BCC202 – Estruturas de Dados I (2023-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)

AULA PRÁTICA 9

- **Data de entrega: Até 30 de julho às 23:59:59.**

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Busca sequencial e Binária

Dado um vetor de entrada, você deve calcular o número de comparações necessárias (envolvendo este vetor) para encontrar um determinado elemento neste vetor utilizando:

- Busca Sequencial.
- Busca Binária

Atenção: você deve ordenar os dados de entrada para utilizar a busca binária. Utilize um algoritmo eficiente pois a entrada pode ser composta vetores grandes.

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. O vetor de inteiros **precisa** ser alocado e desalocado dinamicamente.

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes.

Especificação da Entrada e da saída

Sabe-se que a entrada é dada por vários casos de teste. A primeira linha possui um inteiro M, indicando o número de casos de teste. Cada caso de teste ocupa uma linha da entrada, e é composto pelo elemento X (número inteiro) a ser pesquisado, o número N de elementos do vetor, seguido dos N elementos (números inteiros) que deverão compor o vetor.

A saída de cada caso de teste deve ocupar uma linha, e ser composta pelo número de comparações utilizando busca sequencial seguido do número de comparações utilizando busca binária.

Entrada	Saída
3 1 10 1 2 3 4 5 6 7 8 9 10 5 9 8 3 28 3 9 929 -99 0 5 0 10 3 9 11 -32 5 6 7 8 2 1	1 3 9 1 10 3

Diretivas de Compilação

```
$ gcc -c busca.c -Wall
$ gcc -c pratica.c -Wall
$ gcc busca.o pratica.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed:  0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.