گزارش تمرین برنامه نویسی اول:

برای پیاده سازی الگوریتم ها ، از زبان پایتون استفاده شده که همه فایل ها به پیوست قرار گرفته است .

برای پیاده سازی الگوریتم ها ، ابتدا هر مساله را مدل سازی کرده ایم .
به این صورت که برای هر مساله توابع (حالت اولیه ، عمل های ممکن ، نتیجه هر عمل ، هزینه گام به گام ، مقدار هیوریستیک برای هر استیت و ...) را تعریف کردیم و با توجه به شکل مساله ، این توابع را پیاده سازی کردیم .
اسم های توابع و همچنین مدل کلی پیاده سازی الگوریتم ها نیز به این صورت بوده است که برای مثال ، در کل یک الگوریتم BFS زده شده و در مسایل مختلف ، از آن استفاده شده است .

مساله ربات :

برای این مساله ، حالت اولیه برابر با (۱،۱) در نظر گرفته شده است .
همچنین مطابق تعریف مساله ، مختصات دیوار ها نیز مشخص می شوند و به عنوان یکی از پارامتر های کلاس State ، مقدار دهی می شود .
همچنین تابع بررسی هدف نیز ، صرفا بررسی می کند که آیا مقدار حالت به مثلا (۵،۵) رسیده باشد.
برای به دست آوردن اکشن های ممکن در هر حالت ، از چندین شرط تو در تو استفاده شده است و در آخر ، آرایه ایی حاوی عمل های ممکن به الگوریتم مربوطه پاس داده می شود .
به دست آوردن تابع نتیجه بسیار ساده است . مثلا اگر در استیت (۱،۱) باشیم و تقاضای اکشن R کنیم ، مقدار استیت برابر با (۱،۲) می شود .
همچنین برای استفاده از الگوریتم دو طرفه ، باید یک تابع initializeGoal داشته باشیم که مقدار هدف را به عنوان استیت اولیه به تابع دوطرفه ، پاس دهد . در این تابع صرفا همین عمل انجام می شود .
تابع stepCost نیز همیشه مقدار ۱ را بیرون می دهد . (مطابق تعریف مساله)
برای تابع huristic مجموع فاصله افقی و عمودی هر استیت از هدف ، به عنوان huristic آن استیت ، به تابع A_Star پاس داده می شود .
مثلا برای حالت مثال داده شده در صورت پروژه ، جواب به ازای الگوریتم های مختلف ، به شرح زیر است :

DFS :

Finded
Path = ['R', 'R', 'R', 'R', 'D', 'D', 'L', 'L', 'L', 'L', 'D', 'D', 'R', 'R', 'R', 'R']
Cost = 16
Number of seen nodes = 16
Number of expanded nodes = 16
Memory used : 25

==========

UCS :

Finded
Path = ['R', 'R', 'R', 'R', 'D', 'D', 'D', 'D']
Cost = 8
Number of seen nodes = 25
Number of expanded nodes = 25
Memory used : 25
==========
A_STAR :

Finded
Path = ['D', 'D', 'D', 'D', 'R', 'R', 'R', 'R']
Cost = 8
Number of seen nodes = 25
Number of expanded nodes = 25
Memory used : 25
==========

مشاهده می شود برای این مساله ، الگوریتم DFS بسیار بهتر از دیگر الگوریتم ها عمل کرده است . چون مقدار هیوریستیک مناسبی برای آن در نظر گرفته شده و مانع دور شدن مساله از جواب می شود .(نه لزوما)
با حالت اولیه داده شده ، چون مرتبه زمانی الگوریتم دو طرفه بسیار بالاست ، جواب نمی دهد . ولی به ازای حالات ساده تر ، جواب آماده می شود .

مساله پازل :

برای این مساله ، یک حالت اولیه در نظر می گیریم . برای تابع بررسی حالت نهایی نیز ، حالت نهایی داده شده در صورت سوال را ، هر بار چک می کنیم .
برای تعریف تابع Actions ، ابتدا مقدار صفر در جدول را پیدا کرده و سپس با توجه به مکان قرار گرفتنش ، اکشن های ممکن را در آرایه مربوط ریخته و پاس می دهیم .

برای تابع result نیز ، مانند تابع اکشن ، ابتدا صفر را پیدا کرده و سپس با توجه به اکشن مربوطه ، جابجا می کنیم .

تابع stepCost نیز مانند مساله قبل ، همیشه مقدار یک را بیرون می دهد .

برای به دست آوردن تابع huristic مجموع فواصل مستقیم خانه هر عدد ، با خانه ای که باید در آن حضور داشته باشد .

مانند مساله قبل ، برای استفاده از تابع دوطرفه باید یک تابع initializeGoal تعریف کنیم که مقدار حالت نهایی را به عنوان حالت اولیه به تابع پاس دهد .

مقدار خروجی به ازای الگوریتم های مختلف ، به شرح زیر است :

DFS :

Finded
Path = ['L', 'L']
Cost = 2
Number of seen nodes = 2
Number of expanded nodes = 2
Memory used : 5
==========
UCS :

Finded

Path = ['L', 'L']
Cost = 2
Number of seen nodes = 7
Number of expanded nodes = 7
Memory used : 14
==========
BIDIRECTIONAL :

Finded
Path2 ['L', 'L']
Cost = 0
Number of seen nodes = 2
Number of expanded nodes = 2
Memory used : 6
==========
A_STAR :

Finded
Path = ['L', 'L']
Cost = 2
Number of seen nodes = 3
Number of expanded nodes = 3
Memory used : 5

برای این مساله ، مشاهده می شود که الگوریتم A_Star از همه بهتر عمل کرده (به ازای این مساله خاص)

مساله روبیک :

برای این مساله ، حالت اولیه را برابر با صورت پروژه پر می کنیم .

برای تابع بررسی هدف ، ابتدا یک رنگ یک مربع را به دست می آوریم ، سپس بقیه مربع را بررسی کرده که همان رنگ را داشته باشند.

برای تابع actions ، چون در هر مرحله هر عملی قابل انجام است ، پس تمام اکشن های گفته شده را به الگوریتم ها پاس می دهیم .

برای تابع result ، با داشتن دید سه بعدی (که بنده نداشتم و یک روبیک با کاغذ درست کردم) مقادیری که جابجا می شوند را به ازای عمل موردنظر به دست می آوریم و جابجا می کنیم .

تابع stepCost نیز مانند مساله های قبل ، همیشه مقدار یک را بیرون می دهد .

خروجی به ازای الگوریتم های خواسته شده ، به صورت زیر است :

BFS :

Finded
Path = ['R']
Cost = 1
Number of seen nodes = 1
Number of expanded nodes = 1
Memory used : 6
==========
DFS_LIMITED :

Finded
Path = ['R']

Cost = 1
Number of seen nodes = 1
Number of expanded nodes = 1
Memory used : 6
==========
DFS_ADITIVE :

Finded
Path = ['R']
Cost = 1
Number of seen nodes = 1
Number of expanded nodes = 1
Memory used : 6

به ازای حالت داده شده در صورت سوال ، تمام الگوریتم ها مشابه هم عمل می کنند با دادن تست کیس های بیشتر ، تفاوتشان مشخص می شود .
البته که الگوریتم A_Star چون از تابع هیوریستیک استفاده می کند ، در کل بهتر عمل خواهد کرد .

برای تمام الگوریتم ها ، حالت گرافی و درختی زده شده با این تفاوت که ، برای وارد کردن یک استیت در لیست Frontier ، چک نمی شود که آیا در e_list هست یا نه .

پایان
۹۴۳۱۰۲۵
پدارم نوری