CENTRO UNIVERSITÁRIO FEI

RAFAEL THOMAS PONTES SALGADO

RELATÓRIO 02: Pilhas e Filas com Lista Ligada

São Bernardo do Campo
2017

RESUMO

Nesse relatório buscamos expor a implementação de duas estruturas de dados, Fila e Pilha, utilizando como base a estrutura de Lista duplamente ligada mostrando como são conceitualmente, para quais problemas são utilizadas e exemplos de implementação em linguagem C++.

Palavras-chave: Pilha. Fila. Estrutura de Dados. Algoritmo. C++

SUMÁRIO

1 INTRODUÇÃO

Durante nossa vida nos deparamos com inúmeros problemas diários onde uma organização deve ser dada aos elementos para que possamos resolver os problemas de forma eficaz. Quando vamos ao banco pagar uma conta ou quando simplesmente vamos organizar as cartas de um baralho, certas estruturas naturais estão sempre presentes para nos auxiliar e as mais comumente encontradas são as estruturas de pilha e fila que serão o objetivo desse relatório.

1.1 MOTIVAÇÃO

Por serem estruturas de dados utilizadas de forma recorrente em diversos algoritmos, esse trabalho tem a motivação de estudar as estruturas de dados de pilha, fila e lista duplamente ligada para seu maior entendimento e ajuda na escolha das mesmas em futuras implementações.

1.2 OBJETIVO

Esse relatório tem como objetivo a implementação das estruturas de dados de Pilha, Fila e Lista Duplamente Ligada utilizando a linguagem orientada a objeto C++, verificando sua complexidade de implementação e exemplificando usos das mesmas.

1.3 METODOLOGIA

Para a implementação das estruturas de dados de pilha, fila e lista duplamente ligada, foi utilizada a linguagem orientada a objeto C++ visto a fácil representação das estruturas como objetos. Como IDE de desenvolvimento foi utilizado o programa Xcode, versão 8.3.3 (8E3004b).

Tanto os códigos desenvolvidos como os arquivos fonte .tex podem ser consultados na seguinte página do Github `https://github.com/thomassalgado/computacao-cientifica`

2 TEORIA

A seguir iremos descrever as estruturas de pilha, fila e lista duplamente ligada, bem como suas propriedades e métodos

2.1 PILHA

Pilha é uma estrutura de dados onde um conjunto de elementos é organizado da forma **último a entrar, primeiro a sair**, possuindo as seguintes operações:

a) ***Inserir*** ou ***Push***

b) ***Remover*** ou ***Pop***

c) ***Topo*** ou ***Top***

O método ***Inserir*** adiciona um novo elemento a estrutura de dados fazendo com que o mesmo seja representado como estando no topo da pilha

O metodo ***Remover*** retira o elemento atualmente no top da pilha da estrutura de dados, o elemento subsequente passa a ser o novo topo da pilha ou a mesma ficará vazia.

O metodo ***Topo*** informa qual é o elemento que esta atualmente no topo da pilha, sem removê-lo.

2.2 FILA

Fila é uma estrutura de dados onde um conjunto de elementos é organizado da forma **primeiro a entrar, primeiro a sair**, possuindo as seguintes operações:

a) ***Inserir*** ou ***Enqueue***

b) ***Remover*** ou ***Dequeue***

O método ***Inserir*** adiciona um novo elemento a estrutura de dados fazendo com que o mesmo seja representado como estando no fim da fila. Caso não exista outro elemento na fila, o mesmo é marcado tambem como estando no início da fila e portando sendo o próximo a ser removido pelo método ***Remover***.

O metodo ***Remover*** retira o elemento atualmente no início da fila, o elemento subsequente passa a ser o novo início e caso não existam novos elementos a mesma ficará vazia.

2.3 LISTA DUPLAMENTE LIGADA

A Lista Duplamente Ligada é uma estrutura de dados onde cada um dos seus elementos, alem do valor armazenado recebem dois ponteiros:

a) Um ponteiro para o próximo elemento

b) Um ponteiro para o elemento anterior

Isso permite que não seja usada uma estrutura de array para armazenar os dados e garante a ordem dos elementos inseridos

Uma lista duplamente ligada possui os seguintes métodos:

O método ***Inserir*** adiciona um novo elemento ao final da Lista

O metodo ***Remover*** retira o elemento atualmente no final da Lista.

O método ***InserirInicio*** adiciona um novo elemento ao início da Lista

O metodo ***RemoverInicio*** retira o elemento atualmente no inicio da Lista.

2.4 HERANÇA E ORIENTAÇÃO A OBJETOS

Um dos conceitos abordados neste relatório é o conceito de herança, aplicado a orientação a objetos.

Herança, na orientação a objetos, representa uma relação onde determinada classe, ao ser declarada como herdeira de outra, herda todos os seus atributos e métodos, ou seja, ela passa a se comportar como a classe pai, satisfazendo a relação ***É um***, por exemplo:

A classe ***Animal***, é uma classe que representa todos os seres vivos, como padrão de todos os seres vivos, a classe Animal possuí os seguintes métodos:

a) ***Respirar***

b) ***Comer***

c) ***Excretar***

A classe ***Cachorro*** por sua vez, é uma especialização da classe ***Animal*** e por isso precisa dos métodos implementados na classe ***Animal*** além do método ***AbanarRabo*** que é uma especialidade da classe ***Cachorro***.

Para construir essa relação entre Animal e Cachorro, dizemos que a classe Cachorro extende da classe Animal, herdando assim seus métodos e atributos, fato que em linguagem C++ é declarado da seguinte forma

class Cachorro : public Animal

Tal conceito foi utilizado para a construção das estruturas de dados mencionadas nesse relatório, sendo ***Lista*** a classe pai e ***Fila*** e ***Pilha*** as classes filhas.

3 TRABALHOS CORRELATOS

Por se tratarem de elementos bem difundidos no ramo científico e de implementação simples, não foram encontrados trabalhos ou pesquisas relacionadas diretamente ao estudo das estruturas de pilha e fila, entretanto as mesmas são base base a implementação de uma enorme quantidade de algoritmos.

4 PROPOSTA

4.1 IMPLEMENTAÇÃO DE LISTA

Para a representação dos elementos armazenados na classe Lista, a classe de nome ***No*** foi criada, com sua estrutura definida no arquivo ***No.hpp*** e sua implementação definida no arquivo ***No.cpp***

No.hpp

```
1  //
2  //  No.hpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #pragma once
10
11 #ifndef No_hpp
12 #define No_hpp
13
14 #include <stdio.h>
15 #include <string>
16
17 #endif /* No_hpp */
18
19 using namespace std;
20
21 /*
22  * Classe representando os dados que serao armazenados na lista
23  */
24 class No {
25 private:
26     /*
27      * propriedade repesentando o dado armazenado
28      */
29     string dado;
30     No *anterior;
31     No *posterior;
32 public:
33     /*
34      * Construtor com parametro
35      */
```

```
36      No(string valor);
37      /*
38       * Get do valor armazenado
39       */
40      string getDado();
41      /*
42       * Atribui um novo valor para o ponteiro do óN Anterior
43       */
44      void setAnterior(No *novoAnterior);
45      /*
46       * Atribui um novo valor para o ponteiro do óN Posterior
47       */
48      void setPosterior(No *novoPosterior);
49      /*
50       * Retorna o novo valor para o ponteiro do óN Anterior
51       */
52      No * getAnterior();
53      /*
54       * Retorna o valor para o ponteiro do óN Posterior
55       */
56      No * getPosterior();
57  };
```

No.cpp

```
1   //
2   //  No.cpp
3   //  computacao.cientifica.algoritmos
4   //
5   //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6   //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7   //
8
9   #include "No.hpp"
10
11  No::No(string valor){
12      dado = valor;
13      anterior = NULL;
14      posterior = NULL;
15  }
16
17  string No::getDado(){
18      return dado;
19  }
```

```
20
21 void No::setAnterior(No *novoAnterior){
22     anterior = novoAnterior;
23 }
24
25 void No::setPosterior(No *novoPosterior){
26     posterior = novoPosterior;
27 }
28
29 No * No::getAnterior(){
30     return anterior;
31 }
32
33 No * No::getPosterior(){
34     return posterior;
35 }
```

A classe No possui um atributo de nome ***dado*** responsável por armazenar o valor desejado e um par de ponteiros ***anterior*** e ***posterior*** responsáveis por indicar qual é o elemento predecessor e o elemento sucessor ao Nó na estrutura de lista.

Para a implementação da estrutura de Lista na linguagem C++, a classe de nome Lista foi criada, com sua estrutura definida no arquivo ***Lista.hpp*** e sua implementação definida no arquivo ***Lista.cpp***.

A estrutura de dados não possuí um array para armazenamento dos elementos, a medida que um novo nó é solicitado, o mesmo é criado com o comando ***new*** e esse recebe um ponteiro para o elemento anterior e um para o elemento posterior caso seja necessário.

Lista.hpp

```
1  //
2  //  Lista.hpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #pragma once
10
11 #ifndef Lista_hpp
12 #define Lista_hpp
13
14 #include <stdio.h>
15 #include <string>
```

```
16 #include <iostream>
17 #include "No.hpp"
18
19 #endif /* Lista_hpp */
20
21 using namespace std;
22 /*
23  * Estrutura de dados Lista duplamente ligada
24  */
25 class Lista {
26 private:
27     /*
28      * Ponteiro para o primeiro elemento da lista
29      */
30     No *inicio;
31     /*
32      * Ponteiro para o ultimo elemento da lista
33      */
34     No *fim;
35     /*
36      * Contador de elementos
37      */
38     long elementos;
39 public:
40     /*
41      * Insere um novo elemento no fim da lista
42      */
43     void inserir(string dado);
44     /*
45      * Insere um novo elemento no inicio da lista
46      */
47     void inserirInicio(string dado);
48     /*
49      * Remove um elemento do fim da lista
50      */
51     No * remover();
52     /*
53      * Remove um elemento do inicio da lista
54      */
55     No * removerInicio();
56     /*
57      * Retorna o primeiro elemento da lista
58      */
59     No * getInicio();
60     /*
61      * Retorna o ultimo elemento da lista
```

```
62      */
63     No * getFim();
64     /*
65      * Retorna o tamanho da lista
66      */
67     long size();
68     /*
69      * Sobrecarga para ãimpresso da lista
70      */
71     friend ostream& operator<<(ostream& os, const Lista& dt);
72 };
```

Lista.cpp

```
1  //
2  //  Lista.cpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #include "Lista.hpp"
10
11 void Lista::inserir(string dado){
12     if (inicio == NULL) {
13         inicio = new No(dado);
14         fim = inicio;
15         elementos++;
16     } else {
17         No *novoNo = new No(dado);
18         fim->setPosterior(novoNo);
19         novoNo->setAnterior(fim);
20         fim = novoNo;
21         elementos++;
22     }
23 }
24
25 void Lista::inserirInicio(string dado){
26     if (inicio == NULL) {
27         inicio = new No(dado);
28         fim = inicio;
29         elementos++;
30     } else {
```

```
31          No *novoNo = new No(dado);
32          novoNo->setPosterior(inicio);
33          inicio->setAnterior(novoNo);
34          inicio = novoNo;
35          elementos++;
36      }
37  }
38
39  No * Lista::remover(){
40      if(fim != NULL){
41          No *retorno = fim;
42          if(fim->getAnterior() != NULL){
43              fim->getAnterior()->setPosterior(NULL);
44              fim = fim->getAnterior();
45          } else {
46              fim = NULL;
47              inicio = NULL;
48          }
49          elementos--;
50          retorno->setAnterior(NULL);
51          retorno->setPosterior(NULL);
52          return retorno;
53      } else {
54          return NULL;
55      }
56  }
57
58  No * Lista::removerInicio(){
59      if(inicio != NULL){
60          No *retorno = inicio;
61          if(inicio->getPosterior() != NULL){
62              inicio->getPosterior()->setAnterior(NULL);
63              inicio = inicio->getPosterior();
64          } else {
65              fim = NULL;
66              inicio = NULL;
67          }
68          elementos--;
69          retorno->setAnterior(NULL);
70          retorno->setPosterior(NULL);
71          return retorno;
72      } else {
73          return NULL;
74      }
75  }
76
```

```
77 No * Lista::getInicio(){
78     return inicio;
79 }
80
81 No * Lista::getFim(){
82     return fim;
83 }
84
85 long Lista::size(){
86     return elementos;
87 }
88
89 ostream& operator<<(ostream& os, const Lista& list)
90 {
91     No *leitura = list.inicio;
92     while(leitura != NULL){
93         os << leitura->getDado();
94         leitura = leitura->getPosterior();
95     }
96     return os;
97 }
```

A classe Lista possui os seguintes atributos:

a) ***Inicio***: ponteiro para o primeiro elemento da Lista

b) ***fim***: ponteiro para o último elemento da lista

c) ***elementos***: contador de elementos presentes na Lista

E os seguintes métodos:

a) ***inserir***: insere um novo elemento no fim da Lista

b) ***inserirInicio***: insere um novo elemento no início da Lista

c) ***remover***: remove o último elemento da Lista

d) ***removerInicio***: remove o primeiro elemento da Lista

e) ***getInicio***: retorna o primeiro elemento da Lista sem removê-lo

f) ***getFim***: retorna o último elemento da Lista sem removê-lo

g) ***size***

Para uma melhor formatação da saída da Lista para a console, o operador « foi sobreescrito.

4.2 IMPLEMENTAÇÃO DE PILHA

Para a implementação da estrutura de Pilha na linguagem C++, a classe de nome PilhaComLista foi criada, com sua estrutura definida no arquivo ***PilhaComLista.hpp*** e sua implementação definida no arquivo ***PilhaComLista.cpp***

Utilizando-se da propriedade de herança, a classe PilhaComLista, herdou métodos e atributos da classe Lista, facilitando sua implementação.

PilhaComLista.hpp

```
1  //
2  //  PilhaComLista.hpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #pragma once
10
11 #ifndef PilhaComLista_hpp
12 #define PilhaComLista_hpp
13
14 #include <stdio.h>
15 #include <string>
16 #include "Lista.hpp"
17
18 #endif /* PilhaComLista_hpp */
19
20 using namespace std;
21
22 /*
23  * Pilha implementada com lista duplamente ligada
24  */
25 class PilhaComLista : public Lista {
26 public:
27     /*
28      * Empilha um elemento
29      */
30     void empilha(string entrada);
31     /*
32      * Desempilha um elemento
33      */
34     No * desempilha();
35     /*
```

```
36      * Retorna o topo da pilha sem desempilha-lo
37      */
38     No * topo();
39 };
```

PilhaComLista.cpp

```
1  //
2  //  PilhaComLista.cpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #include "PilhaComLista.hpp"
10
11 void PilhaComLista::empilha(string entrada){
12     Lista::inserir(entrada);
13 }
14
15 No * PilhaComLista::desempilha(){
16     return Lista::remover();
17 }
18
19 No * PilhaComLista::topo(){
20     return Lista::getFim();
21 }
```

O método ***empilha*** recebe como parâmetro uma string e chama o método ***inserir*** da classe pai.

O método ***desempilha*** não recebe parâmetros e retorna a chamada do método ***remover*** da classe pai.

O método ***topo*** não recebe parâmetros e retorna o último elemento da Lista retornado pelo método ***getFim*** da classe pai.

O método para retorno do tamanho da pilha passou a ser o metodo ***size*** herdado da classe pai.

4.3 IMPLEMENTAÇÃO DE FILA

Para a implementação da estrutura de Fila na linguagem C++, a classe de nome FilaComLista foi criada, com sua estrutura definida no arquivo ***FilaComLista.hpp*** e sua implementação definida no arquivo ***FilaComLista.cpp***

Utilizando-se da propriedade de herança, a classe FilaComLista, herdou métodos e atributos da classe Lista, facilitando sua implementação.

FilaComLista.hpp

```
1  //
2  //  FilaComLista.hpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #pragma once
10
11 #ifndef FilaComLista_hpp
12 #define FilaComLista_hpp
13
14 #include <stdio.h>
15 #include "Lista.hpp"
16
17
18 #endif /* FilaComLista_hpp */
19
20 /*
21  * Estrutura de dados de fila implementada com lista duplamente ligada
22  */
23 class FilaComLista : public Lista {
24 public:
25     /*
26      * Insere um elemento no fim da fila
27      */
28     void enfileira(string entrada);
29     /*
30      * remove um elemento do inicio da fila
31      */
32     No * desenfileira();
33 };
```

FilaComLista.cpp

```
1  //
2  //  FilaComLista.cpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Pontes Salgado on 18/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #include "FilaComLista.hpp"
10
11 No * FilaComLista::desenfileira(){
12     return Lista::removerInicio();
13 }
14
15 void FilaComLista::enfileira(string entrada){
16     Lista::inserir(entrada);
17 }
```

O método ***enfileira*** recebe como parâmetro uma string e chama o método ***enfileira*** da classe pai.

O método ***desenfileira*** não recebe parâmetros chama o método ***removeInicio*** da classe pai.

O método para retorno do tamanho da pilha passou a ser o metodo ***size*** herdado da classe pai.

5 RESULTADOS

Para averiguação da correta implementação das estruturas de pilha e fila, o seguinte arquivo de teste main.cpp foi implementado, oferecendo um menu iterativo para efetuar as possíveis operações dentro das estruturas de dados.

main.cpp

```
1  //
2  //  main.cpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Salgado on 07/06/17.
6  //  Copyright © 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9
10 #ifndef main_cpp
11
12 #include <iostream>
13 #include "Pilha.hpp"
14 #include "Fila.hpp"
15 #include "PilhaComLista.hpp"
16 #include "FilaComLista.hpp"
17
18 #endif /* main_cpp */
19
20
21 using namespace std;
22
23 int main(int argc, const char * argv[]) {
24
25     string entrada;
26     PilhaComLista pilha;
27     FilaComLista fila;
28
29     while(true){
30         cout << "Digite:\n";
31         cout << "1 para Pilha\n";
32         cout << "2 para Fila\n";
33         cout << "0 para Sair\n";
34         getline (std::cin,entrada);
35         if(entrada.compare("0") == 0){
36             cout << "Ate breve\n";
37             return 0;
```

```
38          }else if(entrada.compare("1") == 0){
39              cout << "Operacoes em Pilha\n";
40              entrada = "";
41              cout << "Digite:\n";
42              cout << "1 para Inserir\n";
43              cout << "2 para Remover\n";
44              cout << "3 para tamanho da fila\n";
45              getline (std::cin,entrada);
46              if(entrada.compare("1") == 0){
47                  cout << "Digite o valor a ser inserido\n";
48                  entrada = "";
49                  getline (std::cin,entrada);
50                  pilha.empilha(entrada);
51              } else if(entrada.compare("2") == 0){
52                  pilha.desempilha();
53              } else if(entrada.compare("3") == 0) {
54                  cout << "A estrutura possui " << pilha.size() << " ↩
                        elementos\n";
55              } else {
56                 cout << "Opcao invalida\n";
57              }
58          } else if (entrada.compare("2") == 0){
59              cout << "Operacoes em Fila\n";
60              entrada = "";
61              cout << "Digite:\n";
62              cout << "1 para Inserir\n";
63              cout << "2 para Remover\n";
64              cout << "3 para tamanho da fila\n";
65              getline (std::cin,entrada);
66              if(entrada.compare("1") == 0){
67                  cout << "Digite o valor a ser inserido\n";
68                  entrada = "";
69                  getline (std::cin,entrada);
70                  fila.enfileira(entrada);
71              } else if(entrada.compare("2") == 0){
72                  fila.desenfileira();
73              } else if(entrada.compare("3") == 0) {
74                  cout << "A estrutura possui " << fila.size() << " ↩
                        elementos\n";
75              } else {
76                  cout << "Opcao invalida\n";
77              }
78
79          } else {
80              cout << "Opcao invalida\n";
81          }
```

```
82      }
83
84      return 0;
85  }
```

O menu oferece as seguintes opções:

a) 1 para operações em ***Pilha***

b) 2 para operações em ***Fila***

c) 0 para abandonar o programa

Escolhendo ***Pilha*** ou ***Fila***, as seguintes opções estão disponíveis

a) 1 para ***Inserir***

b) 2 para ***Remover***

c) 3 para ***Quantidade de elementos***

Com base no programa de testes foi possível verificar a implementação correta dos algoritmos e sua eficiência para armazenamento de dados.

6 CONCLUSÃO

Pode concluir que apesar de simples, as estruturas de dados de Pilha e Fila são ferramentas poderosas para o desenvolvimento de algoritmos pois a sua manipulação de dados fornece uma base para o desenvolvimento de outras estruturas de dados mais complexas e algoritmos que necessitam armazenar dados de forma simples.

Utilizando-se a abordagem de herança, além de facilitar a implementação de novas estruturas de dados que tenham Lista como base, economizou-se uma grande quantidade de código devido a reutilização de métodos codificados na classe pai.

REFERÊNCIAS

[C17] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms*. McGraw-Hill, 1990.

[1] Paul J. Deitel, Harvey M. Deitel, C++ How to program, Publish hall, 2008, Chapter 12.