Aula 6: Árvores

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA

Árvores

- Nas aulas anteriores examinamos as estruturas de dados lineares (vetores e listas).
- Problema: Estruturas lineares não são adequadas para representar dados que devem ser dispostos de maneira hierárquica.
- Exemplo: Arquivos em diretórios.
- Solução: **Árvores**.

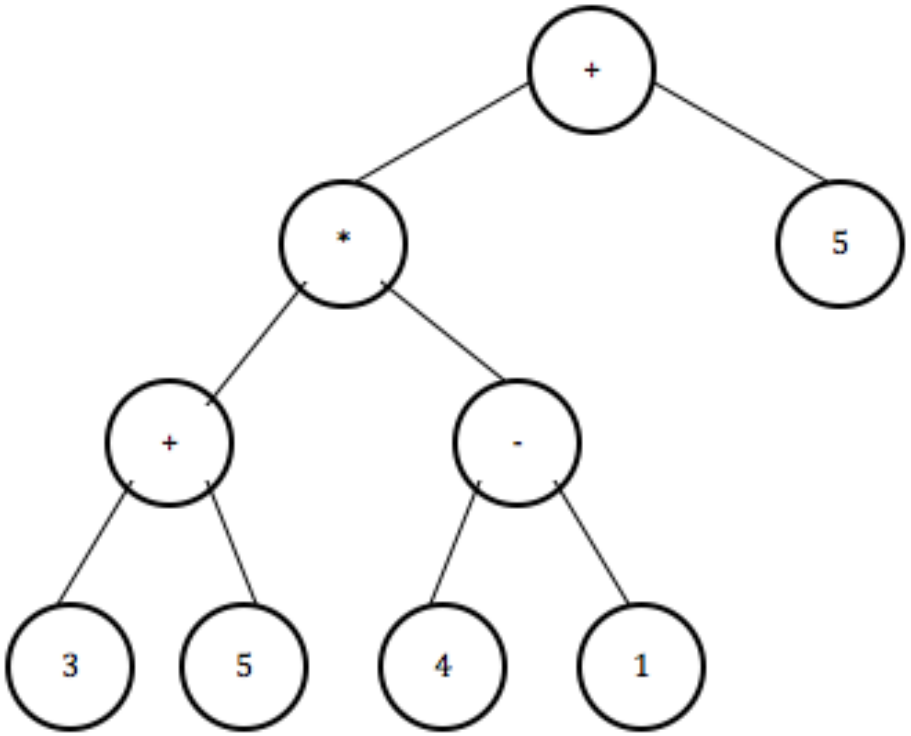
Árvores

- Árvores são as estruturas de dados mais adequadas para a representação de hierarquias.
- A forma mais natural de definir uma estrutura de árvore é usando recursividade.
  - Uma árvore é composta por um conjunto de nós.
  - Existe um nó r, denominado *raiz*, que contém zero ou mais subárvores.
  - Esses nós raízes das subárvores são ditos filhos do nó pai r.
  - Nós com filhos: Nós internos
  - Nós sem filhos: Nós externos ou folhas.

- Tipos de árvores: O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes.

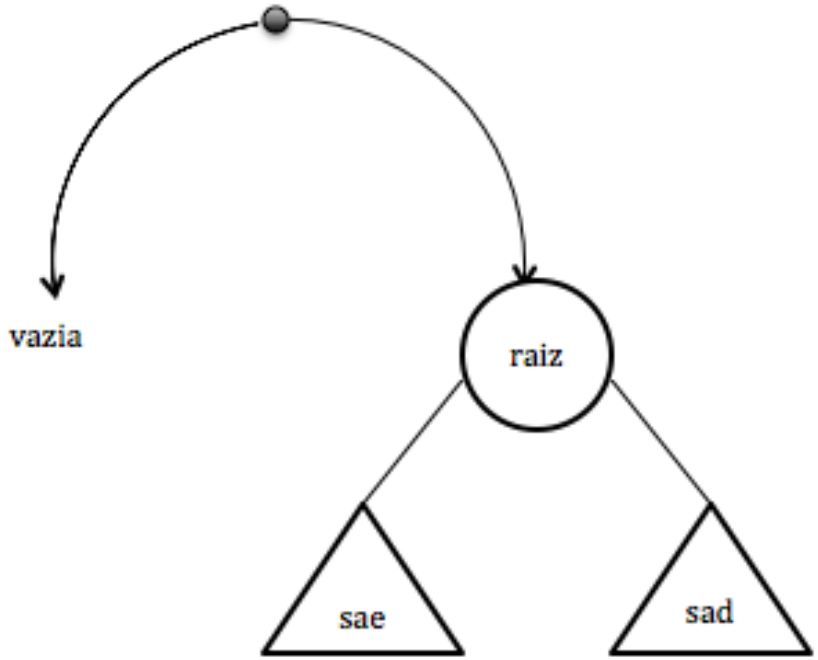
Árvores Binárias

- Árvores binárias: Cada nó tem no máximo dois filhos.
- Exemplo de utilização: avaliação de expressões.



Árvores Binárias

- De maneira recursiva, podemos definir uma árvore binária como sendo:
  - uma árvore vazia, ou;
  - um nó raiz tendo duas subárvores, identificadas como a subárvore da direita (*sad*) e a subárvore da esquerda (*sae*).



Árvores Binárias - Representação em C

- A estrutura de C para representar o nó da árvore pode ser dada por:

```
1 struct arv{
2     char info;
3     struct arv *esq;
4     struct arv *dir;
5 }
6
7 typedef struct arv Arv;
```

Árvores Binárias - Função Cria Vazia

```
1 Arv *arv_criavazia(){
2     return NULL;
3 }
```

Árvores Binárias - Função Cria

Para criar uma árvore não vazia. Essa função tem como retorno o endereço do nó raiz criado.

```
1 Arv *arv_cria(char c, Arv *sae, Arv *sad){
2     Arv *p = (Arv*) malloc (sizeof(Arv));
3     p->info = c;
4     p->esq = sae;
5     p->dir = sad;
6     return p;
7 }
```

Árvores Binárias - Função Vazia

```
1 int arv_vazia(Arv *a){
2     return a == NULL;
3 }
```

Árvores Binárias - Função Imprime

```
1 void arv_imprime(Arv *a){
2     if (!arv_vazia(a)){
3         printf("%c  ", a->info);
4         arv_imprime(a->esq);
5         arv_imprime(a->dir);
6     }
7 }
```

Árvores Binárias - Função Libera

```
1 Arv *arv_libera(Arv *a){
2     if (!arv_vazia(a)){
3         arv_libera(a->esq);
4         arv_libera(a->dir);
5         free(a);
6     }
7     return NULL;
8 }
```

Árvores Binárias - Função Pertence

```
1 int arv_pertence(Arv *a, char c){
2     if (arv_vazia(a))
3         return 0;
4     else
5         return a->info == c || arv_pertence(a->esq, c) || arv_pertence(a
              ->dir, c);
6 }
```

Ordens de percurso em árvores binárias

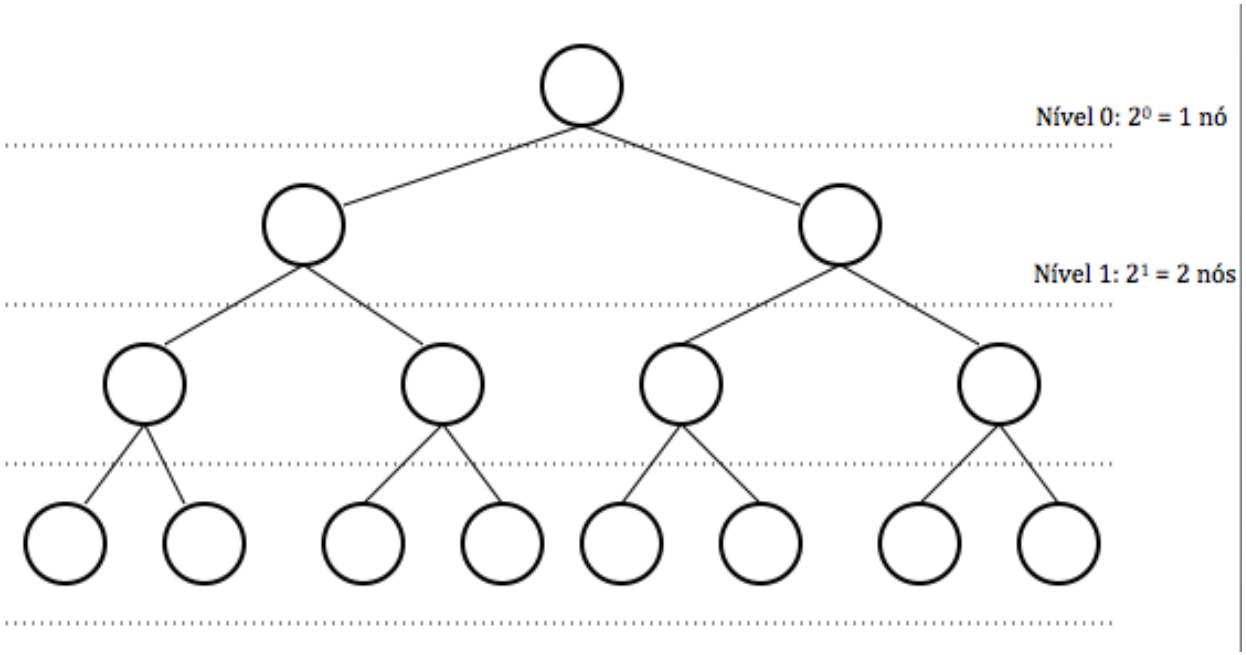
- A programação da operação imprime seguiu a ordem empregada na definição da árvore binária (raiz, subárvore da esquerda, subárvore da direita).
- Entretanto, dependendo da aplicação, essa ordem poderia não ser a preferível.
- É comum percorrer uma árvore em uma das seguintes ordens:
  - pré-ordem: raiz, sae, sad;
  - ordem simétrica (in-order): sae, raiz, sad;
  - pós-ordem: sae, sad, raiz;

Altura de uma árvore

- Uma propriedade fundamental de todas as árvores é que só existe um caminho da raiz para qualquer nó.
- Assim, a altura de uma árvore é o comprimento do caminho mais longo da raiz até uma das folhas.
  - árvore com um único nó raiz: altura zero;
  - raiz está no nível zero, seus filhos no nível 1, e assim por diante. O último nível da árvore é o nível h, sendo h a altura da árvore.

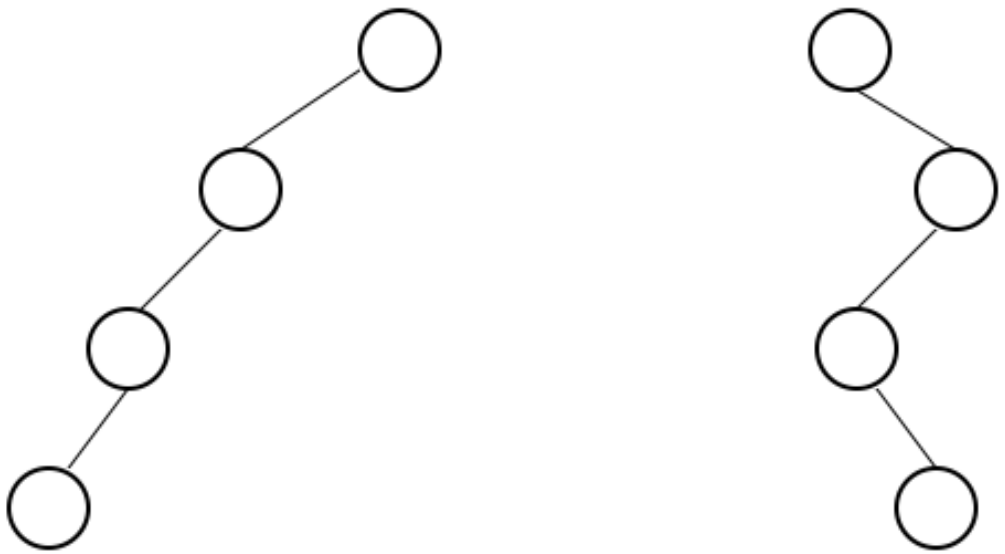
Altura de uma árvore

- Uma árvore binária é dita cheia (ou completa) se todos os seus nós internos têm duas subárvores associadas e todos os nós folha estão no último nível.
- Uma árvore cheia com altura h tem um número de nós dados por $2^{h+1} - 1$.



Altura de uma árvore

- Uma árvore é dita degenerada se todos os seus nós internos têm uma única subárvore associada.
- Uma árvore degenerada de altura h tem $h + 1$ nós.



Altura de uma árvore

- A altura de uma árvore é uma medida importante na avaliação da eficiência com que visitamos os nós de uma árvore.
- Uma árvore binária com n nós tem uma altura mínima proporcional a $logn$ (caso da árvore cheia) e uma altura máxima proporcional a n (caso da árvore degenerada).
- A altura indica o esforço computacional necessário para alcançar qualquer nó da árvore.

Altura de uma árvore - Implementação

```c
static int max2(int a, int b){
    return (a > b) ? a : b;
}

int arv_altura(Arv *a){
    if (arv_vazia(a))
        return -1; //por definição
    else
        return 1 + max2(arv_altura(a->esq), arv_altura(a->dir));
}
```

Árvores com número variável de filhos

- Vamos agora considerar as estruturas de árvores nas quais cada nó pode ter mais do que duas subárvores associadas.
- Existem diversas aplicações computacionais em que precisamos trabalhar com árvores nas quais o número de filhos é limitado.
- Na área de Computação Gráfica, por exemplo, são muito utilizadas as árvores com quatro ou oito filhos por nó: *quadtree* e *octree*.

Árvores com número variável de filhos - Representação em C

A representação do nó para a árvore com número variável de filhos (fixos).

```
1 #define N 3
2
3 struct arv3{
4     char info;
5     struct no *f[N];
6 };
```

PORÉM, existem aplicações em que não há um limite superior no número de filhos.

Árvores com número variável de filhos - Representação em C

A representação do nó para a árvore com número variável de filhos (sem limite).

```
1 struct arvvar{
2     char info;
3     struct arvvar *prim; //ponteiro para o primeiro filho
4     struct arvvar *prox; //ponteiro para irmão
5 }
6
7 typedef struct arvvar ArvVar;
```

Nessa definição, não existe árvore vazia.

Árvores com número variável de filhos - Função Cria

```
1 ArvVar *arvv_cria(char c){
2     ArvVar *a = (ArvVar*) malloc (sizeof(ArvVar));
3     a->info = c;
4     a->prim = NULL;
5     a->prox = NULL;
6     return a;
7 }
```

Árvores com número variável de filhos - Função Insere

```
1 void arvv_insere(ArvVar *a, ArvVar *sa){
2     sa->prox = a->prim;
3     a->prim = sa;
4 }
```

Árvores com número variável de filhos - Função Imprime

```
1 void arvv_imprime(ArvVar *a){
2     ArvVar *p;
3     printf("%c \n", a->info);
4     for (p = a->prim; p != NULL; p = p->prox)
5         arvv_imprime(p);
6 }
```

Note que essa impressão usa pré-ordem. Também podemos pensar na versão pós-ordem, mas a ordem simétrica não faz sentido nesse caso.

Árvores com número variável de filhos - Função Pertence

```
1  int arvv_pertence(ArvVar *a, char c){
2      ArvVar *p;
3      if (a->info == c)
4          return 1;
5      else{
6          for (p = a->prim; p != NULL; p = p->prox){
7              if (arvv_pertence(p, c))
8                  return 1;
9          }
10         return 0;
11     }
12 }
```

Árvores com número variável de filhos - Função Libera

```
1  void arvv_libera(ArvVar *a){
2      ArvVar *p = a->prim;
3      while (p != NULL){
4          ArvVar *t = p->prox;
5          arvv_libera(p);
6          p = t;
7      }
8      free(a);
9  }
```

Árvores com número variável de filhos - Função Altura

```
1  int arvv_altura(ArvVar *a){
2      int hmax = -1; //tratar caso de zero filhos
3      ArvVar *p;
4
5      for (p = a->prim; p != NULL; p = p->prox){
6          int h = arvv_altura(p);
7          if (h > hmax)
8              hmax = h;
9      }
10     return hmax + 1;
11 }
```

Exercícios

1. Implemente os três tipos de percurso em árvores binárias.
2. Implemente uma inserção em árvore binária onde elementos menores que a raiz ficam à esquerda e elementos maiores que a raiz ficam à direita.
3. Dada uma árvore binária, faça uma função para contar o número de folhas, o número de nós com um filho e o número de nós com dois filhos.
4. Considere uma árvore binária, faça uma função que monte um histograma de ocorrência dos caracteres que aparecem na árvore. Modifique a inserção da questão 2 para aceitar valores iguais.

Exercício 2

```
1  Arv *arv_insere(Arv *a, char c){
2      if (a == NULL){
3          a = (Arv*) malloc (sizeof(Arv));
4          a->info = c;
5          a->esq = a->dir = NULL;
6      }
7      else if (c < a->info)
8          a->esq = arv_insere(a->esq, c);
9      else
10         a->dir = arv_insere(a->dir, c);
11    return a;
12 }
```

Na próxima aula...

Exercícios e Prova