马上开始

35tang-C++竞赛系列三阶课程

三五堂

《35tang-C++竞赛系列三阶课程》

枚举和搜索

- 枚举就是把所有可能性列出来，看哪些复合条件。
- 如果条件比较复杂，无法通过循环枚举。
- 搜索枚举会好一些，我们是按照一些条件走下去，相当于枚举里面我们按照规律找可能的。比如迷宫，选数等用搜索，每一步都是枚举（选择）当前所有可能的位置（状态，数），然后进入下一步（下一个位置，下一个状态）。是一种特殊的枚举。
- 比如前面说的排列问题，给出n个数字（1=<n<=100），从里面选k个数字（1=<k<=80）请给出从小到大的所有排列组合。
  - 这个如果是循环枚举，其实就是组合所有可能的k个数字。循环没有办法写了，只能搜索枚举了:一次从多个可能里面选择一个数，进入下一步（选择下一个数），选够k个数就是一条路到头，或者说一个完整的枚举路径，或者说题目要求的排列的一个可能。

复习上节课作业

DFS搜索，一定要搞清楚每一次怎么选，如何走到下一步（下一个分支，下一个情况，下一个状态），也就是走到每一步时侯需要传递哪些参数，或者说哪些东西会变，会影响下一个状态，或者说下一状态怎么才能知道刚才的选择对他的影响。这些参数可以作为函数参数，也可以作为全局变量，是否需要手工回溯其实就是判断在下一步进入不同的分支之前是否被修改了，是否需要在进入另外一个分支之前把修改的东西还原回去。为什么全局变量回溯的可能性大？因为全局变量只有修改了下一个函数才能看到，所以修改可能性大；而参数不一定要修改。

搜索就是一种枚举方法，可以暴力搜索枚举所有情况，可以优化

剪--就是不做不必要的

剪枝与暴力

暴力之前先分析时间函数

暴力也要找到合适的枚举项

判断是否重复出现过，用FLAG（桶）计数（VISITED数组？），实在不行用SET，MAP

很多时候题目的条件隐藏了用什么算法和数据结构

第一步找规律

例题：选数（可重复）

给出n个数字（1=<n<=100），从里面选k个数字（1=<k<=80）请给出从小到大的所有组合

格式如下，第一行n和k空格隔开，第二行n个数字空格隔开:

5 2

1 2 3 4 5

输出

12

13

14

15

21

23

24

25

31

32

34

35

41

42

43

45

51

52

53

54

先对比上节课的题目

- 依次枚举下一个元素
- 有序枚举

```
void dfs(int i)
//i表示从input数组已经选择了第几个
{
    if(outputnum.size()==k) {
        for (int l=0;l<outputnum.size();l++)
                cout<<outputnum[l]<<" ";
        cout<<endl;
        return;
     }
    for (int l=i+1;l<=n;l++)  {
    //这是有序枚举的保证，也是为什么不需要visited数组
        //outputnum[++outlen]=inputnum[l];
        outputnum.push_back(inputnum[l]);
        dfs(l);
        //outlen--;
        outputnum.pop_back();
    }
    return;
}
```

这时候的输出是不重复的，输出如下：

1 2

1 3

1 4

1 5

2 3

2 4

2 5

3 4

3 5

4 5

重复的比如5 4，比如2 1都不会出现

为什么？因为for (int l=i+1;l<=n;l++) 保证了我们每一次按照顺序枚举，只会选择比前一个已经选择的数更靠后的数，所以不会重复。而且也不会选择前面选择过的数。

新思路一全排列，每一步每一个数都要尝试选择

- 仔细观察输出，第一个数可能是所有的输入数字中的一个，第二个也是，第三个也是.....但是如果每条路选择的输出不能重复。比如第一个数已经选择了输入的第2个，那么后面第二个第三个..都不能选这一个。这就类似迷宫，同一个路径上的状态位置不能重复。
- 刚开始设置全局变量visited数组表示原有序列所有的位置是否都选过了，比如上一页的示例数据，设置visited来表示输入数组的1 2 3 4 5是否被选择过，因为同一条路上不能重复选择一个元素。**和迷宫非常类似，不过就是现在每一步是选择一个数，而迷宫的每一步是选择当前的行和列。判断相邻位置或者下一个状态：迷宫只能上下左右没有走过的位置，而这里所有的没有选择过的输入数都可以**。
- 每次遍历输入数组的1到n个位置依次判断visited标记，看是否选过了，如果没有就选择，走过后回溯访问状态。
- 为什么回溯，比如上一次次选了数字1进入到当前状态： 现在选了数字3，递归往下走，递归返回之后（选择3的这条路走完了），这次如果选数字2，数字3还可以再选一次。
- 一起来做一下

DFS模板函数
{
处理当前节点

进入下一个合法节点：
递归调用相邻合法节点

}

全排列举选数DFS函数
{
当前输出数组是否选够了

选择下一个数：递归调用（所有没有选择过的数）
}

有序枚举选数DFS函数
{
当前输出数组是否选够了

选择下一个数：递归调用（下一个数分别选择当前选的数的后面的任意一个可选的数）
}

所以，我们需要的状态是当前输出数组和上一次选择了哪个数，他们可以是全局变量也可以参数传递

所以，我们需要的状态是当前输出数组和当前已经选择了哪些数，下面都通过全局变量实现。已经选择了哪些数不就是visited数组的功能？

```
void dfs()
{
   if(outputnum.size()==k) {
        for (int l=0;l<outputnum.size();l++)   cout<<outputnum[l]<<" ";
        cout<<endl;
        return;
      }
   for (int l=1;l<=n;l++)    { //和有序枚举的区别，每一个数都有可能，而不是从上次选择的下一个开始
      if(!visited[l])
      {
                visited[l]=true;//这个用来标记哪个位置的数选择过了
                outputnum.push_back(inputnum[l]);
                dfs();
                outputnum.pop_back();//回溯
                visited[l]=false; //回溯
      }
   }
   return;
}
```

```
void dfs(int i)//i表示从input数组已经选择了第几个
{
     if(outputnum.size()==k) {
          for (int l=0;l<outputnum.size();l++) cout<<outputnum[l]<<" ";
          cout<<endl;
          return;
      }
     for (int l=i+1;l<=n;l++)                {
     //这是有序枚举的保证，也是为什么不需要visited数组
          //outputnum[++outlen]=inputnum[l];
          outputnum.push_back(inputnum[l]);
          dfs(l);
          //outlen--;
          outputnum.pop_back();
     }
     return;
}
```

剪枝

回顾上节课作业

给定n个不同正整数如何选出若干个数字，使得他们的和是m。列出几种可能。

比如给定输入如下

5　7

5 2 3 4 1

5个数字，和是7，列出所有的可能。

```
void selectn(int i,int sum)        //i是当前该选择a数组的第几个了;sum是当前已经选则的数字的和是多少。
{
                if(sum>m) return;
                if (i==n){//就是说全部找完了一次，到底了，这个时候判断sum==m，如果相等就说明找到了输出
                        if (sum==m)                {
                                for (int j=0;j<len;j++) cout<<outputnum[j]<<" ";
                                cout<<endl;
                        }
                        return;
                }
                //选择i和不选择i两个相邻节点
                //选a[i]
                outputnum[len]=a[i];
                len++;
                selectn(i+1,sum+a[i]) ;
                 //不选 a[i]
                len-- ; //回溯
                 selectn(i+1,sum);
                return;
}
```

每一次sum已经大于m的时候不可能了，不要做下去

剪枝

- 就是把没有必要走下去的路线剪掉（终止，停下来的意思）
- 广义讲，我们前面迷宫找是否可以到达出口的问题里面用的visited数组是不是也算是一种剪枝：因为一个位置走过了就没有必要再走一次了，就可以剪掉了。

2017 NOIP第三题 棋盘

- https://www.luogu.com.cn/problem/P3956
- 棋盘，最大100*100， 有的格子没有颜色，有的红（1），有的黄（0），从左上角到右下角，每次可以选择上下左右四个方向，任何一个时刻，你所站在的位置必须是有颜色的。走到颜色相同的格子不花费金币，颜色不同花费1个金币。你也可以用2金币魔法把下一个无色格子变成红或者黄走上去，但是离开后，颜色变回无色，不能连续使用魔法。问最小花费多少金币。
- 所以：有颜色的格子是可以走的，没有颜色的格子如果使用魔法也是可以走的。
- 先打开真题分析一下示例数据

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。

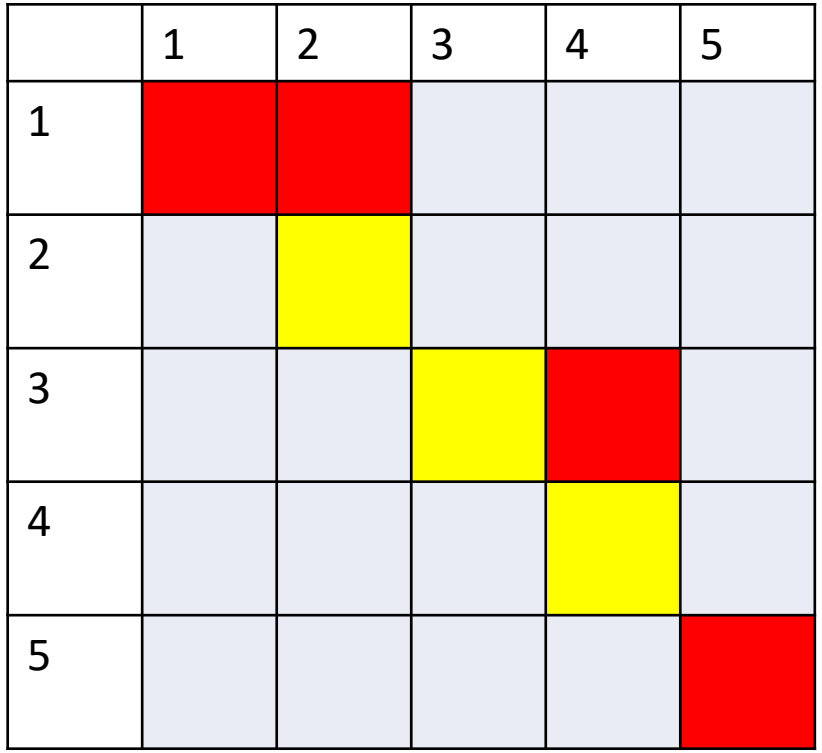
接下来的n行，每行三个正整数x, y,*c*， 分别表示坐标为(x,y)的格子有颜色*c*。

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。

下面示例数据输出是8，为什么？

```
5 7
1 1 0
1 2 0
2 2 1
3 3 1
3 4 0
4 4 1
5 5 0
```



用DFS枚举

- 每一步可能选择上下左右四个相邻点。
- 暴力的方法就是枚举出来所有每一步的可能性。
- DFS是往下走，也就是每一步选择一个位置，进入相邻节点，递归上述逻辑，直到出口，这样就形成了一条路。暴力其实就是枚举出来所有的路线，然后找里面最合适的一个。
- 和上节课作业1”选数的和”不同，这里选择的是每一个位置，由于最后要找最小花费，所以走到下一步不需要传递选择数的和，而是应该传递走到每一个位置的“花费”

怎么做?

- 首先，肯定可以用二维数组把矩阵信息保存起来。和迷宫不同，每一个位置的值不是0或者1，而是颜色。我们枚举路线往下走就是要根据这个矩阵保存的颜色信息来判断。
- 一条路上，走过的格子还有可能走么？其实是不能的，因为重复走肯定花费更大，而且不能来回走造成死循环。但是要回溯，因为每一个位置可能出现在不同的路径。这和迷宫找所有路线的例子类似。
- 所以思路很简单，就是去搜索枚举所有路线，对于每一个位置都去枚举可能的相邻节点，然后进入下一步，当枚举到了出口，那么枚举过的这些位置组成一个路线，我们就是要找代价最小的一个路线。非常类似前面的迷宫找所有路线，不同的地方　：
- 每一步选择相邻节点，迷宫是要去看4个相邻节点是否访问过并且没有障碍，这里虽然也是4个相邻节点，但是需要去看有没有颜色，如果没有颜色并且不能使用魔法，就不能走。
- 迷宫只需要统计多少路线，所以选择了下一个位置之后只要传递行列值走下取就可以了。这里每走一个格子就要加上消耗的金币数目，因为最后每条路想到头了需要打擂得到花费最小的一条路线。
- 迷宫一条路到头了，计数器++，而这里需要对当前路线总的金币数目打擂。

怎么写?

- 和迷宫写起来非常类似，几乎可以用迷宫的标准程序改写。都是从起点到终点。非常类似迷宫找一共多少不同路径的代码。
  - 传递的参数不同，因为最后要判断的是那条路花钱最少，而不是一共多少路线。所以我们应该传递每条路线已经花的钱，并且如果到了终点就把总钱数打擂； 另外，由于魔法不能连续使用，所以我们要传递给下一步上一次魔法用过了没有。
  - 在进入相邻节点的时侯处理复杂一点，要根据当前是否可以施展魔法结合当前row col和相邻节点的颜色，去判断能不能走以及走到下一步的新的money参数。
- void dfs(int row,int col,int money,bool canmagic) 比迷宫找所有路线多了2个参数。

```
void dfs(int row,int col)
{
        if (row==4 && col==4)
        {
                ans++;
                return ;
        }
        visited[row][col]=true;
        //尝试4个方向
        for (int i=0;i<4;i++)
        {
                int newrow=row+rowdir[i];
                int newcol=col+coldir[i];
                if (newrow<0 ||newrow>4 ||newcol<0|| newcol>4
                || visited[newrow][newcol]||maze[newrow][newcol]==1)
                        continue;
                dfs(newrow,newcol); //每条路都要走
        }
        visited[row][col]=false;
        return ;
}
```

参数不同了，需要money和canmagic

出口不同了，到了出口需要打擂

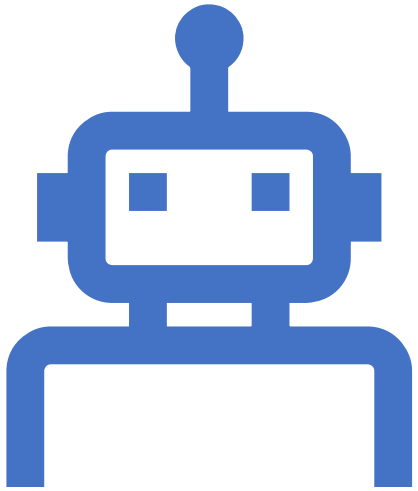
这里不是判断新位置是不是1了，而是如下所述，去判断颜色和canmagic

不是每条路都能走，而是要判断newrow,newcol位置的颜色以及canmagic的状态；

调用下一步的参数需要加上新的money和canmagic

先写出来看看

至少有分数



一起看一下这个超时的程序

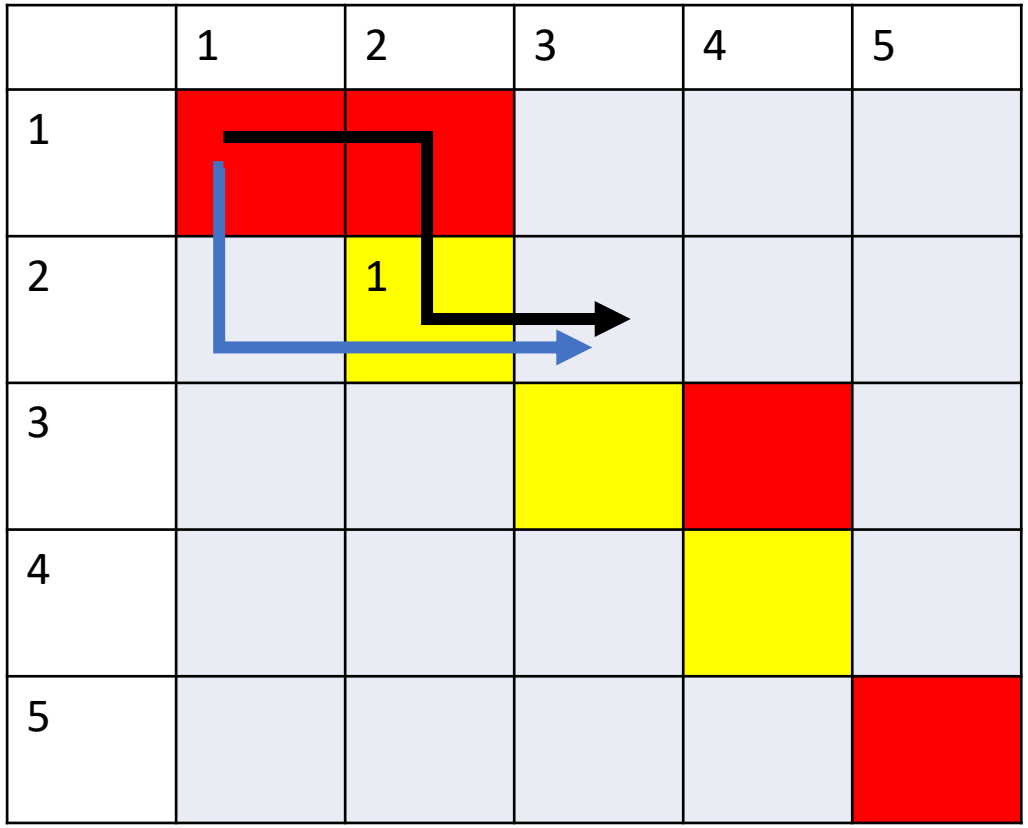
剪枝优化方法一

```
void dfs(int row,int col,int money,bool canmagic)
{
        if(r<=money) return;
/*r是已经走过的路径打擂出来的结果，如果当前路径走到当前row col发现要花的钱已经比r多，就没有必要走下去了，走下去也比r多
但是这句加了还是不行，优化效果有限 */

        if (row==m && col==m) {
                if (money<r) r=money;
                return;
        }
....
```

DFS常用剪枝方法二：记忆化搜索剪枝

- 如图，假设第一次搜索路径如黑色箭头，经过（2，2）点的时侯，花费是1.
- 如果另外一条路如蓝色箭头，走到(2,2)的时侯需要花费是3，这条蓝色的路径有必要继续枚举下去么？这个蓝色路线无论后面怎么走，都不会走出比黑色路径更小的花费。

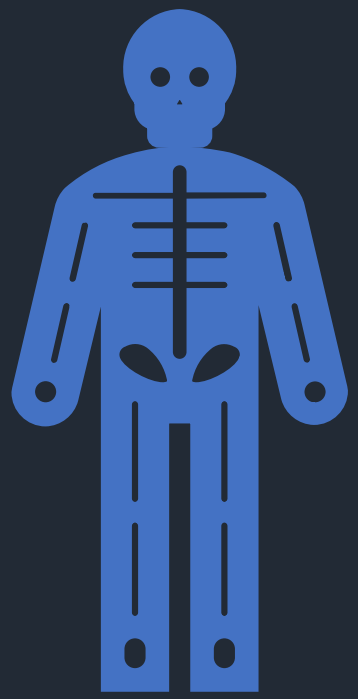


记忆化搜索剪枝

- 使用gridmoney[][]数组来保存走到每一个格子当前的最小花费的金币money，这样，如果下一次如果到达该位置(row,col)的money大于该位置已有的gridmoney[row][col]，说明前面用更小的代价来过这里，这次不需要再走下去了。如果找最大，就不能类似的方法剪枝了：由于上下左右4法方向，所以可能往回绕着走，一条路线走到一个位置的值比另外一个路线走到该位置的值小，并不能证明这条路走到底一定小。
- 注意， gridmoney[][]数组初始化每一个元素为INT_MAX
- 在DFS每次进来之后首先判断：

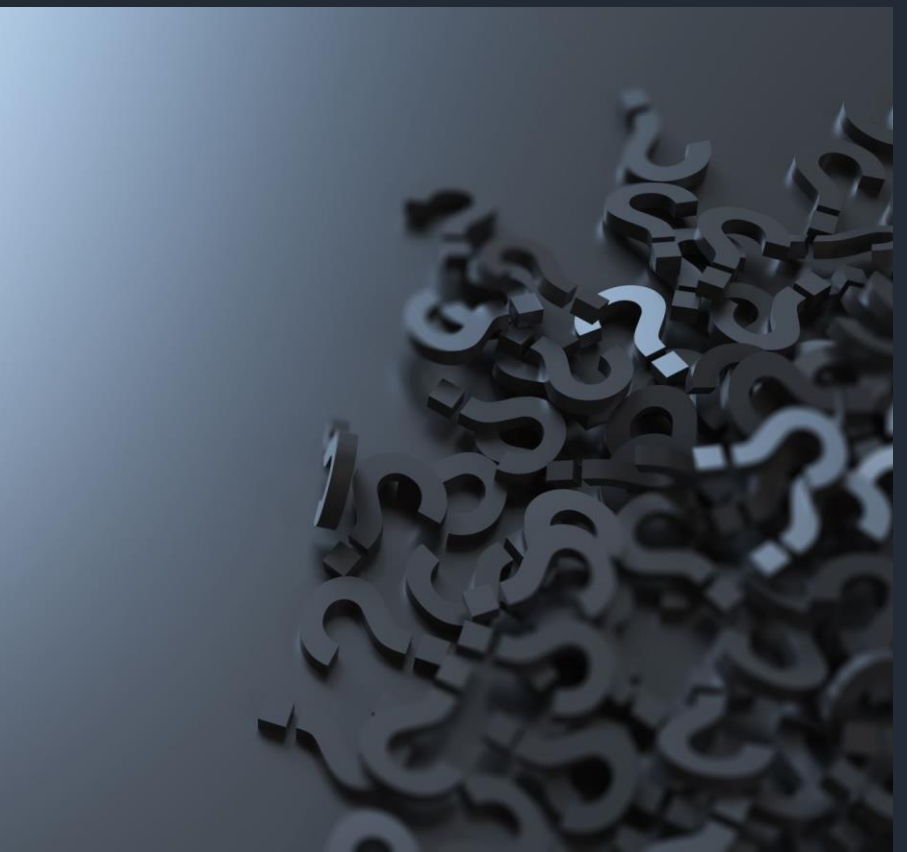
  if (money>=gridmoney[row][col]) return;

- visited数组在这里就不需要了，因为每一次去判断gridmoney[row][col]，如果访问过了当前row，col，同一个路径再次访问这个位置，肯定大于等于上一次的访问。相当于合并了visited数组的功能。



```
void dfs(int row,int col,int money,bool canmagic)
{
        if (money>=gridmoney[row][col]) return;
        //到达该位置的moeny大于该位置已有的gridmoney，说明前面用更小的代价来过这里，这次不需要再走下去了
        gridmoney[row][col]=money;
        if (row==m && col==m) {        if (money<r) r=money;        return;} //到底了
        int newrow,newcol;
        for (int i=0;i<4;i++) //try 4 directions.        {
                newrow=row+rowflag[i]; newcol=col+colflag[i];
                if (newrow<1 ||newrow>m ||newcol<1 ||newcol>m)        continue;//边界判断
                if (grid[newrow][newcol]==0) {  //no color
                        if (canmagic) {
                                grid[newrow][newcol]=grid[row][col];
                                dfs(newrow,newcol,money+2,false);
                                grid[newrow][newcol]=0; //回溯
                        }
                }
                else {
                 if (grid[newrow][newcol]!=grid[row][col])       dfs(newrow,newcol,money+1,true); //different color+1
                 else dfs(newrow,newcol,money,true); //same color no change
                }
        }
        return ;
}
```

学到了什么?



- 根据时间函数选择合适的算法
- 优化搜索：避免重复搜索，避免不必要的搜索
- 剪枝避免一些不必要的路径
- 记忆化搜索剪枝更高级的剪枝
- 注意：不是所有的情况都可以剪枝，一定要具体问题具体分析。

Tips

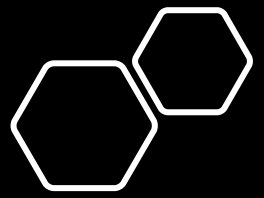
- 交换数组元素
  - swap(a[i],a[j]);
- 现阶段最容易出现的错误：
  - 初始化，比如计数器初始化，数组初始化，bool变量初始化
  - == 写成=
  - 数组从0号元素开始使用还是1号开始要和循环配合
  - 数组越界

下节课例题预习

- 尝试理解题目，思考可能的算法，不需要做
- USACO training Section 1.5 PROB Arithmetic Progressions
- USACO training Section 1.6 PROB Number Triangles
- USACO 2019 December Contest, Bronze Livestock Lineup

作业1 洛谷P1215

- USACO training Section 1.5 PROB Mother's Milk DFS
  - 三个桶开始 C 满的其他空的，互相倒，问最后 A 空的有多少情况
  - 提示：DFS函数内部的相邻节点就是按照规则尝试“倒牛奶”后新的状态，也就是按照规则修改3个桶内已经有多少牛奶了（这就是一个状态，可以参数传递给下一次也可以修改全局变量让下一个函数从全局变量取出新的状态）。
  - 提示：关键是剪枝找重复，怎么标记一个状态是否重复？类似visited数组，但是一维数组不够，需要三维数组，因为同时要标记3个桶的容量。当3个桶的容量都出现重复的时侯就说明这个状态出现过了，这条路不用再走了。
  - 注意：不需要手工回溯，因为和迷宫类似，每一个出现过的状态只需要访问一次。



挑战作业2

提示：参考课堂例题。

这里非常类似，尝试选择每一个没有选过的数，并传递当前选择的和。可以2个参数：除了当前选择的和是多少，还需要多一个参数表示当前选择了几个边。复杂的是在dfs函数里面，如果当前的和==总长度/3，边就+1；如果已经选了3个边说明ok了。

- 等边三角形

  输入一个整数n(<10),后面一行n个空格隔开的数字，表示n个木棒长度。

  输出：如果这些木棒全部用上可以拼成一个等边三角形，输出 “OK” 否则输出“NO”

  示例输入：

  5

  1 2 3 4 5

  示例输出

  OK

  说明：这个例子种，5个木棒可以组成边长5的等边三角形，用长度5的单独一个边，长度2和3的合起来做成一个边，长度1和4的合起来组成另外一个边。

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

https://www.35tang.com

https://www.三五堂.com

扫码关注公众号

添加辅导老师