2017年蓝桥杯软件类大学A组第1题、第2题

华东理工大学计算机系教师　罗勇军

原文链接：https://blog.csdn.net/weixin_43914593/category_10721247.html

文章目录

1. 迷宫

题目链接： http://oj.ecustacm.cn/problem.php?id=1317

题目描述：

X星球的一处迷宫游乐场建在某个小山坡上。它是由10x10相互连通的小房间组成的。

房间的地板上写着一个很大的字母。我们假设玩家是面朝上坡的方向站立，则：

L表示走到左边的房间，R表示走到右边的房间，U表示走到上坡方向的房间，D表示走到下坡方向的房间。

X星球的居民有点懒，不愿意费力思考。他们更喜欢玩运气类的游戏。这个游戏也是如此！

开始的时候，直升机把100名玩家放入一个个小房间内。玩家一定要按照地上的字母移动。

迷宫地图如下：

```
UDDLUULRUL
UURLLLRRRU
RRUURLDLRD
RUDDDDUUUU
```

```
URUDLLRRUU
DURLRLDLRL
ULLURLLRDU
RDLULLRDDD
UUDDUDUDLL
ULRDLUURRR
```

请你计算一下，最后，有多少玩家会走出迷宫? 而不是在里边兜圈子。
输出：输出一个整数表示答案

（1）投机取巧的搞法

第1题是填空题，只交答案就行了。如果不想编码，直接用手一个个去数那100个点，几分钟就数完了，答案是31，比编码还要快。

在 http://oj.ecustacm.cn/ 上这样交就能AC：

```
#include<iostream>
using namespace std;
int main(){
    cout << 31 << endl;
    return 0;
}
```

（2）还是用这题来练练**DFS编码**吧

题解：

一道搜索题，暴力dfs，代码简短。

下面是华东理工大学软件192班倪文迪的代码。罗老师注：倪文迪没有用递归写DFS。参考这篇用递归写的DFS，更好懂：https://www.cnblogs.com/-citywall123/p/12316760.html

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<string>
#include<vector>
#include<cmath>
#include<cstdio>
using namespace std;

int mp[20][20];
int vis[20][20];
bool tag[200];
int cnt[200];
```

```
int X[] = {0, 0, 0, -1, 1};
int Y[] = {0, -1, 1, 0, 0};

void solve(int x, int y, int id)
{
        while(x >= 1 && x <= 10 && y >= 1 && y <= 10 && !vis[x][y]){
                vis[x][y] = id;
                cnt[id]++;
                int now = mp[x][y];
                x += X[now];
                y += Y[now];
        }
        if(x < 1 || x > 10 || y < 1 || y > 10){
                tag[id] = true;
                return ;
        }
        if(vis[x][y]){
                if(tag[vis[x][y]])      tag[id] = true;
        }
        return ;
}

int main(){
        for(int i = 1 ; i <= 10 ; i++){
                string s; cin >> s;
                for(int j = 0 ; j < 10 ; j++){
                        if(s[j] == 'L') mp[i][j + 1] = 1;
                        else if(s[j] == 'R') mp[i][j + 1] = 2;
                        else if(s[j] == 'U') mp[i][j + 1] = 3;
                        else if(s[j] == 'D') mp[i][j + 1] = 4;
                }
        }

        int     id = 1;
        for(int i = 1 ; i <= 10 ; i++){
                for(int j = 1 ; j <= 10 ; j++){
                        if(!vis[i][j]){
                                solve(i, j, id);
                                id++;
                        }
                }
        }
        /*for(int i = 1 ; i <= 10 ; i++){
                for(int j = 1 ; j <= 10 ; j++){
                        printf("%3.d",vis[i][j]);
                }
                cout << endl;
```

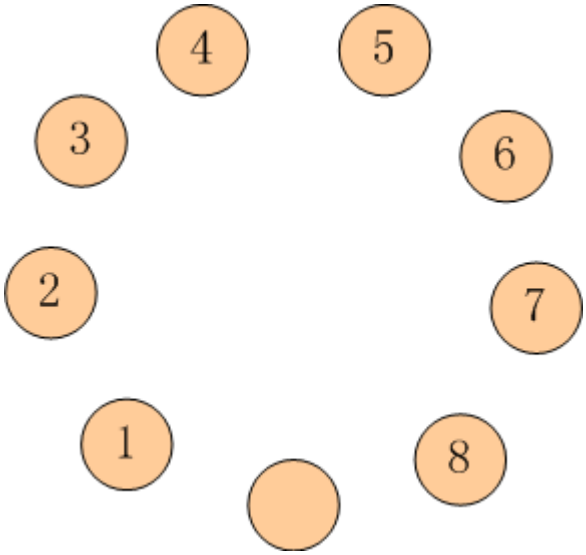
```
        }*/

        int res = 0;
        for(int i = 1 ; i < id ; i++){
                if(tag[i])      res += cnt[i];
        }
        printf("%d\n", res);

        return 0;
}
```

2. 跳蚱蜢

题目链接： http://oj.ecustacm.cn/problem.php?id=1318

题目描述：

如图所示， 有9只盘子，排成1个圆圈。其中8只盘子内装着8只蚱蜢，有一个是空盘。



我们把这些蚱蜢顺时针编号为 1~8。每只蚱蜢都可以跳到相邻的空盘中，也可以再用点力，越过一个相邻的蚱蜢跳到空盘中。

请你计算一下，如果要使得蚱蜢们的队形改为按照逆时针排列，并且保持空盘的位置不变（也就是1-8换位，2-7换位,...），至少要经过多少次跳跃？

输出：输出一个整数表示答案

又是一道填空题，能用手数出答案吗？

提前告诉大家，答案是20。数字好像不大，可是，用手数出来好像不可能，还是老实编码吧。

2.1 建模

直接让蚱蜢跳到空盘有点麻烦，因为有很多蚱蜢在跳，跳晕了。如果看成空盘跳到蚱蜢的位置就简单多了，只有一个空盘在跳。

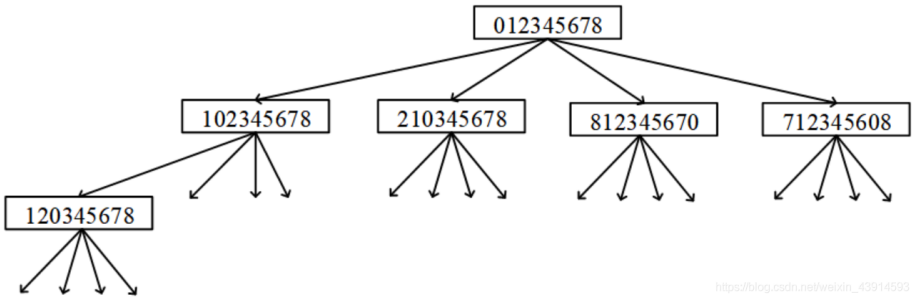
题目给的是一个圆圈，不好处理，此时祭出一个建模大法：“**化圆为线**”！ 把空盘看成0，那么有9个数字{0,1,2,3,4,5,6,7,8}，一个圆圈上的9个数字，拉直成了一条线上的9个数字。

等等，这不就是八数码问题吗？八数码是经典的BFS问题。

八数码有9个数字{0,1,2,3,4,5,6,7,8}，它有9！=362880种排列。也不多，

本题的初始状态是“012345678”，终止状态是“087654321”。

从初始状态跳一次，有4种情况：



2.2 判重

这题要是写个裸的BFS，不判重，能运行出来吗？

第1步到第2步，有4种跳法；第2步到第3步，有4*4种；...；第20步，有4^20=1万亿种！太多了！BFS的队列也放不下呀。

还是得判重，如果跳到一个曾经出现过的情况，就不用往下跳了。八数码只有9！=362880种排列，好判。

如何判重？比赛的时候紧张，当然得用STL，用map、set判重都行，效率都好。另外，有一种数字方法，叫康托判重，得自己写，一般不用。

2.3 map判重

把9个数字的排列定义为一种状态，即字符串s，例如初始状态“012345678”是一个串。对应交换之后产生的s，我们可以使用map判重，将该字符串以及它首次出现的时间作为一个单位推入队列中，由于BFS的性质我们能看出，首次找到结果状态的时间t即是最小的答案。

倪文迪的代码：

```
#include<bits/stdc++.h>
using namespace std;

struct node
```

```
{
        node(){}
        node(string ss, int tt){
                s = ss, t = tt;
        }
        string s;
        int t;
};

map<string, bool> mp;
queue<node> q;

void solve()
{
        while(!q.empty()){
                node now = q.front();
                q.pop();
                string s = now.s; int t = now.t;
                if(s == "087654321"){
                        cout << t << endl;
                        break;
                }
                int i;
                for(i = 0 ; i < 10 ; i++){
                        if(s[i] == '0') break;
                }
                for(int j = i - 2 ; j <= i + 2 ; j++){
                        int k = (j + 9) % 9;
                        if(k == i)      continue;
                        char tmp;
                        tmp = s[i];s[i] = s[k];s[k] = tmp;
                        if(!mp[s]){
                                mp[s] = true;
                                q.push(node(s, t + 1));
                        }
                        tmp = s[i];s[i] = s[k];s[k] = tmp;
                }
        }
}

int main(){
        string s = "012345678";
        q.push(node(s, 0));
        mp[s] = true;
        solve();
```

```
        return 0;
}
```

2.4 set判重

代码参考：https://blog.csdn.net/crazymooo/article/details/108816699

2.5 康托判重

map和set的判重效率是很高的。另外有一种数学判重方法，叫做康托判重，运行起来比map和set快，下面介绍一下。

在《算法竞赛入门到进阶》（清华大学出版社，罗勇军著）47页详解了八数码的康托判重。这里附上截图：

1. 八数码问题

在一个 3×3 的棋盘上放置编号为 1～8 的 8 个方块，每个占一格，另外还有一个空格。与空格相邻的数字方块可以移动到空格里。任务 1：指定初始棋局和目标棋局（如图 4.2 所示），计算出最少的移动步数；任务 2：输出数码的移动序列。

把空格看成 0，一共有 9 个数字。

输入样例：

1 2 3 0 8 4 7 6 5

1 0 3 8 2 4 7 6 5

输出样例：

2

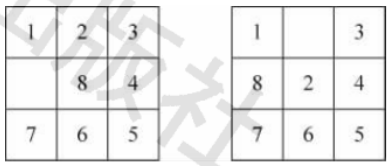


图 4.2　初始棋局和目标棋局

把一个棋局看成一个状态图，总共有 9!＝362 880 个状态。从初始棋局开始，每次移动转到下一个状态，到达目标棋局后停止。

八数码问题是一个经典的 BFS 问题。前面章节中提到 BFS 是从近到远的扩散过程，适合解决最短距离问题。八数码从初始状态出发，每次转移都逐步逼近目标状态。每转移一次，步数加一，当到达目标时，经过的步数就是最短路径。

图 4.3 是样例的转移过程。该图中起点为(*A*，0)，*A* 表示状态，即{1 2 3 0 8 4 7 6 5}这个棋局；0 是距离起点的步数。从初始状态 *A* 出发，移动数字 0 到邻居位置，按左、上、右、下的顺时针顺序，有 3 个转移状态 *B*、*C*、*D*；目标状态是 *F*，停止。

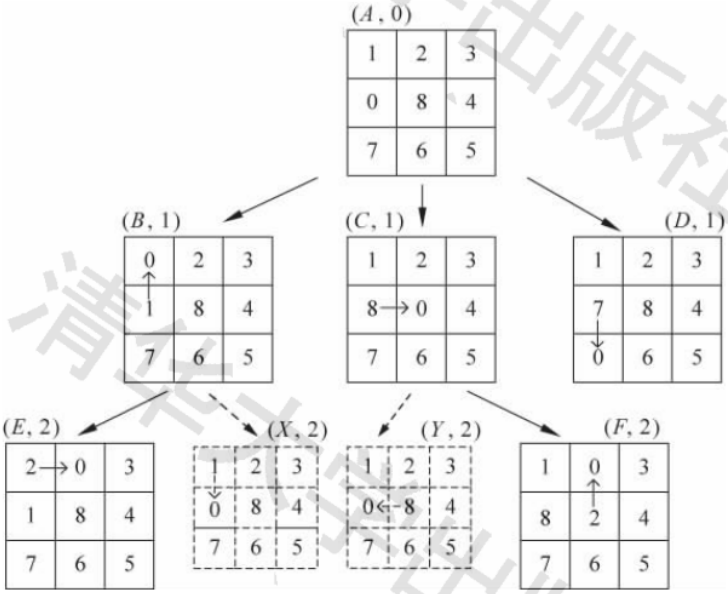


图 4.3　八数码问题的搜索树

用队列描述这个 BFS 过程：

(1) A 进队，当前队列是{A}；

(2) A 出队，A 的邻居 B、C、D 进队，当前队列是{B, C, D}，步数为 1；

(3) B 出队，E 进队，当前队列是{C, D, E}，E 的步数为 2；

(4) C 出队，转移到 F，检验 F 是目标状态，停止，输出 F 的步数 2。

仔细分析上述过程，发现从 B 状态出发实际上有 E、X 两个转移方向，而 X 正好是初始状态 A，重复了。同理 Y 状态也是重复的。如果不去掉这些重复的状态，程序会产生很多无效操作，复杂度大大增加。因此，八数码的重要问题其实是判重。

如果用暴力的方法判重，每次把新状态与 9!=362 880 个状态对比，可能有 9!×9! 次检查，不可行。因此需要一个快速的判重方法。

本题可以用数学方法“康托展开(Cantor Expansion)”来判重。

2. 康托展开

康托展开是一种特殊的哈希函数。在本题中，康托展开完成了如表 4.2 所示的工作：

表 4.2　本题中康托展开完成的工作

状　　态	012345678	012345687	012345768	012345786	…	876543210
Cantor	0	1	2	3	…	362 880−1

第 1 行是 0～8 这 9 个数字的全排列，共 9!=362 880 个，按从小到大排序。第 2 行是每个排列对应的位置，例如最小的{012345678}在第 0 个位置，最大的{876543210}在最后的 362 880−1 这个位置。

函数 Cantor()实现的功能是：输入一个排列，即第 1 行的某个排列，计算出它的 Cantor 值，即第 2 行对应的数。

Cantor()的复杂度为 $O(n^2)$，n 是集合中元素的个数。在本题中，完成搜索和判重的总复杂度是 $O(n!n^2)$，远比用暴力判重的总复杂度 $O(n!n!)$小。

有了这个函数，八数码的程序能很快判重：每转移到一个新状态，就用 Cantor()判断这个状态是否处理过，如果处理过，则不转移。

下面举例讲解康托展开的**原理**。

例子：判断 2143 是{1，2，3，4}的全排列中第几大的数。

计算排在 2143 前面的排列数目，可以将问题转换为以下排列的和：

(1) 首位小于 2 的所有排列。比 2 小的只有 1 一个数，后面 3 个数的排列有 3×2×1=3!个(即 1234、1243、1324、1342、1423、1432)，写成 1×3!=6。

(2) 首位为 2、第 2 位小于 1 的所有排列。无，写成 0×2!=0。

(3) 前两位为 21、第 3 位小于 4 的所有排列。只有 3 一个数(即 2134)，写成 1×1!=1。

(4) 前 3 位为 214、第 4 位小于 3 的所有排列。无，写成 0×0!=0。

求和：1×3!+0×2!+1×1!+0×0!=7，所以 2143 是第 8 大的数。如果用 int visited[24]数组记录各排列的位置，{2143}就是 visited[7]；第一次访问这个排列时，置 visited[7]=1；当再次访问这个排列的时候发现 visited[7]等于 1，说明已经处理过，判重。

根据上面的例子得到康托展开公式。

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：

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

其中，$a[i]$表示原数的第 i 位在当前未出现的元素中排在第几个(从 0 开始)，并且有 $0\leqslant a[i]<i$ $(1\leqslant i\leqslant n)$。

上述过程的反过程是康托逆展开：某个集合的全排列，输入一个数字 k，返回第 k 大的

以上是截图。

下面是书中的代码，用“BFS + 康托(Cantor)”解决了八数码问题，其中BFS用STL的queue实现。

```
#include<bits/stdc++.h>
const int LEN = 362888;        //状态共9!=362880种
using namespace std;
struct node{
    int state[9];        //记录一个八数码的排列，即一个状态
    int dis;              //记录到起点的距离
};

int dir[4][2] = {{-1,0}, {0,-1},{1,0},{0,1}};
          //左、上、右、下顺时针方向。左上角坐标是(0,0)
int visited[LEN]={0};  //与每个状态对应的记录，Cantor函数对它置数，并判重
int start[9];              //开始状态
int goal[9];               //目标状态
long int factory[] = {1,1,2,6,24,120,720,5040,40320,362880};
                              //Cantor用到的常数
bool Cantor(int str[], int n) {      //用康托展开判重
    long result = 0;
    for(int i = 0; i < n; i++) {
        int counted = 0;
        for(int j = i+1; j < n; j++) {
            if(str[i] > str[j])        //当前未出现的元素中是排在第几个
```

```
                ++counted;
            }
            result += counted*factory[n-i-1];
        }
        if(!visited[result]) {            //没有被访问过
            visited[result] = 1;
            return 1;
        }
        else
            return 0;
    }
    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();
            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(memcmp(newnode.state,goal,sizeof(goal)) == 0)
                                                            //与目标状态对比
                        return newnode.dis;              //到达目标状态，返回距离，结束
                    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
```

```
}
int main(){
    for(int i = 0; i < 9; i++)  cin >> start[i];        //初始状态
    for(int i = 0; i < 9; i++)  cin >> goal[i];         //目标状态
    int num = bfs();
    if(num != -1)  cout << num << endl;
    else           cout << "Impossible" << endl;
    return 0;
}
```

2.6 扩展学习

另外，可以用**双向BFS**来进一步优化八数码，参考这篇博文：
https://blog.csdn.net/weixin_43914593/article/details/104761298

参考文献：《算法竞赛入门到进阶》 清华大学出版社，网购：京东 当当