搜索1-枚举、DFS

PKU EECS zld3794955

2018 年 2 月 4 日

前言

前言

这是老师发给我的普及组集训营的选手情况：

“大部分都是普及组省二等奖和省一等奖线附近；他们学习过深搜、广搜的基本入门题；做过一些简单的模拟题；做过递推的入门题；动态规划的最长上升自序列基础题；希望能巩固基础经典算法和基础数据结构，学习提高组经典算法和数据结构的灵活运用能力，开拓下解题技巧。巩固提高下编程代码能力。”

前言

这是老师发给我的普及组集训营的选手情况：

“大部分都是普及组省二等奖和省一等奖线附近；他们学习过深搜、广搜的基本入门题；做过一些简单的模拟题；做过递推的入门题；动态规划的最长上升自序列基础题；希望能巩固基础经典算法和基础数据结构，学习提高组经典算法和数据结构的灵活运用能力，开拓下解题技巧。巩固提高下编程代码能力。”

但我并不觉得你们所有人都是这样的……

前言

这是老师发给我的普及组集训营的选手情况：

“大部分都是普及组省二等奖和省一等奖线附近；他们学习过深搜、广搜的基本入门题；做过一些简单的模拟题；做过递推的入门题；动态规划的最长上升自序列基础题；希望能巩固基础经典算法和基础数据结构，学习提高组经典算法和数据结构的灵活运用能力，开拓下解题技巧。巩固提高下编程代码能力。”

但我并不觉得你们所有人都是这样的……

所以我先从0开始讲，如果一个部分你们大部分人都会了的话，那么就跳过，如果最后有剩时间的话，可以提前下课，也可以给大家介绍下C++整型与浮点型的知识。

今天的内容

今天的内容

1. 枚举法
2. 递归回溯
3. DFS（深度优先搜索）

枚举法

枚举法

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。

枚举法

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。

枚举法

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。

我们先从小学奥数的一道题开始讲讲枚举法。

填数游戏

填数游戏

下图中ABC分别表示一个0到9之间的数码，且ABC对应的数码不一样，确定ABC的所有可能值：

$$\begin{array}{r} AB \\ *\ \ A \\ \hline CCC \end{array}$$

填数游戏

下图中ABC分别表示一个0到9之间的数码，且ABC对应的数码不一样，确定ABC的所有可能值：

$$\begin{array}{r} AB \\ * \quad A \\ \hline CCC \end{array}$$

做法很简单，直接枚举ABC三个数码所对应的值即可。

（代码用记事本现场写）

一个小练习

一个小练习

编写枚举法程序，确定ABCD对应的数码，使得下面两个竖式同时成立：

```
   AA      A
*  AB     *B
------    --
CCDD      CD
```

NOIP2014比例简化

NOIP2014比例简化

给定正整数A, B, L，试确定一个比例$A' : B'$，使得A', B'互质、$A', B' \leq L$、$A'/B' - A/B \geq 0$且尽可能小。

$1 \leq A, B \leq 1000000, 1 \leq L \leq 100, A/B \leq L$

NOIP2014比例简化

给定正整数A, B, L，试确定一个比例$A' : B'$，使得A', B'互质、$A', B' \leq L$、$A'/B' - A/B \geq 0$且尽可能小。

$1 \leq A, B \leq 1000000, 1 \leq L \leq 100, A/B \leq L$

这题初看是一道不太好做的数学题，但其实做法很简单，由于L不大，因此能满足分子分母都不超过L的分数不超过$L^2 \leq 10000$个，直接枚举所有这样的分数并检查是否符合题意即可，分数的加减法和比大小想必大家都会。

NOIP2014比例简化

给定正整数A, B, L，试确定一个比例$A' : B'$，使得A', B'互质、$A', B' \leq L$、$A'/B' - A/B \geq 0$且尽可能小。

$1 \leq A, B \leq 1000000, 1 \leq L \leq 100, A/B \leq L$

这题初看是一道不太好做的数学题，但其实做法很简单，由于L不大，因此能满足分子分母都不超过L的分数不超过$L^2 \leq 10000$个，直接枚举所有这样的分数并检查是否符合题意即可，分数的加减法和比大小想必大家都会。

可以看到，如果答案范围不大，那么枚举法相对于其它方法来说不失为一种简单易用的好方法。

再来一道经典题

再来一道经典题

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连续子矩阵，该子矩阵内的元素全部都是1。

The solution

The solution

显然，这题我们可以使用枚举来完成。

最简单的想法就是每次枚举矩形的左上角，再枚举矩形的右下角，然后判断该连续子矩阵范围内元素是否全1，这样做是$O(n^6)$的。

The solution

显然，这题我们可以使用枚举来完成。

最简单的想法就是每次枚举矩形的左上角，再枚举矩形的右下角，然后判断该连续子矩阵范围内元素是否全1，这样做是$O(n^6)$的。

通过一些巧妙的改进，我们可以很轻松地降低每次判断的代价，将该枚举算法的时间复杂度降为$O(n^4)$（不会的话我就简单介绍下）。

The solution

显然，这题我们可以使用枚举来完成。

最简单的想法就是每次枚举矩形的左上角，再枚举矩形的右下角，然后判断该连续子矩阵范围内元素是否全1，这样做是$O(n^6)$的。

通过一些巧妙的改进，我们可以很轻松地降低每次判断的代价，将该枚举算法的时间复杂度降为$O(n^4)$（不会的话我就简单介绍下）。

但是这个算法的主要问题依然在于枚举的低效，我们需要改变枚举的方式。

do it more efficiently

do it more efficiently

分析一下，为了确定一个连续子矩阵，我们需要确定其四条边的位置，为了降低枚举的时间复杂度，我们不能再枚举全部的四条边了。

do it more efficiently

分析一下，为了确定一个连续子矩阵，我们需要确定其四条边的位置，为了降低枚举的时间复杂度，我们不能再枚举全部的四条边了。

枚举矩形的上下两条边所在的位置，那么这时候矩形待确定的就只有左右两边，而且这时候对于每一列，我们都可以记录一下其在上下两条边范围内的数是否全部为1，从而确定该列是否能包含在矩阵中。

do it more efficiently

分析一下，为了确定一个连续子矩阵，我们需要确定其四条边的位置，为了降低枚举的时间复杂度，我们不能再枚举全部的四条边了。

枚举矩形的上下两条边所在的位置，那么这时候矩形待确定的就只有左右两边，而且这时候对于每一列，我们都可以记录一下其在上下两条边范围内的数是否全部为1，从而确定该列是否能包含在矩阵中。

记c_i为当前情况下，第i列能否包含在矩形中，0表示不行1表示可以，则每个连续的c_i全1的段都对应一个矩形，即我们将找最大矩形的问题转化为找最大连续全1段的问题，而这个问题不需要分别枚举左右两列，只需要扫一遍该序列即可，于是我们就将处理列的时间复杂度由$O(n^2)$压缩到了$O(n)$，从而总的时间复杂度就变为了$O(n^3)$。

do it more efficiently

分析一下，为了确定一个连续子矩阵，我们需要确定其四条边的位置，为了降低枚举的时间复杂度，我们不能再枚举全部的四条边了。

枚举矩形的上下两条边所在的位置，那么这时候矩形待确定的就只有左右两边，而且这时候对于每一列，我们都可以记录一下其在上下两条边范围内的数是否全部为1，从而确定该列是否能包含在矩阵中。

记c_i为当前情况下，第i列能否包含在矩形中，0表示不行1表示可以，则每个连续的c_i全1的段都对应一个矩形，即我们将找最大矩形的问题转化为找最大连续全1段的问题，而这个问题不需要分别枚举左右两列，只需要扫一遍该序列即可，于是我们就将处理列的时间复杂度由$O(n^2)$压缩到了$O(n)$，从而总的时间复杂度就变为了$O(n^3)$。

（可能的写在记事本上的伪代码）

NOIP2016魔法阵

NOIP2016魔法阵

有m样魔法物品，第i样的魔法值为一个正整数$x_i \leq n$，一个魔法阵由编号a, b, c, d的四样物品（分别称为A物品、B物品、C物品、D物品）构成，且满足：

1、$x_a < x_b < x_c < x_d$。

2、$2(x_d - x_c) = x_b - x_a < (x_c - x_b)/3$。

你的任务就是计算每样物品作为某个魔法阵的ABCD物品分别出现了多少次。

$1 \leq n \leq 15000, 1 \leq m \leq 40000$

分析

分析

首先一个魔法阵与物品的编号无关，且不会出现两种魔法数值相同的物品，又魔法值不会超过15000，所以可以用数组t[]记录每种魔法值的物品有几个，并在数组t上分析问题。

分析

首先一个魔法阵与物品的编号无关，且不会出现两种魔法数值相同的物品，又魔法值不会超过15000，所以可以用数组t[]记录每种魔法值的物品有几个，并在数组t上分析问题。

直接枚举四种物品的魔法值的复杂度是$O(n^4)$，不可接受，考虑到题目中有一个等式约束，枚举量可以降为$O(n^3)$，但也不可接受。

分析

首先一个魔法阵与物品的编号无关，且不会出现两种魔法数值相同的物品，又魔法值不会超过15000，所以可以用数组t[]记录每种魔法值的物品有几个，并在数组t上分析问题。

直接枚举四种物品的魔法值的复杂度是$O(n^4)$，不可接受，考虑到题目中有一个等式约束，枚举量可以降为$O(n^3)$，但也不可接受。

分析题目中的约束条件，记$x_d - x_c = i$，那么$x_b - x_a = 2i$，且$(x_c - x_b) > 6i$，故$x_d - x_a > 9i$，因此我们可以确定$x_b - x_a$至多有$n/9$种不同的取值，考虑枚举这个取值，在此基础上进行暴力枚举，时间复杂度为$O(n/9) * O(n^2)$，比原来的做法在常数上稍微好些，但仍然不可接受，我们需要在枚举i之后，有一种方式来优化后面$O(n^2)$的枚举。

接着分析

接着分析

假设我们知道了当前$x_d - x_c = i$，那么求符合条件的魔法阵数量的时间复杂度最好能降到$O(n)$，而不是暴力枚举的$O(n^2)$。

接着分析

假设我们知道了当前$x_d - x_c = i$，那么求符合条件的魔法阵数量的时间复杂度最好能降到$O(n)$，而不是暴力枚举的$O(n^2)$。

枚举B物品的魔法值，那么此时A物品的魔法值也随之确定，而C物品的魔法值要比B物品至少高$6i+1$，从$6i+1$开始均可，因此我们考虑使用部分和来进行优化。

接着分析

假设我们知道了当前$x_d - x_c = i$，那么求符合条件的魔法阵数量的时间复杂度最好能降到$O(n)$，而不是暴力枚举的$O(n^2)$。

枚举B物品的魔法值，那么此时A物品的魔法值也随之确定，而C物品的魔法值要比B物品至少高$6i+1$，从$6i+1$开始均可，因此我们考虑使用部分和来进行优化。

具体地来说，记$r[k]$表示，在当前i值的情况下，只取C, D物品，且C物品的魔法值$\geq k$的方案数（注意到C的魔法值确定则D的也确定了），那么这个值可以按下标从大到小$O(n)$内求出。

接着分析

假设我们知道了当前$x_d - x_c = i$，那么求符合条件的魔法阵数量的时间复杂度最好能降到$O(n)$，而不是暴力枚举的$O(n^2)$。

枚举B物品的魔法值，那么此时A物品的魔法值也随之确定，而C物品的魔法值要比B物品至少高$6i+1$，从$6i+1$开始均可，因此我们考虑使用部分和来进行优化。

具体地来说，记$r[k]$表示，在当前i值的情况下，只取C, D物品，且C物品的魔法值$\geq k$的方案数（注意到C的魔法值确定则D的也确定了），那么这个值可以按下标从大到小$O(n)$内求出。

接下来枚举B物品的魔法值X_B，则X_B在当前i下作为B物品出现的次数即为$t[X_B - 2i] \times r[X_B + 6i + 1]$，$X_A = X_B - 2i$作为$A$物品出现的次数即为$t[X_B] \times r[X_B + 6i + 1]$。

最后的分析

最后的分析

用类似于上述的方法，我们可以求出每个魔法值作为C物品和D物品出现的次数，同样也是$O(n)$的。

最后的分析

用类似于上述的方法，我们可以求出每个魔法值作为C物品和D物品出现的次数，同样也是$O(n)$的。

算法的整个过程即为用$O(n/9)$枚举$x_d - x_c = i$值，并对每个i值用$O(n)$时间进行统计，总时间复杂度为$O(n^2/9)$，足以通过（只要常数不太差的话）。

最后的分析

用类似于上述的方法，我们可以求出每个魔法值作为C物品和D物品出现的次数，同样也是$O(n)$的。

算法的整个过程即为用$O(n/9)$枚举$x_d - x_c = i$值，并对每个i值用$O(n)$时间进行统计，总时间复杂度为$O(n^2/9)$，足以通过（只要常数不太差的话）。

可以看到，改变枚举对象和分析枚举范围可以起到大大优化枚举的作用，而算法最后一步的统计部分是非常经典的优化序列中平方复杂度枚举的一种方式。

枚举法的特点

枚举法的特点

运用枚举法寻找最优解或可行解固然简单，但是有以下两个限制：

枚举法的特点

运用枚举法寻找最优解或可行解固然简单，但是有以下两个限制：

1、要保证最优解或者全部的可行解都能被枚举到。

枚举法的特点

运用枚举法寻找最优解或可行解固然简单，但是有以下两个限制：

1、要保证最优解或者全部的可行解都能被枚举到。

2、枚举量不能过大。

枚举法的特点

运用枚举法寻找最优解或可行解固然简单，但是有以下两个限制：

1、要保证最优解或者全部的可行解都能被枚举到。

2、枚举量不能过大。

为了优化枚举，我们可以采用改变/减少枚举对象（优化枚举量）、调整枚举顺序（优化每种情况的处理时间）等方式，在上面的例子中大家应该很有体会。

递归回溯

递归回溯

显然，前面所提到的枚举方法要求被枚举的对象有比较清楚的结构和确定的层次，然而对于一般的枚举，仅用for循环有点儿不够用……

递归回溯

显然，前面所提到的枚举方法要求被枚举的对象有比较清楚的结构和确定的层次，然而对于一般的枚举，仅用for循环有点儿不够用……

所以递归回溯法就是用于枚举的通用方法（想必大家应该都知道递归），它的思想是从某个状态开始，先朝某个方向尝试一步到达新状态并搜索新状态，当搜索完新状态时，撤销这一步，并朝另一个方向尝试，直到其中某一步找到了我们所要的结果，或者遍历完了所有的可能性。

递归回溯

显然，前面所提到的枚举方法要求被枚举的对象有比较清楚的结构和确定的层次，然而对于一般的枚举，仅用for循环有点儿不够用……

所以递归回溯法就是用于枚举的通用方法（想必大家应该都知道递归），它的思想是从某个状态开始，先朝某个方向尝试一步到达新状态并搜索新状态，当搜索完新状态时，撤销这一步，并朝另一个方向尝试，直到其中某一步找到了我们所要的结果，或者遍历完了所有的可能性。

直观上，递归回溯就是选一个方向，不停地试下去，直到检查所有的可能性确认这样做不行了，才会换一个方向。

基本框架

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

3、沿该方向走一步，改变状态。（尝试）

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

3、沿该方向走一步，改变状态。（尝试）

4、在新的状态上执行这个流程。（递归）

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

3、沿该方向走一步，改变状态。（尝试）

4、在新的状态上执行这个流程。（递归）

5、撤销2中的改变（回溯）

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

3、沿该方向走一步，改变状态。（尝试）

4、在新的状态上执行这个流程。（递归）

5、撤销2中的改变（回溯）

6、回到2。

基本框架

大致步骤如下：

1、判断当前状态是否为所要的状态，视情况进行处理或回退。

2、在当前状态上枚举所有合法的方向，取某一个未尝试过的方向，如果没有就退出。

3、沿该方向走一步，改变状态。（尝试）

4、在新的状态上执行这个流程。（递归）

5、撤销2中的改变（回溯）

6、回到2。

伪代码的话我会写在记事本上，大家可以在脑子里记一记。

经典的例子：全排列

经典的例子：全排列

输入正整数n，输出1到n这n个数的全部$n!$个排列。

运用递归回溯

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

1、当前这n个数是否填完？若已填完，则输出序列并回退，否则转到下一步。

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

1、当前这n个数是否填完？若已填完，则输出序列并回退，否则转到下一步。

2、依次取i为1到n，如果$used[i]$为假，即i未在序列中被填过，那么就在序列当前待填数的位置填上i，并且令$used[i] = true$（尝试），否则就尝试下一个i。

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

1、当前这n个数是否填完？若已填完，则输出序列并回退，否则转到下一步。

2、依次取i为1到n，如果$used[i]$为假，即i未在序列中被填过，那么就在序列当前待填数的位置填上i，并且令$used[i] = true$（尝试），否则就尝试下一个i。

3、在填好i的序列的基础上进行整个过程。（递归）

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

1、当前这n个数是否填完？若已填完，则输出序列并回退，否则转到下一步。

2、依次取i为1到n，如果$used[i]$为假，即i未在序列中被填过，那么就在序列当前待填数的位置填上i，并且令$used[i] = true$（尝试），否则就尝试下一个i。

3、在填好i的序列的基础上进行整个过程。（递归）

4、令$used[i] = false$，清除当前序列填入的i（回溯），找到下一个i，转到2。

运用递归回溯

从一个空的序列开始，每个状态记录已填好的序列，同时，记录一个数组$used[i]$，表示当前状态中数字i是否已经被填过，那么整个回溯法的流程如下：

1、当前这n个数是否填完？若已填完，则输出序列并回退，否则转到下一步。

2、依次取i为1到n，如果$used[i]$为假，即i未在序列中被填过，那么就在序列当前待填数的位置填上i，并且令$used[i] = true$（尝试），否则就尝试下一个i。

3、在填好i的序列的基础上进行整个过程。（递归）

4、令$used[i] = false$，清除当前序列填入的i（回溯），找到下一个i，转到2。

具体代码的实现呢我再在记事本上写一个给大家参考参考。

另一道题

另一道题

给定n个城市，其中某些城市之间有直接的道路相连，现在给定两个城市S和T，你的任务就是找到所有S到T的，不经过重复城市的路径。

运用递归回溯

运用递归回溯

记录状态为从S开始到当前城市的一条路径，再记录一个数组$in[i]$，表示编号为i的城市是否在当前的搜索路径中，则回溯法的流程如下：

运用递归回溯

记录状态为从S开始到当前城市的一条路径，再记录一个数组$in[i]$，表示编号为i的城市是否在当前的搜索路径中，则回溯法的流程如下：

1、判断当前城市i是否为T，如果是，那么就输出当前路径并返回，否则继续进行该流程。

2、令当前城市i的in值为$true$

3、依此取当前城市有直接道路相连的所有其它城市j，如果$in[j]$为$false$，那么就在路径末尾加入城市j，并且从城市j和新路径的基础上执行整个过程（递归），否则就尝试下一个城市

4、令当前城市i的in值为$false$

运用递归回溯

记录状态为从S开始到当前城市的一条路径，再记录一个数组$in[i]$，表示编号为i的城市是否在当前的搜索路径中，则回溯法的流程如下：

1、判断当前城市i是否为T，如果是，那么就输出当前路径并返回，否则继续进行该流程。

2、令当前城市i的in值为$true$

3、依此取当前城市有直接道路相连的所有其它城市j，如果$in[j]$为$false$，那么就在路径末尾加入城市j，并且从城市j和新路径的基础上执行整个过程（递归），否则就尝试下一个城市

4、令当前城市i的in值为$false$

初始时，直接从状态S开始执行整个过程即可。

作为练习，大家自己写一写这段框架对应的代码。

一道练习题

一道练习题

在一个$n \times m$的矩形里填入c种颜色，要求相邻两个格子的颜色不一样，求所有染色方案。

一道练习题

在一个$n \times m$的矩形里填入c种颜色，要求相邻两个格子的颜色不一样，求所有染色方案。

Hint：从左上到右下填颜色，保证每个填入的颜色与已填入的颜色不冲突即可。

DFS（深度优先搜索）

DFS（深度优先搜索）

这玩意儿其实跟递归回溯好像是一样的……所以实现自然也就是递归回溯那套玩意儿。

之所以叫这个名字，是因为在这种搜索方式中状态会不断往状态树的深层走，直到试完了这棵搜索树下全部的可能性后才会往回走。

相对的，明天我们会讲到一个BFS（宽度优先搜索）的概念，它会优先遍历搜索树中深度较浅的节点。

24点游戏

24点游戏

给定4个1到13之间的正整数，你需要用这4个数（每个数只能使用一次）和加减乘除4个运算符（每种运算符次数不限）得出24这个结果，输出所有方案。

你要思考一种尽可能合理的搜索状态和搜索方式，使得代码实现尽量方便简单。

做法

做法

可以看出，24点游戏中，每次运算相当于从当前拥有的数中取出两个数，进行某种运算，再将所得结果放回的过程，比如：4 4 8 3这4个数，取出8 4做除法，所得结果2，那么剩下的数即为4 3 2。

做法

可以看出，24点游戏中，每次运算相当于从当前拥有的数中取出两个数，进行某种运算，再将所得结果放回的过程，比如：4 4 8 3这4个数，取出8 4做除法，所得结果2，那么剩下的数即为4 3 2。

所以，我们在我们当前拥有的数上进行搜索，每步选择都是选择两个数做某种运算，将当前的数的个数-1，而目标状态显然就是最后只剩下一个24。

做法

可以看出，24点游戏中，每次运算相当于从当前拥有的数中取出两个数，进行某种运算，再将所得结果放回的过程，比如：4 4 8 3这4个数，取出8 4做除法，所得结果2，那么剩下的数即为4 3 2。

所以，我们在我们当前拥有的数上进行搜索，每步选择都是选择两个数做某种运算，将当前的数的个数-1，而目标状态显然就是最后只剩下一个24。

为了还原出算24点的过程，在DFS时记录一下每步尝试都分别做了什么操作即可，唯一要注意的是中途结果有可能出现分数。

递归回溯框架和代码就留给大家自行完成了。