به نام خدا

محمدعلی خسروآبادی

تابع generate_sudoku:

این تابع برای تولید یک جدول سودوکو استفاده می‌شود. ابتدا با پر کردن خانه‌های قطری شروع می‌کند، سپس جدول را با استفاده از یک الگوریتم حل می‌کند و در نهایت تعدادی از اعداد را برای ایجاد یک پازل سودوکو حذف می‌کند.

تابع fill_diagonal:

این تابع خانه‌های قطری جدول سودوکو را با اعداد پر می‌کند. برای اینکه هر ۳ بخش قطری از جدول دارای اعداد ۳x۳ یک تا نه باشد، استفاده می‌شود.

تابع fill_box:

این تابع یک جعبه ۳ از جدول سودوکو را با استفاده از اعداد تصادفی پر می‌کند. این اعداد به صورت تصادفی انتخاب شده و به شکلی انتخاب می‌شوند که هر عدد فقط یک بار در هر جعبه ظاهر شود ۳x۳.

تابع is_safe:

این تابع بررسی می‌کند که آیا قرار دادن یک عدد در موقعیت خاصی از جدول امن است یا خیر. این کار با بررسی ردیف، ستون و بخش ۳ محل قرارگیری عدد انجام می‌شود ۳x۳.

تابع used_in_row:

این تابع بررسی می‌کند که آیا یک عدد خاص قبلاً در یک ردیف مشخص استفاده شده است یا خیر.

تابع used_in_column:

مشابه تابع قبل، این تابع بررسی می‌کند که آیا یک عدد در ستون مورد نظر قبلاً استفاده شده است.

تابع used_in_box:

این تابع بررسی می‌کند که آیا عدد مورد نظر در بخش ۳ که شامل محل مورد نظر می‌شود، قبلاً استفاده شده است ۳x۳.

تابع find_unassigned_location:

این تابع برای پیدا کردن اولین خانه خالی (مقدار صفر) در جدول سودوکو به کار می‌رود. اگر تمام خانه‌ها پر شده باشند، نشانه‌ای است که جدول حل شده است.

تابع remove_elements:

این تابع تعداد مشخصی از اعداد را از جدول حل شده سودوکو حذف می‌کند تا یک پازل سودوکو ایجاد کند که باید حل شود.

تابع solve_sudoku:

این تابع الگوریتم بازگشتی را برای حل جدول سودوکو اجرا می‌کند. این کار با انتخاب یک خانه خالی، قرار دادن یک عدد امن در آن، و سپس ادامه این روند تا زمانی که جدول کاملاً پر شود یا هیچ عدد امنی برای قرار دادن باقی نماند، انجام می‌شود.

تابع display_grid:

این تابع جدول سودوکو را برای نمایش چاپ می‌کند. هر خانه جدول روی صفحه نمایش داده می‌شود و برای ردیف‌ها خط جدیدی شروع می‌شود.

پس از تخصیص یک عدد به یک خانه، این تابع دامنه‌های خانه‌های مرتبط را به‌روزرسانی می‌کند و مقدار تخصیص داده شده را از دامنه‌های آن‌ها حذف می‌کند.

تابع`solve_csp`

را به عنوان ورودی می‌پذیرد که نمایانگر مقادیر تخصیص داده شده به هر خانه‌ی `assignment`یک دیکشنری به نام که نشان دهنده مجموعه اعداد مجاز برای هر `domains`جدول تا آن لحظه است. همچنین یک دیکشنری دیگر به نام که تعداد گام‌های انجام شده توسط `steps_counter`خانه‌ی خالی است را دریافت می‌کند. علاوه بر این، یک آرایه به نام الگوریتم را می‌شمارد، ورودی می‌گیرد.

در ابتدای تابع، تعداد گام‌ها یک واحد افزایش می‌یابد تا نشان دهنده اجرای یک مرحله از تابع باشد. سپس با استفاده از تابع `find_unassigned_location` ، خانه‌ی بعدی که هنوز عددی به آن تخصیص داده نشده را جستجو می‌کند. اگر تمام خانه‌ها تخصیص داده شده باشند، به این معنی است که پازل به طور کامل حل شده و تابع مقدار `True`برمی‌گرداند.

در صورتی که خانه‌ای خالی پیدا شود، تابع شروع به امتحان کردن تمام اعداد ممکن در دامنه‌ی آن خانه می‌کند. برای هر عدد، ابتدا با استفاده از تابع `is_valid_assignment` بررسی می‌شود که آیا قرار دادن آن عدد در خانه‌ی مورد نظر با تخصیص‌های قبلی تداخل ندارد. اگر قرار دادن عدد مشکلی ایجاد نکند، عدد به خانه تخصیص داده می‌شود.

فراخوانی می‌شود تا دامنه‌های خانه‌های مرتبط با خانه‌ی تازه تخصیص داده شده `forward_check` پس از آن، تابع به‌روزرسانی شوند و از حضور عدد تخصیص داده شده در دامنه‌های آن‌ها جلوگیری شود. اگر این به‌روزرسانی باعث شود که دامنه‌ای خالی شود و دیگر عددی برای تخصیص دادن وجود نداشته باشد، تابع `False` برمی‌گرداند تا سیگنالی برای بازگشت به گام قبلی باشد.

اگر تخصیص عدد جدید و به‌روزرسانی دامنه‌ها موفقیت‌آمیز باشد، تابع به صورت بازگشتی خودش را با حالت جدید فراخوانی می‌کند. این فرآیند تا زمانی که پازل حل شود یا عددی برای تخصیص دادن باقی `domains` و `assignment` نماند، ادامه پیدا می‌کند.

برمی‌گرداند تا الگوریتم بتواند مسیرهای دیگری را برای حل پازل `False` در نهایت، اگر هیچ راه‌حلی پیدا نشود، تابع شناخته می‌شود، به الگوریتم اجازه می‌دهد تا از (backtracking) امتحان کند. این روش، که به نام بازگشت به گذشته تصمیمات قبلی که به بن‌بست منجر شده‌اند، عقب‌نشینی کند و راه‌حل‌های ممکن دیگری را جستجو کند.