Implementação e Análise da Estrutura de dados Lista Duplamente Encadeada

Lucas Fontes Buzuti
Departamento de Engenharia Elétrica
Centro Universitário FEI
São Bernardo do Campo-SP, Brasil
lucas.buzuti@outlook.com

***Resumo*—Esse artigo tem uma finalidade acadêmica na compreensão e implementação da estrutura de dados lista duplamente encadeada. Uma análise da estrutura é proposta. A implementação foi utilizar a programação orientada a objetos na linguagem *C++*, tendo em foco a otimização e a velocidade na execução de algoritmos.**

***Index Terms*—estrutura de dados, lista duplamente encadeada, C++**

I. Introdução

Esse artigo tem em seu objetivo a compreensão e implementação da estrutura de dados lista duplamente encadeada. Além da compreensão, uma análise crítica da estrutura de dados e suas variações é proposta. A implementação tem como alvo a utilização da linguagem *C++*, pois é uma linguagem focada na otimização e na velocidade da execução de algoritmos.

O contexto de utilizar listas encadeada se dá na fácil implementação de outras estruturas de dados ou algoritmos, tais como: pilhas, filas, árvores binárias, algoritmo árvore de decisão e entre outros [1] [7] [8]. Entretanto, para a formulação de outra estruturas de dados e algoritmos, utiliza-se variações da lista encadeada, sendo estas: lista simplesmente encadeada, duplamente encadeada e quadruplamento encadeada.

II. Teoria

Entre 1955 à 1956, Newell, Shaw e Simon da RAND Corporation desenvolveram as listas encadeadas, como a principal estrutura de dados para Linguagem de Processamento de Informação (Information Processing Language ou IPL) [2]. O IPL proposto pelos autores foram utilizados no desenvolvimentos de diversos programas de inteligência artificial, tais como: o *Logic Theory Machine* e um programa de xadrez para computadores. Relatórios relacionado ao trabalho IPL foi apresentados no IRE Transactions on Information Theory em 1956 [2].

A lista duplamente encadeada é uma estrutura de dados ligadas entre si. Em outras palavras, são conjunto de dados que estão sequencialmente ligados, no qual são denominados de nós (node). A lista duplamente encadeada é uma extensão da lista simplesmente encadeada. Cada nó contem três informações, sendo estas, o dado (a informação que será anexado a lista) e dois *links*. Os *links* são referências para o nó posterior e para o nó anterior da lista (sequência de nós). Os dois terminais da lista conhecidos como: cabeça (head) e cauda (tail), são apontados para um tipo nulo de terminador, para facilitar os limites da lista e o seu percorrimento.

III. Proposta e Implementação

Este artigo propõe utilizar a linguagem *C++* para a implementação da estrutura lista duplamente encadeada. Toda essa implementação foi feita em um computador com sistema operacional *Linux* e com o compilador *GCC*, a versão da linguagem utilizada foi a *C++11*. A estrutura tem em sua implementação a programação orientada a objetos (POO), onde visa a construção de classes e métodos, ilustrada na Figura 1. Além disso, foi proposto a implementação das estruturas de dados pilha e fila a partir da implementação da lista duplamente encadeada. A implementação foi passível devido ao conceito de herança, trazida pela POO, mostrada na Figura 2.

Implementou-se duas classes[1], sendo essas a classe *LinkedList* (implementação da lista) e a classe *Node* (implementação do nó). Na classe *LinkedList* foi efetuada as operações de uma lista, essas operações são os métodos de classe, portanto, denominou-se os métodos *getSize* (o retorno do tamanho da lista), *insert* (inserção de um nó a lista) e seus derivados, *remove* (remoção de um nó da lista) e seus derivados, *search* (retorna um nó pesquisado), e *isEmpty*. Na classe *Node*, tem-se os atributos que representam a referência para o nó anterior e posterior, além, da método *getData*, no qual obtém-se o elemento contido no nó. Na Figura 1, pode-se visualizar o diagrama UML da classe *LinkedList* e também a classe de exceção.

A partir da classe da lista duplamente encadeada implementada, pode-se implementar as classes pilha e fila com fundamento de herança. Devido ao conceito dos nós das listas encadeadas serem dinamicamente alocados, portanto, ao utilizar herança na implementação de pilha e fila, acaba-se implementando o conceito de pilha e fila dinamicamente alocadas. Na Figura 2, ilustra-se o diagrama UML das classes *Stack* (pilha) e *Queue* (fila).

IV. Experimentos e Resultado

Para testar a proposta desse artigo foi codificado três arquivos, *mainList*, *mainStack* e *mainQueue* para cada estrutura

[1] https://github.com/buzutilucas/scientific-programming/tree/master/Ex02

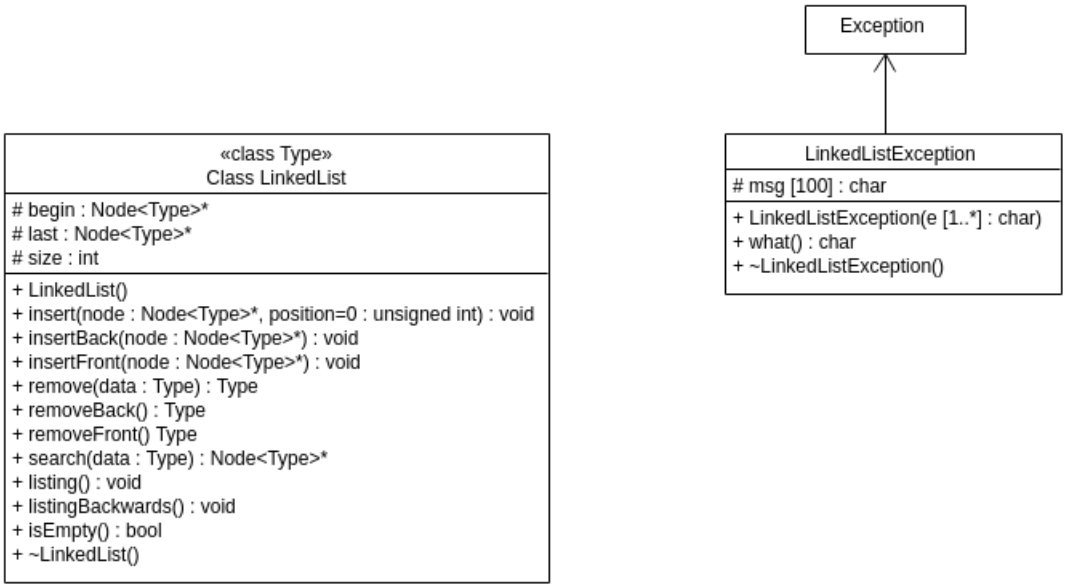

Figura 1. UML da classe *LinkedList*

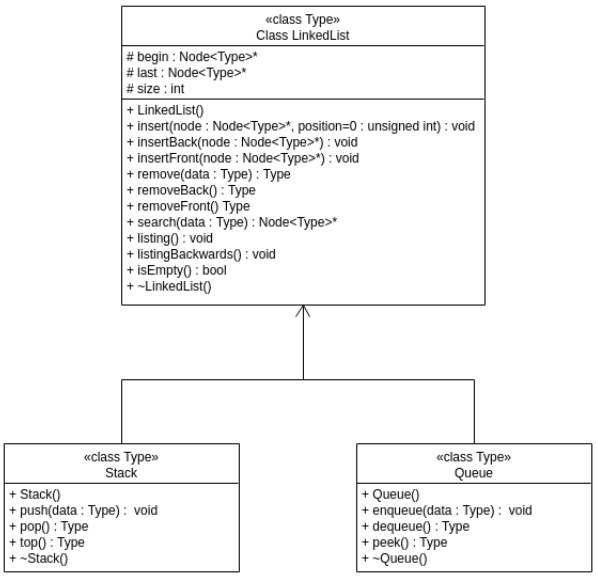

Figura 2. UML das classes *Stack* (pilha) e *Queue* (fila)

de dados, o arquivo foi escrito em *C++*. Nesse arquivo foi introduzido estruturas para testar os métodos de classe, as entradas de dados foram feitas automaticamente, mas podendo ser testada por um serviço de integração contínua hospedado, usado para criar e testar projetos de software. Os resultados do teste pode ser visualizados nas Figuras 3, 4 e 5.

Os métodos implementados nas classes abordada nesse artigo não sofreram erros de compilação, pois os métodos de classe foram projetados para sofrerem tratamento de erros. Em outras palavras, se o usuário da classe tentar adicionar um nó em uma posição invalida, remover ou fazer uma busca, no qual um nó não existe, uma mensagem de exceção é impressa, sendo assim evitando a interrupção do código por erro de compilação.

```
Adding elements in a list...
List: 1 2 3 4 5 6 7 8 9 10
List backwards: 10 9 8 7 6 5 4 3 2 1


Adding the element '100' in the position [5].
List now: 1 2 3 4 5 100 6 7 8 9 10
List backwards: 10 9 8 7 6 100 5 4 3 2 1

Adding another element in the position [12], but an error is expected.
Exception occurred: Position access failure.

Adding the element '50' to the front.
List now: 50 1 2 3 4 5 100 6 7 8 9 10
List backwards: 10 9 8 7 6 100 5 4 3 2 1 50

Adding the element '150' to the back.
List now: 50 1 2 3 4 5 100 6 7 8 9 10 150
List backwards: 150 10 9 8 7 6 100 5 4 3 2 1 50


Removing the element '100'.
Data remove: 100
List now: 50 1 2 3 4 5 6 7 8 9 10 150
List backwards: 150 10 9 8 7 6 5 4 3 2 1 50


Removes an element to the front.
Data remove: 50
List now: 1 2 3 4 5 6 7 8 9 10 150
List backwards: 150 10 9 8 7 6 5 4 3 2 1

Removes an element to the back.
Data remove: 150
List now: 1 2 3 4 5 6 7 8 9 10
List backwards: 10 9 8 7 6 5 4 3 2 1


Searching the element '10'.
Element searched: 10

Searching the element '11'.
Element not found


Removes all element...
List now:
Removes another element...
Exception occurred: List is empty.
```

Figura 3. Resultados dos testes da classe *LinkedList*

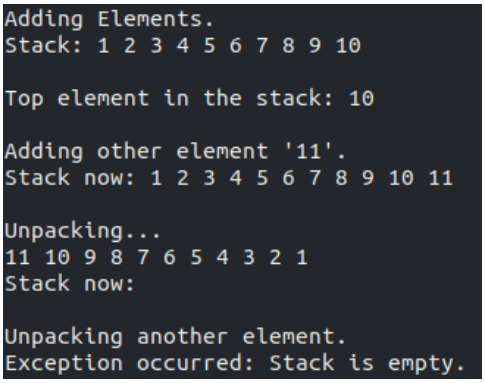

Figura 4. Resultados dos testes da classe *Stack*

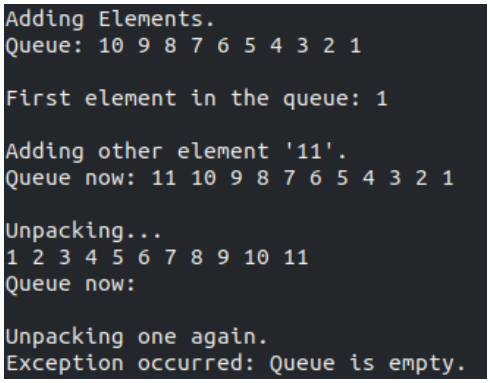

Figura 5. Resultados dos testes da classe *Queue*

Analisando os resultados dos testes das classes *Stack* e *Queue*, conclui-se que o conceito de herança é útil em diversos aspectos, pois comprovou-se que com a implementação de listas foi possível implementar pilha e fila, de jeito rápido e eficaz.

V. Trabalhos Correlatos

Mesmo listas encadeadas sendo estrutura de dados antiga, trabalhos relevantes nos dias atuais e até mesmo trabalhos de importância para os dias atuais propostos na década de 1990, 1980 ou 1970, usufruem nessas estruturas de dados [3] [5]. Algoritmos modernos de inteligência artificial e visão computacional utiliza-se das teorias ou das próprias estruturas de dados em sua modelação matemática [4] [6].

VI. Conclusão

Nesse artigo pode compreender e implementar a estrutura de dados lista duplamente encadeada. Além da compreensão uma análise da utilização da estrutura em outras estruturas de dados e algoritmos foi estabelecida. Conclui-se que a utilização de listas encodeadas para implementar outras estruturas tem um aspecto eficaz devido a sua alocação dinâmica. E na questão de codificação, ao utilizar listas para implementar outras estruturas tais como, pilha e fila, a reutilização de código como herança, faz uma implementação rápida e eficaz. Vislumbra-se, como trabalhos futuros, a utilização dessa estrutura de dados em algoritmos modernos tal como *Deep Learning*.

Referências

[1] Belson, William A. ”Matching and prediction on the principle of biological classification.”Journal of the Royal Statistical Society: Series C (Applied Statistics) 8.2 (1959): 65-75.
[2] Newell, Allen, and Herbert Simon. ”The logic theory machine–A complex information processing system.”IRE Transactions on information theory 2.3 (1956): 61-79.
[3] Shafiei, Niloufar. “Non-Blocking Doubly-Linked Lists with Good Amortized Complexity.” OPODIS (2014).
[4] Zhou, Rong and Eric A. Hansen. “Breadth-First Heuristic Search.” KR (2004).
[5] Donchev, Ivaylo and Emilia Todorova. “Implementation of ADS Linked List Via Smart Pointers.” (2015).
[6] Yang, Hao, Hesham A. Rakha and Mani Venkat Ala. “Eco-Cooperative Adaptive Cruise Control at Signalized Intersections Considering Queue Effects.” IEEE Transactions on Intelligent Transportation Systems 18 (2016): 1575-1585.
[7] Bauer, Friedrich L., et al. ”Formal program construction by transformations-computer-aided, intuition-guided programming.”IEEE Transactions on Software Engineering 15.2 (1989): 165-180.
[8] Sundarapandian, Vaidyanathan. Probability, statistics and queuing theory. PHI Learning Pvt. Ltd., 2009.