# Akademickie Mistrzostwa Polski w Programowaniu Zespołowym

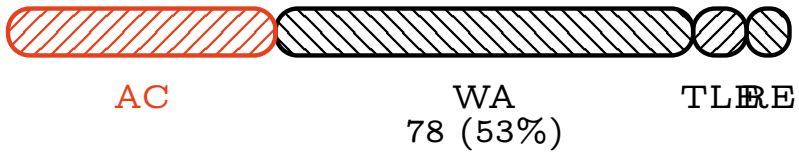
## Prezentacja rozwiązań zadań

28 października 2012

# Jutro

Autor zadania: Tomasz Idziaszek

Zgłoszenia: 146 z 775 (18%)
Zaakceptowane przez 50 z 54 drużyn (92%)



# Jutro

Jak długo można zwlekać, jeśli ma się do wykonania $n$ zadań, z których $i$-te zajmuje $d_i$ kolejnych dni i musi zostać wykonane przed upływem $t_i$ dni?

Obserwacja: dla dowolnego uporządkowania zadań, w którym zadania o późniejszych terminach występują później, istnieje rozwiązanie optymalne, w którym wykonujemy zadania w tej kolejności.

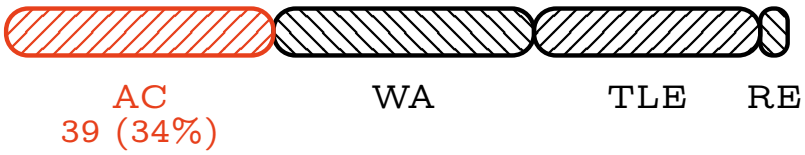
- Zatem sortujemy, a następnie wykonujemy zadania od najpóźniejszych, każde zadanie próbując wykonać jak najpóźniej.
- Jeśli przez $c_i$ oznaczymy dzień, po którym zaczynamy zadanie $i$-te w kolejności, to $c_i = \min(t_i, c_{i+1}) - d_i$. Wynikiem jest $c_1$.

Czas działania $O(n \log n)$.

# Inwersje

Autor zadania: Krzysztof Diks

Zgłoszenia: 114 z 775 (14%)
Zaakceptowane przez 39 z 54 drużyn (72%)



# Inwersje

Dana jest permutacja $a_1, \ldots, a_n$ liczb od 1 do $n$.
Chcemy wyznaczyć liczbę spójnych składowych grafu o $n$ wierzchołkach, w którym wierzchołki $a_i$, $a_j$ są połączone krawędzią jeśli $i < j$ oraz $a_i > a_j$.

- Przeglądamy kolejne elementy i utrzymujemy listę dotychczas napotkanych spójnych składowych. Wystarczy, że dla każdej spójnej składowej pamiętamy jej największy element. Te elementy trzymamy na stosie — na czubku stosu największy z nich.
- Gdy przychodzi nowy element, ściągamy ze stosu wszystkie elementy od niego większe i łączymy wszystko (*find-union*) w jedną spójną składową. Następnie odkładamy ją na stos i idziemy dalej.

Całość działa w czasie $O(n \log^* n)$ lub $O(n \log n)$ w zależności od tego, jak wykonamy sortowanie odpowiedzi.

# Inwersje

Prostsze rozwiązanie uzyskamy, zauważając, że każda spójna składowa będzie spójnym fragmentem ciągu *oraz* spójnym przedziałem numerów wierzchołków.

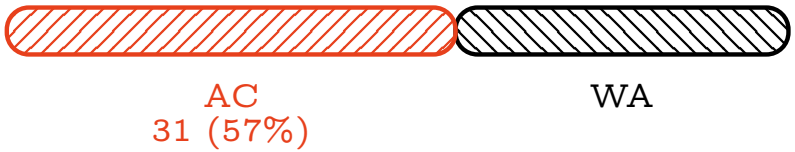
- Przeglądamy kolejne elementy i zgłaszamy znalezienie nowej spójnej składowej, jeśli maksymalny dotychczas wczytany element jest równy liczbie wczytanych elementów.

Czas działania $O(n)$.

# DNA

Autor zadania: Jakub Łącki

Zgłoszenia: 54 z 775 (6%)
Zaakceptowane przez 31 z 54 drużyn (57%)



# DNA

Mamy $n$-literowe słowo $s$ nad alfabetem $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ i chcemy znaleźć $n$-literowe słowo $s'$ takie, żeby najdłuższy wspólny podciąg tych słów był jak najkrótszy.

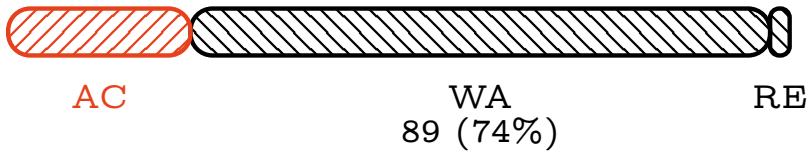
Załóżmy, że litera $\mathtt{G}$ występuje najrzadziej w $s$ ($k$ razy).
Wtedy dla słowa $s' = \underbrace{\mathtt{GG}\ldots\mathtt{G}}_{n}$ mamy $NWP(s, s') = k$.

Dlaczego nie da się lepiej?
Każda litera ma co najmniej $k$ wystąpień w $s$. Ponieważ $k \leqslant n/4$, więc dla dowolnego słowa $s'$ istnieje litera, która występuje co najmniej $k$ razy w $s'$. Tak więc $NWP(s, s') \geqslant k$.
Czas działania $O(n)$.

# Ciąg

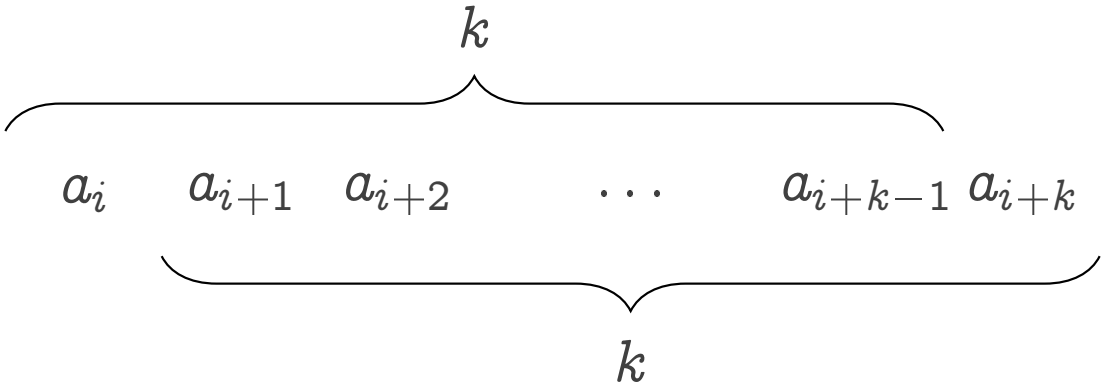
Autor zadania: Jakub Radoszewski

Zgłoszenia: 120 z 775 (15%)
Zaakceptowane przez 28 z 54 drużyn (51%)



# Ciąg

Ile minimalnie liczb należy zmienić w ciągu $a_1, a_2, \ldots, a_n$, żeby każdy spójny fragment długości $k$ miał parzystą sumę?




Rozważając dwa kolejne fragmenty, zauważamy,
że wyrazy $a_i$ oraz $a_{i+k}$ muszą mieć tę samą parzystość.

Poza tym pierwszy fragment musi mieć parzystą sumę.

# Ciąg

| | | | | | |
|---|---|---|---|---|---|
| $a_1$ | $a_2$ | | | | $a_k$ |
| $a_{k+1}$ | $a_{k+2}$ | | | | $a_{2k}$ |
| | | | | | |
| | | | $a_n$ | | |

Wyrazy w każdej kolumnie muszą mieć tę samą parzystość oraz w pierwszym wierszu musi być parzyście wiele nieparzystych wyrazów.
Niech $p_i$, $n_i$ oznaczają liczbę parzystych i nieparzystych wyrazów w $i$-tej kolumnie.

- W każdej kolumnie zmieniamy $\min(p_i, n_i)$ wyrazów.
- Jeśli wskutek tego pierwszy wiersz uzyskał nieparzystą sumę, to poprawiamy kolumnę o minimalnym $\max(p_i, n_i) - \min(p_i, n_i)$.

Czas działania $O(n)$.

# Hydra

Autor zadania: Tomasz Idziaszek

Zgłoszenia: 70 z 775 (9%)
Zaakceptowane przez 34 z 54 drużyn (62%)

# Hydra

Niech $u_i$, $z_i$ będą liczbą machnięć miecza potrzebnych do ucięcia i do zmasakrowania głowy $i$, zaś $G_i$ — zbiorem głów, które powstają po ucięciu głowy $i$.
Niech $koszt[i]$ oznacza minimalną liczbę machnięć potrzebnych do uśmiercenia głowy $i$. Mamy rekurencję:

$$koszt[i] = \min(z_i, u_i + \sum_{j \in G_i} koszt[j]). \tag{1}$$

Wynikiem zadania jest $koszt[1]$.

Pytanie: w jakiej kolejności obliczać rekurencję?

# Hydra

Przez $C$ oznaczymy zbiór tych wierzchołków $i$, dla których obliczyliśmy już $koszt[i]$. Będziemy utrzymywać niezmiennik:

$$\text{jeśli } G_i \subseteq C, \text{ to } i \in C.$$

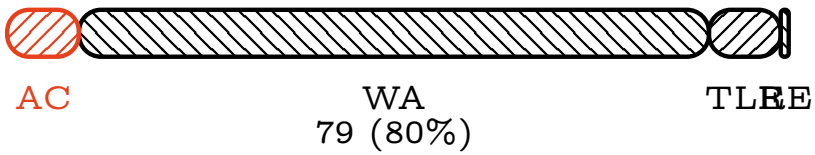
Algorytm składa się z kolejnych faz. Na początku fazy znajdujemy wierzchołek $i \notin C$, o najmniejszej wartości $z_i$. Ustalamy $koszt[i] = z_i$ i dodajemy $i$ do zbioru $C$. W drugiej części fazy przywracamy niezmiennik, tzn. dopóki znajdujemy taki wierzchołek $i$, że $G_i \subseteq C$, $i \notin C$, to obliczamy $koszt[i]$ ze wzoru (1) i dodajemy $i$ do $C$.

Całość możemy wykonać w czasie $O(n \log n + \sum_i k_i)$, gdyż sortujemy wierzchołki po $z_i$ oraz przechodzimy graf transponowany.

# Króliki

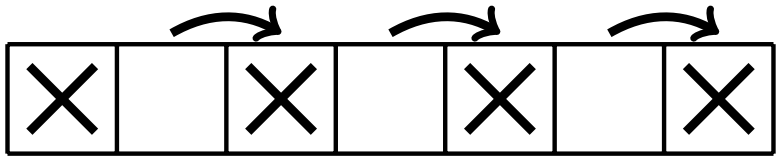
Autor zadania: Tomasz Idziaszek

Zgłoszenia: 98 z 775 (12%)
Zaakceptowane przez 9 z 54 drużyn (16%)



# Króliki

Prostszy problem: ile strzałów potrzeba, by przepędzić wszystkie króliki, które na początku stały na pozycjach z pewnego spójnego przedziału grządek o długości $m$.



Wystarczy zatem $\lfloor \frac{m}{2} \rfloor + 1$ strzałów. Tyle też potrzeba, bo między kolejnymi strzałami nie może być więcej niż jedna grządka przerwy.

Ponadto nie opłaca się mieć jednogrządkowej przerwy pomiędzy kolejnymi przedziałami. Zatem przedziały można przepędzać niezależnie.

# Króliki

Wystarczy więc wybrać zbiór maksymalnych przedziałów, z których chcemy przepędzić króliki.

Zastosujemy programowanie dynamiczne. Stanem będzie:

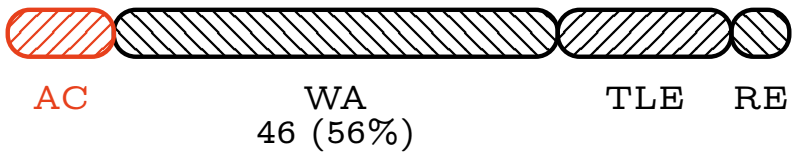
- liczba rozważonych grządek,
- liczba wykonanych strzałów,
- status dwóch ostatnio rozważonych grządek.

Ponieważ rzecz dzieje się na kółku, warto wybrać *a priori* status dwóch ostatnich grządek w całym kółku i też go trzymać w stanie.
Czas działania $O(nk)$.

# Biuro podróży

Autor zadania: Jakub Radoszewski

Zgłoszenia: 81 z 775 (10%)
Zaakceptowane przez 11 z 54 drużyn (20%)



# Biuro podróży

Tworzymy ważony graf: wierzchołkami są pola z atrakcjami, a krawędzie łączą atrakcje o różnych współczynnikach ciekawości (WC) i są skierowane w kierunku większych WC.

- waga krawędzi to odległość w metryce miejskiej między atrakcjami,
- waga wierzchołka to cena za odwiedzenie atrakcji.

Zadanie polega na znalezieniu ścieżki o największej wadze w DAG-u.

Programowaniem dynamicznym kolejno dla każdego WC (rosnąco) będziemy obliczać maksymalne wagi ścieżek, które kończą się w atrakcjach o danym WC. Kluczowa operacja to wyznaczanie wyników dla wierzchołków o WC równym $a+1$ na podstawie wierzchołków o WC równym $a$. Chcemy to zrobić w czasie $O(N_a + N_{a+1})$ (gdzie $N_i$ to liczba wierzchołków o WC równym $i$), wtedy całe rozwiązanie będzie działać w $O(nm)$.

# Biuro podróży

Pomiędzy każdymi dwiema atrakcjami przechodzimy zawsze na jeden z czterech „skosów”; rozpatrujemy każdy z tych kierunków niezależnie.
Ustalmy pewien kierunek, powiedzmy „w prawo i do góry”.
Chcemy teraz obliczyć wyniki dla wszystkich pól o WC równym $a+1$ na podstawie wyników dla pól o WC $a$, przy założeniu że przemieszczamy się tylko w prawo i do góry.

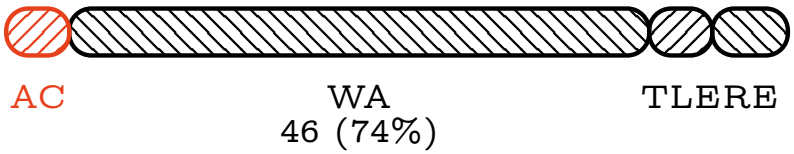
- budujemy tablicę zawierającą jedne i drugie pola i sortujemy je rosnąco po *sumie* ich współrzędnych,
- za pomocą prostej pętli możemy dla każdego punktu o WC $a+1$ znaleźć najlepszy dla niego punkt o WC $a$ — przeglądamy kolejne pola i utrzymujemy aktualnie najlepsze pole o WC równym $a$.

W ten sposób znajdziemy optymalne sposoby dojścia do pól, do których w ostatnim kroku idziemy w prawo i w górę. Co więcej, dla wszystkich innych pól uzyskany wynik będzie na pewno nie większy niż wynik prawidłowy.

# Automat

Autor zadania: Jakub Pachocki

Zgłoszenia: 62 z 775 (8%)
Zaakceptowane przez 5 z 54 drużyn (9%)



# Automat

Jak już ustalimy, które rodzaje batonów chcemy kupić, należy je kupować od najmniejszych rodzajów.

Rozwiązanie opiera się na programowaniu dynamicznym:

- Tablica $zysk[i, l, k]$ określa, jaki największy zysk można uzyskać z $i$ pierwszych rodzajów batonów, jeśli wydamy na nie kwotę $k$, a także kupimy $l$ batonów o numerach wyższych.
- Aby obliczyć $zysk[i+1, \cdot, \cdot]$ z $zysk[i, \cdot, \cdot]$, iterujemy po liczbie kupionych batonów rodzaju $i+1$.

Złożoność czasowa to $O(nkl^2)$. Dla danych z zadania jest to około $40^3 \cdot 64000 \approx 4 \cdot 10^9$, czyli trochę za dużo.

# Automat

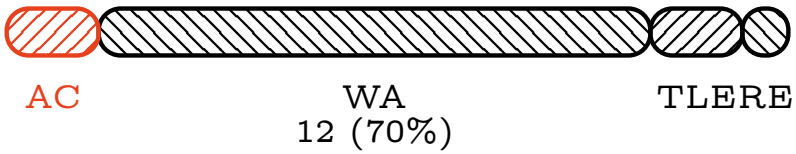
Rozwiązanie jest za wolne. Mamy dwie możliwości:

- Przyspieszyć obliczanie wartości dla stanów (da się je obliczać w zamortyzowanym czasie stałym).
- *Zmniejszyć liczbę stanów.* Zauważmy, że aby wykupić cały automat, nie potrzebujemy wcale wydawać $k = nlc$. Wystarczy nam kwota $k = lc$ — wykupujemy od największych rodzajów. Zatem tak naprawdę mamy $k \approx 40^2$.

# Ewaluacja

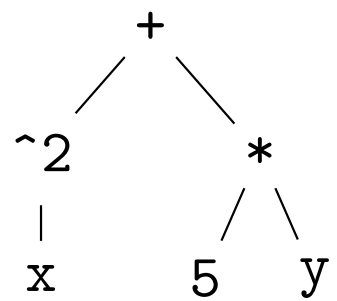
Autor zadania: Jakub Radoszewski

Zgłoszenia: 17 z 775 (2%)
Zaakceptowane przez 2 z 54 drużyn (3%)



# Ewaluacja

Na początku parsujemy wyrażenie i budujemy drzewo wyrażenia. Jest to o tyle proste, że wszystkie elementy wyrażenia są w nawiasach, a zmienne i stałe są jednoznakowe. Czas $O(n)$.
Na przykład dla wyrażenia `((x^2)+(5*y))` mamy drzewo




Teraz w każdym węźle drzewa chcemy obliczyć ciąg $a[0], \ldots, a[p-1]$, w którym $a[i]$ oznacza liczbę sposobów uzyskania reszty $i$ z podwyrażenia odpowiadającego temu węzłowi. Istotnie korzystamy z faktu, że każda zmienna występuje w wyrażeniu co najwyżej raz.

# Ewaluacja

I tak:

- stałej $d$ odpowiada ciąg $a[i] = \begin{cases} 1 & \text{dla } i = d \bmod p, \\ 0 & \text{dla } i \neq d \bmod p, \end{cases}$
- zmiennej $x$ odpowiada ciąg $a[i] = 1$ dla wszystkich $i$,
- wyrażeniu $a + b$ odpowiada ciąg $c$, taki że $c[k] = \left(\sum_{i+j \equiv k \pmod p} a[i] \cdot b[j]\right) \bmod p$,
- wyrażeniu $a \cdot b$ odpowiada ciąg $c$, taki że $c[k] = \left(\sum_{i \cdot j \equiv k \pmod p} a[i] \cdot b[j]\right) \bmod p$,
- wyrażeniu $a^b$ odpowiada ciąg $c$, taki że $c[k] = \left(\sum_{i^b \equiv k \pmod p} a[i]\right) \bmod p$.

Rozwiązanie oparte na podanej metodzie działa w czasie $O(np^2)$ ze względu na operacje dodawania i mnożenia.

# Ewaluacja

Wystarczy przyspieszyć dodawania i mnożenia.
Dodawanie możemy zaimplementować za pomocą mnożenia wielomianów o współczynnikach $a[i]$ i $b[j]$, co robimy w czasie $O(p \log p)$ za pomocą FFT.

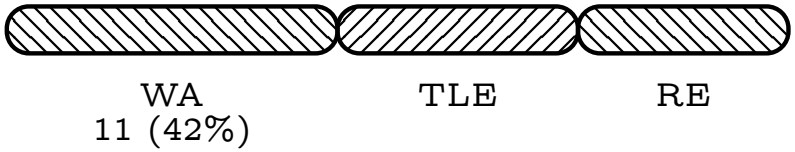
Mnożenie możemy wykonać tak samo, musimy jednak osobno obliczyć $c[0]$ w czasie $O(p)$, a osobno resztę, wybierając dowolny generator $g$ grupy $\mathbb{Z}_p^*$ i traktując mnożenie liczb $i = g^{i'}$ oraz $j = g^{j'}$ jako dodawanie wykładników $i'$ oraz $j'$ modulo $p - 1$.
Generator możemy wyznaczyć brutalnie, sprawdzając każdą kolejną wartość $1, 2, 3, \ldots$ i podnosząc ją do wszystkich możliwych potęg.

Ostatecznie czas $O(np \log p)$.

# Generał kontra gawiedź

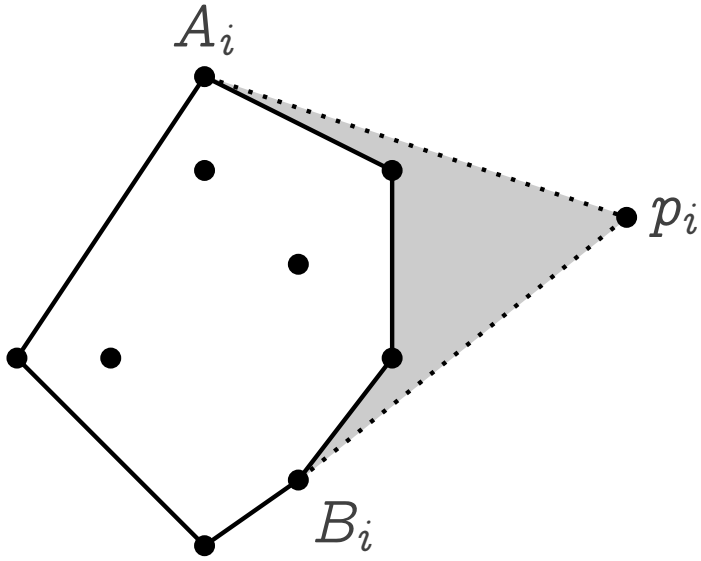
Autor zadania: Tomasz Idziaszek

Zgłoszenia: 26 z 775 (3%)
Zaakceptowane przez 0 z 54 drużyn (0%)



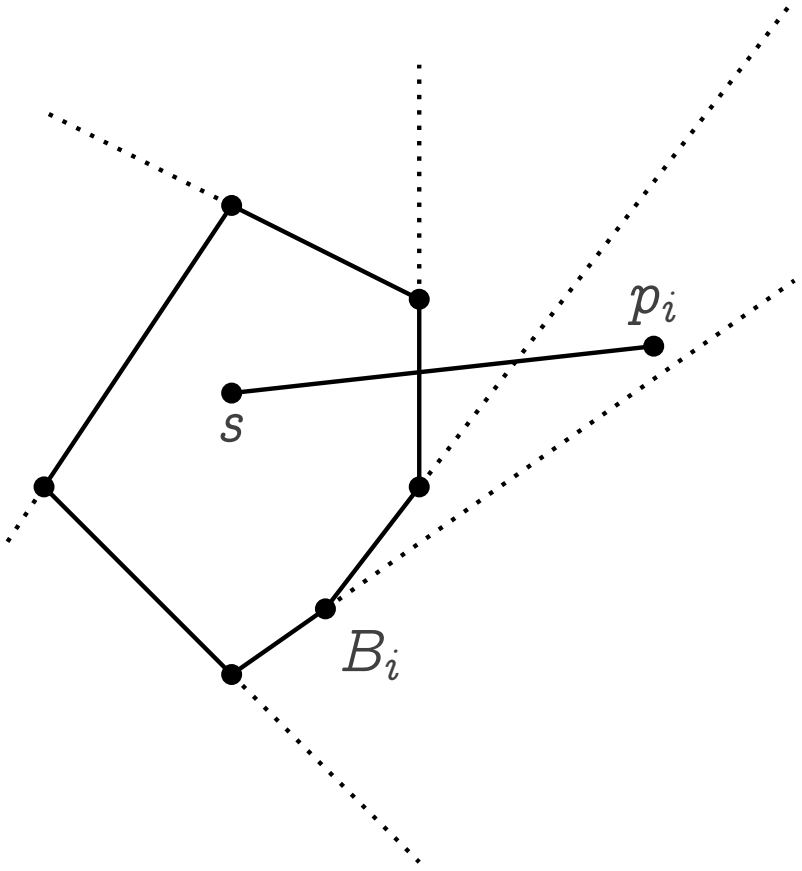
# Generał kontra gawiedź

Obszar chroniony przez żołnierzy jest wypukłą otoczką punktów, w których stoją żołnierze.
Dla każdego z $m$ nieobsadzonych punktów $p_i$ chcemy wyznaczyć pole otoczki wypukłej dla $n$ obsadzonych punktów i punktu $p_i$.




Wyznaczamy wypukłą otoczkę dla $n$ obsadzonych punktów. Wyznaczamy dla każdego punktu $p_i$ punkty $A_i$ oraz $B_i$. Mając obliczone sumy częściowe dla pola wielokąta, wyznaczamy chronione pole w czasie $O(1)$.

# Generał kontra gawiedź




Aby znaleźć $B_i$ dla punktu $p_i$, chcemy wiedzieć, do którego z obszarów wyznaczonych przez półproste przedłużające boki wielokąta należy $p_i$.

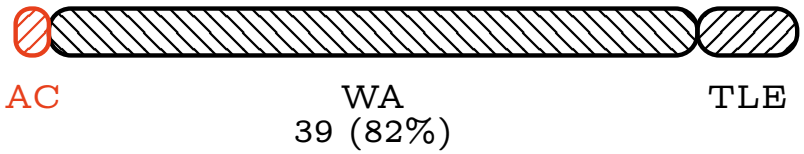
Zamiatamy wszystkie punkty kątowo względem jakiegoś punktu $s$ wewnątrz wypukłej otoczki.

Teraz dla każdego odcinka $sp_i$ znamy bok wielokąta, który go przecina. Aby wyznaczyć obszar, wyszukujemy binarnie, sprawdzając, która półprosta jako ostatnia przecina odcinek $sp_i$. Czas działania $O((n+m)\log(n+m))$.

# Formuła 1

Autor zadania: Tomasz Idziaszek

Zgłoszenia: 47 z 775 (6%)
Zaakceptowane przez 2 z 54 drużyn (3%)



# Formuła 1

Czy istnieje wyścig, w którym startuje $n$ samochodów
i samochód startujący z $i$-tego miejsca wykona $a_i$ wyprzedzań?

Przykład: $a_A = 1, a_B = 3, a_C = 1$.

| | | |
|---|---|---|
| A 1 | B 3 | C 1 |
| A 1 | C 0 | B 3 |
| A 1 | B 2 | C 0 |
| B 1 | A 1 | C 0 |
| A 0 | B 1 | C 0 |
| B 0 | A 0 | C 0 |

Kolejność wyprzedzania: $C, B, B, A, B$.

# Formuła 1

Niech $s_k$ będzie maksymalną liczbą wyprzedzań, które może wykonać $k$-ty samochód, pod warunkiem, że wszystkie pozostałe samochody wykonają zadaną liczbę wyprzedzań. Wyznaczamy $s_k = A_k + B_k$, osobno rozpatrując samochody, które startują przed i za $k$-tym samochodem.

- dla samochodów przed łatwo pokazać, że

$$A_k = \sum_{i=1}^{k-1} (a_i + 1),$$

- aby samochód $k$-ty mógł wyprzedzić samochody startujące za nim, musi poczekać, aż one go wyprzedzą najpierw, co nie zawsze będzie możliwe.

# Formuła 1

Niech $b_i$ będzie minimalną liczbą wyprzedzań, które musi wykonać $i$-ty samochód, aby znaleźć się tuż za samochodem $k$-tym, pod warunkiem, że samochody pomiędzy nimi wyprzedzą samochód $k$-ty, jeśli mogą. Mamy:

$$b_{k+1} = 0, \qquad b_{i+1} = b_i + [a_i - b_i \leqslant 0],$$

gdyż samochód $i+1$ musi wykonać tyle samo wyprzedzań co $i$-ty, plus dodatkowo musi wyprzedzić samochód $i$-ty, jeśli ten ma za małe $a_i$, żeby wyprzedzić samochód $k$-ty.
Sumarycznie:

$$B_k = \sum_{i=k+1}^{n} \max(0, a_i - b_i).$$

Możemy wyznaczyć jedno $s_k$ w czasie $O(n)$.

# Formuła 1

Twierdzenie (niełatwe): wyścig da się zrealizować wtedy i tylko wtedy, gdy dla wszystkich $k$ mamy $s_k \geqslant a_k$.

- Daje nam to algorytm o czasie $O(n^2)$.
- Uważna analiza wzorów pozwala nam wyliczać kolejne $s_k$ w sumarycznym czasie $O(n \log n)$ na drzewie przedziałowym.

Da się lepiej:
Rozważmy *ostatni* samochód $m$, dla którego $A_m \leqslant a_m$, czyli albo „zużywa” on wszystkie wyprzedzania samochodów, które są przed nim, albo musi go wyprzedzić jakiś samochód, który startował za nim.
Twierdzenie: dla wszystkich $k \neq m$ mamy $s_k \geqslant a_k$.

- Zatem wyznaczamy tylko $s_m$, co daje algorytm $O(n)$.

KONIEC