محمدامین پورزارع CA1 - Report 810199392

در ابتدای کار الگوریتم کلی رو میگوییم و نحوه ی مدل کردن مسئله را شرح میدهیم و اینکه الگوریتم های search را باید بر روی چه چیزی پیاده سازی کنیم.

● شرح نحوه ی مدل کردن مسئله (initial state و goal state و action و ...)

در ابتدا ما گرافی داریم که از یکسری نود های صعب العبور و نود هایی که در آنها مرید و دستور پخت ها قرار دارند و یک نود که شروع حرکت سید را مشخص میکند. حال ما باید کوتاه ترین مسیر را به گونه ای پیدا کنیم که شرایط مسئله را برآورده کنیم.

برای این کار ابتدا مسئله مان را به یک درخت که از استیت هایی تشکیل شده است مدل میکنیم. به گونه ای عمل میکنیم که هیچ استیت تکراری ای در این درخت دیده نشود و در هر استیت شرایط موجود برای سید را بررسی میکنیم.

در نتیجه هر استیت شامل موارد زیر میشود:

Class State:

Position: int

NodeTime : int

DoneRecipes : list

DoneDevotees : list

Path : list

HardNodes : dict

Position : این متغیر پوزیشنمان را نشان میدهد یعنی نشان میدهد که ما در چه نودی از گراف مسئله ی مان هستیم.

NodeTime : این متغیر برای هندل کردن نود های صعب العبور به کار میرود به گونه ای که وقتی مجبور هستیم چند ثانیه در یک نود صعب العبور صبر کنیم در هر مرحله که میگذرد یک ثانیه به NodeTime اضافه میکنیم

DoneRecipes : در متغیر DoneRecipes دستور پخت هایی که دیده شده اند (یعنی سید از نود حاوی آن دستور پخت گذر کرده است) در یک لیست نگهداری میشوند.

DoneDevotees : در متغیر DoneDevotees مرید هایی که به طور کامل راضی شده اند(یعنی دستور پخت های درخواستی آنها دیده شده است و سپس از نود حاوی آن مرید سید گذر کرده است) در یک لیست نگهداری میشوند.

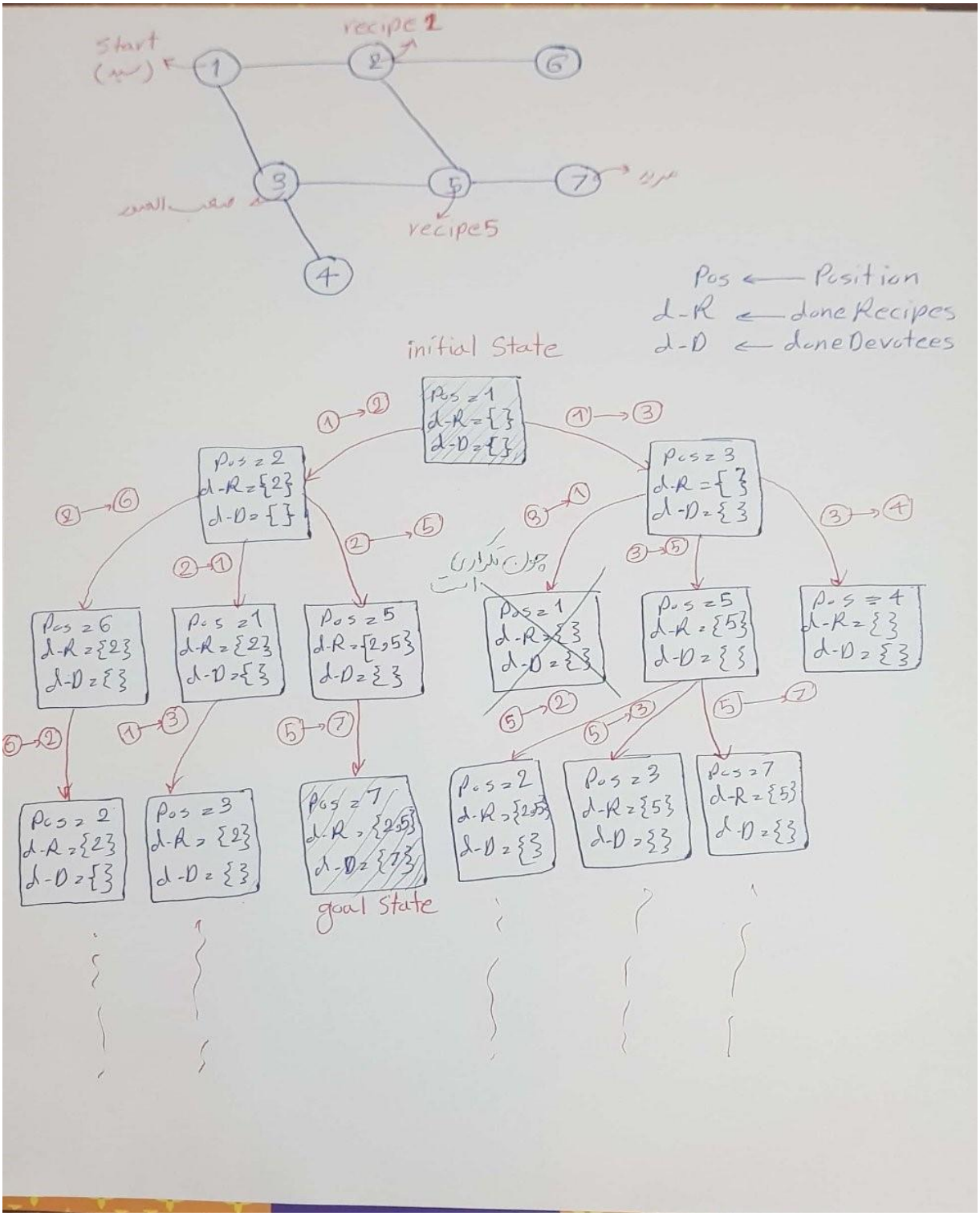
Path : متغیر Path یک لیست برای نگهداری مسیری میباشد که طی شده است

HardNodes : این متغیر به شکل دیکشنری میباشد که key های آن نود های صعب العبور هستند و value های آن تعداد عبور های سید از آن نود ها میباشد

حال باید درخت استیت مان رو شرح دهیم. درختمان به شکلی میباشد که ریشه ی آن initial state میباشد که position آن مکانی است که سید حرکت خود را از آن شروع میکند و ما بقی متغیر های آن هم با توجه به توضیحات بالا مقدار دهی میکنیم.

حال وقتی میخواهیم به استیتی دیگر برویم به همسایه های پوزیشن initial state در گراف نگاه میکنیم و اگر state تکراری ای به وجود نیاید استیت جدید را تشکیل میدهیم. این کار را تا زمانی انجام میدهیم تا به goal state برسیم. goal state به شکلی میباشد که در doneDevotees آن تمام مرید های موجود در گراف وجود داشته باشند. در نتیجه وقتی goal state را میابیم مسیر حرکت را چاپ میکنیم. برای هندل کردن نود های صعب العبور هم به شکلی عمل میکنیم که اگر باید در آن بمانیم در هر مرحله یک استیت جدید به وجود میآوریم که تفاوتش با استیت قبلی این است که NodeTime آن یکی بیشتر است.

حال درخت استیت هایمان را برای مثال بر روی گراف کشیده شده درعکس زیر نشان میدهیم و در شکل goal state و initial state هم مشخص کرده ایم :



● توضیح الگوریتم های پیاد ه سازی شده و تفاوت ها و مزیت های الگوریتم ها نسبت به یکدیگر و اینکه کدام الگوریتم ها جواب بهینه تولید می کنند.

در این پروژه ۴ الگوریتم مختلف را پیاده سازی کردیم که الگوریتم اولی BFS بود. میدانیم که الگوریتم BFS برای درخت به شکلی میباشد که از ریشه ی درخت شروع به پیمایش میکنیم و در مرحله ی بعد همسایه آن را که کمترین عمق را در درخت دارند پیمایش میکنیم و به همین شکل به کارمان ادامه میدهیم و تمام درخت را پیمایش میکنیم.

الگوریتم بعدی IDS میباشد.میتوان گفت این الگوریتم ایده اش از ترکیب BFS و DFS تشکیل شده است. به این شکل میباشد که در هر مرحله یک عمقی داریم که در ابتدا صفر میباشد و در هر مرحله که میگذرد عمق ما یکی بیشتر میشود. حال در هر مرحله الگوریتمی مانند DFS را اجرا میکنیم و اگر goal state را پیدا کردیم آن را برمیگردانیم و اگر پیدا نکردیم یکی به عمق اضافه میکنیم و به مرحله ی بعد میرویم.

الگوریتم بعدی الگوریتم A* میباشد که به شکلی میباشد که برای آن یک heuristic تعریف میکنیم و با توجه به آن قدم های بعدی مان را برای پیمایش انتخاب میکنیم که به طور کامل در قسمت بعد توضیح داده ایم.

و در اخر از Weighted A* استفاده میکنیم که این الگوریتم ممکن است بهترین جواب را به ما ندهد اما برنامه ی ما را سریع تر میکند. فرق آن با A* در این است که یک Alpha تعیین میکنیم و تعیین میکنیم که تا چه عمقی پایین برویم.

الگوریتم های BFS و IDS و A* جواب بهینه را به ما میدهند اما Weighted A* ممکن است جواب بهینه را به ما ندهد.

از لحاظ زمانی هم همه ی الگوریتم ها را در بخش پایین بررسی کرده ایم.

● توضیح heuristic پیاد هسازی شده در بخش جستجوی آگاهانه و consistent بودن یا نبودن آن

در بخش جستجوی آگاهانه از یک heuristic استفاده میکنیم که آن را به شکل زیر تعریف میکنیم:

heuristic = تعداد نود های حاوی مرید راضی نشده یا دستور پخت دیده نشده

توجه شود که ممکن است در یک نود چند مرید یا دستور پخت باشد اما در heuristic آن یکی حساب میشود.

و در نتیجه با توجه به معادله ی f(n) = g(n) + h(n) که g(n) هزینه ای میباشد که تا به اینجای کار برای مسیری که رفته ایم پرداخته ایم.(تعداد نود هایی که از آنها گذر کرده ایم)

پس evaluation function را به این شکل تعریف میکنیم.

از طرفی دیگر heuristic انتخاب شده consistent میباشد. میدانیم تعریف consistent بودن یک heuristic به شکل زیر است :

Cost(A to C) >= h(A) – h(C)

حال میدانیم که اگر بخواهیم از A به C برویم ممکن است اختلاف h هایمان صفر یا یک میتواند باشد. در صورتی که در نود C هیچ مرید یا دستور پختی نباشد اختلاف h هایمان صفر میشود و درغیر این صورت ممکن است یک شود. پس با توجه به اینکه هر استیتی که جابجا میشویم هزینه ی یک ثانیه را برای ما دارد پس رابطه ی بالا برقرار است و heuristic ما consistent میباشد.

● به ازای هر الگوریتم، هر تست کیس را ۳ بار اجرا کنید و میانگین زمان اجرا را ثبت کنید. همچنین جدول زیر را برای هر تست کامل کنید:

Weighted A* 1 -------> Alpha = 2

Weighted A* 2 -------> Alpha = 5

عدد ها بر حسب ms میباشد.

فایل input.txt

	پاسخ مسئله (حداقل زمان لازم برای رساندن دیزی‌ها)	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	1->3->4->5->7->10->11->9->8	80	6.981452306111653
IDS	1->3->4->5->7->10->11->9->8	506	9.649515151977537
A*	1->3->4->5->7->10->11->9->8	54	4.322369893391927
Weighted A* 1	1->3->4->5->7->10->11->9->8	41	2.578337987263993
Weighted A* 2	1->3->4->5->7->10->11->9->8	27	1.999934514363583

الگوریتم BFS :

Test 1 = 5.984306335449219 ms

Test 2 = 7.978200912475586 ms

Test 3 = 6.981849670410156 ms

الگوریتم IDS :

Test 1 = 10.97059249877929 ms

Test 2 = 8.995771408081055 ms

Test 3 = 8.982181549072266 ms

الگوریتم A* :

Test 1 = 3.989934921264648 ms

Test 2 = 4.987239837646484 ms

Test 3 = 3.989934921264648 ms

الگوریتم Weghted A* 1 :

Test 1 = 2.7704238891601562 ms

Test 2 = 1.9991397857666016 ms

Test 3 = 2.9654502868652344 ms

الگوریتم Weghted A* 2:

Test 1 = 1.9958019256591797 ms

Test 2 = 1.9943714141845703 ms

Test 3 = 2.0096302032470703 ms

فایل input2.txt

	پاسخ مسئله (حداقل زمان لازم برای رساندن دیزی‌ها)	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	9->10->9->4->12->3->7->5->8	256	46.8749205271403
IDS	9->10->9->4->12->3->7->5->8	1208	19.94689305623372
A*	9->10->9->4->12->3->7->5->8	132	14.35637474060056
Weighted A* 1	9->4->12->3->7->5->8->10->8->5->7	50	3.602743148803467
Weighted A* 2	9->4->2->10->2->11->3->7->5->8	34	2.326965332031223

الگوریتم BFS :

Test 1 = 46.8754768371582 ms

Test 2 = 45.87745666503906 ms

Test 3 = 47.87182807922363 ms

الگوریتم IDS :

Test 1 = 19.94609832763672 ms

Test 2 = 19.947052001953125 ms

Test 3 = 19.947528839111328 ms

الگوریتم A* :

Test 1 = 15.175819396972656 ms

Test 2 = 13.946056365966797 ms

Test 3 = 13.947248458862305 ms

الگوریتم Weghted A* 1 :

Test 1 = 3.1461715698242188 ms

Test 2 = 3.673076629638672 ms

Test 3 = 3.988981246948242 ms

الگوریتم Weghted A* 2:

Test 1 = 1.9946098327636719 ms

Test 2 = 2.9909610748291016 ms

Test 3 = 1.9953250885009766 ms

فایل input3.txt

	پاسخ مسئله (حداقل زمان لازم برای رساندن دیزی‌ها)	تعداد استیت‌های دیده شده	میانگین زمان اجرا
BFS	13->11->10->3->2->6->12->5->9->4->1->13->11->10	1656	2156.909545262647
IDS	13->11->10->3->2->6->12->5->9->4->1->13->11->10	21464	519.134918848673
A*	13->11->10->3->2->6->12->5->9->4->1->13->11->10	728	294.6446736653646
Weighted A* 1	13->11->10->3->2->6->12->5->9->4->1->13->11->10	47	2.992550532022803
Weighted A* 2	13->11->10->3->2->6->12->5->9->4->1->13->11->10	37	2.427101135253833

الگوریتم BFS :

Test 1 = 2175.184726715088 ms

Test 2 = 2141.2761211395264 ms

Test 3 = 2154.2677879333496 ms

الگوریتم IDS :

Test 1 = 492.68221855163574 ms

Test 2 = 509.20772552490234 ms

Test 3 = 555.5148124694824 ms

الگوریتم A* :

Test 1 = 284.5282554626465 ms

Test 2 = 291.2256717681885 ms

Test 3 = 308.1800937652588 ms

الگوریتم Weghted A* 1 :

Test 1 = 2.9926300048828125 ms

Test 2 = 2.9921531677246094 ms

Test 3 = 2.992868423461914 ms

الگوریتم Weghted A* 2:

Test 1 = 1.9936561584472656 ms

Test 2 = 2.2971630096435547 ms

Test 3 = 2.9904842376708984 ms