

ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



Cơ sở Trí tuệ nhân tạo

LAB 01: Uninformed Search & Informed Search

Sinh viên thực hiện: Nguyễn Phúc Thuận

Giáo viên hướng dẫn:
GS.TS. Lê Hoài Bắc
Thầy Nguyễn Bảo Long

Thành phố Hồ Chí Minh - 2023

Table of Contents

I. Bài toán tìm kiếm.....	3
1. Các thành phần của bài toán tìm kiếm.....	3
2. Bài toán tìm kiếm nói chung.....	3
II. Ý tưởng và đánh giá thuật toán.....	4
1. Depth First Search (DFS).....	4
2. Breadth First Search (BFS).....	5
3. Uniform Cost Search (UCS).....	6
4. A Star (A*).....	7
IV. So sánh các thuật toán.....	10
1. UCS, Greedy và A*.....	10
2. UCS và Dijkstra.....	10
V. Kết quả thực nghiệm.....	11
1. DFS.....	11
2. BFS.....	12
3. UCS và GBFS.....	13
4. Astar.....	15
VI. Tham khảo.....	16

I. Bài toán tìm kiếm

1. Các thành phần của bài toán tìm kiếm

Một bài toán tìm kiếm có 05 thành phần:

- Q : tập hữu hạn các trạng thái.
- S : tập các trạng thái ban đầu. $S \subseteq Q$
- G : tập các trạng thái đích. $G \subseteq Q$
- Hàm sinh trạng thái con P : $Q \rightarrow P(Q)$ là một hàm nhận một trạng thái đầu vào và trả ra kết quả là một tập trạng thái con của trạng thái đó. Nói cách khác, $P(s)$ là “tập các trạng thái có thể đạt được từ s trong một bước”.
- Chi phí cost: $cost(q_1, q_2)$ là hàm nhận đầu vào là 02 trạng thái q_1, q_2 kế nhau làm đầu vào, và trả ra chi phí của bước di chuyển từ trạng thái q_1 đến trạng thái q_2 .

2. Bài toán tìm kiếm nói chung

Có 2 cách giải bài toán tìm kiếm bao gồm: tìm kiếm mù (uninformed search) và tìm kiếm với tri thức bổ sung (informed search).

Tiêu chí	Informed Search	Uninformed Search
Tên gọi khác	Tìm kiếm Heuristic	Tìm kiếm mù
Sử dụng tri thức bổ sung	Có	Không
Hiệu quả tìm kiếm	Nhanh	Chậm
Khả năng hoàn thành	Có thể không hoàn thành	Luôn hoàn thành
Chi phí tìm kiếm	Thấp	Cao
Gợi ý hướng tìm kiếm	Có trước gợi ý dựa trên tri thức bổ sung	Không có bất kì gợi ý nào
Độ phức tạp của việc cài đặt	Cài đặt đơn giản	Cài đặt phức tạp hơn
Khả năng giải bài toán lớn	Có thể giải được trong mức độ cho phép khi bài toán lớn dần.	Khó có thể giải được trong mức độ cho phép khi bài toán lớn dần.
Các thuật toán	Greedy A* search Hill Climbing	DFS BFS Dijkstra

Cách giải một bài toán tìm kiếm nói chung:

<p>Xét tập trạng thái Q, trạng thái bắt đầu S, trạng thái kết thúc G:</p> <p>Khởi tạo tập $closed_set$ rỗng, tập $open_set$ chứa S</p> <p>Khi $open_set$ khác rỗng thực hiện:</p> <ul style="list-style-type: none"> Lấy từ $open_set$ ra một trạng thái q Nếu q là G thì trả ra đường đi đến q Nếu q đã nằm trong $closed_set$ thì bỏ qua nó Chuyển q sang $closed_set$ Lấy danh sách trạng thái kề (trạng thái con) neighbors của q Với mỗi trạng thái con nb trong neighbors: Nếu nb chưa nằm trong $closed_set$ thì thêm nó vào $open_set$ 	<pre>def search(Q, S, G): closed_set = [] open_set = [S] while not open_set.empty(): q = open_set.get() if q == G: return backtrack(q) if q in closed_set: continue open_set.remove(q) closed_set.append(q) neighbors = q.getNeighbors() for nb in neighbors: if not nb in closed_set: open_set.append(nb)</pre>
--	--

II. Ý tưởng và đánh giá thuật toán

Xét các thông số sau:

N là số trạng thái của bài toán, hay $N=|Q|$

B là số trạng thái con trung bình ($B > 1$)

L_{op} là số lần chuyển đổi trạng thái tối ưu (tức là số lần chuyển đổi trạng thái từ S đến G với chi phí thấp nhất)

L_{max} là số lần chuyển đổi trạng thái lớn nhất không chu trình từ S đến trạng thái bất kì nào đó.

N_Q là kích cỡ của hàng đợi ưu tiên $N_Q=|Q_{priority}|$

1. Depth First Search (DFS)

a. Ý tưởng

Từ một đỉnh, ta duyệt theo một nhánh bất kì, sau khi đi hết nhánh đó thì quay lại duyệt các đỉnh còn lại. Thuật toán dừng lại khi tất cả các đỉnh đều đã được duyệt qua.

b. Mã giả

<p>Xét tập trạng thái Q, trạng thái bắt đầu S, trạng thái kết thúc G:</p> <p>Khởi tạo tập $closed_set$ rỗng, tập $open_set$ là stack chứa S</p> <p>Khi $open_set$ khác rỗng thực hiện:</p> <ul style="list-style-type: none"> Lấy từ đỉnh $stack$ $open_set$ ra trạng thái q Nếu q là G thì trả ra đường đi đến q

Nếu q đã nằm trong `closed_set` thì bỏ qua nó
 Chuyển q sang `closed_set`
 Lấy danh sách trạng thái kề (trạng thái con) neighbors của q
 Với mỗi trạng thái con nb trong neighbors:
 Nếu nb chưa nằm trong `closed_set` thì **push** nó vào `đỉnh open_set`

```
def search(Q, S, G):
    closed_set = []
    open_set = stack[S]
    while not open_set.empty():
        q = open_set.pop()
        if q in closed_set: continue
        if q == G:
            return backtrack(q)
        closed_set.append(q)
        neighbors = q.getNeighbors()
        for nb in neighbors:
            if not nb in closed_set:
                open_set.push(nb)
```

Ngoài ra, ta cũng tính thêm trường hợp nb không nằm trong `open_set` để đảm bảo không tạo ra chu trình.

c. Đánh giá thuật toán

- Khả năng hoàn thành: có nếu không có nhánh nào đi đến vô cùng và không có chu trình.
- Tối ưu: không
- Độ phức tạp về thời gian: $O(B^{L_{max}})$
- Độ phức tạp về không gian: $O(L_{max})$

2. Breadth First Search (BFS)

a. Ý tưởng

Thuật toán dựa trên nguyên lý cháy lan của ngọn lửa, bắt nguồn từ một đỉnh duyệt qua tất cả đỉnh kề của nó. Với mỗi đỉnh kề đã duyệt, ta lại tiếp tục thực hiện duyệt sang các đỉnh kề của nó. Thuật toán dừng lại khi tất cả các đỉnh đều đã được duyệt qua.

b. Mã giả

Xét tập trạng thái Q , trạng thái bắt đầu S , trạng thái kết thúc G :
 Khởi tạo tập `closed_set` rỗng, tập `open_set` là queue chứa S
 Khi `open_set` khác rỗng thực hiện:
 Lấy từ đầu queue `open_set` ra trạng thái q
 Nếu q là G thì trả ra đường đi đến q

Nếu q đã nằm trong closed_set thì bỏ qua nó
 Chuyển q sang closed_set
 Lấy danh sách trạng thái kề (trạng thái con) neighbors của q
 Với mỗi trạng thái con nb trong neighbors:
 Nếu nb chưa nằm trong closed_set thì enqueue nó vào đuôi open_set

```
def search(Q, S, G):
    closed_set = []
    open_set = queue[S]
    while not open_set.empty():
        q = open_set.dequeue()
        if q in closed_set: continue
        if q == G:
            return backtrack(q)
        closed_set.append(q)
        neighbors = q.getNeighbors()
        for nb in neighbors:
            if not nb in closed_set:
                open_set.enqueue(nb)
```

Ngoài ra, ta cũng tính thêm trường hợp nb không nằm trong open_set để đảm bảo không tạo ra chu trình..

c. Đánh giá thuật toán

- Khả năng hoàn thành: có nếu bán kính lan rộng hữu hạn.
- Tối ưu: có nếu chi phí chuyển đổi giữa các trạng thái là như nhau.
- Độ phức tạp về thời gian: $O(\min(N, B^{L_{op}}))$
- Độ phức tạp về không gian: $O(\min(N, B^{L_{op}}))$

3. Uniform Cost Search (UCS)

a. Ý tưởng

Tương tự như BFS nhưng thay vì duyệt tất cả trạng thái kề ở mỗi bước thì ta ưu tiên chọn trạng thái có chi phí thấp nhất để duyệt, sau đó cập nhật chi phí của các trạng thái con của trạng thái đó.

b. Mã giả

Xét tập trạng thái Q, trạng thái bắt đầu S, trạng thái kết thúc G:
 Khởi tạo tập closed_set rỗng, tập open_set là priority queue chứa “S với chi phí 0”
 Khi open_set khác rỗng thực hiện:
 Lấy từ đầu queue open_set ra trạng thái q có f(q) nhỏ nhất
 Nếu q là G thì trả ra đường đi đến q

<p>Nếu q đã nằm trong closed_set thì bỏ qua nó Chuyển q sang closed_set Lấy danh sách trạng thái kề (trạng thái con) neighbors của q Với mỗi trạng thái con nb trong neighbors: Nếu nb đã nằm trong closed_set thì bỏ qua Tính toán cost(nb) khi đi qua q: $\text{cost}(\text{nb}) = \text{cost}(q) + \text{cost}(q, \text{nb})$ Nếu nb chưa nằm trong open_set thì thêm nó và cost vào open_set Ngược lại, nếu nb đã nằm trong open_set và cost(nb) hiện tại lớn hơn cost(nb) khi qua q mới thì cập nhật đường đi đến nb và cost(nb) trong open_set</p>
<pre>def search(Q, S, G): closed_set = [] open_set = PriorityQueue{(S, 0)} while not open_set.empty(): q = open_set.get() if q in closed_set: continue if q == G: return backtrack(q) closed_set.append(q) neighbors = q.getNeighbors() for nb in neighbors: if not nb in closed_set: cost_from_q = cost(q) + cost(q to nb) if nb not in open_set: open_set.put((nb, cost_from_q)) elif cost_from_q < cost(nb): update path to nb and cost(nb) in open_set</pre>

c. Đánh giá thuật toán

- Khả năng hoàn thành: có.
- Tối ưu: có.
- Độ phức tạp về thời gian: $O(\log N_Q \times \min(N, B^{L_{op}}))$
- Độ phức tạp về không gian: $O(\min(N, B^{L_{op}}))$

4. A Star (A*)


a. Ý tưởng

Là thuật toán tìm kiếm sử dụng thông tin bổ sung (hàm heuristic).

Tương tự như UCS nhưng thay vì chỉ tính chi phí di chuyển giữa các trạng thái, ta tính thêm một giá trị tri thức bổ sung (ví dụ khoảng cách từ trạng thái hiện tại đến trạng

thái đích), sau đó cập nhật tổng chi phí di chuyển và giá trị tri thức bổ sung của các trạng thái con của trạng thái đó.

b. Mã giả

Xét tập trạng thái Q , trạng thái bắt đầu S , trạng thái kết thúc G ,  là tổng chi phí chuyển đổi trạng thái và giá trị tri thức bổ sung:

Khởi tạo tập `closed_set` rỗng, tập `open_set` là priority queue chứa “ S với  = 0”

Khi `open_set` khác rỗng thực hiện:

Lấy từ đầu queue `open_set` ra trạng thái q có f nhỏ nhất

Nếu q là G thì trả ra đường đi đến q

Nếu q đã nằm trong `closed_set` thì bỏ qua nó

Chuyển q sang `closed_set`

Lấy danh sách trạng thái kề (trạng thái con) neighbors của q

Với mỗi trạng thái con nb trong neighbors:

Nếu nb đã nằm trong `closed_set` thì bỏ qua

Tính toán chi phí đi đến nb khi đi từ q

Nếu nb chưa nằm trong `open_set` thì thêm nó và chi phí đi đến nó vào `open_set`

Ngược lại, nếu nb đã nằm trong `open_set` và chi phí đi đến nb hiện tại lớn hơn chi phí đi đến nb từ q thì cập nhật đường đi đến nb và chi phí của nó trong `open_set`

```
def search(Q, S, G):
    closed_set = []
    open_set = PriorityQueue{(S, 0)}
    while not open_set.empty():
        q = open_set.get()
        if q in closed_set:
            continue
        if q == G:
            return backtrack(q)
        closed_set.append(q)
        neighbors = q.getNeighbors()
        for nb in neighbors:
            if not nb in closed_set:
                cost_from_q = cost(q) + cost(q to nb)
                if nb not in open_set:
                    open_set.put((nb, cost_from q))
                elif cost_from_q < cost(nb):
                    update path, cost to nb
```

c. Đánh giá thuật toán

- Khả năng hoàn thành: có.

- Tối ưu: có khi heuristic đủ tốt.

- Độ phức tạp về thời gian: $O(\log N_Q \times \min(N, B^{L_{op}}))$

- Độ phức tạp về không gian: $O(\min(N, B^{L_{op}}))$

d. Heuristic trong tìm kiếm:

Heuristic là phương pháp giải quyết vấn đề dựa trên phỏng đoán, kinh nghiệm (hay còn gọi là tri thức bổ sung) để tìm ra giải pháp gần như là tốt nhất và nhanh chóng trong khoảng giá trị chấp nhận được. Hàm heuristic trong bài toán tìm kiếm là hàm ứng với mỗi trạng thái sẽ mang một giá trị tri thức bổ sung đối với trạng thái kết thúc, và dựa vào giá trị hàm này, ta lựa chọn hành động tiếp theo (tức là đưa ra quyết định chuyển đến trạng thái con nào).

Một số loại hàm heuristic trong A^* bao gồm:

- Hàm Manhattan là hàm heuristic đơn giản nhất trong thuật toán A^* . Hàm này tính khoảng cách từ trạng thái hiện tại đến trạng thái đích bằng cách lấy tổng giá trị tuyệt đối của hiệu giữa tọa độ x và y của trạng thái hiện tại và trạng thái đích. Hàm Manhattan có thể được sử dụng trong các bài toán tìm đường đi ngắn nhất trên một lưới ô vuông chỉ đi theo 4 hướng.

$$h(q) = |x_q - x_G| + |y_q - y_G|$$

- Hàm Euclid là hàm heuristic được sử dụng phổ biến trong thuật toán A^* . Hàm này tính khoảng cách từ trạng thái hiện tại đến trạng thái đích bằng cách sử dụng định lý Pythagoras để tính khoảng cách giữa 2 điểm trong không gian Euclid. Hàm Euclid có thể được sử dụng trong các bài toán tìm đường đi ngắn nhất trên một lưới ô vuông có thể đi theo 8 hướng.

$$h(q) = \sqrt{(x_q - x_G)^2 + (y_q - y_G)^2}$$

- Hàm Diagonal là hàm heuristic khác được sử dụng trong thuật toán A^* . Hàm Diagonal có thể được sử dụng trong các bài toán tìm đường đi ngắn nhất trên một lưới ô vuông có thể đi theo 8 hướng mà không kể đến chi phí giữa các trạng thái.

$$h(q) = D \times (dx + dy) + (D_2 - 2D) \times \min(dx, dy) \quad \text{với } D = 1, \quad D_2 = \sqrt{2}$$

- Hàm Octile là hàm heuristic cuối cùng trong thuật toán A^* . Hàm Octile có thể được sử dụng trong các bài toán tìm đường đi ngắn nhất trên một lưới ô vuông có thể đi theo 8 hướng.

$$h(q) = \max(dx, dy) + (\sqrt{2} - 1) \times \min(dx, dy)$$

IV. So sánh các thuật toán

1. UCS, Greedy và A*:

Tiêu chí	UCS	Greedy	A*
Chọn trạng thái để mở	Trạng thái có chi phí g thấp nhất	Trạng thái có giá trị heuristic h thấp nhất	Trạng thái có giá trị $f = g + h$ thấp nhất
Số trạng thái mở	Nhiều, có thể mở nhiều trạng thái không nằm trong đường đi tối ưu	Ít	Ít hơn UCS, có thể mở nhiều trạng thái không nằm trong đường đi tối ưu
Hoàn thành	Có	Có thể không	Có
Tối ưu	Có	Không	Có
Tốc độ	Chậm	Nhanh hơn A*	Nhanh hơn UCS

2. UCS và Dijkstra:

Tiêu chí	UCS	Dijkstra
Điều kiện dừng	Tìm được đường đi ngắn nhất đến G	Tìm được đường đi ngắn nhất đến tất cả các trạng thái.
Priority Queue	Trống lúc đầu, thêm các trạng thái trong khi duyệt	Khởi tạo tất cả đỉnh với chi phí vô cực
Bộ nhớ	Ít hơn Dijkstra	Nhiều

V. Kết quả thực nghiệm

1. DFS

```

280 father_node = g.get
281 g.start.set_color(orange)
282 g.goal.set_color(purple)
283 pygame.draw.line(sc, black, father_node, g.start, 1)
284
285
286 g.draw(sc)
287
288 print(f'Path: {path}')
289 print(f'Total cost: {total_cost}')
290

```

Nguyen Phuc Thuan - DFS

Path: [269, 291, 267, 289, 265, 287, 263, 285, 261, 283, 259, 281, 257, 279, 255, 277, 299, 323, 347, 325, 349, 327, 351, 329, 353, 331, 355, 333, 357, 335, 359, 337, 361, 339, 363, 341, 365, 366, 344, 320, 298, 274, 250, 226, 202, 178, 154, 130, 106, 82, 58, 34, 10]

Total cost: 52

Bắt đầu từ đỉnh 10, đi sâu theo một hướng đến khi không thể đi được nữa thì đổi sang hướng khác (thứ tự đổi hướng theo thứ tự ngược lấy đỉnh kề của Space định nghĩa: down right, down left, up right, up left, right, left, down, up).

2. BFS

```

79  r
80  c
81  0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22
82  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
83  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
84  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91
85  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114
86  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137
87  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160
88  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183
89  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206
90  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229
91  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252
92  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275
93  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298
94  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321
95  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344
96  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367

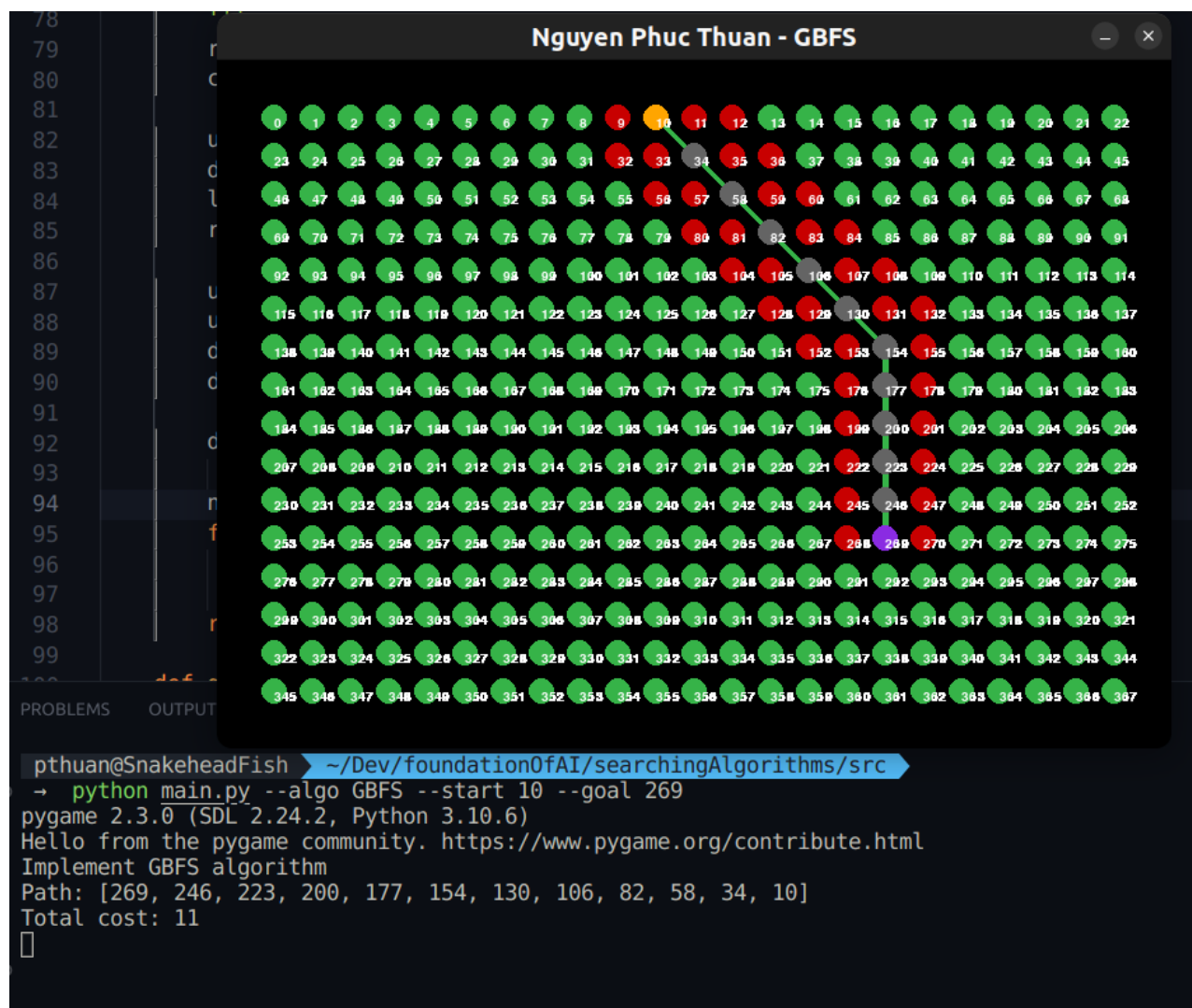
phtuan@SnakeheadFish ~/Dev/foundationOfAI/searchingAlgorithms/src
→ python main.py --algo BFS --start 10 --goal 269
pygame 2.3.0 (SDL 2.24.2, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Implement BFS algorithm
Path: [269, 245, 221, 197, 173, 149, 125, 102, 79, 56, 33, 10]
Total cost: 11

```

Bắt đầu từ đỉnh 10, quét ra 8 hướng tìm được 34, 32, 33, 11, 9, sau đó từ các đỉnh này lại lan sang 8 hướng bên cạnh, sau đó lại tiếp tục với các đỉnh được quét mới.

Mở rộng theo bán kính vuông so với điểm bắt đầu (10).

Greedy BFS:



4. Astar



Bắt đầu từ đỉnh 10, quét ra 8 hướng tìm được 34, 32, 33, 11, 9. Sau đó tính được đỉnh số 34 có $f_{34}=1+heuristic(34)=1+11.18$ nhỏ nhất nên mở rộng sang 34. Sau đó tiếp tục quét ra 8 hướng tìm được 56, 57, 58, 12, 35 và tính được 58 có f nhỏ nhất và mở rộng sang 58. Lặp lại tương tự đến khi tìm được 269.

VI. Tham khảo

- [CS 188 | Introduction to Artificial Intelligence, Fall 2018](#): Week 1 - Uninformed Search & A* Search and Heuristics.
- Artificial Intelligence: a Modern Approach, EBook, Global Edition.
- Thuật Giải A*: <https://www.iostream.vn/giai-thuat-lap-trinh/thuat-giai-a-DVnHj>
- A* Search Algorithms in GeeksForGeeks forum: <https://www.geeksforgeeks.org/a-search-algorithm>

- HẾT -