نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

گزارش پروژه اول

استاد: **جناب آقای روشن فکر**
درس: **مبانی هوش مصنوعی**

فهرست مطالب

نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

گزارش پروژه اول

استاد: **جناب آقای روشن فکر**
درس: **مبانی هوش مصنوعی**

فرموله سازی مسئله

حالت اولیه

همان حالتی میباشد که کاربر وارد برنامه میکند و بصورت یه آبجکت State ذخیره میشود که در جلوتر توضیح داده میشود.

عمل ها

عمل ها همان انتقال کارت ها هستند به شرطی که حرکت مجاز باشد. یعنی عدد کارتمان کوچکتر از کارتی باشد که میخواهیم کارت را روی آن بگذاریم.

مدل انتقال

توضیحات مربوط به مدل انتقال در قسمت توضیحات تابع move_card بیان شده است.

آزمون هدف

توضیحات مربوط به آزمون هدف در قسمت توضیحات تابع goal_test بیان شده است.

تابع هزینه

هزینه هر انتقال کارت را برابر ۱ در نظر میگیریم.

بخش های مشترک میان هر سه بخش

کلاس کارت (Card)

field ها

color
رنگ هر کارت که با یک (یا حتی چند) حرف مشخص میشود.

number
شماره هر عدد که یک عدد میباشد.

کلاس وضعیت (State)

field ها

cards_sections
وضعیت کارت ها در بخش های مختلف میباشد.

depth
فاصله هر وضعیت از نقطه شروع است.

parent
وضعیتی که به یک حرکت از آن به وضعیت فعلی رسیده ایم.

move_from
شماره section که کار از آن منتقل شده تا به این وضعیت برسیم.

move_to
شماره section که کار به آن منتقل شده تا به این وضعیت برسیم.

method ها

expand_children
تمام همسایگان مجاز وضعیت فعلی را در صورت تکراری نبودن تولید میکند. یک پارامتر print_it دارد که باید میکند در هر لحظه اطلاعات وضعیت فعلی چاپ شود یا خیر؛ ما برای خلوت ماندن ورودی آن را False قرار میدهیم اما این قابلیت برای کد ما وجود دارد که تمام وضعیت ها نمایش داده شوند. در نهایت تمام فرزندان در یک list توسط این متد return میشوند.

print_state
وضعیت فعلی را چاپ میکند. موقعیت کارت ها، نحوه جابجایی آخرین کارت و عمق را نشان میدهد.

goal_test
بررسی میکند که وضعیت فعلی هدف است یا خیر. چک میکند که در هر بخش فقط یک رنگ موجود باشد و اینکه شماره کارت ها در هر بخش سورت شده باشند. خروجی آن یک متغیر boolean است که بیانگر هدف بودن/نبودن وضعیت فعلی است. دو نمونه وضعیت هدف در زیر آورده شده است:

```
0 : 5y 4y 3y 2y 1y
1 : 1g
2 : 1r
3 : 5r 4r 3r 2r
4 : 5g 4g 3g 2g
```

```
0 : 5y 4y 3y 2y 1y
1 : #
2 : #
3 : 5r 4r 3r 2r 1r
4 : 5g 4g 3g 2g 1g
```

سایر توابع

split_card
یک رشته بصورت ورودی میگیرد و دو خروجی رشته ای تولید میکند. یکی تمام حروف و یکی هم تمام اعداد موجود در رشته اولیه.

print_cards
وضعیت section را مشخص میکند. بعنوان مثال:

```
0 : 5g 5r 4y
1 : 2g 4r 3y 3g 2y
2 : 1y 4g 1r
3 : 1g 2r 5y 3r
4 : #
```

move_card

میتواند یک card section را بعنوان ورودی بگیرد و یک کارت را انتقال دهد و card section جدید را بازگرداند. در این انتقال حالات مختلفی بررسی میشود. در صورت خالی بودن مقصد و کوچک تر بودن شماره کارت مبدا از شماره کارت مقصد واضح است نباید اجازه این انتقال را بدهیم. یک پارامتر print_it دارد که باید میکند اطلاعات مربوط به این انتقال چاپ شود یا خیر؛ ما برای خلوت ماندن ورودی آن را False قرار میدهیم اما این قابلیت برای کد ما وجود دارد که این اطلاعات نمایش داده شوند. بعنوان مثال اگر در وضعیت زیر باشیم:

```
5 3 5
5g 5r 4y
2g 4r 3y 3g 2y
1y 4g 1r
1g 2r 5y 3r
#
```

میتوان گفت:

```
****  MOVE FROM 4 TO 0  ****
!!! section 4 is empty
0 : 5g 5r 4y
1 : 2g 4r 3y 3g 2y
2 : 1y 4g 1r
3 : 1g 2r 5y 3r
4 : #
```

```
****  MOVE FROM 1 TO 3  ****
>>> moving card from section 1 to section  3
0 : 5g 5r 4y
1 : 2g 4r 3y 3g
2 : 1y 4g 1r
3 : 1g 2r 5y 3r 2y
4 : #
```

```
****  MOVE FROM 2 TO 1  ****
>>> moving card from section 2 to section  1
0 : 5g 5r 4y
1 : 2g 4r 3y 3g 2y 1r
2 : 1y 4g
3 : 1g 2r 5y 3r
4 : #
```

```
****  MOVE FROM 3 TO 4  ****
>>> moving card from section 3 to  empty section  4
0 : 5g 5r 4y
1 : 2g 4r 3y 3g 2y
2 : 1y 4g 1r
3 : 1g 2r 5y
4 : 3r
```

```
****  MOVE FROM 0 TO 1  ****
!!! can not move card from section 0 to section  1
```

get_inputs

میتوان ورودی ها را از کاربر گرفت و در پایان این تابع تعدادی آبجکت کارت خواهیم داشت.

قسمت main

ابتدا ورودی ها گرفته میشود و card section ساخته میشود. توسط الگوریتم مورد نظر مسیر رسیدن به هدف را پیدا میکنیم و بعد با توجه به این مسیر، card section را تغییر میدهیم. در پایان تایم جستجو را نمایش میدهیم.

الگوریتم BFS

در این الگوریتم از دو لیست به نام های frontier و explored بعنوان یک صف استفاده میکنیم. این دو لیست در قسمت main تعریف میشوند.

breadth_first_search

این تابع یک وضعیت اولیه و همچنین متغیر print_it که قبلا در مورد آن توضیحاتی دادیم را به عنوان ورودی میگیرد و هدف آن این است که وضعیت هدف را به ما برگرداند. در هر مرحله فرزندان وضعیت سر صف frontier تولید میشوند و در صورت نبودن در explored آن را به frontier اضافه میکنیم. علت وجود explored این است که میخواهیم از جستجوی گرافی استفاده کنیم.

الگوریتم IDS

در بخش main یک عدد با بعنوان limit برای این الگوریتم تعریف میکنیم که میتوان آن را تغییر داد. تمامی توابعی که در زیر تعریف شده اند ورودی print_it که قبلا توضیح داده شد را دارا هستند.

depth_limited_search

این تابع الگوریتم DLS را پیاده سازی میکند. یک وضعیت و یک عدد limit را بعنوان ورودی میگیرد و سپس این الگوریتم را بصورت بازگشتی اجرا میکند. حالت پایه این الگوریتم زمانی است که goal_test وضعیت مورد بررسی True شود؛ در این حالت وضعیت فعلی را بعنوان وضعیت هدف بازمیگردانیم. اگر limit ۰ شود

یعنی الگوریتم شکست خورده است. فرزندان وضعیت مورد نظر را تولید میکنیم و روی هریک از فرزندان، همین تابع را صدا میزنیم و یک واحد مقدار limit را کم میکنیم.

iterative_deepening_search

این تابع الگوریتم IDS را پیاده سازی میکند. یک وضعیت و یک عدد limit را بعنوان ورودی میگیرد. این تابع یک حلقه while بینهایت دارد که در آن تابع depth_limited_search را با limit که در main مشخص کرده بودیم اجرا میکند. درصورت وجود جواب آن را برمیگرداند ولی در صورتی که جواب ما در عمق مورد نظر وجود نداشت، مقدار limit را یک واحد زیاد میکنیم و دوباره این کار را تکرار میکنیم.

الگوریتم A*

در این قسمت هزینه هر انتقال کارت را ۱ در نظر میگیریم و برای محاسبه هزینه تا اینجا از get_depth استفاده میکنیم.

calculate_heuristic

در این تابع تک تک section های حالت را بررسی میکنیم. دو عامل را عامل زیاد شدن مقدار h میباشند

تعداد inversion های هر section

به تعداد زوج کارت هایی که شماره آن ها ترتیب صعودی خواسته شده را بر هم میزنند مقدار h زیاد میشود.

تعداد رنگ های موجود در هر srction

اگه در هر بخش فقط ۱ برگ وجود داشت، تغییری در h ایجاد نمیکنیم. در غیر این صورت به تعداد رنگ های موجود (بیشتر از ۱) به h اضافه میکنیم.

calculate_f

در این تابع مقدار f که حاصل جمع هزینه مسیر طی شده تا اینجا و تخمین هزینه رسیدن به وضعیت هدف میباشد محاسبه میشود.

node_with_minimum_f

در نود های قابل بسط دادن، نودی که کمترین میزاین f را دارد را انتخاب میکند.

a_star_search

در این تابع از میان گره هایی که قابلیت بسط دادن دارند گرهی که کم ترین مقدار f را دارد را بسط میدهد. در صورتی که گره ای که دارد بررسی میکند گره هدف باشد آن را return میکند.

نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

گزارش پروژه اول

استاد: **جناب آقای روشن فکر**
درس: **مبانی هوش مصنوعی**

تحلیل و مقایسه الگوریتم ها

محاسبه مقیاس ها

نحوه محاسبه تعداد گره تولید شده

در قسمت main یک متغیر به نام number_of_children تعریف میکنیم و در تابع expand_children از آن به عنوان یک متغیر global استفاده میکنیم. هر زمان که یک فرزند از گره فعلی مجاز به ایجاد شدن بود، علاوه بر تولید آن ۱ واحد به مقدار number_of_children نیز اضافه میکنیم. بنابراین در پایان تعداد گره های تولید شده را خواهیم داشت.

نحوه محاسبه تعداد گره بسط داده شده

از آنجا که هر زمان گره ای بسط داده میشود را به داخل explored وارد میکنیم، میتوان برای شمردن تعداد گره های بسط داده شده از size لیست explored استفاده کرد.

نحوه محاسبه عمق جواب

با توجه به ساختار کلاس State، فاصله هر وضعیت با وضعیت شروع را ذخیره شده داریم. بنابراین کافی است از getter این کلاس یعنی get_depth استفاده کنیم تا عمق را داشته باشیم.

مقایسه الگوریتم ها برای یک ورودی یکسان

ورودی

ورودی زیر را در نظر میگیریم:

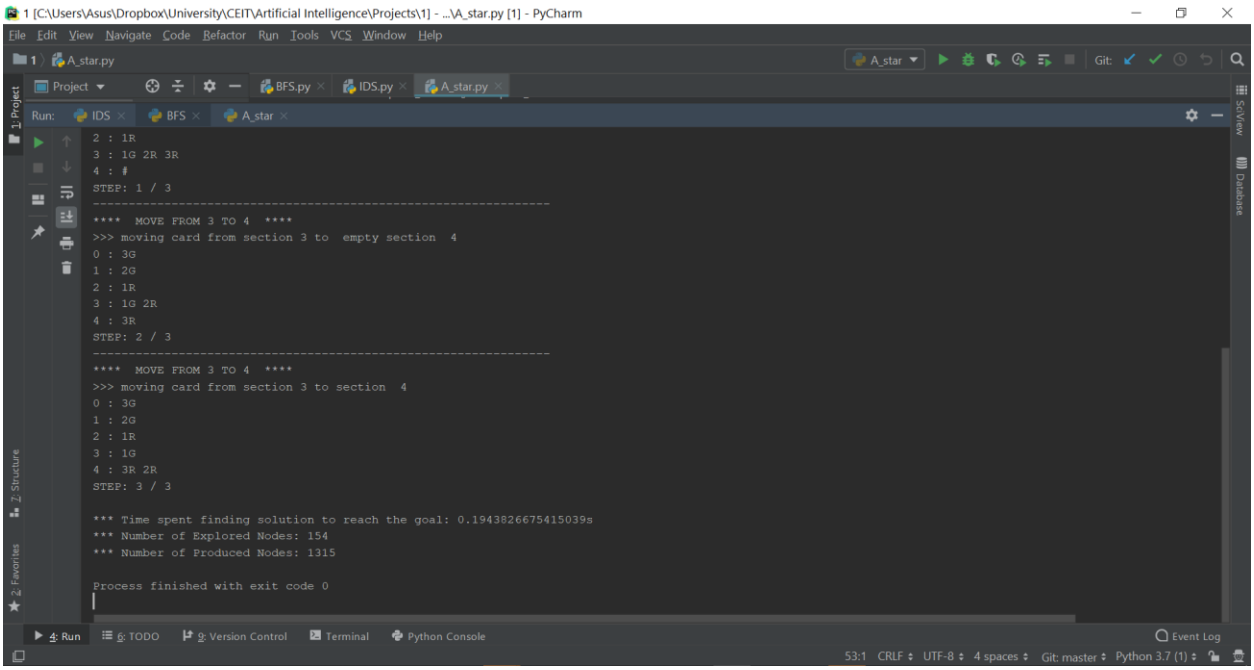
```
5 2 3
#
2G 3G
1R
1G 2R 3R
#
```

نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

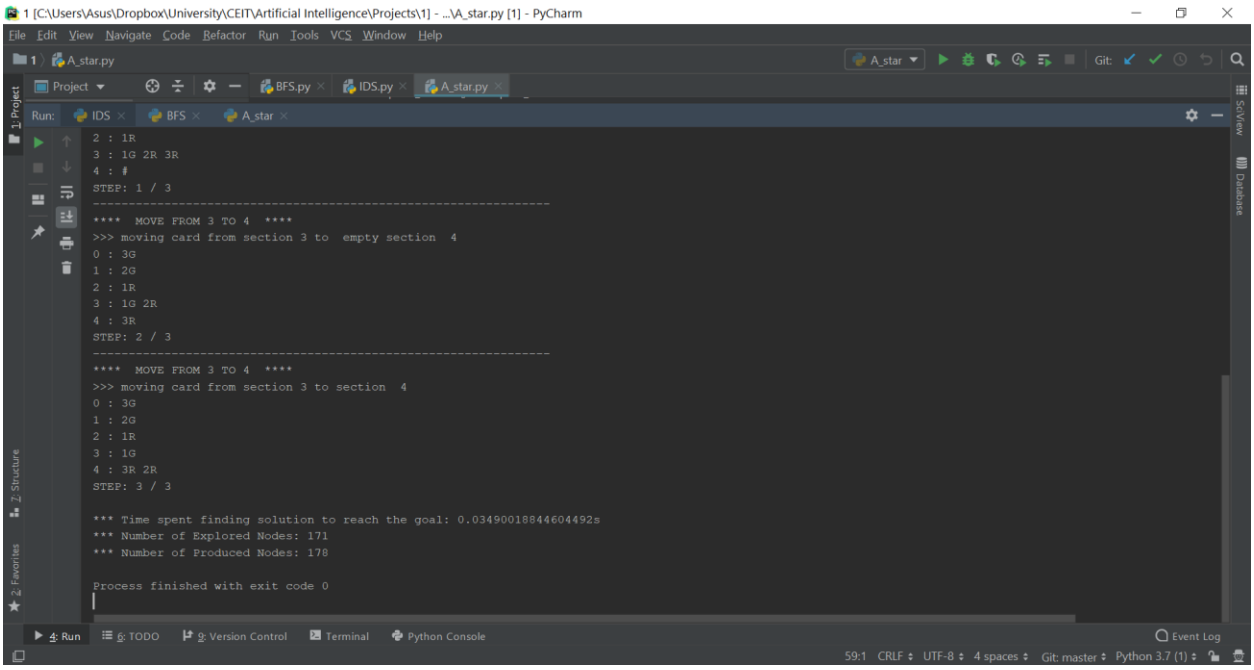
گزارش پروژه اول

استاد: **جناب آقای روشن فکر**
درس: **مبانی هوش مصنوعی**

خروجی BFS



خروجی IDS

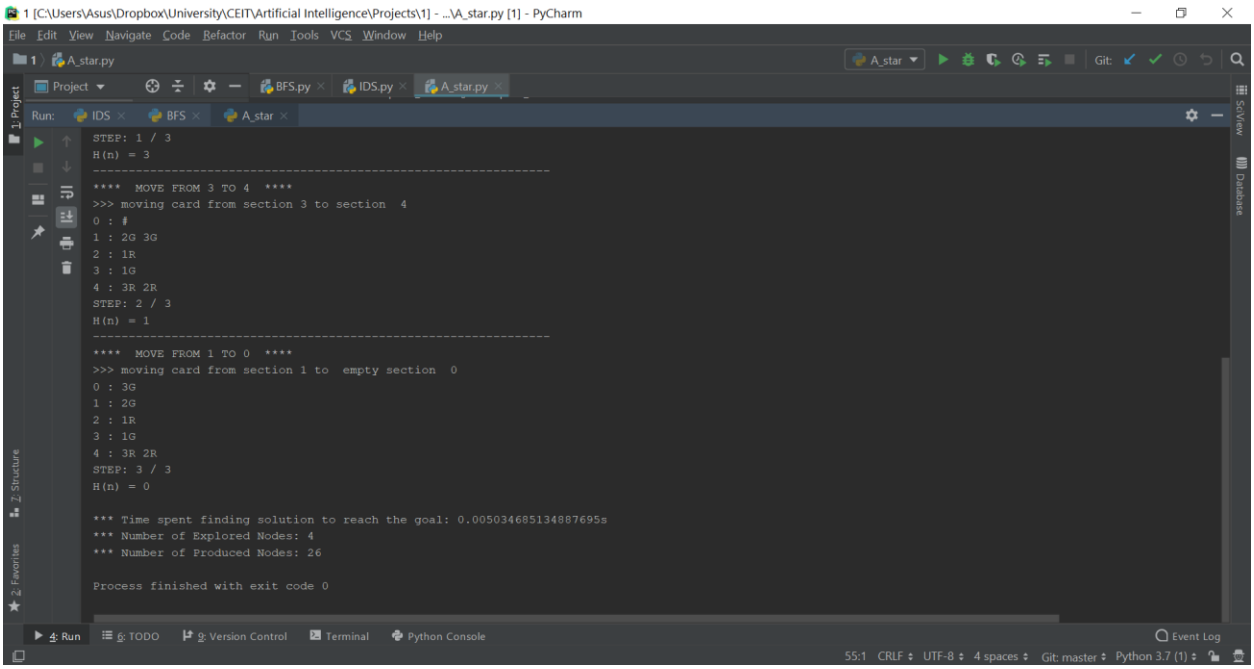


نام و نام خانوادگی: **امیرحسین علی‌بخشی**
شماره‌ی دانشجویی: **۹۷۳۱۰۹۶**

گزارش پروژه اول

استاد: **جناب آقای روشن فکر**
درس: **مبانی هوش مصنوعی**

خروجی A*



همانطور که مشاهده میشود عمق جواب در هر سه الگوریتم برابر مقدار یکسانی است. البته این برابری به limit الگوریتم IDS نیز وابسته میباشد. همانگونه که انتظار داشتیم الگوریتم A* با توجه به آگاهانه بودن نسبت به دو الگوریتم دیگر در زمان کمتر و با گره های تولید شده و بسط داده شده کمتری به هدف برسد. بعد از این الگوریتم، بهترین الگوریتم الگوریتم IDS میباشد که از BFS بهتر عمل کرده است. نکته خیلی همه در این اعداد و ارقام، نحوه پیاده سازی الگوریتم ها و ترتیب بسط دادن گره های فرزند میباشد