# Задача 1 — Сумирай до 13

Дават ни се три числа **a**, **b** и **c**. Отпечатайте дали трите числа могат да се **сумират до** **13**, като единственото нещо което можем да правим е да сменяме знака пред тях.

### Вход

* Входът се чете от конзолата, на първият ред ще получите числата **a**, **b** и **c**, разделени със интервали.

### Изход

* Отпечатайте "**Yes**" ако числата могат да се сумират до 13, или "**No**" ако не могат.

### Ограничения

* **а**, **b** и **c** ще са цели числа в диапазона **[-1 000 000…1 000 000]**.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| ****Вход**** | ****Изход**** | ****Коментари**** |
| **-10 2 -1** | **Yes** | Сменяме знаците на -10 и -1, за да получим 10 + 2 + 1 = 13 съответно изписваме "Yes" |

|  |  |  |
| --- | --- | --- |
| ****Вход**** | ****Изход**** | ****Коментари**** |
| **-1 10 -1** | **No** | Независимо как сменяме знаците, няма как да получим 13 съответно изписваме "No" |

# Задача 2 — Нарисувайте тиква за Хелоуин

Дадено ни е число **n**. Нарисувайте хелоуинската тиква с размер **n** като в примерите долу.

### Вход

* Входът се чете от конзолата и съдържа един единствен ред, съдържащ числото **n**.

### Изход

* Отпечатайте тиква с размер **n** на конзолата.

### Ограничения

* Числото **n** е цяло число в диапазона **[3…100]**.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |
| --- | --- |
| ****Вход**** | ****Изход**** |
| **4** | **...\_/\_...**  **/..^,^..\**  **|.......|**  **\..\\_/../** |

|  |  |
| --- | --- |
| ****Вход**** | ****Изход**** |
| **5** | **....\_/\_....**  **/...^,^...\**  **|.........|**  **|.........|**  **\...\\_/.../** |

# Задача 3 — Дублирани букви

Дава ни се текстовия низ **S**. Целта на задачата е да се премахнат всички **специални дубликати** в него. **Специален дубликат** казваме на две едни и същи поредни букви (например: aa; qq). Премахването на един **специален дубликат** се счита за **една операция**. След като премахнете всички възможни специални дубликати от низа S, отпечатайте остатъчния низ и необходимия брой операции, за да се стигне до него, като ако остатъчният низ е празен, изпечатваме **"Empty String"** вместо него.

### Вход

* Входът се чете от конзолата, на първия ред получаваме низът **S** за обработка.

### Изход

* На първия ред трябва да се отпечата остатъчния низ **S** след премахването на специалните дубликати в него.
* На втория ред трябва да се отпечата броя операции които сме извършили за да стигнем до крайния резултат в следната форма "**N** operations", където **N** е броят операции.

### Ограничения

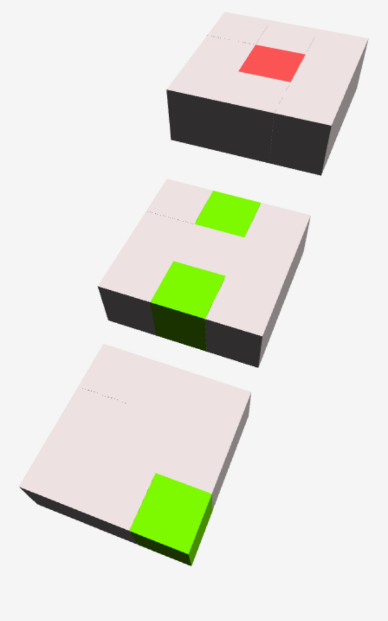
* Буквите в низът **S** винаги ще са малки латински букви в диапазона **[a-z]**.
* **S** ще има дължина в диапазона **[1…10000].**
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| ****Вход**** | ****Изход**** | ****Коментари**** |
| abaabccc | ac  3 operations | Започваме със низът **abaabccc**, премахваме първият намерен специален дубликат **aa**. Остава ни низът **abbccc**, разглеждайки го виждаме следващият специален дубликат **bb**, след премахването получаваме текстът **accc**, махайки и последният специален дубликат **cc**, получавме резултатът **ac. abaabccc -> abbccc -> accc -> ac** За да стигнем до крайният резултат извършихме **3 операции (премахвания)**. |

|  |  |  |
| --- | --- | --- |
| ****Input**** | ****Output**** | ****Comment**** |
| **baabccdd** | **Empty String**  **4 operations** | Започваме със текстът **baabccdd**, премахваме първият специален дубликат намерен **aa**. Остава ни низът **bbccdd**, разглеждайки го виждаме следващият специален дубликат **bb**, след премахването получаваме текстът **ccdd**, махайки следващият дубликат **cc**, получаваме текстът **dd**, след още едно последно премахване получаваме **празен низ**. **baabccdd -> bbccdd -> ccdd -> dd -> <празен низ>** За да стигнем до крайният резултат извършихме **4 операции (премахвания)**. |

# Задача 4 — Змия

 Даден е куб с размер **n \* n \* n**, съдържащ малки латински букви, представен като **n слоя** от квадратни матрици с размер **n \* n**. Кубът е запълнен със символа "**o**", който указва празна клетка (сивите квадратчета във фигурата). Чрез символа "**s**" (червеното във фигурата) ни се указва **къде ще се роди змията**. Чрез символа "**a**" (зелените квадратчета на във фигурата) ни се указват ябълките по картата, които змията може да изяде. На фигурата вдясно е визуализиран примерен куб, разделен на слоеве. Първоначално ни се дава **началната посока** **d** на змията, след което получаваме **команди** във формата "**d** in **j** steps", указващи пътя който змията трябва да измине, като крайната цел е да изпишем **колко ябълки е успяла да събере тя по пътя си**. Ако по пътя си змията излезе от границите на картата, всички последващи команди за движение се игнорират и трябва да изпишем екстра съобщението **"The snake dies."** на вторият ред.

### Вход

* Първият ред съдържа цяло число **n** – размерът на куба.
* На следващите **n** реда са дадени слоевете на куба (започвайки от най-горния), дадени като **n** квадратни матрици, разделени с " **|** ".
* На следващия ред ни е указана първоначалната посока **d**, по която ще тръгне змията.
* Всички последващи редове са **m** на брой команди във формата "**d** in **j** steps", указващи траекторията, която змията трябва да измине. Накрая получаваме команда от тип "end in **j** steps", която ни указва краят на командите.
* .

### Изход

* На първият ред трябва да изпишем броя на ябълките, които змията изяжда по пътя си във следната форма "Points collected: **p**", където **p** е броят ябълки.
* Ако по време на движението си змията излезе от границите на картата, трябва да изпишем "**The snake dies.**" на втория ред.

### Ограничения

* Числото **n**, указващо размера на куба ще е винаги цяло число в диапазона **[2..50].**
* Всички клетки в куба съдържат малки латински букви **[o, s, a]**.
* Началната посока **d**, както и посоката в командите винаги е една от следните **[up, down, forward, backward, left, right]**.
* Числото **j**,указващо броят стъпки преди извършване на командата винаги ще е цяло число в диапазона **[1…10 000]**.
* Командите са **m** на брой, където **m** е цяло число в диапазона **[1…100]**.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| ****Вход**** | ****Изход**** | ****Коментари**** |
| **3**  **ooo | oao | ooo**  **osa | ooo | ooo**  **ooo | oao | ooa**  **forward**  **down in 1 steps**  **backward in 1 steps**  **down in 2 steps**  **right in 1 step**  **end in 1 step** | **Points collected: 3** | **Раждаме змията на символа **s**, със посока напред(forward). Броят стъпки които трябва да направим в текущата посока е даден на долният ред. След като направим 1 стъпка напред, сменяме посоката надолу(down). След 1 стъпка (вече сме на 2рият етаж) сменяме посоката си назад(backwards). Правим две стъпки назад, сменяме посоката надолу, продължаваме с една стъпка и сменяме посоката надясно(right). Прочитайки End командата, разбираме че трябва да направим още една стъпка в текущата посока (надясно), след което сме приключили.** |

|  |  |  |
| --- | --- | --- |
| ****Вход**** | ****Изход**** | ****Коментари**** |
| **2**  **so | oo oa | ao**  **right**  **down in 1 step**  **end in 2 step** | **Points collected: 0**  **The snake dies.** | **Раждаме змията на символа s, тръгваме надясно за 1 стъпка, след което сменяме посоката си надолу. Прочитайки End командата, виждаме че трябва да продължим за още 2 стъпки преди да свършат командите. Движейки надолу, втората стъпка ни изкарва от картата и змията умира, събирайки 0 ябълки.** |

# Задача 5 – Зони за паркиране

Искате да намерите най-доброто място да паркирате колата си, за да отидете на бързо до близкия магазин. Дават ни се **n** зони за паркиране, всяка зона има **x** и **y** (координатите на горният ляв ъгъл на зоната), **дължина**, **ширина** и цена **p** – за **една минута престой** в зоната. Дават ни се още **s** свободни паркоместа под формата на **x** и **y** координатите на горните им леви ъгли, като всяко паркомясто принадлежи към зона за паркиране. Знаем също че всеки **блок** на координатната система е квадрат с размери **1 х 1**. В допълнение ни е даден и блок **t** (магазина) който се опитваме да стигнем (под формата на **x** и **y** координатите на горният му ляв ъгъл), както и константа **k**, която показва **времето нужно за преминаването на един блок** (1 единица на координатната система) **в секунди**. За целите на пресмятането на разстоянието до **t**, движение е позволено само по **четирите главни посоки** (север, юг, изток, запад**)**, **движението по диагонал е** **ЗАБРАНЕНО**.

Целта е да се намери паркомястото от което пътят до **t** и обратно струва **най-малко пари**, ако много такива пътища съществуват, трябва да намерим този който отнема **най-малко време**. Имайте в предвид че за пресмятането на парите, минутите нужни за да стигнем **t** и обратно трябва да бъдат **закръглени нагоре**, защото примерно за 2 минути и 5 секунди биваме таксувани за 3 минути.

### Вход

* На първият ред получаваме броят на зоните за паркиране **n**.
* На следващите **n** реда получаваме зоните във формата "**[Име на зоната]: [x], [y], [дължина], [ширина], [цена/мин]**"
* На следващият ред получаваме списък от свободните паркоместа във формата "**x, y**", разделени със "**;**"
* На следващият ред получаваме блока **t**, пак в формата "**x, y**"
* На последният ред получаваме константата **k**, която показва **времето нужно за прекосяването на един блок в секунди** на координатната система.

### Изход

* На единственият ред изпечатайте текста "**Zone Type: [Име на зоната]; X: [x]; Y: [y]; Price: [Цена]**",  
  където **[Име на зоната]** е името на зоната където е паркомястото с най-добрата цена и време до **t**, **[x]** и **[y]** са координатите на горният ляв ъгъл на паркомястото, а **[Цена]** е цената която трябва да платим за престоя **форматирана до 2 знака след десетичната запетая (имайте в предвид че минутите нужни за да стигнем t и обратно трябва да бъдат закръглени нагоре, защото за 2 минути и 5 секунди биваме таксувани за 3 минути)**.

### Ограничения

* Всяко паркомясто винаги ще е част от дефинирана зона за паркиране.
* Зоните за паркиране никога няма да се препокриват.
* Никога няма да има 2 паркоместа със едно и също разстояние до целта в същата зона.
* Никога няма да има 2 паркоместа със най-добрата цена и време до целта.
* Броят на зоните **n** винаги ще е цяло число в диапазона **[1…10]**.
* Цената за една минута престой в зона за паркиране **p** винаги ще е цяло число в диапазона **[1…1000]**.
* Всички входни стойности освен **p** и **n** ще са цели числа във диапазона **[1…1 000 000]**.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| **2 Blue: 0, 0, 10, 10, 1.0 Green: 0, 10, 10, 10, 0.5 1, 1; 5, 5; 1, 11; 5, 16 15, 10 20** | **Zone Type: Green; X: 1; Y: 11; Price: 5.00** |
| **Коментари** | |
| Намираме най-близкото паркомясто до t от всяка зона.  За **Синята зона**, това е паркомястото с координати **5, 5**. Броят блокове който трябва да извървим е **14**, умножаваме го по **2** за да получим броят нужен за да стигнем до **t и да се върнем**, след това умножаваме по времето нужно за прекосяването на един блок **k** (в този случай **20**) за да получим броят секунди престой. **14 \* 2 \* 20 = 560** / 60 (за да трансформираме секундите в минути) = **9 минути и 20 секунди** (тъй като сме прекосили границата от 9 минути, трябва да платим за 10 минути). **10 \*** **1** (цената/мин) = **10лв**.  За **Зелената зона**, това е паркомястото с координати **1, 11** (не проверяваме местата **[1,1]** и **[5, 16]**, тъй като разстоянието до тях очевидно е по голямо). Правим същите сметки, броят блокове е пак **14**, затова смятаме общият брой **14 \* 2** за отиване и връщане **\* 20** за броят секунди = **560** / 60 = **9м20с**, което закръгляме нагоре до **10**, този път обаче цената/мин за зоната е по малка, така че получаваме **10 \* 0.5** (цена/мин) = **5лв**.  **[1, 1]**  **t [15, 10]**  **[5, 5]**  **[1, 11]**  **[5, 16]**  **15 blocks**  **14 blocks**  **14 blocks**  **[0, 0]**  **[0, 10]** | |

# Задача 6 – Продавач

Вие сте продавач в магазин, дава ни се поредица от числа – **S** представляваща **продукти които имаме на склад** (всяко число представляващо определен продукт примерно 1 = хляб, 2 = брашно и т.н.). Също така ни се дава и **списък с поръчки** на клиенти – **O**, под формата на редица от числа (всяко число в листа с поръчки представлява поръчка за даден продукт примерно 1 = клиент е поставил поръчка за хляб, 2 = клиент е поставил поръчка за брашно и т.н.). Изпълняването на поръчка за даден продукт не го премахва от склада (представете си че имаме неограничени количества от всеки продукт в склада). След изпълняването на поръчка, имаме правото да **сменим един от продуктите** които държим на склад във какъвто продукт поискаме, стига броят на продуктите в склада да остане еднакъв. Знаейки всички поръчки в преднина, нашата задача е да пресметнем **минималният брой смени** които трябва да направим в склада, за да можем да обслужим всички поръчки.

### Вход

* На първият ред получаваме редицата **S**, като всеки елемент от **S** ще бъде разделен от останалите с един интервал.
* На вторият ред, ще получим списъка със поръчките **O**, като всеки елемент от **O** ще бъде разделен от останалите с един интервал.

### Изход

* На единственият ред от изхода изпечатайте **минималният брой смени** нужни за да се обслужат всички поръчки, алтернативно ако обслужването на всички поръчки е невъзможно изпечатайте "**impossible**".

### Ограничения

* Възможните типове продукти ще бъдат цели числа във диапазона **[1…3 000]**.
* Поредицата от продукти на склад – **S** ще съдържа само **уникални** продукти.
* Поредицата от продукти на склад – **S** ще съдържа измежду **[1…1 500]** продукта.
* Поредицата от поръчки – **O** ще съдържа измежду **[1…12 000]** поръчки.
* Позволено време: **140 мс**. Позволена памет: **32 MB**.

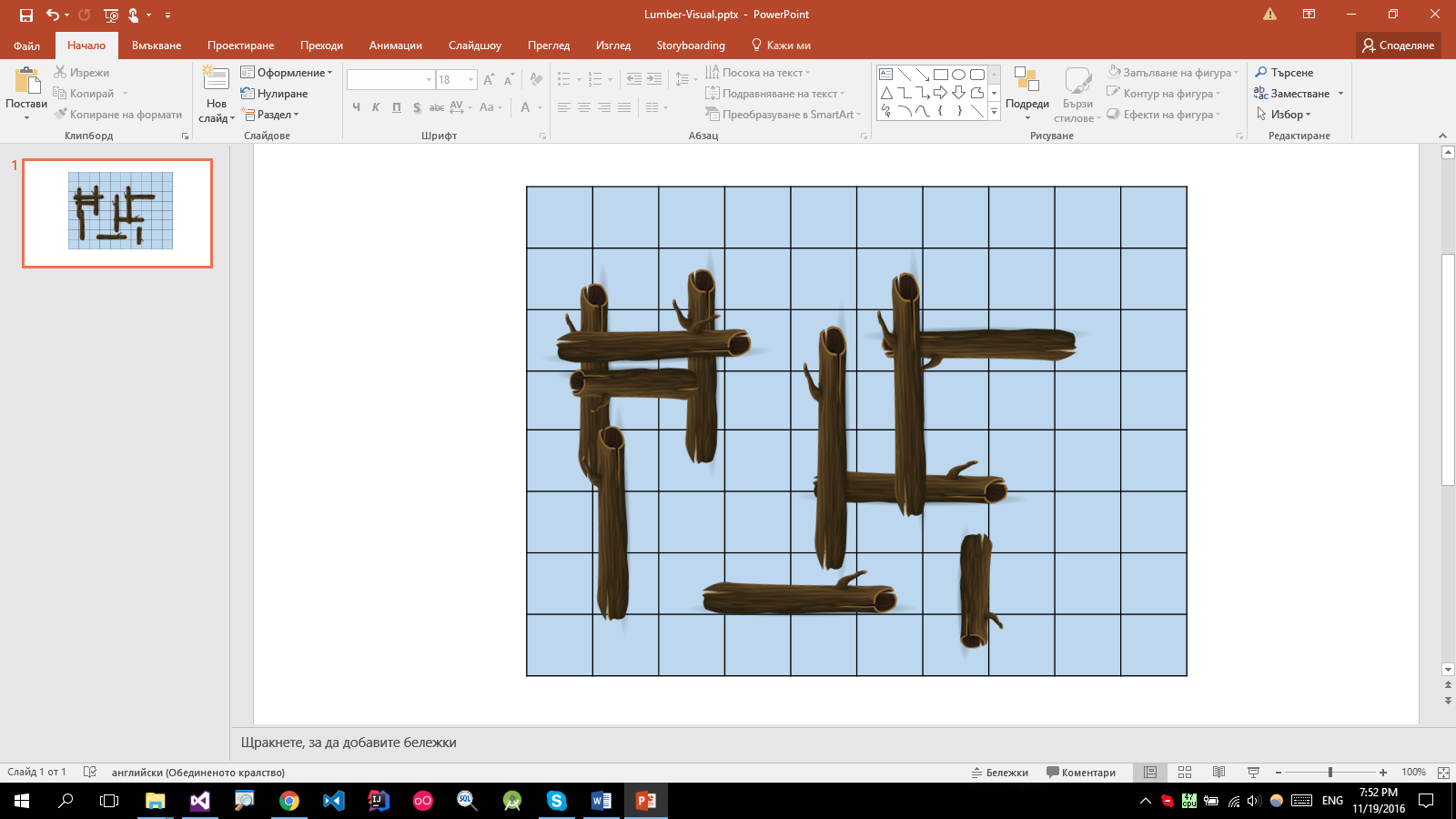
### Примерен вход и изход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 3 1 8  8 10 13 13 5 3 1 13 3 8 14 10 | 5 |

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 6 7 8  1 6 6 6 8 | impossible |
| **Коментари** | |
| Поглеждаме първата поръчка – **1**, нямаме **1** на склад така че не успяваме да изпълним поръчката, съответно изпечатваме "**impossible**". | |

# Задача 7 – Дънери

Реката Искър има много **плаващи дънери** във водите си (виж изображението по долу). Хабиби е бобър, който иска да знае дали може да стигне от дъбер **X** до дънер **Y**.



Дънерите са представени със правоъгълници, като всеки правоъгълник е представен със 2 ъгъла: **горният ляв ъгъл** **A**(**Ax**; **Ay**) и **долният десен ъгъл** **B**(**Bx**; **By**). Хабиби може да пътува между 2 дънера ако се докосват (координатите им се пресичат или покриват).

Напишете програма която казва на Хабиби дали може да пътува измежду два арбитрарни дънера.

### Вход

* На първият ред ще получите броят на дънерите **N** и броят на заявките **M** като 2 цели числа **разделени с интервал**.
* На следващите **N** реда ще получите координатите на всеки дънер в формата "**Ax Ay Bx By**".
* На следващите **M** реда ще получите заявките във формат "**X Y**", където **X** и **Y** съответстват на дънерите в реда в който са получени от входа (започвайки от 1).

### Изход

* За всяка заявка изпечатайте "**YES**" ако двата дънера са свързани или "**NO**" ако не са.

### Ограничения

* Броят на дънерите **N** ще бъде цяло число в диапазона **[2..1000]**.
* Броят на заявките M ще бъде цяло число в диапазона **[1..10000]**.
* Всички координати на дънер ще бъдат валидни цели числа в диапазона **[-100..100]**.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Визуализация** |
| 4 3  -10 30 60 10  -50 20 -30 -20  -35 60 -20 15  -40 -10 50 -30  4 2  3 4  4 1 | YES  YES  NO |  |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Визуализация** |
| 3 3  0 50 30 40  30 50 60 40  40 40 60 1  1 2  2 3  3 1 | YES  YES  YES |  |

# Задача 8 – Цветно кодиране

Дават ни се **n** двойки **поредици от цветове**. Всяка двойка е съставена от **2** редици от цветове. Един цвят може да е **пълен** (представен чрез текстовата си репрезентация примерно "red", "blue") или **частичен** (изписан със скоби примерно "(red)", "(blue)").



Първата редица може да съдържа както **частични** така и **пълни** цветове, докато втората ще съдържа само **пълни**. Разполагайки само със възможностите да **обръщаме частични цветове в пълни** и да **изтриваме всички частични цветове**, имаме задачата да проверим дали първата редица може да се обърне във втората (имайте в предвид че подредбата на редицата има значение). За всяка двойка редици – ако първата редица може да се трансформира на 2рата изпечатайте "**true**", алтернативно изпечатайте "**false**".

### Вход

* На първият ред получаваме броят на двойките редици n.
* На следващите n \* 2 реда ще получим двойки поредици от цветове във формата "**color1 color2 color3… colorN**", всеки цвят разделен от останалите със интервал (виж примерите).

### Изход

* За всяка двойка поредици от цветове, изпечатайте "**true**" или "**false"** в зависимост дали първата редица може да се трансформира във втората.

### Ограничения

* Разменянето на местата на елементи е **ЗАБРАНЕНО** – реда трябва да бъде запазен.
* Един цвят може да бъде някой от следните **[red, green, blue, yellow, orange, black, purple, brown].**
* Броят двойки поредици n ще е цяло число във диапазона **[1…100]**.
* Всяка редица ще се състои от **[1…100]** цвята.
* Позволено време: **100 мс**. Позволена памет: **16 MB**.

### Примерен вход и изход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  (blue) green (red) (blue)  blue red blue  (purple) red (blue) (yellow) blue (purple) (green) black  red blue green black  (red) red bl  red red bl  bl bl  bl | false  true  true |
| **Коментари** | |
| Разглеждайки първата двойка:  Гледайки първата редица виждаме че можем да обърнем частичните цветове в пълни за да получим нужните цветове, но нямаме никакъв начин да се отървем от пълното зелено, следователно е невъзможно да получим втората редица и изписваме **false**.  Разглеждайки втората двойка:  Поглеждаме първата редица, виждаме че имаме пълно червено, синьо и черно и ни трябва само пълно зелено между синьото и черното. За късмет имаме частично зелено точно там което да обърнем в пълно, за да завършим редицата.  Забележете че докато имаме два начина да получим изискваното синьо, сме задължени да използваме пълното такова защото иначе няма как да се отървем от него.  След като сме намерили начин да получим нужната редица изписваме **true**. | |

# Задача 9 – Търговци от Новиград

Търговците от града Новиград искат да знаят всички пътища до града Оксенфърт.

Дават ни се **N** града и **M** **еднопосочни пътя** свързващи ги. Напишете програма която намира всички **уникални** пътища между двата града. Път се счита за **уникален** ако поредицата която образува не е ендаква със никоя друга.

**Забележка:** Два пътя може да свързват едни и същи градове, но ако предицата която образуват е различна се считат за различни пътища.

### Вход

На първият ред от входа ще получите броят градове **N** и броят пътища **M**.

На следващите M реда ще ни дадат 2 цели числа разделени с интервал във формата "**U V"**,представляващи еднопосочен път от град **U** до град **V**.

Градовете ще бъдат номерирани от **1** до **N**. 1 винаги ще представлява Новиград, а N – Оксенфърт.

### Изход

Има два случая:

Ако съществуват безкрайно много пътища от Новиград до Оксенфърт, изпечатайте "**infinite**".

В противен случай изпечатайте **остатъка от броят на пътищата** разделени на 109 (модулно деление) и "**yes**" или "**no**" в зависимост дали съществуват безкрайно много пътища от Новиград до **който и да е друг град** на картата.

### Ограничения

Броят градове N ще бъде цяло число в диапазона **[2..10 000]**.

Броят пътища M ще бъде цяло число в диапазона **[1..100 000]**.

Позволено време: **300 мс**. Позволена памет: **30 MB**.

### Примерен вход и изход

|  |  |  |  |
| --- | --- | --- | --- |
| **Вход** | **Изход** | **Визуализация** | **Коментари** |
| 7 7  1 3  4 2  3 4  3 2  2 7  6 5  5 6 | 2 no |  | Има 2 пътя от **1** (Новиград) до **7** (Оксенфърт):  1->3->4->2->7  1->3->2->7  Няма безкрайно много пътеки до никой друг град затова печатаме "**no**". |

|  |  |  |  |
| --- | --- | --- | --- |
| **Вход** | **Изход** | **Визуализация** | **Коментари** |
| 6 8  1 3  1 2  1 4  2 6  3 6  3 2  4 5  5 4 | 3 yes |  | Има 3 пътя от **1** до **6**:  1->2->6  1->3->2->6  1->3->6  Съществуват безкрайно много пътища от 1 до 5 (въртейки се между 5 и 4 може да удължим пътя колкото искаме) затова печатаме "**yes**". |

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Visual** | **Comments** |
| 5 5  1 3  3 4  2 3  4 2  4 5 | infinite |  | Затворената пътека  3->4->2->3 създава безкраен брой уникални пътеки от 1 до 5, затова изпечатваме "**infinite**". |

# Задача 10 – Убежище

Вие сте командир Шепърд и имате **S** войници които се намират във военна зона. Когато врага изстреля ракета, твоите войници трябва да се скрият в едно от близките **M** убежища. Убежищата обаче имат ограничен капацитет **C** (не могат да съдържат повече от **C** войници).

Войниците и убежищата ще бъдат дадени като **2D точки**. Смятайте че един войник може да премине **1 единица** разстояние за 1 секунда.

Напишете програма която намира **най-краткото време** за което всички войници да се скрият в убежища.

### Вход

На първият ред от входа ще ни бъде даден броят на войниците **S**, броят на убежищата **M** и капацитета на убежищата **C**, отделени един от друг с интервал.

На следващите S реда ще ни бъдат дадени координатите на всеки войник във формат "**X Y**".

На следващите M реда ще ни бъдат дадени координатите на всяко убежище във формат "**X Y**".

### Изход

На единственият ред от изхода, изпечатайте **най-краткото време** за което всички войници могат да се срият в убежища (в секунди, закръглено до 6тата цифра след десетичната запетая).

### Ограничения

Броят войници **S**, убежища **M** и капацитет **C,** ще бъдат цели числа в диапазона **[1..500].**

В 50% от случаите **S, M, K <= 15**.

Винаги ще има достатъчно убежища за да се скрият всички войници.

Всички дадени координати ще бъдат цели числа в диапазона **[-1000…1000].**

Може да има войници/убежища със еднакви координати.

Позволено време: **1000 мс**. Позволена памет: **50 MB**.

### Примерен вход и изход

|  |  |  |  |
| --- | --- | --- | --- |
| **Вход** | **Изход** | **Визуализация** | **Коментари** |
| 5 3 2  3 14  10 8  5 9  14 8  12 4  3 12  11 7  5 13 | 7.071068 |  | Едно от оптималните решения.  Войниците със координати (3,14) и (5,9) отиват във убежище (3,12). (12,4) и (14,8) отиват в убежище (11,7), а войник (10,8) отива в убежище (5,13).  Забележете че (10,8) може да стигне до убежище (11,7) по-бързо, обаче капацитета не го позволява. |