题目描述

考勤记录是分析和考核职工工作时间利用情况的原始依据，也是计算职工工资的原始依据，为了正确地计算职工工资和监督工资基金使用情况，公司决定对员工的手机打卡记录进行异常排查。

如果出现以下两种情况，则认为打卡异常：

1. 实际设备号与注册设备号不一样
2. 或者，同一个员工的两个打卡记录的时间小于60分钟并且打卡距离超过5km。

给定打卡记录的字符串数组clockRecords（每个打卡记录组成为：工号;时间（分钟）;打卡距离（km）;实际设备号;注册设备号），返回其中异常的打卡记录（按输入顺序输出）。

输入描述

第一行输入为N，表示打卡记录数；

之后的N行为打卡记录，每一行为一条打卡记录。

输出描述

输出异常的打卡记录。

备注

- clockRecords长度 ≤ 1000
- clockRecords[i] 格式：{id},{time},{distance},{actualDeviceNumber},{registeredDeviceNumber}
- id由6位数字组成
- time由整数组成，范围为0~1000
- distance由整数组成，范围为0~100
- actualDeviceNumber与registeredDeviceNumber由思维大写字母组成

用例

输入	2 100000,10,1,ABCD,ABCD 100000,50,10,ABCD,ABCD
输出	100000,10,1,ABCD,ABCD;100000,50,10,ABCD,ABCD
说明	第一条记录是异常得，因为第二题记录与它得间隔不超过60分钟，但是打卡距离超过了5km，同理第二条记录也是异常得。

输入	2 100000,10,1,ABCD,ABCD 100001,80,10,ABCE,ABCE
输出	null
说明	无异常打卡记录，所以返回null

输入	2 100000,10,1,ABCD,ABCD 100000,80,10,ABCE,ABCD
输出	100000,80,10,ABCE,ABCD
说明	第二条记录得注册设备号与打卡设备号不一致，所以是异常记录

题目解析

我的解题思路如下：

首先，打卡记录异常，有两种类型：

题目解析

我的解题思路如下：

首先，打卡记录异常，有两种类型：

1、单条打卡记录自身比较，即单条打卡记录的实际设备号和注册设备号不同，则视为异常。

2、同一员工的两条打卡记录对比，如果打卡时间间隔小于60min，但是打卡距离却超过了5km，则视为异常

因此，我们应该先对每条打卡记录进行自身比较，即比较实际设备号，和注册设备号，如果不同，则直接判定异常。

> 另外，我理解，同一员工的（非设备号异常）打卡记录不需要再和（设备号异常）的打卡记录对比。

对于同一员工下设备号一致的打卡记录，需要两两对比，如果两个打卡记录的时间小于60分钟并且打卡距离超过5km，则视为异常。

但是这里有一个疑点，那就是一旦有两条打卡记录对比异常了，那么其他打卡记录是否还需要和这两条异常记录对比吗？

这点，题目描述没说，给的用例也无法验证。我理解是需要的，

另外，本题还有一个关键点就是：异常的打卡记录需要（按输入顺序输出）。

因此，我们在保存异常打卡记录时，还需要附加其输入时的索引，已方便按照输入索引输出。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = lines[0] - 0;
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const clockRecords = lines.map((line) => line.split(","));
    console.log(getResult(clockRecords));
    lines.length = 0;
  }
});

/**
 * @param {*} clockRecords 打卡记录的字符串数组, [工号, 时间, 打卡距离, 实际设备号, 注册设备号]
 */
function getResult(clockRecords) {
  const employees = {};

  const ans = new Set();

  for (let i = 0; i < clockRecords.length; i++) {
    const clockRecord = [...clockRecords[i], i]; // 由于异常打卡记录需要按输入顺序输出，因此这里追加一个输入索引到打卡
    const [id, time, dis, act_device, reg_device] = clockRecord;

    // 实际设备号与注册设备号不一样,则认为打卡异常
    if (act_device != reg_device) {
      ans.add(i);
    } else {
      // 如果实际设备号和注册设备号一样，则统计到该员工名下
      employees[id]
        ? employees[id].push(clockRecord)
        : (employees[id] = [clockRecord]);
    }
  }

  for (let id in employees) {
    // 某id员工的所有打卡记录
    const records = employees[id];
    const n = records.length;

    // 将该员工打卡记录按照打卡时间升序
    records.sort((a, b) => a[1] - b[1]);

    for (let i = 0; i < n; i++) {
      const time1 = records[i][1];
      const dis1 = records[i][2];

      for (let j = i + 1; j < n; j++) {
        const time2 = records[j][1];
        const dis2 = records[j][2];

```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = lines[0] - 0;
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const clockRecords = lines.map((line) => line.split(","));
    console.log(getResult(clockRecords));
    lines.length = 0;
  }
});

/**
 * @param {*} clockRecords 打卡记录的字符串数组, [工号, 时间, 打卡距离, 实际设备号, 注册设备号]
 */
function getResult(clockRecords) {
  const employees = {};

  const ans = new Set();

  for (let i = 0; i < clockRecords.length; i++) {
    const clockRecord = [...clockRecords[i], i]; // 由于异常打卡记录需要按输入顺序输出，因此这里追加一个输入索引到打卡
    const [id, time, dis, act_device, reg_device] = clockRecord;

    // 实际设备号与注册设备号不一样,则认为打卡异常
    if (act_device != reg_device) {
      ans.add(i);
    } else {
      // 如果实际设备号和注册设备号一样，则统计到该员工名下
      employees[id]
        ? employees[id].push(clockRecord)
        : (employees[id] = [clockRecord]);
    }
  }

  for (let id in employees) {
    // 某id员工的所有打卡记录
    const records = employees[id];
    const n = records.length;

    // 将该员工打卡记录按照打卡时间升序
    records.sort((a, b) => a[1] - b[1]);

    for (let i = 0; i < n; i++) {
      const time1 = records[i][1];
      const dis1 = records[i][2];

      for (let j = i + 1; j < n; j++) {
        const time2 = records[j][1];
        const dis2 = records[j][2];

        // 如果两次打卡时间超过60分治，则不计入异常，由于已按打卡时间升序，因此后面的都不用检查了
        if (time2 - time1 >= 60) break;
        else {
          // 如果两次打开时间小于60MIN，且打卡距离超过5KM，则这两次打卡记录算作异常
          if (Math.abs(dis2 - dis1) > 5) {
            // 如果打卡记录已经加入异常列表ans，则无需再次加入，否则需要加入
            if (!ans.has(records[i][5])) {
              ans.add(records[i][5]);
            }

            if (!ans.has(records[j][5])) {
              ans.add(records[j][5]);
            }
          }
        }
      }
    }
  }

  // 如果没有异常打卡记录，则返回null
  if (!ans.size) return "null";

  return [...ans]
    .sort((a, b) => a - b)
    .map((i) => clockRecords[i])
    .join(";");
}
```

Java算法源码

```java
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    String[][] clockRecords = new String[n][];
    for (int i = 0; i < n; i++) {
      clockRecords[i] = sc.next().split(",");
    }

    System.out.println(getResult(clockRecords));
  }

  /**
   * @param clockRecords 打卡记录的字符串数组, [工号, 时间, 打卡距离, 实际设备号, 注册设备号]
   * @return 异常打卡记录列表 (按输入顺序排序)
   */
  public static String getResult(String[][] clockRecords) {
    HashMap<String, ArrayList<String[]>> employees = new HashMap<>();
    TreeSet<Integer> ans = new TreeSet<>((a, b) -> a - b);

    for (int i = 0; i < clockRecords.length; i++) {
      // 由于异常打卡记录需要按输入顺序输出, 因此这里追加一个输入索引到打卡记录中
      String[] clockRecord = Arrays.copyOf(clockRecords[i], clockRecords[i].length + 1);
      clockRecord[clockRecord.length - 1] = i + "";

      String id = clockRecord[0];
      String act_device = clockRecord[3];
      String reg_device = clockRecord[4];

      // 如果实际设备号和注册设备号一样, 则统计到该员工名下
      if (act_device.equals(reg_device)) {
        employees.putIfAbsent(id, new ArrayList<>());
        employees.get(id).add(clockRecord);
      }
      // 实际设备号与注册设备号不一样,则认为打卡异常
      else {
        ans.add(i);
      }
    }

    for (String id : employees.keySet()) {
      // 某id员工的所有打卡记录records
      ArrayList<String[]> records = employees.get(id);

      // 将该员工打卡记录按照打卡时间升序
      records.sort((a, b) -> Integer.parseInt(a[1]) - Integer.parseInt(b[1]));

      for (int i = 0; i < records.size(); i++) {
        int time1 = Integer.parseInt(records.get(i)[1]);
        int dist1 = Integer.parseInt(records.get(i)[2]);

        for (int j = i + 1; j < records.size(); j++) {
          int time2 = Integer.parseInt(records.get(j)[1]);
          int dist2 = Integer.parseInt(records.get(j)[2]);

          // 如果两次打卡时间超过60分治, 则不计入异常, 由于已按打卡时间升序, 因此后面的都不用检查了
          if (time2 - time1 >= 60) break;
          else {
            // 如果两次打开时间小于60MIN, 且打卡距离超过5KM, 则这两次打卡记录算作异常
            if (Math.abs(dist2 - dist1) > 5) {
              // 如果打卡记录已经加入异常列表ans, 则无需再次加入, 否则需要加入
              ans.add(Integer.parseInt(records.get(i)[5]));
              ans.add(Integer.parseInt(records.get(j)[5]));
            }
          }
        }
      }
    }

    // 如果没有异常打卡记录, 则返回null
    if (ans.size() == 0) return "null";

    StringJoiner sj = new StringJoiner(";", "", "");
    ans.stream()
        .map(i -> clockRecords[i])
        .forEach(
            sArr -> {
              StringJoiner sj1 = new StringJoiner(",", "", "");
              for (String s : sArr) {
                sj1.add(s);
              }
              sj.add(sj1.toString());
            });

    return sj.toString();
  }
}
```

Python算法源码

```python
# 输入获取
n = int(input())
clockRecords = [input().split(",") for i in range(n)]


# 算法入口
def getResult(clockRecords):
    employees = {}
    ans = set()

    for i in range(len(clockRecords)):
        id, time, dis, act_device, reg_device = clockRecords[i]

        # 实际设备号与注册设备号不一样,则认为打卡异常
        if act_device != reg_device:
            ans.add(i)
        # 如果实际设备号和注册设备号一样，则统计到该员工名下
        else:
            # 由于异常打卡记录需要按输入顺序输出，因此这里追加一个输入索引到打卡记录中
            clockRecord = [id, time, dis, act_device, reg_device, i]
            if employees.get(id) is None:
                employees[id] = [clockRecord]
            else:
                employees[id].append(clockRecord)

    for id in employees.keys():
        # 某id员工的所有打卡记录
        records = employees[id]
        n = len(records)

        # 将该员工打卡记录按照打卡时间升序
        records.sort(key=lambda x: x[1])

        for i in range(n):
            time1 = int(records[i][1])
            dis1 = int(records[i][2])

            for j in range(i + 1, n):
                time2 = int(records[j][1])
                dis2 = int(records[j][2])

                # 如果两次打卡时间超过60分治，则不计入异常，由于已按打卡时间升序，因此后面的都不用检查了
                if time2 - time1 >= 60:
                    break
                else:
                    # 如果两次打开时间小于60MIN，且打卡距离超过5KM，则这两次打卡记录算作异常
                    if abs(dis2 - dis1) > 5:
                        # 如果打卡记录已经加入异常列表ans，则无需再次加入，否则需要加入
                        if records[i][5] not in ans:
                            ans.add(records[i][5])

                        if records[j][5] not in ans:
                            ans.add(records[j][5])

        if len(ans) > 0:
            tmp = list(ans)
            tmp.sort()
            return ";".join(list(map(lambda i: ",".join(clockRecords[i]), tmp)))
        else:
            return "null"


# 调用算法
print(getResult(clockRecords))
```