

به نام خدا



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



هوش مصنوعی پاییز ۹۸

پروژه یک

یکمن بدون روح

نام و نام خانوادگی

علیرضا زارع نژاد اشکذری

شماره دانشجویی

۸۱۰۱۹۶۴۷۴

شرح مختصر پروژه

هدف از این پروژه آشنایی با روش‌های جست‌وجو A^* , IDS, DFS, BFS می‌باشد. پنج فایل test.txt به ما داده شده است که هر کدام نقشه محیط جست‌وجو را نشان می‌دهد. در هر یک از شکل‌ها هر یک از $\frac{1}{2}$ نشانه‌ی دیوار و هر کدام از شماره‌های ۱ و ۲ و ۳ نشانه‌ی غذایی خاص است که عامل‌های هوشمند P, Q آن‌ها را می‌خورند. غذای ۱ برای p, و غذای ۲ برای q و ۳ مختص هر دو می‌باشد. هدف نهایی خوردن تمام غذاها می‌باشد. همچنین در هر حرکت فقط یکی از عامل می‌تواند حرکت کنند و نمی‌توانند از روی هم رد شوند.

نحوه‌ی مدل کردن به فضای جست و جو

ابتدا شروع به خواندن از روی فایل test می‌کنیم. در اینجا یک کلاس Node تعریف می‌کنیم که اطلاعات خاصی را در خود نگه می‌دارد. مثلاً تعداد غذاها، مختصات عامل‌های هوشمند و مختصات غذاها و هم‌چنین یک آرایه دو بعدی به نام map برای چاپ کردن استیت تعریف کردیم. واضح است که استیت اولیه همان ورودی اولیه است که شامل تعدادی غذا در مختصاتی خاص و همچنین دو عامل هوشمند است. در اینجا استیت نهایی را استیتی در نظر می‌گیریم که تعداد غذاهایش صفر باشد. Action ها را نیز برای رفتن از یک استیت به استیت دیگر به صورت زیر تعریف می‌کنیم که در هر حرکت عامل‌های هوشمند p, q به ترتیب می‌توانند به راست و بالا و چپ و پایین بروند. پس ماکسیمم branch factor برابر هشت می‌باشد. در اینجا با توجه به نوع حرکت انجام شده باید یک سری تغییرات اعمال شود که مثلاً آگه به خانه‌ای که می‌رویم غذا در آن باشد باید آن غذا خورده شود و مختصات آن remove و تعداد غذاها یکی کم شود یا آنکه مختصات عامل‌های هوشمند آپدیت شود. در اینجا ابتدا الگوریتم‌ها توضیح داده می‌شوند و خروجی آن‌ها مقایسه می‌شوند که عمق جواب و تعداد nodeهای ویزیت شده و زمان را بررسی می‌کنیم.

الگوریتم BFS

در این الگوریتم که به جست و جوی سطح معروف است ابتدا node شروع را بسط می‌دهیم و همه‌ی بچه‌های آن را در یک صف قرار می‌دهیم. هر بار یک node از صف خارج می‌کنیم و در صورتی که قبلاً ویزیت نشده بود آن را گسترش می‌دهیم و همین روال را تکرار می‌کنیم. دقت می‌کنیم که نودها سطح به سطح بررسی می‌شوند و در صورتی که یکی از نودها در هر سطح goal باشد یعنی تعداد غذاهایش صفر باشد

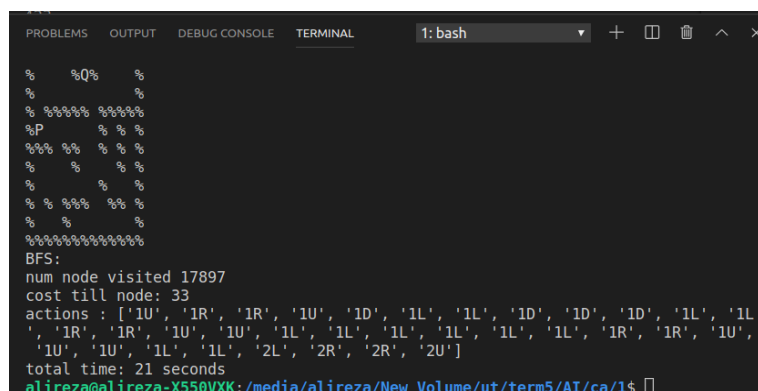
الگوریتم خاتمه می یابد. توجه می کنیم که چون branch factor در اینجا حداکثر ۸ است پس اگر عمق جواب n باشد حداکثر n^8 نود دیده خواهد شد. پیاده سازی این الگوریتم در زیر آورده شده است. این الگوریتم complete و optimal و هزینه ی زمانی آن $O(b^d)$ و همچنین حافظه ی آن $O(b^d)$ می باشد.

```
def BFS(self):
    start_time = datetime.datetime.now()
    fringe = Queue()
    visited = []
    node_start = problem.getStartState()
    if node_start.num_food == 0:
        print("already in goal!")
        node_start.print_map()
        return
    fringe.push(node_start)

    while not fringe.isEmpty():
        popped_element = fringe.pop()
        node = popped_element
        node_pos_set = node.make_set_of_pos()

        if node_pos_set not in visited:
            visited.append(node_pos_set)
            successors = problem.getSuccessors(node)
            for successor in successors:
                child_node = successor
                child_node_pos_set = child_node.make_set_of_pos()
                if self.isGoalState(child_node):
                    end_time = datetime.datetime.now()
                    moves = []
                    nodes_to_goal = []
                    nodes_to_goal.append(child_node)
                    while child_node.parent != None:
                        moves.append(child_node.action)
                        child_node = child_node.parent
                        nodes_to_goal.append(child_node)
                    nodes_to_goal.reverse()
                    moves.reverse()
                    for node in nodes_to_goal:
                        os.system('cls' if os.name == 'nt' else 'clear')
                        node.print_map()
                        time.sleep(.5)
                    print("BFS:")
                    print("num node visited", len(visited))
                    print("cost till node:", nodes_to_goal[-1].path_cost)
                    print("actions :", moves)
                    print("total time:", (end_time - start_time).seconds, "seconds")
                    return
            fringe.push(child_node)
```

حال خروجی هر کدام از تست ها را برای این الگوریتم بررسی می کنیم.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
% %Q% %
% %
% %%% %%%
%P % % %
%% % % %
% % %
% % %
% % % %
% %
% %%% %
% %
% %%% %
BFS:
num node visited 17897
cost till node: 33
actions : ['1U', '1R', '1R', '1U', '1D', '1L', '1L', '1D', '1D', '1D', '1L', '1L', '1R', '1R', '1U', '1U', '1U', '1L', '1L', '1L', '1L', '1L', '1L', '1R', '1R', '1U', '1U', '1U', '1L', '2L', '2R', '2R', '2U']
total time: 21 seconds
alireza@alireza-X550VXK:/media/alireza/New Volume/ut/term5/AI/ca/1$
```

تست اول:

تست دوم:

```
% % % % %
% %
% %
% %
% % % % %
% %
%P %
% % % %
%Q %
% % % % %
BFS:
num node visited 1153
cost till node: 17
actions : ['1U', '1U', '1L', '1L', '1D', '1D', '1D', '1D', '1R', '1R', '1L', '1L', '1L', '1D', '2L', '2L', '2L', '2L']
total time: 0 seconds
```

تست سوم :

```
%%%%%%%%%%
%          Q      %          %          %          %
% %%%%%%%%% %%%%%%%%% %          P          % %%%%%%%%%
%%%%%%%%%
BFS:
num node visited 1353
cost till node: 20
actions : ['1R', '1L', '1D', '1L', '1L', '1L', '1L', '1L', '1L', '1L', '1L', '2L', '2R',
', '2R', '2R', '2R', '2R', '2R', '2R', '2R', '2R']
total time: 0 seconds
alireza@alireza-X550VXK:/media/alireza/New Volume/ut/term5/AI/ca/1$
```

تست چهارم:

```

% % % % % % % % % % % % % % % %
%      %      %      %      %
%      % % % % % % % % % % %
%      %      %      %      %      %
%      %      %
% % % % % % % % % % %
%      %      %      % % % %
% % % P      Q % % % %
%      %      % % % % %
%      %      %
% % % %      % %      % % %
% % % % % % % % % % %
%      %
% % % % % % % % % % % % % % % %
BFS:
num node visited 9113
cost till node: 17
actions : ['1R', '1R', '1D', '1D', '1D', '2L', '2L', '2L', '2L', '2D', '2L', '2L',
           '2U', '2L', '2U', '2L', '2L']
total time: 13 seconds

```

تست پنجم:

```

#####
%      %
%      %
%      %
%      %
%  QP  %
%      %
%      %
#####
BFS:
num node visited 170
cost till node: 14
actions : ['1L', '1L', '1D', '1D', '1L', '1D', '1R', '1R', '1R', '1U', '2L', '2U',
           '2U', '2U']
total time: 0 seconds

```

الگوریتم IDS

مشابه الگوریتم DFS پیش می رویم یعنی جست جو در عمق بدین ترتیب که از stack به جای صف استفاده می کنیم که هر بار یکی pop می کنیم و در صورتی که قبلاً ویزیت نشده بود آن را گسترش می دهیم و به استک push می کنیم. همچنین بر خلاف bfs تشخیص goal بودن هنگام بسط دادن مشخص خواهد شد.

حال به ازای level های مختلف از یک تا جایی که هدف پیدا شود dfs می زنیم. کد الگوریتم در زیر آورده شده است.

که باید گفت الگوریتم complete و optimal و هزینه زمانی برابر $O(b^d)$ می باشد و حافظه $O(bd)$ می باشد. دقت می کنیم که IDS بر خلاف DFS باید جواب اپتیمال بدهد. در اینجا بررسی کردیم که اگر یک node قبلاً ویزیت شده باشد و حال اگر path cost آن بیشتر از path cost جاری باشد باید این نود را ویزیت کرده. توجه می کنیم زمان این الگوریتم نسبت به سایر الگوریتم ها خیلی بیشتر طول می کشد که در خروجی تست ها قابل درک می باشد.

```

def DFS(self, level, start_time):
    print(level)
    fringe = Stack()
    visited_set_pos = []
    visited_node = []
    node_start = problem.getStartState()
    fringe.push(node_start)

    while not fringe.isEmpty():
        node = fringe.pop()
        node_set_of_pos = node.make_set_of_pos()

        if self.isGoalState(node): ...
        else:
            if node.path_cost < level :
                if node_set_of_pos in visited_set_pos :
                    index = visited_set_pos.index(node_set_of_pos)
                    if visited_node[index].path_cost > node.path_cost :
                        del visited_node[index]
                        del visited_set_pos[index]
                        visited_set_pos.append(node_set_of_pos)
                        visited_node.append(node)
                        successors = problem.getSuccessors(node)
                        successors.reverse()
                        for child_node in successors:
                            fringe.push(child_node)
                else:
                    visited_set_pos.append(node_set_of_pos)
                    visited_node.append(node)
                    successors = problem.getSuccessors(node)
                    successors.reverse()
                    for child_node in successors:
                        fringe.push(child_node)

    return False

```

```

def IDS(self):
    start_time = datetime.datetime.now()
    level = 0
    while(True):
        if self.DFS(level, start_time):
            break
        level = level + 1
    return

```

خروجی هر کدام از تست ها به ترتیب آورده شده است.

تست اول:

```
%%%%%%%%%
%      %Q%      %
%      %
% %%% %%% %%%
%P      % % %
% % % % % % %
%      %      %
%      %      %
% % % % % %
%      %
%%%%%%%%%
IDS
num node visited 16046
cost till node: 33
actions : ['1U', '1R', '1R', '1U', '1D', '1L', '1L', '1D', '1D', '1D', '1L', '1L', '1R',
'1R', '1U', '1U', '1L', '1L', '1L', '1L', '1L', '1L', '1R', '1R', '1U', '1U', '1U', '1L',
'1L', '2L', '2R', '2R', '2U']
total time: 3205 seconds
```

تست دوم:

```
%%%%%%%%
%      %
%      %
%      %
% % % %
%      %
%P      %
% % % %
%Q      %
%%%%%%%%
IDS
num node visited 479
cost till node: 17
actions : ['1U', '1U', '1L', '1L', '1D', '1D', '1D', '1D', '1R', '1R', '1L', '1L', '1L',
'1D', '2L', '2L', '2L']
total time: 5 seconds
```

تست سوم:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Q      %      % % % % % % %
% % % % % % %      %      P      % % % % %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
IDS
num node visited 853
cost till node: 20
actions : ['1R', '1L', '1D', '1L', '1L', '1L', '1L', '1L', '1L', '1L', '2L', '2R', '2R',
'2R', '2R', '2R', '2R', '2R', '2R']
total time: 14 seconds
```

تست چهارم:

```
% % % % % % % % % % % % % % % % % % % %  
%      %              %                  %  
%    % % % % % % %   %     %           %       %  
%          %            %         %        % %  
%                %                    % % % % %  
% % % % % % % % % % % % % %             %  
%               %                 % % % % %  
% % % P                   Q % % % % %  
%           %          % % % % % % % % % %  
%           %         %  
% % %           % %                       % %  
% % % % % % % % % % % % % % % % % %  
% %
```

ID5
num node visited 7729
cost till node: 17
actions : ['1R', '1R', '1D',
'2L', '2U', '2L', '2L']
total time: 109 seconds

تست پنجم:

```

%%%%%%%%
%      %
%      %
%      %
%%    %%
%   QP%
%      %
%%    %%
%      %
%%%%%%%%
IDS
num node visited 145
cost till node: 14
actions : ['1L', '1L', '1D', '1D', '1L', '1D', '1R', '1R', '1R', '1U', '2L', '2U', '2U',
'2U']
total time: 0 seconds

```

الگوریتم A*

در اینجا دو عامل در انتخاب استیت انتخابی بعدی مهم است که خود را در هزینه تا نود جاری به اضافه ی هزینه از نود بچه تا goal را شامل می شود. هر کدام از نود های بچه که مقدار کمتری داشته باشند انتخاب می شوند. در حقیقت برای پیاده سازی از priority queue استفاده کردیم.

در اینجا دو روش برای heuristic می‌توانیم انتخاب کنیم. یکی تعداد غذاهای باقی مانده در هر استیت که مشخصاً از هزینه واقعی که حداقل در هر حرکت باید یکی جا به جا شویم تا غذا خورده شود کمتر است و این یعنی admissible است.

در روش دوم می توانیم فاصله p را تا نزدیک ترین غذا و فاصله q را تا نزدیک ترین غذای خود یعنی ۲ را بدست بیاوریم و مجموع را به عنوان خروجی بدهیم. که باز هم چون باید همه غذا ها خورده شود این مجموع از هزینه واقعی کمتر است. دقت می کنیم که p نمی تواند غذای ۲ ، و q نمی تواند غذای ۱ را بخورد.

این الگوریتم optimal و complete است و حافظه زمانی و مکانی آن نمایی است.

در زیر نحوه پیاده سازی الگوریتم آورده شده است. همچنین به وضوح consistency در حالت نخست برقرار است چرا که تعداد غذاها یا ثابت است و یا یکی کم می شود پس هیروستیک در هر نود از هیروستیک بعدی به علاوه یک کمتر است.

```
def heuristic(self,node):
    return node.num_food

    # distance_p = []
    # distance_q = []

    # for pos in node.foods['1']:
    #     distance_p.append(manhattanDistance(pos,node.agent1))
    # for pos in node.foods['2']:
    #     distance_q.append(manhattanDistance(pos,node.agent2))

    # x,y = 0 , 0
    # if len(distance_p):
    #     x = min(distance_p)
    # if len(distance_q):
    #     y = min(distance_q)
    #     y = min(distance_q)
    # return x + y

def aStarSearch(self):
    start_time = datetime.datetime.now()
    fringe = PriorityQueue()
    visited= []

    node_start = problem.getStartState()
    fringe.push(node_start, 0)
    while not fringe.isEmpty():
        node = fringe.pop()
        node_pos_set = node.make_set_of_pos()

        if problem.isGoalState(node): ...
        else:
            if node_pos_set not in visited:
                visited.append(node_pos_set)
                successors = problem.getSuccessors(node)
                for child_node in successors:
                    child_cost = child_node.path_cost
                    fringe.push(child_node, child_cost + self.heuristic(child_node))
    return
```

حال خروجی هر کدام از تست ها آورده شده است.

تست اول:

```

%%%%%%%%%%%%%%%%%%%%%%%%
%           %Q%           %
%           %           %
% %%%%%%%%% %%%%%%%%%
%P           % % %
%%%%%%%% % % % %
%           % % %
%           % %
% % % % % %
%           %
%%%%%%%%
A* search:
num node visited 17166
cost till node: 33
actions : ['2L', '2R', '2R', '2U', '1U', '1R', '1R', '1U', '1D', '1L', '1L', '1D', '1D', '1D', '1L', '1L', '1R', '1R', '1U', '1U', '1L', '1L', '1L', '1L', '1L', '1L', '1R', '1R', '1U', '1U', '1U', '1L', '1L']
total time: 18 seconds

```

تست دوم:

```

%%%%%%%%
%           %
%           %
%           %
%% %%%
%           %
%P           %
%% % %
%Q           %
%%%%%%%%
A* search:
num node visited 1160
cost till node: 17
actions : ['1U', '1U', '2L', '2L', '2L', '1L', '1L', '1D', '1D', '1D', '1D', '1R', '1R', '1L', '1L', '1L', '1D']
total time: 0 seconds

```

تست سوم:

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Q           %           % % % % % %
% %%%%%%%%% % % % % % % P           % %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A* search:
num node visited 1354
cost till node: 20
actions : ['1R', '2L', '1L', '1D', '1L', '1L', '1L', '1L', '1L', '1L', '1L', '2R', '2R', '2R', '2R', '2R', '2R', '2R', '2R']
total time: 0 seconds

```

تست چهارم :

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      %      %      %
%      %%%%%%%%% %      % % % %
%      %      %      %      %      %
%      %      %      %%%%%%%%%
% %%%%%%%%% %%%%%%%%% %%%%%%%%% %
%      %      %      % %%%%%%%%%
% % % P      Q % % % %%%%%%%%%
%      %      %%%%%%%%% %      %
%      %      %      %      %
% % % %      % %      % % %
% % % % % %%%%%%%%% %%%%%%%%% %
%      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A* search:
num node visited 6981
cost till node: 17
actions : ['1R', '1R', '1D', '1D', '1D', '2L', '2L', '2L', '2L', '2D', '2L', '2L', '2U', '2L', '2U', '2L', '2L']
total time: 9 seconds
(base) c:\Users\Ghazal\OneDrive\Documents\New Volume\ut\term5\AI\cs\14

```

تست پنجم:

```

%%%%%%%%
%      %
%      %
%      %
% % %
% QP%
%      %
% % %
%      %
% % %
%%%%%%%%
A* search:
num node visited 163
cost till node: 14
actions : ['1L', '1L', '1D', '1D', '1L', '1D', '1R', '1R', '1R', '1U', '2L', '2U', '2U', '2U']
total time: 0 seconds
(base) c:\Users\Ghazal\OneDrive\Documents\New Volume\ut\term5\AI\cs\14

```

مقایسه الگوریتم ها با توجه به جدول خواسته شده

تست اول:

الگوریتم	فاصله از جواب	تعداد استیت های مجزا دیده شده	تعداد استیت های دیده شده	زمان اجرا
BFS	۳۳	۱۷۸۹۷	۱۰۱۷۷۳	۲۲
IDS	۳۳	۱۶۰۴۶	۵۸۰۳۴۴۸	۳۲۰۵
A* search	۳۳	۱۷۱۶۶	۹۷۱۹۶	۱۷

تست دوم:

الگوریتم	فاصله از جواب	تعداد استیت های مجزا دیده شده	تعداد استیت های دیده شده	زمان اجرا
BFS	۱۷	۱۱۵۳	۶۳۴۳	۰
IDS	۱۷	۴۷۹	۵۵۸۸۳۶	۵
A* search	۱۷	۱۱۶۰	۶۳۰۶	۰

تست سوم:

الگوریتم	فاصله از جواب	تعداد استیت های مجزا دیده شده	تعداد استیت های دیده شده	زمان اجرا
BFS	۲۰	۱۳۵۳	۶۶۵۷	۰
IDS	۲۰	۸۵۳	۱۰۰۶۷۹	۱۳
A* search	۲۰	۱۳۵۴	۶۶۸۰	۰

تست چهارم

الگوریتم	فاصله از جواب	تعداد استیت های مجزا دیده شده	تعداد استیت های دیده شده	زمان اجرا
BFS	۱۷	۹۱۱۳	۴۹۵۴۳	۱۲
IDS	۱۷	۷۷۲۹	۲۸۶۳۴۵	۱۰۷
A* search	۱۷	۶۹۸۱	۳۷۲۶۸	۹

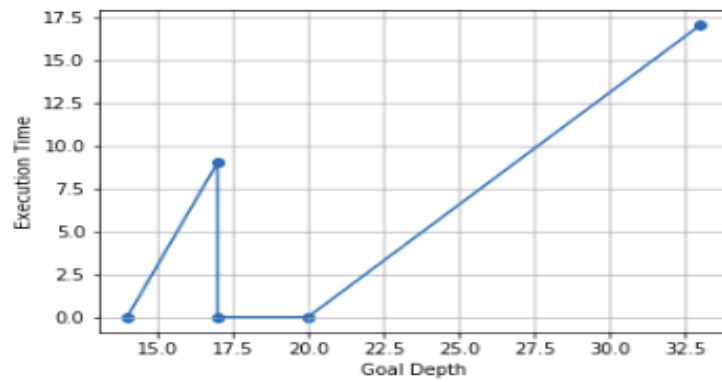
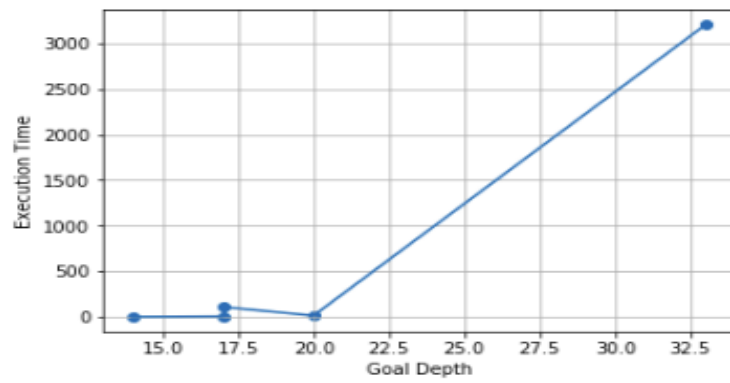
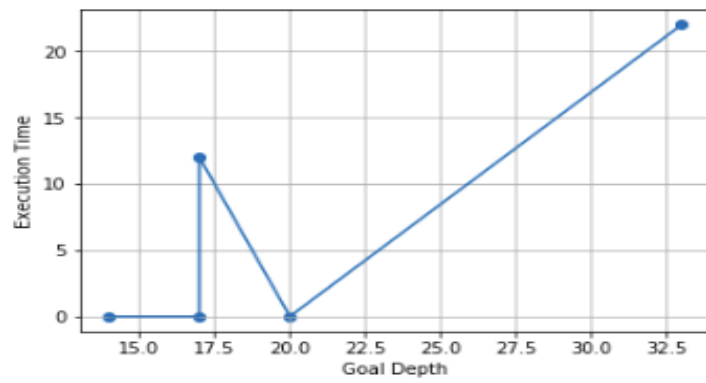
تست پنجم:

الگوریتم	فاصله از جواب	تعداد استیت های مجزا دیده شده	تعداد استیت های دیده شده	زمان اجرا
BFS	۱۴	۱۷۰	۷۶۱	۰
IDS	۱۴	۱۴۵	۸۴۱۷	۰
A* search	۱۴	۱۶۳	۷۳۳	۰

با توجه به اینکه هر کدام از الگوریتم ها مسئله را اپتیمال حل می کنند ولی مدت زمان الگوریتم IDS زیاد است و خب در چنین شرایطی اگر branch factor کم باشد bfs می تواند بهتر عمل کند. در مقابل استفاده از سرچ با آگاهی از رفتن به بعضی استیت ها خود داری می کند و در حقیقت تعداد نود های ویزیت شده و زمان کمتری خواهد داشت.

نمودار

در انتها نمودار سه الگوریتم به ترتیب آمده است که محور x آن فاصله از جواب و y نشانه ی هزینه زمانی است.



پایان

پایان