Bartosz Kostka, Jakub Radoszewski
Tłumaczenie

Dostępna pamięć: 2 GB, czas: 2 s. *IOI 2016, dzień drugi, 16.08.2016*

Wykrywanie wrednej usterki

Ilshat jest inżynierem oprogramowania pracującym nad wydajnymi strukturami danych. Pewnego dnia wymyślił on nową strukturę danych, która przechowuje zbiór **nieujemnych** *n-bitowych liczb całkowitych, gdzie n jest potęgą dwójki (czyli $n = 2^b$ dla pewnego nieujemnego, całkowitego b).*

Struktura danych początkowo jest pusta. Program używający tej struktury danych musi przestrzegać następujących zasad:

- *Program może dodawać do struktury danych elementy, które są n-bitowymi liczbami całkowitymi, każdą pojedynczo, używając funkcji* `add_element(x)`*. Jeżeli program spróbuje dodać do struktury danych element już się w niej znajdujący, nic się nie dzieje.*
- *Po dodaniu ostatniego elementu program powinien wywołać funkcję* `compile_set()` *(dokładnie raz).*
- *Następnie program może wywoływać funkcję* `check_element(x)`*, aby sprawdzać, czy element x znajduje się w strukturze danych. Ta funkcja może być używana wielokrotnie.*

Kiedy Ilshat zaimplementował prototyp swojej struktury danych, miał on buga w funkcji `compile_set()`*. Bug ten zmienia kolejność cyfr binarnych każdego elementu zbioru w pewien ustalony sposób. Ilshat chciałby, abyś wykrył, jakie przestawienie cyfr powoduje bug.*

Formalnie, rozważmy ciąg $p = [p_0, \ldots, p_{n-1}]$, w którym każda liczba od 0 do $n - 1$ występuje dokładnie raz. Taki ciąg nazywamy **permutacją***. Rozważmy element zbioru, którego cyfry w zapisie binarnym to kolejno $a_0, \ldots, a_{n-1}$ (gdzie a_0 jest najbardziej znaczącym bitem). W momencie wywołania funkcji* `compile_set()`*, element ten jest podmieniany na element $a_{p_0}, a_{p_1}, \ldots, a_{p_{n-1}}$.*

Ta sama permutacja p jest używana do zamiany kolejności cyfr każdego elementu zbioru. Permutacja ta może być dowolna. W szczególności, może się zdarzyć, że $p_i = i$ dla każdego $0 \leqslant i \leqslant n - 1$.

Dla przykładu, załóżmy, że $n = 4$, $p = [2, 1, 3, 0]$, a Ty dodałeś do struktury liczby, których binarna reprezentacja to `0000`*,* `1100` *oraz* `0111`*. Wywołanie funkcji* `compile_set` *zmienia te elementy odpowiednio na* `0000`*,* `0101` *i* `1110`*.*

Twoim zadaniem jest napisanie programu, który odkryje permutację p poprzez zadawanie zapytań do struktury danych. Program powinien (w następującej kolejności):

1. *wybrać zbiór liczb n-bitowych,*
2. *dodać te liczby do struktury danych,*
3. *wywołać funkcję* `compile_set`*, aby spowodować buga,*
4. *zbadać istnienie pewnych elementów w strukturze danych,*
5. *użyć tych informacji aby odkryć i zwrócić permutację p.*

Zauważ, że Twój program może wywołać funkcję `compile_set` *jedynie raz.*

Dodatkowo istnieją pewne ograniczenia na liczbę wywołań funkcji bibliotecznych. Twój program może

- *wywołać funkcję* `add_element` *co najwyżej w razy (w jest od angielskiego słowa zapisy – „writes”),*
- *wywołać funkcję* `check_element` *co najwyżej r razy (r jest od angielskiego słowa odczyty – „reads”).*

Szczegóły implementacji

Twoim zadaniem jest zaimplementowanie jednej funkcji (metody):

- `int[] restore_permutation(int n, int w, int r)`
  - `n`: *liczba bitów w binarnej reprezentacji każdego elementu zbioru (a także długość permutacji p),*
  - `w`: *maksymalna liczba operacji* `add_element`*, którą może wykonać Twój program,*
  - `r`: *maksymalna liczba operacji* `check_element`*, którą może wykonać Twój program.*
  - *Funkcja powinna zwracać znalezioną permutację p.*

W języku C sygnatura funkcji jest minimalnie inna:

- `void restore_permutation(int n, int w, int r, int* result)`
  - `n`, `w` *i* `r` *mają takie same znaczenie jak powyżej.*
  - *Funkcja powinna zwracać znalezioną permutację p poprzez zapisanie jej do dostarczonej tablicy* `result`*: dla każdego i, powinna ona zapisać wartość* p_i *do* `result[i]`.

Funkcje biblioteczne

Do komunikowania się ze strukturą danych Twój program powinien używać następujących trzech funkcji (metod):

- `void add_element(string x)`
  - *Funkcja ta dodaje element opisany przez* `x` *do struktury danych.*
  - `x`: *napis złożony ze znaków* `0` *i* `1` *będący reprezentacją binarną liczby, która powinna być dodana do struktury danych. Długość* `x` *musi wynosić n.*
- `void compile_set()`
  - *Funkcja ta powinna być wywołana dokładnie raz. Twój program nie może wywołać funkcji* `add_element()` *po wywołaniu opisywanej funkcji. Twój program nie może także wywołać funkcji* `check_element()` *przed wywołaniem opisywanej funkcji.*

- `boolean check_element(string x)`
  - *Ta funkcja sprawdza, czy element* `x` *znajduje się w strukturze danych (po zastosowaniu permutacji* p*).*
  - `x`: *napis złożony ze znaków* `0` *oraz* `1`*, będący reprezentacją elementu, którego istnienie chcemy sprawdzić. Długość* `x` *musi wynosić* n*.*
  - *zwraca* `true`*, jeżeli element* `x` *znajduje się w strukturze danych, natomiast* `false` *w przeciwnym razie.*

Pamiętaj, że jeżeli Twój program złamie którąś z wymienionych reguł, wynikiem jego sprawdzania będzie „Zła odpowiedź”.

Dla każdego napisu pierwszy znak odpowiada za najbardziej znaczący bit odpowiadającej liczby.

Program sprawdzający ustala permutację p *przed wywołaniem funkcji* `restore_permutation`*.*

Szczegóły implementacji w Twoim języku programowania znajdują się w dostarczonych plikach z szablonami.

Przykład

Program sprawdzający wykonuje następujące wywołanie:

```
restore_permutation(4, 16, 16).
```

Mamy $n = 4$*, a program może wykonać co najwyżej 16 zapisów („writes”) i 16 odczytów („reads”).*

Program zawodnika wykonuje następujące wywołania:

- `add_element("0001")`
- `add_element("0011")`
- `add_element("0100")`
- `compile_set()`
- `check_element("0001")` *zwraca* `false`
- `check_element("0010")` *zwraca* `true`
- `check_element("0100")` *zwraca* `true`
- `check_element("1000")` *zwraca* `false`
- `check_element("0011")` *zwraca* `false`
- `check_element("0101")` *zwraca* `false`
- `check_element("1001")` *zwraca* `false`
- `check_element("0110")` *zwraca* `false`

- `check_element("1010")` *zwraca* `true`
- `check_element("1100")` *zwraca* `false`

Jest tylko jedna permutacja zgodna z wynikami funkcji `check_element()`*, a mianowicie permutacja* $p = [2, 1, 3, 0]$*. Tak więc* `restore_permutation` *powinna zwrócić* `[2, 1, 3, 0]`*.*

Podzadania

Podzadanie 1 (20 punktów): $n = 8$, $w = 256$, $r = 256$, $p_i \neq i$ *zachodzi dla co najwyżej dwóch indeksów* i $(0 \leqslant i \leqslant n - 1)$.
Podzadanie 2 (18 punktów): $n = 32$, $w = 320$, $r = 1024$
Podzadanie 3 (11 punktów): $n = 32$, $w = 1024$, $r = 320$
Podzadanie 4 (21 punktów): $n = 128$, $w = 1792$, $r = 1792$
Podzadanie 5 (30 punktów): $n = 128$, $w = 896$, $r = 896$

Przykładowy program sprawdzający

Przykładowy program sprawdzający wczytuje dane w następującym formacie:

- *wiersz 1: liczby całkowite* n, w, r,
- *wiersz 2:* n *liczb całkowitych będących elementami* p.

XXII Bałtycka Olimpiada Informatyczna,

Helsinki, Finlandia, 2016

Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 2,5 s. *BOI 2016, dzień pierwszy, 12.05.2016*

Park

W stolicy Bajtlandii znajduje się prostokątny park ogrodzony płotem. Drzewa w parku oraz odwiedzający ten park będą w tym zadaniu reprezentowani za pomocą kół.

W parku są cztery bramy, po jednej w każdym rogu (brama 1: na dole po lewej, brama 2: na dole po prawej, brama 3: na górze po prawej, brama 4: na górze po lewej). Odwiedzający mogą wchodzić do parku i wychodzić z niego jedynie poprzez te bramy.

Odwiedzający park może wejść do parku lub wyjść z niego w chwili, gdy reprezentujące go koło dotyka obu boków rogu odpowiedniej bramy. Odwiedzający może dowolnie poruszać się po parku, ale reprezentujące go koło nie może nachodzić na żadne z drzew ani na płot.

Twoim zadaniem jest, dla każdego odwiedzającego park oraz dla danego numeru bramy, przez którą wszedł do parku, określić numery bram, przez które może on wyjść z parku.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite n i m: odpowiednio liczba drzew w parku oraz liczba odwiedzających park.

W drugim wierszu wejścia znajdują się dwie liczby całkowite w oraz h: szerokość oraz wysokość prostokąta opisującego park. Brama w lewym dolnym rogu ma współrzędne $(0, 0)$, natomiast ta w prawym górnym rogu znajduje się w punkcie (w, h).

Następnie dane jest n wierszy opisujące drzewa. Każdy wiersz składa się z trzech liczb całkowitych x, y oraz r: środek koła reprezentującego drzewo to (x, y), a jego promień wynosi r. Drzewa nie nachodzą na siebie nawzajem ani na płot.

W ostatnich m wierszach opisani są odwiedzający park. Każdy wiersz zawiera dwie liczby całkowite r oraz e: promień koła reprezentującego odwiedzającego oraz numer bramy, przez którą wejdzie do parku.

Dodatkowo, żadne drzewo nie nachodzi na kwadratowy obszar $2k \times 2k$ w żadnym rogu, gdzie k to maksymalny promień koła odwiedzającego park.

Wyjście

Dla każdego odwiedzającego park, Twój program powinien wypisać jeden wiersz zawierający numery bram, przez które odwiedzający może opuścić park, w kolejności rosnącej i bez spacji pomiędzy kolejnymi liczbami.

Uwagi

Dwa obiekty dotykają się, kiedy mają punkt wspólny. Dwa obiekty nachodzą na siebie, kiedy mają więcej niż jeden punkt wspólny.

Przykład

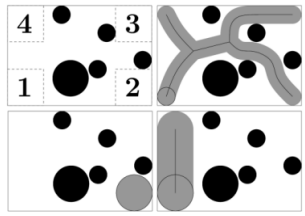
Dla danych wejściowych:

```
5 3
16 11
11 8 1
6 10 1
7 3 2
10 4 1
15 5 1
1 1
2 2
2 1
```

poprawnym wynikiem jest:

```
1234
2
14
```

Obrazek poniżej pokazuje lokalizacje bram oraz możliwe drogi każdego z odwiedzających park:



Podzadania

We wszystkich podzadaniach zachodzi $4k < w, h \leqslant 10^9$*, gdzie* k *jest maksymalnym promieniem koła odwiedzającego park.*

Podzadanie 1 (27 punktów)

- $1 \leqslant n \leqslant 2000$
- $m = 1$

Podzadanie 2 (31 punktów)

- $1 \leqslant n \leqslant 200$
- $1 \leqslant m \leqslant 10^5$

Podzadanie 3 (42 punkty)

- $1 \leqslant n \leqslant 2000$
- $1 \leqslant m \leqslant 10^5$

Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 1,5 s. *BOI 2016, dzień pierwszy, 12.05.2016*

Restrukturyzacja firmy

W firmie, w której zatrudnionych jest n pracowników, będzie przeprowadzana restrukturyzacja. Po jej zakończeniu struktura firmy ma mieć postać drzewa ukorzenionego, którego wierzchołki reprezentują pracowników firmy, a pracownik odpowiadający danemu wierzchołkowi jest bezpośrednim przełożonym pracowników odpowiadających synom tego wierzchołka.

Każdy pracownik przedstawił listę potencjalnych przełożonych, których może zaakceptować. Dodatkowo, należy określić pensje pracowników. Pensja musi być liczbą całkowitą dodatnią oraz pensja każdego pracownika musi być większa od sumy pensji jego bezpośrednich podwładnych.

Twoim zadaniem jest zrestrukturyzować firmę tak, aby wszystkie powyższe warunki były spełnione, a suma wszystkich pensji była jak najmniejsza.

Wejście

W pierwszym wierszu wejścia znajduje się liczba całkowita n: liczba pracowników. Pracownicy są ponumerowani kolejno liczbami $1, 2, \ldots, n$. Następnie wejście zawiera n wierszy, które opisują preferencje pracowników. Wśród tych wierszy, wiersz numer i zawiera liczbę całkowitą k_i, a następnie listę k_i liczb całkowitych. Lista ta zawiera wszystkich pracowników, których i-ty pracownik zaakceptuje jako swojego przełożonego.

Wyjście

Należy wypisać najmniejszą sumaryczną pensję spośród wszystkich sposobów przeprowadzenia restrukturyzacji firmy. Możesz założyć, że istnieje co najmniej jedno poprawne rozwiązanie.

Przykład

Dla danych wejściowych:

```
4
1 4
3 1 3 4
2 1 2
1 3
```

poprawnym wynikiem jest:

```
8
```

Podzadanie 1 (22 punkty)

- $2 \leqslant n \leqslant 10$
- $\sum_{i=1}^{n} k_i \leqslant 20$

Podzadanie 2 (45 punktów)

- $2 \leqslant n \leqslant 100$
- $\sum_{i=1}^{n} k_i \leqslant 200$

Podzadanie 3 (33 punkty)

- $2 \leqslant n \leqslant 5000$
- $\sum_{i=1}^{n} k_i \leqslant 10\,000$

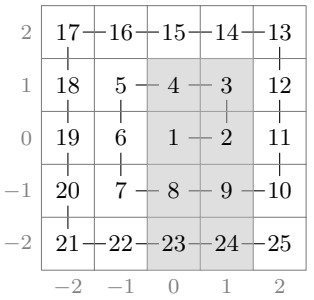
Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 1,5 s. *BOI 2016, dzień pierwszy, 12.05.2016*

Spirala

W pola tablicy o wymiarach $(2n+1)\times(2n+1)$ składającej się z jednostkowych, kwadratowych pól wpisano kolejne liczby naturalne. Liczbę 1 wpisano w środkowe pole, liczbę 2 wpisano w pole znajdujące się po prawej stronie pola z liczbą 1 i sąsiadujące z tym polem, a kolejne liczby wpisano w pola tablicy spiralnie, w kierunku odwrotnym do kierunku poruszania się wskazówek zegara (patrz rysunek).

Twoim zadaniem będzie udzielenie odpowiedzi na q pytań o sumę liczb w prostokątnej podtablicy tej tablicy (modulo 10^9+7). Dla przykładu, w tablicy poniżej mamy $n = 2$, a suma liczb w zacienionym prostokącie to 74:



Wejście

Pierwszy wiersz wejścia zawiera dwie liczby n oraz q: rozmiar tablicy i liczbę pytań.

W każdym z kolejnych q wierszy znajdują się cztery liczby całkowite x_1, y_1, x_2 oraz y_2 $(-n \leqslant x_1 \leqslant x_2 \leqslant n,\ -n \leqslant y_1 \leqslant y_2 \leqslant n)$. Taka czwórka oznacza, że i-te zapytanie dotyczy prostokąta, w którego rogach znajdują się pola o współrzędnych (x_1, y_1) i (x_2, y_2).

Wyjście

Dla każdego pytania wypisz w osobnym wierszu odpowiedź na to pytanie (modulo 10^9+7).

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`2 3`	`74`
`0 -2 1 1`	`9`
`-1 0 1 0`	`14`
`1 2 1 2`	

Podzadania

We wszystkich podzadaniach zachodzi $1 \leqslant q \leqslant 100$.

Podzadanie 1 (12 punktów)

- $1 \leqslant n \leqslant 1000$

Podzadanie 2 (15 punktów)

- $1 \leqslant n \leqslant 10^9$
- $x_1 = x_2$ *i* $y_1 = y_2$

Podzadanie 3 (17 punktów)

- $1 \leqslant n \leqslant 10^5$

Podzadanie 4 (31 punktów)

- $1 \leqslant n \leqslant 10^9$
- $x_1 = y_1 = 1$

Podzadanie 5 (25 punktów)

- $1 \leqslant n \leqslant 10^9$

Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

BOI 2016, dzień drugi, 13.05.2016

Labirynt

Uolevi napisał grę, w której gracz zbiera monety w labiryncie. Aktualnie problemem jest, że gra jest zbyt prosta. Czy możesz zaprojektować trochę labiryntów stanowiących wyzwanie dla grających w grę Uoleviego?

Każdy labirynt reprezentowany jest poprzez prostokątną tablicę zawierającą wolne pola (`.`) i ściany (`#`). Jedno pole jest bazą (`x`), a niektóre pola zawierają monety (`o`). Gracz rozpoczyna grę w bazie i może poruszać się w lewo, w prawo, w górę i w dół. Zadaniem gracza jest zebranie wszystkich monet i powrót do bazy.

Trudność labiryntu określana jest jako długość najkrótszej ścieżki, która rozpoczyna się w bazie, przechodzi przez pola z wszystkimi monetami, a na koniec wraca do bazy.

Wejście

Wejście rozpoczyna się pojedynczą liczbą całkowitą t: liczbą wymaganych labiryntów. Dalej następuje t wierszy. Każdy taki wiersz zawiera trzy liczby całkowite n, m oraz k. Oznaczają one, że rozmiar szukanego labiryntu wynosi $n \times m$ oraz że musi się w nim znaleźć dokładnie k monet.

Wyjście

Wyjście powinno zawierać t opisów labiryntów oddzielonych pustymi wierszami, w tej samej kolejności, w jakiej były podawane na wejściu. Każdy labirynt musi być rozwiązywalny.

Przykład

Dla danych wejściowych:

```
2
3 3 1
4 7 2
```

jednym z możliwych poprawnych wyników jest:

```
###
#.x
#o#

.o.####
.#..x.#
...##.#
###o...
```

Trudność pierwszego labiryntu wynosi 4, a drugiego 18.

Zgłoszenie

W tym zadaniu powinieneś zgłosić odpowiedni plik wyjściowy. Jest tylko jeden plik wejściowy, który podajemy poniżej (w dwóch kolumnach). Twój plik wyjściowy musi zawierać wszystkie labirynty wyszczególnione w pliku wejściowym.

```
50
10 18 2
11 14 3
9 17 4
8 9 6
5 11 4
10 20 5
9 10 8
7 6 9
12 20 6
12 20 12
8 14 8
11 18 4
14 5 7
12 11 4
18 6 6
9 17 6
10 13 4
8 13 6
12 12 5
11 5 5
13 12 11
13 13 6
7 15 8
5 5 4
12 9 12
7 17 10
12 16 2
10 10 4
10 20 6
19 11 10
14 13 6
17 13 8
7 19 1
8 16 8
14 9 12
13 9 2
16 5 2
7 10 1
13 17 6
18 7 11
16 13 1
16 13 12
11 12 3
15 9 5
13 19 12
5 17 1
8 16 8
6 6 10
20 13 2
11 7 3
```

Ocenianie

Dla każdego labiryntu Twoim wynikiem będzie $max(0, 100 - 3(d - x))$*, przy czym* x *jest trudnością Twojego labiryntu, a* d *jest trudnością najbardziej skomplikowanego labiryntu znalezionego przez jury. Twój całkowity wynik za zadanie będzie średnią wyników zaokrągloną w dół do najbliższej liczby całkowitej.*

Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 2 s. *BOI 2016, dzień drugi, 13.05.2016*

Miasta

W Bajtlandii jest n miast, spośród których pewne k to ważne miasta, często odwiedzane przez króla tej krainy.

W krainie jest m dróg łączących pewne pary miast. Są one w bardzo złym stanie, przez co król Bajtlandii nie może na nich rozwinąć prędkości maksymalnej swojego sportowego BMW z obawy przed pęknięciem opon.

Dla każdej drogi znany jest koszt jej naprawy. Twoim zadaniem jest wybrać, które z nich naprawić, tak aby między każdą parą ważnych miast było połączenie używające jedynie naprawionych dróg, a sumaryczny koszt naprawy był najmniejszy możliwy.

Wejście

W pierwszym wierszu wejścia znajdują się trzy liczby całkowite n, k i m: liczba miast, liczba ważnych miast oraz liczba dróg. Miasta są ponumerowane kolejnymi liczbami całkowitymi $1, 2, \ldots, n$.

Drugi wiersz wejścia zawiera k liczb całkowitych oznaczających numery ważnych miast.

Każdy z kolejnych m wierszy zawiera opis jednej drogi. Taki opis składa się z trzech liczb całkowitych a, b i c, oznaczających, że dana droga łączy miasta o numerach a i b, a koszt jej naprawy wynosi c.

Możesz założyć, że z każdego miasta da się dojechać do każdego innego.

Wyjście

W jedynym wierszu wyjścia Twój program powinien wypisać jedną liczbę całkowitą: minimalny koszt naprawy, która zapewnia, że wszystkie ważne miasta są ze sobą nawzajem połączone naprawionymi drogami.

Przykład

Dla danych wejściowych:

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

poprawnym wynikiem jest:

```
11
```

Podzadania

We wszystkich podzadaniach zachodzi $1 \leqslant c \leqslant 10^9$ *oraz* $n \geqslant k$.

Podzadanie 1 (22 punkty)

- $2 \leqslant k \leqslant 5$
- $n \leqslant 20$
- $1 \leqslant m \leqslant 40$

Podzadanie 2 (14 punktów)

- $2 \leqslant k \leqslant 3$
- $n \leqslant 10^5$
- $1 \leqslant m \leqslant 2 \cdot 10^5$

Podzadanie 3 (15 punktów)

- $2 \leqslant k \leqslant 4$
- $n \leqslant 1000$
- $1 \leqslant m \leqslant 2000$

Podzadanie 4 (23 punkty)

- $k = 4$
- $n \leqslant 10^5$
- $1 \leqslant m \leqslant 2 \cdot 10^5$

Podzadanie 5 (26 punktów)

- $k = 5$
- $n \leqslant 10^5$
- $1 \leqslant m \leqslant 2 \cdot 10^5$

Michał Glapa, Karol Kaszuba, Bartosz Kostka
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 1 s. *BOI 2016, dzień drugi, 13.05.2016*

Zamiany

Dany jest ciąg n liczb $x_1, x_2, \ldots, x_n$. Każda liczba $1, 2, \ldots, n$ występuje w tym ciągu dokładnie raz.

Możesz modyfikować ten ciąg, zamieniając pewne pary elementów podczas niektórych spośród $n-1$ tur ponumerowanych $k = 2, 3, \ldots, n$. W turze numer k możesz (ale nie musisz) zamienić miejscami liczby x_k i $x_{\lfloor k/2 \rfloor}$.

Ciąg $a_1, a_2, \ldots, a_n$ jest leksykograficznie mniejszy niż ciąg $b_1, b_2, \ldots, b_n$, jeśli istnieje indeks j ($1 \leqslant j \leqslant n$) taki że $a_k = b_k$ dla każdego $k < j$ i $a_j < b_j$.

Jaki jest najmniejszy leksykograficznie ciąg, który możesz otrzymać w wyniku tych zamian?

Wejście

W pierwszym wierszu znajduje się liczba całkowita n. W drugim wierszu wejścia znajduje się n liczb całkowitych oznaczających kolejne wyrazy ciągu x.

Wyjście

W jedynym wierszu wyjścia Twój program powinien wypisać n liczb całkowitych: najmniejszy leksykograficznie ciąg, który możesz otrzymać.

Przykład

Dla danych wejściowych:

```
5
3 4 2 5 1
```

poprawnym wynikiem jest:

```
2 1 3 4 5
```

Podzadanie 1 (10 punktów)

- $1 \leqslant n \leqslant 20$

Podzadanie 2 (11 punktów)

- $1 \leqslant n \leqslant 40$

Podzadanie 3 (27 punktów)

- $1 \leqslant n \leqslant 1000$

Podzadanie 4 (20 punktów)

- $1 \leqslant n \leqslant 5 \cdot 10^4$

Podzadanie 5 (32 punkty)

- $1 \leqslant n \leqslant 2 \cdot 10^5$

XXIII Olimpiada Informatyczna Europy Środkowej,

Piatra-Neamţ, Rumunia 2016

Kamil Dębowski, Dominik Klemba
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 1 s. *CEOI 2016, dzień pierwszy, 20.07.2016*

Astronauta

Juliusz – Pan Astronauta – rozpoczął właśnie staż na Międzynarodowej Stacji Kosmicznej (MSK). Jego zadaniem w trakcie stażu jest zbadanie cywilizacji Terran z planety Mar Sara. Terranie dotychczas wybudowali N miast, numerowanych od 1 do N, ale nie wybudowali jeszcze żadnych dróg pomiędzy nimi.

Gdy tylko Juliusz zaczął swój staż, cywilizacja Terran rozpoczęła budowę dróg na Mar Sara. Wiadomo, że Terranie muszą połączyć miasta w możliwie krótkim czasie, więc nie będą nigdy budować drogi łączącej miasta, między którymi istnieje już ścieżka (złożona z jednej lub wielu dróg). Za każdym razem, gdy Terranie budują nową drogę, Juliusz chce szybko dowiedzieć się, pomiędzy którymi dwoma miastami została ona wybudowana. Zależy mu, aby poznać odpowiedź, zanim wybudowana zostanie kolejna droga.

Szczęśliwie, Juliusz posiada dostęp do najnowszego satelity SETI, pozwalającego mu na analizę infrastruktury planety. Dla danych dwóch rozłącznych zbiorów miast SETI stwierdza, czy istnieje **bezpośrednia** *droga pomiędzy jakimś miastem z pierwszego zbioru oraz jakimś miastem z drugiego zbioru.*

Przebieg zdarzeń jest następujący:

1. *Terranie budują nową drogę.*
2. *Astronauta używa dobrodziejstw SETI, by oznaczyć dodaną drogę.*
3. *Udaje się on do MSK i podaje indeksy dwóch miast połączonych nową drogą.*
4. *Jeśli nie zostało jeszcze wybudowanych $N - 1$ dróg, wróć do punktu pierwszego.*

Napisz program rozwiązujący problem Astronauty.

Interakcja

Powinieneś zaimplementować funkcję `run()`*, która będzie wywołana raz, na początku działania programu. Z tej funkcji możesz wywołać funkcję* `query()` *za każdym razem, gdy chciałbyś użyć SETI. Za każdym razem, gdy określisz, która droga została wybudowana, powinieneś wywołać* `setRoad()`*. Możesz wołać* `query()` *wielokrotnie pomiędzy wywołaniami funkcji* `setRoad()`*.*

Funkcja biblioteki: `query()`

- *C/C++:* `int query(int size_a, int size_b, int a[], int b[]);`
- *Pascal:* `function query(size_a, size_b : longint, a, b : array of longint) : longint;`

Wywołuj tę funkcję, aby skorzystać z SETI. Argumenty `size_a`*,* `size_b` *powinny określać rozmiar zbiorów, o które pytamy. Tablice* `a[]`*,* `b[]` *powinny reprezentować zbiory (zawierać indeksy miast). Pamiętaj, że zbiory powinny być rozłączne! Funkcja zwraca 1, jeśli istnieje para miast z dwóch zbiorów połączona bezpośrednio drogą. W przeciwnym razie zwraca 0.*

Procedura biblioteki: `setRoad()`

- *C/C++:* `void setRoad(int a, int b);`
- *Pascal:* `procedure setRoad(a, b : longint);`

Użyj tej procedury, aby zakomunikować, iż odkryłeś nowo powstałą drogę pomiędzy miastami `a` *i* `b`*. Jeśli się pomylisz, otrzymasz zero punktów za ten test. W przeciwnym razie, wszystkie przyszłe wywołania* `query()` *będą uwzględniały tę drogę.*

Twoja procedura: `run()`

- *C/C++:* `void run(int N);`
- *Pascal:* `procedure run(N : longint);`

Twoje rozwiązanie musi implementować tę procedurę. W tej procedurze powinieneś wywoływać `query()` *oraz* `setRoad()`*. Przekazywany argument* `N` *reprezentuje liczbę miast wybudowanych przez Terran. Jeśli procedura zakończy się przed odkryciem* $\texttt{N} - 1$ *ścieżek, nie otrzymasz punktów za dany test.*

Uwagi dotyczące języka programowania

- *C/C++: dodaj* `#include "icc.h"`
- *Pascal: musisz zdefiniować* `unit icc` *oraz użyć* `uses graderhelplib`*.*

Punktacja

Zadanie jest podzielone na sześć podzadań. Testy należące do jednego podzadania będą posiadały tę samą wartość N*. Punkty za podzadanie otrzymasz, jeśli poprawnie odgadniesz wszystkie drogi w każdym teście i nie przekroczysz limitu* M *wywołań funkcji* `query()`*. Wartości* N *i* M *podane są wraz z punktami za podzadania w tabeli poniżej.*

Numer podzadania	N *= liczba miast*	M *= limit wywołań* `query()`	*Punkty*
1	*15*	*1500*	*7*
2	*50*	*2500*	*11*
3	*100*	*2250*	*22*
4	*100*	*2000*	*21*
5	*100*	*1775*	*29*
6	*100*	*1625*	*10*

Przykładowe wykonanie

Program zawodnika	*Sprawdzaczka*	*Wyjaśnienie*
`-`	`run(4)`	*Jest $N = 4$ miast. Pierwsza droga powstaje pomiędzy miastami 2 i 4 (uczestnik nie dysponuje tą informacją).*
`query(1, 3, {1}, {2, 3, 4})`	`return 0`	*Zapytanie o zbiory $\{1\}$ i $\{2, 3, 4\}$. Odpowiedź to 0: nie istnieje bezpośrednia droga do miasta 1 z żadnego miasta z drugiego zbioru.*
`query(1, 2, {2}, {3, 4})`	`return 1`	*Dla zbiorów $\{2\}$ i $\{3, 4\}$ odpowiedzią jest 1, bo istnieje droga z miasta 2 do miasta 4.*
`query(1, 1, {2}, {3})`	`return 0`	
`setRoad(2, 4)`	`-`	*Droga została poprawnie określona jako $(2, 4)$. Nowa droga powstaje pomiędzy miastami 1 i 3.*
`query(2, 2, {2, 4}, {1, 3})`	`return 0`	*Zapytanie o zbiory $\{2, 4\}$ i $\{1, 3\}$. Odpowiedź to 0.*
`setRoad(1, 3)`	`-`	*Poprawne określenie drogi $(1, 3)$. Nowa droga jest budowana pomiędzy miastami 1 i 4.*
`query(2, 2, {2, 4}, {1, 3})`	`return 1`	*Pytanie jak powyżej. Tym razem odpowiedź to 1 ze względu na nowo powstałą drogę.*
`query(1, 2, {2}, {1, 3})`	`return 0`	
`query(1, 1, {4}, {3})`	`return 0`	
`setRoad(4, 1)`	`exit`	*Ostatnia droga $(4, 1)$ została poprawnie określona. Program otrzymuje pełną punktację za ten test.*

Kamil Dębowski, Dominik Klemba
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 0,2 s. *CEOI 2016, dzień pierwszy, 20.07.2016*

Kangur

Ogrodnicy nie cierpią kangurów, gdyż te wyżerają im marchew. Rozważmy ogród w kształcie długiego paska złożonego z N pól, ponumerowanych od 1 do N. Początkowo każde pole zawiera jedną marchewkę.

W polu o numerze x pojawił się kangur. W celu zjedzenia wszystkich marchewek ma zamiar odwiedzić wszystkie pola, każde dokładnie raz. Jest on w stanie skoczyć z dowolnego pola na dowolne inne pole. Chce on skończyć na polu o numerze y. Oczywiście, musi on wykonać dokładnie $N - 1$ skoków.

Kangur nie chce zostać złapany, więc z każdym skokiem musi on zmieniać kierunek. Innymi słowy, nie może on skoczyć dwa razy z rzędu w prawo (z pola a do pola b, po czym z b do c, gdzie $a < b < c$) oraz nie może on skoczyć dwa razy z rzędu w lewo (z a do b, po czym z b do c, gdzie $c < b < a$).

Znając liczbę pól N, początkowe pole x oraz końcowe pole y, znajdź liczbę różnych możliwych tras kangura.

Wejście

Plik wejściowy `kangaroo.in` *będzie zawierał trzy liczby całkowite N, x, y, oddzielone odstępami.*

Wyjście

Do pliku wyjściowego `kangaroo.out` *powinieneś wypisać jedną liczbę całkowitą – liczbę różnych możliwych tras kangura modulo 1 000 000 007.*

Limity i uwagi

- $2 \leqslant N \leqslant 2000$
- $1 \leqslant x \leqslant N$
- $1 \leqslant y \leqslant N$
- $x \neq y$
- *W testach wartych 6 punktów zachodzi $N \leqslant 8$.*
- *W testach wartych 36 punktów zachodzi $N \leqslant 40$.*
- *W testach wartych 51 punktów zachodzi $N \leqslant 200$.*
- *Każda trasa jest jednoznacznie wyznaczona przez kolejność odwiedzania pól.*

- *Jest zagwarantowane, że w każdym teście istnieje co najmniej jedna możliwa trasa kangura.*
- *Kangur może wykonać pierwszy skok (z pola x) w dowolnym kierunku.*

Przykład

Dla pliku wejściowego `kangaroo.in`*:*

```
4 2 3
```

poprawnym wynikiem jest plik wyjściowy `kangaroo.out`*:*

```
2
```

Wyjaśnienie do przykładu: *Kangur zaczyna z pola 2 i musi skończyć na polu 3. Dwie możliwe trasy to* $2 \to 1 \to 4 \to 3$ *oraz* $2 \to 4 \to 1 \to 3$.

Kamil Dębowski, Dominik Klemba
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 0,6 s. *CEOI 2016, dzień pierwszy, 20.07.2016*

Sztuczka

Grupa turystów, która spokojnie zwiedzała zamek w Branie, została schwytana przez Hrabiego Drakulę. Szczęśliwie, do grupy należy magik i jego dwaj asystenci. Są oni teraz ostatnią nadzieją schwytanych. Magik postanowił wykonać przed Drakulą niesamowitą sztuczkę. Jeśli uda się zachwycić Hrabiego Drakulę, ten wypuści schwytanych.

Po rozpoczęciu sztuczki asystenci nie mogą komunikować się z magikiem ani ze sobą nawzajem. Magik wręczy Drakuli talię $2N + 1$ kart, ponumerowanych od 0 do $2N$. Hrabia wybierze i schowa jedną kartę. Następnie Hrabia wybierze N z pozostałych $2N$ kart i wręczy wybrane N kart pierwszemu asystentowi, a pozostałe N kart drugiemu asystentowi. Każdy z asystentów wybierze dwie ze swoich N kart i wręczy je magikowi, jedna po drugiej. Wtedy, znając tylko dwie kolejne karty wręczone przez pierwszego asystenta i dwie kolejne karty wręczone przez drugiego asystenta, magik zgadnie kartę schowaną przez Hrabiego Drakulę.

Pomóż magikowi i nie pozwól na to, by turyści stali się kolacją wampira!

Twój program zostanie uruchomiony trzy razy dla każdego testu. Podczas pierwszego uruchomienia Twój program przejmie rolę pierwszego asystenta. Podczas drugiego uruchomienia Twój program przejmie rolę drugiego asystenta. Podczas trzeciego uruchomienia Twój program przejmie rolę magika.

Wejście

Pierwszy wiersz pliku `trick.in` *będzie zawierał jedną liczbę całkowitą T, oznaczającą liczbę przypadków testowych (sztuczka ma być wykonana tyle razy w tym teście). Drugi wiersz będzie zawierał jedną liczbę R należącą do zbioru $\{1, 2, 3\}$, określającą rolę, którą przejmie Twój program we wszystkich przypadkach testowych w tym teście.*

Jeśli $R = 1$, Twój program przejmie rolę pierwszego asystenta. Jeśli $R = 2$, Twój program przejmie rolę drugiego asystenta. W obu przypadkach, wiersz o numerze $2i + 1$ ($1 \leqslant i \leqslant T$) będzie zawierał liczbę całkowitą N_i, mówiącą, że w i-tym przypadku testowym talia będzie składała się z $2N_i + 1$ kart. Wiersz o numerze $2i + 2$ ($1 \leqslant i \leqslant T$) będzie zawierał N_i liczb całkowitych, opisujących zbiór kart otrzymanych przez kontrolowanego asystenta w i-tym przypadku testowym.

Jeśli $R = 3$, Twój program przejmie rolę magika. Wiersz o numerze $2i + 1$ ($1 \leqslant i \leqslant T$) będzie zawierał liczbę N_i oznaczającą to samo co w poprzednim akapicie. Wiersz o numerze $2i + 2$ ($1 \leqslant i \leqslant T$) będzie zawierał cztery liczby całkowite – dwie liczby wypisane przez Twój program dla $R = 1$ oraz dwie liczby wypisane przez Twój program dla $R = 2$ (są to oczywiście liczby wypisane w i-tym przypadku testowym). Te dwie pary liczb podane zostaną w tej samej kolejności, w jakiej wypisał je Twój program dla $R = 1$ i $R = 2$.

Wyjście

Jeśli $R = 1$ lub $R = 2$, to Twój program powinien wypisać w i-tym wierszu ($1 \leqslant i \leqslant T$) pliku wyjściowego `trick.out` *dwie (oddzielone odstępem) liczby całkowite, reprezentujące dwie*

karty wybrane przez danego asystenta i wręczone magikowi w i-tym przypadku testowym. Te dwie liczby muszą być różne i muszą należeć do zbioru N_i liczb danych w pliku wejściowym.

Jeśli $R = 3$, Twój program powinien w i-tym wierszu ($1 \leqslant i \leqslant T$) wypisać jedną liczbę całkowitą, reprezentującą kartę schowaną przez Hrabiego Drakulę w i-tym przypadku testowym.

Limity

- $6 \leqslant N_i \leqslant 1\,234\,567$ $(1 \leqslant i \leqslant T)$
- $S_N \leqslant 1\,234\,567$, gdzie $S_N = N_1 + N_2 + \ldots + N_T$
- *W testach wartych 29 punktów zachodzi* $N_i = 6$ $(1 \leqslant i \leqslant T)$.
- *W innych testach wartych 19 punktów zachodzi* $6 \leqslant N_i \leqslant 30$ $(1 \leqslant i \leqslant T)$ *oraz* $S_N \leqslant 123\,456$.
- *W jeszcze innych testach wartych 30 punktów zachodzi* $6 \leqslant N_i \leqslant 500$ $(1 \leqslant i \leqslant T)$, $S_N \leqslant 123\,456$ *oraz będzie co najwyżej 10 przypadków testowych z* $N_i > 50$.

Przykład

Dla pliku wejściowego `trick.in`*:*

```
2
1
6
6 1 2 5 7 10
6
9 8 2 0 4 6
```

możliwym wynikiem jest plik wyjściowy `trick.out`*:*

```
1 2
8 4
```

dla pliku wejściowego `trick.in`*:*

```
2
2
6
3 0 4 9 12 8
6
7 1 11 10 3 5
```

możliwym wynikiem jest plik wyjściowy `trick.out`*:*

```
4 3
1 3
```

dla pliku wejściowego `trick.in`*:*

```
2
3
6
1 2 4 3
6
8 4 1 3
```

poprawnym wynikiem jest plik wyjściowy `trick.out`*:*

```
11
12
```

Wyjaśnienie do przykładu: *Przedstawione zostały trzy wywołania programu dla tego samego testu, zawierającego dwa przypadki testowe.*

W pierwszym przypadku testowym, pierwszy asystent otrzymuje karty 1, 2, 5, 6, 7 i 10 i wręcza magikowi karty 1 i 2, w tej kolejności. Drugi asystent otrzymuje karty 0, 3, 4, 8, 9 i 12 i wręcza magikowi karty 4 i 3, w tej kolejności. Wtedy magik stwierdza, że kartą schowaną przez Hrabiego Drakulę jest 11.

Kamil Dębowski, Dominik Klemba
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 0,1 s. *CEOI 2016, dzień drugi, 22.07.2016*

Wyrażenie nawiasowe

Poprawnym wyrażeniem nawiasowym jest:

- *ciąg pusty;*
- *ciąg* `(`B`)`*, gdzie B jest poprawnym wyrażeniem nawiasowym;*
- *LR – sklejenie dwóch poprawnych wyrażeń nawiasowych L, R.*

Niech B będzie poprawnym wyrażeniem nawiasowym długości N. i-ty element ciągu B oznaczmy jako B_i. Dla dwóch indeksów i, j (gdzie $1 \leqslant i < j \leqslant N$) powiemy, że B_i, B_j są dopasowanymi nawiasami wtedy i tylko wtedy, gdy spełnione są następujące warunki:

- *$B_i =$* `(`*, $B_j =$* `)` *oraz*
- *$i = j - 1$ lub $C = B_{i+1}B_{i+2}\ldots B_{j-1}$ jest poprawnym wyrażeniem nawiasowym.*

Przyjmijmy, że S jest ciągiem małych liter alfabetu angielskiego. i-ty element ciągu S oznaczmy jako S_i. Powiemy, że poprawne wyrażenie nawiasowe B pasuje do S, jeśli:

- *B i S są tej samej długości oraz*
- *dla dowolnej pary indeksów i oraz j, gdzie $i < j$, jeżeli B_i i B_j są dopasowanymi nawiasami, to $S_i = S_j$.*

Dla danego ciągu S, składającego się z N małych liter alfabetu angielskiego, znajdź leksykograficznie najmniejsze poprawne wyrażenie nawiasowe, które pasuje do S, albo wypisz -1, jeżeli takie wyrażenie nie istnieje.

Wejście

Plik wejściowy `match.in` *zawiera ciąg N małych liter alfabetu angielskiego S.*

Wyjście

Do pliku wyjściowego `match.out` *powinieneś wypisać albo ciąg składający się z N znaków, który przedstawia najmniejsze leksykograficznie poprawne wyrażenie nawiasowe pasujące do danego ciągu S, albo -1, jeżeli takie wyrażenie nie istnieje.*

Limity oraz inne informacje

- *$2 \leqslant N \leqslant 100\,000$*
- *W testach wartych łącznie 10 punktów zachodzi $N \leqslant 18$.*

- *W innych testach wartych łącznie 27 punktów zachodzi* $N \leqslant 2000$.
- *Jeśli* A *i* B *są poprawnymi wyrażeniami nawiasowymi długości* N*, to powiemy, że wyrażenie* A *jest leksykograficznie mniejsze od wyrażenia* B*, jeżeli istnieje indeks* i $(1 \leqslant i \leqslant N)$*, taki że* $A_j = B_j$ *dla każdego* $j < i$*, oraz* $A_i < B_i$.
- *Znak* `(` *jest uznawany za mniejszy leksykograficznie niż znak* `)`.

Przykład

Dla pliku wejściowego `match.in`*:*

```
abbaaa
```

poprawnym wynikiem jest plik wyjściowy `match.out`*:*

```
(()())
```

Wyjaśnienie do przykładu: *Innym pasującym wyrażeniem nawiasowym mogłoby być* `(())()`*, ale nie jest ono najmniejsze leksykograficznie.*

Dla pliku wejściowego `match.in`*:*

```
abab
```

poprawnym wynikiem jest plik wyjściowy `match.out`*:*

```
-1
```

Wyjaśnienie do przykładu: *Nie istnieje poprawne wyrażenie nawiasowe pasujące do tego ciągu.*

Kamil Dębowski, Dominik Klemba
Tłumaczenie

Dostępna pamięć: 256 MB, czas: 2.5 s. *CEOI 2016, dzień drugi, 22.07.2016*

Popeala

Rumuńskie słowo 'popeală' pochodzi z rumuńskiej historycznej noweli „Alexandru Lăpuşneanul", w której tytułowy Książę Mołdawii używa wariacji tego słowa do opisania swojej nadchodzącej zemsty na uzurpatorach. Wyrażenie ostatnio wróciło niespodziewanie do łask w kontekście konkursów programistycznych. Jest używane do opisania każdej sytuacji, w której organizatorzy utrudniają życie zawodnikom w nietypowy i (zazwyczaj) niezamierzony sposób: bardzo restrykcyjne limity, niepoprawne testy, błędne stwierdzenia, literówki i inne takie wszelakie. To zadanie dotyczy właśnie takiego... popeală.

Rozważmy konkurs programistyczny, w którym bierze udział N uczestników. Uczestnicy mają do rozwiązania jedno zadanie sprawdzane za pomocą T testów. Organizatorzy chcą pogrupować testy w co najwyżej S grup.

Jak działają grupy: *każdy test należy do* **dokładnie jednej** *grupy. Grupa może zawierać dowolną dodatnią liczbę testów. Jeśli program zawodnika nie zaliczy* **jakiegokolwiek** *testu z danej grupy, to otrzyma za tę grupę 0 punktów. W przeciwnym przypadku zdobędzie liczbę punktów równą sumie punktów za testy w tej grupie.*

Organizatorzy są złośliwi, więc chcą pogrupować testy po zakończeniu konkursu. Dla każdego zawodnika wiedzą, które testy jego program rozwiązał poprawnie, i chcą **zminimalizować łączną liczbę punktów** *zdobytych przez wszystkich zawodników.*

Formalnie: Dana jest tablica liczb całkowitych Points[] rozmiaru T. i-ty test warty jest Points[i] punktów. Dana jest również dwuwymiarowa tablica Results[][] rozmiaru $N \cdot T$. Results[i][j] jest równe 1 wtedy i tylko wtedy, gdy i-ty zawodnik poprawnie rozwiązał j-ty test. W przeciwnym przypadku to pole będzie równe 0. Organizatorzy zadecydowali, że grupy będą zawierały **spójne przedziały testów**. *Innymi słowy, jeżeli testy X i Y będą w tej samej grupie, to każdy test Z ($X \leqslant Z \leqslant Y$) będzie znajdował się w grupie z nimi.*

Masz złe serce, więc chcesz pomóc organizatorom. Chcą oni wiedzieć, dla każdej wartości $1 \leqslant K \leqslant S$, jaka jest minimalna łączna liczba punktów zdobytych przez zawodników w konkursie, jeżeli organizatorzy zdecydują się podzielić testy na K grup.

Wejście

Plik wejściowy `popeala.in` *w pierwszym wierszu zawiera trzy oddzielone odstępami dodatnie liczby całkowite N, T, S. Drugi wiersz zawiera T oddzielonych odstępami dodatnich liczb całkowitych, reprezentujących kolejne elementy tablicy Points[]. Kolejne N wierszy zawiera binarne ciągi długości T, reprezentujące wiersze tablicy Results[][].*

Wyjście

Plik wyjściowy `popeala.out` *powinien składać się z S wierszy. i-ty z nich powinien zawierać jedną liczbę całkowitą: minimalną* **łączną** *liczbę punktów możliwą do uzyskania przez zawodników, jeżeli organizatorzy zdecydują się podzielić testy na i grup.*

Limity i inne informacje

- $1 \leqslant T \leqslant 20\,000$
- $1 \leqslant N \leqslant 50$
- $1 \leqslant S \leqslant min(50, T)$
- $1 \leqslant Points[i] \leqslant 10\,000$, dla każdego $1 \leqslant i \leqslant T$.
- $(Points[1] + Points[2] + \ldots + Points[T]) \cdot N \leqslant 2\,000\,000\,000$
- W testach wartych 8 punktów zachodzi $T \leqslant 40$.
- W innych testach wartych 9 punktów zachodzi $40 < T \leqslant 500$.
- W jeszcze innych testach wartych 9 punktów zachodzi $500 < T \leqslant 4000$.

Przykład

Dla pliku wejściowego `popeala.in`*:*	*poprawnym wynikiem jest plik wyjściowy* `popeala.out`*:*

`2 3 3`
`4 3 5`
`101`
`110`
	`0`
	`8`
	`16`

Wyjaśnienie do przykładu: *Jest $N = 2$ zawodników, $T = 3$ testów oraz najwyżej $S = 3$ grup. Tablica $Points[]$ to $[4, 3, 5]$.*

W przypadku pojedynczej grupy łączna liczba punktów będzie równa 0, ponieważ żaden z zawodników nie rozwiązał wszystkich testów poprawnie (wszystkie testy muszą być w jednej grupie).

W przypadku dwóch grup są dwa sposoby podzielenia testów. Jeden z nich daje łączną liczbę punktów równą 12, drugi skutkuje łączną liczbą punktów 8. Ponieważ chcemy zminimalizować wynik, wybieramy tę drugą wartość.

Kamil Dębowski, Dominik Klemba
Tłumaczenie

CEOI 2016, dzień drugi, 22.07.2016

Ruter

Adam i Bartek zostali zatrudnieni przez firmę informatyczną z miasta Piatra Neamt. Ich pierwszym projektem jest stworzenie nowego rodzaju rutera, niesamowitego **Connect Ethernet Operating Interface 2016**. *Ruter powinien składać się z:*

- N *wierzchołków wejściowych, ponumerowanych od 1 do* N*;*
- N *wierzchołków wyjściowych, ponumerowanych od* $N + 1$ *do* $2 \cdot N$*;*
- K *wierzchołków wewnętrznych, ponumerowanych od* $2 \cdot N + 1$ *do* $2 \cdot N + K$*;*
- M *skierowanych krawędzi między parami różnych wierzchołków.*

Wierzchołek X *wysyła dane do wierzchołka* Y *(czyli* Y *otrzymuje dane z* X*), jeśli:*

- $X = Y$ *lub*
- *istnieje wierzchołek* Z*, taki że* X *wysyła dane do* Z *oraz istnieje bezpośrednia krawędź z wierzchołka* Z *do wierzchołka* Y*.*

Jeśli wierzchołek X *wysyła dane do innego wierzchołka* Y*, to definiujemy* **ścieżkę danych** *z* X *do* Y *jako zbiór bezpośrednich krawędzi* $\{(A_1, A_2), (A_2, A_3), \ldots, (A_{L-1}, A_L)\}$ *dla pewnego* $L \geqslant 2$*, taki że* $A_1 = X$ *i* $A_L = Y$*.*

Ruter działa poprawnie, jeśli:

- *każdy wierzchołek wejściowy wysyła dane do każdego wierzchołka wyjściowego;*
- *każdy wierzchołek wejściowy otrzymuje dane tylko od siebie samego;*
- *każdy wierzchołek wyjściowy wysyła dane tylko do siebie samego;*
- *dla każdych dwóch różnych wierzchołków* X *i* Y*, jeśli* X *wysyła dane do* Y*, to* Y *nie wysyła danych do* X*;*
- *dla każdych dwóch różnych wierzchołków* X *i* Y*, jeśli* X *wysyła dane do* Y*, to istnieje tylko jedna ścieżka danych z* X *do* Y*. W szczególności, każde dwa wierzchołki* X *i* Y *są połączone co najwyżej jedną bezpośrednią krawędzią.*

Jak każde urządzenie elektroniczne, ruter potrzebuje prądu. Moc potrzebna do funkcjonowania wierzchołka X *jest określona jako* $P_X = IN_X \cdot OUT_X$*, gdzie* IN_X *jest liczbą wierzchołków wejściowych wysyłających dane do* X*, natomiast* OUT_X *jest liczbą wierzchołków wyjściowych otrzymujących dane z* X*. Maksymalną mocą rutera nazywamy*

$$P_{max} = max(P_1, P_2, \ldots, P_{2 \cdot N+K}).$$

Przełożony dostarczył Adamowi i Bartkowi specyfikacje techniczne kilku ruterów testowych, które umieściliśmy w tabeli na następnej stronie. Dla każdej specyfikacji przełożony żąda projektu rutera, który:

- *ma dokładnie N wierzchołków wejściowych i N wierzchołków wyjściowych;*
- *ma co najwyżej M_{lim} bezpośrednich krawędzi;*
- *używa maksymalnej mocy co najwyżej P_{lim};*
- *ma łącznie co najwyżej 500 000 wierzchołków (czyli $N_{total} = 2 \cdot N + K \leqslant 500\,000$).*

Numer testu	N	M_{lim}	P_{lim}	*Punkty*
1	*118*	*1 000 000*	*1 000 000*	*4*
2	*223*	*1 000 000*	*1 000 000*	*5*
3	*1250*	*500 000*	*500 000*	*6*
4	*5101*	*500 000*	*500 000*	*6*
5	*9934*	*500 000*	*500 000*	*26*
6	*9955*	*500 000*	*100 000*	*30*
7	*9978*	*100 000*	*100 000*	*23*

Adam i Bartek dostaną pewną liczbę punktów za każdy poprawnie zbudowany ruter, co wyspecyfikowano w ostatniej kolumnie powyższej tabeli.

Wejście

W tym zadaniu nie powinieneś wysyłać swojego programu. W archiwum pobranym ze strony konkursu znajdziesz pliki `1-router.in`, `2-router.in`, *...*, `7-router.in`. *Pliki te zawierają dane wejściowe dla każdego z testów.*

Każdy z plików wejściowych `1-router.in`, `2-router.in`, *...*, `7-router.in` *opisuje pojedynczy test. Plik zawiera w jednym wierszu trzy liczby całkowite oddzielone odstępami: N (liczbę wierzchołków wejściowych i jednocześnie liczbę wierzchołków wyjściowych), M_{lim} (maksymalną liczbę bezpośrednich krawędzi) oraz P_{lim} (ograniczenie na maksymalną moc rutera).*

Wyjście

Dla każdego pliku wejściowego powinieneś stworzyć odpowiedni plik wyjściowy `1-router.out`, `2-router.out`, *...*, `7-router.out`. *Umieść wszystkie te pliki w folderze nazwanym* `router-out` *i stwórz archiwum* `zip` *zawierające ten katalog. Jako rozwiązanie powinieneś wysłać właśnie to archiwum.*

W każdym z plików wyjściowych `1-router.out`, `2-router.out`, *...*, `7-router.out` *wypisz dwie liczby całkowite (oddzielone odstępem): $N_{total} = 2 \cdot N + K$ (łączna liczba wierzchołków rutera) oraz M (liczba bezpośrednich krawędzi). W każdym z następnych M wierszy wypisz dwie liczby całkowite X i Y, oznaczające bezpośrednią krawędź z wierzchołka X do wierzchołka Y.*

Pomocnicze skrypty

W pobranym archiwum znajdziesz też dwa skrypty `gen-out.sh` *i* `check.sh`, *a także plik wykonywalny* `verif_contestant`. *Jeśli umieścisz te trzy pliki razem z plikami wejściowymi i Twoim plikiem wykonywalnym* `router` *w jednym katalogu, będziesz mógł użyć komendy* `bash gen-out.sh`, *by wygenerować pliki wyjściowe stworzone przez Twój plik wykonywalny dla każdego*

z plików wejściowych. Możesz wtedy użyć komendy `bash check.sh`*, by sprawdzić poprawność wygenerowanych plików wyjściowych dla każdego testu. Plik wykonywalny* `router` *powinien powstać przez skompilowanie Twojego programu, który czyta wejście z pliku* `router.in` *i wypisuje wyjście do pliku* `router.out`*.*

Przykład

Dla pliku wejściowego `router.in`*:*

```
3 100 200
```

poprawnym wynikiem jest plik wyjściowy `router.out`*:*

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

Wyjaśnienie do przykładu: *Adam i Bartek muszą skonstruować ruter z trzema wierzchołkami wejściowymi i trzema wierzchołkami wyjściowymi. Ruter powinien zawierać co najwyżej 100 bezpośrednich krawędzi, a jego maksymalna moc nie powinna być większa niż 200.*

Adam i Bartek użyli łącznie 9 wierzchołków:

- *wierzchołki wejściowe 1, 2 i 3;*
- *wierzchołki wyjściowe 4, 5 i 6;*
- *wierzchołki wewnętrzne 7, 8 i 9.*

Użyli też 8 bezpośrednich krawędzi.

Maksymalna moc rutera to 9. Taką moc ma wierzchołek 8, który:

- *otrzymuje dane z $IN_8 = 3$ wierzchołków wejściowych;*
- *wysyła dane do $OUT_8 = 3$ wierzchołków wyjściowych.*

Dla pliku wejściowego `router.in`*:*

```
3 100 200
```

poprawnym wynikiem jest także plik wyjściowy `router.out`*:*

```
6 9
1 4
1 5
1 6
2 4
2 5
2 6
3 4
3 5
3 6
```

Wyjaśnienie do przykładu: *Dla tej samej zadanej specyfikacji technicznej inny poprawny ruter zawiera jedynie 6 wierzchołków (3 wejściowe i 3 wyjściowe).*

Maksymalna moc tego rutera to 3: każdy wierzchołek wejściowy otrzymuje dane tylko od siebie samego oraz wysyła dane do wszystkich do wszystkich trzech wierzchołków wyjściowych. Podobnie, każdy wierzchołek wyjściowy otrzymuje dane z wszystkich trzech wierzchołków wejściowych i wysyła dane tylko do siebie samego.