Universidade Federal de Minas Gerais **Departamento de Ciência da Computação**	**Estrutura de Dados** **Profs Wagner Meira Jr e Marcio Costa**

Trabalho Prático 2

Fecho Convexo

Valor: 15 pontos
Data de entrega: 30/05/2023

Neste trabalho os alunos são convidados a implementar de forma eficiente algumas técnicas de geometria computacional. Em especial, a manipulação de dados bidimensionais e a determinação do fecho convexo de um conjunto de pontos.

1. Descrição

Você está trabalhando em uma indústria têxtil e seu chefe está tendo problemas para determinar os tamanhos das peças de tecido que devem ser compradas para o próximo bimestre. Assim, ele pede pela sua ajuda.

O seu chefe lhe apresenta uma lista com um conjunto de pontos no plano cartesiano e pede que vocẽ determine uma peça de tecido que consiga cobrir todos os pontos presentes na lista. Como o tecido é excessivamente caro e só é vendido em pedaços convexos (sem buracos e sem cavidades), você deve determinar o menor polígono convexo que encapsule todos os pontos apresentados.

Este polígono é conhecido como **fecho convexo** e temos diversos algoritmos para lidar com esse problema de forma direta. Dois dos mais famosos são: o *scan de Graham* e a *marchar de Jarvis*. A descrição desses dois métodos pode ser encontrada no livro texto na disciplina (Thomas H. *Cormen* - Algoritmos: Teoria e Prática) no capítulo 33, seção 3.

2. O que deve ser feito?

Você deverá implementar o sistema descrito acima. O seu programa deverá receber um arquivo contendo uma sequência de pontos no plano cartesiano, cada ponto em uma linha e com suas ordenadas separadas por espaço. Os pontos sempre possuem coordenadas inteiras. Como no exemplo a seguir:

```
0 0
0 1
1 0
1 1
```

O método scan de Graham utiliza um algoritmo de ordenação para ordenar os pontos dados de entrada. Dessa forma, cada aluno deve implementar o MergeSort, o InsertionSort e um outro método de ordenação linear (Counting, Bucket ou Radix) e integrar esses algoritmos de ordenação com o scan de Graham. Já o marchar de Jarvis não demanda algoritmo de ordenação e deve ser implementado dentro do mesmo programa. Temos então 4 configurações possíveis:

1. Graham + Mergesort
2. Graham + Insertion Sort
3. Graham + (Counting, Bucket ou Radix)
4. Jarvis

O programa deve ler um arquivo de entrada (descrito acima) como um argumento na linha de comando e deve executar as quatro configurações possíveis. A linha de comando deve ter o seguinte formato:

```
fecho arquivoentrada
```

Ao executar as 4 configurações, o programa deve medir o tempo consumido por cada uma dessas configurações (ignorando o tempo necessário para leitura do arquivo) e deve determinar o fecho convexo dos pontos de entradas. O programa então deve imprimir como saída (na saída padrão) os pontos que representam os vértices do fecho conveo e os os tempos de cada um dos métodos em segundos (com 3 casas decimais). Por exemplo (para o arquivo de entrada dado como exemplo):

```
FECHO CONVEXO:
0 0
0 1
1 0
1 1

GRAHAM+MERGESORT: 1.200s
GRAHAM+INSERTIONSORT: 1.500s
GRAHAM+LINEAR: 1.004s
JARVIS: 0.992s
```

Observe que o programa deve ser robusto e tolerante a erros. Em especial, ele deve avisar o usuário sobre problemas em operações matemáticas, por exemplo: divisão por zero. O programa também deve tratar problemas de utilização como, por exemplo, o arquivo especificado para leitura não existir.

3. O que deve ser implementado

Você terá que implementar um Tipo Abstrato de Dados (TADs)/Classe para representar os objetos geométricos Ponto, Reta e Fecho Convexo. Observe que existe uma relação entre esses elementos que pode facilitar a implementação dos mesmos. O TAD proposto deve ser utilizado para implementar as 4 configurações.

4. Entregáveis

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é terminantemente vetado.

Você DEVE utilizar a estrutura de projeto abaixo junto ao Makefile disponibilizado no Moodle:

- TP

    |- src

    |- bin

    |- obj

    |- include

    Makefile

A pasta **TP** é a raiz do projeto; a pasta **bin** deve estar vazia; **src** deve armazenar arquivos de código (*.c, *.cpp, ou *.cc); a pasta **include**, os cabeçalhos (*headers*) do projeto, com extensão ***.h**, por fim a pasta **obj** deve estar vazia. O **Makefile** disponibilizado no *Moodle* deve estar na **raiz do projeto**. A execução do **Makefile** deve gerar os códigos objeto ***.o** no diretório **obj** e o executável do TP no diretório **bin**.

Existe no Moodle um vídeo onde um dos monitores do semestre passado discute sobre a função e uso de makefiles.

5. Documentação

A documentação do trabalho deve ser entregue em formato **pdf** e também **DEVE** seguir o modelo de relatório que será postado no Moodle. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são apresentados:

Título, nome, e matrícula.
Introdução: Contém a apresentação do contexto, problema, e qual solução será empregada.
Método: Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
Análise de Complexidade: Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
Estratégias de Robustez: Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
Análise Experimental: Apresenta os experimentos realizados em termos de desempenho computacional, assim como as análises dos resultados.
Conclusões: A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
Bibliografia: Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.

Instruções para compilação e execução: Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).

***Número máximo de páginas:** 10

A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

Evite a descrição literal do código-fonte na documentação do trabalho.

Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

6. Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no Moodle. A entrega deve ser feita **em um único arquivo** com extensão **.zip**, com nomenclatura nome_sobrenome_matricula.zip, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais. O arquivo **.zip** deve conter a sua documentação no formato **.pdf** e a estrutura de projeto descrita no início da Seção 4.

7. Avaliação

O trabalho será avaliado de acordo com:

- Definição e implementação das estruturas de dados e funções - (30% da nota total)
- Corretude na execução dos casos de teste - (20% da nota total)
- Apresentação da análise de complexidade das implementações - (15% da nota total)
- Estrutura e conteúdo exigidos para a documentação - (20% da nota total)
- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Cumprimento total da especificação - (5% da nota total)

AVISO: Se o programa submetido não compilar[1], seu trabalho não será avaliado e sua nota será 0. Observe que a plataforma de referência dos trabalhos é Linux Ubuntu. Trabalhos entregues com atrasos sofrerão penalização de 2^{d-1} pontos, com d = dias de atraso.

Pontuação Bônus: Os alunos que apresentarem uma animação gráfica do funcionamento dos métodos, ou seja, à medida que seu programa vai produzindo as

[1] Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

retas do fecho convexo, elas vão sendo desenhadas na tela poderão receber até 5 pontos extras, a depender da qualidade da sua animação. Uma sugestão é usar a "graphics.h".

8. Considerações Finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é CRIME. Trabalhos onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na sessão de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos**.

FaQ (Frequently asked Questions)

1. **Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e etc..., do C++? NÃO**
2. Posso utilizar o tipo String? SIM.
3. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO
4. Posso utilizar alguma biblioteca para tratar exceções? SIM.
5. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
6. As análises e apresentação dos resultados são importantes na documentação? SIM.
7. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO
8. Posso fazer o trabalho em dupla ou em grupo? NÃO
9. Posso trocar informações com os colegas sobre a teoria? SIM.
10. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM.
11. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.
12. Posso utilizar a biblioteca de matemática (math.h)? SIM
13. Posso utilizar alguma biblioteca para operações de álgebra linear? NÃO.