جاروبرقی

رضا نقدی

توضیحات:

در پروژه جاروبرقی که با استفاده از زبان پایتون نوشته شده بجای استفاده از توابع گرافیکی پایتون مانند tkniter از html, css بدلیل سهولت بیشتر استفاده شده و توسط فریمورک Flask با بک اند ارتباط داده شده است.

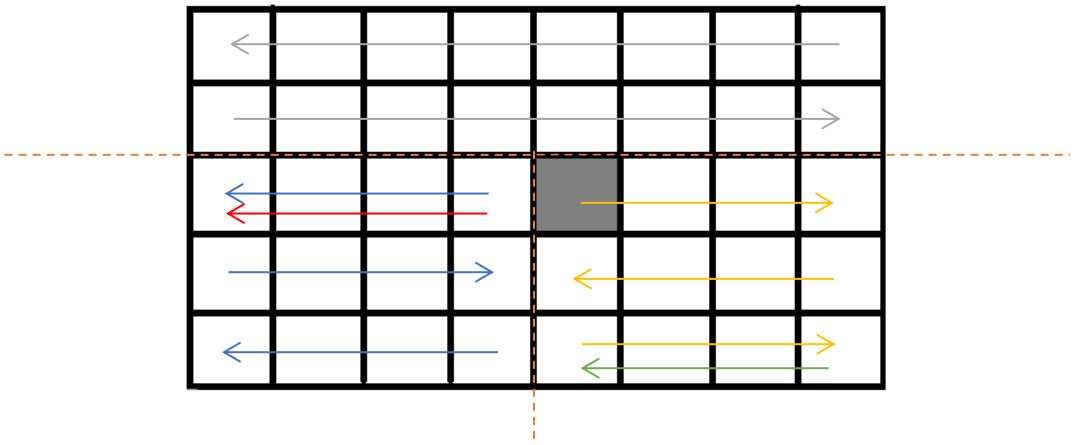
در پروژه Flask به ترتیب سه تابع و صفحه construct,prepare,clean تعریف شده تابع اول (construct) صرفا برای دریافت ابعاد ماتریس و قرار دادن مقادیر آن در متغیر های سراسری row,col استفاده می شود و پس از آن تابع و صفحه prepare را فرا میخواند.

این تابع با استفاده از ابعداد وارد شده اطلاعات سطر و ستون را در لیست rooms ذخیره و به صفحه prepare.html جهت نمایش به صورت ماتریسی از checkbox ها میفرستد (در صفحه html از دو حلقه for برای خواندن اطلاعات لیست ارسالی استفاده شده) بعد از نمایش ماتریس می توان اتاق های کثیف را انتخاب نمود که پس از ثبت دکمه تایید اطلاعات نقاط انتخابی به صورت Requet POST به بک اند ارسال و تابع prepare مجدد فراخوانی میشود تا تگ نقاط انتخابی را به "dirty" تغییر دهد و مجددا در همان صفحه html نمایش دهد, اینبار پس از انتخاب یک نقطه جهت مشخص کردن موقعیت جاروبرقی تابع clean فراخوانی می شود.

تابع clean با استفاده از الگوریتم پیاده سازی شده ترتیب حرکت خانه های ماتریس را به صورت [i,j] در لیست inf ذخیره و همچنین با مقایسه هر آیتم با مقدار قبل خود تشخیص می دهد که جهت حرکت به چه صورت می باشد و آن را در لیست mvs ذخیره میکند در نهایت با تکمیل شدن لیست ها و پایان الگوریتم دو لیست ساخته شده به صفحه html مربوطه ارسال می شوند (علت ارسال تغییرات به صورت لیست و نمایش ندادن لحظه ای حرکات عدم امکان رفرش کردن صفحه html به در یک تابع است که این روند درون صفحه html و با استفاده از JS اصلاح شده است)

الگوریتم:

نحوه تریس کردن ماتریس به این صورت است که الگوریتم ابتدا از نقطه شروع ماتریس را به صورت افقی به دو نیم تقسیم می کند و نیمه پایینی را به دو نیم عمودی تقسیم می کند ابتدا زیر ماتریس پایینی بخش سمت راست, سپس زیر ماتریس سمت چپ و بعد بخش بالایی ماتریس را تریس میکند.



بخش اصلی این الگوریتم تابع move می باشد ورودی این تابع رنج افقی و عمودی هر زیر ماتریس است که مسیر حرکت در هر سطر معکوس می شود (اگر در سطر اول حرکت به سمت راست باشد در سطر بعدی به سمت چپ حرکت می کند)

```
def move(inx):
    nonlocal i, j
    rm = 1
    if i%2 == 0:
        rm = 0
    for i in inx[0]:
        if i%2 == rm:
            for j in inx[1]:
                locate()

        else:
            for j in inx[2]:
                locate()
        locate()
```

اگر جاروبرقی در انتهای زیر ماتریس ابتدایی گیر بیفتد مانند فلش سبز رنگ به سمت چپ باز میگردد و در این حالت فلش قرمز اتفاق نمی افتد در غیر این صورت نیاز است که بعد از اتمام حرکت در زیر ماتریس دوم مانند فلش قرمز به سمت چپ حرکت کرد تا بتوان آخرین بخش را با استفاده از تابع move تکمیل کرد

```
if j != y:
    for j in range(col-2, y-1, -1):
                locate()
```

```
elif j != 0:
    for j in range(y-2, -1, -1):
                locate()
```

هنگامی که ستون اول هر سطر برای نقطه شروع استفاده شود دستورات زیر ماتریس دوم اجرا نمی شوند و نقطه اتمام با فاصله از زیر ماتریس آخر قرار می گیرد که با شرط زیر شده است

```
if i != x:
    for i in range(row-2, x-1, -1):
                locate()
```

پیچیدگی زمانی الگوریتم:

پیچیدگی زمانی تمامی دستورات O(1) است و پیچیدگی زمانی تابع move در صورتی که سطر ها را n و ستون ها را m در نظر بگیریم و نقطه شروع وسط ماتریس باشد دو زیر ماتریس پایینی دارای سطر و ستون 1/2n و 1/2m هستند که پیچیدگی هر دو با هم برابر است با 2(1/2n * 1/2m) = 1/2nm و بخش بالایی ماتریس برابر است با = (1/2n * m) 1/2nm که در نهایت پیچیدگی زمانی برابر با O(nm) می شود و اگر سطر و ستون را برابر در نظر بگیریم پیچیدگی O(n^2) است حالات تکرار خانه های ماتریس در بهترین حالت صفر, در حالت متوسط 1/2m و در بدترین حالت خود n+m است که بدترین حالت زمانی اتفاق می افتد که نقطه شروع [0,0] و تعداد سطرها فرد باشد در این حالت دو شرطی که با دایره قرمز و آبی مشخص شده اند اجرا می شوند

تشخیص جهت:

```
def locate():
    if j > inf[-1][1]:
        mvs.append("RIGHT")
        inf.append([i, j])
    elif j < inf[-1][1]:
        mvs.append("LEFT")
        inf.append([i, j])
    elif i > inf[-1][0]:
        mvs.append("DOWN")
        inf.append([i, j])
    elif i < inf[-1][0]:
        mvs.append("UP")
        inf.append([i, j])
```

برای تشخیص جهت به این گونه عمل می شود که هربار تابع locate() فراخوانی شود موقعیت نقاط جدید با نقطه قبلی درون لیست (ذخیره شده در متغیر برای تشخیص در لحظه) مقایسه می شوند در صورتی که j افزایش یافته باشد حرکت به سمت راست در صورت کاهش سمت چپ و برای سطر ها هم دو شرط به این گونه تعریف شده است

جاوا اسکریپت:

جاوا اسکریپ تعریف شده ترتیب نقاط و جهت هارا دریافت میکند و همچنین تمامی خانه های ماتریس را در یک آرایه ذخیره میکند که اندیس هر خانه برای پیمایش از طریق فرمول i * col + j بدست می آید و در صورتی که خانه ای دارای تگ 'dirty' باشد به عنوانه خانه کثیف در نظر گرفته میشود و برای پاکسازی تگ آن به 'clean' تغییر داده می شود خانه فعال هم دارای تگ 'pos' می باشد که در نهایت به 'clean' تغییر پیدا می کند و نمایش شکل هر کدام از تگ ها با استفاده از رنگی متمایز در css انجام شده است