# مبانی و کاربردهای هوش مصنوعی

## مرحله اول پروژه

**1403/08/11**

**یونس جمشیدی.....4013613020**

# بازی Pacman

در پروژه عامل هوشمندی پیاده سازی کرده ایم که در مورد نحوه دستیابی به غذاهای موجود در بازی معروف Pacman تصمیم گیری می کند. در این مسئله که Corner Problemنام دارد، از الگوریتم های جستجوی آگاهانه و ناآگاهانه برای هدایت عامل Pacmanبه سمت غذاها استفاده میشود. الگوریتم های در نظر گرفته شده برای پیاده سازی DFS ، UCSو*A هستند.

---

## الگوریتم جستجوی عمق‌اول - (DFS)

الگوریتم جستجوی عمق‌اول (Depth-First Search) یک روش پایه‌ای در هوش مصنوعی و نظریه گراف‌ها است که برای پیمایش گره‌های یک گراف یا درخت استفاده می‌شود. در این روش، الگوریتم تا حد امکان در یک شاخه به عمق می‌رود و سپس به نقطه قبلی بازمی‌گردد. این ویژگی باعث می‌شود DFS برای کاوش ساختارهای بزرگ و شاخه‌دار که نیاز به حافظه کمتری دارند، مناسب باشد. این پیاده‌سازی از DFS برای یافتن مسیر به هدف در یک مسئله جستجو طراحی شده است. از یک **پشته (LIFO)** برای مدیریت و کاوش گره‌ها استفاده می‌شود و گره‌ها به ترتیب کشف‌شدن به پشته اضافه می‌شوند. همچنین الگوریتم گره‌های بازدیدشده را ذخیره می‌کند تا از بازبینی آن‌ها اجتناب کرده و به این ترتیب در گراف‌های دارای حلقه دچار بی‌نهایت نشود.

**مراحل اجرا**

1. حالت اولیه را به پشته اضافه می کند.
2. حالت را از پشته بیرون می آورد.
3. اگر حالت هدف بود، مسیر را بازمی گرداند.

4. در غیر این صورت، اگر آن در حالت های بازدید شده نبود آن را به عنوان بازدیدشده علامت زده و فرزندهای آن را به پشته اضافه می کند.
5. تا زمانی که پشته خالی شود یا هدف پیدا شود، ادامه می دهد.

**مزایا:**

- **کارایی در مصرف حافظه** :تنها نیاز به ذخیره مسیر فعلی و گره‌های بازدید شده دارد.
- **پیاده‌سازی ساده** :از یک پشته ساده برای مدیریت مرزها استفاده می‌کند.
- **موثر در یافتن مسیرهای عمیق** :مناسب برای مسائلی که راه‌حل در عمق قرار دارد.

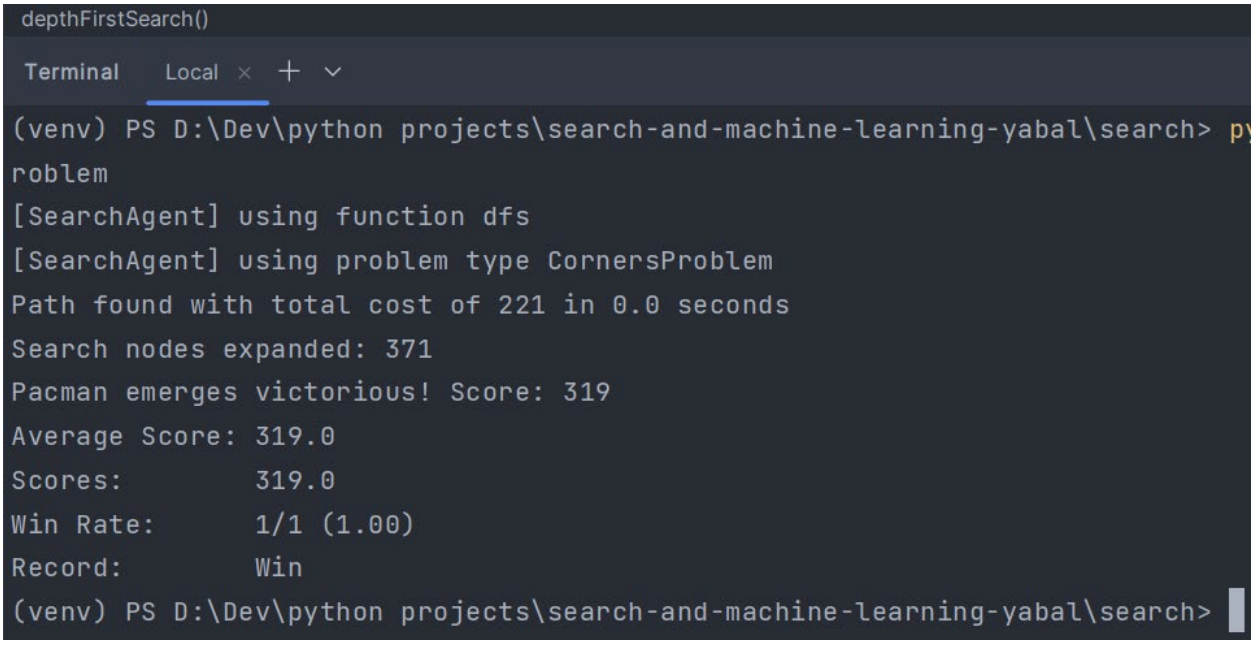
**معایب:**

- **ممکن است کوتاه‌ترین مسیر را پیدا نکند** DFS :تضمینی برای یافتن کوتاه‌ترین مسیر به هدف ندارد.
- **امکان گیر افتادن در شاخه‌های عمیق** :اگر شاخه‌های عمیق و غیرضروری وجود داشته باشد، DFSممکن است قبل از یافتن هدف وقت زیادی را صرف کاوش آن‌ها کند.

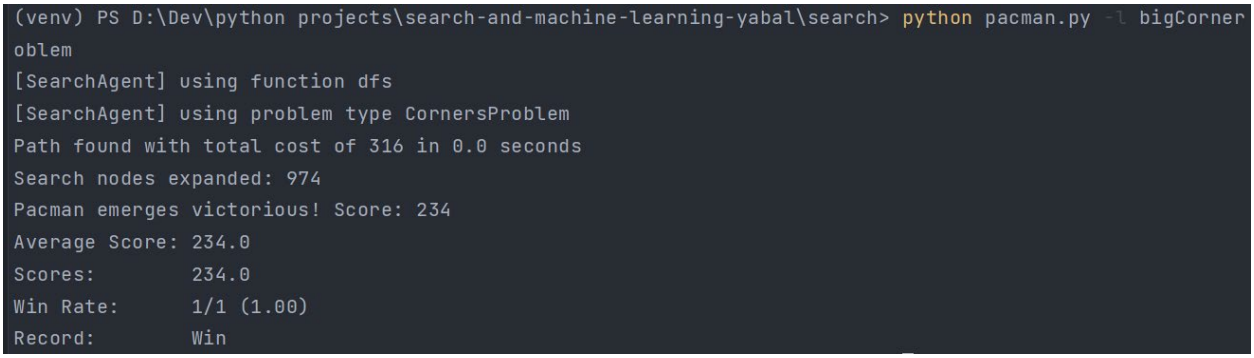
نتیجه simplecorner :

```
[SearchAgent] using function dfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 44 in 0.0 seconds
Search nodes expanded: 202
Pacman emerges victorious! Score: 466
Average Score: 466.0
Scores:        466.0
Win Rate:      1/1 (1.00)
Record:        Win
```

نتیجه hardcorner :




نتیجه bigcorner :




---

## الگوریتم جستجوی هزینه یکنواخت - (UCS)

الگوریتم جستجوی هزینه یکنواخت (Uniform Cost Search) یک روش جستجو است که برای یافتن کم‌هزینه‌ترین مسیر از یک گره شروع به یک گره هدف در یک گراف وزن‌دار استفاده می‌شود.

برخلاف جستجوی عمق‌اول یا جستجوی عرض‌اول، UCSهزینه هر مسیر را در نظر می‌گیرد و همیشه گره‌ای را که کمترین هزینه کل مسیر را دارد، ابتدا گسترش می‌دهد. این ویژگی UCSرا به یک استراتژی جستجوی بهینه برای یافتن کم‌هزینه‌ترین مسیر در گراف‌هایی که لبه‌ها وزن‌های متفاوتی دارند، تبدیل می‌کند.

این پیاده‌سازی از UCS از یک **صف اولویت** برای اولویت‌بندی گره‌ها بر اساس هزینه‌های تجمعی مسیر استفاده می‌کند. همچنین، گره‌های بازدیدشده (رسیده) را با هزینه‌های مربوطه ردیابی می‌کند تا اطمینان حاصل شود که هر گره با کمترین هزینه ممکن گسترش می‌یابد.

**مراحل اجرا**

1. وضعیت اولیه را با هزینه 0 به صف اولویت اضافه می کند.
2. گره با کمترین هزینه تجمعی(هزینه رسیدن از وضعیت اولیه به این گره) را از صف بیرون می کشد.
3. اگر گره هدف بود، مسیر را باز می گرداند.
4. هزینه مسیر را محاسبه می کند.
5. اگر آن حالت در دیکشنری reached_states نبود یا از هزینه محاسبه شده مسیر قبلی به این حالت کمتر بود، reached_states را با مقادیر جدید ست کرده و جانشین‌ها را با هزینه‌های به‌روز شده به صف اولویت اضافه می کند.
6. تا زمانی که صف خالی نشود یا هدف پیدا نشود، ادامه می دهد.

نتیجه simplecorner :

```
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 20 in 0.0 seconds
Search nodes expanded: 225
Pacman emerges victorious! Score: 490
Average Score: 490.0
Scores:        490.0
Win Rate:      1/1 (1.00)
Record:        Win
```

نتیجه hardcorner :

```
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

نتیجه bigcorner :

```
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 210 in 3.9 seconds
Search nodes expanded: 11392
Pacman emerges victorious! Score: 340
Average Score: 340.0
Scores:        340.0
Win Rate:      1/1 (1.00)
Record:        Win
```

---

## الگوریتم جستجوی - A*

الگوریتم جستجوی A* یک روش جستجوی آگاهانه است که عناصر هر دو روش جستجوی با هزینه یکنواخت (Uniform Cost Search) و جستجوی بهترین-اول (Greedy Best-First Search) را ترکیب می‌کند. با استفاده از یک تابع تخمینی (هیوریستیک) برای برآورد هزینه تا رسیدن به هدف، A*به‌طور مؤثر کم‌هزینه‌ترین مسیر را از گره شروع به گره هدف پیدا می‌کند. این الگوریتم در صورتی که هیوریستیک آن **قابل قبول** باشد (یعنی هزینه رسیدن به هدف را بیش از مقدار واقعی تخمین نزند) و **سازگار** باشد (یعنی نابرابری مثلثی را تضمین کند)، به بهینه‌ترین نتیجه دست پیدا می‌کند و ابزاری قدرتمند برای یافتن مسیر و پیمایش گراف‌ها است.

این پیاده‌سازی از A* از **صف اولویت‌دار** برای بررسی گره‌ها بر اساس مجموع هزینه‌ها استفاده می‌کند. هزینه هر گره شامل هزینه رسیدن به آن از گره شروع و هزینه تخمینی از آن گره تا گره هدف است.

**مراحل اجرا**

1. حالت اولیه را به صف اولویت‌دار اضافه می کند. برای اولویت آن نتیجه تابع هیوریستیک را پاس می دهد (از انجا که مقدار هزینه رسیدن به نود اولیه صفر است).
2. گره با اولیت بالاتر (کمترین مقدار مجموع دو تابع هزینه و هیورستیک) را از صف اولویت‌دار خارج می کند.
3. اگر گره هدف بود، مسیر را باز می گرداند.
4. هزینه مسیر را محاسبه می کند.
5. در غیر این صورت، گره را در reached_statesبه‌روزرسانی یا نادیده بگیرید بر اساس هزینه تجمعی.
6. جانشین‌ها را با هزینه‌های به‌روزشده به صف اولویت‌دار اضافه کنید
7. هزینه مسیر را محاسبه می کند.
8. اگر آن حالت در دیکشنری reached_states نبود یا از هزینه محاسبه شده مسیر قبلی به این حالت کمتر بود، reached_states را با مقادیر جدید ست کرده و جانشین‌ها را با هزینه‌های به‌روز شده به صف اولویت اضافه می کند.
9. تا زمانی که صف خالی نشود یا هدف پیدا نشود، ادامه می دهد.

**مزایا**:

- **بهینه و کارآمد** :با داشتن هیوریستیک مناسب و سازگار، مسیر کم‌هزینه را به‌صورت کارآمد پیدا می‌کند.
- **انعطاف‌پذیری بالا** A* :را می‌توان با هیوریستیک‌های مختلف برای برنامه‌های گوناگون سازگار کرد.

- **موثر برای گراف‌های پیچیده** :جستجو را به سمت هدف هدایت می‌کند و زمان محاسباتی را کاهش می‌دهد.

**معایب:**

- **استفاده بالای حافظه** :ذخیره گره‌ها در صف اولویت‌دار و پیگیری حالات بررسی‌شده می‌تواند در گراف‌های بزرگ منابع حافظه زیادی مصرف کند.
- **وابستگی به کیفیت هیوریستیک** :کارایی A* به شدت به کیفیت هیوریستیک مورد استفاده بستگی دارد.

نتیجه simplecorner :

```
(venv) PS D:\Dev\python projects\search-and-machine-learning-yabal\search> python pacman.py -l simpleCorner
ersProblem,heuristic=cornersHeuristic
[SearchAgent] using function astar and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 20 in 0.0 seconds
Search nodes expanded: 69
Pacman emerges victorious! Score: 490
Average Score: 490.0
Scores:        490.0
Win Rate:      1/1 (1.00)
Record:        Win
```

نتیجه hardcorner :

```
(venv) PS D:\Dev\python projects\search-and-machine-learning-yabal\search> python pacman.py -l hardCorner
sProblem,heuristic=cornersHeuristic
[SearchAgent] using function astar and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 692
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

نتیجه bigcorner :

```
(venv) PS D:\Dev\python projects\search-and-machine-learning-yabal\search> python pacman.py -l BigCorner
Problem heuristic=cornersHeuristic
[SearchAgent] using function astar and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 210 in 0.6 seconds
Search nodes expanded: 2668
Pacman emerges victorious! Score: 340
Average Score: 340.0
Scores:        340.0
Win Rate:      1/1 (1.00)
Record:        Win
```

---

## تابع – cornersHeuristic

تابع cornersHeuristicیک تخمین‌گر برای الگوریتم جستجوی A* است که برای حل مسئله "گوشه‌ها" در یک محیط شبکه‌ای (grid environment) استفاده می‌شود. در مسئله " Corners"، عامل باید از همه چهار گوشه هزارتو بازدید کند و از یک موقعیت اولیه شروع می‌کند. این تابع به تخمین هزینه باقی‌مانده برای بازدید از تمام گوشه‌های بازدیدنشده از یک حالت مشخص می‌پردازد و به این صورت به الگوریتم A* کمک می‌کند که مسیر بهینه را به‌طور کارآمد پیدا کند.

### استراتژی تخمین

این تخمین‌گر برای محاسبه حداقل فاصله لازم جهت بازدید از تمامی گوشه‌های بازدیدنشده طراحی شده است و به این ترتیب به جستجو کمک می‌کند تا به مسیر بهینه برسد. این تخمین به مراحل زیر تقسیم می‌شود:

1. **شناسایی گوشه‌های بازدیدنشده** :با توجه به unvisited_corners، گوشه‌هایی که قبلاً بازدید شده‌اند حذف می‌شوند و تمرکز فقط روی گوشه‌های باقی‌مانده است.
2. **انتخاب نزدیک‌ترین گوشه** :از موقعیت فعلی عامل شروع کرده و نزدیک‌ترین گوشه بازدیدنشده را با استفاده از فاصله منهتن پیدا می‌کند (که یک معیار فاصله مناسب برای یک محیط grid است).
3. **جمع‌آوری فواصل** :پس از انتخاب نزدیک‌ترین گوشه، تخمین‌گر:
    - فاصله تا این گوشه را به کل هزینه تخمینی اضافه می‌کند.
    - "موقعیت فعلی" را به این گوشه تغییر می‌دهد.
    - این گوشه را از لیست گوشه‌های بازدیدنشده حذف می‌کند.
4. **تکرار تا بازدید از همه گوشه‌ها** :مراحل 2-3 تکرار می‌شوند و در هر بار نزدیک‌ترین گوشه انتخاب می‌شود تا همه گوشه‌ها به صورت مجازی بازدید شوند. مجموع این فواصل کمینه به عنوان تخمین هزینه بازگشتی استفاده می‌شود.

**چرا این تخمین‌گر کار می‌کند؟**

- **روش حریصانه** :با حرکت به نزدیک‌ترین گوشه در هر مرحله، این تخمین‌گر یک تخمین حداقلی از هزینه لازم برای تکمیل مسئله ارائه می‌دهد و به‌طور موثری جستجو را هدایت می‌کند.
- **قابلیت پذیرش** (Admissibility) :این تخمین‌گر هزینه واقعی بازدید از همه گوشه‌های بازدیدنشده را بیش‌برآورد نمی‌کند، چراکه فقط به فاصله‌های مستقیم منهتن توجه دارد و در نتیجه تخمین پذیرفته‌شده‌ای ارائه می‌دهد. این تخمین محافظه‌کارانه است و همیشه نزدیک‌ترین گوشه را انتخاب می‌کند.