Sada: 1 Příklad: 2 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Procedura findEso($A, fromX, toX, fromY, toY$)

vstup: čtvercová matice A vymezená v intervalu $x \in \langle fromX, toX \rangle, y \in \langle fromY, toY \rangle$

výstup: souřadnice esa v matici, tedy prvek, který je větší, než jeho sousedé

```
if toX = fromX then
    return fromX, fromY // nalezeno eso, vracím souřadnice
fi
halfX ← (toX + fromX)/2
halfY ← (toY + fromY)/2
// maxX a maxY jsou souřadnice maxima z prvků kolem horizontálního a vertikálního
   středu
maxX ← halfX // pro hledání maxima začnu nastavením souřadnic na 1. prvek
maxY ← fromY
for x ← fromX to toX do
    if A[maxX][maxY] < A[x][halfY] then
        maxX ← x // zde procházím sloupec vlevo od středu
        maxY ← halfY
    fi
    if A[maxX][maxY] < A[x][halfY + 1] then
        maxX ← x // zde procházím sloupec vpravo od středu
        maxY ← halfY + 1
    fi
od
for y ← fromY to toY do
    if A[maxX][maxY] < A[halfX][y] then
        maxX ← halfX // zde procházím řádek vlevo od středu
        maxY ← y
    fi
    if A[maxX][maxY] < A[halfX + 1][y] then
        maxX ← halfX + 1 // zde procházím řádek vpravo od středu
        maxY ← y
    fi
od
if maxX <= halfX then
    if maxY <= halfY then
        return findEso (A, fromX, halfX, fromY, halfY)
    else
        return findEso (A, fromX, halfX, halfY + 1, toY)
else
    if maxY <= halfY then
        return findEso (A, halfX + 1, toX, fromY, halfY)
    else
        return findEso (A, halfX + 1, toX, halfY + 1, toY)
```

Sada: 1 Příklad: 2 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Pojmy: abych příliš často nemusel formálně popisovat množinu prvků matice, která se skládá z prvků ve sloupcích odpovídajících $halfX$ a $halfX + 1$ v rozsahu $y \in < fromY, toY >$ a řádkům odpovídajících $halfY$ a $halfY+1$ v rozsahu $x \in < fromX, toX >$, tak pro tuto množinu prvků zavádím pojem *středový kříž*.

Popis algoritmu: tento rekurzivní algoritmus je založen na tom, že se po každém průchodu změnší část matice, se kterou pracuje na čtvrtinu. Volbou maxima ze středového kříže se rozhoduje, kterou ze 4 možných čtvrtin má vybrat. Nalezení maxima zaručuje, že v dané čtvrtině maximum musí být (formálně rozebráno níže), rekurzivně se tedy algoritmus zanoří do dané čtvrtiny. Eso tedy nemusí být maximum z prvního výběru. Algoritmus by šel upravit, aby místo dělení na čtvrtiny dělil matici na poloviny, střídavě horizontálně a vertikálně, díky čemuž by měl trochu menší složitost (asymptotická složitost by však stále byla $\mathcal{O}(n)$), cenou by však byl mnohem složitější zápis. Při volání rekurze je nutno kontrolovat, aby se volala se čtvercovou maticí, což v pseudokódu není zapsáno (z důvodu složitého zápisu). Pokud by měla zvolená čtvrtina jinou šířku než délku, pak je nutno rekurzi předat větší matici (tak, aby jeden řádek navíc zaručil čtvercovost).

Korektnost: algoritmus je totálně korektní, pokud skončí a výstup splňuje výstupní podmínku, kterou je v našem případě, že nalezený prvek je eso. Konečnost je zřejmá, jelikož rozměry matice n po každém kroce klesnou minimálně na $\lceil \frac{n}{2} \rceil$, přičemž pro $n = 1$ (k čemuž algoritmus konverguje) algoritmus skončí.
Abychom mohli dokázat, že je algoritmus parciálně korektní, je nutno dokázat invariant: v každém rekurzivním volání existuje v daném intervalu matice alespoň 1 eso.
Již ze zadání vyplívá, že při prvním volání s celou vstupní maticí musí invariant platit. To je zaručeno unikátností všech prvků.
Předpokládejme platnost invariantu pro m-té rekurzivní volání. V $m+1$-ním rekurzivním volání z předchozího tvrzení víme, že v aktuální submatici se nachází eso. Pak platí, že od maxima ze středového kříže vede rostoucí posloupnost k esu a jediným prvkem této posloupnosti, který se nachází na středovém kříži, je právě nalezené maximum. Pokud by toto tvrzení neplatilo a posloupnost by měla další prvky ležící na středovém kříži, pak by díky tranzitivitě relace *větší než* musel být daný jiný prvek posloupnosti ležící na kříži větší než nalezené maximum a tím pádem by měl být sám maximem.

Asymptotická časová složitost algoritmu je $\Theta(n)$. To lze odvodit z rekurentní rovnice, která popisuje složitost každého volání rekurze a vypadá takto:

$$T(n) = T(\lceil \frac{n}{2} \rceil) + \mathcal{O}(n)$$
$$T(1) = 1$$

Tato rovnice říká, že při každém rekurzivním volání se šířka matice zmenšuje na polovinu (zaokrouhleno nahoru), a před tím se provede $4n$ porovnání při hledání maxima ve středovém kříži. Dalších operací je konstantní počet, takže platí, že $4n + k \in \mathcal{O}(n)$. Horní hranici získáme pomocí master theoremu, přičemž naše rekurentní rovnice nabývá hodnot

Sada: 1 Příklad: 2 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

$a = 1$, $b = 2$ a $d = 1$, tedy $a < b^d \Rightarrow T(n) \in \mathcal{O}(n)$. Složitost máme omezenou také zdola, hned v prvním volání provádíme $4n$ porovnání, tím pádem náš algoritmus patří do $\Theta(n)$.