

中山大学数据科学与计算机学院本科生实验报告

(2016 学年春季学期)

课程名称: Algorithm design

任课教师: 张子臻

年级	15	专业(方向)	软件工程(移动信息工程)
学号	15352408	姓名	张稼伟
电话	13531810182	Email	709075442@qq.com
开始日期		完成日期	

目录

1. 实验题目	3
Soj 1150 简单魔板	3
Soj 1151 魔板	3
Soj 1515 魔板 C	3
Soj 1317 Sudoku	4
Soj 1215 脱离地牢	4
Soj 1171 The Game of Efil	4
Soj 1014 Specialized Four-Dig	4
Soj 1709 PropBot	4
Soj 1108 Online Selection	5
Soj 6276 Pawns	5
Soj 1471 No Left Turns	5
Soj 1182 Context-Free Clock	5
Soj 1710 Painted Calculator	5
Soj 7144 Different Triangles	6

2.实验目的	6
3.程序设计	6
Soj 1150 简单魔板和 Soj1151 魔板.....	6
Soj 1515 魔板 C.....	9
Soj 1317 Sudoku.....	10
Soj 1215 脱离地牢.....	12
Soj 1171 The Game of Efil.....	13
Soj 1014 Specialized Four-Dig	14
Soj 1709 PropBot	15
Soj 1108 Online Selection	15
Soj 6276 Pawns	16
Soj 1471 No Left Turns	18
Soj 1182 Context-Free Clock.....	19
Soj 1710 Painted Calculator	20
Soj 7144 Different Triangles	21
5.实验总结与心得	22
附录、提交文件清单	22

1.实验题目

搜索专题

Soj 1150 简单魔板

题意：魔板由 8 个大小相同方块组成，分别用涂上不同颜色，用 1 到 8 的数字表示。其初始状态是

1 2 3 4

8 7 6 5

对魔板可进行三种基本操作：

A 操作（上下行互换）：

8 7 6 5

1 2 3 4

B 操作（每次以行循环右移一个）：

4 1 2 3

5 8 7 6

C 操作（中间四小块顺时针转一格）：

1 7 2 4

8 6 3 5

现给定步数 n 和目标状态。问能否在 n 步内从初始状态到达目标状态。若不能，则输出-1。若能，则先输出需要的操作数，再输出每一步进行的操作。

约束： $n \leq 10$

Soj 1151 魔板

题意：题意同 1150，唯一区别只有 n 是无限制的。

Soj 1515 魔板 C

题意：此题与 1151 的区别只在于初始状态和三种基本操作有所不同，其他一样。

其初始状态为：

1 2 3 4

5 6 7 8

这题的三种基本操作为：

A 操作（左右两列互换）：

3 4 1 2

7 8 5 6

B 操作（每次以行循环左移一个）：

2 3 4 1

6 7 8 5

C 操作（中间四小块逆时针转一格）：

1 3 7 4

5 2 6 8

约束： $n \leq 100$

Soj 1317 Sudoku

题意：一个 9×9 的大矩阵由 9 个 3×3 的小矩阵组成。数独游戏就是在这个大矩阵中某些格子先填入一些数字，称为初始状态。然后我们要在剩下的格子中填数字，使得大矩阵每一行每一列没有重复数字，每个小矩阵 9 个格也没有重复数字。给定初始状态，求解。若解唯一，输出该方案，若不唯一，输出解的个数，无解输出 no solution。

约束：解的个数不超过 10000

Soj 1215 脱离地牢

题意：给定一个 $n \times m$ 的地图。"."代表通路，"#"代表岩石，"!"代表熔浆。"P"和"H"代表两个人的位置。再给出 4 个字符，为 NSWE 的一个排列，依次表示 P 向北、南、西、东走一步时，H 会向哪个方向走一步。P 不能走到岩石和熔浆的地方，同时由于他的行动导致 H 位置的改变，若 H 碰到岩浆则会死去，若 H 碰到岩石则停留在原地。请问 P 最少移动多少步可以使两人相遇。255 步之后仍不相遇，则输出 "Impossible"

约束： $3 \leq n, m \leq 20$

Soj 1171 The Game of Efil

题意：给定一个 $n \times m$ 的矩阵。在这个矩阵上进行生命游戏。游戏规则如下：1. 如果一个被占领的格子周围有 0, 1, 4, 5, 6, 7, 或 8 个邻居，则该格子的生命体死亡。

2. 如果一个有生命体的各自周围只有 2 或 3 个邻居，则它继续生存

3. 如果一个没有生命体的格子周围有 3 个邻居，则这个格子产生生命体

4. 板的上下边是连通的，左右边也是连通的。

现给定一个初始状态，求有多少个父状态可以达到该初始状态。

约束： $(1 \leq n \times m \leq 16)$

Soj 1014 Specialized Four-Dig

题意：找出所有同时满足下面条件的四位数，从小到大输出：

1. 4 个数字之和等于这个四位数的十六进制形式的各位数字之和。

2. 4 个数字之和等于这个四位数的十二进制形式的各位数字之和

Soj 1709 PropBot

题意：一开始机器人在点(0,0)上，指向+x 方向。机器人每秒有两种行动

方法：向当前方向走 10 个单位长度，或者向右转 45 度。现给出目标点以及最多可以行动的时间 t ，求出在这个过程中，机器人可以达到的离目标点最近的距离，（可以自由选择机器人的行动方法）输出保留 6 位小数。

约束： $0 \leq t \leq 24$ ， t 为整数，坐标范围为 $[-100, 100]$ 的浮点数

Soj 1108 Online Selection

题意：有 n 个问题，编号 $0 \sim n-1$ 每个答案的问题是 0 或 1。回答者有 n 个状态，编号为 $0 \sim n-1$ 。一个人在编号为 i 的状态则回答编号为 i 的问题，假设他回答了 0，则跳转到状态 i_0 ，否则跳转到状态 i_1 。若他的回答=问题的答案 a_i ，则分数+1。现给定 n, k, m ，每个问题的 i_0, i_1 和 a_i 。求一个人回答 k 次问题，得到 m 分，最多可能回答“0”多少次。

约束： $n \leq 20$ ， $k \leq 30$

Soj 6276 Pawns

题意：在一个 $1 \times N$ 的棋盘上玩游戏。棋盘格子从左到右编号 $1 \sim N$ ，上面初始在一些格子有黑子或者白子。每一步可以走黑子或者白子，规则如下：

若走黑子：

1. 黑子可向右一步到旁边的空格子。
2. 如果右边格子有棋子，但是右边的右边格子是空的，那么可以直接跳到右边的右边那个格子。

若走白子：

1. 白子可向左一步到旁边的空格子。
2. 如果左边格子有棋子，但是左边的左边格子是空的，那么可以直接跳到左边的左边那个格子。

给定初始状态，求最少需要多少步，使得白子都在最左边依次排列（例如有三个白子，则依次排在格子 1，格子 2，格子 3），黑子都在最右边依次排列（格子 N ，格子 $N-1, \dots$ ）

约束： $2 \leq N \leq 13$ ，题目保证一定有解，棋盘上至少有黑白子各一个。

Soj 1471 No Left Turns

题意：给出一个 $r \times c$ 矩阵，'X' 不能走，' ' 可以走，'S' 是起点，'F' 是终点，问不左转的情况下从起点到终点所用最少步数是多少（保证路径存在）

约束： $3 \leq r, c \leq 20$

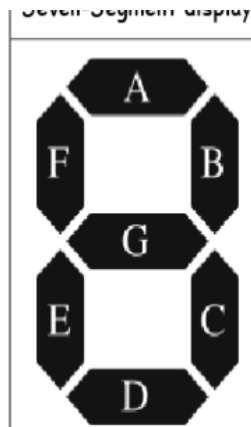
Soj 1182 Context-Free Clock

题意：给出一个时刻 t 和角度 θ ，求时刻 ans ，使得在 ans 时刻中，时针按顺时针到分针的角度为 θ ，且该时刻离 t 最近。得到 ans 后将 ans 按照秒截断后输出（例如答案是 $12:34:35'54$ 的话，输出

12:34:35)

Soj 1710 Painted Calculator

题意：一个计算器，显示数字是 7 段码，如下



7 个 led 每个 led 消耗 5 毫安电量如果那个 led 亮的话。现给出两个操作数 X 和 Y 以及结果 Z 消耗的电量。注意 X, Y, Z 取值都是-999~999 之间的整数。(如果有负数，如 X 是负数，则 X 消耗的电量中有 5 毫安是用来显示电量的。计算器可以进行+, -, *, /操作，求有多少种方案满足 $X \text{ opt } Y = Z$ ，且 X, Y, Z 消耗的电量与题目给出的相同。

Soj 7144 Different Triangles

题意：给定 n 个木棍，请你找出有多少不同的方法可以用其中的三个木棍组成一个三角形。

约束： $3 \leq N \leq 100$

2.实验目的

1. 加深对搜索各种算法的理解，学会如何应用它们。
2. 提高运用课堂所学知识解决实际问题的能力。

3.程序设计

Soj 1150 简单魔板和 Soj1151 魔板

解题思路：这两题唯一的区别只是 1151 比 1150 的允许步数要更大。所以我们只要考虑 1151 的做法就可以直接做完两题。这题最基本的想法就是使用一个队列 queue 实现一个广度优先遍历算法，过程如下：先将初始魔板入队列，每次循环时，队首元素出列，判断是否与目标状态一致，若一直则输出，否则对当前魔板分别进行 A、B、C 操作后入队列，并同时记录操作的步骤和次数。这样的做法在步数小的时候是完全没有问题的。如在 1150 步数限制为 10 的时候最坏情况也就只有 $3^{10}=59049$ ，在可承受范围内。而在

1151 就不可承受了。这时我们就需要对搜索进行剪枝。实际上模板的排列组合之后 $8! = 40320$ 种。而在搜索的时候会因为操作构成回路，即状态 A 现在搜过了，但是 k 步之后又搜到了，这是完全没有必要的，要得到步数最小的解，一个状态只用搜一次就够了。所以我们可以使用 HASH，判断已搜过的状态就不加入队列。我的 hash 办法是将两行数字依次接成一个十位数存进 map 中判断。

时间复杂度 $O(k+k\log k)$ ，k 为搜过的状态数，最大为 $8!$ 。

代码：

```
1 #include<iostream>
2 #include<queue>
3 #include<map>
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 using namespace std;
6 struct node{
7     int a[2][4],dep;
8     string s;
9     node(){
10         rep(i,0,3){
11             a[0][i]=i+1;
12             a[1][i]=8-i;
13             s="";
14         }
15         dep=0;
16     }
17 }ed;
18 int n;
19 map<int,int>f;
20 int HASH(node no){
21     int x=0;
22     rep(i,0,3)x=x*10+no.a[0][i];
23     rep(i,0,3)x=x*10+no.a[1][i];
24     int &ret=f[x];
25     if (ret!=0)return 0;
26     ret=1;
27     return 1;
28 }
29 int check(node no){
30     rep(i,0,1)
31         rep(j,0,3)
32             if (no.a[i][j]!=ed.a[i][j])
33                 return 0;
34     return 1;
35 }
36 void change(node &next,int ty,node &no){
37     if (ty==1){
38         rep(i,0,3){
39             next.a[0][i]=no.a[1][i];
40             next.a[1][i]=no.a[0][i];
41         }
42         next.s=no.s+"A";
43     }
```

```

44     else if (ty==2){
45         rep(i,0,2){
46             next.a[0][i+1]=no.a[0][i];
47             next.a[1][i+1]=no.a[1][i];
48         }
49         next.a[0][0]=no.a[0][3];
50         next.a[1][0]=no.a[1][3];
51         next.s=no.s+"B";
52     }
53     else {
54         rep(i,0,1)
55         rep(j,0,3)
56             next.a[i][j]=no.a[i][j];
57         next.a[0][1]=no.a[1][1];
58         next.a[0][2]=no.a[0][1];
59         next.a[1][2]=no.a[0][2];
60         next.a[1][1]=no.a[1][2];
61         next.s=no.s+"C";
62     }
63     next.dep=no.dep+1;
64 }
65 void bfs(){
66     node st;
67     queue<node> Q;
68     Q.push(st);
69     if (check(st)){
70         cout<<0<<endl;
71         return;
72     }
73     while (!Q.empty()){
74         node no=Q.front();
75         Q.pop();
76         rep(i,1,3){
77             node next_state;
78             change(next_state,i,no);
79             if (!HASH(next_state)) continue;
80
81             if (check(next_state)){
82                 cout<<next_state.s.length()<<" "<<next_state.s<<endl;
83                 return;
84             }
85             if (next_state.dep<n) Q.push(next_state);
86         }
87     }
88     cout<<-1<<endl;
89     return;
90 }
91 int main(){
92     while (cin>>n){
93         if (n==-1) break;
94         f.clear();
95         rep(i,0,1)
96         rep(j,0,3)
97             cin>>ed.a[i][j];
98         bfs();
99     }
100     return 0;
101 }

```


Soj 1515 魔板 C

题意：做法与 1151 一模一样，只需要修改一下初始状态与 ABC 三种基本操作即可。

时间复杂度 $O(k+k\log k)$ ， k 为搜过的状态数，最大为 $8!$

代码：

```
1 #include<iostream>
2 #include<queue>
3 #include<map>
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 using namespace std;
6 struct node{
7     int a[2][4],dep;
8     string s;
9     node(){
10         rep(i,0,3){
11             a[0][i]=i+1;
12             a[1][i]=i+1+4;
13             s="";
14         }
15         dep=0;
16     }
17 }ed;
18 int n;
19 map<int,int>f;
20 int HASH(node no){
21     int x=0;
22     rep(i,0,3)x=x*10+no.a[0][i];
23     rep(i,0,3)x=x*10+no.a[1][i];
24     int &ret=f[x];
25     if (ret!=0)return 0;
26     ret=1;
27     return 1;
28 }
29 int check(node no){
30     rep(i,0,1)
31         rep(j,0,3)
32             if (no.a[i][j]!=ed.a[i][j])
33                 return 0;
34     return 1;
35 }
36 void change(node &next,int ty,node &no){
37     if (ty==1){
38         rep(i,0,1){
39             next.a[0][i]=no.a[0][i+2];
40             next.a[1][i]=no.a[1][i+2];
41             next.a[0][i+2]=no.a[0][i];
42             next.a[1][i+2]=no.a[1][i];
43         }
44         next.s=no.s+"A";
45     }
46     else if (ty==2){
47         rep(i,0,2){
48             next.a[0][i]=no.a[0][i+1];
49             next.a[1][i]=no.a[1][i+1];
50         }
51         next.a[0][3]=no.a[0][0];
52         next.a[1][3]=no.a[1][0];
53         next.s=no.s+"B";
54     }
```

```

55     else {
56         rep(i,0,1)
57         rep(j,0,3)
58             next.a[i][j]=no.a[i][j];
59         next.a[0][1]=no.a[0][2];
60         next.a[0][2]=no.a[1][2];
61         next.a[1][2]=no.a[1][1];
62         next.a[1][1]=no.a[0][1];
63         next.s=no.s+"C";
64     }
65     next.dep=no.dep+1;
66 }
67 void bfs(){
68     node st;
69     queue<node> Q;
70     Q.push(st);
71     if (check(st)){
72         cout<<0<<endl;
73         return;
74     }
75     if (!n){
76         cout<<-1<<endl;
77         return;
78     }
79     while (!Q.empty()){
80         node no=Q.front();
81         Q.pop();
82         rep(i,1,3){
83             node next_state;
84             change(next_state,i,no);
85             if (!HASH(next_state))continue;
86             if (check(next_state)){
87                 cout<<next_state.s.length()<<" "<<next_state.s<<endl;
88                 return;
89             }
90             if (next_state.dep<n)Q.push(next_state);
91         }
92     }
93     cout<<-1<<endl;
94     return;
95 }
96 }
97 int main(){
98     while (cin>>n){
99         if (n==-1)break;
100         f.clear();
101         rep(i,0,1)
102             rep(j,0,3)
103                 cin>>ed.a[i][j];
104         bfs();
105     }
106     return 0;
107 }

```

Soj 1317 Sudoku

解题思路:这题不涉及求最小步数一类的问题，这启示我们应该使用深度搜索去完成。一开始我的做法是从左上角的格子开始，按照从左到右，从上到下搜索每一个格子，遇到一个空格子则枚举该位置填什么数，这里我使用了三个数组 $r[i][x]$, $c[j][x]$, $num[block][x]$ 判断第 i 行是否填过数字 x ，第 j 列是否填过 x ，第 $block$ 个小九宫格是否填过数字 x ，其中易推得 $block=(i/3)*3 + j/3$ 。然而这样做有可能会超时，因为搜索层数可能达 81

层，而每层有 9 种选择。

这里可以用一点小技巧去剪枝。我们预先算出每个空格子能填多少个数字，将它们按照能填的数字数从小到大排序。然后深搜的时候优先搜选择少的格子。为什么这样子能少搜索呢？举个例子，假如当前搜到这个格子之前的时候它能填的数都被之前的格子填了，而前面的格子一共有 x 种填法让当前格子没有数字填，那么假如我们先搜索当前格子，那之前那 x 种不可行的填法就不会搜到，这样就达到了剪枝的目的。

代码：

```
1 #include<iostream>
2 #include<cstring>
3 #include<algorithm>
4 using namespace std;
5 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
6 char s[10][10],ans_str[10][10],dfs_str[10][10];
7 int r[10][10],c[10][10],num[9][10],ans,len;
8 struct node{
9     int x,y,block,choice;
10     node(){}
11     node(int xx,int yy){
12         x=xx; y=yy; choice=0;
13         block=(x/3)*3 + y/3;
14     }
15     void getChoiceNum(){
16         rep(i,1,9)
17             if (!r[x][i] && !c[y][i] && !num[block][i])
18                 choice++;
19     }
20 }point[81];
21 bool cmp(const node &n1,const node &n2){return n1.choice<n2.choice;}
22 void dfs(int now){
23     if (now==len){
24         if (++ans==1)memcpy(ans_str,dfs_str,sizeof(dfs_str));
25         return;
26     }
27     int x=point[now].x,y=point[now].y,block=point[now].block;
28     rep(i,1,9)
29         if (!r[x][i] && !c[y][i] && !num[block][i]){
30             dfs_str[x][y]=char(i+48);
31             r[x][i]=c[y][i]=num[block][i]=1;
32             dfs(now+1);
33             r[x][i]=c[y][i]=num[block][i]=0;
34         }
35 }
36 int main(){
37     int T;
38     cin>>T;
39     rep(tt,1,T){
40         memset(r,0,sizeof(r));
41         memset(c,0,sizeof(c));
42         | memset(dfs_str,0,sizeof(dfs_str));
43         memset(num,0,sizeof(num));
44         len=0;ans=0;
45         rep(i,0,8){
46             cin>>s[i];
47             rep(j,0,8)
48                 if (s[i][j] != '_'){
49                     int x=s[i][j]-'0';
50                     r[i][x]=c[j][x]=1; //第i行和第j行都不能再填x
51                     dfs_str[i][j]=char(x+48);
52                     num[(i/3)*3 + j/3][x]=1; //当前格所在的小九宫格不能再填x
53                 }
54         }
```

```

54         else {
55             point[len++] = node(i, j);
56         }
57     }
58     rep(i, 0, len-1) point[i].getChoiceNum(); // 计算每个空位置可填的数字数
59     sort(point, point+len, cmp); // 按可填数字数从小到大排序
60     dfs(0);
61     if (tt > 1) cout << endl;
62     if (!ans) cout << "Puzzle " << tt << " has no solution" << endl;
63     else if (ans > 1) cout << "Puzzle " << tt << " has " << ans << " solutions" << endl;
64     else {
65         cout << "Puzzle " << tt << " solution is" << endl;
66         rep(i, 0, 8)
67             cout << ans_str[i] << endl;
68     }
69 }
70 return 0;
71 }

```

Soj 1215 脱离地牢

解题思路:在搜索中求最小步数的这类问题一般来说都是使用广度优先搜索。这题也不例外，那么关键就是如何表示搜索状态。这题很显然就是用两个人的位置以及当前已走多少步来表示一个状态，即 $(x1, y1, x2, y2, step)$ 。搜索过程就是，就是先将初始状态入队，每次循环中，队首状态出队，从该状态中扩展出 4 个状态入队（即 P 向 4 个方向走时的状态），同时判断是否有状态满足终止条件。这里的终止条件有两个，一个是，两个人移动到同一格子，另一个就是两人本来在相邻的位置，移动过程中相遇，这种情况等价于两人互换位置。很明显仅仅这样搜索会造成许多重复状态，又发现此题要求最小步数，而 BFS 第一次搜到某个状态所用步数一定是最少的，所以我们可以用一个数组 $v[x1][y1][x2][y2]$ 来表示当前状态是否搜过，若搜过则不入队。

时间复杂度: $O(n^4)$

代码:

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cstring>
4  #include<queue>
5  using namespace std;
6  #define rep(i,u,v) for (int i=(u);i<=(v);i++)
7  struct node{
8      int px,py,hx,hy,dep;
9      node(){}
10     node(int _px,int _py,int _hx,int _hy,int _dep){
11         px=_px;py=_py;hx=_hx;hy=_hy;dep=_dep;
12     }
13 };
14 const int dx[]={-1,1,0,0};
15 const int dy[]={0,0,-1,1};
16 int n,m,px,py,hx,hy;
17 int hdir[4],v[21][21][21][21];
18 char s[25][25];
19 queue<node>Q;
20 void bfs(){
21     Q.push(node(px,py,hx,hy,0));
22     v[px][py][hx][hy]=1;
23     while (!Q.empty()){
24         node n0=Q.front();

```

```

25     Q.pop();
26     rep(i,0,3){
27         int npx=n0.px+dx[i],npy=n0.py+dy[i];
28         int nhx=n0.hx+dx[hdir[i]],nhy=n0.hy+dy[hdir[i]];
29         if (s[npx][npy]!='#' && s[npx][npy]!='!' && s[nhx][nhy]!='!'){
30             if (s[nhx][nhy]=='#') nhx=n0.hx,nhy=n0.hy;
31             if (v[npx][npy][nhx][nhy]) continue;
32             if (npx==nhx&&npy==nhy){ //移动到同一格
33                 printf("%d\n",n0.dep+1);
34                 return;
35             }
36             if (npx==n0.hx&&npy==n0.hy && nhx==n0.px&&nhy==n0.py){
37                 //原本在相邻两个,移动中相遇
38                 printf("%d\n",n0.dep+1);
39                 return;
40             }
41             v[npx][npy][nhx][nhy]=1;
42             Q.push(node(npx,npy,nhx,nhy,n0.dep+1));
43         }
44     }
45 }
46 printf("Impossible\n");
47 }

48 int main(){
49     scanf("%d%d",&n,&m);
50     rep(i,1,n) scanf("%s",s[i]+1);
51     rep(i,1,n)
52         rep(j,1,m)
53             if (s[i][j]=='P') px=i,py=j;
54             else if (s[i][j]=='H') hx=i,hy=j;
55     scanf("%s",s[n+1]);
56     rep(i,0,3){
57         if (s[n+1][i]=='N') hdir[i]=0;
58         else if (s[n+1][i]=='S') hdir[i]=1;
59         else if (s[n+1][i]=='W') hdir[i]=2;
60         else if (s[n+1][i]=='E') hdir[i]=3;
61     }
62     bfs();
63     return 0;
64 }
65

```

Soj 1171 The Game of Efil

分析和解法: 数据范围是 $n*m \leq 16$, 也就意味着格子最多只有 16 个。将格子从左到右, 从上到下依次编号为 0, 1, 2, 3...15。我们可以用一个 16 位的二进制状态来表示当前状态。假设第 i 个格子有生命, 则二进制数的第 i 位为 1, 否则为 0。由于总状态数是 $2^{16}=65536$, 我们完全可以通过二进制暴力枚举完所有状态, 判断每个状态按照游戏规则经过一轮后能否变成给定的状态即可。这题关键是格子编号与格子的二维坐标之间的转换关系。它们的关系易推导出如下:

$$Id = (x*m+y)$$

$$x = Id/m.$$

$$y = Id\%m;$$

时间复杂度: $O(2^{(n*m)} * (n*m))$

代码:

```

1 #include<iostream>
2 #include<cstdio>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)

```

```

5 const int dx[]={-1,-1,-1,0,0,1,1,1};
6 const int dy[]={-1,0,1,-1,1,-1,0,1};
7 int ans,n,m,sta;
8 int play(int w,int now,int ty){
9     int cnt=0,xx=w/m,yy=w%m;
10    rep(i,0,7){
11        int x=xx+dx[i],y=yy+dy[i];
12        if (x<0)x=n-1;
13        else if (x==n)x=0;
14        if (y<0)y=m-1;
15        else if (y==m)y=0;
16        if ( ( 1<<(x*m+y) ) & now ) cnt++;
17    }
18    if (!ty) return !(cnt==2||cnt==3);
19    else return cnt==3;
20 }
21 int check(int now){
22     int next=now;
23     rep(i,0,n*m-1){
24         if ( ( 1<<i) & now ){
25             if (play(i,now,0)) next^=(1<<i); //有0, 1, 4, 5, 6, 7, 8个邻居则死亡
26             else next=next; //否则继续生存
27         }
28         else if (play(i,now,1)) next^=(1<<i); //新生命诞生
29     }
30     return next==sta;
31 }

32 int main(){
33     int k,x,y,cases=0;
34     while (scanf("%d%d",&n,&m)!=EOF){
35         if (!n && !m) break;
36         sta=ans=0;
37         scanf("%d",&k);
38         rep(tt,1,k){
39             scanf("%d%d",&x,&y);
40             sta |= ( 1<<(x*m+y) );
41         }
42         int tot_sta = ( 1<<(n*m) ) -1;
43         rep(i,0,tot_sta)
44             if (check(i)) ans++;
45         if (!ans) printf("Case %d: Garden of Eden.\n",++cases);
46         else printf("Case %d: %d possible ancestors.\n",++cases,ans);
47     }
48     return 0;
49 }

```

Soj 1014 Specialized Four-Dig

解题思路: 由于 4 位数只有 9000 个, 所以我们直接暴力枚举每一个 4 位数, 判断其是否满足题目要求即可。求数字 n 在 $base$ 进制下各位数字和的方法: 我们知道将一个 10 进制的数字转换成其他数字是通过不断除以 $base$ 取其余数排起来。所以求各位数字和即求所有余数的和就是 n 不断 $\%base$, 结果相加直到 $n=0$ 。

时间复杂度: $O(9000)$

代码:

```

1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 int check(int n,int base){
5     int sum=0;
6     while(n){
7         sum+=n%base;
8         n/=base;
9     }
10    return sum==4;
11 }
12 int main(){
13     int n;
14     while (scanf("%d",&n)!=EOF){
15         if (!n) break;
16         int ans=0;
17         rep(i,1,n){
18             if (check(i,10)) ans++;
19         }
20         printf("%d\n",ans);
21     }
22     return 0;
23 }

```



```

9     }
10    return sum;
11 }
12 int main() {
13     rep(i, 1000, 9999) {
14         int t = check(i, 10);
15         if (t == check(i, 12) && t == check(i, 16))
16             cout << i << endl;
17     }
18     return 0;
19 }

```

Soj 1709 PropBot

分析和解法:时间 t 最多只有 24，每个单位时间机器人有两种选择，也就是说一共最多有 2^{24} 次选择，直接暴力搜索出每次选择后机器人的位置，更新其与目的地的最短距离即可。搜索的时候我使用了一点小技巧，用 dir 表示机器人方向与 x 轴正方向的夹角，则每一横纵坐标各自增加 $\cos(dir)$ 和 $\sin(dir)$ 。 dir 初始为 0，每次改变方向就减 $\pi/4$ 既 $\arccos(-1)/4$ 。但是直接实现的时候却发现会超时。于是我加入了一句剪枝，假设剩下的时间都是直线奔向目标点，最后得到的距离还是比现在已记录的最优值要大，则不继续搜下去。其实会超时的原因应是三角函数的效率差，也就是我牺牲了时间去减少代码量。

时间复杂度 $O(2^t)$

代码:

```

1 #include<stdio.h>
2 #include<math.h>
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 const double delta=acos(-1.0)/4.0;
5 inline double dis(double x,double y,double xx,double yy){
6     return sqrt( (x-xx)*(x-xx) + (y-yy)*(y-yy) );
7 }
8 double edx,edy,ans;
9 int limit_t;
10 inline void dfs(double x,double y,double dir,double t){
11     double dist=dis(x,y,edx,edy);
12     if (ans>dist)ans=dist;
13     if (t==limit_t || ans<dist-10*(limit_t-t))return;
14     dfs(x,y,dir-delta,t+1);
15     x=x+cos(dir)*10.0;
16     y=y+sin(dir)*10.0;
17     dfs(x,y,dir,t+1);
18 }
19 int main(){
20     int T,t;
21     scanf("%d",&T);
22     rep(tt,1,T){
23         scanf("%d%lf%lf",&limit_t,&edx,&edy);
24         ans=dis(0,0,edx,edy);
25         dfs(0,0,0,0);
26         printf("%.6lf\n",ans);
27     }
28     return 0;
29 }

```

Soj 1108 Online Selection

分析和解法:每个状态中，只有回答 0 和回答 1 两种情况，所以我们只要从

初始状态开始，每个状态按照两个方向去搜索，最后取得一个最大值。不过由于 n 和 k 分别达到 20 和 30， 2^{30} 肯定超时了，所以直接搜索不可行，也就是我们需要剪枝。我们可以使用 $f[sta][times][score]$ 表示当前在状态 i ，已回答了 $times$ 次，得分 $score$ 时的最多回答了多少次 0。假设后面又搜到了这个状态，但是回答 0 的次数变少了则不继续搜下去。这就是记忆化搜索。另外我还加多了一句剪枝。假设后面剩余的次数全部都回答 0，这样还不能比已记录的最优值更优的话则不继续搜下去。

代码：

```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
5 int a[21][3],f[21][31][31];
6 int n,m,k,ans,T;
7 inline void dfs(int sta,int times,int score,int cnt){
8     if (ans>=k-times+cnt) return; //当前状态往后不可能再比当前最优值更优
9     if (f[sta][times][score]>=cnt) return; //之前搜到这个状态时比现在更优
10    f[sta][times][score]=cnt;
11    if (times==k){
12        if (score==m&&ans<cnt) ans=cnt;
13        return;
14    }
15    dfs(a[sta][0],times+1,score+(a[sta][2]==0),cnt+1);
16    dfs(a[sta][1],times+1,score+(a[sta][2]==1),cnt);
17 }
18 int main(){
19     while (scanf("%d",&T)!=EOF){
20         if (!T) break;
21         scanf("%d%d%d",&n,&k,&m);
22         rep(i,0,n-1) scanf("%d%d%d",&a[i][0],&a[i][1],&a[i][2]);
23         ans=0;
24         memset(f,-1,sizeof(f));
25         dfs(0,0,0,0);
26         printf("%d %d\n",T,ans);
27     }
28     return 0;
29 }
```

Soj 6276 Pawns

分析和解法：这题是求最小步数，也就是应该使用 bfs 求解。我将棋盘的状态用一个三进制数来表示，第 i 位为 0、1、2 就分别表示第 i 个格子是空的、放白子和放黑子三种状态，然后用一个 string 来存。使用广度优先搜索，初始状态入队，暴力枚举移动某一颗棋子，判断移动之后的状态之前有没有搜索过，若有则 continue，若无，判断其是否到达目标状态，若没有到达则入队，继续 bfs。

代码：

```
1 #include<iostream>
2 #include<cstdio>
3 #include<queue>
4 #include<map>
5 using namespace std;
6 #define rep(i,u,v) for(int i=(u);i<=(v);i++)
7 struct node{
8     string sta;
9     int step;
10    node(){}
11    node(string ss,int d){
12        sta=ss;step=d;
13    }
14 };
```



```

15 int n,wnum,bnum,a[15];
16 string orista="";
17 queue<node>Q;
18 map<string,int>f;
19 int check(string sta){
20     int l=0,r=n-1,cnt1=0,cnt2=0;
21     while (l<n&&sta[l]=='1') l++,cnt1++;
22     if (cnt1!=wnum) return 0;
23     while (r>=0&&sta[r]=='2') r--,cnt2++;
24     if (cnt2!=bnum) return 0;
25     return 1;
26 }
27 void bfs(){
28     if (check(orista)){
29         printf("0\n");
30         return;
31     }
32     Q.push(node(orista,0));
33     f[orista]=1;
34     while (!Q.empty()){
35         node n0=Q.front();Q.pop();
36         int l=0,r=n-1;
37         while (l<n&&n0.sta[l]=='1') l++;
38         while (r>=0&&n0.sta[r]=='2') r--;
39         rep(i,l,r){
40             string nextsta=n0.sta;
41             if (n0.sta[i]=='1'){//白子
42                 if (i-1>=0&&n0.sta[i-1]=='0'){//可以向左一步
43                     nextsta[i-1]='1';
44                     nextsta[i]='0';
45                 }
46                 else if (i-2>=0&&n0.sta[i-2]=='0'){//可以跳一步
47                     nextsta[i-2]='1';
48                     nextsta[i]='0';
49                 }
50                 else continue;
51             }
52             else if (n0.sta[i]=='2'){
53                 if (i+1<n&&n0.sta[i+1]=='0'){//可以向右一步
54                     nextsta[i+1]='2';
55                     nextsta[i]='0';
56                 }
57                 else if (i+2<n&&n0.sta[i+2]=='0'){//可以跳一步
58                     nextsta[i+2]='2';
59                     nextsta[i]='0';
60                 }
61                 else continue;
62             }
63             int &ret=f[nextsta];
64             if (ret!=0) continue;
65             ret++;
66             if (check(nextsta)){
67                 printf("%d\n",n0.step+1);
68                 return;
69             }
70             Q.push(node(nextsta,n0.step+1));
71         }
72     }
73 }
74 int main(){
75     scanf("%d",&n);
76     rep(i,1,n){
77         scanf("%d",&a[i]);
78         if (a[i]==1) wnum++;
79         else if (a[i]==2) bnum++;
80         char x=char(a[i]+48);
81         orista+=x;
82     }
83     bfs();
84     return 0;
85 }

```

Soj 1471 No Left Turns

分析和解法: 本题就是一道比较简单的 bfs。将当前位置，已走步数，方向作为一个搜索状态。每次只能向前走一格或者向当前方向的右边走一格。由于是 bfs，每个位置第一次被搜到的时候所用步数是最少的。我们用 $v[x][y][dir]$ 对一个位于 (x, y) 位置，方向为 dir 的状态进行判重，可以节省大量时间和空间。

时间复杂度 $O(n*m*4)$

代码:

```
1 #include<iostream>
2 #include<cstdio>
3 #include<queue>
4 #include<cstring>
5 using namespace std;
6 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
7 const int dx[]={-1,0,1,0,-1};
8 const int dy[]={0,1,0,-1,0};
9 int n,m,edx,edy,ctx,cty;
10 int v[22][22][4];
11 char s[22][22];
12 struct node{
13     int x,y,step,dir;
14     //dir为方向, 0, 1, 2, 3分别表示上右下左
15     node(){}
16     node(int xx,int yy,int s,int d){
17         x=xx;y=yy;step=s;dir=d;
18     }
19 };
20 queue<node>Q;
21 void bfs(){
22     while (!Q.empty())Q.pop();
23     rep(i,0,3){
24         Q.push(node(ctx,cty,0,i));
25         v[ctx][cty][i]=1;
26     }
27     while (!Q.empty()){
28         node n0=Q.front();Q.pop();
29         rep(i,n0.dir,n0.dir+1){
30             int x=n0.x+dx[i],y=n0.y+dy[i];
31             if (x<=0||x>n||y<=0||y>m) continue;
32             if (s[x][y]=='X') continue;
33             if (v[x][y][i%4]) continue;
34             if (s[x][y]=='F'){
35                 printf("%d\n",n0.step+1);
36                 return;
37             }
38             if (s[x][y]=='F'){
39                 printf("%d\n",n0.step+1);
40                 return;
41             }
42             v[x][y][i%4]=1;
43             Q.push(node(x,y,n0.step+1,i%4));
44         }
45     }
46 }
47 int main(){
48     int T;
49     scanf("%d",&T);
50     rep(tt,1,T){
51         scanf("%d%d",&n,&m);
52         getchar();
53         rep(i,1,n)gets(s[i]+1);
54     }
55 }
```

```

50     memset(v,0,sizeof(v));
51     rep(i,1,n)
52         rep(j,1,m)
53             if (s[i][j]=='S') stx=i, sty=j;
54             else if (s[i][j]=='F') edx=i, edy=j;
55     bfs();
56 }
57 return 0;
58 }

```

Soj 1182 Context-Free Clock

分析和解法：由于 24 小时里按秒为最小单位的时间里一共只有 $60 \times 60 \times 24 = 86400$ 个时间。所以我们先预处理出这 86400 个时间里时针按顺时针到分针的角度。然后从当前时间开始，顺时针找到一个角度最接近 θ 的时间输出即可。这里说一下预处理里怎么做。假设当前在时间 $i:j:k$ 。先算出从数字 12 顺时针到时针之间的角度：

$$\text{Degree1} = (i \% 12) * 30 + (60.0 * j + k) / 3600 * 30;$$

因为每一个小时，时针转过 30 度，所以其实这条式子就是先算已经转过多少个 30 度，再加上不足 30 度那一部分。不足 30 度那部分可以按秒去算。然后同理可算出数字 12 顺时针到分针之间的角度：

$$\text{Degree2} = j * 6 + 1.0 * k / 60 * 6;$$

然后算时针和分针的角度要分时针在前还是分针在前两种情况。这里的前后关系是按照 Degree1 和 Degree2 的大小来看的，因为两者取值都是 0-360。假如分针在前，即 $\text{Degree2} > \text{Degree1}$ 则两者夹角为：

$$\text{Degree2} - \text{Degree1}$$

反之则两者夹角为：

$$\text{Degree2} - \text{Degree1} + 360$$

这题还有一个很恶心的地方就是精度问题，首先 abs 函数要自己写，不能用标准库的，其次误差精度要取 0.08，再小一点都会错。这是我 WA 了 n 次摸出来的，充满了血与泪。

时间复杂度 $O(24 \times 60 \times 60)$

代码：

```

1 #include<stdio.h>
2 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
3 double degree[86401];
4 double abs(double x){return x<0?-x:x;}
5 void prepare() {
6     double hourdegree,minutedegree,d;
7     rep(i,0,23){
8         rep(j,0,59){
9             rep(k,0,59){
10                 hourdegree=(i % 12) * 30 + (60.0 * j + k) / 120;
11                 minutedegree=j * 6 + 1.0 * k / 10;
12                 d=hourdegree>minutedegree ? 360-hourdegree+minutedegree : minutedegree-hourdegree;
13                 degree[i * 3600 + j * 60 + k] = d;
14             }
15         }
16     }
17 }
18 const double eps = 0.08;
19 bool check(double d1, double d2) {
20     if (d1 == 0) {
21         return abs(d1-d2)<eps || abs(360.0-d2)<eps;
22     }
23     return abs(d1-d2)<eps;
24 }

```

```

25 int main(){
26     int h, m, s;
27     double A;
28     prepare();
29     while (scanf("%lf %d:%d:%d", &A, &h, &m, &s)!=EOF){
30         if (A== -1)break;
31         int st=h*3600+m*60+s;
32         rep(i,st,86399){
33             if (check(A,degree[i])){
34                 printf("%02d:%02d:%02d\n",i/3600,i%3600/60,i%3600%60);
35                 break;
36             }
37             if (i==86399)i=-1;
38         }
39     }
40     return 0;
41 }

```

Soj 1710 Painted Calculator

分析和解法:这题比较简单了，前两个数字取值都是-999 到 999，那么我们直接枚举出前两个数字的所有组合，每个组合做 4 种运算，看每一次的功耗分别是多少，做好统计，用 $w[x][y][c]$ 存储， $w[x][y][c]$ 就表示第一个数字功耗为 x ，第二个数字功耗为 y ，结果的功耗为 c 这样的组合的情况有多少种。所以最后我们就可以直接输出答案了，要注意当答案为 1 是，输出的 solution 末尾没有 s。

时间复杂度 $O(1000^2)$

代码:

```

1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v) for (int i=(u);i<=(v);i++)
4 int cost[10] = {6, 2, 5, 5, 4, 5, 6, 3, 7, 6};
5 int w[36][36][36];
6 int calc(int x) {
7     if (x==0) return 6;
8     int ans= 0;
9     if (x<0)ans++,x=-x;
10    while (x) {
11        ans+=cost[x%10];
12        x/=10;
13    }
14    return ans;
15 }
16 void init() {
17     int x,y,z;
18     rep(i,-999,999){
19         x=calc(i);
20         rep(j,-999,999){
21             y=calc(j);
22             z=i+j;
23             if (-999<=z&&z<=999)w[x][y][calc(z)]++;
24             z=i-j;
25             if (-999<=z&&z<=999)w[x][y][calc(z)]++;
26             z=i*j;
27             if (-999<=z&&z<=999)w[x][y][calc(z)]++;
28             if (!j)continue;
29             z=i/j;

```

```

30         if (-999<=z&&z<=999)w[x][y][calc(z)]++;
31     }
32 }
33 }
34 int main(){
35     std::ios::sync_with_stdio(false);
36     init();
37     int x, y, z;
38     while (cin>>x&&x) {
39         cin>>y>>z;
40         if (w[x/5][y/5][z/5]==1)cout<<w[x/5][y/5][z/5]<<" solution for "<<x<<" "<<y<<" "<<z<<endl;
41         else cout<<w[x/5][y/5][z/5]<<" solutions for "<<x<<" "<<y<<" "<<z<<endl;
42     }
43 }

```

Soj 7144 Different Triangles

分析和解法: n 只有 100, 直接 n^3 暴力枚举三根棍子判断它们能否组成三角形即可。判断的方法就是三角形任意两边之和大于第三边; 任意两边之差小于第三边。

时间复杂度 $O(n^3)$

代码:

```








1 #include<iostream>
2 using namespace std;
3 #define rep(i,u,v)for (int i=(u);i<=(v);i++)
4 int a[101];
5 bool istriangle(int a, int b, int c){
6     if(a+b>c&&a+c>b&&b+c>a&&a-b<c&&a-c<b&&b-c<a) return true;
7     return false;
8 }
9 int main() {
10     int T,n;
11     cin>>T;
12     while(T--){
13         int ans=0;
14         cin>>n;
15         rep(i,1,n)cin>>a[i];
16         rep(i,3,n)
17             rep(j,2,i-1)
18                 rep(k,1,j-1)
19                     if(istriangle(a[i],a[j],a[k]))ans++;
20         cout<<ans<<endl;
21     }
22 }

```

4.程序运行与测试

14 道题均通过成功运行并通过 sicily 的测试。

						Time	Memory	Length	Time
5109679		smie15352408	1710	C++	Accepted	0.21sec	484 KB	1134 Bytes	2017-05-09 16:44:14
5109652		smie15352408	1182	C++	Accepted	0sec	960 KB	1163 Bytes	2017-05-09 16:06:09
5109628		smie15352408	1471	C++	Accepted	0sec	316 KB	1392 Bytes	2017-05-09 15:14:05

5109572		smie15352408	6276	C++	Accepted	0.37sec	3872 KB	2166 Bytes	2017-05-09 13:19:03
5109533		smie15352408	1317	C++	Accepted	0.29sec	312 KB	2106 Bytes	2017-05-09 11:49:11
5109439		smie15352408	1108	C++	Accepted	0.36sec	388 KB	789 Bytes	2017-05-08 23:30:43
5109437		smie15352408	1108	C++	Accepted	0.21sec	388 KB	787 Bytes	2017-05-08 23:30:30
5109377		smie15352408	1709	C++	Accepted	4.14sec	288 KB	743 Bytes	2017-05-08 22:32:21
5109374		smie15352408	1709	C++	Accepted	0.75sec	308 KB	1189 Bytes	2017-05-08 22:26:29
5109371		smie15352408	1709	C++	Accepted	4.2sec	308 KB	781 Bytes	2017-05-08 22:25:14
5109345		smie15352408	1171	C++	Accepted	1.11sec	308 KB	1345 Bytes	2017-05-08 21:27:32
5109181		smie15352408	1215	C++	Accepted	0sec	1068 KB	1851 Bytes	2017-05-08 00:36:25
5106308		smie15352408	7144	C++	Accepted	0.02sec	348 KB	500 Bytes	2017-05-04 15:31:42
5106149		smie15352408	1014	C++	Accepted	0sec	304 KB	369 Bytes	2017-05-03 21:46:24
5106147		smie15352408	1515	C++	Accepted	0.22sec	2152 KB	2241 Bytes	2017-05-03 21:00:47
5106139		smie15352408	1151	C++	Accepted	0.44sec	2280 KB	2098 Bytes	2017-05-03 20:42:51
5106138		smie15352408	1150	C++	Accepted	0sec	308 KB	2098 Bytes	2017-05-03 20:42:44

5.实验总结与心得

这次搜索专题的练习算是对以前所学知识的一次巩固。搜索的本质就是试探。从某个初始状态开始不断试探直到找到目标状态。搜索一般可以解决如下几种问题：

最小步数问题，这种一般使用 bfs

统计解的个数，根据情况选择 dfs 或 bfs

寻找最优解或一个满足条件的解

图的相关问题。

一般来说当我们对一道题没有什么好想法的时候就可以使用搜索。搜索中比较关键的是状态的表示，很多时候可以用一个几进制状态去表示。然后就要注意适当地剪枝，缩小搜索时间。剪枝的方法有上下界剪枝、状态判重、最优解比较剪枝等。

附录、提交文件清单

实验报告.pdf

1150and1151.cpp

1515. cpp
1317. cpp
1215. cpp
1171. cpp
1014. cpp
1709. cpp
1108. cpp
6276. cpp
1471. cpp
1182. cpp
1710. cpp
7144. cpp