به نام خدا

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

پروژه دوم

سودوکو رنگی

دریا زارع مهذبیه

۹۷۳۱۰۸۶

فهرست

خط مشی اصلی: ۳
ساختمان داده : ۳
دامنه: ۳
سازگارسازی کمان عددی: ۳
هیوریستیک ها: ۳
قلب کد(تابع بازگشتی): ۳
مهره اصلی کد: ٤
تابع forwardcheck رنگ: ٤
نکات: ٥
توجیح روش: ٥
نکته پایانی: ٥

خط مشی اصلی:

برای حل سودوکو رنگی مسعله رو به حل دو بخش جدول عددی و جدول رنگی تقسیم میکنیم

ساختمان داده :

برای هر بخش یک جدول از مقادیر منتسب شده و یک جدول سه بعدی از دامنه های هر خانه از جدول قبل.

رنگ های را در یک دیکشنری با key حروف رنگ و مقداری برابر اولویت آن نگهداری میکنیم که موقع ورودی گرفتن key را به dict میدهیم و مقدار را در جدول مربوطه ذخیره میکنیم.

ضمنا برای چاپ کردن نهایی جدولی با key و مقدار برعکس همین جدول را ذخیره میکنیم تا سریع از جدول بتواند رنگ چاپ کند.

دامنه:

برای سودوکو عددی دامنه 0تاn است و برای جدول رنگ مقادیر 0 تاm مقادیر ۰ نشاندهنده حالی بودن آن خانه(از رنگ یا عدد) است

سازگارسازی کمان عددی:

ابتدا دامنه ها را مطابق ورودی سازگار کمان و نود میکنیم. سپس تابعی بازگشتی برای پر کردن عدد های جدول مینویسیم در هر مرحله تابع forwardcheck را صدا میکنیم که مقدار عددی را از سطر و ستون حذف میکند و اگر خانه خود و خانه همسایه آن رنگی مخالف ۰ داشت دامنه آن خانه را به روزرسانی میکند(اگر رنگ اولویت بیشتر داشت اعداد ۱ تا عدد جدول را از دامنه آن حذف میکنیم) سپس چک میکنیم دامنه های خانه های به روزرسانی شده خالی نشده باشند اگر خالی باشد تابع ۰ برمیگرداند اگر نه دامنه جدید را برمیگرداند.

نکته: برای دامنه جدید باید deepcopy از دامنه اصلی بسازیم تا وقتی تابع backtrack میکند دامنه های تولید شده ازبین روند.

هیوریستیک ها:

هیوریستیک MRV را اعمال میکنیم که لیستی از خانه های جدول منتسب نشده با طول کمینه را برمیگرداند. سپس لیست برگردانده شده را به تابع هیوریستیک degree میدهیم که موثر ترین نود را باز میگرداند(نود های منتسب نشده در سطر و ستون بیشتری داشته باشد).

قلب کد(تابع بازگشتی):

ابتدا هیوریستیک MRV را صدا میزنیم. لیست بازگردانده شده را به تابع degree میدهیم که سطر و ستون یک نود را بازگرداند.

در نهایت بر روی دامنه نود بازگردانده شده پیمایش میکنیم و به ترتیب مقدار را به آن منتسب میکنیم. در هر انتساب تابع را بازفراخوانی میکنیم(تابع بازگشتی) و به جای دامنه اصلی دامنه ای که تابع forwardcheck بازگردانده است را ورودی میدهیم. اگر تابع بازگشتی ۰ برگرداند مقدار بعدی از دامنه به آن نود داده میشود اگر همه انتساب ها انجام شد اما همه ۰ برگرداندند آن خانه را ۰ میکنیم(بی انتساب) و ۰ را برمیگردانیم.(بازمیگردد به انه قبلی منتسب شده)

تابعی با دقیقا همین ساختار برای رنگ ها هم داریم که برای هیوریستیک MRV یک تابع مشترک را فراخوانی میکنند ولی برای forwarrdcheckو هیوریستیک درجه تابع دیگری را که نود هایی که بررسی میشود در degree متفاوت است(همسایه های منتسب نشده از چهارهمسایه)

مهره اصلی کد:

تابع MRV زمانی که همه خانه های جدول پرشده باشند -۱ برمیگرداند که همان شرط خاتمه ی تابع بازگشتی است. در شرط خاطمه تابع بازگشتی عددی تابع بازگشتی رنگ را صدا میزنیم. اگر ان تابع ۱ برگرداند این تابع تمام میشود.

بدین صورت در صورتی که تابع رنگ یک مقداردهی VALID برای جدول پر شده از عدد نیابد ۰ برمیگرداند و تابع عددی دوباره عدد ها را Update میکند تا زمانی که به جواب برسیم.

```
def f(table,domainTable,colortable,colordomainTable,m,n):
    row,col = MRV(domainTable,table,n,1)
    if row == -1:
        # nums filled
        if co(table,colortable,colordomainTable,m)!= 0:
                return 1
        return 0
    for assinNum in domainTable[row][col]:
        table[row][col] = assinNum
        updated = forwadChecking(row,col,assinNum,domainTable,table,colortable)
        if (updated!=0):
            if f(table,updated,colortable,colordomainTable,m,n)!= 0:
                return 1
    table[row][col]=0
    return 0
```

تابع forwardcheck رنگ:

چون جدول از عدد پر است دامنه ها زود تغییر میکنند. بدین صورت که :

```
    colordomainTable = copy.deepcopy(cdt)
    temp = colortable[row][col]
    if col>0:
        if numtable[row][col] > numtable[row][col-1]:
            for color in range(1,temp):
                while True:
                    try:
                        colordomainTable[row][col-1].remove(color)
                    except ValueError:
                        break
        else:
            for color in range(temp,m+1):
                while True:
                    try:
                        colordomainTable[row][col-1].remove(color)
                    except ValueError:
```

```
                    break

        while True:
                try:
                    colordomainTable[row][col-1].remove(temp)
                except ValueError:
                    break
        if (colortable[row][col-1]==0 and len(colordomainTable[row][col-
1])==0) or (colortable[row][col-1]==0 and len(colordomainTable[row][col-1])==0):
            return 0
```

ابتدا مطابق دلایلی که در بالا گفتیم deepcopy از دامنه ورودی میگیرند.

سپس باید برای هریک از چهار همسایه چک کند که آیا ورجود دارد یا خیر.

در صورت وجود باید با توجه به عدد خانه ها دامنه خانه مجاور را به روزرسانی کند به این صورت که اگر عدد خانه مجاور بزرگتر باشد رنگی با اولویت کمتر از رنگ خانه کنونی نمیتواند بگیرد و همه این رنگ ها را حذف میکنیم. و اگر عدد کمتر باشد اولویت های بیشتر را حذف میکنیم.

در نهایت رنگ منتسب به این خانه را از خانه مجاور حذف میکنیم(در else حذف میشود اما برای اطمینان دوباره مینویسیم) در نهایت اگر دامنه ای از خانه مجاور منتسب نشده(رنگ ۰) خالی شد تابع ۰ برمیگرداند. اگر به انتها رسید دامنه جدید را برمیگرداند.

نکات:

این کد ابتدا اگر دو خانه مجاور رنگ داشته باشد دامنه عددی آن ها را هم موقع forwardcheckNum آپدیت میکند. بدین صورت جدول های فقط رنگی را هم به جواب نهایی میرسد.

توجیح روش:

من این مسعله را در دو مرحله عددی و رنگی حل کردم زیرا این روش سریع تر است

زیرا دامنه ها کوچک تر میشود سرعت حل جدول عددی افزایش مییابد و همچنین سرعت forwardcheck در تابع رنگی خیلی زیاد است هر بار به طور میانگین نصف دامنه کاهش می یابد در نهایت به نسبت حالاتی که بازگشت میخورد خیلی کمتر از حالاتی که مستقیم حل میشود دارد و بهینه تر است.

نکته پایانی:

تابع degree در داخل تابع MRV صدا زده میشود . اگر ورودی آخر تابع MRV ۱ باشد تابع درجه عددی و اگر صفر باشد تابع درجه رنگی صدا زده میشود(اگر خانه ای برای انتساب نباشد هیوریستیک درجه صدا زده نمیشود.)