Relatório 1º projeto IA 2020/2021

Grupo: tg001

Alunos: João Miguel Pipa Ferreira Caldeira (93729)

João Tomás Cardoso (93730)

Descrição do Problema e da Solução

O projeto consiste em desenvolver um programa em Python que resolva o problema Ricochet Robots.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

A solução apresentada representa o tabuleiro do jogo na criação de um dicionário com as células numeradas de 1 a N^2 inseridas nas *keys* e as células adjacentes à célula da *key* inseridas nos *values* da *key* correspondente. As células estão numeradas da esquerda para a direita e em seguida de cima para baixo. O exemplo da figura 1 demonstra com clareza a numeração das células no caso de um tabuleiro de tamanho N = 4

Figura 1

.Os robôs são guardados também num dicionário indexados pela sua cor e tendo as suas coordenadas como *value*. As paredes internas do tabuleiro são processadas e são transformadas como “cortes” na adjacência entre posições e a sua representação é a remoção de uma posição adjacente a outra, impossibilitando assim o movimento de um robô entre essas posições. Após obtenção destes dados bem como o tamanho do tabuleiro, posição e cor do alvo, através do tratamento de input, avança-se para a resolução do problema.

Constrói-se o tabuleiro usando a classe *Board* e cria-se o problema Ricochet Robots através da classe *Ricochet Robots* que calcula na sua inicialização a heurística para o problema. A heurística utilizada na solução consiste na atribuição a cada célula do tabuleiro, o número mínimo de passos possíveis para chegar ao alvo considerando condições ideais para cada célula e obtendo assim uma heurística admissível para o problema. Entraremos em mais detalhe sobre a heurística utilizada na análise ao comportamento do programa nos vários testes. Formulado o problema, usa-se uma procura para encontrar a solução. A procura utilizada foi *recursive best first search,* que usa a heurística calculada. Por fim obtém-se a resposta e trata-se o output imprimindo a profundidade da solução, correspondente ao número de jogadas efetuadas, e a ação realizada por cada estado da família de nós da solução.

Avaliação Experimental dos Resultados Obtidos

Foram testadas diferentes procuras para a obtenção da solução do problema. As procuras testadas foram: procura em largura primeiro em árvore (BFS), procura em profundidade primeiro em árvore (DFS), procura gananciosa (Greedy), procura A*, procura em profundidade iterativa (IDS) e uma procura

recursive best search (RBFS). Foram realizados oito testes para comparar estas procuras, que são os oito testes da pasta *instances*, fornecida pelos docentes com inputs já feitos. Os resultados destes testes estão demonstrados nas tabelas abaixo.

Podemos observar que os resultados são mais expressivos no testes i1.txt e i8.txt, isto deve-se ao facto de estes testes terem soluções com uma profundidade superior comparativamente aos restantes testes. A primeira conclusão que podemos observar é que a DFS não conseguiu completar nenhum dos testes, pois esta procura tende a entrar num ciclo infinito não conseguindo chegar a uma solução. Uma forma de coincidência de a DFS conseguir gerar uma solução passa por alterar a ordem de formar *actions* e por "coincidência" gerar *actions* numa ordem perfeita em profundidade que corresponda a uma família de nós com uma solução, mas isto seria fazer "batota". A segunda observação efetuada é de que as procuras não informadas (DFS, BFS, IDS) gastam mais memória que as procuras informadas (Greedy,A*,RBFS), este facto evidencia-se mais quanto melhor a heurística. A BFS é a mais lenta das procuras (complexidade exponencial) e também a que gasta mais memória no entanto garante sempre uma solução, que é também uma solução ótima neste caso pois o custo por ação é 1, e é completa pois o tabuleiro é finito. A procura IDS é melhor que a BFS mantendo-se ótima pois o custo de ação é 1, e é completa, demonstrando ser a segunda procura mais rápida, é também simples de implementar e foi considerada para ser uma possível solução final. Entrando nas procuras informadas denote-se que a heurística usada influencia imenso o desempenho destas procuras, como se pode comparar com os resultados da tabela 8 (resultados obtidos com a melhor heurística), e a tabela 9 (resultados obtidos usando uma heurística primitiva mas também admissível). A procura informada com pior desempenho é a procura Greedy não só em termos de tempo como de memória, sendo uma procura que não é necessariamente completa, (pode entrar em ciclo), e não é ótima (no teste i1.txt obtém uma solução de profundidade 6 > 4(solução ótima); no teste i8.txt obtém uma solução de profundidade 7 > 4(solução ótima)). A segunda melhor opção é a procura A* que obtém tempos inferiores aos tempos da procura Greedy e também utiliza menos memória que a Greedy. É completa tem complexidade exponencial e encontra a solução ótima no entanto não é tão boa quanto a RBFS. Tanto em memória como em tempo de execução a RBFS foi a melhor procura, e portanto foi esta a procura utilizada na solução final. Seria expectável que a RBFS fosse melhor em termos de memória do que a A* no entanto a RBFS é muito melhor temporalmente que a A* e isso não é comum. Isto deve-se à falta de qualidade da heurística. Em teoria se a heurística for realmente boa, a procura A* seria mais rápida mas devido ao problema não foi possível contruir uma heurística muito informada pelo que o branching factor aumenta imenso e portanto a procura A* torna-se ineficiente e lenta e nessas condições a RBFS não é tão afetada. Este facto pode ser comprovado pela diferença de tempos destas procuras entre as tabelas 8 e 9.

I1.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,450	3085	26365
Greedy	0,609	196	1739
A*	0,258	94	790
IDS	0,194	417	3508
RBFS	0,126	43	356
DFS	∞	∞	∞

Tabela 1

I2.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,173	107	1121
Greedy	0,167	2	21
A*	0,167	3	31
IDS	0,168	11	118
RBFS	0,166	2	21
DFS	∞	∞	∞

Tabela 2

I3.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,186	158	11674
Greedy	0,166	3	34
A*	0,173	15	168
IDS	0,146	18	203
RBFS	0,165	3	34
DFS	∞	∞	∞

Tabela 3

I4.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,167	5	40
Greedy	0,166	1	8
A*	0,168	1	8
IDS	0,166	1	8
RBFS	0,168	1	8
DFS	∞	∞	∞

Tabela 4

I5.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,172	15	67
Greedy	0,165	2	9
A*	0,167	2	9
IDS	0,166	5	21
RBFS	0,167	2	9
DFS	∞	∞	∞

Tabela 5

I6.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,230	578	5976
Greedy	0,167	6	65
A*	0,186	30	321
IDS	0,172	61	644
RBFS	0,171	6	65
DFS	∞	∞	∞

Tabela 6

I7.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	0,220	538	5100
Greedy	0,174	20	203
A*	0,196	43	427
IDS	0,171	64	612
RBFS	0,171	10	97
DFS	∞	∞	∞

Tabela 7

I8.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
BFS	1 111	7089	70724
Greedy	0,511	153	1512
A*	0,355	113	1129
IDS	0,231	793	7904
RBFS	0,174	47	477
DFS	∞	∞	∞

Tabela 8

I8.txt	Tempo de Execução(s)	Nº nós expandidos	Nº nós gerados
ID A*	0,25	833	8287
Greedy	17,273	1036	10192
A*	18,839	1120	11204

Tabela 9

```python
def h(self, node: Node):
    """ Função heuristica utilizada para a procura A*. """
    cordAlvo = node.state.board.posAlvo
    corAlvo = node.state.board.corAlvo
    cordRobot = node.state.board.boardRobots[corAlvo]
    if(cordRobot[0] == cordAlvo[0]) or (cordAlvo[1] == cordRobot[1]):
        return 1
    return 2
```

Função heurística primitiva