UNIVERSIDADE FEDERAL DE MINAS GERAIS
Graduação em Engenharia de Sistemas
Algoritmos e Estruturas de Dados III - Trabalho Prático 0
Operações de Nubby
Matheus Silva Araujo - 2013066265

---

1 Introdução

Este trabalho tem como objetivo comparar diferentes soluções para o problema *Operações de Nubby*. O problema consiste na obtenção dos valores máximo, mínimo e soma em intervalos de um vetor de tamanho n.

Os dois métodos a serem comparados são:

- **Matriz**: nesse método, será construída uma matriz $n \times n$ onde cada elemento $i \times j$ armazena os valores máximo, mínimo e soma do intervalo i a j.
- **Árvore**: nesse método, será construída uma árvore de segmentação, onde cada nó de árvore representará os valores máximo, mínimo e soma de um determinado intervalo, e as folhas representam um intervalo de um elemento.

2 Solução do Problema

Foram dados como entrada do problema: o vetor com os valores e também as operações nesse vetor:

- **Min** - Obter o menor valor em um intervalo $i \times j$
- **Max** - Obter o maior valor em um intervalo $i \times j$
- **Sum** - Obter a soma dos valores em um intervalo $i \times j$
- **Add** - Incrementar os valores em um intervalo $i \times j$
- **Sub** - Decrementar os valores em um intervalo $i \times j$

A saída deve ser o resultado das operações de consulta.

Para as duas soluções foi utilizada uma estrutura do tipo *intervalo*, que contém os três valores inteiros: mínimo, máximo e soma.

A matriz é uma matriz dessa estrutura, e a árvore é armazenada em um vetor dessa estrutura.

2.1 Matriz

O método da matriz consiste em construir uma matriz simétrica $n \times n$, onde cada elemento $i \times j$ armazena os valores máximo, mínimo e soma do intervalo i até j.

O algoritmo para construir a matriz é bastante simples: dois laços percorrem o vetor de 1 até n com índices i e j. A cada iteração os valores para o índice $i \times j$ são calculados com base nos valores do índice $i \times j - 1$ e do elemento j do vetor. O pseudo-código é mostrado no Algoritmo 1

Algoritmo 1 Construção da Matriz.

```
1: for i = 0 to n do
2:     m[i,i].max ← v[i]
3:     m[i,i].min ← v[i]
4:     m[i,i].sum ← v[i]
5:     for j = i + 1 to n do
6:         m[i,j].max ← m[j,i].max ← max(m[i,j-1].max, v[j])
7:         m[i,j].min ← ma[j,i].min ← min(m[i,j-1].min, v[j])
8:         m[i,j].sum ← ma[j,i].sum ← sum(m[i,j-1].sum, v[j])
```

Para consultar os valores de um intervalo $i \times j$, basta acessar o elemento de índice $i \times j$.

Nas operações de incremento e decremento, o vetor é atualizado e a matriz é reconstruída [1].

2.2 Árvore

O algoritmo de construção da árvore é uma função recursiva, onde se parte do intervalo 1 a n e a cada nova chamada o intervalo é partido ao meio até que se chegue a uma folha. Na folha, os valores de mínimo, máximo e soma são definidos como o valor correspondente do vetor. Para os demais nós, os valores são calculados em função dos valores dos filhos. O pseudo código é mostrado no algoritmo 2.

Algoritmo 2 Construção da Árvore.

Entradas
ind índice do nó atual no vetor da árvore
vi índice inicial do intervalo
vf índice final do intervalo

```
1: if vi = vf then
2:     a[ind].min ← v[vi]
3:     a[ind].max ← v[vi]
4:     a[ind].sum ← v[vi]
5: else
6:     vm ← valor médio entre vi e vf
7:     e ← nó esquerdo, ind*2 + 1
8:     d ← nó direito, ind*2 + 2
9:     construir(vi, vm, e)                    ▷ chamada recursiva
10:    construir(vm+1, vf, d)                  ▷ chamada recursiva
11:    a[ind].min = min(a[e], a[d])
12:    a[ind].max = min(a[e], a[d])
13:    a[ind].sum = sum(a[e], a[d])
```

Um pseudo-código para a consulta na árvore é mostrado no algoritmo 3. O algoritmo para a atualização é semelhante, com a diferença de que em vez de consulta, ele faz uma atualização dos valores.

[1] Seria possível implementar uma melhoria de forma que apenas o intervalo afetado pela alteração seja atualizado e não toda a matriz. No entanto, pelo anunciado do Trabalho Prático, essa poderia ser a solução. Pgs. 1 e 2

Algoritmo 3 Consulta na Árvore.

```
1: if o intervalo de consulta está totalmente dentro do nó then
2:     retornar valor solicitado
3: else if o intervalo de consulta está totalmente fora do nó then
4:     retonar elemento nulo
5: else
6:     vm ← valor médio entre vi e vf
7:     e ← nó esquerdo, ind∗2 + 1
8:     d ← nó direito, ind∗2 + 2
9:     consultar(vi, vm, e)                    ▷ chamada recursiva
10:    consultar(vm+1, vf, d)                  ▷ chamada recursiva
11:    retornar função dos valores consutaltados nos nós
```

3 Análise de Complexidade

Os dois algoritmos apresentam complexidades diferentes para a construção, atualização e consulta.

3.1 Matriz

- **Construção** A construção da matriz é um algortimo de ordem quadrática, pois necessita percorrer o vetor duas vezes para construí-la. Portanto $O(n^2)$.
- **Atualização** A atualização da matriz é a mesma operação da Construção, $O(n^2)$.
- **Consulta** A consulta na matriz é uma operação simples, bastando acessar uma posição. Portanto, $O(1)$.

3.2 Árvore

- **Construção** A construção da árvore de segmentação tem custo $O(n)$. Há $2 * n - 1$ nós e o valor de cada nó é calculado apenas uma vez.
- **Atualização** Para atualizar os nós, a complexidade é $O(log\, n)$. Ao atualizar uma folha, um nó em cada nível da árvore é atualizado.
- **Consulta** A complexidade da consulta é a mesma da atualização $O(log\, n)$, pela mesma razão. Ao fazer uma consulta, no pior caso, todos os níveis da árvore serão visitados.

4 Uso de memória

Os dois algoritmos também se diferem na utilização da memória.

4.1 Matriz

O algoritmo da matriz utiliza uma matriz $n \times n$, portano irá utilizar $n * n$ posições de memória. [2]

4.2 Árvore

No algoritmo da árvore de segmentação, a altura da árvore é $log_2 n$, e a memória utilizada será $2*2log_2 n-1$ posições de memória.

[2] Por ser uma matriz simétrica também, uma outra melhoria possível para o algoritmo seria armazenar apenas os elmentos $i \times j$ onde $i > j$.

5 Avaliação Experimental

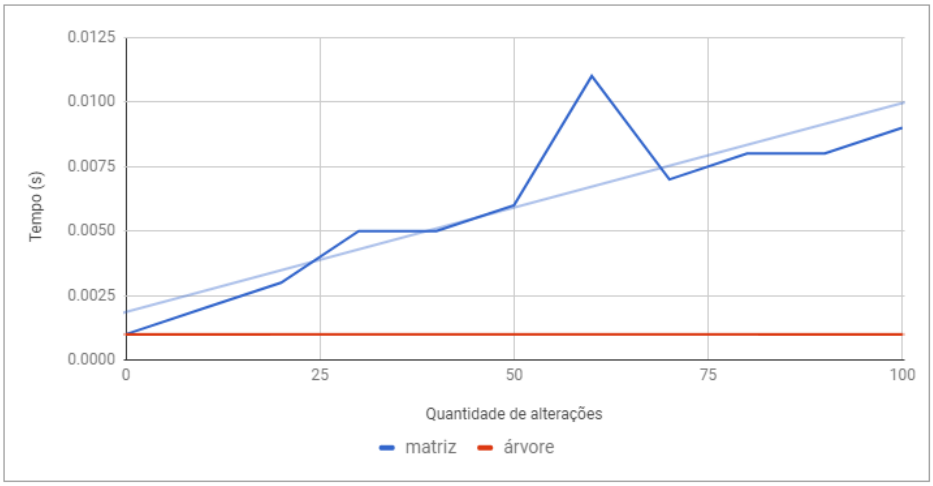
Para avaliar o desempenho das duas soluções foram feitos diversos testes onde o tamanho do vetor foi mantido fixo, 100, e a quantidade de operações de consulta e atualização foram modificadas entre 0 e 100, os resultados são apresentados na Tabela 1. Um gráfico com esses resultados é apresentado na Figura 1.

Os testes foram executados em uma máquina virtual com Linux Ubuntu. As medidas de tempo foram feitas pelo comando *time*.

Tabela 1: Resultados das execuções

Alterações	Consultas	Matriz (s)	Árvore (s)
0	100	0.001	0.001
10	90	0.002	0.001
20	80	0.003	0.001
30	70	0.005	0.001
40	60	0.005	0.001
50	50	0.006	0.001
60	40	0.011	0.001
70	30	0.007	0.001
80	20	0.008	0.001
90	10	0.008	0.001
100	0	0.009	0.001

Figura 1: Resultados da execuções



6 Conclusão

Como era imaginado pela análise de complexidade dos algoritmos, quando a quantidade de operações de alterações aumenta, o algoritmo de Matriz tem pior desempenho, enquanto o algoritmo da árvore de segmentação mantém o mesmo tempo de execução.

Os dois algoritmos solucionam o mesmo problema com diferentes tempos para diferentes configurações das entradas. O algoritmo da árvore tem melhor desempenho no caso geral, no entanto sua implementação é mais complexa e as operações de consulta nele são mais lentas. Dessa forma, a escolha de um ou outro algoritmo irá depender das características do problema real que se deseja resolver.