Prova II - Teoria

Entrega 24 mai em 10:40 **Pontos** 10,4 **Perguntas** 10
Disponível 24 mai em 8:40 - 24 mai em 10:40 aproximadamente 2 horas
Limite de tempo 120 Minutos

Instruções

Nossa segunda prova de AEDs 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 teórica será neste momento. A parte prática será realizada, amanhã, terça.

A prova teórica terá 8 questões fechadas no valor de 0,55 pontos e, em seguida, duas abertas no valor de 3 pontos cada. No total, distribuímos 0,4 pontos a mais. Após terminar uma questão, o aluno não terá a opção de retornar à mesma. No caso das questões abertas, o aluno poderá enviar um arquivo contendo sua resposta. Essa resposta pode ser uma imagem (foto de boa resolução) ou um código fonte. No horário da aula, você pode acessar a Prova II através do menu "Testes / Prova II".

Após a prova, para cada questão aberta, o aluno deve fazer um vídeo de no MÁXIMO 2 minutos explicando sua resposta. Vídeos com mais de 2 minutos serão penalizados. Cada vídeo deverá ser postado no YouTube e seu link submetido no Canvas até às 23:59 no dia de hoje. Vídeos alterados após essa data/horário serão zerados. Questões sem o vídeo serão penalizadas. Você pode acessar a submissão dos vídeos através do menu "Testes / Prova II - submissão de vídeos".

Este teste foi travado 24 mai em 10:40.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	103 minutos	9,3 de 10,4

ⓘ As respostas corretas não estão mais disponíveis.

Pontuação deste teste: **9,3** de 10,4
Enviado 24 mai em 10:23
Esta tentativa levou 103 minutos.

Pergunta 1	**0,55 / 0,55 pts**

Os conceitos de endereço e ponteiro são trabalhados de forma explícita na linguagem C. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Apos a sequência de instruções abaixo, *p é igual a 5.

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

II. A saída na tela abaixo quando o usuário digita 15 como entrada de dados será 10 15

```
int a = 5, *b, c = 10;
b = &a;
scanf("%d", b);
printf("%d %d", a, c);
```

iii. Referente a alocação dinâmica de memória em C, as funções calloc e realloc são usadas para liberar arays.

É correto o que se afirma em

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

Execute os dois códigos. No caso da terceira afirmação, em C, quem libera espaço é a função free.

Incorreta

Pergunta 2

0,55 / 0,55 pts

Ponteiros são variáveis que guardam o endereço de memória. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Apos a sequência de instruções abaixo, podemos afirmar que *p é

igual a 10

```
int *p;
int v[]={10,7,2,6,3};
p = v;
```

II. A saída na tela abaixo será -1 2 1

```
int funcao3 (int* a, int b){
   *a = *a-b;
   return *a+b;
}
int main(){
   int x = 1, y = 2;
   int z = funcao3(&x, y);
   printf("%d %d %d\n", x, y, z);
   return 0;
}
```

III. Dado o código abaixo, os números mínimo e máximo de acessos que precisam ser feitos para localizar um registro nessa estrutura contendo n elementos é 1 e n

```
typedef struct list {
   item_type item;
   struct list *next;
} list;
list *search_list(list *l, item_type x){
   return (l == NULL) ? NULL : ( (l->item == x) ? l : search_li
}
```

É correto o que se afirma em

- (●) I e III, apenas.
- ( ) II, apenas.
- ( ) II e III. apenas.
- ( ) I. II e III.
- ( ) I, apenas.

Execute os três códigos.

Pergunta 3

0,55 / 0,55 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
- 800
- 80
- 4

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 / 0,55 pts

A fila é uma estrutura de dados onde o primeiro elemento que entra é o primeiro a sair. Essa estrutura é usada quando desejamos que a ordem de remoção dos elementos seja igual aquela em que eles foram inseridos em nossa estrutura. Considere o código abaixo pertencente a uma Fila contendo nó cabeça e os ponteiros primeiro e último.

```
public Fila metodo(){
   Fila resp = new Fila();
      Celula i = this.primeiro.prox;
      Celula j = i.prox;

      while (j != null) {
         resp.inserir(j.elemento);
         i.prox = j.prox;
         j.prox = null;
         if (i.prox != null) {
            i = i.prox;
            j = i.prox;
         } else
            j = null;
      }
      this.ultimo = i;
   return resp;
}
```

Considerando o código acima, avalie as afirmações a seguir.

I. A execução do método mantém todos os elementos da fila inicial.

II. O método apresentado funciona corretamente quando a fila está vazia.

III. O objeto *resp* terá os elementos contidos nas posições pares de nossa fila, desconsiderando o nó cabeça.

É correto o que se afirma em

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

I. ERRADA - A execução do método faz com que a fila tenha somente os elementos que estavam nas posições ímpares da fila inicial, desconsiderando o nó cabeça.

II. ERRADA - Quando a fila estiver vazio, teremos um erro de execução no comando Celula j = i.prox.

III. CORRETA O objeto *resp* terá os elementos contidos nas posições pares de nossa fila, desconsiderando o nó cabeça.

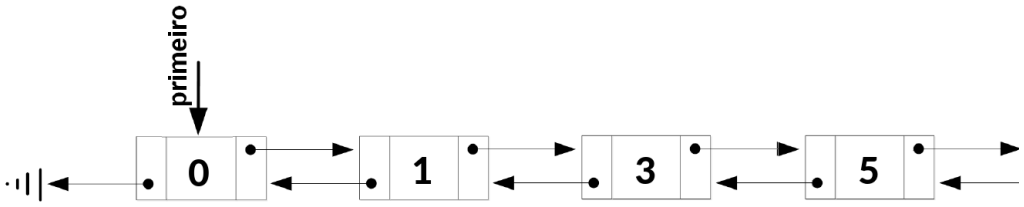
Pergunta 5

0,55 / 0,55 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->prox;
Celula *j = ultimo;
Celula *k;
while (i != j && j->prox != i){
   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: 0 7 5 3 1.

II. A condição *(i != j && j->prox != i)* permite que o código funcione quando nossa lista tem tamanho par ou ímpar.

III. Na lista original, a execução do comando primeiro->prox->prox->prox->prox->ant->elemento exibe na tela o número 7.

É correto o que se afirma em

- II, apenas.
- III, apenas.
- I e III, apenas.
- I, II e III.
- 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 condição *i != j aborta o laço quando a quantidade de elementos úteis é par. A j->prox != i, quando essa quantidade é par.*

III. ERRADA. A execução do comando primeiro->prox->prox->prox->prox->ant->elemento exibe na tela o número 5.

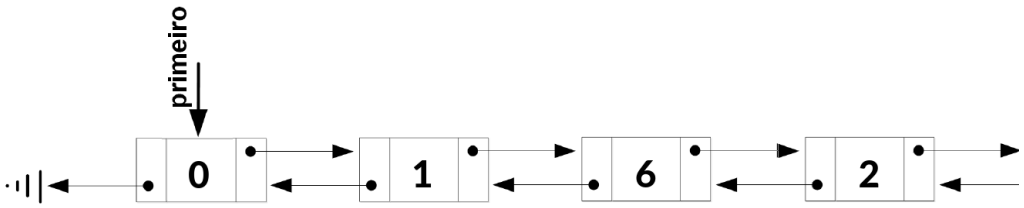
Pergunta 6

0,55 / 0,55 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->prox;
Celula *j = ultimo;
Celula *k;
while (j->prox != i){
   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: *0 0 2 6 1.*

II. Na lista inicial, a execução do comando primeiro->prox->prox->prox-

>ant->elemento exibe na tela o número 6.

III. Sabendo que a primeira célula é o nó cabeça, o algoritmo inverte as células úteis quando temos um número ímpar dessas células.

- II, apenas.
- I e III, apenas.
- I, II e III.
- I e II, apenas.
- III, 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 6

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.*dessas células.

Incorreta

Pergunta 7

0 / 0,55 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. No método abaixo em JAVA para remover a primeira célula, o comando "*tmp.prox = null*" pode ser eliminado sem qualquer prejuízo.

```
public int removerInicio() throws Exception {
   if (primeiro == ultimo) throw new Exception("Erro!");

   CelulaDupla tmp = primeiro;
   primeiro = primeiro.prox;
   int elemento = primeiro.elemento;
   primeiro.ant = null;
   tmp.prox = null;
   tmp = null;
   return elemento;
}
```

PORQUE

II. A Máquina Virtual Java realiza a coleta lixo automática, reivindicando 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.
- 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 I não é uma justificativa correta da I.
- 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.

As duas afirmações são verdadeiras e a segunda justifica a primeira. O comando *tmp.prox. = null* apenas desconecta a última célula da penúltimo. Isso é desnecessário porque a última célula não será mais referenciada e, como, explicado na segunda alternativa, a mesma será coletada pela JVM.

Pergunta 8

0,55 / 0,55 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.

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

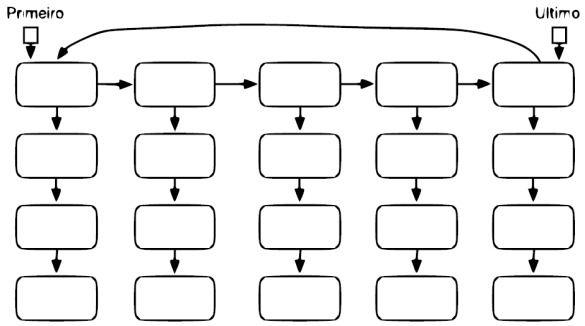
A primeira afirmação é falsa. Realmente, o comando *tmp = NULL*" pode ser eliminado mantendo o funcionamento de nossa lista para outras operações. Isso, porque a variável tmp é uma variável local cujo escopo de vida limita-se a função em questão. A remoção do comando citado não causa qualquer vazamento 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 9

3 / 3 pts

Conforme mostra a Figura, consideramos um Colar a junção de uma **Fila Circular Flexível** e uma **Pilha Flexível**. Nesta estrutura, a célula Último deve referenciar a Célula Primeiro, tornado a **Fila Circular**. E cada célula da **Fila** irá conter um ponteiro para o Topo de uma **Pilha** com n elementos. Pensando nisso, você deve implementar o construtor de sua classe Colar com o seguinte cabeçalho *Colar(int m, int n)*, onde m é a quantidade de elementos na **Fila** e n a quantidade de elementos em cada **Pilha**. Além disso, faça uma análise de complexidade de seu método.



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

Sua Resposta:

```
class CelulaDupla {
   // Atributos
   public int elemento;
   public CelulaDupla prox;
```

```
    public CelulaDupla inf;

    // Metodos especiais
    public CelulaDupla() {
        this.elemento = 0;
        this.prox = null;
        this.inf = null;
    }

    public CelulaDupla(int elemento) {
        this.elemento = elemento;
        this.prox = null;
        this.inf = null;
    }
}

class Celula {
    // Atributos
    public int elemento;
    public Celula prox;

    // Metodos especiais
    public Celula() {
        this.elemento = 0;
        this.prox = null;
    }

    public Celula(int x) {
        this.elemento = x;
        this.prox = null;
    }

}

class PilhaFlex {
    // Atributos
    public CelulaDupla topo;

    // Metodos especiais
    public PilhaFlex() {
        topo = null;
    }
```

```
    public PilhaFlex(int tamanho) {
        for (int i = 0; i < tamanho; i++) {
            CelulaDupla tmp = new CelulaDupla();
            tmp.inf = topo;
            topo = tmp;
            tmp = null;
        }
    }

    // Metodos e funcoes
    // Retorna o topo da pilha
    public CelulaDupla retornarTopo() {
        return this.topo;
    }
}

class Colar {
    //Atributos
    private CelulaDupla primeiro;
    private CelulaDupla ultimo;

    //Metodos Especiais
    public Colar(int m, int n) {
        primeiro = new CelulaDupla();
        PilhaFlex pilha = new PilhaFlex(n - 1);
        CelulaDupla topo = pilha.retornarTopo();
        primeiro.inf = topo;
        topo = null;

        ultimo = primeiro;
        for (int i = 0; i < m - 1; i++) {
            CelulaDupla tmp = new CelulaDupla();
            ultimo.prox = tmp;
            pilha = new PilhaFlex(n - 1);
            topo = pilha.retornarTopo();
            tmp.inf = topo;
            ultimo = tmp;
            tmp = null;
            topo = null;
            pilha = null;
        }
    }
}
```

```
public class Exercicio {
    public static void main(String[] args) {
        Colar colar = new Colar(5, 5);
    }
}
```

Analise de Complexidade:
Melhor caso= O(1)
Caso Médio = O(n)
Pior caso = O (n)

Pergunta 10

3 / 3 pts

Suponha a classe **Lista Flexível** vista na sala e crie o método **R.E.C.U.R.S.I.V.O** *boolean isFibonacciRecursivo(int i)* que recebe como parâmetro um contador *i* e retorna *true* quando os *n* elementos existentes na **Lista** correspondem aos *n* primeiros termos da sequência de Fibonacci. Observe que seu código não deve conter os comandos de repetição *for*, *while* e *do-while*.

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

Sua Resposta:

```
class Celula {
    // Atributos
    public int elemento;
    public Celula prox;

    // Metodos especiais
    public Celula() {
        this(0);
    }

    public Celula(int x) {
        this.elemento = x;
        this.prox = null;
    }

}
```

```
class  ListaSimples {
   // Atributos
   private Celula primeiro, ultimo;

   // Metodos especiais
   public  ListaSimples() {
      primeiro = new Celula();
      ultimo = primeiro;
   }

   // Metodos es funcoes
   public boolean isFibonacciRecursivo(int i) {
      boolean resp = true;

      //Tem que ser diferente de null
      if (primeiro.prox != null) {
         resp = resp && isFibonacciRecursivo(i, primeiro.prox, resp);
      } else {
         resp = false;
      }

      return resp;
   }

   // Retorna o elemento correspondente a sequencia de fibonacci na
   public int Fibonacci(int i) {
      if (i < 2) {
         return i;
      } else {
         return Fibonacci(i - 1) + Fibonacci(i - 2);
      }
   }

   public boolean isFibonacciRecursivo(int i, Celula verificar, boolean re
sp) {
      //Tem que ser diferente de null
      if (verificar != null) {
         if (Fibonacci(i) == verificar.elemento) {
            resp = isFibonacciRecursivo(++i, verificar.prox, resp);
         } else {
            resp = false;
         }
```

```
        }

        return resp;
    }

    // Inserir um elemento na lista
    public void inserirFim(int x ) {
        ultimo.prox = new Celula(x);
        ultimo = ultimo.prox;
    }
}

public class Teste {
    public static void main(String[] args) {
        ListaSimples lista = new ListaSimples();
        lista.inserirFim(1);
        lista.inserirFim(1);
        lista.inserirFim(2);
        lista.inserirFim(3);
        System.out.println(lista.isFibonacciRecursivo(1));
    }
}
```

Considerando a estrutura definida, o aluno deve implementar o mostrar com a verificação se os dois primeiros são valores **1** e se os demais correspondem a soma dos dois anteriores.

Pontuação do teste: **9,3** de 10,4