Pesquisa digital

Algoritmos e Estruturas de Dados II

Pesquisa digital

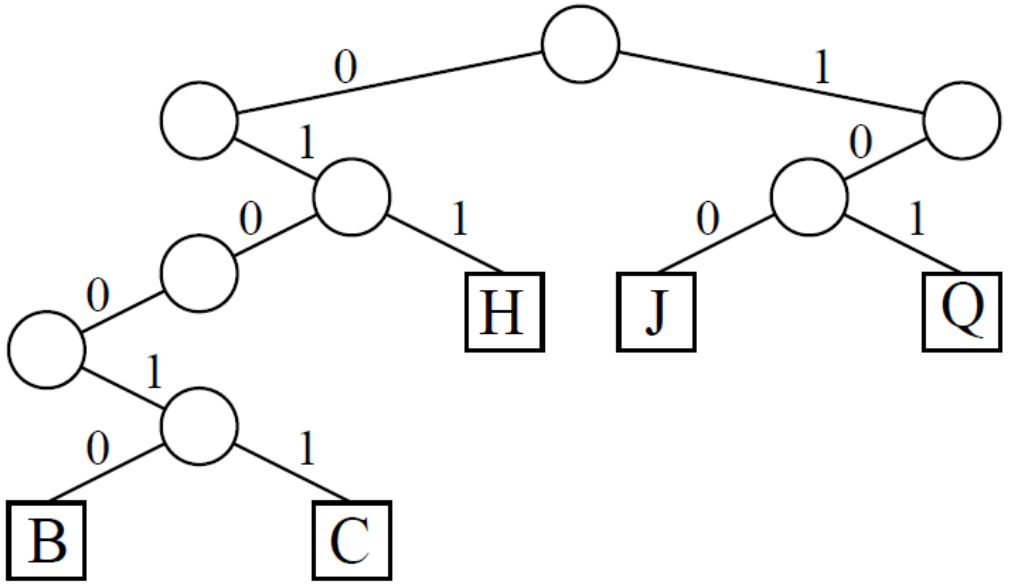
- A pesquisa digital usa a representação das chaves para estruturar os dados na memória
  - Por exemplo, a representação de um número em binário
  - A representação de um *string* com uma sequência de caracteres
- A pesquisa digital está para árvores binárias de pesquisa como radixsort está para os métodos de ordenação
  - Pesquisa não é baseada em comparação de chaves, mas sim em processamento feito sob a chave

Trie binária

- Cada nó no nível i representa o conjunto de todas as chaves que começam com a mesma sequência de i bits
- Cada nó tem duas ramificações, uma para as chaves cujo bit (i+1) é zero e outra para chaves cujo bit (i+1) é um

Trie binária – exemplo de pesquisa

- Busca em uma *trie* binária é parecida com pesquisa em árvore de busca
  - Mas não comparamos chaves
  - Percorremos a *trie* de acordo com os bits da chave



$$
\begin{aligned}
\mathrm{B} &= 010010 \\
\mathrm{C} &= 010011 \\
\mathrm{H} &= 011000 \\
\mathrm{J} &= 100001 \\
\mathrm{Q} &= 101000
\end{aligned}
$$

Trie – estruturas

```
struct no {
    struct no *esq;
    struct no *dir;
    struct registro *reg;
};

struct registro {
    int chave;
    /* outros dados */
};
```

Trie binária – pesquisa

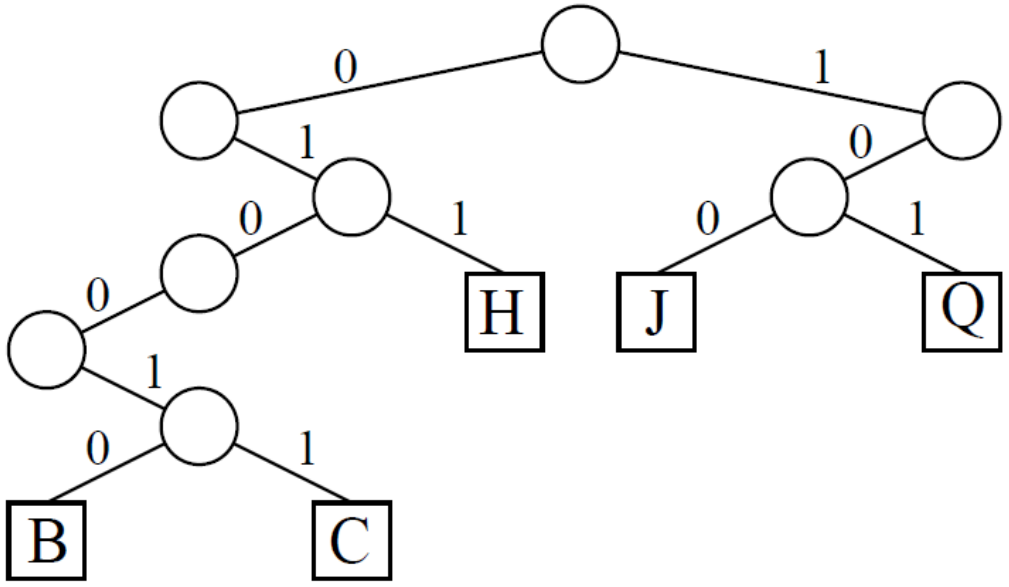
```
struct registro *pesquisaR(struct no *t, int chave, int p) {

    if(t == NULL) return NULL;
    if(t->esq == NULL && t->dir == NULL) {
        int regchave = t->reg->chave;
        if (regchave == chave) { return t->reg; }
        else { return NULL; }
    }
    if(digito(chave, p) == 0) { /*busca sub-árvore esquerda */
        return pesquisaR(t->esq, chave, p+1); }
    else { /* busca sub-árvore direita */
        return pesquisaR(t->dir, chave, p+1); }
}

struct registro *pesquisa(struct no *trie, int chave){
    return pesquisaR(trie, chave, 0);
}
```

Trie binária – exemplo de inserção

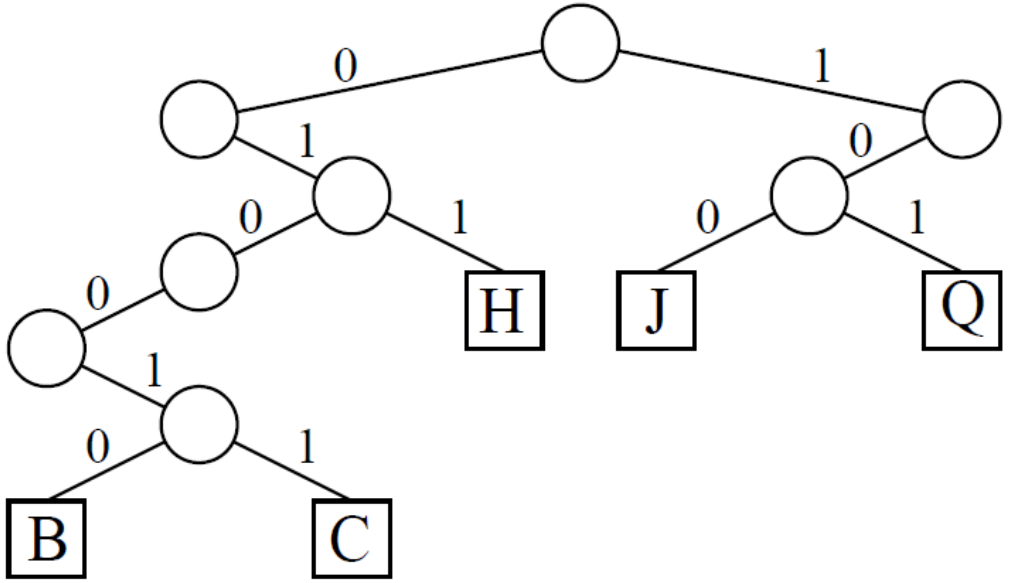
- Fazemos uma pesquisa na árvore para descobrir onde a chave será inserida
  - **Primeiro caso**: se o nó externo onde a pesquisa terminar for vazio, basta cria um novo nó para conter a nova chave
  - Inserindo W = 110110



$$\begin{aligned} \mathrm{B} &= 010010 \\ \mathrm{C} &= 010011 \\ \mathrm{H} &= 011000 \\ \mathrm{J} &= 100001 \\ \mathrm{Q} &= 101000 \end{aligned}$$

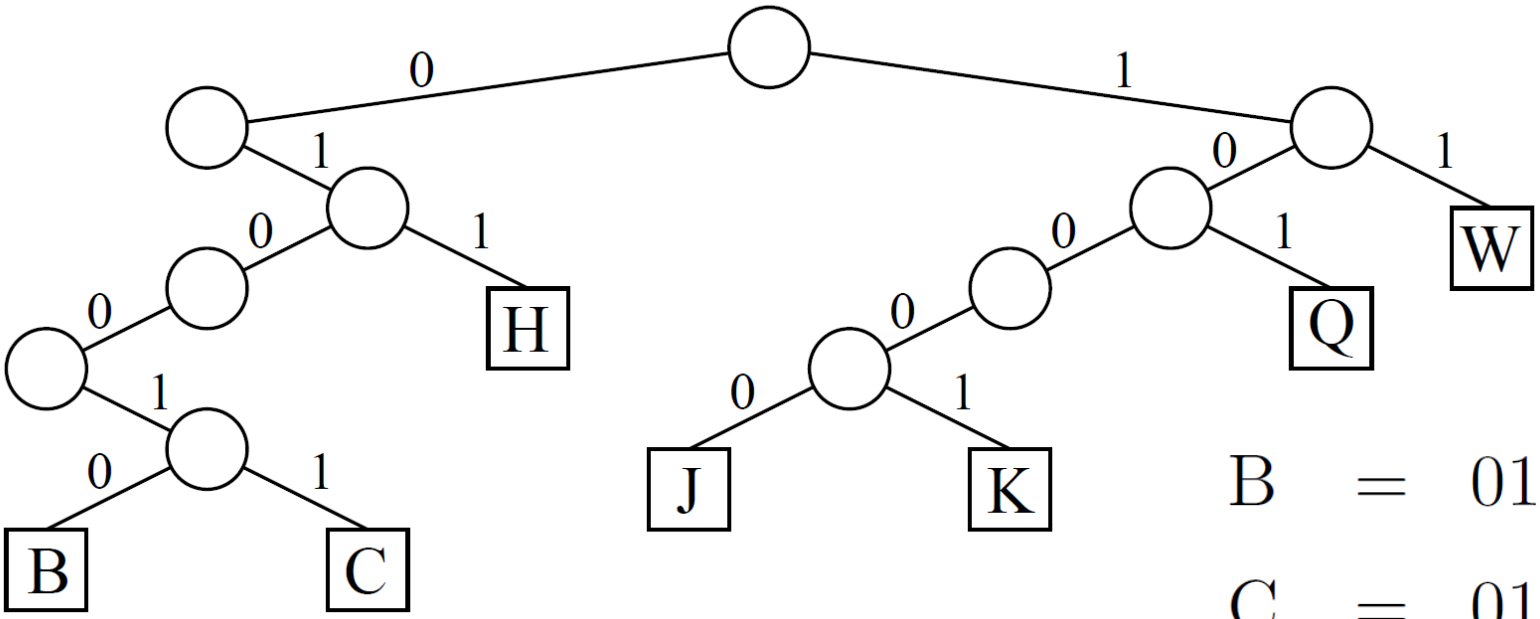
Trie binária – exemplo de inserção

- Fazemos uma pesquisa na árvore para descobrir onde a chave será inserida
  - **Segundo caso**: se o nó externo onde a pesquisa terminar tiver uma chave, criamos nós internos até encontrar o bit onde a nova chave difere da chave já existente
  - Inserindo K = 100010



$$
\begin{aligned}
\mathrm{B} &= 010010 \\
\mathrm{C} &= 010011 \\
\mathrm{H} &= 011000 \\
\mathrm{J} &= 100001 \\
\mathrm{Q} &= 101000
\end{aligned}
$$

Trie – exemplo de inserção



W = 110110

K = 100010

$$
\begin{aligned}
B &= 010010 \\
C &= 010011 \\
H &= 011000 \\
J &= 100001 \\
Q &= 101000
\end{aligned}
$$

Trie – inserção

```
struct no *insereR(struct no *t, struct registro *reg, int p){
int chave = reg->chave;

    if(t == NULL) return cria_trie(reg);
    if(t->esq == NULL && t->dir == NULL) {
        return separa(cria_trie(reg), t, p);
    }
    if(digito(chave, p) == 0) /* insere sub-árvore esquerda */
        t->esq = insereR(t->esq, reg, p+1);
    else /* insere na sub-árvore direita */
        t->dir = insereR(t->dir, reg, p+1);
    return t;
}

void insere(struct no **trie, struct registro *reg) {
    *trie = insereR(*trie, reg, 0);
}
```

Trie – inserção

```
struct no *separa(struct no *no1, struct no *no2, int p){

    novo = cria_trie(NULL);
    int chave1 = no1->reg->chave;
    int chave2 = no2->reg->chave;
    if(digito(chave1, p) == 0 && digito(chave2, p) == 0){
        novo->esq = separa(no1, no2, p+1);
    } else if(/* chave1 == 0 && chave2 == 1 */) {
        novo->esq = no1; novo->dir = no2;
    } else if(/* chave1 == 1 && chave2 == 0 */) {
        novo->dir = no1; novo->esq = no2;
    } else if(/* chave1 == 1 && chave2 == 1 */) {
        novo->dir = separa(no1, no2, p+1);
    }
    return novo;
}
```

Vantagens

- O formato das *tries* **não** depende da ordem em que as chaves são inseridas
  - Depende apenas dos valores das chaves

- Inserção e busca numa *trie* com N chaves aleatórias requer aproximadamente $\lg(N)$ comparações de bits no caso médio

- O pior caso é limitado pelo número de bits das chaves

Desvantagens

- Caminhos de uma única direção acontecem quando chaves compartilham vários bits em comum
  - Por exemplo, as chaves B (00010) e C (00011) são idênticas exceto no último bit
  - Requer inspeção de todos os bits da chave independente do número de registros na *trie*
- Os registros são armazenados apenas nas folhas, o que desperdiça memória em nós intermediário

Patricia

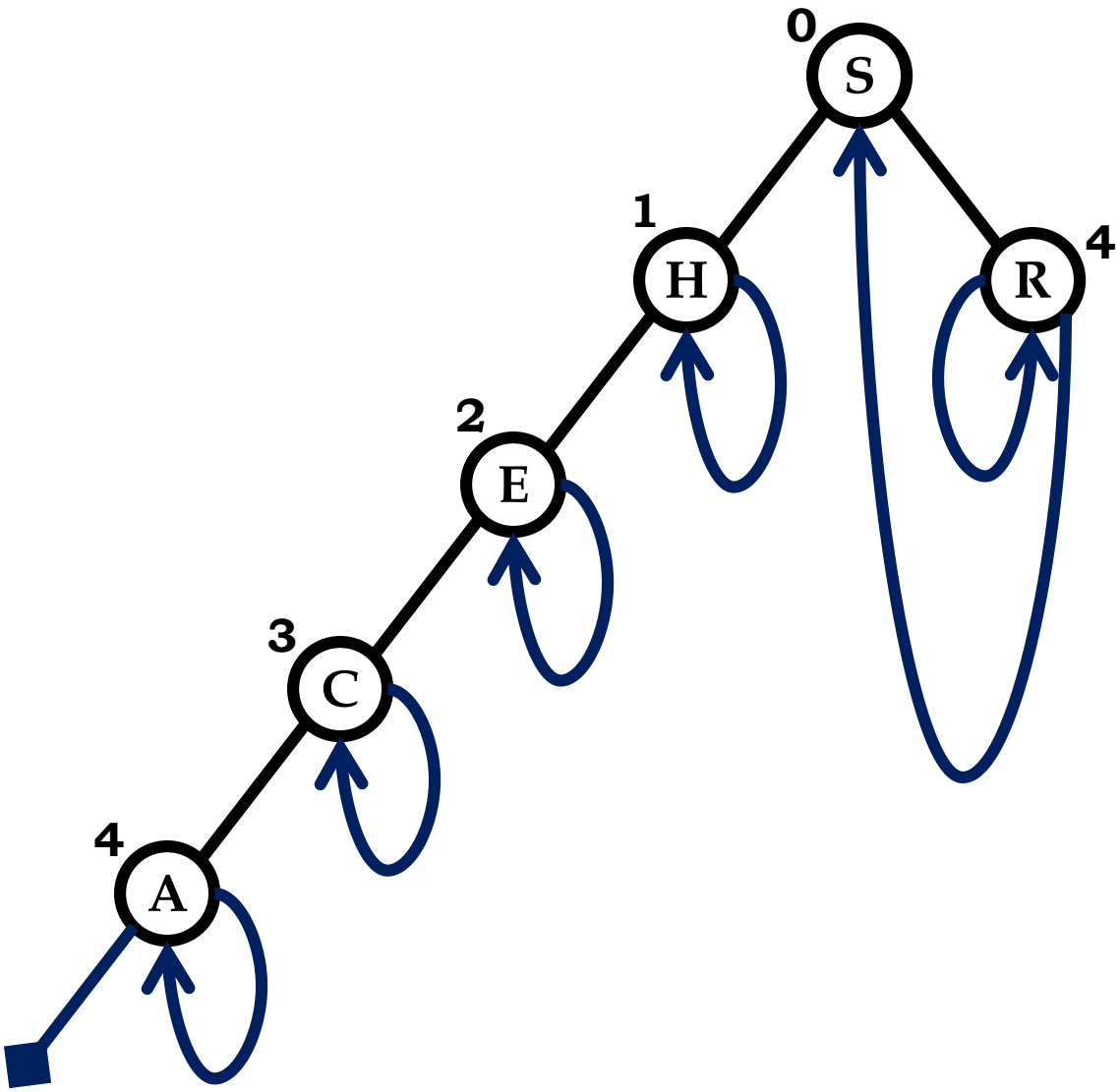
- Practical Algorithm To Retrieve Information Coded in Alphanumeric
- Criada por Morrison 1968 para recuperação de informação em arquivos de texto
- Estendido por Knuth em 73, Sedgewick em 88, Gonnet e Baeza-Yates em 91

Patricia

- Patricia usa o conceito de pesquisa digital, mas estrutura os dados de forma a evitar as desvantagens citadas das *tries*
  - Remove caminhos de única direção
  - Evita o desperdício de memória em nós internos

- Cada nó da árvore contém uma chave e um índice indicando qual bit deve ser testado para decidir qual ramo seguir

Patricia – exemplo de pesquisa



Busca por
R = 10010
I = 01001

Patricia – estruturas

```
struct no {
    struct no *esq;
    struct no *dir;
    int bit;
    struct registro *reg;
};

struct registro {
    int chave;
    /* outros dados */
};
```

Patricia – pesquisa

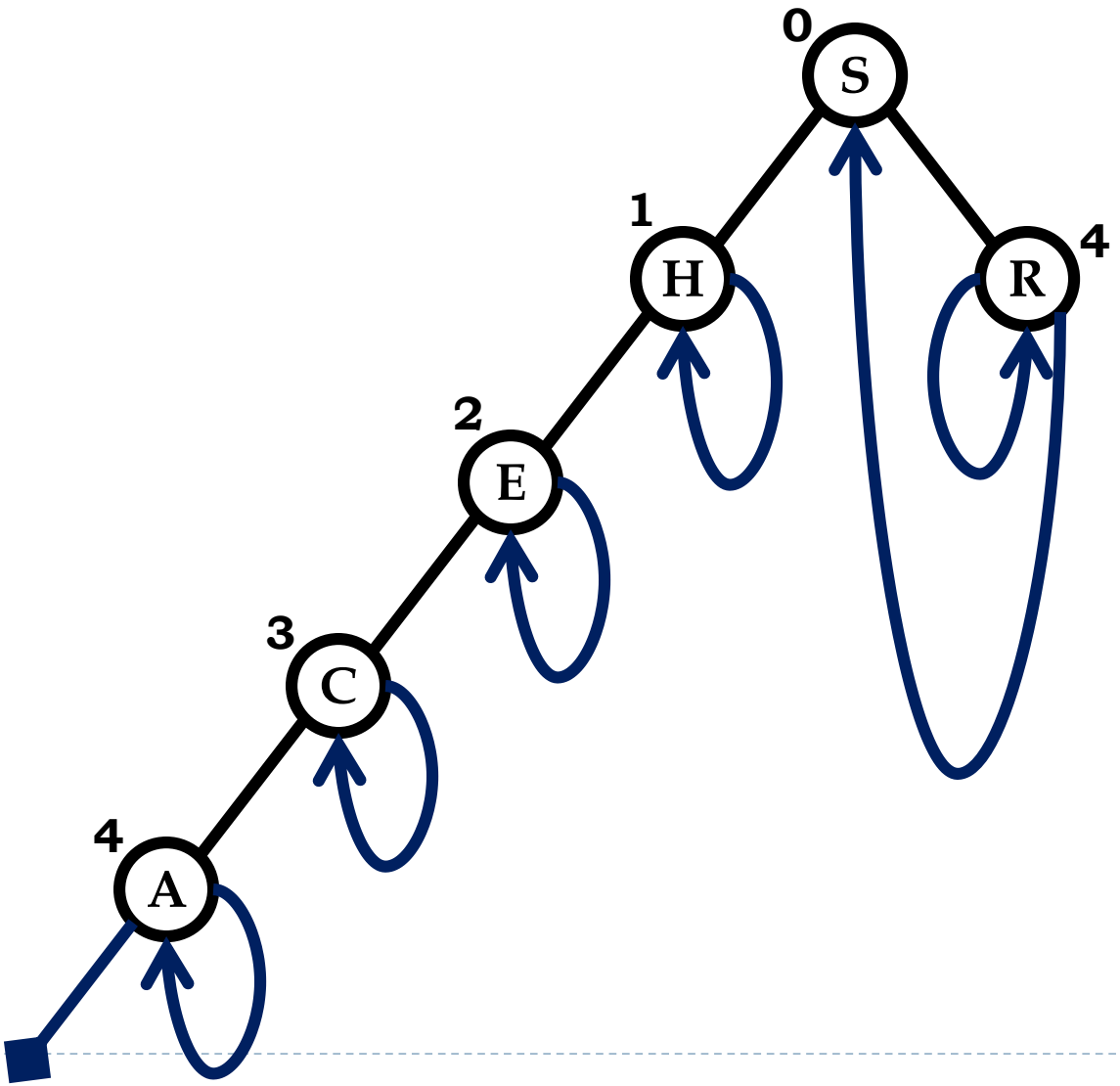
```
struct registro *pesquisaR(struct no *t, int chave,int bit) {

    if(t->bit <= bit) return t;
    if(digito(chave, t->bit) == 0) {
        return pesquisaR(t->esq, chave, t->bit);
    }
    else {
        return pesquisaR(t->dir, chave, t->bit);
    }
}

struct registro *pesquisa(struct no *pat, int chave) {
struct registro *reg;

    reg = pesquisaR(pat->esq, chave, -1);
    if(reg->chave == chave) { return reg; }
    else { return NULL; }
}
```

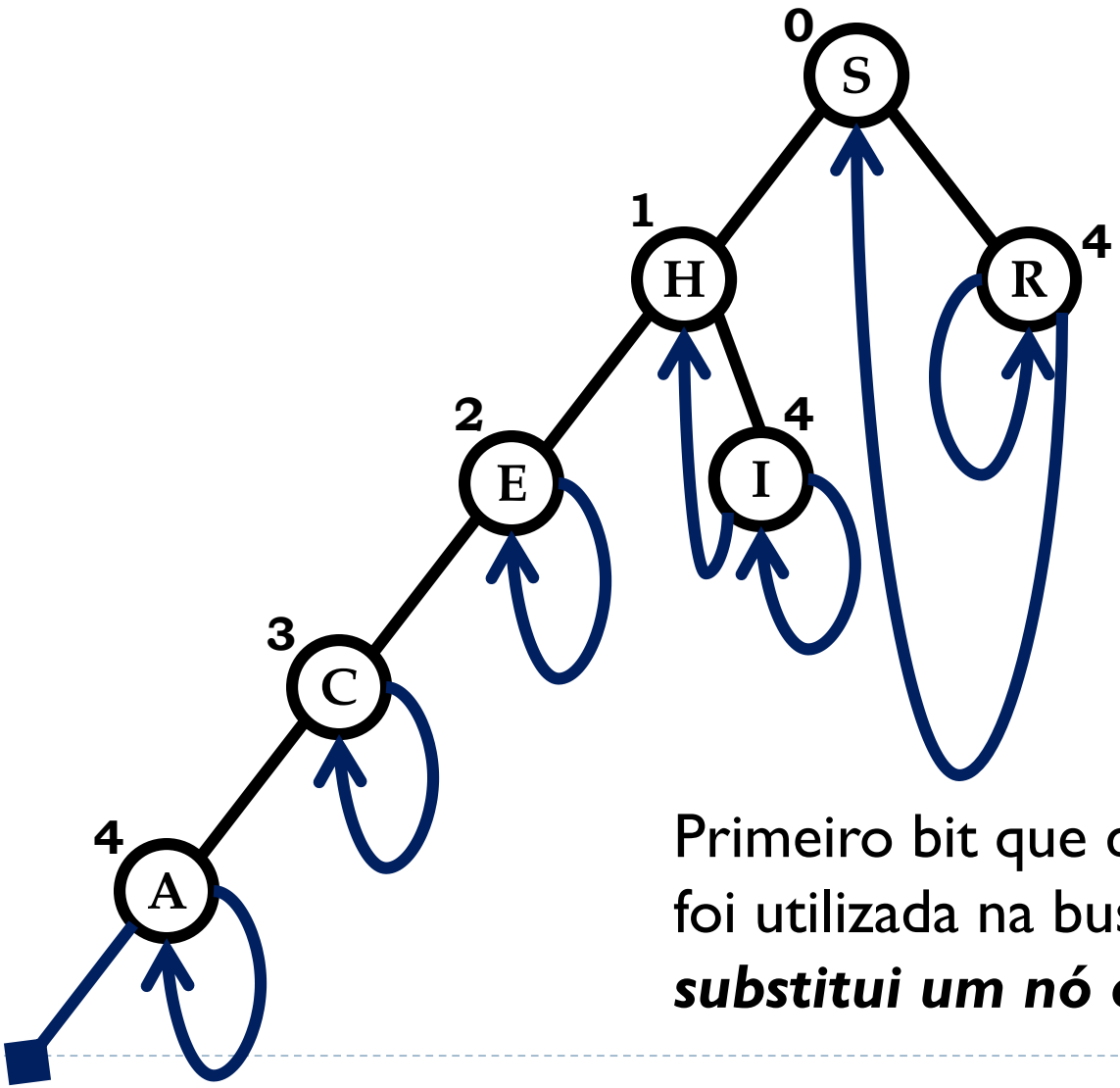
Patricia – exemplo de inserção



Inserindo
I = 01001

H = 01000

Patricia – exemplo de inserção (caso 1)

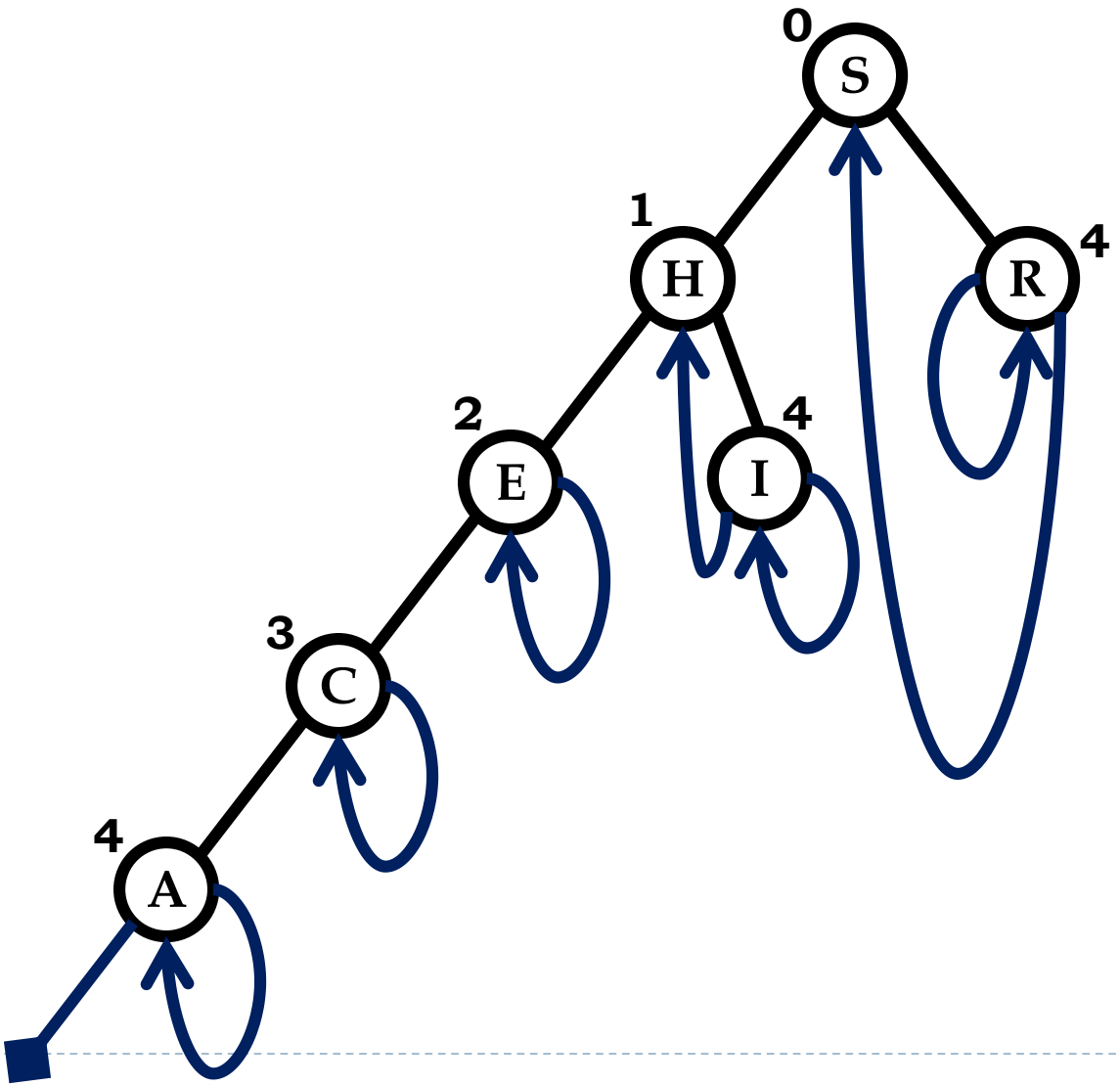


Inserindo
I = 01001

H = 01000

Primeiro bit que diferencia chaves não foi utilizada na busca (***novo nó substitui um nó externo***)

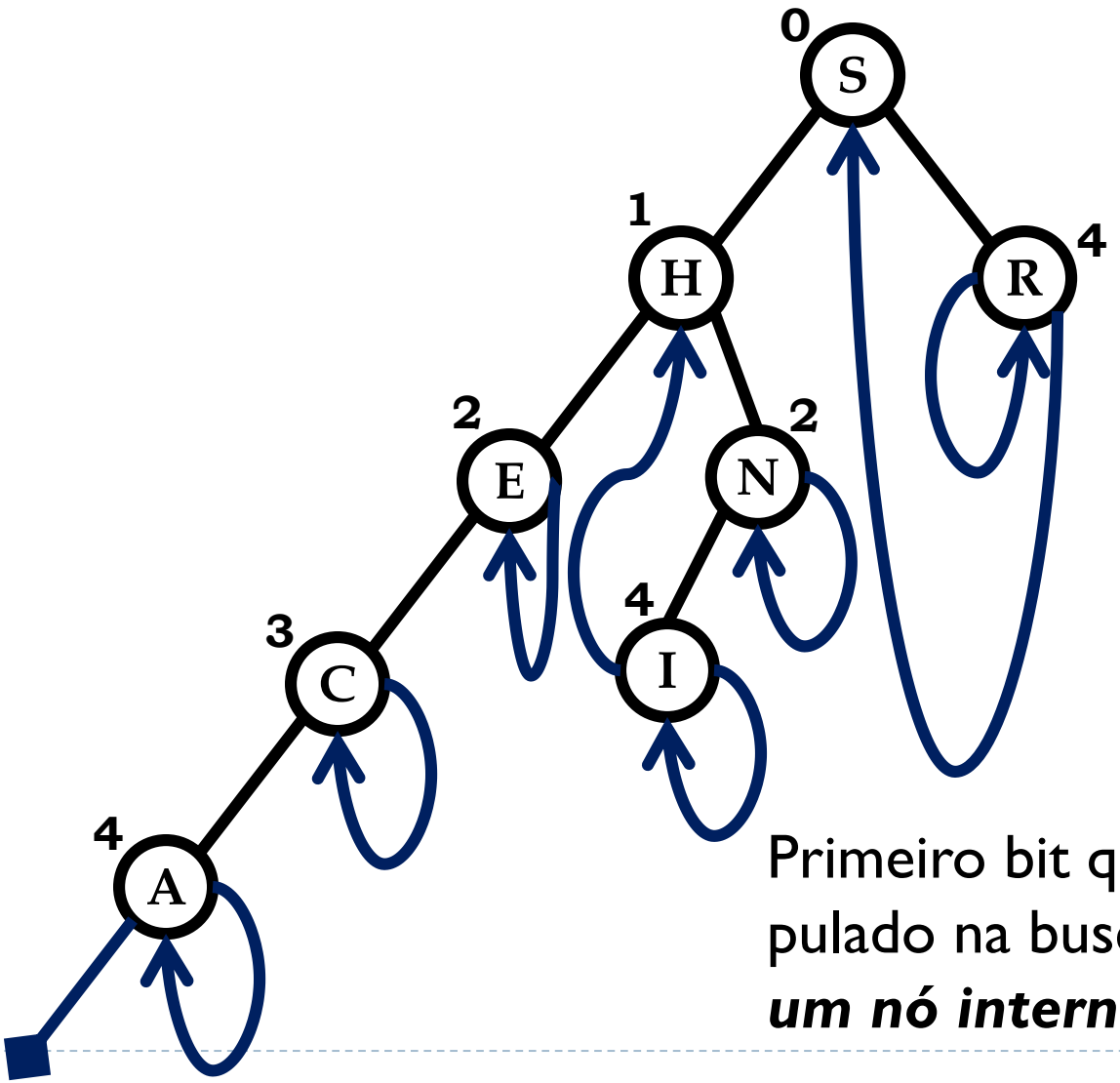
Patricia – exemplo de inserção



Inserindo
N = 01110

I = 01001

Patricia – exemplo de inserção (caso 2)



Inserindo
N = 01110

I = 01001

Primeiro bit que diferencia chaves foi pulado na busca (***novo nó substitui um nó interno***)

Casos da Inserção

- Caso 1: O novo nó substitui um nó externo
  - Acontece quando o bit que diferencia a nova chave da chave encontrada não foi utilizado na busca

- Caso 2: O novo nó substitui um nó interno
  - Acontece quando o bit que diferencia a nova chave da chave encontrada foi pulado durante a busca

```
struct no *inicializa() {
struct no *novo;
   novo = cria_pat(NULL, -1);
   novo->esq = novo->dir = novo;
   return novo;
}
```

Patricia – inserção

```
void insere(struct no *pat, struct registro *reg) {
int chave = reg->chave;
struct registro *ins;
int ichave;
int i = 0;

    ins = pesquisaR(pat->esq, chave, -1);
    ichave = ObtemChave(ins);  /* 0 se não há chaves */
    if (chave == ichave) return; /* chave já existe */

    /* procura pelo bit diferenciador */
    while (digito(chave, i) == digito(ichave, i)) i++;

    /* i é o bit diferenciador */
    pat->esq = insereR(pat->esq, reg, i, pat);
}
```

Patricia – inserção

```
struct no *insereR(struct no *t, struct registro *reg,
                   int bit, struct no *pai) {
int chave = reg->chave;
        /* caso 2 */          /* caso 1 */
    if((bit <= t->bit) || (t->bit <= pai->bit)) {
        struct no *x = cria_pat(reg, bit);
        x->esq = digito(chave, x->bit) ? t : x;
        x->dir = digito(chave, x->bit) ? x : t;
        return x;
    } else if (digito(chave, t->bit) == 0) {
        t->esq = insereR(t->esq, reg, bit, t);
    } else {
        t->dir = insereR(t->dir, reg, bit, t);
    }
    return t;
}
```