Resumo ED2

ÁRVORE BINÁRIA DE BUSCA

Nome | Título do curso | Data

Árvore de Busca Binária

A estrutura de árvores de busca suporta diversas operações de conjuntos dinâmicos, como SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESOR, INSERT, DELETE. Assim, podemos utilizar essa estrutura como um dicionário ou também como uma fila de prioridades.

Essas operações básicas possuem um tempo proporcional à altura da árvore, por exemplo: Uma árvore completa com n nós vai ter complexidade O(lg n), porém se a árvore for uma cadeia linear de n nós, as mesmas operações demoram o tempo de O(n) no pior caso.

A altura esperada de uma árvore construída aleatoriamente é de O(lg n), dessa forma as operações básicas de conjuntos dinâmicos em tal árvore demoram o tempo de O(lg n) em média.

- Precisa de um título? Na guia Página Inicial, na galeria Estilos, clique no estilo de título desejado.
- Observe também os outros estilos na galeria, como o de citação, de lista numerada ou de lista com marcadores, como esta.
- Para obter resultados melhores ao selecionar um texto para copiar ou editar, não inclua espaços à esquerda ou à direita dos caracteres em sua seleção.

O QUE É:

Uma árvore de busca binária é organizada em uma árvore binária. Podemos representar essa estrutura através de uma estrutura de dados ligada, no qual cada nó é um objeto. Além de uma chave e dados, cada nó conteria os atributos esquerda, direita e p, que apontam para os nós correspondentes ao filho da esquerda, filho da direita e seu pai.

Se um filho ou pai estiver ausente, o atributo recebe valor nulo. O único nó que possui pai nulo na árvore é o nó raiz.

As chaves na árvore são armazenadas de modo a satisfazer a propriedade de árvore de busca binária em que: seja x um nó em uma árvore de busca binária. Se y é um nó na subárvore esquerda de x, então a chave de y deve ser menor ou igual que a chave de x, caso contrário se y for um nó na direita da chave x, então a chave de y deve ser maior que a chave de x.

Podemos printar a árvore em sequência ordenada através do algoritmo denominado percurso in-order. Esse algoritmo imprime a chave da raiz de uma subárvore entre a impressão dos valores de sua subárvore esquerda e a impressão dos valores da subárvore direita. Dessa forma podemos analisar também o algoritmo de percurso pre-order, em que a raiz será impressa antes dos valores das sub-árvores e o de pós-order onde a raiz será impressa depois dos valores das sub-árvores.

Percorrer uma árvore demora O(n), pois o procedimento chama a si mesmo recursivamente duas vezes para cada nó, uma vez para o filho esquerda e outra para o filho direita.

Prova:

Se x é a raiz de uma subárvore de n nós, então a chamada da função demora o tempo O(n) porque:

Seja T(n) o tempo tomado pelo percurso inorder chamado na raiz de uma subárvore de n nós. Visto que inorder visita todos os n nós da subárvore, temos T(n) = ômega(n).

Uma vez que o percurso demora um tempo constante e pequeno em uma subárvore vazia, temos T(0) = c, em que C é uma constante positiva maior que zero.

Para n > 0, suponha que o percurso seja chamado em um nó x cuja subárvore esquerda tem k nós e cuja subárvore direita n – k – 1 nós. O tempo para executar a operação é limitado por T(n) = T(k) + T(n – k – 1) + d para alguma constante d > 0, que reflete um limite superior para o tempo de execução do corpo da nossa função.

Usando o método da substituição temos que T(n) = O(n), portanto provamos que T(n) <= (c + d) n + c. Para n = 0, temos (c + d) * 0 + c = c = T(0). E para n > 0 temos, ao final, (c + d)n + c.

Exercícios:

Qual a diferença entre a propriedade de árvore de busca binária e a propriedade de heap mínimo? A propriedade de heap mínimo pode ser usada para imprimir as chaves de uma árvore de n nós em sequência ordenada no tempo O(n)? Justifique sua resposta.

R: Ordenação: Na árvore de busca binária, os elementos são armazenados de forma que o nó filho da esquerda de um nó pai é sempre menor ou igual ao nó pai, e que o nó da direita será sempre maior que o nó pai. No heap de mínimo, os elementos são armazenados de forma que o no pai é sempre menor ou igual que os nós filhos.

Tempo de busca por um elemento: Na árvore binária nosso tempo será log(n) no caso médio e O(n) no pior caso, no heap de mínimo a busca terá um tempo de O(n)

Inserir e deletar: Na árvore inserir e deletar elementos demora O(log n) no caso médio e O(n) no pior caso, já no heap as duas operações demorariam O(log n).

Uso de memória: A árvore requer mais memória já que cada nó guardará dois ponteiros enquanto o heap mínimo usaria menos por armazenar os elementos em um Array, onde um pai teria apenas dois filhos.

Utilidade: As árvores são úteis para buscar elementos e encontrar os elementos mais próximos a ele enquanto o heap mínimo por exemplo, é usado para implementar filas de prioridade, onde elementos com uma prioridade maior são extraídos primeiro.

Você não poderia usar um heap mínimo para printar uma árvore de tamanho n em ordem com uma complexidade O(n), pois para imprimir os elementos você precisaria extrair o elemento mínimo, adicionar a uma nova estrutura e repetir o processo até extrair todos os elementos, logo teríamos uma complexidade O(n log n).

CONSULTAS NA ÁRVORE:

As buscas em árvore tem complexidade O(h), onde h é a altura da árvore.

Busca:

Começamos na raiz e traçamos um caminho simples descendo a árvore, para cada nó x encontrado, comparamos a nossa chave desejada com a chave de x, se as duas chaves são iguais encerramos a nossa busca, se nossa chave for menor continuamos a busca para a esquerda, se for maior continuamos a busca para a direita.

Podemos encontrar o elemento mínimo da árvore percorrendo a esquerda dela até encontrarmos o elemento nulo e o inverso para encontar o máximo. Ambas as buscas possuem O(h).

O sucessor de um nó X é o nó com a menor chave maior que a chave[x]. O predecessor é simétrico.

Temos um problema com as árvores, a inserção de elementos com a mesma chave na estrutura.

Nesse caso temos dois caminhos, permitir chaves duplicadas ou permitir somente chaves únicas.

Com chaves duplicadas, simplesmente iriamos inserir todos os elementos com chaves iguais como nós separados ou como valores adicionais associados a um nó.

Com chaves únicas somente o primeiro item com aquela chave será inserido, e qualquer item subsequente com a mesma chave será rejeitado.

Permitir chaves duplicadas é um grande problema já que teríamos uma árvore maior e possivelmente desbalanceada resultando em complexidades maiores e pior desempenho, pelo outro lado, utilizar somente chaves únicas pode dificultar o armazenamento e gerenciamento de valores com a mesma chave.

As duas operações de inserção terão complexidade O(log n) no caso médio e O(n) no pior caso. Mas no final das contas permitir chaves duplicadas poderá afetar a performance da árvore no geral.

CONSIDERAÇÕES FINAIS:

Árvore de busca binária é uma estrutura de dados bem importante já que nos permite implementar dicionários e filas de prioridades, além de outras funções importantes. Seus principais conceitos são o nó, um objeto que contém um valor e ponteiros para seu filho esquerdo, direito e pai. Ordenação: Os elementos de uma árvore de busca binária são ordenados de forma que seus filhos menores ficam na esquerda e seus filhos maiores ficam na direita.

O tempo de busca em uma árvore é de O(log n) no caso médio e O(n) no pior caso.

A inserção e remoção de itens em uma árvore de busca binária é igual ao tempo de busca.

Podemos percorrer a lista de forma in-order, pre-order e post-order.

Temos também a altura, que é o número de arestas do nó raiz até um nó folha. Em uma árvore balanceada a altura é O(log n_nós).