Estruturas de Dados 2

Prof. Silvia Brandão

2024.1

Momento N3

- Avaliação do projeto: 12 e 19/6
- Mostra Uniube: 20, 21 e 22/6

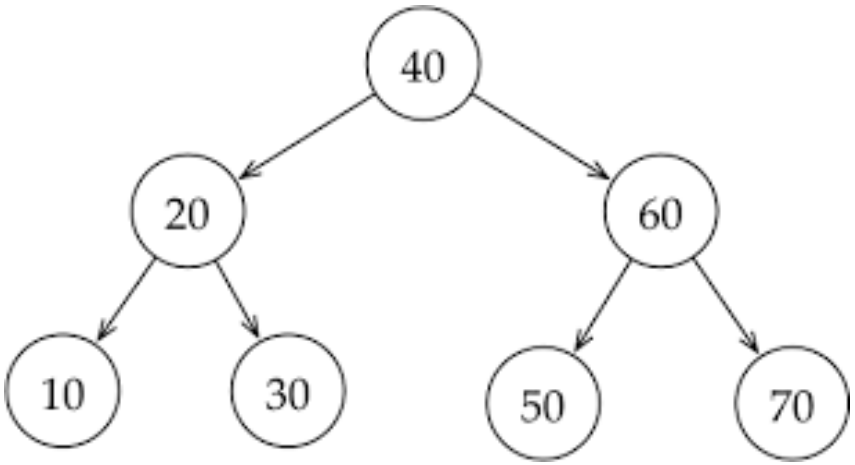
Métodos de Pesquisa em Memória

- Algumas técnicas de busca em memória interna são:
  - **Busca Sequencial**
  - **Busca Binária**
  - **Busca por Interpolação**
  - Busca em Árvores
  - Hashing

- O objetivo é encontrar um dado registro com o menor custo
- Cada técnica possui vantagens e desvantagens

Simulador:
https://www.cs.usfca.edu/~galles/visualization/Search.html

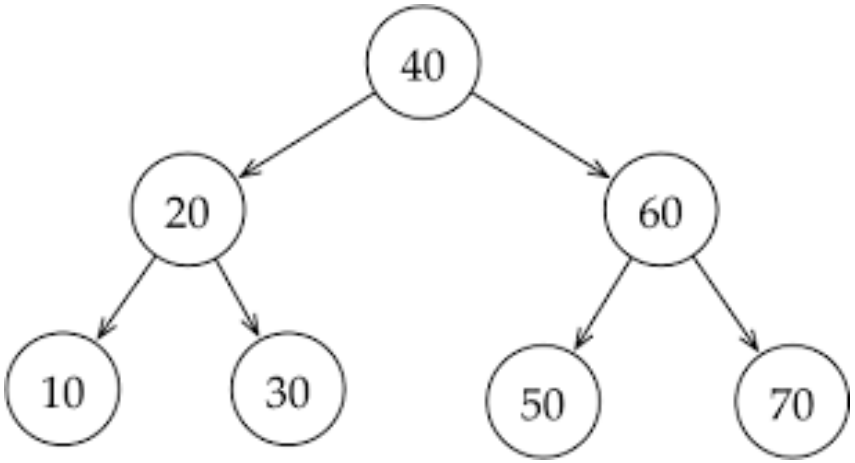
Busca em Árvores



- Na imagem podemos ver um exemplo de **árvore binária de busca**. Esse tipo de árvore tem como base a ideia do algoritmo de busca binária, que permite uma **procura**, **inserção** e **remoção** rápidas dos nós. Sendo esperado que em uma árvore desse tipo, eu caminhe para um lado ou para outro descartando metade dela.

- **Exemplo:** buscando o dado 30 na árvore da imagem, começamos pelo nó-raiz (40), indo para o lado esquerdo (20) que é menor que 30, daí chegamos onde queremos. O lado direito do nó 20 é um dado maior que ele. Assim encontramos o valor 30 sem precisar caminhar pelo lado direito da árvore.
  - Logicamente, pela noção de ordem da árvore binária de busca, o elemento mínimo se encontra do lado esquerdo e o elemento máximo, no direito. Reparando na nossa imagem anterior, vemos que o mínimo e o máximo são as folhas, 10 e 70, que estão do lado esquerdo e direito respectivamente.

Busca em Árvores



- Para inserir elementos na árvore binária de busca, começamos do nó-raiz da árvore e descemos de modo recursivo, procurando pelo local certo para inserir o novo nó.
  - **Exemplo:** temos um nó-raiz 8, o próximo dado seria o 4, ele ficaria a esquerda de 8, pois é menor que ele. Agora temos o dado 6, ele seria o nó filho à direita do 4, pois ele é maior que ele.

- No caso da remoção de um dado, teríamos os seguintes casos:
  - no caso de um nó vazio, não há nada para remover nele,
  - no caso de um nó a ser removido ser um nó folha, removemos a folha,
  - no caso de um nó a ser removido ter um filho, temos que conectar o filho com o pai do nó que será removido. Ainda,
  - se o nó a ser removido tiver dois filhos, primeiro pegamos o sucessor e colocamos no local do nó a ser removido antes de efetuar a remoção.
    - **Exemplo: d**eletando o nó 20 da nossa imagem, primeiro o 30 iria para o seu lugar e o 20 seria removido após isso.

Hashing

- Também conhecido como tabela de *espalhamento* ou de *dispersão*
- *Hashing* é uma técnica que utiliza uma função *h* para transformar uma chave *k* em um endereço
  - O endereço é usado para gravar e buscar registros
  - Deve ser determinística, ou seja, resultar sempre no mesmo valor para uma determinada chave

- Ideia: particionar um conjunto de elementos (possivelmente infinito) em um número finito de classes
  - *m* classes → endereçamento de 0 a *m*–1

Hashing – overflow progressivo

- Exemplo de dificuldade: busca pelo nome “Smith”

h(SMITH) = 7

	...
7	ADAMS
8	JONES
9	MORRIS
10	SMITH

Pode ter que percorrer muitos campos

h(SMITH) = 7

	...
7	ADAMS
8	JONES
9	
10	SMITH

A remoção do elemento no índice 9 pode causar uma falha na busca

Hashing – overflow progressivo

- Exemplo de dificuldade: busca pelo nome “Smith”

h(SMITH) = 7

	...
7	ADAMS
8	JONES
9	MORRIS
10	SMITH

Pode ter que percorrer muitos campos

h(SMITH) = 7

	...
7	ADAMS
8	JONES
9	
10	SMITH

A remoção do elemento no índice 9 pode causar uma falha na busca

h(SMITH) = 7

	...
7	ADAMS
8	JONES
9	#####
10	SMITH

Solução para remoção de elementos: eliminar elemento, mas indicar que a posição foi esvaziada e que a busca deve continuar

Exercício

Assumindo que:

- $m = 10$
- $h(k) = k \% m$
- Colisões são tratadas com sondagem linear, no pior caso,
  - $h(k)' = (h(k)+i) \% m$, com $i = 1, \ldots, m-1$

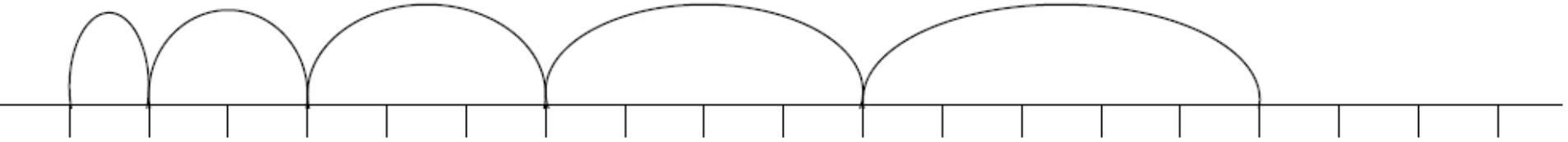
- Insira os seguintes elementos em uma tabela *hash* com *overflow* progressivo
  - 41, 10, 8, 7, 13, 52, 1, 89, 64
- Outro exemplo
  - $h(k)' = (h(k) + c1*i + c2*i^2) \% m$, com $i=1, \ldots, m-1$ e constantes $c1$ e $c2$
  - Chamada sondagem quadrática, considerada melhor do que a linear, pois evita o agrupamento de elementos

Overflow progressivo

Sondagem linear

$h(k)' = (h(k) + i) \% m$

$h(k)$



$h(k)$

Sondagem quadrática

$h(k)' = (h(k) + 0{,}5*i + 0{,}5*i^2) \% m$

c_1 c_2

Específicos para este exemplo

Overflow progressivo

- **Vantagem**
  - Simplicidade
- **Desvantagens**
  - Agrupamento de dados (causado por colisões)
  - Com estrutura cheia, a busca fica lenta
  - Dificulta inserções e remoções
- Alternativamente, podemos representar a sondagem linear como uma única função dependente do número da tentativa *i*:
  - **Por exemplo:** $h(k, i) = (k+i) \% m$, com $i = 0, \ldots, m-1$
    - A função *h* depende agora de dois fatores: a chave *k* e a iteração *i*
    - Note que $i=0$ na primeira execução, resultando na função *hash* tradicional de divisão que já conhecíamos
    - Quando $i=1,\ldots, m-1$, já estamos aplicando a técnica de sondagem linear

Overflow progressivo

Exercício: implemente sub-rotinas de inserção, busca e remoção utilizando a função hash anterior.
Considere que a tabela hash é inicializada com o valor -1 em todos os endereços:

```
void inicializa(int T[], int m) {
     int i;
     for (i = 0; i < m; i++)
         T[i] = -1;
}
```

Considere também que, ao remover um elemento, seu endereço recebe o valor -2.

Overflow progressivo

```
int inserir(int T[], int m, int k) {
     int i, j;
     for (i = 0; i < m; i++) {
          j = (k + i) % m;
          if ((T[j] == -1) || (T[j] == -2)) {
             T[j] = k;
             return j;
          }
     }
     return -1;
}
```

```
int buscar (int T[], int m, int k) {
     int i, j;
     for (i = 0; i < m; i++) {
          j = (k + i) % m;
          if (T[j] == k)
               return j;
          else if (T[j] == -1)
               return -1;
     }
     return -1;
}
```

Overflow progressivo

```
int remover(int T[], int m, int k) {
    int i, j;
    for (i = 0; i < m; i++) {
        j = (k + i) % m;
        if (T[j] == k) {
           T[j] = -2;
           return j;
        } else if (T[j] == -1)
           return -1;
    }
    return -1;
}
```

Principais desvantagens do *hashing*?

- Os elementos da tabela não são armazenados sequencialmente
- Não existe um método prático para percorrê-los em sequência

Hashing

- Para aprofundar em Busca por meio de Hashing, estude:
- https://panda.ime.usp.br/panda/static/pythonds_pt/05-OrdenacaoBusca/Hashing.html

Referência

- Ziviani, Nivio. **Projeto de Algoritmos com implementações em JAVA e C++.**

- Search. https://www.cs.usfca.edu/~galles/visualization/Search.html

- Drozdek, A.. **Estrutura de Dados e Algoritmos em C++**. Cengage Learning, 2002.