**نام و نام خانوادگی** : منا رداد

**پروژه** : ارضا محدودیت(kakuro)

مدل سازی:

در این مسئله متغیر ها را خانه های سفید میگیریم که قرار است آن ها را پر کنیم و دامنه متغیر ها از اعداد 1 تا 9 می باشد

که با توجه به محدودیت های بازی مانند قرار نگرفتن دو عدد یکسان در یک سطر و یک ستون و برابر شدن جمع اعداد با خانه های سبز مسئله و همچنین محدودیت های حل الگوریتم ها ( مانند الگوریتم forward checking که در هر مرحله بررسی میکند جمع خانه های مقدار دهی شده تا الان بیشتر از خانه های سبز مجاور نباشد.) تعیین میشود.

نحوه پیاده سازی الگوریتم ها :

**Backtracking”"**

در این الگوریتم ما 4 تا تابع به نام های ) is\_win,number\_can\_be,backtracking,main ) داریم که ابتدا تابع **main** را توضیح می دهیم : در این تابع ما ابتدا صفحه بازی را مشخص می کنیم board[100][100][3] که تعداد سطرها و ستون ها را مشخص میکند و برای هر کدام از خانه های جدول سه حالت را در نظر می گیریم که 0 تعیین کننده مقدار و 1 تعیین کننده جهت است که جهت به صورت ساعتگرد ( 0 بالا و 1 راست و...) می باشد و 2 تعیین کننده mode بازی میباشد که mode شامل خانه های سیاه(1-) خانه های سفید ( 0 ) و خانه های سبز (1) می باشد. در وهله اول به عنوان ورودی خانه های سفید را مقدار 0 ، خانه های سیاه را مقدار 1- و خانه های سبز را مقدارشان به همراه جهت پر شدنشان را میدهیم.

متغیری به نام dep تعریف می کنیم که نشان دهنده تعداد خانه های سفید می باشد که قرار است پر شوند. سپس تابع backtrack را صدا میکنیم که این تابع زمانی که جدول را کامل حل بکند مقدار true برمیگرداند که در این صورت زمین بازی را چاپ می کند.

**Backtraking**

این تابع صفحه بازی ، تعداد سطر و ستون ها و تعداد خانه های سفید (dep)را می گیرد .

ابتدا با فراخوانی تابع is\_win چک میکند که اگر همه خانه های سفید به صورت درستی پر شده اند مقدار true را بر میگرداند در غیر این صورت به ازای تمام خانه هایی که mode آن ها 0 است یعنی سفید اند و مقدار آن ها نیز 0 است یعنی تا الان مقدار دهی نشده اند تابع number\_can\_be را برای آن ها فراخوانی می کنیم که این تابع vector ای از اعدادی که در یک خانه سفید با توجه به محدودیت های سطر و ستون و ... مجاز به قرار گیری هستند را بر می گرداند. به ازای تمام اعدادی که در این vector قرار می گیرند تابع backtracking را برای بررسی صدا می کنیم. اگر مقدار قرار گرفته در خانه های سفید درست باشد تابع backtracking مقدار true را برمی گرداند در غیر این صورت یعنی مقدار انتخاب شده برای این خانه مناسب نبوده پس مقدار این خانه سفید را دوباره صفر می کنیم.

**Number\_can\_be**

این تابع vector ای شامل اعدادی که می توانند در خانه های سفید قرار بگیرند را برمیگراند به این صورت که علاوه بر تعداد سطر ها و ستون ها موقعیت هر کدام از خانه های سفید را با pi و pj مشخص می کند.

یک ارایه numbers داریم که شامل اعدادی است که میتوانند در خانه های سفید قرار بگیرند . با توجه به محدودیت قوانین بازی که اعداد یک سطر و یک ستون نمیتوانند یکسان باشند ما در دو حلقه for ابتدا برای سطر pi و سپس برای ستون pj اعدادی که در این سطر و ستون قرار گرفتند را با flag true مشخص می کنیم، که این اعداد نمی توانند در لیست اعداد مجاز برای خانه سفید قرار بگیرند. اعدادی که متغیر flag در این دو حلقه false بماند مجاز برای خانه سفید مشخص شده هستند پس بنابراین آن را در vector امان push\_back می کنیم.

**Is\_win**

این تابع ابتدا سطر و ستون ها را چک میکند اگر خونه ای خالی باشد یعنی جدول پر نشده است و بازی تمام نشده پس مقدار false را بر میگرداند. زمانی که تمام خانه ها سفید مقدار دهی شد در هر سطر و ستون بررسی می کنیم که مقدار دهی خانه ها سفید درست انجام شده است یا نه.

برای این منظور متغیر sum را صفر قرار داده و در جستجوی خانه های سبز رنگ هستیم ، اگر خانه ای سبز رنگ باشد و جهت آن 0 باشد یعنی خانه سبز در پایین قرار گرفته و باید مجموع خانه های بالای آن برابر خانه سبز گردد بنابراین یک حلقه بر روی خانه های سفید بالای خانه سبز رنگ بر اساس کاهش سطر ها میزنیم و مقدار هر خانه سفید را با sum جمع می کنیم.در نهایت اگر sum برابر با مقدار خانه سبزرنگ نشد نشان دهنده این است که خانه ها به صورت درست پر نشده اند و مقدار false را برمیگردانیم در غیر این صورت سطر ها و ستون های بعدی را چک می کنیم. حال اگر خانه سبز رنگی وجود داشت که جهت آن 1 یعنی راست بود باید مجموع خانه های سفید سمت راست آن را محاسبه کنیم بنابراین حلقه ای بر اساس افزایش k که نشان دهنده ستون خانه های سفید است میزنیم و دوباره sum را چک میکنیم.سایر جهت ها نیز به همین ترتیب چک می شوند.

**Forward checking”"**

در الگوریتم forward checking باید نگاهی به آینده داشت که زودتر متوجه مسیر اشتباه شد.

در این الگوریتم ما 4 تا تابع به نام های ) forward\_check,number\_can\_be,backtracking,main ) داریم.

در این الگوریتم توابع **main,number\_can\_be** همانند الگوریتم backtracking میباشد و تنها تفاوت در دو تابع دیگر است. تابع **backtracking** در این الگوریتم تفاوت اندکی با همین تابع در الگوریتم قبلی دارد به این صورت که اگر تعداد خانه های سفید صفر باشد و عمل forward\_check روی صفحه بازی با موفقیت انجام شود true برمیگرداند در غیر این صورت اگر خانه سفیدی وجود داشته باشد و forward\_check روی صفحه بازی با وجود خانه های سفید نتیجه درست ندهد مقدار false را برمیگرداند.

تابع **forward\_check** مانند تابع is\_win الگوریتم قبل می باشد با این تفاوت که در اینجا شرط اینکه اگر خانه سفیدی وجود داشت با مقدار 0، مقدار false برگرداند را نداریم، چون با توجه به نگاه آینده امکان بدست آوردن جواب وجود دارد و تغییر دیگر این است که یک متغیر last داریم که هنگامی که همه خانه های جدول مقدار غیر 0 داشته باشند true میشود.

یک متغیر flag\_zero داریم که مقدار اولیه آن false است در این تابع مانند تابع is\_win به دنبال خانه های سبز میگردیم که مجموع سطر و ستون مربوط به آن را بدست بیاوریم البته شرط هایی را به آن اضافه کرده ایم

همانند قبل اگر خانه سبز رنگی با جهت 0 وجود داشت مقادیر خانه های سفید بالای آن را جمع میکند و اگر در این ستون خانه سفیدی با مقدار 0 بود flag\_zero را true میکند. حال اگر خانه سفیدی با مقدار 0 وجود داشته باشد و مجموع خانه های سفید دیگر برابر خانه سبز رنگ باشد مقدار false را برمیگرداند چون جمع خانه های سفید تا به کنون قبل از پر کردن خانه سفید با مقدار 0 برابر با مقدار خانه سبز رنگ شده است . در غیر این صورت اگر جمع خانه های سفید پر شده بیشتر از خانه سبز رنگ باشد باز هم false برمیگرداند ، در غیر این صورت اگر flag last ، true باشد یعنی همه خانه های جدول پر شده باشد و جمع خانه های آن ستون برابر خانه سبزرنگ نشده باشد در نتیجه باز هم false برگردانده میشود. اگر هیچ کدام از این شرط ها برقرار نشده باشد و false برگردانده نشده باشد همین شروط را برای جهات دیگر چک میکنیم و اگر false برگردانده نشد یعنی نتیجه true است و جواب درست را یافته ایم.

**MRV”"**

هدف ما در این الگوریتم انتخاب خانه های سفیدی با کمترین محدودیت برای پر کردن میباشد که محدودیتی که در این الگوریتم در نظر گرفتیم min تعداد اعدادی است که در یک خانه میتوانند قرار بگیرند.

در این الگوریتم ما 4 تا تابع به نام های ) is\_win,number\_can\_be,backtracking,main ) داریم. که تابع **main,numbr\_can\_be** همانند الگوریتم backtracking می باشد.

در تابع **backtracking** با تابع is\_win ابتدا چک میشودکه بازی به پایان رسیده است یا خیر ، اگر win برابر 1- شد یعنی اعدادی که در جدول قرار داده ایم اشتباه است و امکان رسیدن به جواب درست وجود ندازد بنابراین false برمیگرداند برای انتخاب کمترین محدودیت ارایه MRV[n][m] را تعریف می کنیم و به ازای هر خانه سفید با مقدار 0 تابع number\_can\_be را روی آن اجرا میکنیم و تعداد اعداد موجود را در ارایه mrv قرار می دهیم.mrv را به ازای خانه های غیر صفر ( سبز و سیاه) صفر قرار می دهیم. متغیر min را برابر 10 قرار می دهیم زیرا بیشترین مقدار number\_can\_be 9 میتواند باشد.به ازای هر خانه موجود در جدول که mrv ان صفر نباشد ( خانه های سفید ) مقایسه mrv خانه های سفید با min انجام میشود و اینگونه خانه سفید با کمترین محدودیت را پیدا می کنیم. حال مختصات آن خانه سفید را با x,y مشخص کرده و تابع number\_can\_be را برای آن خانه صدا زده و همانند الگوریتم های قبل تابع backtrack را روی آن انجام میدهد و مانند الگوریتم های قبل تابع backtrack را ادامه میدهد.همین روند را برای انتخاب خانه های بعدی با کمترین محدودیت ادامه می دهیم.

در تابع **is\_win** مانند تابع forward\_check در الگوریتم قبلی فرض اینکه اگر خانه سفیدی با مقدار 0 وجود داشت false برگرداند را نداریم به این دلیل که ممکن است خانه های سفید 0 را بتوانیم مقدار دهی کنیم.

تابع is\_win در اینجا سه مقدار برمیگرداند 1 به معنای اینکه همه خانه های سفید مقدار درست دارند و 1- به این معنا که مجموع خانه های یک سطر یا ستون بیشتر از خانه سبز رنگ مربوط به آن شده و خانه های سفید به اشتباه مقدار دهی شده اند و به جواب نمیرسیم و مقدار 0 به معنای آنکه مجموع خانه های سفید کمتر از خانه های سبز است یعنی همچنان امکان برد وجود دارد.

به ازای هر خانه سبز رنگی سطر و ستون مربوط به آن را جمع میکنیم اگر خانه سفیدی وجود داشت که مقدار دهی نشده بود sum سطر یا ستون را با خانه های سبز رنگ مقایسه می کنیم. اگر sum بیشتر از خانه سبزرنگ بود 1- را برمیگرداند در غیر این صورت 0 بر میگرداند. اگر sum سطر یا ستون برابر با خانه سبز رنگ آن نبود مقدار 0 را بر میگرداند و در نهایت برای جهات دیگر چک میشود و اگر true بود یعنی جدول را به درستی پر کرده ایم.