

# II (районний) етап Всеукраїнської учнівської олімпіади з інформатики

Київ, 19 листопада 2017 р.

---

Максимальна оцінка за кожну з чотирьох задач — 100 балів.

Для всіх задач обмеження на час — 1 секунда / тест; обмеження на пам'ять — 256 МБ.

Матеріали олімпіади буде оприлюднено на сайті [kievoi.ippo.kubg.edu.ua](http://kievoi.ippo.kubg.edu.ua), а також на [soi.org.ua](http://soi.org.ua).

Автор задач — Данило Мисак.

---

## 1. Процесор (назва програми: `cpu.cpp` / `cpu.pas` / `cpu.*`)

Основним компонентом кожного комп'ютера є процесор — чип, що безпосередньо виконує усі операції, які потрібні для роботи програми. Уявіть, що вашій програмі необхідно виконати рівно  $n$  операцій, а процесор працює зі швидкістю  $m$  операцій за секунду. Підрахуйте, за скільки повних секунд процесор зможе виконати всі операції.

### Вхідні дані

У єдиному рядку вхідного файлу задано натуральні числа  $n$  та  $m$ , причому обидва числа менші за  $10^9$ .

### Вихідні дані

У вихідний файл виведіть кількість повних секунд, після яких виконання  $n$  операцій буде завершеним.

### Приклади

Вхідний файл <code>cpu.in</code>	Вихідний файл <code>cpu.out</code>
35 5	7
52 7	8

### Коментарі до прикладів

У першому прикладі за 7 секунд процесор виконає рівно  $7 \times 5 = 35$  операцій. У другому прикладі за 7 секунд процесор виконає  $7 \times 7 = 49$  операцій, а виконання решти трьох операцій завершить упродовж восьмої секунди.

### Історична довідка

Перші універсальні процесори на кшталт сучасних почали з'являтися у 1960—70-х роках завдяки зусиллям американських та японських інженерів. Комп'ютери існували й до того, але кожна модель могла виконувати лише свій унікальний набір інструкцій, спрямованих на вирішення конкретного кола задач.

## 2. Миша (назва програми: `mouse.cpp` / `mouse.pas` / `mouse.*`)

Одним із базових засобів для взаємодії з комп'ютером є миша та подібні до неї пристрої. Найчастіше миші мають по дві кнопки — ліву та праву, — а різноманітні дії можна виконати за допомогою таких операцій, як затискання/відпускання лівої кнопки, затискання/відпускання правої кнопки або ж подвійне натискання лівої кнопки миші. Визначимо останню операцію формально: подвійним натисканням назвемо послідовність із трьох дій — затискання лівої кнопки, відпускання лівої кнопки та ще одного затискання лівої кнопки

миші — за умови, що всі три дії було виконано протягом півсекунди, вони не переривалися іншими операціями з мишею (тобто затисканням правої кнопки), а на момент перед виконанням даних трьох дій жодна з кнопок миші не була затиснена.

Ваше завдання — за послідовністю дій із мишею встановити, скільки разів було здійснено подвійне натискання лівої кнопки.

### Вхідні дані

У першому рядку вхідного файлу вказано кількість дій  $n$ ; вона є натуральним числом та не перевищує 1000. У наступних  $n$  рядках задано самі дії:

1. Перше число — тип дії: 1 для затискання лівої кнопки, 2 для відпускання лівої кнопки; 3 для затискання правої кнопки, 4 для відпускання правої кнопки.
2. Друге число — часова мітка дії — момент часу в секундах, у який було зроблено дію. Часова мітка є додатним числом, і для нього вказано рівно три цифри після крапки, яка розділяє цілу та дробову частини числа.

Дії задано в порядку зростання часу, причому жодні дві часові мітки не збігаються. Початково (у нульовий момент часу) жодна з кнопок миші не затиснена. Остання часова мітка у файлі менша за 1000 секунд. Дії задано коректно: наприклад, між двома діями затискання однієї й тієї ж кнопки обов'язково знайдеться дія її відпускання, а між двома діями відпускання однієї кнопки — дія її затискання.

### Вихідні дані

У вихідний файл виведіть кількість подвійних натискань лівої кнопки миші.

### Приклади

Вхідний файл <code>mouse.in</code>	Вихідний файл <code>mouse.out</code>
7 1 0.500 2 0.994 1 1.000 2 5.000 1 8.500 2 8.994 1 9.001	1
15 1 2.345 2 2.346 1 2.347 2 2.348 1 2.349 2 2.350 1 12.345 2 12.346 3 12.347 1 12.348 2 12.349 4 12.350 1 22.345 2 22.346 1 22.347	2

### Коментарі до прикладів

У першому прикладі трійка дій між мітками 8.500 та 9.001 не є подвійним натисканням, бо виконується більше ніж півсекунди. У другому прикладі трійка дій між мітками 2.347 та 2.349 не є подвійним натисканням, бо затискання лівої кнопки на 2.347 є фінальною дією попереднього подвійного натискання.

### Історична довідка

Мишу було винайдено у середині 1960-х років. Спершу її винахідник, американець Дуглас Енгельбарт, називав свій пристрій «жуком», але дріт, який був приєднаний до пристрою з заднього боку, однозначно нагадував мишачий хвіст і змусив інженера змінити думку.

## 3. Клавіатура (назва програми: `keyboard.cpp` / `keyboard.pas` / `keyboard.*`)

Мабуть, основним засобом введення інформації в комп'ютер є клавіатура. Користувачу стає вкрай незручно, якщо клавіатура псується і час від часу не передає комп'ютеру інформацію про клавіші, які користувач натискає. На жаль, з однією клавіатурою трапилося саме таке: інколи клавіатура відключається та ігнорує натискання клавіш; через одне або кілька натискань вона вмикається знову і працює в звичайному режимі.

Ваше завдання — за початковим текстом, який вводив користувач, та остаточним варіантом цього тексту, який опинився на комп'ютері, визначити, яку найменшу кількість разів клавіатура могла відключатися під час друку.

### Вхідні дані

У першому рядку вхідного файлу записано початковий текст, який вводив користувач: текст складається не більш ніж із 5000 малих літер латинської абетки та не містить жодних інших символів (зокрема й пробілів). У другому рядку записано текст, що в підсумку опинився на комп'ютері. Обидва тексти непорожні, і другий текст дійсно міг утворитися з першого.

### Вихідні дані

У вихідний файл виведіть ціле число — найменшу кількість разів, які могла відключатися клавіатура у процесі друку.

### Приклади

Вхідний файл <code>keyboard.in</code>	Вихідний файл <code>keyboard.out</code>
helloworld hello	1
helloworld lord	3

### Історична довідка

Клавіатура є одним із найстарших компонентів комп'ютера: її прототипами були телетайпи та навіть звичайні друкарські машинки, що існували вже два століття тому.

## 4. Монітор (назва програми: `monitor.cpp` / `monitor.pas` / `monitor.*`)

Перед монітором стоїть достатньо непроста задача: щосекунди по кілька десятків разів йому необхідно виводити зображення, що складається з величезної кількості пікселів по горизонталі та по вертикалі, а кожен піксель при цьому може бути зафарбований в один з мільйонів кольорів.

У цій задачі ви маєте сформулювати лише один кадр, який виведе монітор. Екран монітора має прямокутний розмір. На екран потрібно в заданому порядку вивести фігури, які можуть частково або повністю перекривати одна одну: усі фігури — прямокутники однакового розміру та однакової орієнтації, і кожен прямокутник цілком зафарбовано в один і той самий колір.

### Вхідні дані

У першому рядку вхідного файлу вказано ширину  $m$  та висоту  $n$  екрана в пікселях; обидва числа натуральні та не перевищують 1000. У другому рядку вказано ширину  $w$  та висоту  $h$  у пікселях одного прямокутника; розміри прямокутника також є натуральними числами та не перевищують відповідних розмірів екрана. У третьому рядку задано кількість прямокутників  $k$  — натуральне число, не більше за  $3 \times 10^5$ . У наступних  $k$  рядках містяться по три цілих числа, і кожна трійка задає відповідний прямокутник:

1. Перше число — відстань у пікселях від лівої межі екрана до лівої межі прямокутника.
2. Друге число — відстань у пікселях від верхньої межі екрана до верхньої межі прямокутника.
3. Третє число — колір усіх  $w \times h$  пікселів прямокутника — невід'ємне ціле число, менше за  $10^6$ .

Усі  $k$  прямокутників повністю вміщаються на екрані, а задано їх у такому порядку, що наступні прямокутники, якщо накладаються на котрісь із попередніх, відображаються поверх них. Усі непокриті пікселі екрана, якщо такі є, мають колір 0.

### Вихідні дані

У вихідний файл виведіть  $n$  рядків по  $m$  чисел у кожному так, щоб числа задавали кольори відповідних пікселів на екрані монітора.

### Приклад

Вхідний файл <code>monitor.in</code>	Вихідний файл <code>monitor.out</code>
5 3	5 0 0 0 0
3 2	3 3 3 0 0
4	3 3 3 6 0
0 0 5	
1 1 6	
1 0 0	
0 1 3	

### Історична довідка

З англійської мови «монітор» — засіб спостереження. У перших комп'ютерах моніторами служили набори ламп, що почергово світилися і сигналізували таким чином про стан виконання програми. Зрештою інженери зрозуміли, що телевізійний екран значно краще підходить для цих цілей.

# Ідеї розв'язання

## 1. Процесор

Відповідь — округлена у більший бік частка від ділення  $n$  на  $m$ . Її можна отримати, наприклад, за формулою  $\left\lceil \frac{n+m-1}{m} \right\rceil$ , де квадратні дужки позначають найбільше ціле число, що не перевищує дане. У мові Pascal цю формулу найпростіше записати так:  $(n+m-1) \text{ div } m$ .

Крім того, слід врахувати, що двобайтових змінних (`integer` у мові Pascal) може не вистачити для збереження значень  $n$  на  $m$ .

## 2. Миша

Поступово зчитуючи рядки файлу, будемо запам'ятовувати три останні дії, а також те, чи є на даний момент затисненою права кнопка миші. Якщо трьома останніми діями були затискання, відпускання і знову затискання лівої кнопки, між першим і другим затисканням пройшло не більше півсекунди, а права кнопка на даний момент не затиснена, додаємо одиницю до лічильника подвійних натискань. При цьому очищуємо набір із трьох збережених дій, щоб помилково не зарахувати останнє затискання лівої кнопки за перше натискання в наступному подвійному.

Для роботи з часовими мітками достатньо використовувати змінні типу `double`.

## 3. Клавіатура

Задачу можна розв'язати за допомогою підходу, що називається *динамічним програмуванням*. Нехай довжина початкового тексту —  $n$ , а отриманого —  $m$ . Через  $f(i, j, +)$ , де  $i$  та  $j$  — натуральні числа, що не перевищують  $n$  та  $m$  відповідно, позначимо мінімальну кількість відключень клавіатури після друку перших  $i$  символів тексту за умови, що перші  $i$  символів початкового тексту перейшли в перші  $j$  символів остаточного тексту, причому  $i$ -й символ перейшов у  $j$ -й. Якщо такого бути не могло (наприклад, у випадку, коли  $i$ -й символ початкового тексту та  $j$ -й символ отриманого тексту різні), домовимося, що значення  $f(i, j, +)$  дорівнює нескінченності, яку на практиці можна замінити просто дуже великим числом.

Через  $f(i, j, -)$  позначимо мінімальну кількість відключень після друку перших  $i$  символів тексту за умови, що перші  $i$  символів початкового тексту перейшли в перші  $j$  символів остаточного тексту, але  $i$ -й символ не перейшов у  $j$ -й, а випав. У випадку, коли цього статися не могло, аналогічним чином уважатимемо дане значення рівним нескінченності.

Відповіддю до задачі буде число  $\min \{f(n, m, +), f(n, m, -)\}$ , а щоб його підрахувати, визначатимемо значення величин  $f(i, j, +)$  та  $f(i, j, -)$  у порядку збільшення індексів  $i$  та  $j$ . У цьому допоможуть такі *рекурентні формули*:

$$f(i, j, +) = \begin{cases} \min \{f(i-1, j-1, +), f(i-1, j-1, -)\}, & T[i] = t[j] \\ \infty, & T[i] \neq t[j] \end{cases}$$

$$f(i, j, -) = \min \{f(i-1, j, +) + 1, f(i-1, j, -)\}$$

(через  $T[i]$  і  $t[j]$  позначено  $i$ -й символ початкового тексту та  $j$ -й символ отриманого тексту відповідно). При цьому домовимося про такі початкові значення для  $i = 0$  або  $j = 0$ :

$$f(0, 0, +) = 0$$

$$f(i, 0, +) = f(0, j, +) = \infty, \quad i > 0, j > 0$$

$$f(i, 0, -) = 1, \quad i > 0$$

$$f(0, j, -) = \infty, \quad j \geq 0$$

Складність алгоритму —  $O(nm)$ .

## 4. Монітор

Задачу, звичайно, не так складно розв'язати, розглядаючи по порядку прямокутники й безпосередньо присвоюючи усім пікселям, що їм належать, відповідні кольори. Це потребує часу  $O(kwh)$  та дозволяє набрати лише частину балів.

Побудуємо ефективніший розв'язок. Спершу розв'яжемо задачу для  $n = h = 1$ , тобто для випадку, коли екран складається з одного ряду пікселів. Запишімо спершу в кожен піксель, що є першим (лівим) пікселем принаймні одного з  $k$  прямокутників, колір і порядковий номер цього прямокутника; якщо деякий піксель є лівим одразу для кількох прямокутників, запишемо в нього інформацію лише про той прямокутник, що має найбільший порядковий номер (він перекриє решту). Тепер будемо йти по екрану зліва направо, перебираючи його пікселі від першого до останнього і підтримуючи в кожен момент часу список прямокутників, які лежать під даним пікселем, причому в порядку від найвищого прямокутника до найглибшого. Прямокутники, права частина яких ніколи не «визирне» з-під інших прямокутників набору, можна зі списку видаляти, адже на фінальне розфарбування пікселів вони не вплинуть. Оновлювати даний список можна ефективно: перейшовши від попереднього пікселя до наступного, відкинемо зверху списку прямокутники, останнім пікселем яких був попередній; якщо в наступному пікселі починається новий прямокутник, то додамо цей прямокутник до списку ззаду, видаливши за потреби з кінця списку усі ті прямокутники, що опиняться під ним. У процесі проходження екрана дістанемо й відповідь для кожного пікселя — це колір найвищого на даний момент прямокутника зі списку.

Розв'язок задачі для двовимірного випадку — коли екран складається більше ніж з одного рядка пікселів — полягає у комбінюванні наведеного вище підходу із самим собою. Фінальною частиною повного алгоритму буде проходження окремо кожного рядка екрана зліва направо якраз у той спосіб, який наведено вище. Але для того, щоб сформувати для кожного рядка масив з інформацією про те, який прямокутник починається у кожному його пікселі, спершу застосуємо той самий підхід до стовпців: для кожного прямокутника запишемо його колір і порядковий номер у лівий верхній піксель даного прямокутника, після чого пройдемося по кожному стовпцю зверху вниз і запишемо в кожен піксель екрана колір і порядковий номер найвищого прямокутника, ліва межа якого проходить по даному пікселю.

Час виконання наведеного алгоритму є лінійним відносно сумарного розміру вхідних та вихідних даних; складність алгоритму —  $O(k + mn)$ .