Algoritmos e Estruturas de Dados no Big Data
MLP - ESBD1 - Aula 2

- Importância dos breaks a cada 50 min / 1 hora

Overview:

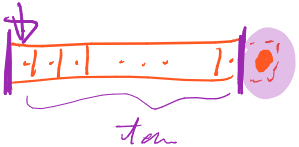
- Analisar a Atividade 1
- Recapitular a primeira aula
- Árvores balanceadas de busca
- Tabelas de espalhamento

busca linear — busca binária ou árvore balanceada busca

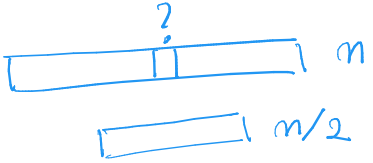
Analisando a Atividade 1: O(n) — O(lg n) — O(1) → Hash Tables (na prática) em segundos

Tam. do Conjunto (n)	Tempo Função 1	Tempo Função 2	Tempo Função 3
10	0.012	0.014	0.0075
100	0.060	0.035	0.0065
1000	0.61	0.055	0.0078
10000	5.9	0.068	0.0070
100000	208.5	0.11	0.0079
1000000	2022.6	0.12	0.0091

10x, 5x, 10x, ×2, ×2, ×3, ×4, 10x, 40x, 10x, x 1000



$n \ll n \lg n$



$2^? > 1000 \Rightarrow ? = 10$

$2^? \cdot 1000 > 1000.000$

$\Rightarrow ? = 10$

$\lg 10 = t$

$\lg 100 = \lg 10^2 = 2 \lg 10 = 2t$

$\lg 10^6 = 6 \lg 10 = 6t$

$\lg k^2 = 2 \lg k$

$\lg 10^6 = \lg (10^3)^2 = 2 \lg 10^3$

Vamos analisar algumas funções e seu padrão de crescimento:

n	10	100	1000	...	10^6
$O(n)$	$1t$	$10 \cdot 10 = 10t$	$100t$	...	$10^5 t$
$O(\lg n)$	$1t$	$2t$	$3t$	...	$6t$

Olhar o padrão de crescimento em cada coluna

- e, em particular, entre as linhas 1000 e 1000000.

Eram duas as perguntas por responder:

- Qual função matemática descreve a eficiência de cada função de busca?
- Consegue especular qual estrutura de dados está implementada em cada função?

Assim, temos três funções de busca,

- Função 1 cresce em $O(n)$ e é uma implementação da busca linear.
- Função 2 cresce em $O(\lg n)$ e pode ser uma implementação
  - da busca binária (ou de uma árvore de busca balanceada).
- Função 3 é quase constante, i.e., $O(1)$, e pode ser uma implementação
  - de uma tabela de espalhamento.

Então a tabela de espalhamento (Hash Table) é a solução para todos os problemas?

Hoje veremos duas categorias de estruturas de dados

- que são usadas para implementar índices e dicionários,
    - pois ambas suportam buscas e inserções eficientes.

Elas são as árvores balanceadas de busca e as tabelas de espalhamento.

Enquanto ambas atacam problemas das estruturas mais simples

    - que vimos na última aula,
- i.e., busca linear em listas ligadas e busca binária em vetor ordenado,
    - a primeira ineficiente na busca e a segunda na inserção,
- vamos descobrir que árvores de busca e tabelas de espalhamento
    - tem abordagens muito diferentes,
        - e apresentam prós e contras bem particulares.

Relembrando princípios e conteúdos da primeira aula:

Algoritmos e estruturas de dados vistos neste curso

- são usados na implementação de algoritmos
    - de aprendizado de máquinas e de bancos de dados, por exemplo.

Saber analisar cada cenário do seu problema para decidir

- qual a ferramenta que melhor se encaixa
    - é a marca do bom projetista (mesmo que mudem as ferramentas).

Conhecer algoritmos e estruturas de dados

- para problemas fundamentais (destaque para a busca),
    - saber prós e contras de cada uma
        - e ter noção das estruturas implementadas.

Vimos vetores e listas ligadas e algoritmos de busca linear e binária,

- que são estruturas e algoritmos fundamentais,
- mas que não resolviam bem o problema da busca em conjuntos dinâmicos.

Árvores balanceadas de busca:

não são binárias

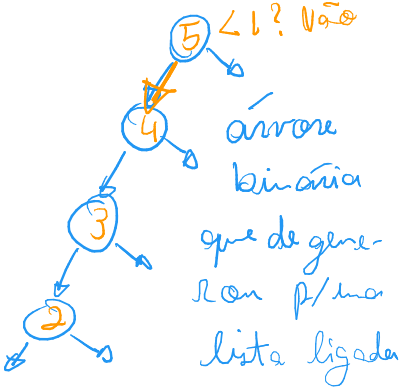
Exemplos são as árvores AVL, rubro-negra, B e B+.

binárias

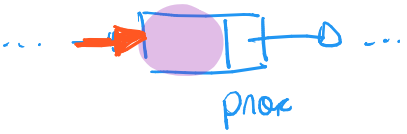
São estruturas ligadas e ordenadas, combinando aspectos

- das listas ligadas e da busca binária em vetor ordenado.

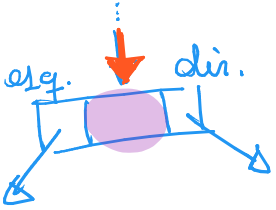
buscando 1

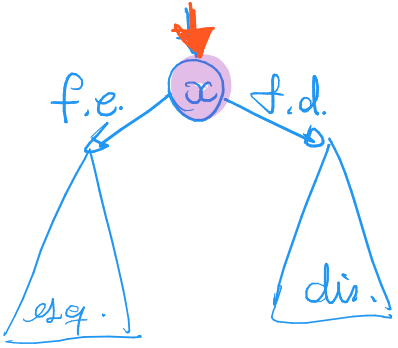


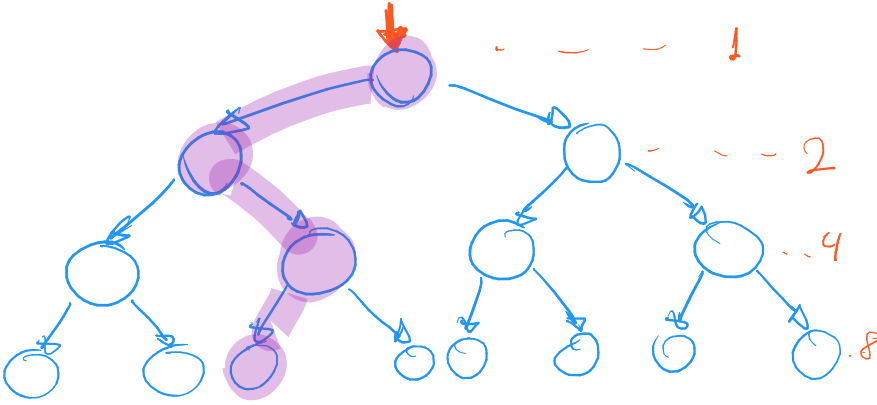
nó de lista ligada



nó árv. binária

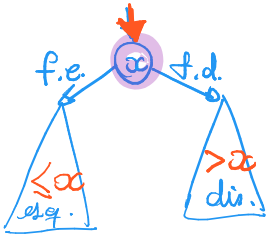


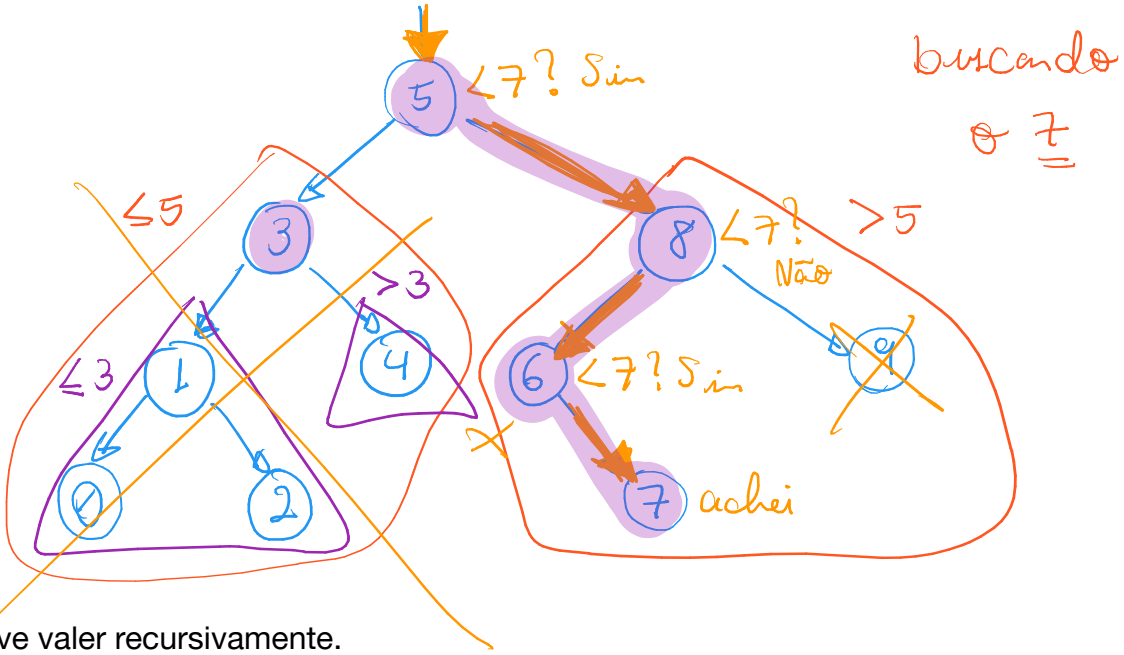




Propriedade de busca: todo nó na subárvore esquerda é $\leq$ que a raiz

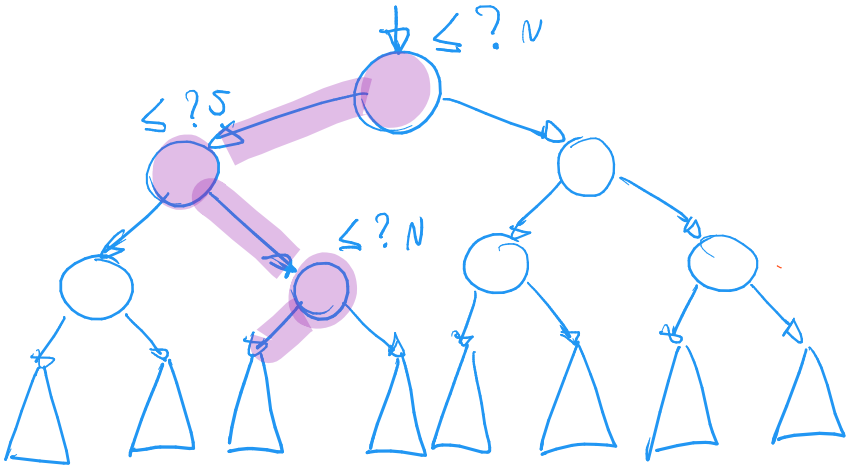
- e todo nó na subárvore direita é $\geq$ que a raiz.

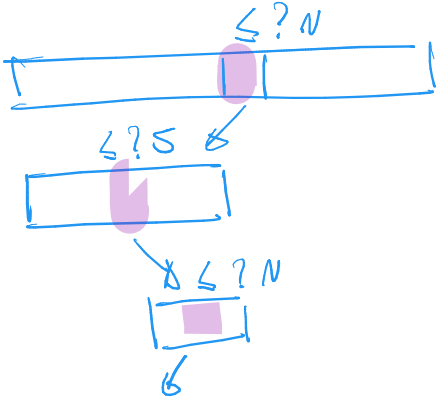




- Deve valer recursivamente.

Buscando numa árvore binária de busca e a comparação com a busca binária.





Quais outras operações são suportadas?

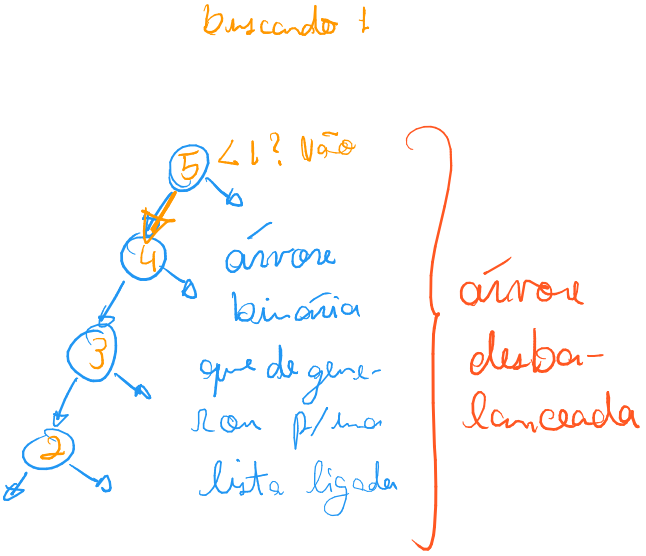
- Suportam muitas operações além de busca, inserção e remoção,
    - como mínimo (máx.), predecessor (suc.), seleção, e intervalo.

Qual a eficiência dessas operações (em particular, das buscas)?

- Praticamente todas com eficiência $O(\lg n)$
    - sendo n o número de elementos do conjunto.
- Do que essa eficiência depende?
    - De sucessivas divisões do espaço de busca.

Desenvolvendo a questão da eficiência: as sucessivas divisões do espaço de busca

- tem relação com a altura e com o balanceamento da árvore.



Curiosidade: o uso de cada árvore varia de acordo com a aplicação.

- Exemplos: uso em memória principal (AVL, rubro-negra) ou em disco (B, B+).

Bônus: mecanismos de balanceamento são rotações e divisões de nós.

https://colab.research.google.com/drive/1hl9Nm-fJrnYDXCD2BWOswpwaJ94iT_WJ

Será que podemos fazer melhor que buscar em tempo $O(\lg n)$?

Tabelas de espalhamento:

Será que podemos fazer melhor que buscar em tempo $O(\log n)$?

- Na prática, Hash Tables permitem fazer
  - busca, inserção e remoção em tempo quase constante,
    - mas não suportam outras operações com eficiência.
- Como isso é possível?

Vamos abandonar a ideia de dividir sucessivamente o espaço de busca,

- e nos inspirar em vetores diretamente indexados/endereçados pelas chaves.

elemento c/ chave K

$\Theta(1)$

Será que isso funciona para chaves distribuídas em um grande intervalo?

Como exemplo, considere um dicionário para guardar sua agenda de telefones.

- Pensem no universo dos números de telefone,
  - i.e., no conjunto com todos os números de telefone possíveis.
- Quão grande é esse universo?

U

$10 \cdot 10 \cdot 10 \cdot \ldots \cdot 10 \cdot 10 = 10^9 = 1$ bilhão = # de chaves = $|U|$ possíveis

Qual o tamanho do vetor diretamente indexado que guarda os contatos?

1 bilhão

Qual o número de posições ocupadas nesse vetor?

amigos/contatos

% de uso da tabela $= \frac{10^3}{10^9} = \frac{1}{10^6} = 0{,}0001\%$

Temos um problema, pois em geral as chaves estão em um universo U

- e meu conjunto tem apenas M elementos.
- Idealmente, a tabela deveria ser proporcional ao número de elementos,
  - e não ao número de diferentes valores que uma chave pode assumir.

Com isso em mente, podemos pensar em tabelas de espalhamento como

- um vetor que utiliza indexação quase direta.

Esse quase é porque nós precisamos de uma função de espalhamento

- para transformar a chave (que pode pertencer a um grande intervalo)
  - em um índice do vetor (que tem tamanho relativamente pequeno).

$f : U \longrightarrow \{0, 1, \ldots, M-1\}$ $\quad |U| = 1 \text{ bilhão}$

$f(999738132) \overset{?}{\longrightarrow} 79$ $\quad M = 100$ $\quad \#\text{chaves por posição} = \frac{10^9}{10^2} = 10^7$

Operações e eficiência (garantida?)

- Busca, inserção e remoção podem ser feitas em tempo quase constante.
  - E as outras operações? Notou que os dados ficam espalhados?
    - O que isso implica?

O tempo quase constante é porque

  - ainda é necessário resolver a função de espalhamento
- e podem ocorrer conflitos,
  - i.e., duas chaves caírem na mesma posição.

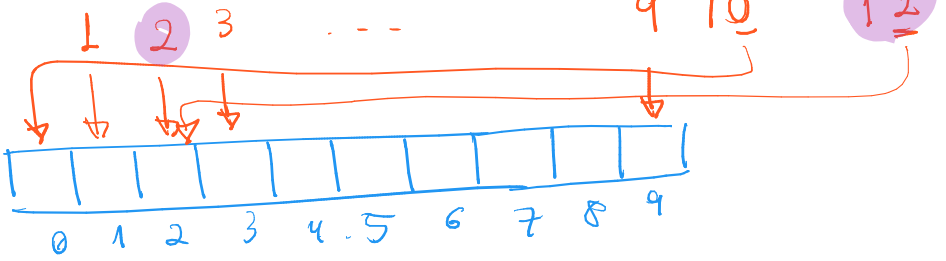
Exemplo de uma função de espalhamento ruim

- e da ocorrência de diversos conflitos.

da dir. p/ a esq.

f (devolve o 1º dígito do # de telefone)

f :



M = 10

f(99796953) → 3

f(98765323) → 3

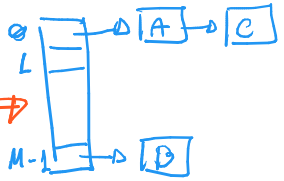
% M

Boas funções de espalhamento tentam minimizar conflitos,

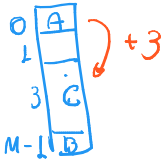
- mas estes são inevitáveis.

Existem duas maneiras principais de tratá-los:

- listas ligadas



- e reespalhamento.



$f(A) = 0$

$f(B) = M-1$

$f(C) = 0$ $g(C) = 3$

Além de uma boa função de espalhamento,

- o número de conflitos pode ser reduzido mantendo uma carga adequada.

A carga diz respeito a quanto da tabela está ocupada.

- Em geral, tabelas de espalhamento são muito eficientes
  - com cargas em torno de 50%,
    - i.e., que usam 2x o espaço mínimo necessário,
  - mas até cargas de 70% mantêm um desempenho bastante bom.

Códigos do Edu - https://colab.research.google.com/drive/1HiUvqln49QWKQBgndx-ciyN5qTYT9ac4?usp=sharing (http://bit.ly/EduardoMolinaAula2) MLP ESBD1 Aula 2 Códigos

Material complementar

Dicionários e índices: árvores de busca balanceadas

- [Playlist] Tabelas de Símbolos (AED2) - http://bit.ly/MarioSanFeliceTabSimbVideo
- [PDF] Apresentação, estruturas de dados, tabelas de símbolos - http://bit.ly/MarioSanFeliceCompTabSimbPDF
- [Playlist] Árvores binárias de busca, altura e balanceamento - http://bit.ly/MarioSanFeliceArvBinBusVideo
- [PDF] Árvores binárias de busca, altura e balanceamento - http://bit.ly/MarioSanFeliceCompArvBinBusPDF
- [Playlist] Rotações e árvores AVL: definição e inserção - http://bit.ly/MarioSanFeliceArvAVLP1Video
- [PDF] Rotações e árvores AVL: definição e inserção - http://bit.ly/MarioSanFeliceCompArvAVLP1PDF
- [Playlist] Árvores AVL: altura máxima e remoção - http://bit.ly/MarioSanFeliceArvAVLP2Video
- [PDF] Árvores AVL: altura máxima e remoção - http://bit.ly/MarioSanFeliceCompArvAVLP2PDF
- [Playlist] Árvores rubro-negras - http://bit.ly/MarioSanFeliceArvRubNegVideo
- [PDF] Árvores rubro-negras - http://bit.ly/MarioSanFeliceCompArvRubNegPDF

Dicionários e índices: tabelas de espalhamento (hash tables)

- [Playlist] Hash tables, espalhamento e colisões - http://bit.ly/MarioSanFeliceHashTabP1Video
- [PDF] Hash tables, espalhamento e colisões - http://bit.ly/MarioSanFeliceCompHashTabP1PDF
- [Playlist] Hash tables, tratando colisões e dimensionando carga - http://bit.ly/MarioSanFeliceHashTabP2Video
- [PDF] Hash tables, tratando colisões e dimensionando carga - http://bit.ly/MarioSanFeliceCompHashTabP2PDF