Code Quest
Zestaw zadań

Doroczny konkurs międzynarodowy — edycja 2022

Sobota, 30 kwietnia 2022

Spis treści

Często zadawane pytania (FAQ)

Na czym polegają zawody?

W celu rozwiązania zadanego problemu wasz zespół ma za zadanie napisać program komputerowy, który odczytuje dane wejściowe ze standardowego kanału wejściowego i podaje dane wyjściowe do konsoli. W opisie każdego problemu podano format danych wejściowych i format, w jakim powinny zostać przedstawione dane wyjściowe. Po ukończeniu programu należy przesłać kod źródłowy programu na stronę internetową konkursu. Strona kompiluje i uruchamia kod, a uczestnicy otrzymują informację, czy odpowiedź jest prawidłowa czy nie.

Kto ocenia odpowiedzi?

W Lockheed Martin wyznaczono zespół pracowników odpowiedzialnych za ocenę rozwiązań konkursowych, ale większość tego procesu jest prowadzona automatycznie przez stronę konkursową. Strona kompiluje i uruchamia kod, a następnie porównuje dane wyjściowe z programu uczestnika do oficjalnych wyników. Jeśli są one jednakowe, wasz zespół otrzymuje punkty za podanie prawidłowej odpowiedzi.

Jak wygląda punktacja każdego problemu?

Każdy z nich ma przypisywaną wartość punktową w zależności od poziomu trudności. Gdy strona internetowa uruchamia wasz program, porównuje jego wyniki z oficjalnymi wynikami dla danego problemu. Jeśli są one jednakowe, wasz zespół otrzymuje punkty za rozwiązanie problemu. Za częściową zgodność punkty nie są przyznawane; dane wyjściowe z przesłanego programu muszą być dokładnie takie, jak oficjalne wyniki. Jeśli wasza odpowiedź została zweryfikowana negatywnie, a jesteście pewni, że jest prawidłowa, ponownie sprawdźcie format danych wyjściowych, a także, czy na końcu nie pojawiły się zbędne spacje lub inne niepotrzebne znaki.

Nie rozumiemy problemu. Kto może nam pomóc?

W przypadku trudności ze zrozumieniem problemu możecie przesyłać pytania do organizatorów za pośrednictwem strony internetowej konkursu. Nie możemy podpowiadać, jak dojść do rozwiązania, ale objaśniamy ewentualne niejasności. Jeśli zespół pracowników organizatora wykryje w trakcie zawodów błąd w jakimś problemie, możliwie jak najszybciej powiadomi o tym wszystkie uczestniczące w zawodach zespoły.

Nasz program działa z przykładowymi danymi wejściowymi/wyjściowymi, ale w ramach zawodów jest weryfikowany negatywnie! Dlaczego?

Pamiętajcie, że oficjalne dane wejściowe i wyjściowe służące do oceny waszych odpowiedzi są ZNACZNIE większe od przekazanych przykładowych danych. Te dane obejmują szerszy zakres badanych przypadków. Opis problemu zawsze wskazuje wartości graniczne danych wejściowych i wyjściowych, ale program musi być w stanie obsługiwać również przypadek wykraczający poza te granice. Wszystkie dane wejściowe i wyjściowe zostały gruntownie przetestowane przez nasz zespół ds. problemów i nie zawierają błędnych wartości.

Nie jesteśmy w stanie dojść do tego, dlaczego nasza odpowiedź jest nieprawidłowa. Co robimy źle?

Do najczęstszych błędów należą:

- Nieprawidłowe formatowanie - Warto dokładnie przyjrzeć się przykładowym danym wyjściowym podanym w problemie i sprawdzić, czy wasz program prezentuje wyniki w tym samym formacie.
- Nieprawidłowe zaokrąglenie - W następnym rozdziale opisano zaokrąglanie liczb dziesiętnych.
- Niewłaściwe liczby - 0 (a także 0,0, 0,00 itd.) NIE JEST liczbą ujemną. 0 może być dopuszczalną odpowiedzią, ale -0 już nie.
- Niepotrzebne znaki - Sprawdźcie, czy na końcu wierszy z wynikami nie ma dodatkowych spacji. Spacje na końcu wiersza nie są częścią danych wyjściowych w żadnym problemie.
- Format dziesiętny - We wszystkich wynikach liczbowych separatorem dziesiętnym jest kropka (.).

Jeśli te podpowiedzi są niewystarczające, możecie przesyłać pytania do organizatorów za pośrednictwem strony internetowej konkursu. Nie możemy podpowiadać, jak dojść do rozwiązania, ale zwykle możemy wyjaśnić, dlaczego dana odpowiedź jest zwracana jako nieprawidłowa.

Podczas przesyłania rozwiązania pojawia się błąd.

Przy przesyłaniu rozwiązań należy wybrać kod źródłowy dla swojego programu (w zależności od języka będą to pliki o rozszerzeniu .java, .cs, .cpp, lub .py). Należy pamiętać o przesłaniu wszystkich plików potrzebnych do skompilowania i uruchomienia programu. Należy również sprawdzić, czy nazwy plików nie zawierają spacji lub innych znaków spoza zakresu znaków alfanumerycznych (np. „Prob01.java” jest dopuszczalna, ale „Prob 01.java” i „Bob’sSolution.java” nie są).

Czy po zakończeniu zawodów mogę dostać rozwiązania problemów?

Tak! Poproście swojego opiekuna o przesłanie emaila na adres codequest-poland.gr-sac@lmco.com.

Jak ustala się kolejność w przypadku remisu?

Na koniec zawodów zespoły są szeregowane według liczby punktów uzyskanej z prawidłowych odpowiedzi na problemy. Jeśli w którejkolwiek kategorii konkursowej miejsca na podium są zajmowane przez zespoły z tą samą liczbą punktów, to kolejność ustala się według następujących kryteriów:

1. Mniej rozwiązanych problemów (co sugeruje, że w zespole rozwiązywano trudniejsze zadania)
2. Mniej nieprawidłowych odpowiedzi (co sugeruje, że popełniono mniej błędów)
3. Kolejność przesłania ostatniej prawidłowej odpowiedzi; im wcześniej, tym lepiej (co sugeruje, że pracowano szybciej)

Proszę pamiętać, że metody ustalania kolejności mogą nie być zawsze uwzględniane na wyświetlanej na żywo tabeli wyników na stronie internetowej zawodów. Ponadto, na 30 minut przed końcem zawodów tabela wyników zostaje zamrożona, dlatego należy przede wszystkim skupić się na wytężonej pracy!

Informacje matematyczne

Zaokrąglanie

W niektórych problemach zostaniecie poproszeni o zaokrąglanie liczb. We wszystkich stosuje się domyślnie zaokrąglanie „od połowy w górę” , chyba że podano inaczej w opisie problemu. W większości przypadków będzie to sposób, którego nauczyliście się w szkole, ale niektóre języki programowania stosują domyślnie inne metody zaokrąglania. **O ile nie jesteście pewni tego, jak wasz język programowania obsługuje zaokrąglanie, zalecamy napisanie własnego kodu do zaokrąglania liczb na podstawie informacji podanych w tym rozdziale.**

W metodzie zaokrąglania „od połowy w górę” liczby są zaokrąglane do najbliższej liczby całkowitej. Na przykład:

- 1.49 jest zaokrąglane w dół do 1.
- 1.51 jest zaokrąglane w górę do 2.

Od połowy w górę” oznacza, że gdy liczba jest dokładnie w połowie między dwoma liczbami całkowitymi, to jest zaokrąglana do liczby z większą wartością bezwzględną (czyli dalszą od 0). Na przykład:

- 1.5 jest zaokrąglane w górę do 2.
- -1.5 jest zaokrąglane w dół do -2.

Błędy zaokrąglania należą do najpowszechniejszych; jeśli w problemie występuje zaokrąglanie i strona internetowa odrzuca program jako podający nieprawidłowe wyniki, warto sprawdzić zaokrąglanie!

Trygonometria

Niektóre problemy mogą wymagać zastosowania funkcji trygonometrycznych, które pokrótce przedstawiono poniżej. Większość języków programowania ma wbudowane funkcje dla $\sin X$, $\cos X$ i $\tan X$; Należy uważnie przeczytać jego dokumentację. O ile nie podano inaczej w opisie problemu, *mocno zalecamy*, by używać wbudowanej w języku wartości dla pi (π), jeśli to koniecznie.

$$\sin X = \frac{A}{C} \quad \cos X = \frac{B}{C} \quad \tan X = \frac{A}{B} = \frac{\sin X}{\cos X}$$

$$X + Y = 90°$$

$$A^2 + B^2 = C^2$$

$$\frac{stopnie * \pi}{180} = radiany$$

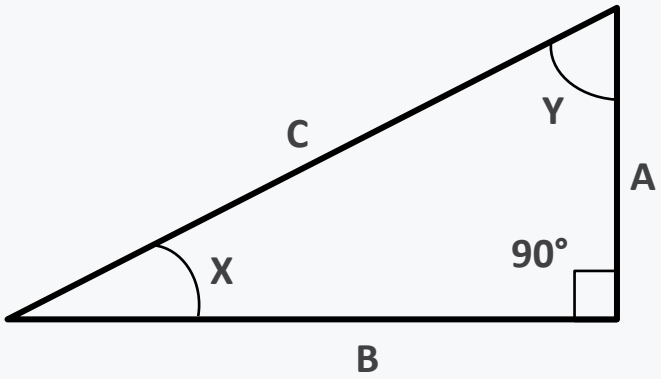


Tabela US ASCII

Dane wejściowe we wszystkich problemach Code Quest korzystają ze znaków drukowalnych należących do tabeli znaków US-ASCII. Znaki niedrukowalne i sterujące nie są używane w problemach, chyba że wyraźnie zaznaczono to w opisie problemu. W niektórych przypadkach może pojawić się polecenie zamiany znaku na jego równoważnik liczbowy i/lub *vice versa*, zgodnie z poniższą tabelą.

Dwójkowy	Dziesiętny	Znak	Dwójkowy	Dziesiętny	Znak	Dwójkowy	Dziesiętny	Znak
0100000	32	(spacja)	1000000	64	@	1100000	96	`
0100001	33	!	1000001	65	A	1100001	97	a
0100010	34	"	1000010	66	B	1100010	98	b
0100011	35	#	1000011	67	C	1100011	99	c
0100100	36	$	1000100	68	D	1100100	100	d
0100101	37	%	1000101	69	E	1100101	101	e
0100110	38	&	1000110	70	F	1100110	102	f
0100111	39	'	1000111	71	G	1100111	103	g
0101000	40	(	1001000	72	H	1101000	104	h
0101001	41	)	1001001	73	I	1101001	105	i
0101010	42	*	1001010	74	J	1101010	106	j
0101011	43	+	1001011	75	K	1101011	107	k
0101100	44	,	1001100	76	L	1101100	108	l
0101101	45	-	1001101	77	M	1101101	109	m
0101110	46	.	1001110	78	N	1101110	110	n
0101111	47	/	1001111	79	O	1101111	111	o
0110000	48	0	1010000	80	P	1110000	112	p
0110001	49	1	1010001	81	Q	1110001	113	q
0110010	50	2	1010010	82	R	1110010	114	r
0110011	51	3	1010011	83	S	1110011	115	s
0110100	52	4	1010100	84	T	1110100	116	t
0110101	53	5	1010101	85	U	1110101	117	u
0110110	54	6	1010110	86	V	1110110	118	v
0110111	55	7	1010111	87	W	1110111	119	w
0111000	56	8	1011000	88	X	1111000	120	x
0111001	57	9	1011001	89	Y	1111001	121	y
0111010	58	:	1011010	90	Z	1111010	122	z
0111011	59	;	1011011	91	[	1111011	123	{
0111100	60	<	1011100	92	\	1111100	124	|
0111101	61	=	1011101	93	]	1111101	125	}
0111110	62	>	1011110	94	^	1111110	126	~
0111111	63	?	1011111	95	_			

Terminologia

W całym pakiecie opisujemy dane wejściowe i wyjściowe stosowane w waszych programach. W celu rozwiania wątpliwości pewne pojęcia są zawsze używane do określania różnych właściwości tych danych. Te pojęcia przedstawiono poniżej.

- **Liczba całkowita** to każda liczba bez ułamków zwykłych lub dziesiętnych. Liczbami całkowitymi są np. -5, 0, 5 i 123456789.
- **Liczba dziesiętna** to każda liczba, która nie jest liczbą całkowitą. Takie liczby mają separator dziesiętny i co najmniej jedną cyfrę na prawo od niego. -1.52, 0.0 i 3.14159 są liczbami dziesiętnymi.
- **Miejsca dziesiętne** oznaczają liczbę cyfr w liczbie dziesiętnej, która następuje po separatorze dziesiętnym. O ile nie podano inaczej w opisie problemu, liczby dziesiętne mogą zawierać dowolną liczbą miejsc dziesiętnych, równą co najmniej 1.
- **Liczba szesnastkowa** lub **łańcuch** stanowią szereg zawierający jeden znak lub więcej, w tym cyfry od 0 do 9 i/lub wielkie litery - A, B, C, D, E i/lub F. W tym konkursie małe litery nie są używane w wartościach szesnastkowych.
- **Liczby dodatnie** to liczby większe od 0. 1 to najmniejsza dodatnia liczba całkowita. 0.000000000001 to bardzo mała dodatnia liczba dziesiętna.
- **Liczby niedodatnie** to liczby, które nie są dodatnie; innymi słowy, wszystkie liczby mniejsze od 0 lub równe 0.
- **Liczby** ujemne to liczby mniejsze od 0. -1 to największa ujemna liczba całkowita. -0.000000000001 to bardzo duża ujemna liczba dziesiętna.
- **Liczby nieujemne** to liczby, które nie są ujemne; innymi słowy, wszystkie liczby większe od 0 lub równe 0.
- Włącznie oznacza, że przedział określony podanymi wartościami obejmuje również te podane wartości (czyli jest domknięty). Na przykład, przedział „od 1 do 3 włącznie” oznacza liczby 1, 2 i 3.
- **Wyłącznie** oznacza, że przedział określony podanymi wartościami nie obejmuje tych podanych wartości (czyli jest otwarty). Na przykład, przedział „od 0 do 4 wyłącznie” oznacza liczby 1, 2 i 3; 0 i 4 nie zawierają się w przedziale.
- **Formaty daty i godziny** są podawane literami, w miejsce których winny pojawić się liczby:
  - **HH** oznacza godziny, zapisywane dwoma cyframi (z ewentualnym zerem poprzedzającym). Opis problemu zawiera informację, czy należy używać formatu 12- czy 24-godzinnego.
  - **MM** oznacza minuty w przypadku czasu, a miesiące w przypadku dat. W obydwu przypadkach liczby są zapisywane dwoma cyframi (z ewentualnym zerem poprzedzającym; styczeń to 01).
  - **YY** lub YYYY oznacza rok, zapisywany dwoma lub czterema cyframi (z ewentualnym zerem poprzedzającym).
  - **DD** oznacza dzień miesiąca, zapisywany dwoma cyframi (z ewentualnym zerem poprzedzającym).

Problem 01: Szyfr zerowy

Punkty: 5

Autor: Javier Jimenez, Marietta, Georgia, Stany Zjednoczone

Kontekst problemu

Istnieją dwie podstawowe metody ukrywania informacji, które chcesz zachować w tajemnicy. Kryptografia wykorzystuje algorytm lub podobny proces do przekształcenia wiadomości w inną formę, czyniąc ją nieczytelną. Celem steganografii jest po prostu ukrycie wiadomości tak, aby niedoszły podsłuchiwacz nie zorientował się, że wiadomość w ogóle istnieje. Steganografia przybierała w historii wiele różnych form: wiadomości zapisane niewidzialnym atramentem, wzory alfabetu Morse'a wydziergane na swetrach, wiadomości skurczone do mikroskopijnych rozmiarów i wydrukowane na przezroczystej folii. Jedna z najprostszych form steganografii znana jest jako „szyfr zerowy”.

Opis problemu

Szyfr zerowy jest skuteczny, ponieważ sprawia wrażenie całkowicie nieszkodliwej wiadomości. Tajna wiadomość — zwana szyfrogramem — jest ukryta w innej wiadomości przez dodanie dużej liczby „zerowych” wartości, słów lub liter, które nie mają nic wspólnego z oryginalną wiadomością. W idealnej sytuacji podsłuchujący widziałby wiadomość i nie zdawałby sobie sprawy, że w środku ukryta jest druga wiadomość. Natomiast odbiorca wiedziałby, że należy usunąć pewne słowa lub znaki, aby przywrócić oryginalną wiadomość.

Firma Lockheed Martin współpracuje z amerykańską Agencją Bezpieczeństwa Narodowego (NSA) w celu przetestowania nieco innej formy szyfru zerowego. NSA zamierza umieścić wiadomość w ciągu losowo wybranych znaków. Pracownicy agencji żywią nadzieję, że podsłuchujący *będzie* podejrzewał, że wiadomość jest ukryta, ale założy, że losowy charakter wiadomości oznacza, że jest ona zaszyfrowana za pomocą szyfru, i straci czas, próbując go złamać. W rzeczywistości wiadomość będzie po prostu rozrzucona po całym ciągu tekstu. Każdy znak, który jest częścią rzeczywistego komunikatu, występuje bezpośrednio po angielskiej samogłosce, czyli po jednej z liter: a, e, i, o lub u. Gdy te znaki wystąpią w rzeczywistym komunikacie i będą następowały po innej angielskiej samogłosce to znak po nich nie jest częścią komunikatu. Na przykład, poniższy ciąg znaków można odczytać jako „hello world”:

fksahnlgueyilfhnalfkjnhdssaokjfhndsfiwaourhnfdjgbalfkjshedfnsf

Mając do dyspozycji kilka przykładowych ciągów znaków wygenerowanych przez NSA, zaprojektuj program, który potrafi wyodrębnić oryginalne wiadomości.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz tekstu składający się z małych liter.

```
3
fksahnlgueyilfhnalfkjnhdssaokjfhndsfiwaourhnfdjgbalfkjshedfnsf
mkjmnacioudhrieeqwthyiugueresjfgwatfhwghfnhgnffn
elruoqywicwnjksakvfbsgyohuehnghiefhggadfgsfsfs
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić pojedynczy wiersz zawierający wiadomość w postaci jawnego tekstu wyodrębnionego z ciągu wejściowego.

```
helloworld
codequest
lockheed
```

Problem 02: Strata przy wycieku

Punkty: 5

Autor: Tai Do, Sunnyvale, Kalifornia, Stany Zjednoczone

Kontekst problemu

Twoja rodzina ma basen naziemny, który został przez Ciebie opróżniony przed zimą, aby nie zamarzł i nie uległ uszkodzeniu. Zbliża się lato, a rodzice poprosili Cię o ponowne napełnienie zbiornika za pomocą węża ogrodowego. Basen napełnia się dużo wolniej niż zwykle, ale w końcu udaje się go napełnić. Kiedy chcesz wyciągnąć węża z basenu zauważasz, że zawór spustowy jest nadal otwarty i wszędzie wylewa się woda! Udało Ci się zamknąć zawór, ale pytanie brzmi: ile wody właśnie zmarnowałeś?

Opis problemu

Oblicz, ile wody wyciekło z basenu podczas jego napełniania. Po napełnieniu basenu wiadomo, że, ilość wody wpływającej do basenu była większa niż ilość wody z niego wypływającej. W związku z tym możesz skorzystać z tego wzoru, aby obliczyć straty:

$$\frac{\textit{Pojemność basenu}}{\textit{(Prędkość napełniania – Prędkość wycieku)}} * \textit{Prędkość wycieku} = \textit{Objętość straty}$$

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz z trzema liczbami dodatnimi, rozdzielonymi spacjami:

- Pojemność basenu w litrach
- Szybkość, z jaką woda napływała do basenu, w litrach na minutę
- Szybkość, z jaką woda wyciekała z basenu, w litrach na minutę

```
2
700 10 3
2000 100 20
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wartość ilości wody, która wyciekła z basenu podczas jego napełniania, w zaokrągleniu do pełnego litra.

```
300
500
```

Problem 03: Dobre dane wejściowe ze statku

Punkty: 10

Autor: Lucas Doty, Orlando, Floryda, Stany Zjednoczone

Kontekst problemu

Aplikacje utrzymania ruchu na poziomie przedsiębiorstwa są powszechne w większości branż, a szczególnie ważne są dla wszystkich rodzajów wojsk. Firma Lockheed Martin dostarcza takie aplikacje wojskom na całym świecie. Aby utrzymać samoloty, ciężarówki, okręty i wszelkie inne pojazdy w stanie gotowości bojowej, żołnierze muszą wiedzieć jakie części należy sprawdzić lub poddać konserwacji w określonym czasie. Muszą także wiedzieć, kiedy dana część może osiągnąć koniec okresu eksploatacji, aby można ją było wymienić. Śledzenie wszystkich prac konserwacyjnych ma kluczowe znaczenie dla zapewnienia powodzenia misji i ograniczenia liczby wypadków do absolutnego minimum.

Tworzenie takich aplikacji wymaga jednak wiele pracy. Wykorzystywane są różnego rodzaju pliki, komunikaty i interfejsy, które są także przesyłane poprzez warstwy aplikacji logistycznej. Mogą one łączyć się z bazami danych w celu przechowywania informacji i prowadzenia archiwów. Celem uzyskania aktualnych informacji może zaistnieć potrzeba komunikacji z innymi aplikacjami. Użytkownicy muszą mieć także możliwość połączenia się z aplikacją, aby uzyskać dostęp do wszystkich tych danych. Niezwykle ważne jest, aby mieli dostęp do potrzebnych im informacji dokładnie wtedy, kiedy ich potrzebują.

Opis problemu

Pracujesz w dziale Rotary and Mission Systems firmy Lockheed Martin nad stworzeniem nowego systemu obsługi technicznej dla Marynarki Wojennej Stanów Zjednoczonych. Marynarka wojenna chce zautomatyzować niektóre czynności związane z raportowaniem, które muszą zostać wykonane, gdy okręt jest wprowadzany do suchego doku w celu przeprowadzenia konserwacji. Dowództwo w szczególności chce, aby sam statek był w stanie informować, które systemy są sprawne i mogą być pominięte podczas inspekcji.

Gdy statek zostanie wprowadzony do suchego doku, komputer pokładowy przeprowadzi kilka automatycznych testów diagnostycznych systemów pokładowych krytycznego znaczenia. Po zakończeniu testów komputer przekaże do komputera znajdującego się w stoczni listę systemów, które pomyślnie przeszły testy. Komputer ten porówna listę sprawnych systemów z listą systemów znajdujących się na statku — ta zostanie pobrana z bazy danych. Wszelkie systemy, które pojawiają się w bazie danych, ale nie ma ich na liście zgłoszonej przez komputer pokładowy, należy zgłosić do przeglądu personelowi obsługi technicznej.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite rozdzielone spacjami:
  - Pierwsza liczba całkowita (**X**) oznacza liczbę systemów wymienionych w bazie danych stoczni.
  - Druga liczba całkowita (**Y**) oznacza liczbę systemów zgłoszonych przez komputer pokładowy. **Y** będzie mniejsze lub równe **X**.
- **X** wierszy zawierających nazwy systemów wymienionych w bazie danych stoczni. Nazwy są unikalne w ramach danego przypadku testowego i mogą zawierać litery oraz spacje.
- **Y** wierszy zawierających nazwy systemów zgłoszonych przez komputer pokładowy jako sprawne. Wszystkie nazwy w tej sekcji będą duplikatami nazw przechowywanych w bazie danych stoczni.

```
1
5 3
Cannon
Engine
Helm
Deck
Anchor
Engine
Helm
Anchor
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego program musi zwrócić listę systemów wymagających sprawdzenia w porządku alfabetycznym (wielkość liter nie ma znaczenia), po jednym systemie w wierszu.

```
Cannon
Deck
```

Problem 04: Specjalne traktowanie

Punkty: 15

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

Kontekst problemu

Znaki specjalne — ogólnie rzecz ujmując — to wszystkie znaki inne niż litera, cyfra lub spacja; mogą często powodować problemy w oprogramowaniu. Ciąg tekstu z umieszczonym w nim cudzysłowem może zostać zinterpretowany niepoprawnie. Ukośnik wsteczny przed literą może uniemożliwić odczytanie tej litery jako litery. W skrajnych przypadkach znaki specjalne mogą zostać wykorzystane do wstrzyknięcia złośliwego kodu do aplikacji, co umożliwi hakerom kradzież danych, a może nawet coś gorszego! Właściwa obsługa znaków specjalnych jest zatem głównym problemem dla firm programistycznych, w tym Lockheed Martin.

Przyjmuje się, że komputery działają zgodnie z zasadą GIGO (ang. Garbage In, Garbage Out; śmieci na wejściu – śmieci na wyjściu). Upewnijmy się, że wynosimy śmieci.

Opis problemu

Twój program będzie musiał wczytać kilka wierszy tekstu zawierającego różne znaki. Musisz usunąć z ciągu wszystkie znaki, które nie są literami, cyframi, ani spacjami, a następnie wypisać wszystkie litery, cyfry i spacje, które pozostały.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy zawiera wiersz tekstu do przetworzenia, który może zawierać dowolny drukowalny znak ASCII.

```
3
This “is” amazing!
Can’t you - yes, you - remove all the special characters?
Anything /not/ a letter, numb3r, or _space_ should go.
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wiersz tekstu z usuniętymi wszystkimi znakami specjalnymi.

```
This is amazing
Cant you  yes you  remove all the special characters
Anything not a letter numb3r or space should go
```

Problem 05: Podstępne zegarowe karty obecności

Punkty: 20

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

Kontekst problemu

W każdym środowisku pracy śledzenie czasu pracy jest ważnym czynnikiem gwarantującym otrzymanie wynagrodzenia. Duże korporacje, takie jak Lockheed Martin, również wykorzystują informacje z zegarowych kart obecności jako podstawę do szacowania kosztów potencjalnych kontraktów — sprawdzając ile czasu poświęcono na poprzednie kontrakty, można domniemywać ile czasu będzie potrzebne na wykonanie podobnej pracy.

Opis problemu

Twój menedżer wyjechał na urlop i poprosił swoją asystentkę, aby ta pod jego nieobecność zajęła się niektórymi z jego obowiązków. Wśród obowiązków jest zatwierdzanie zegarowych kart obecności wszystkich członków zespołu. Niestety, dział IT właśnie wyłączył system ewidencji czasu pracy z powodu prac konserwacyjnych i jego ponownie uruchomienie nie będzie możliwe przed upływem terminu zatwierdzenia! Dobra wiadomość jest taka, że dostęp do informacji z zegarowych kart obecności można uzyskać z innego systemu, ale konieczne będzie ich zsumowanie. Asystentka menedżera poprosiła Cię o pomoc w napisaniu programu, który odczyta te dane i zsumuje je dla niej tak, aby mogła szybko zatwierdzić zegarowe karty obecności wszystkich osób po ponownym uruchomieniu systemu.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz z następującymi informacjami, rozdzielonymi przecinkami:

- Ciąg znaków zawierający imię i nazwisko pracownika zapisane dużymi i małymi literami oraz co najmniej jedną spację. Wszystkie nazwiska pracowników będą niepowtarzalne.
- Pięć wartości czasu, każda w formacie GG:MM, reprezentujących ilość czasu przepracowanego przez danego pracownika w każdym dniu tygodnia (od poniedziałku do piątku).

```
3
Peter Gibbons,01:23,04:16,00:59,02:23,00:00
Milton Waddams,08:00,08:00,08:00,08:00,08:00
Bill Lumbergh,08:31,07:59,06:01,08:55,05:30
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić jeden wiersz zawierający następujące informacje:

- Imię i nazwisko pracownika, zgodnie z wprowadzonymi danymi
- Znak równości (=)
- Całkowity czas przepracowany przez pracownika w ciągu tygodnia, w formacie "H – godziny M – minuty", z zastrzeżeniem poniższych punktów:
    - Jeśli pracownik przepracował od 1 (włącznie) do 2 (wyłącznie) godzin, zamiast słowa „hour” program musi zwrócić słowo „hours”.
    - Jeśli liczba minut jest równa 1, zamiast słowa „minutes” program musi zwrócić słowo „minute”.
    - Jeśli liczba minut jest równa 0, pomiń liczbę minut; zamiast tego program musi zwrócić po prostu liczbę godzin, bez żadnych końcowych znaków niedrukowalnych.

```
Peter Gibbons=9 hours 1 minute
Milton Waddams=40 hours
Bill Lumbergh=36 hours 56 minutes
```

Problem 06: Znajdź brakujący czujnik

Punkty: 25

Autor: Sowmya Chandrasekaran, Sunnyvale, Kalifornia, Stany Zjednoczone

Kontekst problemu

Każda produkowana rzecz musi zawsze zostać przetestowana, aby upewnić się, że wszystkie wymagane części są na swoim miejscu i działają prawidłowo. Powyższe może obejmować diagnostykę elementów elektronicznych w celu zapewnienia, że są one w stanie prawidłowo odbierać dane wejściowe i wysyłać dane wyjściowe. Jeśli zostaną wykryte jakiekolwiek błędy, należy je naprawić, zanim produkt zostanie wysłany do klienta — o wiele łatwiej i mniej kosztownie jest usunąć problemy zanim produkt opuści halę produkcyjną!

Opis problemu

Inżynier materiałowy pracujący w Lockheed Martin Aeronautics testuje czujniki lotu montowane w myśliwcu F-35. Test diagnostyczny, który przeprowadza, wiąże się z wysłaniem sygnału do każdego czujnika; czujniki muszą odpowiedzieć liczbą reprezentującą ich unikalny identyfikator. Niestety, inżynier zauważył problem: brakuje jednego z czujników! Poprosił twój zespół o napisanie programu, który szybko zidentyfikuje brakujący czujnik.

Otrzymasz wyniki diagnostyki przeprowadzonej przez inżyniera. Będzie ona zawierać losowo posortowaną listę identyfikatorów każdego czujnika w formie liczby całkowitej. Każda liczba jest unikalna i należy do zakresu od 1 do liczby czujników, których znalezienia spodziewał się inżynier (N). Brakuje jednej z liczb od 1 do N — inżynierowi należy zgłosić której liczby brakuje.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający jedną dodatnią liczbę całkowitą (**N**) reprezentującą oczekiwaną liczbę czujników.
- Wiersz zawierający listę liczb całkowitych **N-1**, rozdzielonych spacjami i zawierających się w zakresie od 1 do **N** włącznie, reprezentujących identyfikatory czujników roboczych.

```
2
9
5 7 3 2 8 1 4 9
10
7 4 1 10 8 2 9 6 3
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić jeden wiersz zawierający liczbę całkowitą reprezentującą identyfikator brakującego czujnika.

```
6
5
```

Problem 07: Siła do wytrwałości

Punkty: 25

Autor: dr Francis Manning, Owego, Nowy Jork, Stany Zjednoczone

Kontekst problemu

Kiedy rozważamy budowę zautomatyzowanej maszyny do podróżowania i eksploracji poza obecnym zasięgiem ludzkich możliwości — czy to pod powierzchnią morza, w kalderze wulkanu, czy gdzieś poza ziemską atmosferą — zrozumienie ograniczeń i możliwości naszego urządzenia ma zasadnicze znaczenie dla pomyślnej realizacji misji. Jeśli maszyna zepsuje się w tak ekstremalnym środowisku, nie można po prostu wysłać mechanika, aby ją naprawił. Musi być gotowa do podjęcia każdego wyzwania, jakie może napotkać, a jej operatorzy muszą rozumieć, jakie wyzwania przekraczają jej możliwości.

NASA współpracuje z firmą Lockheed Martin nad zaprojektowaniem nowego łazika księżycowego, który mógłby być wykorzystany w przyszłych bezzałogowych misjach na Księżyc. Twój zespół pracuje nad projektem systemu napędowego łazika i ocenia kilka możliwych konstrukcji. Przed przystąpieniem do budowy prototypu musisz ocenić specyfikacje, aby zweryfikować, czy będą miały szansę i możliwość działać.

Opis problemu

Koła łazika będą napędzane przez serię serwomotorów. Każdy silnik może obracać się z określoną prędkością (mierzoną w obrotach na minutę — obr./min lub ang. RPM); każdy obrót wymaga określonej ilości energii z akumulatorów łazika. Aby wykonać jeden obrót kół łazika, potrzeba kilku obrotów silnika. Zadaniem twojego zespołu jest przeanalizowanie danych dotyczących całego procesu i określenie, czy systemy zasilania i napędu łazika są w stanie doprowadzić go do celu; jeśli tak, to ile czasu zajmie dotarcie do wyznaczonego celu.

Istnieje kilka wzorów, które pomogą w dokonaniu tych ustaleń:

- Obwód okręgu: $C = \pi d$
  - d = średnica okręgu
- Moc (mierzona w watach): $P = IV$
  - I = prąd (mierzony w amperach)
  - V = napięcie (mierzone w woltach)

Na przykład: przeanalizujmy silnik zasilany napięciem 12 V, zużywający 6 W mocy na wykonanie jednego obrotu i pracujący z prędkością 10 obr./min. Silnik potrzebuje 6 obrotów, aby obrócić koło o średnicy 15 cm. Łazik musi pokonać dystans 5 metrów. Każdy obrót koła pozwala łazikowi przemieścić się na odległość 47,12 cm; zatem na dystansie pięciu metrów (500 cm) wymaga to wykonania około 10,61 obrotu koła. To z kolei wymaga wykonania 63,66 obrotów silnika, co zajmie 6,367 minuty. Każdy obrót silnika wymaga 6 watów mocy, dlatego w czasie trwania misji silnik pobierze łącznie 381,96 watów. Dzieląc tę wartość przez napięcie (12 V), otrzymamy wynik: do uzyskania takiej mocy potrzeba prądu o

natężeniu 31,83 A. Mnożąc to przez czas trwania misji, otrzymujemy całkowitą ilość energii, którą muszą dostarczyć systemy energetyczne: 202,6616 amperominut lub 3,3777 amperogodzin. Tak długo, jak systemy zasilania będą w stanie dostarczyć tak duży prąd w takim czasie, misja powinna zakończyć się sukcesem.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz z siedmioma liczbami dodatnimi, rozdzielonymi spacjami i reprezentującymi poniższe wartości (kolejno):

- Średnica koła łazika w centymetrach (cm)
- Liczba obrotów silnika wymagana do wykonania pełnego obrotu koła
- Moc potrzebna do wykonania jednego obrotu silnika, w watach
- Prędkość obrotowa silnika, w obrotach na minutę
- Dostępna pojemność układu zasilania w amperogodzinach
- Wymagane napięcie silnika
- Wymagana odległość, jaką musi pokonać łazik, w metrach

```
2
15 6 6 10 4 12 5
12 8 12 10 24 12 8
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić jeden wiersz zawierający następujące informacje:

- Jeśli misja jest możliwa do wykonania przy podanych parametrach:
  - Słowo „Success”
  - Spację
  - Czas potrzebny na pokonanie przez łazik podanego dystansu, w minutach, zaokrąglony do czterech miejsc po przecinku. Uwzględnij wszystkie zera końcowe.
- Jeśli wykonanie misji nie jest możliwe przy podanych parametrach, zwrócona musi zostać informacja „Fail”.

```
Success 6.3662
Fail
```

Problem 08: Przechowywanie DNA

Punkty: 30

Autor: Brett Reynolds, Annapolis Junction, Maryland, Stany Zjednoczone

Kontekst problemu

Zdolność do przechowywania i pobierania danych jest krytycznym elementem każdego systemu komputerowego. Jednak wraz z rozwojem coraz potężniejszych systemów komputerowych staje się to jeszcze większym wyzwaniem. Aby nadążyć za tempem, w jakim nasze komputery są w stanie przetwarzać dane, musimy opracować systemy, które będą w stanie bardzo szybko przechowywać bardzo duże ilości danych. Jak dotąd osiągnęliśmy znaczny postęp — ilość danych, które można przechowywać na jednej karcie MicroSD, jeszcze pół wieku temu wymagałaby całych pomieszczeń wypełnionych taśmami magnetycznymi. Musimy jednak nadal poszukiwać bardziej efektywnych metod przechowywania, a najnowsze badania wykazały, że być może rozwiązanie to mamy w sobie od zawsze.

DNA (ang. deoxyribonucleic acid), czyli kwas deoksyrybonukleinowy, to złożona cząsteczka występująca w komórkach każdej żywej istoty. DNA zawiera instrukcje dotyczące funkcjonowania naszych komórek; jest to w istocie „język programowania" dla życia. Aby opisać jak powinien funkcjonować człowiek, roślina czy zwierzę, potrzeba bardzo dużo DNA, a to przekłada się na dużą ilość danych. Z tego powodu w najnowszych badaniach podjęto próbę zbadania sposobów wykorzystania cząsteczek DNA do przechowywania i pobierania danych obliczeniowych. Pomimo ilości informacji jakie może przechowywać DNA, jest ono nadal cząsteczką, a zatem w kategoriach zajmowanej przestrzeni stanowi niezwykle oszczędny sposób przechowywania danych.

Lockheed Martin współpracuje ze startupem biotechnologicznym nad opracowaniem urządzenia, które może odczytywać (lub „sekwencjonować") cząsteczki DNA w celu przekształcenia danych w binarne dane komputerowe. Uwzględniając strukturę cząsteczki DNA, twoje urządzenie będzie musiało przetłumaczyć i wyświetlić zawarte w niej informacje.

Opis problemu

DNA składa się z dwóch skręconych łańcuchów cząsteczek zwanych "nukleotydami". Podczas odczytywania informacji z cząsteczki DNA specjalne enzymy są wykorzystywane do rozplątywania cząsteczki DNA w celu odczytania nukleotydów zawartych w pojedynczym łańcuchu. W DNA występują cztery rodzaje nukleotydów, z których każdy jest zwykle reprezentowany przez jedną literę: adenina (A), tymina (T), guanina (G) i cytozyna (C). Nukleotydy te tworzą pary, które łączą dwa łańcuchy, tworząc kompletną cząsteczkę DNA. A zawsze łączy się w pary z T, a G zawsze łączy się w pary z C. Ponieważ możliwe są tylko dwie pary, pozwala to na stworzenie cząsteczki DNA, która reprezentuje ciąg binarny bez martwienia się o to, który łańcuch w cząsteczce zostanie odczytany.

W tym zadaniu otrzymasz sekwencję nukleotydów jednego z łańcuchów cząsteczki DNA, która zawiera dane obliczeniowe. Musisz przetłumaczyć zasady nukleotydowe na cyfry binarne, jak pokazano na rysunku:

adenina (A) lub tymina (T) = 0

guanina (G) lub cytozyna (C) = 1

Po przetłumaczeniu na ciąg binarny musisz skonwertować wartości binarne na znaki ASCII. Każdy znak będzie reprezentowany przez siedem bitów (siedem nukleotydów), a wszystkie znaki będą możliwe do wydrukowania. Przykładem może być poniższa sekwencja DNA:

G A T C A T A	G C A T C A G	C G A G C T A	G C A C G T A	C G A G C G C
1 0 0 1 0 0 0	1 1 0 0 1 0 1	1 1 0 1 1 0 0	1 1 0 1 1 0 0	1 1 0 1 1 1 1
H	**e**	**l**	**l**	**o**

Pełna lista wszystkich amerykańskich znaków ASCII i ich odpowiedników binarnych znajduje się na stronie 4 niniejszego zestawu.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie się składał z jednego wiersza tekstu zawierającego wyłącznie znaki A, C, G i/lub T. Liczba znaków w każdym wierszu będzie podzielna przez siedem.

```
2
GATCATAGCATCAGCGAGCTAGCACGTACGAGCGC
CTTATGGCTCATTCTGGATGTACGAATTACCATGTAGCATTATCTAATG
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić tekst reprezentowany przez ciąg binarny zawarty w podanej sekwencji DNA.

```
Hello
CQ2020!
```

Problem 09: Przeszłość to prolog

Punkty: 35

Autor: Carlos Sepúlveda, Orlando, Floryda, Stany Zjednoczone

Kontekst problemu

Organizatorzy szkoleń zarządzający ćwiczeniami szkoleniowymi dążą do osiągnięcia jak najlepszych wyników dla osób w nim uczestniczących. Uchwycenie podstawowych szczegółów tych szkoleń może często decydować o powodzeniu lub niepowodzeniu przyszłych działań szkoleniowych. Znajomość podstawowych informacji, takich jak częstotliwość i czas trwania poprzednich wydarzeń, może pomóc zmaksymalizować efektywność planowania; znajomość liczby uczestników poprzednich wydarzeń może pomóc przewidzieć typowe problemy logistyczne. W tym właśnie mogą pomóc zespoły inżynierskie opracowujące algorytmy predykcyjne, które pomogą organizatorom podejmować świadome decyzje skutkujące doskonałymi wynikami.

Opis problemu

Stowarzyszenie Młodzieżowej Ligi Code Quest w Orlando organizuje codzienne spotkania, aby pomóc drużynom z całej Florydy przygotować się do przyszłorocznego Code Quest i innych konkursów programistycznych. Prowadzą szczegółowe rejestry wszystkich uczestników każdego wydarzenia w bazie danych, w której przechowywane są następujące informacje:

- Identyfikator źródła danych — niepowtarzalny alfanumeryczny identyfikator rekordu danych
- Uczestnik — nazwa zespołu lub imię i nazwisko osoby biorącej udział w wydarzeniu
- Sesja — dzienna („Day”) albo nocna („Night”); ta informacja wskazuje, w której sesji wydarzenia uczestniczył uczestnik
- Identyfikator wydarzenia — niepowtarzalny alfanumeryczny identyfikator wydarzenia
- Zespół — wartość prawda („true”) lub fałsz („false”); ta informacja wskazuje, czy uczestnikiem był zespół czy osoba indywidualna

Stowarzyszenie chce być w stanie przewidzieć, ilu drużyn można się spodziewać na przyszłych imprezach; aby tego dokonać, musi znać frekwencję na poszczególnych sesjach poprzednich imprez. Twój zespół został zatrudniony do opracowania narzędzia, które będzie pobierać te informacje z bazy danych w postaci eksportu rekordów rozdzielanych przecinkami.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający jedną dodatnią liczbę całkowitą (**N**) reprezentującą liczbę rekordów bazy danych.

- **N** wierszy, z których każdy reprezentuje rekord bazy danych zawierający następujące informacje rozdzielone przecinkami:
    - Identyfikator źródła danych rekordu: ciąg znaków zawierający znaki alfanumeryczne; duże litery
    - Nazwa zespołu lub imię i nazwisko uczestnika: ciąg znaków zawierający litery i spacje
    - Sesja: jedna z wartości „Day” lub „Night”.
    - Identyfikator wydarzenia: ciąg znaków zawierający znaki alfanumeryczne; duże litery
    - Wartość logiczna („true” lub „false”): wskazująca, czy uczestnikiem był zespół czy osoba indywidualna

```
1
4
A1,Code Questers,Day,E1,true
A2,Maintenance,Day,E1,false
A3,LM Peeps,Night,E1,true
A4,Code Questers,Day,E2,true
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego, program musi zwrócić jeden wiersz dla każdego unikalnego identyfikatora zdarzenia wymienionego w rekordach bazy danych w porządku alfanumerycznym. Każdy wiersz powinien zawierać następujące informacje rozdzielone przecinkami:

- Identyfikator wydarzenia, podany w danych wejściowych
- Liczba zespołów, które uczestniczyły w sesji dziennej danego wydarzenia
- Liczba zespołów, które uczestniczyły w sesji nocnej danego wydarzenia

```
E1,1,1
E2,1,0
```

Problem 10: Aktualizacja danych na okoliczność nagłych wypadków

Punkty: 40

Autor: Kelly Reust, Denver, Kolorado, Stany Zjednoczone

Kontekst problemu

Każdego roku pracownicy firmy Lockheed Martin są zachęcani do aktualizacji swoich danych kontaktowych na okoliczność nagłych wypadków. Informacje te są najczęściej wykorzystywane do ostrzegania pracowników o zamknięciu zakładu z powodu trudnych warunków pogodowych lub innych ekstremalnych sytuacji. Dodatkowo, mogą być również wykorzystywane do ostrzegania pracowników o sytuacjach, na które powinni zwracać uwagę podczas podróży, takich jak strajki w transporcie lub komunikaty władz lokalnych.

Nadszedł czas corocznej aktualizacji, a dział kadr chce się upewnić, że posiada dokumentację wprowadzonych zmian. Audyty tego rodzaju są ważne dla zapewnienia poprawności danych. Jeśli jakiekolwiek dane są nieprawidłowe, pomagają osobom prowadzącym dochodzenie w ustaleniu źródła błędów. Jesteś nową osobą w dziale i odbywasz w nim staż. Powierzone zostało Ci zadanie przeprowadzenia tego audytu, ale naprawdę nie chcesz robić wszystkiego ręcznie. Napiszmy program, który wykona tę pracę za Ciebie!

Opis problemu

Dział kadr przekazał Ci listę zeszłorocznych danych kontaktowych na okoliczność nagłych wypadków oraz listę tegorocznych danych. Większość pracowników prawdopodobnie zachowa swoje dane bez zmian, ale niektórzy mogli się przeprowadzić lub otrzymać nowe telefony i musieli zaktualizować swoje dane. Ponadto niektórzy pracownicy odeszli z firmy lub dołączyli do niej w ciągu ostatniego roku i ich rekordy należy odpowiednio usunąć lub utworzyć. Twój program będzie musiał zidentyfikować wszystkie te zmiany i dokładnie je opisać.

Każdy rekord w posiadanych plikach zawiera imię i nazwisko pracownika, jego adres i numer telefonu. Imiona i nazwiska pracowników są unikatowe (na potrzeby tego problemu). Jeśli nazwisko pojawia się w „starej” kartotece, ale nie w „nowej”, oznacza to, że pracownik opuścił Lockheed Martin i jego rekord powinien zostać usunięty. Z kolei nazwisko, które pojawia się tylko w „nowej” kartotece, oznacza nowego pracownika, którego rekord należy utworzyć. Jeśli w obu kartotekach pojawia się to samo nazwisko, konieczne będzie porównanie adresu i numeru telefonu w obu wersjach. Jeśli któraś z tych informacji (lub obie) różnią się, należy zaktualizować rekord o nowe dane. Jeśli wszystkie informacje o pracowniku są takie same w obu plikach, nie wprowadzono żadnych zmian i nie ma potrzeby ich zgłaszania.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite (**O** i **N**) rozdzielone spacjami, reprezentujące odpowiednio liczbę starych i nowych rekordów.
- Wiersze **O** zawierające rekordy danych kontaktów na okoliczność nagłych wypadków z ubiegłego roku. Każdy wiersz zawiera następujące wartości, rozdzielone przecinkami:
  - Imię i nazwisko pracownika, które może zawierać duże i małe litery i/lub spacje.
  - Numer telefonu pracownika, który będzie 10-cyfrową dodatnią liczbą całkowitą.
  - Adres pracownika, który może zawierać duże i małe litery i/lub spacje.
- Wiersze **N** zawierające tegoroczne rekordy danych kontaktów na okoliczność nagłych wypadków. Każdy wiersz zawiera następujące wartości, rozdzielone przecinkami:
  - Imię i nazwisko pracownika, które może zawierać duże i małe litery i/lub spacje.
  - Numer telefonu pracownika, który będzie 10-cyfrową dodatnią liczbą całkowitą.
  - Adres pracownika, który może zawierać duże i małe litery i/lub spacje.

```
1
3 3
John Doe,1234567890,123 Anywhere Street
Jane Doe,9876543210,456 Somewhere Road
Billy Bob Joe,1472583690,789 Nowhere Avenue
Jane Doe,9876543210,456 Somewhere Road
Joe Bob Bill,9638520147,159 Over There Lane
John Doe,1597538462,123 Anywhere Street
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego program musi zwrócić listę tych pracowników, których rekordy zostały utworzone, zaktualizowane lub usunięte. W każdym wierszu należy wymienić jednego pracownika, w porządku alfabetycznym według nazwisk (jak w przykładzie; najpierw imię). Wiersze powinny być sformatowane w następujący sposób:

- Jeśli rekord pracownika został utworzony lub usunięty, wygeneruj nazwisko pracownika, a następnie — odpowiednio — określenie „CREATED” potwierdzające utworzenie lub „DELETED” potwierdzające usunięcie.
- Jeśli dane pracownika zostały zaktualizowane, wygeneruj imię i nazwisko pracownika, po którym musi pojawić się określenie „UPDATED” potwierdzające aktualizację, a następnie określenia konkretnych aktualizacji: „PHONE NUMBER” dla numeru telefonu, „ADDRESS” dla adresu lub „BOTH” w przypadku aktualizacji obu wartości.

```
Billy Bob Joe DELETED
Joe Bob Bill CREATED
John Doe UPDATED PHONE NUMBER
```

Problem 11: Rymowana wyliczanka

Punkty: 45

Autor: Wojciech Kozioł, Mielec, Polska

Kontekst problemu

Każdy z nas pamięta z dzieciństwa jakąś odmianę wyliczanki, często bezsensownej rymowanki używanej w celu wybrania lub wyeliminowania jednej osoby z grupy. W Stanach Zjednoczonych i Wielkiej Brytanii rymowanki te zwykle zaczynają się od słów „eeny, meenie, miney, moe”; przykładowa polska wersja brzmi następująco:

Raz, dwa, trzy, cztery.

Maszerują oficery, ptaszek myślał,

że to żołnierz i

narobił mu na kołnierz.

Po wypowiedzeniu każdego słowa wyliczający wskazuje kolejno na jedną osobę z grupy. Po zakończeniu rymowanki osoba, na którą wskazywano, zostaje wyeliminowana z grupy. Wyliczający zaczyna rymowankę od nowa, wskazując na następną osobę w kolejce do pierwszego słowa, i tak dalej, aż zostanie tylko jedna osoba.

Opis problemu

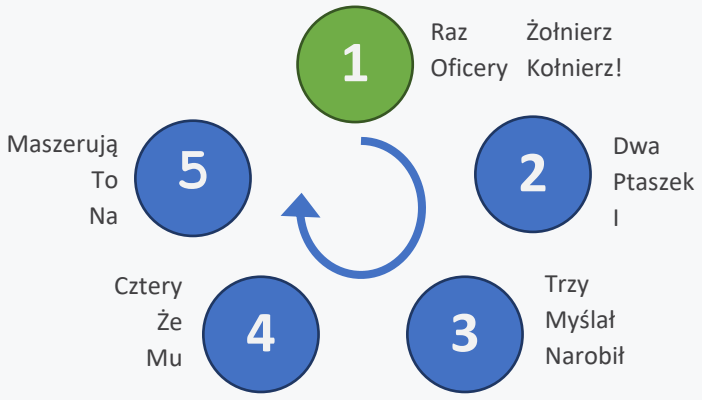
Każdy wie, że prawdziwą sztuką w stosowaniu tych rymów jest wiedzieć, od kogo zacząć, aby nie wyeliminować samego siebie. W przykładowej polskiej wyliczance jest szesnaście oddzielnych słów. Jeśli jesteś w grupie pięciu osób, każda z nich może otrzymać niepowtarzalny numer; sobie nadajesz numer 1. Jeśli dodatkowo zaczniesz rym od siebie…

1. Raz
2. Dwa
3. Trzy
4. Cztery

5. Maszerują
1. Oficery
2. Ptaszek
3. Myślał

4. Że
5. To
1. Żołnierz
2. I

3. Narobił
4. Mu
5. Na
1. Kołnierz



O nie! Wskazujesz na siebie, a więc jesteś wyeliminowany! Nie o to Ci chodziło. Jeśli jednak zaczniesz od kogoś innego, to ta osoba zostanie wyeliminowana. Jeśli zaczniesz od osoby oznaczonej numerem 4, zostanie ona wyeliminowana w pierwszej rundzie. Następnie rozpoczynasz drugą rundę od osoby z numerem 5 (następna osoba w kolejce), eliminując osobę oznaczoną numerem 3. Od osoby oznaczonej numerem 5 rozpoczynasz także trzecią rundę, w której eliminujesz osobę oznaczoną numerem 5. Następnie rozpoczynasz czwartą rundę od siebie i eliminujesz osobę oznaczoną numerem 2. Wygrana jest twoja!

Uwzględniając liczbę osób w grupie oraz liczbę słów w rymowanej wyliczance, napisz program, który określi, od kogo należy zacząć liczenie, aby zagwarantować, że zostaniesz ostatnią osobą.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz z dwiema dodatnimi liczbami całkowitymi rozdzielonymi spacjami:

- **N**: liczba osób w grupie. Każda osoba jest identyfikowana za pomocą unikalnego numeru z zakresu od 1 do **N** włącznie; Ty jesteś numerem 1.
- **K**: liczba słów w rymowanej wyliczance.

```
2
5 16
6 8
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić numer osoby, od której musisz rozpocząć liczenie w pierwszej rundzie, aby zagwarantować, że po zakończeniu wszystkich rund będziesz jedyną niewyeliminowaną osobą.

```
4
5
```

Problem 12: Kod Morse'a

Punkty: 50

Autor: Richard Green, Whiteley, Hampshire, Wielka Brytania

Kontekst problemu

Kod Morse'a, nazwany tak na cześć jego twórcy Samuela Morse'a, to powszechnie znana metoda kodowania tekstu za pomocą serii kropek i kresek. Został on pierwotnie stworzony w 1800 roku do użytku w telegrafie, urządzeniu wykorzystującym impulsy elektryczne do przesyłania wiadomości, ale był również używany do przesyłania sygnałów dźwiękowych, świetlnych i radiowych.

Podczas korzystania z telegrafu operator naciskał przycisk, aby wysłać sygnał elektryczny. Sygnał pozostawał aktywny tak długo, jak długo przycisk był trzymany. Zwolnienie przycisku powodowało przerwanie sygnału. Podczas nadawania komunikatu operator sygnalizował kropkę przytrzymując przycisk przez jedną jednostkę czasu, a kreskę — przytrzymując przycisk przez trzy jednostki czasu. Odstępy między kropkami i kreskami w obrębie znaku byłyby reprezentowane przez zwolnienie przycisku na jedną jednostkę czasu; odstępy między znakami byłyby reprezentowane przez opóźnienie o trzy jednostki, a między słowami — o siedem jednostek.

Opis problemu

Oryginalny kod Morse'a został zaprojektowany z myślą o wydajności; czas potrzebny do przesłania każdej angielskiej litery był odwrotnie proporcjonalny do częstotliwości, z jaką była ona używany. Na przykład najczęściej występująca litera w języku angielskim, czyli E, była reprezentowana przez pojedynczą kropkę. Z kolei Q, jedna z najrzadziej występujących liter, wymagała dwóch kresek, kropki i kolejnej kreski.

Dzięki nowoczesnym komputerom wydajność jest mniej istotna; Ty i twoi przyjaciele bardziej martwicie się o podsłuchiwaczy. Zgodziłeś się na stworzenie własnej wersji kodu Morse'a, która będzie używana podczas wysyłania wiadomości w obie strony. Wystarczy napisać program, który potrafi odczytywać kod Morse'a, a następnie szyfrować i odszyfrowywać wiadomości przy użyciu tego kodu.

Komunikaty generowane kodem Morse'a będą przedstawiane w tekście jako seria kropek, kresek i spacji. Każda kropka i kreska w obrębie litery powinna być rozdzielona pojedynczą spacją. Litery w wyrazie powinny być rozdzielone trzema spacjami. Słowa powinny być rozdzielone siedmioma spacjami.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- 26 wierszy reprezentujących kod Morse'a zastosowany w tym przypadku testowym, wymienione w porządku alfabetycznym. Każdy z tych wierszy będzie się zaczynał wielką literą,

po której następuje spacja, a następnie kombinacja kropek (.), kresek (-) i spacji, reprezentujących konkretną literę zakodowaną kodem Morse'a.
- Wiersz zawierający wiadomość napisaną wielkimi literami i spacjami, która musi być zakodowana przy użyciu kodu Morse'a.
- Wiersz zawierający wiadomość napisaną kodem Morse'a, którą należy rozszyfrować.

```
1
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 - - . .
HELLO WORLD
. .       . - . .   - - -   . . . -   .       - . - .   - - -   - . .   .
- - . -   . . -   .   . . .   -
```

(Pamiętaj, że powyższa zaszyfrowana wiadomość stanowi tylko jeden wiersz tekstu; jest ona zbyt długa, aby zmieścić się na tej stronie. Dokładniejszą reprezentację można znaleźć w dostarczonym przykładowym pliku wejściowym).

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić:

- Wiersz zawierający teraz zaszyfrowaną wiadomość zapisaną kodem Morse'a
- Wiersz zawierający teraz odszyfrowaną wiadomość w języku angielskim

```
. . . .   .   . - . .   . - . .   - - -       . - -   - - -   . - .   .
- . .   - . .
I LOVE CODE QUEST
```

(Pamiętaj, że powyższa zaszyfrowana wiadomość stanowi tylko jeden wiersz tekstu; jest ona zbyt długa, aby zmieścić się na tej stronie. Dokładniejszą reprezentację można znaleźć w dostarczonym przykładowym pliku wyjściowym).

Problem 13: Co robić?

Punkty: 55

Autor: Gary Hoffmann, Denver, Kolorado, Stany Zjednoczone

Kontekst problemu

Komputery mają ograniczoną liczbę dostępnych zasobów — mają tylko tyle pamięci, miejsca na dysku i mocy obliczeniowej, w ile zostały wyposażone. System operacyjny komputera jest odpowiedzialny za zarządzanie konkurencyjnymi żądaniami dotyczącymi tych zasobów i często musi podejmować decyzje o tym, które oprogramowanie ma pierwszeństwo w korzystaniu z tych zasobów.

Współpracujesz z firmą Lockheed Martin Space nad stworzeniem satelity dla amerykańskiej Narodowej Agencji Oceanów i Atmosfery (NOAA), który ma służyć do przeprowadzania różnych eksperymentów wysoko ponad ziemską atmosferą. Twój zespół został poproszony o zaprojektowanie harmonogramu priorytetów, który będzie określał, co satelita powinien robić w danym momencie.

Opis problemu

Komputer satelity dzieli czas na cykle obliczeniowe, zaczynając od cyklu 1. Wykonanie każdego zadania zajmuje określoną liczbę cykli, a po zakończeniu zadania nie powinno być ono wykonywane ponownie. Zadania mają przypisany priorytet. Na początku każdego cyklu obliczeniowego algorytm szeregowania powinien nakazać satelicie pracę nad zadaniem o najwyższym priorytecie (wyższe priorytety mają większe wartości) spośród tych, których jeszcze nie wykonał, ewentualnie wywołując przełączanie się między zadaniami. Jeśli dwa zadania mają ten sam priorytet, uruchom to, które pojawia się jako pierwsze na liście zadań. Zadania mają także ograniczenia czasowe, reprezentowane przez numer cyklu. Zadanie nie może zostać przepracowane, jeśli bieżący numer cyklu nie jest większy lub równy ograniczeniu czasowemu zadania.

Twój algorytm planowania będzie musiał dostarczyć raport o tym, co satelita będzie robił w każdym cyklu, przez określoną liczbę cykli. Jeżeli satelita nie jest w stanie wykonać żadnego z niekompletnych zadań ze względu na ograniczenia czasowe, powinien wykonać zadanie oczekiwania „Wait” przez jeden cykl i ponownie ocenić sytuację na początku następnego cyklu. Wykonanie wszystkich zadań w wyznaczonym czasie może okazać się niemożliwe.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite, rozdzielone przecinkiem, reprezentujące:
  - **C** — liczba cykli, które algorytm szeregowania powinien zaplanować
  - **T** — liczba zadań, które satelita ma wykonać

- **T** wierszy, w których znajdują się informacje o zadaniach do wykonania przez satelitę. Każdy wiersz będzie zawierał następujące informacje, rozdzielone przecinkami:
    - Ciąg znaków stanowiący unikalny opis zadania, który może zawierać litery i spacje
    - Dodatnią liczbę całkowitą oznaczającą poziom priorytetu dla zadania
    - Dodatnią liczbę całkowitą oznaczającą ograniczenie czasowe zadania, czyli najwcześniejszy numer cyklu, w którym zadanie może zostać uruchomione
    - Dodatnią liczbę całkowitą oznaczającą liczbę cykli potrzebnych do wykonania tego zadania

```
1
10,7
Execute Experiment A,11,4,2
Reorient to Target A,4,1,2
Capture Imagery,5,1,3
Station Keeping,10,1,2
Transmit Results,20,8,1
Reorient to Target B,12,4,1
Execute Experiment B,2,2,3
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić **C** wierszy zawierających informacje o wykonywaniu zadań przez satelitę w każdym cyklu obliczeniowym. Każdy wiersz powinien zawierać numer cyklu (zaczynając od 1), przecinek oraz opis zadania, które ma być wykonane w danym cyklu. Jeśli nie ma żadnego zadania do wykonania, zamiast opisu zadania zwrócony musi zostać komunikat oczekiwania „Wait”.

```
1,Station Keeping
2,Station Keeping
3,Capture Imagery
4,Reorient to Target B
5,Execute Experiment A
6,Execute Experiment A
7,Capture Imagery
8,Transmit Results
9,Capture Imagery
10,Reorient to Target A
```

Problem 14: Możesz na mnie polegać

Punkty: 60

Autor: Matt Marzin, King of Prussia, Pensylwania, Stany Zjednoczone

Kontekst problemu

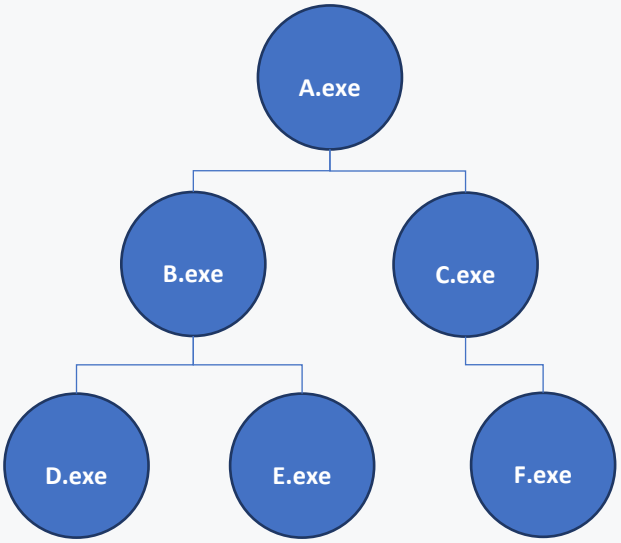
DevSecOps — akronim utworzony od słów „Development”, „Security” i „Operations” — jest definicją tego, aby wszyscy ponosili odpowiedzialni za bezpieczeństwo aplikacji od początku jej tworzenia, do wydania i później. Jako inżynier DevSecOps w Lockheed Martin odpowiadasz za utrzymanie oprogramowania, na którym twój zespół opiera się przy tworzeniu aplikacji. Infrastruktura ta obejmuje narzędzia, które stale sprawdzają aplikację pod kątem błędów, luk w zabezpieczeniach i innych problemów, które należy rozwiązać przed dostarczeniem aplikacji do klienta.

Niestety, programy tworzące tę infrastrukturę są ze sobą ściśle powiązane i w przypadku awarii jednego z nich, należy ponownie uruchomić wszystkie procesy wspierające ten proces. Co więcej, zdarza się to dość regularnie, ponieważ twoi współpracownicy stale rozwijają aplikację i dodają do niej nowe funkcje. Twoim zadaniem jest zaimplementowanie programu, który monitoruje uruchomione procesy w poszukiwaniu awarii, a następnie, gdy taka wystąpi, uruchamia ponownie ten program i wszystkie jego zależności.

Opis problemu

Infrastruktura programistyczna składa się z wielu powiązanych ze sobą procesów. Gdy program przestaje odpowiadać i musi zostać ponownie uruchomiony, najpierw należy ponownie uruchomić wszystkie zależne od niego procesy. Jeśli te procesy mają własne zależności, również należy je zrestartować itd.

Na przykład na poniższym schemacie system obsługuje w sumie pięć programów. Program B.exe zależy od D.exe i E.exe; w przypadku awarii B.exe konieczne będzie ponowne uruchomienie D.exe i E.exe, zanim program B.exe zostanie ponownie uruchomiony.



Jeśli program A.exe przestałby odpowiadać, konieczne byłoby ponowne uruchomienie całej infrastruktury, ponieważ wszystkie inne programy znajdują się poniżej niego w drzewie zależności. Ponieważ D, E i F są zależnościami najniższego poziomu, należy je ponownie uruchomić w pierwszej kolejności; następnie można ponownie uruchomić B i C, a na końcu A. Jeśli na tym samym poziomie zależności istnieje wiele programów (tak jak w przypadku D, E i F), programy należy uruchamiać ponownie w kolejności alfabetycznej według nazwy.

Twój program będzie musiał wczytać listę zależności od programu oraz listę zdarzeń awarii, które muszą zostać obsłużone. Dla każdego zdarzenia należy podać programy, które muszą zostać ponownie uruchomione, oraz kolejność ich uruchamiania. Programy mogą mieć więcej niż jedną zależność, jak pokazano powyżej, i więcej niż jeden program może zależeć konkretnego programu. Nie będzie zależności cyklicznych, tzn. żaden program nie będzie zależny od samego siebie, niezależnie od tego, jak pośrednia byłaby taka zależność.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał następujące elementy:

- Wiersz zawierający dwie dodatnie liczby całkowite, rozdzielone spacją, reprezentujące:
  - **D** — liczba zależności
  - **E** — liczba zdarzeń awarii
- **D** wierszy zawierających informacje o zależnościach programu. Każdy wiersz składa się z dwóch ciągów znaków rozdzielonych spacją. Program o nazwie podanej pierwszym ciągiem zależy od programu o nazwie podanej drugim ciągiem. Wszystkie nazwy programów będą zawierać małe litery i będą kończyć się rozszerzeniem „.exe”.
- **E** wierszy zawierających informacje o zdarzeniu awarii. Każdy wiersz będzie zawierał nazwę programu, wymienionego wcześniej na liście zależności, który uległ awarii i musi zostać ponownie uruchomiony.

```
1
5 2
a.exe b.exe
a.exe c.exe
b.exe d.exe
b.exe e.exe
c.exe f.exe
b.exe
a.exe
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego program musi zwrócić polecenia, które mają zostać wykonane dla każdego zdarzenia awarii — po jednym w wierszu i w kolejności, w której mają zostać wykonane. Po pierwsze, polecenia ponownego uruchomienia składają się z wyrażenia „restart”, po którym następuje spacja i nazwa programu, który ma zostać ponownie uruchomiony. Po tych poleceniach powinno nastąpić pojedyncze polecenie „exit”, aby zakończyć proces odzyskiwania.

```
restart d.exe
restart e.exe
restart b.exe
exit
restart d.exe
restart e.exe
restart f.exe
restart b.exe
restart c.exe
restart a.exe
exit
```

Problem 15: Powrót łamacza kodów

Punkty: 65

Autor: Brett Reynolds, Annapolis Junction, Maryland, Stany Zjednoczone

Kontekst problemu

W 2021 roku przedstawiliśmy problem, w którym amerykańska Agencja Bezpieczeństwa Narodowego poprosiła firmę Lockheed Martin o pomoc w opracowaniu bazy danych analizy częstotliwościowej do łamania szyfrów substytucyjnych. Agencja była pod tak dużym wrażeniem twojej pracy, że poprosiła Cię o kontynuowanie działań!

Dla przypomnienia, szyfry substytucyjne szyfrują wiadomości przez zastąpienie liter w oryginalnej wiadomości innymi literami, co ma na celu ukrycie treści wiadomości. Prosty szyfr substytucyjny polega na zastąpieniu każdej litery następną w alfabecie, na przykład wyraz „cat” miałby postać „dbu”. Są one jednak ogólnie uważane za szyfry słabe, ponieważ w każdym języku pewne litery występują częściej niż inne. Samogłoski są szczególnie popularne ze względu na ich znaczenie dla języka. Jeśli wiesz, że oryginalna wiadomość była w języku angielskim, możesz bezpiecznie założyć, że najczęściej występującą literą będzie prawdopodobnie „E”. Ta metoda łamania szyfrów jest nazywana „analizą częstotliwości”.

W bardziej zaawansowanych szyfrach jest to uwzględniane i dokłada się więcej starań, aby ukryć tekst, jednak bardziej zaawansowana analiza częstotliwości pozwala przezwyciężyć ten problem. Zamiast przyglądać się pojedynczym literom, należy przyjrzeć się parom lub tercetom liter (tzw. digrafom i trigrafom), co może zapewnić jeszcze lepszy wgląd w ich treść i pomóc w identyfikacji poszczególnych liter.

Opis problemu

Jak wspomniano powyżej, amerykańska Agencja Bezpieczeństwa Narodowego zwróciła się do firmy Lockheed Martin o rozszerzenie bazy danych analizy częstotliwości utworzonej w 2021 r. o analizę digrafów i trigrafów. Digraf to para liter, które występują obok siebie w tym samym wyrazie; trigraf to grupa trzech liter w tym samym wyrazie. Podobnie jak w przypadku liter, pewne digrafy i trigrafy występują znacznie częściej w niektórych językach, a niektóre kombinacje nie pojawiają się w ogóle. Na przykład zestaw liter „qxz” nigdy nie występuje obok siebie w żadnym angielskim słowie, ale trigraf „the” jest najczęściej spotykany — głównie dlatego, że „the” jest najczęściej występującym słowem w języku angielskim.

Podczas analizy należy pamiętać, że digrafy i trigrafy nie są identyfikowane w obrębie więcej niż jednego wyrazu, a przed rozpoczęciem analizy należy usunąć wszystkie znaki interpunkcyjne i cyfry. Na przykład wyrażenie „code quest” zawiera następujące digrafy: „co”, „od”, „de”, „qu”, „ue”, „es” oraz „st”. Nie zawiera jednak digrafu „eq” — mimo że litery te występują obok siebie, są rozdzielone spacją. To samo dotyczy trigrafów; wyrażenie to obejmuje następujące trigrafy: „ode” i „que”, ale nie „deq” i „equ”.

Podobnie jak w zadaniu z 2021 r., twój zespół będzie analizował dużą ilość tekstu, aby określić względną częstotliwość występowania w nim digrafów i trigrafów. Względną częstotliwość występowania danego terminu można obliczyć za pomocą tego wzoru:

$$Częstotliwość\ występowania\ = \left(\frac{ilość\ wystąpień\ dane\ wyrazu}{ilość\ wszystkich\ wystąpień\ danego\ typu}\right) \times\ 100\%$$

Jak już wspomniano, podczas wykonywania tego obliczenia należy oddzielić liczbę wszystkich digrafów od liczby wszystkich trigrafów.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dodatnią liczbę całkowitą (**X**) oznaczającą liczbę wierszy tekstu, który ma zostać dostarczony.
- **X** wierszy zawierających tekst do analizy. Każdy wiersz może zawierać do 2000 znaków i może zawierać dowolny znak ASCII.

```
1
3
The quick red fox jumps over the lazy brown dog.
The above sentence contains every letter in the English language.
Don’t forget to ignore punctuation and numbers; they’re not relevant!
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wyniki analizy, najpierw generując jeden wiersz dla każdego digrafu zidentyfikowanego w tekście, a następnie jeden wiersz dla każdego zidentyfikowanego trigrafu. Każda lista powinna być ułożona w porządku alfabetycznym. Każdy wiersz powinien zawierać następujące informacje:

- Digraf lub trigraf, zapisany wielkimi literami
- Dwukropek (:)
- Spację
- Względną częstotliwość występowania danego terminu w tekście przedstawionym w przypadku testowym, w stosunku do innych terminów tego samego typu, zaokrągloną do trzech miejsc po przecinku i zawierającą wszelkie końcowe zera.
- Znak procentu (%)

Przykładowe dane wyjściowe są zbyt duże, aby można je było wygenerować w całości. Zamiast tego przedstawiono niewielki wybór wierszy z przykładowych danych wyjściowych w celu zademonstrowania formatu. Aby uzyskać pełny przykładowy wynik, należy pobrać plik ze strony internetowej konkursu.

```
AB: 0.840%
AG: 0.840%
```

[...dodatkowe wiersze dla AI poprzez WN...]

```
YR: 0.840%
ZY: 0.840%
ABO: 1.124%
AGE: 1.124%
```

[...dodatkowe wiersze dla AIN poprzez VAN...]

```
VER: 2.247%
YRE: 1.124%
```

Problem 16: Serwer synaptyczny

Punkty: 70

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

Kontekst problemu

Armia Stanów Zjednoczonych zleciła firmie Lockheed Martin budowę nowego samojezdnego pojazdu do transportu żołnierzy. Twój zespół został wyznaczony do zbudowania systemu identyfikacji znaków drogowych dla transportu. Twój system ma za zadanie wykonać zdjęcie znaku ulicznego w niskiej rozdzielczości i wyprowadzić ciąg znaków wskazujący typ znaku na zdjęciu; do wyboru są:

- STOP_SIGN
- YIELD
- LANE_ENDS
- SPEED_LIMIT
- CROSSWALK

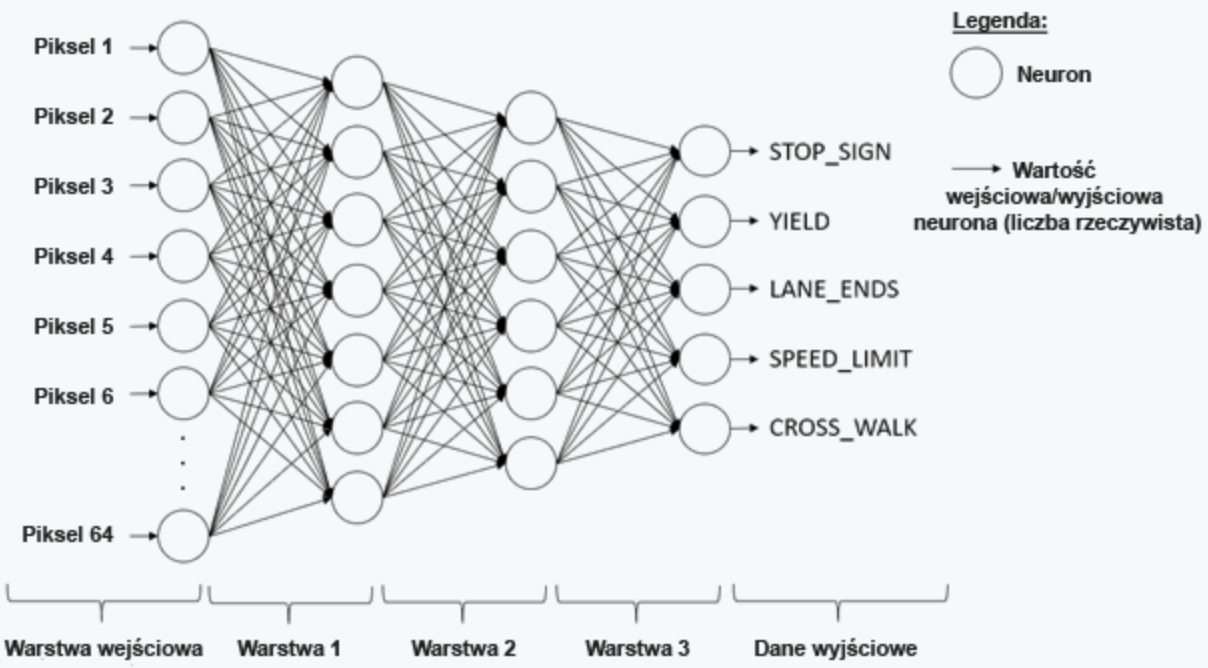
Jest to trudniejsze niż się wydaje, ponieważ żadne dwa zdjęcia nie będą takie same. Dokładną identyfikację może uniemożliwić położenie środka transportu, pora dnia, zła pogoda, a nawet graffiti na znaku. Twój algorytm musi jednak być w stanie przezwyciężyć te problemy, ponieważ błędna decyzja może mieć katastrofalne skutki!

Napisanie tradycyjnej funkcji, która rozwiązałaby ten problem, byłoby prawie niemożliwe, ale wiadomo, że zespół ludzi mógłby z łatwością zidentyfikować znaki. Stwarza to idealne warunki dla klasy algorytmów określanych mianem nadzorowanego uczenia maszynowego, w których komputer „uczy się” nieznanej funkcji, wykorzystując dużą liczbę znanych przykładowych danych.

Opis problemu

Inżynier prowadzący proponuje, aby do rozwiązania tego problemu użyć sieci neuronowej. Sieć neuronowa to algorytm uczenia maszynowego luźno oparty na strukturze mózgu. Sieć neuronowa składa się z dwóch głównych elementów: neuronów i warstw. Neurony są podstawową jednostką obliczeniową w sieci neuronowej i są pogrupowane w szeregu wielu warstw.

Każdy neuron w sieci neuronowej zawiera listę wag ($\vec{w}$) i jedną wartość nastawczą (b). Dzięki tym liczbom neurony są konfigurowalne, co umożliwia komputerowi lub inżynierowi dostosowanie tych wartości w miarę uczenia się przez komputer na podstawie istniejących danych. Każdy neuron jest odpowiedzialny za wygenerowanie jednej „liczby rzeczywistej” (dowolnej liczby dodatniej, ujemnej lub zerowej). Dane wyjściowe wszystkich neuronów w warstwie są następnie gromadzone na jednej liście i przesyłane jako dane wejściowe do każdego neuronu w następnej warstwie. Ilustracja na następnej stronie przedstawia układ sieci neuronowej oraz sposób przekazywania danych wejściowych i wyjściowych między warstwami.



Każdy neuron oblicza swoją wartość wyjściową za pomocą następującego wzoru.

$$y = \max\left(0, b + \sum_{i=0}^{N} x_i w_i\right)$$

W tym wzorze:

- y jest wartością wyjściową neuronu
- $\max(\quad)$ jest funkcją, która zwraca większą z dwóch podanych jej liczb
- b jest wartością nastawczą neuronu (inna dla każdego neuronu)
- Σ jest operatorem sumowania; oznacza, że część równania po nim musi zostać powtórzona dla każdej wartości i od 0 do N włącznie, a następnie wszystkie te wartości muszą zostać dodane do siebie, aby utworzyć jedną liczbę
- N jest liczbą wejść dla neuronu; jest ona równa liczbie neuronów w poprzedniej warstwie
- $\vec{x}$ jest listą (zawierającą $x_0, x_1, \ldots, x_N$) wartości wejściowych dla neuronu (dane wyjściowe ze wszystkich neuronów w poprzedniej warstwie, w kolejności)
- $\vec{w}$ jest listą (zawierającą $w_0, w_1, \ldots, w_N$) wag neuronów dla każdego wejścia (różne dla każdego neuronu)

Wzór ten można również zapisać jako:

$$y = \max(0, b + x_0 w_0 + x_1 w_1 + \cdots + x_N w_N)$$

Wszystkie neurony w tej samej warstwie mają ten sam zestaw danych wejściowych, ale ich unikalne wartości wag i nastawcze sprawiają, że każdy neuron generuje inne dane wyjściowe.

W przypadku sieci neuronowej do transportu oddziałów każdy neuron w pierwszej warstwie otrzyma na wejściu listę 64 liczb z przedziału od 0,0 do 1,0 włącznie; każda liczba na liście reprezentuje wartość

piksela w skali szarości na obrazie znaku ulicznego o wymiarach 8 na 8 pikseli. Neurony te będą wytwarzać sygnały wyjściowe, które zostaną przekazane do następnej warstwy, a ta z kolei wytworzy sygnały wyjściowe, które zostaną przekazane do kolejnej warstwy itd. Ostatnia warstwa będzie zawierała tylko pięć neuronów, reprezentujących pięć typów znaków ulicznych, które należy zidentyfikować, w następujący sposób:

1. STOP_SIGN
2. YIELD
3. LANE_ENDS
4. SPEED_LIMIT
5. CROSSWALK

Dane wyjściowe generowane przez każdy z tych pięciu neuronów określają, na ile sieć jest przekonana, że odpowiadający mu znak jest poprawną odpowiedzią. Twój system musi określić, która wartość wyjściowa jest największa, i zwrócić ten ciąg jako ostateczną odpowiedź.

Na szczęście twój główny inżynier był w stanie znaleźć dane z kilku wstępnie wytrenowanych sieci neuronowych. Twój zespół musi przetestować wszystkie te sieci, aby sprawdzić, jak dobrze sobie radzą z zadaniem.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dodatnią liczbę całkowitą większą lub równą 3 (**L**) oznaczającą liczbę warstw w sieci neuronowej.
- Wiersz zawierający **L** dodatnich liczb całkowitych, rozdzielonych spacjami, reprezentujących liczbę neuronów w każdej warstwie sieci. Ostatnią liczbą całkowitą na tej liście, reprezentującą warstwę wyjściową, będzie zawsze 5. Inne liczby całkowite mogą mieć zakres od 6 do 128 włącznie.
- **L** wierszy zawierających dziesiętne wartości nastawcze i wagi każdego neuronu w warstwie. Wartości będą rozdzielone spacjami. Dla każdego neuronu w warstwie zostanie podana jedna wartość wagi dla każdego wejścia, a następnie wartość nastawcza neuronu. Na przykład warstwa z pięcioma neuronami, która otrzymuje sześć danych wejściowych, wyświetli wartości w następujący sposób:

  $$w_{0,0}\ w_{0,1}\ w_{0,2}\ w_{0,3}\ w_{0,4}\ w_{0,5}\ b_0\ w_{1,0}\ w_{1,1} w_{1,2}\ w_{1,3}\ w_{1,4}\ w_{1,5}\ b_1 \ldots\ w_{4,4}\ w_{4,5}\ b_4$$

- Wiersz zawierający 64 liczby dziesiętne z przedziału od 0,0 do 1,0 włącznie, rozdzielone spacjami, reprezentujące wartości pikseli, które mają być podane jako dane wejściowe do wszystkich neuronów w pierwszej warstwie.

Przykładowe dane wejściowe dla tego zadania są zbyt duże, aby można je było opublikować w pakiecie zadań. Przykładowy zestaw danych wejściowych można pobrać ze strony internetowej konkursu.

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wiersz zawierający nazwę znaku ulicznego o najwyższym poziomie ufności obliczonym przez sieć neuronową. Znaki uliczne powinny zostać wygenerowane dokładnie tak, jak przedstawiono w powyższym opisie.

```
SPEED_LIMIT
YIELD
```

Problem 17: Weź udział w głosowaniu

Punkty: 75

Autor: Brett Reynolds, Annapolis Junction, Maryland, Stany Zjednoczone

Kontekst problemu

Głosowanie jest niezwykle istotnym elementem dobrze działającej demokracji. Głosując w wyborach, dajesz wyraz temu, jaki sposób rządzenia preferujesz. Nawet jeśli wybrani przez Ciebie kandydaci nie wygrają wyborów, to i tak wysyłasz do polityków każdej partii komunikat o zasadach, które popierasz. Każdy, kto ma prawo do głosowania, powinien z niego korzystać przy każdej okazji, gdyż pomaga to zapewnić wszystkim lepszą przyszłość.

Niestety, demokracje i wybory nie są doskonałe. Często okazuje się, że nawet jeśli w wyborach startuje wielu kandydatów, tylko dwóch lub trzech z nich ma realne szanse na zwycięstwo. Jeśli wyborca nie popiera w pełni żadnego z tych kandydatów, może czuć się zmuszony do wyboru „mniejszego zła” lub uznać, że nie warto w ogóle głosować. Aby zwalczyć te odczucia, niektóre jurysdykcje przechodzą na nowy system przeprowadzania wyborów, zwany ordynacją preferencyjną. W tym systemie wyborca, który głosuje na przegranego kandydata, nadal ma wpływ na to, który z pozostałych kandydatów zostanie wybrany.

Opis problemu

Firma Lockheed Martin współpracuje z rządem stanowym nad wdrożeniem nowego elektronicznego systemu głosowania, wykorzystującego ordynację preferencyjną. Twój zespół został poproszony o opracowanie algorytmu, który będzie liczył ostateczne głosy. Istnieje kilka różnych wersji ordynacji preferencyjnej, jednak ta, którą stosuje stan, działa w sposób opisany poniżej.

Wyborcy umieszczają kandydatów na listach w kolejności preferencji; kandydat, którego najbardziej popierają, jest umieszczany na pierwszej pozycji, następnie drugi wybór wyborcy i tak dalej. Po oddaniu wszystkich kart do głosowania, wyniki są obliczane na podstawie pierwszego wyboru wymienionego na każdej karcie. Jeśli któryś z kandydatów uzyska wyraźną większość głosów — połowę całkowitej liczby kart do głosowania plus jeden — wygrywa wybory. W przeciwnym razie kandydat, który uzyskał najmniejszą liczbę głosów, zostaje wyeliminowany i rozpoczyna się drugie liczenie głosów; w przypadku remisu na ostatnim miejscu wszyscy kandydaci z najmniejszą liczbą głosów zostają wyeliminowani łącznie. Z każdej karty do głosowania, na której wyeliminowany kandydat został wybrany jako pierwszy, uwzględniany jest drugi wybór. W przeciwnym razie jeden lub więcej kandydatów zostaje wyeliminowanych i przeprowadza się trzecie liczenie głosów, w którym w razie potrzeby wykorzystuje się kandydatów drugiego lub trzeciego wyboru. Trwa to do momentu wyłonienia wyraźnego zwycięzcy.

Na przykład, rozważmy wybory, w których startuje czterech kandydatów. Po pierwszym liczeniu wyniki są takie, jak pokazano poniżej:

Kandydat	Głosy pierwszego wyboru	Odsetek
Kandydat A	10	40%
Kandydat B	7	28%
Kandydat C	5	20%
Kandydat D	3	12%

Żaden z kandydatów nie uzyskał więcej niż 50% głosów, więc nikt nie wygrał. Jeszcze. Kandydat D otrzymał najmniejszą liczbę głosów, w związku z czym zostaje wyeliminowany. Spośród 3 osób, które głosowały na kandydata D, 2 zaznaczyły kandydata B jako swój drugi wybór, a 1 zaznaczył kandydata C. Po drugim głosowaniu wynik jest następujący:

Kandydat	1. wybór	2. wybór	Razem	Odsetek
Kandydat A	10	0	10	40%
Kandydat B	7	2	9	36%
Kandydat C	5	1	6	24%
~~**Kandydat D**~~	~~3~~	~~0~~	~~0~~	~~Wyeliminowani~~

Nadal nie ma zwycięzcy, więc przeprowadzamy jeszcze jedną eliminację, usuwając kandydata C z konkursu. Każdy, kto głosował na kandydata C lub D jako pierwszy (lub drugi) wybór, musi teraz sięgnąć po swój drugi (lub trzeci) wybór. W tym scenariuszu:

- Osoba, która wybrała kandydata D jako swój pierwszy wybór i kandydata C jako drugi wybór, wybrała kandydata A jako swój trzeci wybór, który został zaliczony.
- 2 osoby, które głosowały na kandydata C jako kandydata pierwszego wyboru, wybrały kandydata D jako kandydata drugiego wyboru. Ponieważ kandydat D zostaje wyeliminowany, sięgają po trzeci wybór: kandydata A.
- Pozostałe 3 osoby, które najpierw głosowały na kandydata C, wybrały kandydata B jako kandydata drugiego wyboru.

Kandydat	1. wybór	2. wybór	3. wybór	Razem	Odsetek
Kandydat A	10	0	3	13	52%
Kandydat B	7	5	0	12	48%
~~**Kandydat C**~~	~~5~~	~~1~~	~~0~~	~~0~~	~~Wyeliminowani~~
~~**Kandydat D**~~	~~3~~	~~2~~	~~0~~	~~0~~	~~Wyeliminowani~~

Po podliczeniu głosów kandydat A ma obecnie zdecydowaną większość — 52% głosów. Zostaje ogłoszony zwycięzcą.

Twój algorytm musi wczytać pewną liczbę kart do głosowania oddanych w wyborach i określić zwycięzcę.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite rozdzielone spacjami:
  - **B**: liczba kart do głosowania oddanych w wyborach
  - **C**: liczba kandydatów startujących w wyborach
- **B**: wiersze reprezentujące każdą z oddanych kart do głosowania Każdy wiersz będzie zawierał **C** znaków, z których każdy będzie reprezentował kandydata, wymienionego w kolejności preferencji.

```
1
25 4
ABCD
ABDC
ACBD
ACDB
ADBC
ADCB
ABCD
ADCB
ACDB
ABDC
BACD
BADC
BCAD
BCDA
BDAC
BDCA
BACD
CDAB
CDAB
CBAD
CBDA
CBAD
DBAC
DBCA
DCAB
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi ogłosić zwycięzcę wyborów, generując zdanie potwierdzające wynik: „Candidate **X** won with **P%** of the vote after **T** tallies”, gdzie:

- **X** to wielka litera użyta do identyfikacji kandydata w obrębie danych wejściowych.
- **P%** to odsetek głosów końcowych, zaokrąglony do jednego miejsca po przecinku i zawierający wszelkie końcowe zera.
- **T** to liczba zliczeń, które należało przeprowadzić, aby wyłonić zwycięzcę.

```
Candidate A won with 52.0% of the vote after 3 tallies.
```

Problem 18: To wyższa szkoła jazdy

Punkty: 75

Autor: Ben Fenton, Ampthill, Reddings Wood, Wielka Brytania

Kontekst problemu

Lockheed Martin i Brytyjska Agencja Kosmiczna współpracują nad opracowaniem nowego systemu rozmieszczania satelitów CubeSat — małych satelitów często wykorzystywanych do badań naukowych. Mały orbitalny pojazd manewrujący SL-OMV (ang. Small Launch Orbital Manoeuvering Vehicle) może pomieścić do 6 satelitów CubeSat jednocześnie i wystrzeliwać je w optymalnym czasie i położeniu dla ich misji. Ułatwia to znacznie wystrzeliwanie satelitów CubeSat, zwłaszcza że są one zwykle zbyt małe, by posiadać jakikolwiek własny system napędowy. Wystrzelenie satelitów nadal nie jest łatwym zadaniem.

Opis problemu

Twój zespół współpracuje z Brytyjską Agencją Kosmiczną nad stworzeniem programu, który określi, ile paliwa należy załadować do SL-OMV na każdą misję. Systemy nawigacyjne na pokładzie pojazdu pozwalają mu poruszać się niczym po dwuwymiarowej siatce. W każdej misji pojazd będzie miał do sześciu satelitów CubeSat, z których każdy musi zostać wystrzelony w określone miejsce na siatce, podczas gdy pojazd będzie zupełnie zatrzymany relatywnie względem siatki. Dzięki temu każdy satelita CubeSat zostanie umieszczony na orbicie odpowiedniej dla jego misji. Pojazd musi jednak wystrzelić każdego satelitę CubeSat w określonym czasie, aby zapewnić, że orbita satelity CubeSat nie będzie kolidować z orbitą innego satelity. Zderzenia w przestrzeni kosmicznej są bardzo poważnym problemem i mogą powodować katastrofalne uszkodzenia nie tylko zderzających się obiektów, ale także innych satelitów na pobliskich orbitach.

SL-OMV ma cztery silniki odrzutowe, z których każdy umożliwia mu przyspieszanie wzdłuż osi X lub Y. Każdy silnik odrzutowy zapewnia podczas odpalania stałe przyspieszenie 1 m/s^2. Należy pamiętać, że pojazd porusza się w przestrzeni kosmicznej, czyli bez oporu powietrza lub innego tarcia, które mogłoby go spowolnić; po wyłączeniu silników odrzutowych pojazd będzie się nadal poruszał ze stałą prędkością. Oznacza to, że musisz odpalić przeciwny silnik odrzutowy (silniki odrzutowe), aby spowolnić pojazd lub spowodować zmianę kierunku jego ruchu.

Załóżmy, na przykład, że SL-OMV musi wystrzelić satelitę CubeSat na współrzędne (3,3) w ciągu 5 sekund. Startuje na współrzędnych (0,0), poruszając się z prędkością 0 m/s (pamiętaj, że wszystkie podane tu prędkości są relatywne względem siatki; w rzeczywistości prędkości orbitalne są mierzone w *kilometrach* na sekundę). Dotarcie do współrzędnych startu jest łatwe; krótkie impulsy w silnikach odrzutowych dodatnich wektorów X i Y powodują, że pojazd zaczyna poruszać się we właściwym kierunku; po zbliżeniu się do współrzędnych startu, impulsy w ujemnych silnikach odrzutowych o równym czasie trwania powodują zatrzymanie pojazdu nad celem. Jednak krótkie, 0,1-sekundowe odpalenie każdego z silników odrzutowych nie spowodowałoby zbyt szybkiego przemieszczenia pojazdu; nie ma możliwości, aby zdążył on wystrzelić satelitę CubeSat bez zakłócania pracy innych satelitów.

Aby wystartować na czas, musimy nieco przyspieszyć ruch pojazdu. CubeSat musi zostać wystrzelony po 5 sekundach. Gdy już rozpędzimy rakietę, będziemy musieli poświęcić tyle samo czasu i paliwa, aby ją spowolnić. Oznacza to, że możemy skupić się na pierwszej połowie podróży, czyli na przyspieszaniu. Po ustaleniu, ile paliwa potrzeba do osiągnięcia połowy drogi, możemy po prostu podwoić nasze obliczenia, aby uzyskać całkowitą ilość paliwa potrzebną do przyspieszenia *oraz* wyhamowania.

Rozpocznijmy nasze obliczenia. Jeśli zależy nam tylko na dotarciu do połowy drogi, musimy znaleźć się na współrzędnych (1,5,1,5) po upływie 2,5 sekundy. Część tego czasu (t_1 sekund) spędzimy na odpalaniu silników odrzutowych; gdy osiągniemy wymaganą prędkość, możemy zgasić silniki i przez pozostały czas (t_2 sekund) po prostu poruszać się ruchem bezwładnym. Ponieważ cała nasza podróż zajmie 2,5 sekundy: $t_2 = 2.5 - t_1$.

Odległość przebytą przez poruszający się obiekt można obliczyć za pomocą tego wzoru:

$$x = \frac{1}{2}at^2 + ut + x_0$$

W tym przypadku x reprezentuje odległość, a reprezentuje przyspieszenie, t reprezentuje czas, a u reprezentuje prędkość początkową obiektu (zanim zaczął przyspieszać). x_0 oznacza pozycję początkową obiektu. Gdy się rozpędzamy, a jest równe 1 m/s^2, a u jest równe 0. Gdy przestaniemy przyspieszać, a przybiera wartość 0, a v staje się prędkością, z jaką się poruszamy (zauważ, że zmieniliśmy v, ponieważ teraz jest to nasza prędkość *końcowa*). Oznacza to, że będziemy musieli użyć dwóch równań, aby dokładnie przedstawić przebytą odległość:

$$x_1 = \frac{1}{2}a{t_1}^2$$

$$x_2 = vt_2$$

$$x = x_1 + x_2$$

$$t = t_1 + t_2$$

Jak szybko będziemy się poruszać, gdy wyłączymy silniki odrzutowe? Jak się okazuje, tak naprawdę nie musimy tego wiedzieć. Prędkość można obliczyć, mnożąc przyspieszenie przez czas, przez który miało miejsce przyspieszenie. Innymi słowy, możemy zamienić v na at_1 i nie musimy się martwić o jego rzeczywistą wartość.

Mając całą tę wiedzę, możemy w końcu połączyć ją w jedno równanie, które rozwiążemy, aby określić, jak długo należy odpalać silniki odrzutowe.

$$x = x_1 + x_2$$

$$x = \frac{1}{2}a{t_1}^2 + vt_2$$

$$x = \frac{1}{2}a{t_1}^2 + at_1(2.5 - t_1)$$

$$x = \frac{1}{2}at_1{}^2 + 2.5at_1 - at_1{}^2$$

$$a = 1; x = 1.5$$

$$1.5 = 2.5t_1 - 0.5t_1{}^2$$

$$0 = -0.5t^2 + 2.5t - 1.5$$

Udało nam się usunąć wszystkie inne zmienne, ale tego równania nie da się rozwiązać samodzielnie. Po przekształceniu równania w przedstawiony powyżej sposób otrzymujemy równanie nazywane równaniem kwadratowym. Do rozwiązania tego typu równania można wykorzystać poniższe równanie:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$t = \frac{-2.5 \pm \sqrt{2.5^2 - (4 * -0.5 * -1.5)}}{2 * -0.5}$$

$$t = \frac{-2.5 \pm \sqrt{6.25 - 3}}{-1}$$

$$t = 2.5 \pm \sqrt{3.25}$$

$$t = 0.6972 \; lub \; 4.3027$$

To równanie zawsze daje dwie odpowiedzi, ale ciekawostką jest to, że zawsze można wybrać tę, która jest sensowna i logiczna w danej sytuacji. W tym przypadku 4,3 sekundy to o wiele za dużo czasu na odpalenie silników odrzutowych; przekroczylibyśmy współrzędne startu i zrujnowalibyśmy misję. Z drugiej strony, odpalenie silników odrzutowych na 0,6972 sekundy bardzo dobrze wpisuje się w nasze ramy czasowe i powinno pozwolić nam dotrzeć do współrzędnych startu na czas.

Należy jednak pamiętać, że jest to tylko połowa trasy. Aby ponownie zwolnić, musimy jeszcze odpalić przeciwległe silniki odrzutowe na 0,6972 sekundy. Ponadto w tych obliczeniach skupiliśmy się tylko na ruchu wzdłuż osi X; obliczenia należałoby powtórzyć dla ruchu wzdłuż osi Y. Okazuje się, że przesuwamy się o tę samą odległość wzdłuż obu osi, więc liczby są takie same. W efekcie końcowy czas pracy silników wyniesie 2,7888 sekundy (0,6972 sekundy w każdym kierunku), co umożliwi przejście z pozycji startowej (0,0) do współrzędnych startowych (3,3) w ciągu dokładnie pięciu sekund.

Również w tym przypadku twój zespół będzie musiał obliczyć, ile paliwa (mierzonego w sekundach czasu spalania) będzie potrzebne, aby doprowadzić rakietę SL-OMV do każdej z podanych współrzędnych startu w określonym czasie, za każdym razem całkowicie się zatrzymując.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający informacje o stanie początkowym startującego pojazdu. Informacje te zawierają następujące wartości rozdzielone spacjami:
    - Dodatnią liczbę całkowitą mniejszą lub równa 6 (**S**), oznaczająca liczbę satelitów CubeSat na pokładzie
    - Liczbę całkowitą reprezentującą początkową współrzędną X (w metrach)
    - Liczbę całkowitą reprezentującą początkową współrzędną Y (w metrach)
- **S** wierszy zawierających informacje o parametrach startowych każdego satelity CubeSat na pokładzie pojazdu nośnego, wymienione w wymaganej kolejności startu. Informacje te zawierają następujące wartości rozdzielone spacjami:
    - Liczbę całkowitą reprezentującą współrzędną X startu (w metrach)
    - Liczbę całkowitą reprezentującą współrzędną Y startu (w metrach)
    - Nieujemna wartość dziesiętna reprezentująca liczbę sekund, które musi zająć pojazdowi przebycie drogi z poprzedniej lokalizacji do lokalizacji o określonych współrzędnych.

```
2
2 0 0
3 3 5.0
5 5 3.0
2 0 0
1 7 10.0
5 3 6.0
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić informację o minimalnej ilości paliwa potrzebnego do wystrzelenia satelitów CubeSat w podanej kolejności i przy podanych parametrach startowych. Ilości paliwa należy mierzyć w sekundach czasu spalania, zaokrąglając je do dwóch miejsc po przecinku i uwzględniając ewentualne końcowe zera.

```
6.79
4.77
```

Problem 19: Ufaj, ale sprawdzaj

Punkty: 80

Autor: dr Francis Manning, Owego, Nowy Jork, Stany Zjednoczone

Kontekst problemu

Kluczowym zagadnieniem w projektowaniu i tworzeniu oprogramowania jest koncepcja walidacji danych. Skąd wiadomo, czy otrzymywane dane wejściowe nie zostały zmanipulowane? Jednym z podstawowych mechanizmów realizacji tego celu jest zastosowanie algorytmu haszującego.

Algorytm haszowania to sposób łączenia dowolnej ilości danych w mniejszą tablicę bajtów o określonym rozmiarze. Są to funkcje jednokierunkowe, więc na podstawie samego wyniku funkcji zwanego sumą kontrolną nie można odtworzyć oryginalnych danych. Ponadto są one deterministyczne, co oznacza, że z dowolnego bloku danych zawsze powstanie taka sama suma kontrolna. Chociaż sumy kontrolne nie są całkiem unikalne, gdyż istnieje nieskończona liczba sposobów łączenia danych, ale skończona liczba wartości sumy kontrolnej, i prawie niemożliwe jest celowe utworzenie zestawu danych, który zwróci taką samą sumę kontrolną jak inny zestaw. Dzięki tym właściwościom funkcje haszujące są idealne do sprawdzania poprawności danych.

Po obliczeniu sumy kontrolnej dla dowolnego zestawu danych można go opublikować. Każdy może ponownie uruchomić ten sam algorytm haszujący na danych źródłowych, a jeśli wynik nie zgadza się z podaną sumą kontrolną, jest to dowód na to, że dane zostały naruszone.

Opis problemu

Twój zespół współpracuje z Biurem Bezpieczeństwa Informacji Korporacyjnej (LMCIS) firmy Lockheed Martin nad wdrożeniem nowego, zautomatyzowanego procesu bezpieczeństwa w zakresie weryfikacji dokumentów. Twój program otrzyma ciąg tekstu reprezentujący dokument pobierany z serwerów firmy Lockheed Martin. Musi on obliczyć sumy kontrolne tych danych, a następnie porównać go ze sumą kontrolną dostarczoną przez inny serwer. Jeśli się ona zgadza, wszystko jest w porządku; jeśli nie, musisz oznaczyć błąd, aby pracownicy Lockheed Martin wiedzieli, że nie powinni otwierać tego dokumentu, gdyż został on naruszony!

W tym celu biuro LMCIS opracowało nowy algorytm haszujący (pamiętaj, że w rzeczywistości jest to zły pomysł. Aby zapewnić najlepsze bezpieczeństwo, należy zawsze używać wcześniej opublikowanych i bezpiecznych kryptograficznie algorytmów haszujących, jeśli tylko jest to konieczne). Algorytm haszujący działa według następującego schematu:

1. Konwertuje podany ciąg tekstowy na postać binarną, zastępując każdy znak w ciąg jego binarnym odpowiednikiem przedstawionym w tabeli ASCII w informacji referencyjnej, z zachowaniem zer wiodących. Na przykład literę „A” zastępuje się ciągiem „1000001”, a znak „.” zastępuje ciągiem „0101110”.
2. Dodaj dodatkowe bity „1” na końcu ciągu binarnego, aż jego długość będzie równa 128.

3. Podziel otrzymany ciąg binarny na „kawałki” po 32 bity każdy.
4. Zainicjuj zmienne o nazwach A, B, C i D jako 32-bitowe binarne odpowiedniki następujących liczb:
    a. A = 792250721
    b. B = 683117105
    c. C = 1215387974
    d. D = 1767829900
5. Dla każdego fragmentu (z kroku 3) oblicz nowe wartości dla A, B, C i D w następujący sposób. M oznacza bieżący fragment; N oznacza indeks tego fragmentu (pierwszy fragment ma indeks 0, drugi — 1 itd.).
    a. S = (B XOR D) AND (C OR (NOT B))
    b. S = A + S
    c. S = M + S
    d. Przesuń cyklicznie S w lewo o (N modulo 32) bitów
    e. Ustaw A = D, D = C, C = B oraz B = S
6. Złącz końcowe wartości A, B, C i D w jeden ciąg binarny (w tej kolejności)
7. Przekształć wynik na 32-znakowy ciąg szesnastkowy, zastępując każde cztery bity odpowiednim znakiem szesnastkowym (0000 = 0, 1111 = F itd.).

Jeśli nie znasz różnych operacji opisanych w kroku 5:

- A OR B: porównuje po kolei każdy bit w A i B. Jeśli co najmniej jeden bit w A lub B ma wartość „1”, odpowiadający mu bit w wyniku również ma wartość „1”. (np. 1100 OR 1010 = 1110)
- A AND B: porównuje po kolei każdy bit w A i B. Jeśli oba bity w A i B mają wartość „1”, to odpowiadający im bit w wyniku również ma wartość „1”. (np. 1100 AND 1010 = 1000)
- A XOR B: porównuje po kolei każdy bit w A i B. Jeśli dokładnie jeden bit w A lub B ma wartość „1”, to odpowiadający mu bit w wyniku również ma wartość „1”. (np. 1100 XOR 1010 = 0110)
- NOT A: odwraca wartość każdego bitu w A. (np. NOT 10 = 01)
- A + B: dodaje do siebie wartości A i B, a następnie przycina wynik, aby dopasować go do długości A i B (np. 1111 + 1010 = 1001).
- Przesunięcie cykliczne o X bitów w lewo powoduje przesunięcie X najbardziej znaczących (lewych) bitów na koniec liczby. (np. obrót o 1 bit w lewo dla 101010 daje wynik 010101; obrót o 2 bity daje wynik 101010).

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający ciąg tekstu, który ma zostać poddany skrótowi, o długości do 2000 znaków. Ten wiersz może zawierać dowolny znak wymieniony w tabeli US ASCII w informacjach referencyjnych.

- Wiersz zawierający 32 duże znaki szesnastkowe reprezentujące sumy kontrolnej, z którą ma zostać porównany.

```
2
This is a sample string for which you will need to calculate a hash.
45F1D2C216714CA65BE85211A364B2DB
Once calculated, compare the hash to the given hash and see if they match.
C2935B67EE5204A671D231E94676C101
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić pojedynczy wiersz ze słowem „TRUE”, jeśli obliczony przez Ciebie skrót pasuje do podanej wartości sumy kontrolnej lub „FALSE” jeżeli jest przeciwnie.

```
TRUE
FALSE
```

Aby ułatwić debugowanie, w tekście drugiego przypadku testowego powinna pojawić się wartość skrótu przedstawionego poniżej. Pamiętaj, że nie jest to część oczekiwanych danych wyjściowych:

```
43A442826ED0CCB0E89B4160E76087B3
```

Problem 20: Plink

Punkty: 85

Autor: Brett Reynolds, Annapolis Junction, Maryland, Stany Zjednoczone

Kontekst problemu

Plink to popularna gra karnawałowa, w której gracz wrzuca kulkę lub krążek do szczeliny, na szczycie planszy wypełnionej kołkami ułożonymi w przesuniętych względem siebie rzędach. Piłka toczy się po planszy, odbijając się od kołków, co powoduje zmianę jej toru. W końcu piłka wpadnie do jednego z kilku kubełków umieszczonych w dolnej części planszy. Obsługa przypisuje każdemu kubełkowi wartość punktową, a celem gry jest doprowadzenie do tego, by piłka wpadła do kubełka wartego najwięcej punktów.

Chociaż celem gry jest umieszczenie piłki w konkretnym kubełku to w rzeczywistości to zadanie jest trudne do zrealizowania. Fakt, że piłka się odbija sprawia, iż porusza się ona niemal losowo. Za każdym razem, gdy piłka uderza w kołek, ma niemal równe szanse na przesunięcie się na prawą lub lewą stronę kołka. W rezultacie piłka częściej trafia do kubełków znajdujących się na środku planszy, ponieważ prowadzi do nich więcej ścieżek, a więc jest większa szansa, że piłka tam wyląduje. Do kubełków znajdujących się blisko krawędzi prowadzi mniej ścieżek, dlatego piłka rzadziej tam trafia. W rezultacie kubełki znajdujące się blisko krawędzi mają zwykle wyższą wartość niż te znajdujące się w środku.

Współpracujesz z firmą produkującą gry, aby przetestować nową wersję gry Plink — wersję, która wymaga większych umiejętności i eliminuje część losowości. W tej wersji gry występują trzy zasadnicze różnice:

- Piłkę można wprawić w ruch tylko na środku planszy, a nie w dowolnym miejscu na jej szczycie.
- Okrągłe kołki zostały zastąpione łopatkami, które gracz może przechylać w lewo lub w prawo, co pozwala mu kierować torem piłki.
- Każdy kołek ma swoją wartość punktową; gdy piłka spada, gracz dodaje do swojego wyniku wartość każdego kołka, w który uderzyła kulka. Wartość ta jest następnie dodawana do wartości kubełka, w którym wylądowała piłka, aby określić ostateczny wynik gracza.

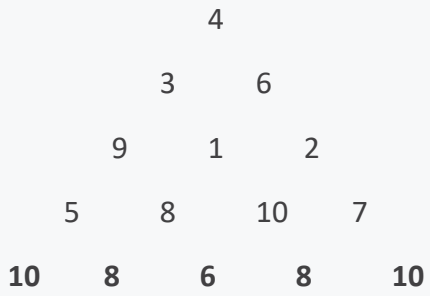
W wyniku zmiany formatu gracz może manipulować torem piłki, aby zmaksymalizować swój wynik. Celowanie w kubełek, któremu przypisana jest wysoka wartość i znajduje się na skraju planszy może nie przynieść najlepszego wyniku, jeśli jedyna ścieżka prowadząca do niego zawiera tylko kołki o niskiej wartości. Aby uzyskać wysoki wynik, gracz musi znaleźć optymalną drogę dla piłki, która spada przez planszę.

Opis problemu

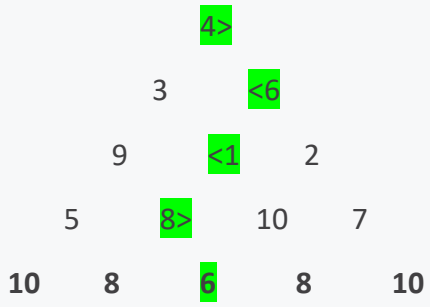
Musisz zaprojektować program, który potrafi określić optymalną ścieżkę dla danej planszy Plink, czyli taką, która jest warta największą liczbę punktów przy określonym układzie wartości punktowych i wartości kubełków. Twoja trasa będzie wyrażona w postaci serii liter (L lub R) wskazujących, czy piłka

powinna poruszać się odpowiednio w lewo czy w prawo od każdej łopatki, w którą uderzy podczas spadania. Na każdym poziomie planszy piłka może trafić tylko w jedną łopatkę, więc liczba podanych kierunków będzie równa liczbie poziomów na planszy.

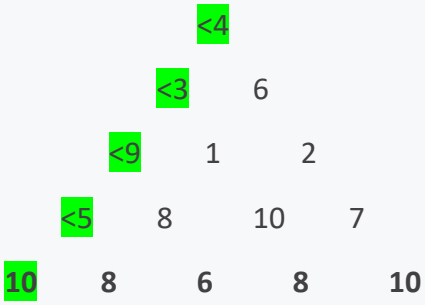
Weźmy na przykład poniższą tablicę. Każda liczba w trójkącie oznacza wartość punktową łopatki w danym położeniu. Liczby wytłuszczone na dole oznaczają wartości kubełków na dole planszy.

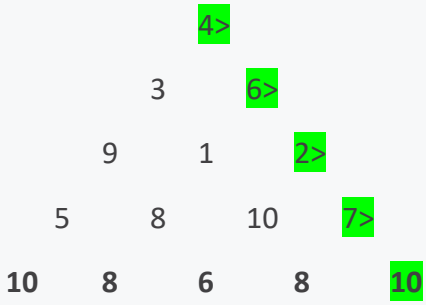


Mimo niewielkich rozmiarów planszy, można na niej znaleźć 16 możliwych ścieżek. Na przykład ścieżka „RLLR” daje wynik 25 punktów:

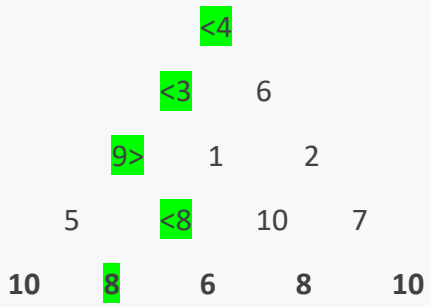


Nie jest to optymalna ścieżka dla tej tablicy. Piłka ląduje w kubełku na środku planszy, któremu przypisana jest niska wartość, a także trafia w łopatkę wartą tylko jeden punkt. W tradycyjnym grze w Plink najlepiej jest celować w krawędzie planszy; tutaj można to osiągnąć za pomocą ścieżek „LLLL” i „RRRR”.





Ścieżka „tylko w lewo” skutkuje uzyskaniem 31 punktów, a ścieżka „tylko w prawo” — 29 punktów. Oba rozwiązania są lepsze niż ścieżka, którą omawialiśmy wcześniej. Jednak żadna z tych ścieżek nie jest optymalna. Jest jedna ścieżka, na której można uzyskać 32 punkty, mianowicie ścieżka „LLRL”:



Nawet jeśli tracimy dwa punkty, umieszczając piłkę w innym kubełku, zyskujemy trzy punkty, celując w 8-punktową łopatkę pośrodku, zamiast pozostawać na krawędzi.

Twoim zadaniem jest napisanie programu, który potrafi obliczyć ścieżkę, która dla danej planszy Plink uzyska maksymalną możliwą liczbę punktów. Ostrzegamy jednak, że przykłady te dotyczyły bardzo małej planszy, składającej się z zaledwie czterech rzędów łopatek. Twój zespół ma nadzieję zaprojektować znacznie większe plansze, aby ponieść poziom trudności gry. Każdy dodatkowy rząd łopatek zwiększa liczbę możliwych rozwiązań o kolejną potęgę liczby dwa. Twój program musi dawać pewność, że określił najlepszą drogę ze wszystkich możliwych.

Przypominamy, że twój program musi zakończyć działanie w ciągu dwóch minut, w przeciwnym razie zostanie oznaczony jako niepoprawny (nawet jeśli ostatecznie zwróciłby poprawną odpowiedź).

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dodatnią liczbę całkowitą (**X**) oznaczającą liczbę rzędów łopatek na planszy Plink.
- **X** wierszy, zawierających wartości punktowe liczb całkowitych łopatek w odpowiednim wierszu tabeli, w kolejności od lewej do prawej. Wartości są rozdzielone spacjami, a wiersze będą zawierały określone wartości, równe numerowi wiersza tablicy (pierwszy wiersz to wiersz 1 i będzie zawierał 1 wartość).
- Linia zawierająca **X+1** liczb całkowitych rozdzielonych spacjami, reprezentujących wartości punktowe kubełków na dole planszy, w kolejności od lewej do prawej.

```
1
4
4
3 6
9 1 2
5 8 10 7
10 8 6 8 10
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wiersz zawierający:

- Ciąg **X** wielkich liter L lub R, reprezentujący optymalną ścieżkę przez daną planszę Plink, zgodnie z powyższym opisem.
- Spację.
- Znak równości.
- Spację.
- Liczbę całkowitą reprezentującą liczbę punktów uzyskanych z konkretnej ścieżki.

```
LLRL = 32
```

Problem 21: Szał zakupów

Punkty: 90

Autorzy: Cindy Gibson i Danny Lin, Greenville, Karolina Południowa, Stany Zjednoczone

Kontekst problemu

W ciągu ostatnich dwóch dekad zakupy przez internet stały się prężną gałęzią przemysłu. Od wielkich korporacji, takich jak Amazon, po małe firmy działające w piwnicach swoich właścicieli — w obecnych czasach niewiele jest rzeczy, których nie można zamówić przez internet i oczekiwać, że zostaną dostarczone do domu kilka dni później. Koncepcja cyfrowego „koszyka” rozszerzyła się nawet na inne zastosowania, a obecnie jest powszechnym, intuicyjnym sposobem przeglądania i organizowania wszystkiego, co zostało wybrane na stronie internetowej.

Większość interfejsów koszyków ma dwie wspólne cechy: użytkownicy mogą przeglądać lub modyfikować zawartość tylko własnego koszyka, a produkty z koszyka nie są kupowane, dopóki użytkownik nie zakończy całego procesu zakupów. Różni się to od fizycznego koszyka na zakupy, jaki można zobaczyć w sklepie spożywczym; tam kupujący fizycznie zdejmuje produkt z półki, czyniąc go niedostępnym dla innych klientów. W internecie samo umieszczenie produktu w koszyku zwykle nie oznacza, że produkt jest w jakikolwiek sposób zarezerwowany. Inna osoba może umieścić ten sam produkt (te same produkty) w swoim koszyku i zakończyć proces swojego zakupu, zanim Ty będziesz mieć szansę to zrobić, co może spowodować, że wybranych przez Ciebie produktów nie będzie już w magazynie.

Opis problemu

W zakładzie Lockheed Martin, w którym pracujesz, powstaje nowy zautomatyzowany sklep, w którym przez cały dzień pracownicy będą mogli kupować przekąski i napoje. Pracownicy mogą wybierać produkty z witryny internetowej, a sprzęt w sklepie umieszcza je w szafce, z której możesz je odebrać. Twój zespół został wyznaczony do zbudowania witryny internetowej. Jak każda witryna internetowa, projektowana przez Ciebie strona będzie musiała przetwarzać polecenia każdego użytkownika w miarę ich napływania. Musisz napisać program do obsługi tego przetwarzania. Ponieważ jednak w tym samym czasie może być połączonych wielu użytkowników, polecenia jednego z nich mogą być mieszane z poleceniami innych.

Jest to nowa koncepcja, dlatego zapasy magazynowe sklepu nie są zbyt duże i może się zdarzyć, że sklep będzie często wyprzedawał towary. Na początku każdego dnia Twój program będzie otrzymywał informacje o stanie magazynowym sklepu i powinien zarządzać nim w miarę napływu zamówień. Klienci nie powinni mieć możliwości dodawania do koszyka produktów, których zapasy są już wyczerpane.

Wraz z zakończeniem przez klienta procesu zakupu program powinien zsumować koszt wszystkich pozycji, które klient ma w koszyku, i usunąć te pozycje z inwentarza. Jeśli w okresie od dodania do koszyka użytkownika do momentu przejścia do kasy wyczerpały się zapasy jakiegokolwiek produktu, użytkownik powinien zostać poinformowany, że te produkty nie są dostępne i nie powinien zostać za nie

obciążony kosztami. Użytkownik będzie mógł powrócić do witryny w późniejszym czasie i spróbować ponownie zamówić produkty, gdy zostaną one ponownie zatowarowane w magazynie.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite rozdzielone spacjami:
  - **I**: liczba rodzajów przedmiotów w inwentarzu
  - **C**: liczba poleceń od klientów otrzymanych przez witrynę
- **I** wierszy z listą pozycji znajdujących się w inwentarzu Każdy wiersz będzie zawierał trzy wartości rozdzielone spacjami:
  - Liczbę całkowitą dodatnią określającą stan magazynowy.
  - Ciąg tekstowy zawierający nazwę przedmiotu. Może ona zawierać duże i małe litery oraz znak podkreślenia (_).
  - Dodatnią wartość dziesiętną określającą koszt pojedynczego przedmiotu tego typu.
- **C** wierszy zawierających listę poleceń otrzymanych od klientów. Każda komenda rozpoczyna się dodatnią liczbą całkowitą reprezentującą identyfikator klienta; wszystkie komendy otrzymane od danego klienta rozpoczynają się od tej samej wartości identyfikatora. Pozostała część polecenia będzie zawierała kilka wartości rozdzielonych spacjami. Możliwe polecenia to:
  - ADD <ITEM> <QTY>: służy do dodawania przedmiotów do koszyka. <ITEM> będzie nazwą przedmiotu w takiej postaci, w jakiej występuje on w raporcie inwentaryzacji, a <QTY> zostanie zastąpione dodatnią liczbą całkowitą określającą liczbę przedmiotów do dodania. Jeśli żądana liczba plus liczba tych przedmiotów, które znajdują się już w koszyku klienta, jest równa lub mniejsza od liczby przedmiotów znajdujących się obecnie w magazynie, dodaj tę liczbę przedmiotów do koszyka klienta, ale nie usuwaj ich w tym momencie z magazynu. W przeciwnym razie polecenie powinno zostać odrzucone.
  - REMOVE <ITEM> <QTY>: służy do usuwania przedmiotów z koszyka. <ITEM> będzie nazwą przedmiotu w takiej postaci, w jakiej występuje on w raporcie inwentaryzacji, a <QTY> zostanie zastąpione dodatnią liczbą całkowitą określającą liczbę przedmiotów do usunięcia. Jeśli żądana liczba i typ przedmiotów znajduje się w koszyku klienta, należy je z niego usunąć. W przeciwnym razie polecenie powinno zostać odrzucone.
  - CHECKOUT: służy do finalizowania zamówienia klienta. Zawartość koszyka użytkownika powinna zostać porównana z aktualnym stanem magazynowym. W przypadku przedmiotów, których klient nie może otrzymać w całkowitej żądanej ilości, nie wydawaj klientowi żadnego z tych przedmiotów; zamiast tego usuń dany przedmiot z koszyka i zgłoś niedobór. Wszystkie pozostałe przedmioty należy następnie usunąć z inwentarza, a całkowity koszt tych przedmiotów przekazać użytkownikowi.

```
1
3 8
2 Candy_Bar 1.50
3 Soda_Bottle 1.60
2 Cheese_Pack 2.50
1 ADD Candy_Bar 1
2 ADD Candy_Bar 2
1 ADD Soda_Bottle 3
1 REMOVE Soda_Bottle 1
2 CHECKOUT
1 CHECKOUT
3 ADD Candy_Bar 1
3 ADD Cheese_Pack 2
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić wyniki każdego otrzymanego polecenia w następującej kolejności:

- Polecenia ADD:
  - Jeśli realizacja polecenia zakończy się powodzeniem, zwrócony musi zostać pojedynczy wiersz w formacie: „Added <QTY> <ITEM> to customer <ID>’s cart”.
  - Jeśli polecenie zostanie odrzucone, zwrócony musi zostać pojedynczy wiersz w formacie: „Not enough <ITEM> for customer <ID>”.
  - W obu przypadkach musisz zastąpić wartość <QTY> liczbą dodanych przedmiotów, <ITEM> — nazwą przedmiotu, a <ID> — numerem identyfikacyjnym klienta.
- Polecenia REMOVE:
  - Jeśli realizacja polecenia zakończy się powodzeniem, zwrócony musi zostać pojedynczy wiersz w formacie: „Removed <QTY> <ITEM> from customer <ID>’s cart”.
  - Jeśli polecenie zostanie odrzucone, zwrócony musi zostać pojedynczy wiersz w formacie: „Customer <ID> does not have that many <ITEM>s”.
  - W obu przypadkach musisz zastąpić wartość <QTY> liczbą dodanych przedmiotów, <ITEM> — nazwą przedmiotu, a <ID> — numerem identyfikacyjnym klienta.
- Polecenia CHECKOUT:
  - Dla każdego przedmiotu, dla którego nie ma wystarczającego zapasu celem realizacji polecenia klienta, zwrócony musi zostać pojedynczy wiersz w formacie: „Out of stock of <ITEM>”. W przypadku niewystarczających zapasów wielu przedmiotów listowanie musi nastąpić w porządku alfabetycznym.
  - Na koniec zwrócony musi zostać wiersz w formacie: „Customer <ID>’s total: $<COST>”, w którym <ID> musi zostać zastąpione numerem identyfikacyjnym klienta, a <COST> — całkowitym kosztem wszystkich przedmiotów pozostałych w koszyku klienta, zaokrąglonym do dwóch miejsc po przecinku i zawierającym wszelkie zera resztkowe.

Po przetworzeniu wszystkich poleceń program musi również zwrócić jeden wiersz dla każdego przedmiotu pozostającego w magazynie, w porządku alfabetycznym, w formacie „<ITEM> - <QTY>”,

gdzie <ITEM> musi zostać zastąpione nazwą przedmiotu, a <QTY> — ilością pozostającą w magazynie. Pomiń pozycje, które są niedostępne w magazynie.

```
Added 1 Candy_Bar to customer 1’s cart
Added 2 Candy_Bar to customer 2’s cart
Added 3 Soda_Bottle to customer 1’s cart
Removed 1 Soda_Bottle from customer 1’s cart
Customer 2’s total: $3.00
Out of stock of Candy_Bar
Customer 1’s total: $3.20
Not enough Candy_Bar for customer 3
Added 2 Cheese_Pack to customer 3’s cart
Cheese_Pack - 2
Soda_Bottle - 1
```

Problem 22: Wezwanie dla wszystkich strażaków

Punkty: 100

Autor: dr Leon Clark, Melbourne, Wiktoria, Australia

Kontekst problemu

W razie katastrofy utrzymywanie otwartych linii komunikacyjnych ze wszystkimi zaangażowanymi osobami — zarówno z osobami, które zareagowały na nią jako pierwsze, jak i ze zwykłymi cywilami, którzy znaleźli się w zasięgu jej oddziaływania — jest ważne dla potrzeb ograniczenia liczby ofiar i rozmiarów szkód. Niestety warunki geograficzne, takie jak ukształtowanie terenu (góry), mogą utrudniać komunikację. Wiele nowoczesnych środków komunikacji opiera się na sygnałach radiowych, które mogą być blokowane przez duże góry lub inne obiekty, które blokują bezpośrednią linię widzenia między wieżą sygnałową a urządzeniem komunikacyjnym.

Opis problemu

Lockheed Martin współpracuje z Australijskimi Siłami Obronnymi (ADF) nad stworzeniem sposobu szybkiego ustanowienia nowych przekaźników komunikacyjnych na wypadek kolejnej rundy katastrofalnych pożarów. Siły ADF planują dostarczyć firmie Lockheed Martin przekrój mapy topograficznej obrazującej układ terenu w obszarze oddziaływania. Mapa ta będzie zawierać numery wskazujące proponowane lokalizacje wieży komunikacyjnej (oznaczonej cyfrą 0) oraz kilku stanowisk dowodzenia (oznaczonych cyframi od 1 do 9). Twój zespół został poproszony o zaprojektowanie algorytmu, który na podstawie położenia wieży komunikacyjnej będzie w stanie określić, które ze stanowisk dowodzenia są możliwe do wykorzystania.

Mapa ma bardzo niską rozdzielczość, dlatego każda komórka na mapie reprezentuje obszar o powierzchni jednego kilometra kwadratowego. Każdy obszar jest uważany za całkowicie wypełniony albo stałym gruntem (#), albo otwartą przestrzenią (spacja lub liczba). Aby stanowisko dowodzenia znajdowało się w odpowiednim miejscu, strażacy muszą mieć możliwość umieszczenia anteny komunikacyjnej w centrum tego miejsca; antena ta musi mieć bezpośredni, niezakłócony widok na wieżę. Nadajniki (zarówno anteny, jak i wieży) będą zlokalizowane dokładnie w środku ich obszarów (pół kilometra od krawędzi komórki wyrysowanej na mapie). Jeśli linia prosta poprowadzona z tych dwóch punktów zetknie się z ziemią w dowolnym miejscu, sygnał z wieży zostanie przynajmniej częściowo zablokowany, co zakłóci łączność i sprawi, że wykorzystanie danej lokalizacji stanowiska dowodzenia nie będzie możliwe.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający trzy dodatnie liczby całkowite, rozdzielone spacjami, jak poniżej:

  - **H**: przedstawiająca wysokość mapy w rzędach (minimum 4)
  - **W**: przedstawiająca szerokość mapy w znakach (minimum 30)
  - **C**: przedstawiająca liczbę stanowisk dowodzenia, które będą wyświetlane na mapie (minimum 2)
- **H** wierszy, z których każdy zawiera do **W** znaków, przedstawiających przekrój mapy topograficznej. Każdy wiersz może zawierać dowolny z poniższych znaków:
  - # reprezentujący stały grunt, który blokuje sygnały radiowe.
  - Spację reprezentującą otwartą przestrzeń. Pamiętaj, że wiersze z danymi wejściowymi nie będą zawierać końcowych znaków niedrukowalnych; jeśli wiersz zawiera mniej niż **W** znaków, można założyć, że wszystkie brakujące znaki po ostatnim widocznym znaku mają być spacjami.
  - Cyfrę 0 (zero), oznaczająca proponowaną lokalizację wieży komunikacyjnej. W każdym przypadku testowym będzie dokładnie jedno 0.
  - Dowolną liczbę od 1 (jeden) do **C** włącznie, oznaczającą proponowaną lokalizację stanowiska dowodzenia. W każdym przypadku testowym będzie dokładnie jedno wystąpienie każdej odpowiedniej liczby.

```
2
6 45 4
                                4
            0                  ####
           ####               #######
    #############   2       ############
    3 ################   ################  1
#############################################
5 45 4
            1                  4####
           ####               #######
    #############   2       ############
    3 ################   ################  0
#############################################
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić jeden wiersz, jak poniżej:

- Jeśli co najmniej jedno stanowisko dowodzenia znajduje się w nadającej się do wykorzystania lokalizacji, zwrócona musi zostać lista numerów odpowiadających tym miejscom, w porządku rosnącym, rozdzielanych spacjami.
- Jeśli żadne stanowiska dowodzenia nie znajdują się w nadających się do wykorzystania lokalizacjach, zwrócony musi zostać komunikat „No viable locations”.

```
2 4
No viable locations
```

Problem 23: Labirynt aMAZEing

Punkty: 100

Autor: Anonimowy wolontariusz, Orlando, Floryda, Stany Zjednoczone

Kontekst problemu

Labirynty od lat wzbudzają zainteresowanie ludzi. Angielskie słowo „maze” pochodzi od staroangielskiego słowa oznaczającego delirium lub urojenie. Choć labirynty określane angielskimi słowami „maze” i „labyrinth” nie są w rzeczywistości tym samym, często używa się ich synonimicznie. Labirynty określane angielskim słowem „labyrinth” to zazwyczaj wiele krętych, zakrzywionych przejść; natomiast labirynty określane angielskim słowem „maze” przypominają raczej puzzle i mają wiele ślepych zakończeń. Mamy starożytną mitologię grecką, w której bohaterowie uciekają przed potworami w labiryncie. Anglicy budowali labirynty z żywopłotu w ogrodach wielu swoich zamków. Dziś wielu rolników po żniwach buduje labirynty na polach kukurydzy.

Rozwiązywanie labiryntów może być świetną zabawą i właśnie na tym polega to wyzwanie. Czy potrafisz znaleźć najkrótszą drogę w prostym labiryncie prostokątnym?

Opis problemu

Otrzymasz kilka prostokątnych labiryntów i twoim zadaniem będzie napisać program, który odczyta schemat labiryntu, a następnie wyznaczy najkrótszą ścieżkę przez labirynt i poda, ile „komórek” trzeba było pokonać, aby przejść od wejścia do wyjścia. Ruchy są wykonywane tylko w poziomie i w pionie, i oczywiście nie można ich wykonywać przez ściany.

Każda „komórka” w labiryncie jest przedstawiona jako poziomy prostokąt znaków o wymiarach 3 na 4, ograniczony w każdym rogu znakiem plusa (+). Każda komórka dzieli swoje krawędzie z sąsiadami, tzn. prawa krawędź jednej komórki jest jednocześnie lewą krawędzią następnej komórki. Kreski (-) i kreski pionowe (|) są używane między znakami plusa, aby wskazać, czy krawędź komórki zawiera nieprzekraczalną ścianę; w przeciwnym razie znaki te będą spacjami, oznaczającymi otwartą przestrzeń, przez którą można przejść. W środku każdej komórki zawsze będą znajdować się dwie spacje. Zobacz przykładowe komórki poniżej:

<pre>+--+--+
| | |
+--+--+</pre>
	<pre>+--+
	   |
	+ +</pre>

Dwie w pełni zamknięte komórki, ze ściankami ze wszystkich stron	Komórka z otwartą lewą i dolną krawędzią

Każdy labirynt będzie miał jedno wejście i jedno wyjście, które będą znajdować się wzdłuż zewnętrznej krawędzi labiryntu (mogą znajdować się po tej samej stronie). Będą one reprezentowane jednym z poniższych zestawów znaków, zastępującym krawędź komórki:

- Dwie małe litery „v” zastąpią kreski, aby reprezentować wejście wzdłuż górnej krawędzi labiryntu i/lub wyjście wzdłuż dolnej krawędzi labiryntu.
- Dwie karety (^^) zastąpią kreski, aby reprezentować wejście wzdłuż dolnej krawędzi labiryntu i/lub wyjście wzdłuż górnej krawędzi labiryntu.
- Prawy nawias ostry (>) zastąpi kreskę pionową, aby reprezentować wejście wzdłuż lewej krawędzi labiryntu i/lub wyjście wzdłuż prawej krawędzi labiryntu.
- Lewy nawias ostry (<) zastąpi kreskę pionową, aby reprezentować wejście wzdłuż prawej krawędzi labiryntu i/lub wyjście wzdłuż lewej krawędzi labiryntu.

W każdym labiryncie może być więcej niż jedna realna droga od wejścia do wyjścia; również w tym przypadku twoim celem jest znalezienie najkrótszej drogi i odnotowanie liczby komórek, przez które przejście było wymagane.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał:

- Wiersz zawierający dwie dodatnie liczby całkowite, rozdzielone spacją, reprezentujące odpowiednio wysokość labiryntu w wierszach (**H**) oraz jego szerokość w znakach drukowanych (**W**).
- **H** wierszy, każdy o długości **W** znaków, przedstawiających schemat labiryntu. Wiersze te mogą zawierać kreski (-), znaki plusa (+), kreski pionowe (|), spacje, karety (^), małe litery „v” i/lub nawiasy ostre (> lub <), jak opisano powyżej.

Aby uniknąć przenoszenia między stronami, przykładowe dane wejściowe zamieszczone są na następnej stronie.

```
2
11 16
+vv+--+--+--+--+
|        |     |
+--+  +--+  +--+
|  |     |     |
+  +--+  +--+  +
|              |
+  +--+--+  +--+
|     |        |
+--+  +  +--+  +
|           |  >
+--+--+--+--+--+
11 31
+--+--+--+--+--+--+--+--+--+--+
<        |     |     |     |  |
+--+  +--+  +--+  +  +  +  +  +
|  |     |        |     |     |
+  +--+  +  +--+--+--+--+--+  +
|  |        |     |           |
+  +--+--+--+  +  +  +--+  +--+
|     |        |     |        |
+--+  +  +--+--+  +--+  +--+  +
>           |     |     |     |
+--+--+--+--+--+--+--+--+--+--+
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić liczbę komórek, które zostały przekroczone podczas przechodzenia od wejścia do wyjścia.

```
9
31
```

Problem 24: Codzienna harówka

Punkty: 110

Autor: John Worley, Fort Worth, Teksas, Stany Zjednoczone

Kontekst problemu

W Stanach Zjednoczonych i wielu innych krajach stosuje się 40-godzinny tydzień pracy. Niezależnie od tego, jak długo pracownik pracuje w ciągu dnia, jego plan pracy zaokrągla się do około 40 godzin w ciągu każdego siedmiodniowego okresu. Jednak sposób, w jaki osiąga się te 40 godzin, bywa różny, a różnice mogą doprowadzić księgowych z działu kadr do szału.

Firma Lockheed Martin przeszła ostatnio na harmonogram pracy 4/10 — pracownicy pracują tylko przez cztery dni w tygodniu (od poniedziałku do czwartku), ale każdego dnia pracują po dziesięć godzin. Harmonogram ten jest jednak elastyczny i pracownicy mogą współpracować ze swoimi przełożonymi, aby wypracować harmonogram, który będzie dla nich najbardziej odpowiedni. Dział kadr potrzebuje pomocy w uporządkowaniu różnych wniosków dotyczących harmonogramów, aby określić, ile pracy zostanie faktycznie wykonane w danym miesiącu.

Opis problemu

W ciągu ostatnich kilkudziesięciu lat pojawiło się wiele różnych harmonogramów pracy, a cztery z nich były ostatnio dość powszechnie stosowane w Lockheed Martin:

- **40-godzinny** czas pracy to tradycyjny tydzień pracy; pracownicy pracują po 8 godzin dziennie przez 5 dni w tygodniu (od poniedziałku do piątku).
- Harmonogram **4/10** skraca tydzień pracy; pracownicy pracują po 10 godzin dziennie przez 4 dni w tygodniu (od poniedziałku do czwartku).
- Harmonogram **9/80** działa nieco inaczej; pracownicy pracują po 9 godzin dziennie, przez 4 dni w tygodniu (od poniedziałku do czwartku), a dodatkowo przez 8 godzin w co drugi piątek. W sumie daje to 80 godzin w okresie dwóch tygodni, czyli średnio 40 godzin tygodniowo. Istnieją dwie wersje tego harmonogramu, 9/80A i 9/80B, które określają, które piątki są wolne. Na potrzeby tego zadania:
    - Harmonogram **9/80A** będzie oznaczał pracę w pierwszy piątek każdego roku kalendarzowego (bez uwzględniania innych lat)
    - Harmonogram **9/80B** będzie oznaczał dzień wolny w pierwszy piątek każdego roku kalendarzowego (bez uwzględniania innych lat).

System księgowy Lockheed Martin rejestruje liczbę dni roboczych według okresów, a nie miesięcy kalendarzowych. Każdy okres jest skoncentrowany wokół miesiąca kalendarzowego, ale nie może zaczynać się ani kończyć w tych samych dniach. Każdy okres kończy się w ostatni piątek miesiąca kalendarzowego, a następny poniedziałek jest początkiem kolejnego okresu. Na przykład, okres dla kwietnia 2022 r. trwa od poniedziałku 28 marca do piątku 29 kwietnia; okres dla maja 2022 r. trwa od poniedziałku 2 maja do piątku 27 maja.

Dział kadr poprosił Twój zespół o stworzenie kalkulatora, który określi prawidłową liczbę dni roboczych dla każdego harmonogramu w okresie zawierającym dowolną datę. Jeśli dana data przypada na weekend (sobotę lub niedzielę), do wyznaczenia właściwego okresu należy użyć poprzedzającego piątku. Nie martw się o święta, urlopy czy inne potencjalne wolne dni; jednym z powodów, dla których dział kadr chce uzyskać te informacje, jest ustalenie, kiedy należy zaplanować wolne dni!

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych twojego programu, **odebrany ze standardowego kanału wejściowego**, będzie zawierał dodatnią liczbę całkowitą reprezentującą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierał jeden wiersz z datą w formacie DD/MM/RRRR.

```
3
14/03/2022
24/04/2022
29/05/2022
```

Przykładowe dane wyjściowe

Dla każdego przypadku testowego twój program musi zwrócić jeden wiersz zawierający liczbę dni roboczych dla każdego harmonogramu pracy w okresie zawierającym podaną datę. Liczba dni roboczych w każdym harmonogramie musi zostać zwrócona jako liczba całkowita, rozdzielona spacjami, w następującej kolejności:

- 40-godzinny
- 4/10
- 9/80A
- 9/80B

```
20 16 18 18
25 20 23 22
20 16 18 18
```