搜索

严格来说，搜索算法也算是一种暴力枚举策略，但是可以跳过一些无效状态，降低问题规模。在算法竞赛中，如果选手无法找到一种高效求解的方法(如贪心、递推、动态规划、公式推导等)，使用搜索也可以解决一些规模较小的情况；而有的任务就是必须使用搜索来完成。

1. **深度优先搜索与回溯法**

有些题目，无法用简单的循环表示所有枚举情况，需要寻找更加一般化的枚举框架；有些题目，本质上是子集枚举或是排列枚举，但是枚举量过大，需要剪掉过多的无效状态……深度优先搜索应运而生。

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 1 | 3 |
| 1 | 3 | 2 | 4 |
| 4 | 2 | 3 | 1 |
| 3 | 1 | 4 | 2 |

**四阶数独：**

给出一个4\*4的格子，每个格子只能填写1到4的整数，要求每行、每列和四等分更小的正方形部分都刚好由1到4组成。如图所示是一个合法的四阶数独的例子。给出空白的方格，请问：一共有多少种合法的填写方法？

***在此介绍一种新的利用函数递归枚举的方式，***枚举每一个填空中所有可能的选项，然后判断这种选项是否合法。如果这个选项合法的话就填写下一个选项，然后继续；如果这个填空中所有的选项都不合法，那就不用继续枚举下去了，而是去尝试更换上一个填空的选项，继续枚举。***这种方式称为回溯算法，常使用深度优先搜索来实现。***

**回溯算法的一般形式如下：**

void dfs(int k){//k代表递归层数，或者说要填第几个空   
 if(所有空已经填完了){  
 判断最优解/记录答案;  
 return;  
 }  
 for(枚举这个空能填的选项)  
 if(这个选项是合法的){  
 记录下这个空(保存现场);  
 dfs(k+1);  
 取消这个空(恢复现场);  
 }  
}

可以得到完整的程序如下：

#include<bits/stdc++.h>  
using namespace std;  
#define size 5  
int a[size\*size],n=4\*4,ans=0;  
int b1[size][5],b2[size][5],b3[size][5];//分别记录横行，竖行，四小块   
void dfs(int x){//第x个空填什么   
 if(x>n){//如果所有空已经填满   
 ans++;//增加结果数量  
 /\*以输出放置方案  
 for(int i=1;i<=n;i++){  
 printf("%d",a[i]);  
 if(i%4==0)puts("");  
 }  
 puts("");  
 \*/  
 return;  
 }  
 int row=(x-1)/4+1;//横行编号   
 int col=(x-1)%4+1;//竖排编号  
 int block=(row-1)/2\*2+(col-1)/2+1;//小块编号   
 for(int i=1;i<=4;i++)  
 if(b1[row][i]==0&&b2[col][i]==0&&b3[block][i]==0){  
 a[x]=i;//记录放置位置   
 b1[row][i]=1;b2[col][i]=1;b3[block][i]=1;//占位   
 dfs(x+1);//下一层递归  
 b1[row][i]=0;b2[col][i]=0;b3[block][i]=0;//取消占位   
 }  
}  
int main() {  
 dfs(1);  
 printf("%d",ans);  
 return 0;  
}

遇到了需要枚举排序的时候，搜索回溯会很好用。不需要生成所有的序列全排列，而是一个一个地填空，保证填空的时候序列是合法的，这样就可以不用枚举很多无效序列，节约程序运行时间。

对于一些枚举或者枚举子集的问题，可以使用搜索回溯来解决。但如果需要枚举的元素比较多(超过几十个)，即使是搜索回溯也相当慢。可以使用动态规划的背包问题模型更高效地解决。

1. **广度优先搜索**

深度优先搜索会优先考虑搜索的深度。形象点说，就是不找到一个答案不回头。当答案在整棵解答树中比较稀疏时，深度优先搜索可能会先陷入过深的情况，一时半会儿找不到解。有时候需要解决连通性、最短路问题时，可以考虑使用广度优先搜索。

**马的遍历：**

有一个n\*m的棋盘(1<n,m<=400),在某个点上有一个马，要求计算出马到达棋盘上任意一个点最少要走几步。

广度优先搜索会优先考虑每种状态的和初始状态的距离，形象点说，与初始状态越接近的情况就会越先考虑。再具体一点：每个时刻(阶段)要做的事情就是从上个时刻(阶段)每个状态扩展出新的状态。

广度优先搜索使用队列实现：先将初始状态加入到空的队列中，然后每次取出队首，找出队首所能转移到的状态，再将其压入队列；如此反复，直到队列为空。这样就能保证一个状态在被访问的时候一定是采用的最短路径。

广度优先搜索的一般形式如下：

Q.push(初始状态);//将初始状态入队  
while(!Q.empty()){  
 State u=Q.front;//取出队首  
 Q.pop();//出队  
 for(枚举所有可扩展状态)//找到u的所有可达状态v  
 if(是合法的)//v需要满足某些条件，如未访问过、未在队内等  
 Q.push(v);//入队(同时可能需要维护某些必要信息)  
}

完整的代码如下：

#include<bits/stdc++.h>  
using namespace std;  
#define maxn 310  
struct coord{//一个结构体存储x，y两个坐标   
 int x,y;  
};  
queue<coord> Q;//队列  
int ans[maxn][maxn];//记录答案，-1表示未访问   
int walk[8][2]={{2,1},{1,2},{-1,2},{-2,1},{-2,-1},{-1,-2},{1,-2},{2,-1}};//马能走的8个方向   
int main(){  
 int n,m,sx,sy;  
 memset(ans,-1,sizeof(ans));  
 cin>>n>>m>>sx>>sy;  
 coord tmp={sx,sy};  
 Q.push(tmp);//使起点入队扩展  
 ans[sx][sy]=0;  
 while(!Q.empty()){//循环直到队列为空  
 coord u=Q.front();//拿出队首以扩展  
 int ux=u.x,uy=u.y;  
 Q.pop();  
 for(int k=0;k<8;k++) {  
 int x=ux+walk[k][0],y=uy+walk[k][1];  
 int d=ans[ux][uy];  
 if(x<1||x>n||y<1||y>m||ans[x][y]!=-1)  
 continue;//若坐标超过地图范围或者该点已被访问过则无需入队  
 ans[x][y]=d+1;//记录答案，是上一个点多走一步的结果  
 coord tmp={x,y};  
 Q.push(tmp);  
 }  
 }  
 for(int i=1;i<=n;i++,puts(""))  
 for(int j=1;j<=m;j++)  
 printf("%-5d",ans[i][j]);//场宽输出   
 return 0;  
}

对于同一个模型，无论使用深度优先搜索还是广度优先搜索，其解答树是一样的，只是搜索顺序不一样。同样是寻找目标解，深度优先搜索寻找操作步骤字典序最小的解，而广度优先搜索可以找到步骤最少的解。需要根据题目的性质来决定使用什么搜索算法。

使用**万能头文件**必须要确定允许使用时才能加上，因为不是所有的比赛都支持使用万能头文件。使用万能头文件会出现一些潜在的副作用(比如一些变量被定义了)，所以应确保程序能够在评测环境中编译通过后谨慎使用。