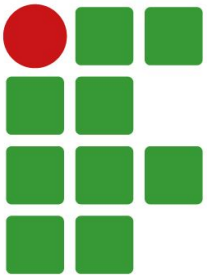

INSTITUTO FEDERAL
São Paulo

Beatriz Araujo Pinto Biagini
Diego Miguel Arruda de Moura
Eloisa Silva de Morais
Gabriel Augusto Franco Zara

Comparação de desempenho
Árvore AVL x Árvore Rubro Negra

Guarulhos 2020

SUMÁRIO

RESUMO

Este projeto tem por objetivo, a elaboração de um programa utilizando a linguagem de programação C, para realizar a experimentação de preenchimento de duas árvores por completo, bem como a medição do tempo gasto para tal preenchimento, sendo realizado uma bateria de preenchimentos em dois estilos de árvores lecionadas e vistas na disciplina: árvore do estilo AVL, e outra no estilo Rubro-negra, que neste caso utilizou-se a caída para a esquerda ou LLRB. Para isto, foi utilizado uma massa de dados disponibilizada para este fim, no formato CSV, contendo todos os dados pré definidos, para a coleta e posterior alimentação dos nós de cada árvore. Além disso, as inserções dos dados em cada árvore se dará por meio de duas vertentes, na qual uma é pautada em apenas na coleta pura do arquivo original da massa de dados, e outra com os dados previamente ordenados em outro arquivo em formato CSV, utilizando um algoritmo de ordenação também visualizado na disciplina, com o propósito de alcançar uma assertividade e melhor efeito comparativo do desempenho temporal de inserções em cada árvore mediante dois cenários distintos: dados desordenados, e ordenados.

Portanto, este é um projeto implementativo de caráter experimental e de base de conhecimento para os integrantes do grupo, como parcela obrigatória para obtenção de insumos avaliativos finais para a disciplina de Estrutura de Dados II, solicitadas pelo professor responsável da matéria.

1 - Desenvolvimento

1.1 - Etapa 1

Em primeiro espectro, decidiu-se realizar uma separação de arquivos para melhor organização de diretórios e subdiretórios, bem como para a própria organização do código como um todo, visto que isto rebate em termos de nota final do projeto. Desta maneira, como de costume, o projeto foi organizado por meio de abertura de projeto na IDE, e dividido em duas pastas: *sources* e *headers*. Na pasta *sources*, localiza-se o arquivo main.c onde é escrito todo o código fonte principal do programa, bem como as chamadas das funções e apontamento para demais arquivos necessários que compõem o projeto. Além disso, estão presentes os arquivos '*arvoreAVL.c*', '*arvoreLLRB.c*' que dispõem toda a implementação de código bem como suas funções, para tratativas de cada árvore, além do '*RadixSort.c'* que é referido o algoritmo de ordenação escolhido pelo grupo, no qual será pautado em breve neste descritivo. Já na pasta *headers*, localiza-se os arquivos referenciados como bibliotecas, que compõem as assinaturas das funções que irão manipular os dados nos módulos "*.c*".
Ademais, foi escolhido o algoritmo de ordenação RadixSort. Tal escolha foi pautada pela obtenção de resultados em testes realizados no projeto anterior desta disciplina, no qual seu objetivo era a medição temporal de 11 algoritmos de ordenação, para posterior verificação de eficiência em termos temporais de ordenação de dados. Desta maneira, com os resultados obtidos pelo grupo na execução deste projeto anteriormente, foi realizada uma análise mediante rendimento dos algoritmos por código e gráficos elaborados, e obtivemos um melhor resultado com o RadixSort, que nos resultou em sua escolha como protagonista nesta função de ordenação neste projeto.

Dificuldades desta etapa: Neste tópico, o desenvolvimento se deu de maneira mais calma, visto que a implementação base do projeto seguia os mesmos padrões de implementação transcorrido nas aulas cujos temas centrais eram as árvores AVL e Rubro negra.

Entendimentos desta etapa: Obteve-se uma maior compreensão de como se dá o desdobramento da estrutura das árvores, visto que já havíamos visto o código e realizado sua pré implementação em atividades anteriormente solicitadas.

.

1.2 - Etapa 2

Nesta etapa, se deu a leitura do arquivo disponibilizado (*'massaDados.csv'*) para fins de obtenção de dados, previamente mencionados. Para atingir este objetivo, teve que adequar o escopo de algumas funções (principalmente a de inserção de dados nas árvores), para que o escopo e assinatura das funções pudessem conversar de maneira conexa e fluída, se adaptando aos dados do arquivo de origem, novo tipo de estrutura a ser utilizado bem como cenário apresentado.

Dificuldades desta etapa: Apresentamos uma série de bloqueios para poder realizar a leitura do arquivo em formato csv, pois ainda não tínhamos tido contato com leituras neste formato, acostumados com o usual .txt, o que acabou levando um esforço maior em termos de morosidade do processo. Todavia, no fim das contas, ao verificar com um olhar mais analítico, percebeu-se que era um erro corriqueiro em relação ao ponteiro, e realizamos os devidos ajustes para dar prosseguimento no desenvolvimento do código.

Entendimentos desta etapa: Como resultado de aprendizado líquido, obtivemos resultados de conhecimento, em como extrair informações externas e realizar uma alocação pertinente no projeto em sua totalidade como padrões teóricos, bem como em performance e aptidão em eficiência no código, pois estimulou-nos a além de utilizar informações externas, também usufruir de outros conceitos e ferramentas apresentadas na disciplina ED1 e ED2, unindo os conhecimentos para suprir uma necessidade comum e ganhar efetividade final, resultando em presenciar novas experiências.

1.3 - Etapa 3

Nesta etapa se deu a inclusão dos arquivos referentes ao algoritmo de ordenação RadixSort no projeto, bem como a refatoração de código para uma adaptabilidade completa do escopo e assinatura de funções com as particularidades de implementação do Radix. Optamos pelo Radix após análise dos resultados do projeto anterior, levando em consideração o desempenho do algoritmo para grandes massas de dados. Além disso, após a implementação do algoritmo de ordenação realizamos a implementação da medição de tempo com base no código criado no projeto anterior, para assim fazer a medição dos tempos de execução nas duas árvores, ordenada e desordenadamente.

Dificuldades desta etapa: Fazer o radix funcionar bem, depois das alterações estarem funcionando passou a dar erro na inserção de dados mesmo sem termos mexido em nada. Continuamos rodando e novamente sem mexermos em nada novamente voltou a funcionar.

Entendimentos desta etapa:
Primeiramente, ficou nítido que erros de memória no C às vezes simplesmente existem, sem motivo ou razão aparentes.

1.4 - Etapa 4

Nesta etapa, decidimos a implementação de como iríamos utilizar as funções que manipulam e entregam os tempos de inserção em cada árvore. Para isso, resolveu-se reaproveitar o código utilizado no projeto anteriormente realizado, para garantir otimização de tempo no desenvolvimento do projeto como um todo.

Dificuldades desta etapa: A única dificuldade que obtivemos, foi ao realizar o printf do tempo em tela, estava resultando em 0, sendo assim, tivemos que adaptar as casas decimais do divisor na função para obter uma eficácia na medição.

Entendimentos desta etapa: Obtivemos um insight em realizar uma conexão com aspectos teóricos da disciplina de Programação Orientada a Objetos, cujo seu principal mandamento é a reuso de código (implementado em classes e encapsulados, lógico), mas de qualquer maneira, o conceito tangencia ao que aplicamos neste projeto.

Gráfico Comparativo - AVL X Rubro Negra

O gráfico acima, possui as medições temporais realizadas para cada bateria de inserção de cada árvore utilizada, com suas vertentes de ordenação (utilizando o Radix Sort) e desordenação mediante massa de dados previamente utilizada. Com efeito, nota-se uma curiosidade em relação aos desempenhos, pois a árvore AVL ordenada e desordenada geram um espaço temporal muito próximos, evidenciando um deslocamento temporal de 0,00001ms em relação uma à outra. Entretanto, ao analisar a rubro-negra Ordenada concomitantemente, nota-se também que o desempenho se iguala em termos quantitativos ao desempenho da AVL ordenada, o que nos supõe concluir que a ordenação faz uma mínima diferença quando aplicada em alguma implementação de código. E o gráfico nos resulta também que, a árvore mais custosa em relação a gasto de tempo é a rubro negra desordenada, gerando uma diferença de 0,020000ms aproximadamente em relação ao desempenho temporal das demais árvores, o que nos pressupõe que, pela rubro negra ter algumas particularidades, como verificações de coloração por exemplo, rebate em tempo gasto para operacionalizar todas as funções, pois vale ressaltar que mesmo na inserção, a rubro negra realiza algumas verificações de coloração para poder realizar os deslocamentos e posições necessários de cada nó na árvore, podendo

sim levar um tempo a mais para realizar tal tarefa, ainda mais por essa massa de dados que foi inserida, estar desordenada, gerando mais caminhos de ida e volta para validações de operações específicas de características próprias desse segmento de árvores.

Sendo assim, portanto, tivemos um empate em saber qual algoritmo gera um melhor desempenho em relação a custo de tempo, entre a AVL ordenada e a rubro negra ordenada, garantindo as duas uma hegemonia e igualdade de valores, podendo assim dizer, como este projeto tem por seu objetivo implícito um caráter experimental, que as árvores AVL e rubro negra ordenadas, possuem um bom desempenho para operações de inserção e manipulação desses dados.