INSTITUTO POLITÉCNICO DE SETÚBAL

Tipos Abstratos de Dados

Milestone1

Realização

Ricardo José Horta Morais
140221066

Responsáveis

Patrícia Alexandra Pires Macedo

José António Sena Pereira

11/16/2015

Este documento promete explicar as decisões tomadas na realização dos tipos abstratos de dados pedidos no enunciado de projeto da unidade curricular de programação avançada.

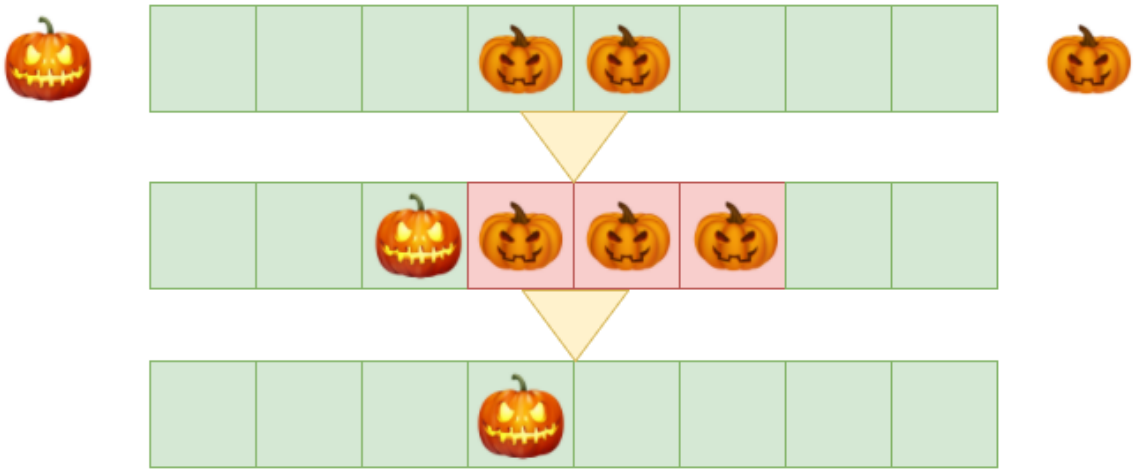
TAD LinhaTres

Enunciado

Uma LinhaTres é uma coleção de itens com um comportamento semelhante ao TAD Deque permitindo inserir e retirar elementos no início e no fim da fila. No entanto as operações de remover elementos, removem sempre 3 elementos de uma vez e a operações de getFirst e getLast, devolvem os 3 primeiros elementos e os 3 últimos respetivamente. O TAD LinhaTres tem a noção de capacidade máxima, gerando um erro quando essa é excedida. Disponibiliza ainda um iterador para percorrer a coleção.

Descrição

O TAD LinhaTres deu especial trabalho. Fez-me pensar naquilo que seria a o produto final, e contrastar com o que seria a solução simples. Deveras como programador perdi alguns segundos do meu tempo e apercebi-me que se implementasse o TAD da maneira usual este teria conflito com a posterior interface gráfica. No que diz respeito ao funcionamento básico do TAD fiz aquilo que foi pedido. Implementei um Deque dinâmico e circular com apenas algumas alterações. O conflito apareceu quando houve a necessidade de adicionar uma restrição na capacidade da coleção. A coleção que posteriormente seria usada para representar uma linha no jogo acabaria por não representar a linha como dita o enunciado do projeto. Consideremos a seguinte sequência de jogadas.



A sequência mostra como um deque simples não mapeia bem o problema apresentado no jogo. Este caso mostra que é preciso considerar que existe a noção de centro no deque e que os elementos podem estar tanto à direito do deque como à esquerda. Por esta razão utilizei um nó de dupla ligação. Este é apenas uma sentinela a que nenhum outro nó se liga, este apenas age como uma referência sendo que os seus lados anterior e posterior correspondem ao primeiro elemento à esquerda e à direita respetivamente. Isto por si não soluciona o caso seguinte.

Um Deque normal com capacidade fixa pode parecer ao início que resolva a situação anterior de verifica a condição de final de jogo, no entanto observando melhor a situação apresentada é possível testemunhar o problema. A imagem apresenta a sequência de fim de jogo prevista. Dando uma capacidade máxima ao Deque de 8 unidades não implicará que o jogo termine quando o deque tiver 8 elementos pois como podemos ver na imagem, tendo apenas 7 elementos o jogo termina. Com isto percebi que deveria existir aqui a noção de lado, um lado esquerdo e um lado direito, ambos relativos ao meu nó de centro. Agora com esta solução é possível verificar que a condição de final de jogo é dada quando todas as casas de um lados estiverem preenchidas.

Com tudo isto em mente fiz a minha implementação e assegurei-me que provavelmente não irão acontecer conflitos com a parte gráfica. Dois iteradores um para cada lado complementaram a solução.

TAD ConjuntoAleatorio

Enunciado

Um conjunto aleatório é uma coleção de itens não repetidos, em que não é possível adicionar ou retirar elementos, e que disponibiliza uma única operação: a operação de visualização aleatória de um item. Ou seja, se tivermos um conjunto aleatório com as cores amarelo, verde e azul, a operação de peek (visualização de um item), devolve aleatoriamente uma das 3 cores.

Descrição

A TAD ConjuntoAleatorio é uma estrutura muito simples, passamos todos os elementos que queremos que a estrutura providencie e sendo o número de elementos um valor estático, basta usar um array. O uso dos índices do array é bastante eficiente e visto que é apenas necessário gerar um valor entre 0 e N(número de elementos) para indexar o array e ir buscar o valor, esta é a solução adequada.

TAD Ranking

Enunciado

Um ranking é uma lista ordenada de elementos, em que os elementos são ordenados segundo um critério específico.

Descrição

A TAD Ranking acaba por ser uma espécie de uma lista. No meu caso como a lista deve estar sempre ordenada, usei a interface Comparador e a Strategy pattern. No fundo a implementação da estratégia (Interface de Comparador) é passada pelo construtor e portanto a TAD Ranking desconhece a sua implementação. Baseando-se na interface Comparador, a TAD Ranking, a cada elemento que é introduzido à lista, faz um Bubble Sort usando o tal critério. A escolha do algoritmo bubble sort deve-se ao facto de este ser simples de entender e alem disso como esta coleção é mantida ordenada, este estará sempre próximo do melhor caso. Em relação ao resto foi tudo implementado usando nós simples e uma única sentinela.

TAD Historico

Enunciado

O Histórico caracteriza-se por ser uma coleção de elementos, com tamanho limitado, onde é apenas possível colocar elementos não sendo possível retira-los. Os elementos são guardados por ordem de entrada para o histórico e ficam associados a uma data (a data de entrada para o histórico). É possível percorrer sequencialmente todos os elementos, assim como saber quais os elementos de um determinado dia, ou dos últimos dias ou do mês corrente etc. Quando o número de elementos é atingido, os elementos mais antigos são descartados automaticamente.

Descrição

A TAD Historico foi implementado quase como um Deque. Os elementos vão entrando pela esquerda e ao chegar à capacidade máxima do Historico, este trata de eliminar o mais antigo mantendo assim o número de elementos inferior ou igual à capacidade. É preciso notar que nesta implementação não existe algoritmo de ordenação e portanto mesmo ao inserir elementos com a correspondente data de entrada não garante que estes fiquem ordenados por data. Uma possibilidade seria quando a data no computador estiver incorreta, a JVM obtem a data do sistema que está a correr e portanto os Historico deixará entrar um elemento com uma data anterior a outro. Visto que nada foi dito acerca da ordem assumi o melhor caso e não implementei qualquer tipo de solução a este problema visto que não existe conflito com os restantes métodos desta TAD.

O mais problemático de implementar foi os iteradores. No enunciado estava especificado:

É possível percorrer sequencialmente todos os elementos, assim como saber quais os elementos de um determinado dia, ou dos últimos dias ou do mês corrente etc.

Isto levou-me a perceber que existem vários critérios de seleção de elementos. Imediatamente pensei na Strategy Pattern juntamente com expressões lambda. Uma interface faz a abstração do critério e dentro do método **getIteradorPorCriterio** que recebe o critério concreto, uma classe anónima que implementa a interface Iterator faz a filtragem dos elementos usando o mesmo critério. Assim basta passar o critério que pode ser construído através de uma expressão lambda e assim obter qualquer tipo de iterador. Por fim disponibiliza-se alguns métodos estáticos de conveniência como o **getDiaIterador**, **getMesIterador** e **getUltimosDiasIterador**.