动态规划

1086 背包问题 V2

- 有N种物品，每种物品的数量为C1，C2......Cn。从中任选若干件放在容量为W的背包里，每种物品的体积为W1，W2......Wn（Wi为整数），与之相对应的价值为P1,P2......Pn（Pi为整数）。求背包能够容纳的最大价值。

引入——简化版问题

- 有N种物品，每种物品都只有1个。从中任选若干件放在容量为W的背包里，每种物品的体积为W1，W2……Wn（Wi为整数），与之相对应的价值为P1,P2……Pn（Pi为整数）。求背包能够容纳的最大价值。

引入——简化版问题

- 搜索。

```
void dfs(int i,int noww,int nowp)
{
    if(i>n)
    {
        if(noww<=W)
            Ans=max(Ans,nowp);
        return;
    }
    dfs(i+1,noww+w[i],nowp+p[i]);
    dfs(i+1,noww,nowp);
}
```

引入——简化版问题

- 搜索的状态有限，有用的只有n*W种。
- 不妨记录f[i][noww]表示考虑前i个物品，总体积为noww时nowp最大为多少，进行记忆化搜索。
- 实际上，我们不需要记忆化，按照顺序递推这个数组就好了。

引入——简化版问题

```
void dp()
{
    for(int i=0;i<n;i++)
        for(int noww=0;noww<=W;noww++)
        {
            if(noww+w[i+1]<=W)
                f[i+1][noww+w[i+1]]=max(f[i+1][noww+w[i+1]],f[i][noww]+p[i+1]);
            f[i+1][noww]=max(f[i+1][noww],f[i][noww]);
        }
}
```

- 当然你也可以优化一下空间。

引入——简化版问题

```
void dp()
{
    for(int i=0;i<n;i++)
        for(int noww=W;noww>=w[i+1];noww--)
            f[noww]=max(f[noww],f[noww-w[i+1]]+p[i+1]);
}
```

1086 背包问题 V2

- 我们也可以用同样的状态来做这个题。

```
void dp()
{
    for(int i=0;i<n;i++)
        for(int noww=W;noww>=w[i+1];noww--)
            for(int j=1;j<=c[i+1]&&noww-w[i+1]*j>=0;j++)
                f[noww]=max(f[noww],f[noww-w[i+1]*j]+p[i+1]*j);
}
```

- 只不过会超时罢了。

1086 背包问题 V2

- 如何优化？
- 二进制，或者单调队列。
- 10以内的所有数字只需要1,2,4,3就可以组合出来。

1503 猪和回文

- 题目好长啊。

1503 猪和回文

- 先考虑搜索，从开头搜，好像并不能记录状态。
- 可以两端一起搜！

1503 猪和回文

- 记录f[a][b][c][d]表示从(1,1)走到(a,b)，(n,m)走到(c,d)的方案数，转移很容易——
- f[a][b][c][d]=f[a+1][b][c-1][d]+f[a][b+1][c-1][d]+f[a+1][b][c][d-1]+f[a][b+1][c][d-1]

1503 猪和回文

- 似乎，超时了？

1503 猪和回文

- 聪明的小朋友一定已经发现了！a+b和c+d之间是有关系的！我们只需要记录a+b,a,c就可以算出d了！
- 那么f[S][a][c]就表示从(1,1)走到(a,b)，(n,m)走到(c,d)的方案数，转移是一样的。
- 似乎，超空间了？

1503 猪和回文

- 我们发现，处理f[S][*][*]的时候只和f[S-1][*][*]有关，那么我们可以只用两个数组处理。这也是动态规划节省空间的常用做法。

1412 AVL树的种类

- 平衡二叉树(AVL树），是指左右子树高度差至多为1的二叉树，并且该树的左右两个子树也均为AVL树。现在问题来了，给定AVL树的节点个数n，求有多少种形态的AVL树恰好有n个节点。
- 让我们直接从设状态入手，不考虑搜索了吧！

1412 AVL树的种类

- 显然，我们希望对左右子树分别统计，最后乘起来就是总数。但是，左子树和右子树之间要满足一定的关系，而这只和深度有关。

- 那我们就把深度塞进状态里吧！

1412 AVL树的种类

- 用f[n][d]表示n个点，深度为d的AVL树有多少种。
- 枚举左子树大小为i，进行转移。
- f[i][d-1]*f[n-i-1][d-1]
- f[i][d-2]*f[n-i-1][d-1]
- f[i][d-1]*f[n-i-1][d-2]

1607 卷积和

- 题目好长啊。

1607 卷积和

- 看起来不那么好做，一点一点来分析。
- 首先，对于区间问题，我们可以转化为两个前缀和。即求小于等于R的再减去小于等于L-1的。

1607 卷积和

- 先考虑对于所有n位数求卷积和。
- 只需要枚举两位，再枚举这两位是多少，算出共有多少组就可以了。

1607 卷积和

- 不妨设R是n位数。只需要计算小于等于R的n位数的卷积和就好了。
- 同样，从一端开始搜索不太方便，从两端开始搜索。

1607 卷积和

- 状态可以记为f[a][0/1][0/1/2]，表示搜索了前i位，第一个0/1表示前面的数字和R的大小关系（只有小于或等于），0/1/2表示后面的数字和R的大小关系。

- 枚举下两个数字是多少进行转移即可。

1607 卷积和

- 状态可以记为f[a][0/1][0/1/2]，表示搜索了前i位，第一个0/1表示前面的数字和R的大小关系（只有小于或等于），0/1/2表示后面的数字和R的大小关系。

- 枚举下两个数字是多少进行转移即可。

1409 加强版贪吃蛇

- 题目好长啊。
- 先不考虑传送。

1409 加强版贪吃蛇

- 由于不能向左走，也不能走重复的格子，所以在某一列只能朝上或者朝下走。
- 很好设出状态。

1409 加强版贪吃蛇

- f[i][j][0/1]表示走到第i列第j行的格子，现在只能向上走或者向下走的最高积分。

- 转移很容易。

1409 加强版贪吃蛇

- 考虑传送。不失一般性，我们只考虑向下走的传送。
- 不管在哪里传送，最后都会变成0。而从上向下走的时候，不能经过重复点，也就是说，我们选择尽量靠下的位置进行传送。
- 直接枚举传送后走几步即可。

1779 逆序对统计

- lyk最近计划按顺序做n道题目，每道题目都分为很多分数档次，lyk觉得这些题太简单了，于是它想到了一个好玩的游戏。lyk决定将每道题目做出其中的某个分数，使得这n道题目的逆序对个数最多。
- 为了方便，假设共有m个分数档次，并且会给m个分数档次分配一个题目编号，表示该题目会出现这个分数档次。
- 题目保证每道题都存在至少一个分数档次。（例如样例中5道题目的分数分别是5,6,3,4,7,共有4个逆序对）

1779 逆序对统计

- 怎么统计逆序对才能不重不漏？
- 按照位置顺序统计？按照大小顺序统计？

1779 逆序对统计

- 按照大小顺序统计！
- 我们只需要知道之前有哪些位置已经被选了。
- 状态压缩。

1780 完美序列

- 如果一个序列的相邻两项差的绝对值小于等于1，那么我们说这个序列是完美的。给出一个有序数列A，求有多少种完美序列排序后和数列A相同。

1780 完美序列

- 如果我们把最大的数字删掉，显然还是个完美序列。
- 不妨从小到大依次添加数字。

1780 完美序列

- 考虑我们填数字需要知道什么。当我们填i+1时，需要知道有多少个i在开头，有多少对i相邻。
- 那么用f[i][a][0/1/2]记录就好了。
- 用组合数划分i+1，并填入。

1849 Clarke and package

- 题目不仅长，而且还不怎么好理解。

1849 Clarke and package

- 由于期望的线性性，我们可以考虑每个物品的贡献。
- 这里dp和背包差不多。

1597 有限背包计数问题

- 你有一个大小为n的背包，你有n种物品，第i种物品的大小为i，且有i个，求装满这个背包的方案数有多少。

1597 有限背包计数问题

- 问题很经典，直接用二进制优化的方法并不能解决。

1597 有限背包计数问题

- 将背包分成两类，小于根号的和大于根号的。最后乘起来。
- 小于根号的只有根号种，设f[i][j]表示用前i个装j的大小的方案数。
- 很好优化。

1597 有限背包计数问题

- 超过根号的并没有数量限制。
- 用划分数的方法进行横向dp。
- g[i][j]表示i个数字和为j的方案。每次可以全部加1或者新增一个根号大小的数字。

1611 金牌赛事

- 题面好长啊。

1611 金牌赛事

- 考虑从前到后依次决定是否修。
- f[i]表示前i段最高收益。

1611 金牌赛事

- 转移就是枚举从哪里开始一直修到i。f[j]加上从j+1到i的所有收入。
- 这个收入可以用线段树维护。

总结：动态规划

- 关键在于设状态，一个好的状态可以很容易推出转移方程。
- 适当掌握一些优化技巧。