به نام خدا

علی عزیزی - 9731042

پروژه اول درس هوش مصنوعی

در ابتدا به توضیح کلاس ها و توابعی که در هر سه فایل مشترک هستند پرداخته می شود و سپس هر کدام از سه الگوریتم خواسته شده به تفصیل بیان شده و در نهایت مقایسه ای میان سه الگوریتم از نظر زمان اجرا، تعداد گره های تولید شده و بسط داده شده، انجام می شود.

کلاس Card :

در این کلاس یک شی کارت شبیه سازی می شود که دارای دو مشخصه ی رنگ و شماره کارت می باشد.

کلاس Node :

در این کلاس یک گره از گراف فضای حالت که قرار است جستجو در آن انجام شود شبیه سازی شده که هر گره دارای مشخصه های زیر است:

عمق ، گره پدر، یک لیست که هر عنصر آن لیستی از کارت ها می باشد، اکشن انجام شده برای رسیدن به وضعیت مذکور و مقدار f_value می باشد.

اکنون به شرح توابع مهم در کد تحویلی می پردازیم.

تابع read_input :

در این تابع ورودی های مسئله مطابق با فرمت گفته شده از یک فایل متنی خوانده می شود و گره اولیه ای با این مشخصات ساخته و آن را به عنوان خروجی بر می گردانیم.

تابع check_goal :

این تابع که یک گره به عنوان ورودی می گیرد با کمک از تابع check_arr_of_cards بررسی می کند که آیا ورودی یک وضعیت هدف می باشد یا خیر. وضعیتی هم به عنوان وضعیت هدف پذیرفته می شود که در آن همه کارت های هر ستون هم رنگ بوده و از نظر عدد روی کارت ترتیب نزولی داشته باشند.

تابع generate_children :

در این تابع همه ی وضعیت هایی که با انجام اعمال مجاز می توان به آن ها منتقل شده محاسبه شده و در قالب یک لیست برگردانده می شود.در این تابع از تابع find_move استفاده شده که این تابع action های مجاز را در قالب لیستی از Tuple ها بر می گرداند.

تابع print_path:

در این تابع که ورودی آن یک حالت هدف می باشد با استفاده از مشخصه ی last_move که در هر گره وجود دارد مسیر رسیدن از وضعیت اولیه به وضعیت هدف مذکور چاپ می شود.

الگوریتم BFS :

در این الگوریتم که از نسخه گرافی آن استفاده شده و آزمون هدف در آن هنگام تولید نود انجام می شود و همچنین برای پیاده سازی frontier در آن از یک queue استفاده شده، مطابق شبه کد بیان شده در اسلاید ها، ابتدا آزمون هدف انجام می شود و در صورت نبودن هدف، به لیست frontier اضافه می شود و در یک حلقه ی بی نهایت که شرط اتمام آن خالی بودن frontier است، کارهای زیر انجام می شود:

یک عنصر از صف pop میشود.

همه ی فرزندان آن تولید میشود.

آزمون هدف بر روی آن ها انجام میشود.

در صورت نبودن هدف و نبودن در لیست های frontier و explored به صف اضافه می شود.

```
def bfs(init_node: Node):
    if check_goal(init_node):
        return init_node
    frontier = [init_node]
    explored = []
    while True:
        if not len(frontier):
            return None
        node = frontier.pop(0)
        global expanded
        expanded = expanded + 1
        explored.append(node.board_game)
        children = generate_children(node)
        for child in children:
            global created
            created = created + 1
            if child.board_game not in explored and not is_in_frontier(child, frontier):
                if check_goal(child):
                    return child
                frontier.append(child)
```

الگوریتم IDS :

در این الگوریتم، الگوریتم جستجوی اول عمق محدود شده در یک for از عمق اولیه تعیین شده تا عدد بزرگی مانند 1000، فراخوانده می شود و در جستجوی DLS هم که یک گره و یک عمق limit را به عنوان ورودی می گیرد، مطابق کد زیر با فراخوانی بازگشتی خود به جستجوی حالت هدف در گراف فضای حالت می پردازد.

```
def dls(node: Node, limit):
    if check_goal(node):
        return node, True
    if limit == 0:
        return None, True
    global created
    global expanded
    expanded = expanded + 1
    if limit > 0:
        cutoff_occurred = False
        for child in generate_children(node):
            created = created + 1
            found, cutoff = dls(child, limit - 1)
            if found:
                return found, True
            if cutoff:
                cutoff_occurred = True
        return None, cutoff_occurred

def ids(root: Node, base_levels: int = 0):
    for depth in range(base_levels, 1000):
        found, cutoff = dls(root, depth)
        if found:
            return found
        elif not cutoff:
            return None
```

الگوریتم A_Star :

در این الگوریتم برای frontier از یک priority queue استفاده شده است که مطابق با تابع f (که در ادامه توضیح داده می شود)، لیست frontier را مرتب می کند و پس از مرتب سازی عنصر اول لیست مذکور pop میشود.

تابع f به صورت روبرو است:

```
g(node) + max(h2(node), h1(node))
```

که تابع g متناظر با عمق هر گره است و تابع h1 و h2 و علت admissible بودن آن ها به شرح زیر است:

هیوریستیک h1 :

در این هیوریستیک برای ریلکس کردن مسئله، رنگ کارت ها را در نظر نمی گیریم و فرض می کنیم کارت ها در یک ستون می توانند جا به جا شوند. جواب این مسئله ریلکس شده را برای همه ی ستون ها به دست می آوریم و پس از محاسبه مجموع آن ها، به عنوان هیوریستیک مسئله اصلی جواب را بر می گردانیم.

روش انجام این کار برای یک ستون نیز به این صورت است که در یک for با پیمایشگر i از صفر تا اندازه ستون، بررسی می کنیم که آیا عنصر i ام لیست ما، بزرگترین عضو زیر آرایه i تا طول لیست هست یا خیر؟ که در صورت منفی بودن جواب عنصر max را به نحوی در لیست منتقل می کنیم که اندیس آن i شود و شمارشگر خود که مقدار اولیه صفر دارد را یک واحد افزایش می دهیم و در انتها شمارشگر را برمی گردانیم و مجموع شمارشگر همه ی ستون ها برابر هیوریستیک h1 ما است.

برای اثبات قابل قبول بودن آن نیز چون در نهایت باید عناصر به صورت نزولی مرتب شود پس حداقل به اندازه نا به جایی ها نیاز به اکشن داریم، پس هیوریستیک ما قابل قبول است.

هیوریستیک h2 :

در این هیوریستیک برای هر ستون، به ازای هر دو کارت مجاور که رنگ یکسانی ندارند، یک واحد به متغیر s خود اضافه می کنیم و مقدار های حاصل از هر ستون را با هم جمع کرده و در نهایت به عنوان هیوریستیک بر می گردانیم.

و از آنجایی که در حالت هدف همه ی کارت های یک ستون می بایست هم رنگ باشند پس واضح است که برای هر ستون حداقل باید به اندازه s (در پاراگراف بالایی گفته شده) اکشن انجام شود، پس هیوریستیک گفته شده قابل قبول است.

هر دو هیوریستیک گفته شده سازگار نیز هستند، چون به ازای هر اکشن حداکثر مقدار افزایش h1 و h2 یک واحد است و چون g فرزند یک واحد بیشتر است، پس اخلالی در سازگاری ایجاد نمی شود.

از آنجایی که در برخی موارد h2 ، h1 را dominate می کند و در برخی موارد برعکس پس از max آنها استفاده می کنیم.

چون هر دو هیوریستیک h1 و h2 قابل قبول هستند، پس ماکسیمم آن ها نیز قابل قبول است.

مقایسه

برای حالت اولیه ای که از روی ورودی های زیر ساخته می شود، نتایج اجرای سه الگوریتم BFS، IDS و A* به صورت جدول زیر است:

5 3 5

1b 1r

5r 4r 3r 5g

5b 4b 3b 2b

4g 1g 2r 2g

3g

	BFS	IDS	A*
Created nodes	8241	206141	820
Expanded nodes	1038	24798	108
depth	6	6	6
Time (s)	3.83	51.15	0.25

```
-------------------------------- BFS --------------------------------
initial state :
1b 1r
5r 4r 3r 5g
5b 4b 3b 2b
4g 1g 2r 2g
3g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

goal state :
5g 2g
5r 4r 3r 2r 1r
5b 4b 3b 2b 1b
4g 1g
3g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

depth : 6
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Actions :
Add the last element of column 1 to the end of column 5 (1r)
Add the last element of column 1 to the end of column 3 (1b)
Add the last element of column 2 to the end of column 1 (5g)
Add the last element of column 4 to the end of column 1 (2g)
Add the last element of column 4 to the end of column 2 (2r)
Add the last element of column 5 to the end of column 2 (1r)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- created nodes = 8241 ---
--- expanded nodes = 1038 ---
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- 0 minutes and 3.8321001529693604 seconds ---
```

اجرای BFS

```
------------------------------- IDS --------------------------------
initial state :
1b 1r
5r 4r 3r 5g
5b 4b 3b 2b
4g 1g 2r 2g
3g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

goal state :
5g 2g
5r 4r 3r 2r 1r
5b 4b 3b 2b 1b
4g 1g
3g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

depth : 6
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Actions :
Add the last element of column 1 to the end of column 5 (1r)
Add the last element of column 1 to the end of column 3 (1b)
Add the last element of column 2 to the end of column 1 (5g)
Add the last element of column 4 to the end of column 1 (2g)
Add the last element of column 4 to the end of column 2 (2r)
Add the last element of column 5 to the end of column 2 (1r)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- created nodes = 206141 ---
--- expanded nodes = 24798 ---
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- 0 minutes and 51.15262985229492 seconds ---
```

اجرای IDS

```
------------------------------- A_Star --------------------------------
initial state :
1b 1r
5r 4r 3r 5g
5b 4b 3b 2b
4g 1g 2r 2g
3g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

goal state :
5g
5r 4r 3r 2r 1r
5b 4b 3b 2b 1b
4g 1g
3g 2g
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

depth : 6
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Actions :
Add the last element of column 4 to the end of column 5 (2g)
Add the last element of column 1 to the end of column 5 (1r)
Add the last element of column 1 to the end of column 3 (1b)
Add the last element of column 2 to the end of column 1 (5g)
Add the last element of column 4 to the end of column 2 (2r)
Add the last element of column 5 to the end of column 2 (1r)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- created nodes = 820 ---
--- expanded nodes = 108 ---
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

--- 0 minutes and 0.2500786781311035 seconds ---
```

اجرای A*