به نام خدا

پروژه ی اول هوش مصنوعی:

در این پروژه قصد بر آن است که بازی ارایه شده در صورت سوال را به وسیله ی ۳ الگوریتم سرچ پیاده سازی نماییم.

الف) BFS

ابتدا باید حالت اولیه ی بازی به همراه تعداد ستون ها و تعداد رنگ ها و تعداد کارت ها را دریافت کنیم. این کار با گرفتن ورودی از کاربر انجام میشودو به دلیل سهولت parse با فرمت مشخصی باید ورودی را به برنامه بدهیم. سپس پس از گرفتن ورودی سطر اول آن را که مربوط به k,m,n می باشد را جدا و بقیه را به عنوان حالت اولیه ی مسیله تعریف میکنیم.

برای هر جست و جویی ابتدا حالت اولیه را بررسی میکنیم که ایا حالت مطلوب است یا خیر و اگر حالت مطلوب نباشد الگوریتم جست و جو را اجرا میکنیم.

برای هر جست و جو دو مجموعه ی frontier و explored داریم که باعث عدم ایجاد لوپ میشوند و در هر مرحله نود های تولید شده یا مقادیر موجود در این دو مجموعه مقایسه میشود و در صورتی که نودی قبلا تولید شده باشد دیگر به مجموعه ی frontier اضافه نخواهد شد.

مجموعه ی frontier در این الگوریتم به صورت صف FirstInFirstOut باید پیاده سازی گردد.(زیرا نودی که اول وارد شده در عمق کمتری قرار دارد و برای بسط دادن باید انتخاب گردد.) برای این منظور از queque استفاده میکنیم که دارای توابعی مانند put,get,empty می باشد و سرعت آن نسبت به لیست برای پیاده سازی صف FIFO بالاتر است.

در ابتدا initial state را باید به مجموعه ی frontier اضافه کنیم سپس درون حلقه ی while تا زمانیکه مجموعه ی frontier خالی نشده است الگوریتم را اجرا میکنیم. (به هنگام اجرای این الگوریتم اگر از recursive call استفاده میکردیم stack پایتون پر میشد و برنامه با خطا مواجه میشد بنابراین از حلقه ی while استفاده کردیم.)

ابتدا باید اولین نود وارد شده به صف frontierرا خارج کنیم و آن را به عنوان نودی که میخواهیم بسط دهیم انتخاب و به مجموعه ی explored اضافه کنیم. سپس در مرحله ی بعد باید فرزندان آن نود را بیابیم.

برای یافتن فرزندان هر نود حالت های مجاز حرکت را می یابیم و بعد از انجام آن حرکت حالت جدید به وجود آمده را با مقادیر موجود در دو مجموعه ی frontier و explored چک میکنیم اگر حالت تکراری نبود آن را به مجموعه ی نود های جدید اضافه میکنیم. برای یافتن مسیر از خروجی به ورودی از متغیر دیکشنری path استفاده میکنیم بدیم صورت که بعد از اضافه شدن هر نود به frontier، برای هر نود والد آن و حرکت مربوطه را اضافه میکنیم. مقدار key برای دیگشنری نمیتواند لیست باشد و نود ها را باید به tuple تبدیل کنیم.

سپس روی مجموعه نود های تولید شده آزمون هدف انجام میدهیم و اگر هدف نبودند همه را به مجموعه ی frontier اضافه میکنیم و مراحل را از سر میگیریم.

پس به صورت خلاصه جست و جوی اول سطح ابتدا فرزندان یک نود را تولید میکند و سپس در مرحله ی بعد نودی را برای گسترش انتخاب میکند که در سطح پایینتری قرار دارد.(عمق کمتر) دقت شود در این الگوریتم آزمون هدف هنگام تولید نود صورت میگیرد پس بعد از تولید فرزندان در هر مرحله باید نود های تولید شده چک گردند که آیا وضعیت مطلوب هستند یا خیر.

برای یافتن عمق جواب از دو لیست level و growing استفاده میکنیم که level نشان دهنده ی عمق جواب است که برخلاف IDS که از صفر شروع میشود من سطح اول را عمق ۱ در نظر گرفتم(می شود سطح اول را نیز عمق صفر درنظر گرفت و جواب هر دو الگوریتم یکی شود ولی الان به دلیل offset یک واحدی عمق ها یک واحد با هم اختلاف دارند.) growing لیستی تعداد نود های تولید شده در هر سطح است که اگر به مقدار نهایی برسد یعنی نود های آن سطح همگی تشکیل شده است و مقدار سطح یکی زیاد میشود.

پیچیدگی زمانی برای ان الگوریتم $O(b^{d+1})$ و پیچیدگی فضایی $O(b^d)$ می باشد.(در بدترین حالت)

تعداد گره های تولید شده را برای ورودی زیر برای الگوریتم ها بررسی میکنیم زیرا با ورودی که در صورت پروژه آمده است نمیتوان با این الگوریتم به جواب رسید.

Input:

4 3 3 , 1r 2y 2g , 3g 1g , 3y 1y , 3r 2r

```
The nodes in the path:
 [(('1r', '2y', '2g'), ('3g', '1g'), ('3y', '1y'), ('3r', '2r')), (('1r', '2y', '2g'), ('3g',), ('3y', '1y'),
 ('3r', '2r', '1g')), (('1r', '2y'), ('3g', '2g'), ('3y', '1y'), ('3r', '2r', '1g')), (('1r', '2y'), ('3g', '
2g', '1y'), ('3y',), ('3r', '2r', '1g')), (('1r',), ('3g', '2g', '1y'), ('3y', '2y'), ('3r', '2r', '1g')), ((
'1r',), ('3g', '2g'), ('3y', '2y', '1y'), ('3r', '2r', '1g')), (('1r',), ('3g', '2g', '1g'), ('3y', '2y', '1y
'), ('3r', '2r')), ((), ('3g', '2g', '1g'), ('3y', '2y', '1y'), ('3r', '2r', '1r'))]
These are the actions:
 ['move 1g from col 2 to col 4', 'move 2g from col 1 to col 2', 'move 1y from col 3 to col 2', 'move 2y from
col 1 to col 3', 'move 1y from col 2 to col 3', 'move 1g from col 4 to col 2', 'move 1r from col 1 to col 4']

Target in bfs:  [[], ['3g', '2g', '1g'], ['3y', '2y', '1y'], ['3r', '2r', '1r']]  The Depth is: 7
```

همانطور که میبینیم برای این ورودی جواب در عمق ۷ (با شروع از عمق ۱) پیدا شده است و نود های و حرکت های solution نوشته شده است.

تعداد گره های بسط داده شده برابر مجموعه ی explored و گره های تولید شده مجموع frontier و explored می باشد.

بسط داده شده: 117

تولید شده: 147=30+117

ب) IDS

الگوریتم اول عمق و تفاوت اساسی با اول سطح دارد. گر انتخاب شده در این الگوریتم بر اساس عمق بیشتر است برخلاف اول سطح و آزمون هدف نیز به هنگام بسط نود انجام می شود نه تولید نود.

گرفتن ورودی همانند قبل می باشد فقط یک پارامتر اضافه برای مشخص کردن limit اولیه برای شروع الگوریتم نیز گرفته می شود.

برای این جست و جو مجموعه ی frontier باید LIFO باشدو میتوانیم از لیست یا tuple استفاده کنیم که به دلیل سرعت بالاتر از tuple استفاده میکنیم.

برای مقدار حد از نقطه ی شروع ابتدا باید حالت اولیه را به مجموعه مرزی اضافه میکنیم و سپس تابع DLS صدا زده میزنیم.

در واقع IDS جست و جوی عمقی تکرار شونده می باشد که از عمق صفر شروع میکند و هر بار جست و جوی DLS را تا ان عمق انجام میدهد و در صورتی که جواب پیدا نشد عمق را یکی افزایش دهد و دوباره DLS را تا آن عمق اجرا میکند.

در جست و جوی DLS ابتدا باید چک بکنیم نودی که میخواهیم بسط بدهیم هدف می باشد یا خیر و اگر هدف نبود باید چک کنیم آیا به ابتدای گراف رسیده ایم یا خیر(limit ≠ 0) و اگر به پایان نرسیده بودیم باید از مجموعه ی frontier نود آخر را برداریم و بسط دهیم.(برای مثال برای حالت اول همان state اولیه تنها عضو مجموعه ی مرزی می باشد.) نود را به مجموعه ی explored اضافه میکنیم و فرزندان آن را تولید میکنیم. سپس آن ها را به مجموعه ی مرزی اضافه میکنیم و تابع را به صورت recursive با limit-1 و depth+1 صدا میزنیم. با صدا زدن به صورت بازگشتی ابتدا یک نود تا حد limit بسط جلو میرود و سپس به سراغ نود بعدی در عمق کمتر میرود.

برای اینکه نودی که میخواهیم چک کنیم همان نودی باشد که میخواهیم بسط دهیم از reversed(ss_nextlevel) برای iteration استفاده میکنیم.

عمق در این حالت از ۰ شروع میشود.

پیچیدگی زمانی این الگوریتم $O(b^d)$ و پیچیدگی فضایی این الگوریتم O(bd) می باشد.

حال برای ورودی زیر جواب مسیله را می یابیم:

Input:

4 3 3 , 1r 2y 2g , 3g 1g , 3y 1y , 3r 2r

Initial limit = 0

```
The path is :
 [(('1r', '2y', '2g'), ('3g', '1g'), ('3y', '1y'), ('3r', '2r')), (('1r', '2y', '2g'), ('3g', '1g'), ('3y',), ('3r', '2r', '1y')
), (('1r', '2y', '2g'), ('3g',), ('3y', '1g'), ('3r', '2r', '1y')), (('1r', '2y', '2g'), ('3g', '1y'), ('3y', '1g'), ('3r', '2r'
)), (('1r', '2y', '2g'), ('3g', '1y'), ('3y',), ('3r', '2r', '1g')), (('1r', '2y'), ('3g', '1y'), ('3y', '2g'), ('3r', '2r', '1g
')), (('1r', '2y'), ('3g', '1y'), ('3y', '2g', '1g'), ('3r', '2r')), (('1r', '2y'), ('3g',), ('3y', '2g', '1g'), ('3r', '2r', '1
y')), (('1r', '2y', '1g'), ('3g',), ('3y', '2g'), ('3r', '2r', '1y')), (('1r', '2y', '1g'), ('3g',), ('3y', '2g', '1y'), ('3r',
'2r')), (('1r', '2y', '1g'), ('3g', '2r'), ('3y', '2g', '1y'), ('3r',)), (('1r', '2y', '1g'), ('3g', '2r'), ('3y', '2g'), ('3r',
 '1y')), (('1r', '2y'), ('3g', '2r'), ('3y', '2g', '1g'), ('3r', '1y')), (('1r', '2y'), ('3g', '2r', '1y'), ('3y', '2g', '1g'),
('3r',)), (('1r', '2y'), ('3g', '2r', '1y'), ('3y', '2g'), ('3r', '1g')), (('1r', '2y', '1y'), ('3g', '2r'), ('3y', '2g'), ('3r
', '1g')), (('1r', '2y', '1y'), ('3g', '2r', '1g'), ('3y', '2g'), ('3r',)), (('1r', '2y', '1y'), ('3g', '2r', '1g'), ('3y',), ('3
r', '2g')), (('1r', '2y', '1y'), ('3g', '2r'), ('3y',), ('3r', '2g', '1g')), (('1r', '2y', '1y'), ('3g',), ('3y', '2r'), ('3r',
'2g', '1g')), (('1r', '2y', '1y'), ('3g',), ('3y', '2r', '1g'), ('3r', '2g')), (('1r', '2y', '1y'), ('3g', '2g'), ('3y', '2r', '
1g'), ('3r',)), (('1r', '2y', '1y'), ('3g', '2g'), ('3y', '2r'), ('3r', '1g')), (('1r', '2y'), ('3g', '2g'), ('3y', '2r', '1y'),
 ('3r', '1g')), (('1r', '2y'), ('3g', '2g', '1g'), ('3y', '2r', '1y'), ('3r',)), (('1r', '2y'), ('3g', '2g', '1g'), ('3y', '2r')
, ('3r', '1y')), (('1r', '2y', '1g'), ('3g', '2g'), ('3y', '2r'), ('3r', '1y')), (('1r', '2y', '1g'), ('3g', '2g', '1y'), ('3y',
 '2r'), ('3r',)), (('1r', '2y', '1g'), ('3g', '2g', '1y'), ('3y',), ('3r', '2r')), (('1r', '2y', '1g'), ('3g', '2g'), ('3y', '1y
'), ('3r', '2r')), (('1r', '2y'), ('3g', '2g'), ('3y', '1y'), ('3r', '2r', '1g')), (('1r', '2y', '1y'), ('3g', '2g'), ('3y',), (
'3r', '2r', '1g')), (('1r', '2y', '1y'), ('3g', '2g', '1g'), ('3y',), ('3r', '2r')), (('1r', '2y'), ('3g', '2g', '1g'), ('3y',),
 ('3r', '2r', '1y')), (('1r',), ('3g', '2g', '1g'), ('3y', '2y'), ('3r', '2r', '1y')), (('1r',), ('3g', '2g', '1g'), ('3y', '2y'
, '1y'), ('3r', '2r')), ((), ('3g', '2g', '1g'), ('3y', '2y', '1y'), ('3r', '2r', '1r'))]
The actions are :
 ['move 1y from col 3 to col 4', 'move 1g from col 2 to col 3', 'move 1y from col 4 to col 2', 'move 1g from col 3 to col 4', 'm
ove 2g from col 1 to col 3', 'move 1g from col 4 to col 3', 'move 1y from col 2 to col 4', 'move 1g from col 3 to col 1', 'move
 1y from col 4 to col 3', 'move 2r from col 4 to col 2', 'move 1y from col 3 to col 4', 'move 1g from col 1 to col 3', 'move 1y f
rom col 4 to col 2', 'move 1g from col 3 to col 4', 'move 1y from col 2 to col 1', 'move 1g from col 4 to col 2', 'move 2g from
 col 3 to col 4', 'move 1g from col 2 to col 4', 'move 2r from col 2 to col 3', 'move 1g from col 4 to col 3', 'move 2g from col
 4 to col 2', 'move 1g from col 3 to col 4', 'move 1y from col 1 to col 3', 'move 1g from col 4 to col 2', 'move 1y from col 3 to
  col 4', 'move 1g from col 2 to col 1', 'move 1y from col 4 to col 2', 'move 2r from col 3 to col 4', 'move 1y from col 2 to col
 3', 'move 1g from col 1 to col 4', 'move 1y from col 3 to col 1', 'move 1g from col 4 to col 2', 'move 1y from col 1 to col 4',
  'move 2y from col 1 to col 3', 'move 1y from col 4 to col 3', 'move 1r from col 1 to col 4']
The depth is 6
```

همانگونه که میبینیم جواب در سطح ۷ ام (عمق ۶ برای این الگوریتم زیرا اولین سطح را دارای عمق ۰ درنظر گرفته ایم) قرار دارد اما تعداد نود های تولید شده کمتر از BFS میباشد.

بسط داده شده: 38

تولید شده: 82=44+38

ج) A*

این الگوریتم یک جست و جوی آگاهانه می باشد بدیم صورت که با انتخاب یک هیوریستیک قابل قبول میتوانیم هوشمندانه تر عمل کنیم و زودتر به نتیجه برسیم. در این الگوریتم برخلاف دو الگوریتم قبلی که بر اساس عمق نود تصمیم میگیرند این الگوریتم بر اساس f(n)=g(n)+h(n) تصمیم میگیرد و هر چه f کوچکتر باشد گزینه ی مناسبتری برای انتخاب است. تابع g هزینه ی هر عمل را نشان می دهد که برای این تابع هزینه ی هر عمل را ۱ در نظر میگیریم پس اگر نودی در سطح ۲ باشد هزنیه ی g آن ۲ و نود سطح ۳ هزینه ی g آن ۳ می باشد.

هیوریستیک را قدر مطلق مقدار جابه جایی لازم هرکارت در هر ردیف هر نود برای رفتن به محل درست خود صرفا بر اساس عدد در نظر میگیریم. برای مثال برای یک ردیف:

1r 4y 2g => 0+(+2)+(-1)=+1

این هیوریستیک قابل قبول است زیرا حرکت های لازم برای رسیدم به حالت مجاز بیشتر از مقداری است که h به ما میدهد زیرا در نظر گرفتن این حرکات در حالت ایده ال و بدون محدودیت حرکت های غیر مجاز هستیم. با توجه به اینکه حرکات ما دارای محدودیت است و بحث شیفت دادن به چپ و راست نیست نیاز به جا به جایی بین ستون ها هم می باشد پس عددی که ازنی تابع به دست می اوریم بسیار کمتر از عدد واقعی است.

برای اینکه این تابع فقط به عدد وابسته نباشد و رفتار درست تری داشته باشد باید آن را به رنگ نیز وابسته کنیم بدین صورت که اگر رنگ و عدد درستی بود (مقایسه با لیست ساخته شده برای هر ستون) یکی از هزینه ی آن کاسته و اگر نابرابر بود یکی به هزینه ی آن اضافه میکنیم.

حال به شرح الگوریتم میپردازیم:

این الگوریتم مانند قسمت قبل دارای آزمون هدف هنگام بسط نود می باشد.

ابتدا حالت اولیه را دریافت کرده و سپس آن را چک کرده که ایا هدف می باشد یا خیر. سپس برای هر نود باید g,h,f را مشخص کنیم همانطور که گفته شد f را متناسب با عمق درنظر میگیریم و h را از تابع هیوریستیک به دست می اوریم.

تا زمانیکه مجموعه ی frontier خالی شود یا به جواب برسیم باید حلقه ی while را تکرار کنیم. در این حلقه ابتدا فرزندان نود را تولید میکنیم سپس در میان مجموعه ی مرزی نودی را پیدا میکنیم که f پایینتری داشته باشد و آن را به explored اضافه میکنیم و این کار را ادامه میدهیم.

مثالی که برای دو سمت نوشتیم را به این مسیله میدهیم.

Input:

4 3 3 , 1r 2y 2g , 3g 1g , 3y 1y , 3r 2r

```
the path is:
 [(('1r', '2y', '2g'), ('3g', '1g'), ('3y', '1y'), ('3r', '2r')), (('1r', '2y', '2g'), ('3g',), ('3y', '1y'), (
'3r', '2r', '1g')), (('1r', '2y'), ('3g', '2g'), ('3y', '1y'), ('3r', '2r', '1g')), (('1r', '2y'), ('3g', '2g',
 '1g'), ('3y', '1y'), ('3r', '2r')), (('1r', '2y'), ('3g', '2g', '1g'), ('3y',), ('3r', '2r', '1y')), (('1r',),
 ('3g', '2g', '1g'), ('3y', '2y'), ('3r', '2r', '1y')), (('1r',), ('3g', '2g', '1g'), ('3y', '2y', '1y'), ('3r'
, '2r')), ((), ('3g', '2g', '1g'), ('3y', '2y', '1y'), ('3r', '2r', '1r'))]
the actions are:
 ['move 1g from col 2 to col 4', 'move 2g from col 1 to col 2', 'move 1g from col 4 to col 2', 'move 1y from co
l 3 to col 4', 'move 2y from col 1 to col 3', 'move 1y from col 4 to col 3', 'move 1r from col 1 to col 4']
Final state:  ((), ('3g', '2g', '1g'), ('3y', '2y', '1y'), ('3r', '2r', '1r'))  the depth is: 7
Explored Nodes:  10
Frontier Nodes:  19
```

همانطور که میبینیم جواب در سطح ۷(عمق ۷) قرار دارد و مسیر رسیدن به جواب را مشاهده میکنیم اما تعداد نود ها نسبت به دو الگوریتم قبل کاهش یافته است.

بسط داده شده: 10

تولید شده: 29=19+10

حال مسیله را برای ورودی سوال حل میکنیم. البته حل این مسیله به زمانی طولانی تر نیاز است (حدود چند دقیقه)

Input:

5 3 5 , 5g 5r , 2g 4r 3y 3g 2y , 1y 4g 1r , 1g 2r 5y 3r , 4y

```
the path is:
 [(('5g', '5r'), ('2g', '4r', '3y', '3g', '2y'), ('1y', '4g', '1r'), ('1g', '2r', '5y', '3r'), ('4y',)), (('5g', '5r'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g'),
('1g', '2r', '5y', '3r'), ('4y',)), (('5g', '5r'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g', '3r'), ('1g', '2r', '5y'), ('4y',)), (('5g', '5r', '4y'), ('2g', '4r',
 '3y', '3g', '2y', '1r'), ('1y', '4g', '3r'), ('1g', '2r', '5y'), ()), (('5g', '5r', '4y'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g', '3r'), ('1g', '2r'), ('5y',))
, (('5g', '5r'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g', '3r'), ('1g', '2r'), ('5y', '4y')), (('5g', '5r'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g', '3
r', '2r'), ('1g',), ('5y', '4y')), (('5g', '5r'), ('2g', '4r', '3y', '3g', '2y', '1r'), ('1y', '4g', '3r', '2r', '1g'), (), ('5y', '4y')), (('5g',), ('2g', '4r', '3y', '3g
', '2y', '1r'), ('1y', '4g', '3r', '2r', '1g'), ('5r',), ('5y', '4y')), (('5g',), ('2g', '4r', '3y', '3g', '2y'), ('1y', '4g', '3r', '2r', '1g'), ('5r',), ('5y', '4y', '1r')
), (('5g',), ('2g', '4r', '3y', '3g'), ('1y', '4g', '3r', '2r', '1g'), ('5r', '2y'), ('5y', '4y', '1r')), (('5g', '3g'), ('2g', '4r', '3y'), ('1y', '4g', '3r', '2r', '1g'),
 ('5r', '2y'), ('5y', '4y', '1r')), (('5g', '3g', '2y'), ('2g', '4r', '3y'), ('1y', '4g', '3r', '2r', '1g'), ('5r',), ('5y', '4y', '1r')), (('5g', '3g', '2y', '1r'), ('2g',
 '4r', '3y'), ('1y', '4g', '3r', '2r', '1g'), ('5r',), ('5y', '4y')), (('5g', '3g', '2y', '1r'), ('2g', '4r'), ('1y', '4g', '3r', '2r', '1g'), ('5r',), ('5y', '4y', '3y')),
 (('5g', '3g', '2y', '1r'), ('2g',), ('1y', '4g', '3r', '2r', '1g'), ('5r', '4r'), ('5y', '4y', '3y')), (('5g', '3g', '2y', '1r'), (), ('1y', '4g', '3r', '2r', '1g'), ('5r'
, '4r'), ('5y', '4y', '3y', '2g')), (('5g', '3g', '2y', '1r'), (), ('1y', '4g', '3r', '2r'), ('5r', '4r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '3g', '2y', '1r'), ('2r'
,), ('1y', '4g', '3r'), ('5r', '4r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '3g', '2y', '1r'), ('2r',), ('1y', '4g'), ('5r', '4r', '3r'), ('5y', '4y', '3y', '2g', '1g'))
, (('5g', '3g', '2y', '1r'), (), ('1y', '4g'), ('5r', '4r', '3r', '2r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '3g', '2y'), (), ('1y', '4g'), ('5r', '4r', '3r', '2r', '1
r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '3g'), (), ('1y', '4g', '2y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g',), ('3g',), ('1y', '4g'
, '2y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g',), ('3g', '2y'), ('1y', '4g'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g
')), (('5g', '4g'), ('3g', '2y'), ('1y',), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '4g', '1y'), ('3g', '2y'), (), ('5r', '4r', '3r', '2r',
'1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '4g', '1y'), ('3g',), ('2y',), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '4g'), ('3g',), ('2y
', '1y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '1g')), (('5g', '4g', '3g'), (), ('2y', '1y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g', '
1g')), (('5g', '4g', '3g'), ('1g',), ('2y', '1y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2g')), (('5g', '4g', '3g', '2g'), ('1g',), ('2y', '1y'), ('5r', '4r',
 '3r', '2r', '1r'), ('5y', '4y', '3y')), (('5g', '4g', '3g', '2g', '1g'), (), ('2y', '1y'), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y')), (('5g', '4g', '3g', '2g', '
1g'), ('1y',), ('2y',), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y')), (('5g', '4g', '3g', '2g', '1g'), ('1y',), (), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y'
, '2y')), (('5g', '4g', '3g', '2g', '1g'), (), (), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2y', '1y'))]
the actions are:
 ['move 1r from col 3 to col 2', 'move 3r from col 4 to col 3', 'move 4y from col 5 to col 1', 'move 5y from col 4 to col 5', 'move 4y from col 1 to col 5', 'move 2r from c
ol 4 to col 3', 'move 1g from col 4 to col 3', 'move 5r from col 1 to col 4', 'move 1r from col 2 to col 5', 'move 2y from col 2 to col 4', 'move 3g from col 2 to col 1', '
move 2y from col 4 to col 1', 'move 1r from col 5 to col 1', 'move 3y from col 2 to col 5', 'move 4r from col 2 to col 4', 'move 2g from col 2 to col 5', 'move 1g from col
3 to col 5', 'move 2r from col 3 to col 2', 'move 3r from col 3 to col 4', 'move 2r from col 2 to col 4', 'move 1r from col 1 to col 4', 'move 2y from col 1 to col 3', 'mov
e 3g from col 1 to col 2', 'move 2y from col 3 to col 2', 'move 4g from col 3 to col 1', 'move 1y from col 3 to col 1', 'move 2y from col 2 to col 3', 'move 1y from col 1 t
o col 3', 'move 3g from col 2 to col 1', 'move 1g from col 5 to col 2', 'move 2g from col 5 to col 1', 'move 1g from col 2 to col 1', 'move 1y from col 3 to col 2', 'move 2
y from col 3 to col 5', 'move 1y from col 2 to col 5']
Final state:  (('5g', '4g', '3g', '2g', '1g'), (), (), ('5r', '4r', '3r', '2r', '1r'), ('5y', '4y', '3y', '2y', '1y'))  the depth is: 35
Explored Nodes:  17847
Frontier Nodes:  43569
```

همانطور که مشاهده شد جواب در عمق 35 قرار دارد و تعدا کل نود ها تولید شده برابر است با 61416