حمیدرضا همتی 9631079

پروژه شماره 2

حل جدول سودکو

- کلاس Node
- کلاس step

Node
+ position: tuple (X,Y) + numericDomain: list + colorDomain: List +hasValue: boolian +assignedValue: integer
+ Neighbors(self): return 2 list(row and column neighbors) + degree(self, list of nodes): integer (calculate degree value of the given node) + MRVUpdate(self, list of nodes): this method update the numeric domain of given node +adjacentNeighbors(self): return 2 list(row and column adjacent neighbors)

Step
+ coordinate: tuple + assignedValue: integer + stepCounter: integer
+ coordinateGetter(self): int + assignedValueGetter(self): int + stepCounterGetter(self): int

- تابع bestNextNode:
  - ❖ این تابع با توجه به مقدار degree و MRV تمام node ها بهترین Node را برای مقدار دهی بعدی انتخاب می‌کند.
- تابع solve:
  - ❖ این تابع در واقع تابع اصلی است نحوه عملکرد آن به صورت زیر است.

1. اول از همه چک میکند که جدول تمام شده است یا نه (با استفاده از تابع endGameCheck)
2. اگر جدول هنوز خانه خالی داشت در این صورت تابع bestNextNode را صدا میکند و مختصات بهترین انتخاب بعدی را میگیرد.
3. از دامنه آن خانه یک مقدار را به آن خانه assign میکند.
4. حال که بهارین خانه انتخاب شده و مقدار آن هم assign شده است یک object از کلاس step میسازد و به لیست path اضافه میکند.
5. سپسforward checking انجام میدهد.
   a. اگر دامنه‌ی هیچ خانه ایی تهی نشده بود ادامه میدهد.
   b. در غیر این صورت تابع backtrack صدا میشود.

- تابع extractInputFile
  - ❖ این تابع فایل input را میخواند و اطلاعات مورد نیاز را استخراج می‌کند.
- تابع numericSudokuMaker
  - ❖ این تابع رنگ ها را نادیده میگیرد و یک جدول Sudoku کلاسیک ایجاد میکند که تنها دارای عدد است و برای خانه هایی که عدد ندارند مقدار "-" را قرار میدهد.
- لیست path
  - ❖ برای track کردن مسیر و ایجاد امکان backtrack کردن

- تابع forwardChecking:
  - ❖ این تابع دامنه تمام خانه ها را چک میکند و اگر دامنه ایی تهی بود مقدار False و اگر این طور نبود مقدار True را بر میگرداند
- تابع backtrack:

توضیحات زیر را به عنوان یک TODO برای خودم نوشته بودم وقتی میخواستم الگوریتم backtracking را پیدا کنم....چون کامل و واضح هست همین رو قرار میدم به عنوان توضیح این تابع

backtracking and forward chaining:

after each value assignment to a node i should update the MRV of nodes

- IF there was no 0 in MRV list, the new assignment was ok
- ELSE there is a problem and that is a sign that we should start backtracking.
  + the problem is the last assignment (or assignments before that) was bad
  - we should look up to the path we take and find the last taken step.
    + in each Step in path there is 3 important attribute
      1. coordinate: coordinate of the node that we update it's value
      2. assignedValue: the value that we assigned to that node
      3.stepCounter: the number of step
  - after find the Step and extract its attributes
    1. set node.hasValue to False: set the bad chosen node hasValue boolean to False
    2. update the MRV of nodes again: to update the value domain of changed neighbors of that bad chosen node
    3. remove assignedValue from the domain of that bad chosen node: because that assignedValue was the problem from the beg
       and if we assign that value to that node again then (HAMON ASHO HAMON KASE همون آش و همون کاسه)
       1. IF the remaining values in domain of the node was >= 1:
          its OK
       2. ELSE: we get FUCKED UP:
          we should backtrack again