MC202—Estruturas de Dados
Lista de Exercícios 2

Islene Calciolari Garcia

Primeiro Semestre de 2007

1 Árvores Binárias

Considere a seguinte declaração para os nós de uma árvore binária:

```
typedef struct no {
    int v;
    struct no *esq, *dir;
} No;
```

1. Escreva uma função que realiza um percurso em largura na árvore. Você pode usar uma fila auxiliar. `void largura(No* t);`

2. Escreva uma função recursiva que conta o número de folhas em uma árvore binária.

   `int conta_folhas(No* t);`

3. Escreva uma função recursiva que verifica se um valor v está presente na árvore t (considere que não há nenhuma garantia a respeito da ordem dos valores na árvore).

   `int busca(No* t, int v);`

4. Escreva uma função `espelho(t)` que retorna uma nova árvore `t`, mas com as sub-árvores esquerda e direita de **todos** os nós trocadas. A árvore original não deve ser alterada.

   `No* espelho(No* t);`

5. Em uma árvore de expressão, um nó que é um operando (uma letra entre `a` e `z`) não deve ter filhos e um nó que é um operador (`+`, `-`, `*` ou `/`) deve ter duas sub-árvores não vazias que correspondem a duas expressões válidas. Escreva uma função que verifica se uma árvore de expressão é válida, considerando também que uma árvore vazia é inválida e que a presença de quaisquer outros caracteres também tornam uma árvore inválida.

   ```
   int expr_valida(ap_no t);
   ```

6. Escreva a árvore binária correspondente à seguinte expressão: $(a+b)*(c-d)-e*f$. Escreva os elementos da árvore considerando os seguintes percursos pré-ordem e pós-ordem.

7. Seja `bal` o fator de balanceamento de um nó dado pela fórmula $h_d - h_e$, onde h_e é a altura da sub-árvore esquerda e h_d é a altura da sub-árvore direita. Escreva uma função recursiva que calcula o `bal` de todos os nós da árvore.

```
typedef struct no {
    int v;
    int bal;
    struct no *esq, *dir;
} No;
```

```
void calcula_bal(No* t);
```

Seria eficiente utilizar uma função auxiliar `altura` para calcular os fatores de balanceamento? Caso você ache que não, escreva uma solução mais eficiente.

8. Dada uma árvore com fatores de balanceamento calculados de acordo com a definição anterior, escreva uma função recursiva que retorna o valor máximo de **bal** na árvore.

```
int max_bal(No* t);
```

9. Desenhe a árvore binária que tem os percursos descritos abaixo.

Pré-ordem:	C	E	A	D	H	K	J	B	M	F	L	G	I
In-ordem:	D	A	K	H	E	C	B	J	L	F	M	G	I

Desenhe uma outra árvore que tenha o mesmo percurso em pré-ordem. Desenhe o percurso em pós-ordem das duas árvores.

2 Árvores de busca

```
typedef struct no {
    int chave;
    struct no *esq, *dir;
} No;
```

1. Escreva uma função **não-recursiva** que recebe uma árvore binária de busca `t` como parâmetro e retorna o apontador para o nó cuja chave possui o valor mínimo ou NULL caso a árvore esteja vazia.

```
No* minimo(No* t);
```

2. Escreva uma função **não-recursiva** que verifica a existência de algum elemento com chave negativa na árvore.

```
int existe_chave_negativa(No* t);
```

3. Escreva uma função que verifica se uma árvore de busca é válida.

```
int verifica_busca(No* t);
```

4. Considerando os algoritmos vistos em sala de aula, desenhe a árvore binária de busca resultante da inserção dos seguintes elementos (nesta ordem): $20, 25, 10, 5, 12, 22$ e 23. Remova a raiz da árvore. Quais elementos podem ser utilizados para substituir a raiz?

5. Escreva uma função que percorre uma árvore de busca e retorna uma lista ligada ordenada contendo todos os elementos da árvore. A sua função deve ter complexidade proporcional ao número de nós da árvore ($O(n)$). Você pode usar funções auxiliares.

```
typedef struct no_lista {
    int chave;
    struct no_lista* prox;
} No_lista;

No_lista* lista(No* t);
```

6. Considere que você dado um vetor ordenado você precisa construir uma árvore de busca que contenha os mesmos elementos. Como você faria a construção da árvore para evitar que esta ficasse desbalanceada?

```
No* vet2arv(int vet[ ], int n);
```

3 Heaps e árvores completas

Para a representação ligada de um heap utilize a mesma declaração da árvore binária (Seção 1).

1. Escreva uma função que dada a representação ligada de uma árvore completa, retorna a sua representação em um vetor. Suponha que o vetor passado como parâmetro tem espaço suficiente para armazenar a árvore.

```
void arv2vet(No* h, int vet[ ], int *n);
```

2. Escreva uma função que dada a representação de uma árvore completa em um vetor, retorna a sua representação ligada.

```
No* vet2arv(int vet[ ], int n);
```

3. Escreva uma função que verifica se uma árvore binária com representação ligada é completa.

```
int completa(No* arv);
```

4. Escreva uma função que retorna o valor do elemento mínimo em um heap de máximo representado em um vetor. Quantas comparações são necessárias? Como você faria para remover este elemento e manter o heap?

```
int valor_minimo(int heap_max[], int n);
```

5. Codifique o algoritmo Heapsort.

```
void heapsort(int vet[], int n);
```