Busca Digital

Busca Digital

Considere um alfabeto formado por dígitos d_1, …, d_m que admitem uma ordenação $d_1 < d_2 < … < d_m$

e

***S* um conjunto de chaves s_1, …, s_n onde cada s_i é formada por uma sequência de dígitos d_j.**

***Exemplo*:**
Alfabeto = {1, 2, 3} tal que 1 < 2 < 3

Chaves: *S* = {13, 132, 22, 123, 211, 113, 32, 31}

Então m = 3 e n = 8

Busca Digital

Exemplo:

Suponha que se deseja armazenar um texto, no qual, frases serão localizadas em operações de busca.

O conjunto *S* das chaves é formado pelas frases, cada s_i sendo uma frase que pode ser buscada.

As chaves podem assumir tamanhos bem distintos. De um modo geral, não é possível armazenar cada chave em uma palavra do computador.

Árvore Digital

A chave, na busca digital, não é tratada como um elemento indivisível.

Assume-se que as chaves são formadas por um conjunto de caracteres definidos em um alfabeto apropriado.

Em vez de se comparar uma chave buscada com uma chave armazenada, a comparação é efetuada dígito a dígito.

Árvore Digital

Definição: Prefixo

Os p primeiros dígitos da chave formam o prefixo de tamanho p.

Definição: Árvore Digital (Trie R-Way)

Uma árvore digital para *S* é uma árvore *m*-ária *T*, não vazia, tal que:

i) Seja *v* um nó de *T*, o *j*-ésimo filho de seu pai, então *v* corresponde ao dígito d_j do alfabeto de *S*, $1 \leq j \leq m$.

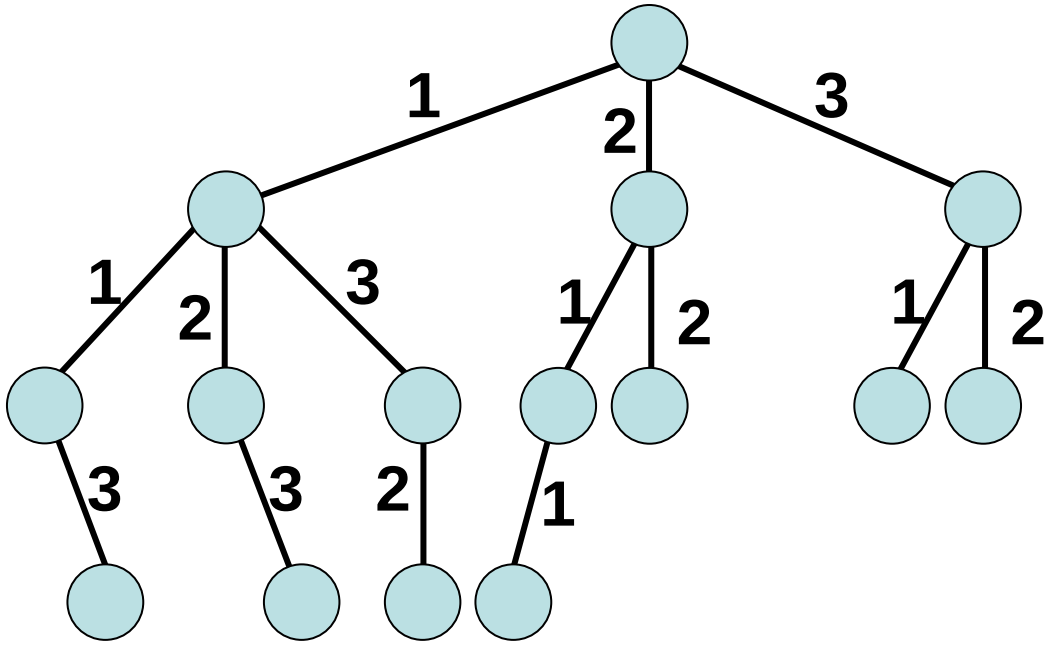
ii) Para cada nó *v*, a sequência de dígitos definida pelo caminho desde a raiz de *T* a *v* corresponde a um prefixo de alguma chave de *S*.

Árvore Digital

OBS: A raiz da árvore digital sempre existe e não corresponde a qualquer dígito do alfabeto considerado.

***Exemplo*:**

S = {13, 132, 22, 123, 211, 113, 32, 31}



Árvore Digital

A condição (i) da definição implica que o j-ésimo filho de um nó qualquer da árvore, se existir, corresponde ao dígito d_j.

OBS 1: A informação dos nós pode ser omitida dado o formato da árvore.

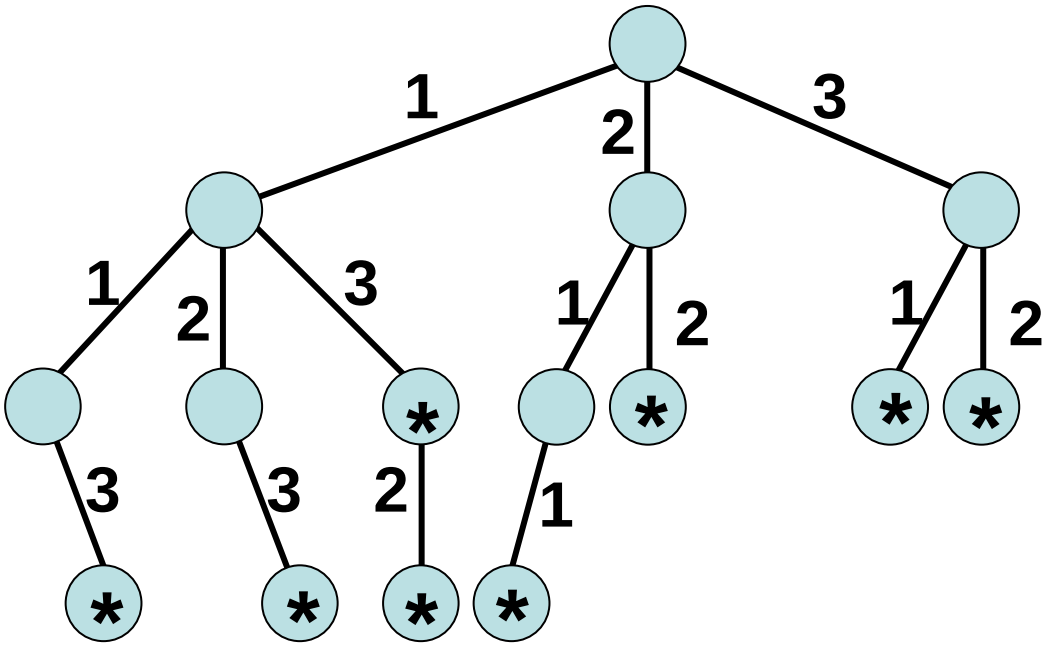
OBS 2: A árvore satisfaz a (*i*) e (*ii*).

OBS 3: Em toda árvore digital existe um nó diferente que corresponde a cada chave do conjunto. Estes nós poderiam conter ponteiros para a frase armazenada.

Árvore Digital

Exemplo:

Conjunto de chaves:
S = {13, 132, 22, 123, 211, 113, 32, 31}



Busca em Árvore Digital

1. Chave válida – Ex: 1 2 3

2. Chave não válida – Ex: 1 2 *

*** A informação no nó para diferenciar entre chaves válidas e não válidas é *t* (terminal) e *nt* (não terminal)**

3. Chave não existente – Ex: 2 1 3

Nó:

Busca em Árvore Digital

A chave *x* procurada possui *k* dígitos: d[1] … d[*k*]

***pt* aponta o nó corrente**

***l* – tamanho do maior prefixo de *x* que corresponde ao prefixo de alguma chave**

***a* = 1 – chave encontrada**

***a* = 0 – chave não encontrada**

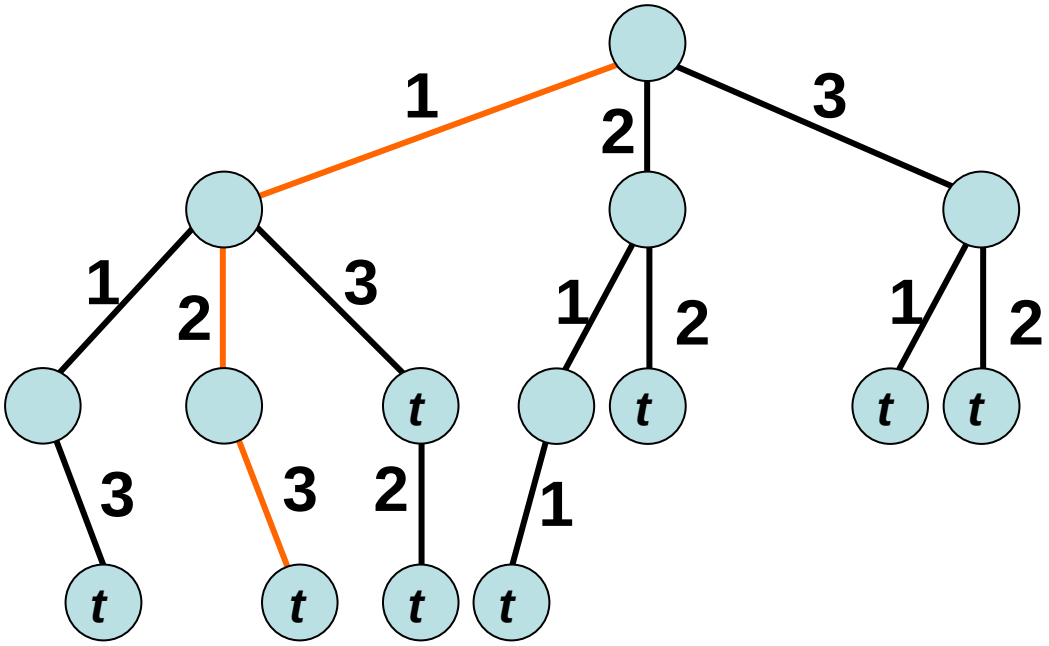
Busca em Árvore Digital

Chamada externa: buscadig(*x*, *ptraiz*, 0, 0)

```
Procedimento buscadig(x, pt, l, a)
  se (l < k) então
      seja j a posição de d(l+1) na ordenação do alfabeto
      se (pt↑.pont[j] ≠ λ) então
          pt ← pt↑.pont[j]
          l ← l + 1
          buscadig(x,pt,l,a)
  senão
      se (pt↑.info = t) então a ← 1
```

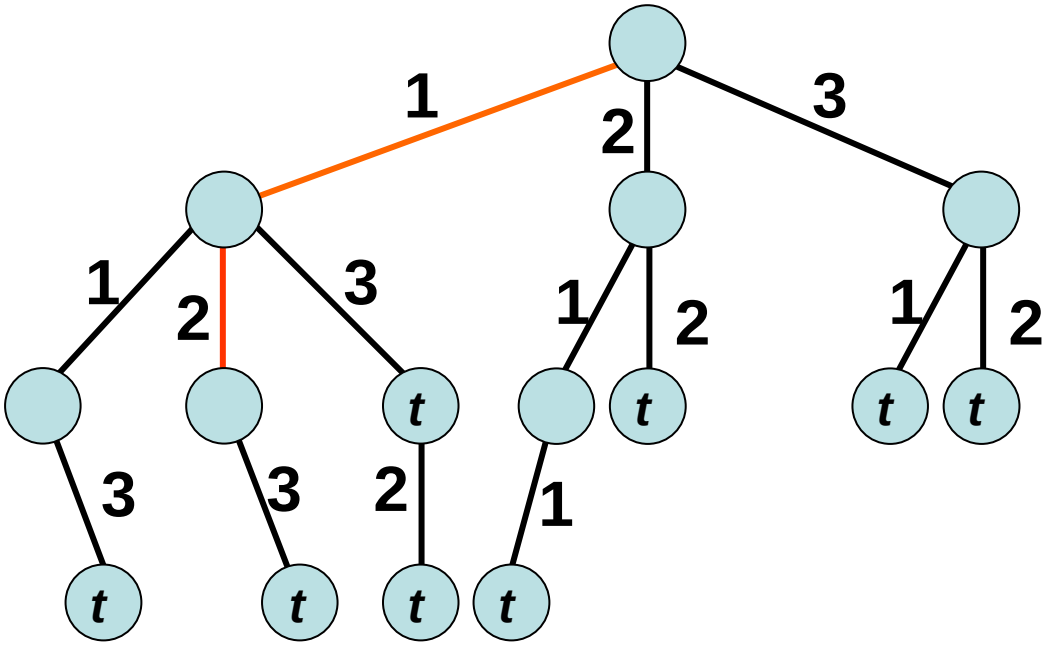
Busca em Árvore Digital

Exemplo: 1 2 3



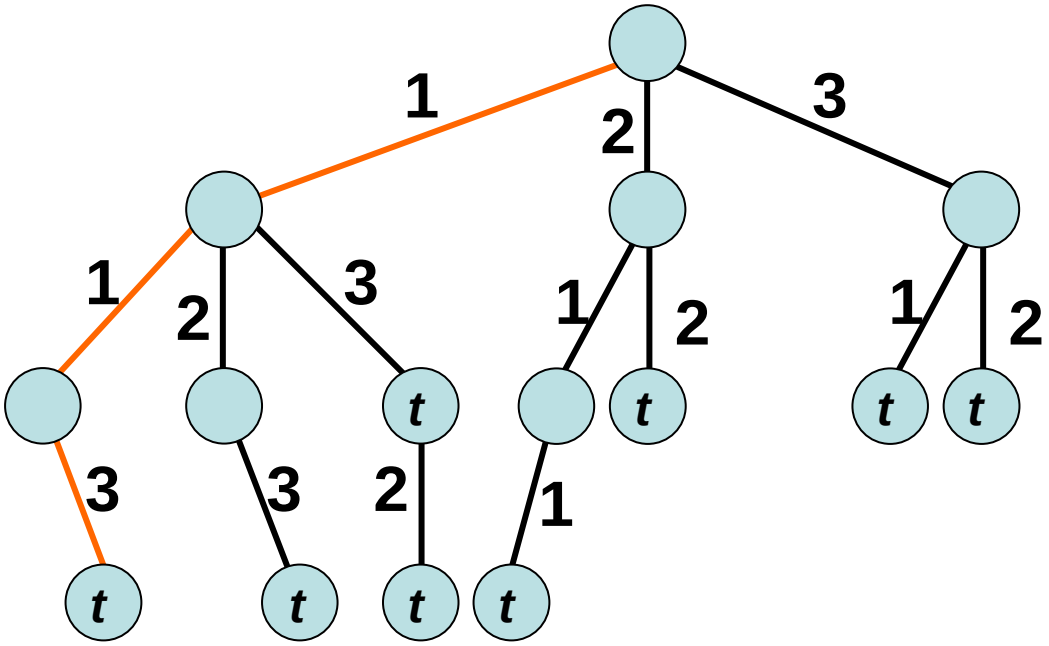
Busca em Árvore Digital

Exemplo: 1 2



Busca em Árvore Digital

Exemplo: 1134



Inserção em Árvore Digital

Busca a chave

Se a chave é válida, FIM!!

Senão

pt aponta para *w* tal que o caminho entre a raiz e *w* corresponde ao maior prefixo de chave que coincide com *x*. Incluir os dígitos restantes.

Inclusão em Árvore Digital

```
Algoritmo Inclusão
   pt ← ptraiz; l ← 0; a ← 0; buscadig(x, pt, l, a)
   se (a = 0) então
        para h ← l + 1 até k faça
            seja j a posição de d[h] no alfabeto
            ocupar(pt1)
            para i ← 1 até m faça
                pt1↑.pont[i] ← λ
            pt↑.pont[j] ← pt1; pt1↑.info ← nt
            pt ← pt1
        fim_para
        pt↑.info ← t
    senão " inclusão inválida"
```

Árvore Digital

COMPLEXIDADE

Busca

- São realizadas *k* iterações

- Determinação de *j* é *O*(log *m*)
  (busca binária)

```
Procedimento buscadig(x, pt, l, a)
  se (l < k) então
     seja j a posição de d(l+1) na ordenação do alfabeto
     se (pt↑.pont[j] ≠ λ) então
        pt ← pt↑.pont[j]
        l ← l + 1
        buscadig(x,pt,l,a)
  senão
     se (pt↑.info = t) então a ← 1
```

Se *m* = 1, então o algoritmo executa *k* passos

Logo, o algoritmo é *O*(*k* log *m* + *k*)

obs: Se *m* << *k*, então o algoritmo é *O*(*k*)

Árvore Digital

COMPLEXIDADE

Inserção

$k = l + k'$

l é o tamanho do maior prefixo

k' é o número de dígitos incluídos

Incluir um nó requer $O(m)$ operações

Isto é feito k' vezes

Logo, o algoritmo é $O(l \log m + k'm)$

```
Algoritmo Inclusão
  pt ← ptraiz; l ← 0; a ← 0; buscadig(x, pt, l, a)
  se (a = 0) então
     para h ← l + 1 até k faça
        seja j a posição de d[h] no alfabeto
        ocupar(pt1)
        para i ← 1 até m faça
           pt1↑.pont[i] ← λ
        pt↑.pont[j] ← pt1; pt1↑.info ← nt
        pt ← pt1
     fim_para
     pt↑.info ← t
   senão " inclusão inválida"
```

Árvore Digital

Exercício

Fazer o algoritmo de remoção em árvore digital e mostrar a complexidade.

Árvore Digital

Observações

A busca não depende do tamanho do arquivo. Depende apenas do tamanho da chave e do alfabeto.

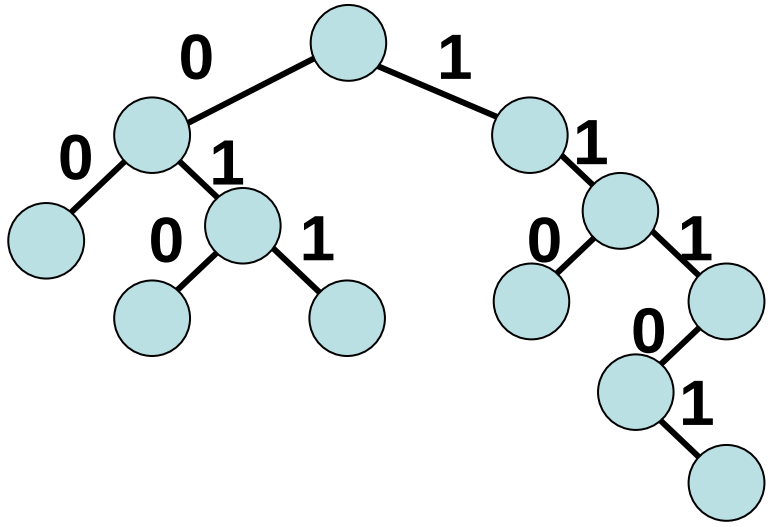
Ao se localizar um certo prefixo *x* , terão sido localizadas todas as chaves que possuem o prefixo *x*.

Utiliza grande quantidade de memória.

Árvore Digital Binária

- ***m* = 2**
- **Alfabeto = {0, 1}**

***Exemplo*:**
***S* = {00, 01, 010, 011, 11, 110, 11101}**



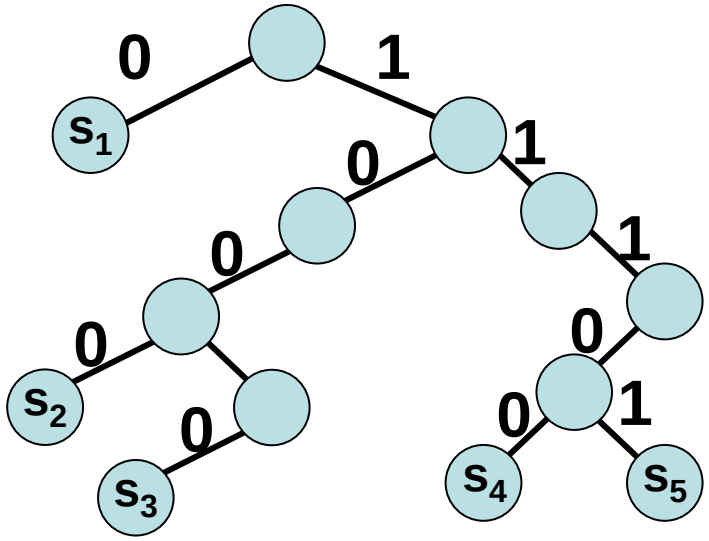
OBS: As chaves 01 e 11 são prefixos de outras chaves

Árvore Binária de Prefixo

É uma árvore digital binária tal que nenhuma chave é prefixo de outra.

Nesta árvore todas as chaves correspondem a folhas.

Exemplo:



s_1 – 0

s_2 – 1000

s_3 – 10010

s_4 – 11100

s_5 – 11101

Árvore Patrícia

- O termo Patricia vem de "*Practical Algorithm To Retrieve Information Coded In Alphanumeric*" .
- A árvore Patricia é uma *trie* comprimida, onde todo caminho que só possui vértices com um único filho são comprimidos em um único nó.
- Dessa forma, os rótulos das arestas representam substrings. E assim como a *trie*, a string é representada pela concatenação dos rótulos da raiz até a sua folha.
- Obs: É importante que não existam palavras que são prefixo de outras.

Árvore Patrícia

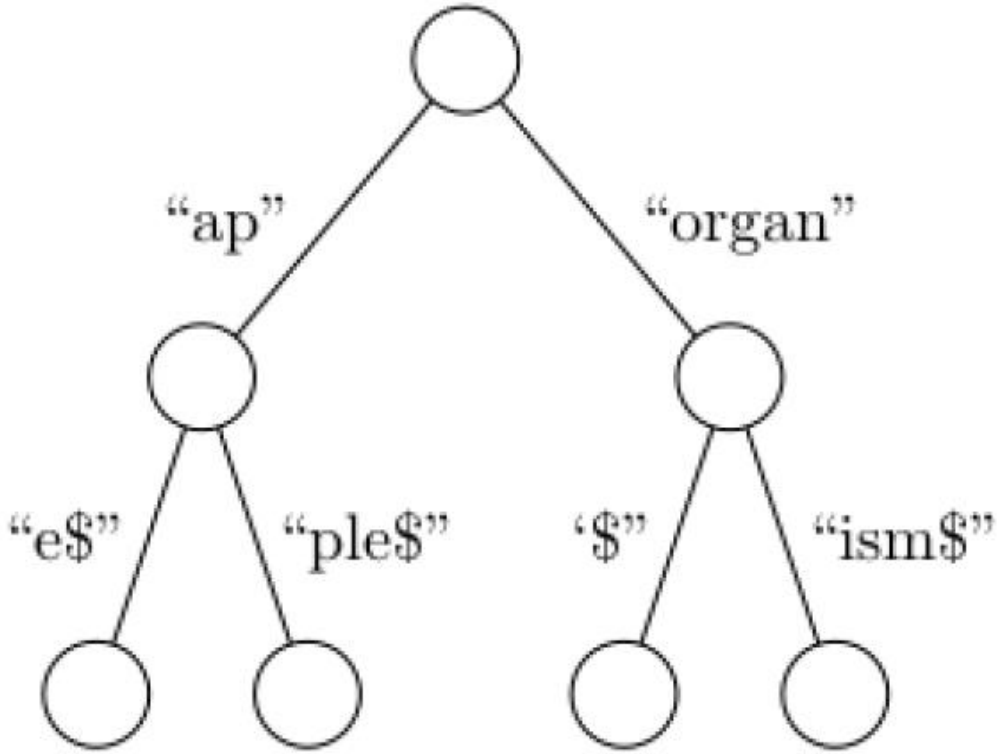


Figura: Exemplo da árvore Patricia das palavras “ape”, “apple”, “organ” e “organism”.

Árvore Patrícia

- As árvores Patricia podem ser representadas de maneira que ao invés de manter explicitamente substrings como rótulo das arestas, usar uma estrutura que mantêm um ponteiro e o tamanho da substring.
- Essa representação exige que as strings estejam armazenadas na memória.
- Obs: Pode ser mais econômico armazenar a informação do rótulo da aresta no filho, já que cada filho possui no máximo um pai.

Árvore Patrícia

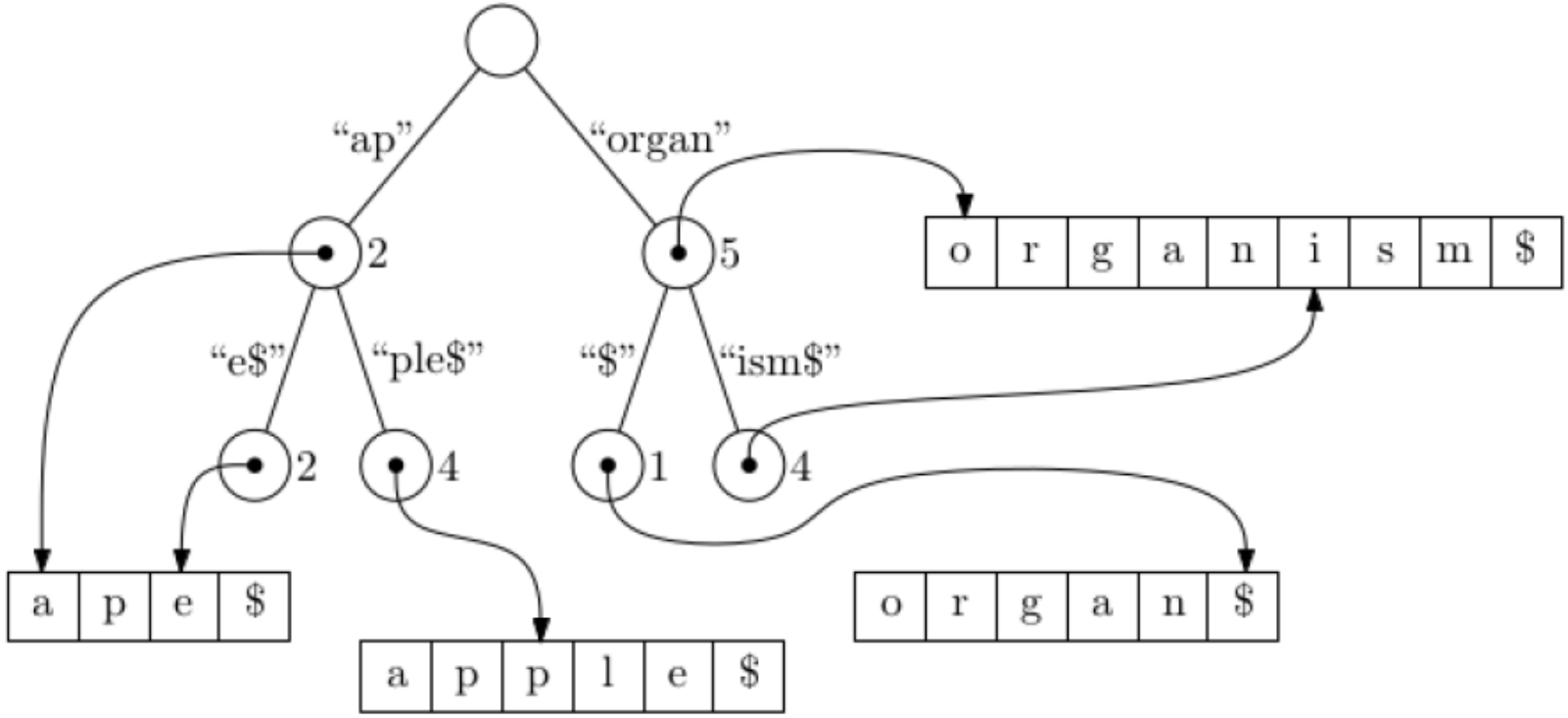


Figura: Exemplo da árvore Patricia das palavras "ape", "apple", "organ" e "organism".

Árvore Patrícia - Busca

- Similar à busca da *trie*, porém ao invés verificar se o símbolo da string corresponde ao símbolo da aresta, verifica-se se toda a substring da aresta corresponde a substring de s.
- Se em algum momento, não exista aresta que corresponda à substring de s, então s não pertence à arvore.
- Caso contrário, a folha que corresponde à string s é encontrada.

Árvore Patrícia - Busca

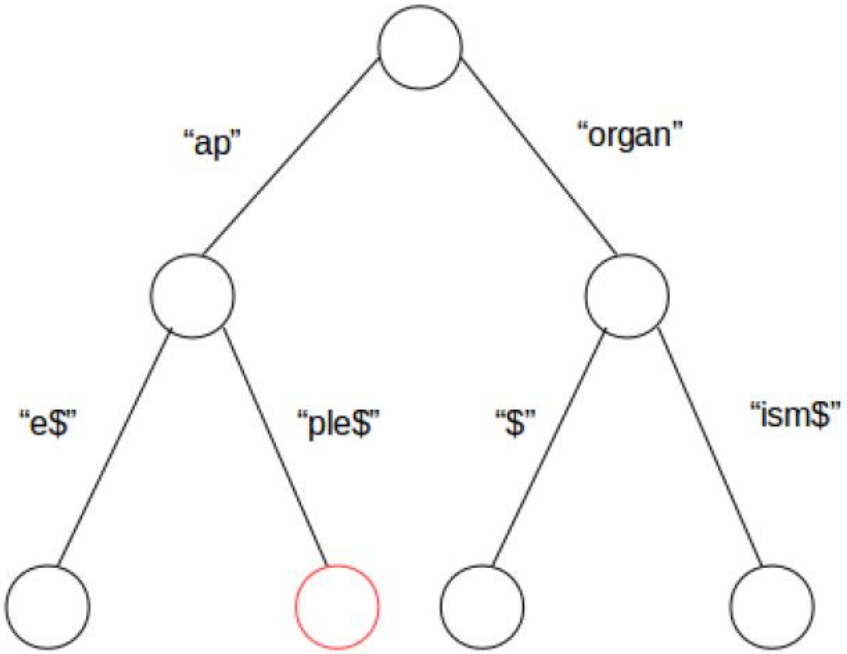


Figura: Busca pela palavra “apple$”.

Árvore Patrícia - Busca

- No pior caso, a busca encerra-se em uma folha tendo que percorrer todos o símbolos da string s.
- A cada nó, somente uma aresta precisa ser verificada, mantendo-se as arestas indexadas pelo primeiro símbolo, de forma similar à *trie R-Way*.
- Portanto, a busca é $O(|s|)$.

Árvore Patrícia - Inserção

- Assim como na *trie* é iniciada uma busca pelo nó.
- Se a busca foi interrompida a partir do segundo símbolo de uma aresta, a aresta é bifurcada em duas arestas.
  - A bifurcação terá uma aresta que leva ao nó apontado pela antiga aresta com o rótulo sendo o restante do rótulo não correspondido.
  - A outra aresta irá direcionar a um novo nó que representa a string s e com rótulo sendo o restante da substring de s não correspondida.
- Caso contrário, a busca é interrompida no primeiro símbolo e um novo nó é adicionado como filho do nó atual e com o rótulo da nova aresta sendo o restante de s.

Árvore Patrícia - Inserção

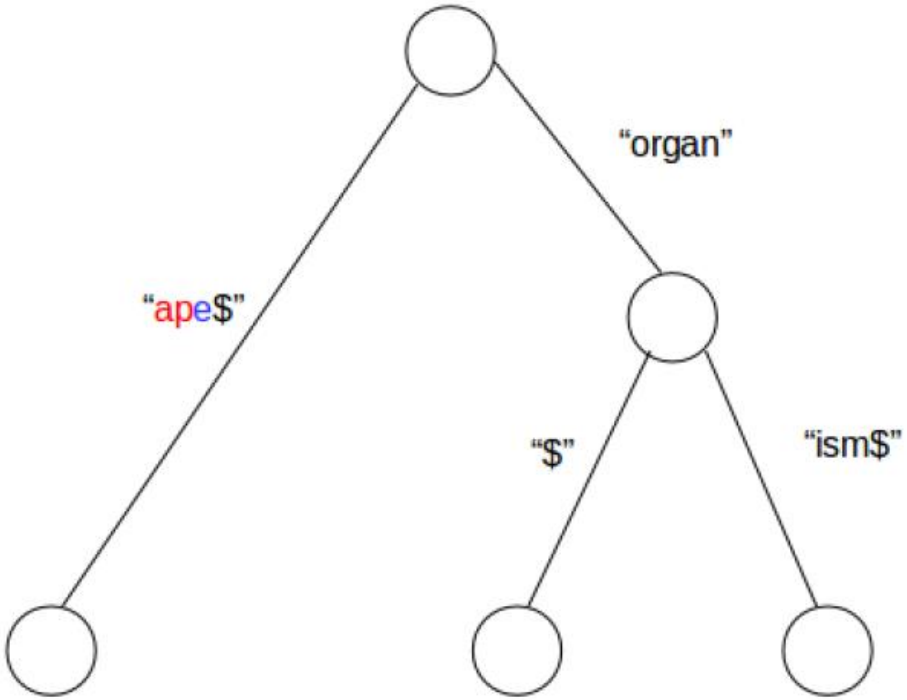


Figura: Inserir palavra "apple$".

Árvore Patrícia - Inserção

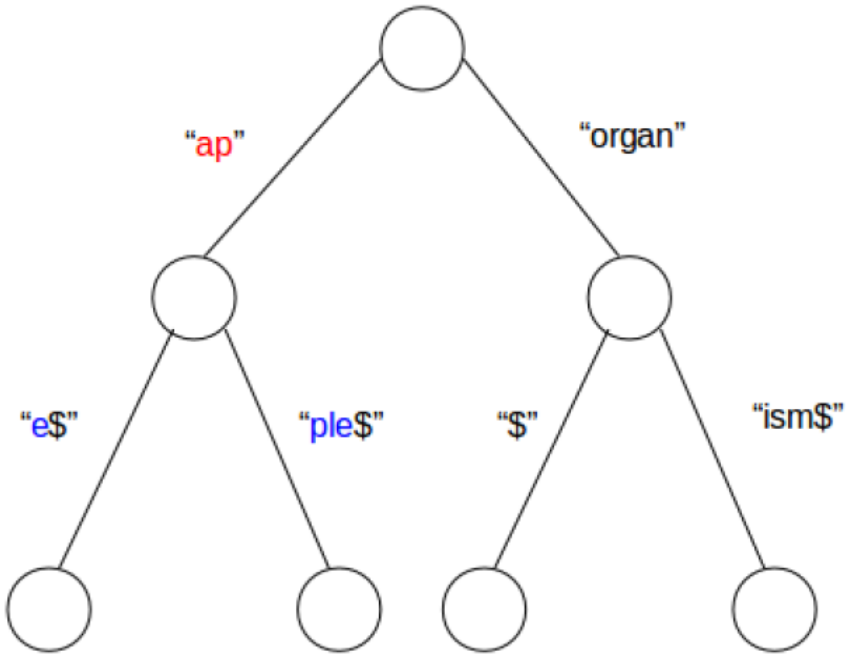


Figura: Após inserção.

Árvore Patrícia - Inserção

- O custo da inserção é dado pelo custo da busca mais o custo da bifurcação da aresta.
- A bifurcação cria dois novos nós e altera os ponteiros dos nós e os tamanhos dos rótulos, tendo o custo de $O(|\Sigma|)$.
- Sendo assim, o custo da inserção é de $O(|\Sigma| + |s|)$.

Árvore Patrícia - Remoção

- A remoção é o processo inverso da inserção.
- A string s é procurada na árvore.
- O nó correspondente a string s é removido.
- Caso o seu pai passe a possuir somente um filho, o pai é removido e a aresta do seu pai e a aresta do seu filho são unidas.

Árvore Patrícia - Remoção

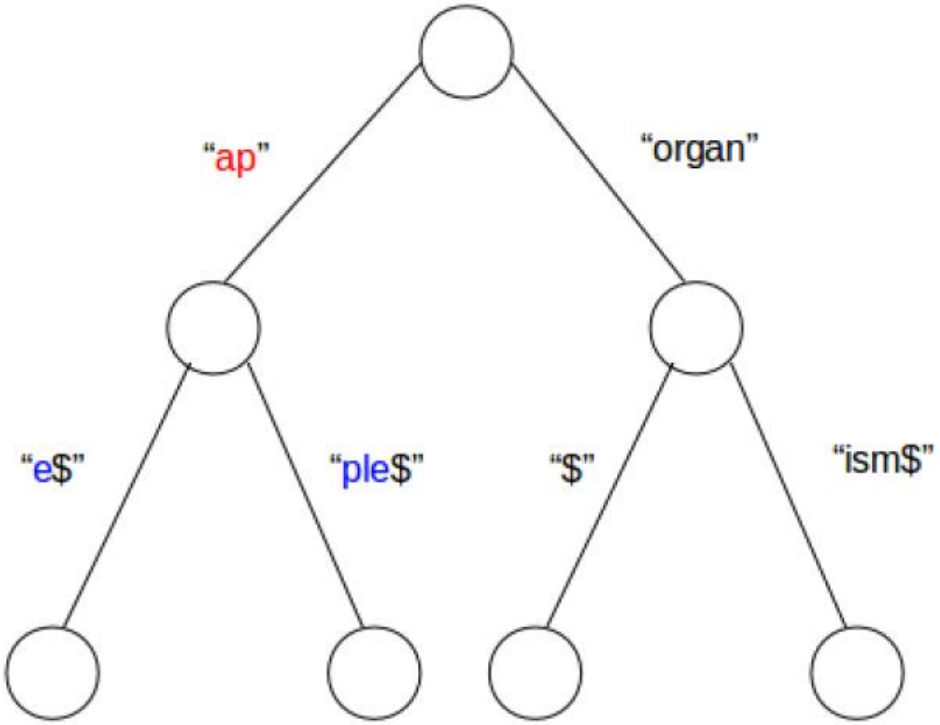


Figura: Remover palavra “ape$”.

Árvore Patrícia - Remoção

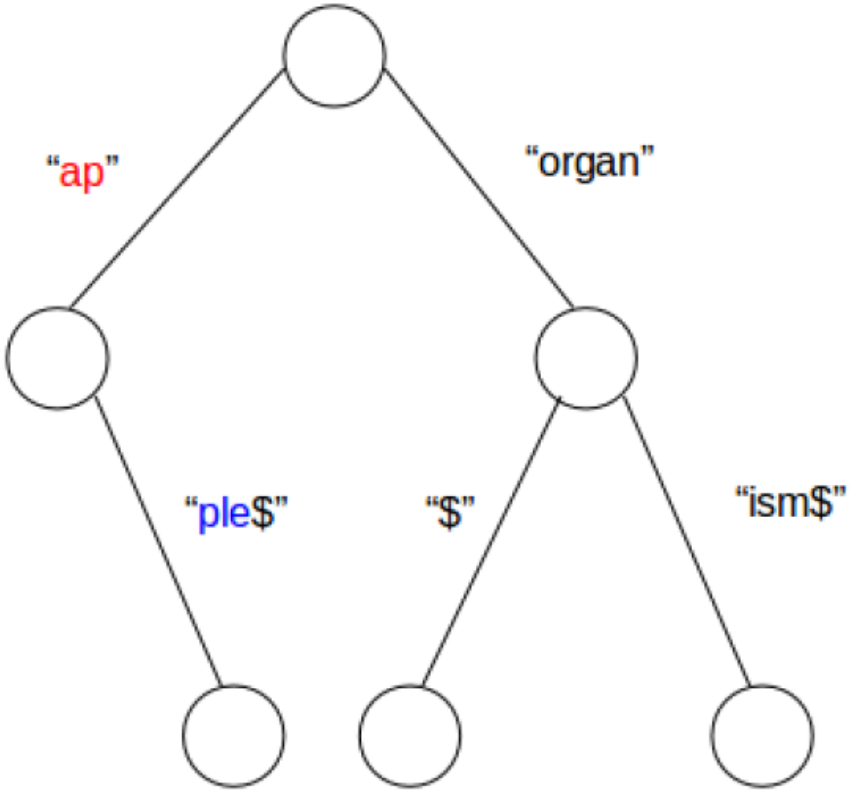


Figura: Após exclusão do nó da aresta “e$”.

Árvore Patrícia - Remoção

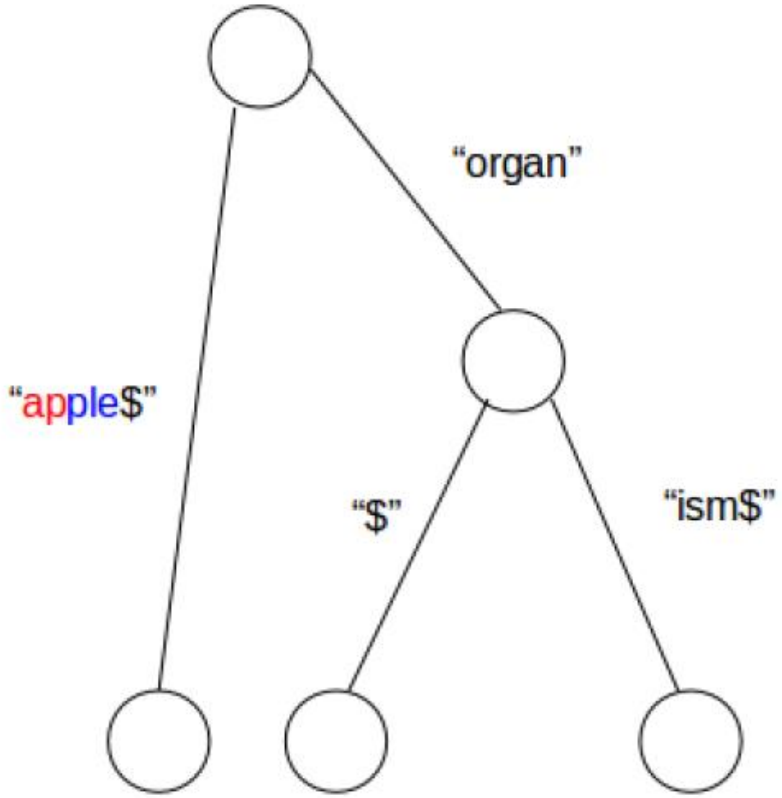


Figura: Após remoção.

Árvore Patrícia - Remoção

- A união das arestas podem ser feitas em tempo constante, somente com manipulação de inteiros e mudança do tamanho dos rótulos.
- O custo da busca é $O(|s|)$.
- Caso a remoção de um nó custe $O(|\Sigma|)$, o custo total da remoção é $O(|\Sigma| + |s|)$.

Árvore Patrícia

- Pelo fato que todo nó interno da árvore Patricia possui no mínimo dois nós, temos que o número de folhas é sempre maior que o número de nós internos (facilmente demonstrável por indução).
- Assim, o número de nós é $O(n)$, onde n é o número de folhas e também o número de strings da árvore, já que as folhas representam as strings.
- Dessa forma, temos que o armazenamento gasto pela Patricia é $O(|\Sigma| \times n)$.
- Entretanto as strings também precisam estar armazenadas, acarretando em um custo total de $O(|\Sigma| \times n + N)$ contra o custo de $O(|\Sigma| \times N)$ da *trie R-Way*.

Exercício

(a) Construa uma árvore Patricia para representar o seguinte conjunto de chaves:
(0010, 010, 00001, 10101, 10111, 1011000).

(b) Explique como é feita a remoção de um nó de uma árvore Patrícia, fazendo um roteiro passo a passo.

(c) Ilustre a execução do procedimento descrito no item (b) removendo da árvore construída no item (a) a chave 10101.