蛟龙四班

深度优先搜索-回溯

Mas

回溯

搜索与回溯(BackTracking)是计算机解题中常用的算法，很多问题无法根据某种确定的计算法则来求解，可以利用搜索与回溯的技术求解。

回溯是搜索算法中的一种控制策略。它的基本思想是：为了求得问题的解，先选择某一种可能情况向前探索，在探索过程中，一旦发现原来的选择是错误的，就退回一步重新选择，继续向前探索，如此反复进行，直至得到解或证明无解。

#1118 迷宫问题

描述 提交 自定义测试 管理

【问题描述】

设有一个 $N \times N(2 \leq N < 10)$方格的迷宫，入口和出口分别在左上角和右上角。

迷宫格子中分别放 0 和 1，0 表示可通，1 表示不能，入口和出口处肯定是 0。

迷宫走的规则如下所示：

即从某点开始，有八个方向可走，前进方格中数字为 0 时表示可通过，为 1 时表示不可通过，要另找路径。

找出所有从入口（左上角）到出口（右上角）的路径(不能重复)，输出路径总数，如果无法到达，则输出 0。

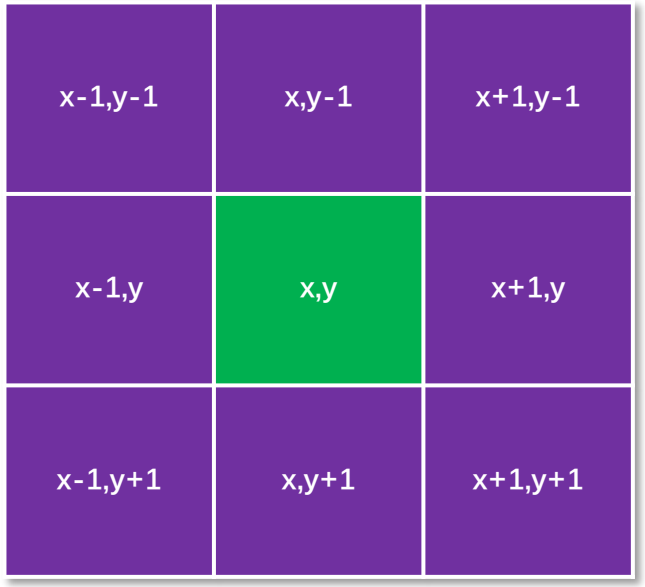
【输入样例】

```
3
0 0 0
0 1 1
1 0 0
```

【输出样例】

```
2
```

迷宫问题



走迷宫的策略:

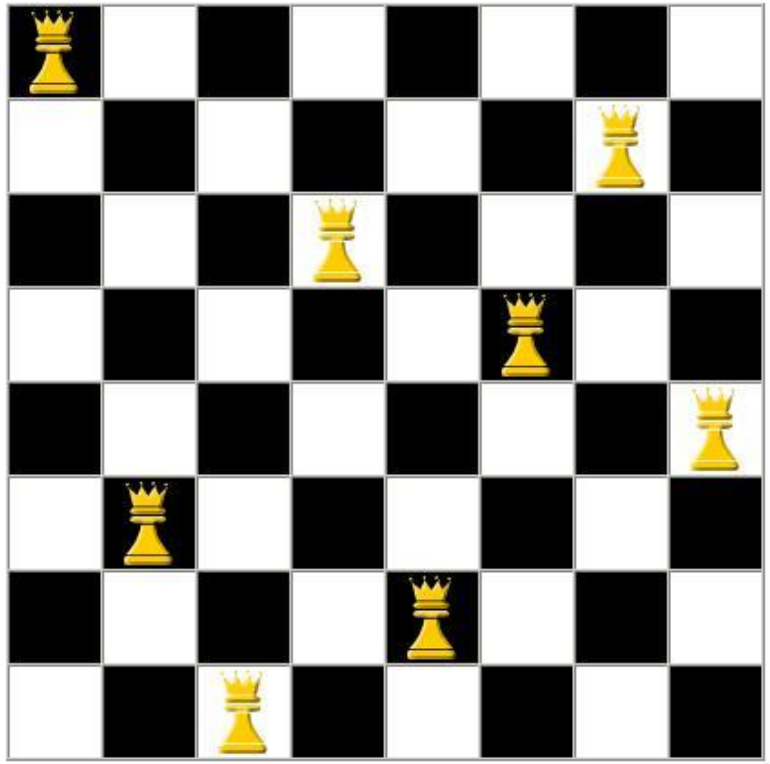
坚定选择一个方向走下去，如果走不通再考虑换条路，已经走过的路就不要再走了

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, maze[11][11], ans = 0, dir[][2] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
bool vis[11][11];
void dfs(int x, int y)
{
    if (x == 0 && y == n - 1)
    {
        ans++;
        return;
    }
    for (int i = 0; i < 8; i++)
    {
        int tx = x + dir[i][0], ty = y + dir[i][1];
        if (tx >= 0 && tx < n && ty >= 0 && ty < n && maze[tx][ty] != 1 && !vis[tx][ty])
        {
            vis[tx][ty] = true;
            dfs(tx, ty);
            vis[tx][ty] = false;
        }
    }
}
int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> maze[i][j];
    vis[0][0] = true;
    dfs(0, 0);
    cout << ans;
    return 0;
}
```

八皇后问题

要在8*8的国际象棋棋盘中放8个皇后，使任意两个皇后都不能互相吃掉。
规则是皇后能吃掉同一行、同一列、同一对角线的棋子。
如图即是一种方案。

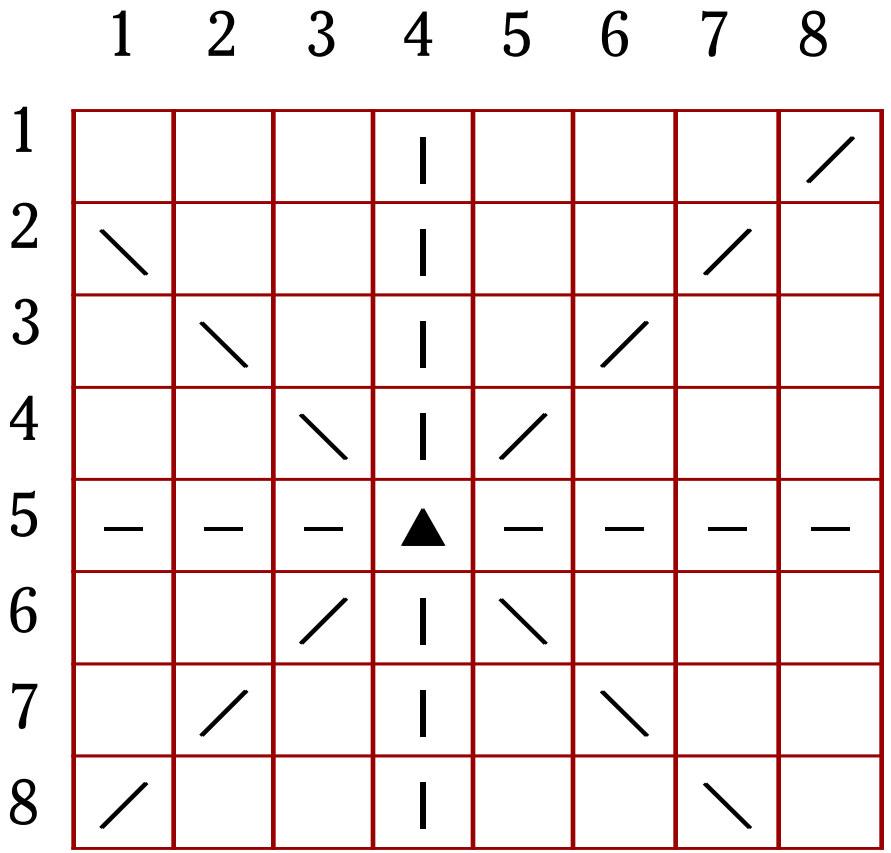
对于每一层，有八种放置。
我们确定每一层的方法,保证不冲突，继续搜索下一层



八皇后问题

考虑每行有且仅有一个皇后，设一维数组ans[1..8]表示皇后的放置：第 i 行皇后放在第 j 列，用ans[i]=j来表示，即下标是行数，内容是列数。例如：ans[3]=5就表示第3个皇后在第3行第5列上。

```
void dfs(int k)
{
    if( k == 8 )
    {
        output();
    }else{
        for(int i = 0;i<8;i++)
        {
            ans[k] = i;
            if( check(i)  )  //检查当前方法是否合法
                dfs(k+1);   //搜索下一层
        }
    }
}
```



八皇后问题

根据上述描述，我们可以得到如果两个皇后Q1(x1, y1)和Q2(x2, y2)不符合要求，则以下四个条件之一必符合。

- x1 == x2 （同一行 ）
- y1 == y2（同一列）
- x1 + y1 == x2 +y2 （斜向正方向）
- x1 - y1 == x2 - y2（斜向反方向）

```
bool check(int row)
{
    for(int i = 0;i<row ;i++)
        if(     ans[row] == ans[i]                  //检测列
                || row + ans[row] == i + ans[i]     //检测正对角线
                || row - ans[row] == i - ans[i] )   //检测反对角线
            return false;
    return true;
}
```

八皇后问题

可以用标记数组优化代码

标记每一列，以及两条斜线

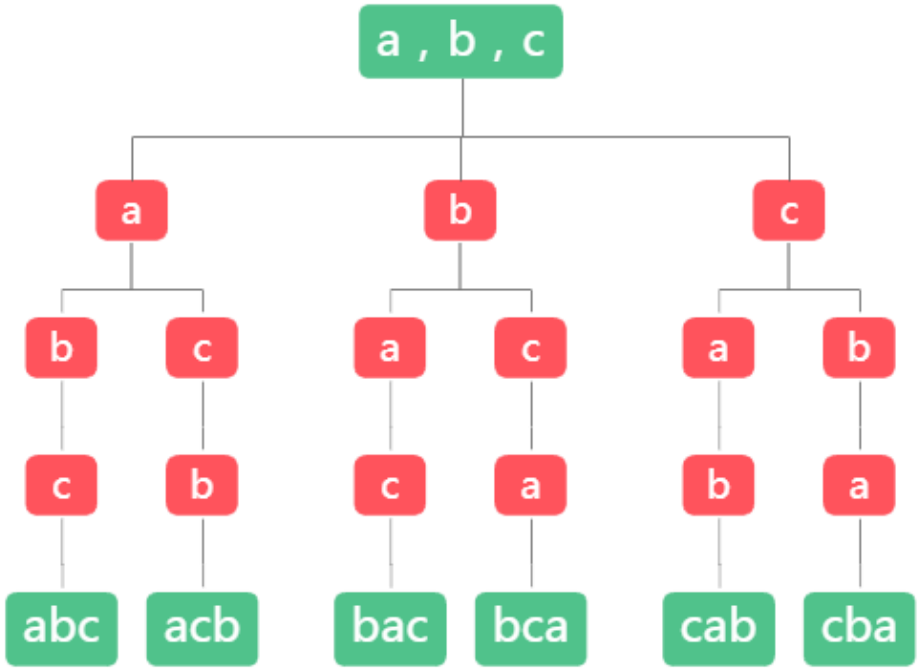
```
void dfs(int k)
{
    if( k == 8 )
    {
        cn++;
        output();
    }else{
        for(int i = 0;i<8;i++)
        {
            ans[k] = i+1;
            if( !col[i] && !line1[k+i] && !line2[k - i + 8 ]  )  //检查当前方法是否合法
            {
                col[i] = true;  //标记列
                line1[k+i] = true; //标记正对角线
                line2[k - i + 8 ] = true; //标记反对角线
                dfs(k+1);   //搜索下一层
                //清除标记
                col[i] = false;
                line1[k+i] = false;
                line2[k - i + 8 ] = false;
            }

        }
    }
}
```

全排列问题

从n个不同元素中任取m（m≤n）个元素，按照一定的顺序排列起来，

叫做从n个不同元素中取出m个元素的一个排列

当m=n时所有的排列情况叫全排列。



全排列问题

【问题描述】

输出自然数 1 到 n 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

【输入格式】

一个数$n(1 \leq n \leq 9)$

【输出格式】

由 $1 \sim n$ 组成的所有不重复的数字序列，每行一个序列。

【输入样例】

```
3
```

【输出样例】

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

位向量标记

可以看成是n层枚举

每层枚举一个从未使用过的数

是否有局限性?

```
int a[] = {1, 2, 3, 4, 5}, ans[11];
bool vis[11];
void permutation(int cur)
{
    if (cur == 5)
    {
        for (int i = 0; i < 5; i++)
            cout << ans[i] << " \n"[i == 4];
        return;
    }

    for (int i = 0; i < 5; i++)
        if (!vis[i])
        {
            vis[i] = true;
            ans[cur] = a[i];
            permutation(cur + 1);
            vis[i] = false;
        }
}
```

邻位交换法

依次选出每一个元素，作为排列的第一个元素，然后对剩余的元素进行全排列，这样递归处理

以对字符串abc进行全排列为例：以abc为例：

- 固定a，求后面bc的排列：abc，acb，求好后，a和b交换，得到bac
- 固定b，求后面ac的排列：bac，bca，求好后，c放到第一位置，得到cba
- 固定c，求后面ba的排列：cba，cab。

```
int a[] = {1, 2, 3}, ans[11];
void permutation(int cur)
{
```

字典序法

设P是1 ~ n的一个全排列:p=p1 p2 ..... pn = p1 p2 ..... pj-1 pj pj+1 ..... pk-1 pk pk+1 ..... pn

1）从排列的右端开始，找出第一个比右边数字小的数字的序号j（j从左端开始计算），即 j=max{ i | pi<pi+1 }

2）在pj的右边的数字中，找出所有比pj大的数中最小的数字pk，即k=max{ i | pi>pj}（右边的数从右至左是递增的，因此k是所有大于pj的数字中序号最大者）

3）对换pi，pk

4）再将pj+1 ..... pk-1 pk pk+1 ..... pn倒转得到排列p'=p1 p2 ..... pj-1 pj pn ..... pk+1 pk pk-1 ..... pj+1，这就是排列p的下一个排列。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n = 4, a[21] = {1, 2, 3, 4};
bool next_permutation()
{
    int pos = n - 1, i;
    while (pos - 1 >= 0 && a[pos - 1] > a[pos])
        pos--;
    pos--;
    if (pos < 0)
        return false;
    for (i = pos + 1; i < n && a[i] > a[pos]; i++)
        ;
    i--;
    swap(a[i], a[pos]);
    reverse(a + pos + 1, a + n);
}
int main()
{
    do
    {
        for (int i = 0; i < n; i++)
            cout << a[i] << " \n"[i == n - 1];
    } while (next_permutation());
    return 0;
}
```

谢谢观看