



UNIOESTE

Universidade Estadual  
do Oeste do Paraná

## Projeto 1 - Algoritmos de Busca

---

*Professora:*

Huei Diana Lee

*Alunos:*

Victor Hugo Almeida Alicino,  
Victor Emanuel Almeida

December 5, 2022

---

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Executando o programa</b>	<b>2</b>
2.1	Requisitos . . . . .	2
2.2	Compilando . . . . .	2
2.3	Executando . . . . .	3
2.4	Saída . . . . .	4
<b>3</b>	<b>Implementação</b>	<b>4</b>
3.1	Estruturas de dados . . . . .	4
3.2	Algoritmos de busca . . . . .	5
3.2.1	Algoritmo A* (melhor solução) . . . . .	5
3.2.2	Busca em largura (pior solução) . . . . .	5
3.2.3	Busca em profundidade (extra) . . . . .	5
<b>4</b>	<b>Testando o sistema</b>	<b>5</b>
4.1	Descrição do teste . . . . .	6
4.2	Resultados dos testes . . . . .	6
<b>5</b>	<b>Conclusão</b>	<b>6</b>

## Lista de Figuras

1	Exemplo de arquivo de saída do programa . . . . .	4
---	---	---

## Lista de Tabelas

# 1 Introdução

O Projeto tem como objetivo a implementação de um sistema para auxiliar turistas em Veneza a chegarem aos diversos museus da cidade, localizados em diferentes ilhas.

Implementado inteiramente em `c++20` sem utilização de bibliotecas externas além das bibliotecas padrão do `c++20`, podendo assim ser compilado em qualquer sistema operacional que possua o compilador adequado.

Essa linguagem foi escolhida pois além de possuir abstrações alto nível com o uso de classes, ainda é extremamente eficiente ao ser compilada diretamente para linguagem de máquina. Além do disso a dupla responsável pelo projeto possui familiaridade com a linguagem.

O sistema é capaz de calcular a rota mais curta entre dois pontos, utilizando os algoritmos  $A^*$ , busca em largura e busca em profundidade.

Visando a simplicidade e levar os algoritmos a seus limites de eficiência, o sistema não implementa uma interface gráfica, todos os argumentos devem ser passados como argumentos de linha de comando, ou coletados em tempo de execução, para ver os argumentos para o programa veja a seção 2.3.

Uma vez que todos os parâmetros para execução do programa podem ser passados antes que ele inicie, pode-se fazer uso de scripts para automatizar a execução do programa gerando resultados para diferentes cenários. Dessa maneira foi implementado um script para testar o desempenho, explicado na seção 4.

## 2 Executando o programa

### 2.1 Requisitos

- `cmake`;
- `make`;
- `g++`;
- `git` (opcional).

### 2.2 Compilando

Antes de realizar a compilação é necessário definir o nível de mensagens que serão exibidas durante a execução do programa.

Para isso basta definir a expressão `DEFAULT_LOG_LEVEL`, no arquivo `logger.hpp`, para um dos valores definidos no enum `LogLevel`:

Listing 1: Enum `LogLevel`

---

```
1  enum LogLevel { DEBUG, INFO, WARNING, ERROR, FATAL };
```

---

Caso o valor seja definido como:

- **DEBUG**: Todas as mensagens serão exibidas, tais como objeto construído, objeto destruído, tempos de execução de funções intermediárias, etc.
- **INFO**: Exibe mensagens “Modo iterativo” dos algoritmos, como a ilha que está sendo visitada, entre outras.
- **WARNING**: Exibe mensagens de aviso como quando um arquivo não é encontrado.
- **ERROR**: Exibe mensagens de erro na qual ainda é possível continuar a execução do programa.
- **FATAL**: Exibe apenas mensagens de erro que impossibilitam a execução do programa, nível que menos exibe mensagens.

Uma vez definido qual o nível de mensagens que serão exibidas, basta compilar o programa com os comandos:

- Entrar na pasta build: `cd build`
- Configurar o projeto: `cmake .`
- Compilar o projeto: `make`

## 2.3 Executando

Para executar o programa basta executar o arquivo

`Graph_Search_Algorithms_1.0.0` gerado na pasta build.

Caso nada seja passado como argumento, o programa irá coletar os argumentos em tempo de execução, sendo eles:

1. O caminho para o arquivo de entrada;
2. O caminho para o arquivo de saída contendo o resumo da execução;
3. O algoritmo a ser utilizado, podendo ser:

- $A \rightarrow A^*$ ;
- $B \rightarrow BFS$ , *Breadth First Search* (busca em largura);
- $D \rightarrow DFS$ , *Depth First Search* (busca em profundidade).

Exemplos de execução:

- **Sem argumentos:** `./Graph_Search_Algorithms_1.0.0`
- **Com argumentos:** `./Graph_Search_Algorithms_1.0.0 input_files/exemplo_prof.txt output.txt A`

## 2.4 Saída

Caso a execução seja bem sucedida, o programa irá gerar um arquivo de saída contendo o resumo da execução, que deverá ser semelhante ao exemplo abaixo:

```
Arquivo de entrada: input_files/exemplo_prof.txt
Data: 05/12/2022
Hora: 01:12:25.373386
Execução do algoritmo demorou: 0:0:0.000015
Caminho: b a
```

Figura 1: Exemplo de arquivo de saída do programa

## 3 Implementação

Essa seção visa explicar a implementação do programa, especificando seus algoritmos e estruturas de dados.

Para mais detalhes da implementação, tais como todas as classes e seus métodos, pode-se consultar a documentação em <https://graph-search-algorithms.pages.dev/>

### 3.1 Estruturas de dados

Segue abaixo a descrição das estruturas de dados utilizadas no programa.

Listing 2: Estrutura de dados para representar um nó do grafo.

```
1 enum NodeState { HAS_NONE = 0, HAS_WEIGHT, HAS_HEURISTIC,
    HAS_BOTH };
```

```
2
3  class AdjacencyNode {
4  private:
5      std::string _id;
6      int16_t _weight;
7      int16_t _heuristic;
8      NodeState _state;
9  };
```

---

Listing 3: Estrutura de dados para representar o grafo.

---

```
1  class Graph {
2  private:
3      std::unordered_map<std::string, std::vector<AdjacencyNode>>
        _nodes;
4      std::string _start_node;
5      std::string _end_node;
6      Algorithms _algorithm;
7  };
```

---

Para representação do grafo foi utilizado uma lista de adjacência, porém ao invés de utilizar um vetor, como normalmente é utilizado, optou-se por usar um *unordered\_map*, tabela hash, para armazenar os nós do grafo. Visto que no arquivo de entrada possui nomes de ilhas e não números para identificar os nós, dessa forma para utilizar um vetor seria necessário um mapeamento entre os nomes e os números, o que poderia dificultar a implementação.

A estrutura grafo também armazena o identificador do nó inicial e final, bem como o algoritmo a ser utilizado. Sendo assim o grafo é o objeto que contém todos os dados necessários para a execução do programa.

## 3.2 Algoritmos de busca

### 3.2.1 Algoritmo A\* (melhor solução)

### 3.2.2 Busca em largura (pior solução)

### 3.2.3 Busca em profundidade (extra)

## 4 Testando o sistema

Com o sistema implementado, foi possível realizar os seguintes testes:

## 4.1 Descrição do teste

- Para cada um dos algoritmos de busca implementados, foi executado  $X$  vezes passando os parâmetros  $arquivo\ saida_i\ algoritmos_j$ , sendo  $i \in \{1, \dots, 100\}$  e  $j \in \{A^*, DFS, BFS\}$ .
- Após cada execução espera-se um tempo de 1s para que o sistema possa ser executado novamente.
- Os testes foram realizados em um computador com as seguintes características:
  - **Processador:** i3-1115G4 4.100GHz;
  - **Memória RAM:** 8GB;
  - **SSD:** 256GB;
  - **Sistema operacional:** Arch Linux <sup>i</sup>.
- A execução dos testes foi realizada logo após a inicialização do computador, sem nenhum outro processo de usuário em execução.

## 4.2 Resultados dos testes

## 5 Conclusão

---

<sup>i</sup>Link para download do sistema operacional <<https://archlinux.org/download/>>