Q1 - Quantos antecedentes tem um nó no nível n em uma árvore binária?

R.: Um nó no nível n tem exatamente n antecedentes (ou seja, n nós entre ele e a raiz).

Q2 - Uma árvore cheia binária com n nós folhas contém quantos nós?

R.: Uma árvore binária cheia com n folhas tem exatamente 2n - 1 nós no total.

Árvores Binárias de Busca (ABB)

Para as próximas questões, assuma que esta é a struct do nó da árvore.

```
typedef struct reg {
  int       chave;
  int       conteudo;
  struct reg *esq, *dir;
} noh;
```

Q3 - Suponha que x->esq->chave ≤ x->chave para cada nó x dotado de filho esquerdo e x->chave ≤ x->dir->chave para cada nó x dotado de filho direito. Essa árvore é de busca?

R.: Sim. A condição:

x->esq->chave <= x->chave

x->chave <= x->dir->chave é a definição de uma árvore binária de busca.

Q4 - Escreva uma função que decida se uma dada árvore binária é ou não é de busca.

```
int ehABB(noh* raiz, int min, int max) {
   if (raiz == NULL) return 1;
   if (raiz->chave < min || raiz->chave > max) return 0;
   return ehABB(raiz->esq, min, raiz->chave) &&
          ehABB(raiz->dir, raiz->chave, max);
}
```

Chamada inicial: ehABB(raiz, INT_MIN, INT_MAX);

Q5 - Suponha que as chaves 50 30 70 20 40 60 80 15 25 35 45 36 são inseridas, nesta ordem, numa árvore de busca inicialmente vazia. Desenhe a árvore que resulta. Em seguida, remova o nó que contém 30.

- Inserindo: 50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45, 36
- A árvore resultante será balanceada à esquerda e à direita.
- Ao remover o nó 30 (com dois filhos), substituímos pelo maior da subárvore esquerda (25) ou menor da direita (35).

Q6 - Considere árvores binárias de busca cujos nós têm a estrutura indicada abaixo. Escreva uma função que receba a raiz de uma tal árvore e a chave de um nó x e devolva o endereço do pai de x.

```
typedef struct reg {
  int       chave;
  int       conteudo;
  struct reg *esq, *dir;
} noh;
```

Se x não pertence à árvore, a função deve devolver NULL. O consumo de tempo de sua função deve ser limitado pela profundidade de x.

```
noh* encontrarPai(noh* raiz, int chave) {
   if (raiz == NULL || raiz->chave == chave) return NULL;
   if ((raiz->esq && raiz->esq->chave == chave) ||
      (raiz->dir && raiz->dir->chave == chave)) return raiz;

   if (chave < raiz->chave)
      return encontrarPai(raiz->esq, chave);
   else
      return encontrarPai(raiz->dir, chave);
}
```

Q7 - Dada uma AVL cuja raiz é um nó folha com a chave 50:
(a) insira os elementos {1, 64, 12, 18, 66, 38, 95, 58, 59, 70, 68, 39, 62, 7, 60, 43, 16, 67, 34, 35} nesta árvore, indicando as rotações necessárias;
(b) Retire os elementos {50, 95, 70, 60, 35} desta árvore, explicitando as rotações.

- Inserir: {1, 64, 12, 18, 66, 38, 95, 58, 59, 70, 68, 39, 62, 7, 60, 43, 16, 67, 34, 35}
- A cada inserção, verificar o fator de balanceamento e aplicar rotações:
  - Simples à esquerda/direita
  - Dupla à esquerda/direita

Grafos

Q8 - Desenhe os seguintes grafos:
a) G(V, E), onde V = {1, 2, 3, 4, 5, 6} e E = {(2, 5), (6, 1), (5, 3), (2, 3)}.

b) G(V, E), onde V = {1, 2, 3, 4, 5, 6} e E = {{2, 5}, {6, 1}, {5, 3}, {2, 3}}.

- a) Grafo direcionado com arestas (2→5), (6→1), (5→3), (2→3)
- b) Grafo não direcionado com arestas {2–5}, {6–1}, {5–3}, {2–3}

Q9 - Seja um grafo G cujos vértices são os inteiros de 1 a 8 e os vértices adjacentes a cada ´ vértice são dados pela tabela abaixo:

Vértice	Vértices Adjacentes
1	2 3 4
2	1 3 4
3	1 2 4
4	1 2 3 6
5	6 7 8
6	4 5 7
7	5 6 8
8	5 7

(a) Desenhe o grafo G.
(b) Represente o grafo por meio de uma matriz de adjacência.
(c) Represente o grafo por meio de uma lista de adjacência.

Q9
Q10