# 【华为OD机考 统一考试机试C卷】灰度图存储（C++ Java JavaScript Python)

## 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试( B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

## 题目描述

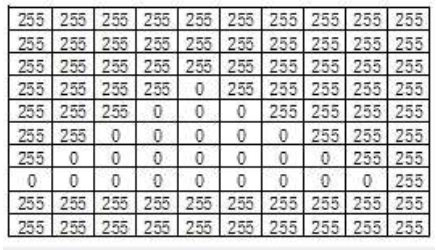
黑白图像常采用灰度图的方式存储，即图像的每个像素填充一个灰色阶段值，256节阶灰图是一个灰阶值取值范围为0-255的灰阶矩阵，0表示全黑，255表示全白，范围内的其他值表示不同的灰度。

但在计算机中实际存储时，会使用压缩算法，其中一个种压缩格式描述如如下：

10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21

1. 所有的数值以空格分隔；
2. 前两个数分别表示矩阵的行数和列数；
3. 从第三个数开始，每两个数一组，每组第一个数是灰阶值，第二个数表示该灰阶值从左到右，从上到下（可理解为二维数组按行存储在一维矩阵中）的连续像素个数。比如题目所述的例子， “255 34” 表示有连续 34 个像素的灰阶值是 255。

4. 如下图所：连续34个255，1个0 再来连续8个255。



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 0 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 0 | 0 | 0 | 255 | 255 | 255 | 255 |
| 255 | 255 | 0 | 0 | 0 | 0 | 0 | 255 | 255 | 255 |
| 255 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 255 | 255 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 255 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |

如此，图像软件在打开此格式灰度图的时候，就可以根据此算法从压缩数据恢复出原始灰度图矩阵。

请从输入的压缩数恢复灰度图原始矩阵，并返回指定像素的灰阶值。

## 输入描述

输入包行两行，第一行是灰度图压缩数据，第二行表示一个像素位置的行号和列号，如 0 0 表示左上角像素。

备注：

1、系保证输入的压缩数据是合法有效的，不会出现数据起界、数值不合法等无法恢复的场景；

2、系统保证输入的像素坐标是合法的，不会出现不在矩阵中的像素；

3、矩阵的行和列数范图为:(0,100];

4、灰阶值取值范图:[0,255];

```
10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21
3 4
```

## 输出描述

输出数据表示的灰阶矩阵的指定像素的灰阶值。

```
0 // 结合上面的图，第三行4列的值为0
```

## 用例1

输入：

```
10 10 56 34 99 1 87 8 99 3 255 6 99 5 255 4 99 7 255 2 99 9 255 21
3 4
```

输出：

```
99
```

> 说明： 将压缩数据恢复后的灰阶矩阵第3行第4列的像素灰阶值是99。

## 用例2

输入：

```
10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21
3 5
```

输出：

```
255
```

说明：

> 将压缩数据恢复后的灰阶矩阵第3行第5列的像案灰阶值是255。

## 解题思路

这题可太简单了，就是把题目中第一行的**一维数组转为二维数组**，然后找二维数组的某一个位置的值。

当然也可以选择不转换，直接用数学公式算！！！

## C++

```cpp
#include <iostream>
#include <vector>
#include <sstream>

using namespace std;
int main() {
    // 读取第一行数据，包含压缩后的图像数据
    string compressedLine;
    getline(cin, compressedLine);
    istringstream compressedStream(compressedLine);

    // 读取第二行数据，包含要查询的像素位置
    string positionLine;
    getline(cin, positionLine);
    istringstream positionStream(positionLine);

    // 解析图像矩阵的行数和列数
    int rows, cols;
    compressedStream >> rows >> cols;

    // 解析目标像素的行号和列号
    int targetRow, targetCol;
    positionStream >> targetRow >> targetCol;

    // 初始化图像矩阵
    vector<vector<int>> imageMatrix(rows, vector<int>(cols));
    int index = 0;  // 设置索引从压缩数据的第三个元素开始
    int value, count;
    int currentRow = 0, currentCol = 0;

    // 循环直到处理完所有压缩数据
    while (compressedStream >> value >> count) {
        // 根据连续像素个数填充图像矩阵
        for (int i = 0; i < count; ++i) {
            imageMatrix[currentRow][currentCol++] = value;
            // 如果当前列达到列数上限，移动到下一行并重置列号
            if (currentCol == cols) {
                currentRow++;
                currentCol = 0;
            }

```

```
        }
    }

    // 获取目标像素的灰阶值并输出
    cout << imageMatrix[targetRow][targetCol] << endl;

    return 0;
}
```

## Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 创建扫描器对象，用于读取输入数据
        Scanner scanner = new Scanner(System.in);
        // 读取第一行数据并分割成字符串数组，包含压缩后的图像数据
        String[] compressedData = scanner.nextLine().split(" ");
        // 读取第二行数据并分割成字符串数组，包含要查询的像素位置
        String[] pixelPosition = scanner.nextLine().split(" ");
        // 解析图像矩阵的行数
        int rows = Integer.parseInt(compressedData[0]);
        // 解析图像矩阵的列数
        int cols = Integer.parseInt(compressedData[1]);
        // 解析目标像素的行号
        int targetRow = Integer.parseInt(pixelPosition[0]);
        // 解析目标像素的列号
        int targetCol = Integer.parseInt(pixelPosition[1]);

        // 初始化图像矩阵
        int[][] imageMatrix = new int[rows][cols];
        // 设置索引从压缩数据的第三个元素开始
        int index = 2;
        // 初始化变量，用于存储灰阶值和连续像素个数
        int value, count;
        // 初始化当前填充的行号和列号
        int currentRow = 0, currentCol = 0;

        // 循环直到处理完所有压缩数据
```

```
        while (index < compressedData.length) {
            // 读取当前的灰阶值
            value = Integer.parseInt(compressedData[index++]);
            // 读取当前的连续像素个数
            count = Integer.parseInt(compressedData[index++]);

            // 根据连续像素个数填充图像矩阵
            for (int i = 0; i < count; i++) {
                // 将灰阶值赋给当前像素
                imageMatrix[currentRow][currentCol++] = value;
                // 如果当前列达到列数上限，移动到下一行并重置列号
                if (currentCol == cols) {
                    currentRow++;
                    currentCol = 0;
                }
            }
        }

        // 获取目标像素的灰阶值
        int targetValue = imageMatrix[targetRow][targetCol];
        // 输出目标像素的灰阶值
        System.out.println(targetValue);
        // 关闭扫描器
        scanner.close();
    }
}
```

## javaScript

```
const readline = require('readline');

// 创建 readline 接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 读取输入数据
rl.on('line', (line) => {
  // 使用闭包来存储状态
```

```
  if (!this.compressedData) {
    // 第一次调用，存储压缩数据
    this.compressedData = line.split(' ').map(Number);
    this.rows = this.compressedData[0];
    this.cols = this.compressedData[1];
  } else {
    // 第二次调用，存储像素位置并处理数据
    const pixelPosition = line.split(' ').map(Number);
    const targetRow = pixelPosition[0];
    const targetCol = pixelPosition[1];
    const imageMatrix = [];

    // 初始化图像矩阵
    for (let i = 0; i < this.rows; i++) {
      imageMatrix[i] = new Array(this.cols).fill(0);
    }

    let index = 2; // 从压缩数据的第三个元素开始
    let currentRow = 0, currentCol = 0;

    // 循环直到处理完所有压缩数据
    while (index < this.compressedData.length) {
      const value = this.compressedData[index++];
      const count = this.compressedData[index++];

      // 根据连续像素个数填充图像矩阵
      for (let i = 0; i < count; i++) {
        imageMatrix[currentRow][currentCol++] = value;
        // 如果当前列达到列数上限，移动到下一行并重置列号
        if (currentCol === this.cols) {
          currentRow++;
          currentCol = 0;
        }
      }
    }

    // 获取目标像素的灰阶值并输出
    console.log(imageMatrix[targetRow][targetCol]);

    // 关闭 readline 接口
```

```
    rl.close();
  }
});
```

## Python

```python
# 读取第一行数据，包含压缩后的图像数据
compressed_data = list(map(int, input().split()))

# 读取第二行数据，包含要查询的像素位置
pixel_position = list(map(int, input().split()))

# 解析图像矩阵的行数和列数
rows, cols = compressed_data[:2]

# 解析目标像素的行号和列号
target_row, target_col = pixel_position

# 初始化图像矩阵
image_matrix = [[0 for _ in range(cols)] for _ in range(rows)]

# 设置索引从压缩数据的第三个元素开始
index = 2
current_row, current_col = 0, 0

# 循环直到处理完所有压缩数据
while index < len(compressed_data):
    value = compressed_data[index]
    count = compressed_data[index + 1]
    index += 2

    # 根据连续像素个数填充图像矩阵
    for i in range(count):
        image_matrix[current_row][current_col] = value
        current_col += 1
        # 如果当前列达到列数上限，移动到下一行并重置列号
        if current_col == cols:
            current_row += 1
            current_col = 0

```

```
35
36  # 获取目标像素的灰阶值并输出
    print(image_matrix[target_row][target_col])
```

**文章目录**