Prova II - Teoria

Entrega 8 de nov de 2021 em 10:40 **Pontos** 10 **Perguntas** 9
Disponível 8 de nov de 2021 em 8:40 - 8 de nov de 2021 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 prova teórica terá 9 questões sendo 6 questões fechadas e 3 abertas. Cada fechada vale 0.5 pontos; a primeira aberta, 3 pontos e as demais abertas, 2 pontos cada. Após terminar uma questão, o aluno **NÃO** terá a opção de retornar à mesma. No caso das questões abertas, o aluno deverá 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".

Este teste não está mais disponível, pois o curso foi concluído.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	100 minutos	0,5 de 10

ⓘ As respostas corretas não estão mais disponíveis.

Pontuação deste teste: **0,5** de 10
Enviado 8 de nov de 2021 em 10:40
Esta tentativa levou 100 minutos.

Pergunta 1 — 0,5 / 0,5 pts

Um dos principais desafios das linguagens C/C++ é o uso correto de ponteiros. Em relação aos ponteiros, avalie as afirmações a seguir.

I. Apos a sequência de instruções abaixo, podemos afirmar que *(v+3) é igual a 6

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

II. A saída na tela abaixo quando o usuário digita 15 como entrada de dados será 5 10

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

III. Apos a sequência de instruções abaixo, os valores de x e y são 2 e 3, respectivamente.

```
int x = 1;
int y = 2;
int *p = &x;
int *q = &y;
y = (*p)+++*q;
```

É correto o que se afirma em

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

Execute os três códigos.

Incorreta

Pergunta 2

0 / 0,5 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, II e III.
- II, apenas.
- I e 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.

io respondida

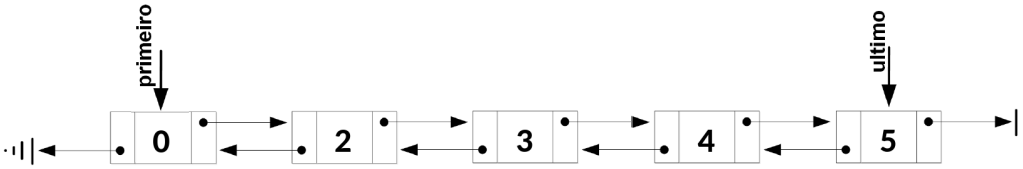
Pergunta 3

0 / 0,5 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 5 4 3 2.

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 4.

É correto o que se afirma em

- I e II, apenas.
- I, II e III.
- I e III, apenas.
- III, apenas.
- 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. CORRETA. A execução do comando primeiro->prox->prox->prox->prox->ant->elemento exibe na tela o número 4.

io respondida

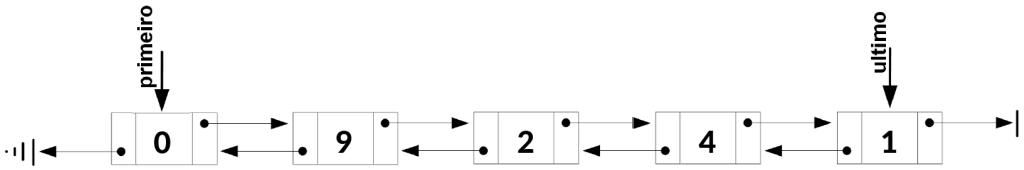
Pergunta 4

0 / 0,5 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 1 4 2 9.

II. Na lista original, a execução do comando primeiro->prox->prox->prox->ant->elemento exibe na tela o número 4.

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

É correto o que se afirma em

- I e III, apenas.
- III, apenas.
- I, II e III.
- I e II, apenas.
- 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. ERRADA. A execução do comando primeiro->prox->prox->prox->ant->elemento exibe na tela o número 2.

III. 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.*

io respondida

Pergunta 5

0 / 0,5 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 "*free(ultimo->prox)*" 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 removerFim() {
   if (primeiro == ultimo) errx(1, "Erro!”);

   int elemento = ultimo->elemento;
   ultimo = ultimo->ant;
   ultimo->prox->ant = NULL;
   free(ultimo->prox);
   ultimo->prox = 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 falsa, e a asserção II é uma proposição verdadeira.
- A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.
- As asserções I e II são proposições verdadeiras, mas a II 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 *free(ultimo->prox)* serve para eliminar uma célula que não é mais referenciada pela nossa estrutura. Como as linguagens C/C++ não possuem coleta de lixo, o programador é o responsável por essa tarefa. A falta dessa tarefa mantém o funcionamento das demais operações da lista, contudo, isso pode gerar um vazamento de memória.

Não respondida

Pergunta 6

0 / 0,5 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.

Não respondida

Pergunta 7

0 / 3 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 exatamente dois dígitos. Em seguida, mostre a complexidade do seu método.

```
boolean busca(){
  return busca(raiz, 0);
}
boolean condicao(char c){
  return (c >= '0' && c <= '9');
}
boolean busca(No i, int n){
  boolean resp = false;
  if(i == null){
    resp = (n == 2);
  } else {
    n += (condicao(i.elemento)) ? 1 : 0;
    resp = busca(i.esq, n);
    if(resp == false){
      resp = busca(i.dir, n);
    }
  }
  return resp;
} //O método faz O(n) visitas onde n é o número de nós da árvore
```

io respondida

Pergunta 8

0 / 2 pts

Considere a classe **Fila Flexível** de inteiros vista na sala de aula. Crie o método *void separar(Fila f1, Fila f2)* que recebe dois objetos do tipo **Fila f1** e **f2** e copia todos os números ímpares da Fila corrente para f1 e os pares para f2. Seu método não pode utilizar os métodos de inserir e remover existentes na fila.

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

Sua Resposta:

Esta questão exige que o aluno desenfileire um elemento da fila principal e teste se o resto da divisão do mesmo por 2 é igual a 0. Assim, ele enfileira em **f1** se for **0** e em **f2** se for **1,** apenas com o ajuste dos valores de ultimo de cada estrutura.

io respondida

Pergunta 9

0 / 2 pts

Usando uma **Pilha Flexível** e suas operações de *int empilhar(int num)* e *int desempilhar(), f*aça um algoritmo **R.E.C.U.R.S.I.V.O** que converta um número decimal em um número binário. Seu algoritmo deve manter uma complexidade linear $O(n)$.

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

Sua Resposta:

Pontuação do teste: **0,5** de 10