SIN5013 - Análise de Algoritmos e Estruturas de Dados

Árvores

Prof. Flávio Luiz Coutinho

Motivação

Busca binária: desempenho muito superior em relação à busca sequencial.

Motivação

Busca binária: desempenho muito superior em relação à busca sequencial.

Qual a mágica? realizar menos comparações, tirando proveito da ordenação prévia de uma sequência de valores.

Motivação

Busca binária: desempenho muito superior em relação à busca sequencial.

Qual a mágica? realizar menos comparações, tirando proveito da ordenação prévia de uma sequência de valores.

Mas para isso, é preciso que se tenha acesso aleatório a um elemento qualquer da sequência em tempo constante.

Motivação

Busca binária: desempenho muito superior em relação à busca sequencial.

Qual a mágica? realizar menos comparações, tirando proveito da ordenação prévia de uma sequência de valores.

Mas para isso, é preciso que se tenha acesso aleatório a um elemento qualquer da sequência em tempo constante. Listas sequenciais possuem esta característica, mas listas ligadas não…

Motivação

Pergunta: seria possível elaborar algum tipo de estrutura que:

- não mantenha os elementos alocados fisicamente de forma sequencial.
- seja dinâmica, ou seja, a memória é alocada sob demanda a cada inserção.
- não ofereça acesso aleatório a um elemento qualquer em tempo constante.

Motivação

Pergunta: seria possível elaborar algum tipo de estrutura que:

- não mantenha os elementos alocados fisicamente de forma sequencial.
- seja dinâmica, ou seja, a memória é alocada sob demanda a cada inserção.
- não ofereça acesso aleatório a um elemento qualquer em tempo constante.

Mas ainda assim permita uma busca eficiente, como a busca binária?

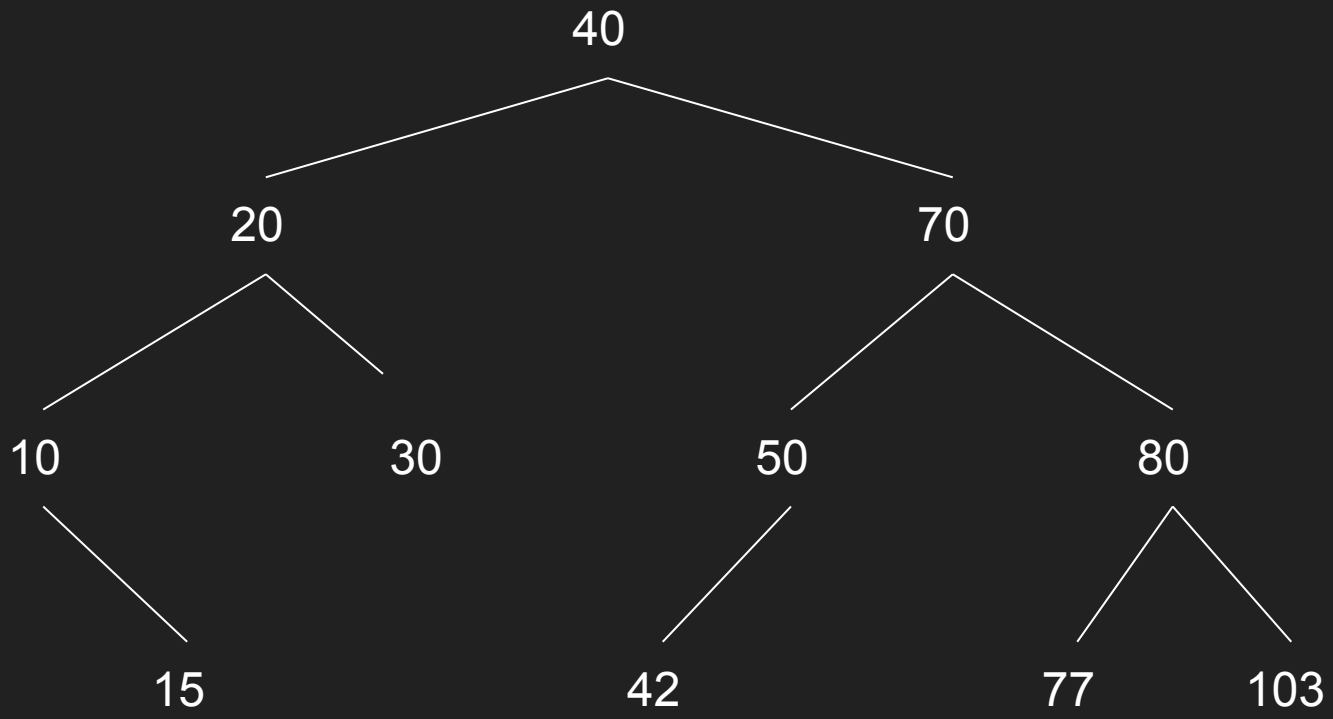
Motivação

Pergunta: seria possível elaborar algum tipo de estrutura que:

- não mantenha os elementos alocados fisicamente de forma sequencial.
- seja dinâmica, ou seja, a memória é alocada sob demanda a cada inserção.
- não ofereça acesso aleatório a um elemento qualquer em tempo constante.

Mas ainda assim permita uma busca eficiente, como a busca binária? Sim, através de uma árvore binária de busca.

Exemplo: árvore binárias de busca



Árvores

Uma árvore binária de busca já se trata de uma especialização de uma estrutura mais genérica denominada árvore.

Árvores

Uma árvore binária de busca já se trata de uma especialização de uma estrutura mais genérica denominada árvore.

Uma árvore é uma estrutura em que:

- há um conjunto de nós (cada um armazenando algum tipo de informação).
- existe um nó denominado “nó raiz”.
- existe uma ou mais sub árvores “penduradas” no nó raiz.

Árvores

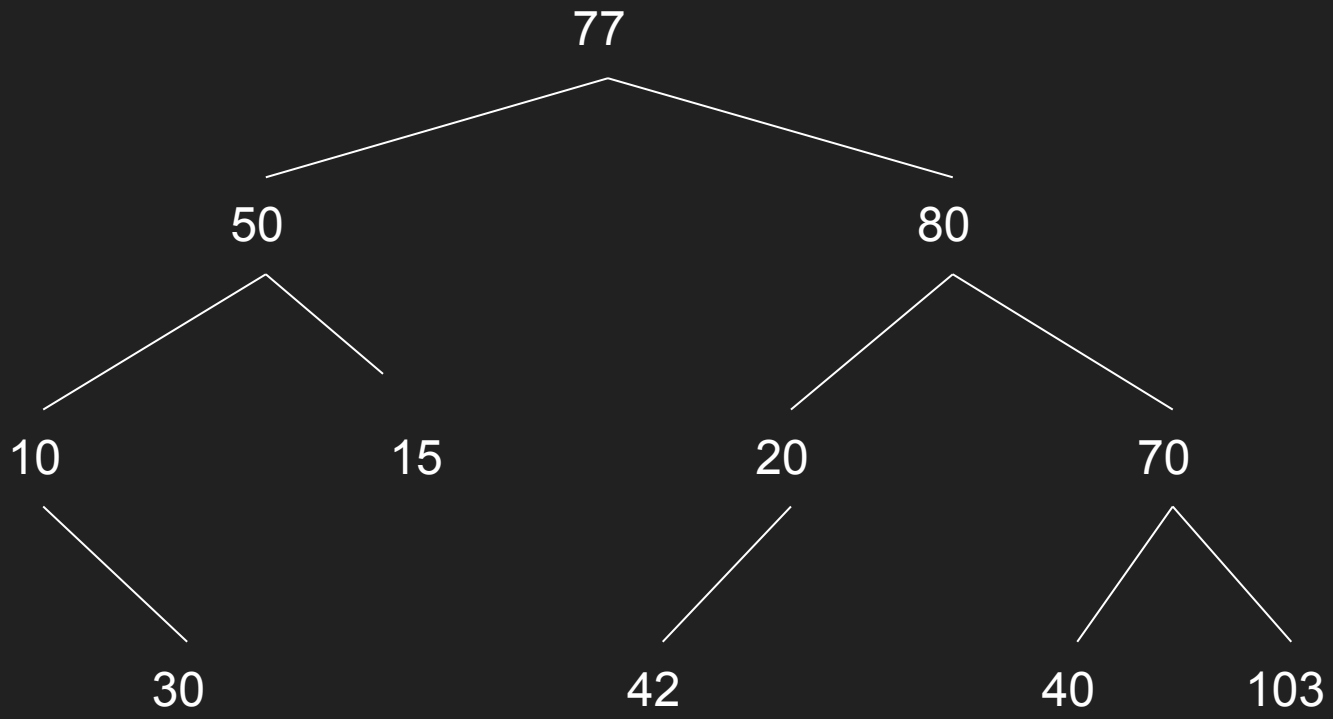
Uma árvore binária de busca já se trata de uma especialização de uma estrutura mais genérica denominada árvore.

Uma árvore é uma estrutura em que:

- há um conjunto de nós (cada um armazenando algum tipo de informação).
- existe um nó denominado “nó raiz”.
- existe uma ou mais sub árvores “penduradas” no nó raiz.

  (sim, é uma definição recursiva)

Exemplo: árvore binárias de busca



Árvores

Chamamos de nós descendentes os nós abaixo de um determinado nó.

O número de sub árvores abaixo de um dado nó é denominado grau do nó.

Em uma árvore binária completa, por exemplo, todos os nós possuem grau 2, com exceção dos “nós folhas” que possuem grau zero.

Árvores

Por convenção, assumimos que o nó raiz se encontra no nível zero.

Já a altura (h) de um nó, corresponde ao comprimento do caminho mais longo que existe entre o nó e uma folha.

A altura de uma árvore corresponde à altura da sua raiz.

Nem sempre uma árvore estará perfeitamente completa e balanceada.

Árvores binárias

Uma árvore binária é aquela em que abaixo de cada nó existem, no máximo, duas subárvores.

Árvores binárias

Uma árvore binária é aquela em que abaixo de cada nó existem, no máximo, duas subárvores.

E como representar uma árvore binária? Ligando nós.

Árvores binárias

Uma árvore binária é aquela em que abaixo de cada nó existem, no máximo, duas subárvores.

E como representar uma árvore binária? Ligando nós.

(lembra, de certa forma, o que fizemos para representar listas ligadas. De fato, uma lista ligada nada mais é do que um caso degenerado de árvore, em que cada nó possui apenas um descendente direto).

Árvores binárias

Estrutura para o nó de uma árvore binária:

```
typedef struct _no_arvore_ {
    Elemento valor;
    struct _no_arvore_ * esq;
    struct _no_arvore_ * dir;
} No;
```

Árvores binárias

Estrutura para representar a árvore binária em si:

```
typedef struct {
    No * raiz;
} ArvoreBinaria;
```

Árvores binárias

Sem assumir, por hora, qualquer tipo de organização dos valores armazenados na árvore (o que será necessário para podermos realizar buscas de forma eficiente), vamos discutir as seguintes operações em árvores binárias:

- criação
- busca
- impressão
- inserção
- remoção

Árvores binárias: criação

Criação de uma árvore vazia é bastante simples. É preciso:

- alocar uma struct do tipo ArvoreBinaria.
- atribuir NULL ao campo raiz.
- devolver o endereço da struct alocada.

Árvores binárias: busca

Busca:

- mesmo sem assumir qualquer tipo de organização dos elementos, devemos ser capazes de verificar a existência ou não de um elemento na árvore.

Árvores binárias: busca

Busca:

- mesmo sem assumir qualquer tipo de organização dos elementos, devemos ser capazes de verificar a existência ou não de um elemento na árvore.

- parece razoável imaginar que teremos que, eventualmente, passar por todos os elementos armazenados na árvore, comparando-os com o elemento procurado.

Árvores binárias: busca

Busca:

- mesmo sem assumir qualquer tipo de organização dos elementos, devemos ser capazes de verificar a existência ou não de um elemento na árvore.

- parece razoável imaginar que teremos que, eventualmente, passar por todos os elementos armazenados na árvore, comparando-os com o elemento procurado.

- Como passar por todos os elementos se a estrutura não é mais linear???

Árvores binárias: busca

Busca:

- Este é um bom momento para aplicar nosso conhecimento sobre recursão!

- Ideia: se o valor procurado estiver presente, ele deverá estar: na raiz ou na sub árvore esquerda ou na sub árvore direita

- Note que a procura em uma sub árvore, corresponde a resolver uma instância menor do mesmo problema inicial (ou seja, a procura nas subárvores será feita através de uma chamada recursiva).

Árvores binárias: busca

```
No * busca_rec(No * no, Elemento e){
    No * aux;
    if(no){
        if(no->valor == e) return no;
        if(aux = busca_rec(no->esq, e)) return aux;
        return busca_rec(no->dir, e);
    }
    return NULL;
}
```

Árvores binárias: busca

Análise de pior caso (valor buscado ausenta, ou é o último a ser verificado):

```
T(n) = c                          para n = 0
T(n) = T(x) + T(n - x - 1) + d    para n > 0
```

x irá variar conforme a distribuição dos elementos pela árvore.

Casos limites: x = 0, e x = (n - 1) / 2.

Árvores binárias: busca

Quando x = 0:

$$T(n) = T(0) + T(n - 1) + d$$

$$T(n) = T(n - 1) + k$$

$$T(n) = \theta(n)$$

Árvores binárias: busca

Quando x = (n - 1) / 2:

$$T(n) = 2T((n - 1) / 2) + d$$

$$T'(n) = 2T(n/2) + d$$

$$T'(n) = \theta(n)$$

Como, $T(n) < T'(n)$, então temos que $T(n) = O(n)$.

Árvores binárias: busca

Partindo do resultado `T(n) = O(n)`, podemos usar prova por indução para demonstrar que tal resultado é válido para qualquer valor de x.

Árvores binárias: busca

Partindo do resultado `T(n) = O(n)`, podemos usar prova por indução para demonstrar que tal resultado é válido para qualquer valor de x.

Não é um resultado melhor do que uma busca realizada sobre uma lista!

Árvores binárias: impressão (percurso)

Impressão:

- De forma similar à busca (não otimizada), devemos passar por cada nó uma vez e imprimir o valor armazenado nele.

- Por se tratar de uma estrutura não linear, há diversas ordens possíveis em que os nós podem ser visitados.

- Três tipos de percursos comuns em árvores binárias são: percurso *in-ordem*, percurso *pré-ordem* e percurso *pós-ordem.*

Árvores binárias: impressão com percurso *in-ordem*

```
void imprime_rec(No * no){
    if(no){
        imprime_rec(no->esq);
        printf(" %d", no->valor);
        imprime_rec(no->dir);
    }
}
```

Árvores binárias: impressão com percurso *pré-ordem*

```
void imprime_rec(No * no){
    if(no){
        printf(“ %d”, no->valor);
        imprime_rec(no->esq);
        imprime_rec(no->dir);
    }
}
```

Árvores binárias: impressão com percurso *pós-ordem*

```
void imprime_rec(No * no){
    if(no){
        imprime_rec(no->esq);
        imprime_rec(no->dir);
        printf(“ %d”, no->valor);
    }
}
```

Árvores binárias: impressão (percurso)

Impressão:

Os percursos *in-ordem*, *pré-ordem* e *pós-ordem* são as únicas formas de se percorrer os valores armazenados em uma árvore?

Árvores binárias: impressão (percurso)

Impressão:

Os percursos *in-ordem*, *pré-ordem* e *pós-ordem* são as únicas formas de se percorrer os valores armazenados em uma árvore?

Não, também podemos passar pelos nós um nível de cada vez. Neste caso, a implementação recursiva deixa de ser a mais “natural”. Para percorrer nível a nível devemos implementar uma função iterativa e usar uma fila como estrutura auxiliar.

Árvores binárias: inserção

Para inserir um novo elemento (valor) em uma árvore como filho de um nó previamente definido (ou seja, para o qual já temos um ponteiro), devemos:

- alocar um novo nó.
- armazenar valor a ser guardado no nó.
- “pendurar” o nó novo abaixo do nó pai (à esquerda ou direita).
- o filho original do pai (à esquerda ou à direita) passa a ser filho do novo nó.

Árvores binárias: inserção

Para inserir um novo elemento (valor) em uma árvore como filho de um nó previamente definido (ou seja, para o qual já temos um ponteiro), devemos:

- alocar um novo nó.
- armazenar valor a ser guardado no nó.
- “pendurar” o nó novo abaixo do nó pai (à esquerda ou direita).
- o filho original do pai (à esquerda ou à direita) passa a ser filho do novo nó.

- caso o novo nó deva ocupar o lugar de raiz da árvore, então a raiz original da árvore torna-se um dos filhos (à esquerda ou direita) do novo nó.

Árvores binárias: remoção

A remoção de um nó (para o qual já se tem um ponteiro, obtido através de uma busca pelo valor que desejamos remover) é uma operação um pouco mais complexa do que a busca, impressão ou inserção.

Quando removemos um nó, temos que lidar com a possibilidade de ter duas sub-árvores “soltas” que precisam ser reconectadas à árvore de alguma maneira.

(Note que se o nó a ser removido é uma folha, ou possui apenas um filho, o processo fica bem mais simples, mas a solução genérica também irá funcionar para estes dois casos).

Árvores binárias: remoção

Quando a remoção não envolver um nó folha, uma estratégia para implementar a remoção é não remover, efetivamente, o nó que é alvo da operação, mas apenas o valor armazenado nele, da seguinte forma:

1) Copiamos o valor de um dos nós filhos (esquerda ou direita), para o nó alvo da operação.
2) Com a realização desta cópia, na prática o valor desejado já foi removido da árvore.
3) Entretanto, a árvore fica em um estado inconsistente: o valor copiado toma o lugar do valor removido, mas continua presente no nó filho. Ou seja, está duplicado.

Árvores binárias: remoção

4) Como lidar com o valor duplicado? É simples. Basta remover o nó filho (que cedeu o valor ao nó alvo da operação) de forma recursiva.
5) O efeito de uma cadeia de chamadas recursivas feitas durante o processo de remoção acaba sendo o de “promover” (copiar para o nível imediatamente superior) aqueles valores presentes em um caminho que liga o nó alvo da operação, até uma folha da árvore.
6) Um nó só é efetivamente removido (perde a ligação com a árvore e sua memória liberada), quando se trata de uma folha (que é o caso base da recursão). Neste caso, precisamos atualizar o nó pai do nó sendo removido (folha) para que o valor nulo seja atribuído ao ponteiro que leva até o nó folha.