CI1001 - Programação I

André Grégio, Fabiano Silva, Luiz Albini e Marcos Castilho
Departamento de Informática – UFPR, Curitiba/PR

Décima lista de exercícios

O Tipo Abstrato de Dados Fila

Sabemos que um Tipo Abstrato de Dados encapsula a estrutura de dados para o usuário, disponibilizando funções que manipulam o TAD, sem que o usuário saiba qual é a real implementação desta estrutura.

Na última aula vimos o TAD *Fila*, que adota o padrão FIFO (*First In Fist Out*) para inserções e remoções na estrutura. As funções mínimas que interessam para este TAD são:

- Inicialização da estrutura fila;
- Teste de fila vazia;
- Inserção, mas sempre no final da fila;
- Remoção, mas sempre no início da fila.

Alguns autores as vezes implementam algumas outras funções que permitem por exemplo: obter o número de elementos na fila, imprimir a fila, dentre outras.

O TAD *Fila*, em termos de implementação, pode ser visto como uma especialização do tipo *Lista*, que foi objeto dos trabalhos práticos definidos nas listas de exercícios 8 e 9. Vocês fizeram duas implementações diferentes para o TAD *Lista*.

A segunda delas, a que usa uma lista duplamente encadeada com duas sentinelas para o início e fim, é particularmente apropriada para manipulação de uma fila, pois permite que as operações de inserção e remoção ocorram em tempo constante.

Assim, suas implementações do TAD *Lista* podem ser rapidamente adaptadas para que você tenha implementado um TAD *Fila*. De fato, o conjunto de funções que manipulam uma fila é um subconjunto das que manipulam uma lista. As inserções ocorrem somente no final e as remoções somente no início.

O que deve ser feito

1. Você deve adaptar a biblioteca de listas implementada como uma lista duplamente encadeada para obter o TAD *Fila*. Espera-se que isso seja rápido, pois as implementações deveriam estar prontas ao término do exercício 9. Faça isso em `C` produzindo dois arquivos:
   - `lib_fila.h`, contendo os *headers*; e
   - `lib_fila.c`, contendo a implementação das funções.
2. Lista de funções para serem implementadas:
   - int inicializa_fila(tipo_fila F);
     retorna 1 se inicialou com sucesso a fila ou zero caso contrário;
   - int fila_vazia(tipo_fila F);
     retorna 1 se a fila está vazia ou zero caso contrário;
   - int tamanho(tipo_fila F);
     retorna o tamanho da fila. Se a fila não existe retorna -1;

- int enfileira(int id, int t, tipo_fila F);
  retorna 1 se inseriu com sucesso o id com autonomia de t UT's no final da fila ou zero caso contrário;
- int desenfileira(int *id, int *t, tipo_fila F);
  retorna 1 se desenfileirou com sucesso o id com autonomia de t UT's que estava no início da fila ou zero caso contrário;
- int remove_fila(int *id, int *t, tipo_fila F);
  retorna 1 se removeu com sucesso o id que tinha autonomia de t UT's da fila, não importa a posição que este id ocupa na fila, ou zero caso contrário;
- void imprime_fila(tipo_fila F);
  imprime a fila caso ela exista, senão não imprime nada.

3. Você deve usar esta biblioteca para resolver o problema abaixo, que foi adaptado de [Ziviani, 2004], produzindo um programa em `C` em um arquivo principal de nome `simulacao.c`. Este programa não deve de maneira nenhuma manipular diretamente o TAD *fila*, mas deve simplesmente usar as funções disponibilizadas na biblioteca do item 1.

O problema:
Simulaçao de aterrisagem e decolagem em um aeroporto.

Suponha um aeroporto que possui três pistas, numeradas como 1, 2 e 3. Existem quatro "prateleiras" de espera para aterrisagem, duas para cada uma das pistas 1 e 2. Aeronaves que se aproximam do aeroporto devem integrar-se à uma das prateleiras (filas) de espera, sendo que essas filas devem procurar manter o mesmo tamanho. Assim que um avião entra em uma fila de aterissagem, ele recebe um número de identificação ID e outro número inteiro que indica a quantidade de unidades de tempo (UT) em que o avião pode permanecer na fila antes que ele tenha de descer (do contrário, seu combustível termina e ele cai).

Existem também filas para decolagem, uma para cada pista. Os aviões que chegam nessas filas também recebem uma identificação ID. Essas filas devem procurar manter o mesmo tamanho.

A cada unidade de tempo, de zero a três aeronaves podem chegar nas filas de decolagem, e de zero a três aeronaves podem chegar nas prateleiras de aterrisagem. A cada unidade de tempo, cada pista pode ser usada para um pouso ou uma decolagem. A pista 3, em geral, só é usada para decolagens, a não ser que um dos aviões nas prateleiras de aterrisagem fique sem combustível, quando então ela deve ser imediatamente usada para pouso.

Quando uma aeronave está com falta de combustível, ela pousará na pista 3; se mais de um avião estiver nessa situação, as outras pistas poderão ser utilizadas (a cada unidade de tempo no máximo três aviões poderão estar nessa desagradável situação, senão alguma aeronave cairá).

Utilize inteiros pares (ímpares) sucessivos para a ID dos aviões que chegam nas filas de decolagem (aterrisagem). A cada unidade de tempo, assuma que os aviões entram nas filas antes que aterrisagens ou decolagens ocorram.

Tente projetar um algoritmo que não permita o crescimento excessivo das filas de aterrisagem ou decolagem. Coloque os aviões sempre no final das filas, que não devem ser reordenadas.

A saída do programa deverá indicar o que ocorre a cada unidade de tempo, pela impressão das seguintes informações:

1. o conteúdo de cada fila;
2. o tempo médio de espera para decolagem;
3. o tempo médio de espera para aterrissagem;

4. o número de aviões que aterrissam sem reserva de combustível;

5. os IDs dos aviões que caíram por falta de combustível.

O formato de saída esperado para uma unidade de tempo, no caso a unidade 3, é dado pelo seguinte exemplo:

```
Unidade de tempo 3
fila aterrissagem 1: 1(19), 9(9), 17(1)
fila aterrissagem 2: 3(13), 11(18), 19(12)
fila aterrissagem 3: 13(6)
fila aterrissagem 4: 7(15), 15(11)
fila decolagem 1: 2, 8, 14, 20
fila decolagem 2: 4, 10, 16
fila decolagem 3: 6, 12, 18
tempo medio para aterrissagem: 27
tempo medio par decolagem: 43
numero de aeronaves que aterrissaram sem combustivel: 0
IDs das aeronaves que cairam: 5
```

Neste formato, 1(19) significa a aeronave com ID 1 que tem 19 UT's de autonomia antes de ter que pousar sem combustível.

Os itens 2 e 3 devem ser calculados para os aviões que já decolaram ou pousaram, respectivamente.

A entrada poderia ser criada aleatoriamente, mas neste trabalho ela será lida a partir do teclado ou a partir de um arquivo usando-se redirecionamento da entrada padrão do *shell*.

A entrada deve ter inicialmente o tempo total da simulação, que é um valor inteiro. Para cada unidade de tempo, a entrada deve ter as seguintes informações:

1. número de aviões (zero a três) chegando nas filas de aterrissagem com respectivas reservas de combustível (de 1 a 20 UT's);

2. número de aviões (zero a três) chegando nas filas de decolagem;

Mais especificamente a entrada deve ser assim prevista:

```
3         # a simulacao vai ter 3 UT's, portanto mais 6 linhas abaixo
3 12 9 15 # chegaram 3 avioes para aterrissagem com suas autonomias de 12, 9 e 15 UT's
2         # chegaram 2 avioes para decolagem
2 6 1     # chegaram mais 2 avioes para aterrissagem (6 e 1 UT's)
3         # chegaram mais 3 avioes para decolagem
1 11      # chegou mais 1 aviao para aterrissagem (11 UT's)
2         # chegaram mais 2 avioes para decolagem
```

Observe que você deve controlar os IDs dos aviões, que nunca se repetirão, as aterrissagens iniciam em 1 (ímpares) e as decolagens em 2 (pares).

Note também que a implementação da fila deve prever o caso de "furar" a fila, uma vez que aviões sem combustível ganham prioridade e devem sair da fila mesmo que não estejam no início. Furar a fila significa remover um avião que não está no início da fila.

Entregáveis

A entrega deve ser feita pelo Moodle na forma de um único arquivo tarball, contendo os arquivos seguintes arquivos:

1. Um arquivo de nome `simulacao.c`: implementa a solução para o problema do aeroporto definido acima;
2. Os arquivos `lib_fila.h` e `lib_fila.c`;
3. Caso você precise de outras funções que não estão na bibioteca base `lib_fila.h`, entregue também os `.h` e os `.c` correspondentes.

Observações

- A implementação **não** deve manipular diretamente o TAD *Fila*, mas deve **usar** as funções da biblioteca `lib_fila.h`.
- Faça boas apresentações dos códigos (legibilidade, identação, nomes adequados de variáveis e de constantes). Bons comentários no código são fundamentais, mas não exagere no número deles, use-os com bom senso em situações pertinentes.

Referências

[Ziviani, 2004] Ziviani, N. (2004). *Projeto de algoritmos: com implementações em Pascal e C*. Pioneira Thomson Learning.