Omówienie zadań

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Pudełka

Najszybsze rozwiązanie: **Kamil Dębowski (0:10)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Pudełka

Treść zadania

Mamy n pudełek i n dziur. Chcemy wsadzić jak najwięcej pudełek do dziur.
Rozmiary i dziur i pudełek są potęgami dwójki.

Pudełka

Treść zadania

Mamy n pudełek i n dziur. Chcemy wsadzić jak najwięcej pudełek do dziur.
Rozmiary i dziur i pudełek są potęgami dwójki.

Rozwiązanie

1. Sortujemy pudełka rosnąco
2. Sortujemy dziury rosnąco
3. Zachłannie przypisujemy kolejne pudełka do kolejnych dziur – do każdej tyle ile się zmieści

Pudełka

Treść zadania

Mamy n pudełek i n dziur. Chcemy wsadzić jak najwięcej pudełek do dziur.
Rozmiary i dziur i pudełek są potęgami dwójki.

Rozwiązanie

1. Sortujemy pudełka rosnąco
2. Sortujemy dziury rosnąco
3. Zachłannie przypisujemy kolejne pudełka do kolejnych dziur – do każdej tyle ile się zmieści

Złożoność $\mathcal{O}(n \log n)$

Składanie ciągu

Najszybsze rozwiązanie: **Marek Sommer (0:42)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Składanie ciągu

Treść zadania

Dane są pozycje jedynek w ciągu binarnym o długości n
($n = 2^k$, dla pewnego naturalnego k).
Możemy wykonywać operacje odwrócenia dowolnego elementu oraz operacje usuwania prawej połowy ciągu, jeżeli w danym momencie cały ciąg jest palindromem.
Mamy obliczyć ile co najmniej operacji odwrócenia musimy wykonać (i dowolną liczbę usunięć), aby otrzymać ciąg o długości 1.

Składanie ciągu

Treść zadania

Dane są pozycje jedynek w ciągu binarnym o długości n
($n = 2^k$, dla pewnego naturalnego k).
Możemy wykonywać operacje odwrócenia dowolnego elementu oraz operacje usuwania prawej połowy ciągu, jeżeli w danym momencie cały ciąg jest palindromem.
Mamy obliczyć ile co najmniej operacji odwrócenia musimy wykonać (i dowolną liczbę usunięć), aby otrzymać ciąg o długości 1.

- Opłaca nam się usunąć prawą połowę kiedy tylko jest to możliwe.

Treść zadania

Dane są pozycje jedynek w ciągu binarnym o długości n
($n = 2^k$, dla pewnego naturalnego k).
Możemy wykonywać operacje odwrócenia dowolnego elementu oraz operacje usuwania prawej połowy ciągu, jeżeli w danym momencie cały ciąg jest palindromem.
Mamy obliczyć ile co najmniej operacji odwrócenia musimy wykonać (i dowolną liczbę usunięć), aby otrzymać ciąg o długości 1.

- Opłaca nam się usunąć prawą połowę kiedy tylko jest to możliwe.
- Przed usunięciem chcemy odwrócić tylko jeden element z każdej pary $\langle a_i, a_{n-i+1} \rangle$ ($a_i \neq a_{n-i+1}$).

Rozwiązanie pierwsze

Dodajmy dodatkową wartość ?, którą mogą przyjmować elementy. Będzie oznaczać, że zapłaciliśmy za zmianę tego elementu, ale nie ustaliliśmy jeszcze na jaki.
Symulujemy kolejne operacje. Kiedy trafiamy na parę $a_i \neq a_{n-i+1}$, to:

- Jeżeli jeden z elementów jest równy ?, to zmieniamy go na wartość drugiego.
- W przeciwnym wypadku elementy to 0 i 1. Inkrementujemy wynik i ustawiamy oba na ?.

Złożoność: $\mathcal{O}(m \log n \log m)$ lub $\mathcal{O}(m \log n)$ (dwa wskaźniki).

Rozwiązanie drugie

Korzystamy z dodatkowej wartości ? tak jak w rozwiązaniu pierwszym. Myślimy o składaniu ciągu na pół (nakładaniu prawej strony na lewą) zamiast o usuwaniu prawej strony. Zauważmy, że sklejenia elementów tworzą drzewo binarne. Obliczmy rekurencyjnie do jakich wartości wyliczą się elementy w lewym i prawym poddrzewie korzenia. Następnie złączmy te wartości tak jak w poprzednim rozwiązaniu.

Złożoność: $\mathcal{O}(m \log n)$.

Łamigłówka 2

Najszybsze rozwiązanie: **Marcin Smulewicz (2:46)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Dwa przedziały

W pierwszym ciągu wyróżnijmy podsłowo aS, gdzie a jest znakiem, a S słowem długości k.

Dwa przedziały

W pierwszym ciągu wyróżnijmy podsłowo aS, gdzie a jest znakiem, a S słowem długości k.
W drugim ciągu wyróżnijmy podsłowo Tb, gdzie b jest znakiem, a T słowem długości $k-1$.

Dwa przedziały

W pierwszym ciągu wyróżnijmy podsłowo aS, gdzie a jest znakiem, a S słowem długości k.
W drugim ciągu wyróżnijmy podsłowo Tb, gdzie b jest znakiem, a T słowem długości $k-1$.
Spójrzmy jakie operacje możemy wykonać.

Łamigłówka 2

Jakie ruchy wykonywać?

aS

Tb

Łamigłówka 2

Jakie ruchy wykonywać?

aS

Tb

Łamigłówka 2

Jakie ruchy wykonywać?

aTb

S

Łamigłówka 2

Jakie ruchy wykonywać?

aTb
S

Łamigłówka 2

Jakie ruchy wykonywać?

aTb

S

Łamigłówka 2

Jakie ruchy wykonywać?

Sb
aT

Łamigłówka 2

Jakie ruchy wykonywać?

```
Sb
aT
```

W dwóch ruchach możemy przerzucić jeden element z pierwszego ciągu do drugiego i jeden z drugiego do pierwszego!

Łamigłówka 2

Jakie ruchy wykonywać?

```
Sb
aT
```

W dwóch ruchach możemy przerzucić jeden element z pierwszego ciągu do drugiego i jeden z drugiego do pierwszego! Przerzucając elementy w odpowiednią stronę wykonamy co najwyżej $\frac{n}{2}$ operacji, z których każda będzie nas kosztować dwa ruchy.

Czy już zaczynać?

Dysponując BST bylibyśmy w stanie rozwiązać zadanie w złożoności $\mathcal{O}(n \cdot \log(n))$.

Czy już zaczynać?

Dysponując BST bylibyśmy w stanie rozwiązać zadanie w złożoności $\mathcal{O}(n \cdot \log(n))$. Na zawodach biblioteczką nie dysponujemy, warto zatem rozejrzeć się za czymś prostszym i szybszym.

Sprytne podejście

Ustawmy okno długości $k + 1$ na pierwszych elementach pierwszego ciągu.

Łamigłówka 2

Sprytne podejście

Ustawmy okno długości $k + 1$ na pierwszych elementach pierwszego ciągu.
Ustawmy okno długości k na ostatnich elementach drugiego ciągu.

Sprytne podejście

Ustawmy okno długości $k + 1$ na pierwszych elementach pierwszego ciągu.
Ustawmy okno długości k na ostatnich elementach drugiego ciągu.
Utrzymujmy oba okna na kolejkach dwustronnych – opisaną operację zasymulujemy w czasie stałym.

Łamigłówka 2

Sprytne podejście

Ustawmy okno długości $k + 1$ na pierwszych elementach pierwszego ciągu.
Ustawmy okno długości k na ostatnich elementach drugiego ciągu.
Utrzymujmy oba okna na kolejkach dwustronnych – opisaną operację zasymulujemy w czasie stałym.
Tak długo jak nie odpowiada nam pierwszy element okna w pierwszym ciągu, w czasie stałym przesuwajmy okno w prawo – analogicznie w drugim ciągu przesuwajmy je w lewo.

Łamigłówka 2

Sprytne podejście

Ustawmy okno długości $k + 1$ na pierwszych elementach pierwszego ciągu.
Ustawmy okno długości k na ostatnich elementach drugiego ciągu.
Utrzymujmy oba okna na kolejkach dwustronnych – opisaną operację zasymulujemy w czasie stałym.
Tak długo jak nie odpowiada nam pierwszy element okna w pierwszym ciągu, w czasie stałym przesuwajmy okno w prawo – analogicznie w drugim ciągu przesuwajmy je w lewo.
Aby przerzucić wszystko przesuniemy okno co najwyżej liniowo wiele razy.

Graf Skokowy

Najszybsze rozwiązanie: **Marek Sommer (3:08)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Uproszczenie

Wyobraźmy sobie, że tuż przed ciągiem dodajemy wartość $n + 1$, a tuż po ciągu dodajemy wartość $n + 2$ do których możemy skakać.

Graf skokowy

Uproszczenie

Wyobraźmy sobie, że tuż przed ciągiem dodajemy wartość $n + 1$, a tuż po ciągu dodajemy wartość $n + 2$ do których możemy skakać.

Struktura grafu

Spójrzmy na pozycję wartości n.

Graf skokowy

Uproszczenie

Wyobraźmy sobie, że tuż przed ciągiem dodajemy wartość $n + 1$, a tuż po ciągu dodajemy wartość $n + 2$ do których możemy skakać.

Struktura grafu

Spójrzmy na pozycję wartości n.
Dzieli ona ciąg na dwa przedziały – aby przejść między nimi albo musimy skoczyć do któregoś z dodanych elementów, albo do owej wartości n.

Struktura grafu

Wyobraźmy sobie przedział, który po obu swoich stronach ma elementy ściśle większe niż wszystko w tym przedziale (nazwijmy je l i p).

Struktura grafu

Wyobraźmy sobie przedział, który po obu swoich stronach ma elementy ściśle większe niż wszystko w tym przedziale (nazwijmy je l i p).
Aby się w nim poruszać, opłaca się albo w nim zostać, albo raz i od razu skoczyć do któregoś z ograniczających elementów.

Struktura grafu

Wyobraźmy sobie przedział, który po obu swoich stronach ma elementy ściśle większe niż wszystko w tym przedziale (nazwijmy je l i p).
Aby się w nim poruszać, opłaca się albo w nim zostać, albo raz i od razu skoczyć do któregoś z ograniczających elementów.
Przydatnym byłoby obliczać sumę odległości od l i od p do wszystkich elementów między nimi.

Graf skokowy

Struktura grafu

Wyobraźmy sobie przedział, który po obu swoich stronach ma elementy ściśle większe niż wszystko w tym przedziale (nazwijmy je l i p).
Aby się w nim poruszać, opłaca się albo w nim zostać, albo raz i od razu skoczyć do któregoś z ograniczających elementów.
Przydatnym byłoby obliczać sumę odległości od l i od p do wszystkich elementów między nimi.

Kluczowa obserwacja

Dla każdego i między l i p odległości od l i od p do i mogą różnić się co najwyżej o 1!

Graf skokowy

Struktura grafu

Wyobraźmy sobie przedział, który po obu swoich stronach ma elementy ściśle większe niż wszystko w tym przedziale (nazwijmy je l i p).
Aby się w nim poruszać, opłaca się albo w nim zostać, albo raz i od razu skoczyć do któregoś z ograniczających elementów.
Przydatnym byłoby obliczać sumę odległości od l i od p do wszystkich elementów między nimi.

Kluczowa obserwacja

Dla każdego i między l i p odległości od l i od p do i mogą różnić się co najwyżej o 1!
Obliczajmy zatem sumę odległości od l oraz liczbę wierzchołków do których p ma bliżej lub dalej o 1.

Finalne podejście

Rozważajmy elementy ciągu od najmniejszych. Załóżmy, że aktualnie rozważany element x łączy rozważone już przedziały A i B, które są ograniczone od lewej i prawej przez jeszcze nierozważone elementy l i p – ciekawy przedział jest postaci $lAxBp$.

Finalne podejście

Rozważajmy elementy ciągu od najmniejszych. Załóżmy, że aktualnie rozważany element x łączy rozważone już przedziały A i B, które są ograniczone od lewej i prawej przez jeszcze nierozważone elementy l i p – ciekawy przedział jest postaci $lAxBp$.
Aby poruszać się w przedziale AxB nie opłaca się wychodzić poza przedział $lAxBp$.

Finalne podejście

Rozważajmy elementy ciągu od najmniejszych. Załóżmy, że aktualnie rozważany element x łączy rozważone już przedziały A i B, które są ograniczone od lewej i prawej przez jeszcze nierozważone elementy l i p – ciekawy przedział jest postaci $lAxBp$.
Aby poruszać się w przedziale AxB nie opłaca się wychodzić poza przedział $lAxBp$.
Aby obliczyć wspomniane wartości dla przedziału AxB wystarczą nam obliczone już wartości dla przedziałów A oraz B.

Finalne podejście

Rozważajmy elementy ciągu od najmniejszych. Załóżmy, że aktualnie rozważany element x łączy rozważone już przedziały A i B, które są ograniczone od lewej i prawej przez jeszcze nierozważone elementy l i p – ciekawy przedział jest postaci $lAxBp$.
Aby poruszać się w przedziale AxB nie opłaca się wychodzić poza przedział $lAxBp$.
Aby obliczyć wspomniane wartości dla przedziału AxB wystarczą nam obliczone już wartości dla przedziałów A oraz B.
Korzystając ze wspomnianych wartości możemy również zaktualizować w czasie stałym sumę odległości od wierzchołków w przedziale AxB – wystarczy nam do tego tablica, na której na koniec policzymy sumy prefiksowe.

Finalne podejście

Rozważajmy elementy ciągu od najmniejszych. Załóżmy, że aktualnie rozważany element x łączy rozważone już przedziały A i B, które są ograniczone od lewej i prawej przez jeszcze nierozważone elementy l i p – ciekawy przedział jest postaci $lAxBp$.
Aby poruszać się w przedziale AxB nie opłaca się wychodzić poza przedział $lAxBp$.
Aby obliczyć wspomniane wartości dla przedziału AxB wystarczą nam obliczone już wartości dla przedziałów A oraz B.
Korzystając ze wspomnianych wartości możemy również zaktualizować w czasie stałym sumę odległości od wierzchołków w przedziale AxB – wystarczy nam do tego tablica, na której na koniec policzymy sumy prefiksowe.
Łączenie przedziałów możemy wykonać za pomącą algorytmu find&union lub kolejek dwustronnych.

Uproszczenie

Na początku założyliśmy, że cały ciąg jest ograniczony dużymi wartościami do których możemy skakać.

Uproszczenie

Na początku założyliśmy, że cały ciąg jest ograniczony dużymi wartościami do których możemy skakać.
Dla prefiksów i sufiksów całego ciągu łatwo uwzględnić, że nie jest to dozwolony ruch.

Liczba Zes-Polona

Najszybsze rozwiązanie: **Antoni Buraczewski (4:40)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Treść zadania

Jeśli $n = (d_k \ldots d_1 d_0)_b$, to:

$$S_b(n) = (d_0 + d_1 + \ldots + d_k) \bmod b$$

Treść zadania

Jeśli $n = (d_k \dots d_1 d_0)_b$, to:

$$S_b(n) = (d_0 + d_1 + \ldots + d_k) \bmod b$$

$$ZP(n) = \begin{cases} 1 & \text{jeśli } S_2(n) = S_3(n) = 0 \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

Treść zadania

Jeśli $n = (d_k \dots d_1 d_0)_b$, to:

$$S_b(n) = (d_0 + d_1 + \ldots + d_k) \bmod b$$

$$ZP(n) = \begin{cases} 1 & \text{jeśli } S_2(n) = S_3(n) = 0 \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

$$P(N) = \sum_{n=0}^{N-1} ZP(n)$$

Zadanie polega na obliczeniu $P(N+1) - 1$ dla wielu (q) różnych $N <= 3 \cdot 10^{10}$.

Etap 1: obliczmy dla małych N

- Wybierzmy A, B (później). Osobno rozważymy najmłodsze A cyfr w podstawie 2 i B cyfr w podstawie 3.

Etap 1: obliczmy dla małych N

- Wybierzmy A, B (później). Osobno rozważymy najmłodsze A cyfr w podstawie 2 i B cyfr w podstawie 3.
- Obliczamy $P(N)$ dla wszystkich $N \leq 2^{A+1} \cdot 3^{B+1}$.
- Reprezentację liczby w systemie 2 i 3 możemy zwiększać o 1 w czasie zamortyzowanym $O(1)$.
- Razem etap 1: czas i pamięć $\mathcal{O}(2^A 3^B)$.

Etap 2: podział na bloki

- Podzielmy przedział $[0, N_{\max}]$ na bloki.
- W każdym bloku ostatnie A cyfr w podstawie 2 i ostatnie B cyfr w podstawie 3 nie zmienia się.
- Zapytania N tylko na końcach bloków.

Etap 2: podział na bloki

- Podzielmy przedział $[0, N_{\max}]$ na bloki.
- W każdym bloku ostatnie A cyfr w podstawie 2 i ostatnie B cyfr w podstawie 3 nie zmienia się.
- Zapytania N tylko na końcach bloków.

Liczba bloków (= liczba końców bloków):

$$\mathcal{O}\left(\frac{N_{\max}}{2^A} + \frac{N_{\max}}{3^B} + q\right)$$

Przetwarzanie kolejnego bloku

Mamy blok $[N_1, N_2)$. Chcemy znaleźć podobny blok $[a, b)$ w już obliczonej tablicy:

- $N_2 - N_1 = b - a$
- $b < 2^{A+1}3^{B+1}$
- $S_2(N_1 + i) = S_2(a + i)$
- $S_3(N_1 + i) = S_3(b + i)$

Wtedy:

- $ZP(N_1 + i) = ZP(a + i)$
- $P(N_2) = P(N_1) + (P(b) - P(a))$

Przetwarzanie kolejnego bloku

Dla bloku $[N_1, N_2)$ chcemy znaleźć takie a, że:

- $a < 2^{A+1}3^{B+1}$
- $a \equiv N_1 \pmod{2^A}$
- $a \equiv N_1 \pmod{3^B}$
- $S_2(a) = S_2(N_1)$
- $S_3(a) = S_3(N_1)$

Spróbujmy $a = (N_1 \bmod 2^A 3^B) + i2^A 3^B$ dla $i = 0, \ldots, 5$. Któreś z nich jest dobre.

Czas działania

Każdy blok przetwarzamy w czasie stałym. Cały czas:

$$\mathcal{O}\left(2^A 3^B + \frac{N_{\max}}{2^A} + \frac{N_{\max}}{3^B} + q\right)$$

Czas działania

Każdy blok przetwarzamy w czasie stałym. Cały czas:

$$\mathcal{O}\left(2^A 3^B + \frac{N_{\max}}{2^A} + \frac{N_{\max}}{3^B} + q\right)$$

Weźmy $2^A = \Theta(N_{\max}^{1/3})$ i $3^B = \Theta(N_{\max}^{1/3})$.
Na przykład dla $N_{\max} = 3 \cdot 10^{10}$: $A = 11$, $B = 7$.
Czas działania:

$$\mathcal{O}\left(N_{\max}^{2/3} + q\right)$$

Kliki

Najszybsze rozwiązanie: **Rafał Mańczyk (2:58)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste
- w kazdym momencie chcemy znać liczbę klik

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste
- w kazdym momencie chcemy znać liczbę klik

Obserwacje

- przecięcie dwóch ścieżek jest ścieżką

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste
- w kazdym momencie chcemy znać liczbę klik

Obserwacje

- przecięcie dwóch ścieżek jest ścieżką
- przecięcie wielu ścieżek jest ścieżką

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste
- w kazdym momencie chcemy znać liczbę klik

Obserwacje

- przecięcie dwóch ścieżek jest ścieżką
- przecięcie wielu ścieżek jest ścieżką
- każdej klice odpowiada jakaś ścieżka

Kliki

Treść

- dynamiczny multizbiór ścieżek drzewowych
- ścieżki połączone krawędzią $\Leftrightarrow$ ich przecięcie jest niepuste
- w kazdym momencie chcemy znać liczbę klik

Obserwacje

- przecięcie dwóch ścieżek jest ścieżką
- przecięcie wielu ścieżek jest ścieżką
- każdej klice odpowiada jakaś ścieżka

Definicja

Węzeł v z $\mathcal{T}$ należy do kliki $\Leftrightarrow$ należy do przecięcia ścieżek składających się na klikę

Kliki

2 liczniki dla każdego węzła v z $\mathcal{T}$:

1. do ilu klik on należy
2. do ilu klik należy on i jego ojciec

2 liczniki dla każdego węzła v z $\mathcal{T}$:

1. do ilu klik on należy
2. do ilu klik należy on i jego ojciec

Obserwacja

Różnica licznikow – kliki dla ktorych dany węzeł jest lca

Kliki

2 liczniki dla każdego węzła v z $\mathcal{T}$:

1. do ilu klik on należy
2. do ilu klik należy on i jego ojciec

Obserwacja

Różnica licznikow – kliki dla ktorych dany węzeł jest lca

Wniosek

Suma różnic w $\mathcal{T}$ – każda klika zliczona dokładnie raz

Aktualizacje

- oba liczniki – na początku wszędzie 1

Aktualizacje

- oba liczniki – na początku wszędzie 1
- pierwszy licznik $* = (/ =)$ 2 na path(a, b)

Aktualizacje

- oba liczniki – na początku wszędzie 1
- pierwszy licznik $*= (/=)$ 2 na path(a, b)
- drugi licznik $*= (/=)$ 2 na path$(a, b) \setminus \text{lca}(a, b)$

Aktualizacje

- oba liczniki – na początku wszędzie 1
- pierwszy licznik $*= (/=)$ 2 na path(a, b)
- drugi licznik $*= (/=)$ 2 na path$(a, b) \setminus \text{lca}(a, b)$

HLD + Drzewo Przedziałowe = zapytanie w $\mathcal{O}(\log^2 n)$

Kliki

Aktualizacje

- oba liczniki – na początku wszędzie 1
- pierwszy licznik $* = (/ =)$ 2 na path(a, b)
- drugi licznik $* = (/ =)$ 2 na path$(a, b) \setminus$ lca(a, b)

HLD + Drzewo Przedziałowe = zapytanie w $\mathcal{O}(\log^2 n)$

Wynik

Rozwiązanie w $\mathcal{O}(q \log^2 n)$

Najemnicy

Najszybsze rozwiązanie: **Kamil Dębowski (4:29)**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Najemnicy

Is this geometry?

Zinterpretujmy najemników i przedmioty jako wektory w pierwszej ćwiartce układu współrzędnych – wygrana z potworem oznacza bycie w zadanej półpłaszczyźnie.

Najemnicy

Is this geometry?

Zinterpretujmy najemników i przedmioty jako wektory w pierwszej ćwiartce układu współrzędnych – wygrana z potworem oznacza bycie w zadanej półpłaszczyźnie.

Is this otoczki?

Aby stwierdzić czy potwora da się w ogóle pokonać wystarczy rozważać jedynie górnoprawą otoczkę wypukłą możliwych statystyk najemników, którzy mogą do niego dotrzeć.

Zorganizowane podejście

Zbudujmy nad ciągiem drzewo przedziałowe.

Zorganizowane podejście

Zbudujmy nad ciągiem drzewo przedziałowe.
W każdym przedziale bazowym obliczajmy otoczkę możliwych bonusów jakie możemy uzyskać odwiedzając sklepy w tym przedziale.

Zorganizowane podejście

Zbudujmy nad ciągiem drzewo przedziałowe.
W każdym przedziale bazowym obliczajmy otoczkę możliwych bonusów jakie możemy uzyskać odwiedzając sklepy w tym przedziale.
Otoczka dla przedziału bazowego jego sumą Minkowskiego otoczek dla jego podprzedziałów bazowych – umiemy je zmergować liniowo.

Zorganizowane podejście część druga

Dla każdego przedziału bazowego policzmy również otoczkę możliwych statystyk najemników zaczynających w nim w momencie wyjścia z niego.

Zorganizowane podejście część druga

Dla każdego przedziału bazowego policzmy również otoczkę możliwych statystyk najemników zaczynających w nim w momencie wyjścia z niego.
Taki najemnik może zacząć w prawym podprzedziale

Zorganizowane podejście część druga

Dla każdego przedziału bazowego policzmy również otoczkę możliwych statystyk najemników zaczynających w nim w momencie wyjścia z niego.
Taki najemnik może zacząć w prawym podprzedziale lub w lewym i wzmocnić się o wszystkie sklepy z prawego podprzedziału

Zorganizowane podejście część druga

Dla każdego przedziału bazowego policzmy również otoczkę możliwych statystyk najemników zaczynających w nim w momencie wyjścia z niego.
Taki najemnik może zacząć w prawym podprzedziale lub w lewym i wzmocnić się o wszystkie sklepy z prawego podprzedziału (znów suma Minkowskiego).

Zorganizowane podejście część druga

Dla każdego przedziału bazowego policzmy również otoczkę możliwych statystyk najemników zaczynających w nim w momencie wyjścia z niego.
Taki najemnik może zacząć w prawym podprzedziale lub w lewym i wzmocnić się o wszystkie sklepy z prawego podprzedziału (znów suma Minkowskiego).
Policzyć otoczkę zbioru punktów również umiemy w czasie liniowym.

Scenariusz ataku

Gdy potwór atakuje podzielmy prefiks przed nim na przedziały bazowe i rozważajmy je od prawej.

Scenariusz ataku

Gdy potwór atakuje podzielmy prefiks przed nim na przedziały bazowe i rozważajmy je od prawej.
Aby stwierdzić, czy w przedziale bazowym istnieje ktoś kto mógłby pokonać potwora możemy wyszukać binarnie po otoczce wypukłej.

Scenariusz ataku

Gdy potwór atakuje podzielmy prefiks przed nim na przedziały bazowe i rozważajmy je od prawej.
Aby stwierdzić, czy w przedziale bazowym istnieje ktoś kto mógłby pokonać potwora możemy wyszukać binarnie po otoczce wypukłej.
Za każdym razem przesuwając się w drzewie w lewo należy rozważyć mijane po prawej sklepy

Scenariusz ataku

Gdy potwór atakuje podzielmy prefiks przed nim na przedziały bazowe i rozważajmy je od prawej.
Aby stwierdzić, czy w przedziale bazowym istnieje ktoś kto mógłby pokonać potwora możemy wyszukać binarnie po otoczce wypukłej.
Za każdym razem przesuwając się w drzewie w lewo należy rozważyć mijane po prawej sklepy – ich maksymalną pomoc przy pokonaniu danego potwora (czyli możliwe zmniejszenie parametru c) również możemy poznać dzięki wyszukiwaniu binarnym.

Optymalizacja

Opisane rozwiązanie dla każdego zapytania dla $\mathcal{O}(\log(n))$ przedziałów bazowych używa wyszukiwania binarnego – da się szybciej.

Optymalizacja

Opisane rozwiązanie dla każdego zapytania dla $\mathcal{O}(\log(n))$ przedziałów bazowych używa wyszukiwania binarnego – da się szybciej.
Rozważając potwory w kolejności kątowej szukany punkt we wszystkich otoczkach w drzewie będzie przesuwał się w jedną stronę

Optymalizacja

Opisane rozwiązanie dla każdego zapytania dla $\mathcal{O}(\log(n))$ przedziałów bazowych używa wyszukiwania binarnego – da się szybciej.
Rozważając potwory w kolejności kątowej szukany punkt we wszystkich otoczkach w drzewie będzie przesuwał się w jedną stronę – będziemy mogli użyć gąsieniczki lub po prostu zrzucać nieoptymalne punkty z otoczek.

Optymalizacja

Opisane rozwiązanie dla każdego zapytania dla $\mathcal{O}(\log(n))$ przedziałów bazowych używa wyszukiwania binarnego – da się szybciej.
Rozważając potwory w kolejności kątowej szukany punkt we wszystkich otoczkach w drzewie będzie przesuwał się w jedną stronę – będziemy mogli użyć gąsieniczki lub po prostu zrzucać nieoptymalne punkty z otoczek.
Uwzględniona optymalizacja pozwoli przemnożyć rozmiar wejścia tylko przez jeden logarytm – wysokość drzewa binarnego oraz potrzebę posortowania kątowo potworów i przedmiotów w sklepach.

Odwrotne zadanie

Najszybsze rozwiązanie: **brak**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Odwrotne zadanie

Nieodwrotne zadanie

Dane jest drzewo o n wierzchołkach. Policzyć liczbę kolorowań n kolorami (modulo M) tak, żeby wierzchołki o odległości ≤ 2 miały różne kolory.

Nieodwrotne zadanie

Dane jest drzewo o n wierzchołkach. Policzyć liczbę kolorowań n kolorami (modulo M) tak, żeby wierzchołki o odległości ≤ 2 miały różne kolory.

- W korzeniu mamy n możliwości.
- Dla d_1 dzieci korzenia mamy $(n-1)(n-2)\ldots(n-d_1)$ możliwości.

Nieodwrotne zadanie

Dane jest drzewo o n wierzchołkach. Policzyć liczbę kolorowań n kolorami (modulo M) tak, żeby wierzchołki o odległości ≤ 2 miały różne kolory.

- W korzeniu mamy n możliwości.
- Dla d_1 dzieci korzenia mamy $(n-1)(n-2)\ldots(n-d_1)$ możliwości.
- Dla $d_i - 1$ dzieci innych wierzchołków mamy $(n-2)\ldots(n-d_i)$ możliwości.

Nieodwrotne zadanie

Dane jest drzewo o n wierzchołkach. Policzyć liczbę kolorowań n kolorami (modulo M) tak, żeby wierzchołki o odległości ≤ 2 miały różne kolory.

- W korzeniu mamy n możliwości.
- Dla d_1 dzieci korzenia mamy $(n-1)(n-2)\dots(n-d_1)$ możliwości.
- Dla $d_i - 1$ dzieci innych wierzchołków mamy $(n-2)\dots(n-d_i)$ możliwości.

Razem:

$$n(n-1)\prod_i (n-2)\dots(n-d_i) = n(n-1)\prod_i f(d_i)$$

Odwrotne zadanie

Treść odwrotnego zadania

Znaleźć najmniejsze n takie i zbiór stopni $d_1, d_2, \ldots, d_n$, aby:

- $\sum_i (d_i - 1) = n - 2$
- $n(n-1) \prod_i f(d_i) \equiv r \pmod{M}$

Gdy znajdziemy taki ciąg d_i, łatwo wygenerować drzewo.

Odwrotne zadanie

Jak szukać dla danego n

- Dzielimy zbiór możliwych stopni $\{2, 3, \ldots, n-1\}$ na dwie grupy, na przykład: $\{2, 3, 4, 5, 6\}$ i $\{7, \ldots, n-1\}$.
- Dla obu grup stopni liczymy **wszystkie** multi-zbiory stopni $\{d_1, d_2, \ldots, d_k\}$, $\sum_i (d_i - 1) \leq n - 2$.

Odwrotne zadanie

Jak szukać dla danego n

- Dzielimy zbiór możliwych stopni $\{2, 3, \ldots, n-1\}$ na dwie grupy, na przykład: $\{2, 3, 4, 5, 6\}$ i $\{7, \ldots, n-1\}$.
- Dla obu grup stopni liczymy **wszystkie** multi-zbiory stopni $\{d_1, d_2, \ldots, d_k\}$, $\sum_i (d_i - 1) \leq n - 2$.
- Dla każdego multizbioru liczymy $A = \sum_i (d_i - 1)$ oraz $B = \prod_i f(d_i)$.
- Łączymy każdy multizbiór z pierwszej grupy z odpowiednim multizbiorem z drugiej grupy tak, żeby $A_1 + A_2 = n - 2$, $n(n-1)B_1 \cdot B_2 \equiv r$.

Odwrotne zadanie

Szacunek złożoności

- Żeby trafić w losową liczbę z $[1, M)$, potrzebujemy próbować około $M \log M$ możliwości.
- Sprawdzenie wszystkich podziałów metodą brute force: $\mathcal{O}(M \log M)$.

Odwrotne zadanie

Szacunek złożoności

- Żeby trafić w losową liczbę z $[1, M)$, potrzebujemy próbować około $M \log M$ możliwości.
- Sprawdzenie wszystkich podziałów metodą brute force: $\mathcal{O}(M \log M)$.
- Dzięki dzieleniu stopni na dwie grupy jest trochę szybciej.
- Największy przypadek ma $n = 125$.
- Czas działania najlepiej sprawdzić empirycznie dla największego drzewa.

Odwrotne zadanie

Lekka optymalizacja

- Lewa grupa multizbiorów $\sqrt{q}$ razy mniejsza, prawa $\sqrt{q}$ razy większa.

Odwrotne zadanie

Lekka optymalizacja

- Lewa grupa multizbiorów $\sqrt{q}$ razy mniejsza, prawa $\sqrt{q}$ razy większa.
- Budujemy strukturę $\sqrt{q}$ razy dłużej, ale zapytania są $\sqrt{q}$ razy szybsze.
- Dla q zapytań: algorytm $\sqrt{q}$ razy szybszy.

Kolonizacja

Najszybsze rozwiązanie: **brak**

GOSPODARZ

Digital Technology Poland

ORGANIZATORZY

UNIWERSYTET WARSZAWSKI

FRI

SPONSORZY

HUAWEI

RTB HOUSE

Kolonizacja

Proces kolonizacji

Drzewo T możemy skolonizować z pomocą k kolonizatorów, jeżeli możemy umieścić ich w jakimś wierzchołku, a następnie:

- Przesuwać w każdej turze jednego kolonizatora na sąsiedni wierzchołek, kolorując ten wierzchołek.
- Nie dopuścić do sytuacji, w której pokolorowany wierzchołek niezawierający aktualnie żadnego kolonizatora sąsiaduje z niepokolorowanym wierzchołkiem.
- Pokolorować w ten sposób wszystkie wierzchołki.

Treść zadania

Dla danego n, oblicz dla każdego $k \in [1, n]$ ile jest nieetykietowanych drzew, które wymagają dokładnie k kolonizatorów (to znaczy da się je skolonizować z pomocą k kolonizatorów, ale nie z pomocą $k - 1$ kolonizatorów).

Ile potrzeba kolonizatorów?

Ukorzeńmy drzewo i załóżmy, że kolonizatorzy zostają początkowo umieszczeni w korzeniu. Mogą teraz kolonizować poddrzewa jedno po drugim, za każdym razem zostawiając jednego z nich w korzeniu. Wyjątkiem jest ostatnie poddrzewo, które skolonizują – do niego mogą przejść wszyscy kolonizatorzy.

Minimalną liczbę kolonizatorów możemy w takim razie liczyć prostym programowaniem dynamicznym:

$$dp(leaf) = 1$$

$$dp(v) = \begin{cases} \max_c dp(c) & \text{kiedy max wartość się nie powtarza} \\ \max_c dp(c) + 1 & \text{w.p.p.} \end{cases}$$

Wystarczy je policzyć dla każdego korzenia i wziąć minimalną wartość.

$$dp(leaf) = 1$$

$$dp(v) = \begin{cases} \max_c dp(c) & \text{kiedy max wartość się nie powtarza} \\ \max_c dp(c) + 1 & \text{w.p.p.} \end{cases}$$

Dla drzewa $\mathcal{T}$ i wierzchołka v, zdefiniujmy D_v jako $dp(v)$ obliczone w drzewie $\mathcal{T}$ ukorzenionym w v.

Obserwacja 1

$$\max_{a,b \in \mathcal{T}} |D_a - D_b| \leq 1$$

Obserwacja 2

$$\max_{v \in \mathcal{T}} D_v = \mathcal{O}(\log n)$$

Zliczanie drzew nieetykietowanych

Zauważmy, że każde drzewo ma jednoznacznie wyznaczony centroid (lub dwa). Zliczmy w takim razie liczbę **ukorzenionych** drzew nieetykietowanych, o rozmiarach poddrzew niewiększych niż $\frac{n}{2}$. Oznaczmy ją przez $trees(n)$.

- Odpowiedzią jest $trees(n)$ jeżeli $2 \nmid n$ (zliczamy drzewa ukorzenione w centroidzie) i $trees(n) + \binom{trees(\frac{n}{2})+1}{2}$ jeżeli $2 \mid n$ (musimy jeszcze dodać przypadki z dwoma centroidami).
- Ukorzenione drzewa nieetykietowane o ograniczonym rozmiarze poddrzew możemy zliczyć za pomocą programowania dynamicznego w czasie $\mathcal{O}(n^2 \log n)$.

Zliczanie ukorzenionych drzew o ustalonym D_{root}

Poprzednie programowanie dynamiczne możemy łatwo zmodyfikować w taki sposób, żeby zliczało drzewa z ustalonym D_{root} oraz ustaloną liczbą dzieci korzenia o maksymalnej wartości D_c (musimy rozróżniać tylko 1, 2, 3 lub więcej takich dzieci).

- Oznaczmy je przez $trees(n, d, cnt)$, gdzie $cnt = 3$ oznacza 3 lub więcej dzieci.
- Możemy je policzyć w czasie $\mathcal{O}(n^2 \log^3 n)$ lub $\mathcal{O}(n^2 \log^2 n)$ (korzystając z pomocniczego dp).

$trees(n, d, cnt)$ zliczają także drzewa, dla których wynik może się zmniejszyć o 1, jeżeli umieścimy kolonizatorów w innym wierzchołku.

Zliczanie ukorzenionych drzew, których nie warto ukorzeniać w korzeniu

Policzmy dodatkowe programowanie dynamiczne $bad(n, d, x)$, które jest równe liczbie drzew $\mathcal{T}$ (o takich samych ograniczeniach jak w poprzednim programowaniu dynamicznym), dla których $D_{root} = d + 1$, ale $\exists_{v \in \mathcal{T}} D_v = d$. Ponadto jeżeli dodamy do korzenia $\mathcal{T}$ drzewo o wyniku x lub mniejszym, to wciąż będzie to prawda, ale jeżeli $x + 1$, to już nie.

- Obliczamy to programowanie dynamiczne w czasie $O(n^2 \log^2 n)$ korzystając z wartości $trees(n, d, cnt)$ oraz sum prefiksowych.
- Korzystając z wartości *trees* oraz *bad* jesteśmy w stanie odtworzyć odpowiedź dla każdego $k \in [1, n]$.