Relatório 2

Estudo de Estruturas de Dados e o reuso de recursos.

Cristiano Lopes Moreira

Matrícula: 119103-0

Sumário

1. Introdução

Nas diversas áreas da engenharia, a administração dos recursos é um dos principais desafios no trabalho do engenheiro. Na engenharia de telecomunicações, são os recursos de espectro, e meios de transporte; na engenharia elétrica, os recursos de linhas de transmissão, e meios de geração e armazenamento de energia; na engenharia de software, os recursos de processadores e memória, como também o próprio código, que pode ser visto como recurso de tempo de desenvolvimento. Um método utilizado na administração dos recursos é o reuso. Através do reuso é possível melhorar o desempenho de sistemas de telecomunicações reutilizando os mesmos meios de transmissão, de se recriar novas rotas de transmissão de energia pelas malhas de gestão dinâmica e também de otimizar o uso de memória e reutilizar os códigos no desenvolvimento de software.

O Objetivo deste relatório é estudar as formas de administração dos recursos de memória e de código, utilizando as a estrutura de dados Lista ligada, Fila e Pilha como base para gerenciamento do recurso de memória e reutilização dos códigos de programação, utilizando técnicas de heranças na programação orientada a objeto.

Neste trabalho iremos modelar os algoritmos de estrutura de dados para Lista Ligada, fazer reuso de seus métodos para as estruturas de dados Fila e Pilha, verificar sua operação de escrita e leitura, realizar sua análise assintótica no impacto de tempo para algoritmos que necessitam dessa forma de estrutura de dados, e quantificar o tempo dessas operações com o aumento de interações com a estrutura de dados, com e sem reuso do código via técnica de herança.

2. Desenvolvimento teórico

2.1. Estrutura de dados:

As operações nas estruturas de dados são agrupadas em operações de consultas, que retornam a informação desejada, e operações de modificações, que possibilitam

alterar o conjunto de informações contidas; basicamente, representadas por 5 operações primitivas:

Busca (S,k): Verifica se o elemento ‘k’ está contido no agrupamento de dados ‘S’ .

Inserir (S,x): Insere o elemento ‘x’ no agrupamento de dados ‘S’.

Remover (S,x): Remove o elemento ‘x’ no agrupamento de dados ‘S’.

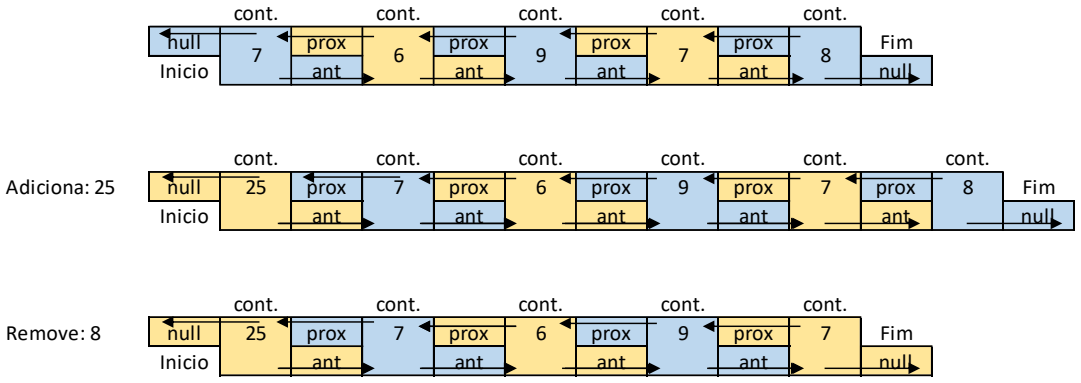
Minimo (S): Retorna o elemento de menor valor dentre os elementos do agrupamento ‘S’.

Maximo (S): Retorna o elemento de maior valor dentre os elementos do agrupamento ‘S’.

Listas ligadas são estruturas de dados dinâmicos que utilizam as operações “buscar”, “Inserir” e “Remover”, como básicas, e algumas operações de suporte, como “Posterior” e “Anterior”, que move os ponteiros da Lista ligada para “caminhar” pelos elementos da lista, e “Estado”, que informa se está vazia ou cheia.

Listas Ligadas

A lista ligada é uma estrutura de dados na qual os objetos são organizados em ordem linear, com o arranjo da lista determinada pelos índices que apontam o elemento anterior e posterior da lista. Os casos em que não existem valores (NULL) nos ponteiros de anterior e posterior, são os casos em que temos, respectivamente, o início e final da lista. As listas podem ainda ser ordenadas, com seus os valores inseridos na lista de forma ordenada, ou não ordenadas, em que os elementos são inseridos sem uma ordem determinada. Uma característica das listas ligadas é a alocação dinâmica de memória, que possibilita o uso/reuso somente da quantidade de memória necessária para a aplicação, não sendo necessário pré-alocar espaços grandes de memória que não serão utilizados.



Listas se assemelham a uma corrente de elos, no qual um elo está diretamente ligado ao seu sucessor e antecessor nesta corrente.

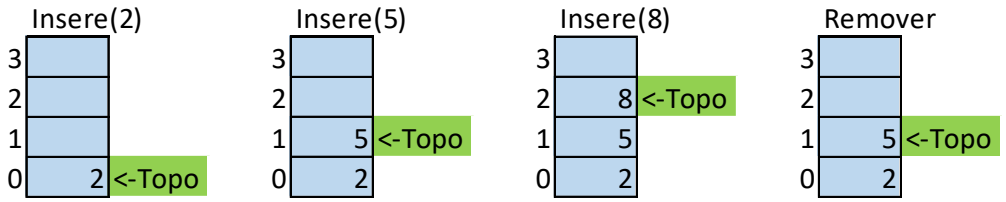
As operações utilizadas são:

- Iniciar Lista
- Inserir no início da Lista
- Inserir no fim da Lista
- Remover do início da Lista
- Remover do fim da lista
- Encontrar na lista
- Mostrar conteúdo do ponteiro da lista
- Estado, verificar estado da lista (vazia, normal)

Pilhas e Filas são estruturas de dados dinâmicos que utilizam as operações “Buscar”, “Inserir” e “Remover”, como básicas, e algumas operações de suporte, como “Proximo” (topo), que informa qual a próxima informação a ser disponibilizada, e “Estado”, que informa se está vazia ou cheia. Para estas estruturas de dados iremos pode ser utilizado o critério de reuso das subestruturas da Lista ligada para criar os objetos Pilha e Fila.

Pilhas

Em uma Pilha, o elemento eliminado do conjunto é o mais recentemente inserido, ela implementa a política LIFO (last in first out). Na estrutura de dados Pilha, é possível ter acesso a somente 01 item do agrupamento de dados por vez, o último a ser inserido. Essa estrutura de dados não se propõe a realizar busca de dados em sua cadeia. Para conhecer e utilizar o dado na penúltima posição de uma pilha é necessário remover o dado da última posição.



Pilhas se assemelham a um empilhamento de pratos, no qual o prato disponível e visível é o último prato que foi empilhado.

As operações utilizadas:

- Iniciar Pilha
- Inserir na Pilha (empilhar) – herança Lista insere no início da Lista
- Remover da Pilha (desempilhar) – herança lista remove do início da Lista
- Topo (Mostrar informação no topo da pilha) - herança lista mostra conteúdo com o ponteiro no início da Lista
- Estado, verificar estado da Pilha (pilha cheia, vazia, normal) – herança Lista Estado

Filas

Em uma Fila, o elemento eliminado do conjunto é o mais antigo inserido, é implementada a política FIFO (first in first out). Na estrutura de dados Fila, é possível

ter acesso a somente a 01 item do agrupamento de dados por vez, o primeiro a ser inserido. Essa estrutura de dados, assim como a Pilha, não se propõe a realizar busca de dados em sua cadeia. Para conhecer e utilizar o dado na segunda posição de uma pilha é necessário remover o dado da primeira posição.

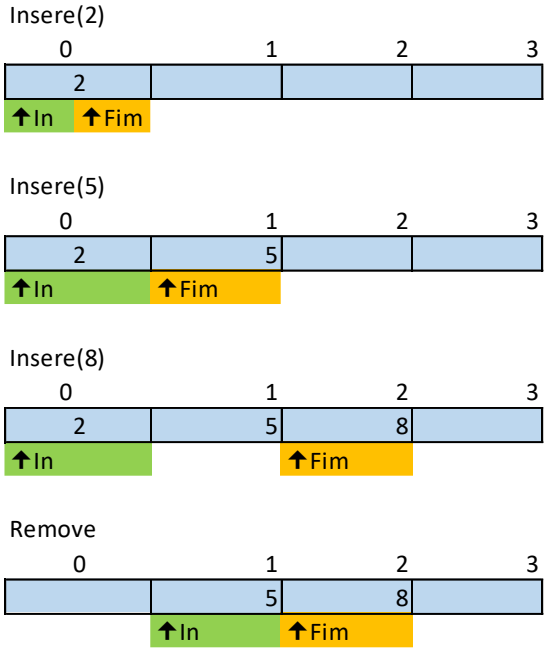
Filas se assemelham a uma fileira de pessoas em uma caixa registradora, a primeira pessoa a ser atendida será a primeira pessoa que chegou na fila, e a segunda a ser a tendida será a segunda que chegou na fila e assim por diante.

Uma característica da Fila, distinta da Pilha, é que a fila trabalha com dois apontadores, o início da Fila (cabeça) e o fim da Fila (cauda); quando um elemento é inserido na Fila ele ocupa seu lugar no final Fila. O elemento retirado da Fila é sempre o que está no início da fila.

Pelo uso dos apontadores, que acompanham o início e fim da fila, o algoritmo se norteia na inserção e remoção de um elemento da Fila. Quando entra um novo elemento, o apontador Fim é acrescido de +1, quando é removido um elemento o apontador Início é acrescido de +1. Esta rotina dinâmica de apontadores possibilita o uso circular da memória, já que toda vez que é liberado um elemento da Fila o “espaço” que este elemento ocupava se torna disponível. Para a utilização circular da memória é necessário implementar uma rotina que verifica se os apontadores alcançaram o final da memória. Quando isso ocorre, verifica-se se o início da memória está disponível, não marcada por nenhum apontador. Caso esteja disponível, é realizada uma volta reinicializando o apontador para primeira posição da memória.

As operações utilizadas:

- Iniciar Fila
- Inserir na Fila (enfileirar) - herança Lista insere no fim da lista
- Remover da Fila (desinfileirar) – herança lista remove do fim da lista
- Primeiro (Mostrar informação do primeiro elemento da Fila) - herança lista mostra conteúdo com o ponteiro no fim da Lista



Para a avaliação do desempenho de uma estrutura de dados, em geral, verifica-se o tempo de execução com a aplicação a que se destina levando-se em consideração o tamanho do conjunto de dados.

2.2. Programação orientada a objeto e reuso:

O desenvolvimento baseado em reuso é um mecanismo para atender a demanda por menor custo de produção e manutenção de software como também de reduzir o tempo de entrega e aumentar a qualidade e confiabilidade dos projetos.

A abordagem de reutilizar os componentes de um sistema, que necessitam de um conhecimento especializado nas construções de sub-rotinas, mostra-se eficaz para reuso de objetos e funções matemáticas através da criação de bibliotecas de

uso comum que possibilita a redução de tempo e recursos no desenvolvimento destas rotinas.

Pela definição de Schmidt et al. (2004), Framework é "... um conjunto integrado de artefatos de software (como classes, objetos e componentes) que colaboram para fornecer uma arquitetura reusável para uma família de aplicações relacionadas... " Schmidt et al. (2004), ou um conjunto de rotinas desenvolvidas com orientação a objetos que visam o reuso, quando agrupadas, é por definição um Framework. Schmidt também classifica os diversos tipos de frameworks como listado abaixo:

> *Fayad e Schmidt (1997) discutem três classes de frameworks*
>
> 1. *Frameworks de infraestrutura de sistema. Esses frameworks apoiam o desenvolvimento de infraestruturas de sistema, como comunicações, interfaces de usuário e compiladores (SCHMIDT, 1997).*
> 2. *Frameworks de integração de middleware. Trata-se de um conjunto de normas e classes de objetos associados que oferecem suporte a componentes de comunicação e troca de informações. Exemplos desse tipo de framework incluem o .NET da Microsoft e o Enterprise Java Beans (EJB). Esses frameworks fornecem suporte para modelos de componentes padronizados.*
> 3. *Frameworks de aplicações corporativas. Esses estão relacionados com domínios de aplicação específicos, como telecomunicações ou sistemas financeiros (BAUMER et al., 1997). Eles incorporam conhecimentos sobre domínios de aplicações e apoiam o desenvolvimento de aplicações de usuário final.* (SOMMERVILLE, 2011, p.301)

A abordagem de reuso de software traz o benefício de reduzir diretamente o tempo de projeto e, consequentemente, os custos deste projeto; porém é importante observar, que dependendo dos requisitos, se existir uma necessidade de especialização ou adequação específicas do projeto, o reuso pode comprometer o desempenho e até a funcionalidade da aplicação; sendo necessária uma avaliação caso a caso das vantagens e desvantagens deste tipo de abordagem em cada projeto.

Para a avaliação dos ganhos com o reuso, em geral, verifica-se o tempo e/ou custo de desenvolvimento. Usaremos como referência neste trabalho a quantidade de linhas de código de cada abordagem (com e sem reuso), considerando uma proporcionalidade entre o tamanho da aplicação e seu tempo de desenvolvimento.

3. Proposta de implementação

É proposta a implementação de 3 objetos, um para a utilização da estrutura de dados Lista Ligada e outros dois, herdeiros do objeto lista, Pilha e Fila, que devem reutilizar os métodos e variáveis da Lista ligada para implementar a características intrinca de cada estrutura de dados; Os algoritmos devem ser avaliados pela quantidade de linhas de código e pelo tempo de execução de seus principais métodos, inserir e remover, na manipulação da estrutura de dados de 0 a 100.000.000 de manipulação de dados, com granularidade de 1.000.000 interações. Para a comparação será utilizado os resultados do PEL_216_Relatorio_1_Cristiano_Moreira que contém dados da implementação dos códigos de Fila e Pilha sem a utilização de herança.

Estrutura de dados Lista Ligada

O Algoritmo da estrutura de dados Lista ligada será implementado via Classe, com métodos para Inserir no início, responsável por enviar a informação que se deseja inserir no início da estrutura de dados; Inserir no final, responsável por enviar a informação que se deseja inserir no final da estrutura de dados; Remover do início, responsável por retirar a informação armazenada no inicio da lista; Remover do final, responsável por retirar a informação armazenada no final da lista Estado, responsável por informar se a estrutura de dados está vazia, quando não existem dados armazenados na Pilha ou normal, quando a estrutura de dados já contém dados e está pronta para receber mais informações; e um métodos adicionais, posterior; responsável por mover na estrutura de dados para a posição posterior à que está apontada a unidade de memória; anterior responsável por mover na

estrutura de dados para a posição anterior à que está apontada a unidade de memória; PegaValor, responsável por mostrar a informação contida na posição de memória apontada na estrutura de dados; e EncontraValor, responsável por encontrar se um valor passado como parâmetro para este método se encontra na estrutura de dados.

Será utilizado um objeto base unitMem, que representa a unidade mínima de memória deste algoritmo, contendo a informação armazenada na variável Conteúdo e dois ponteiros, anterior e posterior, que informam a ligação dos elementos da lista.

O Objeto unitMem

unitMem
+ Conteudo: interger + anterior: class unitMem + posterior: class unitMem
<<constructor>> unitMem()

O Objeto Lista

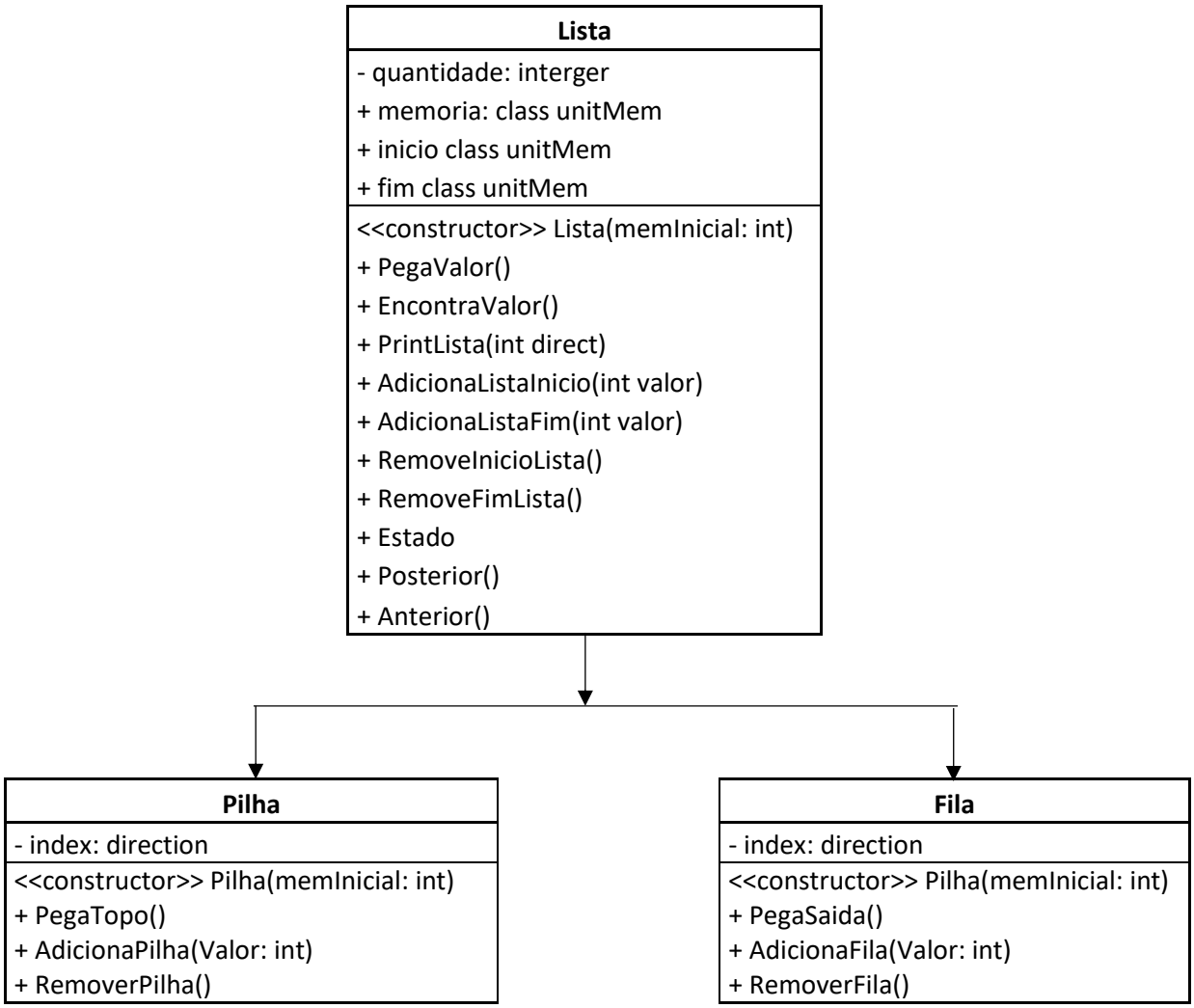
Lista
- quantidade: interger + memoria: class unitMem + inicio class unitMem + fim class unitMem
<<constructor>> Lista(memInicial: int) + PegaValor() + EncontraValor() + PrintLista(int direct) + AdicionaListaInicio(int valor) + AdicionaListaFim(int valor) + RemoveInicioLista() + RemoveFimLista() + Estado + Posterior() + Anterior()

Pseudocódigo:

```
Estado(S)
        Se S.quantidade==0
                Retorna vazio
        Se não
                retorna normal.

AdicionaListaInicio(S,x)
        x.posterior=S.inicio
        Se S.inicio != NULL
                S.inicio.anterior=x
        S.inicio=X
        x.anterior=NULL
        S.quantidade= S.quantidade+1

AdicionaListaFim(S,x)
        x.anterior=S.fim
        Se S.fim != NULL
                S.fim.posterior=x
        S.fim=X
        x.posterior=NULL
        S.quantidade= S.quantidade+1

RemoverInicioLista(S)
        Se S.inicio.anterior !=NULL
                S.inicio=S.inicio.anterior
        Se não
                S.inicio= NULL

RemoverFimLista(S)
        Se S.fim.posterior !=NULL
                S.fim=S.fim.posterior
        Se não
                S.fim= NULL

pegaValor(S)
        retorna S.memoria.Conteudo

Anterior(S)
        S.memoria= S.memoria.anterior

Posterior(S)
        S.memoria= S.memoria.posterior
```

Herança da estrutura de dados Lista para formação da Pilha e Fila

Os Algoritmos Pilha e Fila serão implementados via Classe com herança da classe lista, com métodos para Inserir, Remover, Estado e Pegar o dado, herdados com a interação da lista ligada.



Estrutura de dados Pilha

O Algoritmo da estrutura de dados Pilha será implementado via Classe com herança da classe lista, com métodos para Inserir, responsável por enviar a informação que se deseja inserir na estrutura de dados; Remover, responsável por retirar a informação armazenada no topo da pilha; Estado, responsável por informar

se a estrutura de dados está vazia, quando não existem dados armazenados na Pilha, , ou normal, quando a estrutura de dados já contém dados e está pronta para receber mais informações; e um método adicional, PegaTopo, responsável por mostrar a informação contida no topo da estrutura de dados.

Pseudocódigo:

```
Inserir(S,x)
        S.AdicionaListaInicio(valor);

Remover(S)
        S.RemoverInicioLista();

PegaTopo()
        S.memoria=S.inicio;
        S.PegaValor();
```

Estrutura de dados Fila

O Algoritmo da estrutura de dados Fila será implementado via Classe, com herança da classe lista, com métodos para Inserir, responsável por enviar a informação que se deseja inserir na estrutura de dados; Remover, responsável por retirar a informação armazenada no final da fila; Estado, responsável por informar se a estrutura de dados está vazia, quando não existem dados armazenados na fila, , ou normal, quando a estrutura de dados já contém dados e está pronta para receber mais informações; e um método adicional, PegaSaida, responsável por mostrar a informação contida na Saida da estrutura de dados.

Pseudocódigo:

```
Inserir(S,x)
        S.AdicionaListaFim(valor);

Remover(S)
        S.RemoverInicioLista();
PegaTopo()
        S.memoria=S.inicio;
        S.PegaValor();
```

As estruturas de dados Fila e Pilha não são estruturas eficientes para localizar informações dentro da base de dados e também não são eficientes nas operações de mínimo e máximo, já que não se propõem a ordenar os dados pelo seu valor e sim pela sua posição de entrada na estrutura.

4. Experimentação e Resultados

Tempo de Execução:

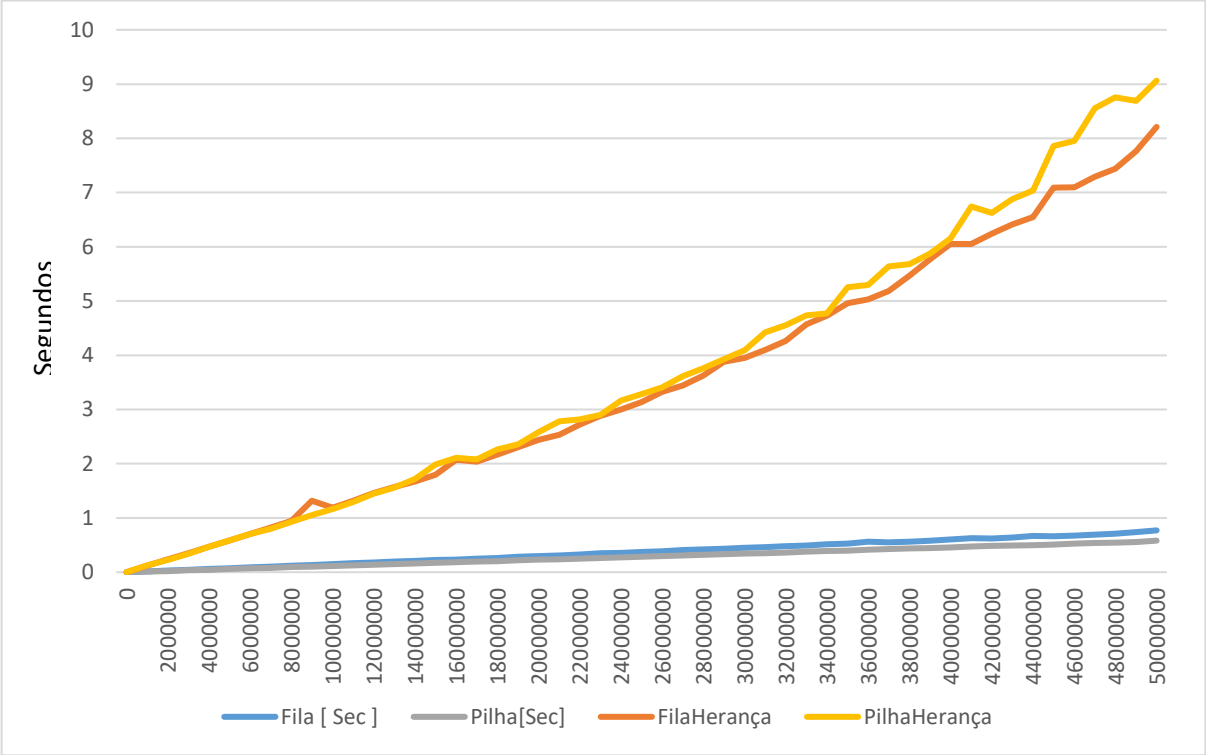
A interação com a estrutura de dados de Lista ou Fila, dos algoritmos especializados sem uso de herança, mostrou o tempo de execução, nas rotinas de inserir e remover dados, uma resposta linear O(1), e seu consumo de tempo para n interações estará em O(n), já as interações com a estrutura de dados com Herança e uso dinâmico da alocação de memória mostram um tempo de execução polinomial e sempre superior em quase 10 vezes o tempo comparados aos casos dos algoritmos especializados.

	Fila [Sec]		Pilha[Sec]	
n interações	Fila [Sec]	FilaHerança	Pilha[Sec]	PilhaHerança
0	0	0	0	0
1000000	0.015	0.117	0.014	0.126
2000000	0.029	0.236	0.022	0.223
3000000	0.045	0.348	0.034	0.339
4000000	0.058	0.47	0.045	0.47
5000000	0.073	0.587	0.059	0.588
6000000	0.092	0.701	0.071	0.702
7000000	0.103	0.822	0.08	0.8
8000000	0.117	0.948	0.096	0.929
9000000	0.133	1.315	0.101	1.054
10000000	0.15	1.187	0.113	1.163
11000000	0.165	1.316	0.125	1.293
12000000	0.177	1.459	0.136	1.449
13000000	0.194	1.57	0.151	1.559
14000000	0.207	1.67	0.159	1.717
15000000	0.224	1.796	0.172	1.986
16000000	0.234	2.072	0.182	2.108
17000000	0.251	2.038	0.196	2.078

18000000	0.264	2.17	0.205	2.261
19000000	0.286	2.306	0.217	2.358
20000000	0.295	2.442	0.234	2.582
21000000	0.311	2.533	0.239	2.779
22000000	0.328	2.722	0.251	2.817
23000000	0.349	2.882	0.264	2.897
24000000	0.358	2.998	0.274	3.165
25000000	0.372	3.137	0.284	3.286
26000000	0.387	3.327	0.299	3.409
27000000	0.406	3.441	0.309	3.613
28000000	0.421	3.628	0.318	3.759
29000000	0.435	3.882	0.33	3.929
30000000	0.45	3.952	0.342	4.09
31000000	0.459	4.099	0.349	4.425
32000000	0.478	4.261	0.364	4.554
33000000	0.49	4.568	0.38	4.736
34000000	0.515	4.729	0.389	4.77
35000000	0.525	4.962	0.399	5.256
36000000	0.56	5.031	0.415	5.295
37000000	0.553	5.186	0.424	5.636
38000000	0.563	5.466	0.436	5.68
39000000	0.579	5.767	0.444	5.875
40000000	0.605	6.054	0.456	6.153
41000000	0.628	6.05	0.472	6.745
42000000	0.621	6.238	0.487	6.622
43000000	0.641	6.409	0.492	6.879
44000000	0.668	6.546	0.5	7.038
45000000	0.664	7.093	0.51	7.856
46000000	0.675	7.095	0.528	7.952
47000000	0.694	7.293	0.537	8.553
48000000	0.712	7.44	0.547	8.758
49000000	0.738	7.762	0.556	8.692
50000000	0.771	8.21	0.582	9.063

Linhas de Código:

Pilha	Pilha Herança	Fila	Fila Herança
62	35	88	35



5. Trabalhos Correlatos

VUILLEMIN, J. Manipulating Priority Queues. Communications of the ACM, Volume 21, Number 4, p.309-315, April 1978

JONES, D. W. An Empirical Comparison of Priority-Queue and Event-Set Implementaitons. Communications of the ACM, Volume 29, Number 4, p.300-311, April 1986

JIANG,Y.F.; CHEN,L.H.;CHIENA.F, A UNISON framework for knowledge management of university–industry collaboration and an illustration, Computers & Industrial Engineering Volume 129, March 2019, Pages 31-43

6. Conclusão

A estrutura de dados Pilha e Fila desenvolvida por herança dos métodos da estrutura de dados Lista Ligada mostra um consumo de tempo polinominal, e como sabemos uma administração mais adequada do uso da memória; e tempo de desenvolvimento e linhas de código menores para a criação da estrutura de dados Fila e Pilha, porém, é observado que o tempo de processamento, quando utilizado a estrutura herdada é bem superior que nos sistemas especializados, o que nos faz concluir que, dependendo da especificação e necessidades de um projeto, a utilização de heranças para redução do tempo e custo de desenvolvimento não é uma resposta direta, sendo sempre necessária uma análise mais profunda dos requisitos do projeto.

7. Referências bibliográficas

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Perason, 2011, 9a ed.