Algoritmo de Simulação de Book de Ofertas de Ações

Cassiano Luis Flores Michel, Leonardo Vargas Schilling
Ciência da Computação — PUCRS
Algoritmos e Estruturas de Dados 2 - Marcelo Cohen

17 de setembro de 2021

PUCRS

Introdução

Neste artigo busca-se descrever o processo de criação de um algoritmo para leitura de um arquivo de texto, onde simula-se o recebimento de uma ordem de compra ou venda de ações a cada linha.

A cada linha lida, devem-se realizar todas as operações possíveis até aquele momento, através das seguintes regras:

1. A empresa não realiza preguízo (Preço de compra > Preço de venda);
2. A empresa prioriza a maior diferença entre Preço de Compra e Venda;
3. A empresa lucra com a diferença entre os valores de compra e venda;
4. A primeira linha do arquivo indica o número de operações.

Formato da linha do arquivo:

<operação> <quantidade> <preço>

Exemplo:

C 129 14 V 302 12

Estratégia

Entende-se que para atingir o seu objetivo, as ordens de **compra** devem ser ordenadas por preço *decrescente*, enquanto as ordens de **venda** devem ser ordenadas por preço *crescente*.

Desta forma a operação com maior lucro para a empresa sempre será através da execução da primeira ordem de venda e da primeira ordem de compra da coleção.

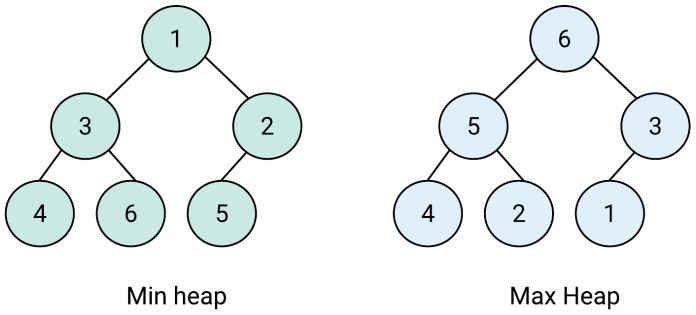
Para cada linha lida, insere a operação na coleção e executa as operações possíveis, por fim, imprime os resultados analisados:

1. Lucro;
2. Tempo de execução aproximado;
3. Total de operações aproximado;
4. Ordens de compra não executadas;
5. Ordens de venda não executadas;
6. Nome do arquivo.

Estruturas de dados

Optou-se por utilizar implementações *opensource* de filas de prioridade através de heaps binários. Um heap binário é uma estrutura de dados baseada em árvores binárias que satisfaz sempre a propriedade de heap, que diz que se o nó B é filho do nó A, então a chave de A é maior ou igual a chave de B.

Demonstração de um Min Heap e Max Heap



Fonte: https://medium.com/@rohan04/easy-way-to-implement-the-heap-data-structure-in-python-5658d0a6d266

Algoritmo

A complexidade de tempo para a inserção (executada N vezes, sendo N o número de linhas do arquivo – 1) é O(log n);

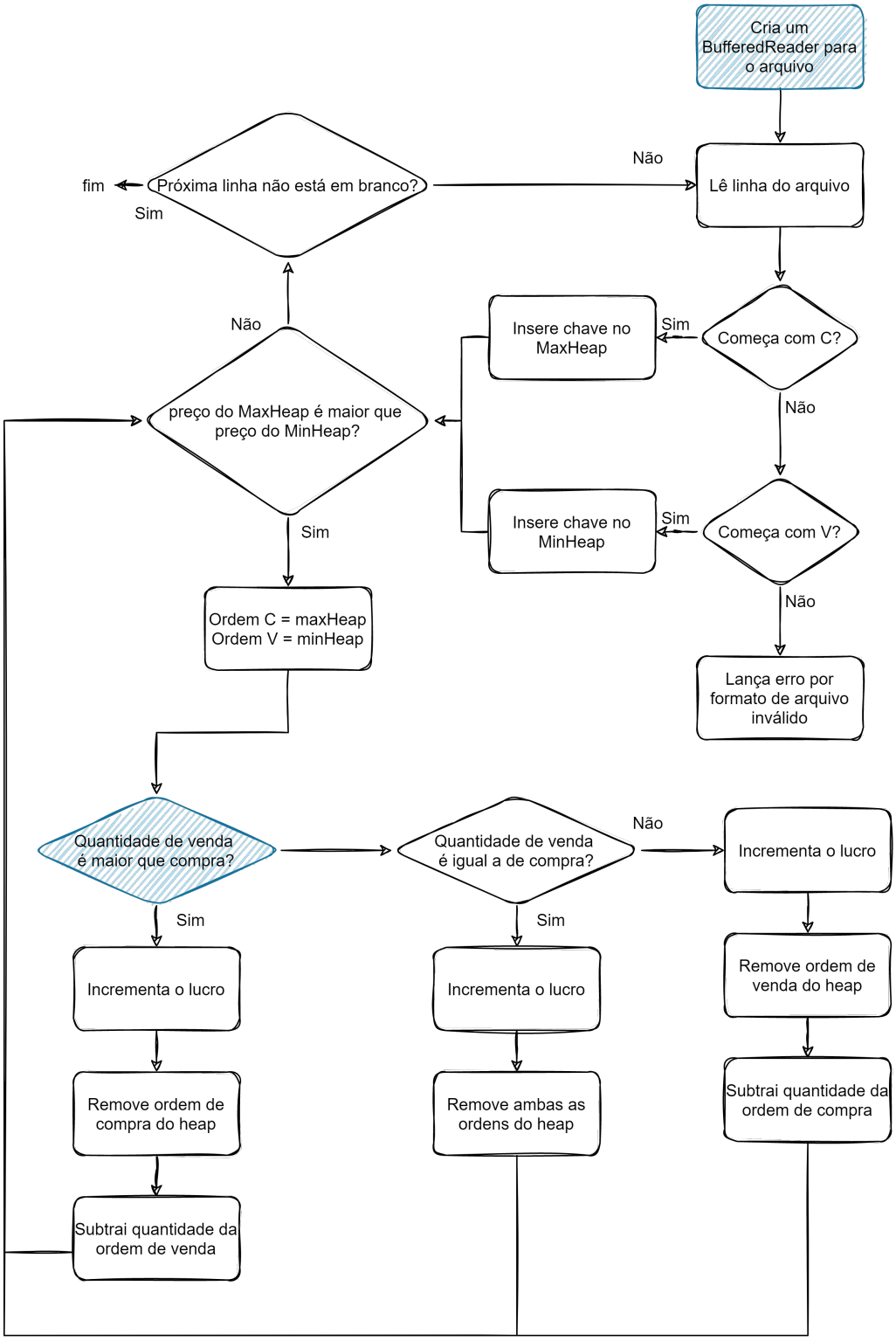
O acesso ao valor máximo ou mínimo dos heaps (valor que será checado para validar se existem operações lucrativas a serem executadas) é O(1);

Caso haja uma operação lucrativa a ser executada (se o preço da raiz do minheap for menor ou igual ao preço da raiz do maxheap), executam-se as ordens possíveis;

Ao executar determinada ordem, caso a quantidade de compra ou venda das ordens prioritárias vire zero, então a ordem é removida (através do método delMin ou delMax) com complexidade O(log n);

Serão executadas N inserções (em O(log n)), e até N remoções (considerando um cenário onde uma ordem de compra executa todas as ordens de venda ou vice-versa) portanto, pode-se considerar a complexidade do algoritmo como O(n log n).

Fluxograma


Cria um BufferedReader para o arquivo
Lê linha do arquivo
Não
Próxima linha não está em branco?
fim
Sim
Começa com C?
Sim
Insere chave no MaxHeap
Não
preço do MaxHeap é maior que preço do MinHeap?
Não
Começa com V?
Sim
Insere chave no MinHeap
Não
Sim
Ordem C = maxHeap
Ordem V = minHeap
Lança erro por formato de arquivo inválido
Quantidade de venda é maior que compra?
Quantidade de venda é igual a de compra?
Não
Incrementa o lucro
Sim
Sim
Incrementa o lucro
Incrementa o lucro
Remove ordem de venda do heap
Remove ordem de compra do heap
Remove ambas as ordens do heap
Subtrai quantidade da ordem de compra
Subtrai quantidade da ordem de venda

Resultados

Após a execução do algoritmo com todos os casos de teste, os dados foram tratados no Excel, para análise do tempo de operação e número de operações.

A contagem de operações se deu através da criação de uma classe **Contador**, cujo atributo *count* foi incrementado em todos os trechos e iterações do fluxo *(inserção, remoção, sink, swim, assertHeap)*.

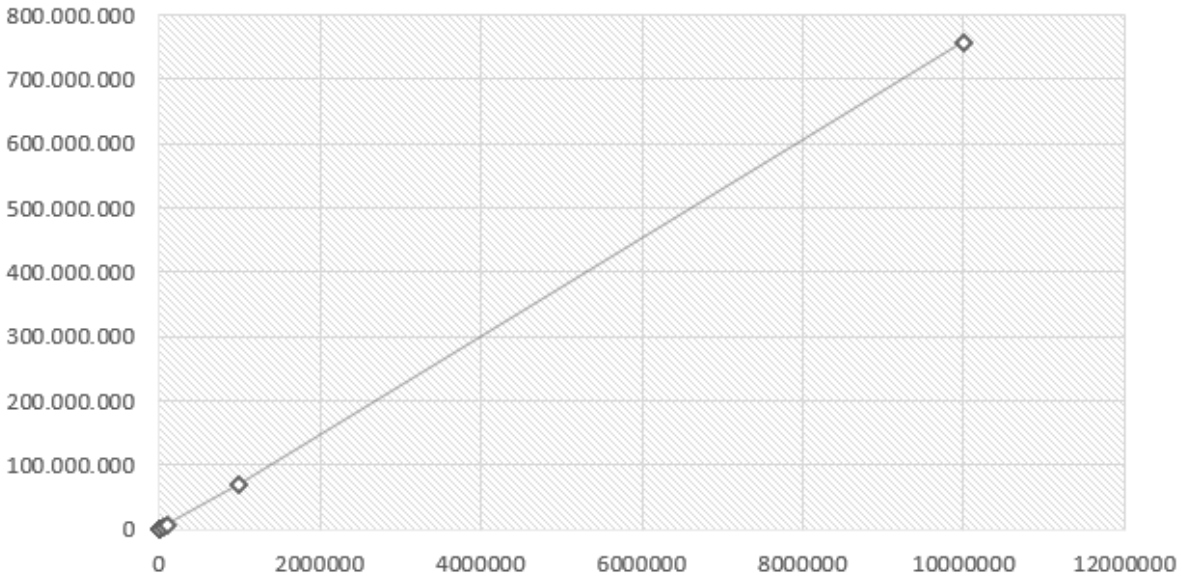
Tabela de Resultados

Nome	Ordens	Operações	Tempo Exec.	Lucro	Compras Restantes	Vendas Restantes
trinta_enunciado.txt	30	755	0,0320	45.538	3	7
trinta.txt	30	686	0,0000	523.013	8	10
cem.txt	100	2.913	0,0010	1.700.632	13	35
trezentos.txt	300	10.727	0,0040	5.296.100	63	67
mil.txt	1000	44.475	0,0080	17.829.179	217	171
tres_mil.txt	3000	150.951	0,0090	52.010.348	543	568
cinco_mil.txt	5000	264.297	0,0220	85.159.113	1.001	917
dez_mil.txt	10000	574.020	0,0230	171.823.410	1.865	1.845
cinquenta_mil.txt	50000	3.322.978	0,1119	856.631.193	9.143	9.259
cem_mil.txt	100000	6.875.371	0,1789	1.716.206.656	18.706	18.462
milhao.txt	1000000	70.837.030	1,1933	17.158.118.767	186.157	187.116
dez_milhoes.txt	10000000	757.241.584	9,5895	171.754.048.271	1.863.975	1.862.567

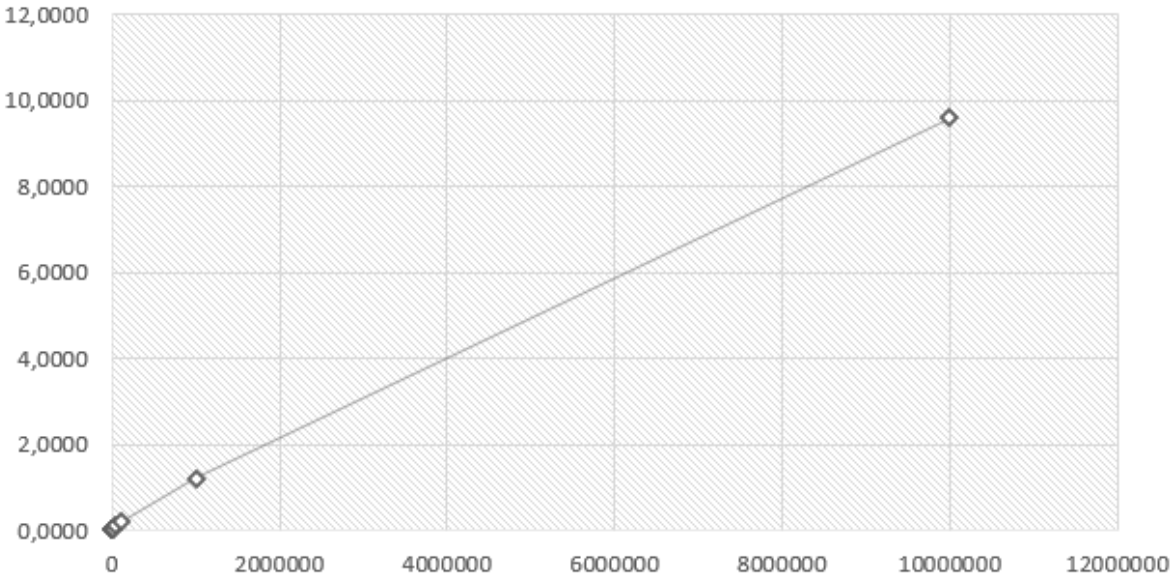
Fonte: os autores

Com os dados de tempo de execução, número de operações e número de ordens (linhas do arquivo de texto), foi possível esboçar dois gráficos:

Número de Operações x Número de Linhas do Arquivo



Tempo de Execução (s) x Número de Linhas do Arquivo



Referências

[1] Sedgewick, Robert; Wayne, Kevin: "**Priority Queues**". Last modified on December 29, 2017. Available in: https://algs4.cs.princeton.edu/24pq

[2] Rohan: "**Easy way to implement the Heap data-structure in Python.**". Last modified on July 17, 2020. Available in: https://medium.com/@rohan04/easy-way-to-implement-the-heap-data-structure-in-python-5658d0a6d266