PUC-Rio – Estruturas de Dados – INF1010
Prova 2 – Turma 3wa – 29/5/2019

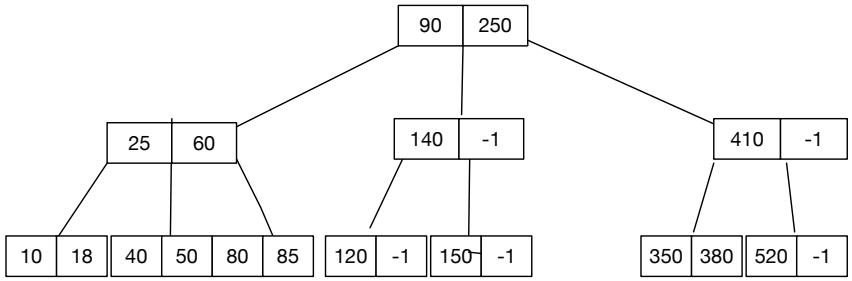
Responda as questões da prova nas folhas em separado distribuídas junto com a prova. As respostas podem ser escritas a lápis ou caneta. Indique claramente a questão que está respondendo e, quando cabível, os passos que seguiu para chegar a sua resposta.

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

Por favor entregue o enunciado juntamente com as folhas de resolução e coloque o nome em todas as folhas.

Boa prova!

1. (2 pontos) Suponha a árvore B 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. *Explique* como você chegou à árvore resultante.

(a) inserção com chave 400

(b) inserção com chave 70

(c) remoção com chave 520

(d) remoção com chave 150

2. (2 pontos) Considere a representação de tabelas de dispersão vista no laboratório, com encadeamento de conflitos por índices:

```
typedef struct smapa Mapa;
typedef struct {
  int chave; /* -1 indica ausência de chave */
  int dados;
  int prox;  /* -1 indica que não há próximo */
} ttabpos;
struct smapa {
  int tam;
  int ocupadas;
  ttabpos *tabpos;
};
```

Escreva uma função C que retorna o tamanho da maior cadeia de conflitos na tabela, isto é, o maior número de chaves encadeadas entre si. Procure evitar percursos inúteis, que analisem apenas cadeias parciais. (*Sugestão:* Para cada posição da tabela, verifique se ela constitui o início de uma cadeia, e caso sim, determine seu tamanho. Caso esse tamanho seja maior que o encontrado até agora, registre o novo tamanho.)

3. Considere a representação abaixo para uma árvore B de ordem TAM+1. O array `k` contém as chaves em cada nó e o array `sub` as subárvores filhas.

```
typedef struct smapa Mapa;
struct smapa {
        int k[TAM];
        Mapa  *sub[TAM+1];
};
```

   (a) (1 ponto) Escreva uma função C que, dado um mapa m com essa representação e uma chave max, mostre, em ordem crescente, todas as chaves presentes no mapa com valores menores que max.

   ```
   void mostrachavesinf  (Mapa *m, int max);
   ```

   (b) (1 ponto) Escreva uma função C que, dado um mapa m com essa representação e valores inteiros min e max, mostre, em ordem crescente, todas as chaves presentes no mapa com valores entre min e max (exclusive). (Use a função anterior como base!)

   ```
   void mostrachavesint  (Mapa *m, int min, int max);
   ```

4. (2 pontos) Considere a representação de lista de prioridade (*maxheap*) vista em laboratório e a implementação de remoção a seguir:

```
typedef struct _listaprio ListaP;
struct _listaprio {
  int max; /* tamanho maximo do heap */
  int pos; /* proxima posicao disponivel no vetor */
  int* prioridade; /* vetor das prioridades */
};
int listap_remove(ListaP* heap) {
  if (heap->pos>0) {
    int topo=heap->prioridade[0];
    heap->prioridade[0]=heap->prioridade[heap->pos-1]; heap->pos--;
    corrige_abaixo(heap->prioridade, 0, heap->pos);
    return topo;
  }
  else {
     printf("Heap VAZIO!"); return -1;
  }
}
```

   Escreva o código C da função *corrige_abaixo*.

5. (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 LEFTRED'` da função `corrigeEsq`, chamada por insere depois de uma inserção na subárvore à esquerda de um nó `m`.

```
static Mapa* corrigeEsq (Mapa *m, Result* status) {
  switch (*status) {
    case OK:
      break;
    case RED:
      if (m->vermelho) *status = LEFTRED; else *status = OK;
      break;
    case LEFTRED:
      if (!m->dir || !m->dir->vermelho) {  /* tio vazio/preto */
        /* COMPLETAR */
      }
      else if (m->dir->vermelho) { /* tio Ã© vermelho */
        /* COMPLETAR */
      }
      break;
    case RIGHTRED:
      if (!m->dir || !m->dir->vermelho) {  /* tio vazio/preto */
        /* NÃO COMPLETAR */
      }
      else if (m->dir->vermelho) {
        /* NÃO COMPLETAR */
      }
      break;
  }
  return m;
}
```