UTFPR - Estrutura de Dados I

Profª Renata Luiza Stange Carneiro Gomes

Trabalho Final: Parte 1

1 Instruções

O trabalho pode ser realizado em dupla, porém o envio e a gravação do vídeo devem ser individual. Sobre colas e plágios respeite o **REGULAMENTO DISCIPLINAR DISCENTE**! Você pode discutir questões e estratégias de solução. geral com outros alunos, mas a avaliação que você enviar deve ser feita apenas por você. É sua responsabilidade manter a resolução em sigilo, pois no caso de detectado cola sua nota também não será considerada. Ao enviar sua solução, tenha o cuidado de seguir as instruções resumidas na Seção 2 deste documento. Leia toda a prova com atenção!

2 Check-List para envio da avaliação

A seguir está uma lista dos critérios mínimos necessários que sua avaliação deve atender para ser considerada satisfatória. Observe que, embora todos esses sejam necessários, atender todos eles ainda pode não ser suficiente para considerar sua submissão é satisfatória.

1. Os algoritmos devem ser implementados utilizando a linguagem de programação Java. Você deve entregar os arquivos `.java`, sem qualquer estrutura de pacote (package), inclusive no código.

2. Comente seu código! Se algum de seus métodos não for comentado corretamente, em relação ao propósito do método (a qual exercício ele corresponde) e parâmetros de entrada/saída, você perderá pontos por falta de documentação.

3. As simulações de execução e exemplos devem estar em um arquivo `pdf`.

4. Envie em um arquivo compactado os arquivos `.java` e um arquivo `.pdf`. O arquivo compactado deve conter o seu NOME.

3 FILA COM PRIORIDADES

Uma fila com prioridades ou filas priorizada (*priority queues*, ou *PQs*) é uma variação da fila "*padrão*" estudadas anteriormente na disciplina. Relembrando que, a fila é uma coleção de elementos gerenciados de maneira FIFO (*Fist In First Out*), ou seja, o primeiro elemento adicionado à coleção é sempre o primeiro elemento extraído; o segundo é o segundo, e assim por diante e assim por diante. Em alguns casos, uma estratégia FIFO pode ser muito simplista para a atividade que está sendo modelada. O pronto-socorro de um hospital, por exemplo, precisa agendar os pacientes de acordo com a prioridade. Um paciente que chega com um problema mais sério deve antecipar os outros, mesmo que eles tenham está esperando há mais tempo. Esta é uma fila com prioridades, onde os elementos são adicionados à fila em ordem arbitrária, mas quando chega a hora de extrair o próximo elemento, é o elemento de maior prioridade na fila que é removido.

3.1 Aplicações de filas priorizadas

Na computação, filas com prioridades têm um papel fundamental na implementação eficiente de diversos algoritmos importantes:

- Escalonamento de tarefas (jobs) pelo sistema operacional de um computador.
- Algoritmo Heapsort (que ordena um vetor).
- Algoritmo de Dijkstra (que encontra um caminho de peso mínimo de um vértice s a um vértice t em um digrafo).
- Algoritmo de Prim (que encontra uma árvore geradora de peso mínimo em um grafo).
- Algoritmo de Kruskal (que encontra uma árvore geradora de peso mínimo em um grafo).
- Algoritmo de Huffman (de compressão de arquivos).

3.2 Formalização do problema

Filas com prioridades são TADs que manipulam conjuntos de coisas comparáveis chamados de *chaves*, *itens* ou *prioridades*. Um item k é **máximo** se nenhum outro item é estritamente **maior** que k e **mínimo** se nenhum item é estritamente **menor** que k. Para distinguir entre uma fila com priordades de máximos ou de mínimos, diremos que a primeira é uma fila com prioridades **decrescente** (ou "*de máximos*") e a segunda é uma fila com prioridades **crescente** (ou "*de mínimos*"). Além disso, uma fila pode ter mais de um item máximo ou mais de um item mínimo. Uma fila com prioridades decrescente ou *PQ de máximo* é um TAD que manipula um conjunto de itens por meio das seguintes operações fundamentais sobre S:

- encontrar um elemento *máximo* de S,

- remover um elemento *máximo* de S,

- inserir um novo item em S,

Uma fila com prioridades crescente ou *PQ de mínimo* é definida de maneira análoga. Entretanto as operações buscam encontrar e remover um elemento *mínimo* de S. Veja a seguir, a interface que especifica as operações de uma *PQ de máximo*. Nela estão as assinaturas dos métodos que constituem a interface pública de uma fila com prioridades. Observe que `T` representa o tipo genérico que deverá ser definido de acordo com a aplicação, isto é, depende do tipo de dado que será colocado na fila com prioridades.

```
1  package utfpr.tsi.ed.assigment;
2
3  public interface PriorityQueue <T>{
4      /**
5       * Returns true if priority queue has no elements
6       *
7       * @return true if the priority queue has no elements
8       */
9      public boolean isEmpty();
10
11     /**
12      * Returns the number of elements in this priority queue.
13      *
14      * @return the number of elements in this priority queue.
15      */
16     public int size();
17
18     /**
19      * Returns the maximum element in the priority queue
20      *
21      * @return the maximum element
22      * @throws EmptyPQException
23      *             if priority queue contains no elements
24      */
25     public T findMax();
26
27     /**
28      * Inserts a new element into the priority queue. Duplicate
           values ARE
29      * allowed.
30      *
31      * @param x
32      *            element to be inserted into the priority queue.
33      */
34     public void insert(T obj);
35
```

```
36      /**
37       * Removes and returns the maximum element from the priority
             queue.
38       *
39       * @return the maximum element
40       * @throws EmptyPQException
41       *             if priority queue contains no elements
42       */
43      public T deleteMax();
44
45      /**
46       * Resets the priority queue to appear as not containing any
             elements.
47       */
48      public void makeEmpty();
49  }
```

3.3 Aspectos de implementação

Embora a **representação externa** (interface) possa dar a ilusão de que armazenamos os dados em ordem (crescente/descrecente) o tempo todo, a verdade é que temos uma boa quantidade de flexibilidade no que escolhemos como **representação interna** (estruturas de dados). Obviamente que, todas as operações da fila com prioridades precisam funcionar corretamente. Embora seja desejável que todas as operações sejam rápidas, nem sempre isto é possível. Buscamos otimizar não somente a **velocidade no acesso e execução das operações**, mas podemos melhorar a **facilidade de implementação** ou **minimizar o consumo de memória**. É possível escolher otimizar para uma operação em detrimento de outras, é necessário decidir o que será priorizado!

4 ENUNCIADO

Esta avaliação é sobre as expectativas do cliente da fila (classe de aplicação da fila), implementação e representação interna. Você deverá dominar o uso de *arrays* e *listas encadeadas* no processo, mas o ponto principal da tarefa - ou o mais importante dos muitos pontos principais - *é que você pode usar qualquer mecanismo que julgar apropriado para gerenciar as partes internas de uma abstração.* Nesta avaliação, você deverá implementar a fila com prioridades de duas maneiras diferentes. Uma mais simples e direta, mas a segunda forma não deve ser tão trivial. Você deverá ser capaz de avaliar qual a principal diferença das implementações em termos do que se busca otimizar em cada uma delas.

4.1 Definição dos problemas a serem implementados

Como comentado em aula, sobre como implementar tipos abastratos de dados (TAD - Fila, Pilha), esta avaliação pede que você implemente a interface *PriorityQueue* usando dois tipos diferentes representações subjacentes (estruturas de dados):

1. Uma **lista simplesmente encadeada** que armazena os elementos com prioridades.
2. Uma **estrutura de dados qualquer** que armazena os elementos com prioridade.

Atenção: Como no uso convencional do inglês, valores numéricos mais baixos correspondem a níveis mais altos de urgência, de modo que uma tarefa com prioridade 1 vem antes de uma tarefa com prioridade 2. Esta interpretação, infelizmente, pode causar confusão com uma frase como "A tem prioridade mais alta do que B" porque A terá, nesse caso, uma prioridade numérica inferior.

4.1.1 Implementação 1: (3 pts)

Nesta implementação, a importância é dada para a simplicidade da implementação.

- **Estrutura de dados:** você **deverá obrigatoriamente** utilizar **lista simplesmente encadeada** para armazenar itens na fila. Faça o que fizer, certifique-se de que implementa a interface `PriorityQueue<T>` fornecida e **NÃO** usa partes da estrutura de coleções Java (Vector, ArrayList, LinkedList, etc).
- **Ordenação:** os elementos da fila são inseridos em qualquer ordem, e não estão ordenados (a ordem depende de como as operações de manipulação dos dados foram implementadas).
- **Operações da fila de prioridade**: você **deverá obrigatoriamente** implementar os métodos fornecidos pela interface `PriorityQueue`, isto é, sua classe `PriorityQueueString` deverá incluir "`implements PriorityQueue<String>`".

  - **Método de busca:** a busca pelo elemento máximo deve ser uma **busca sequencial**, já que os elementos não estão ordenados.

- **Aplicação:** A fila com prioridades será uma coleção de cadeia de caracteres (strings). Strings lexicograficamente **menores** devem ser consideradas de **maior** prioridade do que as lexicograficamente **maiores**, de forma que "ping" tenha maior prioridade do que "pong", independentemente da ordem de inserção.

4.1.2 Implementação 2: (4 pts)

Nesta implementação você deverá ser criativo e implementar uma fila com prioridades de outra forma de sua escolha, isto é diferente da **Implementação 1**. Não é difícil imaginar maneiras de implementar uma fila com prioridades. Nas implementações mais óbvias, algumas das operações ficam rápidas mas as outras ficam lentas. O desafio é inventar uma implementação em que todas as operações sejam rápidas.

- **Estrutura de dados:** você **poderá** usar as estruturas de coleções Java (Vector, ArrayList, LinkedList, etc), ou se preferir poderá utilizar sua própria implementação de estruta de dados. A implementação mais simples possível pode usar apenas um arranjo (array), uma lista simplesmente encadeada ou uma uma lista duplamante encadeada. Faça o que fizer, certifique-se de que implementa a interface `PriorityQueue` fornecida.

  IMPORTANTE: Se optar por uma estrutura de coleções java, você deverá deixar explicar qual o tipo de estrutura de dados a coleção usa. Por exemplo: `LinkedList` em Java implementa uma uma lista duplamente encadeada. Eu sugiro a leitura do artigo *https://www.devmedia.com.br/diferenca-entre-arraylist-vector-e-linkedlist-em-java/29162* para auxiliar nesta escolha.

- **Ordenação:** os elementos da fila deverão ser inseridos em ordem de prioridade. Neste caso, o algoritmo deverá buscar a posição a qual o elemento deverá ser inserido.

  - **Método de busca:** a busca local onde o novo item será inserido deve ser uma **busca binária**, já que os elementos estão ordenados.

- **Operações da fila de prioridade**: você **deverá obrigatoriamente** implementar os métodos fornecidos pela interface `PriorityQueue<T>`, isto é, sua classe `PriorityQueueT` deverá incluir "`implements PriorityQueue<T>`", onde `T` dependo da aplicação e do tipo de items que sua aplicação pretende manter na fila. Além das operações da interface você deverá implementar as seguintes operações:

  - `changePriority(args)`: para aumentar ou diminuir a prioridade de um elemento de S.
  - `sort(args)`: para ordenar a fila com prioridades. Eesta operação deve ser executada sempre que a operação `changePriority(args)` foi executada. O

elemento cuja prioridade foi modificada, deverá ser realocado para a posição adequada na fila, conforme sua prioridade.

    * **Método de ordenação:** Utilize um método de ordenação, não necessariamente os vistos em visto em aula, para manter a fila com prioridades em ordem. Escolha um método que tenha com vantagem a eficiência para os casos onde o vetor está ordenado (ou parcialmente ordenado). Justifique a sua escolha!

- **Aplicação:** Você deverá ser criativo ao escolher uma aplicação para lista de prioridades e elaborar um exemplo que mostre a utilidade da sua fila com prioridades.

4.1.3 Implementação da classe Teste: (1 pts)

Você deverá implementar uma classe teste para cada uma das implementações da fila com prioridades. Esta classe deverá ser implementada de forma que seja possível testar:

- Todas as **operações** das filas com prioridades, isto é todos os métodos devem ser testados.
- Garantir que todas as **decisões lógicas** foram exercitadas nos dois sentidos (true/-false)
- Garantir que todos os **loops** foram exercitados nos seus valores de fronteira
- Garantir que as **estruturas de dados internas** foram exercitadas para assegurar a sua integridade

Você deverá ser bastante "caprichoso" com suas classes teste, de forma que o avaliador (professor) não precise criar novos testes para conseguir testar seus métodos e suas estruturas internas.

4.2 Definição do material complementar para explicar sua resolução (2 pts)

Elabore um material em vídeo explicando o passo a passo da execução da sua solução (o algoritmo em execução), este material deve mostrar que você entendeu os conceitos que estão sendo aplicados no exercício. A seguir algumas recomendações de como elaborar o vídeo. Observe que algumas delas são obrigatórias.

- O vídeo não tem limite de tempo, porém deve ser o suficientemente longo para explicar a sua solução em detalhes.
- Você deve demostrar a execução do seu programa (códigos na tela) e a se, a sua imagem (para que eu possa me certifcar de que foi você que gravou o vídeo) No caso de você não consiga incluir a sua imagem, você poderá ser chamado a apresentar a prova remotamente, se eu tiver dúvida sobre autoria do vídeo.
- Você pode enviar apenas o link do driver com o vídeo, caso ele seja grande demais para ser enviado pelo moodle.