Introduction to Dynamic Programming

You Siki

前言

- 初识DP、背包DP、区间DP、树形DP、状压DP、数位DP

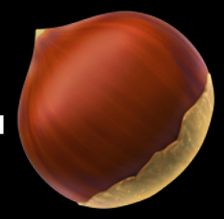
- 大概就是这么多内容，希望大家观影体验良好😊

- 感谢舍友安利的OI Wiki项目，提供了教学大纲🤦

- 如果讲解太快无法理解，请不要害羞，向我报告🙋

- 动态规划适于解决子问题重叠的情况。
- 对于一项子问题，求得最优解后不再重复计算。
- 入门时的一种理解是给搜索加上了记忆化。
- 后期实现一般采用更快速的刷表法。

- 我总结出来的一般步骤——
- 寻找子问题
- 表示子问题
- 写出转移方程
- 确定边界答案

栗子 

- 举个大家熟悉的栗子——汉诺塔问题

- 举个栗子——钢条切割
- 长度为i的钢条，收益为pi元。
- 现在有一个长度为n的钢条，请问经过若干次切割后，总收益最大是多少?

- 举个例子——数字三角形
- 从顶上走到最下层，使得经过的数字之和最大。

- 举个栗子——DAG上的最长简单路径
- 给出一张有向无环图，求最长简单路径的长度。

- 举个栗子——最长上升子序列
- 给出一个序列{ai}，求最长上升子序列的长度。

- 举个栗子——最长公共子序列
- 给出两个序列{ai}和{bi}，求最长公共子序列的长度。

背包DP

01背包

- 有n个物品，第i个物品的体积为Wi，价值为Vi。
- 选出若干个物品放入体积为m的背包内，求最大总价值。

完全背包

- 有n种物品，每种无限个，第i个物品体积为Wi，价值为Vi。
- 选出若干个物品放入体积为m的背包内，求最大总价值。

多重背包

- 有n种物品，每种Ci个，第i个物品体积为Wi，价值为Vi。
- 选出若干个物品放入体积为m的背包内，求最大总价值。

区间DP

- 指一类动态规划问题。
- 特点是子问题用区间[l,r]表示。
- 有时候还会有其他的状态记录。

石子合并

- 其实我也不确定搜索这四个字会出来什么题目。
- 有n堆石子排成一行，第i堆有Ai个。
- 每次合并相邻两堆的代价是其中的石子个数之和。
- 求把这n堆合并为1堆的最小总代价。

石子合并+

- 如果这n堆石子不是排成一行，而是围城一圈。

括号序列

- http://poj.org/problem?id=2955

- 给出一个包含()[]的序列，求最长合法子序列的长度。

加分二叉树

- n个结点的二叉树中序遍历为n的自然排列。
- 每个结点有一个分数值——Di。
- 子树的分数=左子树分数x右子树分数+子树根分数值
- 求最大的设计方案使得整棵树的分数最大。

乘法谜题

- http://poj.org/problem?id=1651

- 给出一行n个数，每次拿走一个数，代价是这个数和左右两数的乘积。

- 求拿走中间n-2个数的最小代价。

树形DP

- 首先，大家都知道“树”吧。
- 这个树不是那个🌲。
- 树形DP依托于树形结构，一般按子树表示子问题。

没有上司的舞会

- 公司内的n名员工构成树形关系。
- 如果一个人的上司参加舞会，那么这个人就不会参加舞会。
- 每个人有一个欢乐值Ri。
- 求最大总欢乐值。

战略游戏

- http://poj.org/problem?id=1463

- 给出一棵树，选取尽可能少的点，覆盖所有边。

树上背包

- n个结点构成树，每个结点是一个物品。
- 要选择一个结点上的物品，必须选择其父节点上的。
- 求体积m的背包能装下的最大总价值。

状压DP

- 状态压缩其实就是找一个比较好的表示方法记录状态。
- 一般都会利用2进制（3进制之类的也有可能呦）。
- 2进制可以表示集合元素是否存在之类的。

玉米地

- http://poj.org/problem?id=3254
- 在NxM的网格地上选取若干个网格。
- 要求选取的网格不相邻。
- 求总方案数。
- N,M≤12

炮兵阵地

- http://poj.org/problem?id=1185

- 题面看POJ或Luogu或Lydsy，太长了。

矩形覆盖

- http://poj.org/problem?id=2836

- 给出平面上的n个点，用最小面积和的矩形们覆盖。

数位DP

- 统计一个区间[l, r]内符合某种要求的整数个数
- count([l, r]) = count([0, r]) - count([0, l - 1])
- 所以只要关心count([0, n])就好了
- 该要求下，可以压缩前几个既定的位的状态
- 因此许多搜索是答案重复的
- 记忆化搜索减少搜索量

不要62

Time Limit: 1000/1000 MS (Java/Others) Memory Limit: 32768/32768 K (Java/Others)
Total Submission(s): 54407 Accepted Submission(s): 20861

Problem Description

杭州人称那些傻乎乎粘嗒嗒的人为62（音：laoer）。
杭州交通管理局经常会扩充一些的士车牌照，新近出来一个好消息，以后上牌照，不再含有不吉利的数字了，这样一来，就可以消除个别的士司机和乘客的心理障碍，更安全地服务大众。
不吉利的数字为所有含有4或62的号码。例如：
62315 73418 88914
都属于不吉利号码。但是，61152虽然含有6和2，但不是62连号，所以不属于不吉利数字之列。
你的任务是，对于每次给出的一个牌照区间号，推断出交管局今次又要实际上给多少辆新的士车上牌照了。

Input

输入的都是整数对n、m（0<n≤m<1000000），如果遇到都是0的整数对，则输入结束。

Output

对于每个整数对，输出一个不含有不吉利数字的统计个数，该数值占一行位置。

Sample Input

```
1 100
0 0
```

Sample Output

```
80
```

不要62

- 当然，预处理一下每个数字是否合法，加上前缀和就能通过此题了

- 但是我们要学习数位DP不是吗?

不要62

- 从高位到低位依次决定
- 首先每一位都不能填4
- 当前位能否填2只与上一位是否是6有关
- 用一个0/1状态记录上一位是否是6

不要62

- 当前位能填哪些数字还与之前的位是否已经满足严格小于要求数字n
- 举个栗子👇
- 如果n=12345，已经填了3位，122_ _，那么后两位是可以不必考虑数字要<=4或<=5的
- 所以再用一个0/1状态记录前位是否小于n了

不要62

- 不妨用搜索形式实现👉

```
int num[10];    // number

int search(int p, int a, int b) {
    /*
     * p表示剩余的位数
     * a表示当前位是否被限制（不能填2）
     * b表示当前位是否可以无视大小（之前位已小于n）
     */

    if (p == 0)
        return 1;   // 边界

    int ans = 0, lim = b ? 9 : num[p];

    for (int i = 0; i <= lim; ++i)
        if (i != 4 && !(i == 2 && a))
            ans += search(p - 1, i == 6, b || (i < num[p]));

    return ans;
}

int count(int n) {
    for (int i = 1; i <= 6; ++i)
        num[i] = n % 10, n /= 10;

    return search(6, 0, 0);
}
```

不要62

- 加个记忆化就完美了👉

```cpp
#include <cstdio>
#include <cstring>

int num[10];    // number
int mem[10][2]; // memory

int search(int p, int a, int b) {
    if (p == 0)
        return 1;   // 边界
    if (b == 1 && ~mem[p][a])
        return mem[p][a];   // 记忆化

    int ans = 0, lim = b ? 9 : num[p];

    for (int i = 0; i <= lim; ++i)
        if (i != 4 && !(i == 2 && a))
            ans += search(p - 1, i == 6, b || (i < num[p]));

    if (b == 1)
        mem[p][a] = ans;

    return ans;
}

int count(int n) {
    for (int i = 1; i <= 6; ++i)
        num[i] = n % 10, n /= 10;

    return search(6, 0, 0);
}

signed main() {
    int l, r;

    memset(mem, -1, sizeof mem);

    while (scanf("%d%d", &l, &r), l || r)
        printf("%d\n", count(r) - count(l - 1));

    return 0;
}
```

F(x)

Time Limit: 1000/500 MS (Java/Others) Memory Limit: 32768/32768 K (Java/Others)
Total Submission(s): 7792 Accepted Submission(s): 3061

Problem Description

For a decimal number x with n digits ($A_nA_{n-1}A_{n-2}$ ... A_2A_1), we define its weight as $F(x) = A_n * 2^{n-1} + A_{n-1} * 2^{n-2} + ... + A_2 * 2 + A_1 * 1$. Now you are given two numbers A and B, please calculate how many numbers are there between 0 and B, inclusive, whose weight is no more than F(A).

Input

The first line has a number T (T <= 10000) , indicating the number of test cases.
For each test case, there are two numbers A and B (0 <= A,B < 10^9)

Output

For every case,you should output "Case #t: " at first, without quotes. The *t* is the case number starting from 1. Then output the answer.

Sample Input

```
3
0 100
1 10
5 100
```

Sample Output

```
Case #1: 1
Case #2: 2
Case #3: 13
```

F(X)

- 不难发现，因为数字最多9位，所以F值上限并不大👇

```
irb(main):001:0> 9 * (2**9 - 1)
=> 4599
```

F(x)

- 和上一题类似，显然数位DP
- 考虑决策后若干位时受到的限制有哪些

F(X)

- 后几位具体是几位?
- 能不能无视n的对应位的大小限制?
- 后若干位的F值的上限是多少?

F(X)

```cpp
#include <cstdio>
#include <cstring>

int f(int x) {
    int ans = 0;
    for (int i = 1; x; i <<= 1, x /= 10)
        ans += (x % 10) * i;
    return ans;
}

int num[10];    // number
int mem[10][5000];  // memory

int search(int p, int a, int b) {
    /*
     * p表示还剩多少位
     * a表示f值的上限
     * b表示是否无视num[p]限制
     */
    if (a < 0)
        return 0;
    if (p == 0)
        return 1;
    if (b && ~mem[p][a])
        return mem[p][a];
    int ans = 0, lim = b ? 9 : num[p];
    for (int i = 0; i <= lim; ++i)
        ans += search(p - 1, a - i * (1 << (p - 1)), b || (i < num[p]));
    if (b)
        mem[p][a] = ans;
    return ans;
}

signed main() {
    memset(mem, -1, sizeof mem);
    int t, a, b;
    scanf("%d", &t);
    for (int i = 1; i <= t; ++i) {
        scanf("%d%d", &a, &b);
        for (int i = 1; i <= 9; ++i)
            num[i] = b % 10, b /= 10;
        printf("Case #%d: %d\n", i, search(9, f(a), 0));
    }
}
```

杂题DP

移动数字

- 给出一个n级排列。
- 每次可以移动一个数字到任意位置。
- 求化成自然排列的最小步数。

矩形嵌套

- 一些形如Ai x Bi的矩形。
- 当且仅当下面两个中至少一个成立——
- 1. A < C 且 B < D
- 2. A < D 且 B < C
- 矩形AxB才能嵌套在CxD里。
- 求最多嵌套几个矩形。

邮局

- http://poj.org/problem?id=1160
- 在数轴上有n个城镇，第i个城镇的坐标是pi。
- 在m个城镇建立邮局，使得每个城镇到最近邮局的距离之和最小，求这个最小距离和。

DP优化（选讲）

关照一下（可能的）基础较高的同学，免得他们太无聊睡着了

二进制多重背包

单调队列优化

斜率优化

四边形不等式

结语

- 感谢观看，坚持听下来一定会有收获👏
- 课上的学习只是一角，课下请诸君务必躬行实践以求真知
- 祝愿大家学习顺利，实力飙升，~~早日脱单~~😊

Thanks for watching

You Siki