Árvores Binárias

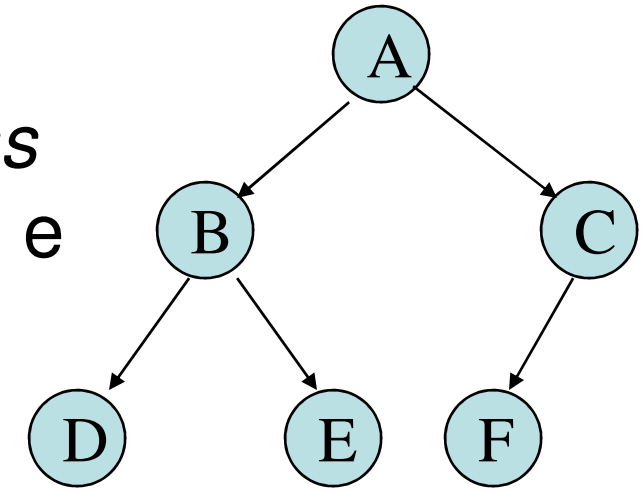
Fernando Vanini
IC - UNICAMP

Árvores Binárias

Definição:

Uma *árvore binária* A é um conjunto de objetos tais que

- A é vazio ou
- A consiste de um objeto R (raiz) e duas *árvores binárias* distintas E e D, disjuntas. (E e D são sub-árvores de A).



Árvores Binárias

Mais definições:

- *grau de um nó*: número de sub-árvores não vazias do nó.
- *nó terminal* ou *folha*: é um nó de grau igual a zero.
- numa árvore A = { R, E, D }, o nó R é *pai* dos nós que são as raízes das sub-árvores E e D. Estes por sua vez são *filhos* de R.
- *nível* de um nó:
  - a raiz tem nível 1
  - se um nó tem nível n então seus filhos têm nível n+1
- altura de uma árvore: é o nível máximo a um nó da árvore.

Árvores Binárias

Algumas propriedades quantitativas:

- O número máximo de nós de uma árvore binária de altura *h* é 2^h-1
- Número mínimo de nós de uma árvore de altura *n*: *n*
- Altura máxima de uma árvore com *n* nós: *n*
- Altura mínima de uma árvore com n nós: $\lceil \log_2(n+1) \rceil$

Implementação

Uma das formas mais comuns de se implementar árvores binárias é através de alocação ligada onde cada nó é representado por uma estrutura formada por:

- conteúdo do nó
- apontador para as raízes das sub-árvores esquerda e direita.
- Exemplo:

```
typedef struct no* apno;
typedef struct no {
          char* info;     /* conteúdo    */
          apno esq,dir;   /* sub-árvores */
        } no;
```

Percurso em árvores binárias

- Um *percurso* numa árvore binária é uma forma *sistemática* de acesso a todos os seus nós, em geral realizando algum tipo de operação.
- Duas categorias de percurso:
  - percurso em profundidade – os nós da sub-árvore atual têm prioridade na ordem de acesso.
  - percurso em largura – os nós de menor nível têm prioridade na ordem de acesso.

Percurso em profundidade

- Três tipos ‘canônicos’:
  - Pré-ordem:
    - Visitar a raiz
    - Percorrer a sub-árvore esquerda em pré-ordem
    - Percorrer a sub-árvore direita em pré-ordem
  - Pós-ordem:
    - Percorrer a sub-árvore esquerda em pós-ordem
    - Percorrer a sub-árvore direita em pós-ordem
    - Visitar a raiz
  - ‘In-ordem’ (simétrica):
    - Percorrer a sub-árvore esquerda em pré-ordem
    - Visitar a raiz
    - Percorrer a sub-árvore direita em pré-ordem

Exemplo: pré-ordem

```
void percorrePre(apno p){
  if(p != NULL){
     printf("%s",p->info);  /* visita raiz          */
     printf("(");
     percorrePre(p->esq);     /* sub-árvore esquerda */
     printf(",");
     percorrePre(p->dir);     /* sub-árvore direita */
     printf(")");
  }
}
```

Exemplo: pré-ordem (cont)

```
apno novoNo(char* ii, apno ee, apno dd){
  apno p = (apno)malloc(sizeof(no));
  p->info = ii;
  p->esq = ee;
  p->dir = dd;
  return p;
}

apno arvoreTeste(){
  apno p = novoNo("um",
                  novoNo("dois",NULL,NULL),
                  novoNo("tres",NULL,novoNo("quatro",NULL,NULL))
                 );
  return p;
}
```

Outro exemplo

- Árvore representando uma ‘expressão aritmética’.

```
/* árvore representando ((36/9)+(6*10)) */
apno arvoreTeste(){
   return novoNo("+",
                  novoNo(
                      "/",
                      novoNo("36",NULL,NULL),
                      novoNo("9",NULL,NULL)
                    ),
                    novoNo(
                      "*",
                      novoNo("6",NULL,NULL),
                      novoNo("10",NULL,NULL)
                    )
                 );
}
```

Outro exemplo (cont)

- Árvore representando uma ‘expressão aritmética’.

```
void imprimeExp(apno p){
  if(p != NULL){
    switch(p->info[0]){
      case '+':
      case '-':
      case '*':
      case '/':
                  printf("(");
                  imprimeExp(p->esq);
                  printf("%s",p->info);
                  imprimeExp(p->dir);
                  printf(")");
                  break;
      default:
                  printf("%s",p->info);
    }
  }
}
```

Outro exemplo (cont)

- Imprimir a ‘expressão aritmética’.

```
void imprimeExp(apno p){
  if(p != NULL){
    switch(p->info[0]){
      case '+':
      case '-':
      case '*':
      case '/':
                  printf("(");
                  imprimeExp(p->esq);
                  printf("%s",p->info);
                  imprimeExp(p->dir);
                  printf(")");
                  break;
      default:
                  printf("%s",p->info);
    }
  }
}
```

Outro exemplo (cont)

- Calcular o valor da ‘expressão aritmética’:

```
int calcExp(apno p){
  if(p == NULL) return 0;
  switch(p->info[0]){
    case '+': return calcExp(p->esq) + calcExp(p->dir);
    case '-': return calcExp(p->esq) - calcExp(p->dir);
    case '*': return calcExp(p->esq) * calcExp(p->dir);
    case '/': return calcExp(p->esq) / calcExp(p->dir);
    default : return atoi(p->info);
  }
}
```

Percurso em largura

```
void visitaEmLargura(apno p){
  apno q;
  apfila f = NULL;          /* fila auxiliar       */
  insereF(&f,p);            /* insere raiz na fila */
  do{
    q = removeI(&f);        /* retira nó q da fila */
    if(q != NULL) {         /* q é não vazio ?    */
      visita(q->info);
      insereF(&f,q->esq); /* insere sub-árvore esq. na fila */
      insereF(&f,q->dir); /* insere sub-árvore dir. na fila */
    }
  }while(f != NULL);
}
```

Pergunta: O que acontece se ao invés de uma fila, for utilizada uma pilha ?

Árvore binária de busca

- Uma árvore binária de busca é uma árvore binária na qual
  - o conteúdo dos nós da sub-árvore esquerda é menor que o conteúdo da raiz.
  - o conteúdo dos nós da sub-árvore direita é maior que o conteúdo da raíz.
  - as sub-árvores esquerda e direita são árvores binárias de busca.

Inserção numa árvore binária de busca

```
void insere(apno* abb, char* nome){
  apno p = novoNo(nome,NULL,NULL);
  int cmp;
  if(*abb == NULL) *abb = p;
  else {
         cmp = strcmp(nome,(*abb)->info);
         if(cmp < 0) insere(&((*abb)->esq),nome); else
         if(cmp > 0) insere(&((*abb)->dir),nome);
       }
}
```

Busca por um valor

```
apno busca(apno abb, char* nome){
  int cmp;
  if(abb==NULL) return NULL;
  cmp = strcmp(nome,abb->info);
  if(cmp == 0) return abb;
  if(cmp < 0)  return busca(abb->esq,nome);
  return busca(abb->dir,nome);
}
```

- Exercício: escrever as versões não recursivas das funções **`insere()`** e **`busca()`**.