Trabalho Prático 2
Fecho Convexo

Giovanna Naves Ribeiro
Matrícula: 2022043647
Data: 04/04/2023

1. Introdução

O trabalho a seguir tem como objetivo comparar o tempo de execução de 4 algoritmos que encontram o **fecho convexo** em um conjunto de pontos (menor polígono convexo que encapsule todos os pontos apresentados). Os algoritmos são:

- Graham Scan com MergeSort
- Graham Scan com InsertionSort
- Graham Scan com BucketSort (método de inserção linear escolhido)
- Jarvis March

2. Método

- Linguagem utilizada: c++
- Especificações do computador utilizado:
  - Sistema Operacional: Ubuntu 22.04.1 LTS
  - Processador: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
  - RAM instalada: 8,00 GB (utilizável: 7,79 GB)

2.1. Estruturas de Dados

Foi escolhido o tipo Struct para armazenar os dados do Point (coordenadas cartesianas x e y de cada ponto) e uma classe para o Fecho Convexo, que será especificada no tópico 2.2.

2.2. Classes, Funções e Algoritmos

O programa utiliza classes para a implementação das estruturas de dados, seguindo os princípios da programação orientada a objetos.

A classe **ConvexHull** (fecho convexo) no arquivo **convexhull.hpp** é utilizada para implementar os fechos pela utilização das seguintes funções e algoritmos:

1) A função **jarvisMarch** implementa o algoritmo do Marchar de Jarvis para encontrar o fecho convexo.
2) A função **grahamScan** implementa o algoritmo do Scan de Graham para encontrar o fecho, e, dentro dela, utilizamos três algoritmos de ordenação (um em cada chamada, de acordo com qual número passado como o parâmetro sortType na chamada da função grahamScan).
   Esses algoritmos são:
   a) O MergeSort, cuja função também se encontra na classe ConvexHull com o nome de **mergeSort**, e que possui uma função auxiliar chamada **merge**, também na classe;

b) O InsertionSort, cuja função se encontra na classe com o nome **insertionSort**;
c) O BucketSort, algoritmo de ordenação linear escolhido, cuja função leva o nome de **bucketSort** e possui uma função auxiliar chamada **digitCountOfMaxX**, também na classe. Além disso, dentro da função bucketSort é chamada a insertionSort para ordenar os baldes.

Além disso, possuímos também na classe ConvexHull, a função **printConvexHull**, que imprime os pontos do fecho convexo no terminal de acordo com o resultado encontrado pela função jarvisMarch.

Fora da classe, possuímos as funções **readPointsFromFile** e **crossProduct**, no arquivo **point.hpp**, que leem os pontos do arquivo de entrada **pontos.txt** e calculam o produto vetorial dos pontos dentro dos algoritmos de fecho convexo respectivamente.

Além disso, temos a função **main** no arquivo **main.cpp**, que executa os algoritmos de fecho convexo com os diferentes métodos de ordenação, mede o tempo de execução de cada um e cria um arquivo .csv onde armazenamos os tempos de execução dos algoritmos para entradas de diferentes tamanhos a método de comparação.

3. Análise de Complexidade

Nossas principais funções têm as seguintes ordens de complexidade (considerando n o número de pontos da entrada):

- MergeSort: **O(n log n)**;
- InsertionSort: **O(n^2)**;
- BucketSort: **O(n+k)**, onde k é o número de baldes escolhido;
- Jarvis March: **O(nh)**, onde h é o número de pontos do fecho convexo;
- Graham Scan: **O(n log n)**.

Logo, o Jarvis March possui complexidade O(nh) e o Graham Scan assume a complexidade do algoritmo de ordenação escolhido para ele em cada execução. Como nosso programa roda todos os algoritmos, ele assume a complexidade do mais custoso - o Graham Scan com InsertionSort (O(n^2)).

4. Estratégias de Robustez

O primeiro cuidado que tomamos no código em relação a possíveis falhas é conferir se foi possível encontrar e abrir o arquivo de entrada. Caso haja algum erro, o informamos ao usuário.

```
// Função para ler os pontos a partir de um arquivo
int readPointsFromFile(const char *filename, Point **points)
{
    ifstream arquivo;
    arquivo.open(filename);
    if (!arquivo.is_open())
    {
        cout << "Erro ao abrir o arquivo." << endl;
        return 0;
    }
```

Além disso, antes de começar os algoritmos de fecho convexo, certificamos que há no mínimo 3 pontos na entrada para que possa ser feito o cálculo de um fecho.

```
// Algoritmo de fecho convexo - Graham Scan
int ConvexHull::grahamScan(Point *points, int numPoints, Point **convexHull, int sortType)
{
    if (numPoints < 3)
    {
        cout << "Não há pontos suficientes para o cálculo de um fecho convexo." << endl;
        return 0;
    }
```

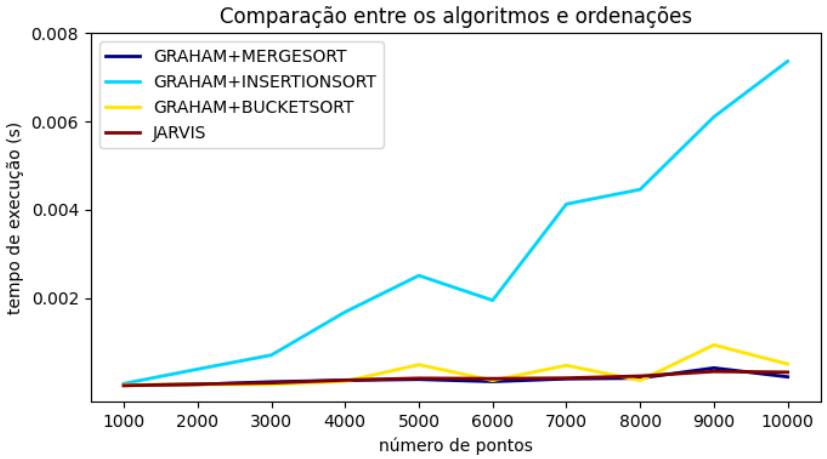
```
// Algoritmo de fecho convexo - Jarvis March
int ConvexHull::jarvisMarch(Point *points, int numPoints, Point **convexHull)
{
    if (numPoints < 3)
    {
        cout << "Não há pontos suficientes para o cálculo de um fecho convexo." << endl;
        return 0;
    }
```

5. Análise Experimental

Utilizamos a função **clock_gettime()**, da biblioteca <sys/time.h>, para medir o tempo de execução dos algoritmos com cada método de ordenação.

Embora tenhamos testado o programa para entradas enormes como 100.000 pontos, a partir de um certo tamanho de entrada, o algoritmo do Scan de Graham com ordenação InsertionSort resultava em tempos tão grandes que, visualizando o gráfico de linhas dos algoritmos, não era possível fazer uma boa comparação entre os outros métodos.

Por isso, montamos um gráfico com entradas menores (de 1.000 pontos a 10.000 pontos com um intervalo de 1.000 entre cada entrada, e com os pontos variando de 0 a 999). Nele, podemos perceber melhor a diferença no tempo de execução (em segundos) em cada caso e comparar a eficiência dos algoritmos.



6. Conclusões

Portanto, podemos concluir que o algoritmo de Graham Scan com a ordenação de MergeSort foi, em média, a mais eficiente, seguida pelo algoritmo de Jarvis March.

O Graham Scan com o BucketSort mostrou tempos de execução ligeiramente maiores e o Graham Scan com o InsertionSort se mostrou o algoritmo menos eficiente, pois aumentou muito em tempo de execução com o aumento do tamanho da entrada.

7. Bibliografia

"Convex Hulls: Graham Scan - inside Code." *YouTube*, 9 Jan. 2022, www.youtube.com/watch?v=SBdWdT_5isI.

"Convex Hulls: Jarvis March Algorithm (Gift-Wrapping) - inside Code." *YouTube*, 26 Dec. 2021, www.youtube.com/watch?v=nBvCZi34F_o.

23, Mar, and 7 Minutes Read. "Bucket Sort in C++ (Code with Example)." *FavTutor*, favtutor.com/blogs/bucket-sort-cpp. Accessed 11 June 2023.

8. Instruções para compilação e execução

Para compilar o programa, o primeiro a se fazer é abrir a pasta TP e copiar o arquivo de entrada (com os pontos) na raiz do projeto, pois é ele que é lido no Makefile do programa. Deve-se renomear esse arquivo para **pontos.txt**.

Enfim, deve-se executar o comando **make** no terminal para que o programa compile. Em seguida, **./bin/fecho pontos.txt** para que o programa execute.