1) O que é árvore binária?

É um conjunto finito de elementos, chamados nós, tal que ou a árvore é vazia, ou há um nó raiz e os demais nós podem ser divididos em subconjuntos disjuntos, subárvores à esquerda e à direita da raiz; estas também árvores binárias.

2) O que é árvore binária de busca?

É uma árvore binária cujo valor do nó à esquerda da raiz é inferior ao valor desta, e o nó à direita da raiz é superior ao valor desta.

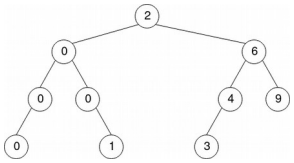
3) O que é árvore estritamente binária, completa e cheia?

Estritamente binária: nós possuem 0 ou 2 filhos.

Binária completa: todo nó que não possuir filhos deve estar ou no último ou no penúltimo nível da árvore.

Binária cheia: todo nó que não possuir filhos deve estar no último nível da árvore.

4) Faça uma árvore binária de busca com sua matrícula.



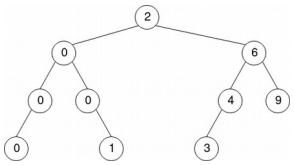
5) Quais os problemas de uma árvore binária de busca?

Depois de muitas operações de inserção e remoção, a árvore pode deformar, aumentando a complexidade da busca. Árvores binárias de busca podem se deformar ao ponto que fiquem em zigue-zague, tornando-se praticamente listas encadeadas.

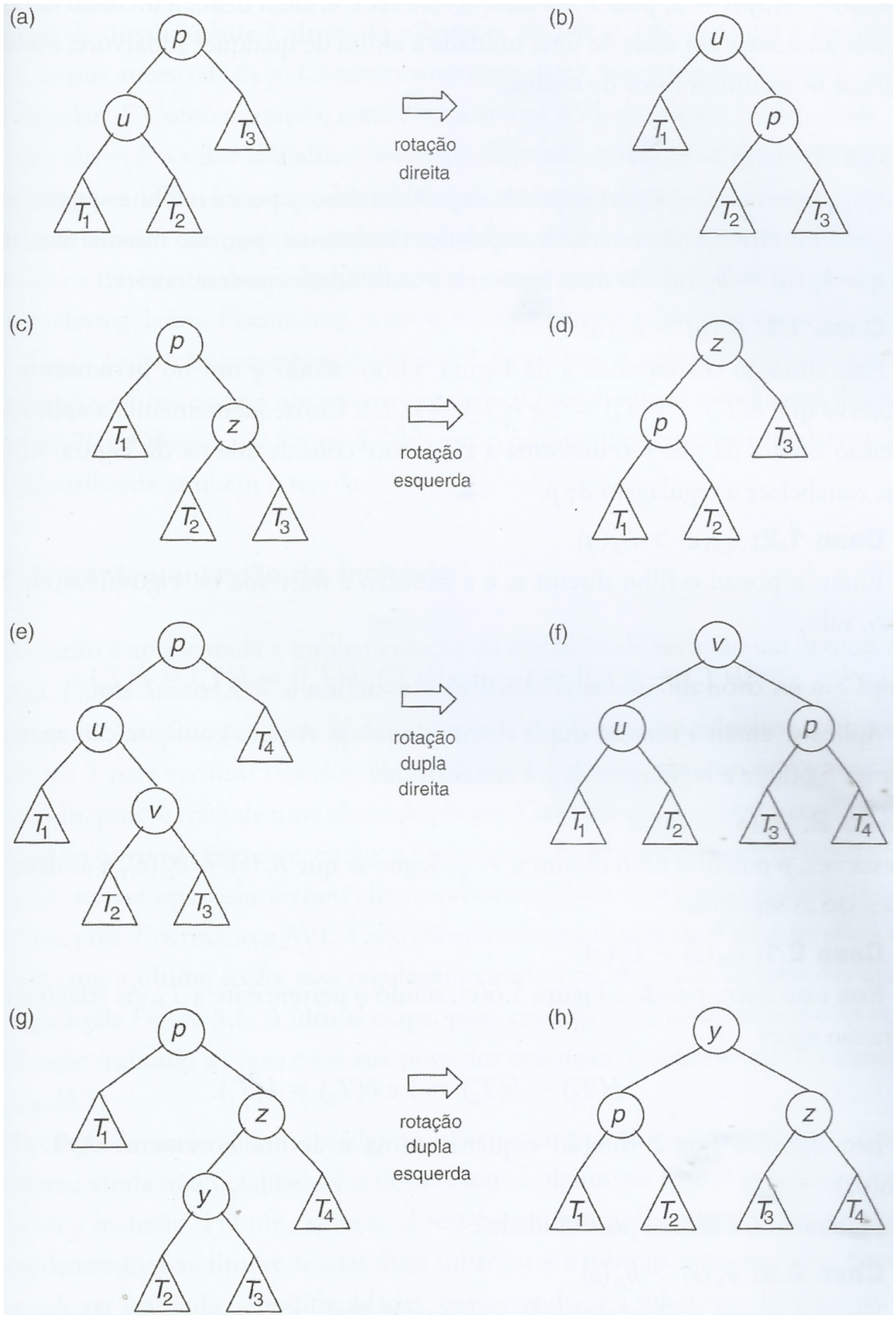
6) O que é uma árvore balanceada?

É uma árvore binária de busca que, a despeito de inclusões ou remoções, mantém o custo de acesso em $O(\log n)$, ou seja, uma árvore de grandeza ótima.

7) Faça uma árvore AVL com sua matrícula.



8) Explique o procedimento de rotação em uma árvore.



9) Qual a justificativa para se usar árvores B?

O fato de que, por manter mais de uma chave em cada nó da estrutura, a árvore B proporcionar uma organização de ponteiros de forma que operações de busca, inserção e remoção sejam executadas rapidamente. No mais, todas as folhas de uma árvore B encontram-se sempre no mesmo nível, não importando a ordem da entrada de dados.

10) O que a ordem significa numa árvore B?

É por meio da ordem que se infere quantos filhos no máximo tem cada nó (2 * ordem + 1), e também quantos filhos no mínimo estes nó, caso não sejam a raiz ou folhas, possuem (ordem + 1.)

11) O que é medida de complexidade?

A complexidade é uma expressão matemática que traduz o comportamento de tempo de um algoritmo, medida de forma analítica.

12) Qual a importância de conhecer a complexidade de um algoritmo?

Por ser uma medida analítica, e não empírica, a complexidade fornece uma maneira segura de comparar a performance de algoritmos sem depender de fatores externos como o *hardware* no qual estão sendo executados, ou o tamanho de suas entradas, o conteúdo destas etc.

13) O que é complexidade de melhor, pior ou médio caso? Para que serve?

A complexidade de melhor caso é quando o dado de entrada leva o menor tempo para ser processado pelo algoritmo. O pior caso, por sua vez, é quando o dado de entrada leva o maior tempo para ser processado pelo algoritmo. O caso médio, por sua vez, é uma média probabilística dos casos possíveis. A complexidade de pior caso é especialmente útil em sistemas *timeshare*, pois somente com ela pode se estimar um tempo de execução mínimo para o pior dos casos. A complexidade média é, em geral, a que se usa cotidianamente. A do melhor caso é útil para melhorar a precisão de análise dos piores casos.

14) Defina balanceamento de árvore em termos de complexidade.

A idéia do balanceamento é, justamente, manter as árvores com altura próxima a log *n*, a fim de que a complexidade dos algoritmos de inserção, busca e remoção seja sempre O(log *n*.) No entanto, a operação de balancear uma árvore binária de busca qualquer a cada inserção é extremamente custoso, beirando o Ω(n), daí a necessidade de utilizar estruturas especiais, como a árvore AVL ou a rubro-negra.

15) Esboce um algoritmo de complexidade: O(n^2), O(n^3) e O(2^n.)

Um algoritmo O(n^2) é soma de matrizes. Um algoritmo O(n^3) é multiplicação de matrizes. Um algoritmo O(2^n) é o recursivo para resolução da torre de Hanói.

16) Explique como o termo log *n* surge nos algoritmos de árvore.

Este termo surge quando a árvore é balanceada, fazendo com que sua altura seja, em média, log *n*, o que faz com que todos os algoritmos de inserção, remoção e busca tenham uma complexidade de O(log *n*.)

NOTAS

Para verificar se a árvore que você montou é uma binária de busca ou não, basta imprimi-la de forma simétrica, isto é, ir para o nó mais à esquerda, imprimir o nó, ir para o nó mais à direita, recursivamente. Se os elementos forem impressos do menor ao maior, a árvore é binária de busca :)

Para verificar se uma árvore é AVL ou não, o índice de desequilíbrio de cada um de seus nós não pode ser menor que -1 ou maior que 1. O índice de desequilíbrio de um nó é calculado da seguinte maneira: para cada nó à esquerda deste nó raiz, subtraia 1; para cada nó à direita deste nó raiz, some 1. Como os nós-folhas não possuem nada à esquerda ou à direita, seu índice é sempre 0.

Para decidir qual rotação realizar em um nó AVL para manter a árvore organizada, deve-se seguir o algoritmo abaixo:

1. Calcular o índice de desequilíbrio do nó (índice chamado Q)
2. Se -1 <= Q <= 1, o nó está equilibrado
3. Se Q > 1:
    1. Se a subárvore da direita tem Q < 0:
        1. Rotação dupla à esquerda
    2. Se a subárvore da direita tem Q > 0:
        1. Rotação à esquerda
4. Se Q < -1:
    1. Se a subárvore da esquerda tem Q > 0:
        1. Rotação dupla à direita
    2. Se a subárvore da esquerda tem Q < 0:
        1. Rotação à direita

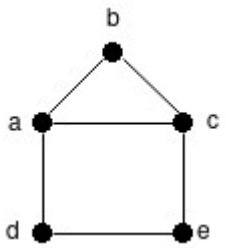
A ordem *o* de uma árvore B, de acordo com o livro que o Rafael usa, implica que cada nó tem no máximo (2*o* + 1) filhos. Cada nó diferente da raiz e das folhas tem, no mínimo, *o*+1 elementos.

Todas as folhas de uma árvore B estão sempre no mesmo nível. Assim, o crescimento da árvore B é sempre para cima. Basicamente: não pode haver chave sem filho em uma árvore B.

GRAFOS

Um grafo G = (v, e) é uma estrutura que consiste em um conjunto de vértices V e um conjunto de arestas E. Uma aresta e∈E representa uma ligação entre dois vértices u, v ∈ V, e é representado por (u, v.) O que isso significa? ¯_(ツ)_/¯

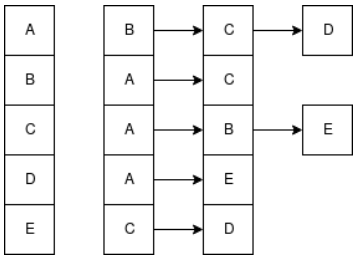
Um grafo pode ser representado de duas formas distintas, sendo elas a **matriz de adjacências** e a **lista de adjacências**. A única diferença entre elas, como o nome sugere, é que uma usa a matriz para representação estrutural, e a outra usa listas. A matriz não é muito recomendada porque, às vezes, precisa-se de muitas linhas e colunas para se representar um grafo.



Na matriz de adjacências, faz-se *v* linhas e *v* colunas, sendo *v* a quantidade de vértices. No que cada vértice for ligado, coloca-se 1, no que não for, 0.

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	0
C	1	1	0	0	1
D	1	0	0	0	1
E	0	0	1	1	0

Na lista de adjacências, há um vetor contendo todos os *v* vértices do grafo e, para cada um dos elementos deste vetor, há uma lista encadeada contendo os vértices com os quais o vértice do elemento tem ligação.



Um grafo pode ser **direcionado.** Neste caso, a ordem do par (u, v) representando uma aresta é relevante. Isto é, pode-se ir de A para B, mas talvez não de B para A.

Um grafo é fortemente conexo se cada um dos seus vértices está fortemente ligado aos demais. Logo, um grafo é fortemente conexo se todos os vértices estão ao alcance um dos outros.

Um grafo é **ponderado** quando seus vértices ou arestas possuem valor. Por exemplo, às vezes não queremos saber o menor caminho para se chegar a um destino, mas sim aquele no qual se pagará menos pedágio. Com isto, colocamos os valores dos pedágios em cada conexão de uma aresta do caminho à outra; isto fará do grafo um grafo ponderado.

Se há um vértice *v* e um vértice *u*, diz-se que eles são **vizinhos** ou **adjacentes** se existir uma aresta e = (u, v.)

Se e_1 = (u, v) e e_2 = (v, w), diz-se que e_1 e e_2 são **incidentes**.

Um **caminho** é uma sequência de vértices $v_1...v_k$ disjuntos em que (v_i, v_{i+1}), com 1 <= i <= k, são arestas. Neste caso, se (v_k, v_i) são arestas também, diz-se que o grafo é **cíclico**.

BUSCA EM LARGURA

Por meio da busca em largura, pode-se procurar o caminho mínimo para se chegar às arestas de um grafo. Utiliza uma fila.

Ela visita um vértice inicial *v*, e então visita todos os seus vizinhos, e então repete este processo para cada um dos vizinhos.

Primeiro, deve-se marcar todos os vértices como não visitados.

Então, visita-se um vértice v_0 e coloca-o na fila.

Deve-se, então, pegar quem está no início da fila, marcar seus vizinhos como visitados e colocá-los no final da fila. Aí, repete-se este passo até que se esvazie a fila.

BUSCA EM PROFUNDIDADE

Fornece a ordem parcial do grafo, além de ser útil para verificar componentes fortemente conexos, ou para resolver quebra-cabeças (labirintos.) Usa pilha.

Primeiramente, visita-se o vértice v_0; ele é marcado como visitado e empilhado.

Depois, visita-se seu primeiro vértice adjacente, que é empilhado. Quando se chegar ao último vértice cujos todos vértices adjacentes já foram visitados, este é desempilhado.

Quando se esvaziar a pilha, o grafo foi todo visitado.