به نام خدا

هوش مصنوعی

گزارش تمرین اول

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

پاییز ۱۴۰۱

استاد:

دکتر مهدیه سلیمانی

دانشجو:

امیرحسین رحمتی

۹۹۱۰۳۹۲۲

heuristic function

کد مربوط به تابع هیوریستیک را در بخش زیر میشاهده میکنید.

```
1  def heuristic(state,target_place):
2      n = len(state)
3      h = 0
4      for i in range(n):
5          for j in range(n):
6              x_dif = abs(j - target_place[state[i][j]][1])
7              y_dif = abs(i - target_place[state[i][j]][0])
8              if (x_dif > n//2 ):
9                  x_dif = n - x_dif
10             if (y_dif > n//2):
11                 y_dif = n - y_dif
12             h = h +  x_dif * 5  + y_dif * 5
13     return h
```

تابع هیوریستیک را به صورت جمع کمترین فاصله خانه هر عدد تا خانه هدفش در نظر گرفتیم. که خانه هدف برای هر عدد در دیکشنری target_place قرار دارد. و نحوه محاسبه آن را در کد زیر مشاهده میکنید.

```
1  def find_target_places(target):
2      target_places = {}
3      for i in range(len(target)):
4          for j in range(len(target)):
5              target_places[target[i][j]] =  (i,j)
6      return target_places
```

همچنین از خط ۸ تا ۱۱ نحوه محاسبه کمترین فاصله حقیقی را مشاده میکنیم. به این صورت که اگر فاصله دو خانه بیشتر از نصف طول یا عرض جدول خانه ها باشد بدین معنی است که با چرخش در جهت مخالف این دو خانه سریع تر به یک دیگر میرسند و فاصله واقعی آنها برابر تفریق فاصله بدست آمده در مرحله قبل از ابعاد جدول است.

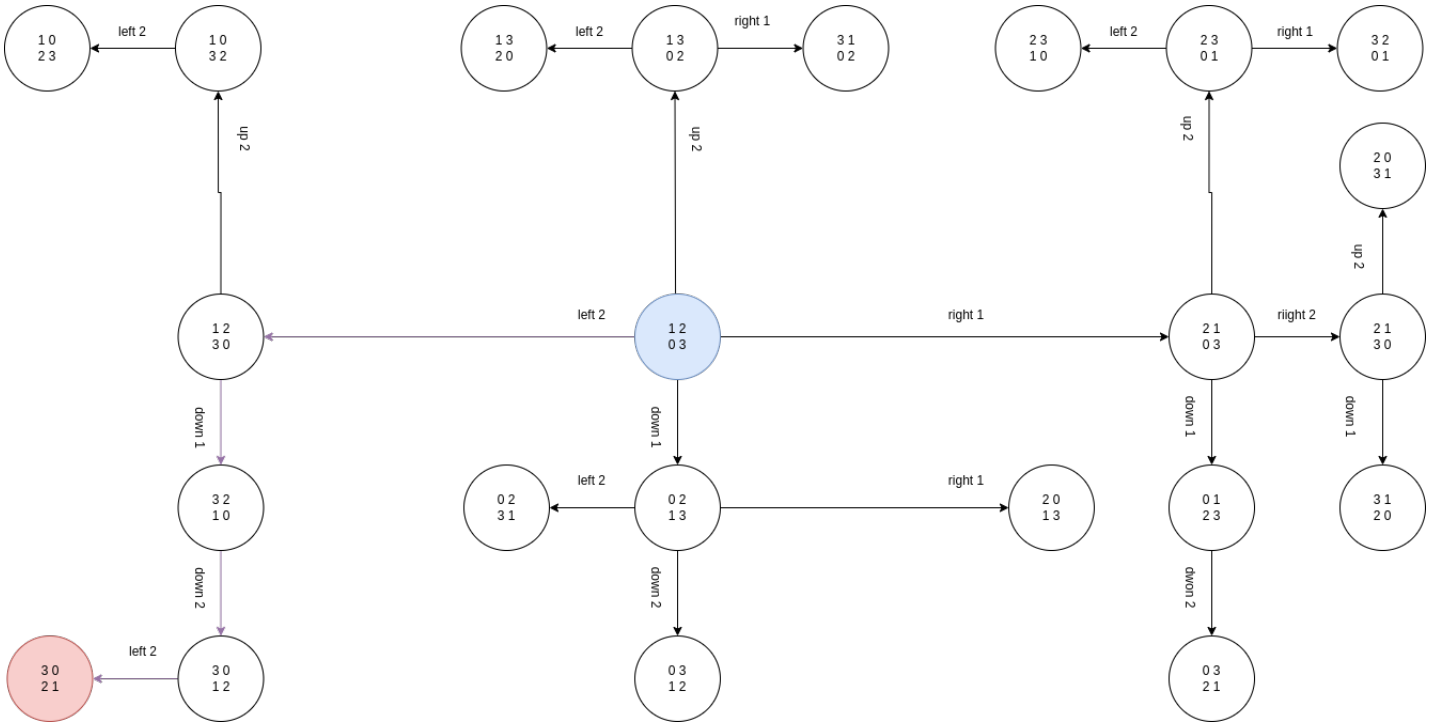
در انتها هنگام جمع کردن کمترین فاصله بدست آمده برای هر عدد آن را در ۵ ضرب میکنیم. اینکار باعث میشود admissibility تابع هیوریستیک از بین برود زیرا این تابع یک تخمین بالا از هزینه واقعی خواهد بود. و از بین رفتن admissibility باعث میشود جواب بدست آمده بهینه نباشد. اما شرط بهینه بودن جواب در سوال موجود نمیباشد.

اما مزیت اینکار هرس شدن راس های بیشتر و سریع تر رسیدن به جواب میباشد. همانطور که میدانیم در الگوریتم A_star تمام راس هایی با جمع هیورستیک و هزینه بیش از هزینه نهایی رسیدن به goal هرس میشوند. و با این ضریب دادن به تابع heuristic راس ها یا state های بیشتری جمع هیورستیک و هزینه بیشتر از هزینه نهایی خواهند داشت. و راس های بیشتری هرس خواهند شد. که این کار به کم شدن زمان حل مسیله کمک خواهد کرد.

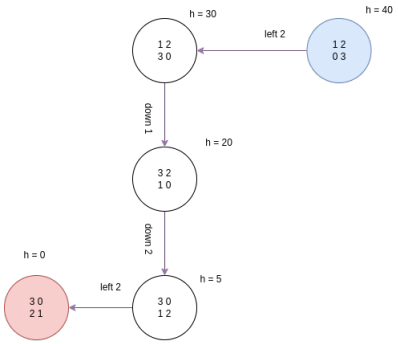
مقایسه

در ادامه عملکرد الگوریتم A_star و BFS را بر روی مثال های داده شده مقایسه خواهیم کرد.
تمام راس های بازدید شده تا رسیدن به goal را برای هر الگوریتم در تصاویر زیر مشاهده میکنید. توجه شود که در هر یک از الگوریتم ها state های تکراری بررسی نمیشوند.

مثال اول :
نحوه عملکرد الگوریتم BFS را در تصویر زیر مشاهده میکنید. همانطور که مشخص است برای رسیدن به goal در این الگوریتم تمام ۲۴ state موجود به صف frontier اضافه میشوند.

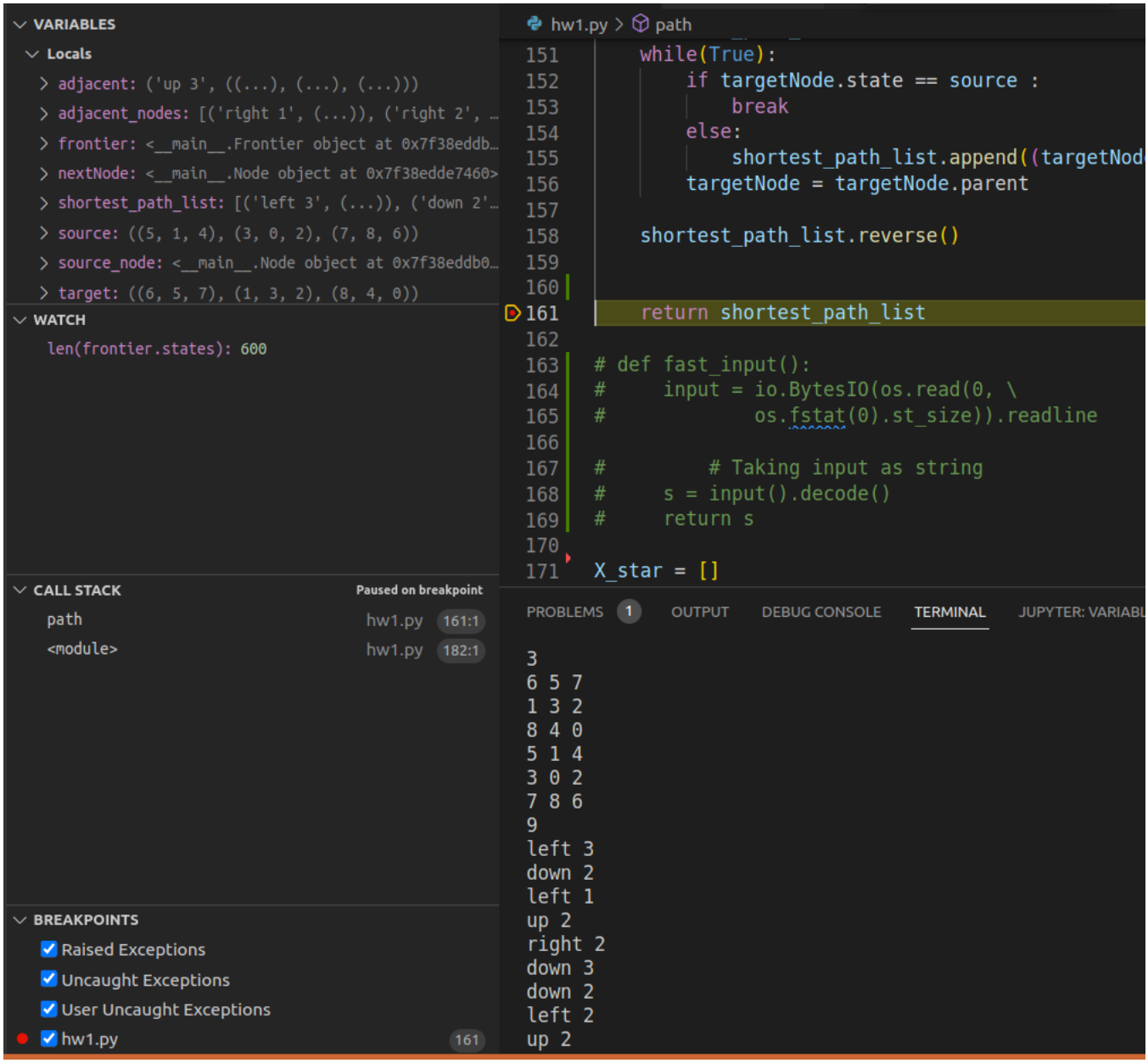


در تصویر بعدی نحوه عملکرد الگوریتم A_star را مشاهده میکنید. با استفاده از این الگوریتم در این مثال تنها ۴ راس بررسی میشود. و برتری مشهودی نسبت به الگوریتم BFS دارد.



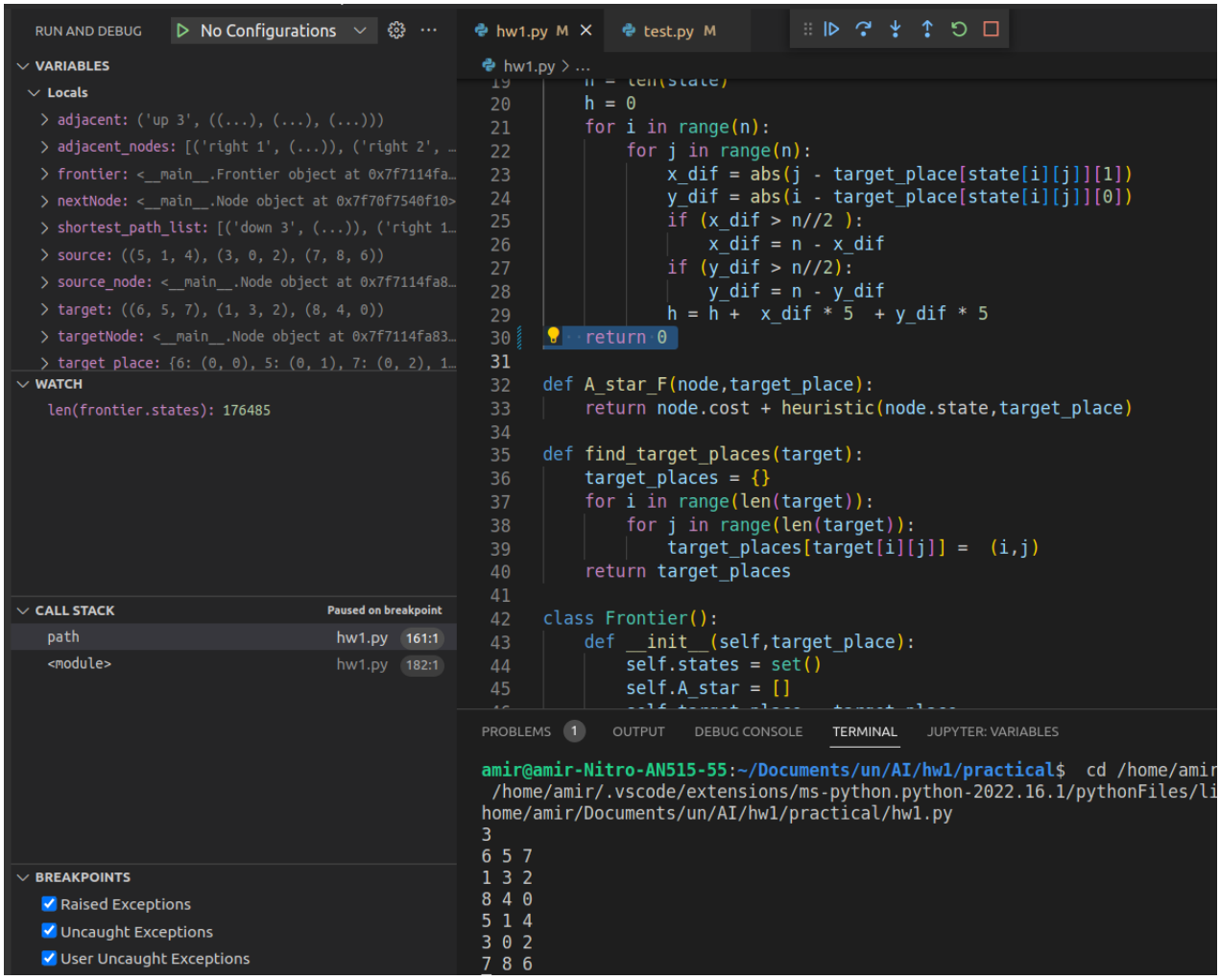
حال میخواهیم به دو مثال بعدی بپردازیم .

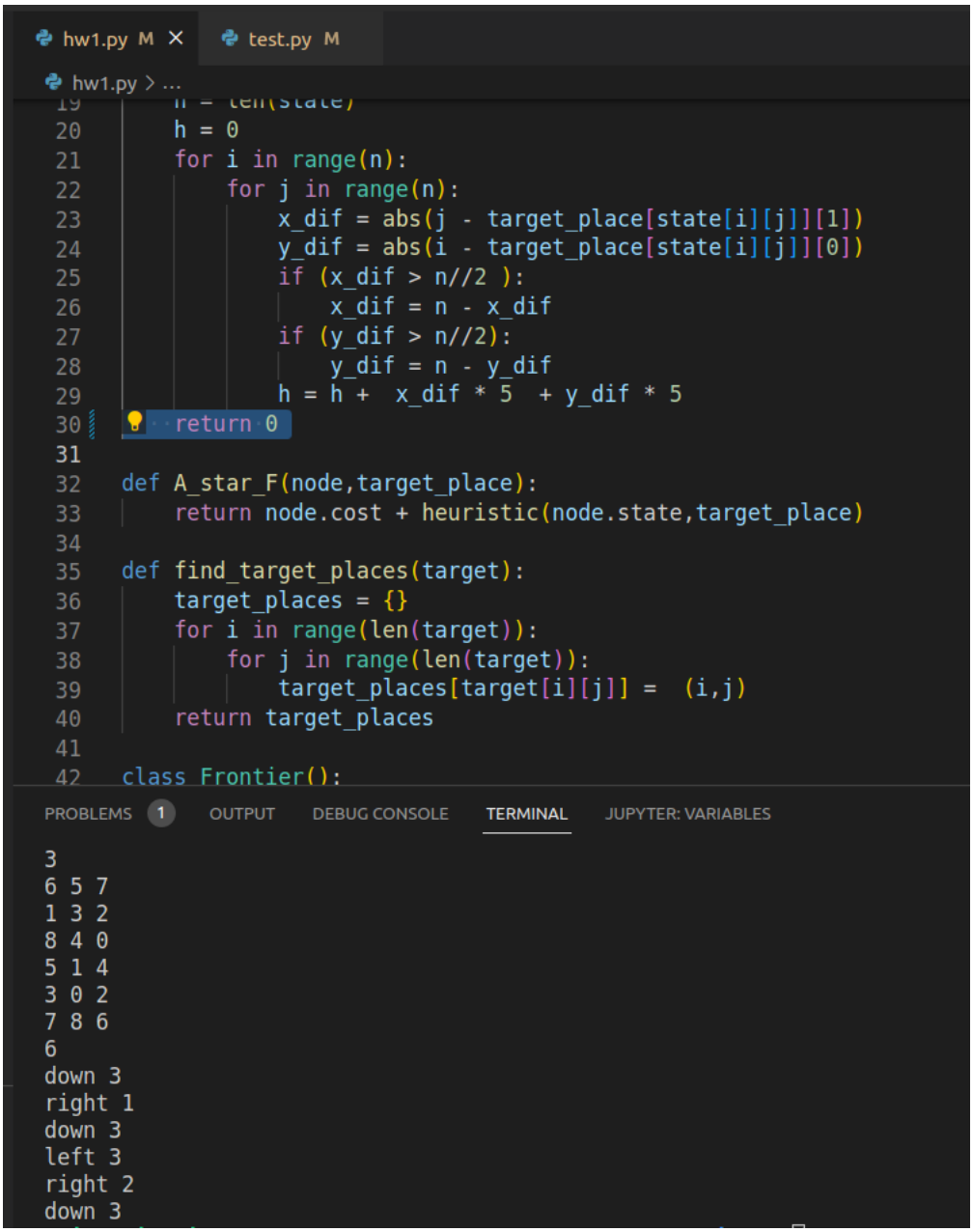
در تصویر زیر نتیجه اجرای برنامه نوشته شده با الگوریتم A_star را مشاهده میکنید. در سمت چپ تصویر در قسمت watch تعداد راس های اضافه شده به صف frontier در طول اجرای برنامه مورد بررسی قرار گرفته است. همانطور که مشخص است ۶۰۰ راس تا خارج شدن راس هدف از صف frontier مورد بررسی قرار گرفته است. و جواب ارایه شده با ۹ حرکت در ترمینال قابل مشاهده است.



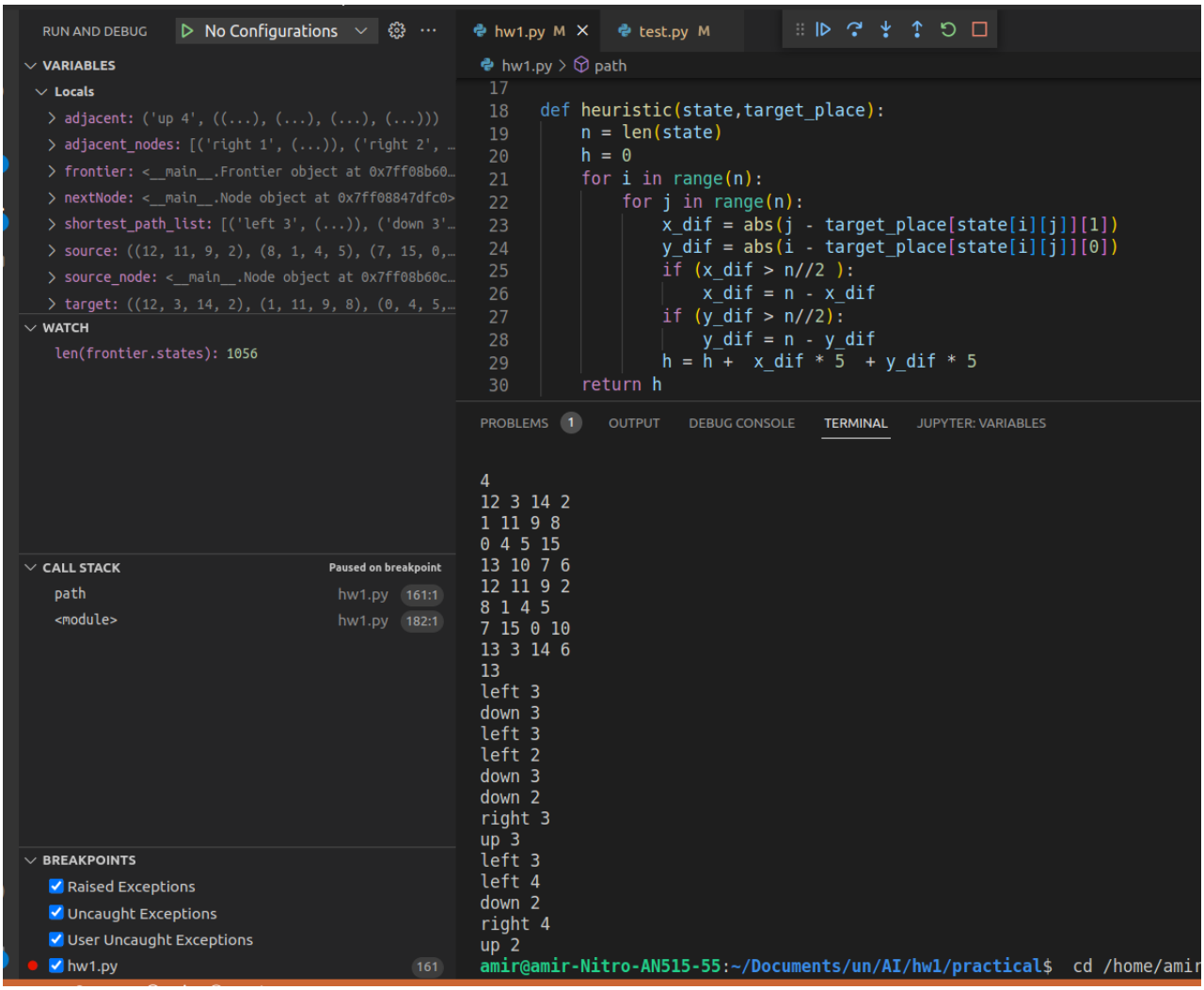
حال نوبت الگوریتم BFS است. برای اجرای الگوریتم BFS در برنامه نوشته شده تابع هیورستیک را برابر صفر قرار میدهیم با توجه به اینکه هزینه تمام حرکات یکسان است. برنامه ما عملا از الگوریتم BFS استفاده خواهد کرد.

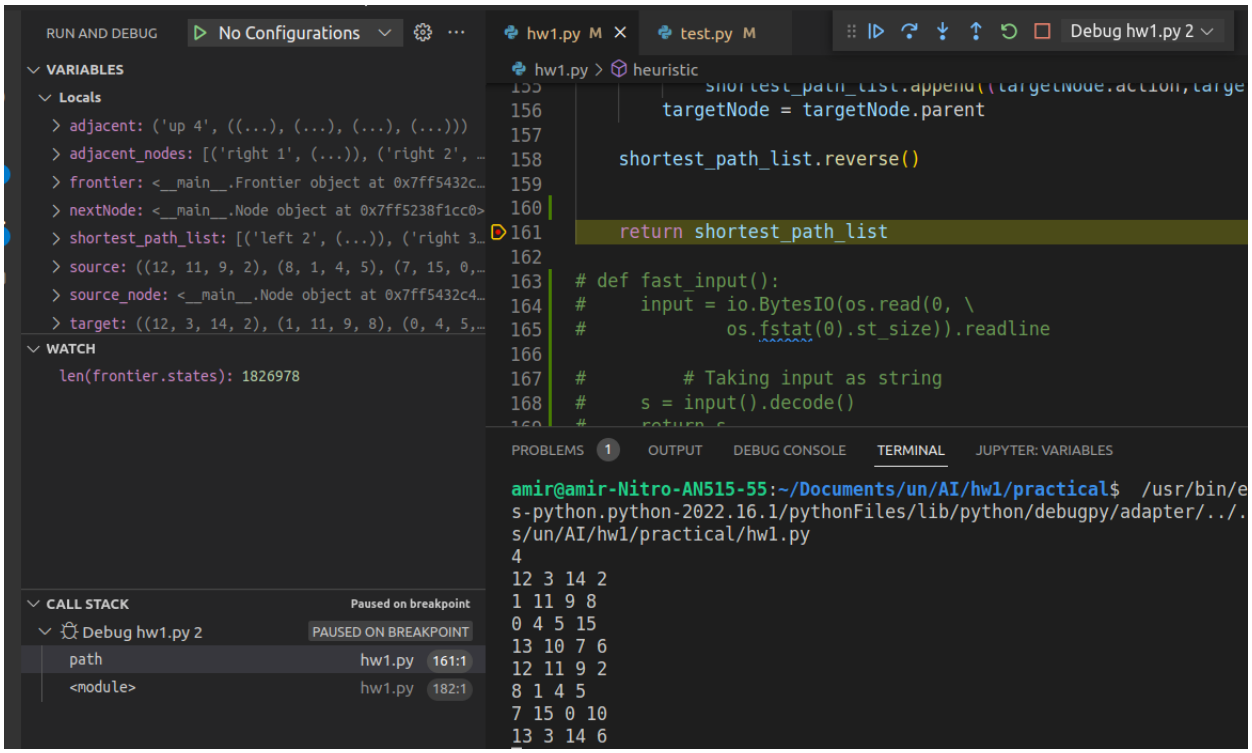
در دو تصویر بعدی اطلاعات مربوط به اجرای این برنامه با این الگوریتم را برای مثال دوم مشاهده میکنید. همانطور که در شکل اول مشخص است. ۱۷۶۴۸۵ راس تا پیدا کردن راس هدف مورد بررسی قرار گرفته است. که تفاوت بسیار قابل ملاحظه ای با این عدد در الگوریتم A_star یعنی ۶۰۰ دارد. همچنین در تصویر دوم مشاهده میکنید که جواب ارایه شده توسط الگوریتم BFS با ۶ حرکت است. که این موضوع همانطور که در ابتدای گزارش ذکر شد به دلیل از بین رفتن admissibility آن الگوریتم A_star جواب بهینه را در اختیار ما نمیگذارد و همانطور که مشاهده شد برای این مثال عدد ۹ را برای جواب گزارش میکند.




hw1.py M
test.py M
hw1.py > ...
h = 0
for i in range(n):
for j in range(n):
x_dif = abs(j - target_place[state[i][j]][1])
y_dif = abs(i - target_place[state[i][j]][0])
if (x_dif > n//2):
x_dif = n - x_dif
if (y_dif > n//2):
y_dif = n - y_dif
h = h + x_dif * 5 + y_dif * 5
return 0
def A_star_F(node,target_place):
return node.cost + heuristic(node.state,target_place)
def find_target_places(target):
target_places = {}
for i in range(len(target)):
for j in range(len(target)):
target_places[target[i][j]] = (i,j)
return target_places
class Frontier():
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER: VARIABLES
3
6 5 7
1 3 2
8 4 0
5 1 4
3 0 2
7 8 6
6
down 3
right 1
down 3
left 3
right 2
down 3

و در انتها نتیجه اجرای مثال آخر را با دو الگوریتم A_star و BFS در سه تصویر مشاهده خواهید کرد. که مجددا مشخص میکند که به چه اندازه الگوریتم A_star سریع تر خواهد بود و همچنین بهینه نبودن جواب آن را به نمایش میگذارد. در اجرای برنامه برای مثال آخر الگوریتم A_star کمتر از ۱۰ ثانیه و الگوریتم BFS بیش از ۵ دقیقه زمان صرف شد. که این موضوع از تعداد راس های بازدید شده توسط هر یک از آنها کاملا مشخص است.




RUN AND DEBUG
No Configurations
VARIABLES
Locals
adjacent: ('up 4', ((...), (...), (...), (...)))
adjacent_nodes: [('right 1', (...)), ('right 2', …
frontier: <__main__.Frontier object at 0x7ff5432c…
nextNode: <__main__.Node object at 0x7ff5238f1cc0>
shortest_path_list: [('left 2', (...)), ('right 3…
source: ((12, 11, 9, 2), (8, 1, 4, 5), (7, 15, 0,…
source_node: <__main__.Node object at 0x7ff5432c4…
target: ((12, 3, 14, 2), (1, 11, 9, 8), (0, 4, 5,…
WATCH
len(frontier.states): 1826978
CALL STACK
Paused on breakpoint
Debug hw1.py 2
PAUSED ON BREAKPOINT
path
hw1.py 161:1
<module>
hw1.py 182:1
hw1.py M
test.py M
Debug hw1.py 2
hw1.py > heuristic
targetNode = targetNode.parent
shortest_path_list.reverse()
return shortest_path_list
def fast_input():
input = io.BytesIO(os.read(0, \
os.fstat(0).st_size)).readline
Taking input as string
s = input().decode()
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER: VARIABLES
amir@amir-Nitro-AN515-55:~/Documents/un/AI/hw1/practical$ /usr/bin/e
s-python.python-2022.16.1/pythonFiles/lib/python/debugpy/adapter/../.
s/un/AI/hw1/practical/hw1.py
4
12 3 14 2
1 11 9 8
0 4 5 15
13 10 7 6
12 11 9 2
8 1 4 5
7 15 0 10
13 3 14 6

```python
def heuristic(state,target_place):
    n = len(state)
    h = 0
    for i in range(n):
        for j in range(n):
            x_dif = abs(j - target_place[state[i][j]][1])
            y_dif = abs(i - target_place[state[i][j]][0])
            if (x_dif > n//2 ):
                x_dif = n - x_dif
            if (y_dif > n//2):
                y_dif = n - y_dif
            h = h +  x_dif * 5  + y_dif * 5
    return 0

def A_star_F(node,target_place):
    return node.cost + heuristic(node.state,target_place)

def find_target_places(target):
    target_places = {}
    for i in range(len(target)):
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER: VARIABLES

```
s/un/AI/hw1/practical/hw1.py
4
12 3 14 2
1 11 9 8
0 4 5 15
13 10 7 6
12 11 9 2
8 1 4 5
7 15 0 10
13 3 14 6
5
left 2
right 3
right 3
down 3
down 2
amir@amir-Nitro-AN515-55:~/Documents/un/AI/hw1/practical$
```