IMD0029 - Estrutura de Dados Básicas 1 – 2017.2

Prof. Eiji Adachi M. Barbosa

Atividade Avaliativa Prática em Laboratório – Recursão & Algoritmos de Busca

ANTES DE COMEÇAR, leia atentamente as seguintes instruções:

- Esta é uma atividade de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A atividade vale 10,0 pontos, com peso de 20% na 1ª unidade, e o valor de cada questão é informado no seu enunciado.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da atividade do candidato (nota igual a zero).
- Junto a este enunciado, você também recebeu uma estrutura de diretórios contendo um diretório para cada questão. Em cada um destes diretórios, já existe uma assinatura de função e uma função main com um pequeno teste executável. A solução da sua questão deverá seguir a assinatura da função já estabelecida. Ou seja, não mude esta assinatura. Se necessário, crie funções auxiliares com outras assinaturas, mas **não mude a assinatura da função original!**

Questão 1 (15%): Dada uma string S, implemente uma função **recursiva** que retorne uma outra string com os caracteres de S em ordem inversa. Obs.: Em C++, strings podem ser comparadas usando os operadores de comparação (<, <=, >, >=, ==, !=), e concatenadas usando o operador de soma (+ ou +=).

Questão 2 (15%): Dado um número natural N, implemente uma função **recursiva** que retorne uma string contendo a representação binária deste número.

Questão 3 (35%): Sequências bitônicas são aquelas que possuem duas sequências, sendo uma sequência inicial crescente, seguida de uma sequência decrescente. Ou seja, os elementos de uma sequência bitônica obedecem a seguinte relação:

$$A0 < ... < Ai-1 < Ai > Ai+1 > ... > An$$

Considere que um array bitônico é um array de inteiros sem repetições cujos elementos representam uma sequência bitônica (na relação acima, o i pode ser igual a 0, ou igual a n). Neste contexto, implemente uma função que consegue fazer uma busca num array bitônico. Isto é, implemente uma função que recebe como entrada um array bitônico A e um número inteiro K e retorna o índice i tal que A[i] == K; caso não seja encontrado elemento em A com valor igual a K, a função deverá retorna -1. Sua solução deverá ter **obrigatoriamente** complexidade $\Theta(\lg(n))$.

Questão 4 (35%): O problema Soma-3 é definido da seguinte forma: dado um array A e um inteiro X, buscam-se inteiros I, J, K tais que A[I] + A[J] + A[K] == X (admite-se a possibilidade de I, J e K serem iguais). Neste contexto, implemente uma função que apenas checa parte do problema Soma-3. Mais especificamente, implemente uma função que recebe um array de inteiro A e um inteiro X, e retorna verdadeiro caso existam inteiros I, J, K tais que A[I] + A[J] + A[K] == X; caso contrário, ela retorna falso. Sua solução deverá ter **obrigatoriamente** complexidade igual a $\Theta(n^2 * \log_2(n))$. Assuma nesta questão que o array de entrada sempre estará ordenado em ordem crescente.

ENTREGÁVEL

O entregável desta atividade deverá seguir a mesma estrutura de diretórios do código fonte que você recebeu com este enunciado, obviamente, contendo os arquivos fonte utilizados para construir sua solução nos diretórios de cada questão. Além disso, o diretório pai deverá ter o seu nome e matrícula, seguindo o padrão `<PRIMEIRO>_<SOBRENOME>-<MATRICULA>`. Por exemplo:

```
> JOAO_SILVA-200012345
   > q1
   > q2
   > q3
   > q4
```

Toda esta estrutura de diretórios, incluindo os arquivos fonte com sua solução, deverá ser compactada num arquivo .zip que também deverá seguir o padrão `<PRIMEIRO>_<SOBRENOME>-<MATRICULA>`. Este arquivo compactado deverá ser entregue via SIGAA até as **22:15**. **Este é um prazo fixo que não será estendido**, exceto em casos muito excepcionais (ex.: SIGAA fora do ar). Ou seja, entregas após este horário não serão aceitas. A atividade do SIGAA permite apenas um envio, portanto certifique-se de que está enviando a versão correta antes de anexar ao SIGAA.

CRITÉRIOS DE CORREÇÃO

Para a correção desta atividade, serão levados em consideração, dentre outros, os seguintes pontos:

- Obediência às regras definidas para as assinaturas de função e para o entregável (arquivo .zip), conforme especificado no enunciado desta atividade
- Existência de erros ou warnings de compilação do código fonte[1]
- Programas executam sem apresentar falhas e produzem os resultados esperados
- Soluções atendem critérios de complexidade, caso estabelecido no enunciado
- Apresentação e organização do código fonte entregue (identação, nome das variáveis, modularização do código em função, etc)

Obs.: Para cada questão, já há uma função main com um pequeno teste executável. Este é um teste simples que **não garante** a corretude da sua implementação. Ou seja, se sua implementação passou no teste executável disponibilizado junto a este enunciado, isto é apenas uma evidência mínima de que ela está correta. Para fins de correção, eu utilizarei outra bateria de testes mais completa, além de analisar manualmente o código produzido.

[1] Compile usando as flags `–Wall –pedantic –std=c++11`