BCC202 – Estruturas de Dados I (2021-02)

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

AULA PRÁTICA 06 - MÉTODOS DE ORDENAÇÃO II

- **Data de entrega: Até 16 de maio às 23:59:59.**

- Procedimento para a entrega:.

1. Submissão: via ***run.codes***.
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 ***run.codes***.
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!

Atividade

Em um campeonato de basquete os times jogam todos entre si em turno único. A vitória vale dois pontos e a derrota vale um ponto (não há empates no basquete). Havendo empates na pontuação do campeonato fica na frente o time com melhor "saldo de cestas" que é dado pela razão entre o número de pontos marcados pelo time dividido pelo número de pontos recebidos – na improvável hipótese de um time vencer todos os jogos do campeonato sem levar cestas seu saldo de cestas é dado pelo número de pontos marcados. Persistindo o empate, leva vantagem quem marcou mais pontos. Ainda havendo empate, o time com o menor número de inscrição na liga fica na frente.

Sua tarefa neste problema é fazer um programa que recebe os resultados dos jogos de um campeonato e imprime a classificação final.

Especificação da Entrada e da saída

São dadas várias instâncias. Para cada instância é dado o número n de times no campeonato. O valor $n = 0$ indica o fim dos dados. A seguir vêm $n \times (n-1)/2$ linhas indicando os resultados das partidas. Em cada linha são dados quatro inteiros x, y, z e w. Os inteiros x e z representam os números de inscrição dos times na liga. Os inteiros y e w são, respectivamente, os números de pontos do time x e do time z na partida descrita.

Cada caso de testes é formado por diferentes instâncias. Para cada instância solucionada, você deverá imprimir um identificador "Instancia h" em que h é um número inteiro, sequencial e crescente a partir de 1. Na linha seguinte, deve ser impressa a permutação dos inteiros 1 a n da classificação do campeonato.

Um espaço em branco deve ser impresso entre cada um desses inteiros e uma linha em branco deve deve ser impressa entre as saídas de duas instâncias.

Entrada	Saída
5 1 102 2 62 1 128 3 127 1 144 4 80 1 102 5 101 2 62 3 61 2 100 4 80 2 88 5 82 3 79 4 90 3 87 5 100 4 110 5 99 0	Instancia 1 1 2 4 5 3

Diretivas de Compilação

```
$ gcc -c ordenacao.c -Wall
$ gcc -c pratica6.c -Wall
$ gcc ordenacao.o pratica6.o -o exe
```

Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. O problema deve ser resolvido pela implementação de um dos últimos três algoritmos de ordenação vistos em aula: merge sort, quick sort ou shell sort. As informações de cada time devem ser armazenadas em um tipo abstrato de dados criado especificamente para este fim. Um vetor dinâmico deste tipo abstrato de dados deve ser alocado, ordenado e posteriormente desalocado para armazenar os dados e resolver o problema. A função compare precisa ser implementada para realizar a comparação entre duas variáveis `struct team_t` e determinar qual delas é **menor**, de acordo com os critérios estabelecidos no enunciado do problema. Esta função deve ser invocada pelo algoritmo de ordenação implementado. Cada caso de teste pode ter várias instâncias e deve ser resolvido em até 1 segundo.

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivo-fonte (`ordenacao.h`, `ordenacao.h` e `pratica5.c`).
- Há no total **cinco** casos de teste. Você terá acesso (entrada e saída) a **dois** deles para realizar os seus testes.