DP 专题

10circle

2025.2

序言

对于一个能用动态规划解决的问题，一般采用如下思路解决：
将原问题划分为若干子问题，子问题再划分为子问题，提取这些子问题的特征，即用这些特征可以描述出一个子问题（称之为状态）；寻找每一个子问题的可能决策，或者说是各状态间的相互转移方式（用数学的语言描述就是状态转移方程）。之后按顺序求解每一个子问题。
如果将每一个状态看作一个节点，状态转移看作一条边，那么整个问题就是一个有向无环图，我们只需要按照拓扑序求解每一个节点即可。

序列 dp

序列 dp，一般是设 f_i 为只考虑序列的前 i 项（或者是以 i 为结尾），要求的东西是什么。也可能有第二维或者更多维，总之都是认为一个前缀是一个子问题。

例题：最长上升子序列链接：
https://www.luogu.com.cn/problem/B3637

最长上升子序列题解

设 f_i 为以 i 为结尾的最长上升子序列。
为了方便，令序列第 0 项为负无穷，$f_0 = 0$。转移方程是
$f_i = \max\limits_{0 \le j < i, a_j < a_i} (f_j + 1)$。
举例：序列 $(1, 3, 2, 1, 4)$，可以计算得到 $f = (1, 2, 2, 1, 3)$。
时间复杂度 $O(n^2)$。

区间 dp

一般状态设在区间上，比如对于某个区间而言，可以得到什么，最小最大是什么诸如此类。
例题：链接：https://www.luogu.com.cn/problem/P4290

区间 dp 例题题解

定义 $f_{l,r,c}$ 表示区间 $[l,r]$ 是否有可能由字符 c 扩充得来。因为每个字符只会扩展成两个，所以转移时可以枚举中间点 k，再枚举两侧都能是什么，再判断是否符合要求。
具体的，设 $a_{i,j,k}$ 表示字符 i 能否扩展成字符 j,k（这个可以预处理），字符集合是 S，那么转移方程是
$f_{l,r,c} = \mathrm{OR}_{k=l+1}^{r} \mathrm{OR}_{j\in S} \mathrm{OR}_{k\in S}(f_{l,k-1,j} \text{ AND } f_{k,r,k} \text{ AND } a_{c,j,k})$。

例题 01 背包

链接：https://www.luogu.com.cn/problem/P2871

题解

设状态 $f_{i,j}$ 表示在只能放前 i 个物品的情况下，容量为 j 的背包所能达到的最大总价值。
考虑转移。假设当前已经处理好了前 $i-1$ 个物品的所有状态，那么对于第 i 个物品，当其不放入背包时，背包的剩余容量不变，背包中物品的总价值也不变，故这种情况的最大价值为 $f_{i-1,j}$；当其放入背包时，背包的剩余容量会减小 w_i，背包中物品的总价值会增大 v_i，故这种情况的最大价值为 $f_{i-1,j-w_i}+v_i$。
由此可以得出状态转移方程：$f_{i,j}=\max(f_{i-1,j},f_{i-1,j-w_i}+v_i)$。

背包

~~背包有用？真的假的？0_o~~
虽然说背包确实用处不大，但是它作为一个基础问题类型还是很重要的。
可以把背包看作一个存储了在花费 i 的空间后能够收获价值 a_i 的数组。可以设为花费小于等于 i 的空间，也可以设为恰好花 i 的空间（恰好时记得初始化负无穷）。
01 背包就是加入一个物品 (w, v)，此时会倒序更新 a 数组，不能用到已经被更新的值。

背包

完全背包就是加入无数个物品 (w, v)，因为有无数个，所以可以正序更新 a 数组，用到已经被更新的值。

多重背包的单调队列优化法，在学习单调队列是怎么做之后，就可以一样的做了。不过多重背包加入物品 (w, v) 时要开 w 个单调队列分别处理模 w 不同的几个部分。

合并两个背包就是类似一个个插入，也可以理解为 $(\max, +)$ 卷积。

树形 dp

顾名思义，在树上做的 dp 就是树形 dp。

例题朴素树形 dp

链接：https://www.luogu.com.cn/problem/P1352

题解

我们设 $f(i,0/1)$ 代表以 i 为根的子树的最优解（第二维的值为 0 代表 i 不参加舞会的情况，1 代表 i 参加舞会的情况）。
对于每个状态，都存在两种决策（其中下面的 x 都是 i 的儿子）：
上司不参加舞会时，下属可以参加，也可以不参加，此时有 $f(i,0)=\sum\max\{f(x,1),f(x,0)\}$；
上司参加舞会时，下属都不会参加，此时有 $f(i,1)=\sum f(x,0)+a_i$。
我们可以通过 DFS，在返回上一层时更新当前结点的最优解。

树形背包

我们设 $f(u,i,j)$ 表示以 u 为根的子树中，已经遍历了 u 的前 i 棵子树，容量为 j 时的最大价值。
枚举 u 点的每个子结点 v，同时枚举以 v 为根的子树选了多少，将子树的结果合并到 u 上。
记点 x 的儿子个数为 s_x，以 x 为根的子树大小为 siz_x，可以写出下面的状态转移方程：

$$f(u,i,j)=\max_{v,k\le j,k\le siz_v} f(u,i-1,j-k)+f(v,s_v,k)$$

树形背包

注意上面状态转移方程中的几个限制条件，这些限制条件确保了一些无意义的状态不会被访问到。
f 的第二维可以很轻松地用滚动数组的方式省略掉，注意这时需要倒序枚举 j 的值。
很重要的一点：该方法要求每个点最多只会让重量（或者其它什么）有 1 的更新，不然复杂度就是错的。完全的树形背包可以看 loj 模板的题解，大概是按照后序遍历的 dfs 序去做序列背包，不过如果不选择 i 的话必须从 i 子树未出现的位置转移。
模板：https://www.luogu.com.cn/problem/P2014

复杂度证明

考虑将树形背包合并的过程看成一棵二叉树。以下内容都基于二叉树分析。

定义极小大子树是 u 大小大于 m，其孩子大小都小于等于 m。极大小子树是 u 大小小于等于 m，而其父亲大小大于 m。

小子树和大子树的大小以 m 为分界。

复杂度证明

- 两个孩子都是小子树：根据两个点在 LCA 贡献一次，得到复杂度是极大小子树大小平方和。可以证明其小于 $O(nm)$。
- 两个孩子都是大子树：这一定合并了两个极小大子树。极小大子树又互不相交，最多 $O(n/m)$ 个，最多合并 $O(n/m)$ 次，总复杂度 $O((n/m)m^2) = O(nm)$。
- 一个大一个小：如果出现这种情况，花费是 m 乘上该极大小子树的大小。极大小子树互不相交，大小之和最多 $O(n)$。

综上，复杂度 $O(nm)$。

换根 dp

过程：先 dfs 一遍得到以 1 为根的 dp 值。之后从 1 开始 dfs，每次切换到当前节点时，只有和切换有关的两个节点的 dp 值会改变。将根为 u 切换到根为 v 时，可以先记录下当前 u, v 的 dp 值，随后先把 u 的 dp 值更新为不含 v 的，再把 v 的 dp 值更新为含 u 的。切换回来的时候直接用记录的 dp 值赋值就好了。
有可减性的 dp 值会好处理一些。
例题：https://www.luogu.com.cn/problem/P3478

图上 dp

就是在图上做 dp，一般是按照拓扑序，不过也有一般的 dp。

例题 1

链接：https://www.luogu.com.cn/problem/P1807

题解

令 f_i 表示 1 到 i 的最长路长度，在拓扑序（也就是 1 到 n）下转移即可。

例题 2 NOIP 2017 逛公园

链接：https://www.luogu.com.cn/problem/P3953

题解

首先考虑无穷的可能性，有无穷条路径当且仅当存在一个点 i，使得 i 连接了一条零边，并且 $dis_{1,i} + dis_{i,n} \leq dis_{1,n} + K$。否则，令 $f_{i,j}$ 表示从 1 到 i，路径长度为 $dis_{1,i} + j$ 的路径数量。可以证明这个状态的转移不会出现环，因此直接记忆化搜索就可以。这样可以避免再去计算 dp 转移的顺序。

状态压缩

顾名思义，将状态压缩存储，一般是将一堆 01 压缩成一个二进制。

例题 1

链接：https://www.luogu.com.cn/problem/P1433

题解

设 $f_{i,S}$ 表示当前在 i，已经吃过了集合 S 中的所有内容的最短时间。
转移可以直接枚举下一个点。

例题 2

链接：https://www.luogu.com.cn/problem/P5911

题解

设 f_S 表示当前已经过河的集合是 S 所用的最短时间。
转移需要枚举一个没过河的人的子集，判断其是否可行并转移。
枚举非空子集的办法：`for(int i = S; i; i = (i - 1) & S) {}`

例题 3：NOIP2017 宝藏

链接：https://www.luogu.com.cn/problem/P3959

题解

这个题的核心在于想到离起点距离相同的边同时处理，这样可以避免记录每个点的距离。
设状态 $f_{i,S}$ 表示当前离起点最远的点的距离为 i，并且已经拿到了集合 S 中的宝藏的最短时间。转移时可以枚举距离为 $i+1$ 的点的集合，计算代价并转移至新状态。在计算代价时，我们并不需要只枚举距离为 $i+1$ 的点到距离为 i 的点的所有边，而可以枚举到任意一个已经到达的点，因为如果不是到距离为 i 的点，那么一定存在另外的转移方式使得被转移到的状态更优。
暴力转移的复杂度是 $O(3^n nm)$，可以通过预处理 $g_{i,S}$ 表示点 i 到集合 S 中的点的最短边长，这样复杂度就是 $O(3^n n^2)$。如果在额外预处理集合到集合的最短边长和，复

数位 dp

相当于依次考虑每一位来 dp，和其它并无不同。不过一般会有许多细节，需要注意。

例题

链接：https://www.luogu.com.cn/problem/P2657

题解

首先我们将问题转化成更加简单的形式。设 ans_i 表示在区间 $[1, i]$ 中满足条件的数的数量，那么所求的答案就是 $ans_r - ans_{l-1}$。
对于一个小于 n 的数，它从高到低肯定出现某一位，使得这一位上的数值小于 n 这一位上对应的数值。而之前的所有位都和 n 上的位相等。

题解

有了这个性质，我们可以定义 $f(i, st, op)$ 表示当前将要考虑的是从高到低的第 i 位，当前该前缀的状态为 st 且前缀和当前求解的数字的大小关系是 op（$op = 1$ 表示等于，$op = 0$ 表示小于）时的数字个数。在本题中，这个前缀的状态就是上一位的值，因为当前将要确定的位不能取哪些数只和上一位有关。在其他题目中，这个值可以是很多东西。状态转移方程则需要对于 op 的值分类讨论。随后枚举下一位的值，判断可行性，并进行转移。

合并本质相同的状态

如果有多个状态的值一定相同（可以利用对称性之类的来证明），可以考虑将它们合并到一个状态。
例题：https://www.luogu.com.cn/problem/P2051

交换 dp 一维和 dp 值

如果 dp 的值非常小，而 dp 的某一维比较大，又有单调性之类的东西的话，可以考虑这个技巧。
例题：
https://atcoder.jp/contests/agc033/tasks/agc033_d

数据结构优化

可能是要对 dp 取一些区间 max、min 之类的，此时就可以用线段树优化 dp。也可能是一些其它的东西，符合单调队列能做的东西的话（比如有单调性的区间最值）就能用单调队列优化。

也有一部分 dp 可以使用前缀和的方式去优化。

例题 1

https://hydro.ac/p/H1069

例题 2

https://www.luogu.com.cn/problem/P2569

矩阵快速幂优化

如果 dp 的一维非常大，而另一维较小的话，可以利用矩阵来记录转移，利用广义矩阵乘法进行转移的复合。

比如，$f_{i,j} = \sum_{k=1}^{m} f_{i-1,k} \times w_{j,k}$，就是一个典型的这类题。其中的转移就是矩阵 w，而转移的复合方式就是普通的矩阵快速幂。最后再让开头的 dp 状态乘上最终转移就能得到最终的 dp 状态。

例题：https://www.luogu.com.cn/problem/P8408

静态问题决策单调性的分治

现在想要求 $f_i = \min_{j=0}^{i-1} w(j,i)$，但是对于 f_i 而言的最优的 j 关于 i 是单调递增的。常见于多层的 dp，下层 dp 已经计算完成，这样 $w(j,i)$ 就不会依赖当前在计算的值。
做法是，定义 solve(l，r，L，R) 表示现在想要求区间 $[l,r]$ 的 dp 值，而它们的决策点在 $[L,R]$ 中。随后取 $m=\frac{l+r}{2}$，计算 f_m 的值和其决策点 p，随后分治下去。若 w 计算是 $O(1)$ 的，那么复杂度是 $O(n\log n)$。
例题：https://www.luogu.com.cn/problem/CF868F

斜率优化

比如一个经典的 dp 方程：$f_i = \min_{j=1}^{i-1}(f_j + a_i b_j) + c_i$，其中 a, b, c 都是提前给定的。它就可以用斜率优化做。

第一种方式是使用李超树解：将 $y = xb_j + f_j$ 看作一个直线，那么只需要查询在 $x = a_i$ 位置时所有直线的最小值就好了。这是李超树板子。

第二种方式是使用凸包解：将 $(-b_j, f_j)$ 看作一个平面上的点，那么只需要查询斜率是 a_i 时，经过某个点后在零处的取值（即截距）的最小值就好了。

一般有 b_i, f_i 单调性之类的题用第二种方式会好写一些，不过第一种方式更加通用。

例题：https://www.luogu.com.cn/problem/P3195

剪切线

以下内容较为困难，时间不多，难以细讲，不明白是正常的。用处也不大，就当听个乐（）
感兴趣的话可以自行搜索 oiwiki 和讲解的博客仔细研究。

四边形不等式

以下内容暂时忽略证明，有兴趣的可以自行查找具体证明。
问题（1）：求 $f_i = \min_{j=1}^{i} w(j, i)$。
四边形不等式：如果对于任意 $a \le b \le c \le d$，均成立 $w(a, c) + w(b, d) \le w(a, d) + w(b, c)$，则称函数 w 满足四边形不等式（简记为「交叉小于包含」）。若等号永远成立，则称函数 w 满足四边形恒等式。
若 w 满足四边形不等式，则问题（1）满足决策单调性。

二分 + 队列优化

如果计算 $w(j,i)$ 需要 f_{j-1}，那么就不能再用分治了。二分队列的核心在于维护每个点的决策点，并且由于四边形不等式，可以得到一些很厉害的单调性。
队列维护的是现在以每个 j 为决策点的 i 的区间。更新就是从后往前，如果新加入的在区间左端点都能打败队列末尾就弹出，否则二分查找第一个能打败的地方并修改队列。

wqs 适用性

如果指定将序列拆分成 m 个子区间。此时需要将拆分后的区间个数作为转移状态的一维。相应地，有 2D1D 状态转移方程如下。

问题（2）：$f(k,i) = \min_{1\le j\le i} f(k-1,j-1) + w(j,i) \qquad (1 \le k \le m,\ \ 1 \le i \le n)$

这里，$f(0,0) = 0$ $f(0,i) = f(k,0) = \infty$ 对任意 $1 \le k \le m$ 和 $1 \le i \le n$ 都成立。

若 w 满足四边形不等式，则对于问题（2）成立 $\mathrm{opt}(k-1,i) \le \mathrm{opt}(k,i) \le \mathrm{opt}(k,i+1)$。

若 w 满足四边形不等式，则问题（2）的最优解 $g(k) = f(n,k)$ 是关于 k 的凸函数。

区间 dp 优化

要将 n 个长度为一的区间 $[i,i]$ 两两合并起来，直到得到区间 $[1,n]$。每次合并 $[j,k]$ 和 $[k+1,i]$ 时都需要支付成本 $w(j,i)$。问题要求找到成本最低的合并方式。对于此类问题，有如下 2D1D 状态转移方程。

问题（3）:

$f(j,i)=\min_{j\le k<i} f(j,k)+f(k+1,i)+w(j,i) \qquad (1\le j<i\le n)$。

这里给定任意初始成本 $f(i,i)=w(i,i)$。

除了四边形不等式以外，区间合并问题的决策单调性还要求成本函数满足区间包含单调性，即，若区间 A 包含区间 B，则 $w(A)\ge w(B)$。

若 w 满足区间包含单调性和四边形不等式，则状态 $f(j,i)$ 满足四边形不等式。

区间 dp 优化

若 w 满足区间包含单调性和四边形不等式，则问题 (3) 中最小最优决策 $\mathrm{opt}(j,i)$ 满足

$\mathrm{opt}(j,i-1) \leq \mathrm{opt}(j,i) \leq \mathrm{opt}(j+1,i). \qquad (j+1<i)$。

利用这一结论，同样可以限制决策点 k 的搜索范围。在这里，正序遍历区间长度 $i-j+1$，再遍历具有同样长度的所有区间 $[j,i]$，暴力搜索 $\mathrm{opt}(j,i-1)$ 和 $\mathrm{opt}(j+1,i)$ 之间的所有 k 求得最优解 $f(j,i)$ 并记录最小最优决策 $\mathrm{opt}(j,i)$。

对于同样长度的所有区间，此算法中决策空间总长度是 $O(n)$ 的，而可能的区间长度的数目同样是 $O(n)$ 的，故而总的算法复杂度为 $O(n^2)$ 的。

铺瓷砖

以其中一个题为例，其余题目都是类似的。
链接：https://www.luogu.com.cn/problem/P8634

铺瓷砖题解

分果果

链接：https://www.luogu.com.cn/problem/P8746

分果果题解

组合数问题

链接：https://www.luogu.com.cn/problem/P8688

组合数问题题解

NOISG 2021 E Pond

链接：https://www.luogu.com.cn/problem/P11303

NOISG 2021 E Pond 题解

https://www.luogu.com.cn/paste/2a2upc8c

Double-Sorted Grid

链接：https://qoj.ac/problem/5810

Double-Sorted Grid 题解