Wiktor Kuchta

3/1

---

procedure GCD(a, b)
    procedure ITER(a, b, m)
        if $a > b$ **then**
            return ITER(b, a, m)
        else if $a = 0$ **then**
            return mb
        else if even a **and** even b **then**
            return ITER($a/2$, $b/2$, $2m$)
        else if even a **and** odd b **then**
            return ITER($a/2$, b, m)
        else if odd a **and** even b **then**
            return ITER(a, $b/2$, m)
        else if odd a **and** odd b **then**
            return ITER(a, $(b - a)/2$, m)
        end if
    end procedure
    return ITER(a, b, 1)
end procedure

W każdym wywołaniu (poza zamianą argumentów), jeden z argumentów jest co najmniej dwa razy mniejszy niż wcześniej. Zatem algorytm wykonuje $O(\lg a + \lg b)$ wykonań rekurencyjnych.

Przy jednorodnym kryterium kosztów operacje w każdym wywołaniu zajmują $O(1)$ czasu, a przy logarytmicznym $O(\lg a + \lg b)$.

Zakładając optymalizację wywołań ogonowych, przy jednorodnym kryterium kosztów algorytm potrzebuje $O(1)$ pamięci, a przy logarytmicznym $O(\lg a + \lg b)$. Trzymanie akumulatora m nie zwiększa zużycia pamięci, bo początkowo zajmuje 1 bit, a zwiększamy liczbę zajmowanych bitów o 1 kiedy zmniejszamy liczbę bitów zajmowanych przez a i b o 1.

Algorytm Euklidesa ma takie samy koszty każdej iteracji i potrzebuje tyle samo pamięci, ale wykonuje tylko $O(\lg \min\{a, b\})$ iteracji.

3/3

Mamy dwie otoczki wypukłe rozdzielone pionową kreską. Musimy znaleźć górną i dolną prostą styczną do obu otoczek. Takie proste styczne przechodzą przez wierzchołki otoczek, więc będziemy rozpatrywać odcinek łączący wierzchołki obu otoczek i zmieniać końce tego przecinka, aż odcinek będzie leżał na odpowiedniej prostej.

Zajmiemy się górną styczną, dla dolnej jest symetrycznie. Zaczniemy z odcinkiem przechodzącym przez punkt najbardziej na prawo lewej otoczki i punkt najbardziej na lewo prawej otoczki. Aby sprawdzić, czy prosta, na której leży ten odcinek, przecina prawą figurę i powinniśmy jej prawy koniec przesunąć do wierzchołka wyżej, wystarczy sprawdzić czy te wierzchołki (obecny i potencjalny nowy koniec) prawej figury leżą po odpowiedniej stronie prostej. Symetrycznie dla lewej figury.

Zatem będziemy przesuwać końce odcinka do góry, aż ta prosta nie będzie przecinać żadnej z figur. Wykonamy takich porównań i przesunięć co najwyżej tyle, ile jest wierzchołków na otoczkach.

3/7

Macierz Toeplitza jest stała na każdej przekątnej. Zatem możemy ją reprezentować przez wektor $2n-1$ liczb, które są wyrazami kolejnych przekątnych.

W tej reprezentacji sumowanie macierzy to po prostu suma wektorów je reprezentujących.

Zauważmy, że macierz Toeplitza rozmiaru parzystego możemy podzielić na 4 macierze blokowe tego samego rozmiaru, gdzie macierze na przekątnej są równe. Zatem iloczyn takiej macierzy i wektora $v = \left[\begin{smallmatrix} v_1 \\ v_2 \end{smallmatrix}\right]$ możemy zapisać jako

$$Av = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} A_1 v_1 + A_2 v_2 \\ A_3 v_1 + A_1 v_2 \end{bmatrix} = \begin{bmatrix} A_1(v_1+v_2) - (A_1 - A_2) v_2 \\ A_1(v_1+v_2) - (A_1 - A_3) v_2 \end{bmatrix}.$$

Wyznaczmy złożoność $T(n)$ rekurencyjnego algorytmu, który korzysta z powyższej własności. Obliczanie v_1+v_2, A_1-A_2, A_1-A_3 zajmuje $O(n)$ operacji. Każde z mnożeń macierzy przez wektor zajmuje $T(n/2)$ operacji. Następnie dodanie wektorów wynikowych zajmuje $O(n)$ operacji. Zatem otrzymujemy

$$T(n) = 3T(n/2) + O(n) = O(n^{\lg_2 3}) \approx O(n^{1.58}).$$

Jeśli n jest nieparzyste, to rozszerzamy powiększamy macierz i wektor o 1, tzn. rozszerzamy każdą przekątną macierzy o 1 i ustawiamy nowe przekątne na 0 i dodajemy 0 na koniec wektora. Wtedy wynik ich iloczynu po zignorowaniu ostatniego wyrazu wektora to wynik mnożenia, który chcieliśmy.

3/8

Niech liczba tablic $k \leqslant 3$, m_i to indeksy median T_i, gdzie T_i ma rozmiar n_i. Weźmy i takie, że $\lceil \frac{n_i}{2} \rceil$ najmniejsze. Mamy $\sum_{i=1}^{k} \lceil \frac{n_i}{2} \rceil$