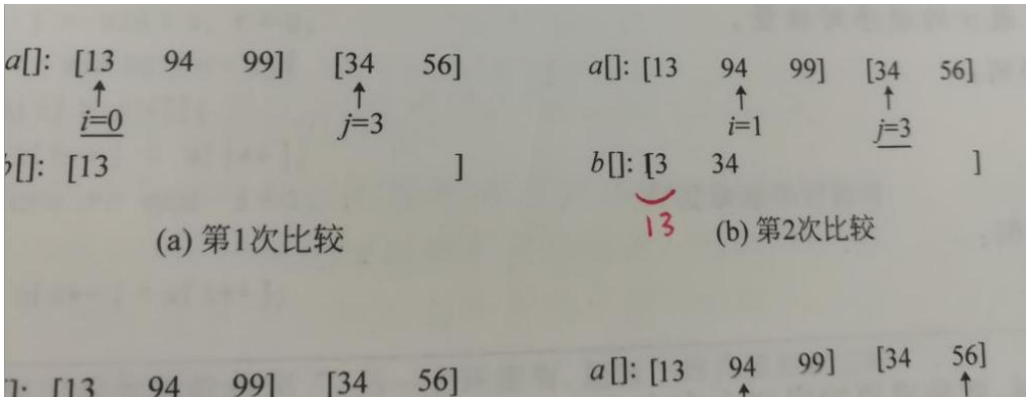
改进：前面 2 印是 120 分钟视频；从第 3 印开始，增加到 350 分钟视频。

下面是修改：

（1）23 页，修改一句。



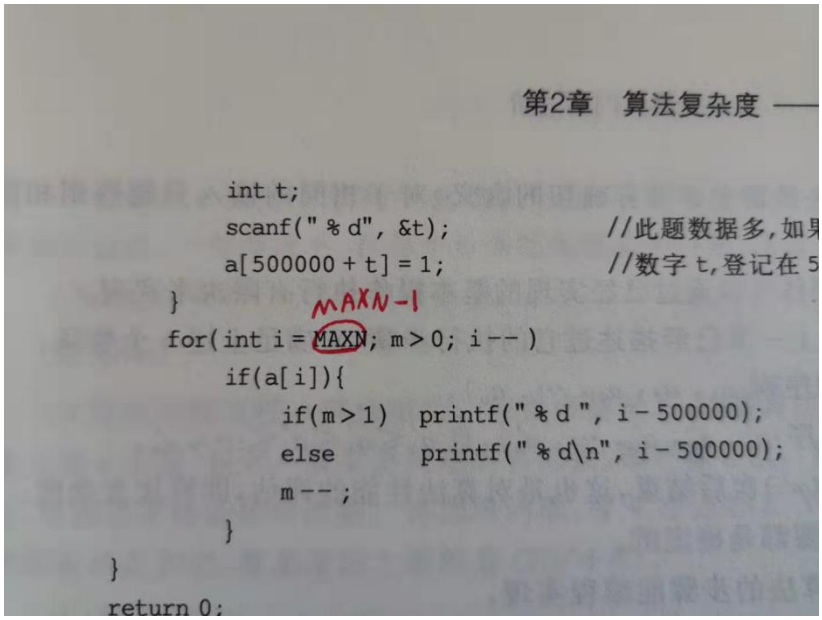
（3）109 页，应该是 13。



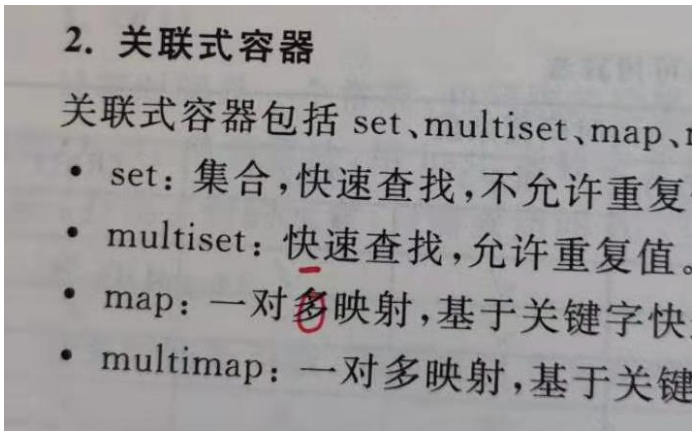
3. 第 4 次印刷（2019.11）

以下内容已经修改：

（1）21 页。把 i=MAXN 改为 i=MAXN-1



（2）24 页。把“map：一对多映射”改为“map：一对一映射”。

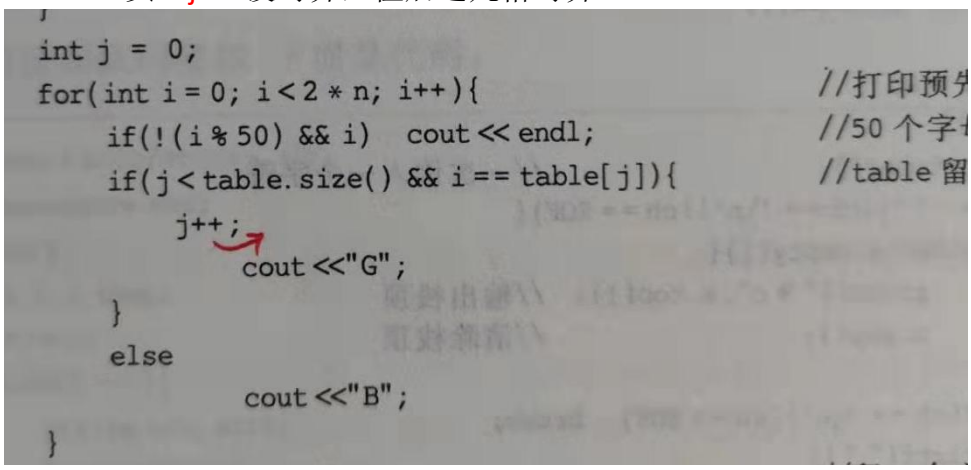


2. 关联式容器

关联式容器包括 set、multiset、map、

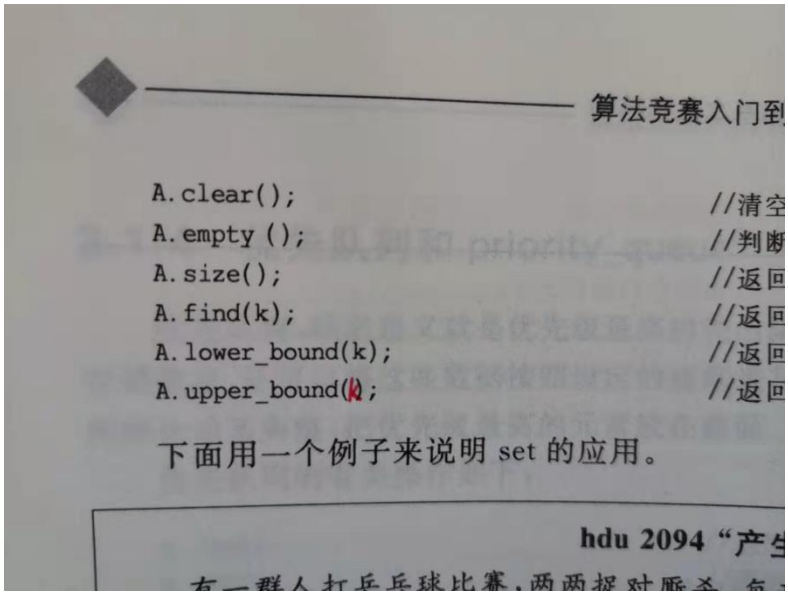
- set：集合，快速查找，不允许重复
- multiset：快速查找，允许重复值。
- map：一对多映射，基于关键字快
- multimap：一对多映射，基于关键

（3）27 页。j++ 没对齐，往后退几格对齐。



```
int j = 0;
for(int i = 0; i < 2 * n; i++){                    //打印预先
    if(!(i % 50) && i)  cout << endl;              //50 个字
    if(j < table.size() && i == table[j]){         //table 留
        j++;
            cout <<"G";
    }
    else
            cout <<"B";
}
```

（4）32 页。第 6 行括号里面少一个 k



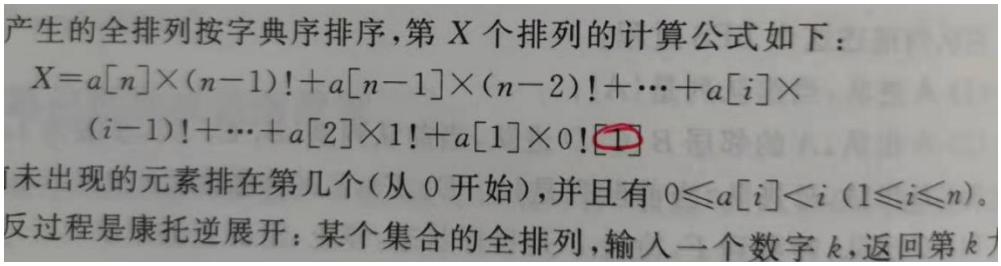
算法竞赛入门到

```
A.clear();                          //清空
A.empty();                          //判断
A.size();                           //返回
A.find(k);                          //返回
A.lower_bound(k);                   //返回
A.upper_bound(k);                   //返回
```

下面用一个例子来说明 set 的应用。

hdu 2094 "产生

（5）48 页。删除公式末尾的 [1]

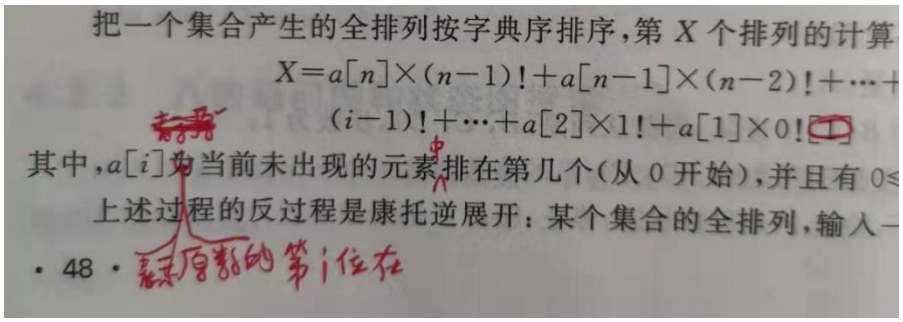

产生的全排列按字典序排序，第 X 个排列的计算公式如下：

$$X=a[n]\times(n-1)!+a[n-1]\times(n-2)!+\cdots+a[i]\times(i-1)!+\cdots+a[2]\times1!+a[1]\times0!$$

未出现的元素排在第几个(从 0 开始)，并且有 $0\leqslant a[i]<i\ (1\leqslant i\leqslant n)$。

反过程是康托逆展开：某个集合的全排列，输入一个数字 k，返回第 k

（6）48 页。把倒数第 2 行的“a[i]为当前未出现的元素排在第几个”改为“a[i]表示原数的第 i 位在当前未出现的元素中排在第几个”


把一个集合产生的全排列按字典序排序，第 X 个排列的计算

$$X=a[n]\times(n-1)!+a[n-1]\times(n-2)!+\cdots+$$

$$(i-1)!+\cdots+a[2]\times1!+a[1]\times0!$$

其中，$a[i]$为当前未出现的元素排在第几个(从 0 开始)，并且有 0≤

上述过程的反过程是康托逆展开：某个集合的全排列，输入一

· 48 ·

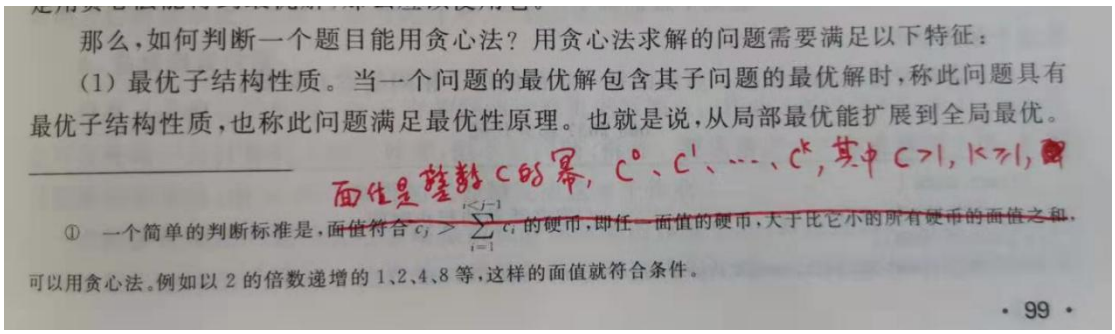
（7）59 页。删除　int t;

```
        return false;
    }
    int main(){
        int t;
        while(cin>>n && n){
            int depth;
            for(depth = 0 ; ; depth ++){
                val[pos = 0] = 1;
                if(ida(0, depth)) break;
```

（8）99 页。

最下面的脚注，原来的说法是错误的。现在改为：

一个简单的判断标准是，面值是整数 c 的幂，c^0、c^1、… 、c^k，其中 c>1，k≥1，可以用贪心法。例如以 2 的倍数递增的 1、2、4、8 等，这样的面值就符合条件。

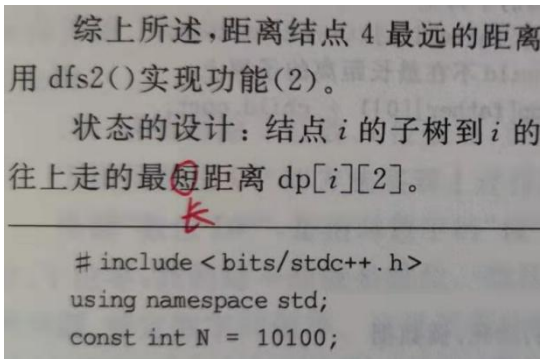

那么，如何判断一个题目能用贪心法？用贪心法求解的问题需要满足以下特征：

(1) 最优子结构性质。当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质，也称此问题满足最优性原理。也就是说，从局部最优能扩展到全局最优。

① 一个简单的判断标准是，面值符合 $c_j>\sum_{i=1}^{i<j-1}c_i$ 的硬币，即任一面值的硬币，大于比它小的所有硬币的面值之和，可以用贪心法。例如以 2 的倍数递增的 1、2、4、8 等，这样的面值就符合条件。

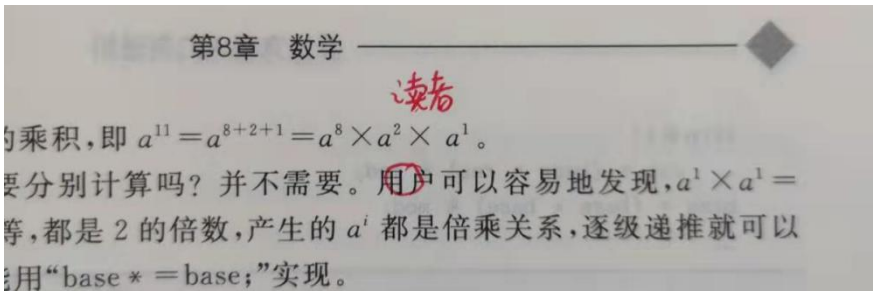
· 99 ·

（9）143 页。把“最短距离”改为“最长距离”。

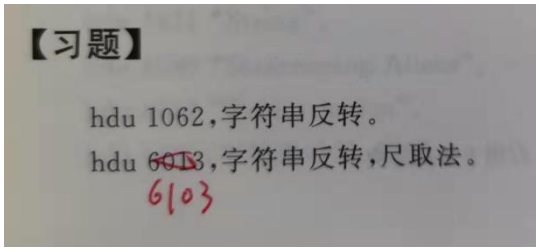

综上所述，距离结点 4 最远的距离
用 dfs2()实现功能(2)。
状态的设计：结点 i 的子树到 i 的
往上走的最短距离 dp[i][2]。

```
#include<bits/stdc++.h>
using namespace std;
const int N = 10100;
```

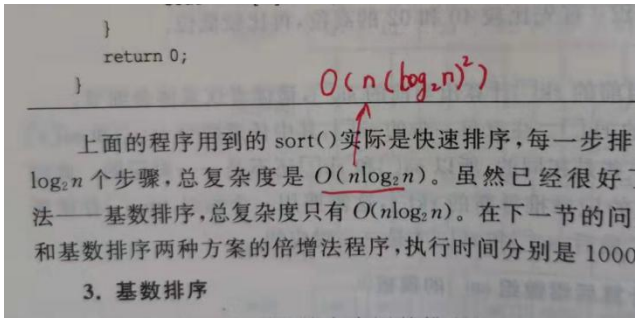
（10）157 页，第 2 行，“用户”改为“读者”



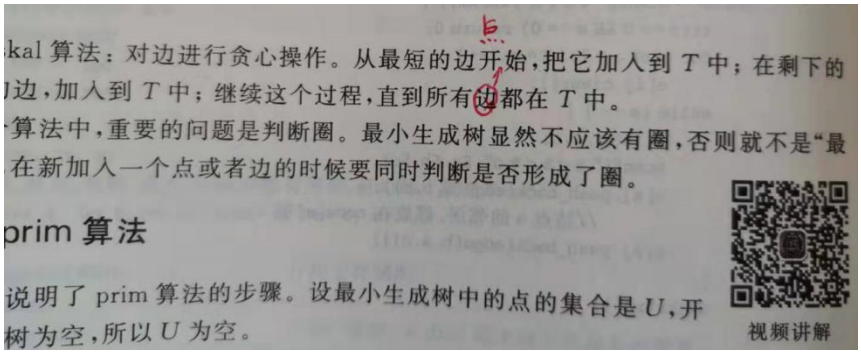
（11）193 页。把 6013 改为 6103



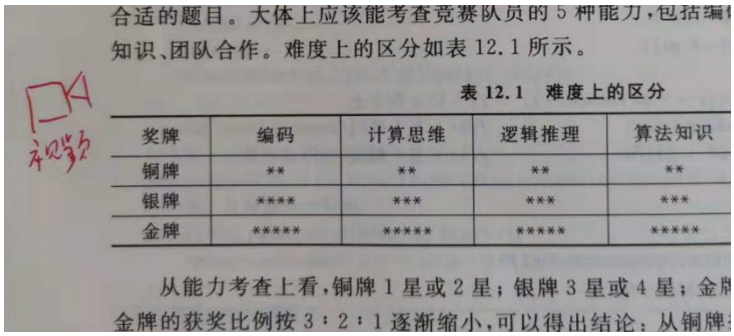
（12）210 页。改为：总复杂度是 $O(n(\log_2 n)^2)$



（13）254 页。kruskal 算法，把“所有边都在 T 中”，改为“所有点都在 T 中”



（14）316 页。加一个视频：“第 12 章-表 12.1.mp4”。



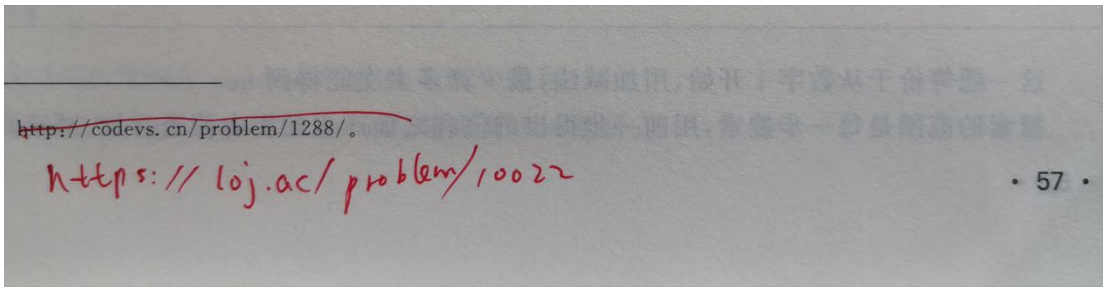
合适的题目。大体上应该能考查竞赛队员的 5 种能力，包括编
知识、团队合作。难度上的区分如表 12.1 所示。

表 12.1 难度上的区分

奖牌	编码	计算思维	逻辑推理	算法知识
铜牌	**	**	**	**
银牌	****	***	***	***
金牌	*****	*****	*****	*****

从能力考查上看，铜牌 1 星或 2 星；银牌 3 星或 4 星；金
金牌的获奖比例按 3∶2∶1 逐渐缩小，可以得出结论：从铜牌

（15）57 页，脚注的 http://codevs.cn/problem/1288 连不上了，改为：https://loj.ac/problem/10022



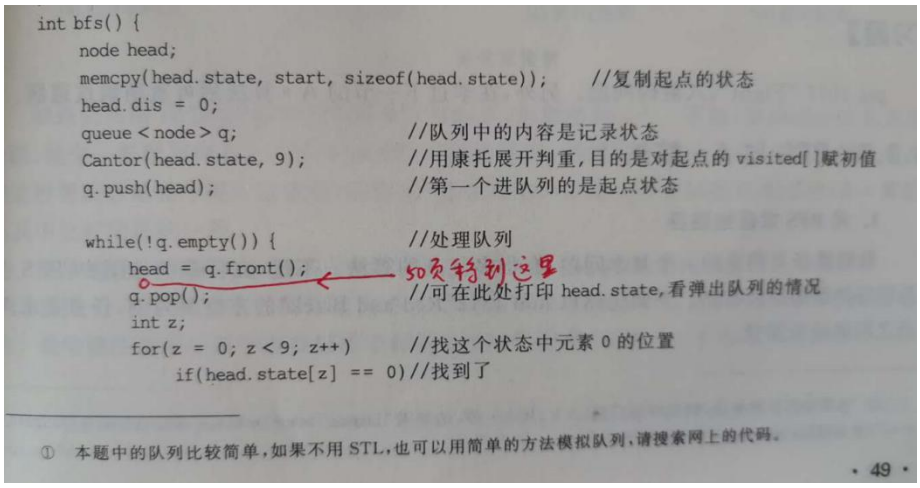
~~http://codevs.cn/problem/1288/~~。
https://loj.ac/problem/10022

· 57 ·

4、第 5 次 印刷

（1）把 50 页的这 2 句代码移动到 49 页，并把其中的 newnode 改为 head

```
if(memcmp(head.state,goal,sizeof(goal)) == 0)    //与目标状态对比
            return head.dis;                      //到达目标状态，返回距离，结束
```

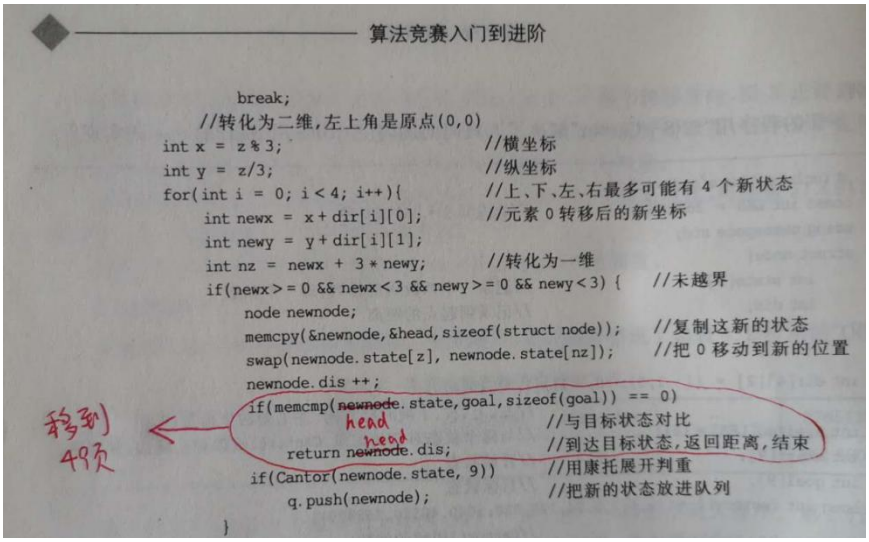


```
int bfs() {
    node head;
    memcpy(head.state, start, sizeof(head.state));    //复制起点的状态
    head.dis = 0;
    queue<node> q;                  //队列中的内容是记录状态
    Cantor(head.state, 9);          //用康托展开判重，目的是对起点的 visited[]赋初值
    q.push(head);                   //第一个进队列的是起点状态

    while(!q.empty()) {             //处理队列
        head = q.front();           //50页移到这里
        q.pop();                    //可在此处打印 head.state,看弹出队列的情况
        int z;
        for(z = 0; z<9; z++)        //找这个状态中元素 0 的位置
            if(head.state[z] == 0)//找到了
```

① 本题中的队列比较简单，如果不用 STL，也可以用简单的方法模拟队列，请搜索网上的代码。

· 49 ·



的判断放在了后面，当初始棋盘和目标棋盘一样时，输出了 2，但是应该输出 0。

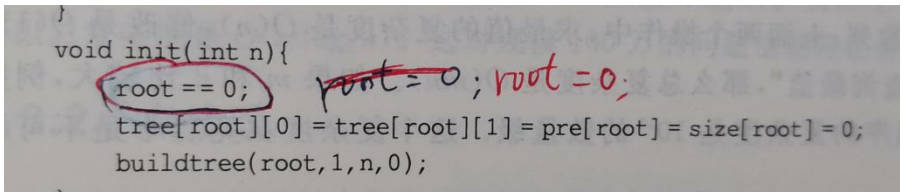
修改：49 页的代码把 if（memcmp）.....这句挪到取队头元素的下面就好了，当然里面的 newnode.state 要换成 head.state

```
int bfs() {
    node head;
    memcpy(head.state, start, sizeof(head.state));  //复制起点的状态
    head.dis = 0;
    queue <node> q;           //队列中放状态
    Cantor(head.state, 9);  //用康托展开判重，目的是对起点的 visited[]赋初值
    q.push(head);             //第一个进队列的是起点状态

    while(!q.empty()) {              //处理队列
        head = q.front();
        if(memcmp(head.state,goal,sizeof(goal)) == 0)   //与目标状态对比
                return head.dis;              //到达目标状态，返回距离，结束
        q.pop();                          //可在此处打印 head.state，看弹出队列的情况
        int z;
        for(z = 0; z < 9; z++)          //找这个状态中元素 0 的位置
            if(head.state[z] == 0)     //找到了
                break;
        //转化为二维，左上角是原点(0, 0)。
        int x = z%3;           //横坐标
        int y = z/3;           //纵坐标
        for(int i = 0; i < 4; i++){   //上、下、左、右最多可能有 4 个新状态
            int newx = x+dir[i][0];     //元素 0 转移后的新坐标
            int newy = y+dir[i][1];
            int nz = newx + 3*newy;     //转化为一维
            if(newx>=0 && newx<3 && newy>=0 && newy<3) {//未越界
                node newnode;
                memcpy(&newnode,&head,sizeof(struct node));//复制这新的状态
```

```
                swap(newnode.state[z], newnode.state[nz]);//把 0 移动到新的位置
                newnode.dis ++;
                if(Cantor(newnode.state, 9))          //用康托展开判重
                    q.push(newnode);                  //把新的状态放进队列
            }
        }
    }
    return -1;              //没找到
}
```

（2）83 页，init()中 root==0,应该为 root=0



（3）87 页，ans[5]改成 ans[4]

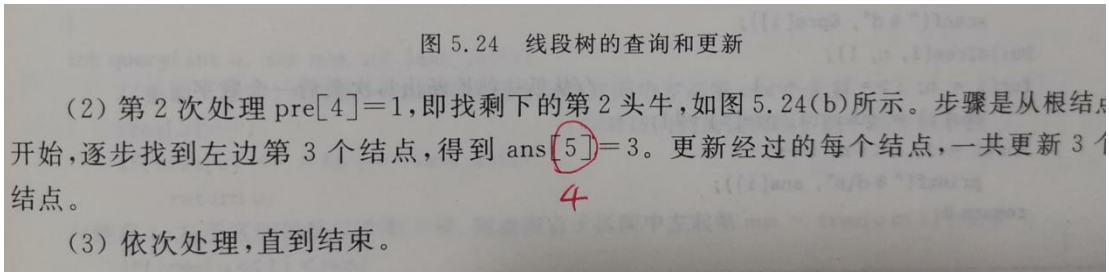


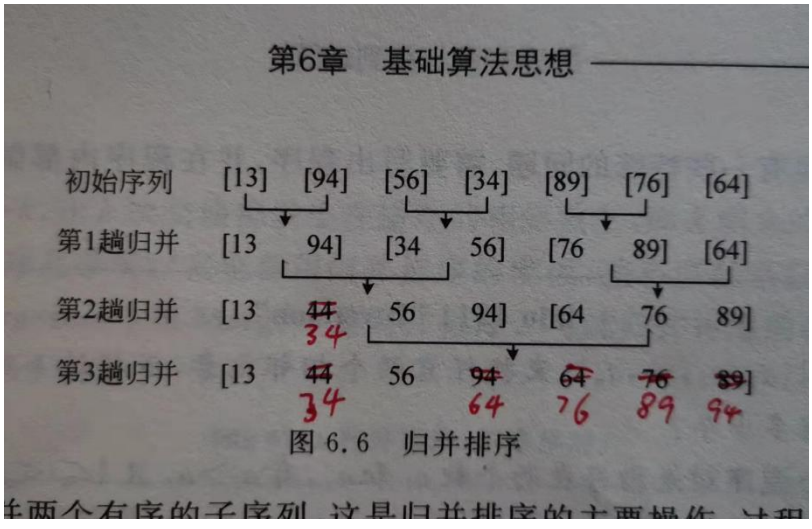
图 5.24　线段树的查询和更新

（2）第 2 次处理 pre[4]＝1，即找剩下的第 2 头牛，如图 5.24(b)所示。步骤是从根结点开始，逐步找到左边第 3 个结点，得到 ans[5]＝3。更新经过的每个结点，一共更新 3 个结点。

4

（3）依次处理，直到结束。

（4）109 页，图 6.6，错了很多数字。



第6章　基础算法思想

初始序列	[13]	[94]	[56]	[34]	[89]	[76]	[64]
第1趟归并	[13	94]	[34	56]	[76	89]	[64]
第2趟归并	[13	~~44~~ 34	56	94]	[64	76	89]
第3趟归并	[13	~~44~~ 34	56	~~94~~ 64	~~64~~ 76	~~76~~ 89	~~89~~ 94]

图 6.6　归并排序

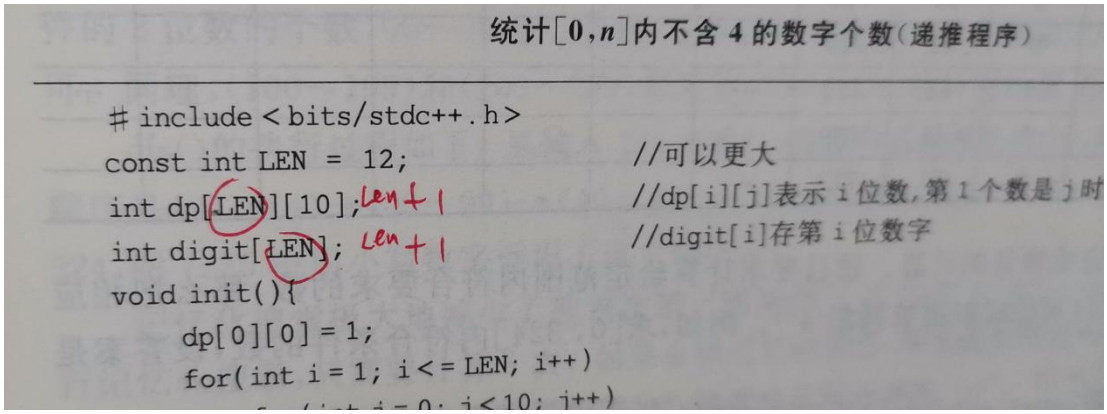
（5）在原书 p145 hdu2089 题的代码中，这一段：

```
int dp[LEN][10];
int digit[LEN];
```

改为：

```
int dp[LEN+1][10];
```

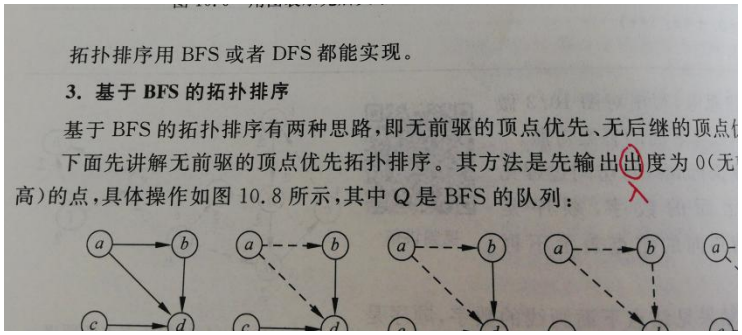
int digit[LEN+1];



统计[0,n]内不含4的数字个数(递推程序)

```
#include<bits/stdc++.h>
const int LEN = 12;                    //可以更大
int dp[LEN][10];                       //dp[i][j]表示i位数,第1个数是j时
int digit[LEN];                        //digit[i]存第i位数字
void init(){
    dp[0][0] = 1;
    for(int i = 1; i<=LEN; i++)
```

（6）220 页。应该是“先输出入度为 0”



拓扑排序用 BFS 或者 DFS 都能实现。

3. 基于 BFS 的拓扑排序

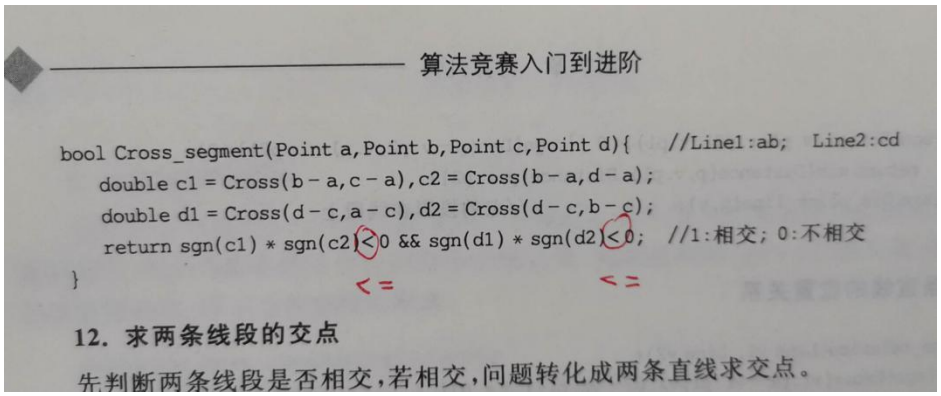
基于 BFS 的拓扑排序有两种思路，即无前驱的顶点优先、无后继的顶点

下面先讲解无前驱的顶点优先拓扑排序。其方法是先输出出度为 0(无

高)的点，具体操作如图 10.8 所示，其中 Q 是 BFS 的队列：

(7) 279 页那个 11：判断两个线段是否相交；不应该只判断两个线段都满足一个线段的两端在令一条线段的两端，如果两条线段相交的话，还有一种情况为一个线段的端点在另一条线段上。

把 280 页第 4 行的<0 >0 改为 <=0 >=0



算法竞赛入门到进阶

```
bool Cross_segment(Point a,Point b,Point c,Point d){    //Line1:ab;  Line2:cd
    double c1 = Cross(b-a,c-a),c2 = Cross(b-a,d-a);
    double d1 = Cross(d-c,a-c),d2 = Cross(d-c,b-c);
    return sgn(c1) * sgn(c2)<0 && sgn(d1) * sgn(d2)<0;  //1:相交; 0:不相交
}
```

<= <=

12. 求两条线段的交点

先判断两条线段是否相交，若相交，问题转化成两条直线求交点。