**تمرین سری اول:** جست‌وجو در فضای حالات

لطفاً به نکات زیر توجه کنید:

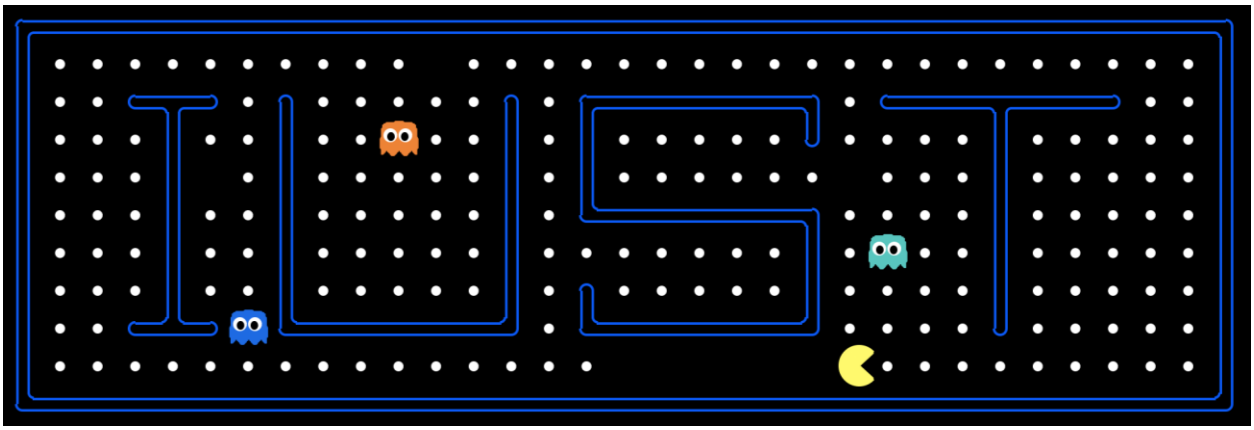
- مهلت ارسال این تمرین ۲۹ مهر است.
- در صورتی که به اطلاعات بیشتری نیاز دارید می‌توانید به صفحه‌ی تمرین در وب‌سایت درس مراجعه کنید.
- این تمرین شامل سوال‌های برنامه‌نویسی می‌باشد، بنابراین توجه کنید که حتماً موارد خواسته‌شده در سوال را رعایت کنید.
- ما همواره هم‌فکری و هم‌کاری را برای حلِ تمرین‌ها به دانشجویان توصیه می‌کنیم. اما هر فرد باید تمامی سوالات را به تنهایی تمام کند و پاسخ ارسالی حتماً باید توسط خود دانش‌جو نوشته‌شده باشد. لطفاً اگر با کسی هم‌فکری کردید نام او را ذکر کنید. **در صورت پیدا کردن تقلب نمره تمرین برابر صفر و به اندازه یک تمرین دیگر نمره منفی دارد.**
- لطفاً برای ارسال پاسخ‌های خود از راهنمای موجود در صفحه‌ی تمرین استفاده کنید.
- هر سوالی درباره‌ی این تمرین را می‌توانید در گروه درس مطرح کنید و یا از دستیاران حلِ تمرین بپرسید.

  - آدرس صفحه‌ی تمرین:

    https://iust-courses.github.io/ai981/assignments/01_search_problems
  - آدرس گروه درس:

    https://iust-courses.github.io/ai981

# دنیای پَک‌من



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 جست‌و جو در فضای حالات، یادگیری تقویتی و استنتاج احتمالی است.

قبل از اینکه به اولین سوال بپردازیم، 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
```

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

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
```

**۲. ساختار فایل‌ها:**

نکته: این فریم‌ورک با زبان پایتون نوشته شده است. بنابراین برای انجام تمرین‌ها نیاز به کمی آشنایی با زبان پایتون دارید. در صورت نیاز می‌توانید از این‌جا 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 دیدِ بهتری از ساختار فریم‌ورک به شما می‌دهد.

| ماژول‌های اصلی، بهتر است نگاهی به آن‌ها بیندازید. | |
|---|---|
| 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. | `paceman.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. | `game.py` |
| چند مورد از عامل‌های کنترل‌کننده‌ی پک‌من در این ماژول پیاده‌سازی شده‌اند. | `pacmanAgents.py` |
| چند مورد از عامل‌های کنترل‌کننده‌ی روح‌ها در این ماژول پیاده‌سازی شده‌اند. | `ghostAgents.py` |
| عامل کنترل‌کننده که دستورات‌آن از صفحه‌کلید گرفته می‌شود. | `keyboardAgents.py` |

[۲] Agent

ابزارها و داده‌ساختارهای کمکی که می‌توانید در تمرین‌ها از آن‌ها استفاده کنید. `util.py`

سایر فایل‌ها که صرفا 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:

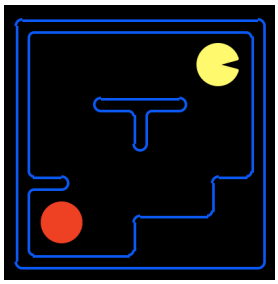
این فریم‌ورک دو حالت مختلف را در خود دارد. حالت اول همان پک‌من کلاسیک است که شخصیت پک‌من باید غذاهای روی زمین را بخورد و هم‌چنین از روح‌ها باید فرار کند. حالت دیگر، «حالت جست‌وجو» است به این صورت که پک‌من باید از نقطه‌ای شروع کند و به هدف مشخصی برسد. حال ممکن است این هدف صرفاً مکان خاصی در زمین باشد 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.

همان‌طور که در مثال ساده‌ی بالا مشاهده می کنیم، این تابع برای رسیدن پک‌من به مقصد (نقطه‌ی قرمز)



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

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.

- 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.

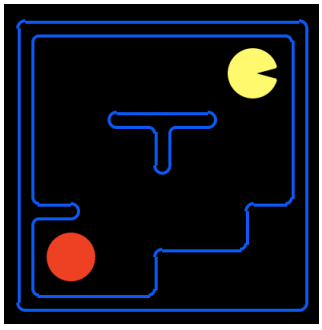
```
class SearchProblem:
    def getStartState(self)
    def isGoalState(self, state)
    def getNextStates(self, state)
    def getCostOfActions(self, actions)
```

- 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 آیتم از این لیست یک سه‌تایی است که به ترتیب: حالت جدید، حرکت لازم برای رسیدن به آن وهزینه‌ی انجام این حرکت.
- getCostOfActions(actions): این تابع لیستی از حرکت‌ها 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 این‌جا هزینه‌ی همه‌ی حرکات برابر یک است).

یک مساله‌ی جست‌وجوی دیگر، مساله‌ی پیداکردن 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)



```python
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 [...]
```

خروجی

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

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 پیاده سازی 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. جستجو اول‌عمق (۱۰ نمره)

در این مسأله شما باید یک عامل هوشمند پیاده‌سازی کنید که به صورت اول-عمق راه خروج را جستجو کند اما عامل شما هر مرحله فقط می‌تواند عمق محدودی از نقشه را مشاهده کند. به این صورت که ابتدا عمق قابل مشاهده برای عامل برابر ۱ و در مرحله بعدی در صورت پیدا نشدن راه خروج عمق به ۲ افزایش می‌یابد. این

روند تا پیدا شدن مسیر خروج ادامه می‌یابد. توجه داشته باشید پیاده‌سازی الگوریتم نباید مختص به هیچ مساله‌ی جستجو خاصی باشد، بلکه باید کاملًا عمومی باشد تا هر نوع مساله‌ی جستجویی که با استفاده از کلاس SearchProblem پیاده‌سازی شده باشد را حل کند. توجه داشته باشید برای اینکه پیاده‌سازی شما کامل باشد نباید حالت‌هایی در آن مرحله قبلاً دیده‌است را دوباره گسترش دهد.

برای پاسخ به این سوال باید بدنه‌ی تابع iddfs(problem) را در فایل searchFunctions.py پر کنید. خروجی تابع، دنباله‌ای از حرکت‌هاست. 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=iddfs
$ python pacman.py -l mediumMaze -p SearchAgent -a fn=iddfs
$ python pacman.py -l bigMaze -p SearchAgent -a fn=iddfs -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 عامل برای رسیدن به جواب، تمامی مکان‌ها را بررسی می‌کند؟

حال فرض کنید برای مساله‌ی جستجو، جوابی وجود نداشته باشد. به طور مثال عامل 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=iddfs
$ python pacman.py -l unreachableGoal -p SearchAgent -a fn=iddfs
```

## ۲. قایم موشک بازی !!!!!!! (۲۰ نمره)

در این سوال به پیاده سازی عامل پک‌من برای پیدا کردن گوشه‌های نقشه بازی که به فرم L میباشد پرداخته میشود به عنوان مثال شکل زیر یک گوشه میباشد:

توجه کنید که عامل شما باید تمام گوشه‌ها را پیدا کند.

عامل پک‌من شما باید بتواند به سمت یک گوشه در زمین بازی حرکت کند و پس از رسیدن به آن گوشه به سمت یک گوشه‌ی دیگر حرکت کند به شکلی که در انتها تمام گوشه‌های صفحه در کمترین زمان ممکن دیده شوند و سپس به سمت هدف حرکت کند.

برای پیاده سازی بدنه‌ی تابع hide_and_seek را در فایل searchFunctions.py پر کنید.

برای اجرا کد خود میتوانید از دستورات زیر استفاده کنید:

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

## ۳. غذاخوردن ستون به ستون(۲۵ نمره)

در این مسأله قرار است عامل‌هوشمند شما قبل از رسیدن به مقصد تمامی غذاهای موجود در نقشه را بخورد. با این شرط که روش حرکت عامل باید به صورت ستون به ستون باشد. بنابراین برای پیاده‌سازی این مسأله، شما باید کلاس بنویسید که از کلاس SearchProblem ارث‌بری کرده باشد و متدهای آن را با توجه به این مسأله خاص پر شده باشد.

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

برای پاسخ به این سوال، شما باید کلاس LineByLineProblem موجود در فایل searchProblems.py و همچنین بدنه تابع bfs(problem) در فایل searchFunction.py را پرکنید. 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 ScaryProblem را مرور کنید .پیاده‌سازی شما در این سوال، بیشتر در توابع getNextStates و isGoalState تفاوت خواهد داشت.

## ۴. پک‌من ترسو (۱۵ نمره)

الگوریتم‌هایی که تا اینجا پیاده‌ســـازی کرده‌اید هزینه حرکت از یک خانه به خانه دیگر را در نظر نمی‌گیرند حال در این ســوال قصــد داریم عامل پک‌من پیاده ســازی کنیم که از مبدا به مقصــد در کمترین زمان ممکن بر سد و همچنین نق شه بازی دارای تعدادی روح ا ست که هر روح در مرکز یک مربع سه در سه قرار دارد و پک‌من از وارد شدن به مربع‌های اطراف هر روح میتر سد بنابراین الگوریتم شما باید از وارد شدن پک‌من به محیط اطراف هر روح جلوگیری کند.

برای اینکار ابتدا cost_function را از کلاس ScaryProblem در فایل searchProblems.py را به شکلی کامل کنید که هزینه وارد شدن به مناطق نزدیک روح زیاد شود.

سپس الگوریتم UCS را در تابع ucs(problem) در فایل searchFunction.py پیاده‌سازی کنید.

می‌توانید از دستور زیر برای اجرا برنامه استفاده کنید:

```
$python pacman.py -l dangMaze -p SearchAgent -a fn=ucs,
prob=ScaryProblem
$python pacman.py -l bigDangMaze -p SearchAgent -a fn=ucs,
prob=ScaryProblem -z 0.5
```

تذکر: بهتر است از PriorityQueue در فایل util.py استفاده کنید.

# سوال‌های تئوری

## ۱. پیچیدگی (۱۰ نمره)

پیچیدگی زمانی و حافظه ای الگوریتم depth-limited search را توضیح دهید. نشان دهید که این الگوریتم از نظر زمان، پیچیدگی یکسانی با جستجو اول-سطح دارد. استفاده از آن چه مزیتی نسبت به جستجو اول-سطح دارد؟

## ۲. پیمایش گراف (۱۰ نمره)

در گراف زیر جستجو از گره A شروع میشود و با رسیدن به گره پایانی H تمام میشود. با استفاده از هر یک از الگوریتم های ذکر شده گراف را پیمایش کنید. برای هر یک از الگوریتم ها موارد زیر را بنویسید.

۱. ترتیب گره هایی که بعد از پایان جستجو برای رسیدن به هدف طی خواهیم کرد.

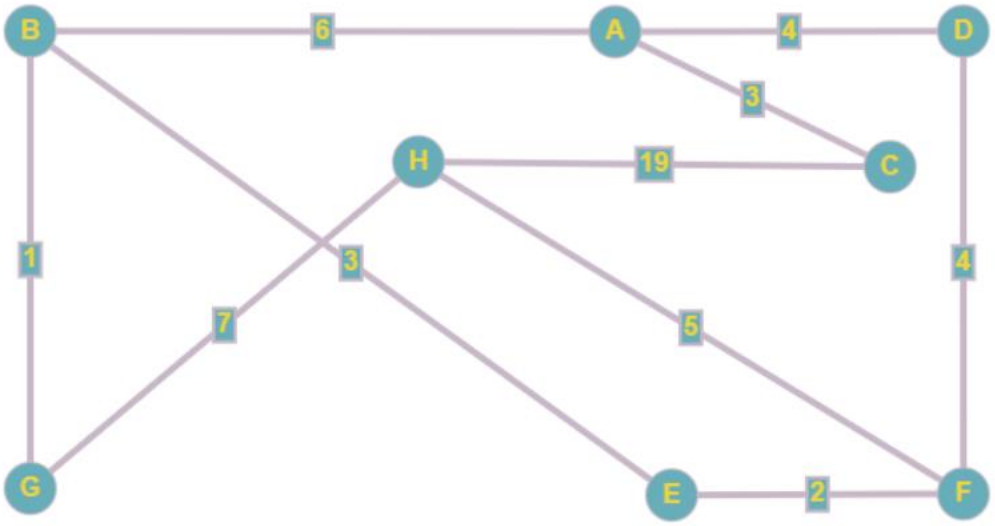
۲. ترتیب گره هایی که در مراحل الگوریتم مشاهده میشوند.

۳. در هر مرحله از اجرا الگوریتم محتویات فرینج را نمایش دهید و مشخص کنید چه گره‌ای در مرحله بعد بسط داده میشود.

الف- DFS

ب- BFS

ج- UCS



## ۳. پیمایش درختی (۱۰ نمره)

در گراف زیر گره ابتدایی A و گره پایانی H میباشد. با استفاده از الگوریتم های زیر جستجو درختی انجام دهید و برای هر الگوریتم موارد زیر را بنویسید. (در مورد تفاوت پیمایش درختی و گرافی جستجو کنید)

۱. ترتیب مشاهده گره ها

۲. محتویات فرینج در هر مرحله

الف- BFS

ب- UCS