پاسخ سوالات پروژه ۱

۱: کاربرد کلاس SearchProblems در فایل py.search را به همراه متود های آن توضیح دهید.

پاسخ: این کلاس، یک کلاس abstract است که ساختار یک مساله جست و جو را بیان می کند.

متدهای getStartState و isGoalState و getSuccessors و getCostOfActions به ترتیب وضعیت شروع، هدف بودن یا نبودن، فرزندان یک وضعیت که شامل آن فرزند، مسیررسیدن به آن از والد و هزینه این انتقال می باشد وهزینه رفتن از یک گره به گره دیگربا استفاده از حرکات مجاز را برمی گردانند.

۲: همچنین به اختصار کاربرد هر یک از کلاس های ,Configuration, Directions, Agent Grid, AgentStateرا که در فایل py.game قرار دارند، بیان کنید.

پاسخ:

Configuration : این کلاس مختصات یک موقعیت به همراه جهت آن را درخود نگه می دارد و شامل متدهای getter و... برای دریافت اطلاعات مربوط به یک وضعیت است .

Directions : این کلاس درواقع درخود کتابخانه هایی را برای تعیین راست ، چپ و معکوس جهات مختلف مختصاتی تعریف کرده است .

Agent : این کلاس دارای متد getAction است که وضعیت بازی را ازکلاس هایی مانند pacman.py و ... دریافت می کند و مطابق آن به یکی از ۴ جهت (شمال، جنوب،شرق وغرب) می رود.

Grid : این کلاس وضعیت نقاط مختلف زمین بازی را به صورت یک آرایه دو بعدی Boolean نگه می دارد که درآن دیوارها مقدارF دارند و نقاط دیگرT .

AgentState : این کلاس وضعیت عامل مانند پیکربندی ، سرعت و ترسیده یا نترسیده بودن را نگه می دارد .

۳ : در غالب یک شبه کد مختصر الگوریتم IDS را توضیح دهید و تغییرات الزم برای تبدیل الگوریتم DFSبه IDS را نام ببرید.

پاسخ:

شبه کد:

```
def IDS (problem):
    for i in range(deepest_level):
        run DFS until depth <= i
```

برای تبدیل DFS به IDS کافی است یه حلقه به متد DFS اضافه کنیم که هربار تاعمق i ام می رود(درحقیقت به فرزندانی که درعمق i ام هستند اجازه نمی دهیم فرزند تولید کنند و داخل fringe بریزند.

۴: الگوریتم BBFS را به صورت مختصر با نوشتن یک شبه کد ساده توضیح دهید و آن را با الگوریتم BFSمقایسه کنید. همچنین ایدهای بدهید که در یک مسئله جستجو که به دنبال بیش از یک هدف هستیم چگونه میتوانیم از BBFS استفاده کنیم.

پاسخ:

الف) شبه کد:

```
Def BBFS(problem):
  While (!s_queue.isEmpty() and !t_queue.isEmpty()):
      bfs(s_queue, s_visited, s_parent)
      bfs(t_queue ,t_visited, t, t_ parent)
      if (exists an i which t_visited[i] == True and s_visited[i]==True):
          return "There is a path between source and targer"
  return "There is not a path between source and targer"
```

ب)درمساله جستجویی که دنبال بیش از یک هدف هستیم می توانیم از BBFS چندین بار استفاده کرده و target ها را درآن ها متفاوت بدهیم . درواقع با توجه به این که BBFS از نظر زمانی بهینه تر از BFS است (از اردر زمانی (b^(m/2) می توانیم با استفاده از این موضوع در مدت زمان معین به جای استفاده از BFSبرای پیداکردن یک target، از BBFS برای پیدا کردن چندین target استفاده کنیم.

۵: آیا ممکن است که با مشخص کردن یک تابع هزینه مشخص برای الگور یتم UCS ،به الگور یتم BFS و یا DFS برسیم؟ در صورت امکان برای هرکدام از الگوریتمهای BFS و یا DFS ،تابع هزینه مشخص شده را با تغییر کد خود توضیح دهید)نیاز به پیاده ساز ی کد جدیدی نیست؛ صرفا تغییراتی را که باید به کد خود اعمال کنید را ذکر نمایید.)

پاسخ:

بله. درصورتی که در الگوریتم UCS هزینه انتقال از یک گره به گره دیگر را مقدار ثابت (مثلا ۱) درنظربگیریم، به الگوریتم BFS می رسیم .

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    fringe = util.PriorityQueue()
    visited = []
    fringe.push((problem.getStartState(), [], 0), 0)
    while not fringe.isEmpty():
        state, path, cost = fringe.pop()
        if state in visited:
            continue
        visited.append(state)
        if problem.isGoalState(state):
            return path
        successors = problem.getSuccessors(state)
        for successor in successors:
            if successor[0] not in visited:
                fringe.push((successor[0], path + [successor[1]], cost + successor[2]), cost + successor[2])

    return []
    # util.raiseNotDefined()
```

برای این کارکافی است در خط۱۸۱ به جای successor[2]، عدد ۱ را قراردهیم.

۶: الگوریتمهای جستجویی که تا به این مرحله پیاده سازی کرده اید را روی openMazeاجرا کنید و توضیح دهید چه اتفاقی می افتد (تفاوت ها را شرح دهید).

پاسخ:

DFS : تمام مسیر به دلیل جستوجو عمقی به صورت طبقه ای کاوش می شود که بهینه نیست و در یافتن هدف تاثیری ندارد.

BFS : :به دلیل این که سطحی جست وجو می کند و هدف در عمیق ترین لایه وجود دارد، کل مسیر پیمایش می شود و هزینه بیشتری دارد.

UCS : مشابه BFS است چون هزینه همه خانه ها یک است.

A* : مشابه ۲ مورد قبلی با این تفاوت که خانه های کمتری برای رسیدن به هدف کاوش می شوند. (به دلیل استفاده از هیوریستیک) در نتیجه بهترین عملکرد را دارد.

۷: هیوریستیک خود (CornersProblem) را توضیح دهید و سازگاری آن را استدلال کنید.

از بین فاصله (mazeDistance)عامل تا گوشه هایی که تا به حال ملاقات نکردیم ،ماکسیمم مقدار را به عنوان هیوریستیک درنظرمی گیریم.در mazeDistance از

BFSاستفاده می شود و اینگونه وجود دیوارها نادیده گرفته نمی شود .این هیوریستیک به دلیل این که همیشه فاصله تا دورترین گوشه ملاقات نشده را برمی گرداند و هزینه واقعی ما هزینه ملاقات همه گوشه هاست پس همیشه کمتر یا مساوی هزینه واقعی است.

۸: هیوریستیک خود (foodHeuristic) را توضیح دهید و سازگاری آن را استدلال کنید وپیاده سازی هیوریستیک خودتان در این بخش و در بخش قبلی را با یکدیگر مقایسه و تفاوت ها را بیان کنید.

در این بخش نیز مانند بخش قبل ماکسیم mazeDistance را به عنوان هیوریستیک درنظرمی گیریم با این تفاوت که دراین بخش فاصله عامل تا غذاها بررسی می شود.

۹: ClosestDotSearcheAgntشما، همیشه کوتاهترین مسیر ممکن در مارپیچ را پیدا نخواهد کرد. مطمئن شوید که دلیل آن را درک کرده اید و سعی کنید یک مثال کوچک بیاورید که در آن رفتن مکرر به نزدیکترین نقطه منجر به یافتن کوتاهترین مسیر برای خوردن تمام نقاط نمیشود.

با توجه به الگوریتم، agent ما همیشه به نزدیکترین نقطه می رود و در نتیجه کوتاه ترین مسیر ممکن است به دست نیاید و هزینه افزایش یابد .مثال، در این جا به جای اینکه ابتدا نقطه بالا سمت راست را بخورد سمت نقاط نزدیک تر سمت چپ می رود و در نهایت دوباره مجبوریم به آن نقطه برگردیم که باعث می شود مسیر اضافه ای طی شود .