Relatório de ALGAV

SEM5PI_Sprint2

Turma 3DJ Grupo 058
1211396 João Batista
1211415 David Dias
1211417 Ezequiel Estima
1211469 Marco Andrade

Data: 26/11/2023

Índice

Parte I - Representação do conhecimento do domínio

Para esta solução os principais conceitos de domínio a representar são edifícios, pisos, elevadores, passagens, salas e o mapa de cada piso, ou seja, as células por onde o robot pode ou não andar.

Edifícios e pisos

A representação dos edifícios e os seus pisos é feita por um predicado do tipo *piso(Edificio,[piso1,piso2,(…)])* onde primeiro argumento é o edifício e o segundo é uma lista dos pisos que ele contém.

Também é utilizado o predicado *dim_ed(edifício,X,Y)* que representa a dimensão (nº de colunas e de linhas) do edifício.

Elevadores

A representação dos elevadores é do tipo *elevador(Edificio,[Piso1,Piso2,(…)])*. O primeiro argumento é o edifício onde o elevador está colocado e o segundo é a lista de pisos do edifício que o elevador serve.

Para representar as coordenadas do elevador (coluna e linha), é utilizado o predicado *coordElevador(Edificio,X,Y)*.

Passagens

Uma passagem é representada por *corredor(Edificio1,Edificio2,Piso1,Piso2)* que indica que par edifícios e pisos estão ligados por uma passagem.

Para as coordenadas, é utilizado *coordCorredor(PisoA,PisoB,xA1,yA1,xA2,yA2,xB1,yB1,xB2,yB2)* que representa os dois pontos de cada piso que correspondem à entrada para a passagem.

Salas

Para representar as salas é utilizado o predicado *salas(Piso,[Sala1,Sala2,(…)])* que indica as salas por cada piso.

Para a posição apenas é necessário as coordenadas da porta, indicadas por um predicado do tipo *coordPorta(Sala,X,Y)*

Mapa de um piso

O mapa de um piso é representado por predicados do tipo *m(piso,X,Y,O)*, equivalentes a cada célula da matriz do piso e que referem (no último parâmetro) se essa célula está ocupada ou não, isto é, se o robot consegue passar pela célula ou não (0 no caso de estar livre, 1 no caso de estar ocupada). Células de portas, elevadores e passagens estão todas marcadas com 0.

Parte II - Obtenção da solução ótima para o Planeamento de movimentação entre pisos de edifícios

A primeira parte da solução pretende calcular caminhos entre os pisos de edifícios, isto é, quais pisos o robot deve percorrer, recorrendo a elevadores e corredores, para ir de um *PisoOrig* a um *PisoDest*

Para isso são utilizados os predicados caminho_edificios/3 e caminho_pisos/3 apresentados a seguir.

```
%%%%%%%%%%%%%%%%%%%%%
caminho_edificios(EdOr,EdDest,LEdCam):-
   caminho_edificios2(EdOr,EdDest,[EdOr],LEdCam).


caminho_edificios2(EdX,EdX,LEdInv,LEdCam):-
   !,
   reverse(LEdInv,LEdCam).

caminho_edificios2(EdAct,EdDest,LEdPassou,LEdCam):-
   (liga(EdAct,EdInt);liga(EdInt,EdAct)),
   \+member(EdInt,LEdPassou),
   caminho_edificios2(EdInt,EdDest,[EdInt|LEdPassou],LEdCam).
%%%%
caminho_pisos(PisoOr,PisoDest,LEdCam,LLig):-
        pisos(EdOr,LPisosOr),
        member(PisoOr,LPisosOr),
        pisos(EdDest,LPisosDest),
        member(PisoDest,LPisosDest),
        caminho_edificios(EdOr,EdDest,LEdCam),
        segue_pisos(PisoOr,PisoDest,LEdCam,LLig).
%%%%%%%%%%%%%%%%%%%%%
```

O predicado *caminho_edificios/3* utiliza o predicado *liga/2* para criar uma lista de edifícios por onde o robot vai passar, informação que depois é complementada pelo *caminho_piso/3* que gera uma lista com elementos do tipo *elev(Piso1,Piso2)* e *cor(Piso1,Piso2)* indicando quais pisos e de que maneira o robot deve transitar entre estes, para que possa chegar ao destino pretendido.

Parte III - Movimentação do robot dentro de um piso de edifício

Para calcular o percurso do robot dentro do piso foram implementados os métodos Primeiro em Profundidade (DFS), Primeiro em Profundidade com a escolha da melhor solução (Better DFS), Primeiro em Largura (BFS), e A*.

Todos os algoritmos funcionam para grafos, por isso foi necessário adaptar a matriz do piso para uma estrutura de dados deste tipo.

BFS, Better DFS e DFS

Para estes algoritmos, foi utilizado o seguinte predicado o:

```
cria_grafo(_,0):-!.
cria_grafo(Col,Lin):-cria_grafo_lin(Col,Lin),Lin1 is Lin-1,cria_grafo(Col,Lin1).

cria_grafo_lin(0,_):-!.
cria_grafo_lin(Col,Lin):-m(Col,Lin,0),!,ColS is Col+1, ColA is Col-1, LinS is Lin+1,LinA is Lin-1,
   ((m(ColS,Lin,0),assertz(ligacel(cel(Col,Lin),cel(ColS,Lin)));true)),
   ((m(ColA,Lin,0),assertz(ligacel(cel(Col,Lin), cel(ColA,Lin)));true)),
   ((m(Col,LinS,0),assertz(ligacel(cel(Col,Lin), cel(Col,LinS)));true)),
   ((m(Col,LinA,0),assertz(ligacel(cel(Col,Lin), cel(Col,LinA)));true)),
   ((m(ColA,LinA,0),assertz(ligacel(cel(Col,Lin), cel(ColA,LinA)));true)),
   ((m(ColS,LinS,0),assertz(ligacel(cel(Col,Lin), cel(ColS,LinS)));true)),
   ((m(ColA,LinS,0),assertz(ligacel(cel(Col,Lin), cel(ColA,LinS)));true)),
   ((m(ColS,LinA,0),assertz(ligacel(cel(Col,Lin), cel(ColS,LinA)));true)),
   Col1 is Col-1,
   cria_grafo_lin(Col1,Lin).
cria_grafo_lin(Col,Lin):-Col1 is Col-1,cria_grafo_lin(Col1,Lin).
```

Este predicado permite a criação de *ligacel/2* que corresponde ao conceito de edge do grafo, permitindo executar os algoritmos e obter os caminhos desejados.

A*

Para o A*, é necessário fazer um tratamento diferente. É necessário criar os factos *node(Id,X,Y)* e *edge(Node1,Node2,Custo)*.

Para a criação dos nodes foi implementado o seguinte predicado:

```
criar_nodes():-
   findall(m(X,Y,0),m(X,Y,0),Res),
   criar_nodes1(Res,0).

criar_nodes1([],_).
criar_nodes1([m(X,Y,0)|RL],Id):-
   Id1 is Id+1,
   assertz(node(Id1,X,Y)),
   criar_nodes1(RL,Id1).
```

Primeiro são procuradas todas as células do piso que não estão ocupadas (*m(X,Y,0)*) e, para cada uma, é criado um node com um id gerado automaticamente e com a posição igual à da célula.

Para a criação das edges:

```
%%%% Criar edges auxiliares %%%%
cria_edges_Aux(_,0):-!.
cria_edges_Aux(Col,Lin):-cria_grafo_lin1(Col,Lin),Lin1 is Lin-1,cria_edges_Aux(Col,Lin1).

cria_grafo_lin1(0,_):-!.
cria_grafo_lin1(Col,Lin):-m(Col,Lin,0),!,ColS is Col+1, ColA is Col-1, LinS is Lin+1,LinA is Lin-1, % Se não for
parede ver os nodes à volta
   ((m(ColS,Lin,0),assertz(edge(Col,Lin,ColS,Lin,1));true)),
   ((m(ColA,Lin,0),assertz(edge(Col,Lin,ColA,Lin,1));true)),
   ((m(Col,LinS,0),assertz(edge(Col,Lin,Col,LinS,1));true)),
   ((m(Col,LinA,0),assertz(edge(Col,Lin,Col,LinA,1));true)),
   ((m(ColA,LinA,0),assertz(edge(Col,Lin,ColA,LinA,sqrt(2)));true)),
   ((m(ColS,LinS,0),assertz(edge(Col,Lin,ColS,LinS,sqrt(2)));true)),
   ((m(ColA,LinS,0),assertz(edge(Col,Lin,ColA,LinS,sqrt(2)));true)),
```

```
   ((m(ColS,LinA,0),assertz(edge(Col,Lin,ColS,LinA,sqrt(2)));true)),
   Col1 is Col-1,
   cria_grafo_lin1(Col1,Lin).
cria_grafo_lin1(Col,Lin):-Col1 is Col-1,cria_grafo_lin1(Col1,Lin). %% Se o ponto for parede dá skip

%%%% Criar edges %%%%
criar_edges_Astar():-
   findall(edge(Col,Lin,ColS,LinS,Custo),edge(Col,Lin,ColS,LinS,Custo),Res),
   criar_edges1(Res).

criar_edges1([]).
criar_edges1([edge(Col,Lin,ColS,LinS,Custo)|RL]):-
   node(Id1,Col,Lin),
   node(Id2,ColS,LinS),
   assertz(edge(Id1,Id2,Custo)),
   criar_edges1(RL).
```

Primeiro são criadas edges auxiliares, com a posição do node em vez do seu respetivo id. Para cada célula não ocupada, são verificadas todas as células vizinhas (adjacentes e na diagonal) e, caso estas também estejam desocupadas, é criada uma edge auxiliar.

Depois de todas as edges auxiliares estarem criadas, são criadas as edges que utilizam os ids dos nodes.

Resultado

Para calcular o caminho basta executar os predicados correspondentes.

-> dfs(Orig,Dest,Path)

->all_dfs(Orig,Dest,LPath)

->bfs(Orig,Dest,Path)

->aStar(Orig,Dest,Cam,Custo)

Para o DFS, Better DFS e BFS, Orig e Dest são factos do tipo *cel(X,Y)* e Path o caminho obtido e para o A* Orig e Dest são ids dos nodes de origem e destino, respetivamente, Cam é a lista de nodes do caminho e Custo é a distância percorrida.

Parte IV - Integração do ponto 2 com o ponto 3

Para integrar os resultados do ponto 2 com o ponto 3 são utilizados 2 predicados: *processar_LLig(LLig, LCelCam)* (1) e *processar_LCelCam(LCelCam,Res)* (2)

(1) processar_LLig(LLig, LCelCam)

```
%%%%%%%%
processar_LLig([],[]).

processar_LLig([cor(Piso1,Piso2)|LLig],[cel1(Piso1,X1,Y1),cel1(Piso2,X2,Y2)|LCelCam]):-
   (coordCorredor(Piso1,Piso2,X1,Y1,_,_,X2,Y2,_,_);coordCorredor(Piso2,Piso1,X2,Y2,_,_,X1,Y1,_,_)),
   processar_LLig(LLig,LCelCam).

processar_LLig([elev(Piso1,Piso2)|LLig],[cel1(Piso1,X,Y),cel1(Piso2,X,Y)|LCelCam]):-
   elevador(Ed,LEd),
   member(Piso1, LEd),
   coordElevador(Ed,X,Y),
   processar_LLig(LLig,LCelCam).
%%%%%%%%
```

Este predicado permite transformar a lista obtida no predicado caminho_pisos/4 (Lista com elementos do tipo *elev(Piso1,Piso2)* e/ou *cor(Piso1,Piso2))* numa lista com elementos do tipo *cel1(Piso1,X1,Y1)*. Para cada elemento de LLig, são adicionados 2 elementos à nova lista que indicam a posição do elevador ou passagem no piso correspondente.

Após a execução, vai ser inserido no início da lista o elemento *cel1(PisoOrig,XOrig,YOrig)* (representa o ponto inicial do caminho) e no fim da lista o elemento *cel1(PisoDest,XDest,YDest)* (representa o ponto final do caminho).

(2) processar_LCelCam(LCelCam,Res)

```
%%%%%%%%%%%
processar_LCelCam([],[]).
processar_LCelCam([cel1(Piso,X1,Y1),cel1(Piso,X2,Y2)|Rl],[cam(Piso,X1,Y1,X2,Y2)|RLCelCamPisos]):-
   processar_LCelCam(Rl,RLCelCamPisos).
%%%%%%%%%%%
```

Este predicado permite transformar a lista obtida anteriormente numa lista com elementos do tipo *cam(Piso1,X1,Y1,X2,3)*. Para cada par elemento de LLig, é adicionado um novo elemento à nova lista, que representa um caminho a realizar num determinado piso.

A partir deste processamento, utilizamos o algoritmo A* para calcular o caminho para cada piso, construindo dinamicamente um grafo para cada piso e determinando o caminho para cada um deles.

À medida que são gerados, o caminho e o custo são adicionados ao caminho e custo total para serem apresentados no final.

Parte V - Estudo da complexidade do problema

DFS, Better DFS, BFS – Matriz quadrada

Col X Lin	Total Nodes	Total Edges	DFS	Custo & Tempo	BFS	Custo & Tempo	Better DFS	Custo & Tempo
4X4	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)	12 & 0,00s	cel(1, 1),cel(2, 2), cel(3,3), cel(4, 4)]	4,24 & 0,00s	cel(1,1), cel(2,2), cel(3,3), cel(4, 4)	5,56 & 59,83s
5X5	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)]	26 & 0,00s	cel(1,1),cel(2,2), cel(3,3),cel(4,4), cel(5, 5)	5,66 & 0,03s	----------	>16,67 min
6X6	36	220	cel(1,1),cel(2,1), cel(3,1),cel(4,1), cel(5,1),cel(6,1),	32 & 0,00s	cel(1,1),cel(2,2), cel(3,3),cel(4,4), cel(5, 5),cel(6,6)	7,07 & 0,59s	----------	------

			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)					
7X7	47	312	cel(1,1),cel(2,1), cel(3,1),cel(4,1), cel(5,1),cel(6,1), 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),	50 & 0,00s	cel(1,1), cel(2,2), cel(3,3), cel(4,4), cel(5,5), cel(6,6), cel(7, 7)	8,49 & 17,07s	----------	----------

			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)					

A* – Matriz quadrada

Col X Lin	Total Nodes	Total Edges	A*	Custo & Tempo
4X4	16	84	node(1,1,1), node(6,2,2), node(11,3,3), node(16,4,4)	4,24 & 0,00001s
5X5	25	144	node(1,1,1), node(7,2,2), node(13,3,3), node(19,4,4), node(25,5,5),	5,66 & 0,00012s
6X6	36	220	node(1,1,1), node(8,2,2), node(15,3,3), node(22,4,4), node(29,5,5), node(36,6,6),	7,07 & 0,00032s
7X7	49	312	node(1,1,1), node(9,2,2), node(17,3,3), node(25,4,4), node(33,5,5), node(41,6,6),	8,48 & 0,00033s

			node(49,7,7),	

DFS, Better DFS, BFS – Corredor

Col X Lin	Total Nodes	Total Edges	DFS	Custo & Tempo	BFS	Custo & Tempo	Better DFS	Custo & Tempo
4X3	12	58	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)	11 & 0,00s	cel(1,1),cel(2,1), cel(3,2),cel(4,3)	3,83 & 0,00s	cel(1,1), cel(2,2), cel(3,3), cel(4, 3)	3,82 & 9,89s
5X3	15	76	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)	14 & 0,00s	cel(1,1),cel(2,1), cel(3,1),cel(4,2), cel(5,3)	4,83 & 0,00s	cel(1,1), cel(2,2), cel(3,1), cel(4,2), cel(5,3)	5,6 & 2,18 min
6X3	18	94	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)	17 & 0,00s	cel(1,1),cel(2,1), cel(3,1),cel(4,1), cel(5,2),cel(6,3)	5,83 & 0,00s	cel(1,1), cel(2,2), cel(3,1), cel(4,2), cel(5,3), cel(6,3)	6,65 & 6 min

7X3	21	112	cel(1,1),cel(2,1), cel(3,1),cel(4,1), cel(5,1),cel(6,1), 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)	20 & 0,00s	cel(1,1),cel(2,1), cel(3,1),cel(4,1), cel(5,1),cel(6,2), cel(7,3)	6,83 & 0,07s	----------	>15 min

A* – Corredor

Col X Lin	Total Nodes	Total Edges	A*	Custo & Tempo
4X3	12	58	node(1,1,1), node(5,2,2), node(8,3,2), node(12,4,3)	3,83 & 0,0002s
5X3	15	76	node(1,1,1), node(4,2,1), node(7,3,1), node(11,4,2), node(15,5,3)	4,83 & 0,0003s
6X3	18	94	node(1,1,1), node(4,2,1), node(7,3,1), node(10,4,1), node(14,5,2), node(18,6,3)	5,83 & 0,0010s
7X3	21	112	node(1,1,1),	6,83 & 0,0019s

			node(4,2,1), node(7,3,1), node(10,4,1), node(13,5,1), node(17,6,2) node(21,7,3)	

Resumo análise complexidade

	DFS Tempo	DFS Custo	BFS Tempo	BFS Custo	BDFS Tempo	BDFS Custo	A* Tempo	A* Custo
4X4	0,00s	12	0,00s	4,24	59,83s	5,56	4,24	0,00001s
5X5	0,00s	26	0,03s	5,66	>16,67 min	-	5,66	0,00012s
6X6	0,00s	32	0,59s	7,07	-	-	7,07	0,00032s
7X7	0,00s	50	17,07s	8,49	-	-	8,48	0,00033s
4X3	0,00s	11	0,00s	3,83	9.89s	3,83	3,83	0,0002s
5X3	0,00s	14	0,00s	4,83	2,18 min	5,6	4,83	0,0003s
6X3	0,00s	17	0,00s	5,83	6 min	6,65	5,83	0,0010s
7X3	0,00s	20	0,07s	6,83	>15 min	-	6,83	0,0019s

Através da análise da tabela, podemos concluir que:

- Algoritmo A* - é o mais eficiente, produz todas as soluções rapidamente, com os menores custos possíveis;
- Algoritmo BDFS – tempo de execução cresce rapidamente conforme o nº de inputs cresce. Este facto impossibilitou obter todos os resultados pretendidos;
- Algoritmo BFS – bom tempo de execução para grafos pouco densos. Uma das desvantagens é os requisitos de memória (tem de manter registos de todos os nodes que pode expandir);
- Algoritmo DFS – tempos de execução muito rápidos, porém apresenta soluções com muito custo.

Parte VI – Conclusão

Em suma, para desenvolver esta solução, numa primeira fase, foram criados os factos que conteriam toda a informação relativa ao campus (edifícios, pisos, elevadores, passagens, salas e mapas), depois foram desenvolvidos algoritmos de cálculo de caminhos entre edifícios e os seus respetivos pisos, e finalmente foram utilizados os algoritmos DFS, BFS e A* para obter o caminho para cada piso em específico. Também foi realizada uma análise de complexidade dos algoritmos anteriores, que permitiu concluir que o algoritmo A* é o mais eficiente entre todos os implementados, o que motivou também a sua utilização no que toca à integração das duas partes.