Relatório de CTC-12 / 2023

Laboratório 02 - Árvore de busca balanceada

Aluno

Marcel Versiani e Silva

Turma COMP.25

Professor

Luiz Gustavo Bizarro Mirisola

Instituto Tecnológico de Aeronáutica - ITA

1

Escolha da estrutura de dados.

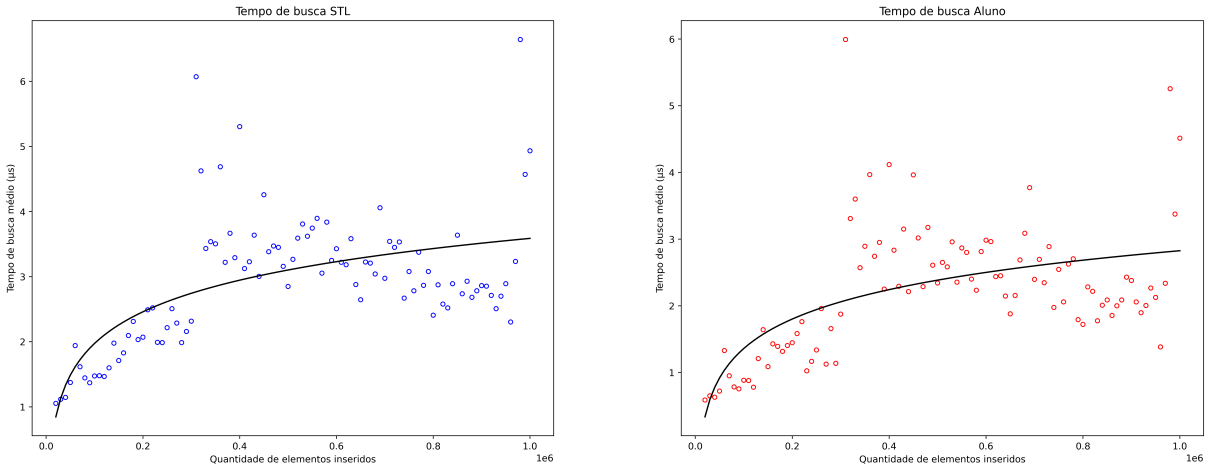
A estrutura de dados escolhida para este laboratório foi a árvore rubro-negra, pois sua altura é limitada por O($logn$) e com isso as operações de busca e inserção na árvore binária de busca, que no pior caso são O(h) sendo h a altura da árvore, serão bastante eficientes para um grande número de dados.

Como base para a implementação, foi utilizada o livro *Introduction to Algorithms - 3rd Ed.* do *Thomas Cormen*, que contém os pseudocódigos dos algoritmos mais importantes das árvores binárias de busca e, especificamente, da árvore rubro-negra e suas nuances computacionais como o algoritmo de *Fix Up* e o nó *NIL*.

No caso deste Laboratório, eu criei dois arquivos adicionais, que são o *rbtree.cpp* e o *rbtree.h* que contêm a minha classe *RB_Tree*, e com isso foi feita a substituição do *std::multimap* pela minha classe de árvore rubro-negra. Desse modo, as funções do *IndexPointAluno* retornarão métodos da nova variável *_tree*. No caso da classe, ela foi feita utilizando uma *struct* com os devidos ponteiros e outras variáveis de dados, e no *private* há o acesso aos ponteiros da *root_* e do *NIL* pelas funções públicas e privadas, além do tamanho da árvore. O construtor e destrutor foram implementados, além do algoritmo de *find* utilizando recursão.

2

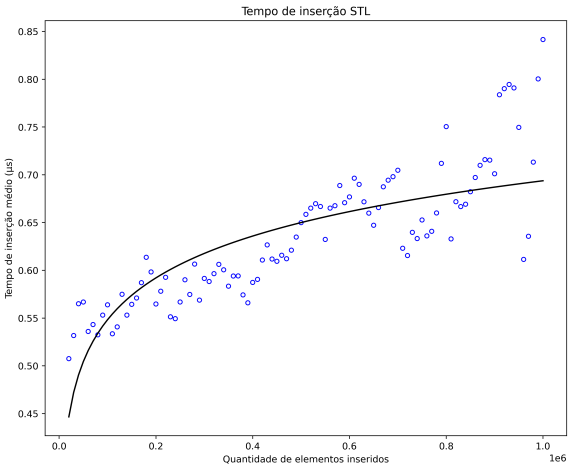
Curvas de tempo do teste *OakByNorm*.



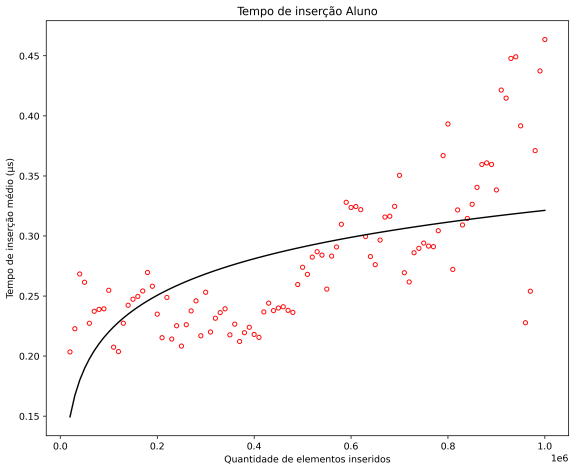
(a) $t(n) = 0.701 \cdot ln(n) - 6.091$ (b) $t(n) = 0.637 \cdot ln(n) - 5.970$

Figura 1: Comparação dos tempos de busca em µs.

Para o algoritmo de busca em intervalos, pode-se observar que ambas as estruturas possuem um comportamento logarítmico no tempo com o aumento da quantidade de elementos inseridos, mas a árvore rubro-negra implementada (vermelho) obteve tempos menores do que o *std::multimap*. Isso se deve ao fato de que a STL é mais genérica e vale ressaltar que ambas estruturas apresentam *outliers* em relação à curva logarítmica esperada nos mesmos intervalos de quantidade de elementos.



(a) $t(n) = 0.063 \cdot ln(n) - 0.179$



(b) $t(n) = 0.044 \cdot ln(n) - 0.286$

Figura 2: Comparação dos tempos de inserção em µs.

Para o algoritmo de inserção de dados, pode-se observar que ambas as estruturas possuem comportamentos logarítmicos no tempo com o aumento da quantidade de elementos inseridos, e mais uma vez a árvore rubro-negra implementada (vermelho) obteve tempos menores que o *std::multimap* pelos mesmos motivos já constados acima. Ademais, ainda obtemos *outliers* em relação ao esperado nos mesmos intervalos em ambas as estruturas.

Nuvem de pontos do teste *VoidSphereSelection*.

Por fim, a figura abaixo mostra a nuvem de pontos obtida no arquivo *torusALU.asc*, que é a mesma obtida no *torusMMP.asc*, a qual utiliza a implementação da STL para buscar os pontos dado um range de distância a um ponto pré definido.

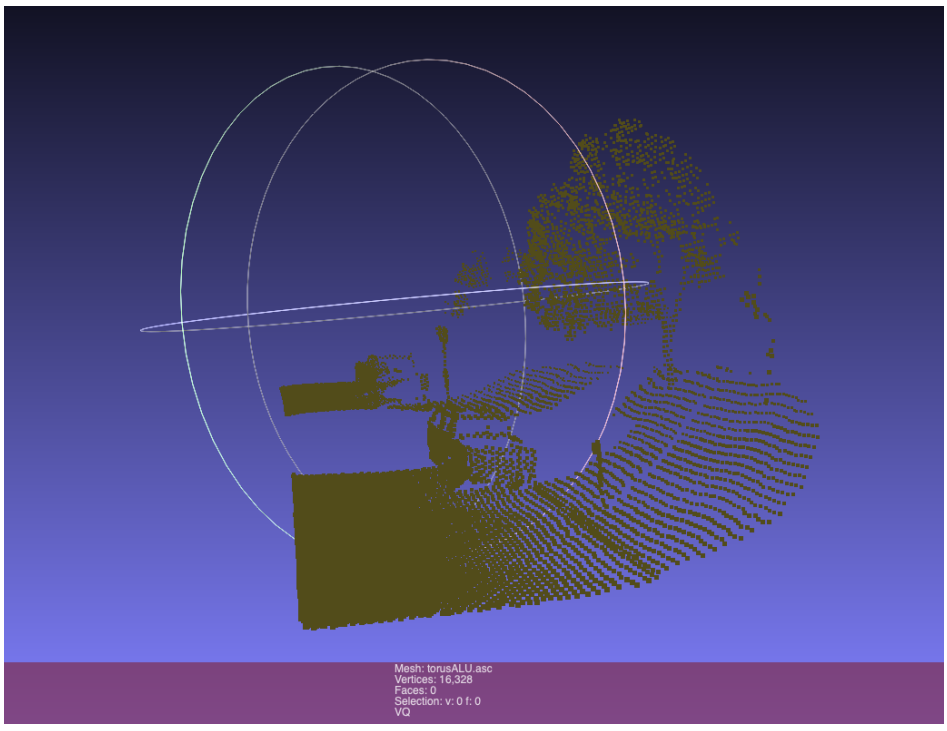


Figura 3: Nuvem de pontos *torusALU.asc*