گزارش پروژه ی میانترم

در این پروژه قصد داریم مسیله ی 8puzzle را به کمک الگوریتم های مختلف سرچ حل کنیم.

برای این منظور و به دلیل مشترک بودن مفاهیم اولیه الگوریتم ها دو کلاس Node و Tree تعریف کردیم که در ادامه توضیح میدهیم.

کلاس Node:

این کلاس مشخص کننده ی ویژگی های هر نود می باشد که شامل وضعیت پازل و نود واد و نود فرزندان به وصورت لیست می باشد. در این کلاس دو متغیر col و size می باشد که نشان دهنده سایز پازل می باشد. در هر پازل باید به موقعیت خانه ی خالی که ان را با ۰ در ارایه نشان میدهیم دسترسی داشته باشیم که اندیس آن را با ind0 تعیین کردیم. یکی از ویژگی های هر نود میتوان به اکشنی که باعث می شود از نود والد به وضعیت کنونی برسد.

وضیت هر پازل به صورت یک آرایه ی یک بعدی 1*size به جای ارایه ی دو بعدی تعریف شده است و برای انجام حرکت های مختلف ابتدا اندیس آن را چک میکنیم که آیا حرکت مجاز می باشد و سپس حرکت را انجام داده نود فرزند را به همراه اکشن و والد آن می سازیم و آن را به لیست فرزندان نود کنونی اضافه میکنیم. برای این منظور ۴ تابع به صورت جداگانه right,left,up,down تعریف شده که هنگام تولید فرزندان ابتدا اندیس خانه ی صفر را می یابیم و سپس حر کت ها را انجام می دهیم.

این کلاس دارای یک تابع تست می باشد که وضعیت پازل خروجی را دریافت میکند که به صورت دیفالت {۰و۸و۷و۶و۵و۴و۳و۲و۱} می باشد را با مقدار پازل مقایسه میکند.

یک تابع findheu برای تابع *A تعریف کردیم که تعداد خانه هایی که نسبت به پازل هدف در جای درستی قرار ندارند را نشان می دهد.

کلاس tree:

هر جست و جوی درختی نیاز به تشکیل دادن درختی دارد که مجموعه هایی نظیر frontier,explored را شامل می شود. در این درخت برای اینکه ایجاد لوپ نشود نیاز به مجموعه ی explored داریم که در آن همه ی نود های بسط داده شده را قرار میدهیم و هنگام انتخاب یک نود برای بسط و اضافه کردن به frontier چک میکنیم که این نود در مجموعه ی explored و frontier حضور نداشته باشد.

برای هر درخت یک یک لیستی از مسیر ها و اکشن ها در نظر میگیریم که هنگام یافتن جواب با تابع pathtrace از هدف به سمت ریشه ی درخت حرکت کرده و یک راه حل برای مسپله برمیگردانیم.

برای نمایش نود ها از رنگ ها استفاده کردیم که برای هر رنگ باید از الگوی \033[39m استفاده کنیم که قسمت اول مربوط کد کلید esc می باشد و ۳۹ کد رنگ مد و برای پایان از m استفده میکنیم.

قبل از شروع هر پازل ابتدا چک میکنیم که ایا پازل با شروع از حالت اولیه قابل حل می باشد یا خیر که این با شمارش زوج های invert حاصل میشود(زوجی که عدد کمتر در اندیس بالاتر قرار دارد) و اگر تعداد این زج ها عدد زوجی باشد مسپله قابل حل می باشد.

الگوریتم BFS:

این الگوریتم به صورت اول سطح عمل میکند که ابتدا نود را چک میکند ایا هدف می باشد یا خیر. در صورتی که هدف بود مسیر را بازمیگرداند و در غیر اینصورت الگوریتم را ادامه میدهد. در قدم بعدی باید فرزندان آن نود را تولید کند و آن ها را با هدف مقایسه میکنیم و در صورتی که هدف نباشند آن هایی که غیر تکراری هستند را به مجموعه ی frontier اضافه میکنیم.

این الگوریتم تا جایی ادامه پیدا میکند که یا مجموعه ی frontier عضوی نداشته باشد یا به نود هدف برسیم.

الگوریتم dls:

الگوریتم dls الگوریتم جست و جوی عمقی DFS در عمق محدود می باشد. الگوریتم dfs به صورت اول عمق عمل میکند که ابتدا یک نود را تا انتها بسط میدهید و سپس نود های بعدی را بسط میدهد. مشکل این الگوریتم جست و جو ممکن است تا سطح زیادی ادامه پیدا کند. برای حل این مشکل عمق محدود میکنیم و الگوریتم جست و جوی عمقی را تا آن عمق ادامه میدهیم.

الگوریتم A*:

این الگوریتم بر اساس تابع هیوریستیک که تعریف کردیم به هر نود یک مقدار اختصاص می دهد و در هر مرحله نودی را انتخاب میکند که هیوریستیک کمتری داشته باشد. این روش نسبت به دو روش قبلی سریعتر می باشد و تعداد نود کمتری را برای رسیدن به هدف بسط می دهد.

لینک repository :

https://github.com/z-arabi/8puzzle.git

* کد من هنوز کامل نیست و قسمت منو هم باید اضافه بشه. برای ساعت ۴ یک پاور پوینت اماده میکنم ولی بعد از کامل شدن هم گزارش و هم پاورپوینت و هم ویدیو رو با هم میفرستم.

متشکرم از شما