Obóz przed II etapem w Krzyżowej

Bartosz Chomiński Anadi Agrawal Michał Kępa Marcin Knapik
Artur Kraska Marcel Szelwiga Jan Wańkowicz Adam Zyzik

16–20 stycznia 2023

Spis treści

Część I

Grupa przedfinalistów

1 Uczestnicy

Uczestnicy zajęć: Weronika Bala, Alex Basałaj, Jagoda Białek, Adam Biel, Artur Bieniek, Maciej Boroń, Justyna Borowiak, Maria Cichosz, Wojciech Cieślik, Wojciech Domin, Mateusz Feczan, Piotr Furman, Antoni Goles, Patryk Górski, Maja Hryń, Amelia Jochna, Kacper Jodłowski, Mateusz Jurach, Tomasz Kośnikowski, Jan Kuffel, Franciszek Kurlej, Filip Latuszek, Hubert Łukasik, Karol Musieliński, Jan Myszka, Nhat Quang Nguyen Doan, Magdalena Nieplowicz, Mikołaj Niezbrzycki, Igor Nowicki, Anna Olejniczak, Bartosz Olizarowski, Magda Pawicka, Szymon Pawlicki, Mikołaj Pesta, Franciszek Pietrusiak, Igor Ptak, Anna Roczek, Justyna Rojek-Nowosielska, Marcin Rolbiecki, Wojciech Rybak, Filip Sobieraj, Wojciech Sobiński, Damian Sosulski, Adam Sowiński, Jakub Stachowicz, Bartłomiej Stefański, Łukasz Stodółka, Szymon Szczygłowski, Dawid Szkudlarski, Michał Szwejda, Jakub Szydło, Bartosz Tarnowski, Tomasz Uram, Maciej Wajda, Kuba Walęga, Mateusz Wawrzyniak, Piotr Węgrzyn, Mikołaj Wieczorek, Małgorzata Wróbel, Mikołaj Wrzesiński, Hanna Zając, Jan Zbrocki, Cyprian Ziółkowski i Karol Żeleźnik.

Zajęcia prowadzili Bartosz Chomiński, Michał Kępa, Artur Kraska, Marcel Szelwiga, Jan Wańkowicz i Adam Zyzik.

2 Treści zadań

Domek z kart **(dom)**

Memory limit: 32 MB Time limit: 2.00 s

Jak to zawsze bywa na obozach, podczas gdy wszyscy uczestnicy świetnie się bawią podczas rozwiązywania pięciogodzinnych zawodów, kadra po prostu się nudzi. Nikt z zawodników nie zadaje pytań, mało kto wysyła jakiekolwiek rozwiązania, a wszystkie zadania na kolejne dni obozu są już przygotowane i przetestowane. Z tego powodu kadrowiczom przychodzą do głowy najróżniejsze pomysły na rozerwanie się.

Pewnego razu, gdy Marcel nie mógł już wytrzymać z nudów, Bartek pożyczył mu coś, co jego zdaniem dostarcza najlepszej możliwej rozrywki – talię kart, zawierających liczby całkowite. Marcel, nie mając pojęcia do czego tak właściwie mogą one służyć, szybko znalazł dla nich nowe zastosowanie – zaczął budować z nich domek z kart. Robił to w następujący sposób: w pierwszej kolejności opierał karty parami o siebie, następnie na powstałych szczytach stawiał kolejne karty, znów opierając je parami o siebie, i tak dalej. Okazało się, że na każdym piętrze, poza ostatnim, liczba szczytów była parzysta, więc zawsze dało się poprawnie zbudować wyższe piętro.

Po zbudowaniu całego domku, Marcel zaczął się zastanawiać co teraz z nim zrobić. Przyszło mu na myśl, żeby wykorzystać wartości znajdujące się na kartach. Chciałby on teraz zdjąć z domku nie więcej niż K kart w taki sposób, aby suma ich wartości była jak największa. Oczywiście domek z kart nie może się przy tym zawalić!

Aby po wyjęciu kart domek nadal był stabilny, Marcel nie może nigdy zdjąć pojedynczej karty, nie wyjmując zarazem jej pary (tzn. tej karty, z którą nawzajem się podpierają). Ponadto nigdy nie może zdjąć karty, nie zdjąwszy wcześniej wszystkich kart, które są wyżej od niej i pośrednio lub bezpośrednio są o nią oparte.

Czy jesteś w stanie pomóc Marcelowi?

Napisz program, który dla danego ustawienia kart policzy największą liczbę punktów jaką jest w stanie zdobyć Marcel, zabierając z domku K kart w taki sposób, żeby się on nie zawalił.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite N i K oddzielone pojedynczym odstępem, oznaczające liczbę pięter karcianego domku i maksymalną liczbę kart, które Marcel może zdjąć.

Kolejne N wierszy wejścia zawiera opisy poszczególnych pięter domku, od najwyższego do najniższego. W $(i+1)$-szym wierszu znajduje się 2^i liczb całkowitych $A_{i,1}, A_{i,2}, \ldots, A_{i,2^i}$ pooddzielanych pojedynczymi odstępami, oznaczających wartości kart na i-tym piętrze od góry, w kolejności od lewej do prawej.

Wyjście

Twój program powinien wypisać na standardowe wyjście dokładnie jedną liczbę całkowitą – maksymalną sumę wartości kart, jaką Marcel może uzyskać, zdejmując maksymalnie K kart z domku, tak aby ten się nie zawalił.

Ograniczenia

$2 \leq N \leq 17$, $2 \leq K \leq 40$, K jest parzyste, dla każdych i, j zachodzi $-1\,000\,000 \leq A_{i,j} \leq 1\,000\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 4$, $K \leq 10$	20
2	$N \leq 8$, $K \leq 30$	30
3	brak dodatkowych ograniczeń	50

Przykład

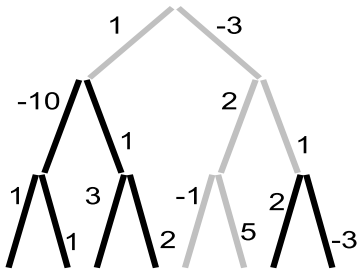
Input

```
3 6
1 -3
-10 1 2 1
1 1 3 2 -1 5 2 -3
```

Output

```
5
```

Explanation



Karty, które Marcel powinien zdjąć z domku, zostały zaznaczone na rysunku szarym kolorem. Mają one wartości: 1, -3, 2, 1, -1, 5, a więc suma ich wartości to 5.

Które zgłoszenie było pierwsze? (gdz)

Memory limit: 128 MB Time limit: 3.00 s

W tym roku, aby zmotywować trochę uczniów do rozwiązywania zadań, kadrowicze postanowili nagrodzić dodatkowo jednego z uczestników. Nagrodą główną będzie uścisk dłoni przewodniczącego kadry, a dostanie go osoba, która wysłała pierwsze zgłoszenie na obozie (czyli zgłoszenie z numerem równym 1). Jednakże, aby wręczyć tę nagrodę, kadrowicze muszą sprawdzić kto wysłał to pierwsze zgłoszenie.

Kadrowicze wiedzą, że sprawdzaczka przetrzymuje wszystkie N wykonanych zgłoszeń w tablicy o indeksach od 1 do N, ale ich kolejność jest nieznana. Przyjmijmy, że liczba $P[i]$ oznacza numer zgłoszenia i. Problem w tym, że sprawdzaczka nie ma w swoim API funkcji, która umożliwiłaby sprawdzenie numeru zgłoszenia w prosty sposób. W zamian za to dostępne są funkcje:

- `f(i, j, d)`: czy różnica między numerem i-tego oraz j-tego zgłoszenia jest podzielna przez d, tzn. czy $d|P[i] - P[j]$?
- `g(i, j)`: czy numer i-tego zgłoszenia jest większy niż numer j-tego zgłoszenia?

W powyższych pytaniach i i j są dowolnymi indeksami ze zbioru $\{1, 2, ..., N\}$, zaś d jest dowolną dodatnią liczbą całkowitą.

Podczas testów kadrowicze odkryli, że funkcja `g` wykonuje się niezwykle wolno, za to funkcja `f` dość szybko. Postanowili zatem, że funkcję `g` wykonają **najmniejszą** możliwą liczbę razy. Funkcję `f` mogą oni wykonywać dowolną (racjonalną) liczbę razy, z dowolnymi wartościami parametrów.

Czy jesteś w stanie napisać za kadrowiczów program, który znajdzie pierwsze zgłoszenie?

Ocenianie

Niech $M(N)$ będzie najmniejszą możliwą liczbą wywołań funkcji typu `g`, za pomocą której da się znaleźć pierwsze zgłoszenie w tablicy o ustalonej długości N, niezależnie od tego, jaka jest ich kolejność. Twój program otrzyma punkty za dany test, tylko jeśli liczba wywołań typu `g` jaką wykona nie przekroczy $M(N)$. Poza tym program musi zmieścić się w limicie czasowym, a więc wykonać rozsądną liczbę wywołań funkcji typu `f` (choć niekoniecznie minimalną).

Komunikacja

Aby użyć biblioteki, należy wpisać na początku programu w C/C++ wiersz

```
#include "cgdzlib.h"
```

Biblioteka udostępnia następujące funkcje i procedury:

- `int inicjuj()` – zwraca liczbę N. Powinna zostać użyta dokładnie raz, na samym początku działania programu.
- `int f(int i, int j, int d)` – wywołuje na sprawdzaczce funkcję typu `f`. Jej wynikiem jest jedna liczba: 1, jeśli $d|P[i] - P[j]$, a 0 w przeciwnym przypadku.
- `int g(int i, int j)` – wywołuje na sprawdzaczce funkcję typu `g`. Jej wynikiem jest jedna liczba: 1, jeśli $P[i] > P[j]$, a 0 w przeciwnym przypadku.
- `void odpowiedz(int k)` – odpowiada, że pierwszym zgłoszeniem było to, które znajduje się w tablicy na pozycji k (tzn. że $P[k] = 1$). Uruchomienie tej funkcji kończy działanie Twojego programu.

Twój program nie może czytać żadnych danych (ani ze standardowego wejścia, ani z plików). Nie może również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (stderr) – pamiętaj jednak, że zużywa to cenny czas.

Przykładowe wykonanie

Poniższa tabela zawiera przykładowy ciąg wywołań funkcji prowadzący do odgadnięcia pozycji, na której znajduje się pierwsze zgłoszenie.

Numer wywołania	Wywołanie	Wynik	Wyjaśnienie
	`inicjuj()`	5	$N = 5$
1	`f(1, 2, 2)`	0	$2 \nmid P[1] - P[2]$
2	`g(1, 2)`	0	$P[1] < P[2]$
3	`f(3, 2, 3)`	1	$3 \mid P[3] - P[2]$
4	`g(2, 5)`	1	$P[2] > P[5]$
5	`f(1, 3, 2)`	1	$2 \mid P[1] - P[3]$
6	`f(1, 4, 3)`	1	$3 \mid P[1] - P[4]$
	`odpowiedz(4)`		Odpowiadamy, że $k = 4$.

Po drugim wywołaniu wiemy, że $P[2] \neq 1$. Stąd po trzecim wywołaniu zachodzi jedna z możliwości: $(P[2] = 2, P[3] = 5)$ lub $(P[2] = 4, P[3] = 1)$, lub $(P[2] = 5, P[3] = 2)$. Czwarte wywołanie eliminuje pierwszą z tych możliwości. Piąte wywołanie pozwala teraz stwierdzić, że $P[1] \in \{3, 4\}$. Skoro więc w szóstym wywołaniu mamy $3 \mid P[1] - P[4]$, to $P[1] = 4$, $P[4] = 1$. Szukaną pozycją w permutacji jest $k = 4$.

Podana sekwencja wywołań poprawnie znajduje pierwsze zgłoszenie, jednak nie uzyskałaby żadnych punktów za ten test, gdyż dla dowolnej kolejności zgłoszeń pierwsze z nich da się znaleźć za pomocą co najwyżej jednego wywołania funkcji typu `g` (tj. $M(5) = 1$). Zauważ, że liczba wykonanych wywołań funkcji typu `f` nie gra tu roli.

Eksperymenty

W dziale z plikami dostępna jest przykładowa biblioteka, która pozwoli Ci przetestować poprawność formalną rozwiązania. Biblioteka wczytuje opis permutacji ze standardowego wejścia w następującym formacie:

- w pierwszym wierszu liczba całkowita N – długość permutacji;
- w drugim wierszu N pooddzielanych pojedynczymi odstępami liczb od 1 do N – kolejne elementy permutacji.

Przykładowe wejście dla biblioteki znajduje się w pliku `gdz0.in`. Po zakończeniu działania programu biblioteka wypisuje na standardowe wyjście informację, czy odpowiedź udzielona przez Twoje rozwiązanie była poprawna, oraz liczbę wywołań funkcji typu `g`. W tym samym katalogu znajduje się przykładowe rozwiązanie `gdz.cpp` korzystające z biblioteki. Rozwiązanie to nie minimalizuje liczby wywołań funkcji typu `g`.

Do kompilacji rozwiązania wraz z biblioteką służy polecenie:

```
g++ -O2 -static cgdzlib.c gdz.cpp -lm -o gdz
```

Plik z rozwiązaniem i biblioteka powinny znajdować się w tym samym katalogu.

Ograniczenia

$1 \leq N \leq 500\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 5\,000$	36
2	$N \leq 150\,000$	32
3	brak dodatkowych ograniczeń	32

Piramida **(piramida)**

Memory limit: 256 MB Time limit: 3.00 s

Uczniowie przyjechali na obóz, rozgościli się w luksusowych pokojach i spokojnie czekali na ogłoszenia obiadowe. Znów zaczął się ten magiczny czas obozu, czas zarwanych nocek i bycia niewyspanym przez cały tydzień – pysznie. Wszystko układało się praktycznie idealnie, aż do chwili ogłoszeń obiadowych, gdzie okazało się, że na obozie nie ma najznamienitszej, najwspanialszej i najpyszniejszej jego części – pana Karola, którego to obecność dodaje ludziom na obozie animuszu, którego krzyk budzi w każdym zachwyt, którego widok sprawia niesamowite wrażenie i ciepło na serduszku każdego ucznia. Nieufni wobec paskudnej kadry uczniowie stwierdzili, że to właśnie paskudna kadra schowała przed nimi pana Karola w najwyższej sali pałacyku. Uczniowie postanowili więc wykonać samobójczą misję odbicia pana Karola. Przeprowadzili burzę mózgów i określili jasno, że wejście po prostu schodami rozczarowałoby, schowanego przed nimi w najwyższej sali pałacyku, pana Karola swoim brakiem finezji. Postanowili więc dostać się do najwyższej sali pałacyku oknem. Potrzebowali więc swego rodzaju rusztowania, wieży, niestety jedne co udało im się wymyślić to zbudowanie piramidy ...

Uczniowie rozpoczęli już budowę piramidy, jednak, jak można się było spodziewać, ich konstrukcja nie była zbyt udana. Postawili oni na początku ciąg N kwadratowych bloczków, jeden przy drugim. Następnie umieszczali wedle swojego uznania kolejne bloczki, każdy ustawiając równo na jednym z już postawionych. Powstałą w ten sposób konstrukcję można opisać ciągiem N liczb całkowitych A_i, opisującym ile bloczków znajduje się w i-tej kolumnie. Jak można się domyślić, przyszli informatycy nie mają zbyt wysokich zmysłów estetycznych, a więc ich konstrukcja niezbyt przypominała piramidę. Mają oni jeszcze K bloczków do wykorzystania. W celu przybliżenia konstrukcji wyglądem do piramidy postanowili, że od tej pory bloczek w i-tej kolumnie będzie można postawić tylko, gdy w kolumnach sąsiednich znajduje się przynajmniej tyle samo bloczków co w i-tej. Sąsiednimi kolumnami dla kolumny i-tej są kolumny o numerach $i-1$ i $i+1$ (w przypadku kolumny pierwszej i ostatniej mają one odpowiednio mniej sąsiednich kolumn). Uczniowie zastanawiają się teraz jak wysoką piramidę uda im się zbudować. Twoim zadaniem będzie policzenie największej możliwej wysokości kolumny, jaką uda się zbudować uczniom przy założeniu, że od tej pory będą stawiali bloczki w optymalny sposób.

Wejście

W pierwszym wierszu wejścia znajdują dwie liczby całkowite N i K oddzielone pojedynczym odstępem, oznaczające długość ciągu bloczków i liczbę pozostałych do ułożenia bloczków. W drugim wierszu znajduje się ciąg N liczb całkowitych A_i pooddzielanych pojedynczymi odstępami, oznaczających liczby bloczków w kolejnych kolumnach.

Wyjście

W pierwszym (jedynym) wierszu wyjścia powinna się znaleźć jedna liczba całkowita – maksymalna wysokość piramidy jaką można zbudować.

Ograniczenia

$1 \leq N \leq 100\,000$, $0 \leq K \leq 10^9$, $1 \leq A_i \leq 10^9$.

Podzadania

Podzadanie	Warunki	Punkty
1	$A_i \leq 2$	10
2	$K \leq 1$	10
3	$K \leq 10$	10
4	$N \leq 20$, $K \leq 20$	10
5	$N \leq 1\,000$	20
6	brak dodatkowych ograniczeń	40

Przykład

Input	Output
8 5 1 3 2 2 4 1 2 1	5

Blokada łańcuchowa (blokada-lancuchowa)

Memory limit: 128 MB Time limit: 2.00 s

Kadra obozu wybrała się kilka dni temu na spotkanie integracyjne do *pokoju zagadek*.

Pierwsza przeszkoda, jaką spotkali, to łańcuch z kłódką szyfrową, na który zamknięte są pierwsze drzwi. Kłódka wymaga podania dwóch liczb całkowitych do otwarcia. Rozpoczyna się burza mózgów, festiwal inteligencji, wręcz kakofonia pomysłów.

Po pewnym namyśle kadra postanowiła połączyć fakty: Artur przypomniał sobie, że mistrz gry wspomniał o ciągu dodatnich liczb całkowitych $A_1, A_2, \ldots, A_N$, Anadi doszedł do wniosku, że kłódka nie akceptuje liczb mniejszych od zera i większych od $1\,000\,000\,006$, Marcin odkrył dwie dodatnie liczby całkowite L_1 i R_1 wydrapane na kłódce, a Marcel zauważył przez szparę przy podłodze $Q-1$ analogicznych pomieszczeń za drzwiami.

Adam połączył kropki i krzyknął: "Już wiem! Jesteśmy w zadaniu programistycznym", na co Michał odpowiedział: "A ja chyba wiem o co w nim chodzi", po czym zaczął opowiadać treść zadania:

Dane są dwie dodatnie liczby całkowite N i Q, ciąg dodatnich liczb całkowitych $A_1, A_2, \ldots, A_N$ oraz Q zapytań opisanych każde parą dodatnich liczb całkowitych L_i oraz R_i. Wynikiem dla każdego zapytania jest liczba

$$A_{L_i} + \cfrac{1}{A_{L_i+1} + \cfrac{1}{A_{L_i+2} + \cfrac{1}{\ddots + \cfrac{1}{A_{R_i}}}}}.$$

Janek zauważył, że coś tu nie gra – przecież każda kłódka wymaga dwóch liczb całkowitych, a nie jednej wymiernej. Pomyślał chwilę i powiedział: "No tak, przecież musimy zapisać każdy wynik w postaci ułamka nieskracalnego $\frac{P}{Q}$ i podać do kłódki reszty z dzielenia liczb P i Q przez $1\,000\,000\,007$".

Pomoż uwięzionej kadrze!

Napisz program, który wczyta długość ciągu, liczbę zapytań, ciąg i zapytania, wyznaczy wynik dla każdego zapytania i wypisze dla każdego zapytania reszty z dzielenia przez $1\,000\,000\,007$ licznika i mianownika wyniku zapytania w postaci ułamka nieskracalnego.

Wejście

W pierwszym wierszu wejścia znajdują się dwie dodatnie liczby całkowite N i Q oddzielone pojedynczym odstępem i oznaczające długość ciągu oraz liczbę zapytań. W drugim wierszu wejścia znajduje się N dodatnich liczb całkowitych $A_1, A_2, \ldots, A_N$ pooddzielanych pojedynczymi odstępami i oznaczających kolejne wyrazy ciągu. W każdym z kolejnych Q wierszy znajdują się po dwie dodatnie liczby całkowite L_i oraz R_i oddzielone pojedynczym odstępem i oznaczające początek i koniec fragmentu ciągu, dla którego należy wyznaczyć wynik.

Wyjście

W i-tym wierszu wyjścia powinna znaleźć się odpowiedź na i-te zapytanie: dwie nieujemne liczby całkowite P_i oraz Q_i oznaczające reszty z dzielenia przez $1\,000\,000\,007$ licznika i mianownika wyniku i-tego zapytania zapisanego w postaci ułamka nieskracalnego oddzielone pojedynczym odstępem.

Ograniczenia

$1 \leq N, Q \leq 300\,000$, $1 \leq A_i \leq 10^9$, $1 \leq L_i \leq R_i \leq N$.

Podzadania

Podzadanie	Warunki	Punkty
1	$A_1 = A_2 = \cdots = A_N = 1$	10
2	$N, Q \leq 1\,000$	20
3	brak dodatkowych ograniczeń	70

Przykład

Input	Output
5 4 3 6 2 4 1 1 2 4 5 2 4 1 5	19 6 5 1 58 9 224 71

Psychotest (psychotest)

Memory limit: 512 MB Time limit: 2.00 s

No nie zostaniemy tu już na zawsze! Zostaniemy tu! Po co nam to było? Mówiłem, żeby tu nie przychodzić! To już jest koniec! To twoja wina! Przecież nigdy nie przejdziemy psychotestu! Nienawidzę Cię! Już po nas, już po nas. Ja nie zrobię tego psychotestu. Nie da się zrobić psychotestu. On jest jakiś NP-trudny. To już jest koniec . . .

Bartosz wykrzykiwał te i inne hasła będąc uwięzionym w escape roomie. Do rozwiązania był psychotest, z którym nikt z nas nie dawał sobie rady. Światła migały, a kolory biegały, niebezpieczeństwo zbliżało się. Czas się kończył, a koniec się zbliżał. Migające światełka oślepiały swym blaskiem, doprowadzając nas do szaleństwa. Ściana tekstu psychotestu była przerażająca. Patrzyliśmy na nią z nadzieją, że psychotest sam się rozwiąże, ale on nie chciał. Psychotest nie chciał sam się rozwiązać. On tylko oceniał nas swym surowym spojrzeniem i śmiał się nam w twarz. Koniec był blisko, koniec był coraz bliżej. Nie przechodziliśmy psychotestu, byliśmy bezradni. Czuliśmy niemoc, chyba jedno z najgorszych uczuć, jakie tylko można mieć. Zbierało nam się na łzy, czy tak właśnie miał wyglądać koniec tej przygody? Czy to już koniec? Psychotest. Czy koniec to psychotest? Czy ostatnie, co w życiu zobaczysz to psychotest? Czy ostatnie słowo, jakie przyjdzie Ci usłyszeć, to "psychotest"? Psychotest, psychotest. Psychotest doprowadzał nas do szaleństwa, do obłędu. Poziom dymu w pokoju się podnosił, znikała widoczność, zupełnie jak w gęstej mgle. A powiększenie było małe. Zmniejszało się? Czy tylko nam się wydaje, że pomieszczenie się zmniejsza? Psychotest. To już jest koniec, końcem jest psychotest. Światła migotały coraz mocniej. Oślepiały nas iluminacje, nie wiedzieliśmy nic, krzyczeliśmy. Psychotest. To właśnie był psychotest. Doprowadzający do obłędu. Złowieszcza poświata psychotestu przyprawiała o ciarki. Psychotest.

Przyciski psychotestu migały. Psychotest. Przycisków psychotestu było nie mniej, nie więcej, a dokładnie N. Psychotest. Przyciski psychotestu numerowane są kolejnymi liczbami naturalnymi od 1 do N. Psychotest. Na ścianie psychotestu napisanych było nie mniej, nie więcej, a dokładnie N słów S_i złożonych z małych liter alfabetu łacińskiego. Psychotest. Słowa psychotestu numerowane są kolejnymi liczbami od 1 do N. Psychotest. Rozwiązaniem psychotestu jest sekwencja N naciśnięć przycisków. Psychotest. i-tym naciśniętym przyciskiem z sekwencji powinien być ten, którego numer (psychotest) odpowiada na pytanie, ile minimalnie słów trzeba powiedzieć (psychotest), by pokryć pierwsze i słów psychotestu i nie pokryć żadnego z pozostałych. Psychotest. Powiedziane słowo również składa się z małych liter alfabetu łacińskiego. Psychotest. Powiedziane słowo pokrywa słowo psychotestu, gdy jest jakimś jego fragmentem początkowym (prefiksem). Psychotest. Przykładowo słowo `psycho` jest fragmentem początkowym słowa `psychotest`. Psychotest. Powiedziane słowo może pokrywać więcej niż jedno słowo psychotestu. Psychotest. Czy jesteś w stanie podać poprawną sekwencję rozwiązującą psychotest? Psychotest.

Psychotest.

Wejście

W pierwszym wierszu wejścia psychotestu znajduje się jedna liczba całkowita N. W każdym z kolejnych N wierszy znajduje się ciąg małych liter alfabetu łacińskiego S_i, i-te ze słów jest i-tym słowem psychotestu. Psychotest.

Wyjście

Na wyjście psychotestu należy wypisać N wierszy, w i-tym z nich powinna być jedna liczba całkowita – numer i-tego przycisku z sekwencji, a więc minimalna liczba (psychotest) słów, którą trzeba powiedzieć, by pokryć pierwsze i słów psychotestu i nie pokryć żadnego z pozostałych słów psychotestu. Gwarantowane jest, że rozwiązanie psychotestu zawsze istnieje. Psychotest.

Ograniczenia

$0 \leq N \leq 100\,000$, $\sum |S_i| \leq 2\,000\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 10, \sum \lvert S_i \rvert \leq 100$	22
2	$N \leq 500, \sum \lvert S_i \rvert \leq 10\,000$	24
3	słowa psychotestu są posortowane leksykograficznie	23
4	brak dodatkowych ograniczeń	31

Przykład

Input

```
5
psychotest
psychika
psychoza
obled
szalenstwo
```

Output

```
1
2
1
2
3
```

Explanation

- By pokryć tylko pierwsze słowo można powiedzieć `psychotest`.
- By pokryć tylko dwa pierwsze słowa można powiedzieć `psychi` i `psychotest`.
- By pokryć tylko trzy pierwsze słowa można powiedzieć `psych`.
- By pokryć tylko cztery pierwsze słowa można powiedzieć `psych` i `obled`.
- By pokryć wszystkie słowa można powiedzieć `psy` i `obled` i `s`.

Input

```
10
orzech
pisk
maska
miska
click
coconut
oreo
cherry
paf
wrrrrr
```

Output

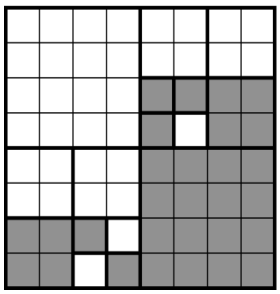
```
1
2
3
3
4
5
5
4
4
5
```

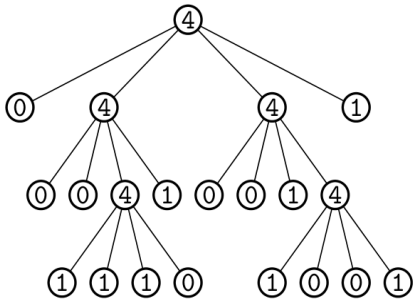
Mapa Bałkanów (mapa-balkanow)

Memory limit: 256 MB Time limit: 5.00 s

Podczas wizyty w escape roomie o nazwie `Historia półwyspu Bałkańskiego`, Michał trafił na ciąg składający się ze znaków `0`, `1` i `4`. Inne wskazówki sugerowały, że powinien on reprezentować mapę kwadratowego obszaru na Morzu Śródziemnym. Michał nie musiał się długo zastanawiać – błyskawicznie domyślił się, że ciąg ten oznacza *drzewo czwórkowe*, reprezentujące szukaną mapę.

Ale zacznijmy od początku. Wyobraźmy sobie kwadratową bitmapę o rozmiarach $2^M \times 2^M$. Każdy piksel bitmapy jest albo biały, albo czarny. W przypadku tej mapy, piksel biały będzie symbolizował wodę, a czarny to obszar pewnej wyspy. Taką bitmapę można reprezentować w formie skompresowanej, właśnie za pomocą drzewa czwórkowego. Jeżeli wszystkie piksele bitmapy są białe, drzewo składa się z jednego wierzchołka z etykietą `0`, a cały obszar reprezentuje wodę. Jeżeli wszystkie piksele są czarne, drzewo ma jeden wierzchołek z etykietą `1`, oznaczający obszar wyspy. W przeciwnym wypadku korzeń drzewa ma etykietę `4` i posiada on cztery poddrzewa, które odpowiadają czterem ćwiartkom mapy o rozmiarach $2^{M-1} \times 2^{M-1}$ (w kolejności lewa górna, prawa górna, lewa dolna i prawa dolna). Drzewo takie można opisać za pomocą słowa złożonego właśnie ze znaków `0`, `1` i `4`: opis drzewa zaczyna się etykietą jego korzenia, po której następują kolejno opisy jego poddrzew. Na poniższym rysunku przedstawiono przykładową mapę dla $M = 3$ oraz odpowiadające jej drzewo czwórkowe, którego opisem jest słowo `40400411101400141001 1`:





Obszarem wyspy nazwiemy maksymalny zbiór sąsiadujących ze sobą pikseli koloru czarnego (przy czym piksele sąsiadują ze sobą, jeśli stykają się bokiem)[1].

Dla danego słowa opisującego mapę, wyznacz liczbę rozłącznych wysp oraz wielkość największej z nich. W powyższym przykładzie mamy dwie wyspy o rozmiarach 24 i 5.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita M, reprezentująca wielkość mapy.

W drugim wierszu znajduje się niepuste słowo kodujące mapę, złożone ze znaków `0`, `1` i `4`.

Możesz założyć, że wejście jest poprawne, w szczególności wysokość drzewa czwórkowego (czyli liczba krawędzi na ścieżce od korzenia do najgłębszego wierzchołka) jest nie większa niż M. Mapa zawiera co najmniej jeden czarny piksel.

Wyjście

Twój program powinien wypisać na wyjście dwa wiersze.

W pierwszym z nich powinna się znaleźć liczba oznaczająca liczbę rozłącznych wysp na mapie.

W drugim wierszu powinna się znaleźć liczba oznaczająca wielkość największej wyspy. Druga z tych liczb może być bardzo duża, więc należy wypisać jej resztę z dzielenia przez $10^9 + 7$.

Ograniczenia

$0 \leq M \leq 10^9$, długość słowa opisującego mapę (dalej oznaczana jako N) nie przekracza $1\,000\,000$.

[1] Formalnie, *obszarem* nazwiemy zbiór pikseli koloru czarnego, taki że z każdego z nich da się dojść do każdego innego, przechodząc przez pewną liczbę pikseli koloru czarnego, z których każde kolejne dwa stykają się bokiem. Obszar nazwiemy *maksymalnym*, jeśli nie można go powiększyć o żaden inny piksel koloru czarnego, tak aby nadal był obszarem. W tym zadaniu rozważamy tylko obszary maksymalne.

Podzadania

Jeśli Twój program poprawnie wypisze **tylko** jedną z liczb na wyjściu, uzyska 50% punktów przewidzianych za test. Nadal musi jednak wypisać dwa wiersze, każdy zawierający liczbę całkowitą od 0 do $10^9 + 6$.

Podzadanie	Warunki	Punkty
1	$M \leq 10$	30
2	$M, N \leq 1\,000$	30
3	$M \leq 1\,000\,000$	30
4	brak dodatkowych ograniczeń	10

Przykład

Input	Output
3 404004111014001410011	2 24

Droga do szkoły (droga-do-szkoly)

Memory limit: 256 MB Time limit: 4.00 s

Królestwo Bajtocji składa się z N miasteczek oraz M dwukierunkowych połączeń między nimi, każde długości jednego kilometra. W miasteczku o numerze 1 mieszka Karol. Jest on przykładnym uczniem Liceum Bajtocjańskiego znajdującego się w miasteczku o numerze 2. Każdego dnia mozolnie pokonywał on długą trasę dzielącą go od ukochanego liceum. Pewnego dnia burmistrz Bajtocji postanowił usprawnić system komunikacji w swoim królestwie i dobudować nowe połączenia pomiędzy niektórymi miasteczkami.

„Hurra!" – pomyślał ucieszony Karol z myślą, że nie będzie już musiał przemierzać tak długiej trasy do swojej szkoły. Chociaż... chwila chwila. Co on będzie mówić kiedyś swoim dzieciom? Przecież każdy prawilny ojciec musi kiedyś opowiadać synom i córkom o tym, jak to musiał przepływać wszystkie rzeki na świecie i wybijać 15 aligatorów w drodze do szkoły. Stwierdził, że musi coś z tym zrobić.

Karol spotkał się z burmistrzem, jednak ten był nieugięty – nie chciał odwołać budowy nowych dróg. Wtedy nasz bohater zaproponował pewne rozwiązanie – Karol pomoże burmistrzowi tak rozplanować budowę nowych dróg, aby było ich jak najwięcej przy założeniu, że najkrótsza ścieżka z domu Karola do jego liceum będzie nie krótsza niż 5 kilometrów. Oczywiście Karol nie może zaproponować takich połączeń, które już istnieją w miasteczku lub takich, które prowadzą z miasteczka do samego siebie. Burmistrz uznał by to wtedy za jakiś przekręt.

„Zgoda!" – odparł burmistrz. Karol wyszedł zadowolny z siebie, jednak zorientował się, że postawił sobie zadanie, którego nie jest w stanie zrealizować... Czy możesz mu w tym pomóc?

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite N oraz M oddzielone pojedynczym odstępem, oznaczające kolejno liczbę miasteczek w Królestwie Bajtocji oraz liczbę połączeń między nimi.

W każdym następnym $(i+1)$-szym wierszu znajdują się dwie liczby całkowite C_i oraz D_i, oznaczające, że istnieje bezpośrednie połączenie między miasteczkami C_i oraz D_i.

Zagwarantowane jest, że każde połączenie występuje na wejściu co najwyżej raz, nie ma połączeń z żadnego miasteczka do samego siebie oraz że istnieje ścieżka z miasteczka 1 do miasteczka 2, a najkrótsza z nich jest nie krótsza niż 5 kilometrów.

Wyjście

Na wyjściu powinna znaleźć się jedna liczba całkowita, oznaczająca maksymalną liczbę połączeń które możemy dodać w Królestwie Bajtocji w taki sposób, aby nie istniała ścieżka długości krótszej niż 5 między miasteczkami 1 i 2.

Ograniczenia

$5 \leq N \leq 40\,000$, $4 \leq M \leq 1\,000\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 7$	15
2	brak dodatkowych ograniczeń	85

Przykład

Input

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

Output

```
10
```

Explanation

Możliwy zbiór połączeń, które można dodać to:
$\{(3,4),(3,6),(4,5),(5,6),(5,8),(6,7),(7,8),(7$

Łańcuszek (lancuszek)

Memory limit: 128 MB Time limit: 2.00 s

Pewnego dnia Karol przybył do szkoły trochę wcześniej niż zawsze. Postanowił więc zrobić to, co lubi najbardziej – podobijać zadanka po wczorajszym konteście. Już kończył pisać jedno z nich, bijąc po raz kolejny swój rekord w pisaniu Dijkstry, kiedy nagle ktoś mu przerwał. Jadzia, jego koleżanka z klasy, chciała mu pokazać własnoręcznie zrobiony łańcuszek z N czerwonych lub zielonych koralików.

„Wrrrr" – odwarknął Karol. Jak można mu przerywać tak ważną rzecz? Chcąc spławić koleżankę, musiał coś szybko wymyślić. „Nie podoba mi się, nie ma w nim nawet fragmentu zawierającego koraliki czerwony–zielony–czerwony". Jadzia odeszła, trochę zasmucona, ale Karol w końcu mógł wrócić do swoich priorytetów. Przynajmniej tak myślał – po chwili dziewczyna wróciła z łańcuszkiem, w którym jeden koralik miał zmieniony kolor.

„Teraz to nie ma nawet fragmentu zielony–zielony". Jadzia znowu odeszła bez słowa, jednak widać było, że nie da za wygraną. Jeśli tak dalej będzie szło, to Karol nic nie dobije! Tak nie może być. Tutaj przybywasz z pomocą Ty – musisz napisać program, który po każdej zmianie jednego koralika znajdzie długość najkrótszego takiego ciągu koralików, który nie występuje jako spójny fragment w łańcuszku Jadzi. W ten sposób Karol nie będzie już tracił tyle czasu na niepotrzebne rzeczy.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite N oraz M oddzielone pojedynczym odstępem, oznaczające kolejno długość łańcuszka Jadzi oraz liczbę zmian pojedynczego koralika.

Drugi wiersz wejścia zawiera słowo złożone z N znaków 0 i 1, oznaczających kolejno koraliki czerwony i zielony.

Kolejne M wierszy zawiera opisy zmian koralików. W i-tym z tych wierszy znajduje się jedna liczba całkowita A_i oznaczająca, że Jadzia zmieniła kolor A_i-tego koralika.

Łańcuszek **nie jest cykliczny**, co oznacza, że N-ty koralik nie sąsiaduje z pierwszym.

Wyjście

Na wyjście należy wypisać dokładnie $M + 1$ wierszy. i-ty z nich powinien zawierać jedną liczbę całkowitą oznaczającą długość najkrótszego podsłowa złożonego ze znaków 0 i 1, które nie występuje jako podsłowo w słowie kodującym kolory koralików w łańcuszku po $i - 1$ zamianach Jadzi.

Ograniczenia

$1 \leq N \leq 100\,000$, $0 \leq M \leq 10\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N, M \leq 1\,000$	46
2	brak dodatkowych ograniczeń	54

Przykład

Input

```
6 2
001010
6
2
```

Output

```
2
3
2
```

Explanation

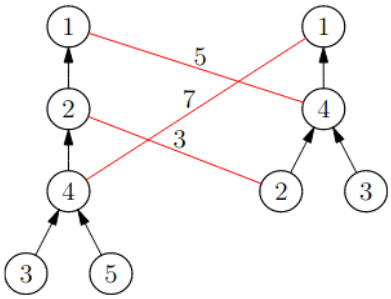
W słowie 001010 najkrótszym niewystępującym podsłowem jest 11 o długości 2. Po zmianie szóstego znaku dostajemy 001011, w którym występują wszystkie podsłowa długości 1 i 2, ale nie występuje np. podsłowo 110 długości 3. Po zmianie drugiego znaku dostajemy 011011, w którym nie występuje podsłowo 00.

Dinozaury (dinozaury)

Memory limit: 512 MB Time limit: 6.00 s

Największą pasją Karola (oczywiście poza pisaniem kontestów) są dinozaury. Ostatnio znalazł w jednej ze swoich książek drzewa genealogiczne dwóch rodzin dinozaurów – Bitozaurów i Bajtocepsów. Drzewo takie składa się z pewnej liczby gatunków wraz z informacją, które z nich są swoimi bezpośrednimi przodkami. Gatunek oznaczony numerem 1 w każdym z drzew powstał jako pierwszy i nie posiada bezpośredniego przodka.

Karol doczytał również w tej samej książce, że obie te rodziny były na przestrzeni wieków swoimi arcywrogami, a niektóre z ich gatunków szczególnie się nie lubiły. W szczególności znalazł on K informacji postaci C_i D_i E_i, które wskazują, że współczynnik agresji pomiędzy C_i-tym gatunkiem Bitozaurów oraz D_i-tym gatunkiem Bajtocepsów wynosi E_i. Poniższy rysunek pokazuje drzewa wraz z połączeniami z testu przykładowego.



Karol zastanawia się teraz nad następującą kwestią – czy jest to możliwe, że relacje między konkretnymi gatunkami dinozaurów są zależne od relacji pomiędzy ich przodkami? Aby ułatwić Karolowi odpowiedź na to pytanie, Twoim celem jest napisać program, który będzie dla podanych gatunków odpowiadać na zapytania o to, jaki jest sumaryczny współczynnik agresji pomiędzy nimi oraz wszystkimi przodkami dwóch konkretnych gatunków.

Wejście

W pierwszym wierszu wejścia znajduje się liczba całkowita N, oznaczająca liczbę gatunków Bitozaurów.

W drugim wierszu wejścia znajduje się $N-1$ liczb całkowitych A_i pooddzielanych pojedynczymi odstępami, oznaczających, że dla każdego i gatunek A_i jest bezpośrednim przodkiem gatunku o numerze $i+1$.

W trzecim wierszu wejścia znajduje się liczba całkowita M, oznaczająca liczbę gatunków Bajtocepsów.

W czwartym wierszu wejścia znajduje się $M-1$ liczb całkowitych B_i pooddzielanych pojedynczymi odstępami, oznaczających, że dla każdego i gatunek B_i jest bezpośrednim przodkiem gatunku o numerze $i+1$.

W piątym wierszu wejścia znajduje się liczba całkowita K, oznaczająca liczbę informacji o współczynnikach agresji.

Każdy z kolejnych K wierszy wejścia zawiera po trzy liczby całkowite C_i, D_i, E_i pooddzielane pojedynczymi odstępami, oznaczające, że współczynnik agresji pomiędzy C_i-tym gatunkiem Bitozaurów a D_i-tym gatunkiem Bajtocepsów wynosi E_i.

W kolejnym wierszu wejścia znajduje się liczba całkowita Q, oznaczająca liczbę zapytań, na które musimy odpowiedzieć.

Każdy z kolejnych Q wierszy wejścia zawiera po dwie liczby całkowite X_i, Y_i oddzielone pojedynczym odstępem, oznaczające zapytanie postaci „jaki jest sumaryczny współczynnik agresji pomiędzy przodkami X_i-tego gatunku Bitozaurów oraz Y_i-tego gatunku Bajtocepsów.

Wyjście

Na wyjściu powinno się znaleźć dokładnie Q wierszy – w i-tym z nich odpowiedź na i-te zapytanie.

Ograniczenia

$1 \leq N, M, K, Q \leq 200\,000$, $1 \leq E_i \leq 10^9$, $1 \leq A_i, C_i, X_i \leq N$, $1 \leq B_i, D_i, Y_i \leq M$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N, M, K, Q \leq 1\,000$	15
2	wszystkie liczby A_i są parami różne	40
3	brak dodatkowych ograniczeń	45

Przykład

Input	Output
5 1 4 2 4 4 4 4 1 3 1 4 5 4 1 7 2 2 3 5 2 4 3 2 5 3 1 1 1 4	5 15 12 0 5

Trójsegmentowe tunele **(trojsegmentowe-tunele)**

Memory limit: 32 MB Time limit: 1.00 s

Mogłoby się wydawać, że Wrocław jest wielkim miastem. To jednak iluzja. Tak naprawdę miasto ma kształt linii, a wszystkie domy znajdują się przy jednej, rozciągającej się z zachodu na wschód ulicy. Są one ponumerowane od 1 do N włącznie. Niestety taki plan miasta niesie ze sobą wiele niedogodności komunikacyjnych. Główna ulica jest bardzo często zakorkowana. Jako iż jesteś światowej sławy projektantem infrastruktury urbanistycznej, to prezydent Wrocławia poprosił Cię o pomoc w uwikłaniu się z tym problemem.

Na szczęście masz pomysł na to, jak pomóc mieszkańcom Wrocławia. Wrocław znany jest ze swoich tramwai, a dokładniej z tego, jak często się wykolejają. Aby ograniczyć ten problem, prezydent postanowił, że no nowe linie tramwajowe będą łączyły bezpośrednio dwa domy i nie będzie żadnych pośrednich przystanków. Dodatkowym wymaganiem jest jeszcze, żeby tramwaje nie szpeciły krajobrazu, więc postanowiono, że będą się one poruszać w tunelach.

Aby sieć tuneli była sprawna, między każdymi dwoma domami musi istnieć (niekoniecznie bezpośrednie) połączenie tunelami, które:

- biegnie cały czas w jednym kierunku (ze wschodu na zachód albo odwrotnie),
- składa się z maksymalnie trzech tuneli.

Jako iż budżet miejskiej spółki komunikacyjnej jest ograniczony, to jesteś zmuszony do zaprojektowania sieci, w której znajduje się maksymalnie $5 \cdot N$ tuneli. Jeżeli to zadanie Cię przerośnie, to prezydent Wrocławia nadal zaoferuje Ci częściową nagrodę za plan zawierający nie więcej niż $10 \cdot N$ tuneli.

Napisz program, który wczyta liczbę domów we Wrocławiu oraz rodzaj podzadania i wypisze na standardowe wyjście plan sieci spełniającej warunki prezydenta.

Wejście

W pierwszym (jedynym) wierszu standardowego wejścia znajdują się dwie liczby naturalne N oraz R, określające liczbę domów we Wrocławiu, oraz numer podzadania, którego dotyczy dany zestaw testowy.

Wyjście

W pierwszym wierszu standardowego wyjścia wypisz liczbę tuneli T, zawartych w Twoim planie. W kolejnych T wierszach wypisz opisy tuneli, składające się z dwóch liczb naturalnych – numerów budynków, które mają być połączone bezpośrednim tunelem.

Nie ma znaczenia, w jakiej kolejności zostaną wypisane tunele oraz budynki połączone pewnym konkretnym tunelem, ale każdy tunel może znaleźć się w planie maksymalnie raz oraz nie jest dozwolone łączenie domu z samym sobą. W przypadku przekroczenia limitu użytych tuneli, zgłoszenie otrzyma werdykt błędnej odpowiedzi.

Jeżeli istnieje wiele odpowiedzi, to wypisz dowolną z nich.

Ograniczenia

$2 \leq N \leq 500$.

Podzadania

Podzadanie	Warunki	Punkty
1	można użyć $10 \cdot N$ tuneli	40
2	można użyć $5 \cdot N$ tuneli	60

Testy ocen

1. $N = 15$, pierwsze podzadanie
2. $N = 74$, drugie podzadanie
3. $N = 100$, pierwsze podzadanie
4. $N = 100$, drugie podzadanie

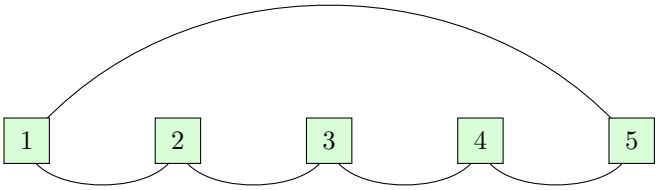
Przykład

Input

```
5 1
```

Output

```
7
1 2
2 3
3 4
4 5
1 3
1 4
2 4
```

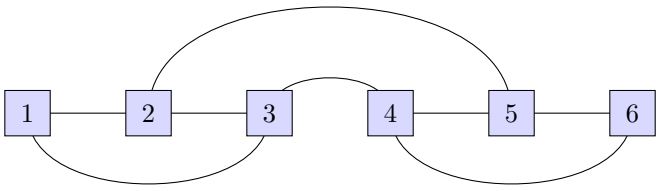


Input

```
6 2
```

Output

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



Wrocławskie krasnale (wroclawskie-krasnale)

Memory limit: 256 MB Time limit: 2.00 s

Wrocławski klub zbieraczy krasnali rozpoczyna noworoczny nabór kandydatów do nowych oddziałów. Celem takiego oddziału jest odnajdywanie kolejnych krasnali i robienie sobie z nimi zdjęć, którymi potem można się pochwalić przed znajomymi.

Po wstępnej analizie chętnych, zarząd klubu wybrał N osób. i-ta z nich jest opisana przez dwie wartości: A_i oraz B_i oznaczające odpowiednio: liczbę krasnali już znalezionych oraz liczbę krasnali, z którymi i-ta osoba chce sobie zrobić zdjęcie. Dla każdego kandydata zachodzi oczywiście $A_i < B_i$ (jeżeli ktoś zrobił już satysfakcjonującą go liczbę zdjęć, to przecież nie będzie się zapisywał do klubu).

Współpraca z osobą, która już zrobiła więcej zdjęć niż my sumarycznie chcemy uzyskać, może wzbudzić zazdrość. Dlatego zarząd zdecydował, że kandydat i nie może być w tym samym oddziale co kandydat j, gdy $A_i \geq B_j$. Z drugiej strony, mottem klubu jest: *"W grupie raźniej"*. Dlatego, jeżeli tylko nie ma przeciwwskazań, żeby chętni byli w jednym oddziale, to muszą być oni w jednym oddziale. Może się jednak zdarzyć, że po przyjęciu kolejnego kandydata ta reguła nie byłaby spełniona. Mając to na uwadze, zarząd będzie rozpatrywał po kolei zgłoszenia każdego z chętnych i akceptował je, gdy tylko będzie mógł. Gdyby jednak przyjęcie nowej osoby oznaczało konieczność podziału na oddziały niezgodne z mottem, to kandydat zostanie odrzucony.

Pomóż zarządowi i napisz program, który dla każdego z kandydatów wypisze `TAK`, gdy zostanie on przyjęty do klubu lub `NIE` w przeciwnym przypadku.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba naturalna N oznaczająca liczbę kandydatów. W kolejnych N wierszach znajduje się opis kandydatów, i-ty z nich zawiera dwie liczby oddzielone pojedynczym odstępem: A_i oraz B_i.

Wyjście

Twój program powinien wypisać N wierszy. W i-tym z nich powinien wypisać `TAK`, gdy i-ty kandydat zostanie przyjęty do klubu lub `NIE` w przeciwnym przypadku.

Ograniczenia

$1 \leq N \leq 500\,000$, $1 \leq A_i < B_i \leq 10^9$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 200$	10
2	$N \leq 5000$	20
3	wszyscy przyjęci kandydaci mogą być w jednym oddziale	10
4	$A_i + 2 = B_i$	10
5	$N \leq 100\,000$	20
6	brak dodatkowych ograniczeń	30

Testy ocen

1. $N = 100000$, $A_i = 2 \cdot i$, $B_i = A_i + 1$
2. $N = 30$, $A_i = ((i \cdot 7) \mod 11) + 3$, $B_i = A_i + ((i \cdot 3) \mod 5) + 1$
3. $N = 30$, $A_i = i$, $B_i = A_i + 5$

Przykład

Input

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

Output

```
TAK
TAK
TAK
TAK
NIE
NIE
NIE
NIE
TAK
TAK
```

Explanation

- Kandydat 1 oczywiście może zostać przyjęty.
- Kandydat 2 i 1 mogą ze sobą współpracować, zatem zgodnie z mottem będą w jednym oddziale.
- Kandydat 3 nie może współpracować z żadnych z poprzednich kandydatów, dlatego będzie w innym oddziale.
- Kandydat 4 również nie może współpracować z dwoma pierwszymi dwoma kandydatami, zaś z kandydatem 3 powinien być w jednej grupie, tak też się dzieje.
- Kandydat 5 może być razem w grupie z kandydatem 1, ale nie może współpracować z kandydatem 2. Pojawia się sprzeczność, zatem kandydat zostaje odrzucony.
- Kandydat 6 powinien być w oddziale z 3, ale nie może współpracować z 4, zostaje odrzucony.
- Kandydat 7 może współpracować z kandydatem 2 i 3. Jednak wtedy musieliby być oni w jednej grupie. Ale kandydat 2 nie może współpracować z kandydatem 3. Dlatego kandydata 7 odrzucamy.
- Kandydat 8 może być w grupie z 3, ale nie może z 4, odrzucamy.
- Kandydat 9 może być w oddziale zarówno z 3, jak i z 4, a nie może z 1 i 2, więc dołączamy go do drugiego oddziału.
- Kandydat 10 nie może współpracować z żadnym już przyjętym kandydatem, dlatego powstanie dla niego oddzielny oddział.

Kryzys termiczny (kryzys-termiczny)

Memory limit: 512 MB Time limit: 6.00 s

`cout<<"Dawno, "; for (int i=0; i<100; i++) cout<<"dawno, "; cout<<`dawno temu we Wrocławiu nastąpił $\mathcal{W}$ielki $\mathcal{K}$ryzys $\mathcal{T}$ermiczny. Dziś w obawie przed ponownym uderzeniem $\mathcal{W}$ielkiego $\mathcal{K}$ryzysu $\mathcal{T}$ermicznego Wrocław wyposażony jest w nowatorskie systemy *konstantywnie* monitorujące stan $\mathbb{W}$rocławskich $\mathbb{Z}$akładów $\mathbb{T}$ermicznych. $\mathbb{W}$rocławskie $\mathbb{Z}$akłady $\mathbb{T}$ermiczne składają się z wielu działów oraz mają ogromną sieć Wodociągów Błyskawicznych↗ ↗. Wodociągi Błyskawiczne↗ ↗ składają się z N segmentów, jednak zanim dowiesz się o tym więcej musisz zostać dokładnie oprowadzony po świecie $\mathbb{W}$rocławskich $\mathbb{Z}$akładów $\mathbb{T}$ermicznych. Chociaż nie zaszkodzi powiedzieć Ci trochę więcej o tym jak działa sieć Wodociągów Błyskawicznych↗ ↗. Sieć Wodociągów Błyskawicznych↗ ↗, jak już zostało powiedziane, składa się z N segmentów. Każdy segment będzie miał określoną przepustowość, która być może będzie zmieniała się z czasem. Nad bezpieczeństem Wrocławian bacznie czuwa ★Zespół Zapobiegania Wielkim Kryzysom Termicznym★, który wykonując skomplikowane obliczenia korzystające nawet z tak skomplikowanych operacji jak *xor* (bitowa alternatywa rozłączna) jest w stanie przewidywać, kontrolować, zapobiegać – wykonywać działania prewencyjne przeciwko kontratakowi $\mathcal{W}$ielkiego $\mathcal{K}$ryzysu $\mathcal{T}$ermicznego. Ta treść już wygląda na tyle paskudnie, że nie mogę na nią patrzeć, myślę więc, że to już czas przejść do jej merytorycznej części.

Dany jest ciąg N liczb naturalnych A_i, który określa przepustowości Wodociągów Błyskawicznych↗ ↗. Zdefinujmy xor zbioru jako xor jego elementów. Napisz program, który wczyta ciąg A i wykona następujące operacje:

- `aktualizacja` X_i Y_i Z_i – ustaw elementy ciągu A_i o indeksach od X_i do Y_i na wartość Z_i,
- `maksymalizacja` X_i Y_i – podaj największy xor podzbioru spośród xorów wszystkich podzirów elementów ciągu A_i o indeksach z przedziału od X_i do Y_i,
- `lustracja` X_i Y_i Z_i – sprawdź czy istnieje podzbiór elementów ciągu A_i o elementach z przedziału od X_i do Y_i, którego xor jest równy wartości Z_i.

Toż to dopiero struktura z klasą.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby naturalne N i Q, oddzielone pojedynczym odstępem i oznaczające odpowiednio długość ciągu A_i i liczbę operacji do wykonania. W drugim wierszu wejścia znajduje się ciąg N liczb naturalnych A_i, pooddzielanych pojedynczymi odstępami. W kolejnych Q wierszach znajdują się opisy operacji zgodne z specyfikacją z treści zadania.

Wyjście

Na wyjście należy wypisać odpowiedzi na zapytania typu `maksymalizacja` i `lustracja` w osobnych wierszach w kolejności, w której pojawiły się one na wejściu. W przypadku zapytania typu `maksymalizacja` należy wypisać jedną liczbę naturalną – największy xor podzbioru spośród xorów wszystkich podzbiorów elementów ciągu A z zadanego przedziału. W przypadku zapytania typu `lustracja` należy wypisać jedno słowo `TAK` lub `NIE` w zależności od tego czy istnieje podzbiór elementów ciągu A z określonego przedziału xorujący się do zadanej wartości.

Ograniczenia

$1 \leq N \leq 100\,000, 1 \leq Q \leq 50\,000, 1 \leq X_i \leq Y_i \leq N, 1 \leq A_i, Z_i \leq 10^9$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 10$, $Q \leq 10$	10
2	$N \leq 1\,000$, $Q \leq 1\,000$	40
3	brak dodatkowych ograniczeń	50

Przykład

Input	Output
5 5 3 7 4 10 1 maksymalizacja 1 4 aktualizacja 3 4 6 maksymalizacja 3 5 sprawdzacja 3 4 10 sprawdzacja 1 3 5	14 7 NIE TAK

;

Naleśniki (nalesniki)

Memory limit: 64 MB Time limit: 1.00 s

Marcelina ma wielką ochotę na naleśniki. Wobec tego zrobiła już ciasto, usmażyła odpowiednio dużo naleśników i właśnie stoi przed bardzo ważnym wyborem – dodatkami smakowymi.

W kuchni Marceliny znajduje się N różnych dodatków smakowych. Każdy z dodatków może zostać użyty na naleśniku w dokładnie trzech możliwych ilościach – wcale, jedna łyżeczka oraz dwie łyżeczki.

Marcelina nie chce przesadzić z ilością dodatków smakowych, ale z drugiej strony nie chce mieć suchego naleśnika. Wobec tego na naleśniku Marceliny powinno się znaleźć dokładnie K łyżeczek dodatków smakowych.

Podczas dobierania idealnej kompozycji dodatków, w głowie Marceliny zaczęło kiełkować pewne pytanie wynikające z chęci podzielenia się naleśnikami z dwiema przyjaciółkami – "jaka jest reszta z dzielenia przez 3 liczby sposobów, na które mogę wybrać dodatki smakowe dla ustalonych N oraz K?".

Z uwagi na to, że wielka ochota na naleśniki może przyjść Marcelinie jeszcze w przyszłości, Marcelina będzie miała do Ciebie Q pytań powyższej postaci.

Pomóż Marcelinie w zaspokojeniu naleśnikowej ciekawości – napisz program, który wczyta pytania Marceliny i wypisze odpowiedzi na wszystkie z nich.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita Q oznaczająca liczbę pytań Marceliny. Po niej następuje Q opisów pytań, po jednym w wierszu.

W każdym z kolejnych Q wierszy znajdują się dwie nieujemne liczby całkowite N_i oraz K_i oddzielone pojedynczym odstępem, oznaczające liczbę dostępnych dodatków smakowych oraz pojemność naleśnika mierzoną w łyżeczkach.

Wyjście

Na wyjściu powinno się znaleźć dokładnie Q wierszy. Wiersz i-ty powinien zawierać jedną nieujemną liczbę całkowitą będąca resztą z dzielenia przez 3 liczby sposobów, na które Marcelina może wybrać dodatki smakowe dla parametrów i-tego pytania.

Ograniczenia

$1 \leq Q \leq 100\,000$, $1 \leq N_i \leq 10^{15}$, $0 \leq K_i \leq 2N_i$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N_i \leq 1\,000$	20
2	N_i jest postaci $2^\varepsilon 3^m$ dla m naturalnego i $\varepsilon \in \{0, 1\}$	10
3	$N_i \leq 1\,000\,000$, $Q = 1$	20
4	brak dodatkowych ograniczeń	50

Przykład

Input

```
5
3 4
2 1
4 0
1 2
4 3
```

Output

```
0
2
1
1
1
```

Explanation

- Liczba sposobów to 6 – $\{1, 1, 2, 3\}$, $\{1, 2, 2, 3\}$, $\{1, 2, 3, 3\}$, $\{1, 1, 2, 2\}$, $\{2, 2, 3, 3\}$, $\{1, 1, 3, 3\}$, zatem wynikiem jest $6 \bmod 3 = 0$.
- Liczba sposobów to 2 – $\{1\}$, $\{2\}$, zatem wynikiem jest $2 \bmod 3 = 2$.
- Liczba sposobów to 1 – $\emptyset$, zatem wynikiem jest $1 \bmod 3 = 1$.
- Liczba sposobów to 1 – $\{1, 1\}$, zatem wynikiem jest $1 \bmod 3 = 1$.
- Liczba sposobów to 16 – 4 sposoby, w których dodatki są parami różne i 12 sposobów, w których jeden z dodatków występuje podwójnie, zatem wynikiem jest $16 \bmod 3 = 1$.

Prostokątna pizza **(prostokatna-pizza)**

Memory limit: 64 MB Time limit: 1.00 s

Pewien mądry człowiek powiedział kiedyś, że “nawet słaba pizza to jednak pizza”. Dlatego też Bartosz i Marcel po udanym obozie udali się do wykwintnej restauracji właśnie w celu spożycia tego włoskiego przysmaku.

Niestety, jak się okazało, wybór pizzy nie będzie łatwy. W wybranej przez nich restauracji każdego dnia przygotowywana jest jedna długa prostokątna pizza podzielona na segmenty, a zamówienie odbywa się poprzez wybranie spójnego przedziału segmentów pizzy.

Każdy segment pizzy ma określony smak, który będziemy oznaczać małymi literami alfabetu łacińskiego. Wobec tego pizzę o N segmentach możemy reprezentować jako słowo długości N złożone z małych liter alfabetu łacińskiego.

Marcel i Bartosz chcą złożyć jedno zamówienie, które zmaksymalizuje *współczynnik klasteryzacji*, który jest przez nich definiowany jako **różnica pomiędzy liczbą wystąpień najczęściej i najrzadziej występującego w tym fragmencie smaku segmentu pizzy**. Naturalnie, najrzadszy występujący w danym fragmencie smak segmentu pizzy musi w nim wystąpić co najmniej jeden raz.

Pomóż Bartoszowi i Marcelowi! Napisz program, który wczyta słowo opisujące pizzę, wyznaczy największy możliwy współczynnik klasteryzacji spójnego przedziału segmentów tej pizzy i wypisze wynik na standardowe wyjście.

Wejście

W pierwszym (i jedynym) wierszu wejścia znajduje się jedna dodatnia liczba całkowita N, oznaczającą długość słowa. Drugi wiersz zawiera słowo składające się z N małych liter alfabetu łacińskiego.

Wyjście

W pierwszym (i jedynym) wierszu wyjścia powinna się znaleźć jedna liczbę całkowita równa maksymalnej wartości różnicy między liczbą wystąpień najczęściej i najrzadziej występującej litery, jaką możemy znaleźć w pewnym spójnym fragmencie danego słowa.

Ograniczenia

$1 \leq N \leq 1\,000\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 100$	30
2	$N \leq 100\,000$	30
3	brak dodatkowych ograniczeń	40

Przykład

Input

```
10
aabbaaabab
```

Output

```
3
```

Explanation

Podsłowo `aaaba` ma współczynnik klasteryzacji równy $\#\mathtt{a} - \#\mathtt{b} = 4 - 1 = 3$ i żadne inne podsłowo nie ma większego współczynnika klasteryzacji.
Również podsłowa `aabbaaa` i `aabbaaaba` mają współczynnik klasteryzacji równy 3.

Najlepsze Brownie **(brownie)**

Memory limit: 512 MB Time limit: 4.00 s

Ewa robi najlepsze Brownie na świecie. Praktykuje sztukę przygotowywania najznamienitszego ciasta na świecie już od wielu lat i doszła w niej do istnej perfekcji. Jednym z wielu sekretów przygotowywania swoistego arcydzieła jest umiejętne rozprowadzenie masła po foremce od ciasta. Ewa postanowiła zrobić dla swojego informatyka najlepsze Brownie na świecie. Jednak ten, zobaczywszy ciasto, stwierdził, że jest jakieś dziwne, bo okrągłe. Ewa jednak nie dała za wygraną i postanowiła przygotować dla swojego informatyka jeszcze jedno ciasto, jednak tym razem w kształcie drzewa – oczywiście informatycznego.

Ewa przygotowała już piękną foremkę składającą się z N wierzchołków i $N-1$ krawędzi między nimi – foremkę w kształcie drzewa. Wierzchołki foremki numerowane są kolejnymi liczbami naturalnymi od 1 do N.

Niestety z tak śmiałym planem wiążą się pewne problemy, już samo rozprowadzenie masła po foremce w kształcie drzewa okazało się problematyczne. Ewa położyła dwie kostki masła w wierzchołkach drzewa o numerach X i Y, teraz musi rozprowadzić masło po całej foremce – wszystkich wierzchołkach drzewa. Ewa może rozsmarować masło z wierzchołka A do wierzchołka B tylko wtedy, gdy w wierzchołku A już znajduje się masło oraz wierzchołki A i B są połączone krawędzią. Rozsmarowanie zajmuje pomijalnie mało czasu, jednak zanim masło będzie mogło być rozsmarowane ponownie z wierzchołka A lub z wierzchołka B, Ewa będzie musiała poczekać minutę. Jest to bardzo ważne podczas rozsmarowywania masła – gdyby nie trzymać się tej reguły ciasto mogłoby nie wyjść idealne, na co Ewa nie może sobie pozwolić. Ewa może w ciągu każdej minuty rozsmarować masło z wielu wierzchołków do wielu innych, jednak w ciągu każdej minuty z każdego wierzchołka masło może zostać rozsmarowane tylko raz i do każdego wierzchołka masło może zostać rozsmarowane tylko raz.

Napisz program, który wczyta opis foremki ciasta i wyznaczy minimalny czas potrzebny Ewie na rozprowadzenie masła po całej foremce.

Wejście

W pierwszym wierszu wejścia znajdują się trzy liczby naturalne N, X i Y pooddzielane pojedynczymi odstępami i oznaczające odpowiednio liczbę wierzchołków foremki i indeksy wierzchołków, w których początkowo znajduje się masło. W kolejnych $N-1$ wierszach znajdują się opisy krawędzi.W i-tym z nich znajdują się dwie liczby naturalne A_i i B_i, oddzielone pojedynczymi odstępami i oznaczające, że wierzchołki o numerach A_i i B_i są połączone krawędzią.

Wyjście

W pierwszym (jedynym) wierszu wyjścia należy wypisać minimalny czas potrzebny Ewie na rozprowadzenie masła po całej foremce na ciasto.

Ograniczenia

$1 \leq N \leq 300\,000$, $1 \leq A_i,\ B_i,\ X,\ Y \leq N$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 10$	10
2	$N \leq 1\,000$	20
3	Graf jest ścieżką	10
4	Graf został wygenerowany losowo	20
5	brak dodatkowych ograniczeń	40

Przykład

Input

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

Output

```
2
```

Explanation

Ewa w pierwszej minucie może rozprowadzić masło na wierzchołki o numerach 3 i 5, a w drugiej na 4 i 6.

Input

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

Output

```
4
```

3 Omówienia zadań

Artur Kraska
Autor omówienia i paczki

Domek z kart

Pierwsze spostrzeżenie jakiego możemy dokonać w zadaniu jest takie, że karty z domku zawsze będziemy zabierać parami. Oznacza to, że każdą parę kart możemy traktować jako jedną liczbę – sumę ich wartości. Zatem parametr K możemy od razu podzielić na 2.

Karty tworzące domek mają strukturę pełnego drzewa binarnego, czyli każda para albo jest liściem (karty te stoją na stole), albo jest wierzchołkiem, który ma dwa poddrzewa (jest to para kart, która stoi na dwóch innych parach kart).

Na początku warto spostrzec, że drzewo takie łatwo przetrzymać w tablicy, gdzie synowie wierzchołka znajdującego się w komórce i będą umieszczeni w komórkach o numerach $2i$ oraz $2i + 1$.

Zatem naszym zadaniem jest wybrać K wierzchołków z drzewa binarnego o maksymalnej sumie, z zastrzeżeniem, że wierzchołek możemy zabrać dopiero gdy wzięliśmy już wierzchołek znajdujący się nad nim (oczywiście nie tyczy się to korzenia).

Rozwiązanie wykładnicze, korzystające z backtrackingu

Napiszmy rozwiązanie rekurencyjne. Nasza funkcja będzie brała dwa parametry – numer wierzchołka v, oraz parametr k, oznaczający, że chcemy wziąć k par kart z poddrzewa v. Funkcja będzie zwracała jaki maksymalny wynik jesteśmy w stanie uzyskać w tej sytuacji. Będzie ona brała pod uwagę dwie opcje – albo nie bierzemy nic i zwracamy 0, albo bierzemy wartość z korzenia i $k-1$ elementów z dwóch poddrzew. W tej drugiej sytuacji mamy do sprawdzenia k opcji – gdy weźmiemy z lewego poddrzewa i wierzchołków, to z prawego możemy wziąć co najwyżej $k - i - 1$. Wyniki dla poddrzew możemy wyliczyć za pomocą tej samej funkcji rekurencyjnej.

Za takie rozwiązanie można było dostać nawet 50 punktów!

Rozwiązanie wzorcowe o złożoności $\mathcal{O}\left(2^N \cdot K^2\right)$

Przyglądając się dokładniej poprzedniemu rozwiązaniu, łatwo zauważyć, że wiele wywołań rekurencyjnych wykonujemy wielokrotnie, co istotnie spowalnia program. Możemy zatem zauważyć, że wystarczy dla każdego wierzchołka wyliczyć tylko K różnych wartości i je zapamiętać. Daje to łącznie $2^N \cdot K$ stanów, a każdy z nich możemy policzyć w czasie $\mathcal{O}(K)$, metodą jak w poprzednim podpunkcie.

Zatem aby zaimplementować to rozwiązanie mamy dwie opcje:

- przeiterować się po wszystkich wierzchołkach od liści do korzenia i dla każdego z nich wyliczyć wspomniane K stanów,
- albo zaimplementować dokładnie rozwiązanie z poprzedniego podpunktu, ale dodać zapamiętywanie wartości stanów, które już kiedyś były wyliczone.

Dokładniejsza analiza złożoności powyższego rozwiązania

Załóżmy, że opisane powyżej rozwiązanie, podczas liczenia wyników dla niższych poddrzew, nie rozważa wartości parametru K większych niż rozmiar danego poddrzewa (co jest całkiem logiczne). Okazuje się, że w tym rozwiązaniu złożoność wynosi $\mathcal{O}\left(2^N \cdot K\right)$.

Na początek zróbmy małe spostrzeżenie. Wyobraźmy sobie, że mamy pełne drzewo binarne (takie jak w treści), które ma r wierzchołków. W każdym jego poddrzewie o rozmiarze s będziemy wykonywać $\left(\frac{s}{2}\right)^2$ operacji (tyle potrzebujemy czasu na obliczenie wszystkich stanów w bardzo niskich poddrzewach).

Sumując całkowity koszt obliczeń w takim poddrzewie otrzymujemy:

- w korzeniu zapłacimy koszt $\frac{r^2}{2^2}$,
- w każdym wierzchołku na drugim poziomie zapłacimy $\frac{r^2}{2^4}$, zatem za oba zapłacimy $\frac{r^2}{2^3}$,
- analogicznie, na trzecim poziomie zapłacimy sumarycznie $2^2 \cdot \frac{r^2}{2^6} = \frac{r^2}{2^4}$,
- a więc na i-tym poziomie zapłacimy $\frac{r^2}{2^{i+1}}$.

Łatwo zauważyć, że teraz suma na wszystkich poziomach w takim drzewie wyniesie $\mathcal{O}\left(r^2\right)$. Całkowitej złożoności możemy dowieść zauważając poniższe fakty.

- Pośród najniższych $\log k$ poziomów drzewa mamy $\mathcal{O}\left(\frac{2^N}{K}\right)$ poddrzew. Każde z nich ma rozmiar $\mathcal{O}\left(K\right)$, więc każde wykona sumarycznie $\mathcal{O}\left(K^2\right)$ operacji.
- Wierzchołki powyżej zajmują poziomy od 1 do $N - \log k$. Jest ich $\mathcal{O}\left(\frac{2^N}{K}\right)$, a każdy wykona maksymalnie $\mathcal{O}\left(K^2\right)$ operacji.

Sumując powyższe wartości otrzymujemy ostateczny dowód, że całkowita złożoność wynosi $\mathcal{O}\left(2^N \cdot K\right)$.

Artur Kraska
Autor omówienia i paczki

Które zgłoszenie było pierwsze?

W tym zadaniu mieliśmy podaną informację, że w pewnym miejscu jest ustawiona pewna permutacja P elementów od 1 do N, lecz nie mieliśmy bezpośredniego dostępu do jej elementów. Mieliśmy za to możliwość wywołania dwóch funkcji:

- `f(i, j, d)` – sprawdź czy $P[i] - P[j]$ jest podzielne przez d,
- `g(i, i)` – sprawdź czy $P[i] < P[j]$.

Naszym zadaniem było znalezienie na której pozycji znajduje się element 1. Mieliśmy jednak zastrzeżenie – funkcja `g` miała zostać wykonana minimalną możliwą liczbę razy.

Pierwszym spostrzeżeniem w tym zadaniu było to, że funkcję `g` będziemy musieli wykonać dokładnie raz (poza trywialnym przypadkiem gdy $N = 1$). Wynika to z faktu, że za pomocą funkcji `f` jesteśmy w stanie znaleźć elementy 1 i N, gdyż jako jedyne są one oddalone od siebie o $N - 1$. Jednakże funkcja ta jest symetryczna, czyli w żaden sposób nie jest nam w stanie odpowiedzieć który element jest mniejszy. Korzystając z tego spostrzeżenia przyjrzyjmy się możliwym rozwiązaniom.

Rozwiązanie o złożoności $\mathcal{O}(N^2)$

Z powyższym spostrzeżeniem pomysł jest bardzo prosty – przejrzyjmy każdą parę elementów i sprawdźmy czy są one odległe od siebie o $N - 1$. Jeśli tak, to za pomocą funkcji `g` sprawdźmy który jest mniejszy i zwróćmy go.

Rozwiązanie takie dawało aż 36 punktów!

Rozwiązanie o złożoności $\mathcal{O}\left(N\sqrt{N}\right)$

Jak to często bywa z prostymi rozwiązaniami o złożoności kwadratowej, warto spróbować czy nie możemy podzielić naszych danych na bloki, obliczyć czegoś dla tych bloków, a potem połączyć informacji ze wszystkich bloków w jedną całość.

Niech B będzie wybranym przez nas pewnym parametrem. Za pomocą niego spróbujmy zrealizować następujący pomysł:

- Podzielmy wszystkie elementy na B zbiorów. Każdy z nich będzie zawierał elementy, które dają taką samą resztę modulo B. Można to zrobić w taki sposób, że dla każdego elementu i, mając dane pewne kubełki z elementami, sprawdzimy czy element i daje taką samą resztę co pierwszy element z kubełka. Jeżeli nie ma takiego kubełka, to tworzymy nowy. Dla wszystkich elementów zajmie to czas $\mathcal{O}(N \cdot B)$.
- Dla każdego kubełka (których jest B) wiemy, że jest w nim $\mathcal{O}(N/B)$ elementów. Zatem w czasie $\mathcal{O}\left((N/B)^2\right)$ możemy znaleźć dwa najdalsze z nich. Całość zajmie nam $\mathcal{O}\left(N^2/B\right)$.
- Skoro wybraliśmy po dwa elementy z każdego kubełka, a wiemy, że są pośród nich elementy 1 i N, to możemy je po prostu znaleźć, tak jak w rozwiązaniu o złożoności kwadratowej. Wszystkich tych elementów jest $2 \cdot B$, zatem zajmie to czas $\mathcal{O}(B^2)$.

Zatem całe to rozwiązanie ma złożoność $\mathcal{O}(N \cdot B + N^2/B + B^2)$, co dla $B = \sqrt{N}$ daje ostateczną złożoność $\mathcal{O}\left(N\sqrt{N}\right)$.

Za takie rozwiązanie można było zdobyć około 68 punktów.

Rozwiązanie o złożoności $\mathcal{O}(N \log N)$

Spróbujmy jeszcze ulepszyć powyższe rozwiązanie. Naszym celem będzie znalezienie dwóch skrajnych elementów zbioru – minimum i maksimum (nie musimy wiedzieć który jest który, zrobimy to na końcu jednym wywołaniem funkcji g).

Teraz, zamiast dzielić cały zbiór na kilka podzbiorów, podzielmy go tylko na dwa – elementy parzyste i nieparzyste (zauważmy, że możemy to zrobić, lecz nie będziemy wiedzieli który zbiór będzie tym parzystym, a który nieparzystym). Gdybyśmy umieli znaleźć po dwa skrajne elementy tych dwóch zbiorów, łatwo (w czasie stałym) moglibyśmy znaleźć skrajne elementy naszego całego zbioru. Odpowiedź nasuwa się sama – rekurencja.

Co powinna w takim razie umieć nasza funkcja? Będzie ona dostawała jako parametry pewną liczbę d oraz pewien zbiór S, o którym wiemy, że zawiera on wszystkie elementy dające pewną resztę x modulo d. Jak znaleźć skrajne elementy tego zbioru? Wystarczy:

- podzielić zbiór S na dwa zbiory – takie, które dają tą samą resztę przy dzieleniu przez $2 \cdot d$,
- na każdym z tych zbiorów możemy wywołać funkcję rekurencyjną znajdującą dwa skrajne elementy,
- a na końcu, gdy mamy dane tylko 4 elementy, wystarczy znaleźć pośród nich skrajne elementy naszego zbioru.

Zauważmy, że nasze zbiory zawsze będą się dzielić na dwa prawie równoliczne podzbiory (mogą się różnić o 1). Oznacza to, że nasze rozwiązanie ma dokładnie taką samą złożoność jak sortowanie przez scalanie (mergesort), czyli $\mathcal{O}(N \log N)$.

Marcel Szelwiga
Autor omówienia i paczki

Piramida

W tym zadaniu mieliśmy obliczyć maksymalną wysokość kolumny jaką mogliśmy uzyskać dokładając K bloczków, zgodnie z podaną regułą. Pierwszą obserwacją w tym zadaniu jest to, że jeśli potrafimy ułożyć konstrukcję, której każdy bloczek spełnia wymagania, to kolejność dokładania bloczków co do zasady nie ma znaczenia.

Rozwiązanie dla podzadania $K, N \leqslant 20$

Przykładowym rozwiązaniem jest oszacowanie maksymalnego wyniku przez $K + \max_{1 \leqslant i \leqslant n} A_i$, a następnie sprawdzanie od największego z możliwych, który jest możliwy do uzyskania. Sprawdzenie takie może polegać na wybraniu każdej kolumny jako najwyższą, a następnie iterowaniu się w prawo i w lewo i sprawdzaniu ile bloczków brakuje w kolejnych kolumnach w obie strony.

Rozwiązanie dla podzadania $N \leqslant 1\,000$

Zauważmy, że faza sprawdzania wyniku działała w czasie $\mathcal{O}\left(N^2\right)$ – iterowanie się po kolumnie, co do której określamy, że to ona będzie największa, a następnie iterowanie się w obie strony w celu sprawdzenia ile bloczków brakuje. Zauważmy, że jeśli jesteśmy w stanie zbudować piramidę o maksymalnej wysokości X, to jesteśmy też w stanie zbudować piramidę o wysokości $X-1$, zatem możemy użyć metody wyszukiwania binarnego względem wyniku i otrzymać złożoność $\mathcal{O}\left(N^2 \cdot \log(A_i + K)\right)$.

Rozwiązanie wzorcowe

W celu otrzymania rozwiązania zadania, potrzebujemy jeszcze usprawnić fazę sprawdzania czy wynik jest poprawny. Załóżmy więc, że znamy już wynik i wynosi on H. Określmy $B_l(x)$ (*blokadę lewą*) jako skrajnie oddaloną na lewo od x pozycję, na którą koniecznie musimy dołożyć bloczek, aby w kolumnie x stało H bloczków. Analogicznie definiujemy $B_p(x)$ jako *blokady prawe*.

Gdybyśmy potrafili łatwo wyznaczyć wartości B_l i wartości B_p, to rozwiązanie sprowadzałoby się do wyprowadzenia odpowiednich zależności określających ile bloczków powinno znajdować się na przedziale od B_l i do B_p, a następnie odjęcia od ich sumy liczby bloczków już znajdujących się na tym przedziale.

Wartości B_l możemy wyznaczać następująco: gdy w kolumnie x mamy y bloczków, to kolumna x będzie blokadą dla kolumn o numerach większych bądź równych $x + (H - y)$. Obliczenie wartości B_l będzie sprowadzało się do obliczenia dla każdej kolumny, dla jakich kolumn będzie ona blokadą. Wartość tę można rozprowadzać obliczając maksima prefiksowe. Analogicznie obliczamy wartości B_p i mamy już wszystko co potrzebne do rozwiązania zadania.

Bartosz Chomiński
Autor omówienia i paczki

Blokada łańcuchowa

Rozwiązanie dla podzadania $A_1 = A_2 = \cdots = A_N = 1$

Skoro wszystkie elementy ciągu A_i są równe, to odpowiedź na zapytanie zależy tylko od wartości $R_i - L_i$. Proste rachunki pozwalają się przekonać, że wyniki dla kolejnych wartości $R_i - L_i$ wynoszą odpowiednio:

$R_i - L_i$	wynik
0	$\frac{1}{1}$
1	$\frac{2}{1}$
2	$\frac{3}{2}$
3	$\frac{5}{3}$
4	$\frac{8}{5}$

Można stąd zaobserwować pewną narzucającą się prawidłowość – wynikiem dla $k = R_i - L_i$ jest $\frac{F_{k+1}}{F_k}$, gdzie F_n to n-ta liczba Fibonacciego.

Dowód tej obserwacji najprościej jest przeprowadzić przy pomocy indukcji matematycznej. Dociekliwemu Czytelnikowi zostawiamy jako wskazówkę rachunek

$$1 + \frac{1}{\frac{F_{k+1}}{F_k}} = 1 + \frac{F_k}{F_{k+1}} = \frac{F_{k+1} + F_k}{F_{k+1}} = \frac{F_{k+2}}{F_{k+1}}.$$

Ponadto wszystkie ułamki postaci $\frac{F_{k+1}}{F_k}$ są nieskracalne, co również łatwo udowodnić indukcyjnie i Dociekliwy Czytelnik z pewnością będzie w stanie szybko się o tym przekonać.

Wobec tego pozostaje tylko wyznaczyć odpowiednio długi fragment początkowy reszt z dzielenia przez $1\,000\,000\,007$ kolejnych wyrazów ciągu Fibonacciego i wypisywać odpowiednie jego wyrazy.

Złożoność obliczeniowa wyniesie $\mathcal{O}(N + Q)$.

Rozwiązanie dla podzadania $N, Q \leqslant 1\,000$

Zacznijmy od pewnego lematu.

Lemat. Niech k, m, n będą liczbami naturalnymi i niech $\frac{m}{n}$ będzie ułamkiem nieskracalnym. Wówczas $k + \frac{1}{\frac{m}{n}} = \frac{km+n}{m}$ również jest ułamkiem nieskracalnym.

Dowód. Przeprowadźmy prosty rachunek:

$$\mathrm{NWD}(km + n, m) = \mathrm{NWD}((k-1)m + n, m) = \cdots = \mathrm{NWD}(m + n, m) = \mathrm{NWD}(n, m) = 1.\blacksquare$$

Lemat ten stwierdza, że nie tylko w pierwszym opisanym podzadaniu nie musimy się przejmować skracaniem wynikowych ułamków, ale nawet, że w całym zadaniu nie musimy się tym przejmować.

Uzbrojeni w powyższy wniosek możemy zaimplementować działające wprost rozwiązanie, które dla każdego zadanego przedziału buduje żądany ułamek łańcuchowy w sposób jawny, licząc kolejne liczniki i mianowniki.

Złożoność obliczeniowa wyniesie $\mathcal{O}(NQ)$.

Pełne rozwiązanie

Zauważmy, że operację $\frac{x}{y} \mapsto z + \frac{1}{\frac{x}{y}} = \frac{zx+y}{x}$ możemy przedstawić za pomocą macierzy 2×2. Istotnie mamy

$$\begin{pmatrix} z & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} zx+y \\ x \end{pmatrix},$$

a więc otrzymaliśmy licznik i mianownik nowego ułamka.

Oznaczmy przez M_i macierz $\begin{pmatrix} A_i & 1 \\ 1 & 0 \end{pmatrix}$. Wówczas każde zapytanie możemy sprowadzić do obliczenia wektora

$$M_{L_i} \cdot M_{L_{i+1}} \cdot \ldots \cdot M_{R_i} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Stąd pozostaje znaleźć sposób na liczenie iloczynów macierzy na przedziale. Można do tego wykorzystać drzewo przedziałowe lub tablicę iloczynów sufiksowych i iloczynów odwrotności sufiksowych. Implementację pierwszego sposobu zostawiamy jako proste ćwiczenie i skupimy się na drugim sposobie.

Najpierw wyznaczmy macierz odwrotną do macierzy M_i. Potrzebujemy takiej macierzy M_i^{-1}, by

$$M_i \cdot M_i^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Rozwiązując układ równań liniowych lub wyznaczając rozwiązanie w jakikolwiek inny sposób otrzymujemy

$$M_i^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -A_i \end{pmatrix}.$$

Niech

$$P_i = M_i \cdot M_{i+1} \cdot M_{i+2} \cdot \ldots \cdot M_N$$

oraz niech

$$Q_i = M_N^{-1} \cdot M_{N-1}^{-1} \cdot \ldots \cdot M_i^{-1}.$$

Wówczas mamy

$$M_{L_i} \cdot M_{L_i+1} \cdot \ldots \cdot M_{R_i} = M_{L_i} \cdot M_{L_i+1} \cdot \ldots \cdot M_{R_i} \cdot \underbrace{M_{R_i+1} \cdot \ldots \cdot M_N \cdot M_N^{-1} \cdot M_{N-1}^{-1} \cdot \ldots \cdot M_{R_i+1}^{-1}}_{\text{identyczność}} = P_{L_i} \cdot Q_{R_i+1}.$$

Złożoność obliczeniowa wyniesie $\mathcal{O}(N+Q)$.

Uwaga. Mnożenie macierzy nie jest przemienne.

Uwaga. Czysty kod do tego zadania powinien wykorzystywać programowanie obiektowe (`struct` lub `class`), by rozsądnie obsłużyć operacje na macierzach i wektorach.

Bartosz Chomiński, Marcel Szelwiga
Autor omówienia

Marcel Szelwiga
Autor paczki

Psychotest

Rozwiązanie dla $N \leqslant 500$ i $\sum |S_i| \leqslant 10\,000$

Rozważmy każdy z N prefiksów podanego zbioru słów do pokrycia oddzielnie.

Zauważmy, że pokrycie całego zbioru $\{S_1, S_2, \ldots, S_i\}$ wymaga w szczególności pokrycia słowa S_1. Zatem w szukanym *zbiorze generatorów* musi znaleźć się prefiks słowa S_1.

Wobec tego wybierzmy najkrótszy prefiks słowa S_1, który jednocześnie nie jest prefiksem żadnego ze słów $S_{i+1}, S_{i+2}, \ldots, S_N$. Łatwo zauważyć, że w jednym z optymalnych rozwiązań będzie prefiks słowa S_1 spełniający powyższy warunek. Oznaczmy ten prefiks słowa S_1 jako P_1.

Następnie rozważmy pierwsze słowo z ciągu $S_1, S_2, \ldots, S_i$, które nie jest pokrywane przez P_1. Analogicznie jak poprzednio, znajdźmy najkrótszy prefiks tego słowa, który nie pokrywa żadnego słowa spośród $S_{i+1}, \ldots, S_N$.

Kontynuujemy ten proces do wyczerpania zbioru słów do pokrycia $\{S_1, S_2, \ldots, S_i\}$, a wynikiem będzie liczba wybranych prefiksów.

Złożoność obliczeniowa wyniesie $\mathcal{O}\left(N(\sum |S_i|)^2\right)$, ale przez odpowiednio częste używanie instrukcji `break` jesteśmy w stanie zredukować faktyczny czas działania programu do akceptowalnego.

Rozwiązanie, gdy podane słowa są posortowane leksykograficznie

Zauważmy, że nigdy nie wystąpi sytuacja, w której na wejściu pojawi się prefiks pewnego słowa i to słowo, ponieważ w treści zostało zagwarantowane istnienie rozwiązania.

Optymalny zbiór generatorów można znaleźć techniką podobną do tzw. digit DP.

Niech $\{S_1, S_2, \ldots, S_i\}$ będzie zbiorem słów, dla którego chcemy znaleźć optymalny zbiór generatorów.

Na początek rozważmy zbiór $\mathcal{P}_1$ jednoliterowych prefiksów słów ze zbioru $\{S_1, S_2, \ldots, S_i\}$.

Załóżmy, że istnieje prefiks z $\mathcal{P}_1$, który pokrywa słowo ze zbioru $\{S_{i+1}, S_{i+2}, \ldots, S_N\}$. Oznaczmy go jako r_1. W przeciwnym razie, tzn. gdy nie istnieje taki prefiks, zbiór $\mathcal{P}_1$ jest optymalnym rozwiązaniem.

Następnie rozważmy zbiór słów $\{S_j : j \leqslant i \text{ oraz } S_j[1] = r_1\}$. Teraz rozważmy zbiór $\mathcal{P}_2$ dwuliterowych prefiksów słów z powyższego zbioru. Postępujemy analogicznie – jeżeli jakiś element r_2 z $\mathcal{P}_2$ pokrywa słowo spośród $S_{i+1}, S_{i+2}, \ldots, S_N$, to rozważamy zbiór $\{S_j : j \leqslant i \text{ oraz } S_j[1..2] = r_2\}$ i wyznaczamy $\mathcal{P}_3$ (i potencjalnie dalsze $\mathcal{P}_k$); jeżeli nie, to $\mathcal{P}_1 \cup \mathcal{P}_2$ jest optymalnym rozwiązaniem.

Zauważmy, że opisany wyżej proces zawsze się skończy, ponieważ zbiór $\mathcal{P}_{|S_i|+1}$ będzie pusty dla dowolnego i.

Wydajna implementacja tego rozwiązania może opierać się np. na wyznaczeniu dla wszystkich możliwych prefiksów wszystkich możliwych słów S_i przedziałów, które te prefiksy pokrywają i wobec tego efektywnym wykonywaniu procesu opisanego wyżej.

Złożoność obliczeniowa wyniesie $\mathcal{O}\left(A \cdot \sum |S_i|\right)$, gdzie A jest rozmiarem alfabetu (tu 26).

Rozwiązanie wzorcowe

Przypomnijmy sobie jaka struktura potrafi w łatwy sposób utrzymywać słowa oraz w jakiś sposób nadaje określoną strukturę zbiorowi słów (względem prefiksów tych słów). Chodzi nam oczywiście o drzewo trie. Zbudujmy zatem drzewo trie dla podanych słów, zauważmy, że gdy jakieś słowo S ma pokrywać słowo P, to wierzchołek reprezentujący słowo S musi leżeć na ścieżce od korzenia do wierzchołka reprezentującego słowo P. Zatem konieczność pokrywania słowa S_i możemy reprezentować poprzez *aktywację* wierzchołków w drzewie trie. A odpowiedź na zapytanie jako znalezienie minimalnej liczby węzłów pokrywającej tylko wierzchołki aktywne, które nie pokryją żadnego wierzchołka, który ma zostać aktywowany w przyszłości. Zadanie sprowadza się zatem do następujących operacji:

- Aktywuj wierzchołek w drzewie trie.
- Oblicz ile wierzchołków jest wymaganych do pokrycia wierzchołków aktywnych.

Załóżmy, że mamy już jakieś pokrycie $\mathcal{C}$ – zbiór wierzchołków. Zauważmy, że gdy dla wierzchołka u w drzewie trie każde jego dziecko należy do $\mathcal{C}$, to możemy zastąpić wszystkie dzieci wierzchołka u, wierzchołkiem u, pokrywane wierzchołki wciąż będą pokryte a rozmiar rozmiar pokrycia $\mathcal{C}$, zmniejszy się gdy wierzchołek u ma co najmniej dwójkę dzieci.

Aktywację wierzchołka możemy przeprowadzić dodając go do pokrycia, a następnie wystarczy wykonywać poprzednie podmiany. Łatwo zauważyć, że prowadzi to do rozwiązania optymalnego. Jak szybko sprawdzać warunek podmiany? Zauważmy, że podmianę będziemy mogli wykonać tylko dla rodzica ostatnio dodanego wierzchołka. Możemy więc po każdej wykonanej operacji wykonywać ciąg podmian tak długo jak możemy. Zauważmy, że przy efektywnej implementacji uzyskamy złożoność $\mathcal{O}(\sum |S_i|)$. Każdy wierzchołek zostanie dodany i usunięty z pokrycia tylko raz.

Artur Kraska
Autor omówienia i paczki

Mapa Bałkanów

W zadaniu mamy dany rysunek – kwadrat, którego piksele są zamalowane albo na czarny, albo na biały kolor. Jest on podany w skompresowanej postaci, jako drzewo czwórkowe, które jest zdefiniowane tak, że:

- albo cały kwadrat jest biały (oznaczony jako `0`),
- albo cały kwadrat jest czarny (oznaczony jako `1`),
- albo kwadrat jest podzielony na 4 mniejsze kwadraty, gdzie każdy z nich jest zaprezentowany rekurencyjnie, za pomocą mniejszego drzewa czwórkowego.

Naszym zadaniem jest policzyć ile spójnych czarnych obszarów zawiera nasz kwadrat oraz jaki jest rozmiar największego z nich (modulo $10^9 + 7$).

Wczytanie drzewa

W tym zadaniu już samo wygenerowanie struktury drzewa jest lekko problematyczne. Jednakże okazuje się, że można to łatwo zrobić za pomocą rekurencyjnej funkcji, która

- jeśli dane drzewo ma reprezentację `0` lub `1`, to tworzy jeden wierzchołek,
- za to w sytuacji gdy wierzchołek ma reprezentację `4`, wykonuje czterokrotnie wczytanie drzewa, po czym ustawia je wszystkie jako odpowiednich synów naszego wierzchołka.

Rozwiązanie o złożoności $\mathcal{O}\left(4^M\right)$

Dla $M \leqslant 10$, mając już wczytane drzewo, możemy łatwo napisać rozwiązanie brutalne. A mianowicie, nasz rysunek będzie miał maksymalnie wymiary 1024×1024. Zatem możemy po prostu na podstawie drzewa zamalować osobno wszystkie piksele, po czym na każdym z nich wywołać algorytm DFS (albo Union Find), zliczający jego rozmiar oraz ilość takich obszarów.

Takie rozwiązanie dostawało 30 punktów.

Zliczenie obszarów dla większych przypadków

Aby zliczyć obszary, potrzebujemy wyznaczyć krawędzie pomiędzy czarnymi kwadratami, które sąsiadują ze sobą. Można to zrobić również rekurencyjnie.

Drzewo składające się z jednego (czarnego/białego) kwadratu oczywiście nie ma wewnątrz siebie żadnej krawędzi. Weźmy zatem kwadrat o reprezentacji `4`, a dla uproszczenia oznaczmy cztery jego pod-kwadraty jako LG, PG, LD i PD. Wszystkie krawędzie łączące czarne obszary wewnątrz tego kwadratu znajdują się:

- albo wewnątrz jednego z czterech pod-kwadratów – takie krawędzie możemy znaleźć wywołują rekurencyjne szukanie krawędzi w każdym z pod-kwadratów,
- albo będą to krawędzie pomiędzy kwadratami LG i PG (nazwę je poziomymi krawędziami), albo (poziome) krawędzie pomiędzy LD i PD, albo (pionowe) krawędzie pomiędzy LG i LD, albo (pionowe) krawędzie pomiędzy PG i PD.

Dla uproszczenia, skupmy się tylko na szukaniu poziomych krawędzi pomiędzy dwoma kwadratami.

Weźmy sobie dwa kwadraty A i B, takie że A jest na lewo od B. Teraz szukanie krawędzi wygląda następująco:

- jeśli jeden z tych kwadratów jest typu `0`, to nie znajdziemy pomiędzy nimi żadnej krawędzi,
- jeśli **oba** są typu `1`, to możemy dodać pomiędzy nimi krawędź,
- jeśli A jest typu `1`, a B typu `4`, to musimy dodać poziome krawędzie pomiędzy A i B_{LG} oraz pomiędzy A i B_{LD},
- jeśli A jest typu `4`, a B typu `1`, przypadek jest analogiczny,
- jeśli oba są typu `4`, to musimy dodać poziome krawędzie pomiędzy A_{PG} i B_{LG} oraz pomiędzy A_{PD} i B_{LD}.

Warto zauważyć, że liczba krawędzi w tak sporządzonym grafie wynosi co najwyżej $4 \cdot N$, gdyż każdy kwadrat ma co najwyżej cztery krawędzie do **nie mniejszych** kwadratów. W ten sposób policzyliśmy każdą krawędź co najmniej raz.

Na tak sporządzonym grafie możemy już wykonywać algorytm DFS (bądź Union Find) tak jak w poprzednim podpunkcie.

Rozwiązanie to dawało połowę punktów.

Znalezienie wielkości największego obszaru

Wydawałoby się, że dodanie liczenia obszarów do poprzedniego podzadania jest dość proste, jednakże daje ono jedynie dodatkowe 15 punktów. Wynika to z tego, że obszary te mogą być bardzo duże i nie jesteśmy w stanie przetrzymać tej informacji w polu typu `long long`.

Możemy sobie jednak z tym poradzić, trzymając rozmiary obszarów w postaci binarnej, za pomocą na przykład struktury `set`. Dla przykładu liczbę $19 = 10011_2$ możemy trzymać jako zbiór $\{0, 1, 4\}$ (bo $2^0 + 2^1 + 2^4 = 19$). Operując na takiej reprezentacji jesteśmy w stanie szybko dodać do zbioru potęgę dwójki (należy pamiętać, że podczas takiego dodania kilka bitów może zostać zgaszonych) oraz potrzebujemy zaimplementować porównywanie takich zbiorów.

Jan Wańkowicz
Autor omówienia i paczki

Droga do szkoły

Formalna treść zadania: mamy dany graf nieskierowany bez multikrawędzi i pętli do samego siebie. Istnieje w nim ścieżka z 1 do 2, a najkrótsza z nich jest długości co najmniej 5. Ile najwięcej można dodać w naszym grafie krawędzi, żeby dalej między 1 a 2 nie istniała ścieżka krótsza niż 5?

Rozwiązanie w złożoności $\mathcal{O}\left(N^2 \cdot 2^{\binom{N}{2}}\right)$

Sprawdzamy wszystkie podzbiory krawędzi, które chcemy dodać, potem BFSem sprawdzamy, czy długość z 1 do 2 dalej jest nie mniejsza niż 5.

Rozwiązanie wzorcowe w złożoności $\mathcal{O}(N + M)$

Podzielmy sobie nasz graf na warstwy, gdzie w warstwie i będą wszystkie wierzchołki odległe od wierzchołka 1 o dokładnie i. W ten sposób jesteśmy również w stanie podzielić dowolny graf wynikowy (takim grafem będziemy nazywać taki z maksymalną liczbą krawędzi). Potencjalnymi krawędziami będziemy nazywać takie, które możemy dodać do naszego grafu i nie popsuje nam to warunku z zadania. Wiemy, że wierzchołek 2 musi być w warstwie $\geqslant 5$. W celu rozwiązania zadania potrzebne nam teraz kilka obserwacji.

Obserwacja: Jeśli dodamy krawędź między wierzchołkami u i v w tej samej warstwie lub między dwoma sąsiednimi warstwami, to podział na warstwy nie zmieni się.

Dowód: Odległości od wierzchołka 1 do obu tych wierzchołków różniły się maksymalnie o 1, stąd dodanie pojedynczej krawędzi z jednego do drugiego nic nie zmieni.

Obserwacja: W grafie wynikowym warstwy $\geqslant 6$ są puste.

Dowód: Zauważmy, że jeżeli ostatnia niepusta warstwa jest $\geqslant 6$, to możemy przesunąć wierzchołki z niej do warstwy o 1 bliższej. Nie zepsujemy wtedy pozostałych warstw ani odległości z 1 do 2, a jedynie możemy zwiększyć liczbę potencjalnych krawędzi.

Obserwacja: W grafie wynikowym warstwa 0 zawiera tylko wierzchołek 1, a warstwa 5 wierzchołek 2.

Dowód: Jeśli warstwa 0 zawiera jakiś wierzchołek poza 1, to możemy go przesunąć do warstwy pierwszej. Nic się nie popsuje, a uzyskujemy nowe potencjalne krawędzie z warstwą 2. Analogicznie dla warstwy 5.

Obserwacja: Istnieje graf wynikowy, w którym w warstwie 1 leżą tylko te wierzchołki, które sąsiadowały z wierzchołkiem 1 w grafie wejściowym, zaś w 4 tylko te, które sąsiadowały z 2 w grafie wejściowym.

Dowód: Jeśli mamy jakiś dodatkowy wierzchołek w warstwie 1, to wiemy, że nie sąsiadował on pierwotnie z wierzchołkiem 1. Stąd, kiedy przesuniemy go do warstwy 2, tracimy jedną potencjalną krawędź z warstwą 0, za to zyskujemy przynajmniej jedną potencjalną krawędź z warstwą 3. Stąd zawsze nam się to opłaca. Analogicznie dla warstwy 4.

Wiemy, że w warstwie 2 muszą być wierzchołki odległe o 2 od wierzchołka 1, zaś w warstwie 3 wierzchołki odległe o 2 od wierzchołka 2. Wszystkie pozostałe, nieustalone jeszcze wierzchołki (te niepołączone z 1 oraz te odległe o więcej niż 2 od wierzchołka 1 i 2) oznaczmy jako zbiór U.

Obserwacja: Biorąc graf wynikowy po modyfikacji z poprzedniej obserwacji, jeśli w warstwie 1 jest więcej wierzchołków niż w 4, to wszystkie wierzchołki z U opłaca się włożyć do warstwy 2, w przeciwnym przypadku opłaca się je włożyć do warstwy 3.

Dowód: Wszystkie wierzchołki z U chcemy na pewno dać do warstwy 2 lub 3. Stąd zawsze będziemy mieli potencjalne krawędzie w obrębie naszego zbioru oraz z pozostałymi wierzchołkami z warstw 2 i 3. Stąd chcemy je dać bliżej tej warstwy, gdzie jest więcej wierzchołków, bo maksymalizuje nam to liczbę potencjalnych krawędzi.

Z powyższych obserwacji mamy podział na warstwy w pewnym wynikowym grafie. Stąd już prosto możemy policzyć ile maksymalnie krawędzi możemy dodać korzystając z naszej pierwszej obserwacji.

Jan Wańkowicz
Autor omówienia i paczki

Łańcuszek

Formalna treść zadania: mamy dane słowo zero-jedynkowe oraz zmiany pojedynczej literki na przeciwną. Po każdej takiej zmianie musimy odpowiedzieć na pytanie „jaka jest długość najkrótszego słowa zero-jedynkowego, które nie występuje w naszym słowie wejściowym jako podsłowo".

Obserwacja: Odpowiedź na nasze zapytania będzie zawsze równa maksymalnie 17 (tj. $\log N$).

Dowód: Zauważmy, że wszystkich możliwych słów długości 17 składających się z zer i jedynek jest dokładnie $2^{17} = 131072$, zaś podsłów długości 17 w naszym napisie jest maksymalnie $N - 16$. Stąd któreś z podsłów nie występuje w naszym słowie.

Rozwiązanie w złożoności $\mathcal{O}(N^2 \cdot M)$ lub $\mathcal{O}(N^2 \log N \cdot M)$

Powyższa obserwacja jest potrzebna tylko do dowodu złożoności poniższego algorytmu, ale nie trzeba było jej mieć, żeby go napisać :).

Będziemy sprawdzać po kolei każde możliwe podsłowo zaczynając od najkrótszych. Maksymalnie nie znajdziemy $\mathcal{O}(N)$-tego słowa. Jeśli chcielibyśmy brutalnie sprawdzać, czy dany string długości $\mathcal{O}(\log N)$ występuje w słowie długości $\mathcal{O}(N)$ potrzebujemy normalnie czasu $\mathcal{O}(N \log N)$. Aby to przyspieszyć, możemy użyć KMP (mocno tego odradzam) lub po prostu traktować nasze podsłowo jako liczbę.

Rozwiązanie wzorcowe w złożoności $\mathcal{O}\left(M \log^2 N + N \log N\right)$

W naszym słowie interesują nas tylko podsłowa długości co najwyżej $\mathcal{O}(\log N)$. Stąd, po każdej zmianie zmieni się co nawyżej $\mathcal{O}\left(\log^2 N\right)$ interesujących nas podsłów.

Korzystając z powyższego faktu, możemy trzymać sobie dla każdego $i \leqslant 17$ listę potencjalnych podsłów długości i, które nie występują w naszym słowie. Na początku wrzucamy do nich wszystkie słowa, stąd listy będą miały sumaryczną długość $\mathcal{O}(N \log N)$. Dla każdego słowa długości co najwyżej $\mathcal{O}(\log N)$ będziemy trzymać jego liczbę wystąpień w naszym słowie i w ten sposób sprawdzać, czy potencjalni kandydaci są dobrzy. Po każdym zapytaniu dochodzi nam potencjalnie $\mathcal{O}\left(\log^2 N\right)$ nowych kandydatów, stąd sumaryczna złożoność algorytmu wynosi $\mathcal{O}\left(M \cdot \log^2 N + N \log N\right)$.

Jan Wańkowicz
Autor omówienia i paczki

Dinozaury

Formalna treść zadania: mamy dane dwa ukorzenione w wierzchołku 1 drzewa oraz połączenia o pewnej wadze pomiędzy dwoma wierzchołkami z różnych drzew. Dla każdego zapytania o wierzchołek z pierwszego drzewa i wierzchołek z drugiego drzewa mamy znaleźć sumę wag takich połączeń, wśród których oba łączone wierzchołki są przodkami (uznajemy, że wierzchołek jest też przodkiem dla samego siebie) tych z zapytania.

Rozwiązanie w złożoności $\mathcal{O}(N + M + QK)$

Dla każdego zapytania iterujemy się po wszystkich połączeniach i sprawdzamy, czy chcemy je wziąć. Potrzebujemy do tego sprawdzania, czy dany wierzchołek a jest przodkiem wierzchołka b. Można to zrobić na 2 sposoby:

- brutalnie policzyć i trzymać dla każdego wierzchołka tablicę mówiącą kto jest jego przodkiem. Wtedy dokładamy do złożoności $\mathcal{O}(N^2 + M^2)$, ale dalej mieścimy się w limitach do pierwszego podzadania,
- policzyć dla każdego wierzchołka kolejność pre-order i post-order, czyli inaczej czas wejścia do danego wierzchołka przez DFSa oraz czas jego wyjścia. Wtedy żeby sprawdzić, czy wierzchołek a jest przodkiem wierzchołka b wystarczy sprawdzić, czy do a weszliśmy wcześniej niż do b (czyli czy pre-order a jest mniejszy lub równy pre-orderowi b) oraz czy później wyszliśmy z a niż z b (czyli czy jego post-order jest większy lub równy).

Rozwiązanie podzadania, w którym pierwsze drzewo jest ścieżką

Najpierw warto było zauważyć, że warunek z podzadania na różność A_i oznaczał, że każdy wierzchołek ma maksymalnie jednego syna, czyli w innych słowach nasze drzewo jest ścieżką.

Będziemy rozwiązywać nasz problem offline, czyli spamiętamy sobie nasze zapytania i wypiszemy odpowiedzi na nie po obliczeniu odpowiedzi na wszystkie z nich. W tym celu do każdego wierzchołka z drugiego drzewa wrzućmy listę połączeń z wierzchołkami z pierwszego drzewa oraz listę zapytań zawierających ten wierzchołek.

Przejdźmy teraz DFSem drugie drzewo. Przy wejściu do wierzchołka chcielibyśmy jakoś zaznaczać połączenia wychodzące z niego jako aktywne, zaś po wyjściu z niego z powrotem jako nieaktywne. Wtedy gdy jesteśmy w jakimś wierzchołku w drugim drzewie, jedyne aktywne połączenia to takie, które mają jeden z końców w jego przodku. Czyli gdy chcemy odpowiedzieć na zapytanie o niego i pewien wierzchołek c w pierwszym drzewie, musimy odpowiedzieć tak naprawdę na pytanie „jaka jest suma wag po takich aktywnych połączeniach, które mają jeden ze swoich końców w przodku c". Możemy teraz skorzystać w końcu z faktu, że nasze pierwsze drzewo jest ścieżką. Gdy chcemy aktywować połączenie, które ma jeden koniec w wierzchołku c z pierwszego drzewa, wystarczy, że dodamy jego wagę dla wszystkich wierzchołków w poddrzewie c, co można zrobić za pomocą prostego drzewa przedziałowego, dodając wartość na pewnym sufiksie wierzchołków. Dezaktywację robimy analogicznie – odejmujemy wartość na sufiksie. Dzięki temu możemy odpowiadać na nasze zapytania pytając się po prostu o wartość w punkcie.

Otrzymujemy złożoność $\mathcal{O}(\log N)$ na zapytanie oraz połączenie, czyli sumarycznie $\mathcal{O}(N + M + (Q + K)\log$

Rozwiązanie wzorcowe

Zauważmy, że chcemy odpowiadać na takie same zapytania dla wierzchołków z pierwszego drzewa jak w poprzednim podzadaniu. Jedyne, czego teraz nie możemy tak prosto zrobić, to dodać wartość dla wszystkich wierzchołków w poddrzewie konkretnego wierzchołka. Zauważmy jednak, że podczas działania DFSa w poddrzewie konkretnego wierzchołka c znajdujemy się w spójnym przedziale czasu. Konkretnie, znajdujemy się tam od wejścia do wierzchołka c aż do wyjścia z niego. Stąd ponownie możemy działać na drzewie przedziałowym, dodając wartość na takim przedziale (wcześniej musimy policzyć momenty wejścia i wyjścia dla każdego wierzchołka). Złożoność otrzymujemy taką samą jak w poprzednim podzadaniu.

Uwaga: Rozwiązanie można było przeprowadzić techniką Heavy-Light Decomposition – wystarczy przechodzić po jednym drzewie DFSem, dodawać i odejmować połączenia punktowo do wierzchołków drugiego drzewa (bez aktualizowania poddrzewa) i przy zapytaniu sprawdzać sumę na ścieżce do korzenia z wierzchołka z drugiego drzewa konkretnego zapytania.

Michał Kępa
Autor omówienia i paczki

Trójsegmentowe tunele

Bez straty ogólności, w celu ułatwienia omówienia, założymy, że interesują nas tylko ścieżki ze wschodu na zachód. Zadanie jest więc następujące: mamy graf N-wierzchołkowy, początkowo bez żadnych krawędzi. Możemy dodawać do niego krawędzie skierowane między dowolną parą wierzchołków (u, v) gdy $u < v$. Chcemy dołożyć jak najmniej krawędzi, tak żeby odległość między dowolną parą wierzchołków (s, k) wynosiła co najwyżej 3.

Rozwiązanie z limitem liczby tuneli $10 \cdot N$

Graf będziemy budować rekurencyjnie. Wybieramy wierzchołek $v = \lfloor \frac{N}{2} \rfloor$. Dla każdego wierzchołka $u < N$ dodajemy krawędź (u, v), a dla każdego $u > N$ krawędź (v, u). Dzięki temu mamy zapewnione, że wszystkie ścieżki zaczynające się w lewej części i kończące się w prawej są poprawne. Teraz musimy rekurencyjnie wywołać procedurę budowy grafu, żeby stworzyć poprawne ścieżki, które zaczynają i kończą się w tej samej połowie grafu.

Takie rozwiązanie tworzy N krawędzi na poziom wywołania rekurencyjnego i ma $\log(N)$ wywołań, a skoro $N \leqslant 500$ to $N \cdot \log(N) \leqslant 10 \cdot N$.

Rozwiązanie alternatywne

Wyróżnijmy co B-ty wierzchołek. Powstanie $B + 1$ grup wierzchołków o numerach ograniczonych przez wyróżnione wierzchołki. Wewnątrz grupy połączmy każdą parę wierzchołków krawędzią. Dodajmy też krawędź między każdą parą wyróżnionych wierzchołków.

Jaka jest długość ścieżki między dowolnymi wierzchołkami $u < v$? Jeżeli są w jednej grupie to ta odległość to 1. Jeżeli są w różnych grupach to odległość to 3: jedną krawędzią przechodzimy do wierzchołka wyróżniającego będącego końcem grupy, w której jest u, drugą przejdziemy do wyróżnionego wierzchołka, który jest początkiem grupy z u, a trzecią dojdziemy do u.

Na cały graf potrzebujemy (szacunkowo): $\frac{1}{2} \cdot B^2$ krawędzi na połączenie wyróżnionych wierzchołków, $\frac{1}{2} \cdot \left(\frac{N}{B}\right)^2$ krawędzi w każdej z B grup. Sumarycznie dodajemy więc $\frac{1}{2} \cdot B^2 + \frac{1}{2} \cdot \frac{N^2}{B}$ krawędzi. Optymalnie jest wybrać $B \approx N^{\frac{2}{3}}$. Wtedy zużyjemy $N^{\frac{4}{3}}$, co dla $N \leqslant 500$ (wtedy $N^{\frac{1}{3}} < 8$) rozwiązuje pierwsze podzadanie.

Rozwiązanie wzorcowe z limitem liczby tuneli $5 \cdot N$

Rozwiązanie jest połączeniem dwóch powyższych rozwiązań. Będziemy rekurencyjnie budować graf, ale zamiast dzielić go na dwie części, to będziemy go dzielić na B części. Tak jak poprzednio, wierzchołki rozdzielające grupy wyróżniamy i łączmy je każdy z każdym. Dodatkowo łączymy każdy wierzchołek z początkiem i końcem grupy. To zapewnia, że wierzchołki z różnych grup łączy ścieżka odpowiedniej długości. Następnie wywołujemy się rekurencyjnie na każdej grupie, żeby zapewnić połączenie wierzchołków wewnątrz jednej grupy.

Ile stworzymy krawędzi? Na jednym poziomie rekurencji zużywamy $\frac{1}{2} \cdot B^2$ krawędzi na połączenie par wierzchołków wyróżnionych oraz $2 \cdot N$ krawędzi na połączenie wierzchołka z końcem i początkiem grupy.

Jeżeli wybierzemy $B = \sqrt{N}$, to jeden poziom wykorzysta $2.5 \cdot N$ krawędzi. Ile będzie poziomów? Skoro za każdym razem zmniejszamy rozmiar do pierwiastka, to poziomów będzie $\log(\log(N)) \approx 3$ dla $N = 500$. Zauważmy jednak, że na pierwszym poziomie rozmiar grupy

to maksymalnie $\sqrt{500} = 22$, więc wywołamy się rekurencyjnie z $N = 20$ (końce nie należą do grupy). Na kolejnym poziomie będą zatem grupy rozmiaru 5, czyli na trzecim poziomie jest $N = 3$ i musimy dodać 2 krawędzie. Takie oszacowanie z grubsza daje $6 \cdot N$ krawędzi.

Łatwo możemy sprawdzić programem (po prostu uruchamiając go na każdym możliwym z 500 testów), że w praktyce nie przekroczymy nigdy limitu $5 \cdot N$ użytych krawędzi.

Michał Kępa
Autor omówienia i paczki

Wrocławskie krasnale

Rozwiązanie $\mathcal{O}(N^3)$

Kiedy nie możemy przyjąć nowego ochotnika? Wtedy, gdy istnieje dwóch innych kandydatów, którzy razem tworzą trójkę $\{A, B, C\}$ taką, że A może współpracować z B, B może z C, ale A nie może współpracować z C. Dla zapytania iterujemy się więc po każdej parze już przyjętych kandydatów i sprawdzamy czy tworzą taką parę.

Rozwiązanie $\mathcal{O}(N^2)$

Obserwacja: Kandydat nie może zostać przyjęty tylko wtedy, gdy może współpracować z dwoma kandydatami z różnych oddziałów lub gdy może współpracować tylko z częścią oddziału.

Wystarczy więc trzymać skład każdego z oddziałów (np. przy użyciu `vector`) i sprawdzić dla nowego kandydata czy:

- w jakimś oddziale jest osoba z którą może współpracować i druga, z która nie może współpracować,
- są co najmniej dwa oddziały, w których istnieje osoba, z którą kandydat może współpracować.

Jeżeli żaden z warunków nie zachodzi, to możemy dodać kandydata do odpowiedniego oddziału (już istniejącego, jeżeli z kimś może on współpracować lub do całkiem nowego oddziału).

Podzadanie $A_i + 2 = B_i$

W tym przypadku kandydat o opisie (A_i, B_i) może współpracować jedynie z kandydatem o opisie ($A_i - 1$, $B_i - 1$) lub ($A_i + 1$, $B_i + 1$). Nowego kandydata nie możemy przyjąć wtedy, gdy razem z przyjętymi już osobami utworzą ciąg trzech kolejnych osób (bo pierwsza osoba z takiej grupy nie może współpracować z trzecią). Wystarczy, więc trzymać przyjętych kandydatów w strukturze `set` lub `unordered_set` i rozpatrywać kolejne zapytania.

Przyjęci kandydaci tworzą jeden oddział

Jak efektywnie reprezentować oddział?

Obserwacja: Wystarczy pamiętać jedynie minimalne i maksymalne wartości A_i oraz B_i kandydatów należących do tego oddziału.

Dowód:

1. Dla osób z tego samego zachodzi $A_i < B_j$, więc w szczególności $\max_A < \min_B$.
2. Kandydat i może należeć do oddziału, gdy może współpracować z każdym z jego członków. Czyli dla każdego członka j zachodzi $A_i < B_j$ oraz $A_j < B_i$. A zachodzi to wtedy i tylko wtedy, gdy $A_i < \min_B$ i $\max_A < B_i$.

3. Kandydat i nie może współpracować z każdym członkiem oddziału, gdy dla każdego członka j zachodzi $A_i \geqslant B_j$ lub $A_j \geqslant B_i$. Zauważmy jednak, że skoro $\max_A < \min_B$, to albo dla wszystkich członków zachodzi pierwszy warunek, albo drugi. Zatem $A_i \geqslant \max_B$ albo $\min_A \geqslant B_i$.

4. Kandydat i nie może zostać przyjęty, gdy istnieje kandydat j, z którym nie może współpracować oraz k, z którym może. Skoro nie może współpracować z j to $A_i \geqslant B_j$ lub $A_j \geqslant B_i$. Rozważmy pierwszy przypadek, drugi będzie analogiczny. Wiemy zatem, że zachodzi też $A_i \geqslant \min_B$. A skoro może współpracować z k, to $A_i < B_k$, więc $A_i < \max_B$. Zatem kandydata i nie można przyjąć gdy $\min_B \leqslant A_i < \max_B$ lub $\min_A < B_i \leqslant \max_A$.

Wystarczy zatem utrzymać aktualny rozmiar oddziału i aktualizować go podczas dodawania kolejnych kandydatów. Złożoność: $\mathcal{O}(N)$.

Rozwiązanie wzorcowe $\mathcal{O}(N \log(N))$

Oddziały będziemy reprezentować jak w rozwiązaniu powyżej.

Obserwacja: Przedziały od min_A do max_B reprezentujące różne oddziały nie przecinają się.

Dowód:

1. Gdy tworzymy nowy oddział, to nie przecina się on z żadnym z istniejących. Gdyby się przecinał, to albo kandydat tworzący nowy oddział mógłby dołączyć do istniejącego, albo nie mógł zostać przyjęty.

2. Gdyby przedziały miały się przeciąć po dołączeniu nowego kandydata, to oznacza, że ten kandydat może być współpracować z osobami z obu tych oddziałów. Ale one nie mogą współpracować ze sobą. Zatem nie powinien zostać przyjęty.

Gdy rozważamy kandydata (A_i, B_i) wystarczy znaleźć oddział, do którego mógłby należeć ten kandydat. Może to być pierwszy oddział o $\min_A > A_i$ albo poprzedni oddział (taki oddział, który ma największe min_A, ale mniejsze od A_i). Musimy sprawdzić, czy kandydat nie popsuł by żadnego z tych dwóch oddziałów, ani czy po jego dodaniu te oddziały nie przecięły się (co oznacza, że mamy osoby z dwóch oddziałów, z którymi kandydat i może współpracować). Szukanie oddziału może wykonać przy użyciu struktury `set`, która będzie trzymała oddziały posortowane względem $\min_A$.

Marcel Szelwiga
Autor omówienia i paczki

Kryzys termiczny

W tym zadaniu mieliśmy napisać strukturę, która umożliwi efektywne wykonywanie następujących operacji:

- ustaw określoną wartość na przedziale,
- sprawdź czy istnieje podzbiór elementów z określonego przedziału, którego xor jest równy zadanej liczbie,
- podaj maksymalnego xora podzbioru elementów z określonego przedziału.

Rozwiązanie dla podzadania $N, Q \leqslant 10$

W tym podzadaniu limity pozwalały nam sprawdzenie dla każdego zapytania każdego możliwego podzbioru. Rozwiązanie to działa w złożoności $\mathcal{O}\left(2^N \cdot Q\right)$.

Rozwiązanie dla $Q = 1$

W zadaniu nie mieliśmy takiego podzadania, przy czym w moim odczuciu to właśnie od niego należałoby zacząć rozwiązanie tego zadania. Załóżmy, że mamy ciąg liczb A_i i jedno zapytanie – czy jesteśmy w stanie wybrać podzbiór ciągu A_i, którego xor jest równy Y.

Wyznaczmy największą liczbę z tego ciągu, bez straty ogólności załóżmy, że jest to element A_1. Zauważmy, że żadna z liczb nie ma większego indeksu bitu wiodącego niż ma A_1. Zamieńmy teraz ciąg A na A' gdzie $A'_1 = A_1$, a $A'_i = A_i \oplus A_1$ gdy A_i posiada ten sam bit wiodący do A_1 lub $A'_i = A_i$ w przeciwnym przypadku, dla $i \geqslant 2$. Zauważmy, że zbiór xorów podzbiorów ciągu A jest równy zbiorowi xorów podzbiorów ciągu A'. Ponadto każda z liczb A'_i dla $i \geqslant 2$ ma mniejszy indeks wiodącego bitu niż ma liczba A'_1.

Zauważmy, że po każdym kroku zostanie nam jedna liczba o wybranym indeksie bitu wiodącego, a pozostałe liczby będą miały ten bit zgaszony. Powtórzmy powyższy proces $\lceil \log_2(\max A_i) \rceil$ razy i zauważmy, że wśród pozostałych liczb co najwyżej $\lceil \log_2(\max A_i) \rceil$ pozostanie niezerowych. Jesteśmy zatem w stanie skompresować ciąg A_i długości N do ciągu A'_i o rozmiarze logarytmicznym względem wielkości elementów ciągu A, którego zbiór xorów podzbiorów będzie równy zbiorowi xorów podzbiorów oryginalnego ciągu.

Rozważmy w jaki sposób efektywnie zaimplementować powyższy proces. Utwórzmy tablicę T, niech początkowo $T[i] = 0$ dla każdego i. Będziemy chcieli, aby $T[i]$ zawierała liczbę o wiodącym bicie na pozycji i-tej. Rozważmy dodawanie liczby x do tablicy $T[i]$ w następujący sposób:

```
while (x != 0 and T[lb(x)])
    x ^= T[lb(x)]
if (x != 0)
    T[lb(x)] = x
```

gdzie `^` symbolizuje operację xorowania, a `lb(x)` zwraca indeks wiodącego bitu x. Zauważmy, że po dodaniu w powyższy sposób do tablicy T wszystkich elementów ciągu A_i otrzymamy analogiczny efekt do przeprowadzonego wcześniej procesu – zbiór xorów podzbiorów tablicy T będzie równy zbiorowi xorów podzbiorów ciągu A.

Nazwijmy powyższą strukturę $\mathcal{G}$ i zauważmy, że do struktury $\mathcal{G}$ możemy w łatwy sposób dodawać liczby oraz w prosty sposób jesteśmy w stanie łączyć dwie struktury $\mathcal{G}$. Możemy również zauważyć, że posiadając strukturę $\mathcal{G}$ jesteśmy już w stanie w łatwy sposób zweryfikować czy możemy utworzyć liczbę Y jako podzbiór elementów ciągu A_i iterując się po tablicy T od najbardziej znaczącego bitu i sprawdzając czy możemy go odpowiednio ustawić.

Rozwiązanie dla podzadania $N, Q \leqslant 1\,000$

Dla każdego zapytania tworzymy strukturę $\mathcal{G}$ i dodajemy do niej naiwnie liczby z zadanego przedziału. W przypadku operacji ustawienia liczby na przedziale możemy po prostu zmienić element ciągu A. Otrzymujemy w ten sposób rozwiązanie o złożoności $\mathcal{O}\left(Q \cdot N \cdot \log^2(\max A_i)\right)$.

Rozwiązanie wzorcowe

Rozwiązaniem wzorcowym jest utworzenie standardowego drzewa przedziałowego utrzymującego strukturę $\mathcal{G}$ w każdym węźle. Otrzymujemy w ten sposób złożoność

$$\mathcal{O}\left((Q+N) \cdot \log(N) \cdot \log^2(\max A_i)\right),$$

co wystarczało do rozwiązania zadania.

Bartosz Chomiński
Autor omówienia i paczki

Naleśniki

Rozwiązanie dla podzadania $N_i \leqslant 1\,000$

Oznaczmy przez $\mathrm{dp}[N][K]$ liczbę sposobów, na które można wybrać K łyżeczek dodatków smakowych z puli N dodatków.

Zauważmy, że zachodzą pewne oczywiste zależności:

$$\mathrm{dp}[N][0] = 1, \quad \mathrm{dp}[N][2N] = 1$$

oraz

$$\mathrm{dp}[N][K] = \mathrm{dp}[N-1][K] + \mathrm{dp}[N-1][K-1] + \mathrm{dp}[N-1][K-2].$$

Druga zależność wynika stąd, że wszystkie sposoby wybrania K łyżeczek dodatków smakowych z puli N dodatków możemy podzielić na trzy kategorie – takie, w których N-ty dodatek występuje w ilości 0 łyżeczek, takie, w których N-ty dodatek występuje w ilości 1 łyżeczki i takie, w których N-ty dodatek występuje w ilości 2 łyżeczek.

W tym podzadaniu wystarczyło zaimplementować obliczanie tablicy dp mod 3 w czasie $\mathcal{O}\left(N^2\right)$.

Rozwiązanie dla podzadania $N_i = 2^{\varepsilon}3^{m}$

Korzystając z rozwiązania poprzedniego podzadania możemy wyznaczyć wyniki dla początkowych wartości N i zauważyć pewną strukturę.

						1						
					1	1	1					
				1	2	3	2	1				
			1	3	6	7	6	3	1			
		1	4	10	16	19	16	10	4	1		
	1	5	15	30	45	51	45	30	15	5	1	
1	6	21	50	90	126	141	126	90	50	21	6	1

Rysunek 1: Tablica dp (w dół wzrasta N, środek wiersza to $\mathrm{dp}[N][N]$)

						1						
					1	1	1					
				1	2	0	2	1				
			1	0	0	1	0	0	1			
		1	1	1	1	1	1	1	1	1		
	1	2	0	0	0	0	0	0	0	2	1	
1	0	0	2	0	0	0	0	0	2	0	0	1

Rysunek 2: Tablica dp mod 3 (w dół wzrasta N, środek wiersza to $\mathrm{dp}[N][N] \bmod 3$)

Wyniki uzyskane w ten sposób skłaniają nas do postawienia pewnych hipotez:

$$\mathrm{dp}[3^m][x] = \begin{cases} 1, & \text{dla } x \in \{0, 3^m, 2 \cdot 3^m\}, \\ 0, & \text{dla pozostałych } x \end{cases}$$

oraz

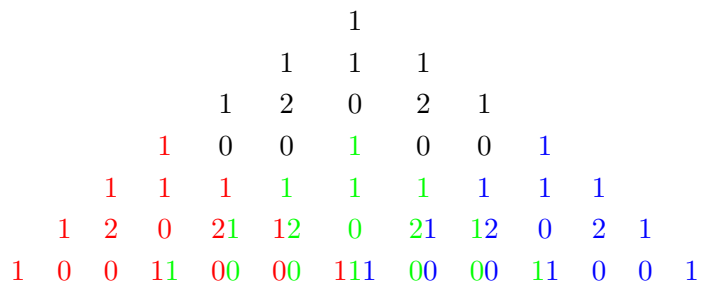
$$\mathrm{dp}[2 \cdot 3^m][x] = \begin{cases} 1, & \text{dla } x \in \{0, 4 \cdot 3^m\}, \\ 2, & \text{dla } x \in \{3^m, 3 \cdot 3^m\}, \\ 0, & \text{dla pozostałych } x. \end{cases}$$

Dowód tych hipotez łącznie można przeprowadzić prostą indukcją matematyczną, co pozostawiamy jako ćwiczenie dla Dociekliwego Czytelnika.

W omawianym podzadaniu wystarczyło więc zaimplementować instrukcje warunkowe zgodne z postawionymi i udowodnionymi wcześniej hipotezami.

Rozwiązanie dla podzadania $N_i \leqslant 1\,000\,000$, $Q = 1$

Na podstawie rozwiązania poprzedniego podzadania spróbujmy znaleźć głębszą zależność rekurencyjną na dp mod 3. Skoro na poziomach o indeksach będących potęgami trójki występują zawsze dokładnie trzy wartości 1 w tablicy dp mod 3, a reszta to zera, to możemy pomyśleć o tych trzech jedynkach jako o nowych „źródłach" całego procesu. Ściślej mówiąc, możemy sobie wyobrazić, że zaczynamy liczyć tablicę dp mod 3 od nowa z punktów $(3^m, 0), (3^m, 3^m), (3^m, 2 \cdot 3^m)$.



Rysunek 3: Tablica dp mod 3 (w dół wzrasta N, środek wiersza to $\mathrm{dp}[N][N]$ mod 3). Wartość komórki to suma liczb, które się w niej znajdują (znaki + zostały pominięte dla czytelności).

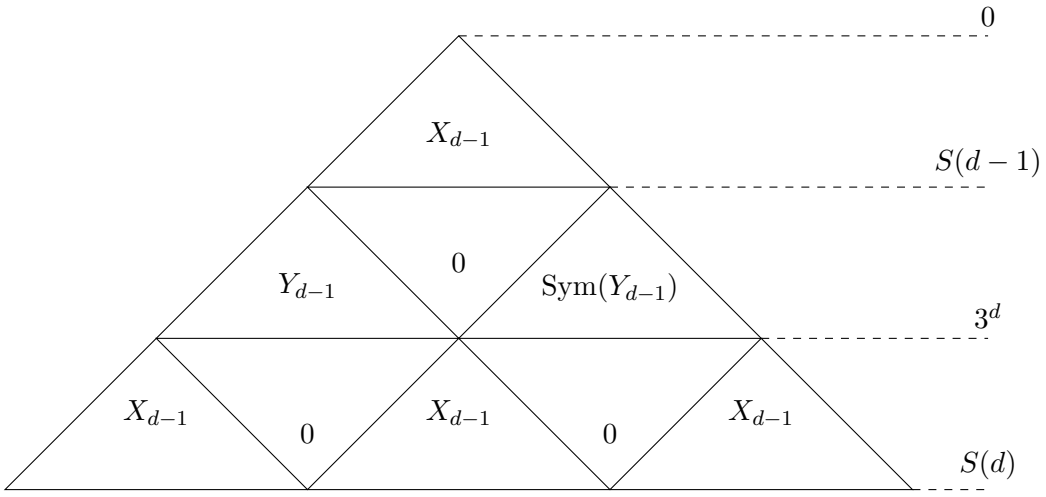
Wszystkie trzy kolorowe trójkąty wyglądają tak samo jak pierwsze cztery linie całego trójkąta, bo zostały wygenerowane w ten sam sposób, ale z innych punktów startowych.

Wobec tego każdą komórkę tablicy dp mod 3 możemy zapisać jako sumę co najwyżej trzech innych komórek tej tablicy, ale o pierwszym indeksie (N) zmniejszonym o co najmniej jedną trzecią.

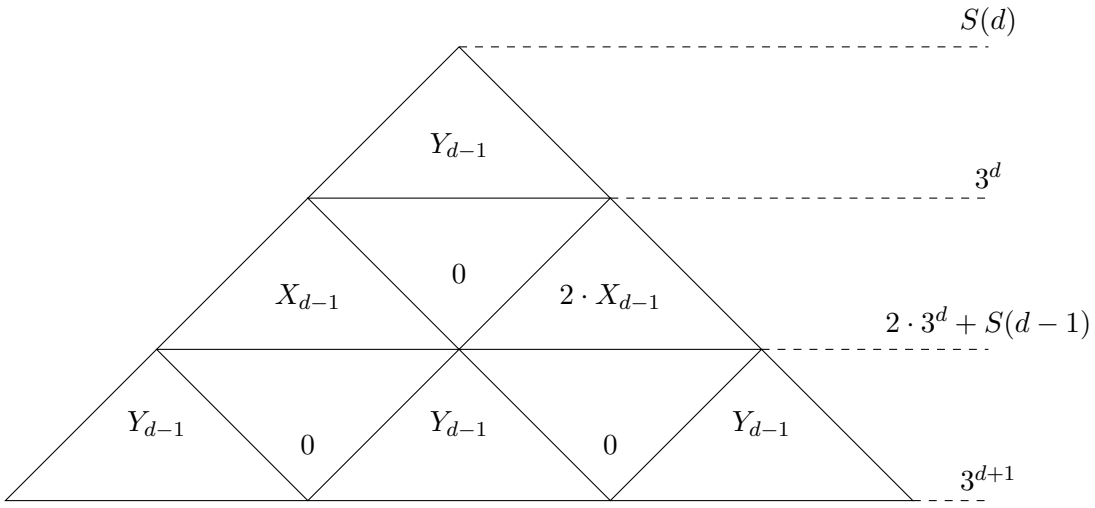
Oszacujmy złożoność obliczeniową tego podejścia. Niech $T(N)$ będzie liczbą operacji potrzebnych do wykonania, by wyznaczyć wartość $\mathrm{dp}[N][K]$. Mamy więc

$$T(N) \leqslant 3 \cdot T(2N/3).$$

Zauważmy, że $T(N) = N \cdot \log_{\sqrt[3]{3/2}} N$ spełnia zadaną nierówność, a więc $T(N) \in \mathcal{O}(N \log N)$.



Rysunek 4: Trójkąt X_d



Rysunek 5: Trójkąt Y_d

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe działa podobnie do rozwiązania poprzedniego podzadania, ale wykorzystuje jeszcze głębsze zależności rekurencyjne, które pozwalają na uzyskanie złożoności obliczeniowej $\mathcal{O}(\log N)$ dla jednego zapytania (w odróżnieniu od poprzedniej $\mathcal{O}(N \log N)$).

Proponowany podział na trójkąty dwóch rodzajów znajduje się rysunkach. Sym oznacza odbicie względem symetralnej poziomego boku trójkąta, a $2\cdot$ oznacza pomnożenie wszystkich elementów trójkąta przez 2 i wzięcie reszty z dzielenia przez 3.

Dociekliwy Czytelnik z pewnością będzie w stanie udowodnić, że przedstawione graficznie zależności rekurencyjne są poprawne.

Rozwiązanie alternatywne

Rozważmy podstawową zależność rekurencyjną na tablicę dp dla N i K podzielnych przez 3. Mamy

$$\mathrm{dp}[3N][3K] = \mathrm{dp}[3N-1][3K] + \mathrm{dp}[3N-1][3K-1] + \mathrm{dp}[3N-1][3K-2].$$

					1							
				1	1	1						
			1	2	0	2	1					
		1	0	0	1	0	0	1				
	1	1	1	1	1	1	1	1	1			
1	2	0	0	0	0	0	0	0	2	1		
1	0	0	2	0	0	0	0	0	2	0	0	1

Rysunek 6: Tablica dp mod 3 (w dół wzrasta N, środek wiersza to $\mathrm{dp}[N][N]$ mod 3). Na czerwono oznaczone komórki, których oba indeksy są podzielne przez 3.

Rozwińmy wszystkie trzy składniki powyższej sumy. Mamy wówczas

$$\begin{aligned}\mathrm{dp}[3N][3K] = \mathrm{dp}[3N-2][3K] + 2\mathrm{dp}[3N-2][3K-1] + 3\mathrm{dp}[3N-2][3K-2] + \\ + 2\mathrm{dp}[3N-2][3K-3] + \mathrm{dp}[3N-2][3K-4].\end{aligned}$$

Ostatni raz rozwińmy już posiadane składniki.

$$\begin{aligned}\mathrm{dp}[3N][3K] = \mathrm{dp}[3N-3][3K] + 3\mathrm{dp}[3N-3][3K-1] + 6\mathrm{dp}[3N-3][3K-2] + \\ + 7\mathrm{dp}[3N-3][3K-3] + 6\mathrm{dp}[3N-3][3K-4] + 3\mathrm{dp}[3N-3][3K-5] + \mathrm{dp}[3N-3][3K-6].\end{aligned}$$

Jako że interesuje nas reszta z dzielenia dp przez 3, możemy odrzucić w powyższej równości składniki podzielne przez 3 i otrzymujemy

$$\mathrm{dp}[3N][3K] \equiv_3 \mathrm{dp}[3N-3][3K] + \mathrm{dp}[3N-3][3K-3] + \mathrm{dp}[3N-3][3K-6],$$

a więc analogiczną zależność rekurencyjną jak oryginalna. Wobec tego

$$\mathrm{dp}[3N][3K] = \mathrm{dp}[N][K].$$

Można to zobrazować zaznaczając w tablicy komórki o obu indeksach podzielnych przez 3 i zauważając, że czerwone liczby tworzą taki sam trójkąt jak cały widoczny na obrazku, jednak rozstrzelony trzykrotnie.

Bartosz Chomiński
Autor omówienia i paczki

Prostokątna pizza

Niech A będzie wielkością alfabetu, który będziemy oznaczać jako Σ (tu $A = 26$ i $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$).

Rozwiązanie dla podzadania $N \leqslant 100$

Wystarczy sprawdzić każdy z $\mathcal{O}(N^2)$ przedziałów w czasie liniowym względem jego długości. Złożoność wyniesie $\mathcal{O}(N^3 + N^2 \cdot A)$.

Można zmniejszyć tę złożoność zliczając wystąpienia liter na prefiksach słowa – w ten sposób unikamy liniowego przejścia po każdym przedziale i uzyskujemy złożoność $\mathcal{O}(N^2 \cdot A)$.

Rozwiązanie dla podzadania $N \leqslant 100\,000$

Rozważmy wszystkie A^2 par liter. Skoro każdy przedział ma literę najczęściej w nim występującą i najrzadziej w nim występującą, to możemy zamiast rozważać przedziały, rozważać te właśnie pary.

Ustalmy parę liter (α, β). Znajdźmy najwyższy współczynnik klasteryzacji wśród przedziałów, w których α występuje najczęściej, a β najrzadziej (ale jednak).

Zauważmy, że gdyby nie było wymogu pojawienia się litery β co najmniej raz w wybranym przedziale, to cały problem moglibyśmy łatwo sprowadzić do problemu PSOMS. Przekształcamy słowo na tablicę liczb w taki sposób, że $\alpha \mapsto 1$, $\beta \mapsto -1$, a pozostałe litery zamieniamy na zera.

Brakuje na wyłącznie uwzględnienia warunku na istnienie β w przedziale, ale możemy łatwo zmodyfikować rozwiązanie problemu PSOMS, by taki warunek wymusić – wystarczy lekko opóźnić aktualizowanie dotychczasowej minimalnej sumy prefiksowej tak, by co najmniej jedna -1 zawsze była między najnowszą minimalną sumą prefiksową a aktualnym indeksem w tablicy.

Złożoność obliczeniowa wyniesie $\mathcal{O}(N \cdot A^2)$, ponieważ problem PSOMS można rozwiązać w złożoności $\mathcal{O}(N)$.

Rozwiązanie wzorcowe

Zastanówmy się, jak poprawić poprzednie rozwiązanie. Zauważmy, że gdy w ciągu powstałym z tablicy rozpatrujemy elementy zerowe, to nie dzieje się nic ciekawego. Dlaczego więc nie pominąć wszystkich zer?

Okazuje się, że jest to dobry pomysł. Trzeba oczywiście pamiętać, żeby wycinając zera, nie przeglądać całego ciągu, gdyż wtedy złożoność pozostanie bez zmian. Należy dla każdej litery stworzyć listę zawierającą pozycje jej wystąpień w słowie, a następnie dla danej pary liter scalać listy odpowiadające tym literom.

Oszacujmy czas działania tak poprawionego algorytmu. Generowanie list wystąpień zajmie nam czas $\mathcal{O}(N + A)$. Czas działania głównej fazy to suma czasów dla wszystkich par liter z alfabetu. Niech $\#\alpha$ oznacza liczbę wystąpień litery α w początkowym słowie. Wtedy czas dla

pojedynczej pary liter (α, β) wynosi $\mathcal{O}(\#\alpha + \#\beta)$. Ostatecznie sumaryczna złożoność obliczeniowa to

$$\mathcal{O}\left(\sum_{(\alpha,\beta)\in\Sigma^2} \#\alpha + \#\beta\right) = \mathcal{O}\left(2\sum_{(\alpha,\beta)\in\Sigma^2} \#\beta\right) = \mathcal{O}\left(A\sum_{\beta\in\Sigma^2} \#\beta\right) = \mathcal{O}(N \cdot A).$$

Marcel Szelwiga
Autor omówienia i paczki

Najlepsze Brownie

W tym zadaniu mieliśmy obliczyć w jakim czasie będziemy w stanie pokolorować całe drzewo startując z dwóch punktów startowych i mogąc w jednej jednostce czasu dla każdego już pokolorowanego wierzchołka pokolorować wybranego sąsiada.

Rozwiązanie dla $N \leqslant 10$

Możemy zasymulować wykładniczo każdą możliwość przeprowadzania procesu.

Rozwiązanie dla grafu będącego ścieżką

W tym podzadaniu wystarczy znać odległości punktów startowych od siebie i względem końców ścieżki. Zależność względem powyższych wartości i wynikiem można łatwo wyprowadzić na kartce.

Rozwiązanie dla jednego wierzchołka startowego

Rozważmy najpierw prostszy wariant zadania, taki, w którym będziemy mieli tylko jeden wierzchołek startowy. Zadanie to możemy rozwiązać prostym programowaniem dynamicznym. Niech $\mathrm{dp}[u]$ oznacza minimalny czas potrzeby do pokolorowania całego poddrzewa wierzchołka u, jeśli u jest już pokolorowany, ale całe jego poddrzewo nie. Co oczywiste dla każdego liścia wartość dp będzie równa 0. Rozważmy teraz w jaki sposób policzyć wartość $\mathrm{dp}[u]$, przy znajomości wartości $\mathrm{dp}[v_i]$ gdzie wierzchołki v_i są dziećmi wierzchołka u. Zauważmy, że najbardziej opłacalnym ruchem będzie pokolorowanie najpierw tego dziecka v_i, którego wartość $\mathrm{dp}[v_i]$ jest największa (trywialny dowód pominę). Stąd dostajemy wzór:

$$\mathrm{dp}[u] = \max_{0 \leqslant i \leqslant k} (\mathrm{dp}[V_i] + i)$$

przy założeniu, że $v_0, v_1, v_2, \ldots v_{k-1}, v_k$ są dziećmi wierzchołka u posortowanymi malejąco względem wartości ich wartości dp.

Rozwiązanie dla $N \leqslant 1\,000$

Zauważmy, że po dowolnym pokolorowaniu drzewa zgodnym z zasadami będzie istniała tylko jedna krawędź łącząca dwa wierzchołki pokolorowane przez różne wierzchołki startowe. Zauważmy, że krawędź ta będzie leżała na ścieżce pomiędzy wierzchołkami startowymi. Rozwiązanie tego podzadania będzie więc polegało na sprawdzeniu każdej możliwości doboru takiej krawędzi usunięciu jej z grafu, a następnie policzenia wyniku z użyciem programowania dynamicznego dla dwóch osobnych przypadków. Otrzymamy w ten sposób złożoność $\mathcal{O}\left(N^2 \cdot \log N\right)$.

Rozwiązanie dla grafu generowanego losowo

W grafie generowanym losowo długość ścieżki pomiędzy dwoma dowolnymi wierzchołkami jest rzędu $\mathcal{O}\left(\log N\right)$. Zatem rozwiązanie do poprzedniego podzadania będzie działało w złożoności $\mathcal{O}\left(N \cdot \log^2 N\right)$.

Rozwiązanie wzorcowe

Spróbujmy przyspieszyć rozwiązanie dla poprzedniego podzadania. Usuńmy na początku krawędź ze środka ścieżki łączącej dwa wierzchołki startowe i policzmy wyniki dla obu powstałych w ten sposób grafów. Jeśli wyniki są równe to łatwo zaobserwować, że jest to optymalny podział. Jednak gdy jeden z nich jest większy, to wiemy, że na pewno nie będzie opłacało się przesuwać krawędzi dzielącej w jedną ze stron. Możemy zrobić zatem wyszukiwanie binarne na ścieżce łączącej wierzchołki startowe, kierując się tym po której stronie cięcia jest większy wynik. Otrzymujemy w ten sposób złożoność $\mathcal{O}\left(N \cdot \log^2 N\right)$, która wystarcza do zrobienia zadania.

Część II

Grupa finalistów

4 Uczestnicy

Uczestnicy zajęć: Antoni Buraczewski, Igor Hańczaruk, Michał Januszkiewicz, Julia Kędziorska, Mateusz Kowalski, Wiktor Krzemiński, Karol Łacina, Rafał Mańczyk, Marek Muzyka, Krzysztof Olejnik, Michał Plata i Paulina Żeleźnik.

Zajęcia prowadzili Anadi Agrawal i Marcin Knapik.

5 Treści zadań

Szkoła Strzelecka (szkola-strzelecka)

Memory limit: 256 MB Time limit: 1.00 s

Piotr właśnie został wybrany na instruktora strzelectwa w słynnej szkole *Strzelec Wyborowy*. Pod swoją opiekę dostał on M kadetów, dla których przygotował on N różnych testów.

Każdy test charakteryzuje się pewną trudnością, trudność i-tego testu wynosi P_i. Kadet, który prawidłowo zaliczy dany test otrzymuje za niego tyle punktów ile wynosi jego trudność.

Po przetestowaniu uczestników na przygotowanych testach, Piotr był pod wrażeniem wyników swoich podopiecznych. Nie chce on jednakże, aby zbytnio "obrośli w piórka", skoro wciąż jest pole do poprawy.

Postanowił on podzielić przygotowane testy na K różnych grup. Każdy test będzie należał do dokładnie jednej grupy. Ponadto, indeksy testów w danej grupie będą tworzyły spójny przedział. Oznacza to, że jeśli testy i oraz j będą w jednej grupie, to również każdy test k, który spełnia $i \leq k \leq j$ również będzie należał do tej grupy.

Po takim pogrupowaniu, kadet zdobywa punkty za test z danej grupy, wtedy i tylko wtedy, gdy zaliczy wszystkie testy z tej grupy.

Twoim zadaniem jest pomóc Piotrowi znaleźć dla każdego K spełniającego $1 \leq K \leq S$ minimalną liczbę punktów, jaką sumarycznie mogą zdobyć kadeci przy optymalnym pogrupowaniu testów.

Wejście

W pierwszym wejściu standardowego wejścia znajdują się trzy liczby naturalne M, N, S, które oznaczają odpowiednio liczbę kadetów, liczbę testów i maksymalną liczbę grup, dla której Piotr chce poznać odpowiedź.

W drugim wierszu znajduje się N liczb naturalnych pooddzielanych pojedynczym odstępem $P_1, P_2, \ldots, P_N$ – trudności poszczególnych testów.

W każdym z kolejnych M wierszy znajduje się binarny ciąg złożony z dokładnie N znaków 0 i 1. Znak j w i-tym wierszu jest równy 1, jeśli kadet i zaliczył test j, w przeciwnym wypadku jest on równy 0.

Wyjście

Na standardowe wyjście powinieneś wypisać S wierszy. W i-tym wierszu powinna znaleźć się pojedyncza liczba naturalna oznaczająca minimalną sumaryczną liczbę punktów, jaką kadeci mogą zdobyć przy optymalnym pogrupowaniu testów na i grup.

Ograniczenia

- $0 \leq N \leq 20\,000$
- $0 \leq M \leq 50$
- $0 \leq S \leq \min(N, 50)$
- $0 \leq P_i \leq 10\,000$
- W testach wartych 8 punktów $N \leq 40$
- W testach wartych kolejne 9 punktów $N \leq 500$
- W testach wartych kolejne 9 punktów $N \leq 4\,000$

Przykład

Input	Output	Explanation
2 3 3 4 3 5 101 110	0 8 16	Przykładowy optymalny podział testów na dwie grupy, to $[1]$ i $[2, 3]$. W tym podziale, obaj kadeci zdobywają po 4 punkty, więc sumarycznie mają ich 8.

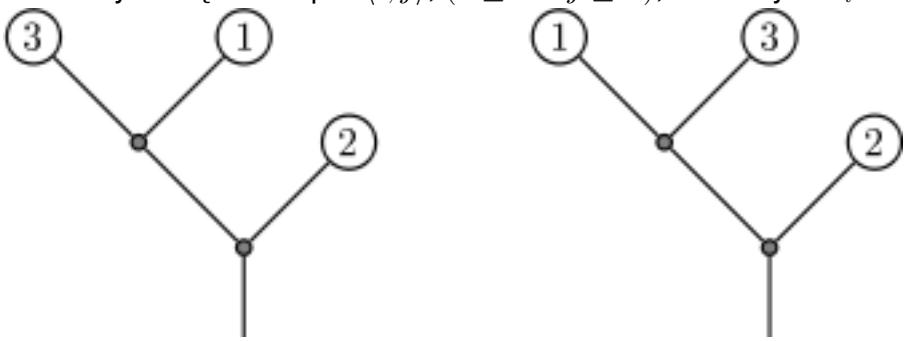
Kino (kino)

Memory limit: 512 MB Time limit: 3.00 s

Bajtola poszła ze znajomymi do kina na sequel jej ulubionego filmu z czasów dzieciństwa. Niestety, po godzinie zorientowała się, że film, chociaż ładny, jest okropnie nudny. Patrząc na malownicze sceny wymyśliła sobie problem algorytmiczny – jak za pomocą *rotacji* efektywnie uporządkować drzewo o następujących własnościach:

- Drzewo składa się z prostych gałęzi, rozgałęzień i liści. Wyrastający z ziemi pień drzewa jest również gałęzią.
- Każda gałąź jest zakończona u góry rozgałęzieniem lub liściem.
- Z rozgałęzienia na końcu gałęzi wyrastają dokładnie dwie dalsze gałęzie – lewa i prawa.
- Każdy liść drzewa zawiera jedną liczbę całkowitą z zakresu $[1, N]$. Liczby w liściach nie powtarzają się.
- Za pomocą pewnych zabiegów, dla dowolnego rozgałęzienia można wykonać tzw. *rotację*, czyli zamienić miejscami lewą i prawą gałąź.

Uporządkowanie drzewa mierzymy liczbą inwersji zawartych w jego koronie, tj. dla korony $a_1, a_2, ..., a_N$ wyznaczamy liczbę takich par $\langle i, j \rangle$, $(1 \leq i < j \leq n)$, dla których $a_i > a_j$.



Powyższy rysunek przedstawia oryginalne drzewo (po lewej) o koronie $3, 1, 2$ zawierające dwie inwersje. Po rotacji otrzymujemy drzewo (po prawej) o koronie $1, 3, 2$, które zawiera tylko jedną inwersję. Każde z tych drzew ma 5 gałęzi.

Bajtola chce rozwiązać zadanie zanim film się skończy! Pomóż jej wyznaczyć minimalną liczbę inwersji zawartych w koronie drzewa, które można otrzymać za pomocą rotacji wyjściowego drzewa.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita N, oznaczająca liczbę liści drzewa wymyśloniego przez Bajtolę. W kolejnych wierszach znajduje się opis drzewa. Drzewo definiujemy rekurencyjnie:

- Jeśli na końcu pnia (czyli gałęzi, z której wyrasta drzewo) znajduje się liść z liczbą całkowitą p, to opis drzewa składa się z jednego wiersza zawierającego jedną liczbę całkowitą – p,
- Jeśli na końcu pnia znajduje się rozgałęzienie, to opis drzewa składa się z trzech części:
  - pierwszy wiersz opisu zawiera jedną liczbę 0,
  - po tym następuje opis lewego poddrzewa (tak, jakby lewa gałąź wyrastająca z rozgałęzienia była jego pniem),
  - a następnie opis prawego poddrzewa (tak, jakby prawa gałąź wyrastająca z rozgałęzienia była jego pniem).

Wyjście

W jedynym wierszu standardowego wyjścia należy wypisać jedną liczbę całkowitą – minimalną liczbę inwersji w koronie drzewa, które można otrzymać za pomocą jakiegoś ciągu rotacji wyjściowego drzewa.

Ograniczenia

$0 \leq N \leq 1\,000\,000$.

W testach wartych przynajmniej 30 punktów zachodzi dodatkowy warunek $N \leq 5000$.

Przykład

Input	Output	Explanation
3 0 0 3 1 2	1	Rysunek ilustruje drzewo z przykładu.

Pandora (pandora)

Memory limit: 512 MB Time limit: 2.00 s

Na Pandorę znowu powrócili ludzie chcący wyeksploatować planetę. Jacek zdaje sobie sprawę, że nie ochroni całej Pandory, więc próbuje przynajmniej obronić jej najbardziej wrażliwe punkty.

Pandorę możemy przedstawić w postaci kwadratowej planszy o wymiarach $N \times M$. W niektórych polach tej planszy znajdują się wrażliwe punkty Pandory, na których ochronie szczególnie Jackowi zależy. W szczególności zawsze takim polem jest jego rodzinna wioska znajdująca się w lewym górnym rogu planszy.

Jedną z możliwości ochrony wszystkich punktów jest otoczenie ich murem, który odetnie je od reszty świata (w szczególności od wszystkiego co jest poza planszą). Taki mur musi być spójny, możemy go przedstawić w taki sposób, że zaczniemy jego budowę w lewym górnym rogu planszy, a następnie będziemy przechodzili po kolejnych bokach pól naszej planszy.

Na niektórych bokach postawienie muru może być bardziej kosztowne niż na innych ze względu na bardziej grząski teren Pandory, więc dla każdego boku został wyznaczony koszt postawienia na tym kawałku muru. Jeśli przez jeden kawałek w powyżej opisanym procesie przejdziemy wielokrotnie, to za każdym razem musimy za niego zapłacić.

Jako że środki, którymi dysponuje Jacek są dość ograniczone, to chcemy chcemy postawić mur jak najtańszym kosztem. Twoim zadaniem jest obliczenie tego kosztu.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby naturalne N, M, które oznaczają wymiary planszy.

W kolejnych N wierszach znajduje się po M liczb całkowitych pooddzielanych pojedynczym odstępem, które oznaczają czy dane pole jest wrażliwym punktem Pandory. Liczba 1 reprezentuje wrażliwe pole, a 0 reprezentuje mało wrażliwe pole.

W kolejnych N wierszach znajduje się po $M+1$ liczb całkowitych pooddzielanych pojedynczym odstępem, które oznaczają koszt postawienia muru na pionowych bokach pól. Dokładniej mówiąc, c-ta liczba w r-tym wierszu oznacza koszt postawienia muru pomiędzy polem (r, c) i $(r, c-1)$. Przyjmujemy, że pola które nie znajdują się w środku planszy reprezentują świat zewnętrzny.

W kolejnych $N+1$ wierszach znajduje się po M liczb całkowitych pooddzielanych pojedynczym odstępem, które oznaczają koszt postawienia muru na poziomych bokach pól. Dokładniej mówiąc, c-ta liczba w r-tym wierszu oznacza koszt postawienia muru pomiędzy polem (r, c) i $(r-1, c)$. Przyjmujemy, że pola, które nie znajdują się w środku planszy reprezentują świat zewnętrzny.

Wyjście

W jedynym wierszu wyjścia powinna się znaleźć jedna liczba całkowita oznaczająca minimalny koszt postawienia muru spełniającego wymagania z treści zadania.

Ograniczenia

- $1 \leq N, M \leq 400$
- Koszt postawienia muru na dowolnym boku v spełnia $1 \leq v \leq 10^9$
- W testach wartych łącznie 30 punktów zachodzi $N, M \leq 40$ i jest co najwyżej 10 wrażliwych punktów Pandory
- W innych testach wartych łącznie 30 punktów zachodzi $N, M \leq 40$

Przykłady

Input	Output

```
3 3
1 0 0
1 0 0
0 0 1
1 4 9 4
1 6 6 6
1 2 2 9
1 1 1
4 4 4
2 4 2
6 6 6
```

```
38
```

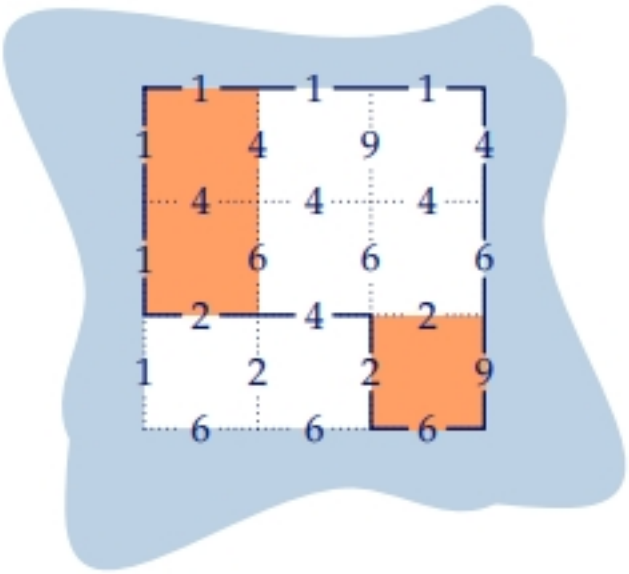
Input

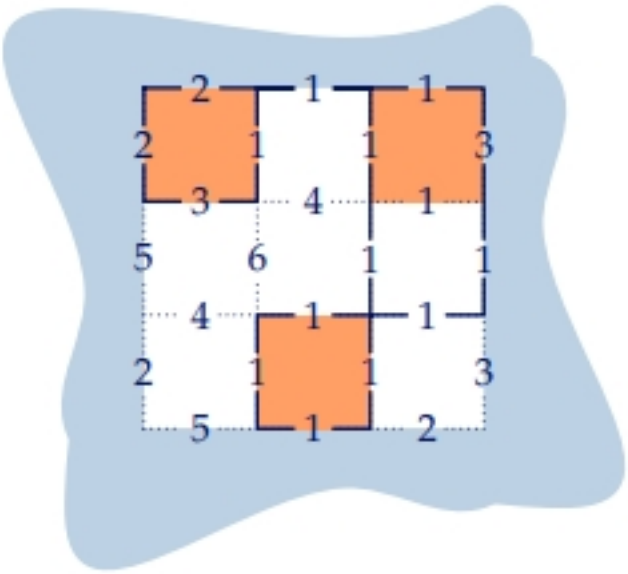
```
3 3
1 0 1
0 0 0
0 1 0
2 1 1 3
5 6 1 1
2 1 1 3
2 1 1
3 4 1
4 1 1
5 1 2
```

Output

```
22
```

Poniższe obrazki odpowiadają testom przykładowym. Na pomarańczowo zaznaczono wrażliwe punkty Pandory, a pogrubioną kreską jest oznaczony optymalny plan muru.





Plan treningowy (plan-treningowy)

Memory limit: 256 MB Time limit: 2.00 s

Po nabawieniu się kolejnej kontuzji na siłowni Marcin stwierdził, że potrzebny mu lepszy plan treningowy. W tym celu wypisał wszystkie znane mu ćwiczenia na tablicy i zaczął się zastanawiać w jakiej kolejności je ułożyć.

Optymalizując efektywność treningu, niektórych par ćwiczeń nie powinno się wykonywać bezpośrednio po sobie. Marcin połączył krawędziami wszystkie pary ćwiczeń, które **mogą** być wykonane jedno po drugim.

Marcin lubi powtarzalność i chciałby realizować swój plan kilka razy podczas jednego treningu, oczywiście bez przerw w środku, a więc pierwsze i ostatnie ćwiczenie również powinny być połączone krawędzią. Z drugiej strony zależy mu na różnorodności, więc w planie nie powinno być powtórzeń i ma zawierać **co najmniej** 4 ćwiczenia. Ostatnim wymaganiem Marcina jest to, żeby żadne dwa ćwiczenia w planie, których nie będzie wykonywał bezpośrednio po sobie, **nie miały** między sobą krawędzi (inaczej Marcinowi mogłoby się pokićkać i w pewnym momencie zacząłby robić złe ćwiczenie).

Pomóż Marcinowi znaleźć satysfakcjonujący plan i wypisz go na standardowe wyjście.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite N i R, oznaczające odpowiednio liczbę ćwiczeń i liczbę krawędzi na tablicy Marcina.

W i-tym z następnych R wierszy znajdują się dwie liczby całkowite a_i i b_i $(1 \leq a_i, b_i \leq N, a_i \neq b_i)$, oznaczające, że ćwiczenia a_i i b_i są połączone krawędzią. Między każdą parą ćwiczeń Marcin narysował co najwyżej jedną krawędź.

Wyjście

Wypisz na standardowe wyjście ciąg $s_1, \ldots, s_m$ parami różnych liczb całkowitych oddzielonych pojedynczymi odstępami, oznaczających kolejne numery ćwiczeń w planie treningowym spełniającym wymagania Marcina.

Jeśli istnieje wiele poprawnych planów, możesz wypisać dowolny z nich.

Jeśli nie istnieje żaden poprawny plan, wypisz `no`.

Ograniczenia

$0 \leq N \leq 1\,000$, $0 \leq R \leq 100\,000$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 10$, $R \leq 25$	30
2	$N \leq 100$, $R \leq 1000$	20
3	$N \leq 300$, $R \leq 6000$	20
4	$N \leq 1000$, $R \leq 100\,000$	30

Przykład

Input	Output
5 6 1 2 1 3 2 3 4 3 5 2 4 5	2 5 4 3

Input	Output
4 5 1 2 2 3 3 4 4 1 1 3	no

Idol (idol)

Memory limit: 256 MB Time limit: 2.00 s

Zawodnik Baj_37 bardzo lubi pieścić swoje ego. W tym celu postanowił porównać swój ranking na platformie BitForces ze słynnym zawodnikiem o pseudonimie Bajtadi, tak by znaleźć jakieś statystyki porównujące go z legendarnym zawodnikiem.

Niestety gdy popatrzymy na ranking zawodnika Baj_37... ładnie to ujmując... odbiega wartościami rankingu od tych Bajtadiego. Baj_37 postanowił się jednak jeszcze nie poddawać.

Pomyślał, że mimo iż jego ranking jest być może nieco niższy od rankingu Bajtadiego, to może przynajmniej różnice pomiędzy kolejnymi wysokościami rankingu ma podobne! Żeby się zanadto nie dołować postanowił nie patrzeć na dokładne wartości rankingu Bajtadiego, a jedynie na to, czy po kolejnych konkursach jego ranking się zwiększał, zmniejszał, albo pozostawał bez zmian.

Baj_37 ma dostęp do ciągu A rozmiaru N – kolejnych wartości swojego rankingu, oraz do ciągu B rozmiaru K symboli ze zbioru $\{<, >, =\}$ oznaczających zmiany rankingu Bajtadiego.

Niestety, wciąż może się okazać, że zmiany rankingu Bajtadiego i Baja_37 nie były takie same, dlatego Twoim zadaniem jest znalezienie największego podzbioru wartości rankingów Baja_37, takich że gdyby pominąć wszystkie inne wartości, to prefiks zmian rankingu Baja_37 zgadzałby się z prefiksem zmian rankingu Bajtadiego (w szczególności każdy podzbiór o rozmiarze 1 zawsze będzie się zgadzał z pustym prefiksem zmian rankingu legendarnego zawodnika). Baj_37 uznał, że czasami może mu nie wystarczyć dopasowanie nawet do całego ciągu zmian Bajtadiego i wtedy traktuje go on jako ciąg cykliczny, czyli dla każdego $i > K$ mówi, że $B_i = B_{i-K}$.

Szukany podzbiór należy wypisać na standardowym wyjściu.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby naturalne N, K oznaczające liczbę stanów rankingu Baja_37 oraz liczbę zmian rankingu Bajtadiego.

W drugim wierszu wejścia znajdują się liczby $A_1, A_2, \ldots, A_N$ pooddzielane pojedynczymi odstępami, które oznaczają kolejne wartości rankingu zawodnika Baj_37.

W trzecim (ostatnim) wierszu wejścia znajdują się symbole $B_1, B_2, \ldots, B_K$ złożone ze znaków <, >, albo =, pooddzielanych pojedynczymi odstępami oznaczające odpowiednio zwiększenie, zmniejszenie, lub pozostanie bez zmian rankingu Bajtadiego po kolejnych konkursach na BitForces.

Wyjście

W pierwszym wierszu wyjścia powinna się znaleźć jedna liczba całkowita m oznaczająca rozmiar największego podzbioru spełniającego wymogi zadania.

W drugim (ostatnim) wierszy wyjścia należy wypisać ciąg m wartości pooddzielanych pojedynczymi odstępami, które są podciągiem ciągu A, a zarazem zgadzają się z odpowiednim prefiksem ciągu B.

Ograniczenia

$1 \leq N, K \leq 500\,000$, $1 \leq A_i \leq 1\,000\,000$,

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 500$, $K \leq 30$	18
2	$N \leq 20\,000$, $K \leq 100$	25
3	Brak dodatkowych ograniczeń	57

Przykład

Input	Output
7 3 2 4 3 1 3 5 3 < > =	6 2 4 3 3 5 3

Bliźniaczy Totem **(twinning-totem)**

Memory limit: 256 MB Time limit: 1.00 s

"Ori, oto Kamienna Mapa, jeden z wielu pradawnych artefaktów stworzonych do pomiaru lasu Nibel w trakcie jego rozrastania."

Kamienna Mapa, z którą ma dzisiaj do czynienia Ori, różni się nieco od tej opisanej powyżej. Totem ten ma dwie strony, w których wyżłobione jest n otworów oraz m rys. Każda rysa łączy dwa otwory, oraz zarówno otwory jak i rysy po obu stronach totemu są takie same. Każdy otwór jest wyżłobiony na wylot, więc jeżeli przyłożymy do niego coś po jednej stronie, to jest to również widoczne po stronie drugiej. Rysy są wyżłobione po obu stronach w ten sam sposób, ale nie przebijają totemu na wylot.

Ori przyłożył jedną *Komórkę Życia* do otworu o numerze u oraz $n-1$ *Komórek Energii* do pozostałych $n-1$ otworów. Następnie Ori musi wybrać $n-1$ rys po każdej ze stron i tchnąć w nie światło (nie pytajcie mnie jak to zrobi, ja tylko tłumaczę treść) by uformować dwa zakorzenione w u drzewa spinające (po jednym na stronie). Jeżeli dla każdej Komórki Energii, przekaże ona swoją energię do wszystkich krawędzi na ścieżce do Komórki Życia (korzenia) w drzewie po jednej stronie, a następnie otrzyma energię z powrotem korzystając z krawędzi drzewa spinającego po drugiej stronie, to żadna inna komórka (poza nią samą i korzeniem) nie może być na tej trasie odwiedzona więcej niż raz.

Czy Ori może znaleźć dwa drzewa spełniające powyższe wymagania?

Wejście

W pierwszej wersji wejścia znajdują się dwie liczby n, m $\left(3 \leq n \leq 10^5, n-1 \leq m \leq \min(2 \cdot 10^5, \frac{n(n-1)}{2})\right)$, oznaczająca odpowiednio liczbę otworów oraz rys na każdej ze stron totemu G.
Spośród kolejnych m wierszy, i-ty zawiera dwie liczby całkowite x_i, y_i $(1 \leq x_i, y_i \leq n)$, oznaczające rysę łączącą otwory x_i i y_i.
Kolejny wiersz zawiera liczbę całkowitą T $(1 \leq T \leq 10^5)$ oznaczającą liczbę zapytań, które zadaje Ori.
Spośród kolejnych T wierszy, i-ty zawiera liczbę całkowitą u_i $(1 \leq u_i \leq n)$, oznaczającą numer otworu, przy którym Ori umieścił Komórkę Życia w i-tym zapytaniu.
Gwarantujemy, że G spójnym grafem bez pętli i wielokrotnych krawędzi oraz, że $n \cdot T \leq 10^5$.

Wyjście

Dla każdego zapytania Ori, jeżeli nie istnieją dwa drzewa spełniające warunki zadania, wypisz jedno słowo `No`. W przeciwnym przypadku w pierwszym wierszu wypisz `Yes`, a w kolejnych $2(n-1)$ wierszach opisz dwa drzewa spinające. W pierwszych $n-1$ wierszach, i-ty wiersz musi zawierać dwie liczby x_i, y_i $(1 \leq x_i, y_i \leq n)$ oddzielone pojedynczym odstępem, oznaczające wybraną rysę łączącą x_i i y_i. $n-1$ rys musi tworzyć drzewo spinające G. Kolejne $n-1$ wierszy przetwórz w ten sam sposób. Twoja odpowiedź zostanie zaakceptowana tylko gdy dwa drzew, które wypisujesz poprawnie rozpinają G I spełniają wymogi zadania (Czyli, dla każdych $v \in V, v \neq u$, ścieżka prosta z u do v na dwóch drzewach nie mają wspólnych wierzchołków, poza końcami u i v).

Przykłady

Input	Output
4 6 1 2 1 3 1 4 2 3 2 4 3 4 1 1	Yes 1 2 2 3 3 4 3 2 4 3 1 4

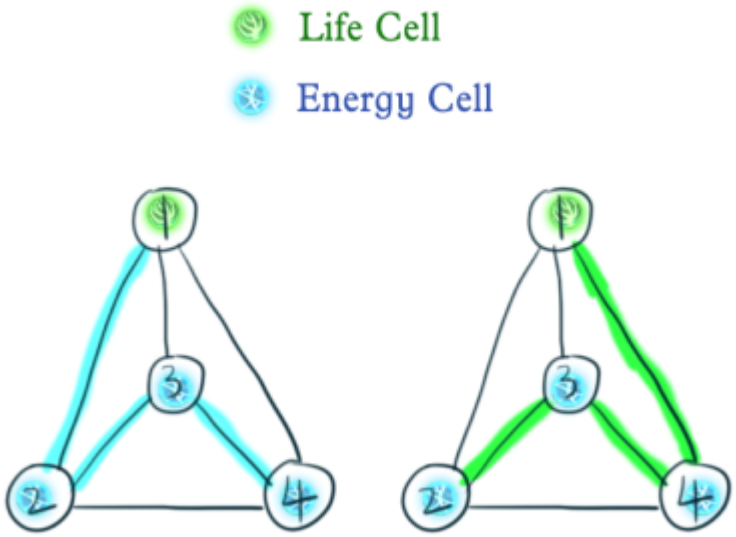
Input	Output
4 4 1 3 2 3 2 4 3 4 4 1 2 3 4	No No Yes 3 1 4 2 3 4 3 1 3 2 2 4 No

Podzadania

Istnieje grupa testowa z $n \leq 1000$ warta 40 procent maksymalnej punktacji.

Wyjaśnienie do przykładu

Dla pierwszego przykładu:



- Dla otworu 2, ścieżki na drzewie to $P_1 = \{2, 1\}, P_2 = \{2, 3, 4, 1\}$.
- Dla otworu 3, ścieżki na drzewie to $P_1 = \{3, 2, 1\}, P_2 = \{3, 4, 1\}$.
- Dla otworu 4, ścieżki na drzewie to $P_1 = \{4, 3, 2, 1\}, P_2 = \{4, 1\}$.

Trójsegmentowe tunele **(trojsegmentowe-tunele)**

Memory limit: 32 MB Time limit: 1.00 s

Mogłoby się wydawać, że Wrocław jest wielkim miastem. To jednak iluzja. Tak naprawdę miasto ma kształt linii, a wszystkie domy znajdują się przy jednej, rozciągającej się z zachodu na wschód ulicy. Są one ponumerowane od 1 do N włącznie. Niestety taki plan miasta niesie ze sobą wiele niedogodności komunikacyjnych. Główna ulica jest bardzo często zakorkowana. Jako iż jesteś światowej sławy projektantem infrastruktury urbanistycznej, to prezydent Wrocławia poprosił Cię o pomoc w uwikłaniu się z tym problemem.

Na szczęście masz pomysł na to, jak pomóc mieszkańcom Wrocławia. Wrocław znany jest ze swoich tramwai, a dokładniej z tego, jak często się wykolejają. Aby ograniczyć ten problem, prezydent postanowił, że no nowe linie tramwajowe będą łączyły bezpośrednio dwa domy i nie będzie żadnych pośrednich przystanków. Dodatkowym wymaganiem jest jeszcze, żeby tramwaje nie szpeciły krajobrazu, więc postanowiono, że będą się one poruszać w tunelach.

Aby sieć tuneli była sprawna, między każdymi dwoma domami musi istnieć (niekoniecznie bezpośrednie) połączenie tunelami, które:

- biegnie cały czas w jednym kierunku (ze wschodu na zachód albo odwrotnie),
- składa się z maksymalnie trzech tuneli.

Jako iż budżet miejskiej spółki komunikacyjnej jest ograniczony, to jesteś zmuszony do zaprojektowania sieci, w której znajduje się maksymalnie $5 \cdot N$ tuneli. Jeżeli to zadanie Cię przerośnie, to prezydent Wrocławia nadal zaoferuje Ci częściową nagrodę za plan zawierający nie więcej niż $10 \cdot N$ tuneli.

Napisz program, który wczyta liczbę domów we Wrocławiu oraz rodzaj podzadania i wypisze na standardowe wyjście plan sieci spełniającej warunki prezydenta.

Wejście

W pierwszym (jedynym) wierszu standardowego wejścia znajdują się dwie liczby naturalne N oraz R, określające liczbę domów we Wrocławiu, oraz numer podzadania, którego dotyczy dany zestaw testowy.

Wyjście

W pierwszym wierszu standardowego wyjścia wypisz liczbę tuneli T, zawartych w Twoim planie. W kolejnych T wierszach wypisz opisy tuneli, składające się z dwóch liczb naturalnych – numerów budynków, które mają być połączone bezpośrednim tunelem.

Nie ma znaczenia, w jakiej kolejności zostaną wypisane tunele oraz budynki połączone pewnym konkretnym tunelem, ale każdy tunel może znaleźć się w planie maksymalnie raz oraz nie jest dozwolone łączenie domu z samym sobą. W przypadku przekroczenia limitu użytych tuneli, zgłoszenie otrzyma werdykt błędnej odpowiedzi.

Jeżeli istnieje wiele odpowiedzi, to wypisz dowolną z nich.

Ograniczenia

$2 \leq N \leq 500$.

Podzadania

Podzadanie	Warunki	Punkty
1	można użyć $10 \cdot N$ tuneli	40
2	można użyć $5 \cdot N$ tuneli	60

Testy ocen

1. $N = 15$, pierwsze podzadanie
2. $N = 74$, drugie podzadanie
3. $N = 100$, pierwsze podzadanie
4. $N = 100$, drugie podzadanie

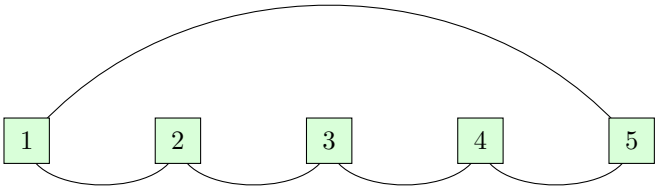
Przykład

Input

```
5 1
```

Output

```
7
1 2
2 3
3 4
4 5
1 3
1 4
2 4
```

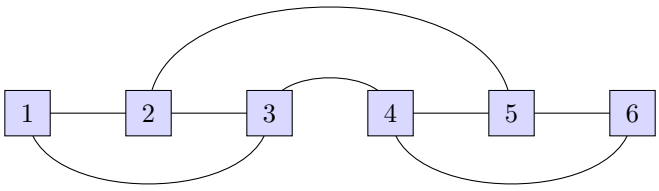


Input

```
6 2
```

Output

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



Promieniowanie (promieniowanie)

Memory limit: 256 MB Time limit: 3.00 s

Dawno temu, mieszkańcy Krzyżowej zbudowali wiele elektrowni jądrowych. Wszystkie dobrze działały przez lata, aż pewnego razu gminę nawiedziło trzęsienie ziemi. Katastrofa spowodowała eksplozję elektrowni jądrowych i promieniowanie zaczęło się rozprzestrzeniać. Gdy udało się okiełznać żywioł, wójt gminy zaczął szacować straty. Twoim zadaniem jest napisać program, który będzie odpowiadał na zapytania o napromieniowanie pewnych regionów Krzyżowej.

Gminę Krzyżowa można przedstawić jako prostokąt składający się z $W \times H$ pól. Każda elektrownia jądrowa zajmuje jedno pole i jest sparametryzowana dwiema liczbami całkowitymi: a i b. Wartość a jest natężeniem promieniowania w polu, w którym znajduje się elektrownia, a wartość b określa jak szybko promieniowanie się zmniejsza w miarę oddalania się od elektrowni.

Dokładniej, natężenie promieniowania, które dotrze do pola $C = [x_C, y_C]$, od elektrowni w polu $P = [x_P, y_P]$, jest równe:

$$\max(0, a - b \cdot d(P, C)),$$

gdzie $d(P, C)$ jest odległością między polami P i C zdefiniowaną tak:

$$d(P, C) = \max(|x_P - x_C|, |y_P - y_C|).$$

Całkowite promieniowanie w jednym polu jest równe sumie promieniowań pochodzących od poszczególnych elektrowni jądrowych.

Dla przykładu, rozważmy elektrownię z $a = 7$ i $b = 3$. Jej wybuch spowoduje promieniowanie wielkości 7 w polu, w którym się ona się znajduje, promieniowanie wielkości 4 w ośmiu sąsiednich polach i promieniowanie wielkości 1 w szesnastu polach, odległych o 2. Gdyby elektrownia była położona na granicy Krzyżowej, lub jedno pole od granicy, to wybuch wpłynąłby również na pewne pola poza gminą. Eksplozję, której promieniowanie rozprzestrzenia się poza granice Krzyżowej, nazwijmy eksplozją graniczną. (W zadaniu nie jest istotne, czy eksplozja jest graniczna, czy nie. Ta definicja przyda się po prostu w sekcji *Podzadania*).

Wójt gminy za każdym razem pyta o średnie napromieniowanie pola w danym prostokątnym regionie. W urzędzie gminy panuje wielka dezorganizacja, więc nie zakładaj niczego na temat zadanych regionów – mogą się powtarzać, pokrywać, zawierać...

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie dodatnie liczby całkowite W i H, oddzielone pojedynczym odstępem i oznaczające odpowiednio szerokość i wysokość Krzyżowej. W drugim wierszu znajduje się liczba całkowita N, oznaczająca liczbę elektrowni jądrowych, które wybuchły. W kolejnych N wierszach znajdują się po cztery liczby całkowite x_i, y_i, a_i, b_i $(1 \leq x_i \leq W, 1 \leq y_i \leq H)$, oznaczające, że na polu $[x_i, y_i]$ wybuchła elektrownia o parametrach a_i, b_i. Na każdym polu znajduje się co najwyżej jedna elektrownia.

W następnym wierszu znajduje się jedna liczba całkowita Q, oznaczająca liczbę zapytań wójta. W kolejnych Q wierszach znajdują się po cztery liczby całkowite x_{1j}, y_{1j}, x_{2j}, y_{2j} $(1 \leq x_{1j} \leq x_{2j} \leq W, 1 \leq y_{1j} \leq y_{2j} \leq H)$, oznaczające zapytanie o region będący prostokątem, którego lewy górny róg jest w polu $[x_{1j}, y_{1j}]$, a prawy dolny róg jest w polu $[x_{2j}, y_{2j}]$.

Możesz założyć, że całkowite promieniowanie w Krzyżowej jest mniejsze niż 2^{63}.

Wyjście

Dla każdego zapytania wypisz jeden wiersz zawierający średnie napromieniowanie pola w zadanym regionie, zaokrąglone do najbliższej liczby całkowitej (połówki zaokrąglane są w górę).

Ograniczenia

$1 \leq N \leq 200\,000$, $1 \leq Q \leq 200\,000$, $W \cdot H \leq 2\,500\,000$, $1 \leq a_i, b_i \leq 10^9$

Podzadania

Jest 14 grup testów. Grupy o nieparzystych numerach zawierają jedynie elektrownie, dla których a jest wielokrotnością b. Dodatkowe warunki w grupach są następujące:

Grupa	Warunki	Punkty
1	$H = 1, N \cdot W \leq 10^8, Q \cdot W \leq 10^8$	3
2	$H = 1, N \cdot W \leq 10^8, Q \cdot W \leq 10^8$	2
3	$N \cdot W \cdot H \leq 10^8, Q \cdot W \cdot H \leq 10^8$	3
4	$N \cdot W \cdot H \leq 10^8, Q \cdot W \cdot H \leq 10^8$	2
5	$H = 1, N \cdot W \leq 10^8$	6
6	$H = 1, N \cdot W \leq 10^8$	4
7	$N \cdot W \cdot H \leq 10^8$	6
8	$N \cdot W \cdot H \leq 10^8$	4
9	$H = 1$	15
10	$H = 1$	10
11	brak eksplozji granicznych	15
12	brak eksplozji granicznych	10
13	brak dodatkowych ograniczeń	12
14	brak dodatkowych ograniczeń	8

Przykład

Input

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

Output

```
4
4
2
2
```

Explanation

Natężenie promieniowania w Krzyżowej po dwóch eksplozjach jest następujące:

$$7\ 6\ 3\ 2$$
$$4\ 6\ 5\ 2$$
$$1\ 3\ 3\ 2$$

Pierwsza eksplozja jest eksplozją graniczną, a druga nie. Jeśli chodzi o zapytania:

- Całkowite promieniowanie w kwadracie 2 na 2 jest równe 14, więc średnie promieniowanie jest równe $\frac{14}{4} = 3.5$ (zaokrąglone do 4).
- Całkowite promieniowanie w Krzyżowej jest równe 44, więc średnie promieniowanie jest równe $\frac{44}{12} \approx 3.67$ (zaokrąglone do 4).
- Średnie promieniowanie w pojedynczym polu jest równe po prostu promieniowaniu w tym polu.
- Średnie promieniowanie w ostatnim wierszu jest równe $\frac{9}{4} = 2.25$ (zaokrąglone do 2).

Kolorowe nawiasowanie **(kolorowe-nawiasowanie)**

Memory limit: 512 MB Time limit: 0.50 s

W trakcie rozmyślań nad zadaniami olimpijskie Karol przykuł swą uwagę do *kolorowych nawiasowań*. Umieścił on poniżej kilka definicji, które pozwolą nam lepiej zrozumieć o co dokładnie mu chodzi.

Ciąg A nawiasów otwierających i zamykających nazwiemy poprawnym nawiasowaniem jeżeli zachodzi chociaż jeden z poniższych warunków:

1. jest pusty,
2. jest postaci (B), gdzie B również jest poprawnym nawiasowaniem,
3. jest postaci PL, gdzie zarówno P jak i L są poprawnymi nawiasowaniami.

Niech B będzie poprawnym nawiasowaniem o długości N. Definiujemy B_i jako i-ty znak w tym nawiasowaniu. Dla dwóch indeksów $1 \leq i < j \leq N$, mówimy, że są ze sobą sparowane jeżeli zachodzą oba poniższe warunki:

1. $B_i =$ '(' oraz $B_j =$ ')',
2. $i = j - 1$ albo nawiasowanie $C = B_{i+1}B_{i+2} \ldots B_{j-1}$ jest poprawne.

Niech S będzie ciągiem małych liter języka angielskiego. Definiujemy S_i jako i-tą literę w tym ciągu. Mówimy, że poprawne nawiasowanie B jest kolorowane przez S jeżeli zachodzą oba poniższe warunki:

1. B i S mają tę samą długość,
2. Dla każdej pary indeksów (i, j) jeżeli B_i i B_j są ze sobą sparowane, to $S_i = S_j$ (litery symbolizują kolory).

Dla danego ciągu S o długości N Karol zastanawia się jakie jest (i czy w ogóle istnieje) najmniejsze leksykograficznie nawiasowanie, które jest kolorowane przez ten ciąg. Ponieważ ma na głowie jeszcze wiele innych ważnych spraw (przede wszystkim zarządzanie prac nad nową wersją systemu Solve), poprosił Cię o pomoc w udzieleniu odpowiedzi na to pytanie. Zakładamy że znak `(` jest mniejszy leksykograficznie od znaku `)`.

Napisz program, który sprawdzi czy szukane nawiasowanie istnieje, a jeżeli tak, to wypisze je na standardowym wyjściu.

Wejście

W pierwszym (jedynym) wierszu wejścia znajduje się napis S składający się z $2 \leq N \leq 100\,000$ małych liter alfabetu angielskiego.

Wyjście

Jeżeli szukane nawiasowanie nie istnieje, to w pierwszym wierszu wyjścia powinna się znaleźć liczba `-1`. W przeciwnym przypadku w wierszu tym powininno znaleźć się najmniejsze leksykograficznie nawiasowanie kolorowane przez S.

Podzadania

Grupa testowa	Dodatkowe warunki	Punkty
1	$N \leq 18$	10
2	$N \leq 2000$	27
3	Brak dodatkowych ograniczeń	63

Przykład

Input	Output	Explanation
`abbaaa`	`(()())`	Innym poprawnym nawiasowaniem jest `((()))`, ale nie jest ono najmniejsze leksykograficznie.

Input	Output	Explanation
`abab`	`-1`	Nie istnieje żadne nawiasowanie kolorowane przez ciąg S.

Cykl flaja **(min-cost-cycle)**

Memory limit: 1024 MB Time limit: 2.00 s

Marcin po swoim wczorajszym wykładzie nie mógł się oderwać od próby rozwiązania *problemu komiwojażera* za pomocą algorytmów ewolucyjnych.

Niestety... próbował i próbował, ale okazywało się, że dla dostatecznie dużych grafów jego zrandomizowane sposoby (choć istotnie ciekawe) nie dawały mu optymalnego wyniku.

"Skoro nie mogę rozwiązać tego problemu, to po prostu rozwiąże nieco inny problem, mówiąc że jest on na podobnym miejscu w skali trudności".

Bohater treści wymyślił następującą wersję problemu – policz minimalny koszt odwiedzenia wszystkich miast oraz powrotu do miasta startowego, jeżeli przejazd między dowolną parą miast i oraz j kosztuje $min(A_i, B_j)$, gdzie A_i oznacza koszt opuszczenia miasta i, a koszt B_j jest związany z opłatą za przybycie do miasta j. Każde miasto należy przy tym odwiedzić dokładnie raz. Wyjątkiem jest miasto startowe, które odwiedzimy dokładnie dwa razy.

Rzeczywiście, taki problem okazuje się "nieco" prostszy do rozwiązania w czasie wielomianowym... ale czy na pewno?

Napisz program, który wczyta liczbę miast, oraz koszta A_i oraz B_i i wyznaczy koszt najtańszego przejazdu spełniającego powyższe kryteria.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita N.

W kolejnych N wierszach znajdują się koszta opuszczenia i wstąpienia do jakiegoś miasta. W i-tym spośród nich znajdują się dwie liczby A_i oraz B_i – parametry dla wierzchołka o numerze i.

Wyjście

W pierwszym (jedynym) wierszu wyjścia powinna się znaleźć jedna liczba całkowita, oznaczająca koszt optymalnego cyklu.

Ograniczenia

$2 \leq N \leq 100\,000$, $1 \leq A_i, B_i \leq 10^9$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 8$	7
2	$N \leq 18$	13
3	$N \leq 1000$	30
4	brak dodatkowych ograniczeń	50

Przykład

Input	Output
3 1 5 4 2 6 3	7

Input	Output

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

```
10
```

Input

```
6
19 92
64 64
78 48
57 33
73 6
95 73
```

Output

```
227
```

Gigantyczny Graf (giant-graph)

Memory limit: 256 MB Time limit: 2.00 s

Po wielu przygodach na obozie z Krzyżowej, nadchodzą ostateczne wyzwania dla najwytrwalszych uczestników. Dla wytrwałych należy się nagroda, a tą nagrodą będzie formalna treść zadania!

Dane są trzy grafy nieskierowane X, Y, Z o N wierzchołkach i odpowiednio M_1, M_2, M_3 krawędziach. Wierzchołki z tych grafów oznaczamy odpowiednio $x_1, \ldots, x_N, y_1, \ldots, y_N, z_1, \ldots, z_N$. W podobny sposób krawędzie z tych grafów oznaczamy przez $(x_a, x_b), (y_a, y_b), (z_a, z_b)$.

Zbudujemy na bazie tych grafów, nieskierowany graf W z N^3 wierzchołkami. Każdy wierzchołek z W odpowiada pewnej trójce wierzchołków z X, Y, Z, oznaczmy tą trójkę przez (x_i, y_j, z_k).

Krawędzie w W są przedstawione w następujący sposób:

- Dla każdej krawędzi (x_u, x_v) z X i w, l, tworzymy krawędź z (x_u, y_w, z_l) do (x_v, y_w, z_l)
- Dla każdej krawędzi (y_u, y_v) z Y i w, l, tworzymy krawędź z (x_w, y_u, z_l) do (x_w, y_v, z_l)
- Dla każdej krawędzi (z_u, z_v) z Z i w, l, tworzymy krawędź z (x_w, y_l, z_u) do (x_w, y_l, z_v)

Wierzchołki w W są ważone. Waga wierzchołka (x_i, y_j, z_k) wynosi $10^{18(i+j+k)}$. Twoim zadaniem jest znaleźć niezależny podzbiór wierzchołków W o maksymalnej wadze. Dla przypomnienia podzbiór wierzchołków jest niezależny, jeśli nie istnieje krawędź łącząca dowolne dwa wierzchołki z tego podzbioru. Jako, że wynik może być bardzo duży, to wystarczy że wypiszesz jego resztę z dzielenia przez $998\ 244\ 353$.

Wejście

W pierwszym wierszu wejścia znajduje się pojedyncza liczba naturalna N – liczba wierzchołków w każdym z grafów X, Y, Z. W kolejnych wierszach następuje opis grafów X, Y, Z.

W pierwszym wierszu opisu każdego grafu znajduje się pojedyncza liczba naturalna M_i – liczba krawędzi w właśnie opisywanym grafie. W każdym z kolejnych M wierszy znajdują się dwie liczby naturalne oddzielone pojedynczym odstępem a_j, b_j ($1 \leq a_j, b_j \leq N$) – krawędź właśnie opisywanego grafu.

Wyjście

Na wyjściu powinna znaleźć się dokładnie jedna liczba całkowita, oznaczająca największą możliwą wagę niezależnego podzbioru W.

Ograniczenia

- $2 \leq N \leq 100\ 000$
- $1 \leq M_i \leq 100\ 000$
- Grafy X, Y, Z są proste – nie posiadają pętli ani multikrawędzi

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 11$	12
2	$N \leq 100$	13
3	$N \leq 2\ 000$	20
4	Brak dodatkowych ograniczeń	55

Przykład

Input	Output

2 1 1 2 1 1 2 1 1 2	46494701

Input	Output
3 3 1 3 1 2 3 2 2 2 1 2 3 1 2 1	883188316

Input	Output
100000 1 1 2 1 99999 100000 1 1 100000	318525248

Najlepsze Brownie **(brownie)**

Memory limit: 512 MB Time limit: 4.00 s

Ewa robi najlepsze Brownie na świecie. Praktykuje sztukę przygotowywania najznamienitszego ciasta na świecie już od wielu lat i doszła w niej do istnej perfekcji. Jednym z wielu sekretów przygotowywania swoistego arcydzieła jest umiejętne rozprowadzenie masła po foremce od ciasta. Ewa postanowiła zrobić dla swojego informatyka najlepsze Brownie na świecie. Jednak ten, zobaczywszy ciasto, stwierdził, że jest jakieś dziwne, bo okrągłe. Ewa jednak nie dała za wygraną i postanowiła przygotować dla swojego informatyka jeszcze jedno ciasto, jednak tym razem w kształcie drzewa – oczywiście informatycznego.

Ewa przygotowała już piękną foremkę składającą się z N wierzchołków i $N-1$ krawędzi między nimi – foremkę w kształcie drzewa. Wierzchołki foremki numerowane są kolejnymi liczbami naturalnymi od 1 do N.

Niestety z tak śmiałym planem wiążą się pewne problemy, już samo rozprowadzenie masła po foremce w kształcie drzewa okazało się problematyczne. Ewa położyła dwie kostki masła w wierzchołkach drzewa o numerach X i Y, teraz musi rozprowadzić masło po całej foremce – wszystkich wierzchołkach drzewa. Ewa może rozsmarować masło z wierzchołka A do wierzchołka B tylko wtedy, gdy w wierzchołku A już znajduje się masło oraz wierzchołki A i B są połączone krawędzią. Rozsmarowanie zajmuje pomijalnie mało czasu, jednak zanim masło będzie mogło być rozsmarowane ponownie z wierzchołka A lub z wierzchołka B, Ewa będzie musiała poczekać minutę. Jest to bardzo ważne podczas rozsmarowywania masła – gdyby nie trzymać się tej reguły ciasto mogłoby nie wyjść idealne, na co Ewa nie może sobie pozwolić. Ewa może w ciągu każdej minuty rozsmarować masło z wielu wierzchołków do wielu innych, jednak w ciągu każdej minuty z każdego wierzchołka masło może zostać rozsmarowane tylko raz i do każdego wierzchołka masło może zostać rozsmarowane tylko raz.

Napisz program, który wczyta opis foremki ciasta i wyznaczy minimalny czas potrzebny Ewie na rozprowadzenie masła po całej foremce.

Wejście

W pierwszym wierszu wejścia znajdują się trzy liczby naturalne N, X i Y pooddzielane pojedynczymi odstępami i oznaczające odpowiednio liczbę wierzchołków foremki i indeksy wierzchołków, w których początkowo znajduje się masło. W kolejnych $N-1$ wierszach znajdują się opisy krawędzi.W i-tym z nich znajdują się dwie liczby naturalne A_i i B_i, oddzielone pojedynczymi odstępami i oznaczające, że wierzchołki o numerach A_i i B_i są połączone krawędzią.

Wyjście

W pierwszym (jedynym) wierszu wyjścia należy wypisać minimalny czas potrzebny Ewie na rozprowadzenie masła po całej foremce na ciasto.

Ograniczenia

$1 \leq N \leq 300\,000$, $1 \leq A_i,\ B_i,\ X,\ Y \leq N$.

Podzadania

Podzadanie	Warunki	Punkty
1	$N \leq 10$	10
2	$N \leq 1\,000$	20
3	Graf jest ścieżką	10
4	Graf został wygenerowany losowo	20
5	brak dodatkowych ograniczeń	40

Przykład

Input

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

Output

```
2
```

Explanation

Ewa w pierwszej minucie może rozprowadzić masło na wierzchołki o numerach 3 i 5, a w drugiej na 4 i 6.

Input

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

Output

```
4
```

Część III

Dodatek

6 Estimathon

1. Ile jest banknotów o nominale 500 zł (stan na koniec roku 2021)?
2. Ile słów zawiera trylogia Sienkiewicza?
3. Ile wynosi suma liczb pierwszych, które są palindromami i są mniejsze od 2023^2?
4. Ile wynosi iloczyn średniej wysokości n.p.m. i liczby mieszkańców wsi Krzyżowa?
5. Ile wynosi suma długości życia wszystkich zmarłych amerykańskich prezydentów (dla każdego prezydenta liczymy rok śmierci − rok urodzenia)?
6. Ile zdobyto goli samobójczych w Ekstraklasie w sezonie 2021/2022?
7. Ile samochodów sprzedała Tesla w Europie w 2022?
8. Ile średnio punktów zdobędzie LeBron James w czasie, który potrzebuje światło na dotarcie od Słońca do sondy Voyager?
9. Ile wynosi iloczyn liczby odcinków Miodowych lat i liczby odcinków Rancza?
10. Ile jest różnych gatunków ptaków w Brazylii?
11. Ile razy przeciętny ślimak zdąży odsłuchać *Never gonna give you up* w czasie przejścia trasy od Pałacu w Krzyżowej do Instytutu Informatyki?
12. Ile wynosi iloczyn miejsc zajmowanych przez Gennady'ego Korotkevicha [tourist] na Międzynarodowej Olimpiadzie Informatycznej?
13. Ile wynosi odwrotność iloczynu najmniejszej i największej odpowiedzi na powyższe pytania?