unesp

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

Faculdade de Ciências

1ª. Prova (P1)

18/Abril/2018

Questão 01		3,0
Questão 02		2,5
Questão 03		2,0
Questão 04		2,5
Total		**10,0**

Disciplina: Estruturas de Dados I – **ED1**
Professora: Dra Simone das Graças Domingues Prado

Nome:_____ **RA:** _____

(Questão 01) Na Questão 27 do POSCOMP (2009), e discutida na 2ª Prova do Método Trezentos, foi correto afirmar que uma fila poderia ser implementada usando duas pilhas e que uma pilha poderia ser implementada usando duas filas. Considere as estruturas e rotinas abaixo:

```
typedef struct no *def_pilha;
typedef struct no *def_fila;  // fila usando uma LCSE
struct no{      int dado;
                struct no*prox;};
```

```
void empilha(def_pilha* pilha, int numero){...}
int desempilha(def_pilha* pilha, int* numero){...}
void enfileira(def_fila* Final, int numero){...}
int desenfileira(def_fila* Final, int* numero){...}
```

a) **(1,5pt)** Escreva a rotina de **remoção de um elemento da "FILA"**, sabendo que esta **"FILA"** é manipulada por 2 pilhas. E após a inserção de valores na **"FILA"**, temos na realidade, os valores em uma pilha.

```
int desenfileira (def_pilha *P1, int *nro){
        def_pilha P2=NULL;   int x;
        if(vazia(*P1))  return 0;
        while(desempilha(P1,&x))        empilha(&P2,x);
        desempilha(&P2,nro);
        while(desempilha(&P2,&x))   empilha(P1,x);
        return 1;
}
```

b) **(1,5pt)** Escreva a rotina de **remoção de um elemento da "PILHA"**, sabendo que esta **"PILHA"** é manipulada por 2 filas. Após a inserção dos valores na **"PILHA"**, temos na realidade os valores em uma fila.

```
int desempilha (def_fila *F1, int *nro){
        def_fila F2=NULL;   int x;
        if(vazia(*F1)) return 0;
        while(desenfileira(F1,&x)) enfileira(&F2,x);
        while(desenfileira(&F2,&x)){
           if(vazia(F2)) *nro = x;
           else enfileira(F1,x);
        }
        return 1;
}
```

(Questão 02) Os percursos (pré-ordem, in-ordem e pós-ordem) que foram estudados servem somente para árvores binárias. Mas quando estamos trabalhando com árvores genéricas, também precisamos percorrer a árvore para encontrar algum elemento ou simplesmente para mostrar os elementos da árvore. Isso é usado muito em árvores de decisão em jogos. Há duas formas de percorrer uma árvore genérica: por largura e por profundidade. Veja a árvore abaixo

Departamento de Computação
Av. Engº Luiz Edmundo Carrijo Coube 14-01 Vargem Limpa Bauru-SP CEP: 17033-360
Fone: (14) 3103-6034 Fax: (14) 3103-6079 email: dcogeral@fc.unesp.br

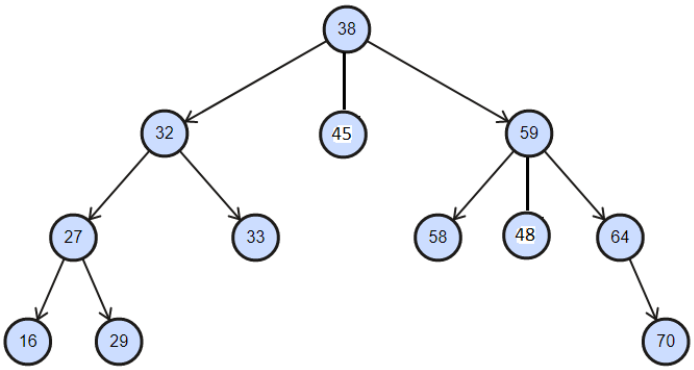


Figura 01. Árvore genérica.

Percurso em Largura: 38, 32, 45, 59, 27, 33, 58, 48, 64, 16, 29, 70
Percurso em Profundidade: 38, 32, 27, 16, 29, 33, 45, 59, 58, 48, 64, 70
Para implementar percurso em largura usamos Fila como estrutura auxiliar e para percurso em profundidade, usamos Pilha.
Usando a definição da árvore genérica de Grau 3 e de fila abaixo

```
typedef struct no{
    int info;
    struct no* primeiro;
    struct no* segundo;
    struct no* terceiro;
    } *def_arvore;
```

```
typedef struct no_fila {
    def_arvore dado;
    struct no_fila* prox;
    } *def_fila;
```

E sabendo que já estão prontas as rotinas de Filas (usando LCSE):

```
int vazia_fila(def_fila Final){....}
void enfileira(def_fila* Final,  def_arvore numero){ .... }
int desenfileira(def_fila* Final,  def_arvore* numero){ ... }
```

a) **(2,0pt)** Construa uma rotina que receba esta árvore genérica e mostre o **percurso em largura** da árvore. Pode ser só a impressão dentro da rotina.

```
void percurso_largura (def_arvore arvore){
    def_fila fila = NULL;
    def_arvore no_raiz;
    if(arvore==NULL) return;
    enfileira(&fila, arvore); printf("\n");
    while(fila!=NULL){
        desenfileira(&fila,&no_raiz); printf("\n - %d",no_raiz->info); getche();
        if(no_raiz->primeiro!=NULL) enfileira(&fila, no_raiz->primeiro);
        if(no_raiz->segundo!=NULL)enfileira(&fila, no_raiz->segundo);
        if(no_raiz->terceiro!=NULL)enfileira(&fila, no_raiz->terceiro);
    }}
```

b) **(0,5pt)** Se fosse percurso em profundidade? O que mudaria na sua rotina?
Manteria a resolução, mas usaria pilha como estrutura auxiliar.

```
void percurso_profundidade (def_arvore arvore){
    def_pilha pilha = NULL;
    def_arvore no_raiz;
    if(arvore==NULL) return;
```

```
empilha(&pilha, arvore); printf("\n");
while(pilha!=NULL){
        desempilha(&pilha,&no_raiz);
        printf("\n - %d",no_raiz->info); getche();
        if(no_raiz->terceiro!=NULL) empilha(&pilha, no_raiz->terceiro);
        if(no_raiz->segundo!=NULL) empilha(&pilha, no_raiz->segundo);
        if(no_raiz->primeiro!=NULL) empilha(&pilha, no_raiz->primeiro);
}}
```

(Questão 03) (2,0pt) Considere uma árvore binária. Faça uma rotina que busque um elemento (X) na árvore e devolva a soma de todos os elementos da subárvore de X. Por exemplo, se X = 32, então S = 32+26+66+11+27+57+74 = 293 (lembre-se que você pode fazer qualquer percurso, a soma de elementos independe da ordem deles). Faça as rotinas que precisar para resolver o problema. Não precisa ser uma só que resolva tudo (a busca e a soma)

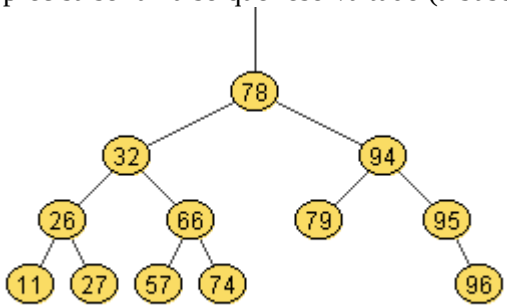
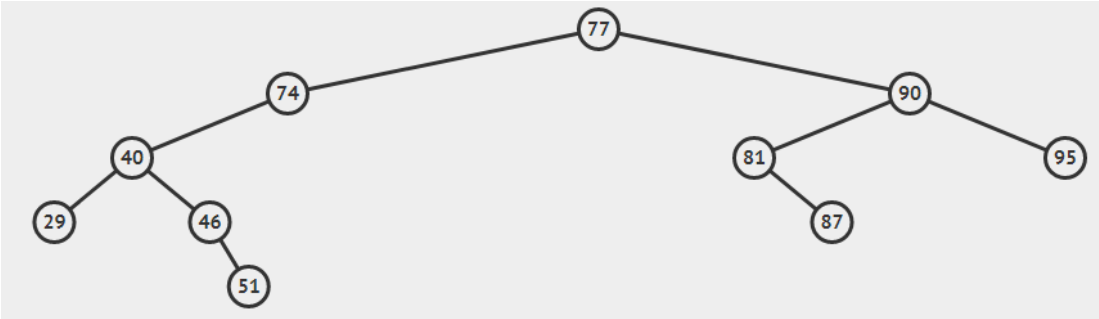


Figura 2. Uma árvore Binária

```
int soma_nos(def_arvore arvore){
  if(arvore==NULL) return 0;
  return(arvore->info + soma_nos(arvore->esq) + soma_nos(arvore->dir));
}
int busca_soma(def_arvore arvore, int valor){
    if (arvore==NULL) return 0;
    if (arvore->info == valor)  return soma_nos (arvore);
    return (busca_soma(arvore->esq,valor) +  busca_soma(arvore->dir,valor));
}
```

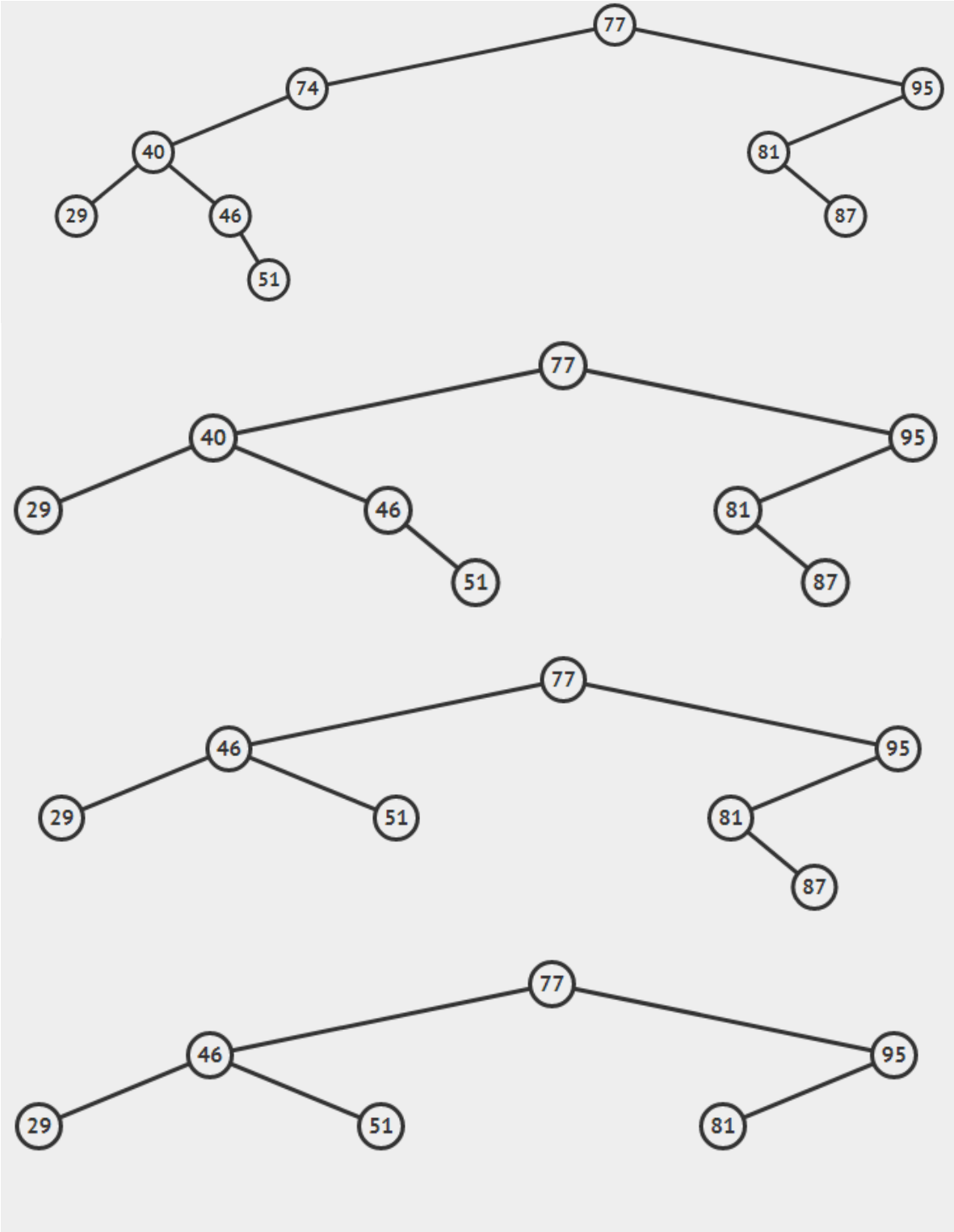
(Questão 04) Considere o uso de uma árvore binária de busca

(a) **(1,0pt)** Construa, de forma gráfica, a árvore binária de busca a partir da sequencia S = 77, 74, 40, 29, 46, 51, 90, 81, 95, 87;

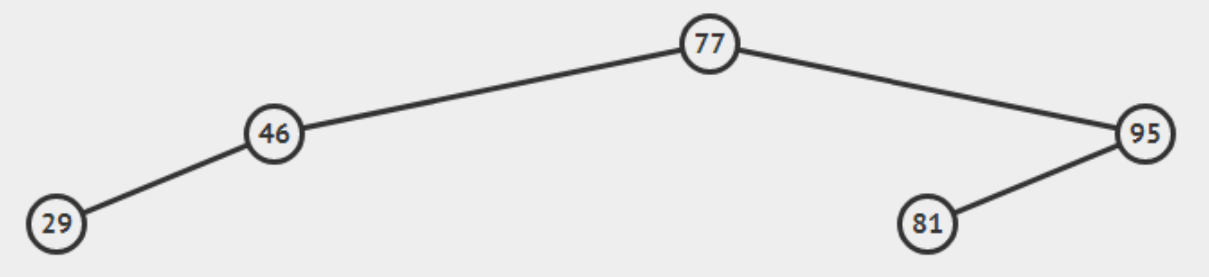


(b) **(0,5pt)** Mostre a sequencia obtida por um percurso pós-ordem;
Percurso : 29, 51, 46, 40, 74, 87, 81, 95, 90, 77

(c) (**1,0pt)** Retire os seguintes elementos, nesta ordem: 90, 74, 40, 87, 51.



Departamento de Computação
Av. Engº Luiz Edmundo Carrijo Coube 14-01 Vargem Limpa Bauru-SP CEP: 17033-360
Fone: (14) 3103-6034 Fax: (14) 3103-6079 email: dcogeral@fc.unesp.br



https://visualgo.net/bn/bst

Boa Prova!

Departamento de Computação
Av. Engº Luiz Edmundo Carrijo Coube 14-01 Vargem Limpa Bauru-SP CEP: 17033-360
Fone: (14) 3103-6034 Fax: (14) 3103-6079 email: dcogeral@fc.unesp.br