|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **PREENCHA SEUS DADOS NOS CAMPOS COM X** |  | |
|  |
| Disc.: | **ECM306 – tópicos avançados em estrutura de dados** | |
| Curso: | **ENGENHARIA DA COMPUTAÇÃO** | |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Aluno: | | **X** | | | | | | | | |  |
| Curso: | **X** | | | Série: | **X** | Período: | **X** | RG: | **X** | |
| São Caetano do Sul, 27 de novembro de 2023. | | | | | | | | | RA: | **X** |
| Assinatura: | | |  | | | | | | Nota: |  |

**Instruções da Prova**

1. Esta prova é individual e prática, devendo ser realizada nos computadores do IMT, sendo permitido ao aluno, se assim desejar, utilizar seu próprio computador, sob sua inteira responsabilidade;
2. Não poderá haver acesso à Internet, sob nenhuma circunstância, exceto ao Open LMS do próprio aluno e, mesmo assim, somente em duas etapas: para receber (“baixar”) as questões de sua prova na máquina em uso; e para entregar (“subir”) as resoluções das questões de sua prova no devido local de entrega, na mesma plataforma.

**IMPORTANTE**: O aluno deverá informar ao professor quando fará os dois acessos permitidos a ele ao Open LMS, pela Internet;

1. Poderá haver consulta a qualquer material do próprio aluno, seja ele físico (livros, artigos, material de aula etc.) ou virtual (livros, artigos, material de aula, exercícios, resoluções etc.), desde que esse material esteja previamente armazenado em seu computador ou em qualquer dispositivo de armazenamento externo (*pendrive*, *hd* externo etc.).

**IMPORTANTE**: Não será permitido o acesso pela Internet a pastas compartilhadas (*Google* *Drive*, *OneDrive* etc.), nem a repositórios virtuais (*GitHub*, *GitLab*, *BitBucket* etc.), mesmo sendo de posse e administração do próprio aluno;

1. O aluno deverá responder às questões da prova no próprio arquivo da prova (.*docx*) devidamente identificado (RA e Nome completo do Aluno) e quando for o caso, gerando os códigos necessários, somente na linguagem de programação JAVA e utilizando sempre e somente o paradigma da Programação Orientada à Objetos visto nas aulas;
2. Para realizar a entrega da prova na plataforma Open LMS, o aluno deverá gerar e entregar um único arquivo compactado (*.rar* ou .*zip*), tendo seu RA e seu NOME completo como nome desse arquivo. Nesse arquivo compactado o aluno deverá fornecer, obrigatoriamente, os seus dados acadêmicos preenchidos na prova, bem como a resolução das questões nela solicitadas (.*docx*), além dos arquivos e códigos gerados em suas resoluções, uma pasta para cada questão, contendo as suas classes e demais arquivos que possibilitem sua posterior execução pelo professor, durante a resolução.

**IMPORTANTE**: Na correção, para executar o código gerado pelo aluno em sua prova, o professor seguirá exatamente as instruções fornecidas pelos alunos e contidas nas resoluções das provas! Caso não obtenha sucesso, a questão será considerada errada.

1. Não poderá haver troca de informações, nem de materiais, sejam físicos ou virtuais, entre os alunos durante a prova;
2. Não é permitido ao estudante se ausentar da sala antes da entrega da prova;
3. Celulares e outros equipamentos eletrônicos (exceto o notebook do aluno, se assim optar) devem permanecer desligados enquanto o estudante estiver na sala;
4. O tempo limite para realização da prova é de **90** minutos;
5. Mantenha sobre a carteira apenas um documento com foto, caneta, lápis e borracha;
6. O entendimento das questões faz parte da avaliação;
7. O tempo mínimo de permanência na sala é de **30** minutos;
8. O estudante que chegar atrasado em até **30** minutos do início da prova poderá fazê-la.

**Questão 1:** (*1 ponto*)

Segundo os autores M.C. Nicoletti e E.R. Hruschka Jr., em seu livro “Fundamentos da Teoria dos Grafos para Computação”, 3ª edição, de 2017, um grafo ***G*** = ( ***V***(***G***), ***E***(***G***) ) ou ***G*** = (***V***, ***E***) consiste de dois conjuntos finitos: ***V***(***G***), (ou ***V***), que é o conjunto de vértices do grafo, o qual é um conjunto **não** vazio de elementos chamados vértices; e ***E***(***G***), (ou ***E***), que é o conjunto de arestas do grafo, o qual é um conjunto (que pode ser vazio) de elementos chamados arestas e a cada aresta ***e*** em ***E*** atribui-se um par não ordenado de vértices (***u***, *v*) chamados vértices-extremidade de ***e***, sendo os Vértices também referenciados como pontos ou nós.

Sendo assim, com base na referida bibliografia e no material de aula utilizados, avalie as afirmações a seguir:

1. Se duas ou mais arestas de ***G*** têm os mesmos vértices-extremidade, essas arestas são chamadas arestas paralelas;
2. Dois vértices que estão unidos por uma aresta são chamados adjacentes ou vizinhos e duas arestas distintas ***ei*** e ***ej*** são adjacentes se têm um vértice em comum;
3. De acordo com a definição, que formalmente define o conceito de grafo, não é possível que o conjunto de arestas ***E*** seja vazio;
4. Um grafo cujo conjunto de arestas é vazio é chamado grafo ***nulo*** e a definição exige que o conjunto de vértices seja vazio.

**É correto, apenas, o que se afirma em:**

1. **I e II;**
2. **I e III;**
3. **II e III;**
4. **II e IV;**
5. **III e IV.**

**Questão 2:** (*1 ponto*)

Segundo os autores M.C. Nicoletti e E.R. Hruschka Jr., em seu livro “Fundamentos da Teoria dos Grafos para Computação”, 3ª edição, de 2017, muitos problemas em Teoria dos Grafos estão relacionados à possibilidade de se chegar a um vértice do grafo a partir de outro, seguindo-se uma sequência de arestas, sendo o passeio em um grafo uma sequência finita *w = v0 e1 v1 e2 v2 ... vk-1 ek vk* cujos elementos são, alternativamente, vértices e arestas tal que, para *1 ≤ i ≤ k*, a aresta ei tem vértices‑extremidades *vi-1* e *vi* .

Sendo assim, com base na referida bibliografia e no material de aula utilizados, avalie as afirmações a seguir:

1. O inteiro *k*, que é o número de arestas do passeio, é chamado comprimento de *w* e em um passeio não pode haver repetições de vértices e arestas;
2. No grafo *G* =(*V, E*), dados dois vértices *u ϵ V* e *v ϵ V* em *G*, um passeio *u – v* é fechado se *u*≠*v* e aberto se *u* = *v*;
3. Cada aresta *ei* é imediatamente precedida e sucedida pelos vértices aos quais é incidente e diz-se que o passeio *w* é um passeio *vo – vk* ou um passeio de *vo*até *vk* ; e
4. O vértice *vo* é chamado origem do passeio *w* e o vértice *vk* é chamado término de *w*, os vértices *vo* e *vk* não precisam ser distintos, sendo que os vértices *v1 ... vk-1*são chamados vértices internos.

**É correto, apenas, o que se afirma em:**

1. **I e II;**
2. **I e III;**
3. **II e III;**
4. **II e IV;**
5. **III e IV.**

**Questão 3:** (*1 ponto*)

Segundo os autores T.H. Cormen, C.E. Leiserson, R.L. Rivest e C. Stein, em seu livro “Algoritmos – Teoria e Prática”, 2ª edição, de 2002, Heap é uma estrutura de dados que pode ser visualizada como uma árvore binária quase completa, onde cada um de seus nós é ocupado por um elemento com as seguintes propriedades: a árvore é completa até o penúltimo nível; no último nível as folhas estão o mais à esquerda possível; e o conteúdo de um nó é maior ou igual ao conteúdo dos nós na subárvore enraizada nele (*max-heap*).

Levando-se em consideração esse contexto, a bibliografia adotada e o material de aula utilizado, avalie as seguintes asserções e a relação proposta entre elas:

I. A implementação do *heap*, pelas suas propriedades, não exige que se crie uma árvore, podendo se implementar um *heap* por meio de um vetor (*array*).

PORQUE

II. O filho a esquerda no nó com endereço (*i*) está na posição (*2 \* i*) e o filho a direita do nó com endereço (*i*) está na posição (*2 \* i + 1*).

**A respeito dessas asserções, assinale a única opção correta:**

**a) As asserções I e II são proposições falsas;**

**b) As asserções I e II são proposições verdadeiras e a II é uma justificativa da I;**

**c) As asserções I e II são proposições verdadeiras e a I é uma justificativa da II;**

**d) A asserção I é uma proposição falsa e a II é uma proposição verdadeira;**

**e) A asserção I é uma proposição verdadeira e a II é uma proposição falsa;**

**Questão 4:** (*1 ponto*)

Segundo o autor Reema Thareja, em seu livro “*Data Structures Using C*”, 1ª edição, de 2010, Tabelas de Dispersão, também chamadas de Tabelas *Hash*, quando bem projetadas, podem ser usadas para se buscar um dado em uma tabela em tempo constante *O(1)*, pagando o preço por isso de se utilizar um pouco mais de memória, implementando os *arrays* associativos ou dicionários (mapeamentos).

Levando-se em consideração esse contexto, a bibliografia adotada e o material de aula utilizado, avalie as seguintes asserções e a relação proposta entre elas:

I. Num *array* associativo, o acesso aos dados é feito de forma direta, uma vez que o índice do *array* coincide com a chave (*key*) e o tempo para isso é constante e a complexidade é *O*(*1*).

PORQUE

II. Num array associativo, conforme a premissa do problema, somente alguns elementos serão alocados no *array* do universo determinado pela chave (*key*), tendo a melhoria na eficiência da busca um preço de maior alocação de memória.

**A respeito dessas asserções, assinale a única opção correta:**

**a) As asserções I e II são proposições falsas;**

**b) As asserções I e II são proposições verdadeiras e a II é uma justificativa da I;**

**c) As asserções I e II são proposições verdadeiras e a I é uma justificativa da II;**

**d) As asserções I e II são proposições verdadeiras e não são justificativas uma da outra;**

**e) A asserção I é uma proposição verdadeira e a II é uma proposição falsa;**

**Questão 5:** (3 *pontos*)

Gerado por qualquer *IDE*, fornecer o código fonte, em *Java*, das classes necessárias (incluindo a de execução), implementadas sob o paradigma da *Programação Orientada a Objetos*, além do programa ***.jar*** que as executa diretamente, objetivando resolver o problema descrito a seguir.

ATENÇÃO:

1. Se o programa *.jar* fornecido não executar automaticamente a aplicação desenvolvida, a resolução da questão será invalidada (0 ponto);
2. As classes soluções desta questão (arquivos *.java*) deverão ser compactadas em um único arquivo (.*zip* ou .*rar*), em conjunto com o respectivo arquivo *.jar* funcional, além deste arquivo *.docx* da prova, contendo todas as respostas para as outras questões e a devida identificação do aluno e entregue em resposta à tarefa do *OPEN LMS* da prova.

Problema:

Utilizando-se apenas os códigos vistos e praticados em aula e os códigos desenvolvidos pelo próprio aluno para esta disciplina, através de uma aplicação implementada durante a prova, deseja-se criar uma **Estrutura de Dados – ED** dinâmica, baseada em **Filas de Prioridades – *Heap***, com raiz e folhas, capaz de armazenar um número inteiro por nó.

Essa ED deverá ser a base de um programa, a ser desenvolvido pelo aluno como solução para esta questão, capaz de controlar o atendimento de um consultório médico, onde deverá haver 4 níveis de prioridade de atendimento dos pacientes:

1. Prioridade Máxima:

Pacientes com 80 anos ou mais – Prefixo 1XXX, onde XXX deverá ser um número incremental e cronológico para o atendimento nesta prioridade;

1. Prioridade Alta:

Pacientes entre 60 anos e 79 anos – Prefixo 2YYY, onde YYY deverá ser um número incremental e cronológico para o atendimento nesta prioridade;

1. Prioridade Média:

Pacientes grávidas, puérperas ou acompanhada por crianças até 5 anos – Prefixo 3ZZZ, onde ZZZ deverá ser um número incremental e cronológico para o atendimento nesta prioridade;

1. Prioridade Baixa:

Qualquer outro tipo de paciente não especificado pelas prioridades 1, 2 ou 3 – Prefixo 4KKK, onde KKK deverá ser um número incremental e cronológico para o atendimento nesta prioridade;

O paciente, ao chegar no consultório, deverá pegar uma senha, de acordo com sua prioridade 1, 2, 3 ou 4. Essa senha, além do prefixo de sua prioridade, deverá conter o devido contador de pacientes relativos àquela prioridade, em fila, seguindo a numeração crescente após o prefixo.

O médico irá acionar o sistema para saber qual será o próximo paciente a ser chamado para a consulta, de acordo com as prioridades descritas anteriormente.

Exemplo: Um paciente 1003 deverá ser chamado pelo sistema antes do paciente 2001, que por sua vez deverá ser chamado antes do paciente 2002 e assim por diante.

Essa ED deverá nascer vazia a todo começo de atendimento do consultório (botão INÍCIO).

Após a chegada de cada paciente e informada sua respectiva prioridade (botões PRIORIDADE 1, 2, 3 ou 4), o sistema deverá gerar, armazenar na ED e apresentar em tela o número de chamada daquele paciente (SENHA), para sua ciência.

Após o médico solicitar ao sistema o próximo paciente (botão PRÓXIMO) para ser atendido, o sistema deverá apresentar em tela esse número de chamada (SENHA), levando em consideração as características da ED adotada, de acordo com suas prioridades, retirando-o a seguir da ED.

A ED obrigatoriamente deverá ser baseada em *Heap* e após cada inserção ou retirada de números de chamada, a ED precisará ser reorganizada de acordo com as regras de *Heap*.

Casos de Teste (sequenciais):

1. Após o comando de execução do programa, ao ser pressionado o botão PRÓXIMO, a ED baseada em Heap deverá informar que não há paciente para ser atendido (vazia);
2. Ao pressionar o botão PRIORIDADE 4, a senha do paciente 4001 deverá ser apresentada em tela;
3. Ao pressionar o botão PRIORIDADE 3, a senha do paciente 3001 deverá ser apresentada em tela;
4. Ao pressionar o botão PRIORIDADE 2, a senha do paciente 2001 deverá ser apresentada em tela;
5. Ao pressionar o botão PRIORIDADE 1, a senha do paciente 1001 deverá ser apresentada em tela;
6. Ao pressionar o botão PRIORIDADE 1, a senha do paciente 1002 deverá ser apresentada em tela;
7. Ao pressionar o botão PRIORIDADE 2, a senha do paciente 2002 deverá ser apresentada em tela;
8. Ao pressionar o botão PRÓXIMO, a senha do paciente 1001 deverá ser apresentada em tela;
9. Ao pressionar o botão PRIORIDADE 1, a senha do paciente 1003 deverá ser apresentada em tela;
10. Ao pressionar o botão PRÓXIMO, a senha do paciente 1002 deverá ser apresentada em tela;
11. Ao pressionar o botão PRÓXIMO, a senha do paciente 1003 deverá ser apresentada em tela;
12. Ao pressionar o botão PRÓXIMO, a senha do paciente 2001 deverá ser apresentada em tela;
13. Ao pressionar o botão PRÓXIMO, a senha do paciente 2002 deverá ser apresentada em tela;
14. Ao pressionar o botão PRIORIDADE 2, a senha do paciente 2003 deverá ser apresentada em tela;
15. Ao pressionar o botão PRÓXIMO, a senha do paciente 2003 deverá ser apresentada em tela;
16. Ao pressionar o botão PRIORIDADE 4, a senha do paciente 4002 deverá ser apresentada em tela;
17. Ao pressionar o botão PRÓXIMO, a senha do paciente 3001 deverá ser apresentada em tela;
18. Ao pressionar o botão PRÓXIMO, a senha do paciente 4001 deverá ser apresentada em tela;
19. Ao pressionar o botão INICIAR, deverá ser apresentada em tela “ED VAZIA”;
20. Ao pressionar o botão PRÓXIMO, deverá ser apresentada em tela a informação de que não há paciente para ser atendido (vazia);

**Questão 6:** (3 *pontos*)

Gerado por qualquer *IDE*, fornecer o código fonte, em *Java*, das classes necessárias (incluindo a de execução), implementadas sob o paradigma da *Programação Orientada a Objetos*, além do programa ***.jar*** que as executa diretamente, objetivando resolver o problema descrito a seguir.

ATENÇÃO:

1. Se o programa .jar fornecido não executar automaticamente a aplicação desenvolvida, a resolução da questão será invalidada (0 ponto);
2. As classes soluções desta questão (arquivos *.java*), deverão ser compactadas em um único arquivo (.*zip* ou .*rar*), em conjunto com o respectivo arquivo *.jar* funcional, além deste arquivo *.docx* da prova, contendo todas as respostas para as outras questões e a devida identificação do aluno e entregue em resposta à tarefa do *OPEN LMS* da prova.

Problema:

Utilizando-se apenas os códigos vistos e praticados em aula e os códigos desenvolvidos pelo próprio aluno para esta disciplina, através de uma aplicação implementada durante a prova, deseja-se criar uma **Estrutura de Dados – ED** dinâmica, baseada em ***Hashing***, capaz de armazenar os nomes dos alunos da sala em uma Tabela de *Hashing*, através de campos do tipo ***String***. A Chave *Hash* será a letra inicial do nome do aluno, sendo que a respectiva função *hash* deverá gerar um número de acordo com essa letra, estipulando assim o índice da tabela onde será alocado o nome. Eventuais **colisões** deverão ser resolvidas pelo método de **Hashing Aberto**, também conhecido por Encadeamento Separado.

A respectiva ED deverá ser projetada e construída a partir dos conceitos de Tabela Hashing vistos e estudados em aula, bem como o tratamento de suas colisões. Não poderá ser utilizada nenhuma implementação criada anteriormente pela própria Linguagem Java, devendo todo e qualquer código apresentado utilizar-se de ferramentas desenvolvidas pelo próprio aluno.

Após a inserção de cada nome de aluno, o aplicativo deverá apresentar a respectiva Tabela de Hash em tela, bem como o tratamento de suas colisões, para o usuário verificar como está sendo implementada a ED.

Casos de Teste (sequenciais):

1. Após o comando de execução do programa, a ED a ser apresentada deverá ser:

“VAZIA”

1. Ao inserir o nome “PAULO”, a ED apresentada será:

Índice Nome

15 PAULO;

1. Ao inserir o nome “ZULEIDE”, a ED apresentada será:

Índice Nome

15 PAULO;

25 ZULEIDE;

1. Ao inserir o nome “ANTONIO”, a ED apresentada será:

Índice Nome

0 ANTONIO;

15 PAULO;

25 ZULEIDE;

1. Ao inserir o nome “PEDRO”, a ED apresentada será:

Índice Nome

0 ANTONIO;

15 PAULO, PEDRO;

25 ZULEIDE;

1. Ao inserir o nome “PALOMA”, a ED apresentada será:

Índice Nome

0 ANTONIO;

15 PAULO, PEDRO, PALOMA;

25 ZULEIDE;

1. Ao solicitar a busca na Tabela de *Hashing* do nome “PAULO”, a ED deverá informar que localizou o respectivo nome em 1 (uma) tentativa;
2. Ao solicitar a busca na Tabela de *Hashing* do nome “ZULEIDE”, a ED deverá informar que localizou o respectivo nome em 1 (uma) tentativa;
3. Ao solicitar a busca na Tabela de *Hashing* do nome “PALOMA”, a ED deverá informar que localizou o respectivo nome em 3 (três) tentativas.