Trabalho Prático 2
Fecho Convexo

Wagner de Oliveira Ferreira Duarte - 2020027229
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

wagnerofduarte@gmail.com

1. Introdução

A documentação a seguir propõe explicar, de forma sucinta, como foi desenvolvido o trabalho prático 2. A proposta do projeto é criar um sistema que seja capaz de ler um arquivo possuindo diversos pontos com coordenadas cartesianas **x** e **y** e retornar ao usuário qual é o fecho convexo correspondente, ou seja, o menor polígono convexo que engloba todos os pontos dados. Deve-se utilizar dois algoritmos para isso: o *Graham Scan* e *Jarvis March*. O algoritmo GrahanScan precisa de um método de ordenação em subrotina para funcionar, logo 3 métodos de ordenação foram implementados em conjunto: o Insertion sort, Merge sort e o Radix sort. Por fim, deve-se mostrar ao usuário o fecho convexo e o tempo gasto pelo computador para realizar cada um desses algoritmos. Segue na documentação como foi desenvolvido.

2. Método

2.1. Configurações da máquina

Sistema Operacional: macOS - Ventura 13.3.1 (a)
Linguagem de Programação: C++
Compilador: G++
Processador: Apple M1 Pro
Memória RAM: 16 GB

2.2. Estruturas de dados

Nesse projeto, somente uma TAD foi utilizada. Tal estrutura utilizada foi a ***Lista Encadeada***:

Lista Encadeada: A lista encadeada foi utilizada para armazenar dinamicamente cada ponto fornecido pelo arquivo, assim como armazenar as retas do fecho convexo calculado. Simultaneamente foram criadas dois tipos

de células para essa lista, de forma que ela possa armazenar, ou um ponto, ou uma reta.

2.3. Classes

Foram criadas as seguintes classes para execução do programa:

FechoConvexo: Essa classe objetiva armazenar as Retas do Fecho convexo encontrado pelos algoritmos *Scan de Graham* e *Marchar de Jarvis*. Possui uma Lista de Retas como atributo, que armazena tais retas.

AlgoritimosFechoConvexo: Essa classe foi criada para armazenar os algoritmos de *Scan de Graham* e a *Marchar de Jarvis*, de forma que para o uso de tais funções, basta instanciar essa classe e fazer as chamadas necessárias. Ela também possui uma função que calcula as orientações de três pontos.

Reta: Essa classe existe para armazenar uma única reta que compõe o fecho convexo. Possui dois pontos como atributos, que guardam as propriedades de tais pontos.

Ponto: Essa classe objetiva armazenar um ponto, com coordenadas x e y, passadas pelo arquivo lido no início da execução do programa. Possui outro atributo importante, que é o ângulo polar entre o ponto em questão, e o menor dos pontos da Lista de Pontos (tal propriedade é crucial para o algoritmo *Scan de Graham* rodar corretamente).

Lista: Essa classe implementa a TAD de *Lista Encadeada*, com algumas de suas várias formas de inserção, remoção, etc. A principal modificação feita nessa TAD, é a possibilidade de escolher qual tipo de célula será utilizada, no momento da instanciação do objeto

CelulaListaPonto: Essa classe representa uma das opções de célula que a classe Lista pode ter. Essa em específico armazena um Ponto qualquer (a classe Ponto será descrita em breve).

CelulaListaReta: Essa classe representa uma das opções de célula que a classe Lista pode ter. Essa em específico armazena uma Reta qualquer (a classe Ponto será descrita em breve).

AlgoritimosDeOrdenacao: Essa classe foi criada para armazenar os algoritmos de ordenação, de forma que para o uso de tais funções, basta instanciar essa classe e fazer as chamadas necessárias. Os algoritmos de ordenação implementados foram o *Insertion Sort*, *Merge Sort* e *Radix Sort*. Também foram implementadas nessa classe algumas sub-rotinas desses algoritmos.

Arquivos: Essa classe foi criada para ler e retornar ao programa o conteúdo do arquivo passado como parâmetro.

2.4. Funcoes Principais

O projeto possui diversas funções e métodos, todas necessárias para o funcionamento integrado do sistema. Apesar disso, algumas dessas funções possuem uma relevância extra-importante, sendo elas:

- **Ler Arquivo**
- **Algoritmos de Ordenacao**
- **Algoritmos de Fecho Convexo**

Ler Arquivo

- **Descrição:** Esse método da classe *Arquivos*, objetiva ler cada linha do arquivo de entrada e fazer as devidas conversões para ser armazenado em um objeto da classe *Ponto*.
- **Complexidade de Tempo:** Essa função depende diretamente da quantidade de linhas presente no arquivo, pois cada uma delas será lida. Logo, a sua complexidade é ***O(n)***

Algoritmos de Ordenacao

- **Descrição:** Os algoritmos de ordenação utilizados no projeto foram o *Insertion Sort*, *Merge Sort* e *Radix Sort*. Cada uma dessas heurísticas objetivam ordenar um conjunto de dados, porém possuem ordem de complexidade de tempo diferentes.
- **Complexidade de Tempo:** O método do Insertion Sort possui, em seu pior caso, complexidade ***O(nˆ2)***, pois precisa comparar cada elemento com todos os outros e realocar na lista de elementos se necessário. Já o Merge Sort possui complexidade ***O(n logn)***, já que tal algoritmo utiliza da técnica de dividir para conquistar (que reduz drasticamente o tempo de execução). Por fim, o Radix Sort possui complexidade ***O(d * (n + k))***, onde "*n*" é o número de elementos a serem ordenados, "*d*" é o número de dígitos nos elementos e "*k*" é o número de valores possíveis para cada dígito. No contexto desse trabalho, ***k*** é conhecido, e seria 10 dígitos, pois está sendo utilizada a base 10 de numeros; ***n*** seria a quantidade de linhas/pontos do arquivo; e ***d*** dependerá da quantidade de casas decimais presentes nas coordenadas dos pontos fornecidos.

Algoritmos de Fecho Convexo

- **Descrição:** Os algoritmos de fecho convexo utilizados no projeto foram o scan de graham e marcha de Jarvis. Ambos retornam um objeto da classe FechoConvexo, porém possuem complexidades distintas.
- **Complexidade de Tempo:** O algoritmo de marcha de jarvis, diferentemente do scan de graham, não precisa de um método de ordenação embutido para funcionar. Além disso possui uma ordem de complexidade de *O(hn)*, em que ***n*** é a quantidade de pontos fornecidos e ***h*** é a quantidade de pontos do fecho convexo resultante (ou seja, caso todos os pontos fazem parte do fecho, sua complexidade será *O(nˆ2)*, que define o pior caso. Já a complexidade do Scan de Graham, dependerá bastante do método de ordenação utilizado em sua subrotina. A ordenação dos pontos usando um algoritmo eficiente, como o Mergesort, tem uma complexidade média de ***O(n log n)***, por exemplo.

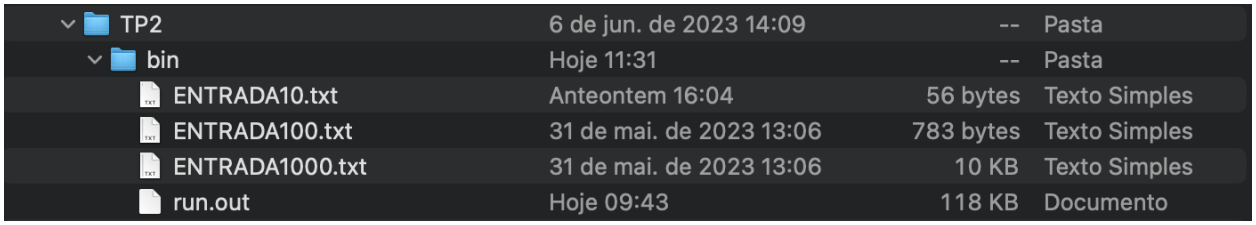
3. Conclusão

Esse projeto foi, de fato, uma oportunidade de aprofundar no assunto de algoritmos de ordenação e de conhecer um outro tipo de algoritmos, sendo eles os algoritmos para encontrar o fecho convexo de um conjunto de pontos. Utilizar tais algoritmos, além de fazer as devidas adaptações a cada um deles, foi uma experiência de grande aprendizado.
Espera-se que com esta documentação, entender a lógica e ideias por trás do código se torne mais simples e de clara compreensão.

4. Instruções de Execução

Para que o programa rode corretamente, tais condições devem ser satisfeitas:

- Os arquivos de teste precisam ser inseridos na pasta *bin* na pasta TP2:

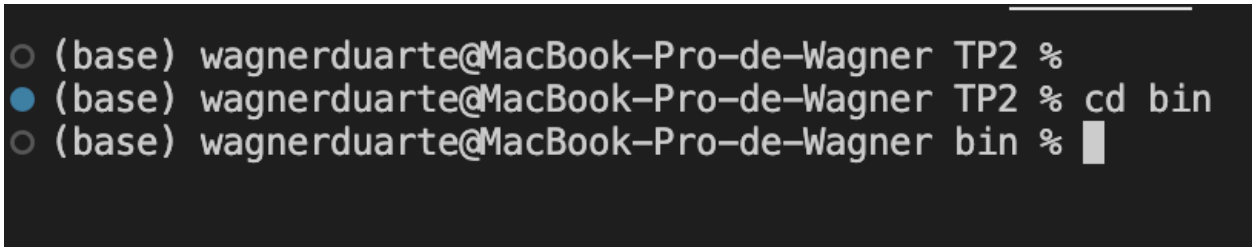


*Nessa imagem, o arquivo "ENTRADA10.txt", "ENTRADA100.txt" e "ENTRADA1000.txt" são exemplos de arquivos que possuem os pontos para formar o fecho convexo.

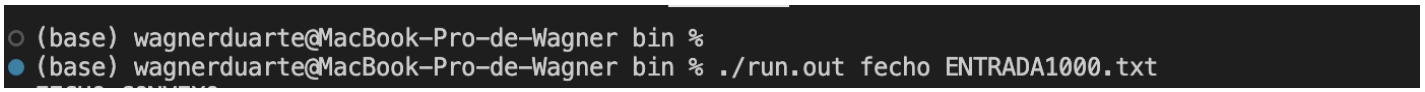
- Para compilar os arquivos, basta usar o comando *make*, no diretório da pasta TP2:

```
(base) wagnerduarte@MacBook-Pro-de-Wagner TP2 % make
g++ -std=c++11 -g -Wall -c src/AlgoritimosFechoConvexo.cpp -o obj/AlgoritimosFechoConvexo.o -I./include/
g++ -std=c++11 -g -Wall -c src/AlgoritimosOrdenacao.cpp -o obj/AlgoritimosOrdenacao.o -I./include/
g++ -std=c++11 -g -Wall -c src/Arquivos.cpp -o obj/Arquivos.o -I./include/
g++ -std=c++11 -g -Wall -c src/CelulaListaPonto.cpp -o obj/CelulaListaPonto.o -I./include/
g++ -std=c++11 -g -Wall -c src/CelulaListaReta.cpp -o obj/CelulaListaReta.o -I./include/
g++ -std=c++11 -g -Wall -c src/FechoConvexo.cpp -o obj/FechoConvexo.o -I./include/
g++ -std=c++11 -g -Wall -c src/Lista.cpp -o obj/Lista.o -I./include/
g++ -std=c++11 -g -Wall -c src/Ponto.cpp -o obj/Ponto.o -I./include/
g++ -std=c++11 -g -Wall -c src/Reta.cpp -o obj/Reta.o -I./include/
g++ -std=c++11 -g -Wall -c src/main.cpp -o obj/main.o -I./include/
g++ -std=c++11 -g -Wall -o ./bin/run.out  ./obj/AlgoritimosFechoConvexo.o  ./obj/AlgoritimosOrdenacao.o  ./obj/Arquivos.o  ./obj/CelulaListaPonto.o
  ./obj/CelulaListaReta.o  ./obj/FechoConvexo.o  ./obj/Lista.o  ./obj/Ponto.o  ./obj/Reta.o  ./obj/main.o
(base) wagnerduarte@MacBook-Pro-de-Wagner TP2 %
```

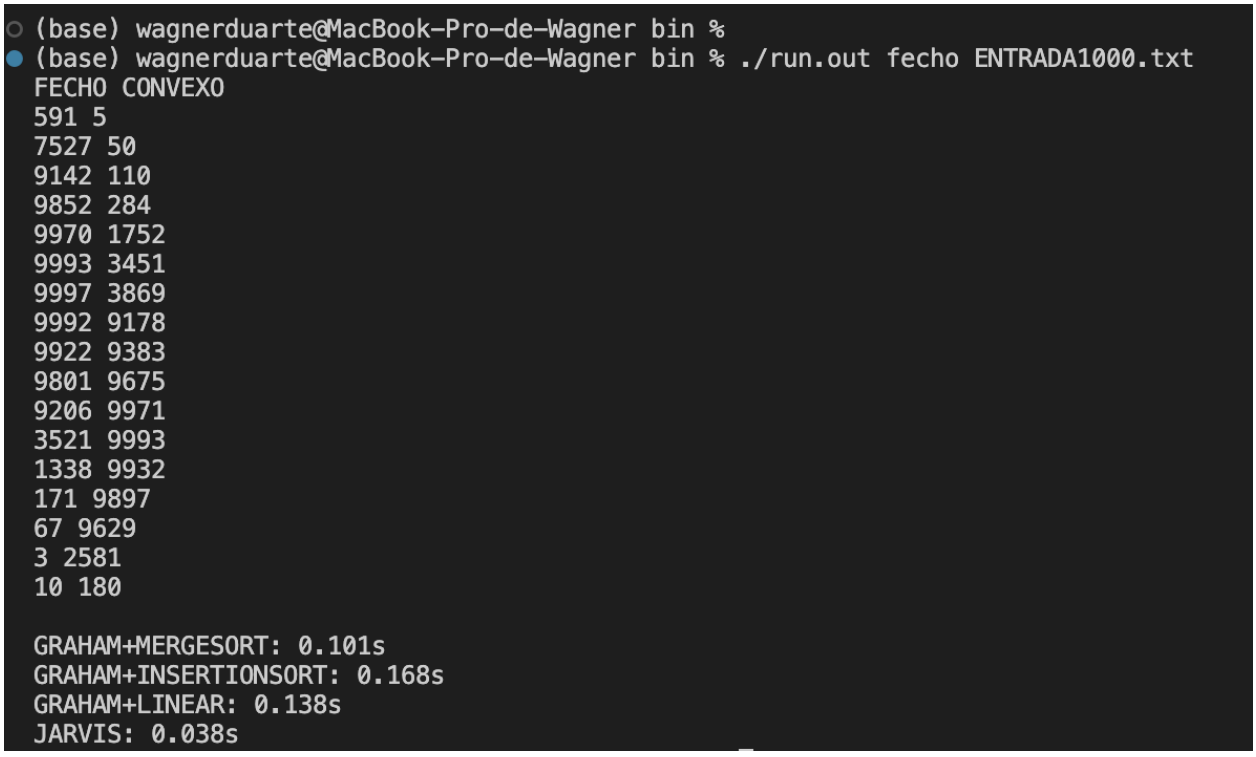
- A sua execução deve acontecer na pasta *bin* de TP2:



- O nome do arquivo à ser lido deve ser passado como parâmetro de argv[2], no seguinte formato:
    - ./run.out fecho NomeDoArquivo



- Com esses passos, a saída com o fecho e os tempos de cada algoritmo será impressa na tela:



5. Bibliografia e Referências

Vídeos:

- *https://www.youtube.com/watch?v=XiuSW_mEn7g*
- *https://www.youtube.com/watch?v=SBdWdT_5isI&t=355s*
- *https://www.youtube.com/watch?v=7zuGmKfUt7s*
- *https://www.youtube.com/watch?v=5prE6Mz8Vh0&t=26s*
- *https://www.youtube.com/watch?v=S5no2qT8_xg&t=419s*
- *https://www.youtube.com/watch?v=nBvCZi34F_o*

Livros:

- Thomas H. Cormen - Algoritmos: Teoria e Prática

Aulas e slides disponibilizados também foram amplamente utilizados.