Relatório ALGAV SPRINT B

2 de December

1211089 – José Gouveia

1211128 – Tiago Oliveira

1211131 – Pedro Pereira

1211151 – Alexandre Geração

isep Instituto Superior de **Engenharia** do Porto

US510

Como gestor de tarefas pretendo encontrar caminhos entre edifícios que tentem otimizar um dado critério

Este predicado vai escolher o caminho que envolva a menor utilização de elevadores e no caso de igualdade vai escolher o que tiver uma menor utilização dos corredores.

```
melhor_caminho_pisos(PisoOr,PisoDest,LLigMelhor,LPsCam):-
    findall(LLig,caminho_pisos(PisoOr,PisoDest,_,LLig,LPsCam),LLLig),
    menos_elevadores(LLLig,LLigMelhor,_,_),
    extract_floor_sequence(LLigMelhor,LPsCam).
```

Este predicado compara o número de elevadores e corredores numa lista de ligações com outra lista de ligações e retorna a lista com menor numero de elevadores e corredores.

```
menos_elevadores([LLig],LLig,NElev,NCor):-
    conta(LLig,NElev,NCor).

menos_elevadores([LLig|OutrosLLig],LLigR,NElevR,NCorR):-
    menos_elevadores(OutrosLLig,LLigM,NElev,NCor),
    conta(LLig,NElev1,NCor1),
    (((NElev1<NElev;(NElev1==NElev,NCor1<NCor)),!,
        NElevR is NElev1, NCorR is NCor1,LLigR=LLig);
        (NElevR is NElev,NCorR is NCor,LLigR=LLigM)).
```

Estes predicados contam o número de elevadores e de corredores numa lista de ligações.

conta([],0,0).

conta([elev(_,_)|L],NElev,NCor):- conta(L,NElevL,NCor),NElev is NElevL+1.

conta([cor(_,_)|L],NElev,NCor):-conta(L,NElev,NCorL),NCor is NCorL+1.

Finalmente estes predicados extraem a sequência de pisos de uma lista de ligações.

extract_floor_sequence([], []).

extract_floor_sequence([elev(From, To) | Rest], [From, To | Result]) :-

extract_floor_sequence(Rest, Result).

extract_floor_sequence([cor(_, To) | Rest], [To | Result]) :-

extract_floor_sequence(Rest, Result).

US700

Como arquiteto da solução pretendo que a informação sobre robots seja partilhada entre os módulos de Administração de dados, Planeamento e visualização

Para a resolução desta US foi criado um predicado dinâmico para criar os factos sobre robots.

:-dynamic robot/3. % robot(Code, Type, OperationStatus(true or false))...

Para ser possível buscar a informação ao módulo MasterDataBuilding (MDB) foi criado o predicado get_robots() que inicialmente irá apagar todos os factos de robots que estiverem no módulo de planeamento. De seguida vai fazer um pedido HTTP ao URL definido, neste caso a API do MDB relacionada a robots e vai buscar todos os presentes na base de dados. Depois é feita a tradução da resposta HTTP JSON para Prolog e, por fim, é chamado o predicado create_robot(ResObj) que vai tratar os dados recebidos.

get_robots():-

delete_robots(),

http_open('http://localhost:4000/api/robots/listAll', ResJSON, [cert_verify_hook(cert_accept_any)]),

json_read_dict(ResJSON, ResObj),

create_robot(ResObj).

O predicado create_robot() vai então pegar nos dados recebidos e iterar por cada robot, criando factos robot/3 com a ajuda do assertz, guardando o seu código, tipo e estado de operação.

```
create_robot([]).

create_robot([Robot|T]):-

    assertz(robot(Robot.code, Robot.type, Robot.operationStatus)),

    create_robot(T).
```

O predicado delete robot faz uso do retractall/1 para apagar todos os factos do tipo robot/3.

```
delete_robots():-

    retractall(robot(_,_,_)).
```

Passando agora para o tipo de robot, foi criado o seguinte predicado dinâmico para criar os factos sobre os tipos de robot.

```
:-dynamic robotType/3. % robotType(TypeID, AvailableTasks[])...
```

Para ser possível buscar a informação ao módulo MasterDataBuilding (MDB) foi criado o predicado get_robots types() que inicialmente irá apagar todos os factos de tipos de robots que estiverem no módulo de planeamento. De seguida vai fazer um pedido HTTP ao URL definido, neste caso a API do MDB relacionada a tipos de robots e vai buscar todos os presentes na base de dados. Depois é feita a tradução da resposta HTTP JSON para Prolog e, por fim, é chamado o predicado create_robotType(ResObj) que vai tratar os dados recebidos.

```
get_robot_types():-

    delete_robotType(),

    http_open('http://localhost:4000/api/robotTypes/listAllRobotTypes', ResJSON, [cert_verify_hook(cert_accept_any)]),

    json_read_dict(ResJSON, ResObj),

    create_robotType(ResObj).
```

O predicado create_robotType() vai então pegar nos dados recebidos e iterar por cada tipo de robot, criando factos robotType/3 com a ajuda do assertz, guardando o seu código, tipo e estado de operação.

```
create_robotType([]).

create_robotType([RobotType|T]):-

  assertz(robotType(RobotType.robotTypeID, RobotType.availableTasks)),

  create_robotType(T).
```

O predicado delete robot type faz uso do retractall/1 para apagar todos os factos do tipo robotType/2.

```
delete_robotType():-

  retractall(robotType(_,_)).
```

US710

Como arquiteto da solução pretendo que a informação sobre edifícios, pisos, passagens, elevadores e mapas seja partilhada entre os módulos de Administração de dados, Planeamento e visualização

Para a resolução desta US é necessário criar um predicado dinâmico para criar os factos sobre os vários elementos de um edifício.

Começando com o building, temos o seguinte predicado dinâmico.

```
:-dynamic building/1. % building(A)...
```

Para ser possível buscar a informação ao módulo MasterDataBuilding (MDB) foi criado o predicado get_buildings() que inicialmente irá apagar todos os factos de buildings que estiverem no módulo de planeamento. De seguida vai fazer um pedido HTTP ao URL definido, neste caso a API do MDB relacionada a buildings e vai buscar todos os presentes na base de dados. Depois é feita a tradução da resposta HTTP JSON para Prolog e, por fim, é chamado o predicado create_building(ResVal) que vai tratar os dados recebidos.

```
get_buildings():-

  delete_buildings(),

  http_open('http://localhost:4000/api/buildings/listAllBuildings', ResJSON,
[cert_verify_hook(cert_accept_any)]),

  json_read_dict(ResJSON, ResObj),
```

create_building(ResVal).

O predicado create_building() vai então pegar nos dados recebidos e iterar por cada building, criando factos building/3 com a ajuda do assertz, guardando o seu código, tipo e estado de operação.

create_building([]).

create_building([Building|T]):-

assertz(building(Building.buildingCode)),

create_building(T).

O predicado delete building faz uso do retractall/1 para apagar todos os factos do tipo building/1.

delete_buildings():-

retractall(building(_)).

O predicado dinâmico dos floors, é o seguinte.

:-dynamic floor/2. % floor(A, 1)...

O predicado get_floors/0 foi desenvolvido para facilitar a busca de informações no módulo MasterDataBuilding (MDB). Inicialmente, realiza a limpeza de todos os fatos relacionados a andares presentes no módulo de planejamento. Em seguida, faz uma requisição HTTP para a API do MDB, obtendo todos os prédios na base de dados. Processa cada prédio individualmente, chamando o predicado process_buildings/1. Utiliza o corte (!) para evitar backtracking e otimizar a execução.

get_floors() :-

delete_floors(),

findall(X, building(X), Buildings),

process_buildings(Buildings),

!.

O predicado process_buildings/1 serve como base de recursão para a lista de prédios vazia. Para cada prédio em [Building|RestBuildings], obtém informações sobre os andares chamando o predicado get_building_floors/1.

process_buildings([]).

process_buildings([Building|RestBuildings]) :-

get_building_floors(Building),

process_buildings(RestBuildings).

O predicado get_building_floors/1 realiza uma requisição HTTP específica para obter informações sobre os andares do prédio especificado. Em seguida, chama handle_http_status/3 para processar a resposta.

get_building_floors(Building):-

atom_concat('http://localhost:4000/api/floors/listAllFloors/', Building, Url),

http_open(Url, ResJSON, [cert_verify_hook(cert_accept_any), status_code(Status)]),

handle_http_status(Status, ResJSON, Building).

O predicado handle_http_status/3 processa a resposta HTTP bem-sucedida com código 200, lendo o JSON e chamando o predicado create_floor/2 para tratar os dados recebidos. É também utilizado para lidar com outros códigos de status HTTP.

handle_http_status(200, ResJSON, Building) :-

json_read_dict(ResJSON, ResObj),

create_floor(Building, ResObj).

O predicado create_floor/2 serve como base de recursão para a lista de andares vazia. Adiciona fatos representando andares associados ao prédio com create_floor(Building, [Floor|T]).

handle_http_status(_, _, _).

O predicado create_floor(_, []) serve como base de recursão para a lista de andares vazia. Adiciona fatos representando andares associados ao prédio com create_floor(Building, [Floor|T]).

create_floor(_, []).

create_floor(Building, [Floor|T]):-

assertz(floor(Building, Floor.floorId)),

create_floor(Building, T).

O predicado create_floor/2 serve como base de recursão para a lista de andares vazia. Adiciona factos representando andares associados ao edifício com create_floor(Building, [Floor|T]).

create_floor_map(_, []).

create_floor_map([Floor|T]):-

assertz(floorMap(Floor.floorId, Floor.floorMap)),

create_floor_map(Building, T).

O predicado delete_floors faz uso do retractall/1 para apagar todos os factos do tipo floor/2.

delete_floors():-

retractall(floor(_,_)).

Seguindo agora para o predicado dinâmico de passageways.

:-dynamic passageway/5. % passageway(1, A, 2, B, 2)...

O predicado get_passageways/0 realiza as seguintes etapas: primeiro, remove todas as passagens existentes; em seguida, faz uma requisição HTTP à API para obter todas as passagens disponíveis. Posteriormente, converte a resposta JSON para um objeto Prolog e chama o predicado process_passageways/1 para processar as passagens obtidas.

get_passageways():-

delete_passageways(),

http_open('http://localhost:4000/api/passageways/listAll', ResJSON, [cert_verify_hook(cert_accept_any)]),

json_read_dict(ResJSON, ResObj),

process_passageways(ResObj).

O predicado findBuildingByFloorId/2 encontra o edifício associado a um determinado identificador de andar (FloorId). Usa a relação floor/2 para encontrar o edifício correspondente.

findBuildingByFloorId(FloorId, Building):-

floor(Building, FloorId).

O predicado process_passageways/1 serve como base de recursão para a lista de passagens vazia. Para cada passagem em [Passageway|T], encontra os edifícios associados aos andares de origem e destino, chamando findBuildingByFloorId/2. Em seguida, chama o predicado create_passageway/3 para criar uma passagem e processa as passagens restantes de forma recursiva.

process_passageways([]).

process_passageways([Passageway|T]) :-

findBuildingByFloorId(Passageway.floor1Id, Building1),

findBuildingByFloorId(Passageway.floor2Id, Building2),

create_passageway(Passageway, Building1, Building2),

process_passageways(T).

O predicado create_passageway/3 adiciona um facto representando uma passagem, com informações como o identificador da passagem (Passageway.passagewayId), os edifícios de origem e destino, bem como os andares associados.

create_passageway(Passageway, Building1, Building2):-

assertz(passageway(Passageway.passagewayId, Building1, Passageway.floor1Id, Building2, Passageway.floor2Id)).

O predicado delete_passageway faz uso do retractall/1 para apagar todos os factos do tipo passageway/5.

delete_passageways():-

retractall(passageway(_,_,_,_,_)).

O predicado dinâmico do elevator é o seguinte.

:-dynamic elevator/3. % elevator(1, A, [1,2,3])...

Estes predicados irão criar os factos relacionados com os elevadores, começando por eliminar os factos dos elevadores e realizando um pedido HTTP para obter os elevadores.

get_elevators():-

delete_elevators(),

http_open('http://localhost:4000/api/elevators/listAll', ResJSON, [cert_verify_hook(cert_accept_any)]),

json_read_dict(ResJSON, ResObj),

create_elevator(ResObj).

Este predicado vai criar os factos dos elevadores.

create_elevator([]).

create_elevator([Elevator|T]):-

assertz(elevator(Elevator.elevatorId, Elevator.buildingCode, Elevator.floorsId)),

create_elevator(T).

O predicado delete_elevators faz uso do retractall/1 para apagar todos os factos do tipo elevators/5.

```
delete_elevators():-
   retractall(elevators(_,_,_,_,_)).
```

Finalmente o predicado dinâmico do room é o seguinte.

```
:-dynamic room/3. % room(Name, A, 1)...
```

Estes predicados irão criar os factos referentes aos rooms. O predicado principal get_rooms() apaga todos rooms existentes na base de conhecimentos e depois encontra todos os edifícios para posteriormente retirar os rooms de cada edifício.

```
get_rooms():-
   delete_rooms(),
   findall(X, building(X), Buildings),
   process_buildings2(Buildings),
   !.
```

Este predicado irá retirar os rooms de cada building.

```
process_buildings2([]).
process_buildings2([Building|RestBuildings]) :-
   get_building_rooms(Building),
   process_buildings2(RestBuildings).
```

Este predicado faz uma chamada HTTP para obter todos os rooms de uma building, que depois são passados ao handle_http_status1.

```
get_building_rooms(Building):-
   atom_concat('http://localhost:4000/api/rooms/listAllInBuilding/', Building, Url),
   http_open(Url, ResJSON, [cert_verify_hook(cert_accept_any), status_code(Status)]),
   handle_http_status1(Status, ResJSON, Building).
```

Este predicado com a lista dos buildings e cria a base de conhecimento com os rooms associando os aos buildings e floors.

```
handle_http_status1(200, ResJSON, Building) :-
```

json_read_dict(ResJSON, ResObj),

create_room(Building, ResObj).

handle_http_status1(_, _, _).

Este predicado irá criar factos associando os rooms com os seus respetivos building e floor.

create_room(_, []).

create_room(Building, [Room|T]):-

assertz(room(Room.roomName, Building, Room.floorId)),

create_room(Building, T).

O predicado delete_rooms faz uso do retractall/1 para apagar todos os factos do tipo room/3.

delete_rooms():-

retractall(room(_,_,_)).

US730

Como arquiteto da solução pretendo que a informação sobre percursos entre edifícios seja partilhada entre os módulos de Planeamento, SPA e Visualização

US 515

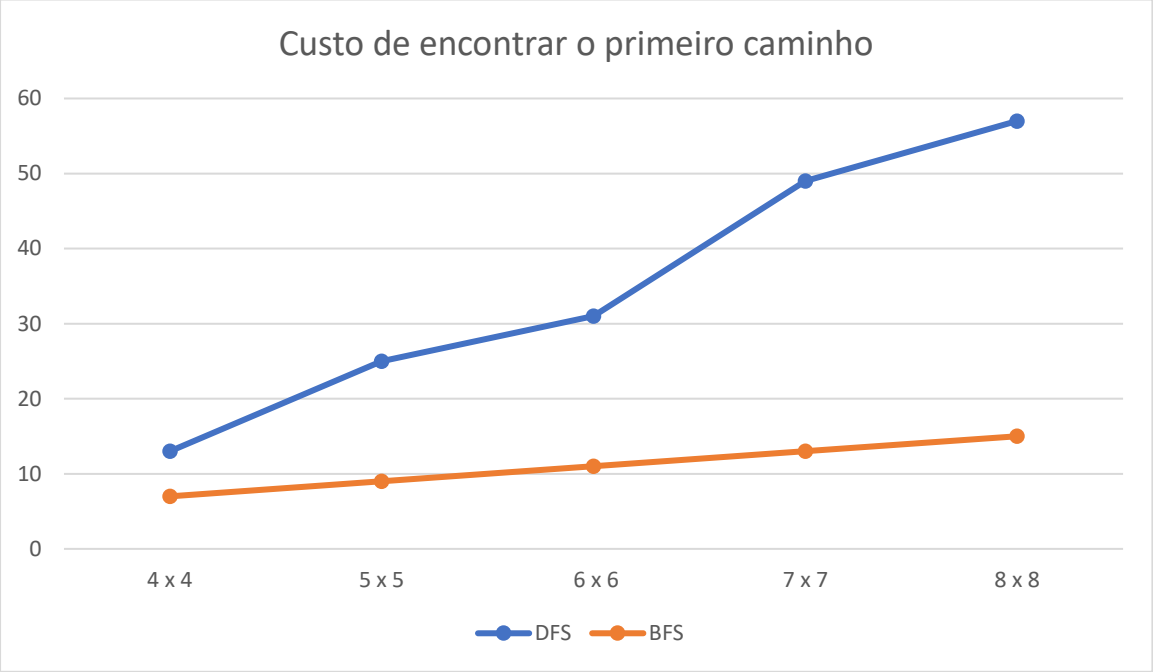
Análise da complexidade da pesquisa em grafos obtidos das ligações entre células retangulares de posicionamento dos robots

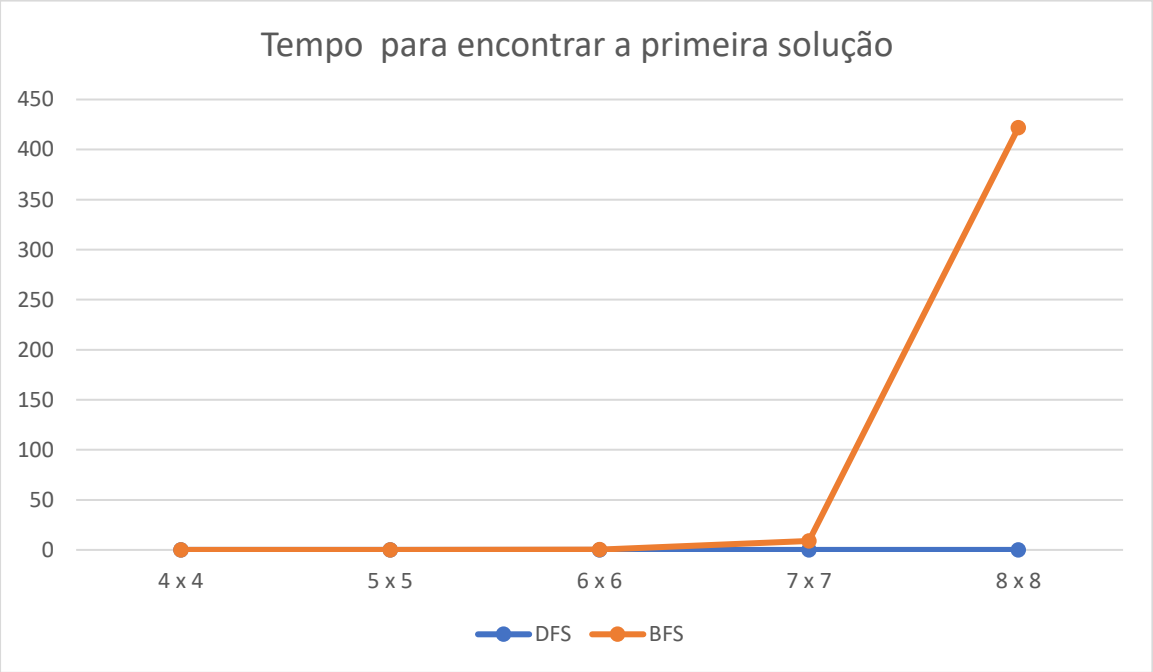
NColxLin	Nº nós	Nº lig	Solução 1º prof	Custo e tempo (em segundos)	Solução 1º larg	Custo e tempo (em segundos)
4 x 4	16	84	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(4, 2), cel(3, 2), cel(2, 2), cel(1, 2), cel(1, 3), cel(2, 3), cel(3, 3), cel(4, 3), cel(4, 4)	Custo=13 Tempo=2.522e-5	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(4, 2), cel(4, 3), cel(4, 4)	Custo=7 Tempo=0.0004675
5 x 5	25	144	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(5, 2), cel(4, 2), cel(3, 2), cel(2, 2), cel(1, 2), cel(1, 3), cel(2, 3), cel(3, 3), cel(4, 3), cel(5, 3), cel(5, 4), cel(4, 4), cel(3, 4), cel(2, 4), cel(1, 4), cel(1, 5), cel(2, 5), cel(3, 5), cel(4, 5), cel(5, 5)	Custo=25 Tempo=4.019e-5	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(5, 2), cel(5, 3), cel(5, 4), cel(5, 5)	Custo=9 Tempo=0.007769
6 x 6	36	220	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(6, 1), cel(6, 2), cel(5, 2), cel(4, 2), cel(3, 2), cel(2, 2), cel(1, 2), cel(1, 3), cel(2, 3), cel(3, 3), cel(4, 3), cel(5, 3), cel(6, 3), cel(6, 4), cel(5, 4), cel(4, 4), cel(3, 4), cel(2, 4), cel(1, 4), cel(1, 5), cel(2, 5), cel(3, 5), cel(4, 5), cel(5, 5), cel(6, 5), cel(6, 6)	Custo=31 Tempo=6.2892e-5	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(6, 1), cel(6, 2), cel(6, 3), cel(6, 4), cel(6, 5), cel(6, 6)	Custo=11 Tempo=0.2253
7 x 7	49	312	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(6, 1),	Custo=49 Tempo=0.0001262	cel(1, 1), cel(2, 1), cel(3, 1),	Custo=13 Tempo=0.0653

			cel(7, 1), cel(7, 2), cel(6, 2), cel(5, 2), cel(4, 2), cel(3, 2), cel(2, 2), cel(1, 2), cel(1, 3), cel(2, 3), cel(3, 3), cel(4, 3), cel(5, 3), cel(6, 3), cel(7, 3), cel(7, 4), cel(6, 4), cel(5, 4), cel(4, 4), cel(3, 4), cel(2, 4), cel(1, 4), cel(1, 5), cel(2, 5), cel(3, 5), cel(4, 5), cel(5, 5), cel(6, 5), cel(7, 5), cel(7, 6), cel(6, 6), cel(5, 6), cel(4, 6), cel(3, 6), cel(2, 6), cel(1, 6), cel(1, 7), cel(2, 7), cel(3, 7), cel(4, 7),cel(5, 7), cel(6, 7), cel(7, 7)		cel(4, 1), cel(5, 1), cel(6, 1), cel(7, 1), cel(7, 2), cel(7, 3), cel(7, 4), cel(7, 5), cel(7, 6), cel(7, 7)	
8 x 8	64	420	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(6, 1), cel(7, 1), cel(8, 1), cel(8, 2), cel(7, 2), cel(6, 2), cel(5, 2), cel(4, 2), cel(3, 2), cel(2, 2), cel(1, 2), cel(1, 3), cel(2, 3), cel(3, 3), cel(4, 3), cel(5, 3), cel(6, 3), cel(7, 3), cel(8, 3), cel(8, 4), cel(7, 4), cel(6, 4), cel(5, 4), cel(4, 4), cel(3, 4), cel(2, 4), cel(1, 4), cel(1, 5), cel(2, 5), cel(3, 5), cel(4, 5), cel(5, 5), cel(6, 5), cel(7, 5), cel(8, 5), cel(8, 6), cel(7, 6), cel(6, 6), cel(5, 6), cel(4, 6), cel(3, 6), cel(2, 6), cel(1, 6), cel(1, 7),	Custo=57 Tempo=0.000102	cel(1, 1), cel(2, 1), cel(3, 1), cel(4, 1), cel(5, 1), cel(6, 1), cel(7, 1), cel(8, 1), cel(8, 2), cel(8, 3), cel(8, 4), cel(8, 5), cel(8, 6), cel(8, 7), cel(8, 8)	Custo=15 Tempo=421,763

			cel(2, 7), cel(3, 7), cel(4, 7), cel(5, 7), cel(6, 7), cel(7, 7), cel(8, 7), cel(8, 8)			

BFS vs DFS





Conclusões:

- No algoritmo DFS é possível verificar que à medida que o tamanho da matriz aumenta, o custo para obter a solução é superior, mas o tempo da solução mantem se sempre baixo. Este é um algoritmo preferível para usar em matrizes de maior dimensão.
- No algoritmo BFS, o custo da solução é sempre menor que no algoritmo DFS, no entanto enquanto para matrizes de tamanho menor, o tempo pode ser considerado igual ao DFS, com matrizes de maior dimensão este aumenta exponencialmente, fazendo com que este seja apenas benéfico para matrizes de menor dimensão.