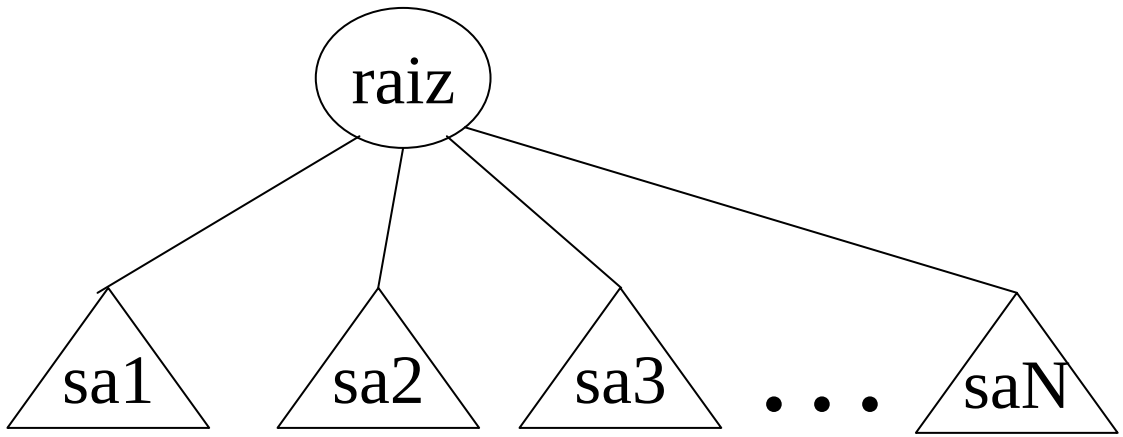
Métodos Computacionais

Árvores com Número Variável de Filhos

Árvores com Número Variável de Filhos

- Cada nó pode ter mais de dois filhos.
- Não existe sentido em falar de sub-árvore da direita ou da esquerda. Fala-se em primeira sub-árvore (sa1), segunda (sa2), terceira (sa3), etc



Representação em C

- Diferentes estratégias podem ser usadas para representar em C árvores com número variável de filhos
- Dependendo da aplicação, a estrutura em C que utilizamos para a implementação de um nó da árvore pode mudar
  - Estrutura com um ponteiro para cada sub-árvore
  - Estrutura com um vetor para todas as sub-árvores
  - Estrutura com uma lista de sub-árvores

Ponteiro para cada Sub-árvore

- A estrutura de um nó deve ser composta por: um campo que guarda a informação e um ponteiro para cada sub-árvore.
- Vantagens
  - Bom quando sabemos o número máximo de filhos
  - Bom quando número máximo de filhos pequeno
- Desvantagens
  - Limita número de filhos
  - Ruim para número grande de filhos

- Em C:

```
struct arvvar{
  char info;
  struct arvvar* sa1;
  struct arvvar* sa2;
  struct arvvar* sa3;

} ;
```

Vetor de Sub-árvores

- A estrutura de um nó deve ser composta por: um campo que guarda a informação e um vetor de sub-árvores.
- Vantagens
  - Maneira sistemática de acessar todas as sub-árvores
  - Mesmo código pode ser aplicado para árvores com outros limites de filhos
- Desvantagens
  - Limita número de filhos
  - Possível desperdício de memória

- Em C:

```
#define N 3
struct arvvar{
 char info;
 struct arvvar* sa[N];

} ;
```

Lista de Sub-árvores

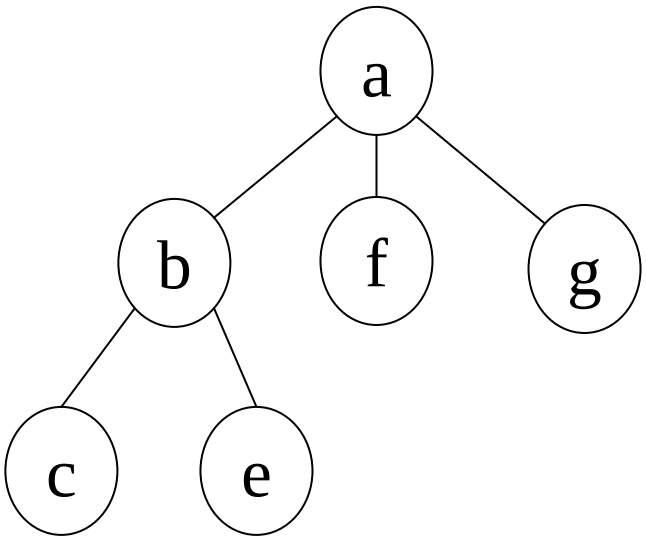
- A estrutura de um nó deve ser composta por: um campo que guarda a informação e 2 ponteiros: um para primeiro filho e outro para o próximo irmão.
- Vantagens
  - Número arbitrário de filhos
  - Acesso sistemático às sub-árvores
  - Otimização de memória
- Desvantagens
  - Complexidade de manipulação de lista

- Em C:

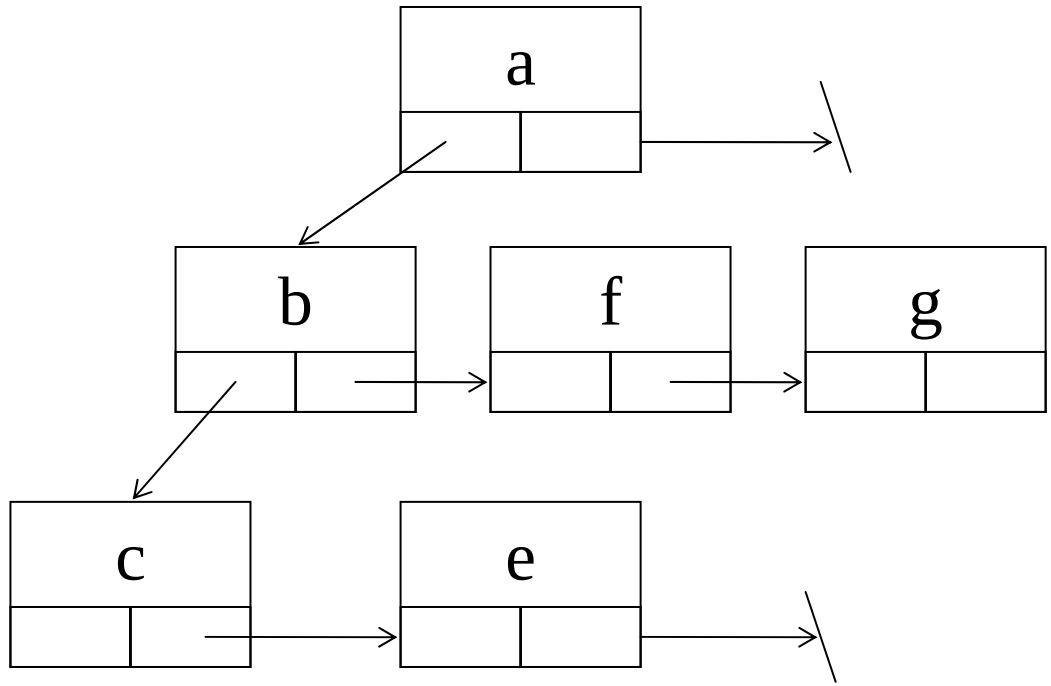
```
struct arvvar{
  char info;
  struct arvvar* prim;
  struct arvvar *prox
} ;
```

Lista de Sub-árvores

- Representação Gráfica



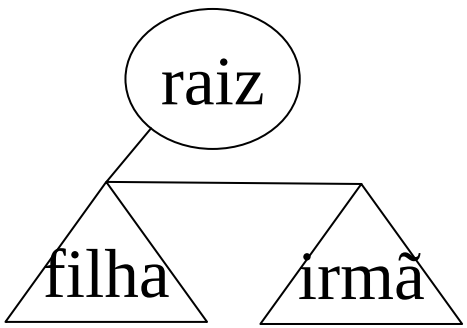
- Representação em C



Definições

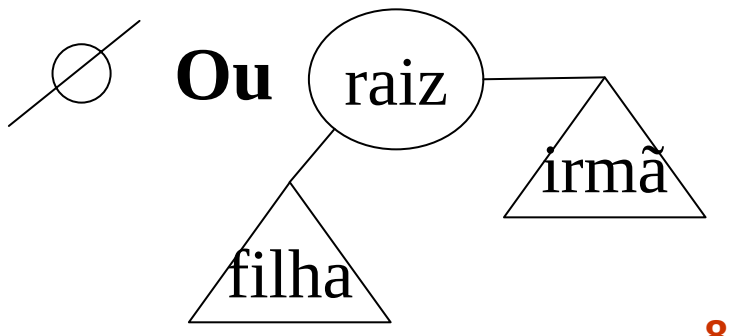
- Na implementação, podemos definir árvores com número variável de filhos de 2 formas:

- Definição 1
  - Um nó raiz; e
  - Zero ou mais sub-árvores



Ou

- Definição 2 (topologia binária)
  - Uma árvore vazia; ou
  - Um nó raiz tendo duas sub-árvores -filha e irmã



Exemplo de TAD

- Criar um nó folha
- Insere uma nova sub-árvore como filha de um dado nó
- Verificar se um elemento pertence a árvore
- Liberar uma árvore
- Imprimir os nós da árvore

- Em C:

```
/* arquivo arvore.h */

typedef struct arvvar ArvVar;

ArvVar* arvv_cria (char c) ;

void arvv_insere(ArvVar* a,ArvVar* sa);

int arvv_pertence(ArvVar* a,char c);

Arv* arvv_libera (ArvVar* a);

void arvv_imprime (ArvVar* a);
```

Implementação de Árvores com Número Variável de Filhos

- Função de Criação

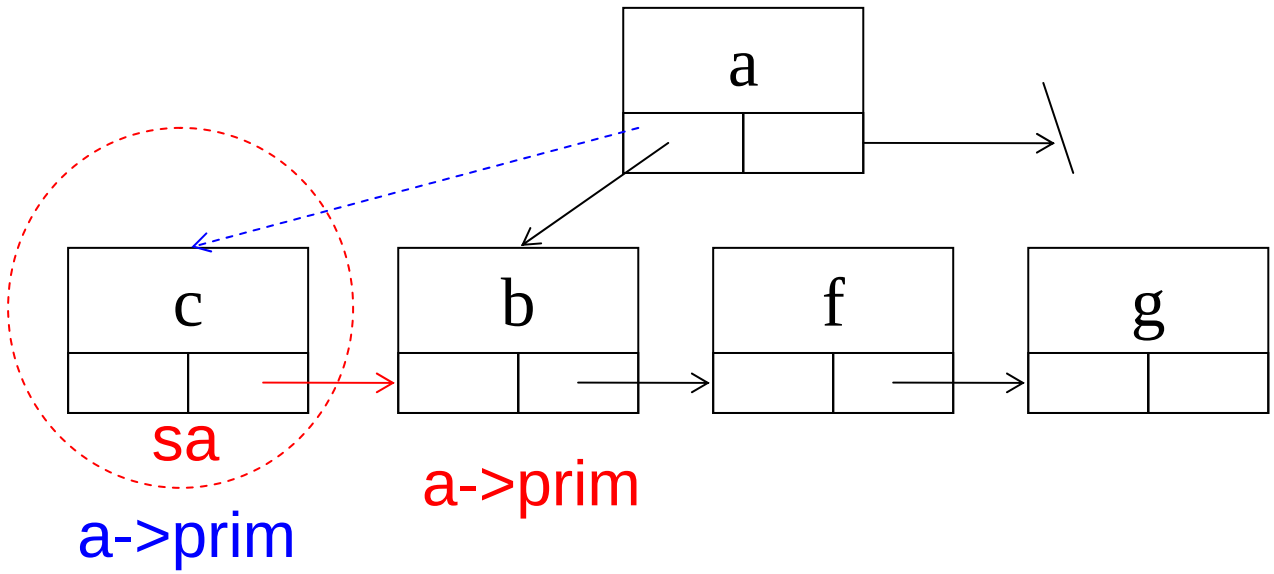
```
/*Função para criar um nó folha */
ArvVar* arvv_cria (char c) {
  ArvVar* a = (ArvVar*) malloc (sizeof(ArvVar));
  a->info = c;
  a->prim = NULL;
  a->prox = NULL;
  return a;
}
```

Sub-árvore vazia ou Zero sub-árvores

Implementação de Árvores com Número Variável de Filhos

- Função de Inserção

```
/*Função para inserir sub-árvore */
ArvVar* arvv_insere (ArvVar* a, ArvVar* sa) {
  sa->prox = a->prim;
  a->prim = sa;
}
```



Implementação de Árvores com Número Variável de Filhos

- Função Pertence (Versão 1)

```
int arvv_pertence (ArvVar* a, char c) {
  ArvVar* p;
  if (a->info == c)
   return 1;
  else {
   for (p= a->prim;p!=NULL;p=p->prox) {
       if (arvv_pertence(p,c))
          return 1;
     }
   return 0;
}
```

Esta versão considera que uma árvore NÃO pode ser vazia

Implementação de Árvores com Número Variável de Filhos

- Função Pertence (Versão 2)

```
int arvv_pertence (ArvVar* a, char c) {
  if (a==NULL)
   return 0;
  else
   return  a->info == c ||
           arvv_pertence(a->prim,c) ||
           arvv_pertence(a->prox,c);
}
```

Esta versão considera que uma árvore pode ser vazia

Implementação de Árvores com Número Variável de Filhos

- Função que libera a estrutura da árvore (Versão 1)

```
void arvv_libera (Arv* a) {
  ArvVar* p = a->prim;

  while (p != NULL) {
     ArvVar* t = p->prox;
     arvv_libera(p);
     p = t;
  }

  free (a);
}
```

Deve-se liberar recursivamente todos os elementos das sub-árvores primeiro

Implementação de Árvores com Número Variável de Filhos

- Função que libera a estrutura da árvore (Versão 2)

```
void arvv_libera (Arv* a) {

  if (a != NULL) {
     arv_libera(a->prim); /* libera filha */
     arv_libera(a->prox);  /* libera irmã*/
     free(a);
  }
}
```

Deve-se liberar recursivamente todos os elementos das sub-árvores primeiro

Implementação de Árvores com Número Variável de Filhos

- Função que imprime elementos da árvore (Versão 1)

```
/* exibe o conteúdo da árvore em pré-ordem*/
void arvv_imprime (ArvVar* a) {
  ArvVar* p;
  printf("%c ", a->info);  /* mostra raiz */
  for (p = a->prim; p != NULL; p = p->prox)
     arv_imprime(p);/* imprime cada sub-árvore filha*/
   }
}
```

Implementação recursiva: primeiro visita a raiz e depois as sub-árvores filhas

Poderia também ser implementada usando pós-ordem

Implementação de Árvores com Número Variável de Filhos

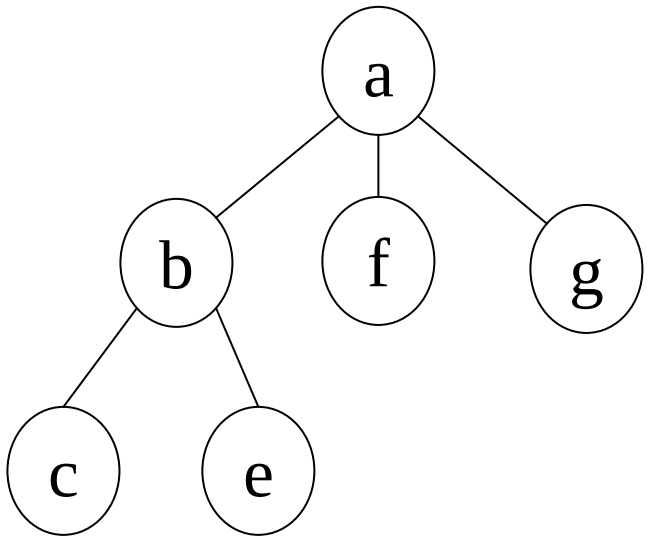
- Função que imprime elementos da árvore (Versão 2)

```
/* exibe o conteúdo da árvore */
void arvv_imprime (ArvVar* a) {
  if (a != NULL) {
     printf("%c ", a->info);  /* mostra raiz */
     arv_imprime(a->prim);  /* imprime filha */
     arv_imprime(a->prox);   /* imprime irmã*/
   }
}
```

Esta versão considera que uma árvore pode ser vazia

Criação de uma Árvore com Número Variável de Filhos

- Exemplo




```
/* Criando nós como folhas */
ArvVar* a = arvv_cria('a');
ArvVar* b = arvv_cria('b');
ArvVar* c = arvv_cria('c');
ArvVar* e = arvv_cria('e');
ArvVar* f = arvv_cria('f');
ArvVar* g = arvv_cria('g');
arvv_insere(b,e);
arvv_insere(b,c);
arvv_insere(a,g);
arvv_insere(a,f);
arvv_insere(a,b);
```

Função para Cálculo da Altura

- Função recursiva para calcular altura (Versão 1)

```
/* calcula a altura */
int  arvv_altura (ArvVar* a) {
   int hmax = -1;
   ArvVar* p;
   for (p = a->prim; p != NULL; p = p->prox) {
      int h = arvv_altura(p);
  if (h >hmax)
     hmax = h;
    }
    return hmax + 1;
}
```

Função para Cálculo da Altura

- Função auxiliar para calcular o máximo entre dois inteiros

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

- Função recursiva para calcular altura (Versão 2)

```
/* calcula a altura usando a topologia binária */
int  arvv_altura (Arv* a) {
   if (a == NULL)
      return –1;
   else
      return 1 + max2 (arvv_altura(a->prim),
                       arv_altura(a->prox));
}
```