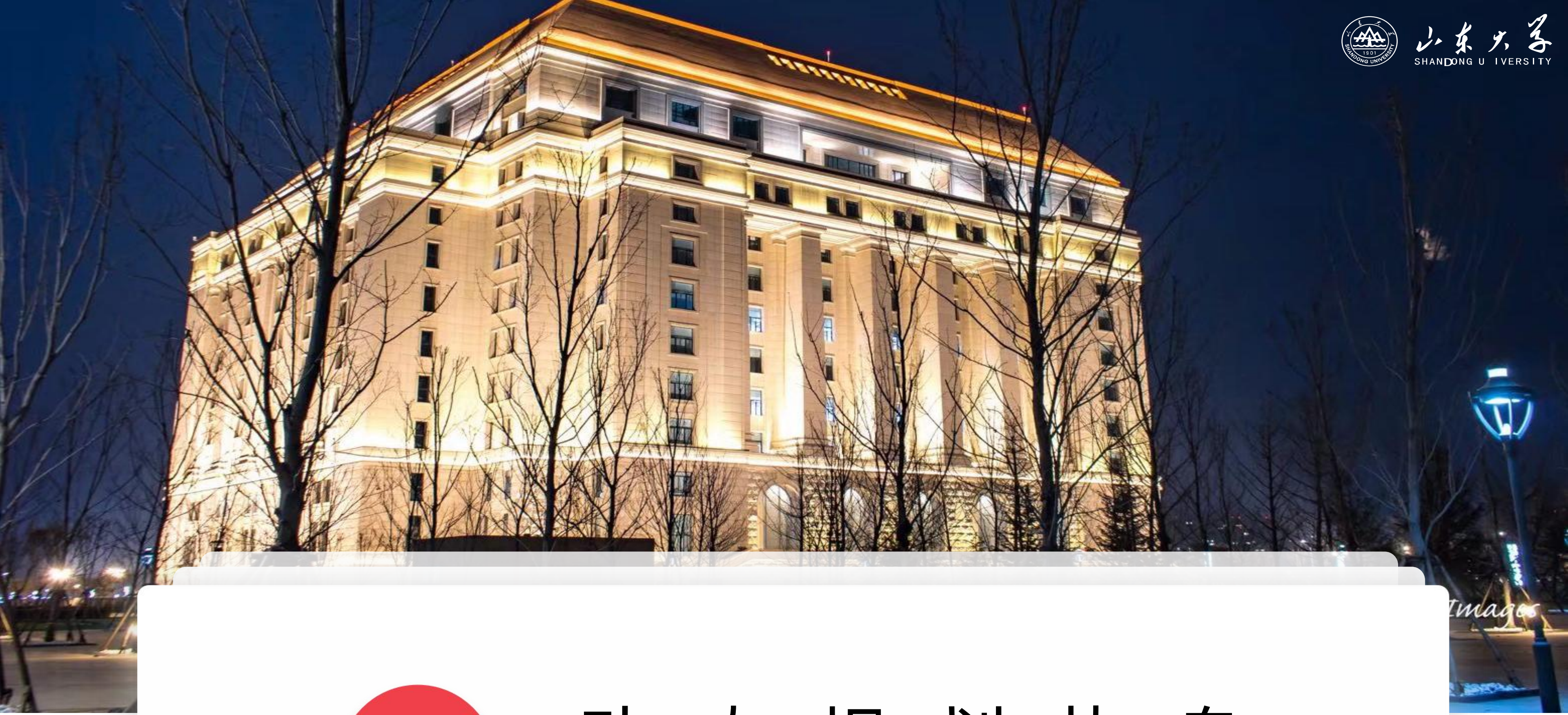
程序设计思维与实践

Thinking and Practice in Programming

动态规划（一） | 内容负责：尹浩飞

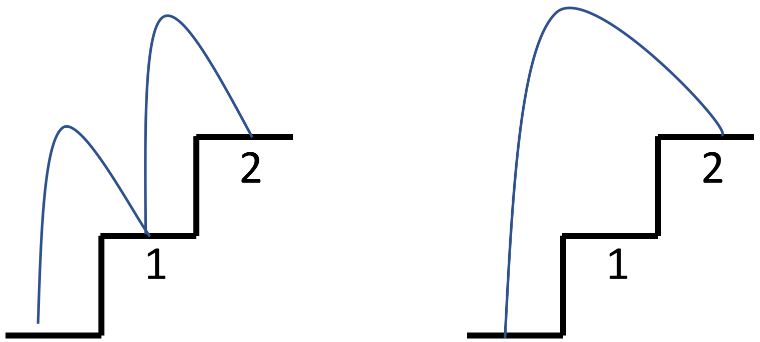


1 动 态 规 划 热 身

Dynamic Programming Warm-Up

动态规划热身

- 例题1.爬台阶问题：
  - 小 Y 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。
  - $n \le 10^6$
  - n = 2， ans = 2
    - 1+1
    - 2
  - n = 4， 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?
  - 可以 O(n) 去求斐波那契数列第 n 项
  - 可以 O(log n) 求解（通项公式或矩阵乘法快速幂）

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：

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

  - 方案 2 —— 递推：

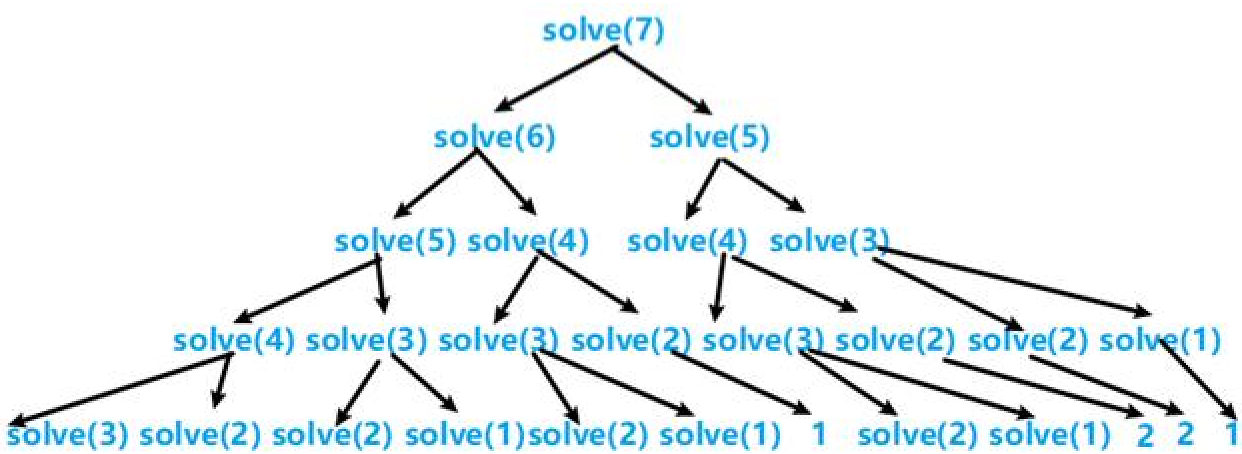
```
int solve(int n){
    f[1] = 1;
    f[2] = 2;
    for(int i=3;i<=n;++i)
        f[i] = f[i-1] + f[i-2];
    return f[n];
}
```

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

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

  - 假如要求 solve(7)，会发生什么?

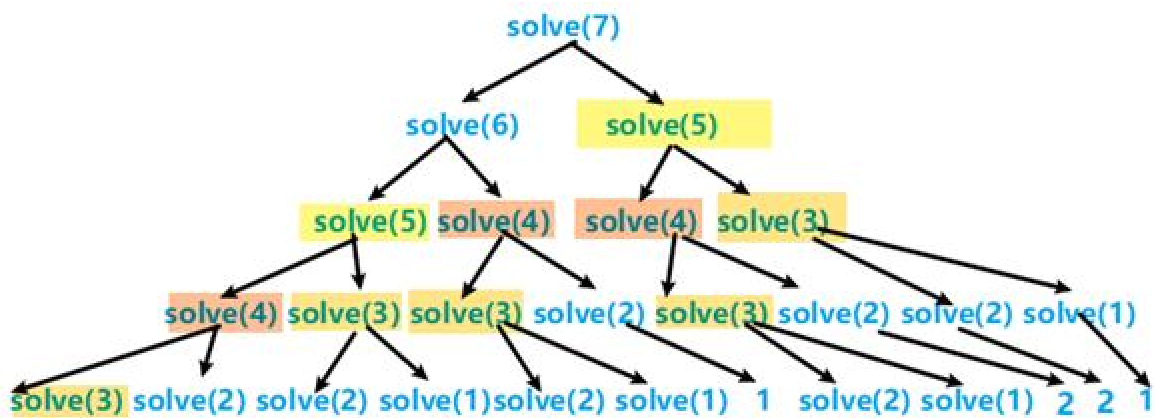


  - 什么问题?

动态规划热身

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

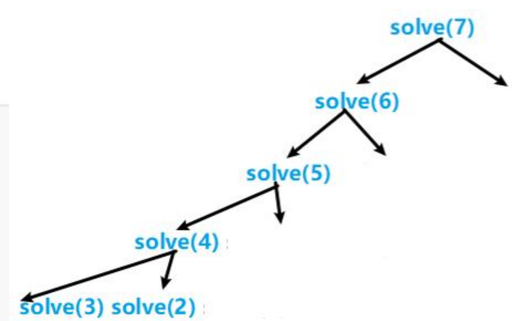
- 重复计算



- 引入 —— 记忆化

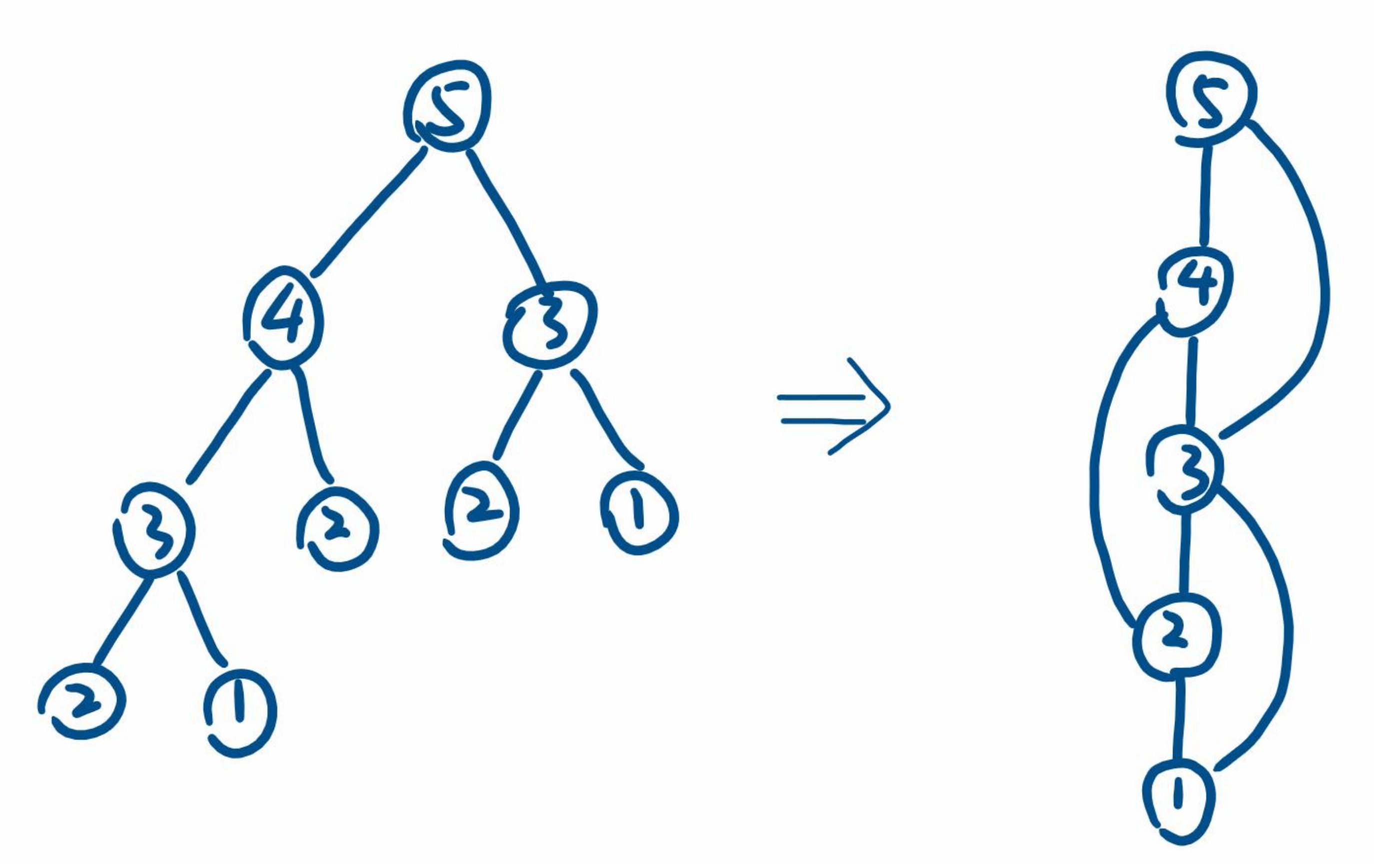
```
int f[maxn];

int solve(int n){
    if(f[n]) return f[n]
    if(n == 1)return f[1] = 1;
    if(n == 2)return f[2] = 2;
    return f[n] = solve(n - 2) + solve(n - 1);
}
```



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

```
int f[maxn];

int solve(int n){
    if(f[n]) return f[n]
    if(n == 1)return f[1] = 1;
    if(n == 2)return f[2] = 2;
    return f[n] = solve(n - 2) + solve(n - 1);
}
```

- 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。

——《算法导论》第15章

```
int solve(int n){
    f[1] = 1;
    for(int i = 1; i < n; i ++){
        f[i+1] += f[i];
        f[i+2] += f[i];
    }
    return f[n];
}
```

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

动态规划热身

- **例题2. 爬台阶问题 II（作业）：**
  - 小 Y 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，同时有些台阶不能落脚，问走到第 n 阶的方案数。
  - $n \le 10^6$

  - n = 4，不能落脚的台阶 = [2], k = 2,　ans = 1
    - 0→1→3→4

  - n = 5，不能落脚的台阶 = [2], k = 3,　ans = 5
    - 0→1→3→4→5
    - 0→1→3→5
    - 0→1→4→5
    - 0→3→4→5
    - 0→3→5

动态规划热身

- **例题2. 爬台阶问题 II（作业）：**
  - 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（使用前缀和进行维护）：

$$f_i = \sum_{j=i-k}^{i-1} f_j$$

    - 对于不正常的台阶：$f_i = 0$

动态规划热身

- **例题2. 爬台阶问题 II（作业）：**
- n = 5, k = 2

台阶编号	0	1	2	3	4	5
不能走的台阶				*		
f[]	1					
f[]的前缀和	1					

动态规划热身

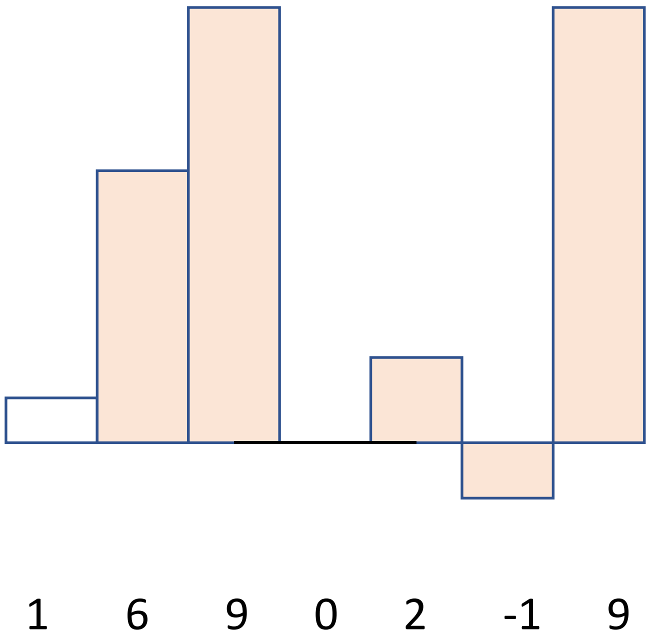
- 例题3. 最大区间和：
  - N 个数字，每个数字∈ $[-10^9,10^9]$，求最大区间和。
    - [1,2,3,4,5] → [1,2,3,4,5]
    - [1,2,-100,4,5] → [1,2,-100,4,5]
  - $n \le 10^6$

动态规划热身

- 例题3. 最大区间和：
  - N 个数字，每个数字∈ $[-10^9,10^9]$，求最大区间和。
    - [1,2,3,4,5] → [1,2,3,4,5]
    - [1,2,-100,4,5] → [1,2,-100,4,5]
  - $n \le 10^6$
  - 贪心： O(n^3) -> O(n^2) -> 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（$0 \le L-1 < R$），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：
- a = { 1 5 3 -9 2 -3 10}
- sum = {1 6 9 0 2 -1 9 }



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
    - $dp_i = \max(dp_{i-1} + a_i, a_i)$
    - 要么第 i 个数自己成为一个区间
    - 要么第 i 个数和上一个数能构成的最大区间合起来
  - dp_i 中最大值即为答案

动态规划热身

- 例题4. 最大区间和 II：
  - 在 例题3 的基础上，可以额外使用一次魔法，使得其中一个数变成 X，如果这个魔法至多能使用 1 次，求最大区间和。
    - X = 100，[1,2,3,4,5] → [100,2,3,4,5]
    - X = 100，[1,2,-100,4,5] → [1,2,100,4,5]
  - $n \leq 10^6$

动态规划热身

- 例题4. 最大区间和 II：
  - 在 例题3 的基础上，可以额外使用一次魔法，使得其中一个数变成 X，如果这个魔法至多能使用 1 次，求最大区间和。
    - X = 100，[1,2,3,4,5] → [100,2,3,4,5]
    - X = 100，[1,2,-100,4,5] → [1,2,100,4,5]
  - $n \le 10^6$
  - 定义
    - $dp_{i,0}$ 表示 i 为右端点时，该区间**没有使用魔法**，可以取到的最大和
    - $dp_{i,1}$ 表示 i 为右端点时，该区间**已使用过魔法**，可以取到的最大和

动态规划热身

- 例题4. 最大区间和 II：
  - 在 例题3 的基础上，可以额外使用一次魔法，使得其中一个数变成 X，如果这个魔法至多能使用 1 次，求最大区间和。
    - X = 100，[1,2,3,4,5] → [100,2,3,4,5]
    - X = 100，[1,2,-100,4,5] → [1,2,100,4,5]
  - $n \leq 10^6$

$$dp_{i,0} = \max\{dp_{i-1,0} + a_i, a_i\}$$

$$dp_{i,1} = \max\{dp_{i-1,1} + a_i, dp_{i-1,0} + X, X\}$$

  - 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！$dp_{i,1} = \max\{dp_{i-1,1} + a_i, dp_{i-1,0} + X, X\}$
  - 在前面的例题中，比如上面的方程，体现了**“抉择”**的过程；
  - 对上一个状态具有最优性的解，进行**“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(在数组中)，再次用到时直接使用先前的结果，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，再次遇到这些子问题时直接使用保存的答案即可。

  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**有向无环**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”

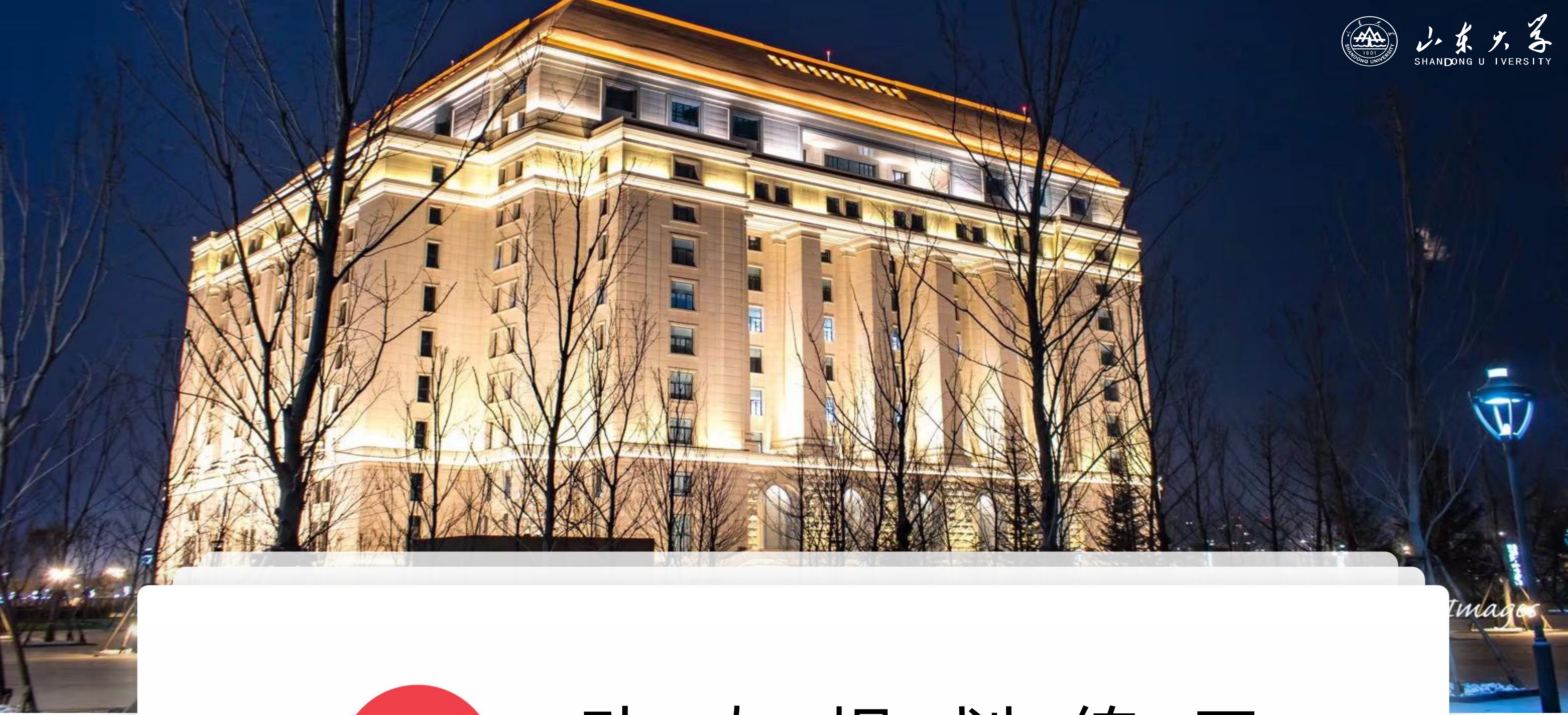
- 动态规划的一般格式为：

```
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
```

```
for(int i = 1; i <= n; i ++){
    dp[i][0] = max(dp[i - 1][0] + a[i], a[i]);
    dp[i][1] = max(dp[i - 1][1] + a[i], max(dp[i - 1][0] + x, x));
}
```

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

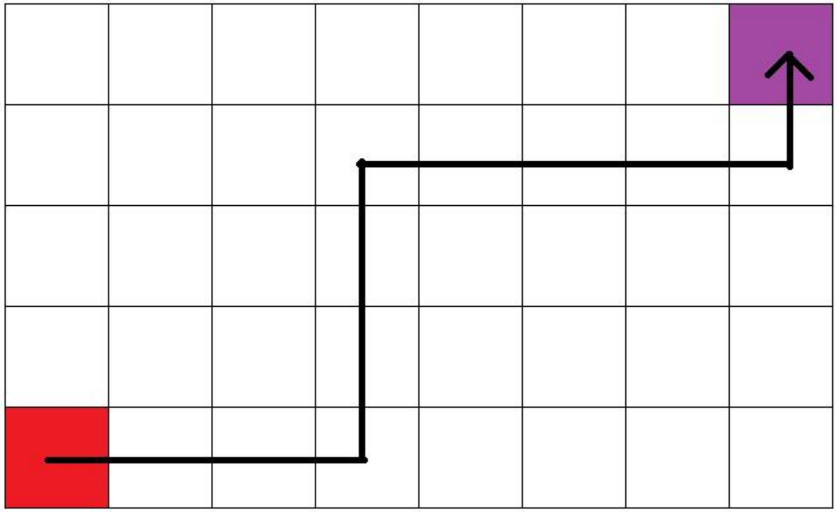


3 动 态 规 划 练 习

Dynamic Programming Exercise

动态规划热身

- 例题5. 走地图：
  - 在一个 n×m 的地图上，起点为左下角，终点为右上角，只能向右或向上行走，询问走到终点的方案数。
  - $n, m \leq 1000$
  - (可以通过排列组合解决)



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]
  - 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6. 拿数问题（作业）：**
  - 给 n 个数A[1..n]，可以选择若干数拿走，得分为这些数的和；
  - 所有拿走的数中，任意两数差值的绝对值不能为 1；
  - 求最大分数。
  - $n,\ A_i \le 10^5$

动态规划热身

- **例题6. 拿数问题（作业）：**
  - 给 n 个数A[1..n]，可以选择若干数拿走，得分为这些数的和；
  - 所有拿走的数中，任意两数差值的绝对值不能为 1；
  - 求最大分数。
  - $n,\ A_i \leq 10^5$

  - 题意的转换：如果拿走的一个数字为x，那么不能拿值为x-1和x+1的数字

  - 解题与 A 中数字的顺序无关，只与每个数字的出现次数有关
  - 从 A 序列转换到 CNT 序列，CNT[i] 代表值为 i 的数字出现了多少次

  - dp[i]， 仅考虑 大小为 1..i 的数，能拿到的最大分数

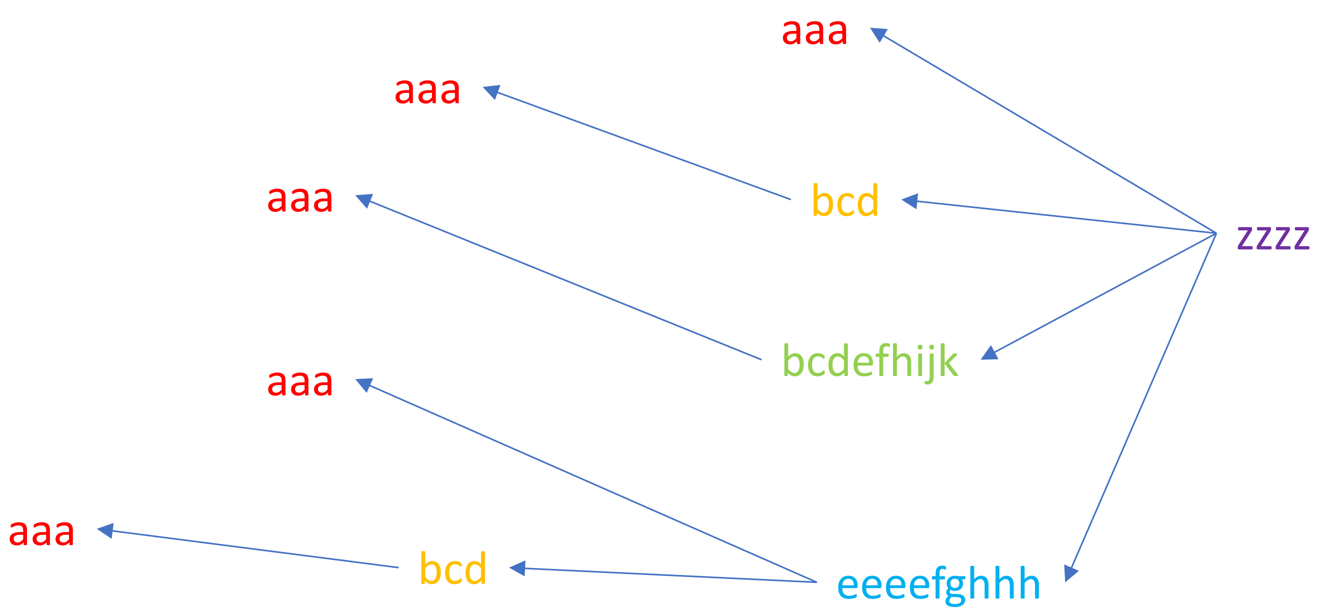
  - dp[i] = max(dp[i-1], dp[i-2]+CNT[i] * i)

动态规划热身

- 例题7.高昂的旋律：
  - 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.
    目前已经有了 n 段高昂的旋律，需要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？$(n \le 10^6$, 字符串长度之和 $\le 10^6)$
  - 5
    bcdefhijk
    bcd
    aaa
    eeeefghhh
    zzzz
  - 答案是 19（aaa, bcd, eeeefghhh, zzz）

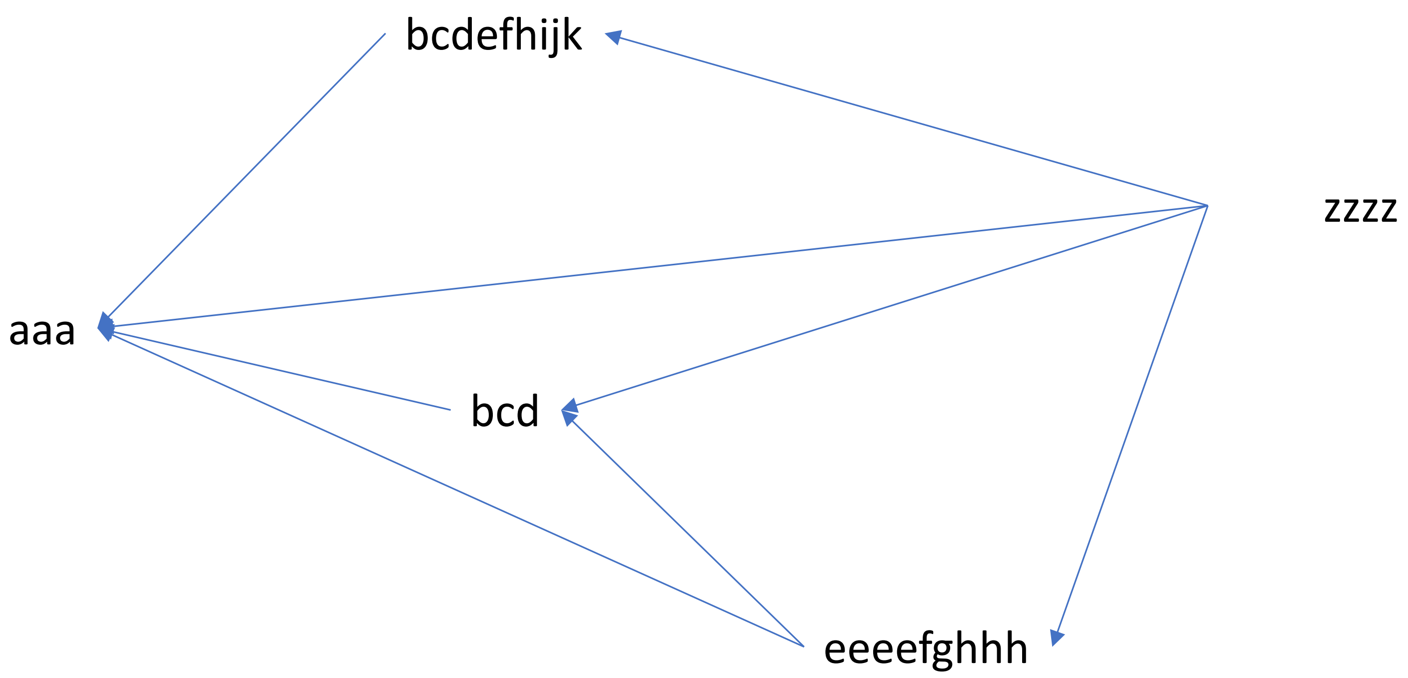
动态规划热身

- 例题7.高昂的旋律：
- 先考虑暴力搜索?
- 每个节点表示以该字符串为结尾的最长字符串



动态规划热身

- 例题7.高昂的旋律：



动态规划热身

- 例题7.高昂的旋律：
  - 为了不产生后效性，要保证一个单词 i 在处理之前，所有结尾字符 ASCII 小于 i 开头的，都已经完成了求解。（若首尾相同，可以按照任意顺序处理）
  - 于是按照每个字符串的最后一个字母排序，按照此顺序处理。
  - 定状态：定义 dp[i] 表示以第i个字符串结尾的最长字符串长度。
  - 状态转移方程为：
    - dp[i] = max{dp[i], dp[j]+str[i].size | 0<=j<i 且 str[j].back <= str[i].front}
  - 初始化：dp[i] = str[i].size
  - 输出答案：max{dp[i], 0<=i<n}
  - 如何优化?

动态规划热身

- **例题8. 矩阵选数（作业）**：
  - 给定一个矩阵 3*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，得到一个序列
  - 让序列“相邻两数差值的绝对值”求和尽可能小
  - $n \leq 10^6$
  - 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）**：
  - 当前列的决策与上一列的选择有关，且每一列只与前一列有关联。
  - 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[][]=INF, dp[1][0/1/2]=0
  - 输出答案：min{dp[n][i], 0<=i<=2}

动态规划热身

- 例题9. 装风力发电机：
  - 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为 $d_1, d_2, \ldots, d_n$ 以一维坐标的形式给出，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 $e_i > 0$。如何安排风力发电机的位置以产生最大化的能量?
  - $n \leq 10^6$

动态规划热身

- 例题9. 装风力发电机：
  - 设计状态：$dp[i]$ 表示考虑前 i 个风力发电可以得到的最大能量
  - 初始化：$dp[0] = 0$
  - 状态转移：$dp[i] = \max(dp[j] + e_i, dp[i-1])$
    - 其中 j 是满足 $|d[j] - d[i]| \geq D$ 的最大的 j。
  - 输出：$dp[n]$ 是最大化能量

  - 如何快速找出符合条件的 j?
    - 随着 i 的增长，j 的增长是是单调的
    - 维护指针，在 i 增长时，尝试增长 j 的值即可。

4 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 $A_1, A_2, \ldots, A_n$，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。
  - $n \leq 10^6$
  - 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 ans = 4

经典问题

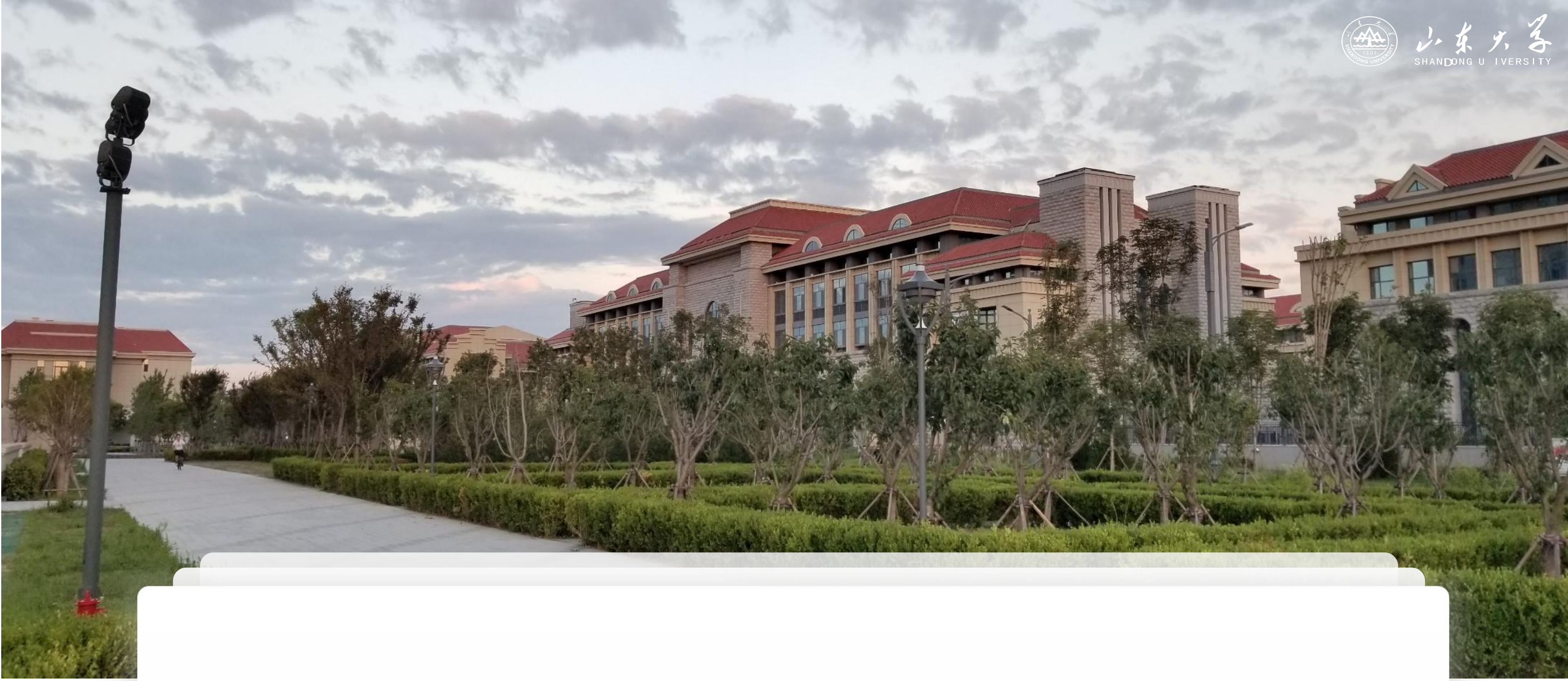
- 最长上升子序列
  - 状态：定义 f_i 表示以 A_i 为结尾的最长上升序列的长度。
  - 初始化：$f_1 = 1$
  - 转移过程：$f_i = \max\{f_j \mid j < i \wedge A_j < A_i\} + 1$
  - 输出答案：max{f[i], i=1...n}
  - 时间复杂度：$O(n^2)$

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(n²) 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 Ai 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(nlogn)。
  - 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

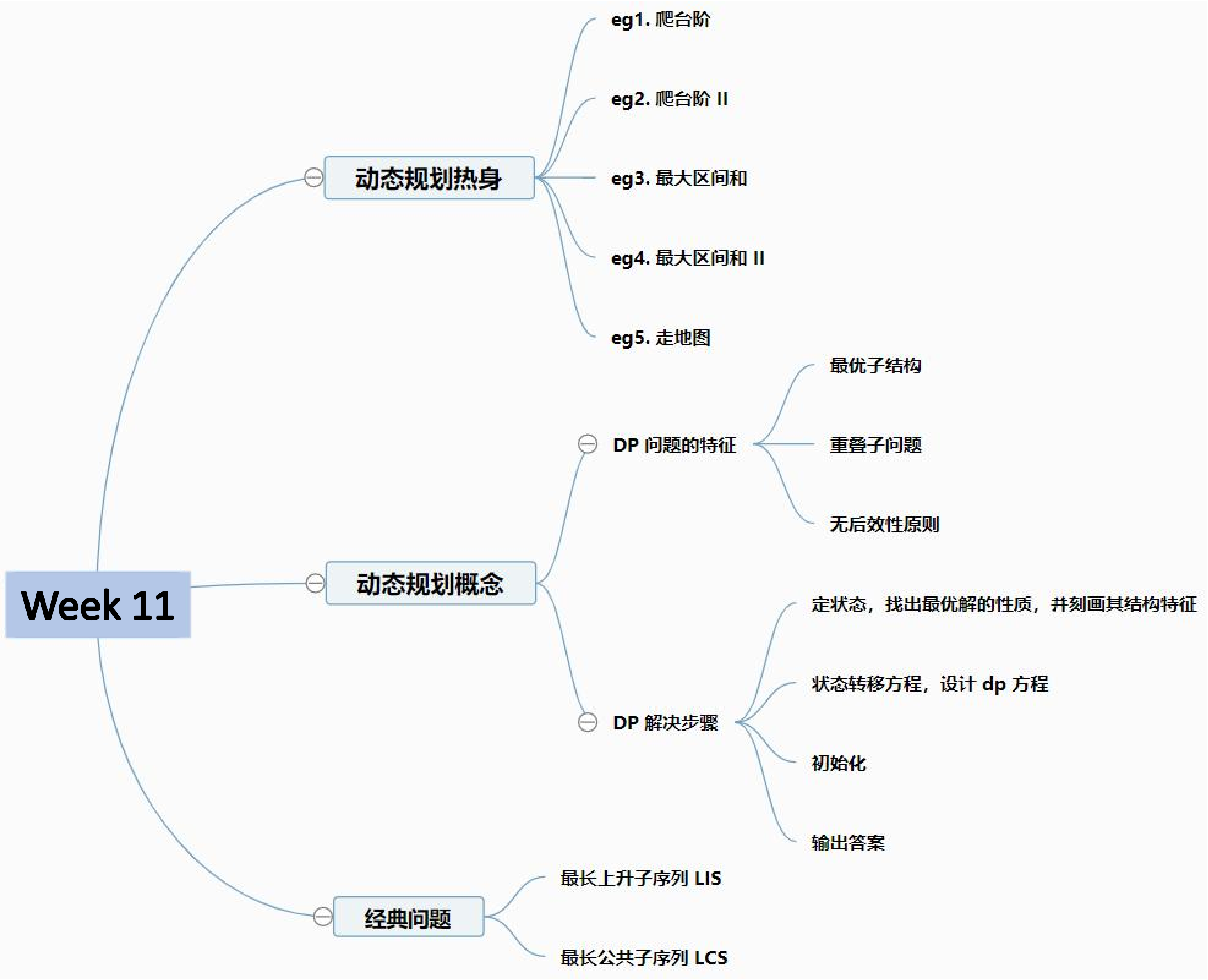
经典问题

- 最长公共子序列
  - 设计状态：假设 f[i][j] 为 $A_1, A_2, \ldots, A_i$ 和 $B_1, B_2, \ldots, B_j$ 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(mlogn)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


Week 11
动态规划热身
eg1. 爬台阶
eg2. 爬台阶 II
eg3. 最大区间和
eg4. 最大区间和 II
eg5. 走地图
动态规划概念
DP 问题的特征
最优子结构
重叠子问题
无后效性原则
DP 解决步骤
定状态，找出最优解的性质，并刻画其结构特征
状态转移方程，设计 dp 方程
初始化
输出答案
经典问题
最长上升子序列 LIS
最长公共子序列 LCS

感谢收听

Thank You For Your Listening