به نام خدا

سینا سلیمیان 810197528 گزارش کار پروژه اول هوش مصنوعی

الگویتم های سرچ:

در این پروژه قرار است به بررسی روش های مختلف جستجو برای حل مساله، نحوه پیاده سازی آن ها و تمایز و برتری هر یک نسبت به دیگری بپردازیم.

توضیح کلی پروژه:

در این پروژه یک نقشه داریم و مختصات شروع agent به ما داده شده است. هدف رسیدن به مختصات مقصد است با این شرط که تمام گوی هایی که در نقشه قرار گرفته اند در جای مخصوص خود قرار بگیرند و در این صورت است که ما به goal state می رسیم.

برای راحتی کار، یک کلاس Map داریم که تمام داده ها را پس از خواندن از فایل به فرمت ایده آل در می آوریم و در آن ذخیره می کنیم. همچنین نقشه را در یک آرایه دو بعدی map ذخیره می کنیم.

استیت هایمان را به این صورت تعیین می کنیم که هر استیت شامل مختصات نقطه ای از نقشه که در آن قرار داریم، مختصات مبدا و مقصد توپ هایی که تا به آن زمان برداشته نشده اند و مختصات مقصد توپ هایی که در کوله پشتی مان قرار دارند می باشد. همچنین اطلاعات اضافه مانند عمق نود و مقدار heuristic و توپ های داخل کوله پشتی را در استیت ذخیره می کنیم ولی عامل تمایز استیت ها از هم، 3 صفت اول هستند.

یک کلاس نود برای نگهداری پدر، استیت، نوع حرکت بعدی در مختصات استیت که منجر به رسیدن به goal state نیز داریم.

6 حالت برای بوجود آمدن استیت های مختلف وجود دارد(4 جهت بالا، راست، پایین، چپ و برداشتن توپ و قرار داردن توپ) که در همه ی حالت به جز حالت قرار دادن توپ(drop) ارتفاع درخت یک واحد زیاد می شود. در حالتی که توپ را قرار می دهیم، یک استیت جدید می سازیم ولی آن را expand نمی کنیم.

تمام استیت های دیده شده را در دیکشنری explored ذخیره می کنیم و همچنین تمام نود های قابل بررسی را در آرایه frontier ذخیره می کنیم.

در هر دور یک نود از frontier بسته به الگوریتم مورد نظر انتخاب کرده و pop می کنیم و در explored قرار می دهیم و استیت های جدید که به واسطه ی استیت انتخاب شده ممکن است به وجود بیایند را در frontier اضافه می کنیم.

برای initial state ، مقدار coordinates را برابر مختصات داده شده و مقدار توپ های باقی مانده را برابر مختصات تمام توپ ها و توپ های داخل کوله پشتی را صفر قرار می دهیم.

برای goal state ، مقدار coordinates را برابر مختصات داده شده و مقدار توپ های باقی مانده و توپ های داخل کوله پشتی را برابر صفر قرار می دهیم.

توضیح الگوریتم های پیاده سازی شده:

BFS: در این الگوریتم به این صورت عمل می کنیم که در ابتدا initial node در frontier قرار می گیرد و تا زمانی که frontier خالی نباشد، حالت های مختلف ساختن استیت جدید را بررسی می کنیم و اگر استیتی قابل ساختن بود (مختصات از نقشه خارج نشود و یا توپی در دست داشته باشیم) و قبلا آن را در frontier و یا explored قرار نداده باشیم(حذف حالات تکراری)، نود آن را در frontier قرار می دهیم و همچنین استیت آن را در explored قرار می دهیم تا سرعت بیشتر شود. اما قبل از این کار با تابع is_goal_state() بررسی می کنیم که استیتی که می خواهیم در دیکشنری قرار دهیم، استیت نهایی ما اگر باشد، به جواب رسیده ایم.

IDS: در این الگوریتم یک تابع ids() وجود دارد که الگوریتم depth limited search را از عمق صفر تا ده به توان 9 (یک عدد بزرگ) اجرا می کند و در هر عمق اگر به goal state رسیدیم، به جواب بهینه رسیده ایم. در dls() هر سری frontier و explored را initialize می کنیم و تا زمانی که frontier خالی نباشد، آخرین عضو آن را خارج کرده و در صورت مجاز بودن، استیت های جدید اضافه می کنیم. اگر جستجوی dfs به جواب نرسیدیم باشد این را در نظر بگیریم که برخی استیت ها در explored قرار دارند که ممکن است برای جستتجوی عمقی سایر فرزندان مشکل ایجاد کنند. برای حل این مشکل اگر به استیت تکراری رسیدیم، آن استیتی که عمق کمتری دارد را در explored نگه می داریم.

*A : مانند الگوریتم bfs عمل می کنیم با این تفاوت که در اینجا برای هر استیت یک مقدار heuristic در هنگام ساختن استیت محاسبه و اضافه می کنیم(با تابع calculate_heuristic) و همچنین به جای ذخیره نود ها در صف، آن ها را به صورت درخت heap در نظر می گیریم که با O(logn) به درخت اضافه می شوند و با O(1) مقدار استیت با مینیمم heuristic را پیدا می کنیم. از دو تابع heuristic استفاده می کنیم:

1.محاسبه ی heuristic به صورت:(manhattan)

```
abs(goal state (y) - current state (y)) +
abs(goal state (x) - current state (x))
```

این تابع admissible است زیرا بهترین و نزدیک ترین حالت برای رسیدن از یک نقطه به نقطه دیگر را به ما می دهد پس همیشه h(n) <= h*(n) می باشد.

همچنین این تابع consistent هم هست. برای برقراری این شرط باید داشته باشیم:

$$H(N) <= c(N,P) + h(p)$$

در حالتی که از کوتاه ترین مسیر خارج شویم، فرض می کنیم که با مسیری کوتاه تر به استیت نهایی رسیده ایم که این با فرض اولیه ما تناقض دارد و اگر به مختصات مجاور برویم و از آن جا بخواهیم به مقصد برسیم، در بهترین حالت مساوی با مسیر فعلی باید هزینه بپردازیم.

.2

```
max(abs(goal state (y) - current state(y),
abs(goal state (x) - current state (x))
```

این تابع جواب کوتاه تری نسبت به تابع اول به ما می دهد پس قطعا admissible است و همچنین اگر از یک مختصات به مختصات مجاور یا هر جای دیگر برویم شرط consistency میبینیم که برقرار است پس consistent است.

Weighted A* : در این الگوریتم به heuristic دوم یک ضریب آلفا می دهیم تا دقت را کاهش ولی سرعت را افزایش دهد.

Test1:

	فاصله جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	میانگین زمان اجرا
BFS	20	605	249	0.0023s
IDS	20	6471	3039	0.0190s
A*(1)	20	498	222	0.0020s
A*(2)	20	550	233	0.0026
Weighted A*(α=4)	26	578	245	0.0019s
Weighted A*(α=6)	24	437	187	0.0013s

BFS path:

Init State -> Right -> Right -> Right -> Down & Pick -> Down & Drop -> Left & Pick -> Right -> Up -> Up -> Left -> Left & Drop -> Right & Pick -> Right -> Right -> Right -> Down & Drop -> Right -> Goal State

Test2:

	فاصله جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	میانگین زمان اجرا
BFS	48	9014	2304	0.0220s
IDS	48	1439299	388504	3.2290s
A*(1)	48	710	299	0.0020s
A*(2)	48	3718	1036	0.0120s
Weighted A*(α=15)	48	190	99	0.0006s
Weighted A*(α=4)	48	199	99	0.0003s

BFS path: Init State -> Right -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down Down -> Down -> Down -> Down-> -> Down -> Down -> Down -> Down -> Down -> Down -> Goal State

Test3:

	فاصله جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	میانگین زمان اجرا
BFS	68	816587	287270	4.430s
IDS	68	8900666	3493091	27.833s
A*(1)	68	275939	107110	1.597s
A*(2)	68	283534	110900	1.714s
Weighted A*(α=15)	68	81572	35248	0.394s
Weighted A*(α=4)	68	124236	51528	0.635s

BFS path:

Init State -> Right & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Up & Pick -> Down & Drop -> Right -> Goal State

Test4:

	فاصله جواب	تعداد استیت دیده شده	تعداد استیت مجزای دیده شده	میانگین زمان اجرا
BFS	92	1019	457	0.0047s
IDS	92	222839	100036	0.5721s
A*(1)	92	1019	457	0.0048s
A*(2)	92	1019	457	0.0048s
Weighted A*(α=15)	92	739	337	0.0043s
Weighted A*(α=4)	92	722	331	0.0040s

BFS path:

Init State -> Right -> Right -> Right -> Down -> Down -> Right -> Right -> Right -> Right -> Right -> Right -> Down -> Down -> Right -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Left -> Left -> Left -> Down -> Down -> Down -> Left -> Left -> Left -> Down -> Down -> Down -> Down -> Down -> Right -> Right -> Right -> Right -> Right -> Right -> Right -> Down -> Down -> Right -> Right -> Right -> Right -> Right -> Right -> Up -> Up -> Up -> Up -> Right -> Right -> Right -> Up -> Up -> Right -> Right -> Right -> Up -> Up -> Up -> Right -> Right -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Down -> Left -> Left -> Left -> Down -> Down -> Down -> Down -> Down -> Right -> Right -> Right -> Right -> Right -> Right -> Right -> Goal State

نتیجه گیری:

در الگوریتم های uninformed search که محل goal state را نمی دانیم، بسته به اینکه مختصات مقصد کجا است هر الگوریتم می تواند نسبت به دیگری ارجحیت زمانی داشته باشد ولی در تمام این مثال ها الگوریتم BFS با سرعت بیشتری به جواب رسید.

در الگوریتم های informed search چون محل دقیق goal state را می دانیم، agent ما درکی از محل احتمالی دارد و هر استیتی که انتخاب می کند را با یک درکی نسبت به هدفش بر می گزیند به همین دلیل تعداد استیت های دیده شده برای رسیدن به جواب کمتر از uninformed search می شود.

با weighted A* می توانیم سرعت را برای سرچ افزایش دهیم ولی باید حواسمان باشد که این افزایش سرعت بهای کاهش دقت را به همراه دارد و الزاما مسیر بهینه را برای ما پیدا نمی کند.