动态规划

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.12

目录

01

概念

TEACHING
COURSEWARE

TEACH

概念

动态规划的核心思想
动态规划是一种解决复杂问题的方法，它将问题分解为相对简单的子问题，通过解决子问题并保存子问题的解（避免重复计算），最终高效地解决原问题。其核心在于：

重叠子问题： 原问题可以分解成若干子问题，这些子问题在求解过程中会被重复计算多次。

最优子结构： 原问题的最优解包含其子问题的最优解。也就是说，我们可以通过组合子问题的最优解来构造原问题的最优解。

状态转移方程：描述如何从子问题的解推导出当前问题的解（是前两个性质的具体体现）。

02

解题关键

TEACHING
COURSEWARE

TEACH

解题关键

定义状态：
这是最重要也最困难的一步。
状态就是描述子问题的一组变量。你需要找到能唯一确定一个子问题并刻画其“进展程度”的变量。
通常用一个数组 dp[i] 或 dp[i][j] 来表示状态，其中 i，j 等是状态变量（也叫状态参数）。
问需要存储什么信息才能在后续步骤中利用已经解决的更小规模的问题
例子：

斐波那契数：dp[i] 表示第 i 个斐波那契数。

爬楼梯：dp[i] 表示爬到第 i 阶楼梯的方法数。

解题关键

确定状态转移方程：
它定义了如何用已经计算出来的、更小的子问题的解来推导出当前状态的解。
明确 dp[i][j]（或 dp[i]）的值是如何由 dp[i-1][j], dp[i][j-1], dp[i-1][j-1] 等已知状态的值计算出来的。
通常包含一个或多个决策（选择），并取最优（如 max 或 min）。
问自己： “当前状态（i, j）的结果，可以由哪些更小的、我已经知道答案的状态（x, y）的结果，通过什么操作（选择/决策）得到？”
例子：
斐波那契数：dp[i] = dp[i-1] + dp[i-2]。
爬楼梯：dp[i] = dp[i-1] + dp[i-2]（假设每次爬1或2阶）。

03

经典例题

TEACHING
COURSEWARE

TEACH

经典例题

题目描述

复制 Markdown 展开 进入 IDE 模式

给出一个长度为 n 的序列 a，选出其中连续且非空的一段使得这段和最大。

输入格式

第一行是一个整数，表示序列的长度 n。

第二行有 n 个整数，第 i 个整数表示序列的第 i 个数字 a_i。

输出格式

输出一行一个整数表示答案。

输入输出样例

输入 #1	复制	输出 #1	复制
7 2 -4 3 -1 2 -4 3		4	

说明/提示

样例 1 解释

选取 $[3,5]$ 子段 $\{3,-1,2\}$，其和为 4。

数据规模与约定

- 对于 40% 的数据，保证 $n \le 2 \times 10^3$。
- 对于 100% 的数据，保证 $1 \le n \le 2 \times 10^5$，$-10^4 \le a_i \le 10^4$。

经典例题

采用动态规划（DP）求解：

1. 定义状态 dp[i] 表示以第 i 个元素结尾的最大子段和

2. 状态转移方程：dp[i] = max(dp[i-1] + a[i], a[i])

选择 1：将第 i 个元素加入前一个子段（dp[i-1] + a[i]）

选择 2：从第 i 个元素重新开始一个子段（a[i]）

3. 最终答案是所有 dp[i] 中的最大值

经典例题


```cpp
#include<bits/stdc++.h>
using namespace std;
// 数组存储原始数据
int a[200005];
// dp数组存储以第i个元素结尾的最大子段和
int dp[200005];
int main()
{
    int n;
    cin >> n;  // 输入序列长度
    // 读入序列并初始化dp数组
    for(int i = 1 ; i <= n ; i++)
    {
        cin >> a[i];
        dp[i] = a[i];  // 初始化为自身（从当前元素开始的子段）
    }
    int ans = -1e9;  // 初始化答案为极小值，应对全负数情况
```

经典例题


```
    // 动态规划计算
    for(int i = 1 ; i <= n ; i++)
    {
        // 状态转移：选择继续前一段或重新开始
        dp[i] = max(dp[i-1] + a[i], a[i]);
        // 更新全局最大值
        ans = max(ans, dp[i]);
    }
    cout << ans << endl;  // 输出结果
    return 0;
}
```

经典例题

题目描述

复制 Markdown 展开 进入 IDE 模式

这是一个简单的动规板子题。

给出一个由 $n(n \leq 5000)$ 个不超过 10^6 的正整数组成的序列。请输出这个序列的**最长上升子序列**的长度。

最长上升子序列是指，从原序列中**按顺序**取出一些数字排在一起，这些数字是**逐渐增大**的。

输入格式

第一行，一个整数 n，表示序列长度。

第二行有 n 个整数，表示这个序列。

输出格式

一个整数表示答案。

输入输出样例

输入 #1	复制	输出 #1	复制
6 1 2 4 1 3 4		4	

说明/提示

分别取出 1、2、3、4 即可。

经典例题

状态定义：设 dp[i] 表示以序列中第 i 个元素（nums[i]）为结尾的最长上升子序列的长度。这里的 “结尾” 意味着该子序列必须包含 nums[i]，且 nums[i] 是子序列的最后一个元素。
状态转移：对于每个 i（从 2 到 n），需检查其前面所有元素 j（从 1 到 i-1）。若 nums[j] < nums[i]，说明 nums[i] 可接在以 nums[j] 结尾的子序列后，形成更长的子序列，此时 dp[i] 可更新为 dp[j] + 1（即 j 对应的最长子序列长度加 1）。为保证 dp[i] 始终记录最大值，需取当前 dp[i] 与 dp[j] + 1 中的较大值。最终，dp 数组的最大值即为整个序列的最长上升子序列长度。

经典例题


```cpp
#include<bits/stdc++.h>
using namespace std;
int nums[5005];
int dp[5005];
int main() {
    int n;
    cin >> n;
    // 输入序列元素（下标从1开始）
    for (int i = 1; i <= n; i++) {
        cin >> nums[i];
    }
    // 初始化dp数组：每个元素的初始LIS长度为1
    for (int i = 1; i <= n; i++) {
        dp[i] = 1;
    }
    // 记录全局最长的LIS长度（至少为1，因为n≥ 1）
    int ans = 1;
```

经典例题

```
    // 第一层循环：遍历每个元素i（作为子序列的结尾，从2开始，因为i=1前面没有元素）
    for (int i = 2; i <= n; i++) {
        // 第二层循环：检查i前面的所有元素j（j从1到i-1）
        for (int j = 1; j < i; j++) {
            // 若nums[j] < nums[i]，说明可以接在j后面形成更长的子序列
            if (nums[j] < nums[i]) {
                // 更新dp[i]：取当前dp[i]和dp[j]+1中的较大值
                if (dp[j] + 1 > dp[i]) {
                    dp[i] = dp[j] + 1;
                }
            }
        }
        // 每次计算完dp[i]后，更新全局最大长度
        if (dp[i] > ans) {
            ans = dp[i];
        }
    }
    // 输出结果
    cout << ans << endl;

    return 0;
}
```

04

总结

TEACHING
COURSEWARE

TEACH

总结

问需要存储什么信息才能在后续步骤中利用已经解决的更小规模的问题

例子：

斐波那契数：dp[i] 表示第 i 个斐波那契数。

爬楼梯：dp[i] 表示爬到第 i 阶楼梯的方法数。