【华为OD机考 统一考试机试C卷】贪心歌手（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
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

题目描述

一个歌手准备从A城去B城参加演出。

1. 按照合同，他必须在 T 天内赶到
2. 歌手途经 N 座城市
3. 歌手不能往回走
4. 每两座城市之间需要的天数都可以提前获知。
5. 歌手在每座城市都可以在路边卖唱赚钱。

   经过调研，歌手提前获知了每座城市卖唱的收入预期： 如果在一座城市第一天卖唱可以赚M，后续每天的收入会减少D（第二天赚的钱是 M - D，第三天是 M - 2D …）。如果收入减少到 0 就不会再少了。
6. 歌手到达后的第二天才能开始卖唱。如果今天卖过唱，第二天才能出发。

贪心的歌手最多可以赚多少钱?

输入描述

第一行两个数字 T 和 N，中间用空格隔开。

- T 代表总天数，0 < T < 1000
- N 代表路上经过 N 座城市，0 < N < 100

第二行 N+1 个数字，中间用空格隔开。代表每两座城市之间耗费的时间。

- 其总和 ≤ T。

接下来 N 行，每行两个数字 M 和 D，中间用空格隔开。代表每个城市的输入预期。

- 0 < M < 1000
- 0 < D < 100

输出描述

一个数字。代表歌手最多可以赚多少钱。以回车结束。

用例

输入

```
10 2
1 1 2
120 20
90 10
```

输出

```
540
```

说明

> 总共10天，路上经过2座城市。
> 路上共花 1+1+2=4 天
> 剩余6天最好的计划是在第一座城市待3天，在第二座城市待3天。

> 在第一座城市赚的钱：120 + 100 + 80 = 300
> 在第二座城市赚的钱：90 + 80 + 70 = 240
> 一共 300 + 240 = 540。

解题思路

贪心算法

贪心算法在此问题中的应用

1. **最大化每天收益**：对于每个城市，根据其收入预期M和收入递减值D，可以计算出如果在该城市卖唱，每一天的收益是多少。随着时间的推移，这个城市的日收益会递减。在这个过程中，算法每天都会计算并尝试选择当天的最大可能收益。
2. **选择最佳卖唱天数**：由于总天数T是有限的，所以需要在所有可选的卖唱天数中挑选出最有利的组合。这里的贪心选择是通过优先队列（最小堆）来实现的。优先队列中存储的是当前选择的卖唱天数的收益。如果遇到一个新的天数其收益比队列中最小的收益要高，那么就替换掉这个最小收益的天数。这样，我们总是保持了收益最大化的天数组合。

具体 步骤

1. 初始化一个优先队列，用于记录每天的收益。优先队列的特性是，队列中的元素总是按照一定的顺序排列，这里是按照收益的大小排列。
2. 遍历每个城市，对于每个城市，计算从第一天开始每天的收益，并将其加入优先队列。
3. 如果优先队列的大小超过了剩余的天数，那么就取出优先队列中最小的收益，并与当天的收益进行比较。如果当天的收益更高，那么就将最小的收益移出队列，并将当天的收益加入队列。这样做的目的是，始终保持队列中的元素是最高的收益。
4. 当收益为0时，即当前城市的收益已经减少到0，那么就不再在该城市卖唱，跳出循环，继续处理下一个城市。
5. 最后，将优先队列中的所有收益相加，得到最大收益，并输出。

核心思想是贪心算法，即每一步都做出在当前看来最好的选择，最终得到的就是全局最优解。

C++

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

```cpp
using namespace std;

int main() {
    int T, N; // 分别表示总天数和城市数量
    cin >> T >> N;

    vector<int> travelDays(N + 1); // 存储每两座城市之间耗费的时间
    for (int i = 0; i <= N; i++) {
        cin >> travelDays[i];
    }

    vector<int> M(N), D(N); // M存储每个城市的收入预期，D存储每个城市的收入递减值
    for (int i = 0; i < N; i++) {
        cin >> M[i] >> D[i];
    }

    // 计算必须花费的路程时间
    int roadCost = 0;
    for (int i = 0; i <= N; i++) {
        roadCost += travelDays[i];
    }
    // 可用于卖唱赚钱的时间
    int remain = T - roadCost;

    // 使用优先队列记录每天的收益
    priority_queue<int, vector<int>, greater<int>> earningsQueue;

    // 遍历每个城市
    for (int i = 0; i < N; i++) {
        int days = 0; // 当前城市卖唱的天数
        while (true) {
            int profitToday = max(M[i] - days * D[i], 0); // 计算今天的收益
            if ((int)earningsQueue.size() < remain) {
                earningsQueue.push(profitToday);
            } else {
                if (!earningsQueue.empty() && profitToday > earningsQueue.top()) {
                    earningsQueue.pop(); // 移除最小收益
                    earningsQueue.push(profitToday); // 加入今天的收益
                }
            }

```

```
            if (profitToday == 0) break; // 如果收益为0，不再卖唱
            days++;
        }
    }

    // 计算总收益
    int maxEarnings = 0;
    while (!earningsQueue.empty()) {
        maxEarnings += earningsQueue.top();
        earningsQueue.pop();
    }

    cout << maxEarnings << endl; // 输出总收益
    return 0;
}
```

Java

```
import java.util.PriorityQueue;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt(); // 总天数
        int N = scanner.nextInt(); // 城市数量
        int[] travelDays = new int[N + 1]; // 每两座城市之间耗费的时间
        for (int i = 0; i <= N; i++) {
            travelDays[i] = scanner.nextInt();
        }
        int[] M = new int[N]; // 每个城市的收入预期M
        int[] D = new int[N]; // 每个城市的收入递减值D
        for (int i = 0; i < N; i++) {
            M[i] = scanner.nextInt();
            D[i] = scanner.nextInt();
        }
        scanner.close();

        // 计算必须花费的路程时间
        int roadCost = 0;
```

```
        for (int i = 0; i <= N; i++) {
            roadCost += travelDays[i];
        }
        // 可用于卖唱赚钱的时间
        int remain = T - roadCost;

        // 使用优先队列记录每天的收益
        PriorityQueue<Integer> earningsQueue = new PriorityQueue<>();

        // 遍历每个城市
        for (int i = 0; i < N; i++) {
            int days = 0; // 当前城市卖唱的天数
            while (true) {
                int profitToday = Math.max(M[i] - days * D[i], 0);
                if (earningsQueue.size() < remain) {
                    earningsQueue.add(profitToday);
                } else {
                    if (!earningsQueue.isEmpty() && profitToday > earningsQueue.peek()) {
                        earningsQueue.poll(); // 移除最小收益
                        earningsQueue.add(profitToday); // 加入今天的收益
                    }
                }
                if (profitToday == 0) break; // 如果收益为0，不再卖唱
                days++;
            }
        }

        // 计算总收益
        int maxEarnings = 0;
        while (!earningsQueue.isEmpty()) {
            maxEarnings += earningsQueue.poll();
        }

        System.out.println(maxEarnings);
    }
}
```

javaScript

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

let lines = [];
rl.on('line', (line) => {
  lines.push(line);
}).on('close', () => {
  const [T, N] = lines[0].split(' ').map(Number);
  const travelDays = lines[1].split(' ').map(Number);
  const M = [];
  const D = [];
  for (let i = 2; i < 2 + N; i++) {
    const [m, d] = lines[i].split(' ').map(Number);
    M.push(m);
    D.push(d);
  }

  // 计算必须花费的路程时间
  const roadCost = travelDays.reduce((acc, val) => acc + val, 0);
  // 可用于卖唱赚钱的时间
  const remain = T - roadCost;

  // 记录每天的收益，不使用优先队列
  let earnings = [];

  // 遍历每个城市
  for (let i = 0; i < N; i++) {
    let days = 0; // 当前城市卖唱的天数
    while (true) {
      // 计算今天的收益，如果收益小于0，则按0计算
      const profitToday = Math.max(M[i] - days * D[i], 0);
      // 如果收益数组的大小小于剩余天数，直接添加今天的收益
      if (earnings.length < remain) {
        earnings.push(profitToday);
      } else {
        // 找到收益数组中最小的收益并替换
        const minEarningIndex = earnings.indexOf(Math.min(...earnings));
```

```
        if (earnings[minEarningIndex] < profitToday) {
          earnings[minEarningIndex] = profitToday;
        }
      }
      // 如果今天的收益为0，结束循环
      if (profitToday === 0) {
        break;
      }
      days++;
    }
  }

  // 计算总收益
  const maxEarnings = earnings.reduce((acc, val) => acc + val, 0);

  // 输出总收益
  console.log(maxEarnings);
});
```

Python

```
import heapq

# 读取输入
T, N = map(int, input().split())  # 总天数和城市数量
travel_days = list(map(int, input().split()))  # 每两座城市之间耗费的时间
M = []  # 每个城市的收入预期M
D = []  # 每个城市的收入递减值D
for _ in range(N):
    m, d = map(int, input().split())
    M.append(m)
    D.append(d)

# 计算必须花费的路程时间
road_cost = sum(travel_days)
# 可用于卖唱赚钱的时间
remain = T - road_cost

# 使用优先队列（最小堆）记录每天的收益
earnings_queue = []
```

```python

# 遍历每个城市
for i in range(N):
    days = 0  # 当前城市卖唱的天数
    while True:
        # 计算今天的收益，如果收益小于0，则按0计算
        profit_today = max(M[i] - days * D[i], 0)
        # 如果优先队列的大小小于剩余天数，直接添加今天的收益
        if len(earnings_queue) < remain:
            heapq.heappush(earnings_queue, profit_today)
        else:
            # 如果今天的收益大于优先队列中的最小收益，则替换
            if earnings_queue and profit_today > earnings_queue[0]:
                heapq.heappop(earnings_queue)
                heapq.heappush(earnings_queue, profit_today)
        # 如果今天的收益为0，结束循环
        if profit_today == 0:
            break
        days += 1

# 计算总收益
max_earnings = sum(earnings_queue)

# 输出总收益
print(max_earnings)
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>

// 获取输入中的整数数组
int* getIntArray(char* line, int* size) {
    int* array = (int*)malloc(sizeof(int) * 100); // 假设最多100个整数
    int num, i = 0;
    while (sscanf(line, "%d", &num) > 0) {
        array[i++] = num;
        // 移动到下一个整数
        while (*line != ' ' && *line != '\0') line++;
        while (*line == ' ') line++;
```

```
    }
    *size = i;
    return array;
}

// 计算数组中所有元素的和
int sumArray(int* array, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += array[i];
    }
    return sum;
}

// 找到数组中的最小值的索引
int findMinIndex(int* array, int size) {
    int minIndex = 0;
    for (int i = 1; i < size; i++) {
        if (array[i] < array[minIndex]) {
            minIndex = i;
        }
    }
    return minIndex;
}

int main() {
    char line[1024]; // 假设每行输入不超过1024个字符
    int T, N;
    int* travelDays;
    int* M;
    int* D;
    int travelDaysSize, MSize, DSize;
    int roadCost, remain;
    int* earnings;
    int earningsSize = 0;

    // 读取第一行输入，获取总天数T和城市数量N
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d", &T, &N);
```

```
    // 读取第二行输入，获取每两座城市之间耗费的时间
    fgets(line, sizeof(line), stdin);
    travelDays = getIntArray(line, &travelDaysSize);

    // 分配内存空间给M和D数组
    M = (int*)malloc(sizeof(int) * N);
    D = (int*)malloc(sizeof(int) * N);

    // 读取后续行输入，获取每个城市的收入预期M和收入递减值D
    for (int i = 0; i < N; i++) {
        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d %d", &M[i], &D[i]);
    }

    // 计算必须花费的路程时间
    roadCost = sumArray(travelDays, travelDaysSize);
    // 可用于卖唱赚钱的时间
    remain = T - roadCost;

    // 初始化收益数组
    earnings = (int*)malloc(sizeof(int) * remain);

    // 遍历每个城市
    for (int i = 0; i < N; i++) {
        int days = 0; // 当前城市卖唱的天数
        while (1) {
            // 计算今天的收益，如果收益小于0，则按0计算
            int profitToday = M[i] - days * D[i];
            if (profitToday < 0) profitToday = 0;

            // 如果收益数组的大小小于剩余天数，直接添加今天的收益
            if (earningsSize < remain) {
                earnings[earningsSize++] = profitToday;
            } else {
                // 找到收益数组中最小的收益并替换
                int minEarningIndex = findMinIndex(earnings, earningsSize);
                if (earnings[minEarningIndex] < profitToday) {
                    earnings[minEarningIndex] = profitToday;
                }
            }

```

```
            // 如果今天的收益为0，结束循环
            if (profitToday == 0) {
                break;
            }
            days++;
        }
    }

    // 计算总收益
    int maxEarnings = sumArray(earnings, earningsSize);

    // 输出总收益
    printf("%d\n", maxEarnings);

    // 释放内存
    free(travelDays);
    free(M);
    free(D);
    free(earnings);

    return 0;
}
```

文章目录

C语言