**به نام خدا**



دانشگاه اصفهان

گروه مهندسی کامپیوتر

**گزارش پروژه­ی نهایی هوش مصنوعی**

**موضوع:**

**حل هزارتو (Maze) به کمک الگوریتم‌های کلاسیک**

استاد:

خانم دکتر مرضیه حسینی

تهیه­کنندگان:

حدیث گنجی 963613080

محمد امینازاده 973653002

**زمستان 99**

# **مقدمه**

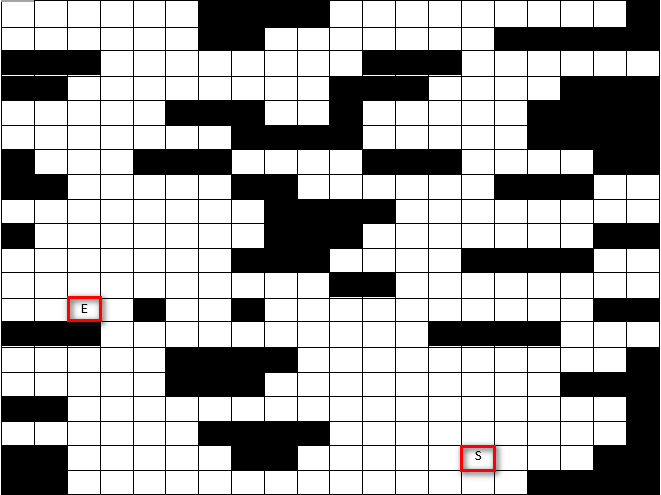
هدف ما از انجام این پروژه، پیاده­سازی یک عامل برای حل هزارتو (maze) است. برای انجام بازی سه نوع روش جستجو ارائه کرده­ایم که شامل جستجوی اول سطح (BFS) ، جستجوی عمیق شونده تکراری (IDFS) و جستجوی A\* است که جستجوی اول سطح و عمیق شونده تکراری از دسته جستجوی های ناآگاهانه و جستجوی A­­\* از دسته جستجوهای آگاهانه است. هرگاه عامل بازی را با هزینه کمتری با موفقیت به اتمام برساند، یعنی عملکرد بهینه تر و بهتری داشته است.

فضای حالت بازی به این گونه است که یک maze به صورت یک آرایه دو بعدی 20 در 20 در نظر می گیریم. هر سلول این آرایه یک موقعیت (x, y) دارد که از پایین چپ با شماره (0, 0) شروع شده و در بالا راست با (19, 19) خاتمه می­یابد. هر سلول از این آرایه یا خالی یا پر است که به ترتیب میتوان با رنگ های سفید و سیاه نشان داد. همچنین دو خانه شروع و پایان نیز در آرایه وجود دارند که به صورت تصادفی انتخاب می­شوند.

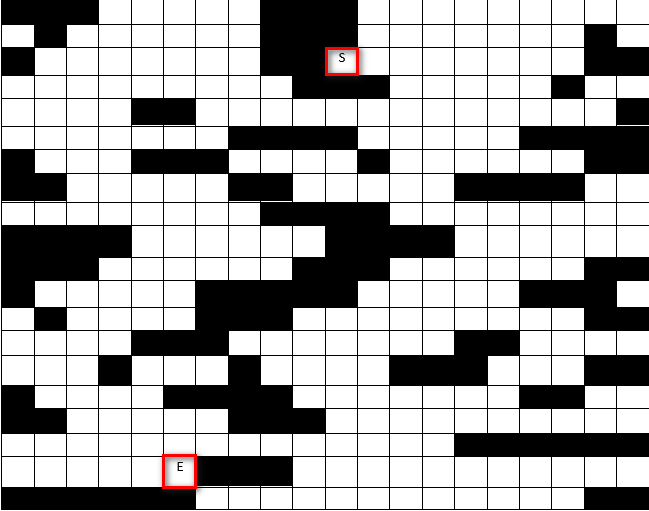
عامل می­تواند به بالا، پایین، چپ و راست حرکت کند اما امکان حرکت قطری ندارد. همچمین نمی­نواند به سلول­های پر وارد شود یا به خارج از maze حرکت کند. وظیفه­ی این عامل پیدا کردن مسیری از نقطه­ی شروع به نقطه­ی پایان است. هزینه­ی مسیر تعداد حرکت­های عامل در آن مسیر است.

از آن جا که انتخاب محل شروع و پایان بازی و شکل maze را به عهده کاربر گذاشتیم و در واقع هر سه پویا هستند، برای هر سه روش جستجویی که پیشنهاد کرده ایم، کاربر باید در ابتدا فضای حالت مورد نظر خود را در قالب یک فایل متنی که شامل 20 سطر است در اختیار برنامه نوشته شده قرار دهد. در این فایل هر سطر باید شامل 20 کاراکتر "." و یا "#" باشد و ما بین هر کدام یک "فاصله" قرار دهد. "." نشان دهنده خانه های سفید و "#" نشان دهنده خانه های سیاه است. علاوه بر این دو، کاربر باید در هر فایل متنی تنها یک خانه را به عنوان خانه شروع بازی و یک خانه را به عنوان خانه پایان بازی مشخص کند که برای این کار، "." و یا "#" را در خانه مورد نظرش به "X" برای نمایش خانه شروع (Start) و یا به "O" به عنوان خانه پایان بازی (End) تغییر دهد.

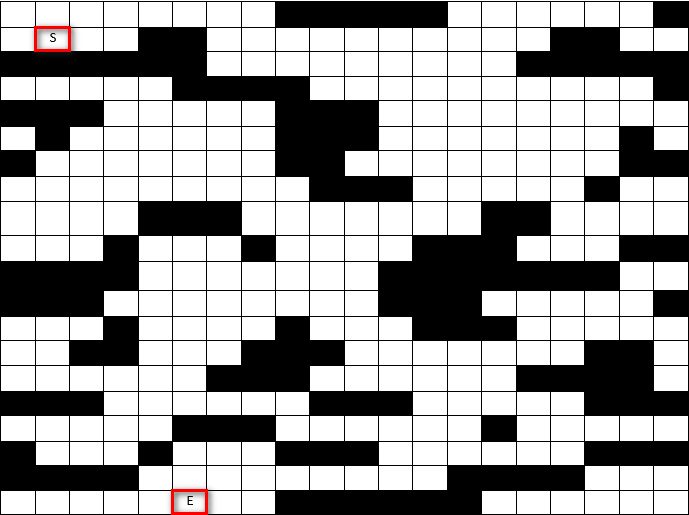
ما از سه تست کیس برای نمایش نوع کارکرد هر روش ارائه کرده ایم که فایل متنی آن ها به گزارش پیوست شده است (برای فهم بهتر قالب فایل و به درستی نوشتن فضای حالت مورد نظر خود، می توانید به آن ها مراجعه کنید). هر کدام از این تست کیس ها را در هر سه روش جستجو بررسی خواهیم کرد. این سه تست­کیس به­صورت زیر هستند.



TestCase1



TestCase2



TestCase3

در این گزارش، برای هر جستجو به ­صورت مجزا، ابتدا توضیحاتی کلی درمورد نحوه­ی عملکرد آن جستجو می­دهیم و سپس نحوه­ی پیاده­سازی عامل و الگوریتم، به­همراه کد و توابع به­کار رفته شده را به­تفصیل شرح می­دهیم. سپس خروجی واقعی کد در کنسول، به­نمایش درمی­آید و پس از آن مسیر طی شده توسط عامل را رسم می­کنیم. همچنین هزینه و نودهای بسط داده شده به­ترتیب بسط داده شدن لیست می­شوند.

برای پیاده­سازی این سه جستجو از زبان برنامه­نویسی پایتون استفاده شده است و هر الگوریتم بر روی سه Testcase متفاوت اجرا می­شوند و نتیجه شرح داده می­شود.

# **جستجوی اول سطح (Breadth First Search)**

## **نحوه­ی عملکرد**

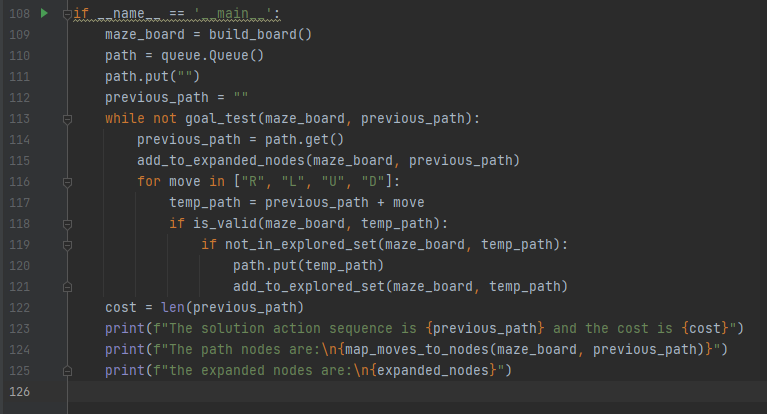
شیوه کلی کار این نوع جستجوی ناآگاهانه به این صورت است که گره هایی با کمترین عمق را زودتر از سایر گره ها بسط می دهد. درنتیجه ابتدا گره های درون یک سطر کم عمق را بسط می دهد و سپس به سراغ بسط دادن گره های سطر عمیق تر بعدی می رود و آنقدر این کار را تکرار می کند تا در عمیق ترین سطری که در حال بسط دادن آن است به هدف برسد و سپس جستجو را متوقف می کند.

این الگوریتم جستجو از یک مجموعه تحت عنوان مجموعه Frontier برای نگه داری گره هایی که در مرجله بعد می تواند بسط دهد، استفاده می کند. این مجموعه یک صف FIFO است. یعنی قدیمی ترین گره ای که به این مجموعه اضافه شده را زودتر از سایر گره ها از مجموعه خارج می کند و بسط می دهد.

این روش جستجو یک روش جستجوی کامل (اگر ضریب انشعاب در هر گره محدود باشد، حتما هدف را در عمق موردنظر پیدا می کند.) و بهینه (در صورتی که با افزایش عمق، هزینه مسیر افزایش یابد.) است. پیچیدگی زمانی و فضایی آن در بدترین حالت O(bd) است. ( که b ضریب انشعاب و d عمق هدف است).

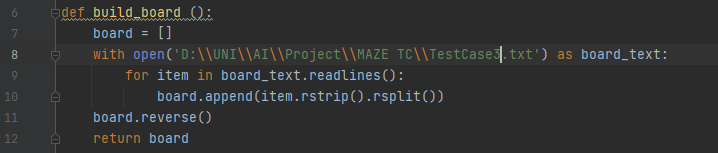
## **الگوریتم و کد پیاده­سازی**

برای پیاده­سازی جستجوی اول سطح به ­این ترتیب عمل می کنیم که یک صف (Queue) تعریف می کنیم که برای هر نود حالت FIFO را به­وجود آوریم. هر نود به­صورت دنباله­ای از اعمال (Actions) مشخص می­شود و تابعی مجزا برای نگاشت این اعمال به یک نود با مختصات خاص تعریف کرده­ایم. ابتدا دنباله­ی اعمال به­صورت یک رشته (String) خالی است. سپس تا زمانی که هدف یافت نشده است، به ازای تمام اعمال ممکن (R, L, U, D) ، اعمالی که قابل قبول هستند (از صفحه خارج نمی­شود یا به مانع برخورد نمی­کند یا به خانه­ای که قبلا بوده برنمی­گردد) را به رشته­ی عملی که در اول صف قرار دارد اضافه می­کند و آن عمل قدیمی را از صف حذف می­کند و سپس اعمال جدید تولید شده را به انتهای صف اضافه می­کند. با این روش، نودهایی (به­صورت دنباله­ی اعمال) که زودتر به صف اضافه شده­اند، زودتر بسط داده می­شوند و نودها به­صورت سطر به سطر بسط داده می­شوند. در هر مرحله پس از اضافه شدن یک عمل جدید به دنباله­ی اعمال گذشته، چک می­شود که آیا به خانه­ی هدف رسیده­ایم یا خیر. اگر رسیده باشیم حلقه قطع می­شود و آخرین رشته­ی تولید شده، همان رشته­ی اعمالی است که عامل را به هدف می­رسانند.

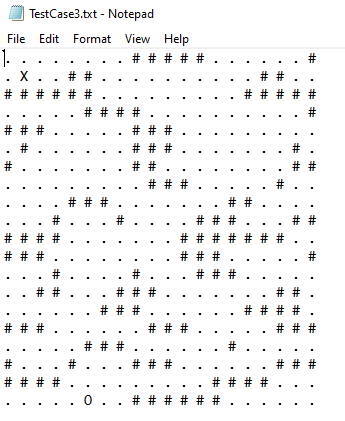


* اکنون توابع دیگری که در برنامه به­کار رفته­ است را شرح می­دهیم:

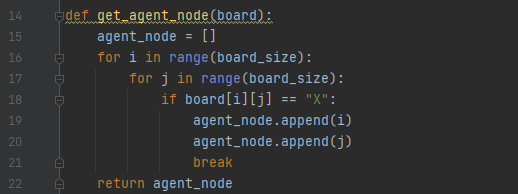
Build\_board(): این تابع برای ساخت صفحه­ی maze به­کار می­رود. صفحه به­صورت یک فایل .txt است. پس از خواندن فایل هر خط به صورت یک لیست از کاراکترها ذخیره می­شود. کل لیست­های خطوط درون یک لیست بزرگتر قرار دارند که صفحه­ی بازی را تشکیل می­دهند.



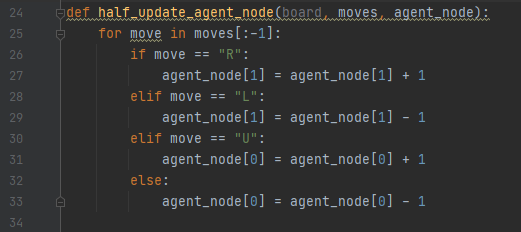
نمونه ای از یک فایل متنی ورودی به این تابع، به صورت زیر است:



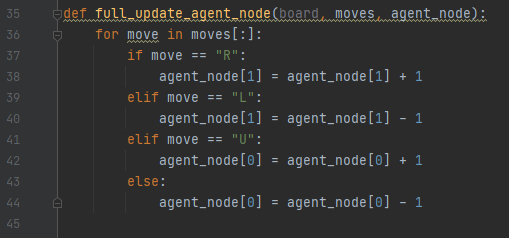
get\_agent\_node(board): این تابع وظیفه­ی یافتن نقطه­ی ابتدایی عامل بر روی صفحه را دارد.



Half\_update\_agent\_node(board, moves, agent\_node): این تابع وظیفه دارد مکان فعلی عامل را بر اساس مکان اولیه و رشته­ی اعمال تا آخرین عمل به­روز رسانی کند، اما آخرین عمل را به­حساب نمی­آورد.



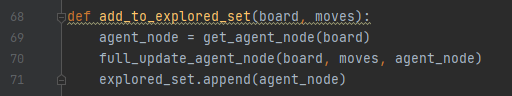
Full\_update\_agent\_node(board, moves, agent\_node): این تابع نیز وظیفه­ی به­روزرسانی مختصات عامل را دارد اما آخرین حرکت او را نیز به­حساب می­آورد.



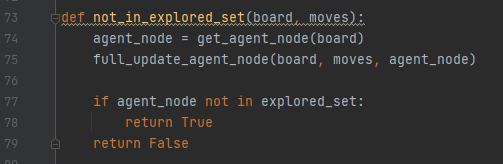
Is\_valid(board, moves): این تابع که یک مقدار True/False برمی­گرداند نشان می­دهد که اضافه شدن یک عمل به یک رشته از اعمال مجاز است یا خیر. مجاز بودن بر اساس بیرون نزدن از صفحه، یا برخورد نکردن با مانع یا برنگشتن به نودی که قبلا در آن بوده است، می­باشد. عملکرد این تابع به­این صورت است که مکان عامل را تا عمل آخر اما بدون احتساب آن حساب می­کند. سپس بر اساس آخرین عمل که کدام یک از چهار عمل باشد مشخص می­کند که آیا رفتن به آن نود مجاز است یا خیر.



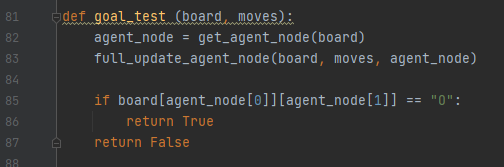
Add\_to\_explored\_set(board, moves): وظیفه­ی این تابع اضافه کردن یک نود به لیست نودهای ملاقات­شده است.



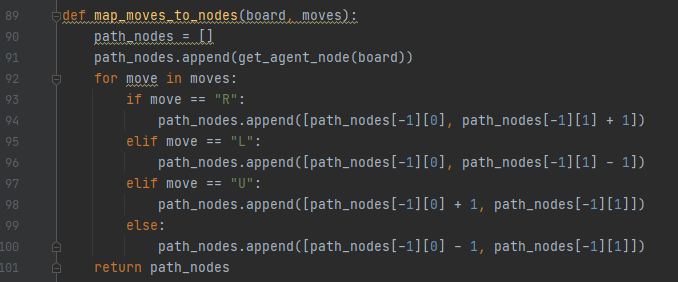
Not\_in\_explored\_set(board, moves): اگر یک نود در explored set نباشد مقدار True را برمی­گرداند. در غیر این صورت False برمی­گرداند.



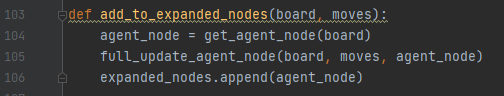
Goal\_test(board, moves): این تابع درصورتی مقدار True را برمی­گرداند که رشته اعمال شده کنونی، او را از خانه­ی ابتدایی به هدف برساند. درغیر این صورت False برمی­گرداند.



Map\_moves\_to\_nodes(board, moves): این تابع وظیفه­ی برگرداندن تمام نودهایی است که در نتیجه­ی آن رشته اعمال پیمایش می­شوند. در واقع وظیفه­ی نگاشت رشته اعمال به دنباله­ای از نودها را دارد.

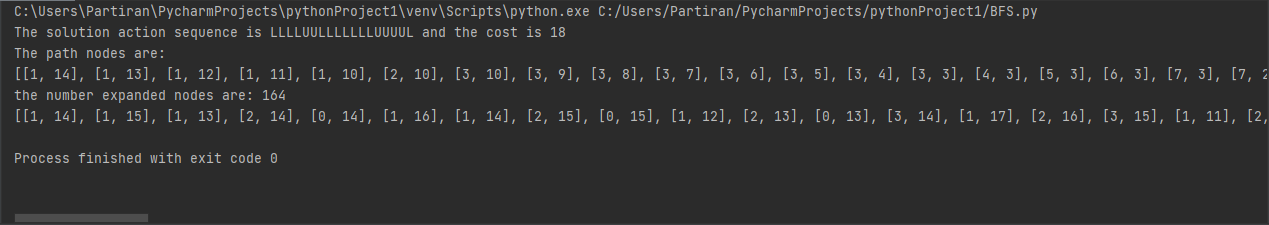


Add\_to\_expanded\_nodes(board, moves): این تابع آن نودی که توسط رشته اعمال شده از نود ابتدایی حرکت عامل ایجاد می­شود را به نودهای بسط داده شده اضافه می کند. (این تابع در راستای نمایش نودهای بسط داده شده در طول اجرای الگوریتم، در صورت پروژه است).



## **خروجی برنامه**

* **خروجی TestCase1 به صورت زیر است:**



مشاهده می­شود که رشته اعمال LLLLUULLLLLLLUUUUL راه حل است که هزینه­ی آن برابر 18 است.

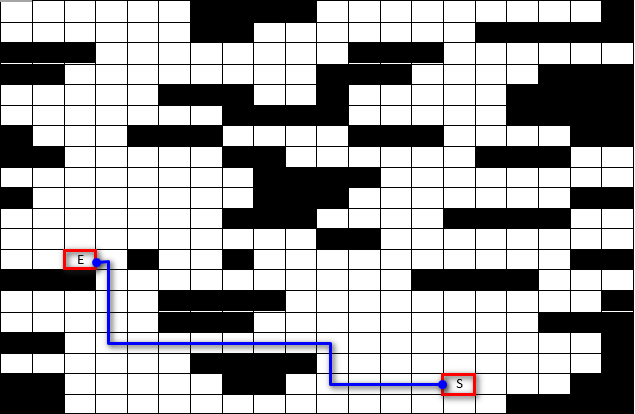
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[1, 14], [1, 13], [1, 12], [1, 11], [1, 10], [2, 10], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [4, 3], [5, 3], [6, 3], [7, 3], [7, 2]

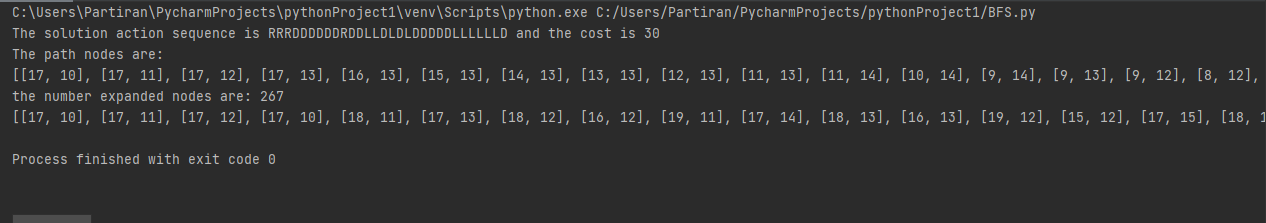
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 164 نود را شامل می­شود.

[1, 14], [1, 15], [1, 13], [2, 14], [0, 14], [1, 16], [1, 14], [2, 15], [0, 15], [1, 12], [2, 13], [0, 13], [3, 14], [1, 17], [2, 16], [3, 15], [1, 11], [2, 12], [0, 12], [3, 13], [4, 14], [2, 17], [3, 16], [4, 15], [1, 10], [2, 11], [0, 11], [3, 12], [4, 13], [5, 14], [2, 18], [3, 17], [4, 16], [5, 15], [1, 9], [2, 10], [0, 10], [3, 11], [4, 12], [5, 13], [3, 18], [5, 16], [0, 9], [3, 10], [4, 11], [5, 12], [5, 17], [0, 8], [3, 9], [4, 10], [5, 11], [6, 12], [5, 18], [6, 17], [0, 7], [3, 8], [4, 9], [5, 10], [6, 11], [7, 12], [6, 18], [7, 17], [0, 6], [3, 7], [4, 8], [5, 9], [6, 10], [7, 11], [7, 13], [8, 12], [6, 19], [7, 16], [8, 17], [0, 5], [1, 6], [3, 6], [6, 9], [7, 10], [7, 14], [8, 13], [9, 12], [7, 15], [8, 16], [8, 18], [0, 4], [1, 5], [3, 5], [6, 8], [7, 9], [8, 14], [9, 13], [9, 11], [10, 12], [8, 15], [8, 19], [9, 18], [0, 3], [1, 4], [2, 5], [3, 4], [6, 7], [7, 8], [8, 9], [10, 13], [9, 10], [10, 11], [11, 12], [9, 19], [0, 2], [1, 3], [2, 4], [3, 3], [4, 4], [6, 6], [8, 8], [10, 14], [11, 13], [12, 12], [1, 2], [2, 3], [3, 2], [4, 3], [5, 4], [6, 5], [7, 6], [8, 7], [10, 15], [11, 14], [12, 13], [12, 11], [2, 2], [4, 2], [5, 3], [6, 4], [7, 5], [8, 6], [10, 16], [11, 15], [12, 14], [12, 10], [2, 1], [4, 1], [5, 2], [6, 3], [8, 5], [9, 6], [10, 17], [11, 16], [13, 14], [12, 9], [13, 10], [2, 0], [4, 0], [5, 1], [7, 3], [8, 4], [9, 5], [10, 6], [11, 17], [13, 15], [14, 14], [13, 9], [5, 0], [7, 2]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase1 طی می­شود به­صورت زیر است:



* **خروجی TestCase2 به­صورت زیر است:**

****

مشاهده می­شود که رشته اعمال RRRDDDDDDRDDLLDLDLDDDDDLLLLLLD راه حل است که هزینه­ی آن برابر 30 است.

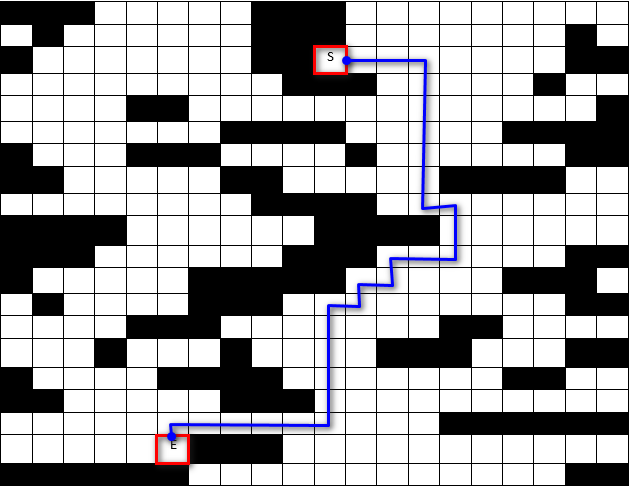
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [17, 13], [16, 13], [15, 13], [14, 13], [13, 13], [12, 13], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12], [8, 11], [7, 11], [7, 10], [6, 10], [5, 10], [4, 10], [3, 10], [2, 10], [2, 9], [2, 8], [2, 7], [2, 6], [2, 5], [2, 4], [1, 4]

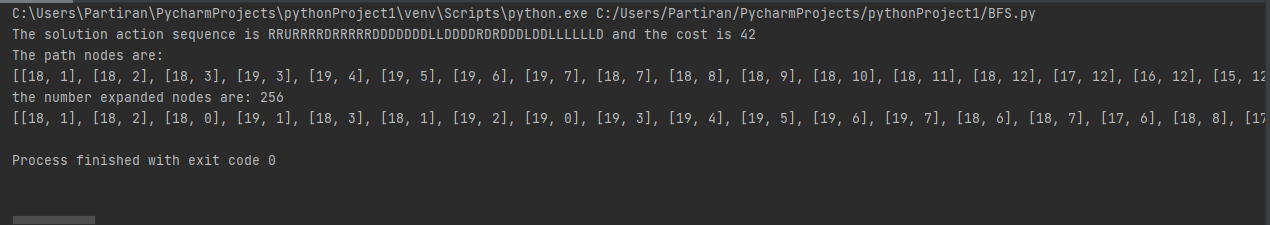
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 267 نود را شامل می­شود.

[17, 10], [17, 11], [17, 12], [17, 10], [18, 11], [17, 13], [18, 12], [16, 12], [19, 11], [17, 14], [18, 13], [16, 13], [19, 12], [15, 12], [17, 15], [18, 14], [16, 14], [19, 13], [15, 13], [15, 11], [14, 12], [17, 16], [18, 15], [16, 15], [19, 14], [15, 14], [14, 13], [15, 10], [14, 11], [13, 12], [17, 17], [18, 16], [16, 16], [19, 15], [15, 15], [14, 14], [13, 13], [15, 9], [12, 12], [18, 17], [19, 16], [15, 16], [14, 15], [13, 14], [12, 13], [15, 8], [12, 11], [11, 12], [19, 17], [15, 17], [13, 15], [11, 13], [15, 7], [16, 8], [12, 10], [19, 18], [15, 18], [13, 16], [11, 14], [15, 6], [16, 7], [12, 9], [13, 10], [19, 19], [16, 18], [13, 17], [11, 15], [10, 14], [16, 6], [14, 6], [17, 7], [13, 9], [18, 19], [16, 19], [11, 16], [10, 15], [9, 14], [16, 5], [17, 6], [14, 5], [18, 7], [13, 8], [11, 17], [10, 16], [9, 15], [9, 13], [8, 14], [16, 4], [17, 5], [18, 6], [14, 4], [19, 7], [13, 7], [11, 18], [10, 17], [9, 16], [8, 15], [9, 12], [8, 13], [7, 14], [16, 3], [17, 4], [18, 5], [19, 6], [14, 3], [11, 19], [12, 18], [10, 18], [9, 17], [7, 15], [8, 12], [7, 13], [16, 2], [17, 3], [15, 3], [18, 4], [19, 5], [14, 2], [13, 3], [12, 19], [10, 19], [7, 16], [8, 11], [7, 12], [6, 13], [16, 1], [17, 2], [15, 2], [18, 3], [19, 4], [14, 1], [13, 2], [12, 3], [7, 17], [6, 16], [7, 11], [6, 12], [16, 0], [17, 1], [15, 1], [18, 2], [19, 3], [14, 0], [13, 1], [12, 2], [12, 4], [11, 3], [6, 17], [5, 16], [7, 10], [6, 11], [15, 0], [11, 2], [12, 5], [11, 4], [6, 18], [5, 17], [5, 15], [7, 9], [6, 10], [5, 11], [11, 1], [12, 6], [11, 5], [10, 4], [6, 19], [4, 15], [6, 9], [5, 10], [4, 11], [11, 0], [11, 6], [10, 5], [9, 4], [4, 14], [3, 15], [6, 8], [5, 9], [4, 10], [4, 12], [3, 11], [11, 7], [10, 6], [9, 5], [9, 3], [8, 4], [4, 13], [3, 14], [3, 16], [6, 7], [5, 8], [4, 9], [3, 10], [3, 12], [2, 11], [10, 7], [9, 6], [8, 5], [8, 3], [7, 4], [3, 13], [3, 17], [2, 10], [2, 12], [1, 11], [10, 8], [9, 7], [7, 5], [8, 2], [7, 3], [2, 13], [3, 18], [2, 9], [1, 10], [1, 12], [0, 11], [10, 9], [9, 8], [8, 1], [7, 2], [6, 3], [1, 13], [3, 19], [4, 18], [2, 8], [1, 9], [0, 10], [0, 12], [6, 2], [1, 14], [0, 13], [4, 19], [2, 7], [1, 8], [0, 9], [6, 1], [5, 2], [1, 15], [0, 14], [2, 6], [0, 8], [6, 0], [5, 1], [4, 2], [1, 16], [0, 15], [2, 5], [3, 6], [0, 7], [7, 0], [5, 0], [4, 1], [4, 3], [3, 2], [1, 17], [0, 16], [2, 4], [3, 5], [0, 6], [4, 4], [3, 3], [2, 2], [1, 18], [0, 17], [2, 3], [3, 4], [1, 4]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase2 طی می­شود به­صورت زیر است:



* **خروجی TestCase3 به صورت زیر است:**



مشاهده می­شود که رشته اعمال RRURRRRDRRRRRDDDDDDDLLDDDDRDRDDDLDDLLLLLLD راه حل است که هزینه­ی آن برابر 42 است.

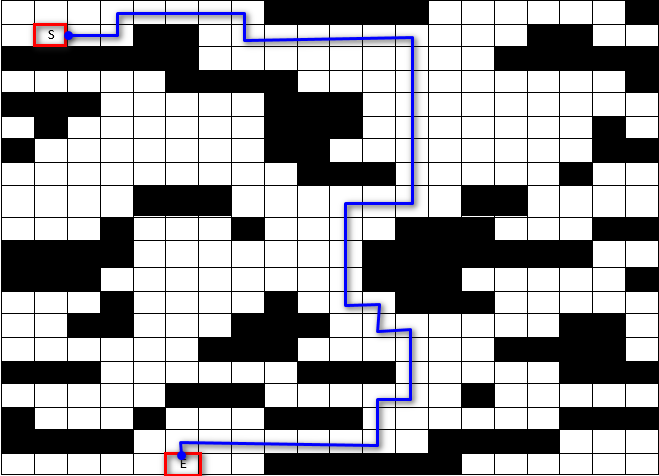
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [17, 12], [16, 12], [15, 12], [14, 12], [13, 12], [12, 12], [11, 12], [11, 11], [11, 10], [10, 10], [9, 10], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [5, 12], [4, 12], [3, 12], [3, 11], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [0, 5]

همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 256 نود را شامل می­شود.

[18, 1], [18, 2], [18, 0], [19, 1], [18, 3], [18, 1], [19, 2], [19, 0], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [18, 8], [17, 7], [18, 9], [17, 8], [18, 10], [17, 9], [18, 11], [17, 10], [16, 9], [18, 12], [17, 11], [16, 10], [18, 13], [17, 12], [16, 11], [18, 14], [19, 13], [17, 13], [16, 12], [15, 11], [18, 15], [19, 14], [17, 14], [16, 13], [15, 12], [14, 11], [19, 15], [16, 14], [15, 13], [14, 12], [13, 11], [19, 16], [16, 15], [15, 14], [14, 13], [13, 12], [13, 10], [19, 17], [16, 16], [15, 15], [14, 14], [13, 13], [12, 12], [19, 18], [16, 17], [15, 16], [14, 15], [13, 14], [12, 13], [11, 12], [18, 18], [16, 18], [15, 17], [14, 16], [13, 15], [12, 14], [11, 13], [11, 11], [18, 19], [15, 18], [14, 17], [13, 16], [12, 15], [11, 10], [10, 11], [15, 19], [13, 17], [12, 16], [11, 9], [10, 10], [14, 19], [11, 16], [11, 8], [10, 9], [9, 10], [11, 17], [10, 16], [11, 7], [12, 8], [10, 8], [9, 9], [8, 10], [11, 18], [10, 17], [10, 15], [12, 7], [9, 8], [8, 9], [7, 10], [11, 19], [12, 18], [12, 6], [13, 7], [9, 7], [8, 8], [7, 9], [7, 11], [6, 10], [12, 19], [12, 5], [13, 6], [14, 7], [9, 6], [8, 7], [6, 11], [5, 10], [12, 4], [13, 5], [14, 6], [15, 7], [9, 5], [10, 6], [8, 6], [7, 7], [6, 12], [5, 11], [5, 9], [12, 3], [13, 4], [14, 5], [15, 6], [9, 4], [10, 5], [8, 5], [7, 6], [6, 13], [5, 12], [12, 2], [13, 3], [11, 3], [14, 4], [15, 5], [10, 4], [8, 4], [7, 5], [6, 6], [6, 14], [5, 13], [4, 12], [12, 1], [13, 2], [11, 2], [14, 3], [15, 4], [8, 3], [7, 4], [6, 5], [6, 15], [5, 14], [4, 13], [3, 12], [12, 0], [13, 1], [11, 1], [14, 2], [10, 2], [15, 3], [16, 4], [6, 4], [5, 5], [6, 16], [7, 15], [4, 14], [3, 13], [3, 11], [2, 12], [11, 0], [10, 1], [16, 3], [5, 4], [4, 5], [7, 16], [8, 15], [4, 15], [2, 13], [3, 10], [2, 11], [1, 12], [10, 0], [16, 2], [5, 3], [4, 4], [4, 6], [7, 17], [8, 16], [8, 14], [4, 16], [3, 15], [2, 14], [3, 9], [1, 11], [16, 1], [5, 2], [4, 3], [3, 4], [4, 7], [7, 18], [8, 17], [3, 16], [2, 15], [3, 8], [1, 10], [16, 0], [5, 1], [3, 3], [4, 8], [7, 19], [8, 18], [3, 17], [2, 16], [1, 9], [5, 0], [6, 1], [3, 2], [2, 3], [6, 19], [9, 18], [3, 18], [1, 8], [6, 0], [7, 1], [3, 1], [2, 2], [5, 19], [9, 19], [3, 19], [1, 7], [7, 0], [7, 2], [3, 0], [2, 1], [1, 6], [2, 7], [0, 7], [1, 5], [2, 6], [0, 6], [1, 4], [2, 5], [0, 5]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase3 طی می­شود به­صورت زیر است:



# **جستجوی عمیق­شونده­ی تکراری**

## **نحوه­ی عملکرد**

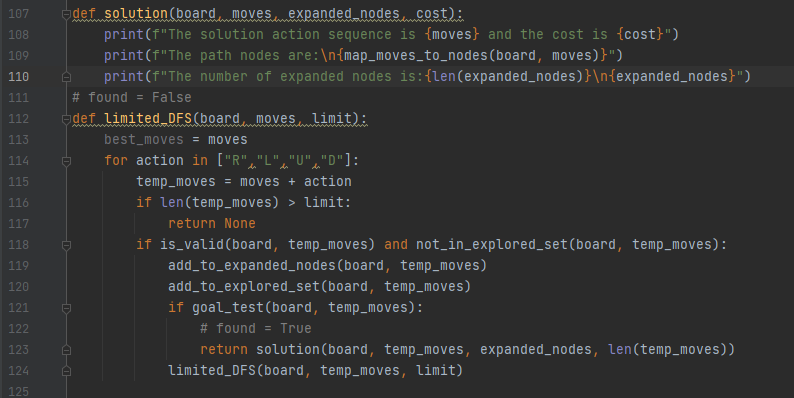
این روش جستجوی ناآگاهانه، همان جستجوی اول عمق (DFS) است که عمق را به صورت تدریجی زیاد می کند تا در کم عمق ترین سطح ممکن، گره هدف را بیابد. این روش جستجو نیز از یک مجموعه تحت عنوان مجموعه Frontier برای گره های کاوش نشده استفاده می کند. با این تفاوت که یک صف LIFO است. یعنی در هر مرحله آخرین گره ای که به مجموعه اضافه شده را برای بسط دادن انتخاب می کند.

این روش جستجو یک روش کامل (در صورت متناهی بودن ضریب انشعاب) و بهینه (در صورت افزایش هزینه مسیر با افزایش عمق گره) است. پیچیدگی زمانی این الگوریتم O(bd) و پیچیدگی فضایی آن O(bd) است. (که d عمق هدف و b ضریب انشعاب است.)

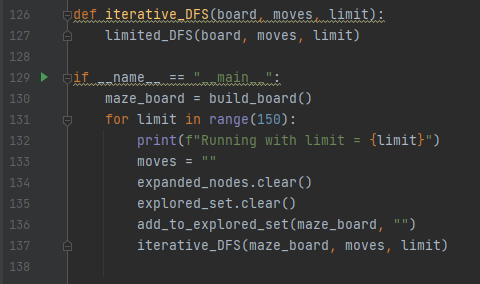
درواقع این الگوریتم مشکل نمایی بودن پیچیدگی فضایی روش جستجوی اول سطح که منجر به نیاز به حافظه بسیار زیاد می شد را حل کرد.

## **الگوریتم و کد پیاده­سازی**

برای پیاده­سازی این الگوریتم ابتدا الگوریتم و تابع جستجوی عمق محدود را پیاده­سازی کردیم (limited\_DFS). الگوریتم این تابع به­این صورت است که ابتدا رشته­ی اعمال خالی است. سپس به­ازای تمام اعمال ممکن، آن عمل به آخرین رشته اعمال تولید شده اضافه می­شود. سپس چک می­شود که طول رشته (که بیانگر عمقی است که عامل پیش رفته است.) بیشتر از محدودیت عمق (limit) نشود. سپس چک می­شود که آیا رشته اعمال تولید شده مجاز است یا خیر (شرط مجاز بودن در بخش قبل شرح داده شده است.). اگر مجاز بود، آن نود جدید به explored set و همچنین لیست نودهای بسط داده شده اضافه می­شود. پس از آن چک می­شود که آیا آن نود هدف است یا خیر. در صورت هدف بودن تابع خاتمه می­یابد و یک تابع دیگر به­نام solution(board, moves, limit) صدا زده می­شود و راه حل چاپ می­شود. درصورت هدف نبودن آن نود، تابع limited\_DFS به صورت بازگشتی برای همان نود صدا زده می­شود. به­این صورت نودها به­صورت عمقی به پیش می­روند.



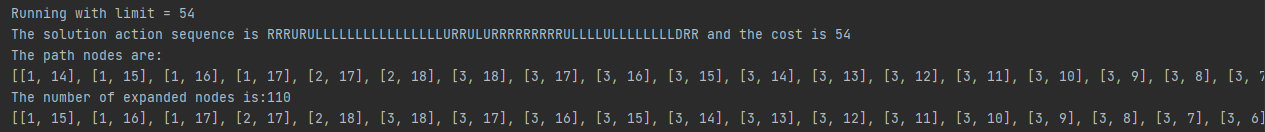
سپس در تابع جستجوی عمیق­شونده­ی تکراری (Iterative\_DFS) تابع عمق محدود را برای یک محدودیت عمق خاص (limit) صدا می­زنیم. سپس main به ازای تمام عمق­ها (معمولا از عمق 200 به بالا برای این محیط احتیاج نیست.) این تابع صدا زده می­شود. در خروجی به ازای هر عمق اگر هدف یافت شود راه حل چاپ می­شود. در غیر این صورت به سراغ عمق بعدی می­رود.



از دیگر توابع که در بخش اول سطح استفاده شده است نیز استفاده شده است.

## **خروجی برنامه**

* **خروجی TestCase1 به­صورت زیر است:**

****

همانطور که مشاهده می­شود هدف را در limit = 54 یافته است و برای عمق­های قبلی به نتیجه نرسیده است. رشته اعمال RRRURULLLLLLLLLLLLLLLLURRULURRRRRRRRRULLLLULLLLLLLLDRR راه حل است که هزینه­ی آن برابر 54 است.

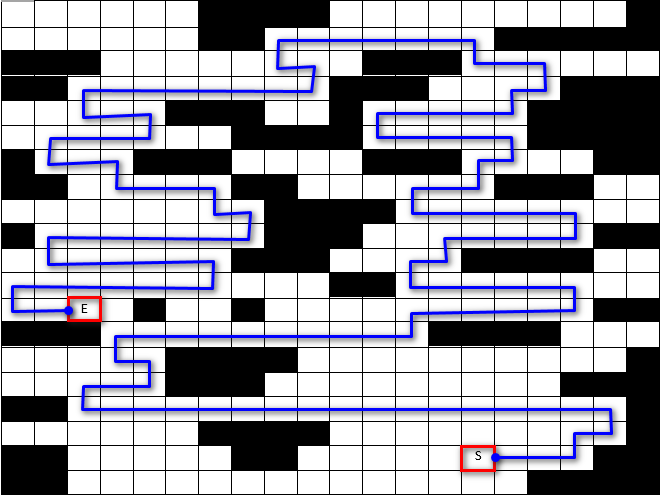
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[1, 14], [1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [3, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 3], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9], [6, 10], [6, 11], [6, 12], [7, 12], [7, 11], [7, 10], [7, 9], [7, 8], [8, 8], [8, 7], [8, 6], [8, 5], [8, 4], [8, 3], [8, 2], [8, 1], [8, 0], [7, 0], [7, 1], [7, 2]

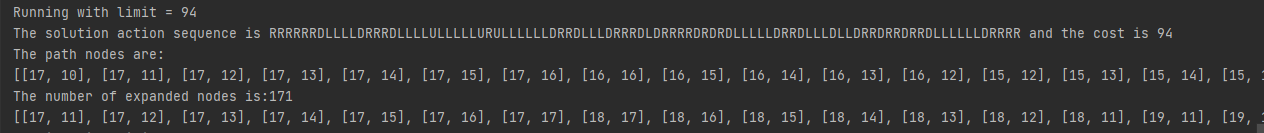
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 110 نود را شامل می­شود.

[1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [3, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 3], [5, 2], [5, 1], [5, 0], [4, 0], [4, 1], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9], [6, 10], [6, 11], [6, 12], [7, 12], [7, 13], [7, 14], [7, 15], [7, 16], [7, 17], [8, 17], [8, 18], [8, 19], [9, 19], [9, 18], [8, 16], [8, 15], [8, 14], [8, 13], [8, 12], [9, 12], [9, 13], [10, 13], [10, 14], [10, 15], [11, 14], [10, 12], [10, 11], [11, 12], [11, 13], [12, 13], [9, 11], [9, 10], [6, 17], [6, 18], [6, 19], [5, 18], [5, 17], [5, 16], [5, 15], [5, 14], [5, 13], [5, 12], [5, 11], [5, 10], [4, 11], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [7, 11], [7, 10], [7, 9], [7, 8], [8, 8], [8, 9], [8, 7], [8, 6], [8, 5], [8, 4], [8, 3], [8, 2], [8, 1], [8, 0], [9, 0], [9, 1], [9, 2], [10, 1], [7, 0], [7, 1], [7, 2]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase1 طی می­شود به­صورت زیر است:



* **خروجی TestCase2 به­صورت زیر است.**



همانطور که مشاهده می­شود هدف را در limit = 94 یافته است و برای عمق­های قبلی به نتیجه نرسیده است. رشته اعمال زیر راه حل است که هزینه­ی آن برابر 94 است.

RRRRRRDLLLLDRRRDLLLLULLLLLURULLLLLLDRRDLLLDRRRDLDRRRRDRDRDLLLLLDRRDLLLDLLDRRDRRDRRDLLLLLLDRRRR

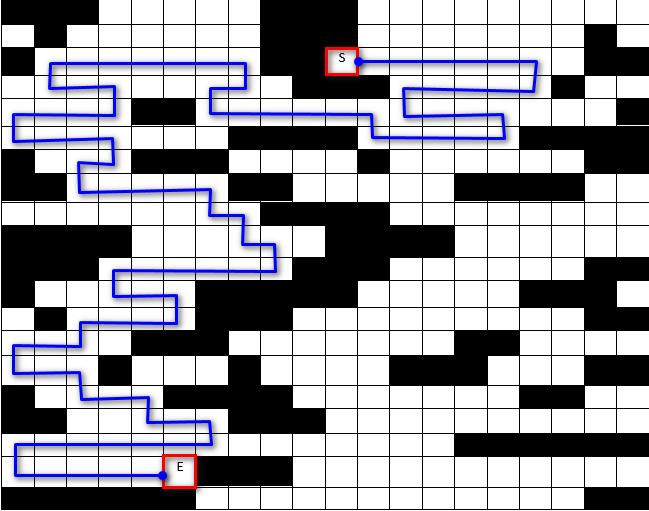
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [17, 13], [17, 14], [17, 15], [17, 16], [16, 16], [16, 15], [16, 14], [16, 13], [16, 12], [15, 12], [15, 13], [15, 14], [15, 15], [14, 15], [14, 14], [14, 13], [14, 12], [14, 11], [15, 11], [15, 10], [15, 9], [15, 8], [15, 7], [15, 6], [16, 6], [16, 7], [17, 7], [17, 6], [17, 5], [17, 4], [17, 3], [17, 2], [17, 1], [16, 1], [16, 2], [16, 3], [15, 3], [15, 2], [15, 1], [15, 0], [14, 0], [14, 1], [14, 2], [14, 3], [13, 3], [13, 2], [12, 2], [12, 3], [12, 4], [12, 5], [12, 6], [11, 6], [11, 7], [10, 7], [10, 8], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [9, 3], [8, 3], [8, 4], [8, 5], [7, 5], [7, 4], [7, 3], [7, 2], [6, 2], [6, 1], [6, 0], [5, 0], [5, 1], [5, 2], [4, 2], [4, 3], [4, 4], [3, 4], [3, 5], [3, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [2, 1], [2, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4]

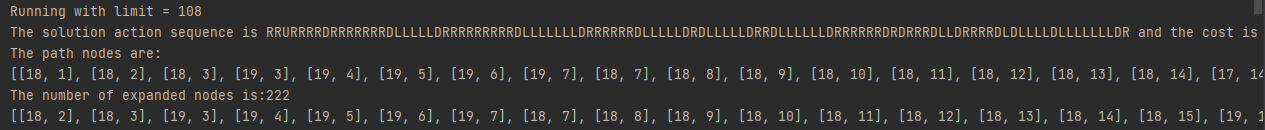
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 171 نود را شامل می­شود.

[17, 11], [17, 12], [17, 13], [17, 14], [17, 15], [17, 16], [17, 17], [18, 17], [18, 16], [18, 15], [18, 14], [18, 13], [18, 12], [18, 11], [19, 11], [19, 12], [19, 13], [19, 14], [19, 15], [19, 16], [19, 17], [19, 18], [19, 19], [18, 19], [16, 16], [16, 15], [16, 14], [16, 13], [16, 12], [15, 12], [15, 13], [15, 14], [15, 15], [15, 16], [15, 17], [15, 18], [16, 18], [16, 19], [14, 15], [14, 14], [14, 13], [14, 12], [14, 11], [15, 11], [15, 10], [15, 9], [15, 8], [15, 7], [15, 6], [16, 6], [16, 7], [16, 8], [17, 7], [17, 6], [17, 5], [17, 4], [17, 3], [17, 2], [17, 1], [16, 1], [16, 2], [16, 3], [16, 4], [16, 5], [15, 3], [15, 2], [15, 1], [15, 0], [16, 0], [14, 0], [14, 1], [14, 2], [14, 3], [14, 4], [14, 5], [14, 6], [13, 3], [13, 2], [13, 1], [12, 2], [12, 3], [12, 4], [12, 5], [12, 6], [11, 6], [11, 7], [10, 7], [10, 8], [10, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [9, 3], [8, 3], [8, 4], [8, 5], [7, 5], [7, 4], [7, 3], [7, 2], [8, 2], [8, 1], [6, 2], [6, 3], [6, 1], [6, 0], [7, 0], [5, 0], [5, 1], [5, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 5], [5, 6], [3, 4], [3, 5], [3, 6], [2, 6], [2, 7], [2, 8], [2, 9], [2, 10], [2, 11], [2, 12], [2, 13], [3, 13], [3, 14], [3, 15], [3, 16], [4, 15], [4, 14], [4, 13], [3, 12], [3, 11], [3, 10], [4, 11], [4, 12], [1, 13], [1, 14], [1, 15], [1, 16], [0, 15], [0, 14], [0, 13], [1, 12], [1, 11], [1, 10], [0, 11], [0, 12], [1, 9], [1, 8], [0, 8], [0, 9], [0, 10], [0, 7], [0, 6], [2, 5], [2, 4], [2, 3], [2, 2], [2, 1], [2, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase2 طی می­شود به­صورت زیر است:



* **خروجی TestCase3 به­صورت زیر است:**



همانطور که مشاهده می­شود هدف را در limit = 108 یافته است و برای عمق­های قبلی به نتیجه نرسیده است. رشته اعمال زیر راه حل است که هزینه­ی آن برابر 108 است.

RRURRRRDRRRRRRRDLLLLLDRRRRRRRRRDLLLLLLLDRRRRRRDLLLLLDRDLLLLLDRRDLLLLLLDRRRRRRDRDRRRDLLDRRRRDLDLLLLDLLLLLLLDR

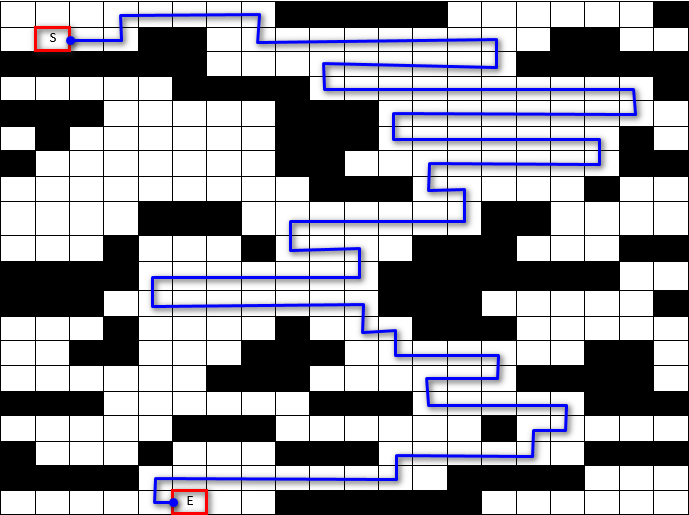
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [17, 14], [17, 13], [17, 12], [17, 11], [17, 10], [17, 9], [16, 9], [16, 10], [16, 11], [16, 12], [16, 13], [16, 14], [16, 15], [16, 16], [16, 17], [16, 18], [15, 18], [15, 17], [15, 16], [15, 15], [15, 14], [15, 13], [15, 12], [15, 11], [14, 11], [14, 12], [14, 13], [14, 14], [14, 15], [14, 16], [14, 17], [13, 17], [13, 16], [13, 15], [13, 14], [13, 13], [13, 12], [12, 12], [12, 13], [11, 13], [11, 12], [11, 11], [11, 10], [11, 9], [11, 8], [10, 8], [10, 9], [10, 10], [9, 10], [9, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [6, 13], [6, 14], [5, 14], [5, 13], [5, 12], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [3, 16], [3, 15], [2, 15], [2, 14], [2, 13], [2, 12], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [1, 4], [0, 4], [0, 5]

همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 222 نود را شامل می­شود.

[18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [18, 15], [19, 15], [19, 16], [19, 17], [19, 18], [18, 18], [18, 19], [19, 14], [19, 13], [17, 14], [17, 13], [17, 12], [17, 11], [17, 10], [17, 9], [17, 8], [17, 7], [17, 6], [18, 6], [16, 9], [16, 10], [16, 11], [16, 12], [16, 13], [16, 14], [16, 15], [16, 16], [16, 17], [16, 18], [15, 18], [15, 19], [14, 19], [15, 17], [15, 16], [15, 15], [15, 14], [15, 13], [15, 12], [15, 11], [14, 11], [14, 12], [14, 13], [14, 14], [14, 15], [14, 16], [14, 17], [13, 17], [13, 16], [13, 15], [13, 14], [13, 13], [13, 12], [13, 11], [13, 10], [12, 12], [12, 13], [12, 14], [12, 15], [12, 16], [11, 16], [11, 17], [11, 18], [11, 19], [12, 19], [12, 18], [10, 17], [10, 16], [10, 15], [11, 13], [11, 12], [11, 11], [11, 10], [11, 9], [11, 8], [11, 7], [12, 7], [12, 8], [12, 6], [12, 5], [12, 4], [12, 3], [12, 2], [12, 1], [12, 0], [11, 0], [11, 1], [11, 2], [11, 3], [10, 2], [10, 1], [10, 0], [13, 1], [13, 2], [13, 3], [13, 4], [13, 5], [13, 6], [13, 7], [14, 7], [14, 6], [14, 5], [14, 4], [14, 3], [14, 2], [15, 3], [15, 4], [15, 5], [15, 6], [15, 7], [16, 4], [16, 3], [16, 2], [16, 1], [16, 0], [10, 8], [10, 9], [10, 10], [10, 11], [9, 10], [9, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [10, 4], [10, 5], [10, 6], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [6, 13], [6, 14], [6, 15], [6, 16], [7, 16], [7, 17], [7, 18], [7, 19], [6, 19], [5, 19], [8, 18], [8, 17], [8, 16], [8, 15], [8, 14], [7, 15], [9, 18], [9, 19], [5, 14], [5, 13], [5, 12], [5, 11], [5, 10], [5, 9], [6, 10], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [3, 16], [3, 17], [3, 18], [3, 19], [3, 15], [2, 15], [2, 16], [2, 14], [2, 13], [2, 12], [2, 11], [3, 11], [3, 12], [3, 13], [3, 10], [3, 9], [3, 8], [4, 8], [4, 7], [4, 6], [4, 5], [4, 4], [4, 3], [5, 4], [3, 4], [5, 5], [6, 5], [1, 11], [1, 12], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [1, 4], [0, 4], [0, 5]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase3 طی می­شود به­صورت زیر است:



# **جستجوی A\***

## **نحوه­ی عملکرد**

این الگوریتم جستجوی آگاهانه از دو تابع برای پیدا کردن بهینه­ترین مسیر استفاده می­کند. اگر از اولین گره (ریشه درخت جستجو) شروع کرده باشد و اکنون در گره n باشد، یک تابع g(n) که هزینه مسیر از گره شروع تا گره n است و یک تابع h(n) که تخمینی از ارزان­ترین مسیر از گره کنونی n تا گره هدف است که به آن تابع اکتشافی (heuristic) گفته می شود را ارائه می­کند و برای گره n یک تابع f(n) به­صورت زیر می­سازد:

f(n) = g(n) + h(n)

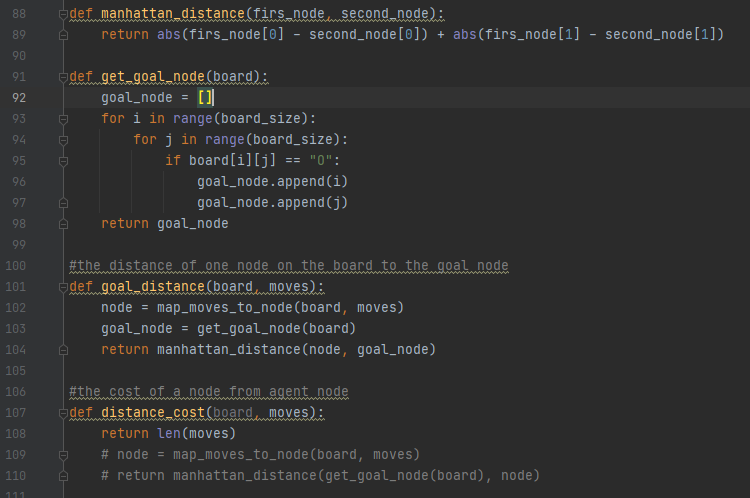
و در هر مرحله گره­ای را بسط می­دهد که تابع f برای آن کمینه باشد. دلیل آگاهانه بودن این نوع جستجو، وجود همین تابع اکتشافی است که چشم انداز حدودی از مسیر آینده به عامل می دهد.

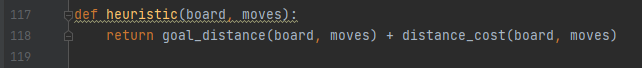
اگر تابع اکتشافی هیچگاه هزینه رسیدن به هدف را از مقدار واقعی آن تخمین نزند یک تابع اکتشافی قابل قبول است و تابع اکتشافی h(n) سازگار است اگر برای هر گره n و هر پسین آن گره، n’ که با عمل a تولید شده است، هزینه­ی برآورد شده رسیدن به هدف از n بزرگتر از هزینه گام رسیدن به n’ به علاوه هزینه برآورد شده رسیدن به هدف از n’ نباشد.

این الگوریتم یک روش جستجوی کامل (در صورتی که ضریب انشعاب متناهی باشد.) و بهینه (اگر تابع heuristic در جستجوی گراف سازگار و در جستجوی درخت قابل قبول باشد.) است. پیچیدگی زمانی و فضایی آن O(b𝜖𝑑 ) است. (𝜖 میزان خطای نسبی تابع اکتشافی است.)

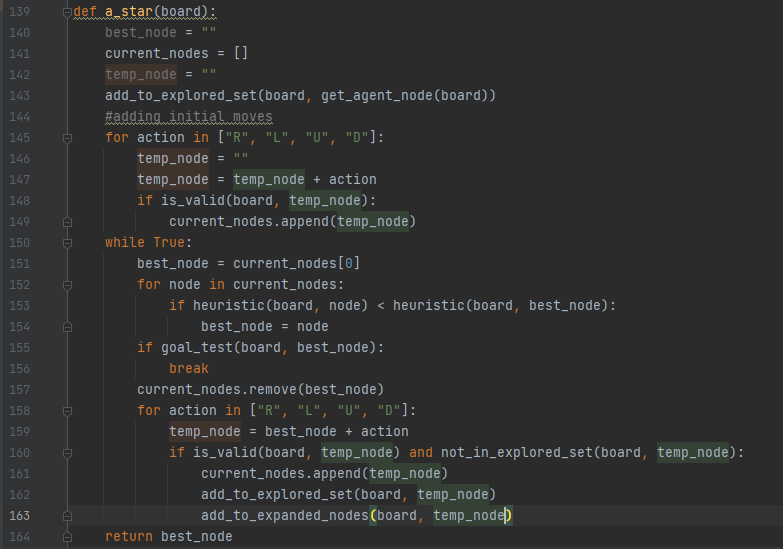
## **الگوریتم و کد پیاده­سازی**

همانطور که در بالا شرح داده شد، الگوریتم به­این صورت پیاده­سازی شده که در هر مرحله نود با کمترین f(n) را بسط می­دهد. در کد تابع f(n) را heuristic نامیده­ایم. همچنین تابع g(n) را distance\_cost و تابع h(n) را goal\_distance نامیده­ایم. تابع هیوریستیک پیشنهادی ما (goal\_distance) فاصله­ی منهتن نود n تا نود هدف برمی­گرداند. پیاده­سازی این سه تابع به­شکل زیر بوده است.



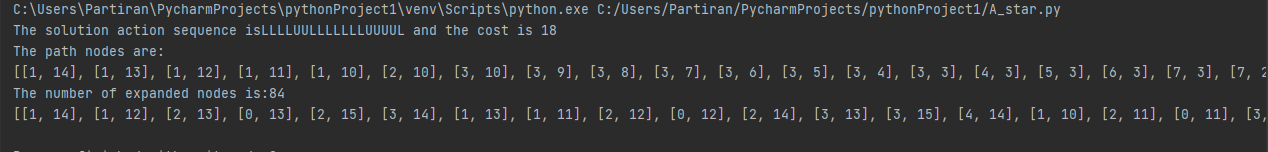


تابع اصلی این جستجو a\_star نام دارد. یک لیست از نودهای فعلی که آماده­ی بسط داده شدن هستند، وجود دارد که در ابتدا نودهای ابتدایی بر اساس اعمال مجاز از نود اولیه­ی حرکت عامل به این لیست اضافه می­شود. سپس در یک حلقه­ی بی­نهایت هر سری از میان نودهای آماده­ی بسط داده شدن بهترین نود که کمترین heuristic را داراست انتخاب می­شود. سپس چک می­شود که ایا نود هدف است یا خیر. اگر هدف بود حلقه break می­شود. درغیر این صورت آن نود بسط داده می­شود و از لیست نودهای آماده­ی بسط داده شدن حذف می­شود. سپس تمام نودهای فرزند این نود که با هرکدام از چهار عمل، قابل دسترسی هستند، به لیست نودهای آماده بسط داده شدن اضافه می­شوند. این کار آنقدر ادامه می­یابد تا هدف یافت شود و return شود.



## **خروجی برنامه**

* **خروجی TestCase1 به­صورت زیر است:**



مشاهده می­شود که رشته اعمال LLLLUULLLLLLLUUUUL راه حل است که هزینه­ی آن برابر 18 است.

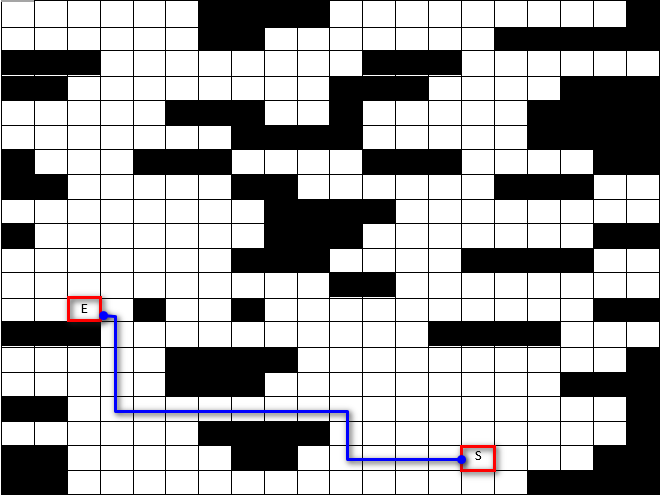
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[1, 14], [1, 13], [1, 12], [1, 11], [1, 10], [2, 10], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [4, 3], [5, 3], [6, 3], [7, 3], [7, 2]

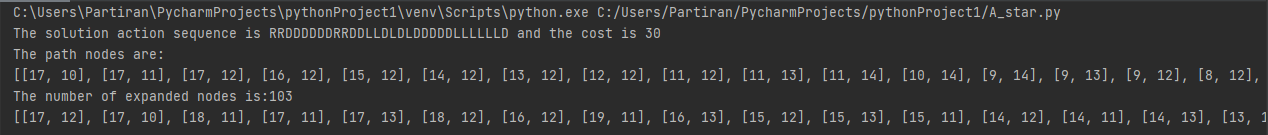
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 84 نود را شامل می­شود.

[1, 14], [1, 12], [2, 13], [0, 13], [2, 15], [3, 14], [1, 13], [1, 11], [2, 12], [0, 12], [2, 14], [3, 13], [3, 15], [4, 14], [1, 10], [2, 11], [0, 11], [3, 12], [4, 13], [4, 15], [5, 14], [1, 9], [2, 10], [0, 10], [3, 11], [4, 12], [5, 13], [5, 15], [0, 9], [3, 10], [4, 11], [5, 12], [3, 9], [4, 10], [5, 11], [6, 12], [3, 8], [4, 9], [5, 10], [6, 11], [7, 12], [3, 7], [4, 8], [5, 9], [6, 10], [7, 11], [7, 13], [8, 12], [3, 6], [6, 9], [7, 10], [3, 5], [6, 8], [7, 9], [3, 4], [2, 5], [6, 7], [7, 8], [8, 9], [3, 3], [4, 4], [2, 4], [6, 6], [8, 8], [3, 2], [4, 3], [2, 3], [5, 4], [6, 5], [7, 6], [4, 2], [2, 2], [5, 3], [6, 4], [7, 5], [8, 6], [4, 1], [5, 2], [6, 3], [8, 5], [5, 1], [7, 3], [7, 2], [8, 3]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase1 طی می­شود به­صورت زیر است:



* **خروجی TestCase2 به­صورت زیر است:**



مشاهده می­شود که رشته اعمال RRDDDDDDRRDDLLDLDLDDDDDLLLLLLD راه حل است که هزینه­ی آن برابر 30 است.

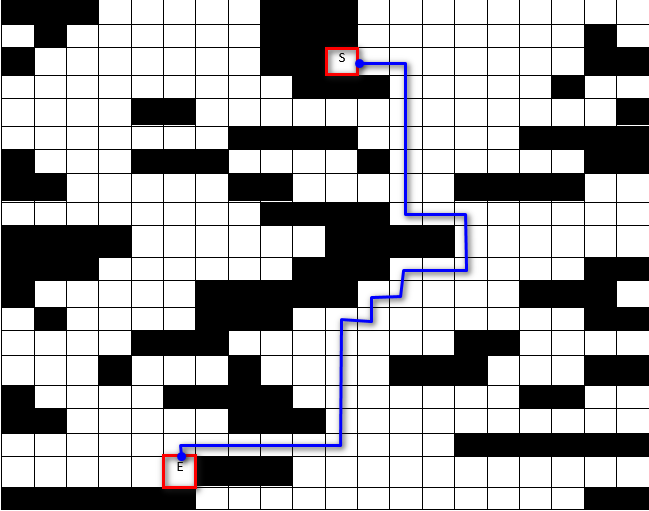
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [16, 12], [15, 12], [14, 12], [13, 12], [12, 12], [11, 12], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12], [8, 11], [7, 11], [7, 10], [6, 10], [5, 10], [4, 10], [3, 10], [2, 10], [2, 9], [2, 8], [2, 7], [2, 6], [2, 5], [2, 4], [1, 4]

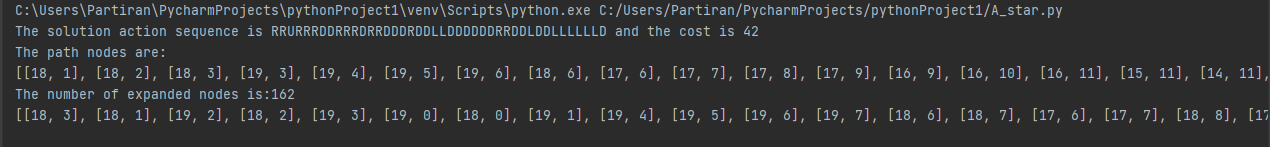
همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 103 نود را شامل می­شود.

[17, 12], [17, 10], [18, 11], [17, 11], [17, 13], [18, 12], [16, 12], [19, 11], [16, 13], [15, 12], [15, 13], [15, 11], [14, 12], [14, 11], [14, 13], [13, 12], [13, 13], [12, 12], [12, 13], [12, 11], [11, 12], [12, 10], [11, 13], [12, 9], [13, 10], [13, 9], [17, 14], [18, 13], [19, 12], [16, 14], [15, 14], [14, 14], [13, 14], [11, 14], [13, 8], [13, 7], [17, 15], [18, 14], [19, 13], [16, 15], [15, 15], [14, 15], [13, 15], [11, 15], [10, 14], [10, 15], [9, 14], [9, 15], [9, 13], [8, 14], [9, 12], [8, 13], [8, 15], [7, 14], [8, 12], [7, 13], [7, 15], [8, 11], [7, 12], [6, 13], [7, 11], [6, 12], [7, 10], [6, 11], [7, 9], [6, 10], [5, 11], [6, 9], [5, 10], [4, 11], [6, 8], [5, 9], [4, 10], [4, 12], [3, 11], [6, 7], [5, 8], [4, 9], [3, 10], [3, 12], [2, 11], [2, 10], [2, 12], [1, 11], [2, 9], [1, 10], [1, 12], [0, 11], [2, 8], [1, 9], [0, 10], [2, 7], [1, 8], [0, 9], [2, 6], [0, 8], [2, 5], [3, 6], [2, 4], [3, 5], [2, 3], [3, 4], [1, 4]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase2 طی می­شود به­صورت زیر است:



* **خروجی TestCase3 به­صورت زیر است:**

****

مشاهده می­شود که رشته اعمال RRURRRDDRRRDRRDDDRDDLLDDDDDDRRDDLDDLLLLLLD راه حل است که هزینه­ی آن برابر 42 است.

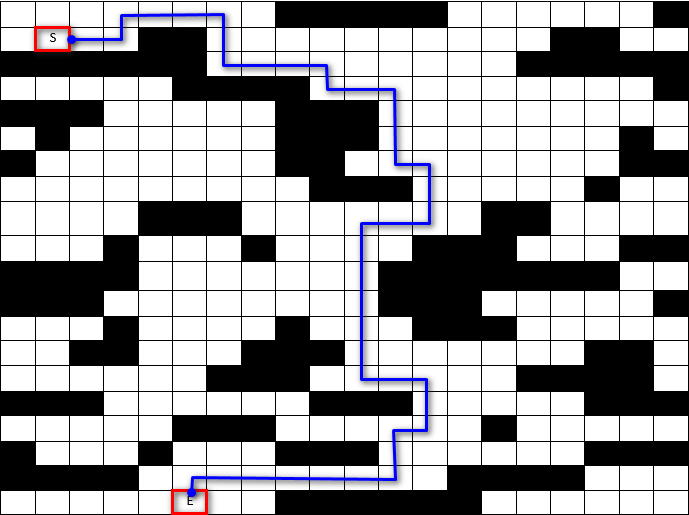
همچنین نودهای مسیر از ابتدا تا مقصد به­صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [18, 6], [17, 6], [17, 7], [17, 8], [17, 9], [16, 9], [16, 10], [16, 11], [15, 11], [14, 11], [13, 11], [13, 12], [12, 12], [11, 12], [11, 11], [11, 10], [10, 10], [9, 10], [8, 10], [7, 10], [6, 10], [5, 10], [5, 11], [5, 12], [4, 12], [3, 12], [3, 11], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [0, 5]

همچنین ترتیب نودهای بسط داده شده به­صورت زیر است که 162 نود را شامل می­شود.

[18, 3], [18, 1], [19, 2], [18, 2], [19, 3], [19, 0], [18, 0], [19, 1], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [17, 7], [18, 8], [17, 8], [18, 9], [17, 9], [18, 10], [17, 10], [16, 9], [16, 10], [18, 11], [17, 11], [16, 11], [18, 12], [17, 12], [16, 12], [15, 11], [15, 12], [14, 11], [14, 12], [13, 11], [13, 12], [13, 10], [18, 13], [17, 13], [16, 13], [15, 13], [14, 13], [13, 13], [12, 12], [12, 13], [11, 12], [11, 13], [11, 11], [11, 10], [10, 11], [11, 9], [10, 10], [11, 8], [10, 9], [9, 10], [11, 7], [12, 8], [10, 8], [9, 9], [8, 10], [12, 7], [9, 8], [8, 9], [7, 10], [9, 7], [8, 8], [7, 9], [7, 11], [6, 10], [9, 6], [8, 7], [6, 11], [5, 10], [9, 5], [10, 6], [8, 6], [7, 7], [5, 11], [5, 9], [9, 4], [10, 5], [8, 5], [7, 6], [8, 4], [7, 5], [6, 6], [7, 4], [6, 5], [6, 4], [5, 5], [5, 4], [4, 5], [4, 6], [4, 4], [18, 14], [19, 13], [17, 14], [16, 14], [15, 14], [14, 14], [13, 14], [12, 14], [12, 6], [13, 7], [6, 12], [5, 12], [10, 4], [8, 3], [5, 3], [4, 7], [4, 3], [3, 4], [12, 5], [13, 6], [3, 3], [12, 4], [13, 5], [18, 15], [19, 14], [16, 15], [15, 15], [14, 15], [13, 15], [12, 15], [14, 7], [6, 13], [5, 13], [4, 12], [5, 2], [4, 8], [14, 6], [3, 2], [2, 3], [12, 3], [13, 4], [14, 5], [4, 13], [3, 12], [2, 2], [3, 13], [3, 11], [2, 12], [3, 10], [2, 11], [2, 13], [1, 12], [3, 9], [1, 11], [3, 8], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [2, 7], [0, 7], [1, 5], [2, 6], [0, 6], [1, 4], [2, 5], [0, 5]

مسیری که توسط این الگوریتم در صفحه­ی بازی TestCase3 طی می­شود به­صورت زیر است:



# **مقایسه**

همانطور که گفتیم هر سه روش جستجو با رعایت کردن شرایطی، کامل و بهینه هستند و نیز گفتیم که پیچیدگی زمانی و فضایی هر دو روش جستجوی BFS و A\* نمایی و پیچیدگی زمانی IDFS نیز نمایی بود. بنابراین سوالی که پیش می آید این است که کدام یک بهینه تر و مناسب تر است؟

برای درک بهتر تفاوت های این سه روش جستجو، می توانیم نتایج حاصل شده برای هر سه Test Case را با هم مقایسه کنیم.

|  |  |  |
| --- | --- | --- |
| **A\*** | **IDFS** | **BFS** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cost** | **تعداد نود بسط داده شده** | **Cost** | **تعداد نود بسط داده شده** | **Cost** | **تعداد نود بسط داده شده** |  |
| 18 | 84 | 54 | 110 | 18 | 164 | **TestCase1** |
| 30 | 103 | 94 | 171 | 30 | 267 | **TestCase2** |
| 42 | 162 | 108 | 222 | 42 | 256 | **TestCase3** |

با توجه به داده هایی که در جدول می­بینیم و نیز با علم به این که جستجو A\* آگاهانه است، **بهترین و بهینه ترین روش از بین سه روش پیشنهاد شده، روش جستجوی A\* است.**

اگر بخواهیم با جزییات بیشتر به مقایسه بپردازیم، می­بینیم که از نظر هزینه صرف شده برای رسیدن به هدف، جستجوی A\* و BFS بهترین عملکرد را داشته و بهتر از IDFS عمل کرده­اند.

از نظر تعداد نود بسط داده شده، A\* بهترین عملکرد را داشته و پس از آن IDFS از BFS بهتر عمل کرده است.

از نظر پیچیدگی زمانی در حین Run کردن کد دیدیم روش جستجوی A\* بهترین عملکرد را داشت و روش جستجوی IDFS و BFS تقریبا مشابه هم و بیشتر از روش A\*  زمان برد تا به هدف برسند. البته این نتیجه با توجه به پیچیدگی زمانی کلی حساب شده برای دو روش ناآگاهانه که O(bd) و برای روش آگاهانه که O(b𝜖𝑑 ) بود نیز واضح است. زیرا 𝜖 خطای نسبی است و مقداری اعشاری و بسیار کوچک دارد و در نتیجه یک عدد نمایی کوچکتر به نسبت دو روش دیگر ایجاد می­کند.

از نظر پیچیدگی فضایی نیز با توجه به اینکه برای روش IDFS یک رابطه غیر نمایی به­صورت O(bd) داشتیم، بنابراین، این روش بهینه ترین است و بعد از آن A\* از BFS بهتر است زیرا برای روش A\* ، توان رابطه نمایی در یک اپسیلون اعشاری کوچک ضرب می شود و بنابراین پیچیدگی فضایی کمتری نسبت به BFS خواهد داشت.