Trabalho Prático 1

Gabriela Tavares Barreto

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

gbarreto@ufmg.br

1 Introdução

Este documento apresenta a implementação e análise de cinco algoritmos de busca aplicados ao problema de encontrar o menor caminho em um ambiente bidimensional representado por uma matriz de terrenos com custos variados. Os algoritmos implementados incluem três estratégias de busca sem informação — Busca em Largura (BFS), Aprofundamento Iterativo (IDS) e Busca de Custo Uniforme (UCS) — e duas estratégias informadas — Busca Gulosa e A*.

O objetivo principal deste trabalho é explorar as diferenças entre esses algoritmos no contexto de busca em grafos, avaliando suas eficiências em termos de número de estados expandidos, tempo de execução e precisão das soluções. Para os métodos informados (Gulosa e A*), foi necessário definir funções heurísticas válidas e avaliar sua admissibilidade no problema.

O programa desenvolvido é capaz de processar entradas específicas fornecidas em arquivos texto que descrevem o mapa e seus terrenos, identificar o método de busca a ser utilizado e computar o menor caminho entre dois pontos no espaço do problema. Como saída, o programa exibe o custo total do caminho encontrado, bem como a sequência de pontos que compõem o percurso. Adicionalmente, análises quantitativas e qualitativas são realizadas para comparar os algoritmos implementados.

A documentação a seguir apresenta as decisões de implementação, os fundamentos teóricos dos algoritmos, as heurísticas utilizadas e uma análise comparativa dos métodos. Além disso, inclui um guia detalhado para execução do programa e instruções para compilação. Este trabalho busca não apenas implementar as técnicas de busca de forma eficiente, mas também compreender suas limitações e aplicações em diferentes contextos.

2 Implementação

Nessa seção será detalhada a implementação de todos métodos implementados, que recevem um grafo representado como um mapa não ordenado de nós. O objetivo principal das funções é encontrar o menor caminho entre um nó inicial (`start`) e um nó de destino (`end`), considerando o custo total acumulado das arestas.

2.1 Descrição da Entrada

Todas as funções recebem os seguintes parâmetros de entrada:

- `graph`:
  - Representa o grafo no qual será realizada a busca. O grafo é modelado como um `unordered_map`, onde:
    - A chave (`int`) é o identificador único de um nó.
    - O valor (`Node`) é um objeto que contém:
      - `neighbors`: Uma lista de inteiros representando os nós vizinhos conectados ao nó atual.
      - `weight`: O custo associado às arestas que conectam os nós.
- `start`:
  - O identificador do nó inicial da busca.
- `goal`:
  - **O** identificador do nó objetivo da busca.
- `expandedStates`:
  - **U**ma referência a um inteiro que será atualizado durante a execução da função para indicar o número total de estados expandidos.
- `executionTime`:
  - **U**ma referência a uma variável do tipo `double`, onde será armazenado o tempo total de execução da função em milissegundos.

2.2 BFS

Esta seção descreve as estruturas utilizadas e a modelagem dos componentes do algoritmo.

2.3 Estruturas Utilizadas

A função utiliza as seguintes estruturas de dados:

- **Grafo:** Representado como um `unordered_map<int, Node>`, onde cada nó é uma instância da classe `Node`. Um nó possui:
  - `neighbors`: Uma lista de inteiros que representa os nós vizinhos conectados.
  - `weight`: O peso associado às arestas conectadas ao nó.
- `parent`**:** Um `unordered_map<int, int>` que rastreia o nó pai de cada nó visitado. Essa estrutura é usada para reconstruir o caminho da origem ao destino ao final da busca.
- `visited`**:** Um `unordered_map<int, bool>` que marca os nós já visitados, evitando ciclos ou a reexploração de nós.
- `cost`**:** Um `unordered_map<int, float>` que rastreia o custo acumulado de cada nó desde o nó inicial.
- `q`**:** Uma fila (`queue<int>`) usada para explorar os nós em ordem de chegada, implementando a estratégia de busca em largura (FIFO).

2.4 Modelagem dos Componentes

A modelagem dos principais componentes do algoritmo é apresentada a seguir:

2.4.1 Estado

Cada nó do grafo é considerado um estado. O estado inicial é definido como `start`, e o estado objetivo é definido como `end`.

2.4.2 Função Sucessora

Para cada nó atual, a função sucessora itera sobre os vizinhos listados em `neighbors` do grafo. Cada vizinho ainda não visitado é adicionado à fila `q` para ser explorado posteriormente.

2.4.3 Custo de Transição

O custo para alcançar cada vizinho é acumulado com base nos pesos das arestas, armazenados na estrutura `cost`. A transição do nó atual para o vizinho adiciona o peso da aresta correspondente ao custo acumulado do nó atual.

2.4.4 Critério de Parada

O algoritmo termina quando o nó objetivo `end` é encontrado. Caso a fila de nós a explorar seja esvaziada sem encontrar o objetivo, o algoritmo retorna um custo de `-1`, indicando que não há caminho entre os nós.

2.5 IDS

O Aprofundamento Iterativo (IDS) é um algoritmo que combina a abordagem da busca em profundidade (DFS), explorando estados até um limite máximo de profundidade, com a completude da busca em largura (BFS), ao incrementar progressivamente o limite de profundidade em cada iteração. Ele recomeça a busca a partir do estado inicial para cada novo limite até encontrar o objetivo ou esgotar o espaço de busca.

2.6 Modelagem dos Componentes da Busca

A modelagem dos componentes fundamentais da busca é descrita a seguir:

2.6.1 Estado

Mesma representação do algortimo anterior.

2.6.2 Função Sucessora

A função sucessora itera sobre a lista de vizinhos (`neighbors`) de um nó atual (`current`), retornando todos os estados acessíveis diretamente a partir do nó atual.

2.6.3 Critérios de Parada

- A busca termina imediatamente quando o estado atual (`current`) é igual ao estado objetivo (`goal`).
- No caso do *depth-limited search*, a busca também para se a profundidade máxima permitida (`depth`) for atingida.

2.6.4 Estratégia de Exploração

- **Aprofundamento Iterativo (IDS):**
  - Explora o grafo com profundidades progressivamente maiores.
  - Cada iteração invoca uma busca em profundidade limitada (`depthLimitedSearch`), que percorre o grafo até a profundidade máxima definida para aquela iteração.
- **Busca em Profundidade Limitada:**
  - A busca recursiva é limitada pela profundidade especificada.
  - Se a profundidade permitida for atingida antes de encontrar o objetivo, a busca retorna sem sucesso.

2.7 Fluxo do Algoritmo

3 Uniform Cost Search

A função implementa o algoritmo de Busca de Custo Uniforme (*Uniform Cost Search*), que encontra o menor caminho em um grafo ponderado com base nos custos acumulados. Este algoritmo utiliza uma fila de prioridade para garantir que o nó com o menor custo seja sempre explorado primeiro. A seguir, detalhamos os componentes modelados e o fluxo do algoritmo.

3.1 Estruturas Utilizadas

- **Fila de prioridade (`priority_queue`)**: Utilizada para manter os estados a serem explorados, ordenados pelo custo acumulado. Os estados com menor custo são processados primeiro.
- No mais, usa os mesmos tipos de de estrutura para controlar os nós já visitados, a variável `visited`.

3.1.1 Função Sucessora

- Para cada nó expandido, os vizinhos conectados ao nó atual são identificados como sucessores.
- O custo de transição para cada sucessor é calculado somando o custo acumulado do nó atual ao peso da aresta conectando ao sucessor.

3.1.2 Critérios de Parada

- A busca termina quando o nó objetivo é extraído da fila de prioridade, indicando que o menor caminho foi encontrado.
- Se a fila de prioridade for esvaziada sem alcançar o objetivo, conclui-se que não há caminho possível.

4 Busca Gulosa

A função implementa o algoritmo de Busca Gulosa, que utiliza uma heurística para orientar a busca em direção ao objetivo. A heurística empregada é a distância de Manhattan, amplamente utilizada em ambientes bidimensionais devido à simplicidade de cálculo e eficiência.

4.1 Função Heurística: Distância de Manhattan

A heurística utilizada é definida como a distância de Manhattan entre dois pontos, dada pela fórmula:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

- **Descrição:** Mede o custo estimado de alcançar o objetivo a partir de um nó atual, assumindo que o movimento é limitado a direções ortogonais (horizontal e vertical).
- **Aplicação:** No algoritmo de Busca Gulosa, a distância de Manhattan é utilizada para priorizar a exploração dos nós mais próximos do objetivo em termos de estimativa heurística.

4.2 Modelagem dos Componentes da Busca

4.2.1 Estado

Mesma descrição do algoritmo anterior.

4.2.2 Função Sucessora

- A função sucessora itera sobre os vizinhos (`neighbors`) do nó atual, retornando todos os estados diretamente acessíveis a partir do nó atual.
- Cada sucessor é priorizado com base na estimativa heurística calculada pela distância de Manhattan em relação ao nó objetivo.

4.2.3 Critérios de Parada

- A busca termina imediatamente quando o nó objetivo (`goal`) é retirado da fila de prioridade, indicando que o caminho mais promissor foi explorado até o objetivo.
- Se a fila de prioridade for esvaziada sem encontrar o objetivo, conclui-se que não há caminho possível.

5 A-Estrela

O algoritmo A* (A-Estrela) é uma técnica de busca informada que combina as vantagens da Busca de Custo Uniforme (UCS) e da Busca Gulosa. Ele utiliza uma função de avaliação $f(n) = g(n) + h(n)$, onde:

- $g(n)$: O custo acumulado para chegar ao nó n a partir do nó inicial.
- $h(n)$: Uma heurística que estima o custo de n ao objetivo.

5.1 Estruturas Utilizadas

As estruturas específicas desse algoritmo são:

- **Fila de prioridade (`priority_queue`):**
  - Utilizada para armazenar os nós a serem explorados, ordenados pela função de avaliação $f(n)$.
- `gCost`:

  - Um mapa que armazena o custo acumulado $g(n)$ para chegar a cada nó n a partir do nó inicial.

- `fCost`:
  - Um mapa que armazena o valor da função de avaliação $f(n) = g(n) + h(n)$ para cada nó.

5.2 Modelagem dos Componentes da Busca

5.2.1 Estado

Mesma representação do algoritmo anterior.

5.2.2 Função Sucessora

- Para cada nó expandido, a função sucessora retorna os nós vizinhos (`neighbors`) conectados ao nó atual.
- O custo de transição para cada sucessor é calculado como o custo acumulado do nó atual ($g(n)$) somado ao peso da aresta conectando ao sucessor.

5.2.3 Função de Avaliação $f(n)$

- A função de avaliação é definida como:

$$f(n) = g(n) + h(n)$$

  onde:
  - $g(n)$: Custo acumulado para alcançar o nó atual a partir do nó inicial.
  - $h(n)$: Heurística utilizada para estimar o custo restante até o objetivo. Neste caso, é utilizada a distância de Manhattan.

5.2.4 Critérios de Parada

- A busca termina imediatamente quando o nó objetivo (`goal`) é retirado da fila de prioridade.
- Se a fila de prioridade for esvaziada sem alcançar o objetivo, conclui-se que não há caminho possível.

6 Análise Comparativa dos Algoritmos

6.1 Principais Diferenças

Os algoritmos implementados apresentam diferenças fundamentais na forma como exploram o espaço de estados e na avaliação de qual estado explorar em seguida:

- **Busca em Largura (BFS):**
  - **Descrição:** Explora todos os estados no mesmo nível de profundidade antes de avançar para níveis mais profundos.
  - **Vantagens:** Completo e ótimo para custos uniformes.
  - **Complexidade de Tempo:** $O(b^d)$, onde:

    - $*$ b: Fator médio de ramificação do grafo.
    - $*$ d: Profundidade do objetivo no espaço de estados.
  - **Complexidade de Espaço:** $O(b^d)$, pois todos os estados no mesmo nível devem ser armazenados simultaneamente.
- **Aprofundamento Iterativo (IDS):**
  - **Descrição:** Combina a eficiência em memória da busca em profundidade com a completude da busca em largura. Explora o grafo com profundidades progressivamente maiores.
  - **Vantagens:** Uso eficiente de memória.
  - **Complexidade de Tempo:** $O(b^d)$, devido ao reprocessamento de nós em profundidades menores.
  - **Complexidade de Espaço:** $O(bd)$, pois mantém apenas os estados no caminho atual.
- **Busca de Custo Uniforme (UCS):**
  - **Descrição:** Sempre expande o estado com o menor custo acumulado $g(n)$.
  - **Vantagens:** Completo e ótimo, independentemente dos custos.
  - **Complexidade de Tempo:** $O(b^{C^*/\epsilon})$, onde:
    - $*$ b: Fator médio de ramificação.
    - $*$ C^*: Custo do menor caminho até o objetivo.
    - $*$ ϵ: Custo mínimo da aresta.
  - **Complexidade de Espaço:** $O(b^{C^*/\epsilon})$, devido ao armazenamento de todos os nós gerados.
- **Busca Gulosa:**
  - **Descrição:** Prioriza a exploração do estado que parece mais próximo do objetivo, com base em uma heurística $h(n)$.
  - **Vantagens:** Rápida em muitos casos.
  - **Complexidade de Tempo:** $O(b^d)$, onde:
    - $*$ b: Fator médio de ramificação.
    - $*$ d: Profundidade do objetivo no espaço de estados.
  - **Complexidade de Espaço:** $O(b^d)$, pois a fila de prioridade pode conter todos os estados gerados.
- **A*:**
  - **Descrição:** Combina os méritos da UCS e da Busca Gulosa, utilizando a função de avaliação $f(n) = g(n) + h(n)$.
  - **Vantagens:** Completo e ótimo, desde que a heurística seja admissível e consistente.
  - **Complexidade de Tempo:** $O(b^d)$, onde:
    - $*$ b: Fator médio de ramificação.
    - $*$ d: Profundidade da solução ótima no espaço de estados.

- **Complexidade de Espaço:** $O(b^d)$, devido ao armazenamento de todos os nós gerados na fila de prioridade.

6.2 Especificação das Heurísticas Utilizadas

A heurística utilizada nos algoritmos informados é a **Distância de Manhattan**, definida como:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

- **Admissibilidade:**
  - A heurística é admissível porque nunca superestima o custo real para alcançar o objetivo em um espaço onde o movimento é restrito a direções ortogonais (horizontal e vertical). O custo estimado $h(n)$ é sempre menor ou igual ao custo real $g(n)$.
- **Consistência:**
  - A heurística é consistente porque respeita a desigualdade triangular:

    $$h(n) \leq h(n') + \text{custo}(n, n')$$

    Isso significa que o custo estimado de n ao objetivo nunca será maior que o custo de n' ao objetivo somado ao custo de transitar de n para n'.
- **Aplicação nos Algoritmos:**
  - **Busca Gulosa:** A heurística é utilizada diretamente para priorizar estados, ignorando o custo acumulado $g(n)$.
  - **A*:** A heurística é combinada com o custo acumulado para formar a função de avaliação $f(n) = g(n) + h(n)$, garantindo completude e otimalidade.

7 Análise Quantitativa & Resultados

7.1 Comparativo

Nesta seção, será apresentada uma análise quantitativa dos algoritmos implementados (`BFS`, `IDS`, `UCS`, `Greedy`, e `A*`) em termos de dois parâmetros principais:

- **Número de estados expandidos:** Mede a eficiência do algoritmo na exploração do espaço de estados.
- **Tempo de execução:** Representa a eficiência computacional do algoritmo.

Todos algoritmos foram implementados de forma que retornassem o tempo total de execução e o número de estados expandidos, além do caminho entre o ponto inicial e final. Desta forma foi possível coletar os dados plotados a seguir. No mais, os experimentos foram conduzidos em mapas diferentes. Abaixo estão os resultados em forma de tabelas e gráficos.

7.1.1 Gráficos Comparativos

O programa foi testado na seguinte instância:`./pathfinder mapas/cidade.map BFS 1 130 166 192`

Os gráficos a seguir mostram como o número de estados expandidos e o tempo de execução variam com a distância do ponto inicial ao objetivo:

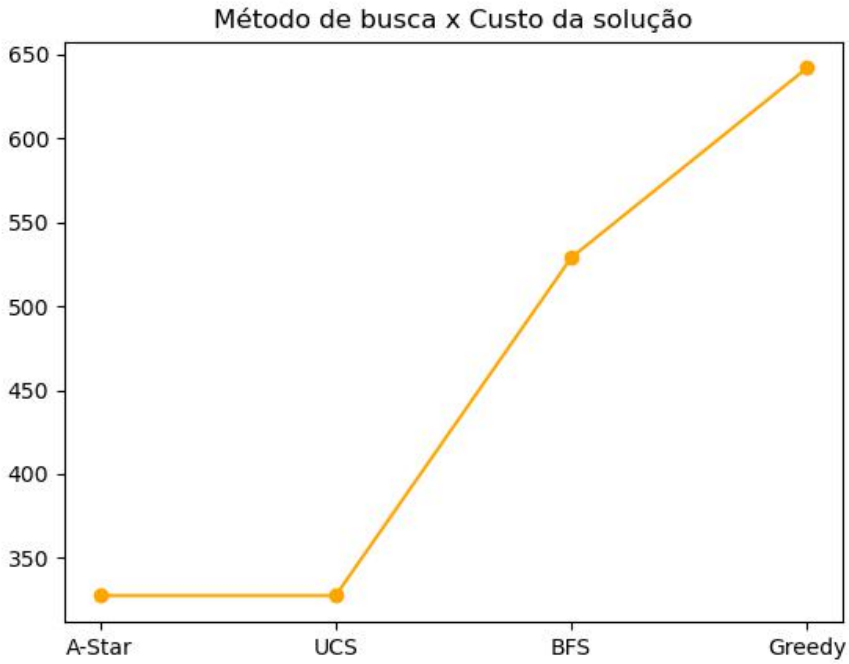
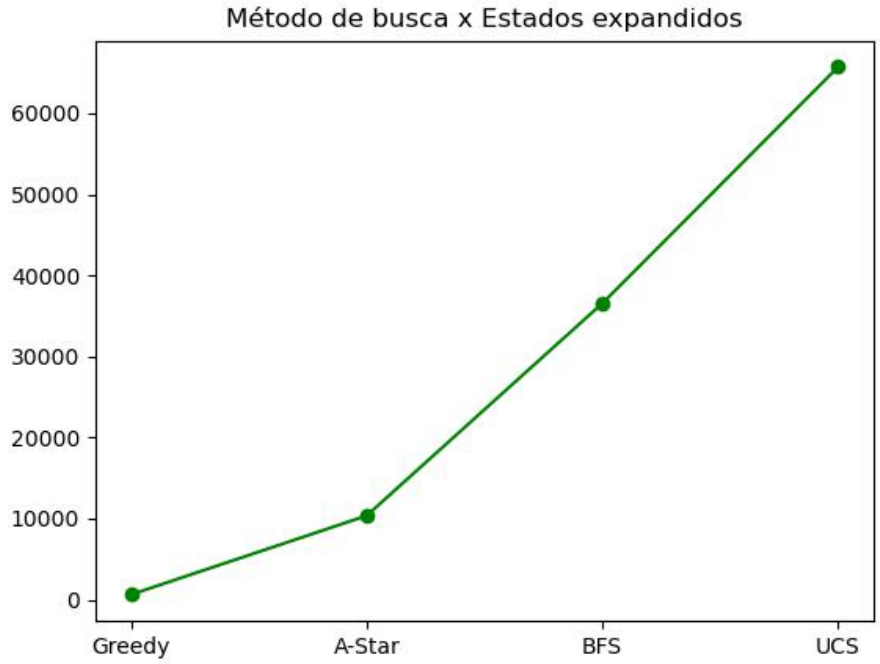


Figura 1: Custo da solução dos algoritmos em uma mesma instância.



7.2 Discussão dos Resultados

Os resultados obtidos revelam diferenças significativas no desempenho dos algoritmos em relação ao número de estados expandidos e ao tempo de execução. Abaixo estão as principais observações:

- **BFS e IDS:**
  - **Estados expandidos:** Ambos os algoritmos expandem muitos estados, especialmente em mapas com maiores distâncias, devido à falta de uma heurística.
  - **Tempo de execução:** O BFS é geralmente mais rápido que o IDS, uma vez que evita trabalho redundante ao não reexplorar profundidades menores. Para instâncias grandes, o IDS demorou bastante na execução, e não foi possível plotar resultados. -**Otimalidade:** Para o BFS ser ótimo, é necessário que a função de custo seja uma função não decrescente da profundidade do nodo, o que não é o caso neste problema. Assim, o BFS não retorna soluções de custo ótimo.
- **UCS:** A busca de custo uniforme (UCS) é um algoritmo que expande os estados com base no custo acumulado $g(n)$, sempre priorizando o menor custo na fila de prioridade. Apesar de garantir completude e otimalidade, UCS pode expandir mais estados e ser mais demorada do que a busca em largura (BFS) em determinados cenários. A seguir,os motivos possíveis para isso são explicados.
  - **Exploração Baseada em Custos:**
    - O UCS prioriza estados com menor custo acumulado, mesmo que esses estados estejam mais distantes do objetivo.
    - Isso pode levar o algoritmo a expandir muitos estados intermediários que não contribuem diretamente para alcançar o objetivo.

    - Em contraste, a BFS expande os estados em níveis de profundidade, sem considerar custos, o que pode ser mais eficiente em ambientes com custos uniformes.

- **Custos Variáveis:**
    - Em grafos com custos de arestas variáveis, o UCS precisa calcular e comparar todos os custos possíveis para os vizinhos de um estado.
    - Isso aumenta significativamente o número de estados analisados em comparação com a BFS, que ignora os custos.

- **Reavaliação de Estados:**
    - Estados podem ser inseridos na fila de prioridade várias vezes, caso sejam alcançados por caminhos com custos diferentes.
    - Apenas o menor custo é processado, mas as múltiplas inserções aumentam o número de operações.
    - Esse comportamento não ocorre na BFS, que insere cada estado uma única vez.

- **Complexidade do UCS:**
    - A complexidade de tempo do UCS é $O(b^{C^*/\epsilon})$;
    - Já a BFS tem complexidade $O(b^d)$, onde d é a profundidade do objetivo.
    - Quando C^*/ϵ é significativamente maior que d, o UCS tende a expandir mais estados que a BFS.

- **Estrutura da Fila de Prioridade:**
    - O UCS utiliza uma fila de prioridade, que realiza inserções e remoções ordenadas. Isso consome mais tempo computacional do que a fila FIFO simples da BFS.

- **Estados com Caminhos de Custo Similar:**
    - Em grafos com muitos caminhos de custo próximo ao ótimo, o UCS pode expandir quase todos os estados antes de encontrar o objetivo.
    - **Greedy:**
        - **Estados expandidos:** Expande poucos estados, graças ao uso da heurística, mas falha em encontrar o caminho ótimo, e retornou soluções de custo mais alto em todas instâncias testadas.
        - **Tempo de execução:** Muito rápido em mapas com boas heurísticas, mas pode ser enganado por estados não promissores.
    - **A*:**
        - **Estados expandidos:** Expande o menor número de estados entre os algoritmos depois da busca gulosa, graças à combinação eficiente de $g(n)$ e $h(n)$.
        - **Tempo de execução:** Apesar de consumir mais memória, é eficiente e garante soluções ótimas em mapas com heurísticas admissíveis e consistentes.

7.3 Conclusão da Análise

Os resultados mostram que os algoritmos informados (`Greedy` e `A*`) são mais eficientes em termos de estados expandidos e tempo de execução, especialmente em cenários

com heurísticas bem definidas. O `A*`, em particular, oferece um equilíbrio ideal entre eficiência e otimalidade, sendo o algoritmo mais indicado para mapas com custos não uniformes e grandes distâncias. Em contraste, os algoritmos não informados (`BFS`, `IDS`, e `UCS`) são menos eficientes, mas oferecem garantias de completude e, em alguns casos, de otimalidade.

References

Material usado em sala.