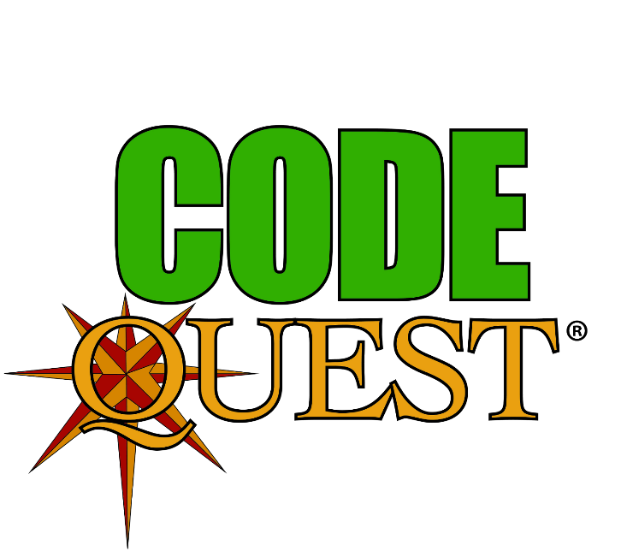


**2019**

**Pakiet problemów**

**DO ROZPOCZĘCIA KONKURSU NIE WOLNO OTWIERAĆ PAKIETU**



**Pakiet problemów 2019**

|  |  |  |  |
| --- | --- | --- | --- |
| **Lp.** | **Nazwa problemu** | **Punkty** | **Strona** |
| **1** | Nie krzyczmy | 5 | 8 |
| **2** | Podsumujmy | 5 | 9 |
| **3** | Kłopotliwe goryle | 5 | 10 |
| **4** | Mandat za prędkość | 10 | 11 |
| **5** | Dom z cegły | 10 | 13 |
| **6** | Dookoła świata | 15 | 14 |
| **7** | Kompresowanie obrazu | 15 | 16 |
| **8** | Renderowanie miejsca fiksacji oka | 15 | 18 |
| **9** | W pułapce czasu | 20 | 21 |
| **10** | Szyfr Cezara | 20 | 23 |
| **11** | Licz do 10 | 25 | 25 |
| **12** | Monty Hall | 25 | 26 |
| **13** | Saper | 25 | 30 |
| **14** | Powrót do domu | 30 | 32 |
| **15** | LMCoin | 35 | 34 |
| **16** | Gdzie są moje klucze? | 35 | 37 |
| **17** | Gra w życie Conwaya | 40 | 40 |
| **18** | Zbiór Mandelbrota | 45 | 43 |
| **19** | Strażnik sieci | 50 | 46 |
| **20** | Obserwacja ptaków | 55 | 49 |
| **21** | Ukrywaj swoich szpiegów | 55 | 53 |
| **22** | Czyszczenie dysku | 60 | 56 |
| **23** | Ewakuacja! | 70 | 58 |
| **24** | Sudoku | 80 | 61 |

*Kopie opisów problemów są również dostępne na stronie konkursowej.*

# Często zadawane pytania (FAQ)

**Jakie są zasady konkursu?**

W celu rozwiązania danego problemu wasz zespół musi napisać program komputerowy, który odczyta dane wejściowe ze standardowego wejścia i wyświetli wynik na konsoli. Każdy problem zawiera informacje o formacie danych wejściowych i przewidywanym formacie danych wyjściowych (wyniku). Po ukończeniu programu należy przesłać jego przetestowany kod źródłowy za pomocą strony konkursowej. Strona kompiluje i uruchamia kod, po czym informuje, czy odpowiedź jest prawidłowa czy nie.

**Kto ocenia odpowiedzi?**

Za ocenę odpowiedzi konkursowych odpowiada zespół pracowników Lockheed Martin, ale większość tego procesu jest realizowana automatycznie przez stronę konkursową. Strona konkursowa kompiluje i uruchamia kod, a następnie porównuje wynik dostarczony przez wasz program do oficjalnego wyniku przewidzianego przez organizatorów. Jeśli dane wyjściowe waszego programu są zgodne z oficjalną odpowiedzią, wasz zespół otrzyma punkty za poprawną odpowiedź. Nasz zespół sędziowski monitoruje pracę strony konkursowej, ale w większości przypadków zostanie podtrzymana ocena uzyskana w trakcie automatycznego procesu.

**Jak punktowane są problemy?**

Każdy problem ma przydzieloną wartość punktową w zależności od poziomu trudności. Po uruchomieniu przesłanego przez was programu strona konkursowa porówna podawany przez niego wynik do wyniku oficjalnego. Jeśli są one identyczne, wasz zespół otrzyma punkty za rozwiązanie problemu. Nie przewiduje się punktacji cząstkowej; wyniki muszą być *identyczne*. Jeśli otrzymacie informację, że odpowiedź jest nieprawidłowa, a jesteście pewni, że była poprawna, sprawdźcie ponownie format danych wyjściowych i upewnijcie się, czy nie ma w nich np. spacji na zakończenie wiersza czy innych niepotrzebnych znaków.

**Nie rozumiemy problemu. Kto może nam udzielić pomocy?**

Jeśli macie trudności ze zrozumieniem problemu, możecie zwrócić się z pytaniem do zespołu objaśniającego problemy za pośrednictwem strony konkursowej. Co prawda, nie możemy sugerować, jak rozwiązać problem, ale możemy objaśniać wszelkie niejasności. Jeśli zespół objaśniający problemy zauważy podczas konkursu błąd w jakimkolwiek problemie, jak najszybciej wyślemy powiadomienie w tej sprawie do wszystkich uczestników.

**Nasz program działa z przykładowymi danymi wejściowymi/wyjściowymi, ale nadal jest oznaczany jako nieprawidłowy! Dlaczego?**

Należy pamiętać, że oficjalne dane wejściowe i wyjściowe służące do oceny odpowiedzi są ZNACZNIE większe od przekazanych przykładowych danych. Obejmują one szerszy zakres przypadków testowych. Opis problemu zawiera informację o wartościach granicznych tych danych, ale jednak wasz program musi umieć przetworzyć każdy przypadek testowy mieszczący się w tych granicach. Wszystkie dane wejściowe i wyjściowe zostały gruntownie przetestowane przez nasz zespół ds. problemów i nie zawierają nieprawidłowości.

**Nie umiemy ustalić, dlaczego nasza odpowiedź jest nieprawidłowa. Gdzie popełniamy błąd?**

Do najczęstszych błędów należą:

* Nieprawidłowe formatowanie - Sprawdźcie ponownie przykładowe dane wyjściowe podane w opisie problemu i sprawdźcie, czy wasz program podaje wynik w poprawnym formacie.
* Nieprawidłowe zaokrąglanie - Więcej informacji o zaokrąglaniu ułamków dziesiętnych podano w następnym podrozdziale.
* Nieprawidłowe liczby - 0 (lub też 0,0, 0,00 itd.) NIE jest liczbą ujemną. 0 może być dopuszczalną odpowiedzią, ale -0 nie.
* Niepotrzebne znaki - Należy sprawdzić, czy na końcu wiersza nie znajduje się niepotrzebna spacja. Nie mogą się one znajdować w żadnym wyniku.
* Format dziesiętny - We wszystkich liczbach znakiem rozdzielającym część dziesiętną od podstawy jest kropka (.).

Jeśli te podpowiedzi nie pomogły, możecie zwrócić się z pytaniem do zespołu objaśniającego problemy za pośrednictwem strony konkursowej. Nie możemy sugerować, jak rozwiązać problem, ale możemy wyjaśnić, dlaczego odpowiedź jest zwracana jako nieprawidłowa.

**Podczas przesyłania rozwiązania pojawił się komunikat błędu.**

Przesyłając rozwiązanie należy wybrać kod źródłowy dla swojego programu (w zależności od języka może to być plik .java, .c, .h, .cpp, .py lub .vb). Należy koniecznie przesłać wszystkie pliki potrzebne do skompilowania i uruchomienia programu. Ponadto, nazwy plików nie mogą zawierać spacji ani znaków spoza zakresu klawiszy alfanumerycznych w tym polskich znaków (np. „Prob01.java” jest dopuszczalne, ale „Prob 01.java” i „Bob’sSolution.java” nie są).

**Czy po konkursie możemy otrzymać odpowiedzi?**

Oczywiście! Po konkursie członek naszego zespołu ds. problemów będzie odpowiadał na wszelkie pytania (i omawiał przesłane nieprawidłowe odpowiedzi, aby wyjaśnić, gdzie występował błąd). Jeśli chcielibyście otrzymać kopię rozwiązań problemów lub też rozwiązania przesłane przez inne zespoły, poproście trenera o przesłanie emaila na adres naszego kierownika ds. problemów, Bretta Reynoldsa, czyli [brett.w.reynolds@lmco.com](mailto:brett.w.reynolds@lmco.com). Możemy również dostarczyć kopie oficjalnych danych wejściowych i wyjściowych używanych do oceny waszych rozwiązań.

**Co w przypadku remisu?**

Na koniec konkursu zespoły są układane w kolejności, odpowiednio do liczby punktów zdobytych za poprawne odpowiedzi na problemy konkursowe. Jeśli na trzech czołowych pozycjach znajdują się zespoły z identycznym dorobkiem punktowym, to kryteria ustalania kolejności są następujące.

* Najmniej rozwiązanych problemów (to zwykle oznacza, że zostały przesłane rozwiązania trudniejszych problemów)
* Najmniej niepoprawnych odpowiedzi (to oznacza mniej popełnionych błędów)
* Zespół, który najwcześniej przesłał ostatnią prawidłową odpowiedź (to oznacza szybszą pracę)

Należy pamiętać, że te kryteria ustalania kolejności w przypadku remisu nie zawsze są widoczne na stale aktualizowanej tablicy wyników widocznej na stronie konkursowej. Ponadto, aktualizacja tablicy wyników zostaje zatrzymana na 30 minut przed końcem konkursu, dlatego należy skupić się na jak najlepszej pracy.

# Zaokrąglanie

W niektórych problemach w rozwiązaniu należy podać zaokrąglone liczby. O ile w opisie problemu nie podano inaczej, należy zawsze zaokrąglać od połowy w górę. Najprawdopodobniej dokładnie takiej metody uczono was w szkole, ale niektóre języki programowania domyślnie stosują inne metody zaokrąglania. **Jeśli nie jesteście pewni, jak wasz język programowania zaokrągla liczby, zalecamy napisanie własnego kodu do zaokrąglania liczb, korzystając przy tym z informacji podanych w tym podrozdziale.**

Przy zaokrąglaniu od połowy w górę liczby są zaokrąglane do najbliższej liczby całkowitej. Na przykład:

* 1,49 zaokrąglamy do 1
* 1,51 zaokrąglamy do 2

Metoda zaokrąglania od połowy w górę oznacza, że gdy dana liczba jest dokładnie pośrodku, to jest zaokrąglana do liczby o najwyższej wartości bezwzględnej (czyli położonej na skali najdalej od 0). Na przykład:

* 1,5 zaokrąglamy do 2
* -1,5 zaokrąglamy do -2

Błędy przy zaokrąglaniu są dość powszechne; jeśli w problemie występuje zaokrąglanie, a strona konkursowa informuje, że odpowiedź jest nieprawidłowa, ponownie sprawdźcie zaokrąglanie!

# Terminologia

W całym pakiecie wielokrotnie opisujemy dane wejściowe i wyjściowe dla waszych programów. Aby uniknąć nieporozumień, do określania różnych właściwości tych danych stosujemy pewne pojęcia. Poniżej przedstawiamy ich definicje.

* **Liczba całkowita** to dowolna liczba niebędąca ułamkiem, czyli niezawierająca części dziesiętnej lub ułamkowej: -5, 0, 5 czy 123456789 to liczby całkowite.
* **Liczba dziesiętna** to dowolna liczba niebędąca liczbą całkowitą. Te liczby zawierają separator dziesiętny (w Polsce przecinek, natomiast w danych konkursowych jest to kropka, stosowana w krajach anglosaskich), a za nim co najmniej jedną cyfrę. -1.52, 0.0 czy 3.14159 to liczby dziesiętne.
* **Miejsca dziesiętne** odnoszą się do cyfr w liczbie dziesiętnej umiejscowionych za separatorem dziesiętnym. O ile nie podano inaczej w opisie problemu, liczby dziesiętne mogą zawierać dowolną liczbę miejsc dziesiętnych, równą co najmniej 1.
* **Liczba szesnastkowa** lub **łańcuch szesnastkowy** korzystający ze znaków obejmujących cyfry od 0 do 9 i/lub wielkie litery A, B, C, D, E i F. W konkursie nie stosuje się małych liter do wartości podawanych w systemie szesnastkowym.
* **Liczby dodatnie** to liczby większe od 0. Najmniejszą dodatnią liczbą całkowitą jest 1; z kolei 0.000000000001 to bardzo mała dodatnia liczba dziesiętna.
* **Liczby niedodatnie** to wszystkie liczby niebędące liczbami dodatnimi, to jest, wszystkie liczby mniejsze od 0 wraz z 0.
* **Liczby ujemne** to liczby mniejsze od 0. Największą ujemną liczbą całkowitą jest -1; z kolei - 0,000000000001to bardzo duża liczba ujemna.
* **Liczby nieujemne** to wszystkie liczby niebędące liczbami ujemnymi, to jest, wszystkie liczby większe od 0 wraz z 0.
* **Włącznie** oznacza, że przedział określony przez podane wartości obejmuje również obie te wartości (tzw. przedział domknięty). Na przykład, przedział od 1 do 3 włącznie zawiera liczby 1, 2 i 3.
* **Rozłącznie** oznacza, że przedział określony przez podane wartości nie obejmuje tych wartości (tzw. przedział otwarty). Na przykład, przedział od 0 do 4 rozłącznie zawiera liczby 1, 2 i 3; 0 i 4 nie należą do przedziału.
* **Format daty i godziny** jest podawany literami, które zastępuje się odpowiednio cyframi:
  + **HH** oznacza godziny, zapisywane dwoma cyframi (z poprzedzającym zerem w razie potrzeby). Opis problemu zawiera informację, czy stosowany jest 12 czy 24 godzinny format.
  + **MM** oznacza minuty w przypadku czasu lub miesiące w przypadku daty. W obydwu przypadkach są one zapisywane dwoma cyframi (z poprzedzającym zerem w razie potrzeby). Styczeń to miesiąc 01.
  + **RR** lub **RRRR** (w oryginalnym zapisie **YY** lub **YYYY**) oznacza rok, zapisywany dwoma lub czterema cyframi (z poprzedzającym zerem w razie potrzeby).
  + **DD** to dzień miesiąca, zapisywany dwoma cyframi (z poprzedzającym zerem w razie potrzeby).

# Problem 1: Nie krzyczmy

**Punkty:** 5

**Autor:** Brett Reynolds, Orlando, Floryda, Stany Zjednoczone

## Wprowadzenie do problemu

Powszechnie wiadomo, że PISANIE W INTERNECIE WIELKIMI LITERAMI JEST NIEGRZECZNE. Wygląda to tak, jakbyśmy krzyczeli na swoich rozmówców, co raczej nie jest dobrym pomysłem na podtrzymanie rozmowy. Chcielibyśmy was prosić o stworzenie rozszerzenia do przeglądarki, które (w sposób wymuszony) zaprowadza spokój zamieniając KRZYK na szept (czyli WIELKIE LITERY na małe). Podczas rozwiązywania problemu postarajcie się zachować spokój.

## Opis problemu

Do waszego programu otrzymacie wiersze tekstu, w których użyto wyłącznie wielkich liter. Musicie zamienić je na małe, nie zmieniając przy tym treści.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz tekstu składającego się z małych liter, liczb, spacji i/lub znaków interpunkcyjnych.

2

THIS SENTENCE IS IN ALL CAPS

SHOUTING ISN’T NICE.

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić podany łańcuch, w którym wielkie litery zastąpiono małymi. Spacje, liczby i znaki interpunkcyjne nie powinny ulec zmianie.

this sentence is in all caps

shouting isn’t nice.

# Problem 2: Podsumujmy

**Punkty:** 5

**Autor:** Shelly Adamie, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

Dodawanie liczb jest bardzo proste, chyba że podkręcimy poziom trudności. Jeśli liczby są identyczne, zsumujcie ich sumy!

## Opis problemu

Do waszego programu otrzymacie dwie liczby. Jeśli są różne, program powinien zwrócić ich sumę. Jeśli są równe, program powinien zwrócić dwukrotność ich sumy.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz składający się z dwóch nieujemnych liczb całkowitych oddzielonych spacjami.

5

1 3

2 2

3 2

13 13

125 9

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić wartość obliczoną zgodnie z podanymi powyżej regułami.

4

8

5

52

134

# Problem 3: Kłopotliwe goryle

**Punkty:** 5

**Autor:** Shelly Adamie, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

W miejscowym zoo najpopularniejszym obiektem odwiedzin jest wybieg dwóch goryli. Ale ta para sprawia pewne problemy dozorcom. W przypadku kłopotów na wybiegu goryli musimy zaalarmować dozorców.

## Opis problemu

Do waszego programu otrzymacie informację, czy goryle się uśmiechają czy też nie. Musimy informować dozorców, czy obydwa goryle się uśmiechają (co może oznaczać, że coś knują) lub czy żaden się nie uśmiecha (co może oznaczać, że zaraz się pobiją). Jeśli tylko jeden z goryli prezentuje uśmiech, najprawdopodobniej wszystko jest w porządku.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz składający się z dwóch wartości logicznych („prawda” (ang. „true”) lub „fałsz” (ang. „false”)) oddzielonych spacjami.

2

true false

true true

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić „true”, jeśli należy powiadomić dozorców o potencjalnych kłopotach lub „false”, jeśli wszystko jest w porządku.

false

true

# Problem 4: Mandat za prędkość

**Punkty:** 10

**Autor:** Holly Norton, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

Trochę za bardzo dodałeś gazu w swoim wozie i zostałeś zatrzymany przez policjanta. Funkcjonariusz zastanawia się nad wysokością twojego mandatu, ale masz szczęście: postanowił, że podaruje ci ten wybryk, jeśli masz dziś urodziny.

## Opis problemu

Wasz program powinien obliczyć kwotę mandatu, w zależności od prędkości prowadzonego samochodu:

* Jeśli ta prędkość wynosiła nie więcej niż 60, policjant nie wystawi mandatu.
* Jeśli wynosiła od 61 do 80 włącznie, mandat będzie niewielki.
* Jeśli ta prędkość wynosiła więcej niż 81, mandat będzie wysoki.

Jeśli dziś są twoje urodziny, wszystkie te liczby zostają powiększone o 5 (na przykład, jazda z prędkością nie większą niż 65 pozwoli uniknąć mandatu).

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz, zawierający dwie wartości oddzielone spacjami:

* Dodatnia liczba całkowita, będąca prędkością prowadzonego samochodu
* Wartość „true” oznaczająca, że dziś są twoje urodziny lub wartość „false” oznaczająca, że nie masz szczęścia.

3

60 false

82 false

83 true

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić pojedynczy wiersz zgodnie z poniższymi regułami:

* „No ticket”, jeśli policjant nie wystawia mandatu,
* „Small ticket”, jeśli mandat jest niewielki,
* „Big ticket”, jeśli mandat jest wysoki.

no ticket

big ticket

small ticket

# Problem 5: Dom z cegły

**Punkty:** 10

**Autor:** Holly Norton, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

Chcemy ułożyć rząd cegieł w aktualnie budowanym ceglanym domu. Długość tego rzędu cegieł wynosi określoną liczbę cali, a do dyspozycji mamy pewną liczbę małych i dużych cegieł. Waszym zadaniem jest napisać aplikację, która odpowie na pytanie, czy ten rząd cegieł można ułożyć wykorzystując część posiadanych cegieł lub wszystkie z nich. Wykorzystanie wszystkich posiadanych cegieł nie jest obowiązkowe!

## Opis problemu

Do waszego programu otrzymacie docelową długość ceglanej ściany oraz liczbę dostępnych małych i dużych cegieł. Małe cegły mają 1 cal długości. Duże cegły mają 5 cali długości. Należy ustalić, czy da się ułożyć rząd cegieł o zadanej długości z użyciem dostępnych cegieł.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz, zawierający trzy nieujemne liczby całkowite oddzielone spacjami.

* Pierwsza liczba całkowita to liczba dostępnych małych, jednocalowych cegieł
* Druga liczba całkowita to liczba dostępnych dużych, pięciocalowych cegieł
* Trzecia liczba całkowita to docelowa długość ściany **X** (w calach)

3

3 1 8

3 1 9

3 2 10

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić pojedynczy wiersz ze słowem „true”, jeśli da się zbudować ścianę o długości **X** cali wyłącznie z dostępnych cegieł. W przeciwnym wypadku wynikiem powinno być słowo „false”.

true

false

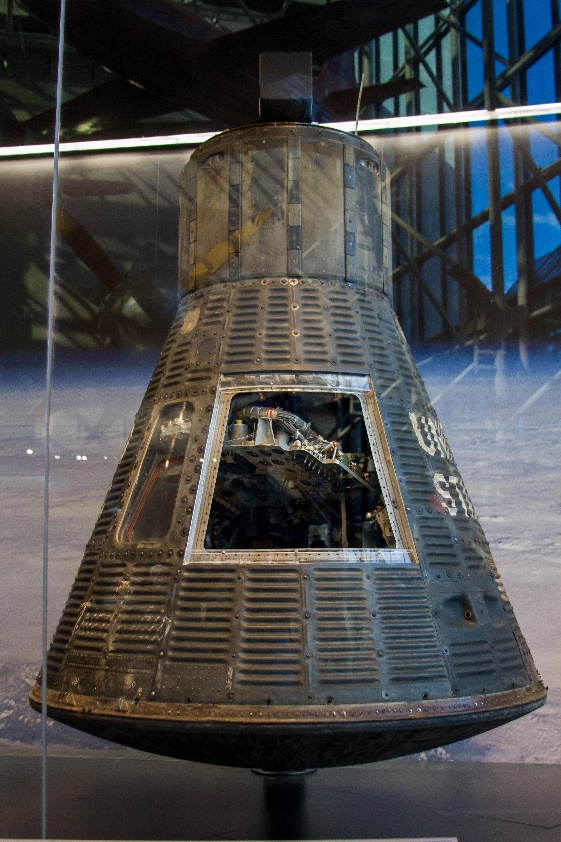
true

# Problem 6: Dookoła świata

**Punkty:** 15

**Autor:** Chris Mason, Sunnyvale, Kalifornia, Stany Zjednoczone

## Wprowadzenie do problemu

W 1962 roku John Glenn dokonał historycznego wyczynu w dziedzinie lotów kosmicznych, trzykrotnie okrążając Ziemię w niewielkim statku kosmicznym. Ten lot był jednym z wielu, które utorowały drogę do epoki odkrywania przestrzeni kosmicznej, która została zwieńczona lądowaniem na Księżycu po zaledwie siedmiu latach. Pomimo pozornie prostej charakterystyki lotu Glenna ta podróż wymagała bardzo precyzyjnych obliczeń gwarantujących, że pozostanie on na orbicie okołoziemskiej i nie odleci w przestrzeń kosmiczną ani nie rozbije się przy powrocie na Ziemię.

Obiekty na orbicie nie pozostają w przestrzeni kosmicznej tylko dlatego, że opuściły atmosferę ziemską; przyciąganie ziemskie powoduje, że stale opadają. Natomiast po uzyskaniu odpowiednio dużej prędkości ich trajektoria lotu zbliża się do krzywizny kuli ziemskiej, dzięki czemu pozostają na orbicie. Podczas swojego historycznego lotu John Glenn poruszał się z prędkością orbitalną 17 544 mil na godzinę (28 234,8 kilometrów na godzinę). Taka prędkość pozwoliłaby odbyć podróż z Nowego Jorku do Londynu w niecałe 12 minut. W trakcie całego lotu, trwającego nieco poniżej pięciu godzin, Glenn przebył odległość 75 679,3 mili (121 794 kilometry).

Wasze zadanie to ustalić, jaką odległość może przebyć obiekt na orbicie okołoziemskiej, na określonej wysokości, podczas jednokrotnego okrążenia.

## Opis problemu

Do waszego programu otrzymacie pułap (wysokość nad poziomem morza) obiektu okrążającego Ziemię wzdłuż równika. Z pomocą tej informacji wasz program musi obliczyć całkowitą odległość przebytą przez obiekt podczas pojedynczego okrążenia Ziemi. Przydatna może być przy tym informacja, że długość równika to 40 075 kilometrów.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz, zawierający liczbę całkowitą stanowiącą wysokość obiektu nad poziomem morza (w kilometrach). Wysokość ta będzie nie niższa niż 160 (najniższy możliwy pułap orbitalny kuli ziemskiej).

3

160

200

265

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić odległość przebytą przez obiekt okrążający Ziemię wzdłuż równika na podanym pułapie (w kilometrach). Każdą wartość należy zaokrąglić do najbliższej części dziesiętnej kilometra (jedno miejsce dziesiętne).

41080.3

41331.6

41740.0

# Problem 7: Kompresowanie obrazu

**Punkty:** 15

**Autor:** Steve Brailsford, Marietta, Georgia, Stany Zjednoczone

## Wprowadzenie do problemu

Obrazy można zapisywać na komputerze w przeróżnych formatach, z których każdy ma swoje wady i zalety. Pliki JPEG (lub JPG) są powszechnie stosowane do fotografii, ponieważ ich format umożliwia skompresowanie informacji o obrazie, co zmniejsza wielkość pliku (i umożliwia zrobienie większej liczby zdjęć). Wadą tego rozwiązania jest fakt, że wielokrotna edycja pliku JPEG powoduje stopniowe obniżanie jakości obrazu; po każdym jego zapisaniu dane są ponownie kompresowane, w rezultacie czego tracone są drobne detale.[[1]](#footnote-1)

Proces kompresowania plików JPEG jest skomplikowany, ale można go podzielić na kilka etapów. Jednym z nich jest kwantyzacja, która przetwarza duże przedziały liczb utworzonych podczas poprzedniego etapu, zamieniając je na mniejsze, w skali łatwiejszej do obsługi. Jak wspomniano powyżej, w ten sposób dochodzi do utraty szczegółów obrazu; dwie różne, ale zbliżone liczby są zamieniane na jedną i tę samą liczbę. Niemniej ludzkie oko nie jest w stanie wychwycić zmian o bardzo wysokiej częstotliwości, zatem takie straty są zwykle niezauważalne.

## Opis problemu

Wasz program powinien zaimplementować przykładowy algorytm kwantyzacji, który odczytuje wartości jasności i zamienia je na liczbę całkowitą w przedziale od 0 do 255 włącznie. Otrzymacie do niego listę wartości dziesiętnych reprezentujących parametry jasności (jakie mogłyby być odczytywane przez skaner). Wasz program musi znaleźć największą (maksymalną) i najmniejszą (minimalną) wartość na liście, a następnie zamienić wszystkie wartości na liście na skalę docelową z wykorzystaniem poniższego równania:

Wszystkie wyniki należy zaokrąglić do najbliższej liczby całkowitej.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy ma zawierać następujące wiersze danych wejściowych:

* Dodatnia liczba całkowita, **X**, reprezentująca liczbę wartości na liście
* **X** wierszy, z których każdy zawiera liczbę dziesiętną przeznaczoną do zamiany

2

5

0.0

25.0

50.0

75.0

100.0

6

12.3

-67.1

122.8

428.4

-15.9

221.0

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić listę zamienionych liczb, z zachowaniem tej samej kolejności. W każdym wierszu powinna się znajdować jedna liczba, zaokrąglona do najbliższej liczby całkowitej.

0

64

128

191

255

41

0

98

255

26

148

# Problem 8: Renderowanie miejsca fiksacji oka

**Punkty:** 15

**Autor:** Gary Hoffmann, Denver, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

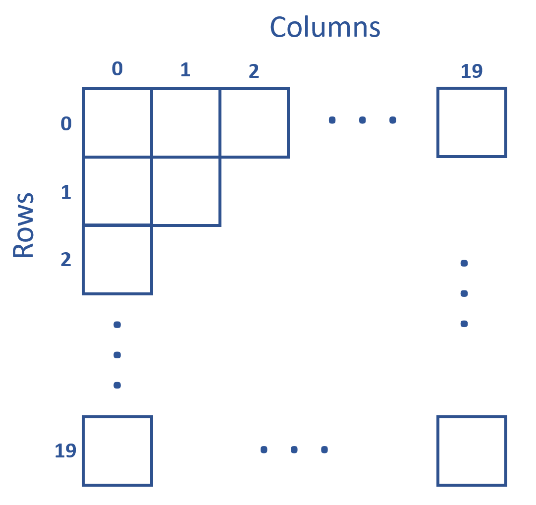
W ostatnich pięciu latach Wirtualna Rzeczywistość wkroczyła z przytupem na rynek, znajdując zastosowanie w wielu dziedzinach, od gier i rozrywki przez projektowanie wyrobów do inżynierii. Jednym z najnowszych osiągnięć w projektowaniu zestawów VR jest dodanie funkcji śledzenia oka, która rozbudowuje działanie zestawu.

Charakterystyczną cechą ludzkiego wzroku jest fakt, że pole widzenia, w jakim występuje pełna ostrość a także możliwe jest rozpoznanie drobnych szczegółów jest bardzo wąskie. Ostrość widzenia jest zależna od dołka środkowego siatkówki oka (ang. fovea), małego wgłębienia w siatkówce oka, które specjalizuje się w tym zakresie. Jednak ze względu na rozmiar dołka środkowego ludzkie oko postrzega otoczenie wystarczająco wyraźnie jedynie w polu widzenia o szerokości kątowej poniżej 10°. Resztą zajmuje się nasz mózg, który zapełnia puste miejsca obrazami rejestrowanymi przez nas przy rozglądaniu się.

Z tego względu zestaw VR musi renderować obrazy o najwyższej rozdzielczości jedynie w tym miejscu, w które patrzy użytkownik. Obrazy poza tym polem widzenia mogą być renderowane z mniejszą jakością, co poprawia wydajność systemu.

## Opis problemu

Waszym zadaniem jest napisanie modułu do aplikacji Wirtualnej Rzeczywistości, który będzie określać jakość renderowania dla każdego fragmentu ekranu zestawu VR. Dla uproszczenia, wasz moduł będzie uwzględniał jedno oko dla pojedynczego ekranu. Ekran będzie podzielony na siatkę o długości boku wynoszącej 20 pól.

Do waszego programu otrzymacie współrzędne pola umieszczonego na siatce, na którym aktualnie użytkownik skupił swój wzrok, a wynik powinien zawierać poziom renderowania dla każdego pola siatki, wiersz po wierszu.

Pole, w które aktualnie patrzy użytkownik powinno być renderowane z pełną jakością czyli 100%. Wszystkie pola graniczące z polem określonym powyżej powinny być renderowane z połowiczną jakością (50%), natomiast pola graniczące z nimi powinny być renderowane z jakością 25%. Cała reszta powinna być renderowana na minimalnym poziomie 10%.

Na przykład, jeśli użytkownik patrzy w pole w wierszu 7 i kolumnie 10, to jakość renderowania każdego pola siatki będzie następująca:

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz składający się z dwóch liczb całkowitych oddzielonych spacjami, które będą reprezentować numer rzędu i numer kolumny pola siatki, na którym użytkownik skupia swój wzrok. Numery rzędów i kolumn mieszczą się w przedziale od 0 do 19 włącznie.

2

7 10

0 0

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić procentową wartość jakości renderowania dla każdego pola siatki. Każdy rząd powinien mieścić się w oddzielnym wierszu, a kolumny powinny być oddzielone spacjami.

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 25 25 25 25 25 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 25 50 50 50 25 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 25 50 100 50 25 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 25 50 50 50 25 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 25 25 25 25 25 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

100 50 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

50 50 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

25 25 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

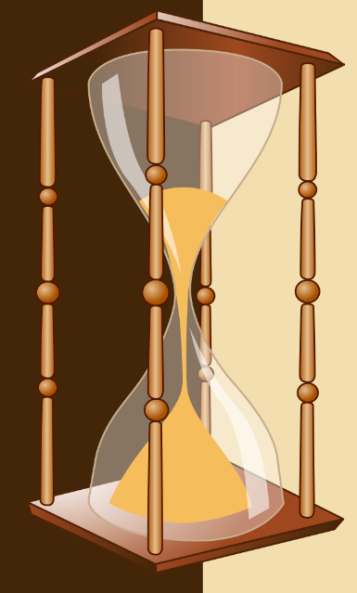
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

# Problem 9: W pułapce czasu

**Punkty:** 20

**Autor:** Jonathan Brown, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

Są różne sposoby na określanie czasu i jego odcinków. Różnice krajowe, regionalne czy nawet osobiste doprowadziły do powstania szerokiej gamy formatów, które są stosowane przy określaniu czasu. Z tego powodu panuje spore zamieszanie: czy 01/03 to 3 stycznia czy 1 marca... a może styczeń 2003 r.? Czy 8:45 to ranek czy wieczór?

Waszym zadaniem będzie uporządkowanie tego galimatiasu poprzez zamianę listy wartości czasowych na nowy, spójny format.

## Opis problemu

Do waszego programu otrzymacie listę odcinków czasu, które są określone godzinami, minutami i/lub sekundami.

* Godziny będą podane jako nieujemne liczby całkowite z dodaną małą literą „h” (np. 2h). Ich zakres to od 0 do 99 włącznie.
* Minuty będą podane jako nieujemne liczby całkowite z dodaną małą literą „m” (np. 2m). Ich zakres to od 0 do 59 włącznie.
* Sekundy będą podane jako nieujemne liczby całkowite z dodaną małą literą „s” (np. 2s). Ich zakres to od 0 do 59 włącznie.

Niekoniecznie będą one podawane w tej kolejności. Będą oddzielane spacjami, przecinkami i/lub spójnikiem „and”; wyraz ten należy zignorować. Niekiedy może brakować jednej z kategorii; na przykład, będą podane tylko minuty i sekundy. W przypadku pominięcia jakichkolwiek wartości należy założyć, że wynoszą zero.

Niezależnie od dostarczonych informacji wasz program powinien wyświetlać odcinki czasowe w prostym i spójnym formacie:

HH:MM:SS

W tym formacie HH to dwucyfrowa liczba oznaczająca godziny (w razie potrzeby z zerem poprzedzającym). MM to dwucyfrowa liczba oznaczająca minuty (w razie potrzeby z zerem poprzedzającym). Z kolei SS to dwucyfrowa liczba oznaczająca sekundy (również w razie potrzeby z zerem poprzedzającym). Każda liczba jest oddzielana od poprzedniej dwukropkiem, a kolejność wyświetlania jest niezmienna. W wyniku muszą się zawierać wszystkie liczby, nawet te równe 0.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz danych wejściowych z łańcuchem określającym pewny odcinek czasowy, przy czym mogą one mieć różny format, zgodnie z poniższym przykładem.

5

1 m and 45s

10m,10s

32s, and 12h

76h

1s

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić w pojedynczym wierszu ten sam odcinek czasowy w formacie HH:MM:SS, zgodnie z poniższym przykładem.

00:01:45

00:10:10

12:00:32

76:00:00

00:00:01

# Problem 10: Szyfr Cezara

**Punkty:** 20

**Autor:** Steve Gerali, Denver, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

Szyfr Cezara to jeden z najwcześniejszych znanych nam szyfrów, a w dodatku jeden z najprostszych do nauki. To „szyfr podstawieniowy”, w którym każdą literę tekstu niezaszyfrowanego przesuwa się o określoną liczbę miejsc do przodu w alfabecie. Na przykład, w przypadku przesunięcia równego 1, zamiast A podstawia się B, zamiast B C itd. Metoda została nazwana na część Juliusza Cezara, który stosował ją do komunikowania się ze swoimi generałami.

Aby możliwe było przekazanie zaszyfrowanej wiadomości, zarówno przekazujący, jak i odbierający muszą znać klucz szyfru, aby przekazujący mógł go zaszyfrować, a odbierający odszyfrować. W przypadku szyfru Cezara kluczem jest liczba liter, o którą należy przesunąć alfabet.

## Opis problemu

Pracujecie dla telewizji historycznej, która chce odszyfrować wszystkie komunikaty przesłane przez Juliusza Cezara do jego generałów, aby wykorzystać je w nowym dokumencie o rzymskim imperatorze. Otrzymacie listę zaszyfrowanych wiadomości i klucz, który prawdopodobnie został użyty do ich szyfrowania. Wasz program musi odszyfrować wiadomości.

Na potrzeby problemu będziemy stosować alfabet angielski, przedstawiony poniżej w standardowym porządku (z przesunięciem 0).

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Po zaszyfrowaniu wiadomości z przesunięciem 1 każda litera w tekście niezaszyfrowanym zostanie zastąpiona odpowiednią literą podaną w poniższym alfabecie przesuniętym o 1.

B C D E F G H I J K L M N O P Q R S T U V W X Y Z A

Podczas odszyfrowywania wiadomości proces jest odwracany: litera w tekście zaszyfrowanym jest zastępowana odpowiednią literą z normalnego alfabetu angielskiego.

Spacje nie są szyfrowane i podczas odszyfrowywania pozostają na swoim miejscu.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać dwa wiersze:

* Wiersz z pojedynczą liczbą całkowitą oznaczającą klucz wiadomości - liczbę liter, o jaką alfabet został przesunięty podczas szyfrowania
* Wiersz zawierający małe litery i spacje, stanowiący zaszyfrowaną wiadomość

3

1

buubdl bu ebxo

3

ghvwurb wkh fdvwoh

6

yzkgr znk ynov

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić odszyfrowaną wiadomość. Wiadomości powinny być wyświetlane małymi literami, z zachowanymi spacjami.

attack at dawn

destroy the castle

steal the ship

# Problem 11: Licz do 10

**Punkty:** 25

**Autor:** Ryan Regensburger, Huntsville, Alabama, Stany zjednoczone

## Wprowadzenie do problemu

Podczas testowania programów lub sprzętu komputerowego dobrą praktyką jest przetestowanie każdej możliwej sytuacji, by dowieść, że kod lub urządzenie będzie stabilnie pracować w każdych warunkach. Na przykład, jeśli mamy chip z ośmioma diodami LED, warto zaświecić je w każdej możliwej kombinacji, by przetestować ich działanie. Zasadniczo jest to 8-bitowy licznik dwójkowy, który wyświetla każdą liczbę z przedziału od 0 do 255.

## Opis problemu

Waszym zadaniem będzie wygenerowanie danych testowych do licznika dwójkowego podobnego do opisanego powyżej. Dostaniecie liczbę bitów do wykorzystania w liczniku, a wynik powinien zawierać listę wszystkich liczb w układzie dwójkowym w rosnącej kolejności.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz z dodatnią liczbą całkowitą, stanowiącą liczbę bitów do wykorzystania.

1

3

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić listę liczb w układzie dwójkowym, z przedziału od 0 do maksymalnej wartości w podanej liczbie bitów włącznie. Liczby muszą być umieszczone po jednej w każdym wierszu, w porządku rosnącym. Należy użyć zer poprzedzających, aby każdy bit miał wymaganą długość.

000

001

010

011

100

101

110

111

# Problem 12: Monty Hall

**Punkty:** 25

**Autor:** Christian Lin, Greenville, Karolina Południowa, Stany Zjednoczone

## Wprowadzenie do problemu

Paradoks Monty/ego Halla to problem statystyczny, którego nazwa pochodzi od pierwszego prowadzącego amerykańskiej wersji teleturnieju „Idź na całość” (czyli „Let/s Make a Deal!”). W jednym z najsłynniejszych fragmentów programu gospodarz proponuje uczestnikowi wybór jednej z trzech bramek. Za jedną z nich był samochód, natomiast za dwoma pozostałymi kozy. Uczestnik wybiera bramkę - na przykład, bramkę numer 1 - a gospodarz, wiedząc, co kryje się za każdą z nich, wybiera inną - na przykład, bramkę numer 3 - i otwiera ją. Za bramką, którą wskazuje gospodarz nigdy nie ma nagrody (za to jest tam koza). Następnie gospodarz proponuje uczestnikowi zmianę wyboru (w tym przypadku na bramkę numer 2).

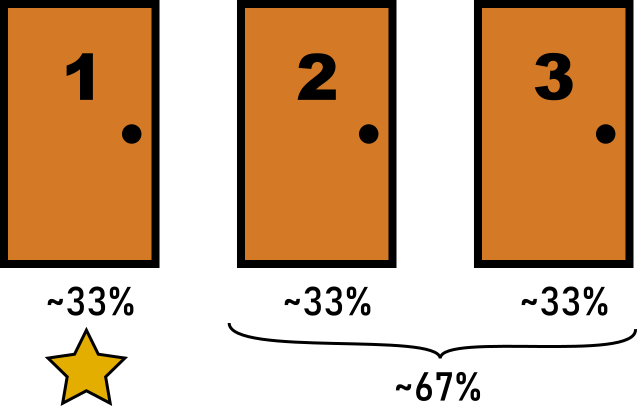
Problem jest następujący:

Czy zmiana bramki jest korzystna dla uczestnika konkursu?

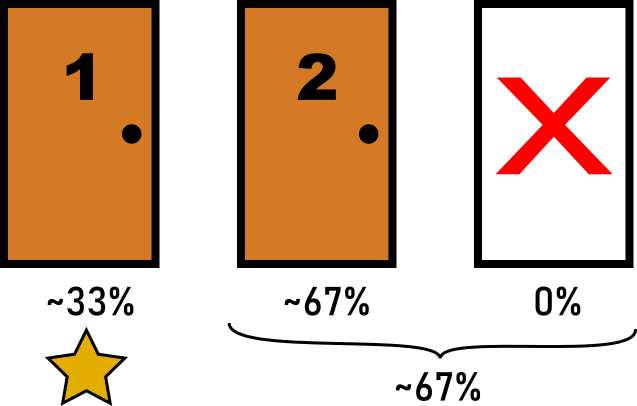
Czy zmiana wyboru będzie mieć znaczenie?

Odpowiedź wydaje się przeczyć intuicji, ale brzmi: tak, po zmianie wybranej bramki uczestnik zwiększa prawdopodobieństwo wygranej dwukrotnie! To był ciężki orzech do zgryzienia nawet dla matematyków, dopóki nie dowiedziono prawidłowości rozwiązania. Jest ono następujące:

Po pierwszym wyborze uczestnika do każdej bramki jest przyporządkowane prawdopodobieństwo wygranej (czyli znalezienia samochodu) równe 1/3 (33%). Innymi słowy, prawdopodobieństwo przegranej - czyli że samochód znajduje się za bramką, która nie została wybrana - wynosi 2/3 (67%).



Gdy gospodarz otwiera jedną z niewłaściwych bramek, prawdopodobieństwa nie ulegają zmianie: w bramce wybranej przez uczestnika prawdopodobieństwo wygranej ciągle wynosi 1/3, a prawdopodobieństwo przegranej 2/3. Co uległo zmianie? W tym momencie wiadomo, że jedna z pozostałych bramek nie zawiera nagrody, a zatem dla niej prawdopodobieństwo wygranej wynosi 0 - co oznacza, że teraz prawdopodobieństwo 2/3, poprzednio należące do dwóch bramek, których nie wybrał uczestnik, teraz jest przyporządkowane do jednej.



Podsumowując, prawdopodobieństwo błędnego wyboru ciągle wynosi 2/3, ale do wyboru pozostaje tylko jedna bramka, co oznacza, że w przypadku zmiany prawdopodobieństwo *właściwego* wyboru wyniesie właśnie 2/3. To nie gwarantuje wygranej, ale jednak dwukrotnie zwiększa jej prawdopodobieństwo!

## Opis problemu

Zostaliście zatrudnieni przez studio telewizyjne, które chce opracować nowy teleturniej oparty na paradoksie Monty/ego Halla. Skoro jednak uczestnicy mogą znać ten przypadek, niezbędne będą zmiany, by program był atrakcyjniejszy.

Gra rozpoczyna się z określoną liczbą bramek (większą od 3). Podobnie jak poprzednio, tylko za jedną bramką znajduje się nagroda. Na początku gry uczestnik konkursu wybiera jedną z bramek. Gospodarz otwiera co najmniej jedną z pozostałych bramek, ale wyłącznie spośród tych niezawierających nagrody. Następnie uczestnik dostaje szansę zmiany wybranej bramki. Proces jest kontynuowany aż do ostatniej rundy, w której uczestnik po raz ostatni może zmienić bramkę. Następnie otwierana jest bramka z nagrodą.

Studio chce przeprowadzić symulację najgorszego możliwego scenariusza, zgodnie z ich przewidywaniami: bardzo inteligentny uczestnik i bardzo pomocny gospodarz. Konkretnie, chcą przeprowadzić symulację, w której uczestnik i gospodarz zachowują się zgodnie z poniższymi regułami:

* W momencie otrzymania możliwości wyboru uczestnik wybiera bramkę z największym prawdopodobieństwem wygranej. W przypadku równych prawdopodobieństw uczestnik wybiera spośród bramek o tym samym prawdopodobieństwie tę z najniższym numerem.
* Podczas otwierania bramek gospodarz otwiera bramki, które po poprzedniej rundzie mają przyporządkowane największe prawdopodobieństwo wygranej. Gospodarz nigdy nie otwiera bramki z nagrodą i nigdy nie otwiera bramki wybranej (aktualnie czy poprzednio) przez uczestnika.

Dla przykładu rozważmy grę z dziesięcioma bramkami. Nagroda znajduje się za bramką numer 6 (oznaczoną na zielono), a w każdej z dwóch rund otwierane są trzy bramki. Uczestnik wybiera bramkę z najniższym numerem, czyli 1 (oznaczoną na żółto).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |

Następnie gospodarz otwiera trzy bramki. Ponieważ każda bramka ma identyczne prawdopodobieństwo wygranej, otwiera trzy ostatnie bramki: 8, 9 i 10. To zwiększa prawdopodobieństwo wygranej w bramkach, których nie wskazał uczestnik:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10% | 15% | 15% | 15% | 15% | 15% | 15% | X | X | X |

Następnie uczestnik otrzymuje możliwość zmiany wybranej bramki. Wszystkie niewybrane i nieotwarte bramki mają prawdopodobieństwo wygranej równe 15%, zatem uczestnik wybiera pierwszą z nich. Gospodarz otwiera trzy kolejne bramki z dużym prawdopodobieństwem wygranej. Powinien otworzyć 5, 6 i 7, ale za bramką numer 6 znajduje się wygrana. Zatem pomija bramkę numer 6 i otwiera bramki o numerach 4, 5 i 7:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10% | 15% | 37,5% | X | X | 37,5% | X | X | X | X |

Teraz do wyboru pozostają tylko cztery bramki, a uczestnik ponownie może zmienić bramkę. Dwie bramki mają prawdopodobieństwo wygranej równe 37,5%, zatem uczestnik wybiera pierwszą z nich. Niestety, wybrał źle, ale i tak ponad trzykrotnie zwiększył prawdopodobieństwo swojej wygranej wyłącznie dzięki zmianom wyboru.

Waszym zadaniem będzie napisać program, który zasymuluje kilka odmian takiej gry. W każdej symulacji uczestnik i gospodarz mają postępować zgodnie z powyższymi regułami. Wynik powinien określać, jaka jest na koniec gry szansa uczestnika na wygraną.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz składający się z trzech dodatnich liczb całkowitych oddzielonych spacjami. Te liczby kolejno reprezentują:

* Liczbę bramek na początku gry
* Liczbę rund, w trakcie których gospodarz otwiera bramki
* Liczbę bramek otwieranych w ramach każdej rundy

3

10 2 3

10 3 2

10 4 1

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić pojedynczy wiersz zawierający prawdopodobieństwo wygrania przez uczestnika nagrody po zakończeniu gry, przy założeniu, że uczestnik i gospodarz postępują według wyżej opisanych reguł. Prawdopodobieństwa mają być podane w formie procentu z dwoma miejscami dziesiętnymi (z ewentualnymi zerami następującymi).

37.50%

57.86%

24.61%

# Problem 13: Saper

**Punkty:** 25

**Autor:** Lourdes Tuma, Denver, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

|  |  |  |  |
| --- | --- | --- | --- |
| **☼** | **2** | **1** | **1** |
| **1** | **3** | **☼** | **2** |
| **0** | **2** | **☼** | **3** |
| **0** | **1** | **2** | **☼** |

Saper to gra logiczna dla pojedynczego gracza, w której gracz kolejno wybiera różne pola prostokątnej siatki. Pole siatki może być zajęte przez minę lub puste. Jeśli gracz wybierze pole zajęte przez minę, bomba „wybucha” i gra się kończy. W przeciwnym wypadku w odsłoniętym polu pojawia się liczba sąsiednich pól, które zawierają bomby. Pole uznaje się za sąsiadujące, jeśli stykają się z odsłoniętym polem przez jeden z boków lub przez narożnik.

## Opis problemu

Musicie napisać program, który po otrzymaniu rozmiaru siatki do sapera i lokalizacji min na tej siatce jest w stanie wyświetlić całą siatkę. Wynik powinien zawierać lokalizacje wszystkich min oraz liczby w „bezpiecznych” polach oznaczające liczbę sąsiadujących min.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać:

* Wiersz składający się z trzech dodatnich liczb całkowitych oddzielonych spacjami, reprezentujących:
  + Liczbę rzędów (wierszy) w siatce do sapera, **R**
  + Liczbę kolumn w siatce do sapera, **C**
  + Liczbę min rozmieszczonych na siatce do sapera, **B**
* **B** wierszy określających lokalizację każdej miny na siatce. Każdy wiersz zawiera dwie liczby całkowite oddzielone spacjami, reprezentujące:
  + Numer rzędu (wiersza) pola z miną. Rząd na samej górze ma numer 0. Zakres wartości sięga od 0 (włącznie) do **R** (rozłącznie).
  + Numer kolumny pola z miną. Pierwsza kolumna po lewej ma numer 0. Zakres wartości sięga od 0 (włącznie) do **C** (rozłącznie).

2

2 2 2

0 0

1 1

5 3 4

1 2

2 2

4 0

4 1

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić siatkę do gry w sapera opisaną danymi wejściowymi. Każdy wiersz powinien oznaczać kolejny rząd siatki, a każdy znak odrębne pole siatki. Pola z minami powinny być przedstawione jako gwiazdka (\*); z kolei pola „bezpieczne” powinny być przedstawione jako liczba (z zakresu od 0 do 8 włącznie) równa liczbie min w polach sąsiednich.

\*2

2\*

011

02\*

02\*

232

\*\*1

# Problem 14: Powrót do domu

**Punkty:** 30

**Autor:** Steve Brailsford, Marietta, Georgia, Stany Zjednoczone

## Wprowadzenie do problemu

Po długich rozmyślaniach nad doborem trasy akwizytor zakończył swoją podróż i odebrał po drodze zamówienia od swoich klientów. Właśnie dotarł do magazynu firmy, a teraz chce dostarczyć klientom zamówienia kierując się do domu. Aby nie pominąć nikogo, postanowił ruszyć tą samą drogą, co uprzednio, tylko w przeciwnym kierunku. Niestety po drodze do okienka kasowego na lotnisku potknął się i rozsypał swoje karty pokładowe! Teraz potrzebuje pomocy przy ułożeniu kart we właściwej kolejności, aby mógł odtworzyć swoją podróż i zorientować się, jak ma dotrzeć do domu.

## Opis problemu

Do waszego programu otrzymacie kilka par miast, wraz z czasem przylotu i odlotu podanymi na karcie pokładowej akwizytora. Te pary są jednak w dowolnej kolejności. Musicie odtworzyć jego plan podróży ustalając właściwą kolejność kart pokładowych, a następnie podać trasę, z jakiej powinien skorzystać wracając do domu (pierwotna trasa podróży, ale w odwrotnej kolejności).

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Dodatnia liczba całkowita, **X**, reprezentująca liczbę kart pokładowych
* **X** wierszy, a w każdym dwie nazwy miast, oddzielone spacją. Pierwsza nazwa to miasto, z którego akwizytor odlatuje, a druga to miasto, do którego przylatuje z użyciem danej karty pokładowej. Nazwy miast będą się składać z małych i wielkich liter oraz podkreśleń (\_).

2

4

Fort\_Worth Denver

Washington Toronto

Orlando Fort\_Worth

Denver Washington

5

Riyadh Singapore

Madrid London

Chicago Madrid

Berlin Riyadh

London Berlin

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić każde miasto, do jakiego musi trafić akwizytor podczas swojej podróży do domu - w każdym wierszu jedno miasto - poczynając od miasta końcowego w pierwszej podróży.

Toronto

Washington

Denver

Fort\_Worth

Orlando

Singapore

Riyadh

Berlin

London

Madrid

Chicago

# Problem 15: LMCoin

**Punkty:** 35

**Autor:** Ben Fenton, Faslane, Helensburgh, Wielka Brytania

## Wprowadzenie do problemu

W dzisiejszych czasach obserwujemy szybki wzrost popularności bitcoina i innych kryptowalut. Różnią się one znacząco od tradycyjnych środków płatności, bo nie wymagają udziału banku ani instytucji wydającej karty kredytowe jako pośrednika między nabywcą i sprzedawcą. Dzięki temu strony transakcji nie muszą przekazywać prowizji do banku czy innej organizacji.

Zamiast tego bitcoin trafia bezpośrednio do drugiej strony transakcji. Jednak wiąże się to z istotnym problemem - jak dostarczyć dowód zapłaty za towar? Albo jak w ogóle dowieść faktu posiadania pieniędzy, jeśli nie liczyć możliwości uzyskania poświadczenia innej osoby? Jest to znane pod nazwą „problemu podwójnego wydawania”.

Zamiast banku, który rejestruje wszystkie transakcje w głównej księdze to użytkownicy bitcoina rejestrują wszystkie transakcje jednocześnie. Oznacza to, że próba oszukania systemu zostanie zauważona, a transakcja nie zostanie przeprowadzona. Służy do tego zabezpieczenie zwane łańcuchem bloków (ang. blockchain). W omawianym problemie zbudujemy prosty łańcuch bloków dla LMCoin, naszej własnej waluty cyfrowej.

Jak sugeruje nazwa, łańcuch bloków zawiera kilka „bloków” danych, z których każdy reprezentuje oddzielną transakcję. Każdy blok jest identyfikowany za pomocą niepowtarzalnego hash’a (funkcja skrótu), wartości generowanej z wykorzystaniem wszystkich informacji przechowywanych w bloku; zalicza się do nich hash poprzedniego bloku w łańcuchu. Dlatego też w miarę przesuwania się wzdłuż łańcucha jego spójność rośnie i gwarantuje, że poprzednie transakcje nie ulegną zmianie. Bitcoin i inne kryptowaluty są tworzone w procesie zwanym „kopaniem”; dodawaniem nowego bloku do łańcucha, który tworzy niepowtarzalny „hash” w danym zakresie - wiąże się to ze sporym wysiłkiem włożonym w zgadywanie!

## Opis problemu

Waszym zadaniem jest napisanie programu implementującego łańcuch bloków LMCoin. Jak sugeruje nazwa, łańcuch bloków zawiera kilka „bloków” danych, z których każdy reprezentuje oddzielną transakcję. Oprócz bloku startowego (który nie zawiera elementu poprzedniego bloku) każdy z bloków zawiera cztery informacje:

1. Sygnaturę czasową, przedstawiającą czas wygenerowania bloku
2. Pewne dane (np. zamówienie na pizzę)
3. Indeks (położenie danego bloku w łańcuchu)
4. Hash poprzedniego bloku w łańcuchu

Algorytm hash’a wykorzystywany w naszej walucie działa następująco:

Gdzie:

* to indeks bloku w łańcuchu; pierwszy blok w łańcuchu ma indeks 1.
* to hash bloku o indeksie . ()
* to sygnatura czasowa bloku o indeksie .
* to wartość numeryczna danych zawartych w bloku o indeksie (wyjaśnienie poniżej).

Dane bloku są zamieniane na wartość numeryczną przez zsumowanie wartości każdej litery w łańcuchu danych, zgodnie z poniższą regułą:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Litera | a | b | c | d | e | f | g | h | i | j | k | l | m |
| Wartość | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Litera | n | o | p | q | r | s | t | u | v | w | x | y | z |
| Wartość | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |

Na przykład, wartość łańcucha „cheese” to 3 + 8 + 5 + 5 + 19 + 5 = 45.

Sygnatury czasowe są podawane w formacie DDMMRRHHMM (dwie cyfry dla, kolejno, dnia, miesiąca, roku, godziny i minuty); na przykład, 27 kwietnia 2019 r., 8:30 zostanie zapisana jako 2704190830.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz zawierający listę wartości danych dla pierwszych dziesięciu bloków w łańcuchu, oddzielonych spacjami. Wartości danych będą zawierać jedynie małe litery.
* Wiersz zawierający listę sygnatur czasowych dla pierwszych dziesięciu bloków w łańcuchu, oddzielonych spacjami. Sygnatury czasowe będą w formacie podanym powyżej.

2

pepperoni veggie ham peppers cheese olives mushroom chicken beef bacon

2704191000 2704191030 2704191100 2704191130 2704191200 2704191230 2704191300 2704191330 2704191400 2704191430

candy candy salad chips pretzel icecream apple fries cookie sandwich

2602201200 2602201300 2702201200 2702201300 2802201200 2802201300 2902201200 2902201300 0103201200 0103201300

*(Należy pamiętać, że po liczbie przypadków testowych znajdują się tylko cztery wiersze danych wejściowych; po prostu wiersz z sygnaturami czasowymi jest za długi i nie mieści się na stronie.)*

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić hash dziesiątego bloku w łańcuchu, obliczony z wykorzystaniem podanych wartości. Wyświetlone hash’e mają być zaokrąglone do najbliższej pełnej liczby. Nie wolno zaokrąglać pośrednich wartości hash’y używanych do obliczeń.

1393884230

219309065

# Problem 16: Gdzie są moje klucze?

**Punkty:** 35

**Autor:** Marcus Garza, Fort Worth, Teksas, Stany Zjednoczone

## Wprowadzenie do problemu

W kryptografii istnieje koncepcja szyfrowania, którą w pewnych warunkach uznaje się za idealną (nie do złamania); jest to tzw. szyfr z kluczem jednorazowym (OTP). Szyfr ten zakłada, że każdy klucz jest tak długi jak sama wiadomość i jest używany tylko raz.

Do słabości szyfru z kluczem jednorazowym należą prawdopodobieństwo, że osoba próbująca złamać szyfr zna zaszyfrowaną informację („leksykon”) oraz rozmiar klucza. Ale nawet jeśli osoba próbująca złamać szyfr zna leksykon, to i tak atak brute force - sprawdzający każdą możliwą kombinację - jest praktycznie niewykonalny.

Załóżmy, że próbujemy zaszyfrować wiadomość z kluczem o długości 2512 bitów. 2512 to OGROMNA liczba:

13 407 807 929 942 597 099 574 024 998 205 846 127 479 365 820 592 393 377 723

561 443 721 764 030 073 546 976 801 874 298 166 903 427 690 031 858 186 486 050

853 753 882 811 946 569 946 433 649 006 084 096

Jeśli osoba łamiąca szyfr chce wypróbować na tej wiadomości atak brute force i byłaby w stanie testować jeden klucz co nanosekundę (0,000000001 sekundy), to sprawdzenie wszystkich kluczy zajęłoby jej 100 trylionów trylionów trylionów trylionów trylionów trylionów trylionów trylionów trylionów (to 9 „trylionów”) lat. To wielkość o kilka rzędów większa niż wiek wszechświata i choć ostatecznie ta metoda pozwoliłaby uzyskać poprawną wiadomość, to również jej efektem byłoby uzyskanie wszystkich innych potencjalnie poprawnych wiadomości, co uniemożliwiłoby ustalenie, która odpowiedź jest *właściwa*.

## Opis problemu

Wasz program ma zaimplementować szyfr z kluczem jednorazowym. Otrzymacie do niego 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący szyfrowaną wiadomość. Dodatkowo otrzymacie drugi 128-znakowy łańcuch w układzie szesnastkowym, który będzie stanowił klucz do szyfru. Wiadomość niezaszyfrowana składa się z 64 znaków ASCII. Aby zamienić wiadomość zaszyfrowaną na niezaszyfrowaną, należy użyć procesu przedstawionego w poniższej tabeli.

1. Weź dwa kolejne znaki szesnastkowe z wiadomości zaszyfrowanej (np. „4F”)
2. Zamień wartość szesnastkową na jej 8-bitowy odpowiednik w układzie dwójkowym (4F = 01001111)
3. Weź dwa kolejne znaki szesnastkowe z klucza (np. „0C”)
4. Zamień wartość szesnastkową na jej 8-bitowy odpowiednik w układzie dwójkowym (0C = 00001100)
5. Użyj funkcji XOR (albo; alternatywa rozłączna) dla tych dwóch wartości, aby utworzyć nową liczbę dwójkową
6. Zamień tę nową liczbę dwójkową na wartość dziesiętną ASCII i wyświetl ten znak
7. Powtórzy proces dla reszty wiadomości

Szesnastkowe 4F = Dziesiętne 79 = Dwójkowe 01001111

Szesnastkowe 0C = Dziesiętne 12 = Dwójkowe 00001100

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4F | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0C | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| XOR | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

Dwójkowe 01000011 = Dziesiętne 67 = „C” w układzie ASCII

Jeśli nie znacie funkcji XOR, jest to funkcja boolowska (logiczna) sprawdzający, czy dwie różne wartości logiczne są różne. Jeśli tak, wynikiem jest „true” (1). Jeśli są takie same, wynikiem jest „false” (0).

|  |  |  |
| --- | --- | --- |
| XOR | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Większość języków programowania pozwala użyć funkcji XOR do dwóch liczb, aby utworzyć nową liczbę; te operacje „bitowe” służą do wykonania porównania XOR na każdym bicie dwójkowej reprezentacji tych liczb, podobnie jak wykonaliśmy w powyższym przykładzie: 79 XOR 12 = 67. W językach Java, C, C++ i Python służy do tego kareta (^), a w języku VB.NET operator Xor.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze:

* Wiersz zawierający dodatnią liczbę całkowitą, **X**, reprezentującą liczbę używanych kluczy
* Wiersz zawierający 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący zaszyfrowaną wiadomość
* **X** wierszy, z których każdy zawiera 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący klucz

1

2

4F6F0E14089E040E286156061893404F658D1F6510D5744098DB1DF8904D5F0DF23710D30230F4F985D4FAAE50F4984AF40B4C98F70E98F94998F043DF16D89F

0C006A7128CF716B5B15766F6BB3263A0BAC3F227FBA1060F4AE7E93B0393069934E31F3515F988FE0F48EC63F87FD6A847923FA9B6BF58A68B8D063FF36F8BF

1B07676728EE686F410F226360E7602704FE3F1178B05433F9B678D8F3242F65974564B67A44D49BF0A0DACF7090F12C926E3EFD997AB8922CE1DE639E5799DE

*(Należy pamiętać, że powyżej znajdują się tylko trzy wiersze tekstu w układzie szesnastkowym; są zawinięte, bo nie mieszczą się na stronie)*

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić 64-znakową odszyfrowaną wiadomość uzyskaną przez odszyfrowanie zaszyfrowanej wiadomości za pomocą przekazanego klucza - jedną dla każdego wiersza. Wiadomość odszyfrowana musi być umieszczona w nawiasach kwadratowych. Może zawierać na końcu łańcucha dodatkowe spacje, ale muszą mieścić się wewnątrz nawiasów.

[Code Quest is fun! Good luck today! Solve those problems! ]

[This plaintext has the same ciphertext but a different key. AAAA]

# Problem 17: Gra w życie Conwaya

**Punkty:** 40

**Autor:** Louis Ronat, Denver, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

W 1940 r. John von Neumann, naukowiec zajmujący się maszynami liczącymi, zdefiniował życie jako twór, który może się powielać i symulować maszynę Turinga: w skrócie, urządzenie działające według zestawu reguł. Ta definicja dała początek serii matematycznych doświadczeń. Do najsłynniejszych należy „gra” stworzona w 1970 r. przez matematyka, Johna Conwaya, nazwana *Życie*. *Życie* Conwaya opiera się na czterech regułach, według których ma postępować komputer, wychodząc od zadanego stanu początkowego siatki wypełnionej komórkami „żywymi” i „martwymi”.

W każdym pokoleniu:

1. Każda żywa komórka sąsiadująca z jedną żywą komórką lub niesąsiadująca z żadną żywą komórką umiera (z samotności).
2. Każda żywa komórka sąsiadująca z dwoma lub trzema żywymi komórkami żyje dalej.
3. Każda żywa komórka sąsiadująca z co najmniej czterema żywymi komórkami umiera (od przeludnienia).
4. Każda martwa komórka sąsiadująca z dokładnie trzema żywymi komórkami ożywa (przez powielanie).

Komórki leżące po przekątnej uznaje się za sąsiadujące. *Życie* rozwija się poprzez stosowanie powyższych reguł do „świata” reprezentowanego przez siatkę. Reguły zostają zastosowane, „świat” jest odpowiednio modyfikowany, i tak bez końca.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T = 0** | | | | |  | **1** | | | | |  | **2** | | | | |  | **3** | | | | |  | **4** | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | |  | | | | |  | | | | |  | | | | |  | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | |  | | | | |  | | | | |  | | | | |  | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | |  | | | | |  | | | | |  | | | | |  | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Te pozornie proste reguły są całkowicie deterministyczne; to jest, każde pokolenie jest determinowane wyłącznie przez stan poprzedniego pokolenia. Mimo to można zaobserwować bardzo złożone zachowania. Teoretycznie, *Życie* jest „uniwersalną maszyną Turinga”; to znaczy, że wszystko co da się obliczyć algorytmem da się obliczyć w ramach *Życia*.

## Opis problemu

Musicie zaprojektować program, który zaimplementuje *Życie* Conwaya na siatce o wymiarach 10 na 10. Do waszego programu otrzymacie dane stanu początkowego pierwszego pokolenia. Następnie program musi ustalić stan świata po zadanej liczbie pokoleń. Należy pamiętać, że komórki poza granicami siatki 10 na 10 są zawsze uznawane za martwe.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz zawierający dodatnią liczbę całkowitą, **X**, reprezentująca liczbę pokoleń, jakie należy obliczyć
* Dziesięć wierszy zawierających po dziesięć znaków, a każdy odzwierciedla początkowy stan świata. Używane znaki to „1”, który reprezentuje żywą komórkę i „0”, który reprezentuje martwą komórkę.

1

6

0000000000

0000000000

0000000000

0000010000

0000111000

0000111000

0000010000

0000000000

0000000000

0000000000

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić stan świata po podanej liczbie pokoleń. Każdy przypadek testowy powinien zawierać dziesięć wierszy po dziesięć znaków każdy.

0000000000

0000000000

0000111000

0001000100

0000000000

0000000000

0001000100

0000111000

0000000000

0000000000

# Problem 18: Zbiór Mandelbrota

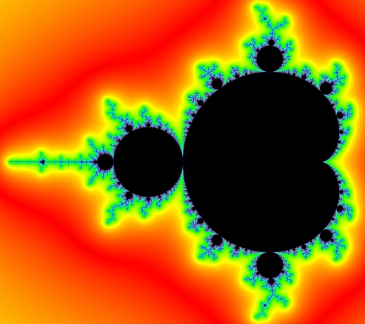
**Punkty:** 45

**Autor:** Louis Ronat, Denver, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

Zbiór Mandelbrota jest rysowany poprzez założenie funkcji rekurencyjnej , gdzie to liczba zespolona formy (w matematyce to liczba urojona o wartości ; stąd ). Poprzez wielokrotną iterację, używając każdej wartości do obliczenia kolejnej wartości, dowiadujemy się, że w przypadku niektórych wartości początkowych rośnie bez końca. W innych przypadkach pozostaje ograniczona.

Aby narysować zbiór Mandelbrota, korzystamy z „płaszczyzny zespolonej”, gdzie pozioma oś X reprezentuje wartość , a pionowa oś Y reprezentuje wartość . Każdy punkt ma swoją barwę zależną od liczby iteracji (), jakie możemy wykonać, zanim wartość bezwzględna () przekroczy określoną wartość. W takim momencie mówimy, że funkcja staje się rozbieżna (ulega dywergencji). Na poniższym obrazie barwa czarna oznacza, że pozostała poniżej podanej wartości dla wszystkich wartości . Niebieskie piksele reprezentują punkty, w których po wielu iteracjach przekroczyła tę wartość; w przypadku czerwonych pikseli liczba iteracji była mniejsza.



Rozważmy funkcję korzystającą z wartości .

Niezależnie od wartości , wartość pozostaje zawsze równa 0. Możemy z tego skorzystać w celu określenia wartości :

To pokazuje, że dla każdej wartości jest . Teraz musimy ustalić, czy funkcja stała się rozbieżna. Na potrzeby tego problemu uznajmy, że funkcja stała się rozbieżna, jeśli . Ponieważ to liczba urojona, użyjemy tego równania do ustalenia wartości bezwzględnej liczb formy :

2,2825 to mniej niż 100, zatem funkcja nie stała się jeszcze rozbieżna. Potrzebujemy więcej iteracji, by ustalić, czy w ogóle i kiedy funkcja stanie się rozbieżna:

(Należy pamiętać, że , zatem powyżej .)

to ciągle mniej niż 100, zatem funkcja nie stała się jeszcze rozbieżna. Ilu iteracji potrzebujemy na dotarcie do tego miejsca?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***n*** | ***Z*** | ***a*** | ***b*** | **|*Z*|** |
| 1 | 1.1 + 2*i* | 1.1 | 2 | 2.2825 |
| 2 | -1.69 + 6.4*i* | -1.69 | 6.4 | 6.6194 |
| 3 | -37.0039 - 19.632*i* | -37.0039 | -19.632 | 41.8892 |
| 4 | 984.9732 + 1454.9211*i* | 984.9732 | 1454.9211 | 1756.9769 |

Zatem przy widzimy, że wartość . Oznacza to, że dla tej wartości funkcja stała się rozbieżna w 4 kroku iteracji. Barwimy punkt o współrzędnych x = 1,1 i y = 2 kolorem odpowiednim dla tej wartości i przechodzimy do następnej sprawdzanej wartości.

## Opis problemu

Wasz program musi zidentyfikować barwę używaną do renderowania zbioru Mandelbrota dla zadanej wartości . Z pomocą poniższej tabeli i wyjaśnień podanych powyżej należy ustalić, jakiej barwy użyć:

|  |  |
| --- | --- |
| Wartość , gdy funkcja staje się rozbieżna | Barwa |
| ≤ 10 | RED (CZERWONA) |
| 11-20 | ORANGE (POMARAŃCZOWA) |
| 21-30 | YELLOW (ŻÓŁTA) |
| 31-40 | GREEN (ZIELONA) |
| 41-50 | BLUE (NIEBIESKA) |
| ≥ 51 | BLACK (CZARNA) |

W przykładowych obliczeniach funkcja staje się rozbieżna w , zatem prawidłowa barwa dla tej wartości to czerwona.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać pojedynczy wiersz składający się z dwóch liczb dziesiętnych oddzielonych spacjami. Te liczby to odpowiednio wartości i . Należy pamiętać, że .

4

1.1 2.0

-0.7 0.2

-0.5 0.65

-0.5 0.608

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić wartość , po której następuje spacja, a następnie barwa użyta do renderowania tej wartości, zgodnie z powyższą tabelą. Barwę należy wyświetlić wielkimi literami. Wartości dziesiętne powinny być wyświetlone tak, jak zostały podane w danych wejściowych.

1.1+2.0i RED

-0.7+0.2i BLACK

-0.5+0.65i ORANGE

-0.5+0.608i BLUE

# Problem 19: Strażnik sieci

**Punkty:** 50

**Autor:** Brett Reynolds, Orlando, Floryda, Stany Zjednoczone

## Wprowadzenie do problemu

Dlaczego Internet jest jak poczta? W jednym i drugim korzysta się z adresów!

Komputery i inne urządzenia łączące się z Internetem mają przyporządkowywany adres IP (Internet Protocol). Choć dostępny jest nowszy format, większość systemów ciągle stosuje w tych adresach format IPv4. W tym formacie adres IP składa się z czterech liczb oddzielonych kropkami. Zakres tych liczb to od 0 do 255. Na przykład, adres 127.0.0.1 zawsze oznacza aktualnie używany komputer („localhost”).

Podobnie jak z innymi danymi, komputer przechowuje te adresy w formacie dwójkowym. Każda liczba adresu jest reprezentowana przez 8-bitowy łańcuch dwójkowy składający się z zer i jedynek; te łańcuchy są sklejane z innymi w celu utworzenia pełnego adresu. Na przykład, adres IP 166.23.250.209 jest zamieniany na:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 166 | | | | | | | | 23 | | | | | | | | 250 | | | | | | | | 209 | | | | | | | |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

Podobnie jak adresy zamieszkania można grupować za pomocą kodów pocztowych, adresy IP można grupować blokami. Usługodawcy internetowi mogą rezerwować te bloki chcąc wykorzystywać je do przypisywania adresów IP klientom, za pomocą systemu o nazwie Classless Inter-Domain Routing (CIDR). Blok CIDR definiuje się zapisując adres IP, po którym następuje ukośnik (ang. slash) i liczba identycznych bitów wspólnych dla całego bloku (licząc od lewej strony każdego adresu). Na przykład, adresy IP 192.168.0.0 i 192.168.108.68 są reprezentowane przez następujące liczby dwójkowe:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 192 | | | | | | | | 168 | | | | | | | | 0 | | | | | | | | 0 | | | | | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 192 | | | | | | | | 168 | | | | | | | | 108 | | | | | | | | 65 | | | | | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Pierwsze 17 bitów obydwu adresów jest identycznych, zatem adresy są częścią bloku 192.168.0.0/17 (po pierwszej niepasującej parze bitów wszystkie inne pasujące bity są już pomijane). Można to również zapisać jako blok 192.168.108.65/17, ale zwyczajowo, podczas takiego zapisu, używa się pierwszego (najmniejszego) adresu w bloku.

Bloki mogą mieć rozmiar sięgający od /0 do /32. Blok /32 wymaga, aby wszystkie 32 bity były identyczne, zatem reprezentuje on pojedynczy adres. Blok /0 w ogóle nie zawiera kryterium identyczności bitów, zatem reprezentuje on cały Internet!

W omawianym problemie współpracujecie z wydziałem cyberprzestępstw Centralnego Biura Śledczego z zamiarem wyśledzenia szajki oszustów internetowych korzystających z oprogramowania typu ransomware do ataków na niewinnych ludzi. Udało wam się wyśledzić listę adresów IP wykorzystywanych przez oszustów. CBŚ chce uzyskać nakaz przeszukania, aby dowiedzieć się, kto korzysta z tych adresów IP, ale sędzia nie wyda nakazu, dopóki nie uda wam się zidentyfikować jak najmniejszego zbioru obejmującego wszystkie te adresy.

## Opis problemu

Do waszego programu otrzymacie listę adresów IPv4, a waszym zadaniem będzie identyfikacja najmniejszego bloku CIDR, który zawiera każdy adres z tych na liście. Każdy blok CIDR powinien być zapisany z wykorzystaniem pierwszego (najmniejszego) adresu w bloku; zatem 192.168.0.0/16 może być dopuszczalną odpowiedzią, ale 192.168.0.1/16 już nie.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz zawierający dodatnią liczbę całkowitą, **X**, oznaczającą liczbę adresów IP użytych w tym przypadku testowym
* **X** wierszy, z których każdy zawiera adres IPv4

2

2

192.168.0.0

192.168.255.255

4

32.73.94.16

32.73.89.172

32.73.95.210

32.73.92.82

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić jak najmniejszy zakres CIDR, który zawiera każdy adres IP z podanej listy, zgodnie z podanym powyżej formatem.

192.168.0.0/16

32.73.88.0/21

# Problem 20: Obserwacja ptaków

**Punkty:** 55

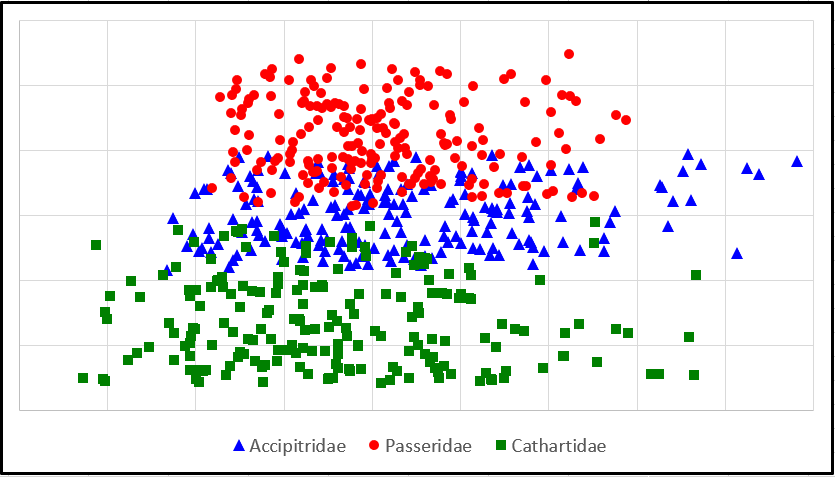
**Autor:** Joe Worsham, Colorado Springs, Kolorado, Stany Zjednoczone

## Wprowadzenie do problemu

Uczenie maszynowe (ML - Machine Learning) to wymyślny rodzaj algorytmu sztucznej inteligencji, który wykorzystuje wzorce w uprzednio zarejestrowanych danych do budowy przewidywań dotyczących danych, z jakimi się jeszcze nie spotkał. Algorytmy ML mogą być bardzo złożone i wykorzystywane do rozwiązywania bardzo trudnych problemów; na przykład, sieci neuronowe symulują zachowanie poszczególnych komórek w prawdziwym, żywym mózgu. Niektóre z nich są jednak proste, zwyczajne i bardzo skuteczne w rozwiązywaniu problemów dotyczących wzorców. W omawianym problemie macie stworzyć własny system uczenia maszynowego!

Współpracujecie ze służbami lokalnego parku, aby posortować wykonywane przez nich ostatnio zdjęcia ptaków. Chcą używać tych zdjęć do śledzenia populacji pewnych gatunków, ale zdjęć jest bardzo dużo i potrzebna będzie właściwa metoda organizacyjna. Postanowili skorzystać z metod stosowanych w taksonomii i poukładać zdjęcia według rodziny ptaka widocznego na fotografii. Przy takiej liczbie fotografii potrzebują automatycznej metody do poukładania zdjęć.

Wasza współpracowniczka proponuje, by użyć pomiarów ptaków uzyskanych ze zdjęć do przewidywania rodziny taksonomicznej danego osobnika. Znajduje zbiór informacji na temat długiej listy gatunków ptaków i pokazuje, że między pewnymi pomiarami a rodziną ptaka istnieje widoczna korelacja:



Powyższy wykres to algorytm T-distributed Stochastic Neighbor Embedding (t-SNE), który zamienia pewną liczbę pomiarów (w tym przypadku cztery) na dwuwymiarowe współrzędne. Każdy punkt jest oznaczony kolorem wskazującym na rodzinę taksonomiczną ptaka. Zgodnie ze spostrzeżeniami współpracowniczki większość kolorowych punktów jest pogrupowana w pobliżu siebie, co nadaje wagi jej hipotezie o możliwości znalezienia wzorca. Proponuje, by ustalić, jak „daleko” znajduje się nieznany ptak ze zdjęcia od tych pomiarów i użyć tej wartości do postawienia hipotezy dotyczącej rodziny tego ptaka.

## Opis problemu

Pomysł waszej współpracowniczki jest nazywany algorytmem k-Nearest Neighbor (kNN). Pozwala on przewidywać rodzinę nieznanego ptaka na podstawie dostępnych danych taksonomicznych i pomiarów badanego osobnika. Algorytm kNN działa obliczając odległość między nieznanym punktem danych i każdym znanym punktem danych. Następnie *k* znanych punktów danych, znajdujących się najbliżej nieznanej danej, używa się do „zagłosowania” w celu otrzymania ostatecznej decyzji.

W omawianym problemie wasz algorytm powinien obliczać odległość od nieznanego punktu danych do każdego znanego punktu danych. Po obliczeniu wszystkich tych odległości należy policzyć, ile razy każda rodzina ptaków pojawia się w **K** najbliższych punktów. To właśnie jest proces nazwany powyżej „głosowaniem”. Rodzina otrzymująca najwięcej głosów zostaje wybrana jako rodzina nieznanego osobnika.

Zasadniczo, wartość **K** w tym algorytmie musi być liczbą całkowitą; jeśli możliwe są tylko dwie odpowiedzi, jest to zwykle nieparzysta liczba całkowita, aby podczas głosowania nie padł remis. Tutaj mamy trzy możliwe odpowiedzi, zatem remisy będą dopuszczalne. Aby uwzględnić tę przesłankę, zacznijmy od początkowej wartości **K** = 5 dla wszystkich nieznanych ptaków. Jeśli padnie remis, należy zwiększyć **K** o jeden tyle razy, by przechylić szalę na korzyść jednego z wyników; następnie przy kolejnym nieznanym ptaku ponownie zacząć od **K** równego 5.

Wzór służący do obliczania odległości między *N*-wymiarowymi punktami jest następujący:

Każdy ptak będzie reprezentowany przez (nie licząc rodziny) cztery punkty danych, czyli jego długość, szerokość ciała, rozpiętość skrzydeł i kąt ustawienia skrzydeł w stosunku do ciała.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz składający się z dwóch dodatnich liczb całkowitych oddzielonych spacjami: **X**, reprezentującą liczbę znanych ptaków oraz **Y**, reprezentującą liczbę nieznanych ptaków.
* **X** wierszy zawierających informacje o znanych ptakach. Każdy wiersz będzie zawierać następujące wartości oddzielone spacjami:
  + Jedno ze słów „Accipitridae”, „Passeridae” lub „Cathartidae”, będące rodziną taksonomiczną ptaka
  + Liczbę dziesiętną reprezentującą długość ptaka w calach.
  + Liczbę dziesiętną reprezentującą szerokość ciała ptaka w calach.
  + Liczbę dziesiętną reprezentującą rozpiętość skrzydeł ptaka w calach.
  + Liczbę dziesiętną reprezentującą kąt skrzydeł ptaka w stopniach.
* **Y** wierszy zawierających informacje o nieznanych ptakach widocznych na fotografiach. Każdy wiersz będzie zawierać następujące wartości oddzielone spacjami:
  + Liczbę dziesiętną reprezentującą długość ptaka w calach.
  + Liczbę dziesiętną reprezentującą szerokość ciała ptaka w calach.
  + Liczbę dziesiętną reprezentującą rozpiętość skrzydeł ptaka w calach.
  + Liczbę dziesiętną reprezentującą kąt skrzydeł ptaka w stopniach.

1

15 3

Accipitridae 12.30 7.03 25.32 88.59

Accipitridae 21.38 7.57 22.18 88.71

Passeridae 16.57 7.05 25.88 89.27

Passeridae 13.34 6.24 21.37 88.95

Passeridae 15.75 6.58 22.16 89.35

Accipitridae 15.16 5.17 22.43 89.04

Cathartidae 18.61 6.68 23.37 88.83

Accipitridae 21.32 8.14 20.09 88.55

Cathartidae 18.35 7.01 20.64 88.14

Cathartidae 13.61 5.33 23.72 90.21

Cathartidae 16.88 6.63 24.59 88.48

Accipitridae 15.63 8.66 23.19 88.51

Passeridae 17.29 7.62 26.46 89.31

Passeridae 20.03 8.68 20.97 89.05

Cathartidae 19.19 7.74 22.31 88.09

19.37 15.35 17.30 15.28

12.76 21.96 14.41 16.84

20.33 15.51 16.29 17.10

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić przewidywaną rodzinę taksonomiczną każdego nieznanego ptaka, po jednej w każdym wierszu.

Accipitridae

Cathartidae

Accipitridae

# Problem 21: Ukrywaj swoich szpiegów

**Punkty:** 55

**Autor:** Wojciech Kozioł, Mielec, Polska

## Wprowadzenie do problemu

Współpracujecie z agencją wywiadowczą, a waszym zadaniem jest poprowadzenie szpiega przez sekretny ośrodek wroga. Wróg ma w całym budynku kamery z polem widzenia 360°; jeśli szpieg zostanie dostrzeżony, misja się nie uda! Na szczęście, na miejscu jest kilka ścian zasłaniających pole widzenia kamer, za którymi szpieg może znaleźć schronienie. Waszym zadaniem jest ustalenie, czy szpieg zostanie zaobserwowany przez kamerę, przy zadanym położeniu kamer, szpiega i ścian w pomieszczeniu.

## Opis problemu

Jeśli przyjmiecie to zadanie, to będziecie musieli ustalić, czy między kamerą umieszczoną w miejscu o zadanych współrzędnych i szpiegiem umieszczonym w miejscu o innych zadanych współrzędnych występuje niezaburzona widoczność (tj. czy szpieg będzie widoczny dla kamery). W pomieszczeniu będzie umieszczonych kilka ścian; jeśli ściana przecina linię biegnącą od kamery do szpiega, to szpieg jest schowany i nie zostanie wykryty. Musicie napisać program sprawdzający, czy szpieg jest schowany i zgłaszający, czy został wykryty czy też nie.

|  |  |
| --- | --- |
|  |  |
| *Linia ściany nie przecina linii biegnącej od kamery do szpiega. Szpieg został wykryty!* | *Między szpiegiem a kamerą mieści się ściana, przecinająca linię obserwacji. Szpieg pozostaje w ukryciu.* |

Aby ustalić, czy linie się przecinają, należy zlokalizować punkt, w którym linie przecięłyby się, gdyby biegły nieskończenie w obydwu kierunkach. Należy pamiętać, że (niepionową) linię można zdefiniować za pomocą równania

określa się mianem „pochylenia” linii i można je obliczyć z wykorzystaniem dowolnych dwóch punktów i , zgodnie z poniższymi obliczeniami:

(Jeśli , to jest nieokreślone, a linia jest pionowa.) Po znalezieniu można obliczyć z jej wykorzystaniem , dodatkowo używając współrzędnych jednego z punktów na linii:

Aby uzyskać obydwa równania linii, należy wykonać ten proces dla obydwu linii, które są badane pod kątem potencjalnego przecięcia. Następnie można użyć obydwu równań do obliczenia punktu , w którym linie się przecinają. Jeśli ten punkt mieści się w granicach zbioru znanych punktów, to ściana blokuje pole obserwacji kamery!

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz składający się z pięciu liczb całkowitych oddzielonych spacjami, reprezentujących kolejno poniższe informacje:
  + Współrzędną X szpiega w badanym pomieszczeniu
  + Współrzędną Y szpiega w badanym pomieszczeniu
  + Współrzędną X kamery w badanym pomieszczeniu
  + Współrzędną Y kamery w badanym pomieszczeniu
  + Liczbę ścian w badanym pomieszczeniu, **W**
* **W** wierszy zawierających cztery liczby całkowite oddzielone spacjami, przy czym każdy wiersz zawiera informacje o ścianie obecnej w pomieszczeniu:
  + Współrzędną X początku ściany
  + Współrzędną Y początku ściany
  + Współrzędną X końca ściany
  + Współrzędną Y końca ściany

2

2 2 6 4 1

2 5 5 5

2 2 6 4 2

4 1 4 5

1 5 4 5

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program musi wyświetlić pojedynczy wiersz zawierający słowo „YES” (co oznacza, że szpieg został wykryty przez kamerę) lub „NO” (co oznacza, że szpieg uniknął wykrycia).

YES

NO

# Problem 22: Czyszczenie dysku

**Punkty:** 60

**Autor:** Doug Kelley, Palmdale, Kalifornia, Stany Zjednoczone

## Wprowadzenie do problemu

Miejsca na dysku komputera jest coraz mniej. Potrzebujemy algorytmu do archiwizowania najstarszych plików (których najpewniej już nie będziemy używać) i największych plików (które zajmują najwięcej miejsca).

Zakładamy, że kilobajt (KB) to 1000 bajtów (B), a megabajt (MB) to 1000 kilobajtów (kB). Gigabajt to 1000 megabajtów (MB).

## Opis problemu

Do waszego programu otrzymacie listę plików na komputerze i pewne informacje ich dotyczące. Do każdego pliku należy przyporządkować punktację zależną od jego wieku i rozmiaru. Wykorzystując dzisiejszą datę, 27 kwietnia 2019 r., ustalcie wiek pliku w dniach. Plik utworzony nad ranem (od północy (12:00 AM) do 11:59 (11:59 AM)) należy uznać za ½ dnia starszy od pliku utworzonego tego samego dnia, ale popołudniu (od 12:00 (12:00 PM) do 23:59 (11:59 PM)); stąd plik zmodyfikowany ostatnio wczoraj o godzinie 13:00 (1:00 PM) ma 0,5 dnia; z kolei plik zmieniany wczoraj rano ma 1 dzień. Pomnóżcie wiek pliku w dniach przez jego rozmiar w MB, by uzyskać punktację pliku. Należy pamiętać o uwzględnieniu w obliczeniach lat przestępnych.

Na przykład, plik o rozmiarze 1500 KB został ostatnio zmieniony 27 kwietnia 2018 r. o godzinie 22:00 (10:00 PM). Plik ma 364,5 dnia (365 dni minus 0,5 dnia ze względu na popołudniową godzinę modyfikacji). Pomnożenie tej wartości przez rozmiar pliku – 1500 KB, czyli 1,5 MB - daje punktację 546,75.

Wasz program musi wypisać pliki o najwyższej punktacji wraz z punktacją, których sumaryczny rozmiar będzie wynosić co najmniej 25% pojemności twardego dysku.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz zawierający dodatnią liczbę całkowitą, **F**, oznaczającą liczbę plików na komputerze, spację i dodatnią liczbę dziesiętną, **C**, oznaczającą rozmiar dysku w GB.
* **F** wierszy zawierających następujące informacje. Elementy danych są oddzielane spacjami.
  + Datę ostatniej zmiany pliku w formacie DD/MM/RRRR. Data nie może być późniejsza niż 26 kwietnia 2019 r.
  + Godzinę ostatniej zmiany pliku w formacie HH:MM.
  + Skrót „AM” lub „PM” pokazujący, czy sygnatura czasowa odnosi się do godzin porannych czy popołudniowych.
  + Dodatnią liczbę całkowitą oznaczającą rozmiar pliku w KB.
  + Nazwę pliku. Nazwy plików mogą zawierać małe i wielkie litery, liczby i kropki (.).

1

10 1.0

25/04/2019 10:30 AM 125000 file1.txt

02/03/2019 02:15 PM 62500 file2.exe

01/01/2019 05:34 PM 62500 file3.mov

31/12/2018 11:36 AM 31250 file4.gif

14/02/2019 10:42 PM 31250 file5.doc

23/08/2018 12:00 PM 31250 file6.sh

29/02/2016 09:20 AM 31250 file7.mp3

05/12/2018 01:30 PM 15625 file8

26/04/2019 01:30 PM 15625 file9.png

01/01/2000 04:15 PM 1000 file10.jpg

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić podane poniżej informacje dla plików z najwyższą punktacją. Lista plików powinna zawierać pliki, których sumaryczny rozmiar jest co najmniej równy 25% **C**.

* Nazwa pliku
* Spacja
* Punktacja pliku, zaokrąglona do trzech miejsc dziesiętnych. Należy uwzględnić na miejscach dziesiętnych zera następujące (bez obcinania końcowych zer)

file7.mp3 36031.250

file6.sh 7703.125

file3.mov 7218.750

file10.jpg 7055.500

file4.gif 3656.250

file2.exe 3468.750

file5.doc 2234.375

# Problem 23: Ewakuacja!

**Punkty:** 70

**Autor:** Richard Green, Whiteley, Hampshire, Wielka Brytania

## Wprowadzenie do problemu

To wasz pierwszy dzień pracy w Lockheed Martin w roli inżyniera oprogramowania. Właśnie zakończyło się oprowadzanie, trafiacie do swojego biurka, pełna gotowość do pracy, a tu nagle...

ALARM!

Wybuchł pożar! Nie znacie jeszcze budynku i nie wiecie, którędy się wydostać! Na szczęście, współpracownicy pomagają się wam wydostać, a poza tym to były tylko ćwiczenia, ale zdarzenie daje wam do myślenia. A co, jeśli mielibyście na telefonie apkę, która poprowadzi was do najbliższego wyjścia? Dzielicie się pomysłem z menedżerem, a on zatwierdza projekt! (Szybka uwaga: Takie coś naprawdę może się wydarzyć!)

## Opis problemu

Wasz program będzie odczytywać obraz z planem piętra budynku, a następnie znajdzie najkrótszą drogę do wyjścia na zewnątrz przy zadanym położeniu startowym. Podczas poszukiwania najkrótszej ścieżki możecie przemieszczać się w każdym z głównych kierunków - góra, dół, lewo, prawo. Nie możecie przemieszczać się na ukos ani przechodzić przez ściany. Jeśli okaże się, że jest kilka dostępnych tras zapewniających najkrótszą drogę do wyjścia, należy wybrać tę, która kończy się najbliżej lewego górnego narożnika mapy. Chociaż mapa będzie prostokątna (lub kwadratowa), to plan budynku nie musi taki być.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze danych wejściowych:

* Wiersz składający się z dwóch liczb całkowitych oddzielonych spacjami:
  + Pierwsza liczba całkowita to szerokość mapy **W**
  + Druga liczba całkowita to wysokość mapy **H**
* **H** wierszy, każdy o długości maksymalnie **W** znaków, które stanowią mapę budynku.
  + Znak # (hashtag) oznacza ścianę budynku.
  + Spacja oznacza korytarz lub pomieszczenie, po którym można się poruszać.
  + Mała litera „o” oznacza położenie początkowe w budynku.
  + Duża litera „X” oznacza wyjście z budynku.
  + Wiersze mogą być krótsze niż **W** znaków; „brakujące” miejsca znajdują się poza budynkiem i nie są brane pod uwagę przez program. Należy pamiętać, że w wyniku na końcach wierszy nie mogą występować żadne spacje.

2

10 10

########X#

X ## #

##### #

# # ## #

# # # ## #

#o# # # #

### # ##

#### ## X

X ##

######X###

30 20

##X###############

## ####

##### ## #### #

# # ## # ### #

# # ## # # ### #

# ### # # #

# # ## #######

# ##### ## ###

# ###### #

##### #### # #

# #### ##### #

## ####### # #############

## # ###### #

# # ## # ########## #

### # ## # ##### ## #

# ### # # # # ##########

# # ## #o##### #

# ######## ################# #

# # #

##################X###########

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić oryginalną mapę budynku, na której najkrótsza droga do wyjścia zostanie zaznaczona kropkami (.) umieszczanymi zamiast spacji widocznych powyżej.

########X#

X ## #

##### #

#...# ## #

#.#.# ## #

#o#.# # #

###.. # ##

####.## X

X ... ##

######X###

##X###############

##....... ####

##### ##.#### #

# # ##.# ### #

# # ##.# # ### #

# ### # ...# #

# # ##.#######

# ##### ##.....###

# ######. #

##### ####.....# #

# ####.##### #

## #######.# #############

## #.###### #

# # ## #....... ########## #

### # ## # #####.## #

# ### # # #.....# ##########

# # ## #o##### #

# ######## ################# #

# # #

##################X###########

# Problem 24: Sudoku

**Punkty:** 80

**Autor:** Brett Reynolds, Orlando, Floryda, Stany Zjednoczone

## Wprowadzenie do problemu

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4** | ***6*** | **2** | ***5*** | ***7*** | ***1*** | ***8*** | ***3*** | ***9*** |
| ***9*** | ***1*** | ***3*** | ***4*** | **6** | ***8*** | ***5*** | ***7*** | ***2*** |
| ***7*** | **5** | **8** | **9** | ***2*** | ***3*** | ***1*** | ***4*** | ***6*** |
| ***1*** | **9** | ***4*** | ***7*** | **5** | ***6*** | ***2*** | **8** | ***3*** |
| ***8*** | ***2*** | ***7*** | **3** | **4** | ***9*** | ***6*** | **5** | **1** |
| ***6*** | ***3*** | ***5*** | ***8*** | ***1*** | ***2*** | ***4*** | ***9*** | ***7*** |
| ***5*** | ***4*** | **1** | **6** | ***9*** | ***7*** | **3** | **2** | ***8*** |
| ***2*** | **8** | ***9*** | **1** | ***3*** | ***4*** | ***7*** | ***6*** | ***5*** |
| **3** | ***7*** | **6** | **2** | ***8*** | ***5*** | **9** | ***1*** | **4** |

Sudoku to popularna zagadka logiczna, często umieszczana w gazetach, czasopismach i na stronach internetowych. Najprawdopodobniej powstała w 1979 r. w Indianie w USA. W latach 80. cieszyła się dużym uznaniem w Japonii, a w nowym tysiącleciu stała się zjawiskiem o skali światowej. Zwłaszcza gazety przyczyniły się do ugruntowania pozycji sudoku jako popularnej łamigłówki, ze względu na podobieństwo do krzyżowek.

Sudoku rozgrywa się na planszy 9 na 9 pól, podzielonych na kwadraty o boku 3 pól każdy. Każde pole jest wypełniane liczbami od 1 do 9 włącznie tak, aby po zapełnieniu planszy każda cyfra była obecna dokładnie raz w każdym wierszu, każdej kolumnie i każdym mniejszym kwadracie. Początkowo zagadka jest zawsze pusta, z jedynie kilkoma cyframi umieszczonymi jako podpowiedzi. Korzystając z tych podpowiedzi, rozwiązujący musi dojść do tego, jak wypełnić pozostałe pola na drodze eliminacji, logicznego rozumowania oraz metodą prób i błędów. Na pokazanej ilustracji liczby zaznaczone kolorem czarnym to podpowiedzi umieszczone w łamigłówce; liczby oznaczone kursywą i kolorem czerwonym zostały wpisane podczas rozwiązywania. Aby Sudoku było takie jak należy, przy danym zestawie podpowiedzi musi być możliwe tylko jedno, niepowtarzalne rozwiązanie.

Właściwości łamigłówek Sudoku zachęciły matematyków i naukowców z dziedziny nauk informatycznych do poświęcenia im wielu godzin badań. Sporo czasu poświecono badając minimalną liczbę wskazówek, jakie można umieścić w zagadce tak, by nadal możliwe było tylko jedno niepowtarzalne rozwiązanie (17), a także badając zagadki tworzone według określonych wzorców. Umiejętne rozwiązywanie Sudoku jest dość trudnym zadaniem w informatyce; mieści się ono w kategorii problemów znanych jako problemy „NP-zupełne”. Oznacza to, że uważa się, że nie istnieje algorytm, który mógłby rozwiązać Sudoku w czasie krótszym od czasu wielomianowego (bez pętli zagnieżdżonych co najmniej dwukrotnie).

## Opis problemu

Waszym zadaniem jest napisanie programu, który jest w stanie odczytać Sudoku i znaleźć jej rozwiązanie. Aby rozwiązać Sudoku, musicie wypełnić wszystkie puste pola liczbami z przedziału od 1 do 9 włącznie tak, aby każda z nich pojawiała się dokładnie raz w każdym rzędzie, każdej kolumnie i każdym kwadracie o boku 3 pól.

Wszystkie podane zagadki będą „właściwymi” łamigłówkami Sudoku; co oznacza, zgodnie z powyższym opisem, że każda z nich ma dokładnie jedno prawidłowe rozwiązanie.

## Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać dziewięć wierszy tekstu. Każdy wiersz będzie zawierać tylko liczby z przedziału od 1 do 9 włącznie oraz podkreślenia (\_). Podkreślenia to puste miejsca w zagadce, które należy wypełnić. Cyfry oznaczają podpowiedzi, które w rozwiązaniu powinny być w tym samym miejscu.

2

4\_2\_\_\_\_\_\_

\_\_\_\_6\_\_\_\_

\_589\_\_\_\_\_

\_9\_\_5\_\_8\_

\_\_\_34\_\_51

\_\_\_\_\_\_\_\_\_

\_\_16\_\_32\_

\_8\_1\_\_\_\_\_

3\_62\_\_9\_4

**\_\_\_\_\_\_\_\_\_**

**\_\_\_16\_\_52**

**\_7\_5\_4\_\_6**

**39\_\_\_\_\_\_\_**

**62\_\_\_\_39\_**

**\_\_\_\_\_\_\_6\_**

**9\_\_\_3\_\_\_\_**

**\_5\_71\_94\_**

**2\_\_6\_\_5\_7**

## Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić rozwiązaną łamigłówkę Sudoku, podając dziewięć wierszy po dziewięć cyfr każdy.

462571839

913468572

758923146

194756283

827349651

635812497

541697328

289134765

376285914

568327419

439168752

172594836

394286175

625471398

781953264

947835621

856712943

213649587

1. *Obraz do tego problemu jest objęty licencją Creative Commons Attribution 3.0 Unported License (więcej informacji znajduje się pod linkiem* [*https://creativecommons.org/licenses/by/3.0/*](https://creativecommons.org/licenses/by/3.0/)*), a jego autorami są Michael Gäbler i AzaToth z Wikimedia Commons.* *Można go swobodnie używać pod warunkiem podania autorstwa, zgodnie z warunkami licencji.* [↑](#footnote-ref-1)