BFS IMPLEMENTATION

FUNCDTIONS:

{check_region,goal_test,make_children,bfs}

قبل از توضیح دادن کارایی هرتابع گفتن یک سری مقدمه ملزوم میبشاد.

۱)اول از همه یک کلاس نود تعریف کرده ام که دارای یک مقدار میباشد و مقدار یک لیست است که در هر خانه آن شکل مناطق نشان داده شده است

برای مثال:

2g 3r 5g 1b

نمونه ای از منطقه میباشد.

۲)تغییری در ساختار جستجوی اول سطح داده ام آن هم این است که فرانتیر و کشف شده را خارج از آن تعریف کرده ام تا بتوانم طول آن هارا در پایان برنامه محاسبه کنم

Check_region(string):

کاربرد این تابع به این صورت است که یک منطقه را به عنوان ورودی میگیرد و چک میکند که ایا این منطقه با توجه به محدودیت های مسیله قابل قبول است یا خیر.به این صورت عمل میکند که اگر اعداد به صورت نزولی مرتب شده بود و همه رنگ ها یکسان بود این منطقه قابل قبول میشود.

Goal_test(node):

این تابع تک تک مناطق داخل نود را بررسی میکند اگر همه مناطق قابل قبول بودند درست برمیگرداند در غیر این صورت غلط برمیگرداند

Make_children(node):

این تابع تمامی فرزندان یک نود را تولید میکند به طوری که تمامی شرایط و محدودیت های مسیله نقض نشود و در نهایت یک لیست از فرزندان برمیگرداند.

Bfs():

این تابع دقیقا همانند تدریس استاد پیاده سازی شده است و تنها تفاوتی که دارد این است که مجموعه فرانتیر و کاوش شده خارج از تابع تعریف شده اند.به این دلیل که بتوان طول آن ها را در پایان برنامه محاسبه کرد.

نمونه ورودی که برای مسیله داده ام:

نمونه ۱

5 3 5

5b 4b 1b 2b 3b

5g 4g 3g 2g 1g

#

5r

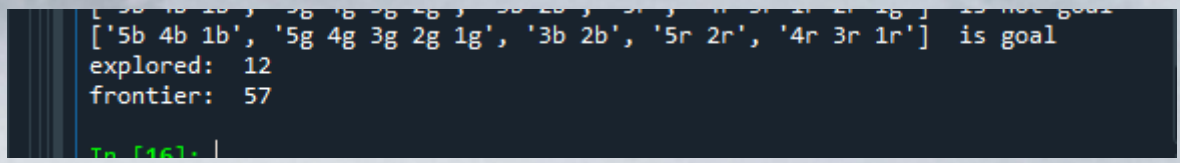
4r 3r 1r 2r

Out put is:

['5b 4b 1b', '5g 4g 3g 2g 1g', '3b 2b', '5r 2r', '4r 3r 1r'] is goal

explored: 12

frontier: 57



نمونه ۲

5 3 5

5b 4b 1b 3b 2b

5g 4g 3g 2g 1g

#

3r

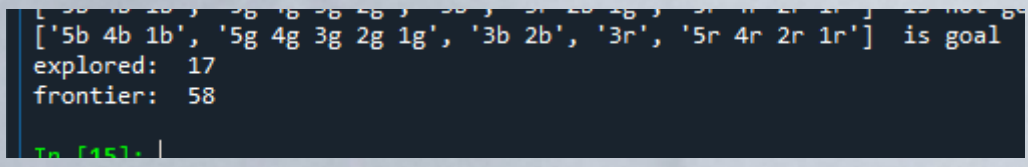
5r 4r 2r 1r

Out put is:

['5b 4b 1b', '5g 4g 3g 2g 1g', '3b 2b', '3r', '5r 4r 2r 1r'] is goal

explored: 17

frontier: 58



نمونه ۳:

5 3 5

5g 4g 1g 3g 2g

#

5r 4r 3r 1r 2r

3b

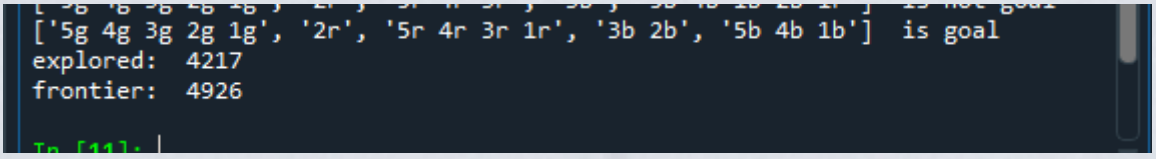
5b 4b 1b 2b

Out put is:

['5g 4g 3g 2g 1g', '2r', '5r 4r 3r 1r', '3b 2b', '5b 4b 1b'] is goal

explored: 4217

frontier: 4926



IDS IMPLEMENTATION

FUNCDTIONS:

{check_region,goal_test,make_children,recursive_dls,depth_limited_search,iterative_deepening_search}

توابع و کلاس های مشترک با تمرین اول دقیقا همان کارایی را دارند

حال باید فقط توابع جدید و کارایی آن ها را بررسی کرد.

Recursive_dls(myNode,limit):

این تابع همانند شبه کدی که در کلاس تدریس شده است عملکرد دارد البته برای کارا تر شدن یکسری تغییراتی اعمال کردم.و تعداد نود های تولید شده را نیز در آن تابع محاسبه کردم.به این صورت که تاع به صورت بازگشتی تا عمق خواسته شده را بررسی میکند.

Depth_limited_search(limit):

این تابع همان کارایی تابع قبل را دارد فقط لیمیت را به آن میدهیم و تابع قبل را برمیگرداند.

Iterative_deeping_search():

این تابع از عمق صفر تا بینهایت را به ترتیب بررسی میکند و در هر عمقی که به جواب برسد توقف میکند.

نمونه ورودی ها(همانند سوال اول) و خروجی ها

ورودی و خروجی ۱:

```
enter n,m,k: 5 3 5

enter region1: 5b 4b 1b 2b 3b

enter region2: 5g 4g 3g 2g 1g

enter region3: #

enter region4: 5r

enter region5: 4r 3r 1r 2r
depth: 3
['5b 4b 1b', '5g 4g 3g 2g 1g', '3b 2b', '5r  2r', '4r 3r 1r']
number of generated nodes: 123
```

نمونه ورودی و خروجی ۲:

```
enter n,m,k: 5 3 5

enter region1: 5b 4b 1b 3b 2b

enter region2: 5g 4g 3g 2g 1g

enter region3: #

enter region4: 3r

enter region5: 5r 4r 2r 1r
depth: 3
['5b 4b 1b', '5g 4g 3g 2g 1g', '3b 2b', '3r', '5r 4r 2r 1r']
number of generated nodes: 177

In [17]:
```

نمونه ورودی و خروجی ۳:

```
enter n,m,k: 5 3 5

enter region1: 5g 4g 1g 3g 2g

enter region2: #

enter region3: 5r 4r 3r 1r 2r

enter region4: 3b

enter region5: 5b 4b 1b 2b
depth: 8
['5g 4g 3g 2g 1g', '2r', '5r 4r 3r 1r', '3b 2b', '5b 4b 1b']
number of generated nodes: 3185166
```

توجه شود که اگر دارای جواب باشیم الگوریتم جواب را برمیگرداند در غیر اینصورت نال را برمیگرداند(اگر تا عمق خاصی مد نظر باشد).البته توجه شود که اگر الگوریتم تا بینهایت رود و جواب نداشته باشد هیچ وقت پایان نمیابد و جوابی برگردانده نمیشود.

A* IMPLEMENTATION

{check_region,goal_test,make_children,Astar,region_heuristic,node_heuristic,distance_calc}

قسمت ۳ مشابه قسمت های قبلی پیاده سازی شده است اما با تفاوت های اندکی که بتوان آن را تبدیل به الگوریتم مورد نظر کرد اول از همه به کلاس نود پدر و هیوریستیک و فاصله طی شده را اضافه کردم.

مبنای کار هیوریستیک به این شکل است که مجموع تعداد تناقض های موجود در آرایه صعودی را با هم جمع میکند و چنانچه نود هدف باشد هیوریستیک ما برابر صفر میشود.

به این صورت کار میکند که از اول ارایه شروع کرده و هرخانه را با خانه یکی بعدیش مقایسه میکند اگر خانه یکی بعدیش بیشتر بود به عدد هیوریستیک اضافه میشود.

و برای این که الگوریتم ما الگوریتم آگاهانه مورد نظر باشد برای مقایسه یک نود با دیگری از مجموع هیوریستیک و مسیر پیموده شده استفاده میکنیم.مسیر پیموده شده به این صورت است که تعداد نود های والد یک نود را تا جایی میشماریم که به نال برسیم.

توابع مشترک همانند حالت قبل است فقط تابع بچه ساز پدر را نیز مشخص میکند برای هر نود

region_heuristic(string):

در این بخش یک رشته (یک منطقه) را میگیریم و هیوریستیک را روی آن محاسبه میکنیم.به این شکل که هر جا عدد سمت راستی از عدد سمت چپیش بیشتر بود یکی اضافه میکنیم به مجموع.

Node_heuristic(string):

در این بخش یک نود را میگیریم و هیوریستیک را برابر مجموع هیوریستیک های روی روی مناطق آن حساب میکنیم.

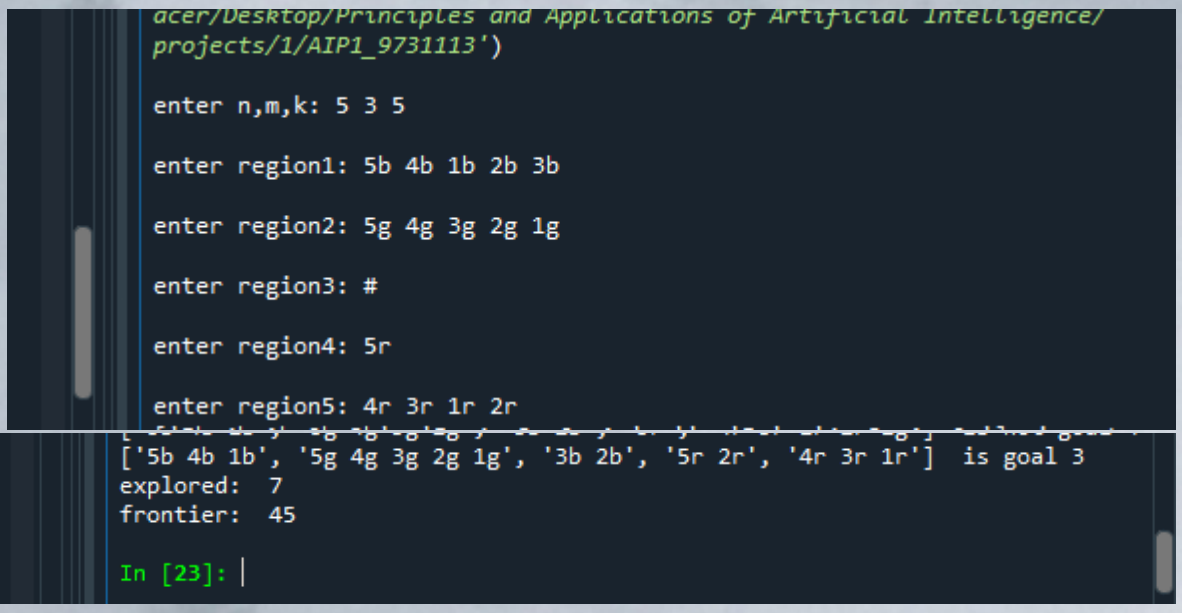
Distance_calc(node):

در این بخش مجموع نود های پدر و نوادگان یک نود را حساب میکنیم تا وقتی که به نال برسیم.

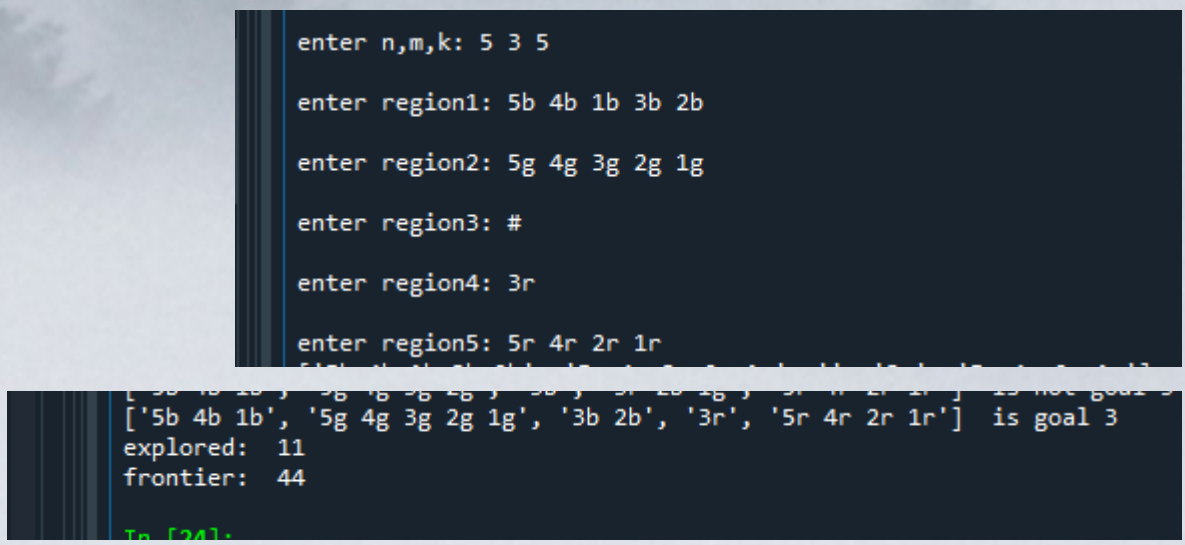
Astar(node):

همانند جستجوی اول سطح است با این تفاوت که از یک صف اولویت برحسب مجموع هیوریستیک و مسیر استفاده میکند.

نمونه ورودی و خروجی ۱:



نمونه ورودی و خروجی ۲:



نمونه ورودی و خروجی ۳:

5 3 5

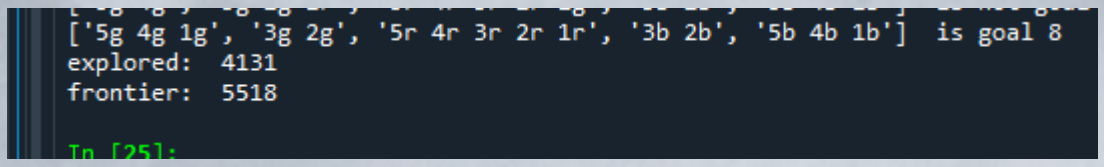
5g 4g 1g 3g 2g

#

5r 4r 3r 1r 2r

3b

5b 4b 1b 2b



توجه شود که جستجوی آگاهانه تعداد نود های کمتری را کاوش کرد.

تمامی کد های سوال ۱و۲و۳ کامنت گزاری شده اند در صورت هرگونه ابهام در کد از یکی از طرق زیر میتوانید با بنده در تماس باشید:

Telegram id:@mcaptain79

Gmail:meymani79@gmail.com