گزارش پروژه میان ترم برنامه نویسی پیشرفته

حل پازل (3*3)

در این پروژه سعی شده است تا پازل های 3*3 را با استفاده از الگوریتم BFS که در ادامه توضیح داده شده است حل کنیم.

الگوریتم BFS (Breath First Search):

این الگوریتم که از نوع BST میباشد یک طریقه ی جستجو به ترتیب است که به میتواند هدف مورد نظر را با کمترین عمق پیدا کند البته که سرعت این الگوریتم به دلیل بررسی همه ی شاخه بدون هیچ شرطی نسبت به سایر الگوریتم ها پایین تر است . چون در حل اینگونه پازل ها شرطی نمیتوان برای شاخه ها اعمال کرد پس میتواند یکی از روش های پیشنهادی باشد

این الگوریتم به این صورت عمل میکند تمام Node های موجود در یک سطر را بررسی کرده و سپس به سطر بعدی میرود به همین صورت آنقدر جلو میرود تا به Node مورد نظرش برسد

روش حل و تعریف توابع:

برای حل این پازل ها ابتدا یک کلاس از نوع Node تعریف کردیم که هر Node نماینده از قرارگرفتن اعداد 1 تا 8 در یک جدول 3*3 است مانند شکل زیر:

2	4	1
3		7
5	6	8

در ابتدا باید متغیرها و توابع کلاس Node را بیان کنیم :

متغیر ها :

@ id : این متغیر یک عدد اختصاصی برای هر Node است که توسط آن شناخته میشود به این ترتیب که برای هر Node به ترتیب اعداد هر ردیف را به ترتیب از سمت چپ به راست اضافه میشود و در نهایت یک عدد 9 رقمی در آن ذخیره میشود به ازای خانه خالی عدد 0 قرار میگیرد به طور مثال برای Node قبلی عدد 241307568 به عنوان id ذخیره میشود

@ zero_row و zero_clumn : این دو متغیر محل قرارگیری خانه خالی Node را ذخیره میکند هر Node 3 ستون و 3 ردیف دارد که از 0 تا 2 عددگذاری میشوند به طور مثال در Node بالا خانه ی (1,1) خالی است عدد اول در zero_row و دومی در zero_column ذخیره میشود

@ members : این متغیر که از نوع یک vector دو بعدی است که اعداد هر Node در خانه های مورد نظر ذخیره میکند

@ childs و parent : این دو متغیر هر کدام به نوعی اطلاعات Node یا Node هایی در آنها ذخیره میشود هر Node تعدادی child دارد که این child ها از نوع Node هستند به طوری که یکی از خانه های اطراف خانه خالی با خانه خالی عوض شده است به طور مثال Node زیر یک child برای Node ای که مثال زده شد میباشد :

2	4	1
3	7	
5	6	8

همانطور که مشاهده میشود خانه خالی با خانه ی شماره 7 عوض شده است

متغیر parent برای هر Node یک Node دیگر است که Node به عنوان child آن Node تعریف شده است

باید به این توجه داشت که ما باید از تکرار های بیهوده پرهیز کنیم بنابراین زمانی که child در حال شناسایی است parent نیز جز child های یک Node بحساب می آید که آن را محاسبه نمیکنیم

توابع :

@ make_child : این تابع برای هر Node ، child هایش را شناسایی میکند و در متغیر childs ذخیره میکند

@ random : این تابع یک Node با اعداد 0_8 با جایگشت های تصادفی تولید میکند

@ show : این تابع برای نمایش یک Node به صورت یک ماتریس 3*3 بکار میرود

@ make_id : این تابع با توجه به اعضای یک Node که از متغیر members آن میگیرد یک شناسه اختصاصی برای Node میسازد و در id ذخیره میکند

@ solvable : این تابع قابل حل بودن یا نبودن یک Node را مشخص میکند این تابع بر اساس تعداد نابجایی ها که یک عدد کوچکتر از عدد در خانه ای بعد از آن قرار گرفته است انجام میشود که اگر تعداد آنها فرد باشد پازل غیرقابل حل میباشد

حل مساله:

در این برنامه از یک تابع اصلی به نام **solve_puzzle** استفاده شده است که به عنوان ورودی یک Node و goal_id دریافت میکند و وضعیت یک Node را برای رسیدن به Node هدف بررسی میکند به طور پیشفرض gola_id برابر 123456780 میباشد که همان Node هدف است

این تابع با استفاده از الگوریتم BFS به دنبال Node هدف میگردد به این صورت که در ابتدا برای Node دریافتی خود child میسازد و آن ها را با مقایسه میکند اگر مقایسه با Node هدف درست نباشد برای هر کدام از child ها child میسازد و آن ها را بررسی میکند این بررسی تا

جایی ادامه پیدا میکند که Node هدف پیدا شود در این صورت به صورت برگشت پذیر مجموعه ای از Node ها که به ترتیب باعث رسیدن به Node هدف شده اند را برمیگرداند

نکته مهم اینجاست که ما بازهم باید از تکرار های بیهوده جلوگیری کنیم برای همین در زمان شناسایی child های یک Node به تمامی Node های ساخته شده تا آن لحظه نگاه میکنیم و در صورت عدم وجود آن Node اضافه میشود

در صورتی که برنامه به جایی برسد که تعداد child های تولید شده برای یک ردیف صفر شده باشد متوجه میشود که این پازل قابل حل نیست و یک آرایه خالی برمیگرداند

در تابع **Solve_Random** ابتدا یک Node ساخته میشود و تابع random آن صدا زده میشود اگر قابل حل نباشد اعلام میکند در غیر این صورت با استفاده از تابع Solve_puzzle روش حل را بررسی میکند و مراحل حل پازل را به صورت گام به گام با تاخیر 0.5 ثانیه نمایش میدهد و در غیر این صورت پیغامی مبنی بر غیر قابل حل بودن puzzle اعلام میکند

در تابع **Solve_your_puzzle** در ابتدا از کاربر شناسه Node مدنظر خواسته میشود سپس Node را میسازد و مانند تابع Solve_random عمل میکند

در توابع **solve_random_with_special_node** و **solve_your_puzzle_with_special_goal** همانند توابع قبلی عمل میشود با این تفاوت که کاربر میتواند Node هدف خود را مشخص نماید

در تابع **solve_by_your_self** در ابتدا به کاربر یک پازل رندوم داده میشود و از او خواسته میشود با توجه به راهنمایی که دارد پازل را حل نماید در صورت موفقیت پازل حل میشود و در غیر این صورت کاربر میتواند از برنامه بخواهد تا ادامه مسیر را خودش ادامه دهد