PUC-Rio – Estruturas de Dados – INF1010
Prova 4 – Turma 3wb – 11/12/2017

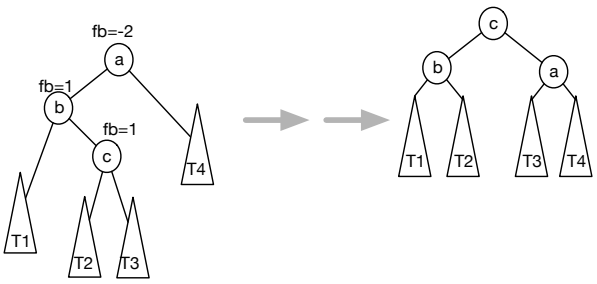
1. (1,5 pontos) Considere a representação abaixo para nós de uma árvore vermelho-negra:

```
typedef struct smapa Mapa;
struct smapa {
  int chave;
  int dado;
  char vermelho;
  struct smapa* esq;
  struct smapa *dir;
};
```

   Escreva uma função em C para determinar a propriedade de árvores rubro-negras de que um nó vermelho só pode ter filhos negros. A função deve ter o seguinte protótipo:

```
int filhosOK (Mapa *m);
/* Retorna 0 se existe algum nó vermelho com pelo menos um filho vermelho na árvore    */
/* com raiz em m, 1 em caso contrário.                                                  */
```

2. (1,5 pontos) Considere a operação de rotação dupla mostrada abaixo, para correção de árvore AVL desbalanceada. Diga quais serão os fatores de balanceamento finais dos nós a, b e c, *explicando* como podem ser calculados a partir da configuração inicial.

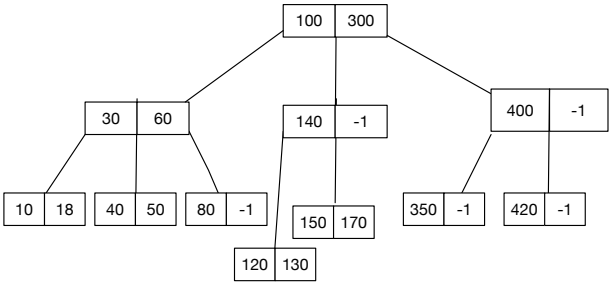


3. (1 ponto) Suponha que estamos implementando uma lista de prioridades máximas (max-heap) implementado com array, como visto em sala. Se soubermos que a ordem de inserção (um a um) foi:

   100-50-190-10-280-180-40-165-175

   Desenhe a árvore em cada passo (cada inserção) e no final mostre como ficará o array. E, se depois de inseridos esses elementos, fazemos duas retiradas sucessivas. Mostre como fica o array depois de cada retirada. Explique como chegou à sua resposta.

4. (1,5 pontos) Suponha a árvore 2-3 a seguir.



Como ficará a árvore se realizarmos cada uma das operações a seguir, *seguindo o algoritmo de inserção para árvores B visto na disciplina*? Cada uma das operações deve ser realizada sobre a árvore original. Desenhe (na folha a parte) a nova árvore para cada caso.

(a) inserção com chave 160

(b) inserção com chave 5

(c) remoção com chave 350

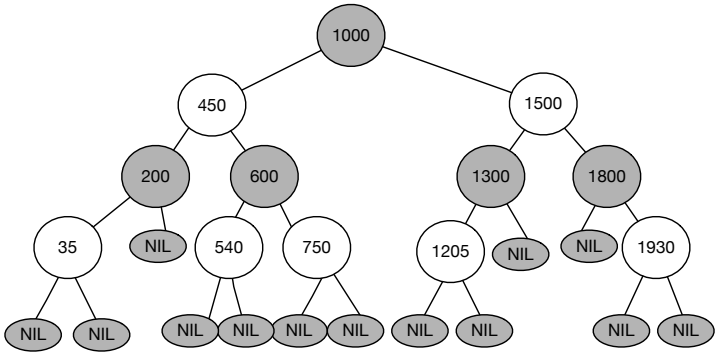
5. (2 pontos) Considere a estrutura de grafos vista nos laboratórios.

```
typedef struct _grafo Grafo;
typedef struct _viz Viz;
struct _viz {
  int noj;
  float peso;
  Viz*  prox;
};
struct _grafo {
  int nv;           /* numero de nos ou vertices */
  int na;           /* numero de arestas */
  Viz** viz;         /* viz[i] aponta para a lista de arestas vizinhas do no i */
};
```

Considere que estamos tratando grafos não direcionados conexos. Escreva uma função `temCiclos`, *usando a idéia de busca em profundidade e sem usar estruturas de união e busca*, que determine se um grafo g tem ciclos ou não. Sua função deve retornar 1 caso o grafo contenha ciclos e 0 caso contrário.

```
int temCiclos (Grafo* g);
```

6. (1 ponto) Suponha a árvore rubro-negra mostrada a seguir, onde nós em cinza representam os nós negros e nós em brancos representam os nós vermelhos.



Como ficará a árvore se realizarmos cada uma das operações a seguir, *seguindo o algoritmo de inserção para árvores rubro-negras visto na disciplina*? *Cada uma das operações deve ser realizada sobre a árvore original.* Desenhe (na folha a parte) a nova árvore para cada caso.

(a) inserção com chave 1400

(b) inserção com chave 1850

7. (1,5 pontos) Considere a implementação de tabelas hash explorada no laboratório 8, A estrutura de dados usada era a seguinte:

```
typedef struct smapa Mapa;
typedef struct {
  int chave;
  int dados;
  int prox;
} ttabpos;
struct smapa {
  int tam;
  ttabpos *tabpos;
};
```

A implementação trata conflitos através de encadeamento interno: Cada posição da tabela contém, além da chave e dos dados, um campo prox, apontando para o índice da tabela contendo a próxima chave mapeada para o mesmo valor de hash que a chave na posição atual. Se não houver essa "próxima"chave, o campo prox deve conter o valor -1. A inserção funciona como visto em aula e no laboratório: Caso a posição mapeada pela função de *hash* esteja ocupada, escolhe-se uma posição livre e, conforme o caso seja de conflito primário ou secundário, ou a nova entrada ou a entrada atual é "expulsa" para a posição escolhida.

Escreva a função `busca`, com protótipo abaixo, supondo a implementação de tabela descrita. Sua função pode fazer chamadas à função `hash`, que vc pode supor já implementada.

```
int busca (Mapa *m, int chave);
```