Estruturas de Dados — MC202ABC
1º Semestre 2018

Instituto de Computação — UNICAMP

3ª Lista de Exercícios

Entregue os exercícios 1 e 2 até 10/06/2018 pelo SuSy.
Leia com cuidado as instruções abaixo.

Instruções:

- Os 2 exercícios deverão ser entregues separados em 2 tarefas diferentes no SuSy: o Exercício 1 deve ser entregue na tarefa `l3ex1` e o Exercício 2 deve ser entregue na tarefa `l3ex2`. Caso não faça algum dos exercícios, basta não submeter na tarefa referente ao exercício.
- A nota da lista será dada em relação aos 2 exercícios, apesar da submissão ocorrer em 2 tarefas diferentes do SuSy.
- Os exercícios podem ser digitados em um computador (preferencialmente) ou serem escritos à mão e digitalizados (você pode inclusive tirar uma foto). Porém, a entrega precisa ser feita necessariamente por meio de um arquivo **PDF** com tamanho de no máximo **2MB**. Outros formatos *não serão aceitos*.
- Garanta que o documento tenha qualidade o suficiente para ser lido. Documentos escritos à mão devem ter letras legíveis e feitos preferencialmente à caneta, já que o lápis tem pouco contraste para a digitalização. Assim, se possível, opte por fazer o exercício no computador ao invés de digitalizar.
- Caso tenha problemas com o tamanho do arquivo gerado, procure por um compressor de PDFs online. Existem vários serviços que recebem um PDF e geram o mesmo PDF com um tamanho menor. Porém, tome cuidado de garantir que o arquivo continue legível.
- Lembre-se de indentar corretamente o seu código para facilitar o entendimento.
- **Não se esqueça de colocar nome e RA em cada exercício.**
- A correção de cada exercício será enviada para o seu email institucional (da DAC). Garanta que há espaço na sua cota para receber os emails[1] e verifique o spam se necessário.

[1] Considere fazer o redirecionamento do seu email da DAC para o seu email pessoal: https://www.dac.unicamp.br/portal/estudantes/webmail-mais-informacoes.

Exercício 1 (`l3ex1`): Uma **árvore completa** é uma árvore em que todos os níveis, exceto o último, possui o máximo número de nós possível. No caso em que o último nível não possua o máximo número de nós possíveis, para a árvore ser considerada **completa**, todos os nós deste nível devem estar o mais à esquerda possível.

Considere que a estrutura apresentada no Código 1 representa cada nó de uma árvore binária. Considere que uma árvore vazia é representada como um ponteiro nulo. (Uma árvore vazia é uma árvore completa.) Implemente uma função em linguagem C que receba uma árvore como parâmetro e retorne um inteiro diferente de 0 se a árvore é completa ou *retorne 0 se a árvore não é completa.* Utilize o protótipo de função mostrada no Código 2. *Não* é permitido implementar funções auxiliares, ou seja, a função `eh_completa` deve ser autocontida (porém é permitido chamadas recursivas, se necessário).

```c
typedef struct no {
  int x;
  struct no *esq;
  struct no *dir;
} No;

typedef No * p_no;
```

Código 1: Estrutura utilizada para representar a árvore binária.

```c
int eh_completa(p_no raiz);
```

Código 2: Protótipo da função para verificar se uma árvore binária é completa.

Dica: Note que a função deve retornar um inteiro diferente de 0 caso a árvore seja completa. Portanto, o retorno pode ser **tanto positivo quanto negativo**, o que permite representar se a árvore/subárvore é completa e cheia ou se é completa e não cheia.

Exercício 2 (`l3ex2`): Note que existe uma grande semelhança entre uma árvore binária e uma lista duplamente ligada: Os nós de ambas contém dois ponteiros. O nó da árvore binária tem um ponteiro para o filho esquerdo e um para o filho direito, enquanto que o nó da lista duplamente ligada tem um ponteiro para o nó anterior e um para o próximo nó. Assim, as estruturas dos nós são idênticas, exceto pelo nome nos campos dos ponteiros. Portanto, a mesma `struct` pode ser usada na implementação de uma árvore binária de busca e em uma *lista duplamente ligada circular*.

Considerando a informação acima, escreva uma função em C que dada uma árvore binária de busca, *reorganiza os ponteiros da árvore* para torná-la uma lista duplamente ligada circular (sem cabeça), onde os elementos estão *ordenados em ordem crescente*. Em particular, sua função deve modificar a árvore considerando que o ponteiro para o filho esquerdo será modificado para apontar para o elemento anterior na lista e que o ponteiro para o filho direito será modificado para apontar para o próximo elemento na lista. Sua função deve devolver um ponteiro para o primeiro nó da lista (o de menor valor).

Considere o tipo `No` declarado no Código 1 representando os nós da árvore binária de busca (ABB). Use o protótipo declarado no Código 3 na implementação da função a realizar a transformação. Poderão ser implementadas funções auxiliares para a implementação da função `transforma_em_lista`.

```
1 p_no transforma_em_lista(p_no raiz);
```

Código 3: Protótipo da função a ser implementada para transformar uma ABB em uma lista duplamente ligada circular.

Atenção à informação de que o algoritmo deve *reorganizar os ponteiros da árvore*. Isso indica que não deve haver cópia dos nós da árvore, nem deve haver cópia de todos os ponteiros para os nós da árvore.