MAT A40 - Estrutura de Dados e Algoritmos I [1]

Dr. George Lima
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade Federal da Bahia

[1] Partes deste módulo foram baseadas em conteúdos presentes em "A. V. Aho, J. D. Ullman, *Foundations of Computer Science* (C edition).

Módulo 4

Árvores e árvores binárias de busca

O tipo Árvore

Definição

Uma árvore é um conjunto de pontos e linhas, chamados nós e arestas, respectivamente, que satisfaz as seguintes condições:

1. Apenas um dos nós é identificado como raiz da árvore.
2. Qualquer nó f não-raiz da árvore é conectado a outro nó p, este chamado de pai de f. O nó f é chamado de filho do nó p.
3. Para cada nó não-folha f, há apenas um nó pai, mas um nó pai pode possuir mais de um nó filho.
4. Numa árvore de raiz r, a partir de qualquer um de seus nós f, é possível alcançar r indo de f ao seu pai, deste ao pai do pai de f e assim sucessivamente. Não há pai do nó r.

O tipo Árvore

Definição recursiva (cada filho de um nó é uma subárvore)

1. Um nó é uma árvore.
2. Se T é uma árvore, nenhum nó aparece em T mais de uma vez.
3. Suponha que $T_1, T_2, \ldots, T_k$ são árvores com respectivas raizes $f_1, f_2, \ldots, f_k$ e seja r um novo nó. Uma nova árvore T pode ser construída tendo r como raiz após as seguintes operações:
   3.1 Faça r a raíz de T.
   3.2 Adicione uma aresta de r a cada um dos nós $f_1, f_2, \ldots, f_k$.

Caminho numa árvore

- A sequência $f_1, f_2, \ldots, f_k$ de nós numa árvore T tal que existe arestas ligando f_1 a f_2, f_2 a f_3, e assim sucessivamente, é um caminho em T e seu tamanho (ou comprimento) é $k-1$. Um caminho de um único nó tem comprimento nulo.
- Numa árvore T, a altura de um nó f é o maior caminho de f até um nó folha. A altura de T é a altura de seu nó raiz.
- O nível de um nó f é o comprimento do caminho da sua raiz até f.
- Num caminho $f_1, f_2, \ldots, f_k$, f_i é ancestral de f_j e f_j descendente de f_i se o nível de f_i for menor que o nível de f_j, $j > i$.
- Uma folha é um nó de uma árvore com nenhum descendente. Um nó com descendentes é chamado de interno à árvore.

Estrutura de dados – Árvore

Várias são as maneiras para se representar uma árvore, cada uma se adequando melhor a algum tipo de aplicação.

1. Representação n-ária em cada nó usando vetor de ponteiros. Permite acesso aos filhos do nó em $O(1)$, mas gasta espaço caso haja poucos filhos por nó. Requer que o número máximos de filhos para cada nó seja conhecido.
2. Representação filho à esquerda, irmão à direita. Mais eficiente em espaço, porém o acesso ao k-ésimo filho é em tempo $O(k)$. Pode-se incluir mais filhos por nó em tempo de execução.
3. Manter ponteiro para o nó pai. Útil quando precisa-se de um caminho rápido em direção à raiz.
4. A árvore é representada como um vetor de nós. Ineficiente na maioria dos casos. Em situações especiais, pode ser uma representação compacta e eficiente.

Exemplos e aplicação de árvore

1. TRIE
2. Árvore B
3. Diretório de arquivos
4. Expressões aritméticas
5. e muitas outras...

Árvore binária

Definição e caracterização

Uma árvore cujos nós tem cada um até dois filhos. Além disso, para árvores binárias, para cada um dos seus nós, distingui-se a sub-árvore direita da sub-árvore esquerda. Por exemplo, uma árvore com raiz r tendo como filhos esquerdo e direito os nós f_1 e f_2, respectivamente, é distinta daquela com raiz r e com filhos esquerdo e direito sendo f_2 e f_1, respectivamente.

Observações:

1. É conveniente definir uma árvore vazia, com nenhum nó.
2. Todas as definições feitas para árvore (caminho, subárvore, altura etc) se aplicam à árvore binária.

Árvore binária – TAD

```c
typedef struct node node_t;
struct node {
  int element;
  node_t *left, *right;
};
```

Observações:

- Operações de busca, inserção e remoção em árvore binária são comumente realizadas em *árvores binárias de busca* – a serem abordadas em breve.
- Operações de busca baseadas em varredura da árvore serão tratadas a seguir e não requerem árvore binária de busca.
- Consultas à altura e nível de um nó independem do tipo da árvore.

Exercício:

1. Mostre que numa árvore binária com n nós e com N ponteiros `*left` e `*right` iguais a `NULL`, $N = 1 + n$.

Varredura em árvore binária

- Varredura (ou busca) em profundidade – *Depth-First Search* (DFS). Requer uso de pilha (explícita ou implicitamente).
  - Pré-ordem. Nó → filho à esquerda → filho à direita.
  - Em-ordem. Filho à esquerda → nó → filho à direita.
  - Pós-ordem. Filho à esquerda → filho à direita → nó.
- Varredura (ou busca) em largura – *Breadth-First Search* (BFS). Requer uso de fila.
  - Insere o nó raiz na fila.
  - Enquanto a fila não estiver vazia
    - Retira um nó da fila.
    - Insere seus filhos (se há algum) na fila.
    - Processa o elemento retirado da fila.

DFS

```
void in_order_print(node_t *r) {
  if (r != NULL) {
      in_order_print(r -> left);
      printf("%d \t",r -> element);
      in_order_print(r -> right);
    }
}
void post_order_print(node_t *r) {
  if (r != NULL) {
      post_order_print(r -> left);
      post_order_print(r -> right);
      printf("%d \t",r -> element);
    }
}
void pre_order_print(node_t *r) {
  if (r != NULL) {
    printf("%d \t",r -> element);
    pre_order_print(r -> left);
    pre_order_print(r -> right);
  }
}
```

Exercícios

- Mostre o código em C para realizar BFS numa árvore binária e forneça a complexidade do mesmo.
- Utilizando pilha como estrutura auxiliar, construa versões não recursivas das funções `in_order_print`, `post_order_print` e `pre_order_print`. Estas versões possuem a mesma complexidade que as correspondentes versões recursivas?
- Escreva uma função em C que recebe um endereço de um nó x de uma árvore binária (não necessariamente sua raiz) e devolve o nó sucessor de x considerando uma varredura em-ordem.
- Construa uma função em C que retorna a altura de um nó numa árvore. A função deve receber o endereço do nó para o qual deseja-se determinar a altura. Considere que uma árvore vazia possui altura -1.

BFS

Resposta de exercício proposto:

```c
void bfs_print(node_t *r) {

 if (r == NULL) return;

  queue_t *q = newQueue(); // a queue for tree nodes

  enqueue(r,q);
  while(!isempty(q)) {
    r = dequeue(q);
    if (r -> left)  enqueue(r -> left,q);
    if (r -> right) enqueue(r -> right,q);
    printf("\n %d", r -> element);
  }
  delete(q);
}
```

- Qual a complexidade de tempo da função `bfs_print`?
- Qual a complexidade de espaço da função `bfs_print`?

Altura de um nó

Resposta de exercício proposto:

```
#define MAX(a,b) ((a) > (b) ? (a) : (b))

int height(node_t *r) {

  if (r)
    return 1+MAX(height(r -> left), height(r->right));
  else
    return -1;

}
```

- Qual a complexidade de tempo da função `height`?
- Qual a complexidade de espaço da função `height`?

Árvore Binária de Busca

Definição

Árvore binária de busca é uma árvore binária rotulada (e ordenada) que satisfaz a seguinte propriedade para todo nó x: todos os nós da sub-árvore esquerda de x (se há algum) são menores ou iguais a x; e todos os nós da sub-árvore direita de x (se há algum) são maiores que x.

Observações:

- Árvores binárias de busca oferecem uma maneira eficiente de organizar informações hierarquicamente.
- Algoritmos para inclusão, remoção de informação na árvore devem preservar a propriedade de ordem da árvore.

Busca numa Árvore Binária de Busca

Busca de um elemento *e* numa árvore de raiz *r*

Se a árvore está vazia ou se o elemento *e* está contido na raiz, a busca termina (caso base). Caso contrário, deve-se buscar *e* na sub-árvore esquerda ou na sub-árvore direita de *r*.

Observações:

- Notar o argumento indutivo em torno da busca de um elemento na árvore. Deste, deriva-se imediatamente um algoritmo recursivo.
- Assume-se aqui que não existem rótulos iguais para nós distintos.

Busca numa Árvore Binária de Busca

Quais as diferenças (em tempo de execução e espaço) entre as duas versões abaixo?

```c
node_t *search(int e, node_t *r) {
  if (r == NULL || r -> element == e)
    return r;
  else if (r -> element < e)
    return search(e, r -> left);
  else
    return search(e, r -> right);
}
```

```c
node_t *search(int e, node_t *r)  {
  while (r != NULL && r -> element != e) {
    if (r -> element < e)
      r = r -> left;
    else
      r = r -> right;
  }
    return r;
}
```

Inserção numa Árvore Binária de Busca

Inserir um elemento e numa árvore de raiz r

- Caso base. Se a árvore está vazia, cria-se um novo nó contento e. Se a árvore de raiz r não está vazia e o nó r contém e, retorna-se sem fazer nada (ou alternativamente, indica-se um erro).
- Caso geral. Insere-se e na sub-árvore esquerda ou direita de r se e for menor que ou maior que o valor contido em r, respectivamente.

Observações:

- O argumento indutivo aparece novamente na descrição do procedimento de inserção. A inserção de fato ocorrerá numa sub-árvore vazia (filho de algum nó folha).

Inserção numa Árvore Binária de Busca: Versão Recursiva

A função `insert` desta implementação sempre retorna a raiz da árvore após a inserção.

```
node_t *insert(int e, node_t *r) {

  if (r == NULL) {
    r = newNode(); // aloca memoria para novo elemento
    r -> element = e;
  }
  else if (r -> element < e)
    r -> left = insert(e, r -> left);
  else if (r -> element > e)
    r -> right = insert(e, r -> right);
  return r;
}
```

Inserção numa Árvore Binária de Busca: Versão Iterativa

```
node_t *inserti(int e, node_t *r) {

  node_t *f, *p; // pai e filho durante o caminho

  f = r;
  p = NULL;

  /* Busca local para inserir elemento */
  while (f != NULL) {
    p = f;
    if (f -> element > e)
      f = f -> left;
    else if (f -> element < e)
      f  = f -> right;
    else // elemento estah na arvore
      return r;
  }

    /* Insere elemento */
  f = newNode();
  f -> element = e;
  if (p != NULL) {
    if (p -> element > e)
      p -> left = f;
    else if (p -> element < e)
      p -> right = f;
    return r;
  } else
      return f;
}
```

Remoção numa Árvore Binária de Busca

Remover um elemento e de uma árvore de raiz r

- Buscar o elemento e na árvore. Caso este não seja encontrado, não há nada a fazer.
- Se e está na árvore, ele pode estar num nó folha ou num nó interno.
  1. Se e está armazenado num nó folha, remove-se este nó.
  2. Se e está num nó interno x, há dois casos a considerar
     - 2.1 O nó x possui apenas um filho. Remove-se x, substituindo-o pelo filho.
     - 2.2 O nó x possui dois filhos. Busca-se o nó y na sub-árvore de raiz x que contém o sucessor (ou antecessor) de e, transfere-se y para a posição de x e remove-se x.

Observação:

- A operação de remoção não altera a propriedade de ordem da árvore binária de busca.

Remoção numa Árvore Binária de Busca

```
node_t *delete(int e, node_t *r) {
  node_t *p;
  if (r == NULL) // nao encontrado
    return r;

  if (r -> element < e) // procura pela direita
    r -> right = delete(e, r -> right);
  else if (r -> element > e) // procura pela esquerda
    r -> left = delete(e, r -> left);
  else {  // r contem elemento e
    if (r -> left == NULL) {// r sem filho esq.
      p = r;
      r = r -> right;
      free(p);
    } else if (r -> right == NULL) {// r sem filho dir.
      p = r;
      r = r -> left;
      free(p);
    } else { // r com ambos os filhos
      p = get_max(r -> left);
      r -> element = p -> element;
      r -> left = delete(p -> element, r -> left);
    }
  }
  return r;
}
```

Exercícios

1. Qual a complexidade de tempo e de espaço dos algoritmos para inserção, remoção e busca de elementos numa árvore binária de busca? As versões iterativas e recursivas destes algoritmos apresentam diferenças de complexidade? Justifique.
2. Mostre o código da função `get_max` usada na lina 16 da função `delete`.
3. Construa uma versão iterativa da função `delete` para remover um elemento de uma árvore binária de busca.
4. Modifique a função `delete` da página anterior para que, ao invés de copiar o conteúdo do nó folha `p` para o nó `r`, (linhas 17-18), desloca-se o nó `p` para substituir o nó `r`. Tal alteração pode ser recomendável quando o conteúdo a ser copiado é grande e a cópia do mesmo é lenta.