# Sparse Table

Laboratório de Programação Competitiva - 2020

Pedro Henrique Paiola

# Introdução

- A Sparse Table é uma estrutura de dados que permite responder a consultas em intervalos (*range queries*).
- A maioria das *queries* são realizadas em complexidade O(log(n))
- Porém, para certos problemas, como a RMQ (*Range Mininum Query*), a resposta por ser calculada em tempo O(1).
- Desvantagem: essa estrutura só pode ser usada em arrays imutáveis. Qualquer alteração implica na reconstrução da Sparse Table: O(n.log(n))

# Intuição

- Sabemos que qualquer número não negativo pode ser representado por uma soma de potências decrescentes de 2. Esta é a base para a representação binária de um número
  - Ex: 13 = $(1101)_2$ = 8 + 4 + 1
- Para um número x, pode haver no máximo $\lceil log_2 x \rceil$ números somando.
- Pelo mesmo raciocínio, qualquer intervalo pode ser representado exclusivamente como a união de intervalos com comprimentos que são potências de dois.

# Intuição

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

- Intervalo [2,8]= [2,5] U [6,7] U [8,8]
- | [2,5] | = 4
- | [6,7] | = 2
- | [8,8] | = 1

# Intuição

- A ideia principal por trás das Sparse Tables é pré-calcular todas as respostas para os intervalos com tamanho de potências de 2.
- Posteriormente, uma consulta de intervalo diferente pode ser respondida dividindo o intervalo em subintervalos de tamanho de potência de 2.
- Como já sabemos que para um número n pode haver no máximo log(n) números somando, então essa divisão em intervalo pode ter complexidade O(log n).

# Pré-computação

- Vamos armazenar as consultas pré-calculadas em uma matriz st.
- A posição st[i][j] armazenará a resposta para o intervalo [i, i + $2^j$ - 1], de tamanho $2^j$.
- O tamanho da matriz será MAXN x (K + 1), onde K >= $\log_2$(MAXN).
- Para construir a matriz st considerando um vetor v e uma função f (soma, mínimo, ...), usaremos programação dinâmica baseada na seguinte relação de recorrência:

$$st(i,j) = \begin{cases} f(v[i]) & \text{se } j = 0 \\ f(st(i,j-1), st(i+2^{j-1}, j-1) & \text{c.c.} \end{cases}$$

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | | | | |
| | | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | | | | | | | |
| | | | | | | | |

$[i,j] \leftarrow [i,j-1]$ e $[i + 2^j, j-1]$

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | 3 | | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | 3 | 1 | | | |
| | | | | | | | |

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | 3 | 1 | | | |
| 1 | | | | | | | |

# Pré-computação

v =

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | 3 | 1 | | | |
| 1 | | | | | | | |

# Pré-computação

```
int st[MAXN][K+1];
void buildSparseTable(int v[], int n)
{
    for(int i = 0; i < n; i++)
        st[i][0] = f(v[i]);
    for(int j = 1; j <= K; j++)
        for(int i = 0; i + (1 << j) <= N; i++)
            st[i][j] = f(st[i][j-1],
                         st[i + (1 << (j-1))][j-1]);
}
//Complexidade de tempo e memória O(nlog(n))
```

# Consultas em O(log n)

- Para realizar uma consulta em O(log n) em uma Sparse Table, vamos partir da ideia de dividir o intervalo em subintervalos de comprimento potência de 2.
- Para um intervalo [L,R}, iteramos sobre todas as potências de dois, começando pela maior. Assim que uma potência $2^j$ for menor ou igual ao comprimento do intervalo (R - L + 1), processamos a primeira parte do intervalo [L, L+$2^j$-1] e continuamos com o intervalo restante [L + $2^j$, R].

# Consultas em O(log n)

```
int query(int L, int R)
{
    int acc = elem_neutro;
    for(int j = K; j >= 0; j--){
        if ((1 << j) <= R - L + 1){
            acc = f(acc, st[L][j]);
            L += 1 << j;
        }
    }
}
```

# Consultas em O(log n)

```
int query(int L, int R) //Exemplo: versão específica para soma
{
    int sum = 0;
    for(int j = K; j >= 0; j--){
        if ((1 << j) <= R - L + 1){
            sum += st[L][j];
            L += 1 << j;
        }
    }
}
```

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | **1** | **5** | **3** | **4** | **7** | **6** | **1** |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

$C = R - L + 1 = 7$

$j = 3 \rightarrow 2^3 = 8 > 7$

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

C = R - L + 1 = 7

j = 2 → $2^2$ = 4 <= 7 → sum += st[1][2]

sum = 13

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | **1** | **5** | **3** | **4** | **7** | **6** | **1** |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

C = R - L + 1 = 3

j = 1 ➜ $2^1$ = 2 <= 3 ➜ sum += st[5][1]

sum = 26

# Consultas em O(log n)

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | **1** | **5** | **3** | **4** | **7** | **6** | **1** |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 4 | 6 | 8 | 7 | 11 | 13 | 7 | |
| 12 | 13 | 19 | 20 | 18 | | | |
| 30 | | | | | | | |

C = R - L + 1 = 1

j =0 ➜ $2^0 = 1$ <= 1 ➜ sum += st[7][0]

sum = 27

# Consultas em O(1)

- A Sparse Table apresenta grande vantagem quando estamos trabalhando com funções que permitem a **sobreposição** de problemas. Ou seja, ao dividimos um intervalo em subintervalos, **NÃO** precisamos nos preocupar se eles são **disjuntos**.
- Ou ainda, quando temos funções **idempotentes**: f(x,x) = x
- Isso vale para calcularmos o mínimo de um intervalo, por exemplo, mas não vale para a soma.
    - min([1, 6]) = min([1,4], [3,6])
    - sum([1, 6]) != sum([1,4], [3,6]) //Os elementos 3 e 4 são somados duas vezes

# Consultas em O(1)

- Se a função com que estamos trabalhando tiver esta propriedade, iremos dividir nosso intervalo em apenas dois subintervalos sobrepostos com comprimentos de potência de 2.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

# Consultas em O(1)

- Em resumo, para uma consulta (L,R) vamos selecionar como j a maior potência de 2 tal que $2^j$ <= R - L + 1, e então pegar o intervalo que começa em L de tamanho $2^j$, e o intervalo que termina em R de tamanho $2^j$.

$$min(st[L][j], st[R - 2^j + 1][j])$$

$$\text{onde } j = \lfloor log_2(R - L + 1) \rfloor$$

# Pré-computação

v =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |

st =

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 5 | 3 | 4 | 7 | 6 | 1 |
| 1 | 1 | 3 | 3 | 4 | 6 | 1 | |
| 1 | 1 | 3 | 3 | 1 | | | |
| 1 | | | | | | | |

# Consultas em O(1)

- Para esta operação, precisamos ser capazer de calcular $\log_2(R - L + 1)$ rapidamente. Usualmente, fazemos isso pré-computando todos os logaritmos

```
int log[MAXN+1];

log[1] = 0;
for(int i = 2; i <= MAXN; i++)
    log[i] = log[i/2] + 1;
```

# Consultas em O(1)

- Então, tendo construído a Sparse Table, a consulta de um intervalo [L, R] se torna bastante simples:

```
int j = log[R - L + 1];
int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
```

# Variações

- Existem estruturas de dados semelhantes a Sparse Table que conseguem realizar consultas em O(1) mesmo com funções que não sejam idempotentes.
- Exemplos:
  - Disjoint Sparse Table: [Tutorial] Disjoint Sparse Table - tutorial
  - Sqrt Tree: Sqrt Tree

# Comparativo

| Segment Tree | BIT | Sparse Table |
| --- | --- | --- |
| Complexidades<br>• Preprocess: O(n)<br>• Query: O(log n)<br>• Update: O(log n)<br><br>Obs:<br>• Bastante versátil<br>• Vantagem com atualizações em intervalos com lazy propagation | Complexidades<br>• Preprocess: O(n)<br>• Query: O(log n)<br>• Update: O(log n)<br><br>Obs:<br>• Apenas para RSQ (ou funções similares)<br>• Embora tenha a mesma complexidade da SegTree, na prática é mais eficiente tanto em relação a tempo quanto memória | Complexidades<br>• Preprocess: O(n log(n))<br>• Query: O(1) ~ O(log n)<br>• Update: O(n log n)<br><br>Obs:<br>• Grande vantagem em *queries* de funções idempotentes. *Query* em O(1)<br>• Não indicado para quando temos atualizações. |

# Referências


https://cp-algorithms-brasil.com/Estruturas%20de%20dados/sparsetable.html
https://cp-algorithms.com/data_structures/sparse-table.html
https://brilliant.org/wiki/sparse-table/
https://www.geeksforgeeks.org/sparse-table/