# Análise do Problema de Fecho Convexo por Meio de Algoritmos de Graham e de Jarvis

**Bruno Oliveira Souza Santos**

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

brunooss@ufmg.br

## 1 Introdução

Tem-se por fecho convexo de um conjunto de pontos o menor conjunto convexo que contém estes pontos, isto é, o menor polígono convexo que engloba todos estes em um plano bidimensional, sendo o conceito em questão um problema clássico de geometria computacional.

A fim de se obter o fecho convexo a partir de um conjunto de pontos, é possível implementar uma variedade de métodos conhecidos, em específico os algoritmos de Jarvis e de Graham, os quais serão abordados mais detalhadamente a seguir.

### 1.1 Algoritmo de Jarvis

O algoritmo de Jarvis, também conhecido como Envolvente Convexo de Jarvis, é um dos métodos de relevância para o trabalho em questão, que se baseia em se determinar o ponto mais à esquerda do conjunto (isto é, o ponto com menor valor em sua coordenada x) e, a partir deste, percorrem-se e agrupam-se os demais pontos no sentido anti-horário, sendo o próximo ponto o que possui orientação anti-horária em relação a qualquer outro. Ou seja, a cada iteração, o algoritmo fixa um ponto e percorre os demais, a fim de determinar se este é anti-horário em relação aos outros. Caso afirmativo, este integra o fecho convexo.

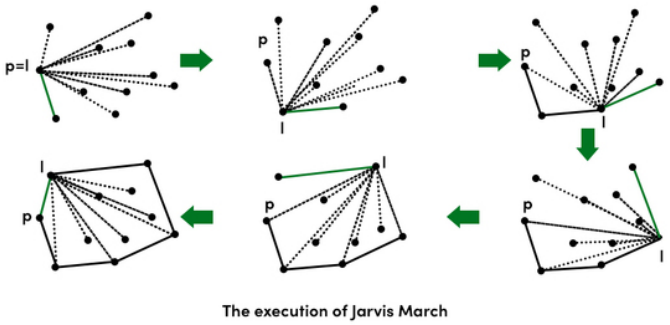


Figura 1: Demonstração de execução do Algoritmo de Jarvis.

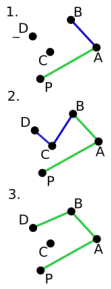


Figura 2: Demonstração de execução do Algoritmo de Graham.

### 1.2 Algoritmo de Graham

O algoritmo de Graham é uma outra abordagem para se solucionar o problema proposto, que se inicia a partir da determinação do ponto com menor valor em sua coordenada y e, caso haja mais de um ponto com este mesmo valor de y, é selecionado o ponto com menor valor em sua coordenada x. Em seguida, ordena-se o conjunto, tomando-se como referência o ângulo que cada ponto faz em relação ao selecionado anteriormente. Em seguida, o algoritmo percorre todos os pontos do conjunto e, considerando este os seus dois antecessores, é feita a seguinte comparação: caminhando-se do primeiro ao segundo e, em seguida, ao terceiro ponto, caso tenha se formado uma "curva à esquerda", o ponto em questão não faz parte do fecho convexo, e o último ponto passa a ser o próximo ponto da lista ordenada. Tal processo se repete até que o conjunto dos três últimos pontos represente uma curva à direita.

## 2 Metodologia

A implementação do problema de fecho convexo se deu a partir da linguagem de programação C++, e se iniciou por meio do desenvolvimento de classes que representam os Tipos Abstratos de Dados (TAD) de Ponto, de Reta e de Fecho Convexo, que serão devidamente especificados a seguir.

### 2.1 Definição de TADs

Inicialmente, fez-se necessária a declaração da classe "Ponto", a fim de sendo este o elemento mais elementar da metodologia do problema. Este TAD possui coordenadas x e y como valores inteiros, de forma que cada elemento ponto é declarado a partir desses dois valores.

Posteriormente, desenvolveu-se a classe "FechoConvexo", responsável pela resolução do problema em questão. Este Tipo Abstrato de Dados é inicializado a partir de uma lista de pontos, a partir da qual o algoritmo encontrará o fecho convexo do conjunto, com funções para resolução por meio do algoritmo de Jarvis ou do algoritmo de Graham - podendo este ser realizado a partir de diferentes métodos de ordenação, de acordo com a opção selecionada.

Ademais, desenvolveu-se o TAD "Reta", que consiste unicamente em dois pontos, um de origem e outro de destino, a fim de se representar geometricamente as arestas do fecho convexo.

Além disso, foram desenvolvidos TADs específicos para a manipulação dos pontos durante a execução dos algoritmos do fecho convexo, sendo estes uma lista encadeada e uma pilha encadeada, em que cada nó possui um valor que é instância do tipo "Ponto"e um apontador para outro nó.

### 2.2 Execução do Programa

Na função principal do programa, tem-se por objetivo ler os valores das coordenadas x e y do conjunto de pontos de entrada, a partir de um arquivo de texto passado como argumento para a execução. Em seguida, é criado uma instância da classe que representa o fecho convexo, sobre a qual são executadas quatro funções, que indicam os métodos de se encontrar o fecho convexo, de modo que o programa as executa, em ordem, de acordo com a configuração a seguir:

- GRAHAM+MERGESORT: tempo de execução do algoritmo de Graham com ordenação por Merge Sort
- GRAHAM+INSERTIONSORT: tempo de execução do algoritmo de Graham com ordenação por Insertion Sort
- GRAHAM+LINEAR: tempo de execução do algoritmo de Graham com ordenação por um algoritmo de complexidade linear. Neste caso, Radix Sort
- JARVIS: tempo de execução do algoritmo de Jarvis

Por fim, o programa terá como saída o conjunto de pontos que formam o fecho convexo, um por linha a partir de suas coordenadas x e y, e o tempo de execução de cada uma das configurações mencionadas acima.

Tal execução ocorrerá três vezes, para conjuntos de pontos com 100, 1000, 5000 e 10000 pontos, de modo a se obter o tempo de execução para cada uma das 4 configurações nesses três casos. Em seguida, objetiva-se analisar o crescimento do tempo de execução para cada configuração.

## 3 Análise de Complexidade

### 3.1 Complexidade de Tempo

O algoritmo de Jarvis possui complexidade de tempo de execução da ordem de $O(nh)$, sendo n o número de pontos e h o número destes que se encontram no fecho convexo do conjunto inicial. Tal ordem de complexidade se justifica pelo fato de o algoritmo percorrer todos os $n$ pontos repetidamente, inicialmente a fim de encontrar o ponto extremo inicial e, nas demais, a fim de encontrar cada próximo ponto no fecho convexo, totalizando $h$ repetições desse processo.

O algoritmo de Graham, por sua vez, possui complexidade de tempo da ordem de $O(nlogn)$, sendo $n$ a quantidade de pontos do conjunto inicial. Isso se dá pelo fato de o algoritmo classificar, a partir de um ponto de referência, os outros pontos em relação ao ângulo polar destes em relação ao primeiro, podendo esta etapa ser realizada a partir de métodos de ordenação com complexidade $O(nlogn)$, a exemplo do Quicksort e do Mergesort.

### 3.2 Complexidade de Espaço

Em relação à complexidade de espaço, o algoritmo de Jarvis possui complexidade $O(n + h)$, uma vez que este armazena em uma lista $n$ pontos iniciais e, posteriormente $h$ pontos pertencentes ao fecho convexo e, como $h$ sempre será menor ou igual a $n$, temos que a complexidade corresponde a $O(n)$.

Em relação ao algoritmo de Graham, este possui complexidade $O(n)$, em que $n$ é o número de pontos de entrada, uma vez que este armazena em uma lista os pontos de entrada e, além disso, armazena também uma pilha dos pontos já visitados, utilizada para determinar o fecho convexo. Como esta pilha adicional possui uma complexidade de espaço proporcional ao número de pontos

no fecho convexo, sendo este limitado pelo número de pontos de entrada, temos que a complexidade de espaço do algoritmo de Graham é $O(n)$.

# 4 Estratégias de Robustez

A implementação do problema do Fecho Convexo apresenta técnicas que contribuem para a sua robustez, com o objetivo de lidar com situações de erro e entradas inesperadas ou incorretas. Tais estratégias são melhor descritas a seguir:

## 4.1 Verificação de Argumentos em Função

No início da execução do programa, este realiza uma verificação da quantidade e do valor dos argumentos fornecidos na linha de comando, sendo estes o comando e o nome do arquivo que contém os pontos a serem considerados durante a execução do algoritmo.

A entrada esperada a ser inserida na linha de comando para a execução do programa é dada a seguir:

```
./bin/main fecho <arquivo.txt>
```

Nesse caso, temos que a entrada esperada é dada pelo arquivo executável do programa, dado por './bin/main' e, em seguida, o comando 'fecho' - que denota a funcionalidade da aplicação - e o nome do arquivo de texto.

Nesse sentido, é realizada a verificação do comando inserido, com o intuito de conferir que o comando fornecido como argumento seja correspondente a 'fecho'. Caso contrário, é lançada uma exceção de argumento inválido.

Além disso, o programa verifica se o caminho do arquivo fornecido não está vazio e se é também um arquivo de texto com extensão '.txt', de movo que uma exceção de argumento inválido é lançada caso tais condições não sejam satisfeitas.

Dessa forma, o programa é robusto no sentido de evitar erros durante a leitura dos dados advindos da linha de comando, relacionados à entrada do usuário.

## 4.2 Validação de Abertura do Arquivo

Após a verificação da entrada do usuário a partir da linha de comando, é conferido se a abertura do arquivo especificado se dá de maneira bem-sucedida e, caso contrário, uma exceção que descreve o problema é lançada ao usuário informando sobre a falha na abertura do arquivo. Essa estratégia confere ao programa a identificação prévia de um problema durante o acesso ao arquivo, o que previne a continuidade da sua execução.

## 4.3 Validação de Coordenadas dos Pontos

Durante a leitura de cada par de coordenadas X e Y do arquivo, o programa emprega expressões regulares para verificar se os valores que representam essas coordenadas são, de fato, valores inteiros. Caso alguma das coordenadas não atenda à condição acima, isto é, caso estes valores sejam valores decimais ou entradas diferentes de números, é lançada uma exceção que informa o usuário do erro, especificando o valor incorreto. Em seguida, é informado que o ponto em questão é ignorado, dando continuidade com a leitura, sem considerar o ponto incorreto. Essa estratégia contribui para

garantir a integridade dos dados lidos, evitando a inserção de valores inválidos no processamento subsequente, mas sem interromper o programa.

### 4.4 Validação de Opção de Ordenação para Algoritmo de Graham

Como é possível especificar a opção de ordenação a ser utilizada pelo algoritmo de Graham, sendo as opções $MergeSort$, $InsertionSort$ ou $RadixSort$, faz-se necessário verificar se a opção selecionada corresponde a uma das opções possíveis, de maneira que, se for inserida uma opção não prevista para ordenação, seja lançada uma exceção informativa para o usuário.

### 4.5 Tratamento de exceções

Por último, as exceções possivelmente lançadas durante a execução do programa é tratada no código, a fim de informar para o usuário a causa da interrupção. Diferentes tipos de exceção são tratados separadamente, de modo que o programa retorne mensagens de erro personalizadas para cada potencial cenário de erro.

## 5 Resultados e Análises

A partir do desenvolvimento das etapas descritas na metodologia, foram realizados testes a respeito do tempo de execução do programa para diferentes valores de entrada - isto é, para diferentes quantidades de pontos -, com o intuito de se analisar o comportamento dos algoritmos descritos conforme a quantidade de pontos aumenta.

Os testes foram conduzidos utilizando conjuntos de pontos contendo 100, 1000, 5000 e 10000 elementos, a partir dos quais foram registrados os tempos de execução para os quatro casos do algoritmo, sendo estes:

- GRAHAM+MERGESORT
- GRAHAM+INSERTIONSORT
- GRAHAM+LINEAR
- JARVIS

Para cada tamanho de conjunto, o programa foi executado múltiplas vezes e os tempos de execução foram registrados. Os tempos foram medidos em milissegundos (ms) e representam o tempo total necessário para calcular o fecho convexo dos pontos. A seguir, seguem os resultados para os casos de teste especificados:

- 100 elementos:

```
GRAHAM+MERGESORT:     0.001s
GRAHAM+INSERTIONSORT: 0.000s
GRAHAM+LINEAR:        0.000s
JARVIS:               0.000s
```

- 1000 elementos:

```
GRAHAM+MERGESORT:     0.047s
GRAHAM+INSERTIONSORT: 0.019s
GRAHAM+LINEAR:        0.019s
JARVIS:               0.057s
```

- 5000 elementos:

```
GRAHAM+MERGESORT:     1.303s
GRAHAM+INSERTIONSORT: 0.487s
GRAHAM+LINEAR:        0.584s
JARVIS:               2.307s
```

- 10000 elementos:

```
GRAHAM+MERGESORT:     5.677s
GRAHAM+INSERTIONSORT: 1.833s
GRAHAM+LINEAR:        1.853s
JARVIS:               9.841s
```

Com base nos resultados obtidos, podem ser observadas tendências que serão discutidas a seguir:

- O algoritmo de Jarvis apresentou desempenho comparável às outras configurações para conjuntos relativamente pequenos de pontos (até o teste de 1000 elementos), o que indica que, para conjuntos menores, o algoritmo de Jarvis é uma escolha interessante em termos de tempo de execução.
- O algoritmo de Graham se demonstrou ser mais eficiente à medida que o conjunto de pontos aumenta, sendo este mais indicado para conjuntos maiores de pontos.
- 3. Apesar da complexidade $O(nlogn)$ do algoritmo Merge Sort, é possível observar que esta configuração do algoritmo de Graham performou de maneira menos eficaz que os algoritmos de ordenação linear e de inserção. Tal fato é possivelmente causado pela sua natureza recursiva, que consome mais recursos da pilha de ativação e, consequentemente, tempo de execução do programa.

# 6 Conclusão

Neste trabalho, foi desenvolvido um programa para calcular o fecho convexo de conjuntos de pontos. Foram realizados experimentos para avaliar o desempenho do programa em diferentes tamanhos de conjuntos de pontos e com diferentes algoritmos de resolução.

Com base nos resultados obtidos, constatou-se que o algoritmo de Graham, com configurações de Insertion Sort ou algum método linear para a ordenação dos seus pontos é eficiente para conjuntos maiores de pontos, sendo que a configuração com Merge Sort se mostrou menos eficientes por conta de esta ser chamada recursivamente. Ainda assim, esta configuração se mostra eficiente para conjuntos maiores de pontos.

# Referências Bibliográficas

StdExcept. C++ Documentation, 14/06/2023. Disponível em: https://cplusplus.com/reference/stdexcept/

Casco Convexzo - Algoritmo de Jarvis. Acervo Lima, 14/06/2023. Disponível em: https://acervolima.com/casc convexo-conjunto-1-algoritmo-de-jarvis-ou-wrapping/

Graham Scan - Convex Hull. Opengenus, 14/06/2023. Disponível em: https://iq.opengenus.org/graham-scan-convex-hull/

## Instruções para Compilação

A fim de se compilar corretamente o programa desenvolvido, recomenda-se que sejam seguidos os seguintes passos:

- Após ter descompactado a pasta do projeto, deve-se certificar que o g++, compilador necessário para a compilação ideal do projeto, está instalado. Para isso, deve-se digitar, em um terminal com a pasta em questão aberta: 'g++ –version'
- Caso o compilador g++ não esteja instalado, deve-se digitar no terminal: 'sudo apt-get install g++', ou um comando similar, a depender da distribuição Linux a ser utilizada. Neste caso, o comando está adaptado ao ambiente de execução citado anteriormente.
- Após verificar a instalação do compilador do programa, convém digitar 'ls', a fim de verificar se o usuário se encontra na pasta correta. Caso afirmativo, este deve ver pastas como 'bin', 'include', 'obj' e 'src' como saída deste comando. Caso contrário, faz-se necessário navegar até a pasta descompactada previamente.
- A fim de se observar os resultados próprios, deve-se editar o arquivo 'pontos.txt' localizado na mesma pasta, com valores próprios coerentes com a documentação.
- Após a devida edição do arquivo de entrada, é possível executar o comando 'make run', a fim de se compilar todas as dependências do projeto.
- Por fim, a fim de verificar a saída do programa, deve-se digitar no terminal make clean && make all'. Este comando irá rodar o arquivo executável gerado pela compilação. Em seguida, a saída estará disponível no terminal.