SISTEMAS OPERACIONAIS

(Respostas do professor)

1)

Estado do processo;

Valor do apontador de instruções;

Espaço de armazenamento dos valores de registradores;

Informações para escalonamento do processo;

Informações para o gerenciamento de memória;

Informações para contabilidade do processo;

2)

Swap é memória virtual que funciona como uma extensão da memória RAM, que fica armazenada no disco. A memória virtual ajuda a evitar falhas no sistema, uma vez que "amplia" a memória disponível para a execução de processos (disponibilizando memória para outros processos) através das trocas temporárias realizadas entre a memória principal e o armazenamento em disco.

3)

Sua resposta está correta.

A resposta correta é:

Técnicas podem ser aplicadas para permitir que a memória física seja estendida por outros espaços de armazenamento, como o disco..

4)

Sua resposta está correta.

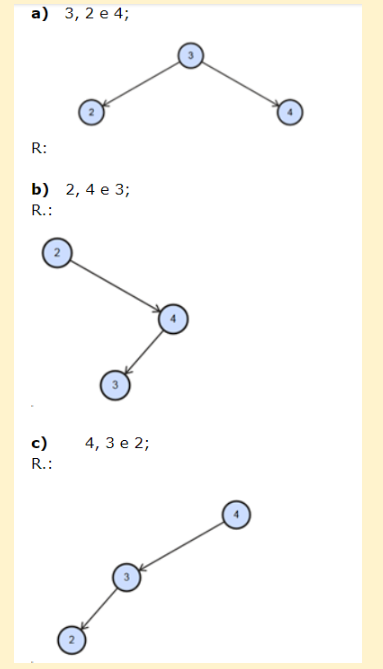
A resposta correta é:

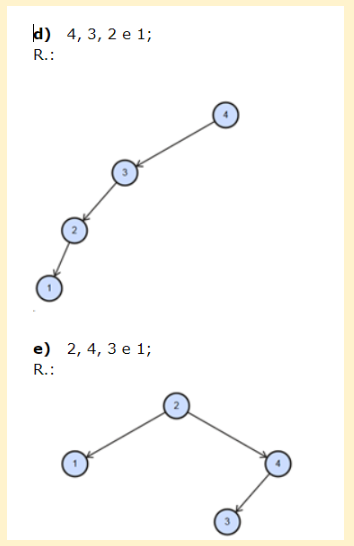
DMA.

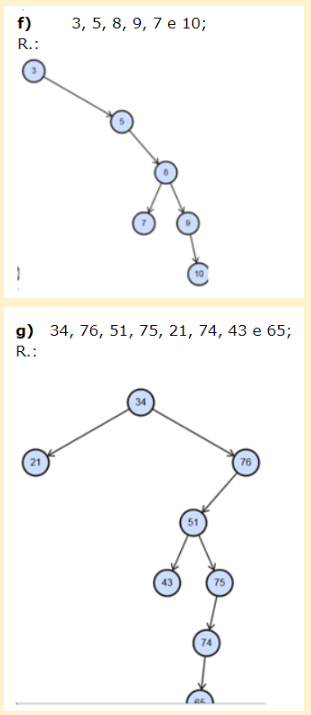
ESTRUTURA DE DADOS

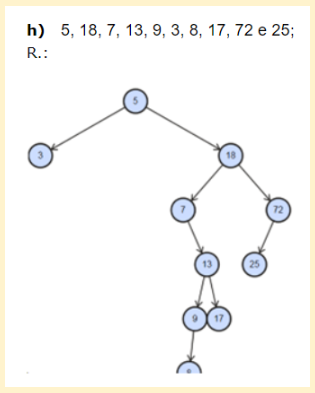
(Respostas do professor)

1)









2)

typedef struct no // estrutura que contém cada nó da árvore

{

int dado;

struct no\* esq;

struct no\* dir;

} no;

no\* criar\_no(int dado)

{

no \*novo\_no = (no\*)malloc(sizeof(no)); // aloca memória para o nó

if(novo\_no == NULL) // não foi possível criar o nó

{

fprintf (stderr, "Sem memória!!! (criar\_no)\n");

exit(1);

}

novo\_no->dado = dado; // inserimos o dado no nó e inicializamos os filhos

novo\_no->esq = NULL; // esquerda e

novo\_no->dir = NULL; // direita

return novo\_no;

}

typedef int (\*comparador)(int, int); // um comparador de valores inteiros

node\* insere\_no(no \*raiz, comparador compare, int dado)

{

if(raiz == NULL) // se não há um nó raiz, ele é criado

{

raiz = criar\_no(dado);

}

else // já há a raiz,

{

int e\_esq = 0;

int r = 0;

no\* cursor = raiz;

no\* ant = NULL;

while(cursor != NULL) // percorremos a árvore até a posição correta

{

r = compare(dado,cursor->dado);

ant = cursor;

if(r < 0) // valor menor, à esquerda

{

e\_esq = 1;

cursor = cursor->esq;

}

else if(r > 0) // valor maior, à direita

{

e\_esq = 0;

cursor = cursor->dir;

}

}

if(is\_left)

ant->esq = criar\_no(dado);

else

ant->dir = criar\_no(dado);

}

return raiz;

}

no\* deletar\_no(no\* raiz, int dado, comparador compare)

{

// da mesma forma que na inserção, buscamos o nó a ser apagado

if(raiz == NULL)

return NULL;

no \*cursor;

int r = compare(dado,raiz->dado);

if( r < 0)

raiz->esq = deletar\_no( raiz->esq, dado,compare);

else if( r > 0 )

raiz->dir = deletar\_no(raiz->dir,dado,compare);

else

{

if (raiz->esq == NULL)

{

cursor = raiz->dir;

free(raiz);

raiz = cursor;

}

else if (raiz->right == NULL)

{

cursor = raiz->esq;

free(raiz);

raiz = cursor;

}

else //2 filhos

{

cursor = raiz->dir;

no \*pai = NULL;

while(cursor->esq != NULL)

{

pai = cursor;

cursor = cursor->esq;

}

raiz->dado = cursor->dado;

if (pai != NULL)

pai->esq = deletar\_no(pai->esq, pai->esq->dado,compare);

else

raiz->dir = deletar\_no(raiz->dir, raiz->dir->dado,compare);

}

}

return raiz;

}

3)

Quando a exclusão for realizada em um nó que possui dois filhos, existem duas possibilidades de resolução. A primeira é a substituição do valor do nó, que deve ser excluído pelo valor sucessor, ou seja, o nó mais à esquerda da sub-árvore; e a segunda resolução consiste na substituição do valor do nó a ser excluído pelo valor do antecessor, que é o nó mais à direita da sub-árvore.

POO

1)  
Herança é um princípio da programação orientada a objetos onde uma classe (subclasse) pode herdar características (atributos e métodos) de outra classe (superclasse). Em Java, usamos a palavra-chave extends para definir uma subclasse que herda de uma superclasse.

2)  
Ao estender uma classe, a subclasse herda todos os métodos e atributos da superclasse, podendo sobrescrever métodos para alterar comportamentos e também adicionar novas funcionalidades específicas.

3)  
Coesão refere-se ao grau em que os métodos de uma classe estão focados em uma única responsabilidade. Acoplamento é o nível de dependência entre diferentes classes. Classes com alto acoplamento dependem fortemente umas das outras.

4)  
Em um bom projeto, queremos alta coesão (para que cada classe seja responsável por uma única tarefa) e baixo acoplamento (para reduzir dependências entre classes), facilitando manutenção, modificação e reutilização do código.

5)

public class Ponto2D {

private float x = 0.0f;

private float y = 0.0f;

public Ponto2D() {

}

public Ponto2D(float x, float y) {

this.x = x;

this.y = y;

}

public float getX() {

return x;

}

public void setX(float x) {

this.x = x;

}

public float getY() {

return y;

}

public void setY(float y) {

this.y = y;

}

public float[] getXY() {

return new float[] {x, y};

}

public void setXY(float x, float y) {

this.x = x;

this.y = y;

}

}

public class Ponto3D extends Ponto2D {

private float z = 0.0f;

public Ponto3D() {

super();

}

public Ponto3D(float x, float y, float z) {

super(x, y);

this.z = z;

}

public float getZ() {

return z;

}

public void setZ(float z) {

this.z = z;

}

public float[] getXYZ() {

return new float[] {getX(), getY(), z};

}

public void setXYZ(float x, float y, float z) {

setXY(x, y);

this.z = z;

}

}

public class TestaPontos {

public static void main(String[] args) {

Ponto2D p2d = new Ponto2D();

p2d.setX(1.5f);

p2d.setY(2.3f);

float[] xy = p2d.getXY();

System.out.printf("Coordenadas de p2d: x: %.1f, y: %.1f\n", xy[0], xy[1]);

Ponto3D p3d = new Ponto3D(3.0f, 5.1f, 2.1f);

p3d.setZ(4.0f);

float[] xyz = p3d.getXYZ();

System.out.printf("Coordenadas de p3d: x: %.1f, y: %.1f, z: %.1f\n", xyz[0], xyz[1], xyz[2]);

}

}