

Inteligência Artificial

Docente Joaquim Gonçalves

Licenciatura em Engenharia Informática Médica

2023/2024

**Algoritmos de Pesquisa**

*Profundidade, largura e A\**

23544 - Ana Beatriz Machado Carvalho

23548 – Ana Margarida Maia Pinto

23552 - Diana Alexandra da Costa Dinis

Índice

[1. Introdução 1](#_Toc152600058)

[1.1. Enquadramento 1](#_Toc152600059)

[1.2. Estrutura do documento 1](#_Toc152600060)

[2. Algoritmos de Pesquisa 2](#_Toc152600061)

[2.1. Largura 2](#_Toc152600062)

[2.2. Profundidade 3](#_Toc152600063)

[2.3. A\* 4](#_Toc152600064)

[3. Testes e Resultados 7](#_Toc152600065)

[3.1. Preparação 7](#_Toc152600066)

[3.2. Resultados com um objetivo 7](#_Toc152600067)

[3.3. Resultados com dois objetivos 15](#_Toc152600068)

[4. Conclusão 16](#_Toc152600069)

[4.1. Ferramentas utilizadas 16](#_Toc152600070)

[5. Bibliografia 18](#_Toc152600071)

**Índice de Figuras**

[Figura 1 - Representação gráfica do algoritmo de pesquisa em largura 2](#_Toc152600072)

[Figura 2 - Representação gráfica do algoritmo de pesquisa em profundidade 3](#_Toc152600073)

[Figura 3 - Visualização gráfica de uma matriz com a heurística 4](#_Toc152600074)

[Figura 4 - Visualização gráfica de um ponto de decisão do A\* 5](#_Toc152600075)

[Figura 5 - Ponto de decisão por causa do custo 5](#_Toc152600076)

[Figura 6 - Decisão de "voltar atrás" do A\* 5](#_Toc152600077)

[Figura 7 - Caminho escolhido pelo algoritmo A\* 6](#_Toc152600078)

[Figura 8 - DFS para a matriz2530 com fim a (9,2) 8](#_Toc152600079)

[Figura 9 - BFS para a matriz2530 com fim a (9,2) com tempo de espera 8](#_Toc152600080)

[Figura 10 - BFS para a matriz2530 com fim a (9,2) com tempo de espera 9](#_Toc152600081)

[Figura 11 - Representação visual da matriz 10045 15](#_Toc152600082)

**Índice de Tabelas**

[Tabela 1 - Tabela de resultados para 30% de obstáculos com um objetivo 12](#_Toc152600083)

[Tabela 2 - Tabela de resultados para 50% de obstáculos com um objetivo 14](#_Toc152600084)

**Lista de siglas e acrónimos**

**BFS** Algoritmo de pesquisa em largura

**DFS** Algoritmo de pesquisa em profundidade

**FIFO** *First In First Out*

**FILO** *First In Last Out*

**IA** Inteligência Artificial

# Introdução

Nos dias de hoje, há uma crescente adesão à tecnologia, sendo a automação e inteligência artificial inerente.

No contexto de armazéns, a automação deste pode melhorar a produtividade logística, tendo efeitos também na segurança dos trabalhadores, com a diminuição de acidentes de trabalho com a maquinaria pesada utilizada em armazéns de grande porte.

O trabalho terá como plano a automação de um armazém de uma empresa de encomendas, onde um *robot* irá receber um pedido de encomenda de um item e terá de calcular um caminho desde a sua posição até à localização do item encomendado no armazém.

A fase inicial do trabalho será a implementação de um único *robot* com um único objeto de encomenda.

A fase seguinte será a implementação de um único *robot* com dois objetivos.

O trabalho será implementado, em *Python*, uma representação do armazém, assim como os algoritmos que o *robot* terá de modo a conseguir uma solução ao problema. Serão avaliados o tempo que o algoritmo demora a encontrar uma solução e se essa solução será a solução ótima. Este conceito aplicado a este projeto estará explicado posteriormente.

## Enquadramento

Este trabalho enquadra-se da unidade curricular de Inteligência Artificial lecionada pelo docente Joaquim Gonçalves da Licenciatura de Engenharia Informática Médica, no Instituto Politécnico do Cávado e do Ave.

## Estrutura do documento

Este documento encontra-se dividido em essencialmente cinco capítulos. Inicia com o contexto do trabalho, onde estará exposto os objetivos do trabalho. Segue-se com o capítulo dos algoritmos de pesquisa, onde estará uma introdução teórica a cada um dos algoritmos escolhidos. A seguir, o capítulo dos testes e resultados, onde os algoritmos serão aplicados ao problema em Python e, por fim, a conclusão, onde os resultados serão comparados e será escolhido o melhor algoritmo para cada situação.

# Algoritmos de Pesquisa

Este capítulo tratará da introdução teórica e implementação de cada algoritmo escolhido para o desenvolvimento do trabalho prático.

Os algoritmos de pesquisa, de modo a conseguirem prosseguir com a procura, utilizam uma *frontier*, que irá gerir os nós a expandir, de modo que o algoritmo saiba as opções que tem para prosseguir com a procura, sendo um conceito comum dos algoritmos.

## Largura

O algoritmo de pesquisa em largura, ou BFS, irá seguir múltiplas direções ao mesmo tempo, explorando vários caminhos simultaneamente. Para tornar isso possível, a *frontier*, será montada como uma *queue*, tendo como método o FIFO – *First In First Out*.

Assim, todos os novos caminhos possíveis são adicionados à fila e expandidos conforme a ordem que foram adicionados. Montando uma árvore, visível na Figura 1, observa-se que se expande todos os nós do mesmo nível antes de passar para outro nível:

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Figura 1 - Representação gráfica do algoritmo de pesquisa em largura

Para isto, na definição de *frontier*, que será a *queueFrontier,* definida como a classe deque da biblioteca *collections*, neste caso será feito a eliminação do primeiro item guardado na variável, com recurso ao método popleft(), como se pode observar a seguir:

queueFrontier.popleft()

O par de coordenadas eliminado será armazenado como (x, y).

Caso esse par seja o fim, o caminho será reconstruído a partir do dicionário de pais, definido no início da função de bfs.

Se não, adiciona os novos vizinhos à *frontier* que tenham coordenadas válidas conforme a matriz selecionada, guardado o parente destes.

As vantagens do algoritmo de pesquisa em largura são os seguintes:

* + Garante que a solução é o menor caminho possível;
  + Implementação simples ;
  + Fácil compreensão.

As desvantagens deste algoritmo são as seguintes:

* No pior caso, é o algoritmo que demora o maior tempo possível;
* Ocupa muita memória, mantendo todos os nós explorados.

## Profundidade

O algoritmo de pesquisa em profundidade, ou DFS, é o oposto do algoritmo de pesquisa em largura. Enquanto o algoritmo em largura exausta todos os nós a expandir do mesmo nível, seguindo vários caminhos ao mesmo tempo, o algoritmo em profundidade segue apenas um caminho até ao fim, representado na Figura 2.

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Figura 2 - Representação gráfica do algoritmo de pesquisa em profundidade

Para isso, a estratégia deste algoritmo é programada para agir como uma *stack*, agindo em FILO – *First In Last Out*. Com isso, ao expandir os nós, o último a ser adicionado à *frontier* irá ser o próximo a ser expandido. O último nó adicionado é selecionado através do método pop() da biblioteca *collections*, da seguinte forma:

stackFrontier.pop()

Sendo armazenado também num par ordenado (x, y), seguindo a estratégia já exposta no subcapítulo do BFS para a determinação do caminho com a *frontier*.

As vantagens do algoritmo de pesquisa em profundidade:

* Tendo a sorte de o escolher, encontra o caminho de forma mais rápida;
* Não precisa de tanta memória, apenas guarda os nós do caminho atual;

As desvantagens deste algoritmo são os seguintes:

* A solução encontrada não é garantidamente a ótima;
* No pior dos casos, explora todos os caminhos possíveis até ao fim antes de encontrar a solução, demorando o maior tempo possível.

## A\*

O A\* é um algoritmo de pesquisa informada. Ao contrário dos outros dois algoritmos anteriores, este possui conhecimento do problema além da que vai recolhendo com a pesquisa.

De modo que o algoritmo saiba qual o próximo nó a expandir, em vez do conceito de *frontier*, o A\* utiliza uma função heurística, que calcula a distância da posição inicial até ao objetivo e mantém o registo do custo do início até à posição atual. Da Figura 3 à Figura 7 está representado o modo como este algoritmo encontra um caminho:

Na Figura 3 está representado um exemplo de matriz, onde os números representam a heurística da posição até ao objetivo:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 3 - Visualização gráfica de uma matriz com a heurística

Na Figura 4, o número que se encontra a somar com a heurística representa o custo desde o início até à posição atual, onde o algoritmo chega a um ponto de decisão e irá escolher o nó cuja soma do custo e da heurística seja o menor, sendo, portanto, em frente.

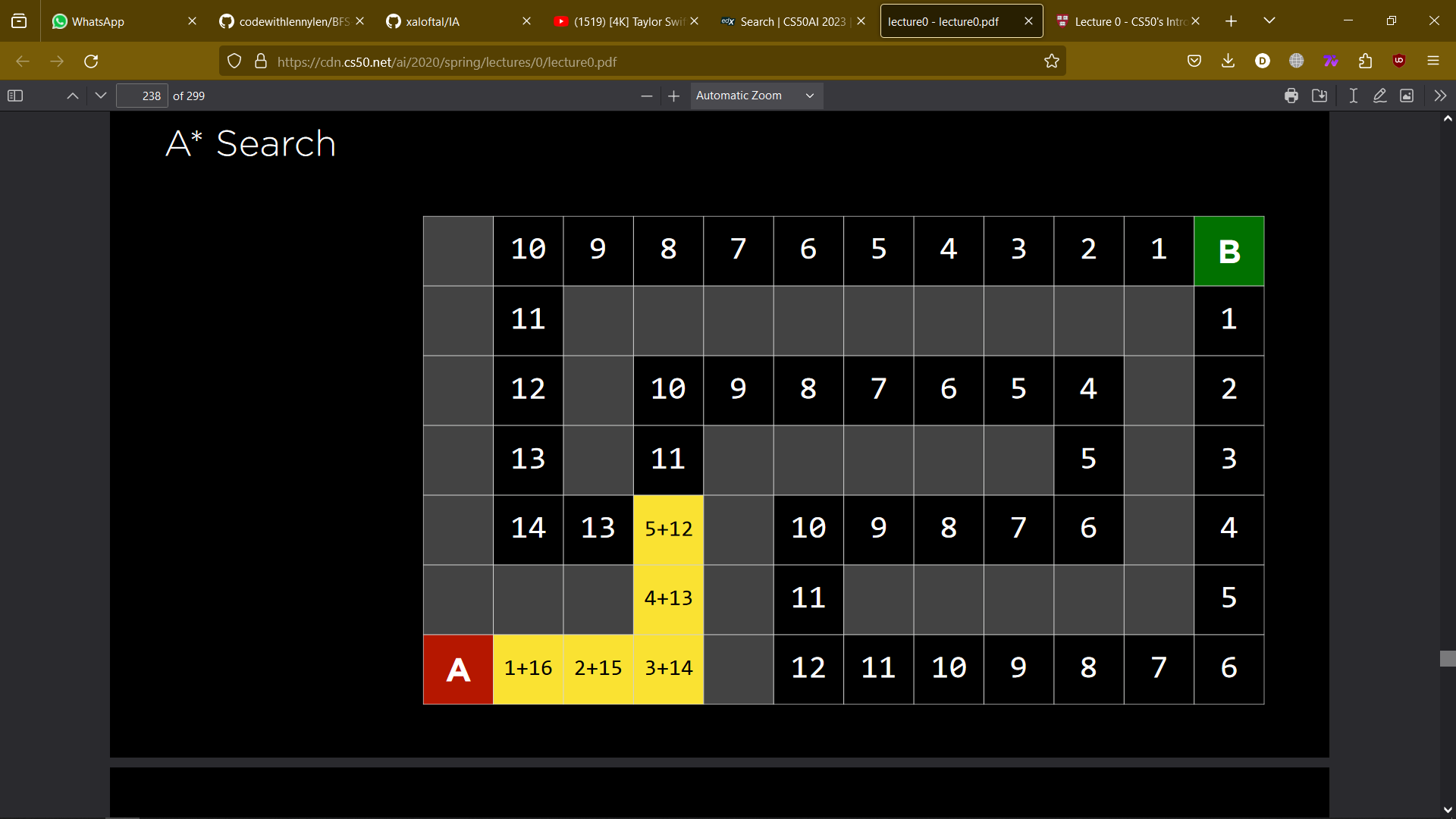


Figura 4 - Visualização gráfica de um ponto de decisão do A\*

Na Figura 5 o algoritmo chega a um nó em que o custo da posição atual mais a heurística do próximo nó é superior do que num ponto de decisão anterior:

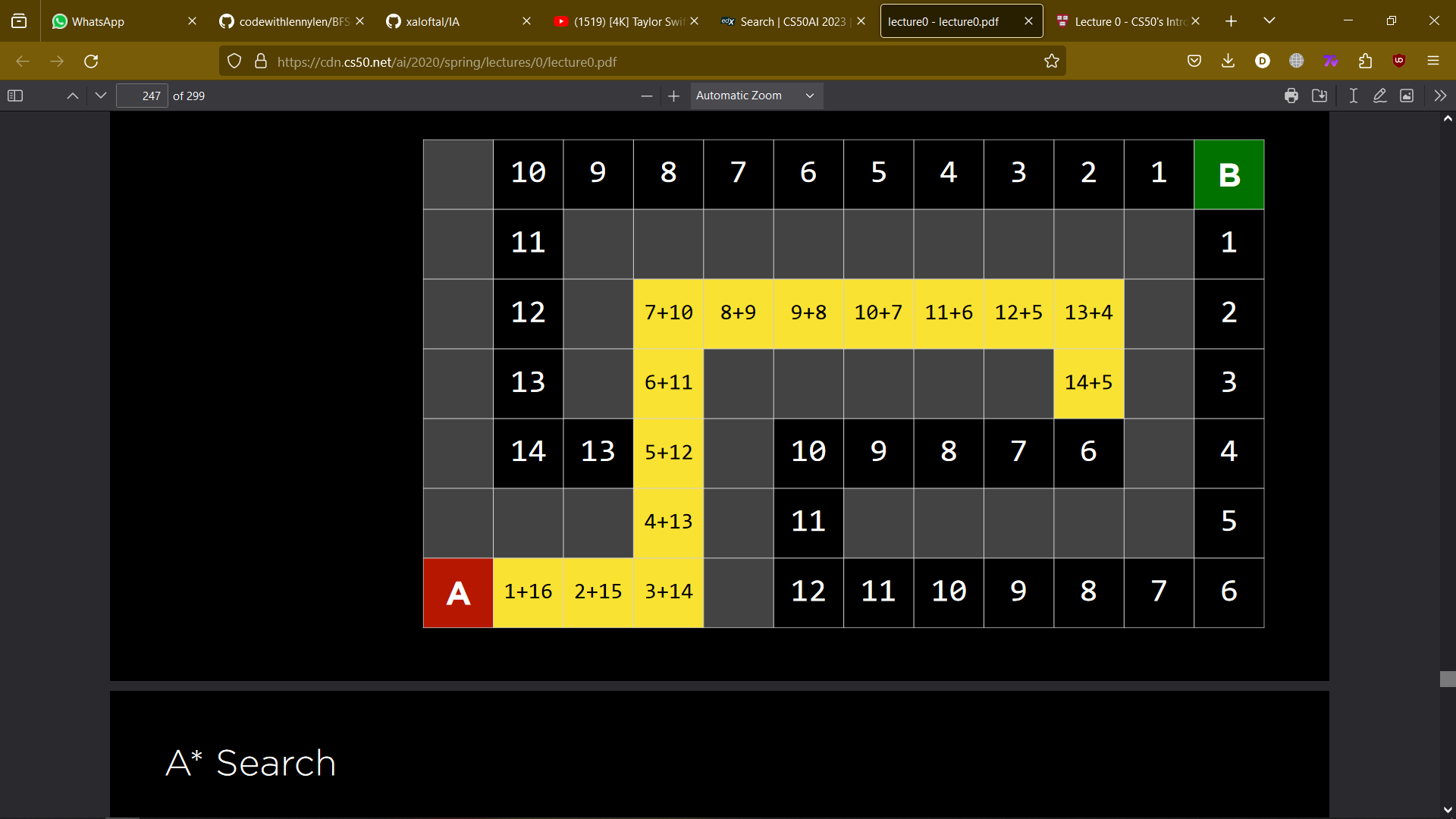


Figura 5 - Ponto de decisão por causa do custo

O algoritmo opta, portanto, por “voltar atrás”, mesmo que a heurística seja menor no nó seguinte, para explorar o nó de menor custo, como visto na Figura 6:

Uma imagem com texto, captura de ecrã, Software de multimédia, software

Descrição gerada automaticamente

Figura 6 - Decisão de "voltar atrás" do A\*

De seguida, o algoritmo escolhe continuar o caminho anterior, seguindo a seguir o segundo escolhido e, como o custo começa a ser sempre inferior a 16+7, o algoritmo segue o mesmo caminho, até ao objetivo, como se pode observar na Figura 7:

Uma imagem com texto, captura de ecrã, Software de multimédia, software

Descrição gerada automaticamente

Figura 7 - Caminho escolhido pelo algoritmo A\*

# Testes e Resultados

Neste capítulo estarão expostos os testes a cada algoritmo e a comparação entre os resultados de cada.

A solução para este projeto será o caminho encontrado pelos algoritmos desde a posição inicial até ao objetivo, ou seja, até ao item no armazém. Uma solução pode ser considerada a solução ótima quando esta for o menor caminho, com o menor custo, tendo também peso o tempo de processamento do algoritmo.

## Preparação

De modo a representar o armazém, com objetivo de testar os algoritmos, surgiu a necessidade de criar matrizes, onde 0 representa os caminhos livres e 1 as paredes. Estas serão sempre quadradas, com o mesmo número de linhas e colunas.

Com isso, definiu-se uma forma de gerar matrizes de forma automática, no ficheiro randomMatrices.py, com o método *random.choice* da biblioteca *NumPy*:

np.random.choice([0, 1], size=(10, 10), p=[0.7, 0.3])

Onde o 0.7 representa a probabilidade de aparecer 0 e 0.3 a probabilidade de 1, ou seja, a matriz gerada aqui terá o tamanho 10x10 com probabilidade de obstáculos a 30%. A inserção do tamanho da matriz e as probabilidades tem de ser manipulada diretamente no código.

A matriz, após ser gerada, é armazenada no ficheiro outputMatrix.txt e, manualmente, tem de ser guardada no ficheiro fixedMatrices.py, onde estarão todas as matrizes armazenadas, com um nome que permite identificar o seu tamanho e a probabilidade de obstáculos, da seguinte forma:

matrix (tamanho)(probabilidade de obstáculos)

Ou seja, a matriz do exemplo de cima terá o nome matrix1030.

## Resultados com um objetivo

A primeira matriz a ser testada foi a matriz2530.

Com o início nas coordenadas (0,0) e o objeto a (9,2), da Figura 8 à Figura 10 estão representados os resultados no terminal do *VSCode* para os algoritmos em profundidade, largura e A\*, respetivamente:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 8 - DFS para a matriz2530 com fim a (9,2)

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Figura 9 - BFS para a matriz2530 com fim a (9,2) com tempo de espera

No fim do processamento, de modo a conseguir apresentar o resultado sem ser como 0,0, foi introduzido um tempo de espera de 1 segundo, sendo, então, o resultado real é dado como 0,0080556869506836 segundos. De modo a serem comparáveis, a todos os tempos de processamento foi adicionado o *delay* de 1 segundo.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 10 - BFS para a matriz2530 com fim a (9,2) com tempo de espera

O algoritmo A\* apresenta como resultado também a heurística e o custo total do caminho, contando com as mudanças de direção.

Na Tabela 1 está representado testes efetuados com matrizes com 30% de obstáculos com um objetivo, onde H é a heurística, P é o tempo de processamento do algoritmo, em segundos, S o número de passos, sem contar com a orientação, C o custo final, por unidade de tempo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz1030** | 2,2 | H | - | - | 4 |
| P | 1.00116420 | 1.00155950 | 1.00257897 |
| S | 4 | 4 | 4 |
| C | 6 | 6 | 6 |
| 6,6 | H | - | - | 12 |
| P | 1.00279140 | 1.00204992 | 1.00638866 |
| S | 18 | 20 | 18 |
| C | 28 | 32 | 27 |
| 9,5 | H | - | - | 14 |
| P | 1.00301433 | 1.00358558 | 1.00770950 |
| S | 22 | 26 | 22 |
| C | 35 | 44 | 33 |
| **Matriz2530** | 9,2 | H | - | - | 11 |
| P | 1.00203562 | 1.01353097 | 1.00477171 |
| S | 13 | 37 | 13 |
| C | 16 | 50 | 16 |
| 20,24 | H | - | - | 44 |
| P | 1.01008010 | 1.00426006 | 1.02925110 |
| S | 50 | 86 | 52 |
| C | 69 | 124 | 62 |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz2530** | 15,24 | H | - | - | 39 |
| P | 1.00963545 | 1.00543404 | 1.02402639 |
| S | 49 | 83 | 49 |
| C | 67 | 121 | 59 |
| **Matriz5030** | 20,20 | H | - | - | 40 |
| P | 1.01865435 | 1.03863645 | 1.02528620 |
| S | 40 | 476 | 40 |
| C | 59 | 693 | 48 |
| 7,45 | H | - | - | 52 |
| P | 1.04536414 | 1.04878712 | 1.08652806 |
| S | 74 | 110 | 74 |
| C | 102 | 158 | 98 |
| 49,49 | H | - | - | 98 |
| P | 1.05562305 | 1.01004887 | 1.15826678 |
| S | 98 | 212 | 98 |
| C | 144 | 312 | 118 |
| **Matriz10030** | 7,15 | H | - | - | 22 |
| P | 1.00657392 | 1.16657400 | 1.01130104 |
| S | 22 | 2006 | 22 |
| C | 28 | 2901 | 25 |
| 24,67 | H | - | - | 91 |
| P | 1.08573294 | 1.23285437 | 1.15550828 |
| S | 93 | 1989 | 93 |
| C | 120 | 2873 | 114 |
| 50,50 | H | - | - | 100 |
| P | 1.09785151 | 1.10227418 | 1.18692088 |
| S | 102 | 1444 | 104 |
| C | 145 | 2075 | 124 |
| 87,23 | H | - | - | 110 |
| P | 1.16704106 | 1.08559990 | 1.25977111 |
| S | 130 | 1188 | 130 |
| C | 190 | 1703 | 162 |
| 89,99 | H | - | - | 188 |
| P | 1.21259832 | 1.05355668 | 1.41732001 |
| S | 188 | 712 | 190 |
| C | 260 | 1002 | 219 |
| **Matriz25030** | 20,15 | H | - | - | 35 |
| P | 1.01235867 | 2.10968184 | 1.02670479 |
| S | 37 | 13221 | 37 |
| C | 55 | 19433 | 46 |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz25030** | 80,214 | H | - | - | 294 |
| P | 1.89781213 | 1.62477112 | 1.88524461 |
| S | 312 | 7442 | 314 |
| C | 417 | 10918 | 378 |
| 125,175 | H | - | - | 300 |
| P | 1.87504387 | 2.23479438 | 2.58426595 |
| S | 302 | 8158 | 304 |
| C | 422 | 11992 | 363 |
| 145,203 | H | - | - | 348 |
| P | 2.16749072 | 2.28154635 | 3.02689338 |
| S | 354 | 6916 | 356 |
| C | 495 | 10156 | 426 |
| 230,240 | H | - | - | 470 |
| P | 2.38899708 | 2.41292787 | 3.61785507 |
| S | 474 | 2214 | 474 |
| C | 668 | 3276 | 562 |
| **Matriz30030** | 10,25 | H | - | - | 35 |
| P | 1.01602316 | 2.77100778 | 1.04349995 |
| S | 43 | 4375 | 43 |
| C | 56 | 6402 | 55 |
| 57,244 | H | - | - | 301 |
| P | 2.03632331 | 2.68545580 | 2.41652226 |
| S | 335 | 2293 | 339 |
| C | 434 | 3338 | 408 |
| 150,150 | H | - | - | 300 |
| P | 1.80108166 | 2.33435774 | 2.27136159 |
| S | 302 | 14020 | 304 |
| C | 450 | 20587 | 365 |
| 170,236 | H | - | - | 406 |
| P | 2.55018115 | 1.47505808 | 3.38367772 |
| S | 410 | 7424 | 412 |
| C | 579 | 10926 | 494 |
| 285,280 | H | - | - | 565 |
| P | 3.03281331 | 1.08241057 | 6.67739487 |
| S | 577 | 1687 | 585 |
| C | 843 | 2489 | 703 |
| **Matriz40030** | 20,15 | H | - | - | 35 |
| P | 1.00884962 | 1.01345134 | 1.01531230 |
| S | 35 | 227 | 35 |
| C | 47 | 324 | 45 |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz40030** | 148,365 | H | - | - | 513 |
| P | 3.78244400 | 4.29580593 | 5.50188088 |
| S | 531 | 28359 | 535 |
| C | 716 | 41425 | 664 |
| 200,200 | H | - | - | 400 |
| P | 2.56183648 | 3.07781935 | 3.44757605 |
| S | 402 | 18232 | 402 |
| C | 572 | 26601 | 494 |
| 365,254 | H | - | - | 619 |
| P | 4.57950211 | 2.24433804 | 7.18098235 |
| S | 627 | 8229 | 633 |
| C | 920 | 11918 | 761 |
| 380,380 | H | - | - | 760 |
| P | 4.53331947 | 1.86880231 | 13.74813533 |
| S | 762 | 3564 | 770 |
| C | 1089 | 5137 | 942 |
| **Matriz50030** | 36,22 | H | - | - | 58 |
| P | 1.04492068 | 6.77946758 | 1.11498642 |
| S | 68 | 42410 | 68 |
| C | 96 | 62027 | 80 |
| 194,421 | H | - | - | 615 |
| P | 4.88392234 | 5.38607812 | 6.82255340 |
| S | 637 | 31617 | 639 |
| C | 849 | 46180 | 784 |
| 249,238 | H | - | - | 487 |
| P | 3.47062302 | 6.80727792 | 5.05881596 |
| S | 487 | 28707 | 495 |
| C | 697 | 41937 | 592 |
| 350,379 | H | - | - | 729 |
| P | 5.84736872 | 2.58046269 | 10.90567636 |
| S | 731 | 14685 | 745 |
| C | 1041 | 21432 | 884 |
| 480,480 | H | - | - | 960 |
| P | 6.89649749 | 1.96339464 | 16.89161181 |
| S | 960 | 3182 | 970 |
| C | 1380 | 4636 | 1160 |

Tabela 1 - Tabela de resultados para 30% de obstáculos com um objetivo

Tendo a Tabela 1 como exemplo, para armazéns maiores com menos paredes e o objeto mais distante da zona inicial, os robôs conseguem encontrar um caminho mais rapidamente utilizando um algoritmo de pesquisa em profundidade, no entanto, o custo real deste é significativamente muito superior ao custo dos outros dois algoritmos, que possuem um custo parecido, sendo melhor o algoritmo de pesquisa em largura por causa do tempo de processamento, que pode ter diferenças de até 4 segundos para com o A\*. Para objetivos mais próximos do início, o A\* seria a melhor opção, priorizando o custo real. Observando, por exemplo, a matriz 10030 para o objetivo a (24, 67), os dois algoritmos, BFS e A\*, percorrem o mesmo número de nós, no entanto, o custo real do A\* é menor por realizar menos mudanças de direção, diminuindo o tempo que este perde a percorrer o mesmo número de passos, tornando-se mais eficiente, não havendo mudanças significativas no tempo de processamento para um objetivo tão próximo.

Na Tabela 2 estão os resultados dos testes efetuados com matrizes com probabilidade de 45% de obstáculos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz1045** | 2,3 | H | - | - | 5 |
| P | 1.00052547 | 1.00201702 | 1.00156784 |
| S | 7 | 7 | 7 |
| C | 9 | 10 | 9 |
| 5,5 | H | - | - | 10 |
| P | 1.00113153 | 1.00242615 | 1.00190306 |
| S | 10 | 12 | 10 |
| C | 13 | 18 | 13 |
| 9,8 | H | - | - | 17 |
| P | 1.00140953 | 1.00167060 | 1.00283027 |
| S | 17 | 19 | 17 |
| C | 24 | 30 | 21 |
| **Matriz2545** | 9,2 | H | - | - | 11 |
| P | 1.00685763 | 1.00586200 | 1.00732756 |
| S | 45 | 47 | 45 |
| C | 61 | 64 | 60 |
| 15,8 | H | - | - | 23 |
| P | 1.00369143 | 1.00218868 | 1.00481272 |
| S | 39 | 45 | 39 |
| C | 53 | 60 | 53 |
| 24,22 | H | - | - | 46 |
| P | 1.00604868 | 1.00371146 | 1.00770307 |
| S | 54 | 82 | 54 |
| C | 76 | 115 | 74 |
| **Matriz** | **Objetivo** | **Métrica** | **Algoritmos** | | |
| **BFS** | **DFS** | **A\*** |
| **Matriz5045** | 7,37 | H | - | - | 44 |
| P | 1.00947809 | 1.00332689 | 1.00931311 |
| S | 54 | 64 | 54 |
| C | 70 | 87 | 69 |
| 20,20 | H | - | - | 40 |
| P | 1.00821733 | 1.00700545 | 1.00649691 |
| S | 52 | 80 | 52 |
| C | 73 | 116 | 70 |
| 28,45 | H | - | - | 73 |
| P | 1.02075505 | 1.01224232 | 1.03987217 |
| S | 107 | 163 | 107 |
| C | 149 | 249 | 144 |

Tabela 2 - Tabela de resultados para 50% de obstáculos com um objetivo

Observando a Tabela 2, os resultados não são muito conclusivos devido ao tamanho menor das matrizes. Não foram testadas matrizes de maiores dimensões com o 45% de probabilidade de obstáculos devido ao número de ilhas inacessíveis que é criado, tendo algumas das criadas terem sofrido alterações manuais de modo a conseguirem encontrar o objeto. Um exemplo de uma matriz com 45% de probabilidade de obstáculos de tamanho 100x100 está disponível na Figura 11:

Uma imagem com padrão, arte, esboço, desenho

Descrição gerada automaticamente

Figura 11 - Representação visual da matriz 10045

Como se pode observar, escolhendo um ponto mais distante do ponto inicial definido por (0, 0), está praticamente tudo envolvido de paredes, sendo impossível de chegar às coordenadas do objeto.

Comparando os dois resultados com a matriz de dimensão 50, com o maior número de obstáculos o algoritmo de pesquisa em profundidade acompanha mais os outros dois algoritmos. Com o aumento de obstáculos, o DFS não tem tanta liberdade de ir de uma ponta a outra da matriz, sendo mais restringido diminuindo o número de passos quando o objeto se encontra mais próximo do início, como se observa na primeira tabela, por exemplo, comparando na matriz de dimensão 100 com o objetivo a (7, 15), o DFS percorre um total de 2006 nós do inicio até ao objeto, enquanto que os outros dois modelos encontram o caminho igual à heurística, tendo o A\* o menor custo total.

Com o aumento das paredes, o DFS torna-se mais eficiente e retorna caminhos melhores do que com o reduzido número de paredes.

## Resultados com dois objetivos

Para os dois algoritmos não informados, estes deverão começar a procura e retornar um caminho que contenha os dois objetivos definidos anteriormente, não tendo uma ordem específica.

Os algoritmos começariam no início e após encontrar o primeiro objetivo, voltava a correr o algoritmo, desta vez do primeiro objetivo que encontre até ao que falta guardando os nós visitados para que não se perca eficiência em pesquisar nós já percorridos.

Já o A\*, começaria pelo objetivo com a menor heurística e correria o algoritmo duas vezes, uma no início e outra no primeiro objetivo, quando o encontrar, retornando o caminho com os dois objetivos da mesma forma.

# Conclusão

Após os testes e a comparação dos resultados, chega-se à conclusão de que cada algoritmo tem as suas vantagens, como o algoritmo em profundidade encontrar um caminho mais rapidamente se tiver distante, e as suas desvantagens, como este mesmo algoritmo encontrar caminhos mais longos, em comparação com o A\*, que pode demorar mais tempo a encontrar um caminho mas este ser o mais barato em quesito de tempo pelo custo real e o em largura, que encontrará sempre o menor caminho em passos, mas mais caro que o encontrado pelo A\*.

A aplicação de um ou outro algoritmo dependerá do ambiente em que é inserido, do objetivo deste e o que será priorizado por quem o irá implementar.

A realização deste trabalho permitiu desenvolver competências em Python não adquiridas noutras unidades curriculares e um conhecimento mais profundo sobre algoritmos de pesquisa.

## Ferramentas utilizadas

***Version Control***

* GitHub

Repositório de versões.

<https://github.com/xaloftal/IA>

**Planeamento**

* Trello

Ferramenta de organização e planeamento de projetos

**Implementação**

* Visual Studio Code - VSCode

IDE de programação

**Comunicação**

* Discord

Rede social de comunicação por voz

Utilizado para as reuniões do grupo

* Whatsapp

Rede social de mensagens instantâneas

Utilizado para a comunicação mais informal

# Bibliografia

https://www.mecalux.pt/blog/automatizacao-armazens

<https://www.geeksforgeeks.org/depth-first-traversal-dfs-on-a-2d-array/>

<https://www.baeldung.com/cs/dfs-vs-bfs-vs-dijkstra>

<https://www.pygame.org/tags/algorithm>

https://www.pygame.org/docs/ref/time.html

<https://cs50.harvard.edu/ai/2023/notes/0/>

https://www.techwithtim.net/tutorials/game-development-with-python/snake-pygame/snake-tutorial-1

https://www.inf.ed.ac.uk/teaching/courses/inf2d/timetable/05\_SearchB\_wbg\_4up.pdf

https://en.wikipedia.org/wiki/Breadth-first\_search