علی بازشوشتری ۹۹۲۳۰۰۸

گزارش پروژه‌ی اول مبانی و کاربردهای هوش مصنوعی

بخش صفر:

سوال)

کلاس SearchProblems یک کلاس انتزاعی است که ساختار مسئله‌ی جستجو را مشخص می‌کند.

متود getStartState حالت اولیه‌ی مسئله جستجو را برمی‌گرداند.

متود isGoalState به ما می‌گوید که حالت و وضعیتی که دریافت کرده، حالت هدف است یا خیر

متود getSuccessors یک حالت دریافت می‌کند و حالت‌های بعدی که می‌توان از این حالت ورودی به آنها را رفت را در قالب لیستی از tupleها برمی‌گرداند. این tupleها حاوی حالت بعدی، عمل لازم و هزینه‌ی عمل می‌باشد.

متود getCostOfActions لیستی از عمل‌ها(اقدامات) را دریافت می‌کند و مجموع هزینه‌ی دنباله‌ی آنها را برمی‌گرداند.

کاربرد کلاس‌های فایل game.py:

کلاس Actions: حاوی تعدادی متود استاتیک است برای اقدامات جابه‌جا شدن (move)

کلاس Configuartion: مختصات عامل را به همراه جهت حرکت نگه میدارد

کلاس Directions: ذخیره جهت‌های جغرافیایی و نسبی

کلاس Agent: پیاده سازی عامل و بازگردانی عمل‌های آن

کلاس Grid: صفحه مختصات ۲ بعدی مربوط به نقشه‌ی پک من(یک آرایه ی ۲بعدی)

کلاس AgentState: وضعیت عامل را نگه میدارد.

بخش ۱)

سوال)

پیچیدگی زمانی DFS در بهترین حالت O(bm) و در بدترین حالت O(b^m) است. در اینجا b تعداد فرزندان هر راس در گراف و m عمق درخت جستجوی DFS است. همچنین، پیچیدگی فضایی DFS برابر با O(bm) است که در بدترین حالت همانند پیچیدگی زمانی، به تعداد راس های درخت جستجو وابسته است

خیر جواب بهینه نمیدهد و اولین جوابی که به آن برسد را میدهد.

سوال)

الگوریتمIDS : این الگوریتم در واقع ترکیبی از الگوریتم bfs و dfs است . به طوری که در سطح های مختلف گراف الگوریتم dfs اجرا می شود. یک بار dfs را تا عمق اول اجرا میکند. بار بعدی تا عمق دوم. بار بعدی تا عمق سوم و همینطور الی آخر. همانطور که در کد زیر مشخص است ، تابع IDS در هر سطح(برای این کار از حلقه استفاده شده است) تابع DLS را صدا می زند. این تابع به صورت عمقی تا سطح مشخص شده جست و جو می کند و زمانی که به هدف رسید true و در غیر این صورت false بر می گرداند.

اگر مسئله تنها یک جواب در عمق زیاد داشته باشد، آنگاه dfs از ids بهتر خواهد بود چون جواب در هردو یکی است اما در dfs زمان کمتری صرف شده است

```
bool IDS(src, target, max_depth)
    for limit from 0 to max_depth
        if DLS(src, target, limit) == true
            return true
    return false

bool DLS(src, target, limit)
    if (src == target)
        return true;

    if (limit <= 0)
        return false;

    foreach adjacent i of src
        if DLS(i, target, limit?1)
            return true

    return false
```

بخش ۲)

سوال)

بله

سوال)

شبیه به bfs است اما هم از طرف گره‌ی مبدا به مقصد حرکت میکند و هم از طرف گره‌ی مقصد به مبدا حرکت میکند و این جرکت تا زمانی ادامه دارد که به یک گره ی مشترک برسند. برای مسائل جستجویی مناسب است که گره ی مبدا و مقصد مشخص هستند. به لحاظ زمانی از bfs بهینه تر است.

```
BIDIRECTIONAL_SEARCH
 1   Q_I.Insert(x_I) and mark x_I as visited
 2   Q_G.Insert(x_G) and mark x_G as visited
 3   while Q_I not empty and Q_G not empty do
 4       if Q_I not empty
 5           x ← Q_I.GetFirst()
 6           if x = x_G or x ∈ Q_G
 7               return SUCCESS
 8           forall u ∈ U(x)
 9               x' ← f(x, u)
10               if x' not visited
11                   Mark x' as visited
12                   Q_I.Insert(x')
13               else
14                   Resolve duplicate x'
15       if Q_G not empty
16           x' ← Q_G.GetFirst()
17           if x' = x_I or x' ∈ Q_I
18               return SUCCESS
19           forall u^-1 ∈ U^-1(x')
20               x ← f^-1(x', u^-1)
21               if x not visited
22                   Mark x as visited
23                   Q_G.Insert(x)
24               else
25                   Resolve duplicate x
26   return FAILURE
```

سوال)

از نظر زمانی هر دو به O(V+E) نیاز دارند که Vتعداد رئوس و E تعداد یال هاست .از نظر فضایی dfs به O(m) و bfsبه O(b^m) نیاز دارد که m عمق درخت است و b نیز branching factor میباشد.

Dfs از لحاظ پیچیدگی حافظه بهتر است اما bfs از جواب بهینه را پیدا میکند درحالی که dfs اینگونه نیست و الزاما جواب بهینه را پیدا نمیکند

بخش ۳)

سوال)

بله.

اگر تابع هزینه را برابر یک عدد ثابت درنظر بگیریم به bfs میرسیم چون در bfs ارزش افزوده ای به گره ها داده نمیشود و به ترتیب بازدید از گره ها مسیر کوتاهتر پیدا میشود.

برای dfs نیز تعداد تلاش ها برای رسیدن به هدف میتواند باشد

سوال)

مزایا:

تضمین پیدا کردن جواب بهینه

مناسب برای مسائل با یال های وزن دار

معایب:

پیاده سازی پیچیده تر

نیازمند حافظه ی بیشتر به جهت نگهداری هزینه ها

در مسائل با هزینه ی یکنواخت ممکن است کارامد نباشد

بخش ۴)

سوال)

Dfs اصلا به جواب بهینه نرسید و اولین جوابی که به آن رسید را برگرداند

Bfs و ucs مشابه یکدیگر رفتار کردند

*A از همه بهینه تر بود زیرا گره های کمتری را بسط داد

مسیر طی شده در همگی به جز dfs یکسان و بهینه بود و امتیاز کسب شده برابر بود

سوال)

دایجسترا کوتاه ترین مسیر از مبدا تا تمام گره ها را پیدا میکند در حالی که a* یک تابع هیوریستیک دارد که تخمینی از هزینه تا مقصد را به ما میدهد و با کنار قرار دادنش با هزینه ای که تا الان صرف شده تلاش میکند هرچه سریعتر و با صرف هزینه ی کمتر به مقصد برسد.

تفاوت اصلی آنها در همین تابع هیوریستیک میباشد به گونه ای که با انتخاب یک هیوریستیک مناسب، a* مسیرهایی را که احتمالا به مقصد ختم نمیشوند را بررسی نمیکند

	Dijkstra	A*
Graph	finite	both finite and infinite graphs
Edge costs	non-negative	strictly positive
Time complexity	$O(\|V\|^2)$ or $O(\|E\| + \|V\| \log \|V\|)$	$O(\|E\| + \|V\|)$ or $O\left(b^{\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor + 1}\right)$
Space complexity	$O(\|V\|)$	$O(\|V\|)$ or $O\left(b^{\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor + 1}\right)$
Search contours	uniform	stretched toward the goal(s)
In practice	slower than A*	fast if the heuristic is good
Computing	inside the algorithm	heuristics incur computational overhead
Input	require only the problem definition	designing a good heuristic requires time

بخش ۶)

سوال)

هیوریستیک: هربار فاصله ی منهتن تا گوشه‌ی ویزیت نشده محاسبه می‌شود و عامل به سمت نزدیک ترین گوشه حرکت می‌کند.

در هر بخش از مسیر، در بهترین حالت که هیچ مانعی نباشد، با توجه به نحوه حرکت عامل پک من، هزینه برابر با فاصله منهتن میشود و در صورت وجود موانع، افزایش می‌یابد، میتوان نتیجه گرفت که هیوریستیک قابل قبول و سازگار است. اگر فاصله تا گوشه ها یک مقدار مشخصی باشد، با حرکت به سمت یکی از گوشه ها، فاصله ی منهتن مربوط به آن گوشه یکی کاهش می‌یابد و برای باقی گوشه ها افزایش می‌یابد. از این رو هزینه‌ی کل در حال افزایش است پس هیوریستیک سازگار است.

بخش ۷)

سوال اول و دوم)

این هیوریستیک فاصله‌ی عامل تا غذاهای باقی مانده را با کمک mazeDistance بدست می آورد و سپس ماکسیمم را برمیگرداند. استدلال آن اینگونه است که عامل نمیتواند تمامی غذاها را در تعداد حرکت کمتری نسبت به آن ماکسیمم فاصله بخورد. اثبات سازگاری آن نیز کمی مشابه قبلی است به علاوه اینکه از mazeDistance استفاده میکنیم و مقداری نابزرگ تر از هزینه برگردانده می‌شود. همچنین چون فاصله‌ی واقعی از فاصله ی منهتن بزرگ تر مساوی است پس سازگار است با حرکت به سمت یک غذا(مشابه قسمت قبل) مقدار همگی به جز یکی افزایش میابد و درنتیجه هزینه کل افزایش میابد. در قسمت قبلی فاصله تا گوشه ها را بررسی میکردیم اما اینجا فاصله تا غذاهارا.

سوال۳)

زیرا در ارتباط با هدف اطلاعاتی را دریافت می کند و تخمینی از آن دارد(هیوریستیک) و از آن جهت رسیدن به هدف استفاده میکند(هدایت کردن جستجو) و سعی میکند گره هایی که در مسیر رسیدن به هدف هستند را ویزیت کند نه آنکه تمام گره ها را ویزیت کند.

بخش۸)

سوال)

مسلما با رفتن به نزدیک ترین خانه نمیتوان مطئن بود که بهینه ترین مسیر را طی میکنیم.

در مثال زیر عامل مسیر قرمز را طی میکند درحالی که بهینه نیست و بهتر بود عامل اول آن غذای بالایی را بخورد و سپس سراغ باقی غذاها برود.

علی بازشوشتری ۹۹۲۳۰۰۸