امیرمحمد رمضان نادری
۶۱۰۳۹۸۱۲۶
توضیحات تمرین دوم هوش مصنوعی

اجزای پازل:

- ابتدا state ها را توضیح می دهیم. ورودی مسئله ها یا همان state برابر یک آرایه دو بعدی ۳*۳ می باشد که از اعداد ۰ تا ۸ تشکیل شده اند. عدد صفر نماینده جایگاه خالی می باشد. البته ورودی تست ها به صورت یک آرایه یک بعدی می باشند که در ادامه آن را به یک آرایه دو بعدی تبدیل می کنیم. در زیر می توانید یک نمونه از آن را ببینید:
[[4,7,8],[5,3,0],[1,6,2]]

- اکشن ها در این مسئله برابر چهار حرکت {بالا, چپ, راست, پایین} می باشد.
این حرکت ها برای جابجایی کاشی صفر می باشد. یکی از این transition ها را می توانید در زیر مشاهده کنید:

[[2, 5, 3], [4, 1, 6], [0, 7, 8]] → [2, 5, 3], [0, 1, 6], [4, 7, 8]]

با اعمال اکشن بالا به همچین state ای می رسیم.

- در این بخش state نهایی را تایین می کنیم. State نهایی به شکل زیر می باشد:
[[1,2,3] , [4,5,6] , [7,8,0]]

هدف این مسئله این است که با انتخاب الگوریتم مناسب و انجام اکشن های مناسب به state نهایی برسیم.

- Path cost را برای این سوال یک در نظر می گیریم. به این معنا که برای انجام هر کدام از چهار اکشن گفته شده یک واحد به cost اضافه می شود.

- تابع heuristic را طبق اسلاید های درس می توانیم منهتن و misplaced در نظر بگیریم. در اسلاید ها ثابت شد هر دو تابع admissible هستند و طبق نظریه ای تابعی که مقدار بیشتری دارد را باید انتخاب کنیم زیرا به مقدار واقعی نزدیک تر است. پس تابع منهتن را انتخاب می کنیم. این تابع می گوید که ما چند کاشی مجاور را صرف نظر از اینکه خالی باشد یا نه باید جا به جا کنیم تا به state نهایی برسیم.

- حال کلاسی را که در کد پیاده سازی کردیم را تعریف می کنیم:
کلاس Puzzle را تعریف می کنیم که در init یک آرایه دو بعدی ۳*۳ به عنوان حالت اولیه

پازل می گیرد. سپس پدر هر نود, مقدار تابع heuristic و g_n و مقدار f_n که برابر جمع g_n و h_n می باشد و همچنین لیستی که فرزندان کاشی خالی که می تواند به آن خانه برود را تایین می کنیم. تایین جابجایی کاشی خالی را در تابع neighbors می توانید مشاهده کنید. تابع h_n برابر تابع منهتن است و g_n برابر هزینه جابجایی می باشد که برابر با یک در نظر گرفتیم. برای مثال برای state ای که در زیر تعریف شده, state های فرزندان را به همراه اکشن آن می توانید مشاهده کنید:

```
obj = Puzzle([[1,6,2],[5,3,0],[4,7,8]])
goal_obj = Puzzle([[1,2,3],[4,5,6],[7,8,0]])
obj.neighbors()[0]
```

```
[('up', [[1, 6, 0], [5, 3, 2], [4, 7, 8]]),
 ('left', [[1, 6, 2], [5, 0, 3], [4, 7, 8]]),
 ('down', [[1, 6, 2], [5, 3, 8], [4, 7, 0]])]
```

حال ۵ الگوریتم A_star BFS DFS IDS و UCS را اجرا می کنیم. ابتدا یک state آغازین به صورت زیر تعریف می کنیم:

```
test_obj = Puzzle([[2,5,3],[4,1,6],[0,7,8]])
```

سپس الگوریتم A_star را بر روی آن اجرا می کنیم. الگوریتم A_star به صورت زیر می باشد:

```
Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
while the OPEN list is not empty {
  Take from the open list the node node_current with the lowest
       f(node_current) = g(node_current) + h(node_current)
  if node_current is node_goal we have found the solution; break
  Generate each state node_successor that come after node_current
  for each node_successor of node_current {
    Set successor_current_cost = g(node_current) + w(node_current, node_successor)
    if node_successor is in the OPEN list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
    } else if node_successor is in the CLOSED list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
      Move node_successor from the CLOSED list to the OPEN list
    } else {
      Add node_successor to the OPEN list
      Set h(node_successor) to be the heuristic distance to node_goal
    }
    Set g(node_successor) = successor_current_cost
    Set the parent of node_successor to node_current
  }
  Add node_current to the CLOSED list
}
if(node_current != node_goal) exit with error (the OPEN list is empty)
```

نتیجه دنباله اکشن ها برای رسیدن به هدف نیز به صورت زیر می باشد:

```
[[[2, 5, 3], [4, 1, 6], [0, 7, 8]],
 [[2, 5, 3], [0, 1, 6], [4, 7, 8]],
 [[2, 5, 3], [1, 0, 6], [4, 7, 8]],
 [[2, 0, 3], [1, 5, 6], [4, 7, 8]],
 [[0, 2, 3], [1, 5, 6], [4, 7, 8]],
 [[1, 2, 3], [0, 5, 6], [4, 7, 8]],
 [[1, 2, 3], [4, 5, 6], [0, 7, 8]],
 [[1, 2, 3], [4, 5, 6], [7, 0, 8]],
 [7, 8, 0],
 [4, 5, 6],
 [1, 2, 3]]
```

بقیه الگوریتم ها را نیز می توانید در کد جوپیتر مشاهده کنید.

- با استفاده از کتابخانه های time و tracemalloc می توانیم زمان و مموری را برای هر الگوریتم به دست بیاوریم. state اولیه را همان قبلی در نظر می گیریم. نتیجه را برای الگوریتم را می توانید در زیر مشاهده کنید:

1. A_star: زمان: 0.01436 ns و مموری: 677236
2. DFS: زمان: 0.18935 ns و مموری: 1742204
3. BFS: زمان: 0.04737 ns و مموری: 1742204
4. IDS: زمان: 0.07644 ns و مموری: 1742204
5. UCS: زمان: 0.04918 و مموری: 1742204

جدول اکسل را نیز می توانید مشاهده کنید.