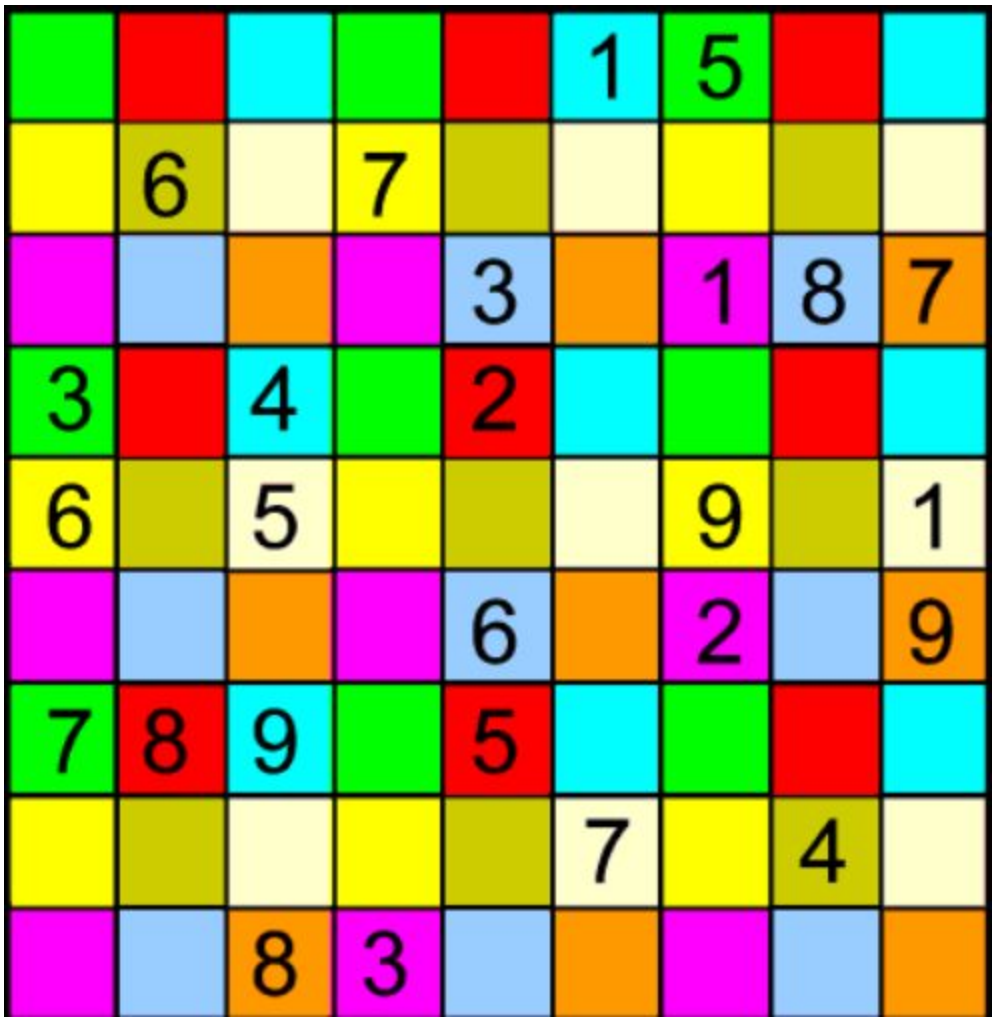
به نام خدا

علی عزیزی - 9731042

پروژه دوم درس هوش مصنوعی



در ابتدا به فرمول بندی مسئله و نحوه ی در نظر گرفتن متغیرها، دامنه ها و محدودیت ها می پردازیم، سپس در ادامه بخش های مهم پیاده سازی انجام شده، کلاس ها و توابع را به تفصیل بررسی خواهیم کرد.

فرمول بندی مسئله :

متغیر ها : برای فرمول بندی متغیر ها در این مسئله، یک کلاس به نام Variable در نظر گرفته ایم که شامل دو متغیر x و y است که مختصات هر cell را نشان می دهد و هنگام ورودی گرفتن،پس ساختن متغیر ها، همه ی آن ها را در یک لیست قرار می دهیم.

دامنه : برای دامنه نیز در این پیاده سازی، یک کلاس به نام Domain در نظر گرفته ایم که شامل موارد زیر می باشد:
یک مقدار int که نشان دهنده عدد هر cell و یک کاراکتر که نشان دهنده ی رنگ هر cell می باشد.

برای نگهداری دامنه های مختص به هر متغیر از ساختمان داده دیکشنری استفاده شده است، به اینصورت که در این ساختمان داده، هر key مربوط به یک variable است و value متناظر با آن لیستی از domain ها می باشد.
این دیکشنری هنگام ورودی گرفتن ایجاد شده و در ادامه تحت هر forward checking ، دامنه هر متغیر دستخوش تغییرات می شود.

محدودیت ها : از آنجایی که ما در این مسئله به سه نوع محدودیت گوناگون مواجه هستیم، یک سوپر کلاس به نام Constraint در نظر می گیریم و کلاس هایی که برای محدودیت های مختلف در نظر می گیریم و در ادامه به شرح آن ها پرداخته شده است، از این سوپر کلاس ارث بری می کنند.
سوپر کلاس Constraint شامل موارد زیر است:

- لیستی از متغیرها که در آن محدودیت مشترک هستند.
- تابع satisfied که باید در هر کدام از زیرکلاس ها override شود و مشخص می کند تحت چه شرایطی محدودیت مورد نظر ارضا می شود.

به منظور سهولت در محاسبات بعدی محدودیت ها در این مسئله به صورت باینری در نظر گرفته شده اند و زیر کلاس هایی که از کلاس اصلی ارث بری می کنند بصورت زیر می باشند.

```
class different_number_constraint(Constraint[Variable, Variable]):
    def __init__(self, v1: Variable, v2: Variable) :
        super().__init__([v1, v2])
        self.v1 = v1
        self.v2 = v2
    def satisfied(self, assignment: Dict[Variable, Domain]) -> bool:
        if self.v1 not in assignment or self.v2 not in assignment:
            return True
        x1 = int(assignment[self.v1].number)
        x2 = int(assignment[self.v2].number)
        return x1 != x2
```

```
class different_color_constraint(Constraint[Variable, Variable]):
    def __init__(self, v1: Variable, v2: Variable) :
        super().__init__([v1, v2])
        self.v1 = v1
        self.v2 = v2
    def satisfied(self, assignment: Dict[V, D]) -> bool:
        if self.v1 not in assignment or self.v2 not in assignment:
            return True
        return assignment[self.v1].color != assignment[self.v2].color
```

```python
class color_priority_constraint(Constraint[Variable, Variable]):
    def __init__(self, v1: Variable, v2: Variable) :
        super().__init__([v1, v2])
        self.v1 = v1
        self.v2 = v2
    def satisfied(self, assignment: Dict[V, D]) -> bool:
        if self.v1 not in assignment or self.v2 not in assignment:
            return True
        x1 = int(assignment[self.v1].number)
        x2 = int(assignment[self.v2].number)
        if x1 > x2:
            return color_priority[assignment[self.v1].color] > color_priority[assignment[self.v2].color]
        if x1 < x2:
            return color_priority[assignment[self.v1].color] < color_priority[assignment[self.v2].color]
```

توضیح کد :

کلاس CSP :

این کلاس شامل متغیرها (attribute) های زیر است:

- لیستی از متغیر ها
- یک دیکشنری domains که هر key آن یک متغیر و value آن لیستی از آبجکت هایی از کلاس Domain می باشد.
- یک دیکشنری به نام constraints که کلید آن متغیر و value آن لیستی از محدودیت ها می باشد که این دیکشنری نشان دهنده آن است که هر متغیر در کدام محدودیت ها شرکت دارد و برای مقدار دهی باید آن ها را satisfy کرد.

توابع کلاس CSP به شرح زیر است:

consistent(self, variable: V, assignment: Dict[V, D])

از این تابع برای اضافه کردن یک محدودیت جدید استفاده می شود.

consistent(self, variable: V, assignment: Dict[V, D])

هنگامی که این تابع فراخوانده می شود، مشخص می کند که آیا یک مقدار دهی مشخص همراه با مقدار دهی متغیر جدید آیا باعث ارضای محدودیت ها می شود و یا خیر

minimum_remaining_values(self, unassigned: List[Variable], local_domain: Dict[V, List[D]]):

این تابع همان mrv است که متغیرها را بر اساس اندازه دامنه آنها مرتب کرده و آن هایی که کمترین تعداد دامنه را دارند در قالب یک لیست بر می گرداند.

degree(self, l: List[Variable])

این تابع خروجی لیست تابع قبلی را به عنوان ورودی می گیرد و سپس هیوریستیک درجه را بر روی آن ها اعمال کرده و یک متغیر به عنوان خروجی بر می گرداند که مجموع این دو هیوریستیک در تابع select متمرکز شده است.

is_failure(self, ds: Dict[V, List[D]])

این تابع دامنه همه متغیر ها را چک می کند و بررسی می کند آیا با شکست مواجه شده ایم یا خیر، یعنی بررسی می کند آیا دامنه متغیری تهی شده است یا نه؟

backtracking_search(self, ds: Dict[V, List[D]], assignment: Dict[V, D])

این تابع همانطور که از نام آن مشخص است جستجوی backtrack را انجام میدهد که در انجام این جستجو از هیوریستیک های درجه و mrv که پیش از این توضیح داده شده اند و همچنین forward checking استفاده شده است که با استفاده از سه تابع محدود سازی دامنه را انجام می دهد که کد آن ها در ادامه آورده شده است.

```python
def number_forward_checking(vs: List[Variable], ds: Dict[V, List[D]], num):
    for v in vs:
        for i, element in enumerate(ds[v]):
            if int(element.number) == int(num):
                ds[v][i] = -1
        while ds[v].count(-1) > 0:
            ds[v].remove(-1)
```

```python
def color_forward_checking(vs: List[Variable], ds: Dict[V, List[D]], c):
    for v in vs:
        for i, element in enumerate(ds[v]):
            if element.color == c:
                ds[v][i] = -2
        while ds[v].count(-2) > 0:
            ds[v].remove(-2)
```

```python
def color_priority_forward_checking(vs: List[Variable], ds: Dict[V, List[D]], value: Domain):
    for v in vs:
        for i, element in enumerate(ds[v]):
            flag1 = color_priority[element.color] < color_priority[value.color] and int(element.number) > int(value.number)
            flag2 = color_priority[element.color] > color_priority[value.color] and int(element.number) < int(value.number)
            if flag1 or flag2:
                ds[v][i] = -3
        while ds[v].count(-3) > 0:
            ds[v].remove(-3)
```

شرح مهمترین توابع استفاده شده در برنامه :

تابع **read input** : در این تابع ورودی های مطابق فرمت گفته شده در صورت پروژه از یک فایل متنی خوانده می شوند و متغیر های مهم نظیر دیکشنری های domains و لیست متغیر ها مقدار دهی می شوند.

تابع **number neighbors** : این تابع یک متغیر به عنوان ورودی می گیرد و لیستی از متغیر ها را بر می گردند که از لحاظ عدد با متغیر ورودی در محدودیت هستند، به عبارت دیگر این تابع متغیر هایی که سطر یا ستون مشترک با متغیر داده شده را دارند به عنوان خروجی بر می گرداند.

تابع **color neighbors** : این تابع نیز یک متغیر به عنوان ورودی می گیرد و لیستی از متغیر هایی که به واسطه رنگ با متغیر ورودی ، در محدودیت هستند را بر می گرداند.

تابع **main** : در این تابع میان ابتدا یک آبجکت از کلاس CSP با متغیر هایی که در که در تابع read_input مقداردهی شده اند، ساخته می شود. سپس میان هر دو متغیری که نیاز است، بسته به موقعیت آن متغیر ها یک محدودیت باینری ایجاد می شود و در نهایت تابع backtrack_search روی آبجکت ساخته شده فراخوانی می شود و نتیجه توسط تابع **print_res** چاپ می شود.
(به صورت رنگی رنگی و خوشگل :))))

نمونه ورودی و خروجی :

در نهایت دو نمونه ورودی و خروجی هایی که به ازای که به ازای آن ها توسط برنامه تولید شده، آورده شده است.

ورودی 1 (نمونه ورودی موجود در دستور پروژه) :

3 5

r g b y p

#* b* 1#

#* 3r #*

#* g 1#*

خروجی 1 :

```
Colored Sudoku


1y 2b 3r

2b 3r 1y

3g 1b 2g

--- 0 minutes and 0.005982875823974609 seconds ---

Process finished with exit code 0
```

ورودی 2 :

5 5

r g b y c

#* #* #* #* 1#

#* b *# *# 5#*

g *# *y* 4# #*

#* #* #* #* #*

2# 1# #* #* 3#

خروجی 2 :

```
Colored Sudoku


1y 3r 2y 4b 5r

4b 2y 1c 5g 3b

2y 4b 5g 3b 1y

5r 1c 3b 2y 4g

3g 5r 4g 1c 2y

--- 0 minutes and 0.41610169410705566 seconds ---

Process finished with exit code 0
```