PROGRAMMING CHALLENGES
Resumo - Capítulo 2: Data Structures

1. Estruturas de Dados Elementares

1.1 Pilhas: Tipo Abstrato de Dados cujo principal caraterística é que o último elemento a ser inserido é o primeiro a ser retirado/removido. As operações básicas de uma pilha são:

I. PilhaPush(S, X): Insere item S no topo da pilha S.
II. PilhaPop(S): Retorna (e remove) o topo da pilha S.
III. PilhaInicializa(S): Cria uma pilha vazia S.
IV. PilhaCheia(S), PilhaVazia(S): Testa se a pilha S está cheia ou vazia, respectivamente.

1.2 Filas: Tipo Abstrato de Dados cujo principal caraterística é que o primeiro elemento a entrar é também o primeiro elemento a sair. As operações básicas de uma fila são:

I. FilaPush(Q, X): Insere o item X no final da fila Q.
II. FilaPop(Q): Retorna (e remove) o item na frente da fila Q.
III. FilaInicializa(Q): Cria uma fila vazia Q.
IV. FilaCheia(Q), FilaVazia(Q): Testa se a fila q está cheia ou vazia, respectivamente.

1.3 Dicionários: Permite retornar um item a partir de sua chave, ao invés de utilizar a posição do elemento. As operações básicas de um dicionário são:

I. DicInsere(D, X): Insere item X no dicionário D.
II. DicRemove(D, X): Remove item X (ou o item para o qual X aponta) do dicionário D.
III. DicBusca(D, K): Retorna o item com chave K do dicionário D (se o item existir).
IV. DicInicializa(D): Inicializa o dicionário D.
V. DicCheio(D), DicVazio(D): Testa se o dicionário D está cheio ou vazio, respectivamente.

1.4 Filas de prioridade: é uma estrutura de dado que mantém uma coleção de elementos, cada um com uma prioridade associada. Suas operações básicas são:

I. FPInsere(Q, X): Insere item X na fila de prioridade Q.
II. FPRemove(Q): Remove e retorna o maior item da fila de prioridade Q.
III. FPMaximo(Q): Retorna o maior item da fila de prioridade Q.
IV. FPInicializa(Q): Inicializa a fila de prioridade Q.
V. FPCheio(Q), FPVazio(Q): Testa se a fila de prioridade Q está cheia ou vazia, respectivamente.

1.5 Conjuntos: nada mais são do que uma coleção de elementos. Contemplam as seguintes operações:

I. SetMembro(S, X): Retorna TRUE se o item X for membro do conjunto S.
II. SetUniao(A, B): Retorna um novo conjunto A ∪ B.
III. SetIntersecao(A, B): Retorna um novo conjunto A ∩ B.
IV. SetInsere(S, X), SetRemove(S, X): Insere e remove, respectivamente, o item X do conjunto S.
V. SetInicializa(S): Inicializa um conjunto S vazio.

2. A Biblioteca de Modelo Padrão C ++

A Biblioteca de Modelo Padrão C ++ (STL) fornece implementações de todas as estruturas de dados definidas acima. Cada objeto de dados deve ter o tipo de seus elementos fixo. No exemplo abaixo, temos a declaração de duas pilhas com diferentes tipos de dados:

```
#include <stl.h>

stack<int> S;

stack<char> T;
```

Na tabela a seguir temos a apresentação das funções presentes na biblioteca “stl.h” referentes as estruturas de dados analisadas.

Estrutura	Funções
Pilha (P)	P.push(), P.top(), P.pop(), e P.empty().
Fila (F)	F.front(), F.back(), F.push(), F.pop(), e F.empty().
Dicionário (D)	D.erase(), D.find(), e D.insert()
Fila de Prioridade (Q)	Q.top(), Q.push(), Q.pop(), and Q.empty().
Conjunto	set<chave, comparador>, set_union e set_intersect

3. Testando e Depurando

A depuração pode ser particularmente frustrante com o juiz do robô, já que não é possível visualizarmos o caso de teste no qual o programa falhou. Abaixo estão alguns bons meios de teste:

I. **Teste a entrada fornecida:** A maioria das especificações de problemas incluem entrada e saída de exemplos. Atenção! É possível que seu programa dê a resposta certa para o exemplo de entrada disponível, entretanto, ele não esteja correto para demais casos de teste.

II. **Teste entradas incorretas:** Se a especificação do problema indicar que o programa deve tomar medidas sobre entradas incorretas, lembre-se de testá-las.

III. **Teste condições limites:** O erro do seu programa pode estar acontecendo devido a uma entrada específica. É importante testar seu código para entrada vazia, com um único item, com dois itens e valores iguais à zero.

IV. **Teste instâncias onde você sabe a resposta correta:** Uma parte crítica do desenvolvimento de um bom caso de teste é ter certeza de que você sabe qual é a resposta certa. Seus casos de teste devem se concentrar em instâncias simples o suficiente para que você possa resolvê-las manualmente.

V. **Teste instâncias grandes onde você não sabe a resposta correta:** Experimente algumas instâncias grandes facilmente construídas, como dados aleatórios ou os números 1 a n inclusive, apenas para garantir que o programa não falhe ou faça algo estúpido.

Outras dicas importantes...

- Conheça bem seu debugador
- Exiba sua estrutura de dados
- Faça testes invariantes
- Verifique seu código
- Faça seus “prints” significarem alguma coisa
- Crie seus arranjos um pouco maiores do que o necessário
- Tenha certeza de que seus bugs são bugs de verdade