دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش پنجم

پیاده سازی حل یک مساله تصمیم گیری با استفاده از ارضای محدودیت

نگارش

سروش آریانا

استاد

دکتر مهدی قطعی

فروردین ۱۴۰۰

مقدمه

این گزارش شامل پیاده سازی الگوریتم جستجوی backtracking در پایتون است .جستجوی backtracking یک الگوریتم بازگشتی است که برای یافتن راه حل های مسئله های ارضای محدودیت (CSP) استفاده می شود. در این برنامه برای حل یک سودوکو، که از مسائل معروف ارضای محدودیت است از این الگوریتم استفاده میکنیم.
کد برنامه در فایل zip به همراه گزارش ارسال شده است.

۱- توضیحات درباره الگوریتم و پیاده سازی

یک الگوریتم جستجوی backtracking سعی می کند در هر بازگشت مقداری به یک متغیر اختصاص دهد و backtrack کند اگر که مقدار مجاز دیگری برای اختصاص دادن نداشته باشد .یک الگوریتم خالص backtracking نسبتا کند است، که می توانیم با هدایت آن در جهت صحیح، عملکرد آن را بهبود ببخشیم.به طور مثال میتوانیم از سازگاری یال برای تسریع در این الگوریتم استفاده کنیم ، استفاده همزمان از این دو بدین معناست که ما فقط مقادیر قانونی را در دامنه برای هر متغیر گنجانده ایم و بنابراین مقادیر کمتری برای انتخاب در هر بازگشت وجود دارد .همچنین می توانیم از هیوریستیک محدودترین متغیر(حداقل-باقی مانده- مقادیر)برای انتخاب متغیر با کمترین مقادیر قانونی نیز استفاده کنیم.
در ابتدا یک کلاس برای نمایاندن سودوکو تعریف میکنیم و از تابع __init__ برای ساخت سودوکو استفاده میشود . در این تابع ابتدا متغیر های مسئله تنظیم شده و سپس برای اعداد سودوکو با استفاده از سازگاری یال دامنه تعریف میکنیم.

```
class SdkWithBT():

    def __init__(self, states: [], length, subColLen, subRowLen):

        self.states = states
        self.length = length
        self.subColLen = subColLen
        self.subRowLen = subRowLen
        self.scope = {}
        self.changeScope()
```

پس از آن برای چک کردن اینکه بدانیم یک مقدار میتواند در یک سلول خاص از سودوکو قرار بگیرد تابع consistencyCheck را تعریف میکنیم .در این تابع ابتدا سازگاری برای قرار گرفتن در سطر و سپس در ستون چک میشود و در صورتی که در هر کدام غیرمجاز باشیم، تابع مقدار false را برمیگرداند .پس از آن مقدار شروع در سطر و ستون تنظیم شده و با استفاده از آنها سازگاری در هر زیر ماتریس (subMatrix)نیز چک می شود.

```
def consistencyCheck(self, thisnum, row, col) -> bool:
    for i in range(self.length):
        if self.states[row][i] == thisnum:
            return False
    for j in range(self.length):
        if self.states[j][col] == thisnum:
            return False
    rowInit = (row // self.subRowLen) * self.subRowLen
    colInit = (col // self.subColLen) * self.subColLen;
    for i in range(rowInit, rowInit + self.subRowLen):
        for j in range(colInit, colInit + self.subColLen):
            if self.states[i][j] == thisnum:
                return False
    return True
```

تابع بعدی، changeScope برای به روزرسانی دامنه سلولها تعریف میشود .در این تابع در دو حلقه ی تودرتو ابتدا چک میشود که آیا سلول داده شده خالی است یا خیر(صفر بودن مقدار سلول به معنای خالی بودن آن است .) سپس

به ازای تمام مقادیر ممکن سازگاری آن با تابع قبلی چک میشود و در صورت درست بودن خروجی تابع این مقدار به مقدارهای مجاز اضافه میشود.

```
def changeScope(self):
    self.scope = {}
    vars = []

    for i in range(self.length):
        for j in range(self.length):
            if (self.states[i][j] == 0):
                vars = []
                for k in range(1, self.length + 1):
                    if (self.consistencyCheck(k, i, j) == True):
                        vars.append(k)

                if (len(vars) >= 1):
                    self.scope[(i, j)] = vars
```

نام تابع BT مخفف Backtracking است و پیاده سازی الگوریتم در این قسمت میباشد.
در ابتدا اولین خانه ی خالی از بورد را پیدامی کنیم، به ازای تمام مقادیر ممکن سازگاری چک شده و با اولین مقدار مجاز آن خانه مقدار دهی میشود .پس از آن با اجرای بازگشتی الگوریتم ابتدا شرایط حل شدن پازل سودوکو و اتمام آن بررسی می شود، در صورت برقراری شرط اتمام الگوریتم اعلام شده و در غیر اینصورت خانه ی خالی بعدی پیدا شده و اجرا تکرار میشود.

```
def BT(self) -> bool:
    y, x = self.findFirstEmpty()

    if y is None or x is None:
        return True

    for k in range(1, self.length + 1):
        if self.consistencyCheck(k, y, x):
            self.states[y][x] = k
            if self.BT():
                return True
            self.states[y][x] = 0
    return False
```

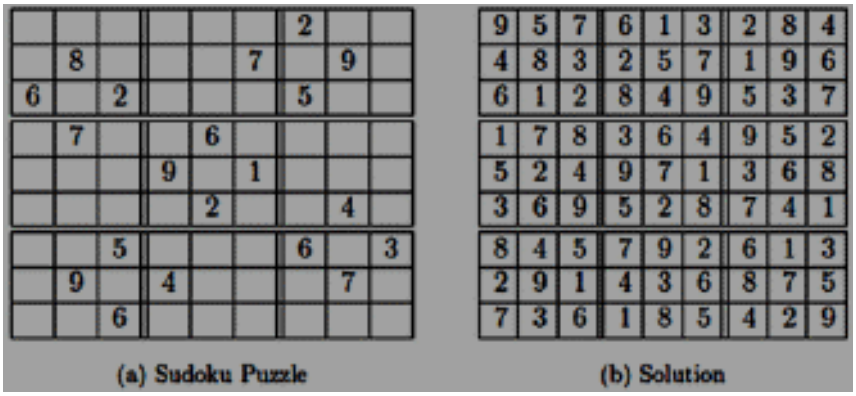
تابع boardPrint برای نمایش خروجی به شکلی شبیه به جدول سودوکو تعریف شده است:

```
def boardPrint(self):

    for i in range(self.length):
        print('# ', end='')
        if i != 0 and i % self.subRowLen == 0:
            for k in range(self.length):
                print(' # ', end='')
                if (k + 1) < self.length and (k + 1) % self.subColLen == 0:
                    print(' * ', end='')
            print(' #')
            print('# ', end='')
        for j in range(self.length):
            if j != 0 and j % self.subColLen == 0:
                print(' # ', end='')
            digit = str(self.states[i][j]) if len(str(self.states[i][j])) > 1 else ' ' + str(self.states[i][j])
            print('{0} '.format(digit), end='')
        print(' #')
```

۲- نتایج و خروجی کد

به عنوان مثال پازل سمت چپ در بدنه برنامه به عنوان ورودی داده میشود و انتظار میرود پازل سمت راست به عنوان خروجی دریافت شود:



(a) Sudoku Puzzle (b) Solution

تعریف ورودی و ساخت پازل از آرایه ورودی:

```
inpboard = [0, 0, 0, 0, 0, 0, 2, 0, 0,
            0, 8, 0, 0, 0, 7, 0, 9, 0,
            6, 0, 2, 0, 0, 0, 5, 0, 0,
            0, 7, 0, 0, 6, 0, 0, 0, 0,
            0, 0, 0, 9, 0, 1, 0, 0, 0,
            0, 0, 0, 0, 2, 0, 0, 4, 0,
            0, 0, 5, 0, 0, 0, 6, 0, 3,
            0, 9, 0, 4, 0, 0, 0, 7, 0,
            0, 0, 6, 0, 0, 0, 0, 0, 0]
length = 9
subColLen = 3
subRowLen = 3
```

```
initial_state = []
row = []
counter = 0
for i in inpboard:
    counter += 1
    row.append(i)
    if counter >= length:
        initial_state.append(row)
        row = []
        counter = 0

board = SdkWithBT(initial_state, length, subColLen, subRowLen)
```

و در نهایت پس از اجرای تابعBT ، پس از 20 ثانیه خروجی برنامه:

```
Solution for  this board:
#  9  5  7  #  6  1  3  #  2  8  4  #
#  4  8  3  #  2  5  7  #  1  9  6  #
#  6  1  2  #  8  4  9  #  5  3  7  #
#  #  #  #  *  #  #  #  *  #  #  #  #
#  1  7  8  #  3  6  4  #  9  5  2  #
#  5  2  4  #  9  7  1  #  3  6  8  #
#  3  6  9  #  5  2  8  #  7  4  1  #
#  #  #  #  *  #  #  #  *  #  #  #  #
#  8  4  5  #  7  9  2  #  6  1  3  #
#  2  9  1  #  4  3  6  #  8  7  5  #
#  7  3  6  #  1  8  5  #  4  2  9  #

--- 20.078938961029053 seconds ---
```

منابع و مراجع

[1] https://stackoverflow.com

[2] https://www.tutorialspoint.com

[3] https://www.geeksforgeeks.org