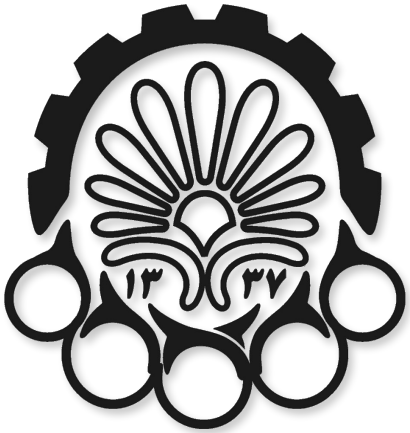
به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تمرین برنامه‌نویسی اول

سینا ملکوتی
۹۴۳۱۸۰۷

شرح پروژه:

در این پروژه مجموعه ای از الگوریتم‌های جست‌و جو که بعدا شرح داده می شوند پیاده‌سازی شده است و از آنها برا حل چند مسئله مختلف استفاده شده است.

الگوریتم‌های پیاده‌سازی شده :

۱- BFS

۲- DFS، IDS، DLS

۳- UCS

۴- دو جهته

۵- A*

توجه نمایید تمامی الگوریتم‌ها هم به صورت جست‌وجو درختی و هم گرافی پیاده‌سازی شده اند.

تمامی الگوریتم ها از یک interface ، AGENT استفاده می کنند، که در این اینترفیس یک تابع قرار داده شده است که هر الگوریتم آن را رو مسائل اجرا میکند. خروجی این تابع یک ساختمان داده‌ی Solution می باشد که پاسخ اجرای الگوریتم روی این مسئله می باشد.

همچنین یک ساختمان داده ی گره نیز استفاده شده است که توسط آن درخت جست‌و جو ساخته میشود این ساختمان داده حاوی اطلاعات زیر می باشد:

```
private State state;
    private Node parent;
    private Action action;
    private  int pathCost;
    private  int depth;
    private int fn ;
```

مسائل:

به ازای هر یک از مسائل یک ساختمان داده‌ی STATE که هر یک از حالت های مجاز

مسئله را تعریف می کند ساخته شده است. همچنین هر یک از مسائل یک ساختمان داده‌ی ACTION نیز دارند که در این کلاس یک فیلد move دارد و نشان دهنده‌ی اکشنی است که انجام شده است.
همچنین تمامی کلاس های problem از یک کلاس abstract، Problem به ارث رسیده اند که تابع های مورد نیاز هر یک از مسائل در آنها پیاده سازی شده است.
به عنوان مثال :
۱- تابع actionsFor(state s): تمام اکشن های مجاز حالت s را میدهد.
۲- تابع move(state s,action a): در واقع تابع مابعد میباشد. یعنی نشان می دهد با انجام اکشن a در حالت s به چه حالتی خواهیم رفت.
۳- تابع isgoal: همان تابع ازمون هدف می باشد.
۹....

توجه : منظور از visited nodes: تعداد گره هایی است که وارد وارد frontier شده اند.
و منظور از expanded nodes : تعداد گره‌هایی است که گسترش داده شده است.

۱- ربات امدادگر

حالت مسئله : مکان‌هایی که ربات می تواند در آن قرار بگیرد.
حالت اولیه: نقطه‌ی (0,0)
حالت هدف: نقطه‌ی (4,4)

نتایج حاصل از جست و جو هزینه یکنواخت
:
برا پیاده سازی این الگوریتم از minHeap استفاده شده است زیرا باید بتوان هر مرتبه گره با کمترین fn را پیدا کرد. fn = gn + hn
درختی:

```
visited Nodes :76
expanded nodes : 61
cost : 8
memory usage : 18.0
path is :( 0, 0 ) -> ( 1, 0 ) -> ( 2, 0 ) -> ( 3, 0 ) -> ( 3, 1 ) -> ( 4, 1 ) -> ( 4, 2 ) -> ( 4, 3 ) -> ( 4, 4 )
result is :   D   D   D   R   D   R   R   R
```

گرافی

```
visited Nodes :25
expanded nodes : 26
cost : 8
memory usage : 27.0
path is :( 0, 0 ) -> ( 1, 0 ) -> ( 2, 0 ) -> ( 2, 1 ) -> ( 2, 2 ) -> ( 2, 3 ) -> ( 2, 4 ) -> ( 3, 4 ) -> ( 4, 4 )
result is :    D   D   R   R   R   R   D   D
```

نتایج حاصل از جست و جو عمق اول گرافی:

این الگوریتم کامل هست یعنی حتما به جوام میرسد اما لزوما جواب بهینه نیست یعنی الگوریتم بهینه نیست.
برای پیاده سازی این الگوریتم از ساختمان داده‌ی پشته استفاده شده است.

```
visited Nodes :24
expanded nodes : 17
cost : 16
memory usage : 25.0
path is :( 0, 0 ) -> ( 0, 1 ) -> ( 0, 2 ) -> ( 0, 3 ) -> ( 0, 4 ) -> ( 1, 4 ) -> ( 2, 4 ) -> ( 2, 3 ) -> ( 2, 2 ) -> ( 2, 1 ) -> ( 2, 0 ) -> ( 3, 0 ) -> ( 4, 0 ) -> ( 4, 1 ) -> ( 4, 2 )
 -> ( 4, 3 ) -> ( 4, 4 )
result is :    R   R   R   R   D   D   L   L   L   L   D   D   R   R   R   R
```

نتایج حاصل از جست و جو دوطرفه:
از دو آرایه یکی برای رفت و یکی برای بگشت استفاده شده است.

```
visited Nodes :16
expanded nodes : 21
cost : 8
memory usage : 40.0
path is :( 0, 0 ) -> ( 1, 0 ) -> ( 2, 0 ) -> ( 3, 0 ) -> ( 4, 0 ) -> ( 4, 1 ) -> ( 4, 2 ) -> ( 4, 3 ) -> ( 4, 4 )
result is :    D   D   D   D   R   R   R   R
```

نتایج حاصل از جست و جو *A:

برا پیاده سازی این الگوریتم از minHeap استفاده شده است زیرا باید بتوان هر مرتبه گره با کمترین fn را پیدا کرد. fn = gn + hn

```
visited Nodes :12386
expanded nodes : 3636
cost : 8
memory usage : 8751.0
path is :( 0, 0 ) -> ( 1, 0 ) -> ( 2, 0 ) -> ( 2, 1 ) -> ( 3, 1 ) -> ( 3, 2 ) -> ( 3, 3 ) -> ( 4, 3 ) -> ( 4, 4 )
result is :    D   D   R   D   R   R   D   R
```

نتیجه گیری:
الگوریتم DFS بهینه نیست و بر اساس ترتیب جست و جویی که دارد (اول کدام اکشن را انجام داده و نود فرزند را گسترش می دهد) به جواب های بهتری می رسد. بقیه الگوریتم ها توانسته اند بهترین جواب را پیدا کنند.

بهترین الگوریتم برای این تست کیس :
زیرا در مواردی مانند میزان حافظه مصرفی با تست کیس های مختلف ممکن است ترتیب الگوریتم ها عوض شود.
در کل به نظر می رسد در این مثال بهترین عملکرد را الگوریتم UCS گرافی داشته است.
و بدترین عملکرد را DFS گرافی زیرا به جواب بهینه نرسیده است.

۱- پازل ۸ تایی:

حالت ها : جایگشت مختلف قرار گیری اعداد در این مسئله
حالت اولیه : ورود کاربر
حالت پایانی : قرار گیری به صورت زیر :

۰	۱	۲
۳	۴	۵
۶	۷	۸

همانطور که میدانید این مسئله تنها در صورتی قابل حل است که تعداد inversion های آن زوج باشد. به همین علت در کلاس Problem2 یک تابع isSolvable نوشته شده است.
به عنوان مثلا مسئله به ازای ورودی داده شده در صورت پروژه:

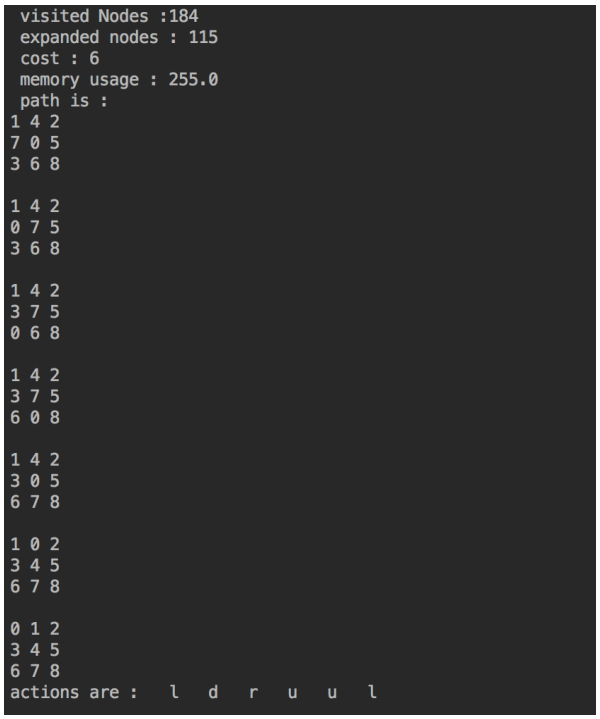
```
int [] board = {4,5,2,1,7,3,0,6,8};
```

قابل حل نمی باشد.

ورودی : int [] board = {1,4,2,7,0,5,3,6,8};

نتیجه اجرای الگوریتم UCS :

برا پیاده سازی این الگوریتم از minHeap استفاده شده است زیرا باید بتوان هر مرتبه گره با کمترین fn را پیدا کرد. fn = gn + hn



```
 visited Nodes :184
 expanded nodes : 115
 cost : 6
 memory usage : 255.0
 path is :
1 4 2
7 0 5
3 6 8

1 4 2
0 7 5
3 6 8

1 4 2
3 7 5
0 6 8

1 4 2
3 7 5
6 0 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8
actions are :   l   d   r   u   u   l
```

نتیجه اجرای الگوریتم دوجهته :
از دو آرایه یکی برای رفت و یکی برای بگشت استفاده شده است.

```
 visited Nodes :22
 expanded nodes : 22
 cost : 6
 memory usage : 58.0
 path is :
1 4 2
7 0 5
3 6 8

1 4 2
0 7 5
3 6 8

1 4 2
3 7 5
0 6 8

1 4 2
3 7 5
6 0 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8
actions are :   l   d   r   u   u   l
```

نتیجه اجرای الگوریتم DFS گرافی :

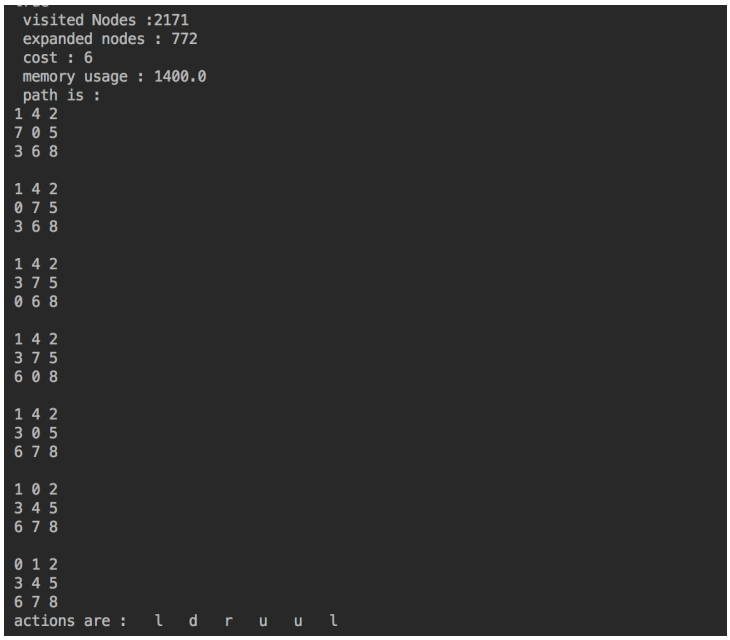
```
 visited Nodes :41027
 expanded nodes : 23540
 cost : 22648
 memory usage : 41028.0
 path is :

actions are :
```

چون خیلی طولانی بود مسیر را پرینت نکردم.

این مسئله نشان میدهد خروجی و زمان طول کشیدن الگوریتم بسیار وابسته به ورودی و ترتیب انجام اعمال می باشد. و اصلا الگوریتم مناسبی برای این مسئله نیست.

اجرای الگوریتم *A:

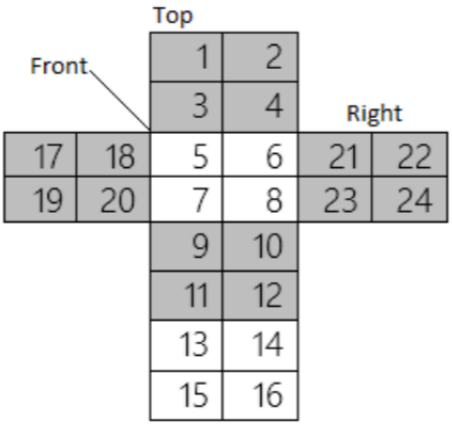


نتیجه :
بدیهی است که الگوریتم dfs اصلا مناسب نیست حتی اگر الگوریتم DLS را با عمق محدود ۸ اجرا کنیم به جواب می رس پس نشان می دهد الگوریتم DFS معمولی اصلا مناسب نیست. بقیه الگوریتم ها به جواب بهینه رسیده اند و به نظر می رسد در این مثال الگوریتم دو جهته از نظرم حافظه نتیجه بهتری داشته است.

۳- مکعب روبیک

حالت ها : ترتیب قرار گرفتن رنگ ها در خانه های مختلف

حالت اولیه : ورودی کاربر



به عنوان مثال ورودی زیر معادل مکعب مقابل می باشد :

```
Input :
y b y b g y g y w g w g b w b w r r r r o o o o
```

حالات

پایانی: حالت هایی که هر وجه تنها یک رنگ داشته باشد.

ورودی :

اجرا الگوریتم اول سطح:

```
 visited Nodes :1
 expanded nodes : 1
 cost : 1
 memory usage : 1.0
 path is :
5 2 5 2 1 5 1 5 3 1 3 1 2 3 2 3 0 0 0 0 4 4 4 4

5 5 5 5 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0 0 4 4 4 4
actions are :   R
```

عمق اول با عمق محدود ۱۴ :

```
 visited Nodes :7815
 expanded nodes : 7815
 cost : 13
 memory usage : 7815.0
 path is :
5 2 5 2 1 5 1 5 3 1 3 1 2 3 2 3 0 0 0 0 4 4 4 4

5 5 2 2 4 4 1 5 3 1 3 1 2 3 0 0 1 5 0 0 3 2 4 4

2 5 2 5 3 2 1 5 3 1 3 1 2 3 1 5 4 4 0 0 0 0 4 4

2 2 5 5 0 0 1 5 3 1 3 1 2 3 4 4 3 2 0 0 5 1 4 4

5 2 5 2 5 1 1 5 3 1 3 1 2 3 3 2 0 0 0 0 4 4 4 4

5 5 2 2 4 4 1 5 3 1 3 1 2 3 0 0 5 1 0 0 2 3 4 4

2 5 2 5 2 3 1 5 3 1 3 1 2 3 5 1 4 4 0 0 0 0 4 4

2 2 5 5 0 0 1 5 3 1 3 1 2 3 4 4 2 3 0 0 1 5 4 4

2 2 0 3 1 0 5 0 4 1 3 1 2 3 4 4 2 3 0 1 5 5 5 4

2 2 5 5 0 0 1 5 3 1 3 1 2 3 4 4 2 3 0 3 1 5 4 4

5 2 5 2 1 5 1 5 3 1 3 1 2 3 2 3 0 0 0 3 4 4 4 4

5 5 5 5 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0 3 4 4 4 4

5 5 3 0 1 1 1 1 4 4 3 3 2 2 2 2 0 3 0 3 5 4 5 4

5 5 5 5 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0 0 4 4 4 4
actions are :   T   T   T   T   T   T   T   F   FC  T   R   F   FC
```

اجرای الگوریتم IDS :

در کد این الگوریتم تا ماکزیمم عمقی که اجرا می شود را نیز تعیین کرده ام (hardcode) و مقدار ان ۲۰ می باشد.

```
 visited Nodes :6
 expanded nodes : 6
 cost : 1
 memory usage : 6.0
 path is :
5 2 5 2 1 5 1 5 3 1 3 1 2 3 2 3 0 0 0 0 4 4 4 4

5 5 5 5 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0 0 4 4 4 4
actions are :   R
```

نتیجه :

همانطور که مشاهده می شود الگوریتم DSL(14) به جواب بهینه نرسیده است. البته در صورتی که ماکزیمم عمق انرا کم کنیم میرسد. که درواقع همان الگوریتم IDS می شود که هر بار از کمرین عمق ممکن الگوریتم DLS را اجرا میکند.

چون پاسخ در عمق اول بود. بنابراین الگوریتم ids و bfs نسبتا مشابه عمل کرده اند.