# DCC207 – Algoritmos 2

Aula 03 – Estruturas de dados para casamento de padrões (árvore Trie e de sufixos)

Professor Renato Vimieiro

DCC/ICEx/UFMG

# Trie (árvores de prefixos)

- Trie é uma árvore M-ária cujos nós possuem M ligações para cada símbolo (dígito|caracter) possível do alfabeto
- O texto é quebrado em palavras, que são inseridas na Trie
- Os símbolos das palavras guiam a busca na árvore
- Todas as palavras em uma subárvore compartilham o mesmo prefixo

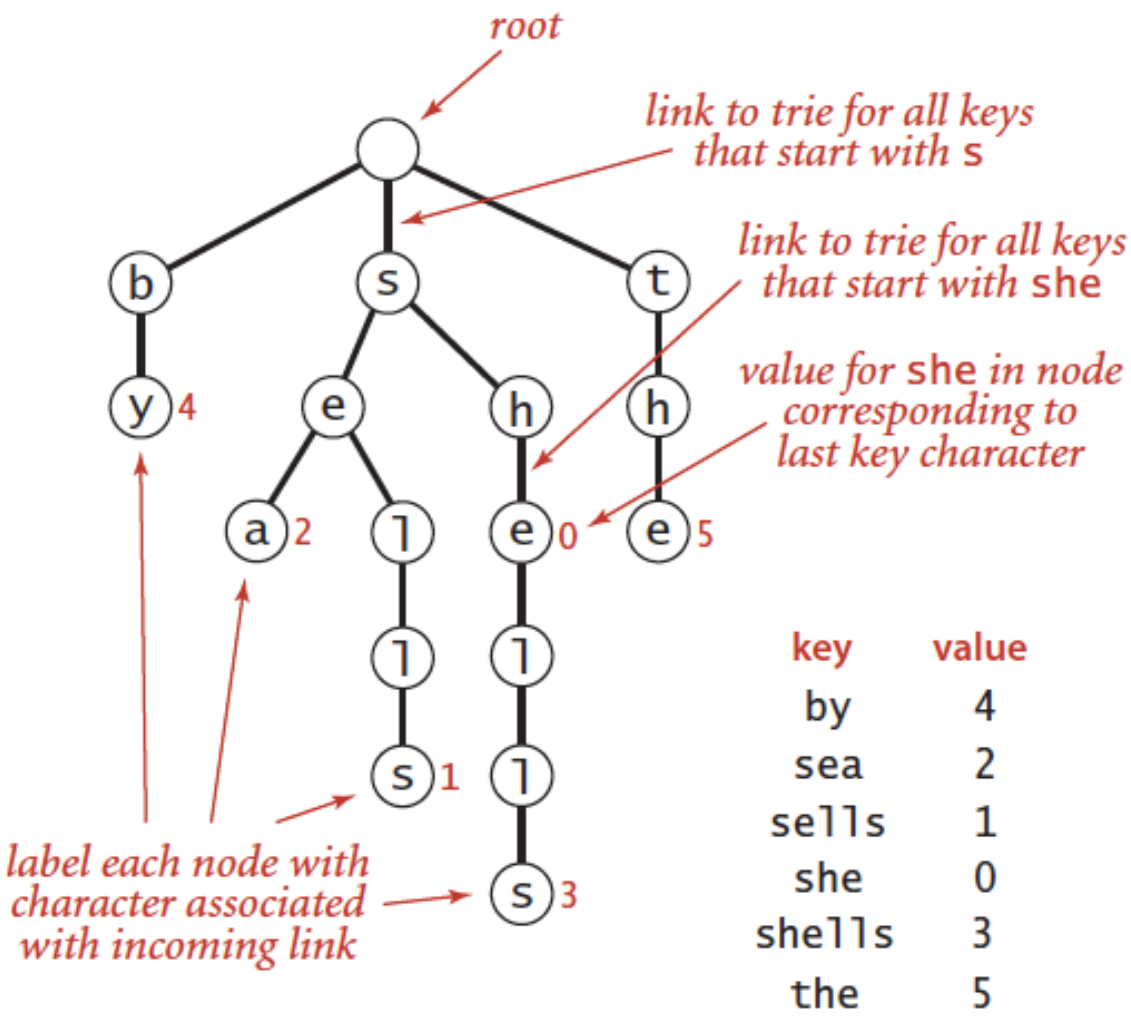
# Trie

- Úteis para:
  - Construção de índices invertidos (localização de palavras em textos/documentos)
  - Autocompletar (navegador, editor de texto …)
  - Corretores ortográficos
  - Mineração de dados
  - E também para casamento de padrões
- Tempo de busca é O(m)

# Trie (exemplo)

- A árvore ao lado é o resultado da inserção do texto a seguir:
  - Estrutura funciona como um vetor associativo
  - Palavras são chaves
  - Posição é valor

*She sells sea shells by the sea*




Anatomy of a trie

# Trie (busca)

- Iniciar busca pela raiz
- Seguir ligações usando símbolos da chave
  - No i-ésimo nível, avalia-se o i-ésimo símbolo da chave
- Possíveis situações:
  - Nó vazio é encontrado (chave não pertence à árvore)
  - Todos os símbolos da chave foram avaliados e valor vazio (não pertence à árvore)
  - Chave encontrada (símbolos avaliados e valor não vazio)

# Trie (inserção)

- Buscar local de inserção
- Se um nó vazio é encontrado durante a busca, criar nós para cada símbolo restante e inserir valor no último nó
- Se busca terminar antes de um nó vazio, inserir nesse nó (ou alterar valor)

# Trie (remoção)

- Buscar chave na árvore
- Alterar valor para nulo
- Se nó não tiver filhos, apagar nó
- Se nó pai ficar sem filhos, apagar nó

# Trie (tradeoff espaço x tempo)

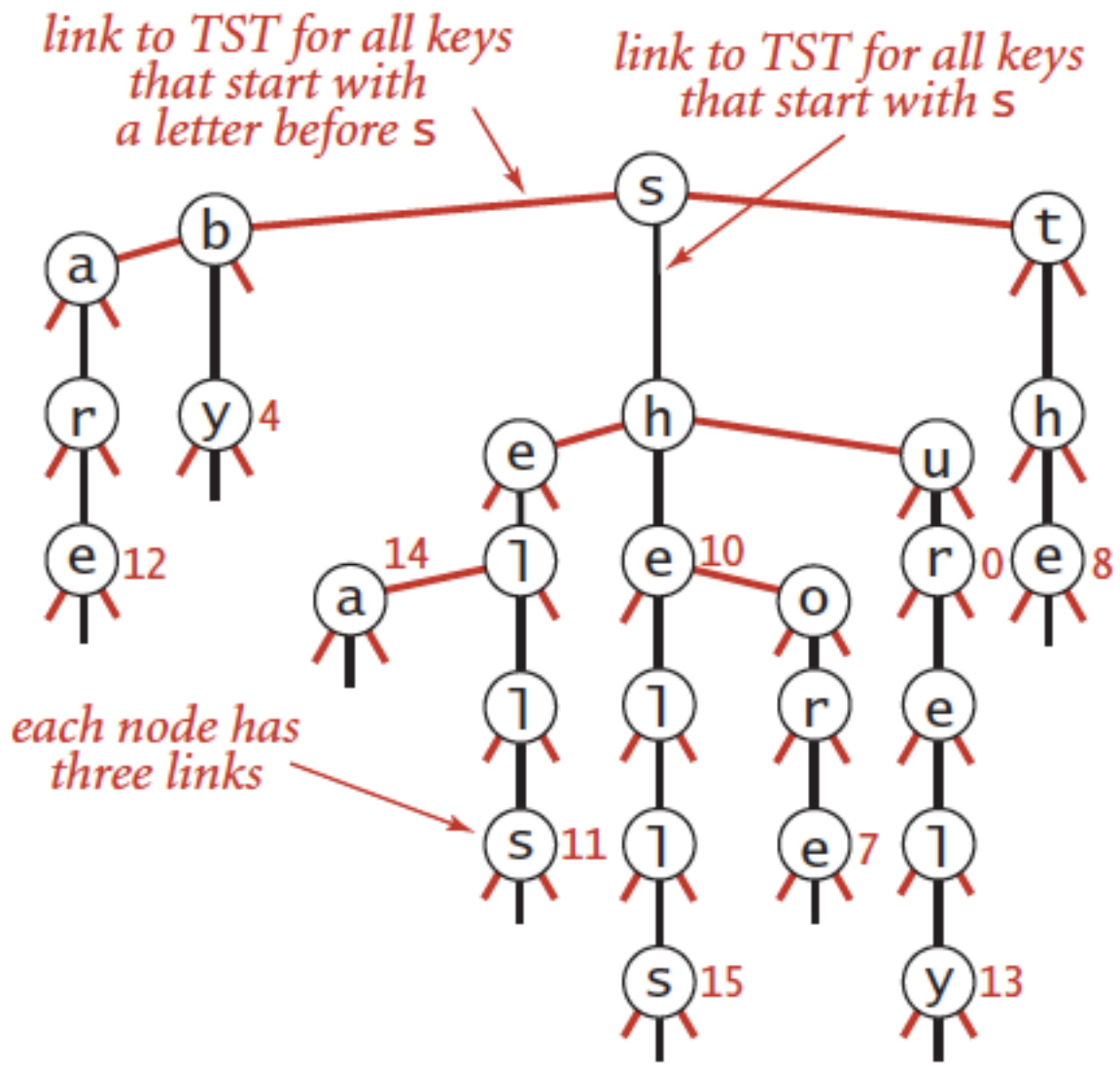
- Tries oferecem excelente desempenho para busca
- Alto custo de armazenamento (dependente do tamanho do alfabeto e número de palavras)
- Custo inviável para palavras muito longas
- Buscar alternativas mais compactas

# Trie Ternária

- Muitos nós na Trie existem somente para facilitar a busca
- Nós continuam armazenando símbolos e valores, porém passam a ter 3 ligações (elementos maiores, menores e iguais)
- Busca:
  - Compara-se o símbolo no nó ao k-ésimo símbolo da chave procurada
  - Se o símbolo for igual, incrementa-se o valor de k e busca-se o elemento na ligação ‘igual’
  - Se o símbolo for menor ou maior, mantém-se o valor de k e busca-se na ligação apropriada
  - Se nó ou valor forem nulos, chave não encontrada

# Trie Ternária

- Exemplo: buscar ‘sea’



# Trie Ternária

- Inserção:
  - Buscar a chave na árvore
  - Ao encontrar nó nulo, proceder da inserção como nas tries

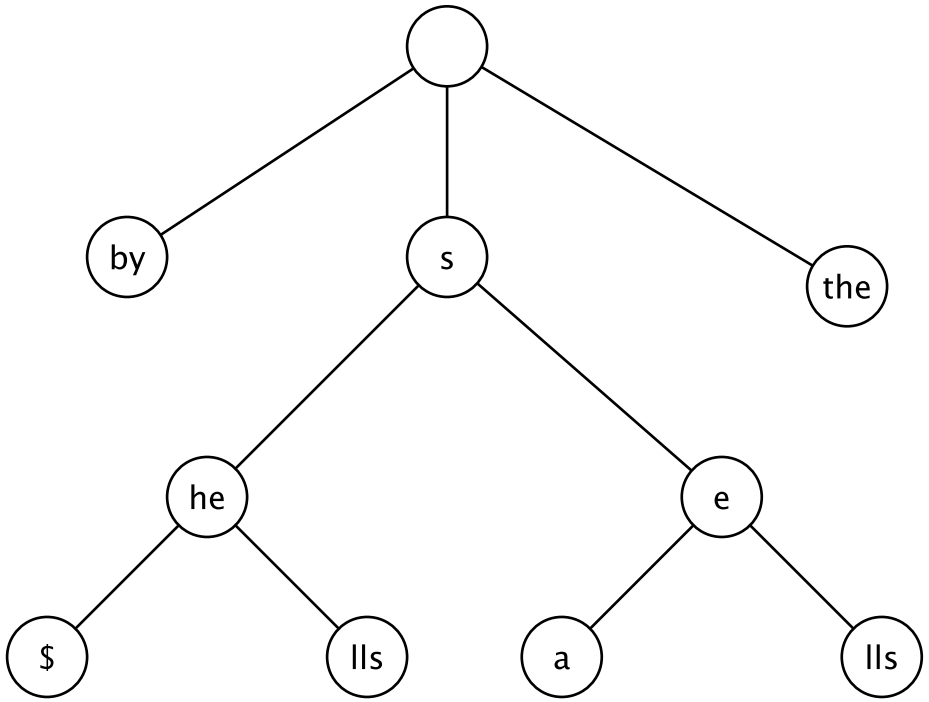
- Exemplo: construir árvore com

She sells sea shells by the sea

# Trie Compacta

- Outra alternativa para compactar Tries
- Solução: condensar nós com um único filho nos pais
- Nós guardam prefixo das chaves na subárvore
- Somente folhas armazenam valores

# Trie Compacta



# Trie Compacta (busca)

- Acumula-se o prefixo ao caminhar na árvore
- Escolha ligação baseado no próximo símbolo da chave pós-prefixo
- Se nó é folha e ainda há símbolos a processar, chave não encontrada

# Trie Compacta (inserção)

- Se árvore vazia, criar nó folha com prefixo igual à chave
- Buscar elemento na árvore
- Se, durante a busca, chave difere de prefixo
  - Criar novo nó interno (*aux*) com subprefixo comum entre chave e prefixo do nó
  - Alterar prefixo do nó para parte restante (prefixo.ante - subprefixo) e inseri-lo em *aux*
  - Criar nó folha com sufixo da chave e inseri-lo em *aux*
  - Retornar *aux*

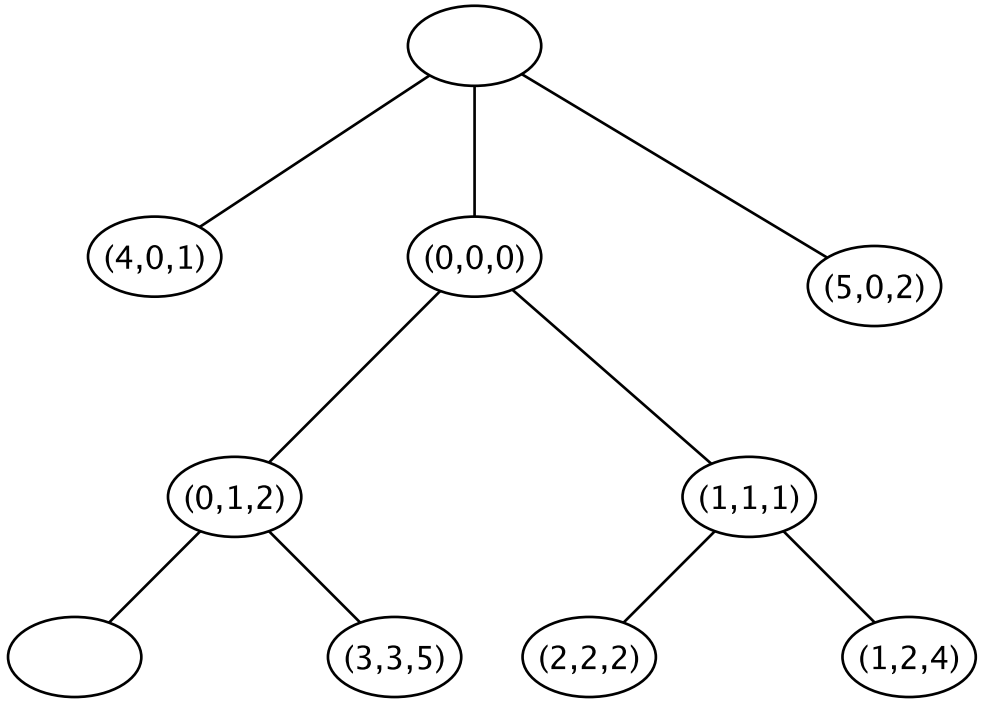
# Trie Compacta (remoção)

- Buscar nó folha contendo a chave
- Se o nó for não nulo, remover nó
- Se pai ficar com apenas 1 filho, concatenar prefixo do pai e filho, e remover filho

# Trie Compacta (representação compacta)

- A Trie Compacta é de fato mais compacta que a árvore Trie tradicional?
- Há claramente uma redução do número de nós, mas ao custo de se armazenar mais dados em cada nó
- Se as palavras estão armazenadas em um vetor, então podemos reduzir o custo de armazenamento através de índices
- Cada nó armazena uma tripla (p,e,d)
  - p é o índice da palavra cuja substring é armazenado no nó
  - e é o início da sequência
  - d é o fim

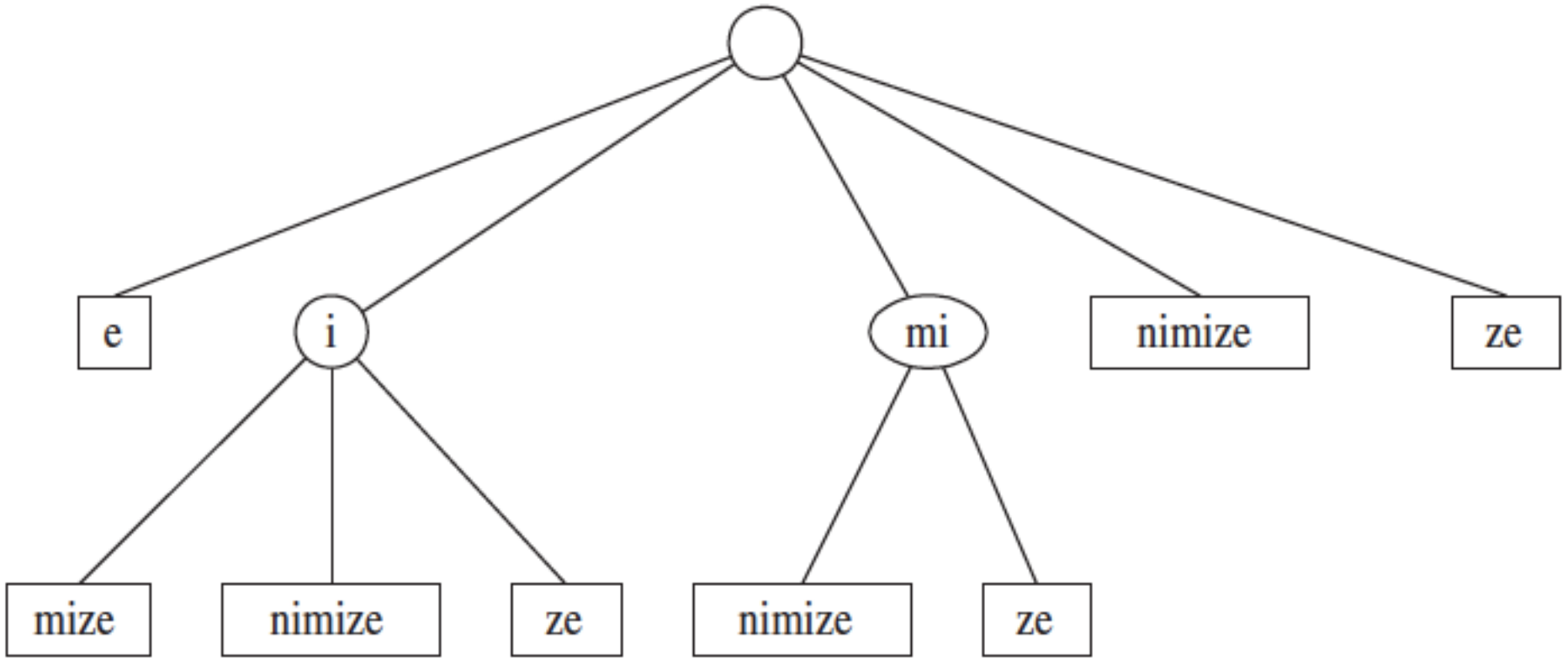
# Trie Compacta (representação compacta)



# Árvore de sufixos

- As árvores Trie compactas podem auxiliar no casamento de padrões
- Elas são úteis quando se deseja realizar diversas consultas em um mesmo texto
- Dessa forma, o pré-processamento ocorre no texto ao invés do padrão
- Como vimos, o padrão é sempre um prefixo de algum sufixo do texto
  - Se P ocorre em T, então temos que $T=\alpha P\beta$.
  - P é um sufixo de $\alpha P$
  - P é um prefixo de $P\beta$
- Portanto, podemos inserir todos os sufixos do texto em uma Trie compacta e processar as consultas em seguida

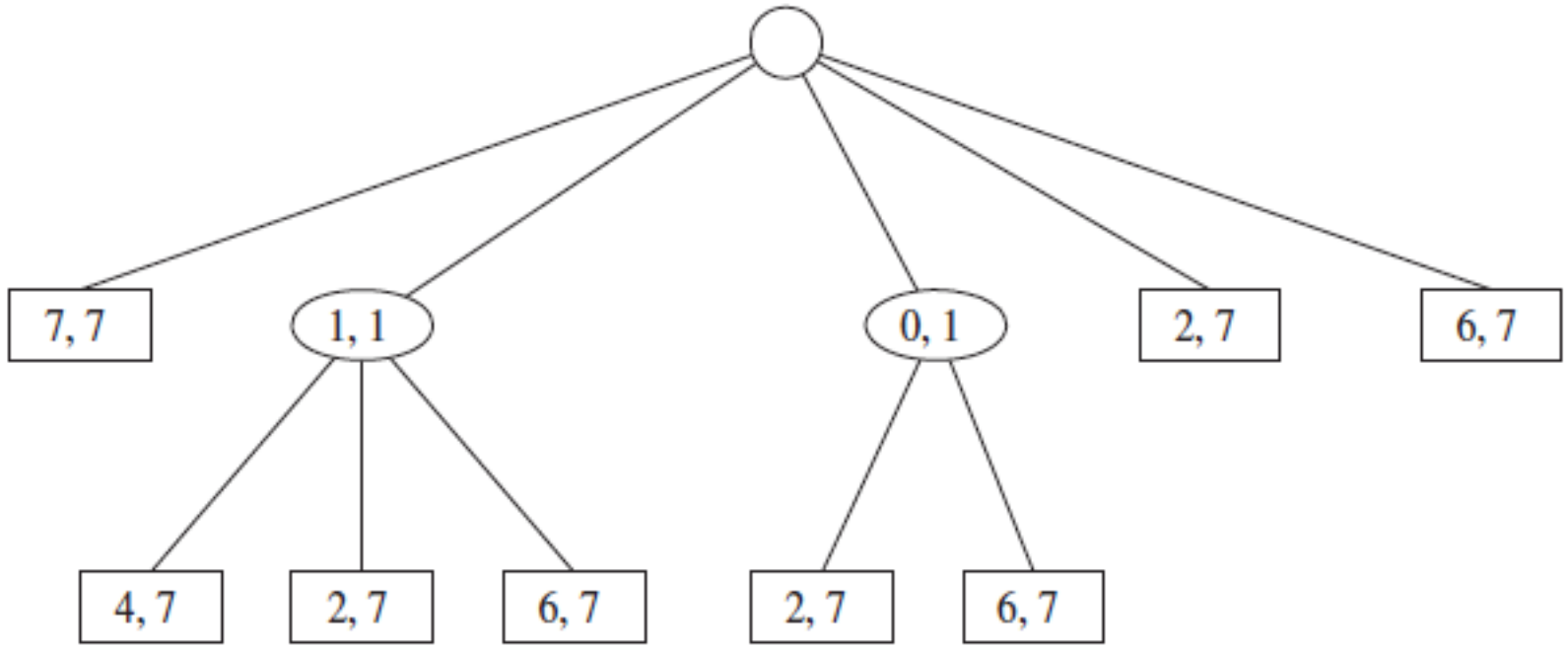
# Árvores de sufixos



# Árvores de sufixos

- A economia de espaço de uma Trie compacta fica mais evidente nas árvores de sufixo
- Devem ser inseridos *n* sufixos de um texto *T* de tamanho *n*
  - Espaço na Trie tradicional: $O(n^2)$ <- um ramo para cada sufixo
  - Espaço na Trie compacta: O(n) <- um nó para cada sufixo
    - Cada nó interno tem no mínimo 2 filhos e no máximo |Σ| (pior caso, árvore binária)
- Árvore de sufixo se torna mais útil quando usada como índice de uma outra estrutura
  - Texto armazenado num vetor de caracteres
  - Nós armazenam índices desse vetor (rótulo é um par com início e fim da substring referente ao nó)

# Árvore de sufixos




| m | i | n | i | m | i | z | e |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Árvore de sufixos

- Devemos ressaltar um fato importante em relação a substrings
- Se *x* é uma substring de *w*, então *x* é um prefixo de algum sufixo de *w*
- Note que se *x* é uma substring de *w*, então *w* = *α*x*β*, para $\alpha, \beta \in \Sigma^*$
- Portanto $x \sqsubset x\beta$ e $x\beta \sqsupset w$

# Casamento de padrões na árvore de sufixos

- Iniciar busca pela raiz
- Verificar filho cujo primeiro símbolo é idêntico ao símbolo corrente do padrão
- Se o tamanho do (restante do) padrão for menor que o rótulo do nó
  - Comparar o padrão com o rótulo e retornar resultado
    - Se encontrar, posição é o início do nó menos posição corrente de P
- Caso contrário, verificar casamento parcial do padrão com rótulo do nó e, em caso positivo, atualizar caractere corrente do padrão e repetir a busca

# Casamento de padrões na árvore de sufixos

**Algorithm** suffixTrieMatch($T, P$):
*Input:* Compact suffix trie $T$ for a text $X$ and pattern $P$
*Output:* Starting index of a substring of $X$ matching $P$ or an indication that $P$ is not a substring of $X$

```
p ← P.length() // length of suffix of the pattern to be matched
j ← 0 // start of suffix of the pattern to be matched
v ← T.root()
repeat
    f ← true // flag indicating that no child was successfully processed
    for each child w of v do
        i ← start(w)
        if P[j] = T[i] then
            // process child w
            x ← end(w) − i + 1
            if p ≤ x then
                // suffix is shorter than or of the same length of the node label
                if P[j..j + p − 1] = X[i..i + p − 1] then
                    return i − j // match
                else
                    return "P is not a substring of X"
            else
                // suffix is longer than the node label
                if P[j..j + x − 1] = X[i..i + x − 1] then
                    p ← p − x // update suffix length
                    j ← j + x // update suffix start index
                    v ← w
                    f ← false
                    break out of the for loop
until f or T.isExternal(v)
return "P is not a substring of X"
```

# Casamento de padrões na árvore de sufixos

- Exemplo:
  - P=inimi
  - T=minimize

# Leitura

- Seção 5.2 (Sedgewick e Wayne)
- Seção 5.4 (Ziviani)
- Seção 23.5 (Goodrich e Tamassia)
- http://cglab.ca/~morin/teaching/5408/notes/strings.pdf

# DCC207 – Algoritmos 2

Aula 03 – Estruturas de dados para casamento de padrões (árvore Trie e de sufixos)

Professor Renato Vimieiro

DCC/ICEx/UFMG