Aluno: Eloyse Fernanda da Silva Costa

Lista de Exercícios - Curso de Estruturas de Dados I (em C/C++) Prof. Igor Machado Coelho
- Tópico: Estruturas Lineares

Observação: os exercícios devem ser feitos em C/C++ (ou similar!). Foque mais na lógica do que emerrosbásicos de programação e SEMPRE discuta a proposta do algoritmo (não quero apenas código!). Sempreanalise a complexidade assintótica dos métodos implementados.

1. Considere um tipo chamado Deque, que inclui manipulação de dois extremos em uma estruturalinear(como se operasse como Pilha e Fila simultaneamente).

```
template<typename Agregado, typename Tipo>
 concept bool DequeTAD = requires(Agregado a, Tipo t){ // requer operação de
                                consulta ao elemento 'inicio'
  { a.inicio() };
  // requer operação de consulta ao elemento 'fim'
  { a.fim() };
  // requer operação 'insereInicio' sobre tipo 't'
  { a.insereInicio(t) };
  // requer operação 'insereFim' sobre tipo 't'
  { a.insereFim(t) };
  // requer operação 'removeInicio' e retorna tipo 't' { a.removeInicio() };
  // requer operação 'removeFim' e retorna tipo 't'
  { a.removeFim() };
};
```

1.a) Satisfaça as seguintes operações de um DequeTAD para o tipo ‘char’, utilizando uma estruturaSequencial OU uma estrutura encadeada:

```
#include <iostream>

using namespace std;
template<typename Agregado, typename Tipo>

concept bool
       DequeTAD = requires(Agregado a, Tipo t){
{ a.inicio() };
{ a.libera() };
{ a.insereInicio(t) };
{ a.insereFim(t) };
{ a.removeInicio() };
{ a.removeFim() };
};

   constexpr int MAXN = 100'000;

class Deque {
public:

   char elementos[MAXN];
   int N;
   int inicioFila, topo ;

   //Complexidade: O(1)
```

```
    void inicio(){
        this-> N = 0;
        this-> inicioFila = 0;
        this-> topo = 0;
    }

    void libera(){

    }
    //Complexidade: O(1)
    void insereInicio(char dado){
        this->elementos[this->inicioFila] = dado;
        this->inicioFila = (this->inicioFila + 1) % MAXN;
        this->N++;
    }

    //Complexidade: O(1)
    void insereFim(char dado){
        this->elementos[this->topo] = dado;
        this->topo = (this->topo + 1) % MAXN;
        this->N++;
    }

    //Complexidade: O(1)
    char removeInicio(){
        inicioFila = (inicioFila + 1) % MAXN;
        N--;
        return this->elementos[N];
    }

    //Complexidade: O(1)
    char removeFim(){
        this->topo = (MAXN - 1);
        this->N--;
        return elementos[N];
    }
};
static_assert(DequeTAD<Deque, char>);


int main()
{
    Deque deque;
    deque.inicio();
        //Insere
        deque.insereFim('A');
        deque.insereFim('B');
        deque.insereFim('C');

        //Imprime --- Complexidade: O(n)
        while (deque.N > 0)
            cout << deque.removeFim() << endl;

    return 0;

}
```

```
static_assert(DequeTAD<Deque, char>); // testa se Deque está correto
```

1.b*) Implemente uma estrutura PilhaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

```
#include <iostream>
using namespace std;

template<typename Agregado, typename Tipo>
concept bool
       DequeTAD = requires(Agregado a, Tipo t){
   { a.inicio() };
   { a.fim() };
   { a.insereInicio(t) };
   { a.insereFim(t) };
   { a.removeInicio() };
   { a.removeFim() };
};

//capacidade maxima do deque
constexpr int MAXN = 100'000;

class Deque {
public:

   char elementos[MAXN];
   int N;
   int inicioFila, topo ;

// inicializa o deque
//Complexidade: O(1)
   void inicio(){
       this-> N = 0;
       this-> inicioFila = 0;
       this-> topo = 0;
   }

// libera o deque
   void fim(){

   }

//Complexidade: O(1)
   void insereInicio(char dado){
       this->elementos[this->inicioFila] = dado;
       this->inicioFila = (this->inicioFila + 1) % MAXN;
       this->N++;
   }


   //Complexidade: O(1)
   void insereFim(char dado){
       this->elementos[this->topo] = dado;
       this->topo = (this->topo + 1) % MAXN;
       this->N++;
   }
   //Complexidade: O(1)
   char removeInicio(){
```

```
        inicioFila = (inicioFila + 1) % MAXN;
        N--;
        return this->elementos[N];
    }

    //Complexidade: O(1)
    char removeFim(){
        this->topo = (MAXN - 1);
        this->N--;
        return elementos[N];
    }
};


template<typename Agregado, typename Tipo>
concept bool
        PilhaTAD = requires(Agregado a, Tipo t)
{
    { a.topo() };
    { a.empilha(t) };
    { a.desempilha() };
};

class PilhaDeque{
public:
    Deque d;

    //Complexidade: O(1)
    void cria (){
        d.N = 0;
        d.inicioFila = 0;
        d.topo = 0;
    }

    void libera (){
        //Nada para liberar
    }

    //Complexidade: O(1)
    char topo(){
        return d.elementos[d.topo-1];
    }

    //Complexidade: O(1)
    void empilha(char data){
        d.insereFim(data);
    }

    //Complexidade: O(1)
    char desempilha(){
        d.removeFim();
    }

};

static_assert(DequeTAD<Deque, char>);
static_assert(PilhaTAD<PilhaDeque, char>);
```

```
int main(){

   PilhaDeque pilha;
   pilha.cria();

   pilha.empilha('A');
   cout << pilha.topo() << endl;

   pilha.empilha('B');
   cout << pilha.topo() << endl;

   pilha.empilha('C');
   cout << pilha.topo() << endl << endl;

   //Complexidade: O(n) -- imprime pilha desempilhada
   while (pilha.d.N > 0)
       cout << pilha.desempilha() << endl;

   pilha.libera();

   return 0;
}
```

1.c*) Implemente uma estrutura FilaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

```
#include <iostream>

using namespace std;


template<typename Agregado, typename Tipo>
concept bool
       DequeTAD = requires(Agregado a, Tipo t){

   { a.inicio() };
   { a.fim() };
   { a.insereInicio(t) };
   { a.insereFim(t) };
   { a.removeInicio() };
   { a.removeFim() };
};
constexpr int MAXN = 100'000;

class Deque {
public:

   char elementos[MAXN];
   int N;
   int inicioFila, topo ;

   //Complexidade: O(1)
   void inicio(){
       this-> N = 0;
       this-> inicioFila = 0;
```

```
        this-> topo = 0;
    }
    void fim(){

    }
    //Complexidade: O(1)
    void insereInicio(char dado){
        this->elementos[this->inicioFila] = dado;
        this->inicioFila = (this->inicioFila + 1) % MAXN;
        this->N++;
    }

    //Complexidade: O(1)
    void insereFim(char dado){
        this->elementos[this->topo] = dado;
        this->topo = (this->topo + 1) % MAXN;
        this->N++;
    }

    //Complexidade: O(1)
    char removeInicio(){
        inicioFila = (inicioFila + 1) % MAXN;
        N--;
        return this->elementos[N];
    }
    //Complexidade: O(1)
    char removeFim(){
        this->topo = (MAXN - 1);
        this->N--;
        return elementos[N];
    }
};

//=============================================

template<typename Agregado, typename Tipo>
concept bool
    FilaTAD = requires(Agregado a, Tipo t)
{
        { a.inicializa() };
        { a.primeiroFila() };
        { a.enfileira(t) };
        { a.desenfileira() };
};

class FilaDeque{

public:
    Deque d;
    //Complexidade: O(1)
    void inicializa(){
        d.N = 0;
        d.inicioFila = 0;
        d.topo = 0;
    }
    //Complexidade: O(1)
    char primeiroFila(){
```

```
        return this->d.elementos[d.inicioFila];
    }
    //Complexidade: O(1)
    void enfileira(char dado){
        this->d.elementos[d.N] = dado;
        this->d.N++;
    }
    //Complexidade: O(1)
    char desenfileira(){
        d.removeInicio();
    }

};
static_assert(DequeTAD<Deque, char>);
//static_assert(PilhaTAD<PilhaDeque, char>);
static_assert(FilaTAD<FilaDeque, char>);

int main(){
    FilaDeque pd;

    pd.inicializa();
    pd.enfileira('A');
    pd.enfileira('B');
    pd.enfileira('C');

    //Testes
    cout <<pd.primeiroFila()<< endl;
    pd.desenfileira();
    cout <<pd.primeiroFila()<< endl;
    pd.desenfileira();
    cout <<pd.primeiroFila()<< endl<<endl;
    pd.desenfileira();

    return 0;
}
```

2) Implemente uma estrutura que satisfaz o TAD Pilha para o tipo ‘char’ e somente utiliza duas Filas comoarmazenamento interno (mais espaço constante):

```
#include <iostream>
#include <queue>

using namespace std;
template<typename Agregado, typename Tipo>
concept bool
        PilhaTAD = requires(Agregado a, Tipo t)
{
    { a.topo() };
    { a.empilha(t) };
    { a.desempilha() };
};
class Pilha {
    std::queue<char> fila1;
    std::queue<char> fila2;

public:
    //Complexidade: O(1)
    char topo()
```

```
    {
        //Complexidade: O(1)
        if (!fila1.empty())
            return fila1.front();
    }

    void empilha(char data)
    {
        fila2.push(data);
        //Complexidade: O(n)
        while (!fila1.empty()) {
            fila2.push(fila1.front());
            fila1.pop();
        }
        std::queue<char> f = fila1;
        fila1 = fila2;
        fila2 = f;
    }

    void desempilha()
    {
        //Complexidade: O(1)
        if (!fila1.empty()){
            fila1.pop();
        }
    }

};

static_assert(PilhaTAD<Pilha, char>);

int main()
{
    Pilha p;
    p.empilha('A');
    p.empilha('B');
    p.empilha('C');

    cout << p.topo() << endl;
    p.desempilha();
    cout << p.topo() << endl;
    p.desempilha();
    cout << p.topo() << endl;
    return 0;
}
```

3) Implemente uma estrutura que satisfaz o TAD Fila para o tipo ‘char’ e somente utiliza duas Pilhas comoarmazenamento interno (mais espaço constante):

```
#include <iostream>
#include <stack>
#include <queue>


using namespace std;

template<typename Agregado, typename Tipo>
```

```
concept bool
       FilaTAD = requires(Agregado a, Tipo t){

   { a.enfileira(t) };
   { a.desenfileira() };
   { a.tamanho() };

};

class Fila2P{
public:

   std::stack<char> pilha1;
   std::stack<char> pilha2;

   void enfileira(char data){
       //Complexidade: O(n)
       while(!pilha1.empty()){
           pilha2.push(pilha1.top());
           pilha1.pop();
       }
       pilha1.push(data);

       //Complexidade: O(n)
       while(!pilha2.empty()){
           pilha1.push(pilha2.top());
           pilha2.pop();
       }
   }

   char desenfileira(){

       //Complexidade: O(1)
       if(!pilha1.empty()){
           char r = pilha1.top();
           pilha1.pop();
           return r;
       }

   }

   int tamanho(){
       if(!pilha1.empty())
           return pilha1.size();
   }



};

static_assert(FilaTAD<Fila2P, char>);
```

4) Escreva um algoritmo que dada uma pilha padrão P externa passada como parâmetro, inverte oconteúdode P. Somente utilize as estruturas extras permitidas como armazenamento externo(mais espaço constante)

a) Uma Fila

```
#include <queue>
#include <iostream>
#include <stack>
using namespace std;

void inverte(std::stack<char>p) {

   std::queue<char> f;

   //Complexidade: O(n)
   while(p.size()){
       f.push(p.top());
       p.pop();
   }
   //Complexidade: O(n)
   while(f.size()){
       p.push(f.front());
       f.pop();
   }
}
```

b) Duas Pilhas

```
#include <queue>
#include <iostream>
#include <stack>
using namespace std;

void inverte(std::stack<char>pilhaPrincipal) {

   //pilhaPrincipal A B C
   std::stack<char> p1;
   std::stack<char> p2;

   //Complexidade: O(n)
   while(pilhaPrincipal.size()-1){
       p1.push(pilhaPrincipal.top()); // C B
       pilhaPrincipal.pop();
   }

   //Complexidade: O(n)
   while(p1.size()){
       pilhaPrincipal.push(p1.top());
       p1.pop();
   }
   p2.push(pilhaPrincipal.top());
   pilhaPrincipal.pop();

   pilhaPrincipal.push(p2.top());

   //C B A
   //while(pilhaPrincipal.size()){
   //    cout << pilhaPrincipal.top() << endl;
   //    pilhaPrincipal.pop();
   //}

}
```

c) Uma Pilha

```
void inverte(std::stack<char>pilhaPrincipal) {

    //pilhaPrincipal A B C
    std::stack<char> p1;

    //Recebe C B A
    //Complexidade: O(n)
    while(pilhaPrincipal.size()){
        p1.push(pilhaPrincipal.top());
        pilhaPrincipal.pop();
    }

    // A B C
    //Complexidade: O(n)
    while(p1.size()){
        pilhaPrincipal.push(p1.top());
        p1.pop();
    }

    //C B A - IMPRIMIR
    /*while(pilhaPrincipal.size()) {
        cout << pilhaPrincipal.top() << endl;
        pilhaPrincipal.pop();
    }*/

}
```

5) Escreva um algoritmo que dada uma fila padrão F externa passada como parâmetro, inverte o conteúdodeF. Somente utilize as estruturas extras permitidas como armazenamento externo (mais espaço constante)

a) Uma Pilha

```
#include <iostream>
#include <queue>
#include <stack>
using namespace std;

void inverte(std::queue<char> f){

    std::stack<char> pilha;

    //Complexidade: O(n)
    while(f.size()){
        pilha.push(f.front());
        f.pop();
    }

    //Complexidade: O(n)
    while(pilha.size()){
        f.push(pilha.top());
        pilha.pop();
    }

    //Imprime complexidade: O(n)
    /*while(f.size()){
```

```
        cout<< f.front() << endl;
        f.pop();

    }*/

}
```

b) Duas Filas

```
#include<iostream>
#include<queue>
using namespace std;

void inverte(std::queue<char> f){
    std::queue<char> fila1;
    std::queue<char> fila2;

    //Complexidade: O(n²)
    while(true){
        int tamanho = f.size();

        //Complexidade: O(1)
        if (tamanho != 0) {
            //Complexidade: O(n)
            for (int i = 0; i < tamanho - 1; i++) {
                fila1.push(f.front());
                f.pop();
            }
            fila2.push(f.front());
            f.pop();
        }else{
            break;
        }

        int tamanho2 = fila1.size();
        //Complexidade: O(1)
        if (tamanho2 != 0) {
            //Complexidade: O(n)
            for (int i = 0; i < tamanho2 - 1; i++) {
                f.push(fila1.front());
                fila1.pop();
            }
            fila2.push(fila1.front());
            fila1.pop();
        } else {
            break;
        }

    }
//Complexidade: O(n)
    while(fila2.size() > 0){
        cout << fila2.front() << endl;
        fila2.pop();
    }


}
```

6) Criar uma implementação do TAD Pilha para o tipo ‘int’, chamada PilhaMin, que oferece os métodosdoTAD e também o método obterMinimo(), que retorna o menor elemento da pilha. O método obterMinimo()deve operar em tempo constante.

```
#include <iostream>
using namespace std;

template<typename Agregado, typename Tipo>
concept bool PilhaTAD = requires(Agregado a, Tipo t){

   { a.topo() };
   { a.empilha(t) };
   { a.desempilha() };
   { a.cria() };
   { a.obterMin() };

};
constexpr int MAXN = 100'000;

class Pilha{
public:

   int elementos[MAXN];
   int N;

   void cria(){
       this->N = 0;
   };
   //Complexidade: O(1)
   int topo(){
       return this->elementos[N-1];
   };

   void empilha(int dado){
       //Complexidade: O(1)
       this->elementos[N] = dado;
       this->N++;
   };

   int desempilha(){
       //Complexidade: O(1)
       this->N--;
       return elementos[N];
   }

   int obterMin(){

       int menor = INT_MAX;

       int cont = N-1;
       //Complexidade: O(n)
       while(cont > 0){
           //Complexidade: O(1)
           if(elementos[cont] < menor){
               menor = elementos[cont];
           }
```

```
            //Complexidade: O(1)
            cont--;
        }

        return menor;
    }


};
static_assert(PilhaTAD<Pilha, int>);
```

7) Escreva um algoritmo que converte uma expressão aritmética parentizada usando as 4 operações paraaexpressão correspondente em notação polonesa reversa.
Exemplo:
Entrada: “((A+B)*(C-(F/D)))”
Saída: “AB+CFD/-*”

```
count_vetor = 1;
polones = 0;
topo = 0;


enquanto count_vetor ≤ fim faça
   se expressao[count_vetor] é operando então
       polones = polones + 1
       pol[count_polones] := expressao[count_vetor]

       senão se expressao[count_vetor] é operador então
           topo = topo+1
           pilha[topo] = expressao[count_vetor]
       se não se expressao[count_vetor] = ")" então:
               se topo != 0 então
               operador = pilha[topo]
                       topo = topo - 1;
                       count_polones = count_polones + 1;
                       pol[count_polones] = operador
               se não "expressao errada"
   count_vetor = count_vetor + 1


Legenda:
    count_vetor -> indice do vetor de expressao
    count_polones -> indice polones
    fim = tamanho do vetor expressao
```