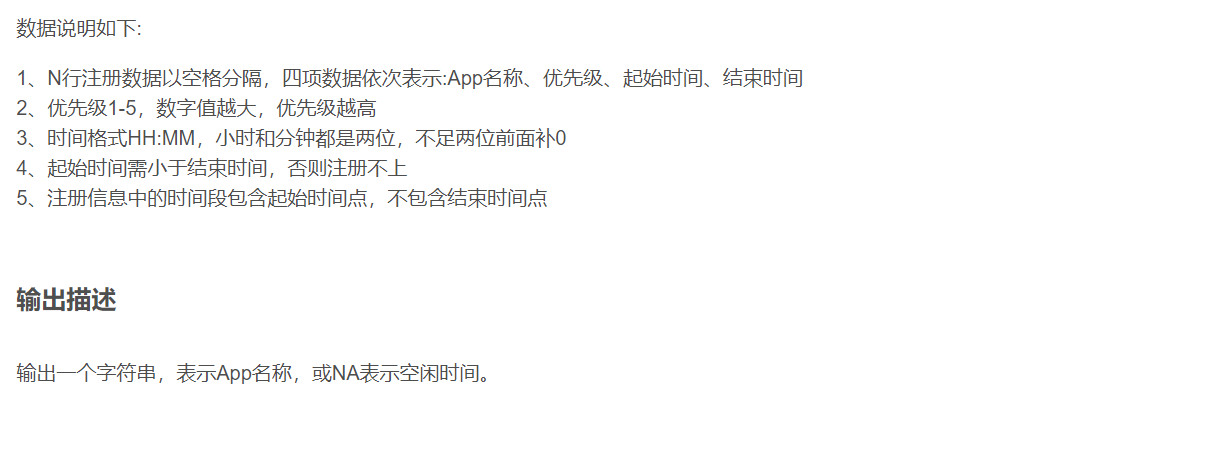
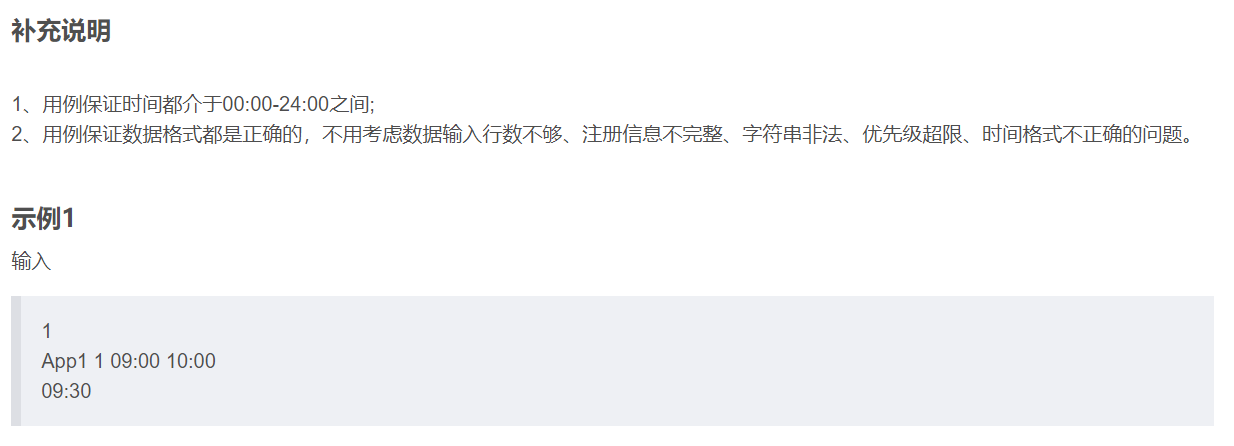
# **E卷-手机App防沉迷系统[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





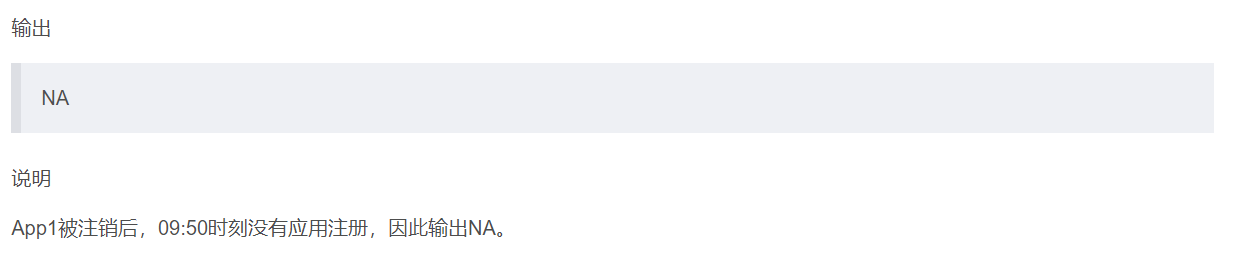














import java.util.\*;

class Slot {

String name;

int index;

Slot(String name, int index) {

this.name = name;

this.index = index;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取注册的App数

int n = scanner.nextInt();

scanner.nextLine(); // 处理换行符

Slot[] slots = new Slot[24 \* 60];

for (int i = 0; i < slots.length; i++) {

slots[i] = new Slot("NA", 0);

}

Map<Map.Entry<String, Integer>, Integer> priority = new HashMap<>();

List<int[]> timeRange = new ArrayList<>(Collections.nCopies(n, new int[2]));

// 读取所有注册App的信息

for (int i = 0; i < n; ++i) {

String app = scanner.next();

int prio = scanner.nextInt();

String start = scanner.next();

String end = scanner.next();

int startHour = Integer.parseInt(start.substring(0, 2));

int startMinute = Integer.parseInt(start.substring(3, 5));

int endHour = Integer.parseInt(end.substring(0, 2));

int endMinute = Integer.parseInt(end.substring(3, 5));

int startTime = startHour \* 60 + startMinute;

int endTime = endHour \* 60 + endMinute;

if (startTime >= endTime) continue; // 时间跨度非法，跳过

priority.put(new AbstractMap.SimpleEntry<>(app, i), prio);

timeRange.set(i, new int[]{startTime, endTime});

boolean conflict = false;

Set<Integer> conflictingIndexes = new HashSet<>();

for (int j = startTime; j < endTime; ++j) {

if (!slots[j].name.equals("NA")) {

// 检查是否有优先级高于当前注册的App

if (priority.get(new AbstractMap.SimpleEntry<>(slots[j].name, slots[j].index)) >= prio) {

conflict = true;

break;

} else {

conflictingIndexes.add(slots[j].index);

}

}

}

if (!conflict) {

for (int idx : conflictingIndexes) {

// 清除冲突的低优先级App的注册信息

int[] range = timeRange.get(idx);

for (int k = range[0]; k < range[1]; ++k) {

slots[k] = new Slot("NA", 0);

}

}

for (int j = startTime; j < endTime; ++j) {

slots[j] = new Slot(app, i); // 注册高优先级App

}

}

}

// 读取查询时间

scanner.nextLine(); // 跳过行末换行符

String queryTime = scanner.nextLine();

int queryHour = Integer.parseInt(queryTime.substring(0, 2));

int queryMinute = Integer.parseInt(queryTime.substring(3, 5));

int queryTotalMin = queryHour \* 60 + queryMinute;

// 输出查询结果

System.out.println(slots[queryTotalMin].name);

scanner.close();

}

}

// 1

// App1 1 09:00 10:00

// 09:30

// App1



# 读取输入的App数

n = int(input())

# 初始化一个长度为24\*60的列表arr表示一天中的每分钟状态，默认值是("NA", 0)，表示默认没有注册的App，优先级为0

arr = [("NA", 0)] \* (24 \* 60)

# 使用字典存储App的优先级

from collections import defaultdict

mp = defaultdict(int)

# 使用字典存储每个App的注册时间区间

fr = defaultdict(list)

# 解析每一行的App注册数据

for i in range(n):

a, b, c, d = input().split()

# 存储App的优先级

mp[(a, i)] = int(b)

# 转换起始时间

p = c.split(":")

time1 = int(p[0]) \* 60 + int(p[1])

# 转换结束时间

p = d.split(":")

time2 = int(p[0]) \* 60 + int(p[1])

# 如果开始时间不小于结束时间，则注册无效

if time1 >= time2:

continue

# 记录该App的注册时间段

fr[i].append(time1)

fr[i].append(time2)

# 标记是否可以成功注册

f = 0

# 记录将被注销的App的索引

st = set()

# 检查当前时间段内，是否有优先级更高的App注册

for j in range(time1, time2):

x, y = arr[j]

if x != "NA":

st.add(y)

if x != "NA" and mp[(x, y)] >= int(b):

f = 1

# 如果可以成功注册，即当前时间段内没有优先级更高的App

if f == 0:

# 注销优先级低的App

for x in st:

[lc, rc] = fr[x]

for j in range(lc, rc):

arr[j] = ("NA", 0)

# 注册当前的App

for j in range(time1, time2):

arr[j] = (a, i)

# 读取输入的查询时间并转换为分钟

p = input().split(":")

tt = int(p[0]) \* 60 + int(p[1])

# 输出查询时间点对应的App名称，如果在该时间点没有App注册，返回"NA"

print(arr[tt][0])



#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <set>

using namespace std;

// 定义每个时刻的信息

struct Slot {

string name;

int index;

};

int main() {

int n;

// 读取注册的App数

cin >> n;

vector<Slot> slots(24 \* 60, {"NA", 0});

map<pair<string, int>, int> priority;

vector<pair<int, int>> timeRange(n);

// 读取所有注册App的信息

for (int i = 0; i < n; ++i) {

string app, pp, start, end;

cin >> app >> pp >> start >> end;

int prio = stoi(pp);

int start\_hour = stoi(start.substr(0, 2));

int start\_minute = stoi(start.substr(3, 2));

int end\_hour = stoi(end.substr(0, 2));

int end\_minute = stoi(end.substr(3, 2));

int start\_time = start\_hour \* 60 + start\_minute;

int end\_time = end\_hour \* 60 + end\_minute;

if (start\_time >= end\_time) continue; // 时间跨度非法，跳过

priority[ {app, i}] = prio;

timeRange[i] = {start\_time, end\_time};

bool conflict = false;

set<int> conflictingIndexes;

for (int j = start\_time; j < end\_time; ++j) {

if (slots[j].name != "NA") {

// 检查是否有优先级高于当前注册的App

if (priority[ {slots[j].name, slots[j].index}] >= prio) {

conflict = true;

break;

} else {

conflictingIndexes.insert(slots[j].index);

}

}

}

if (!conflict) {

for (const auto& idx : conflictingIndexes) {

// 清除冲突的低优先级App的注册信息

for (int k = timeRange[idx].first; k < timeRange[idx].second; ++k) {

slots[k] = {"NA", 0};

}

}

for (int j = start\_time; j < end\_time; ++j) {

slots[j] = {app, i}; // 注册高优先级App

}

}

}

string queryTime;

getline(cin,queryTime);

getline(cin,queryTime);

int query\_hour = stoi(queryTime.substr(0, 2));

int query\_minute = stoi(queryTime.substr(3, 2));

int query\_total\_min = query\_hour \* 60 + query\_minute;

// 输出查询结果

cout << slots[query\_total\_min].name << endl;

return 0;

}

// 1

// App1 1 09:00 10:00

// 09:30

// App1

// 2

// App1 1 09:00 10:00

// App2 2 09:10 09:30

// 09:20

//App2



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 定义每个App注册信息的数据结构

typedef struct {

char name[20]; // App名称

int priority; // 优先级

} App;

// 定义每个时刻的信息

typedef struct {

char name[20];

int index;

} Slot;

int main() {

int n;

// 读取注册的App数

scanf("%d", &n);

// 使用何种方式在一天的分钟数空间划分，提高代码便捷

Slot slots[24 \* 60];

for (int i = 0; i < 24 \* 60; i++) {

strcpy(slots[i].name, "NA");

slots[i].index = 0;

}

App appRegistry[n];

int startTimes[n], endTimes[n]; // 一级通用时间转换为分钟

// 读取所有注册App的信息

for (int i = 0; i < n; i++) {

char app[20], start[6], end[6];

int priority;

scanf("%s %d %s %s", app, &priority, start, end);

int startHour, startMin, endHour, endMin;

sscanf(start, "%d:%d", &startHour, &startMin);

sscanf(end, "%d:%d", &endHour, &endMin);

int startTime = startHour \* 60 + startMin;

int endTime = endHour \* 60 + endMin;

if (startTime >= endTime) continue; // 时间跨度非法，跳过

strcpy(appRegistry[i].name, app);

appRegistry[i].priority = priority;

startTimes[i] = startTime;

endTimes[i] = endTime;

int conflict = 0;

int conflictingIndexes[n];

int conflictCount = 0;

for (int j = startTime; j < endTime; ++j) { // 检查冲突

if (strcmp(slots[j].name, "NA") != 0) {

if (appRegistry[slots[j].index].priority >= priority) {

conflict = 1;

break;

} else {

conflictingIndexes[conflictCount++] = slots[j].index;

}

}

}

if (!conflict) { // 无冲突，覆盖注册

for (int j = 0; j < conflictCount; ++j) {

for (int k = startTimes[conflictingIndexes[j]];

k < endTimes[conflictingIndexes[j]]; ++k) {

strcpy(slots[k].name, "NA");

slots[k].index = 0;

}

}

for (int j = startTime; j < endTime; ++j) {

strcpy(slots[j].name, app);

slots[j].index = i; // 记录注册的索引

}

}

}

char queryTime[6];

scanf("%s", queryTime); // 读取查询时间

int queryHour, queryMin;

sscanf(queryTime, "%d:%d", &queryHour, &queryMin);

int queryTotalMin = queryHour \* 60 + queryMin;

printf("%s\n", slots[queryTotalMin].name); // 输出查询结果

return 0;

}



const readline = require("readline");

// 建立读取输入的接口

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let lines = []; // 存储所有输入行

let currentLine = 0; // 当前处理的行数

// 读取输入的每一行

rl.on("line", (line) => {

lines.push(line);

});

// 读取注册的App数

rl.on("close", () => {

let n = parseInt(lines[currentLine++]);

// 使用数组记录一天的每一分钟的状态

let slots = Array(24 \* 60).fill({ name: "NA", index: 0 });

let priorityMap = new Map(); // 记录每个App的优先级

let timeRanges = []; // 记录每个App的时间范围

// 读取所有注册App的信息

for (let i = 0; i < n; i++) {

let [app, prio, start, end] = lines[currentLine++].split(" ");

let [startHour, startMinute] = start.split(":").map(Number);

let [endHour, endMinute] = end.split(":").map(Number);

let startTime = startHour \* 60 + startMinute;

let endTime = endHour \* 60 + endMinute;

if (startTime >= endTime) continue; // 时间跨度非法，跳过

priorityMap.set(`${app}-${i}`, parseInt(prio));

timeRanges.push([startTime, endTime]);

let conflict = false;

let conflictingIndexes = new Set();

for (let j = startTime; j < endTime; j++) {

if (slots[j].name !== "NA") {

if (

priorityMap.get(`${slots[j].name}-${slots[j].index}`) >=

parseInt(prio)

) {

conflict = true;

break;

} else {

conflictingIndexes.add(slots[j].index);

}

}

}

if (!conflict) {

for (let idx of conflictingIndexes) {

for (let k = timeRanges[idx][0]; k < timeRanges[idx][-1]; k++) {

slots[k] = { name: "NA", index: 0 };

}

}

for (let j = startTime; j < endTime; j++) {

slots[j] = { name: app, index: i };

}

}

}

let queryTime = lines[currentLine++];

let [queryHour, queryMinute] = queryTime.split(":").map(Number);

let queryTotalMin = queryHour \* 60 + queryMinute;

// 输出查询结果

console.log(slots[queryTotalMin].name);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 定义每个时间片段的信息

type Slot struct {

name string

index int

}

func main() {

scanner := bufio.NewScanner(os.Stdin)

scanner.Scan()

n, \_ := strconv.Atoi(scanner.Text()) // 读取注册的App数

// 初始化一天所有时间片段

slots := make([]Slot, 24\*60)

for i := range slots {

slots[i] = Slot{name: "NA", index: 0}

}

priority := make(map[string]int)

timeRange := make([][2]int, n)

// 读取所有注册App的信息

for i := 0; i < n; i++ {

scanner.Scan()

parts := strings.Split(scanner.Text(), " ")

app := parts[0]

prio, \_ := strconv.Atoi(parts[1])

start := parts[2]

end := parts[3]

startHour, \_ := strconv.Atoi(start[:2])

startMin, \_ := strconv.Atoi(start[3:])

endHour, \_ := strconv.Atoi(end[:2])

endMin, \_ := strconv.Atoi(end[3:])

startTime := startHour\*60 + startMin

endTime := endHour\*60 + endMin

if startTime >= endTime {

continue

}

priority[fmt.Sprintf("%s-%d", app, i)] = prio

timeRange[i] = [2]int{startTime, endTime}

conflict := false

conflictingIndexes := make(map[int]bool)

for j := startTime; j < endTime; j++ {

if slots[j].name != "NA" {

if priority[fmt.Sprintf("%s-%d", slots[j].name, slots[j].index)] >= prio {

conflict = true

break

} else {

conflictingIndexes[slots[j].index] = true

}

}

}

if !conflict {

// 清除低优先级的App时间段

for idx := range conflictingIndexes {

for k := timeRange[idx][0]; k < timeRange[idx][1]; k++ {

slots[k] = Slot{name: "NA", index: 0}

}

}

// 注册新App到时间段

for j := startTime; j < endTime; j++ {

slots[j] = Slot{name: app, index: i}

}

}

}

scanner.Scan()

queryTime := scanner.Text()

queryHour, \_ := strconv.Atoi(queryTime[:2])

queryMin, \_ := strconv.Atoi(queryTime[3:])

queryTotalMin := queryHour\*60 + queryMin

// 输出查询时间该时刻的App名字

fmt.Println(slots[queryTotalMin].name)

}