



浙江财经大学

Zhejiang University Of Finance & Economics



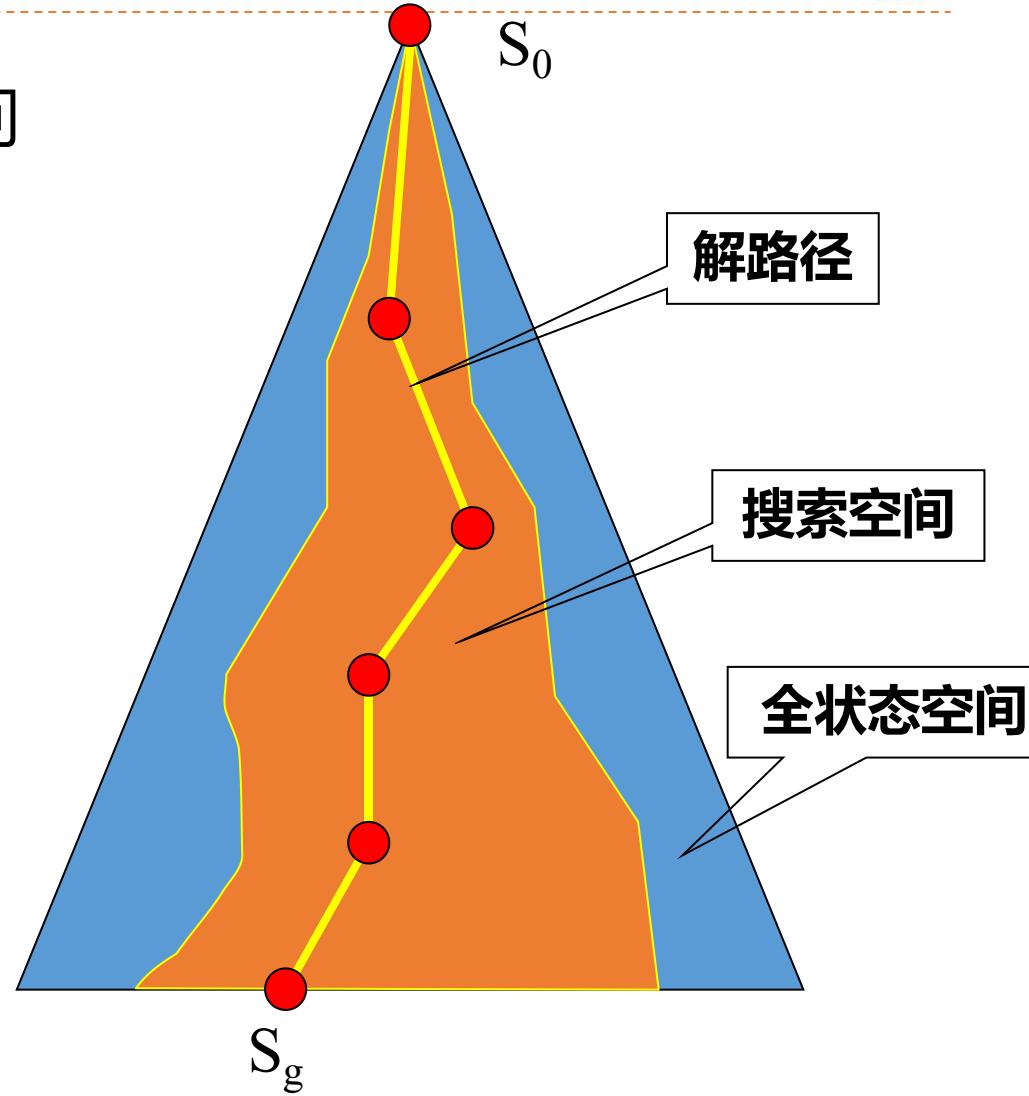
DFS实践

信智学院 陈琰宏



3.1.1 搜索算法

由此，可以看出这类问题的解，就是一个合法状态的序列，其中序列中第一个状态是问题的初始状态，而最后一个状态则是问题的结束状态。如图所示即搜索问题的示意图：





3.1.2 搜索算法常见分类

- 枚举算法
- 深度优先搜索 (DFS) 与回溯
- 广度优先搜索 (BFS)
- 双向广度优先搜索
- A*算法
- 群智能搜索算法





3.1.3 回溯搜索概念

回溯法也称试探法。

它的基本思想是：从问题的某一种状态（初始状态）出发，搜索从这种状态出发所能达到的所有“状态”，当一条路走到“尽头”的时候（不能再前进），再后退一步或若干步，从另一种可能“状态”出发，继续搜索，直到所有的“路径”（状态）都试探过。

这种不断“前进”、不断“回溯”寻找解的方法，就称作“回溯法”。



搜索框架

递归回溯：由于回溯法是对解空间的深度优先搜索，因此在一般情况下可用递归函数来实现回溯法如下：

递归回溯法算法框架[一]

```
int Search(int k)
```

```
{
```

```
for (i=1;i<=算符种类数;i++)
```

```
if (满足条件)
```

```
{
```

```
    保存结果
```

```
    if (到目的地) 输出解;
```

```
        else Search(k+1);
```

```
    恢复：保存结果之前的状态{回溯一步}
```

```
}
```

```
}
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| 1 | ■ | | | ■ | | | ■ | | ■ | ■ |
| 2 | ■ | | | ■ | | | | ■ | | ■ |
| 3 | ■ | | | | ■ | | ■ | | ■ | |
| 4 | ■ | | | | ■ | ■ | | | | |
| 5 | ■ | | | | | ■ | | | | |
| 6 | | | ■ | | | | | ■ | | |
| 7 | | | | ■ | ■ | ■ | | | ■ | |
| 8 | | | | | ■ | | | | | |
| 9 | | | | | | ■ | | | | |

搜索框架

递归回溯法算法框架[二]

```
int Search(int k)
```

```
{
```

```
    if (到目的地) 输出解;
```

```
    else
```

```
        for (i=1;i<=算符种类数;i++)  
            if (满足条件)
```

```
{
```

```
    保存结果;
```

```
    Search(k+1);
```

```
    恢复：保存结果之前的状态{回溯一步}
```

```
}
```

```
}
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|----|----|----|----|----|
| 0 | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| 1 | ■ | ■ | 1 | 2 | 3 | 4 | 5 |
| 2 | ■ | ■ | 6 | 7 | ■ | ■ | ■ |
| 3 | ■ | ■ | 8 | 9 | 10 | 11 | ■ |
| 4 | ■ | ■ | 12 | ■ | ■ | 13 | 14 |
| 5 | ■ | ■ | 15 | 16 | 17 | ■ | 18 |
| 6 | ■ | ■ | ■ | ■ | ■ | ■ | ■ |



1[3302]素数环

素数环:从1到n这n个数摆成一个环，要求相邻的两个数的和是一个素数。
如，n=8是，素数环为：

1 2 3 8 5 6 7 4

1 2 5 8 3 4 7 6

1 4 7 6 5 8 3 2

1 6 7 4 3 8 5 2

总数为4

输入

输入n的值 (n不大于15)

输出

样例输入

8

样例输出

4



算法分析

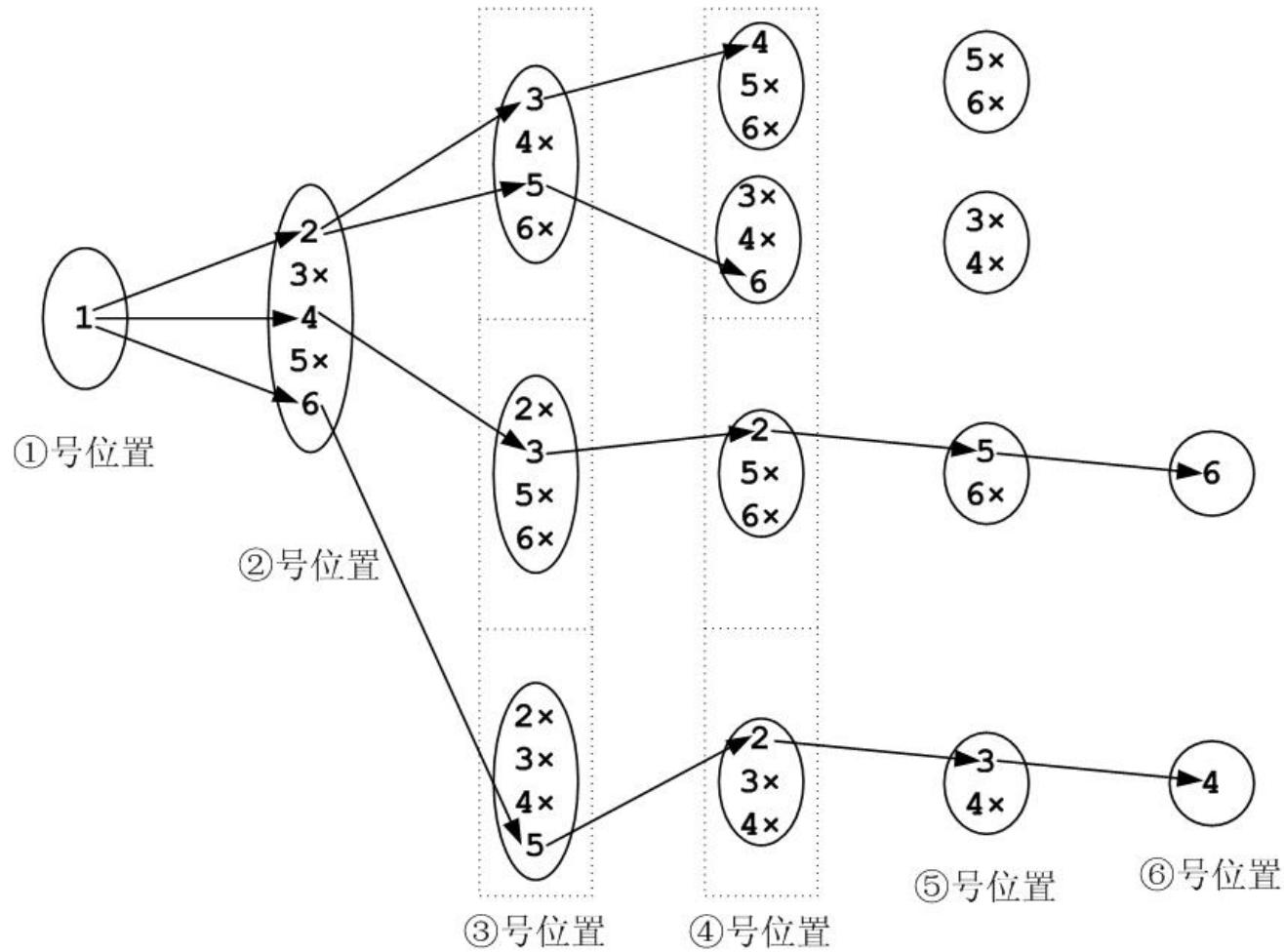
非常明显，这是一道回溯搜索的题目。从1开始，每个空位有20种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第20个数还要判断和第1个数的和是否素数。

【算法流程】

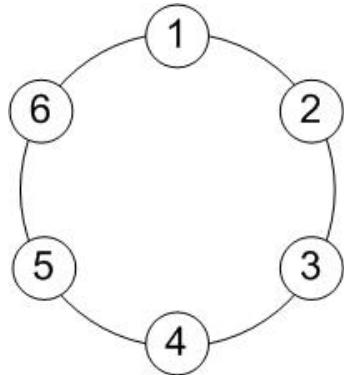
- 1、数据初始化；
- 2、递归填数：判断第*i*个数填入是否合法；
 - A、如果合法：填数；判断是否到达目标（20个已填完）：是，打印结果；不是，递归填下一个；
 - B、如果不合法：选择下一种可能；



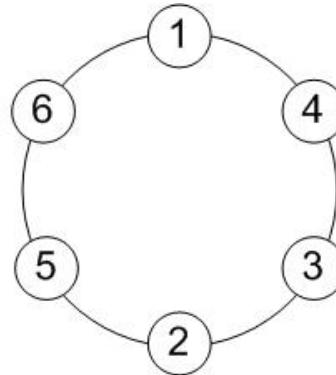
搜索策略 (n=6)



搜索策略 ($n=6$)



(a) 环上的 n 个位置



(b) 在 n 个位置放置自然数 $1 \sim n$

在①号位置上放置数1。在②号位置上可供选择的数为2~6，其中3和5是不可行的，对2、4和6一一试探，先试探2。

②号位置上放置2以后，③号位置上可供放置的数为3~6，其中4和6是不可行的，对3和5也是一一试探，先试探3。

搜索策略 (n=6)

③号位置上放置3以后，④号位置上可供放置的数为4~6，其中只有4是可行的，所以放置4。这时可供⑤号位置上放置的数为5和6，均不可行。这些方案都不可行。

再考虑③号位置上放置5，④号位置上只能放置6，此后⑤号位置上放置3和4，均不可行。这些方案都排除。

再考虑②号位置上放置4，③号位置上只能放置3，④号位置上只能放置2，⑤号位置上只能放置5，⑥号位置只能放置6，这个方案是可行的。

...

如此搜索，直到试探所有方案为止。



代码

```
1 //3302 素数环  
2  
3 #include <iostream>  
4 #include<iomanip>  
5 #include<cmath>  
6 using namespace std;  
7 int a[100]={0},b[100]={0};  
8 //数组a用来存储环中的每个元素 b表示当前的值i有没有没使用过  
9 int n,sum=0;  
10 int prime(int a,int b){  
11     int c=a+b;  
12     for(int i=2;i<=sqrt(c);i++ )  
13         if(c%i==0) return 0;  
14     return 1;  
15 }
```



代码

```
18 int search(int k)//k表示处理的位置
19 {
20     if(k==n+1&&prime(a[1],a[n]))sum++;// if (到目的地) 输出解;
21     else{
22         for(int i=2;i<=n;i++){//for (i=1;i<=字符种类数;i++)
23             if(prime(i,a[k-1])&&b[i]==0){//if (满足条件)
24                 a[k]=i,b[i]=1;//保存结果
25                 search(k+1);
26                 b[i]=0;//恢复
27             }
28         }
29     }
30 }
31
32 int main()
33 {
34     //memset(a,0,sizeof(a));
35     //memset(b,0,sizeof(b));
36     cin>>n;
37     a[1]=1,b[1]=1;
38     search(2);
39     cout<<sum<<endl;
40     return 0;
41 }
```



框架一

```
33 int search(int k)//k表示处理的位置
34 {
35     if(____1_____)sum++; // if (到目的地) 输出解;
36     else{
37         for(int i=2;i<=n;i++){//for (i=1;i<=算符种类数;i++)
38             if(prime(i,a[k-1])&&____2_____{//if (满足条件)
39                 ____3_____; //保存结果
40                 search(k+1);
41                 ____4_____; //恢复
42             }
43         }
44     }
45 }
```



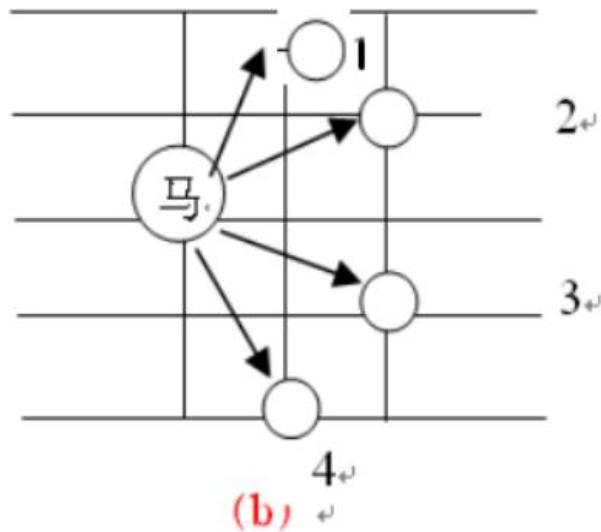
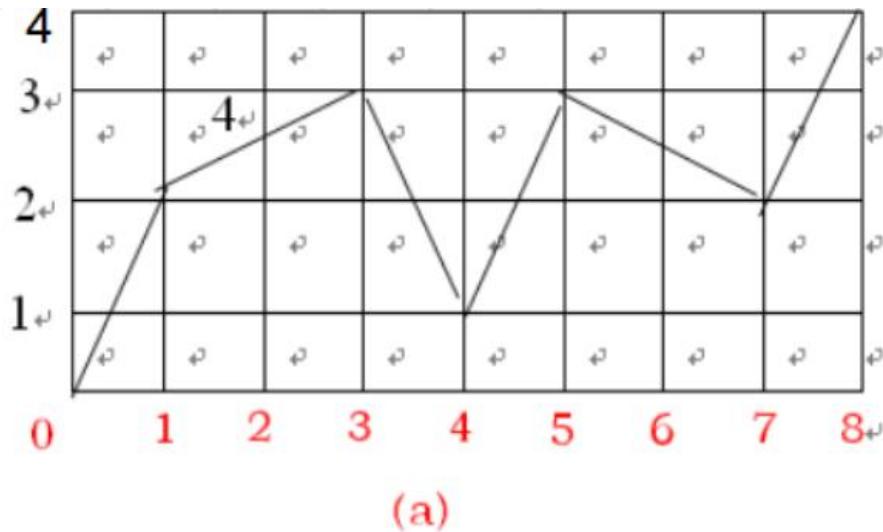
框架二

```
19 int search(int k)//k表示处理的位置
20 {
21     for(int i=2;i<=n;i++){//for (i=1;i<=算符种类数;i++)
22         if(prime(i,a[k-1])&&b[i]==0){//if (满足条件)
23             a[k]=i,b[i]=1;//保存结果
24             if(k==n){
25                 if(prime(a[n],a[1]))sum++; //if (到目的地) 输出解;
26             }
27             else
28                 search(k+1); //else Search(k+1);
29             b[i]=0; //恢复
30         }
31     }
32 }
33 }
```



2 [3306] 马的遍历

中国象棋半张棋盘如图4（a）所示。马自左下角往右上角跳。今规定只许往右跳，不许往左跳。比如图4（a）中所示为一种跳行路线，路线为：0,0->2,1->3,3->1,4->3,5->2,7->4,8...



分析

```
7 void dfs(int k){  
8     if(到达目的地)做相应操作;  
9     else{  
10        for(int i=0;i<选择的种数;i++){  
11            if(当前选择满足条件){  
12                保存结果;  
13                dfs(k+1);继续搜索;  
14                恢复: 恢复到保存结果之前的状态;  
15            }  
16        }  
17    }  
18}  
19 }
```

2. 满足条件

```
24 int isOk(int x, int y ){  
25     if(x<=8&&x>=0&&y<=4&&y>=0)  
26         return 1;  
27     else return 0;  
28 }
```

1. 初始状态与目的地

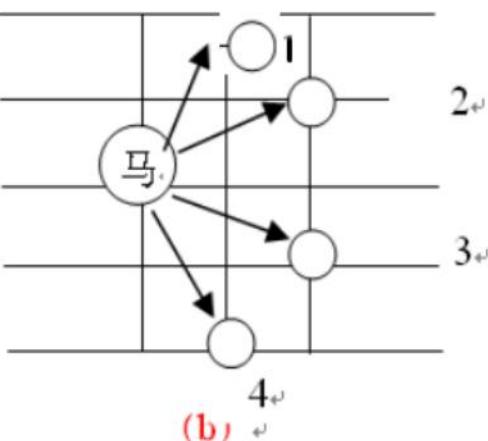
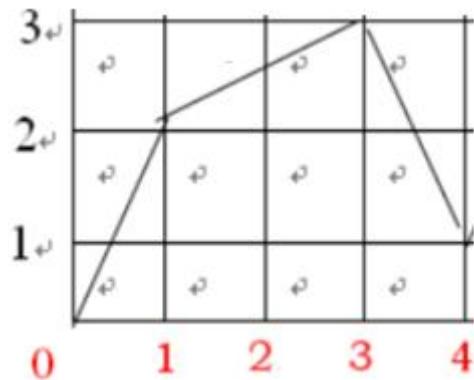
```
dfs(0,0);  
30 if(x==8&&y==4){  
31     sum++;  
32 }
```

3. 保存结果与回溯

```
35 for(int i=0;i<4;i++){  
36     x=x+dir[i][0];  
37     y=y+dir[i][1];  
38     if(isOk(x,y)){  
39         dfs(x,y);  
40     }  
41     x=x-dir[i][0];  
42     y=y-dir[i][1];  
43 }
```



过程



```
0 0  
2 1  
4 0  
4 2  
3 3  
4 1  
1 2  
2 0  
4 1  
3 2  
4 0  
3 1  
4 3  
3 3  
4 1  
1  
5   int sum=0;  
6   int dir[4][2]={{1,-2},{2,-1},{2,1},{1,2}};  
7   void dfs(int x,int y){  
8       cout<<x<<" "<<y<<endl;  
9       if(x==4&&y==3){  
10           sum++;  
11           cout<<"以上是第"<<sum<<"种走法的步骤"<<endl;  
12           return;  
13       }  
14       else{  
15           for(int i=0;i<4;i++){  
16               int xx=x+dir[i][0];  
17               int yy=y+dir[i][1];  
18               if(xx>=0&&xx<=4&&yy>=0&&yy<=3)  
19                   dfs(xx,yy);  
20           }  
21       }  
22   }
```

以上是第1种走法的步骤



代码

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int sum=0;
4 int dir[4][2]={{1,2},{2,1},{2,-1},{1,-2}};
5 int isOk(int x, int y){
6     if(x<=8&&x>=0&&y<=4&&y>=0)
7         return 1;
8     else return 0;
9 }
10 void dfs(int x,int y){
11     if(x==8&&y==4){
12         sum++;
13         return;
14     }
15     for(int i=0;i<4;i++){
16         x=x+dir[i][0],y=y+dir[i][1];
17         if(isOk(x,y)){
18             dfs(x,y);
19         }
20         x=x-dir[i][0],y=y-dir[i][1];
21     }
22 }
23 int main(){
24     dfs(0,0);
25     cout<<sum<<endl;
26     return 0;
27 }
```

```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 int sum=0;
5 int dir[4][2]={{1,-2},{2,-1},{2,1},{1,2}};
6 int vis[9][5];
7 int isOk(int x,int y){//条件判断
8     if(x<=8&&y<=4&&x>=0&&y>=0&&vis[x][y]==0)
9         return 1;
10    else
11        return 0;
12 }
13 void search(int x, int y){
14     if(x==8&&y==4)sum++;//if (到目的地) 输出解;
15     for(int i=0;i<4;i++){
16         x=x+dir[i][0],y=y+dir[i][1];
17         if(isOk(x,y)){
18             vis[x][y]=1;//保存结果
19             search(x,y);
20             vis[x][y]=0;//恢复
21         }
22         x=x-dir[i][0],y=y-dir[i][1];
23     }
24 }
25 int main(){
26     memset(vis,0,sizeof(vis));
27     search(0,0);
28     cout<<sum<<endl;
29     return 0;
30 }
```



3 [3299] n皇后问题

会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将n个皇后放在棋盘上（有 $n \times n$ 个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。

输入n输出，输出放置的具体方法，以及总共的方法数。

样例输入

4

样例输出

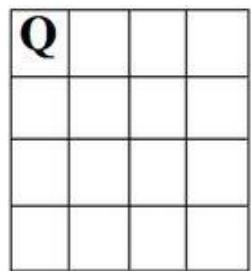
2 4 1 3

3 1 4 2

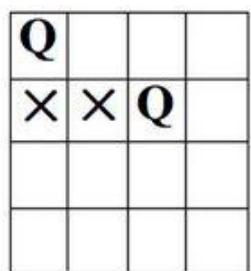
2



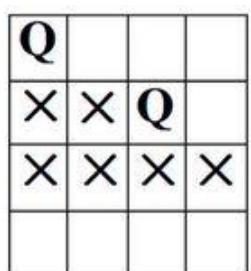
搜索策略 (n=4)



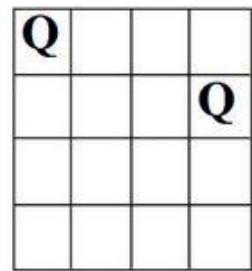
(a)



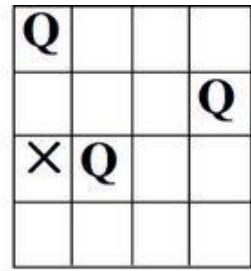
(b)



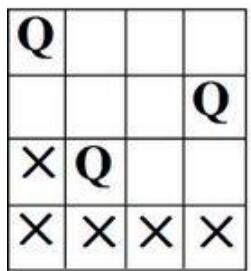
(c)



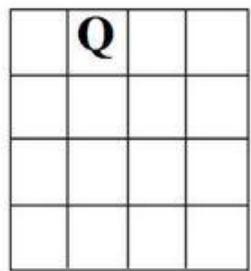
(d)



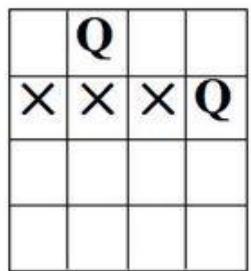
(e)



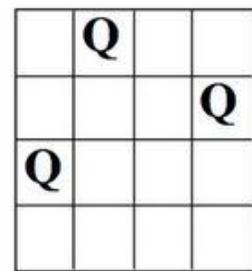
(f)



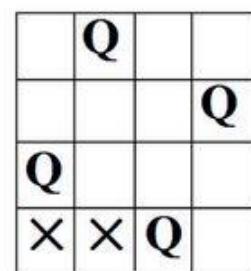
(g)



(h)



(i)



(j)



核心点分析

问题的关键在于如何判定某个皇后所在的行、列、斜线上是否有别的皇后；可以从矩阵的特点上找到规律。

如果在同一行，则行号相同；

如果在同一列上，则列号相同；

如果同在 / 斜线上的行列值之和相同；

如果同在 \ 斜线上的行列值之差相同；

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | / |
| 2 | \ | | | | | | | / |
| 3 | | \ | | | | / | | |
| 4 | | | \ | | / | | | |
| 5 | - | - | - | ▲ | - | - | - | - |
| 6 | | / | | | \ | | | |
| 7 | / | | | | | \ | | |
| 8 | / | | | | | | \ | |



核心点分析

如果在同一行，则行号相同；
如果在同一列上，则列号相同；

array[i] == array[k]

如果同在 / 斜线上的行列值之和相同；
如果同在 \ 斜线上的行列值之差相同；

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| / | | | | | | | |
| \ | | | | | | | |
| \ | | | | | | | |
| | \ | | | | | | |
| - | - | - | ▲ | - | - | - | - |
| / | | | | | | | |
| / | | | | | | | |

abs(array[i] - array[k]) == i - k

```
12  bool isOk(int i, int array[]) { // 检查当前棋盘布局是否合理
13      for (int k = i - 1; k >= 1; k--)
14          if (array[i] == array[k] || abs(array[i] - array[k]) == i - k)
15              return false;
16      return true;
17  }
```



核心点分析

```
1 void search(int i,int n, int array[]) {  
2     if(____1_____) Print(n, array);  
3     else {  
4         for(int j = 1; j <= n; j++) {  
5             ____2_____  
6             if(____3_____) search(i+1,n, array);  
7             ____4_____  
8         }  
9     }  
10 }  
  
1 void search(int i,int n, int array[]) {  
2     if(i > n) Print(n, array);  
3     else {  
4         for(int j = 1; j <= n; j++) {  
5             array[i] = j; //在第i行第j列放置一个皇后  
6             if(isOk(i, array)) search(i+1,n, array);  
7             array[i] = 0; //移走第i行第j列放置的皇后  
8         }  
9     }  
10 }
```

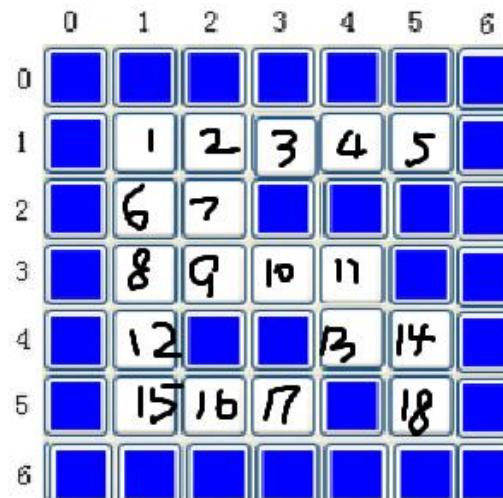
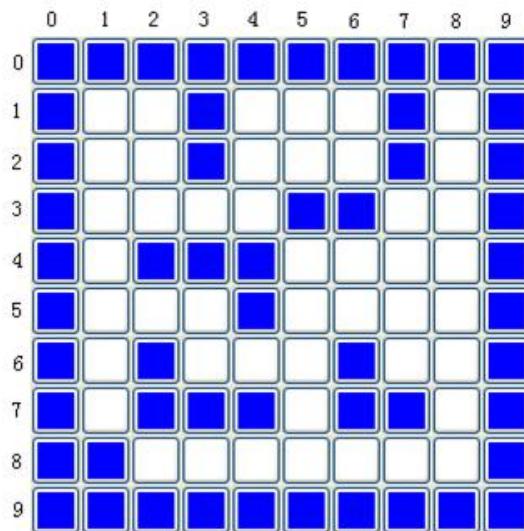
代码

```
1 #include<iostream> //八皇后
2 #include<cstdlib>
3 using namespace std;
4 int sum=0;
5 void Print(int n, int array[]){
6     sum++;
7     for (int j = 1; j <= n; j++)
8         cout<<array[j]<<" ";
9     cout<<endl;
10 }
11
12 bool isOk(int i, int array[]) { //检查当前棋盘布局是否合理
13     for (int k = i - 1; k >= 1; k--)
14         if(array[i] == array[k] || abs(array[i] - array[k]) == i - k)
15             return false;
16     return true;
17 }
18
19 void search(int i,int n, int array[]) {
20     //如果i已经大于最大的行数了, 那么说明布局完成, 则打印结果
21     if(i > n) Print(n, array);
22     else {
23         for(int j = 1; j <= n; j++) {
24             array[i] = j; //在第i行第j列放置一个皇后
25             if(isOk(i, array)) search(i+1,n, array);
26             array[i] = 0; //移走第i行第j列放置的皇后, 回溯。
27         }
28     }
29 }
30 int main() {
31     int n;
32     cin>>n;
33     int Chess[100] = {0}; //棋盘的初始状态, 棋盘上无任何皇后
34     search(1, n, Chess); //摆放皇后
35     cout<<sum<<endl;
36     return 0;
37 }
```



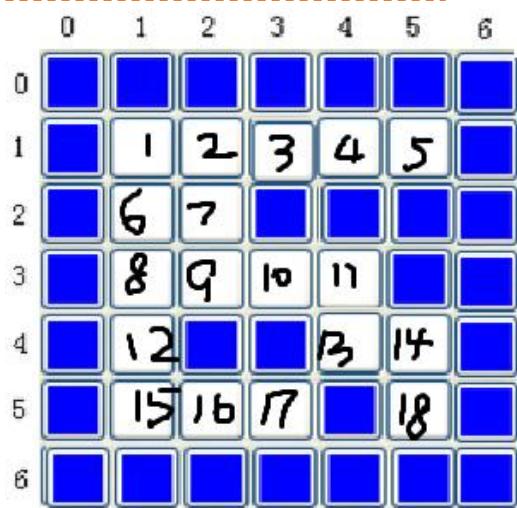
4 [3727]迷宫3求路径次数

给定一个 $M \times N$ 的迷宫图，求从指定入口 $(1,1)$ 到出口 (M,N) 有多少种走法。例如迷宫图如图所示 ($M=10, N=10$)，其中的方块图表示迷宫。对于图中的每个方块，用空白表示通道，用阴影表示墙。要求所求路径必须是简单路径，即在求得的路径上不能重复出现同一通道块。



分析

如图所示，蓝色的表示墙壁，白色的表示可走的方块，题目求解从入口位置1（坐标（1,1））出发，到达位置18（坐标（5,5））。



1. 确定搜索方向

对于每一个方块（位置）而言，共有四种走法：上下左右。
对于计算机而言所有的操作都是有序的，因此需要规定搜索过程中，寻路的顺序，如上下左右，左右上下等。

本解题采用的搜索顺序为左上右下，定义方向数组：

```
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
```



分析

2. 确定搜索"到达目的地"的条件

假设 (xe, ye) 表示迷宫出口位置，则到达迷宫出口的代码可以表示成：

```
if(xe==m&&ye==n){sum++;}
```

3. 记录搜索过程中走过位置的状态

当一个位置被走过以后，为了避免来回重复走，需要记录当前的位置有没有被走过，如果被走过，这个位置就不能再走。假设用 $vis[]$ 数组记录位置的状态，初始时 vis 数组的值全部初始化为0， $vis[x][y]=0$ 表示该位置没有走过， $vis[i][j]=1$ 表示该位置已经被走过。

实现代码如下：

```
vis[x][y]=1;//用1表示该位置没有被走过
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|----|----|----|----|---|---|---|
| 0 | | | | | | | |
| 1 | 1 | 2 | 3 | 4 | 5 | | |
| 2 | 6 | 7 | | | | | |
| 3 | 8 | 9 | 10 | 11 | | | |
| 4 | 12 | | 13 | 14 | | | |
| 5 | 15 | 16 | 17 | 18 | | | |
| 6 | | | | | | | |



分析

```
8 ┌ void dfs(int x,int y){  
9 ┌   if(____1______){  
10    sum++; //到达目的地  
11 }  
12 ┌ else{  
13 ┌   for(int i=0;i<4;i++){  
14     int xx=____2_____ ; //下一个点  
15     int yy=____2_____ ;  
16     if(____3______){  
17       ____4______ ; //记录走过  
18       ____5______ ;  
19       ____6______ ; //退格， 下一次还可以走  
20     }  
21   }  
22 }  
23 }
```



代码实现

```
8 void dfs(int x,int y){  
9     if(x==xe&&y==ye){  
10        sum++; //到达目的地  
11    }  
12    else{  
13        for(int i=0;i<4;i++){  
14            int xx=x+dir[i][0]; //下一个点  
15            int yy=y+dir[i][1];  
16            if(a[xx][yy]==0&&vis[xx][yy]==0){  
17                vis[xx][yy]=1; //记录  
18                dfs(xx,yy);  
19                vis[xx][yy]=0; //退  
20            }  
21        }  
22    }  
23 }
```

```
1 #include<iostream>  
2 #include<cstring>  
3 using namespace std;  
4 int a[12][12];//定义迷宫  
5 int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};  
6 int vis[12][12];//标记有没有走过  
7 int sum=0,xe,ye;//xe,ye终点  
25 int main(){  
26     int m,n,i,j;  
27     cin>>m>>n;  
28     xe=m,ye=n;  
29     memset(a,1,sizeof(a));  
30     for(i=1;i<=m;i++)  
31         for(j=1;j<=n;j++)  
32             cin>>a[i][j];  
33     vis[1][1] = 1; //记录第一点没访问过  
34     dfs(1,1);  
35     cout<<sum<<endl;  
36     return 0;  
37 }  
38 }
```



代码实现（理解回溯）

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次走
    }
}
```

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次走
    }
}
```

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次走
    }
}
```

```
for(int i=0;i<4;i++){
    int xx=x+dir[i][0];//下一个点
    int yy=y+dir[i][1];
    if(a[xx][yy]==0&&vis[xx][yy]==0){
        vis[xx][yy]=1;//记录走过
        dfs(xx,yy);
        vis[xx][yy]=0;//退格，下一次走
    }
}
```

| | | | | | | |
|---|----|----|----|----|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 2 | 3 | 4 | 5 | | |
| 2 | 6 | 7 | | | | |
| 3 | 8 | 9 | 10 | 11 | | |
| 4 | 12 | | 13 | 14 | | |
| 5 | 15 | 16 | 17 | 18 | | |
| 6 | | | | | | |



4 [3727] 迷宫3（二）栈模拟

```
1 #include<iostream>
2 #include<cstring>
3 #include<stack>
4 using namespace std;
5 int a[12][12];//定义迷宫
6 int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
7 int vis[12][12];//标记有没有走过
8 int sum=0,xe,ye;//xe,ye终点
9
10 struct node{
11     int x,y;
12     int dir;
13 };
```

| | | | | | | |
|---|----|----|----|----|----|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | | | | | | |
| 1 | 1 | 2 | 3 | 4 | 5 | |
| 2 | 6 | 7 | | | | |
| 3 | 8 | 9 | 10 | 11 | | |
| 4 | 12 | | 13 | 14 | | |
| 5 | 15 | 16 | 17 | | 18 | |
| 6 | | | | | | |



主函数

```
15 int main(){
16     int m,n,i,j;
17     cin>>m>>n;
18     int xe=m,ye=n;
19     memset(a,1,sizeof(a));
20     for(i=1;i<=m;i++)
21         for(j=1;j<=n;j++)
22             cin>>a[i][j];
23
24     node start,t;
25     start.x=1,start.y=1,start.dir=0;
26     stack<node>st;
27     st.push(start);
28     while(!st.empty())
29     {
30
31         cout<<sum<<endl;
32         return 0;
33     }
34 }
```

| | | | | | | | |
|---|---|----|----|----|----|----|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | | | | | | | |
| 1 | | 1 | 2 | 3 | 4 | 5 | |
| 2 | | 6 | 7 | | | | |
| 3 | | 8 | 9 | 10 | 11 | | |
| 4 | | 12 | | 13 | 14 | | |
| 5 | | 15 | 16 | 17 | | 18 | |
| 6 | | | | | | | |



栈模拟过程1

```
28     while(!st.empty())
29     {
30         node e=st.top(); //获取栈点元素
31         vis[e.x][e.y]=1; //记录状态为已处理
32         if(e.x==xe&&e.y==ye)
33             { //到达出口
34                 sum++;
35                 vis[e.x][e.y]=0; //重新可用
36                 st.pop();
37             }
38         while(!st.empty()&&st.top().dir==4) //走不通了回退
39             vis[st.top().x][st.top().y]=0, st.pop();
40         if(st.empty())
41             break; //处理完毕
42         e=st.top();
43         st.pop(); //弹出栈点元素
44         for(int i=e.dir; i<4; i++)
```

| | | | | | | | |
|---|----|----|----|----|----|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | | | | | | | |
| 1 | | 1 | 2 | 3 | 4 | 5 | |
| 2 | 6 | 7 | | | | | |
| 3 | 8 | 9 | 10 | 11 | | | |
| 4 | 12 | | 13 | 14 | | | |
| 5 | 15 | 16 | 17 | | 18 | | |
| 6 | | | | | | | |



栈模拟过程2

```
44     for(int i=e.dir;i<4;i++)
45     {//处理下一个点
46         int xx=e.x+dir[i][0];//下一个点
47         int yy=e.y+dir[i][1];
48         if(a[xx][yy]==0&&vis[xx][yy]==0){//可走
49             node r;
50             r.x=xx,r.y=yy,r.dir=0;
51             e.dir=i+1;//更新e的方向
52             st.push(e);//更新栈内e的值，因为修改了方向
53             st.push(r);
54             break;//找到一个可走点，退出继续搜索
55         }
56         if(i==3)
57         {//该方向搜索完毕
58             e.dir=i+1;
59             st.push(e);
60         }
61     }
62 }
63 cout<<sum<<endl;
64 return 0;
65 }
```



5 [2769] 迷宫

一天Extense在森林里探险的时候不小心走入了一个迷宫，迷宫可以看成是由 $n * n$ 的格点组成，每个格点只有2种状态，.和#，前者表示可以通行后者表示不能通行。同时当Extense处在某个格点时，他只能移动到东南西北(或者说上下左右)四个方向之一的相邻格点上，Extense想要从点A走到点B，问在不走出迷宫的情况下能不能办到。如果起点或者终点有一个不能通行(为#)，则看成无法办到。

输入

第1行是测试数据的组数k，后面跟着k组输入。每组测试数据的第1行是一个正整数n ($1 \leq n \leq 100$)，表示迷宫的规模是 $n * n$ 的。接下来是一个 $n * n$ 的矩阵，矩阵中的元素为.或者#。再接下来一行是4个整数ha, la, hb, lb，描述A处在第ha行，第la列，B处在第hb行，第lb列。注意到ha, la, hb, lb全部是从0开始计数的。

输出

k行，每行输出对应一个输入。能办到则输出"YES"，否则输出"NO"。



5 [2769] 迷宫

样例输入

2
3
.##
.##
#..
0 0 2 2
5
....

####.#
.##..
####..
.##.

0 0 4 0

样例输出

YES
NO



分析

```
1 #include <iostream>
2 using namespace std;
3 int n;
4 char a[105][105];
5 int ha,la,hb,lb;//起点、终点
6 int d[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//方向数组
7 int flag=0;//用于标记是否走到出口
8 void dfs(int x,int y){
9     if(x==hb&&y==lb){
10         _____;
11         return;
12     }
13     else{
14         for(int i=0;i<4;i++){
15             _____;
16             if(a[xx][yy]=='.'){
17                 _____;
18                 dfs(xx,yy);
19                 _____;
20             }
21         }
22     }
23 }
```



代码

```
1 #include <iostream>
2 using namespace std;
3 int n;
4 char a[105][105];
5 int ha,la,hb,lb;//起点、终点
6 int d[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//方向数组
7 int flag=0;//用于标记是否走到出口
8 void dfs(int x,int y){
9     if(flag==1) return; //找到了就return
10    if(x==hb&&y==lb){
11        flag=1;
12        return;
13    }
14    else{
15        for(int i=0;i<4;i++){
16            int xx=x+d[i][0];
17            int yy=y+d[i][1];
18            if(a[xx][yy]=='.'){
19                a[xx][yy]='#';//保存结果
20                dfs(xx,yy);
21                a[xx][yy]='.';
22            }
23        }
24    }
25 }
```

```
27 int main(){
28     int k;
29     cin>>k;
30     while(k--){
31         flag=0;
32         cin>>n;
33         for(int i=0;i<n;i++)
34             for(int j=0;j<n;j++)
35                 cin>>a[i][j];//输入地图
36         cin>>ha>>la>>hb>>lb;//输入起点和终点
37         dfs(ha,la);
38         if(flag==1) cout<<"YES"=<<endl;
39         else cout<<"NO"=<<endl;
40     }
41 }
42
43 }
```

6 [2804] 走迷宫

一个迷宫由R行C列格子组成，有的格子里有障碍物，不能走；有的格子是空地，可以走。

给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走，不能斜着走。

输入

第一行是两个整数，R和C，代表迷宫的长和宽。（ $1 \leq R, C \leq 40$ ）

接下来是R行，每行C个字符，代表整个迷宫。

空地格子用‘.’表示，有障碍物的格子用‘#’表示。

迷宫左上角和右下角都是‘.’。

输出

输出从左上角走到右下角至少要经过多少步（即至少要经过多少个空地格子）。计算步数要包括起点和终点。



6 [2804] 走迷宫

样例输入

5 5

..###

#....

#.#.#

#.#.#

#.#..

样例输出

9



思路

```
2 #include<cstring>
3 using namespace std;
4 char a[12][12];//定义迷宫
5 int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
6 int vis[12][12];//标记有没有走过
7 int minstep=10000,xe,ye;//xe,ye终点
8 void dfs(int x,int y,int step){
9     if(x==xe&&y==ye){
10         _____1_____ ;
11     }
12     else{
13         for(int i=0;i<4;i++){
14             _____2_____ ;
15             if( _____3_____ ){
16                 vis[xx][yy]=1;//记录走过
17                 _____4_____ ;
18                 vis[xx][yy]=0;//退格， 下一次还可以走
19             }
20         }
21     }
22 }
```



代码

```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 char a[12][12];//定义迷宫
5 int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
6 int vis[12][12];//标记有没有走过
7 int minstep=10000,xe,ye;//xe,ye终点
8 void dfs(int x,int y,int step){
9     if(x==xe&&y==ye){
10         if(step<minstep)minstep=step;
11     }
12     else{
13         for(int i=0;i<4;i++){
14             int xx=x+dir[i][0];//下一个点
15             int yy=y+dir[i][1];
16             if(a[xx][yy]=='.')&&vis[xx][yy]==0{
17                 vis[xx][yy]=1;//记录走过
18                 dfs(xx,yy,step+1);
19                 vis[xx][yy]=0;//退格，下一次还可以走
20             }
21         }
22     }
23 }
```

```
25 int main(){
26     int m,n,i,j;
27     cin>>m>>n;
28     xe=m,ye=n;
29     memset(a,'#',sizeof(a));
30     memset(vis,0,sizeof(a));
31     for(i=1;i<=m;i++)
32         for(j=1;j<=n;j++)
33             cin>>a[i][j];
34     dfs(1,1,1);
35     cout<<minstep<<endl;
36     return 0;
37 }
```



7 [6922] zb的生日

今天是阴历七月初五，acm队员zb的生日。zb正在和C小加、never在武汉集训。他想给这两位兄弟买点什么庆祝生日，经过调查，zb发现C小加和never都很喜欢吃西瓜，而且一吃就是一堆的那种，zb立刻下定决心买了一堆西瓜。当他准备把西瓜送给C小加和never的时候，遇到了一个难题，never和C小加不在一块住，只能把西瓜分成两堆给他们，为了对每个人都公平，他想让两堆的重量之差最小。每个西瓜的重量已知，你能帮帮他么？

输入

多组测试数据 ($<= 1500$)。数据以EOF结尾第一行输入西瓜数量N ($1 \leq N \leq 20$)第二行有N个数， w_1, \dots, w_n ($1 \leq w_i \leq 10000$)分别代表每个西瓜的重量

输出

输出分成两堆后的质量差

样例输入

5

5 8 13 27 14

样例输出

3



分析

因为只有20个西瓜，所以西瓜的组合为 2^{20} ，约 10^6 ，可以考虑搜索，搜索的过程类似全排列。为了实现“为了对每个人都公平，他想让两堆的重量之差最小”，策略应该怎么考虑？

让每个人的重量无限接近 $v/2$ ，就可以达到最小，因此问题可以转化为“求 $\min(\text{ans}-v/2)$ ”



```
1 #include <iostream>
2 #include <cstdio>
3 #include <string.h>
4 using namespace std;
5 int a[21], v, n, m;
6 void dfs(int i, int ans)
7 {
8     if (ans>v) return; //超过一半退出
9     if (i>n)
10    { //判断完毕
11        if (m<ans) m = ans;
12        return;
13    } //对于每个西瓜两种选择，取和不取
14    dfs(i+1, ans+a[i]);
15    dfs(i+1, ans);
16 }
```

```
18 int main()
19 {
20     int sum;
21     while (~scanf("%d", &n))
22    {
23        sum = 0;
24        for (int i=1; i<=n; i++)
25        {
26            scanf("%d", &a[i]);
27            sum += a[i];
28        }
29        v = sum/2; //把一半体积作为边界
30        m = 0;
31        dfs(1, 0);
32        cout<<sum-2*m<<endl;
33    }
34 }
35 }
```



8 [7506] 等式搜索

有一个未完成的等式： 1 2 3 4 5 6 7 8 9=N
当给出整数N的具体值后，请你在2, 3, 4, 5, 6, 7, 8, 9 这8个数字的每一个前面，或插入一个运算符号“+”号，或插入一个运算符号“-”号，或不插入任何运算符号，使等式成立，并统计出能使等式成立的算式总数，若无解，则输出0。

例如：取N为108时，共能写出15个不同的等式，以下就是其中的二个算式：

$$1+23+4+56+7+8+9=108 \quad 123-45+6+7+8+9=108$$

输入只有1个数，即整数N的值。

输出只有一行，该行只有1个数，表示能使等式成立的算式总数。

样例输入 108

样例输出 15

提示 对于所有的n, $-30000 \leq n \leq 1000000$



1 2 3 4 5 6 7 8 9

8个位置，每个位置3种选择（插入 +， -， 或不插），共 3^8 种，数据不大。直接暴力搜索，用一个数字串c记录已算好的数字，数字串b记录要与c进行加减法的数。



一、DFS（第几轮迭代，当前层的数字，当前总和，判断是否可以用减法）

当迭代的下一层中间是加法的时候,进入下一层 ($x+1$) ,
当前层的数字 (x) , 当前的总和 ($sum+num$) , 判断是否
过掉1 (x) $dfs(x+1, num, sum+num, x)$

二、当是减法的时候，因为开头的1前面不能加减号，所以
就如果最后一个数字是1的话，直接跳过，不进入减法的情
况，所以要用 $if(k!=1)$ 进行判断是否可以进入减法的迭代。
如果不可以就，不进入 dfs ，如果可以，就进入
 $dfs(x+1, x, sum-num, x), 1$ (开头) 计算过后，之后的数字可以
跟上减法



三、当没有操作符的时候，也就是和下一个数字相连，下一步也就是`dfs(x+1,num*10+x,sum,k)`，如果开头为1，下一步仍然不进行任何操作的话，就k保留为1。

四、当走到倒数第二层($x==10$)的时候，最后要么加，要么减，如果最后的答案恰好等于n，`ans++`

五、最后输出`ans`



```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int n,ans;
4 void dfs(int a,int b,int c,int k){
//a记录已用数字的数量+1，也是下一个要进行操作的数字
//b表示后面的数字串,c表示前面的数字串,k记录b开头的数字
7 if(a==10){//当A为10时，就可以满足条件进入循环
8     if(b+c==n||c-b==n)ans++;
9     return;//当他们进行加减后为N的值，就进行统计方案数。
10 }
11 dfs(a+1,a,c+b,a);//加法
12 if(k!=1)dfs(a+1,a,c-b,a);//减法
13 dfs(a+1,b*10+a,c,k);//处理合并数的情况
14 return;
15 }
```

```
16 int main(){
17     cin>>n;
18     dfs(2,1,0,1);
19     cout<<ans;
20 }
21 }
```



9 [7507] 旅行

某趟列车的最大载客容量为 V 人，沿途共有 n 个停靠站，其中始发站为第1站，终点站为第 n 站。在第1站至第 $n-1$ 站之间，共有 m 个团队申请购票搭乘，若规定：（1）对于某个团队的购票申请，要么全部满足，要么全部拒绝，即不允许只满足部分。（2）每个乘客的搭乘费用为其所乘站数。问：应如何选择这些购票申请，能使该趟列车获得最大的搭乘费用？其中，每个团队的购票申请格式是以空格分隔的三个整数： a b t ，即表示有 t 个人需要从第 a 站点乘至第 b 站点（注：每个团队的所有人员都必须同时在 a 站上车，且必须同时在后面的 b 站下车）。



输入 有若干行。其中：

第1行只有三个整数n, m, v, 分别表示站点数、申请数、列车的最大载客容量。这三个整数之间都以一个空格分隔。第2行至第m+1行，每行有三个整数，中间都以一个空格分隔。其中第k+1行的三个整数a, b, t表示第k个申请，含义为：有t个人需要从第a站乘至第b站。

输出 只有一行，该行只有一个整数，为该列车能获得的最大搭乘费用。

样例输入

3 3 5

1 2 2

2 3 5

1 3 4

样例输出

8

提示

当只选择第3个申请时，能获得的最大搭乘费用为 $(3-1)*4=8$ 。



没什么想法，看到数据较小，就直接暴搜了。搜索每一个团队选或不选的方案，看看可不可行，记录可行方案中的最大值。



```
1 #include<bits/stdc++.h>
2 #include<string>
3 using namespace std;
4 typedef long long ll;
5 const int mod=998244353;
6 const int N=2e2+10;
7 int n,m,v,maxx=0;
8 int a[20],b[20],t[20],vis[20],w[15];
9 int isok(){
10     int num=0;
11     for(int i=1;i<=n;i++){
12         num+=w[i];//num记录这次方案中每个车站上人的数量
13         if(num>v) return 0;
14     }
15     return 1;
16 }
```

```

17 void dfs(int x){//遍历每一个团队选或不选的方案
18     if(x==m+1){
19         memset(w,0,sizeof(w));
20         int ans=0;
21         for(int i=1;i<=m;i++){
22             if(vis[i]){//如果该团队被选
23                 w[a[i]]+=t[i];//w记录这次方案中每个车站人数的变化;
24                 w[b[i]]-=t[i];
25                 ans+=t[i]*(b[i]-a[i]);
26             }
27         }
28         if(isok()){//判断方案是否可行
29             maxx=max(ans,maxx);
30         }
31         return;
32     }
33     dfs(x+1);
34     vis[x]=1;
35     dfs(x+1);
36     vis[x]=0;
37     return;
38 }

```

```

39 int main(){
40     cin>>n>>m>>v;
41     for(int i=1;i<=m;i++)
42         cin>>a[i]>>b[i]>>t[i];
43     dfs(0);
44     cout<<maxx;
45 }

```





10 [2017_3] 棋盘

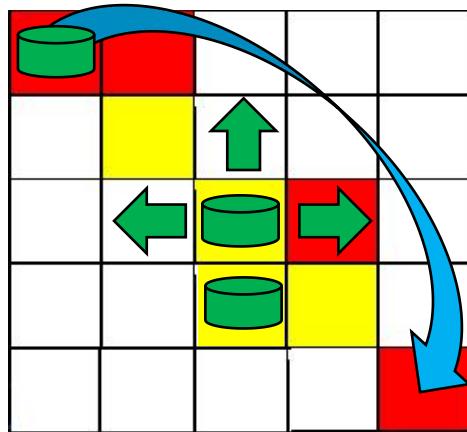
有一个 $m \times m$ 的棋盘，棋盘上每一个格子可能是红色、黄色或没有任何颜色的。你现在要从棋盘的最左上角走到棋盘的最右下角。

任何一个时刻，你所站在的位置必须是有颜色的（不能是无色的），你只能向上、下、左、右四个方向前进。当你从一个格子走向另一个格子时，如果两个格子的颜色相同，那你不需要花费金币；如果不同，则你需要花费 1 个金币。另外，你可以花费 2 个金币施展魔法让下一个无色格子暂时变为你指定的颜色。但这个魔法不能连续使用，而且这个魔法的持续时间很短，也就是说，如果你使用了这个魔法，走到了这个暂时有颜色的格子上，你就不能继续使用魔法；只有当你离开这个位置，走到一个本来就有颜色的格子上的时候，你才能继续使用这个魔法，而当你离开了这个位置（施展魔法使得变为有颜色的格子）时，这个格子恢复为无色。



[2017_3]棋盘

现在你要从棋盘的最左上角，走到棋盘的最右下角，求花费的最少金币是多少？

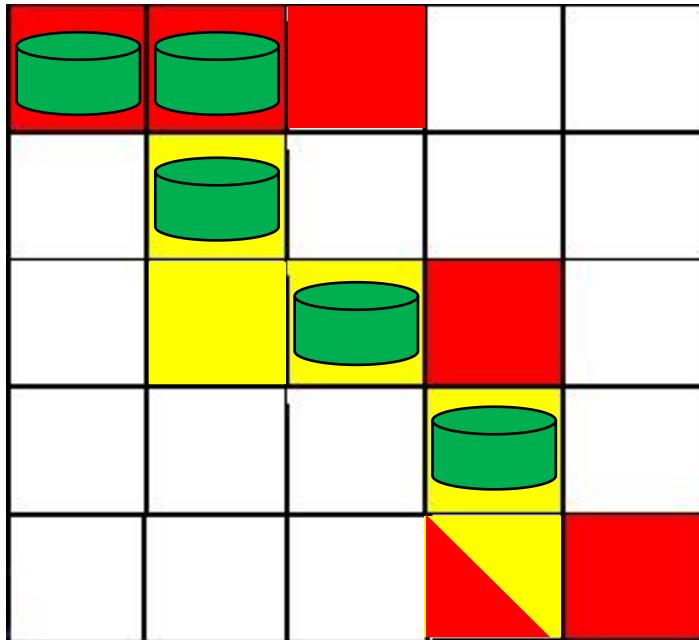




• 输入格式：

- 数据的第一行包含两个正整数 m, n , 以一个空格分开, 分别代表棋盘的大小, 棋盘上有颜色的格子的数量。
- 接下来的 n 行, 每行三个正整数 x, y, c , 分别表示坐标为 (x, y) 的格子有颜色 c
 - 其中 $c=1$ 代表黄色, $c=0$ 代表红色。相邻两个数之间用一个空格隔开。棋盘左上角的坐标为 $(1, 1)$, 右下角的坐标为 (m, m) 。
 - 棋盘上其余的格子都是无色。保证棋盘的左上角, 也就是 $(1, 1)$ 一定是有颜色的。
- 输出格式
 - 输出一行, 一个整数, 表示花费的金币的最小值, 如果无法到达, 输出-1。

搜索过程



样例输入

| | | |
|---|---|---|
| 5 | 7 | |
| 1 | 1 | 0 |
| 1 | 2 | 0 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |
| 3 | 4 | 0 |
| 4 | 4 | 1 |
| 5 | 5 | 0 |

样例输出

| |
|---|
| 8 |
|---|







主函数代码

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define inf 0xffffffff
4 int fx[4] = {-1, 0, 1, 0}; // x偏移量
5 int fy[4] = {0, -1, 0, 1}; // y偏移量
6 int f[110][110]; // 存储每个点的最优解
7 int mp[110][110]; // 存储初始棋盘
8 int m, n, ans = inf;
9 void dfs(int x, int y, int sum, bool frog)
10 {
11     if (x < 0 || y < 0 || x > m || y > n) return;
12     if (f[x][y] != inf) return;
13     if (sum >= ans) return;
14     if (frog) sum++;
15     if (mp[x][y] == 1) sum++;
16     if (mp[x][y] == 2) sum--;
17     if (sum <= ans) f[x][y] = sum;
18     else f[x][y] = inf;
19     for (int i = 0; i < 4; i++) {
20         int nx = x + fx[i], ny = y + fy[i];
21         dfs(nx, ny, sum, !frog);
22     }
23 }
24
25 int main()
26 {
27     memset(f, 0x7f, sizeof(f));
28     scanf("%d %d", &m, &n);
29     for (int i = 1; i <= n; ++i) {
30         int x, y, c;
31         scanf("%d %d %d", &x, &y, &c);
32         mp[x][y] = c + 1;
33         // 1红色 2黄色 0无色 (未赋值, 其初值为0)
34     }
35     dfs(1, 1, 0, false);
36     printf("%d", ans == inf ? -1 : ans);
37     return 0;
38 }
```



DFS代码-1

```
9 void dfs(int x, int y, int sum, bool frog)
10 {
11     if(x < 1 || y < 1 || x > m || y > m) return; // 超出边界
12     if(sum >= f[x][y]) return; // 不符合最优解
13     // 以上两种情况剪枝
14     f[x][y] = sum;
15     if(x==m && y==m) // 终点, 坐标为(m, m)
16     {
17         if(sum < ans) ans = sum; // 更新最优解
18         return;
19     }
20     for(int i = 0; i < 4; ++i)
21     {
22         int xx = x + fx[i];
23         int yy = y + fy[i];
24         if(mp[xx][yy]) // 若下一格有色
25         {
26             if(mp[xx][yy] == mp[x][y]) // 若颜色相同
27                 dfs(xx, yy, sum, false); // 则不花钱
28             else dfs(xx, yy, sum+1, false); // 否则花费+1
29         } else // 否则 (若下一格无色)
30         {
31             if(!frog) // (且) 没有用过魔法
32             {
33                 mp[xx][yy] = mp[x][y];
34                 dfs(xx, yy, sum+2, true); // 使用魔法
35                 mp[xx][yy] = 0; // 回溯
36             }
37         }
38     }
39 }
```



11 [6866]遭遇战

小林和小华在一个 $n \times n$ 的矩形方格里玩游戏，矩形左上角为 $(0, 0)$ ，右下角为 $(n-1, n-1)$ 。两人同时进入地图的随机位置，并以相同速度进行走位。为了隐蔽性，两人都不会再走自己走过的格子。如果两人向某一方向前进，那么他们会跑到不能跑为止，当不能跑的时候，小林会向右转，小华则会向左转，如果不能跑，则不再动。现在已知两人进入地图的初始位置和方向，请算出两人遭遇的位置。

[输入格式]

第 1 行 1 个正整数 t ，表示测试数据组数， $1 \leq t \leq 10$ 。

接下来的 t 组数据，每组数据的第 1 行包含 1 个整数 n ， $1 \leq n \leq 1000$ 。

第 2 行包含 3 个整数 x 、 y 和 d ，表示小林的初始位置和一开始跑的方向。其中， $d=0$ 表示东； $d=1$ 表示南； $d=2$ 表示西； $d=3$ 表示北。

第 3 行与第 2 行格式相同，但描述的是小华。

[输出格式]

输出 t 行，若会遭遇，则包含两个整数，表示他们第一次相遇格子的坐标，否则输出"-1"。





[6866]遭遇战

[输入格式] 第 1 行 1 个正整数 t , 表示测试数据组数, $1 \leq t \leq 10$ 。

接下来的 t 组数据, 每组数据的第 1 行包含 1 个整数 n , $1 \leq n \leq 1000$ 。

第 2 行包含 3 个整数 x 、 y 和 d , 表示小林的初始位置和一开始跑的方向。其中, $d=0$ 表示东; $d=1$ 表示南; $d=2$ 表示西; $d=3$ 表示北。

第 3 行与第 2 行格式相同, 但描述的是小华。

[输出格式] 输出 t 行, 若会遭遇, 则包含两个整数, 表示他们第一次相遇格子的坐标, 否则输出"-1"。

[输入样例]

2

2

0 0 0

0 1 2

4

0 1 0

3 2 0

[输出样例]

-1

1 3





```
35 int main()
36 {
37     int t;
38     scanf("%d", &t);
39     for(int i=1; i<=t; i++){
40         memset(book1, 0, sizeof(book1));
41         memset(book2, 0, sizeof(book2));
42         stop1=0, stop2=0; // 初始化
43         scanf("%d%d%d%d%d%d", &n, &x1, &y1, &d1, &x2, &y2, &d2);
44         if(x1==x2||y1==y2) printf("%d %d\n", x1, y1); // 特判，当两人开始时就在同一点，则直接输出
45     else{
46         for(;;){ //一直跑下去！！！
47             if(!stop1) rr1(0);
48             if(!stop2) rr2(0);
49             if(stop1&&stop2){
50                 printf("-1\n"); // 都凉了
51                 break;
52             }
53             if(x1==x2&&y1==y2){ // 碰上了
54                 printf("%d %d\n", x1, y1);
55                 break;
56             }
57         }
58     }
59 }
60 return 0;
61 }
```





12 [3248] 靶形数独

题目描述：

靶形数独的方格同普通数独一样，在9格宽×9格高的大九宫格中有9个3格宽×3格高的小九宫格（用粗黑色线隔开的）。在这个大九宫格中，有一些数字是已知的，根据这些数字，利用逻辑推理，在其他的空格上填入1到9的数字。每个数字在每个小九宫格内不能重复出现，每个数字在每行、每列也不能重复出现。但靶形数独有一点和普通数独不同，即每一个方格都有一个分值，而且如同一个靶子一样，离中心越近则分值越高。

上图具体的分值分布是：最里面一格（黄色区域）为10分，黄色区域外面的一圈（红色区域）每个格子为9分，再外面一圈（蓝色区域）每个格子为8分，蓝色区域外面一圈（棕色区域）每个格子为7分，最外面一圈（白色区域）每个格子为6分，如上图所示。比赛的要求是：每个人必须完成一个给定的数独（每个给定数独有可能有不同的填法），而且要争取更高的总分数。而这个**总分数即每个方格上的分值和完成这个数独时填在相应格上的数字的乘积的总和**。



[3248] 靶形数独

输入

每组输入数据一共9行，每行9个整数（每个数都在0—9的范围内），表示一个尚未填满的数独方格，未填满的空格用“0”表示。每两个数字之间用一个空格隔开。

输出

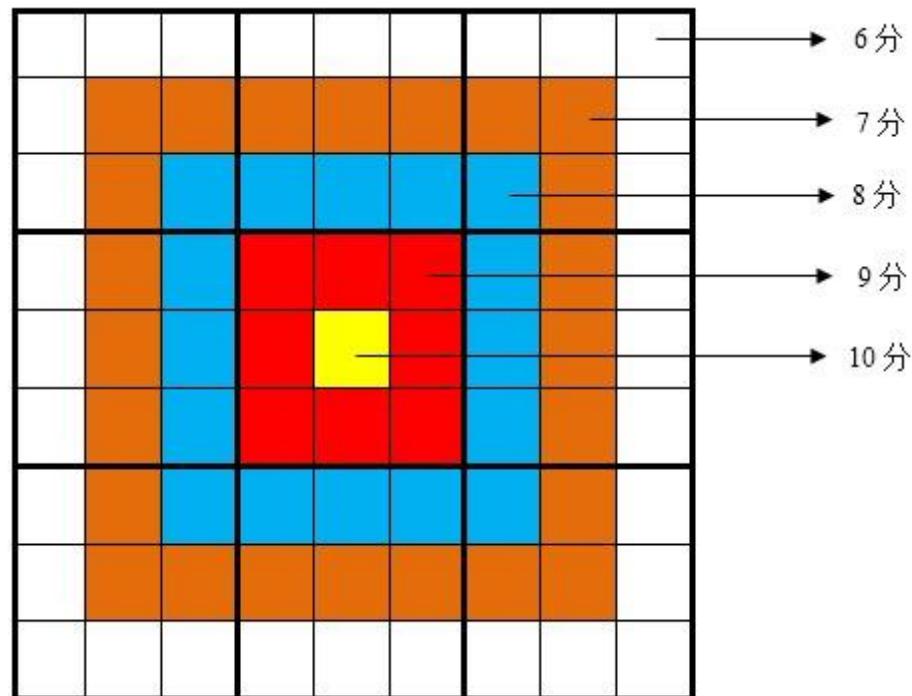
每组输出共1行。输出可以得到的靶形数独的最高分数。如果这个数独无解，则输出整数-1。

数据规模：

40%的数据，数独中非0数的个数不少于30；

80%的数据，数独中非0数的个数不少于26；

100%的数据，数独中非0数的个数不少于24





[3248] 靶形数独

样例输入

```
7 0 0 9 0 0 0 1  
1 0 0 0 0 5 9 0 0  
0 0 0 2 0 0 0 8 0  
0 0 5 0 2 0 0 0 3  
0 0 0 0 0 0 6 4 8  
4 1 3 0 0 0 0 0 0  
0 0 7 0 0 2 0 9 0  
2 0 1 0 6 0 8 0 4  
0 8 0 5 0 4 0 1 2
```

样例输出

2829

样例 最大结果

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 5 | 4 | 9 | 3 | 8 | 2 | 6 | 1 |
| 1 | 2 | 8 | 6 | 4 | 5 | 9 | 3 | 7 |
| 6 | 3 | 9 | 2 | 1 | 7 | 4 | 8 | 5 |
| 8 | 6 | 5 | 4 | 2 | 9 | 1 | 7 | 3 |
| 9 | 7 | 2 | 3 | 5 | 1 | 6 | 4 | 8 |
| 4 | 1 | 3 | 8 | 7 | 6 | 5 | 2 | 9 |
| 5 | 4 | 7 | 1 | 8 | 2 | 3 | 9 | 6 |
| 2 | 9 | 1 | 7 | 6 | 3 | 8 | 5 | 4 |
| 3 | 8 | 6 | 5 | 9 | 4 | 7 | 1 | 2 |



[3248] 靶形数独

分析：

仔细理解可以看出这是一道深度优先搜索的题目，对于每个值为零的点需要搜索填入相应的值，最后在所有结果中取最大值。

对于每个填入的点，有3个约束条件：1.行不充分2.列不重复3.小九宫格不重复

本体数据规模较大，正常搜索会使得代码超时，需要进行一点的优化



[3248]靶形数独-1

```
3 using namespace std;
4 bool r[10][10],l[10][10],s[10][10];//行, 列, 小九宫格
5 int a[10][10],b[10][10];
6 int ans=-1,score;//若无答案, 输出-1
7 int getscore(int x,int y,int k)//计算所在分值
8 {
9     if(x==5&&y==5) return 10*k;
10    else if(x>=4&&x<=6&&y>=4&&y<=6) return 9*k;
11    else if(x>=3&&x<=7&&y>=3&&y<=7) return 8*k;
12    else if(x>=2&&x<=8&&y>=2&&y<=8) return 7*k;
13    else return 6*k;
14 }
15 bool fillin(int x,int y,int k)
16 {
17     if(r[x][k]||l[y][k]||s[(x-1)/3*3+(y-1)/3+1][k]) return 0;
18     b[x][y]=k;
19     r[x][k]=l[y][k]=s[(x-1)/3*3+(y-1)/3+1][k]=1;
20     score+=getscore(x,y,k);//在填入的时候一并计算分值
21     return 1;
22 }
```



[3248]靶形数独-1

```
28 void search(int x,int y)
29 {
30     if(x==10&&y==1){ans=max(ans,score);return;}//到达
31
32     if(b[x][y]){//已有数值，无需填入
33         if(y==9)search(x+1,1);
34         else search(x,y+1);
35     }
36     else{
37         for(int i=1;i<=9;i++){
38             int t=score;
39             if(fillin(x,y,i)){
40                 if(y==9)search(x+1,1);
41                 else search(x,y+1);
42                 del(x,y,i);
43                 score=t;//分数回溯
44             }
45         }
46     }
47 }
48 int main()
49 {
50     for(int i=9;i>0;i--)
51     for(int j=9;j>0;j--)[
52         cin>>a[i][j];
53         if(a[i][j])fillin(i,j,a[i][j]);
54     }
55     search(1,1);
56     cout<<ans;
57     return 0;
58 }
```

23 void del(int x,int y,int k)//回溯
24 {
25 b[x][y]=0;
26 r[x][k]=l[y][k]=s[(x-1)/3*3+(y-1)/3+1][k]=0;
27 }

[3248]靶形数独-2

如果在搜索的时候可以从已知条件多的地方开始，这样就会大大减少处理与回溯，从而降低时间复杂度。
通过结构体与排序实现搜索

```
44 int main(){
45     for(int i=0;i<9;i++){
46         so[i].num = i;
47         for(int j=0;j<9;j++){
48             cin>>a[i][j];
49             if(a[i][j]){//数字已出现
50                 vx[i][a[i][j]] = vy[j][a[i][j]] = vc[i/3*3+j/3][a[i][j]] = 1;
51                 so[i].cnt++;
52             }
53         }
54     }
55 }
56 sort(so,so+9,cmp);//排序决定处理次序
57 dfs(0,0);
58 cout<<ans<<endl;
59
60 }
```



[3248]靶形数独-2

```
4 int a[10][10];
5 int vx[10][10], vy[10][10], vc[10][10];
6 int ans = -1;
7 struct Node{
8     int num, cnt;
9     }so[10];
10 bool cmp(Node x, Node y){
11     return x.cnt > y.cnt;
12 }
13 void dfs(int x, int y){
14     int sx = x;
15     x = so[sx].num; // 把原始行拿出来
16     if(sx==9){
17         int sum=0;
18         for(int i=0; i<9; i++){
19             for(int j=0; j<9; j++){
20                 sum += a[i][j] * min(10-abs(i-4), 10-abs(j-4));
21             }
22         }
23         ans = max(ans, sum);
24         return ;
25     }
26     if(y==9){
27         dfs(sx+1, 0);
28         return ;
29     }
30     if(a[x][y]){
31         dfs(sx, y+1);
32     }else{
33         for(int i=1; i<=9; i++){
34             if(!vx[x][i] && !vy[y][i] && !vc[x/3*3+y/3][i]){
35                 vx[x][i] = vy[y][i] = vc[x/3*3+y/3][i] = 1;
36                 a[x][y] = i;
37                 dfs(sx, y+1);
38                 vx[x][i] = vy[y][i] = vc[x/3*3+y/3][i] = 0;
39                 a[x][y] = 0;
40             }
41         }
42     }
43 }
```

13 [3038] 单词接龙

单词接龙是一个与我们经常玩的成语接龙相类似的游戏，现在我们已知一组单词，且给定一个开头的字母，要求出以这个字母开头的最长的“龙”
(每个单词都最多在“龙”中出现两次)，在两个单词相连时，其重合部分合为一部分，例如beast和astonish，如果接成一条龙则变为beastonish，另外相邻的两部分不能存在包含关系，例如at和atide间不能相连。

输入

每个测试文件只包含一组测试数据，每组输入的第一行为一个单独的整数n ($n \leq 20$) 表示单词数，以下n行每行有一个单词，输入的最后一行为一个单个字符，表示“龙”开头的字母。你可以假定以此字母开头的“龙”一定存在。



3.2.8 [3038] 单词接龙

输出

对于每组输入数据，输出以此字母开头的最长的“龙”的长度。

下面的测试样例最后连成的“龙”为atoucheatactactouchchoose。

样例输入

5

at

touch

cheat

choose

tact

a

样例输出

23



思路：首先要知道两个单词合并时，合并部分取的是最小重叠部分，相邻的两部分不能存在包含关系就是说如果存在包含关系，就不能标记为使用过，每个单词最多出现两次。搜索的时候开个vis标记数组，用来标记每个单词使用的次数，从开头字母开始搜索，两层for，第一层for搜索每一个单词，第二层for是判断我们搜索的单词的第几位和枚举的单词的首字母相同，比如搜索的单词是touch，当遍历到cheat这个单词时发现touch的第四位和枚举的单词cheat相同时，我们用while循环找出重叠部分长度，那么当前的长度就是=当前长度 + 枚举单词长度 - 重叠部分的长度。回溯的时候vis标记减一，恢复长度就好了。注意初始化now为1，因为开头是单个字母。



```
//本题由于都涉及到了单词，用string更加直观一些；  
//string a[22],可以等价为 char a[99999999][22]  
/*
```

这里介绍一下 substr 方法 第一次写的时候一个个
复制 strcpy 有达不到部分复制的效果

很麻烦，我就百度到了这个函数 substr();
比如 string str1 = "0123456789";
string a1 = str1.substr(5);
a1 = "56789";
而如果为 a1 = str1.substr(1,5);
a1 = "12345";
*/



```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 string a[25],start;
5 int ans,vis[22],n;
6
7 int judge(string s,string d)//用来记录两书字符串相同的部分长度
8 {
25 void dfs(string s,int now)
26 {
44 int main()
45 {
46     cin>>n;
47     memset(vis,0,sizeof(vis));
48     ans=-1;
49     for(int i=1;i<=n;i++)
50         cin>>a[i];
51     cin>>start;
52     dfs(start,0);
53     cout<<ans<<endl;
54     return 0;
55 }
```



```
7 int judge(string s, string d) //用来记录两书字符串相同的部分长度
8 {
9     if(s.size() == 1) //开始第一个字母
10    {
11        if(s[0] == d[0])
12            return d.size(); //d就是开头的单词
13        else
14            return 0;
15    }
16    for(int i = 1; i <= s.size() - 1; i++)
17    {
18        string tem1 = s.substr(s.size() - i); //第i+1个字母 到最后一个字母
19        string tem2 = d.substr(0, i); //从第一个字母到 i-1 个字母 前i个单词
20        if(tem1 == tem2)
21            return d.size() - i; //string 可以直接比较
22    }
23    return 0;
24 }
```



```
25 void dfs(string s,int now)
26 {
27     if(now!=0)//记录最长度
28         ans=max(ans,now);
29     for(int i=1;i<=n;i++)
30     {
31         if(vis[i]!=2)//如果出现两次以上 就不需要判断此点
32         {
33             int temp=judge(s,a[i]);//重合之后需要添加的单词长度
34             if(temp!=0)
35             {
36                 vis[i]++; //该单词出现次数+1
37                 dfs(a[i],now+temp); //继续展开搜索
38                 vis[i]--;//回溯
39             }
40         }
41     }
42     return;
43 }
```





14 [3371]泉水

小f住在农村，离他的家不远有一口井，传说是小f的祖先开掘的。虽然小f的村子里通了自来水，但是由于这口井井水质量非常的好，因此小f仍然喝这口井里的水。小f非常喜欢这口井，所以他经常去打水。小f的家里有n（n是偶数）只桶，这些桶虽然大小相等，但是由于很多都有些破损，所以认为它们是不同的。小f经常挑一根扁担（带两只空桶，必须是空的，且是2只）去井边打水。小f每次去井旁都会把桶中的水装到极限（假设水量无穷，且小f都能够担得动）。设小f挑得是x、y两只桶，则打水一趟需要走time[x,y]分钟。小f想要在最少的时间内用自己的力量把家里所有的空桶装满。小f觉得这是个难题，于是来找你帮忙。。





14 [3371]泉水

第一行有一个数字，是n ($n \leq 20$)。接下来n行，每行n个数字，代表了time矩阵。time矩阵中每一个数都是正整数，且time矩阵中 $\text{time}[i,i]$ 是没有用的。（注意 $\text{time}[i,j]=\text{time}[j,i]$ ）

输出仅包含一行，就是最佳挑水方案的最少时间。

4

0 58 49 81

58 0 38 76

49 38 0 48

81 76 48 0

106





分析

此题可以抽象为在一个上（下）三角矩阵中选择两个受到约束的值，使得这两个值的和最大。

题目 $n \leq 20$, 最大选择范围2000, 时间复杂度最大为 10^6 , 可以用类似求组合数的枚举求解, 采用DFS实现。





分析1：目标状态

目标状态：搜索完每一个桶，判断是否是最小值。

```
if(i>n){//目标状态是把所有的都遍历完
    if(sum<minn){
        minn=sum;
    }
    return;
}
```





分析2：搜索过程

一个扁担两个桶，先搜索第一个桶。如果当前桶不可用，直接继续搜索下一个。

确定一个可用后，找到第二个桶。确定两个桶后，记录状态。开始下一轮搜索。

```
15 | if(b[i]==1)dfs(i+1,sum);
16 | //如果当前i不可用要处理
17 | else{
18 |   for(int j=i+1;j<=n;j++){
19 |     if(b[i]==0&&b[j]==0){
20 |       b[i]=1,b[j]=1;
21 |       dfs(i+1,sum+a[i][j]);
22 |       b[i]=0,b[j]=0;
23 |
24 |
25 |     }
26 |   }
```





分析3：剪枝

1. sum值一旦超过已知的最小值，该搜索树分支就可以结束。

```
if(sum>minn)//剪枝  
    return;
```





代码（主函数）

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int n,a[20][20],b[20]={0};
4 int minn=0x3f3f3f;
5 void dfs(int i,int sum){
30 int main(){
31     cin>>n;
32     for(int i=1;i<=n;i++){
33         for(int j=1;j<=n;j++){
34             cin>>a[i][j];
35         }
36     }           // 从第一个桶开始搜索
37     dfs(1,0); // 初始时间为0
38     cout<<minn;
39     return 0;
40 }
```





代码 (DFS1)

```
5 void dfs(int i,int sum){  
6     if(sum>minn)//剪枝  
7         return;  
8  
9     if(i>n){//目标状态是把所有的都遍历完  
10        if(sum<minn){  
11            minn=sum;  
12        }  
13        return;  
14    }
```





代码 (DFS2)

```
15     if(b[i]==1)dfs(i+1,sum);
16         //如果当前i不可用要处理
17     else{
18         for(int j=i+1;j<=n;j++){
19             if(b[i]==0&&b[j]==0){
20                 b[i]=1,b[j]=1;
21                 dfs(i+1,sum+a[i][j]);
22                 b[i]=0,b[j]=0;
23             }
24         }
25     return;
26 }
27 }
```





分析





1143: 木瓜地

育才中学的小A同学在外春游，不小心游荡到郊外，看到了一块木瓜地，木瓜是小A的最爱了，一看到木瓜胃口就会变得无限大。这块木瓜地被分割成一个 R行C列的网格($1 \leq R \leq 40, 1 \leq C \leq 40$)。小A可以从一个格沿著一条跟x轴或y轴平行的直线走到邻接的另一个格。小A发现一开始她自己在木瓜林的(1,1)，也就是第一行第一列慢悠悠地吃着木瓜。小A总是用她最信赖地双筒望远镜去数每一个邻接的格的木瓜的数目。然后她就游荡到那个有最多没有被吃掉的木瓜的邻接的格子（保证这样的格子只有一个）。按照这种移动方法，最终小A总是会在(R,C)停止然后吃掉那里的木瓜。给定这个木瓜地的大小及每个格的木瓜数 F_{ij} ($1 \leq F_{ij} \leq 100$)，要求小A一共吃了多少个木瓜。



今天的课程结束啦.....



下课了...
同学们再见！

