数据结构与算法
（十）
动态规划进阶

杜育根

Ygdu@sei.ecnu.edu.cn

10. 动态规划进阶

- 本课程，我们将介绍更多形式的动态规划，有区间动态规划、树形动态规划，然后介绍几类决策问题的优化方法——单调队列优化，斜率优化，四边形不等式优化。

10.1. 区间动态规划

- 区间 DP：是指在一段区间上进行的一系列动态规划。
- 对于区间 DP 这一类问题，我们需要计算区间 [1,n] 的答案，通常用一个二维数组 dp表示，其中 $dp[x][y]$ 表示区间 [x,y]。
- 有些题目，$dp[l][r]$ 由 $dp[l][r-1]$ 与 $dp[l+1][r]$ 推得；也有些题目，我们需要枚举区间 $[l,r]$内的中间点，由两个子问题合并得到，也可以说 $dp[l][r]$ 由 $dp[l][k]$ 与 $dp[k+1][r]$ 推得，其中 $l \leq k < r$。
- 对于长度为n的区间DP，我们可以先计算 [1,1],[2,2]...[n,n]的答案，再计算 [1,2],[2,3]...[n−1,n]，以此类推，直到得到原问题的答案。

例 1：石子归并问题

- 题目描述：当前有N堆石子，他们并列在一排上，每堆石子都有一定的数量。我们需要把这些石子合并成为一堆，每次合并都只能把 相邻 的两堆合并到一起，每一次合并的代价都是这两堆石子的数量之和，经过 N−1 次合并后成为一堆。求把这些石子合并成一堆所需的最小代价。
- 解析：根据动态规划的思想，我们只要求出每两堆石子合并的最小代价，然后再求出每三堆石子合并的最小代价，并以此类推就能最终求出 n堆石子合并的最小代价。
- 我们把 $dp[i][j]$ 定义为合并第 i 堆石子到第 j堆石子所需的最小代价。
- 很容易就能得到 $dp[i][j] = min(dp[i][k] + dp[k+1][j]) + sum[j] - sum[i-1]$。显然通过这个式子，我们可以按区间长度从小到大的顺序进行枚举来不断让石子进行合并，最终就能获得合并成一堆石子的最小代价。时间复杂度是 $O(n^3)$。

```
1.  memset(dp, 0, sizeof(dp));
2.  for (int l = 2; l <= n; ++l) {
3.      for (int i = 1, j; i <= n - l + 1; ++i) {
4.          j = i + l - 1;
5.          dp[i][j] = inf;
```

```
6.      for (int k = i; k < j; ++k) {
7.          dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
8.          }
9.          dp[i][j] += sum[j] - sum[i - 1];
10.     }
11. } //其中 sum表示预处理的石子的前缀和。
```

例 2：括号匹配问题

- 题目描述: 当前有一个串，它由(、)、[、]四种括号组成，其中每种括号的次序或数量不定，求最少向这个串中插入多少个括号就能够使整个串中所有的括号左右配对。如：当前的串为([])，向其中添加一个[就能达到我们的要求。
- 解析: 我们可以求出当前这个串中，能够一一配对的括号的总数，这里称之为最大匹配，然后再用串的总长度减去最大匹配就能够得到我们需要的答案。
- 如：当前的串为([])，它的最大匹配就是 4。
- 因此，现在我们的问题就转变成了如何求出当前这个串的最大匹配数目。我们可以用 dp[i][j] 来表示从第 i个括号到第 j个扩号之间的最大匹配数目。然后根据第 i 个和第 j 个括号是否匹配，以及 dp[i][j] 的值很容易的得出 dp[i+1][j−1] 了。
- 我们可以枚举所有的 dp[i][j]，来求出这个串的最大匹配数，再用这个串的总长度−最大匹配数就是我们所求的答案。
- 在枚举过程中：//judge() 函数功能为判断第 i 个括号与第 j 个括号是否匹配,若匹配返回 true，否则返回 false。

$$dp[i][j] = judge(i,j)\ ?\ dp[i+1][j-1] + 2 : dp[i+1][j-1];$$

- 并且 dp[i][j] 也可以通过把区间 [i,j]，分成区间 [i][k] 和区间 [k+1][j] 来得到，即：dp[i][j]=dp[i][k]+dp[k+1][j]。我们枚举 k，取最大值即可。
- 状态转移方程为：$dp[i][j] = max(dp[i][j], dp[i][k] + dp[k+1][j])$ 。

```
memset(dp, 0, sizeof(dp));
for (int l = 2; l <= n; ++l) {
    for (int i = 1, j; i <= n - l + 1; ++i) {
        j = i + l - 1;
        dp[i][j] = judge(i, j) ? dp[i + 1][j-1] + 2 : dp[i + 1][j - 1];
        for (int k = i; k < j; ++k) {
            dp[i][j] = max(dp[i][j], dp[i][k] + dp[k + 1][j]);
        }
    }
}
```

例 3：整数划分问题

- 题目描述: 给你一个长度为 n的数字序列，要求在这个数字序列中加入 m−1 个乘号，使得这个式子的结果最大。
- 解析: 定义 dp[i][j] 为到第 i个数结尾共加入 j−1 个乘号所得到的最大值。
- 我们依次枚举 i, j，计算 dp[i][j]，即可得到最后的答案。
- 状态转移方程为：dp[i][j]=max(dp[i][j],dp[k][j−1]×a[k+1][i]) ，其中 a[i][j] 表示数字序列中从第 i 个数到第 j个数连起来的值。

```
memset(dp, 0, sizeof(dp));
for (int i = 1; i <= n; n++) {
    dp[i][1] = a[1][i];
}
for (int j = 2; j <= m; j++) {
    for (int i = j; i <= n; i++) {
        for (int k = 1; k < i; k ++) {
            dp[i][j] = max(dp[i][j], dp[k][j - 1] * a[k + 1][i]);
        }
    }
}
printf("%lld\n", dp[n][m]);
```

习题：奇怪的二叉树

- 小明最近遇到一道难题，想请聪明的你来帮忙解决一下。题目给了一棵奇怪的二叉树，树上有n个结点，每个结点按中序遍历的顺序依次编号为 1到 n。每个结点都有一个权值，第 i个结点的权值为 wi。每棵子树也有一个权值，记 $s(i)$为以编号 i为根结点的子树的权值，结点 i 的左孩子结点编号为 i_{left}，右孩子结点编号为 i_{right}，则 s(i)的计算方法为：$s(i) = s(i_{left}) \times s(i_{right}) + wi.$
- 需要注意的是，如果结点 i为叶子结点，则 $s(i) = wi$。如果结点i不是叶子结点，但是其左子树为空，则 $s(ileft) = 1$；同样，如果结点 i的右子树为空，则 $s(iright) = 1$。
- 现在需要你来设计一棵二叉树，使得中序遍历得到的结点序列为1到n，且使二叉树的根结点 $root$ 的 $s(root)$ 最大。
- 输入格式: 输入有两行。输入第一行是一个整数 n（1≤n≤30），表示二叉树上一共有 n个结点。输入第二行是 n个整数，每两个整数用一个空格隔开，第i个整数对应第i个结点的权值 wi $(1 \leq wi \leq 100)$。
- 输出格式: 输出两行。第一行输出一个整数，表示二叉树根结点 root 能获得的最大 s(root)，结果保证在 int 范围内。第二行输出 n 个整数，每两个整数之间用一个空格隔开。输出二叉树的前序遍历结果。
- 本题答案不唯一，符合要求的答案均正确
- 样例输入
- 6
- 3 5 1 7 9 4
- 样例输出
- 297
- 3 1 2 5 4 6

习题：卡牌游戏

- 小明手里有 n张卡牌，编号从 1 到 n，每张卡牌上面有一个数字 num_i。现在小明将 n 张卡牌排成一行，组成一个序列，执行以下操作：从序列中抽取一张编号为 i的卡牌，则该张卡牌贡献的得分为 $num_{i-1} \times num_i \times num_{i+1}$，即卡牌上的数字同左右两张相邻的卡牌上的数字乘积。但是不能抽取序列中最左边和最右边的卡牌，即 i≠1 且 i≠n。抽到的卡牌就从序列中去掉。重复上述操作，直到序列里只剩两张卡牌。抽取的总得分为每次抽取的得分之和。
- 现在小明想知道，怎么进行卡牌抽取，可以使得总得分最小。
- 输入格式
- 输入有两行。
- 第一行输入一个整数 n（$3 \le n \le 100$），表示一共有 n 张卡牌。
- 第二行输入 n 个整数 num_i（$1 \le num_i \le 100$），表示 n张卡牌上面的数字。
- 输出格式
- 输出一行，输出一个整数，表示卡牌抽取的最小总得分。
- 样例输入
- 5
- 20 30 5 18 3
- 样例输出
- 2520

10.2. 树形动态规划

- 树形动态规划，就是在树这一特殊的结构上维护、更新状态的最优解。
- 通常，我们从根结点出发，向子结点作深度优先搜索，并由其子结点的最优解合并得到该结点的最优解。
- 比如，我们现在需要计算子树u的大小size[u]，首先遍历其所有子结点childs[u]，再由子结点的size累加得到，即：$size[u] = \sum size[v], v \in children(u)$。
- 有些问题，我们还需再次从根结点出发，向子结点作深度优先搜索。对于树上的每个结点（除根结点以外），由父结点的信息（父结点合并后的信息，除去该孩子的信息，就是其余孩子的信息）更新该结点的信息。

例 1：POI2008 STA

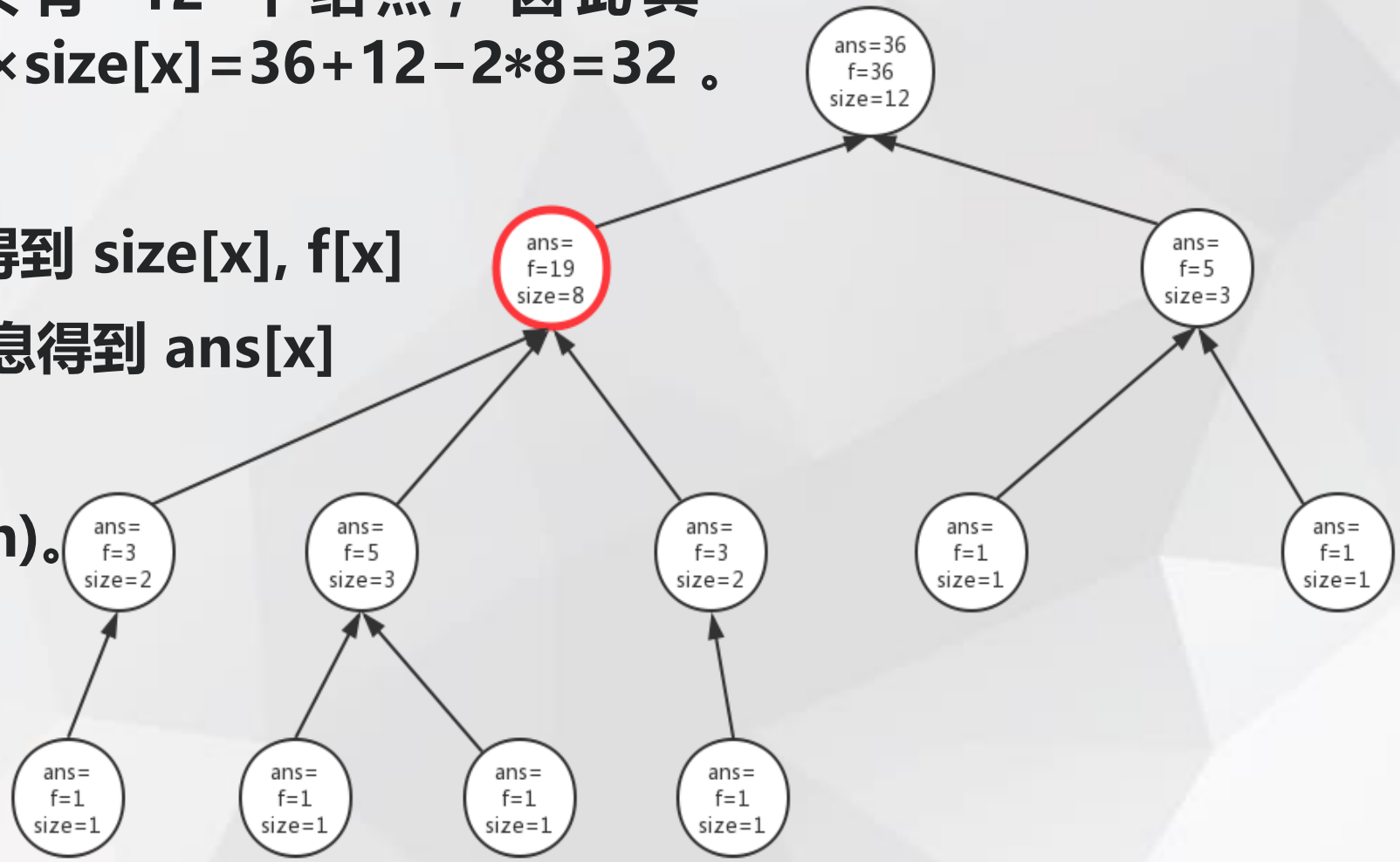
- 题目描述: 给出一个 n 个结点的树，找出一个结点为根，使得树上所有结点的深度之和最大。
- 解析: 定义：
  - fa[x]：表示 x 的父结点；
  - size[x]：表示以 x 为根的子树的结点数量；
  - f[x]：表示以 x为根的子树的所有结点的深度之和；
  - ans[x]：表示整棵树以 x 为根时所有结点的深度之和；
- 很容易想到递推式：$f[x]=\sum(f[y]+size[y])$，其中 y 是 x 的子结点，因为加上 x 以后，y 的所有子结点深度都会加 1，所以递推式里需要加上 size[y]；
- 我们用 ans[x] 表示以x为根时所有结点的深度之和，则对于根结点来说，$ans[x]=f[x]$，而对于非根结点 x 的 ans[x] 需要从父亲转移而来：$ans[x]=ans[fa[x]]+n-2\times size[x]$。

例子

- 比如这个例子，我们自上而下计算到标红结点时，其父结点的ans是 36，一共有 12 个结点，因此其ans[x]=ans[fa[x]]+n−2×size[x]=36+12−2*8=32 。
- 整个算法的流程如下：

1. dfs 一遍，由子结点信息得到 size[x], f[x]
2. 再 dfs 一遍，由父结点信息得到 ans[x]
3. 求出最大值

- 算法的整体时间复杂度 O(n)。



例 2：寻找结点集合

- 题目描述：当前存在一棵有若干结点的树，我们需要从中选取一些结点，要求最终选出的所有结点中，不存在有直接父子关系的结点对。请给出满足这个条件的包含最多结点的选择方案。
- 解析：我们可以用树形动态规划来解决这个问题。
- 定义 $dp[i][0]$ 为不选择编号为i的结点时的最优解，此时
- $dp[i][0] = \sum_{j \in childs_{[i]}} \max(dp[j][1], dp[j][0])$，其中 j 为 i的子结点。
- 定义 $dp[i][1]$ 为选择编号为i的结点时的最优解，此时
- $dp[i][1] = \sum_{j \in childs_{[i]}} dp[j][0]$，其中 j 为 i 的子结点。

算法的核心代码如下：

```
int n;  // 结点个数
int dp[maxn][2];  // dp[i][0] 表示不选择结点 i，dp[i][1]
                  //表示选择结点 i
int father[maxn];  // father 记录了结点的父结点编号
bool visited[maxn];  // 用来标记结点是否在树形 DP 过程
                     //中被访问过
void tree_dp(int node) {
    visited[node] = 1;  // 标记为已访问
    dp[node][0] = 0;
    dp[node][1] = 1;
    for(int i = 1; i <= n; i++) {
        if(!visited[i] && father[i] == node) {
                        // i 为 node 的子结点
            tree_dp(i);  // 递归计算子结点
            // 关键
            dp[node][1] += dp[i][0];
                        // 选择父结点，则必须不选择子结点
            dp[node][0] += max(dp[i][1], dp[i][0]);
                // 不选择父结点，则可以选择或不选择子结点
        }
    }
}
int main() {
    int f, c, root;
    scanf("%d", &n);
    memset(father, 0, sizeof(father));
    memset(visited, 0, sizeof(visited));
    root = 0;  // 记录树的根结点
    while (scanf( “%d %d” , &c, &f), c || f) {
        // 读入父子关系，前一个结点是后一个结点的孩子
        father[c] = f;
        root = f;
    }
    while(father[root]) {  // 查找根结点
        root = father[root];
    }
    tree_dp(root);
            // 从根结点出发进行动态规划
    printf("%d\n", max(dp[root][0], dp[root][1]));
            // 求出最终答案，根可以选或不选
    return 0;
}
```

习题：小明的建设方案

- 某国有 n座城市，编号从 1 到 n。小明做为该国的道路工程设计师，打算在各个城市之间建设 n−1 条双向通行的道路，使得任意两个城市都是可以互相到达的。假设建设每i条道路的成本为 w_i，第 i条道路的长度为 $length_i$，道路两边连接的城市数分别为 $num_a\ (i)$ 和 $num_b\ (i)$ ，则所有道路的修建总成本 C为：

$$w_i = \boldsymbol{length_i} \times |\ num_a\ (i) - num_b\ (i)\ |,$$

$$C = \sum_{i=1}^{n} w_i = \sum_{i=1}^{n} \boldsymbol{length_i} \times |\ num_a\ (i) - num_b\ (i)\ |.$$

- 现在小明设计了一个建设方案，想请你帮他计算一下，该方案的总成本 C。
- 输入格式：输入第一行是一个整数 $n\ (1 \le n \le 1{,}000{,}000)$ ，表示某国的 n座城市。接下来输入 n−1 行，每行输入三个整数 s、t、$c\ (1 \le s,t \le n,\ 1 \le c \le 1{,}000{,}000)$ ，表示城市 s 和 t 之间有一条长度为 c 的道路。
- 输出格式：输出一行，输出一个整数，表示道路建设的总成本 C。
- 样例输入
- 5
- 1 2 2
- 2 3 1
- 3 4 4
- 3 5 5
- 样例输出
- 34

习题：电网改造

- 某国有 n 座城市，编号从 1 到 n，城市间有 n−1条电网，且保证任意两座城市之间是连通的，每条电网有一个改建费 wi。现在小明要重新设计电网铺设方案，他决定在原来的基础上进行修改。
- 小明会将 n座城市分成 m个集合，编号从 1 到 m，集合之间没有交集，每座城市必须属于且只能属于其中一个集合，集合不能为空。其中，编号为 1的城市必须在编号为 1 的集合里，且该集合必须要有 k 座城市。在分配好后，如果第 i条电网两端的城市属于同一个集合，则小明改建第 i 条电网需要的改建成本为电网对应的改建费 wi；如果电网两端的城市不属于一个集合，视为电网自动拆除，不考虑其改建成本。小明改建电网的总成本，为每条电网的改建成本之和。小明想尽可能减少改建的总成本，请问该如何设计呢。
- 输入格式: 第一行输入三个整数 n(1≤n≤300)，m(2≤m≤n)，k(1≤k≤n)，表示一共有 n 座城市，需要分成 m 个集合，第 1 个集合需要有 k 座城市。接下来输入 n−1行，每行输入三个整数 a(1≤a≤n)，b(1≤b≤n)，wi(0≤wi≤10^5)，表示城市 a 和城市 b 之间存在一条改建费为 wi 的电网。
- 输出格式: 输出一行，输出一个整数，表示符合条件下，小明需要的最小改建总成本；如果设计不出符合条件的方案请输出 −1。
- 样例输入
- 5 2 3　　样例输出
- 1 2 10　　15
- 1 3 30
- 2 4 20
- 1 5 15

提示: 这是一道树上的多重背包问题。树上的背包转移问题和普通的背包转移是一样的原理。2 个提示点。
1. 状态开成 dp[n][n][2] , dp[i][j][k] 表示在城市 i及其子树中电网 1 有 j 个城市，k 表示当城市 i本身是否输入电网 1 。
2. 分 m=2 和 m>2, 2 种情况转移。m>2 的时候，只需要考虑电网 1 的费用，其他电网可以让其费用为 0, m=2的时候需要考虑电网 2 的费用。

习题：搜集钻石

- 某国有 n座城市，编号从 1到 n，城市间有 n−1 条道路，且保证任意两座城市之间是连通的。每一座城市有一定数量的钻石。小明想在该国搜集钻石。他从城市 1出发，每天他可以通过城市之间道路开车到另外的城市。当小明头第一次到一个城市的时候，他可以搜集完这个城市的所有钻石，如果他后面再来到这个城市，就没有砖石可以收集了。
- 小明只有 K天时间，请你帮算小明计算他最多可以搜集多少钻石。
- 输入格式: 第一行输入三个整数 n(1≤n≤100)，K(0≤k≤200)，表示一共有 n 座城市，小明有 K 天时间。接下里一行输入 n 个用空格隔开的整数，表示每个城市的钻石数量。每个城市的钻石数量不大于 1000。接下来输入 n−1 行，每行输入两个整数 a(1≤a≤n)，b(1≤b≤n)，表示城市 a和城市 b之间存在一条双向道路。
- 输出格式: 输出一行一个整数表示小明最大能获取的钻石数量。
- 样例输入1
- 3 2
- 3 8 3
- 1 3
- 3 2
- 样例输出1
- 14
- 样例输入2
- 6 2
- 5 9 8 4 9 2
- 1 6
- 6 2
- 2 5
- 5 3
- 5 4
- 样例输出2
- 16

提示: 也是树上的多重背包问题。状态用 dp[i][j][k], 表示在点 i用了 j 天是否回到点 i的最大数量。

10.3. 单调队列

- 单调队列，顾名思义，就是指元素单调排列（单调递增或单调递减）的队列。和普通队列的区别在于以下两点：
- • 可以删除队首或队尾元素
- • 在队尾插入元素时，为了确保队列的单调性，需要不断弹出当前的队尾元素，直到插入元素后队列仍然单调
- 例如，要往队列 q=[1,4,5] 中插入元素 3，则需要进行如下的操作：
- • 弹出队尾元素 5
- • 弹出队尾元素 4
- • 将元素 3 插入队尾

- 由于每个元素最多入队一次、出队一次，所以单调队列对于每次操作的均摊时间复杂度为 O(1)。

例题

- 给定一个长度为 n的整数序列 $a_0, a_1, \dots a_{n-1}$ 和一个滑动窗口的长度 k，需要计算如下的结果：

$$f_i = \max\{a_{i-k+1}, a_{i-k+2}, \dots, a_i\}(i = k-1, k, \dots n-1)$$

- 换句话说，就是用一个长度为 k的滑动窗口在序列上移动，计算滑动窗口内元素的最大值。
- 解法 1
- 我们可以很容易地想到一个解法，对于每个 i，固定长度为 k 的滑动窗口，然后计算 k次找到滑动窗口内的最大值。整体时间复杂度为 $O(nk)$。
- 显然，对于枚举 i 的效率很难有进一步的提升，但对于计算区间内的最大值，是否有办法可以优化计算效率呢?

解法 2

- 如何运用单调队列来高效地求解这道题呢?
- 我们可以借助一个单调递减队列来解决这道题。当顺序枚举到第 i 个元素 a_i 时，判断该元素与队尾元素的大小关系，如果队尾元素小于或等于 a_i，则弹出队尾元素，直到队列为空或队尾元素大于 a_i。
- 如何处理滑动窗口长度 k呢？我们需要在队列的元素中额外记录每个元素的原始下标。对于队首元素的下标 $front_{id}$，如果$front_{id} \leq i - k$，则说明队首元素已经不在以 i 为最后一个元素的滑动窗口中了，需要将队首元素弹出，直到队首元素的下标满足要求。
- 在将已经超出滑动窗口的元素弹出后，队首元素的值就是当前滑动窗口内的最大值。

完整的 C++ 实例代码如下：

```
#include <iostream>
using namespace std;
const int N = 10000;  // 数组长度的最大值
struct data {
    int id, value;
} q[N];
int a[N], n, k, l, r;
int main() {
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
    }
    l = 0, r = -1;  // 队首和队尾在数组 q 中的下标
    for (int i = 0; i < n; ++i) {
        while (l <= r && q[l].id <= i - k) l++;  // 不断弹出不符合要求的队首元素
        while (l <= r && q[r].value <= a[i]) r--;  // 不断弹出不符合要求的队尾元素
        q[++r] = {i, a[i]};  // 将当前元素入队
        printf("%d %d\n", i, q[l].value);  // 输出当前滑动窗口内的最大值
    }
    return 0;
}
```

习题：最重要的客户

- 小明在某银行当大堂经理，他的工作是接待客户。每当有新的客户来的时候，小明需要询问客户办理什么业务。这样不同的客户在小明心中都有不同的重要程度。
- 柜台为客户办理业务按照先后顺序来办理，也就是先来的先办理、后来的后办理。每当有客户离开或者进来的时候，小明想知道当前等待的所有客户中最重要的客户的重要程度是多少。
- 输入格式: 输入第一行用START来表示开始。
- 接下来输入若干行，按照IN val或者OUT或者END的格式输入。其中IN val表示进来一位重要程度为 val($0 \le val \le 10^9$) 的客户，OUT表示有一位客户完成业务离开。END表示输入结束。
- 保证IN和OUT总数不超过 1000000。
- 输出格式: 每次有客户进来或者离开的时候，都输出一行表示当前等待的客户中最重要的客户的重要程度。如果有非法OUT或者没有客户等待了，输出-1。

样例输入	样例输出
START	12
IN 12	16
IN 16	16
OUT	16
IN 6	6
OUT	10
IN 10	
END	

提示: 在使用 C++ 语言时，建议用scanf和printf进行输入和输出。

单调队列优化dp

- 在讲优化之前，我们先说一些概念。我们习惯用 aD/bD 的形式来表示一个线性动态规划问题规模。其中a 表示状态的维度，b表示一次转移的维度。比如 $dp[i] = \min_{1 \le j < i}\{dp[j] + 1\}$ 这种是一个 1D/1D 的规划问题。之前学习的区间动态规划是一个 2D/1D 的规划问题。1D/1D 和 2D/1D 是比赛中最常见的动态规划的规模。当然有时候也会遇到 2D/2D 甚至更高维的问题。
- 单调队列优化
- 前面我们已经学习了单调队列的使用，在这节课程我们来学习如何用单调队列来优化有一类 1D/1D 的决策类动态规划问题。
- 看下面这个问题：

Mowing the Lawn修剪草坪

- FJ 有 N(1≤N≤100000) 只排成一排的奶牛，编号为 1...N。每只奶牛的效率是不同的，奶牛 i的效率为 Ei(0≤Ei≤1000000000)。靠近的奶牛们很熟悉，因此，如果 FJ 安排超过 K(1≤K≤N) 只连续的奶牛，那么，这些奶牛就会罢工去开派对。现在 FJ 需要你帮助如何挑选奶牛，才能使她们的工作能力之和最高，而且不会罢工呢?
- 分析：我们用 dp[i] 表示前 i只奶牛的最大效率。用 sum[i] 表示前 i 只奶牛的效率之和。可以得到如下转移:

$$dp[i] = \max_{i-K \le j \le i} \{dp[j-1] + sum[i] - sum[j]\}$$

- 对于上述转移，如果直接转移复杂度是 O(NK)，对于本题来说时间复杂度还不够。再仔细分析分析，实际上对于 dp[i] 来说，我们需要找到一个决策 j(i−K≤j≤i) 使得 dp[j−1]−sum[j] 最大化。由于可选区间的左端点和右边端点都是单调增加的，所以我们实际上可以用单调队列来维护这个决策变量区间。
- 具体的，我们用一个单调队列来记录决策 j的下标，维护一个随着 j递增 dp[j−1]−sum[j] 减小的单调队列。每次转移的时候，单调队列的队首下标就是最优决策。由 i过渡到 i+1 的时候，决策区间整体向右平移 1。这时候 i−K 这个决策被踢出决策集，而 i+1 被加入决策集合。前面已经讲解了如何维护单调队列，所以这里就不再详细讲解了。给出一份代码，见下一页。

代码

```
#include <stdio.h>
typedef long long LL;
const int maxn = 100010;
LL dp[maxn], sum[maxn];
int que[maxn], E[maxn];
int head, tail;
LL max(LL a, LL b) {
    return a > b? a : b;}
void add(int j) {
    while (head < tail && dp[j - 1] - sum[j] >=
            (que[tail - 1] > 0 ? dp[que[tail - 1] - 1] :
            0) - sum[que[tail - 1]]) {
        --tail;
    }
    que[tail++] = j;
}
void del(int j) {
    if (head < tail && que[head] == j) {
        ++head;
    }
}
int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    sum[0] = 0;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &E[i]);
        sum[i] = sum[i - 1] + E[i];
    }
    dp[0] = 0;
    que[tail++] = 0;
    for (int i = 1; i <= n; ++i) {
        add(i);
        del(i - k - 1);
        int j = que[head];
        dp[i] = (j > 0 ? dp[j - 1] : 0) + sum[i] -
                        sum[j];
    }
    LL ans = max(dp[n], dp[n - 1]);
    printf("%lld\n", ans);
    return 0;
}
```

单调队列优化多重背包

- 多重背包的转移方程如下，其中 v[i],w[i], c[i] 分别代表第 i个物品的体积，价值和数量。

$$dp[i][j] = \max_{0 \le k \le c[i]} \{dp[i-1][j-k*v[i]] + k*w[i]\}$$

- 我们把背包容量按照 v[i] 的剩余分成 v[i] 组。

组数	分组
0	0, v[i], 2v[i], 3v[i]...
1	1, 1+v[i], 1+2v[i], 1+3v[i]...
...	...
v[i]-1	v[i]-1, 2v[i]-1, 3v[i]-1

- 仔细观察转移方程，会发现真正的转移只会发生在同一组内，不同的组是不可能转移的。我们改一下上面的转移方程：

$$dp[i][b+xv[i]] = \max_{0 \le k \le c[i]} \{dp[i-1][b+(x-k)v[i]] + kw[i]\}$$

- 令 t=x−k，转移变成如下

$$dp[i][b+xv[i]] = \max_{x-c[i] \le t \le x} \{dp[i-1][b+tv[i]] - tw[i]\} + xw[i]$$

- 这里的转移实际上一对 (x,b) 唯一对应一个 j。我们固定 b，那么对于 x 的决策 t 的集合正好是一个滑动区间，而 max 里面是一个和 x无关式子，所以可以用单调队列来维护。
- 具体的算法如下：

  1. 枚举一个 i 和 $b(0 \le b < v[i])$ 。
  2. 从小到大枚举 x，并且用单调队列来维护 $dp[i-1][b+tv[i]] - tw[i]$ 关于 t单调下降，更新 $dp[i][b+xv[i]]$ 的值。

- 由于每个剩余类最多只有 $\frac{V}{v[i]}$个元素，那么对于一个 i，转移的总时间是 $O(\frac{V}{v[i]}v[i]) = O(V)$，所以算法中的时间复杂度为 $O(NV)$，其中 V代表背包总容量。这样我们就优化掉了多重背包的物品个数这一维。

总结

- 一般单调队列用来优化类似于下面的转移

$$dp[i] = (\max) \min_{l[i] \le j \le r[i]} \{dp[j] + f[j]\} + g[i]$$

- 其中，l[i] 和 r[i] 是关于 i 同时递增或者同时递减的序列。对于滑动区间类的问题，实际上只是单调队列优化的一个特例，当然比赛中遇到的最多的还是滑动区间类的问题。

习题：凑钱

- 小明手上有 n 种钱币，每种钱币都有不同的数量，小明想知道他能凑出来多少小于等于 m 的钱数。
- 输入格式：第一行输入 n,m(1≤n≤100,1≤m≤100000)。
- 接下来一行输入 n个整数，表示每种钱币的面值。钱币面值不大于 100000。
- 接卸来一行输入 n个整数，表示每种钱币的个数。每种钱币个数不大于 1000。
- 输出格式：输出一行表示答案。
- 样例输入
- 3 10
- 1 2 4
- 2 1 1
- 样例输出
- 8

习题：校门外的树

- 校门外刚植了一排树，但是这些树高度不一，很影响美观。假设有 n颗树，第 i颗数的高度为 h_i cm。这些不美观度给学校带来的损失的计算方法为 $C \times \sum_{i=1}^{n-1} | hi + 1 - hi |$。
- 不过学校有止损方法，就是给树增高，但是给树增高的成本很大，给任意一颗树增高 H cm 的费用为 H^2。
- 请你帮学校计算最少的损失是多少。
- 输入格式
- 第一行输入两个整数 n(1≤n≤50000,1≤C≤100) 。
- 接下来一行输入 n个表示每颗数的高度 $(1 \leq h_i \leq 100)$。
- 输出格式
- 输出最小的损失。
- 样例输入
- 5 2
- 2 3 5 1 4
- 样例输出
- 15

习题：植树

- 植树节这天，老师组织班上的同学们去植树。
- 一共有 K 个同学，一共有 N 个植树点从左边到右排列成一排。第 i 个同学站在一个植树点 Si 上，每个同学站的植树点都不一样。第 i 位同学最多能值包含 Si 位置在内的连续的 Li 颗树，每植一棵树，第 i 位同学可以获得 Pi 点快乐值。当然也可以让某位同学不植树，并且每个植树点只能被植树一次。
- 老师应该怎么样安排植树，使得同学们的快乐值之和最大。
- 输入格式
- 输入第一行 2 个整数 N,K(1≤N≤16000,1≤K≤100,K≤N)。
- 接下来 K行，每行输入 Li,Pi,Si，其中 1≤Li≤N1 ， 1≤Pi≤10000, 1≤Si≤N。
- 保证所有 Si 都不重复。
- 输出格式
- 输出一行表示快乐值的和的最大值。
- 样例输入
- 8 4
- 3 2 2
- 3 2 3
- 3 3 5
- 1 1 7
- 样例输出
- 17

10.4. 斜率优化

- 斜率优化
- 斜率优化是针对一类 1D/1D 决策类动态规划问题，优化转移为 O(1)，能优化成 1D/0D 问题，时间复杂度从 $O(n^2)$ 优化成 $O(n)$。我们结合一些实际的问题来理解斜率优化。

[HNOI2008]玩具装箱

- P 教授要去看奥运，但是他舍不下他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。
- P 教授有编号为 1…N的 N件玩具，第 i 件玩具经过压缩后变成一维长度为 Ci。为了方便整理，P 教授要求在一个一维容器中的玩具编号是连续的。同时如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物，形式地说如果将第 i 件玩具到第 j个玩具放到一个容器中，那么容器的长度将为 $x = j - i + \sum_{k=i}^{j} C_k$ 。
- 制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 x,其制作费为 $(x - L)^2$。其中 L是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 L。但他希望费用最小。

分析：

- 用 $dp[i]$ 表示前面 i 个装箱的最小花费。我们枚举 $(j+1)-i$ 装入一个容器，那么有如下转移

$$dp[i]=\min_{0\le j<i}\{dp[j]+(\sum_{k=j+1}^{i}Ck+i-j-1-L)^2\}$$

- 如果我们用 $sum[i]$ 表示前 i 个容器的长度和，转移可以变成如下

$$dp[i]=\min_{0\le j<i}\{dp[j]+(\boldsymbol{sum[i]}-\boldsymbol{sum[j]}+i-j-1-L)^2\}$$

- 然后令$f[i]=sum[i]+i$，转移最终可以简化成下面形式

$$dp[i]=\min_{0\le j<i}\{dp[j]+(\boldsymbol{f[i]}-\boldsymbol{f[j]}-1-L)^2\}$$

- 用常规的动态规划方法来写，是 $O(n^2)$ 的时间复杂度，下面我们尝试用斜率优化将时间复杂度优化成 O(n)。
- 对于 dp[i]，我们假设有两个决策 $j1,j2(1\le j1<j2<i)$，并且决策 j2 优于 j1，我们可以得到。

$$dp[j1]+(\boldsymbol{f[i]}-\boldsymbol{f[j1]}-1-L)^2\ge dp[j2]+(\boldsymbol{f[i]}-\boldsymbol{f[j2]}-1-L)^2$$

- 将平方项目展开得到

$$dp[j1]+f[i]^2-2f[i](f[j1]+1+L)+(f[j1]+1+L)^2\ge$$
$$dp[j2]+f[i]^2-2f[i](f[j2]+1+L)+(f[j2]+1+L)^2$$

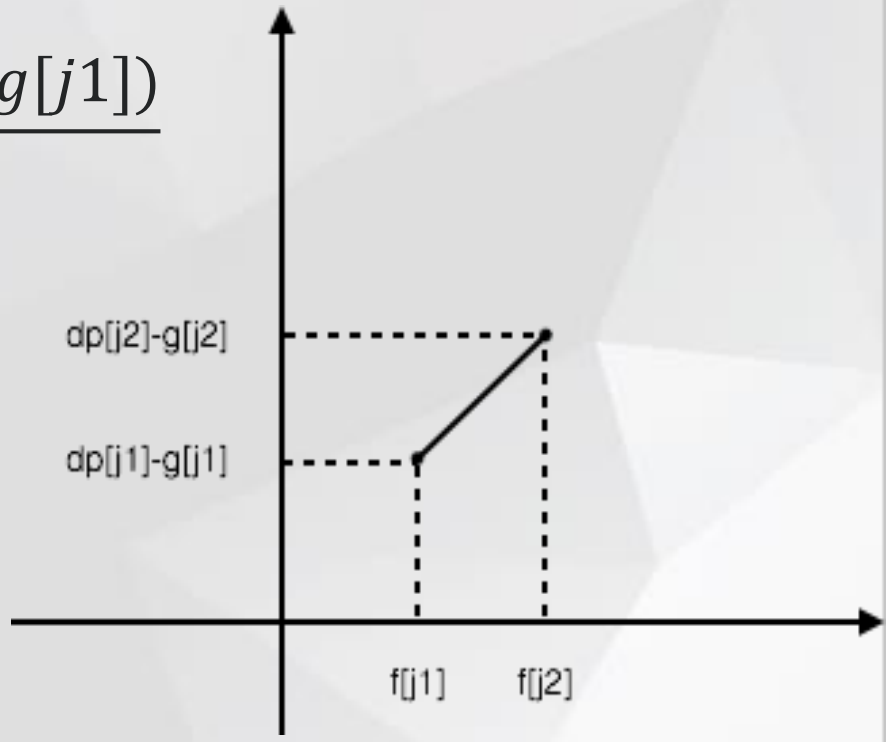
继续化简

$$2f[i](f[j2]+1+L)-2f[i](f[j1]+1+L)\geq$$
$$dp[j2]+(f[j2]+1+L)^2-(dp[j1]+(f[j1]+1+L)^2)$$
$$2f[i]\geq\frac{dp[j2]+(f[j2]+1+L)^2-(dp[j1]+(f[j1]+1+L)^2)}{f[j2]-f[j1]}$$

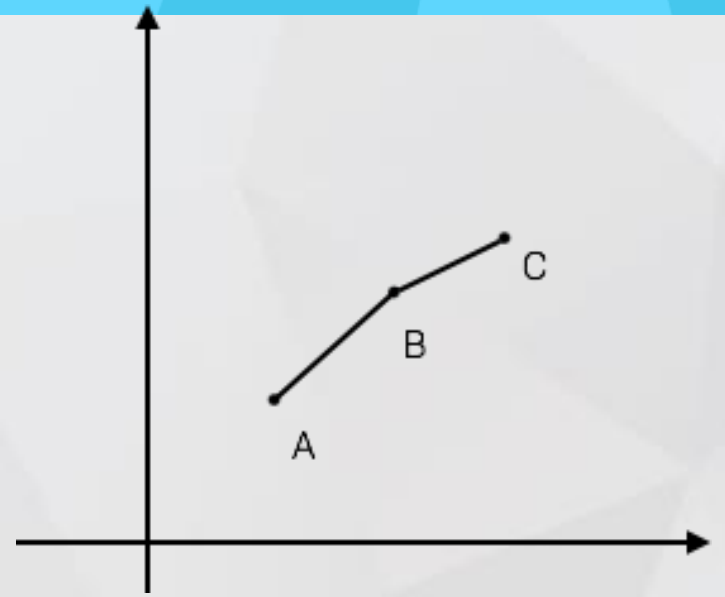
- 然后我们令 $g[i]=(f[i]+L+1)^2$，可以得到

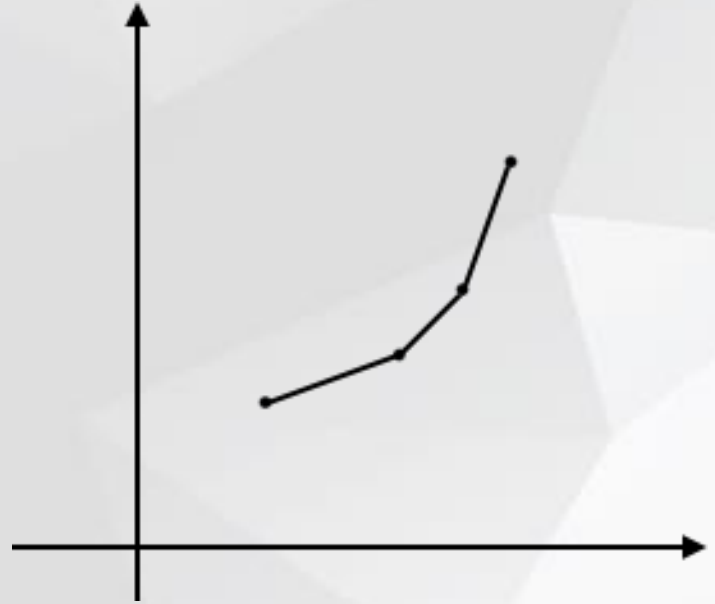
$$2f[i]\geq\frac{dp[j2]+g[j2]-(dp[j1]+g[j1])}{f[j2]-f[j1]}$$

- 也就是说如果 $j1,j2$ 满足上面的不等式，那么决策 $j2$ 永远都会优于 $j1$。如果我们把 $f[x]$ 看成横坐标，$dp[x]+g[x]$ 看成纵坐标，实际上面的不等式右侧是可以用斜率表示。如下图，我们可以用斜率表示不等式右侧，也就是当这个斜率小于等于$2f[i]$的时候，决策$j2$优于$j1$。



- 得到这个神奇的结论以后，我们就能用这个结论来搞事情了。如右上图所示的图形中 A,B,C 三个点分别对应 3 个决策。可以证明决策点 B 永远不可能成为最优的决策点。如果 B 点能作为最优决策点，那么 B 优于 A，AB 的斜率小于 2f[i]，BC 的斜率是小于 AB 从而小于 2f[i]，所以必然有决策点 C 优于 B。所以 B 点不可能是最优决策点。那么 B 点可以直接从决策点中剔除。
- 所以我们需要维护的是一个斜率单调上升的点集作为决策点，实际上对应一个下凸包，如右下图。
- 对于 dp[i]，我们去寻找其最优决策点，由于斜率单调上升，那么我们只需要找到斜率小于 2f[i] 的分界点就是最优决策点。自己想想为什么？由于是单调，可以直接用二分查找，这样一次转移的复杂度是 O(lgn)。
- 实际上这个 lgn 我们完全可以优化掉，由于 f[i] 是关于i 递增的，我们按照 i 从小到大的顺序转移的时候，二分的这个斜率的分界点必然是不减的，我们可以像单调队列那样，对于当前的 i， 前面的斜率小于 2f[i] 的点都可以抛弃掉，这样下来，时间复杂度是 O(n)，因为最多只有 n个点，每个点最多进一次集合，最多出一次集合。





总结

- 如果题目是求最大值，实际上就需要维护一个上凸包。关于斜率优化的题目，主要就是推导。假设两个决策一个优于另外一个，把关于 i的项全部移动到一边，然后得到一个和 i无关的斜率表达式。如果转移能表示成 $dp[i] = \max\{a[i]b[j] + c[j]\} + d[i]$的形式，其中 $a[i]$ 是关于 i 递增的，都可以用斜率优化。
- 上面的问题由于 f[i]，g[i] 关于 i 是递增的，所以我们维护的凸包的斜率都是正的，我们只需要维护 $\frac{1}{4}$凸包。有些时候，斜率可能是负的，这样我们就需要维护整个上凸包或者下凸包，一般都是借助 splay tree 来维护。当然这是比较高层次的内容了，本课程内不会涉及，想深入的同学可以自行学习（前提是先把本课程内的内容先学好）。

习题：仓库建设

- L 公司有 N个工厂，由高到底分布在一座山上。工厂 1 在山顶，工厂 N 在山脚。由于这座山处于高原内陆地区（干燥少雨），L 公司一般把产品直接堆放在露天，以节省费用。突然有一天，L 公司的总裁 L 先生接到气象部门的电话，被告知三天之后将有一场暴雨，于是 L 先生决定紧急在某些工厂建立一些仓库以免产品被淋坏。
- 由于地形的不同，在不同工厂建立仓库的费用可能是不同的。第 i个工厂目前已有成品 Pi 件，在第 i个工厂位置建立仓库的费用是 Ci。对于没有建立仓库的工厂，其产品应被运往其他的仓库进行储藏，而由于 L 公司产品的对外销售处设置在山脚的工厂 N，故产品只能往山下运（即只能运往编号更大的工厂的仓库），当然运送产品也是需要费用的，假设一件产品运送 1个单位距离的费用是 1。假设建立的仓库容量都都是足够大的，可以容下所有的产品。
- 你将得到以下数据：工厂 i 距离工厂 1 的距离 Xi（其中 $X1 = 0$）；工厂 i目前已有成品数量 Pi；在工厂 i 建立仓库的费用 Ci。请你帮助 L 公司寻找一个仓库建设的方案，使得总的费用（建造费用+运输费用）最小。
- 输入格式：第一行包含一个整数 N(1≤N≤100000)，表示工厂的个数。接下里 N行每行包含两个整数 Xi, Pi, Ci，意义如题中所述。所有输入数据都小于 1000000。
- 输出格式：输出最优方案的费用。

样例输入	样例输出
3 0 5 10 5 3 100 9 6 10	32

习题：玩具装箱

- P 教授要去看奥运，但是他舍不下他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。
- P 教授有编号为 1… … N的 N件玩具，第 i 件玩具经过压缩后变成一维长度为 C_i。为了方便整理，P 教授要求在一个一维容器中的玩具编号是连续的。同时如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物，形式地说如果将第 i件玩具到第j个玩具放到一个容器中，那么容器的长度将为 $x = j - i + \sum_{k=i}^{j} C_k$。
- 制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 x,其制作费为$(x - L)^2$。其中 L 是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 L。但他希望费用最小。
- 输入格式：第一行输入两个整数 N，L。接下来一行输入C_i。$1 \le N \le 50000, 1 \le L, C_i \le 10^7$。
- 输出格式：输出最小费用。
- 样例输入
- 5 4
- 3 4 2 1 4
- 样例输出
- 1

习题：电磁屏蔽

- 有 n台超电磁炮排列成一行，第 i 台超电磁炮的功率为 w_i。众所周知，超电磁炮之间存在电磁干扰。他们之间的干扰程度为 $\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_i \cdot w_j$。
- 现在有 m 个电磁屏蔽装置，一个电磁屏蔽装置可以放置到两个相邻的超电磁炮中间，这样就可以阻止装置左边和右边的超电磁炮之间发生项目干扰。通过 m 个电磁屏蔽装置可以把超电磁炮个隔离成 m+1 个干扰组。
- 帮忙编程计算最小的干扰程度。
- 输入格式：第一行输入两个整数 n(1≤n≤1000)，m(0≤m<n)。
- 接下来一行输入 w_i (1≤w_i≤100)。
- 输出格式: 输出最小干扰程度。
- 样例输入1
- 4 1
- 4 5 1 2
- 样例输出1
- 17
- 样例输入2
- 4 2
- 4 5 1 2
- 样例输出2
- 2

提示: 转移第二维可以用斜率优化。当然这题还可以用后面提到的四边形不等式优化来优化。

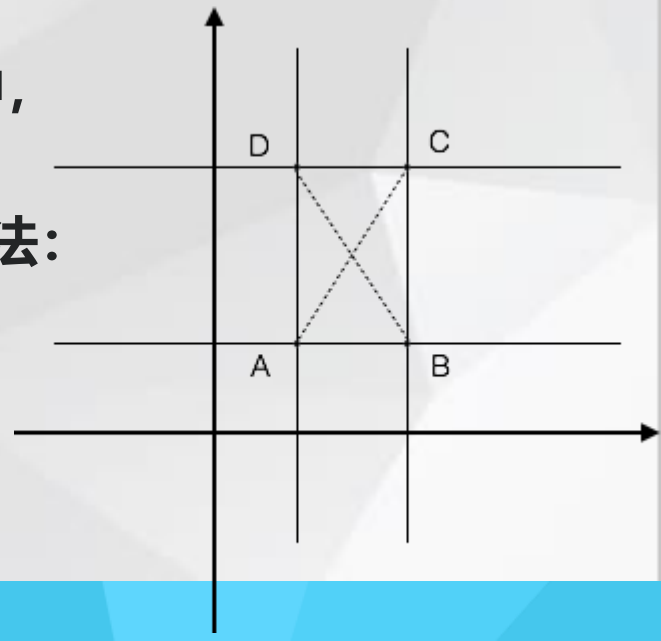
10.5. 四边形不等式优化

- 四边形不等式
- 对于一类常见的（2D/1D）的决策类动态规划问题，有一个常见的状态转移方程（不同的问题，边界条件可能不一样，但是对问题的讨论没有影响）：

$$dp(i,j)=\begin{cases}\min_{i\le k\le j}\{dp(i,k-1)+dp(k,j)+w(i,j)\}, & i<j\\ 0, & i=j\\ \infty, & i>j\end{cases}$$

- 实际上上述转移是一个典型的区间动态规划问题。 对于任意 $i\le i'<j\le j'$，如果满足不等式 $w(i,j)+w(i',j')\le w(i',j)+w(i,j')$，那么我们称函数 w 满足四边形不等式。

- 之所以称为四边形不等式，可以用如下图来形象化理解。四边形 ABCD 中，对角线 AC 端点的权值之和不大于对角线 BD端点的权值之和。
- 为了判断 w(i,j)w(i,j)w(i,j) 是否满足四边形不等式。有几个简单的方法方法：
  1. 证明 $w(i,j)+w(i+1,j+1)\le w(i+1,j)+w(i,j+1)$
- 2. 固定一个 j 算出 w(i,j+1)−w(i,j)w(i,j+1)-w(i,j)w(i,j+1)−w(i,j) 关于 i 的表达式，如果是关于 i递增，那么 w(i,j)w(i,j)w(i,j) 满足四边形不等式。同样也可以固定 i。



四边形不等式性质

1. 如果函数 w 满足四边形不等式，那么函数 dp也满足四边形不等式。即对于任意 $i \le i' < j \le j'$，满足

$$dp(i,j) + dp(i',j') \le dp(i',j) + dp(i,j')$$

- 证明略。

1. 我们定义 $s(i,j)$ 为函数 $dp(i,j)$ 对应的决策变量的最大值。所谓决策变量就是令 $dp(i,j)$ 最大的转移 k。即

$$s(i,j) = \max_{i \le k \le j} \{ k \mid dp(i,j) = w(i,j) + dp(i,k-1) + dp(k,j) \}$$

   假如 $dp(i,j)$ 满足四边形不等式，那么 $s(i,j)$ 单调，即 $s(i,j) \le s(i,j+1) \le s(i+1,j+1)$。

- 证明略。

优化

- 有了这个结论以后，状态转移方程等价于

$$dp(i,j)=\begin{cases}\min_{s(i,j-1)\le k\le s(i+1,j)}\{dp(i,k-1)+dp(k,j)+w(i,j)\}, & i<j\\ 0, & i=j\\ \infty, & i>j\end{cases}$$

- 由于动态规划的阶段是 l=j−i，所以在求 dp(i,j) 的时候，s(i,j−1) 和 s(i+1,j) 必然已经求出了。通过这样缩小决策变量的枚举范围，从而进行优化。可以证明，上面的转移的时间复杂度是 $O(n^2)$ 的。

$$\begin{aligned}&\mathcal{O}(\sum_{l=2}^{n}\sum_{i=1}^{n+1-l}(1+s(i+1,i+1-1)-s(i,i+l-2)))\\&=\mathcal{O}(\sum_{l=2}^{n}(n+1-l+s(n+2-l,n)-s(1,l-1)))\\&\le\mathcal{O}(\sum_{l=2}^{n}(2n+1-2l))=\mathcal{O}((n-1)^2)\end{aligned}$$

习题：最优二分检索树

- 给出 n个数据 $a_1 \le a_2 \le a_3 \le \cdots \le a_n$，第 i个点有一个权值 f_i。
- 用这 n 个数据构建一个二分检索树。二分检索树是一颗二叉树，任意节点满足说有左边子节点关键字小于等于当前节点，所有右子节点关键字大于等于当前节点。
- 构建这颗树的代价为所有节点的权值 f_i 乘上深度 d_i的和。即 $\sum_{i=1}^{n} f_i \cdot d_i$。根节点深度为 1。请你计算构建树的最小代价是多少。
- 输入格式
- 输入第一行一个整数 n(1≤n≤2000)。
- 接下里一行输入 f_i (1≤f_i≤100)。
- 输出格式
- 输出构建树的最小的代价。
- 样例输入
- 5
- 3 2 2 5 2
- 样例输出
- 28

习题：集合分割

- 如果 T 是一个整数集合，另 MIN 为 T 中最小的元素，MAX 为 T 中最大的元素，定义 $cost(T) = (MAX - MIN)^2$。
- 现在有一个集合 S，我们要找到 M 个 S的子集$S_1, S_2, S_3 \cdots S_m$，使得$S_1 \cup S_2 \cup S_3 \cdots S_m = S$，同时，使得 $\sum_{i=1}^{m} cost(S_i)$最小。
- 输入格式
- 输入第一行两个整数 n(1≤n≤2000)，m(1≤m≤min(n,1000))。
- 接下里一行输入 n 个 10^6以内的整数。
- 输出格式
- 输出答案。
- 样例输入
- 4 2
- 4 7 10 1
- 样例输出
- 18

习题：邮局选址

- 有 n个坐标不同的村庄排列在一条直线上。需要重从其中选择 m 个来建邮局。每个村庄使用离它最近的邮局。
- 该如何选择邮局，使得各村庄到其最近的邮局的距离总和最小。
- 输入格式
- 输入第一行两个整数 n(1≤n≤2000)，m(1≤m≤min(n,1000)) 。
- 接下里一行输入 n个村庄的坐标 Xi(1≤Xi≤10000)。
- 输出格式
- 输出最小的距离总和。
- 样例输入
- 10 5
- 1 2 3 6 7 9 11 22 44 50
- 样例输出
- 9