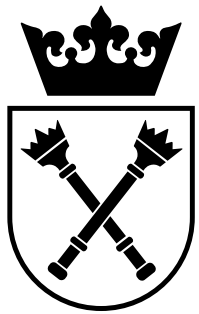
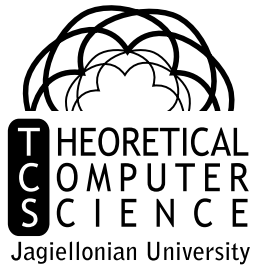
AMPPZ 2019: Prezentacja rozwiązań

Uniwersytet Jagielloński

27 października 2019





Statystyki

Zgłoszeń: 594
Zaakceptowanych: 208

Statystyki

Zgłoszeń: 594
Zaakceptowanych: 208

Pierwsze wbite zadanie: (C) Koło (z) Matematyków
uw7 [Partyka, Wiśniewski, Wasilewski]

Statystyki

Najbardziej zdeterminowani:

uw7 [Partyka, Wiśniewski, Wasilewski]
16 prób na zadaniu K

Zadanie A
Asymilacja

Zgłoszeń: 84
Zaakceptowanych: 36

Pierwsze rozwiązanie:
uj8 [Kapała, Tokarz, Deka]
00:21:23

Autor: Lech Duraj

Mamy n planet do opanowania, o wartościach $a_1, \ldots, a_n$ oraz k statków asymilacyjnych. W jednym ruchu możemy dokonać inwazji na planetę o wartości m, lądując na niej $s \geq m$ statkami. Planeta robi się wtedy opanowana, a jej nowa wartość to $m + s$. Z takiej planety można w przyszłości (ale tylko raz) podnieść $m + s$ statków. Ile co najmniej razy trzeba podnieść, by podbić wszystko?

Mamy n planet do opanowania, o wartościach $a_1, \ldots, a_n$ oraz k statków asymilacyjnych. W jednym ruchu możemy dokonać inwazji na planetę o wartości m, lądując na niej $s \geq m$ statkami. Planeta robi się wtedy opanowana, a jej nowa wartość to $m + s$. Z takiej planety można w przyszłości (ale tylko raz) podnieść $m + s$ statków. Ile co najmniej razy trzeba podnieść, by podbić wszystko?

Zadanie inspirowane mechaniką migracji barbarzyńców w grze *Imperator: Rome*.

Mamy n planet do opanowania, o wartościach $a_1, \ldots, a_n$ oraz k statków asymilacyjnych. W jednym ruchu możemy dokonać inwazji na planetę o wartości m, lądując na niej $s \geq m$ statkami. Planeta robi się wtedy opanowana, a jej nowa wartość to $m + s$. Z takiej planety można w przyszłości (ale tylko raz) podnieść $m + s$ statków. Ile co najmniej razy trzeba podnieść, by podbić wszystko?

Zadanie inspirowane mechaniką migracji barbarzyńców w grze *Imperator: Rome*.



(dobrze zastosowana poniższa strategia = lawina barbarzyńców!)

Jeśli liczba statków na początku starczy na opanowanie wszystkich planet od razu, to odpowiedź jest oczywiście 0.

Jeśli liczba statków na początku starczy na opanowanie wszystkich planet od razu, to odpowiedź jest oczywiście 0.
Ale co, jeśli nie jest?

Jeśli liczba statków na początku starczy na opanowanie wszystkich planet od razu, to odpowiedź jest oczywiście 0.
Ale co, jeśli nie jest?

Na pewno jest jakaś planeta, którą będziemy mobilizować. Co więcej, na pewno jest planeta P, którą będziemy mobilizować jako pierwszą.

Jeśli liczba statków na początku starczy na opanowanie wszystkich planet od razu, to odpowiedź jest oczywiście 0.
Ale co, jeśli nie jest?

Na pewno jest jakaś planeta, którą będziemy mobilizować. Co więcej, na pewno jest planeta P, którą będziemy mobilizować jako pierwszą.

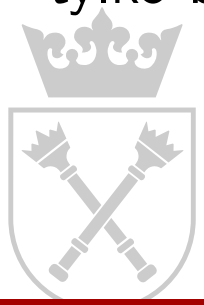
Ale w takim razie można *od razu* opanować P i *od razu* ją zmobilizować – dalej będzie tylko łatwiej.

Jeśli liczba statków na początku starczy na opanowanie wszystkich planet od razu, to odpowiedź jest oczywiście 0.
Ale co, jeśli nie jest?

Na pewno jest jakaś planeta, którą będziemy mobilizować. Co więcej, na pewno jest planeta P, którą będziemy mobilizować jako pierwszą.

Ale w takim razie można *od razu* opanować P i *od razu* ją zmobilizować – dalej będzie tylko łatwiej.

Czyli wiemy, że pierwszym naszym ruchem będzie opanowanie jakiejś planety i natychmiastowa mobilizacja. To teraz oczywiste jest, którą planetę wybrać – największą, na jaką możemy się porwać. Inny wybór tylko by nam utrudnił sprawę, bo mielibyśmy dalej mniej statków.



Pełny algorytm wygląda tak:

Q – zbiór jeszcze nie opanowanych planet
s – suma wartości planet w Q
k – liczba naszych statków
wynik – liczba mobilizacji, jakie wykonaliśmy.

- Jeżeli $k \geq s$, to opanuj wszystkie planety i zakończ algorytm.
- W przeciwnym razie, weź p – największą planetę z Q mniejszą lub równą k.
- $Q = Q - \{p\}$; $s = s - p$; $k = k + p$; $wynik = wynik + 1$.

Pełny algorytm wygląda tak:

Q – zbiór jeszcze nie opanowanych planet
s – suma wartości planet w Q
k – liczba naszych statków
wynik – liczba mobilizacji, jakie wykonaliśmy.

- Jeżeli $k \geq s$, to opanuj wszystkie planety i zakończ algorytm.
- W przeciwnym razie, weź p – największą planetę z Q mniejszą lub równą k.
- $Q = Q - \{p\}$; $s = s - p$; $k = k + p$; $wynik = wynik + 1$.

Najprościej implementować Q jako `multiset`, co daje algorytm $\mathcal{O}(n \log n)$
Ale można też użyć posortowanej tablicy – nieco dłuższy kod, nieco lepsza stała w złożoności.



```
multiset<int> s{10, 20, 30};
long long x = 123000123000LL;
auto it = s.lower_bound(x);
if(it != s.end()) {
    cout << *it;
}
```

(ten kod wypisze liczbę 10)

Zadanie B
Robaczek

Zgłoszeń: 38

Zaakceptowanych: 5

Pierwsze rozwiązanie:
uw1 [Kondraciuk, Skiba, Paluszek]
01:36:15*

Autor: Adam Polak

- Weźmy dowolny wierzchołek na najkrótszej ścieżce pomiędzy pozycją początkową i końcową Robaczka. Nazwijmy ten wierzchołek v.

- Weźmy dowolny wierzchołek na najkrótszej ścieżce pomiędzy pozycją początkową i końcową Robaczka. Nazwijmy ten wierzchołek v.
- Zadanie teraz jest następujące: mamy pozycję Robaczka i chcemy dotrzeć końcem Robaczka do wierzchołka v. Jeśli mamy maszynkę, która potrafi rozwiązywać takie zadanie, to możemy odpalić ją dla pozycji początkowej i końcowej robaczka – potem sklejamy wyniki.

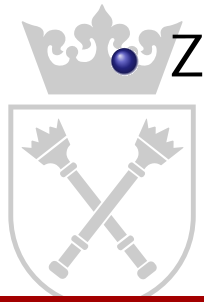
- Weźmy dowolny wierzchołek na najkrótszej ścieżce pomiędzy pozycją początkową i końcową Robaczka. Nazwijmy ten wierzchołek v.
- Zadanie teraz jest następujące: mamy pozycję Robaczka i chcemy dotrzeć końcem Robaczka do wierzchołka v. Jeśli mamy maszynkę, która potrafi rozwiązywać takie zadanie, to możemy odpalić ją dla pozycji początkowej i końcowej robaczka – potem sklejamy wyniki.
- Załóżmy, że robaczek leży na ścieżce (x, y). Wtedy tak naprawdę chcemy, żeby robaczek dotknął jednym ze swoich końców wierzchołka $z := lca(x, y)$.

- Niech x oznacza jeden z końców Robaczka. Wtedy spróbujmy pójść do najgłębszego wierzchołka w poddrzewie x. Po tej operacji mamy dwie możliwości:
  1. Drugi koniec Robaczka jest w tym samym poddrzewie wierzchołka c, co i wierzchołek v. Wtedy wygraliśmy.
  2. Robaczek leży na ścieżce (x', y'). Wtedy powtarzamy dokładnie to, co wcześniej, tylko z innym końcem Robaczka.

- Niech x oznacza jeden z końców Robaczka. Wtedy spróbujmy pójść do najgłębszego wierzchołka w poddrzewie x. Po tej operacji mamy dwie możliwości:
  1. Drugi koniec Robaczka jest w tym samym poddrzewie wierzchołka c, co i wierzchołek v. Wtedy wygraliśmy.
  2. Robaczek leży na ścieżce (x', y'). Wtedy powtarzamy dokładnie to, co wcześniej, tylko z innym końcem Robaczka.
- Jeśli w jakimś momencie Robaczek przestał "podnosić się" jednym ze swoich końców – przerywamy i mówimy "NIE".

- Niech x oznacza jeden z końców Robaczka. Wtedy spróbujmy pójść do najgłębszego wierzchołka w poddrzewie x. Po tej operacji mamy dwie możliwości:
  1. Drugi koniec Robaczka jest w tym samym poddrzewie wierzchołka c, co i wierzchołek v. Wtedy wygraliśmy.
  2. Robaczek leży na ścieżce (x', y'). Wtedy powtarzamy dokładnie to, co wcześniej, tylko z innym końcem Robaczka.
- Jeśli w jakimś momencie Robaczek przestał "podnosić się" jednym ze swoich końców – przerywamy i mówimy "NIE".
- Zauważmy, że za każdym razem, kiedy schodzimy Robaczkiem do najgłębszego poddrzewa – odwiedzamy nowy wierzchołek. Zatem można zaimplementować to rozwiązanie tak, żeby Robaczek wykonał nie więcej niż n ruchów.

- Niech x oznacza jeden z końców Robaczka. Wtedy spróbujmy pójść do najgłębszego wierzchołka w poddrzewie x. Po tej operacji mamy dwie możliwości:
  1. Drugi koniec Robaczka jest w tym samym poddrzewie wierzchołka c, co i wierzchołek v. Wtedy wygraliśmy.
  2. Robaczek leży na ścieżce (x', y'). Wtedy powtarzamy dokładnie to, co wcześniej, tylko z innym końcem Robaczka.
- Jeśli w jakimś momencie Robaczek przestał "podnosić się" jednym ze swoich końców – przerywamy i mówimy "NIE".
- Zauważmy, że za każdym razem, kiedy schodzimy Robaczkiem do najgłębszego poddrzewa – odwiedzamy nowy wierzchołek. Zatem można zaimplementować to rozwiązanie tak, żeby Robaczek wykonał nie więcej niż n ruchów.
- Złożoność czasowa: $\mathcal{O}(n)$.

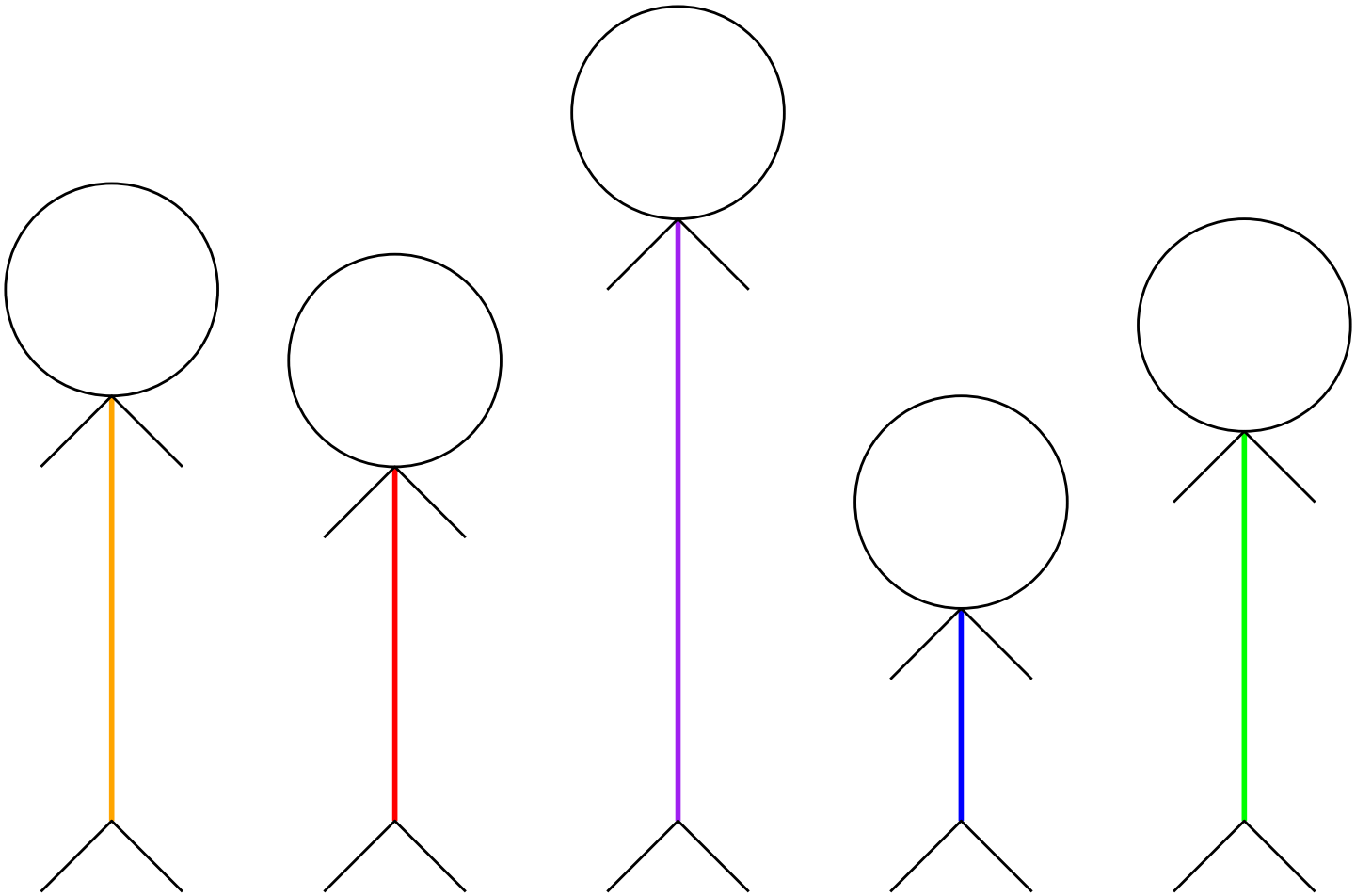


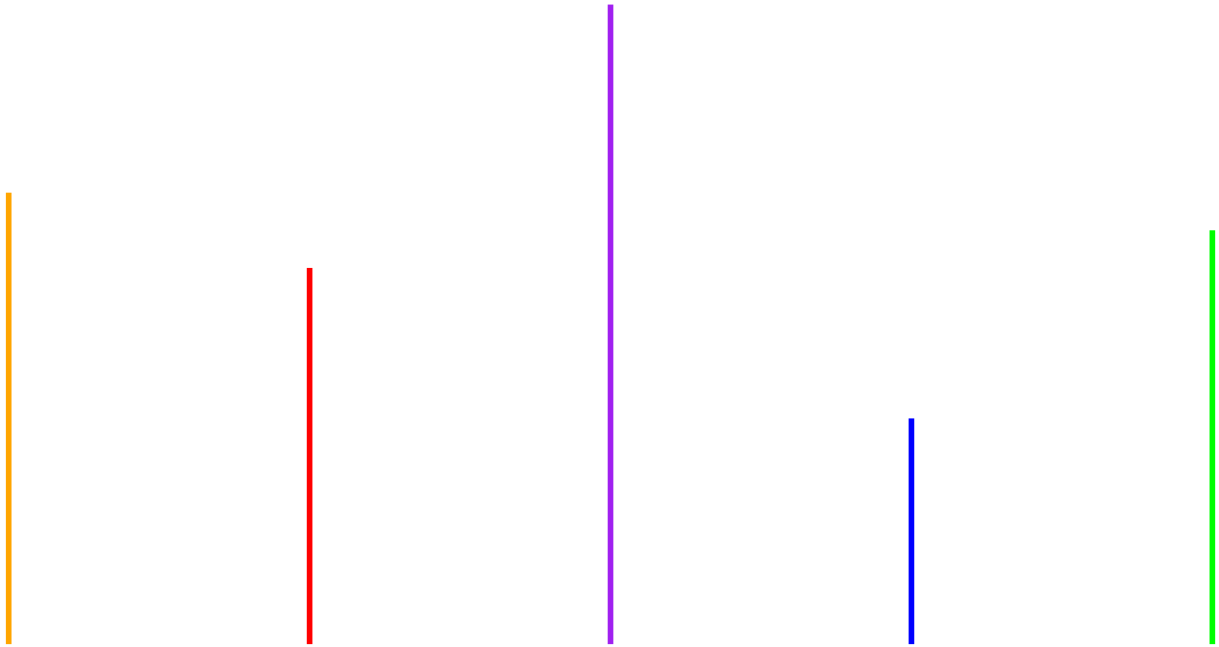
Zadanie C
Koło (z) Matematyków

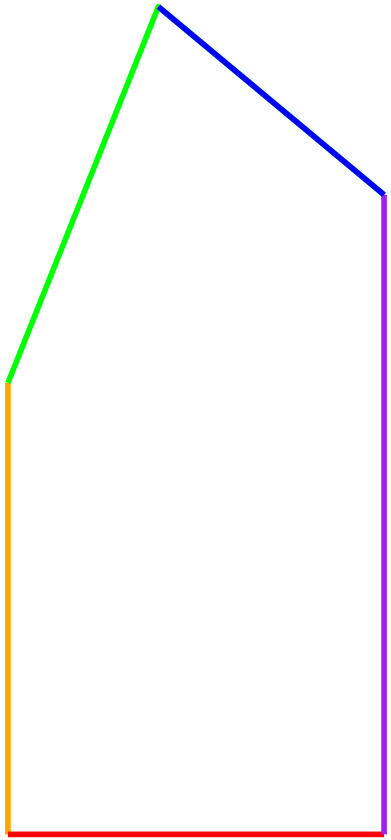
Zgłoszeń: 64
Zaakceptowanych: 42

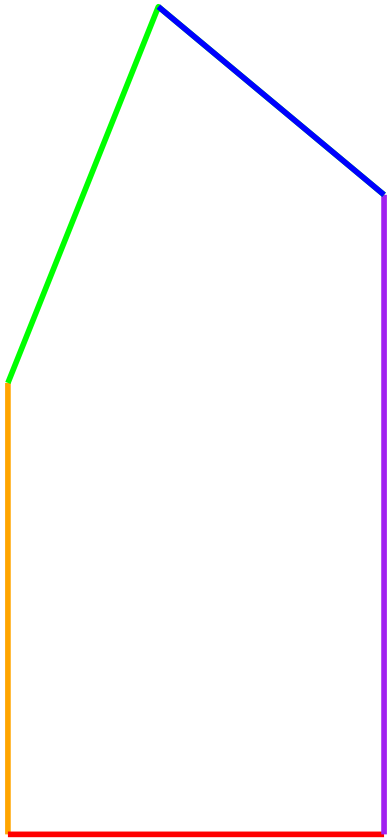
Pierwsze rozwiązanie:
uw7 [Partyka, Wiśniewski, Wasilewski]
00:07:36

Autor: Vladyslav Hlembotskyi



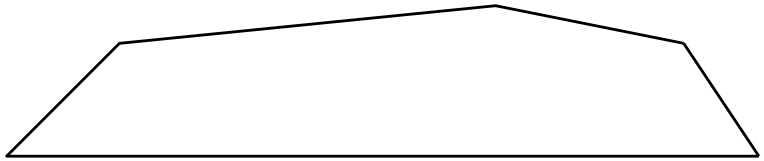




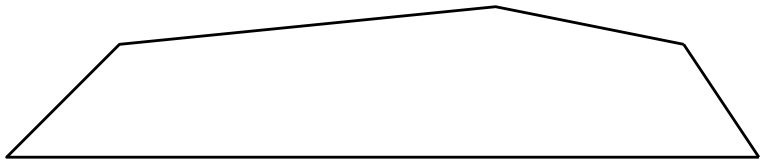



KOŁO · MATEMATYKÓW · STUDENTÓW · UJ · IM · PROF · STANISŁAWA · ZAREMBY
ΑΕΙ · Ο · ΘΕΟΣ · ΓΕΩΜΕΤΡΕΙ

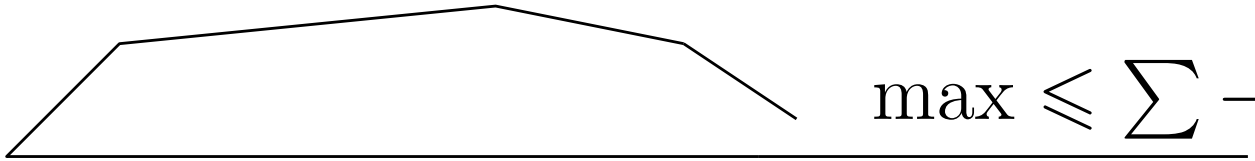
THEORETICAL
COMPUTER
SCIENCE
Jagiellonian University



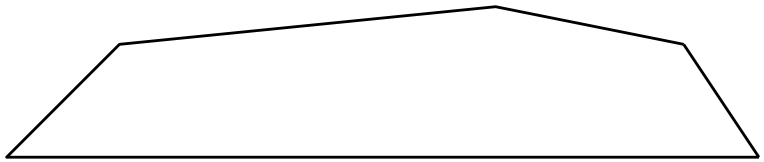
$$\max > \sum - \max \quad \checkmark$$



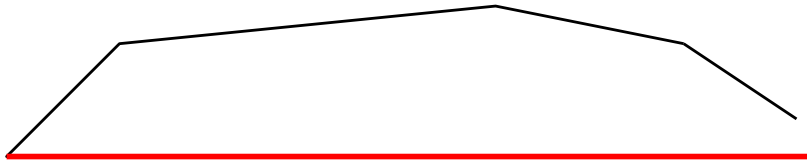
$$\max > \sum - \max \quad \checkmark$$



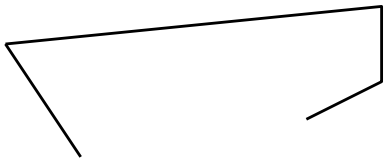
$$\max \leqslant \sum - \max \quad :($$

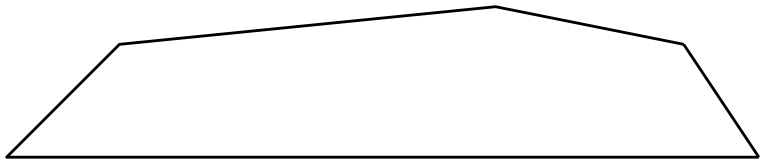


$$\max > \sum - \max \quad \checkmark$$

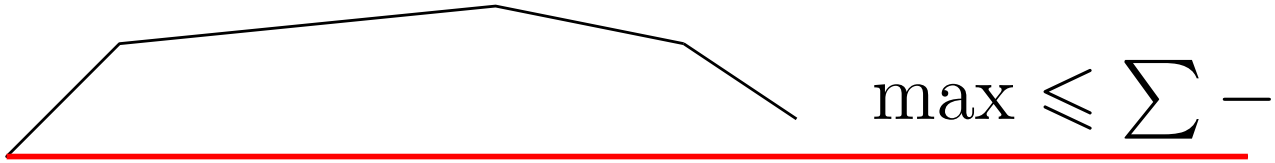


$$\max \leqslant \sum - \max \quad :($$

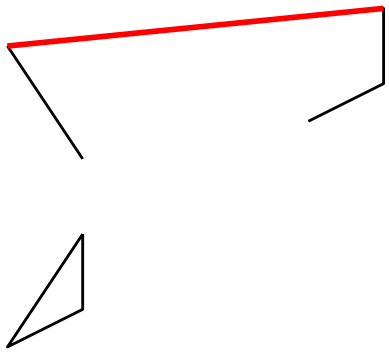


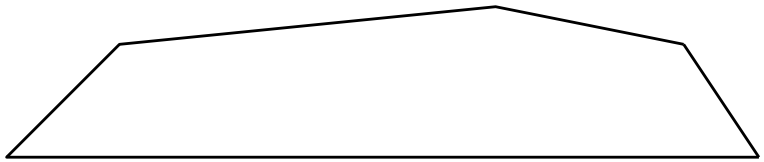


$$\max > \sum - \max \quad \checkmark$$

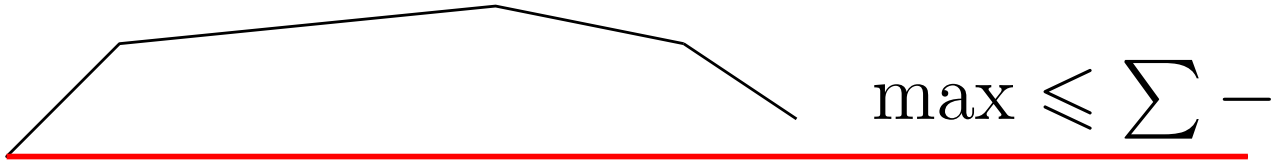


$$\max \leqslant \sum - \max \quad :($$

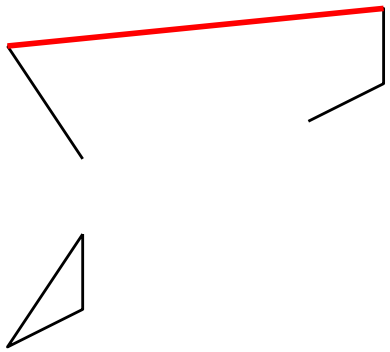




$$\max > \sum - \max \quad \checkmark$$



$$\max \leqslant \sum - \max \quad :($$



```
while max >= sum - max
    remove max
```

Zadanie D
Żaby

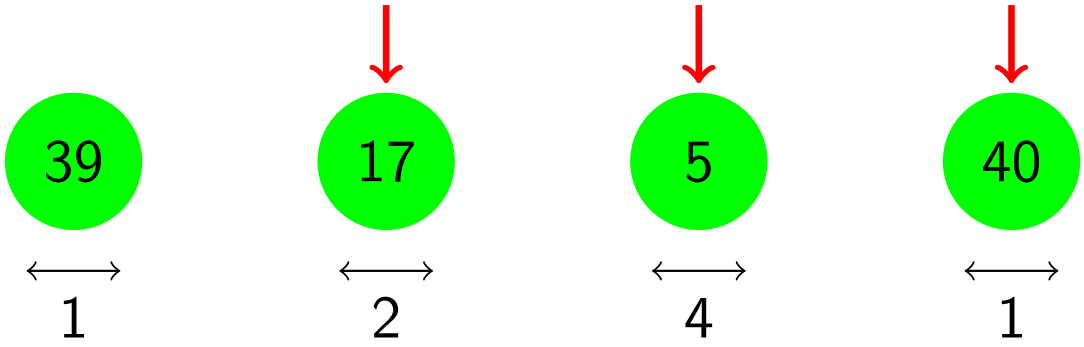
Zgłoszeń: 74
Zaakceptowanych: 35

Pierwsze rozwiązanie:
uj2 [Zub, Orap, Denha]
00:08:57

Autor: Kamil Dębowski

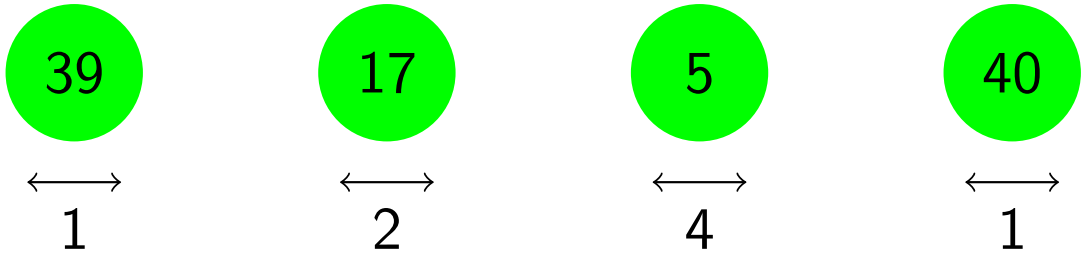
Jest n kamieni w rzędzie, na każdym mieszka jedna żaba. Każda żaba ma pewien poziom s_i oraz może skoczyć raz w lewo lub w prawo o co najwyżej r_i. Chcemy wybrać drużynę złożoną z trzech żab, dla której:

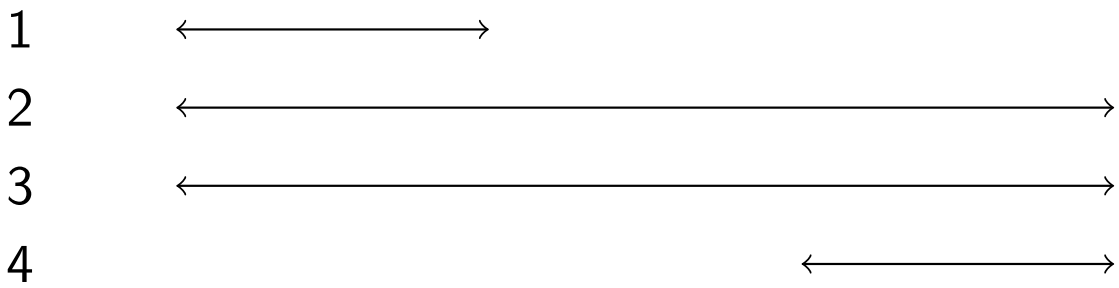
- cała drużyna może trenować na jednym kamieniu, tj. istnieje kamień, na który mogą skoczyć wszystkie trzy żaby,
- suma poziomów tych żab jest jak największa.



Dla każdej żaby możemy znaleźć jej **zasięg**, to jest znaleźć pierwszy i ostatni kamień, na który może skoczyć:

$$[\max(1, i - r_i), \min(n, i + r_i)]$$





Początki i końce zasięgów nazwiemy zdarzeniami i będziemy rozpatrywać od lewej do prawej (wcześniej je sortując):

- kiedy rozpoczyna się zasięg nowej żaby, dorzucamy jej poziom do struktury (np. multiseta),
- jeżeli w strukturze mamy co najmniej trzy żaby, liczymy sumę poziomów trzech najlepszych żab i aktualizujemy wynik,
- kiedy kończy się zasięg danej żaby, usuwamy jej poziom.

Warto zwrócić uwagę, że jeżeli kamień jest zarówno początkiem i końcem zasięgu dla niektórych żab, to najpierw trzeba rozważyć początki (dodawanie do struktury), a później końce (usuwanie ze struktury), tak aby te żaby mogły się spotkać na tym kamieniu.

Złożoność czasowa: $\mathcal{O}(n \log n)$.

Zadanie E
Drony

Zgłoszeń: 21
Zaakceptowanych: ??

Pierwsze rozwiązanie:
?

Autor: Krzysztof Maziarz

W przestrzeni trójwymiarowej znajduje się n dronów, początkowo w pozycjach $(x_i, y_i, 0)$ dla $i \in \{1, 2, ..., n\}$. Pomiędzy dronami jest m połączeń, które tworzą spójny graf planarny. Dla każdego z połączeń znana jest maksymalna długość – jeśli drony które łączy oddalą się bardziej niż ta odległość, połączenie się zrywa.

Otrzymujemy k manewrów; jeden manewr to zmiana trzeciej współrzędnej jednego z dronów. Podczas manewrów niektóre połączenia się zrywają.

Dla każdej z q par dronów (u_i, v_i) rozstrzygnąć, kiedy przestała istnieć ścieżka pomiędzy u_i oraz v_i (prowadząca po niezerwanych połączeniach).

Zauważmy najpierw, że aby odpowiedzieć na zapytania, wystarczy stwierdzić dla każdej krawędzi kiedy się zerwała.

Zauważmy najpierw, że aby odpowiedzieć na zapytania, wystarczy stwierdzić dla każdej krawędzi kiedy się zerwała.

- Możemy przewtorzyć krawędzie od najpóźniej do najwcześniej zerwanej ("odwrócić czas").

Zauważmy najpierw, że aby odpowiedzieć na zapytania, wystarczy stwierdzić dla każdej krawędzi kiedy się zerwała.

- Możemy przewtorzyć krawędzie od najpóźniej do najwcześniej zerwanej ("odwrócić czas").
- Nasze zapytania przyjmują postać: dla każdej pary (u_i, v_i), podaj kiedy wierzchołki u_i oraz v_i znalazły się w jednej spójnej składowej.

Zauważmy najpierw, że aby odpowiedzieć na zapytania, wystarczy stwierdzić dla każdej krawędzi kiedy się zerwała.

- Możemy przewtorzyć krawędzie od najpóźniej do najwcześniej zerwanej ("odwrócić czas").
- Nasze zapytania przyjmują postać: dla każdej pary (u_i, v_i), podaj kiedy wierzchołki u_i oraz v_i znalazły się w jednej spójnej składowej.
- Utrzymujemy obecny podział wierzchołków na składowe, oraz dla każdej z nich zbiór identyfikatorów zapytań o jednym końcu w tej składowej. Łącząc przerzucamy "mniejszy do większego".

Zauważmy najpierw, że aby odpowiedzieć na zapytania, wystarczy stwierdzić dla każdej krawędzi kiedy się zerwała.

- Możemy przewtorzyć krawędzie od najpóźniej do najwcześniej zerwanej ("odwrócić czas").
- Nasze zapytania przyjmują postać: dla każdej pary (u_i, v_i), podaj kiedy wierzchołki u_i oraz v_i znalazły się w jednej spójnej składowej.
- Utrzymujemy obecny podział wierzchołków na składowe, oraz dla każdej z nich zbiór identyfikatorów zapytań o jednym końcu w tej składowej. Łącząc przerzucamy "mniejszy do większego".
- Ta część rozwiązania działa w czasie $\mathcal{O}(m \log^2 m)$.

Pozostaje wyznaczyć dla każdej krawędzi kiedy się zerwała.

Pozostaje wyznaczyć dla każdej krawędzi kiedy się zerwała.

Dla uproszczenia zauważmy, że skoro pierwsze dwie współrzędne dronów pozostają stałe, to znając maksymalną długość na którą można rozciągnąć krawędź (u, v) możemy wyliczyć maksymalną różnicę trzecich współrzędnych wierzchołków u i v.

Pozostaje wyznaczyć dla każdej krawędzi kiedy się zerwała.

Dla uproszczenia zauważmy, że skoro pierwsze dwie współrzędne dronów pozostają stałe, to znając maksymalną długość na którą można rozciągnąć krawędź (u, v) możemy wyliczyć maksymalną różnicę trzecich współrzędnych wierzchołków u i v.

Chcemy zasymulować manewry, wykrywając zrywające się krawędzie. Niestety, naiwne zasymulowanie manewru na wierzchołku v zajęłoby czas $\mathcal{O}(deg(v))$.

Fakt: w grafie planarnym istnieje wierzchołek o stopniu co najwyżej 5. Wynika to na przykład z tego, że liczba krawędzi grafu planarnego jest mniejsza niż $3n$ (lub równoważnie ze wzoru Eulera).

Fakt: w grafie planarnym istnieje wierzchołek o stopniu co najwyżej 5. Wynika to na przykład z tego, że liczba krawędzi grafu planarnego jest mniejsza niż $3n$ (lub równoważnie ze wzoru Eulera).

Istnieje więc porządek wierzchołków σ, taki że każdy wierzchołek v ma tylko ≤ 5 krawędzi "w prawo".

Symulujemy manewry, ale gdy wykonujemy manewr na wierzchołku v, to przeglądamy tylko krawędzie z v w prawo.

Symulujemy manewry, ale gdy wykonujemy manewr na wierzchołku v, to przeglądamy tylko krawędzie z v w prawo.

Patrząc na krawędź (v, u), to może być tak, że zerwał ją właśnie wykonany manewr - ale ta sytuacja jest prosta. Musimy jeszcze uwzględnić możliwość, że zerwał ją pewien manewr wykonywany na u gdzieś pomiędzy poprzednim manewrem na v a obecnym.

Symulujemy manewry, ale gdy wykonujemy manewr na wierzchołku v, to przeglądamy tylko krawędzie z v w prawo.

Patrząc na krawędź (v, u), to może być tak, że zerwał ją właśnie wykonany manewr - ale ta sytuacja jest prosta. Musimy jeszcze uwzględnić możliwość, że zerwał ją pewien manewr wykonywany na u gdzieś pomiędzy poprzednim manewrem na v a obecnym.

Pomiędzy dwoma kolejnymi manewrami v jego trzecia współrzędna była stała. Żeby krawędź (v, u) się nie zerwała, trzecia współrzędna u musiała cały czas znajdować się w odpowiednim przedziale.

Chcemy wiedzieć, czy w pewnym przedziale czasu $[l, r]$, trzecia współrzędna u wyszła poza przedział $[h_l, h_r]$, i jeśli tak, to jaki był pierwszy moment kiedy to się stało.

Chcemy wiedzieć, czy w pewnym przedziale czasu $[l, r]$, trzecia współrzędna u wyszła poza przedział $[h_l, h_r]$, i jeśli tak, to jaki był pierwszy moment kiedy to się stało.

Dla każdego u, możemy rozważyć sekwencję wartości jego trzecej współrzędnej, i nad tą sekwencją zbudować drzewo przedziałowe z minimami i maksimami. Wtedy na powyższe pytanie możemy odpowiedzieć w czasie $\mathcal{O}(\log k)$ chodząc po drzewie przedziałowym.

Chcemy wiedzieć, czy w pewnym przedziale czasu $[l, r]$, trzecia współrzędna u wyszła poza przedział $[h_l, h_r]$, i jeśli tak, to jaki był pierwszy moment kiedy to się stało.

Dla każdego u, możemy rozważyć sekwencję wartości jego trzecej współrzędnej, i nad tą sekwencją zbudować drzewo przedziałowe z minimami i maksimami. Wtedy na powyższe pytanie możemy odpowiedzieć w czasie $\mathcal{O}(\log k)$ chodząc po drzewie przedziałowym.

Finalna złożoność: $\mathcal{O}((n+k)\log(n+k) + m\log^2 m)$.

Zadanie F
Fantastyczna kompresja

Zgłoszeń: 64
Zaakceptowanych: 20

Pierwsze rozwiązanie:
uw6 [Kowalska, Boguta, Kaszuba]
00:56:46*

Autor: Krzysztof Maziarz

Dana jest skompresowana permutacja w postaci sumy wszystkich jej spójnych podciągów o długości k ($k \leq 6$). Znaleźć wszystkie permutacje pasujące do te skompresowanej postaci.

Dla skompresowanej postaci z $k = 3$:

$$[8, 10, 12]$$

pasujące permutacje to:

$$[1, 2, 5, 3, 4] \text{ oraz } [2, 1, 5, 4, 3]$$

Zauważmy, że mając sąsiednie sumy, tj. s_j oraz s_{j+1}, możemy znaleźć różnicę między elementem a_i oraz a_{i+k}:

$$\begin{aligned} s_j &= a_i + a_{i+1} + \ldots + a_{i+k-1} \\ s_{j+1} &= a_{i+1} + a_{a+2} + \ldots + a_{i+k} \\ s_{j+1} - s_j &= a_{i+k} - a_i \end{aligned}$$

Możemy zatem znaleźć tę różnicę dla każdej pary elementów odległych o dokładnie k.

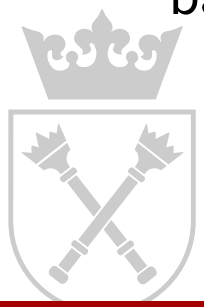
Mamy zatem k ciągów, dla których znamy wszystkie różnice pomiędzy elementami:

$$(a_0, a_k, a_{2k}, \ldots,)$$
$$(a_1, a_{k+1}, a_{2k+1}, \ldots,)$$
$$(a_2, a_{k+2}, a_{2k+2}, \ldots,)$$
$$\ldots$$

Wystarczy, że znajdziemy wartość jednego elementu w takim ciągu i znamy wartości wszystkich elementów.

- Wiemy jednak, że wszystkie te elementy muszą tworzyć permutację, zatem na pewno któryś z tych ciągów musi zawierać 1.
- Możemy sprawdzić każdy z tych k ciągów (k jest niewielkie) jako potencjalny ciąg zawierający 1.
- Wtedy poznajemy wszystkie inne wartości liczb z tego ciągu. Teraz możemy znaleźć kolejną najmniejszą liczbę i wybrać któryś z pozostałych $k-1$ ciągów, jako ten który zawiera tę liczbę.
- To prowadzi do rekurencji, która sprawdza wszystkie $k!$ możliwości. Warto zauważyć, że istnieje co najwyżej $k!$ permutacji, które pasują do zakodowanej postaci.
- Na końcu warto sprawdzić, czy uzyskana permutacja faktycznie pasuje, co możemy łatwo zrobić licząc wszystkie sumy (okienkiem, bądź brutalnie).

Złożoność czasowa: $\mathcal{O}(n \cdot k!)$.



Zadanie G
Antykwariat

Zgłoszeń: 54

Zaakceptowanych: 8

Pierwsze rozwiązanie:
uwr01 [Rzepecki, Górniak, Agrawal]
01:02:36*

Autor: Krzysztof Maziarz

Mamy dany ciąg n liczb $a_1, ..., a_n$ z przedziału $[1, M]$. Dostajemy dużo zapytań (L, R), dla każdego musimy podać liczbę spójnych przedziałów naszego ciągu które mają wszystkie wartości w przedziale $[L, R]$.

Mamy dany ciąg n liczb $a_1, ..., a_n$ z przedziału $[1, M]$. Dostajemy dużo zapytań (L, R), dla każdego musimy podać liczbę spójnych przedziałów naszego ciągu które mają wszystkie wartości w przedziale $[L, R]$.

W zadaniu mieliśmy jeszcze jeden bardzo ważny warunek: elementy ciągu a_i zostały wygenerowane *losowo* (niezależnie i jednostajnie z przedziału $[1, M]$). Dokładniej, użyliśmy do tego funkcji *lrand48* – ale do rozwiązania nie trzeba wiedzieć jak ta funkcja działa.

Spostrzeżenie: jeśli wylosujemy n liczb z $[1, M]$, to prawdopodobieństwo że ostatnia z ich jest największa nie przekracza $\frac{1}{n}$.

Spostrzeżenie: jeśli wylosujemy n liczb z $[1, M]$, to prawdopodobieństwo że ostatnia z ich jest największa nie przekracza $\frac{1}{n}$.

Z tego wynika, że jeśli czytamy n losowych liczb, i utrzymujemy obecne maksimum, to oczekiwana liczba zmian tego maksimum to

$$\sum_{i=1}^{n} \frac{1}{i} = \mathcal{O}(\log n)$$

Spostrzeżenie: jeśli wylosujemy n liczb z $[1, M]$, to prawdopodobieństwo że ostatnia z ich jest największa nie przekracza $\frac{1}{n}$.

Z tego wynika, że jeśli czytamy n losowych liczb, i utrzymujemy obecne maksimum, to oczekiwana liczba zmian tego maksimum to

$$\sum_{i=1}^{n} \frac{1}{i} = \mathcal{O}(\log n)$$

Oczywiście jeśli utrzymujemy parę (min, max), to oczekiwana liczba zmian któregokolwiek komponentu tej pary to również $\mathcal{O}(\log n)$.

Czytamy ciąg od prawej do lewej. Gdy jesteśmy na pozycji i, utrzymujemy stos zawierający takie pozycje j, na których zmienia się maksimum ciągu $a[i, ..., j]$. Analogicznie utrzymujemy drugi stos dla minimum.

Czytamy ciąg od prawej do lewej. Gdy jesteśmy na pozycji i, utrzymujemy stos zawierający takie pozycje j, na których zmienia się maksimum ciągu $a[i, ..., j]$. Analogicznie utrzymujemy drugi stos dla minimum.

Łatwo możemy aktualizować nasze stosy gdy przesuwamy i, a z obserwacji z poprzedniego slajdu ich rozmiar zawsze będzie $\mathcal{O}(\log n)$.

Czytamy ciąg od prawej do lewej. Gdy jesteśmy na pozycji i, utrzymujemy stos zawierający takie pozycje j, na których zmienia się maksimum ciągu $a[i, ..., j]$. Analogicznie utrzymujemy drugi stos dla minimum.

Łatwo możemy aktualizować nasze stosy gdy przesuwamy i, a z obserwacji z poprzedniego slajdu ich rozmiar zawsze będzie $\mathcal{O}(\log n)$.

Stosy pozwalają nam rozważyć wszystkie przedziały o lewym końcu w i, i pogrupować je po parze (min, max) na $\mathcal{O}(\log n)$ grup. Grupując tak dla każdego i otrzymujemy podział wszystkich $\mathcal{O}(n^2)$ podprzedziałów na $\mathcal{O}(n \log n)$ grup (każda grupa ma to samo minimum i maksimum).

Zamiatamy wartości od M do 1. Gdy rozważamy wartość x, to najpierw dodajemy do naszej struktury wszystkie grupy o $min = x$, a następnie odpowiadamy na zapytania o $L = x$.

Zamiatamy wartości od M do 1. Gdy rozważamy wartość x, to najpierw dodajemy do naszej struktury wszystkie grupy o $min = x$, a następnie odpowiadamy na zapytania o $L = x$.

Nasza struktura musi obsługiwać operację dodania grupy o zadanym max i pewnej liczności, oraz zapytania o sumę liczności grup o $max \leq x$. Do tego wystarczy proste drzewo przedziałowe.

Zamiatamy wartości od M do 1. Gdy rozważamy wartość x, to najpierw dodajemy do naszej struktury wszystkie grupy o $min = x$, a następnie odpowiadamy na zapytania o $L = x$.

Nasza struktura musi obsługiwać operację dodania grupy o zadanym max i pewnej liczności, oraz zapytania o sumę liczności grup o $max \leq x$. Do tego wystarczy proste drzewo przedziałowe.

Wykonujemy $\mathcal{O}(n \log n)$ operacji na drzewie, więc złożoność rozwiązania to $\mathcal{O}(n \log^2 n)$.

Zadanie H
Oscypki

Zgłoszeń: 81

Zaakceptowanych: 31

Pierwsze rozwiązanie:
uwr01 [Rzepecki, Górniak, Agrawal]
00:12:29*

Autor: Krzysztof Maziarz










4 7 2 3 6 4 2



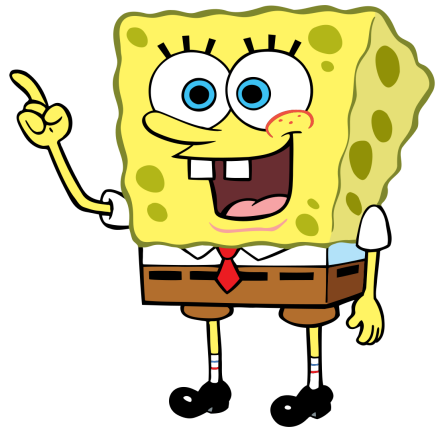



4
7
2
3
6
4
2






7
4
6
4
2
4
2
3





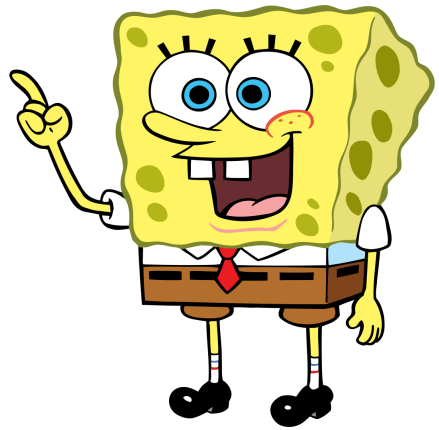


niedozwolony ruch



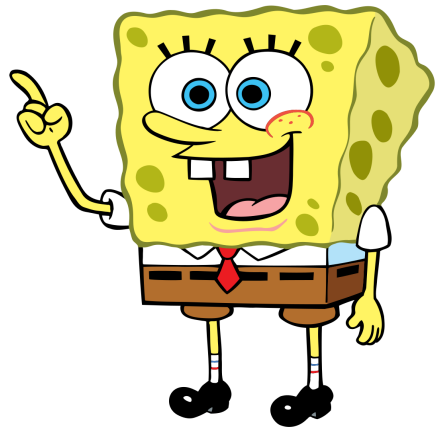



7
4
3
6
4
2
4
2

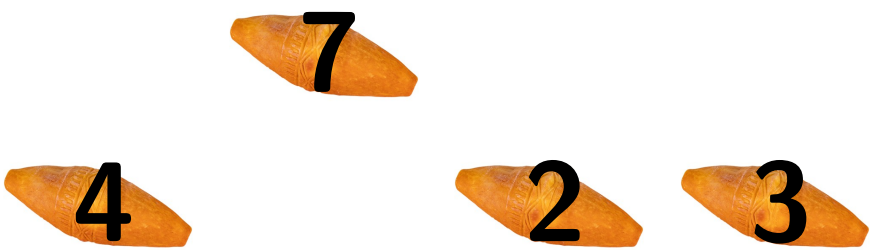


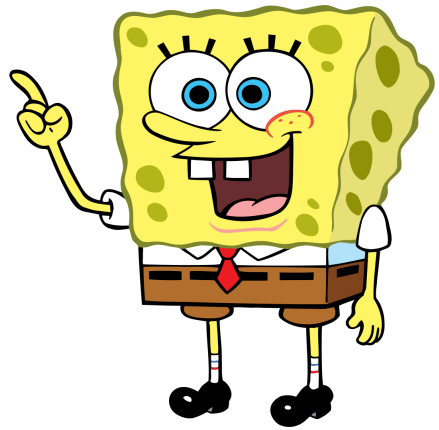



7
6
4
2
3
4
2






7
6
4
2
4
2
3





19 :)









9 :(





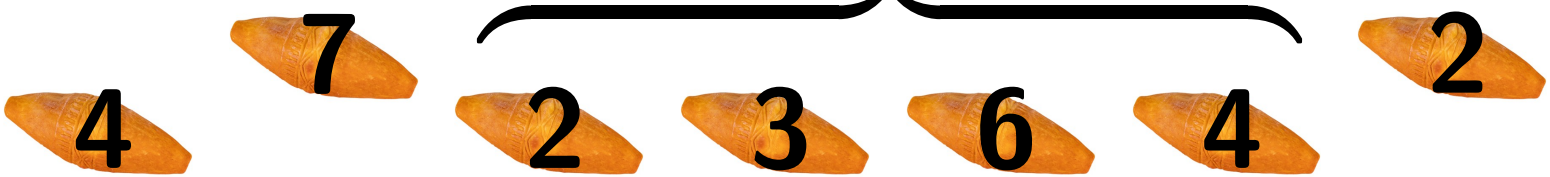
Pierwsza runda:

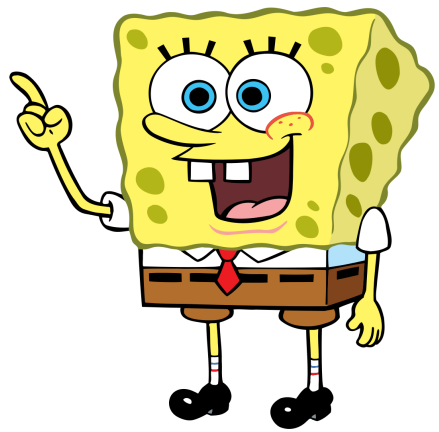






Pierwsza runda: bez dziur $\geqslant 3$



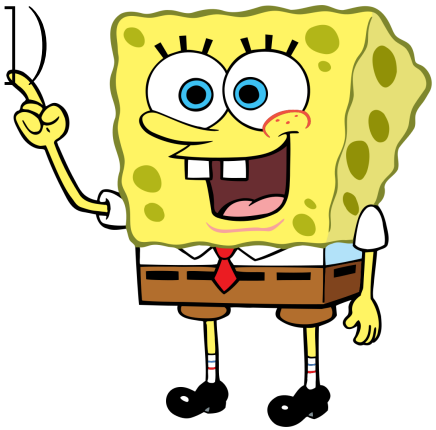




Pierwsza runda: bez dziur $\geqslant 3$



Druga runda: $a[i]$ lub $\max(a[i], a[i+1])$



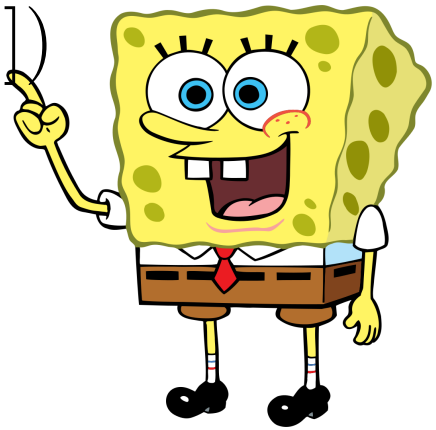


Pierwsza runda: bez dziur $\geqslant 3$

Trzecia runda: reszta



Druga runda: $a[i]$ lub $\max(a[i], a[i+1])$



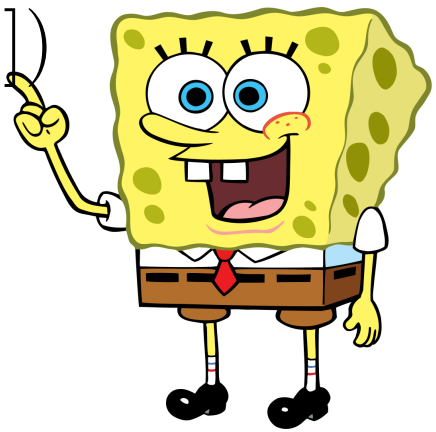


Pierwsza runda: bez dziur $\geqslant 3$

Trzecia runda: reszta



Druga runda: $a[i]$ lub $\max(a[i], a[i+1])$



$$DP[n] = \max \begin{cases} DP[n-2] + a[n], \\ DP[n-3] + a[n] + \\ \quad + \min(a[n-2], a[n-1]) \end{cases}$$

Zadanie I
Henryk Portier i Promień Palindromiczny

Zgłoszeń: 78
Zaakceptowanych: 24

Pierwsze rozwiązanie:
uj4 [Grzybowska, Rajtar, Burczyński]
00:29:06

Autor: Krzysztof Maziarz

Jest pewne słowo A złożone ze znaków $\{0,1\}$. Nie znamy jednak tego słowa, a tylko *promienie palindromiczne*, czyli dla każdego i – długość najdłuższego palindromu o środku w $A[i]$. Zadaniem jest odtworzyć słowo A – jeśli jest więcej rozwiązań, trzeba wypisać wszystkie.

Jest pewne słowo A złożone ze znaków $\{0,1\}$. Nie znamy jednak tego słowa, a tylko *promienie palindromiczne*, czyli dla każdego i – długość najdłuższego palindromu o środku w $A[i]$. Zadaniem jest odtworzyć słowo A – jeśli jest więcej rozwiązań, trzeba wypisać wszystkie.

Pierwsze dwa znaki to 00, 01, 10 lub 11. Sprawdźmy wszystkie te możliwości.

Jest pewne słowo A złożone ze znaków $\{0,1\}$. Nie znamy jednak tego słowa, a tylko *promienie palindromiczne*, czyli dla każdego i – długość najdłuższego palindromu o środku w $A[i]$. Zadaniem jest odtworzyć słowo A – jeśli jest więcej rozwiązań, trzeba wypisać wszystkie.

Pierwsze dwa znaki to 00, 01, 10 lub 11. Sprawdźmy wszystkie te możliwości.

Dla $i \geq 3$ możemy łatwo wyznaczyć $A[i]$ – jeśli promień w $A[i-1]$ jest równy co najmniej 1, to musi być $A[i] = A[i-2]$. Jeśli jest równy 0, to musi być $A[i] \neq A[i-2]$, co też wyznacza $A[i]$ jednoznacznie.

Czyli na podstawie pierwszych dwóch znaków i promieni palindromicznych wyznaczamy całe słowo A. Ale nie mamy gwarancji, że nasze słowo faktycznie pasuje do podanych promieni.

Dla każdego z 4 otrzymanych słów sprawdzamy zatem, czy dla niego wyszłyby takie promienie, jak podane w zadaniu. Służy do tego tzw. algorytm Manachera, ale można też radzić sobie bez niego (np. hashowaniem).

Ciekawostka: zawsze jest albo 0, albo 4 rozwiązania.

Zadanie J
Anteny

Zgłoszeń: 4

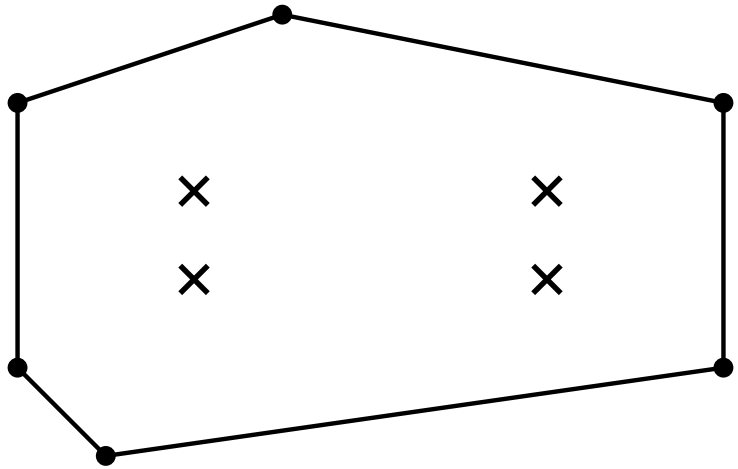
Zaakceptowanych: ??

Pierwsze rozwiązanie:
?

Autor: Daniel Goc

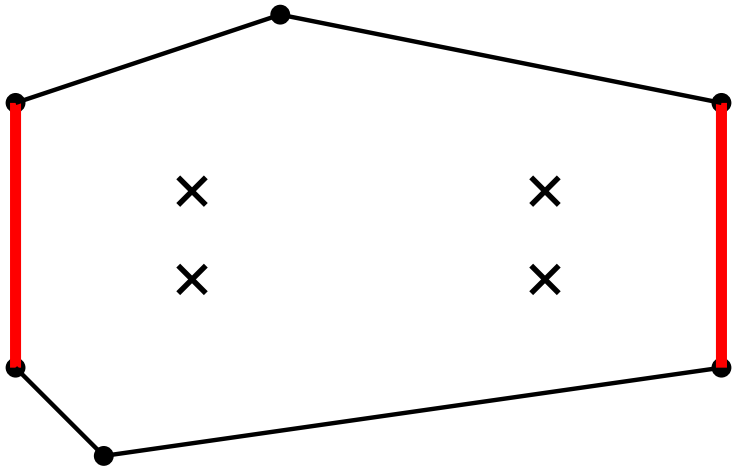
Dany jest wielokąt wypukły i m punktów w środku. Mamy ciąg zapytań postaci „dla danej pary boków wyznaczyc liczbę prostych przechodzących przez pewne dwa spośród tych punktów”.

Żadne 3 punkty nie są współliniowe.



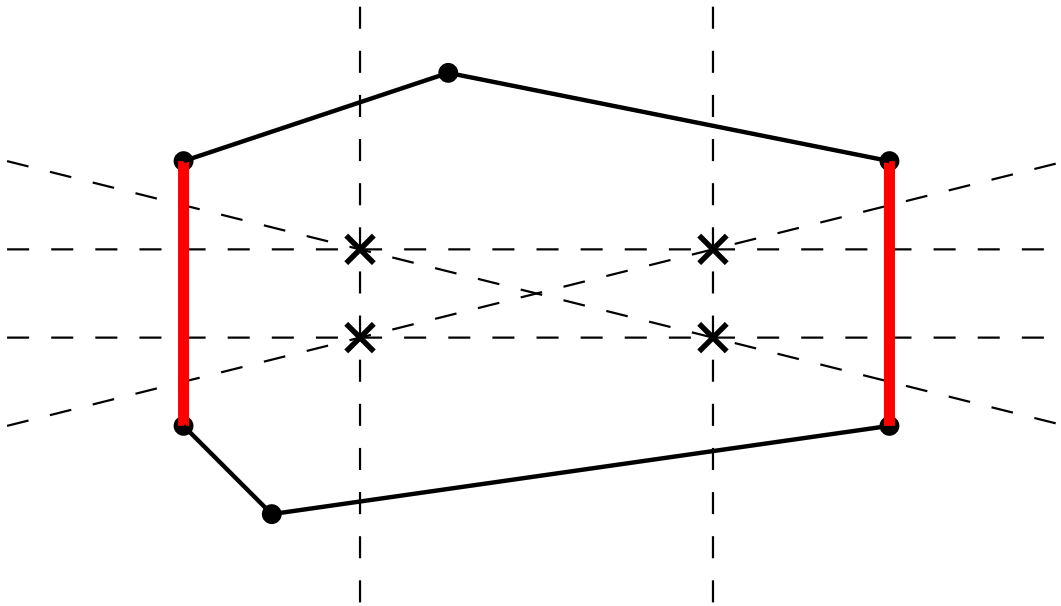
Dany jest wielokąt wypukły i m punktów w środku. Mamy ciąg zapytań postaci „dla danej pary boków wyznaczyc liczbę prostych przechodzących przez pewne dwa spośród tych punktów”.

Żadne 3 punkty nie są współliniowe.



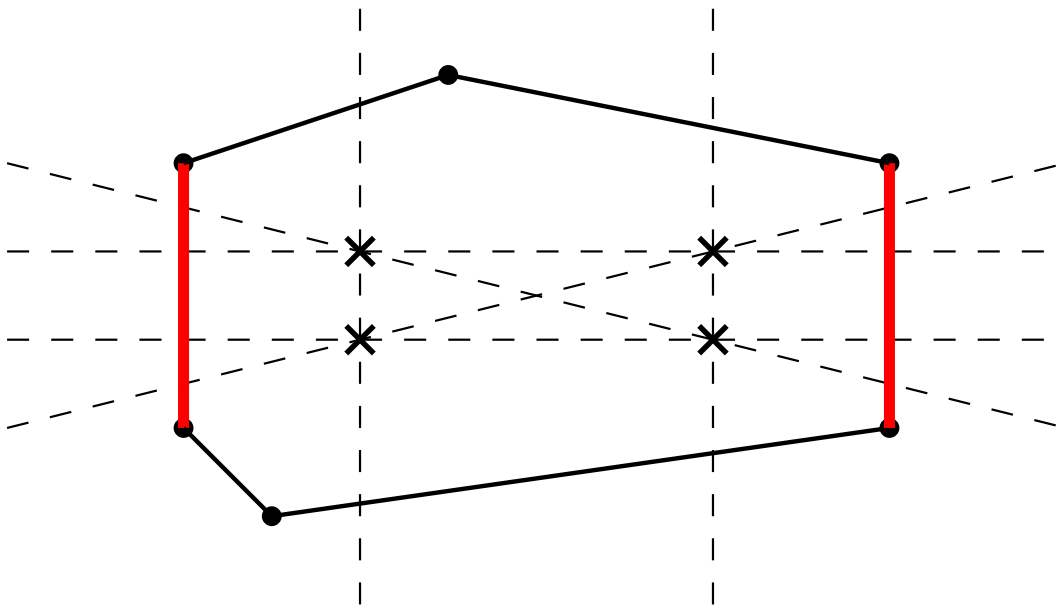
Dany jest wielokąt wypukły i m punktów w środku. Mamy ciąg zapytań postaci „dla danej pary boków wyznaczyc liczbę prostych przechodzących przez pewne dwa spośród tych punktów”.

Żadne 3 punkty nie są współliniowe.



Dany jest wielokąt wypukły i m punktów w środku. Mamy ciąg zapytań postaci „dla danej pary boków wyznaczyć liczbę prostych przechodzących przez pewne dwa spośród tych punktów”.

Żadne 3 punkty nie są współliniowe.



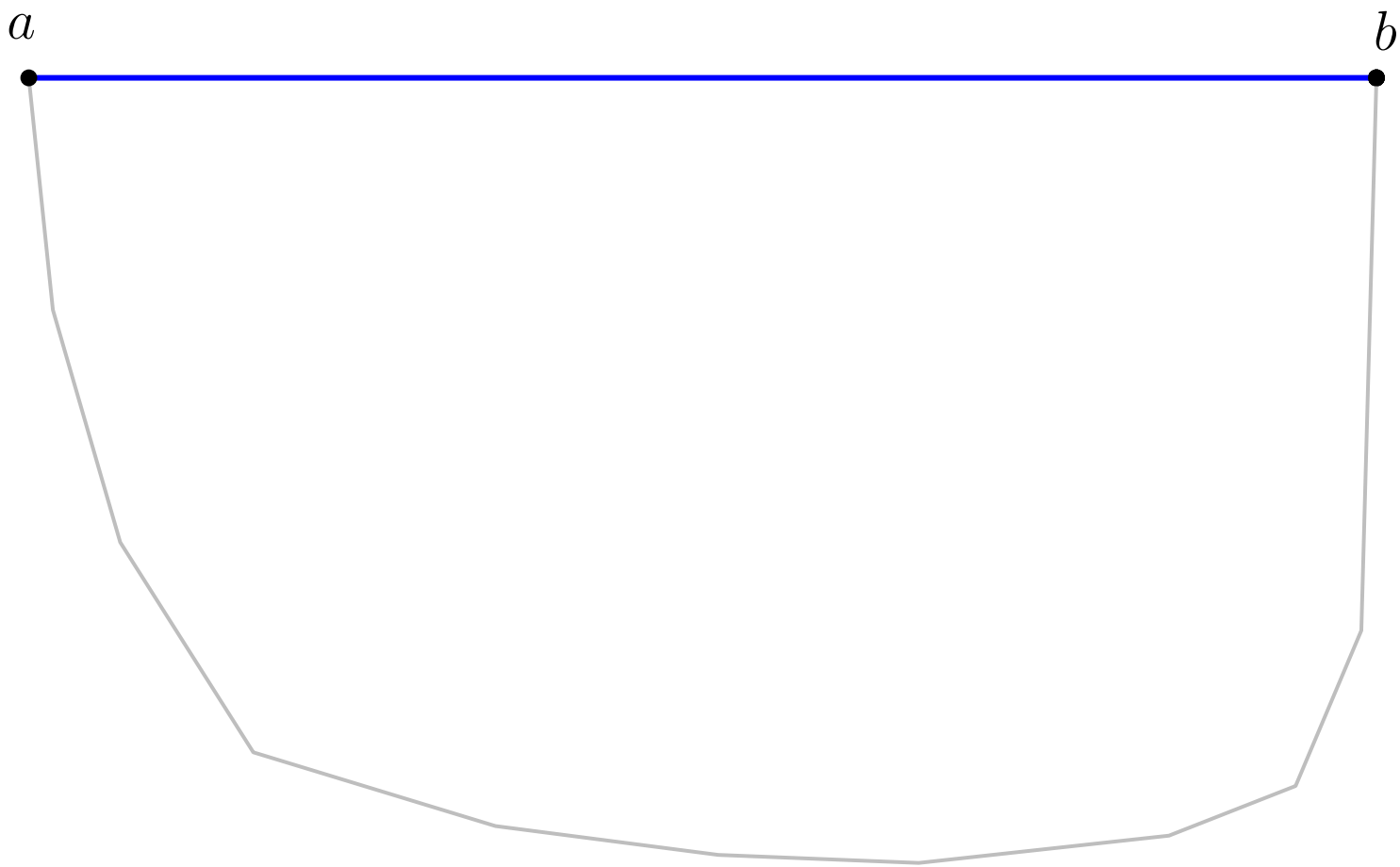
Rozwiązanie: obsługujemy każde zapytanie osobno w czasie $\mathcal{O}(m \log m)$.



\# prostych przecinających boki aa' oraz bb' =
\# prostych poniżej odcinka ab
- \# prostych poniżej odcinka ab'
- \# prostych poniżej odcinka $a'b$
+ \# prostych poniżej odcinka $a'b'$

THEORETICAL COMPUTER SCIENCE
Jagiellonian University

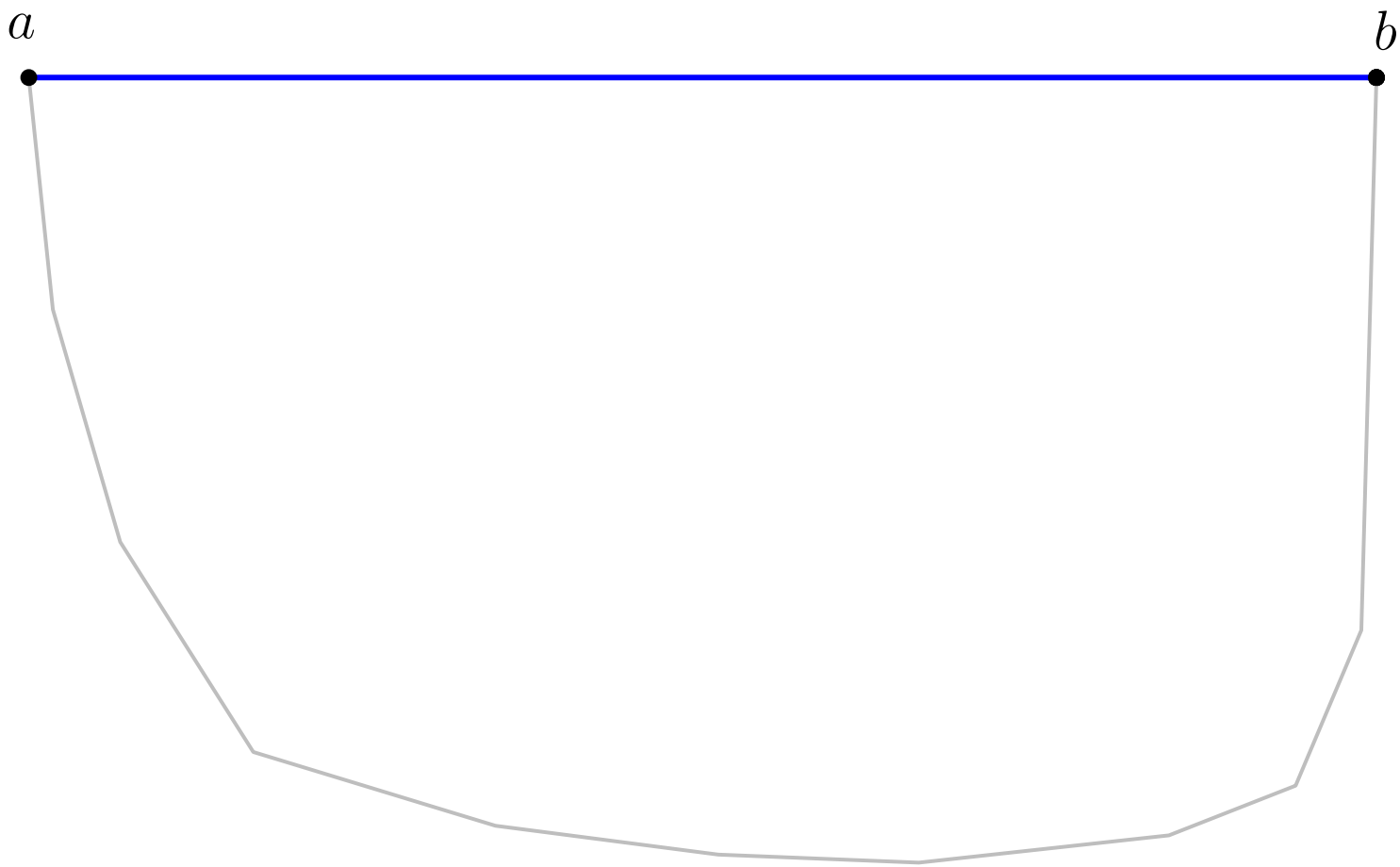
Ile prostych jest poniżej odcinka ab?



~~Ile prostych jest ponizej odcinka *ab*?~~

Zostawmy tylko punkty poniżej odcinka *ab*.

Ile prostych przecina odcinek *ab*?

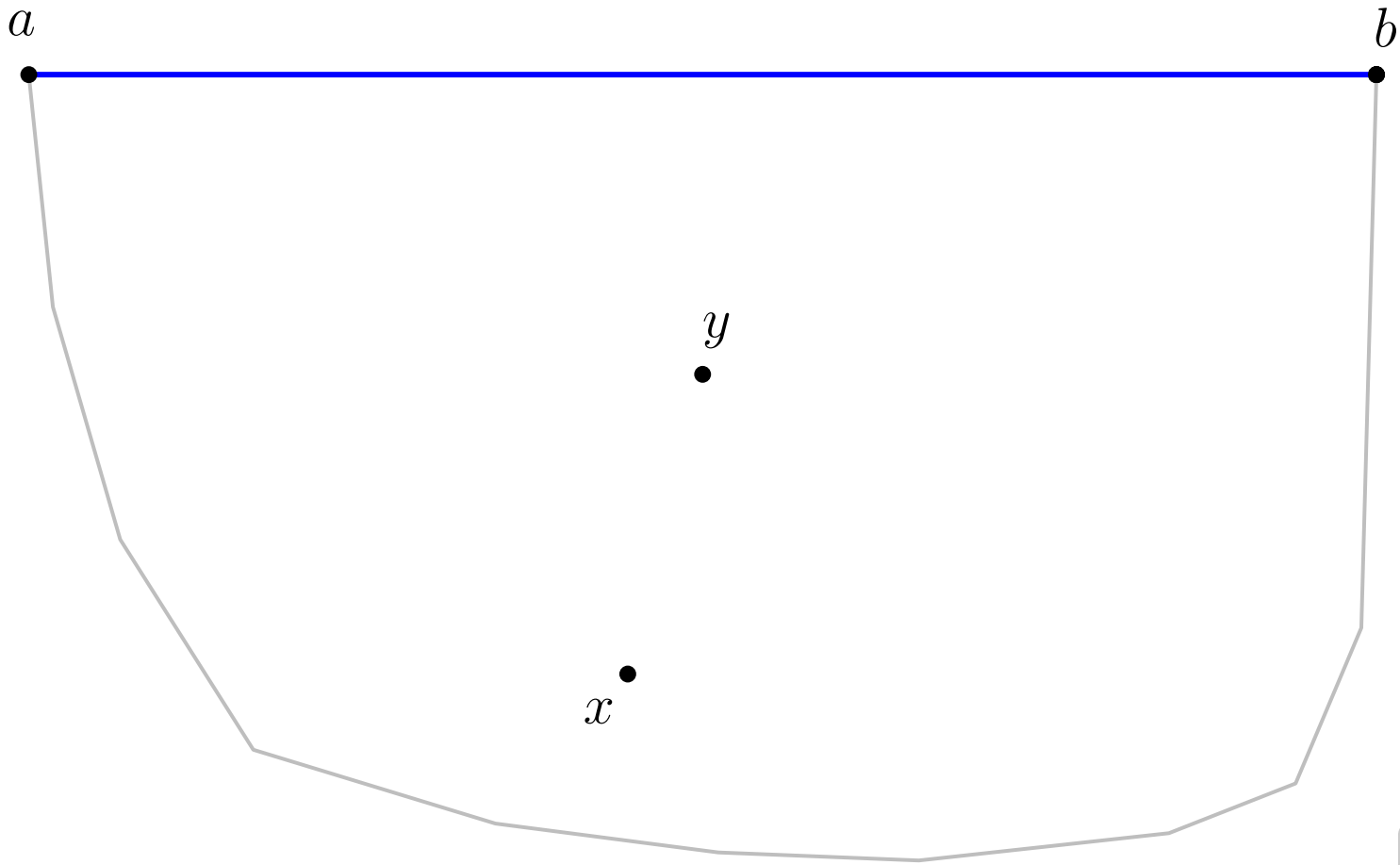


~~Ile prostych jest ponizej odcinka *ab*?~~

Zostawmy tylko punkty poniżej odcinka *ab*.

Ile prostych przecina odcinek *ab*?

Jaki warunek musi spełniać punkt *y*, by prosta *xy* przecinała odcinek *ab*?

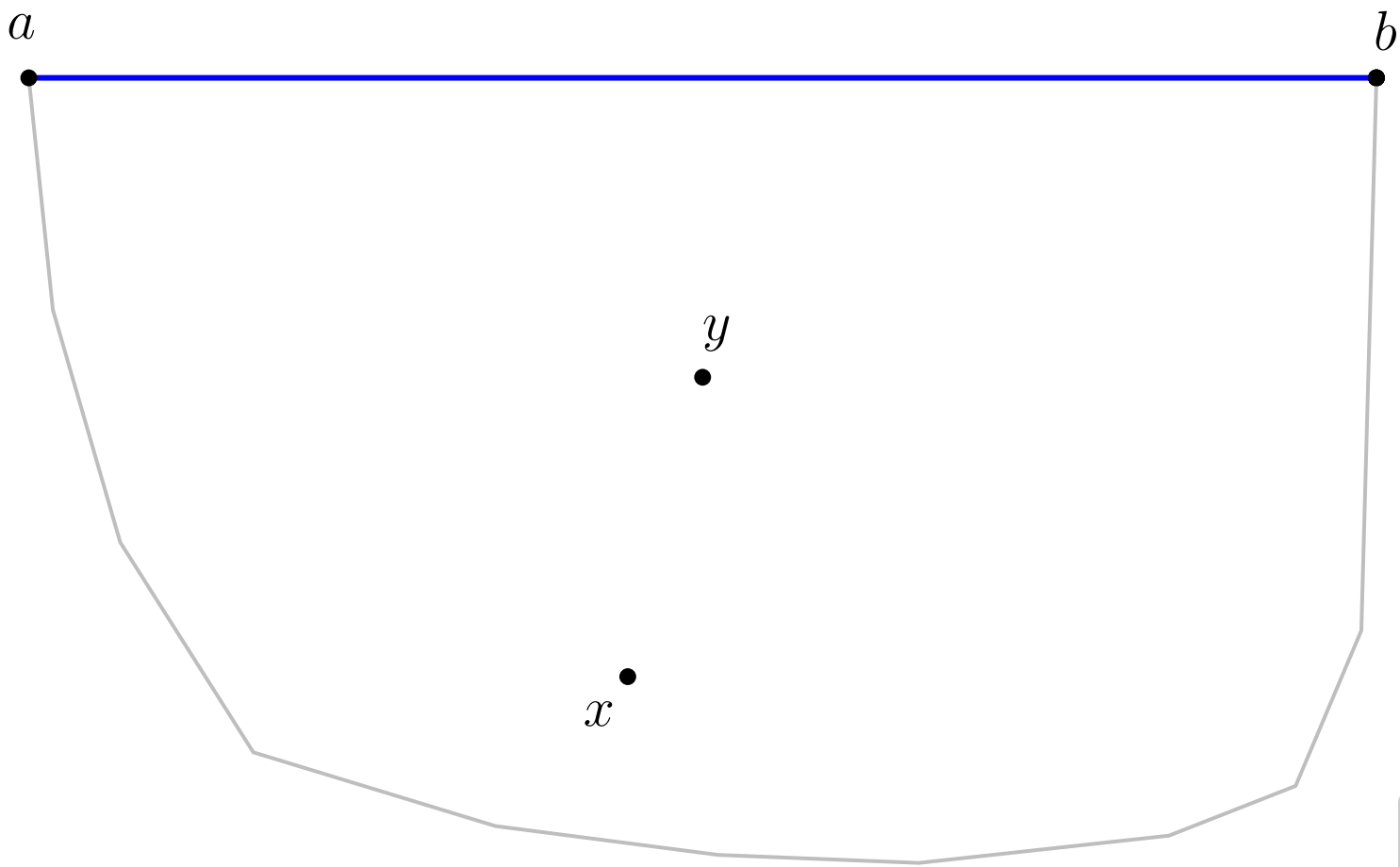


~~Ile prostych jest ponizej odcinka *ab*?~~

Zostawmy tylko punkty poniżej odcinka *ab*.

Ile prostych przecina odcinek *ab*?

Jaki warunek musi spełniać punkt *y*, by półprosta *xy* przecinała odcinek *ab*?

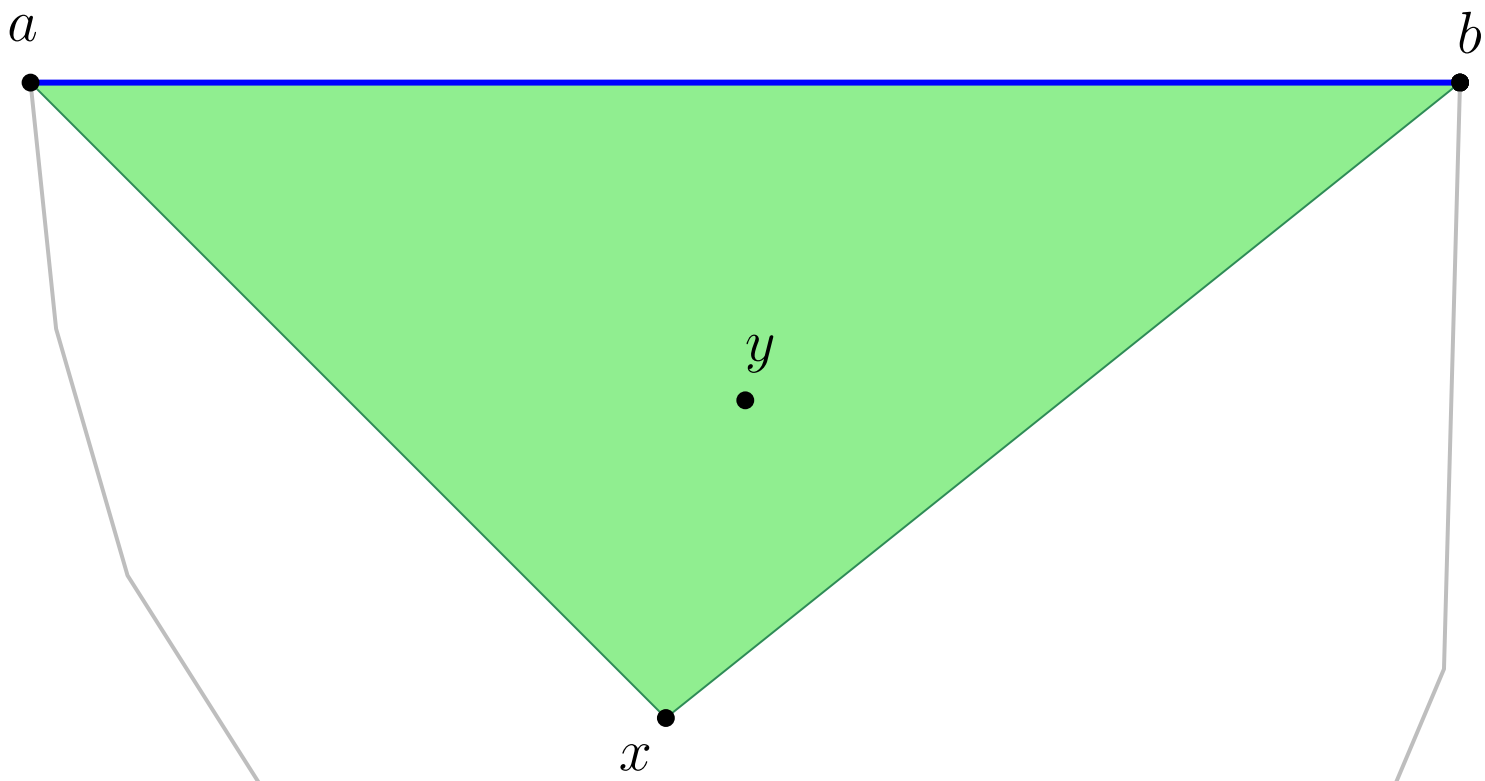


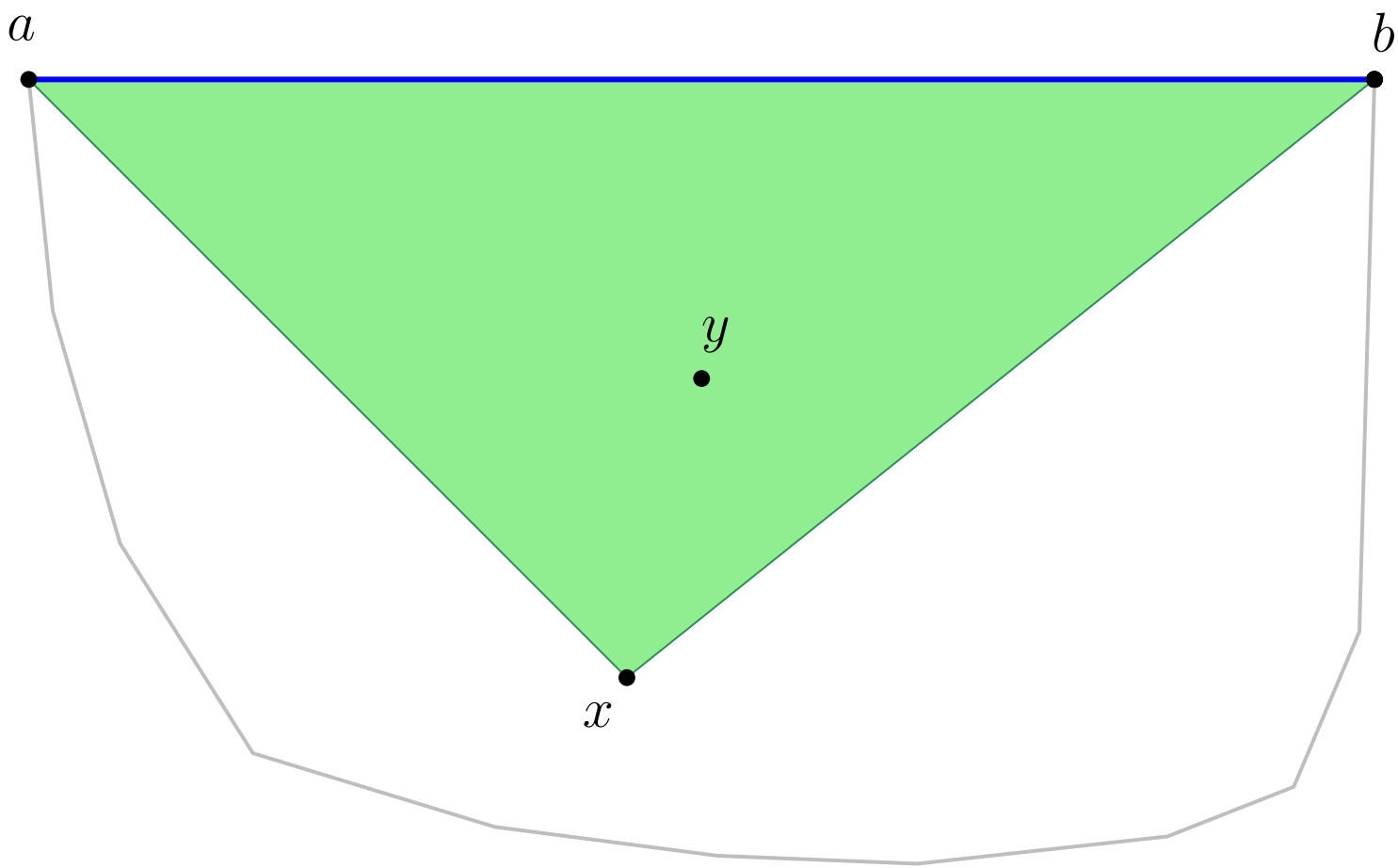
~~Ile prostych jest ponizej odcinka *ab*?~~

Zostawmy tylko punkty poniżej odcinka *ab*.

Ile prostych przecina odcinek *ab*?

Jaki warunek musi spełniać punkt *y*, by półprosta *xy* przecinała odcinek *ab*?





Punkt y musi należeć do trójkąta axb.

WKW na przecięcie półprostej xy z odcinkiem ab:
Punkt y musi należeć do trójkąta axb. $\iff$
W porządku kątowym wokół a punkt y musi być na lewo od punktu x, zaś w porządku kątowym wokół b punkt y musi być na prawo od punktu x.

WKW na przecięcie prostej xy z odcinkiem ab:
w porządku kątowym wokół a punkty x, y muszą leżeć INACZEJ niż w porządku kątowym wokół b.

Chcemy policzyć pary x, y, leżące względem siebie w jednym porządku inaczej niż w drugim.

Problem sprowadza się do policzenia inwersji w ciągu, który rozwiązuje się w czasie $\mathcal{O}(m \log m)$ np. za pomocą algorytmu *mergesort*.



Zadanie K
Duch

Zgłoszeń: 21
Zaakceptowanych: ??

Pierwsze rozwiązanie:
?

Autor: Krzysztof Maziarz

Dane są prostokąty, każdy poruszający się ruchem liniowym z jakimś wektorem prędkości. Policz największe pole przecięcia prostokątów w każdym z zadanych przedziałów czasu.

Każdy prostokąt daje cztery niezależne ograniczenia (półpłaszczyzny) na przecięcie prostokątów: dolne i górne ograniczenie na x oraz na y.

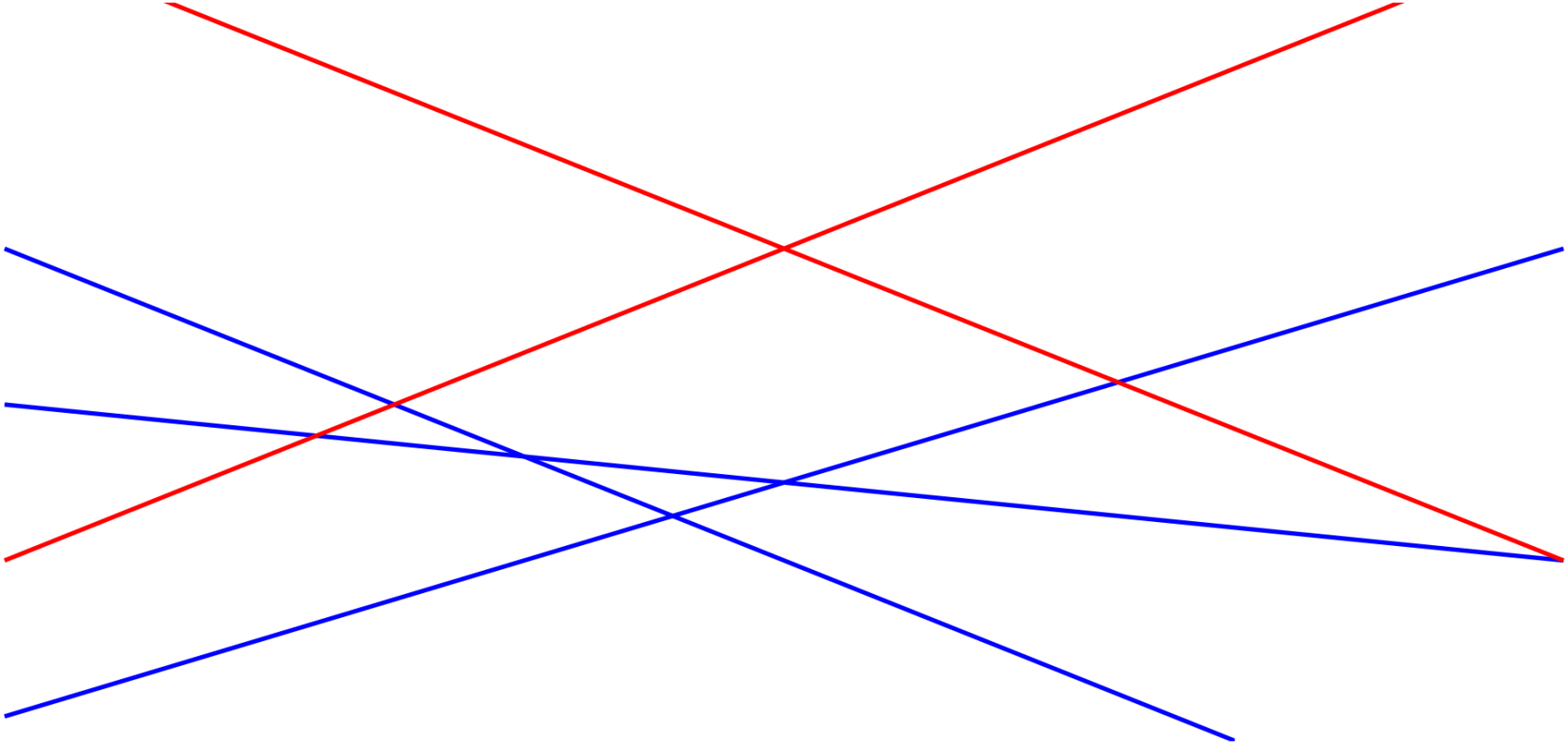
Skupmy się na analizowaniu i wyznaczaniu ograniczeń na x.

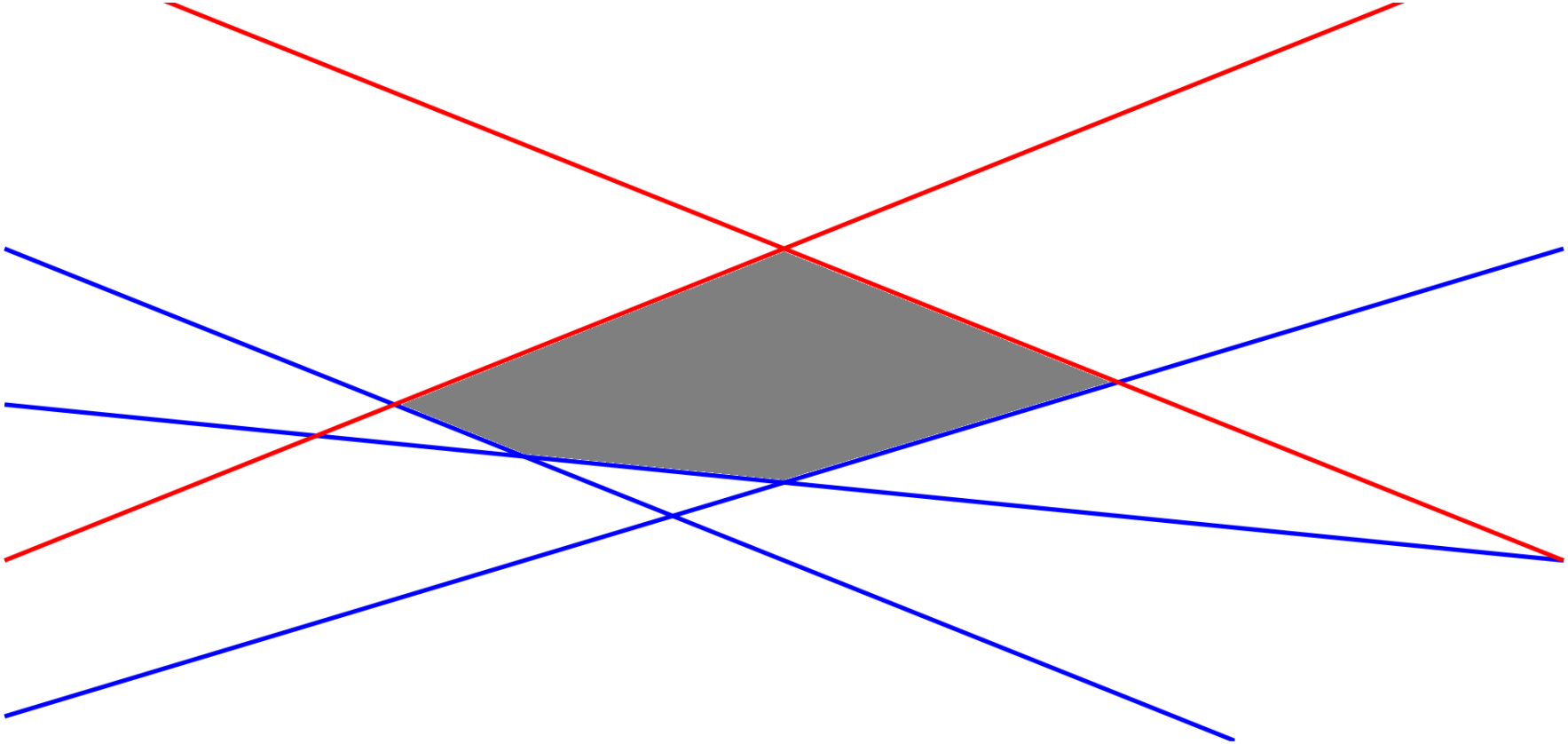
Każdy prostokąt wyznacza następujące ograniczenia w momencie t:

$$x \geq x_1 + t \cdot v_x$$

$$x \leq x_2 + t \cdot v_x$$

N ograniczeń dolnych daje nam N funkcji liniowych (półpłaszczyzn), to samo dla górnych.





Wysokość obszaru pomiędzy ograniczeniami dolnymi a górnymi jest równa szerokości przecięcia prostokątów z zadania, co nazwiemy $szer(t)$ dla momentu t.

Wysokość obszaru pomiędzy ograniczeniami dolnymi a górnymi jest równa szerokości przecięcia prostokątów z zadania, co nazwiemy $szer(t)$ dla momentu t.

Funkcja $szer(t)$ jest bitoniczna, kawałkami liniowa oraz wklęsła, o ile ograniczymy się do przedziału czasu t, gdzie jest to funkcja niezerowa.

Funkcja $wys(t)$ (dla współrzędnych y) spełnia podobne własności. Interesuje nas maksymalizowanie pola przecięcia czyli:

$$pole(t) = szer(t) \cdot wys(t)$$

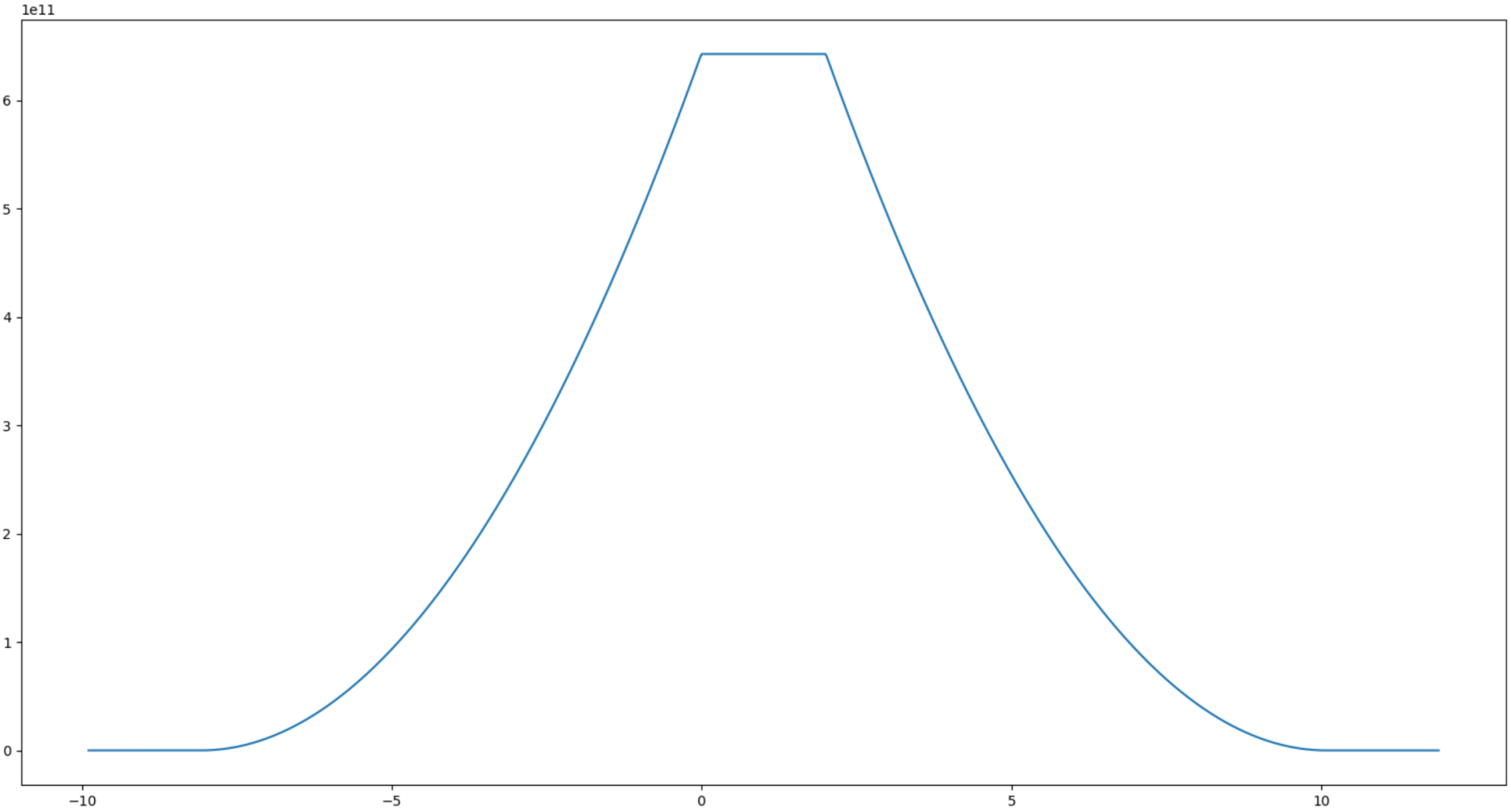
co jest bitoniczną funkcją kawałkami kwadratową.

Funkcja $wys(t)$ (dla współrzędnych y) spełnia podobne własności. Interesuje nas maksymalizowanie pola przecięcia czyli:

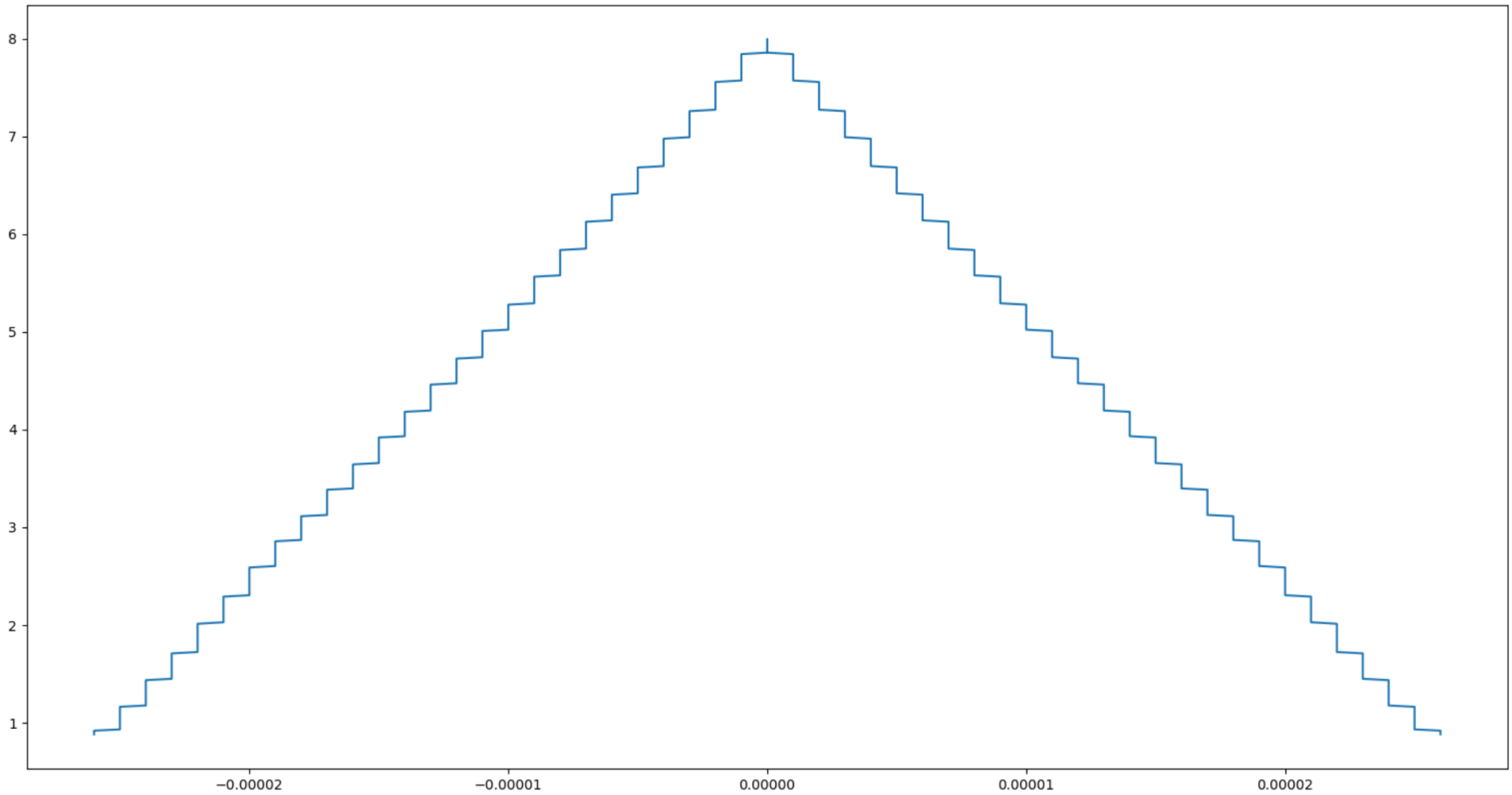
$$pole(t) = szer(t) \cdot wys(t)$$

co jest bitoniczną funkcją kawałkami kwadratową.

Dowód bitoniczności pozostawiamy czytelnikowi.



Funkcja $pole(t)$ nie musi być ani wypukła ani wklęsła.
(a nawet różnie bywa)



Funkcja *pole*(t) nie musi być ani wypukła ani wklęsła.
(a te schodki nie są idealnie poziome)

Rozwiązanie I (bez bitoniczności)

Zbieramy ograniczenia na x i wyznaczamy przecięcie półpłaszczyzn, to samo dla y. Dzielimy czas na $\mathcal{O}(n)$ przedziałików, takich że w każdym $szer(t)$ oraz $wys(t)$ są funkcjami liniowymi – wtedy ich iloczyn jest funkcją kwadratową.

W każdym przedziałiku szukamy maksimum wzorem lub wyszukiwaniem ternarnym, budujemy na tych wartościach drzewo przedziałowe i w $\mathcal{O}(\log n)$ odpowiadamy na zapytania, uważając na końcówki przedziału z zapytania.

Da się łatwiej. Skoro funkcja $pole(t)$ jest bitoniczna, to kusi wyszukiwanie ternarne, by znaleźć globalne maksimum.

Ale nie da się wyszukiwać ternarnie po funkcji, która ma remisy poza maksimum!

Da się łatwiej. Skoro funkcja $pole(t)$ jest bitoniczna, to kusi wyszukiwanie ternarne, by znaleźć globalne maksimum.

Ale nie da się wyszukiwać ternarnie po funkcji, która ma remisy poza maksimum!

...więc ograniczamy się do przedziału czasu, gdzie wartości są niezerowe. Tam funkcja $pole(t)$ ściśle rośnie, potem być może ma remis na maksimum, potem ściśle maleje.

Rozwiązanie II (z bitonicznością)

Po znalezieniu przecięcia półpłaszczyzn wyszukujemy ternarnie (lub binarnie po pochodnej) globalne maksimum.

Jeśli zapytanie zawiera ten optymalny moment, to znamy od razu wynik. Jeśli nie, to wystarczy rozważyć tylko początek i koniec przedziału.

W tych dwóch momentach liczymy wynik przez policzenie $szer(t)$ i $wys(t)$ (wyszukujemy binarnie odcinek otoczki w którym ląduje t).

Złożoność obu rozwiązań to zgrubsza $\mathcal{O}(n \log n)$. Rozwiązanie II ma logarytm z trochę większej wartości, bo musimy zrobić ternary search po zakresie (i do zadanej dokładności).

Ciekawostka: Rozwiązanie I da się zaimplementować w arytmetyce liczb wymiernych.

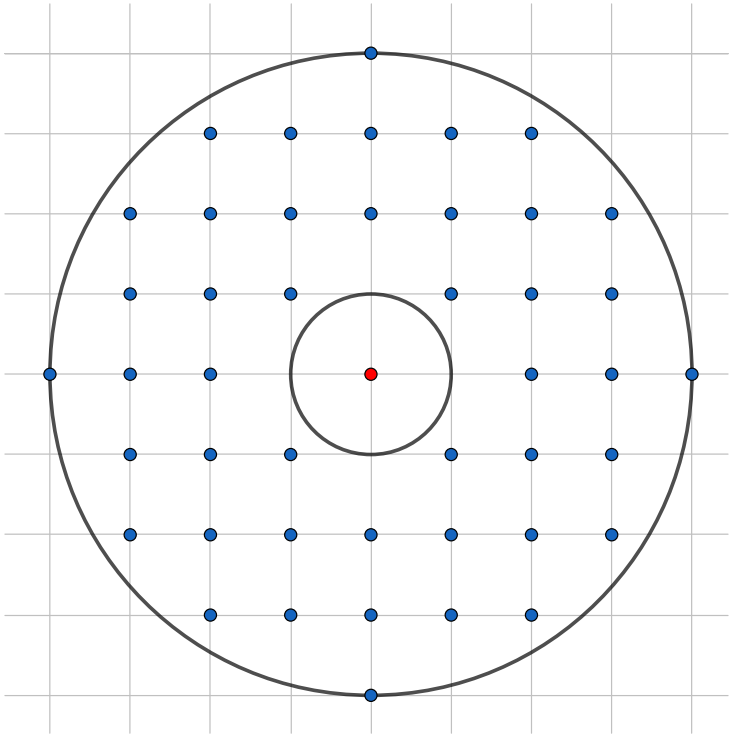
Zadanie L
Obwarzanek

Zgłoszeń: 11
Zaakceptowanych: 2

Pierwsze rozwiązanie:
uw1 [Kondraciuk, Skiba, Paluszek]
02:53:11

Autor: Krzysztof Maziarz

Dla punktu kratowego (a, b) i liczb całkowitych nieujemnych L oraz R, rozważamy zbiór punktów kratowych których odległość od (a, b) jest w przedziale $(L, R]$. Każdy zbiór który można uzyskać w opisany sposób dla pewnych a, b, L, R nazywamy *obwarzankiem*.



Zaczynamy od pustego zbioru punktów S, i n razy dodajemy do niego nowy punkt kratowy. Po każdej takiej operacji należy stwierdzić, czy obecny zbiór S jest obwarzankiem.

Dodatkowo, mamy *bardzo mało dostępnej pamięci* – nie jesteśmy nawet w stanie utrzymywać w pamięci zbioru S.

Za to mamy gwarancję, że wszystkie punkty mają niezbyt duże współrzędne (na moduł nie przekraczające $C = 5000$).

Możemy utrzymywać najmniejszą i największą współrzędną x po punktach znajdujących się w S (l_x, r_x), oraz analogicznie dla y (l_y, r_y).

Warunkiem koniecznym aby obecny zbiór punktów był obwarzankiem jest $r_x - l_x = r_y - l_y$. Ponadto, możemy wyliczyć zewnętrzny promień

$$R = \frac{r_x - l_x}{2}$$

oraz środek

$$(a, b) = \left(\frac{l_x + r_x}{2}, \frac{l_y + r_y}{2} \right)$$

Załóżmy, że dla każdego $r \in \{1, 2, ..., 5000\}$ znamy liczbę punktów kratowych zawartych w kole o środku w $(0, 0)$ i promieniu r. Wtedy mając wyliczony promień zewnętrzny R, i znając liczbę wczytanych punktów, możemy jednoznacznie wyznaczyć L.

Znamy już teraz wartości a, b, L, R – albo S jest dokładnie takim obwarzankiem, albo nie jest obwarzankiem wcale.

By odpowiedzieć na pytanie czy S faktycznie jest obwarzankiem o zadanych parameterach, musimy utrzymywać jeszcze jeden rodzaj informacji. Niech p będzie liczbą pierwszą, a $h(x, y)$ pewną funkcją hashującą $\mathbb{Z}^2 \to \mathbb{Z}_p$.

Zbiór punktów S hashujemy sumując h po jego elementach:

$$hash(S) = \left(\sum_{(x,y) \in S} h(x, y) \right) \mod p$$

Zakładając, że $h(x, y)$ jesteśmy w stanie wyliczać w czasie stałym, możemy bez problemu utrzymywać wartość $hash(S)$.

Żeby S był zadanym obwarzankiem, musi zachodzić

$$hash(S) = (circHash(a, b, R) - circHash(a, b, L)) \mod p$$

gdzie $circleHash(a, b, r) = hash(\{(x, y) : dist((x, y), (a, b)) \leq r\})$.

Załóżmy, że znane nam są wartości $circleHash(0, 0, r)$ dla $r \in \{1, 2, ..., 5000\}$. Jeśli funkcja h będzie miała taką własność, że znając sumaryczny hash zbioru możemy wyznaczyć hash tegoż zbioru po przesunięciu jego wszystkich punktów o zadany wektor, to z $circleHash(0, 0, r)$ wyznaczymy $circleHash(a, b, r)$.

Czas wybrać funkcję h. Niech

$$h(x, y) = r^x s^y \mod p$$

gdzie r i s są stałymi wylosowanymi jednostajnie z $[1, p-1]$.

Po odpowiednim preprocessingu, $h(x, y)$ jestesmy w stanie liczyć w czasie stałym; ponadto, jeśli przesuniemy S o wektor (a, b), to $hash(S)$ zmieni się na $r^a s^b hash(S)$.

Podczas naszego rozwiązania założylismy, że dla każdego $r \in \{1, 2, ..., 5000\}$ znamy liczbę punktów kratowych w kole o promieniu r i środku w $(0, 0)$, oraz sumaryczny hash tych punktów.

Wartości te możemy policzyć na samym początku, wykonując preprocessing w czasie $\mathcal{O}(C^2)$.

Złożoność: $\mathcal{O}(C^2 + n \log C)$ (po drobnych optymalizacjach $\mathcal{O}(C^2 + n)$).

Dlaczego to działa?

- szansa, że dla dwóch różnych zbiorów A, B zajdzie $hash(A) = hash(B)$ jest rzędu $\mathcal{O}(\frac{C}{p})$ (z lematu Schwartza–Zippela)
- szansa, że zbiór S błędnie sklasyfikujemy jako obwarzanek, nie przekracza $\mathcal{O}(\frac{C^2}{p})$

Jury zawodów

Kamil „Errichto” Dębowski
Lech Duraj
Grzegorz Guśpiel
Vladyslav Hlembotskyi
Bartosz Kostka
Krzysztof Maziarz
Adam Polak

Betatesterzy

Jan Gwinner
Witold Jarnicki
Mateusz Radecki
Maciej Wawro

Dziękujemy!