

## 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

 伏城之外 已关注  2   6  4  已订阅

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

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let h1, m1, h2, m2, n;
11 rl.on("line", (line) => {
12   lines.push(line);
13 })
```

```

1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let h1, m1, h2, m2, n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 3) {
15     [h1, m1] = lines[0].split(" ").map(Number);
16     [h2, m2] = lines[1].split(" ").map(Number);
17     n = lines[2].trim();
18   }
19
20   if (n && lines.length === n + 3) {
21     const idcs = lines.slice(3).map((line) => line.split(" ").map(Number));
22     getResult(h1, m1, h2, m2, idcs);
23     lines.length = 0;
24   }
25 });
26
27 /**
28  *
29  * @param {*} h1 当前时间的小时数
30  * @param {*} m1 当前时间的分钟数
31  * @param {*} h2 指定完成核算时间的小时数
32  * @param {*} m2 指定完成核算时间的分钟数
33  * @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]]
34  */
35 function getResult(h1, m1, h2, m2, idcs) {
36   const start = h1 * 60 + m1;
37   const end = h2 * 60 + m2;
38
39   const ans = idcs
40     .map((idc) => {
41       let [id, distance, count] = idc;
42       const money = distance * 10;
43       let road = distance * 10; // 花在路上的时间
44       let arrived = start + road; // 到达核酸检测点的时间
45
46       // 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
47       if (arrived < 8 * 60) {
48         arrived = 8 * 60;
49         road = arrived - start;
50       }
51
52       // [出发时间, 到达时间]
53       // 和[8:00, 10:00]的交集, 每分钟净增2人
54       const add1 = getIntersection([start, arrived], [8 * 60, 10 * 60]);
55       if (add1 !== false) {
56         count += 2 * add1;
57       }
58
59       // 和[10:00, 12:00]的交集, 每分钟净减1人
60       const min1 = getIntersection([start, arrived], [10 * 60, 12 * 60]);
61       if (min1 !== false) {
62         count -= min1;
63         count = Math.max(0, count); // 注意至多减到0
64       }
65
66       // 和[12:00, 14:00]的交集, 每分钟净增9人
67       const add2 = getIntersection([start, arrived], [12 * 60, 14 * 60]);
68       if (add2 !== false) {
69         count += 9 * add2;
70       }
71
72       // 和[14:00, 20:00]的交集, 每分钟净减1人
73       const min2 = getIntersection([start, arrived], [14 * 60, 20 * 60]);
74       if (min2 !== false) {
75         count -= min2;
76         count = Math.max(0, count); // 注意至多减到0
77       }
78
79       return [id, count + road, money];
80     })
81     .filter(([, real_end, money]) => real_end <= end)
82     .sort((a, b) =>
83       a[1] !== b[1] ? a[1] - b[1] : a[2] !== b[2] ? a[2] - b[2] : a[0] - b[0]
84     );
85
86   console.log(ans.length);
87   ans.forEach((irm) => {
88     console.log(irm.join(" "));
89   });
90 }
91
92 // 获取交集长度 前闭后开区间返回false

```

```

73     const min2 = getIntersection([start, arrived], [14 * 60, 20 * 60]);
74     if (min2 !== false) {
75         count -= min2;
76         count = Math.max(0, count); // 注意至多减到0
77     }
78
79     return [id, count + road, money];
80 })
81 .filter(([id, real_end, money]) => real_end <= end)
82 .sort((a, b) =>
83     a[1] !== b[1] ? a[1] - b[1] : a[2] !== b[2] ? a[2] - b[2] : a[0] - b[0]
84 );
85
86 console.log(ans.length);
87 ans.forEach(irm => {
88     console.log(irm.join(" "));
89 });
90 }
91
92 // 获取交集长度，如果没有交集返回false
93 function getIntersection(ran1, ran2) {
94     const [s1, e1] = ran1;
95     const [s2, e2] = ran2;
96
97     if (s1 < s2) {
98         if (s2 >= e1) return false;
99         else return e1 - s2;
100     } else if (s1 > s2) {
101         if (s1 >= e2) return false;
102         else return e2 - s1;
103     } else {
104         return Math.min(e1, e2) - s1;
105     }
106 }

```

#### Java算法源码

```

1  import java.util.Arrays;
2  import java.util.Scanner;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner sc = new Scanner(System.in);
7
8          int h1 = sc.nextInt();
9          int m1 = sc.nextInt();
10
11         int h2 = sc.nextInt();
12         int m2 = sc.nextInt();
13
14         int n = sc.nextInt();
15
16         int[][] idcs = new int[n][3];
17         for (int i = 0; i < n; i++) {
18             idcs[i][0] = sc.nextInt();
19             idcs[i][1] = sc.nextInt();
20             idcs[i][2] = sc.nextInt();
21         }
22
23         getResult(h1, m1, h2, m2, idcs);
24     }
25
26     /**
27      * @param h1 当前时间的小时数
28      * @param m1 当前时间的分钟数
29      * @param h2 指定完成核算时间的小时数
30      * @param m2 指定完成核算时间的分钟数
31      * @param idcs [(核点的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)]
32      */
33     public static void getResult(int h1, int m1, int h2, int m2, int[][] idcs) {
34         int start = h1 * 60 + m1;
35         int end = h2 * 60 + m2;
36
37         int[][] ans =
38             Arrays.stream(idcs)
39                 .map(
40                     idc -> {
41                         int id = idc[0];
42                         int distance = idc[1];
43                         int count = idc[2];
44
45                         int money = distance * 10;
46                         int road = distance * 10; // 花在路上时间
47                         int arrived = start + road; // 到达核酸检测点的时间
48
49                         // 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
50                         if (arrived < 8 * 60) {
51                             arrived = 8 * 60;
52                             road = arrived - start;
53                         }
54
55                         // 出发时间，结束时间
56                         int[] ran1 = {start, arrived};

```



伏城之外 已关注

👍 2



🌟 6



💬 4



专栏目录

已阅

```

1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int h1 = sc.nextInt();
9         int m1 = sc.nextInt();
10
11         int h2 = sc.nextInt();
12         int m2 = sc.nextInt();
13
14         int n = sc.nextInt();
15
16         int[][] idcs = new int[n][3];
17         for (int i = 0; i < n; i++) {
18             idcs[i][0] = sc.nextInt();
19             idcs[i][1] = sc.nextInt();
20             idcs[i][2] = sc.nextInt();
21         }
22
23         getResult(h1, m1, h2, m2, idcs);
24     }
25
26     /**
27      * @param h1 当前时间的小时数
28      * @param m1 当前时间的分钟数
29      * @param h2 指定完成核算时间的小时数
30      * @param m2 指定完成核算时间的分钟数
31      * @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]]
32      */
33     public static void getResult(int h1, int m1, int h2, int m2, int[][] idcs) {
34         int start = h1 * 60 + m1;
35         int end = h2 * 60 + m2;
36
37         int[][] ans =
38             Arrays.stream(idcs)
39                 .map(
40                     idc -> {
41                         int id = idc[0];
42                         int distance = idc[1];
43                         int count = idc[2];
44
45                         int money = distance * 10;
46                         int road = distance * 10; // 花在路上的时间
47                         int arrived = start + road; // 到达核酸检测点的时间
48
49                         // 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
50                         if (arrived < 8 * 60) {
51                             arrived = 8 * 60;
52                             road = arrived - start;
53                         }
54
55                         // 出发时间, 结束时间
56                         int[] ran1 = {start, arrived};
57
58                         // 和[8:00, 10:00]的交集, 每分钟净增2人
59                         int[] ran2 = {8 * 60, 10 * 60};
60                         int add1 = getIntersection(ran1, ran2);
61                         if (add1 != -1) {
62                             count += 2 * add1;
63                         }
64
65                         // 和[10:00, 12:00]的交集, 每分钟净减1人
66                         int[] ran3 = {10 * 60, 12 * 60};
67                         int min1 = getIntersection(ran1, ran3);
68                         if (min1 != -1) {
69                             count -= min1;
70                             count = Math.max(0, count); // 注意至多减到0
71                         }
72
73                         // 和[12:00, 14:00]的交集, 每分钟净增9人
74                         int[] ran4 = {12 * 60, 14 * 60};
75                         int add2 = getIntersection(ran1, ran4);
76                         if (add2 != -1) {
77                             count += 9 * add2;
78                         }
79
80                         // 和[14:00, 20:00]的交集, 每分钟净减1人
81                         int[] ran5 = {14 * 60, 20 * 60};
82                         int min2 = getIntersection(ran1, ran5);
83                         if (min2 != -1) {
84                             count -= min2;
85                             count = Math.max(0, count); // 注意至多减到0
86                         }
87
88                         return new int[] {id, count + road, money};
89                     })
90                 .filter(arr -> arr[1] <= end)
91                 .sorted((a, b) -> a[1] != b[1] ? a[1] - b[1] : a[2] != b[2] ? a[2] - b[2] : a[0] - b[0])
92                 .toArray(int[]::new);

```



```

70         count = Math.max(0, count); // 注意至多减到0
71     }
72
73     // 和[12:00, 14:00]的交集, 每分钟净增9人
74     int[] ran4 = {12 * 60, 14 * 60};
75     int add2 = getIntersection(ran1, ran4);
76     if (add2 != -1) {
77         count += 9 * add2;
78     }
79
80     // 和[14:00, 20:00]的交集, 每分钟净减1人
81     int[] ran5 = {14 * 60, 20 * 60};
82     int min2 = getIntersection(ran1, ran5);
83     if (min2 != -1) {
84         count -= min2;
85         count = Math.max(0, count); // 注意至多减到0
86     }
87
88     return new int[] {id, count + road, money};
89 })
90 .filter(arr -> arr[1] <= end)
91 .sorted((a, b) -> a[1] != b[1] ? a[1] - b[1] : a[2] != b[2] ? a[2] - b[2] : a[0] - b[0])
92 .toArray(int[1]::new);
93
94 System.out.println(ans.length);
95 for (int[] arr : ans) {
96     System.out.println(arr[0] + " " + arr[1] + " " + arr[2]);
97 }
98 }
99
100 // 获取交集长度, 如果没有交集返回-1
101 public static int getIntersection(int[] ran1, int[] ran2) {
102     int s1 = ran1[0], e1 = ran1[1];
103     int s2 = ran2[0], e2 = ran2[1];
104
105     if (s1 < s2) {
106         if (s2 >= e1) return -1;
107         else return Math.min(e1, e2) - s2;
108     } else if (s1 > s2) {
109         if (s1 >= e2) return -1;
110         else return Math.min(e1, e2) - s1;
111     } else {
112         return Math.min(e1, e2) - s1;
113     }
114 }
115 }

```

#### Python算法源码

```

1 # 输入获取
2 h1, m1 = map(int, input().split())
3 h2, m2 = map(int, input().split())
4 n = int(input())
5 idcs = [list(map(int, input().split())) for i in range(n)]
6
7
8 # 算法入口
9 def getResult(h1, m1, h2, m2, idcs):
10     start = h1 * 60 + m1
11     end = h2 * 60 + m2
12
13     ans = []
14
15     for id, dis, count in idcs:
16         money = dis * 10
17         road = dis * 10 # 花在路上的时间
18         arrived = start + road # 到达核酸检测点的时间
19
20         # 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
21         if arrived < 8 * 60:
22             arrived = 8 * 60
23             road = arrived - start
24
25         # 张三的[出发时间, 到达时间]
26         tmp = [start, arrived]
27
28         # 和[8:00, 10:00]的交集, 每分钟净增2人
29         add1 = getIntersection(tmp, [8 * 60, 10 * 60])
30         if add1 is not False:
31             count += 2 * add1
32
33         # 和[10:00, 12:00]的交集, 每分钟净减1人
34         min1 = getIntersection(tmp, [10 * 60, 12 * 60])
35         if min1 is not False:
36             count -= min1
37             count = max(0, count) # 注意至多减到0
38
39         # 和[12:00, 14:00]的交集, 每分钟净增9人
40         add2 = getIntersection(tmp, [12 * 60, 14 * 60])
41         if add2 is not False:
42             count += 9 * add2
43
44         # 和[14:00, 20:00]的交集, 每分钟净减1人

```



```

1 # 输入获取
2 h1, m1 = map(int, input().split())
3 h2, m2 = map(int, input().split())
4 n = int(input())
5 idcs = [list(map(int, input().split())) for i in range(n)]
6
7
8 # 算法入口
9 def getResult(h1, m1, h2, m2, idcs):
10     start = h1 * 60 + m1
11     end = h2 * 60 + m2
12
13     ans = []
14
15     for id, dis, count in idcs:
16         money = dis * 10
17         road = dis * 10 # 花在路上的时间
18         arrived = start + road # 到达核酸检测点的时间
19
20         # 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
21         if arrived < 8 * 60:
22             arrived = 8 * 60
23             road = arrived - start
24
25         # 张三的[出发时间, 到达时间]
26         tmp = [start, arrived]
27
28         # 和[8:00, 10:00]的交集, 每分钟净增2人
29         add1 = getIntersection(tmp, [8 * 60, 10 * 60])
30         if add1 is not False:
31             count += 2 * add1
32
33         # 和[10:00, 12:00]的交集, 每分钟净减1人
34         min1 = getIntersection(tmp, [10 * 60, 12 * 60])
35         if min1 is not False:
36             count -= min1
37             count = max(0, count) # 注意至多减到0
38
39         # 和[12:00, 14:00]的交集, 每分钟净增9人
40         add2 = getIntersection(tmp, [12 * 60, 14 * 60])
41         if add2 is not False:
42             count += 9 * add2
43
44         # 和[14:00, 20:00]的交集, 每分钟净减1人
45         min2 = getIntersection(tmp, [14 * 60, 20 * 60])
46         if min2 is not False:
47             count -= min2
48             count = max(0, count) # 注意至多减到0
49
50         ans.append([id, count + road, money])
51
52     ans = list(filter(lambda x: x[1] <= end, ans))
53     ans.sort(key=lambda x: (x[1], x[2], x[0]))
54
55     print(len(ans))
56     for irm in ans:
57         print(" ".join(map(str, irm)))
58
59
60 # 获取交集长度, 如果没有交集返回false
61 def getIntersection(ran1, ran2):
62     s1, e1 = ran1
63     s2, e2 = ran2
64
65     if s1 < s2:
66         if s2 >= e1:
67             return False
68         else:
69             return min(e1, e2) - s2
70     elif s1 > s2:
71         if s1 >= e2:
72             return False
73         else:
74             return min(e1, e2) - s1
75     else:
76         return min(e1, e2) - s1
77
78
79 # 调用算法
80 getResult(h1, m1, h2, m2, idcs)

```

复制