

Estruturas de Dados

Algumas Respostas da Lista 3.

J. L. Rangel

1. *provar que uma árvore binária de altura h tem, no mínimo, h+1 nós, e, no máximo, $2^{h+1} - 1$.*

(número mínimo) Se a árvore tem altura h, deve existir um caminho de comprimento h da raiz até um dos nós, digamos n_0, n_1, \dots, n_h , e todos os h+1 nós deste caminho devem ficar em níveis diferentes. Assim, a árvore deverá ter pelo menos h+1 nós.

(número máximo) Para que uma árvore tenha o número máximo de nós, todos os nós, exceto os do último nível, deverão ter dois filhos. Assim, a árvore terá 1 nó no nível 0 (o da raiz), 2 no seguinte, 4 no seguinte, e assim por diante. Portanto, o número total de nós de uma árvore com altura h e número máximo de nós deverá ser

$$1 + 2 + 4 + \dots + 2^h = 2^{h+1} - 1$$

3. *Modifique a programação de show, de forma que a saída impressa reflita, além do conteúdo de cada nó, a estrutura da árvore, usando a notação introduzida anteriormente. Assim, show teria a seguinte saída para a árvore do Exemplo 1 do Cap. 6:*

<a<b<><d<><>>><c<e<><>><f<><>>>>

A função show a que o exercício se refere é a da página 6-6,

```
void show(ARV a){
    if (!vazia(a)){
        printf("%c ", a->val);    /* mostra raiz */
        show(a->esq);             /* mostra SAE */
        show(a->dir);             /* mostra SAD */
    }
}
```

Com a modificação, teremos

```
void show(ARV a){
    printf("<");
    if (!vazia(a)){
        printf("%c", a->val);    /* mostra raiz */
        show(a->esq);            /* mostra SAE */
        show(a->dir);            /* mostra SAD */
    }
    printf(">");
}
```

9. *Escrever uma função para converter uma árvore binária em árvore.*

Suponha declarações de tipos BIN (árvore binária) e ARV (árvore)

```
typedef struct nob NOB, *PTB;
struct nob {
    int val;
    PT esq, dir;
}
typedef PTB BIN;
```

e

```

typedef struct noa NOA, *PTA;
struct noa {
    int val;
    PT prim, prox;
}
typedef PTA ARV;

```

Para criar nós da árvore, vamos usar também uma função

```

ARV cria(int v, PT pm) {
    PTA p=(PTA) malloc(sizeof(NOA));
    p->val=v;
    p->prim=pm;
    p->prox=NULL;
    return p;
}

```

Esta função cria nós do tipo NOA com os valores de val e prim fornecidos, devolvendo um apontador para o nó criado. A função auxiliar conv supõe que sua entrada é uma árvore binária não vazia.

```

ARV conv(BIN b) {
    PTA pe=NULL, pd=NULL;
    if (b->esq!=NULL)
        pe=conv(b->esq);
    if (b->dir!=NULL)
        pd=conv(b->dir);
    if (pe!=NULL) {
        pe->prox=pd;
        return cria(b->val, pe);
    }
    return cria(b->val,pd);
}

```

A função converte trata o caso da árvore vazia:

```

ARV converte(BIN b) {
    if (b==NULL)
        return NULL;
    return conv(b);
}

```

Q10. Escrever funções não recursivas para listar os conteúdos dos nós de uma árvore binária em pré-ordem, ordem simétrica e pós-ordem.

Vamos mostrar apenas uma solução para o caso da pré-ordem. Vamos usar o tipo BIN visto acima. No caso da pré-ordem, devemos visitar um nó (a raiz), descer para seu filho da esquerda, e depois descer para seu filho direito. Descer para o filho esquerdo não é problema, mas quando descemos pelo lado esquerdo, não temos como subir de novo para visitar o filho da direita. Para guardar os filhos da direita, vamos usar uma pilha auxiliar, que servirá para a volta, permitindo retirar os nós na ordem inversa.

A pilha deve ser uma pilha de PTB, com operações

```

void init(void);
PTB pop(void);
void push(PTB p);
int vazia(void);

```

(Note que não precisamos de um tipo “pilha de PTB”, mas apenas de uma “pilha de PTB”.) A versão não recursiva da função show pode então ser:

```
void show(BIN b) {
    PTB p;
    init();
    push(b);
    do {
        p=pop();
        while (p!=NULL) {
            printf("%d ",p->val);
            push(p->dir);
            p=p->esq;
        }
    } while (!vazia());
}
```

Q21. Escreva uma função com protótipo

```
int max(ARV a);
```

que calcule o valor máximo de todos os campos val da árvore a.

Esta função supõe que a árvore não é vazia, para evitar a necessidade de representar o valor $-\infty$, que é o máximo do conjunto vazio.

```
int max(ARV a) {
    int x=b->val;
    PTA p=a->prim;
    while (p!=NULL) {
        int y=max(p);
        if (y>x)
            x=y;
        p=p->prox;
    }
    return x;
}
```

Q24. Complete o exemplo do conjunto acrescentando funções para fazer união, interseção e diferença de conjuntos.

Vamos mostrar apenas a operação de união para o caso de conjunto representado por lista encadeada ordenada:

```
struct no {
    int elem;
    struct no *prox;
};
typedef struct no *SET;
```

usando a operação ins

```
void ins(SET *s, int v);
```

A operação de união pode ser

```

SET uniao(SET s1, SET s2) {
    SET s=NULL;
    SET p1=s1, p2=s2;
    while ((p1!=NULL) && (p2!=NULL)) {
        int v1=p1->val, v2=p2->val;
        if (v1==v2) {
            ins(&s,v1);
            p1=p1->prox;
            p2=p2->prox;
        } else if (v1<v2) {
            ins(&s,v1);
            p1=p1->prox;
        } else {
            ins(&s,v2);
            p2=p2->prox;
        }
    }
    /* aqui, p1==NULL ou p2==NULL */
    while (p1!=NULL) {
        ins(&s,p1->val);
        p1=p1->prox;
    }
    while (p2!=NULL) {
        ins(&s,p2->val);
        p2=p2->prox;
    }
    return s;
}

```

Esta solução percorre as duas listas, comparando seus elementos. Se os dois elementos são iguais, seu valor é inserido em *s*, e o avanço é feito nas duas listas; se são diferentes, o menor valor é incluído, e o avanço é feito apenas na lista correspondente.

Esta solução é ineficiente porque insere os elementos em *s* em ordem crescente, de forma que todas as vezes que *ins* é chamada tem de atravessar toda a lista *s*, de maneira que a inserção é feita sempre num tempo proporcional ao comprimento da lista. Uma solução melhor usaria um ponteiro para o fim da lista *s*, e faria a inserção diretamente no fim da lista, em tempo constante.

Q25. Descreva uma estrutura “lista de dados duplamente encadeada”, e escreva as funções para introduzir ou remover um elemento, supondo que introdução e inserção serão possíveis em qualquer uma das duas extremidades.

Podemos ter os tipos

```

typedef struct s_no *PT;
struct s_no {
    int val;
    PT ant, suc; /* antecessor e sucessor */
};
typedef struct s_l2 {
    PT ini, fim;
} *L2;

```

struct s_l2 é o tipo do nó cabeça da lista: uma estrutura que guarda ponteiros para o início e para o fim da lista propriamente dita. A lista vazia tem ini e fim com valor NULL.

As operações que vamos mostrar são a operação de criação da lista vazia (init), e as inserções e remoções nas duas extremidades (insI, insF, remI, remF).

```
L2 init(void) {
    L2 l=(L2)malloc(sizeof(struct s_l2));
    l->ini=NULL;
    l->fim=NULL;
    return l;
}
PT cria(int v) {
    PT p=(PT)malloc(sizeof(struct s_no));
    p->val=v;
    p->ant=NULL;
    p->suc=NULL;
    return p;
}
void insI(L2 l,int v) {
    PT p=cria(v);
    if (l->ini==NULL) {
        l->ini=p;
        l->fim=p;
        return;
    }
    p->suc=l->ini;
    l->ini=p;
}
void insF(L2 l,int v) {
    PT p=cria(v);
    if (l->ini==NULL) {
        l->ini=p;
        l->fim=p;
        return;
    }
    p->ant=l->fim;
    l->fim=p;
}
/* remI e remF supõem lista não vazia */
int remI(L2 l) {
    PT p=l->ini;
    int i=p->val;
    l->ini=l->ini->suc;
    l->ini->ant=NULL;
    free(p);
    if (l->ini==NULL)
        l->fim=NULL;
    return i;
}
```

```

int remF(L2 l) {
    PT p=l->fim;
    int i=p->val;
    l->fim=l->fim->ant;
    l->fim->suc=NULL;
    free(p);
    if (l->fim==NULL)
        l->ini=NULL;
    return i;
}

```

Q26. Complete o exemplo da “lista de LISP”, para permitir o uso de átomos que são cadeias de símbolos.

O tipo AL incluía apenas dois casos: int ou LIST. Agora devemos permitir também char *. Para isso, AL pode incluir um caso novo, tipo=2.

```

typedef struct s_al AL;
struct s_al {
    int tipo;      /* 0=átomo inteiro, 1=lista,
                    2=átomo cadeia */
    union {
        int atom;
        LIST list;
        char *cad;
    } al;
};

```

Podemos acrescentar uma função cadeia para construir um AL do tipo cadeia:

```

AL cadeia(char *s) {
    AL a;
    a.tipo=2;      /* char* */
    a.cad=s;
    return a;
}

```

e acrescentar um terceiro caso em show:

```

void show(LIST l) {
    PT p;
    printf("( ");
    for (p=l; p!=NULL; p=p->prox)
        switch(p->val.tipo) {
            case 0:      /* átomo */
                printf("%d ",p->val.al.atom);
                break;
            case 1:      /* lista */
                show(p->val.al.list);
                break;
            case 2:      /* cadeia */
                printf("\'%s\'",p->val.al.cad);
                break;
        }
    printf(") ");
}

```

(mai 00)