PUC-Rio – Estruturas de Dados – INF1010
Prova 2 – Turma 3wb – 7/11/2016

Responda as questões da prova nas folhas em separado distribuídas junto com a prova. As respostas podem ser escritas a lápis ou caneta. Indique claramente a questão que está respondendo e, quando cabível, os passos que seguiu para chegar a sua resposta.

Por favor guarde seu celular/tablet/o_que_for antes de começar a prova e não o utilize até sair da sala.

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

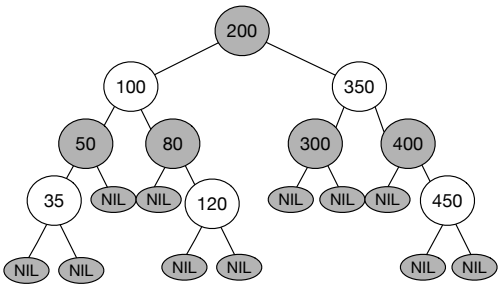
```
struct smatrix {
  int l, c; /* número de linhas e colunas */
  int nelems; /* número de elementos não nulos na matriz */
  int *vals; /* lista de valores da matriz (em ordem de linhas) */
  int *cols; /* cols[i] indica a coluna em que se encontra vals[i] */
  int *ri; /* ri[i] indica a posição do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

   Suponha que o arquivo de interface seja o seguinte:

```
esparsas.h:
typedef struct smatrix tsm;
tsm* createFromFile (FILE *h);
int numlinhas (tsm *m);
int numcolunas (tsm *m);
int elemento (tsm* m, int i, int j);
/* retorna o elemento na linha i e coluna j da matriz m */
```

   Implemente a função `elemento`. (Fique a vontade para escrever funções auxiliares, mas não suponha que elas já existem prontas.)

2. (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.

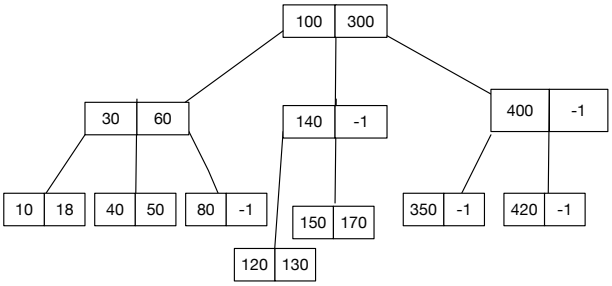


   (a) Como ficará a árvore se realizarmos cada uma das operações a seguir, *seguindo o algoritmo de inserção para árvores rubro-negras visto na disciplina*? Cada uma das operações deve ser realizada sobre a árvore original. Desenhe (na folha a parte) a nova árvore para cada caso.

      i. inserção com chave 500

      ii. inserção com chave 40

3. (2 pontos) Suponha a árvore B mostrada a seguir.



(a) 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.

i. inserção com chave 180
ii. inserção com chave 45
iii. remoção com chave 80
iv. remoção com chave 300

4. (1 ponto) Escreva uma função C que, dada uma árvore B m, com a representação interna a seguir, e uma chave c, imprima todas as chaves *estritamente menores que c* presentes na árvore. Sua função não deve fazer chamadas ou testes desnecessários.

```
typedef struct smapa Mapa;
struct smapa {
        int  kp, kg;
        Mapa  *pai;
        Mapa  *esq;
        Mapa  *meio;
        Mapa  *dir;
};
...
int menoresque (Mapa *m, int c);
```

5. (2 pontos) Em exercício do curso vimos tabelas de dispersão que tratam colisões criando uma lista encadeada de elementos cuja chave é mapeada para a mesma posição. Uma outra forma de tratar a colisão é por *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 é um array de elementos onde cada um dos elementos contém a chave e um ponteiro para dados, 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.

(a) Com essa técnica, o que acontece na remoção de um elemento? Como essa remoção vai interagir com a busca?

(b) Como podemos implementar a estrutura de dados para usar essa técnica e inserção, busca e remoção funcionarem direito?

(c) Qual o custo computacional da inserção, com essa técnica, se a tabela estiver vazia e a função de dispersão funcionar bem? E se a tabela estiver muito cheia?

(d) (desrelacionado com a técnica específica) Que características são importantes para a função de dispersão usada para mapear as chaves em uma tabela de dispersão?

6. (2 pontos) Imagine que queremos implementar a interface `mapa.h` de sempre com árvores vermelho-negras. Supondo que existem (estão implementadas e você pode utilizar) as funções `rotaciona_esq`

e `rotaciona_dir`, que fazem as rotações vistas *alterando apenas* campos `esq` e `dir` dos nós envolvidos, e a função `trocacores`, complete **apenas a parte** `'case RIGHTRED'` da função `corrigeDir`, chamada por insere depois de uma inserção na subárvore à direita de um nó `m`

```
static Mapa* corrigeDir (Mapa *m, Result* status) {
  switch (*status) {
    case OK:
      break;
    case RED:  /* filho à direita vermelho */
      if (m->vermelho) *status = RIGHTRED;
      else *status = OK;
      break;
    case RIGHTRED: /* filho à direita e seu filho à direita vermelhos */
      if (!m->esq) {  /* tio vazio - podemos tratar como preto */
      /* completar */
      }
      else if (m->esq->vermelho) {  /* tio vermelho */
      /* completar */
      }
      else { /* tio é preto */
      /* completar */
      }
      break;
    case LEFTRED: /* filho à direita e seu filho à esquerda vermelhos */
      if (!m->esq) {
      /* não completar */
      }
      else if (m->esq->vermelho) {
      /* não completar */
      }
      else {
      /* não completar */
      }
      break;
  }
  return m;
}
```

Boa prova!