动态规划 Dynamic Programming

GKxx

视频在这里

最大子段和

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1} \rangle$，求 $\max\left\{\sum_{i=l}^{r} a_i \mid 0 \leqslant l \leqslant r < n\right\}$。

最大子段和

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1}\rangle$ ，求 $\max\left\{\sum_{i=l}^{r} a_i \mid 0 \leqslant l \leqslant r < n\right\}$ 。

设 f_i 表示**以** a_i **结尾的**最大子段和。最终答案是 $\max_{i=0}^{n-1}\{f_i\}$ 。

- 考虑 a_i 应该和前一段连在一起，还是"另起炉灶"？
  - 另起炉灶，那么答案就是 a_i
  - 和前一段连在一起，那么答案是 $f_{i-1} + a_i$
- $$f_i = \max\{f_{i-1} + a_i, a_i\}$$

最大子段和

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1}\rangle$ ，求 $\max\left\{\sum_{i=l}^{r} a_i \mid 0 \leqslant l \leqslant r < n\right\}$ 。

设 f_i 表示**以 a_i 结尾的**最大子段和。最终答案是 $\max_{i=0}^{n-1}\{f_i\}$ 。

$$f_i = \max\{f_{i-1} + a_i, a_i\}$$

```cpp
std::vector f(n, 0ll);
f[0] = a[0];
for (auto i = 1; i != n; ++i)
  f[i] = std::max(f[i - 1] + a[i], a[i]);
std::cout << std::ranges::max(f) << std::endl;
```

- 这个问题足够简单：从 $f_i = a_i + \max\{f_{i-1}, 0\}$ 的角度理解，这也是一种贪心。

动态规划的第一层理解：**稍微复杂一点的递推**。

最长上升子序列

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1} \rangle$，求最长的子序列 $a_{s_0}, a_{s_1}, \cdots, a_{s_{k-1}} (s_0 < s_1 < \cdots < s_{k-1})$ 满足 $a_{s_0} \leqslant a_{s_1} \leqslant \cdots \leqslant a_{s_{k-1}}$。

最长上升子序列

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1} \rangle$，求最长的子序列 $a_{s_0}, a_{s_1}, \cdots, a_{s_{k-1}} (s_0 < s_1 < \cdots < s_{k-1})$ 满足 $a_{s_0} \leqslant a_{s_1} \leqslant \cdots \leqslant a_{s_{k-1}}$。

设 f_i 表示 $\langle a_0, a_1, \cdots, a_i \rangle$ 中的最长上升子序列长度？那么最终答案是 f_{n-1}。

- 和前一个子问题相比，增加的信息是 a_i。
- 考虑 a_i 带来的影响：它可能在最长上升子序列中，也可能不在。
  - 不在，那么答案是 f_{i-1}。
  - 在，那它会接在谁的后面？你怎么知道能接不能接？

最长上升子序列

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1} \rangle$ ，求最长的子序列 $a_{s_0}, a_{s_1}, \cdots, a_{s_{k-1}} (s_0 < s_1 < \cdots < s_{k-1})$ 满足 $a_{s_0} \leqslant a_{s_1} \leqslant \cdots \leqslant a_{s_{k-1}}$ 。

设 f_i 表示**以** a_i **结尾的**最长上升子序列长度？那么最终答案是 $\max_{i=0}^{n-1} f_i$ 。

- a_i 可能会接在前面的某一个 a_j 之后，只要 $a_j \leqslant a_i$ 即可。
- $f_i = \max\{f_j + 1 \mid 0 \leqslant j < i, a_j \leqslant a_i\}$

最长上升子序列

给定一个序列 $\langle a_0, a_1, \cdots, a_{n-1} \rangle$ ，求最长的子序列 $a_{s_0}, a_{s_1}, \cdots, a_{s_{k-1}} (s_0 < s_1 < \cdots < s_{k-1})$ 满足 $a_{s_0} \leqslant a_{s_1} \leqslant \cdots \leqslant a_{s_{k-1}}$ 。

设 f_i 表示**以 a_i 结尾的**最长上升子序列长度？那么最终答案是 $\max_{i=0}^{n-1} f_i$ 。

- a_i 可能会接在前面的某一个 a_j 之后，只要 $a_j \leqslant a_i$ 即可。
- $f_i = \max\{f_j + 1 \mid 0 \leqslant j < i, a_j \leqslant a_i\}$

时间复杂度：状态量为 $O(n)$ ，计算一个 f_i 需要枚举 $j = 0, \cdots, i-1$ ，所以 总时间复杂度为 $O\left(n^2\right)$ 。

> 这个问题可以做到 $O(n \log n)$ ，但它超出了 CS101 的范围。

最长上升子序列

$$f_i = \max\{f_j + 1 \mid 0 \leqslant j < i, a_j \leqslant a_i\}$$

```cpp
std::vector f(n, 0);
for (auto i = 0; i != n; ++i) {
  f[i] = 1;
  for (auto j = 0; j != i; ++j)
    if (a[j] <= a[i] && f[j] + 1 > f[i])
      f[i] = f[j] + 1;
}
std::cout << std::ranges::max(f) << std::endl;
```

0-1 背包

有 n 个物品和一个载重为 W 的背包，第 $i(0 \leqslant i < n)$ 个物品的重量是 w_i ，价值是 v_i 。如何选择一些物品装入背包，使得价值总和最大？

- “0-1”背包：每个物品要么选，要么不选。

0-1 背包

有 n 个物品和一个载重为 W 的背包，第 $i(0 \leqslant i < n)$ 个物品的重量是 w_i ，价值是 v_i 。如何选择一些物品装入背包，使得价值总和最大？

设 $f(i,j)$ 表示在前 i 个物品中选择一些，放入载重为 j 的背包，最大价值和是多少，那么答案就是 $f(n-1,W)$ 。

> 先别管怎么想到定义 f 的

- 第 i 个物品选还是不选？

0-1 背包

有 n 个物品和一个载重为 W 的背包，第 $i(0 \leqslant i < n)$ 个物品的重量是 w_i ，价值是 v_i 。如何选择一些物品装入背包，使得价值总和最大？

设 $f(i, j)$ 表示在前 i 个物品中选择一些，放入载重为 j 的背包，最大价值和是多少。

> 先别管怎么想到定义 f 的

- 第 i 个物品选还是不选？
  - 选，前提是 $w_i \leqslant j$ ，选了之后背包就还剩下 $j - w_i$ 的载重，这种情况下的答案是 $f(i-1, j-w_i) + v_i$ 。
  - 不选，那么答案是 $f(i-1, j)$ 。
- $$f(i, j) = \begin{cases} \max\{f(i-1, j-w_i) + v_i, f(i-1, j)\}, & w_i \leqslant j, \\ f(i-1, j), & \text{otherwise.} \end{cases}$$

0-1 背包

设 $f(i,j)$ 表示在前 i 个物品中选择一些，放入载重为 j 的背包，最大价值和是多少。

$$f(i,j)=\begin{cases}\max\{f(i-1,j-w_i)+v_i, f(i-1,j)\}, & w_i\leqslant j,\\ f(i-1,j), & \text{otherwise.}\end{cases}$$

时间复杂度：**状态量** $\times$ **转移平均复杂度**

- 为了求出最后的答案，我们需要把所有 $f(*,*)$ 全都算一遍，一共有 nW 个。
- 按照转移方程计算一个 $f(i,j)$ 需要 $O(1)$ 的时间。
- 所以时间复杂度为 $O(nW)$ 。

空间复杂度： f 数组需要 $O(nW)$ ，除此之外没了。

可以对着代码再看看复杂度。

0-1 背包

设 $f(i,j)$ 表示在前 i 个物品中选择一些，放入载重为 j 的背包，最大价值和是多少。

$$f(i,j)=\begin{cases}\max\{f(i-1,j-w_i)+v_i,f(i-1,j)\}, & w_i\leqslant j,\\ f(i-1,j), & \text{otherwise.}\end{cases}$$

代码：“`for` 一 `for` 就好了”

```
std::vector f(n, std::vector(w + 1, 0ll));
for (auto j = 0; j <= w; ++j)
  f[0][j] = weight[0] <= j ? value[0] : 0;
for (auto i = 1; i != n; ++i)
  for (auto j = 0; j <= w; ++j) {
    f[i][j] = f[i - 1][j];
    if (weight[i] <= j && f[i - 1][j - weight[i]] + value[i] > f[i][j])
      f[i][j] = f[i - 1][j - weight[i]] + value[i];
  }
std::cout << f.back().back() << std::endl;
```

0-1 背包

设 $f(i,j)$ 表示在前 i 个物品中选择一些，放入载重为 j 的背包，最大价值和是多少。

$$f(i,j)=\begin{cases}\max\{f(i-1,j-w_i)+v_i,f(i-1,j)\}, & w_i\leqslant j,\\ f(i-1,j), & \text{otherwise.}\end{cases}$$

一个简单的空间优化：**滚动数组**。考虑到 $f(i,\cdot)$ 只依赖于 $f(i-1,\cdot)$，我们可以直接把第一维 $\bmod 2$，空间复杂度降到了 $O(W)$

```
std::vector f(2, std::vector(w + 1, 0ll));
for (auto j = 0; j <= w; ++j)
  f[0][j] = weight[0] <= j ? value[0] : 0;
for (auto i = 1; i != n; ++i)
  for (auto j = 0; j <= w; ++j) {
    f[i % 2][j] = f[(i - 1) % 2][j];
    if (weight[i] <= j
        && f[(i - 1) % 2][j - weight[i]] + value[i] > f[i % 2][j])
      f[i % 2][j] = f[(i - 1) % 2][j - weight[i]] + value[i];
  }
```

伪多项式时间 (pseudo-polynomial time)

$O(nW)$ 是多项式吗？**如是**。

- “多项式时间算法”：算法的运行时间不超过一个关于**输入的长度**的多项式。
- W 是**输入的值的大小**，而非输入长度。
- n 也是输入的值？不不不， n 只是为了方便你输入而已，它实际上是 $\{w_i\}$ 和 $\{v_i\}$ 序列的长度。

伪多项式时间 (pseudo-polynomial time)

$O(nW)$ 是多项式吗？**如是**。

- “多项式时间算法”：算法的运行时间不超过一个关于**输入的长度**的多项式。
- W 是**输入的值的大小**，而非输入长度。
- n 也是输入的值？不不不， n 只是为了方便你输入而已，它实际上是 $\{w_i\}$ 和 $\{v_i\}$ 序列的长度。

这个 $O(nW)$ 的算法是**伪多项式时间**的，而非“多项式时间”的。

- 它只是看起来很好而已，实际上当 $n = 10, W = 10^9$ 的时候，它还不如 $O\left(2^n\right)$ 地枚举子集快！

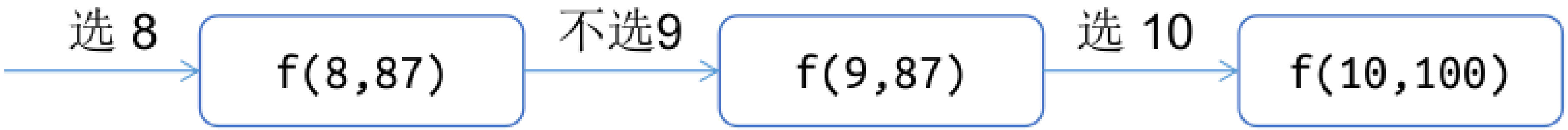
0-1 背包：记录方案

如何记录你选取的物品是哪些？

0-1 背包：记录方案

在解决一个子问题 $f(i,j)$ 的时候，我们比较了两种**决策**：选或不选第 i 个物品。

为每一个子问题（状态）记录当时的最优决策，然后就可以顺藤摸瓜地找出转移路径。

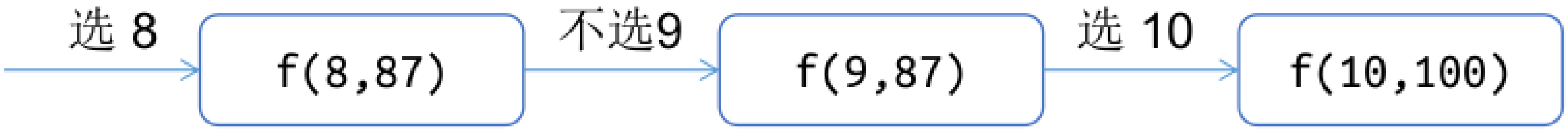



```
std::vector choose(n, std::vector(w + 1, false));
// ...
for (auto i = 1; i != n; ++i)
  for (auto j = 0; j <= w; ++j) {
    f[i][j] = f[i - 1][j];
    if (weight[i] <= j && f[i - 1][j - weight[i]] + value[i] > f[i][j]) {
      f[i][j] = f[i - 1][j - weight[i]] + value[i];
      choose[i][j] = true;
    }
  }
```

0-1 背包：记录方案

在解决一个子问题 $f(i,j)$ 的时候，我们比较了两种**决策**：选或不选第 i 个物品。

为每一个子问题（状态）记录当时的最优决策，然后就可以顺藤摸瓜地找出转移路径。




```cpp
std::vector<int> selected_items;
for (int i = n - 1, j = w; i >= 0; --i) {
  if (choose[i][j]) {
    selected_items.push_back(i);
    j -= weight[i];
  }
}
```

有没有在哪见过？

最短路径：记路径

为每个结点 `x` 记录 `pre[x]` ，表示从起点到 `x` 的最短路径上 `x` 的前一个结点是谁。松弛时更新。

```
if (dist[v] + w < dist[x]) {
  dist[x] = dist[v] + w;
  pre[x] = v;
}
```

```
if (f[i - 1][j - weight[i]] + value[i] > f[i][j]) {
  f[i][j] = f[i - 1][j - weight[i]] + value[i];
  choose[i][j] = true;
}
```

有点儿像？

转化为最长路径问题

建一张图，结点是二元组 $(i,j), i \in \{-1,0,1,\cdots,n-1\}, j \in \{0,1,\cdots,W\}$ ，按照转移方程连边：

$$f(i,j) = \begin{cases} \max\{f(i-1,j-w_i)+v_i, f(i-1,j)\}, & w_i \leqslant j, \\ f(i-1,j), & \text{otherwise.} \end{cases}$$

- 从 $(i-1,j)$ 向 (i,j) 连一条边，边权为 0 。
- 如果 $w_i \leqslant j$ ，再从 $(i-1,j-w_i)$ 向 (i,j) 连边，边权为 v_i 。

问题转化为这张图上从 $(-1,*)$ 到 $(n-1,W)$ 的最**长**路径！

- 添加一个超级源 s 向所有 $(-1,*)$ 连边。
- 除了 Dijkstra 之外的最短路算法都能求最长路径，边权取负即可。

转化为最长路径问题

建一张图，结点是二元组 $(i,j), i \in \{-1,0,1,\cdots,n-1\}, j \in \{0,1,\cdots,W\}$ ，按照转移方程连边：

- 从 $(i-1,j)$ 向 (i,j) 连一条边，边权为 0 。
- 如果 $w_i \leqslant j$ ，再从 $(i-1,j-w_i)$ 向 (i,j) 连边，边权为 v_i 。
- 添加一个超级源 s 向所有 $(-1,*)$ 连边。

问题转化为这张图上从 s 到 $(n-1,W)$ 的最**长**路径。

- **这张图是一个 DAG ！** 我们只需按照拓扑序遍历，就能求出最长路径。
- **拓扑序正好是 i 从小到大、 j 从小到大！** 所以“`for` 一 `for` 即可”。

动态规划的第二层理解：状态转移图

状态 $\Longleftrightarrow$ 图的结点

转移 $\Longleftrightarrow$ 图的边

- 如果状态 v 能从状态 u 转移而来，就连边 (u, v) 。

按照拓扑序遍历各个结点，计算各个状态的答案。

- 如果拓扑序比较简单，直接“`for` 一 `for`”就好了。
- 如果拓扑序比较复杂，或者图根本不是 DAG ，就需要一个图论算法： DFS, BFS, 最短路等等。

练习：画出最长上升子序列问题的状态转移图。

矩阵乘法

计算一个 $m \times n$ 的矩阵和一个 $n \times p$ 的矩阵的乘积，需要 mnp 次标量乘法。现在我们试图计算

$$A_0 A_1 \cdots A_{n-1}, \quad A_i \in \mathbb{R}^{p_i \times p_{i+1}}.$$

如何给这个式子加括号，使得总的标量乘法次数最少？

矩阵乘法

计算一个 $m \times n$ 的矩阵和一个 $n \times p$ 的矩阵的乘积，需要 mnp 次标量乘法。现在我们试图计算

$$A_0A_1\cdots A_{n-1}, \quad A_i \in \mathbb{R}^{p_i \times p_{i+1}}.$$

如何给这个式子加括号，使得总的标量乘法次数最少？

设 $f(i)$ 表示计算 $A_0A_1\cdots A_i$ 所需的最少标量乘法次数？

- 最后一步乘法可能是 $(A_0A_1\cdots A_k)\,(A_{k+1}A_{k+2}\cdots A_i)$ ，它需要 $p_0p_{k+1}p_{i+1}$ 次标量乘法。

矩阵乘法

计算一个 $m \times n$ 的矩阵和一个 $n \times p$ 的矩阵的乘积，需要 mnp 次标量乘法。现在我们试图计算

$$A_0 A_1 \cdots A_{n-1}, \quad A_i \in \mathbb{R}^{p_i \times p_{i+1}}.$$

如何给这个式子加括号，使得总的标量乘法次数最少？

设 $f(i)$ 表示计算 $A_0 A_1 \cdots A_i$ 所需的最少标量乘法次数？

- 最后一步乘法可能是 $(A_0 A_1 \cdots A_k)\,(A_{k+1} A_{k+2} \cdots A_i)$，它需要 $p_0 p_{k+1} p_{i+1}$ 次标量乘法。

  $$f(i) = \min\{f(k) + ??? + p_0 p_{k+1} p_{i+1} \mid 0 \leqslant k < i\}.$$

  好像转移不了，缺少 $(A_{k+1} A_{k+2} \cdots A_i)$ 的信息。

矩阵乘法

计算一个 $m \times n$ 的矩阵和一个 $n \times p$ 的矩阵的乘积，需要 mnp 次标量乘法。现在我们试图计算

$$A_0 A_1 \cdots A_{n-1}, \quad A_i \in \mathbb{R}^{p_i \times p_{i+1}}.$$

如何给这个式子加括号，使得总的标量乘法次数最少？

设 $f(l, r)$ 表示计算 $A_l A_{l+1} \cdots A_r$ 所需的最少标量乘法次数。

- 最后一步乘法可能是 $(A_l A_{l+1} \cdots A_k)(A_{k+1} A_{k+2} \cdots A_r)$，它需要 $p_l p_{k+1} p_{r+1}$ 次标量乘法。

$$f(l, r) = \min\{f(l, k) + f(k+1, r) + p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

矩阵乘法

设 $f(l,r)$ 表示计算 $A_l A_{l+1} \cdots A_r$ 所需的最少标量乘法次数。

$$f(l,r) = \min\{f(l,k) + f(k+1,r) + p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

时间复杂度：**状态数** $\times$ **平均转移**

- 状态数为 $O\left(n^2\right)$ ，求解一个 $f(l,r)$ 需要枚举 $k = l, l+1, \cdots, r-1$ ，所以时间复杂度为 $O\left(n^3\right)$ 。
- 也可以严格地计算

$$\sum_{0 \leqslant l \leqslant k < r < n} 1 = \sum_{0 \leqslant l < k+1 < r+1 \leqslant n} 1 = \binom{n+1}{3} = O\left(n^3\right).$$

矩阵乘法

设 $f(l, r)$ 表示计算 $A_l A_{l+1} \cdots A_r$ 所需的最少标量乘法次数。

$$f(l, r) = \min\{f(l, k) + f(k+1, r) + p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

代码怎么写？

矩阵乘法

设 $f(l,r)$ 表示计算 $A_l A_{l+1} \cdots A_r$ 所需的最少标量乘法次数。

$$f(l,r) = \min\{f(l,k) + f(k+1,r) + p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

能不能这样？

```
for (auto l = 0; l < n; ++l)
  for (auto r = l + 1; r < n; ++r) {
    f[l][r] = infinity;
    for (auto k = l; k < r; ++k) {
      auto result = f[l][k] + f[k + 1][r] + p[l] * p[k + 1] * p[r + 1];
      if (result < f[l][r])
        f[l][r] = result;
    }
  }
```

矩阵乘法

$$f(l,r)=\min\{f(l,k)+f(k+1,r)+p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

能不能这样？

```
for (auto l = 0; l < n; ++l)
  for (auto r = l + 1; r < n; ++r) {
    f[l][r] = infinity;
    for (auto k = l; k < r; ++k) {
      auto result = f[l][k] + f[k + 1][r] + p[l] * p[k + 1] * p[r + 1];
      if (result < f[l][r])
        f[l][r] = result;
    }
  }
```

在计算 $f(l,r)$ 的时候， $f(l,k)$ 和 $f(k+1,r)$ 求过了吗？

- $\Leftrightarrow$ 这个 DP 的状态转移的拓扑序是什么？

矩阵乘法

$$f(l,r)=\min\{f(l,k)+f(k+1,r)+p_l p_{k+1} p_{r+1} \mid l \leqslant k < r\}.$$

在求 $f(l,r)$ 的时候，必须保证 $\forall k \in [l,r)$ ， $f(l,k)$ 和 $f(k+1,r)$ 都已经求过了。

矩阵乘法

$$f(l,r)=\min\{f(l,k)+f(k+1,r)+p_lp_{k+1}p_{r+1}\mid l\leqslant k<r\}.$$

在求 $f(l,r)$ 的时候，必须保证 $\forall k\in[l,r)$ ， $f(l,k)$ 和 $f(k+1,r)$ 都已经求过了。

按照区间的长度，从小到大计算！

```
for (auto len = 2; len < n; ++len)
  for (auto l = 0; l + len - 1 < n; ++l) {
    auto r = l + len - 1;
    f[l][r] = infinity;
    for (auto k = l; k < r; ++k) {
      auto result = f[l][k] + f[k + 1][r] + p[l] * p[k + 1] * p[r + 1];
      if (result < f[l][r])
        f[l][r] = result;
    }
  }
```

所以，状态怎么设计？

南大 JYY 在 JSOI2018 夏令营上说：

- “动态规划的状态是对搜索空间的概括”
- “找到搜索空间中的冗余 $\Leftrightarrow$ 定义动态规划的状态”

但是这远远超出了 CS101 的范围 ……

所以，状态怎么设计？

CS101 会涉及的三类模型：

- "设 $f(i)$ 表示前 i 个东西 / 以第 i 个东西结尾的 ……"：**前缀**
- "设 $f(i,j)$ 表示前 i 个东西，某属性之和不超过 / 恰好等于 / ... j"：**背包**
- "设 $f(l,r)$ 表示下标区间为 $[l,r]$ 中的这几样东西 ……"：**区间**

哪怕对于竞赛生来说，在起步阶段也是靠**刷题**，见多识广。