Trabalho Prático 3

Árvore de Segmentação

Raquel Gonçalves Rosa - 2020105815

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

raquelgr@ufmg.br

1. Introdução

Neste trabalho, nos foi proposto resolver um desafio matemático peculiar para as crianças da cidade de Nlogônia. Para resolver esse desafio, foi necessário implementar uma árvore de segmentação para armazenar as matrizes 2x2, e então realizar as transformações lineares. Tendo em vista isso, este documento possui o objetivo de explicitar a implementação da solução para o problema proposto.

2. Método

2.1 Implementação

O programa foi desenvolvido em C++ com compilador G++, em ambiente Linux.

2.2 Estrutura de dados

Para a implementação do programa foi criada uma classe e uma struct:

A classe **SegTree** (arquivo SegTree.hpp), responsável por armazenar a Árvore de Segmentação, está estruturada nos seguintes métodos:

- **Métodos privados**
    - SegTree:: MultiplicaMatrizes
    - SegTree:: AtualizaArvore
    - SegTree:: BuscaIntervalo

- **Métodos públicos**
    - SegTree:: Atualiza
    - SegTree:: Consulta

A struct **Matriz** (arquivo SegTree.hpp) é uma simples matriz 2x2, ela é utilizada na classe SegTree pois cada "nó" da árvore é uma matriz 2x2.

2.3 Organização

O programa está dividido em:

- 1 cabeçalho, contido na pasta *include*, **SegTree.hpp**.
- 2 arquivos de código, contidos na pasta *src*, sendo eles **SegTree.cpp** e **Main.cpp,** sendo este o responsável pela leitura dos comandos passados via terminal e chamada das funções necessárias.

3. Análise de Complexidade

A seguir será realizada a análise de complexidade de tempo de todos os métodos implementados no projeto.

→ **Método MultiplicaMatrizes:**
- O método de multiplicação de matrizes é o caso clássico, sua complexidade é cúbica, ou seja $O(n^3)$.

→ **Método AtualizaArvore:**
- A atualização da árvore se dá por meio do conceito de dividir para conquistar, ou seja, dependendo da posição passada, percorre-se o lado esquerdo ou o direito da árvore, analisando apenas essa parte, o método é $O(\log \ n)$. Entretanto, sempre que ocorre a atualização, ocorre também a pré-computação da matriz do nó, como supracitado, o método de multiplicação é $O(n^3)$, sendo essa a maior complexidade, o método num geral é $O(n^3)$.

→ **Método BuscaIntervalo:**
- A busca de um intervalo na árvore acontece de forma similar a atualização, ou seja, dependendo do intervalo passado, percorre-se o lado esquerdo ou o direito da árvore, buscando apenas nessa parte, o método é $O(\log \ n)$. Entretanto, no pior caso, é necessário chamar a função de multiplicar matrizes e como supracitado, o método de multiplicação é $O(n^3)$, sendo essa a maior complexidade, o método num geral é $O(n^3)$.

4. Estratégias de Robustez

Algumas verificações foram implementadas a fim de garantir a corretude dos algoritmos, foram feitas validações quanto ao nascimento e morte do instante e uma verificação de posição válida para a função de atualização. Para verificar a ausência de vazamento de memória, foi utilizado a ferramenta V*algrind,* como mostra a figura abaixo.

5. Análise Experimental

Utilizando a ferramenta *gprof* com a entrada:

4 6

u 0

0 1

99999999 0

u 1

0 1

99999999 0

u 2

0 1

99999999 0

u 3

0 1

99999999 0

q 0 3 1 0

q 0 3 0 1

Foram obtidos os seguintes resultados:

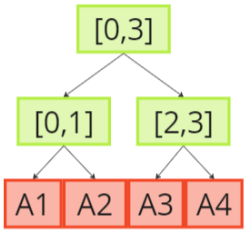
```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        8     0.00     0.00  SegTree::MultiplicaMatrizes(Matriz, Matriz)
  0.00      0.00     0.00        4     0.00     0.00  SegTree::AtualizaArvore(int, int, int, int, Matriz)
  0.00      0.00     0.00        4     0.00     0.00  SegTree::Atualiza(int, Matriz)
  0.00      0.00     0.00        2     0.00     0.00  SegTree::BuscaIntervalo(int, int, int, int, int)
  0.00      0.00     0.00        2     0.00     0.00  SegTree::Consulta(int, int)
  0.00      0.00     0.00        2     0.00     0.00  frame_dummy
  0.00      0.00     0.00        1     0.00     0.00  SegTree::SegTree(int)
  0.00      0.00     0.00        1     0.00     0.00  SegTree::~SegTree()
```

```
index % time    self  children    called     name
                0.00    0.00       2/2           main [6]
[1]      0.0    0.00    0.00       2         frame_dummy [1]
-----------------------------------------------
                0.00    0.00       8/8           SegTree::AtualizaArvore(int, int, int, int, Matriz) [9]
[8]      0.0    0.00    0.00       8         SegTree::MultiplicaMatrizes(Matriz, Matriz) [8]
-----------------------------------------------
                                   8             SegTree::AtualizaArvore(int, int, int, int, Matriz) [9]
                0.00    0.00       4/4           main [6]
[9]      0.0    0.00    0.00       4+8       SegTree::AtualizaArvore(int, int, int, int, Matriz) [9]
                0.00    0.00       8/8           SegTree::MultiplicaMatrizes(Matriz, Matriz) [8]
                                   8             SegTree::AtualizaArvore(int, int, int, int, Matriz) [9]
-----------------------------------------------
                0.00    0.00       4/4           main [6]
[10]     0.0    0.00    0.00       4         SegTree::Atualiza(int, Matriz) [10]
-----------------------------------------------
                0.00    0.00       2/2           SegTree::Consulta(int, int) [12]
[11]     0.0    0.00    0.00       2         SegTree::BuscaIntervalo(int, int, int, int, int) [11]
-----------------------------------------------
                0.00    0.00       2/2           main [6]
[12]     0.0    0.00    0.00       2         SegTree::Consulta(int, int) [12]
                0.00    0.00       2/2           SegTree::BuscaIntervalo(int, int, int, int, int) [11]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[13]     0.0    0.00    0.00       1         SegTree::SegTree(int) [13]
-----------------------------------------------
                0.00    0.00       1/1           main [6]
[14]     0.0    0.00    0.00       1         SegTree::~SegTree() [14]
-----------------------------------------------
```

Para esta entrada, era esperada a seguinte árvore de segmentação:



Com base nos resultados do gprof, observamos que, para cada atualização (flag ‘u’), a função AtualizaArvore foi chamada recursivamente 2 vezes, pois era indispensável acessar as folhas, e a função MultiplicaMatrizes foi chamada exatamente 8 vezes, 2 para cada atualização, pois como explicado anteriormente é necessário atualizar os intervalos que contém o nó que sofreu a atualização.

Da mesma forma, o gprof mostra que, para cada consulta (flag ‘q’), a função BuscaIntervalo foi chamada apenas 2 vezes, não sendo necessário a chamada recursiva uma vez que em ambas as consultas foram passados os instantes 0 e 3, e esses correspondiam exatamente ao intervalo “pai” - a raiz da árvore.

6. Conclusões

A resolução do problema usando uma árvore de segmentação foi de fato uma ótima solução, pois é um algoritmo simples e com uma baixa complexidade, apesar da necessidade de multiplicar matrizes, a complexidade foi pouco afetada como mostram os resultados do *gprof* e dessa maneira o programa como um todo se torna bem eficiente.

7. Bibliografia

FOLLOW, S. C. S. **Two dimensional segment tree**. Disponível em: <https://www.geeksforgeeks.org/two-dimensional-segment-tree-sub-matrix-sum/>. Acesso em: 4 dez. 2023.

Segment tree. Disponível em: <https://www.geeksforgeeks.org/segment-tree-efficient-implementation/>. Acesso em: 4 dez. 2023.

Segment tree - algorithms for competitive programming. Disponível em: <https://cp-algorithms.com/data_structures/segment_tree.html>. Acesso em: 4 dez. 2023.