سوال عملی ۱

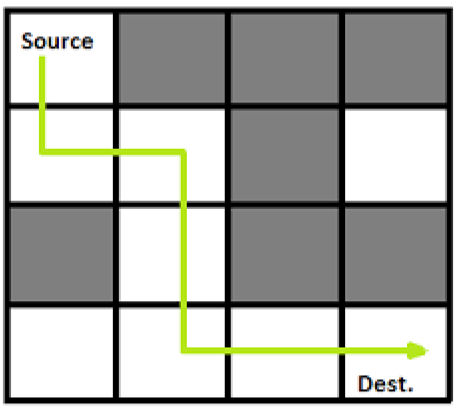
```
        new=(node[0]+1,node[1])
        if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N and playGround[new[0]][new[1]] == 0:
            g[new]=g[node]+1
            if new not in closed and (f(new,goal,g),new) not in frontier:
                heapq.heappush(frontier,(f(new,goal,g),new))
                path[new]=node

        #upper neighbor
        new=(node[0],node[1]+1)
        if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N and playGround[new[0]][new[1]] == 0:
            g[new]=g[node]+1
            if new not in closed and (f(new,goal,g),new) not in frontier:
                heapq.heappush(frontier,(f(new,goal,g),new))
                path[new]=node

        #lower neighbor
        new=(node[0],node[1]-1)
        if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N and playGround[new[0]][new[1]] == 0:
            g[new]=g[node]+1
            if new not in closed and (f(new,goal,g),new) not in frontier:
                heapq.heappush(frontier,(f(new,goal,g),new))
                path[new]=node



A_star_for_maze()
```

[14] ✓ 28.9s

```
The cost of the optimal path is:  6
Optimal path:
go to house  (1, 0)
go to house  (1, 1)
go to house  (2, 1)
go to house  (3, 1)
go to house  (3, 2)
go to house  (3, 3)
```

خروجی بالا مربوط به زمین بازی زیر است

در سوال از هیوریستیک منهتن استفاده شده است

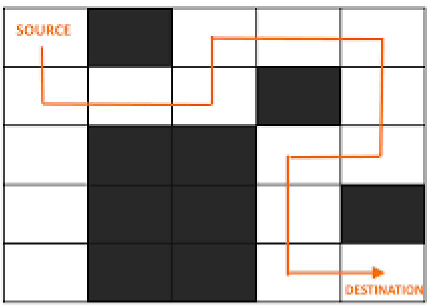


در تصاویر زیرخروجی الگوریتم به ازای ورودی دیگری آمده است

```python
            #upper neighbor
            new=(node[0],node[1]+1)
            if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N and playGround[new[0]][new[1]] == 0:
                g[new]=g[node]+1
                if new not in closed and (f(new,goal,g),new) not in frontier:
                    heapq.heappush(frontier,(f(new,goal,g),new))
                    path[new]=node

            #lower neighbor
            new=(node[0],node[1]-1)
            if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N and playGround[new[0]][new[1]] == 0:
                g[new]=g[node]+1
                if new not in closed and (f(new,goal,g),new) not in frontier:
                    heapq.heappush(frontier,(f(new,goal,g),new))
                    path[new]=node



    A_star_for_maze()
```

[2] ✓ 45.4s

```
The cost of the optimal path is:  12
Optimal path:
go to house  (1, 0)
go to house  (1, 1)
go to house  (1, 2)
go to house  (0, 2)
go to house  (0, 3)
go to house  (0, 4)
go to house  (1, 4)
go to house  (2, 4)
go to house  (2, 3)
go to house  (3, 3)
go to house  (4, 3)
go to house  (4, 4)
```



سوال عملی ۲

وقتی از هیوریستیک منهتن استفاده می کنیم نسبت به وقتی که از هیوریستیک فاصله اقلیدسی استفاده می کنیم زود تر به جواب میرسیم

در زیر خروجی الگوریتم برای نمونه زیر با دو هیوریستیک فوق آمده است

```python
            new=(coordinates_of_0[0][0],coordinates_of_0[0][1]-1)
            if new[0] >= 0 and new[1] >= 0 and new[0] < N and new[1] < N:
                new_puzzle=np.copy(node)
                new_puzzle[coordinates_of_0[0][0]][coordinates_of_0[0][1]],new_puzzle[new[0]][new[1]]=new_puzzle[new
                g[tuple(new_puzzle.flatten())]=g[tuple(node.flatten())]+1
                if tuple(new_puzzle.flatten()) not in closed and (f(new_puzzle,goalPuzzle,g,flag,N),tuple(new_puzzle
                    heapq.heappush(frontier,(f(new_puzzle,goalPuzzle,g,flag,N),tuple(new_puzzle.flatten())))
                    path[tuple(new_puzzle.flatten())]=node




    A_star_for_N_Puzzle()
```

[2] ✓ 38.9s

```
We use Manhattan distance heuristic.
Number of steps required to get from staring puzzle to goal puzzle:  31
Path from starting puzzle to goal puzzle:
change puzzle to  [[0. 8. 6.]
 [5. 4. 7.]
 [2. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [0. 4. 7.]
 [2. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 7.]
 [0. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 7.]
```

زمان اجرا با استفاده از هیوریستیک منهتن ۳۸ ثانیه بوده است

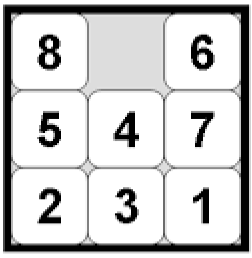
```
A_star_for_N_Puzzle()
```

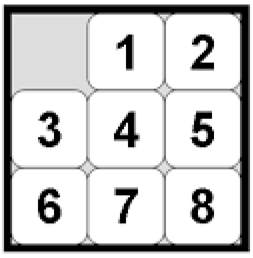
[3] ✓ 3m 7.9s

```
We use Euclidean distance heuristic.
Number of steps required to get from staring puzzle to goal puzzle is:  31.0
Path from starting puzzle to goal puzzle:
change puzzle to  [[0. 8. 6.]
 [5. 4. 7.]
 [2. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [0. 4. 7.]
 [2. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 7.]
 [0. 3. 1.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 7.]
 [3. 0. 1.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 7.]
 [3. 1. 0.]]
change puzzle to  [[5. 8. 6.]
 [2. 4. 0.]
 [3. 1. 7.]]
change puzzle to  [[5. 8. 0.]
 [2. 4. 6.]
 [3. 1. 7.]]
change puzzle to  [[5. 0. 8.]
...
 [6. 7. 8.]]
change puzzle to  [[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

زمان اجرا با استفاده از هیوریستیک فاصله اقلیدسی حدود ۳ دقیقه است

مسئله





سوال عملی ۳
در این سوال برای تولید پترن دیتابیس از پازل نهایی شروع کرده و همه ی عناصر به جز ۰ و اعدادی که میخواهیم در پترن دیتابیس استفاده کنیم را برابر −۱ قرار می دهیم و سپس با شروع از پازل به دست امده هر بار خانه ی ۰ را با یکی از خانه های دیگر جا به جا می کنیم و در صورتی که در دیتا بیس نباشد در آن به همراه تعداد پازل هایی که با شروع از پازل نهایی طی کرده ایم تا به این پازل خاص برسیا ذخیره می کنیم
به این شیوه پترن دیتابیس را می سازیم و سپس از آن به عنوان یک هیوریستیک admissible استفاده می کنیم
این روش به حافظه بیشتری برای ذخیره دیتابیس نیاز دارد ولی از انجایی که فاصله یک پازل تا پازل نهایی را دقیق تر از هیوریستیک های منهتن و فاصله اقلیدسی محاسبه میکند باعث گسترش تعداد کمتری از نود ها شده و سریع تر مراحل رسیدن از پازل داده شده به پازل نهایی را پیدا می کند