题目描述

张三要去外地出差，需要做核酸，需要在指定时间点前做完核酸，请帮他找到满足条件的核酸检测点。

- 给出一组核酸检测点的距离和每个核酸检测点当前的人数
- 给出张三要去做核酸的出发时间 出发时间是10分钟的倍数，同时给出张三做核酸的最晚结束时间
- 题目中给出的距离是整数，单位是公里，时间1分钟为一基本单位

去找核酸点时，有如下的限制：

- 去往核酸点的路上，每公里距离花费时间10分钟，费用是10元
- 核酸点每检测一个人的时间花费是1分钟
- 每个核酸点工作时间都是8点到20点中间不休息，核酸点准时工作，早到晚到都不检测
- 核酸检测结果可立刻知道
- 在张三去某个核酸点的路上花费的时间内，此核酸检测点的人数是动态变化的，变化规则是

1. 在非核酸检测时间内，没有人排队
2. 8点-10点每分钟增加3人
3. 12点-14点每分钟增加10人

要求将所有满足条件的核酸检测点按照优选规则排序列出：
优选规则：

1. 花费时间最少的核酸检测点排在前面。
2. 花费时间一样,花费费用最少的核酸检测点排在前面。
3. 时间和费用一样，则ID值最小的排在前面

输入描述

```
H1 M1
H2 M2
N
ID1 D1 C1
ID2 D2 C2
...
IDn Dn Cn
```

H1: 当前时间的小时数。
M1：当前时间的分钟数，
H2：指定完成核算时间的小时数。
M2：指定完成核算时间的分钟数。
N：所有核酸检测点个数。
ID1：核酸点的ID值。
D1：核酸检测点距离张三的距离。
C1：核酸检测点当前检测的人数。

输出描述

```
N
I2 T2 M2
I3 T3 M3
```

N：满足要求的核酸检测点个数
I2:选择后的核酸检测点ID
T2:做完核酸花费的总时间(分钟)
M3:去该核算点花费的费用

用例

10 30 14 50

N：满足要求的核酸检测点个数
I2:选择后的核酸检测点ID
T2:做完核酸花费的总时间(分钟)
M3:去该核算点花费的费用

用例

输入	10 30 14 50 3 1 10 19 2 8 20 3 21 3
输出	2 2 80 80 1 190 100
说明	无

题目解析

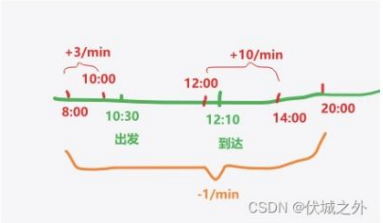
用例意思是：

张三在10:30出门，要在14:50之前做完核酸。

现在张三可选三个核酸检测点：

- 检测点1：距离张三10公里，10:30的时候有19个人排队
- 检测点2：距离张三8公里，10:30的时候有20个人排队
- 检测点3：距离张三21公里，10:30的时候有3个人排队

张三赶到检测点1，需要10*10 = 100元，10*10=100分钟，而在张三到达检测点时12:10，此时排队的人数是为：



通过上图，我们可以看出：在10:30~12:00期间不会有人加入，只会有人离开，每分钟离开1人，因此到12:00时，最多离开 12*60 - (10*60+30) = 90人，而10:30时只有19人排队，因此到12:00时，检测点1只有0人排队。

然后12:00到12:10阶段，每分钟离开1人，增加10人，因此相当于每分钟净增9人，因此到12:10，即张三到达时，检测点共有：10 * 9 = 90人。

因此张三还需排90分钟，才能做完核酸。

因此张三到检测点1的代价是：路上100分钟，到达后等待90分钟，共需190分钟，花费100元。

同理，可得张三去其他检测点的代价。

然后，过滤掉花费时间超出限制的代价，剩下的按照花费时间、花费金额排序即可。

我们可以通过求区间交集的方式，来获取张三【出发时间，到达时间】 和 【8:00，10:00】以及【10:00, 12:00】，以及【12:00, 14:00】以及【14:00，20:00】的交集。

其中，

- 和 【8:00，10:00】的交集，每分钟净增2人
- 和 【10:00, 12:00】的交集，每分钟净减1人
- 和 【12:00, 14:00】的交集，每分钟净增9人
- 和 【14:00，20:00】的交集，每分钟净减1人

2023.1.17补充说明：

根据网友m0_71826536的提示，如果张三在8:00前就赶到了核酸监测点，但是8:00前是不给排队的，因此张三还要等待到8:00，因此张三花费的时间其实是：路上时间 + 等待时间 + 排队时间

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 h1, m1, h2, m2, n;
rl.on("line", (line) => {
  lines.push(line);
```

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 h1, m1, h2, m2, n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 3) {
    [h1, m1] = lines[0].split(" ").map(Number);
    [h2, m2] = lines[1].split(" ").map(Number);
    n = lines[2] - 0;
  }

  if (n && lines.length === n + 3) {
    const idcs = lines.slice(3).map((line) => line.split(" ").map(Number));
    getResult(h1, m1, h2, m2, idcs);
    lines.length = 0;
  }
});

/**
 *
 * @param {*} h1 当前时间的小时数
 * @param {*} m1 当前时间的分钟数
 * @param {*} h2 指定完成核算时间的小时数
 * @param {*} m2 指定完成核算时间的分钟数
 * @param {*} idcs [[核酸点的ID值, 核酸检测点距离张三的距离,核酸检测点当前检测的人数]]
 */
function getResult(h1, m1, h2, m2, idcs) {
  const start = h1 * 60 + m1;
  const end = h2 * 60 + m2;

  const ans = idcs
    .map((idc) => {
      let [id, distance, count] = idc;
      const money = distance * 10;
      let road = distance * 10; // 花在路上的时间
      let arrived = start + road; // 到达核酸检测点的时间

      // 如果在8：00之前就赶到了，那么其实要等待到8:00才能排队，这里其实花费的时间应该包括等待的时间
      if (arrived < 8 * 60) {
        arrived = 8 * 60;
        road = arrived - start;
      }

      // [出发时间, 到达时间]
      // 和[8:00, 10:00]的交集，每分钟净增2人
      const add1 = getIntersection([start, arrived], [8 * 60, 10 * 60]);
      if (add1 !== false) {
        count += 2 * add1;
      }

      // 和[10:00, 12:00]的交集，每分钟净减1人
      const min1 = getIntersection([start, arrived], [10 * 60, 12 * 60]);
      if (min1 !== false) {
        count -= min1;
        count = Math.max(0, count); // 注意至多减到0
      }

      // 和[12:00, 14:00]的交集，每分钟净增9人
      const add2 = getIntersection([start, arrived], [12 * 60, 14 * 60]);
      if (add2 !== false) {
        count += 9 * add2;
      }

      // 和[14:00, 20:00]的交集，每分钟净减1人
      const min2 = getIntersection([start, arrived], [14 * 60, 20 * 60]);
      if (min2 !== false) {
        count -= min2;
        count = Math.max(0, count); // 注意至多减到0
      }

      return [id, count + road, money];
    })
    .filter(([id, real_end, money]) => real_end <= end)
    .sort((a, b) =>
      a[1] !== b[1] ? a[1] - b[1] : a[2] !== b[2] ? a[2] - b[2] : a[0] - b[0]
    );

  console.log(ans.length);
  ans.forEach((irm) => {
    console.log(irm.join(" "));
  });
}

// 获取交集长度，如果没有交集返回false
```

```
      const min2 = getIntersection([start, arrived], [14 * 60, 20 * 60]);
      if (min2 !== false) {
        count -= min2;
        count = Math.max(0, count); // 注意至多减到0
      }

      return [id, count + road, money];
    })
    .filter(([id, real_end, money]) => real_end <= end)
    .sort((a, b) =>
      a[1] !== b[1] ? a[1] - b[1] : a[2] !== b[2] ? a[2] - b[2] : a[0] - b[0]
    );

  console.log(ans.length);
  ans.forEach((irm) => {
    console.log(irm.join(" "));
  });
}

// 获取交集长度，如果没有交集返回false
function getIntersection(ran1, ran2) {
  const [s1, e1] = ran1;
  const [s2, e2] = ran2;

  if (s1 < s2) {
    if (s2 >= e1) return false;
    else return e1 - s2;
  } else if (s1 > s2) {
    if (s1 >= e2) return false;
    else return e2 - s1;
  } else {
    return Math.min(e1, e2) - s1;
  }
}
```

Java算法源码

```
import java.util.Arrays;
import java.util.Scanner;

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

    int h1 = sc.nextInt();
    int m1 = sc.nextInt();

    int h2 = sc.nextInt();
    int m2 = sc.nextInt();

    int n = sc.nextInt();

    int[][] idcs = new int[n][3];
    for (int i = 0; i < n; i++) {
      idcs[i][0] = sc.nextInt();
      idcs[i][1] = sc.nextInt();
      idcs[i][2] = sc.nextInt();
    }

    getResult(h1, m1, h2, m2, idcs);
  }

  /**
   * @param h1 当前时间的小时数
   * @param m1 当前时间的分钟数
   * @param h2 指定完成核算时间的小时数
   * @param m2 指定完成核算时间的分钟数
   * @param idcs [[核酸点的ID值, 核酸检测点距离张三的距离,核酸检测点当前检测的人数]]
   */
  public static void getResult(int h1, int m1, int h2, int m2, int[][] idcs) {
    int start = h1 * 60 + m1;
    int end = h2 * 60 + m2;

    int[][] ans =
        Arrays.stream(idcs)
            .map(
                idc -> {
                  int id = idc[0];
                  int distance = idc[1];
                  int count = idc[2];

                  int money = distance * 10;
                  int road = distance * 10; // 花在路上的时间
                  int arrived = start + road; // 到达核酸检测点的时间

                  // 如果在8：00之前就赶到了，那么其实要等待到8:00才能排队，这里其实花费的时间应该包括等待的时间
                  if (arrived < 8 * 60) {
                    arrived = 8 * 60;
                    road = arrived - start;
                  }

                  // 出发时间，结束时间
                  int[] ran1 = {start, arrived};
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

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

    int h1 = sc.nextInt();
    int m1 = sc.nextInt();

    int h2 = sc.nextInt();
    int m2 = sc.nextInt();

    int n = sc.nextInt();

    int[][] idcs = new int[n][3];
    for (int i = 0; i < n; i++) {
      idcs[i][0] = sc.nextInt();
      idcs[i][1] = sc.nextInt();
      idcs[i][2] = sc.nextInt();
    }

    getResult(h1, m1, h2, m2, idcs);
  }

  /**
   * @param h1 当前时间的小时数
   * @param m1 当前时间的分钟数
   * @param h2 指定完成核算时间的小时数
   * @param m2 指定完成核算时间的分钟数
   * @param idcs [[核酸点的ID值, 核酸检测点距离张三的距离,核酸检测点当前检测的人数]]
   */
  public static void getResult(int h1, int m1, int h2, int m2, int[][] idcs) {
    int start = h1 * 60 + m1;
    int end = h2 * 60 + m2;

    int[][] ans =
        Arrays.stream(idcs)
            .map(
                idc -> {
                  int id = idc[0];
                  int distance = idc[1];
                  int count = idc[2];

                  int money = distance * 10;
                  int road = distance * 10; // 花在路上的时间
                  int arrived = start + road; // 到达核酸检测点的时间

                  // 如果在8：00之前就赶到了，那么其实要等待到8:00才能排队，这里其实花费的时间应该包括等待的时间
                  if (arrived < 8 * 60) {
                    arrived = 8 * 60;
                    road = arrived - start;
                  }

                  // 出发时间，结束时间
                  int[] ran1 = {start, arrived};

                  // 和[8:00, 10:00]的交集，每分钟净增2人
                  int[] ran2 = {8 * 60, 10 * 60};
                  int add1 = getIntersection(ran1, ran2);
                  if (add1 != -1) {
                    count += 2 * add1;
                  }

                  // 和[10:00, 12:00]的交集，每分钟净减1人
                  int[] ran3 = {10 * 60, 12 * 60};
                  int min1 = getIntersection(ran1, ran3);
                  if (min1 != -1) {
                    count -= min1;
                    count = Math.max(0, count); // 注意至多减到0
                  }

                  // 和[12:00, 14:00]的交集，每分钟净增9人
                  int[] ran4 = {12 * 60, 14 * 60};
                  int add2 = getIntersection(ran1, ran4);
                  if (add2 != -1) {
                    count += 9 * add2;
                  }

                  // 和[14:00, 20:00]的交集，每分钟净减1人
                  int[] ran5 = {14 * 60, 20 * 60};
                  int min2 = getIntersection(ran1, ran5);
                  if (min2 != -1) {
                    count -= min2;
                    count = Math.max(0, count); // 注意至多减到0
                  }

                  return new int[] {id, count + road, money};
                })
            .filter(arr -> arr[1] <= end)
            .sorted((a, b) -> a[1] != b[1] ? a[1] - b[1] : a[2] != b[2] ? a[2] - b[2] : a[0] - b[0])
            .toArray(int[][]::new);
```

```
                    count = Math.max(0, count); // 注意至多减到0
                }

                // 和[12:00, 14:00]的交集，每分钟净增9人
                int[] ran4 = {12 * 60, 14 * 60};
                int add2 = getIntersection(ran1, ran4);
                if (add2 != -1) {
                  count += 9 * add2;
                }

                // 和[14:00, 20:00]的交集，每分钟净减1人
                int[] ran5 = {14 * 60, 20 * 60};
                int min2 = getIntersection(ran1, ran5);
                if (min2 != -1) {
                  count -= min2;
                  count = Math.max(0, count); // 注意至多减到0
                }

                return new int[] {id, count + road, money};
              })
            .filter(arr -> arr[1] <= end)
            .sorted((a, b) -> a[1] != b[1] ? a[1] - b[1] : a[2] != b[2] ? a[2] - b[2] : a[0] - b[0])
            .toArray(int[][]::new);

    System.out.println(ans.length);
    for (int[] arr : ans) {
      System.out.println(arr[0] + " " + arr[1] + " " + arr[2]);
    }
  }

  // 获取交集长度，如果没有交集返回-1
  public static int getIntersection(int[] ran1, int[] ran2) {
    int s1 = ran1[0], e1 = ran1[1];
    int s2 = ran2[0], e2 = ran2[1];

    if (s1 < s2) {
      if (s2 >= e1) return -1;
      else return Math.min(e1, e2) - s2;
    } else if (s1 > s2) {
      if (s1 >= e2) return -1;
      else return Math.min(e1, e2) - s1;
    } else {
      return Math.min(e1, e2) - s1;
    }
  }
}
```

Python算法源码

```
# 输入获取
h1, m1 = map(int, input().split())
h2, m2 = map(int, input().split())
n = int(input())
idcs = [list(map(int, input().split())) for i in range(n)]


# 算法入口
def getResult(h1, m1, h2, m2, idcs):
    start = h1 * 60 + m1
    end = h2 * 60 + m2

    ans = []

    for id, dis, count in idcs:
        money = dis * 10
        road = dis * 10  # 花在路上的时间
        arrived = start + road  # 到达核酸检测点的时间

        # 如果在8：00之前就赶到了，那么其实要等待到8:00才能排队，这里其实花费的时间应该包括等待的时间
        if arrived < 8 * 60:
            arrived = 8 * 60
            road = arrived - start

        # 张三的[出发时间，到达时间]
        tmp = [start, arrived]

        # 和[8:00, 10:00]的交集，每分钟净增2人
        add1 = getIntersection(tmp, [8 * 60, 10 * 60])
        if add1 is not False:
            count += 2 * add1

        # 和[10:00, 12:00]的交集，每分钟净减1人
        min1 = getIntersection(tmp, [10 * 60, 12 * 60])
        if min1 is not False:
            count -= min1
            count = max(0, count)  # 注意至多减到0

        # 和[12:00, 14:00]的交集，每分钟净增9人
        add2 = getIntersection(tmp, [12 * 60, 14 * 60])
        if add2 is not False:
            count += 9 * add2

        # 和[14:00, 20:00]的交集，每分钟净减1人
```

Python算法源码

复制

```python
# 输入获取
h1, m1 = map(int, input().split())
h2, m2 = map(int, input().split())
n = int(input())
idcs = [list(map(int, input().split())) for i in range(n)]


# 算法入口
def getResult(h1, m1, h2, m2, idcs):
    start = h1 * 60 + m1
    end = h2 * 60 + m2

    ans = []

    for id, dis, count in idcs:
        money = dis * 10
        road = dis * 10  # 花在路上的时间
        arrived = start + road  # 到达核酸检测点的时间

        # 如果在8：00之前就赶到了，那么其实要等待到8:00才能排队，这里其实花费的时间应该包括等待的时间
        if arrived < 8 * 60:
            arrived = 8 * 60
            road = arrived - start

        # 张三的[出发时间，到达时间]
        tmp = [start, arrived]

        # 和[8:00, 10:00]的交集，每分钟净增2人
        add1 = getIntersection(tmp, [8 * 60, 10 * 60])
        if add1 is not False:
            count += 2 * add1

        # 和[10:00, 12:00]的交集，每分钟净减1人
        min1 = getIntersection(tmp, [10 * 60, 12 * 60])
        if min1 is not False:
            count -= min1
            count = max(0, count)  # 注意至多减到0

        # 和[12:00, 14:00]的交集，每分钟净增9人
        add2 = getIntersection(tmp, [12 * 60, 14 * 60])
        if add2 is not False:
            count += 9 * add2

        # 和[14:00, 20:00]的交集，每分钟净减1人
        min2 = getIntersection(tmp, [14 * 60, 20 * 60])
        if min2 is not False:
            count -= min2
            count = max(0, count)  # 注意至多减到0

        ans.append([id, count + road, money])

    ans = list(filter(lambda x: x[1] <= end, ans))
    ans.sort(key=lambda x: (x[1], x[2], x[0]))

    print(len(ans))
    for irm in ans:
        print(" ".join(map(str, irm)))


# 获取交集长度，如果没有交集返回false
def getIntersection(ran1, ran2):
    s1, e1 = ran1
    s2, e2 = ran2

    if s1 < s2:
        if s2 >= e1:
            return False
        else:
            return min(e1, e2) - s2
    elif s1 > s2:
        if s1 >= e2:
            return False
        else:
            return min(e1, e2) - s1
    else:
        return min(e1, e2) - s1


# 调用算法
getResult(h1, m1, h2, m2, idcs)
```