Exercício 3.3.30 - Software Caching (Algs 4)
Nome: Gabriel de Russo e Carmo
N USP: 9298041
Data: 17/04/2016

Relatório sobre o exercício

Esse exercício teve como intuito a comparação de duas árvores rubros-negras esquerdistas implementadas de maneira muito semelhante. A diferença da primeira para a segunda é um nó de *caching*, que guarda o último nó que foi acessado em alguma operação de *put* ou *get*. Esse nó, chamado de *cache*, permite que algumas operações sejam executadas em tempo constante.

No caso, testei as implementações num programa de contagem de palavras. Em detalhes, o programa conta a quantidade de palavras (únicas e distintas) com tamanho maior ou igual a *k* e diz qual a palavra mais frequente respeitando a mesma restrição. Para textos pequenos, a diferença de desempenho é muito pequena, inviabilizando qualquer veredito. Para textos grandes, temos resultados mais interessantes. Segue uma tabela de desempenho em quatro textos, comparando os tempos médios aproximados de cada versão em 5 execuções para k = 3, k = 7 e k = 10, respectivamente.

	Sem caching (k = 3/7/10)	Com caching (k = 3/7/10)
leipzig1M.txt	41s/24s/14s	26s/19s/13s
leipzig1Msorted.txt	39s/24s/14s	27s/19s/13s
dicionario_repeticao.txt	17s/15s/12s	10s/10s/10s
bible.txt	2s/2s/1s	1s/1s/1s

O arquivo leipzig1M.txt é um arquivo de um jornal alemão com 1 milhão de linhas.
O arquivo leipzig1Msorted.txt é semelhante ao arquivo anterior, mas com as linhas ordenadas.
O arquivo dicionario_repeticao.txt é um dicionário da língua portuguesa multiplicado 500 vezes (foi reordenado).
O arquivo bible.txt é a bíblia cristã em inglês.

Nota-se que a versão com caching tem desempenho superior para textos grandes e valores pequenos de *k.* Para maiores valores de *k*, o desempenho ainda é levemente superior, mas devido a menor quantidade de palavras grandes num texto, o número de palavras processadas é reduzido e consequentemente o número de operações nas árvores também.

***P**: Em comparação com a versão original no site do livro (que usa a tabela de símbolos ST.java), a sua versão é mais eficiente? Caso sim, como você explica essa diferença? Como é implementada a tabela de símbolos ST.java?*

R: Minha versão é mais eficiente do que a do livro que usa a tabela de símbolos *ST.java* pois essa tabela instancia um *TreepMap* do Java, que nada mais é do que uma árvore rubro-negra sem caching (não esquerdista). Desse modo, seu desempenho é muito semelhante a versão sem caching esquerdista testada na comparação. Na verdade, mostrou-se um pouco mais eficiente do que a última, provavelmente pelo fato de não ser esquerdista.

***P**: A versão com caching é mais rápida que a versão sem caching? Se sim, como você explica essa diferença?*

R: Sim. Como vimos nos resultados experimentais, essa diferença é muito mais visível para textos grandes. Essa diferença se dá pelo fato já mencionado no começo deste relatório: algumas operações são executadas em tempo constante. Como as palavras se repetem muito em textos, economizar operações de *put* e *get* pode fazer uma boa diferença, uma vez que ambas tem tempo proporcional a altura da árvore. Entretanto, note que esta diferença não se mostrou maior que o dobro, já que nossa árvore é aproximadamente balanceada e logo tem altura relativamente pequena, proporcional ao logaritmo da quantidade de palavras que contém. Para valores grandes de *k*, veja que as palavras processadas tem muita chance de se repetir, já que existem poucas palavras grandes. No exemplo do jornal *Leipzig,* a maioria das palavras para *k = 10* é *government*.