

Zad 7

Kwadratowa macierz Toeplitza:

$$A = \begin{bmatrix} a & b & c & d & \dots \\ e & a & b & c & \dots \\ f & e & a & b & \dots \\ g & f & e & a & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Przekątne mają tę samą wartość na każdym polu.

- a) Możemy łatwo zauważyc, że wystarczy zapamiętać $2n-1$ wartości w pierwszym wierszu: w pierwszej kolumnie. Dzięki temu możemy dodawać do siebie dając takie macierze o wymiarach $n \times n$ (dodawać do siebie dwa wektory $2n-1$ -elementowe) w czasie $O(n)$.

- b) Rozszerzymy macierz A do rozmiaru $k \times k$, gdzie $k = 2^{\lceil \log n \rceil}$.

Nowopowstałe pola uzupełniamy zgodnie z przekątnymi macierzy A. Na nowych przekątnych wstawiamy zero. Wektor V przez który chcieliśmy mnożyć również rozszerzamy do rozmiaru k dopisując zero. W ten sposób otrzymalismy macierzą A' i wektor V':

$$A' = \begin{bmatrix} a & b & c & \dots & 0 \\ d & a & b & \dots & \vdots \\ e & d & a & \dots & c \\ \vdots & \vdots & \vdots & \ddots & b \\ 0 & \dots & e & d & a \end{bmatrix} \quad V' = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Zauważmy, że pomnożenie macierzy A' przez wektor V' nie zmieni nam pierwszych n wartości w wektorze wynikowym dzięki zerom w wektorze V'. Natomiast wiersze $n+1 \dots k$ w macierzy wynikowej wystarczy po prostu pominać i w ten sposób otrzymamy macierz B o wymiarach $n \times 1$, gdzie $B = A \cdot V$.

Dlaczego w takim razie na dodanych polach w macierzy A' wpisywalismy też wartości zgodnie z przekątnymi A?

Na potrzeby algorytmu:

Chcemy pomnożyć $A^1 \cdot V^1$. Możemy zauważyc, że:

$$A^1 = \begin{bmatrix} P & Q \\ R & P \end{bmatrix} \quad \text{oraz} \quad V^1 = \begin{bmatrix} X \\ Y \end{bmatrix}$$

Gdzie P, Q, R to kwadratowe macierze wymiaru $\frac{k}{2} \times \frac{k}{2}$, a X, Y to wektory rozmiaru $\frac{k}{2}$.

$$A^1 \cdot V^1 = \begin{bmatrix} P \cdot X + Q \cdot Y \\ R \cdot X + P \cdot Y \end{bmatrix}$$

Ważymy D, E, F jako:

$$D = (P+Q) \cdot Y$$

$$E = (R+P) \cdot X$$

$$F = P \cdot (Y-X)$$

Wtedy:

$$A^1 \cdot V^1 = \begin{bmatrix} P \cdot X + Q \cdot Y \\ R \cdot X + P \cdot Y \end{bmatrix} = \begin{bmatrix} PY - PY + PX + QY \\ PX - PX + RX + PY \end{bmatrix} = \begin{bmatrix} D - F \\ E + F \end{bmatrix}$$

Musimy więc wyznaczyć wartości D, E, F , a następnie obliczyć $D-F$; $E+F$

$$(*) \quad T(n) = \begin{cases} 1 & \text{gdy } n=1 \\ 3T\left(\frac{n}{2}\right) + O(n) & \text{gdy } n>1 \end{cases} \Rightarrow T(n) = O(n^{\log_2 3})$$

(*) $O(n)$ oznacza, że dodawanie macierzy możemy wykonać w czasie liniowym, ponieważ koniunktamy 2 notacji: 2 podp. a dla każdej macierzy D, E, F .

Dokładniej, wykonujemy dwie sumy macierzy Toeplitza rozmiaru $\frac{n}{2} \times \frac{n}{2}$, co wymaga $2 \cdot 2 \frac{n}{2} - 1 = 2n-2$ operacji, jedną sumę wektorów $Y-X \rightarrow 1 \cdot \frac{n}{2} = \frac{n}{2}$, na koniec dwie sumy wektorów $D-F; E+F: 2 \cdot \frac{n}{2} = n$. Stąd $2n-2 + \frac{n}{2} + n = \frac{7n-4}{2}$ operacji.

$$T(n) = \begin{cases} 1 & n=1 \\ 3T\left(\frac{n}{2}\right) + \frac{7n-4}{2} & n>1 \end{cases} \quad T(n) = O(n^{\log_2 3})$$

Zad 8

a) Mamy trzy posortowane, n_k -elementowe tablice T_1, T_2, T_3 .

Mozemy wyznaczyć ich mediany m_1, m_2, m_3 w czasie $O(1)$.

Bez straty ogólnosci założymy, że $m_1 \leq m_2 \leq m_3$. (Mozemy wyznaczyć tę równosć w czasie $O(1)$ i w razie konieczności zmienić oznaczenia T_k, m_k).

Zauważmy teraz, że m_1 : elementy T_1 mniejsze od m_1 , mozymy usunąć, gdyż nie mogą być medianą. (Lub są równe medianie, ale to nie zmieni wyniku). Analogicznie, mozymy usunąć m_3 : elementy T_3 większe od m_3 .

Dopóki T_1, T_2, T_3 są niepuste:

$$m_1, m_2, m_3 \leftarrow \text{mediany } T_1, T_2, T_3$$

$$n_1, n_2, n_3 \leftarrow \text{długości tablic } T_1, T_2, T_3$$

if ($m_1 \leq m_2 \leq m_3$):

if ($n_1 < n_2$):

usun pierwsze $\lceil \frac{n_1}{2} \rceil$ elementów z T_1

usun ostatnie $\lceil \frac{n_1}{2} \rceil$ elementów z T_3

else:

usun pierwsze $\lceil \frac{n_2}{2} \rceil$ elementów z T_1

usun ostatnie $\lceil \frac{n_2}{2} \rceil$ elementów z T_3

// Pozostałe przypadki pierwszego if-a symetrycznie

$$S_1, S_2 \leftarrow \text{Też } T_1, T_2, T_3, \text{ które są niepuste}$$

Dopóki S_1, S_2 są niepuste:

$$m_1, m_2 \leftarrow \text{mediany } S_1, S_2$$

$$n_1, n_2 \leftarrow \text{długości } S_1, S_2$$

if ($m_1 \leq m_2$):

if ($n_1 < n_2$):

usun pierwsze $\lceil \frac{n_1}{2} \rceil$ elementów z S_1

usun ostatnie $\lceil \frac{n_1}{2} \rceil$ elementów z S_2

else

usun pierwsze $\lceil \frac{n_2}{2} \rceil$ elementów z S_1

usun ostatnie $\lceil \frac{n_2}{2} \rceil$ elementów z S_2

// Drugi przypadek symetryzacji

Wynik - mediana z niepustego z S_1 lub S_2

Do drugiej części algorytmu:

Mamy dwie posortowane, n_1 -elementowe tablice S_1, S_2 .

Mieliśmy wyznaczyć ich mediany m_1, m_2 w czasie $O(1)$.

Bez straty ogólności założymy, że $m_1 \leq m_2$;

Zauważmy, że dla $n = \min(n_1, n_2)$, n najmniejsze elementy w S_1 i n największe elementy w S_2 nie będą medianą lub będą jej równe. Mieliśmy więc je usunąć, a takta operacja nie zmieni nam mediany w wyniku.

Zad 1.

Stworzymy tablicę T_{ij} , $i \in 0, 1, \dots, n+1$, $j \in 1, \dots, m$. t_{ij} oznacza koszt dojścia od dwoistego pola pierwszej kolumny do pola a_{ij} i jest to koszt minimalny.

Zauważmy, że do dwoistego pola a_{ij} możemy dojść z jednego z pięciu pól: $a_{i-1,j-1}$, $a_{i,j-1}$, $a_{i+1,j-1}$, $a_{i+1,j}$, $a_{i-1,j}$.

Wykonujemy programowanie dynamiczne, aby obliczyć koszty dróg.

Algorytm:

for j from 1 to m

Ilość czasowa: $O(n \cdot m)$

$$t_{0,j} \leftarrow \infty$$

$$t_{n+1,j} \leftarrow \infty$$

for i from 1 to n

$$t_{i,1} \leftarrow a_{i,1}$$

for i from 2 to m

for i from 1 to n

$$t_{ij} \leftarrow a_{ij} + \min\{t_{i-1,j-1}, t_{i,j-1}, t_{i+1,j-1}\}$$

for i from 1 to n

$$t_{ij} \leftarrow \min\{t_{ij}, t_{i-1,j} + a_{ij}\}$$

for i from 1 to n

$$t_{ij} \leftarrow \min\{t_{ij}, t_{i+1,j} + a_{ij}\}$$

return $\min d_{1,m}, \dots, d_{n,m}$

Znaleźć drogę, moimy cyklicznie pole 2 ostatniej kolumny o najniższym koszcie i cofać się w każdym kroku sprawdzając które 2 pięciu pól ma najmniejszą wartość.

Algorytm: procedure Path(i, j)

if $j = 1$

return pair(i, j)

if $t_{i-1, j-1} < t_{i, j-1}$

$k \leftarrow i - 1$

else

$k \leftarrow i$

if $t_{i+1, j-1} < t_{k, j-1}$

$k \leftarrow i + 1$

if $t_{i+1, j} < t_{k, j-1}$

$k \leftarrow i + 1$

$l \leftarrow j$

else

$l \leftarrow j - 1$

if $t_{i-1, j} < t_{k, l}$

$k \leftarrow i - 1$

$l \leftarrow j$

return concat(Path(k, l), pair(i, j))

Zad 2

procedure $\text{lps}(\text{str})$:

$n \leftarrow \text{str.length}$, $\text{max} = 0$

$T \leftarrow \text{new } T[n][n]$

for i from 1 to n

$T[i][i] = 1$, $\text{max} = 1$

for len from 2 to n \leftarrow długość rozpatrywanego podcięcia

for i from 1 to $n - \text{len}$

$j = i + \text{len} - 1$

if $\text{str}[i] == \text{str}[j]$ AND $\text{len} == 2$:

$T[:][j] = 2$, $\text{max} = 2$

else if $\text{str}[i] == \text{str}[j]$ AND $T[i+1][j-1] != 0$

$T[i][j] = \text{len}$

$\text{max} = \text{len}$

return max

Idea: Wykorzystamy programowanie dynamiczne do znalezienia najbliższego palindromu. W tablicy $T[i][j]$ będziemy przechowywać informację, czy podciąg powstały z liter na indeksach od i do j jest palindromem. Na początku sprawdzimy dwuznakowe podcięgi, a następnie uzupełnimy jedynkami, ponieważ kiedy wyróż jednoznakowy jest palindromem. Do rozwiązywania problemu wystarczy teraz zauważyć, że aby sprawdzić czy stóra jest palindromem należy sprawdzić czy dwa jego skrajne wyróż są identyczne i czy środek jest palindromem: informacja ta jest zawarta w polu $T[i+1][j-1]$.



