Luiz Henrique da Silva Gonçalves
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
Trabalho prático de Introdução à Inteligência Artificial
Matrícula:2018054559
Luiz10hdsg@ufmg.br

# 1. Introdução

O trabalho apresentado visa a implementação de 5 algoritmos de busca dentro de um cenário ilustrativo, através de abordagens e métodos diferentes,com o intuito de resolver o problema de encontrar o menor caminho entre dois pontos.. Dessa forma, o objetivo proposto é praticar os conceitos e algoritmos de busca em espaço de estados.

Sendo assim, os algoritmos propostos são os seguintes: Busca em Largura (BFS), Aprofundamento Iterativo (IDS), Busca de Custo Uniforme (UCS), Gulosa e A*.

# 2. Implementação

O programa foi desenvolvido na linguagem C + +, compilado pelo compilador G++ da GNU Compiler Collection e executado no sistema operacional Linux , na distribuição Ubuntu. A entrada dos dados é a entrada padrão do sistema(stdin) através de um arquivo texto.A saída também é a padrão do sistema(stdout).

## 2. 1 Estrutura dos Arquivos

Para modularizar a implementação do sistema e agilizar um pouco o processo, foi simplificado a implementação, montando somente o arquivo main.
No arquivo main.cpp contém os algoritmos referentes ao funcionamento do sistema, estruturação dos dados com a declaração de suas estruturas, com a leitura do arquivo de entrada e arquivamento dados, em relação a  modelagem pensada, junto com a declaração da classe referente ao grafo modelado e seus métodos, com a implementação dos algoritmos de busca do problema proposto.

# 3. Estruturas usadas e da modelagem dos componentes da busca

Para a modelação do problema, usamos algumas estruturas para guardar as informações necessárias para cada procedimento. A entrada de dados é estruturada em uma matriz. Cada entrada dessa matriz é composta por um símbolo com um valor correspondente.Para representar o mapa ilustrativo, foi usada uma representação de grafo. Cada símbolo do mapa representa um vértice do grafo e suas arestas indicam a possibilidade de tráfego em relação a uma posição do mapa até outro. Dessa forma, pelo Path-Finding, é possível caminhar pelo mapa em busca

de um vértice específico e contabilizar o custo desse caminho.Sendo assim, umas das particularidades do problema proposto é obstáculos intransponíveis e posições fora do mapa, usando uma representação por grafo, resolvemos tal problema não ligando arestas nesses vértices intransponíveis e também definindo os limites do grafo através do tamanho da matriz de entrada, ou seja estipulando exatamente um vértice para cada coordenada da matriz de entrada.

O programa implementado, ler o arquivo de entrada que contém a matriz de dados, para cada coordenada ler os símbolos e mapeia tais símbolos ao seu custo correspondente em uma estrutura de dados vetor.Em seguidas passamos novamente pelo arquivo e construirmos o grafo, adicionando suas arestas e depois mapeando para cada vértice uma coordenada em relação ao matriz de entrada e guardamos em pair de valores em um vetor, ou seja é possível dessa forma acessar para cada vértice do grafo sua coordenada na matriz original.

Para cada método de busca implementado, criamos uma função, cada uma com abordagens, métodos e estruturas particulares.

O primeiro passo a que é executado pelo programa implementado é a adoção dos custos do caminho do arquivo de entrada e construção do grafo e mapeamento dos vértices do grafo as coordenadas da matriz.

## 4. Descrição dos algoritmos.

### 4.1) BFS (Breadth-first search)

No algoritmo BFS é feita uma travessia, onde inicia-se a partir de um nó selecionado (no final da busca) e percorre o grafo em camadas explorando os nós vizinhos (nós que estão diretamente conectados ao nó de origem). Você deve então se mover em direção aos nós vizinhos do próximo nível.

- Primeiro mova horizontalmente e visite todos os nós da camada atual
- Mover para a próxima camada
- Esse processo se repete até encontrar a posição target procurada
- Durante a travessia calculamos o custo desse caminho olhando a estrutura de dados responsável por guardar os custos de cada coordenada do mapa.

#### Implementação

- Pegamos as coordenadas dos pontos inicial e final e mapeamos tais pontos em vértices do grafo, e a partir deles seguimos na busca percorrendo com o BFS, guardando o caminho seguido.
- É declarada uma fila e inserimos o vértice inicial.
- Inicializamos um array visitado e marcamos o vértice inicial como visitado.
- Seguimos o processo abaixo até que a fila fique vazia:
    - Remova o primeiro vértice da fila.

- Marcamos esse vértice como visitado.
- Inserimos todos os vizinhos não visitados do vértice na fila

. **4.2) IDS (Aprofundamento Iterativo)**

O algoritmo implementado pesquisa cada ramo de um nó da esquerda para a direita até atingir a profundidade necessária, que aqui consideramos como o tamanho limite do grafo que também é o tamanho da matriz de entrada. Depois disso, o IDS volta ao nó raiz e explora uma ramificação diferente semelhante ao DFS.

**Implementação**

- Os passos iniciais seguem a mesma estratégia adotada no BFS (a propósito a mesma estratégia inicial é seguida para todos os algoritmos de busca), iniciando com o mapeamento dos vértices dos pontos de entrada e cálculo dos símbolos em relação ao custo .
- Se o nó objetivo estiver em uma profundidade menor que d, o caminho subótimo pode ser alcançado, enquanto no caso de profundidade maior que d, o nó objetivo será inalcançável.
- Na implementação, é adotado duas funções **`IDDFS`** e **`IDS`**
- O DLS falha nos dois casos a seguir: O nó objetivo está além do limite de profundidade ou não há caminho, pois o nó objetivo não existe no grafo.
- Para o IDDFS, o algoritmo aplica repetidamente a pesquisa limitada em profundidade com limites crescentes.Ele aumenta gradualmente os limites de 0,1,...d, até que o nó objetivo seja encontrado.
- Ele termina nos dois casos seguintes: Quando o nó objetivo é encontrado ou o nó objetivo não existe no grafo.

### 4.3) UCS (Busca de Custo Uniforme)

A busca de custo uniforme é um algoritmo de pesquisa sem informado que usa o menor custo cumulativo para encontrar um caminho da origem ao destino. Os nós são expandidos, a partir da raiz, de acordo com o custo cumulativo mínimo. A pesquisa de custo uniforme é então implementada usando uma estrutura de fila de prioridade, nesse caso adotamos a fila para priorizar o no filho que obtém o menor custo acumulado em relação ao caminho adotado no mapa de entrada.

**Implementação**

- É inserido o nó raiz(vértice inicial) na fila de prioridade.
- Remova o elemento com a prioridade mais alta( menor custo acumulado ).
- *Se o nó removido for o destino,* imprima o custo total e o caminho feito e pare o algoritmo *Else if* , Verifique se o nó está na lista visitada
- *Else* Enfileira todos os filhos do nó atual para a fila de prioridade, com seu custo cumulativo da raiz como prioridade e o atual não para a lista visitada.

## 4.4) Greedy (Guloso)

Para a implementação desse algoritmo adotamos a estratégia best fit search **(bfs)** que adota essa busca ganancioso do melhor custo possível, primeiro o algoritmo tenta explorar o nó que está mais próximo do objetivo. Este algoritmo avalia os nós usando a função heurística h(n), que nesse caso da implementação, adotamos uma função de avaliação que é igual à função heurística, f(n) = h(n). Essa equivalência é o que torna o algoritmo de busca 'ganancioso'. Na implementação f(n)=h(n), sendo assim algoritmo escolherá em cada estágio explorar o nó que sabemos ser o caminho de menor custo em relação a posição atual do mapa, até chegar no no objetivo.

**Implementação**

- Usamos uma fila de prioridade  para armazenar os custos dos nós que possuem o menor valor da função de avaliação. Então a implementação é uma variação do BFS, mas pegando o caminho de menor custo.
- A partir de um nó inicial percorremos o grafo comparando os custos dos nós vizinhos ao mesmo, e seguimos pelo de menor custo. Continuamos esse processo até encontrar o vértice procurado.

## 5. Análise quantitativa dos métodos e discussões dos resultados

Cada método implementado apresenta características próprias em prol do seu funcionamento de busca, adotando diferentes abordagens em relação a adoção de estruturas de dados e caminhamentos através de loops e travessias no grafo. É evidente que a demora de execução de todos os algoritmos implementados é provocada pelo tamanho dos dados de entrada. Para a implementação feita, adotamos um grafo por meio de uma lista de adjacência  para representar o mapa de custos. Para guardar os custos do mapa e guardar as coordenadas de cada grafo, percorremos duas vezes o arquivo com a matriz de entrada, isso obviamente aumenta o custo de execução do programa, outras abordagens mais eficientes poderiam ter sido tomadas, mas por simplicidade adotamos essa. Outra abordagem adotada é buscar as coordenadas dos pontos de inicial e final na execução dos dados e também na estrutura que guarda os caminhos feitos, visto que nas buscas feitas, apenas adotamos os vértices do grafo não suas coordenadas. Outrossim, temos também os custos de complexidade dos algoritmos de busca implementados, de novo, a complexidade de tempo vai andar junto com o tamanho da matriz de entrada, sendo esse o gargalo de tempo do problema. Nos testes feitos com dados de entrada grandes percebemos que o tempo de execução supera os 7 minutos em algumas das implementações.Analisando a performance de cada implementação em relação ao maior arquivo de teste fornecido.

BFS : Complexidade de tempo: O(V+E), onde V é o número de nós e E é o número de arestas. Nos testes feitos o tempo de execução superou os 10 minutos.

IDS: Parece que o IDS tem uma grande sobrecarga na forma de passar repetidamente pelos mesmos nós repetidamente, mas isso acaba sendo um grande problema. Isso ocorre porque a maioria dos nós do grafo têm vértices de níveis

inferiores, que são visitados apenas uma ou duas vezes pelo algoritmo. Por causa disso, o custo é mantido no mínimo, pois os nós de nível superior não compõem a maioria dos nós em uma árvore.Para um problema com fator de ramificação b onde a primeira solução está na profundidade k, a complexidade de tempo do aprofundamento iterativo é O(b^K).Nos testes feitos foi o mais demorado, considerando um tempo considerável para a execução.

UCS:Nos testes foi também levou um tempo considerável para executar, chegando a mais de 5 minutos na execução.A busca de custo uniforme é guiada por custos de caminho ao invés de profundidades, então sua complexidade não é facilmente caracterizada em termos de b e d. Em vez disso, seja C o custo da solução ótima e suponha que cada ação custa pelo menos ε. Então, a complexidade de tempo do pior caso do algoritmo é O(b^(1+C/ε)), que pode ser muito maior que b^d.

Greedy: Nos testes foi bem lento também, superando os 10 minutos.O algoritmo percorre o caminho mais curto pelo menor custo que vai estar em primeiro na fila. A complexidade de tempo do algoritmo é dada por O(n*logn). minutos de execução.

**OBS:** Em razão ao tempo dedicado a implementação dos métodos e o tempo curto para entrega do trabalho prático, não foi possível a implementação do último algoritmo proposto, o algoritmo A*.