Universidade Federal de Goiás
Goiânia 08 de Junho de 2018
Edson Júnior Frota Silva Matrícula: 201515412
Engenharia de Computação Prof°(a): Renata Dutra
Estrutura de Dados II

Atividade de Árvores de Pesquisa

Pesquisa em Memória Primária
- Árvores AVL, página 143
- Árvores Patrícia, página 129

Pesquisa em Memória Secundária:
- B-Trees, página 170

- Breve resumo sobre cada árvore. Atividade em dupla. Postagem até o dia 11/06/2018
- Funcionamento.
- Desvantagens e desvantagens
- Implementação do Algoritmo (inserção e busca, pelo menos) (independente de linguagem de programação)

Árvore AVL: Podemos dizer que uma árvores AVL(Adelson Velskye Landis) é uma árvore de busca binária balanceada com relação a altura de suas subárvores, ou seja em uma árvore AVL nós verificamos a altura da árvore da esquerda e da direita de cada nó e garantindo que a diferença seja no máximo +-1. Em uma árvore avl o fator de balanceamento e calculado a cada nó, e é importante ressaltar que a altura de uma árvore vazia é -1.
Se tratando de AVL entram alguns fatores que precisama ser discutidos que são as rotações. Temos 2 tipos de rotação, a rotação a esquerda e a rotação a direita, elas servem para manter o fator balanceamento da nossa árvore. Em alguns casos, temos que reordenar os elementos da nossa estrutura para garantir que ela se encaixe nos termos para ser uma árvore do tipo AVL.

Vantagens:
- Inserção, remoção e pesquisa são O(log N), já que as árvores AVL são sempre balanceadas na altura.
- O balanceamento da altura adiciona apenas um fator constante à velocidade da inserção.

Desvantagens:
- Difícil de programar e depurar; mais espaço parao fator de balanceamento;
- O balanceamento tem um custo;
- Buscas em bases de dados maiores utilizam outras estruturas como Árvores B ou B+;

Algoritmo de inserção em uma árvore do tipo AVL:

```
1  se estiver desbalanceado para a direita  bal(a) < 1
```

```
 2          f = direita(a)
 3          se bal(direita(a)) == 1
 4             rotaçãoEsquerda(a)
 5             bal(a) = bal(f) = 0
 6          senão
 7             neto = esquerda(f)
 8             rotaçãoDireita(direita(a))
 9             rotaçãoEsquerda(a)
10             se  bal(neto) == 0
11               bal(a) = bal(f) = 0
12             senão se bal(neto) > 0
13               bal(a)= 0
14               bal(f) = 1;
15             senão
16               bal(a)  = 1;
17               bal(f) = 0;
18          bal(neto) = 0;
19        se estiver desbalanceado para a esquerda  bal(a) > 1
20          f = esquerda(a)
21          se bal(f) == 1
22             rotaçãoDireita(a)
23             bal(a) = bal(f) = 0
24          senão
25             neto = direita(f)
26             rotaçãoEsquerda(esquerda(a))
27             rotaçãoDireita(a)
28             se bal(n) == 0
29               bal(a) = bal(f) = 0
30             senão se bal(neto) > 0
31               bal(a) = 1;
32               bal(f) = 0;
33             senão
34               bal(a) = 0;
35               bal(f) = 1;
36             bal(neto) = 0
37  Apontador Insere(Registro r, Apontador p, int *mudouAltura){
38    if( p == nodoNull ){
39      *mudouAltura = TRUE;
40      return criaNo( r, nodoNull, nodoNull );
41    }
42    if( r.Chave <= p>Reg.Chave ){
43      p>Esq = Insere( r, p>Esq, mudouAltura );
44      if( *mudouAltura ){
45        p>bal++;
46        if( p>bal == 0 )
47          *mudouAltura = FALSE;
48        else if( p>bal == 2){
49          *mudouAltura = FALSE;
50          balanceia( &p );
51        }
52      }
53    } else {
54      p>Dir = Insere( r, p>Dir, mudouAltura );
```

```
55      if( *mudouAltura ){
56        p>bal;
57        if( p>bal == 0 )
58          *mudouAltura = FALSE;
59        else if( p>bal == 2){
60          *mudouAltura = FALSE;
61          balanceia( &p );
62        }
63      }
64    }
65    return p;
66  }
```

Exemplo de Busca em uma árvore do tipo AVL:

```
 1  busca_AVL(@pt_u:^no_AVL, K:inteiro):logico;
2  inicio
3  se  pt_u é NULO então retornar Falso;
4  se K = pt_u->chave então retornar Verdadeiro;
5  senão se K < pt_u->chave então
6           retornar busca_AVL(K, u->esq);
7        senão retornar busca_AVL(K, u->dir);
8  fim.
```

Árvore Patricia: Por definição a árvore patricia (Practical Algorithm To Retrieve Information Coded in Alphanumeric) deve ser uma árvore estritamente binária todos os seus nós, exceto as folas tem apenas 2 descendentes. Sua principal função é facilitar a busca dando apenas um caminho para comparações.

Vantagens:

- Permite armazenar um número de posições para qual é movido para fentre antes de fazer a próxima comparação.
  [elimina comparações desnecessárias → melhora o desempenho]

Desvantagens:

- Considera apenas 2 subárvores. Se mais do que duas chaves são distintas na mesma posição do caratere, é necessário adicionar nós extras ao índice para separa-lo. [Se n chaves são distintas na mesma posição, então serão necessários n-1 nós para separá-los. Se muitos casos destes acontecem, é preferível utilizar TRIE].

Algoritmo de inserção em uma árvore do tipo patricia:

```
1  insere( raiz, reg )
2    k = reg.chave
3    p = buscaR( raiz>esq, k, 1 )
4    pk = p>chave
```

```
 5     se k == pk retorna;
 6     i = 0;
 7
 8     enquanto digito(k, i)==digito(pk, i)
 9       i++;
10     raiz>esq = insereR (raiz>esq, reg, i, raiz)
11   insereR( p, reg, bit, paiP )
12     se p>bit >= bit ou p>bit <= paiP>bit
13         n = criaNodo( reg, bit )
14         se digito(reg.chave, bit) == 0
15            n>esq = n;  n>dir = p;
16         senão
17            n>esq = p;  n>dir = n;
18         retorna n
19     se digito(reg.chave, bit) == 0
20        p>esq = insereR( p>esq, reg, p>bit, p )
21     senão
22        p>dir = insereR( p>dir, reg, p>bit, p )
23     retorna p
```

Algoritmo de busca em uma árvore do tipo patricia:

```
 1   busca(raiz, k)
 2     p = buscaR( raiz>esq, k, 1 );
 3     se p>reg.Chave == k retorna p
 4     senão retorna nodoNulo
 5   buscaR( p, k, bit )
 6     se p>bit <= bit retorna p;
 7     se digito( k, p>bit) == 0
 8        retorna buscaR( p>esq, k, p>bit )
 9     senão
10        retorna buscaR( p>dir, k, p>bit )
```

B Tree

Árvore B: É uma árvore muito parecida com a árvore binária, a diferença está que ao invés de ter apenas 2 filhos, a árvore B pode ter vários filhos. Cada filho desta árvore está associado a uma chave, neste tipo de árvore os nós também são chamados de páginas por que quando surgiu esse tipo de árvore cada bloco de chave era preciso ser armazenado em paginação.
Existe uma regra que toda página precisa pelo menos de 50% de ocupação exceto a raiz, para que a estrtura seja eficiente.

Algoritmo de representação de uma árvore do tipo B:

```
1   const t = 2;
2   typedef struct no_arvoreB arvoreB;
3   struct no_arvoreB {
4      int num_chaves;
5      char chaves[2*t1];
6      arvoreB *filhos[2*t];
7      bool folha;
8   };
```

Algoritmo de busca em uma árvore do tipo B:

```
 1  Busca (x, k)
 2    i = 0
 3    enquanto (i <= num[x] e k > chave_i[x])
 4      i = i+1
 5    se i <= num[x] e k = chave_i[x]
 6      retorna (x, i)
 7    se folha[x]
 8      retorna NIL
 9    else
10      le_disco(p_i[x])
11      retorna Busca(p_i[x], k)
```

Cada árvore apresentada tem uma grande utilização no campo da ciência da computação, sua criação proporcionaram ao homem uma grade eficiência em relação a econômia tempo. A partir disso podemos buscar informações em um banco de dados que contém um grande número de informações de forma bem rápida e prática, basta escolher o método correto, o que será mais útil para seu problema, podemos dizer que não mudaram a vida apenas dos cientistas da computação, mas de todas as pessoas de forma geral.

UFG
Edson Júnior Frota Silva
Engenharia de Computação