POTYCZKI ALGORYTMICZNE 2024

OMÓWIENIE ZADAŃ

DIGITAL TECHNOLOGY POLAND

UNIWERSYTET WARSZAWSKI

FRD

RTB HOUSE

HUAWEI

C. Podciągi DNA

Najszybsze rozwiązanie: **Adam Gąsienica-Samek (4:49)**

Podciągi DNA

Treść

Alfabet: `ACGT`.

Słowo s jest *k-bogate*, gdy zawiera wszystkie k-literowe słowa nad `ACGT` jako podciągi.

Podciągi DNA

Treść

Alfabet: `ACGT`.

Słowo s jest *k-bogate*, gdy zawiera wszystkie k-literowe słowa nad `ACGT` jako podciągi.

Wejście: słowo s długości $n = |s|$ ($n \leq 200\,000$).

Podciągi DNA

Treść

Alfabet: `ACGT`.

Słowo s jest *k-bogate*, gdy zawiera wszystkie k-literowe słowa nad `ACGT` jako podciągi.

Wejście: słowo s długości $n = |s|$ ($n \leq 200\,000$).

Wyjście: dla każdego $k \in [1, n]$: Ile znaków trzeba zmienić w s, by stało się k-bogate? Czy w ogóle się da?

Podciągi DNA

Czy słowo s jest k-bogate?

TTCTGAAGATAGCTAGATTAACGGC

Podciągi DNA

Czy słowo s jest k-bogate?

TTCTGAAGATAGCTAGATTAACGGC

Podciągi DNA

Czy słowo s jest k-bogate?

TTCTGA|AGATAGC|TAGATTAACGGC

Podciągi DNA

Czy słowo s jest k-bogate?

Podciągi DNA

Czy słowo s jest k-bogate?

$k = 3 \quad \Rightarrow \quad$ TAK

Podciągi DNA

Czy słowo s jest k-bogate?

TTCTGA|AGATAGC|TAGATTAAC|GGC

$k = 3 \quad \Rightarrow \quad$ TAK

$k = 4 \quad \Rightarrow \quad$ NIE, słowo ACCT

Podciągi DNA

Czy słowo s jest k-bogate?

$k = 3 \quad \Rightarrow \quad$ TAK

$k = 4 \quad \Rightarrow \quad$ NIE, słowo ACCT

Koszt sprawienia, by słowo s było k-bogate

Podciągi DNA

Czy słowo s jest k-bogate?

$k = 3 \quad \Rightarrow \quad$ TAK

$k = 4 \quad \Rightarrow \quad$ NIE, słowo ACCT

Koszt sprawienia, by słowo s było k-bogate

$k > \frac{n}{4} \quad \Rightarrow$ nie da się.

Podciągi DNA

Czy słowo s jest k-bogate?

$$\boxed{\text{TTCTGA}}\boxed{\text{AGATAGC}}\boxed{\text{TAGATTAAC}}\text{GGC}$$

$k = 3 \quad \Rightarrow \quad$ TAK

$k = 4 \quad \Rightarrow \quad$ NIE, słowo ACCT

Koszt sprawienia, by słowo s było k-bogate

$k > \frac{n}{4} \quad \Rightarrow$ nie da się.

$k \leq \frac{n}{4} \quad \Rightarrow$ minimalny koszt podziału s na podsłowa długości ≥ 4.

(Koszt podsłowa = # brakujących literek, koszt podziału = suma kosztów podsłów.)

Koszt sprawienia, by słowo s było k-bogate

$k \leq \frac{n}{4} \quad \Rightarrow$ minimalny koszt podziału s na podsłowa długości ≥ 4.

(Koszt podsłowa = # brakujących literek, koszt podziału = suma kosztów podsłów.)

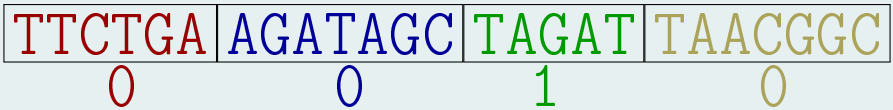
TTCTGAAGATAGCTAGATTAACGGC

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

$k \leq \frac{n}{4} \quad \Rightarrow$ minimalny koszt podziału s na podsłowa długości ≥ 4.

(Koszt podsłowa = # brakujących literek, koszt podziału = suma kosztów podsłów.)



Wynik dla $k = 4$ to 1.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

$k \leq \frac{n}{4} \quad \Rightarrow$ minimalny koszt podziału s na podsłowa długości ≥ 4.

(Koszt podsłowa = # brakujących literek, koszt podziału = suma kosztów podsłów.)

TTCTGA	AGATAGC	TAGAT	TAACGGC
0	0	1	0

Wynik dla $k = 4$ to 1.

TTCTGA	AGAT	AGCT	AGAT	TAACGGC
0	1	0	1	0

Wynik dla $k = 5$ to 2.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.
- **Tracisz 1 monetę** za brakującą literkę `A`/`C`/`G`/`T` w podsłowie.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.
- **Tracisz 1 monetę** za brakującą literkę `A`/`C`/`G`/`T` w podsłowie.

Zgromadź jak najwięcej monet.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.
- **Tracisz 1 monetę** za brakującą literkę `A`/`C`/`G`/`T` w podsłowie.

Zgromadź jak najwięcej monet.

Nietrudne ćwiczenie: rozwiązać podproblem w $\mathcal{O}(n)$.

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.
- **Tracisz 1 monetę** za brakującą literkę `A`/`C`/`G`/`T` w podsłowie.

Zgromadź jak najwięcej monet.

Nietrudne ćwiczenie: rozwiązać podproblem w $\mathcal{O}(n)$. (Dość prosty dynamik.)

Podciągi DNA

Koszt sprawienia, by słowo s było k-bogate

Funkcja kosztu dla $0 \leq k \leq \frac{n}{4}$ wypukła! $\Rightarrow$ *parametric search*.

Podproblem do rozwiązania w *parametric searchu*

Podziel s na podsłowa długości ≥ 4.

- **Otrzymujesz X monet** za każde podsłowo.
- **Tracisz 1 monetę** za brakującą literkę `A`/`C`/`G`/`T` w podsłowie.

Zgromadź jak najwięcej monet.

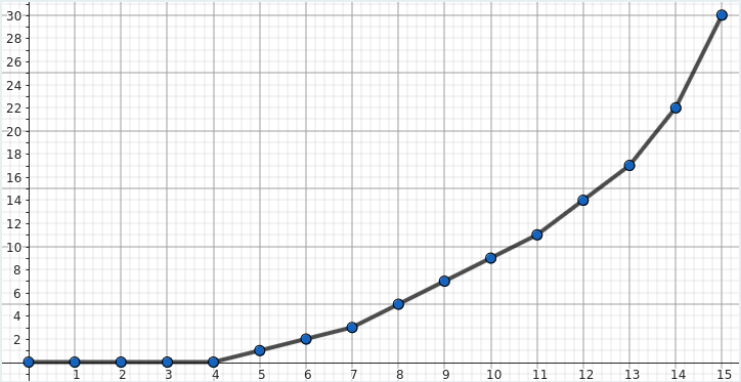
Nietrudne ćwiczenie: rozwiązać podproblem w $\mathcal{O}(n)$. (Dość prosty dynamik.)

Standardowy *parametric search* $\Rightarrow$ Dla jednego k rozwiązanie w $\mathcal{O}(n \log n)$.

Podciągi DNA

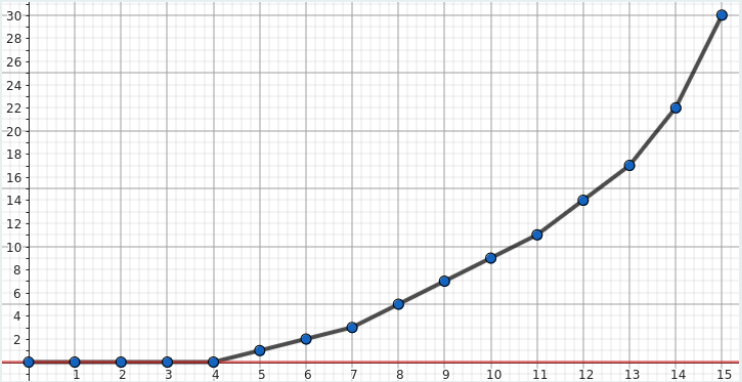
Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



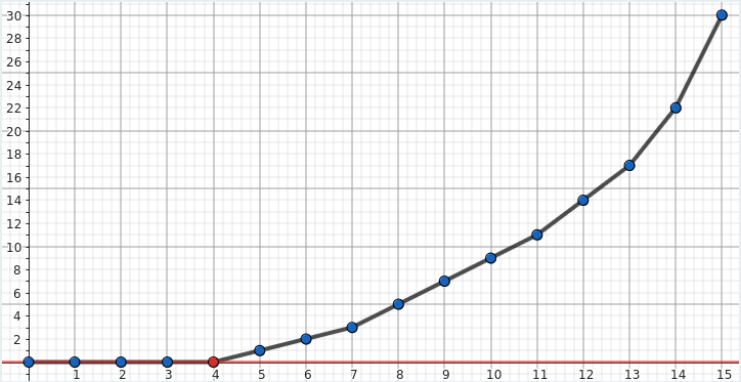
$\boxed{X = 0}$

Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$\boxed{X = 0}$

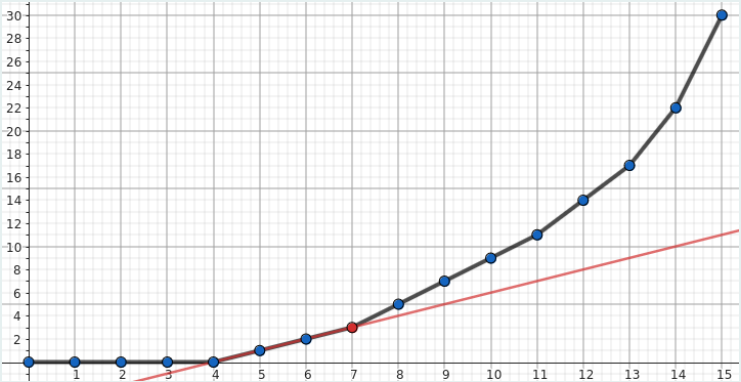
Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$\boxed{X = 1}$

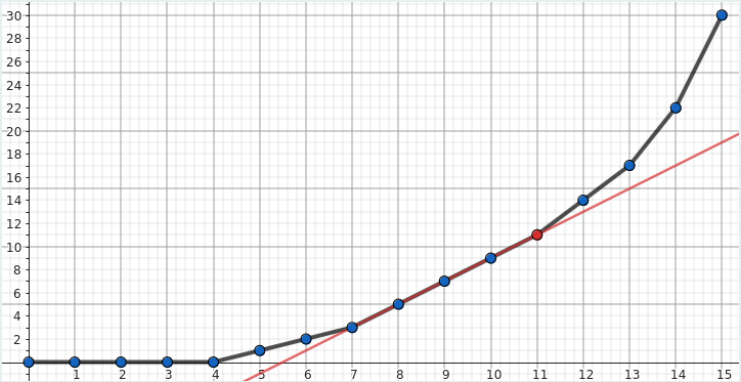
Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$\boxed{X = 2}$

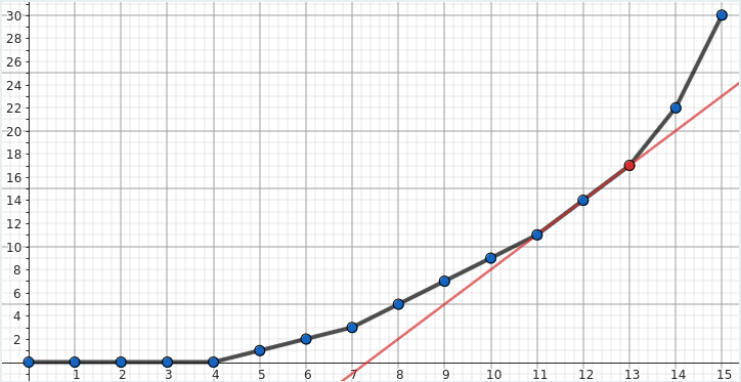
Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$\boxed{X = 3}$

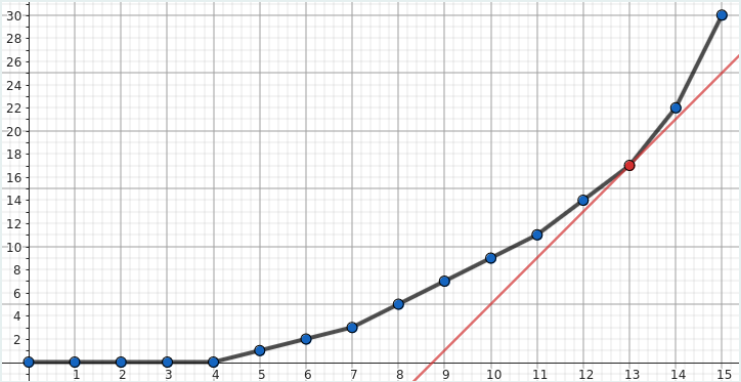
Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$\boxed{X = 4}$

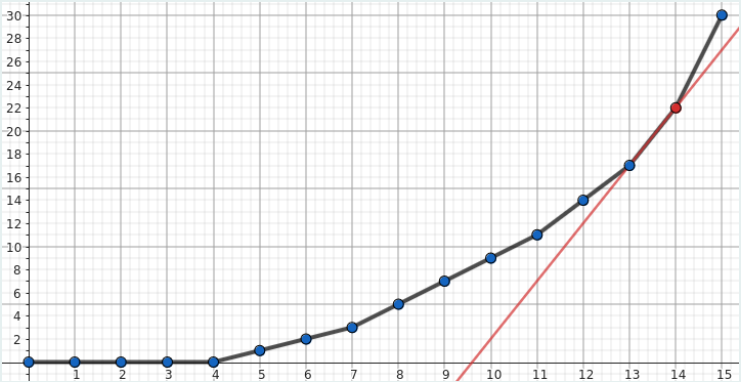
Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



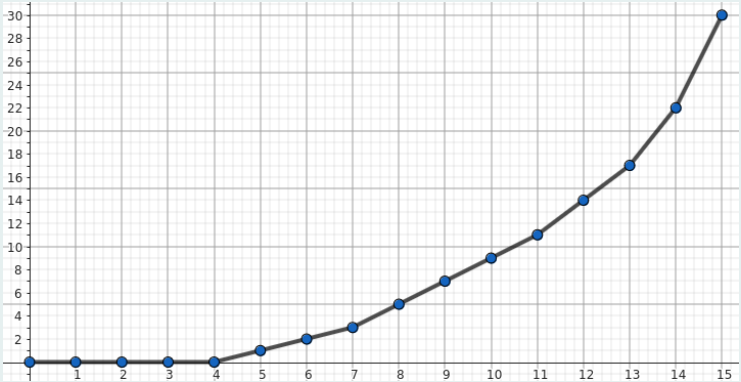
$\boxed{X = 5}$

Podzadanie dla nagrody X monet za podsłowo = styczna do f o nachyleniu X.

Dodatkowo: max na prawo punkt styczności.

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.



$f(0) = 0$, $f(\frac{n}{4})$ uzyskujemy z pomocą podzadania dla $X = n$.

Podciągi DNA

Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

- Uruchom podzadanie dla $X = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor$.

Podciągi DNA

Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

- Uruchom podzadanie dla $X = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor$.
- **If** punkt (x_2, y_2) leży na stycznej $\Rightarrow$ ukryta funkcja jest liniowa!

Podciągi DNA

Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

- Uruchom podzadanie dla $X = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor$.
- **If** punkt (x_2, y_2) leży na stycznej $\Rightarrow$ ukryta funkcja jest liniowa!
- **Else** niech punkt $(x, y) = \max$ na prawo punkt styczności.

Podciągi DNA

Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

- Uruchom podzadanie dla $X = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor$.
- **If** punkt (x_2, y_2) leży na stycznej $\Rightarrow$ ukryta funkcja jest liniowa!
- **Else** niech punkt $(x, y) =$ max na prawo punkt styczności.
  Mamy $x_1 < x < x_2$.

Podciągi DNA

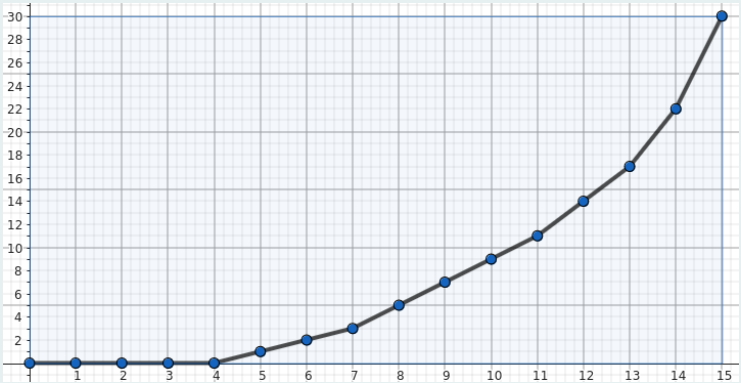
Rekurencja

Ukryta funkcja wypukła $f : \{x_1, \ldots, x_2\} \to \{y_1, \ldots, y_2\}$. Mamy $f(x_1) = y_1$, $f(x_2) = y_2$. Odzyskaj f.

- Uruchom podzadanie dla $X = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \right\rfloor$.
- **If** punkt (x_2, y_2) leży na stycznej $\Rightarrow$ ukryta funkcja jest liniowa!
- **Else** niech punkt $(x, y) =$ max na prawo punkt styczności.
  Mamy $x_1 < x < x_2$.
  Rekurencja dla $f : \{x_1, \ldots, x\} \to \{y_1, \ldots, y\}$.
  Rekurencja dla $f : \{x, \ldots, x_2\} \to \{y, \ldots, y_2\}$.

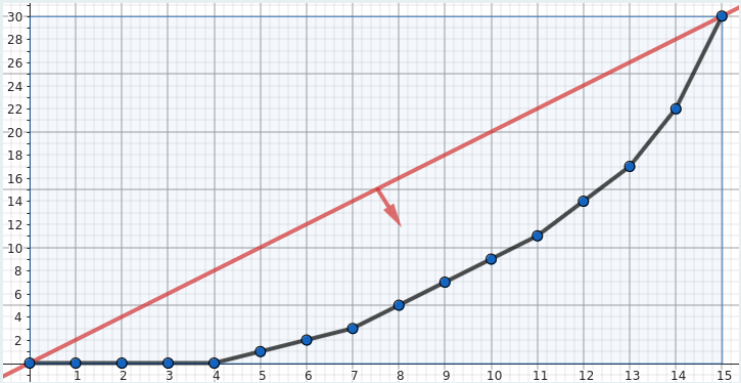
Rekurencja – przykład

Ukryta funkcja wypukła $f : \{0, \ldots, 15\} \to \{0, \ldots, 30\}$. Mamy $f(0) = 0$, $f(15) = 30$.



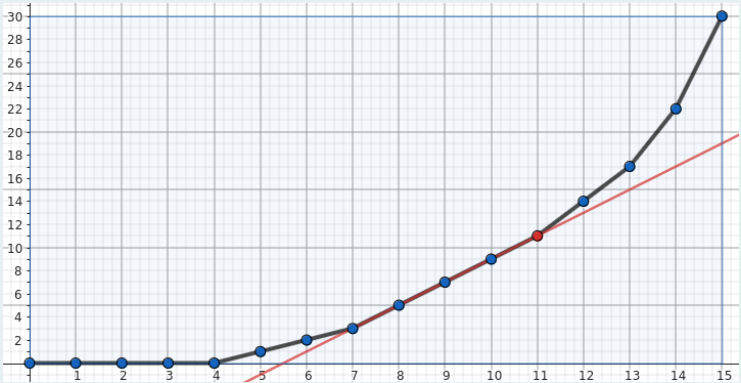
Rekurencja – przykład

Ukryta funkcja wypukła $f : \{0, \ldots, 15\} \to \{0, \ldots, 30\}$. Mamy $f(0) = 0$, $f(15) = 30$.



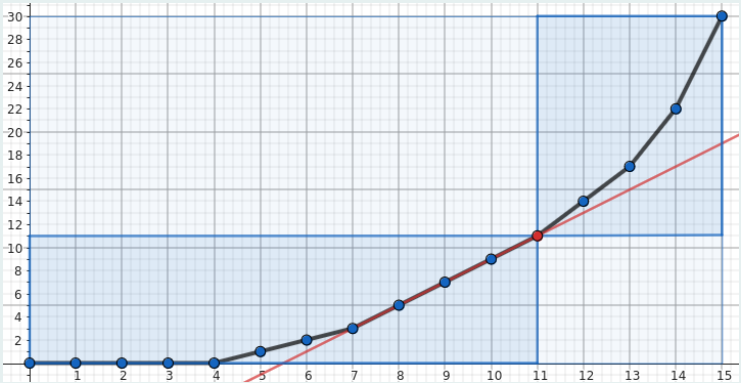
Rekurencja – przykład

Ukryta funkcja wypukła $f : \{0, \ldots, 15\} \to \{0, \ldots, 30\}$. Mamy $f(0) = 0$, $f(15) = 30$.



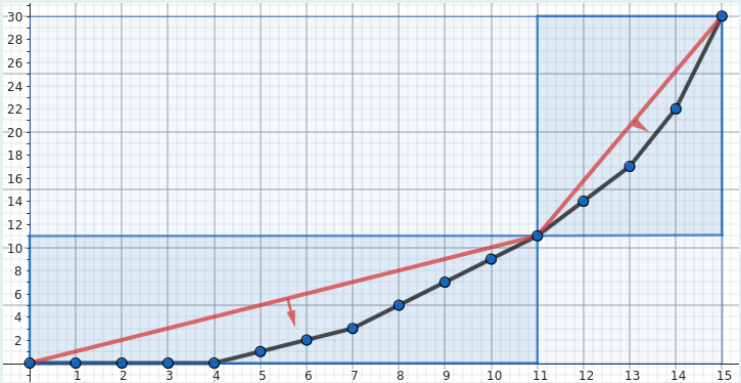
Rekurencja – przykład

Ukryta funkcja wypukła $f : \{0, \ldots, 15\} \to \{0, \ldots, 30\}$. Mamy $f(0) = 0$, $f(15) = 30$.



Rekurencja – przykład

Ukryta funkcja wypukła $f : \{0, \ldots, 15\} \to \{0, \ldots, 30\}$. Mamy $f(0) = 0$, $f(15) = 30$.



Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Alternatywne rozwiązanie

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Alternatywne rozwiązanie

- Piszemy kwadratowego dynamika $\mathrm{dp}[n, k]$.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Alternatywne rozwiązanie

- Piszemy kwadratowego dynamika $\mathrm{dp}[n, k]$.
- **Obserwacja:** $\forall_{n' \leq n}$ funkcja $\mathrm{dp}[n', x]$ jest wypukła dla $x \in [0, \frac{n'}{4}]$!

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Alternatywne rozwiązanie

- Piszemy kwadratowego dynamika $\mathrm{dp}[n, k]$.
- **Obserwacja:** $\forall_{n' \leq n}$ funkcja $\mathrm{dp}[n', x]$ jest wypukła dla $x \in [0, \frac{n'}{4}]$!
- $\Rightarrow$ Kompresujemy wszystkie funkcje $\mathrm{dp}[n', x]$ do postaci $\mathcal{O}(\sqrt{n})$ odcinków.

Podciągi DNA

Rozwiązanie dla wszystkich $0 \leq k \leq \frac{n}{4}$

Ukryta funkcja wypukła $f : \{0, \ldots, \frac{n}{4}\} \to \{0, \ldots, n\}$.

Każdy krok rekurencji: czas $\mathcal{O}(n)$ i **odkrywa nowy punkt otoczki**!

Obserwacja: f ma maksymalnie $\mathcal{O}(\sqrt{n})$ punktów na otoczce.

Wniosek: ostateczna złożoność $\mathcal{O}(n\sqrt{n})$.

Alternatywne rozwiązanie

- Piszemy kwadratowego dynamika $\mathrm{dp}[n, k]$.
- **Obserwacja:** $\forall_{n' \leq n}$ funkcja $\mathrm{dp}[n', x]$ jest wypukła dla $x \in [0, \frac{n'}{4}]$!
- $\Rightarrow$ Kompresujemy wszystkie funkcje $\mathrm{dp}[n', x]$ do postaci $\mathcal{O}(\sqrt{n})$ odcinków.
- Trochę męczarni $\Rightarrow$ inne rozwiązanie w $\mathcal{O}(n\sqrt{n})$.

D. Giełda paprotek

Najszybsze rozwiązanie: **Yahor Dubovik (4:15)**

Giełda paprotek

Treść

Ciąg liczb całkowitych $a_1, a_2, \ldots a_n$ oznacza koszty sprzedaży oraz kupna paprotek w kolejne dni.

Giełda paprotek

Treść

Ciąg liczb całkowitych $a_1, a_2, \ldots a_n$ oznacza koszty sprzedaży oraz kupna paprotek w kolejne dni.
Przez $f(a)$ oznaczamy maksymalny możliwy zysk z handlu paprotkami, jeśli każdego dnia możemy sprzedać lub kupić co najwyżej jedną paprotkę, liczba posiadanych paprotek nie może spaść poniżej zera oraz wszystkie koszty znamy zawczasu.

Giełda paprotek

Treść

Ciąg liczb całkowitych $a_1, a_2, \ldots a_n$ oznacza koszty sprzedaży oraz kupna paprotek w kolejne dni.
Przez $f(a)$ oznaczamy maksymalny możliwy zysk z handlu paprotkami, jeśli każdego dnia możemy sprzedać lub kupić co najwyżej jedną paprotkę, liczba posiadanych paprotek nie może spaść poniżej zera oraz wszystkie koszty znamy zawczasu.
Przez $g(n)$ oznaczamy sumę po $f(p)$, gdzie jako p podstawiamy wszystkie permutacje n-elementowe.

Giełda paprotek

Treść

Ciąg liczb całkowitych $a_1, a_2, \ldots a_n$ oznacza koszty sprzedaży oraz kupna paprotek w kolejne dni.
Przez $f(a)$ oznaczamy maksymalny możliwy zysk z handlu paprotkami, jeśli każdego dnia możemy sprzedać lub kupić co najwyżej jedną paprotkę, liczba posiadanych paprotek nie może spaść poniżej zera oraz wszystkie koszty znamy zawczasu.
Przez $g(n)$ oznaczamy sumę po $f(p)$, gdzie jako p podstawiamy wszystkie permutacje n-elementowe. Należy obliczyć $g(1), g(2), \ldots, g(k)$ dla zadanego k.

Giełda paprotek

Klasyczny problem statyczny

Giełda paprotek

Klasyczny problem statyczny

Dla zadanego statycznego ciągu kosztów iterujemy się po nim utrzymując kolejkę priorytetową.

Giełda paprotek

Klasyczny problem statyczny

Dla zadanego statycznego ciągu kosztów iterujemy się po nim utrzymując kolejkę priorytetową. Mijając koszt x wrzucamy liczbę x na kolejkę priorytetową, następnie usuwamy z niej minimum, a następnie wrzucamy x jeszcze raz.

Giełda paprotek

Klasyczny problem statyczny

Dla zadanego statycznego ciągu kosztów iterujemy się po nim utrzymując kolejkę priorytetową. Mijając koszt x wrzucamy liczbę x na kolejkę priorytetową, następnie usuwamy z niej minimum, a następnie wrzucamy x jeszcze raz.

Czy to się przyda?

Giełda paprotek

Klasyczny problem statyczny

Dla zadanego statycznego ciągu kosztów iterujemy się po nim utrzymując kolejkę priorytetową. Mijając koszt x wrzucamy liczbę x na kolejkę priorytetową, następnie usuwamy z niej minimum, a następnie wrzucamy x jeszcze raz.

Czy to się przyda?

Raczej nie...

Giełda paprotek

Klasyczny problem statyczny

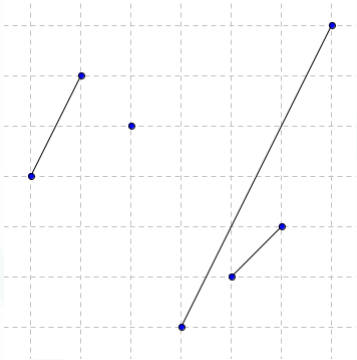
Dla zadanego statycznego ciągu kosztów iterujemy się po nim utrzymując kolejkę priorytetową. Mijając koszt x wrzucamy liczbę x na kolejkę priorytetową, następnie usuwamy z niej minimum, a następnie wrzucamy x jeszcze raz.

Czy to się przyda?

Raczej nie... (aczkolwiek przyda się do dowodu poprawności finalnego algorytmu)

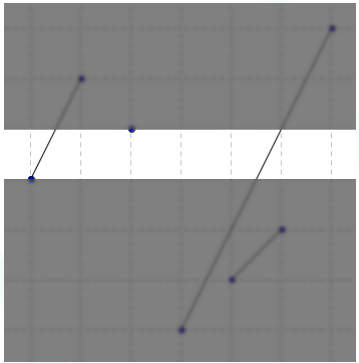
Alternatywne spojrzenie

Dla wybranego x skupmy się na przedziale wartości między x, a $x + 1$.



Alternatywne spojrzenie

Dla wybranego x skupmy się na przedziale wartości między x, a $x + 1$.



Ograniczenie górne

Dla takiego przedziału możemy obliczyć maksymalną liczbę odcinków, które przez niego przejdą.

Ograniczenie górne

Dla takiego przedziału możemy obliczyć maksymalną liczbę odcinków, które przez niego przejdą. Suma tych wartości po wszystkich x będzie ograniczać wynik od góry.

Giełda paprotek

Ograniczenie górne

Dla takiego przedziału możemy obliczyć maksymalną liczbę odcinków, które przez niego przejdą. Suma tych wartości po wszystkich x będzie ograniczać wynik od góry.

Obserwacja

Suma tych wartości jest dokładnie równa wynikowi.

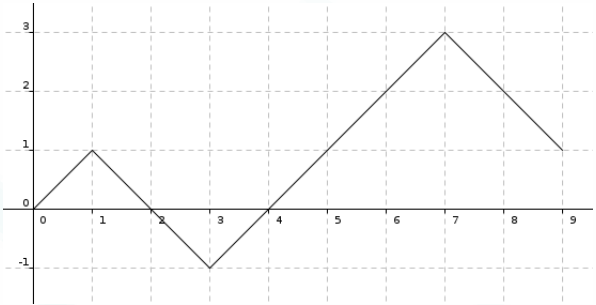
Ciąg binarny

Dla x zmieniamy wartości $\leq x$ w -1 i $> x$ w 1. Każde ułożenie 1 i -1 znajduje się w $x! \cdot (n-x)!$ permutacjach.

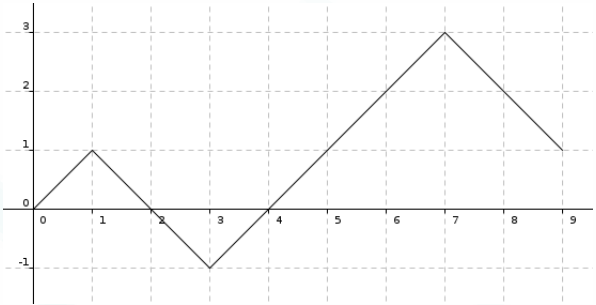
Ciąg binarny

Dla x zmieniamy wartości $\leq x$ w -1 i $> x$ w 1. Każde ułożenie 1 i -1 znajduje się w $x! \cdot (n-x)!$ permutacjach.
Zależnie od ułożenia możemy prosto wyrazić maksymalną liczbę odcinków, które mogą przeciąć przedział, rysując wykres sum prefiksowych ciągu.

Giełda paprotek



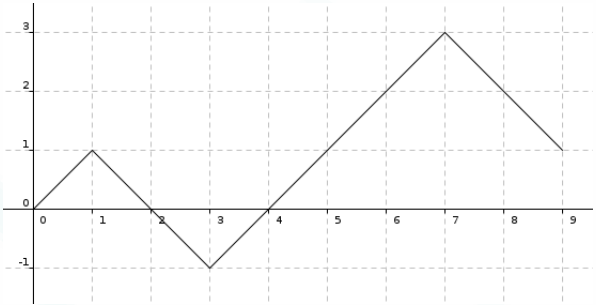
Giełda paprotek



Prosty wzorek

Maksymalna liczba odcinków to $n - x$ (liczba jedynek) zmniejszone o maksymalną sumę prefiksową ciągu.

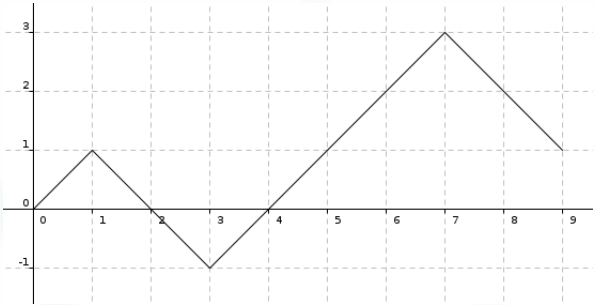
Giełda paprotek



Prosty wzorek

Maksymalna liczba odcinków to $n - x$ (liczba jedynek) zmniejszone o maksymalną sumę prefiksową ciągu. Dla ustalonego x chcemy posumować $n - x - max_pref$ po wszystkich ciągach z $x - 1$ i $(n - x)$ 1.

Giełda paprotek



Prosty wzorek

Maksymalna liczba odcinków to $n - x$ (liczba jedynek) zmniejszone o maksymalną sumę prefiksową ciągu. Dla ustalonego x chcemy posumować $n - x - max_pref$ po wszystkich ciągach z $x - 1$ i $(n - x)$ 1. Po drobnym przekształceniu interesuje nas jedynie suma po maksymalnych sumach prefiksowych.

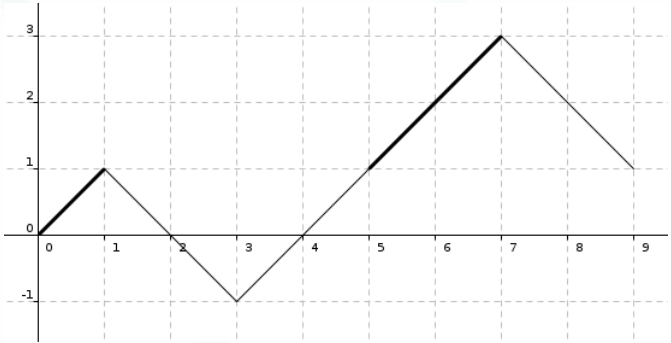
Giełda paprotek

Maksymalna suma prefiksowa

Maksymalna suma prefiksowa to liczba prefiksów, których sumy są większe niż sumy wszystkich prefiksów przed nimi.

Maksymalna suma prefiksowa

Maksymalna suma prefiksowa to liczba prefiksów, których sumy są większe niż sumy wszystkich prefiksów przed nimi.



Giełda paprotek

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu.

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu. Obracając rysunek o 180° chcemy obliczać ciągi, których suma prefiksowa nigdy nie spada do zera.

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu. Obracając rysunek o 180° chcemy obliczać ciągi, których suma prefiksowa nigdy nie spada do zera.
Posłuży do tego $DP_1[x][y]$, które po inicjalizacji $DP_1[1][0] = 1$ jest prosto obliczyć ze wzoru $DP_1[x][y] = DP_1[x-1][y] + DP_1[x][y-1]$.

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu. Obracając rysunek o 180° chcemy obliczać ciągi, których suma prefiksowa nigdy nie spada do zera.
Posłuży do tego $DP_1[x][y]$, które po inicjalizacji $DP_1[1][0] = 1$ jest prosto obliczyć ze wzoru $DP_1[x][y] = DP_1[x-1][y] + DP_1[x][y-1]$.
Należy pilnować, by zawsze zachodziło $x > y$.

Giełda paprotek

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu. Obracając rysunek o 180° chcemy obliczać ciągi, których suma prefiksowa nigdy nie spada do zera.
Posłuży do tego $DP_1[x][y]$, które po inicjalizacji $DP_1[1][0] = 1$ jest prosto obliczyć ze wzoru $DP_1[x][y] = DP_1[x-1][y] + DP_1[x][y-1]$.
Należy pilnować, by zawsze zachodziło $x > y$.

Drugi dynamik

Możemy teraz dla x i y obliczyć sumę po maksimach prefiksowych ciągów z x jedynkami i y minus jedynkami.

Pierwszy dynamik

Chcemy dla każdej liczby jedynek (x) i minus jedynek (y) obliczyć liczbę ciągów, których suma jest większa niż suma ich dowolnego prefiksu. Obracając rysunek o 180° chcemy obliczać ciągi, których suma prefiksowa nigdy nie spada do zera.
Posłuży do tego $DP_1[x][y]$, które po inicjalizacji $DP_1[1][0] = 1$ jest prosto obliczyć ze wzoru $DP_1[x][y] = DP_1[x-1][y] + DP_1[x][y-1]$.
Należy pilnować, by zawsze zachodziło $x > y$.

Drugi dynamik

Możemy teraz dla x i y obliczyć sumę po maksimach prefiksowych ciągów z x jedynkami i y minus jedynkami.
Oznaczając ją przez $DP_2[x][y]$ obliczamy ją ze wzoru
$DP_2[x][y] = DP_2[x-1][y] + DP_2[x][y-1] + DP_1[x][y]$.

Cały algorytm

Opisane programowanie dynamiczne działa w czasie $\mathcal{O}(n^2)$ i oblicza potrzebne wartości nie tylko dla n, ale również dla wszystkich mniejszych.

Giełda paprotek

Cały algorytm

Opisane programowanie dynamiczne działa w czasie $\mathcal{O}(n^2)$ i oblicza potrzebne wartości nie tylko dla n, ale również dla wszystkich mniejszych.

Bonus

Stosując odpowiednie przekształcenia wzorów da się obliczyć $g(n)$ w czasie liniowym.

Giełda paprotek

Cały algorytm

Opisane programowanie dynamiczne działa w czasie $\mathcal{O}(n^2)$ i oblicza potrzebne wartości nie tylko dla n, ale również dla wszystkich mniejszych.

Bonus

Stosując odpowiednie przekształcenia wzorów da się obliczyć $g(n)$ w czasie liniowym. Podpowiedź: przyda się do tego odbijanie wykresów względem poziomej prostej, podobnie jak we wzorze na liczby Catalana.

E. Jeden bit

Najszybsze rozwiązanie: **Adam Gąsienica-Samek (0:07)**

Jeden bit

Treść

Zmienić jak najmniejszą liczbę elementów w ciągu tak, żeby każde dwa kolejne elementy różniły się co najwyżej jednym bitem.

Jeden bit

Rozwiązanie kwadratowe

$dp[i]$ - największa liczba elementów, które możemy zostawić niezmienione na prefiksie długości i tak, żeby prefiks ten spełniał założenia zadania oraz dodatkowo ostatni element pozostał niezmieniony.

Jeden bit

Rozwiązanie kwadratowe

$dp[i]$ - największa liczba elementów, które możemy zostawić niezmienione na prefiksie długości i tak, żeby prefiks ten spełniał założenia zadania oraz dodatkowo ostatni element pozostał niezmieniony.

$dp[i] = \max(1, \max_{j<i, popcount(i \oplus j) \leq j-i}(dp[j] + 1))$

Jeden bit

Rozwiązanie kwadratowe

$dp[i]$ - największa liczba elementów, które możemy zostawić niezmienione na prefiksie długości i tak, żeby prefiks ten spełniał założenia zadania oraz dodatkowo ostatni element pozostał niezmieniony.
$dp[i] = \max(1, \max_{j<i, popcount(i \oplus j) \leq j-i}(dp[j] + 1))$
Odpowiedzią jest $n - \max_i(dp[i])$.

Jeden bit

Szybsze rozwiązanie

W optymalnym rozwiązaniu dystans między dwoma kolejnymi niezmienionymi elementami nie przekroczy $2 \log(\textit{zakres})$.

Szybsze rozwiązanie

W optymalnym rozwiązaniu dystans między dwoma kolejnymi niezmienionymi elementami nie przekroczy $2\log(\mathit{zakres})$.
W przeciwnym przypadku można by zostawić środkowy element z przerwy długości $> 2\log(\mathit{zakres})$ niezmieniony.

Jeden bit

Szybsze rozwiązanie

W optymalnym rozwiązaniu dystans między dwoma kolejnymi niezmienionymi elementami nie przekroczy $2\log(zakres)$.
W przeciwnym przypadku można by zostawić środkowy element z przerwy długości $> 2\log(zakres)$ niezmieniony.
Możemy więc dla każdego i rozważać tylko $2\log(zakres)$ poprzednich elementów, co daje złożoność $\mathcal{O}(n\log(zakres))$.

F. Łamigłówka 4

Najszybsze rozwiązanie: **Marek Sommer (0:57)**

Łamigłówka 4

Treść

Dana permutacja $1, 2, \ldots, n$.
Dla sąsiednich a, b ($|a - b| = 1$) można:

- dodać b do a
- odjąć b od a

o ile wynik w przedziale $[1, n]$.

Limity

- długość: $n \leq 30000$
- limit operacji: 2 500 000

Łamigłówka 4

krok I – Dwie jedynki

1. $\ldots, 1, x, \ldots$ (lub $\ldots, x, 1$)

Łamigłówka 4

krok I – Dwie jedynki

1. $\ldots, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 1, x-1, \ldots$

Łamigłówka 4

krok I – Dwie jedynki

1. $\ldots, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 1, x-1, \ldots$
3. $\ldots, 1, x-2, \ldots$

Łamigłówka 4

krok I – Dwie jedynki

1. $\ldots, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 1, x-1, \ldots$
3. $\ldots, 1, x-2, \ldots$
4. $\ldots$
5. $\ldots, 1, 1, \ldots$

Łamigłówka 4

krok I – Dwie jedynki

1. $\ldots, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 1, x-1, \ldots$
3. $\ldots, 1, x-2, \ldots$
4. $\ldots$
5. $\ldots, 1, 1, \ldots$

Liczba operacji $\leq n$

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2.
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 1, 2, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 3, 2, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 3, 5, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 8, 5, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 8, 13, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 13, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3.
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 21, 34, x - 34, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 21, 34, x - 34 - 34, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 21, 13, x - 34 - 34, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 8, 13, x - 34 - 34, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 8, 13, x - 34 - 34 - 13, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 8, 5, x - 34 - 34 - 13, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 3, 5, x - 34 - 34 - 13, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 3, 2, x - 34 - 34 - 13, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 2, x - 34 - 34 - 13, \ldots$
4.

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 2, x - 34 - 34 - 13 - 2, \ldots$
4.

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 1, x - 34 - 34 - 13 - 2, \ldots$
4.

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 1, x - 34 - 34 - 13 - 2, \ldots$
4. $\ldots, 1, 1, 1, \ldots$

Łamigłówka 4

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 1, x - 34 - 34 - 13 - 2, \ldots$
4. $\ldots, 1, 1, 1, \ldots$

Liczba operacji: $\sim 3 \log(x)$

krok II – Trzy jedynki

1. Weźmy $\ldots, 1, 1, x, \ldots$ (lub $\ldots, x, 1$)
2. $\ldots, 21, 34, x, \ldots$
3. $\ldots, 1, 1, x - 34 - 34 - 13 - 2, \ldots$
4. $\ldots, 1, 1, 1, \ldots$

Liczba operacji: $\sim 3 \log(x) \leq 52$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$
5. $\ldots, y, x, 1, 1, \ldots, 1, 1$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$
5. $\ldots, y, x, 1, 1, \ldots, 1, 1$
6. $\ldots, y, 1, 1, 1, \ldots, 1, 1$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$
5. $\ldots, y, x, 1, 1, \ldots, 1, 1$
6. $\ldots, y, 1, 1, 1, \ldots, 1, 1$
7. $\ldots$
8. $1, 1, \ldots, 1, 1$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$
5. $\ldots, y, x, 1, 1, \ldots, 1, 1$
6. $\ldots, y, 1, 1, 1, \ldots, 1, 1$
7. $\ldots$
8. $1, 1, \ldots, 1, 1$

Liczba operacji: $\sum_{x=2}^{n} 3 \log x$

Łamigłówka 4

krok III – n jedynek

1. $\ldots, 1, 1, 1, x, y, \ldots$
2. $\ldots, 1, 1, 1, 1, y, \ldots$
3. $\ldots$
4. $\ldots, 1, 1, \ldots, 1, 1$
5. $\ldots, y, x, 1, 1, \ldots, 1, 1$
6. $\ldots, y, 1, 1, 1, \ldots, 1, 1$
7. $\ldots$
8. $1, 1, \ldots, 1, 1$

Liczba operacji: $\sum_{x=2}^{n} 3 \log x \leq 52 \cdot 30\,000 = 1\,560\,000$

Łamigłówka 4

krok VI – $1, 2, 3, 4, \ldots$

1. $1, 1, 1, 1, 1, 1, \ldots$

Łamigłówka 4

krok VI – $1, 2, 3, 4, \ldots$

1. $1, 1, 1, 1, 1, 1, \ldots$
2. $1, 2, 1, 1, 1, 1, \ldots$

Łamigłówka 4

krok VI – $1, 2, 3, 4, \ldots$

1. $1, 1, 1, 1, 1, 1, \ldots$
2. $1, 2, 1, 1, 1, 1, \ldots$
3. $1, 2, 3, 1, 1, 1, \ldots$

Łamigłówka 4

krok VI – $1, 2, 3, 4, \ldots$

1. $1, 1, 1, 1, 1, 1, \ldots$
2. $1, 2, 1, 1, 1, 1, \ldots$
3. $1, 2, 3, 1, 1, 1, \ldots$
4. $1, 2, 3, 4, 1, 1, \ldots$

Łamigłówka 4

krok VI – $1, 2, 3, 4, \ldots$

1. $1, 1, 1, 1, 1, 1, \ldots$
2. $1, 2, 1, 1, 1, 1, \ldots$
3. $1, 2, 3, 1, 1, 1, \ldots$
4. $1, 2, 3, 4, 1, 1, \ldots$
5. $\ldots$
6. $1, 2, 3, \ldots, n-2, 1, 1$
7. $1, 2, 3, \ldots, n-2, n-1, 1$
8. $1, 2, 3, \ldots, n-2, n-1, n$

Liczba operacji: $n - 1$

Sumarycznie: $n \sum_{x=2}^{n} 3 \log x + n \leq 30\,000 + 1\,560\,000 + 30\,000 \leq 2\,500\,000$

H. Merge sort

Najszybsze rozwiązanie: **Krzysztof Potępa (3:41)**

Merge sort

Treść

Dla ustalonych liczb a, b zliczyć permutacje $1 \ldots n$ takie, że a zostanie porównane z b podczas sortowania przez scalanie.

Merge sort

Obserwacja 1

Gdy a i b trafią do tej samej listy podczas sortowania, to nie zostaną już nigdy porównane.

Merge sort

Obserwacja 1

Gdy a i b trafią do tej samej listy podczas sortowania, to nie zostaną już nigdy porównane.

Obserwacja 2

Jest $\mathcal{O}(\log n)$ różnych długości list, które będą argumentem dla `MergeSort`.

Merge sort

Obserwacja 1

Gdy a i b trafią do tej samej listy podczas sortowania, to nie zostaną już nigdy porównane.

Obserwacja 2

Jest $\mathcal{O}(\log n)$ różnych długości list, które będą argumentem dla `MergeSort`. Możemy w szczególności je wszystkie wygenerować wraz z licznościami.

Obserwacja 3

Bez straty ogólności $a < b$. Aby a zostało porównane z b podczas łączenia list, to w liście, w której znajduje się b nie może znajdować się t takie że $a < t < b$.

Merge sort

Obserwacja 3

Bez straty ogólności $a < b$. Aby a zostało porównane z b podczas łączenia list, to w liście, w której znajduje się b nie może znajdować się t takie że $a < t < b$.

Rozwiązanie

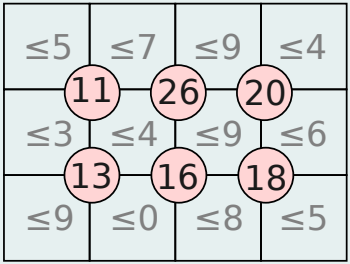
Iterujemy się po długości listy i tym, z której strony jest a.
Permutacje spełniające założenia jesteśmy w stanie zliczyć standardowymi technikami w $\mathcal{O}(1)$ po preprocessingu silni i ich odwrotności.
Trochę prościej jest wyprowadzić wzorki myśląc o prawdopodobieństwie.

J. Osiedle

Najszybsze rozwiązanie: **brak**

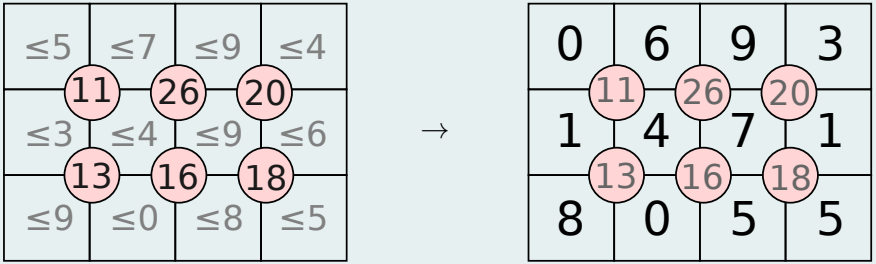
Osiedle

Treść

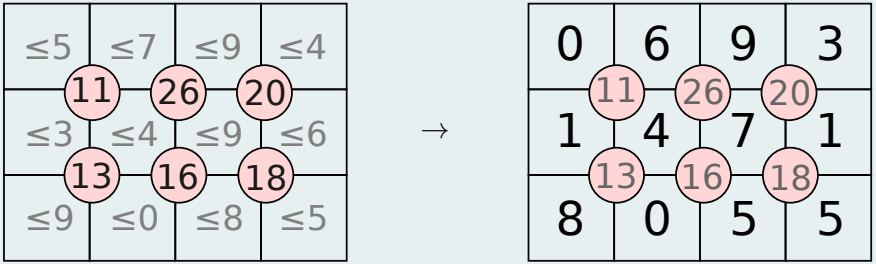


Osiedle

Treść

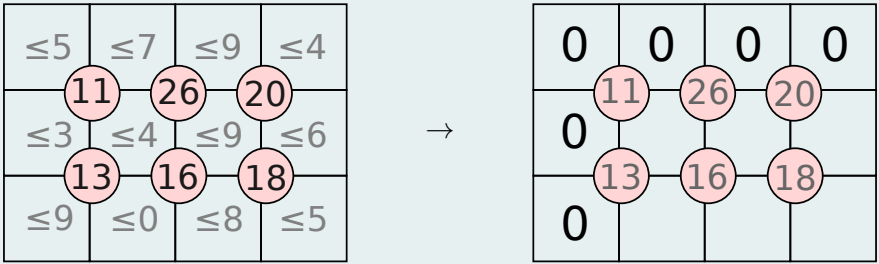


Treść



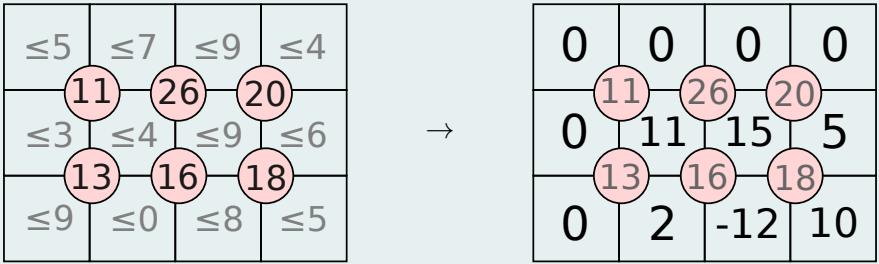
Wysokość (n), szerokość (m) ≤ 300.

Ignorujemy dolne i górne ograniczenia...

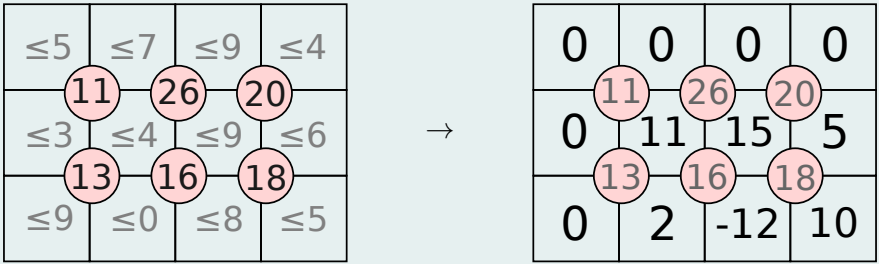


Osiedle

Ignorujemy dolne i górne ograniczenia...

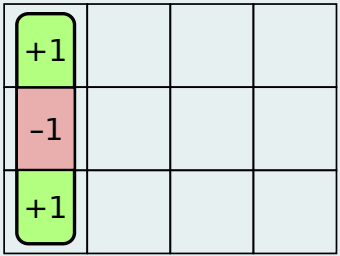


Ignorujemy dolne i górne ograniczenia...

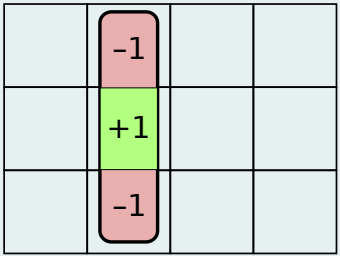


Czas $\mathcal{O}(nm)$, **poprawne sumy**, **popsute dolne i górne ograniczenia**.

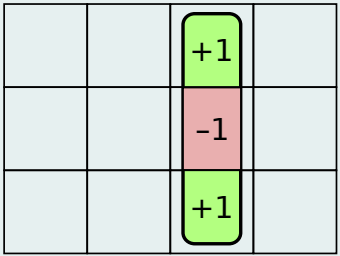
Lokalne zmiany zachowujące sumy 2×2



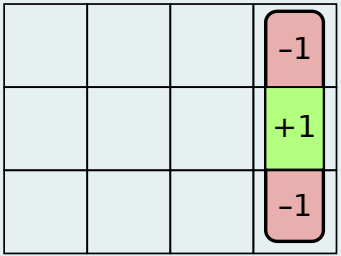
Lokalne zmiany zachowujące sumy 2×2



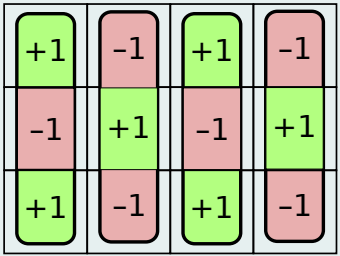
Lokalne zmiany zachowujące sumy 2×2



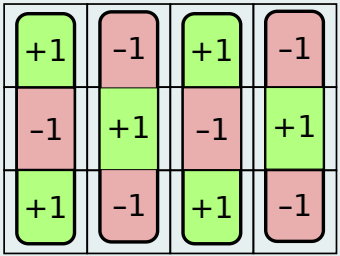
Lokalne zmiany zachowujące sumy 2×2

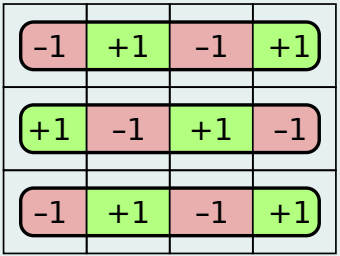


Lokalne zmiany zachowujące sumy 2×2

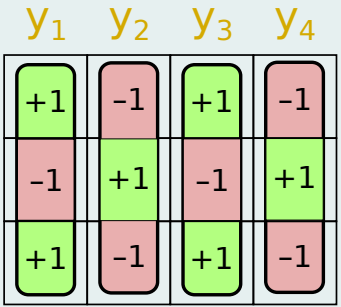


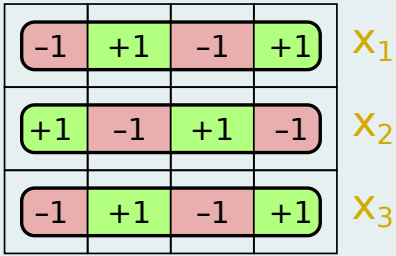
Lokalne zmiany zachowujące sumy 2×2





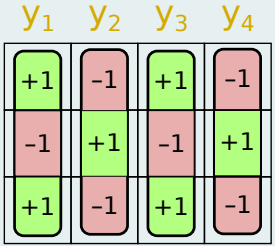
Lokalne zmiany zachowujące sumy 2×2



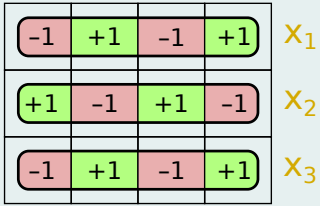


$x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathbb{Z}$.

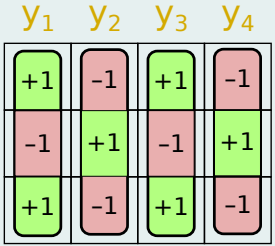
Naprawianie ograniczeń



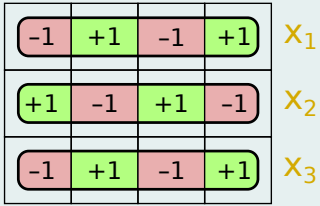
≤5 0	≤7 0	≤9 0	≤4 0
≤3 0	≤4 11	≤9 15	≤6 5
≤9 0	≤0 2	≤8 -12	≤5 10



Naprawianie ograniczeń

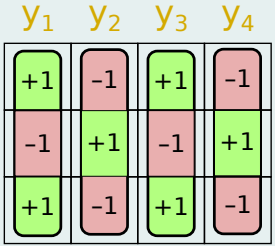


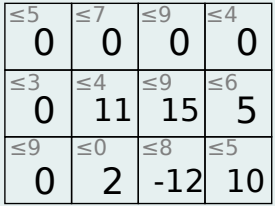
≤5 0	≤7 0	≤9 0	≤4 0
≤3 0	≤4 11	≤9 15	≤6 5
≤9 0	≤0 2	≤8 -12	≤5 10

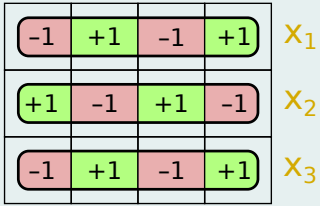


Pole $(1, 1)$: $0 + y_1 - x_1 \in [0, 5]$

Naprawianie ograniczeń

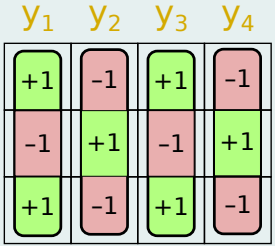




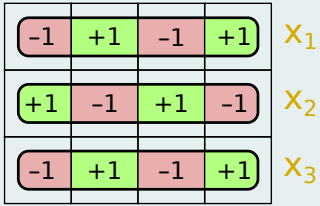


Pole $(1,1)$: $0 + y_1 - x_1 \in [0,5] \quad \Rightarrow \quad -5 \leq x_1 - y_1 \leq 0$

Naprawianie ograniczeń



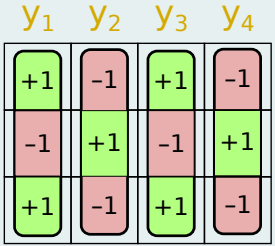
≤5 0	≤7 0	≤9 0	≤4 0
≤3 0	≤4 11	≤9 15	≤6 5
≤9 0	≤0 2	≤8 -12	≤5 10



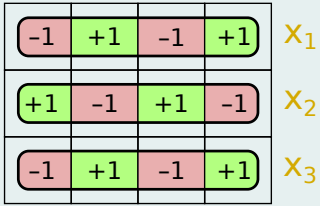
Pole $(1,1)$: $0 + y_1 - x_1 \in [0,5] \quad \Rightarrow \quad -5 \leq x_1 - y_1 \leq 0$

Pole $(2,3)$: $15 - y_3 + x_2 \in [0,9]$

Naprawianie ograniczeń



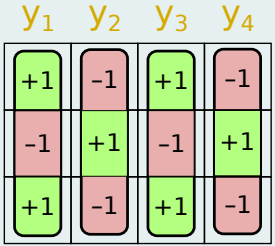
≤5 0	≤7 0	≤9 0	≤4 0
≤3 0	≤4 11	≤9 15	≤6 5
≤9 0	≤0 2	≤8 -12	≤5 10

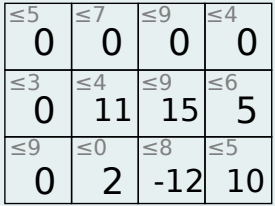


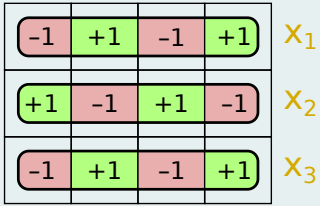
Pole $(1,1)$: $0 + y_1 - x_1 \in [0,5] \quad \Rightarrow \quad -5 \leq x_1 - y_1 \leq 0$

Pole $(2,3)$: $15 - y_3 + x_2 \in [0,9] \quad \Rightarrow \quad -15 \leq x_2 - y_3 \leq -6$

Naprawianie ograniczeń





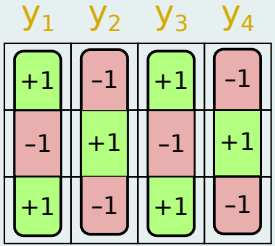


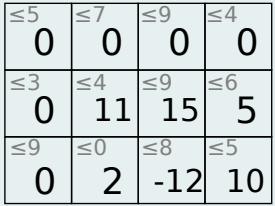
Pole $(1,1)$: $0 + y_1 - x_1 \in [0,5] \quad \Rightarrow \quad -5 \leq x_1 - y_1 \leq 0$

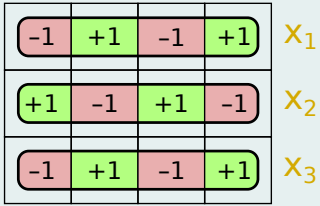
Pole $(2,3)$: $15 - y_3 + x_2 \in [0,9] \quad \Rightarrow \quad -15 \leq x_2 - y_3 \leq -6$

Ograniczenia różnicowe! $n + m$ zmiennych, $2nm$ ograniczeń.

Naprawianie ograniczeń







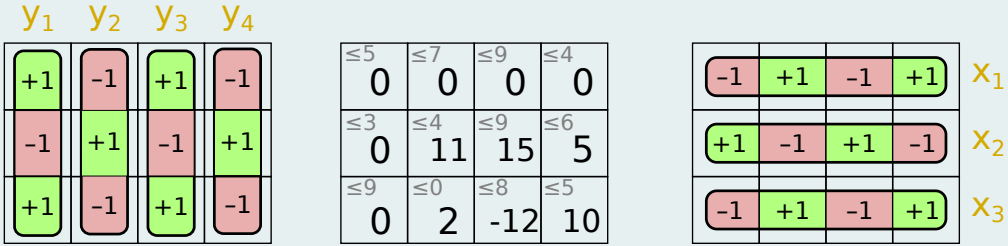
Pole $(1,1)$: $0 + y_1 - x_1 \in [0,5] \quad \Rightarrow \quad -5 \le x_1 - y_1 \le 0$

Pole $(2,3)$: $15 - y_3 + x_2 \in [0,9] \quad \Rightarrow \quad -15 \le x_2 - y_3 \le -6$

Ograniczenia różnicowe! $n+m$ zmiennych, $2nm$ ograniczeń.

Bellman–Ford: $O(nm(n+m))$.

Naprawianie ograniczeń



$y_1 = 0, \quad y_2 = -6, \quad y_3 = 9, \quad y_4 = -3, \qquad x_1 = 0, \quad x_2 = 1, \quad x_3 = -8$

Naprawianie ograniczeń

y_1	y_2	y_3	y_4
+1	-1	+1	-1
-1	+1	-1	+1
+1	-1	+1	-1

≤5 0	≤7 0	≤9 0	≤4 0
≤3 0	≤4 11	≤9 15	≤6 5
≤9 0	≤0 2	≤8 -12	≤5 10

-1	+1	-1	+1	x_1
+1	-1	+1	-1	x_2
-1	+1	-1	+1	x_3

$y_1 = 0, \quad y_2 = -6, \quad y_3 = 9, \quad y_4 = -3, \qquad x_1 = 0, \quad x_2 = 1, \quad x_3 = -8$

0	6	9	3
1	4	7	1
8	0	5	5

Ostateczna złożoność

$\mathcal{O}(nm(n+m))$ z Bellmana–Forda.

Ostateczna złożoność

$\mathcal{O}(nm(n+m))$ z Bellmana–Forda.

Albo $\mathcal{O}((n+m)^3)$ z Floyda–Warshalla.

Ostateczna złożoność

$\mathcal{O}(nm(n+m))$ z Bellmana–Forda.

Albo $\mathcal{O}((n+m)^3)$ z Floyda–Warshalla.

Albo $\approx \mathcal{O}(nm \log^3(nm) \log W)$ z nowych teoretycznych algorytmów dla Single-Source Shortest Paths with negative weights.

K. Projekty

Najszybsze rozwiązanie: **brak**

Projekty

Treść

- $n \leq 13$ programistów

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu
  - mogą się podzielić x_i oraz $2 \cdot (t_i - x_i)$

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu
  - mogą się podzielić x_i oraz $2 \cdot (t_i - x_i)$
- $q \leq 10\,000$ zapytań

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu
  - mogą się podzielić x_i oraz $2 \cdot (t_i - x_i)$
- $q \leq 10\,000$ zapytań
  - dany $X \subseteq \{1, 2, \ldots, n\}$

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu
  - mogą się podzielić x_i oraz $2 \cdot (t_i - x_i)$
- $q \leq 10\,000$ zapytań
  - dany $X \subseteq \{1, 2, \ldots, n\}$
  - każdy z X ma ograniczony czas T
    szukamy najmniejszego T, tż. da się skończyć wszystkie projekty

Projekty

Treść

- $n \leq 13$ programistów
- $m \leq 200$ projektów
  - a_i potrzebuje t_i czasu
  - b_i potrzebuje $2 \cdot t_i$ czasu
  - mogą się podzielić x_i oraz $2 \cdot (t_i - x_i)$
- $q \leq 10\,000$ zapytań
  - dany $X \subseteq \{1, 2, \ldots, n\}$
  - każdy z X ma ograniczony czas T
    szukamy najmniejszego T, tż. da się skończyć wszystkie projekty
  - wypisać w postaci x/y

Projekty

Program liniowy

Projekty

Program liniowy

minimize T

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

Projekty

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Projekty

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1 \qquad (T)$

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1 \qquad (T)$

$\forall_{ij} \quad w_{ij} - v_i \leq 0 \qquad (x_{ij})$

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1 \qquad (T)$

$\forall_{ij} \quad w_{ij} - v_i \leq 0 \qquad (x_{ij})$

$\forall_{ij} \quad \frac{w_{ij}}{2} - v_j \leq 0 \qquad (y_{ij})$

Program liniowy

minimize T

$\forall_{ij} \quad x_{ij} + \frac{y_{ij}}{2} \geq t_{ij} \qquad (w_{ij})$

$\forall_{i} \quad T - \sum_j x_{ij} - \sum_j y_{ji} \geq 0 \qquad (v_i)$

$T \geq 0$

$\forall_{ij} \quad x_{ij} \geq 0$

$\forall_{ij} \quad y_{ij} \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1 \qquad (T)$

$\forall_{ij} \quad w_{ij} - v_i \leq 0 \qquad (x_{ij})$

$\forall_{ij} \quad \frac{w_{ij}}{2} - v_j \leq 0 \qquad (y_{ij})$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_{i} \quad v_i \geq 0$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

Projekty

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych

Projekty

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

- $\sum_i v_i = 1$ 1

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

- $\sum_i v_i = 1$ — 1
- $v_i = 0$ — z

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

- $\sum_i v_i = 1$ — 1
- $v_i = 0$ — z
- $w_{ij} = 0$ — $n^2 - (n - z)^2$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

- $\sum_i v_i = 1$ — 1
- $v_i = 0$ — z
- $w_{ij} = 0$ — $n^2 - (n - z)^2$
- $w_{ij} = v_i \vee w_{ij} = 2 \cdot v_i$ — $(n - z)^2$

Program dualny

maximize $\sum_{ij} t_{ij} \cdot w_{ij}$

$\sum_i v_i \leq 1$

$\forall_{ij} \quad w_{ij} \leq v_i$

$\forall_{ij} \quad w_{ij} \leq 2 \cdot v_j$

$\forall_{ij} \quad w_{ij} \geq 0$

$\forall_i \quad v_i \geq 0$

$n^2 + n$ zmiennych $\Rightarrow$ $n^2 + n$ lnz. nierówności równościami

$\sum_i v_i = 1$	1
$v_i = 0$	z
$w_{ij} = 0$	$n^2 - (n - z)^2$
$w_{ij} = v_i \vee w_{ij} = 2 \cdot v_i$	$(n - z)^2$
$w_{ij} = v_i \wedge w_{ij} = 2 \cdot v_j$	$n - z - 1$

Struktura rozwiązania

Struktura rozwiązania

- $n - z$ zmiennych $v_i \neq 0$

Struktura rozwiązania

- $n - z$ zmiennych $v_i \neq 0$
- $n - z - 1$ równań $v_i = 2 \cdot v_j$

Struktura rozwiązania

- $n - z$ zmiennych $v_i \neq 0$
- $n - z - 1$ równań $v_i = 2 \cdot v_j$
- $v_i = 2 \cdot v_j$ krawędzie (i, j) tworzą drzewo!

Struktura rozwiązania

- $n - z$ zmiennych $v_i \neq 0$
- $n - z - 1$ równań $v_i = 2 \cdot v_j$
- $v_i = 2 \cdot v_j$ krawędzie (i, j) tworzą drzewo!
- istnieje m tż. $v_i = 0 \cup v_i = \frac{2^{k_i}}{m}$

Struktura rozwiązania

- $n - z$ zmiennych $v_i \neq 0$
- $n - z - 1$ równań $v_i = 2 \cdot v_j$
- $v_i = 2 \cdot v_j$ krawędzie (i, j) tworzą drzewo!
- istnieje m tż. $v_i = 0 \cup v_i = \frac{2^{k_i}}{m}$
  $m = \sum_i 2^{k_i}$

Programowanie dynamiczne

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T$

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T \cdot m$

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $\mathrm{DP}[X][|X|] = \sum_{i,j \in X} w_{ij}$

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $\mathrm{DP}[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia

Programowanie dynamiczne

- definicja: $DP[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $DP[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $\mathrm{DP}[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $\mathrm{DP}[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}]$

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \to T \cdot m$
- bazowe stany:
  $\mathrm{DP}[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $\mathrm{DP}[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}] \cdot 2 +$

Programowanie dynamiczne

- definicja: $DP[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $DP[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $DP[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}] \cdot 2 +$
  $\sum_{i \in X, j \in X \setminus Y} 2 \cdot w_{ij} +$

Programowanie dynamiczne

- definicja: $DP[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $DP[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $DP[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}] \cdot 2 +$
  $\sum_{i \in X, j \in X \setminus Y} 2 \cdot w_{ij} + \sum_{i \in X \setminus Y, j \in X} w_{ij}$

Programowanie dynamiczne

- definicja: $DP[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $DP[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $DP[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}] \cdot 2 +$
  $\sum_{i \in X, j \in X \setminus Y} 2 \cdot w_{ij} + \sum_{i \in X \setminus Y, j \in X} w_{ij}$

Ostateczny wynik:

Programowanie dynamiczne

- definicja: $\mathrm{DP}[\{v_i \neq 0\}][m] \rightarrow T \cdot m$
- bazowe stany:
  $\mathrm{DP}[X][|X|] = \sum_{i,j \in X} w_{ij}$
- przejścia (k_i zwiększamy o jeden i dodajemy zbiór $k_i = 0$):
  $\mathrm{DP}[X][m] = \max_{Y \subset X} DP[Y][\frac{m-(|Y|-|X|)}{2}] \cdot 2 +$
  $\sum_{i \in X, j \in X \setminus Y} 2 \cdot w_{ij} + \sum_{i \in X \setminus Y, j \in X} w_{ij}$

Ostateczny wynik:

$T(X) = max_{Y \subseteq X} \frac{DP[Y][m]}{m}$

Złożoność

$$DP[X][m]$$

- Stany

Złożoność

$$DP[X][m]$$

- Stany $m \leq 2^{|X|}$

Złożoność

$$DP[X][m]$$

- Stany $m \leq 2^{|X|}$

$$\mathcal{O}(3^n)$$

- Przejścia

Złożoność

$$DP[X][m]$$

- Stany $m \leq 2^{|X|}$

$$\mathcal{O}(3^n)$$

- Przejścia
  $Y \subset X$

Złożoność

$$DP[X][m]$$

- Stany $m \leq 2^{|X|}$

$$\mathcal{O}(3^n)$$

- Przejścia
  $Y \subset X$
  $m \leq 2^{|Y|}$

Złożoność

$$DP[X][m]$$

- Stany $m \leq 2^{|X|}$

$$\mathcal{O}(3^n)$$

- Przejścia
  $Y \subset X$
  $m \leq 2^{|Y|}$

$$\mathcal{O}(4^n)$$

N. Funkcja zliczająca

Najszybsze rozwiązanie: **Krzysztof Potępa (1:22)**

Funkcja zliczająca

Treść

Mamy dany ciąg liczb a. Rozważamy składanie ze sobą funkcji f przekształcającej ciąg w ciąg liczności wystąpień elementów. Musimy obsługiwać zmiany elementów ciągu a oraz zapytania o $f^k(a)_v$.

Funkcja zliczająca

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.

Funkcja zliczająca

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.

Funkcja zliczająca

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.
Więc wszystkie niezerowe elementy $f(f(f(a)))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i jest ich co najwyżej $\mathcal{O}(\sqrt{n})$.

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.
Więc wszystkie niezerowe elementy $f(f(f(a)))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i jest ich co najwyżej $\mathcal{O}(\sqrt{n})$.
Więc wszystkie elementy $f(f(f(f(a))))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i znajdują się na prefiksie długości $\mathcal{O}(\sqrt{n})$.

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.
Więc wszystkie niezerowe elementy $f(f(f(a)))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i jest ich co najwyżej $\mathcal{O}(\sqrt{n})$.
Więc wszystkie elementy $f(f(f(f(a))))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i znajdują się na prefiksie długości $\mathcal{O}(\sqrt{n})$.
Oznacza to, że dla pewnego $c = \mathcal{O}(\log \log n)$ zachodzi $f^c(a) = 1, 0, 0, 0....$

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.
Więc wszystkie niezerowe elementy $f(f(f(a)))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i jest ich co najwyżej $\mathcal{O}(\sqrt{n})$.
Więc wszystkie elementy $f(f(f(f(a))))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i znajdują się na prefiksie długości $\mathcal{O}(\sqrt{n})$.
Oznacza to, że dla pewnego $c = \mathcal{O}(\log \log n)$ zachodzi $f^c(a) = 1, 0, 0, 0....$
Chyba, że $a = 0, 0, 0, ...$

Funkcja zliczająca

Ciąg szybko się przerzedza

Jeżeli ciąg a składa się z n elementów ograniczonych z góry przez n, to suma elementów $f(a)$ wynosi co najwyżej n.
Więc jest co najwyżej $\mathcal{O}(\sqrt{n})$ różnych niezerowych elementów w $f(f(a))$.
Więc wszystkie niezerowe elementy $f(f(f(a)))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i jest ich co najwyżej $\mathcal{O}(\sqrt{n})$.
Więc wszystkie elementy $f(f(f(f(a))))$ są ograniczone przez $\mathcal{O}(\sqrt{n})$ i znajdują się na prefiksie długości $\mathcal{O}(\sqrt{n})$.
Oznacza to, że dla pewnego $c = \mathcal{O}(\log \log n)$ zachodzi $f^c(a) = 1, 0, 0, 0....$
Chyba, że $a = 0, 0, 0, ...$
W praktyce dla limitów z zadania wystarcza $c = 13$.

Funkcja zliczająca

Rozwiązanie

Będziemy utrzymywać ciągi $a, f(a), f(f(a)), ..., f^c(a)$.

Funkcja zliczająca

Rozwiązanie

Będziemy utrzymywać ciągi $a, f(a), f(f(a)), ..., f^c(a)$.
Jeżeli jesteśmy pytani o $f^k(a)_v$ dla $k \leq c$, to możemy po prostu podać odpowiedź w $\mathcal{O}(1)$.

Funkcja zliczająca

Rozwiązanie

Będziemy utrzymywać ciągi $a, f(a), f(f(a)), ..., f^c(a)$.
Jeżeli jesteśmy pytani o $f^k(a)_v$ dla $k \leq c$, to możemy po prostu podać odpowiedź w $\mathcal{O}(1)$.
Jeżeli $k > c$, to odpowiedzią jest 1 jeżeli $v = 1$ oraz a nie składa się z samych zer i 0 w przeciwnym przypadku.

Funkcja zliczająca

Aktualizacje

Zmiana wartości $f^i(a)_v$ z x na y wiąże się z ze zmianami wartości $f^{i+1}(a)_x$ oraz $f^{i+1}(a)_y$.

Funkcja zliczająca

Aktualizacje

Zmiana wartości $f^i(a)_v$ z x na y wiąże się z ze zmianami wartości $f^{i+1}(a)_x$ oraz $f^{i+1}(a)_y$.
Możemy napisać funkcje, która rekurencyjnie wykonuje zmiany i wywołuje się dla dalszych ciągów.

Aktualizacje

Zmiana wartości $f^i(a)_v$ z x na y wiąże się z ze zmianami wartości $f^{i+1}(a)_x$ oraz $f^{i+1}(a)_y$.
Możemy napisać funkcje, która rekurencyjnie wykonuje zmiany i wywołuje się dla dalszych ciągów.
Złożoność $\mathcal{O}(q2^c)$.

Funkcja zliczająca

Szybsze aktualizacje

Zamiast iść w głąb drzewa zmian, możemy rozważać poziomy jeden po drugim.

Funkcja zliczająca

Szybsze aktualizacje

Zamiast iść w głąb drzewa zmian, możemy rozważać poziomy jeden po drugim.
Kilka aktualizacji, które miałyby zostać wykonane na tym samym elemencie zrobimy w ten sposób za jednym razem.

Funkcja zliczająca

Szybsze aktualizacje

Zamiast iść w głąb drzewa zmian, możemy rozważać poziomy jeden po drugim.
Kilka aktualizacji, które miałyby zostać wykonane na tym samym elemencie zrobimy w ten sposób za jednym razem.
Dzięki temu na i-tym poziomie zamiast wykonać pesymistycznie 2^i operacji wykonamy ich $\min(2^i, len_i)$ gdzie len_i to długość prefiksu niezerowych wartości na i-tym poziomie.

Funkcja zliczająca

Szybsze aktualizacje

Zamiast iść w głąb drzewa zmian, możemy rozważać poziomy jeden po drugim.
Kilka aktualizacji, które miałyby zostać wykonane na tym samym elemencie zrobimy w ten sposób za jednym razem.
Dzięki temu na i-tym poziomie zamiast wykonać pesymistycznie 2^i operacji wykonamy ich $\min(2^i, len_i)$ gdzie len_i to długość prefiksu niezerowych wartości na i-tym poziomie.
Przy limitach z zadania daje to ograniczenie rzędu kilkudziesięciu operacji.

OMÓWIENIE ZADAŃ

DIGITAL TECHNOLOGY POLAND
UNIWERSYTET WARSZAWSKI
FRI
RTB HOUSE
HUAWEI
FRIEDHELM LOH GROUP

A. AGI

Najszybsze rozwiązanie: **Adam Sołtan (1:24)**

Autor zadania: **Jakub Onufry Wojtaszczyk**

Treść zadania

- Dane jest n przedziałów $[a_i, b_i)$
- Jest jeden punkt x na osi rzeczywistej, którego położenia nie znamy, dla każdego przedziału musimy odpowiedzieć `TAK` albo `NIE`, czy ukryty punkt x należy do tego przedziału w taki sposób, żeby niezależnie od położenia punktu udzielić poprawnej odpowiedzi dla co najmniej $\left\lfloor \frac{n-1}{2} \right\rfloor$ przedziałów
- Jest zagwarantowane, że rozwiązanie istnieje

Rozwiązanie

- Udowodnimy, że rozwiązanie istnieje dla każdego zbioru przedziałów

Rozwiązanie

- Udowodnimy, że rozwiązanie istnieje dla każdego zbioru przedziałów
- Jeżeli istnieją dwa przedziały $[a_i, b_i)$ i $[a_j, b_j)$ takie, że $[a_i, b_i) \subseteq [a_j, b_j)$ możemy odpowiedzieć `NIE` dla predziału $[a_i, b_i)$, `TAK` dla przedziału $[a_j, b_j)$, wówczas dla co najmniej jednego z nich odpowiedzieliśmy prawidłowo, możemy oba te przedziały usunąć i rozwiązać zadanie dla pozostałych

Rozwiązanie

- Udowodnimy, że rozwiązanie istnieje dla każdego zbioru przedziałów
- Jeżeli istnieją dwa przedziały $[a_i, b_i)$ i $[a_j, b_j)$ takie, że $[a_i, b_i) \subseteq [a_j, b_j)$ możemy odpowiedzieć `NIE` dla predziału $[a_i, b_i)$, `TAK` dla przedziału $[a_j, b_j)$, wówczas dla co najmniej jednego z nich odpowiedzieliśmy prawidłowo, możemy oba te przedziały usunąć i rozwiązać zadanie dla pozostałych
- Jeśli takiej pary przedziałów nie ma, wówczas po posortowaniu wszystkich przedziałów rosnąco po początkach mamy ciąg przedziałów $[a_1, b_1), [a_2, b_2), \ldots, [a_n, b_n)$ takich, że $a_1 < a_2 < \ldots < a_n$ oraz $b_1 < b_2 < \ldots < b_n$. Pokażemy, że możemy odpowedzieć `NIE` dla przedziałów z nieparzystym i, `TAK` dla tych z parzystym i dla każdego x trafimy co najmniej wymagane $\left\lfloor \frac{n-1}{2} \right\rfloor$

Dowód poprawności

- Dla ustalonego x ciąg poprawnych składa się z (być może pustych) przedziałów złożonych kolejno z samych odpowiedzi `NIE`, `TAK`, `NIE`. Dopóki któryś z tych przedziałów ma długość co najmniej 2, to dla jednego z nich odpowidzieliśmy prawidłowo, więc możemy je oba usunąć
- Jeśli po usunięciu zostaną ≤ 2 przedziały, to nie musimy odpowiedzieć poprawnie dla żadnego z nich, jeśli zostaną 3, to poprawne odpowiedzi to `NIE`, `TAK`, `NIE`, więc wszystkie 3 odpowiemy poprawnie

Implementacja

- Aby wydajnie wyznaczyć pary przedziałów zawierające się w sobie możemy posortować wszystkie przedziały po lewych końcach, trzymać je na stosie, dla każdego przedziału jeśli zawiera się w przedziale znajdującym się na szczycie stosu łączymy je w parę i oba usuwamy, w przeciwnym przypadku dodajemy na szczyt stosu.
- Cała złożoność $O(n \log n)$

B. Collatzowa Suma

Najszybsze rozwiązanie: **Yahor Dubovik (0:34)**

Autorzy zadania: **Franciszek Witt, Arkadiusz Czarkowski**

Collatzowa Suma

Treść zadania

$$f(x) = \begin{cases} x/2, & \text{gdy } x \text{ jest parzyste,} \\ 3x+1, & \text{gdy } x \text{ jest nieparzyste} \end{cases}$$

Collatzowa Suma

Treść zadania

$$f(x) = \begin{cases} x/2, & \text{gdy } x \text{ jest parzyste,} \\ 3x+1, & \text{gdy } x \text{ jest nieparzyste} \end{cases}$$

$$g(x) = \underbrace{f(f(\ldots f}_{k \text{ razy}}(x)\ldots))$$

Collatzowa Suma

Treść zadania

$$f(x) = \begin{cases} x/2, & \text{gdy } x \text{ jest parzyste,} \\ 3x+1, & \text{gdy } x \text{ jest nieparzyste} \end{cases}$$

$$g(x) = \underbrace{f\big(f(\ldots f}_{k \text{ razy}}(x)\ldots)\big)$$

$$\sum_{x=1}^{N} g(x) \mod 10^9 + 7 = ?$$

Collatzowa Suma

Treść zadania

$$f(x) = \begin{cases} x/2, & \text{gdy } x \text{ jest parzyste,} \\ 3x+1, & \text{gdy } x \text{ jest nieparzyste} \end{cases}$$

$$g(x) = \underbrace{f\big(f(\ldots f}_{k \text{ razy}}(x)\ldots)\big)$$

$$\sum_{x=1}^{N} g(x) \mod 10^9 + 7 = ?$$

$$k \leq 32, N \leq 10^{12}$$

Collatzowa Suma

$k = 0$

Ciąg to $1, 2, 3, 4, \ldots, N$. Suma to $N(N+1)/2$.

Collatzowa Suma

$k = 0$

Ciąg to $1, 2, 3, 4, \ldots, N$. Suma to $N(N+1)/2$.

$k = 1$

Ciąg rozbija się na liczby parzyste, i nieparzyste.
Dla liczb parzystych mamy, kolejno, $1, 2, 3, 4, \ldots, N/2$.
Dla liczb nieparzystych mamy, kolejno, $4, 10, 16, 22, \ldots, 3N+1$ (albo $3(N-1)+1$).

Collatzowa Suma

$k = 0$

Ciąg to $1, 2, 3, 4, \ldots, N$. Suma to $N(N+1)/2$.

$k = 1$

Ciąg rozbija się na liczby parzyste, i nieparzyste.
Dla liczb parzystych mamy, kolejno, $1, 2, 3, 4, \ldots, N/2$.
Dla liczb nieparzystych mamy, kolejno, $4, 10, 16, 22, \ldots, 3N+1$ (albo $3(N-1)+1$).

$k = 2$

Pierwszy ciąg znowu rozbija się na dwa, jak poprzednio: $1, 2, 3, 4, \ldots, N/4$ i $4, 10, 16, 22, \ldots, (3N/2+1)$.
Drugi ciąg jest cały parzysty, więc przekształca się w $2, 5, 8, 11, \ldots, (3N+1)/2$.

Aplikujemy funkcję Collatza do ciągu arytmetycznego

Ciąg arytmetyczny $ai + b$ dla $i = 1, 2, \ldots m$ oznaczmy przez $\{a, b, m\}$.
Jak wygląda ciąg $f(ai + b)$?

Collatzowa Suma

Aplikujemy funkcję Collatza do ciągu arytmetycznego

Ciąg arytmetyczny $ai + b$ dla $i = 1, 2, \ldots m$ oznaczmy przez $\{a, b, m\}$.
Jak wygląda ciąg $f(ai + b)$?

- Jeśli a i b są parzyste, to $f(\{a, b, m\}) = \{a/2, b/2, m\}$.
- Jeśli a jest parzyste, a b nie, to $f(\{a, b, m\}) = \{3a, 3b + 1, m\}$.

  To się nigdy nie wydarzy
- Jeśli a jest nieparzyste, to w wyniku mamy dwa podciągi. Jeśli b jest parzyste, są to $\{a/2, b/2, \lceil m/2 \rceil\}$ oraz $\{3a, 3(a + b) + 1, \lfloor m/2 \rfloor\}$, drugi przypadek jest symetryczny.

Szkic rozwiązania

Umiemy zatem w $O(1)$:

- Przyłożyć f do ciągu arytmetycznego, otrzymując 1 lub 2 ciągi.
- Zsumować ciąg arytmetyczny (no, umiemy)

To możemy zamienić w rozwiązanie rekurencyjne:

$$Wynik(\{a, b, m, k\}) = Wynik(f(\{a, b, m\}), k - 1).$$

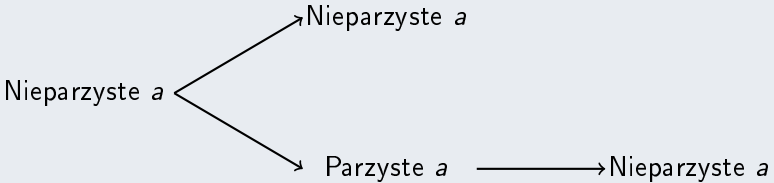
Collatzowa Suma

Złożoność

Niech $Fib(k)$ oznacza liczbę ciągów, w które zmieni się pojedynczy ciąg z nieparzystym a po przyłożeniu Collatza k razy.

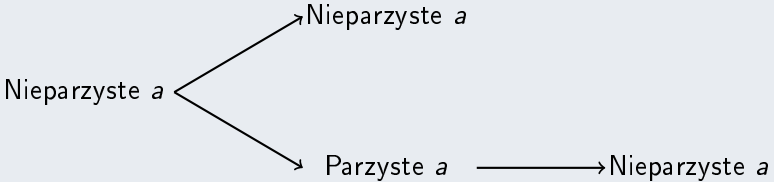
Złożoność

Niech $Fib(k)$ oznacza liczbę ciągów, w które zmieni się pojedynczy ciąg z nieparzystym a po przyłożeniu Collatza k razy.



Złożoność

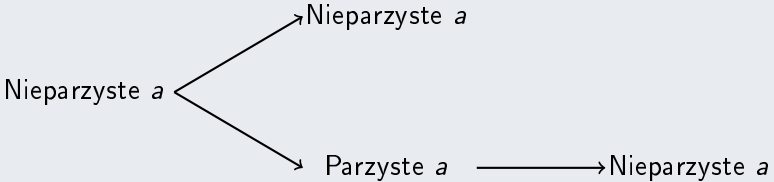
Niech $Fib(k)$ oznacza liczbę ciągów, w które zmieni się pojedynczy ciąg z nieparzystym a po przyłożeniu Collatza k razy.



$$Fib(k) = Fib(k-1) + Fib(k-2).$$

Złożoność

Niech $Fib(k)$ oznacza liczbę ciągów, w które zmieni się pojedynczy ciąg z nieparzystym a po przyłożeniu Collatza k razy.



$$Fib(k) = Fib(k-1) + Fib(k-2).$$

$$Fib(32) = 5\,702\,887$$

G. Meble

Najszybsze rozwiązanie: **brak**

Autor zadania: **Marcin Smulewicz**

Meble

Treść zadania

- Mamy daną planszę $A \times B$ oraz n prostokątów o wymiarach $c_i \times d_i$
- Chcemy rozmieścić prostokąty wszystkie tak, żeby żadne dwa na siebie nie nachodziły (mogą się stykać), tak żeby każdy stał przy przedniej lub tylnej ścianie (tej długości A) i stykał się z nią bokiem długości c_i
- Wystarczy tylko odpowiedzieć czy się da

Kluczowa obserwacja

Jeśli istnieje rozwiązanie, to istnieje również rozwiązanie, w którym przy dolnej ścianie wszystkie meble są dosunięte do lewej ściany i posortowane malejaco po głębokościach, a na górnej dosunięte do prawej i posortowane rosnąco

Kluczowa obserwacja

Jeśli istnieje rozwiązanie, to istnieje również rozwiązanie, w którym przy dolnej ścianie wszystkie meble są dosunięte do lewej ściany i posortowane malejaco po głębokościach, a na górnej dosunięte do prawej i posortowane rosnąco

Rozwiązanie

- Podzielimy wszystkie meble na takie o wysokości większej niż $B/2$ (nazwiemy je dużymi) oraz nie większej niż $B/2$ (nazwiemy je małymi)

Kluczowa obserwacja

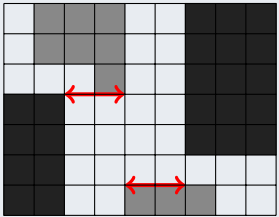
Jeśli istnieje rozwiązanie, to istnieje również rozwiązanie, w którym przy dolnej ścianie wszystkie meble są dosunięte do lewej ściany i posortowane malejaco po głębokościach, a na górnej dosunięte do prawej i posortowane rosnąco

Rozwiązanie

- Podzielimy wszystkie meble na takie o wysokości większej niż $B/2$ (nazwiemy je dużymi) oraz nie większej niż $B/2$ (nazwiemy je małymi)
- Posortujemy wszystkie meble malejąco po: d_i dla dużych mebli i $B - d_i$ dla małych, a z równych wartości najpierw małe, wówczas każdy mały mebel koliduje tylko z dużymi meblami, które są przed nim, a każdy duży koliduje ze wszystkimi dużymi.

Rozwiązanie

W rozwiązaniu będziemy korzystać z programowania dynamicznego: dla każdego prefiksu mebli chcemy znać wszystkie możliwe stany gdzie ustawialiśmy każdy mały mebel maksymalnie z prawej przy dolnej ścianie, lub maksymalnie z lewej przy górnej, a każdy duży maksymalnie z lewej przy dolnej, lub z prawej przy górnej gdzie interesuje nas jak dużo fragment z małymi meblami wystaje nad fragment z dużymi po lewej oraz po prawej.



Rozwiązanie

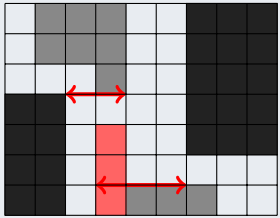
- Dla pustego prefiksu mebli wybieramy od razu wielkość pustego fragmentu po lewej przy dolnej ścianie i po prawej przy górnej. Suma ich długości jest równa $2A - \sum_{i=1}^{n} c_i$, co daje nam $O(A)$ stanów.

Rozwiązanie

- Dla pustego prefiksu mebli wybieramy od razu wielkość pustego fragmentu po lewej przy dolnej ścianie i po prawej przy górnej. Suma ich długości jest równa $2A - \sum_{i=1}^{n} c_i$, co daje nam $O(A)$ stanów.
- Dla każdego mebla rozpatrujemy dwa przypadki w którym miejscu chcemy go postawić.

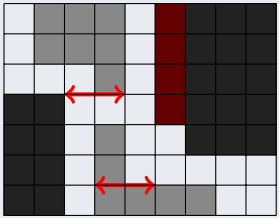
Rozwiązanie

- Dla pustego prefiksu mebli wybieramy od razu wielkość pustego fragmentu po lewej przy dolnej ścianie i po prawej przy górnej. Suma ich długości jest równa $2A - \sum_{i=1}^{n} c_i$, co daje nam $O(A)$ stanów.
- Dla każdego mebla rozpatrujemy dwa przypadki w którym miejscu chcemy go postawić.
- Ustawiając małego mebla zwiększamy odpowiedni wystający fragment o jego szerokość



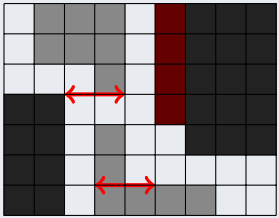
Rozwiązanie

- Ustawiając dużego zmniejszamy odpowiedni wystający fragment o jego szerokość



Rozwiązanie

- Ustawiając dużego zmniejszamy odpowiedni wystający fragment o jego szerokość



- Jeżeli w jakimkolwiek momencie fragment z dużymi meblami wystaje ponad fragment z krótkimi to kolejny mały mebel będzie kolidował z ustawionym już dużym

Rozwiązanie

- Suma długości wystających fragmentów na górnej i dolnej ścianie jest równa $2A - \sum_{i=1}^{n} c_i + \sum_{i:\text{małe}} c_i - \sum_{i:\text{duże}} c_i$, czyli jest taka sama dla wszystkich stanów co daje $O(A)$ stanów, co daje rozwiązanie całego zadania w $O(nA)$

Rozwiązanie

- Suma długości wystających fragmentów na górnej i dolnej ścianie jest równa $2A - \sum_{i=1}^{n} c_i + \sum_{i:\text{małe}} c_i - \sum_{i:\text{duże}} c_i$, czyli jest taka sama dla wszystkich stanów co daje $O(A)$ stanów, co daje rozwiązanie całego zadania w $O(nA)$
- $n \leq 1000, A \leq 10^6$, czyli wciąż za wolno

Rozwiązanie

- Suma długości wystających fragmentów na górnej i dolnej ścianie jest równa $2A - \sum_{i=1}^{n} c_i + \sum_{i:\text{małe}} c_i - \sum_{i:\text{duże}} c_i$, czyli jest taka sama dla wszystkich stanów co daje $O(A)$ stanów, co daje rozwiązanie całego zadania w $O(nA)$
- $n \leq 1000, A \leq 10^6$, czyli wciąż za wolno
- Dla każdego stanu interesuje nas jedynie, czy dany stan jest osiągalny, możemy trzymać stany na bitsecie, na i-tym indeksie wartość 1 jeśli stan z zapasem wielkości i w dolnej części jest osiągalny i 0 w przeciwnym przypadku.

Rozwiązanie

- Suma długości wystających fragmentów na górnej i dolnej ścianie jest równa $2A - \sum_{i=1}^{n} c_i + \sum_{i:\text{małe}} c_i - \sum_{i:\text{duże}} c_i$, czyli jest taka sama dla wszystkich stanów co daje $O(A)$ stanów, co daje rozwiązanie całego zadania w $O(nA)$
- $n \leq 1000, A \leq 10^6$, czyli wciąż za wolno
- Dla każdego stanu interesuje nas jedynie, czy dany stan jest osiągalny, możemy trzymać stany na bitsecie, na i-tym indeksie wartość 1 jeśli stan z zapasem wielkości i w dolnej części jest osiągalny i 0 w przeciwnym przypadku.
- Potrzebujemy operacji ora dwóch bitsetów, które dostaniemy dodając mebel na górze lub na dole oraz przesunięcia bitowego i wyzerowania fragmentu gdzie $i >$ `suma długości zapasów` (czyli takich, gdzie drugi wystający fragment jest ujemny), a wszystko da się zrobić na bitsecie w $O(A/64)$, co daje całą złożoność $O(nA/64)$

I. Mikołaj

Najszybsze rozwiązanie: **Kamil Dębowski (0:15)**

Autor zadania: **Marcin Smulewicz**

Mikołaj

Treść zadania

Dane:

Treść zadania

Dane:

- N prezentów: jakości $-10^6 \leq j_i \leq 10^6$.

Treść zadania

Dane:

- N prezentów: jakości $-10^6 \leq j_i \leq 10^6$.
- M dzieci: grzeczności $-10^6 \leq g_i \leq 10^6$.

Treść zadania

Dane:

- N prezentów: jakości $-10^6 \leq j_i \leq 10^6$.
- M dzieci: grzeczności $-10^6 \leq g_i \leq 10^6$.
- K prezentów trzeba dziś rozdać.

Treść zadania

Dane:

- N prezentów: jakości $-10^6 \leq j_i \leq 10^6$.
- M dzieci: grzeczności $-10^6 \leq g_i \leq 10^6$.
- K prezentów trzeba dziś rozdać.

Za przydzielenie (j_i, g_l), zyskujemy $j_i \cdot g_l$.

Treść zadania

Dane:

- N prezentów: jakości $-10^6 \leq j_i \leq 10^6$.
- M dzieci: grzeczności $-10^6 \leq g_i \leq 10^6$.
- K prezentów trzeba dziś rozdać.

Za przydzielenie (j_i, g_l), zyskujemy $j_i \cdot g_l$. Szukamy:

$$\max \sum_{i=1}^{K} j_{f(i)} g_{h(i)}$$

dla różnowartościowych f oraz h.

Sortujemy

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1$$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1 \qquad f_1 \cdot g_1 + f_2 \cdot g_2$$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1 \qquad f_1 \cdot g_1 + f_2 \cdot g_2$$

$$(f_2 - f_1) \cdot (g_2 - g_1)$$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1 \qquad f_1 \cdot g_1 + f_2 \cdot g_2$$

$$0 \leq (f_2 - f_1) \cdot (g_2 - g_1)$$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1 \qquad f_1 \cdot g_1 + f_2 \cdot g_2$$

$$0 \leq (f_2 - f_1) \cdot (g_2 - g_1) = f_1 \cdot g_1 + f_2 \cdot g_2 - f_1 \cdot g_2 - f_2 \cdot g_1.$$

Sortujemy

- $f_1 \leq f_2 \leq \cdots \leq f_n$
- $g_1 \leq g_2 \leq \cdots \leq g_m$

Przypadek $N = M = K = 2$

$$f_1 \cdot g_2 + f_2 \cdot g_1 \leq f_1 \cdot g_1 + f_2 \cdot g_2$$

$$0 \leq (f_2 - f_1) \cdot (g_2 - g_1) = f_1 \cdot g_1 + f_2 \cdot g_2 - f_1 \cdot g_2 - f_2 \cdot g_1.$$

Istnieje $f_i \cdot g_j > 0$

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne
- Oba dodatnie

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1)

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy `OPT` używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1 – analogicznie – mamy zysk.

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1 – analogicznie – mamy zysk.
- Można wziąć oba zachłannie!

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1 – analogicznie – mamy zysk.
- Można wziąć oba zachłannie! – i zachłannie sparować $f_1 \cdot g_1$.

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1 – analogicznie – mamy zysk.
- Można wziąć oba zachłannie! – i zachłannie sparować $f_1 \cdot g_1$.

$f_1 \cdot g_1 < f_n \cdot g_m$ – analogicznie.

Mikołaj

Istnieje $f_i \cdot g_j > 0$

- Oba ujemne – najlepiej $f_1 \cdot g_1$.
- Oba dodatnie – najlepiej $f_n \cdot g_m$.

Załóżmy $f_1 \cdot g_1 \geq f_n \cdot g_m$.

Czy OPT używa f_1 i g_1?

- Nie używa żadnego – wyrzucamy dowolne $f_i \cdot g_j$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa g_1 (używa f_1) – wyrzucamy $f_1 \cdot g_i$, dokładamy $f_1 \cdot g_1$ – mamy zysk.
- Nie używa f_1 – analogicznie – mamy zysk.
- Można wziąć oba zachłannie! – i zachłannie sparować $f_1 \cdot g_1$.

$f_1 \cdot g_1 < f_n \cdot g_m$ – analogicznie. Powtarzamy w pętli.

Mikołaj

Wszystkie $f_i \cdot g_j \leq 0$

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n$

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

Mikołaj

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k$

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)
Wniosek:

Mikołaj

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)
Wniosek: bierzemy zachłannie $f_1, \ldots, f_k$.

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)
Wniosek: bierzemy zachłannie $f_1, \ldots, f_k$.
Analogicznie, bierzemy zachłannie $g_{m-k+1}, g_{m-k+2}, \ldots, g_m$.

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięło parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)
Wniosek: bierzemy zachłannie $f_1, \ldots, f_k$.
Analogicznie, bierzemy zachłannie $g_{m-k+1}, g_{m-k+2}, \ldots, g_m$.
Parujemy $f_1 \cdot g_{m-k+1}$, $f_2 \cdot g_{m-k+2}, \ldots, f_k \cdot g_m$.

Mikołaj

Wszystkie $f_i \cdot g_j \leq 0$

- Dodatnie w jednym; Ujemne w jednym.
- W tym samym: $f_1 < 0 < f_n \Rightarrow \forall_i g_i = 0$ – parujemy jakkolwiek.
- W innych: $\forall_i f_i \geq 0 \wedge g_i \leq 0$.

f_i nieujemne; g_i niedodatnie

Załóżmy, że `OPT` nie używa f_i dla $i \leq k \Rightarrow$ wzięto parę $f_j \cdot g_l$ dla $j > k$.
Wyrzucamy $f_j \cdot g_l$, dokładamy $f_i \cdot g_l$ – mamy zysk ($g_l \leq 0 \wedge f_i \leq f_j$)
Wniosek: bierzemy zachłannie $f_1, \ldots, f_k$.
Analogicznie, bierzemy zachłannie $g_{m-k+1}, g_{m-k+2}, \ldots, g_m$.
Parujemy $f_1 \cdot g_{m-k+1}$, $f_2 \cdot g_{m-k+2}, \ldots, f_k \cdot g_m$.

Złożoność: $O(n \log n)$ (Sortowanie)

L. Skierowane Hanoi

Najszybsze rozwiązanie: **Kamil Dębowski (2:22)**

Autor zadania: **Marcin Smulewicz**

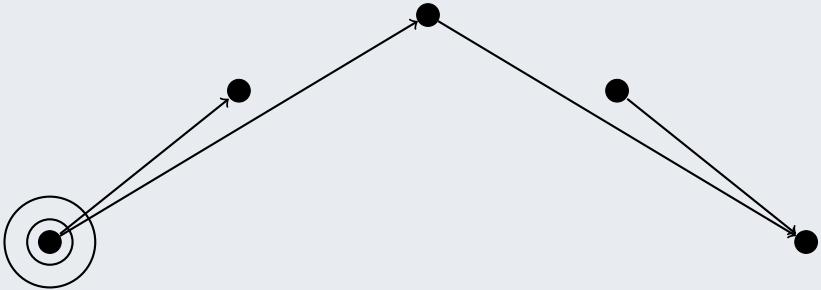
Skierowane Hanoi

Treść zadania

- Dany jest graf skierowany acykliczny na $N \leq 500$ słupkach.
- Mamy K krążków, o rozmiarach od 1 do K, ułożonych malejąco na słupku 1.
- Możemy przenieść najmniejszy krążek z dowolnego słupka A na inny słupek B, o ile:
  - Istnieje ścieżka ze słupka A do słupka B
  - Na słupku B nie ma mniejszego krążka
- Dla jakiego największego K da się wszystkie krążki przenieść na słupek N?

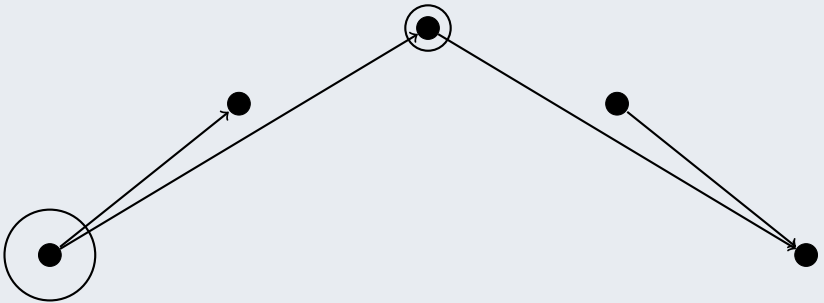
Skierowane Hanoi

Przykład 1



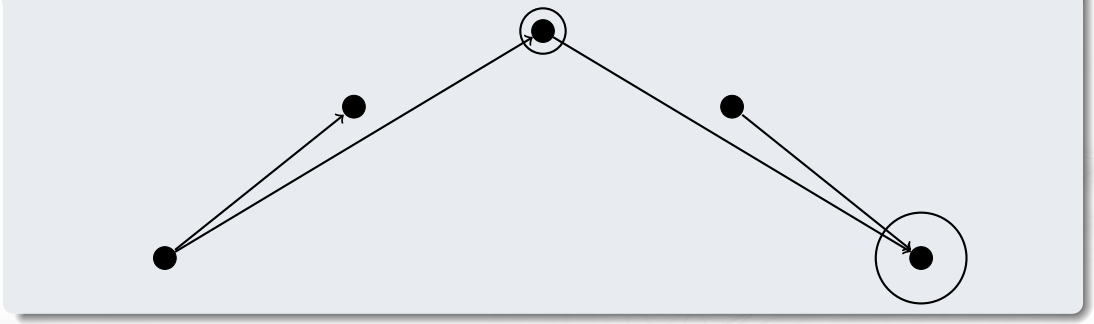
Skierowane Hanoi

Przykład 1



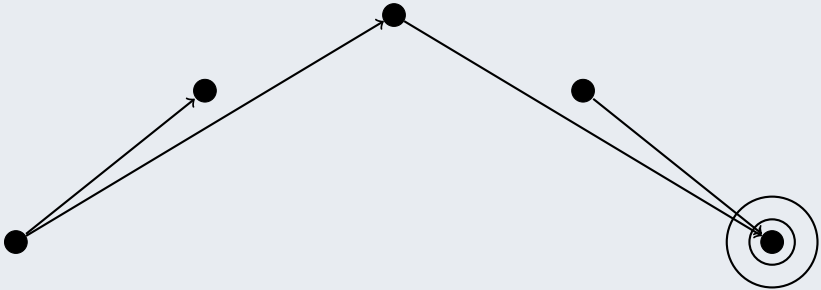
Skierowane Hanoi

Przykład 1



Skierowane Hanoi

Przykład 1



Proste przypadki

- Możemy wyrzucić z grafu wszystkie słupki, do których nie da się dojść z 1.

Proste przypadki

- Możemy wyrzucić z grafu wszystkie słupki, do których nie da się dojść z 1.
- I słupki, z których nie da się dojść do N.

Proste przypadki

- Możemy wyrzucić z grafu wszystkie słupki, do których nie da się dojść z 1.
- I słupki, z których nie da się dojść do N.
- Jeśli z 1 nie da się dojść do N, to odpowiedzią jest 0.

Skierowane Hanoi

Proste przypadki

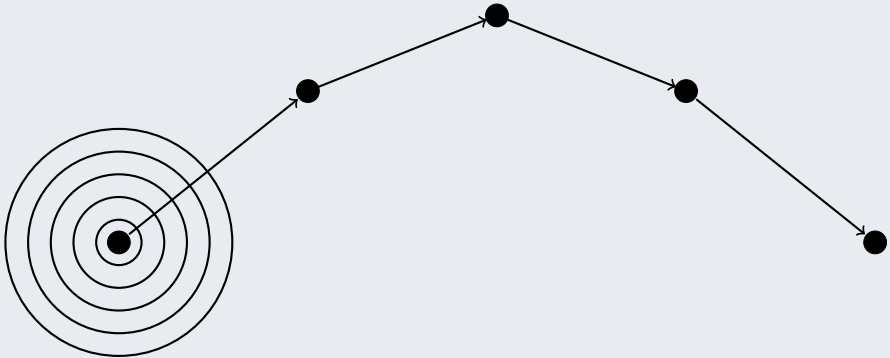
- Możemy wyrzucić z grafu wszystkie słupki, do których nie da się dojść z 1.
- I słupki, z których nie da się dojść do N.
- Jeśli z 1 nie da się dojść do N, to odpowiedzią jest 0.
- W przeciwnym razie, widać, że da się liczba wierzchołków połączonych zarówno z 1 jak i N, minus 1.

Skierowane Hanoi

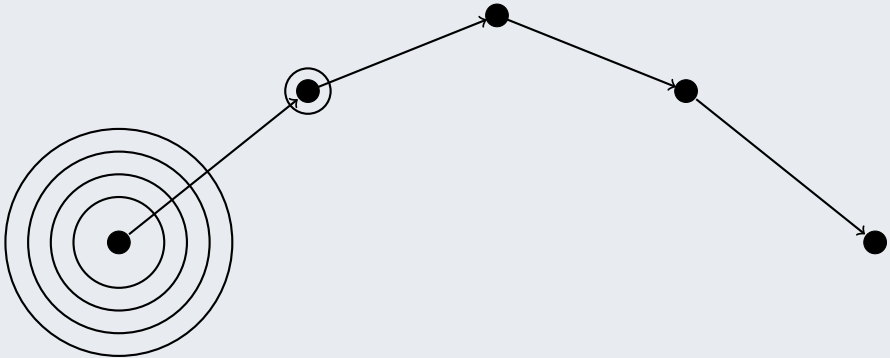
Proste przypadki

- Możemy wyrzucić z grafu wszystkie słupki, do których nie da się dojść z 1.
- I słupki, z których nie da się dojść do N.
- Jeśli z 1 nie da się dojść do N, to odpowiedzią jest 0.
- W przeciwnym razie, widać, że da się liczba wierzchołków połączonych zarówno z 1 jak i N, minus 1. Czy da się lepiej?

Przykład 2

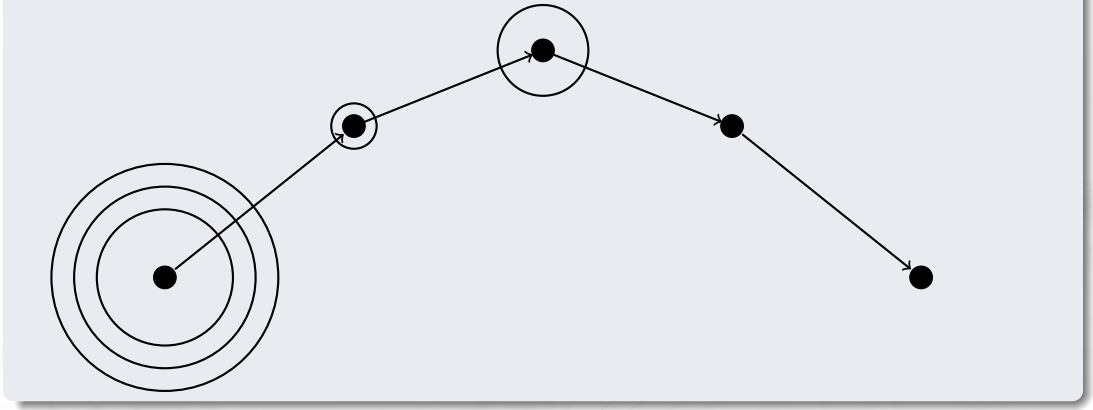


Przykład 2



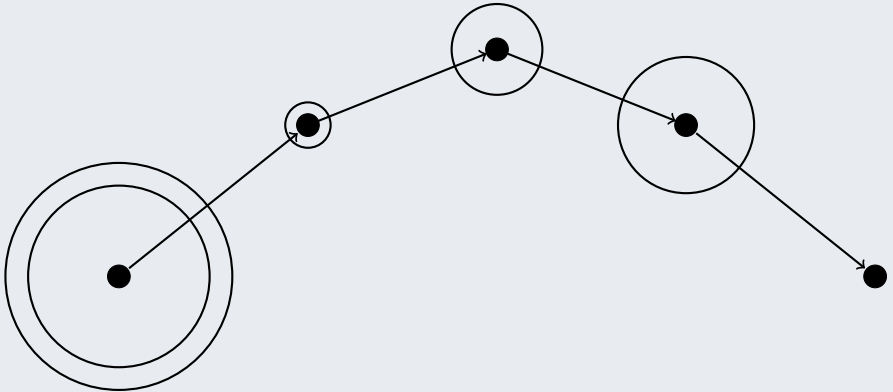
Skierowane Hanoi

Przykład 2



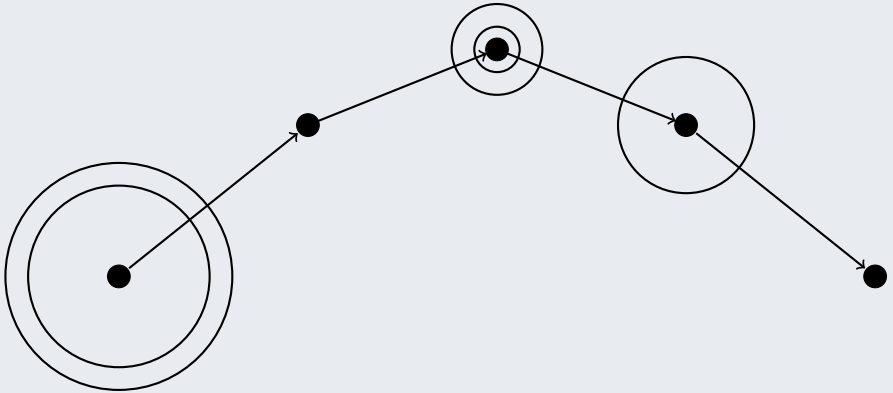
Skierowane Hanoi

Przykład 2

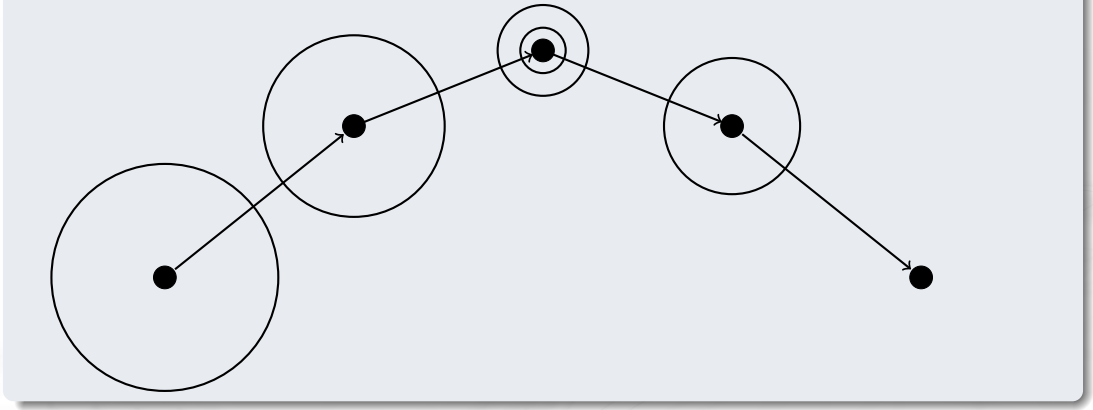


Skierowane Hanoi

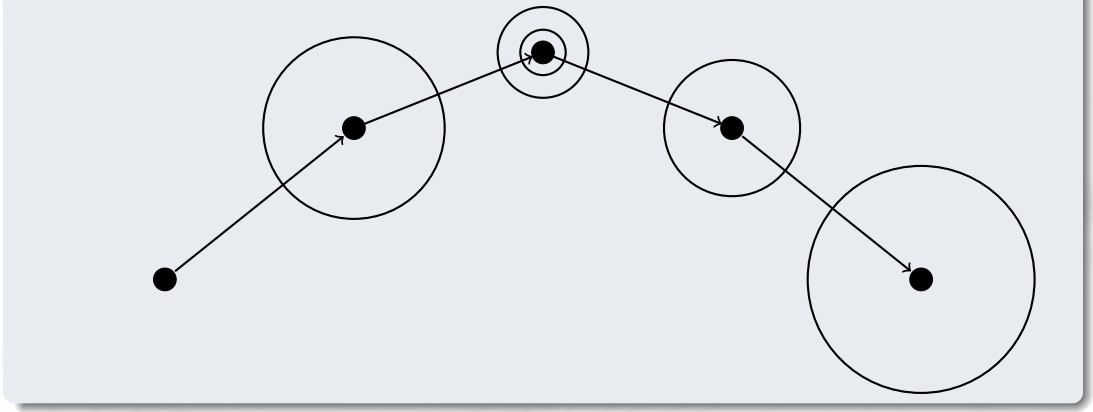
Przykład 2



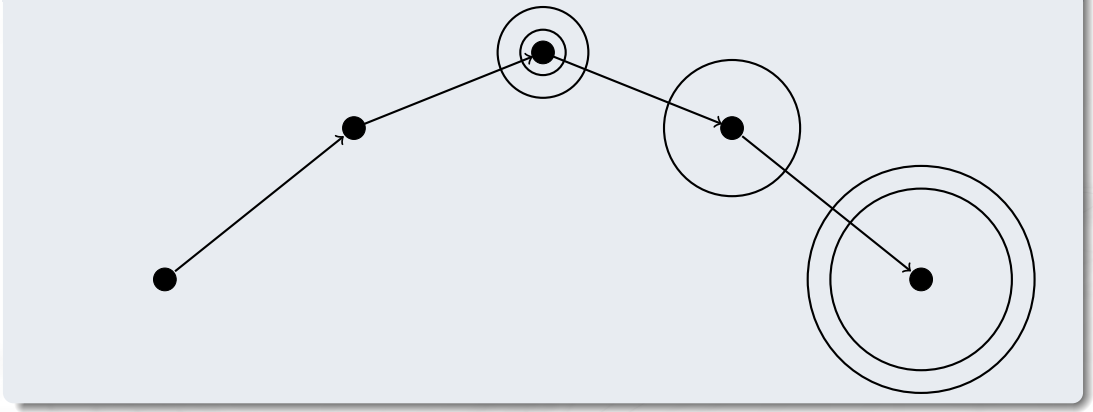
Przykład 2



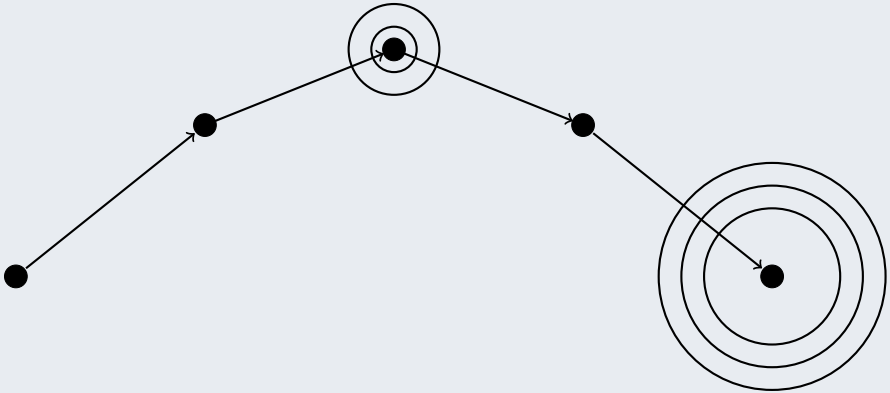
Przykład 2



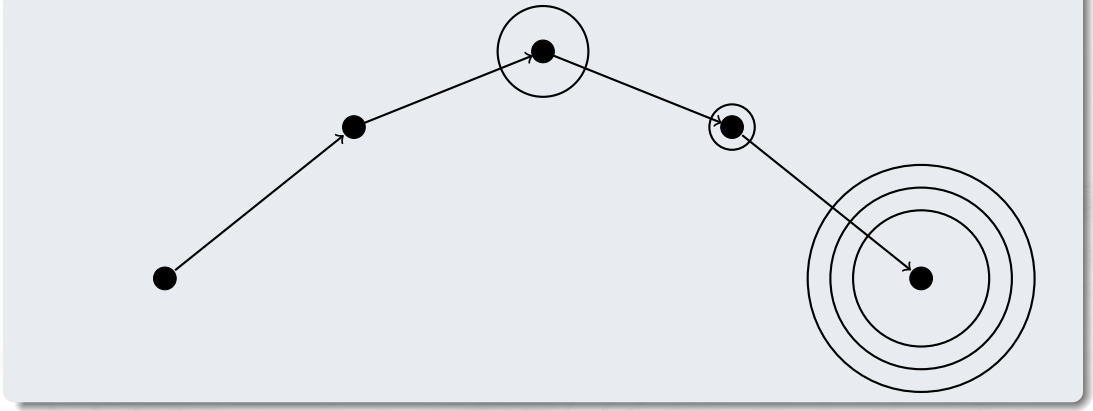
Przykład 2



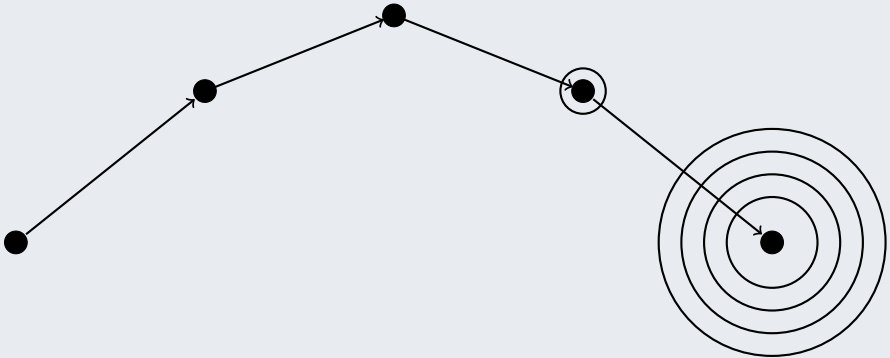
Przykład 2



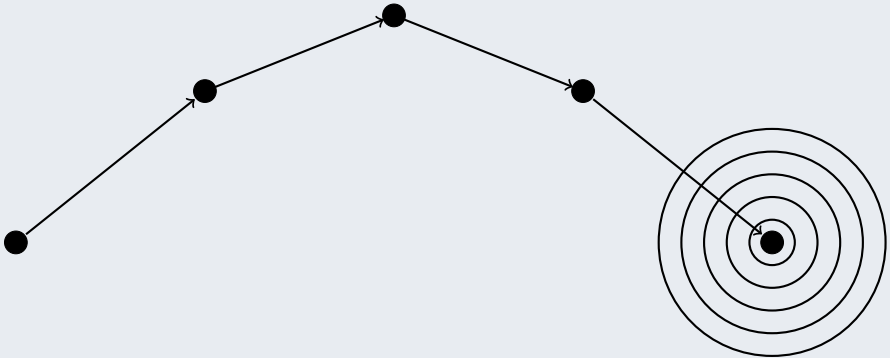
Przykład 2



Przykład 2



Przykład 2



Szkic rozwiązania

Przykład pokazał kluczową ideę rozwiazania. Aby przenieść K krążków z A na B:

- Wybieramy słupek C gdzieś pośrodku
- Przenosimy x krążków z A na C
- Przenosimy pozostałe $K - x$ krążków z A na B nie wykorzystując C
- Przenosimy x krążków z C na B

Jak wybrać C oraz x?

Szkic rozwiązania

Jeśli znamy C, to wybór x jest prosty — chcemy jak największe. Dokładniej:

- Musimy być w stanie przesunąć x krążków z A na C
- Musimy być w stanie przesunąć x krążków z C na B

Niech $f(A, B)$ oznacza największą liczbę krążków, które jesteśmy w stanie przesunąć z A na B. To teraz będziemy chcieli wziąć $x = \min\{f(A, C), f(C, B)\}$.

Skierowane Hanoi

Szkic rozwiązania

To, co wydaje się trudne, to zrozumienie, jak usunięcie C wpłynie na to, ile możemy przesunąć krążków z A do B.

Skierowane Hanoi

Szkic rozwiązania

To, co wydaje się trudne, to zrozumienie, jak usunięcie C wpłynie na to, ile możemy przesunąć krążków z A do B. Otóż prawie wcale nie wpłynie.

Szkic rozwiązania

Jak wygląda przesuwanie klocków z A do B z pominięciem C? Znowu, wybieramy D pomiędzy A i B tak, by $x' = \min\{f'(A, D), f'(D, B)\}$ było jak największe, gdzie f' to liczba krążków, które możemy przesunąć w grafie z usuniętym C.

Szkic rozwiązania

Jak wygląda przesuwanie klocków z A do B z pominięciem C? Znowu, wybieramy D pomiędzy A i B tak, by $x' = \min\{f'(A, D), f'(D, B)\}$ było jak największe, gdzie f' to liczba krążków, które możemy przesunąć w grafie z usuniętym C.

Pokażemy, że usunięcie C nic nie zmieniło, czyli że

$$\min\{f'(A, D), f'(D, B)\} = \min\{f(A, D), f(D, B)\}.$$

Skierowane Hanoi

Szkic rozwiązania

- Graf jest acykliczny, więc istnieje tylko jedna ze ścieżek $C \to D$ lub $D \to C$, powiedzmy, że ta pierwsza.

Skierowane Hanoi

Szkic rozwiązania

- Graf jest acykliczny, więc istnieje tylko jedna ze ścieżek $C \to D$ lub $D \to C$, powiedzmy, że ta pierwsza.
- Jeśli nie istnieje żadna, to usunięcie C nie wpływa na przenoszenie klocków z ani do D. Powiedzmy więc, że istnieje tylko ścieżka $C \to D$.

Szkic rozwiązania

- Graf jest acykliczny, więc istnieje tylko jedna ze ścieżek $C \to D$ lub $D \to C$, powiedzmy, że ta pierwsza.
- Jeśli nie istnieje żadna, to usunięcie C nie wpływa na przenoszenie klocków z ani do D. Powiedzmy więc, że istnieje tylko ścieżka $C \to D$.
- Wobec tego, $f'(D, B) = f(D, B) - C$ nie znajduje się na żadnej ścieżce z D do B, więc nie wykorzystujemy go przekładając krążki z D na B, czyli usunięcie go nic nie zmienia.

Szkic rozwiązania

- Graf jest acykliczny, więc istnieje tylko jedna ze ścieżek $C \to D$ lub $D \to C$, powiedzmy, że ta pierwsza.
- Jeśli nie istnieje żadna, to usunięcie C nie wpływa na przenoszenie klocków z ani do D. Powiedzmy więc, że istnieje tylko ścieżka $C \to D$.
- Wobec tego, $f'(D, B) = f(D, B)$ — C nie znajduje się na żadnej ścieżce z D do B, więc nie wykorzystujemy go przekładając krążki z D na B, czyli usunięcie go nic nie zmienia.
- Z drugiej strony, $f'(A, D) \geq f(A, C)$. Jeśli mamy strategię przeniesienia $f(A, C)$ krążków z A na C, to dokładnie tą samą strategią możemy przenieść te same klocki na D, nie używając C — kiedy mamy przełożyć klocek na C, to zamiast tego przekładamy go na D.

Szkic rozwiązania

- Graf jest acykliczny, więc istnieje tylko jedna ze ścieżek $C \to D$ lub $D \to C$, powiedzmy, że ta pierwsza.
- Jeśli nie istnieje żadna, to usunięcie C nie wpływa na przenoszenie klocków z ani do D. Powiedzmy więc, że istnieje tylko ścieżka $C \to D$.
- Wobec tego, $f'(D, B) = f(D, B)$ — C nie znajduje się na żadnej ścieżce z D do B, więc nie wykorzystujemy go przekładając krążki z D na B, czyli usunięcie go nic nie zmienia.
- Z drugiej strony, $f'(A, D) \geq f(A, C)$. Jeśli mamy strategię przeniesienia $f(A, C)$ krążków z A na C, to dokładnie tą samą strategią możemy przenieść te same klocki na D, nie używając C — kiedy mamy przełożyć klocek na C, to zamiast tego przekładamy go na D.
- Stąd, $f'(A, D) \geq f(A, C) \geq \min\{f(A, C), f(C, B)\} \geq \min\{f(A, D), f(D, B)\}$.

Szkic rozwiązania

Jeśli usuwamy wierzchołki pomiędzy A i B w kolejności od największego $\min\{f(A,C), f(C,B)\}$, to $\min\{f(A,D), f(D,B)\}$ dla pozostałych D się nie będzie zmieniać.

Łącznie z A do B przepchniemy $\sum_C \min\{f(A,C), f(C,B)\} + 1$ krążków (to $+1$ to ostatni krążek, który przesuniemy bezpośrednio po usunięciu wszystkich wierzchołków pomiędzy).

Wartości $f(A,B)$ dla wszystkich A, B możemy policzyć dynamikiem w $O(N^3)$.

Podsumowanie

- $f(A, B) = 1 + \sum_C \min\{f(A, C), f(C, B)\}$ po C takich, że istnieją ścieżki $A \to C$ i $C \to B$.

Podsumowanie

- $f(A, B) = 1 + \sum_C \min\{f(A, C), f(C, B)\}$ po C takich, że istnieją ścieżki $A \to C$ i $C \to B$.
- Więcej się nie da — w momencie, w którym przenosimy największy krążek z A, to wszystkie pozostałe krążki muszą znajdować się na jakichś pośrednich C, i na żadnym C nie może się znajdować więcej niż $f(A, C)$ (bo nie mamy jak ich tam włożyć), ani więcej niż $f(C, B)$, bo ich nie zdejmiemy.

Podsumowanie

- $f(A, B) = 1 + \sum_C \min\{f(A, C), f(C, B)\}$ po C takich, że istnieją ścieżki $A \to C$ i $C \to B$.
- Więcej się nie da — w momencie, w którym przenosimy największy krążek z A, to wszystkie pozostałe krążki muszą znajdować się na jakichś pośrednich C, i na żadnym C nie może się znajdować więcej niż $f(A, C)$ (bo nie mamy jak ich tam włożyć), ani więcej niż $f(C, B)$, bo ich nie zdejmiemy.
- Strategia osiągnięcia tej wartości to przenoszenie krążków na słupki od największego $\min\{f(A, C), f(C, B)\}$.

Podsumowanie

- $f(A, B) = 1 + \sum_C \min\{f(A, C), f(C, B)\}$ po C takich, że istnieją ścieżki $A \to C$ i $C \to B$.
- Więcej się nie da — w momencie, w którym przenosimy największy krążek z A, to wszystkie pozostałe krążki muszą znajdować się na jakichś pośrednich C, i na żadnym C nie może się znajdować więcej niż $f(A, C)$ (bo nie mamy jak ich tam włożyć), ani więcej niż $f(C, B)$, bo ich nie zdejmiemy.
- Strategia osiągnięcia tej wartości to przenoszenie krążków na słupki od największego $\min\{f(A, C), f(C, B)\}$.
- Wartości f wyliczamy dynamikiem w $O(N^3)$.

M. Trefle

Najszybsze rozwiązanie: **Kamil Dębowski (4:47)**

Autor zadania: **Konrad Paluszek**

Trefle

Treść zadania

Algosia ma $a_i, a_2, \ldots$ lub a_k trefli, Bajtek ma $b_1, b_2, \ldots$ lub b_l trefli.

Trefle

Treść zadania

Algosia ma $a_i, a_2, \ldots$ lub a_k trefli, Bajtek ma $b_1, b_2, \ldots$ lub b_l trefli.
Jeśli łącznie mają X lub więcej trefli, optymalny kontrakt to $N + 1$, jeśli mniej — optymalny kontrakt to N.

Trefle

Treść zadania

Algosia ma $a_i, a_2, \ldots$ lub a_k trefli, Bajtek ma $b_1, b_2, \ldots$ lub b_l trefli.
Jeśli łącznie mają X lub więcej trefli, optymalny kontrakt to $N + 1$, jeśli mniej — optymalny kontrakt to N.
Kontrakt osiąga się przez licytację. Algosia i Bajtek ruszają się naprzemiennie. W ruchu gracz:

- proponuje kontrakt (liczba całkowita dodatnia większa niż poprzednio zaproponowana), lub
- akceptuje ostatni zaproponowany kontrakt, co kończy licytację

Pierwszy gracz musi zaproponować jakiś (całkowity dodatni) kontrakt.

Trefle

Treść zadania

Algosia ma $a_i, a_2, \ldots$ lub a_k trefli, Bajtek ma $b_1, b_2, \ldots$ lub b_l trefli.
Jeśli łącznie mają X lub więcej trefli, optymalny kontrakt to $N + 1$, jeśli mniej — optymalny kontrakt to N.
Kontrakt osiąga się przez licytację. Algosia i Bajtek ruszają się naprzemiennie. W ruchu gracz:

- proponuje kontrakt (liczba całkowita dodatnia większa niż poprzednio zaproponowana), lub
- akceptuje ostatni zaproponowany kontrakt, co kończy licytację

Pierwszy gracz musi zaproponować jakiś (całkowity dodatni) kontrakt.
Wygrywają, jeśli uda im się wylicytować optymalny kontrakt.

Trefle

Treść zadania

Możliwe liczby trefli na ręce Algosi i Bajtka, oraz wybrany system licytacji, to wiedza wspólna. Konkretna liczba trefli na ręce gracza jest znana tylko temu graczowi.

Trefle

Treść zadania

Możliwe liczby trefli na ręce Algosi i Bajtka, oraz wybrany system licytacji, to wiedza wspólna. Konkretna liczba trefli na ręce gracza jest znana tylko temu graczowi.
Dla jakiego najmniejszego N istnieje strategia licytacji, która gwarantuje uzyskanie optymalnego kontraktu?

Limity

- $k, l \leq 1\,000$
- $a_i, b_i, X \leq 10^9$

Trefle

Przykład

Algosia ma $1, 5, 7, 10$ lub 12 trefli, Bajtek ma $2, 6$ lub 9. Optymalny kontrakt rośnie przy łącznie 13 treflach na ręku.

Przykład

Algosia ma $1, 5, 7, 10$ lub 12 trefli, Bajtek ma $2, 6$ lub 9. Optymalny kontrakt rośnie przy łącznie 13 treflach na ręku.

	1	5	7	10	12
2					
6					
9					

Przykład

Algosia ma $1, 5, 7, 10$ lub 12 trefli, Bajtek ma $2, 6$ lub 9. Optymalny kontrakt rośnie przy łącznie 13 treflach na ręku.

	1	5	7	10	12
2	**3**	**7**	**9**	**12**	**14**
6	**7**	**11**	**13**	**16**	**18**
9	**10**	**14**	**16**	**19**	**21**

Przykład

W tym przykładzie, w treści zadania, podany jest przykład strategii licytacyjnej działającej dla $N = 3$. Prześledźmy go, jeśli Algosia ma 7 trefli, a Bajtek — 6.

	1	5	7	10	12
2	**3**	**7**	**9**	**12**	**14**
6	**7**	**11**	**13**	**16**	**18**
9	**10**	**14**	**16**	**19**	**21**

Przykład

W tym przykładzie, w treści zadania, podany jest przykład strategii licytacyjnej działającej dla $N = 3$. Prześledźmy go, jeśli Algosia ma 7 trefli, a Bajtek — 6.

	1	5	7	10	12
2		**7**	**9**	**12**	
6		**11**	**13**	**16**	
9		**14**	**16**	**19**	

Przykład

W tym przykładzie, w treści zadania, podany jest przykład strategii licytacyjnej działającej dla $N = 3$. Prześledźmy go, jeśli Algosia ma 7 trefli, a Bajtek — 6.

	1	5	7	10	12
2					
6		11	13	16	
9					

Przykład

W tym przykładzie, w treści zadania, podany jest przykład strategii licytacyjnej działającej dla $N = 3$. Prześledźmy go, jeśli Algosia ma 7 trefli, a Bajtek — 6.

	1	5	7	10	12
2					
6			13		
9					

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.
- Jeśli w jakiejś kolumnie są same zielone liczby, możemy od razu powiedzieć $N + 1$.
- Czyli optymalny kontrakt nie zmieni się, jeśli dodamy lub usuniemy taką kolumnę.

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.
- Jeśli w jakiejś kolumnie są same zielone liczby, możemy od razu powiedzieć $N + 1$.
- Czyli optymalny kontrakt nie zmieni się, jeśli dodamy lub usuniemy taką kolumnę.
  - Dlaczego to nie działa dla całych czerwonych kolumn?
  - Bo istnieje potencjalnie sensowna odpowiedź na propozycję N inna niż zaakceptowanie, a nie istnieje inna sensowna odpowiedź na propozycję $N + 1$.

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.
- Jeśli w jakiejś kolumnie są same zielone liczby, możemy od razu powiedzieć $N + 1$.
- Czyli optymalny kontrakt nie zmieni się, jeśli dodamy lub usuniemy taką kolumnę.
- W przeciwnym razie nie możemy powiedzieć $N + 1$ (bo wynikiem może być N).

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.
- Jeśli w jakiejś kolumnie są same zielone liczby, możemy od razu powiedzieć $N + 1$.
- Czyli optymalny kontrakt nie zmieni się, jeśli dodamy lub usuniemy taką kolumnę.
- W przeciwnym razie nie możemy powiedzieć $N + 1$ (bo wynikiem może być N).
- Wobec tego, jeśli w jakiejś kolumnie są same zielone liczby, to nasz partner będzie mógł zaproponować $N + 1$.
- Czyli możemy też usunąć wszystkie kolumny, które są całe zielone.

Wstępne redukcje

- Wartości liczbowe w tej tabeli nie są istotne — liczą się tylko kolory.
- Jeśli w jakiejś kolumnie są same zielone liczby, możemy od razu powiedzieć $N + 1$.
- Czyli optymalny kontrakt nie zmieni się, jeśli dodamy lub usuniemy taką kolumnę.
- W przeciwnym razie nie możemy powiedzieć $N + 1$ (bo wynikiem może być N).
- Wobec tego, jeśli w jakiejś kolumnie są same zielone liczby, to nasz partner będzie mógł zaproponować $N + 1$.
- Czyli możemy też usunąć wszystkie kolumny, które są całe zielone.
- Jeśli są dwie takie same wiersze, lub dwie takie same kolumny, to możemy je utożsamić (grać dla nich tą samą strategią, i wszystko jedno, który z nich jest prawdą).

Tabela po redukcjach — przykład

	1	5	7	10	12
2	**3**	**7**	**9**	**12**	**14**
6	**7**	**11**	**13**	**16**	**18**
9	**10**	**14**	**16**	**19**	**21**

Tabela po redukcjach — przykład

	1	5	7	10	12
2	**3**	**7**		**12**	
6	**7**	**11**		**16**	
9	**10**	**14**		**19**	

Tabela po redukcjach — przykład

	1	5	7	10	12
2	**3**	**7**		**12**	
6	**7**	**11**		**16**	
9	**10**	**14**		**19**	

Tabela po redukcjach:

- Zawsze będzie kwadratowa
- Pierwsza kolumna i pierwszy wiersz będą całe czerwone
- Rozmiar tabeli jednoznacznie wyznacza zawartość.

Strategia — małe tabele

- Niech $R(x)$ oznacza maksymalną liczbę wierszy taką, że istnieje optymalna strategia rozróżniająca kontrakt x od $x+1$ dla tabeli $R(x) \times R(x)$ (po redukcji).

Strategia — małe tabele

- Niech $R(x)$ oznacza maksymalną liczbę wierszy taką, że istnieje optymalna strategia rozróżniająca kontrakt x od $x+1$ dla tabeli $R(x) \times R(x)$ (po redukcji).
- $R(0) = 0$. Algosia musi zaproponować przynajmniej 1, więc strategia istnieje tylko wtedy, gdy po redukcji nie zostaną żadne wiersze (czyli tabela jest cała zielona).

Strategia — małe tabele

- Niech $R(x)$ oznacza maksymalną liczbę wierszy taką, że istnieje optymalna strategia rozróżniająca kontrakt x od $x+1$ dla tabeli $R(x) \times R(x)$ (po redukcji).
- $R(0) = 0$. Algosia musi zaproponować przynajmniej 1, więc strategia istnieje tylko wtedy, gdy po redukcji nie zostaną żadne wiersze (czyli tabela jest cała zielona).
- $R(1) = 1$. Dla tabeli 1×1, Algosia proponuje 1 i Bajtek akceptuje. Dla tabeli 2×2 już się nie uda:
  - Algosia nie może zaryzykować powiedzenia na początku 2 (bo Bajtek może być w całym czerwonym wierszu)
  - Bajtek po usłyszeniu 1, jeśli jest w mieszanym wierszu, nie wie, czy podbić do 2.

Możliwości Bajtka

Powiedzmy, że Algosia i Bajtek grają próbując odróżnić optymalny kontrakt x od $x+1$, na tabeli $a \times a$. Algosia proponuje $x - y$, co Bajtek wie, że Algosia zrobi dla pewnej grupy b kolumn (w zredukowanej tabeli). Jak wygląda teraz sytuacja Bajtka?

Możliwości Bajtka

Powiedzmy, że Algosia i Bajtek grają próbując odróżnić optymalny kontrakt x od $x + 1$, na tabeli $a \times a$. Algosia proponuje $x - y$, co Bajtek wie, że Algosia zrobi dla pewnej grupy b kolumn (w zredukowanej tabeli). Jak wygląda teraz sytuacja Bajtka?

- Bajtek może usunąć z tabeli wszystkie kolumny poza b, a następnie zredukować tabelę (do tabeli $b \times b$).

Możliwości Bajtka

Powiedzmy, że Algosia i Bajtek grają próbując odróżnić optymalny kontrakt x od $x+1$, na tabeli $a \times a$. Algosia proponuje $x-y$, co Bajtek wie, że Algosia zrobi dla pewnej grupy b kolumn (w zredukowanej tabeli). Jak wygląda teraz sytuacja Bajtka?

- Bajtek może usunąć z tabeli wszystkie kolumny poza b, a następnie zredukować tabelę (do tabeli $b \times b$).
- Jeśli $y = 0$, to jedyną opcją Bajtka jest zaakceptować (nie może powiedzieć $x+1$ w zredukowanej tabeli, bo każdy wiersz zawiera czerwone pola), czyli $b = 1$.

Możliwości Bajtka

Powiedzmy, że Algosia i Bajtek grają próbując odróżnić optymalny kontrakt x od $x+1$, na tabeli $a \times a$. Algosia proponuje $x-y$, co Bajtek wie, że Algosia zrobi dla pewnej grupy b kolumn (w zredukowanej tabeli). Jak wygląda teraz sytuacja Bajtka?

- Bajtek może usunąć z tabeli wszystkie kolumny poza b, a następnie zredukować tabelę (do tabeli $b \times b$).
- Jeśli $y = 0$, to jedyną opcją Bajtka jest zaakceptować (nie może powiedzieć $x+1$ w zredukowanej tabeli, bo każdy wiersz zawiera czerwone pola), czyli $b = 1$.
- Jeśli $y = 1$, to Bajtek może wyłącznie powiedzieć x (nie może powiedzieć $x+1$, jak wyżej, i nie może zaakceptować, bo $x-1$ nigdy nie jest optymalne). Więc znowu $b = 1$.

Możliwości Bajtka

Powiedzmy, że Algosia i Bajtek grają próbując odróżnić optymalny kontrakt x od $x+1$, na tabeli $a \times a$. Algosia proponuje $x-y$, co Bajtek wie, że Algosia zrobi dla pewnej grupy b kolumn (w zredukowanej tabeli). Jak wygląda teraz sytuacja Bajtka?

- Bajtek może usunąć z tabeli wszystkie kolumny poza b, a następnie zredukować tabelę (do tabeli $b \times b$).
- Jeśli $y = 0$, to jedyną opcją Bajtka jest zaakceptować (nie może powiedzieć $x+1$ w zredukowanej tabeli, bo każdy wiersz zawiera czerwone pola), czyli $b = 1$.
- Jeśli $y = 1$, to Bajtek może wyłącznie powiedzieć x (nie może powiedzieć $x+1$, jak wyżej, i nie może zaakceptować, bo $x-1$ nigdy nie jest optymalne). Więc znowu $b = 1$.
- A ogólniej, dla $y > 0$, Bajtek musi powiedzieć coś większego od $x-y$. Czyli sytuacja Bajtka jest taka, jakby grał jako pierwszy na tabeli $b \times b$ próbując odróżnić y od $y+1$. Czyli $b \leq R(y)$.

Ruchy Algosi

Zatem Algosia może:

- Powiedzieć x dla jednej kolumny,
- Powiedzieć $x-1$ dla $R(1)=1$ kolumny,
- Powiedzieć $x-2$ dla $R(2)$ kolumn, ...
- Powiedzieć 1 dla $R(x-1)$ kolumn.

Zatem $R(x)=1+R(1)+R(2)+\ldots+R(x-1)$, czyli $R(x)=2^{x-1}$.

Przykład

Prześledźmy to na przykładzie.
$X = 13,\ a = (1, 2, 5, 7, 10, 12),\ b = (2, 5, 6, 9)$.

Przykład

Prześledźmy to na przykładzie.

	1	2	5	7	10	12
2						
5						
6						
9						

Przykład

Prześledźmy to na przykładzie.

	1	2	5	7	10	12
2	3	4	7	9	12	14
5	6	7	10	12	15	17
6	7	8	11	13	16	18
9	10	11	14	16	19	21

Przykład

Redukujemy kolumy 12 (Algosia od razu mówi $N+1$) i 1 (Algosia gra tak samo jak dla kolumy 2) i dostajemy tabelę 4×4.

	2	5	7	10
2	**4**	**7**	**9**	**12**
5	**7**	**10**	**12**	**15**
6	**8**	**11**	**13**	**16**
9	**11**	**14**	**16**	**19**

Przykład

$R(3) = 4$, czyli gracze chcą odróżnić 3 od 4. Algosia mówi 1 dla dowolnych dwóch kolumn (powiedzmy 5 i 10), 2 dla jednej kolumny (powiedzmy 2) i 3 dla jednej kolumny (powiedzmy 7).
Co robi Bajtek jeśli Algosia powie 1?

	2	5	7	10
2	**4**	**7**	**9**	**12**
5	**7**	**10**	**12**	**15**
6	**8**	**11**	**13**	**16**
9	**11**	**14**	**16**	**19**

Przykład

Co robi Bajtek jeśli Algosia powie 1 (dla kolumn 5 i 10)?

- Usuwa kolumny 2 i 7.

		5		10
2		7		12
5		10		15
6		11		16
9		14		19

Przykład

Co robi Bajtek jeśli Algosia powie 1 (dla kolumn 5 i 10)?

- Usuwa kolumny 2 i 7.
- Redukuje wiersz 9 (odpowiada 4) i wiersz 5 (gram tak samo jak dla wiersza 6).

		5		10
2		**7**		**12**
6		**11**		**16**

Przykład

Co robi Bajtek jeśli Algosia powie 1 (dla kolumn 5 i 10)?

- Usuwa kolumny 2 i 7.
- Redukuje wiersz 9 (odpowiada 4) i wiersz 5 (gram tak samo jak dla wiersza 6).
- Gra jak w tabeli 2×2 — mówi 2 dla jednego z wierszy (powiedzmy 5) i 3 dla drugiego (powiedzmy 2).

	5	10
2	**7**	**12**
6	**11**	**16**

Podsumowanie

- Tworzymy tablicę boolowską $k \times l$, wstawiając w pole (i, j) wartość $a_i + b_j \geq X$
- Usuwamy wszystkie wiersze i kolumny z samych jedynek
- Usuwamy wszystkie powtarzające się wiersze i kolumny (będą sąsiadować)
- Jeśli nic nie zostało, zwracamy 0
- Jeśli zostało x wierszy, zwracamy $\lceil \log_2 x \rceil + 1$