PUC-Rio – Estruturas de Dados – INF1010
Prova 4 – Turma 3wb – 16/12/2016

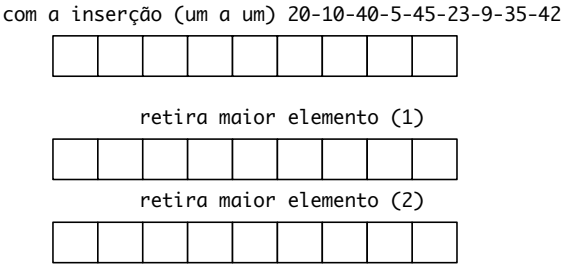
1. (2 pontos) Considere a representação de matrizes esparsas vista em laboratório, esboçada a seguir:

```
typedef struct smatrix tsm;
struct smatrix {
  int l, c; /* número de linhas e colunas */
  int nelems; /* número de elementos não nulos na matriz */
  int *vals; /* lista de valores da matriz (em ordem de linhas) */
  int *cols; /* cols[i] indica a coluna em que se encontra vals[i] */
  int *ri; /* ri[i] indica a posição do array vals onde começa a linha i */
           /* ri[i] = -1 quando a linha i é toda nula */
};
```

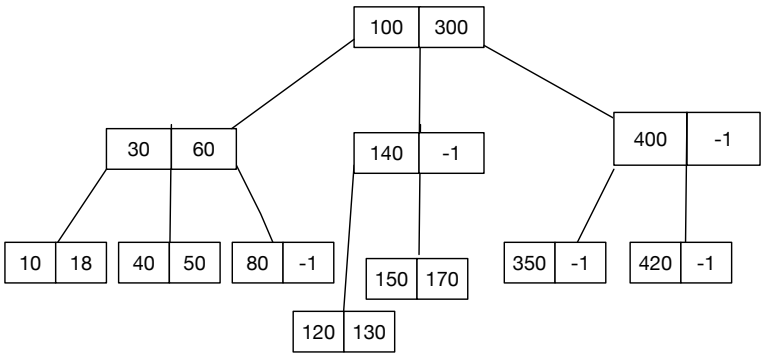
   Implemente a função `maxnaonulos`, com o protótipo abaixo, que retorna o número da linha com o maior número de elementos não nulos. Caso exista empate, sua função pode retornar o menor número de linha entre as linhas empatadas. Caso todas as linhas da matriz sejam nulas, sua função deve retornar -1.

```
int linhacommenosnulos (tsm* m);
```

2. (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 20-10-40-5-45-23-9-35-42, como ficará o array? Preencha abaixo. 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.



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

(b) inserção com chave 125

(c) remoção com chave 420

4. 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 */
};
```

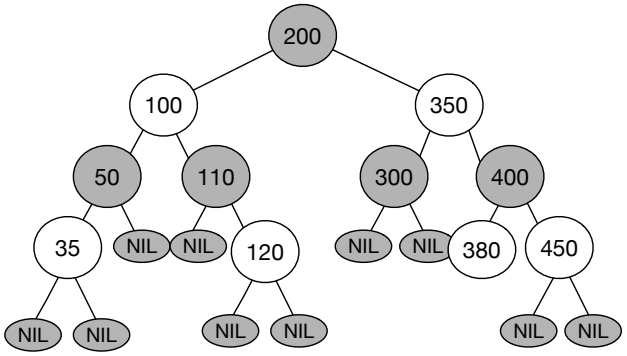
(a) (1,5 pontos)
Escreva uma função com a assinatura abaixo, que receba um grafo e um array `percursos` que indica caminhos (possivelmente disjuntos) dentro deste grafo e mostre (imprima) o caminho usado para chegar a cada nó *na ordem do percurso*. Sugiro usar recursão para facilitar a exibição dos caminhos na ordem pedida.
Cada posição i do array contém o último nó de um caminho até o nó i. Um valor de -1 em `percursos[i]` indica que i é o nó inicial nesse caminho. (*Atenção:* Nem todos os caminhos começam da mesma origem.)

```
void mostrapercursos (Grafo* g, int* percursos);
```

(b) (1 ponto) Escreva uma função com o protótipo abaixo que determina se o grafo contém ciclos ou não. Sua função deve utilizar uma estrutura de união e busca como auxiliar. Não esqueça de alocar e liberar memória quando necessário.

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

5. (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 391

(b) inserção com chave 115

6. (2 pontos) Considere a implementação de tabelas de dispersão usando *sondagem linear*: caso a posição indicada pela função de dispersão já esteja cheia, a inserção procura a primeira posição vazia a partir desta, voltando para a posição 0 caso chegue ao final da tabela de dispersão. A tabela contém um array de elementos onde cada um dos elementos contém a chave e um valor (ou ponteiro), e possivelmente outros campos. Nesse caso, a busca por determinada chave deve começar a partir da posição indicada pela função de dispersão e prosseguir também linearmente, como a inserção, até encontrar a chave procurada ou uma posição vazia. Caso a busca alcance uma posição vazia, a chave procurada não está na tabela.

```
typedef struct shash Mapa;
int hashfun (int chave);
struct selem {
  int chave;
  int valor;
};
struct shash {
  int tam; /* tamanho do array de pares (valor, chave) */
  int fp; /* ignore esse campo */
  struct selem *tabela;
};
```

Escreva a função `busca`, com protótipo abaixo, implementando o comportamento descrito. Suponha que as chaves são sempre positivas, e que as posições vazias da tabela são marcadas com chaves de valor -1. Sua função pode fazer chamadas a `hashfun`, que vc pode supor já implementada.

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