引入 —— 一道例题

时间限制：2000MS　　内存限制：768KB

给出一个 $P \times T$ 的棋盘，棋盘每个格子都有一些金币，要从棋盘左上角走到右下角，每次只能向右或向下走，每走到一个格子都会把格子里的金币都拿走，最多能够拿到多少金币？要求输出方案！

对于30%的数据，P、$T \leq 100$。

对于100%的数据，P、$T \leq 5000$

首先对于 30% 的数据，$n \leq 100$ ，直接 DP 空间大小是可以承受的，$f_{i,j} = min\{f_{i-1,j} + f_{i,j-1}\} + a_{i,j}$ ，然后记录路径即可。

注意到 $f_{i,j}$ 只与 $f_{i-1,j}$ 和 $f_{i,j-1}$ 有关，即只与它的**左方**和**上方**有关

那么就可以使用类似于完全背包顺序枚举的方法开一个**滚动数组**，对于每一行 i ，$f_j = min\{f_{j-1}, f_j\} + a_{i,j}$ ，这里引用的 f_j 就相当于原来的 $f_{i-1,j}$

这样这个 DP 的时间复杂度仍是 $O(NM)$，但是空间复杂度就只有 $O(M)$ 级别，数组就可以开得下了！

但是问题又随之而来，要怎么求出最短路径？？？

原本 DP 记录路径用的是记录前驱的方法，即从 $path_{N,M}$ 往回推到 $path_{1,1}$ 。但是这种做法在滚动数组下是显然不可行的。滚动数组只能记录当前一行的信息，前面行的信息被覆盖了，也就无法通过记录前驱的方法往回推。

有一种方法是对于每一行都倒序重新做一遍 DP 得到这一行的信息，这样就可以通过这一行的信息往回推到上一行。但是不幸的是，这样的时间复杂度是 $O(N^2M)$ 会 TLE 。

那还有什么方法，可以做到不 MLE 又不 TLE ？？？

答案就是——分治。

我们可以把动态规划的数组 $f_{i,j}$ 分成两部分，从起点和终点都分别向中间进行 DP 找出最小值，最后在中间寻找最小的一个点使左右（上下）两部分的和最小，对它继续进行转移。

就是说把 $(x_1, y_1) \sim (x_2, y_2)$ 这个矩阵分成两个子矩阵 $(x_1, y_1) \sim (\lfloor \frac{x_1+x_2}{2} \rfloor, j)$ 和 $(\lfloor \frac{x_1+x_2}{2} \rfloor + 1, j) \sim (x_2, y_2)$ ，j 是使得这两个子矩阵分别正向DP的 f 和反向DP的 f' 使得 $f_j + f'_j$ 最大的列编号。

因为正向DP表示从第一个子矩阵左上方到右下方（第 j 列）的最大金币数，反向DP表示第二个子矩阵从右下方到左上方（第 j 列）的最大金币数，它们在第 j 列可以进行合并，合并成了一条从当前矩阵的左上方到右下方的**完整路径**，也就代表了从 j 列走能获得的**最大金币数**。这样就符合**分治**的思想——把当前这个矩阵分割成两个子矩阵，分别处理后再合并。

话不多说，来看一个例子就明白了。

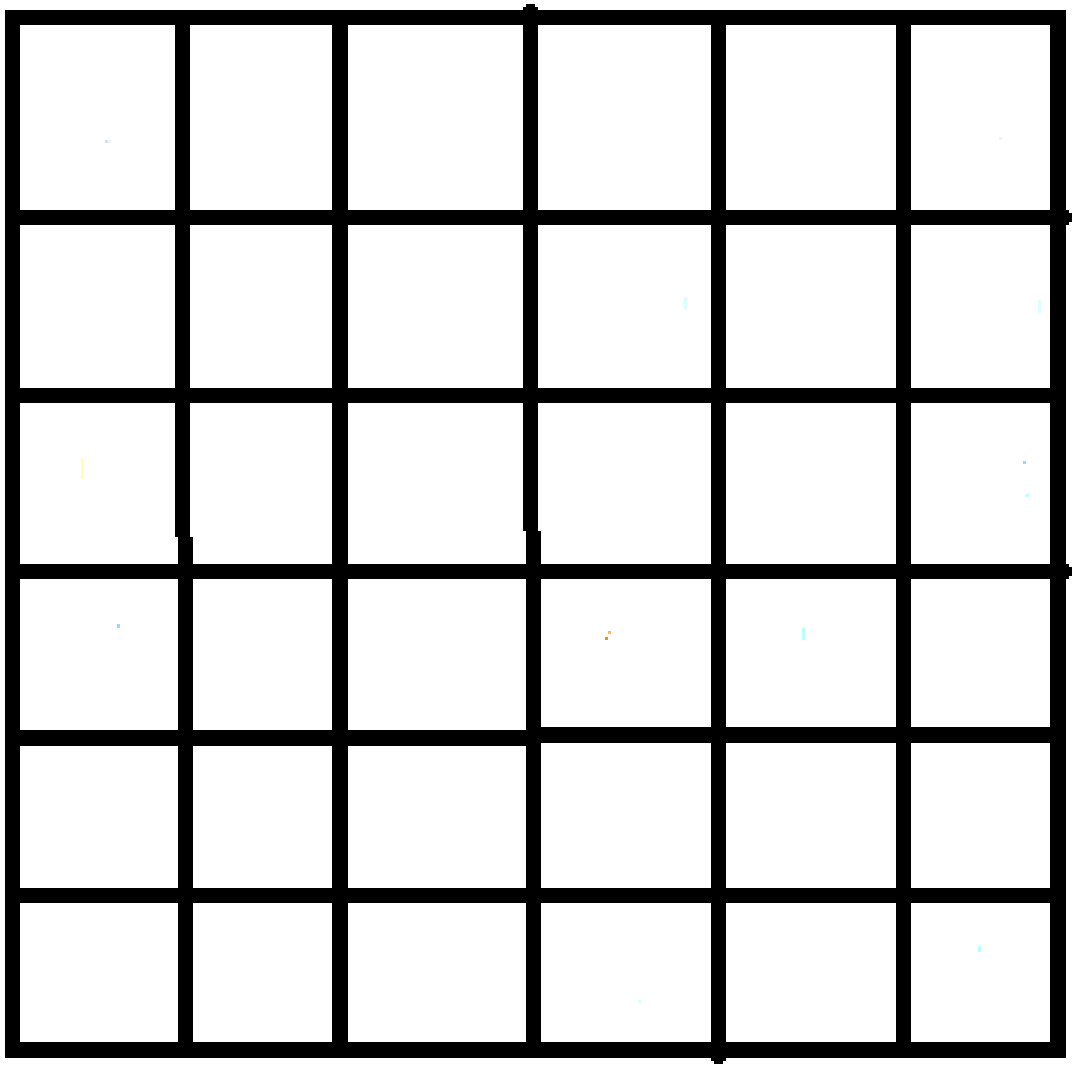
举个例子，假设有原数组 $\begin{pmatrix} 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 1 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 2 & 1 & 2 & 2 \\ 2 & 2 & 2 & 1 & 1 & 1 \end{pmatrix}$，那么它的正向转移数组就是 $\begin{pmatrix} 1 & 3 & 5 & 7 & 9 & 11 \\ 2 & 3 & 5 & 7 & 9 & 11 \\ 4 & 4 & 6 & 8 & 10 & 12 \\ 6 & 5 & 6 & 7 & 9 & 11 \\ 8 & 7 & 8 & 8 & 10 & 12 \\ 10 & 9 & 10 & 9 & 10 & 11 \end{pmatrix}$，反向转移数组就是 $\begin{pmatrix} 11 & 11 & 12 & 11 & 12 & 11 \\ 10 & 9 & 10 & 9 & 10 & 9 \\ 10 & 8 & 8 & 7 & 8 & 7 \\ 9 & 7 & 6 & 5 & 6 & 5 \\ 10 & 8 & 6 & 4 & 4 & 3 \\ 9 & 7 & 5 & 3 & 2 & 1 \end{pmatrix}$。

把它用丑的不行的画图画出来

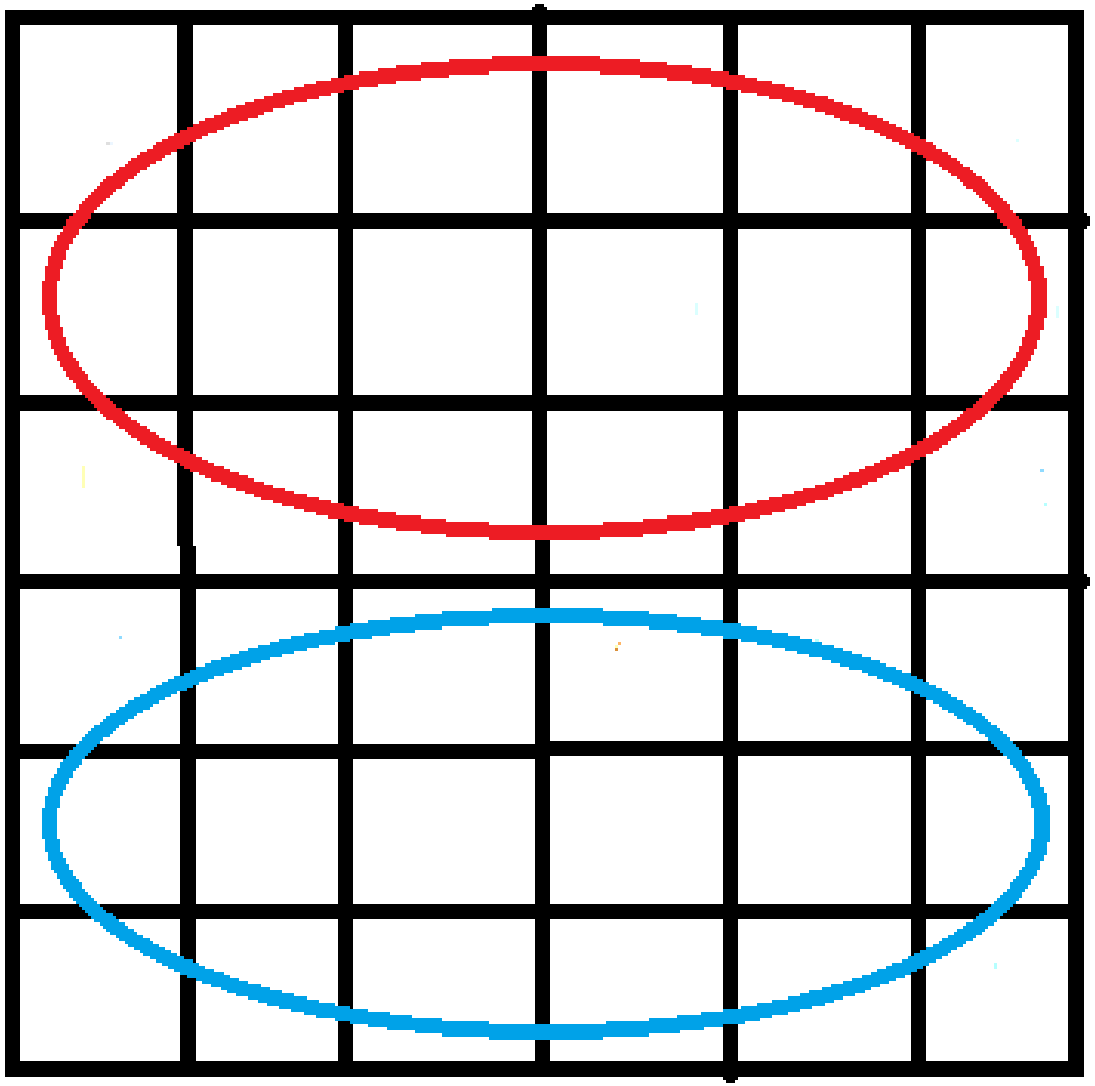
这是原数组

1	2	2	2	2	2
1	1	2	2	2	2
2	1	2	2	2	2
2	1	1	1	2	2
2	2	2	1	2	2
2	2	2	1	1	1

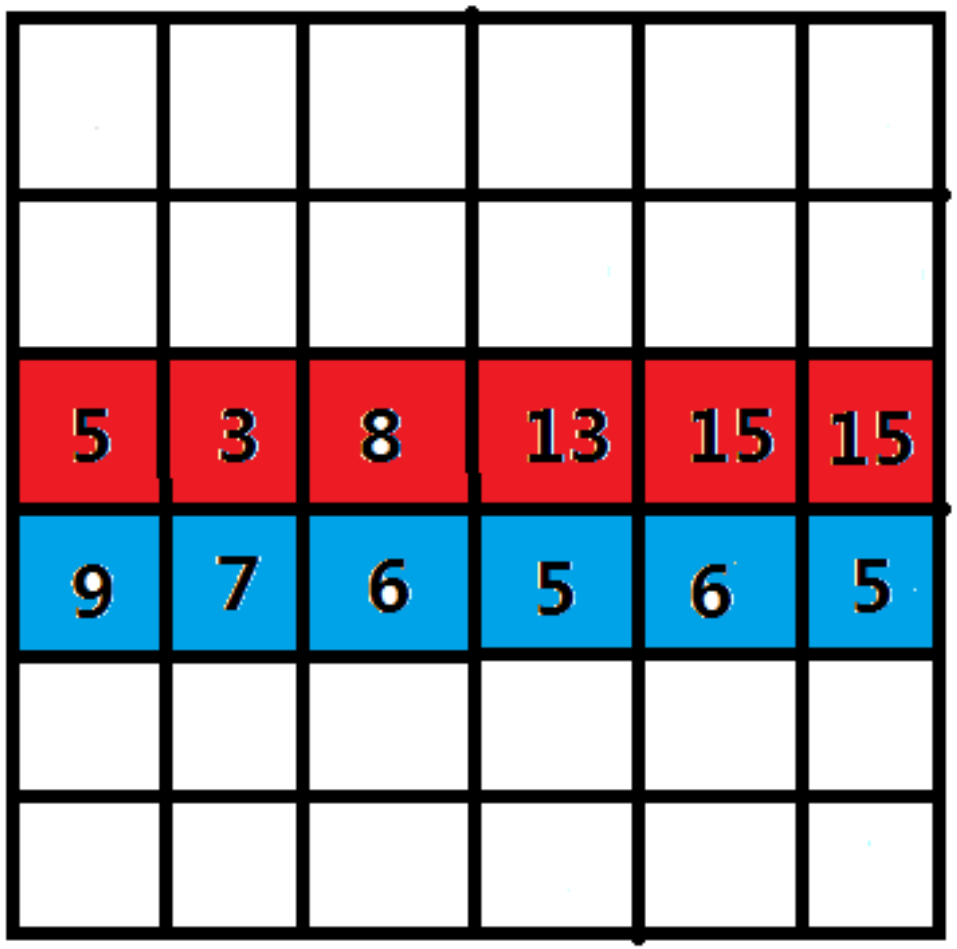
下面用它来表示矩阵的分割和合并



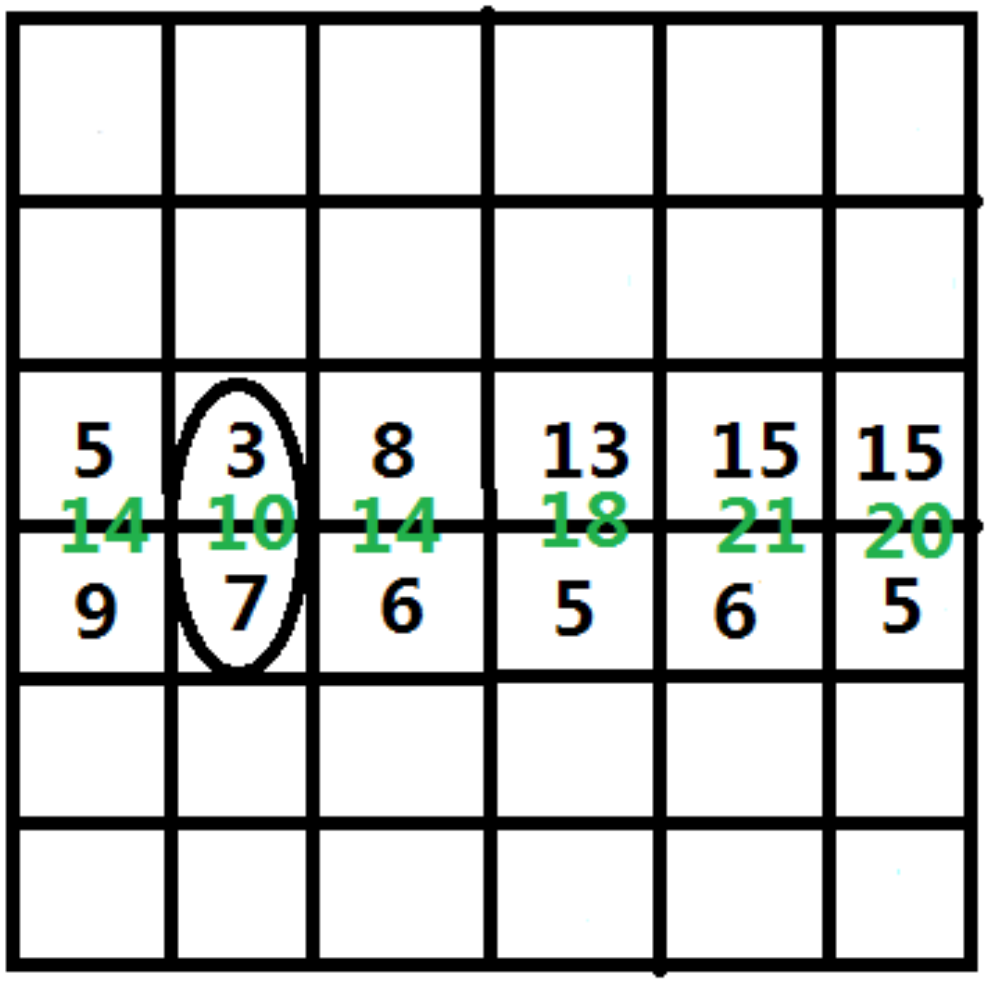
首先很简单，把它分成上下两个子矩阵。



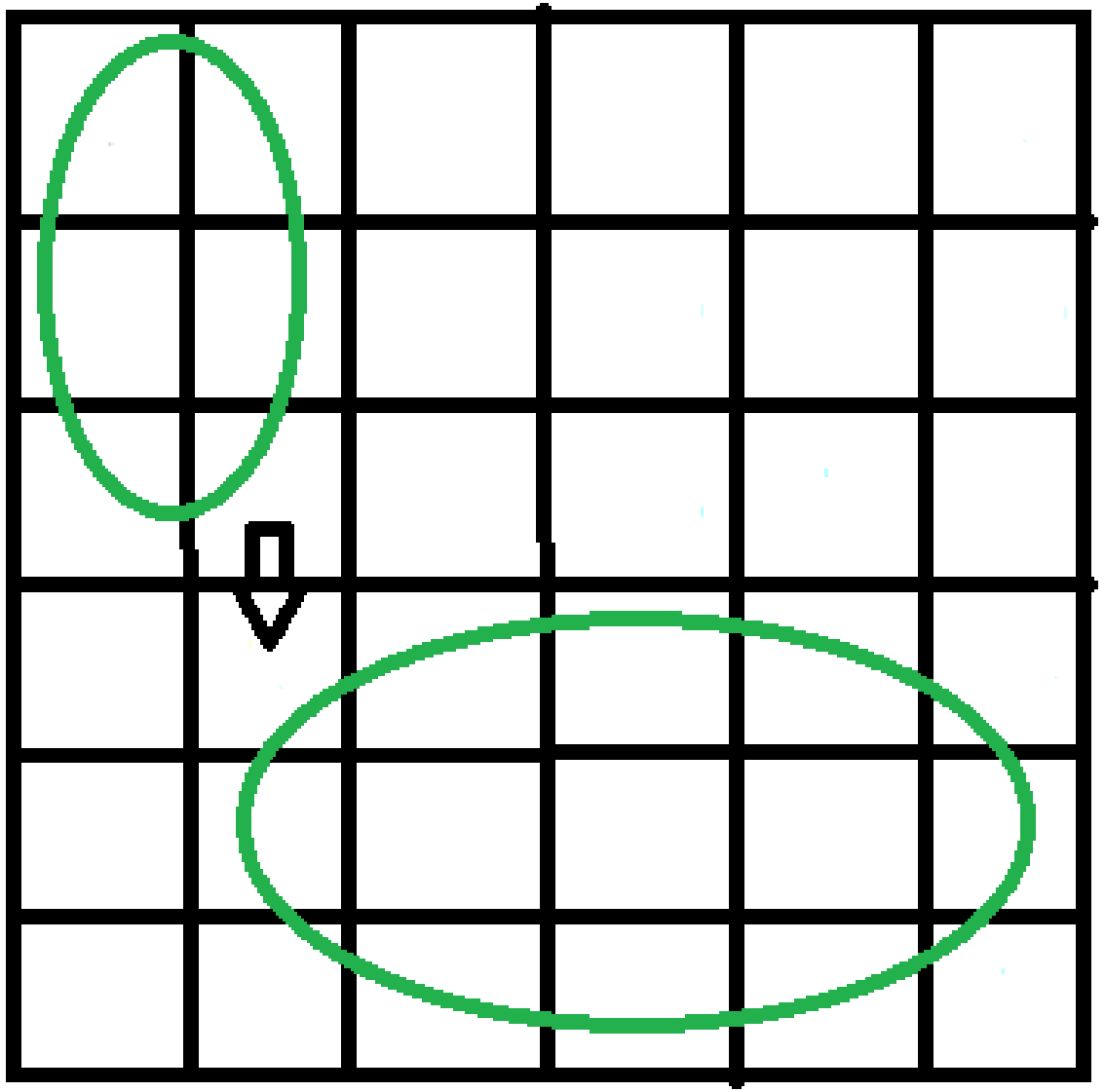
然后对这两个子矩阵分别做一遍 DP ，找到最小值。 注意，因为空间的限制，这个 DP 只能求出最终状态的那一行。在这里，对于上面的部分从起点 $(1,1)$ 开始朝中间点 $(3,6)$ 做 DP ，对于下面的部分从终点 $(6,6)$ 开始朝中间点 $(4,1)$ 反向做 DP ，这样就可以得到第 3 行和第 4 行的状态，也就是下图中涂色的部分。



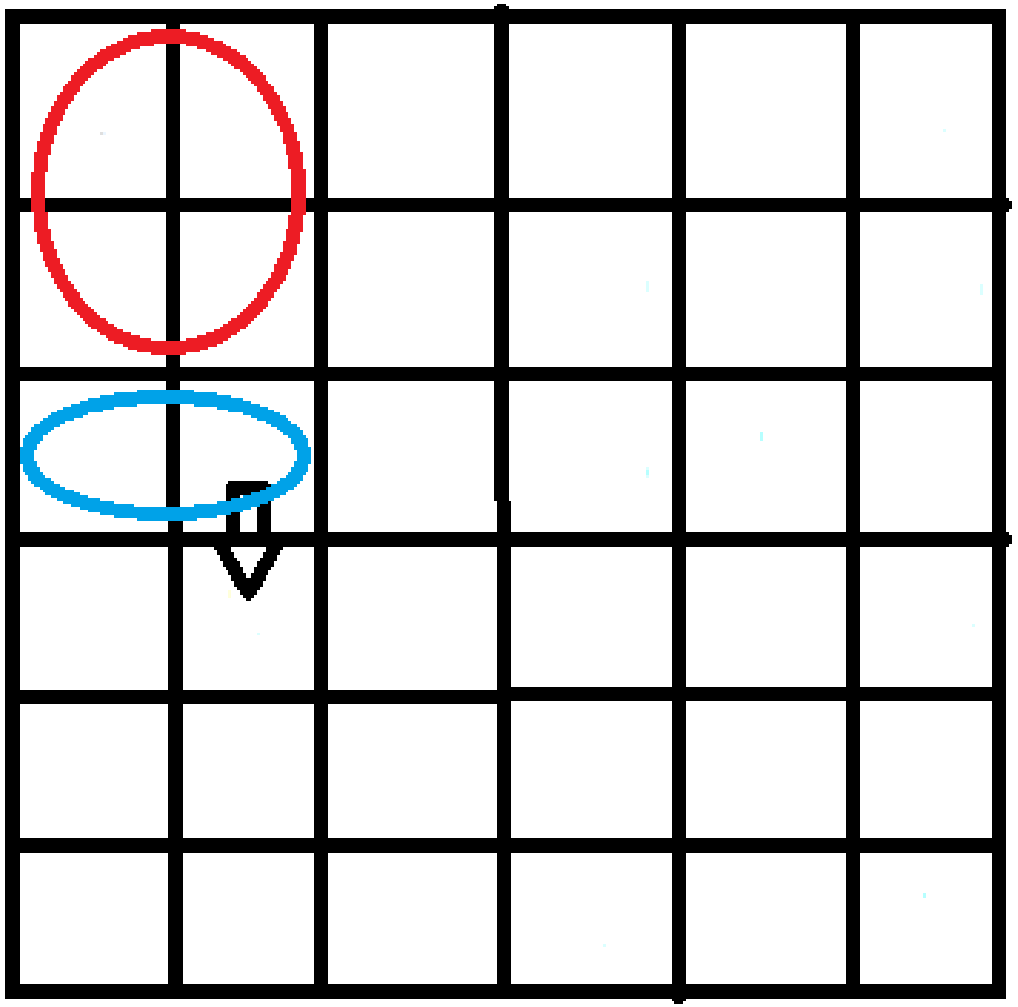
有了这两行的状态，其实已经可以合并一条路径了。从起点到中间点，再从中间点到终点，把它们合起来其实就是一条完整的路径。所以再寻找第 3 行和第 4 行的第 j 列的元素，使这上下两部分的和最小，如下图。这样就找到了一条最短的路径，且它的中间点也知道了。



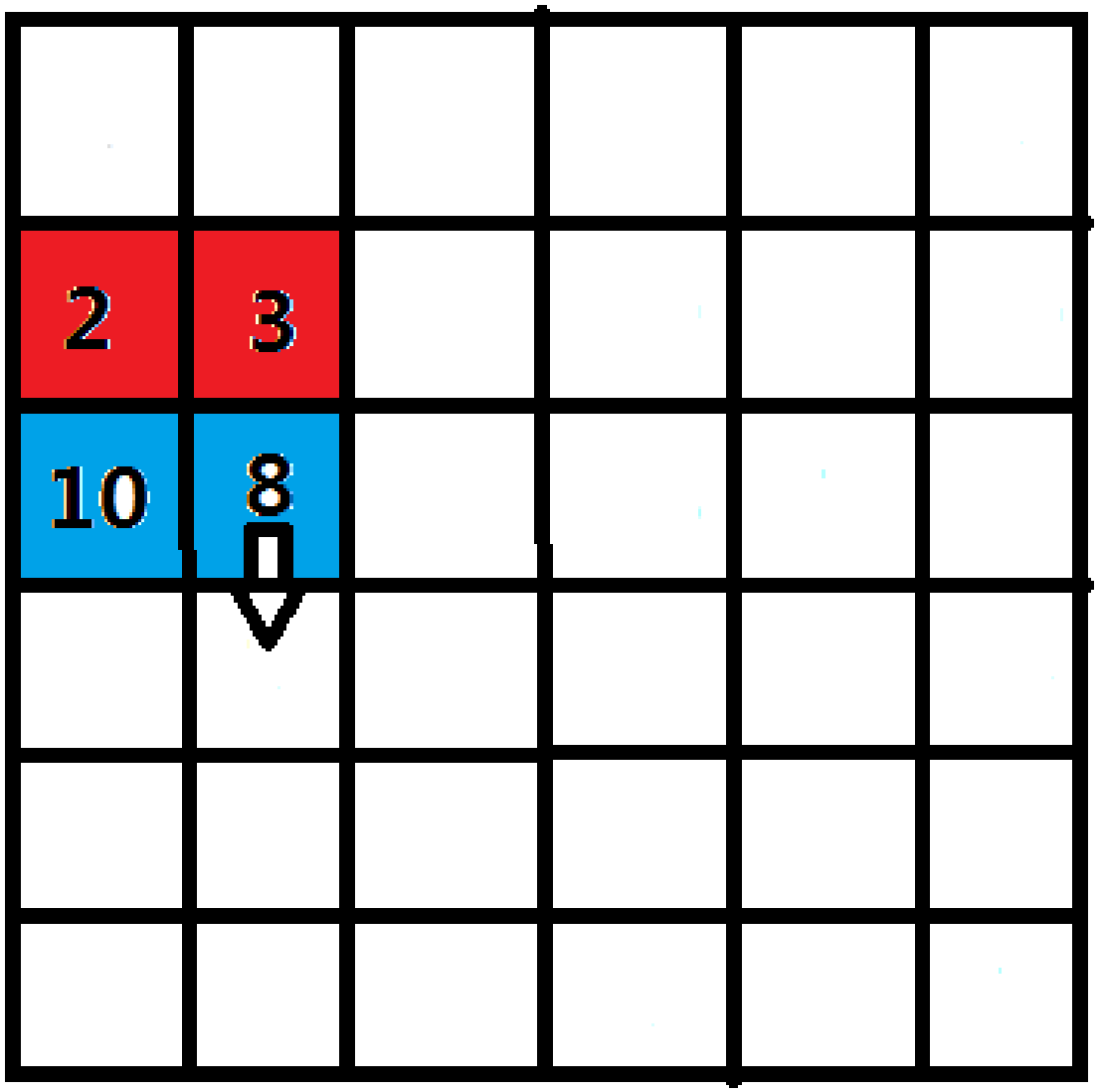
接着就又可以把它分成两个子子矩阵，再在其中分别进行同样的操作。



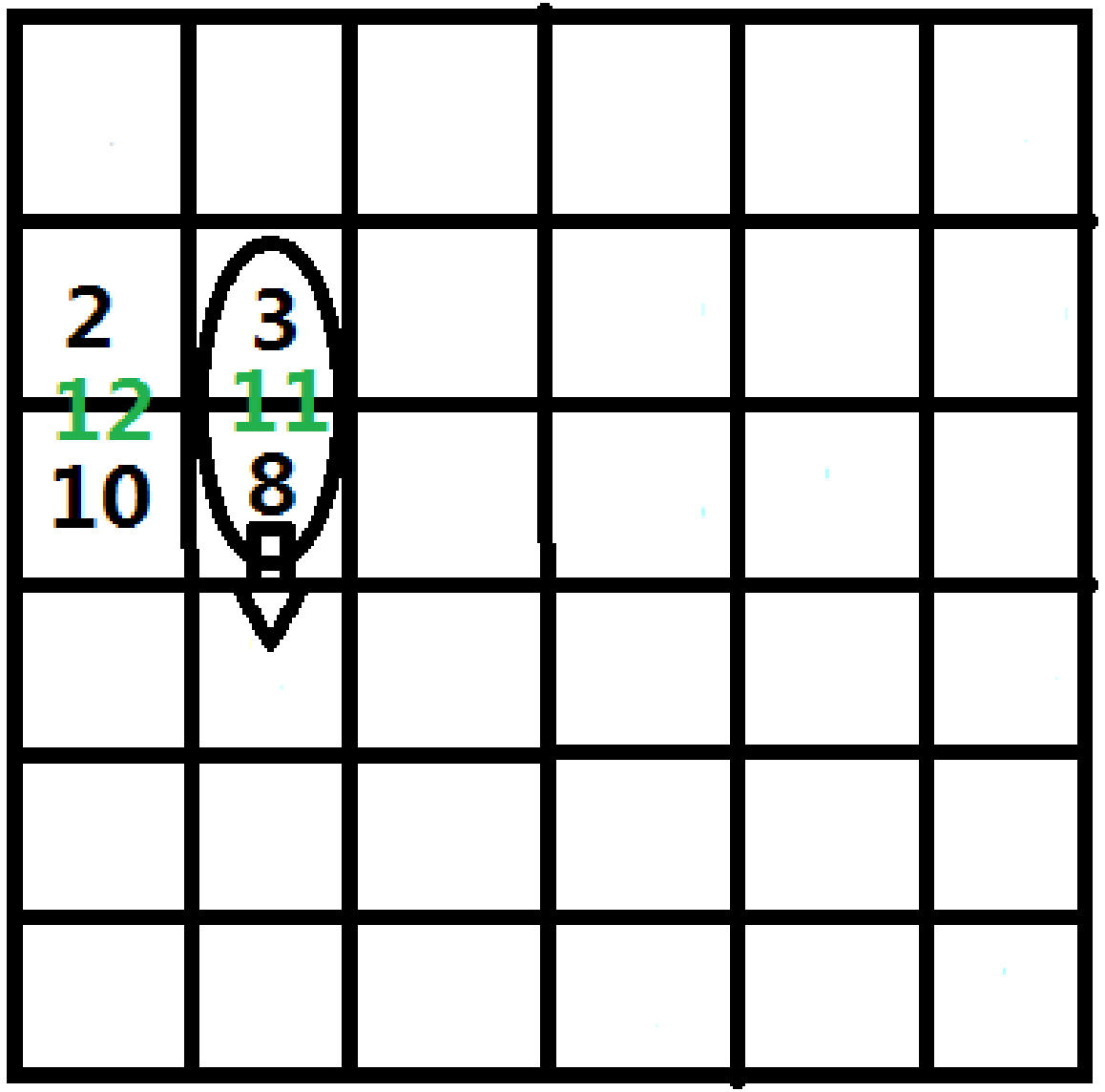
这里拿上面的子子矩阵作为例子。上面的那个子子矩阵可以再分成上下两个子子子矩阵。



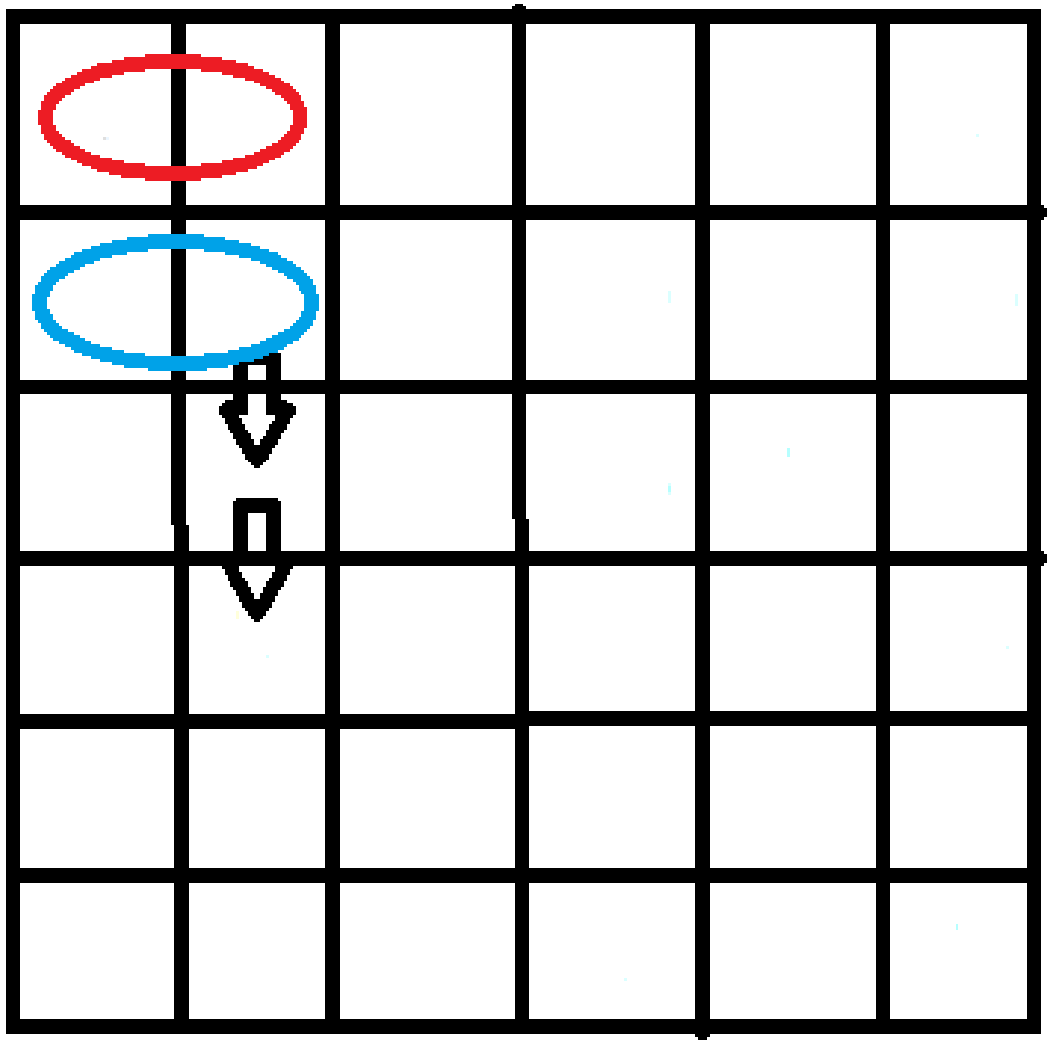
同样 DP 求出中间行的状态。



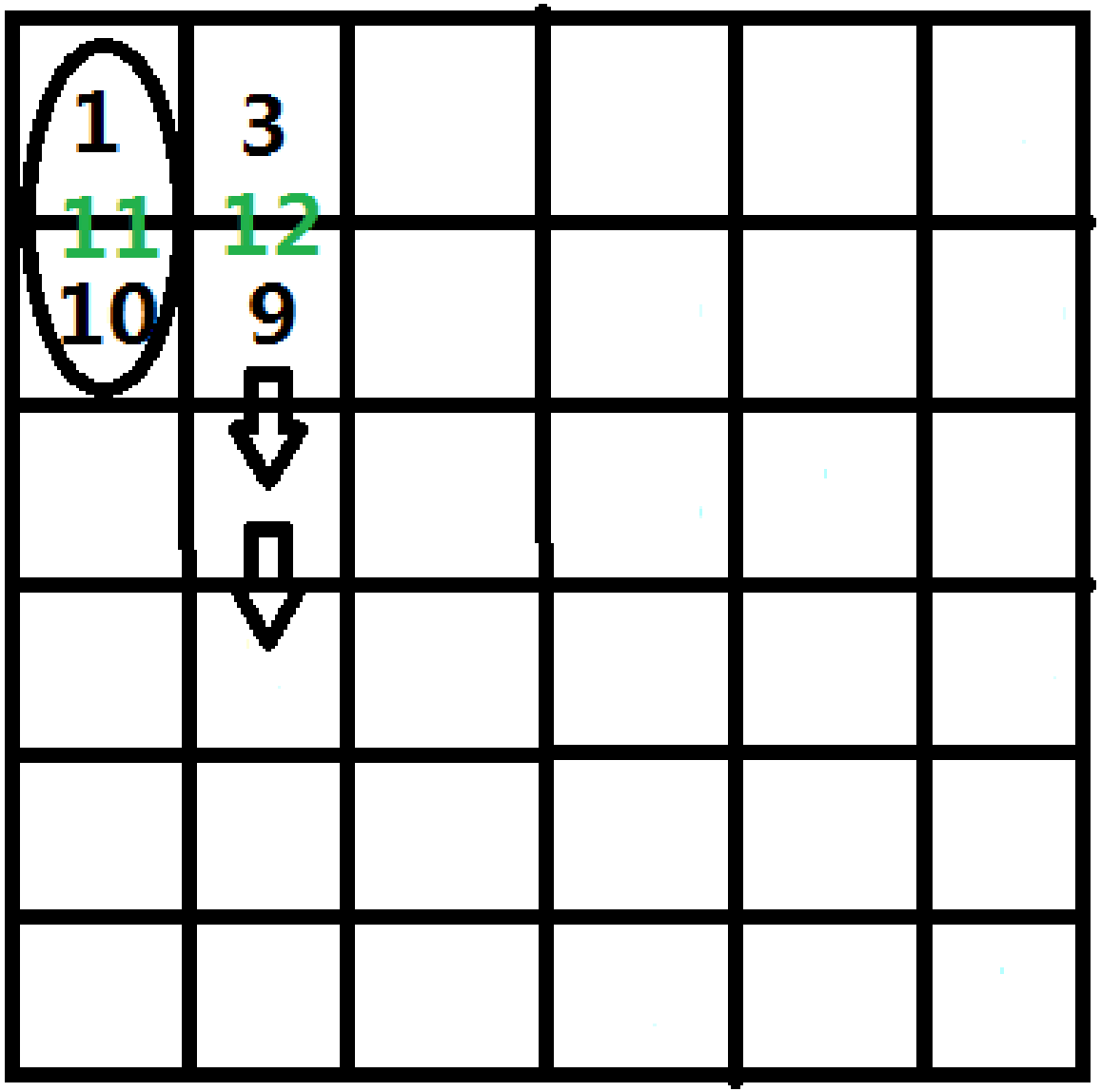
选择中间点使上下两个部分的和最小。



发现底下那部分已经处理完了，只要处理上面的部分。同样再分成上下两部分。

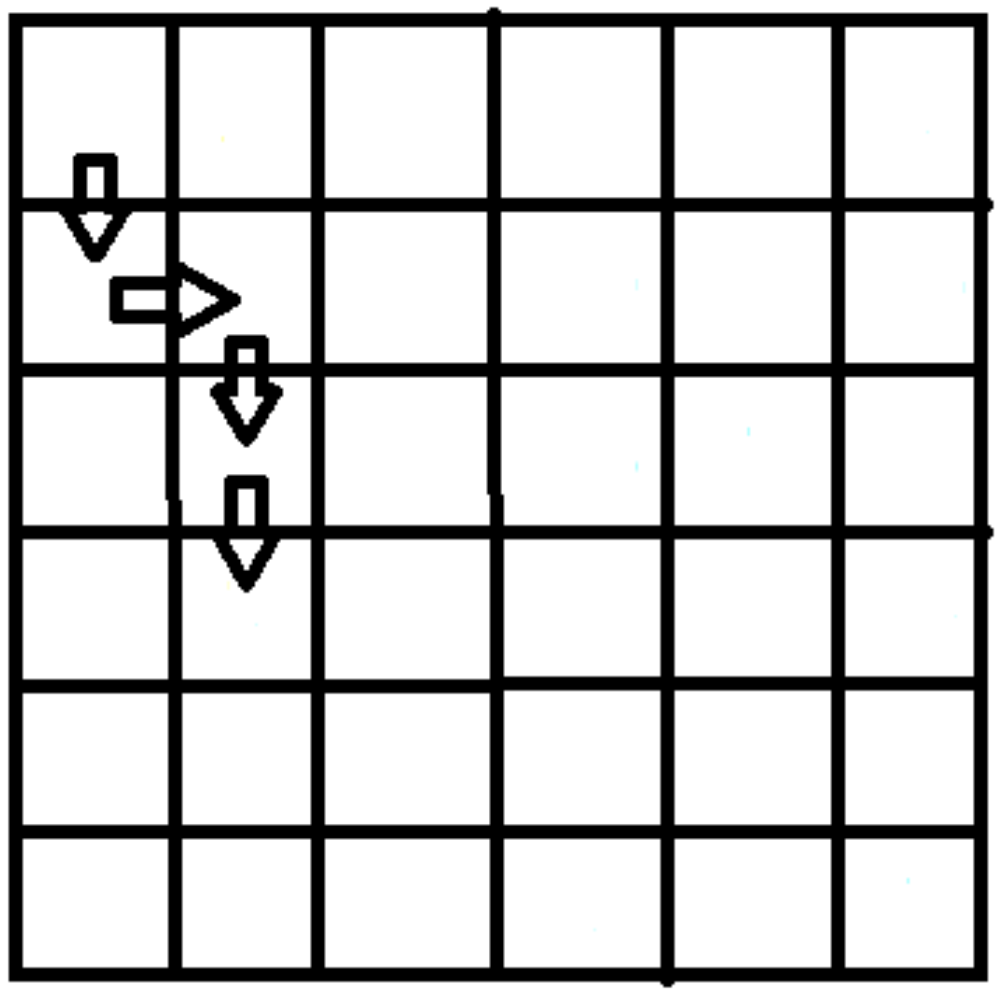


找出中间点。

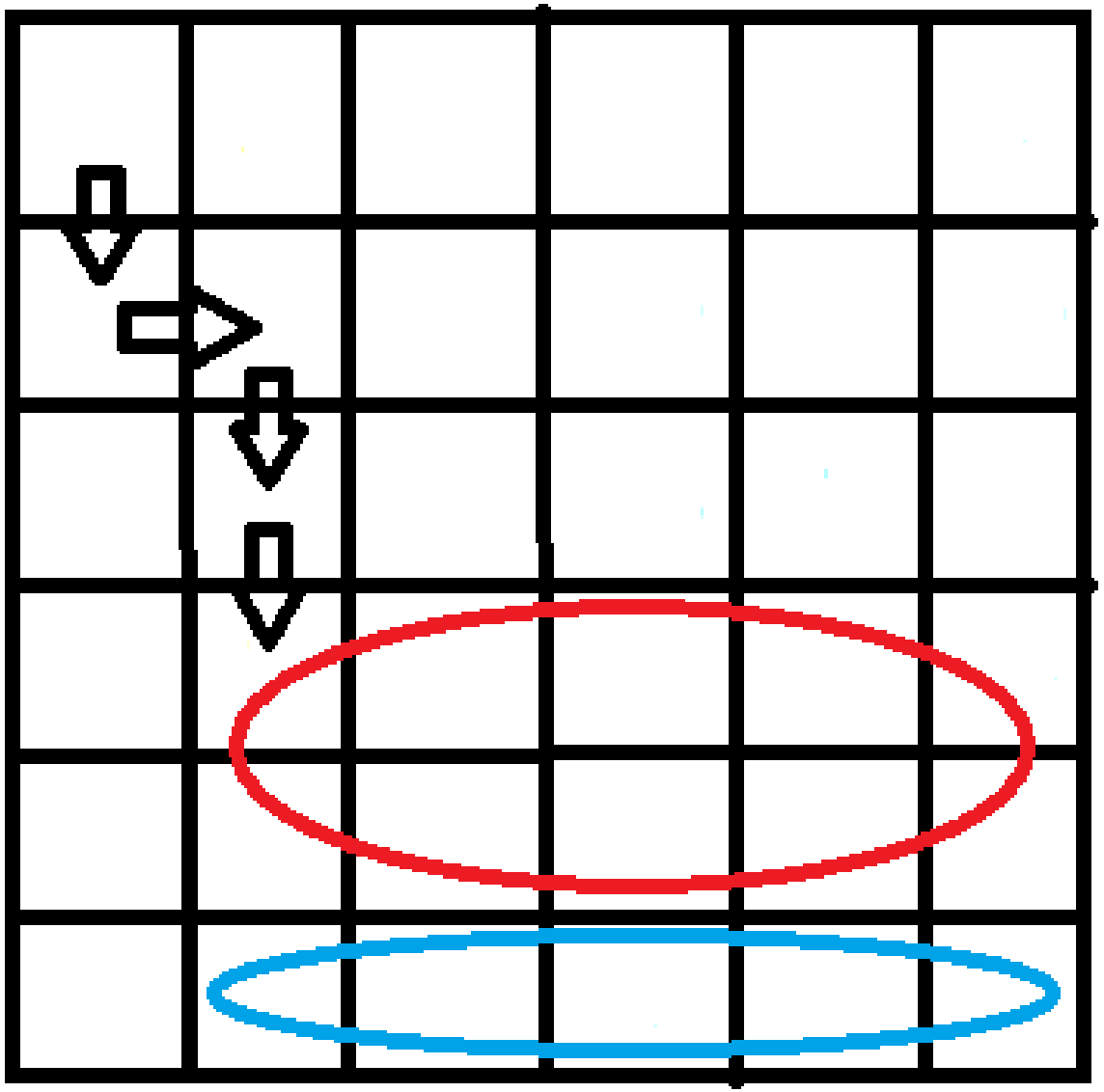


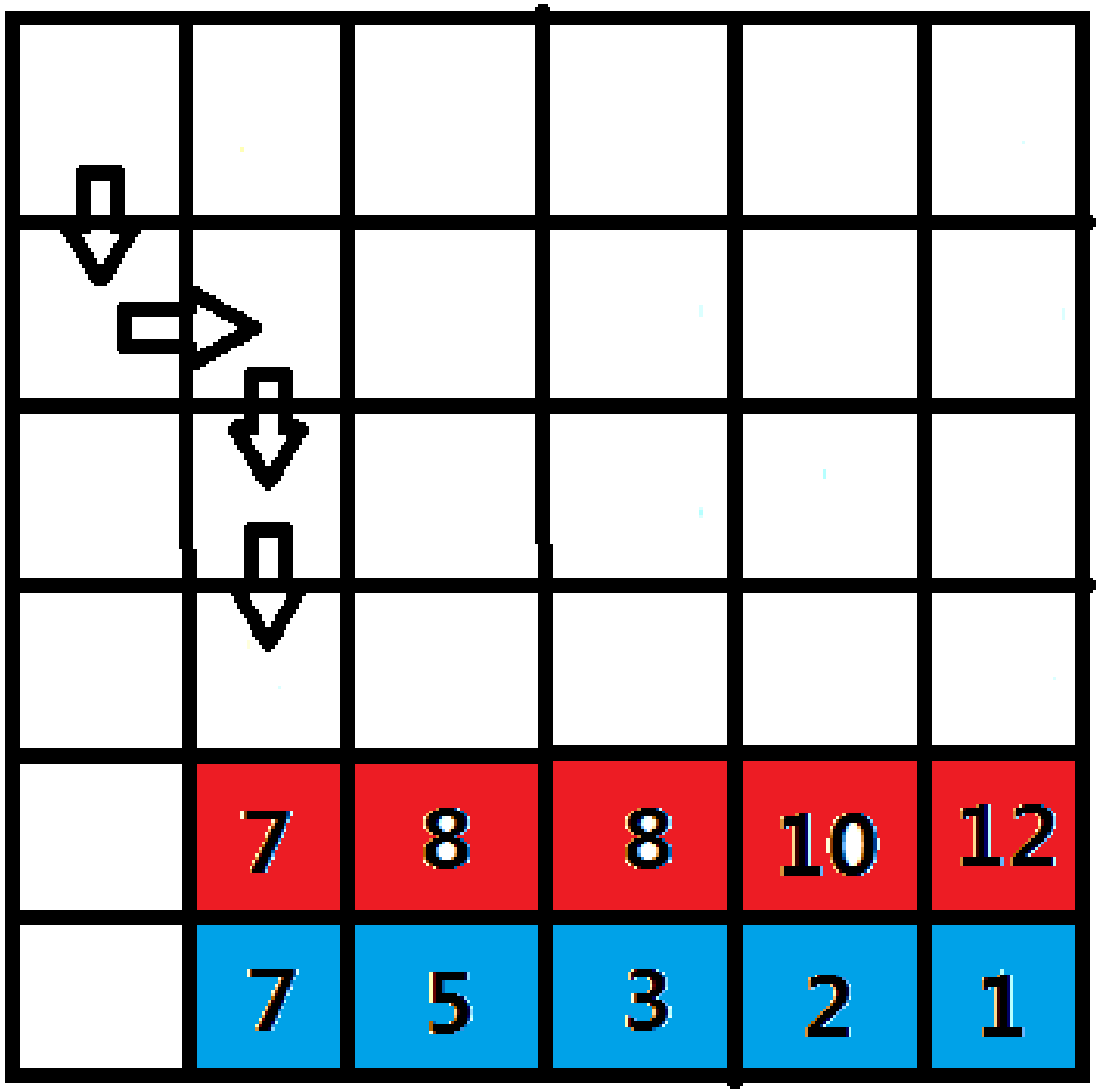
发现再分成的子子子子矩阵都已经做完了，这时就得到了整个矩阵左上部分答案。

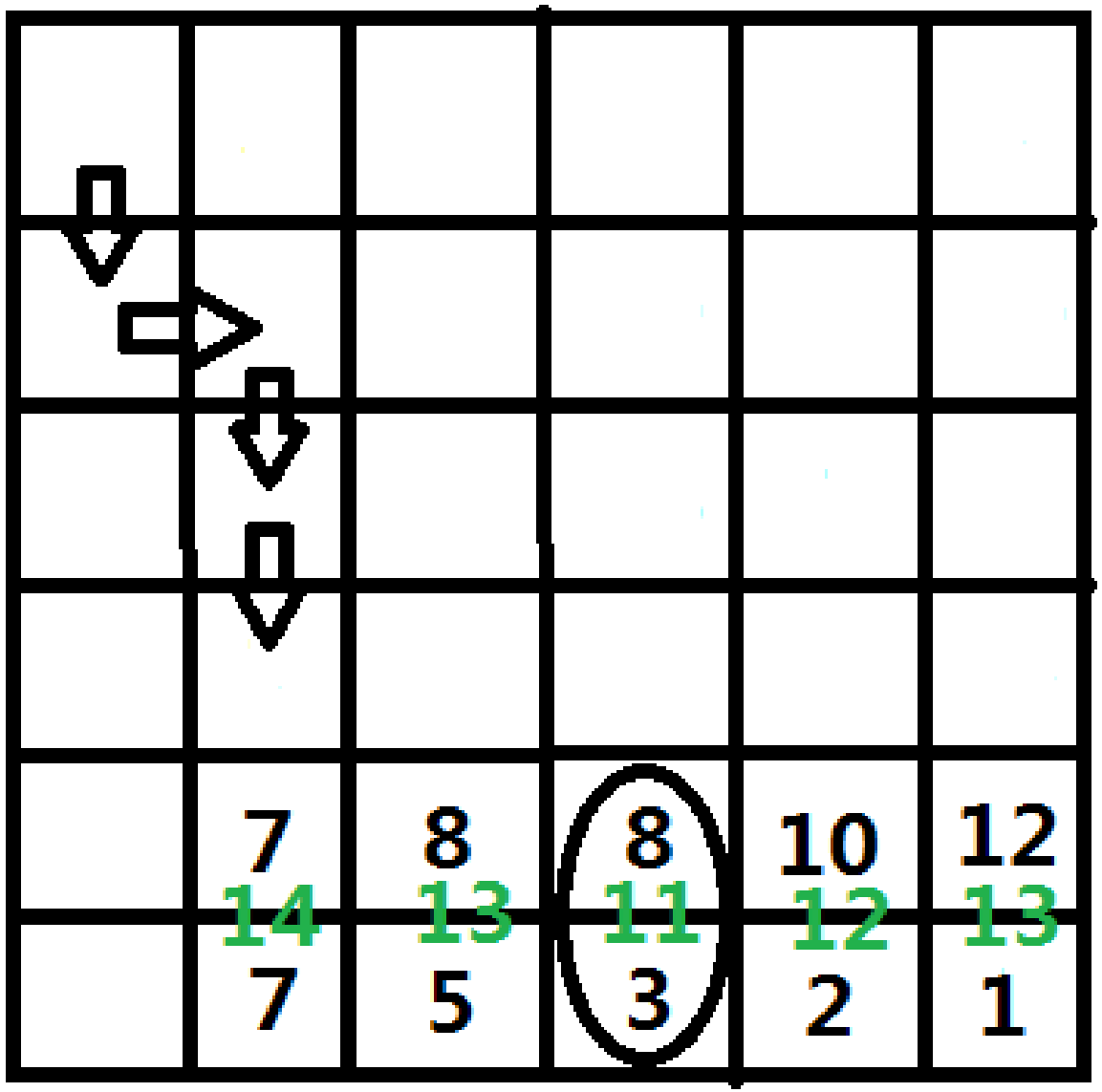
注意，向右的部分是因为要从 $(2,1)$ 走到 $(2,2)$（第 1 列走到第 2 列）。

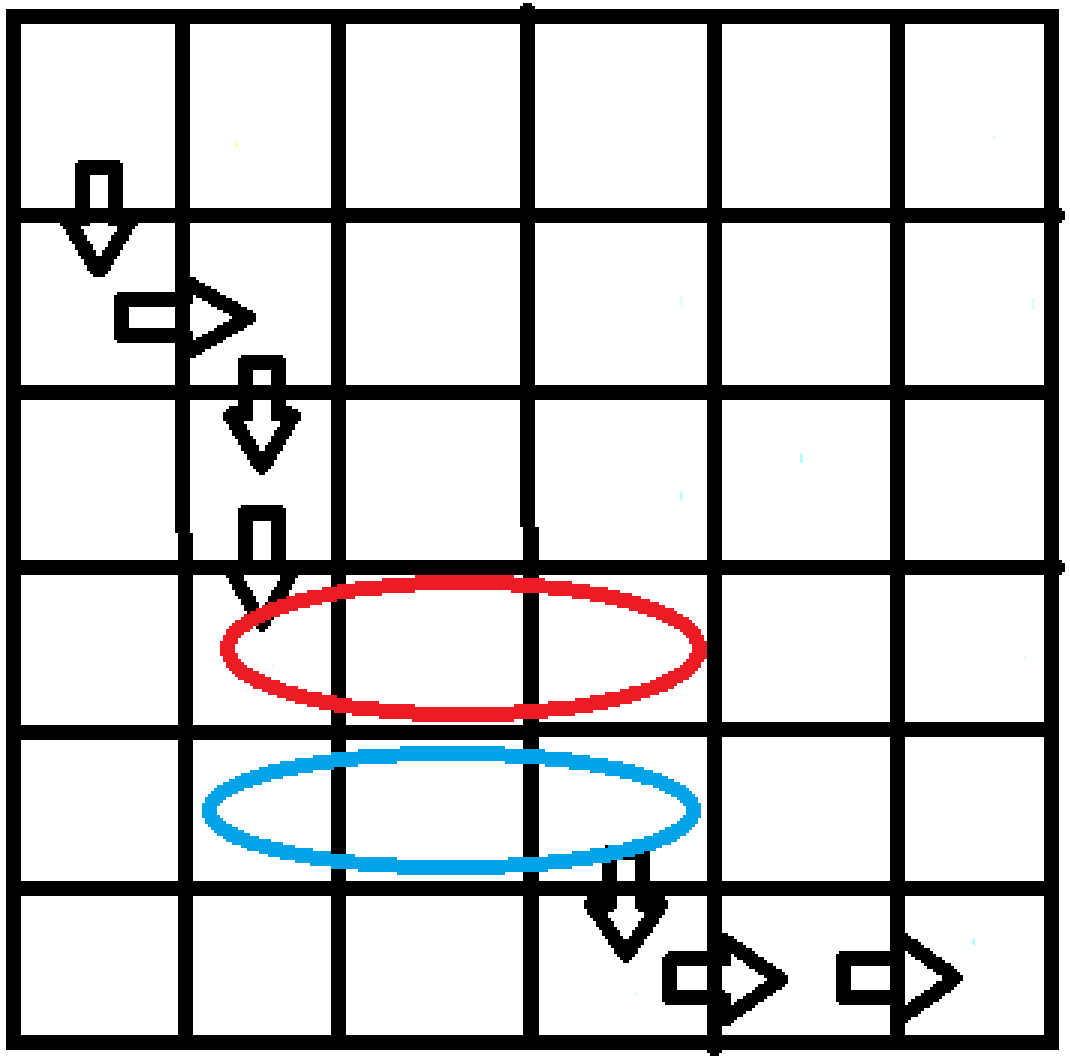


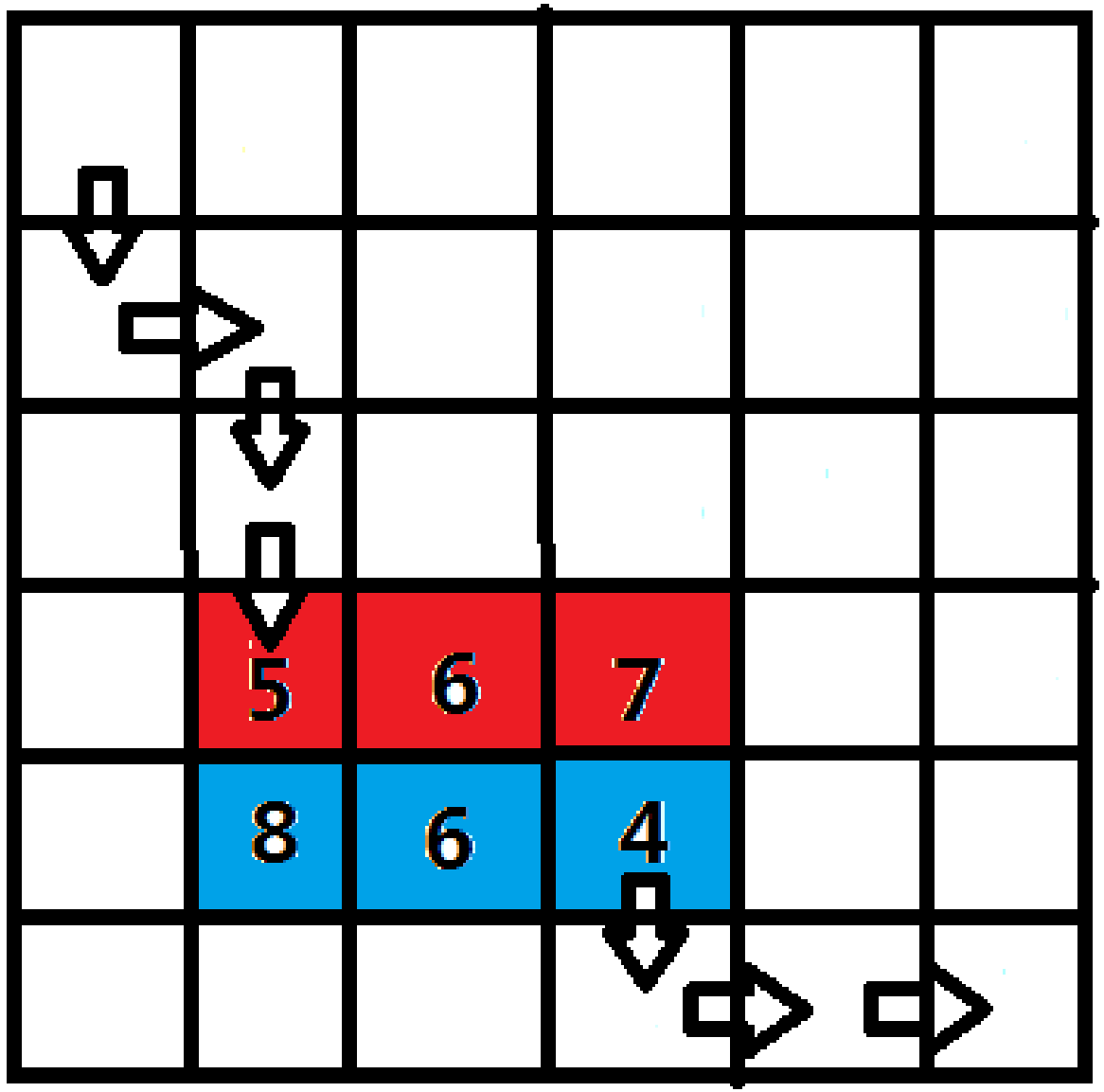
对下面的那个子矩阵也是同样的操作。

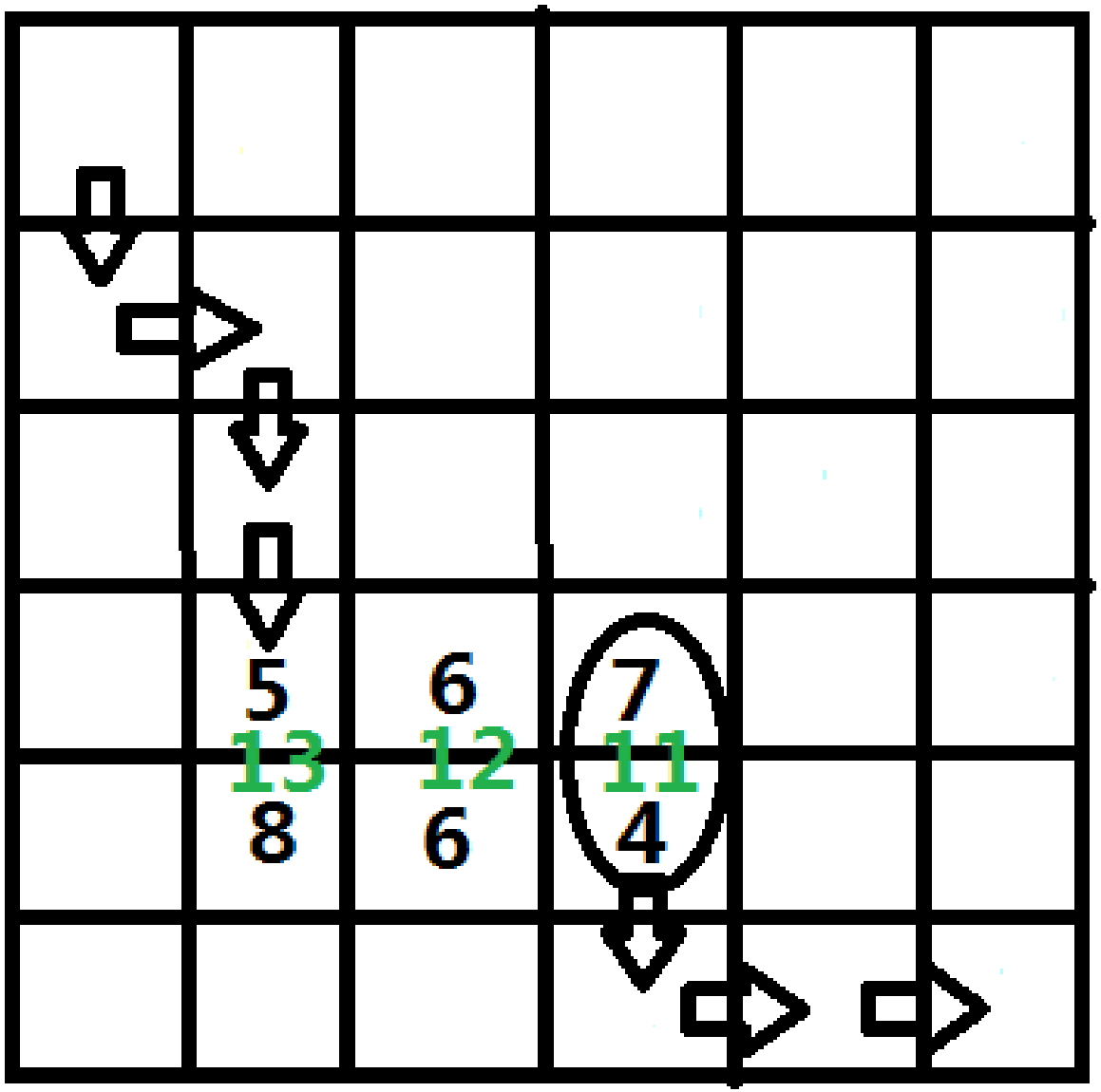



7
8
8
10
12
7
5
3
2
1

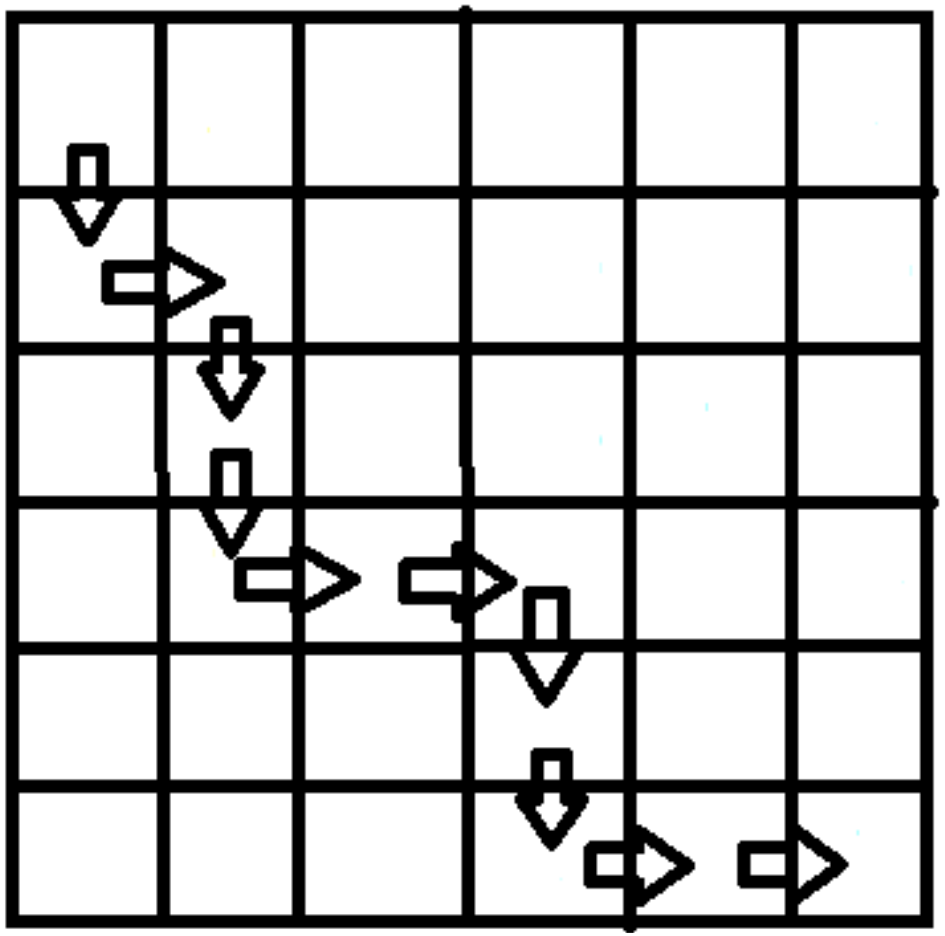

7
8
8
10
12
14
13
11
12
13
7
5
3
2
1




5
6
7
8
6
4


5
6
7
13
12
11
8
6
4

至此，因为分治的子区间不重复不遗漏，所以所有的路径都被找出来了，顺带最小值的计算也可通过路径上的所有 a 相加得到。



时间复杂度 $O(NM + \frac{NM}{2} + \frac{NM}{4} + \cdots) \approx O(NM)$ ，但是空间复杂度只有 $O(N + M)$ 了。

由此得出，分治的基本特征——

可以把原问题分解成多个子问题

子问题间互不影响

可以快速合并多个子问题

题目A

时间限制：1000MS　　内存限制：131072KB

商店里总共有 N 种装饰品，每种装饰品只有一个。第 i 种装饰品的价格为 p_i 元，好看度为 b_i。但是这家商店有一个特殊的地方，就是每天都会把一种装饰品藏起来。

假设 ***** 在某一天来到了这家商店，这一天第 a 种装饰品被藏了起来，他带了 c 块钱，他想知道他最多能为他的小♂屋增加多少的好看度呢？

对于 30% 的数据，$N \leq 100, Q \leq 1000$。

对于 100% 的数据，$N \leq 1000, Q \leq 100000, 1 \leq a \leq N, c \leq 1000$。

按照惯例，这题可以用一种奇怪的做法卡过去：正反两遍背包，然后对于每个询问 $O(c)$ 查找答案。时间复杂度 $O(Qc \approx 10^8)$ ，数据比较水就卡过去了。

但是我们必须思考正解而不是套算法！

首先，由于“删除”这个操作不是很好实现，所以把思维逆转过来，考虑“**增加**”这个操作。

接着，还发现一个**性质**：如果不选的物品 a 在区间 $[l,r]$ 中，那么 a 不是在 $[l,\lfloor\frac{l+r}{2}\rfloor]$ 中就是在 $[\lfloor\frac{l+r}{2}\rfloor+1,r]$ 中。这提示我们，可以用**分治**的思想把这个问题分成多个子问题来解决。

对于一个区间 $[l,r]$ ，如果 a 在 $[l,\lfloor\frac{l+r}{2}\rfloor]$ 中，那么**另一个区间** $[\lfloor\frac{l+r}{2}\rfloor+1,r]$ 中的物品**都是可以选**的，所以对这些物品做01背包，然后继续处理 $[l,\lfloor\frac{l+r}{2}\rfloor]$ ；若 a 在 $[\lfloor\frac{l+r}{2}\rfloor+1,r]$ 中也是同样。

但是很明显，在分治的时候需要不选 $[l,r]$ 区间物品，其他物品都选时的动态规划 f 数组作为**初始状态**，这样才能继续对这个区间进行规划和分割。同时必须在回溯的时候把这个数组还原。对于每次递归和回溯都重新对除了 $[l,r]$ 中的物品做一遍01背包代价是很大的，可以达到 $O(N^2c)$ 等于是退化了。

注意到分治的递归树最多只有 $log_2N^2\approx 19$ 层，所以可以直接把**每一层**的动态规划 f 数组都**存下来**，在对这一层进行规划的时候，直接把上一层的 f 当做初始状态即可。这样回溯的时候也不必考虑如何保存并还原 f 数组了。

最后，当分治到 $l=r$ 时，这个 a 元素也就被确定下来了，更新去掉 a 的答案的 f' 数组即可。

这样，时间复杂度就只有 $O(NclogN)$ 足以通过本题。

分治的运用特征——

询问是离线的

利用数据结构处理比较麻烦

能进行子问题的分割和合并

题目B

时间限制： 1000ms　　内存限制： 256MB

给定一个长度为 n 的非负整数序列 a 和一个正整数 m 。

现在有 q 组询问，每组询问给定两个正整数 l, r ，每次可以选择满足 $l \leq i \leq r$ 的若干个 a_i （也可以一个都不选），使得这些 a_i 的和是 m 的非负整数倍，并输出满足条件的选择方案数对 $10^9 + 7$ 取模后的余数。

对于 100% 的数据，有 $1 \leq n, q \leq 2 \times 10^5$ ，$1 \leq m \leq 20$ ，$0 \leq a_i \leq 10^9$ 。

保证每组询问的 l, r 满足 $1 \leq l \leq r \leq n$ 。

（注：以下的取模操作均在**非负整数**意义下。）

最暴力的做法就是对于每一个询问的区间暴力做一遍**背包计数**，容量就变成 $\bmod m$ 的余数 $0\sim(m-1)$ 了。转移就是 $f_{i,j}=f_{i-1,j}+f_{i-1,(j-a_i)\%m}$ ，表示 a_i 不选和选的情况数之和。这样的复杂度可以达到 $O(nmq)$ 是无法承受的。

还可以想到线段树进行背包合并，但是这样的复杂度为 $O((n+q)m^2logn)$ ，会 TLE 。

所以想到进行分治。

首先**离线**处理询问，然后考虑区间 $[l,r]$ 如何对这个询问产生贡献。首先，如果这个询问区间完全在 $[l,mid-1]$ 中，那么先**不用处理**它，在递归进 $[l,mid-1]$ 时处理就好了（**注意**，区间端点 $=mid$ 的会在下面进行处理）。同样如果这个询问区间完全在 $[mid+1,r]$ 中也是同理。

如果这个询问区间跨过了 mid（$l \le ql \le mid \le qr \le r$），那么这个询问的答案就可以被处理出来。注意因为 $[l,r]$ 被分割成了 $[l,mid]$ 和 $[mid+1,r]$ 这两个子区间，所以询问区间 $[ql,qr]$ 也可以被分割成两个子区间 $[ql,mid]$ 和 $[mid+1,qr]$。要快速的合并，所以对 $[mid+1,r]$ 中的元素做一遍**正向DP**设得到 f 数组，对 $[l,mid]$ 中的元素做一遍**反向DP**设得到 g 数组，这样 f 和 g 合并就可以得到跨过 mid 的一段区间的完整的答案了。合并的时候就直接把 f 中的方案数和 g 中的方案数**相乘**即可，即 $[ql,qr]$ 这个询问的答案是 $\sum\limits_{j+k=m} f_{qr,j} \times g_{ql,k}$。

这样处理下去，时间复杂度只有 $O(m(nlogn+q))$ 足以通过本题。

例题2

时间限制：1000MS　　内存限制：262144KB

有 n 个正整数构成序列 a ，定义一个区间 $[l,r]$ 的代价为 满足 $l \le i,j \le r$ 并使得 $a_i = a_j$ 的**无序对** $[i,j]$ 的数量。

现要把 a 分成 k 个互不相交且不为空的连续的区间，求出在所有分法中，分出区间的**最小代价和**是多少。

对于 100% 的数据，$2 \le n \le 10^5$ ，$2 \le k \le min\{20,n\}$ 。

保证 $1 \le a_i \le n$ 。

设 $f_{i,j}$ 为做到第 i 个数，$1\sim i$ **中分了** j **段的最小代价。**

所以 $f_{i,j}=\min\limits_{k=1}^{i-1}\{f_{k,j-1}+w_{k,i}\}$ ，$w_{k,i}$ 表示 $[k+1,i]$（**注意范围**）为一段的代价（两两相同的数的**无序对对数**），发现 j 这一维可以直接**滚动**掉~~（不滚动其实也没什么 www）~~。

就变成了 $g_i=\min\limits_{k=1}^{i-1}\{f_k+w_{k,i}\}$ ，时间复杂度 $O(n^2k)$ 无法通过本题。

注意到**决策的单调性**。即对于 $a<b<c<d$，如果 c 从 b 转移过来比从 a 转移过来更优，那么 d 从 b 转移过来也比从 a 转移过来更优。即若满足 $f_b+w_{b,c}\le f_a+w_{a,c}$ 那么一定满足 $f_b+w_{b,d}\le f_a+w_{a,d}$。注意到只要满足 $w_{b,d}-w_{b,c}\le w_{a,d}-w_{a,c}$ 那么上面的结论一定成立。发现这个条件其实就是四边形不等式 $w_{a,c}+w_{b,d}\le w_{a,d}+w_{b,c}$

.

对于本题，假设颜色 a 分别在 $(a,b]$ 、 $(b,c]$ 、 $(c,d]$ 中的数量为 x,y,z。那么就有如下关系（倒推）：

$$w_{a,c}+w_{b,d}\le w_{a,d}+w_{b,c}$$

$$C_{x+y}^2+C_{y+z}^2\le C_{x+y+z}^2+C_y^2$$

$$(x+y)(x+y-1)+(y+z)(y+z-1)\le(x+y+z)(x+y+z-1)+(y+z)(y+z-1)$$

$$0\le xz$$

$0\le xz$ ~~那是废话~~一定正确，也就证明了决策的单调性。

同时还发现此题的 $w_{k,i}$ 无法对于指定的 k, i 在 $O(1)$ 时间内求出，所以无法使用决策二分栈等方法，考虑操作是单调并且**离线**，所以使用**分治**。

当前求解的区间为 $[l, r]$ ，而最优决策可能在的区间为 $[kl, kr]$ 。则对于 $mid = \lfloor \frac{l+r}{2} \rfloor$ ，需要暴力找出 $[kl, min\{mid - 1, kr\}]$ 中最优的转移点 k 。

要处理 $[l, mid)$ 时，它的决策区间为 $[kl, k]$ 。因为 w 原本就表示 $[kl, l]$ 这一区间的代价，所以不需要加任何处理即可直接递归进这个子区间。

值得注意的是，处理 $(mid, r]$ 时，它的决策区间为 $[k, kr]$ 。对照上面的图发现有 $[l, k)$ 这一段信息还未处理，所以按照老办法处理，然后就可以递归进这个子区间了。记得回溯的时候复原。

这样分治的优秀复杂度 $O(nklogn)$ 就可以通过本题了。

分治法的简单运用

第一种类型

处理离线询问

原问题必须可以被分解成若干个子问题

合并子问题的复杂度要有保证

第二种类型

DP 优化

满足决策单调性

本质：优化重复的状态

例题及题目： YZOJ P2384, P2202, P4199, P4203

AAS：分治-初级-…

如果你想看高级也没有问题