Trabalho Prático 2
Fecho Convexo

Isabela Saenz Cardoso

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

isabelasaenz@ufmg.br
Matrícula: 2021040032

1. Introdução

O programa implementa algoritmos para calcular o fecho convexo de um conjunto de pontos no plano cartesiano. Um fecho convexo é o menor polígono convexo que contém todos os pontos de um conjunto.

Para encontrar os pontos que formam o perímetro do fecho convexo, são utilizados o algoritmo de scan de Graham e o algoritmo de Marcha de Jarvis.

2. Método

2.1. Configurações da máquina e do ambiente

- Máquina virtual com Ubuntu 22.04.2
    - Memória principal: 3000MB
    - Processadores: 2
- Compilador G++ / GNU Compiler Collection
- As animações dos resultados podem ser feitas usando o gnuplot

2.2. Estruturas de dados

struct Ponto: Representa um ponto no plano cartesiano com coordenadas x e y.

struct Reta: Representa uma reta formada por dois pontos.

struct FechoConvexo: Representa um fecho convexo, composto por um conjunto de pontos e seu tamanho

2.3. Funções

adicionarPonto: Adiciona um ponto ao conjunto de pontos do fecho convexo. Essa função recebe a referência de um vetor dinâmico de pontos, o tamanho atual do vetor e as coordenadas x e y do novo ponto. A função cria um novo vetor com tamanho atualizado, copia os pontos existentes e adiciona o novo ponto no final. Em seguida, ela deleta o vetor antigo e atualiza a referência para o novo vetor e incrementa o tamanho.

orientacao: Determina a orientação de três pontos no plano cartesiano. Esta função recebe três pontos como argumentos e calcula o valor do produto cruzado. Retorna 0 se os pontos

são colineares, 1 se a orientação é à direita (sentido horário) e -1 se a orientação é à esquerda (sentido anti-horário).

Funções de ordenação: São implementadas as funções merge, mergeSort, insertionSort e countingSort para ordenar os pontos com base nas coordenadas x e y. A função sortPontos é uma função auxiliar que utiliza insertionSort para ordenar os pontos.

encontrarPontoInicial: Encontra o ponto inicial para o algoritmo de Jarvis. Esta função recebe um array de pontos e seu tamanho e retorna o índice do ponto com menor coordenada y. Se houver mais de um ponto com a menor coordenada y, o ponto com menor coordenada x é escolhido.

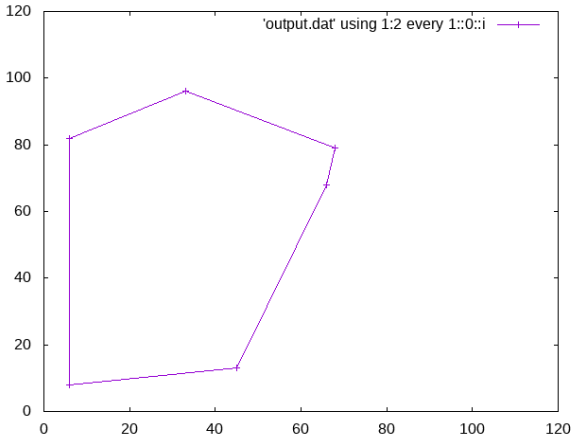
scanGraham: Implementação do algoritmo de Graham para encontrar o fecho convexo. Esta função recebe um array de pontos, seu tamanho e o tipo de ordenação a ser utilizado. A função retorna um objeto FechoConvexo contendo o array de pontos do fecho convexo e seu tamanho.

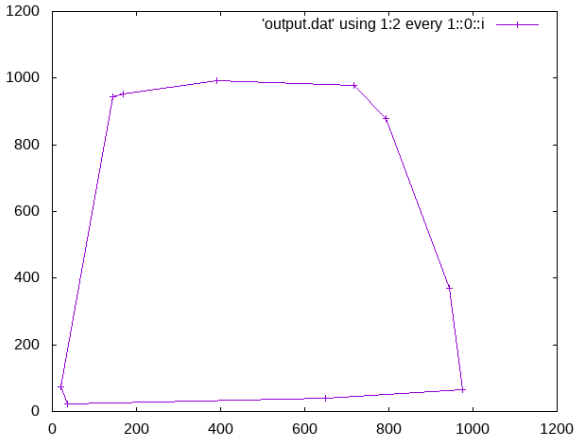
marcharJarvis: Implementação do algoritmo de Jarvis para encontrar o fecho convexo. Esta função recebe um array de pontos e seu tamanho. A função retorna um objeto FechoConvexo contendo o array de pontos do fecho convexo e seu tamanho.

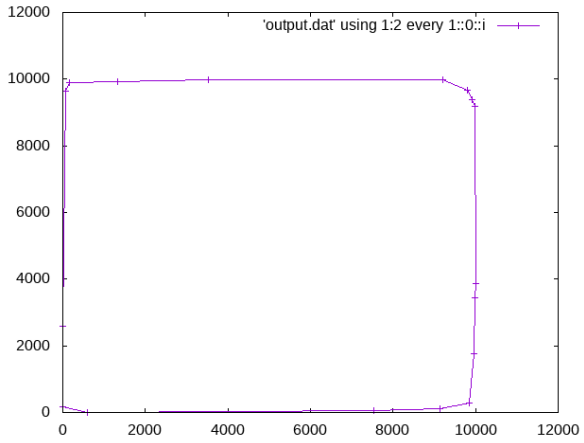
2.5. Animações

As animações foram feitas usando a biblioteca *gnuplot* e são aplicadas aos exemplos de 10, 100 e 1000 linhas, que podem ser encontrados dentro da pasta *input*. A execução do programa gera um arquivo *output.dat* e, ao executar o comando `gnuplot utils/<arquivo com comandos>`, um GIF mostra o processo dos pontos formando as retas na mesma ordem que as coordenadas são definidas pelo programa.

Imagens estáticas das animações geradas:







3. Análise de complexidade

3.1. Algoritmo de Graham:

Complexidade de tempo: O(nlog(n)) + O(n) = O(nlog(n)), onde n é o número de pontos no conjunto.

Complexidade de espaço: O(n), onde n é o número de pontos no conjunto.

3.2. Algoritmo de Jarvis:

Complexidade de tempo: O(n*h), onde n é o número de pontos no conjunto e h é o número de pontos no fecho convexo. Isso ocorre porque, para cada ponto do fecho convexo, todos os pontos restantes são percorridos para encontrar o próximo ponto no fecho

Complexidade de espaço: O(n), onde n é o número de pontos no conjunto.

O algoritmo de Graham é geralmente mais rápido para conjuntos de pontos grandes. No entanto, em casos específicos em que h é muito pequeno, o algoritmo de Jarvis pode ser mais rápido. A complexidade de espaço de ambos os algoritmos é a mesma, O(n), pois os pontos no fecho convexo são armazenados em uma estrutura de dados.

4. Estratégias de Robustez

Verificação de entradas: Verifica se as entradas fornecidas são válidas e se os pontos são distintos antes de realizar os cálculos.

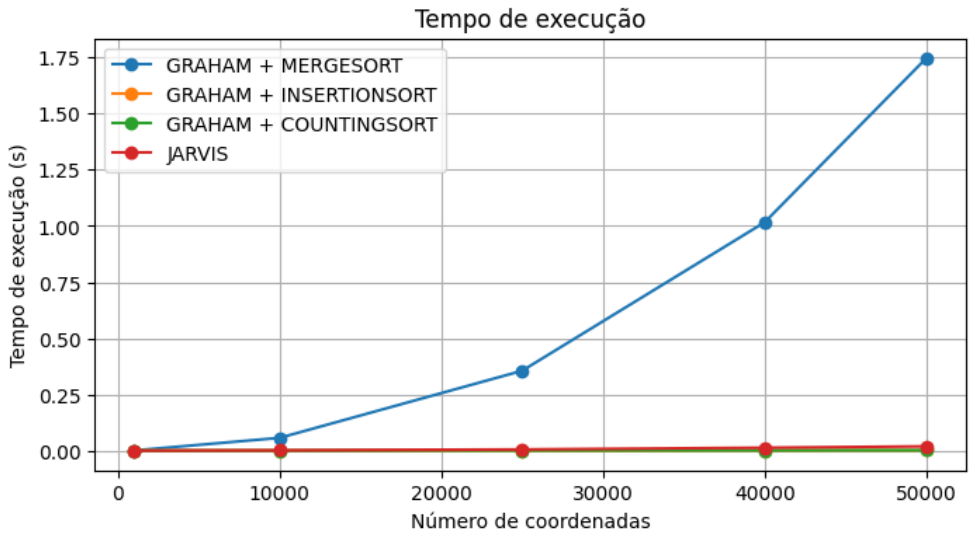
Tratamento de casos especiais: Trata casos especiais, como pontos colineares e pontos coincidentes, para garantir a correta formação do fecho convexo.

5. Análise Experimental

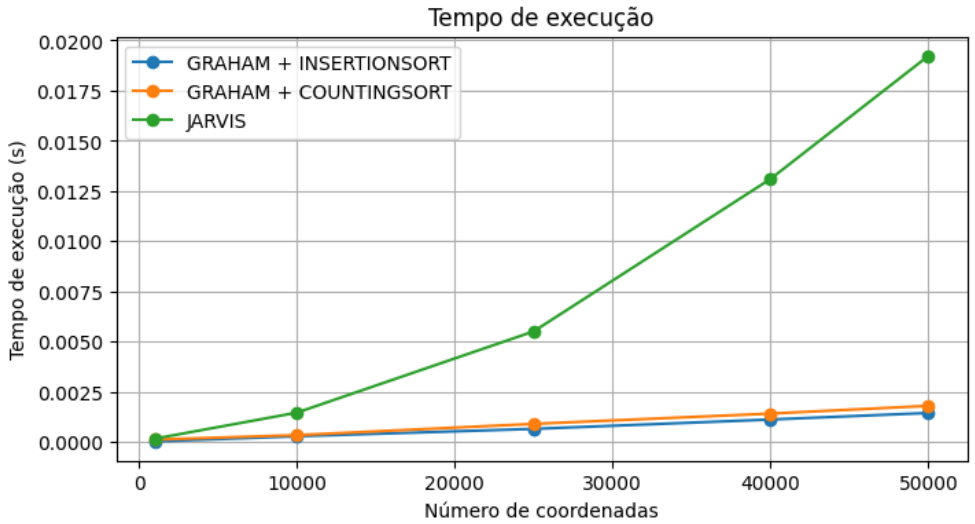
O método de cronometragem especificado no enunciado do trabalho (3 casas decimais) normalmente tem como resultado 0.000s, devido ao baixo tempo de execução das funções, principalmente quando a entrada é muito pequena.

O algoritmo de Jarvis apresenta um desempenho intermediário entre os algoritmos de Graham com diferentes métodos de ordenação. Isso sugere que, dependendo do tamanho do conjunto de pontos e da distribuição dos pontos, o algoritmo de Jarvis pode ser uma opção viável em termos de desempenho.

Exemplo com 1000, 10000, 25000, 40000 e 50000 coordenadas:



Exemplo com 1000, 10000, 25000, 40000 e 50000 coordenadas, exceto MergeSort:



Com base nos resultados, pode se observar que o algoritmo de Graham com InsertionSort apresenta o melhor desempenho em termos de tempo de execução para todos os tamanhos de conjuntos de pontos testados. O algoritmo de Graham com MergeSort apresenta o pior desempenho em todos os cenários, o que indica que esse método de ordenação pode não ser a melhor escolha para esse problema específico.

6. Conclusões

O programa implementado permite calcular o fecho convexo de um conjunto de pontos no plano cartesiano utilizando os algoritmos de Graham Scan e Marcha de Jarvis. Durante o desenvolvimento deste trabalho, aprendi sobre os conceitos de fecho convexo.

Também ganhei ainda mais experiência na implementação de estruturas de dados e algoritmos em C++, algoritmos de ordenação e técnicas para análise de complexidade.

7. Bibliografia

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. The MIT Press.

2. Documentação da linguagem C++. Disponível em: <https://cplusplus.com/doc/>.

3. Ronan Lopes. Convex Hull: Algoritmos Paralelizados em C para Obtenção do Fecho Convexo no Plano.
Disponível em: <https://ronanlopes.me/convex-hull-algoritmo-paralelizado-em-c/>

4. Documentação gnuplot 5.5 -
Disponível em: <http://gnuplot.info/docs_5.5/gnuplot5.html>

5. Microsoft. Learn C++.
Disponível em: <https://learn.microsoft.com/en-us/cpp/?view=msvc-170>.

6. Paulo Feofiloff. DCC-IME-USP.
Disponível em: <https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html>.

7. GeeksforGeeks. Analysis of Algorithms | Big-O analysis.
Disponível em: <https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/>.

8. Claudio Esperança e Paulo Roma Cavalcanti. Geometria Computacional- Fecho Convexo II.
Disponível em: <http://orion.lcg.ufrj.br/gc/download/Fecho%20Convexo%20II.pdf>

9. Material apresentado em sala e disponível no Portal Minhas Turmas.

10. Convex Hull. Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Convex_hull>

8. Instruções para compilação e execução

8.1. Compilação e execução

A compilação e a execução do programa são feitas diretamente no terminal:

- Compilar e executar: `make fecho <arquivo de entrada>`
- Excluir objetos e arquivos executáveis: `make clean`

8.2. Animações

Para gerar as animações, é necessário ter a biblioteca *gnuplot* instalada. O arquivo *output.dat* é gerado antes do final da execução do programa.

Os arquivos .plt contem os comandos para criar gráficos baseados nos arquivos com 10, 100 e 1000 coordenadas, e podem ser ajustados para outros tamanhos ao alterar os valores máximos de `set xrange` e set yrange.

O arquivo .gif é gerado a partir do comando `gnuplot utils/<arquivo com comandos>`
Exemplo: `gnuplot utils/plot100.plt`