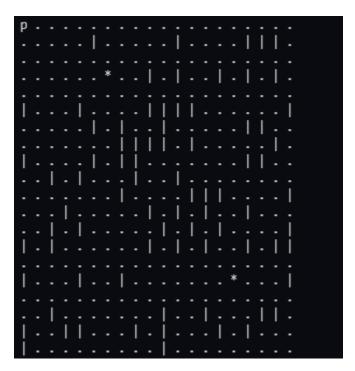
به نام خدا پروژهی هوش مصنوعی

الگوريتم اول (DFS):

در ابتدای کار تابعی نوشتم که بتواند مازم را تشکیل داده و خروجی آن را به درستی چاپ کندو در آن دو میوه و ۸۰ دیوار قرار دادم.

برای پیاده سازی این الگوریتم یک تابعی به نام maze2graph نوشتم. این تابع همسایه ی هر نود را به ما می دهد یعنی در واقع این تابع مجموعه ای را برمی گرداند که شامل همه ی نود های موجود است و نشان می دهد که هر نود با چه اکشنی به کدام یک از نود های اطرافش می رود. در این تابع بر اساس مازی که داریم چندین شرط قرار دادم و در نهایت خروجی آن به صورت زیر شد:



نتیجه تابع maze2graph را مربوط به تابع بالا در شکل زیر آمده است:

{(0,0): [('S', (1,0)), ('E', (0,1))], (1,0): [('S', (2,0)), ('E', (1,1))], (2,0): [('S', (3,0)), ('N', (1,0)), ('E', (2,1))], (3,0): [('S', (4,0)), ('N', (2,0)), ('E', (3,1))], (4,0): [('N', (3,0)), ('E', (4,1))], (5,0): [], (6,0): [('S', (7,0)), ('E', (6,1))], (7,0): [('N', (6,0)), ('E', (7,1))], (8,0): [], (9,0): [('S', (10,0)), ('E', (11,1))], (12,0): [('N', (11,0)), ('E', (11,0)), ('N', (9,0)), ('E', (14,1))], (12,0): [('N', (11,0)), ('E', (12,1))], (13,0): [], (14,0): [('S', (14,1)), ('E', (02,2))], (1,1): [('S', (17,0)), ('E', (16,1))], (17,0): [('N', (16,0)), ('E', (12,1))], (13,0): [], (10,0): [], (0,1): [('S', (1,1)), ('E', (02,2))], (1,1): [('S', (12,1)), ('N', (1,1)), ('N', (2,0)), ('E', (2,2))], (3,1): [('S', (4,1)), ('N', (1,0)), ('E', (13,2))], (4,1): [('S', (3,1)), ('N', (4,1)), ('N', (2,0)), ('E', (2,2))], (3,1): [('S', (4,1)), ('N', (1,1)), ('N', (3,1)), ('N', (4,0)), ('E', (2,2))], (3,1): [('S', (4,1)), ('N', (1,0)), ('E', (13,2))], (4,1): [('S', (11,1)), ('N', (12,1)), ('N', (11,1)), ('N', (12,1)), ('N', (11,1)), ('N', (1

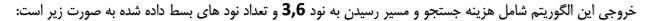
الگوریتم کاره است یک صف داشته باشیم که گره هایی که باید پردازش شوند را داخلش قرار دهیم و یک آرایه هم داشته باشیم که گره هایی که باید پردازش شوند را داخلش قرار دهیم و یک آرایه هم داشته باشیم که گره ها یا نود هایی که پردازش شده اند را داخلش قرار دهیم که دوباره آن ها را پردازش نکنیم. اگر همه ی گرههای مجاور گره جاری پیمایش شدهاند، گره بالای صف را به عنوان گره جاری از صف حذف کرده ومجدد پیمایش می کنیم.

خروجی این الگوریتم شامل هزینه جستجو و مسیر رسیدن به نود 3,6 و تعداد نود های بسط داده شده به صورت زیر است:

همین طور که میبینیم تعداد زیادی از نود ها پردازش شده اند و مسیر طولانی است این بدان دلیل است که الگوریتم علمی یالها یک الگوریتم جستجوی نااگاهانه است و در بزرگترین حالت تمامی نودها را پیمایش کرده و نیاز به بررسی تمامی یالها دارد. پس می توان گفت این الگوریتم بهینه نیست و به ما تضمینی برای یافتن بهترین راه حل نمی دهد.

الگوريتم Uniform Cost Search

مراحل اولیه ی ساخت ماز و تابع تبدیل maze2graph را لازم است برای این الگوریتم هم به کار ببریم و استفاده کنیم. این الگوریتم بر اساس هزینه ی هر نود جلو می رود در اینجا هزینه ی هر نود برابر فاصله ی آن نود تا مبدا است. در واقع الگوریتم به این صورت است که یک صف به نام frontier داریم که در آن نود هایی که قابل بسط داده شدن هستند را می نویسیم و در ابتدا نود صفر در آن قرار دارد. یک آرایه ای هم داریم به نام explored که شامل نود هایی است که از صف frontier انتخاب می کنیم و گسترشش می دهیم. لازم به ذکر است که صف frontier باید صف اولویت باشد به طوری که بر حسب هزینه بتوانیم آن را مرتب کنیم. در UCS هر بار راسی که کمترین فاصله رو داره انتخاب می کنیم و چک می کنیم که آیا راس هدف هست یا نه اگر راس هدف نبود گسترش داده میشه و تا رسیدن به هدف این کار رو ادامه میدیم.





این الگوریتم هم یک الگوریتم جست وجوی ناآگاهانه است. منظور از ناآگاهانه این است که این استراتژی ها فقط میتوانند پسین ها را تولید کنند و حالت هدف را از حالت غیر هدف تشخیص دهند.این الگوریتم بهینه است چون در هر مرحله مسیری با کمترین هزینه انتخاب می شود و مسیرها با اضافه شدن گرهها هرگز کوتاهتر نمی شوند و اطمینان حاصل می شود که جستجو گرهها را به ترتیب هزینه مسیر بهینه شان گسترش می دهد.