Projeto I - Buscas

CTC- 17 Inteligência Artificial

Prof. Paulo André Castro
Alyson Fernandes Basilio
04 de setembro de 2017

1.Objetivo

Exercitar e fixar conhecimentos adquiridos sobre Resolução de Provas através de Busca. A linguagem utilizada para a implementação dos algoritmos foi o Python.

2. Descrição Atividade 1

Encontre o menor caminho entre as cidades 203 e 600 do Uruguai (arquivo Uruguay.csv). O arquivo tem os seguintes campos: ID da cidade, coordenada x, coordenada y e lista de adjacências. Cada lista de adjacências da cidade C contém os IDs das cidades para as quais há ligação a partir da cidade C. A distância entre as cidades pode ser calculada a partir das coordenadas cartesianas (x,y) disponibilizadas no arquivo Uruguay.csv. Utilize os algoritmos greedy e A* para fazer o trabalho, compare os resultados e explicite as funções de avaliação usadas heurística e no relatório.

2.1. Algoritmo Greedy

Em primeiro lugar, calculou-se a distância entre cada cidade do mapa e a cidade destino. A partir da cidade inicial seguiu-se o algoritmo descrito no pseudo-código a seguir:

```
paths = {}
c = origem
distances = {}
while c!=destino:
    visitedAllNeighboorsOfc = True
    for city in Neighboors(c):
        if not_visited(city):
            if city is not in cities_queue:
                paths[city] = c # 'c' is the city before 'city'
                cities_queue.append(city)
            visitedAllNeighboorsOfc = False
    if visitedAllNeighboors:        # if already visited all neighboors, return
        c = paths[c]                # to the city before
    else:   # else, select the nearest city to the destiny from cities_queue
        for item in sorted(distances, key=distances.get):
            if(not graph[item][1]):
                c = item
                graph[item][1] = True
                break
return paths
```

A distância total percorrida foi de: 121.74
O número total de cidades percorridas no algoritmo foi de: 438
O número de cidades no caminho foi de: 116

2.2. Algoritmo A*

Em segundo lugar, calculou-se a distância entre cada cidade do mapa e a cidade destino. A partir da cidade inicial seguiu-se o algoritmo descrito no pseudo-código a seguir:

```
paths = {}
c = origem
distances = {}
distances[c]=0
estimated_distances = {}
estimated_distances[origem] = getDistance(origem,destino)
graph[origem][1]=True
paths[c]=0
while c!=destino:
    for city in Neighboors(c):
        dist_estim = distances[c]+getDistance(c,city)+i[1]
        # dist_estim = distance traveled up to c + distance between c and city +
        # straigh distance between city and destiny
        if estimated_distances.get(city) is None:    # if city is not in
            paths[city] = c                          # estimated_distances, add
            estimated_distances[i[0]] = dist_estim   # that path to city

        elif dist_estim<estimated_distances[i[0]]:   # if the estimated distance
            paths[i[0]] = c                          # to destiny is smaller
            estimated_distances[i[0]] = dist_estim   # than the estimated
                                                     # distance calculated
                                                     # before, update the path
    # get the shortest path
    for item in sorted(estimated_distances, key=estimated_distances.get):
        if(not graph[item][1]):
            distances[item] = distances[paths[item]] +
                              getDistance(paths[item],item)
            c = item
            graph[item][1] = True
            break
return paths
```

A distância total percorrida foi de: 93.56
O número total de cidades percorridas no algoritmo foi de: 593
O número de cidades no caminho foi de: 112

2.3. Resultados e Discussões

Caminho encontrado pelo Algoritmo Greedy:

600, 595, 590, 586, 584, 580, 576, 575, 570, 567, 563, 559, 558, 555, 551, 548, 544, 541, 539, 535, 530, 527, 526, 523, 519, 517, 513, 508, 504, 500, 499, 494, 490, 485, 483, 478, 473, 469, 465, 460, 459, 454, 451, 447, 442, 439, 438, 435, 432, 431, 426, 424, 422, 419, 416, 412, 411, 408, 404, 399, 395, 394, 392, 387, 384, 381, 380, 375, 374, 371, 365, 364, 359, 357, 355, 353, 348, 346, 345, 340, 335, 330, 325, 321, 317, 312, 310, 305, 300, 296, 294, 292, 288, 283, 280, 277, 274, 270, 265, 262, 257, 253, 248, 245, 242, 239, 236, 231, 230, 227, 224, 219, 217, 211, 207, 205, 203

Caminho encontrado pelo Algoritmo A*:

600, 595, 590, 586, 584, 581, 577, 574, 569, 566, 561, 563, 560, 556, 554, 549, 546, 542, 537, 535, 530, 527, 526, 523, 519, 515, 513, 508, 504, 500, 499, 494, 490, 485, 483, 478, 476, 470, 466, 461, 458, 454, 451, 447, 442, 439, 438, 435, 432, 431, 426, 424, 422, 419, 413, 411, 408, 404, 400, 398, 396, 393, 388, 383, 382, 378, 373, 371, 365, 364, 359, 354, 349, 344, 342, 339, 335, 330, 325, 321, 317, 312, 308, 307, 302, 297, 293, 288, 283, 280, 277, 272, 268, 265, 262, 257, 253, 248, 246, 244, 241, 239, 235, 232, 230, 225, 223, 218, 214, 217, 211, 206, 203

Tabela 1: Resultados Gerais

	Greedy	**A***
Distância total percorrida	121.74	93.56
Cidades percorridas pelo algoritmo	438	593
Número de cidades no caminho	116	112

Como esperado, o algoritmo A* obteve o caminho mais curto, contudo ele percorreu mais cidades no grafo para encontrar este resultado.

3. Descrição Atividade 2

Crie um agente capaz de jogar o tic tac toe (jogo da velha) contra um usuário humano. A interface gráfica pode ser bastante simples inclusive em modo texto, porém deve permitir ao usuário perceber qual a situação atual do jogo e selecionar sua próxima jogada. O usuário humano é o "X" e sempre dá o primeiro lance. Implemente o algoritmo básico de minimax e decida entre utilizar uma função heurística de sua escolha ou implementar a poda alpha-beta com a utilidade do estados finais do jogo. Explique a razão da sua decisão e qual o impacto no "desempenho" do agente?
Obs.:(Você pode utilizar bibliotecas que lidam com as estruturas de dados árvore e/ou grafos)

3.1. Algoritmo Minimax

Pseudo código:

```
def minimax_move(self, grid):
    if self.isLeaf(grid): # check if its a Leaf
        return self.check_if_win(grid)   # a win for computer is 1, for human
                                         # is -1 and a tie is 0
    if self.isMaxNode(grid):
        v = -2
        for i in range(9):
            aux = self.translate_move(i + 1)
            if grid[aux[0]][aux[1]] == '-':
                vLinha = self.minimax_move(self.heuristic(self.copy(grid), i +
1, self.player))
                if vLinha>v:
                    v = vLinha
        return v
    if not self.isMaxNode(grid):
        v = 2
        for i in range(9):
            aux = self.translate_move(i + 1)
            if grid[aux[0]][aux[1]] == '-':
                vLinha = self.minimax_move(self.heuristic(self.copy(grid), i +
1,self.other_player))
                if vLinha<v:
                    v = vLinha
        return v
```

3.2. Resultados

Exemplo de Jogo empatado:

```
Let's play Tic-Tac-Toe!
Enter the coordinates for your move.
Example: 6 is the middle row right column.

  0 1 2
0 -  -  -
1 -  -  -
2 -  -  -
Your move, X: 1

  0 1 2
0 X - -
1 -  -  -
2 -  -  -
move =  5

  0 1 2
0 X - -
1 - O -
2 -  -  -
Your move, X: 9

  0 1 2
0 X - -
1 - O -
2 -  - X
move =  2

  0 1 2
0 XO -
1 - O -
2 -  - X
Your move, X: 8

  0 1 2
0 XO -
1 - O -
2 - X X
move =  7

  0 1 2
0 XO -
1 - O -
2 OXX
Your move, X: 3

  0 1 2
0 XOX
1 - O -
2 OXX
move =  6

  0 1 2
0 XOX
1 - OO
2 OXX
Your move, X: 4

  0 1 2
0 XOX
1 XOO
2 OXX
Game Over.  It's a tie.
```

Exemplo de vitória do computador:

```
0 1 2
0 -  -  -
1 -  -  -
2 -  -  -
Your move, X: 2
```

```
  0 1 2
0 - X -
1 -  -  -
2 -  -  -
move =  1
```

```
  0 1 2
0 OX -
1 -  -  -
2 -  -  -
Your move, X: 6
```

```
  0 1 2
0 OX -
1 -  - X
2 -  -  -
move =  7
```

```
  0 1 2
0 OX -
1 -  - X
2 O-  -
Your move, X: 4
```

```
  0 1 2
0 OX -
1 X-X
2 O- -
move =  5
```

```
  0 1 2
0 OX-
1 XOX
2 O-  -
Your move, X: 9
```

```
  0 1 2
0 OX -
1 XOX
2 O- X
move =  3
```

```
  0 1 2
0 OXO
1 XOX
2 O- X
You win, O!
```

Não foi possível simular um jogo em que o usuário humano ganhasse, apenas empatasse.

4. Conclusões

Foi interessante aprender soluções de inteligência para resolução de problemas, tanto de otimização quanto de competição.

Na primeira atividade houve uma pequena dificuldade na hora de implementar a heurística.

A segunda foi interessante pois ficou impossível ganhar da máquina, afinal ela sempre dava um jeito de virar o jogo.