Студентска опашка

locked

Problem | Submissions | Leaderboard | Discussions

В рамките на тази задача ще трябва да реализирате "Студентска опашка" – структура, която максимално наподобява случващото се пред студентски стол. Когато нов студент се нарежда на опашката, вместо да застане накрая й, той първо претърсва за свои "познати" в нея и ако открие такива се включва точно зад тях. Ако не намери, студентът е нямал късмет и застава накрая на опашката.

Изключването става по стандартния начин – само от началото на опашката.

Считаме, че времето отнемащо за обслужване на един студент е 2мин, а пък нов студент се нарежда на опашката на всяка минута. За определеност, в съвпадащите моменти първо се нарежда новият студент и чак след това първият напуска опашката.

Всеки студент има име - уникален низ от главни и малки латински букви(без интервали) и уникален номер на група - цяло положително число. **Студентите с еднакъв номер на група определяме като "познати".**

Input Format

На първия ред са зададено две числа – **N** и **M** - броят на студентите и броят на групите.

Следват **N** реда с информация за всеки един от студентите. Всеки ред съдържа името и групата на текущия студент. Считаме, че те идват при опашката точно в реда, в който са въведени.

Constraints

$1 \le N \le 100000$

$1 \le M \le 100000$

Имената на студентите са с дължина между 1 и 30 символа.

Output Format

Програмата ви трябва да изведе на екрана точния ред, в който студентите ще излязат от опашката. За всеки студент изведете по един ред, съдържащ неговото име, времето, в което се е наредил на опашката и времето, в което е излязъл.

Sample Input 0

```
8 666
Ivan 10
Nikolay 10
Vasil 3
Daniel 4
Yoanna 3
Maria 3
Pesho 666
Gosho 10
```

Sample Output 0

```
Ivan 0 2
Nikolay 1 4
Vasil 2 6
Yoanna 4 8
Maria 5 10
Daniel 3 12
Pesho 6 14
Gosho 7 16
```

Submissions: 94
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
struct Student{
    string name;
    int group;
    int time;
};
int lastSameElement(vector<Student> vec,int toSearch){
int result=vec.size()-1;
    for(int i=0;i<vec.size();i++){
        if(vec[i].group==toSearch){
            result=i;
        }
    }
    return result+1;
}
class studentsQueue{
    vector<Student> queue;
    public:
    void push(Student student){
        queue.insert(queue.begin()+lastSameElement(queue,student.group),student);
    }
};
int main() {
    int studentsNumber;
    cin>>studentsNumber;
    int groupsNumber;
    cin>>groupsNumber;
    vector<Student> queue;
    vector<Student> outQueue;
    int time=2;
    for(int i=1;i<=studentsNumber;i++){
        Student curStudent;
        cin>>curStudent.name;
        cin>>curStudent.group;
        curStudent.time=i-1;
        queue.insert(queue.begin()+lastSameElement(queue,curStudent.group),curStudent);
        if(i==3||(i%2!=0&&i!=1)){
            cout<<queue[0].name<<" "<<queue[0].time<<" "<<time<<endl;
        time+=2;
            queue.erase(queue.begin());
        }
    }


    for(int i=0;i<queue.size();i++){
        cout<<queue[i].name<<" "<<queue[i].time<<" "<<time<<endl;
        time+=2;
    }
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code