بنام خداوند بخشنده و مهربان

# گزارش تمرین اول برنامه نویسی درس هوش مصنوعی

سپیده ملانوروزی ۹۴۳۱۰۷۲

مسئله ربات امدادگر :

حالت اولیه : یک وکتور شامل (۱,۱) که مختصات محل حضور ربات را نگه داشته است.
اعمال : با استفاده از تابع actions(state) میتوان عمل های ممکن برای هر حالت را بدست آورد که در این مسئله خروجی این تابع حرکت های ممکن بین u,d,l,r است. و به صورت یک وکتور برگردانده میشود.
مدل انتقال: تابع result(state , action) حالتی را به ما میدهد با انجام عمل ذکر شده از حالت ذکر شده به آن میرویم. در پیاده سازی این قسمت به محل وجود دیوار ها توجه میشود و اگر برای رفتن از مبدا به مقصد دیوار باشد مقصد را برابر مبدا قرار میدهیم و ینی تکان نمیخورد!
آزمون هدف: حالت هدف در این مسئله زمانی است که وکتور حالت برابر (n,m) شود.
هزینه گام : در این مسئله برابر ۱ در نظر گرفته شده است.
هزینه مسیر : همان هزینه ی رسیدن به نود هدف است.

پیاده سازی الگوریتم ها:

تنها تفاوت روش درختی و گرافی در نگه داشتن مجموعه ی explored set است. من در کد الگوریتم ها را به صورت گرافی در نظر گرفتم و میتوان با حدف این مجموعه الگوریتم را به درختی تبدیل کرد. وکتور visited همه نودهایی که تا به حال تولید شده اند را نگه میدارد پس مقدار حداکثر حافظه مورد نیاز برابر اندازه ی این وکتور است.

برای الگوریتم UCS باید Frontier به صورت صف اولویت پیاده سازی شود. پس هربار بعد از اد شدن مقداری در آن، آن را مرتب میکنیم. با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 25

num of expanded nodes : 24

solution: ddddrrrr

path cost : 8

برای پیاده سازی DFS باید frontier به صورت یک صف LIFO در نظر گرفته شود. . با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 24

num of expanded nodes : 19

solution: rrrrddldd

path cost : 9

برای پیاده سازی جست و جوی دو طرفه در هر طرف از BFS استفاده شده است. با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 1514

num of expanded nodes : 21

path cost : 8

solution : ddddrrrr

برای پیاده سازی الگوریتم A* تابع شهودی فاصله مستقیم برای هر حالت تا حالت هدف برابر مجموعه فواصل افقی و عمودی تا حالت هدف در نظر گرفته شده است. با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 25

num of expanded nodes : 24

solution: ddddrrrr

path cost : 8

مسئله پازل ۸ تایی:

حالت اولیه : یک وکتور به اندازه ۹ که در هر خانه ی آن عددی قرار دارد

0 1 2

3 4 5 → 0 1 2 3 4 5 6 7

6 7 8

اعمال : با استفاده از تابع actions(state) میتوان عمل های ممکن برای هر حالت را بدست آورد که در این مسئله خروجی این تابع حرکت های ممکن بین u,d,l,r است برای حرکت دادن عدد ۰ در وکتور است و به صورت یک وکتور برگردانده میشود.

مدل انتقال: تابع result(state , action) حالتی را به ما میدهد با انجام عمل ذکر شده از حالت ذکر شده به آن میرویم. در پیاده سازی این قسمت جای عدد ۰ با عددی که با حرکت مورد نظر به جای آن خواهیم رفت عوض میشود.

آزمون هدف: حالت هدف در این مسئله زمانی است که وکتور حالت برابر حالتی شود که index هر عدد با خودش برابر شود.

هزینه گام : در این مسئله برابر ۱ در نظر گرفته شده است.

هزینه مسیر : همان هزینه ی رسیدن به نود هدف است.

پیاده سازی الگوریتم ها:

تنها تفاوت روش درختی و گرافی در نگه داشتن مجموعه ی explored set است. من در کد الگوریتم ها را به صورت گرافی در نظر گرفتم و میتوان با حدف این مجموعه الگوریتم را به درختی تبدیل کرد. وکتور visited همه نودهایی که تا به حال تولید شده اند را نگه میدارد پس مقدار حداکثر حافظه مورد نیاز برابر اندازه ی این وکتور است.

مثال زیر را برای تست الگوریتم در نظر میگیریم.

1 2 5

3 4 0

6 7 8

برای الگوریتم UCS باید Frontier به صورت صف اولویت پیاده سازی شود. پس هربار بعد از اد شدن مقداری در آن، آن را مرتب میکنیم. با اجرای مثال داده شده مقادیر زیر بدست آمد:

num of visited nodes : 22

num of expanded nodes : 10

solution: ull

path cost : 3

برای پیاده سازی DFS باید frontier به صورت یک صف LIFO در نظر گرفته شود. . با اجرای مثال داده شده مقادیر زیر بدست آمد:

num of visited nodes : 2128

num of expanded nodes : 1177

solution:
dllurdlurdluurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurd
luurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddlur
dlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdluurrddllurdlurdluurddlurdlurdru
ldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddlurdlurdruldlurdlurd
ruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddlurdlurdruldlurdlurdruldlurdlu
rdruldlurdlurdruldlurdluurrddllurdlurdluurddlurdlurdruldlurdlurdruldlurdlurdruldl
urdlurdruldlurdlurrdllurdlurdluurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdru
ldlurdlurrdllurdlurdluurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdluur
rddllurdlurdluurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlu
rdluurddlurdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddl
urdlurdruldlurdlurdruldlurdlurdruldlurdlurdruldlurdluurrddllurdlurdluurddlurdlurd
ruldlurdlurdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddlurdlurdruldlurdlu
rdruldlurdlurdruldlurdlurdruldlurdlurrdllurdlurdluurddlurdlurdruldlurdlurdruldlur
dlurdruldlurdlurdruldlurdlu

path cost : 1086

برای پیاده سازی جست و جوی دو طرفه در هر طرف از BFS استفاده شده است. با اجرای مثال داده شده مقادیر زیر بدست آمد:

num of visited nodes : 5

num of expanded nodes : 2

path cost : 12

solution : ull

برای پیاده سازی الگوریتم A* تابع شهودی فاصله مستقیم برای هر حالت تا حالت هدف برابر مجموعه فواصل هر عدد از جایی که باید باشد در نظر گرفته شده است. با اجرای مثال داده شده مقادیر زیر بدست آمد:

num of visited nodes : 7

num of expanded nodes : 3

solution: ull

path cost : 3

مسئله روبیک ۲*۲ :

حالت اولیه : یک وکتور شامل ۲۴ رنگ است که هر رنگ با یک رشته ی تک حرفی مشخص میشود و به یک خانه از روبیک اشاره دارد (در این سوال خانه های روبیک به جای از ۱ تا ۲۴ از ۰ تا ۲۳ در نظر گرفته شده است و رنگ مربوط به هر خانه در ایندکس مربوطه در وکتور رنگ هاست).
اعمال : با استفاده از تابع actions(state) میتوان عمل های ممکن برای هر حالت را بدست آورد که در این مسئله خروجی این تابع حرکت های ممکن بین T, TC, R, RC, F, FC است. و به صورت یک وکتور از رشته های تک حرفی برگردانده میشود.
مدل انتقال: تابع result(state , action) حالتی را به ما میدهد با انجام عمل ذکر شده از حالت ذکر شده به آن میرویم. در پیاده سازی این قسمت با توجه به عمل انجام شده باید ببینیم که رنگ ها چگونه در خانه ها تغییر میکنند.
هزینه گام : در این مسئله برابر ۱ در نظر گرفته شده است.
هزینه مسیر : همان هزینه ی رسیدن به نود هدف است.

پیاده سازی الگوریتم ها:

تنها تفاوت روش درختی و گرافی در نگه داشتن مجموعه ی explored set است. من در کد الگوریتم ها را به صورت گرافی در نظر گرفتم و میتوان با حدف این مجموعه الگوریتم را به درختی تبدیل کرد. وکتور visited همه نودهایی که تا به حال تولید شده اند را نگه میدارد پس مقدار حداکثر حافظه مورد نیاز برابر اندازه ی این وکتور است.

برای الگوریتم BFS باید Frontier به صورت صف FIFO در نظر گرفته شود. با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 5

num of expanded nodes : 1

solution: R

path cost : 1

برای پیاده سازی DLS باید frontier به صورت یک صف LIFO در نظر گرفته شود. . با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

num of visited nodes : 5

num of expanded nodes : 1

solution: R

path cost : 1

برای پیاده سازی الگوریتم IDS ازالگوریتمDLS با مقادیرمختلف برایlimit از صفر تا زمانی که جواب بگیریم استفاده شده است. با اجرای مثال داده شده در صورت پروژه مقادیر زیر بدست آمد:

limit : 0 Unsuccessful

num of visited nodes : 5

num of expanded nodes : 1

solution: R

path cost : 1

limit : 1 Successful