枚举与递归回溯法

PKU EECS zld3794955

2017 年 7 月 18 日

前言

这是今天的第二份课件~

做到这份课件的时候，听说去年省夏令营有的出题人题目一小时被穿，身败名裂，作为出题人的我感到十分害怕。

这份课件我们讲一讲枚举法，除此之外我们要讲枚举的一种重要方法——递归回溯。

递归回溯这里只讲基本框架和简单应用，其优化剪枝在明天n+e大佬的课上会详细提到。

另外，课程安排中的分支限界法我实在没法分清其与BFS有什么区别，所以我也丢给明天的课了。

枚举

含义很简单，即将所有解一一列出来，一一判断其可行性和最优性。

这也应该是OI界求解问题最基本的算法了。

但是枚举也不是那么naive的，有时需要通过限制枚举范围或者寻找合适的枚举量来解决问题，而且绝大多数时候，枚举都需要与别的算法相结合。

我们下面来些简单的例子。

一道入门题

n个人参加一次有m道判断题的考试，恰有k_1个人考了满分，恰有k_2个人考了0分。

现在已知这场考试中每个人的对每道题提交的答案，试给出该次考试一组可能的正确答案，或说明得分可能有问题。

$1 \leq n, m \leq 50$，$0 \leq k_1 + k_2 \leq n$。

The solution

如果$k_1 + k_2 = 0$，那么我们可以任意枚举答案（比如random），显然，枚举出$2n + 1$个不同的答案之后，一定会有一个答案符合题意（想想为什么？）。

否则，不妨设$k_1 > 0$，那么一定存在一个人的答案是全部正确的，我们直接枚举这个人是谁，并且以他的答案为标准检查一下其他人的得分情况并验证即可。

无解的情况也很简单，大家自己想想。

一道经典例题

已知一个$n \times n$的01矩阵，现在你需要从中找到一个最大的子矩阵，该子矩阵内的元素全部都是1。

Easy：$1 \leq n \leq 200$

Hard：$1 \leq n \leq 1000$

The solution

我们考虑可以枚举矩阵的哪些量。

Easy难度的话，我们可以用l枚举矩阵的上边界在哪一行，r枚举矩阵的下边界在哪一行，这里有$O(n^2)$的时间复杂度，接下来我们就要找最长的连续列，其中每列的l到r行都是1。

判断矩阵每列的$[l, r]$行范围内是否全1可以使用预处理或者在枚举r的过程中进行维护。

接下来我们再$O(n)$扫一遍列，即可知道上边界为l、下边界为r的全1矩阵面积的最大值，将所有(l, r) 的结果算出来，就可以得到答案。

时间复杂度$O(n^3)$。

The solution

Hard难度的话，我们只要枚举矩形的下边界r。

枚举r时，我们同时对每列求出最大的h，使得该列的第$r-h+1$行到第r行的数字均为1，这个可以用$O(n^2)$的总时间复杂度维护。

接下来，我们就相当于要寻找一个列的区间，使得区间的列数$\times$区间内h的最小值的结果最大，这个结果即为最大的下边界为r的全1子矩阵的面积。

具体求法已经在刚刚的贪心课件中讲过了，单次计算的时间复杂度为$O(n)$，每枚举一次r就要计算一次，故总时间复杂度为$O(n^2)$。

递归回溯法

以上都是一些非常简单的枚举的例子。

但实际上，我们可能要枚举的东西相当的复杂，层次不清晰。

这时候我们就有必要引入我们暴力枚举的大杀器——递归回溯法，又称深度优先搜索（DFS）。

回溯法是什么

回溯法是一种普遍的搜索方法，其基本思想就是不断尝试，如果当前尝试下去的结果不符合题意（比如不可行或者答案不够优）那么便撤销当前尝试，并尝试下一个选择。

比如我们要求有输出所有长度为n的，不存在两个1相邻的01序列$x_1, x_2, \ldots, x_n$，那么我们便可以用回溯法，其最基本的过程如下：

1、给序列的当前位置pos填上0。

2、如果未填到最后一个数，那么尝试填下一个位置$pos + 1$，否则进行检验，若符合题意就输出。

3、在第1步中，我们给当前位置尝试着填了0，如果该状态已尝试完成，那么我们试着给当前位置填1，再执行一次步骤2，之后回退。

递归回溯法

递归是实现回溯最简单的一种方式，其基本框架如下（我们以上一页的问题为例）：

```
int n,x[100];
void tryy(int pos)
{
    if(pos>n)
    {
        if(/*序列符合条件*/)
            {};//输出序列x
        //返回
    }
    x[pos]=0; tryy(pos+1);
    x[pos]=1; tryy(pos+1);
}
```

调用的时候直接写tryy(1);一句，表示从第一个位置开始试着填写数字即可。

思考与讨论

如何枚举输出所有不存在两个1相邻的$n \times n$的01矩阵？

基本框架类似，大家可以自己想想，我也会在课上用txt写出来。

全排列

假如我们要输出1到n的全排列。

考虑排列的第一个位置——选什么都行。

当我们考虑第i个位置的时候，前面的位置上的数就不能再重复出现了，我们用一个数组$used$来记录，当我们选择了某个数x，便令$used[x] = true$，在回溯撤销的时候，x不再使用，令$used[x] = false$即可。

这种过程可以很方便地用递归回溯实现，其伪代码在下一页。

递归回溯全排列伪代码

```
int n,x[N]; //当前排列
bool used[N];
void tryy(int pos)
{
    if(pos>n)
        输出一个排列并返回
    for( 在1~n内枚举i )
        if( used[i]==false )
        {
            x[pos]=i;
            used[i]=true;
            tryy(pos+1);
            used[i]=false;
        }
}
int main()
{
    tryy(1);
}
```

遍历树

对一棵树的遍历，我们一般从根节点开始，DFS一棵树的基本框架如下：

```
void tryy(int now)
{
    输出now，表示遍历到了这个节点
    for(next in now的子结点集)
        tryy(next);
}
```

初始时调用tryy(root节点编号，一般取为1);即可。

因为树递归到叶节点时，子节点集为空，无法继续递归，所以树的搜索回溯一般会方便一些。

统计子树信息

假设我们要统计一下以当前节点为根的子树内一共有多少个节点，那么对伪代码稍做修改即可，如下：

```
int size[N];
void tryy(int now)
{
    size[now]=1; //算上自己
    for(next in now的子结点集)
    {
        tryy(next);
        size[now]+=size[next];
    }
}
```

类似，我们还可以求出树的高度，也可以对昨天所讲的Trie进行处理，求出每个前缀出现在多少个字符串中（对每个串s，在其对应的节点p上记录一下串s本身在Trie中出现的次数$num[p]$即可，其它类似）。

图的DFS遍历

在树的定义中，保证按不同子节点下去遍历到的节点是不重复的，但是图并不能保证这一点，因此我们需要打一个标记来表示这个点是否走到过。

同时，因为一般的图中并没有太特殊的点，我们每次随意选择一个未遍历过的节点开始遍历即可。

其基本框架我写在了下一页……

伪代码

```
int n; //节点数
bool vis[N]; //是否遍历过
void tryy(int now)
{
    访问now
    vis[now]=true;
    for(next in now可走到的节点集)
        if(!vis[next])
            tryy(next);
}
int main()
{
    for(int i=1;i<=n;++i)
        if(!vis[i])
            tryy(i);
}
```

最后一个例子

我们再来一个综合性的例子——找到并输出一张图中从u到v的所有不经过重复的点的路径。

同全排列一样，我们用一个数组$inpath$来表示某个点当前是否在搜索路径中。

在这个问题中，u必定为起点，所以我们从u开始遍历起，全过程伪代码见下一页。

伪代码

```
int n,way[N]; //节点数、当前路径
bool inpath[N];
void tryy(int now,int pos)
{
    way[pos]=now;
    if( now即为目标节点v )
        输出路径并返回
    inpath[now]=true;
    for(next in now可走到的节点集)
        if(!inpath[next])
            tryy(next); //递归
    inpath[now]=false; //回溯
}
int main()
{
    tryy(u,1);
}
```

后记

讲完啦~

大家消化一下。

Questions are welcomed!