AiSD

Lista 3.

Zadanie 7

Maksymilian Perduta 308080

21 kwietnia 2020

1 Treść.

Macierz A rozmiaru n x n nazywamy macierzą **Toeplitza**, jeśli jej elementy spełniają równanie $A[i, j] = A[i-1, j-1]$ dla $2 \leq i, j \leq n$.

(a) Podaj reprezentację macierzy Toeplitza, pozwalającą dodawać dwie takie macierze w czasie $O(n)$.

(b) Podaj algorytm, oparty na metodzie "dziel i zwyciężaj", mnożenia macierzy Toeplitza przez wektor. Ile operacji arytmetycznych wymaga takie mnożenie?

2 Reprezentacja macierzy Toeplitza.

Zauważmy, że macierz Toeplitza będzie mieć co najwyżej $2n-1$ różnych wartości, ponieważ każdy element w pierwszym wierszu i pierwszej kolumnie jest dowolny, a reszta jest już przez nie ustalona. Zauważmy też, że ostatnia kolumna składa się z dokładnie tych samych elementów co pierwszy rząd, a ostatni rząd składa się z dokładnie tych samych elementów co pierwsza kolumna. Wiedząc to możemy zapisać macierz jako listę o długości $2n-1$. W celu łatwiejszego dodawania tak zapisanych macierzy, kolejne elementy tej listy będziemy uzupełniać w następujący sposób: zaczynamy od ostatniego, dolnego wiersza i zapisujemy po kolei wszystkie wartości występujące w tym wierszu. Po dotarciu do końca wiersza, przechodzimy po kolumnie w górę, również zapisując wszystkie wartości do listy(pomijając oczywiście

pierwszy element, który został już zapisany). Innymi słowy dla danej macierzy Toeplitza A, rozmiaru n x n jej reprezentacja w postaci listy będzie wyglądać następująco: $a_{n1}, a_{n2}, ..., a_{nn}, a_{n-1n}, ..., a_{1n}$. Zauważmy, że dzięki takiemu ustawieniu danych, każde kolejne n elementów w tej liście odpowiada kolejnym wierszom macierzy wejściowej, patrząc od dołu. W takim razie, by otrzymać listę wynikową wystarczy dodać do siebie elementy tych list znajdujące się na tych samych pozycjach (ten sposób odpowiada normalnemu dodawaniu macierzy). Otrzymamy wtedy listę wynikową, z której w wyżej wspomniany sposób możemy łatwo odtworzyć postać macierzową. Samo dodawanie dwóch list działa w czasie $O(n)$, (dokładnie $2n-1$).

3 Mnożenie macierzy Toeplitza przez wektor.

W przypadku mnożenia macierzy Toeplitza przez wektor, posłużymy sie metodą "dziel i zwyciężaj". Będziemy chcieli podzielić sobie daną macierz M rozmiaru n x n na 4 mniejsze macierze A, B, C, oraz D w następujący sposób: niech $z = \lfloor \frac{n}{2} \rfloor + 1$ oznacza numer kolumny i wiersza po których prowadzimy linię podziału. W ten sposób mamy macierz A rozmiaru z x z, B rozmiaru z x $(z-1)$, C rozmiaru $(z-1)$ x z oraz macierz D rozmiaru $(z-1)$ x $(z-1)$. Dany wektor V, przez który mnożymy naszą macierz również dzielimy na dwie części E i F, gdzie E jest górną jego częścią o rozmiarze z x 1. Z definicji mnożenia macierzy mamy:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E \\ F \end{bmatrix} = \begin{bmatrix} AE + BF \\ CE + DF \end{bmatrix} \tag{1}$$

W tym przypadku mamy 4 mnożenia macierzy, które możemy policzyć w czasie $O(n^2)$ i 2 dodawania, które liczymy w czasie $O(n)$. Rekurencyjnie możemy taki algorytm zapisać następująco:

$$T(M, V) = polacz(T(A, E) + T(B, F), T(C, E) + T(D, F)); \tag{2}$$

Funckcja **polacz** scala dwie części wektora w jeden. W zależności od sposobu deklarowania wektora ostatni element podwektora E może na przykład wskazywać na pierwszy element podwektora F, co w rezultacie da nam cały wynikowy wektor. Nazwijmy ten sposób mnożenia "**zwykłym**" Możemy spróbować zoptymalizować ten algorytm. Zauważmy, że jeśli n jest parzyste to macierz D równa jest macierzy A tzn. zachodzi:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A & B \\ C & A \end{bmatrix} \tag{3}$$

Przekształćmy teraz naszą macierz wynikową.

$$\begin{bmatrix} AE+BF \\ CE+DF \end{bmatrix} = \begin{bmatrix} AE+BF \\ CE+AF \end{bmatrix} = \begin{bmatrix} A(E+F)+(B-A)F \\ A(E+F)+(C-A)E \end{bmatrix} \tag{4}$$

Dla takich danych mamy 3 mnożenia zamiast początkowych 4, co daje nam lepszą złożoność(mamy dodatkowe operacje dodawania i odejmowania, ale działają one w czasie $O(n)$, podczas gdy mnożenie w $O(n^2)$). Rekurencyjnie możemy taki algorytm zapisać następująco:

$$T(M,V) = polacz(T(A,(E+F))+T((B-A),F), T(A,(E+F))+T((C-A),E)); \tag{5}$$

Nazwijmy ten sposób mnożenia"**sprytnym**"(pamiętając oczywiście o zapisaniu gdzieś wartości $T(A,(E+F))$, żeby nie liczyć jej dwa razy). Pseudokod dla tak zadeklarowanych danych wygląda następująco:
Dane:
M - macierz wejściowa,
V - wektor wejściowy,
A,B,C,D - macierze pomocnicze określone na dwa wyżej wymienione sposoby, E,F - wektory pomocnicze określone w wyżej wymieniony sposób, n - rozmiar M,
T - wynik mnożenia M i V,
wart(A) - funkcja, która dla macierzy A rozmiaru 1x1 wyciąga jej wartość,

$T(M,V)$:
if $n == 1$ **then**
  return $wart(M) \cdot wart(V)$;
else if $(n \bmod 2) == 0$ **then**
  skonstruuj macierze A,B,C i wektory E i F;
  return $sprytny(M,V)$;
else if $(n \bmod 2) == 1$ **then**
  skonstruuj macierze A,B,C,D i wektory E i F;
  return $zwyky(M,V)$;
end if
return

Nasz sposób dzielenia macierzy na podmacierze polega na dzieleniu ich na możliwie najbardziej równe sobie części, co odpowiada dzieleniu przez dwa ich rozmiarów. Gwarantuje nam to, że zawsze wejdziemy do przypadku gdy $n = 1$, więc ten algorytm zawsze zwróci wynik i nie zapętli się. Ustaliliśmy wcześniej w jaki sposób mnożymy macierze, oraz jak je łączymy, więc ostateczny kod polega jedynie na rozróżnieniu sytuacji, w których ich używamy.

Dokładny algorytm czyli np. funkcje "połącz" a także implementacja sposobów mnożenia, zależy od sposobu deklarowania macierzy i wektora. Ogólny sposób został zaprezentowany wyżej. Złożoność tego algorytmu musimy wyznaczyć rozpatrując normalny sposób obliczania macierzy. W każdym kroku uruchamiamy 4 rekurencje i wykonujemy 4 mnożenia oraz 2 dodawania liczb, które zachodzą w czasie stałym. Dzielimy w każdym kroku rozmiar macierzy przez 2, więc całkowita liczba kroków będzie równa $\log_2 n$. Mamy wtedy $4^{\log_2 n} = 2^{2\log_2 n} = n^2$ operacji. Stąd, złożoność tego algorytmu to $O(n^2)$. Zauważmy na koniec, że w przypadku sprytnym mielibyśmy $3^{\log_2 n}$ operacji, a więc w praktyce ostateczna złożoność będzie nieco mniejsza (więcej niż $O(n)$, ale mniej niż $O(n^2)$