به نام خدا

گزارش پروژه میانترم

برنامه نویسی پیشرفته(1)

رضا مرادی 9623100

- **number_puzzle.h**

کلاس NumberPuzzle به این صورت تعریف شده است که داخل آن یک کلاس nested به اسم Node وجود دارد و متد های لازم برای Node در آن تعبیه شده است و بقیه ی متد ها که برای کل پازل لازم است در کنار کلاس Node تعریف شده است در اینجا قصد داریم پازل را از دو الگوریتم BFS و DFS حل کنیم.

- **number_puzzle.cpp**

خط 3 تا 5- برای اندازه گیری زمان داینامیک اجرای الگوریتم DFS این دو متغیر را باید گلوبال تعریف میکردیم امکان استفاده از static وجود نداشت چون برنامه طوری طراحی شده است که هرچقد کاربر بخواد از اول ران میشود.

خط 8 تا 56- پازل طوری طراحی شده است که ابعاد آن نیز متغیر است به این صورت که کاربر میتواند آنرا تعیین کند برای این منظور متغیر col برابر تعداد ستون های پازل و n تعداد المان های پازل است و متغیر puzzle نیز المان های پازل میباشد در contructor های Node این مقادیر initialize میشوند تابع ()set_puzzle وظیفه ی مقدار دهی اولیه ی puzzle را دارد که میتواند پیش فرض انجام شود یا یک نوع پازل خاص برای آن تعیین شود متغیر x در واقع اندیس المان صفر یا خالی است.

خط 58 تا 81- متغیر goal_puzzle در واقع آن حالت نهایی و هدف است که پازل در صورت برابر بودن با آن حل شده است اینکار را با تابع ()goal_test بررسی میکنیم و این حالت را نیز قرار داده ایم که به جای مقایسه کردن با متغیر goal_puzzle با پازلی که از ورودی میگیرد اینکار را کند در تابع ()set_x نیز مقدار x را ست میکنیم.

خط 83 تا 227-در این قسمت حرکات پازل را شبیه سازی کرده ایم ولی این حرکات به دوقسمت تقسیم میشوند: یک زمانی که میخواهیم واقعا پازل به حرکت بعدی تغییر کند و دو نیز زمانی که میخواهیم از حرکات پازل یک گراف بسازیم. حالت اول زمانی است که کاربر خودش پازل را حل میکند اما برای حالت دوم ابتدا نیاز داریم که بین Node های یک جور ارتباط برقرار کنیم برای اینکار از std::shared_ptr استفاده میکنیم که #include<memory> را انجام داده ایم اگر از پوینتر استفاده نکنیم به ارور بر میخوریم زیرا استفاده از آبجکت یک کلاس در تعریف آن کلاس بدون استفاده از پوینتر مجاز نیست حال برای این منظور یک نود parent و یک لیست از نود بچه ها به نام children تعریف میکنیم برای این منظور از دنباله ی deque استفاده میکنیم که دنباله ی مناسبی برای اینکار میباشد. برای شبیه سازی حرکات در حالتی که میخواهیم گراف را تشکیل بدهیم در کل باید ابتدا بررسی کنیم که اصلا حرکت مجاز هست یا نه در ورودی

تابعی مثل move_to_right(int* p, int i) متغیر i اندیس المان خالی یا صفر است برای سریع شدن برنامه از ورودی دریافت میشود و متغیر p آن پازلی است که حرکات نسبت به آن تولید میشود در تابع بررسی میکنیم که حرکت مجاز است یا نه سپس به وسیله ی تابع coppy_puzzle پازل p را در پازل pc کپی میکنیم که پازل ورودی دچار تغییر نشود سپس جابه جایی را انجام میدهیم و یک نود بچه به وسیله ی پازل جدید میسازیم و آنرا به به children اضافه میکنیم و نود فعلی را به عنوان والد نود بچه قرار میدهیم تا روابط بین انها به خوبی برقرار باشد در نهایت برای اینکه بفهمیم حرکت مجاز بوده است یا نه یک متغیر bool برمیگردانیم. در توابعی که نیز ورودی نمیگیرند تنها پازل را تغییر میدهیم.

خط 229 تا 246- پازل به صورت مربعی چاپ میشود برای تغییر رنگ console از "\033[1;33m" استفاده کرده ایم که 1 برای bold شدن و 33 نشان دهنده ی رنگ زرد است و در اخر با "\033[0m" این تنظیمات را reset کرده ایم.

خط 248 تا 272- ابتدا بررسی میکنیم که پازل نود فعلی با پازل ورودی یکسان هست یا نه سپس در تابع بعدی اندیس یک المان خاص را بر میگردانیم و در تابع اخر تمام بچه های یک نود را بوسیله ی جابه جایی ها ست میکنیم.

خط 274 تا 306- در کاستراکتور NumberPuzzle مانند نود ابعاد را مقدار دهی میکنیم و یک متغیر goal_puzzle نیز داریم بوسلیه ی توابع آنرا مقدار دهی میکنیم در دیستراکتور تمام پوینتر هارا nullptr میکنیم تا فضا به سیستم عامل برگردد.

خط 308 تا 335- در این تابع قصد داریم که یک پازل رندم بسازیم این تابع تعداد حرکات را از کاربر میگیرد و به همان تعداد حرکات را روی پازل خروجی انجام میدهد در ابتدا یک نود برای خروجی تعریف میکنیم سپس باید عدد رندم بسازیم برای اینکار include<random> میکنیم و ابتدا srand(time(0)) را انجام میدهیم طبق این دستور از زمان صفر هر یک ثانیه یک ثانیه عدد رندم تولید میشود ینی rand() آپدیت میشود یک count تعریف میکنیم تا حرکات خالص را بشماریم باقی مانده ی عدد رندم را به 4 حساب میکنیم و هر کدام را به صورت قرار دادی به یک حرکت نظیر میکنیم خروجی حرکت هارا در متیغر bool میریزیم تا اگاه شویم که حرکات انجام شده است یانه در نهایت بررسی میکنیم که اگر در یک حالت هیچ کدام از حرکت ها انجام نشده بود از count کم شود که این جز حرکت ها حساب نشود.

خط 337 تا 402- میخواهیم پازل را با استفاده از الگوریتم BFS حل کنیم برای اینکار ابتدا زمان شروع تابع را در start ذخیره میکنیم و include<chrono> میکنیم خروجی تابع از نوع لیست نود است که درواقع

لیست مسیر از حالت نهایی به حالت آغازی است در ورودی نود ریشه یا آغازی و عمق ماکزیمم را میگیریم در ابتدا بررسی میکنیم که نود ورودی خود جواب هست یا نه سپس وارد یک حلقه میشویم ابتدا زمان فعلی را پرینت میکنیم تا نشان دهیم تابع در حال حل پازل است یک لیست داریم به نام open_list که قبل از حلقه نود ریشه را در آن ذخیره میکنیم یک نود داریم به اسم current_node که هر دفعه برابر با اولین عضو لیست open_list قرار داده میشود سپس این نود را به closed_list انتقال میدهیم و انرا از ابتدای closed_list حذف میکنیم سپس بوسیله ی ()expand_node بچه هارا ست میکنیم یک حلقه روی بچه ها تشکیل میدهیم سپس شرط پیروزی را روی تک تک بچه ها بررسی میکنیم سپس اگر به حالت نهایی رسیدیم goal_reached را برابر با true قرار میدهیم و به کمک تابع path_trace مسیر رسیدن از نودی که قرار داریم به نود ریشه را داخل path_to_solution میریزیم بررسی میکنیم اگر بچه ی فعلی نه در open_list ونه در closed_list بود به open_list اضافه شود تا در دور های بعدی بچه هایش بررسی شوند در نهایت عمق خالص را محاسبه کنیم تا تشخیص دهیم در چه عمقی قرار داریم در نهایت زمانی که طول کشید تابع کارش را انجام دهد را پرینت میکنیم (علت استفاده از این دولیست برای این است که از حرکات تکراری و افتادن توی لوپ جلوگیری شود.)

خط 404 تا 423- تابع اول بررسی میکنیم که توی یک لیست ورودی نود ورودی وجود دارد یا نه اینکار را به وسیله ی بررسی کردن is_same_puzzle روی تک تک نود های لیست انجام میدهد و تابع path_trace نیز بوسیله ی رابطه ی والد فرزندی تک تک نود ها را از نود ورودی تا نود ریشه انجام میدهد لیست ورودی باید به صورت & باشد تا مقادیر تا تغییر کند این دوتابع در الگوریتم BFS به کار برده شدند.

خط 425 تا 600- ابتدا لیست خروجی را میسازیم برای الگوریتم DFS از روش بازگشتی یا Recursive استفاده شده است اینکار سرعت را به شدت بالا برده است ولی سختی پیاده سازی را بیشتر کرده است زیرا از توابع قبلی نمیتوان در این الگوریتم استفاده کرد برای این الگوریتم کلا رویکرد پازلی جدید را در پیش میگیریم برای راحتی کار در اینجا پازل را تبدیل به دو بعدی میکنیم پس برای اینکار grid را میسازیم که متغیر کلاس نیز هست و آنرا برابر با پازل نود ریشه قرار میدهیم دو متغیر origin_x و origin_y مختصات المان صفر یا خالی هستند که مقدار آنرا پیدا میکنیم سپس تابع search_dfs را صدا میزنیم تا به جواب برسد که در ادامه توضیح داده خواهد شد زمان پردازش را پرینت میکنیم.

در تابع search_dfs چند ورودی داریم که به ترتیب مختصات صفر و عمق فعلی و حرکت بازی شده ی قبلی است در ابتدا ی تابع بررسی میکنیم که عمق مقدارش زیاد نشود در اینصورت تابع ریترن میکند متغیر moves تمام حرکات بازی شده را ذخیره میکند به این صورت که تنها المان جابه جا شده را ذخیره میکند

سپس با کمک تابع is_correct() بررسی میکنیم که پازل به جواب نهایی رسیده است یا نه در این صورت best_depth را برابر عمقی قرار میدهیم که در آن به جواب رسیده ایم در واقع این همان طول کوتاه ترین مسیر رسیدن به جواب است و جابه جایی های ما نیز در کوتاه ترین حالت نیز قرار دارد و باید انرا در best_moves بریزیم این کار چنیدن بار آپدیت میشود تا به بهرین حالت برسد سپس تایم فعلی را پرینت میکنیم برای جابه جایی ها از x1 تا x4 و از y1 تا y4 استفاده میکنیم اگر حرکت قبلی مشابه حرکت فعلی باشد متغیر هارا 1- میدهیم تا تکرار نشوند اگر حرکت مجاز باشد ان حرکت را روی grid انجام میدهیم سپس خود تابع را دوباره صدا میزنیم و سپس حرکت را برمیگردیم انگار grid فرقی نکرده است این کار را برای 4 حرکت انجام میدهیم اینکار تاجایی ادامه پیدا میکند که دیگر عمق اجازه ندهد و ریترن کنیم بعد از اینکه serach_dfs کار خودش را انجام داد زمان گذرانده شده را چاپ میکنیم حال باید best_moves را به یک لیست نود تبدیل کنیم که همان path_to_solution است ابتدا از ریشه شروع میکنیم سپس تک تک حرکات را داخل لیست ذخیره میکنیم بدین صورت که با کمک find_element حرکت را انجام میدهیم و ذخیره میکنیم و path_to_solution را ریترن میکنیم.

- **mid_project.cpp**

یک تابع به اسم run_puzzle() داریم که کل پازل را ران میکند به این صورت که ابتدا پیام خوش آمد گویی به کاربر را با رنگ بنفش چاپ میکند و با کمک متن هایی که چاپ میشود کاربر قادر خواهد بود که در راستایی که میخواهد برنامه اجرا شود این برنامه محدودیتی برای ابعاد پازل ندارد پس میتواند 2*2 یا 3*3 یا 4*4 و ... باشد ولی برای جلوگیری از وقت گیر بودن مسئله ابعاد 2 تا 6 لحاظ شده است در مواقعی که کاربر میخواد داده را روی فایل به کاربر بدهد یک فایل input.txt در مسیر h/. وجود دارد که کاربر میتواند آنرا ادیت کند یا یک فایل به جای آن با همین نام replace کند داخل فایل فقط باید المان های یک پازل باشد که میتواند به صورت خطی باشد یا ماتریسی در نهایت از کاربر سوال میشود که میخواهد دوباره پازل ران شود یا نه اگر موافق بود true و اگر مخالف بود false ریترن میشود.

برای سرعت گرفتن انجام عملیات کاربر میتواند از دستورات زیر به طور مثال در ابتدای برنامه استفاده کند:

nyyn100n30: حل پازل رندوم با 100 حرکت با دی اف اس با ماکزیمم عمق 30

nyyn50y30: حل پازل رندوم با 50 حرکت با بی اف اس با ماکزیمم عمق 30

- **main.cpp**

در یک لوپ تا زمانی که کاربر میخواهد پازل ران میشود و در هر بار با دستور system(“clear”) محتوای console به کلی پاک میشود.

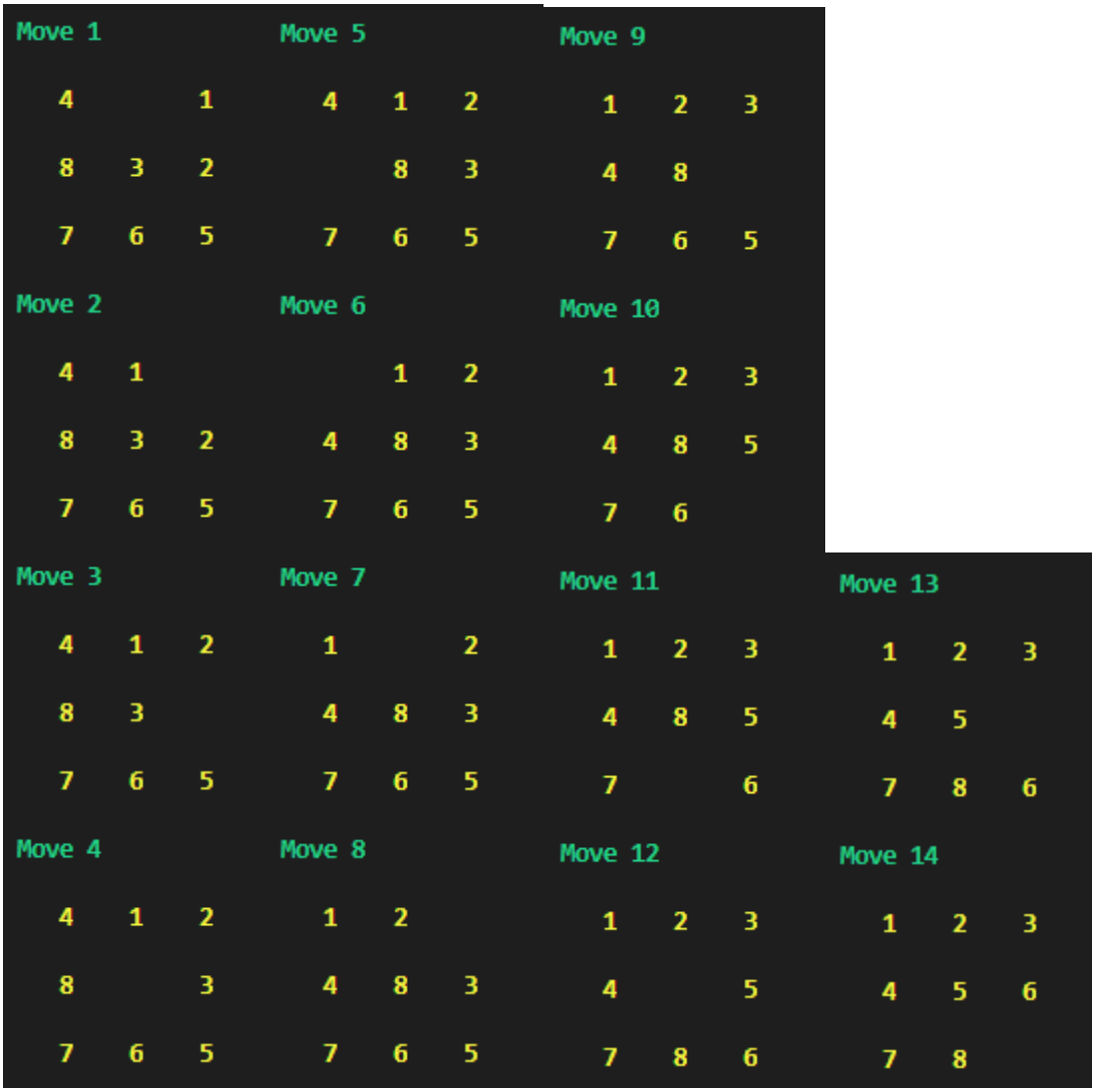
به طور مثال nyyn100n30 را وارد میکنیم تا یک پازل را به صورت پیش فرض با DFS حل کند.

Github repo link: https://github.com/morapoly/ap1399-midproject

مطابق شکل ها داریم:

```
------------------------
Welcom to Number Puzzle.
------------------------

Do you want to solve or see?
(Enter y to solve a puzzle, or n to see how a puzzle is solved:)
nyyn100n30

The default puzzle is 8-Puzzle which is 3x3, Do you want to continue?
(Enter y to continue, or n to change the puzzle:)

The puzzle is set to 8-Puzzle or 3x3.
The goal puzzle is:

   1    2    3

   4    5    6

   7    8

Do you want to continue?
(Enter y to continue or n to set the goal puzzle:)

Do you want to set the initial puzzle which is started from it?
(Enter y to set, or n to be set random, automatically:)

How many moves do you want the randomizer moves?
(Enter an integer greater equal than 10, great numbers make it harder to solve:)
```

```
The initial puzzle is set to:

   4    3    1

   8         2

   7    6    5

It's going to solve puzzle.
Do you want to be going to solved by BFS algorithm or DFS?
DFS is optimized and is so faster!

(Enter y for BFS or n for DFS:)
(Enter an integer greater equal than 30 for max depth:)
************
Goal reached!
************


>> Elapsed time: 158.754 ms


Tracing path...
Move 0

   4    3    1

   8         2

   7    6    5
```



>> در انتهای برنامه میتوان دوباره پازل را ران کرد تا آنرا با حالات قبل مقایسه کرد.
