CI1057: Algoritmos e Estruturas de Dados III
Árvores Digitais M-árias

Profa. Carmem Hara

Departamento de Informática/UFPR

29 de abril de 2024

Árvore Digitais N-árias

- generalização de tries, na qual chaves são codificadas em uma base qualquer, não necessariamente binária

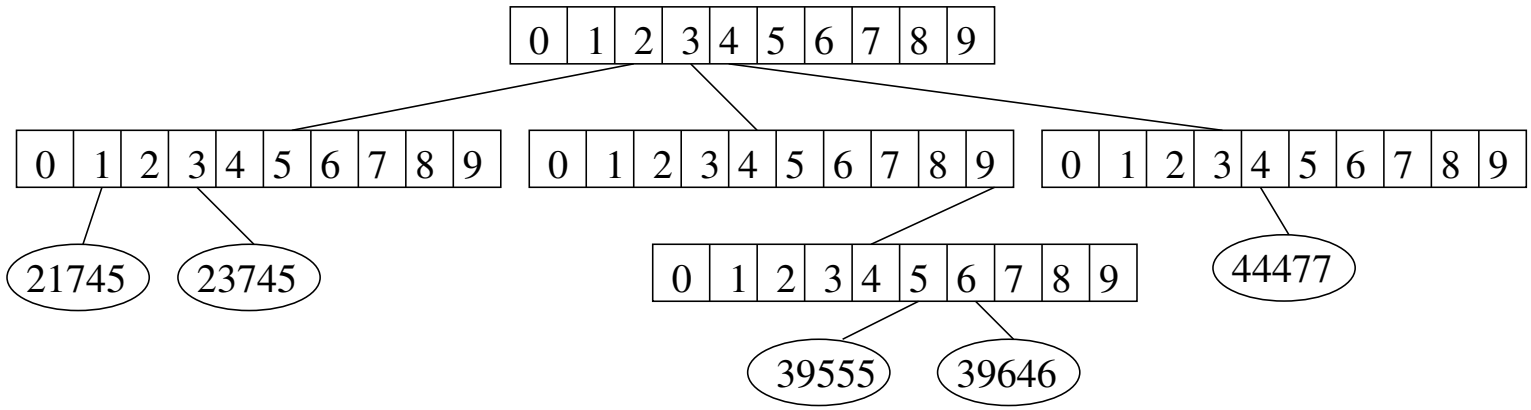
Trie N-ária

Uma trie n-ária possui chaves armazenadas nas folhas. Ela é definida recursivamente da seguinte forma:

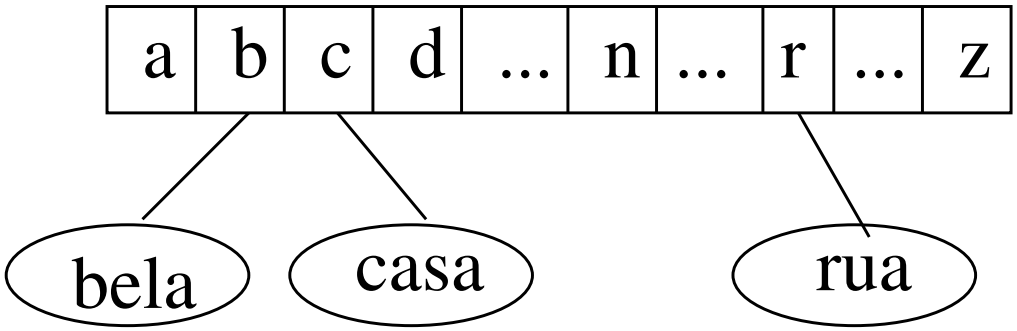
- uma trie para um conjunto vazio de chaves corresponde ao apontador nulo;
- uma trie com uma única chave corresponde a uma folha contendo esta chave;
- uma trie com cardinalidade maior que um é um nodo interno com apontadores referentes a tries com chaves começando com cada um dos dígitos possíveis, com este digito desconsiderado na construção das subárvores.

Exemplo: Base decimal, chave com 5 dígitos

39646
39555
21745
23745
44477



Exemplo: Palavras



Inserir as palavras:

- cara
- numero

Estrutura de Dados

```
1  #define NUMLETRA 26
2  #define ITEMNULO ""
3  #define indC(c) (c-'a')       /* indC('a')=0, indC('b')=1, ... */
4
5  typedef struct Nodo *ApNodo;
6  typedef struct Nodo {
7    TipoItem item;
8    ApNodo prox[NUMLETRA];
9  } Nodo;
10
11 void criaArv( ApNodo *raiz );
12 void freeArv( ApNodo raiz );
13 ApNodo busca( TipoItem v, ApNodo raiz);
14 ApNodo insere( TipoItem v, ApNodo raiz );
15 void escreve( ApNodo raiz );
```

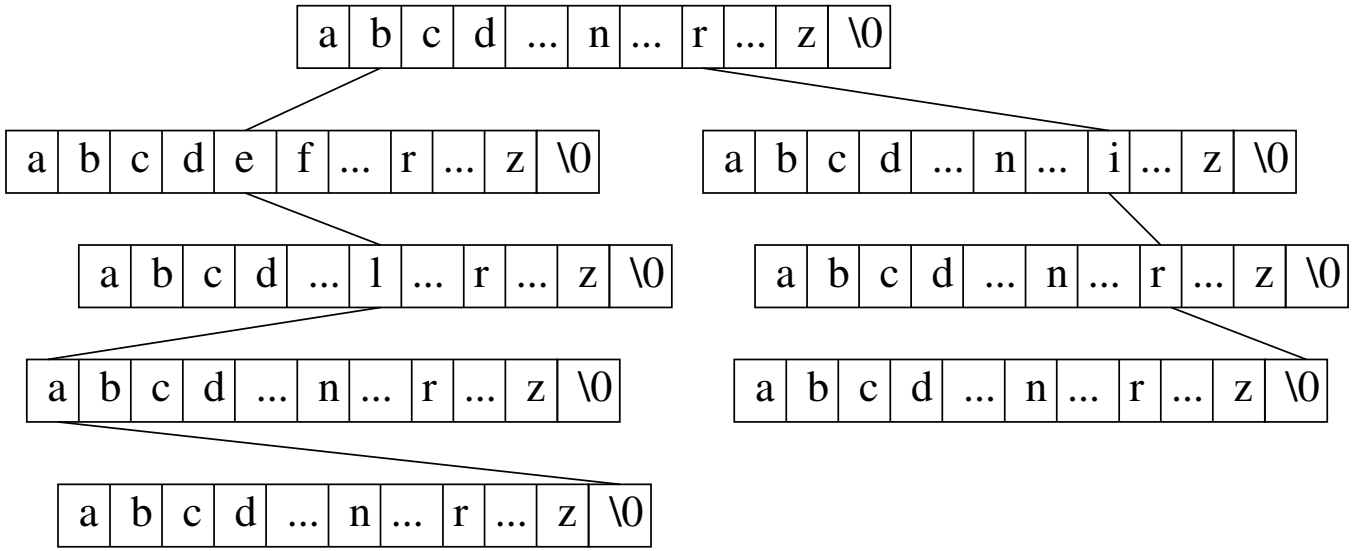
Trie existencial

- Não guarda informação sobre o registro, apenas se uma determinada chave está presente ou não na trie n-ária.

Uma trie existencial para um conjunto de chaves é definida recursivamente da seguinte forma:

- uma trie para um conjunto vazio de chaves corresponde ao apontador nulo;
- uma trie para um conjunto não vazio de chaves corresponde a um nodo interno com apontadores para nodos filhos contendo valores para cada valor de dígito possivel. Nestas subárvores o primeiro digito é removido para sua construção, de forma recursiva.

Trie existencial - Exemplo



Inserir as palavras:

- bebe
- rico

Quando uma chave pode ser prefixo de outra

Duas possíveis soluções:

- chaves de tamanho fixo
- "marcador"de final de chave (p.ex '\0')

Estrutura de Dados

```
1  #define NUMLETRA 27            /* [a-z] mais '\0' */
2  #define DIGITONULO 26
3  #define indC(c) (c-'a')        /* indC('a')=0, indC('b')=1, ... */
4
5  typedef struct Nodo *ApNodo;
6  typedef struct Nodo {
7    ApNodo prox[NUMLETRA];
8  } Nodo;
9
10 void criaArv( ApNodo *raiz );
11 void freeArv( ApNodo raiz );
12 ApNodo busca( TipoItem v, ApNodo raiz);
13 ApNodo insere( TipoItem v, ApNodo raiz );
14 void escreve( ApNodo raiz );
```

Trie N-ária e Existencial - Características

- árvore com altura baixa
- grande número de apontadores nulos

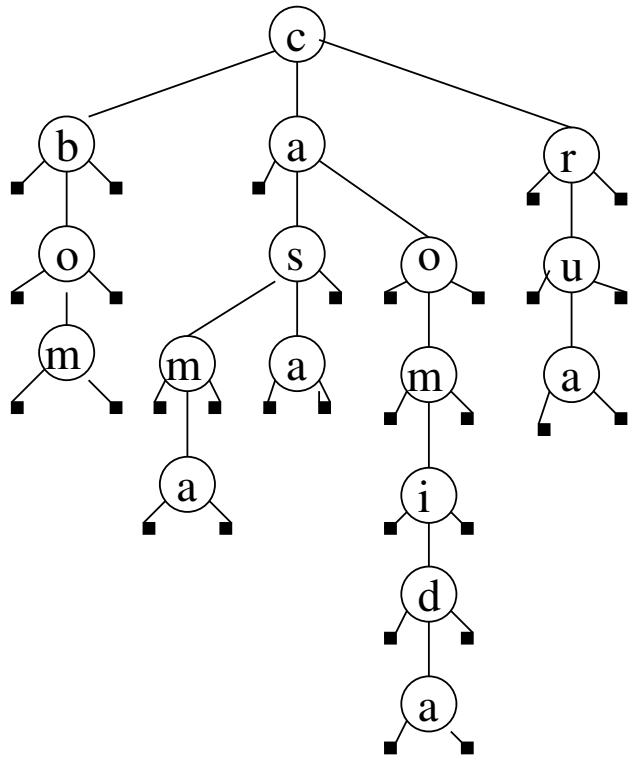
Consequências:

- baixo tempo de busca e inserção
- grande desperdício de espaço

Trie Ternária

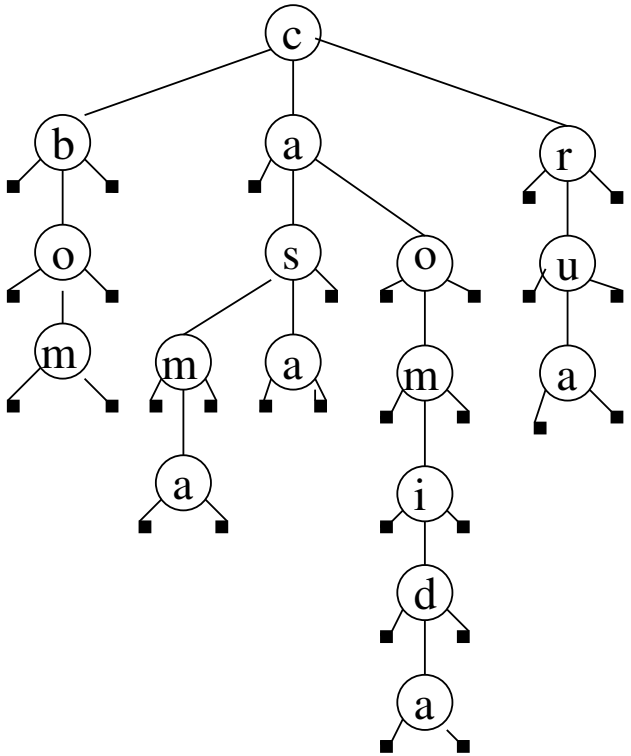
- similar às árvores binárias digitais, mas não necessariamente binárias
- cada nodo tem 3 apontadores: para chaves que começam com um dígito menor que o corrente, iguais e maiores

casa
bom
rua
cama
comida



Árvore Ternária - Exemplo

casa
bom
rua
cama
comida



Inserir as chave:

- bazar
- casado

Árvore Ternária - Características

- Tempo de busca: proporcional ao tamanho da chave
- Número de links: no máximo 3 vezes o tamanho total do conjunto de chaves
- Fácil manipulação

Estrutura de Dados

```
1  #define NUMLETRA 27            /* [a-z] mais '\0' */
2  #define indC(c) (c-'a')        /* indC('a')=0, indC('b')=1,... indC('z')=25 */
3  #define DIGITONULO 26
4
5  typedef struct Nodo *ApNodo;
6  typedef struct Nodo {
7    char c;
8    ApNodo dir, meio, esq;;
9  } Nodo;
10
11 void criaArv( ApNodo *raiz );
12 void freeArv( ApNodo raiz );
13 ApNodo busca( TipoItem v, ApNodo raiz);
14 ApNodo insere( TipoItem v, ApNodo raiz );
15 void escreve( ApNodo raiz );
```

Árvore Ternária - Criação da estrutura

```
1  void criaArv( ApNodo *raiz){
2    *raiz = NULL;
3  }
4
5  ApNodo criaNodo( int d ){
6    ApNodo n;
7
8    n = (ApNodo) malloc( sizeof( Nodo ));
9    n->d = d;
10   n->dir = n->meio = n->esq = NULL;
11   return n;
12 }
```

Árvore Ternária - Busca

```
1  ApNodo busca( TipoItem v, ApNodo raiz ){
2    return buscaR( v, raiz, 0 ); }
3
4  ApNodo buscaR( TipoItem v, ApNodo p, int w ){
5    int i = digito(v, w);
6    if( p == NULL )
7      return NULL;
8    if( p == DIGITONULO )
9      return p;
10   if(i < p->d ) return buscaR( v, p->esq, w );
11   if( i == h->d ) return buscaR( v, p->meio, w+1 );
12   return buscaR( v, p->dir, w );
13 }
```

Árvore Ternária - Inserção

```
1  ApNodo insere( TipoItem v, ApNodo raiz){
2    return insereR( item, raiz, 0);
3  }
4
5  ApNodo insereR( Item v, Apontador p, int w ){
6    int i;
7
8    i = digito(v, w);
9    if( p == NULL )
10     p = criaNodo( i );
11   if( i == DIGITONULO )
12     return p;
13   if(i < p->d )          p->esq=  insereR( v, p->esq, w );
14   else if( i == h->d ) p->meio= insereR( v, p->meio, w+1 );
15   else                   p->dir = insereR( v, p->dir, w );
16   return p;
17 }
```

Árvore Ternária - Vantagens

- adapta-se às irregularidades (desbalanceamento dos caracteres que aparecem) nas chaves de pesquisa
- não dependem da quantidade de dígitos (caracteres) possíveis
- quando a chave nao está armazenada na árvore, a quantidade de dígitos comparados tende a ser pequena (mesmo quando a chave de busca é longa)
- ela é flexivel:
  - pode ser usada para obter chaves que casam com dígitos específicos da chave de pesquisa
  - pode ser usada para obter chaves que diferem em no máximo uma posição da chave de pesquisa
- árvore Patricia oferece vantagens similares, mas comparando bits ao invés de bytes.

Busca por padrão

Busca de todas as palavras que casam com "ca.", onde "."pode ser qualquer caracter.

```
1  void casa( ApNodo p, char *k ){
2    char s[MAX];
3
4    return casaR( p, k, s, 0 );
5  }
6
7  void casaR( ApNodo p, char *k, char s[], int i ){
8    if( p == NULL ) return;
9    if((*k == '\0') && (p->d == '\0')){
10     s[i] = p->d;
11     printf( "%s\n", s );
12   }
13   if((*k == '.') || (*k == p->d)){
14     s[i] = p->d;
15     casaR( p->meio, k+1, s, i+1 );
16   }
17   if((*k == '.') || (*k < p->d))
18     casaR( p->esq, k, s, i );
19   if((*k == '.') || (*k > p->d))
20     casaR( p->dir, k, s, i );
21 }
```

Possíveis Melhorias na Árvore

- como a maioria dos nodos próximos das folhas possuem apenas um único filho, utilizar a mesma idéia das trie n-árias de manter a chave na folha no nível que a distingue das demais chaves. Isso torna a árvore *independente* do tamanho das chaves
- utilizar a idéia da árvore Patricia de manter as chaves nos nodos internos, usando os apontadores "para cima".
- usar um vetor com N elementos, um para cada dígito possível somente na raiz.
  - busca similar a um catálogo telefônico. O objetivo é diminuir a altura da árvore e o número de comparações em uma busca.
    - conjunto de chaves = 1 bilhão de chaves
    - conjunto de digitos = 256
    - raiz com vetor de $256^2 = 65.536$
    - número de comparações de bytes é em torno de 10

Referências

- Livro Sedgewick Cap. 15 (Seção 15.4)