CENTRO UNIVERSITÁRIO FEI

RAFAEL THOMAS PONTES SALGADO

RELATÓRIO 01: Pilhas e Filas

São Bernardo do Campo
2017

RESUMO

Nesse relatório buscamos expor a implementação de duas estruturas de dados, Fila e Pilha, 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++

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 Introdução 7
1.1 Motivação 7
1.2 Objetivo 7
1.3 Metodologia 7
2 Teoria 8
2.1 Pilha 8
2.2 Fila 8
3 Trabalhos correlatos 9
4 Proposta 10
4.1 Implementação de Pilha 10
4.2 Implementação de Fila 12
5 Resultados 17
6 Conclusão 18
ÍNDICE 18

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 e fila 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 e Fila 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 e fila, 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 e fila, 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.

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 PILHA

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

Os elementos da estrutura de dados serão inseridos no vetor de nome ***memória*** que foi definido com o tamanho de 1000 pela constante ***SIZE***

Pilha.hpp

```
1  //
2  //  Pilha.hpp
3  //  computacao.cientifica.algoritmos.pilha
4  //
5  //  Created by Rafael Thomas Salgado on 07/06/17.
6  //  Copyright 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #ifndef Pilha_hpp
10 #define Pilha_hpp
11 #define SIZE 1000
12
13
14 #include <stdio.h>
15 #include <string>
16 #include <iostream>
17
18 #endif /* Pilha_hpp */
19
20 using namespace std;
21
22 class Pilha {
23 private:
24     string memoria[SIZE];
25     int posicaoTopo;
26
27 public:
28     Pilha();
29     void empilha(string entrada);
30     string desempilha();
31     string topo();
32     int qtdElementos();
```

```
};
```

Pilha.cpp

```
//
//  Pilha.cpp
//  computacao.cientifica.algoritmos.pilha
//
//  Created by Rafael Thomas Salgado on 07/06/17.
//  Copyright 2017 Rafael Thomas Salgado. All rights reserved.
//

#include "Pilha.hpp"

Pilha::Pilha(){
    posicaoTopo = 0;
}

void Pilha::empilha(string entrada){
    if(posicaoTopo < SIZE){
        memoria[posicaoTopo] = entrada;
        posicaoTopo++;
        cout << "Elemento \"" << entrada << "\" empilhado com sucesso\↩
            n";
    } else {
        cout << "A Pilha esta cheia\n";
    }
    return;
}

string Pilha::desempilha(){

    if(posicaoTopo == 0){
        cout << "A Pilha esta vazia\n";
        return "";
    } else {
        posicaoTopo--;
        string aux = memoria[posicaoTopo];
        memoria[posicaoTopo]="";
        return aux;
    }
}

string Pilha::topo(){
```

```
40      if(posicaoTopo == 0){
41          cout << "A Pilha esta vazia\n";
42          return "";
43      } else {
44          return memoria[posicaoTopo - 1];
45      }
46  }
47
48  int Pilha::qtdElementos(){
49      return posicaoTopo;
50  }
```

A variável ***posicaoTopo*** indica a posição em que um novo elemento deverá inserido e tambem representa a quantidade de elementos inseridos na pilha

O método ***empilha*** recebe como parâmetro uma string e verifica se ainda existe espaço na pilha para a inserção de novos itens. caso exista, insere o novo elemento na pilha, retornando uma mensagem de sucesso, caso contrário, retorna uma mensagem de erro informando que a pilha esta cheia.

O método ***desempilha*** não recebe parâmetros e retorna o elemento que se encontra no topo da pilha, removendo-o da mesma. Caso a pilha esteja vazia, uma mensagem de erro é retornada.

O método ***topo*** não recebe parâmetros e retorna o elemento que se encontra no topo da pilha, sem removê-lo. Caso a pilha esteja vazia, uma mensagem de erro é retornada.

O método auxiliar ***qtdElementos*** informa a quantidade de elementos que estão na pilha no momento.

4.2 IMPLEMENTAÇÃO DE FILA

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

Os elementos da estrutura de dados serão inseridos no vetor de nome ***memória*** que foi definido com o tamanho de 1000 pela constante ***SIZE***

Fila.hpp

```
1 //
2 //  Fila.hpp
3 //  computacao.cientifica.algoritmos
4 //
5 //  Created by Rafael Thomas Salgado on 08/06/17.
6 //  Copyright 2017 Rafael Thomas Salgado. All rights reserved.
```

```
7  //
8
9  #ifndef Fila_hpp
10
11 #define Fila_hpp
12 #define SIZE 1000
13 #include <stdio.h>
14 #include <string>
15 #include <iostream>
16
17 #endif /* Fila_hpp */
18
19 using namespace std;
20
21 class Fila {
22 private:
23     string memoria[SIZE];
24     int inicio;
25     int fim;
26     int elementos;
27
28 public:
29     Fila();
30     string desenfileira();
31     void enfileira(string entrada);
32     int qtdElementos();
33 };
```

Fila.cpp

```
1  //
2  //  Fila.cpp
3  //  computacao.cientifica.algoritmos
4  //
5  //  Created by Rafael Thomas Salgado on 08/06/17.
6  //  Copyright 2017 Rafael Thomas Salgado. All rights reserved.
7  //
8
9  #include "Fila.hpp"
10
11 Fila::Fila(){
12     inicio = 0;
13     fim = 0;
14     elementos = 0;
```

```
15 }
16
17 void Fila::enfileira(string entrada){
18     if(fim < SIZE){
19         memoria[fim] = entrada;
20         fim++;
21         elementos++;
22         cout << "Elemento \"" << entrada << "\" empilhado com sucesso\↩
               n";
23     } else {
24         cout << "A Fila esta cheia\n";
25     }
26     return;
27 }
28
29 string Fila::desenfileira(){
30     if(inicio < SIZE && inicio < fim){
31         string aux = memoria[inicio];
32         memoria[inicio] = "";
33         inicio++;
34         elementos--;
35         return aux;
36     } else {
37         return "A Fila esta vazia\n";
38     }
39 }
40
41 int Fila::qtdElementos(){
42     return elementos;
43 }
```

A variável ***inicio*** indica a posição de inicio da fila, sendo a referencia para remoção do proximo elemento.

A variável ***fim*** indica a posição do próximo elemento que será inserido na fila.

A variável ***elementos*** funciona de forma auxiliar para indicar a quantidade de elementos que existem na fila.

O método ***enfileira*** recebe como parâmetro uma string e verifica se ainda existe espaço na fila para a inserção de novos itens. caso exista, insere o novo elemento na fila, incrementando a posição fim da fila e retornando uma mensagem de sucesso, caso contrário, retorna uma mensagem de erro informando que a fila esta cheia.

O método ***desenfileira*** não recebe parâmetros e verifica se a fila não esta vazia, analisando se os valores de inicio e fim são iguais (o que evidenciaria uma fila vazia), ou se o valor de fim não ultrapassa o tamanho total da fila definido em SIZE, retornando uma mensagem de erro em qualquer um dos casos anteriores. Caso nenhuma das condições anteriores seja validada, o

valor de correspondente a posição de fim é retornado, incrementando-se a posição de fim para a próxima casa do vetor.

O algoritmo implementado de fila leva em consideração como limite o tamanho total da fila definido em SIZE, caso esse valor seja ultrapassado, mesmo que existam posições disponíveis no array memória, as mesmas não serão utilizadas. Para sanar esse problemas, uma implementação de fila circular deverá ser feita, entretanto sua abordagem esta fora do escopo desse relatório.

5 RESULTADOS

Nunc molestie nunc ac lorem dictum, sit amet pl

6 CONCLUSÃO

Nunc molestie nunc ac lorem dictum, sit amet pl