CES-12 - Relatório Lab 2 - Balanced Tree

Aluno: Bruno Costa Alves Freire

14 de Maio de 2020

1 Estrutura de Dados e Implementação

A implementação do problema do índice foi feita utilizando a estrutura de Árvore Rubro-Negra, baseado totalmente no livro do Cormen, *Introduction to Algorithms, 3rd Ed.*, capítulos 12 e 13. A estrutura criada na classe consiste apenas de um nó raiz e um nó *nil*, conforme feito no Cormen, além de uma variável para o tamanho.

Para a implementação do método `add()`, foi seguido o pseudocódigo no capítulo 13 do Cormen, com a criação de métodos privados na classe principal para performar os métodos de rotação e garantir a validade das propriedades rubro-negras. Para o método `find()`, foi implementado um percurso em-ordem da árvore com a condição de que, sempre que a chave do nó sendo visitado estivesse fora do intervalo desejado, descartamos (do percurso) esse nó e sua subárvore esquerda ou direita, conforme a chave do nó esteja abaixo ou acima do intervalo.

2 Comparação da Complexidade da Implementação

Para verificar a corretude da implementação (no sentido da complexidade; os testes passaram corretamente), vamos comparar os tempos de busca e inserção na árvore para a implementação STL e para a implementação desenvolvida no lab.

Após gerar o arquivo `oak_search.csv` a partir dos testes, fez-se um tratamento dos dados visando a mitigação do ruído, ou seja, geramos os dados do arquivo várias vezes e tomamos uma média móvel exponencial dos valores do arquivo sobre as várias versões geradas. É o equivalente a medirmos os tempos das operações várias vezes e tomarmos uma média, visando mitigar o ruído.

Na figura 1, temos o gráfico em escala logarítmica das curvas de tempo vs. quantidade de elementos para as duas implementações, do método de busca, juntamente com a curva logarítmica ajustada (expressão do tipo $a + b \cdot \log n$).

Como podemos observar, a curva relativa à minha implementação ficou bem abaixo da curva da implementação STL, a qual sabemos que opera em $O(\log(n))$. Além disso, observando a disposição da curva ajustada, vemos que há um ajuste relativamente bom, apesar do ruído nas medidas de tempo. Escolhendo constantes de ajuste diferentes, é possível obter uma envoltória logarítmica para as curvas de tempo, mostrando que de fato o método de busca (em ambas as implementações) roda em $O(\log(n))$.

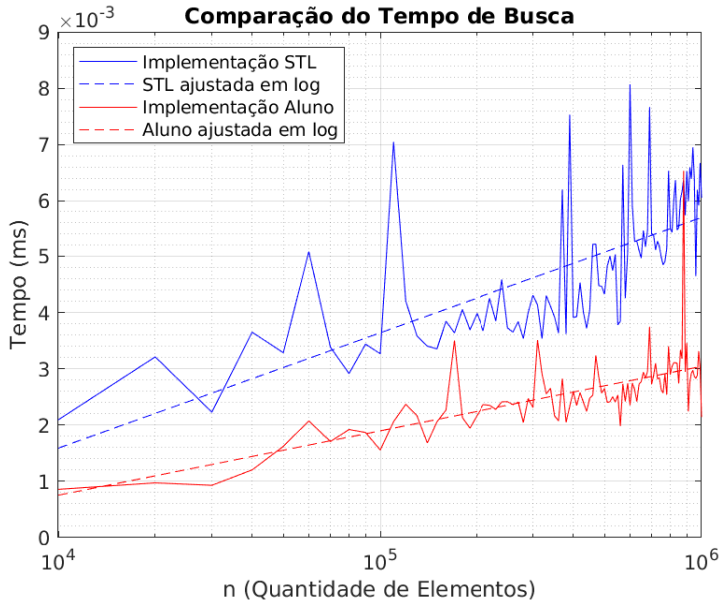

Figura 1: Gráfico comparativo do tempo da operação de busca em função do tamanho do índice, na implementação STL e na minha.

Na figura 2, temos o gráfico comparativo das duas implementações para o método de inserção. Novamente podemos observar a maior rapidez da minha implementação em relação à implementação STL. Note, dessa vez, que os tempos absolutos estão numa ordem de grandeza bem maior que os tempos da operação de busca.

A análise das curvas de tempo é semelhante à do gráfico anterior e nos mostra que também a inserção conforme implementada no lab opera em $O(\log(n))$.

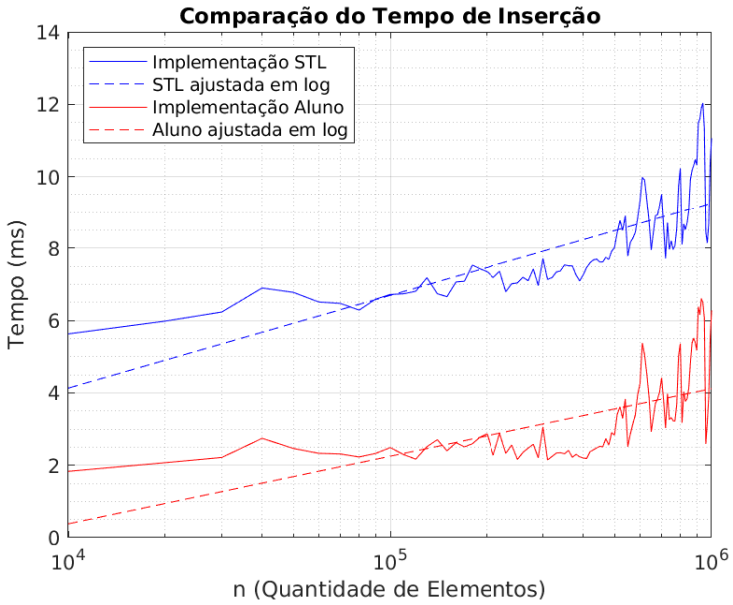

Figura 2: Gráfico comparativo do tempo da operação de inserção em função do tamanho do índice, na implementação STL e na minha.

Como uma última checagem da corretude da implementação, temos a comparação feita no teste `VoidSphereSelection`. Os arquivos de pontos gerados `torusMMP.asc` e `torusALU.asc` possuem ambos 16328 pontos, e os tempos de seleção (para uma execução, apenas a título de exemplo, uma vez que é um dado ruidoso) foram de 1.1315 ms no MMP e de 0.795058 ms no aluno.

As malhas geradas nos arquivos `torusMMP.asc` e `torusALU.asc` podem ser analisadas no *meshlab*, gerando as figuras 3D conforme os *prints* na figura 3

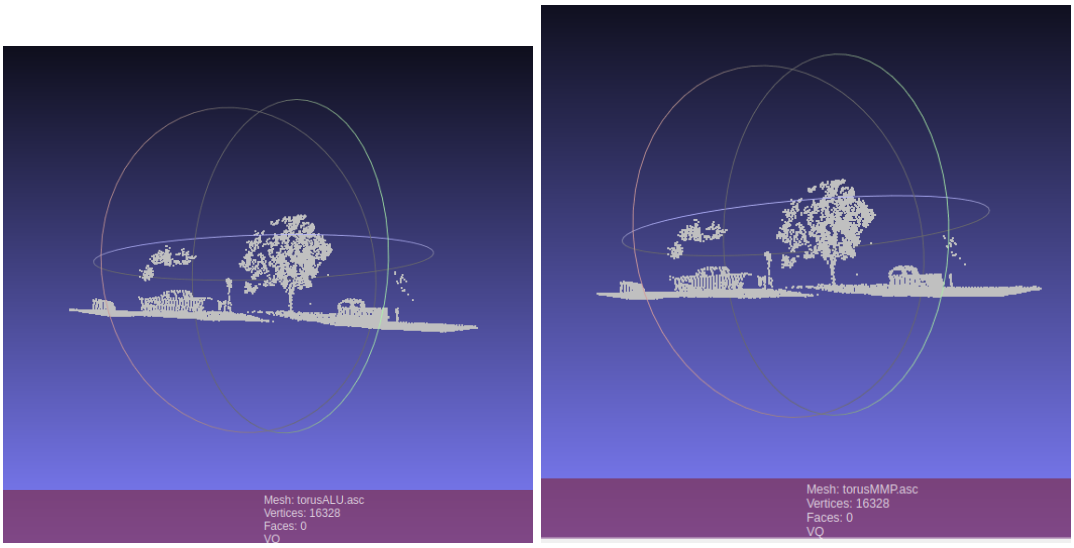

Figura 3: Malhas de pontos geradas pela seleção do teste `VoidSphereSelection` a partir das duas implementações.