Artificial Intelligence

Project 1

Mahtab Farrokh 9431016

تمرین ۱ -

برای مدلسازی این مسئله state ما میشود [i, j] که دوتایی لوکیشن ما در نقشه است

و هدف ما رسیدن به نقطه ی نهایی [m , n] و با شروع از خانه ی [1,1] است

و با داشتن یک تابع که action های ممکن ما را در هر لوکیشنی که هستیم برمیگرداند

و با دادن یک action و state کنونی به تابع result میفهمیم که nextstate ما چه است.

و همچنین تابع هایی که هزینه حرکت انجام شده و هزینه آینده را پیش بینی میکنند.

و این تابع ها را به کلاس الگوریتم های مختلفی که پیاده سازی کردیم میدهیم تا مسیر و هزینه و بقیه اطلاعات خواسته شده را چاپ کند.

حال به بررسی نتایج از توابع میپردازیم :

- هزینه یکنواخت

path found:

['d', 'd', 'd', 'd', 'r', 'r', 'r', 'r']

num visited: 25

num closed list: 24

max memory use: 34

path cost : 8

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی خانه ها را پیموده است.

- عمق اول (گرافی)

path found:

d d d d r u r d r u u u u r d d d d

num visited: 18

num closed list: 0

max memory use: 18

همانطور که میبینیم این الگوریتم از لحاظ حافظه شاید مناسب عمل کرده اما از لحاظ مسیری که

پیدا کرده اصلا مناسب عمل نکرده

- دوجهته

path found:

d d d r r r r d

num visited: 23

num closed list: 18

max memory use: 36

path cost : 8

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی

خانه ها را پیموده است.

- فاصله مستقیم A*

['d', 'd', 'd', 'r', 'r', 'r', 'd', 'r']

num visited: 24

num closed list: 24

max memory use: 34

path cost : 8.0

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی

خانه ها را پیموده است.

تمرین ۲ -

برای مدلسازی این مسئله state ما میشود یک ماتریس ۳ در ۳

و هدف ما رسیدن به نقطه ی نهایی [[0, 1, 2], [3, 4, 5], [6, 7, 8]]و با شروع از خانه ی حالتی که کاربر ورودی داده شروع میکنیم

و با داشتن یک تابع که action های ممکن ما را در هر لوکیشنی که هستیم برمیگرداند

و با دادن یک action و state کنونی به تابع result میفهمیم که nextstate ما چه است.

و همچنین تابع هایی که هزینه حرکت انجام شده و هزینه آینده را پیش بینی میکنند.

و این تابع ها را به کلاس الگوریتم های مختلفی که پیاده سازی کردیم میدهیم تا مسیر و هزینه و بقیه اطلاعات خواسته شده را چاپ کند.

حال به بررسی نتایج از توابع میپردازیم :

- هزینه یکنواخت

[[-1, 0, 'l'], [0, 1, 'd'], [0, 1, 'd'], [1, 0, 'r']]

num visited: 41

num closed list: 23

max memory use: 41

path cost : 4

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی

خانه ها را پیموده است.

- عمق اول (گرافی)

path found:

[0, 1, 'd'] [0, 1, 'd'] [-1, 0, 'l'] [0, -1, 'u'] [0, -1, 'u'] [1, 0, 'r'] [0, 1, 'd'] [0, 1, 'd'] [-1, 0, 'l'] [0, -1, 'u']

[0, -1, 'u'] [1, 0, 'r'] [0, 1, 'd'] [0, 1, 'd'] [-1, 0, 'l'] [0, -1, 'u'] [0, -1, 'u'] [1, 0, 'r'] [0, 1, 'd'] [0, 1, 'd'] [-1, 0, 'l'] [0, -1, 'u'] [0, -1, 'u'] [1, 0, 'r'] [0, 1, 'd'] [0, 1, 'd']

num visited: 26

num closed list: 0

max memory use: 26

همانطور که میبینیم این الگوریتم از لحاظ حافظه شاید مناسب عمل کرده اما از لحاظ مسیری که

پیدا کرده اصلا مناسب عمل نکرده

- دوجهته

path found:

[-1, 0, 'l']

[0, 1, 'd']

[0, 1, 'd']

[1, 0, 'r']

num visited: 11

num closed list: 6

max memory use: 14

path cost : 4

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی

خانه ها را پیموده است. همینطور نسبت به هزینه یکنواخت راس های کمتری را دیده و بهتر عمل

کرده

- فاصله مستقیم A*

path found:

[[-1, 0, 'l'], [0, 1, 'd'], [0, 1, 'd'], [1, 0, 'r']]

num visited: 9

num closed list: 4

max memory use: 10

path cost : 7.0

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و همه ی

خانه ها را پیموده است. همینطور نسبت به هزینه یکنواخت راس های کمتری را دیده و بهتر عمل

کرده

تمرین ۳ -

برای مدلسازی این مسئله state ما میشود یک ماتریس با ۲۴ مقدار که خانه های مکعب هستند

و هدف ما رسیدن به نقطه ی نهایی است که هر ۴ تایی که در یک وجه هستند با هم مقدار برابر داشته باشن و با شروع از خانه ی حالتی که کاربر ورودی داده شروع میکنیم

و با داشتن یک تابع که action های ممکن ما را در هر لوکیشنی که هستیم برمیگرداند

و با دادن یک action و state کنونی به تابع result میفهمیم که nextstate ما چه است و تمام نکته در نوشتن همین تابع result بود.

حال به بررسی نتایج از توابع میپردازیم :

- عمق محدود اگر ۳ باشد :

path found:

RC RC RC

num visited: 3

num closed list: 0

max memory use: 3

همانطورکه میبینیم این الگوریتم حافظه زیادی استفاده نکرده است تا به اینجا اما بهترین مسیر را هم پیدا نکرده است.
البته اینجا اولین action پیشنهادی RC بوده اما اگر آن اولین پیشنهاد را R بگذاریم

میتواند بهترین مسیر رانتخاب کند و اگر بجای RC هم چیزی دیگری بذاریم بسیار بسیار زیاد طول

میکشد و به نتیجه نمیرسیم. بطور کل این الگوریتم از همه ناکارامد تر است.

بطور مثال برای همان مثال اگر با محدودیت ۷ تست کنیم میبینیم که :

path found:

FC F RC RC RC FC F

num visited: 21523

num closed list: 4155

max memory use: 21520

و همانطور که مشاهده میشود یک مسیری پیدا شد که نه بهینه بود و هم زمان و حافظه زیادی

مصرف شده و حتی برای محدودیت عمق ۱۴ تا جایی که من گذاشتم اجرا بشه جواب نداد.

- عمق افزایشی

path found:

R

num visited: 6

num closed list: 0

max memory use: 1

این الگوریتم چون سطح را افزایش میدهد همواره بهترین جواب را پیدا میکند و تنها مشکل آن این

است که باید از ابتدا دوباره شروع کرد و زمان بیشتری میبرد اما تضمین میدهد بهترین جواب را پیدا

کند و همانطور که میبینیم حافظه زیادی هم نسبت به bfs استفاده نکرده است

- بی اف اس

path found:

['R']

num visited: 33

num closed list: 6

max memory use: 33

path cost : 1

همانطور که میبینم این الگوریتم بهترین مسیر را پیدا کرده اما حافظه زیادی مصرف کرده و حالت های زیادی را بررسی کرده است.