Prova – P2

Descrição

- Proponha solução para a prova abaixo, individualmente.

O que entregar

- Documento contendo nome, RA, enunciado da prova e as soluções propostas.

Quando entregar

- Até 2 horas após o início da prova.

Onde entregar

- No ambiente de interação da disciplina no Google Classroom, no link indicado.

Nome:_____
RA_____ Data:_____

Orientações Gerais

- **Tempo para elaboração da prova: 2:00h (2 horas e 00 minutos**).

Orientações Quanto a Notação, Nomes das Variáveis, e Estruturas

- Use os **mesmos nomes fornecidos no enunciado**. Utilize variáveis auxiliares temporárias, o tanto quanto for necessário. É só declarar e usar. Mas **não considere a existência de nenhuma outra variável permanente ou funções**, salvo **se explicitamente indicado no enunciado da questão.**
- Considere as **estruturas exatamente conforme definido no enunciado**, seja no texto da questão, seja nos diagramas.
- Para o desenvolvimento de algoritmos, use preferencialmente a notação de C ou C++.

Questão 1 (4 pontos)

Considere que o nó de uma árvore binária de busca tenha a seguinte declaração:

```
typedef struct node {
        int chave;
        struct node *esq;
        struct node *dir;
} Node;
```

Observe que no nó acima a informação sobre o tamanho da árvore não está armazenada.

Considere que o tamanho de uma árvore R é o número de nós da árvore, ou seja:
Para R = NULL o tamanho da árvore é 0;
Para R != NULL o tamanho da árvore é composto pelo número de nós da sub-árvore esquerda de R + número de nós da sub-árvore direita de R + 1

a) Escreva uma função que dada uma árvore R, retorne o tamanho da mesma. A função deve ter a seguinte assinatura:

```
//retorna o tamanho do no raiz.
int getSize(Node * R);
```

a) Escreva uma função que dado um nó raiz, retorne o tamanho do mesmo. A função deve ter a seguinte

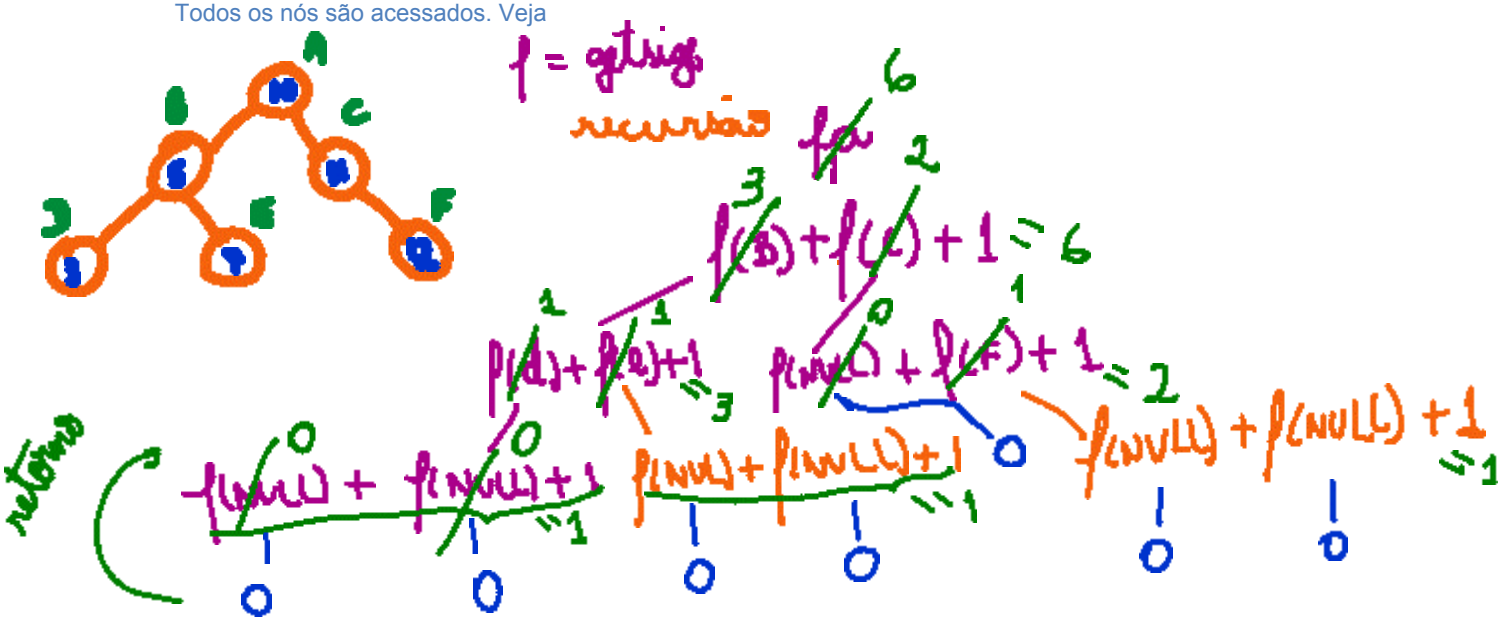
Resposta:

```
int getSize(Node * R){
    if (raiz == NULL)
        return 0;
   return ( 1 + getSize(R->esq) + getSize(R->dir)  );
}
```

Erros mais comuns (desconto de 1 ponto para cada)
- esquecer de condição de parada;
- somar somente um dos lados;

- esquecer de somar 1 para o nó corrente.

b) Qual é a ordem de eficiência de tempo a função do item (a).
Resposta:
O(n)
Todos os nós são acessados. Veja



Erros mais comuns (desconto de 1 ponto para cada)
- colocar ordem errada, por exemplo, O(h) ou O(log n)
- obs: se aluno falar (k*n + b) ok, está certo, pois a ordem fica O(n)

c) A função abaixo faz uma busca em uma Árvore Binária de Busca de raiz R. A função abaixo é ineficiente. Altere a função para que ela passe a ter a eficiência O(h), onde h é a altura da árvore binária.

```
// Retorna um ponteiro para o nó com a chave ch ou NULL
Node * busca (Node * R, int ch){
   if (R == NULL)
      return NULL;
   if (R ->chave == ch)
      return R;
   Node * resp1 = busca(R ->esq,ch);
   Node * resp2 = busca(R ->dir,ch);
   if(resp1!=NULL)
      return resp1;
   return resp2;
}
```

Reposta:
O código da questão procura desnecessariamente em ambos os lados da árvore. Para otimizá-lo basta verificar se a chave é menor do que a chave do nó corrente, se sim, procura a esquerda, senão procura a direita.

```
// Retorna um ponteiro para o nó com a chave ch ou NULL
Node * busca (Node * R, int ch){
   if (R == NULL)
      return NULL;
   if (R->chave == ch)
      return raiz;
```

```
   if(R->chave > ch)
       return  busca(R->esq,ch);
   return busca(R->dir,ch).
}
```

Erros mais comuns
- inverter lado ao procurar o elemento, por exemplo, procurar o maior na esquerda, desconto de 0,5 ponto
- não encontrar o erro (neste caso desconto de 1 ponto)

Questão 2 (3 Pontos)

Escreva uma função que decida se um **max-heap armazenado em um vetor v[0 .. m - 1]** é ou não **coerente**. Para ser um max-heap coerente o filho esquerdo de um nó tem que ser sempre menor que o filho direito, ou seja, para um nó i, v[fesq(i)] < v[fdir(i)]. Use as definições abaixo na sua função.

```
#define pai(i) ((i – 1) / 2 )
#define fesq(i) (i * 2  +  1)
#define fdir(i) (i * 2  +  2 )
```

Resposta:

```
// Retorna 1 se coerente e 0 caso contrário
// recebe um max-heap v de tamanho n
int ehcoerente(int * v, int n){
    int i;
    for (i = 0; i <= n/2; i++)
        if ( fesq(i) < n &&  fdir(i) < n && v[f(esq(i)] > v[f(dir(i)])
              return 0;
    return 1;
}
```

Erros mais comuns (-1 ponto por erro)
- não verificar se i tem filho esquerdo e direito antes de fazer a comparação;
- fazer a comparação entre esquerda e direita invertida.
- não retornar se é ou não max-heap

Questão 3 (3 Pontos)

a) Escreva uma **função de ordenação** que ordena um vetor v de tamanho n.

   void ordena(int * v, int n)

b) Qual é o custo computacional em número de operações da função implementada no item (a)? Qual é o custo dessa função em termos de memória da função implementada no item (a)?

c) Responda: A **busca binária** é mais ou menos eficiente do que a busca sequencial? Por que? explique.

Reposta:
Cada item desta questão vale 1 ponto.

As questões a) e b) são variáveis, podem ser implementadas ou ordenação por seleção, inserção ou método bolha.

A busca binária é mais eficiente que a busca sequencial, pois em cada passo reduz o problema pela metade. Consiste em dado um vetor ordenado, comparar o item procurado sempre com o valor do meio do subvetor atual e descartar a metade do mesmo. Descarta-se a metade superior, se o valor procurado for menor, ou descartar a metade inferior, caso contrário. A complexidade da busca binária é O ($\log_2(n)$). Já a busca sequencial não se necessita que o vetor esteja ordenado, mas todos os elementos do vetor são testados até que seja encontrado o mesmo no vetor ou todo o vetor seja testado. Assim, a busca sequencial é O (n).