Implementando Pilhas e Filas em C++

Danilo Sanchez Tuzita
(danilo_st@hotmail.com)

I. Resumo

Este é um trabalho que tem como objetivo implementar as estruturas de dados Pilha e Fila em *C++* e compara-la com as estruturas pré instaladas.

II. Introdução

Mesmo no mundo moderno, onde o *hardware* não é mais um gargalo expressivo, para certas aplicações, ainda é necessário que os *softwares* sejam eficientes. Nesse trabalho foi desenvolvido duas estruturas de dados que são mais eficientes do que as pré instaladas.

III. Teoria

Para o entendimento desse trabalho é necessário conhecimentos de algoritmos, ponteiros, classes, estruturas de dados e *templates*.

IV. Proposta e Implementação

Ambas as estruturas de dados implementadas são dinâmicas, ou seja, elas podem ter o tamanho que for necessário sem que seja especificado previamente um tamanho. Outra característica dessas estruturas é o uso de *Templates*, isso significa que pode ser guardado qualquer tipo de dado dentro dessas estruturas de dados. O diagrama de classes das estruturas pode ser observado na Fig. 1 e exemplos das estruturas nas Fig. 2 e 3

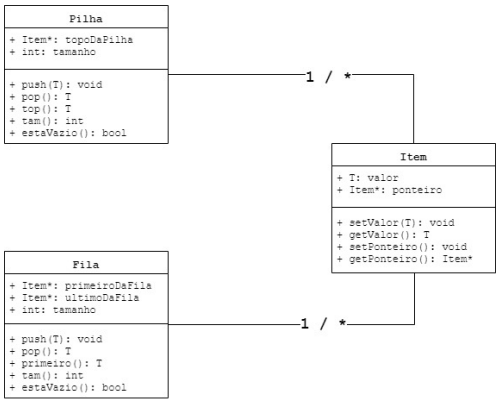

Fig. 1. Diagrama de classes das estruturas de dados deste trabalho

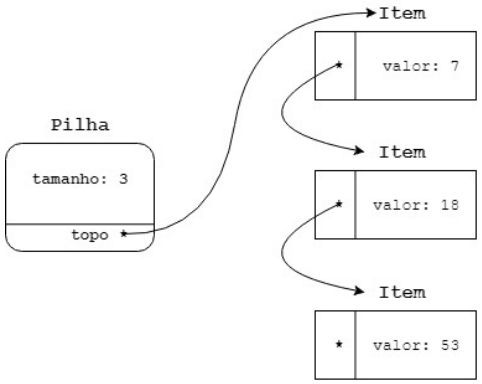

Fig. 2. Exemplo de uma pilha

A. Item

Um $Item$ é uma classe que guardará um valor (v) e um ponteiro (ptr) para outro $Item$. O ponteiro dessa classe é sempre inicializado apontando para o endereço de memória 0 ($nullptr$). Essa classe é usada em ambas as estruturas.

B. Pilha

A $Pilha$ é uma classe que guardará apenas dois valores: Um ponteiro para o topo da fila ($topo$) e seu tamanho. Na sua inicialização, o $topo$ apontará para $nullptr$.

Assim que for pedido para ser empilhado o primeiro $novoItem$, o $topo$ passará a apontar para o endereço do $novoItem$. Nas próximas iterações, antes do $topo$ apontar para o $novoItem$, faz-se que o ptr do novo $novoItem$, passe a apontar para o topo e depois disso o topo passa a apontar para o $novoItem$.

Caso seja pedido para desempilhar a pilha, o $topo$ passará a apontar para o ptr do $topo$ e será deletado o antigo topo.

C. Fila

A $Fila$ é uma classe que guardará três valores: Um ponteiro para o primeiro da fila ($primeiro$), um ponteiro para o último da fila ($ultimo$) e seu tamanho. Seus dois ponteiros são inicializados apontando para $nullptr$.

Quando o tamanho da fila é zero e é inserido o primeiro $novoItem$ na fila, é necessário que ambos os ponteiros $primeiro$ e $ultimo$, apontem para esse $novoItem$. No momento da inserção dos próximos itens, aponta-se o ptr do $ultimo$ para o $novoItem$ e o $ultimo$ passa a ser o $novoItem$.

Para desenfileirar um item, aponta-se o $primeiro$ para o ptr do $primeiro$ e deleta-se o antigo primeiro.

V. Experimentos

Para testar a eficiência das estruturas implementadas, foi feito um teste de estresse na estruturas de dados desenvolvidas neste trabalho que chamaremos de $Pilha^{42}$ e $Fila^{42}$ e nas estruturas pré implementadas do *C++* que chamaremos de $PilhaCPP$ e $FilaCPP$.

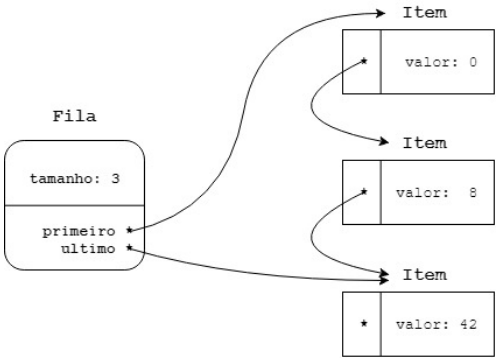

Fig. 3. Exemplo de uma fila

O teste de estresse foi executado da seguinte forma: Para cada estrutura foi incluídos um milhão de valores, removidos quinhentos mil, incluídos mil e removido tudo até a estrutura ficar vazia. Foi executado o teste 100 vezes para ambas as estruturas, como representado no algorítimo 1.

Algorithm 1 Avaliação de desempenho Pilha

repeat
  $stack = newPilha^{42}$
  for $i = 0; i < 1000000; i++$ **do**
    $stack.push(i)$
  end for
  for $i = 0; i < 500000; i++$ **do**
    $stack.pop()$
  end for
  for $i = 0; i < 1000; i++$ **do**
    $stack.push(i)$
  end for
  while $stack.size > 0$ **do**
    $stack.pop()$
  end while

  $stack = newPilhaCPP$
  for $i = 0; i < 1000000; i++$ **do**
    $stack.push(i)$
  end for
  for $i = 0; i < 500000; i++$ **do**
    $stack.pop()$
  end for
  for $i = 0; i < 1000; i++$ **do**
    $stack.push(i)$
  end for
  while $stack.size > 0$ **do**
    $stack.pop()$
  end while
until 100

VI. Resultados

Para os resultados foi calculado a quantidades de *ticks* do processador para completar uma rodada de testes. As tabelas I e II, indicam a média e o desvio padrão quantidade de *ticks* e o tempo levou para completar uma rodada de testes.

TABLE I
Resultados dos experimentos para a Pilha

Pilha				
	$Pilha^{42}$		$PilhaCPP$	
	Ticks	Tempo (s)	*Ticks*	Tempo (s)
Média	608.43	0.60843	1920.46	1.92046
Desvio Padrão	32.9273	0.032927301	94.325829	0.094325829

TABLE II
Resultados dos experimentos para a Fila

Fila				
	$Fila^{42}$		$FilaCPP$	
	Ticks	Tempo (s)	*Ticks*	Tempo (s)
Média	570.96	0.57096	1939.47	1.93947
Desvio Padrão	27.811	0.027811022	71.698084	0.071698084

As Figuras 4, 5, 6 e 7, indicam visualmente os resultados das tabelas I e II.

VII. Conclusão

De acordo com os resultados, o algorítimo implementado neste trabalho é consideravelmente mais rápido que o pré-instalado, cerca de 4 vezes mais rápido. É importante que o foco desse trabalho é velocidade, não foi pensado sobre a segurança do algoritímo.

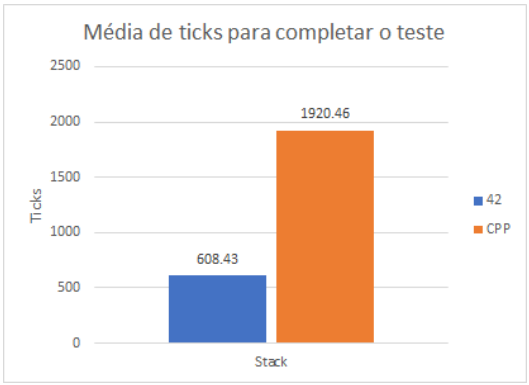

Fig. 4. *Ticks* para completar o teste com a estrutura de dados Pilha

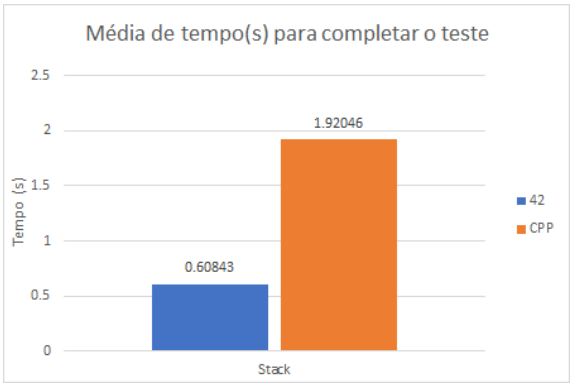

Fig. 5. Tempo para completar o teste com a estrutura de dados Pilha

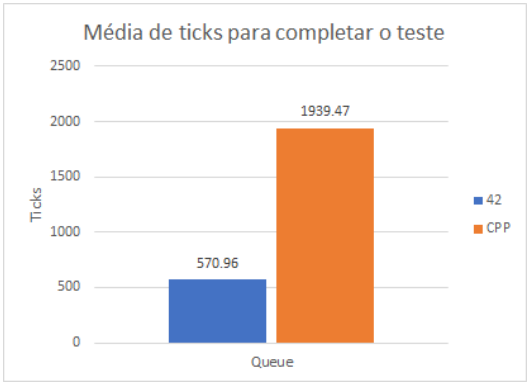

Fig. 6. *Ticks* para completar o teste com a estrutura de dados Fila

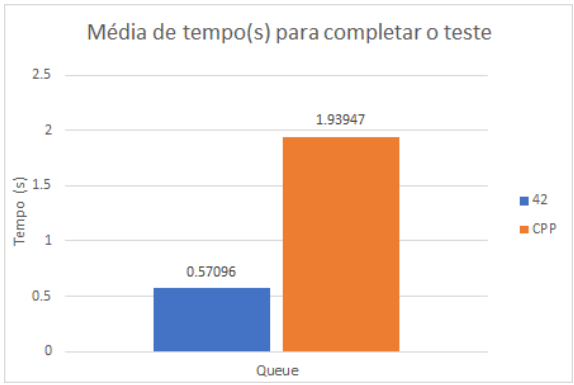

Fig. 7. Tempo para completar o teste com a estrutura de dados Fila