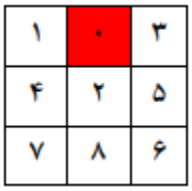
حسن اردشیر ۶۱۰۳۹۸۲۰۲

State ها در این مسئله ، وضعیت چینش کاشی ها می باشد که می توان آن را به صورت ارایه ای ۳*۳ نشان داد که هر خانه این ارایه نشان دهنده شماره کاشی در پازل متناظر است (به ازای خانه خالی (قرمز) عدد صفر در نظر گرفته می شود .) درنتیجه به ازای پازل زیر ، ماتریس زیر را داریم :



$$A = [[1,0,3],[4,2,5],[7,8,6]]$$

در نتیجه Goal State (حالت هدف) متناظر با ماتریس زیر می شود :

$$G = [[1,2,3],[4,5,6],[7,8,9]]$$

و برای Operator ها نیز ، حرکت خانه قرمز به UP و DOWN و RIGHT و LEFT را داریم که متناظر این حرکات در ماتریس بیان شده به صورت زیر می باشد . (لازم به ذکر است که فرض می شود که در جدول حرکت مربوطه قابل انجام است در غیر اینصورت این حرکت در ماتریس نیز تعریف نمی شود.)

$$UP: swap(A[i][j], A[i-1][j])$$

$$DOWN: swap(A[i][j], A[i+1][j])$$

$$RIGHT: swap(A[i][j], A[i][j+1])$$

$$LEFT: swap(A[i][j], A[i][j-1])$$

همچنین Path cost را نیز برابر تعداد اعمال انجام شده قرار می دهیم یعنی هزینه هر عمل یک واحد می باشد.

روشن است که برای Transition model این مسئله ، تنها با انجام اعمال در ماتریس ، ماتریس جدیدی به دست خواهد آمد که متناظر با پازل بعد از انجام عمل می باشد. پس تابعی دو متغیره (اولی ماتریس متناظر با پازل ورودی و دومی action مربوطه می باشد) داریم که یک کپی از از ماتریس ورودی گرفته و براساس action، ماتریس جدید را تغییر می دهد و خروجی می دهد. (در واقع ماتریس جدید متناظر با پازل جدید است .)

در این مسئله دو تابع Heuristic زیر را می توان بیان کرد:

۱) H1 : تعداد کاشی هایی که سر جای اصلی خود نیستند (موقعیت آنها با موقعیتشان در Goal State متفاوت است.)
۲) H2 : جمع فاصله منهتنی هر کاشی اشتباه با سر جای اصلی خود .

در تعاریف فوق ، خانه قرمز در محاسبات حساب نمی شود یعنی برای مثال H1 و H2 در مثال زیر برابر ۱ می باشد. (خانه ۶ خانه ای است که در سر جای اصلی خود نیست.)

۱	۲	۳
۴	۵	۰
۷	۸	۶

پس می دانیم برای این که یک کاشی اشتباه در سرجای خود قرار گیرد نیاز به حداقل یک عمل می باشد و در نتیجه H1 کم تر مساوی تعداد عمل های مورد نیاز برای رسیدن حالت ورودی به حالت هدف است.

همچنین برای این که یک خانه اشتباه به سر جای اصلی خود برود ، باید حداقل به اندازه فاصله منهتنی خود جابه جا شود و می دانیم هر اکشن ، تنها یک خانه را جابه جا می کند . پس در نتیجه حداقل به اندازه جمع فاصله منهتنی هر کاشی اشتباه با سر جای اصلی خود ، حرکت نیاز داریم پس H2 کم تر مساوی تعداد عمل های مورد نیاز است.

پس ثابت کردیم هر دو تابع مربوطه ، Heuristic می باشند و مشخص است که H2 از H1 بزرگ تر مساوی است پس H2 از H1 تابع بهتری بوده و درنتیجه H2 را اتخاب کردیم.

حال به ازای تمام مثال های گفته شده ، با تمام الگوریتم های مذکور ، مسئله را حل کرده ایم . لازم به ذکر است که نتیجه در فایل result – target.xlsx آورده شده است. برای سادگی جواب هر الگوریتم به صورت رشته ای متوالی از اختصار اعمال آورده شده است :

U: UP D: DOWN R:RIGHT L:LEFT

همچنین مقدار زمان مصرفی برای هر مثال با هر الگوریتم نیز ذکر شده است.

برای هر مثال با هر الگوریتم ، پارامتر MemPeak محاسبه شده است که نشان دهنده حداکثر فضای ذخیره سازی مموری در محاسبات است.

یکی از مشکلات به وجود آمده در حل این مسئله این بود که چون در الگوریتم DFS محدودیتی وجود نداشت ، الگوریتم برای هر مثال به مدت زیادی کار می کرد . بدین منظور دقیقا یک حداکثر ارتفاع برای تمام مثال ها تعیین شد تا الگوریتم در بیشتر مثال ها به جواب برسد (در بعضی مثال ها به جواب نرسیده است که با N/A نمایش داده شده است)

در نهایت هم میانگین زمان مصرفی و MemPeak مثال ها برای هر الگوریتم محاسبه شده است.

همانطور که مشخص است روابط زیر برقرار است:

$$t_{IDS} > t_{DFS^*} > t_{BFS} > t_{UCS} > t_{AStar}$$

$$m_{UCS} > m_{BFS} > m_{DFS} > m_{IDS} > m_{AStar}$$

که در آن t_x و m_x به ترتیب زمان مصرفی و MemPeak الگوریتم x می باشند.

البته لازم به ذکر است که همانطورکه بیان شد الگوریتم مورد بررسی DFS نبود چرا که الگوریتم DFS بسیار طولانی برای حتی مثال اول اجرا می شد در نتیجه مشابه IDS ، با گذاشتن البته تنها یک حداکثر ارتفاع ، برای بیشتر مثال ها حل شد و به همین دلیل $t_{IDS} > t_{DFS^*}$ برقرار است.

پس همانطور که مشخص است الگوریتم A* بهترین الگوریتم در بین الگوریتم های مربوطه هم در زمان مصرفی و هم در MemPeak می باشد و همچنین در بین الگوریتم های دیگر جواب های بهتری را ارائه کرده است (با در نظر گرفتن طول دنباله اعمال جواب ، چرا که path cost برابر تعداد اعمال انجام شده در نظر گرفته شده است .)

و با این که الگوریتم UCS در مقام دوم (بعد از A*) از نظر زمان مصرفی را دارد اما از نظر MemPeak در رتبه آخر است و بیشترین MemPeak را به خود اختصاص داده است چرا که همانطور که می دانیم در هر مرحله g هر خانه در ان state ذخیره می شود.

لازم به ذکر است که واحد MemPeak برابر 10^{-6} مگابایت می باشد.