INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO - CAMPUS BRAGANÇA PAULISTA

SOLEMAR

JOSIE APARECIDA BATISTA

MARCEL

Trabalho 2 da disciplina Estrutura de Dados I

Bragança Paulista
2016

Sumário

1 INTRODUÇÃO

Este trabalho tem como objetivo apresentar os conceitos e características das estruturas de dados, buscando demonstrar a importância da sua aplicação no armazenamento e organização das informações.

Algumas estruturas exercem tarefas específicas e , são adequadas para diferentes tipos de aplicações, caso sejam utilizadas incorretamente, o desempenho do sistema será afetado e gerará saídas erradas. Diante disso, este trabalho deseja contribuir na especificação do papel que cada estrutura exerce, cabendo ao programador escolher qual estrutura é essencial para o domínio do problema a ser resolvido.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo abordará a pesquisa bibliográfica dos conceitos de Estrutura de dados, Queue, Stack, ArrayList, LinkedList, HashSet e HashMap.

2.1 Queue

As Filas (queue), são estruturas de dados dinâmicas que admitem remoção de elementos e inserção de novos objetos. Sempre que uma remoção for feita, o primeiro objeto inserido na fila é o primeiro a ser removido. Essa política é conhecida pela sigla FIFO (*First-In-First-Out*).

De acordo com Koffman e Wolfgang (2008, p. 246), as filas (queue), são utilizadas em sistemas operacionais para controlar tarefas. Desta forma, as tarefas devem ser certificadas para que sejam executadas na ordem em que foram geradas.

“ Existem várias possibilidades de aplicações para fila. Lojas, teatros, centrais de reservas e outros serviços similares normalmente processam as requisições dos clientes de acordo com o princípio FIFO.” (GOODRICH e TAMASSIA, 2013, p.2018).

De acordo com os autores a fila suporta dois métodos fundamentais:

- enqueue(*e*): Insere o elemento *e* no fim da fila
- dequeue(): Retira e retorna o objeto da frente da fila. Ocorre um erro se a fila estiver vazia.

A fila também inclui os seguintes métodos auxiliares:

- size(): Retorna o número de objetos na fila.
- isEmpty(): Retorna um booleano indicando se a fila está vazia.

Esses dois métodos são conhecidos como métodos de acesso, pois retornam um

valor e não alteram o conteúdo da estrutura de dados.

A implementação dessa estrutura requer a utilização de um vetor, onde um elemento é inserido no final da fila (a direita) e retirado do início da fila (a esquerda). Esta implementação é dificultada devido ao fato da fila se locomover dentro do vetor. Desta forma, é necessário controlar essa locomoção.

A utilização da fila é uma representação abstrata do mundo real, ou seja, se um cliente estiver em uma fila de um banco há mais tempo , ele sairá primeiro mantendo a ordem de chegada. O mesmo acontece com a estrutura de dados fila, o primeiro dado armazenado será o primeiro a ser retirado.

2.2 Stack

Uma pilha (stack) é uma coleção de objetos que são inseridos e retirados de acordo com o princípio de que o último que entra é o primeiro que sai.

Segundo Goodrich e Tamassia (2013, p.202), " o nome "pilha" deriva-se da metáfora de uma pilha de pratos em uma cantina. Neste caso, as operações envolvem a colocação e retirada de pratos da pilha."

As pilhas são umas da estruturas de dados mais simples, apesar de estarem entre as mais importantes, elas suportam os dois métodos que serão apresentados a seguir:

- Quando um item é adicionado em uma pilha, usa-se a operação Push (inserir);
- Quando um item é retirado de uma pilha, usa-se a operação Pop (retirar);

Adicionalmente, podem-se definir os seguintes métodos:

- Size(): Retorna o número de elementos na pilha.
- isEmpty(): Retorna um booleano indicando se a pilha está vazia.
- Top(): Retorna o elemento no topo da pilha, sem retirá-lo; ocorre erro se a pilha estiver vazia.

As pilhas ocorrem em estruturas de natureza recursiva (como árvores). Elas são utilizadas para implementar a recursividade.

Neste contexto, as pilhas seguem uma ordem de que o primeiro que entra é o

último que saí, podemos exemplificar com uma pilha de pratos, sempre retiramos o primeiro para ser utilizado, ao tentarmos retirar o último prato empilhado, dependendo da quantidade, causaria um trabalho enorme na sua retirada. Desta forma, o primeiro prato é mais vantajoso a ser utilizado.

2.3 Vantagens e Desvantagens das Filas e Pilhas

As filas (queue) e as pilhas (stack), são listas lineares , ou seja, são estruturas de dados onde elementos do mesmo tipo estão organizados sequencialmente e formados por um conjunto de dados de um mesmo tipo. Esses elementos não necessariamente estão fisicamente em ordem na memória.

Quando os elementos estão fisicamente em ordem é chamado de lista linear sequencial (ou contigua).

Quando não estão fisicamente em ordem é chamado de lista linear encadeada.

De acordo com Mafra (2009), as vantagens e desvantagens das listas lineares usando arranjos são:

Vantagem:

- Acesso direto indexado a qualquer elemento da lista.
- Economia de memória (não utiliza apontadores).

Desvantagens:

- Custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;
- Tamanho máximo da lista pré-estimado (definido em tempo de compilação).

Usando apontadores:

Vantagens:

- Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).
- Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido a *priori)*.

Desvantagem:

- Utilização de memória extra para armazenar os apontadores.

A maneira de implementar listas lineares depende da classe de operações mais frequentes. Não existe, em geral, uma única implementação para a qual todas as operações são eficientes.

2.4 ArrayList

ArrayList é uma classe para coleções (classe genérica), que implementam a interface List, utilizando internamente um array dinâmico de objetos, assim, caso o tamanho do array interno não atenda a uma nova inserção, um novo array será alocado dinamicamente.

Com essas coleções é possível realizar várias operações como:

Adicionar e retirar elementos, ordená-los, procurar por um elemento específico, apagar um elemento, acessar através de índice, limpar o ArrayList dentre outras possibilidades.

Segundo Deitel (2010, p.221), o ArrayList pode alterar dinamicamente seu tamanho para acomodar mais elementos. Quando se declara um novo ArrayList, é necessário informar o tipo de elementos que ele irá conter.

A seguir é demonstrado um exemplo de declaração do ArrayList:

ArrrayList<String> list;

O list é declarado como uma coleção ArrayList que só poderá armazenar Strings.

Assim que está definido e instanciado, é possível efetuar uma série de ações.

A seguir serão exemplificados alguns métodos utilizados nessa estrutura:

- add – adiciona um elemento ao fim do ArrayList;
- clear – remove todos os elementos do ArrayList;

- contains – retorna true se o arrayList contiver o elemento especificado; do contrário, retorna false;
- get – retorna o elemento no índice especificado;
- indexOf – retorna o índice da primeira ocorrência do elemento especificado em ArrayList;
- remove – remove o elemento no índice especificado;
- size – retorna o número de elementos aramazenados no ArrayList;
- trimToSize – reduz a capacidade de ArrrayList de acordo com o número e elementos atual.

Utiliza-se a classe ArrayList nas seguintes situações:

- Quando o tamanho do objeto for previsível, evitando que o máximo do tamanho do vetor seja alcançado e exija realocações. Apesar de possível, a realocação consome recursos;
- Quando as operações de inserção e deleção são feitas, em sua maioria, no fim da lista (evitando deslocamentos);
- Quando a lista for mais lida do que modificada (uso otimizado para leitura aleatória).

Os ArrayList são estruturas mais rápidas para acessar os elementos, no entanto é mais lenta para inserção e remoção, em algumas aplicações sua utilização é essencial, caso algumas situações não sejam atendidas , podemos optar pela classe LinkedList, que veremos a seguir.

2.5 LinkedList

De acordo com Mendes (2011, p.44), a classe LinkedList implementa a interface List e utiliza internamente uma lista encadeada. Recomenda-se usar LinkedList quando as modificações (inserção ou deleção) na coleção são feitas na maioria das vezes no início e no final da lista, e ainda quando a navegação entre os elementos é feita de forma sequencial por meio de um Iterator. Caso seja necessário

buscar elementos usando índices, devemos optar pelo uso da classe ArrayList. Uma coleção do tipo LinkedList permite o uso do método get() para buscar um objeto em uma posição específica.

Não é possível criar objetos do tipo LinkedList informando o tamanho inicial na chamada ao construtor, ou seja, para cada novo valor a ser inserido na coleção, um novo objeto será criado automaticamente.

Vantagens:

- são estruturas de dados dinâmicos que alocam a memória necessária enquanto o programa está funcionando;
- operações de inserir e deletar (nodes) são facilmente implementadas;
- estruturas de dados lineares tais como, pilhas e filas são facilmente executadas com uma lista encadeada;
- eles podem reduzir o tempo de acesso e podem aumentar em tempo real sem overhead de memória.

Desvantagens:

- Desperdiçam memória pelo fato dos ponteiros alocarem novos espaços de armazenamento;
- Nós em uma lista encadeada devem ser lidos no início;
- Dificuldades ao percorrer de trás para frente, essa tarefa causa desperdício de memória.

Vimos que o LinkedList é mais rápido e utilizam a memória necessária enquanto

o programa está em funcionamento, porém algumas tarefas causam desperdício de memória, desta forma é necessário avaliar o uso dessa estrutura para que o programa não perca seu desempenho.

2.6 HashSet

Segundo Mendes (2011, p.25), a interface Set estende a interface Collection e define uma coleção que não permite a inclusão de nenhum objeto duplicado, ou seja, será ignorada a adição de um objeto caso outro objeto equivalente já exista na coleção. Suas características são:

- não aceita a duplicação de elementos;
- não possui forma direta de apresentação de seus itens;
- velocidade na pesquisa de dados, sendo mais rápida que um objeto do tipo List;
- não precisa especificar a posição para adicionar um elemento.

Utiliza o seguintes métodos:

- add: adiciona um item por vez;
- addAll: adiciona itens de outra coleção;
- remove: remove um item;
- removeAll: remove itens de outra coleção;
- clear: limpa o conjunto;
- contains: verifica se contém determinado objeto no conjunto;
- containsAll: verifica se contém elementos de outra coleção retornando true ou false;
- size: verifica quantidade de itens ;
- isEmpty: verifica se o conjunto está vazio;
-

Para Manzano e Junior (2001, p.237), “ a interface set é utilizada em conjunto

com a classe HashSet(), um arranjo do tipo set é usado para criar uma lista de objetos, permitindo a existência de elementos únicos.” Possui características como:

- não tem ordenação na varredur a ou impressão. A ordem de saída não é a mesma de entrada;
- aceita valores do tipo null;
- não é sincronizada;
- velocidade no acesso, leitura e modificações de dados.

O uso dessa estrutura permite ter acesso aos elementos de forma rápida, tanto para leitura quanto para modificações, entretanto deve-se fazer uma avaliação da situação em que a mesma será utilizada, pois não garantem a ordem de seus elementos, mesmo sendo a estrutura de melhor desempenho da interface Set.

2.7 HashMap

A interface Map não estende a interface Collection, ou seja, tem sua própria hierarquia de interfaces e classes que são utilizadas para gerenciar associações entre chaves e elementos. Estas definem um array associativo, por exemplo, ao adicionar um novo elemento ao mapa, este o associa a uma chave, que será utilizada para recuperar o elemento adicionado. (MENDES, 2011, p.48).

Ainda segundo o autor , o HashMap implementa a interface Map e utiliza uma tabela hash para armazenar seus elementos. Quando adicionamos elementos a uma coleção do tipo HashMap, devemos informar a chave e o conteúdo que estará associado à chave. Caso seja incluído algum elemento em uma chave já ocupada, o objeto HashMap substituirá tal valor.

Características do HahMap:

- os elementos não são ordenados;
- é rápida na busca e inserção de dados;

- permite inserir valores e chaves nulas.

Possui os seguintes métodos:

- clear: remove todos elementos do mapa;
- containsKey: retorna true se o mapa contém a chave solicitada;
- containsValue: retorna true se o mapa contém o valor solicitado;
- equals: compara um objeto com o mapa para igualdade;
- get: recupera o valor da chave solicitada;
- keySet: retorna um conjunto que contém todas as chaves do mapa;
- put: adiciona o par de chave-valor solicitado no mapa;
- remove: remove a chave solicitada w seu valor a partir do mapa, se a chave existe
- size: retorna o número de pares de chave

Os elementos armazenados no mapas podem ser localizados rapidamente através de sua chave. A chave é utilizada como um identificador único, ou seja, um objeto serve como um tipo de localização para um determinado valor. (GOODRICH e TAMASSIA, 2013, p.386).

Essa estrutura é muito útil em situações que seja necessário buscar objetos por chave, desta forma cada elemento possuirá um identificador que determinará sua localização. Um exemplo seria a identificação de uma pessoa, cada uma delas possui um número de documento único, ou seja, cada pessoa possui seu identificador.

REFERÊNCIAS

Coleções. Disponível em: <http://www.miscelaneadoconhecimento.com.br/java/suplementos/colecoes/map.html> Acesso em: 15 Jun. 2016

Conhecendo a interface Map do java. Disponível em: <http://www.linhadecodigo.com.br/artigo/3670/conhecendo-a-interface-map-do java.aspx> Acesso em: 15 Jun. 2016

DEITEL Paul ; DEITEL, Harvey. **Java como programar.** 8 ed. São Paulo: Person Prentice Hall, 2010, 639 p.

Estruturas de Dados Estáticas e Dinâmicas. Disponível em: <http://www.din.uem.br/~ronaldo/LivroAED-Capitulos-7-Estruturas.pdf> Acesso em: 15 Jun. 2016

Filas. Disponível em: <http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html> Acesso em: 15 Jun. 2016

GOODRICH , T. Michael ; TAMASSIA , Roberto .**Estrutura de Dados & Algoritmos em Java ™.** 5ª ed. Porto Alegre : Bookman, 2013, 713p.

KOFFMAN , B. Elliot ; WOLFGANG , A.T. Paul . **Objetos, Abstração, Estruturas de Dados e Projeto Usando Java versão 5.0.** Rio de Janeiro : LTC, 2008, 695 p.

Listas lineares e Sequenciais. Disponível em: <http://www.lncc.br/~rogerio/ed/02%20%20Listas%20Lineares%20Sequenciais.pdf> Acesso em: 15 Jun. 2016

MANZANO , N.G. Augusto José ; JUNIOR , da Costa Affonso Roberto . **Java como programar.** 1ª ed. São Paulo: Érica, 2011, 384 p.

MENDES, Rocha Augusto**. Programação Java em Ambiente Distribuído: Ênfase no Mapeamento Objeto-Relacional com JPA, EJB e Hibernate**. Disponível em: <https://novatec.com.br/livros/programacaojava/capitulo9788575222621.pdf> Acesso em: 24 Jun. 2016

MAFRA, Juliana. **Tipos Especiais de Listas.** Disponível em: <www.cin.ufpe.br/~jndm/edados/slides/Aula4_Filas.ppt> Acesso em: 15 Jun. 2016

Trabalhando com a interface Set. Disponível em: <http://www.linhadecodigo.com.br/artigo/3669/trabalhando-com-a-interface-set-no-java.aspx> Acesso em: 15 Jun. 2016