



CEUB

EDUCAÇÃO SUPERIOR

ceub.br

Estruturas de Dados

Professor Doutor

Auto Tavares da Camara Junior

Perito Criminal Federal

Apresentação

- Junior auto.junior@ceub.edu.br
 - Perito Criminal Federal
 - Doutor em Ciência da Informação (UnB)
 - Mestre em Ciência da Informação (UnB)
 - Pós Graduado MBA em Administração Estratégica de Sistemas de Informação (FGV)
 - Bacharel em Ciência da Computação (UnB)

Objetivo

- Reconhecer e utilizar estruturas de alocação dinâmica de memória unidimensionais
- Reconhecer e utilizar estruturas de alocação dinâmica de memória multidimensionais
- Implementar estruturas de alocação dinâmica de memória
- Implementar algoritmos de otimização

Organização

CEUB

EDUCAÇÃO SUPERIOR

- Aulas
 - Práticas
- Avaliação
 - Conceito (C)
- Nota Final
 - 100% (C)

Agenda

Terça-Feira

Data	Conteúdo	Data	Conteúdo	Data	Conteúdo
25/07	Vetores	01/08	Algoritmos de Ordenação Algoritmos de Busca	08/08	Listas Encadeadas
15/08	Filas Pilhas	22/08	Matrizes Esparsas	29/08	Árvore
05/09	Conceito 1	12/09	Árvore Binária	19/09	Árvore de Busca
26/09	Árvore Rubro-Negras	03/10	Árvore AVL	10/10	Árvore B
17/10	Conceito 2	24/10	Grafos	31/10	Percursos em Grafos
07/11	Algoritmos em Grafos	14/11	Árvore Geradora Mínima	21/11	Conclusão
28/11	Conceito 3	05/12	Conclusão	12/12	Conclusão

Agenda

Quarta-Feira

Data	Conteúdo	Data	Conteúdo	Data	Conteúdo
26/07	Vetores	02/08	Algoritmos de Ordenação Algoritmos de Busca	09/08	Listas Encadeadas
16/08	Filas Pilhas	23/08	Matrizes Esparsas	30/08	Árvore
06/09	Conceito 1	13/09	Árvore Binária	20/09	Árvore de Busca
27/09	Árvore Rubro-Negras	04/10	Árvore AVL	11/10	Árvore B
18/10	Conceito 2	25/10	Grafos	01/11	Percursos em Grafos
08/11	Algoritmos em Grafos	15/11	Proclamação da República	22/11	Árvore Geradora Mínima
29/11	Conceito 3	06/12	Conclusão	13/12	Conclusão

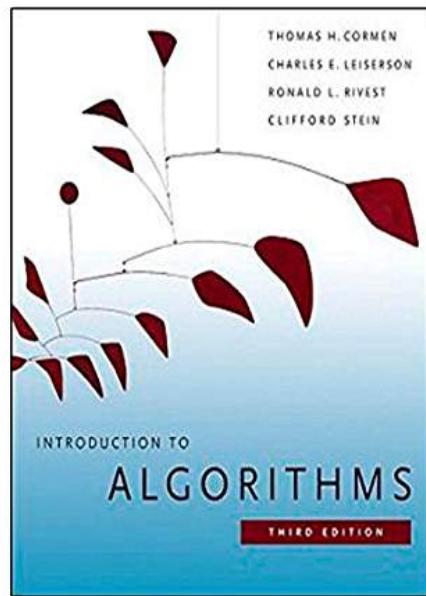
Agenda

Quinta-Feira

Data	Conteúdo	Data	Conteúdo	Data	Conteúdo
27/07	Vetores	03/08	Algoritmos de Ordenação Algoritmos de Busca	10/08	Listas Encadeadas
17/08	Filas Pilhas	24/08	Matrizes Esparsas	31/08	Árvore
07/09	Independência do Brasil	14/09	Árvore Binária	21/09	Árvore de Busca
28/09	Árvore Rubro-Negras	05/10	Árvore AVL	12/10	Dia das Crianças
19/10	Árvore B	26/10	Grafos	02/11	Finados
09/11	Percursos em Grafos	16/11	Algoritmos em Grafos	23/11	Árvore Geradora Mínima
30/11	Dia do Evangélico	07/12	Conclusão	14/12	Conclusão

Referências

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introduction to Algorithms



Vetores

Vetores

- Alocação estática de memória
- Os itens de dados de um vetor são chamados elementos
- Todos os elementos devem ser do mesmo tipo de dado
- Todos os elementos são contíguos na memória
- O nome do vetor representa o endereço de seu primeiro elemento

Vetores

- `int[] numeros = new int[3];`
 - `numeros[0] = 1;`
 - `numeros[1] = 2;`
 - `numeros[2] = 3;`
- `char vogais[] = {'a', 'e', 'i', 'o', 'u'};`
- Determinação do tamanho
 - `sizeof`
 - `length`

Vetores

- `int[][] tabela = new int[3][3];`
 - `tabela[0][0] = 11;`
 - `tabela[0][1] = 12;`
 - `tabela[0][2] = 13;`
 - `tabela[1][0] = 21;`
 - `tabela[1][1] = 22;`
 - `tabela[1][2] = 22;`
 - `tabela[2][0] = 31;`
 - `tabela[2][1] = 32;`
 - `tabela[2][2] = 33;`

Vetores

- `int[][][] tabela3D = new int[2][2][2];`
 - `tabela3D[0][0][0] = 111;`
 - `tabela3D[0][0][1] = 112;`
 - `tabela3D[0][1][0] = 121;`
 - `tabela3D[0][1][1] = 122;`
 - `tabela3D[1][0][0] = 211;`
 - `tabela3D[1][0][1] = 212;`
 - `tabela3D[1][1][0] = 221;`
 - `tabela3D[1][1][1] = 222;`

Vetores

- Programa preenchimento de vetor
- Programa inversão de valores de vetor
- Programa criação da matriz transposta
 - original[m][n]
 - transposta[n][m]
- Programa multiplicação de matrizes
 - operando1[m][p]
 - operando2[p][n]
 - resultado [m][n]

Algoritmos de Ordenação

Algoritmos de Busca

Algoritmos de Ordenação

CEUB

EDUCAÇÃO SUPERIOR

- Ordenação direta
- Ordenação BubbleSort

Algoritmos de Busca

- Busca exaustiva
- Busca binária

Listas Encadeadas

Listas Encadeadas

- Alocação dinâmica de memória
- Aritmética de ponteiros em C
- Instanciação de objetos em Java

Listas Encadeadas

- Programa lista de encadeamento simples
- Programa lista de encadeamento duplo
- Programa lista de encadeamento circular



Filas
Pilhas

Filas

- Política FIFO (first in - first out)

Pilhas

- Política FILO (first in - last out)

Matrizes Esparsas

Matrizes Esparsas

- Cache de memória secundária

Árvores



Árvores

- Uma das mais importantes estruturas de dados não lineares.
- Conceitualmente, diferente das listas, em que os dados se encontram numa sequência, nas árvores os dados estão dispostos de forma hierárquica.
- São estruturas eficientes e simples em relação ao tratamento computacional, por meio de algoritmos recursivos.
- Há inúmeros problemas no mundo real que podem ser modelados e resolvidos por meio de árvores.
 - Estruturas de pastas de um sistema operacional
 - Interfaces gráficas
 - Bancos de dados
 - Sites da Internet

Árvores

- Formada por um conjunto de elementos que armazenam informações chamados nós.
- Toda a árvore possui o elemento chamado raiz, que possui ligações para outros elementos denominados filhos.
- Esses filhos podem estar ligados a outros elementos que também podem possuir outros filhos.
- O elemento que não possui filhos é conhecido como folha.

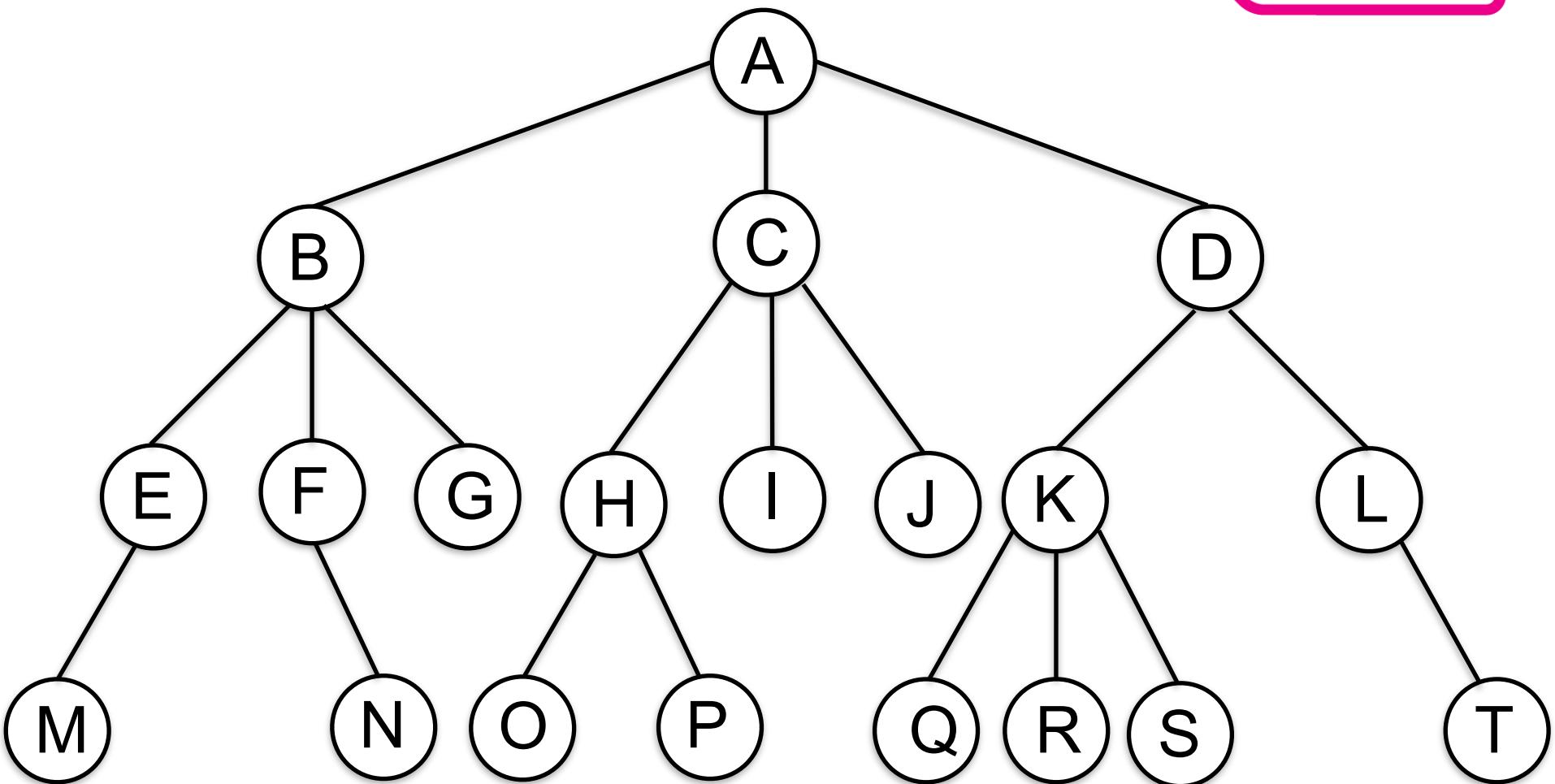
Árvores

- Formalmente, define-se uma árvore T como um conjunto finito de zero ou mais nós tal que:
 - Se o número de nós = 0
 - A árvore é vazia
 - Se o número de nós > 0
 - Existe um nó especialmente denominado raiz de T
 - Os nós restantes formam $m \geq 0$ conjuntos disjuntos $p_1, p_2, p_3, \dots, p_m$, sendo cada um desses conjuntos uma árvore em si, chamada subárvore da raiz de T

Árvores

- Ordem
 - A quantidade de filhos dos nós da árvore
- Nível
 - A distância entre um nó e a raiz
- Altura
 - O maior nível entre todas as folhas da árvore + 1

Árvores



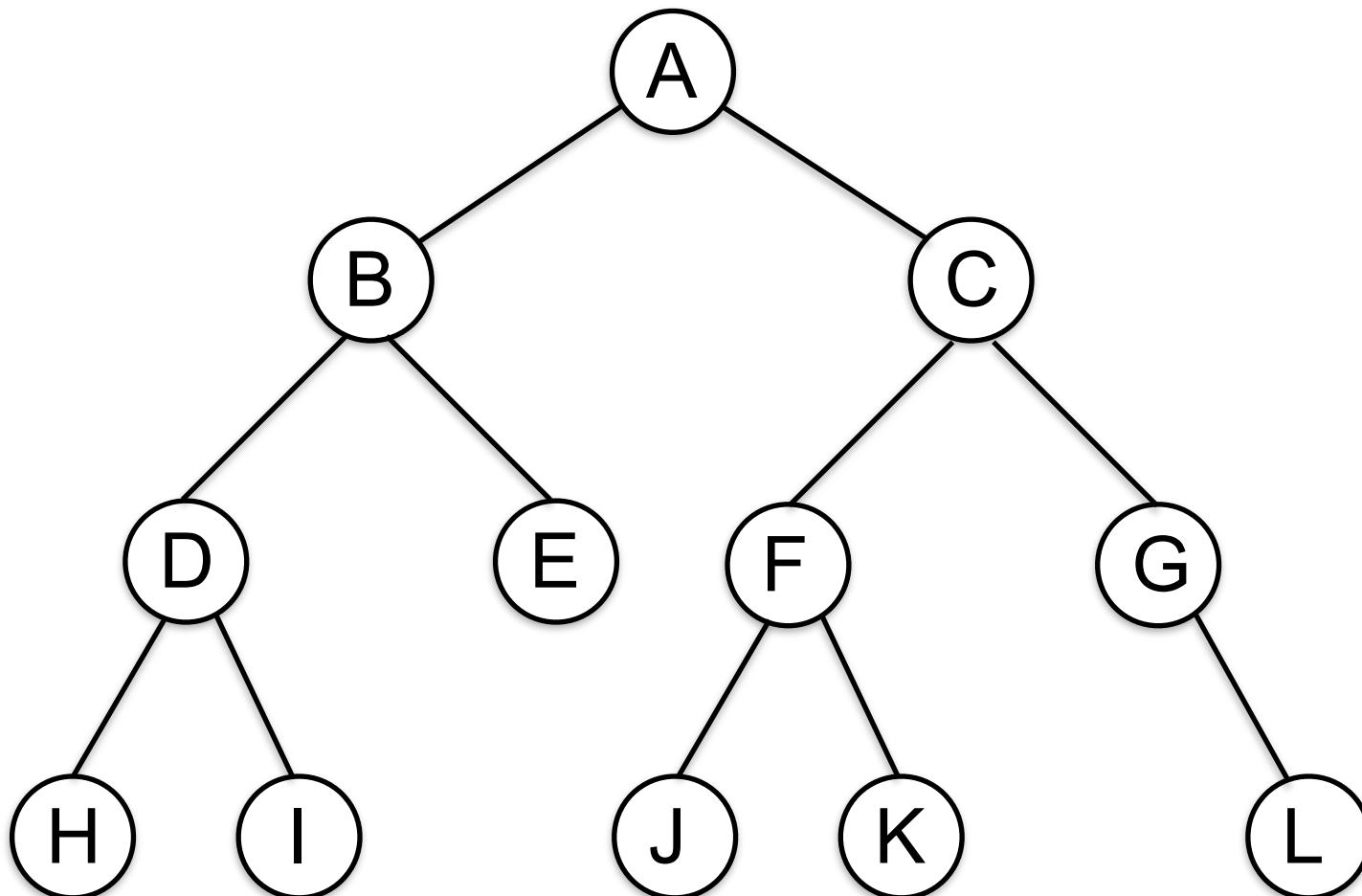
Árvores

- Programa construção de uma árvore
 - Ordem 5
 - Sorteio para criação dos nós
 - Navegação

Árvores Binárias

Árvores Binárias

- Árvores cuja ordem é igual a 2



Árvores Binárias

- Se a altura é h , então a árvore:
 - tem no mínimo h nós
 - tem no máximo $2^h - 1$ nós
- Se a árvore tem $n \geq 1$ nós, então:
 - A altura é no mínimo $\log(n + 1)$
 - Quando a árvore é completa
 - A altura é no máximo n
 - Quando cada nó não-terminal tem apenas um filho

Árvores Binárias

- Travessia
 - Pré-ordem
 - Raiz
 - Subárvore esquerda
 - Subárvore direita

- Travessia
 - Em-ordem
 - Subárvore esquerda
 - Raiz
 - Subárvore direita

Árvores Binárias

- Travessia
 - Pós-ordem
 - Subárvore esquerda
 - Subárvore direita
 - Raiz

Árvores Binárias

- Travessia
 - Pré-ordem
 - ABDHIECFJKGL
 - Em-ordem
 - HDIBEAJFKCGL
 - Pós-ordem
 - HIDEBJKFLGCA

Árvores Binárias

- Programa construção da árvore binária
 - Sorteio para criação dos nós
 - Navegações

Árvores de Busca

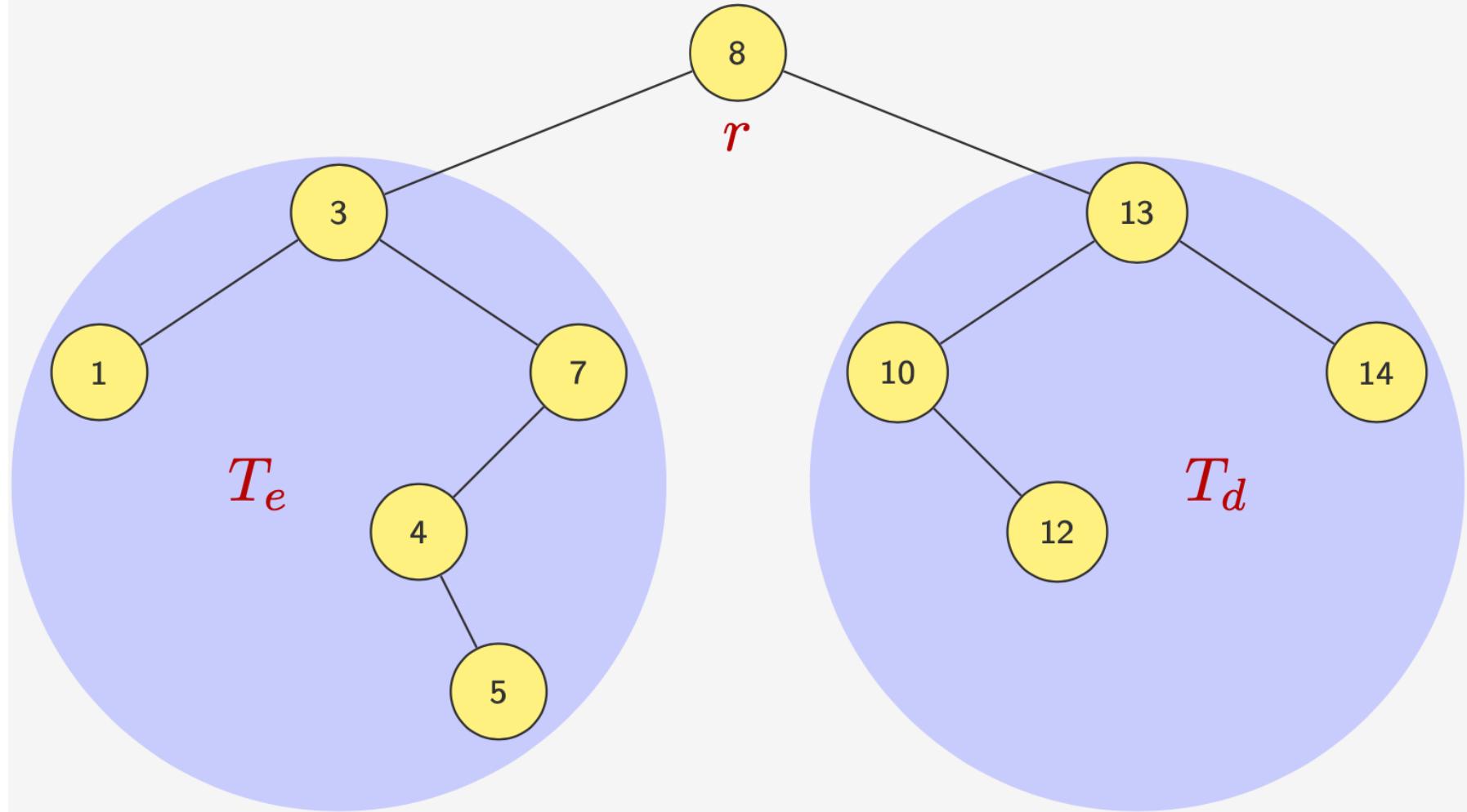
Árvores de Busca

- Complexidade para inserção, remoção e busca
 - Listas encadeadas
 - Inserir e remover em $O(1)$
 - Buscar em $O(n)$
 - Vetores não-ordenados
 - Inserir e remover em $O(1)$
 - Inserir no final
 - Remover trocar com o último e remover o último
 - Buscar em $O(n)$
 - Vetores ordenados
 - Buscar em $O(\log n)$
 - Inserir e remover em $O(n)$
 - Árvores binárias de busca
 - As três operações levam $O(\log n)$

Árvores de Busca

- Uma árvore de busca é uma árvore binária em que cada nó contém um elemento de um conjunto ordenável
- Cada nó r , com subárvore esquerda T_e e direita T_d satisfaz as seguintes propriedades
 - $e < r$ para todo elemento $e \in T_e$
 - $d > r$ para todo elemento $d \in T_d$

Árvores de Busca



Árvores de Busca

- Busca de elemento
 - Algoritmo recursivo para encontro do valor
 - Se estiver na raiz, devolve
 - Se for menor do que a raiz, procurar à esquerda
 - Se for maior do que a raiz, procurar à direita
- Inserção de elemento
 - Algoritmo de busca
 - Inserir onde o elemento deveria estar

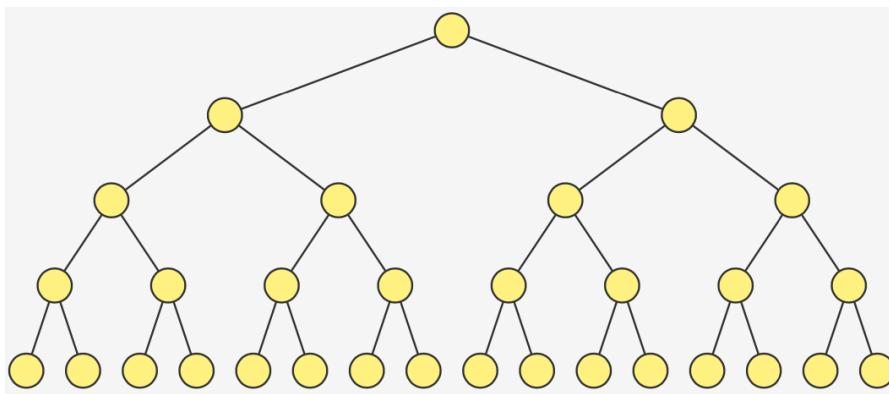
Árvores de Busca

- Programa construção da árvore de busca
 - Busca de elemento
 - Inserção de elemento
 - Navegação ordenada

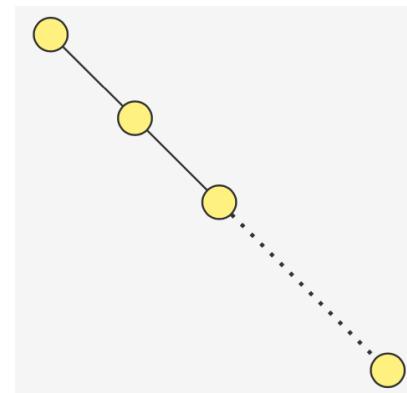
Árvores Rubro-Negras

Árvores Rubro-Negras

- Complexidade para inserção, remoção e busca
 - Depende, na verdade, da altura da árvore



Melhor árvore



Pior árvore

- A pior árvore é formada simplesmente por inserção na ordem crescente
 - A melhor solução é manter a árvore balanceada

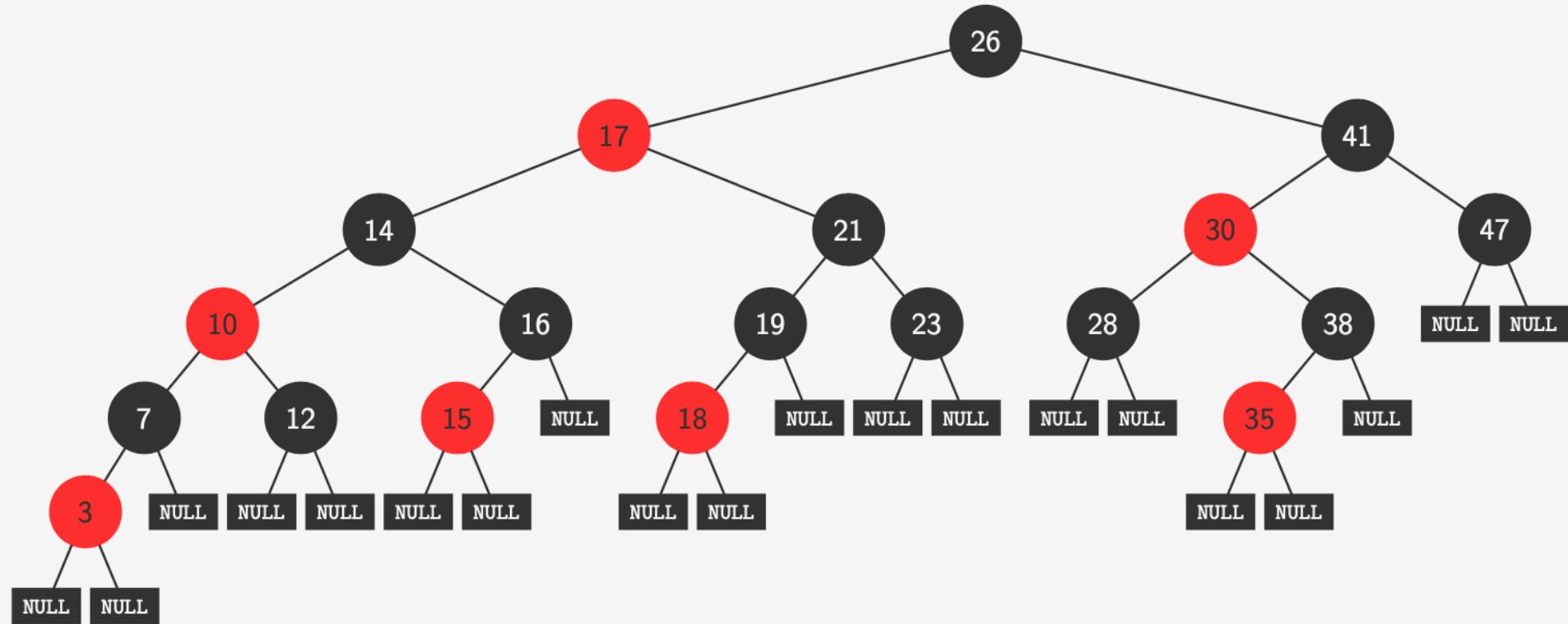
Árvores Rubro-Negras

- Usadas nativamente como árvore padrão
 - C++
 - JAVA
 - Kernel do Linux

Árvores Rubro-Negras

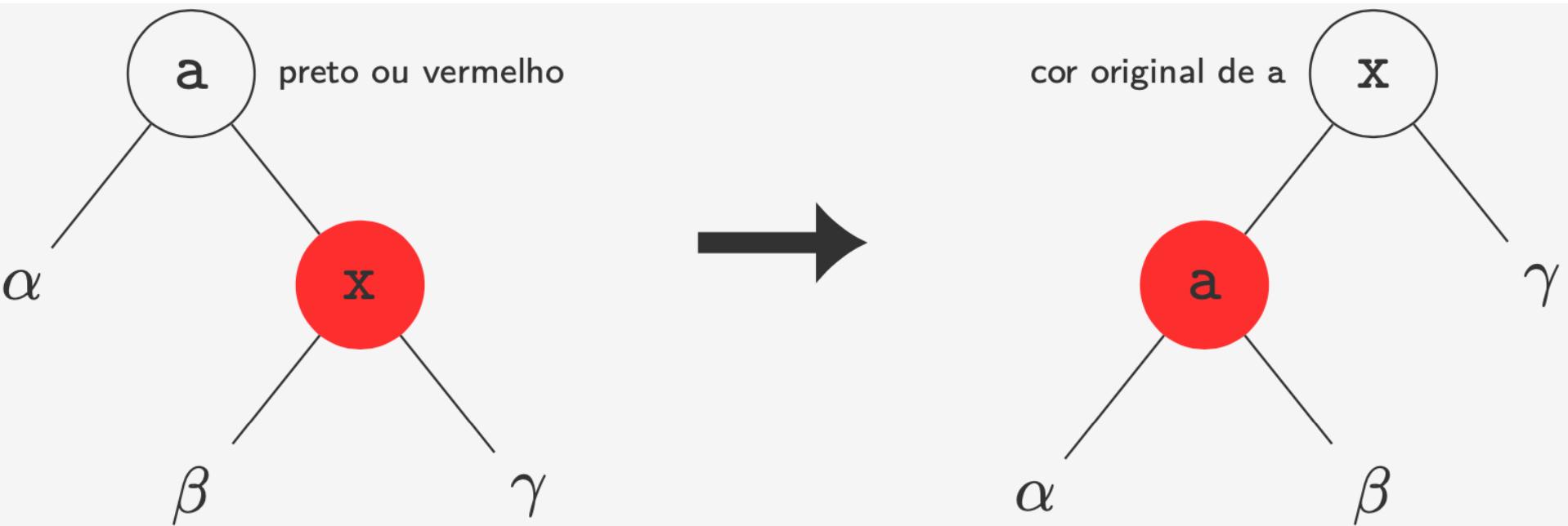
- Uma árvore rubro-negra esquerdista é uma árvore binária de busca tal que
 - Todo nó é ou vermelho ou preto
 - A raiz é preta
 - As folhas são NULL e tem cor preta
 - Se um nó é vermelho, seus dois filhos são pretos
 - E ele é o filho esquerdo do seu pai (por isso, esquerdista)
 - Em cada nó, todo caminho dele para uma de suas folhas descendentes tem a mesma quantidade de nós pretos
 - Não se conta o próprio nó
 - Denomina-se a altura-negra do nó

Árvores Rubro-Negras



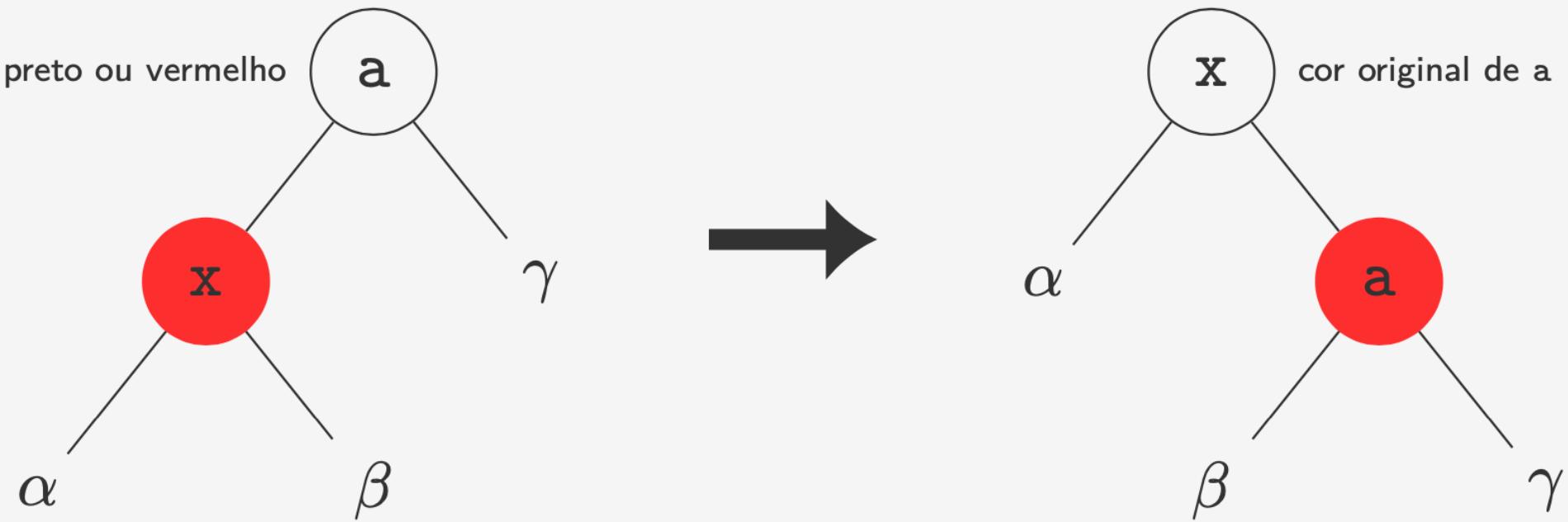
Árvores Rubro-Negras

- Rotação à esquerda



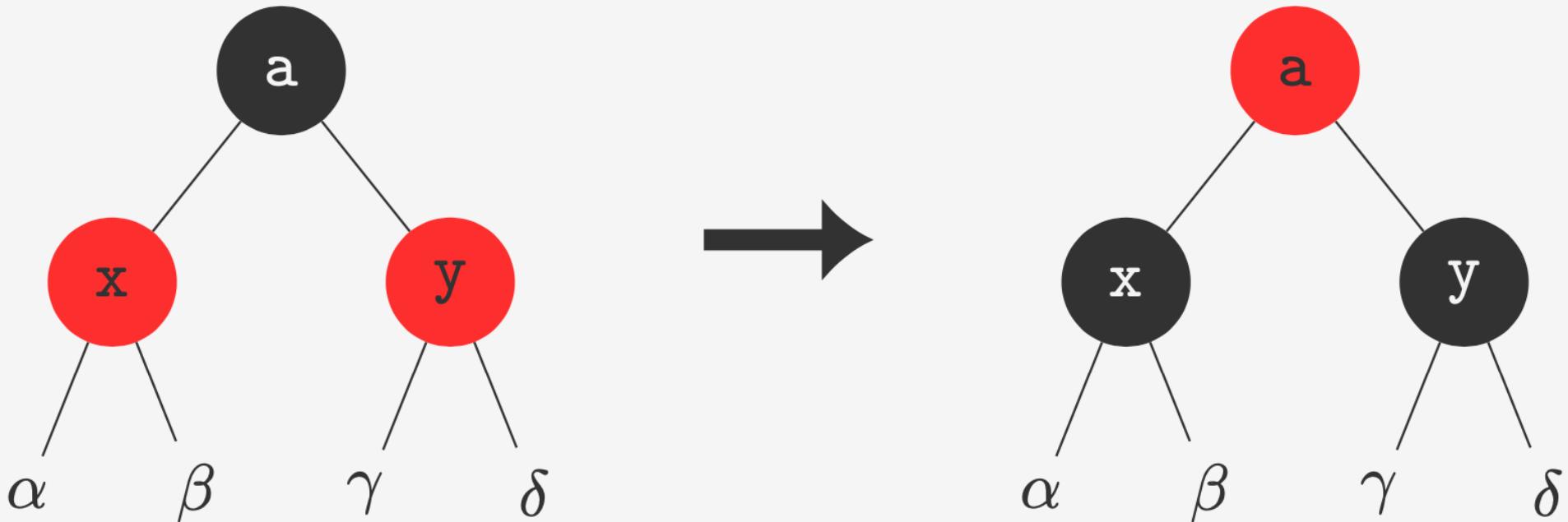
Árvores Rubro-Negras

- Rotação à direita



Árvores Rubro-Negras

- Subida do vermelho

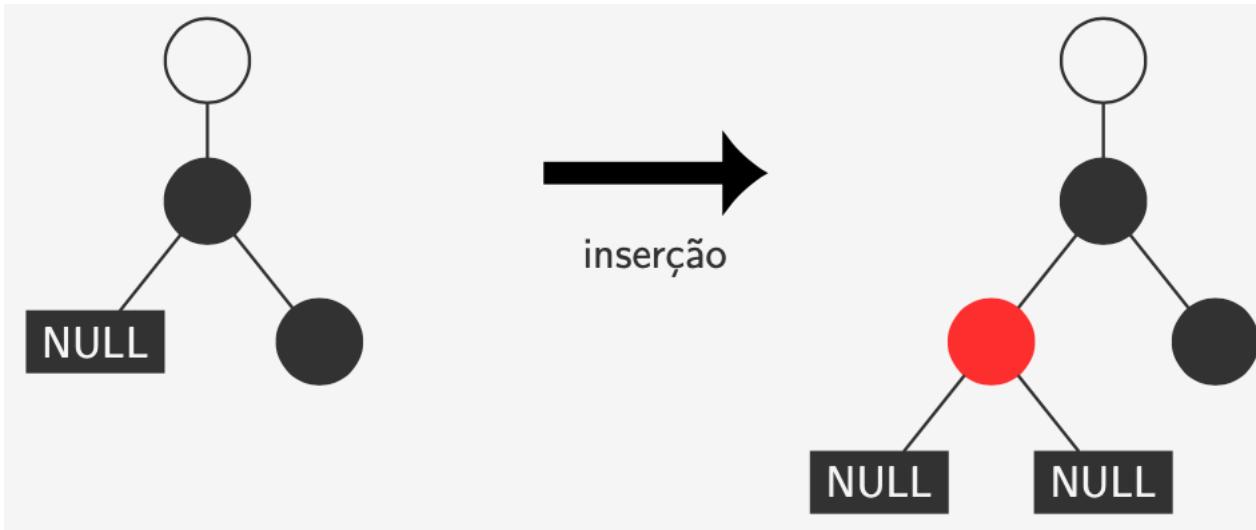


Árvores Rubro-Negras

- Inserção de elemento
 - Inserção exatamente igual a uma árvore de busca
 - Novo nó sempre vermelho
 - No entanto, correções precisam ser feitas
 - NÃO pode filho direito vermelho e filho esquerdo preto
 - Rotacionar à esquerda
 - NÃO pode neto esquerdo vermelho de filho vermelho
 - Rotacionar à direita
 - NÃO pode ambos filhos vermelhos
 - Subir vermelho

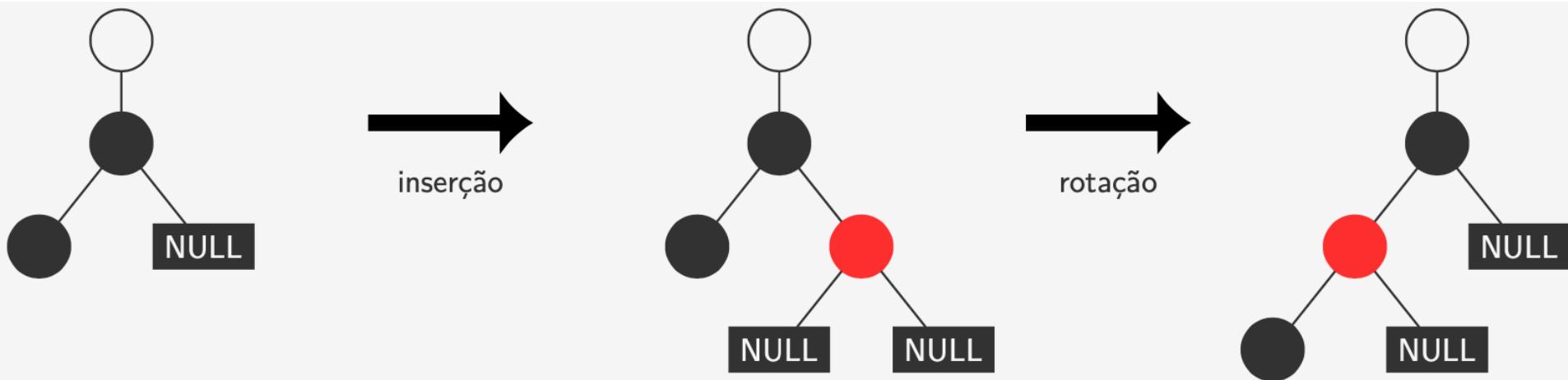
Árvores Rubro-Negras

- Inserção de elemento
 - Caso 1: nó preto, filho direito preto



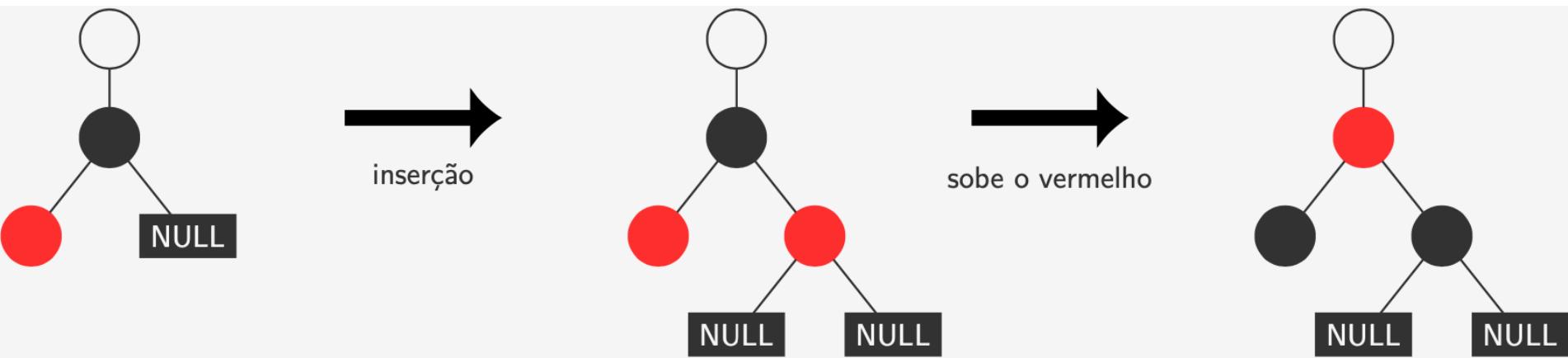
Árvores Rubro-Negras

- Inserção de elemento
 - Caso 2: nó preto, filho esquerdo preto



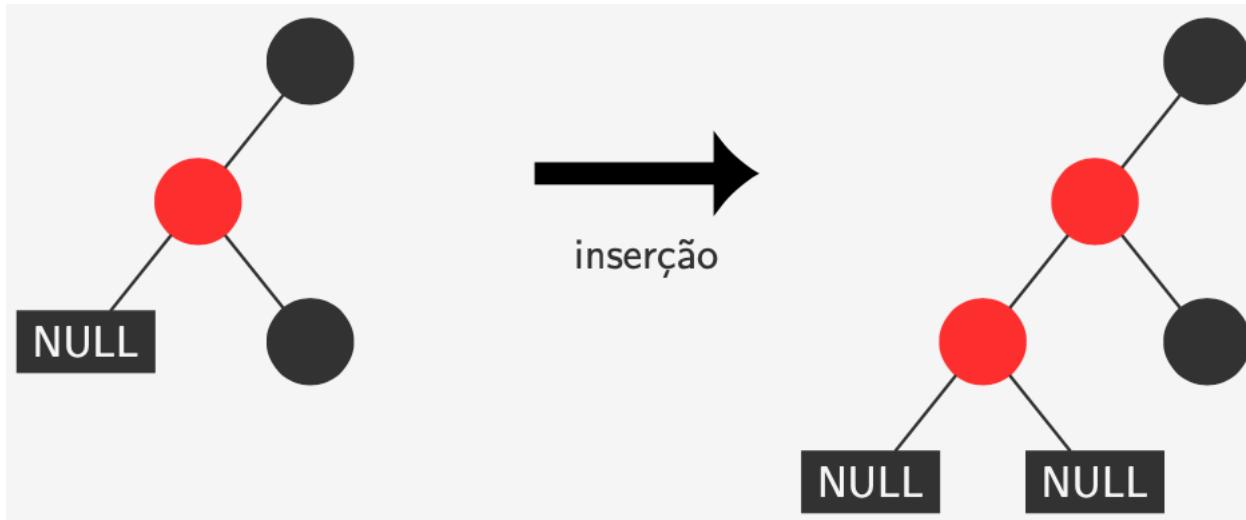
Árvores Rubro-Negras

- Inserção de elemento
 - Caso 3: nó preto, filho esquerdo vermelho



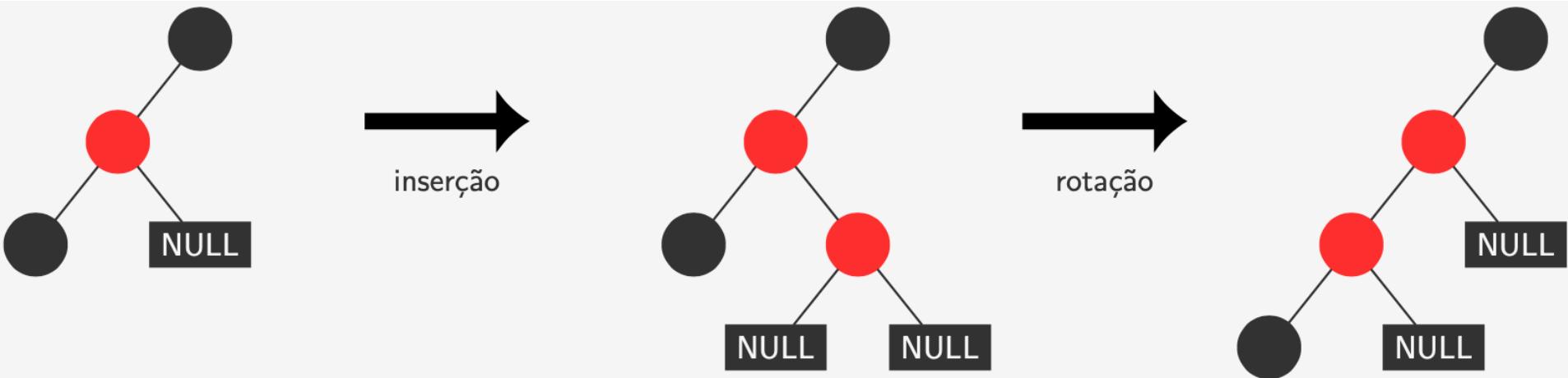
Árvores Rubro-Negras

- Inserção de elemento
 - Caso 4: nó vermelho, filho direito preto



Árvores Rubro-Negras

- Inserção de elemento
 - Caso 5: nó vermelho, filho esquerdo preto



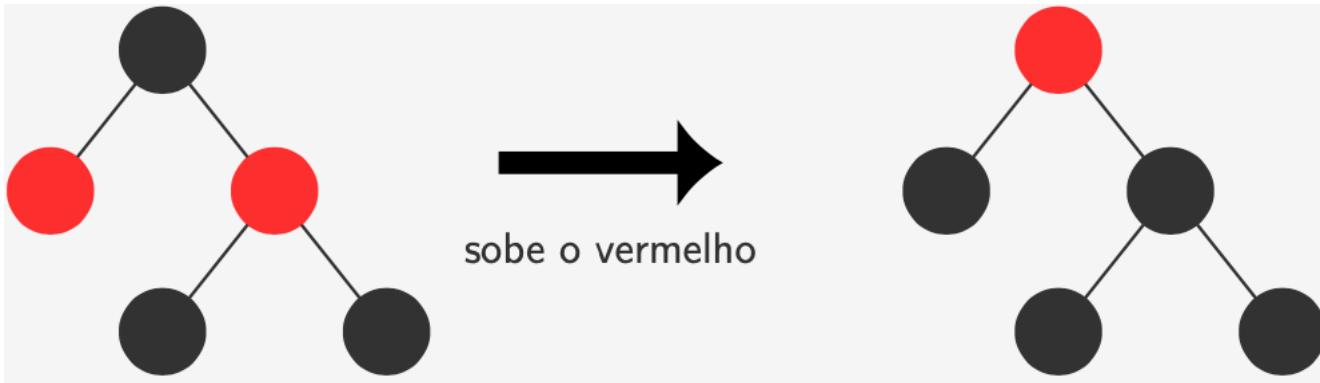
Árvores Rubro-Negras

- Inserção de elemento
 - Correções no pai
 - Caso 1: filho esquerdo preto, filho direito vermelho



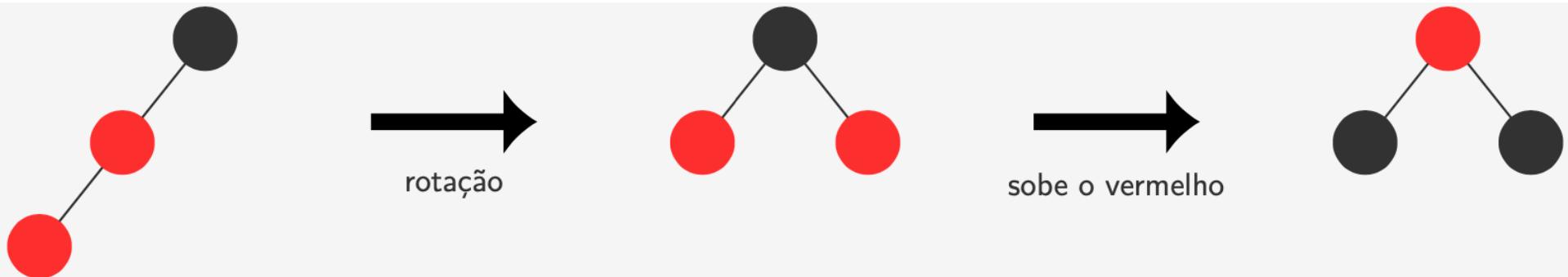
Árvores Rubro-Negras

- Inserção de elemento
 - Correções no pai
 - Caso 2: filho esquerdo e filho direito vermelhos



Árvores Rubro-Negras

- Inserção de elemento
 - Correções no pai
 - Caso 3: filho esquerdo e neto à esquerda vermelhos



Árvores Rubro-Negras

```
1 p_no rotaciona_para_esquerda(p_no raiz) {
2     p_no x = raiz->dir;
3     raiz->dir = x->esq;
4     x->esq = raiz;
5     x->cor = raiz->cor;
6     raiz->cor = VERMELHO;
7     return x;
1 p_no rotaciona_para_direita(p_no raiz) {
2     p_no x = raiz->esq;
3     raiz->esq = x->dir;
4     x->dir = raiz;
5     x->cor = raiz->cor;
6     raiz->cor = VERMELHO;
7     return x;
8 }
1 void sobe_vermelho(p_no raiz) {
2     raiz->cor = VERMELHO;
3     raiz->esq->cor = PRETO;
4     raiz->dir->cor = PRETO;
5 }
```

Árvores Rubro-Negras

```
1 p_no inserir_rec(p_no raiz, int chave) {
2     p_no novo;
3     if (raiz == NULL) {
4         novo = malloc(sizeof(No));
5         novo->esq = novo->dir = NULL;
6         novo->chave = chave;
7         novo->cor = VERMELHO;
8         return novo;
9     }
10    if (chave < raiz->chave)
11        raiz->esq = inserir_rec(raiz->esq, chave);
12    else
13        raiz->dir = inserir_rec(raiz->dir, chave);
14    /* corrige a árvore */
15    if (ehVermelho(raiz->dir) && ehPreto(raiz->esq))
16        raiz = rotaciona_para_esquerda(raiz);
17    if (ehVermelho(raiz->esq) && ehVermelho(raiz->esq->esq))
18        raiz = rotaciona_para_direita(raiz);
19    if (ehVermelho(raiz->esq) && ehVermelho(raiz->dir))
20        sobe_vermelho(raiz);
21    return raiz;
22 }
```

Árvores Rubro-Negras

- Programa construção da árvore rubro-negra
 - Busca de elemento
 - Inserção de elemento
 - Navegação ordenada

Árvores AVL

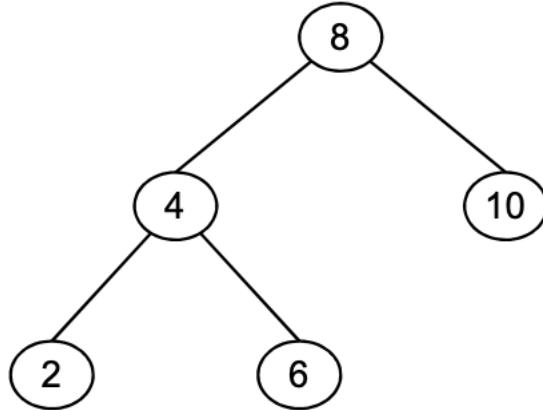
Árvores AVL

- Uma árvore binária é denominada AVL quando a diferença de altura entre as subárvores esquerda (sae) e direita (sad) de um nó qualquer não é superior a 1
- Em 1962 os matemáticos russos G.M. Adelson-Velskki e E.M.Landis sugeriram uma definição para “near balance” e descreveram procedimentos para inserção e eliminação de nós nessas árvores.
- Esses algoritmos de balanceamento de árvore são chamados algoritmos AVL

Árvores AVL

- Quando um novo nó é inserido em uma árvore balanceada podem ocorrer três situações, em função da configuração atual da árvore
 - Se $h(\text{sae}) == h(\text{sad})$, após a inserção as alturas vão ser diferentes, porém a árvore continua balanceada
 - se $h(\text{sae}) > h(\text{sad})$ e o nó for inserido na sad, as alturas ficam iguais e o balanceamento foi melhorado
 - se $h(\text{sae}) > h(\text{sad})$ e o nó for inserido na sae, o balanceamento fica violado

Árvores AVL



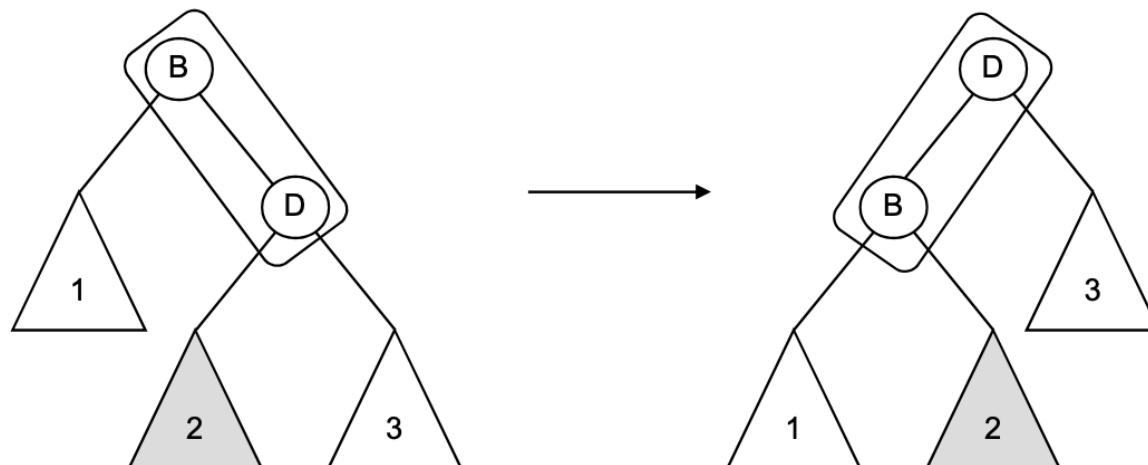
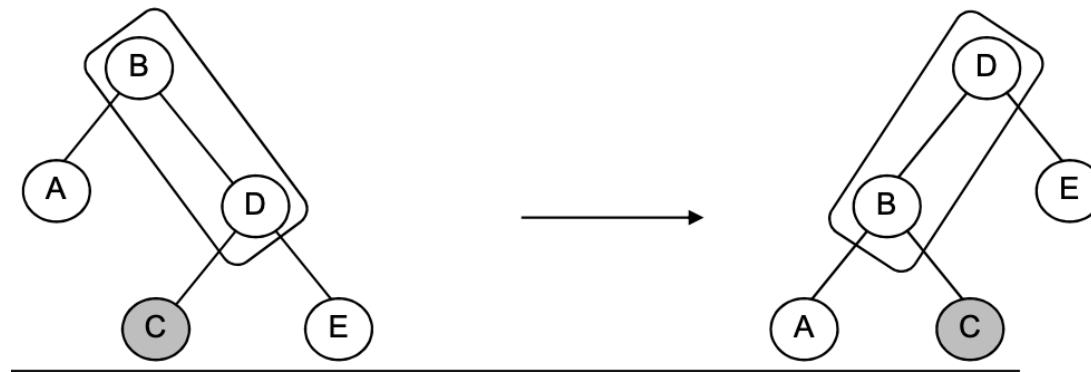
- Os nós 9 e 11 poderiam ser inseridos sem desbalancear a árvore
- Já os nós 1, 3, 5 e 7 gerariam desbalanceamento

Árvores AVL

- Rotações para rebalancear uma árvore AVL
 - Rotação à esquerda
 - Rotação à direita
 - Dupla rotação à esquerda (Esquerda + direita)
 - Dupla rotação à direita (Direita + Esquerda)
- Com apenas uma dessas rotações, após a inserção de um novo item, a árvore volta a ser balanceada

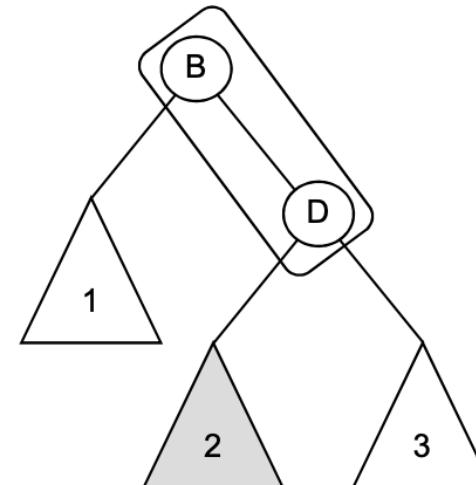
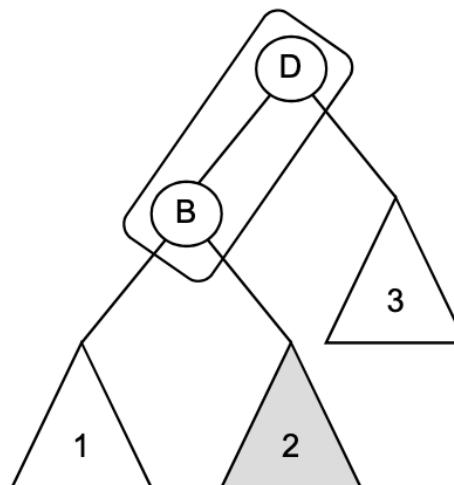
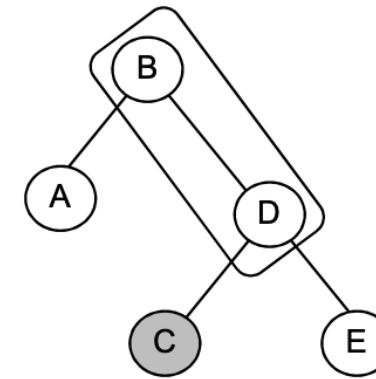
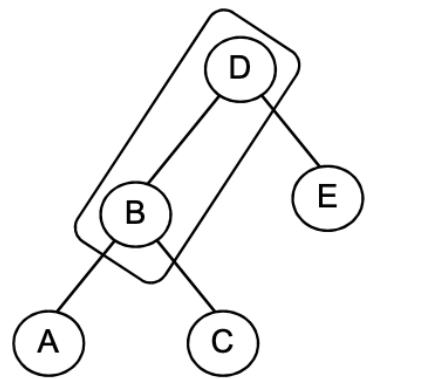
Árvores AVL

- Rotação à esquerda



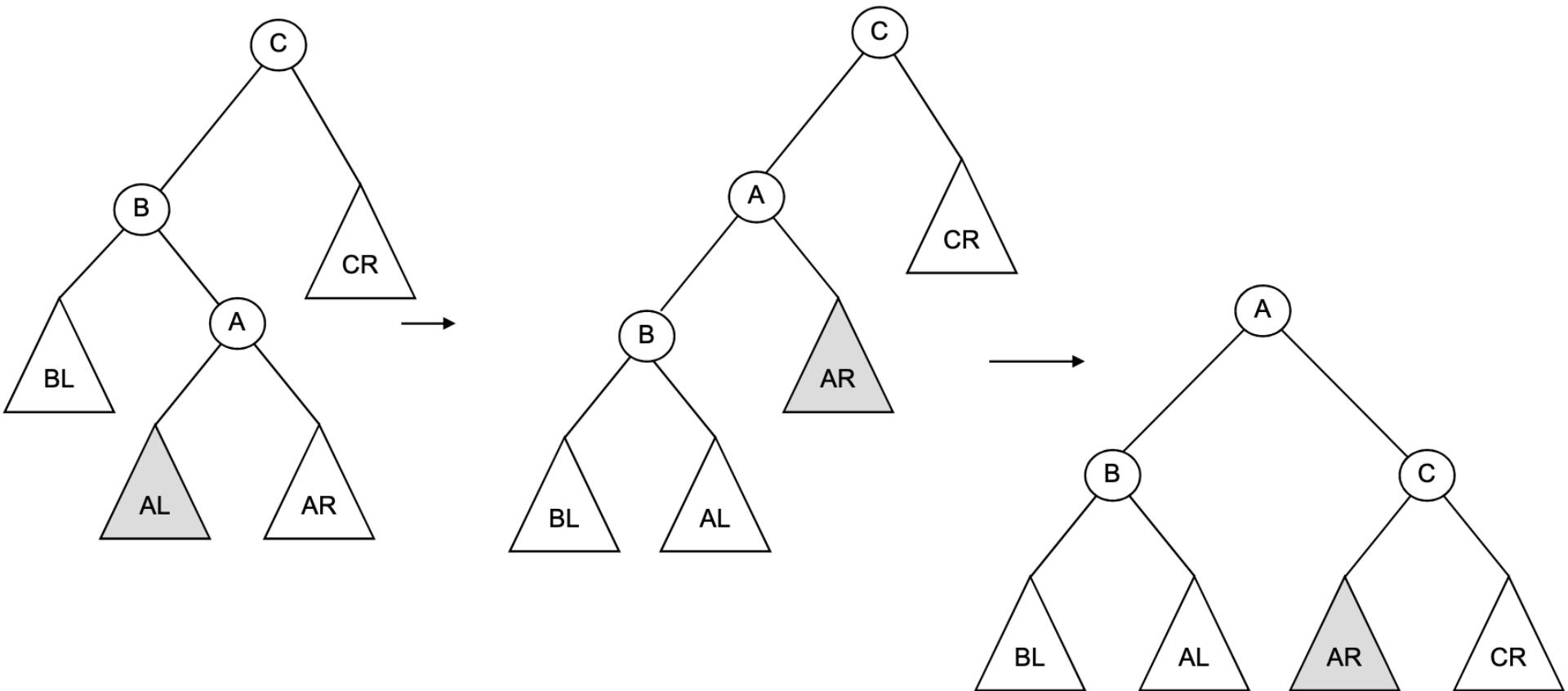
Árvores AVL

- Rotação à direita



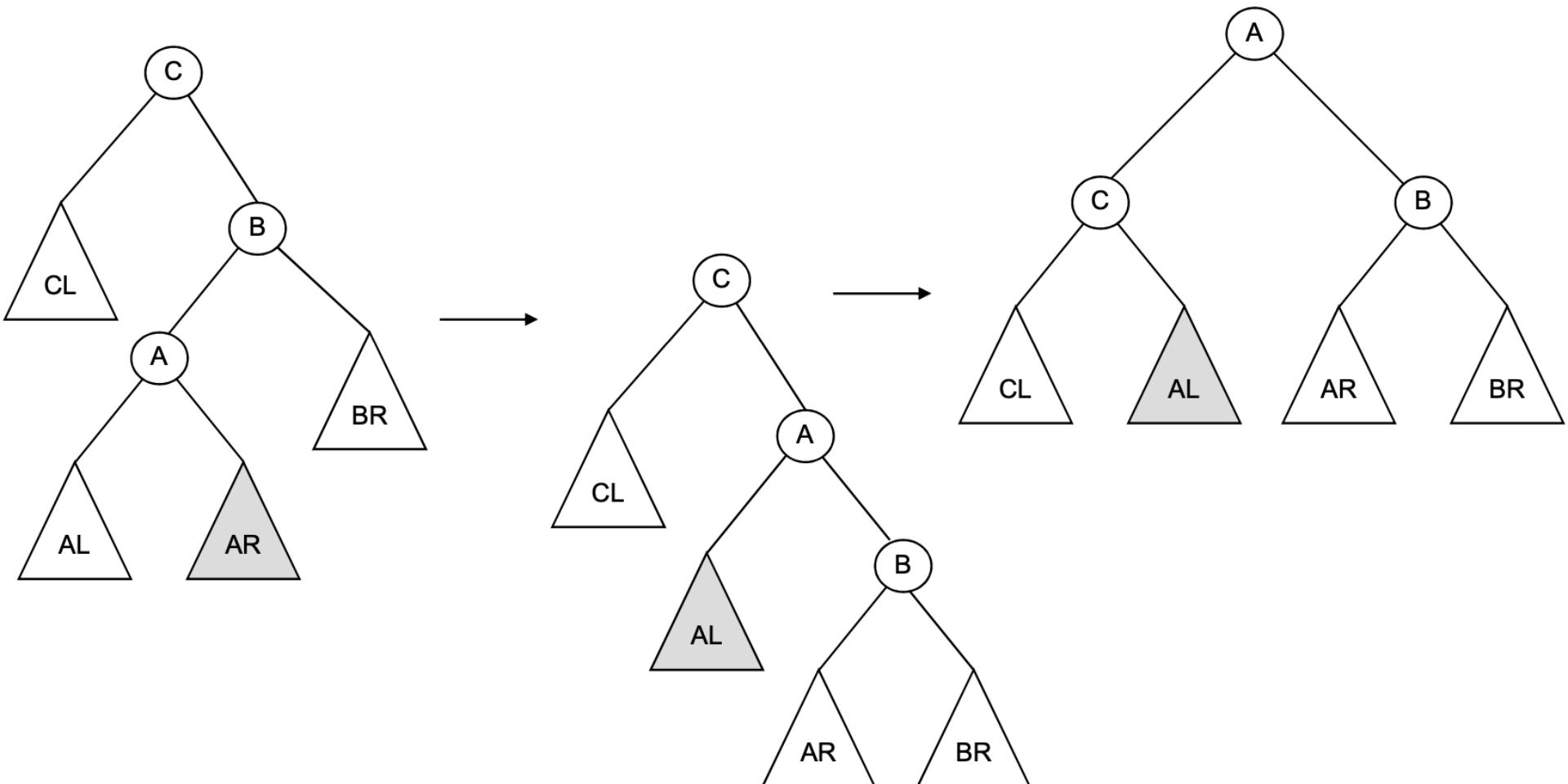
Árvores AVL

- Dupla rotação à esquerda



Árvores AVL

- Dupla rotação à direita



Árvores AVL

```
int getAlturaDoNo(struct no *raiz) {  
    int retorno = 0;  
  
    if (raiz != NULL) {  
        retorno = raiz -> altura;  
    }  
  
    return retorno;  
}  
  
int getMaximoEntreDoisNumeros(int a, int b) {  
    return (a > b) ? a : b;  
}  
  
int getBalanco(struct no *raiz) {  
    int retorno = 0;  
  
    if (raiz != NULL) {  
        retorno = (getAlturaDoNo(raiz -> esquerdo) - getAlturaDoNo(raiz -> direito));  
    }  
  
    return retorno;  
}
```

Árvores AVL

```
struct no *rotacaoAEsquerda(struct no *raiz) {
    struct no *novaRaiz = raiz -> direito;
    struct no *temp = novaRaiz -> esquerdo;

    novaRaiz -> esquerdo = raiz;
    raiz -> direito = temp;

    raiz -> altura = (getMaximoEntreDoisNumeros(getAlturaDoNo(raiz -> esquerdo), getAlturaDoNo(raiz -> direito)) + 1);
    novaRaiz -> altura = (getMaximoEntreDoisNumeros(getAlturaDoNo(novaRaiz -> esquerdo), getAlturaDoNo(novaRaiz -> direito)) + 1);

    return novaRaiz;
}

struct no *rotacaoADireita(struct no *raiz) {
    struct no *novaRaiz = raiz -> esquerdo;
    struct no *temp = novaRaiz -> direito;

    novaRaiz -> direito = raiz;
    raiz -> esquerdo = temp;

    raiz -> altura = (getMaximoEntreDoisNumeros(getAlturaDoNo(raiz -> esquerdo), getAlturaDoNo(raiz -> direito)) + 1);
    novaRaiz -> altura = (getMaximoEntreDoisNumeros(getAlturaDoNo(novaRaiz -> esquerdo), getAlturaDoNo(novaRaiz -> direito)) + 1);

    return novaRaiz;
}
```

Árvores AVL

```
struct no *inserir(struct no *raiz, int numero) {
    if (raiz == NULL) {
        raiz = (struct no *) malloc(sizeof(struct no));
        raiz -> numero = numero;
        raiz -> altura = 1;
        raiz -> esquerdo = NULL;
        raiz -> direito = NULL;
    } else if (raiz -> numero > numero) {
        raiz -> esquerdo = inserir(raiz -> esquerdo, numero);
    } else if (raiz -> numero < numero) {
        raiz -> direito = inserir(raiz -> direito, numero);
    }

    raiz -> altura = (1 + getMaximoEntreDoisNumeros(getAlturaDoNo(raiz -> esquerdo), getAlturaDoNo(raiz -> direito)));

    int balanco = getBalanco(raiz);

    if ((balanco > 1) && (numero < raiz -> esquerdo -> numero)) {
        raiz = rotacaoADireita(raiz);
    }

    if ((balanco < -1) && (numero > raiz -> direito -> numero)) {
        raiz = rotacaoAEsquerda(raiz);
    }

    if ((balanco > 1) && (numero > raiz -> esquerdo -> numero)) {
        raiz -> esquerdo = rotacaoAEsquerda(raiz -> esquerdo);
        raiz = rotacaoADireita(raiz);
    }

    if ((balanco < -1) && (numero < raiz -> direito -> numero)) {
        raiz -> direito = rotacaoADireita(raiz -> direito);
        raiz = rotacaoAEsquerda(raiz);
    }

    return raiz;
}
```

Árvores AVL

- Programa construção da árvore AVL
 - Busca de elemento
 - Inserção de elemento
 - Navegação ordenada

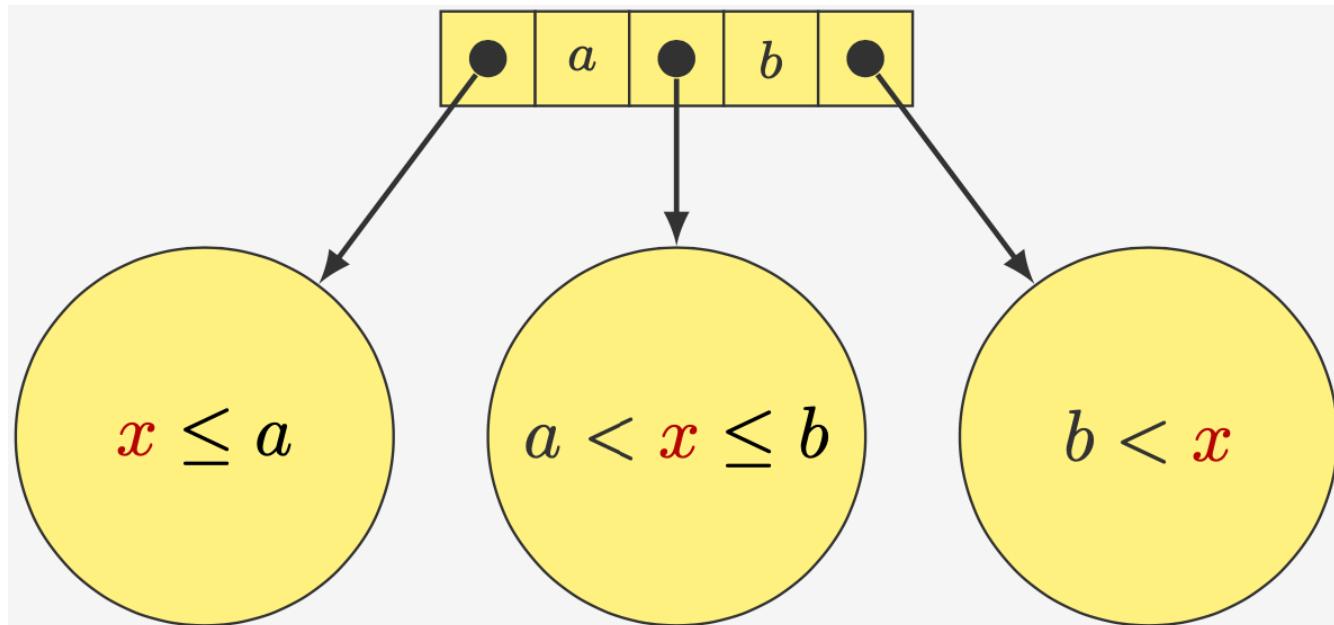
Árvores B



Árvores B

- Árvores utilizadas para armazenamento muito volumétrico
 - Bancos de dados
- Tentativa de diminuir a altura da árvore para diminuir número de leituras na memória secundária
- Árvores B são normalmente muito largas e muito baixas

- Árvores M-árias de busca
 - É possível generalizar árvores binárias de busca
 - Exemplo: árvore ternária de busca
 - Nó pode ter 0, 1, 2 ou 3 filhos



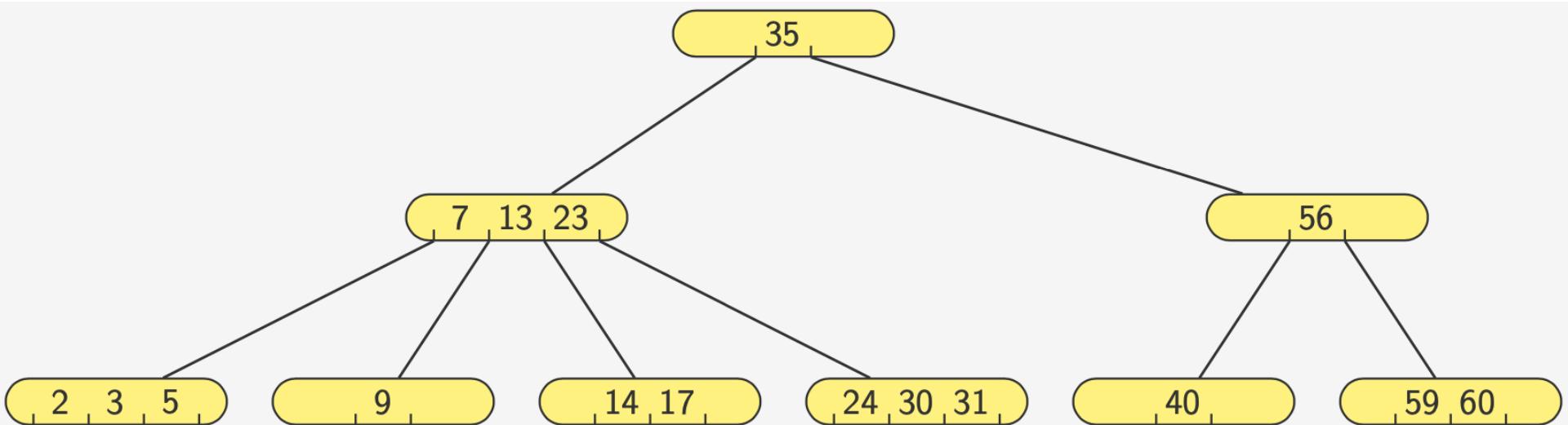
Árvores B

- Árvores M-árias de busca com propriedades adicionais
 - Cada nó tem os seguintes campos
 - n é o número de chaves armazenadas
 - $\text{chave}[i]$ é a i -ésima chave armazenada
 - $\text{chave}[1] < \text{chave}[2] < \dots < \text{chave}[n]$
 - folha é um booleano que indica se o nó é uma folha ou não
 - $n + 1$ ponteiros
 - $c[i]$ é o ponteiro para o i -ésimo filho
 - se a chave k está na subárvore $c[i]$, então
 - » $k < \text{chave}[1]$ se $i == 1$
 - » $k > \text{chave}[n]$ se $i == n + 1$
 - » senão $\text{chave}[i - 1] < k < \text{chave}[i]$
 - Raiz indica o nó que é a raiz da árvore

- Árvores M-árias de busca com propriedades adicionais
 - Toda folha está à mesma distância h (altura) da raiz
 - Constante t é o grau mínimo da árvore
 - Todo nó, exceto a raiz, precisa ter pelo menos $t - 1$ chaves
 - Cada nó interno tem pelo menos t filhos
 - Todo nó tem no máximo $2*t - 1$ chaves
 - Cada nó interno tem no máximo $2*t$ filhos

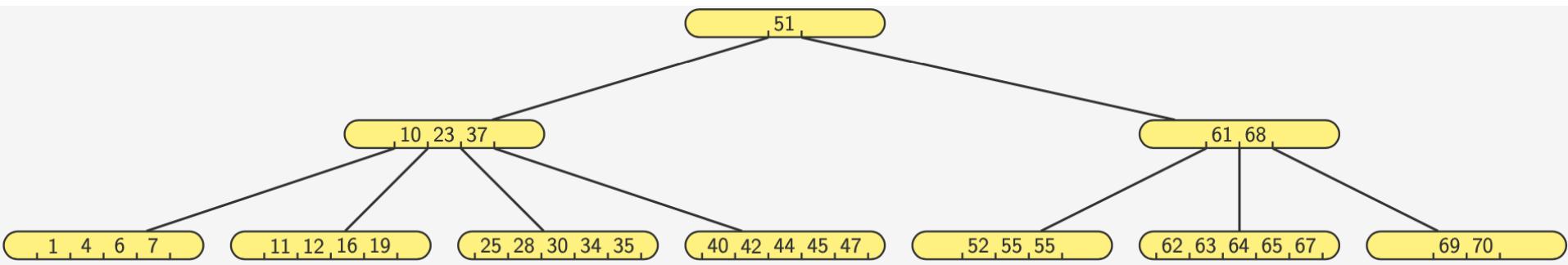
Árvores B

- Para $t = 2$
 - Cada nó não raiz tem pelo menos 1 registro
 - Cada nó tem no máximo 3 registros



Árvores B

- Para $t = 3$
 - Cada nó não raiz tem pelo menos 2 registros
 - Cada nó tem no máximo 5 registros



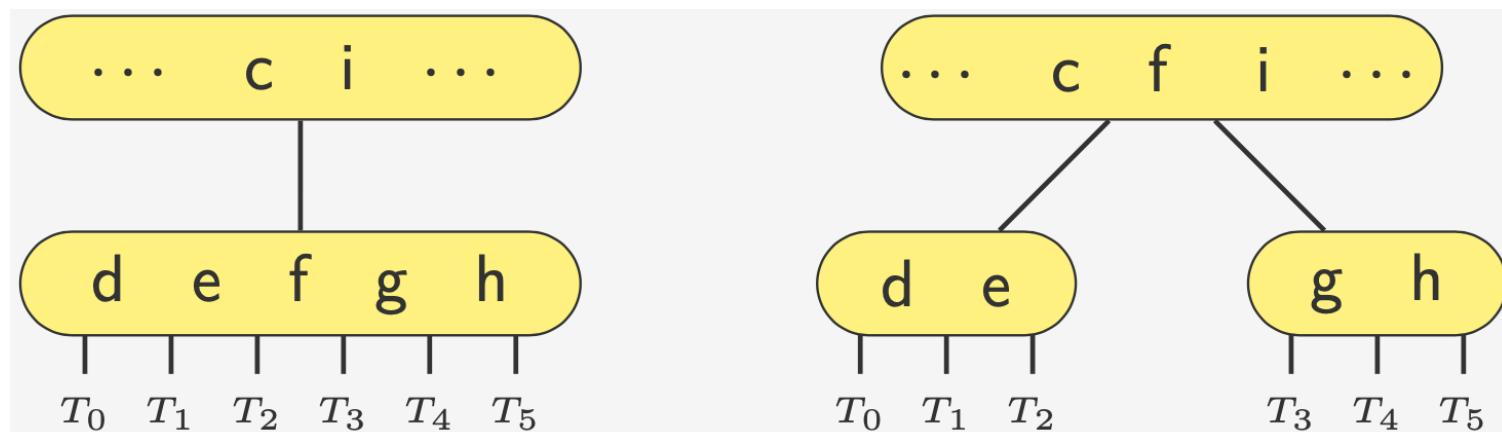
Árvores B

- Uma árvore B com n chaves tem altura
 - $h < \log(n+1)/2$ na base t
 - Se $t = 1001$ e $h = 2$
 - Armazenam-se até 10^9 chaves
 - Acessíveis com míseros 2 acessos a disco

Árvores B

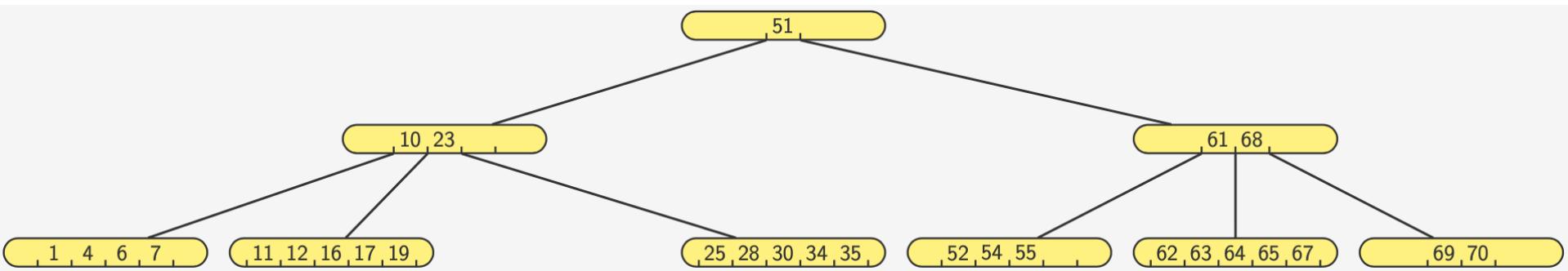
- Busca de elemento
 - Procedimento análogo às árvores de busca

- Inserção de elemento
 - Sempre nas folhas
 - Se a folha estiver cheia
 - Dividir na metade
 - Subir o elemento central para o pai
 - Criar duas folhas com cada metade



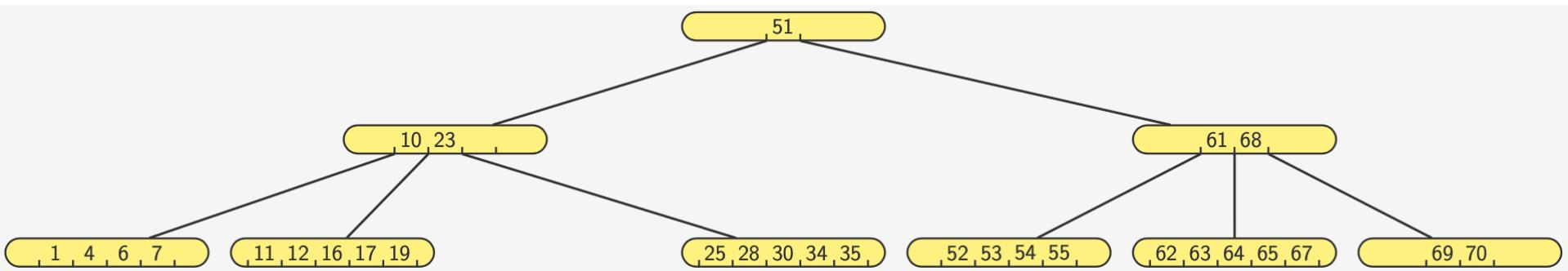
Árvores B

- Inserir elemento 53



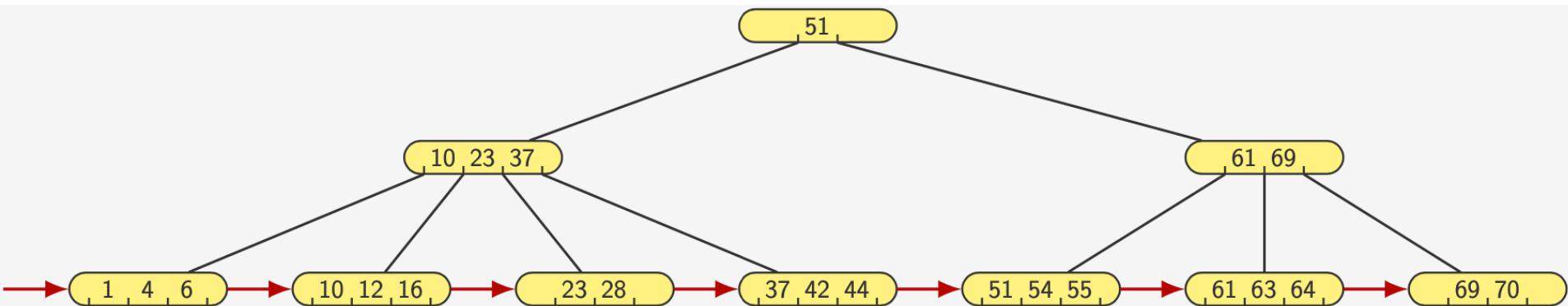
Árvores B

- Inserir elemento 18



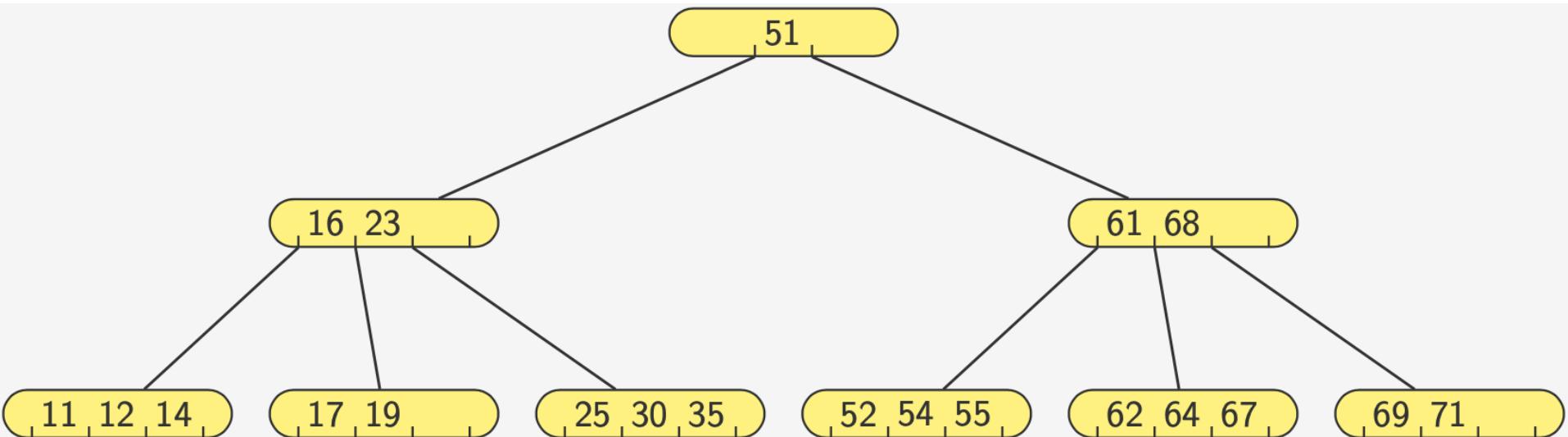
Árvores B

- Árvores B+
 - Mantêm cópias das chaves nos nós internos
 - Chaves e registros são armazenados nas folhas
 - Permitem acesso sequencial dos dados



Árvores B

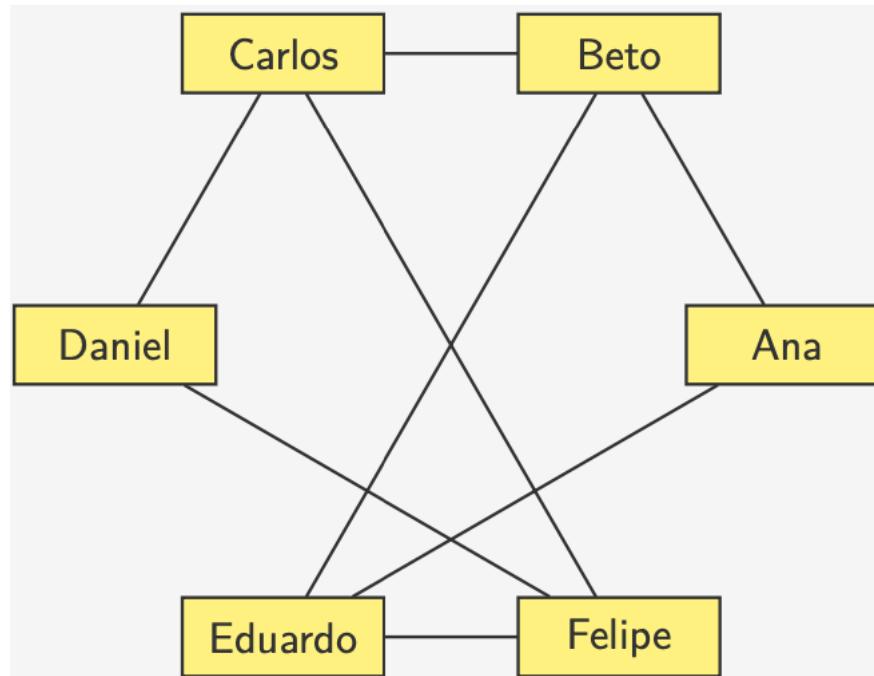
- Inserir elemento 13
- Inserir elemento 33



Grafos

Grafos

- Como representar amizades em uma rede social ?

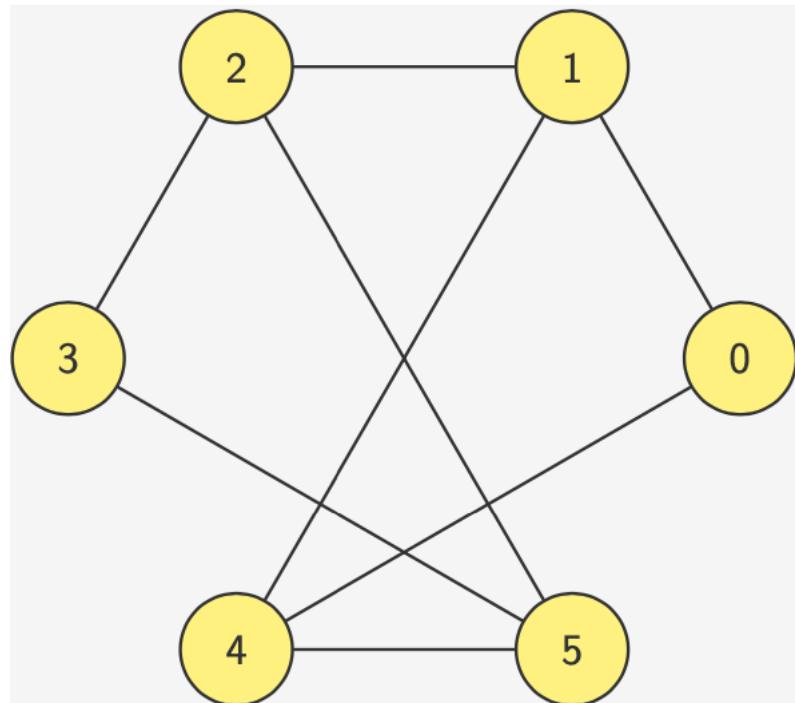


Grafos

- Um Grafo é um conjunto de objetos ligados entre si
 - Os objetos denominam-se **vértices**
 - As conexões entre os objetos denominam-se **arestas**
- Um grafo é visualmente representado
 - Com os vértices representados por pontos
 - Com as arestas representadas por curvas ligando dois vértices

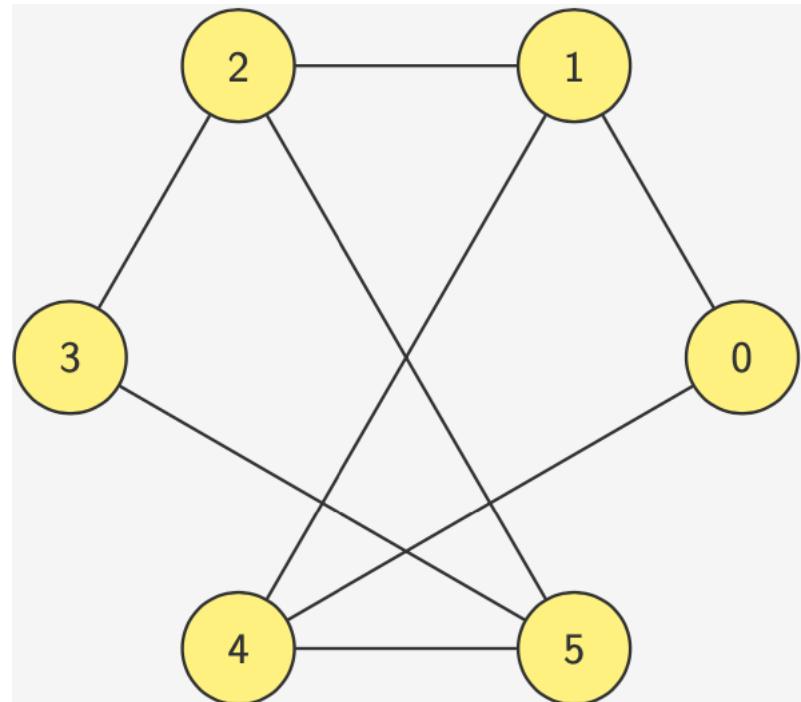
Grafos

- Formalmente, um grafo G é um par ordenado (V, E)
 - V é o conjunto de vértices do grafo
 - $V = \{0, 1, 2, 3, 4, 5\}$
 - E é o conjunto de arestas do grafo
 - Representa-se uma aresta ligando $u, v \in V$ como $\{u, v\}$
 - Para toda aresta $\{u, v\}$ em E , temos que $u \neq v$
 - Existe no máximo uma aresta $\{u, v\}$ em E
 - $E = \{\{0,1\}, \{0,4\}, \{5,3\}, \{1,2\}, \{2,5\}, \{4,5\}, \{3,2\}, \{1,4\}\}$



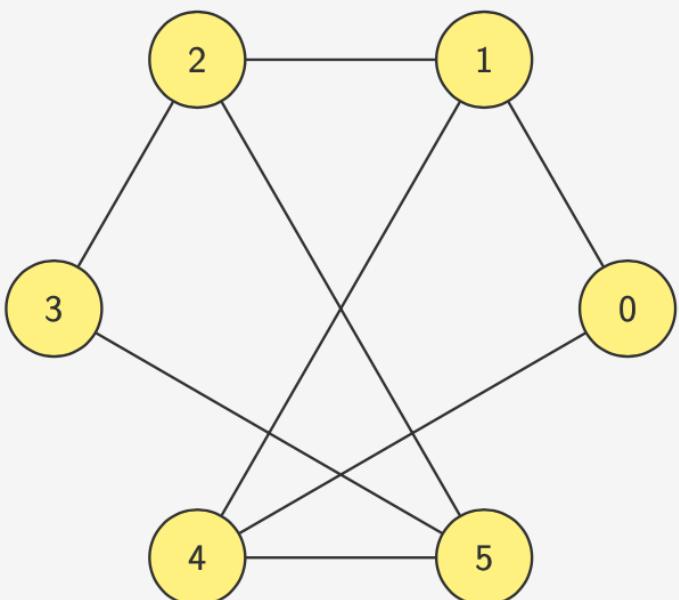
Grafos

- O vértice 0 é vizinho do vértice 4
 - 0 e 4 são adjacentes
- Os vértices 0, 1 e 5 são a vizinhança do vértice 4



Grafos

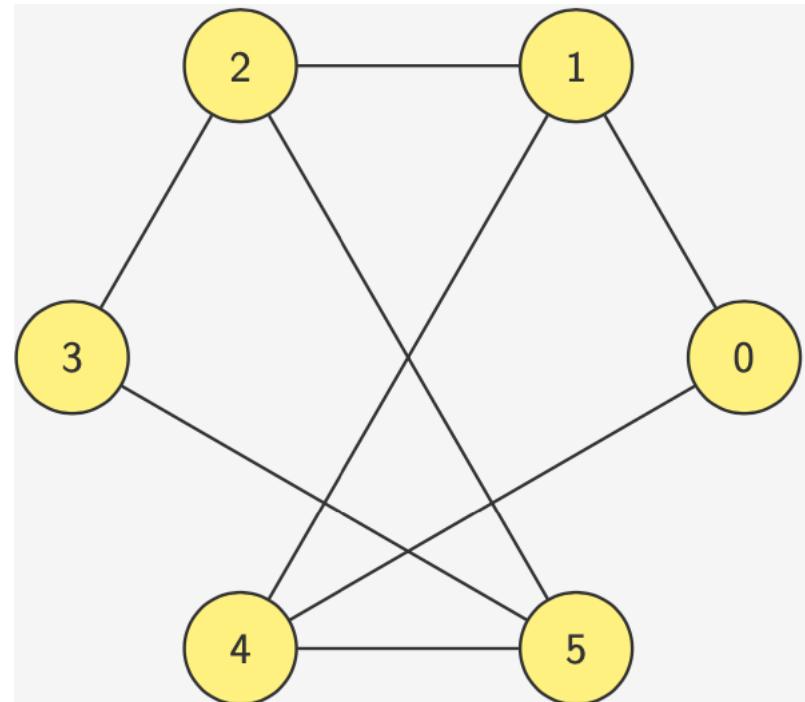
- Matriz de adjacências



	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	1	1	0	0	0	1
5	0	0	1	1	1	0

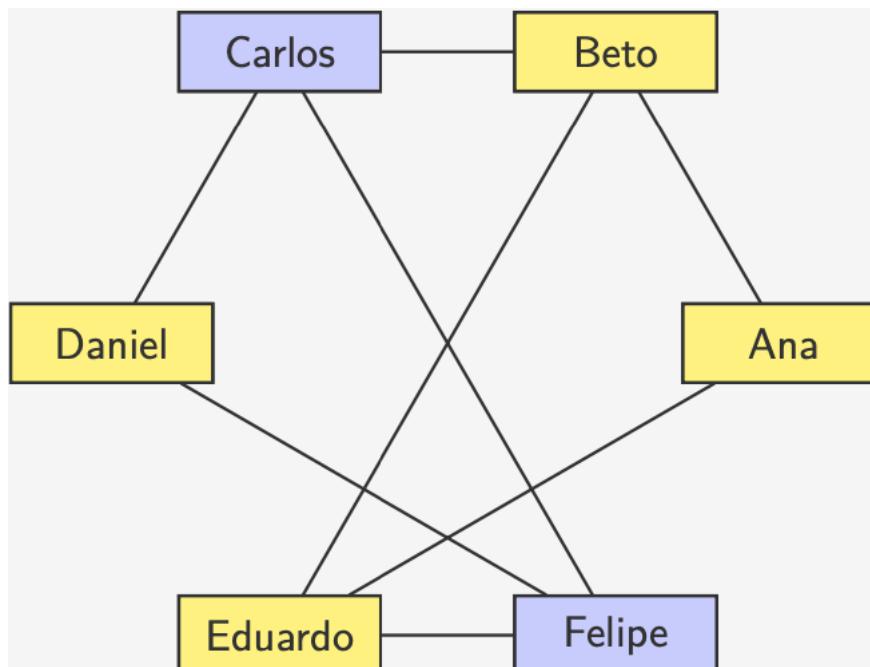
Grafos

- Grau de um vértice é o número de vizinhos



Grafos

- Recomendação de amigos
 - Quem são os amigos dos amigos da Ana ?

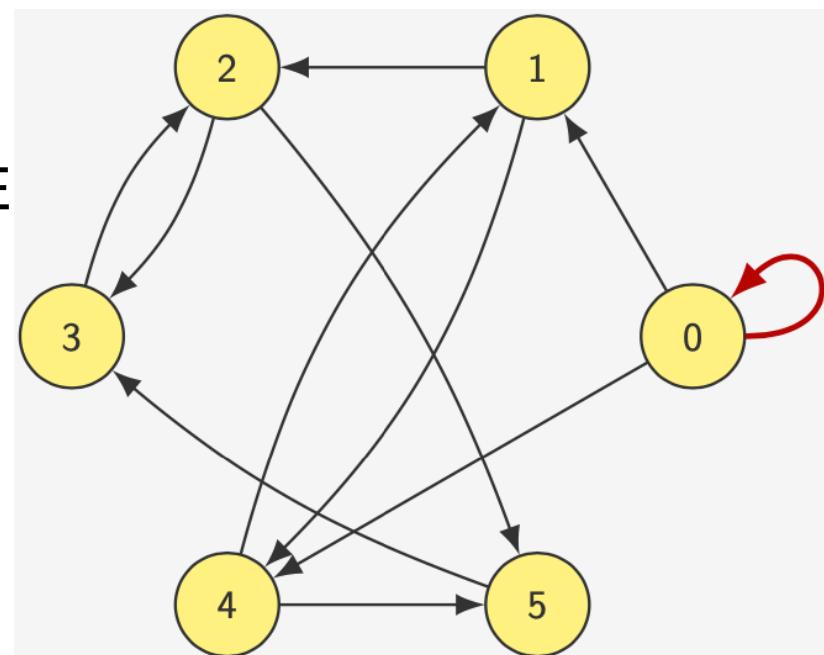


Grafos

- Dígrafo
 - Grafo dirigido
 - Conjunto de vértices
 - Conectados por meio de um conjunto de arcos
 - Arestras dirigidas, indicando início e fim
 - Um dígrafo é visualmente representado
 - Com os vértices representados por pontos
 - Com os arcos representados por curvas com uma seta na ponta ligando dois vértices

Grafos

- Formalmente, um digrafo G é um par ordenado (V, A)
 - V é o conjunto de vértices do grafo
 - A é o conjunto de arcos do grafo
 - Representa-se um arco ligando $u, v \in V$ como $\{u, v\}$
 - u é a calda ou origem
 - v é a cabeça ou destino
 - Pode haver um laço $\{u, u\}$ em E
 - Existe no máximo um arco $\{u, v\}$ em A



Grafos

- Número máximo de arestas de um grafo
 - n vértices
 - $n * (n - 1) / 2$
 - $O(n^2)$

Grafos

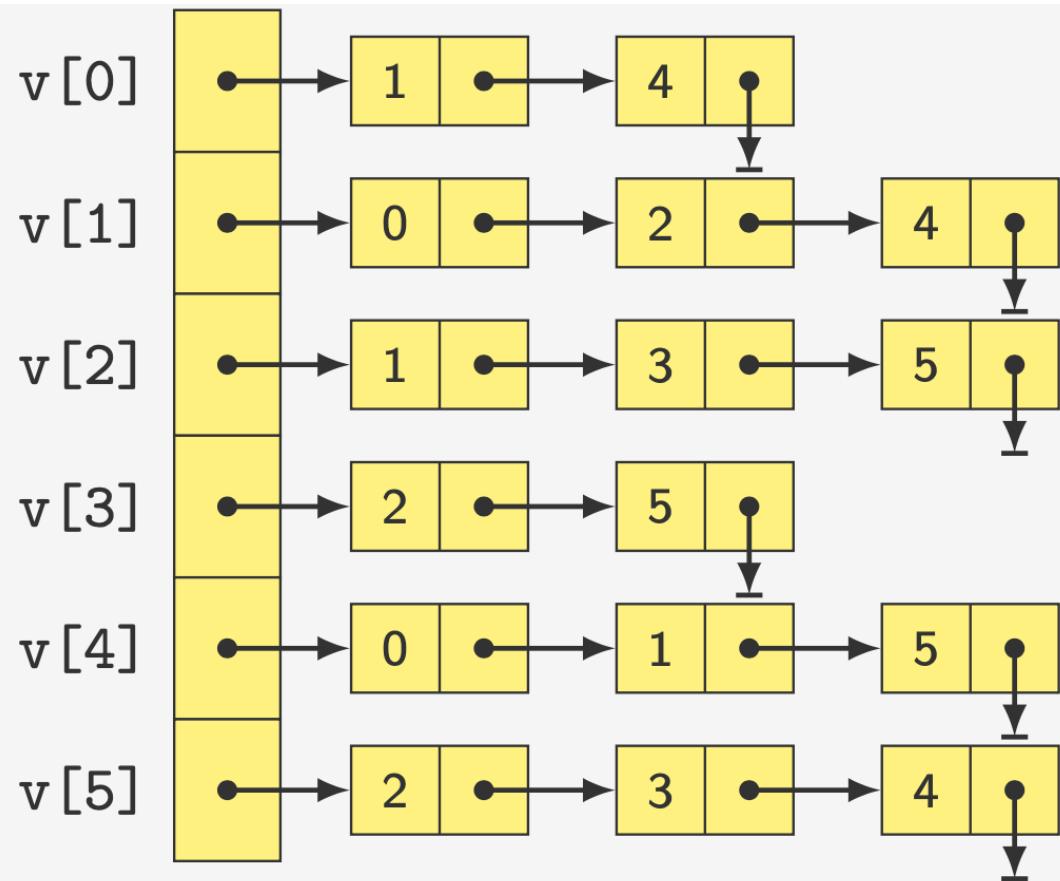
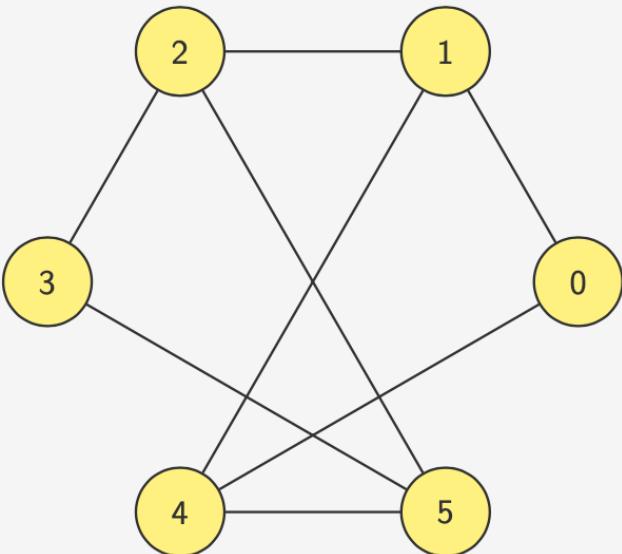
- Grafos esparsos
 - Um grafo pode ter no máximo $n * (n - 1) / 2$ arestas, se todos os vértices estiverem conectados a todos os outros
 - Facebook tem 2,2 bilhões de usuários ativos/mês
 - Uma matriz de adjacências teria $4,84 * 10^{18}$ posições
 - 605 petabytes (usando um mísero bit por posição)
 - Verificar se duas pessoas são amigas leva $O(1)$
 - Supondo uma memória capaz de suportar esse volume
 - Imprimir todos os amigos de uma pessoa leva $O(n)$
 - É necessário percorrer 2,2 bilhões de posições
 - Um usuário comum tem bem menos amigos do que isso
 - Facebook coloca um limite de 5000 amigos

Grafos

- Grafos esparsos
 - Um grafo é dito esparso se a quantidade de arestas for assintoticamente menor do que a quantidade máxima de arestas (ordem de grandeza menor)
 - Facebook
 - Cada usuário tem no máximo 5000 amigos
 - O máximo de arestas é $5,5 * 10^{12}$
 - Bem menos do que $4,84 * 10^{18}$

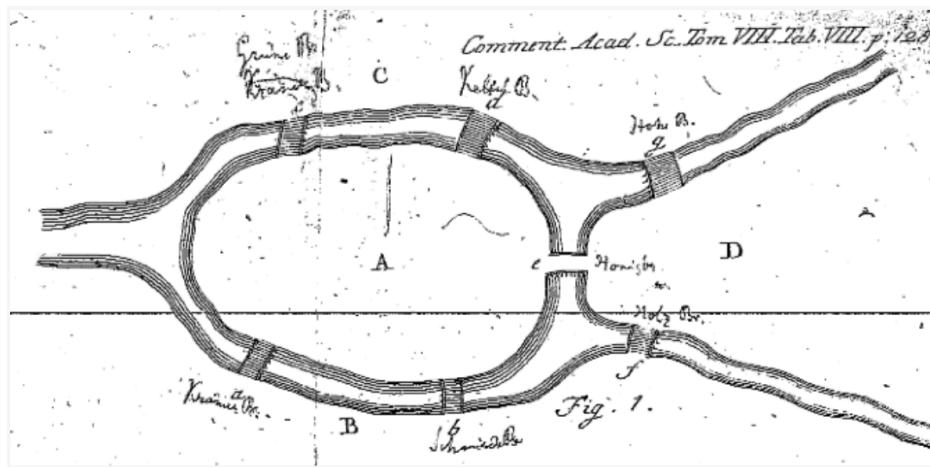
Grafos

- Lista de adjacências



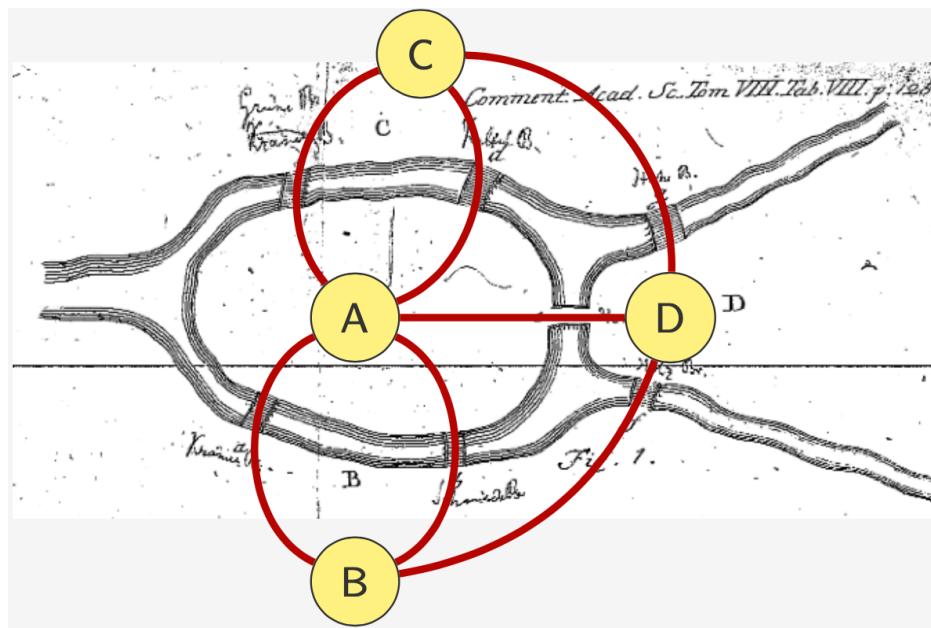
Grafos

- O Problema das Pontes de Königsberg
 - Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes
 - Acreditava-se que era possível passear por toda a cidade atravessando cada ponte exatamente uma única vez



Grafos

- O Problema das Pontes de Königsberg
 - Euler, em 1736, modelou o problema como um grafo
 - Provou que tal passeio não é possível
 - Fundou a Teoria dos Grafos



Grafos

- Multigrafo
 - A estrutura usada por Euler é o que se chama por multigrafo
 - Pode haver arestas paralelas (ou múltiplas)
 - Ao invés de um conjunto de arestas, há um multiconjunto de arestas
 - Pode ser representado por lista de adjacência
 - Por matriz de adjacências é mais difícil

Grafos

- Comparação matriz x lista de adjacências
 - Matriz tem tamanho fixo $O(|V|^2)$
 - Lista tem tamanho variável $O(|V| + |E|)$
 - Matriz é mais rápido
 - Lista é mais econômico
 - Decisão de uso depende da aplicação, da infraestrutura, do tipo de grafo, se o grafo for esparso, ...

Grafos

- Amplamente usados na Matemática e na Ciência da Computação para a modelagem de problemas
 - Redes Sociais: forma de representar uma relação entre duas pessoas
 - Mapas: cidade como um grafo para procurar o menor caminho entre dois pontos
 - Páginas na Internet: links são arcos de uma página para a outra
 - É possível avaliar qual é a página mais popular
 - Redes de Computadores: a topologia de uma rede de computadores é claramente um grafo
 - Circuitos Eletrônicos: é possível criar algoritmos para avaliar se há curto-circuito
 - ...

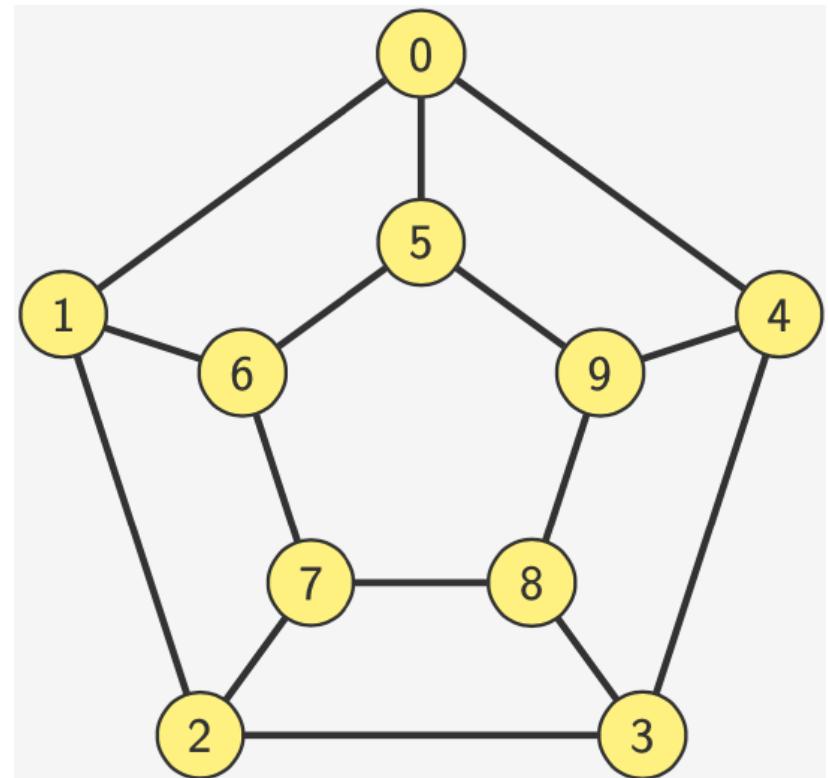
Grafos

- Programa construção de grafo
 - 5 vértices
 - Definição das conexões
 - Impressão das conexões
 - Descoberta do vértice mais popular

Percursos em Grafos

Percursos em Grafos

- Caminho de s para t em um grafo
 - Uma sequência sem repetição de vértices vizinhos
 - Começando em s e terminado em t
 - Exemplos de 0 para 8
 - 0, 1, 6, 7, 2, 3, 8
é um caminho
 - 0, 5, 8
não é um caminho
 - 0, 1, 2, 7, 6, 1, 2, 3, 8
não é um caminho



Percursos em Grafos

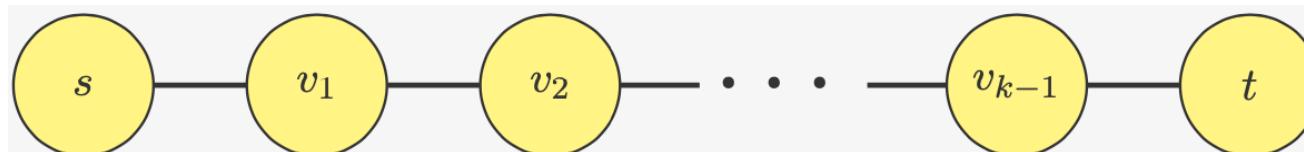
- Formalmente, caminho de s para t em um grafo é
 - Uma sequência de vértices v_0, v_1, \dots, v_k onde
 - $v_0 = s$ e $v_k = t$
 - $\{v_i, v_{i+1}\}$ é uma aresta para todo $0 \leq i \leq k-1$
 - $v_i \neq v_j$ para todo $0 \leq i < j \leq k$
 - k é o comprimento do caminho
 - $k = 0$ se, e somente se, $s = t$

Percursos em Grafos

- Componentes conexas
 - Um par de vértices está na mesma componente se, e somente se, existe caminho entre eles
 - Não há caminho entre vértices de componentes distintas
 - Um grafo conexo tem apenas uma componente conexa

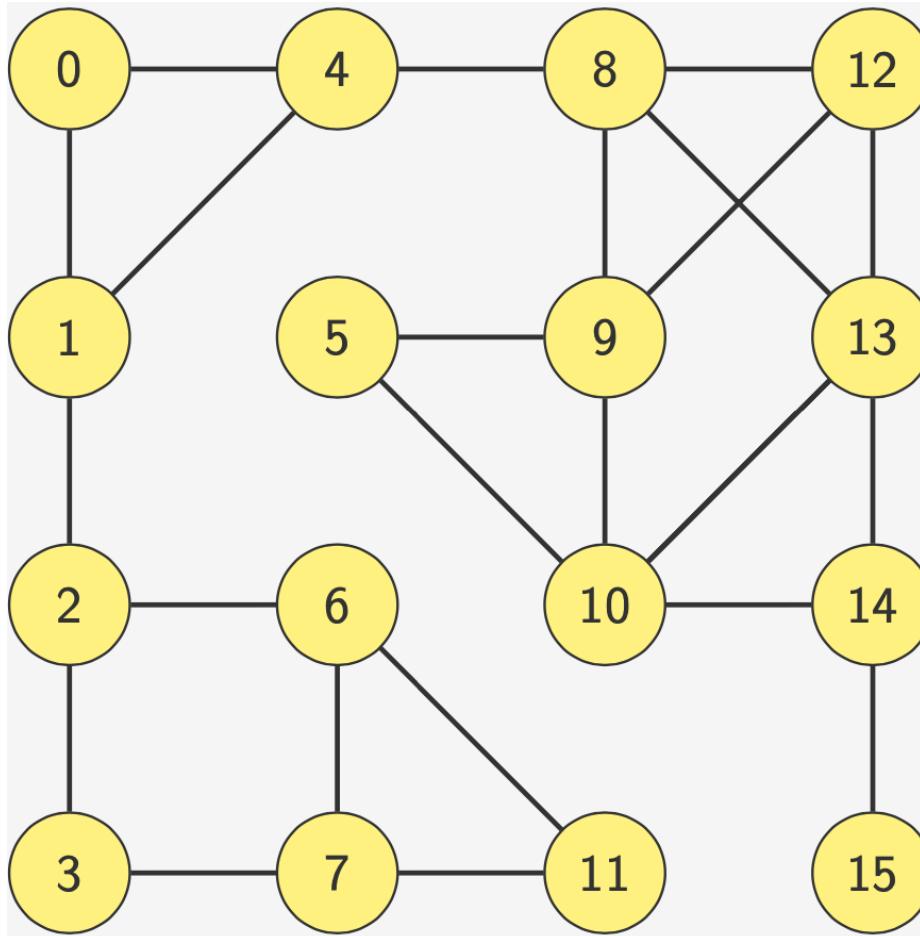
Percursos em Grafos

- Existe caminho de s para t ?
- Mesma pergunta: o vértice s e o vértice t estão na mesma componente conexa ?
 - Se existe caminho e $s \neq t$, existe um segundo vértice v_1
 - E v_1 é vizinho de s
 - Então, ou $v_1 = t$, ou existe um terceiro vértice v_2
 - E v_2 é vizinho de v_1
 - E assim sucessivamente...
 - A dificuldade é acertar qual vizinho v_1 de s deve ser escolhido
 - Melhor solução atual: força bruta !



Percursos em Grafos

- Existe um caminho de 0 a 15 ?



- Busca em profundidade
 - Sempre escolher o menor vértice
 - Ir o máximo possível em uma direção
 - Se não encontrar o destino, voltar o mínimo possível
 - Pegar um novo caminho por um vértice não visitado

Percursos em Grafos

```
int procurarPercorsoRecursivo(int grafo[QTDVERTICES] [QTDVERTICES], int visitado[QTDVERTICES], int inicio, int fim) {
    int vizinho = 0;
    int retorno = 0;

    if (inicio == fim) {
        retorno = 1;
    } else {
        visitado[inicio] = 1;
        for (vizinho = 0 ; vizinho < QTDVERTICES ; vizinho++) {
            if ((grafo[inicio][vizinho]) && (! visitado[vizinho])) {
                if (procurarPercorsoRecursivo(grafo, visitado, vizinho, fim))
                    retorno = 1;
                printf("%d <- ", vizinho);
                break;
            }
        }
    }
}

return retorno;
}
```

Percursos em Grafos

```
int procurarPercorso(int grafo[QTDVERTICES] [QTDVERTICES], int inicio, int fim)
{
    int i = 0;
    int visitado[QTDVERTICES];
    int encontrei = 0;

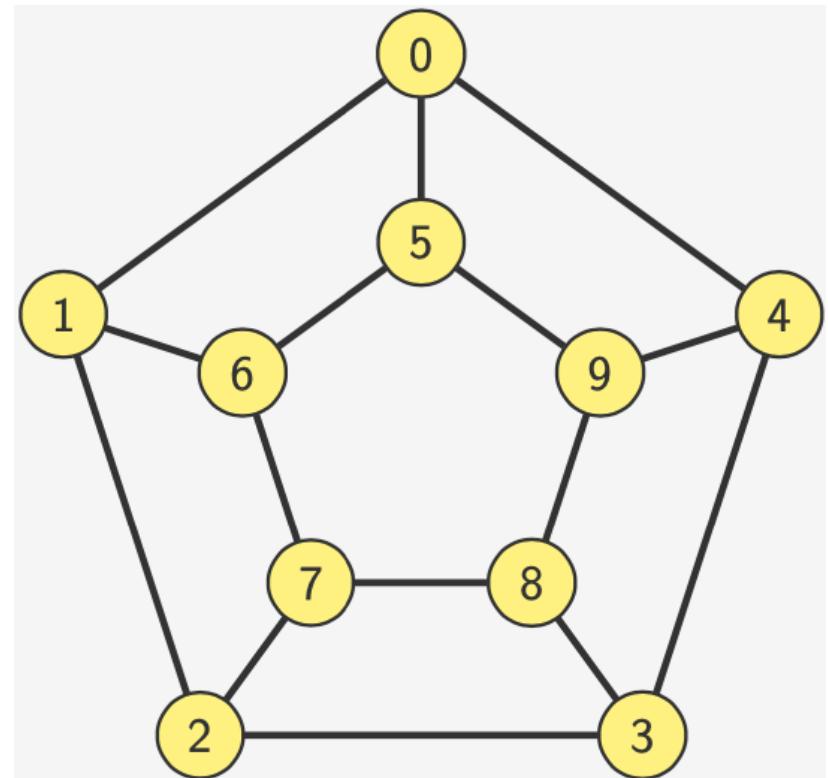
    for (i = 0 ; i < QTDVERTICES ; i++) {
        visitado[i] = 0;
    }

    encontrei = procurarPercorsoRecursivo(grafo, visitado, inicio, fim);
    printf("%d\n", inicio);

    return encontrei;
}
```

Percursos em Grafos

- Ciclo em um grafo
 - Sequência de vértices vizinhos sem repetição exceto pelo primeiro e o último vértice que são idênticos
 - Exemplos
 - 5, 6, 7, 8, 9, 5 é um ciclo
 - 1, 2, 3 não é um ciclo
 - 1, 2, 7, 6, 1 é um ciclo
 - 1, 2, 7, 6, 1, 0
não é um ciclo



- Subgrafo
 - Grafo obtido a partir da remoção de vértices e arestas
 - É possível considerar que árvores e florestas são subgrafos de um grafo dado
- Árvore
 - Grafo conexo acíclico
- Floresta
 - Grafo acíclico
 - Suas componentes conexas são árvores

Percursos em Grafos

- É possível realizar busca em profundidade recursivamente ou com pilhas
- É possível realizar busca em largura com filas
- Todo caminho revela uma árvore por meio da qual é possível navegar de qualquer vértice até a raiz

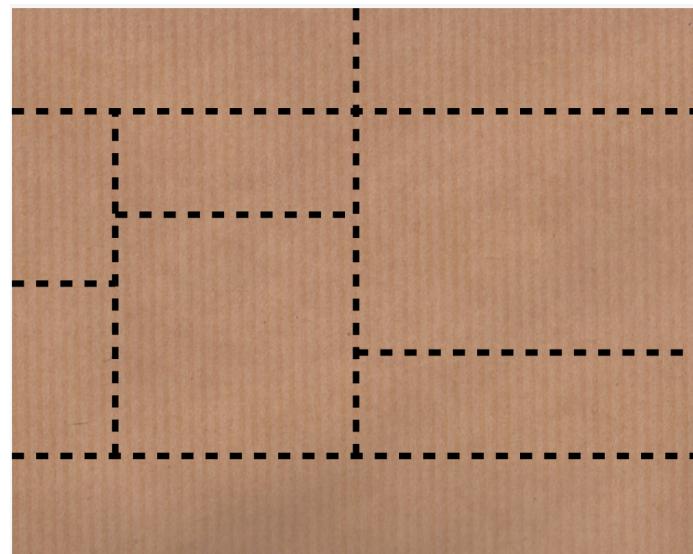
Percursos em Grafos

- Programa busca em profundidade em um grafo
 - 7 vértices
 - Definição das conexões
 - Descoberta do caminho entre 2 vértices

Algoritmos em Grafos

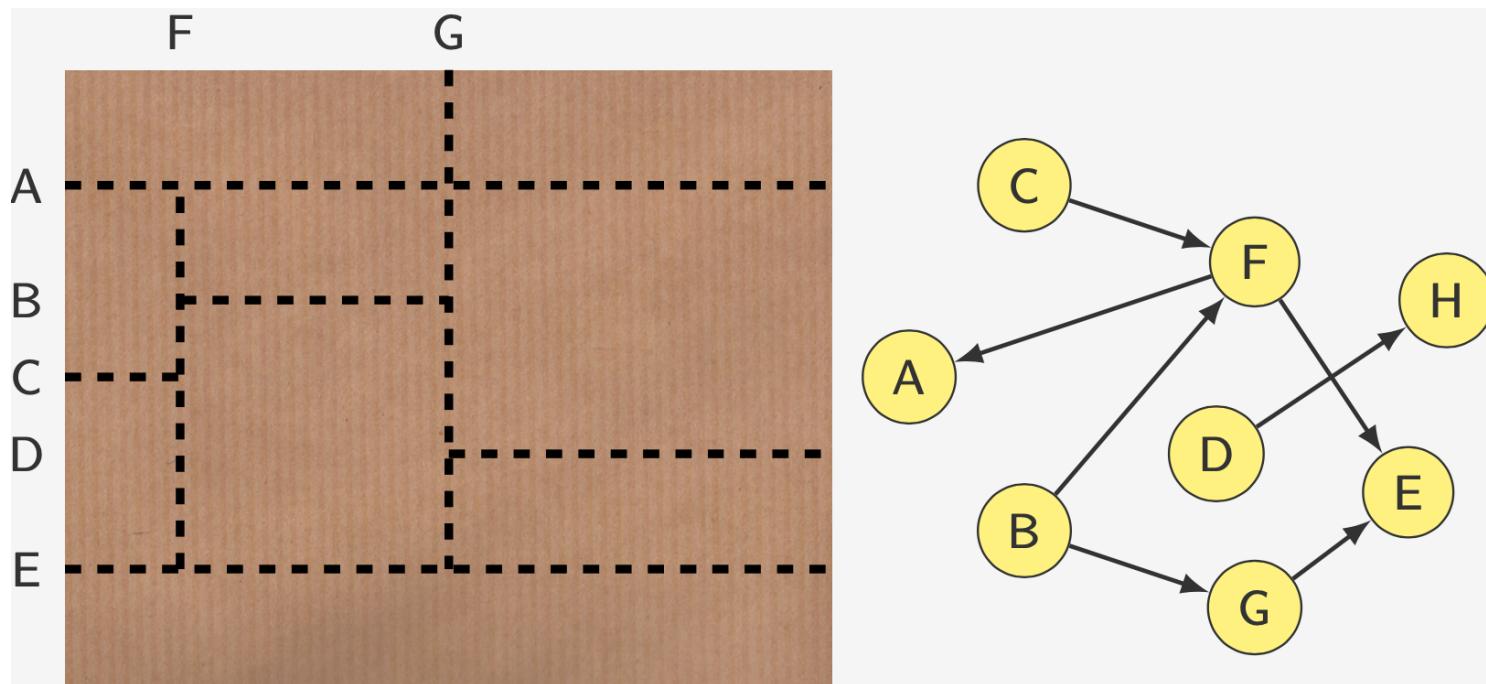
Algoritmos em Grafos

- Corte de material
 - Uma fábrica precisa cortar papelão retangular
 - Ela usa uma grande guilhotina (maior que o papelão)
 - Existe um padrão de corte escolhido para evitar desperdício
 - Os cortes podem ser feitos pela guilhotina ?
 - Em que ordem eles devem ser feitos ?



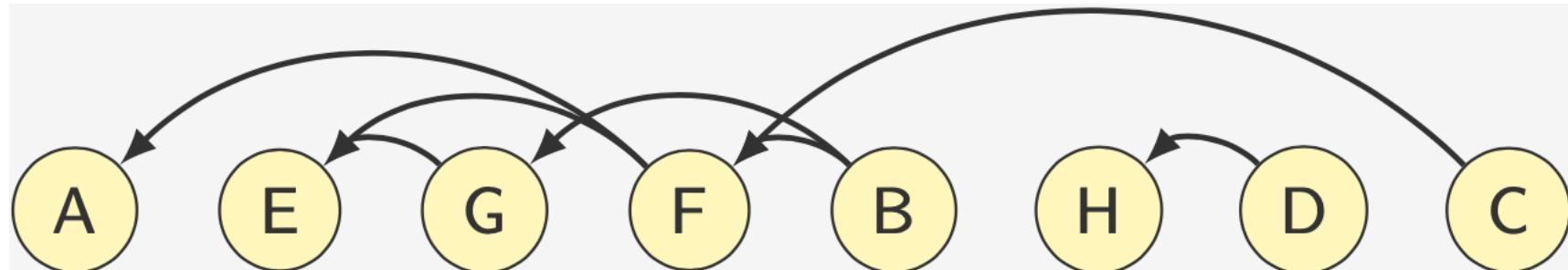
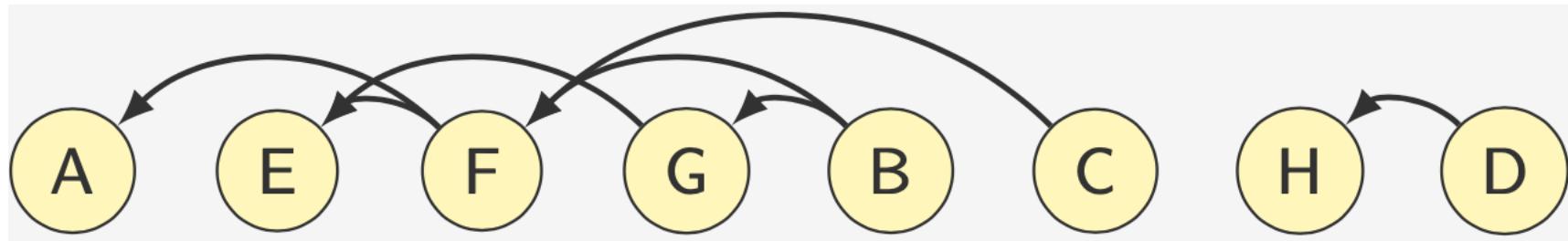
Algoritmos em Grafos

- Modelagem como um grafo
 - Vértices: cada corte da guilhotina
 - Arestas: se um corte deve acontecer antes de outro



Algoritmos em Grafos

- Ordenação topológica
 - Uma ordenação topológica de um grafo acíclico é uma ordenação dos vértices cujas arestas estão na mesma direção



Algoritmos em Grafos

- Como encontrar um ordenação topológica ?
 - Considere um vértice u
 - Todo v tal que (u, v) é um arco deve aparecer antes u
 - Todo vértice w tal que existe arco (v, w) e arco (u, v) deve aparecer antes de u
 - E assim sucessivamente...
 - Devem-se considerar todos os w tal que exista caminho de u para w antes de considerar u
 - Lembra uma pós-ordem em árvores binárias
 - Como encontrar todo w tal que exista caminho de u para w ?
 - Busca em profundidade

Algoritmos em Grafos

```
void ordenarTopologicoRecursivo(int grafo[QTDVERTICES] [QTDVERTICES], int
visitado[QTDVERTICES], int vertice) {

    int i;

    visitado[vertice] = 1;

    for (i = 0 ; i < QTDVERTICES ; i++) {
        if (grafo[vertice][i]) {
            if (! visitado[i]) {
                ordenarTopologicoRecursivo(grafo, visitado, i);
            }
        }
    }

    printf("%d\t", vertice);
}
```

Algoritmos em Grafos

```
void ordenarTopologico(int grafo [QTDVERTICES] [QTDVERTICES]) {  
    int vertice;  
    int visitado [QTDVERTICES];  
  
    for (vertice = 0 ; vertice < QTDVERTICES ; vertice++) {  
        visitado[vertice] = 0;  
    }  
  
    for (vertice = 0 ; vertice < QTDVERTICES ; vertice++) {  
        if (! visitado[vertice]) {  
            ordenarTopologicoRecursivo(grafo, visitado, vertice);  
        }  
    }  
  
    printf("\n");  
}
```

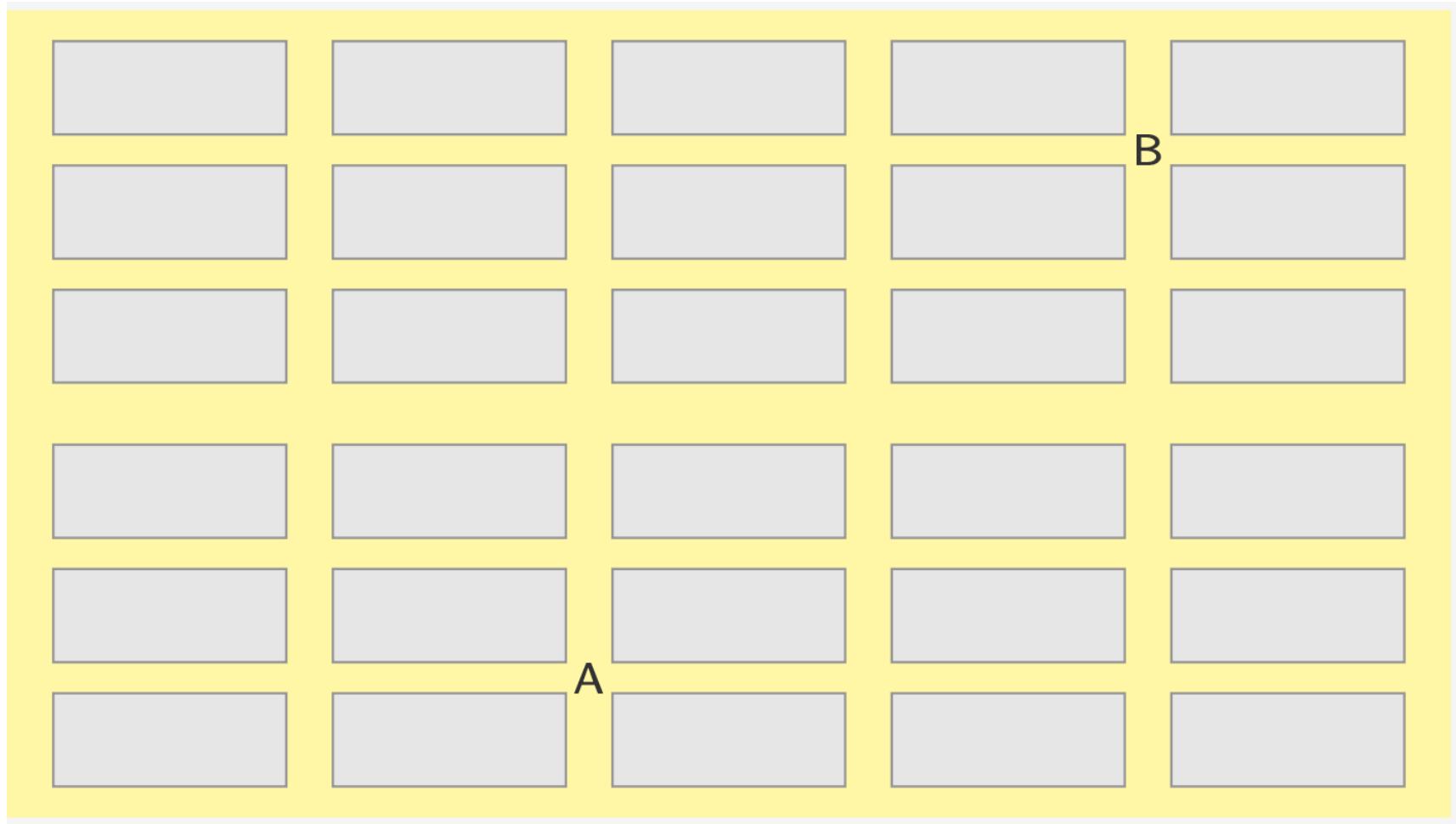
Algoritmos em Grafos

- Como encontrar o menor tempo para ir de A a B ?



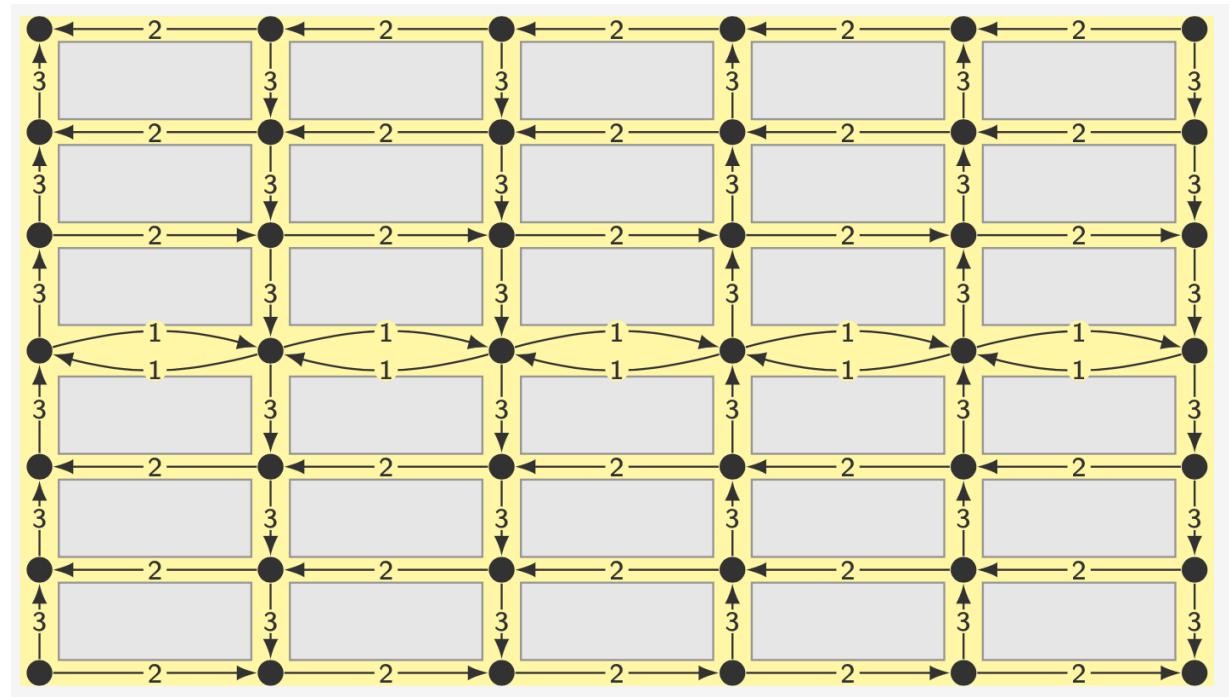
Algoritmos em Grafos

- Como encontrar o menor tempo para ir de A a B ?



Algoritmos em Grafos

- Modela-se um digrafo com pesos nos arcos
 - Um vértice em cada cruzamento
 - Um arco entre vértices consecutivos
 - O peso do arco (u, v) é o tempo de viagem de u para v

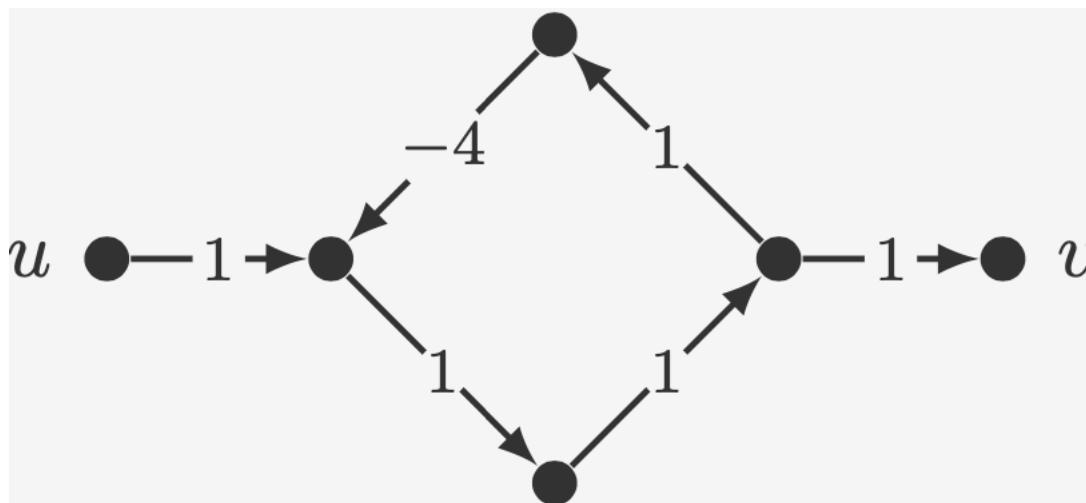


Algoritmos em Grafos

- Modela-se um digrafo com pesos nos arcos
 - Na lista de adjacências
 - Basta adicionar um campo peso no nó da lista
 - Na matriz de adjacências
 - Basta colocar o peso na matriz
 - Trocar a indicação 0 por -1 para diferenciar tempo 0 de não haver aresta entre dois vértices

Algoritmos em Grafos

- Caminhos de custo mínimo
 - Encontrar um caminho de peso mínimo de u para v no digrafo
 - Considerem-se pesos não-negativos
 - Risco de ciclo negativo infinito



Algoritmos em Grafos

- Caminhos de custo mínimo
 - Como é o caminho mínimo de u para v ?
 - Ou u é vizinho de v
 - Ou o caminho passa por um vizinho w de v
 - Soma do peso do caminho de u para w e de (w, v) é mínima
 - Este caminho de u a w tem que ter peso mínimo

Algoritmos em Grafos

- Caminhos de custo mínimo
 - Árvore de caminhos mínimos a partir de u
 - Dado u , o algoritmo encontra uma árvore enraizada em u
 - O caminho de v para u na árvore é um caminho mínimo de u para v no digrafo

Algoritmos em Grafos

- Algoritmo de Dijkstra
 - Algoritmo guloso que gera solução ótima em tempo polinomial
 - Inicia no vértice u
 - A cada passo, buscam-se as adjacências com menor distância relativa a u
 - Constrói-se um conjunto S de menores caminhos contendo todos os vértices alcançáveis por u .
 - Arestras que ligam vértices já pertencentes a S são desconsideradas
 - Finalmente determina o menor caminho com final em v

Algoritmos em Grafos

```
1 int * dijkstra(p_grafo g, int s) {
2     int v, *pai = malloc(g->n * sizeof(int));
3     p_no t;
4     p_fp h = criar_fprio(g->n);
5     for (v = 0; v < g->n; v++) {
6         pai[v] = -1;
7         insere(h, v, INT_MAX);
8     }
9     pai[s] = s;
10    diminuiprioridade(h, s, 0);
11    while (!vazia(h)) {
12        v = extrai_minimo(h);
13        if (prioridade(h, v) != INT_MAX)
14            for (t = g->adj[v]; t != NULL; t = t->prox)
15                if (prioridade(h, v)+t->peso < prioridade(h, t->v)) {
16                    diminuiprioridade(h,t->v,prioridade(h,v)+t->peso);
17                    pai[t->v] = v;
18                }
19    }
20    return pai;
21 }
```

Algoritmos em Grafos

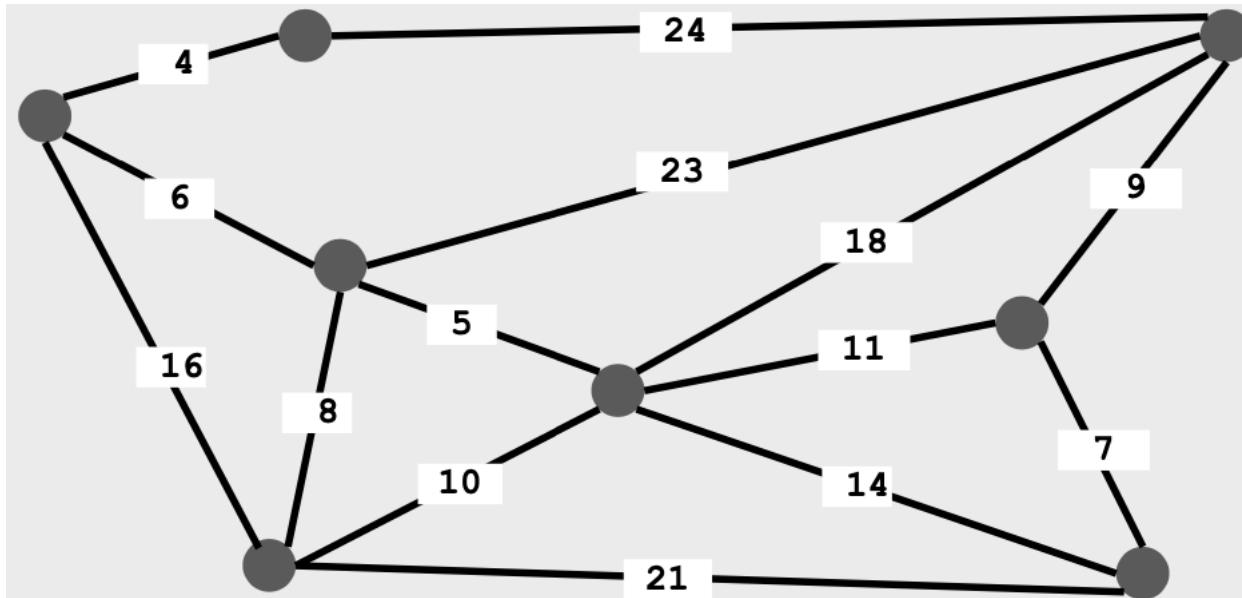
- Programa ordenação topológica do grafo
 - 8 vértices
 - Definição das conexões
 - Descoberta da ordenação topológica do grafo

Árvores Geradoras Mínimas



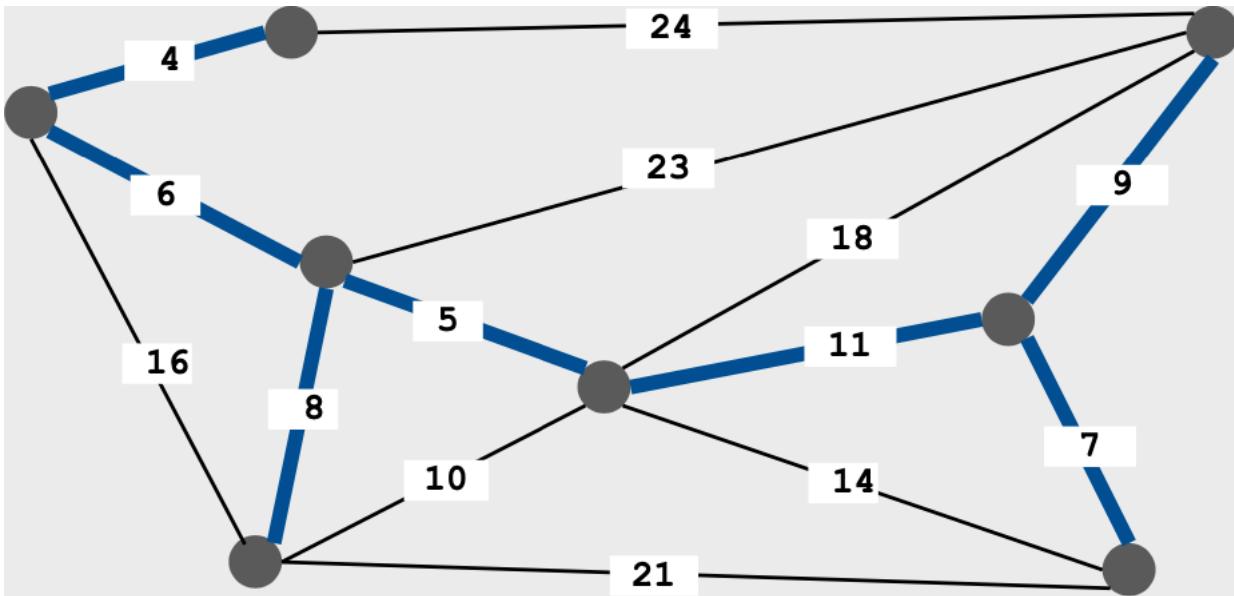
Árvores Geradoras Mínimas

- Dado um grafo não direcionado conectado com pesos positivos nas arestas
- Objetivo: Encontrar o conjunto de arestas com peso total mínimo que conecte todos os vértices do grafo



Árvores Geradoras Mínimas

- Por força bruta
 - Não é fácil de implementar
 - Complexidade espacial muito grande
 - $V^V(V - 2)$ árvores possíveis no grafo com V vértices



Árvores Geradoras Mínimas

CEUB

EDUCAÇÃO SUPERIOR

- Problema fundamental com diversas aplicações
 - Projeto de redes
 - Telefonia
 - Elétricas
 - Hidráulicas
 - TV a cabo
 - Computadores
 - Estradas
 - Algoritmos de aproximação para problemas NP-difíceis
 - Caixeiro viajante
 - Árvore de Steiner

Árvores Geradoras Mínimas

CEUB

EDUCAÇÃO SUPERIOR

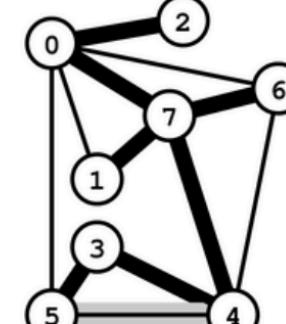
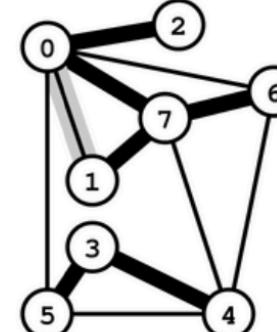
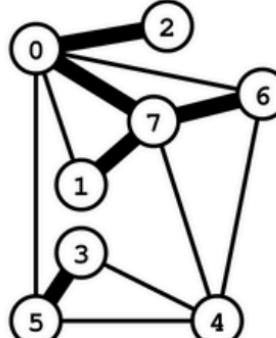
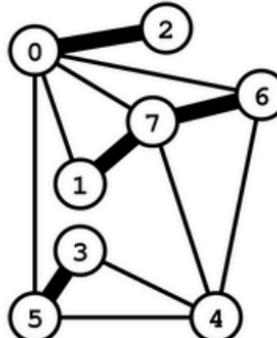
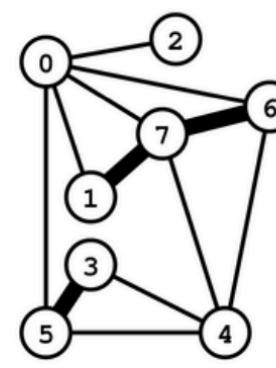
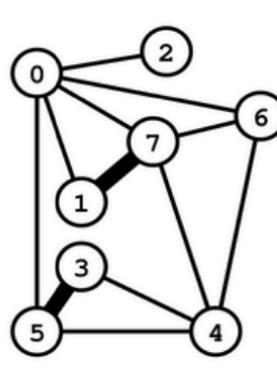
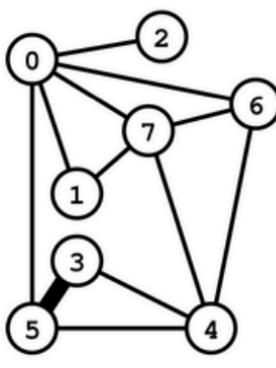
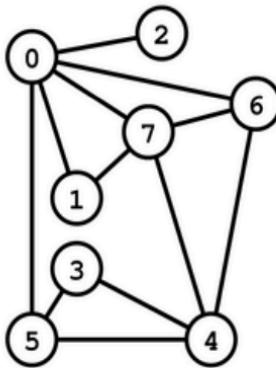
- Problema fundamental com diversas aplicações
 - Aplicações indiretas
 - Caminhos de gargalos máximos
 - Códigos LDPC para correção de erros na transmissão de dados
 - Registro de imagens com entropia de Renyi
 - Aprendizagem de características salientes na verificação de rostos em tempo real
 - Redução do espaço de armazenamento no sequenciamento de aminoácidos de uma proteína
 - Modelagem da interação localizada de partículas em fluxos turbulentos de fluídos
 - Protocolos autoconfiguráveis de roteamento Ethernet para evicção de ciclos em uma rede
 - Clusterização

- Algoritmo de Kruskal
 - 1956
 - Relacionar as arestas em ordem crescente de pesos
 - Adicionar cada aresta à árvore, na ordem, a menos que o acréscimo crie um ciclo

Árvores Geradoras Mínimas

- Algoritmo de Kruskal

3-5	0.18
1-7	0.21
6-7	0.25
0-2	0.29
0-7	0.31
0-1	0.32
3-4	0.34
4-5	0.40
4-7	0.46
0-6	0.51
4-6	0.51
0-5	0.60



0-2

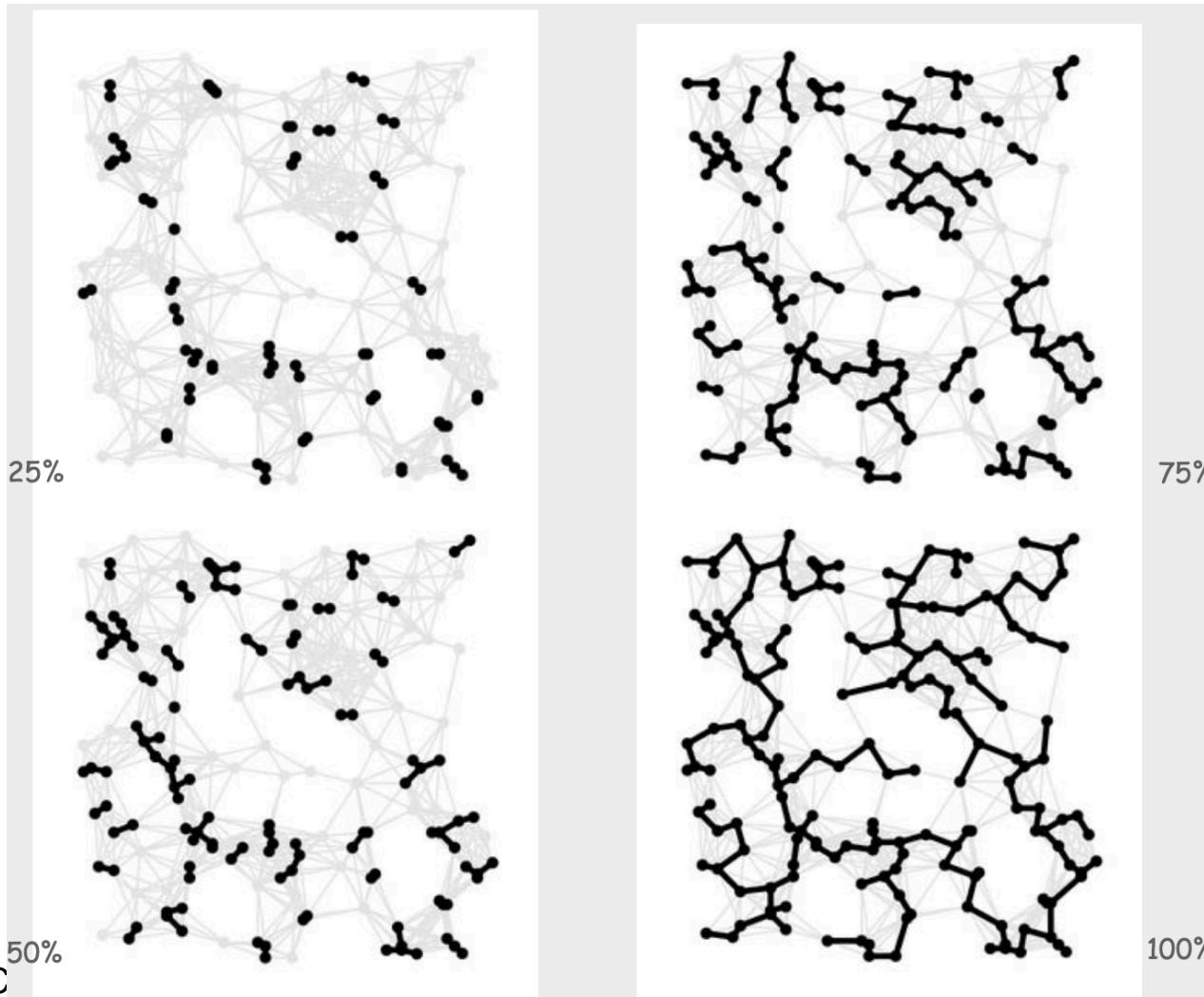
0-7

0-1 3-4

4-5 4-7

Árvores Geradoras Mínimas

- Algoritmo de Kruskal

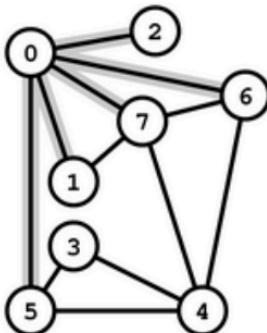


- Algoritmo de Prim
 - 1959
 - Inicia a árvore no vértice 0
 - A cada passo, buscam-se as adjacências relativas às folhas da árvore, escolhendo-se a menor delas

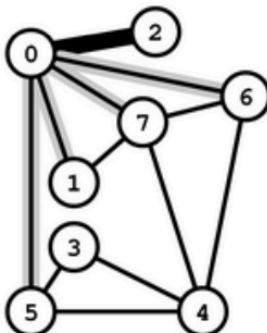
Árvores Geradoras Mínimas

- Algoritmo de Prim

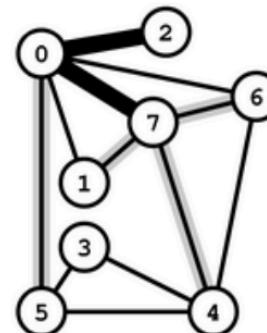
0-1 0.32
0-2 0.29
0-5 0.60
0-6 0.51
0-7 0.31
1-7 0.21
3-4 0.34
3-5 0.18
4-5 0.40
4-6 0.51
4-7 0.46
6-7 0.25



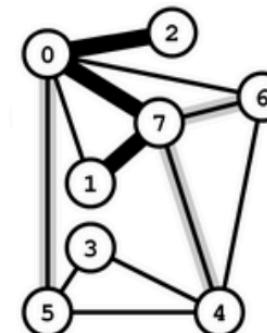
0-2 0-7 0-1 0-6 0-5



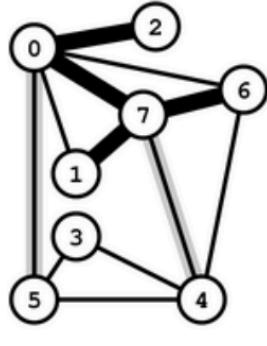
0-7 0-1 0-6 0-5



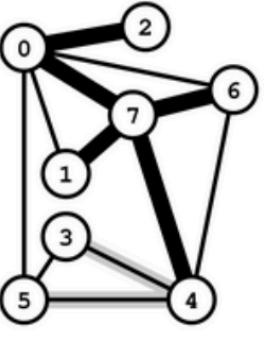
7-1 7-6 7-4 0-5



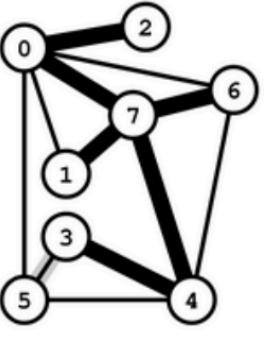
7-6 7-4 0-5



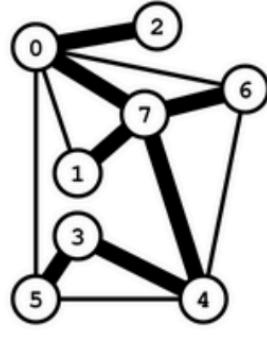
7-4 0-5



4-3 4-5

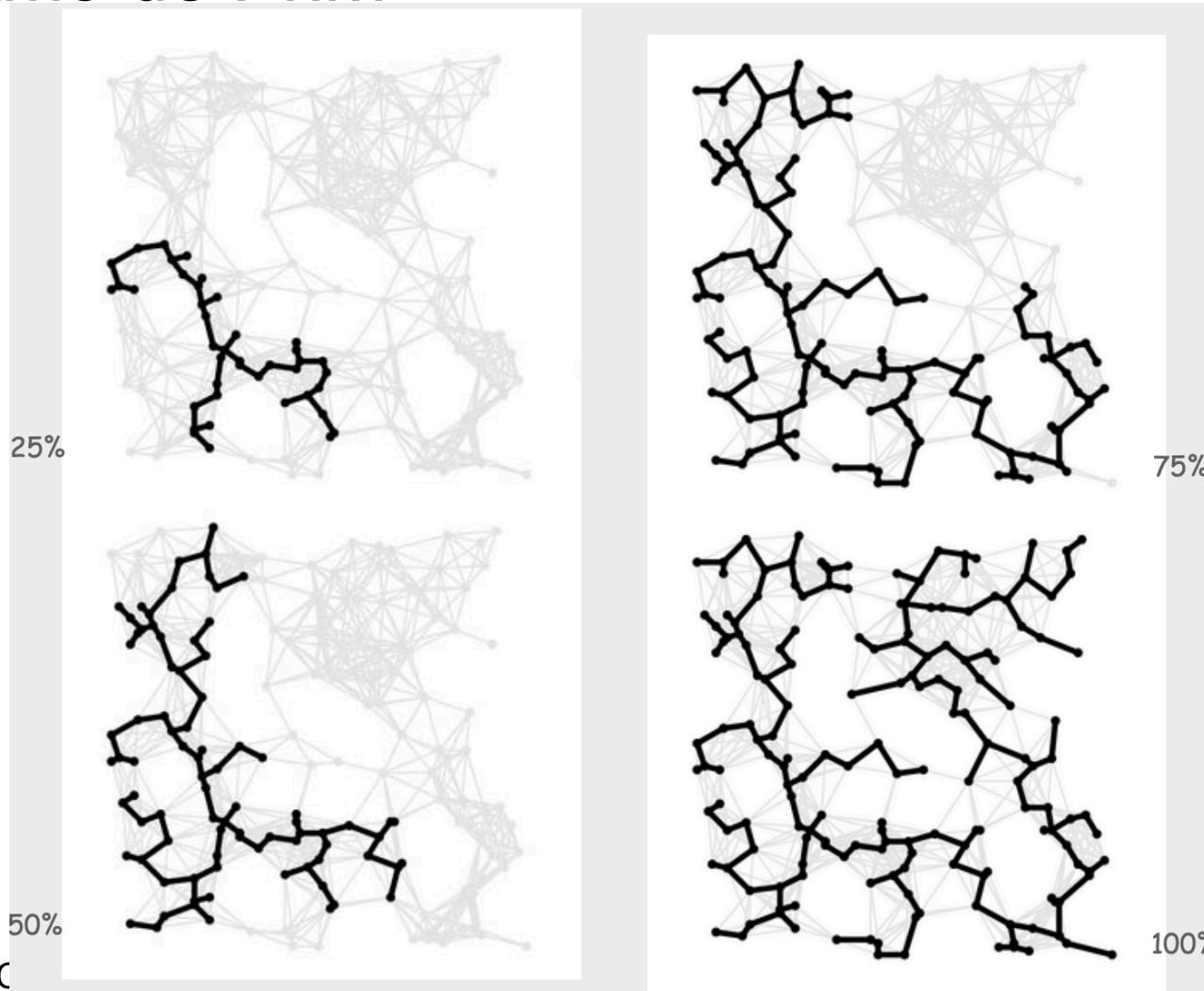


3-5



Árvores Geradoras Mínimas

- Algoritmo de Prim



Árvores Geradoras Mínimas

CEUB

EDUCAÇÃO SUPERIOR

Ano	Pior Caso	Descoberto por
1975	$E \log(\log(V))$	Yao
1976	$E \log(\log(V))$	Cheriton-Tarjan
1984	$E V * \log(V)$	Fredman-Tarjan
1986	$E \log(\log(V))$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) * \log(\alpha V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	ótimo	Pettie-Ramachandran
20??	E	???

!! Obrigado !!

ceub.br

