Universidade de São Paulo

EP2 - Geo IP Lookup

São Paulo
2015

Objetivo

Semelhante a um exercicio já realizado (Creative exercise 4.4.8 de IntroCS), um arquivo deve ser recebido (dbip-city-2015-05h.csv, dado no enunciado, que pode ser encontrado em: https://db-ip.com/db/), e dependendo do número de argumentos passados na linha de comando, realizar uma das operações:

- **1 Argumento**: O programa deve receber uma lista de IPs da entrada padrão, e de acorco com as informações do arquivo, imprimir o endereço correspondente ao IP, no formato "**cidade, estado, país**". Por exemplo, ao receber o IP **143.107.45.22**, a saida será **Miami, Mato Grosso Do Sul, BR**.

- **2 Argumentos ou mais**: O programa deve receber um arquivo *log*, extraído do arquivo system.log, que mostra as tentativas de acesso ao computador, por *ssh*. Depois, usar o arquivo *dbip-city-2015-05h.csv* para localizar a origem da tentativa de acesso remoto. A saída deve conter cada local uma única vez, e em ordem decrescente em relação ao número de tentativas de acesso.

A solução deve ser implementada usando uma ST, uma BST e uma RedBlackBST.

Métodos

Como a implementação é semelhante nos 3 casos (ST, BST e RedBlackBST), os métodos serão representados considerando a implementação em uma ST (por exemplo, na implementação com uma RedBlackBST, é usada uma RedBlackBST ao invés de uma ST). Além disso, apenas o código de IPLookUpDeLuxeST.java está comentado, já que as outras implementações possuem códigos semelhantes.

CreateCity

Cria uma tabela de símbolos das cidades. Ela recebe uma ST<Long, String>, e para cada linha em *dbip-city-2015-05h.csv*, cria uma ocorrência.

CreateLog

Recebe uma ST completa com os endereços, e cria uma outra ST <Integer, String> com o número de vezes que cada cidade foi encontrada, para todos os IPs do arquivo log. Como é pedido a impressão ordenada decrescente do número de acessos e sua respectiva localização, e algumas vezes o número de ocorrências se repete para cidades diferentes (por exemplo, 2 cidades foram encontradas 1 única vez), foi criada uma nova ST <Integer, Queue<String>>, para que possamos imprimir todas as cidades.

SetLongIPValue

Dado um IP, esse método retorna a representação em Long do valor do IP.

Problemas na implementação da BST

Na hora da criação da BST, o fato do arquivo *dbip-city-2015-05h.csv* estar ordenado de forma crescente e a BST não ser balanceada, gera o pior caso de uma BST: a árvore se torna uma lista simples ligada, e isso faz com que a pilha do sistema estoure, para a implementação recursiva, e que não tenha a memória necessária para armazenar uma quantidade tão grande de dados (4920676 linhas). Por isso, foi usado o comando shuf, no próprio shell, para embaralhar o arquivo. O comando foi:

shuf -o *dbip-city-2015-05h.csv* < *dbip-city-2015-05h.csv*

Com o arquivo embaralhado, temos o caso da inserção aleatória, possibilitando a leitura e processamento do arquivo.

Simulações

Para a analise do tempo do algoritmo, foi usado o comando **-Xmx3G**, para que a memória usada no processamento dos dados não atrapalhasse na visualização.

Como a classe ST.java usa uma TreeMap, que no caso, usa uma Árvore Rubro-Negra, e RedBlackBST.java implementa também uma Árvore Rubro-Negra, esperamos que os tempos de execução sejam semelhantes. Além disso, por ser balanceada, os comandos de inserção e busca tem tempo proporcional a lg n, enquanto uma BST tem tempo proporcional a log n. Ambas possuem bom desempenho, porém existem casos em que a criação da BST gera uma lista, como o caso explicado anteriormente. As simulações foram geradas com o arquivo embaralhado. Segue abaixo algumas execuções das soluções:

	ST.java	BST.java	RedBlackBST.java
200 mil entradas 1 argumento	real 0m59.592s user 1m34.172s sys 0m2.396s	real 1m0.432s user 1m36.192s sys 0m2.584s	real 0m58.259s user 1m32.632s sys 0m2.356s

200 mil entradas 2 argumentos	real 0m46.908s user 1m32.816s sys 0m1.648s	real 0m47.282s user 1m34.008s sys 0m1.536s	real 0m49.707s user 1m32.724s sys 0m1.512s
200 mil entradas 2 argumentos	real 0m42.985s user 1m23.640s sys 0m1.372s	real 0m50.710s user 1m36.956s sys 0m1.552s	real 0m49.008s user 1m28.308s sys 0m1.552s

Vimos que, como esperado, o desempenho de RedBlackBST e ST são superiores a BST, além de serem sempre balanceadas.