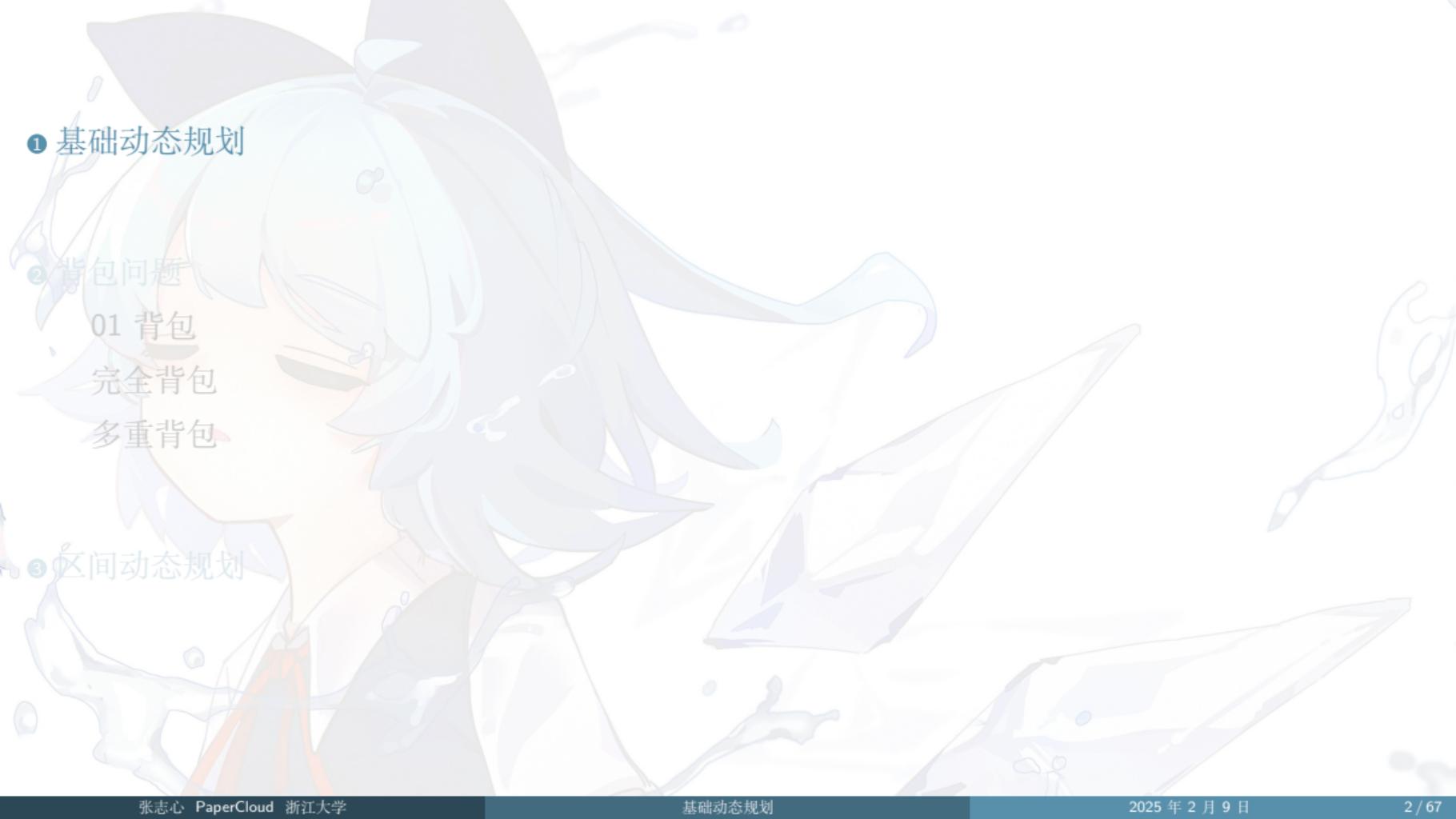


基础动态规划

张志心 PaperCloud
浙江大学

2025 年 2 月 9 日



① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

方格取数

从一个实际问题说起

有一个 $n \times m$ 的矩阵，初始位于 $(1, 1)$ ，只能向右或向下走，求经过的格子之和的最大值。

$1 \leq n, m \leq 2000$ 。

方格取数

贪心法的反例

直观做法是贪心：每次选取可走的格子中数字较大的格子。但可以举出很显然的反例

1	1	10	100
2	5	20	20
3	300	20	20

按贪心策略，我们会走 $1 \rightarrow 10 \rightarrow 100 \rightarrow 20 \rightarrow 20$ ，而实际上最优策略是 $1 \rightarrow 5 \rightarrow 300 \rightarrow 20 \rightarrow 20$ 。

方格取数

dfs

另一个容易想到的思路是深度优先搜索算法。即枚举所有路径并找出和最大的。因为路径前面经过了哪些点并不影响后面可以选择的点，所以我们只将路径最后经过的点加入状态。

```
1 int dfs(int x, int y) {
2     if (x == n && y == m) return a[x][y];
3     int res = 0;
4     if (x < n) res = max(res, dfs(x+1, y));
5     if (y < m) res = max(res, dfs(x, y+1));
6     return res + a[x][y];
7 }
```

方格取数

记忆化搜索

我们发现，上面的搜索算法中有很多量是重复计算的，而且还会计算很多本该已经被排除的量。

例如调用 `dfs(2,3)` 时，会递归调用 `dfs(3,3)`；而调用 `dfs(3,2)` 时同样会递归调用 `dfs(3,3)`。

换句话说，我们在同一棵搜索树中重复做了相同的事情。

那我们不妨将已经计算出的结果用数组记下来。

方格取数

记忆化搜索

设 $f[x][y]$ 为 $\text{dfs}(x, y)$ 返回的结果。它的实际意义就是从 (x, y) 出发，走到 (n, m) 能经过的数字之和的最大值。

```
1  bool vis[N][N];
2  int f[N][N];
3  int dfs(int x, int y) {
4      if (x == n && y == m) return a[x][y];
5      if (vis[x][y]) return f[x][y];
6      int res = 0;
7      if (x < n) res = max(res, dfs(x, y+1));
8      if (y < m) res = max(res, dfs(x+1, y));
9      vis[x][y] = 1;
10     return f[x][y] = res + a[x][y];
11 }
```

上面这个看似简单的搜索优化，其实就是**动态规划**。

我们分析一下它的复杂度。对每个点对 (x, y) ，只会有一次递归会进入后面的分支。故只会访问 $O(1)$ 次。而点对的数量也只有 $O(nm)$ ，因此时间复杂度就是 $O(nm)$ 。

也就是说，我们用 $O(nm)$ 的空间为代价，将时间复杂度从 $O(\binom{n+m}{n})$ 降低为 $O(nm)$ 。

动态规划

动态规划的三要素

动态规划本质上是搜索的优化。那么它的要素自然也和搜索是相通的：

- 状态：搜索树上节点的状态；
- 转移：搜索树上一个节点的子节点（事实上加入记忆化后，搜索树的很多状态合并了，它也不再是树了，而是 DAG）对自身的贡献；
- 边界：递归终止的条件。动态规划的空间复杂度一般是状态的数量，时间复杂度一般是状态数乘转移数。

动态规划

动态规划的状态转移方程和递推写法

动态规划的转移可写成递推的形式。例如上例中

- $f(x, y) = \max\{f(x + 1, y), f(x, y + 1)\} + a_{x,y}$
- $f(n, m) = a_{n,m}, f(n + 1, i) = f(i, m + 1) = 0, \forall i$

如果我们能计算出一个合理的顺序，使得一个状态所需的所有子节点状态会在它之前被计算（这本质上是对状态节点和转移边形成的 DAG 进行拓扑排序），那么我们就可以把递归形式的记忆化搜索改写成递推。

例如上例不难发现 $(n, m), (n, m - 1), \dots, (n, 1), (n - 1, m), \dots, (1, 1)$ 是一种合理的顺序。

```
1 for (int i = n; i; --i)
2     for (int j = m; j; --j)
3         f[i][j] = max(f[i+1][j], f[i][j+1]) + a[i][j];
```

动态规划

动态规划问题的思维方式

对一个动态规划的问题，可以按如下顺序思考：

- 首先可以设计一个搜索算法，然后尝试优化状态的复杂度（一般是看转移可以由哪些信息唯一确定）。
- 将状态的结果记忆化。需要确保状态不存在循环转移的情况（否则重新设计状态）。
- 直接写记忆化搜索，或者写出状态转移方程后转为递推。

在熟练掌握动态规划算法后，可以跳过第一步直接设计状态和转移方程。

最长上升子序列 (LIS)

给定一个序列 $\{a_1, \dots, a_n\}$, 求最长的子序列 $\{a_{i_1}, a_{i_2}, \dots, a_{i_k}\}$ 满足 $a_{i_1} \leq a_{i_2} \leq \dots \leq a_{i_k}$ 。

$1 \leq n \leq 5000$ 。

最长上升子序列 (LIS)

题解

转移能否进行只和选择的序列的末尾元素下标有关。

设 $f(i)$ 为以 a_i 为末尾的最长上升子序列，则

$$f(i) = \max_{j < i, a_j \leq a_i} f_j + 1, f(0) = 0.$$

转移顺序 $1, 2, \dots, n$, 时间复杂度 $O(n^2)$, 空间复杂度 $O(n)$ 。

最长公共子序列 (LCS)

给定两个序列 a, b , 求最长的子序列 $\{i_1 < i_2 < \dots < i_k\}$ 和 $\{j_1 < j_2 < \dots < j_k\}$ 满足 $a_{i_t} = b_{j_t}, \forall t$ 。
 $1 \leq n, m \leq 5000$ 。

最长公共子序列 (LCS)

题解

转移能否进行只和选择的两个序列的末尾元素下标有关。

设 $f(i, j)$ 为以 a_i 和 b_j 为末尾的最长公共子序列。则

$$f(i, j) = \max_{k < i, l < j} f_{k, l} + 1, a_i = a_j$$

其他 $f(i, j) = 0$ 。

但这样状态和转移都达到了 $O(nm)$, 复杂度太高。

转移是一个前缀 \max 的形式, 考虑优化: $F(i, j) = \max_{k \leq i, l \leq j} f(k, l)$, 则

$$\begin{aligned} F(i, j) &= F(i - 1, j - 1) + 1, & a_i = a_j \\ F(i, j) &= \max(F(i - 1, j), F(i, j - 1)), & a_i \neq a_j \end{aligned}$$

这样转移复杂度降至 $O(1)$, 总时空复杂度均为 $O(nm)$ 。

机器分配问题

给定 $n \times m$ 的矩阵 A 和一个整数 s 。将 s 划分成 n 个不超过 m 的整数 p_1, \dots, p_m ，使得 $\sum_{i=1}^n A(i, p_i)$ 最大。

$$1 \leq n, m \leq 200, 1 \leq s \leq 2000.$$

机器分配问题

题解

转移能否进行只和当前划分到第几个数和当前划分的总和有关。

设 $f(i, j)$ 为划分到第 i 个数，划分总和为 j ，最大的 $\sum_{k \leq i} A(k, p_k)$ ，则

$$f(i, j) = \max_{0 \leq k \leq \max(j, m)} (f(i - 1, j - k) + A(i, k)), f(0, 0) = 0.$$

状态复杂度 $O(ns)$ ，转移复杂度 $O(m)$ 。总复杂度 $O(nms)$ 。

最大 k 子段和问题

给一个序列，求 k 段不交可空连续子序列使其和最大。

$$1 \leq n \leq 10^5, 1 \leq k \leq 10$$

最大 k 子段和问题

题解

设 $f(i, j) =$ 前 i 个数选 j 段能达到的最大值，考虑第 i 个数的选择情况。

如果 i 选且 $i - 1$ 选则新开一段，否则延续上一段。因此还要再开一维 0/1 表示最后一个数是否选择。

故有转移

$$f(i, j, 0) = \max\{f(i - 1, j, 0), f(i - 1, j, 1)\}, f(i, j, 1) = \max\{f(i - 1, j - 1, 0), f(i - 1, j, 1)\} + a_i$$

目标状态 $\max\{f(n, j, 0), f(n, j, 1)\}, 1 \leq j \leq k$ 。

状态 $O(kn)$ ，转移 $O(1)$ 。

动态规划的常见优化方法

从状态角度的优化

状态的设计是 DP 程序设计的基础，它直接影响程序的空间复杂度，同时也是时间复杂度的重要影响因素。

从状态的角度出发，DP 的优化思路包括：

- 消除冗余状态（避免不必要的计算）
- 合并同类状态（降低状态数）
- 滚动记录状态（降低空间复杂度）

动态规划的常见优化方法

从转移角度的优化

状态转移是动态规划的核心，转移方程的设计直接决定了程序的正确性，同时也是时间复杂度的本质来源。

从转移的角度出发，DP 的优化思路包括：

- 排除无效转移（在最优化等问题中，基于已有信息，提前排除一些不可能成为最优解的转移）
- 数据结构加速转移（区间取 `max`、`sum` 等计算都可以用合适的数据结构优化），最常见的就是前缀 `max`、前缀 `sum` 等。

需要注意的是，由于 DP 和递推是互通的（同样有状态、转移和边界），所以上面这些优化方法也可以用于递推。

传纸条

有一个 $n \times m$ 的矩阵，初始位于 $(1, 1)$ ，只能向右或向下走，走到 (n, m) 。取两条路径，重复经过的点权不重复计算，求最大值。

$$1 \leq n, m \leq 500.$$

传纸条

题解

设 $f(i, j, k, l)$ 为两条路径分别以 (i, j) 和 (k, l) 为结尾时点权之和的最大值。则

$$f(i, j, k, l) = \max(f(i-1, j, k-1, l), f(i-1, j, k, l-1), f(i, j-1, k-1, l), f(i, j-1, k, l-1)) + a_{i,j} + a_{k,l}$$

状态复杂度 $O(n^2 m^2)$, 无法通过。

传纸条

题解

重新设计状态: $f_{i,j,k}$ = 第一个人走到 (i, j) , 第二个人走到 $(k, i+j-k)$ 时两人经过格子的最大的和, 则

$$f(i, j, k) = \max\{f(i-1, j, k-1), f(i, j-1, k-1), f(i-1, j, k), f(i, j-1, k)\} + a_{i,j} + a_{k, i+j-k} [(i, j) \neq (k, i+j-k)]$$

目标状态是 $f(m, n, m)$, 边界状态是 $f(1, 1, 1) = a_{1,1}$ 。

边界条件是 $1 \leq i, k \leq m, 1 \leq j, i+j-k \leq n$ 。

状态复杂度降到了 $O(n^2m)$, 转移还是 $O(1)$ 。可以通过本题。

子串

给两个字符串 S, T , 在 S 中选取 k 个不相交非空子串使得这些子串连接而成的串和 T 相等, 求选取方案数。

$$1 \leq n \leq 1000, 1 \leq k \leq m \leq 200.$$

子串

题解

我们设 $f(i, j, k) = S$ 的前 i 个字符，已匹配 T 的前 j 个字符，强制 S_i 与 T_j 匹配，共选了 k 段的方案数，考虑转移。

分情况讨论：如果 $S_i \neq T_j$ ，则 $f(i, j, k) = 0$ ；

否则，考虑 S_{i-1} 的选择情况。如果选了，则可以新开一段也可以不开；如果没选，则必须新开一段。即

$$f(i, j, k) = \sum_{l=1}^{i-1} f(l, j-1, k-1) + f(i-1, j-1, k)$$

状态 $O(nm^2)$ ，转移 $O(n)$ ，总时间复杂度 $O(n^2m^2)$ ，空间复杂度 $O(nm^2)$

我们发现 $f_{i,j,k}$ 的转移形式就是前缀和。因此令 $g(i, j, k) = \sum_{l=1}^i f(l, j, k)$ ，则

$$f(i, j, k) = [S_i = T_j](g(i - 1, j - 1, k - 1) + f(i - 1, j - 1, k))$$

时间复杂度降至 $O(nm^2)$ ，可以通过本题。

另外，空间方面可以使用**滚动数组**优化。注意滚动数组只能滚掉第一维，而第一维又要前缀和，所以 g 和 f 要同步更新。

这样空间复杂度降至 $O(m^2)$ 。

着色方案

有 n 个格子， k 种颜色，第 i 种颜色可以涂 c_i 个格子， $\sum_{i=1}^k c_i = n$ ，相邻两个格子不同色，求着色方案数。

$$1 \leq k \leq 15, 1 \leq c_i \leq 5$$

着色方案

算法一

如果我们直接将“每种颜色剩余数量”和“上一个格子的颜色”作为状态来 DP，需要开 $c + 1$ 维数组：

$$f(x_1, x_2, \dots, x_k, i) = \sum_{j \neq i} [x_j > 0] f(x_1, \dots, x_{i-1}, x_i - 1, x_{i+1}, \dots, x_k, j), 0 \leq x_j \leq c_j, x_i \geq 1$$

状态数达到了 $O(c^k k)$ ，无法通过而且实现非常繁琐（这里可以用将来要学习的“状态压缩”来简化实现）。

注意到 $1 \leq c_i \leq 5$ 的条件，于是将颜色按剩余能涂的格子数合并，如下设计状态：

状态定义 设 $f(c_1, c_2, c_3, c_4, c_5, \text{last})$ 表示，在：

- 还能涂 i 块的颜色有 c_i 种 ($1 \leq i \leq 5$)；
- 已涂部分的最后一块涂的是一种**现在还能涂 last 块的颜色**

的情况下，**涂完剩下的格子**的方案数。状态数为 $O(k^c c)$ ，可以承受。

状态转移 计算 $f(c_1, c_2, c_3, c_4, c_5, \text{last})$ 时，从 1 至 5 枚举 i ，表示下一个格子涂的是某种当前还能涂 i 块的颜色，则在涂色后，这种颜色之后还能涂 $i - 1$ 块，能 i 块的颜色减少一种，能涂 $i - 1$ 块的颜色增加一种。后续格子的涂色方案是一个子问题，可以递归求解。

着色方案

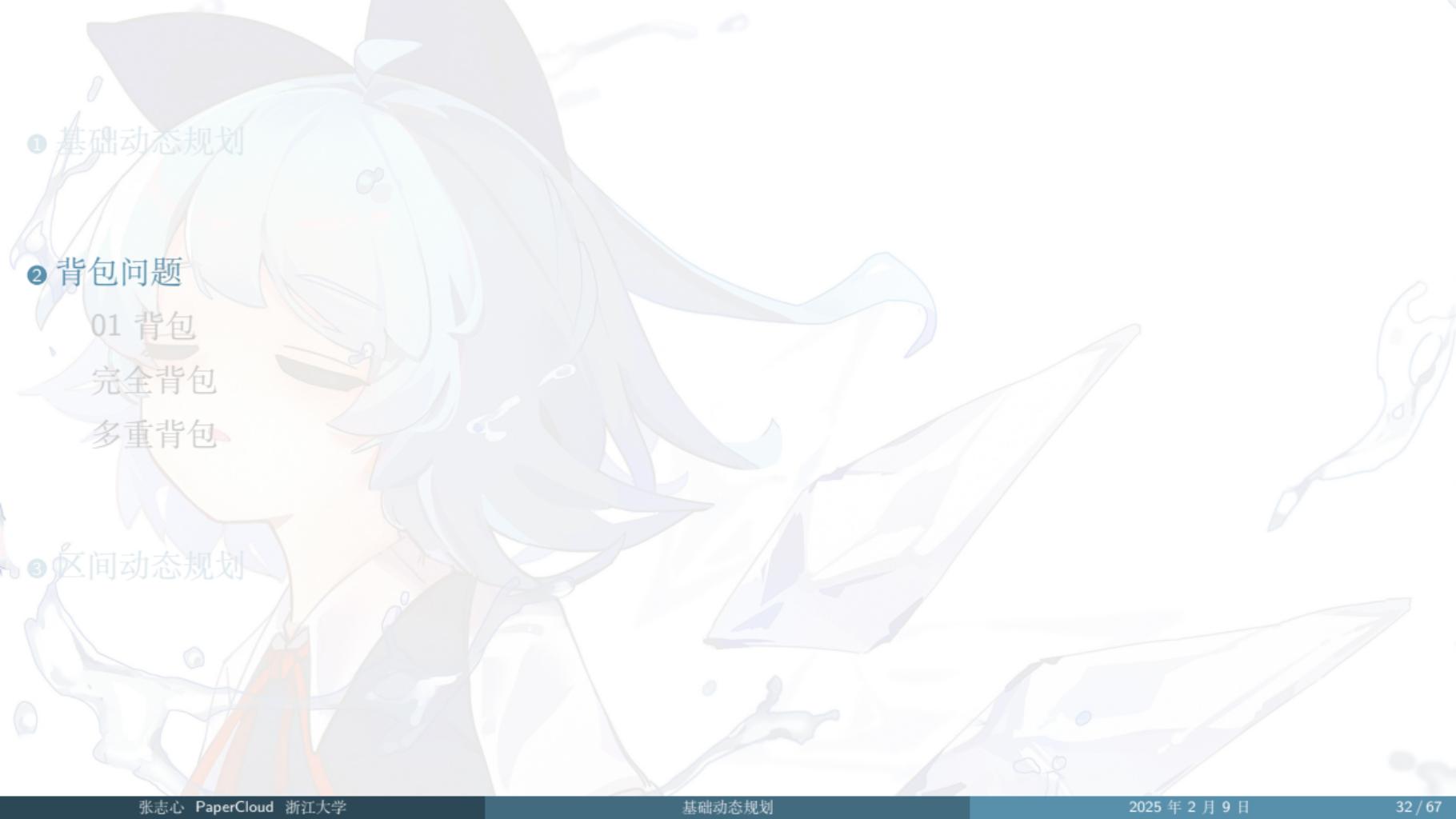
算法二

接下来还要考虑这个格子具体涂了哪种颜色，将可选颜色种数与子问题的方案数相乘累加到的答案上。由于相邻颜色不能相邻的限制：

- 若 $i = \text{last}$ ，则在全部 c_i 种还能涂 i 块的颜色种，需要之前已涂部分最后一块的那种颜色，还有 $c_i - 1$ 种选择；
- 否则，可在 c_i 种颜色中任选。

于是列出状态转移方程：

$$\begin{aligned} f(c_1, c_2, c_3, c_4, c_5, \text{last}) = & [c_1 > 0] f(c_1 - 1, c_2, c_3, c_4, c_5, 0) \times (c_1 - [\text{last} = 1]) \\ & + [c_2 > 0] f(c_1 + 1, c_2 - 1, c_3, c_4, c_5, 1) \times (c_2 - [\text{last} = 2]) \\ & + [c_3 > 0] f(c_1, c_2 + 1, c_3 - 1, c_4, c_5, 2) \times (c_3 - [\text{last} = 3]) \\ & + [c_4 > 0] f(c_1, c_2, c_3 + 1, c_4 - 1, c_5, 3) \times (c_4 - [\text{last} = 4]) \\ & + [c_5 > 0] f(c_1, c_2, c_3, c_4 + 1, c_5 - 1, 4) \times c_5 \end{aligned}$$



① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

01 背包

问题描述

有一个承重为 m 的背包和 n 件物品，每个物品有价值 v_i 和重量 w_i ，求背包能装下的物品的最大价值和。

$$1 \leq n \leq 100, 1 \leq m \leq 10^5.$$

01 背包

dp 方程

设 $f(i, j)$ 为将前 i 个物品放入承重为 j 的背包的最大价值，则

$$f(i, j) = \max(f(i - 1, j), f(i - 1, j - w_i) + v_i), f(0, 0) = 0.$$

01 背包

滚动空间优化

如果我们按 i 正序, j 倒序的顺序更新, 我们就可以直接用 $f(i, j)$ 的值覆盖 $f(i - 1, j)$ 的值而不影响转移时的 dp 值。这样我们就可以将 i 这一维直接滚动掉。

```
1 memset(f, 0, sizeof(f));
2 for (int i = 1; i <= n; ++i)
3     for (int j = m; j >= w[i]; --j)
4         f[j] = max(f[j], f[j-w[i]] + v[i]);
```

背包的可行性和方案数

可行性 有一个承重为 m 的背包和 n 件物品，每个物品有重量 w_i ，求背包能否恰好装满：

```
f[j] |= f[j-w[i]];
```

方案数 有一个承重为 m 的背包和 n 件物品，每个物品有重量 w_i ，求背包恰好装满的方案数：

```
f[j] += f[j-w[i]];
```

这两个问题的转移方式与经典的背包问题是一样的。区别在于把“取 \max 运算”分别改为“逻辑或”和“求和”。

超大 01 背包

有一个承重为 m 的背包和 n 件物品，每个物品有价值 v_i 和重量 w_i ，求背包恰好装满时，所装物品的最大价值和。

题面同原题，但数据范围改为 $1 \leq n \leq 36, 1 \leq m \leq 10^{18}$

因为 m 非常大，所以状态复杂度与 m 有关的 dp 时间上和空间上都是无法承受的。

n 较小，可以想到暴力搜索所有装包情况。但这样做的时间复杂度是 $O(2^n)$ ，也无法承受。

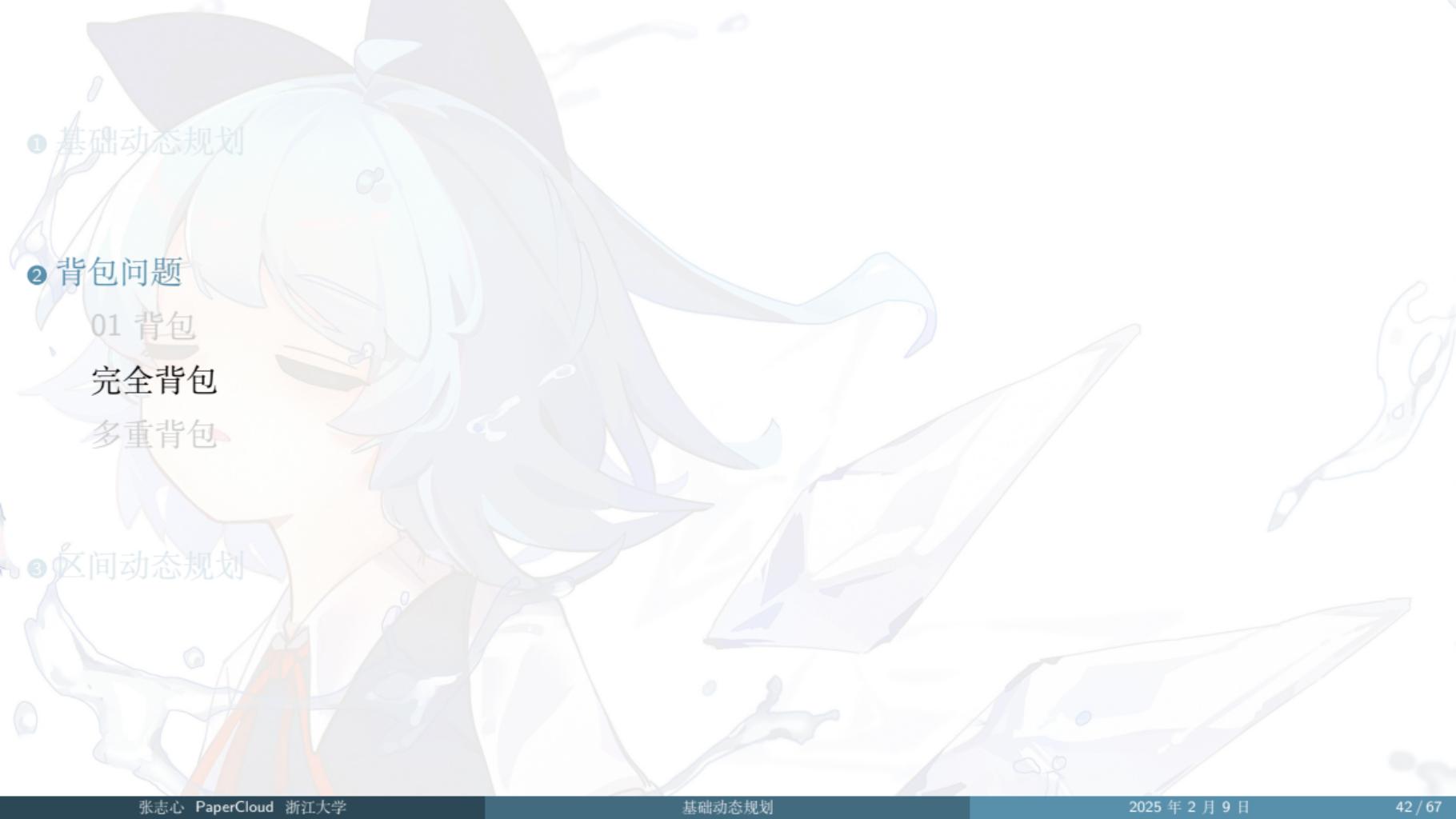
注意到 $2^{\frac{n}{2}} \approx 3 \times 10^5$ 是可以承受的运算量，考虑将物品等分成两部分。

对前 $\frac{n}{2}$ 个物品，我们暴力搜索其 $2^{\frac{n}{2}}$ 种装包方案，得到对应的重量和价值并用 `map` 存下来。

然后再考虑后 $2^{\frac{n}{2}}$ 种装包方案，假设某种方案的重量为 W ，价值为 V ，在 `map` 中查询重量为 $m - W$ 的方案对应的最大价值，用 `mp[m-W] + V` 更新答案。时间复杂度 $O(2^{\frac{n}{2}} n)$ ，空间复杂度 $O(n)$ 。

有负重量的背包

在某些实际问题中，可能会有物品的“重量”是负数。此时的转移方向也应做相应调整：从 m 倒序转移改为从 w_i 正序转移。另外，由于数组无法处理负下标，所以应将 dp 数组整体平移足够大的一个数 $d \geq \sum |w_i|$ ，用 $f[i+d]$ 来存储 $f(i)$ 。



① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

完全背包

问题描述

有一个承重为 m 的背包和 n 种物品，每种物品有价值 v_i 和重量 w_i ，每种物品都有无穷多个，求背包能装下的物品的最大价值和。

$$1 \leq n \leq 100, 1 \leq m \leq 10^5.$$

完全背包

dp 方程

设 $f(i, j)$ 为将前 i 种物品放入承重为 j 的背包的最大值，则

$$f(i, j) = \max(f(i - 1, j), f(i, j - w_i) + v_i), f(0, 0) = 0.$$

如果按 i 正序， j 正序的顺序更新，则同样可以用 $f(i, j)$ 的值覆盖 $f(i - 1, j)$ 的值而不影响转移。

完全背包

滚动空间优化

如果我们按 i 正序, j 倒序的顺序更新, 我们就可以直接用 $f(i, j)$ 的值覆盖 $f(i - 1, j)$ 的值而不影响转移时的 dp 值。

```
1 for (int i = 1; i <= n; ++i)
2     for (int j = w[i]; j <= m; ++j)
3         f[j] = max(f[j], f[j-w[i]] + v[i]);
```

给定 $A = \{x_1, \dots, x_n\}$, 若存在 $c_1, \dots, c_n \geq 0$ 使得 $s = \sum_{i=1}^n c_i x_i$, 则称 x_1, \dots, x_n 可以生成 s 。称 A 能生成的所有的 s 为 A 的生成集。

已知 $A = \{a_1, \dots, a_n\}$, 求最小的集合 S , 使得 S 的生成集与 A 的生成集相等。

$$n \leq 100, \sum_{i=1}^n a_i \leq 25000.$$

生成集是无限集，我们不可能一一验证。

但注意到生成关系的传递性（即 A 生成 B , B 生成 C , 则 A 生成 C ），我们只需找到最小的能生成 A 中所有数的子集即可。

将 A 从小到大排序， S 初始为空集。依次考虑每个数能否被 S 中的数生成。若不能生成则将其加入 S 中。

这是完全背包的可行性问题。复杂度 $O(n \sum a_i)$ 。

飞扬的小鸟

给一张 $n \times m$ 的网格图，其中某些格子是空地，某些格子是障碍。一开始你在 $(0, j_0)$ ， j_0 由你自己决定。

每一步，你可以

- 花费 k 的代价使你的格子从 (i, j) 移动到 $(i + 1, \min\{j + kx_i, m\})$ ；
- 不花费代价移动到 $(i + 1, j - y_i)$ 。

在任意时刻，只要移动到障碍上或纵坐标 ≤ 0 都算失败。

若能成功，求成功移动到地图最右端所需最小代价。否则，求能到达的最右位置。

$5 \leq n \leq 10000, 5 \leq m \leq 1000$ 。

飞扬的小鸟

题解

设 $f(i, j) =$ 在 (i, j) 的最小代价，如果达不到就是 ∞ 。

根据题目中的移动规则，转移方程为：

$$f(i, j) = \min\{f(i - 1, j - kx_i) + k, f(i - 1, j + y_i) | k \geq 1, j \geq kx_i\}$$

$$f(i, m) = \min\{f(i - 1, j) + \lceil \frac{m - j}{x_i} \rceil, f(i - 1, m) + 1 | 1 \leq j < m\}$$

注意能走的点用转移方程，不能走的点直接刷成 ∞ 。

状态 $O(nm)$ ，转移 $O(m)$ ，时间复杂度 $O(nm^2)$ 无法通过。

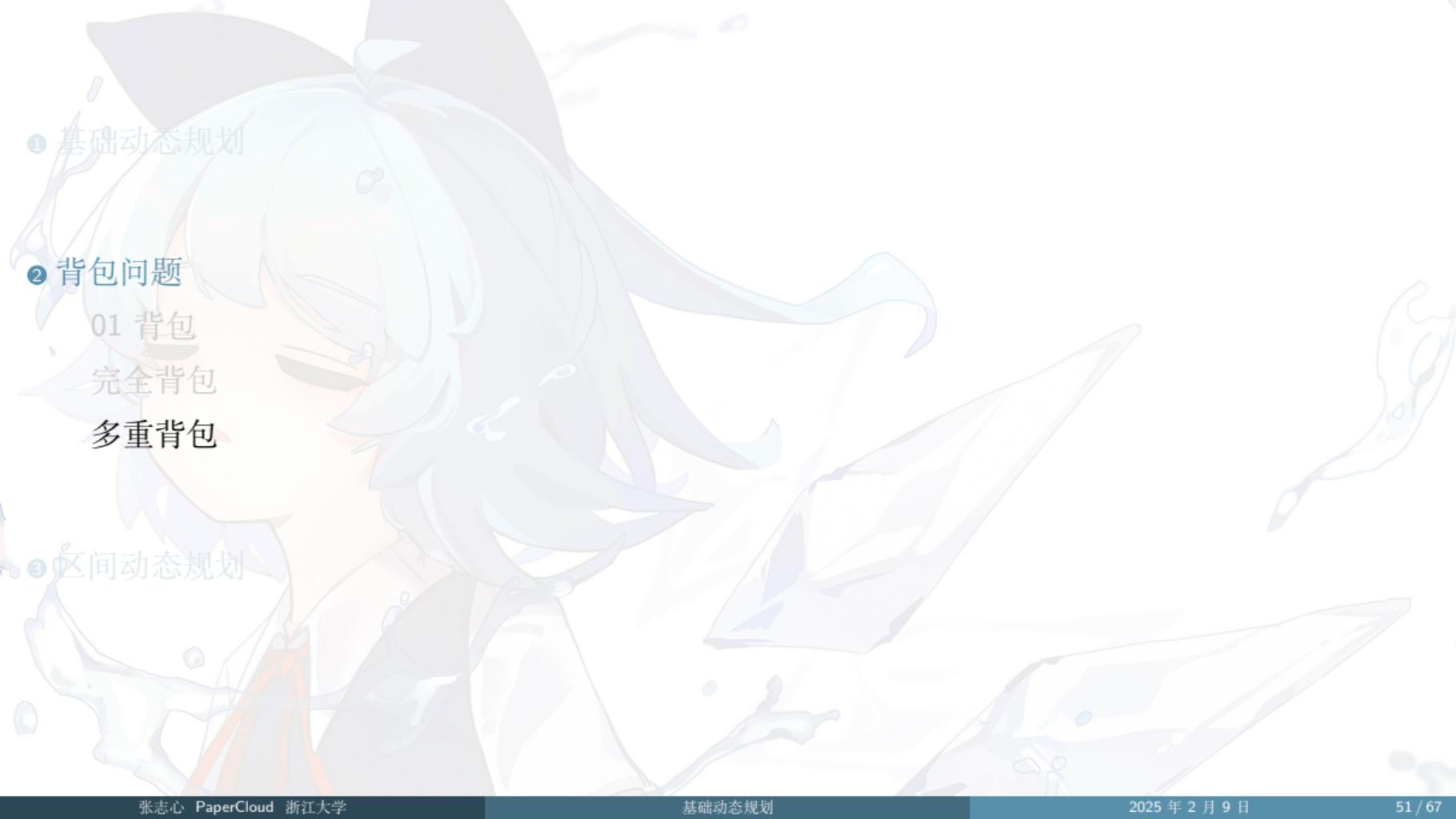
飞扬的小鸟

题解

注意到本题的转移和完全背包类似，我们直接利用完全背包的方法优化。

$$f(i, j) = \min\{f(i - 1, j - x_i) + 1, f(i - 1, j + y_i) + 1, \textcolor{red}{f(i, j - x_i) + 1}\}$$

$$f(i, m) = \min\{f(i - 1, j), f(i, j) | m - x_i \leq j \leq m\}$$



① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

多重背包

问题描述

有一个承重为 m 的背包和 n 种物品，每种物品有价值 v_i 和重量 w_i ，第 i 种物品有 c_i 个，求背包能装下的物品的最大价值和。

$$1 \leq n \leq 100, 1 \leq m \leq 1000, 1 \leq c_i \leq 100.$$

多重背包

朴素 dp

设 $f(i, j)$ 为将前 i 种物品放入承重为 j 的背包的最大值，则

$$f(i, j) = \max_{k \leq c_i} (f(i - 1, j - kw_i) + kv_i).$$

时间复杂度 $O(m \sum c_i)$, 空间复杂度 $O(nm)$ 。由于要枚举 k , 所以不能将 i 这一维滚动优化。

多重背包

二进制拆分

将 c_i 拆为 $1, 2, 2^2, \dots, 2^{k-1}, c_i - 2^k + 1$, 其中 $2^k - 1 \leq c_i \leq 2^{k+1} - 1$ 。把它们当成 $k+1$ 个不同物品。

不难发现这些数相加可以组成 0 到 c_i 的每个数，且不能组成大于 c_i 的数。

因此用这 $k+1$ 个重量为 $w_i, 2w_i, 2^2 w_i, \dots, (c_i - 2^k + 1)w_i$, 价值为 $v_i, 2v_i, 2^2 v_i, \dots, (c_i - 2^k + 1)v_i$ 的物品各一个即可代替这 c_i 个物品。

于是问题转化为 01 背包。时间复杂度为 $O(nm \log c_i)$, 空间复杂度为 $O(m)$ 。

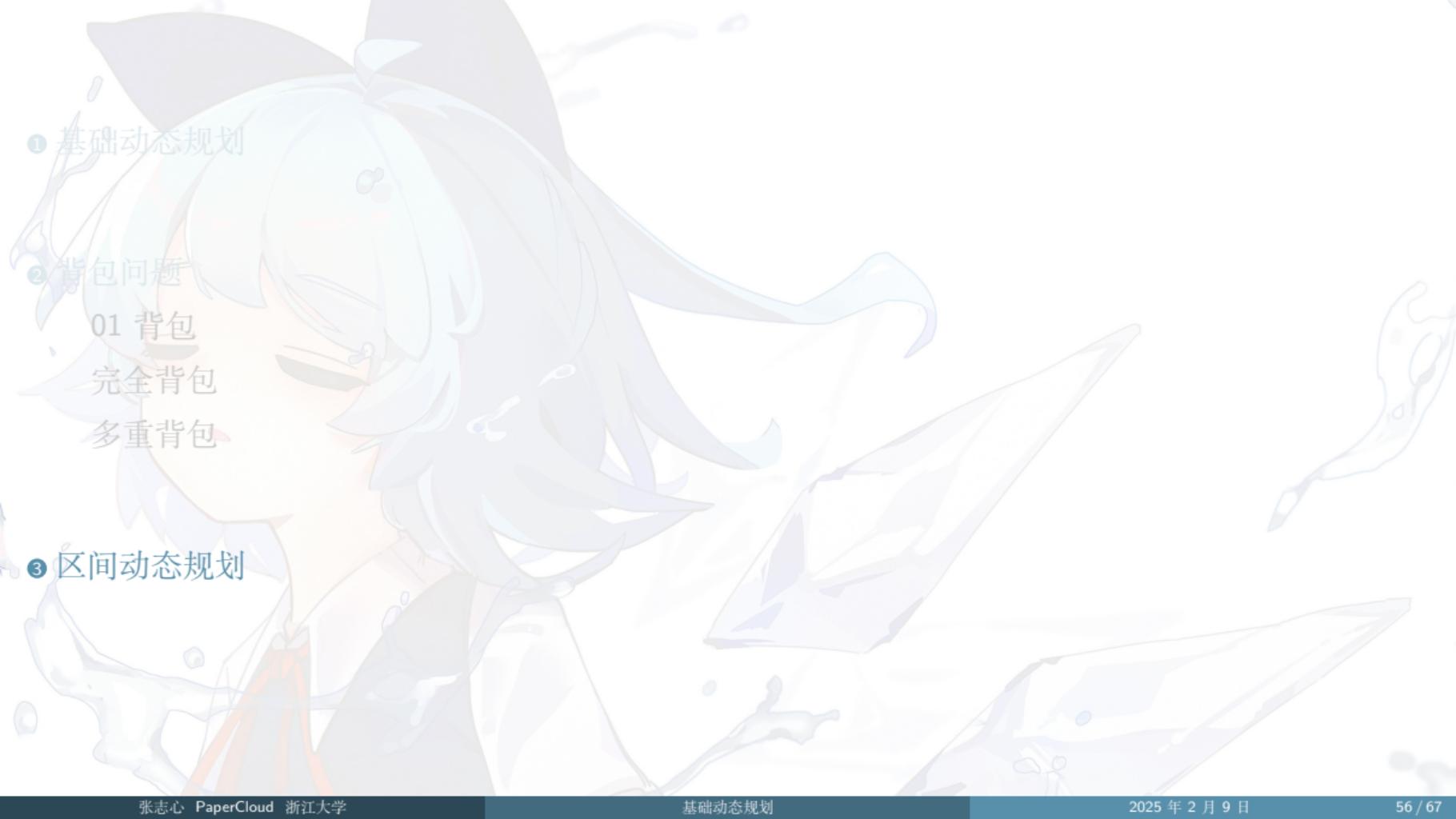
注意：二进制拆分只能用于最优化或可行性问题，不能用于方案数问题（会重复计数）。

多重背包

二进制拆分

参考实现：

```
1 for( int i=1;i<=n;++i){  
2     int l=0;  
3     while(1<<l<=c[i]+1)++l;  
4     --l;  
5     for( int k=0;k<l;++k)  
6         for( int w=a[i]<<k,r=b[i]<<k,j=u;j>=w;--j )  
7             f[j]=max(f[j],f[j-w]+r);  
8     int re=c[i]-(1<<l)+1;  
9     if(re>0)  
10        for( int w=a[i]*re,r=b[i]*re,j=u;j>=w;--j )  
11            f[j]=max(f[j],f[j-w]+r);  
12 }
```



① 基础动态规划

② 背包问题

01 背包

完全背包

多重背包

③ 区间动态规划

区间动态规划 介绍

区间动态规划解决决策涉及到相邻区间合并的问题。

它的转移方向是由小区间向大区间转移，所以在转移的时候，需要注意转移顺序。

区间动态规划的数据范围一般都比较小，因为它的状态是 $O(n^2)$ ，转移一般也要 $O(n)$ 以上。

区间动态规划

引入问题：石子合并

以石子合并问题为例：

有 n 个石子排成一列，第 i 个石子权值为 a_i ，一次操作可以将相邻的两个石子合并成一个石子，合并后的石子的权值为两个石子的权值之和，一次操作的代价等于合并后的石子的权值。求将所有石子合并成一个的最小的代价。

区间动态规划

引入问题：石子合并

有 n 个石子排成一列，第 i 个石子权值为 a_i ，一次操作可以将相邻的两个石子合并成一个石子，合并后的石子的权值为两个石子的权值之和，一次操作的代价等于合并后的石子的权值。求将所有石子合并成一个的最小的代价。

设 $f(l, r)$ 表示将区间 $[l, r]$, $1 \leq l \leq r \leq n$ 的石子合并的最小代价。

$$f(l, r) = \min\{f(l, k) + f(k + 1, r) \mid l \leq k \leq r\} + sum(l, r)$$

其中 $sum(l, r) = \sum_{k=l}^r a_k$ ，初始条件为 $f(i, i) = 0, 1 \leq i \leq n$ 。最终答案为 $f(1, n)$ 。

区间动态规划

引入问题：石子合并

转移的时候，要注意从小区间转移到大区间。参考代码如下：(令 $s_i = \sum_{j=1}^i a_j$)

```
1 for (int len=2; len<=n; ++len)
2     for (int i=1; i<=n-len+1; ++i){
3         int j=i+len-1;
4         for (int k=i; k<j; ++k)
5             f[i][j]=min(f[i][j], f[i][k]+f[k+1][j]+s[j]-s[i-1]);
6     }
```

或

```
1 for (int i=n-1; i; --i)
2     for (int j=i+1; j<=n; ++j)
3         for (int k=i; k<j; ++k)
4             f[i][j]=min(f[i][j], f[i][k]+f[k+1][j]+s[j]-s[i-1]);
```

能量项链

n 个数对 $(a_1, a_2), (a_2, a_3), \dots, (a_n, a_1)$ 排成一个环，每次可以合并相邻两个数对 (a_i, a_k) 和 (a_k, a_j) ，成为新数对 (a_i, a_j) ，并得到 $a_i a_j a_k$ 的价值。求最大价值。

$n \leq 100$ 。

能量项链

题解

破坏为链 把原数组复制一份接在后面，即可将环的问题转化为序列的问题。

设 $f(i, j) =$ 将 (i, j) 合并的最大价值，则

$$f(i, j) = \max\{f(i, k) + f(k + 1, j) + a_i a_{k+1} a_{j+1} \mid i \leq k < j\}$$

初始状态为 $f(i, i) = 0$ ，目标状态为 $\max_{i=1}^n \{f(i, i + n - 1)\}$ 。

注意因为 $f(i, j)$ 的转移方程中有 a_{j+1} ，所以还要 $a_{2n+1} = a_1$ 。

复杂度 $O(n^3)$ 和 $O(n^2)$ 。

括号序列

- 空序列是合法括号序列；
- 若 S 是合法括号序列，则 (S) 和 $[S]$ 是合法括号序列；
- 若 S 和 T 是合法括号序列，则 ST 是合法括号序列。

给一个包含左右圆括号和左右方括号的括号序列，添加最少的字符使其合法。

括号序列

题解

设 $f(i, j)$ 表示将 i 到 j 合并需要添加的字符的最少数量，则

$$f(i, j) = \min\{f(i + 1, j) + 1, f(i, j - 1) + 1, f(i, k) + f(k + 1, j) | i \leq k < j\}$$

$$f(i, j) = \min\{f(i, j), f(i + 1, j - 1)\} (a_i \ a_j \)$$

$$f(i, i) = 1 \quad f(i, i - 1) = 0$$

多边形游戏

给一个 n 边形，每个节点有权值 v_i ，边上也有运算符 + 或 *。先任选一条边删去，之后每一步合并相邻的两个顶点，新点为两个点的权值经边上的运算符运算得到的值。求最后得到的点的最大权值。

$n \leq 50, -10 \leq a_i \leq 10$ (注意这里特别提到 a_i 可能是负数)

先破坏为链。

设 $f(i, j)$ 为区间 (i, j) 合并的最大权值，枚举断点 k ，则

如果 $(k, k + 1)$ 之间的符号为 $+$ ，两边都取最大值即可；

如果 $(k, k + 1)$ 之间的符号为 $*$ ，因为负负得正，所以两边都取最大值或两边都取最小值都可能得到最大的乘积。

因此还要维护 $g(i, j) = \text{区间 } (i, j) \text{ 合并的最小权值}。$

多边形游戏

题解

现在推导转移方程。

对于加法操作：

- $f(l, r) \leftarrow f(l, k) + f(k + 1, r)$
- $g(l, r) \leftarrow g(l, k) + g(k + 1, r)$

对于乘法操作：左右两区间的 \max 和 \min 互相相乘共会产生四个值 ($\max * \max$, $\min * \min$, $\max * \min$, $\min * \max$)，显然相乘结果的最大、最小值均会在这四个值中取得。因此我们取这四个值中的最大值更新 $f(l, r)$, 最小值更新 $g(l, r)$ 即可。