动态规划

laofu

陈江伦 (IIIS)

July 30, 2022

动态规划

求解决策过程最优化的数学方法

动态规划 (dynamic programming) 是运筹学的一个分支，是求解决策过程 (decision process) 最优化的数学方法。
20 世纪 50 年代初美国数学家 R.E.Bellman 等人在研究多阶段决策过程 (multistep decision process) 的优化问题时，提出了著名的最优化原理 (principle of optimality)，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法——动态规划。
1957 年出版了他的名著《Dynamic Programming》，这是该领域的第一本著作。

动态规划举例

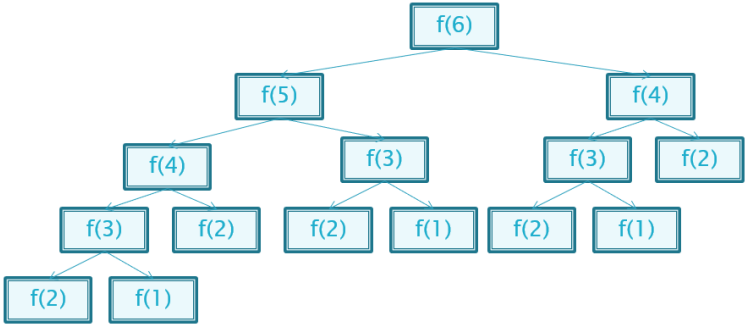
走楼梯问题

一个楼梯有 n 级，每次走 1 级或 2 级，从底到顶共有几种走法。

递归算法：$f(n) = f(n-1) + f(n-2)$，不断递归
动态规划算法：从 3 开始，$f(i) = f(i-1) + f(i-2)$

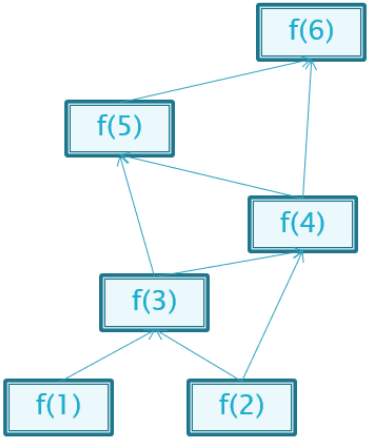
动态规划举例

递归算法求解过程



动态规划举例

动态规划算法求解过程



为什么有动态规划?

贪心：直接取最优
动态规划：划分阶段，每个阶段取最优

最优子结构

问题的最优解由相关子问题的最优解组合而成，一个问题的最优解包含其子问题的最优解
典型的有背包问题和钢条切割我问题
所谓子问题就是一中组合，将一个问题分成许多子问题的集合
某个子问题转化为问题时，所需要的代价是固定的

动态规划中的概念

阶段 把所给求解问题的过程恰当地分成若干个相互联系的阶段，以便于求解，过程不同，阶段数就可能不同．描述阶段的变量称为阶段变量

状态 状态表示每个阶段开始面临的自然状况或客观条件，它不以人们的主观意志为转移，也称为不可控因素

决策 一个阶段的状态给定以后，从该状态演变到下一阶段某个状态的一种选择（行动）称为决策。描述决策的变量称决策变量

策略 由每个阶段的决策组成的序列称为策略

什么时候需要动态规划？

- 贪心策略失效
- 搜索、模拟等一般方法明显超时、超空间
- 可以看出有阶段划分，大问题可化为小问题，且问题之间又相互依赖（大问题的解决取决于小问题的解决）
- 对于每一个问题答案比过程重要，之前的决策与之后决策无影响或可以用简单状态表示出来
- 求最优化问题、概率问题等
- 考虑可行性（如设计状态、转移等）

动态规划的使用条件

- 最优化原理：一个最优策略的子策略，对于它的初态和终态而言也必是最优的
- 无后效性：某阶段的状态一旦确定，则此后过程的演变不再受此前各种状态及决策的影响
- 子问题的重叠性：可以递推，解决冗余，以空间换时间

动态规划的过程

1. 划分阶段
2. 确定状态和状态变量
3. 确定决策并写出状态转移方程
4. 寻找边界条件

动态规划的难点

- 状态设计与阶段划分: 当题目出现环时?
- 设置状态转移方程：注意转移的精确和转移的时间开销
  - 有用的最优状态所占比重越高越好。
  - 状态转移方程的复杂度越低越好。
- 复杂的条件: 主要考虑处理的先后问题
- 输出方案或方案个数

最长上升子序列问题

给一个序列，求最长的递增子序列

最长上升子序列问题

给一个序列，求最长的递增子序列
状态:$f[i]$ 表示以 i 结尾的最长上升子序列的长度
转移:

$$f[i] = 1 + \max_{j<i, a_j<a_i} f[j]$$

使用数据结构可以优化到 $n \log n$

最长上升子序列问题

二分栈

- 假设已经处理完了一部分，当前最长上升子序列长度为 k

最长上升子序列问题

二分栈

- 假设已经处理完了一部分，当前最长上升子序列长度为 k
- 对于 $\forall i \in [1, k]$，可能存在很多长度为 i 的子序列

最长上升子序列问题

二分栈

- 假设已经处理完了一部分，当前最长上升子序列长度为 k
- 对于 $\forall i \in [1, k]$，可能存在很多长度为 i 的子序列
- 我们只需要保留最优的一个：结尾最小

最长上升子序列问题

二分栈

- 假设已经处理完了一部分，当前最长上升子序列长度为 k
- 对于 $\forall i \in [1, k]$，可能存在很多长度为 i 的子序列
- 我们只需要保留最优的一个：结尾最小
- 定义 $g[i]$ 表示长度为 i 的子序列结尾最小值

最长上升子序列问题

二分栈

- 假设已经处理完了一部分，当前最长上升子序列长度为 k
- 对于 $\forall i \in [1, k]$，可能存在很多长度为 i 的子序列
- 我们只需要保留最优的一个：结尾最小
- 定义 $g[i]$ 表示长度为 i 的子序列结尾最小值
- 加入一个数：只会改变一个 $g[i]$

关路灯

Codevs1258

多瑞卡得到了一份有趣而高薪的工作，每天早晨他必须关掉他所在村庄的街灯
所有的街灯都被设置在一条直路的同一侧
多瑞卡每晚到早晨 5 点钟都在晚会上，然后他开始关灯
开始时，他站在某一盏路灯的旁边，之后会以一个恒定速度走路
到达一个灯下方时可以瞬间将其关闭
每盏灯都有一个给定功率的电灯泡
因为多端卡有着自觉的节能意识，他希望在耗能总数最少的情况下将所有的灯关掉
灯的数量 ≤ 1000

关路灯

Codevs1258

- 观察：任何时刻，关掉的路灯一定位于一段区间

关路灯

Codevs1258

- 观察：任何时刻，关掉的路灯一定位于一段区间
- $f[l][r][k]$ 表示已经关闭了 $[l,r]$ 内所有的灯，现在在 k

关路灯

Codevs1258

- 观察：任何时刻，关掉的路灯一定位于一段区间
- $f[l][r][k]$ 表示已经关闭了 $[l,r]$ 内所有的灯，现在在 k
- k 这一维可以用 0/1 优化

没有上司的舞会

Ural 大学有 n 个职员，编号为 $1 \sim n$。他们有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。
每个职员有一个快乐指数。现在有个周年庆宴会，要求与会职员的快乐指数最大。但是，没有职员愿和直接上司一起与会。

没有上司的舞会

求最优解，每一个节点取舍关系到全局，可用树形动态规划解决
对于每个点，设计 $f[i][1]$ 为可以选这个点的最大快乐值，$f[i][0]$ 为未选这个点的最大快乐值

有限电视网络

pku1155 TELE

有一个电视台要用电视网络转播节目。这种电视网络是一棵树，树的节点为中转站或者用户

树节点的编号为 $1 \sim n$, 其中 1 为总站，$2 \sim (n-m)$ 为中转站，（总站和中转站统称为转发站）

$n-m+1 \sim n$ 为用户，电视节目从一个地方传到另一个地方都要费用，同时每一个用户愿意出相应的钱来付电视节目

现在的问题是，在电视台不亏本的前提下，要你求最多允许有多少个用户可以看到电视节目

如果某个用户要收到节目（叶子结点），那么电视台到该用户的路径节点的费用都要付

$n \leq 3000$

有线电视网络

pku1155 TELE

- 和上题类似

有线电视网络

pku1155 TELE

- 和上题类似
- 设置 $f[i][j]$ 表示以 i 为起点，选择 j 个用户的盈利

有线电视网络

pku1155 TELE

- 和上题类似
- 设置 $f[i][j]$ 表示以 i 为起点，选择 j 个用户的盈利
- 自顶向下 dp，对于根节点 a 每次 dp 分两种情况：

有线电视网络

pku1155 TELE

- 和上题类似
- 设置 $f[i][j]$ 表示以 i 为起点，选择 j 个用户的盈利
- 自顶向下 dp，对于根节点 a 每次 dp 分两种情况：
- 1. 子节点 b 为用户，直接更新，用 $f[a][j-1]+pay[b]-weight[ap]$ 更新 $f[a][j]$，并且必须生成 $f[a][num[a]+1], num[a]++$
  2、子节点 b 为中转站，先计算子节点 b，再用 $f[a], f[b]$ 数组更新 $f[a]$，$num[a]+=num[b]$

有线电视网络

pku1155 TELE

- 和上题类似
- 设置 $f[i][j]$ 表示以 i 为起点，选择 j 个用户的盈利
- 自顶向下 dp，对于根节点 a 每次 dp 分两种情况：
- 1. 子节点 b 为用户，直接更新，用 $f[a][j-1]+pay[b]-weight[ap]$ 更新 $f[a][j]$，并且必须生成 $f[a][num[a]+1], num[a]++$
  2、子节点 b 为中转站，先计算子节点 b，再用 $f[a], f[b]$ 数组更新 $f[a]$，$num[a]+=num[b]$
- 最后对于根节点 1，从 M 扫到 N。若 $f[1][j]\geq 0$，则最多可以选 j 个用户

选课

Codevs 1378

学校实行学分制。每门的必修课都有固定的学分，同时还必须获得相应的选修课程学分。
学校开设了 $n(n < 300)$ 门的选修课程，每个学生可选课程的数量 m 是给定的。学生选修了这 m 门课并考核通过就能获得相应的学分
在选修课程中，有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修
例如《Frontpage》必须在选修了《Windows 操作基础》之后才能选修。我们称《Windows 操作基础》是《Frontpage》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。每门课都有一个课号，依次为 $1, 2, 3, \ldots$

选课

Codevs 1378

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

表中 1 是 2 的先修课,2 是 3,4 的先修课
如果要选 3，那么 1 和 2 都一定已被选修过
你的任务是为自己确定一个选课方案，使得你能得到的学分最多，并且必须满足先修课优先的原则
假定课程之间不存在时间上的冲突

选课

Codevs 1378

和上题相似，但略微有所不同，没有中转站这个概念。
对于这个题，每一点必须在其父节点选后才可选，换言之，对于以一个节点为根的子树，根必选，也就是说要在更新时必须保证 $f[A][1] = score[A]$，并且更新中必须保证 $f[A][1]$ 使用到

重建道路

pku1947

给你一棵树，求通过求通过删除最少的边，使得存在一棵剩余子树有给定的节点数。

重建道路

pku1947

令 $f[i][j]$ 表示 i 子树中剩余包括 i 结点在内共 j 点所需删去最少边数

$f[fa][p] = min(f[fa][p-j] + f[son][j], f[fa][p])$

除设定的根节点外，其余结点更新 ans 方法为

$Ans = min(f[now][p] + 1, ans)$, 加一表示删去该子树与树上其他部位联系所需的经过的路径

对于设定的根节点 $Ans = min(f[root][p], ans)$

Tree

pku1848

题意：对于一棵 $3 \leq n \leq 100$) 个结点的树，添加最少的边使得树中每一个结点正好落在一个（仅一个）环中

Tree

pku1848

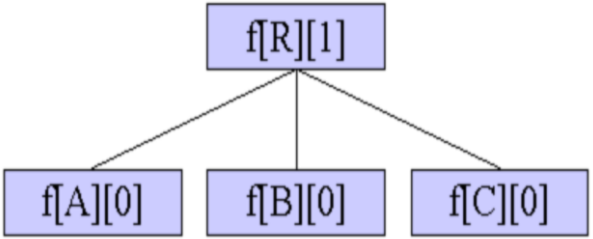
- 先分析各种情况。。。构成环至少需三个点
- 不妨定义 $f[i][0 \sim 2], f[i][0]$ 记录包括 i 在内所有 i 子树上的构成环所需添加的边数
- $f[i][1]$ 记录不包括 i 在内所有 i 子树上的构成环所需添加的边数
- $f[i][2]$ 不包括 i 及与其相连的一条链在内所有 i 子树上的构成环所需添加的边数
- 可以证明，子树内 i 若与两条链相连却不处理的情况非法

四种情况

pku1848

Case1：i 的各个儿子节点都在环中，i 无法构成环

$f[fa][1] = sum(f[son][0])$

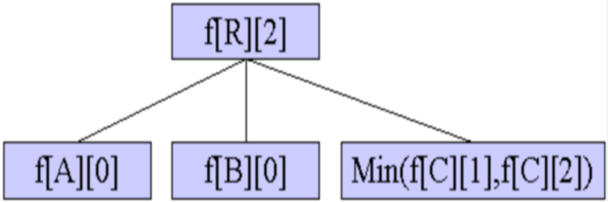


四种情况

pku1848

Case2：除 p 点或 p 链不在环中 i 的各个儿子节点都在环中，i 与 p 为链

$f[fa][2] = sum(f[son][0]) + (max(f[p][1], f[p][2]) - f[p][0])$



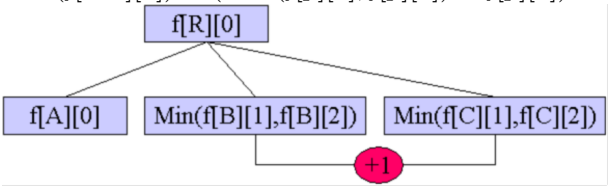
四种情况

pku1848

Case3：除 p 点或 p 链、q 点或 q 链不在环中 i 的各个儿子节点都在环中，i 与 p,q 为环

$F[fa][0] =$

$sum(f[son][0]) + (max(f[p][1], f[p][2]) - f[p][0]) + (max(f[q][1], f[q][2]) - f[q][0]) + 1$

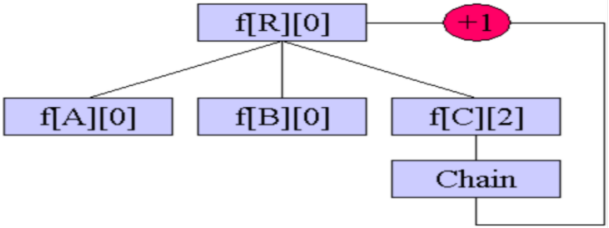


四种情况

pku1848

Case4：除 p 链不在环中 i 的各个儿子节点都在环中，i 与 p 链为环

$F[fa][0] = sum(f[son][0]) + (f[p][2] - f[p][0]) + 1$



边界情况

pku1848

i 为叶子节点时，$f[i][0], f[i][2]$ 非法
$f[son][0]$ 非法超过两次时，无法满足答案，退出
$f[son][0]$ 非法时必须选其 $f[son][1]$ 或 $f[son][2]$（无论是否合算）

逃学的小孩

NOI2003

在一棵树中，每条边都有一个长度值，现要求在树中选择 3 个点 X, Y, Z，满足 X 到 Y 的距离不大于 X 到 Z 的距离，且 X 到 Y 的距离与 Y 到 Z 的距离之和最大，求这个最大值。

逃学的小孩

NOI2003

咋看题目，类似求最长路径问题

逃学的小孩

NOI2003

咋看题目，类似求最长路径问题

思考：如果 x, y, z 在同一条链上，则：

一：x 与 z 在 y 的同一侧时 $s = xy + zy > max(xy\ zy\ xz)$

二：x 与 z 在 y 的异侧时 $s = xy + zy = xz = max(xy\ zy\ xz)$

故 x 必在 y, z 之间。又：如果 x 从一条分链上走到 y, z 路径上一点 A 则更优

逃学的小孩

NOI2003

朴素算法 $s = XA + YA + YZ = XA + 2YA + ZA$
又：$YA = XY - XA < XZ - XA = ZA$
在 A 确定,X, Y 允许互换的情况下 $XA < YA$ 比 $XA > YA$ 更优
为此枚举分叉点 $o(n)$ 似乎不错；但由于每次以分叉点为根计算复杂度为 $o(n)$，总的时间复杂度为 $o(n^2)$

逃学的小孩

NOI2003

两遍 dfs，第一遍后根遍历以任一点为根节点，记录其不再同一子树路径的最大值与次大值，并记录其由谁转移

第二遍先根遍历，转移最值同时处理答案：

Case 父节点的最大值不由该子树提供：$ans = fmax[fa] + fmax[son] + fmid[son] + w$

Case 父节点的最大值由该子树提供：$ans = fmid[fa] + fmax[son] + fmid[son] + w$

之后记得用父节点最值更新子节点最值 $f[son] = f[fa] + w, g[son] = fa$

Apple Tree

pku2486

给定一棵 $n(n \leq 100)$ 个节点的树，每个节点上有一个值
从 root 出发，走 $k(k \leq 200)$ 步，每步必须往其相邻的点走，每到一个节点将此节点上的值累加起来（不能重复累加）
问最多可以得到多大的累加值

Apple Tree

pku2486

易证：对于每棵子树我们只需进入一次；对于每条路径，我们最多往返各一次

定义 $f[i][j][0,1]$

$f[i][j][0]$ 表示从 i 点向其子树走 j 步并最终返回 i 点可得苹果数，$f[i][j][1]$ 表示从 i 点向其子树走 j 步不需返回 i 点可得苹果数

Apple Tree

pku2486

对于叶子节点：$f[i][0][0] = f[i][0][1] = apple[i]$

对于其他节点：$F[fa][j][0] =$ { 选 k 个子树往返即 $f[sonx][i][0]$, 在各子树共走 $j - 2 * k$ 步 }

$F[fa][j][1] =$ { 选 $k-1$ 个子树往返即 $f[sonx][i][0]$, 另一子树不需返回, 在各子树共走 $j - 2 * k + 1$ 步 }

River

IOI2004

在 Byteland 国，有 n 个伐木的村庄，这些村庄都座落在河边
所有的河水都汇聚并流进了一条大河，最后这条大河流进了大海
这条大河的入海口处有一个村庄名叫 Bytetown
在 Bytetown，有一个巨大的伐木场，它处理着全国砍下的所有木料。木料被砍下后，顺着河流而被运到 Bytetown 的伐木场。
国王决定，为了减少运输木料的费用，再额外地建造 k 个伐木场。这 k 个伐木场将被建在其他村庄里。这些伐木场建造后，木料就不用都被送到 Bytetown 了，它们可以在运输过程中第一个碰到的新伐木场被处理。
显然，如果伐木场座落的那个村子就不用再付运送木料的费用了。它们可以直接被本村的伐木场处理。

River

IOI2004

注意：所有的河流都不会分叉，也就是说，每一个村子，顺流而下都只有一条路——到 bytetown

国王的大臣计算出了每个村子每年要产多少木料，你的任务是决定在哪些村子建设伐木场能获得最小的运费

其中运费的计算方法为：每一块木料每千米 1 分钱

计算最小的运费

$n \leq 100, k \leq 50$

每个村庄每年生产的木料数 ≤ 10000

每个村庄到下游最近的村庄距离 ≤ 10000

Apple Tree

IOI2004

- 朴素算法：列出一个自然方程
- If（$f[son][i][p]$ 存在）$f[fa][k][w] = f[son][i][p] + f[fa][k-i][w-p] + p * w$
- 明显超时超空间
- 思考：目前村庄、建场数、未处理木块、费用，似乎一个也不能少呀。。

Apple Tree

IOI2004

- “未来型动态规划”
- 在状态里设出当前结点祖先中最近的伐木场
- 这样每 DP 一个结点就可以计算当前节点木料的运输成本

01 背包问题

有若干物品，每个物品有一个体积和价值，你有一个指定容量的背包，求能够装的物品的最大价值。

01 背包问题

有若干物品，每个物品有一个体积和价值，你有一个指定容量的背包，求能够装的物品的最大价值。

$f[i][j]$ 表示只考虑前 i 个物品，用 j 单位容量能装物品的最大价值。

Algorithm 2 01 背包问题

$f[0\cdots n][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i][j] \leftarrow \max(f[i-1][j], f[i-1][j-v_i] + w_i)$
  end for
end for

空间优化

Algorithm 3 滚动数组

$f[0\cdots1][0\cdots m]\leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i\&1][j]\leftarrow \max(f[1-(i\&1)][j], f[1-(i\&1)][j-v_i]+w_i)$
  end for
end for

Algorithm 4 滑动数组

$f[0\cdots m]\leftarrow 0$
for i from 1 to n **do**
  for j from m downto v_i **do**
    $f[j]\leftarrow \max(f[j], f[j-v_i]+w_i)$
  end for
end for

完全背包问题

有若干类物品，每类物品有一个体积和价值，数量有无限个，你有一个指定容量的背包，求能够装的物品的最大价值。

完全背包问题

有若干类物品，每类物品有一个体积和价值，数量有无限个，你有一个指定容量的背包，求能够装的物品的最大价值。

$f[i][j]$ 表示只考虑前 i 个物品，用 j 单位容量能装物品的最大价值。

Algorithm 6 01 背包问题

$f[0\cdots n][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i][j] \leftarrow \max(f[i-1][j], f[i][j-v_i] + w_i)$
  end for
end for

空间优化

Algorithm 7 滚动数组

$f[0\cdots1][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i\&1][j] \leftarrow \max(f[1-(i\&1)][j], f[i\&1][j-v_i]+w_i)$
  end for
end for

Algorithm 8 滑动数组

$f[0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[j] \leftarrow \max(f[j], f[j-v_i]+w_i)$
  end for
end for

多重背包问题

有若干类物品，每类物品有一个体积和价值，每类物品有一定数量，你有一个指定容量的背包，求能够装的物品的最大价值

多重背包问题

有若干类物品，每类物品有一个体积和价值，每类物品有一定数量，你有一个指定容量的背包，求能够装的物品的最大价值

算法一

把一个可以使用 k 次的物品看成若干个体积和价值都为这个物品的 2 的整数次幂倍的 01 物品
显然根据二进制原理这个拆分数不超过 log 个，那么对这 log 个物品做 01 背包即可

单调队列

$$f[i][j] = \max_{t \in [0,k]} (f[i-1][j - t * v] + t * w)$$

把模 v 意义下相同的 j 放在一起做，那么问题就变成了求所有长度为 k 的区间的最大值，单调队列维护即可

依赖背包问题

有若干个物品形成一棵树，每个物品有一个体积和价值，你有一个指定容量的背包，要求如果选了某个物品，则它的父亲也必须被选择，求能装的物品的最大价值。

依赖背包问题

有若干个物品形成一棵树，每个物品有一个体积和价值，你有一个指定容量的背包，要求如果选了某个物品，则它的父亲也必须被选择，求能装的物品的最大价值。

首先我们把树的 DFS 序造出来。

转移时，如果选择了某个物品，则可以正常向后转移，如果未选择某个物品，则它的子树中不能选择任何物品，而 dfs 序上一棵子树是一段连续的区间，我们直接跳过这个区间转移到之后的状态即可。

树形背包复杂度

$f[i][j]$ 表示在 i 的子树中的最大收益，每次的转移就是合并两棵子树
假设节点 i 的每个孩子的子树大小为 $s_1 \cdots s_n$，则转移复杂度为

$$\begin{aligned}
\sum_{i=2}^{n} s_i \sum_{j=1}^{i-1} s_j &= \sum_{i=1}^{n} \left(s_i \times \sum_{j=1}^{n} s_j - s_i^2 \right) \\
&\leq \sum_{i=1}^{n} \left(s_i \sum_{j=1}^{n} s_i - s_i^2 \right) \\
&= \sum_{i=1}^{n} \left(s_i \times sum - s_i^2 \right) \\
&= sum^2 - \sum_{i=1}^{n} s_i^2
\end{aligned}$$

数位 DP

数位 DP 是一类计数问题的统称，它要求记录在一个大小范围内的数/字符串的某种权值。
比如最常见的数位 DP 是：求在一段区间 [l,r] 内满足一定条件的数的个数。
首先，面对这种问题一般都会先把两个限制拆成一个，求 [0,r] 内满足条件的个数减去 [0,l-1] 内满足条件的个数。
对于这种只有上限的数位 DP，有两种基本的方法。

方法一

我们把上限 n 先数位分解为 $n[0..m]$，从高往低位扫，扫到第 k 位时我们统计以 $n[k+1\cdots m]$ 为前缀，且第 k 位严格小于 n[k] 的数的信息和。如果这项信息能够 O(1) 直接计算就可以带入，如果不能，可以先预处理一个数组 f[i][j] 表示最高位 a[i]=j，且 $0\sim i-1$ 位都任意填的所有数的信息和。

方法二

方法 1 有一定的局限性，它要求对于 $n[k+1..m]$ 这些数位的信息要能够和 $0..k-1$ 位的信息快速合并。还有另一种记忆化搜索的方式可以有效解决这个问题。我们从高位往低位搜索，每次枚举某一位填的数，然后用 $pre[i][S]$ 表示 $0 \sim i$ 位任意填，同时高位的数的信息状态为 S 的信息总和。在 dfs 时我们还加一个 bool 参数 t，表示当前枚举的高位是否严格等于 n。如果 t 为 0，那么可以对这一部分记忆化。如果 t 为 1，那么可以递归下去。我们注意到最多只有 o(位数) 个状态的 t=1，所以这一部分做一次的复杂度是 o(位数) 的，其它部分进行记忆化，复杂度是 O(状态数) 的

Problem

我们称一个正整数 n 是好的，当且仅当 n 能被他的所有数位整除。现在需要计算在一个给定的区间 [l, r] 中的好数的个数

状态表示方法

- 用一个整数 s 表示一个集合，第 k 个数用 s 的第 k 位来表示

状态表示方法

- 用一个整数 s 表示一个集合，第 k 个数用 s 的第 k 位来表示
- 用 $s >> k\&1$ 来的值来判断 k 是否属于集合 s

状态表示方法

- 用一个整数 s 表示一个集合，第 k 个数用 s 的第 k 位来表示
- 用 $s >> k\&1$ 来的值来判断 k 是否属于集合 s
- 添加一个数：$s| = 1 << k$

状态表示方法

- 用一个整数 s 表示一个集合，第 k 个数用 s 的第 k 位来表示
- 用 $s >> k\&1$ 来的值来判断 k 是否属于集合 s
- 添加一个数：$s| = 1 << k$
- 删除一个数：$s = s\ xor\ 1 << k$

状态表示方法

- 用一个整数 s 表示一个集合，第 k 个数用 s 的第 k 位来表示
- 用 $s >> k\&1$ 来的值来判断 k 是否属于集合 s
- 添加一个数：$s| = 1 << k$
- 删除一个数：$s = s\ xor\ 1 << k$
- 判断 s' 是否是 s 的子集：$s'\&s == s'$

哈密顿路径

求 n 个点 m 条无向带权边的图的最短哈密顿路长度

$n \leq 15, m \leq \frac{n(n-1)}{2}$

哈密顿路径

$F_{s,k}$ 代表当前到了第 k 个点，已经走过的点集为 s 的路径条数
转移的时候枚举上一步从 k 走到哪个点转移

$$F_{s,k} = \min_{(t,k)\in E, t\in S, t\neq k} F_{s-(1<<k),t} + w_{k,t}$$