به نام خدا

حدیث غفوری

شماره دانشجویی: ۹۸۲۵۴۱۳

تدوین هدف:

پیداکردن دنباله ای از شهرها که کمترین مسافت را از شهر مبدا به شهر مقصد داشته باشند.

هر زمان که بخواهیم هر نوع مساله ای را به یک مساله جستجو تبدیل کنیم، باید شش چیز را تعریف کنیم:

۱. مجموعه ای از همه حالت هایی که ممکن است در نهایت در آنها قرار بگیریم

۲. حالت شروع و پایان

۳. یک بررسی پایان (روشی برای بررسی اینکه آیا در وضعیت تمام شده هستیم)

۴. مجموعه ای از اقدامات ممکن

۵.یک تابع پیمایش (عملکردی که به ما می‌گوید اگر در مسیر خاصی برویم به کجا می‌رسیم)

۶.مجموعه ای از هزینه های جابجایی از حالتی به حالت دیگر(که همان وزن یال ها در گراف است)

تدوین مساله:

حالات: بودن در شهرهای مختلف

اعمال: رانندگی بین شهرها

یافتن حل: دنباله ای از شهرها که کمترین مسافت را از مبدا به مقصد داشته باشند.

اجرای حل

توضیح درباره ی نحوه ی کارکرد الگوریتم *A:

*Aمبتنی بر استفاده از روش های اکتشافی برای دستیابی به جستجوای بهینه و کامل است و گونه ای از الگوریتم بهترین اول است.

هنگامی که یک الگوریتم جستجو دارای خاصیت بهینه است، به این معنی است که تضمین می شود بهترین راه حل ممکن را پیدا کند، در مورد این مساله کوتاه ترین مسیر تا حالت پایان.

هنگامی که یک الگوریتم جستجو دارای خاصیت کامل بودن است، به این معنی است که اگر یک راه حل برای یک مسئله معین وجود داشته باشد، الگوریتم تضمین می کند که آن را پیدا کند.

هر بار که *A وارد یک حالت می شود، هزینه f(n) (n گره همسایه است) را برای سفر به تمام گره های همسایه محاسبه می کند و سپس گره ای را با کمترین مقدار f(n) وارد می کند.

این مقادیر با فرمول زیر محاسبه می شوند:

$$f(n) = g(n) + h(n)$$

g(n) مسافت مسیر طی شده از گره شروع به گره n است و h(n) یک تقریب اکتشافی از مقدار گره است.

کارایی A*بسیار به مقدار اکتشافی h(n) وابسته است و بسته به نوع مسئله، ممکن است برای یافتن راه‌حل بهینه نیاز به استفاده از یک تابع اکتشافی متفاوت برای آن داشته باشیم.

در این پیاده سازی از A* مقدار اکتشافی h(n) را کوتاه ترین فاصله ی بین دو شهر درنظر گرفتیم که همان طول مسیر مستقیم بین دو شهر است.

در این برنامه ما سه فایل داریم:

۱.فایل ProvinceCenterDistances

مسافت واقعی هردو شهر در جدول را نشان میدهد که اعداد موجود در این فایل به عنوان وزن یال های گراف که تعیین کننده ی فاصله ی بین دوشهر هستند، استفاده می شوند.

۲.فایل ProvinceCentersNeighbours

نشان میدهد آیا دو شهر مجاور یکدیگر هستند یا خیر. اگر عدد موجود در جدول ۱ باشد به این معنی است که دو شهر موجود در سطر و ستون با یکدیگر مجاور هستند و می توان دو شهر را در لیست مجاورت گراف درنظر گرفت.
اگر عدد موجود در جدول صفر باشد به این معنا است که دو شهر با یکدیگر مجاورتی ندارند.

هدف اصلی استفاده از این فایل ساخت ماتریس مجاورت برای گراف است.

۳.فایل ProvinceCentersStraightLineDistances

اعداد این فایل به کمک google map بدست امدند که نشان دهنده ی کوتاه ترین فاصله ی دوشهر از یکدیگر است. به دلیل اینکه h(n) باید قابل پذیرش باشد یعنی هزینه ی رسیدن به هدف هیچ گاه بیش تخمین زده نشود پس باید مسافت خط مستقیم را برای مکاشفه ی h(n) انتخاب کنیم.

۱.open_list

لیستی از گره هایی است که بازدید شده است، اما همسایگان ان نودها هنوز به طور کامل بازرسی نشده اند، با گره اولیه(ریشه ی درخت) شروع می شود.

۲.closed_list

لیستی از گره هایی است که بازدید شده است و همسایگان ان نودها هم به طور کامل بازرسی شده است.

۳.دیکشنری g

شامل فواصل جاری از start_node تا تمام گره‌های دیگر است که مقدار پیش‌فرض (اگر در نقشه یافت نشد) + infinity است.

الگوریتم A*جستجوی یک گراف را برای ورودی ها شروع و مقصد انجام میدهد و در صورت وجود یک مسیر آن را به عنوان خروجی برمی گرداند.

برای انجام این کار از دو لیست به نام های *باز شده *و بسته استفاده می کند. لیست باز شده شامل گره هایی است که امکان انتخاب وجود دارد و بسته شامل گره هایی است که قبلا انتخاب شده اند. ابتدا، الگوریتم مقدار اکتشافی اولین گره را محاسبه می کند و آن گره را در لیست باز شده (مرحله اولیه سازی) اضافه می کند.

پس از آن، گره اولیه را از لیست باز شده حذف کنید و در لیست بسته قرار دهید. سپس، الگوریتم فرزندان گره انتخاب شده را گسترش داده و مقدار اکتشافی هر یک از آنها را محاسبه می کند. اگر فرزندی در هر دو لیست وجود نداشته باشد یا در لیست باز شده

باشد اما با مقدار اکتشافی بزرگتر باشد، فرزند مربوطه در لیست باز شده در موقعیت گره مربوطه با مقدار اکتشافی بالاتر اضافه می شود. در غیر این صورت حذف می شود.

در هر مرحله، گره با حداقل مقدار اکتشافی انتخاب شده و از لیست باز شده حذف می شود.

کل فرآیند زمانی خاتمه می یابد که راه حلی پیدا شود، یا لیست باز شده خالی باشد، به این معنی که راه حل ممکنی برای مشکل مرتبط وجود ندارد.

توضیح درباره ی نحوه ی کارکرد الگوریتم BFS

هنگام پیاده‌سازی BFS، معمولاً از ساختار FIFO مانند Queue برای ذخیره گره‌هایی استفاده می‌کنیم که بعداً بازدید می‌شوند.

برای استفاده از Queue در پایتون، باید کلاس Queue مربوطه را از ماژول صف وارد کنیم.

ما باید توجه داشته باشیم که با بازبینی مجدد و مکرر گره‌های یکسان در حلقه‌های بی‌نهایت قرار نگیریم، که به راحتی می‌تواند با نمودارهایی که دارای چرخه هستند اتفاق بیفتد. با در نظر گرفتن این موضوع، گره هایی که بازدید شده اند را پیگیری می کنیم. این اطلاعات لازم نیست به طور صریح ذخیره شوند، ما به سادگی می توانیم گره های والد را ردیابی کنیم تا پس از بازدید از آنها به یکی از آنها برنگردیم.

مراحل الگوریتم:

گره root/start را به Queue اضافه کنید.

برای هر گره، تنظیم کنید که گره والد تعریف شده نداشته باشد.

تا زمانی که صف خالی شود:

گره را از ابتدای صف استخراج کنید.

پردازش لازم را روی گره انجام دهید.

برای هر همسایه گره فعلی که والد تعریف شده ندارد (بازدید نمی شود)، آن را به صف اضافه کنید و گره فعلی را به عنوان والد خود تنظیم کنید.

مقایسه ی الگوریتم جستجوی عرض نخست و A*

عرض نخست از یک صف استفاده می کند در حالی که A*از صف اولویت استفاده می کند. به طور کلی، صف ها بسیار سریعتر از صف های اولویت هستند. مزیت A*این است که معمولاً گره های بسیار کمتری را نسبت به عرض نخست گسترش می دهد، اما اگر اینطور نباشد، BFS سریعتر خواهد بود. اگر اکتشافی مورد استفاده ضعیف باشد، یا اگر نمودار بسیار پراکنده یا کوچک باشد، یا اگر اکتشافی برای یک گراف معین شکست بخورد، ممکن است این اتفاق بیفتد.

نکته‌ی مهم این است که BFS فقط برای گراف های بدون وزن مفید است. اگر گراف وزن دارد، باید از الگوریتم Dijkstra استفاده کنیم. این الگوریتم از یک صف اولویت استفاده می کند و به این ترتیب تقریباً هرگز نباید سریعتر از A*باشد، مگر در مواردی که اکتشافی از کار بیفتد.

همان گونه که در خروجی برنامه مشاهده میشود، در پیمودن مسافت بین برخی از شهرها مثل مسیر همدان تا تهران، مسیر طی شده در الگوریتم A* همانند مسیر طی شده در الگوریتم عرض نخست است و مسافت کل هم یکسان است.
ولی در برخی از مسیرها مثل مسیر اراک به اردبیل، مسیرها متفاوت است و مسافت کلی که با الگوریتم A* پیموده شده کمتر از مسافتی است که با الگوریتم عرض نخست طی شده که به این دلیل است که زمانی که عرض نخست یک نود با عرض کمتر مشاهده میکند که در مسیر هدف است صرف نظر از وزن یال ها آن نود را انتخاب میکند و همین که مقایسه ای بین وزن یال‌ها انجام نمی‌شود باعث افزایش مسافت طی شده می‌شود.

جستجوی عرض نخست:

از روش جستجوی نااگاهانه است.

کامل است اگر ضریب انشعاب محدود باشد و بهینه است اگر هزینه ی هر مرحله برابر باشد پس اینجا که هزینه ها متفاوت است بهینه نیست.

پیچیدگی زمان و پیچیدگی فضا: چون در این پیاده سازی، گره بعد از ایجاد تست هدف میشود پس پیچیدگی زمانی و فضا هردو O(b^d) است.

جستجوی A*:

از روش جستجوی اگاهانه است. (هنگام بسط رئوس میتوان از دانشی که داریم استفاده کنیم و بررسی کنیم چه رئوسی بر رئوس دیگر برای بسط دادن برتری دارند)

برخلاف جستجوی عرض نخست از یک تابع ارزیابی f(n) استفاده میکنیم.

```
PS C:\Users\Win 10\Desktop\university\term7\AI\homeworks> python -u "c:\Users\Win 10\Desktop\university\term7\AI\homeworks\codes\a_star_function_approach.py"
Traveled Path with
1.a_star approach: ['Hamedan', 'Arak', 'Tehran']
2.bfs approach: ['Hamedan', 'Arak', 'Tehran']

total distance from Hamedan to Tehran is
1.a_star approach: 447
2.bfs approach: 447

Traveled Path with
1.a_star approach: ['Arak', 'Hamedan', 'Zanjan', 'Ardebil']
2.bfs approach: ['Arak', 'Ghazvin', 'Rasht', 'Ardebil']

total distance from Arak to Ardebil is
1.a_star approach: 687
2.bfs approach: 754

Traveled Path with
1.a_star approach: ['Arak', 'Isfahan', 'Shiraz', 'BandarAbbas']
2.bfs approach: ['Arak', 'Isfahan', 'Shiraz', 'BandarAbbas']

total distance from Arak to BandarAbbas is
1.a_star approach: 1338
2.bfs approach: 1338

Traveled Path with
1.a_star approach: ['Ghom', 'Tehran', 'Sari']
2.bfs approach: ['Ghom', 'Tehran', 'Sari']

total distance from Ghom to Sari is
1.a_star approach: 428
2.bfs approach: 428
```