Yasaman Mirmohammad 9431022

به نام خدا

پروژه ی برنامه نویسی شماره ی ۱

درس هوش مصنوعی

---

**در تمام مساله ها**:

مدل کردن مسائل :

۱. تابع :initiate : مقدار start و goal را به روزرسانی میکند.
۲. تابع action : مجموعه ی حرکت های موجود برای این پازل r d u l است که بر اساس جایگاه جای خالی موجود در پازل حرکت های ممکن بررسی میشوند. (تقریبا مانند مثال قبل)
۳. تابعfinalAction: نتیجه ی حاصل از هر اکشن پیاده سازی شده است.
این تابع با گرفتن استیت مشخص میکند که با هر حرکت ممکن در این استیت چه استیت های جدیدی ایجاد می شود.
۴. تابع costFunction : که طبق صورت سوال برای هر یال برابر یک است .
۵. تابع goalFunction : در صورت یکی بودن استیت فعلی با استیت goal ، true بر میگرداند .
۶. تابع heuristic : فاصله ی مستقیم هر عدد در این استیت تا جای اصلی ای که باید قرار داشته باشد
در نهایت مقدار تابع h برابر مجموع این مقادیر قرار داده میشود.. .

**۱.روبات مسیر یاب**

همه ی الگوریتم های dfs (در ۳ مدل) ، bfs ، astar ، ucs و bidirectional پیاده سازی شده اند.

Mem نشان دهنده ی زمانی ست که تعداد نود ها ی ذخیره شده در f و e بیشترین مقدار خود را دارد.

e : نود های گسترش یافته

visited : نود های دیده شده.

path: بهترین مسیر یافته شده.

بیشتری مصرف حافظه برای bidirectionalاست چون دو جست و جو را همزمان جلو می برد پس بیشتری حافظه لازم دارد..
کمترین حافظه مربوط به اول عمق است چون بین بقیه ی الگوریتم ها از حافظه ی کمتری استفاده میکند .

ورودی:

```
5,5,4
```

.

```
3,2,4,2
3,3,4,3
2,3,2,4
3,3,3,4
```

**Astar**:

act ['r', 'd', 'r', 'd']
max memory 10

**uniform cost Search**:

act ['r', 'd', 'r', 'd']
max memory 8

**dfs unlimited:**
act ['r', 'd', 'r', 'd']
max memory 5

**bidirectional search**:

act ['d', 'r', 'r', 'd']
max memory 14

**۲..پازل هشت تایی.**

8 puzzle:

8-Puzzle is an interesting game which requires a player to move blocks one at a time to solve a picture or a particular pattern

Steps:

- Get the current state (refers to the board or game in real world).
- Find the available moves and their cost.
- Choose the move with the least cost and set it as the current state.
- Check if it matches the goal state, if yes terminate, if no move to step 1.

Suppose the program is executed for breadth-first search starting from the initial state *1,2,3,,4,5,6,7,8,0* as follows:

ورودی:

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

خروجی:

**Astar**:

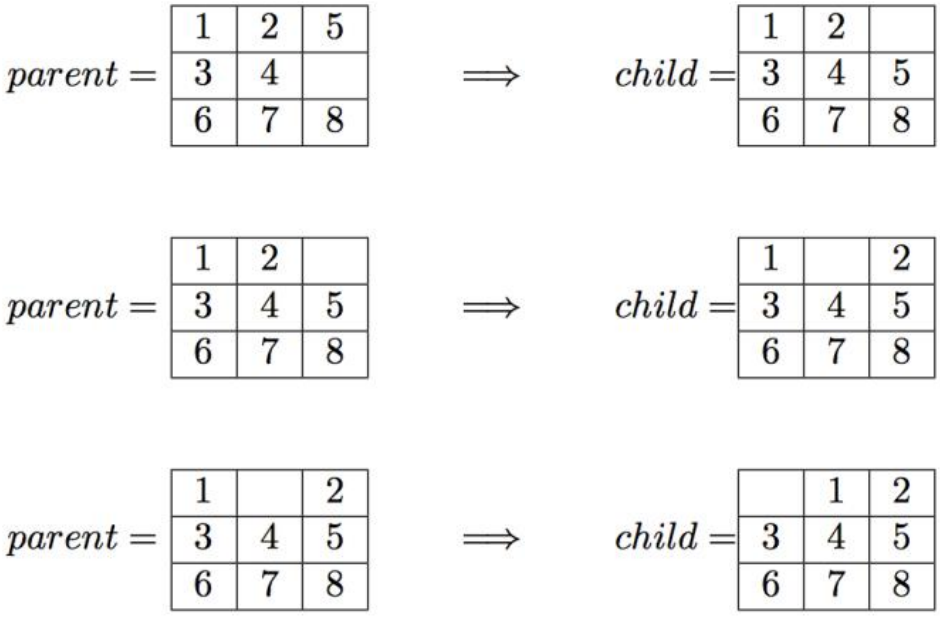
path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
visited [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
act ['l']
max memory 5

**uniform cost Search**
path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
visited [[1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
act ['l']
max memory 5

**bidirectional search**:

path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
visited [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
act ['l']
max memory 7

:



The output file should contain exactly the following lines:

path_to_goal: [‘Up’, ‘Left’, ‘Left’]
cost_of_path: 3
nodes_expanded: 10
fringe_size: 11
max_fringe_size: 12
search_depth: 3
max_search_depth: 4

running_time: 0.00188088
max_ram_usage: 0.07812500

مشکل اصلی این سوال این بود که چون تعداد حالات به وجود آمده برای بعضی مثال ها خیلی زیاد میشد stack over flow رخ می داد.

نتیجه:

بدترین عملکرد مربوط به dfs است که به خاطر نوع عملکرد الگوریتم است که ابتدا عمق را بررسی می کند.همچنین بیشترین حافظه نیز برای dfs است پس این الگوریتم برای حل این سوال مناسب نمی باشد.

کمترین حافظه های مصرفی : ucsو astar

از منظر نود های ویزیت شده و زمان، بقیه ی الگوریتم ها تقریبا در یک سطح قرار دارند. .

این سوال توسط چهار الگوریتم بررسی و حل شد.

uniform cost Search

bidirectional search
dfs
A*

**۳.:روبیک**

dfs جواب بهتری دارد ولی این نکته وجود دارد که در این سوال جواب نهاییR است

در بعضی موارد stack over flow نیز رخ میدهد که به دلیل حجم بالای پردازش dfs است .

به طور کلی برای این سوال همان bfs جواب بهتری پیدا میکند تا dfs .

ورودی:

```
['y', 'b', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'r', 'r', 'r', 'o', 'o', 'o',
 'o'],
['b', 'b', 'b', 'b', 'y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'r', 'r', 'r', 'r', 'o', 'o', 'o',
 'o'])
```

خروجی:

```
1
finish
2
iterative dfs limited
path : [['y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['y', 'b', '
max memory 1
visited []
expand []
act ['R']
```