Trabalho Prático 2
Fecho Convexo

Bernardo Dutra Lemos - 2022043949

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil
bernardolemos@dcc.ufmg.br

- ***Introdução:***

A ideia principal do trabalho é encontrar o fecho convexo a partir de um conjunto de pontos usando dois algoritmos distintos: A varredura de Graham e o Marchar de Jarvis, também conhecido como embrulho para presentes. Os dois métodos são comumente usados, mas apresentam algumas diferenças cruciais, o algoritmo de Graham utiliza a ordenação para resolver o problema, enquanto o de Jarvis não é necessário ordenar.

A maior parte do trabalho é implementar o método de Graham com três tipos de ordenação diferentes: *merge sort*, *insertion sort* e algum linear, no caso o escolhido foi ao *bucket sort*. Outro tópico do trabalho consiste em comparar o tempo que cada um dos métodos leva para encontrar o fecho convexo, no caso do Graham o tempo gasto para cada um dos três tipos de ordenação e no caso do Jarvis apenas o tempo necessário.

- ***Método:***

A implementação da solução foi feita usando uma classe base chamada **fecho convexo**, que possui dois atributos privados, um vetor de pontos e seu tamanho, além disso, possui diversos métodos que são usados nas duas implementações, a de Graham e a de Jarvis, esses serão descritos brevemente em breve.

Os dois algoritmos de resolução citados anteriormente foram criados como classes filhas da fecho convexo, nomeadas de Graham e Jarvis, cada uma dessas com seus próprios métodos exclusivos e implementação de métodos virtuais da classe herdada. O uso de herança visa evitar a repetição de códigos que seriam usados pelos dois algoritmos e tornar o código mais limpo e mais fácil de ler.

Os pontos foram armazenados através de um struct que possui uma coordenada x,y e um terceiro atributo chamado coeficiente que foi usado para auxiliar nas ordenações. E uma classe de pilha foi utilizada para auxiliar nos dois algoritmos, essa classe usa polimorfismo paramétrico tornando a possível a criação de pilhas de pontos.

A seguir uma breve descrição dos métodos da classe base:

- ➔ **Preenche:** Recebe uma string como parâmetro, que é o nome do arquivo a ser lido, e armazena os pontos contidos nesse arquivo no atributo privado da classe, além de salvar o tamanho do vetor
- ➔ **Troca:** Recebe dois pontos e realiza a troca eles de posição.
- ➔ **dist_eucl:** Recebe dois pontos e calcula a distância euclidiana entre eles.
- ➔ **Orientação:** Recebe três pontos e verifica se o caminho gerado por eles é uma curva à direita, à esquerda ou se são colineares.
- ➔ **Coeficientes:** Recebe um ponto p e calcula o ângulo da reta gerada por esse ponto e cada um dos pontos armazenados no vetor privado da classe.
- ➔ **RemoveColineares:** Considerando que o vetor armazenado na classe está ordenado por coeficiente e distância euclidiana, remove os pontos colineares deixando apenas o com maior distância euclidiana.
- ➔ **Tempo:** Recebe uma string que define qual dos métodos a ser medido e devolve o tempo de execução.
- ➔ **ImprimePontos:** Imprime todos os pontos armazenados no vetor, quando usado após o método resolve, imprime os pontos do fecho convexo.
- ➔ **ImprimeRetas:** Imprime as retas formadas pelos pontos do fecho convexo na forma ax + b = y (devido a uma mudança na saída esperada, não é usada, mas é funcional).
- ➔ **Resolve:** Método virtual que recebe uma string com qual dos métodos de ordenação será usado, caso seja a classe de jarvis, esse parâmetro é indiferente.

A seguir uma breve descrição dos métodos da classe Graham:

Os três métodos a seguir ordenam o vetor privado da posição 1 a n-1 de acordo com o coeficiente (atributo da struct ponto), e usando a maior distância euclidiana como desempate. Esse coeficiente é calculado no construtor do programa assim que os pontos são salvos e o elemento com menor y e menor x é colocado na primeira posição.

- ➔ **Merge**: Ordena o vetor privado usando *merge sort* e remove os colineares em sequência.
- ➔ **Insertion**: Ordena o vetor privado usando *insertion sort* e remove os colineares em sequência..

- ➔ **Linear**: Ordena o vetor privado usando *bucket sort* e remove os colineares em sequência.
- ➔ **Resolve:** Recebe uma string com qual dos métodos de ordenação será usado e resolve. Caso a string não esteja no formato correto, o método usado por padrão é o linear. Atualiza o vetor privado para conter apenas os pontos que formam o fecho convexo.

O mais difícil de implementar desses três métodos de ordenação foi o bucket sort, e a forma como foi resolvida foi o uso de 360 baldes sendo cada um deles responsável por uma faixa de 0.5 graus, pois o maior coeficiente possível era o ângulo de 180 graus, pois o ponto de referência é o ponto de menor altura, logo não existem pontos abaixo dele. E cada balde, que é uma pilha de pontos, foi ordenado usando merge sort ou insertion sort dependendo do tamanho dele.

A seguir uma breve descrição dos métodos da classe Jarvis:

- ➔ **EstaNoSegmento:** Verifica se um ponto está no segmento entre outros 2 pontos, usado de forma a retirar pontos colineares no método de Jarvis.
- ➔ **Resolve:** Usa o algoritmo de jarvis para resolver o fecho convexo e armazenar os pontos no vetor privado da classe.

● Análise de Complexidade:

Começando pela resolução do fecho convexo usando a varredura de Graham, temos que cada ponto é acessado no máximo duas vezes, para adicionar ou remover do fecho, logo esse processo tem uma complexidade de O(n), como é necessário realizar a ordenação do vetor a complexidade desse passo dependerá do método de ordenação usado. É importante lembrar que para gerar o coeficiente de ordenação é necessário percorrer o vetor uma vez. Temos então O(n) + O(2n) + custo de ordenação.

Logo, temos 3 casos de custo para a varredura de graham, um para cada método de inserção:

1. Insertion sort: $O(n^2)$
2. Merge sort: O(nlogn)
3. Bucket sort: O(n)*

   * No caso do bucket sort, é O(n) no melhor caso, caso cada um dos baldes tiver exatamente um ponto, na maioria das vezes é necessário realizar uma pequena ordenação nos baldes o que faz o custo variar, mas é na faixa do linear.

Analisando a resolução do fecho pelo algoritmo de Jarvis temos que o custo é O(nh), sendo n o número de pontos dados e h o número de pontos no fecho convexo. Podemos ter um pior caso no Jarvis, que é quando todos os pontos dados estão no fecho convexo, nesse caso o algoritmo se torna $O(n^2)$.

Quanto a complexidade de memória, todos as 4 formas de resolver o fecho convexo precisam armazenar no mínimo n pontos, mas nos métodos que utilizam o merge sort e o bucket sort, devido a implementação de ambos não ser in-place, eles consomem um espaço auxiliar de 2n. Logo a complexidade para todos os casos é na ordem de O(n).

- **Estratégias de Robustez:**

Durante o desenvolvimento do trabalho foram encontrados alguns problemas com divisão por zero, mas esses foram resolvidos e tratados, então não há problemas matemáticos no projeto. Entretanto, foram tratados alguns problemas que podem ser gerados por entradas, como é o caso que o arquivo passado não existe, ou nenhum arquivo foi passado como parâmetro ou até mesmo quando menos de 3 pontos não colineares são passados como parâmetro. Em todos esses casos uma mensagem de erro é imprimida e o programa encerrado. Linhas com apenas apenas uma das coordenadas são consideradas inválidas, dessa formas elas são ignoradas.

- **Análise Experimental:**

Os testes realizados foram feitos com o objetivo de analisar o comportamento do programa para diversos números de pontos na entrada. A forma como essa análise foi feita foi utilizando gráficos para comparar os tempos entre cada um dos 4 métodos usados. Foi feito um gráfico para o merge sort, bucket sort e para o algoritmo de jarvis, já o insertion sort foi utilizado outro gráfico, porque a diferença de tempos entre ele os os demais era muito grande e mudava a escala.

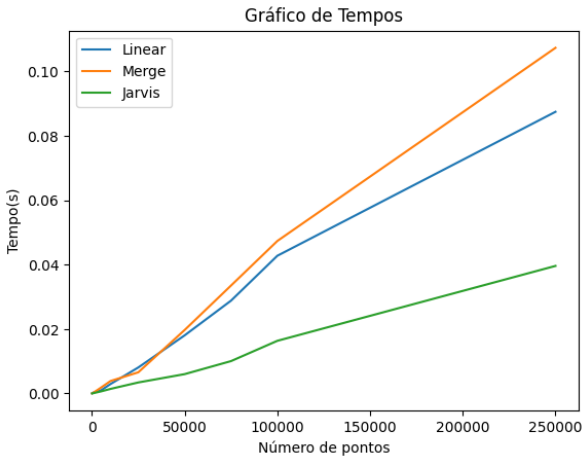

Figura 1. Merge x Jarvis x Linear

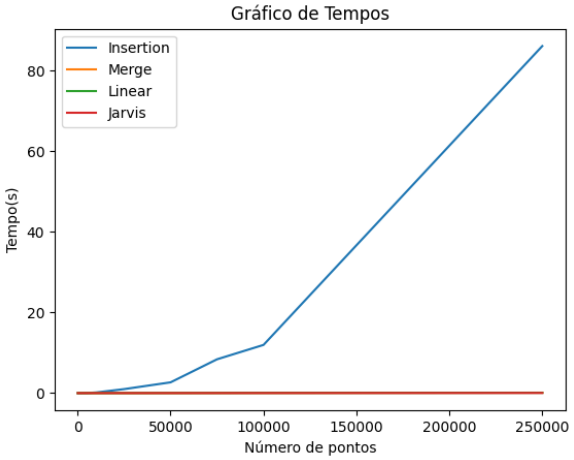

Figura 2. Insertion x Outros

Como pode ser observado nos gráficos o tempo, medido em segundos, cresce de forma muito maior quando o insertion sort é utilizado, o que faz sentido considerando o fato que ele é um método $O(n^2)$, já para os outros três é um pouco mais equilibrado, entretanto o algoritmo de jarvis é mais eficiente para os casos testados. Enquanto para os dois métodos de graham restante, o merge e o linear é bem equilibrado, mas para casos onde o n é grande, o linear tende a ser melhor.

● Conclusões:

O problema do fecho convexo foi resolvido usando dois algoritmos, a varredura de Graham e a marcha de Jarvis, no primeiro foram usados diferentes métodos de ordenação e além disso todos os casos foram testados em questão de performance.

Foi um trabalho bastante desafiador e muito interessante de se realizar, além de utilizar os conhecimentos aprendidos em sala de aula sobre os métodos de ordenação, onde foi possível ver na prática como cada um dos tipos se comporta e ver que os métodos quadráticos, o insertion sort no caso, são extremamente lentos quando comparados com os outros dois usados, o merge e o bucket. Também foi super interessante fazer essas ordenações mudando o parâmetro de comparação.

Como citado previamente, o mais desafiador foi a ordenação linear, pois houveram diversos problemas de vazamentos de memória e falhas de segmentação e foram necessárias várias tentativas para fazer o bucket sort funcionar da forma que devia e sem causar problemas de memória.

Uma observação válida é que a ordem dos pontos encontrada pelo método de jarvis é diferente da encontrada por graham, mas isso é causado pelo fato que os pontos de início de cada um deles é diferente, mas eles seguem uma mesma sequência.

● Bibliografia:

- Disponível em: <https://web.tecgraf.puc-rio.br/~celes/docs/inf2604/fechoconvexo.pdf>. Acesso em 01 de Junho de 2023

- Disponível em: <https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/>. Acesso em 01 de Junho de 2023

- CORMEM, Thomas. Algoritmos - Teoria e Prática. 3.ed. GEN LTC, 2012

- **Compilação:**

1. Abrir o terminal
2. Entrar na pasta tp
Comando: cd / <caminho para a pasta /TP2 >
3. Compilar o programa com o Makefile disponibilizado na pasta
Comando: make
4. Executar o programa
Comando: ./bin/main <nome do arquivo de entrada .txt>
*Caso queira que as retas que formam o fecho convexo sejam printadas, executar o comando da seguinte forma
Comando: ./bin/main <nome do arquivo de entrada .txt> -r
5. Após terminar de utilizar o programa, delete os arquivos desnecessários
Comando: make clean