# СофтУниада 24 и 25 Март 2018

## Готини числа

Жоро найстина обича числа, но според него някои числа са **по-готини** от други, помогни на Жоро да напише програма да **класифицира числа**. Според Жоро числата се класифицират по **3 условия**:

* Дали са **нечетни**
* Дали са **отрицателни**
* Дали се **делят без остатък** нанеговото **любимо число**

Според 3те условия Жоро класифицира числата така:

* Ако едно число **не спазва нито едно от условията** (не е нечетно, не е отрицателно и не се дели без остатък на любимото му число) то Жоро мисли че числото е “**boring”**
* Ако числото **спазва само едно** от условията, то Жоро смята че е “**awesome**”
* Ако числото **спазва 2 от условията**, то е “**super awesome**”
* Ако числото **спазва** **всичките 3 условия**, то то е “**super special awesome**”

### Вход

От конзолата се четат два реда:

* На първият ред ще е числото което Жоро иска да класифицира
* На вторият ред ще е **любимото** **число** на Жоро

### Изход

* На единственият ред от изхода трябва да се изпечата как Жоро класифицира числото:
  + Не спазва условията -> “**boring”**
  + Спазва 1 условие ->“**awesome**”
  + Спазва 2 условия ->“**super awesome**”
  + Спазва 3 условия ->“**super special awesome**”

### Ограничения

* **Числото което Жоро иска да класифицира** ще е **цяло число** в интервала **[-1 000 000…1 000 000]**
* **Любимото число** на Жоро ще е **цяло число** в интервала **[-1 000 000…1 000 000]**
* Позволено време: **100 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 13 7 | **awesome** | Числото **13**: нечетно **√** отрицателно **X** дели се на **7** **X** изпълнява само едно от условията съответно пишем „**awesome**” |
| **Вход** | **Изход** | **Обяснения** |
| -27 9 | **super special awesome** | Числото **-27**: нечетно **√** отрицателно **√** дели се на **9** **√** изпълнява всички условия съответно пишем „**super special awesome**” |
| **Вход** | **Изход** | **Обяснения** |
| 151734 2152 | **boring** | Числото **151734**: нечетно **X** отрицателно **X** дели се на **2152** **X** не изпълнява нито едно условие съответно пишем „**boring**” |
| **Вход** | **Изход** | **Обяснения** |
| -1158 -6 | **super awesome** | Числото **-1158**: нечетно **X** отрицателно **√** дели се на **-6** **√** изпълнява две от условие съответно пишем „**super awesome**” |
| **Вход** | **Изход** | **Обяснения** |
| -33 12 | **super awesome** | Числото **-33**: нечетно **√** отрицателно **√** дели се на **12** **X** изпълнява две от условие съответно пишем „**super awesome**” |

## Великденско яйце

Дасе напише програма, която прочита от конзолата **цяло четно число n**, както в примерите   
по-долу. Великденското яйце е **с ширина – 5 \* n** и **височина - 2 \* n + 3**. **Вашата задача е да съставите програма, която да отпечатва великденско яйце.**

### Вход

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

### Изход

* Да се отпечата **на конзолата великденско яйце**, точно както в примерите.

### Ограничения

* Числото **n** ще е винаги **цяло четно число** в интервала **[2…28]**
* Позволено време: **100 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |  |  |
| --- | --- | --- | --- |
| **Вход** | **Изход** | **Вход** | **Изход** |
| 6 | ............\*\*\*\*\*\*............  ..........\*++++++++\*..........  ........\*\*++++++++++\*\*........  ......\*\*\*++++++++++++\*\*\*......  .....\*\*================\*\*.....  ....\*\*==================\*\*....  ...\*\*====================\*\*...  ...\*\*~~~~HAPPY EASTER~~~~\*\*...  ...\*\*====================\*\*...  ....\*\*==================\*\*....  .....\*\*================\*\*.....  ......\*\*\*++++++++++++\*\*\*......  ........\*\*++++++++++\*\*........  ..........\*++++++++\*..........  ............\*\*\*\*\*\*............ | 8 | ................\*\*\*\*\*\*\*\*................  ..............\*++++++++++\*..............  ............\*\*++++++++++++\*\*............  ..........\*\*\*++++++++++++++\*\*\*..........  ........\*\*\*\*++++++++++++++++\*\*\*\*........  .......\*\*======================\*\*.......  ......\*\*========================\*\*......  .....\*\*==========================\*\*.....  ....\*\*============================\*\*....  ....\*\*~~~~~~~~HAPPY EASTER~~~~~~~~\*\*....  ....\*\*============================\*\*....  .....\*\*==========================\*\*.....  ......\*\*========================\*\*......  .......\*\*======================\*\*.......  ........\*\*\*\*++++++++++++++++\*\*\*\*........  ..........\*\*\*++++++++++++++\*\*\*..........  ............\*\*++++++++++++\*\*............  ..............\*++++++++++\*..............  ................\*\*\*\*\*\*\*\*................ |

## Бинго генератор

Мишо и Дени са измислили нова игра на име "Бинго", но ти ще трябва да им помогнеш и да създадеш генератор, който да проверява дали въведенето **четирицифрено** **число** печели.

Правилата на играта са следните:

1. Въвеждате се **четирицифрено** **число**, като от него трябва да се образуват нови **две** **двуцифрени**.
2. **Двете** **двуцифрени** числа се образуват по следният начин:
   1. **първото** **число** е съставено от цифрата на **хилядните** и цифрата на **десетиците** на четирицифреното.
   2. **второто** **число** е съставено от неговите **стотици** и **единици**.
3. След като получим **2те** **двуцифрени** **числа**, трябва да пресметнем **тавана. Тавана** се калкулира като съберем **2те** **двуцифрени числа**. Например **1234** – първото двуцифрено е **13**, а второто е **24**, сбора им е 13 + 24 = **37**,съответно **тавана** е **37.**
4. Като следваща стъпка трябва да образуваме ново четерицифрено число – **началният** **елемент**. **Началният** **елемент** се образува като **слепим** **второто** **двуцифрено** **число** **за** **краят** **на** **първото**, Примерно ако първото двуцифрено число е **13**, а второто **24**, началният елемент ще бъде **1324**.
5. Следващата стъпка е да изгенерираме всички четерифрени числа които спазват следните правила. Редът в които генерираме числата трябва да е следният – започваме от **началният** **елемент**, първо **увеличаваме** **второто** **двуцифрено** **число** докато не стигне лимита си, след което **увеличаваме** **първото** **двуцифрено** **число**, **рестартираме** **второто** двуцифрено число и **повтаряме** цялата операция докато изчерпаме всички числа спазващи условията. Примерно при **начален** **елемент** **1123** и **таван** **37**, генерираме числата в следният ред 1123 -> 1124 -> 1125… 1137->1223->1224->1225…->1237->1323…:
   1. Са **по големи или равни** на **началният елемент**
   2. Двуцифреното число което образуваме като вземем **първите две цифри** на новогенерираното четерицифрено число е **по малко или равно** на **тавана**
   3. Двуцифреното число което образуваме като вземем **вторите две цифри** на новогенерираното четерицифрено число е **по малко или равно** на **тавана**

Примерно ако вземем числото **1334** като пример с **таван** – **37** и **начален елемент** **1324**, можем да видим че спазва ограниченията**:**

* Числото е по-голямо от началният елемент => **1334 > 132**4
* Двуцифреното число образувано от първите две цифри е по-малко от тавана => **13 < 37**
* Двуцифреното число образувано от вторите две цифри е по-малко от тавана => **34 < 37**

Ако вземем числото **1342** като пример обаче, то **НЕ е** **валидно** защото не отговоря на условието вторите две цифри да са по малки от тавана => **42 > 37**

1. След съставянето на новите **четирицифрени** **числа** трябва да се провери **дали те** **се делят на 12 или 15** и да отпечатаме делящите се в две групи. Ако броят на числата, които се делят на 12 и броят на числата които се делят на 15 **са равни** играчът въвел четирицифреното число печели.

**\*ЗАБЕЛЕЖКА: Ако някое от новогенерираните четирицифрени числа се дели и на 12, и на 15 тогава трябва да го отпечатате и в двете групи.**

### Вход

* **Един ред** – въведеното четирицифрено число

### Изход

**Три реда:**

* **Първи ред :** Dividing on 12: {новогенериреаните четирицифрени числа делящи се на 12 в нарастващ ред}
* **Втори ред :** Dividing on 15: {новогенериреаните четирицифрени числа делящи се на 15 в нарастващ ред}
* **Трети ред:** Ако броят на числата делящи се на 12 е равен на броя на числата делящи се на 15 да се отпечата: **“**!!!BINGO!!!”, в противен случай отпечатайте **“**NO BINGO!”

### Ограничения

* Четерицифреното число ще е цяло число между **[1000…9999]**
* Ако **тавана** е **по голям или равен** **на** **99** то всички двуцифрени числа са валидни.
* Позволено време: **100 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 1213 | Dividing on 12: 1128 1224 1332 1428 1524 1632 1728 1824 1932 2028 2124 2232 2328 2424 2532 2628 2724 2832 2928 3024 3132 3228 3324 3432  Dividing on 15: 1125 1230 1425 1530 1725 1830 2025 2130 2325 2430 2625 2730 2925 3030 3225 3330  NO BINGO! | Взимаме **първата цифра** (цифрата на хилядните **1**) и **третата цифра** (цифрата на десетиците **1**) от четирицифреното число => образуваме **първото двуцифрено число** , което е 11. След това образуваме **другото двуцифрено число**, което е 23 ( цифрата на стотните **2** и цифрата на единиците **3**). **Максималната стойност на всяко едно от новите двуцифрени числа може да бъде сборът им, т.е 11 + 23 = 34**. След като генерирараме числата (1123, 1124, … , 1224, … , 3434) проверяваме всяко едно от тях и го поставяме в една от двете групи делящи се на 12 или на 15.  **Понеже броят на делящите се на 12, който е 24 е различен от броя на делящите се на 15, който е 16 изписваме** NO BINGO! |
| 1214 | Dividing on 12: 1128 1224 1332 1428 1524 1632 1728 1824 1932 2028 2124 2232 2328 2424 2532 2628 2724 2832 2928 3024 3132 3228 3324 3432 3528  Dividing on 15: 1125 1230 1335 1425 1530 1635 1725 1830 1935 2025 2130 2235 2325 2430 2535 2625 2730 2835 2925 3030 3135 3225 3330 3435 3525  !!!BINGO!!! | |

## Грешни резултати

Даден е куб (триизмерна матрица) с целочислени стойности. Матрицата представлява резултатите от проведен експеримент в лаборатория, но при събирането на същите е възникнал проблем и в матрицата са добавени **елементи с грешни стойности**. Знаем **координатите на една** от грешните стойности в матрицата **(element).** Съответно трябва да сменим всички числа с равна на **element** стойности.

Задачата е да се **намерят в матрицата грешните стойности и да се заменят с верните**. Като вярната стойност е резултата от **сумата на най-близките съседни елементи** - тези намиращи се на разстояние от един индекс (от грешната стойност), където няма такива – съответно не се прибавят. Вземат се стойностите в **ляво, дясно, отгоре, отдолу, отпред и отзад** на конкретния елемент. Ако елемента няма правилни съседни стойности, които могат да се сумират, същия приема стойност равна на нула.

Възможно е да има съседен елемент, чиято стойност също да е грешна – в този случай стойността на съседа **не се прибавя** към общия сбор от съседите (тоест този елемент се пропуска), независимо дали за него има вече изчислена вярна стойност или не.

### Вход

* Първият ред съдържа цяло число **size** – размера на куба.
* На следващите **size** реда са дадени **size** на брой матрици разделени с " **|** ", като всяка една представлява един слой от триизмерната матрица и съдържа **size** **\* size** елемента.
* На последния ред са дадени координатите **x, y, z -** индексите на елемента за който знаем, че има грешна стойност.

### Изход

* На първият ред трябва да изведем броя на откритите и сменени елементи под формата **"**Wrong values found and replaced: elementsCount**"**, където **elementsCount** е броят на елементите, които сме подменили .
* На следващите **size \* size** редове трябва да изведем всички **size** на брой елементи на матрицата с правилните стойности разделени с празен символ **" "**.

### Ограничения

* Стойността **size** на куба ще бъде винаги цяло число в диапазона **[3..25].**
* Всички клетки в куба съдържат цели числа в диапазон **[-1000…1000]**.
* Координатите на елемента с грешна стойност винаги ще бъдат цели числа намиращи се в границите на матрицата.
* Няма да има матрица съставена от една стойност.
* Позволено време: **200 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Вход** | **Изход** | **Коментари** | | | |
| **3**  **1 2 4 | 6 7 8 | 1 1 4**  **4 6 7 | 4 9 3 | 4 4 4**  **8 9 4 | 4 2 4 | 7 9 4**  **0 1 0** | **Wrong values found and replaced: 11**  **1 2 17**  **15 6 7**  **8 9 16**  **6 7 8**  **15 9 3**  **17 2 5**  **1 1 9**  **8 19 3**  **7 9 9** | **Според дадените координати грешната стойност е равна на 4.Търсим всички елементи равни на 4. Намираме първият елемент и изчисляваме сбора на близките му съседи 15 = (1 + 6 + 8). Текущия елемент няма съседи от предната и от лявата си страни, от задната си страна има елемент равен на 4, за това не го добавяме към сбора. По аналогичен начин откриваме всички елементи в матрицата, изчисляваме сумата на близките им съседи и заменяме стойностите.** | | | |
| **4**  **4 0 0 0 | 12 12 10 1 | 14 10 6 12 | 11 12 14 12**  **11 1 9 11 | 9 10 2 7 | 6 8 5 7 | 10 7 6 12**  **5 0 12 12 | 12 0 3 0 | 12 1 12 7 | 12 8 4 12**  **7 4 1 12 | 12 11 9 12 | 12 8 8 12 | 12 2 12 12**  **0 2 3** | | | **Wrong values found and replaced: 21**  **4 0 0 0**  **11 1 9 11**  **5 0 13 11**  **7 4 1 1**  **27 30 10 1**  **9 10 2 7**  **14 0 3 0**  **18 11 9 9**  **14 10 6 14**  **6 8 5 7**  **7 1 28 7**  **8 8 8 15**  **11 42 14 14**  **10 7 6 13**  **18 8 4 11**  **2 2 14 0** | **Грешната стойност е равна на 12. При елемента с индекси 0 2 3 имаме само един съсед, чиято стойност може да се вземе и това е елемента от горната страна 11, всички други съществуващи съседи са с грешна стойност за това не прибавяме тяхната сума. При последния елемент в матрицата няма, нито един съсед, чиято стойност, може да се сумира, за това елемента на този индекс става равен на 0.** |

## Описана окръжност

Дадени са поредица от фигури - **триъгълник** и **кръг** в двумерна Декартова координатна система. Намерете, дали **окръжността е описана около триъгълника (окръжността е описан около триъгълник, когато върховете на триъгълника се намират върху външната част на окръжността)** и дали **нейният център се намира в триъгълника или извън него (за точки, лежащи на някоя от страните на триъгълника приемаме, че се намират в него).**

Всеки **триъгълник** се дефинира с три точки: долна лява , долна дясна и горна . Всеки **кръг** се дефинира с център и радиус.

### Вход

* Входът се чете от конзолата.
* На първия ред е броят на двойките фигури - **n**.
* На следващите **2** **\*** **n** реда се подават входните данни на фигурите, като винаги са по една фигура на ред и имат следния формат:
  + **circle Ox, Oy, R**
  + **triangle Ax, Ay, Bx, By,Cx, Cy**
* Параметрите винаги ще се подават в тази последователност.

### Изход

* Изходът винаги се състои от един ред за всяка двойка фигури.
* Като за всяка проверена комбинация, следва да се отпечата на един ред:
  + Окръжността **е описана** около триъгълника и **центъра се намира** в триъгълника, отпечатано на един ред 🡪 “**The circle is circumscribed and the center is inside.”**
  + Ако окръжността **е** **описана** около триъгълника, но **центъра не се намира** в него да се отпечата

🡪 “**The circle is circumscribed and the center is outside.”**

* + Окръжността **не е описана** около триъгълника, **но центъра се намира** в триъгълника, отпечатано на един ред 🡪 “**The circle is not circumscribed and the center is inside.”**
  + Ако окръжността **не е описана** около триъгълника и **центъра не се намира** в него да се отпечата 🡪 “**The circle is not circumscribed and the center is outside.”**

### Ограничения

* **n** ще е цяло число в интервала **[1…1000]**.
* , , , , , , , , и са реални числа в интервала **[-1000...1000]** с не повече от 5 цифри след десетичната точка.
* винаги ще е положително число.
* За всички реални числа във входа разделителят е „.“, например „1.45“ и „2.5“.
* Когато правите изчисления, приемете, че две стойности се смятат за еднакви, ако разликата помежду им е по-малко от 0.01 единици.
* Позволено време: **200 ms**. Позволена памет: **32 MB**.

### Примерни вход и изход

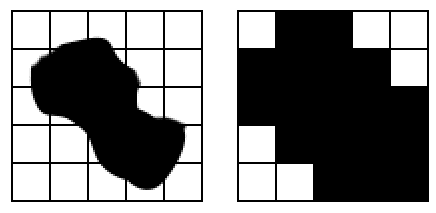
|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 1  circle -13.5, 19.5, 22.63  triangle -36.13, 19.5, 6, 8, -13.5, 42.13 | The circle is circumscribed and the center is inside. | Центъра на окръжността **се намира на равни разстояния от всички върхове на триъгълника, следователно е описана около него**. Сбора от площта на триъгълниците образувани между центъра на окръжността и върховете на триъгълника, **е равна** на площта на триъгълника, следователно центъра на окръжността се намира в триъгълника. |
| 1  circle 3.5, 3, 2.69  triangle 1, 2, 6, 2, 1, 4 | The circle is circumscribed and the center is inside. | Центъра на окръжността **се намира на равни разстояния** от всички върхове на триъгълника, следователно е описана около него. Сбора от площта на триъгълниците **е равна** на площта на триъгълника, следователно центъра на окръжността се намира в триъгълника. |
| 1  circle -2, 1.5, 6.9  triangle -6, 4.5, 3.2, 1, 2, 6 | The circle is not circumscribed and the center is outside. | Центъра на окръжността **не се намира** на равни разстояния от всички върхове на триъгълника, следователно **не е описана** около него. Сбора от площта на триъгълниците образувани между центъра на окръжността и върховете на триъгълника, **не е равна** на площта на триъгълника, следователно центъра на окръжността се намира извън триъгълника. |
| 1  circle 3.5, 3, 4.58  triangle 1, 4, 3.5, 0.31, 6, 4 | The circle is not circumscribed and the center is inside. | Центъра на окръжността **не се намира** на равни разстояния от всички върхове на триъгълника, следователно **не е описана** около него. Сбора от площта на триъгълниците образувани между центъра на окръжността и върховете на триъгълника, **е равна** на площта на триъгълника, следователно центъра на окръжността се намира в триъгълника. |

## Астероиди

Вие сте капитан на космически кораб в далечна планетна система. Предстои Ви най-важният и опасен етап от пътуването: да преминете през гъсто населения астероиден пояс. На предния монитор се разкрива картина, подобна на тази:



Разполагате с програма, която обработва суровата картина. Тя генерира правоъгълна мрежа (поле) от монохромни пиксела. Ако част от астероид запълва някаква част от площта на даден пиксел, той има стойност **1**, в противен случай – **0**. Пример за начина на работа на тази програма е даден долу:



Намерете **общия брой** астероиди. **Групирайте ги по площ** (общ брой пиксели на екрана) – от най-големите (които са най-опасни) към най-малките. Астероидът от примера има площ 18.

### Вход

Получавате няколко екрана, които трябва да обработите.

Всеки екран започва с ред, съдържащ числата **N** и **M**, разделени с единствен символ „**x**“. Това е размерът на екрана.

Следващите реда съдържат точно на брой двоични стойности: **0** или **1**.

Последният ред от входа съдържа единствено символния низ „**end**“.

### Изход

За всеки екран, изведете групите астероиди във формат **GxS**, където е **площта на астероида**, а **– броят астероиди в полето**, които имат съответната площ (да се чете „**G**“ астероида с площ “**S**”). **Подредете ги от най-голяма към най-малка площ**.

На края на всеки екран, изведете общия брой астероиди в него във формат: **Total: T**.

### Ограничения

* Във всяко поле ще има поне един астероид
* Позволено време: **350 мс**. Позволена памет: **32 MB**.

### Примерни вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 5x5  01100  11110  11111  01111  00111  end | 1x18  Total: 1 | Примерът от условието. |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 3x3  100  000  001  5x4  1011  0011  1000  1111  0000  2x2  10  01  end | 2x1  Total: 2  1x5  1x4  1x1  Total: 3  2x1  Total: 2 | Три екрана с различни размери, съдържащи астероиди. |

## Кола

Кольо си е купил нова кола втора употреба за осми пореден път и е решил да я пробва из Софийските улици по време на нощта. Той обаче трябва да внимава след всеки завой и да увеличава или намалява скоростта на мощномобила си преди всяка улица.

Програмата ви ще получи списък от **C** на брой цели числа, където **i-тото число** означава с колко иска Кольо да **промени** **скоростта** преди **i-тата** улица. Кольо може да избере - или да **увеличи текущата скорост** с даденото число, или съответно **да я** **намали** с него. Освен това програмата ви ще получи и едно цяло число **B** – **първоначалната** **скорост** на колата – и едно число **M** – **максималната** **възможна** **скорост** на колата. Тоест, Кольо **не може да шофира с по-малка скорост от 0** и **не може да шофира с по-голяма скорост от M** (но може с точно 0 или точно M). Програмата трябва да сметне **колко е максималната скорост** на колата, която Кольо може да ползва **за последната улица**. Ако няма начин да се направят последователно всичките промени в скоростта, без тя да стане над M или под 0, програмата трябва да изведе -1.

### Вход

Входните данни се четат от конзолата и ще са винаги валидни:

* На първия ред от входа ще бъде числото **C** – броят на числата, с които Кольо иска да смени скоростта на колата си.
* На втория ред ще има **C** на брой цели числа – където всяко поредно число означава **промяната**, която Кольо иска да направи в скоростта на колата преди поредната улица.
* На следващия ред ще бъде числото **B** – **първоначалната** **скорост** на колата на Кольо.
* На следващия ред ще бъде числото **M** – **максималната** **скорост** на колата на Кольо.

### Изход

Изходните данни се печатат на конзолата:

**На единствения ред от изхода програмата трябва да изведе или числото -1, или максималната скорост, с която Кольо може да премине на последната улица.**

### Ограничения

* Броят на опитите за промяна на скоростта - **C** ще бъде цяло число между **[1...50]**
* В **95%** от тестовите случаи **C** ще е по-малко от **34**
* Всяка промяна на скоростта ще бъде цяло число между **[1...M]**
* Максималната скорост **M** ще бъде цяло число между **[1...1000]**
* Първоначалната скорост **B** ще е цяло число между **[0…M]**
* Позволено време: **100 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3  5 3 7  5  10 | 10 |

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  15 2 9 10  8  20 | -1 |

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 14  74 39 127 95 63 140 99 96 154 18 137 162 14 88  40  243 | 238 |

## Логистика

Държите куриерска фирма разнасяща пратки, на първият ред се подава числото **N** – броят пратки които ще получите. На следващите **N** реда са подадени **пратките** – всяка **пратка** има:

* **цена** (сумата която ще бъде получена за доставяне)
* **краен срок за доставяне** (номерът на деня след който пратката става невалидна и не може да бъде доставена).

Ако една пратка има краен срок **ден 4** примерно, то това означава че тя може да се достави във всеки от дните от **1** до **4** – съответно пратката може да се достави в **ден 1, ден 2**, **ден 3** и **ден 4** (но не в ден 5 и нагоре). Пратките се **индексират** в ред на подаването им от входа (първата подадена пратка е с индекс **1**, втората – с индекс **2** и така нататък).

Фирмата разполага с **1** **микробус** с които да разнася поръчките, като той може да разнася по **1 пратка** **на ден**. Шофиорът обаче все още е нов и от време на време попада в **катастрофи**, ако шофиорът катастрофира, фирмата **не получава пари за доставката**, като освен това трябвада **плати цена от** **50** за поправки на микробуса, както и обезщетение на клиента равно на **цялата цена на пратката с която е катастрофирал микробуса**.

Целта на задачата е да се извади лист от пратките които трябва да достави микробуса във всеки ден така че фирмата да има най-добър **краен баланс** (максимална печалбаили минимална загуба, ако шофиорът катастрофира прекалено често).

Ако съществуват няколко оптимални разпределения, трябва да се следват следните правила за разпределяне:

* Примерно ако погледнем следните пратки:
  + Пратка 1 – цена 20, краен срок 2
  + Пратка 2 – цена 10 краен срок 2

Има 2 начина да се разпределят оптимално – Вариант 1:

* + Ден 1 -> пратка 1 (цена 20)
  + Ден 2 -> пратка 2 (цена 10)

или Вариант 2

* + Ден 1 -> пратка 2 (цена 10)
  + Ден 2 -> пратка 1 (цена 20)
* В случай с пратки които ще се **доставят** **успешно**, трябва да се използва **нарастващ ред** (Вариант 2).
* Алтернативно ако знаехме че, **микробуса ще** **катастрофира** и в 2та дни, следва пратките да се подреждат в **намаляващ** **ред** (Вариант 1)

### Вход

* На първият ред получаваме числото **N** – броят на пратките.
* На всеки от следващите **N** реда получаваме по една пратка в формата **„{цена} {краен срок}”**.
* На последния ред са изредени всички **дни в които микробуса ще катастрофира** разделени един от друг със спейсове - **„{ден на катастрофа} { ден на катастрофа} … { ден на катастрофа}”,** ако катастрофи няма да – на реда пише **„**none**”**.

### Изход

* На първият ред от изхода трябва да се изпечата **крайният баланс**(сумата от парите които са получени от клиенти минус разходите за поправки и обезщетения).
* На вторият ред от изхода се изпечатват поредицата от поръчки които микробуса трябва да вземе във формата **„{индекса на пратката която ще се достави в ден 1} { индекса на пратката която ще се достави в ден 2} … { индекса на пратката която ще се достави в последният ден}”**

### Ограничения

* Броят на пратките ще е цяло число между **[1…100 000].**
* Никога няма да има 2 пратки с еднаква цена.
* Цената на пратка ще е цяло число между **[1…1 000 000]**.
* Крайният срок за доставяне ще е цяло число между **[1…100 000].**
* Броят на катастрофите ще е между **[0…100 000],** но не повече от **най-големият краен срок.**
* Позволено време: **450 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  25 3  15 1  10 1  50 3  none | 90  2 1 4 |
| **Коментари** | |
| Индексирани пратките изглеждат така: #1 25 3  #2 15 1  #3 10 1  #4 50 3  Преглеждаме всички пратки, веднага виждаме че **#4** и **#1** са най-скъпи, но ако изберем да доставим някоя от тях през **ден 1**, няма да имаме възможността да доставим 2 пратки през следващите 2 дена (пратки **#2** и **#3** не могат да се доставят след **ден 1**, защото крайният им срок е изтекъл). Преглеждаме 2те пратки със срок **ден 1**, с цени съответно **15** и **10**. **#2** има по-голямата цена затова избираме нея. За ден 2 и ден 3 имаме избор между 2 пратки – **#1**, с цена **25** и **#4** с цена **50**. Понеже и 2те пратки имат краен срок **ден 3**, имаме 2 начина за разпределяне, следвайки правилата за разпределение описани в условието изпозлваме нарастващият ред, съответно в **ден 2** избираме **#1**, а в **ден 3** - **#4** Така крайният ни план за доставкa е следният: **Краен баланс** = 15(#2) + 25(#1) + 50(#4) = **90.00.**  **Разпределение на пратките в микробуса**  2 1 4 | |

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5  30 2  14 2  13 2  43 1  17 2  2 1 | -127  2 3 |
| **Коментари** | |
| Индексирани пратките изглеждат така: #1 30 2  #2 14 2  #3 13 2  #4 43 1  #5 17 2  От входа виждаме че микробусаще катастрофира през **ден 1** и през **ден 2**. Понеже микробуса ще катастрофира избираме да доставим най-евтините поръчки, това са пратки **#2** и **#3**. Понеже и 2те пратки имат краен срок **ден 2** имаме 2 начина да ги разпределим, следвайки правилата от условието за разпределяне при катастрофи, използваме намаляващ ред, съответно слагаме пратка **#2** през **ден 1** и **#3** през **ден 2**.  Микробусъткатастрофира докато доставя пратка **#2** през **ден 1**, в следствие на което трябва да платим 50 за поправки и 14 на клиента като обезщетение, правейки баланса за деня 0 – 50 – 14 = **-64**  През **ден 2** имаме още една катастрофа, този път с пратка **#3**, като резултат плащаме още 50 за поправка и 13 за обезщетение. Така крайният ни баланс след **ден 2** е следният: **Краен баланс** = -64(ден 1) - 50(катастрофа ден 2) - 13(обезщетение за #3) = **-127.**  **Разпределение на пратките в микробуса**  2 3 | |

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 7  21 2  18 2  13 4  2 1  6 1  4 3  33 4  2 | 13  1 6 3 7 |
| **Коментари** | |
| Индексирани пратките изглеждат така: #1 21 2  #2 18 2  #3 13 4  #4 2 1  #5 6 1  #6 4 3  #7 33 4  Разглеждаме входа и виждаме че имаме катастрофа през **ден 2**, съответно избираме най евтината пратка достъпна през **ден 2** – пратка **#6**. Разглеждайки пратките най-скъпата пратка която можем да изберем през **ден 1** очевидно е **#7**, ако доставим пратка **#7** през **ден 1** обаче, няма да можем да доставим пратки **#1** или **#2** (защото **ден 1** ще е зает от пратка **#7**, а през **ден 2** има катастрофа). Ако изтестваме вариантите, можем да забележим че най-правилната подредба би била: **Разпределение на пратките в микробуса**  1 6 3 7  Давайки ни **Краен баланс** = 21(#1) – 50(катастрофа ден 2) – 4(обезщетение за #6) + 13(#3) + 33(#7) = **13**  Нека проверим:  В **ден 1** ще вземем **#1** има краен срок **ден 2**.  В **ден 2** ще вземем **#6** – с краен срок **ден 3**.  В **ден 3** ще вземем **#3** с краен срок **ден 4**.  В **ден 4** ще вземем **#7** с краен срок **ден 4**. | |

## Мафия

Вашата задача е да разбивате глобални кибермафиотски организации. Във всяка организация, хората са свързани взаимно, като всяка връзка определя едно познанство между хора.

Организацията разполага с на брой компютъра. Всеки от тях е означен с **уникален** **идентификатор** от до . Компютърът с идентификатор принадлежи на **шефа**, а компютърът с идентификатор – на неговия **най-близък подчинен**. Останалите компютъра са само частично във Ваш контрол. Не знаете точно кои хора използват компютрите, но знаете следните неща:

* Компютрите в цялата организация са свързани в мрежа
* Всяка директна връзка между два компютъра има определен капацитет : пакети, които могат да преминат всяка секунда.

Искате да осъществите DDoS атака, за да прекъснете връзката между шефа и неговия най-близък подчинен. При даден модел на мрежата, трябва да оцените колко е максималният капацитет (в пакети за секунда) на мрежата между компютрите на шефа и най-близкия му подчинен.

### Вход

Входът съдържа информация за няколко на брой престъпни огранизации.

Описанието на всяка организация започва с числото **N** единствено на ред – общият брой компютри в нея. Следващите редове съдържат връзките между отделните компютри във формат **A-B P**, където **A** е идентификаторът на първия компютър, **B** е идентификаторът на втория компютър и **P** е капацитетът на връзката . Тирето във входните данни има смисъл на „връзка“, не на знак за отрицание. **Връзките са двупосочни**.

Последният ред от входа съдържа единствено символния низ „**end**“.

### Изход

За всяка кибермафиотска организация, изведете номера ѝ и капацитета на връзката между шефа на мафията и неговия най-близък подчинен. Изведете резултата във формат „**Group G: T**“, където е номерът на групата (**1** за първата въведена от входа група, **2** за втората и т. н.), а е общия капацитет на мрежата.

### Ограничения

* Броят на **престъпните** **организации** в един тест ще е между **[1…100]**
* Позволено време: **350 мс**. Позволена памет: **32 MB**.

### Примерни вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 4  1-2 14  3-4 14  1-3 6  2-4 6  1-4 5  end | Group 1: 17 | Директната връзка 1-4 има капацитет 5. Индиректните връзки 1-2-4 и 1-3-4 имат капацитет 6.  Общият капацитет е 5 + 6 + 6 = 17. |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 3  1-2 5  2-3 4  5  1-2 4  4  1-2 14  3-4 14  1-3 6  2-4 6  2-3 5  end | Group 1: 4  Group 2: 0  Group 3: 17 | Примерът показва поведението на програмата при няколко мрежи. В първата мрежа има един възможен път: 1-2-3 с капацитет 4. Във втората мрежа само компютрите 1 и 2 са свързани; шефът на мафията няма връзка със заместника си. Третата мрежа повтаря горния примерен вход. |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 6  1-2 10  2-3 20  1-4 15  4-5 7  3-6 13  5-6 22  4-3 19  end | Group 1: 20 |  |

## Лабиринта на Дурин

Ти си известен археолог, който намира загубеният лабиринт на Дурин, в предверието виждаш голяма карта детайлно описваща лабиринта. Лабиринта представлява серия от **стаи** свързани помежду си с тесни **тунели**, като до всяка **стая** има написан номер (**индексът** на стаята), на картата също така са специфично отбелязани 2 стаи – **предверието** в което се намираш и **съкровищницата**. Под картата е написана загадка, която казва че за да отключиш **съкровищницата**, трябва да знаеш **тайният код**, в загадката е също така описано как **тайният код** може да се пресметне. Според загадката независимо кой път вземеш от **предверието** към **съкровищницата**, има някои **стаи** **през които** **винаги ще трябва да минеш**, загадката нарича тези стай – **ключови стаи**. В загадката също така специфично е упоменато че **предверието** и **съкровищницата** **НЕ се считат за ключови стаи**. Според текста **тайният код** може да се получи като се съберат номерата (**индексите**) на всички **ключови стаи**.

### Вход

* На първият ред получаваме числото **N** – броят на стаите в лабиринта.
* На вторият ред получаваме числото **M –** броят на тунелите в лабиринта.
* На всеки от следващите **M** реда получаваме информация за **тунел** който свързва 2 **стаи** в формата **„{индекс на стая 1} {индекс на стая 2}“**.

### Изход

* На първият ред от изхода трябва да се изпечата **тайният код**.
* На вторият ред от изхода трябва да се изпечатат **индексите** на всички **ключови стаи**, в реда в който сме минали през тях от **предверието** до **съкровищницата**, в формата **„{индекс на ключова стая 1}->{индекс на ключова стая 2}->…{индекс на последната ключова стая}”**.

### Ограничения

* **Индексите** на стаите винаги ще са числата **[0…N-1]**.
* **Предверието** винаги ще е стаята с индекс **0**.
* **Съкровищницата** винаги ще е стаята с индекс **N-1**.
* Броят на **стаите** ще е между **[3…20 000]**.
* Броят на **тунелите** ще е между **[2…30 000]**.
* Винаги ще има път от **предверието** до **съкровищницата**.
* Винаги ще има **поне една ключова стая**.
* Всички **тунели** са **двупосочни**.
* Позволено време: **100 мс**. Позволена памет: **32 MB**.

### Примерен вход и изход

|  |  |  |
| --- | --- | --- |
| **Вход** | **Визуализация** | **Изход** |
| 6  6  0 1  0 2  1 2  1 3  3 4  3 5 | **0**  **2**  **5**  **1**  **4**  **3** | 4  1->3 |
| **Коментари** | | |
| Започваме от **предверието** (**#0**) и искаме да стигнем **съкровищницата** (**#5**), като погледнем картата е очевидно че който и път да изберем, ще трябва да минем през стаи **#1** и **#3** – съответно е очевидно че ключовите стаи в този лабиринт са стаи **#1** и **#3**. Нека да разгледаме възможните пътища от **предверието** до **съкровищницата**.  Път 1:  **0**->**1**->**3**->**5**  Път 2:  **0**->2->**1**->**3**->**5**  След като знаем **ключовите стаи** е време да намерим **тайният ключ** – събираме **индексите** на **ключовите стаи** и получаваме 1 + 3 = **4**.Виждаме че независимо от пътя винаги първо ще трябва да стъпим в стая **#1** след което в стая **#3**, съответно имаме и правилната подредба на ключвете стаи **1->3**, с което приключваме задачата. | | |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Визуализация** | **Изход** |
| 10  11  0 6  6 7  6 1  7 4  7 8  1 2  1 4  2 3  2 5  3 4  5 9 | **0**  **7**  **9**  **5**  **2**  **4**  **3**  **1**  **6**  **8** | 13  6->2->5 |

|  |  |  |
| --- | --- | --- |
| **Вход** | **Визуализация** | **Изход** |
| 12  15  0 1  0 3  0 4  1 2  1 10  2 10  3 4  4 5  4 8  8 9  5 6  5 7  6 7  11 6  7 11 | **0**  **1**  **2**  **3**  **4**  **7**  **11**  **8**  **9**  **5**  **6**  **10** | 9  4->5 |