

1. INTRODUÇÃO

1.1. **Sobre o Trabalho.**

1.2. **Notação.**

1.3. **Matrizes.** Explicar o que são matrizes online e offline.

1.4. **Implementações.** Explicar os padrões que estou usando pra implementar os programas. Por exemplo: Funções como argumentos, 0-index (em contraste com o 1-index do pseudo-código) e wrappers.

2. MATRIZES MONGE

Nesta seção vão ser apresentados e explorados os conceitos de matrizes Monge e monotonicidade, essenciais para o desenvolvimento do restante do texto.

3. DIVISÃO E CONQUISTA

Aqui será apresentada uma técnica que pode ser usada para encontrar máximos ou mínimos de linhas em matrizes monótonas convexas ou côncavas em tempo $O((m+n)\lg(n))$, onde m é a quantidade de colunas e n a quantidade de linhas da matriz. Ao final, apresentamos exemplos de aplicações desta ideia em programação dinâmica.

3.1. Técnica. Dada uma matriz $A \in \mathbb{Q}^{n \times m}$ monótona convexa, queremos encontrar, para toda linha i da matriz, o menor índice j tal que $A[i][j]$ é o valor máximo da linha i . Formalmente, queremos encontrar o vetor $R \in \mathbb{N}^n$ onde, para todo $i \in [n]$,

$$R[i] = \min\{j \mid A[i][j] \geq A[i][j'] \text{ para todo } j' \in [n]\}.$$

Se, para alguma linha i , encontrarmos o valor $R[i]$, sabemos, já que A é monótona convexa, que para todo $i' < i$, $R[i'] \leq R[i]$ e, para todo $i' > i$, $R[i'] \geq R[i]$, isto é, sabemos que os máximos de menor índice das outras linhas se encontram nas submatrizes $A[1..i-1][1..R[i]]$ e $A[i+1..n][R[i]..m]$. Basta, agora, resolver o mesmo problema para estas submatrizes e conseguiremos resolver o problema original.

Algoritmo 3.1 Máximo de linhas com divisão e conquista

```

1: função DIVCONQ( $A$ )
2:    $n \leftarrow$  quantidade de linhas de  $A$ 
3:    $m \leftarrow$  quantidade de colunas de  $A$ 
4:    $i \leftarrow \lceil n/2 \rceil$ 
5:    $R[i] \leftarrow \min\{j \mid A[i][j] \geq A[i][j'] \text{ para todo } j' \in [n]\}$ 
6:   se  $i > 1$  então
7:      $R[1..i-1] \leftarrow \text{DIVCONQ}(A[1..i-1][1..R[i]])$ 
8:   se  $i < n$  então
9:      $R[i+1..n] \leftarrow R[i] + \text{DIVCONQ}(A[i+1..n][R[i]..m])$ 
10:  devolve  $R$ 

```

3.2. Análise. Será feita uma análise do tempo de execução do algoritmo acima assumindo que as atribuições feitas nas linhas 7 e 9 custam tempo constante, futuramente, em 3.3, iremos apresentar uma implementação em C++ que está de acordo com a análise realizada.

Se $A \in \mathbb{Q}^{n \times m}$, o tempo gasto por $\text{FINDROWMAX_DIVCONQ}(A)$ pode ser expresso pela seguinte recorrência:

$$T(n, m) = \begin{cases} m & , \text{ se } n = 1, \\ m + T(1, r) \text{ para algum } r \in [m] & , \text{ se } n = 2, \\ \begin{aligned} & m + T(\lceil n/2 \rceil - 1, m - r + 1) \\ & + T(\lfloor n/2 \rfloor, r) \text{ para algum } r \in [m] \end{aligned} & , \text{ caso contrário.} \end{cases}$$

Proposição 2. *Para todo $n, m \geq 1$, $T(n, m) \leq (m + n) \lg(2n)$ e, portanto, a técnica da divisão e conquista consegue encontrar o máximo de todas as linhas em tempo $O((m + n) \lg(n))$*

Demonstração. Vamos usar indução em n para provar a tese. Se $n = 1$ e $m \geq 1$, $T(1, m) = m \leq (m + 1) \lg(2)$. Se $n = 2$ e $m \geq 1$, existe $r \in [m]$ tal que $T(2, m) = m + r \leq 2m \leq (m + 2) \lg(4)$. Agora, se $n > 2$ e $m \geq 1$, existe um $r \in [m]$ tal que

$$T(n, m) = m + T(\lceil n/2 \rceil - 1, r) + T(\lfloor n/2 \rfloor, m - r + 1)$$

assumimos que com $1 \leq n' < n$ e $m' \geq 1$ vale a tese para $T(n', m')$. Com isso, já que $1 \leq \lceil n/2 \rceil - 1 \leq n$, $1 \leq \lfloor n/2 \rfloor \leq n$, $r \geq 1$ e $m - r + 1 \geq 1$, temos, com a equação acima e o fato de que $\lceil n/2 \rceil - 1 \leq \lfloor n/2 \rfloor \leq n/2$,

$$\begin{aligned} T(n, m) &\leq m + (r + \lceil n/2 \rceil - 1 + m - r + 1 + \lfloor n/2 \rfloor) \lg(n) \\ &\leq m + (m + n) \lg(n) \leq (m + n)(\lg(n) + 1) = (m + n) \lg(2n). \end{aligned}$$

□

3.3. Implementação. Aqui serão discutidos os detalhes de implementação do algoritmo 3.1.

Primeiramente a matriz A deve ser facilmente manipulada, não podemos gerar novas matrizes para alimentar as chamadas recursivas da função `DIVCONQ`. Além disso, como explicado em 1.4, o parâmetro a ser passado será uma função e não uma matriz. Note que a matriz nas chamadas recursivas é sempre formada por um intervalo das linhas e um intervalo das colunas da matriz A original, portanto, podemos guardar 4 inteiros, ls , lt , cs e ct , representando, respectivamente, a primeira linha, a última linha, a primeira coluna e a última coluna acessíveis na chamada atual da recursão.

Agora, devemos tomar cuidado com as atribuições feitas nas linhas 7 e 9. A forma como elas foram apresentadas sugere que os vetores R recebidos pelas funções sejam recebidos e copiados para o vetor R . Ao invés de fazer isso, iremos passar o endereço do vetor R recursivamente e garantir que cada chamada só complete o subvetor $R[ls..lt]$, referente a seu subproblema. Além disso, a linha 9 possui uma adição, o que sugere que o valor $R[i]$ seja adicionado elemento-a-elemento, o que não será feito, os vetor R já será preenchido com o índice correto da coluna na matriz original A da primeira vez que for acessado.

```
void divconq (std::function< int(int, int) > A, int ls, int lt, int cs, int
    int i = (lt - ls)/2;

    r[i] = cs;
    for (int j = cs + 1; j <= ct; j++) {
        if (A(i, j) > A(i, r[i]))
            r[i] = j;
    }
```

```
    if (i > ls)
        divconq(A, ls, i-1, cs, r[i], r);
    if (i < lt)
        divconq(A, i+1, lt, r[i], ct, r);
}
```

REFERÊNCIAS