



程序设计思维与实践

Thinking and Practice in Programming

动态规划（一） | 内容负责：尹浩飞/魏安然



1

动态规划热身

Dynamic Programming Warm-Up

动态规划热身

- 例题1.爬台阶问题：
 - TT想去图书馆学习，他现在要爬台阶进去，其中：一共有 n 级台阶，一步可以走 1 阶或 2 阶，问走到第 n 阶有多少种方案。
 - $n \leq 1000000$
 - $n = 2, \text{ ans} = 2$
 - 1+1
 - 2
 - $n = 4, \text{ ans} = 5$
 - 1+1+1+1
 - 1+1+2
 - 1+2+1
 - 2+1+1
 - 2+2

动态规划热身

- 例题1.爬台阶问题：
 - 比较简单的递推问题。
 - 定义 f_i —— 表示走到第 i 级的方案数，可以有递推式
 - $f_1 = 1, f_2 = 2$
 - $f_i = f_{i-1} + f_{i-2}, i \geq 3$
 - 这好像是 —— 斐波那契数列？
 - 所以我们可以 $O(n)$ 去求斐波那契数列第 n 项，
也可以 $O(\log n)$ 去求（矩阵乘法快速幂，后续课程会讲）

动态规划热身

- 例题1.爬台阶问题 $f_i = f_{i-1} + f_{i-2}, i \geq 3$
 - 对于该方程，我们有两种实现
 - 方案1 —— 递归：

```
1 int solve(int n){
2     if(n == 1)return 1;
3     if(n == 2)return 2;
4     return solve(n - 2) + solve(n - 1);
5 }
```

- 方案2 —— 递推：

```
1 int solve(int n){
2     f[1] = 1;
3     f[2] = 2;
4     for(int i=3;i<=n;++i)
5         f[i] = f[i-1] + f[i-2];
6     return f[n];
7 }
```

动态规划热身

- 例题1.爬台阶问题 $f_i = f_{i-1} + f_{i-2}, i \geq 3$

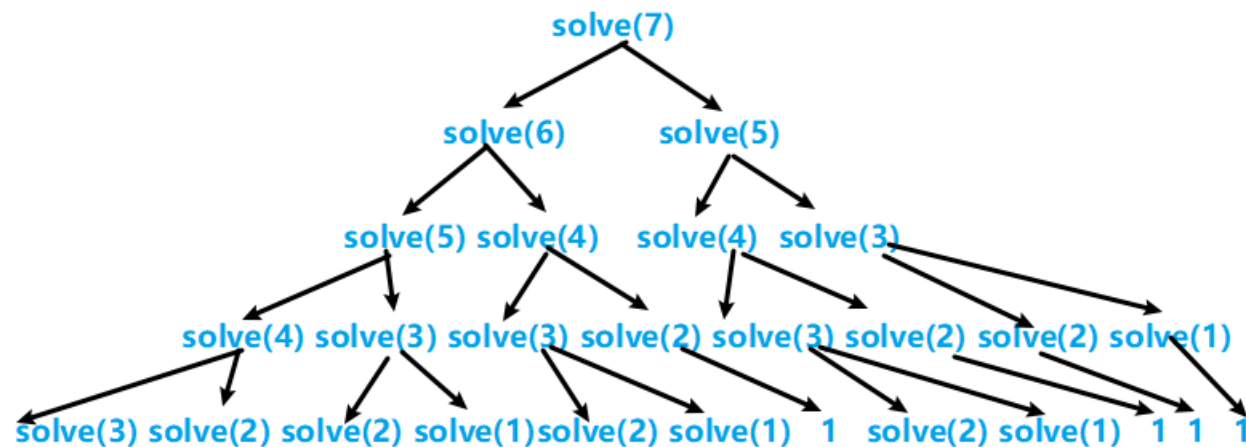
- 来说方案 1 —— 递归

```

1 int solve(int n){
2     if(n == 1)return 1;
3     if(n == 2)return 2;
4     return solve(n - 2) + solve(n - 1);
5 }

```

- 假如要求 solve(7), 会发生什么?



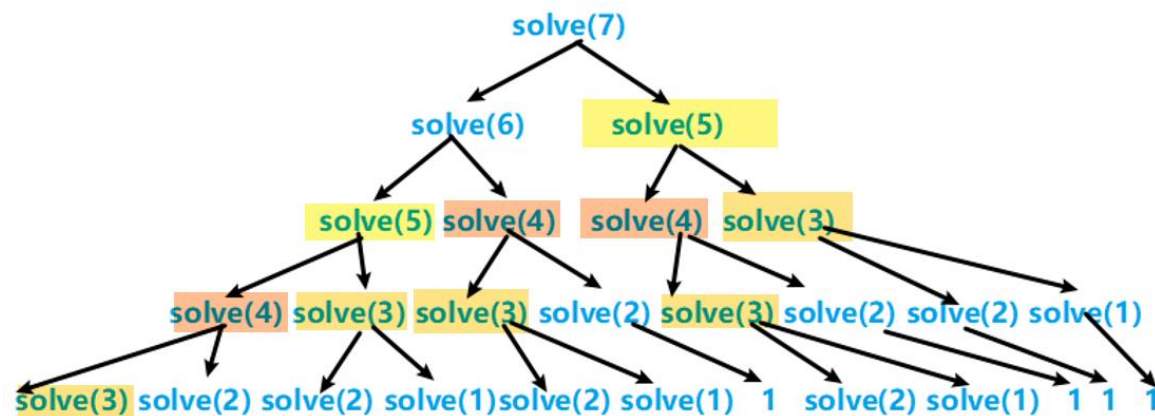
- 什么问题?

动态规划热身

- 例题1.爬台阶问题 $f_i = f_{i-1} + f_{i-2}, i \geq 3$



重复计算!!!

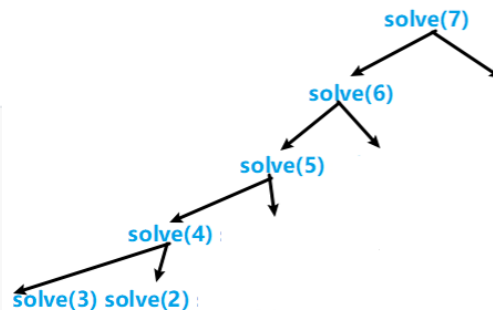


- 引入——记忆化

```

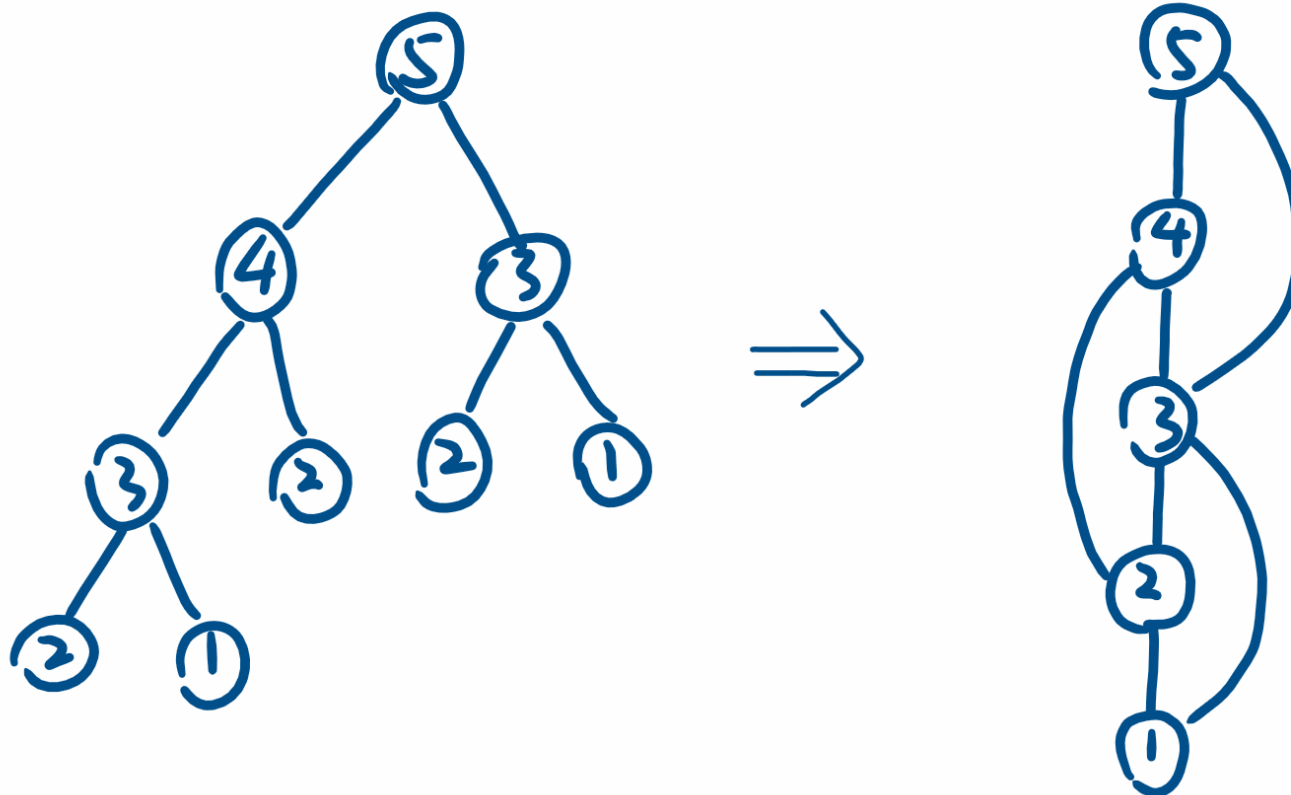
1  int f[maxn];
2
3  int solve(int n){
4      if(f[n]) return f[n]
5      if(n == 1) return f[1] = 1;
6      if(n == 2) return f[2] = 2;
7      return f[n] = solve(n - 2) + solve(n - 1);
8  }

```



动态规划热身

- 例题1.爬台阶问题：
 - 记忆化思想，避免同一状态被重复访问。



动态规划热身

- 动态规划的两种基本思想

- 带备忘的自顶向下法 (top-down with memoization)

- 按自然递归编写过程，保存每个子问题的解（通常保存在一个数组或散列表中）。当需要一个子问题的解时，首先检查是否已经保存过此解，如果是则直接返回；否则，按通常方式计算这个子问题。

- 自底向上法 (bottom-up method)

- 一般需要恰当定义子问题“规模”的概念，使得任何子问题的求解都只依赖于“更小的”子问题的求解。按从小到大的顺序依次进行求解子问题。当求解某个子问题时，它所依赖的更小的子问题都已求解过。

```
1 int f[maxn];
2
3 int solve(int n){
4     if(f[n]) return f[n]
5     if(n == 1)return f[1] = 1;
6     if(n == 2)return f[2] = 2;
7     return f[n] = solve(n - 2) + solve(n - 1);
8 }
```

```
1 int solve(int n){
2     f[1] = 1;
3     f[2] = 2;
4     for(int i=3;i<=n;++i)
5         f[i] = f[i-1] + f[i-2];
6     return f[n];
7 }
```

——《算法导论》第15章

动态规划热身

- 例题2. 爬台阶问题 II (作业) :

- 还是TT, 他要去 N3 楼做实验, 要爬台阶到三楼, 其中一共有 n 级台阶, 一步可以走 1, 2, 3, ..., k 阶, 诶, 同时有些台阶不能落脚, 问走到第 n 阶的方案数。
- $n \leq 1000000$
- $n = 4$, 不能落脚的台阶 = [2], $k = 2$, $\text{ans} = 1$
 - $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$
- $n = 5$, 不能落脚的台阶 = [2], $k = 3$, $\text{ans} = 5$
 - $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$
 - $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$
 - $0 \rightarrow 1 \rightarrow 4 \rightarrow 5$
 - $0 \rightarrow 3 \rightarrow 4 \rightarrow 5$
 - $0 \rightarrow 3 \rightarrow 5$

动态规划热身

- 例题2. 爬台阶问题 II (作业) :
 - 思考一会儿...
 - 定义 f_i —— 表示走到第 i 级的方案数, 可以有
 - 初始化: $f_0=1$
 - 转移过程: $f_i = \sum f_j (i - k \leq j < i, j \text{ 台阶正常})$
 - 对于不正常的台阶: $f_i = 0$

动态规划热身

- 例题3. 最大区间和：
 - N 个数字，每个数字 $\in [-1e9, 1e9]$ ，TT想选定一个区间使得和最大，求这个和。
 - $[1, 2, 3, 4, 5] \rightarrow [1, 2, 3, 4, 5]$
 - $[1, 2, -100, 4, 5] \rightarrow [1, 2, -100, 4, 5]$
 - $n \leq 1000000$

动态规划热身

- 例题3. 最大区间和：
 - N 个数字，每个数字 $\in [-1e9, 1e9]$ ，TT想选定一个区间使得和最大，求这个和。
 - $[1, 2, 3, 4, 5] \rightarrow [1, 2, 3, 4, 5]$
 - $[1, 2, -100, 4, 5] \rightarrow [1, 2, -100, 4, 5]$
 - $n \leq 1000000$
 - 贪心： $O(n^3) \rightarrow O(n^2) \rightarrow O(n)$
 - $O(n^2)$ ：维护前缀和 $sum[]$ ，假如我们选了 $[L, R]$ ，那么答案就是：
 - $ans = sum[R] - sum[L-1]$
 - $O(n)$ ：观察上述答案表达式，若固定 R 不变，则让 ans 最大，则可让 $sum[L-1]$ 最小 ($1 \leq L-1 < R$)，这个值无需每次遍历，维护即可。

动态规划热身

- 例题3. 最大区间和：
 - 不用贪心，用动态规划的思想做，怎么列方程？
 - 定义 dp_i —— 表示 i 为区间右端点时，可以取到的最大和
 - $dp_i = \max(dp_{i-1} + a_i, a_i)$
 - 要么就取 a_i / 要么把 a_i 与前面的并在一起

动态规划热身

- 例题4. 最大区间和 II:
 - 在 例题3 的基础上, π 会一个魔法, 使得其中一个数变成 x , 如果这个魔法至多能使用 1 次, 求答案。
 - $x = 100$, $[1, 2, 3, 4, 5] \rightarrow [100, 2, 3, 4, 5]$
 - $x = 100$, $[1, 2, -100, 4, 5] \rightarrow [1, 2, 100, 4, 5]$
 - $n \leq 1000000$

动态规划热身

● 例题4. 最大区间和 II:

- 在 例题3 的基础上, π 会一个魔法, 使得其中一个数变成 x , 如果这个魔法至多能使用 1 次, 求答案。

- $x = 100, [1, 2, 3, 4, 5] \rightarrow [100, 2, 3, 4, 5]$

- $x = 100, [1, 2, -100, 4, 5] \rightarrow [1, 2, 100, 4, 5]$

- $n \leq 1000000$

- 定义 $dp_{i,0}$ —— 表示 i 为右端点时, 并且没用魔法, 可以取到的最大和

$dp_{i,1}$ —— 表示 i 为右端点时, 并且用了魔法, 可以取到的最大和

$$\begin{cases} dp_{i,0} = \max(dp_{i-1,0} + a_i, a_i) \\ dp_{i,1} = \max(dp_{i-1,1} + a_i, a_i, dp_{i-1,0} + X, X) \end{cases}$$

- 实现起来就是一个二维数组 $dp[n][2]$ 。答案是 $\max\{dp[i][0], dp[i][1]\}$



2

动态规划

Dynamic Programming

动态规划

- 动态规划 —— 引入！
$$dp_{i,1} = \max(dp_{i-1,1} + a_i, a_i, dp_{i-1,0} + X, X)$$
 - 在前面的例题中，比如上面的方程，尤其体现了“抉择”
 - 对上一个状态的“抉择”，对最优性的“抉择”，或者说“决策”
- 动态规划的基本概念
 - 通过合理“组合”子问题的解 从而解决 整个问题解的一种算法。其中的子问题并不是独立的，这些子问题又包含有公共的子问题。
 - 动态规划算法就是对每个子问题只求一次，并将其结果保存在一张表中(数组)，以后再用到时直接从表中拿过来使用，避免重复计算相同的子问题。
 - “不做无用功”的求解模式，大大提高了程序的效率。
 - 动态规划算法常用于解决统计类问题（统计方案总数）和最优值问题（最大值或最小值），尤其普遍用于最优化问题。

动态规划

- 动态规划问题的特征：

一、最优子结构

- 如果问题的一个最优解中包含了子问题的最优解，则该问题具有最优子结构。也称最优化原理。

二、重叠子问题

- 在解决整个问题时，要先解决其子问题，要解决这些子问题，又要先解决他们的子问题.....而这些子问题又不是相互独立的，有很多是重复的，这些重复的子问题称为重叠子问题。
- 动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只解一次，而后将其解保存在一个表中，以后再遇到这些相同问题时直接查表就可以，而不需要再重复计算，每次查表的时间为常数。

三. 无后效性原则

- 已经求得的状态，不受未求状态的影响。
- 设计的状态满足拓扑关系。

动态规划

- 设计动态规划法的步骤：
 1. 分析题目特性，转化为抽象模型（找出最优解的性质）；
 2. 设计动态规划方程；
 3. 定义初始化条件；
 4. 以自底向上（递推）或带备忘的自顶向下（记忆化搜索）的方式进行计算；
 5. 统计答案并输出。

动态规划

- 动态规划的关键：
 - 设计状态：
 - 进行动态规划的基础
 - 结合动态规划的性质进行设计，注意时空复杂度
 - 状态转移方程的构造：
 - 动态规划过程中最重要的一步，也是最难的一步。
 - 通过多做题来积累经验。

动态规划

- 动态规划 与 贪心法 的不同：
 - 贪心算法： “先决策，再求解子问题”
 - 动态规划： “先求得子问题的解，然后决策”
- 动态规划的一般格式为：

```
for U取遍所有状态 do      //枚举状态
    for X取遍所有决策 do    //枚举决策
        // 状态转移
```

动态规划

- 动态规划常见模型
 - 线性型
 - 坐标型
 - 背包型
 - 区间型
 - 状态压缩型
 - 树型
 - 矩阵型



2

动态规划热身 2

Dynamic Programming Warm-Up 2

动态规划热身

- 例题5. 走地图：
 - TT在一个 $n \times m$ 的地图上，起点为左下角，终点为右上角，TT只能向右或向上行走，询问走到终点的方案数。
 - $n, m \leq 1000$
 - （其实可以通过排列组合解决，可以自行思考）
 - 假定有些点存在障碍物而不可行走，又应当如何设计？

动态规划热身

- 例题5. 走地图：
 - 定状态：定义 $f[i][j]$ 表示从起点到 (i,j) 点的方案数。
 - 状态转移方程为： $f[i][j] = f[i-1][j] + f[i][j-1]$, (当 $(i-1,j)$ 和 $(i,j-1)$ 两点合法时)
 - 初始化： $f[1][1] = 1$
 - 输出答案： $f[n][m]$

动态规划热身

- 例题6. 拿数问题（作业）：

- 给 n 个数，每一步能拿走一个数，比如拿第 i 个数， $A_i = x$ ，得到相应的分数 x ，但拿掉这个 A_i 后， $x+1$ 和 $x-1$ (如果有 $A_j = x+1$ 或 $A_j = x-1$ 存在) 就会变得不可拿（但是有 $A_j = x$ 的话可以拿这个 x ）。求最大分数。
- $n, A_i \leq 100000$

动态规划热身

● 例题6. 拿数问题（作业）：

- 给 n 个数，每一步能拿走一个数，比如拿第 i 个数， $A_i = x$ ，得到相应的分数 x ，但拿掉这个 A_i 后， $x+1$ 和 $x-1$ (如果有 $A_j = x+1$ 或 $A_j = x-1$ 存在) 就会变得不可拿（但是有 $A_j = x$ 的话可以拿这个 x ）。求最大分数。
- $n, A_i \leq 100000$
- 思维的转换，从 A 序列转换到 CNT 序列
- $dp[i]$ ，仅考虑大小为 $1..i$ 的数，能拿到的最大分数
- $dp[i] = \max(dp[i-1], dp[i-2] + CNT[i] * i)$

动态规划热身

- 例题7. TT弹钢琴：

- TT最近喜欢弹钢琴(在梦中)，一段旋律中的每个音符都可以用一个小写英文字母表示。当组成一段旋律的字符 ASCII 码是非递减的，旋律被称为是高昂的，例如 aaa,bcd.

现在TT已经学会了 n 段高昂的旋律，他想利用它们拼接处一个尽可能长的高昂的旋律，问最长长度是多少？ ($n \leq 1e6$, 字符串长度和 $\leq 1e6$)

- 5
bcdefhijk
bcd
aaa
eeeefghhh
zzzz

- 答案是 19 (aaa, bcd, eeeefghhh, zzz)

动态规划热身

- 例题7. TT弹钢琴：
 - 先按照每个字符串的最后一个字母排序。
 - 定状态：定义 $dp[i]$ 表示仅考虑前 i 个串的最大长度。
 - 状态转移方程为：
 - $dp[i] = \max\{dp[i], dp[j] + \text{str}[i].\text{size} \mid 0 \leq j < i \text{ 且 } \text{str}[j].\text{back} \leq \text{str}[i].\text{front}\}$
 - 初始化： $dp[i] = \text{str}[i].\text{size}$
 - 输出答案： $\max\{dp[i], 0 \leq i < n\}$
 - 如何优化？
 - 二维偏序

动态规划热身

- 例题8. 矩阵选数（作业）：

- 给定一个矩阵 $3 \times n$ ，比如

5 10 5 4 4

1 7 8 4 0

3 4 9 0 3

从每一列选择一个数，求出后一列减去前一列的绝对值的和的最小值

- $n \leq 1000000$

- 比如这里就是 5 4 5 4 4，使得 $|4-5|+|5-4|+|4-5|+|4-4|$ 最小，输出是 3

动态规划热身

- 例题8. 矩阵选数（作业）：
 - 发现每一列只会和前一列关联
 - 定状态：定义 $dp[i][0/1/2]$ 表示仅考虑前 i 列，并且第 i 列选的是第 $0/1/2$ 数的最小答案。
 - 状态转移方程为：
 - $dp[i][0] = \min(dp[i-1][0] + |a[0][i] - a[0][i-1]|, dp[i-1][1] + |a[0][i] - a[1][i-1]|, dp[i-1][2] + |a[0][i] - a[2][i-1]|)$
 - 其他两个转移方程 $dp[i][1]$ 、 $dp[i][2]$ 同理
 - 初始化： $dp[0][0] = INF, dp[1][0/1/2] = 0$
 - 输出答案： $\max\{dp[n][i], 0 \leq i \leq 2\}$

动态规划热身

练习题 1：山大青岛校区给同学们装空调。海信公司派出了 n 批施工队伍帮助大家装空调，每一个施工队最多负责 10 个宿舍。但是为了避免人员浪费，海信希望我们让每一个施工队所负责的宿舍尽可能多，所以承诺给予我们一定的经济补贴。

| 宿舍数目 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|------|---|---|---|---|---|----|----|----|----|----|
| 补贴金额 | 1 | 4 | 7 | 8 | 9 | 16 | 18 | 22 | 23 | 24 |

如果我们让一个施工队负责 1 间宿舍，那么补贴就是 1 元，如果我们让一个施工队负责 4 间宿舍，那么补贴就是 8 元，我们准备先给 55 间宿舍装空调，测试一下施工队的水平。现在委派你与海信进行交涉。

(1) 最少使用几个施工队，我们可以拿到多少补贴？

且工程队的数量尽可能多

(2) 我们想拿到尽可能多的补贴，需要多少个施工队？（设计动态规划算法，并给出详细的计算步骤）

动态规划热身

- 例题9. 装空调：

(2)设计状态： $dp[i]$ 表示装 i 间宿舍空调补贴的最大金额， $t[i]$ 表示装 i 间宿舍空调补贴获得最大金额的方案的施工队数目， $m[i]$ 表示安排一个施工队装 i 间宿舍空调获得的补贴（题设常量）。

初始化： $dp[0] = t[0] = 0$

状态转移： $dp[i] = \max(dp[i - j] + m[j] \mid j = 1, \dots, \min(i, 10))$ ， $i = 1, \dots, 55$ ，同时 $t[i] = t[i - j] + 1$ ， j 是使 $dp[i - j] + m[j]$ 最大的取值。此处还需考虑在补助相同的情况下，优先选择所需时间最少的方案。

输出： $dp[55]$ 、 $t[55]$ 。

结果：55间房间，尽可能多的拿补助，可拿150元补贴，需要7个施工队。

动态规划热身

● 例题10. 装风力发电机：

练习题 4 :我们要沿着道路放置风力发电机以产生能量。由于地理原因,道路上共有 n 个位置可放置发电机,但是为了更高效,两个发电机之间的距离必须至少为 D 。可放置发电机的位置 d_1, d_2, \dots, d_n 作为线上的坐标给出,其中,第 i 个位置和最左边位置的距离为 d_i ,且满足 $d_1 = 0, d_i < d_{i+1}$ 。在 i 位置风力发电机产生的能量为 $e_i > 0$ 。如何安排风力发电机的位置以产生最大化的能量?

- (1) 举一个反例说明以下贪心策略不一定得到最优解：“每次均选择一个能产生最大能量的位置,直到无法再放置风力发电机”。

(1)举例： $d = [0, 1, 2, 3], e = [4, 5, 4, 1], D = 2$ 。根据题设贪心策略,解为 $t = [\times, \sqrt{}, \times, \sqrt{}]$ (最大化能量为 $5 + 1 = 6$) , 然而最优解为 $t = [\sqrt{}, \times, \sqrt{}, \times]$ (最大化能量为 $4 + 4 = 8$)。

- (2) 设计动态规划算法,尽可能高效的得到最大化的能量。

动态规划热身

- 例题10. 装风力发电机：
 - 想一下，可能要预处理

(2)设计状态： $dp[i]$ 表示仅考虑前 i 个风力发电机可得到的最大化能量。

预处理数据： $L[i]$ 表示左边第一个离 i 位置距离大于等于 D 的位置， $L[i]$ 初始为0。（维护一个队列 Q 存储位置 i ，从 d_1 开始到 d_n 不断将位置 i 入队， d_i 入队后，不断弹出队首元素直至队首元素与队尾元素 d_i 相差小于 D ，弹出过程中只要队首元素与队尾元素 d_i 距离大于等于 D ，则更新 $L[i] = Q.head$ 。从 d_1 执行到 d_n 后，预处理结束。

初始化： $dp[0] = 0, dp[1] = e_1$

状态转移： $dp[i] = \max(dp[L[i]] + e_i, dp[i-1])$

输出： $dp[n]$ ，即为最大化能量



最长上升子序列

Longest Increasing Subsequence

经典问题

- 最长上升子序列
 - 给定 n 个整数 A_1, A_2, \dots, A_n ，按从左到右的顺序选出尽量多的整数，组成一个上升子序列。输出最长上升子序列的长度。
 - $n \leq 1000000$
 - 例如，序列 1,6,2,3,7,5
 - 上升子序列可以是 1,6,2,3,7,5；也可以是 1,6,2,3,7,5
 - 最长上升子序列为 1,2,3,5，其长度为 4，故 $\text{ans} = 4$

经典问题

- 最长上升子序列
 - 状态：定义 f_i 表示以 A_i 为结尾的最长上升序列的方程。
 - 初始化： $f_1 = 1$
 - 转移过程： $f_i = \max\{f_j \mid j < i \wedge A_j < A_i\} + 1$
 - 输出答案： $\max\{f[i], i=1\dots n\}$
 - 时间复杂度： $O(n^2)$

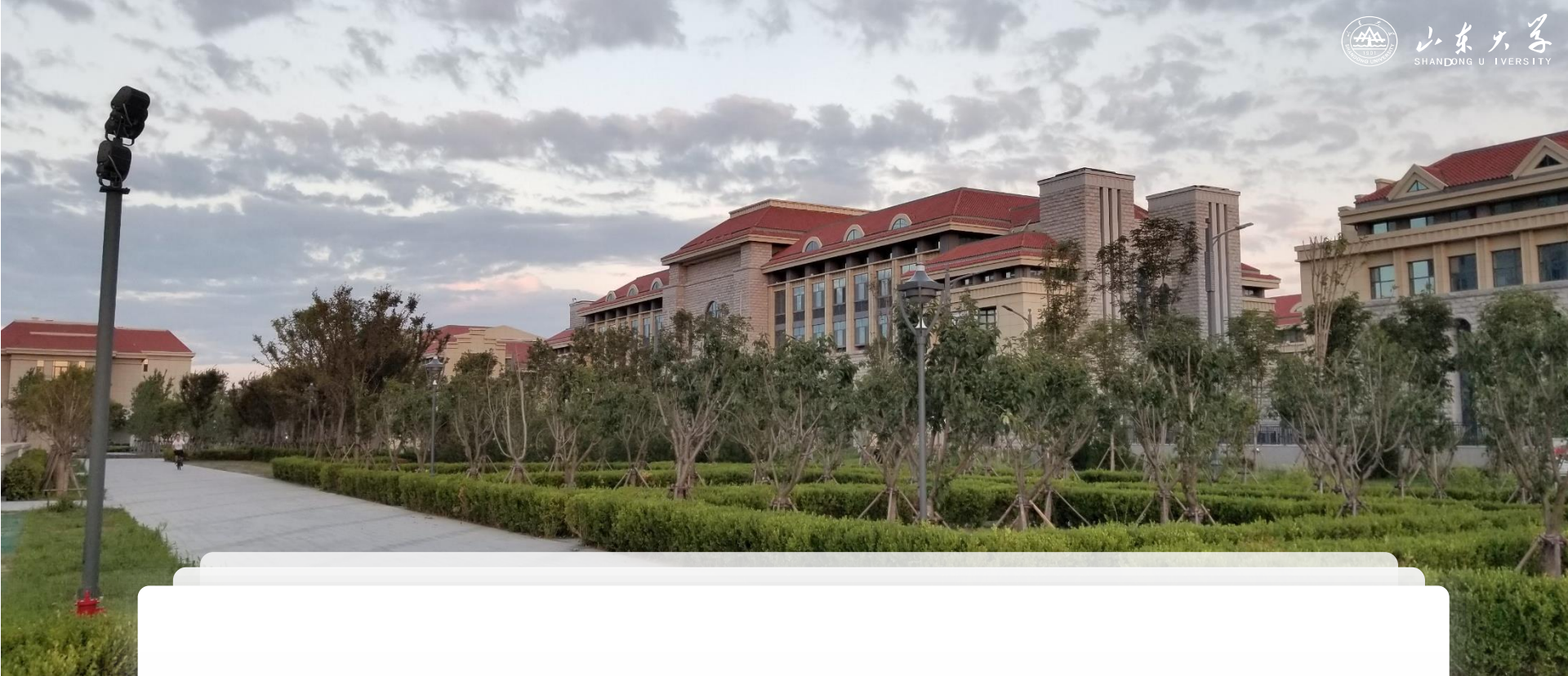
经典问题

- 最长上升子序列

- 通过数据结构的应用可以更加高效的解决当前问题。

$$f_i = \max\{f_j \mid j < i \wedge A_j < A_i\} + 1$$

- 观察上式可知，我们原本的 $O(n^2)$ 的时间复杂度，其中一个 n 就在于 j 的枚举上，这个过程相当于求解 i 之前，所有小于 A_i 的元素的 $f[j]$ 的最大值。
- 这个问题与之前介绍过的逆序对问题十分相似，也是一个**二维偏序**问题。所以可以使用**树状数组**来优化，于是算法的时间复杂度可以优化到 $O(n \log n)$ 。
- 其实除了使用树状数组进行优化，也可以借助数值关于答案的单调性，使用二分查找找到最优的答案，有兴趣的同学可以自己摸索。



5

最长公共子序列

Longest Common Subsequence

经典问题

- 最长公共子序列
 - 给两个序列 $A[1..n]$ 和 $B[1..m]$, 求长度最大的公共子序列的长度
 - $n, m \leq 5000$
 - 例如:
A – 1,5,2,6,8,7
B – 2,3,5,6,9,8,4
 - 最长公共子序列为:
A – 1,5,2,6,8,7
B – 2,3,5,6,9,8,4 或
A – 1,5,2,6,8,7
B – 2,3,5,6,9,8,4

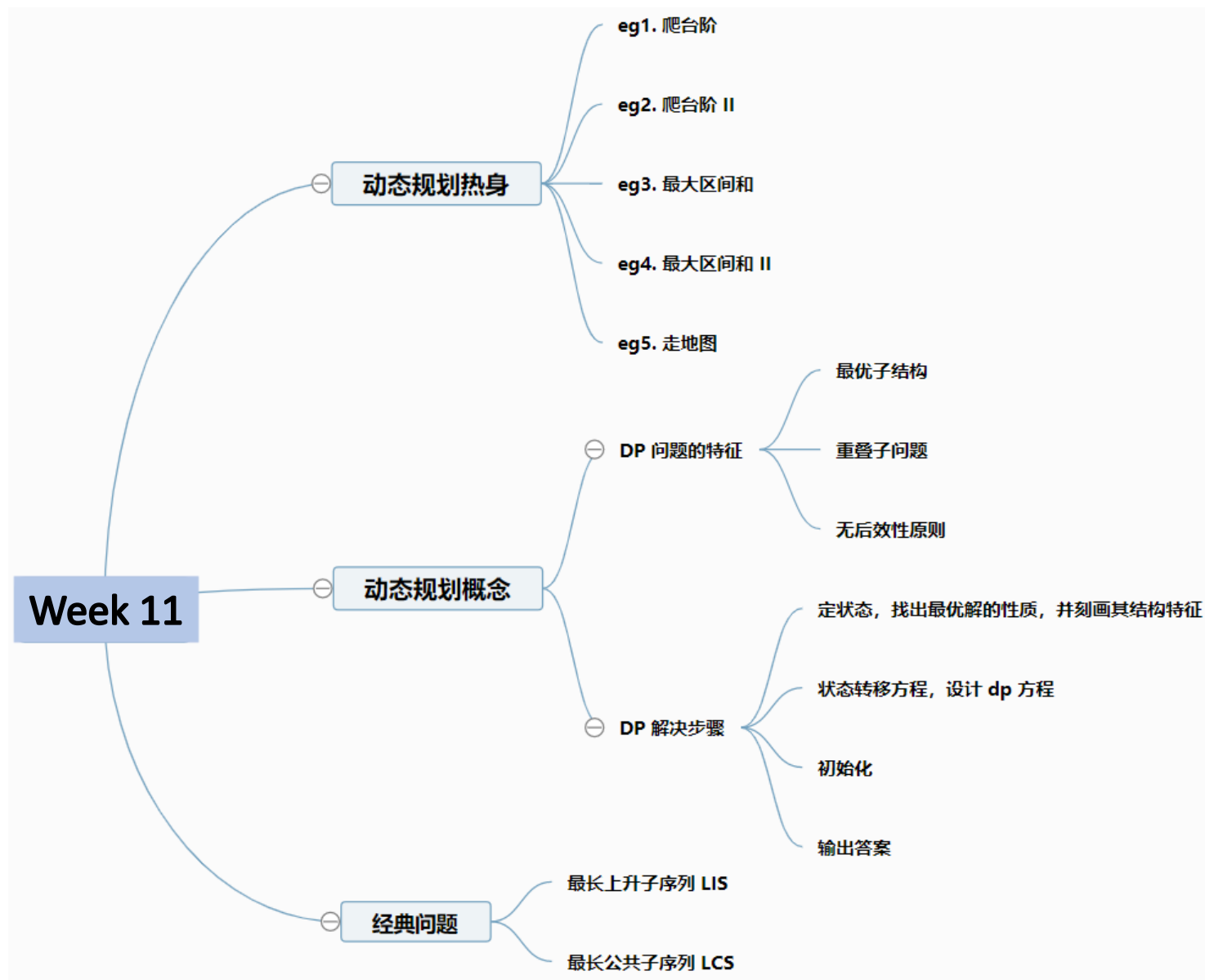
经典问题

- 最长公共子序列
 - 设计状态：假设 $f[i][j]$ 为 A_1, A_2, \dots, A_i 和 B_1, B_2, \dots, B_j 的 LCS 长度
 - 初始化：初始 $f[1][0] = f[0][1] = f[0][0] = 0$
 - 转移方程：当 $A_i == B_j$ 时， $f[i][j] = f[i-1][j-1] + 1$
 - 否则 $f[i][j] = \max(f[i-1][j], f[i][j-1])$
 - 输出答案： $f[n][m]$
 - 时间复杂度： $O(nm)$

经典问题

- 最长公共子序列
 - 在同一个序列中的内容互不相同的前提下，本问题也存在 $O(m\log n)$ 的方法，思想是转换成最长上升子序列进行处理。
 - 注：普遍的最长公共子序列问题，目前没有低于 $O(n^2)$ 的解法。
 - 有兴趣的同学可以自行阅读相关内容

总结





为天下储人才
为国家图富强

感谢收听

Thank You For Your Listening