Estruturas de Dados Retroativas: um Mapeamento Sistemático

José Wagner de A. Júnior
Inst. de Engenharia de Sistemas e Tecnologia da Computação
Universidade Federal de Itajubá
Av. BPS, 1303 – Caixa Postal 50
Itajubá – Minas Gerais – Brasil
junior.andrade.cco@gmail.com

Rodrigo Duarte Seabra
Inst. de Matemática e Computação
Universidade Federal de Itajubá
Av. BPS, 1303 – Caixa Postal 50
Itajubá – Minas Gerais – Brasil
rodrigo@unifei.edu.br

Adler Diniz de Souza
Inst. de Matemática e Computação
Universidade Federal de Itajubá
Av. BPS, 1303 – Caixa Postal 50
Itajubá – Minas Gerais – Brasil
adlerdiniz@unifei.edu.br

ABSTRACT

Given the miniaturization of electronic devices and the amount of data processed by them, the applications developed need to be efficient in terms of memory consumption and temporal complexity. Retroactive data structures are data structures in which it is possible to make a modification in the past and observe the effect of this modification on its timeline. These data structures are used in some geometric problems and in problems related with graphs, such as the minimum path problem in dynamic graphs. However, the implementation of these data structures in an optimized way is not trivial. In this scenario, this work presents the results of a research related to retroactive data structures, in order to compare the performance of the implementations proposed by several authors in relation to the trivial implementations of these data structures. The research method used was the study of the articles related to retroactive data structures, from a systematic mapping, and the performance analysis of these data structures coded in C++ language. The data structures identified in this study presented better results in terms of space consumption and processing time in relation to their implementations by brute force, but, in some cases, with high constants.

Keywords

Retroactivity; Retroactive Data Structures; Dynamization of Algorithms; Systematic Mapping.

RESUMO

Face à miniaturização dos aparelhos eletrônicos e devido à quantidade de dados processados por eles, as aplicações desenvolvidas necessitam ser eficientes em termos de consumo de memória e complexidade temporal. Estruturas de dados retroativas são estruturas em que é possível realizar uma modificação no passado e observar o efeito dessa modificação em sua linha temporal. Essas estruturas são utilizadas em alguns problemas geométricos e em outros relacionados a grafos, como o problema do caminho mínimo em grafos dinâmicos. Contudo, a implementação dessas estruturas de maneira otimizada nem sempre é trivial. Diante desse cenário, este trabalho apresenta os resultados de uma pesquisa relacionada a estruturas de dados retroativas, visando comparar o desempenho das implementações propostas por variados autores em relação às implementações triviais dessas estruturas. O método de pesquisa adotado foi o estudo dos artigos relacionados às estruturas de dados retroativas, a partir de um mapeamento sistemático, e a análise de desempenho dessas estruturas codificadas na linguagem C++. As estruturas identificadas nesse mapeamento apresentaram melhores resultados no que tange ao consumo de espaço e tempo de processamento com relação às suas implementações por força bruta, porém, em alguns casos, com constantes altas.

Palavras-Chave

Retroatividade; Estruturas de Dados Retroativas; Dinamização de Algoritmos; Mapeamento Sistemático.

CCS Concepts

• Theory of computation → Design and analysis of algorithms → Data structures design and analysis.

1. INTRODUÇÃO

As aplicações computacionais desenvolvidas atualmente precisam ser escaláveis e eficientes tanto em questão de espaço de armazenamento quanto de tempo. Entretanto, em geral, o espaço de armazenamento e o tempo são grandezas inversamente proporcionais, no sentido de que, dado um mesmo problema, soluções mais eficientes do ponto de vista temporal têm a tendência de serem mais custosas com relação à memória, e vice-versa. Porém, isso não é uma regra, existindo algumas exceções.

Por exemplo, o problema da soma dos elementos de um sub-vetor pode ser executado em tempo linear no número de elementos do sub-vetor sem utilização de espaço adicional, ou em tempo logarítmico no número de elementos do vetor utilizando uma árvore de segmentos com custo adicional em memória de $O(n \lg(n))$. Não obstante, esse problema também pode ser resolvido com um vetor extra de soma cumulativa, respondendo às consultas em tempo constante e utilizando apenas espaço adicional em memória de $O(n)$.

As estruturas de dados devem ser otimizadas para minimizar os impactos dos *trade-offs* entre memória e velocidade, gerando aplicações de qualidade. Existem algoritmos que são implementados sobre estruturas de dados como, por exemplo, algoritmos de busca em grafos (fila e pilha), algoritmos de árvore geradora mínima (lista ordenada e fila de prioridade) e algoritmos de caminho mínimo em grafos (fila de prioridade).

Dentre essas estruturas de dados, existem algumas que objetivam mostrar as diferentes versões dessas estruturas ao longo do tempo. Primeiramente, foram criadas as estruturas persistentes, que

consistem em estruturas nas quais todas as suas versões criadas no decorrer do tempo estão disponíveis para acesso e possível modificação [1, 2]. Como um exemplo da ideia de persistência, têm-se as ferramentas de controle de versão no desenvolvimento de *software*. Novas versões são criadas a partir de versões passadas, tornando-se disponíveis para a criação de outras versões. Nesse contexto, essa classe de estruturas de dados foi dividida em dois tipos: *parcialmente* e *totalmente* persistentes. Estruturas *parcialmente* persistentes permitem acesso a toda estrutura, porém, só permitem a criação de uma nova versão a partir da versão mais recente da estrutura [3]. Já estruturas *totalmente* persistentes permitem a criação de uma versão nova a partir de qualquer uma de suas versões anteriores [3]. Estruturas *parcialmente* retroativas podem ser vistas como um caminho contínuo, enquanto estruturas *totalmente* retroativas podem ser observadas como um grafo dirigido acíclico (DAG).

Outro tipo de estrutura de dados em que se mantêm as informações ao longo do tempo é chamada *retroatividade* [4]. Nessa classe de algoritmos, em vez da criação de novas versões a partir de versões anteriores, que são, de certa forma, disjuntas entre si, o objeto de estudo é a influência de uma alteração na estrutura com relação a uma modificação realizada no passado. Por exemplo, em uma fila de prioridade na qual se tem permissão para modificação em qualquer ponto do passado, e acesso apenas ao tempo presente, a adição de uma operação de inserção nessa estrutura pode causar um efeito cascata nas versões intermediárias da estrutura.

Na Figura 1, tem-se o exemplo do efeito cascata que acontece na estrutura ao adicionar-se uma operação de inserção na fila de prioridade. Pode-se observar que o tempo em que os elementos ficam na estrutura, bem como seu estado final, muda completamente. Em vermelho, pode-se notar o aumento da linha de permanência de cada elemento nessa fila de prioridade.

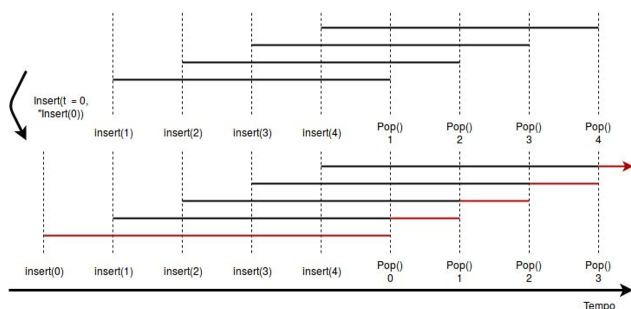

Figura 1. Exemplo do efeito cascata em uma fila de prioridade após a adição de uma inserção. Fonte: Os autores.

De maneira similar ao que acontece nas estruturas persistentes, em estruturas retroativas também há subdivisões relacionadas ao tipo de consulta que essas estruturas respondem. Existem três tipos de estruturas retroativas: *parcialmente*, *totalmente* retroativas [4] e estruturas com retroatividade *não-consistente* [5]. Em estruturas *parcialmente* retroativas, as atualizações podem ser realizadas em qualquer momento da estrutura, porém, as consultas são realizadas somente na versão mais recente. Já as estruturas *totalmente* retroativas permitem consultar as versões de todos os tempos da estrutura. Finalmente, estruturas de dados com retroatividade *não-consistente* permitem ao programador observar o primeiro ponto na linha temporal da estrutura em que uma operação de consulta foi afetada pela modificação realizada no passado.

Alguns desses efeitos cascata são difíceis de prever, e, em algumas vezes, estruturas de dados lentas e mal otimizadas são implementadas nas aplicações que necessitam desse tipo de solução. Por exemplo, uma solução direta para gerar a versão retroativa de uma estrutura de dados seria a manutenção das operações nela realizadas ordenadas por tempo e, quando uma consulta for realizada, executar as operações na estrutura de dados de maneira sequencial por força bruta. Entretanto, nessa solução é necessário percorrer todas as operações da estrutura, o que nem sempre é uma solução otimizada. Portanto, por meio do estudo em tela, almeja-se coletar informações sobre as complexidades estudadas para essas estruturas, bem como suas implementações a partir de uma revisão sistemática da literatura.

A seção 2 define o escopo da pesquisa e as questões de interesse do estudo. A seção 3 apresenta considerações sobre os estudos identificados na execução da busca sobre retroatividade. A seção 4 apresenta dados coletados nesses artigos referentes à complexidade temporal e espacial dessas estruturas. A seção 5 apresenta testes realizados sobre a implementação, na linguagem C++, do paradigma da retroatividade nas estruturas de dados pilha, fila e fila de prioridade. Finalmente, na seção 6, são tecidas as considerações finais do trabalho.

2. DEFINIÇÃO DA PESQUISA

O método de pesquisa selecionado para este trabalho foi o mapeamento sistemático. O objetivo consistiu em obter uma visão geral sobre estruturas de dados retroativas e fornecer uma implementação dessas estruturas com uma posterior análise relacionada aos seus desempenhos temporal e espacial. Esta pesquisa se justifica, pois explora a literatura em busca dessas estruturas, disponibilizando a visualização de seus desempenhos práticos. Ademais, também permite identificar lacunas nas quais essas estruturas de dados poderiam ser aplicadas, além dos usos já conhecidos como, por exemplo, no algoritmo de caminho mínimo proposto por Dijkstra [6]. Nesse algoritmo, filas de prioridade são comumente utilizadas e, portanto, uma implementação retroativa dessa estrutura permite a utilização do algoritmo de Dijkstra em grafos dinâmicos, nos quais arestas são removidas ou adicionadas após a execução completa do algoritmo para grafos estáticos.

2.1. Questões de Interesse

O objetivo desse estudo consiste em fornecer uma visão geral sobre estruturas de dados retroativas, comparando o desempenho dessas estruturas no que se refere às suas implementações por força-bruta. Foram definidas duas questões de interesse:

Q1: Os algoritmos e soluções propostos para as estruturas de dados estão próximos à sua solução ótima com relação à complexidade espacial e temporal no contexto da retroatividade?

A principal questão dessa pesquisa consiste em coletar dados na literatura sobre as estruturas de dados retroativas, comparando o desenvolvimento dessas estruturas do ponto de vista temporal e espacial.

Q2: Essas estruturas foram implementadas de modo que as complexidades temporal e espacial, aferidas por meios teóricos, possam ser testadas?

É importante a comparação com algum outro método de implementação das estruturas, pois algumas constantes, por vezes, são ignoradas na análise teórica de complexidade, podendo influenciar a complexidade prática dessas estruturas.

2.2. Execução da Pesquisa

Para delinear o escopo da pesquisa foram estabelecidos critérios para garantir, de forma equilibrada, a viabilidade de sua execução, o acesso aos dados e a abrangência do estudo. A pesquisa foi realizada a partir de bibliotecas digitais por meio dos seus respectivos engenhos de busca e, quando os dados não estavam disponíveis eletronicamente, com base em consultas manuais. A pesquisa foi executada utilizando três bases de dados: Scopus, IEEE e Compendex. Elas foram escolhidas por conterem o maior número de artigos relacionados à computação, além de serem as bases de dados mais utilizadas na área. Esse estudo compreende todos os documentos disponíveis nessas bases de dados.

A expressão de busca foi definida da seguinte maneira: inicialmente, uma busca manual nos primeiros artigos relacionados a estruturas de dados retroativas foi realizada para a extração de palavras-chave. Deste modo, foi gerada a seguinte expressão para busca nas bases de dados:

title-abs-key(("Retroactive data structures") OR ("Non-oblivious retroactive data structures") OR ("Fully retroactive data structures") OR ("Partial retroactive data structures"))

Após a definição da expressão de busca, foi realizada uma execução preliminar da pesquisa nas bases de dados supramencionadas. A busca com essa expressão retornou um total de 18 artigos (Scopus: 10 artigos, Compendex: sete artigos e IEEE: um artigo). Como nem todos os documentos estão alinhados com a linha de pesquisa desejada, esses artigos passaram por um processo de triagem para que somente aqueles relevantes ao estudo fossem utilizados. Os critérios de inclusão (CI) podem ser vistos na Tabela 1.

Tabela 2. Critérios de inclusão. Fonte: Os autores.

Identificador	Descrição
CI-01	Artigos que contenham no resumo especificações sobre estruturas de dados retroativas
CI-02	Artigos que contenham problemas que utilizem estruturas de dados retroativas em sua solução

Foram removidos artigos duplicados, além de outros que não possuíam relação direta com estruturas de dados retroativas. Também foram excluídos trabalhos que não estavam completamente disponíveis em bases de dados *online*, bem como artigos sem resumo. Os critérios de exclusão (CE) e seus identificadores podem ser vistos na Tabela 2.

Tabela 2. Critérios de exclusão. Fonte: Os autores.

Identificador	Descrição
CE-01	Serão excluídas duplicatas de artigos.
CE-02	Artigos que continham somente a palavra ”Retroatividade”em seu texto, porém, apenas com sentido do efeito cascata após uma alteração da estrutura no passado, sem relação com as estruturas de dados em si.

Para a remoção, foi realizada, inicialmente, a leitura do resumo e da introdução dos artigos. Dentre os aceitos, procedeu-se à leitura completa dos trabalhos. Dos 18 artigos inicialmente selecionados, seis foram excluídos pelo critério CE-01 e cinco pelo critério CE-02. Após essa triagem, foram retornados sete resultados considerados relevantes, como pode ser visto na Tabela 3.

3. CONSIDERAÇÕES SOBRE O ESTUDO

Demaine *et al.* [4] e Acar *et al.* [5] foram os primeiros a definirem as ideias sobre retroatividade e as possíveis maneiras de transformar uma estrutura comum em sua versão retroativa. Demaine *et al.* [4] propuseram transformar qualquer estrutura parcialmente retroativa em sua versão totalmente retroativa com um aumento multiplicativo de $\sqrt{m}$ na complexidade e na memória, em que m consiste no tamanho da linha temporal que a estrutura está contida. Essa pesquisa também mostra como obter versões retroativas de filas, pilhas e filas de prioridade (*min-heap*).

Tabela 3. Artigos encontrados. Fonte: Os autores.

Nº	Ano	Título da publicação	Autores	Veículo de Publicação
01	2019	A Retroactive Approach for Dynamic Shortest Path Problem	Sunita e Garg (2019)	National Academy Science Letters
02	2018	Nearly optimal separation between partially and fully retroactive data structures	Chen *et al.*(2018)	Leibniz International Proceedings in Informatics, LIPIcs
03	2018	Dynamizing Dijkstra: A solution to dynamic shortest path problem through retroactive priority queue	Garg *et al.*(2018)	Journal of King Saud University Computer and Information Sciences
04	2015	Polylogarithmic fully retroactive priority queues via hierarchical checkpointing	Demaine *et al.*(2015)	Lecture Notes in Computer Science (including subseries Lecture Notes in artificial Intelligence and Lecture Notes in Bioinformatics)
05	2010	Cloning Voronoi diagrams via retroactive data structures	Dickerson *et al.*(2010)	Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
06	2007	Retroactive data structures	Demaine *et al.*(2007)	ACM Transactions on Algorithms
07	2007	Non-oblivious retroactive data structures	Acar *et al.*(2007)	Technical report, Carnegie Mellon University

Demaine *et al.* [4] também explicam como gerar uma fila de prioridade totalmente retroativa utilizando o aumento na complexidade de $\sqrt{m}$ por meio de várias filas de prioridade parcialmente retroativas.

Acar *et al.* [5] propuseram uma outra classe de problemas relacionados à retroatividade, que é a noção de retroatividade não-consistente. Nessa classe de problemas, o que se deseja é, a partir da sequência de operações realizadas durante toda a estrutura, descobrir, após uma modificação, o primeiro instante de tempo após a modificação em que a estrutura se tornou inconsistente. Essa noção é importante para resolver problemas em que uma modificação no passado gera algum tipo de efeito cascata na estrutura, e as mudanças geradas por esse efeito afetam outras estruturas na resolução do problema em que a estrutura está sendo aplicada.

Dickerson *et al.* [7] utilizaram estruturas de dados retroativas para uma aplicação geométrica. Nesse estudo, os autores propuseram um algoritmo em que é possível clonar Diagramas de Voronoi utilizando estruturas de dados retroativas. Esse algoritmo é uma

ε-aproximação do clone de um Diagrama de Voronoi, permitindo a clonagem desse diagrama em tempo $O(n\ lg(1 / \varepsilon))$.

Demaine *et al.* [8] propuseram um decréscimo na complexidade temporal e espacial por meio da utilização de árvores de segmento e do fato de que as filas de prioridade parcialmente são *time-fusibles*. Uma estrutura é dita *time-fusible* se existem duas estruturas S1 e S2, em que S1 corresponde a um intervalo contínuo [l, t] e S2 corresponde a um intervalo [t + 1, r], sendo possível gerar algum tipo de relacionamento que represente o intervalo [l, r] a partir de S1 e S2. Foi provado que é possível gerar uma fila de prioridade totalmente retroativa em tempo polilogarítmico $O(lg^2(n))$ por atualização e $O(lg^2(m)\ lg(lg(n)))$ por consulta, tendo um acréscimo espacial de $O(m\ lg(m))$.

Garg *et al.* [9] e Garg *et al.* [10] tratam de uma aplicação da retroatividade não-consistente para a dinamização de grafos em problemas de obtenção de caminho mínimo. Pode-se observar que, nesse caso, a alteração de uma aresta no grafo gera um efeito cascata na árvore de caminhos gerada pelo algoritmo de Dijkstra, afetando o vetor de distâncias mínimas a partir de um ponto. Portanto, é necessária a utilização da versão não retroativa da estrutura, pois, dessa forma, é possível atualizar o vetor de distâncias de maneira mais otimizada em vez de simplesmente percorrer todo o grafo novamente. Porém, no pior caso, o algoritmo mais otimizado ainda percorrerá todos os vértices desse grafo (no caso de uma aresta afetar todos os caminhos mínimos de todos os vértices).

Chen *et al.* [11] propõem uma otimização na complexidade temporal com relação à transformação de estruturas parcialmente retroativas em suas versões totalmente retroativas. No artigo de Demaine *et al.* [4], foi provado que, se é gasta complexidade temporal de $T(n, m)$ em uma estrutura parcialmente retroativa, é possível transformá-la em sua versão totalmente retroativa com complexidade temporal de $O(T(n, m)\ \sqrt{m})$, em que n é o tamanho da estrutura e m é o número de operações em sua linha temporal. Já no estudo em tela, foi provado que é possível reduzir a diferença de complexidade entre as retroatividades parcial e total em $O(min\{\sqrt{m}, nlg(m) \wedge \{1 - o(1)\}\})$.

4. DADOS COLETADOS

Pelas estruturas propostas e pela análise de complexidade realizadas por Demaine *et al.* [4] e Demaine *et al.* [8], obteve-se as Tabelas 4 e 5 relacionadas às complexidades das versões parcialmente e totalmente retroativas das estruturas. Nessas tabelas, a variável m representa o número de possíveis espaços temporais para a realização de modificações e consultas na estrutura; já a variável n diz respeito ao número de operações realizadas.

Tabela 4. Complexidade para implementação das estruturas parcialmente retroativas. Fonte: Os autores.

Estrutura de dados	Tempo	Espaço	Autores
Dicionário	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Fila	$O(1)$	$O(n\lg(m))$	Demaine *et al.*(2007)
Pilha	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Union-Find	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Fila de prioridade	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)

Nos artigos citados, as estruturas parcialmente e totalmente retroativas não foram implementadas em nenhuma linguagem, sendo apresentado somente o método para a criação dessas estruturas. Para a obtenção das complexidades apresentadas, foram utilizadas árvores binárias balanceadas, como Treaps [12], AVL's ou árvores rubro-negras [13], em que, dentro de cada nó dessas árvores, foram armazenadas algumas informações, por exemplo, o mínimo/máximo da soma dos nós contidos em uma sub-árvore e a soma dos valores dos nós de uma sub-árvore.

A única estrutura que se difere das outras com relação às estruturas de dados internas para a implementação da estrutura retroativa é a relacionada à união de conjuntos *(Union-find)*. Nessa estrutura, é possível dois tipos de operação: a união de dois conjuntos disjuntos e a consulta relacionada a qual conjunto pertence um dado elemento. Nesse problema, para a obtenção da estrutura totalmente retroativa, é necessária a utilização de uma estrutura de dados chamada *Link-cut tree* [14], que é um tipo de estrutura baseada em *Splay-Trees* [15].

Tabela 5. Complexidade para implementação das estruturas totalmente retroativas. Fonte: Os autores.

Estrutura de dados	Tempo	Espaço	Autores
Dicionário	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Fila	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Pilha	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Union-Find	$O(\lg(m))$	$O(n\lg(m))$	Demaine *et al.*(2007)
Fila de prioridade (2007)	$O(\sqrt{m})$ atualizações $O(\sqrt{m}\lg(m))$ consultas	$O(n\lg(m)\sqrt{m})$	Demaine *et al.*(2007)
Fila de prioridade (2015)	$O(\sqrt{m})$ atualizações $O(\sqrt{m}\lg(m))$ consultas	$O(n\lg^2(m))$	Demaine *et al.*(2015)

Acar *et al.* [5] propuseram soluções com as seguintes complexidades para as estruturas de dados com retroatividade não consistente (Tabela 6).

Tabela 6. Complexidade para implementação da retroatividade não consistente. Fonte: Os autores.

Estrutura de dados	Tempo	Espaço	Autores
Dicionário	$O(\lg(\lg(m)))$	$O(m)$	Acar *et al.*(2007)
Fila	$O(\lg(\lg(m)))$	$O(m)$	Acar *et al.*(2007)
Pilha	$O(\lg(m)\lg(\lg(m)))$	$O(m)$	Acar *et al.*(2007)
Fila de prioridade	$O(\lg(m))$	$O(m)$	Acar *et al.*(2007)

Na Tabela 6, novamente, a variável m consiste no número de espaços temporais existentes na estrutura. Observe que os limites superiores, tanto de espaço quanto de tempo, para as estruturas com retroatividade não consistente tendem a ser mais otimizados que as suas versões parcialmente e totalmente retroativas.

Como nenhuma dessas estruturas de dados foi implementada de modo prático por parte de seus autores, para os testes empíricos desta pesquisa, foram implementadas as estruturas na linguagem C++. Os códigos e os testes realizados podem ser vistos em https://github.com/juniorandrade1/Master/tree/master/src.

5. TESTES EMPÍRICOS

Foram realizados alguns testes práticos relacionados às estruturas propostas por Demaine *et al.* [4] sobre estruturas de dados retroativas. Foram realizadas duas implementações do paradigma temporal para cada uma das estruturas: por força-bruta e utilizando os conceitos propostos por Demaine *et al.* [4]. No algoritmo por força bruta, foi utilizado o método de *rollback*, que consiste em refazer todas as operações de uma estrutura à medida que as consultas estiverem sendo realizadas. Em outras palavras, nessa abordagem armazenam-se as operações de atualização ordenadas por tempo de execução e, quando uma operação de consulta acontece, as operações são executadas sequencialmente

na estrutura de dados não retroativa até o tempo desejado. As linhas em verde representam o desempenho dos algoritmos por força-bruta, utilizando o método de *rollback*, enquanto as linhas vermelhas representam o desempenho dos algoritmos retroativos apresentados por esse algoritmo.

Todos os testes apresentados nesta pesquisa foram executados em um computador Intel Core i5-4200U CPU @ 1.60GHz x 4 com 8 gigabytes de memória. Eles foram gerados de maneira totalmente aleatória, sempre mantendo a consistência (ou seja, as operações realizadas sempre serão válidas para todos os tempos da estrutura), e de modo que todas as operações sejam realizadas em tempos distintos. Para aferição de desempenho foram utilizadas duas ferramentas: (i) GTest, uma ferramenta do Google para automação de testes e avaliação de desempenho temporal de um código; e (ii) Valgrind, para o cálculo do consumo de memória dos algoritmos.

Na Figura 2 é apresentado o desempenho da estrutura anteriormente sugerida com relação a uma implementação por força bruta da retroatividade parcial. Pode-se observar que, para instâncias menores, o algoritmo comum é competitivo pela constante alta do algoritmo logarítmico. Contudo, em entradas aleatórias com mais de mil operações, o algoritmo apresentado já é executado aproximadamente cinco vezes mais rápido que o algoritmo padrão, e essa distância de processamento entre os dois algoritmos cresce rapidamente à medida que o número de operações aumenta.

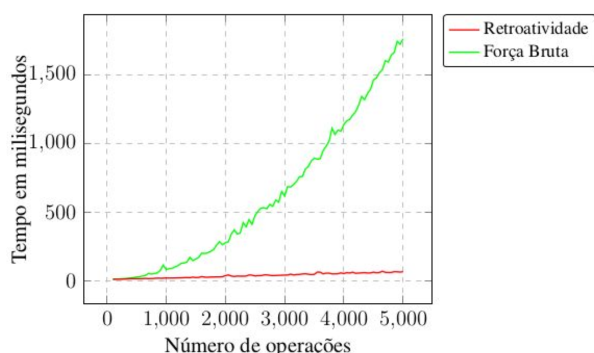

Figura 2. Execução da fila parcialmente retroativa comparada à solução padrão. Fonte: Os autores.

Todavia, o consumo de memória do algoritmo retroativo é maior que o da solução força bruta para o problema, como pode ser visto na Figura 3. A imagem confirma a tendência do algoritmo proposto por Demaine *et al*. [4] para a fila parcialmente retroativa de ser mais custoso em termos de memória, uma vez que esse algoritmo utiliza duas árvores binárias balanceadas, enquanto no algoritmo força bruta é necessário somente a manutenção de uma árvore binária balanceada.

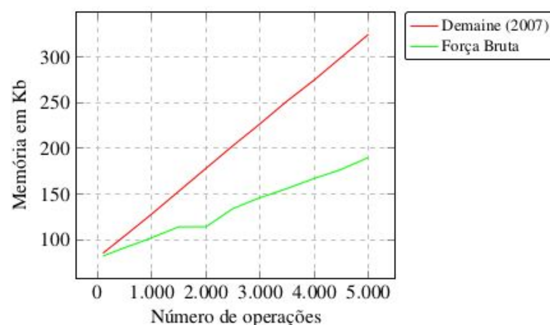

Figura 3. Consumo de memória de uma fila parcialmente retroativa. Fonte: Os autores.

Na Figura 4 tem-se o gráfico comparativo com a velocidade de execução de duas abordagens para a implementação da estrutura fila totalmente retroativa. Nesse caso, a execução do algoritmo por força bruta é ligeiramente mais veloz que a sua versão parcialmente retroativa. Isso é explicado pelo fato de que na fila parcialmente retroativa é necessário executar todas as operações realizadas na estrutura, enquanto na fila totalmente retroativa essa reconstrução depende do valor *t* correspondente à versão da estrutura consultada. Novamente, pode-se observar que nos casos maiores, a vantagem da utilização da estrutura apresentada é expressiva.

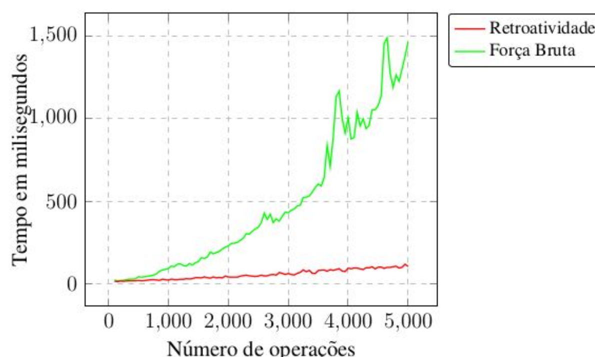

Figura 4. Execução da fila totalmente retroativa comparada à solução padrão. Fonte: Os autores.

Na Figura 5 são apresentados os consumos de memória pelos algoritmos propostos. Novamente, o algoritmo retroativo proposto por Demaine *et al*. [4] consome mais memória que o algoritmo força bruta, pois armazena uma árvore binária balanceada a mais que o algoritmo força bruta. Contudo, é importante observar que em ambos os algoritmos para a fila retroativa, tanto em sua versão parcialmente retroativa quanto em sua versão totalmente retroativa, a complexidade de memória é de *O(n)*.

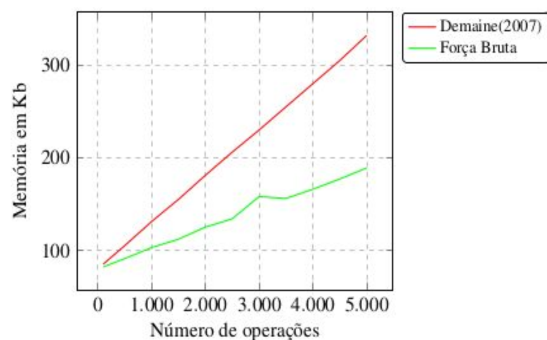

Figura 5. Consumo de memória de uma fila totalmente retroativa. Fonte: Os autores.

Na Figura 6 tem-se a execução de testes aleatórios em uma pilha parcialmente retroativa. Pode-se observar que pela natureza da estrutura em que a pilha está implementada no algoritmo mais eficiente, a constante na complexidade desse algoritmo é mais alta, e, portanto, para testes extremamente pequenos, o algoritmo por força bruta é mais rápido que o algoritmo eficiente. Porém, à medida que o número de operações cresce, o algoritmo que implementa a persistência parcial abre larga vantagem com relação ao tempo de execução, chegando a ser trinta vezes mais rápido em casos com cinco mil operações.

Na Figura 7 é apresentado o consumo de memória da execução dos mesmos testes em uma pilha parcialmente retroativa. É notável a diferença de consumo em termos de memória da implementação baseada no algoritmo de Demaine *et al*. [4] com relação ao algoritmo força bruta. O algoritmo de Demaine *et al*. [4] consome mais memória, pois utiliza uma série de estruturas mais sofisticadas para manutenção da estrutura, enquanto o algoritmo força bruta somente armazena as operações.

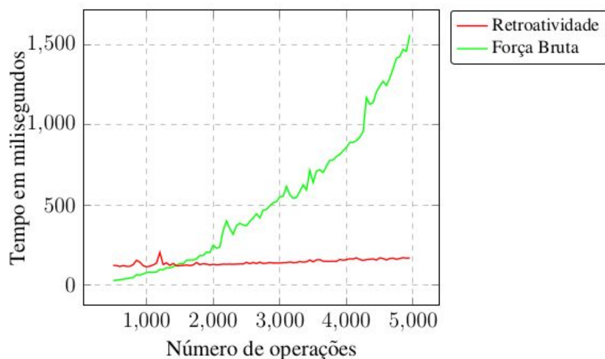

Figura 6. Teste de velocidade de execução de uma pilha parcialmente retroativa. Fonte: Os autores.

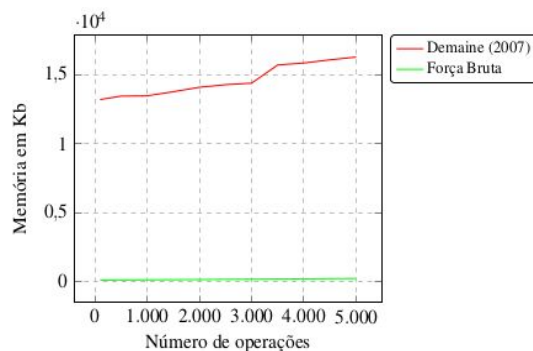

Figura 7. Consumo de memória de uma pilha parcialmente retroativa. Fonte: Os autores.

Na Figura 8, mais uma vez, pode-se observar a vantagem na utilização da estrutura totalmente retroativa com relação à sua versão força-bruta. Para casos com poucas operações, o algoritmo trivial é mais veloz que a sua versão retroativa, pois a constante implícita na complexidade do algoritmo retroativo é alta. Entretanto, à medida que o número de operações cresce, a pilha totalmente retroativa é mais rápida que a sua implementação trivial.

Na Figura 9 são apresentados os resultados relacionados ao consumo de memória na execução dos mesmos casos de teste utilizados na Figura 8. Novamente, existe uma diferença considerável entre o consumo de memória dos dois algoritmos. O algoritmo força bruta consome memória proporcional ao número de operações realizadas, enquanto o algoritmo implementado, baseado na solução proposta por Demaine *et al*. [4] para a manutenção de uma pilha totalmente retroativa de maneira otimizada, consome memória proporcional ao tamanho da linha do tempo em que a estrutura está inserida.

Na Figura 10, é possível observar o gráfico relacionado aos testes em uma fila de prioridade parcialmente retroativa. Os testes de velocidade mostraram que, no caso geral, a fila de prioridade parcialmente retroativa implementada é consideravelmente mais rápida que em sua implementação refazendo a estrutura a cada consulta. A partir de mil operações realizadas já é possível que o algoritmo apresentado seja, em média, cinco vezes mais rápido que o algoritmo por força bruta, chegando a ser 25 vezes mais rápido para casos próximos a 5000 operações.

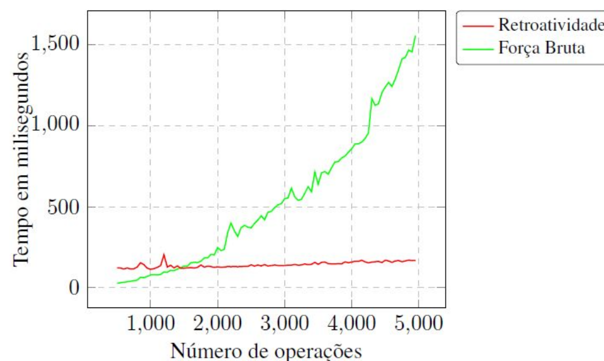

Figura 8. Teste de velocidade de execução de uma pilha totalmente retroativa. Fonte: Os autores.

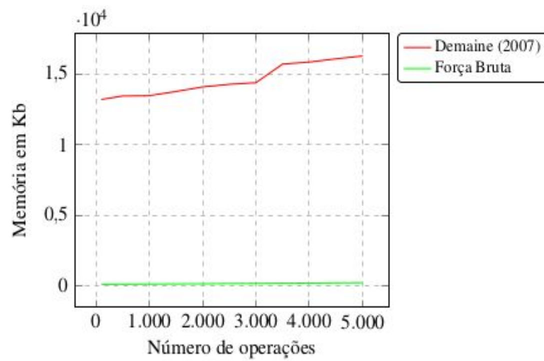

Figura 9. Consumo de memória de uma pilha totalmente retroativa. Fonte: Os autores.

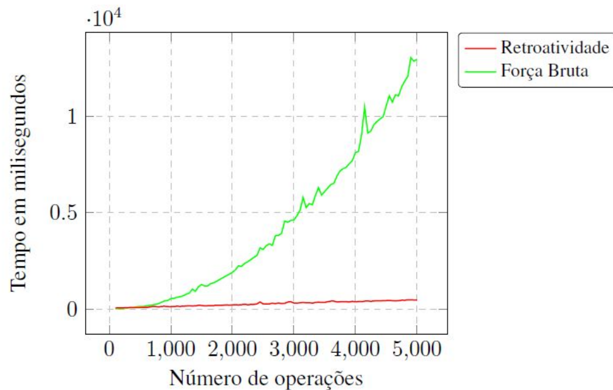

Figura 10. Teste de velocidade de execução de uma fila de prioridade parcialmente retroativa. Fonte: Os autores.

Na Figura 11 é apresentado o resultado para o teste de memória das implementações da fila de prioridade parcialmente retroativa. O algoritmo de Demaine *et al*. [4] apresenta uma curva de crescimento um pouco mais inclinada que a implementação força bruta da estrutura.

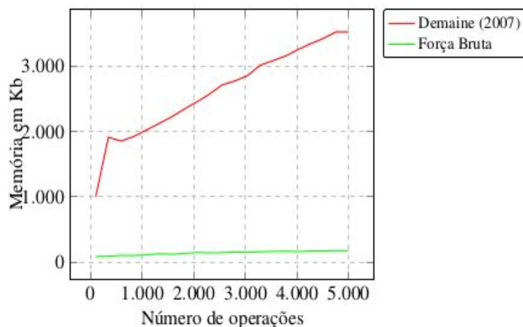

Figura 11. Consumo de memória de uma fila de prioridade parcialmente retroativa. Fonte: Os autores.

Na Figura 12, tem-se os testes comparativos do algoritmo força bruta com relação a duas implementações propostas por Demaine *et al*. em [4] e [8]. A fila de prioridade retroativa por força bruta foi implementada com uma *min-heap*, e cada consulta do tipo *Get(t)* foi realizada em tempo *O(n lg(n))*. Os dois algoritmos retroativos propostos por Demaine *et al*. são mais "estáveis" do ponto de vista temporal, uma vez que a curva de crescimento do algoritmo por força bruta é mais inclinada que a curva de crescimento dos algoritmos retroativos.

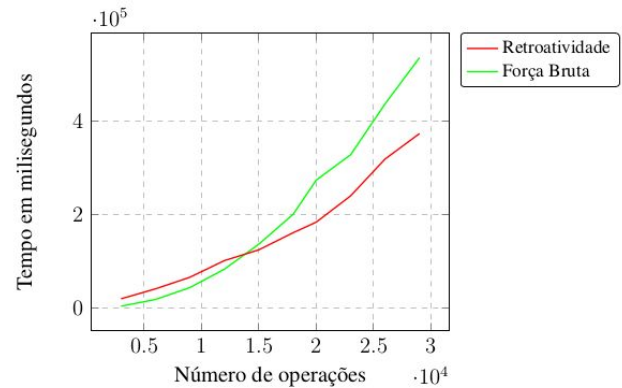

Figura 12. Teste de velocidade de execução de uma fila de prioridade totalmente retroativa. Fonte: Os autores.

Na Figura 13 são apresentados os testes referentes aos consumos de memória nas implementações da fila de prioridade totalmente retroativa. O algoritmo força bruta tem um consumo de memória ínfimo comparado ao consumo de memória dos algoritmos mais elaborados. O algoritmo força bruta somente mantém as operações ordenadas, consumindo muito tempo de processamento nas consultas, mas com um baixo consumo de memória. O algoritmo de Demaine *et al*. [4] consome muita memória, pois além de armazenar várias filas de prioridade parcialmente retroativas, ao realizar uma operação de atualização, no pior caso, o objeto atualizado será inserido em $\sqrt{m}$ filas de prioridade parcialmente retroativas. O algoritmo de Demaine *et al*. [8], no pior caso de uma atualização, modifica *lg(m)* filas de prioridade parcialmente retroativas, o que explica a diferença de consumo de memória nesses dois algoritmos.

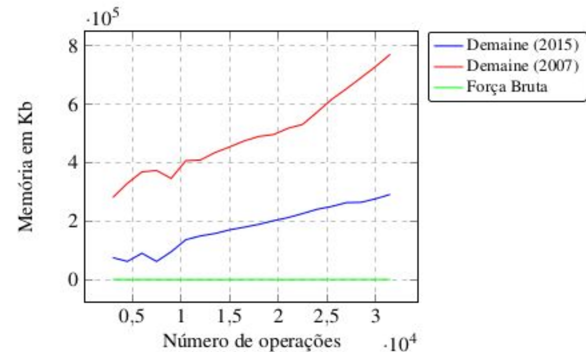

Figura 13. Consumo de memória de uma fila de prioridade totalmente retroativa. Fonte: Os autores.

Finalmente, não foram realizados testes em problemas que utilizam algum conceito de retroatividade, pois esses problemas envolvem outras estruturas de dados que não são temporais. Porém, todas as estruturas de dados retroativas propostas por Demaine *et al*. [4, 8] foram testadas nesta pesquisa.

6. CONSIDERAÇÕES FINAIS

Esse artigo apresentou os resultados obtidos pelos estudos, até o presente momento, sobre estruturas de dados retroativas. Alguns artigos em que essas estruturas retroativas são utilizadas foram

apresentados, por exemplo, na dinamização de algoritmos como o Dijkstra [9, 10], e aplicações geométricas como a clonagem de Diagramas de Voronoi [7].

Também foram realizados alguns testes práticos com as estruturas propostas por Demaine *et al*. [4] para análise e comparação com relação às implementações força-bruta dessas estruturas. Os testes mostraram que, em geral, as estruturas propostas são algumas vezes mais velozes que as suas versões força-bruta. As versões força bruta são mais eficientes apenas em casos com instâncias pequenas, por conta da constante implícita em alguns desses algoritmos. Porém, à medida que o número de operações e de tamanho da linha temporal aumenta, os algoritmos retroativos tendem a ser mais estáveis, no sentido de que ficam mais lentos em uma proporção menor que os algoritmos triviais. Pode-se observar que as estruturas de dados retroativas são, em grande parte, implementadas sobre estruturas de dados otimizadas, como árvores binárias balanceadas, que, no geral, tendem a ter um bom desempenho tanto em tempo quanto em espaço.

Destaca-se, ainda, o baixo número de artigos encontrados especificamente sobre estruturas de dados retroativas, e isso se deve a dois fatores principais: dificuldade na adaptação das estruturas nos problemas e tempo de descoberta das estruturas. Essas estruturas, em geral, não têm uma aplicação prática direta, o que dificulta sua utilização em problemas, sendo necessárias, às vezes, algumas modificações nas ideias sugeridas. Outro ponto é o tempo de descoberta dessas estruturas. O primeiro artigo relacionado às estruturas foi apresentado em 2007, e, portanto, houve pouco tempo para o desenvolvimento de outras estruturas, bem como da literatura sobre otimização das estruturas propostas anteriormente.

Sobre possíveis direções de pesquisa, pode-se observar que as estruturas parcialmente retroativas já estão próximas a um limite na complexidade temporal (próximo a $lg(m)$ por operação). Já as estruturas totalmente retroativas são um pouco mais custosas nos aspectos temporal e espacial como, por exemplo, no caso de filas de prioridade totalmente retroativas. Portanto, uma direção possível para uma pesquisa futura seria a criação de novos métodos para a transformação de cada estrutura de sua retroatividade parcial para total.

AGRADECIMENTOS

Os autores agradecem à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pela concessão da bolsa de mestrado para o primeiro autor e pelo apoio financeiro para a realização desta pesquisa.

REFERÊNCIAS

[1] Driscoll, J. R., Sarnak, N., Sleator, D. D., and Tarjan, R. E. (1989). Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124.

[2] Mehta, D. P. and Sahni, S. (2004). Handbook of Data Structures and Applications. Chapman and Hall/CRC.

[3] Conchon, S. and Filliâtre, J. C. (2008). Semi-persistent data structures. In *European Symposium on Programming*, pages 322–336. Springer.

[4] Demaine, E. D., Iacono, J., and Langerman, S. (2007). Retroactive data structures. *ACM Transactions on Algorithms* (TALG), 3(2):13.

[5] Acar, U. A., Blelloch, G. E., and Tangwongsan, K. (2007). Non-oblivious retroactive data structures. *Technical report*, Carnegie Mellon University.

[6] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269-271.

[7] Dickerson, M. T., Eppstein, D., and Goodrich, M. T. (2010). Cloning voronoi diagrams via retroactive data structures. In *European Symposium on Algorithms*, pages 362–373. Springer.

[8] Demaine, E. D., Kaler, T., Liu, Q., Sidford, A., and Yedidia, A. (2015). Polylogarithmic fully retroactive priority queues via hierarchical checkpointing. In *Workshop on Algorithms and Data Structures*, pages 263–275. Springer.

[9] Garg, D. et al. (2018). Dynamizing dijkstra: A solution to dynamic shortest path problem through retroactive priority queue. *Journal of King Saud University-Computer and Information Sciences*, no prelo.

[10] Garg, D. et al. (2019). A retroactive approach for dynamic shortest path problem. *National Academy Science Letters*, 42(1):25–32.

[11] Chen, L., Demaine, E. D., Gu, Y., Williams, V. V., Xu, Y., and Yu, Y. (2018). Nearly optimal separation between partially and fully retroactive data structures. ArXiv preprint arXiv:1804.06932.

[12] Martínez, C. and Roura, S. (1998). Randomized binary search trees. *Journal of the ACM* (JACM), 45(2):288–323.

[13] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*, The MIT Press, 3rd edition.

[14] Sleator, D. D. and Tarjan, R. E. (1983). A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391.

[15] Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM* (JACM), 32(3):652–686.