CI1057: Algoritmos e Estruturas de Dados III
Árvores Digitais

Profa. Carmem Hara

Departamento de Informática/UFPR

24 de abril de 2024

Árvores Digitais

- Vantajosas para chaves grandes
- Vantajosas para chaves de tamanho variável (p.ex. palavras e frases)
- São eficientes se houver uma forma eficiente de acessar os bits ou bytes (elementos que compõem a chave)

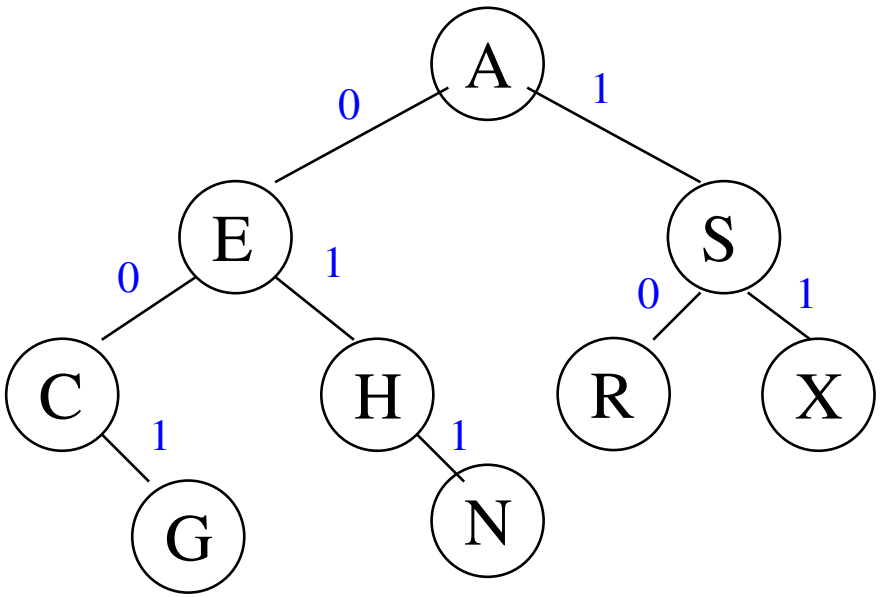
Árvores Digitais

- As chaves são representadas por uma sequência de caracteres ou dígitos
- Tipos de árvores
  - Árvore de Pesquisa Digital
  - Trie
  - Árvore Patricia

Árvore de Pesquisa Digital Binária

- Considera a representação binária da chave
- Cada nível da árvore considera apenas um dígito da chave (bit) e não a chave inteira

A = 00001
S = 10011
E = 00101
R = 10010
C = 00011
H = 01000
X = 11000
N = 01110
G = 00111



Árvore de Pesquisa Digital Binária - Interface

```
1  typedef long TipoItem;
2  typedef struct Nodo *ApNodo;
3  typedef struct Nodo {
4    TipoItem item;
5    ApNodo esq, dir;
6  } Nodo;
7
8  ApNodo criaArv();
9  void freeArv( ApNodo raiz );
10 ApNodo busca( TipoItem v, ApNodo raiz);
11 ApNodo insere( TipoItem v, ApNodo raiz );
12 ApNodo remove( TipoItem v, ApNodo raiz );
```

Árvore de Pesquisa Digital Binária - Busca

```
1  ApNodo busca( TipoItem v, ApNodo raiz){
2    return buscaR( v, raiz, 0 );
3  }
4
5  ApNodo buscaR( TipoItem v, ApNodo p, int b){
6    if( p == NULL )
7      return NULL;
8    if( eq( p->item, v) )
9      return p;
10   if (digito( v, b ) == 0 )
11     return buscaR( v, p->esq, b+1 );
12   else
13     return buscaR( v, p->dir, b+1 );
14 }
```

Árvore de Pesquisa Digital Binária - Inserção

Exemplo: inserção das chaves A, S, E, R, C, H, I, N

```
A 00001
S 10011
E 00101
R 10010
C 00011
H 01000
I 01001
N 01110
```

Árvore de Pesquisa Digital Binária - Inserção

```
1  ApNodo insere( TipoItem v, ApNodo raiz){
2    return insereR( v, raiz, 0 );
3  }
4
5  ApNodo insereR( TipoItem v, ApNodo p, int b ){
6    if( p == NULL )
7      return criaNodo( v );
8    if( eq( p->item, v) )
9      return p;
10   if (digito( v, b ) == 0 )
11     p->esq = insereR( v, p->esq, b+1 );
12   else
13     p->dir = insereR( v, p->dir, b+1 );
14   return p;
15 }
```

Árvore de Pesquisa Digital Binária - Remoção

Pergunta: pode ser usada a mesma estratégia da remoção da árvore binária de busca?

Árvore de Pesquisa Digital Binária - Pior Caso

Exemplo: inserção das chaves P, H, D, B, A

```
P 10000
H 01000
D 00100
B 00010
A 00001
```

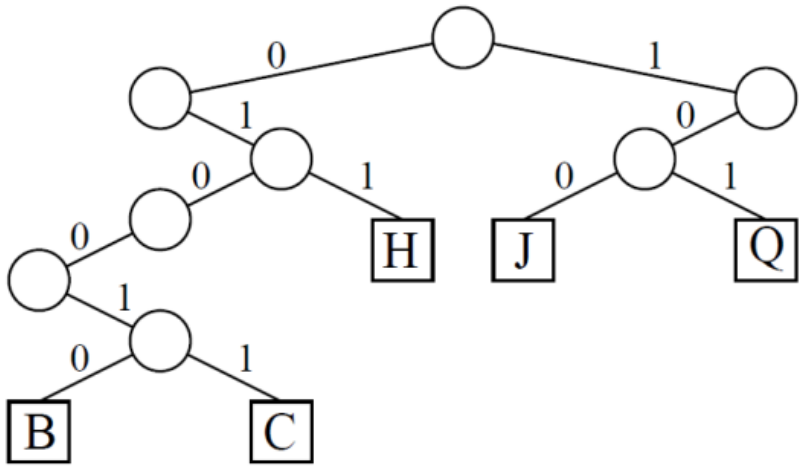
- árvore degenerada
- altura da árvore é o tamanho das chaves
- exceto por 00000, nenhuma outra chave de 5 bits aumenta a altura da árvore

Árvore de Pesquisa Digital Binária - Características

- as chaves estão *em algum nodo* no caminho especificado por seus bits
- as chave não estão ordenadas – a chave de um nodo n pode ser maior ou menor que os valores na subárvore com raiz em n
- se as chaves contém w bits, a árvore pode conter 2^w chaves distintas
- a altura da árvore é limitada a quantidade de bits da chave
- busca de chaves de 32 bits requer menos que 32 comparações
  - mesmo para bilhões de chaves
  - comparável ao desempenho de uma árvore rubro-negra
- pode ser usada para chaves de tamanho variável

Trie

- Também chamada de **árvore de prefixo**
- É uma árvore de busca digital que mantem as chaves em ordem
- Chaves são armazenadas apenas nas folhas
- Chaves podem ter tamanho variável, mas inicialmente vamos consider que tem tamanho fixo



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

Trie

Uma trie á uma árvore binária que possui chaves associadas aos nodos folhas e definida recursivamente da seguinte forma:

- a trie para um conjunto vazio de chaves é apenas um apontador NULL;
- a trie para apenas uma chave é composta apenas por um nodo folha que contém esta chave
- a trie para um conjunto de chaves maior que um é composta por um nodo interno, sendo o filho esquerdo uma trie contendo chaves cujo bit inicial é 0 e o filho direito é uma trie contendo chaves cujo bit inicial é 1. O primeiro bit é então removido para a construção das subárvores direita e esquerda.

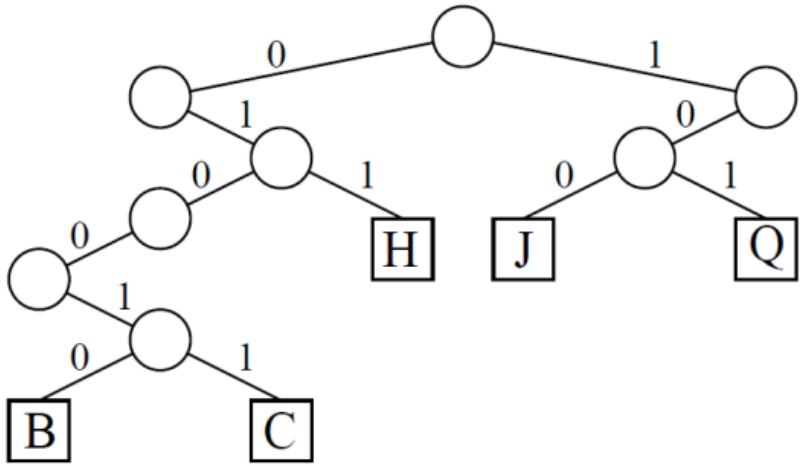
Trie - Interface

```
1  #define ITEMNULO -1
2
3  typedef long TipoItem;
4  typedef struct Nodo *ApNodo;
5  typedef struct Nodo {
6    TipoItem item;
7    ApNodo esq, dir;
8  } Nodo;
9
10 ApNodo criaArv();
11 void freeArv( ApNodo raiz );
12 ApNodo busca( TipoItem v, ApNodo raiz);
13 ApNodo insere( TipoItem v, ApNodo raiz );
14 ApNodo remove( TipoItem v, ApNodo raiz );
```

- Assume a existência de uma constante ITEMNULO que é armazenado como valor do item em nodos que não são folhas
- Pode ser melhorado para que nodos internos contenham apenas apontadores e folhas apenas chaves
- **Pergunta:** como alterar para chaves de tamanho variável?

Trie - Busca

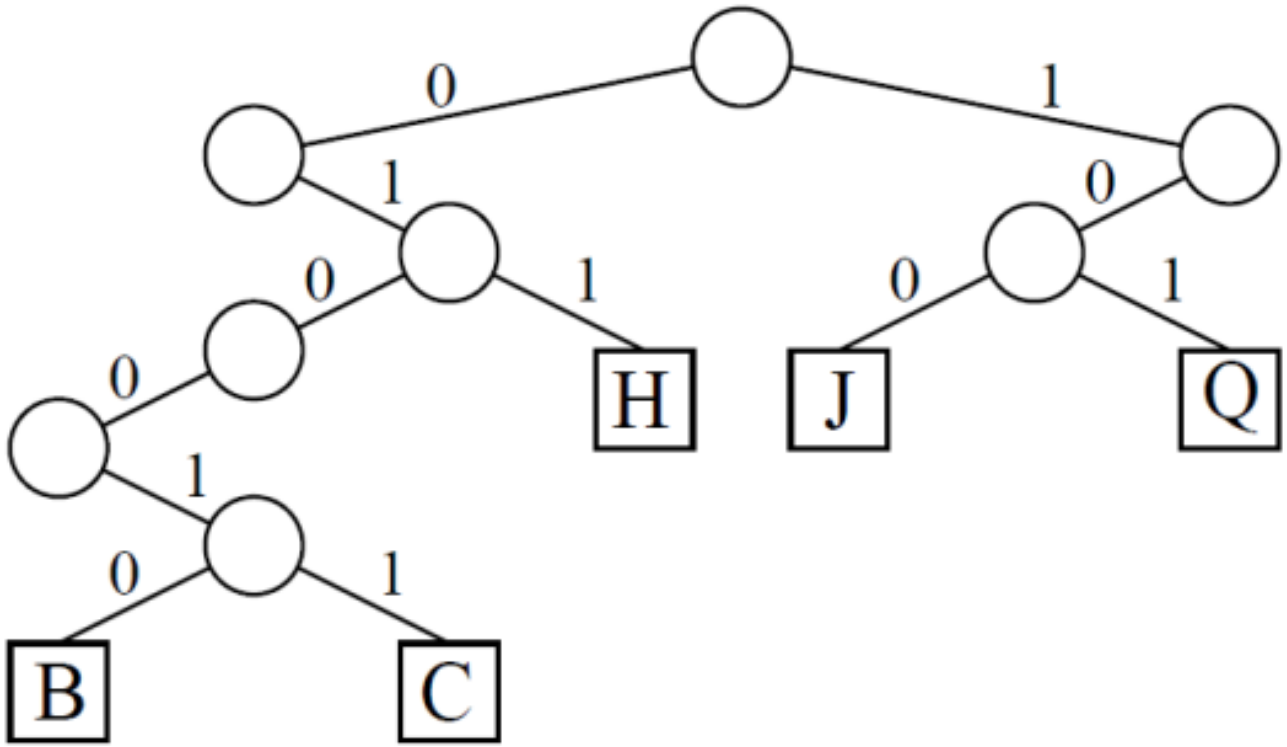
- Busca da chave C = 010011
- Busca da chave M = 101000
- Busca da chave E = 010110



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

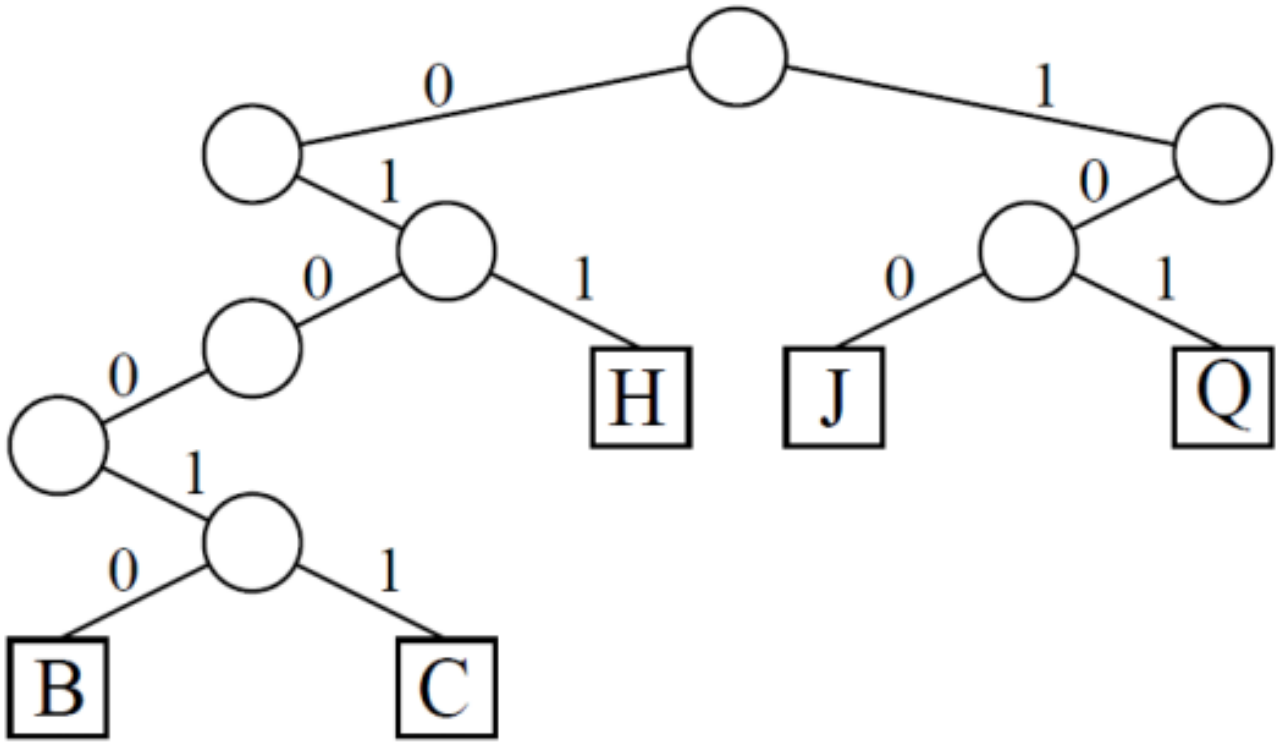
Trie - Inserção - Caso 1

- Passo 1: busca na árvore para encontrar onde inserir a chave
- Caso 1: Se encontrar um nodo externo, criar um novo nodo com a chave
- Inserção da chave W = 110110



Trie - Inserção - Caso 2

- Caso 2: Se a busca terminar em uma folha. São criados múltiplos nodos internos até encontrar o bit onde a nova chave difere da chave já existente.
- Inserção da chave K = 100010



Trie - Inserção

```
1  ApNodo insere( TipoItem v, ApNodo raiz){
2    return insereR( v, raiz, 0 );
3  }
4
5  ApNodo insereR( TipoItem v, ApNodo p, int b ){
6    if( p == NULL )
7      return criaNodo( v );
8    if( folha( p ))
9      return split( criaNodo( v ), p, b );
10   if (digito( v, b ) == 0 )  p->esq = insereR( v, p->esq, b+1 );
11   else                       p->dir = insereR( v, p->dir, b+1 );
12   return p;
13 }
14
15 ApNodo split( ApNodo p1, ApNodo p2, int b){
16   int d1, d2;   ApNodo n;
17
18   n = criaNodo( ITEMNULO );
19   d1 = digito(p1, b);  d2 = digito(p2, b);
20   if( d1 == d2 )
21     if( d1 == 0 ) n->esq = split( p1, p2, b+1 );
22     else          n->dir = split( p1, p2, b+1 );
23   else
24     if( d1 == 0 ){ n->esq = p1;  n->dir = p2;}
25     else         { n->esq = p2;  n->dir = p1; }
26   return n;
27 }
```

Trie - Inserção - Exemplo

Exemplo: inserção das chaves A, S, E, R, C, H, I, N

```
A 00001
S 10011
E 00101
R 10010
C 00011
H 01000
I 01001
N 01110
```

Trie - Vantagens

- O formato das tries não depende da ordem em que as chaves são inseridas
  - depende apenas dos valores das chaves
  - "balanceamento natural"
- Inserção e busca com N chaves aleatórias requer aproximadamente $lg)N)$ comparações de bits no caso médio
  - não depend do tamanho da chave
- O pior caso é limitado pelo número de bits das chaves

Trie - Desvantagens

- Caminhos de uma única direção acontecem quando chaves compartilham vários bits em comum
  - P.ex. chaves B (00010) e C (00011) só diferem no último bit
  - requer a 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 intermediários
  - uma trie com N chaves tem aproximadamente $1.44N$ nodos

Referências

- Livro Sedgewick Cap. 15 (Seção 15.1, 15.2)