

# Unidade X:

## Árvores TRIE



**PUC Minas**

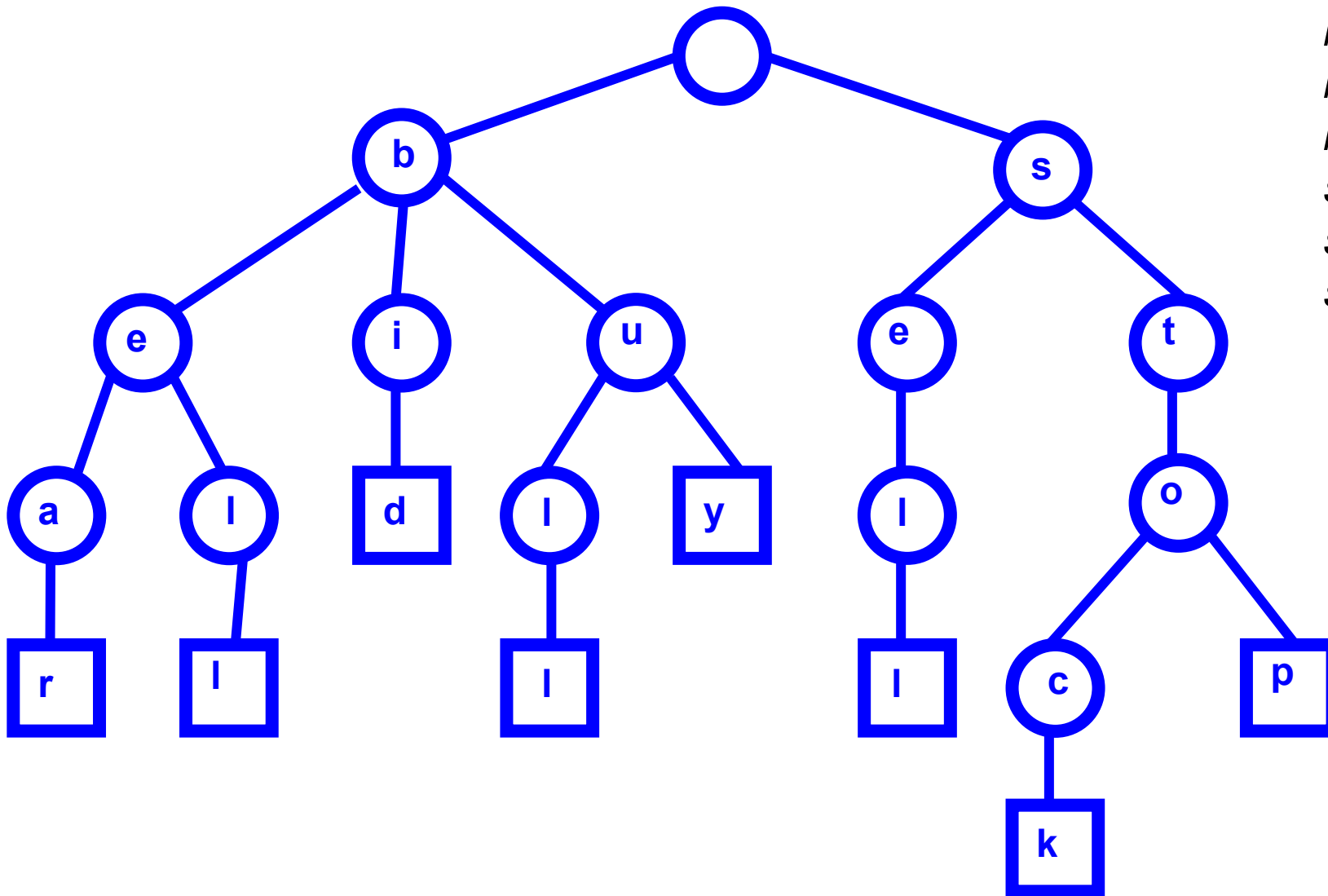
Instituto de Ciências Exatas e Informática  
Departamento de Ciência da Computação

- Estruturas de dados para a procura rápida de padrões
- Usadas em aplicações de pré-processamento do texto
- Nome derivado da palavra *retrieval* (recuperação)
- Aplicadas, por exemplo, em: índices, armazenamento de palavras (dicionários) e busca de uma sequência de DNA em uma base de genomas

# Definição

- Tem-se uma coleção de  $S$  cadeias de caracteres (ou palavras) utilizando o mesmo alfabeto e as operações primárias suportadas são:
  - Procura de padrões
  - Procura de prefixos: Recebe-se uma palavra  $X$  e retornam-se todas as palavras que têm  $X$  como prefixo

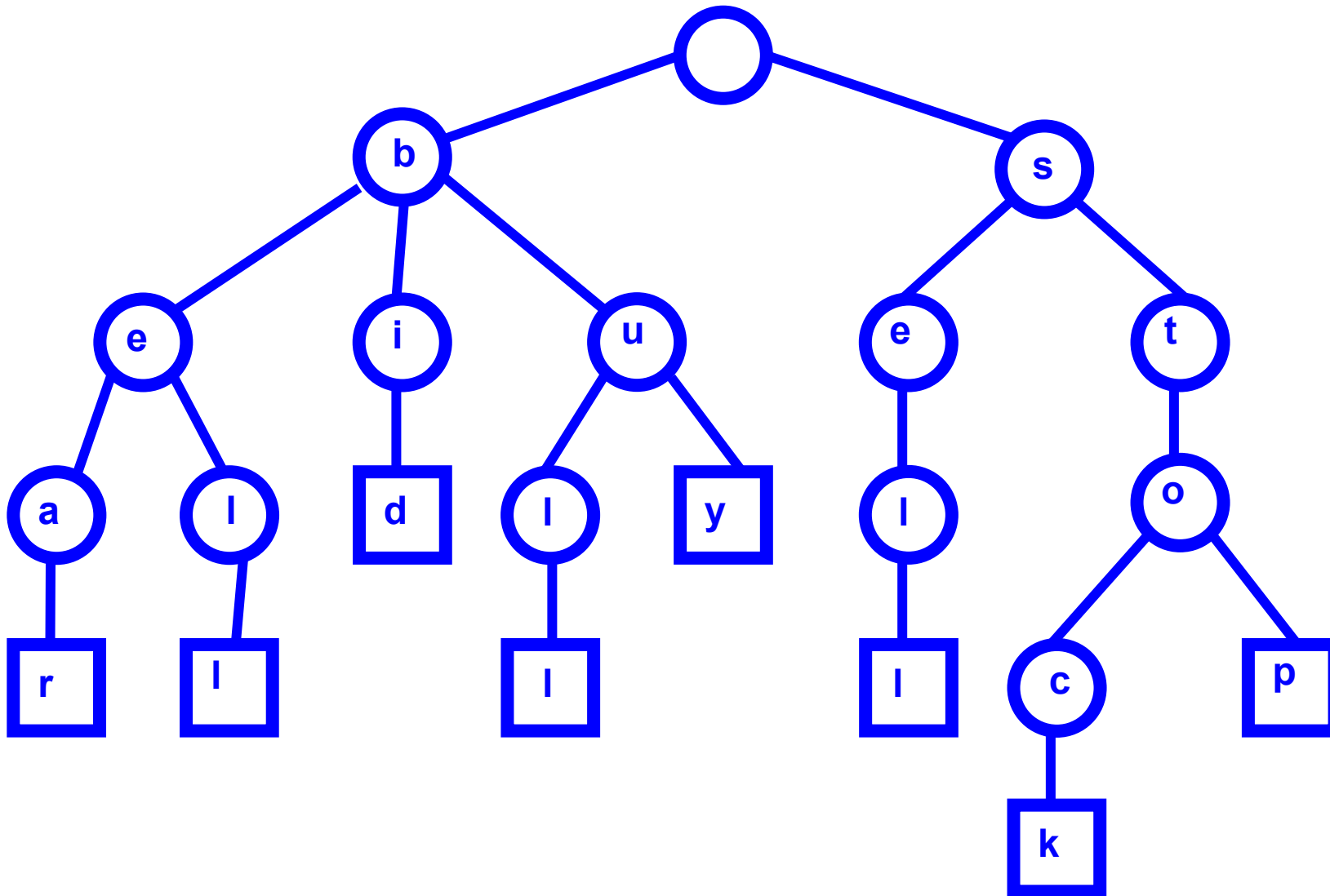
# Exemplo



*bear* - urso  
*bell* - sino  
*bid* - oferta  
*bull* - touro  
*buy* - compra  
*sell* - vende  
*stock* - ação  
*stop* - parar

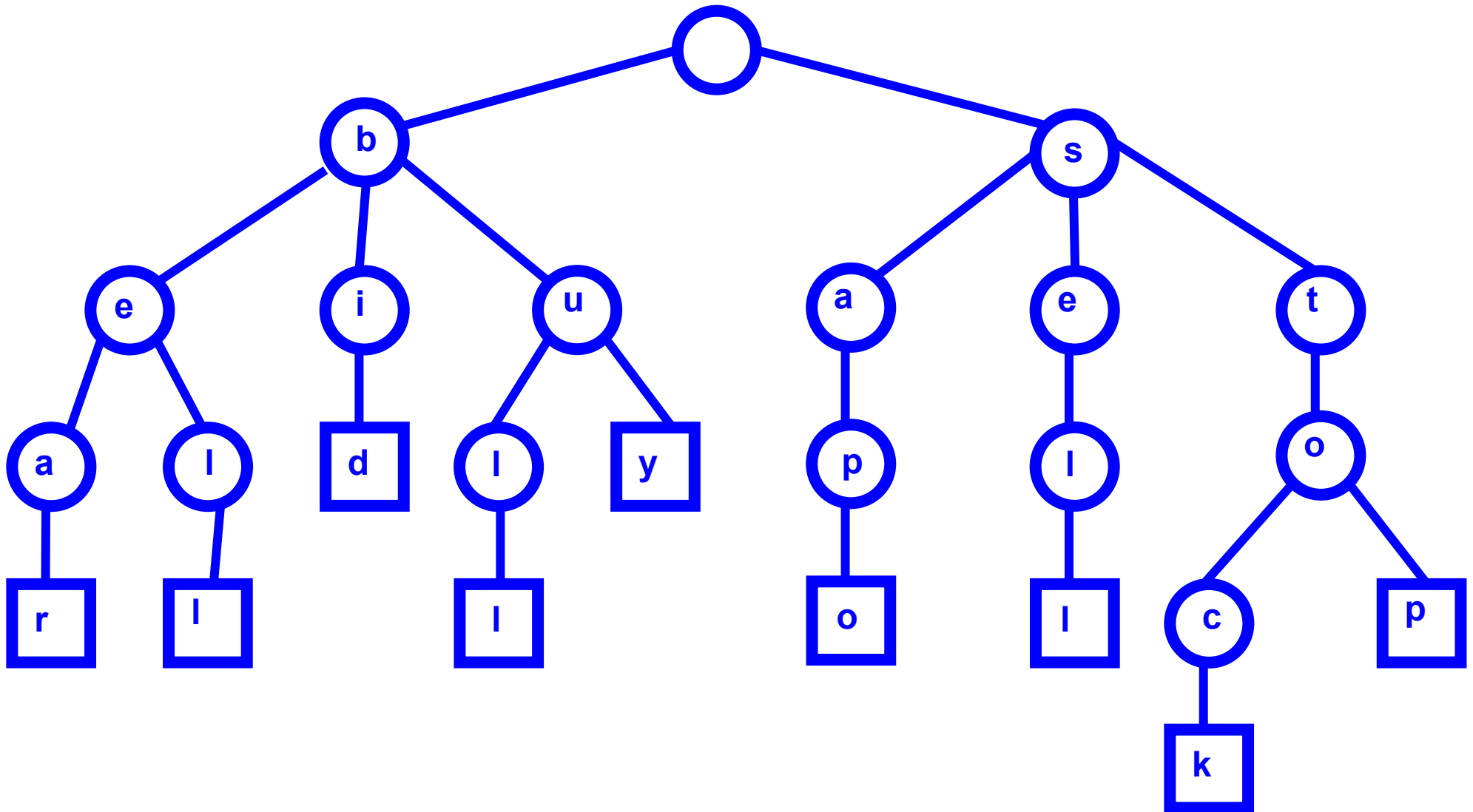
## Exercício Resolvido (1)

- Insira as palavras sapo e sapato na árvore abaixo



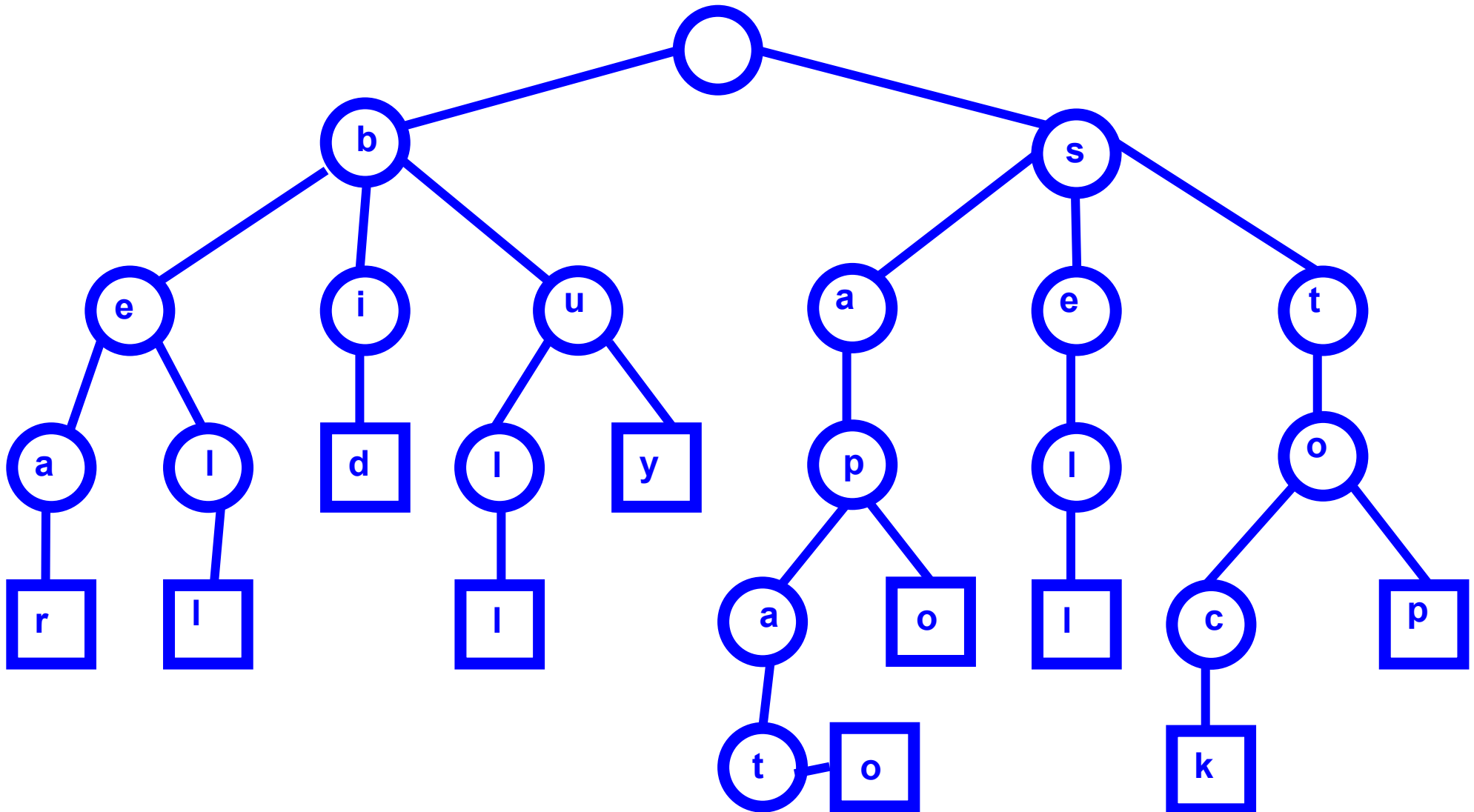
## Exercício Resolvido (1)

- Insira as palavras **sapo** e sapato na árvore abaixo



## Exercício Resolvido (1)

- Insira as palavras sapo e sapato na árvore abaixo



# Propriedades

- Nenhuma cadeia de  $S$  é prefixo de outra cadeia
- Cada nó (exceto a raiz) é rotulado com um caractere do  $\Sigma$
- A árvore tem  $s$  folhas, uma para cada cadeia de  $S$
- Em um caminho da raiz até uma folha, a concatenação dos rótulos resulta na cadeia associada a essa folha



# Propriedades

- Cada nó interno tem no máximo  $d = |\Sigma|$  filhos
- Em geral, a *trie* é uma árvore múltipla ( $0 \dots d$  filhos)
- Quando  $d$  igual a 2, a *trie* será uma árvore binária
- A altura da árvore é igual ao tamanho da maior cadeia da coleção,  $d_{\text{máximo}}$

# Propriedades

- Seja  $n$  o número total de caracteres de  $S$ , o número de nós é  $\Theta(n)$
- O pior caso para o número de nós acontece quando não existe qualquer prefixo comum entre as cadeias, fazendo com que todos os nós internos (exceto a raiz) tenham um filho

# Pesquisar por uma Cadeia de Caracteres

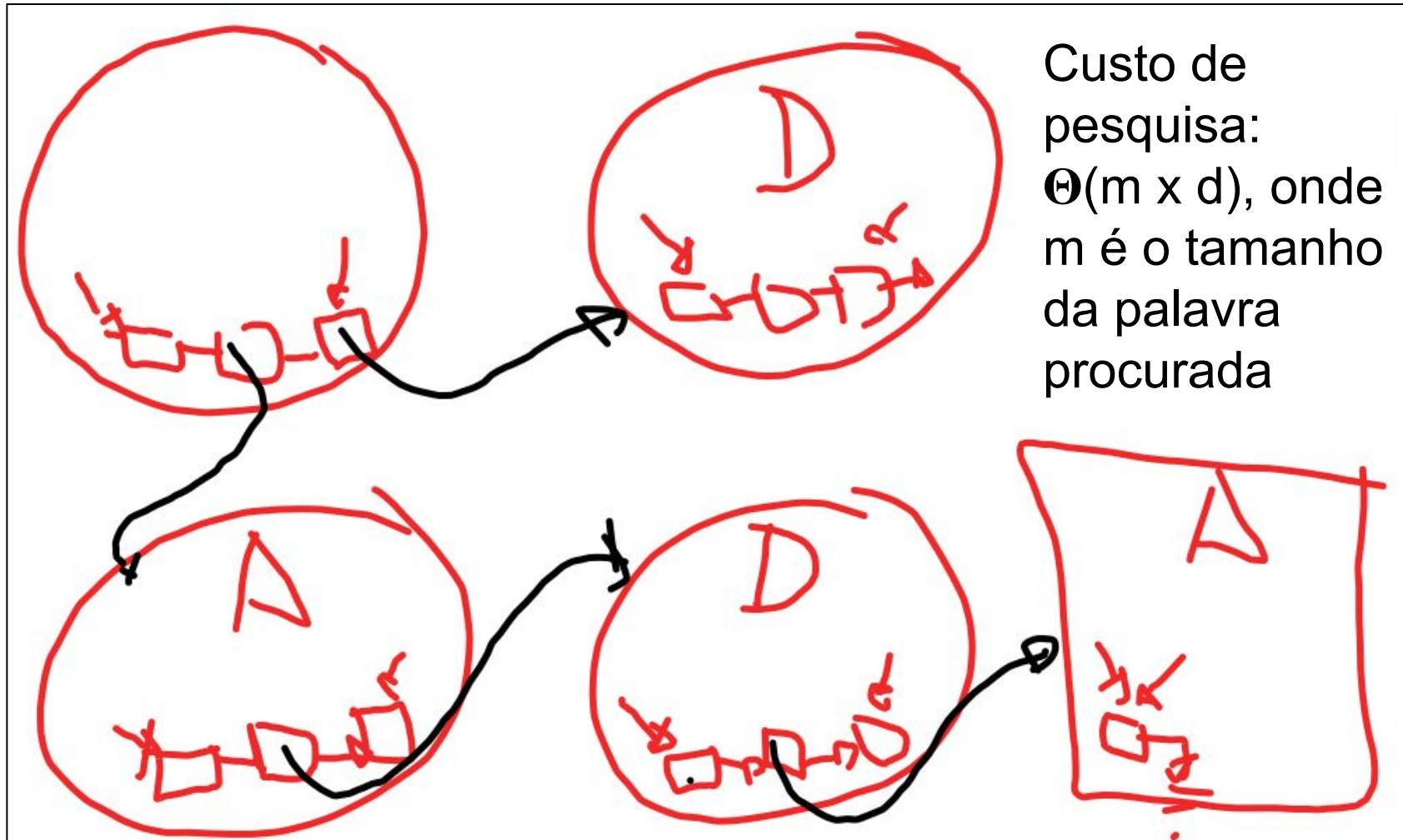
- A partir da raiz, verificamos caractere-a-caractere se existe um caminho na árvore correspondendo à cadeia desejada
- Por definição, um caminho sempre termina em uma folha

## Exercício Resolvido (2)

- Implemente a Classe Nó da Árvore Trie e mostre seu custo de pesquisa

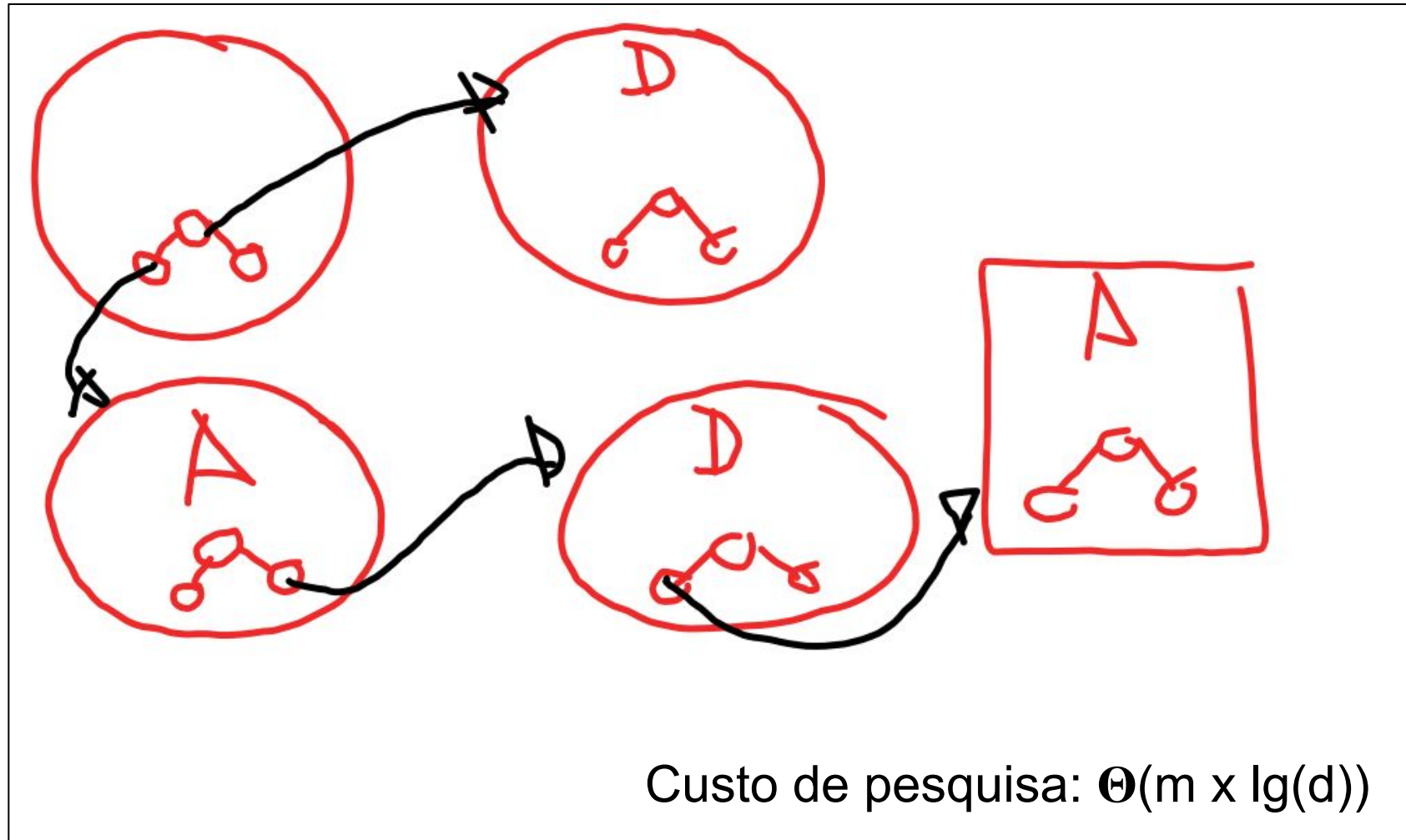
## Exercício Resolvido (2)

- Implemente a Classe Nó da Árvore Trie (**Lista flexível**) e mostre seu custo de pesquisa



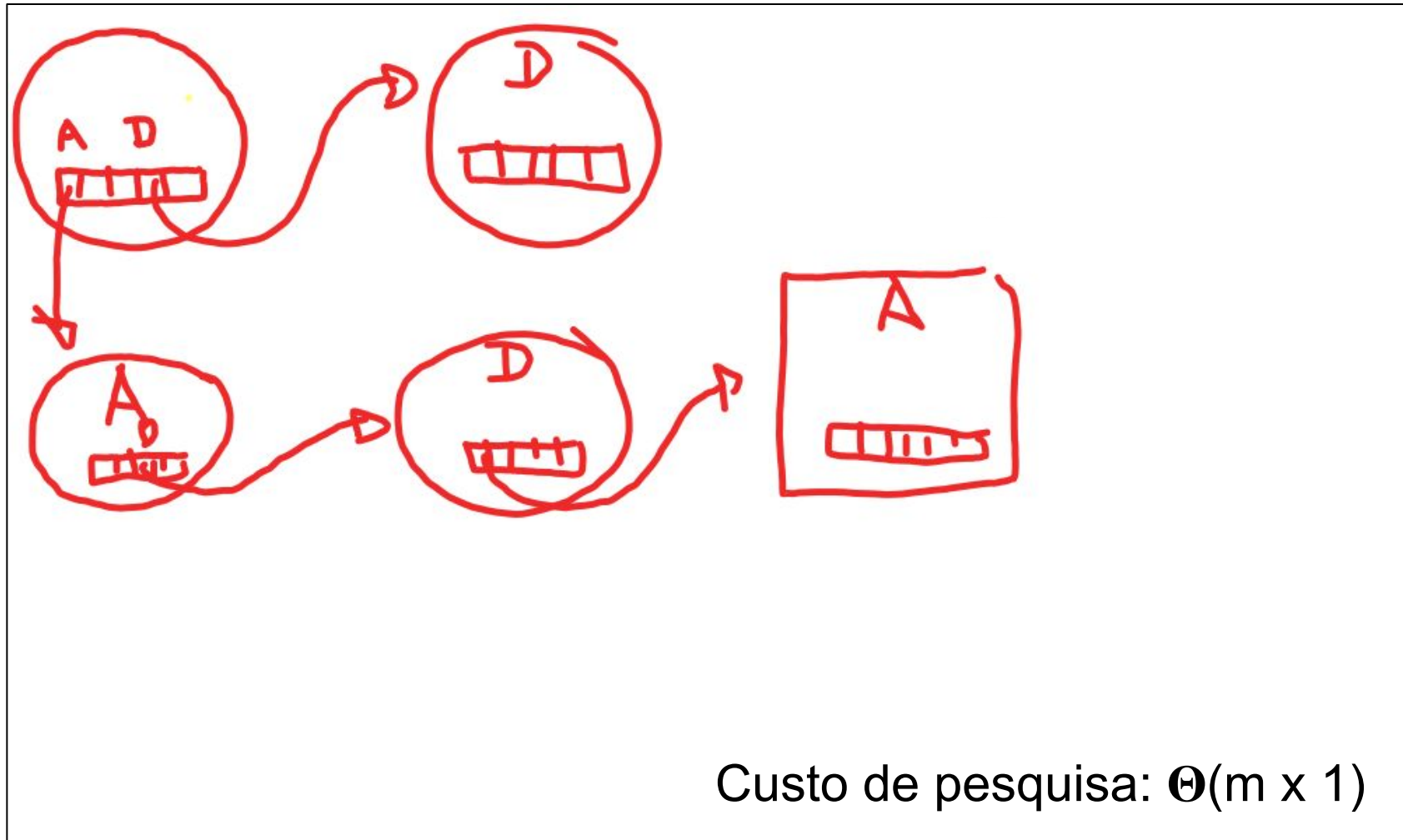
## Exercício Resolvido (2)

- Implemente a Classe Nó da Árvore Trie (**AB Balanceada**) e mostre seu custo de pesquisa



## Exercício Resolvido (2)

- Implemente a Classe Nó da Árvore Trie (**Hash Perfeita**) e mostre seu custo de pesquisa



## Exercício Resolvido (3)

- Implemente a Classe Nó da Árvore Trie usando: a) *Hash* Perfeita; b) Árvore Binária; c) Lista Flexível



## Exercício Resolvido (3)

- Implemente a Classe Nó da Árvore Trie usando: **a) Hash Perfeita**; b) Árvore Binária; c) Lista Flexível

```
class No {  
    public char elemento;  
    public final int tamanho = 255;  
    public No[] prox;  
    public boolean folha;  
  
    public No () {  
        this(' ');  
    }  
  
    public No (char elemento) {  
        this.elemento = elemento;  
        prox = new No [tamanho];  
        for (int i = 0; i < tamanho; i++) prox[i] = null;  
        folha = false;  
    }  
  
    public static int hash (char x) {  
        return (int)x;  
    }  
}
```

## Exercício Resolvido (3)

- Implemente a Classe Nó da Árvore Trie usando: a) *Hash* Perfeita; **b) Árvore Binária**; c) Lista Flexível

ver código em [u08/java/trieVariacoes/trieAB](#)

## Exercício Resolvido (3)

- Implemente a Classe Nó da Árvore Trie usando: a) *Hash* Perfeita; b) Árvore Binária; **c) Lista Flexível**

ver código em [u08/java/trieVariacoes/trieListaFlexivel](#)

# Inserção de uma Cadeia de Caracteres

- Caminhamos na *trie* casando cada caractere da nova cadeia
- Quando **não** existe um nó para um caractere, criamos o nó e repetimos esse passo para os demais caracteres da cadeia
- Lembrando que nenhuma cadeia é prefixo de outra
- Tempo de inserção é  $\Theta(m)$ , onde  $m$  é o tamanho da palavra procurada
- Espaço de construção da árvore é  $\Theta(n)$ , onde  $n = |S|$

## Exercício Resolvido (4)

- Implemente a árvore *trie* usando uma **tabela hash perfeita** em seus nós:  
construtor, pesquisar, inserir e mostrar

## Exercício Resolvido (4)

- Implemente a árvore *trie* usando uma **tabela hash perfeita** em seus nós:  
construtor, pesquisar, inserir e mostrar

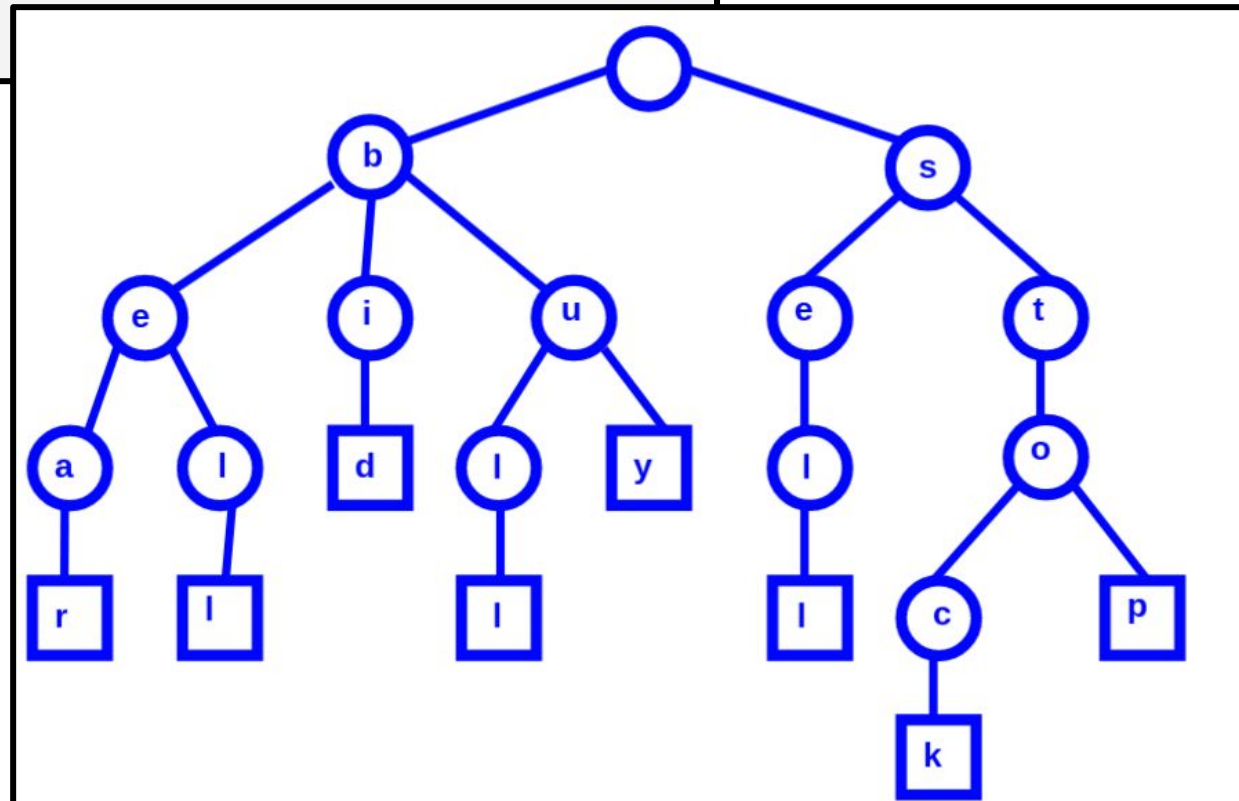
```
class ArvoreTrie {  
    private No raiz;  
    public ArvoreTrie(){  
        raiz = new No();  
    }  
    public boolean pesquisar(String s) throws Exception {  
        return pesquisar(s, raiz, 0);  
    }  
    private boolean pesquisar(String s, No no, int i) throws Exception { ... }  
    public void inserir(String s) throws Exception {  
        inserir(s, raiz, 0);  
    }  
    private void inserir(String s, No no, int i) throws Exception { ... }  
    public void mostrar(){  
        mostrar("", raiz);  
    }  
    private void mostrar(String s, No no) { ... }  
}
```

```

public boolean pesquisar(String s) throws Exception {
    return pesquisar(s, raiz, 0);
}
private boolean pesquisar(String s, No no, int i) throws Exception {
    boolean resp;
    if (no.prox[s.charAt(i)] == null){
        resp = false;
    } else if (i == s.length() - 1){
        resp = (no.prox[s.charAt(i)].folha == true);
    } else if (i < s.length() - 1){
        resp = pesquisar(s, no.prox[s.charAt(i)], i + 1);
    } else {
        throw new Exception("Erro ao pesquisar!");
    }
    return resp;
}

```

S	T	O	P
0	1	2	3



# Resolvido (4)

eita em seus nós:

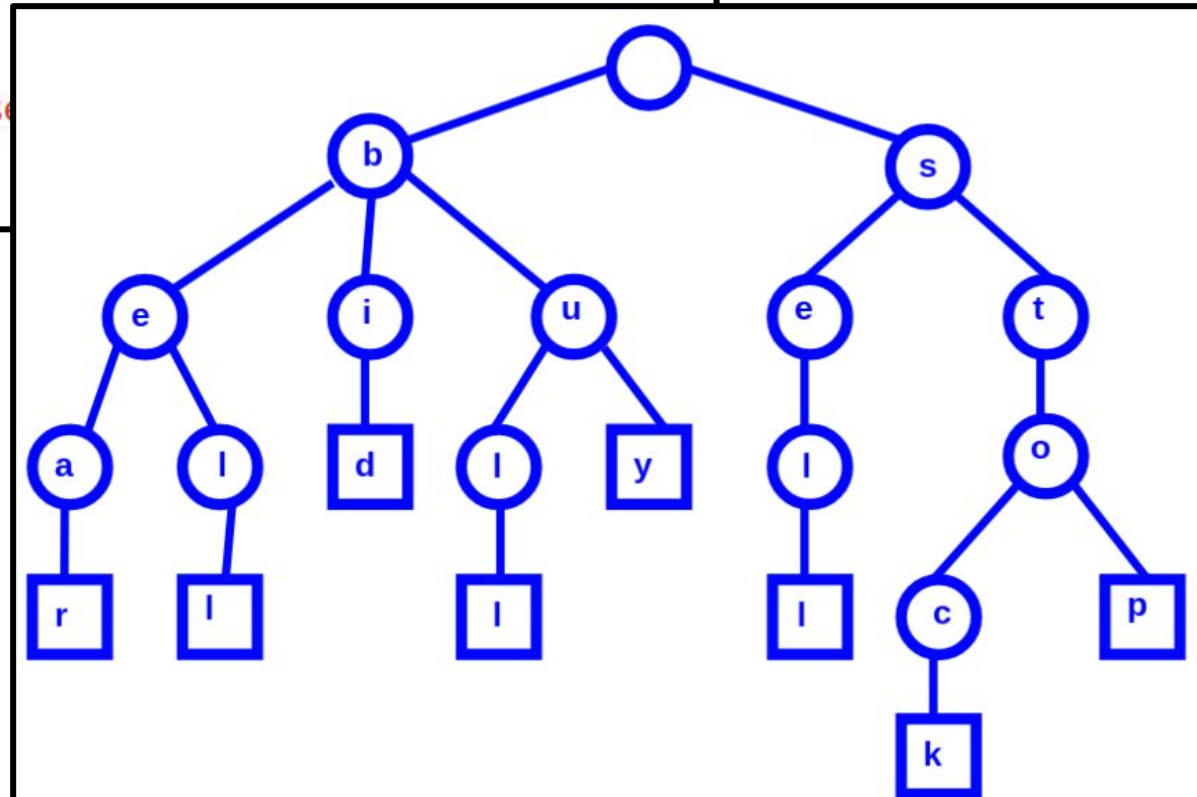
# Resolvido (4)

Feita em seus nós:

```
private void inserir(String s, No no, int i) throws Exception {
    System.out.print("\nEM NO(" + no.elemento + ") (" + i + ")");
    if(no.prox[s.charAt(i)] == null){
        System.out.print("--> criando filho(" + s.charAt(i) + ")");
        no.prox[s.charAt(i)] = new No(s.charAt(i));

        if(i == s.length() - 1){
            System.out.print("(folha)");
            no.prox[s.charAt(i)].folha = true;
        }else{
            inserir(s, no.prox[s.charAt(i)], i + 1);
        }
    } else if (no.prox[s.charAt(i)].folha == false && i < s.length() - 1){
        inserir(s, no.prox[s.charAt(i)], i + 1);
    } else {
        throw new Exception("Erro ao inserir");
    }
}
```

S	A	P	O
0	1	2	3



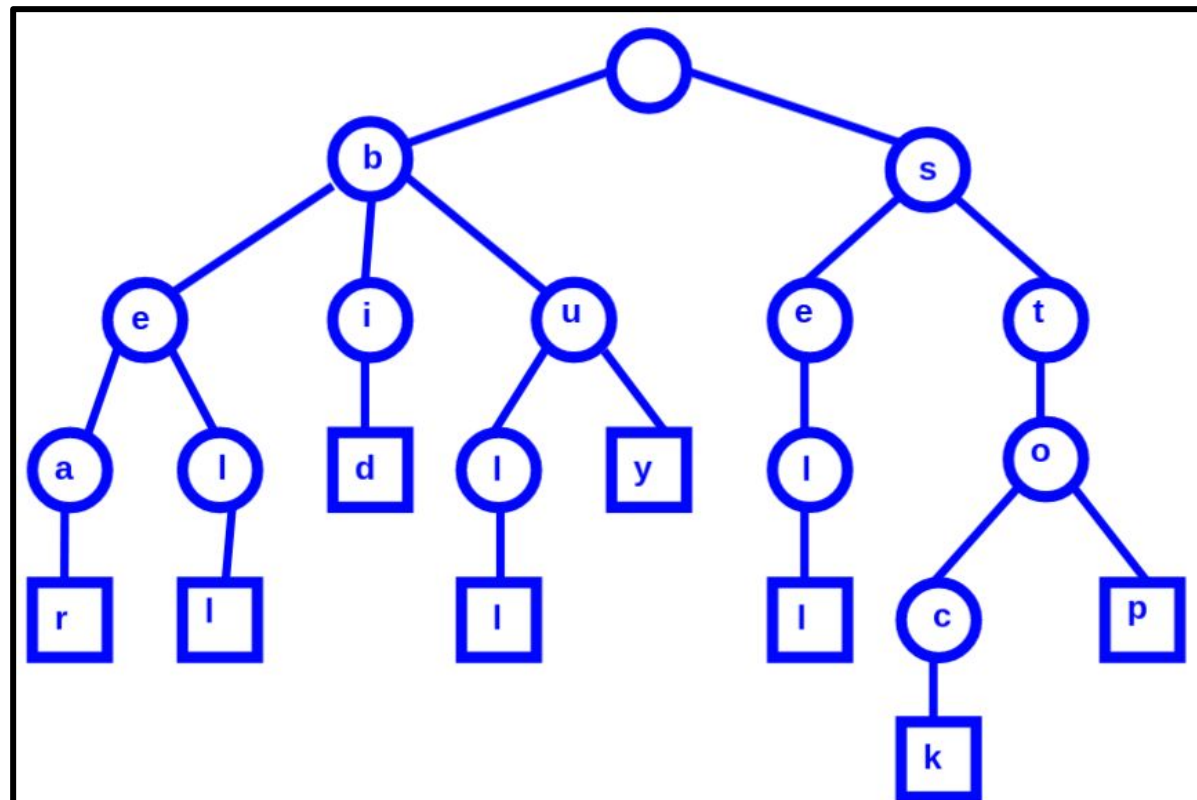


```

public void mostrar(){
    mostrar("", raiz);
}

public void mostrar(String s, No no) {
    if(no.folha == true){
        System.out.println("Palavra: " + (s + no.elemento));
    } else {
        for(int i = 0; i < no.prox.length; i++){
            if(no.prox[i] != null){
                System.out.println("ESTOU EM (" + no.elemento + ") E VOU PARA (" + no.prox[i].elemento + ")");
                mostrar(s + no.elemento, no.prox[i]);
            }
        }
    }
}
}

```



## Exercício Resolvido (5)

- Na árvore *trie* usando uma **tabela hash perfeita**, faça um método para contar o número de caracteres A

## Exercício Resolvido (5)

- Na árvore *trie* usando uma **tabela hash perfeita**, faça um método para contar o número de caracteres A

```
public int contarAs(){
    int resp = 0;
    if(raiz != null){
        resp = contarAs(raiz);
    }
    return resp;
}

public int contarAs(No no) {
    int resp = (no.elemento == 'A') ? 1 : 0;

    if(no.folha == false){
        for(int i = 0; i < no.prox.length; i++){
            if(no.prox[i] != null){
                resp += contarAs(no.prox[i]);
            }
        }
    }
    return resp;
}
```

## Exercício Resolvido (6)

- Implemente a árvore *trie* usando uma **árvore binária balanceada** em seus nós

ver código em [u08/java/trieVariacoes/trieAB](#)

## Exercício Resolvido (6)

- Implemente a árvore *trie* usando uma **árvore binária balanceada** em seus nós

ver código em [u08/java/trieVariacoes/trieAB](#)

## Exercício Resolvido (7)

- Implemente a árvore *trie* usando uma **lista flexível** em seus nós

ver código em [u08/java/trieVariacoes/trieListaFlexivel](#)

## Exercício Resolvido (7)

- Implemente a árvore *trie* usando uma **lista flexível** em seus nós

ver código em [u08/java/trieVariacoes/trieListaFlexivel](#)

## Exercício Resolvido (8)

- Implemente a árvore *trie* usando uma **tabela *hash* perfeita** em seus nós **aceitando a inserção de prefixos**



## Exercício Resolvido (8)

- Implemente a árvore *trie* usando uma **tabela *hash* perfeita** em seus nós **aceitando a inserção de prefixos**

ver código em [u08/java/trieVariacoes/trieAceitandoPrefixos](#)