第 397 场周赛

1. 100296. 两个字符串的排列差

给你两个字符串 `s` 和 `t`，每个字符串中的字符都不重复，且 `t` 是 `s` 的一个排列。

排列差 定义为 `s` 和 `t` 中每个字符在两个字符串中位置的绝对差值之和。

返回 `s` 和 `t` 之间的 **排列差** 。

示例 1：

输入： s = "abc", t = "bac"

输出： 2

解释：

对于 `s = "abc"` 和 `t = "bac"`，排列差是：

- `"a"` 在 `s` 中的位置与在 `t` 中的位置之差的绝对值。
- `"b"` 在 `s` 中的位置与在 `t` 中的位置之差的绝对值。
- `"c"` 在 `s` 中的位置与在 `t` 中的位置之差的绝对值。

即，`s` 和 `t` 的排列差等于 `|0 - 1| + |2 - 2| + |1 - 0| = 2`。

示例 2：

输入： s = "abcde", t = "edbac"

输出： 12

解释： `s` 和 `t` 的排列差等于 `|0 - 3| + |1 - 2| + |2 - 4| + |3 - 1| + |4 - 0| = 12`。

提示：

- `1 <= s.length <= 26`
- 每个字符在 `s` 中最多出现一次。
- `t` 是 `s` 的一个排列。
- `s` 仅由小写英文字母组成。

思路：依据题目的意思遍历即可

1. Java

```java
class Solution {
    public int findPermutationDifference(String s, String t) {
        int[] pos = new int[26];
        for (int i = 0; i < s.length(); i++) {
            pos[s.charAt(i) - 'a'] = i;
        }
        int ans = 0;
        for (int i = 0; i < t.length(); i++) {
            ans += Math.abs(i - pos[t.charAt(i) - 'a']);
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def findPermutationDifference(self, s: str, t: str) -> int:
        pos = {c: i for i, c in enumerate(s)}
        return sum(abs(i - pos[c]) for i, c in enumerate(t))
```

2. 100274. 从魔法师身上吸取的最大能量

在神秘的地牢中，`n` 个魔法师站成一排。每个魔法师都拥有一个属性，这个属性可以给你提供能量。有些魔法师可能会给你负能量，即从你身上吸取能量。

你被施加了一种诅咒，当你从魔法师 `i` 处吸收能量后，你将被立即传送到魔法师 `(i + k)` 处。这一过程将重复进行，直到你到达一个不存在 `(i + k)` 的魔法师为止。

换句话说，你将选择一个起点，然后以 `k` 为间隔跳跃，直到到达魔法师序列的末端，**在过程中吸收所有的能量。**

给定一个数组 `energy` 和一个整数 `k`，返回你能获得的 **最大** 能量。

示例 1：

输入： energy = [5,2,-10,-5,1], k = 3

输出： 3

解释： 可以从魔法师 1 开始，吸收能量 2 + 1 = 3。

示例 2：

输入： energy = [-2,-3,-1], k = 2

输出： -1

解释： 可以从魔法师 2 开始，吸收能量 -1。

提示：

- `1 <= energy.length <= 105`
- `-1000 <= energy[i] <= 1000`
- `1 <= k <= energy.length - 1`

思路：

1. 动态规划

我们定义 $dp[i]$ 为当前位置可以吸取能量的最大值，因为每个位置的能量都依赖于后面位置的能量，所以必须倒序遍历

2. 枚举 O(1) 空间做法

枚举起点为 i , 我们要计算的是下标为

$$i, i+k, i+2k, \cdots, i+kx$$

的元素之和的最小值。

由于 $i+kx < n$, 解得 x 的最大值为

$$\lfloor \frac{n-i-1}{k} \rfloor * k$$

我们可以从终点 $i+kx$ 开始倒着枚举，一边枚举一边累加元素和，取元素和的最大值，即为答案。

i 可以从 0 开始枚举到 $k-1$

1. Java

1. 动态规划

```java
class Solution {
    public int maximumEnergy(int[] energy, int k) {
        int n = energy.length;
        int[] dp = new int[n];
        int maxEnergy = Integer.MIN_VALUE;
        for (int i = n - 1; i >= 0; i--) {
            dp[i] = energy[i];
            if (i + k < n) {
                dp[i] += dp[i + k];
            }
            maxEnergy = Math.max(maxEnergy, dp[i]);
        }
        return maxEnergy;
    }
}
```

2. 枚举

```java
class Solution {
    public int maximumEnergy(int[] energy, int k) {
        int ans = Integer.MIN_VALUE;
        for (int i = 0; i < k; i++) {
            int s = 0;
            for (int j = i + (energy.length - i - 1) / k * k; j >= 0; j -= k) {
                s += energy[j];
                ans = Math.max(ans, s);
            }
        }
        return ans;
    }
}
```

2. Python

1. 动态规划

```python
class Solution:
    def maximumEnergy(self, energy: List[int], k: int) -> int:
        n = len(energy)
        dp = [0] * n
        maxEnergy = -inf
        for i in range(n - 1, -1, -1):
            dp[i] = energy[i]
            if i + k < n:
                dp[i] += dp[i + k]
            maxEnergy = max(maxEnergy, dp[i])
        return maxEnergy
```

2. 枚举

```python
class Solution:
    def maximumEnergy(self, energy: List[int], k: int) -> int:
        ans = -inf
        for i in range(k):
            end = i + (len(energy) - i - 1) // k * k
            ans = max(ans, max(accumulate(energy[j] for j in range(end, -1, -k))))
        return ans
```

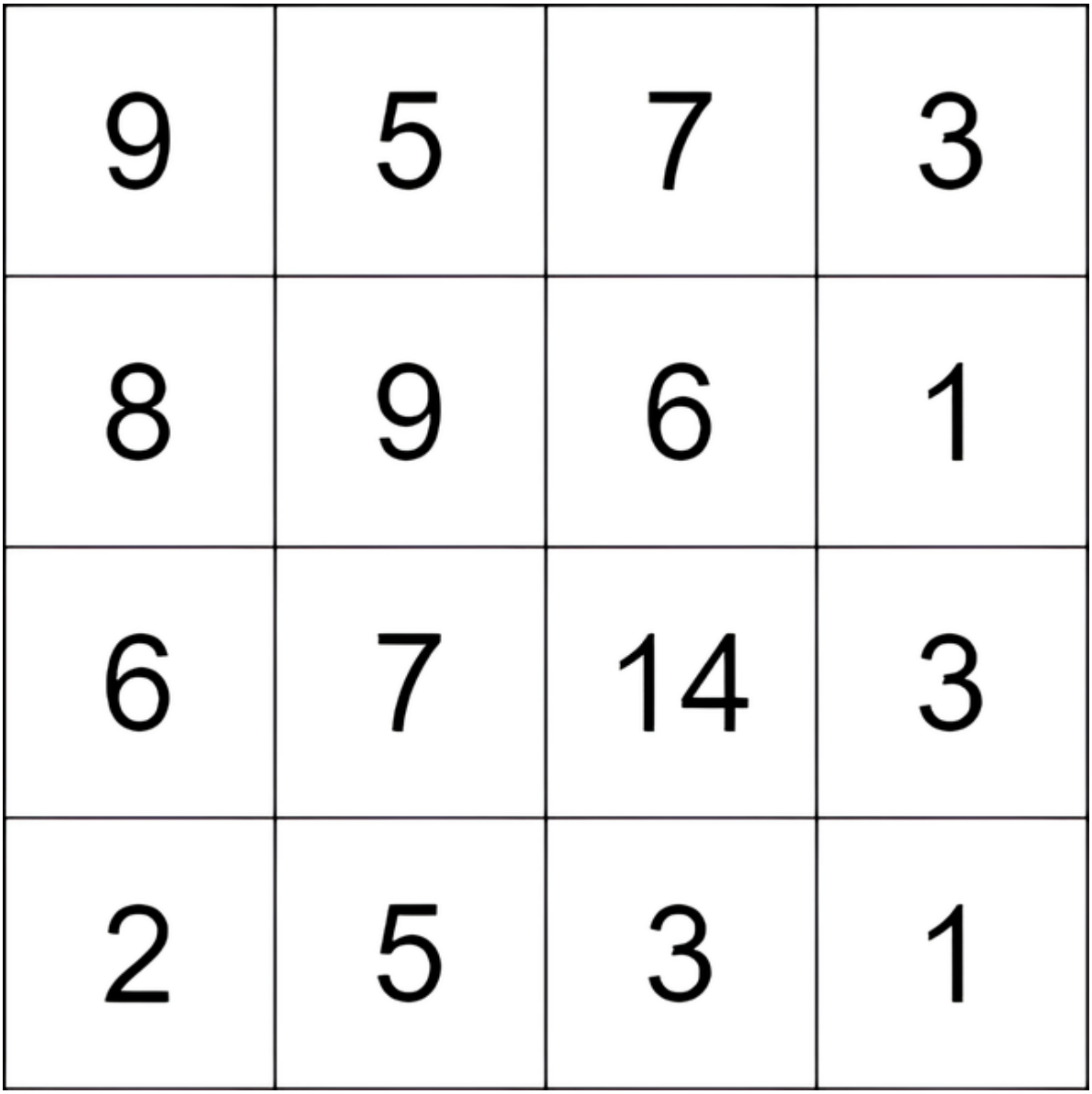
3. 100281. 矩阵的最大得分

给你一个由 **正整数** 组成、大小为 `m x n` 的矩阵 `grid`。你可以从矩阵中的任一单元格移动到另一个位于正下方或正右侧的任意单元格（不必相邻）。从值为 `c1` 的单元格移动到值为 `c2` 的单元格的得分为 `c2 - c1` 。

你可以从 **任一** 单元格开始，并且必须至少移动一次。

返回你能得到的 **最大** 总得分。

示例 1：

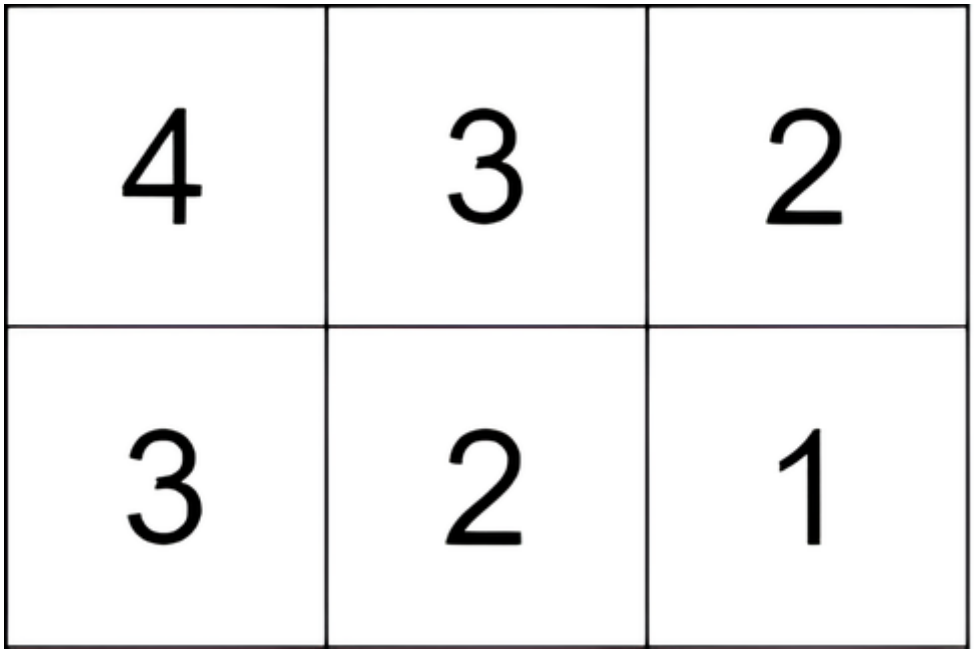


输入： grid = [[9,5,7,3],[8,9,6,1],[6,7,14,3],[2,5,3,1]]

输出： 9

解释： 从单元格 `(0, 1)` 开始，并执行以下移动：
- 从单元格 `(0, 1)` 移动到 `(2, 1)`，得分为 `7 - 5 = 2` 。
- 从单元格 `(2, 1)` 移动到 `(2, 2)`，得分为 `14 - 7 = 7` 。
总得分为 `2 + 7 = 9` 。

示例 2：



输入： grid = [[4,3,2],[3,2,1]]

输出： -1

解释： 从单元格 `(0, 0)` 开始，执行一次移动：从 `(0, 0)` 到 `(0, 1)` 。得分为 `3 - 4 = -1` 。

提示：

- `m == grid.length`
- `n == grid[i].length`
- `2 <= m, n <= 1000`
- `4 <= m * n <= 105`
- `1 <= grid[i][j] <= 105`

思路：

把 $grid[i][j]$ 视作海拔高度，题目要计算的是重力势能的变化量之和，也就是终点和起点的海拔高度之差。

枚举终点海拔高度，那么起点海拔高度越小越好，且起点只能在 (i,j) 的左上方向（可以是正左或者正上）。

按照二维前缀的思路，定义 $f[i+1][j+1]$ 表示左上角在 $(0,0)$，右下角在 (i,j) 的子矩阵的最小值。

类似二维前缀和，我们有

$$f[i+1][j+1]=min(f[i+1][j],f[i][j+1],grid[i][j])$$

注意起点终点不能重合，如果终点在 $(i,j)(i,j)(i,j)$，那么起点的最小值为

$$min(f[i+1][j],f[i][j+1])$$

取终点起点之差的最大值，即为答案。

1. Java

```java
class Solution {
    public int maxScore(List<List<Integer>> grid) {
        int ans = Integer.MIN_VALUE;
        int m = grid.size(), n = grid.get(0).size();
        int[][] f = new int[m + 1][n + 1];
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                f[i][j] = Integer.MAX_VALUE;
            }
        }
        for (int i = 0; i < m; i++) {
            List<Integer> row = grid.get(i);
            for (int j = 0; j < n; j++) {
                int mn = Math.min(f[i + 1][j], f[i][j + 1]);
                int x = row.get(j);
                ans = Math.max(ans, x - mn);
                f[i + 1][j + 1] = Math.min(mn, x);
            }
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def maxScore(self, grid: List[List[int]]) -> int:
        ans = -inf
        m, n = len(grid), len(grid[0])
        f = [[inf] * (n + 1) for _ in range(m + 1)]
        for i, row in enumerate(grid):
            for j, x in enumerate(row):
                mn = min(f[i + 1][j], f[i][j + 1])
                ans = max(ans, x - mn)
                f[i + 1][j + 1] = min(mn, x)
        return ans
```