به نام خدا

منیره صفری ۹۴۳۱۰۴۴

گزارش پروژه ی اول هوش مصنوعی :

سوال ۱ :

نحوه ی مدل کردن مسئله :

برای این کار توابع مورد نیاز پیاده سازی شدند که شامل :

۱. **تابع مقدار دهی اولیه : مقدار start و goal را به روزرسانی میکرد.**
۲. **تابع action : مجموعه ی حرکت های موجود برای ربات r d u l بود که بر اساس جایگاه ربات برخی از این حرکت هارا نمیشد انجام داد در این تابع بر اساس استیت فعلی ربات حرکات قابل انجام مشخص می شدند . (به طور مثال اگر ربات در استیت (1,1) باشد دو حرکت رفتن به بالا و رفتن به چپ را نمیتواند انجام دهد.)**
۳. **تابع result : نتیجه ی حاصل از هر اکشن پیاده سازی شد. این تابع با گرفتن استیت مشخص میکند که با هر حرکت ممکن در این استیت چه استیت های جدیدی ایجاد می شود. مثلا با حرکت u مقدار x استیت یکی کم می شود .**
۴. **تابع cost : که طبق صورت سوال برای هر یال برابر یک است .**
۵. **تابع goal_test : در صورت یکی بودن استیت فعلی با استیت goal ، true بر میگرداند .**
۶. **تابع h : طبق صورت سوال برابر فاصله مستقیم دو نقطه قرار داده شد .**

همه ی الگوریتم های dfs (در ۳ مدل) ، bfs ، astar ، ucs و bidirectional پیاده سازی شدند .

در هر الگوریتم مقدار max memory نشان دهنده ی زمانی ست که تعداد نود ها ی ذخیره شده در f و e بیشترین مقدار خود را دارد.

در هر الگوریتم مقدار e نشان دهنده ی نود های گسترش یافته است.

در هر الگوریتم مقدار visited نشان دهنده ی نود های دیده شده است .

در هر الگوریتم مقدار path بهترین مسیر یافته شده را نشان می دهد .

این سوال توسط ۴ الگوریتم حل شد که در پایین خروجی آن ها خواهد امد :

ورودی :

```
3,3
1,3,2,3
2,1,3,1
2,2,3,2
```

خروجی :

uniform cost Search
path : [[1, 1], [1, 2], [2, 2], [2, 3], [3, 3]]
max memory 8
visited [[1, 2], [2, 1], [1, 3], [2, 2], [2, 2], [2, 3], [2, 3], [3, 3], [3, 3]]
expand [[1, 1], [1, 2], [2, 1], [1, 3], [2, 2], [2, 3]]
act ['r', 'd', 'r', 'd']

bidirectional search
path : [[1, 1], [2, 1], [2, 2], [2, 3], [3, 3]]
max memory 14
visited [[1, 1], [3, 3], [1, 2], [2, 3], [2, 1], [3, 2], [2, 2]]
expand [[1, 1], [1, 2], [2, 1], [1, 3], [2, 2], [2, 2], [3, 3], [2, 3], [3, 2], [3, 1]]
act ['d', 'r', 'r', 'd']

dfs unlimited
path : [[1, 1], [1, 2], [2, 2], [2, 3], [3, 3]]
max memory 5
visited [[1, 2], [1, 3], [2, 2], [2, 3]]
expand [[1, 3]]
act ['r', 'd', 'r', 'd']

astar
path : [[1, 1], [1, 2], [2, 2], [2, 3], [3, 3]]
max memory 10
visited [[1, 1], [1, 2], [2, 1], [1, 3], [2, 2], [2, 3], [3, 3]]
expand [[1, 1], [1, 2], [2, 1], [1, 3], [2, 2], [2, 2], [2, 3], [2, 3]]
act ['r', 'd', 'r', 'd']

طبق خروجی های دیده شده بیشتری مصرف حافظه برای جست و جوی دو طرفه است که دلیل اصلی آن این است که دو جست و جو را همزمان جلو می برد پس منطقی ست که بیشتری حافظه را داشته باشد .
کمترین حافظه مربوط به اول عمق است که می دانیم بین بقیه ی الگوریتم ها از حافظه ی کمتری استفاده میکند .
ولی برای جواب های طولانی تر جست و جوی دو طرفه و astar سرعت در حل کردن مسئله دارند ولی همانطور که میبینیم از حافظه ی زیادی استفاده می کنند .

سوال۲ :

نحوه ی مدل کردن مسئله :

برای این کار توابع مورد نیاز پیاده سازی شدند که شامل :

1. **تابع مقدار دهی اولیه : مقدار start و goal را به روزرسانی میکرد.**
2. **تابع action : مجموعه ی حرکت های موجود برای این پازل r d u l بود که بر اساس جایگاه جای خالی موجود در پازل حرکت های ممکن بررسی میشوند. (تقریبا مانند مثال قبل)**
3. **تابع result : نتیجه ی حاصل از هر اکشن پیاده سازی شد. این تابع با گرفتن استیت مشخص میکند که با هر حرکت ممکن در این استیت چه استیت های جدیدی ایجاد می شود. مثلا اگر جای خالی در ۵مین خانه باشد و خانه ی سمت چپ ان را به این جای خالی منتقل کنیم پازل در چه وضعیتی قرار خواهد گرفت.**
4. **تابع cost : که طبق صورت سوال برای هر یال برابر یک است .**
5. **تابع goal_test : در صورت یکی بودن استیت فعلی با استیت goal ، true بر میگرداند .**
6. **تابع h : فاصله ی مستقیم هر عدد در این استیت تا جای اصلی ای که باید قرار داشته باشند محاسبه شد و مقدار h برابر مجموع این فاصله ها قرار داده شد .**

مشکل اصلی این سوال این بود که چون تعداد حالات به وجود آمده برای حل برخی حالت ها خیلی زیاد میشد stack over flow رخ داده یا خیلی طول میکشید به خاطر همین برای بررسی از مثال های ساده ر استفاده شد .

این سوال توسط ۴ الگوریتم حل شد که در پایین خروجی آن ها خواهد امد :

ورودی :

```
1,2,3,4,5,6,7,0,8
```

خروجی :

uniform cost Search
path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
max memory 5
visited [[1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
act ['l']

bidirectional search
path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
max memory 7
visited [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
act ['l']

dfs unlimited
path : [[1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 5, 0, 7, 8, 6], [1, 2, 3, 4, 0, 5, 7, 8, 6], [1, 2, 3, 0, 4, 5, 7, 8, 6], [1, 2, 3, 7, 4, 5, 0, 8, 6], [1, 2, 3, 7, 4, 5, 8, 0, 6], [1, 2, 3, 7, 0, 5, 8, 4, 6], [1, 2, 3, 0, 7, 5, 8, 4, 6], [1, 2, 3, 8, 7, 5, 0, 4, 6], [1, 2, 3, 8, 7, 5, 4, 0, 6], [1, 2, 3, 8, 0, 5, 4, 7, 6], [1, 2, 3, 0, 8, 5, 4, 7, 6], [1, 2, 3, 4, 8, 5, 0, 7, 6], [1, 2, 3, 4, 8, 5, 7, 0, 6], [1, 2, 3, 4, 8, 5, 7, 6, 0], [1, 2, 3, 4, 8, 0, 7, 6, 5], [1, 2, 3, 4, 0, 8, 7, 6, 5], [1, 2, 3, 0, 4, 8, 7, 6, 5], [1, 2, 3, 7, 4, 8, 0, 6, 5], [1, 2, 3, 7, 4, 8, 6, 0, 5], [1, 2, 3, 7, 0, 8, 6, 4, 5], [1, 2, 3, 0, 7, 8, 6, 4, 5], [1, 2, 3, 6, 7, 8, 0, 4, 5], [1, 2, 3, 6, 7, 8, 4, 0, 5], [1, 2, 3, 6, 0, 8, 4, 7, 5], [1, 2, 3, 0, 6, 8, 4, 7, 5], [1, 2, 3, 4, 6, 8, 0, 7, 5], [1, 2, 3, 4, 6, 8, 7, 0, 5], [1, 2, 3, 4, 6, 8, 7, 5, 0], [1, 2, 3, 4, 6, 0, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 7, 4, 6, 0, 5, 8], [1, 2, 3, 7, 4, 6, 5, 0, 8], [1, 2, 3, 7, 0, 6, 5, 4, 8], [1, 2, 3, 0, 7, 6, 5, 4, 8], [1, 2, 3, 5, 7, 6, 0, 4, 8], [1, 2, 3, 5, 7, 6, 4, 0, 8], [1, 2, 3, 5, 0, 6, 4, 7, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8]]
max memory 41
visited [[1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8], [1, 2, 3, 5, 0, 6, 4, 7, 8], [1, 2, 3, 5, 7, 6, 4, 0, 8], [1, 2, 3, 5, 7, 6, 0, 4, 8], [1, 2, 3, 0, 7, 6, 5, 4, 8], [1, 2, 3, 7, 0, 6, 5, 4, 8], [1, 2, 3, 7, 4, 6, 5, 0, 8], [1, 2, 3, 7, 4, 6, 0, 5, 8], [1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 6, 0, 7, 5, 8], [1, 2, 3, 4, 6, 8, 7, 5, 0], [1, 2, 3, 4, 6, 8, 7, 0, 5], [1, 2, 3, 4, 6, 8, 0, 7, 5], [1, 2, 3, 0, 6, 8, 4, 7, 5], [1, 2, 3, 6, 0, 8, 4, 7, 5], [1, 2, 3, 6, 7, 8, 4, 0, 5], [1, 2, 3, 6, 7, 8, 0, 4, 5], [1, 2, 3, 0, 7, 8, 6, 4, 5], [1, 2, 3, 7, 0, 8, 6, 4, 5], [1, 2, 3, 7, 4, 8, 6, 0, 5], [1,

2, 3, 7, 4, 8, 0, 6, 5], [1, 2, 3, 0, 4, 8, 7, 6, 5], [1, 2, 3, 4, 0, 8, 7, 6, 5], [1, 2, 3, 4, 8, 0, 7, 6, 5], [1, 2, 3, 4, 8, 5, 7, 6, 0], [1, 2, 3, 4, 8, 5, 7, 0, 6], [1, 2, 3, 4, 8, 5, 0, 7, 6], [1, 2, 3, 0, 8, 5, 4, 7, 6], [1, 2, 3, 8, 0, 5, 4, 7, 6], [1, 2, 3, 8, 7, 5, 4, 0, 6], [1, 2, 3, 8, 7, 5, 0, 4, 6], [1, 2, 3, 0, 7, 5, 8, 4, 6], [1, 2, 3, 7, 0, 5, 8, 4, 6], [1, 2, 3, 7, 4, 5, 8, 0, 6], [1, 2, 3, 7, 4, 5, 0, 8, 6], [1, 2, 3, 0, 4, 5, 7, 8, 6], [1, 2, 3, 4, 0, 5, 7, 8, 6], [1, 2, 3, 4, 5, 0, 7, 8, 6]]
act ['l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'u', 'l', 'l', 'r', 'l', 'r', 'u']

astar
path : [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
max memory 5
visited [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 0, 5, 6, 4, 7, 8]]
expand [[1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 0, 7, 8]]
act ['l']

در این مثال میبینیم که بدترین عملکرد مربوط به dfs است که به خاطر نوع عملکرد الگوریتم است که ابتدا عمق را بررسی می کند.همچنین بیشترین حافظه نیز برای dfs است پس این الگوریتم برای حل این سوال مناسب نمی باشد.
کمترین حافظه های مصرفی برای astar و ucs است و از نظر زمان و نود های دیده شده ۳ الگوریتم دیگر (به جز dfs) تقریبا در سطح یکسانی قرار دارند .

سوال ۳ :

نحوه ی مدل کردن مسئله :

برای این کار توابع مورد نیاز پیاده سازی شدند که شامل :

۱. **تابع مقدار دهی اولیه : مقدار start و goal را به روزرسانی میکرد.**
۲. **تابع action : در این سوال برای حالت های مختلف action ها یکسان بود و محدودیتی در action ها وجود نداشت پس هر۶ حرکت گفته شده در صورت سوال در هر استیتی قابل انجام بود.**
۳. **تابع result : نتیجه ی حاصل از هر اکشن پیاده سازی شد. این تابع با گرفتن استیت مشخص میکند که با هر حرکت ممکن در این استیت به چه استیت جدید می رویم. برای این کار به ازای هر حرکت در هر استیت تمامی تغییرات در رنگ خانه های روبیک اعمال شد.**
۴. **تابع cost : برای این سوال نیازی نداشتیم ولی میتوانستیم مانند سوال های قبل هزینه ی هر یال را برابر ۱ بگیریم.**
۵. **تابع goal_test : در صورت یکی بودن استیت فعلی با استیت goal ، true بر میگرداند .**
۶. **تابع h : در این سوال نیازی به آن نداشتیم .**

این سوال توسط ۳ الگوریتم حل شد که در پایین خروجی آن ها خواهد امد :

ورودی :

y b y b g y g y w g w g b w b w r r r r o o o o

خروجی :

bfs
path : [['y', 'b', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o']]
max memory 29
visited [['y', 'y', 'b', 'b', 'o', 'o', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'r', 'r', 'g', 'y', 'r', 'r', 'w', 'b', 'o', 'o'], ['b', 'b', 'y', 'y', 'r', 'r', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'o', 'o', 'b', 'w', 'r', 'r', 'g', 'y', 'o', 'o'], ['y', 'y', 'r', 'r', 'g', 'g', 'y', 'y', 'o', 'o', 'w', 'g', 'b', 'w', 'r', 'r', 'g', 'w', 'r', 'g', 'y', 'b', 'b', 'o'], ['y', 'y', 'o', 'o', 'y', 'y', 'g', 'g', 'r', 'r', 'w', 'g', 'b', 'w', 'r', 'r', 'g', 'b', 'r', 'y', 'g', 'o', 'w', 'o'], ['y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['y', 'w', 'y', 'w', 'g', 'b', 'g', 'b', 'w', 'y', 'w', 'y', 'b', 'g', 'b', 'g', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['b', 'y', 'b', 'y', 'w', 'b', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'g', 'y', 'o', 'o', 'r', 'r', 'r', 'r', 'o', 'o'], ['b', 'y', 'r', 'y', 'g', 'o', 'y', 'o', 'o', 'w', 'w', 'g', 'b', 'w', 'g', 'y', 'o', 'w', 'r', 'g', 'b', 'r', 'b', 'o'], ['b', 'y', 'w', 'o', 'o', 'y', 'o', 'g', 'y', 'r', 'w', 'g', 'b', 'w', 'g', 'y', 'o', 'b', 'r', 'b', 'g', 'b', 'w', 'o'], ['y', 'o', 'b', 'y', 'o', 'g', 'g', 'g', 'w', 'w', 'w', 'r', 'b', 'y', 'r', 'b', 'g', 'y', 'r', 'r', 'b', 'o', 'w', 'o'], ['y', 'w', 'b', 'r', 'o', 'y', 'g', 'b', 'w', 'o', 'w', 'y', 'b', 'g', 'r', 'g', 'g', 'y', 'r', 'r', 'o', 'w', 'o', 'b'], ['b', 'y', 'b', 'y', 'b', 'w', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'y', 'g', 'o', 'o', 'r', 'r', 'r', 'r', 'o', 'o'], ['y', 'b', 'r', 'w', 'g', 'r', 'y', 'r', 'o', 'g', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'w', 'r', 'g', 'y', 'o', 'y', 'o'], ['y', 'b', 'g', 'o', 'r', 'y', 'r', 'g', 'w', 'r', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'y', 'r', 'y', 'g', 'y', 'w', 'o'], ['b', 'r', 'y', 'y', 'r', 'g', 'g', 'g', 'w', 'w', 'w', 'o', 'b', 'b', 'o', 'y', 'b', 'w', 'r', 'r', 'y', 'o', 'g', 'o'], ['b', 'w', 'y', 'o', 'r', 'b', 'g', 'y', 'w', 'r', 'w', 'y', 'b', 'g', 'o', 'g', 'b', 'w', 'r', 'r', 'o', 'g', 'o', 'y'], ['r', 'y', 'r', 'y', 'y', 'b', 'y', 'y', 'o', 'o', 'w', 'g', 'b', 'w', 'g', 'w', 'g', 'g', 'r', 'g', 'r', 'r', 'b', 'o'], ['y', 'r', 'y', 'r', 'g', 'w', 'y', 'y', 'o', 'o', 'w', 'g', 'b', 'w', 'b', 'y', 'r', 'r', 'r', 'g', 'g', 'g', 'b', 'o'], ['r', 'y', 'g', 'w', 'y', 'g', 'y', 'g', 'b', 'y', 'w', 'g', 'b', 'w', 'g', 'w', 'g', 'o', 'r', 'o', 'r', 'r', 'r', 'o'], ['r', 'y', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'g', 'w', 'g', 'r', 'r', 'r', 'o', 'b', 'o', 'o'], ['y', 'g', 'r', 'y', 'g', 'o', 'y', 'g', 'o', 'w', 'w', 'r', 'b', 'y', 'r', 'r', 'g', 'w', 'r', 'g', 'b', 'o', 'y', 'b'], ['y', 'w', 'r', 'r', 'g', 'y', 'y', 'r', 'o', 'g', 'w', 'y', 'b', 'o', 'r', 'g', 'g', 'w', 'r', 'g', 'b', 'y', 'o', 'b'], ['o', 'y', 'o', 'y', 'g', 'o', 'g', 'g', 'r', 'r', 'w', 'g', 'b', 'w', 'g', 'b', 'y', 'y', 'r', 'y', 'r', 'r', 'w', 'o'], ['y', 'o', 'y', 'o', 'g', 'b', 'g', 'g', 'r', 'r', 'w', 'g', 'b', 'w', 'o', 'g', 'r', 'r', 'r', 'y', 'y', 'y', 'w', 'o'], ['o', 'y', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'g', 'b', 'y', 'r', 'r', 'r', 'o', 'r', 'o', 'o'], ['o', 'y', 'g', 'w', 'y', 'g', 'y', 'g', 'b', 'y', 'w', 'g', 'b', 'w', 'g', 'b', 'y', 'o', 'r', 'o', 'r', 'o', 'r', 'o'], ['y', 'y', 'o', 'g', 'y', 'r', 'g', 'g', 'r', 'w', 'w', 'r', 'b', 'y', 'r', 'o', 'g', 'b', 'r', 'y', 'o', 'o', 'g', 'w'], ['y', 'w', 'o', 'r', 'y', 'y', 'g', 'o', 'r', 'y', 'w', 'g', 'b', 'r', 'r', 'g', 'g', 'b', 'r', 'y', 'w', 'g', 'o', 'o']]

act ['R']

limit 1
finish
dfs limited
path : [['y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['y', 'b', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o']]
max memory 1
act ['R']

iterative dfs limited
path : [['y', 'y', 'y', 'y', 'g', 'g', 'g', 'g', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o'], ['y', 'b', 'y', 'b', 'g', 'y', 'g', 'y', 'w', 'g', 'w', 'g', 'b', 'w', 'b', 'w', 'r', 'r', 'r', 'r', 'o', 'o', 'o', 'o']]
max memory 1
act ['R']

با توجه به خروجی ها dfs جواب بهتری دارد ولی این نکته وجود دارد که در این سوال جواب نهاییR است و این که ایا DFS به جواب می رسد یا نه به این بستگی دارد که R چندمین اکشن باشد در واقع اگر اولین فرزندی که بررسی میشود فرزند مربوط به اکشن R باشد آنگاه DFS جواب میدهد اما اگر قبل آن فرزند حاصل از حرکت دیگری را چک کند تا عمق ۱۴ هم جواب نهایی پیدا نخواهد شد و در بعضی موارد stack over flow نیز رخ میدهد که به دلیل حجم بالای پردازش dfs است .
به طور کلی برای این سوال همان bfs جواب بهتری پیدا میکند تا dfs .