AISD lista 2

Dominik Szczepaniak

March 25, 2024

1 Zadanie 1

Mamy:

$$T(n) = \begin{cases} 1 & \text{dla } n = 1 \\ 2 * T(n/2) + n/logn & \text{dla } n > 1 \end{cases}$$

Używając metody drzewa rekursji, możemy zauważyć, że na każdym poziomie rekursji mamy n dwukrotnie więcej niż na poprzednim poziomie, a także pojawia się dodatkowo wyrażenie $\frac{n}{\log \frac{n}{2^k}}$, gdzie k to numer poziomu rekursji.

Możemy przyjąć, że głębokość rekursji wynosi $\log n$, ponieważ za każdym razem dzielimy n przez 2, aż do osiągnięcia 1.

Oszacujmy sumę wyrażeń $\frac{n}{\log \frac{n}{2^k}}$. Wartość k będzie od 0 do $\log n$, ponieważ dzielimy n na 2^k części, aż do osiągnięcia 1.

$\sum_{k=0}^{\log n} \frac{n}{\log \frac{n}{2^k}} \leq n \sum_{k=0}^{\log n} \frac{1}{\log \frac{n}{2^k}}$

Możemy przyjąć, że suma jest mniejsza lub równa n razy maksymalna wartość z całego wyrażenia. Maksymalna wartość tego wyrażenia wystąpi, gdy $k = 0$, ponieważ wtedy mamy najmniejszy mianownik.

$\sum_{k=0}^{\log n} \frac{1}{\log \frac{n}{2^k}} \leq \log n \cdot \frac{1}{\log \frac{n}{2^0}} = \log n \cdot \frac{1}{\log n} = 1$

Więc:

$\sum_{k=0}^{\log n} \frac{n}{\log \frac{n}{2^k}} \leq n \cdot 1 = n$

Ostatecznie, $T(n) = \Theta(n)$.

2 Zadanie 2

Jesteśmy na osi OY na wysokości nieskończoność.

Kazda z linii ma dlugosc nieskonczona.

Zadne trzy linie nie przecinaja sie w jednym punkcie.

Sortujemy po a_i

Jeśli jakaś prosta ma $a_i = 0$ to możemy przez nią przejść.

1. Posortujmy linie pod względem nachylenia - wartość bezwzględna z a_i

3 Zadanie 3

a)

Niech nasza liczba A = $(a_n, a_{n-1}, ..., a_1, a_0)_r$ - liczba z n+1 liczbami o podstawie r.

Wynik: Liczba C = A*A = A^2

Algorytm:

1. Jeśli n = 1 zwróć A * A = A^2

2. Podzielmy A na dwie równe części $A_L i A_R$:

$A = A_L * r^{n/2} + A_R$

3. Obliczmy:

$d_1 = reku(A_L)$

$d_0 = reku(A_R)$

$d_{0,1} = reku(A_L + A_R)$

4. Zwróćmy:

C = $d_1 * r^n + (d_{0,1} - d_0 - d_1) * r^{n/2} + d_0$

$A * A = (A_L * r^{n/2} + A_R)^2 = (A_L^2 * r^n + 2 * A_L * r^{n/2} * A_R + A_R^2)$

Niech $d_0 = A_L^2$

Niech $d_1 = A_R^2$

Niech $d_{0,1} = (A_L + A_R)^2$

Czyli $d_{0,1} - d_0^2 - d_1^2 = 2 * A_L * A_R * r^{n/2}$

W takim razie możemy iść rekurencyjnie i dostaniemy poprawny wynik.

Zlozonosc taka sama jak podstawowy karatsuba, bo robi dokladnie to samo przeciez:

$O(n^{log_2 3})$

b)

Liczba a = $a_0 * x^2 + a_1 * x + a_0 * x^0$

$x = 10^{\frac{n}{3}}$

Nazwijmy $a_0 = a, a_1 = b, a_2 = c$

Wtedy a * a $= a * a * x^4 + 2ab * x^3 + (2ac + b^2) * x^2 + 2bc * x + c^2$

Niech
$a^2 = C_4$
$2ab = C_3$
$2ac + b^2 = C_2$
$2bc = C_1$
$c^2 = C_0$

Wtedy nasz algorytm:
$X_0 = C_0 = c^2$
$X_1 = (C_4 + C_3 + C_2 + C_1 + C_0) = (a + b + c)^2$
$X_2 = (C_4 - C_3 + C_2 - C_1 + C_0) = (a - b + c)^2$
$X_3 = (16C_4 + 8C_3 + 4C_2 + 2C_1 + C_0) = (4a + 2b + c)^2$
$X_4 = C_4 = a^2$

X_4 oraz X_0 liczymy rekurencyjnie wywołując nasz algorytm na odpowiednio c^2 oraz a^2

$X_1 - X_2 = 2(C_4 + C_2 + C_0)$
Znamy już C_4 oraz C_0, więc:
$C_2 = \frac{X_1 - X_2}{2} - C_4 - C_0$
Dzielenie przez 2 robimy brute-forcując

Jak bruteforce:
Mamy liczbe 12345678
liczymy:
1 / 2 < 1
12 / 2 = 6
3 / 2 = 1 r 1
4+10 / 2 = 7
5 / 2 = 2 r 1
16 / 2 = 8
7 / 2 = 3 r 1
18 / 2 = 9
Liczba:
6172839

Jak na końcu zostanie reszta 1 to ją ignorujemy - wtedy liczba była nieparzysta.

$X_3 - 2X_1 = 14C_4 + 6C_3 + 2C_2 - C_0$

Znamy $C_4, C_2 i C_0$, więc z tego mamy C_3.

14 = (10+4)

6 = (10-4)

$10(C_4 + C_3) + 4(C_4 - C_3)$

10 to dodanie zera na koniec.

Mnożenie przez 14, 6 i 2 wykonujemy standardowo bruteforcując Te liczby mają krótki zapis binarny - do 5 cyfr, więc nie jest to problemem przy dużych liczbach.

$X_1 - X_2 - X_3 + 2X_4 = C_1$

Złożoność $T(n) = 5T(n/3) + O(n) = O(n^{log_3(5)})$

Dla ogólnego przypadku mamy:

$(a1 * 10^{\frac{(k-1)*n}{k}} + a2 * 10^{\frac{(k-2)*n}{k}} + ... + a_k * 1)^2 =$

$(a*10^{3n/4} + b*10^{2n/4} + c*10^{n/4} + d)^2 = a^2*10^{6n/4} + 2ab*10^{5n/4} + 2ac*10^n + 2ad*10^{3n/4} + b^2*10^n + 2bc*10^{3n/4} + 2bd*10^{2n/4} + c^2*10^{2n/4} + 2cd*10^{n/4} + d^2$
$= a^2*10^{6n/4} + 2ab*10^{5n/4} + 10^n(2ac + b^2) + 10^{3n/4}(2ad + 2bc) + 10^{2n/4}(2bd + c^2) + 2cd*10^{n/4} + d^2$

Ogólnie mamy liczbe:

$a * x^{(k-1)n/k} + b * x^{(k-2)n/k} + ... + z * x^0$

Nasze mnożenie to mnożenie dwóch macierzy:

$[a, b, c, d, ..., k]*[x^(k-1)n/k, x^(k-2)n/k, ..., x^1, x^0]*[a, b, c, d, ..., k]*[x^(k-1)n/k, x^(k-2)n/k, ..., x^1, x^0]$

Macierz kx1 * 1xk = kxk

Na samej górze będą tylko elementy z a.

Potęgi będą:

$[x^(2k-2)n/k, x^(2k-3)n/k, x^(2k-4)n/k, x^(2k-5)n/k, ..., x^(k-1)n/k]$

Poniżej z b:

$[x^(2k-3)n/k, x^(2k-4)n/k, x^(2k-5)n/k, x^(2k-6)n/k, ..., x^(k-2)n/k]$

Później:

$[x^(2k-4)n/k, ...]$

Dla każdego wiersza k-1 elementów będzie taka sama jak w poprzednim.
W takim razie ile będzie unikalnych elementów?
Pierwszy element z pierwszego wiersza
ostatni element z ostatniego wiersza
i wszystkie elementy między $[x^(2k-3)n/k, x^1] -> [x^1, x^(2k-3)n/k]$
między 2 a 3 są 3 - 2 + 1 elementów
między 2k-3 i 1 jest 2k-3 - 1 + 1 = 2k-3 elementów
dodać element pierwszy z pierwszej oraz ostatni z ostatniej - 2
czyli 2k-1 elementów.
Czyli nie jest szybszy od algorytmu mnozenia, bo jego zlozonosc to bedzie $O(n^logk(2k-1))$, czyli tak jak algorytm mnozenia

4 Zadanie 4

Jeśli dzielimy wobec prostej to robimy następujący algorytm łączenia wyników:

1. Wybieramy najbardziej wysunięty na prawo punkt z lewej otoczki (p) i najbardziej wysunięty na lewo punkt z prawej otoczki (q).

2. punkt p pozostaje nieruchomo, w nim zaczepimy nasza "wskazówkę" (prostą z p do q). Wyznaczamy punkt q', który będzie kolejnym wierzchołkiem w prawej otoczce, idąc zgodnie z ruchem wskazówek zegara. Teraz sprawdźmy, jak przesunęła się nasza wskazówka, jeśli przeciwnie do ruchu wskaówek zegara to nasze q' to nowe q i powtarzamy ten krok, wpp. przechodzimy do punktu 2).

3. punkt q pozostaje nieruchomo, w nim zaczepimy nasza "wskazówkę" (prostą z q do p). Wyznaczamy punkt p', który będzie kolejnym wierzchołkiem w lewej otoczce, idąc przeciwnie z ruchem wskazówek zegara. Teraz sprawdźmy, jak przesunęła się nasza wskazówka, jeśli zgodnie z ruchem wskaówek zegara to nasze p' to nowe p
i powtarzamy ten krok, wpp. przechodzimy do punktu 3).

4. powtarzamy punkty 1) i 2), aż do momentu, w którym
i się "ustabilizują" tzn. nie będą się już zmieniały

Zauważmy, że algorytm ten zachowuje własność STOP-u, ponieważ otoczki wypunkłe, które towżymy w każdym z kroków, są konstruowane na podstawie dwóch wielokątów wypukłych.Zatem jeżeli znajdziemy wierzchołek, dla którego wybór kolejnego wierzchołka nie jest już poprawny, to wiemy, że każdy kolejny

Analogicznie, szukamy dolnej granicy.

Porównajmy:
jak wyznaczamy kiedy ok?
górna q zgodnie przeciwnie
górna p przeciwnie zgodnie
dolna q przeciwnie zgodnie
dolna p zgodnie przeciwnie

Na podstawie Cormen strona 1040:

Aby sprawdzić, czy odcinek skierowany qp jest położony zgodnie z ruchem wskazówek zegara w stosunku do odcinka skierowanego qp' względem ich wspólnego końca q, wykonujemu przesunięcie punktu q do początku układu. To znaczy, oznaczmy jako $p - q$ to wektor $p_1' = (x_1', y_1')$, gdzie $x_1' = x_1 - x_0$, a $y_1' = y_1 - y_0$ i podobnie definiujemy $p' - q$. Następnie obliczamy iloczyn wektorowy $(p-q) \times (p'-q) = (x_1 - x_2) * (y_2 - y_0) - (x_2 - x_0) * (y_1 - y_0)$. Jeśli jego wartość jest dodatnia to odcinek skierowany qp jest położony zgodnie ze wskazówkami zegara, w stosunku do qp' (jeśli jest ujemna to przeciwnie).

Algorytm:

Przechowujemy wierzchołki znajdujące się na otoczkach L i P (lewa i prawa) na listach dwukierunkowych. Będziemy na niej przechowywać współrzędne punktu oraz wskaźnik do poprzedniego i następnego wierzchołka.

zwraca 1 jeśli zgodnie
zwraca 0 jeśli przeciwnie

```
funkcja czy_zgodnie(a, b, b')
     fi = (b.x - a.x)(b'.y - a.y) - (b'.x - a.x)(b.y - a.y);
     if fi > 0 : return 0
     wpp.: zwroc 1
```

```
funkcja krawedz_gorna():
    p = najbardziej wysuniety na prawo wierzcholek w L
    q = najbardziej wysuniety na lewo wierzcholek w P
    p' = p.poprzedni
    q' = q.nastepny
    Wykonaj:
            flaga = 0
            Dopoki czy_zgodnie(p, q, q') == 0:
                    q = q'
                    q' = q'.nastepny
                    flaga = 1
            Dopoki czy_zgodnie(q, p, p') == 1:
                    p = p'
                    p' = p'.poprzedni
                    flaga = 1
    Dopoki flaga == 1;


funkcja krawedz_dolna():
    p = najbardziej wysuniety na prawo wierzcholek w L
    q = najbardziej wysuniety na lewo wierzcholek w P
    p' = p.nastepny
    q' = q.poprzedni
    Wykonaj:
            flaga = 0
            Dopoki czy_zgodnie(p, q, q') == 1:
                    q = q'
                    q' = q'.poprzedni
                    flaga = 1
            Dopoki czy_zgodnie(q, p, p') == 0:
                    p = p'
                    p' = p'.nastepny
                    flaga = 1
    Dopoki flaga == 1;

    wpp.: zwroc 0
```

Definicja: Wielokąt wypukły to taki, w którym wszystkie kąty mają miary <= 180

Poprawność:

Wiemy że otoczka wypukła, którą wyzanczył powyższy algorytm zawiera wszystkie wierzchołki, ponieważ za każdym razem złączamy dwie otoczki wypukłe, a one zawierają wszystkie punkty swojego zbioru.

Wyznaczona otoczka jest wielokątem wypukłym, ponieważ wiemy, że algorytm znajduje najwyżej położoną prostą, której zmiana jednego z wierzchołków na kolejny wierzchołek poprzedniej otoczki liniowej spowodowałby wykluczenie z otoczki co najmniej jednego punktu, który w tej otoczce powinien sie znajdować.

Działania opieramy na figurze wypukłej, czyli figurze, której kąty wewnętrzne nie przekraczają 180 stopni. Jeżeli więc za punktem q', który nie spełnia warunku naszego algorytmu, znalazłby sie punkt położony wyżej od aktualnego q to okazałoby sie, że jest to figura wklęsła.

Wiemy że gdybysmy "przeszli się" po otoczce liniowej to zawsze będziemy skręcać w jedną stronę - tylko w lewo lub tylko w prawe.

5 Zadanie 5

gowno jebane skip

6 Zadanie 6

7 Zadanie 7

8 Zadanie 8

9 Zadanie 9