# Decomposicao em raiz Quadrada

**Wladimir Araújo Tavares** [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

5 de junho de 2018

# Decomposicao em raiz quadrada

Problema:

- Entrada: Dado um vetor $A$ e conjunto de queries $(L, R)$
- Saída: Para cada query, compute o valor de $F(A[L], \ldots, A[R])$
- Complexidade Query : $O(\sqrt{n})$

# Intuição

- A ideia básica da decomposicão em raiz quadrada é dividir o vetor $a$ em blocos de tamanho aproximadamente $\sqrt{n}$
- Dessa maneira, o número de blocos e o tamanho de blocos será aproximadamente iguais.
- Para cada bloco $i$, armazene o valor da query em $b[i]$
- Seja $s = \lceil \sqrt{n} \rceil$, então o vetor $a$ é dividido em blocos da seguinte maneira:
- $b[0] = F(a[0], \ldots, a[s-1])$
- $b[1] = F(a[s], \ldots, a[2s-1])$
- $\ldots$
- $b[i] = F(a[is], \ldots, a[(i+1)s-1])$
- $\ldots$
- $b[s-1] = F(a[(s-1)s], \ldots, a[n])$
- O último bloco pode não ser completo.

# Intuição

- Para computar uma *query*$(l, r)$, temos que descobrir quais são os blocos que estão por completo dentro do intervalo $[l, r]$
- Em alguns casos, precisamos computar "à mão" a query para os blocos imcompletos.

# Range Sum Query

```
1  int n;
2  cin >> n;
3  vector<int> a (n);
4  int len = (int) sqrt (n + .0) + 1;
5  vector<int> b (len);
6  for (int i=0; i<n; ++i){
7    cin >> a[i];
8    b[i / len] += a[i];
9  }
```

# Range Sum Query

```
cin >> q;
for (int j = 0; j < q;j++) {
  int l, r;
  cin >> l >> r;
  int sum = 0;
  for (int i=l; i<=r; ){
    if (i % len == 0 && i + len - 1 <= r) {
      // se todo o bloco comecando em i está em [l, r]
      sum += b[i / len];
      i += len;
    }
    else {
      sum += a[i];
      ++i;
    }
  }
  cout << sum << endl;
}
```

# Evitando cálculos desnecessários

```
1  int sum = 0;
2  int c_l = l / len,    c_r = r / len;
3  if (c_l == c_r)
4    for (int i=l; i<=r; ++i)
5      sum += a[i];
6  else {
7    //Complexidade O(\sqrt{n})
8    for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
9      sum += a[i];
10   //Complexidade O(\sqrt{n})
11   for (int i=c_l+1; i<=c_r-1; ++i)
12     sum += b[i];
13   //Complexidade O(\sqrt{n})
14   for (int i=c_r*len; i<=r; ++i)
15     sum += a[i];
16 }
```

# Range Minimum Query

```
1 vector<int> a (n);
2 int len = (int) sqrt (n + .0) + 1;
3 vector<int> b;
4 b.assign(len, INT_MAX);
5 for (int i=0; i<n; ++i){
6   cin >> a[i];
7   b[i / len] = min( b[i / len] , a[i]);
8 }
```

# Range Minimum Query

```
1  int minimum = INT_MAX;
2  int c_l = l / len,    c_r = r / len;
3  if (c_l == c_r)
4    for (int i=l; i<=r; ++i)
5      minimum = min( minimum, a[i]);
6  else {
7    for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
8      minimum = min( minimum, a[i]);
9
10   for (int i=c_l+1; i<=c_r-1; ++i)
11     minimum = min( minimum, b[i]);
12
13    for (int i=c_r*len; i<=r; ++i)
14     minimum = min( minimum, a[i]);
15 }
```

# Notas

- Na decomposição em raiz quadrada, podemos permitir atualizações individuais nos elementos do vetor. Por exemplo, no caso range sum query, se o elemento $a[i]$ muda, basta modificar o bloco $b[i/s]$

$$b[i/s] += a_{novo}[i] - a_{velho}[i] \tag{1}$$

- Atualizar o valor de bloco inteiro pode ser realizado em $O(s) = O(\sqrt{n})$
- Alguns problemas podem ser encontrados aqui: `https://www.hackerearth.com/practice/algorithms/advanced-algorithms/square-root-decomposition/practice-problems/`