Pesquisa Digital

- Pesquisa digital é baseada na representação das chaves como uma seqüência de caracteres ou de dígitos.
- Os métodos de pesquisa digital são particularmente vantajosos quando as chaves são grandes e de tamanho variável.
- Estruturas geralmente utilizadas para a pesquisa digital são as árvores digitais
- Exemplos:
  - Trie
  - Patrícia

Trie

- Uma trie é uma árvore M -ária cujos nós são vetores de M componentes com campos correspondentes aos dígitos ou caracteres que formam as chaves.

- Cada nó no nível i representa o conjunto de todas as chaves que começam com a mesma seqüência de i dígitos ou caracteres.

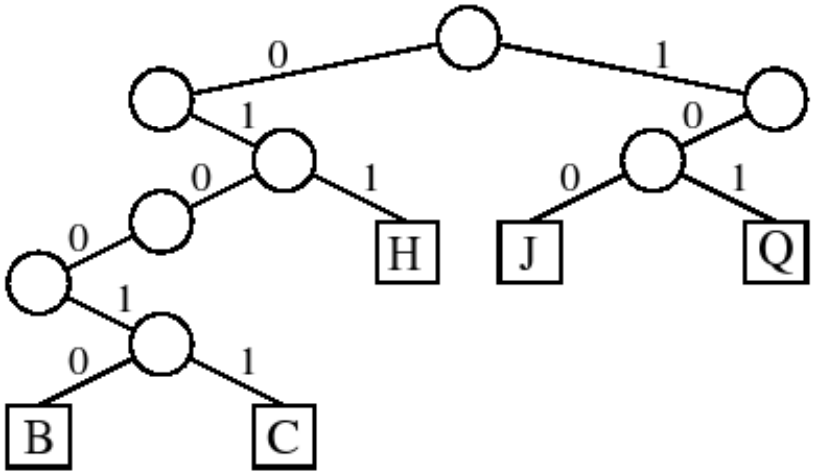
- Este nó especifica uma ramificação com M caminhos dependendo do (i + 1)-ésimo dígito ou caractere de uma chave.

Trie

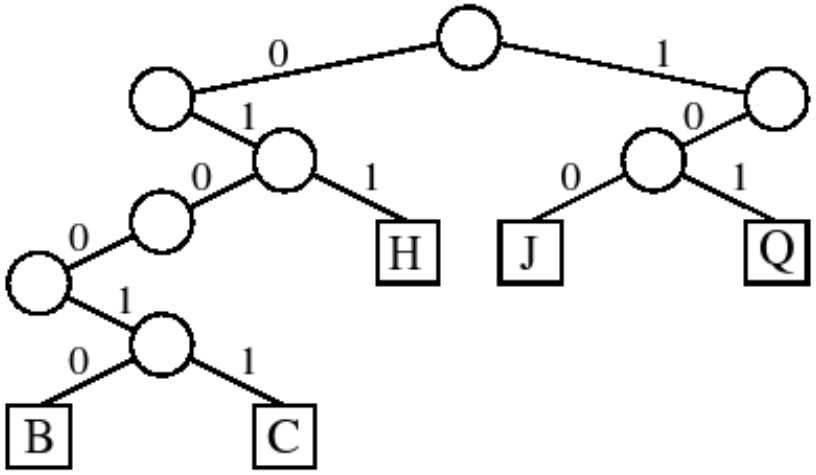
- **Considerando as chaves como seqüência de bits (isto é, M = 2), o algoritmo de pesquisa digital é semelhante ao de pesquisa em árvore, exceto que, em vez de se caminhar na árvore de acordo com o resultado de comparação entre chaves, caminha-se de acordo com os bits de chave.**

Exemplo

- Dada as chaves de 6 bits:
  - B = 010010
  - C = 010011
  - H = 011000
  - J = 100001
  - M = 101000

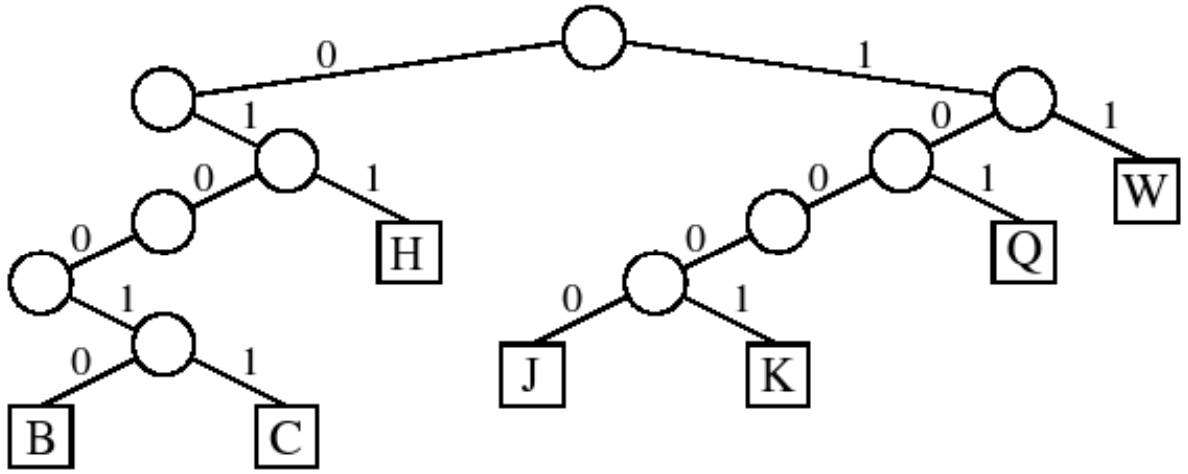


Inserção das Chaves W e K na Trie Binária



- Faz-se uma pesquisa na árvore com a chave a ser inserida. Se o nó externo em que a pesquisa terminar for vazio, cria-se um novo nó externo nesse ponto contendo a nova chave, exemplo: a inserção da chave W = 110110.
- Se o nó externo contiver uma chave cria-se um ou mais nós internos cujos descendentes conterão a chave já existente e a nova chave. exemplo: inserção da chave K = 100010.

Inserção das Chaves W e K na Trie Binária



Considerações Importantes sobre as Tries

- O formato das tries, diferentemente das árvores binárias comuns, não depende da ordem em que as chaves são inseridas e sim da estrutura das chaves através da distribuição de seus bits.

- Desvantagem:
  - Uma grande desvantagem das tries é a formação de caminhos de uma só direção para chaves com um grande número de bits em comum.
  - **Exemplo**: Se duas chaves diferirem somente no último bit, elas formarão um caminho cujo comprimento é igual ao tamanho delas, nao importando quantas chaves existem na árvore.
  - Caminho gerado pelas chaves B e C.

Patricia – Practical Algorithm To Retrieve Information Coded in Alphanumeric

- Criado por Morrison D. R. 1968 para aplicacao em recuperacao de informacao em arquivos de grande porte.
- Knuth D. E. 1973 : novo tratamento algoritmo
- Reapresentou-o de forma mais clara como um caso particular de pesquisa digital, essencialmente, um caso de arvore trie binaria
- Sedgewick R. 1988 apresentou novos algoritmos de pesquisa e de insercao baseados nos algoritmos propostos por Knuth
- Gonnet, G. H. e Baeza-Yates R. 1991 propuseram tambem outros algoritmos

Mais sobre Patricia

- O algoritmo para construção da árvore Patricia é baseado no método de pesquisa digital, mas sem apresentar o inconveniente citado para o caso das tries.
- O problema de caminhos de uma só direção é eliminado por meio de uma solução simples e elegante: cada nó interno da árvore contém o índice do bit a ser testado para decidir qual ramo tomar.
- Exemplo: dadas as chaves de 6 bits:

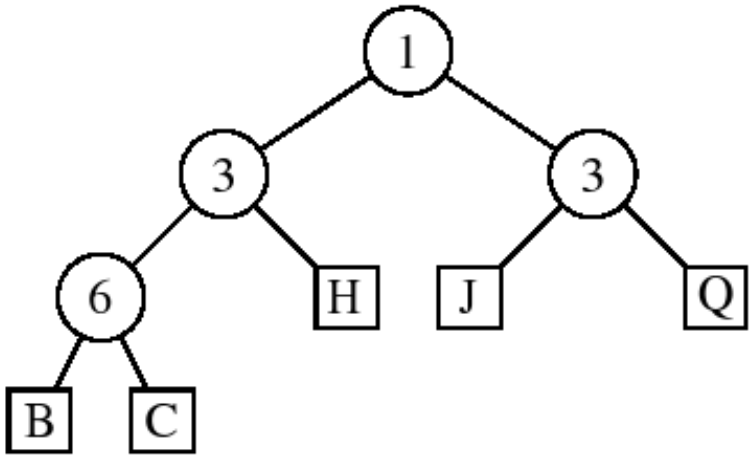
  B = 010010

  C = 010011

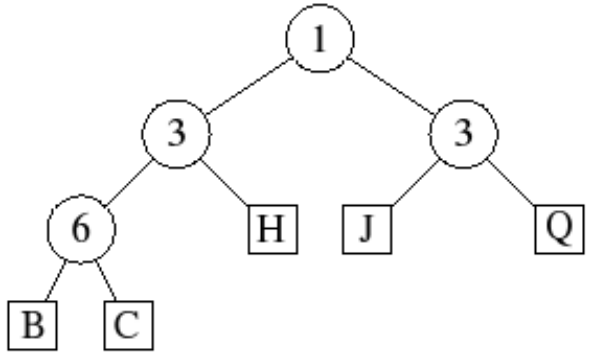
  H = 011000

  J = 100001

  Q = 101000



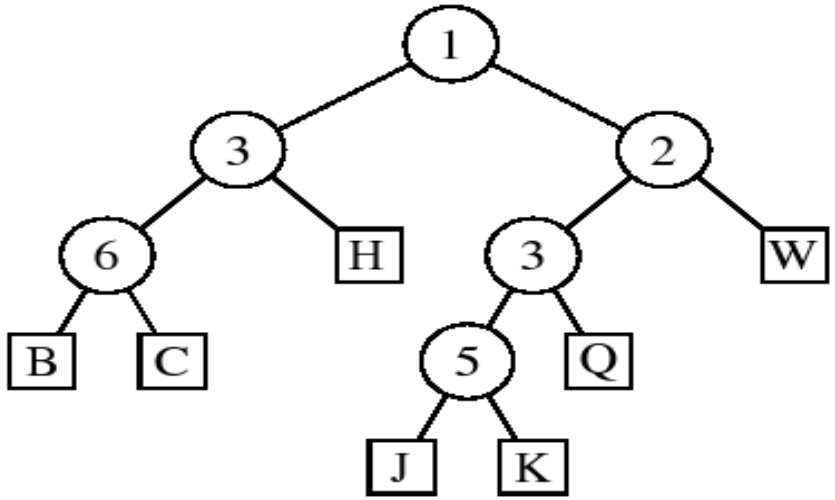
Inserção da Chave K



- Para inserir a chave K = 100010 na árvore acima, a pesquisa inicia pela raiz e termina quando se chega ao nó externo contendo J.
- Os índices dos bits nas chaves estão ordenados da esquerda para a direita.
  - Bit de índice 1 de K é 1 → a subárvore direita
  - Bit de índice 3 → subárvore esquerda que neste caso é um nó externo.
- Chaves J e K mantêm o padrão de bits 1x0xxx, assim como qualquer outra chave que seguir este caminho de pesquisa.

Inserção da Chave W

- A inserção da chave W = 110110 ilustra um outro aspecto.
- Os bits das chaves K e W são comparados a partir do 1o para determinar em qual índice eles diferem, sendo, neste caso, o de índice 2.
- **Portanto**: o ponto de inserção agora será no caminho de pesquisa entre os nós internos de índice 1 e 3.
- Cria-se aí um novo nó interno de índice 2, cujo descendente direito é um nó externo contendo W e cujo descendente esquerdo é a subárvore de raiz de índice 3.



Estrutura de Dados

```
#define D       8   /* depende de
                     ChaveTipo */

typedef unsigned char ChaveTipo;

typedef unsigned char IndexAmp;

typedef unsigned char Dib;

typedef enum {
  Interno, Externo
} NoTipo;

typedef struct PatNo* Arvore;
```

```
typedef struct PatNo {
  NoTipo nt;
  union {
        struct {
              IndexAmp Index;
              Arvore Esq, Dir;
        } NInterno ;
        ChaveTipo Chave;
    } NO;
} PatNo;
```

Funções Auxiliares

```
Dib Bit(IndexAmp i, ChaveTipo k)
{ /* Retorna o i-esimo bit da chave k a partir da esquerda */
  int  c, j;
  if (i == 0)  return 0;
  else  {
      c = k;
      for (j = 1; j <= D - i; j++)  c /= 2;
      return (c & 1);
  }
}

short EExterno(Arvore p)
{ /* Verifica se p^ e nodo externo */
  return (p->nt == Externo);
}
```

Procedimento para Criar os Nos

```
Arvore CriaNoInt(int i, Arvore *Esq,
                        Arvore *Dir)
{
   Arvore p;
   p= (Arvore)malloc(sizeof(PatNo));
  p->nt = Interno;
  p->NO.NInterno.Esq = *Esq;
  p->NO.NInterno.Dir = *Dir;
  p->NO.NInterno.Index = i;
  return p;
}
```

```
Arvore CriaNoExt(ChaveTipo k)
{
    Arvore p;
     p =(Arvore)malloc(sizeof(PatNo));
     p->nt = Externo;
     p->NO.Chave = k;
     return p;
}
```

Algoritmo de pesquisa

```
void Pesquisa(ChaveTipo k, Arvore t)
{
    if (EExterno(t))  {
        if (k == t->NO.Chave) printf("Elemento encontrado\n");
        else printf("Elemento nao encontrado\n");
        return;
    }

    if (Bit(t->NO.NInterno.Index, k) == 0)
                Pesquisa(k, t->NO.NInterno.Esq);

    else Pesquisa(k, t->NO.NInterno.Dir);
}
```

Descrição Informal do Algoritmo de Inserção

- Cada chave k é inserida de acordo com os passos, partindo da raiz:

1 - Se a subárvore corrente for vazia, então é criado um nó externo contendo a chave k (isto ocorre somente na inserção da primeira chave) e o algoritmo termina.

Descrição Informal do Algoritmo de Inserção

2- Se a subárvore corrente for simplesmente um nó externo, os bits da chave k são comparados, a partir do bit de índice imediatamente após o último índice da seqüência de índices consecutivos do caminho de pesquisa, com os bits correspondentes da chave k' deste nó externo até encontrar um índice i cujos bits difiram. A comparação dos bits a partir do último índice consecutivo melhora consideravelmente o desempenho do algoritmo. Se todos forem iguais, a chave já se encontra na árvore e o algoritmo termina; senão, vai-se para o Passo 4.

3 - Se a raiz da subárvore corrente for um nó interno, vai-se para a subárvore indicada pelo bit da chave k de índice dado pelo nó corrente, de forma recursiva.

Algoritmo de inserção

4 - Depois são criados um nó interno e um nó externo: o primeiro contendo o índice i e o segundo, a chave k. A seguir, o nó interno é ligado ao externo pelo apontador de subárvore esquerda ou direita, dependendo se o bit de índice i da chave k seja 0 ou 1, respectivamente.

5- O caminho de inserção é percorrido novamente de baixo para cima, subindo com o par de nós criados no Passo 4 até chegar a um nó interno cujo índice seja menor que o índice i determinado no Passo 2. Este é o ponto de inserção e o par de nós é inserido.

Algoritmo de inserção

```
Arvore InsereEntre(ChaveTipo k, Arvore *t, int i)
{
  Arvore p;
  if ((EExterno(*t)) ||
       (i < (*t)->NO.NInterno.Index))   { /* cria um novo no externo */
      p = CriaNoExt(k);
      if (Bit(i, k) == 1)   return (CriaNoInt(i, t, &p));
      else   return (CriaNoInt(i, &p, t));
  }
  else  {
      if (Bit((*t)->NO.NInterno.Index, k) == 1)
              (*t)->NO.NInterno.Dir = InsereEntre(k, &(*t)->NO.NInterno.Dir, i);
      else  (*t)->NO.NInterno.Esq = InsereEntre(k, &(*t)->NO.NInterno.Esq, i);
      return (*t);
    }
}
```

Algoritmo de inserção

```
Arvore Insere(ChaveTipo k, Arvore *t)
{ Arvore p;       int i;
    if (*t == NULL)   return (CriaNoExt(k));
    else  {
        p = *t;
        while (!EExterno(p))  {
            if (Bit(p->NO.NInterno.Index, k) == 1)  p = p->NO.NInterno.Dir;
           else   p = p->NO.NInterno.Esq;
        }
        /* acha o primeiro bit diferente */
        i = 1;
        while ((i <= D) & (Bit((int)i, k) == Bit((int)i, p->NO.Chave))) i++;
         if (i > D)    {
                printf("Erro: chave ja esta na arvore\n");       return (*t);
         }
         else  return (InsereEntre(k, t, i));
   }
}
```