TDA – FILA

Exercícios

Tabela 1: Funcionalidade Básica para as Filas.

	Descrição
Vetor: int cria(desc **pp, int tamVet, int tamInfo); Encadeada: int cria(desc **pp, int tamInfo);	Instancia a estrutura de dados.
int insere(desc *p, void *novo);	Insere na estrutura de dados um registro de informação de uso na aplicação.
int tamanho(desc *p);	Retorna o número atual de informação, de uso na aplicação, inserida na estrutura de dados.
void reinicia(desc *p);	Retorna a estrutura de dados ao estado inicial (vazia de elementos de informação).
void destroi(desc **pp);	Remove a estrutura de dados da memória, desfaz a instanciação.
int buscaNaCauda(desc *p, void *dest);	Pesquisa o item de informação inserido no final da estrutura de dados (fila).
int buscaNaFrente(desc *p, void *dest);	Pesquisa o item de informação inserido na frente da estrutura de dados (fila).
int remove_(desc *p, void *dest);	Remove um item de informação.
int testaSeVazia(desc *p);	Testa se a estrutura de dados está vazia de informação.
Apenas para a(s) fila(s) com suporte em vetor: int testaSeCheia(desc *p);	Testa se a estrutura de dados está cheia.
int inverte(desc *p);	Inverte a ordem dos registros de informação atualmente inseridos na estrutura de dados.

1) Implemente a funcionalidade na Tabela 1 para a TDA Fila Estática Simples

2) Implemente a funcionalidade na Tabela 1 para o TDA Fila Estática com Movimentação de Dados na Remoção.

3) Implemente a funcionalidade na Tabela 1 para o TDA Fila Estática com Movimentação de Dados na Inserção.

4) Implemente a funcionalidade na Tabela 1 para o TDA Fila Estática Circular.

5) Implemente a funcionalidade na Tabela 1 para uma Multi-fila Estática onde cada uma das filas é circular.

6) Implemente a funcionalidade na Tabela 1 para a Fila Estática Circular considerando a inexistência do campo que determina a quantidade de elementos inseridos na fila (campo *tamanhoDaFila*, no descritor). Proponha as (mínimas) alterações necessárias.

7) Compare as implementações da inserção e remoção nas estratégias exibidas na Fig. 1. Qual dessas abordagens de ligação é mais eficiente em termos do número de passos para

realização da tarefa?

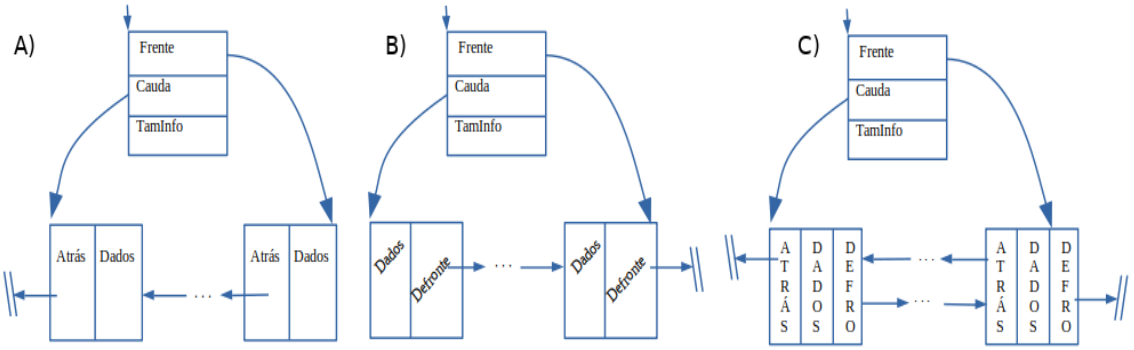

Figura 1: Uma FDDE e duas FDSEs implementadas com elo de ligação em direções opostas.

8) Implemente o TDA Fila Dinâmica Duplamente Encadeada (FDDE) cujo descritor possui "ponteiro-móvel": o mesmo sempre se localiza na extremidade acessada pela última operação (Tabela 1) realizada, portanto não está ancorado/fixado nem na cauda nem na frente da fila, apontará para o nó inicial ou o nó final a depender da última operação ter acessado (respectivamente) ou o início ou o final da fila.

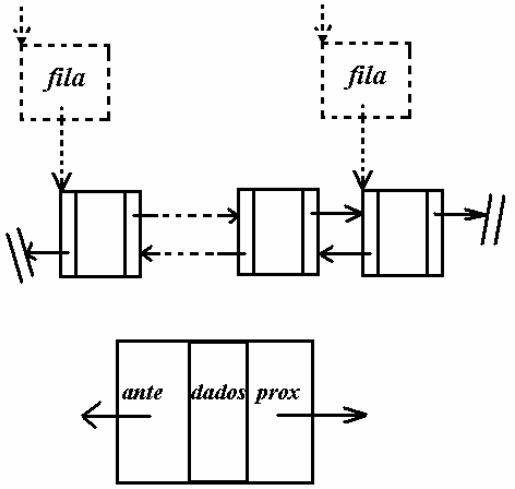

Figura 2: FDDE com descritor "móvel".

9) Implemente uma operação para criar uma cópia da própria fila. Implemente esta operação para uma Fila Dinâmica a sua escolha.

10) Uma fila convencional permite alterações apenas pelas suas extremidades, de maneira que as inserções ocorrem pela cauda e remoções pela frente. Uma *Double-Ended Queue* (DEQUE) é similar a uma fila comum porém permite inserções e remoções tanto pela cauda quanto pela frente. Implemente uma *Double-Ended Queue.*

    Um deque pode ser usado no escalonamento de tarefas para vários processadores separando threads a serem executadas para cada processador. Uma outra aplicação é a verificação de palindromos (ex: RADAR).

11) Construa o TDA FDDE de Prioridade o qual executa inserções posicionando o item de dados segundo o valor de atributo(s) apresentado(s). Aqui será necessário que o módulo cliente forneça uma função que permita ao TDA a definição da relação entre um novo item aqueles já inseridos na fila.

12) Implemente um novo TDA fila dinâmica cujo descritor não contém o tamanho da informação. Nesse caso, além de modificar a definição do descritor, serão necessárias modificações o nó de dados que deverá guardar o tamanho da informação por ele referenciada. Todas as operações deverão ser modificadas, especialmente a operação de inserção a qual deverá receber o tamanho (em bytes) do item que será inserido. Abaixo segue uma figura ilustrando uma PDDE nas condições descritas.

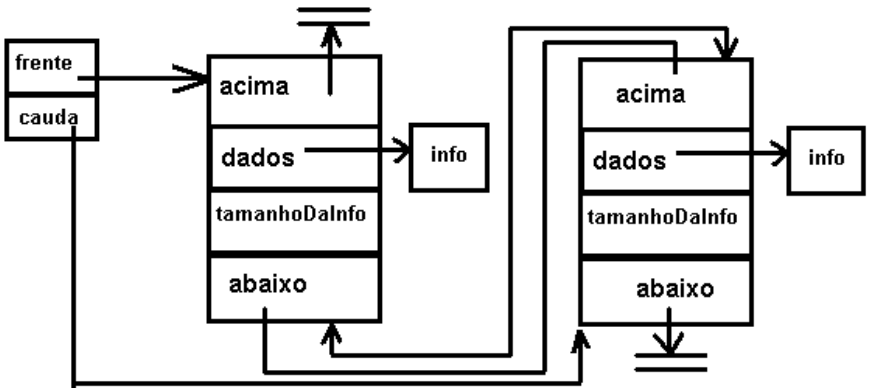

Figura 3: FDDE genérica.

13) Considerando a simulação da fila de um banco na qual a prioridade é determinada pelo tipo de documento bancário trazido pelo cliente. Refaça a inserção na fila considerando que tal operação passou a respeitar apenas o critério de idade, de forma que as pessoas mais velhas ocupem posições mais à frente na fila. No caso de empate de idades, o cliente que primeiro chegou na fila ocupará uma posição à frente do recém-chegado de mesma idade. Implemente uma estratégia para "sortear" a idade das pessoas que chegam ao banco.

14) Refaça a inserção na fila do banco considerando que tal operação passou a respeitar um critério baseado na combinação entre a idade e o saldo em conta-corrente, de forma que as pessoas mais velhas e com menos saldo ocupem posições mais à frente na fila. Implemente uma estratégia para "sortear" a idade e saldo das pessoas que chegam ao banco.

15) Implemente a função abaixo para o TDA multiFES (existem N filas simples do tipo FES) exibido na Fig. 4. A função retorna FRACASSO no caso da fila estar vazia, caso contrário copia o item na frente da fila-alvo para o endereço em *pDestino* e retorna SUCESSO.

Protótipo: *int buscaEmUmaFila(pMultiFES p, int idFila, void *pDestino);*

ATENÇÃO:

a) L é o comprimento da partição de uma fila qualquer f_i, $1 \leq i \leq N$;

b) O início de cada partição (IniPart) consta no descritor individual de cada fila;

c) $-1 \leq cauda_i \leq (L-1)$; $0 \leq frente_i \leq L$;

d) $cauda_i < frente_i$ indica f_i vazia;

e) Cada descritor individual guarda o seu *tamInfo*, pois os dados variam de tamanho a cada fila;

Os campos foram inicializados na criação desse TDA.

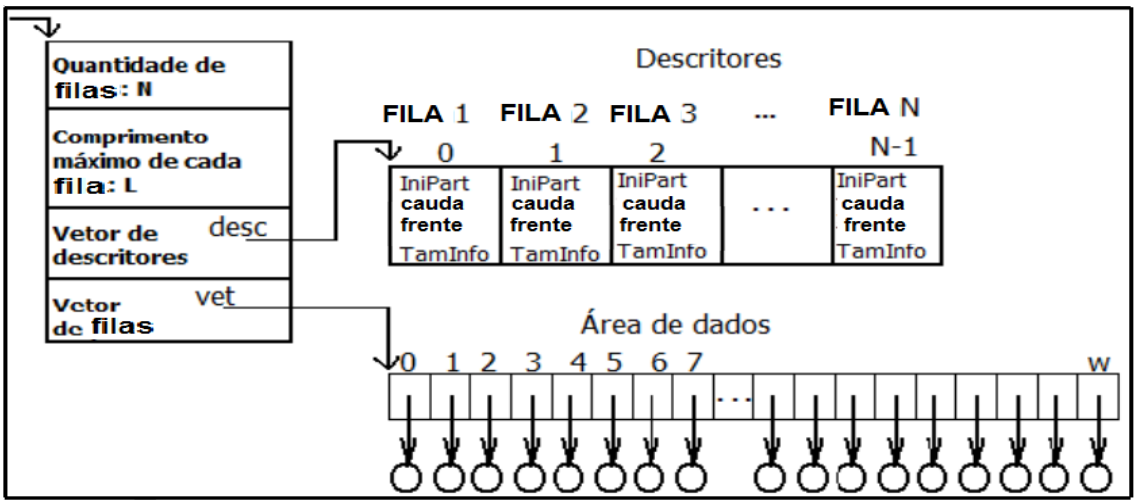

Figura 4: Multifila com vetor de descritores em separado.

16) Implemente a mesma função da questão 15 considerando que o tamanho de partição (L) varia para cada fila. Portanto, L tem que constar como atributo em cada descritor individual, não mais fazendo parte do edscritor geral.

*int buscaEmUmaFila(pMultiFES p, int idFila, void *pDestino);*

17) O joalheiro dispõe de quantidades iguais (centenas) de diamantes e anéis todos apresentam diversos quilates. Ele pretende maximizar o valor dos anéis de brilhante (anel de ouro com uma pedra de diamante), para tanto ele tentará casar cada diamante com um anel de ouro de qualidades parecidas. A ideia é que o diamante de menor quilate case com o ouro de menor quilate e assim sucessivamente, casam-se os pares diamante/anel até que o diamante de maior quilate case com o ouro de maior quilate disponível. Implemente a aplicação de uma fila. Descreva o TDA-Fila que você utilizou.
Saída final: exibição dos pares casados e seus respectivos quilates: (anel, diamante). Os pares descasados devem ser listados.

18) Implemente uma Multi Fila de filas circulares onde a região de descritores encontra-se no final do vetor, conforme a Fig. 5.

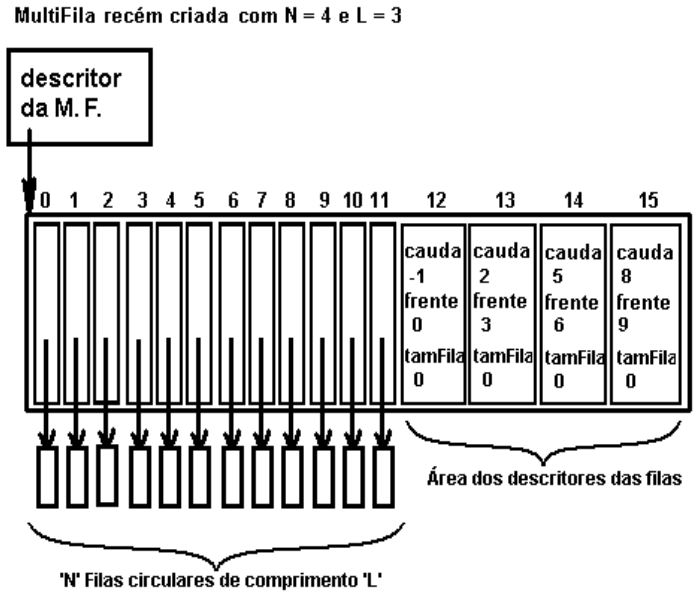

Figura 5: MfEC em um único vetor com descritores no final.

19) Considere uma Multi-fila de filas circulares (Fig. 6) na qual a região de descritores ocorre no início de um vetor de unions, além disso considere que cada descritor de cada fila circular possui os campos: frente, cauda, tamanhoDafila e os campos IP, para indicar onde se inicia a partição da respectiva fila e FP, para indicar o final da mesma partição (ao lado). Para este TDA construa a operação de inserção com o seguinte protótipo: int insere(pMF p, int filaAlvo, void pReg); onde filaAlvo ∈ [1, N].

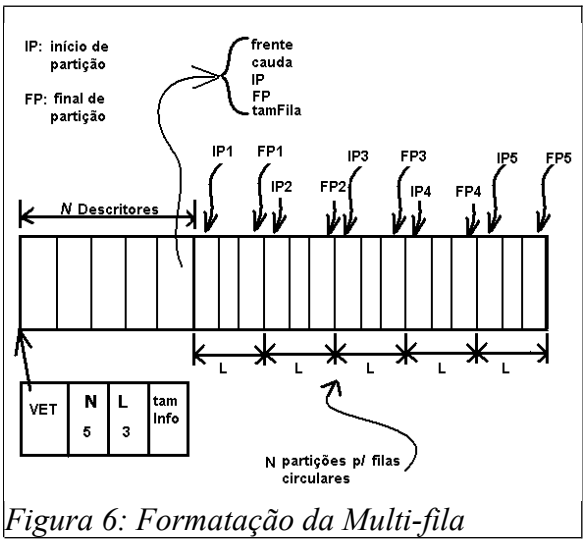

Figura 6: Formatação da Multi-fila

```
/* Nó da Multi-Fila – União de descritor e referência para dados*/
    typedef union {
    DescFila descritor;
    void * dados;
} NoMF, *pNoMF;
```

20) Suponha que o quilate (o qual mede a qualidade do metal) é um valor numérico anotado em cada peça de ouro ou diamante. Uma determinada repartição pública atende a três tipos de serviços, cada um deles possui seu próprio guichê exclusivo, ou seja, o cidadão com serviço tipo *i* só poderá ser atendido pelo respectivo *i*-ésimo guichê. Cada cliente da repartição traz sempre um único conjunto de atividades (chamaremos a isso de *serviço*) a realizar. Cada guichê é identificado com o tipo de *serviço* que este atende, sendo utilizado um *display* (letreiro) eletrônico, visível a todas as pessoas na fila, para indicar a disponibilidade de atendimento dos guichês (quando um guichê está livre o respectivo *display* deve piscar). Apesar dos guichês serem exclusivos para cada tipo de *serviço*, há apenas uma fila na repartição.

A saída da fila para atendimento nos guichês deve respeitar os seguintes critérios: sairá da fila para atendimento aquele elemento que está a mais tempo na fila e que possuir o serviço adequado a um guichê livre. A entrada na fila é convencional (pelo final da mesma).

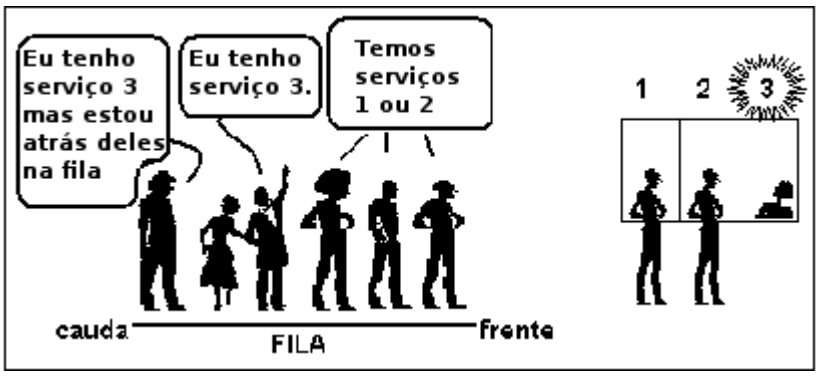

Figura 7 - Atendimento: elemento mais antigo na fila e cujo serviço é adequado ao guichê livre (3).

Pede-se:

A) Uma implementação da operação de remoção, para uma FDDE, que deverá atender aos requisitos para a saída da fila da repartição. O protótipo deve ser o seguinte:
int retira(pFila p, void *itemRemovido, void *prioridade , int (*cmp)(void *p1, void *p2));
B) A implementação da simulação desta repartição aplicando-se uma Fila Dinâmica Duplamente Encadeada (FDDE), ao final da simulação deve-se informar quantos clientes cada guichê atendeu;
C) A estrutura da informação a ser trabalhada pela fila.

Perceba que esta remoção permite, ao seu chamador, acesso ao item removido.
Considere a existência de uma função int comparaServiços(void *guichê, void *cliente) que receberá duas referências, uma para o número de um guichê (livre) e outra para um registro de dados, e comparando o guichê com o serviço trazido por certo cliente, retornará:

i) MENOR se guichê < o campo serviço do cliente;
ii) MAIOR se guichê > o campo serviço do cliente;
iii) IGUAL caso o guichê = o campo serviço do cliente.

Considere também a existência das funções int clienteChegou( ), que informa a chegada ou não de cliente e int serviço(int *servico , int * tempo) que informa o tipo de serviço e o tempo aproximado para realização deste.

Bibliografia

Backus, John. Evolução das Principais Linguagens de Programação. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 49-110.

Barr, Michael. Programming embedded Systems in C and C++. First Edition. Sebastopol: O'Reilly and Associates, 1999. 174p. cap 1.: Introduction: C: The least Common Denominator.

Fishwick, P. A. Computer Simulation Potentials, IEEE , Volume: 15 , Issue: 1 , P:24–27.Feb.-March.1996.

Gersting, Judith L. "Fundamentos Matemáticos para a Ciência da computação". Ed. LTC.

Goldberg, Adele. Suporte para Programação Orientada a Objeto. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 417-466.

Horowitz, E. & Sahni, S. "Fundamentos de Estruturas de Dados". Ed. Campus. 1984.

Pereira, Sílvio L. Estruturas de Dados Fundamentais – Conceitos e Aplicações. 7a. ed. Érica, 2003,p. 73

Parnas, D. L. On the Criteria to be Used in Decomposing systems into Modules Communications of the ACM**,** Volume 15 Issue 12. December 1972.

Ritchie, Dennis. Subprogramas. In: Sebesta, R. W. Conceitos de Linguagens de Programação. 4ª ed. Porto Alegre: Editora Bookman, 2000. p. 315-359.

Szwarcfiter, Jayme L. & Markenzon L. "Estruturas de Dados e Seus Algoritmos". Rio de Janeiro. Ed. LTC. 1994.

Tenembaum, Aaron M. e outros. "Estruturas de Dados Usando C". São Paulo. Ed.Makron Books. 1995.

Tremblay, J.; Bunt, R..Ciência dos Computadores. Uma abordagem algorítmica, McGraw Hill, 1983.

Van Gelder, Allen. A Discipline of Data Abstraction using ANSI C. Documento disponível em http://www.cse.ucsc.edu/~avg/. 02/2005.

Veloso, Paulo et al. "Estruturas de Dados". Ed. Campus. 1984.

Wirth, Niklaus. "Algorthms + Data structures = Programs", Ed. Prentice Hall, 1976.