ESCOLA DE PRIMAVERA DA MARATONA SBC DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG UNIFEI UFMG UFU UFV

APOIO:

beecrowd SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

BIT

Por: Gabriel Ubiratan

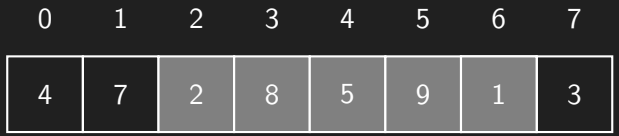
01-PROBLEMA MOTIVADOR

Problema:

Dado um array **a** de **n** inteiros, precisamos responder várias queries da seguinte forma:

- Query: dado um intervalo **[l,r]**, qual a soma do subarray **a[l..r]**
- $\sum_{l \leq i \leq r} a[i]$

Exemplo:



02-SOMA DE PREFIXO

Criamos um array com as somas dos prefixos

- $p[i] = \sum_{j \leq i} a[i]$

Exemplo:

	0	1	2	3	4	5	6	7
a:	4	7	2	8	5	9	1	3
p:	4	11	13	21	26	35	36	39

02-SOMA DE PREFIXO

Criamos um array com as somas dos prefixos

- $p[i] = \sum_{j \leq i} a[i]$

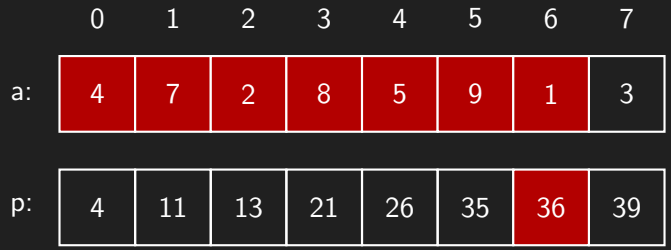
Exemplo:

02-SOMA DE PREFIXO

Criamos um array com as somas dos prefixos

- $p[i] = \sum_{j \leq i} a[i]$

Exemplo:

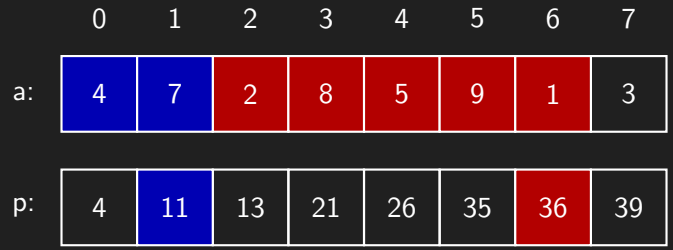


02-SOMA DE PREFIXO

Criamos um array com as somas dos prefixos

- p[i] = $\sum_{j \leq i}$ a[i]

Exemplo:

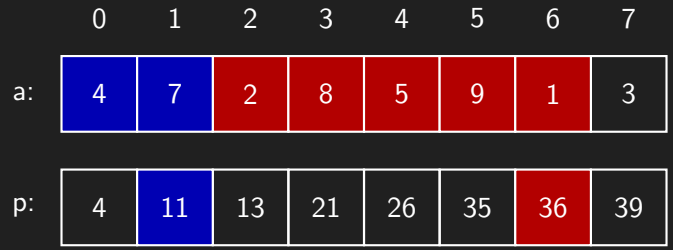


02-SOMA DE PREFIXO

Criamos um array com as somas dos prefixos

- p[i] = $\sum_{j \leq i}$ a[i]

Exemplo:



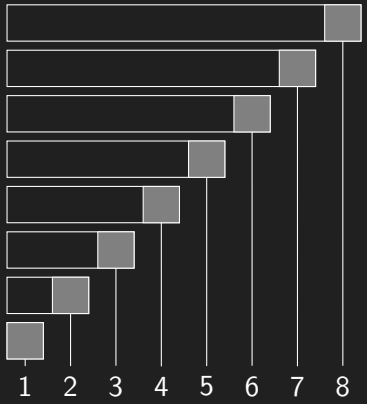
CÓDIGO

```cpp
int n; cin >> n;
vector<int> a(n), p(n);
for (int& ai : a) cin >> ai;

p[0] = a[0];
for (int i = i; i < n; i++)p[i] = p[i-1]+ a[i];

int q;
cin >> q;
while(q--){
    int l, r; cin >> l >> r;
    int ans = p[r];
    if(l > 0) ans -= p[l-1];
    cout << ans << endl;
}
```

ESTRUTURA



03-PROBLEMA MOTIVADOR 2

Problema:
Dado um array **a** de **n** inteiros, precisamos responder várias queries e processar updates da seguinte forma:

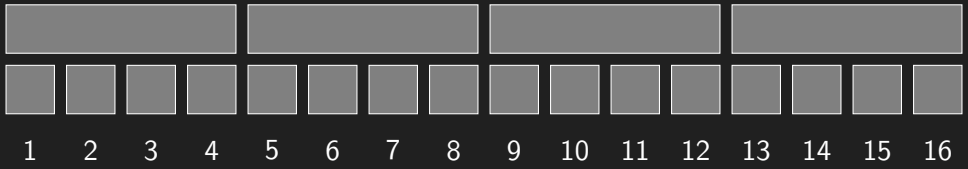
- Query: dado um intervalo **[l,r]**, qual a soma do subarray **a[l..r]**
- Update: dado um índice **i** e um inteiro **c**, adicionar **c** ao **a[i]**

Exemplo:

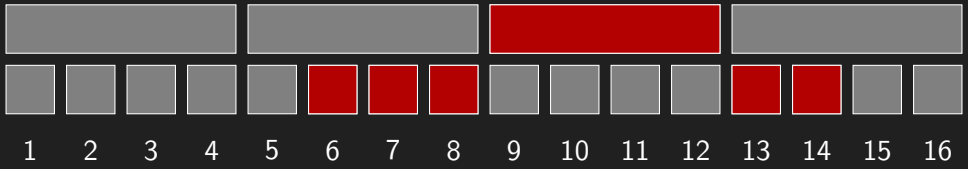
04-COMPLEXIDADE

algoritimo	ingênuo	somas de prefixos	sqrt tree
query	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$
update	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$

SQRT TREE



SQRT TREE

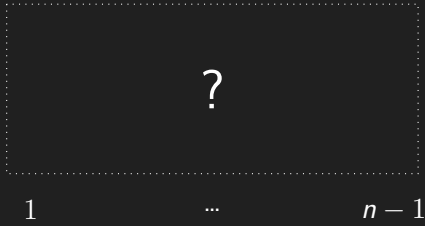


IDEIA

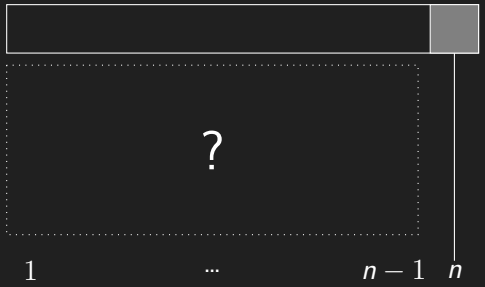
Como vimos na soma de prefixos, se conseguimos calcular a soma em prefixos rapidamente, também conseguimos calcular a soma em intervalos rapidamente.
E como vimos na sqrt tree, se cada índice está em no máximo k dos intervalos que estamos mantendo, conseguimos atualizar os intervalos em $\mathcal{O}(k)$.
Então queremos escolher intervalos de forma que:

- todo prefixo pode ser calculado como a soma de poucos intervalos
- todo índice está em poucos intervalos

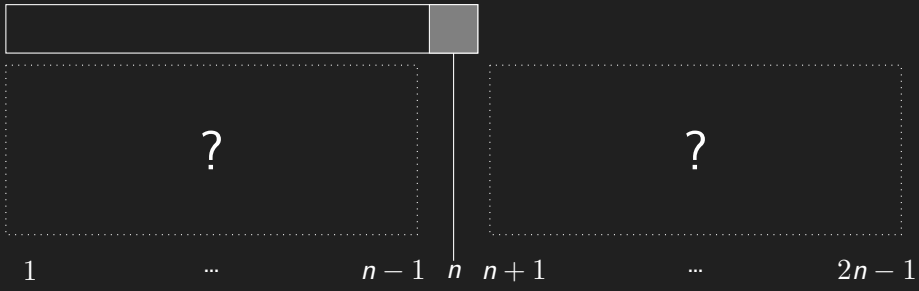
IDEIA



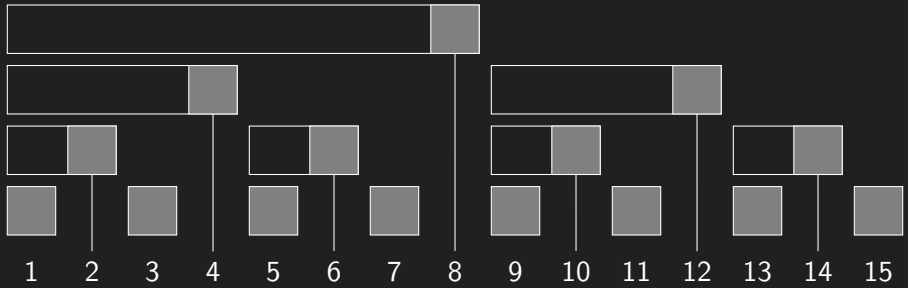
IDEIA



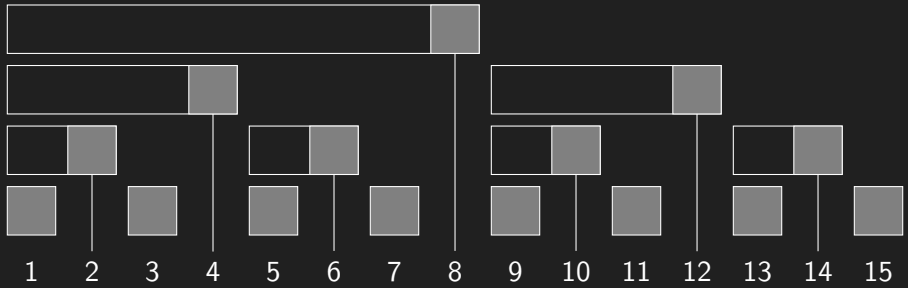
IDEIA



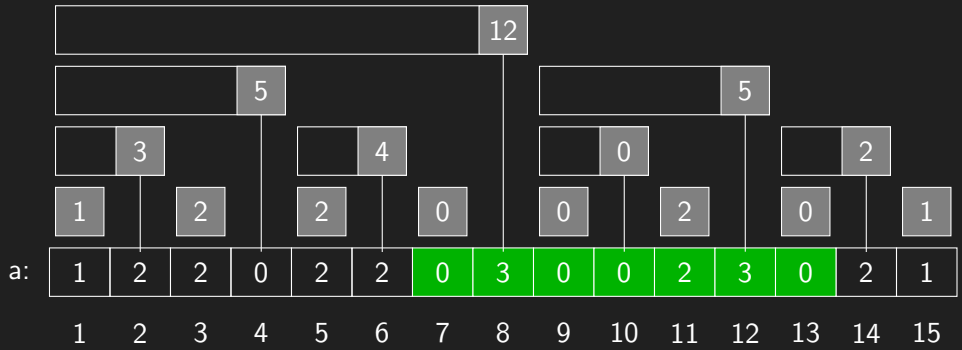
05-BIT



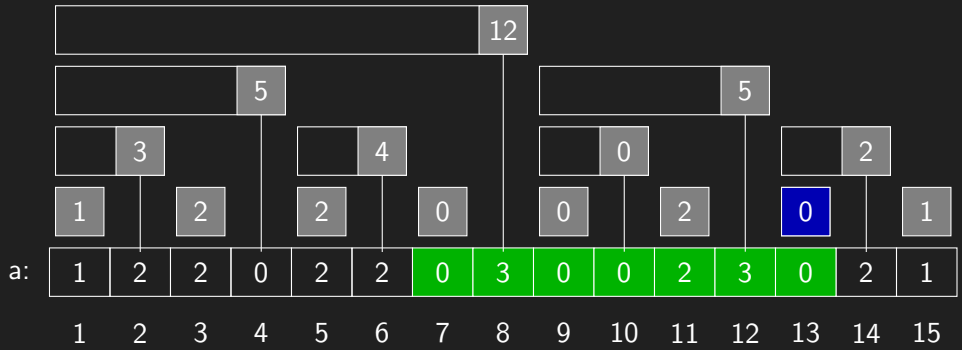
05-BIT



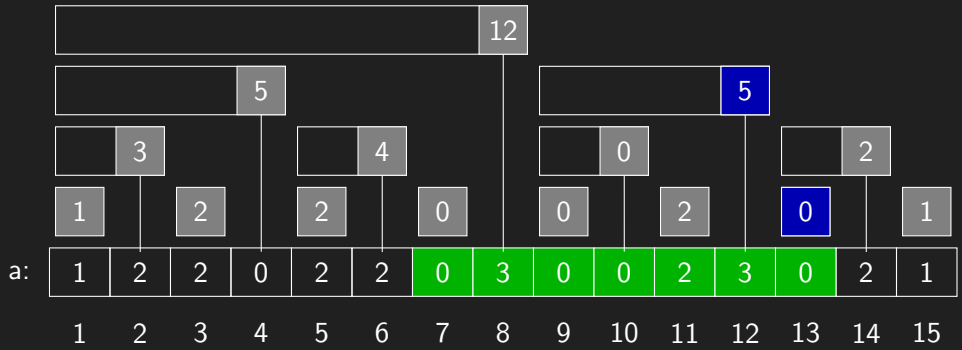
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

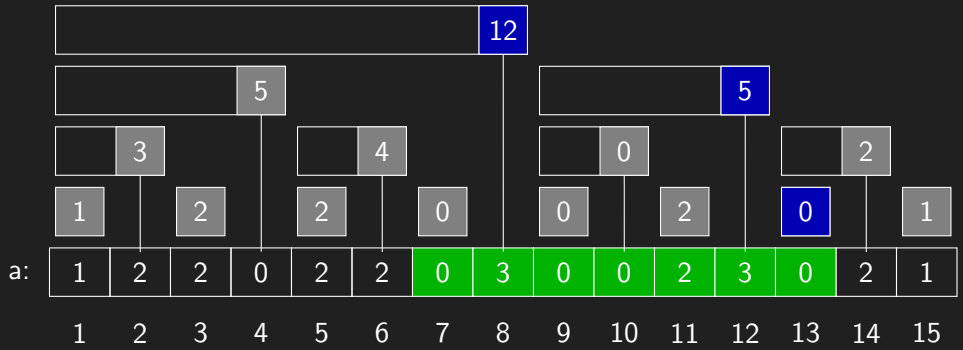
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

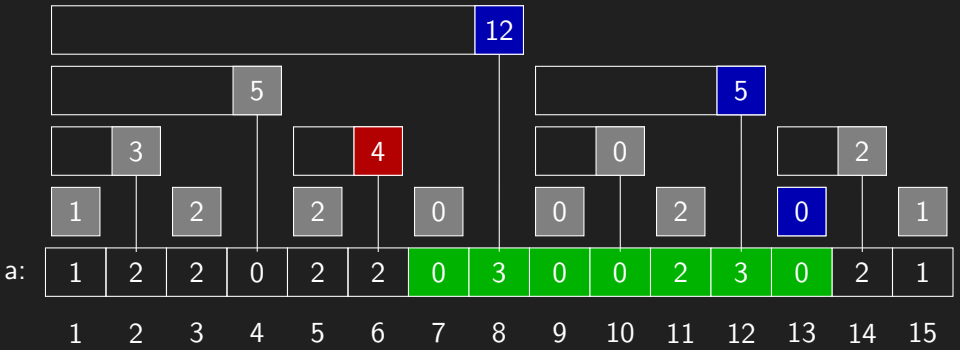
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

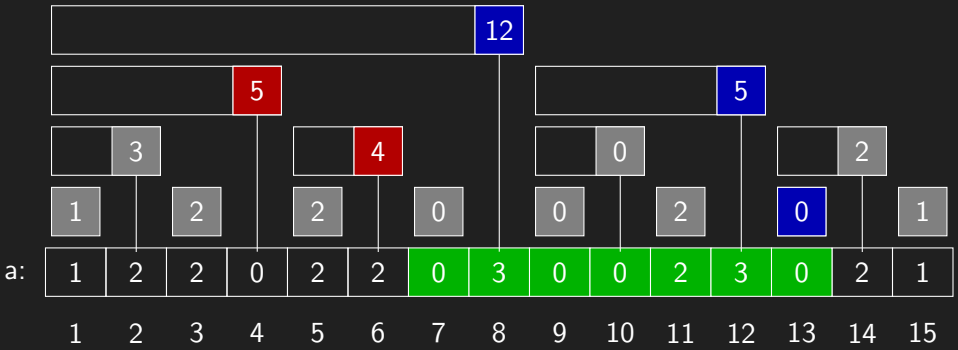
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

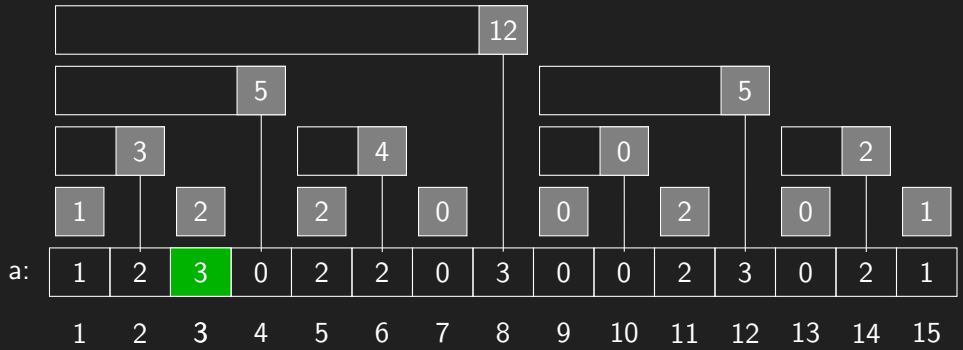
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

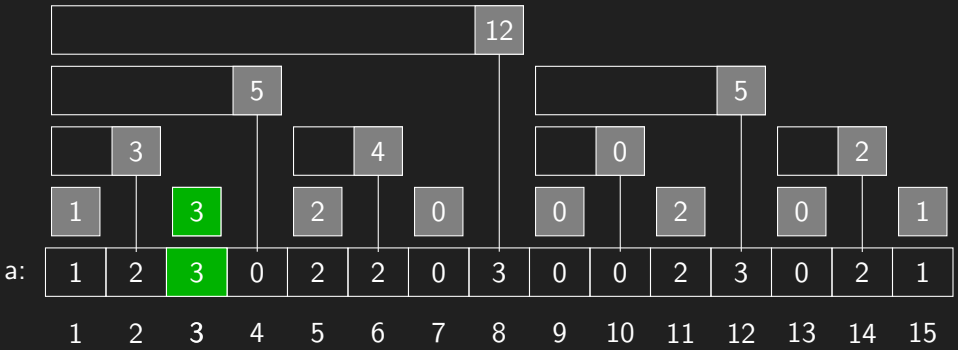
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 2 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

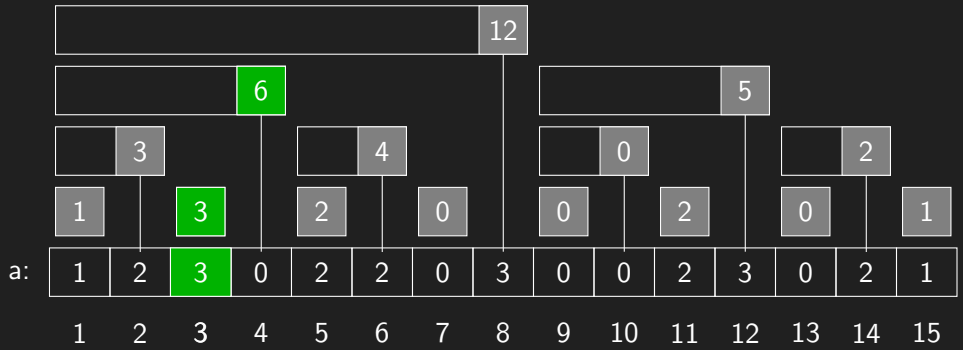
EXEMPLO


12
5
5
3
4
0
2
1
2
2
0
0
2
0
1
a:
1 2 3 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

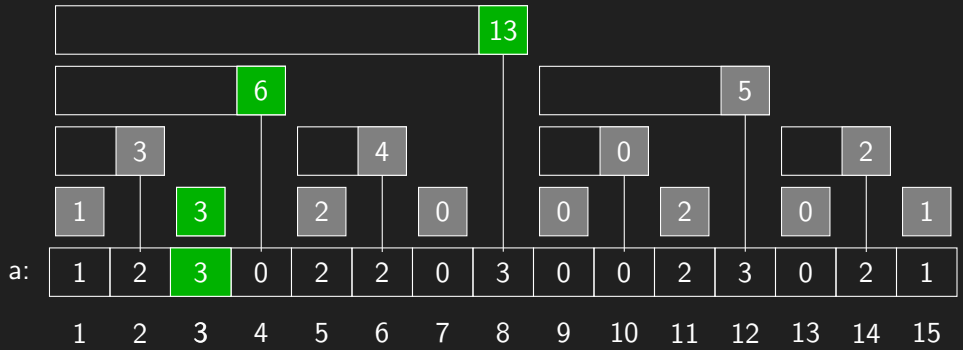
EXEMPLO


12
5
5
3
4
0
2
1
3
2
0
0
2
0
1
a:
1 2 3 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

EXEMPLO


12
6
5
3
4
0
2
1
3
2
0
0
2
0
1
a:
1 2 3 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

EXEMPLO


13
6
5
3
4
0
2
1
3
2
0
0
2
0
1
a:
1 2 3 0 2 2 0 3 0 0 2 3 0 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

COMPLEXIDADE

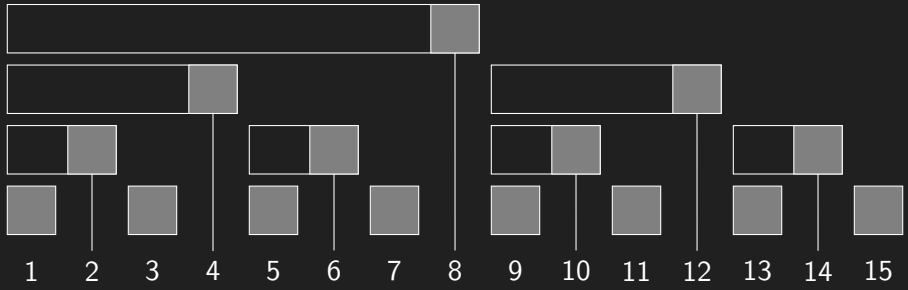
Se fizermos **K** iterações da construção recursiva, construímos uma BIT para um array de tamanho $2^{\mathbf{K}} - 1$ em que cada índice está em no máximo $\mathbf{K} + 1$ intervalos e cada prefixo pode ser construído com $\mathbf{K} + 1$ intervalos.

COMPLEXIDADE

Se fizermos **K** iterações da construção recursiva, construímos uma BIT para um array de tamanho $2^{\mathbf{K}} - 1$ em que cada índice está em no máximo $\mathbf{K} + 1$ intervalos e cada prefixo pode ser construído com $\mathbf{K} + 1$ intervalos.
Logo, para lidarmos com um array de tamanho n, precisamos no pior caso duplicá-lo para ele ficar do tamanho $2^{\mathbf{K}} - 1$ e neste caso updates e queries custam:

$$\mathcal{O}(K + 1) = \mathcal{O}(log(2n) + 1) = \mathcal{O}(log(n))$$

OBSERVAÇÕES



CÓDIGO

```
struct Bit {
  int n;
  vector<ll> bit;
  Bit(int _n=0) : n(_n), bit(n + 1) {}

  void update(int i, ll x) { // soma x na posicao i
    for (i++; i <= n; i += i & -i) bit[i] += x;
  }

  ll pref(int i) { // soma [0, i]
    ll ret = 0;
    for (i++; i; i -= i & -i) ret += bit[i];
    return ret;
  }
};
```

PROBLEMA EXEMPLO

3284 - Mega Inversões

06-PROBLEMA MOTIVADOR 3

Problema:
Dado um array **a** de **n** inteiros, precisamos responder várias queries e processar updates da seguinte forma:

- Query: dado um índice **i**, qual o valor de **a[i]**
- Update: dado um intervalo **[l,r]** e um inteiro **c**, adiciona **c** a todo **a[i]** no intervalo **[l,r]**

Exemplo:

06-PROBLEMA MOTIVADOR 3

Problema:
Dado um array **a** de **n** inteiros, precisamos responder várias queries e processar updates da seguinte forma:

- Query: dado um índice **i**, qual o valor de **a[i]**
- Update: dado um intervalo **[l,r]** e um inteiro **c**, adiciona **c** a todo **a[i]** no intervalo **[l,r]**

Exemplo:

	0	1	2	3	4	5	6	7
a:	4	7	3	9	6	10	2	3

07-RESOLUÇÃO DO PROBLEMA MOTIVADOR

Usaremos uma BIT como caixa preta.
Considere o array **b** definido como:

$$b[0] = a[0], b[i] = a[i] - a[i-1]$$

Observe que:

$$\sum_{i \leq r} b[i] = \sum_{1 \leq i \leq r} (a[i] - a[i-1]) + a[0] = a[r]$$

Logo, se mantivermos uma bit do b[i], conseguimos processar queries em $\mathcal{O}(\log n)$.

08-RESOLUÇÃO DO PROBLEMA MOTIVADOR

Observe que se adicionarmos uma constante **c** em um intervalo **[l,r]** apenas os **b[i]** nas pontas são modificados.

$$b'[l] = a'[l] - a'[l-1] = a[l] + c - a[l-1] = b[l] + c$$

$$b'[r+1] = a'[r+1] - a'[r] = a[r+1] - a[r] - c = b[r+1] - c$$

Logo conseguimos atualizar *b* com duas chamadas do update.

08- RESTRIÇÕES DOS ALGORÍTIMOS

- associatividade $((a \times b) \times c = a \times (b \times c))$
- comutatividade $(a \times b = b \times a)$
- inversa$(\exists a^{-1} : a^{-1} \times a = 1)$

algorítimos	sqrt tree	soma de prefixo	BIT
associatividade	NECESSÁRIO	NECESSÁRIO	NECESSÁRIO
inversa	DESNECESSÁRIO	NECESSÁRIO	NECESSÁRIO
comutatividade	DESNECESSÁRIO	DESNECESSÁRIO	NECESSÁRIO

09-EXEMPLO DE OPERAÇÕES

propriedades	associatividade	inversa	comutatividade
soma modular	SIM	SIM	SIM
produto módulo primo	SIM	exceto 0	SIM
bitwise XOR	SIM	SIM	SIM
mínimo	SIM	NÃO	SIM
float	cancelamento catastrófico		SIM
produto matricial	SIM	apenas invertíveis	NÃO
NAND	NÃO	NÃO	SIM

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg