Árvore de Segmentos
Professor Wladimir

Soma de intervalos

Imagine que temos um vetor de inteiros $A[0 \dots n-1]$ e queremos responder rapidamente à seguinte pergunta várias vezes:

Qual é a soma dos elementos entre as posições i e j (inclusive)?

Por exemplo, se $A = [1, 3, 2, 6, 4]$, e queremos saber a soma dos elementos de $A[1]$ até $A[3]$, a resposta é:

$$3 + 2 + 6 = 11$$

Solução Ingênua (Tempo: $\mathcal{O}(n)$ por consulta)

Para cada consulta de soma, iteramos do índice i até j e somamos os elementos. Isso é ineficiente se houver muitas consultas.

Solução Eficiente com Soma Acumulada (Tempo: $\mathcal{O}(1)$ por consulta)

Construímos um vetor auxiliar S, chamado de **soma acumulada** (*prefix sum*), onde:

$$S[k] = A[0] + A[1] + \cdots + A[k]$$

Ou seja, $S[k]$ guarda a soma dos $k+1$ primeiros elementos de A.
Então, a soma no intervalo $A[i \dots j]$ pode ser calculada como:

$$\text{Soma}(i, j) = S[j] - S[i-1]$$

No caso especial $i = 0$, temos:

$$\text{Soma}(0, j) = S[j]$$

Comparativo

Operação	**Custo**	**Explicação**
Construção	$\mathcal{O}(n)$	Calculamos o vetor de prefixos com um loop simples.
Consulta	$\mathcal{O}(1)$	A soma de um intervalo $[i, j]$ consultando o vetor S.
Atualização	$\mathcal{O}(n)$	Se alteramos um valor do vetor original, todos os prefixos seguintes precisam ser recalculados.

Arvore de Segmentos

Considere o seguinte vetor:

	0	1	2	3	4	5	6	7
L	1	3	2	6	4	7	6	5

Vamos construir uma árvore binária na qual cada nó possui **exatamente 0 ou 2 filhos**, e todos os nós com 0 filhos (isto é, as folhas) estão localizados **no penúltimo ou no último nível** da árvore. Essa estrutura reflete o padrão típico de uma **árvore de segmentos**, garantindo que a árvore tenha **altura mínima possível**, isto é, $\lceil \log_2 n \rceil$, onde n é o número de elementos no vetor original.

- **Caso 1:** Quando n é uma potência de 2, ou seja, $n = 2^h$, a árvore terá todas folhas exatamente no último nível. O número total de nós da árvore (incluindo internos e folhas) será:

$$\text{Nodes} = 1 + 2 + 2^2 + \ldots + 2^h = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 = 2n - 1 \tag{1}$$

- **Caso 2:** Quando n não é uma potência de 2, podemos escrevê-lo como $n = 2^l + k$, com $0 < k < 2^l$. Nesse caso, o menor valor de l tal que $n < 2^{l+1}$ satisfaz:

$$2^l < n < 2^{l+1}$$

A árvore terá todas as folhas distribuídas entre o penúltimo e o último nível. O número de nós será, no máximo, o total de uma árvore com todos os nós folhas no último nível com altura $l + 1$, ou seja:

$$\text{Nodes} < \sum_{i=0}^{l+1} 2^i = 2^{l+2} - 1 = 4 \cdot 2^l - 1 \tag{2}$$

Como $n > 2^l$, e, portanto:

$$\text{Nodes} < 4 \cdot 2^l - 1 < 4n - 1 \tag{3}$$

Assim, podemos concluir que o número total de nós em uma árvore de segmentos binária construída sobre um vetor de tamanho n é, no pior caso, limitado superiormente por $4n - 1$. Uma estimativa prática e segura para alocação de memória é:

$$\boxed{\text{Número máximo de nós} \leq 4n}$$

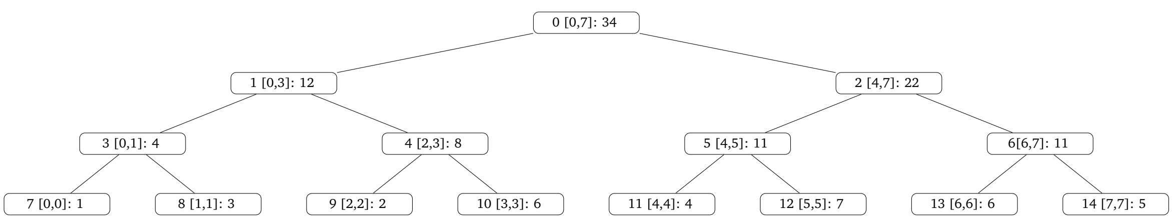

Figura 1: Árvore de segmentos para resolver o problema da soma de intervalos para o vetor = [1,3,2,6,4,7,6,5]

Construção da árvore de segmentos para soma de intervalos

```
#include<bits/stdc++.h>

#define MAXN 100000

using namespace std;

int tree[4*MAXN];
int N;

void buildAux(vector <int> & a, int index, int l, int r){
    if(l==r){
        tree[index] = a[l];
    }else{
        int mid = (l+r)/2;
        buildAux(a, 2*index+1, l, mid);
        buildAux(a, 2*index+2, mid+1, r);
        tree[index] = tree[2*index+1] + tree[2*index+2];

    }

    printf("tree[%d] = %d (%d, %d)\n", index, tree[index], l, r);
}

void build(vector <int> & a){
    buildAux(a, 0, 0, N-1);
}

int main(){
    int v[8] = {1,3,2,6,4,7,6,5};
    vector <int> a(v, v+8);
    N = a.size();
    build(a);

}


```

A saída do programa será:

```
tree[7] = 1 (0, 0)
tree[8] = 3 (1, 1)
tree[3] = 4 (0, 1)
tree[9] = 2 (2, 2)
tree[10] = 6 (3, 3)
tree[4] = 8 (2, 3)
tree[1] = 12 (0, 3)
tree[11] = 4 (4, 4)
tree[12] = 7 (5, 5)
tree[5] = 11 (4, 5)
tree[13] = 6 (6, 6)
tree[14] = 5 (7, 7)
tree[6] = 11 (6, 7)
tree[2] = 22 (4, 7)
tree[0] = 34 (0, 7)
```

Consulta

```
int consultaAux(int index, int tl, int tr, int l, int r){
    if( l > r ) return 0;
    if( l == tl && r == tr) return tree[index];
    int tm = (tl+tr)/2;
    int esq = consultaAux(2*index+1, tl, tm, l, min(r, tm) );
    printf("consulta esq (%d,%d) em (%d,%d) igual a %d\n", l, min(r, tm), tl, tm,  esq );
    int dir = consultaAux(2*index+2, tm+1, tr, max(l, tm+1), r );
    printf("consulta dir (%d,%d) em (%d,%d) igual a %d\n",  max(l, tm+1), r, tm+1, tr , dir );
    printf("consulta (%d,%d) em (%d,%d) igual a %d\n",  l, r , tl, tr,  esq + dir );

    return esq + dir;
}

int consulta(int l, int r){
    return consultaAux(0, 0, N-1, l, r);
}

int main(){
    int v[8] = {1,3,2,6,4,7,6,5};
    vector <int> a(v, v+8);
    N = a.size();
    build(a);
    consulta(2, 6);
}
```

```
consulta esq (2,1) em (0,1) igual a 0
consulta dir (2,3) em (2,3) igual a 8
consulta (2,3) em (0,3) igual a 8
consulta esq (2,3) em (0,3) igual a 8
consulta esq (4,5) em (4,5) igual a 11
consulta esq (6,6) em (6,6) igual a 6
consulta dir (7,6) em (7,7) igual a 0
consulta (6,6) em (6,7) igual a 6
consulta dir (6,6) em (6,7) igual a 6
consulta (4,6) em (4,7) igual a 17
consulta dir (4,6) em (4,7) igual a 17
consulta (2,6) em (0,7) igual a 25
```

Exercícios

1. Construa a **árvore de segmentos** para resolver consultas de **soma de intervalos** no vetor:

$$v = [1, 3, 2, 6, 4, 7, 6, 5, 8, 3, 4, 2]$$

   Consultas de soma de intervalos são consultas do tipo $\texttt{sum}(i, j) = \sum_{k=i}^{j} v[k]$.

2. Construa a **árvore de segmentos** para resolver consultas de **máximo em intervalos** sobre o mesmo vetor:

$$v = [1, 3, 2, 6, 4, 7, 6, 5, 8, 3, 4, 2]$$

   Consultas de máximo de intervalos são consultas do tipo $\texttt{max}(i, j) = max(v[i], v[i+1], \ldots, v[j])$.

3. Construa a **árvore de segmentos** para responder a consultas do **máximo divisor comum (MDC)** em intervalos do vetor:

$$v = [1, 3, 2, 6, 4, 7, 6, 5, 8, 3, 4, 2]$$

   Consultas de mdc de intervalos são consultas do tipo $\texttt{mdc}(i, j) = mdc(v[i], v[i + 1], \ldots, v[j])$.

4. Construa uma **árvore de segmentos** para responder a consultas de **ordenação de intervalos** no vetor:

$$v = [1, 3, 2, 6, 4, 7, 6, 5, 8, 3, 4, 2]$$

   Esse tipo de consulta, denotada por $\texttt{ordena}(i, j)$, retorna os elementos do subvetor $v[i \ldots j]$ em ordem crescente.

   Cada nó da árvore deve armazenar um vetor ordenado com os elementos do intervalo correspondente.

```
vector<int> tree[4 * MAXN];
```

   Implemente uma rotina que realiza a fusão de dois vetores ordenados (merge). Esse procedimento será útil para a construção e a consulta da árvore de segmentos.

5. Construa uma **árvore de segmentos** para responder a consultas de **interseção de conjuntos** em intervalos do vetor:

$$v = [\{1, 2, 3, 4\},\ \{2, 3, 5\},\ \{3, 4, 5\},\ \{1, 3, 5\},\ \{2, 3, 6\},\ \{0, 3, 4, 5\},\ \{3, 5, 7\},\ \{2, 3, 5, 8\}]$$

   As consultas, denotadas por $\texttt{inter}(i, j)$, devem retornar a interseção dos conjuntos armazenados nas posições $v[i], v[i + 1], \ldots, v[j]$, isto é:

$$\texttt{inter}(i, j) = \bigcap_{k=i}^{j} v[k]$$

   Cada nó da árvore armazena um conjunto contendo a interseção dos conjuntos do intervalo correspondente. Por exemplo, a raiz armazenará a interseção de todos os conjuntos de $v[0]$ até $v[7]$.

```
set<int> tree[4 * MAXN];
```