Prova II - Teoria

Entrega 23 mai em 10:30 **Pontos** 10 **Perguntas** 10
Disponível 23 mai em 8:30 - 23 mai em 10:30 2 horas
Limite de tempo 120 Minutos

Instruções

Nossa segunda prova de Algoritmos e Estruturas de Dados II tem duas partes: teórica e prática. Cada uma vale 10 pontos. A prova teórica será realizada no Canvas e a prática, no Verde.

A prova teórica tem 10 questões. A primeira é uma autoavaliação da sua preparação para esta prova (incluindo a qualidade da cola autorizada para esta prova) e vale 0,5 ponto. Em seguida, temos 4 questões fechadas, 2 de verdadeiro ou falso e 3 abertas. Cada fechada vale 0,4 pontos; cada verdadeira ou falso, 0,2 pontos; e cada aberta, 2,5 pontos. No caso das questões abertas, o aluno poderá enviar um arquivo contendo sua resposta ou entregá-la em na folha em branco disponibilizada pelo professor.

Abaixo, seguem as regras para a prova.
1) O código de acesso à prova será fornecido pelo professor no início da prova.
2) Após o envio de uma questão não é permitido que o aluno volte na mesma.
3) A prova é individual e é permitida a consulta à cola que contém o nome do aluno.
4) A interpretação faz parte da prova.
5) Se existir algum erro, após a divulgação do gabarito, peça a anulação da questão.
6) Os alunos da manhã 1/manhã farão a prova nos lab 1.
7) Os alunos da turma 2/manhã farão a prova nos lab 2.
8) Os alunos da tarde farão a prova no lab 11.

Desejamos uma excelente prova para todos.

Este teste foi travado 23 mai em 10:30.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	83 minutos	1,6 de 10

ⓘ As respostas corretas não estão mais disponíveis.

Pontuação deste teste: **1,6** de 10
Enviado 23 mai em 10:13

Esta tentativa levou 83 minutos.

Pergunta 1

0,5 / 0,5 pts

Autoavaliação sobre sua preparação para esta prova (mínimo 0 e máximo 0,5).

0,5

Incorreta

Pergunta 2

0 / 0,4 pts

Um ponteiro ou apontador é uma variável que armazena um endereço de memória. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Referente a alocação dinâmica de memória em C, a função clear é usada para limpar o conteúdo de um ponteiro.

II. Apos a sequência de instruções abaixo, se executarmos a instrução "*p = 7;", x passará a ter o valor 7.

```
int *p;
int x = 5;
p = &x;
```

III. A saída na tela abaixo será 24

```
int soma(int *a, int *b){
   *a = *a + *b;
   return *a;
}
int main(){
```

É correto o que se afirma em

- II e III. apenas.

II, apenas.

I e III, apenas.

I. II e III.

I, apenas.

Execute os dois códigos. No caso da primeira afirmação, em C, não existe a função clear e, sim, a free.

Incorreta

Pergunta 3

0 / 0,4 pts

Assumindo que o tamanho do int é igual a 4 bytes, mostre a saída na tela para o código abaixo:

```
#define R 10
#define C 20

int main()
{
   int (*p)[R][C];
   printf("%d",  (int)sizeof(*p));
   return 0;
}
```

200

80

4

800

Basta executar o programa. Lembre-se que consideramos o int com 4 bytes. Esta questão foi obtida no www.geeksforgeeks.org.

Incorreta

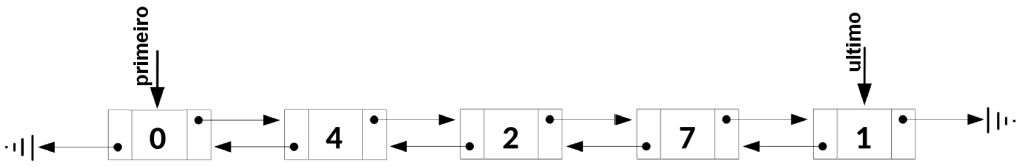
Pergunta 4

0 / 0,4 pts

Os conjuntos são tão fundamentais para a ciência da computação quanto o são para a matemática. Enquanto os conjuntos matemáticos são invariáveis, os conjuntos manipulados por algoritmos podem crescer, encolher ou sofrer outras mudanças ao longo do tempo. Chamamos tais conjuntos de conjuntos dinâmicos.

CORMEN T. H., et al., **Algoritmos: teoria e prática**, 3ed, Elsevier, 2012

A figura apresentada abaixo contém uma lista duplamente encadeada, um tipo de conjunto dinâmico.



O algoritmo abaixo é manipula os elementos de uma lista duplamente encadeada.

```
Celula *i = primeiro;
Celula *j = ultimo;
Celula *k;
while (i != j){
   int tmp = i->elemento;
   i->elemento = j->elemento;
   j->elemento = tmp;
   i = i->prox;
   for (k = primeiro; k->prox != j; k = k->prox); j = k;
}
```

Considerando a figura e o código acima e seus conhecimentos sobre listas duplamente encadeadas, avalie as afirmações a seguir:

I. A execução do algoritmo na lista faz com que o conteúdo da lista (da primeira para a última célula) seja: 1 7 2 4 0.

II. Na lista inicial, a execução do comando primeiro->prox->prox->prox->ant->elemento exibe na tela o número 2.

III. O algoritmo apresentado funciona corretamente quando nossa lista tem um número ímpar de células.

É correto o que se afirma em

- III, apenas.
- II, apenas.
- I, II e III.
- I e III, apenas.
- I e II, apenas.

I. CORRETA. O algoritmo em questão inverte a ordem dos elementos. A ordem do primeiro elemento é mantida dado que ele é o nó cabeça.

II. CORRETA. A execução do comando primeiro->prox->prox->prox->ant->elemento exibe na tela o número 2

III. ERRADA - A condição *i != j aborta o laço quando a quantidade de células é ímpar. A j->prox != i, quando essa quantidade é par.*

Incorreta

Pergunta 5

0 / 0,4 pts

Uma lista encadeada é uma estrutura de dados flexível composta por células sendo que cada célula tem um elemento e aponta para a próxima célula da lista. A última célula aponta para *null*. A lista encadeada tem dois ponteiros: primeiro e último. Eles apontam para a primeira e última célula da lista, respectivamente. A lista é dita duplamente encadeada quando cada célula tem um ponteiro anterior que aponta para a célula anterior. O ponteiro anterior da primeira célula aponta para *null*. A respeito das listas duplamente encadeadas, avalie as asserções a seguir.

I. Na função abaixo em C para remover a última célula, o comando "*tmp = NULL*" pode ser eliminado mantendo o funcionamento de nossa lista para outras operações. Contudo, isso pode gerar um vazamento de memória porque o espaço de memória relativo à célula ``removida'' só será liberado no final da execução do programa.

```
int removerInicio() {
   if (primeiro == ultimo) errx(1, "Erro!”);

   CelulaDupla *tmp = primeiro;
   primeiro = primeiro->prox;
   int elemento = primeiro->elemento;
   primeiro->ant = NULL;
   tmp->prox = NULL
   free(tmp);
   tmp = NULL;
   return elemento;
}
```

PORQUE

II. As linguagem C e C++ não possuem coleta automática de lixo como na linguagem JAVA. Essa coleta do JAVA é realiza pela Máquina Virtual Java que reivindica a memória ocupada por objetos que não são mais acessíveis.

A respeito dessas asserções, assinale a opção correta.

- A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.
- As asserções I e II são proposições falsas.

As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.

A asserção I é uma proposição falsa, e a asserção II é uma proposição verdadeira.

As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.

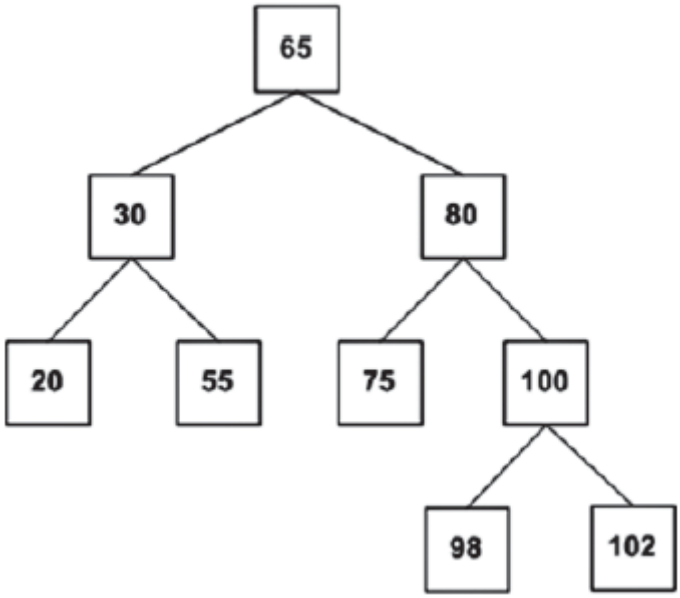
A primeira afirmação é falsa. Realmente, o comando *tmp = NULL*" pode ser eliminado mantendo o funcionamento de nossa lista para outras operações. Isso, porque a variável tmp é uma variável local cujo escopo de vida limita-se a função em questão. A remoção do comando citado não causa qualquer vazamento de memória.

A segunda afirmação é verdadeira dado que as linguagem C e C++ não possuem coleta automática de lixo e o Java possui.

Pergunta 6

0,2 / 0,2 pts

A estrutura de dados AVL é uma árvore binária de busca balanceada criada pelos soviéticos Adelson, Velsky e Landis em 1962. Podemos afirmar que a figura abaixo representa uma árvore AVL (PETROBRAS'12, adaptada).



- Verdadeiro
- Falso

A afirmação é verdadeira conforme o conceito de árvore AVL.

Pergunta 7

0,2 / 0,2 pts

Nas linguagens Java e C++, os atributos de uma classe podem ter as visibilidades public, private ou protected. Um atributo público é acessível dentro e fora da classe. Um privado, somente dentro da classe. Um protegido, somente dentro da classe ou de classes filhas.

- Verdadeiro
- Falso

As linguagens Java e C++ são orientadas por objetos e possuem os três tipos de visibilidade apresentados com suas respectivas definições.

io respondida

Pergunta 8

0 / 2,5 pts

Suponha uma árvore binária (não necessariamente de pesquisa) de caracteres e faça um método eficiente que retorna true se o caminho entre a raiz e alguma folha tiver duas vogais consecutivas. Em seguida, mostre a complexidade do seu método.

```
boolean busca(){
  return busca(raiz);
}
boolean condicao(char c){
  boolean resp;
  c = (c >= 'a' && c <= 'z') ? (char)(c - 32) : (c);
  return (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');
}
boolean busca(No i){
  boolean resp = false;
  if(i != null){
    resp = condicao(i.elemento) && ((i.esq != null &&
condicao(i.esq)) || (i.dir != null && condicao(i.dir)));
    if(resp == false){
      resp = busca(i.esq);
      if(resp == false){
        resp = busca(i.dir);
      }
    }
  }
  return resp;
} //O método faz O(n) visitas onde n é o número de nós da
árvore
```

Pergunta 9

0,3 / 2,5 pts

Seja nossa classe F**ila Flexível**, faça um método **R.E.C.U.R.S.I.V.O** que recebe três objetos do tipo **Fila** e mostra na tela a soma do primeiro elemento das três Filas, a do segundo, do terceiro e, assim, sucessivamente. Lembre-se que as filas podem ter tamanhos distintos.

Nesta questão você deve apresentar o código fonte da solução e explicá-lo via vídeo.

Sua Resposta:

```
class Fila{
  private Celula prim, ult;
```

```
  public Celula Fila(){
    prim = new Celula();
   Celula c1= new Celula();
   Celula c2 = new Celula();
   Celula c3 = new Celula();
    ult = prim;
}
 public Celula(Celula c1, Celula c2, Celula c3){
  this.c1 = c1;
  this.c2 = c2;
  this.c3 = c3;
}
int soma(){
return soma(c1,c2,c3);
}

int soma(Celula c1, Celula c2, Celula c3){
int resp = 0;
while( c1 != null || c2 != null || c3 != null){
 resp += c1.elemento + c2.elemento + c3.elemento;
}
}
```

Nesta questão o aluno deve *desenfileirar* e exibir um elemento de cada fila, intercalando. Assim, caso uma das fila fique vazia primeiro, a exibição deve continuar até que a outra estrutura fique vazia. Atenção as consistências que devem ser feitas para cada estrutura, verificando se a mesma esta ou não vazia.

Não respondida

Pergunta 10

0,4 / 2,5 pts

Assuma que você tem uma **Pilha Flexível** com as seguintes operações: *void empilhar(int i), int desempilhar(), boolean vazia()*. Como você pode usar essas operações para simular uma **Fila Flexível**. Em particular as operações *void enfileirar(int i), int desenfileirar()*? Apresente as implementações para essas duas operações. Dica: use duas pilhas, uma principal e outra temporária.

Nesta questão você deve apresentar o código fonte da solução e explicá-lo via vídeo.

Sua Resposta:

Nesta questão espera-se que o aluno implemente uma Fila com o auxilio de duas Pilhas para garantir a política **FIFO**. Assim, para **enfileirar**, o conteúdo da pilha principal deve ser movido para temporária e assim o elemento ser inserido na fila principal. Para **desenfileirar**, basta que ele desempilhe na pilha principal.

Pontuação do teste: **1,6** de 10