什么是动态规划?

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。

这不是等于什么都没说吗?

一般说来，满足以下的几个条件就可以算得上是动态规划:

最优子结构性质

不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的各个决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。

无后效性

将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。

这还是看不懂啊。。。

还有一些性质，比如说子问题需要重叠（好家伙，又出现一个新的概念，叫做子问题），等等，越看越晕。

而且产生的疑惑反而更多了?

计数问题不是最优化问题，那么所有的计数问题都不能算动态规划?

无负权图的最短路问题也是最优化问题，能不能用动态规划?

动态规划和其他算法到底有什么区别?

我建议大家的理解方式是：动态规划 = 状态 + 转移 + 转移顺序

状态，决策和转移

要想深入理解动态规划，我们首先要知道什么是状态，什么是决策，什么是状态转移（简称转移）。

以我们熟悉的01背包为例，它的状态是:

> $dp[i][j]$: 已经讨论了前i个物品，使用了不超过j个单位的体积，最大能够取得多少价值

它的决策是:

> 对于每个物品，是拿还是不拿。

根据决策我们就可以得出状态转移:

> $dp[i][j] = max(dp[i-1][j], dp[i-1][j-a[i]] + b[i]);$ //$a[i]$表示第i个物品的体积，$b[i]$表示第i个物品的价值

动态规划的状态就是**某种特定的条件**。比如限定"前i个物品，使用的体积不超过j"就是这样的限定条件。它也可以称作**子问题。状态本身必须是无后效性的，状态值在所有转移完成后必须是最优的。**

状态转移把一个状态和另一个状态联系起来。通过枚举每个决策的状态，就可以从状态的定义自然推出状态转移。

枚举决策一般有两种形式。

第一种是我们比较常见的形式：通过枚举通过何种决策能够变成当前这个状态，得出状态转移。**（其他状态->当前状态）**

> 如何能够得到“前i个物品，使用的体积不超过j”这个状态呢？只需要讨论第i个物品是不是被选中了。
>
> 如果第i个物品没有被选中，那么就从“前$i-1$个物品，使用的体积不超过j”转移过来，此时的价值是$dp[i-1][j]$
>
> 如果第i个物品已经被选中了，那么就从“前$i-1$个物品，使用的体积不超过j-a[i]”转移过来，此时的价值是$dp[i-1][j-a[i]]+b[i]$
>
> 因此得到状态转移方程：
>
> $$dp[i][j]=max(dp[i-1][j],dp[i-1][j-a[i]]+b[i]);$$

另外一种是相对不常见的形式，但有时会很有用：通过枚举当前状态能够通过决策到达其他的什么状态，得出状态转移。**（当前状态->其他状态）**

> 我现在已经计算完“前i个物品，使用的体积不超过j”这个状态了，它能够推出其他的什么状态呢？
>
> 既然前i个物品已经讨论完了，接下来应该讨论第$i+1$个物品。
>
> 如果接下来不选第$i+1$个物品，那么就可以转移到“前$i+1$个物品，使用的体积不超过j”这个状态，此时的价值是$dp[i][j]$
>
> 如果接下来选第$i+1$个物品，那么就可以转移到“前$i+1$个物品，使用的体积不超过$j+a[i+1]$”这个状态，此时的价值是$dp[i][j]+b[i+1]$
>
> 所以得到状态转移方程：（注意初始化）
>
> $$dp[i+1][j]=max(dp[i+1][j],dp[i][j]);$$
>
> $$dp[i+1][j+a[i+1]]=max(dp[i+1][j+a[i+1]],dp[i][j]+b[i+1]);$$

转移顺序

思考这样一个问题：

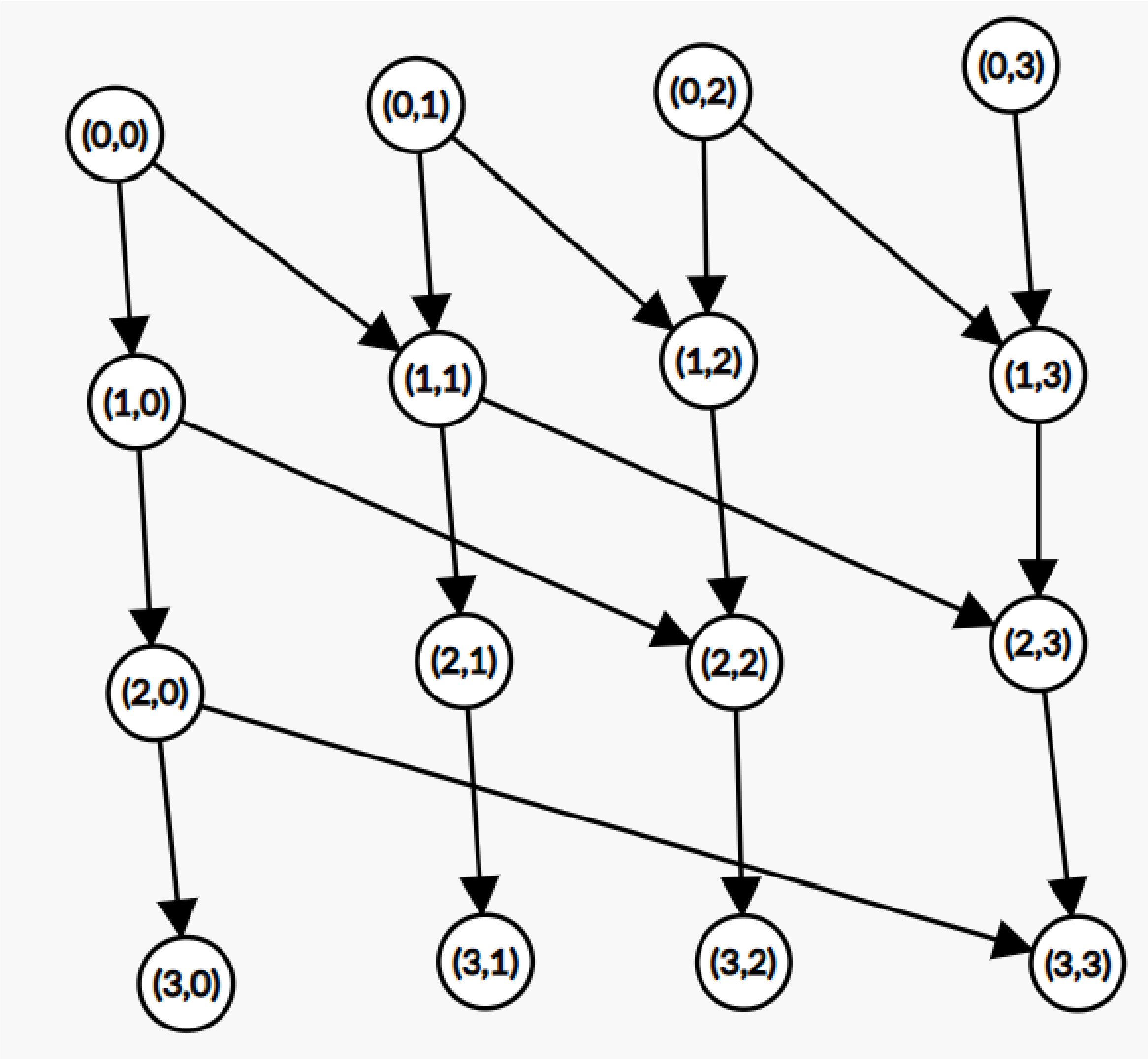
> 题目要求一个带权有向图中，从S到T的最短路。
>
> 定义$dp[i]$代表从i到T的最短路。dp数组初始化为正无穷，但是$dp[T]$初始化为0。
>
> 枚举决策：要从i走到T，显然需要先走到一个和i相邻的点。因此枚举i的所有出边，得到：
>
> $dp[i]=min\{dp[j]+w(i,j)\};$
>
> 其中j枚举所有与i相邻的点，$w(i,j)$代表从i到j的边权。
>
> 请问这个做法能实际解决问题吗？

把动态规划的状态理解成点，转移理解成有向边，那么它**必须是一个有向无环图。**

动态规划的转移顺序必须是**这个有向无环图的其中一种拓扑序。**

> 简单理解就是：只能用完全算好的状态推出其他状态。

仍然用01背包举例，如果背包体积为3，物品有3个，第1个物品的体积为1，第二个物品的体积为2，第三个物品的体积为3，它的转移图是：（为了方便绘图，省略了价值）



看图可以发现，刚才我们讨论的枚举的两种形式，实际就是枚举每个点的入度还是出度。它们对应的状态图当然是没有区别的。

请思考：把01背包的转移写成下面两种形式，它们都是正确的吗？

```
for (int i = 1; i <= n; i++)
    for (int j = 0; j <= V; j++)
        if (j >= a[i]) dp[i][j] = max(dp[i-1][j], dp[i-1][j-a[i]] + b[i]);

for (int j = 0; j <= V; j++)
    for (int i = 1; i <= n; i++)
        if (j >= a[i]) dp[i][j] = max(dp[i-1][j], dp[i-1][j-a[i]] + b[i]);
```

让我们看一个更复杂的例子：

（洛谷P1880石子合并，略改）

> 有n堆石子排成一排，每次可以合并两堆相邻的石子，得到的分数是这两堆石子个数之和。经过$n-1$操作后，所有的石子合并为一堆。求最大可能的得分。
>
> 做法：定义$dp[L][R]$的含义是，合并区间$[L,R]$所有的石子，能够得到的最大分数。
>
> 枚举最后一步的合并，它将“$[L,k]$合并之后的一堆石子”与“$[k+1,R]$合并之后的一堆石子”合并到了一起。
>
> 所以得到状态转移方程：

> $dp[L][R] = max\{dp[L][k] + dp[k+1][R] + sum(L,R)\};$
>
> 其中$L \le k < R$，$sum(L,R)$代表第L堆石子到第R堆石子的个数总和。
>
> 如何确定转移顺序?

如何思考动态规划

动态规划的核心就是状态定义。有了状态定义就能推出转移，有了状态定义和转移就能知道转移顺序。即使转移需要优化，那也是需要先有这个定义才能继续向下想。

首先需要纠正几个误区：

状态就是一张表格?

不要把状态理解成表格！

把状态理解成有向无环图，不是把它想得更复杂了，它反而是简化思维的模型。我们心里**不需要想出整张有向无环图的全貌**，我们只需要想出状态（对应有向无环图里的点）和转移（对应有向无环图里的入度，有时是出度）就可以了。

应该从边界状态开始思考问题?

动态规划的实际运行逻辑确实就是从边界状态开始，但如果我们也按照这种方式思考，那就舍本逐末了！

只需要想清楚状态定义，其实边界往往是最简单的，也经常放到最后考虑。

当然考虑要周全，边界条件确实是易错点。

看到产生后效性或不满足最优子结构了，就直接放弃这个思路?

很多时候状态产生后效性固然是因为设计状态时考虑不周，但也不代表所有的有后效性的状态定义都没有进一步思考的价值。比如下面的例子：

> 问题：在数组中不能选择任何相邻的数，求选出的数的最大总和。
>
> $dp[i]$: 前i个数选出的最大总和。（有后效性）
>
> $dp[i][0/1]$: 前i个数，其中第i个数不选/选，此时的最大总和。（无后效性）
>
> $dp[i]$: 前i个数，一定会选择最后一个数，此时的最大总和。（无后效性）

> 问题：01背包，但是需要价值$\%m$之后最大。
>
> $dp[i][j]$: 前i个物品，花费了j个单位的体积，得到的最大价值。（不满足最优子结构）
>
> $dp[i][j][k]$：前i个物品，花费了j个单位的体积，得到的价值$\%m = k$，是否可以取到。（满足最优子结构）

必须从学过的或熟悉的模型开始联想?

能联想到自然最好，联想不到不要强求。学习动态规划的模型是让你熟悉动态规划的思想和常见技巧，不是让你生搬硬套的。

下面给出思考动态规划问题的基本思维导图：

- 动态规划
  - 如何想到这道题是动态规划？
    - 做题经验
      - 看数据范围感觉是动态规划？
      - 看上去不太能用贪心等做法，所以是动态规划？
    - 知识积累
      - 见过类似的题目
      - 想到了学习过的模型
  - 设计状态
    - 直接法：从需要求解的问题出发
    - 间接法：从题目给出的其他条件或性质出发
  - 这个状态代表什么含义？
    - 它对计算答案有什么帮助？
    - 它是无后效性的吗？
      - 重新思考状态
      - 想办法改进状态
      - 或许这道题不是动态规划？
  - 有什么其他的状态能够转移到这个状态？
    - 转移的时候如何计算和更新状态值？
    - 如何安排转移顺序？
  - 这个做法能通过吗？
    - 空间复杂度
    - 时间复杂度
      - 总转移数 × 平均转移代价
      - 状态数 × 平均转移数 × 平均转移代价
    - 计算之后发现不能通过
      - 状态有优化空间？
      - 转移有优化空间？
      - 转移代价有优化空间？
      - 这题其实不是动态规划？
    - 是否需要输出方案？
      - 输出任意方案？
      - 输出最小字典序/题目要求的其他顺序？
        - 是否需要重新考虑转移顺序？
  - 代码实现
    - 边界和初始化？
    - 使用何种实现方式？
      - 循环
      - 记忆化搜索
      - 拓扑排序