پروژه نخست

حل مسئله با جستجو

مدرس: مسعود مظلوم

هوش مصنوعی

نیمسال اول ۱۴۰۲-۱۴۰۱

علوم کامپیوتر

دانشکده علوم ریاضی

Amin Amiri Torshizi – 9812399206

داکیومنتیشن

- **فاز اول :**

پیاده سازی تابع $Successor$ و توابع و کلاس هایی که برای استفاده در الگوریتم های جستجو نیاز است .

مدلسازی مسئله

هر محل محیط (میز) میتواند یک یا چند حالت داشته باشد . حالت ها به صورت زیر دسته بندی میشود

- در محل کره وجود دارد (b)
- محل جایگاهیست که کره درخواست داده شده است (p)
- در محل مانع وجود دارد (x)
- در محل ربات وجود دارد (r)
- محل خالی است

اشیا کره و ربات ، تنها اشیا متحرک محیط است و بقیه حالت ها (هدف ها ، موانع) ثابت هستند .

فرض کنیم محیط $n \times m$ باشد ، برای هدف ها و موانع دو ماتریس زیر را درنظر میگیریم

$$goals \coloneqq [g_{ij}]_{n\times m} \,;\, if\ g_{ij}\ not\ goal : (g_{ij} = None)\ else : \left(g_{ij} = Obj(i,j,"p")\right)$$

$$blockages \coloneqq [b_{ij}]_{n\times m} \,;\, if\ b_{ij}\ not\ block : (b_{ij} = None)\ else : \left(b_{ij} = Obj(i,j,"x")\right)$$

ربات هنگاه عبور از محل محیط با توجه به نوع محل باتری آن مصرف میشود ، باتری مصرف شده را هزینه عبور از هر محل در نظر میگیریم و برای هزینه ماتریس زیر را درنظر میگیریم (اگر در محل i,j مانع وجود داشت c_{ij} را بیشترین مقدار ممکن در نظر میگیریم)

$$costs \coloneqq [c_{ij}]_{n\times m}$$

جایگاه کره هایی که در هدف اند و کره هایی که در هدف نیستند و ربات را $State$ در نظر میگیریم و هر شی یا حالت را Obj در نظر میگیریم .

- **کلاس Obj**
- **ویژگی ها**
  - محل قرار گیری حالت یا شی (دو متغیر از نوع $integer$) که در چه سطر و ستونی قرار دارد
  - نوع حالت یا شی (x,r,b,p)
- **متدها**
  - بازنویسی متد های مقایسه ، جمع ، هش و کپی
  - متد $get_location$ که سطر و ستون را به خروجی میبرد

مقایسه دو شی در مکانی که قرار دارند فقط انجام میشود و شامل نوع شی نیست . جمع شی سطر و ستون آن ها باهم جمع میشود .

- **کلاس $State$**
- **ویژگی ها**
  - ربات (متغیری از نوع Obj)
  - لیستی از کره هایی که در هدف قرار ندارند (متغیر های لیست از نوع Obj)
  - لیستی از کره هایی که در هدف قرار دارند (متغیر های لیست از نوع Obj)
  - عمق $State$ (متغیر از نوع $integer$)
  - هزینه $State$ (متغیر از نوع $integer$)
- **متدها**
  - باز نویسی متد های مقایسه ، بزرگتر ، کوچکتر ، کپی و هش

مقایسه دو $State$ به صورت مقایسه جایگاه ربات در دو $State$ و مقایسه تعداد کره هایی که در مکان مشابه قرار دارند در دو $State$ است . مقایسه بزرگتری و کوچکتری بر اساس هزینه $State$ انجام میشود .

- تابع $Successor$
  - ورودی : $state$ و مجموعه ای از $state$ های دیده شده
  - خروجی : لیستی از $State$ های تولیده شده توسط $state$ ورودی

- نحوه کارکرد

یک لیست از $State$ ها در نظر میگیریم ، در هر $state$ ربات میتواند به 4 جهت حرکت کند (بالا ، پایین ، چپ ، راست) بردار های حرکت این 4 جهت را درنظر میگیریم

$$top := (-1,0), down := (1,0), left := (0,-1), right := (0,1)$$

و ابتدا با یک تابع بررسی میکنیم که ربات میتواند با این بردار حرکت کند یا خیر و اگر میتواند حرکت کند $true$ برگرداده شود و اگر حرکت ربات موجب شده کره ای حرکت کند ، کره را هم به خروجی ببرد .

$$check_location_free(state, move) \rightarrow butter\ , robot\ can\ move$$

اگر کره میتواند حرکت کند از نوع Obj است و در غیر این صورت $False$ است

اگر ربات میتواند حرکت کند $True$ و اگر نتواند حرکت کند $False$ است

حال اگر ربات توانست حرکت کند یک $State$ جدید با جابجایی ربات ساخته میشود (اگر کره هم جابجا شده بود آن را هم در نظر میگیری برای $state$ جدید) و اگر این $State$ در مجموعه $State$ وجود نداشت به لیست $state$ ها افزوده میشود و در پایان $state$ ورودی به مجموعه $state$ ها افزوده میشود و لیست $state$ ها به خروجی برده میشود .

- مرتبه زمانی

اگر تعداد کره ها را k در نظر بگیریم مرتبه زمانی تابع از $O(k^2)$ میباشد .

- **فاز دوم :**

پیاده سازی الگوریتم های

1. BFS (Breadth First Search)
2. DFS (Depth First Search)
3. IDS (Iterative Deepening Search)
4. UCS (Uniform Cost Search)

- پیاده سازی BFS

در نظر میگیریم $state$ اصلی (زمانی که هنوز ربات حرکتی نکرده) را $root$.
با استفاده از صف میخواهیم BFS را پیاده سازی کنیم ، فرض کنیم $Queue$ صف باشد . $root$ را به $Queue$ اضافه میکنیم و این الگوریتم را روی $Queue$ اجرا میکنیم .
$state$ را از صف خارج میکنیم بررسی میکنیم که تمام کره ها در سرجایشان است اگر نبود $State$ های گسترش یافته $Succsessor$ را به صف اضافه کرده و همین روند را تکرار میکنیم تا به هدف برسیم .

- پیاده سازی DFS

در نظر میگیریم $state$ اصلی (زمانی که هنوز ربات حرکتی نکرده) را $root$.
با استفاده از پشته میخواهیم DFS را پیاده سازی کنیم ، فرض کنیم $Stack$ پشته باشد . $root$ را به $Stack$ اضافه میکنیم و این الگوریتم را روی $Stack$ اجرا میکنیم .
$state$ را از پشته خارج میکنیم بررسی میکنیم که تمام کره ها در سرجایشان است اگر نبود $State$ های گسترش یافته $Succsessor$ را به پشته اضافه کرده و همین روند را تکرار میکنیم تا به هدف برسیم .

- پیاده سازی IDS

روند الگوریتم DFS را با شرط توقف در عمق k ادامه میدهیم و k را در هر مرتبه تکرار تغییر میدهیم .

- پیاده سازی UCS

در نظر میگیریم $state$ اصلی (زمانی که هنوز ربات حرکتی نکرده) را $root$.
با استفاده از مجموعه میخواهیم UCS را پیاده سازی کنیم ، فرض کنیم $stateSet$ مجموعه باشد . $root$ را به مجموعه اضافه میکنیم و این الگوریتم را روی $stateSet$ اجرا میکنیم .
در $setaetSet$ ، $state$ که دارای کمترین هزینه است را خارج میکنیم بررسی میکنیم که تمام کره ها در سرجایشان است اگر نبود $State$ های گسترش یافته $Succsessor$ را به مجموعه اضافه کرده و همین روند را تکرار میکنیم تا به هدف برسیم .