PUC-Rio – Estruturas de Dados – INF1010
Prova 4 – Turma 3wa – 8/7/2019

1. (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 `numcadeias` que retorna o número de cadeias de conflito na tabela, isto é, o número de vezes em que pelo menos duas chaves estão mapeadas para a mesma posição pela função de *hash*. chaves encadeadas entre si. (*Sugestão:* Para cada posição da tabela, verifique se ela constitui o início de uma cadeia ou não.

```
unsigned int hash (Mapa* m, int a);
/* supor implementada */
int numcadeias (Mapa *m);
/* implementar! */
```

2. Sobre heaps de prioridades:

   (a) (0,9 pontos) Diga quais dos arrays abaixo representam implementações válidas de um heap de prioridades máximas (max-heap) implementado com array, como visto em sala. Em caso de não ser uma representação válida, indique o por que.

400	300	310	200	305	270	15	150	110

100	90	50	80	70	40			

45	40	35	20	25	31	36	10	12

   (b) (0,6 pontos) Como ficará o heap a seguir depois de uma chamada a `listap_remove`?

455	400	421	301	352	91	100	105	180

3. (2,5 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 */
};
```

Escreva uma função com a assinatura abaixo, que receba um grafo g e dois nós, $n1$ e $n2$, pertencentes ao grafo, faça uma busca em largura a partir de $no1$ até encontrar $no2$ e retorne o custo (soma do peso das arestas) do caminho encontrado por sua busca em largura de $no1$ até $no2$. (*atenção:* O que queremos calcular é o custo de percorrer as arestas que levam $no1$ até $no2$ nessa visitação, e não a soma de todas as arestas visitadas pela busca.) Não esqueça de alocar e liberar memória quando necessário.

```
float custopercursolarg (Grafo* g, int no1, int no2);
```

Para seu código, pode supor que existe uma estrutura de fila já implementada oferecendo as funções a seguir.

```
SQ* enqueue(SQ* queue, int info);
SQ* dequeue(SQ* queue,int* info);
```

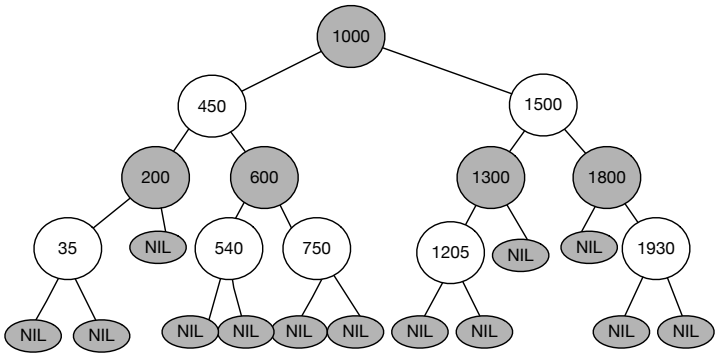
4. (2 pontos) Considere a representação de árvores 2-3 (árvores B de ordem 3) vista em sala. Escreva uma função C que, dado um mapa m com essa representação e valores inteiros min e max, mostre, em ordem crescente, todas as chaves presentes no mapa com valores entre min e max (exclusive). Não use visitas à árvore inteira pois isso não é necessário.

```
typedef struct smapa Mapa;
struct smapa {
  int  kp, kg;
  Mapa  *pai;
  Mapa  *esq;
  Mapa  *meio;
  Mapa  *dir;
};
...
void mostrachavesentre  (Mapa *m, int min, int max);
/* implementar supondo que vc está dentro de "mapa.c"*/
```

5. (1 ponto) Escreva uma função que calcula a altura de uma árvore avl usando a representação a seguir, usada em laboratório. Não use visitas à árvore inteira pois isso não é necessário.

```
typedef struct smapa Mapa;
struct smapa {
  int chave;
  int conteudo;
  short int bf;
  struct smapa* esq;
  struct smapa* dir;
};
...
int altura (Mapa *a);
```

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 1900

(b) inserção com chave 800