«به نام پروردگار»

گزارش پروژه اول مبانی هوش مصنوعی

«جستجوهای آگاهانه و ناآگاهانه در بازی simple spider solitaire»

استاد: دکتر بهنام روشنفکر

دانشجو: محسن محمدیان

شماره دانشجویی: 9831502

بررسی اجمالی:

این پروژه پیاده سازی سه نوع جستجو، که دوتا از آنها ناآگاهانه هستند؛ یعنی جستجوی BFS و IDS و یک جستجوی آگاهانه یعنی *A در فضای بازی کارت ها می باشد که به نوعی شبیه ساده شده ی بازی spider solitaire هست. این سه جستجو فضای حالت را شبیه درخت در می آورند و روی درخت جستجو می کنند تا به حالت هدف برسند. در هر سه جستجو تعداد نودهای expand شده و تولید شده و عمق رسیدن به وضعیت هدف (Goal state) مشخص شده که می توان این الگوریتم های جستجو را با این معیارها مقایسه کرد.

فرموله سازی مساله:

برای آنکه یک مساله را فرموله کنیم، باید حالت اولیه(initial state)، حالات (states)، اعمال (actions) را مشخص کنیم. هچنین باید آزمون هدف و هیزنه مسیر را درصورتی که مساله ما به آن نیاز داشته باشد مشخص کنیم.

حالت اولیه: در این مساله حالت اولیه همان محتوای موجود در فایل ورودی هست که یکسری کارت به صورت نامرتب و با رنگ های مختلف در ردیف های متفاوتی قرار گرفته اند.

حالات: تمامی حالاتی که در آن یکسری کارت به صورت مرتب یا نامرتب با رنگ های یکسان یا غیر یکسان در ردیف های متفاوتی قرار گرفته اند.

اعمال: برداشتن یک کارت از انتهای یک ردیف و قرار دادن آن در ردیفی دیگر، به شرطی که شماره کارت زیری اش بزرگتر از شماره کارت قرار داده شده باشد یا ردیف خالی باشد.

آزمون هدف: حالت هدف در این مساله وضعیتی خواهد بود که یکسری ردیف (به تعداد رنگ ها ی موجود) پر از کارت های همرنگ باشند و کارت های همرنگ نیز به ترتیب نزولی شماره کارت خود مرتب شده باشند.

هزینه مسیر: تعداد اعمال انجام شده تا رسیدن به وضعیت هدف می باشد.

شیوه پیاده سازی:

برای پیاده سازی هر کدام از الگوریتم ها در این پروژه، از دو فایل پایتون استفاده شده و از برنامه نویسی شی گرا بهره مند شده ایم. فایل های P1.py و P2.py و P3.py به ترتیب مربوط به الگوریتم BFS و IDS و A* می باشند که هر کدام از این فایل ها یک فایل همراه Child_node دارند که در آن کلاس Child را داریم و متغیرها و field هایی که یک نود فرزند می تواند داشته باشد، در آن کلاس تعریف شده است.

هنگامی که یک نود والد با انجام عمل بر روی وضعیت خود، به وضعیت جدیدی می رود، نود تازه ای تولید می شود و مقادیر موجود در شی child، مقداردهی (initialize) می شوند. در کلاس Child، ما متغیرهای زیر را داریم:

1. state : shows state of child node (type : <class list>)
2. parent: shows state of parent node of child node (type : <class list>)
3. parent_object: shows node (address) of parent of child node(type: <class Child_node_bfs.Child>
4. actions: shows actions of child node can do (type : <class list>)
5. action: shows action have done to child's parent state (type: <class str >)

همچنین دو متغیر دیگر هم داریم که برای پیاده سازی A* لازم هستند و در Child_node.py تعریف شده اند:

6. h: heuristic of state node (type: <class float>)
7. path_cost: shows cost of path to this node (type: <class int>)

همچنین یک فایل input.txt داریم که فضای حالت مساله در آن قرار دارد و هنگامی که فایل های P1, P2, P3 اجرا می شود، ورودی برنامه از فایل input.txt گرفته می شود و سپس فرآیند جستجو آغاز می شود.

توابع پیاده سازی شده مشترک در هر سه الگوریتم:

:main()

در این تابع ابتدا فایل input.txt خوانده می شود و محتویات آن در داخل متغیر text_input ریخته می شود و پس از آنکه این محتویات ردیف ها به کمک متغیر بالا در text_list ریخته شد، تابع breadth_first_search فراخوانی (call) می شود.

:actions(state)

این تابع actionهای مجاز برای وضعیت یک نود (state) را بر می گرداند. به این ترتیب که یک state را به عنوان ورودی می گیرد و ردیف ها خالی و غیر خالی آن را مشخص می کند و در ردیف هایی که در آن ها یکسری کارت قرار دارند مشخص می شود که چه کارتی از آن ردیف به چه ردیف های دیگری می تواند منتقل شود و برای هر ردیف این کار انجام می شود و در لیست action_list ریخته می شود و در آخر مجموع این actionها بر گردانده (return) می شود.

:result(curr_state, action, state_actions)

این تابع یک state و action را می گیرد و state جدید را بر می گرداند. (مشابه تعریف همان تابع result در اسلایدهای درس)

:goal_teste(state, ncolumns)

این تابع برای آن هست که مشخص کند یک state وضعیت هدف هست یا خیر و باتوجه به ویژگی هایی که یک وضعیت هدف می تواند داشته باشد، آن را بررسی می کند. ncolumns در ورودی تابع نیز تعداد کارت هایی است که هر رنگ می تواند داشته باشد و به کمک آن بررسی می شود که اگر حالتی وجود داشته باشد که تعداد کارت های موجود در ردیف آن به اندازه ی تعداد کارت هایی که باید داشته باشد، نباشد، آنرا هدف نگیرد و False برگرداند. (خروجی این تابع True یا False هست)

:solution(answer_node, expanded_nodes, generated_nodes, nrow)

در این تابع پاسخ مساله چاپ (print) می شود. سه ورودی دارد که اولی answer_node می باشد که یک شئ (Object) از Class type: Child هست و گره ی جواب و وضعیت هدف ما می باشد. چون در هر نود آدرس والدش نیز ذخیره می شود، با استفاده از این گره می توان مرحله به مرحله پیش رفت تا به ریشه رسید و به این طریق مسیر جواب بدست می آید.

همچنین چون action انجام شده روی والد یک نود نیز در آن ذخیره می شود همانطور که مرحله به رحله پیش می رویم، می توانیم action های انجام شده روی هر نود تا رسیدن به نود هدف را مشاهده کنیم.

این دو تابع دو ورودی expanded_node و generated_nodes نیز دارد که آنها را از تابع breadth_first_search دریافت می کند.

پیاده سازی الگوریتم BFS:

کد مربوط به این الگوریتم همانطور که پیش تر هم ذکر شد، در دو فایل پیاده سازی شده است؛ یکی فایل P1.py که در آن توابع مورد نیاز برای جستجو و فرموله سازی مساله پیاده شده و دیگری فایل Child_node_bfs.py که در این فایل یک کلاس Child قرار دارد که attribute هایی که یک گره (node) در درخت می تواند داشته باشد را مشخص می کند.

این الگوریتم نودها را سطر به سطر تولید و جستجو می کند و آزمون هدف را در هنگام تولید نود اعمال می کند به این صورت که اگر نود فرزند تولید شده هدف بود، جستجو متوقف می شود و همچنین چون از جستجوی گرافی بهره می بریم، نودهایی با حالت تکراری به صف frontier اضافه نخواهند شد.

توابع پیاده سازی شده در BFS:

:Breadth_first_search(initial_state, ncolumns)

در این تابع ابتدا state ابتدایی را می گیریم و سپس گره ی ریشه ی درخت را با استفاده از کلاس Child و new کردن یک object می سازیم؛ سپس بررسی می کنیم. اگر هدف بود تابع ()solution فراخوانی می شود و True برگردانده می شود؛ در غیر اینصورت ادامه داده می شود و لیست های frontier و explored و frontier_state تعریف و مقداردهی می شوند.

الگوریتم این تابع تا زمانی اجرا می شود که صف frontier ما که یک صف FIFO هست، تهی شود. در ابتدا یک نود از ابتدای frontier برداشته و حذف می شود و به explored اضافه می شود و سپس به ازای تمامی actionهای مجاز روی آن، نودهای فرزندش تولید می شوند.

هرکدام از فرزندان یک نود، اگر درگذشته بررسی نشده باشند و در لیست explored قرار نداشته باشند و همچنین در صف frontier موجود نباشند، بررسی می شوند. اگر هدف بودند، جستجو متوقف می شود و تابع solution فراخوانی و True برگردانده می شود. در غیر اینصورت به صف frontier اضافه می شوند.

نتیجه برخی از تست کِیس های داده شده در صفحه ی بعد آمده است.

```
5 3 5
5g 4g 3g 2g 1g
5r 4r 3r
5y 4y 3y 2r 1y
1r 2y
#
Depth of solution is:5
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r', '1y'], ['1r', '2y'], ['#']]
Action done to this state:  move card 1y from row3 to row5
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r'], ['1r', '2y'], ['1y']]
Action done to this state:  move card 2r from row3 to row2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y'], ['1r', '2y'], ['1y']]
Action done to this state:  move card 2y from row4 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y'], ['1r'], ['1y']]
Action done to this state:  move card 1r from row4 to row2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y'], ['#'], ['1y']]
Action done to this state:  move card 1y from row5 to row3
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:1037
Number of expanded nodes:402
```

```
5 3 5
5y 4y 3y 2y 1y
#
#
5r 4r 3r 2r 1r
5g 4g 3g 2g 1g
------------------------------------------------------------
Depth of solution is:0
Final State: [['5y', '4y', '3y', '2y', '1y'], ['#'], ['#'], ['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g']]
------------------------------------------------------------
Number of generated nodes:1
Number of expanded nodes:0
```

```
5 2 5
5r 4r 2r 1g
5g 4g 3g 1r
3r 2g
#
#
------------------------------------------------------------
Depth of solution is:7
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '1r'], ['3r', '2g'], ['#'], ['#']]
Action done to this state:  move card 1r from row2 to row4
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g'], ['3r', '2g'], ['1r'], ['#']]
Action done to this state:  move card 2g from row3 to row2
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '2g'], ['3r'], ['1r'], ['#']]
Action done to this state:  move card 1g from row1 to row2
------------------------------------------------------------
[['5r', '4r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['#']]
Action done to this state:  move card 2r from row1 to row5
------------------------------------------------------------
[['5r', '4r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['2r']]
Action done to this state:  move card 3r from row3 to row1
------------------------------------------------------------
[['5r', '4r', '3r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['2r']]
Action done to this state:  move card 2r from row5 to row1
------------------------------------------------------------
[['5r', '4r', '3r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['#']]
Action done to this state:  move card 1r from row4 to row1
------------------------------------------------------------
Final State: [['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:5814
Number of expanded nodes:2497
```

```
5 3 5
5g 4g 3g 2r 1g
5r 4r 3r
5y 4y 3y 2g 1y
#
1r 2y
------------------------------------------------------------
Depth of solution is:8
[['5g', '4g', '3g', '2r', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['#'], ['1r', '2y']]
Action done to this state:  move card 1g from row1 to row4
------------------------------------------------------------
[['5g', '4g', '3g', '2r'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 2r from row1 to row2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 1y from row3 to row2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2g'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 2g from row3 to row1
------------------------------------------------------------
[['5g', '4g', '3g', '2g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 1g from row4 to row1
----------------|-------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y'], ['#'], ['1r', '2y']]
Action done to this state:  move card 2y from row5 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2y'], ['#'], ['1r']]
Action done to this state:  move card 1y from row2 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['1r']]
Action done to this state:  move card 1r from row5 to row2
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['
------------------------------------------------------------
Number of generated nodes:5749
Number of expanded nodes:2903
```

پیاده سازی الگوریتم IDS:

کد مربوط به این الگوریتم، در دو فایل پیاده سازی شده است؛ یکی فایل P2.py که در آن توابع مورد نیاز برای جستجو و فرموله سازی مساله پیاده شده و دیگری فایل Child_node_bfs.py که در این فایل یک کلاس Child قرار دارد که attribute هایی که یک گره (node) در درخت می تواند داشته باشد را مشخص می کند.

این الگوریتم درواقع بهبودیافته ی الگوریتم DLS و DFS هست به اینصورت که مانند DLS درخت را به صورت اول عمق پیمایش می کند و وقتی به عمق cutoff رسید، به ریشه برگشته و عمق cutoff را یکی اضافه می کند و دوباره جستجو را از ریشه آغاز می کند. درواقع یک DLS تکراری می باشد که عمق محدود (limit) آن می تواند از صفر شروع شود و تا زمانی که فرصت دارد عمق را یکی اضافه کند و به جستجو به پردازد.

آزمون هدف در این الگوریتم در هنگام بسط نود اعمال می شود و همچنین چون از جستجوی درختی در این الگوریتم استفاده م کنیم، صف frontier و explored برای بررسی تکراری بودن نودها نداریم.

توابع پیاده سازی شده در IDS:

recursive_dls(node, limit, ncolumns, frontier, explored):

در این تابع ابتدا یک نود دریافت می شود و state آن بررسی می شود. اگر حالت هدف بود،تابع ()solution، فراخوانی می شود و سپس مقدار True توسط این تابع برگردانده می شود ولی اگر limit برابر با صفر بود، "Cutoff" برگردانده می شود و در غیراینصورت، به ازای تمامی action های مجاز رو آن وضعیت گره، فرزاندنش به ترتیب تولید می شوند؛ این تابع به صورت بازگشتی فراخوانی می شود و هربار یک عمل روی یک نود انجام می شود و فرزندی برای آن تولید می شود. اگر در فراخوانی بازگشتی یک نود به limit برابر صفر رسید، به پدرش برگشته و یک نود فرزند دیگر در تابع پدر بسط داده می شود.

تا زمانی عملیات بالا انجام می شود که همه ی نودهای فرزند ریشه تا عمق "Cutoff" بررسی شده باشند و در این صورت این تابع "Cutoff" بر می گرداند.

depth_limit_search(node, limit, ncol):

در این تابع نود ریشه (root_node) ساخته می شود و سپس تابع recursive_dls() فراخوانی می شود و مقدار برگردانده شده توسط آن تابع، توسط این تابع نیز برگردانده می شود.

iterative_deepening_search(start_state, ncol):

ورودی این تابع یک وضعیت شروع و تعداد کارت های هر رنگ می باشد. این تابع تازمانی که تابع depth_limit_search به جواب برسد و True برگرداند، آن تابع را فراخوانی می کند و تا عمق بی نهایت به جستجو ادامه می دهد.

یک متغیر cutoff در این تابع تعریف شده است که عمق اولیه برای جستجو را مشخص می کند. اگر این تابع به جواب رسید که هیچ وگرنه متغیر cutoff را یکی اضافه کرده و depth_limit_search را فراخوانی می کند.

نتیجه برخی از تست کِیس های داده شده در صفحه ی بعد آمده است.

```
5 3 5
5g 4g 3g 2g 1g
5r 4r 3r
5y 4y 3y 2r 1y
1r 2y
#
------------------------------------------------------------
Depth of solution is:5
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r', '1y'], ['1r', '2y'], ['#']]
Action done to this state:  move card 1y from row3 to row5
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r'], ['1r', '2y'], ['1y']]
Action done to this state:  move card 2r from row3 to row2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y'], ['1r', '2y'], ['1y']]
Action done to this state:  move card 2y from row4 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y'], ['1r'], ['1y']]
Action done to this state:  move card 1r from row4 to row2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y'], ['#'], ['1y']]
Action done to this state:  move card 1y from row5 to row3
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:31221
Number of explored nodes:3970
```

```
5 3 5
5y 4y 3y 2y 1y
#
#
5r 4r 3r 2r 1r
5g 4g 3g 2g 1g
------------------------------------------------------------
Depth of solution is:0
Final State: [['5y', '4y', '3y', '2y', '1y'], ['#'], ['#'], ['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g']]
------------------------------------------------------------
Number of generated nodes:1
Number of explored nodes:0

Process finished with exit code 0
```

```
1611778
1611786
Depth of solution is:7
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '1r'], ['3r', '2g'], ['#'], ['#']]
Action done to this state:  move card 1r from row2 to row4
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g'], ['3r', '2g'], ['1r'], ['#']]
Action done to this state:  move card 2g from row3 to row2
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '2g'], ['3r'], ['1r'], ['#']]
Action done to this state:  move card 1g from row1 to row2
------------------------------------------------------------
[['5r', '4r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['#']]
Action done to this state:  move card 2r from row1 to row5
------------------------------------------------------------
[['5r', '4r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['2r']]
Action done to this state:  move card 3r from row3 to row1
------------------------------------------------------------
[['5r', '4r', '3r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['2r']]
Action done to this state:  move card 2r from row5 to row1
------------------------------------------------------------
[['5r', '4r', '3r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['#']]
Action done to this state:  move card 1r from row4 to row1
------------------------------------------------------------
Final State: [['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:1825298
Number of explored nodes:213506
[['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['#'], ['#']]
1825298

Process finished with exit code 0
```

```
Depth of solution is:8
[['5g', '4g', '3g', '2r', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['#'], ['1r', '2y']]
Action done to this state:  move card 1g from row1 to row4
------------------------------------------------------------
[['5g', '4g', '3g', '2r'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 2r from row1 to row2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 1y from row3 to row2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2g'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 2g from row3 to row1
------------------------------------------------------------
[['5g', '4g', '3g', '2g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y'], ['1g'], ['1r', '2y']]
Action done to this state:  move card 1g from row4 to row1
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y'], ['#'], ['1r', '2y']]
Action done to this state:  move card 2y from row5 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2y'], ['#'], ['1r']]
Action done to this state:  move card 1y from row2 to row3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['1r']]
Action done to this state:  move card 1r from row5 to row2
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:2071950
Number of explored nodes:268995
```

پیاده سازی الگوریتم *A:

کد مربوط به این الگوریتم، در دو فایل پیاده سازی شده است؛ یکی فایل P2.py که در آن توابع مورد نیاز برای جستجو و فرموله سازی مساله پیاده شده و دیگری فایل Child_node.py که در این فایل یک کلاس Child قرار دارد که attribute هایی که یک گره (node) در درخت می تواند داشته باشد را مشخص می کند.

در این الگوریتم ما همه ی نود های فرزند یک نود را تولید کرده و از میان آنها آن نودی را بر می گیزینیم که هزینه مسیر به اضافه ی مقدار تابع هیوریستیک برای آن کمتر از سایر نودهای frontier باشد. در واقع در هر مرحله آن نودی انتخاب می شود که تابع f کمتری داشته باشد. (f(node) = g(node) + h(node))

آزمون هدف در این الگوریتم در هنگام بسط نود اعمال می شود و همچنین چون از جستجوی درختی در این الگوریتم استفاده می کنیم، صف frontier و explored برای بررسی تکراری بودن نودها نداریم.

توابع پیاده سازی شده در *A:

heuristic(state, ncolumns):

این تابع یک حالت یا state را می گیرد و یک مقدار برای آن حالت تعیین کرده و بر می گرداند. تابع هیوریستیک ما از سه قسمت یا ویژگی تشکیل می شود:

1. تعداد رنگ ها متفاوت در هر ردیف. چون درنهایت همه ی کارت های موجود در هر ردیف باید یکرنگ باشند و هر state ای که تعداد رنگ های متفاوت کمتری داشته باشد، می تواند بهتر باشد.
2. تعداد ردیف هایی که تعداد کارت های موجود در آن ها کمتر از تعداد کارت های هر رنگ هست. چون در نهایت تنها به ازای تعداد رنگ های مساله ما ردیف پر خواهیم داشت و سایر ردیف ها خالی خواهند ماند.
3. فاصله: فاصله ی هر کارت تا جایگاه اصلی اش؛ از نظر ترتیب نزولی شماره کارت ها. هر کارت یک شماره دارد که باید پس از یک کارت با شماره بزرگتر قرار گیرد.

$$heuristic = 0.6 \times distant + 1.4 \times ((number\ of\ different\ colors) + (number\ of\ not\ full\ and\ not\ empty\ rows))$$

درواقع برای طراحی هیوریستیک این مساله، از دو روش استفاده شده است. یکی روش تعدیل مساله که درنظر گرفته شده هر کارت را می توان روی هر کارت دیگری قرار داد و جابجا کرد تا به جواب رسید(همان ویژگی فاصله) یعنی هر کارت حداکثر می تواند با یک حرکت در جای درست خود قرار گیرد و از شرط هایی چون بزرگتر بودن شماره کارت قبلی در ردیف صرف نظر می شود. همچنین با استفاده از روش تعدیل مساله، مساله ی راحت دیگری طرح کردیم که تنها باید به تعداد رنگ کارت ها ردیف پر داشته باشد و در هر ردیف تمامی رنگ ها یکسان باشد.

دومین روش، روش یادگیری هیوریستیک با استفاده از تجربه هست که دو هیوریستیک بدست آمده در بالا را می آید باهم ترکیب می کند و به هرکدام یک ضریب اختصاص داده می شود؛ که ضرایب داده شده در کد این تابع، با استفاده از تجربه بدست آمده است.

برای اثبات قابل قبول بودن این هیوریستیک نیز میتوان گفت چون ما هر کارت را با حداکثر یک جابجایی در جای خود قرار می دهیم در حالی که در مساله اصلی اگر در state والد وضعیت هدف باشیم یک action نیاز هست انجام دهیم پس در مساله اصلی هزینه ی ما در هر state حداقل 1 خواهد شد.

همچنین اگر مثلا در یک ردیف کارت هایی با رنگهای مختلف وجود داشته باشد، هر کارت با حداکثر یک حرکت می تواند به ردیف مناسب خود منتقل شود. درصورتی که در مساله اصلی برای آنکه هر کارت به ردیف همرنگ خود منتقل شود، حداقل یک حرکت نیاز هست.

پس هرکدام از ویژگی های ما اگر به عنوان تکی به عنوان هیوریستیک مساله درنظر گرفته شوند، می توانند قابل قبول باشند.

حال وقتی دو ویژگی دیگر یعنی تعداد رنگ های مختلف و تعداد ردیف های نیمه پر را با این ویژگی جمع می کنیم، می تواند این مقدار حاصل جمع از هزینه ی اصلی بیشتر شود، بنابراین این جا هست که از روش یادیگری هیوریستیک با تجربه بهره می بریم و به صورت تجربی ضرایب مختلفی را روی یک مساله اعمال می کنیم تا به یک مقدار مشخص و ثابت برای سایر مسائل که کمتر از هزینه ی اصلی نیز هست برسیم.

توجه شود چون هرسه ی این ویژگی ها می توانند روی هم تاثیرگذار باشند، به هرکدام یک ضریبی را نسبت دادیم تا تعادل میان آنها برقرار شود.

:a_star(initial_state, ncolumns)

این تابع وظیفه ی جستجو را به کمک الگوریتم A* بر عهده دارد. در این تابع ابتدا بررسی می شود که نود ریشه هدف هست یا نه. اگر نبود، یک نود از ابتدای صف frontier حذف (pop) و برداشته می شود(که در اجرای اول این این تابع همان نود ریشه خواهد بود) سپس ازای تمامی actionهای مجاز برای یک نود، فرزندان آن تولید می شوند و تابع هیوریستیک برای هر کدام از آنها فراخوانی می شود تا هزینه ی رسیدن به هدف برای آن ها محاسبه شود سپس این هزینه با هزینه ی مسیر آن نود جمع می شود تا هزینه ی نهایی برای آن محاسبه شود.

پس اضافه شدن نود ها به صف frontier، این صف به صورت صعودی مرتب می شود تا در مرحله ی بعد، نودی با کمترین هزینه انتخاب شود.

نتیجه برخی از تست کِیس های داده شده در صفحه ی بعد آمده است.

```
5 3 5
5g 4g 3g 2g 1g
5r 4r 3r
5y 4y 3y 2r 1y
1r 2y
#
------------------------------------------------------------
Depth of solution is:5
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r', '1y'], ['1r', '2y'], ['#']]
Action done to this state: 3 to 5
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2r'], ['1r', '2y'], ['1y']]
Action done to this state: 3 to 2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y'], ['1r', '2y'], ['1y']]
Action done to this state: 4 to 3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y'], ['1r'], ['1y']]
Action done to this state: 4 to 2
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y'], ['#'], ['1y']]
Action done to this state: 5 to 3
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:266
Number of explored nodes:49

Process finished with exit code 0
```

```
5 3 5
5y 4y 3y 2y 1y
#
#
5r 4r 3r 2r 1r
5g 4g 3g 2g 1g
------------------------------------------------------------
Depth of solution is:0
Final State: [['5y', '4y', '3y', '2y', '1y'], ['#'], ['#'], ['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g']]
------------------------------------------------------------
Number of generated nodes:1
Number of explored nodes:0

Process finished with exit code 0
```

```
5 2 5
5r 4r 2r 1g
5g 4g 3g 1r
3r 2g
#
#
------------------------------------------------------------
Depth of solution is:7
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '1r'], ['3r', '2g'], ['#'], ['#']]
Action done to this state: 2 to 4
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g'], ['3r', '2g'], ['1r'], ['#']]
Action done to this state: 3 to 2
------------------------------------------------------------
[['5r', '4r', '2r', '1g'], ['5g', '4g', '3g', '2g'], ['3r'], ['1r'], ['#']]
Action done to this state: 1 to 2
------------------------------------------------------------
[['5r', '4r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['#']]
Action done to this state: 1 to 5
------------------------------------------------------------
[['5r', '4r'], ['5g', '4g', '3g', '2g', '1g'], ['3r'], ['1r'], ['2r']]
Action done to this state: 3 to 1
------------------------------------------------------------
[['5r', '4r', '3r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['2r']]
Action done to this state: 5 to 1
------------------------------------------------------------
[['5r', '4r', '3r', '2r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['1r'], ['#']]
Action done to this state: 4 to 1
------------------------------------------------------------
Final State: [['5r', '4r', '3r', '2r', '1r'], ['5g', '4g', '3g', '2g', '1g'], ['#'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:2342
Number of explored nodes:471

Process finished with exit code 0
```

```
5 3 5
5g 4g 3g 2r 1g
5r 4r 3r
5y 4y 3y 2g 1y
#
1r 2y
------------------------------------------------------------
Depth of solution is:8
[['5g', '4g', '3g', '2r', '1g'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['#'], ['1r', '2y']]
Action done to this state: 1 to 4
------------------------------------------------------------
[['5g', '4g', '3g', '2r'], ['5r', '4r', '3r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state: 1 to 2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2g', '1y'], ['1g'], ['1r', '2y']]
Action done to this state: 3 to 2
------------------------------------------------------------
[['5g', '4g', '3g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2g'], ['1g'], ['1r', '2y']]
Action done to this state: 3 to 1
------------------------------------------------------------
[['5g', '4g', '3g', '2g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y'], ['1g'], ['1r', '2y']]
Action done to this state: 5 to 3
------------------------------------------------------------
[['5g', '4g', '3g', '2g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2y'], ['1g'], ['1r']]
Action done to this state: 4 to 1
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1y'], ['5y', '4y', '3y', '2y'], ['#'], ['1r']]
Action done to this state: 2 to 3
------------------------------------------------------------
[['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['1r']]
Action done to this state: 5 to 2
------------------------------------------------------------
Final State: [['5g', '4g', '3g', '2g', '1g'], ['5r', '4r', '3r', '2r', '1r'], ['5y', '4y', '3y', '2y', '1y'], ['#'], ['#']]
------------------------------------------------------------
Number of generated nodes:3719
Number of explored nodes:913
```

مقایسه تعداد گره های تولید شده، بسط داده شده و عمق پاسخ در سه الگوریتم:

Test1)

5 3 5

5g 4g 3g 2g 1g

5r 4r 3r

5y 4y 3y 2r 1y

1r 2y

#

Test1			
	Generated nodes	Expanded nodes	Depth of solution
BFS	1037	402	5
IDS	31221	3970	5
A*	266	49	5

Test2)

5 3 5

5y 4y 3y 2y 1y

#

#

5r 4r 3r 2r 1r

5g 4g 3g 2g 1g

Test2			
	Generated nodes	Expanded nodes	Depth of solution
BFS	1	0	0
IDS	1	0	0
A*	1	0	0

Test3)

5 2 5

5r 4r 2r 1g

5g 4g 3g 1r

3r 2g

#

#

Test1			
	Generated nodes	Expanded nodes	Depth of solution
BFS	5814	2497	7
IDS	1825298	213506	7
A*	2342	471	7

باسپاس از توجه شما