极客时间算法训练营
第十三课
动态规划（三）

李煜东

《算法竞赛进阶指南》作者

目录

动态规划的优化

引入

给定 n 个二元组 $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$，已经按照 x 从小到大排好序了

求 $y_i+y_j+\left|x_i-x_j\right|$ 的最大值（$i\neq j$）

朴素 $O(n^2)$

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        if (i != j) ans = max(ans, y[i] + y[j] + abs(x[i] - x[j]));
```

优化：上面的计算中有哪些冗余?

第一步优化

给定 n 个二元组 $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$，已经按照 x 从小到大排好序了

求 $y_i+y_j+|x_i-x_j|$ 的最大值（$i\neq j$）

式子的值与 i,j 的顺序无关，不妨设 $j<i$

计算量减少了一半，可惜还是 $O(n^2)$

x_i,x_j 大小关系已知，绝对值也可以拆了

```
for (int i = 2; i <= n; i++)
    for (int j = 1; j < i; j++)
        ans = max(ans, y[i] + y[j] + x[i] - x[j]);
```

第二步优化

```cpp
for (int i = 2; i <= n; i++)
    for (int j = 1; j < i; j++)
        ans = max(ans, y[i] + y[j] + x[i] - x[j]);
```

$y[i] + x[i]$ 并不随着 j 而变化，可以提出来在外边算

减少了一些加法的次数，虽然没什么卵用，还是 $O(n^2)$

```cpp
for (int i = 2; i <= n; i++) {
    int temp = -1000000000;
    for (int j = 1; j < i; j++)
        temp = max(temp, y[j] - x[j]);
    ans = max(ans, y[i] + x[i] + temp);
}
```

第三步优化

```
for (int i = 2; i <= n; i++) {
    int temp = -1000000000;
    for (int j = 1; j < i; j++)
        temp = max(temp, y[j] - x[j]);
    ans = max(ans, y[i] + x[i] + temp);
}
```

重点来了，请问对于每个 i，你都计算了哪些 j 和哪些算式？

$i = 2$	$j = 1$	$\max\{y_1 - x_1\}$
$i = 3$	$j = 1,2$	$\max\{y_1 - x_1,\ y_2 - x_2\}$
$i = 4$	$j = 1,2,3$	$\max\{y_1 - x_1,\ y_2 - x_2,\ y_3 - x_3\}$
$i = 5$	$j = 1,2,3,4$	$\max\{y_1 - x_1,\ y_2 - x_2,\ y_3 - x_3,\ y_4 - x_4\}$

第三步优化

$i=2$	$j=1$	$\max\{y_1-x_1\}$
$i=3$	$j=1,2$	$\max\{y_1-x_1,\ y_2-x_2\}$
$i=4$	$j=1,2,3$	$\max\{y_1-x_1,\ y_2-x_2,\ y_3-x_3\}$
$i=5$	$j=1,2,3,4$	$\max\{y_1-x_1,\ y_2-x_2,\ y_3-x_3,\ y_4-x_4\}$

这里有大量的冗余!

$y_1-x_1,\ y_2-x_2,\ y_3-x_3$ 的最大值明明已经在 $i=4$ 的时候算过了，$i=5$ 为啥还要再算一遍呢?

记 $temp=\max\{y_1-x_1,\ y_2-x_2,\ y_3-x_3\}$

$i=5$ 时我们只需要计算 $\max(temp,\ y_4-x_4)$

对于每个 i，我们只需要让已有的 $temp$ 与最新的“候选项” $y_{i-1}-x_{i-1}$ 取max!

O(n) 代码

```
for (int i = 2; i <= n; i++) {
    temp = max(temp, y[i - 1] - x[i - 1]);
    ans = max(ans, y[i] + x[i] + temp);
}
```

你能想象这两份代码是等价的吗？

```
for (int i = 2; i <= n; i++)
    for (int j = 1; j < i; j++)
        ans = max(ans, y[i] + y[j] + x[i] - x[j]);
```

动态规划的转移优化思想

刚才我们完成上面的优化，主要依靠两点：

- 分离 i 和 j。与 i 有关的式子放一边，与 j 有关的式子放一边，不要互相干扰。
- 观察内层循环变量 j 的取值范围随着外层循环变量的变化情况

在动态规划中经常遇到类似的式子，i 是状态变量，j 是决策变量

- 分离状态变量和决策变量。当循环多于两重时，关注最里边的两重循环，把外层看作定值。
- 对于一个状态变量，决策变量的取值范围称为“决策候选集合”，观察这个集合随着状态变量的变化情况

一旦发现冗余，或者有更高效维护“候选集合”的数据结构，就可以省去一层循环扫描！

满足不等式的最大值

https://leetcode-cn.com/problems/max-value-of-equation/

给定 n 个二元组 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$，一个整数 k

求 $|x_i - x_j| < k$ 的前提下，$y_i + y_j + |x_i - x_j|$ 的最大值（$i \neq j$）

还是设 $j < i$，多了一个 $x_i - x_j < k$ 的条件

也就是 j 和 i 离得不能太远

当 i 增大时，j 的取值范围上下界同时增大，要维护 $y_j - x_j$ 的max

滑动窗口最大值！

环形子数组的最大和

https://leetcode-cn.com/problems/maximum-sum-circular-subarray/

给定一个由整数数组 A 表示的环形数组 C，求 C 的非空子数组的最大可能和。

请大家回顾之前学过的：

- 最大子序和
- “滚动型”环形动态规划的处理方法

这道题是一个“区间型”环形动态规划

环形子数组的最大和

最大子序和，是对每一个 $S[i]$，找它前面最小的 $S[j]$，其中 S 是前缀和，也就是：

设 $F[i] = S[i] - \min\limits_{1 \le j < i}\{S[j]\}$

目标 $\max\limits_{2 \le i \le n}\{F[i]\}$

如果把 i 看作状态，j 看作决策，可以发现 i 不仅只跟 $i-1$ 有关，而是跟 $j = 1 \sim i-1$ 都有关

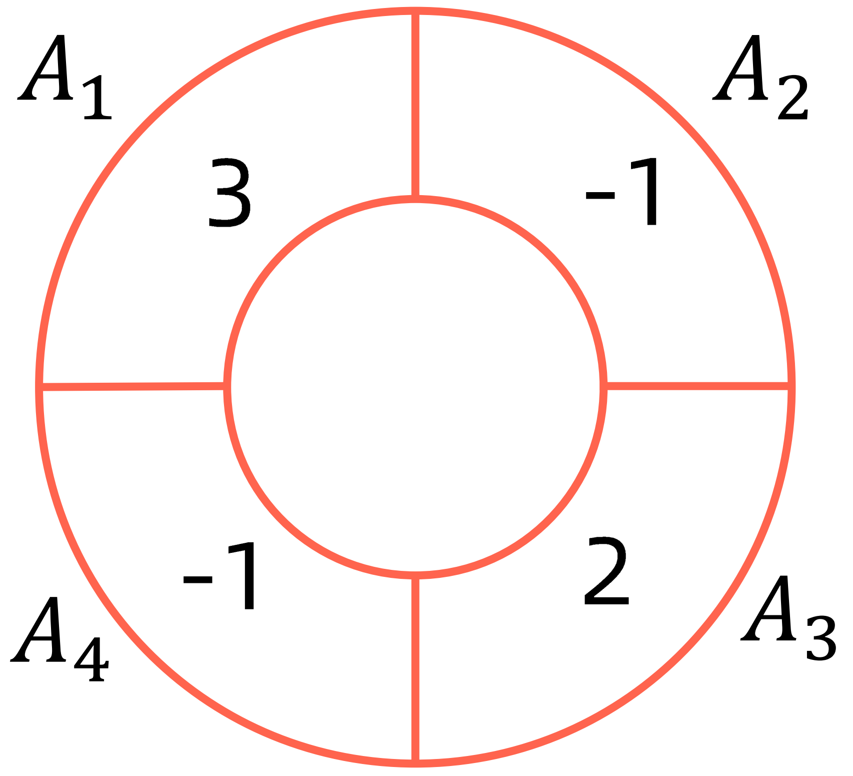
把数组看作线性（$1 \sim n$），然后复制一倍接在后面，变成长度为 $2n$ 的数组，求前缀和

设 $F[i] = S[i] - \min\limits_{i-n \le j < i}\{S[j]\}$

目标 $\max\limits_{2 \le i \le 2n}\{F[i]\}$

滑动窗口最小值！

环形子数组的最大和



AA	3	-1	2	-1	3	-1	2	-1
SS	3	2	4	3	6	5	7	6

$$2 + (-1) + 3 = 6 - 2 = 4$$

区间动态规划

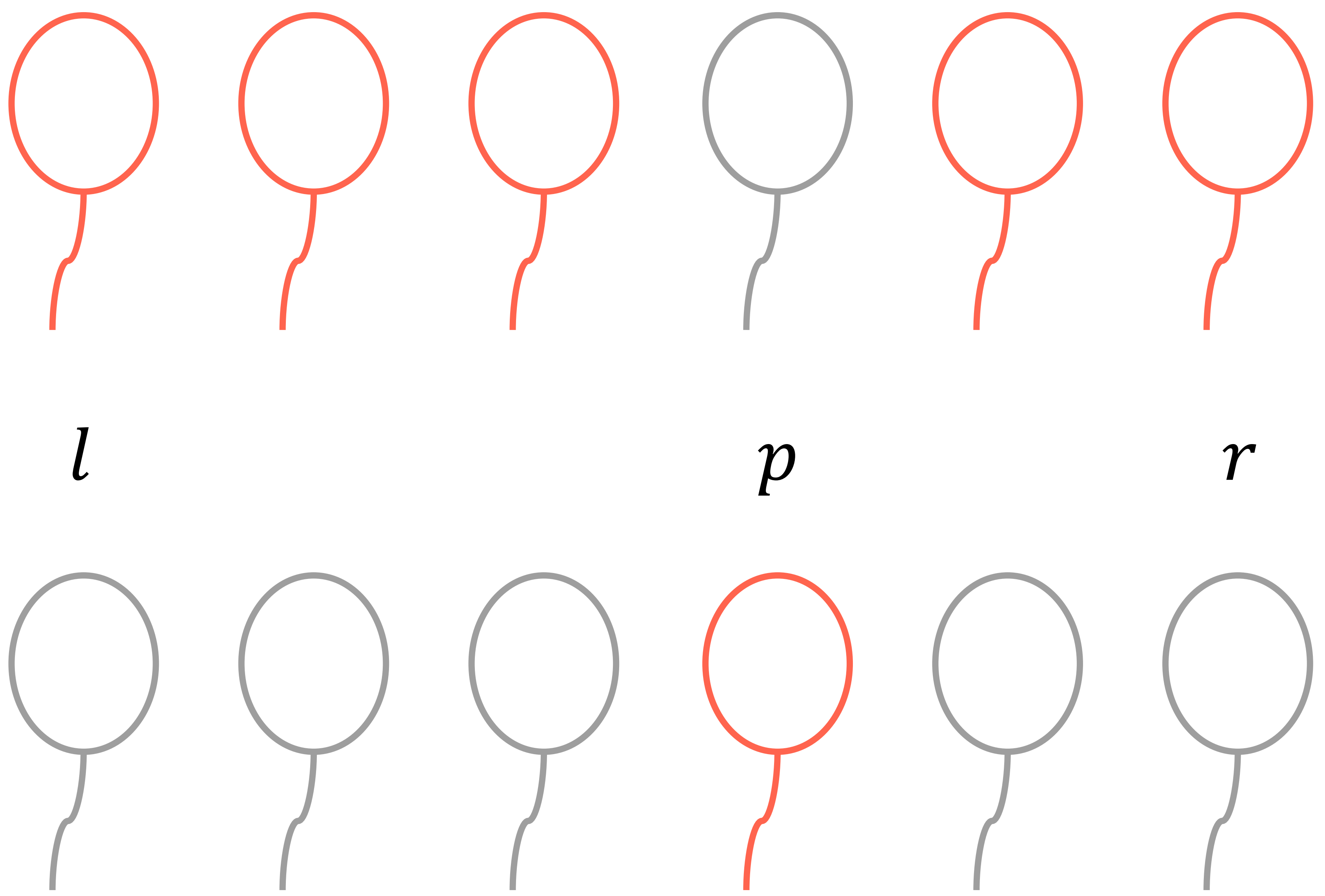
戳气球

https://leetcode-cn.com/problems/burst-balloons/

思路一：先戳哪个气球？

戳完 p 以后，子问题 $[l,p-1]$ 和 $[p+1,r]$ 两端
相邻的气球发生了变化！

它们和 $[l,r]$ 不再是同类子问题！



思路二：最后一个戳的是哪个气球？

先戳完 $[l,p-1]$ 和 $[p+1,r]$，最后戳 p

子问题两端相邻的气球不变，只有区间端点是变化信息

满足同类子问题！

戳气球

$f[l,r]$ 表示戳破闭区间 $l \sim r$ 之间的所有气球，所获硬币的最大数量

决策：最后一个戳的是 p

$$f[l,r] = \max_{l \le p \le r}\{f[l,p-1] + f[p+1,r] + nums[p] * nums[l-1] * nums[r+1]\}$$

初值：当 $l > r$ 时，$f[l,r] = 0$

目标：$f[1,n]$

区间动态规划的子问题是基于一个区间的

区间长度作为DP的“阶段”，区间端点作为DP的“状态”

在计算区间长度为 len 的子问题时，要先算好所有长度 $< len$ 的子问题

戳气球（C++ Code）

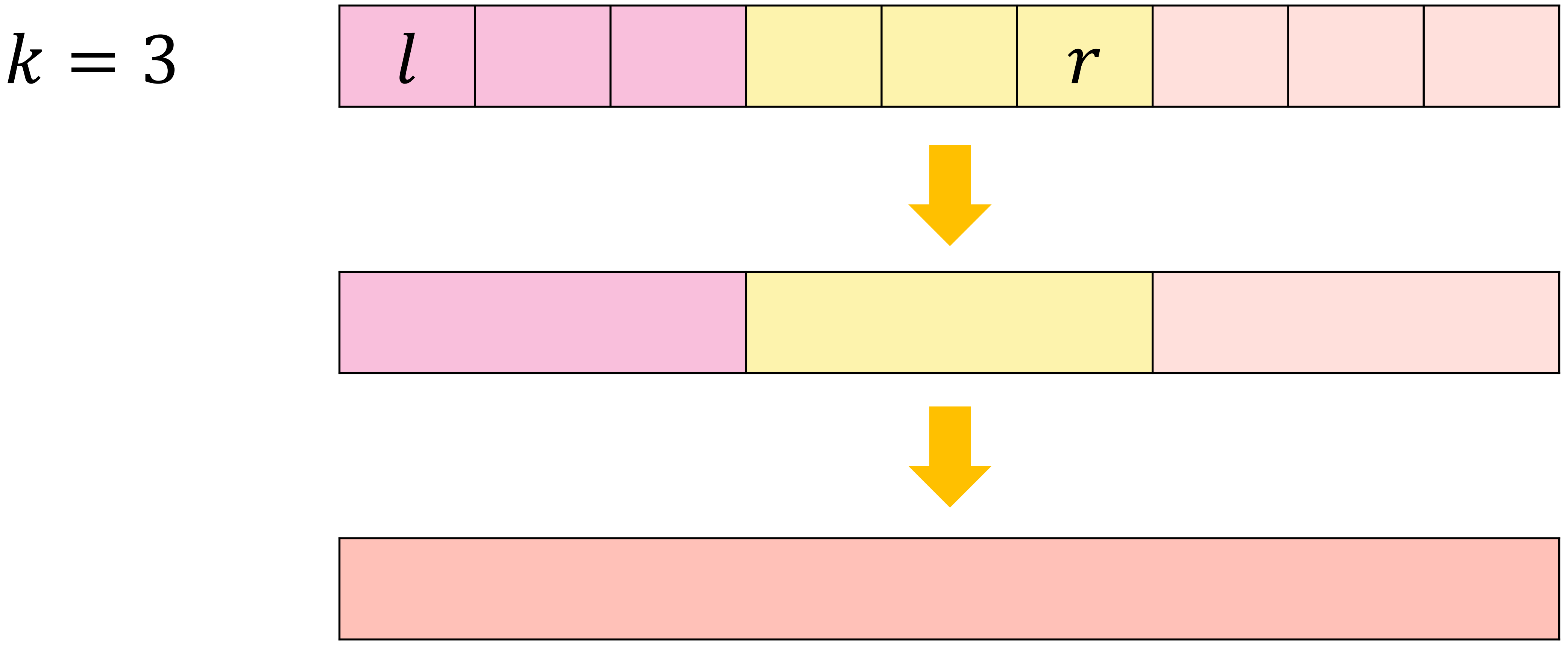
```cpp
int maxCoins(vector<int>& nums) {
    int n = nums.size();
    nums.insert(nums.begin(), 1);
    nums.push_back(1);
    vector<vector<int>> f(n + 2, vector<int>(n + 2, 0));
    for (int len = 1; len <= n; len++)  // 区间动态规划，最外层循环区间长度
        for (int l = 1; l <= n - len + 1; l++) {  // 然后循环左端点
            int r = l + len - 1;  // 计算出右端点
            for (int p = l; p <= r; p++)
                f[l][r] = max(f[l][r], f[l][p - 1] + f[p + 1][r] +
                                       nums[p] * nums[l - 1] * nums[r + 1]);
        }
    return f[1][n];
}
```

合并石头的最低成本（选做）

https://leetcode-cn.com/problems/minimum-cost-to-merge-stones/

$f[l,r]$ 表示 $l\sim r$ 合成一堆的最低成本？

不行！$l\sim r$ 不一定要合成一堆，可能会合成若干堆，然后跟其他部分一起凑齐 k 堆，再合成一堆



如何表示 “$l\sim r$ 合成若干堆”这个子问题？信息不够，往状态里加！

合并石头的最低成本（选做）

$f[l, r, i]$ 表示把 $l \sim r$ 合并成 i 堆的最低成本

决策一：恰好凑成 k 堆，合成一堆

$f[l, r, 1] = f[l, r, k] + \sum_{p=l}^{r} nums[p]$

决策二：分成两个子问题，$l \sim p$ 合成 j 堆，$p + 1 \sim r$ 合成 $i - j$ 堆，一共 i 堆

$f[l, r, i] = \min_{l \le p < r, 1 \le j < i} \{f[l, p, j] + f[p + 1, r, i - j]\}$，其中 $i > 1$

时间复杂度 $O(n^3 k^2)$

决策二可以优化：不需要枚举 j，考虑第一堆是哪一段就行了

$f[l, r, i] = \min_{l \le p < r} \{f[l, p, 1] + f[p + 1, r, i - 1]\}$，其中 $i > 1$

时间复杂度 $O(n^3 k)$

树形动态规划

树形动态规划

树形动态规划与线性动态规划没有本质区别

其实只是套在深度优先遍历里的动态规划（在DFS的过程中实现DP）

子问题就是“一棵子树”，状态一般表示为“以 x 为根的子树”，决策就是“x 的子结点”

复杂的题目可以在此基础上增加更多与题目相关的状态、决策

打家劫舍 III

https://leetcode-cn.com/problems/house-robber-iii/

$f[x,0]$ 表示以 x 为根的子树，在不打劫 x 的情况下，能够盗取的最高金额

$f[x,1]$ 表示以 x 为根的子树，在打劫 x 的情况下，能够盗取的最高金额

$$f[x,0] = \sum_{y\ is\ a\ son\ of\ x} \max(f[y,0], f[y,1])$$

$$f[x,1] = val(x) + \sum_{y\ is\ a\ son\ of\ x} f[y,0]$$

目标：$\max(f[root,0], f[root,1])$

THANKS

极客时间 | 训练营