گزارش پروژه میان ترم برنامه نویسی پیشرفته

حل پازل (3*3)

در این پروژه سعی شده است تا با استفاده از دو الگوریتم BFS و DLS اینگونه پازل ها را حل کنیم

این پروژه از دو قسمت تشکیل شده است که یک قسمت مربوط به پازل ها و دستورات مربوط آنهاست که به نام node نام گذاری شده است و قسمت دیگر که کربوط به ساختار برنامه میباشد و قسمت اصلی را تشکیل میدهد با نام main نامگذاری شده است

در ابتدا با ساختار node و دستورات مربوط به آنها آشنا میشویم

کلاس Node :

2	4	1
3		7
5	6	8

هر node نماینده یک حالت از قرارگیری اعداد در یک پازل 3*3 میباشد که نمونه ای از آنها را در بالا مشاهده میکنیم حال باید متغیر ها و توابع این کلاس را بشناسیم

متغیرها :

Members :

این متغیر یک آرایه دوبعدی 3*3 از اعداد صحیح میباشد که اعضای node را به ترتیب ذخیره میکند

Id :

این متغیر یک عدد 9 رقمی است که برای هر node بکار میرود و که چیدمان آن به این صورت که که عددی که در خانه ی (1,1) است دارای بیشترین ارزش مکانی و عددی که در خانه ی (3,3) قرار دارد در یکان این عدد قرار میگیرد این متغیر بیشتر برای ساده سازی مقایسه ها و جلوگیری از ذخیره سازی آرایه ها بکار میرود

Zero_row و Zero_column :

این دو متغیر محل قرارگیری خانه خالی را مشخص میکنند که به ترتیب در چه ردیف و چه ستونی واقع شده است

Count :

این برای شمارش بکار میرود و یک بررسی را انجام میدهد که در بخش DLS به کار میرود که در آن قسمت بیشتر توضیح داده شده است

قبل از بررسی سایر متغیر ها ابتدا باید مفهوم child و parent را مشخص کنیم

Child به node ای گفته میشود که تنها یک خانه با node اصلی متفاوت است هر node میتواند تعدادی Child داشته باشد به طور مثال node زیر یک child برای node ای است که در ابتدا به آن اشاره کردیم که مشاهده میشود عدد 7 تغییر کرده و به محل خالی آمده است

2	4	1
3	7	
5	6	8

همانطور که مشاهده شد مفهوم child واضح است حال باید بدانیم برای هر child ساخته شده node ای که از آن ساخته شده اند parent آن node میباشد

Parent :

این متغیر یک پوینتر است که به parent آن node اشاره میکند

Childs :

این متغیر نیز یک آرایه است که child های یک node در آن ذخیره میشود

** از تعریف child و parent به این نتیجه میرسیم که parent یک node میتواند child آن نیز باشد بنابراین برای جلوگیری از تکرار های بیهوده child ی که شبیه به parent آن node باشد از child های node به حساب نمی آید

توابع :

تابع random :

این تابع اعداد 0 تا 8 را به صورت تصادفی در داخل node قرار میدهد توجه شود که عدد 0 نماینده خانه ی خالی در پازل میباشد

تابع show :

این تابع شکل ظاهری node به صورت یک ماتریس 3*3 را نمایش میدهد

تابع make_childs :

این تابع با استغلده از تعاریفی که قبلا داشتیم child های یک node را میسازد و در داخل متغیر childs ذخیره میکند

تابع make_id :

این تابع نیز مقدار متغیر id را برای node میسازد

تابع solvable :

این تابع نیز با بررسی که انجام میدهد مشخص میکند که پازل قابل حل است یا نه

بعد از تعریف کلاس node باید قسمت اصلی برنامه را مشخص کنیم این قسمت از تعدادی تابع تشکیل شده است که دارای 3 تابع اصلی میباشد که روش حل پازل را بررسی میکند که هر کدام را به ترتیب در پایین به همراه الگوریتم های آنها توضیح میدهیم

توابع روش حل پازل :

تابع solve_puzzle_bfs :

این تابع برای حل پازل از الگوریتم BFS استفاده میکند که یک node و یک goal_id برای حل پازل دریافت میکند و پازل را تا جایی که به goal_id برسد حل مینماید الگوریتم به این صورت است که برای یک node تا جایی که بتواند child میسازد و به ترتیب child ها را بررسی میکند اگر به هدف نرسیده باشد برای هر کدام از آنها نیز child میسازد و در ردیف

بعدی قرار میدهد سپس به ردیف بعدی رفته و child هایی که در این ردیف قرار دارند را بررسی مینماید اینقدر این کار تکرار میشود تا به node هدف برسد

** برای جلوگیری از تکرار های بیهوده هر child ی که ساخته میشود اگر قبلا ساخته شده باشد به ردیف بعدی اضافه نمیشود

پس از پیدا کردن node هدف آرایه ای از node ها را به این گونه که parent ها به ترتیب در آن قرار دارند را برمیگرداند

تابع solve_puzzle_dls :

این تابع برای حل از الگوریتم DLS استفاده میکند . این تابع علاوه بر ورودی های تابع قبل یک متغیر نیز به عنوان limit دریافت میکند این تابع ابتدا به این گونه عمل میکند که به node اشاره میکند و شرط درست بودن شرط را بررسی میکند اگر درست نباشد از آن child میسازد و به child ای اشاره میکند که count شماره آن را مشخص میکند و سپس این مسیر تکرار میشود با هر بار رفتن به جلو متغیر depth اضافه میشود اگر به جایی برسیم که depth برابر limit شود دیگر اجازه پیشروی نداریم و باید برگردیم در این صورت متغیر count را در parent یکی اضافه میکنیم و سپس به parent اشاره میکنیم در صورتی که متغیر count نیز از تعداد child های node بیشتر باشد نیز به parent آن اشاره میکنیم این کار تا زمانی پیش میرود که به goal_id برسیم در این صورت همانند تابع قبلی آرایه ای از parent ها را برمیگردانیم

تابع bidirectional :

این تابع که از نام آن مشخص است سعی میکند که به طور همزمان پازل را به دو روش بالا حل نماید و هرکدام که زودتر جواب آن را به عنوان پاسخ نمایش دهد این تابع ورودی هایی مشابه تابع solve_puzzle_bfs دارد. در این تابع با استفاده از برنامه نویسی threading دو تابع به طور موازی به حل میپردازند و در صورت رسیدن یکی از آنها به جواب تابع دیگر با توجه به متغیر global ی به نام complete متوقف شده و آرایه ای که ساخته شده برمیگردد

توابع برنامه :

تابع solve_random_puzzle :

این تابع یک پازل تصادفی ایجاد میکند سپس بررسی میکند که آیا قابل حل است یا خیر . در صورت قابل حل بودن روش حل را از کاربر سول میکند و با دریافت دستور مناسب آن را جل مینماید و مراحل حل پازل را با تاخیرهایی که کاربر بتواند به خوبی آن ها را ببیند نمایش میدهد

تابع solve_your_puzzle :

این تابع ابتدا از کاربر یک id دریافت میکند و با استفاده از آن یک پازل میسازد سپس قابل حل بودن آن را بررسی مینماید در صورت حل پذیری مانند تابع قبلی عمل میکند

تابع solve_puzzle_by_your_self :

این تابع به کاربر یک پازل تصادفی نمایش میدهد و با راهنمایی که به آن میدهد از او میخواهد تا پازل را خودش حل نماید در صورت اینکه کاربر خسته شود میتواند بقیه کار تکمیل پازل را به برنامه بسپارد

تابع solve_random_puzzle_with_special_goal :

این تابع نیز از کاربر یک حالت دلخواه را دریافت میکند سپس یک پازل تصادفی ایجاد میکند پس از بررسی قابل حل بودن آن روش حل را از کاربر دریافت کرده و سعی میکند پازل تصادفی را به node هدف برساند و سپس مراحل را به کاربر نمایش میدهد

تابع solve_your_puzzle_with_special_goal :

این تابع همانند تابع قبلی میباشد اما با این تفاوت که به جای پازل تصادفی این بار از کاربر میخواهد که پازل را مشخص نماید