SSC0951 **Desenvolvimento de Código Otimizado**

Atividade 3 **Utilizando Ferramenta de Profiling**

Nome: **Gabriel Van Loon Bodê da Costa Dourado Fuentes Rojas**

Alberto Campos Neves

---

1. Introdução

Neste relatório iremos utilizar a Ferramenta de Profiling Gprof para analisar um programa escrito em C que se trata de um trabalho feito por um dos integrantes no 2º Semestre de graduação na disciplina de Algoritmos e Estruturas de Dados.

O objetivo do relatório é o de encontrar e analisar os *hotspots* utilizando a visualização em formato de grafo e discorrer se os resultados obtidos são condizentes com os esperados.

2. Resumo do Programa em Análise

O programa utilizado neste trabalho tinha o objetivo de comparar o tempo de inserção, busca e eliminação de elementos em diferentes estruturas: Busca Binária, Lista Ordenada, Lista Ordenada com Sentinela, Árvore Binária de Busca, Árvore AVL e Lista Circular Dinâmica Duplamente Encadeada organizada pela frequência de Busca.

Em resumo, o programa realizava a inserção de N elementos de maneira ordenada e aleatória com um número determinado de repetições e então realizava a média dos resultados a fim de exibi-los de forma adequada em um .txt.

A execução completa do programa tomava em torno de 1 à 2 horas e portanto executamos uma versão resumida apenas com os casos de N=100 e N=1000 elementos.

ps: estamos fazendo profiling de um trabalho de um profiling. seria isso um inception?

3. Resultados Obtidos do Programa

Primeiramente, vamos ver abaixo o output produzido pelo programa que está sendo analisado, uma vez que a meta dele também é a de averiguar a eficiência das funções em cada uma das estruturas.

Como os resultados são extensos, enviamos junto um arquivo *`resultados.txt`* contendo a saída bruta produzida pelo programa.

Podemos então nos atentar à alguns pontos chaves (focando nos resultados em que N=1000 uma vez que nessa coluna as diferenças de tempo são mais acentuadas):

- Analisando a inserção vemos que:
    - de 2 em 3 casos a Árvore de Busca Binário possui uma performance inferior se comparada às outras estruturas.
    - a Lista Ordenada se mostrou um pouco mais cara que as outras se tratando de inputs ordenados em ordem decrescente e também aleatória.
- Analisando a remoção:
    - novamente vemos que a Árvore de Busca Binária possui uma performance ruim nos inputs ordenados.
    - no quesito de inputs decrescentes, vemos que grande parte das tads tem uma performance bastante próxima.
- Analisando a busca:
    - novamente a Árvore de Busca Binária possui performance ruim em inputs ordenados.
    - nos casos ordenados, com exceção da AVL e do vetor com Busca Binária, todas as TAD's tiveram uma performance próxima.
    - no caso aleatório, vemos que além da AVL, a Árvore de Busca binária também teve uma performance interessante.

Com os pontos acima levantados, vamos agora analisar o nosso programa utilizando a ferramenta Gprof e verificar se o comportamento visto acima é validado pelos resultados do grafo de tempo.

4. Resultados Obtidos pelo Gprof

Utilizando a ferramenta Gprof e algumas outras bibliotecas de python, obtivemos o seguinte grafo que mapeia o tempo gasto por cada uma das funções executadas no nosso programa (para facilitar a visualização, a imagem também se encontra em anexo com o relatório).

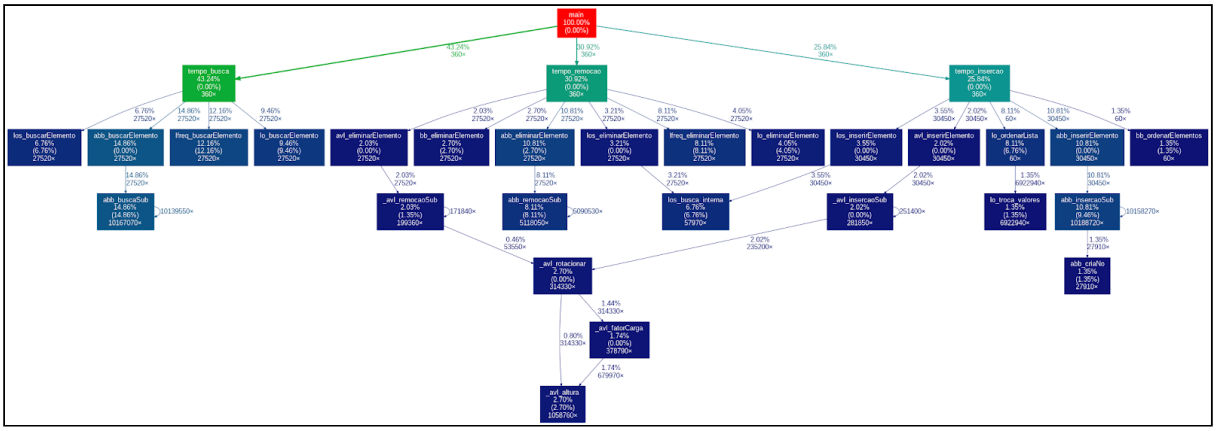

Figura 1: Grafo de consumo de tempo por função durante a execução do programa.

Primeiramente, vemos que o Gprof consegue lidar com bastante cuidado com ponteiros de funções, e isso é bastante interessante pois conseguiu agrupar as operações de maneira ideal. Vemos que no geral, as operações de buscas são as operações mais caras. Mas vamos analisar agora as operações separadamente para cada uma das estruturas.

Começando com a inserção, vemos que a ABB (Árvore de Busca Binária) realmente consome a maior porcentagem de tempo do programa **(10,81%)**, e isso se deve em grande parte devido à sua chamada recursiva.

Vemos também que a inserção da lista ordenada mal aparece, uma vez que o custo de inserir um elemento em uma lista é muito barato, no entanto o custo de ordenar uma lista ordenada pós inserção se mostrou alto, ocupando o segundo pior lugar nos casos de inserção **(8,11%)**. É importante pontuar que a TAD LO foi feita para ordenar os elementos ao fim de todas as inserções, enquanto a LOS (Lista Ordenada com Sentinela) foi feita para já realizar a inserção do elemento em sua posição ordenada e por isso faz a chamada da função '*busca_interna*'.

Quanto às remoções, Novamente a ABB foi responsável pelos piores desempenhos, ocupando cerca de **(10,81%)** enquanto a segunda pior performance neste caso ficou com a Lista de Frequência **(8,11%)** e isso é bastante interessante, uma vez que analisando novamente os resultados do programa vemos que mesmo que ela não tenha tido resultados muito diferentes, foi a única que manteve o mesmo consumo de tempo nos 3 cenários de remoção.

Por fim, quanto às buscas, vemos que a Busca Binária e a AVL são tão rápidas para executar estas operações que nem aparecem no grafo. Por outro lado, as outras TAD's

apresentaram valores não tão próximos quanto o levantado anteriormente, mas isso provavelmente se deve ao fato dos resultados nos 3 cenários de inserção terem sidos somados. Vemos, contudo, que conforme havíamos previsto a ABB foi de fato a estrutura com pior performance para busca **(14,86%)**, e isso se deve claramente às inserções não ordenadas que degeneram a árvore em uma lista.

5. Conclusões

Vimos que a utilização de ferramentas de profiling são interessantes pois ajudam a levantar outras perspectivas acerca do funcionamento do projeto, principalmente quando auxiliadas por outros mecanismos de visualização.

Os resultados obtidos acerca da ferramenta foram condizentes com a análise levantada pelo programa analisado e ainda ajudou a levantar outros pontos que ao se analisar apenas os valores não ficariam tão evidentes.