«گزارش کار پروژه اول AI »

1-نحوه مدل کردن:

State:

برای این مسئله استیت وضعیت هر لحظه پازل و در واقع وضعیت نشان داده شده از لامپ های سالن است.

Initial state:

ماتریس ورودی n\*n که نشان دهنده وضعیت روشن یا خاموش بودن لامپ ها می باشد.در واقع یک پازل که به عنوان ورودی از کلاس تعریف شده به عنوان ارگومان به توابع میدهیم.

Goal state:

ماتریسی که همه لامپ ها خاموش باشند یعنی تمام مختصات ماتریس عدد 0 را نشان دهند.

Action:

فشردن هر کلید یا مختصات ماتریس به شکل (x,y)

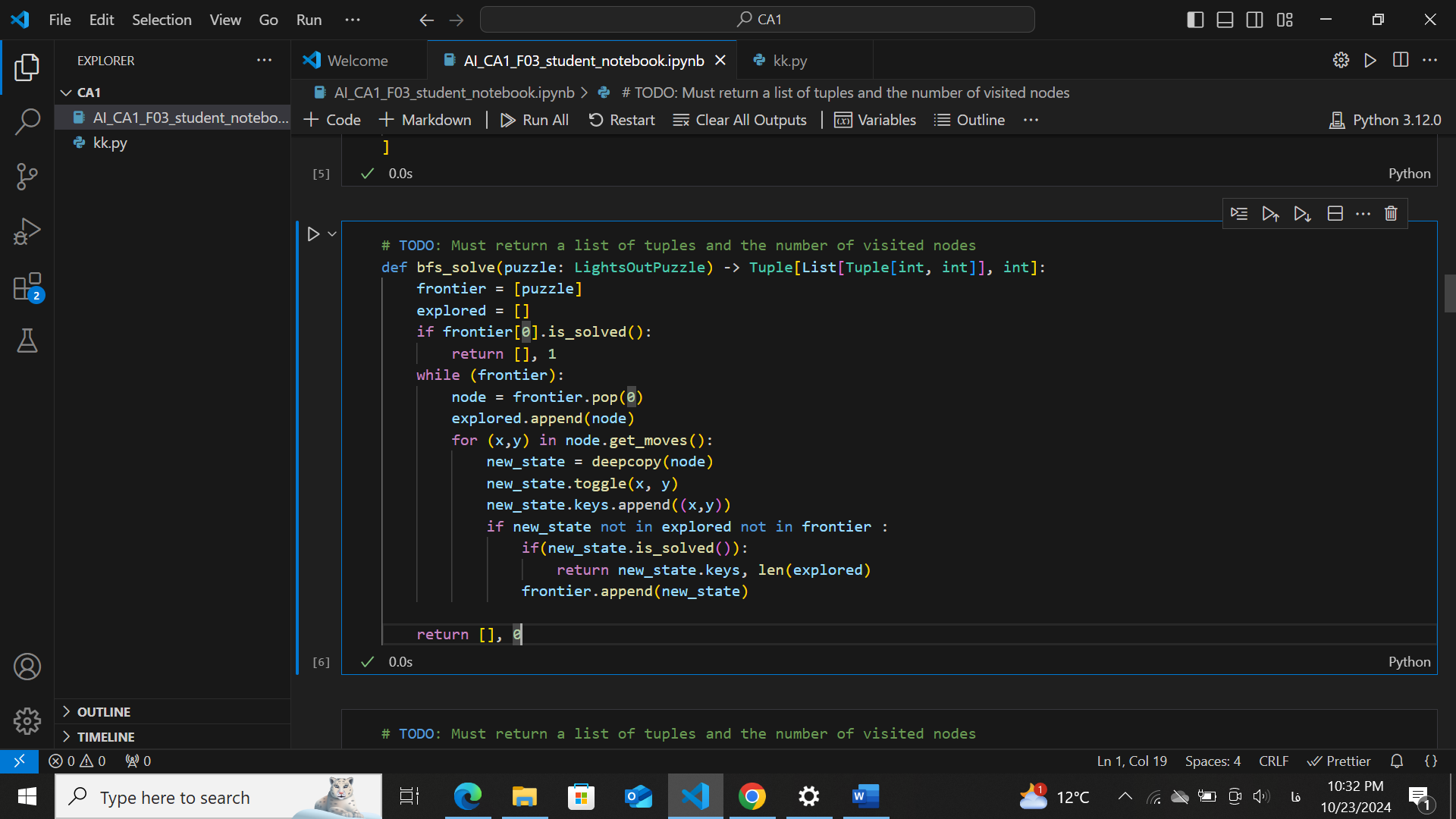
Transition model:

خاموش یا روشن شدن خود و خانه های مجاور آن(بالا،پایین،چپ،راست) که بعد از انجام یک اکشن اتفاق میوفتد و ماتریس را به استیت دیگر می برد.یعنی یک حالتی که چراغ ها تغییر وضعیت میدهند.

Path cost:

در این مسئله هزینه toggle کردن و از یک استیت به استیت بعدی رفتن را هزینه ای ثابت و برابر 1 در نظر میگیریم.

2-الگوریتم BFS:



این الگوریتم به این صورت عمل میکند که ابتدا تمام حالات ممکن در یک سطح را جستجو میکند و سپس به سطح بعدی میرود . به این صورت که هر بورد از حالات ماتریس یک نود در نظر گرفته میشود و هر بار با رفتن به یک نود و قرار دادن بچه های ان نود( که به تعداد اندازه طول لیست get.move() خواهد بود) زیرا این تابع به ما یک لیستی از مختصات ماتریس را میدهد و تعداد اکشن های ما به اندازه تعداد خانه های ماتریس می باشد (n\*n) که در واقع همان بچه های ماتریس اولیه میشود.

سپس الگوریتم BFS را روی لیست frontier خود میزنیم که در واقع همان بورد های تاگل شده پدر خود هستند.

و در واقع این یک پیمایش سطحی است که هر بار در یک عمق چک میکند که ایا به استیت هدف رسیدیم یا خیر . و اگر رسیده باشد return میکند و اگراز بچه های ان سطح نیز نباشد پازل پدر را در لیست explored ذخیره میکند و سپس به دنبال اولین ایندکس لیست میرود و این در واقع همان اجرای الگوریتم BFS می باشد.

به طور کلی ما دو لیست داریم :

Frontier:

این لیست در واقع یک لیست FIFO می باشد یعنی هر بار از ابتدای لیست انتخاب میشود برای بررسی فرزندانش که با toggle کردن نود پدر به ان میرسیم . و سپس نود پدر که همان ابتدای صف است را با عمل pop(0) از لیست حذف میکنیم.

Explored:

این لیست در واقع نود های ویزیت شده را میگیرد و در اخر نیز تعداد نودی که ویزیت شده را به عنوان خروجی میدهد.

همینطور یک متغییر در کلاس اولیه ایجاد کردیم : self.keys

تا بتوانیم تعداد کلید هایی که میزنیم تا پازل تعییر استیت بدهد را برگردانیم.

همچنین هنگامی که می خواهیم یک نود را وارد لیست frontier کنیم باید علاوه بر اینکه چک کنیم در لیست explored هست یا نه اینکه در لیست frontier هست یا نه را نیز چک کنیم و این به این دلیل است که اگر نودی قبلا در لیست frontier باشد قطعا قبلا و با هزینه کمتر ویزیت شده و دیگر نیازی به add کردن دوباره ان در لیست نیست.

بنابراین الگوریتم BFS یک الگوریتم بهینه است و همواره جواب optimal را بر میگرداند.

* الگوریتمIDS:

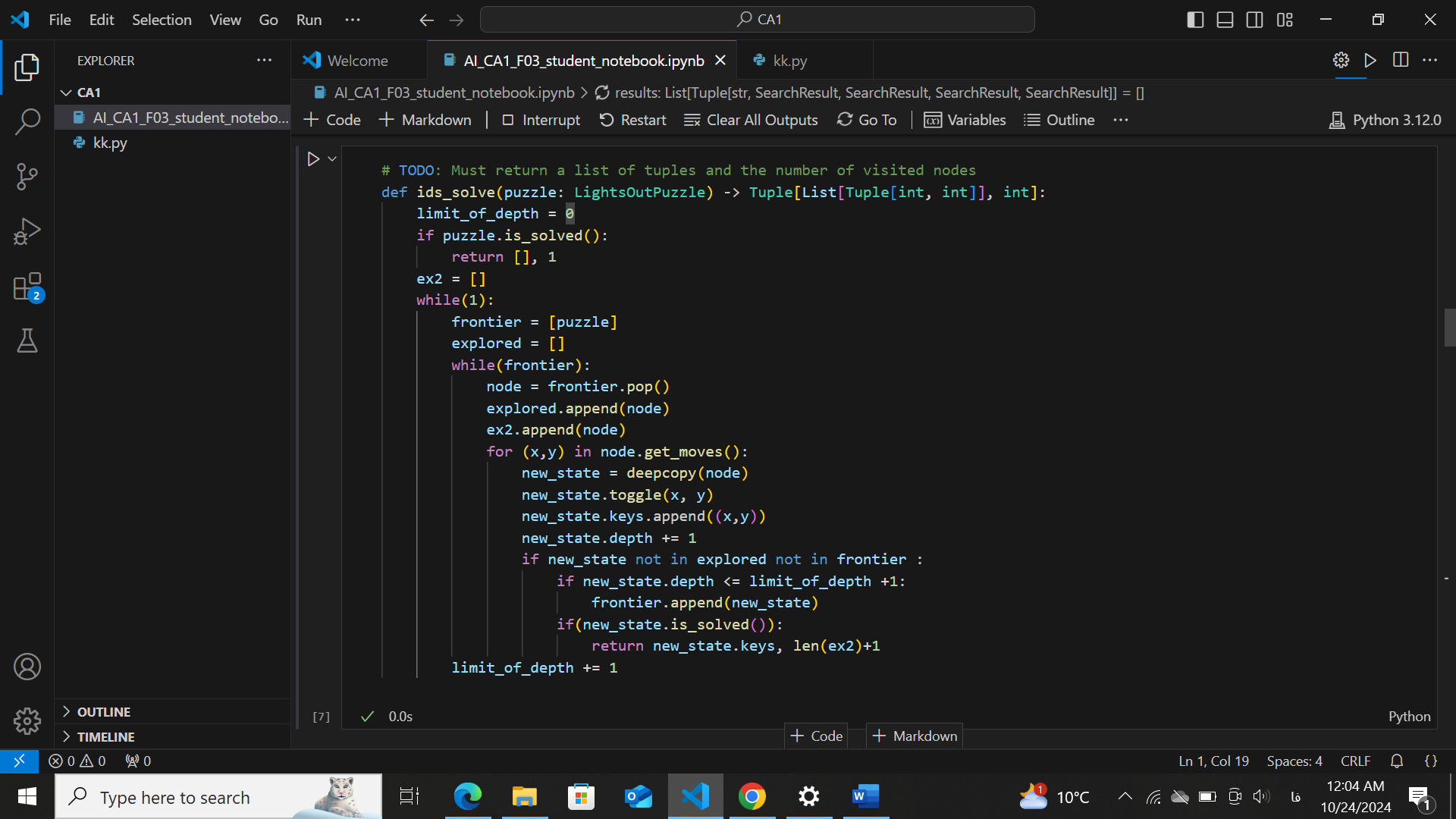
این الگوریتم به این صورت عمل میکند که هربار با الگوریتم DFSرا روی یک سطح یا عمق اجرا میکند .

به این منظور باید یک محدودیتی برای الگوریتم DFS قائل شویم که هربار و در هر مرحله بیشتر از یک سطح را چک نکند و اگر استیت مد نظر (سالن با چراغ خاموش یا ماتریس یا خانه های صفر)که دیگر به سطح بعدی نرود اما اگر در این سطح یافت نشود، حالا با اضافه کردن یک واحد به همان محدودیت ایجاد شده (limit\_of\_depth) ، دوباره الگوریتم DFS اما این بار در یک عمق بیشتر .

برای اجرای این الگوریتم هم نیاز به یک attribute جدید برای دادن عمق به پازل داریم که در کلاس تعریف میکنیم.

این الگوریتم در واقع الگوریتم BFS میباشد با این تفاوت که اینبار در واقع پازل های toggle شده را در یک استک یا یک صف LIFO نگه داری میشود و هر بار به جای بررسی اولین عضو لیست ،اخرین عضو لیست را بررسی و در explore ذخیره میکنیم در این الگوریتم هم از انجایی که بودن در لیست frontier باز هم بررسی میشود و اگر مسیری قبلا وجود داشته باشد یعنی قبلا دیده شده با هزینه کمتر .

و دیگر نیازی به ویزیت کردن ان نیست بنابراین این الگوریتم هم الگوریتمی optimal میباشد و جوابی بهینه میدهد.



* الگوریتمA\*

این الگوریم نیاز به تعریف یک تابع تخمین دارد تا با استفاده از ان بتوانیم مقصد بعدی برای ویزیت استیت را برگزینیم .

این الگوریتم در واقع مشابه الگوریتم BFS میباشد با این تفاوت که هربار برای ویزیت کردن یک نود و pop() کردن ان نودی انتخاب میشود که f(n) کوچک ترین داشته باشد میدانیم f(n) = h(n) +g(n)

که h(n) همان تابع تخمین میباشد که برای الگوریتم تعریف میکنیم.و g(n) برابر میشود با تعداد کلید های فشرده شده.

بنابراین برای اینکه بتوانیم در این عملیات نود با f(n) کمینه را ویزیت کنیم از minheap استفاده میکنیم و همچنین زمانی که f(n) یکسان داشته باشیم نودی انتخاب میشود که زود تر تولید شده باشد .

همچنین این الگوریتم زمانی بهینه میشود که تابع h(n) ما admissible باشد اگر این چنین باشد الگوریتم ما optimal خواهد بود.

* الگوریتم weighted\_A\*:

این الگوریتم در واقع همان الگوریتم A\* می باشد با این تفاوت که ما برای یک وزن برای ان تعریف میکنیم برای تسریع فرایند .

در واقع ما تابع h(n) را در یک ضریف ثابت ضرب میکینم که باعث میشود نود های کمتری ویزیت شود اما این باعث میشود optimal بودن الگوریتم از بین برود.

از طرفی تفاوت بین دو مقدار h(n) بیشتر و فرایند الگوریتم سریع تر میشود.

بنابراین الگوریتم مثل A\* است که f(n) جدید را در نظر میگیرد.

* مقایسه الگوریتم ها:

از نظر complete بودن همه الگوریتم ها complete هستن.

از نظر سرعت الگوریتم A\*weighted> A\*> IDS =BFS

از نظر فضای مورد نیاز A\*weighted =A\* >BFS < IDS

3-

H(n) اولی که تعریف کردیم برابر تعداد 1 های داخل پازل در هر نود است یعنی تعداد لامپ های روشن در سالن در هر استیت.

این تابع یک h(n) admissible نیست زیارا مثلا اگر ماتریس ما به شکل زیر باشد

[[0 1 0]

[1 1 1]

[0 1 0]]

H(n) = 5 میباشد

در صورتی که اگر لامپ (1و1) زده شود با هزینه 1 چراغ ها خاموش میشود.

و میبینیم که هزینه تخمینی ما از هزینه واقعی بیشتر است پس admissible نیست.

از طرفی consistent هم نیست زیاره برای مثال همین پازل بالا را اگر با فشردن کلید (1,1) به استیت بعدی برویم میبینیم که به Goalstate رسیدیم اما در این حالت h(n) برابر صفر است و هزینه رفتن به استیت بعدی نیز 1 است و رابطه

C(state1 to state2) + h(state2) >h(state 1) برقرار نیست پس

Consistent نیست.

4-

در واقع توابع h(n) که تعریف میکنیم مسیر های متفاوتی رو برای رسیدن به هدف طی میکند .

و h(n) که consistent باشد مسیر بهینه تری را برای رسیدن به هدف میرود اما اگر این ویژگی را نداشته باشد صرفا مارا به جواب میرساند و لزوما بهینه نیست.

تابع h(n) تعریف شده با اینکه consistent و admissible نیست اما تخمین خوبی می باشد و جواب بهینه تری به ما میدهد.

5- در تمام تست کیس ها در قسمت time زمان اجرای هر الگوریتم اضافه شده است:

Running test:

[[1 1 1]

[1 1 1]

[1 0 0]]

Solving with BFS:

Solution: [(0, 2), (1, 0)]

Nodes visited: 4

Time: 0.001

Solving with IDS:

Solution: [(1, 0), (0, 2)]

Nodes visited: 8

Time: 0.000

Solving with A\*(heuristic1):

Solution: [(1, 0), (0, 2)]

Nodes visited: 4

Time:0.000

Solving with A\*(heuristic2):

Solution: [(1, 0), (0, 2)]

Nodes visited: 8

Time:0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(1, 0), (0, 2)]

Nodes visited: 6

Time:0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(1, 0), (0, 2)]

Nodes visited: 6

Time: :0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(1, 0), (0, 2)]

Nodes visited: 4

Time: :0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(1, 0), (0, 2)]

Nodes visited: 4

Time: :0.000

-------------------------------

Running test:

[[1 1 0]

[1 0 0]

[1 1 1]]

Solving with BFS:

Solution: [(0, 0), (0, 1), (1, 0), (1, 1), (1, 2)]

Nodes visited: 137

Time: 0.002

Solving with IDS:

Solution: [(1, 1), (1, 0), (0, 1), (1, 2), (0, 0)]

Nodes visited: 567

Time: 0.358

Solving with A\*(heuristic1):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 155

Time:0.000

Solving with A\*(heuristic2):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 2867

Time: 14.487

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 72

Time: 0.027

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 77

Time: 0.031

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 504

Time: 0.595

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(1, 0), (1, 1), (1, 2), (0, 1), (0, 0)]

Nodes visited: 155

Time: 0.001

-------------------------------

Running test:

[[0 0 1]

[0 1 1]

[1 0 0]]

Solving with BFS:

Solution: [(0, 0), (0, 2), (1, 0)]

Nodes visited: 12

Time:0.000

Solving with IDS:

Solution: [(1, 0), (0, 2), (0, 0)]

Nodes visited: 49

Time: 0.000

Solving with A\*(heuristic1):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 13

Time: 0.000

Solving with A\*(heuristic2):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 65

Time: 0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 10

Time: 0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 10

Time: 0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 18

Time: 0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(0, 2), (1, 0), (0, 0)]

Nodes visited: 13

Time: 0.000

-------------------------------

Running test:

[[1 0 0 0]

[0 0 0 1]

[1 1 0 0]

[0 0 0 0]]

Solving with BFS:

Solution: [(0, 2), (0, 3), (1, 0), (1, 1)]

Nodes visited: 334

Time: 1.435

Solving with IDS:

Solution: [(1, 1), (1, 0), (0, 3), (0, 2)]

Nodes visited: 814

Time:0.000

Solving with A\*(heuristic1):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 14

Time: 0.001

Solving with A\*(heuristic2):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 706

Time: 0.819

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 7

Time:0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 5

Time:0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 57

Time: 0.033

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(1, 0), (1, 1), (0, 2), (0, 3)]

Nodes visited: 14

Time:0.000

-------------------------------

Running test:

[[0 0 0 1]

[0 1 1 0]

[0 1 1 0]

[1 1 1 0]]

Solving with BFS:

Solution: [(0, 3), (1, 2), (3, 1)]

Nodes visited: 62

Time: 0.116

Solving with IDS:

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 58

Time:0.000

Solving with A\*(heuristic1):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 4

Time:0.000

Solving with A\*(heuristic2):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 46

Time:0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 4

Time: 0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 4

Time: 0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 6

Time: 0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(3, 1), (1, 2), (0, 3)]

Nodes visited: 4

Time: 0.000

-------------------------------

Running test:

[[0 1 0 0]

[1 0 1 0]

[0 0 1 0]

[1 0 0 0]]

Solving with BFS:

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 86

Time: 0.000

Solving with IDS:

Solution: [(3, 0), (2, 1), (1, 1)]

Nodes visited: 68

Time: 0.000

Solving with A\*(heuristic1):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 5

Time: 0.000

Solving with A\*(heuristic2):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 60

Time: 0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 5

Time: 0.000

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 5

Time:0.001

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 7

Time: 0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(1, 1), (2, 1), (3, 0)]

Nodes visited: 5

Time: 0.000

-------------------------------

در این تست کیس به دلیل بزرگ بودن پازل time limit میخوریم

Running test:

[[0 0 0 0 1]

[1 0 0 1 0]

[0 0 0 1 1]

[0 1 0 1 1]

[1 1 0 1 0]]

Time limit of 180 seconds exceeded for BFS

Time limit of 180 seconds exceeded for IDS

Time limit of 180 seconds exceeded for A\*(heuristic1)

Time limit of 180 seconds exceeded for A\*(heuristic2)

Time limit of 180 seconds exceeded for weighted A\*(weighted\_heuristic1, alpha = 2)

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(2, 3), (4, 1), (4, 3), (3, 4), (1, 4), (1, 3), (2, 2), (3, 1), (4, 0), (3, 0), (2, 0), (0, 2), (1, 1), (0, 1), (0, 0)]

Nodes visited: 287

Time: 1.695

Time limit of 180 seconds exceeded for weighted A\*(weighted\_heuristic2, alpha = 2)

Time limit of 180 seconds exceeded for weighted A\*(weighted\_heuristic2, alpha = 5)

-------------------------------

Running test:

[[1 0 1 0 1]

[1 0 0 1 1]

[0 1 0 0 0]

[1 0 0 1 0]

[1 1 0 0 1]]

Time limit of 180 seconds exceeded for BFS

Time limit of 180 seconds exceeded for IDS

Solving with A\*(heuristic1):

Solution: [(4, 0), (0, 1), (1, 1), (3, 4), (2, 4), (1, 3), (0, 4)]

Nodes visited: 150

Time: 1.596

Time limit of 180 seconds exceeded for A\*(heuristic2)

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(4, 0), (1, 4), (3, 4), (0, 1), (1, 1), (2, 4), (1, 3), (1, 4), (0, 4)]

Nodes visited: 31

Time: 0.062

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(4, 0), (0, 3), (0, 0), (0, 4), (3, 4), (2, 4), (0, 4), (0, 2), (1, 1), (0, 1), (0, 0), (0, 2), (1, 3), (0, 3), (0, 4)]

Nodes visited: 56

Time: 0.904

Time limit of 180 seconds exceeded for weighted A\*(weighted\_heuristic2, alpha = 2)

Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(4, 0), (0, 1), (1, 1), (3, 4), (2, 4), (1, 3), (0, 4)]

Nodes visited: 150

Time: 0.578

-------------------------------

Running test:

[[1 0 0 0 0]

[0 0 1 0 0]

[1 0 1 0 0]

[0 1 0 0 0]

[0 0 0 0 0]]

Solving with BFS:

Solution: [(0, 0), (1, 1), (2, 1)]

Nodes visited: 32

Time: 5.625

Solving with IDS:

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 281

Time: 9.045

Solving with A\*(heuristic1):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 4

Time:0.000

Solving with A\*(heuristic2):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 53

Time: 0.016

Solving with weighted A\*(weighted\_heuristic1, alpha = 2):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 4

Time: 0.015

Solving with weighted A\*(weighted\_heuristic1, alpha = 5):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 4

Time:0.000

Solving with weighted A\*(weighted\_heuristic2, alpha = 2):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 6

Time: 0.108

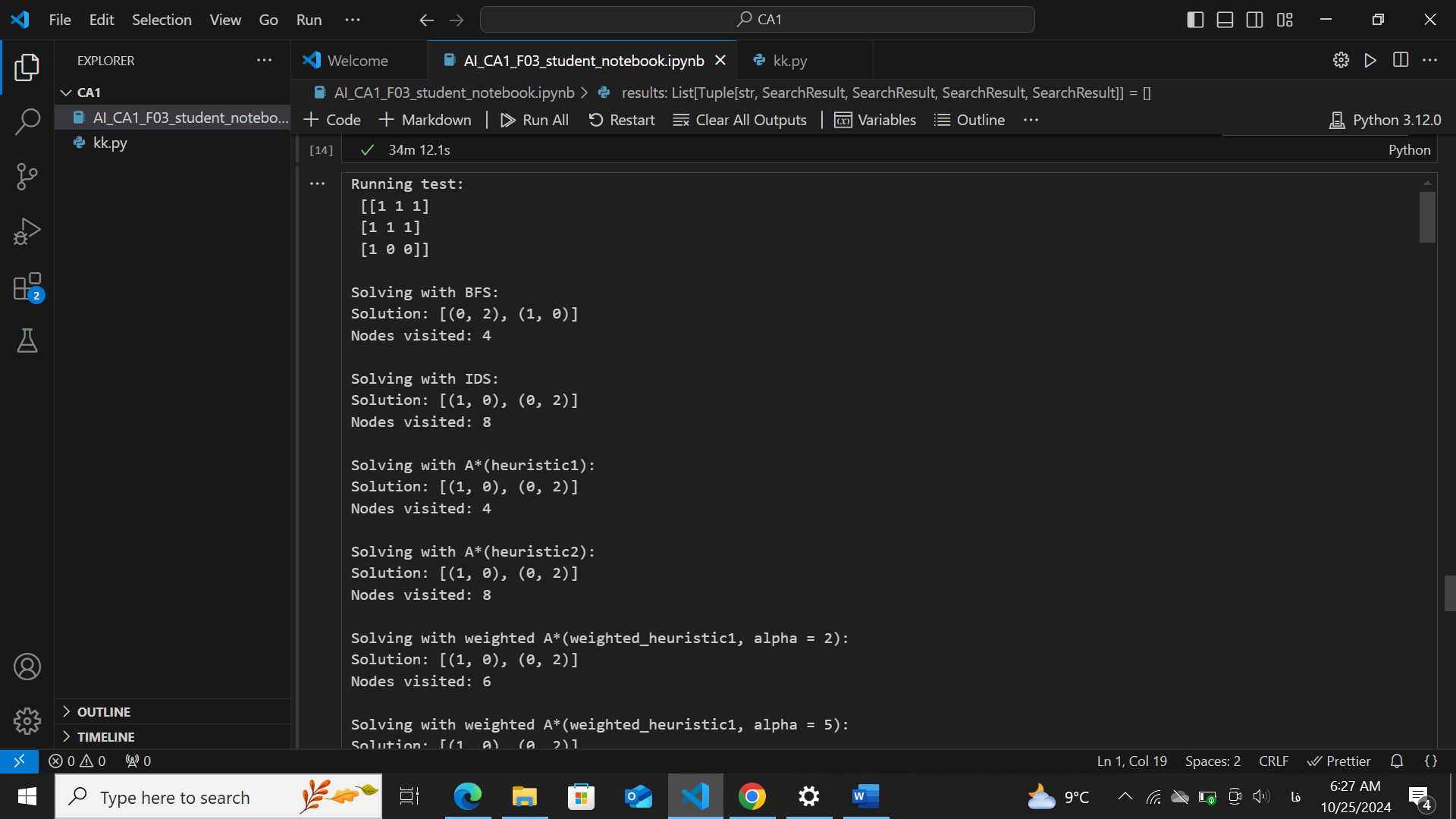
Solving with weighted A\*(weighted\_heuristic2, alpha = 5):

Solution: [(2, 1), (1, 1), (0, 0)]

Nodes visited: 4

Time:0.000

Run time:



در زمانی که time limit خوردیم مقدار اجرا را به اندازه ماکسیسمم زمان اجرا در صورت سوال قرار میدهیم:

میانگین اجرای الگوریتم BFS:

0.7976666667

میانگین اجرای الگوریتم IDS:

1.044777

میانگین اجرای الگوریتمA\*:

13.5107777

میانگین اجرای الگوریتم alpha=2 A\*weighed:

13.3448888

میانگین اجرای الگوریتم alpha=5 A\*weighed:

0.2945555