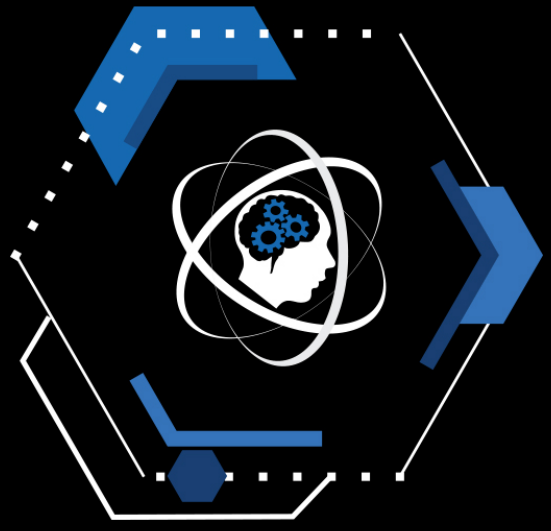
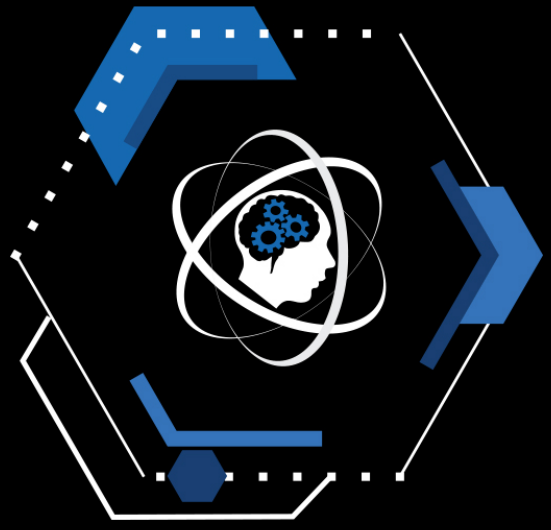


Omówienie zadań

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE



Sumka

Najszybsze rozwiązanie: **Marek Sokołowski (0:13)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Przedstaw liczbę naturalną jako sumę co najwyżej 5 liczb trójkątnych.

Twierdzenie [Gauss]

Każdą liczbę naturalną da się przedstawić jako sumę co najwyżej trzech liczb trójkątnych.

Twierdzenie [Gauss]

Każdą liczbę naturalną da się przedstawić jako sumę co najwyżej trzech liczb trójkątnych.

Obserwacja

Niech $T(n)$ oznacza największą liczbę trójkątną nie większą od n. Wówczas $n - T(n) < \sqrt{2n}$.

Twierdzenie [Gauss]

Każdą liczbę naturalną da się przedstawić jako sumę co najwyżej trzech liczb trójkątnych.

Obserwacja

Niech $T(n)$ oznacza największą liczbę trójkątną nie większą od n. Wówczas $n - T(n) < \sqrt{2n}$.

- Korzystając z obserwacji, możemy zachłannie dwiema liczbami trójkątnymi zbić n do $2^{\frac{3}{4}} n^{\frac{1}{4}} < 100000$.

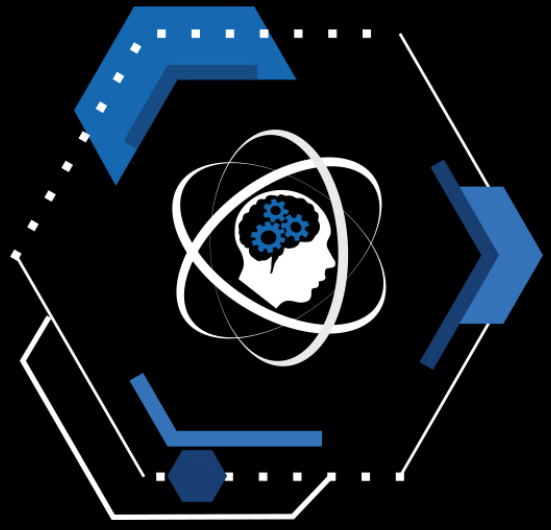
Twierdzenie [Gauss]

Każdą liczbę naturalną da się przedstawić jako sumę co najwyżej trzech liczb trójkątnych.

Obserwacja

Niech $T(n)$ oznacza największą liczbę trójkątną nie większą od n. Wówczas $n - T(n) < \sqrt{2n}$.

- Korzystając z obserwacji, możemy zachłannie dwiema liczbami trójkątnymi zbić n do $2^{\frac{3}{4}} n^{\frac{1}{4}} < 100000$.
- Przedstawienie liczby m jako sumę co najwyżej trzech liczb trójkątnych można policzyć w czasie $O(m)$ sprawdzając wszystkich kandydatów na dwie z trzech liczb trójkątnych.



Handel

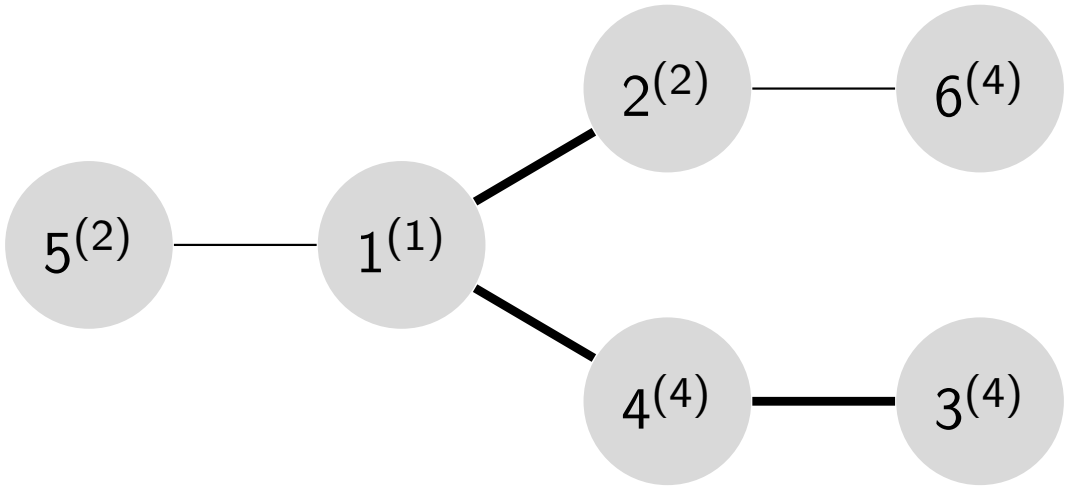
Najszybsze rozwiązanie: **Krzysztof Maziarz (0:13)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Dane jest drzewo o wierzchołkach numerowanych od 1 do n. Wierzchołek i ma kolor k_i. Znajdź najdłuższą ścieżkę między wierzchołkami różnych kolorów.



Rozwiązanie 1.

- Programowanie dynamiczne – idziemy po drzewie od liści do korzenia.

Rozwiązanie 1.

- Programowanie dynamiczne – idziemy po drzewie od liści do korzenia.
- Chcemy "znaleźć" poszukiwaną ścieżkę będąc w jej najwyższym punkcie v

Rozwiązanie 1.

- Programowanie dynamiczne – idziemy po drzewie od liści do korzenia.
- Chcemy "znaleźć" poszukiwaną ścieżkę będąc w jej najwyższym punkcie v
- Końcami zatem będą najdłuższa ścieżka w poddrzewie v w jakimś kolorze k_1, i najdłuższa ścieżka w innym poddrzewie v w jakimś innym kolorze k_2

Rozwiązanie 1.

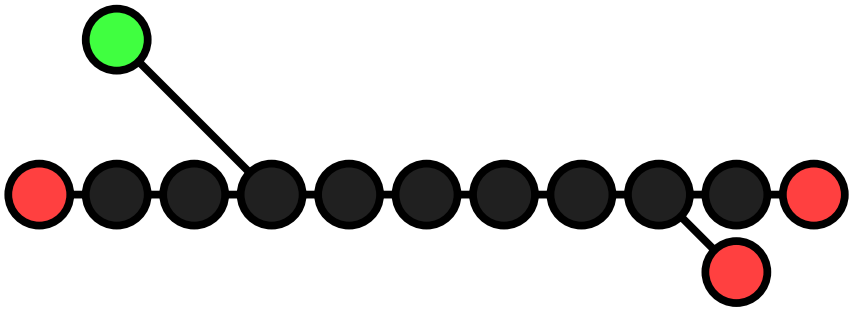
- Programowanie dynamiczne – idziemy po drzewie od liści do korzenia.
- Chcemy "znaleźć" poszukiwaną ścieżkę będąc w jej najwyższym punkcie v
- Końcami zatem będą najdłuższa ścieżka w poddrzewie v w jakimś kolorze k_1, i najdłuższa ścieżka w innym poddrzewie v w jakimś innym kolorze k_2
- Wystarczy, że każdy syn przekazuje ojcu najdłuższe ścieżki w dwóch różnych kolorach

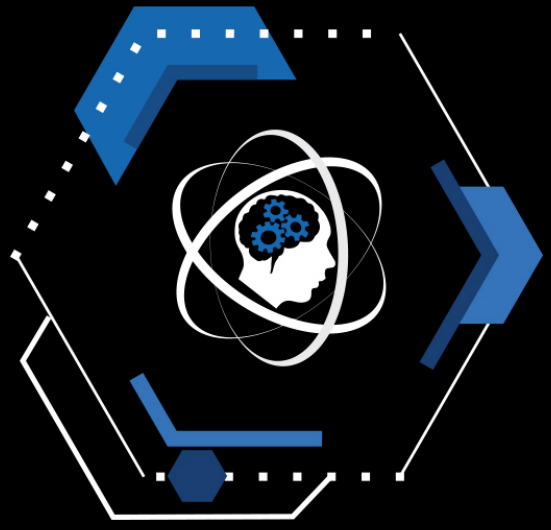
Rozwiązanie 2.

- Znajdujemy *średnicę* drzewa, czyli najdłuższą ścieżkę
  - Zaczynamy w dowolnym wierzchołku v
  - Znajdujemy w najdalszy od v
  - Znajdujemy x najdalszy od w
  - Średnicą jest ścieżka od w do x

Rozwiązanie 2.

- Znajdujemy *średnicę* drzewa, czyli najdłuższą ścieżkę
  - Zaczynamy w dowolnym wierzchołku v
  - Znajdujemy w najdalszy od v
  - Znajdujemy x najdalszy od w
  - Średnicą jest ścieżka od w do x
- Jeśli końce v_1 i v_2 są różnych kolorów, to wygraliśmy
- W przeciwnym wypadku znajdujemy najdalszy wierzchołek innego koloru od v_i dla $i = 1, 2$
- Dłuższa ze znalezionych ścieżek to poszukiwana ścieżka





System dwudziesiętny

Najszybsze rozwiązanie: **Mateusz Radecki (0:13)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

$f(N)$ to liczba sposobów zapisu N w systemie *dwu-dziesiętnym*, czyli systemie binarnym z cyframi 0–9. Dla danego L i R, mamy policzyć sumę:

$$\sum_{i=L}^{R} f(i)$$

Zajmijmy się najpierw policzeniem pojedynczego $f(N)$ w czasie wielomianowym od $\log N$.

- Zapiszmy N w systemie binarnym.
- Dynamik od najbardziej znaczących cyfr.
- Ustalamy każdą kolejną cyfrę i trzymamy różnicę: prefiks w binarnym minus prefiks w dwu-dziesiętnym.

- Zapiszmy N w systemie binarnym.
- Dynamik od najbardziej znaczących cyfr.
- Ustalamy każdą kolejną cyfrę i trzymamy różnicę: prefiks w binarnym minus prefiks w dwu-dziesiętnym.

Przykład

1101...
0023...

Binarny: $8 + 4 + 1 = 13$.
Dwu-dziesiętny: $4 + 3 = 7$.
Różnica: $13 - 7 = 6$.

- Zapiszmy N w systemie binarnym.
- Dynamik od najbardziej znaczących cyfr.
- Ustalamy każdą kolejną cyfrę i trzymamy różnicę: prefiks w binarnym minus prefiks w dwu-dziesiętnym.

Przykład

1101...
0023...

Binarny: $8 + 4 + 1 = 13$.
Dwu-dziesiętny: $4 + 3 = 7$.
Różnica: $13 - 7 = 6$.

Lemat

Różnica musi być w przedziale $[0, 9]$.
Inaczej byłaby za duża lub za mała, by „nadgonić" dalszymi cyframi.

Złożoność

Wystarczy zatem $dp[i][roznica]$, gdzie $i < \log N$ i $roznica \leq 9$. By przejść dalej w prawo, iterujemy kolejną wstawianą cyfrę.
Złożoność to $\mathcal{O}(\log N \cdot 9^2)$.

Dla przedziału

$f(L, R) = f(1, R) - f(1, L - 1)$
Musimy umieć policzyć sumę $f(i)$ po i mniejszych niż R.

Dla przedziału

$f(L, R) = f(1, R) - f(1, L - 1)$
Musimy umieć policzyć sumę $f(i)$ po i mniejszych niż R.

Musimy teraz iterować też cyfrę w systemie binarnym. Stan dynamika to $dp[i][roznica][b]$, gdzie boolowe b mówi, czy mamy już liczbę binarną mniejszą niż R.

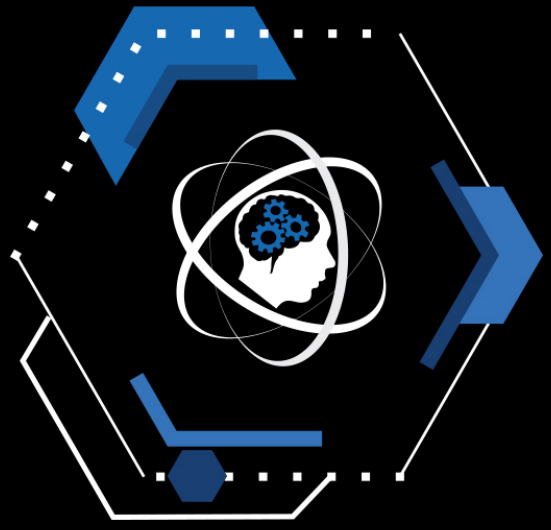
Dla przedziału

$f(L, R) = f(1, R) - f(1, L - 1)$
Musimy umieć policzyć sumę $f(i)$ po i mniejszych niż R.

Musimy teraz iterować też cyfrę w systemie binarnym. Stan dynamika to $dp[i][roznica][b]$, gdzie boolowe b mówi, czy mamy już liczbę binarną mniejszą niż R.

Złożoność

Wciąż jest to $\mathcal{O}(\log N \cdot 9^2)$.



Chochlik

Najszybsze rozwiązanie: **Maciej Wawro (2:06)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Jest ustalony string binarny s długości 100, którego nie znamy. Możemy zadawać zapytania w postaci stringa długości 100. Jeśli jest on równy s, to wygrywamy. W przeciwnym wypadku otrzymujemy odpowiedź w postaci liczby pozycji na których ten string oraz s się nie różnią. Jednak po każdym takim (niewygranym) zapytaniu losowana jest liczba i ze zbioru $\{0, 1, \ldots, 99\}$ i w przypadku gdy string z zapytania oraz s są na i-tej pozycji równe, to $s[i]$ się zmienia na swoją negację.

Wygraj w co najwyżej 222 zapytaniach.

Nie taki chochlik straszny

Jeżeli próbujemy zwiększać liczbę poprawnych pozycji, to Chochlik nam przeszkadza. Jeśli zamiast tego będziemy próbować zmniejszać liczbę poprawnych pozycji, to nam pomaga!

Nie taki chochlik straszny

Jeżeli próbujemy zwiększać liczbę poprawnych pozycji, to Chochlik nam przeszkadza. Jeśli zamiast tego będziemy próbować zmniejszać liczbę poprawnych pozycji, to nam pomaga!

Strategia prawie dobra

Zgadujmy cały czas imię złożone z samych zer. Jeśli dostaniemy odpowiedź 0, to znaczy, że prawdziwym imieniem jest napis złożony z samych jedynek. W przeciwnym wypadku z prawdopodobieństwem $\frac{\text{liczba zer w obecnym imieniu}}{100}$ liczba zer w imieniu chochlika się zmniejszy. Więc po oczekiwanie co najwyżej $100 \cdot \log(100)$ zapytaniach imię chochlika będzie złożone z samych jedynek.

Nie taki chochlik straszny

Jeżeli próbujemy zwiększać liczbę poprawnych pozycji, to Chochlik nam przeszkadza. Jeśli zamiast tego będziemy próbować zmniejszać liczbę poprawnych pozycji, to nam pomaga!

Strategia prawie dobra

Zgadujmy cały czas imię złożone z samych zer. Jeśli dostaniemy odpowiedź 0, to znaczy, że prawdziwym imieniem jest napis złożony z samych jedynek. W przeciwnym wypadku z prawdopodobieństwem $\frac{\text{liczba zer w obecnym imieniu}}{100}$ liczba zer w imieniu chochlika się zmniejszy. Więc po oczekiwanie co najwyżej $100 \cdot \log(100)$ zapytaniach imię chochlika będzie złożone z samych jedynek.

Trzeba lepiej

$100 \cdot \log(100) > 222$.

Załóżmy że mamy już ustalony prefiks $a_1 a_2 \ldots a_p$. Jeśli pytając o $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ dostajemy odpowiedź w_0, a pytając o $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ odpowiedź w_1, to możemy powiedzieć o w_0 i w_1 następujące rzeczy:

Załóżmy że mamy już ustalony prefiks $a_1 a_2 \ldots a_p$. Jeśli pytając o $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ dostajemy odpowiedź w_0, a pytając o $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ odpowiedź w_1, to możemy powiedzieć o w_0 i w_1 następujące rzeczy:

- $|w_0 - w_1| \leq 2$

Załóżmy że mamy już ustalony prefiks $a_1 a_2 \ldots a_p$. Jeśli pytając o $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ dostajemy odpowiedź w_0, a pytając o $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ odpowiedź w_1, to możemy powiedzieć o w_0 i w_1 następujące rzeczy:

- $|w_0 - w_1| \leq 2$
- Jeśli $w_0 > w_1$, to w chwili zadawania pierwszego pytania na pozycji $p+1$-wszej stało 0 oraz chochlik po pierwszym pytaniu nie wylosował liczby $p+1$. Czyli w szczególności po drugim pytaniu wciąż na pozycji $p+1$-wszej jest 0.

Załóżmy że mamy już ustalony prefiks $a_1 a_2 \ldots a_p$. Jeśli pytając o $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ dostajemy odpowiedź w_0, a pytając o $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ odpowiedź w_1, to możemy powiedzieć o w_0 i w_1 następujące rzeczy:

- $|w_0 - w_1| \leq 2$
- Jeśli $w_0 > w_1$, to w chwili zadawania pierwszego pytania na pozycji $p+1$-wszej stało 0 oraz chochlik po pierwszym pytaniu nie wylosował liczby $p+1$. Czyli w szczególności po drugim pytaniu wciąż na pozycji $p+1$-wszej jest 0.
- Jeśli $w_0 \leq w_1$, to znaczy, że w chwili zadania pierwszego pytania na pozycji $p+1$-wszej stała 1.

Załóżmy że mamy już ustalony prefiks $a_1 a_2 \ldots a_p$. Jeśli pytając o $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ dostajemy odpowiedź w_0, a pytając o $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ odpowiedź w_1, to możemy powiedzieć o w_0 i w_1 następujące rzeczy:

- $|w_0 - w_1| \leq 2$
- Jeśli $w_0 > w_1$, to w chwili zadawania pierwszego pytania na pozycji $p+1$-wszej stało 0 oraz chochlik po pierwszym pytaniu nie wylosował liczby $p+1$. Czyli w szczególności po drugim pytaniu wciąż na pozycji $p+1$-wszej jest 0.
- Jeśli $w_0 \leq w_1$, to znaczy, że w chwili zadania pierwszego pytania na pozycji $p+1$-wszej stała 1.

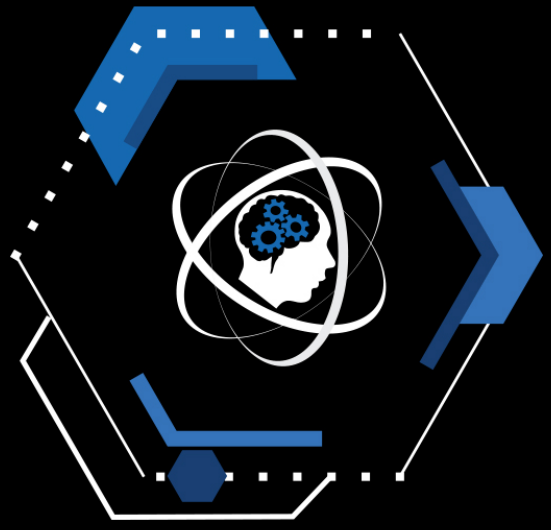
Przedłużenie prefiksu

Pytamy na przemian o stringi $\neg a_0 \neg a_2 \ldots \neg a_p 000 \ldots 0$ i $\neg a_1 \neg a_2 \ldots \neg a_p 100 \ldots 0$ aż dostaniemy że nowa odpowiedź jest mniejsza niż stara. W momencie zatrzymania się w ostatnim stringu o który pytaliśmy na pozycji $p+1$-wszej stoi liczba różna od tej w prawdziwym imieniu.

I to w zasadzie wszystko

- Szansa, że zadamy więcej niż 3 pytania przy wydłużaniu prefiksu o jeden jest bardzo mała.
- Aby oszczędzić zapytań zauważmy, że pierwsze pytanie o nową pozycję jest tożsame z ostatnim pytaniem o starą pozycję, więc dla pozycji $1, 2, \ldots, 99$ oszczędzamy po jednym pytaniu.

W ten sposób zadając $200 + \varepsilon$ pytań dostaniemy flipnięte imię chochlika, ale możemy zauważyć, że ta liczba jest tak naprawdę istotnie mniejsza, bo rzeczy się amortyzują.



Parada

Najszybsze rozwiązanie: **Marek Sokołowski (1:26)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Dane jest $N = 300\,000$ punktów na płaszczyźnie. Należy wybrać $K = 500$ z nich i znaleźć na nich najkrótszy cykl Hamiltona.

Lemat

Każdy ciąg długości N ma podciąg nie-malejący lub podciąg nie-rosnący długości $\sqrt{N}$.

Lemat

Każdy ciąg długości N ma podciąg nie-malejący lub podciąg nie-rosnący długości $\sqrt{N}$.

Dowód

Algorytm szukania najdłuższego podciągu rosnącego z podziałem na kubełki.

Lemat

Każdy ciąg długości N ma podciąg nie-malejący lub podciąg nie-rosnący długości $\sqrt{N}$.

Dowód

Algorytm szukania najdłuższego podciągu rosnącego z podziałem na kubełki.

Wniosek

Da się wybrać ciąg $\sqrt{N}$ punktów, które idą w prawo-górę lub prawo-dół, tzn. każda współrzędna jest monotoniczna.
Zauważmy jeszcze, że $K \leq \sqrt{N}$.

Chcemy znaleźć najkrótszy cykl Hamiltona dla ciągu punktów, gdzie $x_i \leq x_{i+1}$ i $y_i \leq y_{i+1}$.

Chcemy znaleźć najkrótszy cykl Hamiltona dla ciągu punktów, gdzie $x_i \leq x_{i+1}$ i $y_i \leq y_{i+1}$.

Lemat

Nie warto się zawracać, idąc ścieżką od pierwszego do ostatniego punktu, ani potem idąc od ostatniego do pierwszego punktu. Innymi słowy, jeśli mielibyśmy kolejność w stylu $1, 7, 3, 9$, to lepiej posortować te punkty.

Chcemy znaleźć najkrótszy cykl Hamiltona dla ciągu punktów, gdzie $x_i \leq x_{i+1}$ i $y_i \leq y_{i+1}$.

Lemat

Nie warto się zawracać, idąc ścieżką od pierwszego do ostatniego punktu, ani potem idąc od ostatniego do pierwszego punktu. Innymi słowy, jeśli mielibyśmy kolejność w stylu $1, 7, 3, 9$, to lepiej posortować te punkty.

Dowód

Ścieżka od 1 do N przez jakiś zbiór innych punktów nie jest krótsza niż MST tych wszystkich punktów. Algorytm szukania MST połączy punkty w kolejności, w jakiej są posortowane.

Rozwiązanie

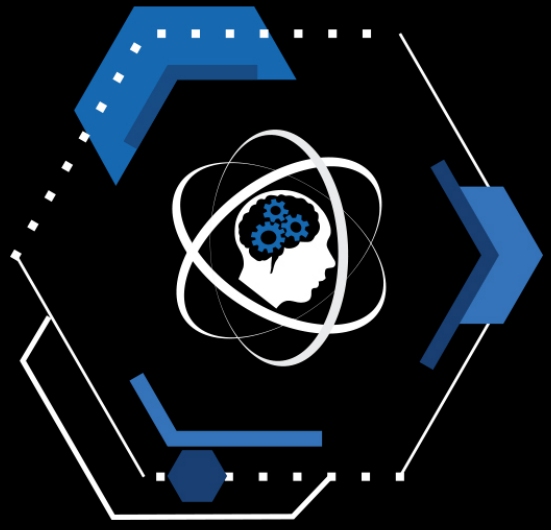
Idziemy dwoma ścieżkami od punktu 1 do punktu K.
Niech $dp[i][j]$ $(i < j)$ oznacza: najmniejsza łączna długość dwóch ścieżek, które kończą się w punktach i i j i już odwiedziły wszystko na prefiksie $[1, j]$.
Przejścia: z $dp[i][j]$ do $dp[i][j+1]$ i do $dp[j+1][j]$.

Rozwiązanie

Idziemy dwoma ścieżkami od punktu 1 do punktu K.
Niech $dp[i][j]$ $(i < j)$ oznacza: najmniejsza łączna długość dwóch ścieżek, które kończą się w punktach i i j i już odwiedziły wszystko na prefiksie $[1, j]$.
Przejścia: z $dp[i][j]$ do $dp[i][j+1]$ i do $dp[j+1][j]$.

Złożoność

$\mathcal{O}(n \log n + k^2) = \mathcal{O}(n \log n)$



Podobieństwo genetyczne

Najszybsze rozwiązanie: **Mateusz Radecki (0:51)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Dane są dwa słowa nad alfabetem czteroliterowym. Policz podciągi długości m ($m \leq 20$), które występują w obu słowach.

Definicja

Przez `next(t, i, j)` oznaczmy najbliższą literę j występująca za pozycją i w słowie s_t (lub $|s_t| + 1$ jeśli taka pozycja nie istnieje).

Definicja

Przez `next(t, i, j)` oznaczmy najbliższą literę j występująca za pozycją i w słowie s_t (lub $|s_t| + 1$ jeśli taka pozycja nie istnieje).

Obserwacja

Najkrótszy prefiks słowa s_t, który zawiera słowo w jako podciąg może zostać znaleziony w czasie $\mathcal{O}(|w|)$ za pomocą tablicy `next(t, i, j)`. Analogiczna obserwacja jest poprawna dla najkrótszego sufiksu.

Definicja

Oznaczmy sobie przez `pref(t, w)` najkrótszy prefiks słowa s_t w którym zawiera się słowo w. Analogicznie oznaczmy przez `suf(t, w)` jako najkrótszy sufiks słowa s_t w którym zawiera się słowo w.

Definicja

Oznaczmy sobie przez `pref(t, w)` najkrótszy prefiks słowa s_t w którym zawiera się słowo w. Analogicznie oznaczmy przez `suf(t, w)` jako najkrótszy sufiks słowa s_t w którym zawiera się słowo w.

Algorytm

Zastosujemy algorytm *meet in the middle* i dla każdego słowa długości $\frac{m}{2}$ znajdziemy tablice `pref(t, w)` i `suf(t, w)`. Możemy zauważyć, że słowo $w_1 w_2$ występuje w obu słowa, wtedy i tylko wtedy gdy `pref(0,` w_1`)` $<$ `suf(0,` w_2`)` i `pref(1,` w_1`)` $<$ `suf(1,` w_2`)`. Możemy łatwo wyznaczyć liczbę takich par za pomocą zamiatania.

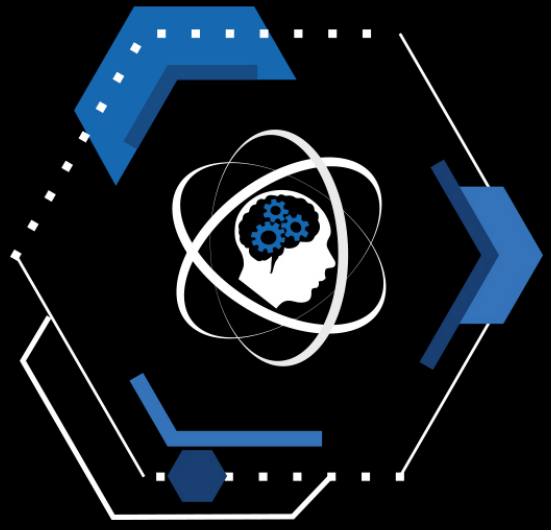
Definicja

Oznaczmy sobie przez `pref(t, w)` najkrótszy prefiks słowa s_t w którym zawiera się słowo w. Analogicznie oznaczmy przez `suf(t, w)` jako najkrótszy sufiks słowa s_t w którym zawiera się słowo w.

Algorytm

Zastosujemy algorytm *meet in the middle* i dla każdego słowa długości $\frac{m}{2}$ znajdziemy tablice `pref(t, w)` i `suf(t, w)`. Możemy zauważyć, że słowo $w_1 w_2$ występuje w obu słowa, wtedy i tylko wtedy gdy `pref(0,` w_1`)` $<$ `suf(0,` w_2`)` i `pref(1,` w_1`)` $<$ `suf(1,` w_2`)`. Możemy łatwo wyznaczyć liczbę takich par za pomocą zamiatania.

Złożoność: $\mathcal{O}(|s_1| + |s_2| + m4^{m/2})$.



Nowy Kontrakt 2

Najszybsze rozwiązanie: **Marek Sokołowski (2:41)**

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Mamy daną tablicę liczb zapisanych w systemie binarnym.
Dostajesz wiele zapytań online o zmianę jednego bitu w jednej liczbie oraz znalezienie takiego zastąpienia niektórych jedynek zerami, że kolejne liczby są ściśle rosnące a ich suma jest minimalna.

Obserwacja

Rozwiązanie zachłanne, które rozpatruje kolejne liczby i dodaje następną liczbę w taki sposób, że minimalizuje jej wartość jest poprawne.

Obserwacja

Rozwiązanie zachłanne, które rozpatruje kolejne liczby i dodaje następną liczbę w taki sposób, że minimalizuje jej wartość jest poprawne.

Lemat

Rozwiązanie zachłanne dla nowej liczby wybierze pewien prefiks jedynek z poprzedniego rozwiązania i dorzuci dokładnie jedną nową jedynkę, bądź stwierdzi, że rozwiązanie nie istnieje.

Obserwacja

Rozwiązanie zachłanne, które rozpatruje kolejne liczby i dodaje następną liczbę w taki sposób, że minimalizuje jej wartość jest poprawne.

Lemat

Rozwiązanie zachłanne dla nowej liczby wybierze pewien prefiks jedynek z poprzedniego rozwiązania i dorzuci dokładnie jedną nową jedynkę, bądź stwierdzi, że rozwiązanie nie istnieje.

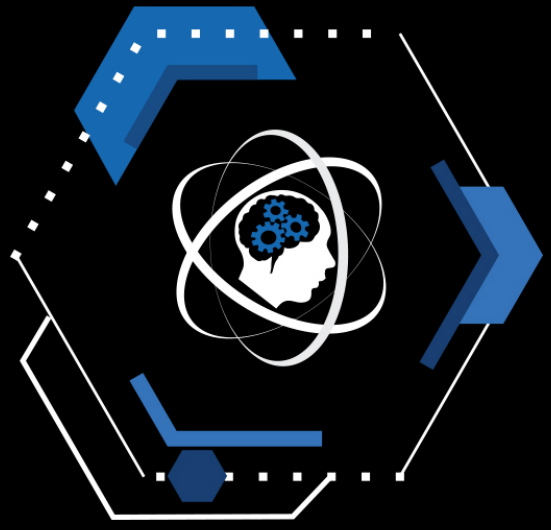
Szkic dowodu

Spośród dorzuconych jedynek skupmy się na tej co występuje na największym bicie. Jeśli na wcześniejszych bitach mieliśmy wcześniej jedynkę, ale ją wyrzuciliśmy to nasza liczba jest mniejsza niż poprzednia co jest sprzeczne z założeniem. Jeśli na późniejszych bitach mamy jakieś jedynki to po ich wyrzuceniu wciąż nasza liczba jest większa od poprzedniej, więc spełnia warunki zadania a jest mniejsza.

Algorytm

Będziemy utrzymywać strukturę, która dla ustalonej pozycji bitu będzie w stanie odpowiedzieć na zapytanie "indeks pierwszej liczby w tablicy na pozycji dalszej niż r, że ma zgaszony ten bit".
Teraz przeiterujemy się po kolejnych liczbach i będziemy utrzymywać strukturę mówiącą nam o pozycjach kolejnych jedynek w poprzedniej liczbie oraz dla każdej wybranej jedynki najbliższą liczbę w której ten bit jest zgaszony.
Możemy więc usunąć jedynki, które nie występują już w naszym wierszu (pewien sufiks) oraz pewien sufiks pozostałych, tak aby dodana jedynka była na jak najmniejszym bicie (zachowując warunek o rosnącym ciągu). Należy uważać na przypadki gdy odpowiedź jest równa -1.

Złożoność: $\mathcal{O}(Q_1 logN + Q_2 N log(Q_1 + N + M))$, gdzie Q_1 oznacza zapytania o zmianę bitu, a Q_2 oznacza zapytania o zmianę części jedynek w zera.



Sznurowadła

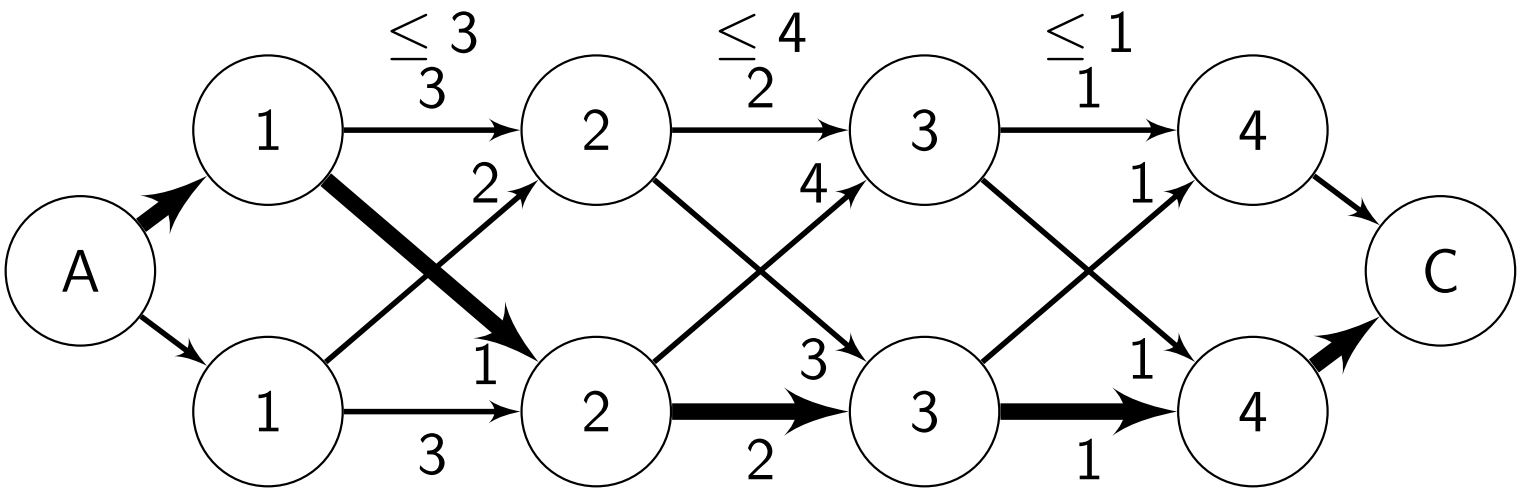
Najszybsze rozwiązanie: ??? (?:??)

PARTNER Google

ORGANIZATOR UNIWERSYTET WARSZAWSKI FUNDACJA ROZWOJU INFORMATYKI

SPONSOR ATENDE

Dany jest graf-sznurowadło. Krawędziom łączącym wierzchołki i oraz $i+1$ przypisujemy całkowite koszty z przedziału od 1 do a_i. Dla każdego z możliwych $\prod a_i^4$ układów określamy najtańszy koszt dotarcia z dowolnego 1 do dowolnego n. Podać sumę otrzymanych wartości.



Pierwsze podejście:

programowanie dynamiczne

- stan = numer aktualnej pary miast, koszt dotarcia do pierwszego miasta w parze, koszt dotarcia do drugiego miasta w parze
- dla każdego stanu liczymy liczbę układów, które do niego prowadzą
- koszt $O(N\ (NM)^2\ M^4)$, gdzie N – liczba miast, M – ograniczenie na a_i

Drugie podejście:

kompresja stanów

- jeśli mamy dwa stany (i, a, b) i (i, c, d), gdzie $|a - b| = |c - d|$, to te dwa stany zachowują się prawie tak samo
- nowy stan = numer aktualnej pary miast, różnica kosztów dotarcia do miast w tej parze ($< M$)
- dla każdego stanu liczymy liczbę układów, które do niego prowadzą, oraz sumę wartości $\min(a, b)$ dla wszystkich tych układów
- koszt $O(NM\ M^4)$

```
for d in 0..M {
  for a00 in 1..M
  for a01 in 1..M
  for a10 in 1..M
  for a11 in 1..M {
    int r0 = min(a+a00, a10);
    int r1 = min(a+a01, a11);
    int d' = abs(r0-r1);
    dp'[d'] += dp[d];
    bonus'[d'] += dp[d] * min(r0, r1) + bonus[d];
    }
  }
```

- ulepszamy do $O(N\ M^3)$ poprzez policzenie wartości t_r mówiącej na ile sposobów możemy uzyskać każde $r_0 = r$ (równoważnie $r_1 = r$) – jest to funkcja kawałkami liniowa

$$t_r = \begin{cases} M & 1 \le r \le d \\ 2M + 1 - 2r + d & d < r \le M \end{cases}$$

- dla każdej pary (`d`, `d'`) do `dp'[d']` i `bonus'[d']` dodajemy `dp[d]` i `bonus[d]` pomnożone przez pewne współczynniki
- współczynniki te możemy policzyć jako sumy ciągów postaci

$$\sum_{i=0}^{q-1} (A + ai)(B + bi)(C + ci)$$

dla odpowiednich q, A, a, B, b, C, c, co można policzyć w czasie $O(1)$

- Ostateczna złożoność: $O(NM^2)$