



نیم‌سال دوم سال ۹۷-۹۸

تمرین سری اول: جست‌وجو در فضای حالات

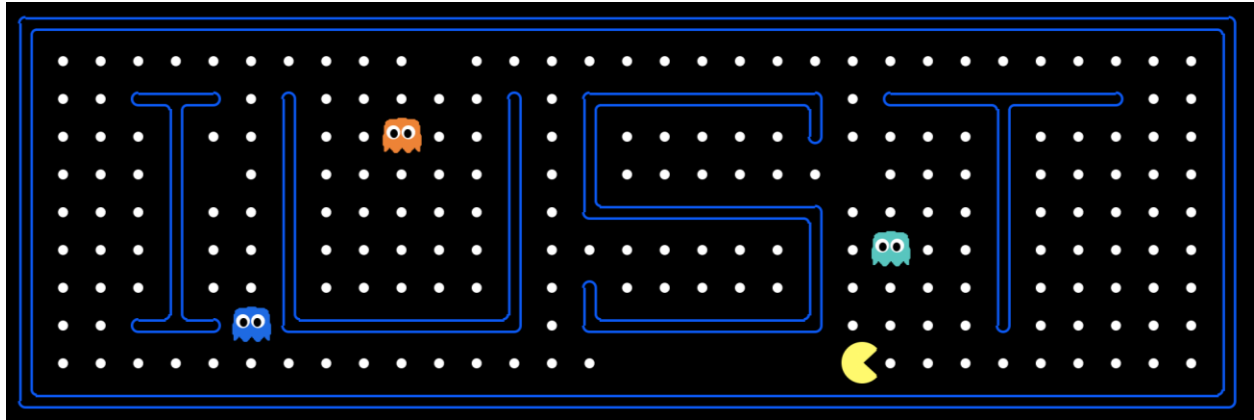
لطفاً به نکات زیر توجه کنید:

- مهلت ارسال این تمرین تا ۱۵ اسفند است.
- در صورتی که به اطلاعات بیشتری نیاز دارید می‌توانید به صفحه‌ی تمرین در وب‌سایت درس مراجعه کنید.
- این تمرین شامل سوال‌های برنامه‌نویسی می‌باشد، بنابراین توجه کنید که حتماً موارد خواسته‌شده در سوال را رعایت کنید. در صورتی که به هر دلیلی سامانه‌ی داوری نتواند آن را اجرا کند مسئولیت آن تنها به عهده‌ی شماست.
- ما همواره هم‌فکری و هم‌کاری را برای حل تمرین‌ها به دانشجویان توصیه می‌کنیم. اما هر فرد باید تمامی سوالات را به تنهایی تمام کند و پاسخ‌ارسانی حتماً باید توسط خود دانش‌جو نوشته‌شده باشد. لطفاً اگر با کسی هم‌فکری کردید نام او را ذکر کنید. در صورتی که سامانه‌ی تطبیق، تقلبی را تشخیص دهد متأسفانه هیچ مسئولیتی بر عهده‌ی گروه تمرین نخواهد بود.
- لطفاً برای ارسال پاسخ‌های خود از راهنمای موجود در صفحه‌ی تمرین استفاده کنید.
- هر سوالی درباره‌ی این تمرین را می‌توانید از دستیاران حل تمرین بپرسید.

- آدرس گروه درس: <https://groups.google.com/forum/#!forum/ai972>

- صفحه تمرین: <https://quera.ir/course/assignments/8175/problems>

موفق باشید



فریم‌ورکی^۱ که در این سری از تمرین‌ها با آن کار می‌کنید یک نسخه‌ی ساده و البته کامل از بازی معروف پکمن است. هدف از ایجاد این چهارچوب، پیاده‌سازی و یادگیری مفاهیم و تکنیک‌های پایه در هوش مصنوعی مانند جست‌و‌جو در فضای حالات، یادگیری تقویتی و استنتاج احتمالی است.

قبل از اینکه به اولین سوال پردازیم، ابتدا باید کمی با نحوه‌ی کارکرد این فریم‌ورک آشنا شویم.

۱. نحوه‌ی اجرا:

فایل زیپ را از صفحه‌ی تمرین دانلود کنید و آن را از حالت فشرده خارج کرده، سپس دستورات زیر را اجرا کنید:

```
$ cd assignment01  
$ python pacman.py
```

می‌توانید زمین بازی را به نقشه‌ی دلخواهتان تغییر دهید (سایر نقشه‌ها را در پوشه‌ی layouts می‌توانید پیدا کنید):

```
$ python pacman.py --layout powerClassic
```

می‌توانید عامل^۲ کنترل‌کننده‌ی پکمن و حتی روح‌ها را هم عوض کنید:

```
$ python pacman.py --pacman GreedyAgent --ghost DirectionalGhost
```

برای مشاهده‌ی تمام قابلیت‌های بازی می‌توانید از دستور زیر استفاده کنید:

```
$ python pacman.py -h
```

^۱ این فریم‌ورک ابتدا در دانشگاه برکلی توسعه یافته و سپس برای این درس شخصی‌سازی شده است.

^۲ Agent

۲. ساختار فایل‌ها:

نکته: این فرمورک با زبان پایتون نوشته شده است. بنابراین برای انجام تمرین‌ها نیاز به کمی آشنایی با زبان پایتون دارید. در صورت نیاز می‌توانید از این جا استفاده کنید.

اطلاعاتی که برای انجام این تمرین نیاز دارید کاملاً در قسمت بعد آمده است بنابراین این قسمت مستقیماً مورد سوال نیست اما مطالعه‌ی آن دید بهتری از ساختار فریمورک به شما می‌دهد.

ماژول‌های اصلی، بهتر است نگاهی به آن‌ها بیندازید.	
paceman.py	این فایل، نقطه‌ی شروع برنامه است و جزئیات مخصوص به بازی پک‌من مانند سیاست‌های برد و باخت، نحوه‌ی حرکت شخصیت‌های بازی و تعاملات آن‌ها با یک‌دیگر را مدل می‌کند.
game.py	موتور اصلی بازی و نحوه‌ی کنترل آن در این فایل قرار دارد. داده‌ساختارهای AgentState (وضعیت شخصیت)، Agent (شخصیت‌ها) و Grid (نقشه‌ی بازی) در آن پیاده‌سازی شده‌اند.
pacmanAgents.py	چند مورد از عامل‌های کنترل‌کننده‌ی پک‌من در این ماژول پیاده‌سازی شده‌اند.
ghostAgents.py	چند مورد از عامل‌های کنترل‌کننده‌ی روح‌ها در این ماژول پیاده‌سازی شده‌اند.
keyboardAgents.py	عامل کنترل‌کننده که دستورات آن از صفحه‌کلید گرفته می‌شود.
util.py	ابزارها و داده‌ساختارهای کمکی که می‌توانید در تمرین‌ها از آن‌ها استفاده کنید.

سایر فایل‌ها که صرفاً برای پیاده‌سازی بازی هستند. می‌توانید آن‌ها را رد کنید.

graphicDisplay.py, graphicUtils.py, layout.py, projectParams.py, test*.py

زمین بازی یک صفحه‌ی دوبعدی است که هر خانه‌ی آن یا دیوار است یا خالی و طبیعتاً تنها در صورتی که آن خانه خالی باشد می‌توان وارد آن شد. ممکن است در هر خانه‌ی زمین یک غذا و یا یک کپسول موجود باشد. همچنین همه‌ی عامل‌های بازی می‌توانند به وضعیت تمام زمین از جمله غذاها، دیوارها، کپسول‌ها و همچنین محل و جهت سایر عامل‌ها دسترسی داشته باشند.

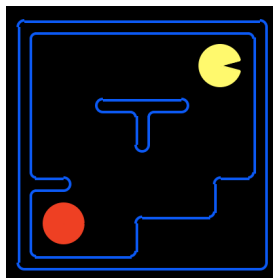
در این فریمورک تقریباً تمام بازی پیاده‌سازی شده است؛ وظیفه‌ی شما تنها پیاده‌سازی یک عامل هوشمند است که کنترل شخصیت پک‌من یا یکی از روح‌ها را بر عهده می‌گیرد. کلاس Agent به همین منظور تعبیه شده است. در هر مرحله موتور بازی وضعیت همه‌ی المان‌های بازی را محاسبه می‌کند و سپس با فراخوانی متد `getAction` از این کلاس و همچنین پاس‌دادن وضعیت زمین به آن، حرکت بعدی عامل را درخواست می‌کند. این روند تا پایان بازی تکرار خواهد شد.

۳. حالت‌های بازی:

این فریم‌ورک دو حالت مختلف را در خود دارد. حالت اول همان پک‌من کلاسیک است که شخصیت پک‌من باید غذاهای روی زمین را بخورد و همچنین از روح‌ها باید فرار کند. حالت دیگر، «حالت جست‌وجو» است به این صورت که پک‌من باید از نقطه‌ای شروع کند و به هدف مشخصی برسد. حال ممکن است این هدف صرفاً مکان خاصی در زمین باشد یا گذشتن از ۴ گوشه‌ی زمین و یا حتی خوردن همه غذاها و یا حتی همه‌ی این‌ها با هم. کاملاً می‌توان مسائل و فضای جست‌وجوی دلخواهی را برای آن تهیه کرد.

این سری از تمرین‌ها فقط در مورد حالت دوم است. در ادامه برای حل سوالات مربوط به جست‌وجو نیاز نیست عامل را از اول پیاده‌سازی کنید. فریم‌ورک این را برای شما فراهم کرده است. کلاس SearchAgent برای این حالت طراحی شده‌است. این کلاس دو ورودی می‌گیرد ۱- الگوریتم جست‌وجو ۲- مساله جست‌وجو. در این تمرین مسائل جست‌وجو مختلفی را خواهید دید، بعضی از آن‌ها برای شما پیاده‌سازی شده‌اند و بعضی هم به عهده‌ی شماست. همچنین در سوالات ابتدایی شما چند الگوریتم جست‌وجو را نیز پیاده‌سازی خواهید کرد.

برای الگوریتم جست‌وجو، کافی است تابعی را پیاده‌سازی کنید که مساله‌ی جست‌وجو را به عنوان ورودی گرفته و دنباله‌ای از حرکاتی که پک‌من باید انجام دهد تا به هدف مساله برسد را به عنوان خروجی برگرداند. حرکتهایی که پک‌من می‌تواند انجام دهد شامل حرکت به سمت شمال، جنوب، شرق، غرب و یا ایست است.



```
from game import Directions

def search_algorithm(problem):
    s = Directions.SOUTH
    w = Directions.WEST
    return [s, s, w, s, w, w, s, w]
```

همان‌طور که در مثال ساده‌ی روبه‌رو مشاهده می‌کنیم، این تابع برای رسیدن پک‌من به مقصد (نقطه‌ی قرمز) دنباله‌ای شامل ۸ حرکت را خروجی می‌دهد (در مثال بالا از پارامتر problem استفاده نشده است اما در ادامه به این پارامتر نیاز خواهید داشت).

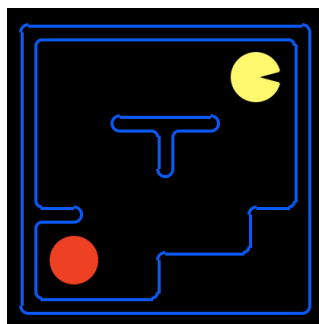
پارامتر problem متغیری از جنس کلاس SearchProblem است. این کلاس، کلاسی انتزاعی و عمومیست که بیان‌گر و مدل‌کننده‌ی هر نوع مساله‌ی جست‌وجو و فضای مربوط به آن است. بنابراین هر مساله‌ای باید جداگانه آن را پیاده‌سازی بکند. این کلاس به شما حالت شروع، حالت هدف (پایان)، حرکتهای مجاز از یک حالت خاص و همچنین هزینه‌ی هر دنباله‌ی دلخواهی از حرکات را برمی‌گرداند.

```
class SearchProblem:
    def getStartState(self)
    def isGoalState(self, state)
    def getNextStates(self, state)
    def getCostOfActions(self, actions)
```

- `getStartState()`: این تابع `state` شروع جست‌وجو را به شما می‌دهد؛ به طور مثال در سوالات ابتدایی که مساله جست‌وجو فقط براساس مکان است، خروجی این تابع مختصات نقطه‌ی شروع پک‌من در نقشه است.
- `isGoalState(state)`: این تابع یک `state` می‌گیرد و اگر آن حالت هدف (مقصد) جست‌وجو باشد مقدار `True` برمی‌گرداند و در غیر این صورت `False`.
- `getNextStates(state)`: این تابع با گرفتن یک `state`، حالت‌های بعدی را که می‌توان با حرکات مجاز رفت، خروجی می‌دهد. هر آئتم از این لیست یک سه‌تایی است که به ترتیب: حالت جدید، حرکت لازم برای رسیدن به آن و هزینه‌ی انجام این حرکت.
- `getCostOfActions(actions)`: این تابع لیستی از حرکات‌ها را می‌گیرد و هزینه‌ی این دنباله را حساب می‌کند. طبیعتاً همه‌ی حرکات باید مجاز باشند.

حال هر مساله‌ی جست‌وجویی با توجه به شرایطش باید این توابع را پیاده‌سازی کند. به طور مثال اگر مساله، جست و جو در گراف باشد، این کلاس باید ریشه، گره‌ی هدف، بچه‌های هر گره و هزینه‌ی حرکات را خروجی دهد (در این جا هزینه‌ی همه‌ی حرکات برابر یک است).

یک مساله‌ی جست‌وجوی دیگر، مساله‌ی پیدا کردن یک نقطه‌ی خاص در نقشه‌ی بازی پک‌من است. این کلاس به صورت پیش‌فرض برای شما پیاده‌سازی شده است (کلاس `PositionSearchProblem` در فایل `searchAgents.py`) به طور مثال در این مساله، کلاس ذکر شده باید مختصات نقطه‌ی شروع، مختصات نقطه‌ی پایان و مکان‌هایی را که با هر حرکت به آن می‌رسیم خروجی دهد. در صفحه‌ی بعد عملکرد این کلاس را می‌توان مشاهده کرد، توجه کنید که شکل سمت چپ تصویری از نقشه را نشان می‌دهد. (مبدا مختصات پایین سمت چپ است)



```
def search_algorithm(problem):
    # problem is an instance of PositionSearchProblem
    print problem.getStartState()
    print problem.isGoalState((5, 5))
    print problem.isGoalState((1, 1))
    print problem.getNextStates(problem.getStartState())
    return [...]
```

```
(5, 5)
False
True
[ ( (5, 4), 'South', 1 ), ( (4, 5), 'West', 1 ) ]
```

خروجی

برای اجرای حالت جست‌وجو، از دستور زیر می‌توانید استفاده کنید:

```
$ python pacman.py -p SearchAgent -a fn=<search_fn>,prob=<search_problem> -l=<search_map>
```

- `<search_fn>`: نام تابعی است که الگوریتم جست‌وجو را پیاده‌سازی می‌کند و حتماً باید در فایل `searchFunctions.py` موجود باشد.
- `<search_problem>`: نام کلاسی است که مسالهی جست‌وجو را پیاده‌سازی می‌کند و حتماً باید در فایل `searchProblems.py` موجود باشد. اگر این پارامتر را مقدار ندهید، به صورت پیش‌فرض مسالهی جست‌وجوی مکانی بارگذاری می‌شود.
- `<search_map>`: نام نقشه‌ای است که از این حالت پشتیبانی می‌کند (در هر سوال نقشه‌ی مورد نظر به شما گفته می‌شود)

به طور مثال:

```
$ python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

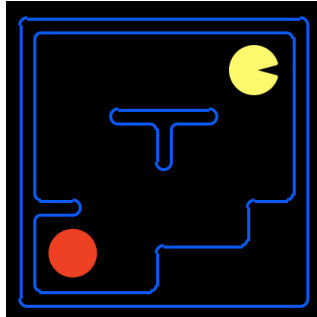
نکته: در سوال‌های پیش‌رو مساله جست‌وجوی مورد بحث، جست‌وجوی مکانیست (مگر خلاف آن گفته شود).

سوال‌های عملی

۱. اولویت اعمال (۱۰ نمره)

عامل پک‌من در هر لحظه با توجه به حالتی که در آن قرار دارد، action های مختلفی را می‌تواند انجام دهد. برای مثال در شکل روبه‌رو، پک‌من می‌تواند جهت پایین و چپ را انجام دهد. یکی از روش‌های یافتن مسیر درست در یک Maze، تعیین اولویت ثابت برای action هاست. برای مثال یک اولویت‌بندی (نه لزوماً درست) می‌تواند این باشد که همیشه در صورت امکان جهت شمال انتخاب شود. در صورت عدم وجود جهت شمال، جهت جنوب انتخاب شود و ...

با تعیین اولویت‌بندی درست action های پک‌من، آن را به نقطه‌ی نهایی هدایت کنید.



نکته: اعمالی که قصد اولویت دادن به آن‌ها را دارید، می‌توانند نسبی باشند یا قطعی.

برای پاسخ به این سوال باید بدنه تابع `action_priority_search(problem)` را در فایل `searchFunctions.py` پر کنید. خروجی تابع، دنباله‌ای از حرکت هاست. برای تست کد خود می‌توانید از دستورات زیر استفاده کنید:

```
$ python pacman.py -l tinyMaze -p SearchAgent -a fn=action_priority_search
$ python pacman.py -l rhmsMaze -p SearchAgent -a fn=action_priority_search
```

۲. جست‌وجو در فضای مساله – اول عمق (۲۰ نمره)

حالا وقت آن است که با الگوریتم بهتری به دنبال هدف بگردیم. در کلاس، الگوریتم جست‌وجوی اول-عمق (dfs) را یاد گرفتیم. در این سوال شما باید این الگوریتم را پیاده‌سازی کنید. توجه داشته باشید پیاده‌سازی الگوریتم نباید مختص به هیچ مساله‌ی جست‌وجوی خاصی باشد، بلکه باید کاملاً عمومی باشد تا هر نوع مساله‌ی جست‌وجویی که با استفاده از کلاس `SearchProblem` پیاده‌سازی شده باشد را حل کند. در سوال‌های بعد به این پیاده‌سازی نیاز خواهید داشت. توجه داشته باشید برای اینکه پیاده‌سازی شما کامل باشد نباید حالت‌هایی که قبلاً دیده‌است را دوباره گسترش دهد. برای پاسخ به این سوال باید بدنه‌ی تابع `dfs(problem)` را در فایل `searchFunctions.py` پر کنید. خروجی تابع، دنباله‌ای از حرکت هاست. برای پیاده‌سازی خود می‌توانید از داده‌ساختارهایی که در فایل `util.py` آمده‌است استفاده کنید. هم‌چنین درستی کد خود را با دستورات زیر تست کنید:

```
$ python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
$ python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
$ python pacman.py -l bigMaze -p SearchAgent -a fn=dfs -z 0.5
```

نکته: بازی به ازای هر مکانی در نقشه که الگوریتم شما بررسی می‌کند، رنگ قرمزی روی آن می‌کشد. هرچه رنگ قرمز روشن‌تر باشد یعنی این مکان زودتر بررسی شده است و هرچه تیره‌تر، دیرتر. آیا روند بررسی خانه‌های نقشه همانیست که انتظار داشتید؟ آیا پک‌من برای رسیدن به جواب، تمامی مکان‌ها را بررسی می‌کند؟

حال فرض کنید برای مساله‌ی جست‌وجو، جوابی وجود نداشته باشد. به طور مثال پک‌من در مکان بسته قرار بگیرد. پاسخ پیاده‌سازی شما چه خواهد بود؟ برای تکمیل پیاده‌سازی خود، آن را طوری تغییر دهید که اگر جوابی وجود نداشت یک لیست فقط شامل Directions.STOP برگرداند. برای تست کد خود می‌توانید از دستورات زیر استفاده کنید:

```
$ python pacman.py -l trappedPacman -p SearchAgent -a fn=dfs
$ python pacman.py -l unreachableGoal -p SearchAgent -a fn=dfs
```

۳. جست‌وجو در فضای مساله – اول سطح (۲۰ نمره)

در این سوال باید الگوریتم اول-سطح (bfs) را پیاده‌سازی کنید. یکی از ویژگی‌های این الگوریتم تضمین رسیدن به پاسخ بهینه (از لحاظ هزینه) است. برای همین اگر در تست‌های زیر برنامه‌ی شما به پاسخ بهینه نرسید باید دوباره آن را بررسی کنید.

برای پاسخ به این سوال باید بدنه‌ی تابع bfs(problem) را در فایل searchFunctions.py پر کنید. خروجی تابع، دنباله‌ای از حرکت‌هاست. برای تست کد خود می‌توانید از دستورات زیر استفاده کنید:

```
$ python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
$ python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z 0.5
```

۴. غذاخوردن سطر به سطر (۲۰ نمره)

در سوالات قبل روش‌هایی برای گشتن در فضای جست‌وجو پیاده‌سازی کردیم و آن‌ها را در مساله‌ی نسبتاً ساده‌ی PositionSearchProblem تست کردیم. از آنجایی که قرار بود پیاده‌سازیمان عمومی باشد بنابراین باید بتوانند در هر مساله‌ی جست‌وجویی کار بکنند. در این سوال می‌خواهیم مساله‌ی دیگری را پیاده‌سازی کنیم. مساله‌ی جست‌وجو این است که پک‌من باید قبل از رسیدن به مقصد مشخصش، ابتدا تمامی غذاهای روی زمین را بخورد. با این شرط که روش حرکت پک‌من باید سطر به سطر باشد. (تضمین می‌شود پک‌من از گوشه‌ی بالای سمت چپ شروع به حرکت می‌کند). بنابراین برای پیاده‌سازی این مساله، شما باید کلاسی بنویسید که از SearchProblem ارث‌بری کرده باشد و متدهای آن را با توجه به این مساله‌ی خاص پر شده باشد.

برای راحتی کار بهتر است ابتدا فضای حالت را برای این مساله‌ی خاص در نظر بگیرد و جزئیات آن را به دست آورد. توجه کنید فضای حالتی که در نظر می‌گیرید نباید اطلاعات غیر ضروری‌ای در خود داشته باشد چرا که باعث می‌شود اندازه‌ی فضای حالات شما بیهوده بزرگ شود.

برای پاسخ به این سوال، شما باید کلاس `LineByLineProblem` موجود در فایل `searchProblems.py` را پر کنید. برای تست کد خود می‌توانید از دستورات زیر استفاده کنید:

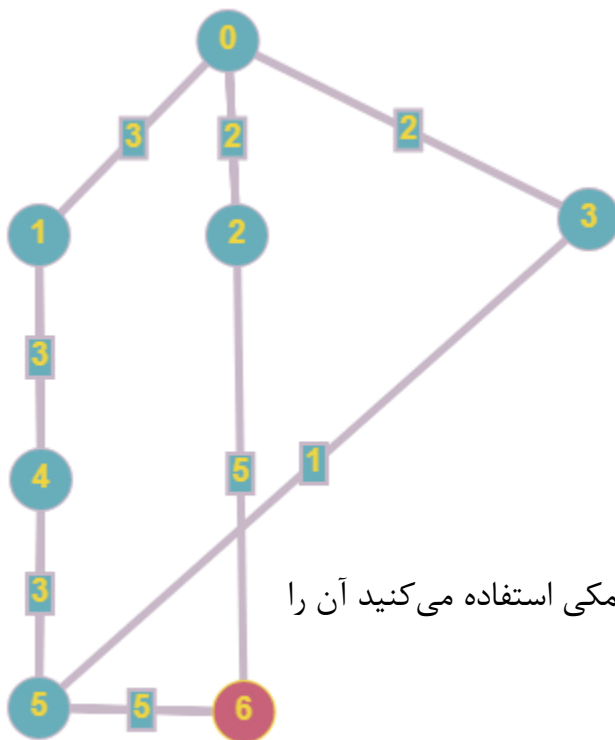
```
$ python pacman.py -l openSearch -p SearchAgent -a fn=bfs,prob=LineByLineProblem
$ python pacman.py -l mediumCFoodMaze -p SearchAgent -a fn=bfs,prob=LineByLineProblem
$ python pacman.py -l bigCFoodMaze -p SearchAgent -a fn=bfs,prob=LineByLineProblem -z 0.5
```

راهنمایی: برای مثال می‌توانید کلاس `PositionSearchProblem` را مرور کنید. پیاده‌سازی شما در این سوال، بیشتر در توابع `getNextStates` و `isGoalState` تفاوت خواهد داشت.

سوال های تئوری

۱. پیمایش گراف (۱۰ نمره)

در گراف زیر، یک گرهی ابتدایی (0) داریم که جستوجو را از آن شروع می‌کنیم و یک گرهی انتهایی (6) داریم کخ وقتی به آن برسیم، جستوجو به پایان رسیده‌است. در هر نوع از عملیات جستوجوی خواسته‌شده، اول به ترتیب نام گره‌هایی که مشاهده می‌شوند و سپس ترتیب گره‌هایی که بعد از پایان عملیات جستوجو برای رسیدن از 0 به 6 طی خواهیم کرد را بنویسید. (برای عملیات جستوجو ممکن است گره‌های مازاد بر مسیر اصلی را مشاهده کنید)

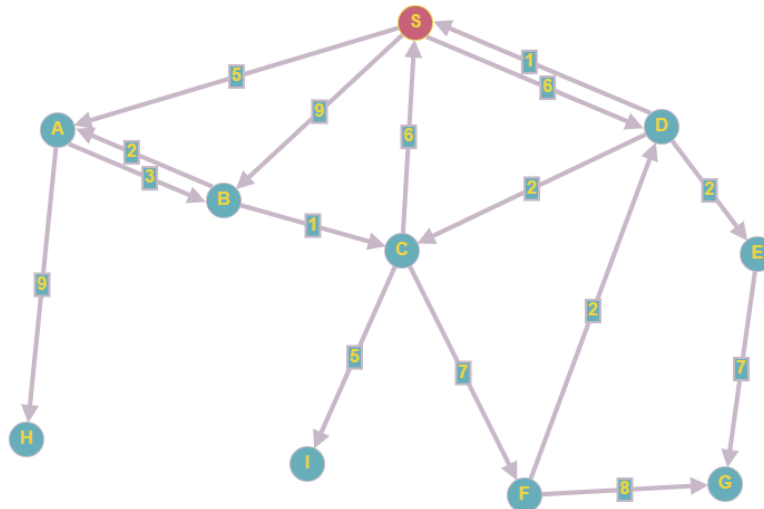


الف - DFS

ب - BFS

ج - UCS (اگر برای این روش از جدول یا گراف کمکی استفاده می‌کنید آن را ضمیمه کنید)

۲. پیمایش درخت (۱۰ نمره)



در گراف بالا یک گرهی ابتدایی (S) و سه گروهی هدف (G,I,H) را داریم که رسیدن به هر کدام از گره‌های هدف، معادل رسیدن به هدف اصلی است (گره‌های هدف از نظر ارزش باهم تفاوت ندارند).

الف- در درخت به روش DFS از گرهی S شروع کنید و ترتیب مشاهده‌ی تمام گره‌های موجود در درخت را بنویسید.

ب- در درخت به روش BFS از گرهی S شروع کنید و ترتیب مشاهده‌ی تمام گره‌های موجود در درخت را بنویسید.

ج- در درخت به روش UCS از گرهی S شروع کنید و ترتیب مشاهده‌ی تمام گره‌های موجود در درخت را بنویسید و در نهایت مراحل و هزینه‌ی هر کدام از هدف‌ها را مشخص کنید.

۳. جواب یکتا! (۱۰ نمره)

فرض کنید یک گراف داریم که فقط یک گرهی شروع و یک گرهی هدف دارد. ما به سه روش DFS, BFS و UCS در این گراف به جست‌وجو می‌پردازیم. در کدام روش‌ها جواب حاصل یکتاست؟ چرا؟ در کدام روش‌ها جواب حاصل یکتا نیست؟ چرا؟ (بلندی و کوتاهی پاسخ اهمیتی ندارد. منظور خود را به هر روش که مایل هستید انتقال دهید. ذکر مثال می‌تواند کمک کننده باشد)