Relatório de Computação II
Binary Search Tree Symbol Table Caching

Bruno Daiki Yamada

19 de abril de 2016

Data do procedimento: 19 de abril de 2016
Professor: Yoshiharu Kohayakawa
Monitor: Gabriel Guilhoto

1 Objetivos

Implementar e verificar os efeitos do caching no algoritmo de *Binary Search Tree* ao executar um software de contagem de palavras em um aquivo `txt`.

2 Procedimento

No caso, foi adicionado na classe BST uma nova private, `Node Cache`, que armazena a última node acessada pelas funções `get()`e `put()`. Então, antes de executar a busca na árvore, nas funções `get()` e `put()`, o programa verifica se a `key` do `Cache` é igual ao fornecido, evitando a busca se é possível, e assim, possivelmente reduzindo o tempo de execução do programa `MyFrequencyCounterCache`, que executa funções de `get`e `put`, repetitivamente para o mesmo `key`.

3 Resultados

O tempo de execução de cada algoritmo para o arquivo `Leizpig1m.txt`:

Algoritmo	Dado 1	Dado 2	Dado3	Média dos Dados
Sem caching	124,823	110,762	106,491	114,025
Com caching	155,574	140,482	160,799	152,285

O tempo de execução de cada algoritmo para o arquivo `Leizpig100k.txt`:

Algoritmo	Dado 1	Dado 2	Dado3	Média dos Dados
Sem caching	13,7	13,602	14,562	13,954
Com caching	11,015	10,835	11,052	10,967

4 Discussão

Através dos dados obtidos, é possível concluir que para arquivos grandes *caching* de fato diminui consideravelmente o tempo de execução. Reduzindo a média de execução em cerca de 38 segundos. Para arquivos menores O tempo ganho na execução foi bem menor, por vezes menor do que a flutuação experimental causado pela performance do computador.

5 Conclusão

Atravé dos testes executados, é possível concluir que o uso de caching pode diminuir o tempo de execução para testes com alto tempo de execução. Entretanto para arquivos menores, o ganho com o *caching* é bem menor, surtindo pouco efeito.