Árvores

Paulo José da Silva e Silva

2 de maio de 2001

1 Conceitos básicos

Definição 1.1. Uma *árvore* é um conjunto não vazio T de nós tal que:

1. Existe um nó especial chamado *raiz*.
2. Os nós restantes são particionados em $m \geq 0$ conjuntos disjuntos $T_1, \ldots, T_m$ cada um dos quais é uma árvore, chamadas de *sub-árvores* da raiz.

Dadas duas árvores:

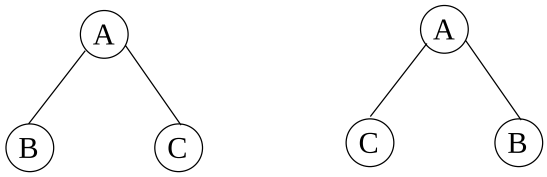

Figura 1: Elas são iguais?

- Árvores ordenadas: considera-se que a ordem das sub-árvores importa. Neste caso as árvores da Figura 1 são diferentes.
- Árvores orientadas: Não se considera que a ordem das sub-árvores importa. Neste caso as árvores da Figura 1 são iguais.

Representação gráfica de uma árvore

Mais definições e terminologia

Definição 1.2. O *Grau de um nó* é o número de sub-árvores desse nó.
Ex: $g(A) = 3$, $g(C) = 1$ e $g(L) = 0$.

Definição 1.3. Um *nó terminal ou folha* é um nó de grau zero.
Ex: K, L, F, G, M, I, J.

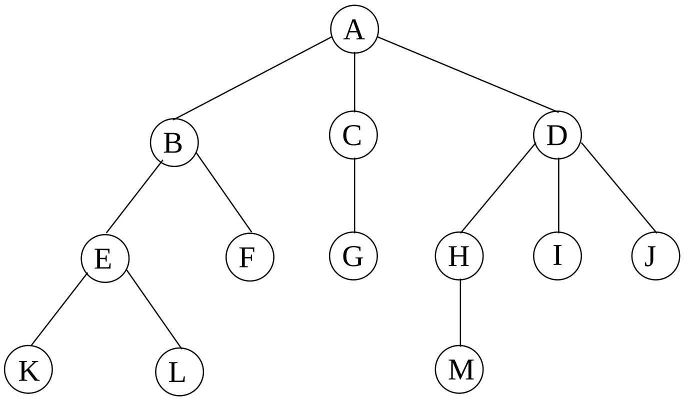

Figura 2: Representação de uma árvore

Definição 1.4. Os *filhos de um nó* são as raízes de suas sub-árvores. Neste caso o nó é chamado de *pai* das raízes.

Ex: Os filhos de D são H, I e J. Os filhos de E são K e L. Já G não tem filhos.

Definição 1.5. Um *caminho na árvore* é uma seqüência $x_1, \ldots, x_n$ de nós tais que para todo $i = 1, \ldots, n-1$, x_i é pai de x_{i+1}.

Ex: G; A, C, G e D, H, M.

Definição 1.6. Os *ancestrais de um nó* são os nós no caminho que vai da raiz até esse nó.

Ex: Os ancestrais de K são A, B e E. Os ancestrais de H são A e D.

Definição 1.7. Os *Descendentes de um nó* são todos os nós de suas sub-árvores.

Ex: descendentes de B são E, F, K e L. O descendente de H é somente o M.

Definição 1.8. O *nível de um nó* é dado por:

1. A raiz da árvore tem nível zero;
2. Se um nó tem nível l, todos os seus filhos têm nível $l + 1$.

Ex: O nível de E é 2.

Definição 1.9. A *profundidade* ou *altura* de uma árvore é o máximo dos níveis de seus nós.

Ex: A altura da árvore na Figura 1 é 3.

Definição 1.10. Uma *floresta* é um conjunto de árvores.

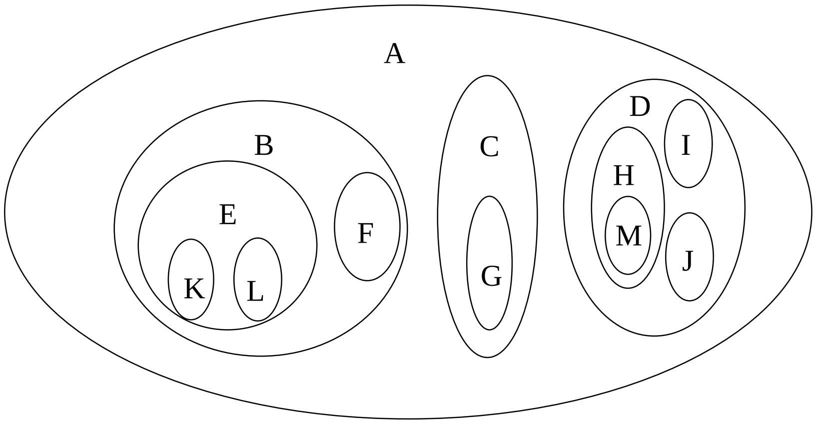

Figura 3: Diagrama de VENN

Outras representações de árvores

1. Diagrama de VENN:
2. "Por endentação":

```
A
    B
        E
            K
            L
        F
    C
        G
    D
        H
            M
        I
        J
```

3. (raiz $(T_1, \ldots, T_m)$):
   Ex: (A(B(E(K,L),F),C(G),D(H(M),I,J)))

2 Como representar árvores no computador

Usando *alocação ligada*. A forma mais comum é considerar que sabemos o número máximo de filhos de um nó:

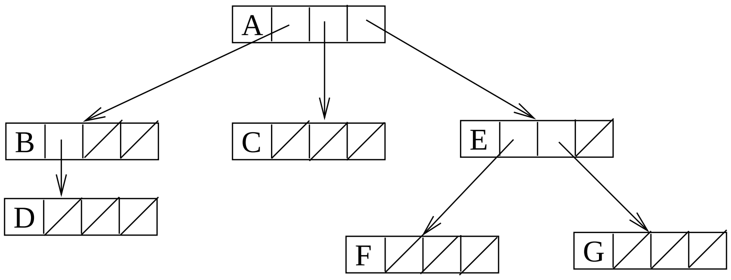

Figura 4: Alocação ligada

Caso não saibamos um número máximo para a quantidade de filhos podemos usar uma lista ligada de ponteiros para guardar os filhos de um nó.

É claro que há muito outras formas...

3 Árvores binárias

Definição 3.1. Uma *árvore binária* é um conjunto finito de nós que podem ser:

1. Um árvore vazia;
2. Uma raiz e duas sub-árvores binárias, chamadas de sub-árvore à direita e sub-árvore à esquerda da raiz.

Note que aceitamos árvores vazias e consideramos que árvores binárias são sempre ordenadas.

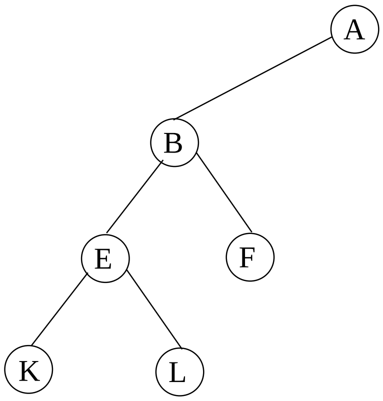

Figura 5: A sub-árvore à direita de A é vazia, já á sub-árvore à esquerda de B é composta dos nós E, K e L.

Note que uma árvore binária tem no nível i no máximo 2^i nós. Logo, uma árvore binária de altura h tem até $2^{h+1} - 1$ nós (basta usar a fórmula da soma dos termos de uma PG para ver isso).

3.0.1 Varredura de uma árvore

Uma das tarefas mais comuns que devem ser executas numa árvore é a aplicação de uma operação P a todos os elementos dela. Para isso precisamos de um algoritmo que visite todos os elementos de uma árvore. Existem três maneiras principais de ser varrer uma árvore binária:

1. Pré-ordem: Aplica-se a operação P à raiz e depois visita-se as sub-árvores esquerda e direita da raiz (nesta ordem);

2. In-ordem: Visita-se a sub-arvore à esquerda da raiz, aplica-se P a raiz e depois visita-se a sub-árvore direita;

3. Pós-ordem: Visita-se a sub-arvore esquerda, seguida da direita e por fim aplica-se a operação P à raiz.

Por exemplo, usando-se a árvore da Figura 3 temos:

1. Pré-ordem: A B E K L F

2. In-ordem: K E L B F A

3. Pós-ordem: K L E F B A

Vamos para um pouco de código:

```
/* Estruturas para uma arvore de inteiros. */
typedef struct _no {
  int chave;
  /* Outros campos de dados aqui */
  struct _no *esq, *dir;
} no;

void P(no *p) {

  printf("%d ", p->chave);
}

void pre_ordem(no *raiz) {
  if (raiz != NULL) {
    P(raiz);
    pre_ordem(raiz->esq);
    pre_ordem(raiz->dir);
  }
}
```

```
void in_ordem(no *raiz) {
  if (raiz != NULL) {
    in_ordem(raiz->esq);
    P(raiz);
    in_ordem(raiz->dir);
  }
}

void pos_ordem(no *raiz) {
  if (raiz != NULL) {
    pos_ordem(raiz->esq);
    pos_ordem(raiz->dir);
    P(raiz);
  }
}
```

Note que aqui a operação P a ser executada é representada por uma chamada de função.

3.1 Árvore binária de busca

Uma *árvore binária de busca* é uma árvore organizada de tal forma que para cada nó t_i todas as chaves dos nós da sub-árvore à esquerda de t_i sejam menores que a chave de t_i e todas as chaves da sub-árvore à direita de t_i sejam maiores que a chave de t_i.

Neste caso é fácil fazer a busca de um elemento na árvore:

```
/* Busca x em uma arvore binaria de busca. */
no *busca(int x, no *raiz) {
  while ((raiz != NULL) && (raiz->chave != x)) {
    if (raiz->chave < x) raiz = raiz->dir;
    else raiz = raiz->esq;
  }
  return raiz;
}
```

Para criar uma árvore binária de busca precisamos apenas de uma rotina que insira um elemento nela obedecendo sua estrutura de ordenação. A forma mais fácil de fazer isto é o uso de uma rotina recursiva, afinal árvores são estruturas recursivas por excelência. A seguir apresentamos de inserção de um novo elemento com chave x.

```
void insere(int x, no **p) {

  /* Se a arvore esta vazia insere. */
  if (*p == NULL) {
```

```
    *p = (no *) malloc(sizeof(no));
    (*p)->chave = x;
    /* Caso houvesse outros dados eles deveriam ser copiados aqui. */
    (*p)->dir = (*p)->esq = NULL;
  }
  else {
    if (x <= (*p)->chave) insere(x, &(*p)->esq);
    else insere(x, &(*p)->dir);
  }
}
```

Por fim, uma rotina para imprimir uma árvore binária na forma endentada é apresentada é apresentada abaixo. Ela pode ser muito útil na hora da depuração de um programa.

```
void imprime(no *raiz, int nivel) {
  /* Esta rotina e adequada para imprimir arvores com chaves de */
  /* comprimento maximo igual a 6.                               */

  int i;

  if (raiz == NULL) return;

  imprime(raiz->dir, nivel + 1);

  for(i = 0; i < nivel; i++) printf("      ");
  printf("%6d\n", raiz->chave);

  imprime(raiz->esq, nivel + 1);
}
```

3.2 Exercício

1. Escreva um programa que lê uma seqüência de inteiros positivos terminada por zero e imprime uma lista com todos os inteiros lidos com o número de vezes que estes apareceram. Para guardar a lista você *tem* que usar uma árvore binária de busca. Além disto, você deve alterar os procedimentos de busca e inserção apresentados acima para permitir a busca e atualização do número de ocorrências de um inteiro, com eventual inserção de um novo elemento, em apenas uma varredura da árvore.

2. A organização de uma árvore binária de busca faz, implicitamente, uma ordenação dos dados lidos. Como faria para imprimir os dados armazenados ordenados?

Referências

[1] Notas de aula da Nami.

[2] Wirth, Niklaus. *Algoritmos e Estrturas de Dados*, Prentice Hall do Brasil, 1986.