Trabalho prático 2

Cálculo de Fecho Convexo

Iago da Silva Rodrigues Alves – 2022035881

iagosilva92@ufmg.br

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte - MG – Brasil

1. Introdução

O projeto em questão é um programa em C++ para a manipulação de dados bidimensionais e a determinação do fecho convexo de um conjunto de pontos. Os pontos são fornecidos em um arquivo texto e podem ter o fecho calculado através dos métodos Graham e Jarvis. Para o método de Graham o usuário pode escolher entre três tipos de ordenação (InsertionSort, MergeSort e BucketSort).

2. Método

2.1 Ferramentas

Esse projeto utiliza as seguintes ferramentas e especificações:

- Linguagem: C++11
- Compilação: g++ 11.3.0
- Sistema Operacional: *Ubuntu 22.04*
- GNU Make 4.3

2.2 Tipos Abstratos de Dados

Este projeto contém seis arquivos de cabeçalho para definição dos Tipos Abstratos de Dados, são eles:

Ponto.hpp: Fornece uma estrutura para manipular e realizar operações com pontos no plano cartesiano. Armazena as coordenas x e y, e possui métodos que auxiliam nos cálculos do Fecho Convexo.

Pilha.hpp: A classe Pilha é uma classe auxiliar que é utilizada para armazenar os pontos nas operações de Graham. Se trata de uma pilha encadeada usada em operações de Graham.

ListaEncadeada.hpp: Essa lista é implementada como uma estrutura de dados dinâmica que permite armazenar objetos da classe "Ponto". Em resumo, o TAD ListaEncadeada fornece uma estrutura de dados flexível que permite a adição, remoção e acesso aos elementos da lista encadeada.

FechoConvexo.hpp: Representa um fecho convexo de pontos em um plano. O fecho convexo é armazenado em uma lista encadeada. Assim é utilizado para representar e manipular fechos convexos de pontos no plano.

Graham.hpp: O TAD "Graham" implementa o algoritmo de Graham, que é utilizado para encontrar o fecho convexo de um conjunto de pontos no plano. Ele utiliza diferentes algoritmos de ordenação para realizar a ordenação dos pontos antes de calcular o fecho convexo. O resultado é armazenado em um objeto do tipo FechoConvexo.

Jarvis.hpp: Apresenta a definição de um TAD chamado "Jarvis" que implementa o algoritmo de Jarvis para calcular o fecho convexo de um conjunto de pontos no plano. Possui um método para calcular o fecho que posteriormente o armazena em um objeto tipo FechoConvexo.

2.3 Funções

O programa possui inúmeras funções, no entanto as principais são aquelas que trabalham diretamente na construção do Fecho Convexo. São elas: Os três algoritmos de ordenação implementados em Graham e os dois métodos de cálculo de fecho implementados em Graham e Jarvis.

Alguns outros métodos secundários são implementados no TAD Ponto e são utilizados no momento do cálculo do fecho, como as funções de comparação, de orientação e de cálculo de distância.

3. Análise de Complexidade

A complexidade de tempo do programa dependerá do tipo de execução escolhida pelo usuário. Dentre o cálculo do fecho pelo algoritmo de Graham existem três opções:

1. InsertionSort: Complexidade $O(n^2)$
2. MergeSort: Complexidade O(nlogn)
3. BucketSort: Complexidade O(n)

Já para o algoritmo de Jarvis, como ele não necessita de ordenação, sua complexidade é de O(nh) onde h representa o número de pontos do casco convexo resultante.

Ademais, falando sobre a complexidade de espaço do programa fica claro que é determinada principalmente pela alocação dinâmica de memória para armazenar os pontos. Essa complexidade é O(n), onde n é o tamanho do vetor de pontos lidos do arquivo de entrada. Além disso, há uma quantidade constante de espaço utilizado para outras variáveis, estruturas de dados e objetos no código, que contribuem para uma complexidade de espaço constante (O(1)). Portanto, a complexidade de espaço geral do programa é dominada pela alocação dinâmica de memória proporcional ao tamanho do conjunto de pontos lidos, resultando em uma complexidade de espaço final de O(n), onde n é o número de pontos.

4. Estratégias de Robustez

O programa implementa mecanismos de programação defensiva e tolerância a falhas a fim de garantir a robustez e confiabilidade do sistema, abaixo estão algumas das características do programa que contribuem para essa finalidade:

Tratamento de exceções: O programa implementa um tratamento adequado de exceções em várias partes do código. Especificamente, durante a leitura do arquivo de entrada, o programa verifica se o arquivo foi aberto com sucesso e trata a situação em que não foi possível abri-lo. Além disso, durante o cálculo do fecho convexo e a medição dos tempos, o programa captura exceções e as reporta. Esse tratamento de exceções evita que o programa seja encerrado abruptamente e permite ao usuário obter informações detalhadas sobre erros ocorridos.

Verificação de argumentos de linha de comando: O programa verifica se o número correto de argumentos foi fornecido na linha de comando. Caso contrário, exibe uma mensagem de erro indicando que o arquivo não foi fornecido adequadamente. Essa verificação ajuda a evitar erros decorrentes de entradas inválidas ou ausentes.

```
if (argc < 2) {
  std::cout << "Erro: Inclua o arquivo na linha de comando." << std::endl;
  return 1;
}
std::string arquivo = argv[1];

std::ifstream arquivoEntrada(arquivo);
if (!arquivoEntrada) {
  std::cout << "Erro: não foi possível abrir o arquivo." << std::endl;
  return 1;
}
```

Validação dos dados de entrada: Durante a leitura do arquivo de entrada, o programa realiza a conversão dos dados lidos em objetos do tipo Ponto. Fazendo uma verificação explícita dos valores lidos para garantir que eles sejam válidos. Se os

dados de entrada contiverem valores inválidos ou não numéricos, é possível que ocorra uma exceção ou resultados imprecisos. Portanto, é importante garantir que o arquivo de entrada contenha apenas valores válidos para obter resultados confiáveis.

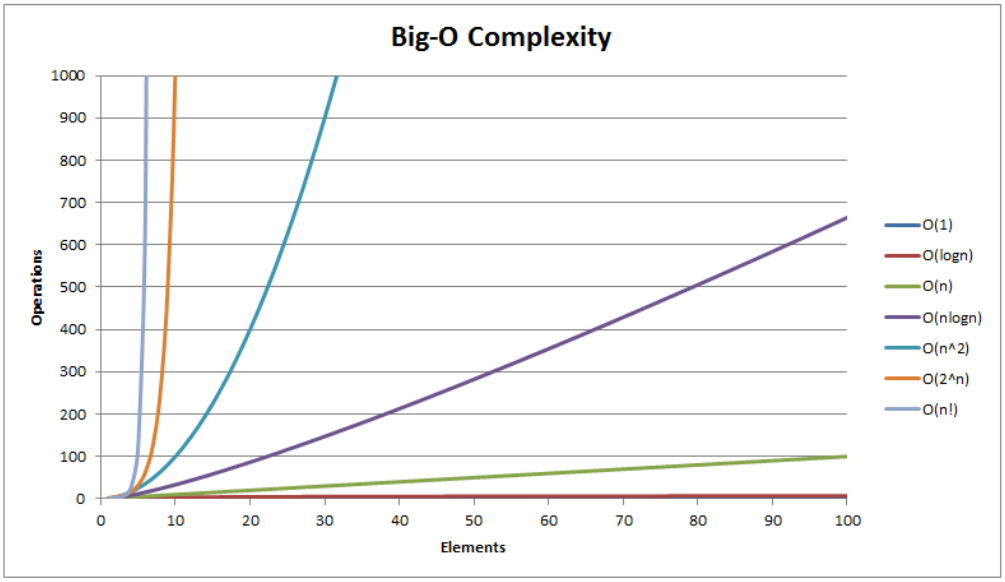
Liberação adequada da memória: O programa aloca memória dinamicamente para criar cópias dos pontos de entrada e liberar a memória alocada no final do uso. Isso evita vazamentos de memória e garante que os recursos sejam liberados corretamente, mesmo em caso de exceções.

Dessa forma, como o programa lida com um grande número de pontos para calcular o fecho convexo e medir os tempos, é crucial que ele seja robusto em relação ao desempenho e à estabilidade. O tratamento adequado de exceções e a liberação correta da memória ajudam a evitar problemas de consumo excessivo de recursos ou falhas de execução devido a problemas internos.

Em resumo, o programa implementa medidas de robustez e confiabilidade, como tratamento de exceções, validação de entrada, verificação de argumentos e liberação adequada de recursos. Essas medidas garantem que o programa possa lidar com uma variedade de situações adversas e forneça resultados confiáveis.

5. Análise Experimental

Ademais, além da análise de complexidade teórica, outra abordagem importante para avaliar o desempenho de um algoritmo é a análise experimental. No entanto, é importante ressaltar que a análise experimental também tem suas limitações, uma vez que os resultados podem ser afetados por fatores externos, como a arquitetura do computador, o sistema operacional e a carga de trabalho do sistema no momento dos testes. Por isso, é importante realizar vários testes para obter uma estimativa mais precisa do desempenho do algoritmo.



Após realizar testes no programa, foi observado que a complexidade de tempo dos algoritmos implementados segue os padrões esperados. Para o algoritmo de Graham, que utiliza diferentes métodos de ordenação, foram identificadas as seguintes curvas de entrada x tempo:

1. Ordenação por InsertionSort: A curva de tempo segue uma tendência quadrática, representada pela curva azul. Isso indica que a complexidade de tempo do algoritmo de Graham com InsertionSort como método de ordenação é $O(n^2)$, onde n é o número de pontos.
2. Ordenação por MergeSort: A curva de tempo apresenta um comportamento logarítmico, representada pela curva roxa. Isso indica que a complexidade de tempo do algoritmo de Graham com MergeSort como método de ordenação é O(nlogn), onde n é o número de pontos.
3. Ordenação por BucketSort: A curva de tempo segue uma tendência linear, representada pela curva verde. Isso indica que a complexidade de tempo do algoritmo de Graham com BucketSort como método de ordenação é O(n), onde n é o número de pontos.

Além disso, ao calcular o fecho convexo utilizando o algoritmo de Jarvis, que possui uma complexidade um pouco maior que O(n), foi observado que a curva de tempo é levemente mais inclinada que a curva verde (O(n)). Isso indica que a complexidade de tempo do algoritmo de Jarvis é um pouco maior que O(n), mas ainda segue uma tendência linear.

Em resumo, os resultados dos testes confirmam que as complexidades de tempo dos algoritmos implementados no programa estão de acordo com as análises teóricas esperadas. A escolha do método de ordenação utilizado afeta diretamente a eficiência do algoritmo de Graham, e o algoritmo de Jarvis apresenta uma complexidade de tempo ligeiramente superior, porém ainda mantendo uma eficiência próxima à complexidade linear.

6. Conclusões

Em conclusão, neste projeto de cálculo de fecho convexo, foi desenvolvida uma implementação em C++ que retorna o fecho através dos algoritmos de Graham ou Jarvis. A implementação apresenta recursos abrangentes, incluindo para Graham a ordenação tipo InsertionSort, MergeSort e BucketSort.

Durante o desenvolvimento, foram aplicados conceitos e técnicas aprendidas em sala de aula, como modularização, depuração, construção de tipos abstratos de dados e algoritmos de ordenação. Além disso, o programa foi projetado para ser robusto e tolerante a falhas, incorporando mecanismos de programação defensiva e estratégias de tratamento de exceções. Essas características proporcionam uma base sólida para futuras extensões e melhorias no projeto.

7. Bibliografia

Slides da disciplina de Estrutura de Dados. Disponibilizados via Moddle. 1° semestre 2023.

Convex hulls: Graham scan - Inside code
Disponível em: <Convex hulls: Graham scan - Inside code>
Acesso em junho 2023

Convex hulls: Jarvis march algorithm (gift-wrapping) - Inside code
Disponível em: <Convex hulls: Jarvis march algorithm (gift-wrapping) - Inside code>
Acesso em junho 2023

Imagem gráfica utilizada
Disponível em: <https://estevestoni.medium.com/iniciando-com-a-nota%C3%A7%C3%A3o-big-o-be996fa3b47b>
Acesso em junho 2023

8. Instruções para compilação e execução

Passo 1: Baixe o arquivo zipado referente ao trabalho e extraia seu conteúdo. Abra o diretório **TP2;**

Passo 2: Para compilar, digite o comando make no terminal.

Passo 3: Crie um arquivo texto (.txt) dentro do diretório. Nesse arquivo coloque as entradas para a execução do programa.

- **Atenção**: Cada linha do arquivo indica uma entrada para o programa, é necessário colocar apenas pontos na entrada no seguinte formato: x y

Passo 4: Para executar o programa existem três alternativas.

1. **make run ARGS="-o nomedoarquivo.txt"**
2. **bin/resolvedor -o nomedoarquivo.txt**
3. **Adicionar o PATH da pasta bin no seu bash e escrever na linha de comando: fecho arquivo_de_entrada.txt**

Limpeza: Para limpar os diretórios criados, contendo os objetos e o executável, digite **make clean** no terminal.