دانشکده مهندسی کامپیوتر

دکتر رضا انتظاری ملکی

پاییز ۱۳۹۹

پروژه پایانی

تحلیل و طراحی الگوریتم ها

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

تاریخ تحویل: ۱۳۹۹/۱۱/۱۲

اجرای پروژه:

ابتدا باید علاوه بر پایتون کتابخانه های PIL و numpy و os روی سیستم شما نصب باشد. فایل seam carving.py را اجرا کنید.

ورودی: ابتدا آدرس عکس را به عنوان ورودی می گیرد. و سپس تعداد ستون هایی که میخواهید حذف شود را می پرسد و بعد هم تعداد سطر هایی که میخواهید حذف کنید را.(فقط یک عدد هستند). مثال:

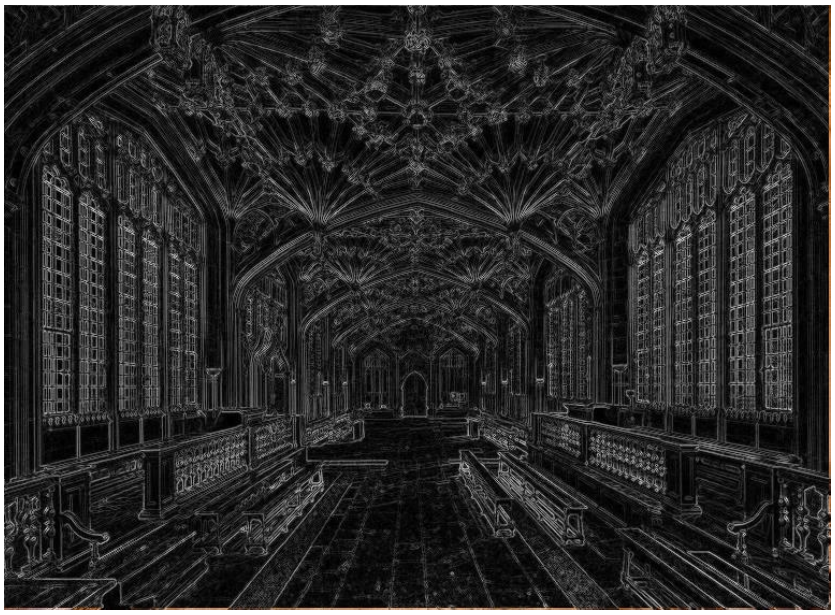
```
"C:\Program Files\Python36\python.exe" "D:/iust/term 5/design algorithms/project/Seam Carving.py"
please type the path of image: D:\iust\term 5\design algorithms\project\Sample Data\In_8.jpg
your photo size is 770 * 563
how many vertical seams do you want to remove? 20
how many horizontal seams do you want to remove? 13
```

سپس برنامه شروع به اجرا میکند و کمی زمان میبرد. (با توجه به ابعاد عکس و تعداد seam هایی که قرار است پیدا کند o آن متفاوت است، که جلوتر محاسبه خواهیم کرد.)

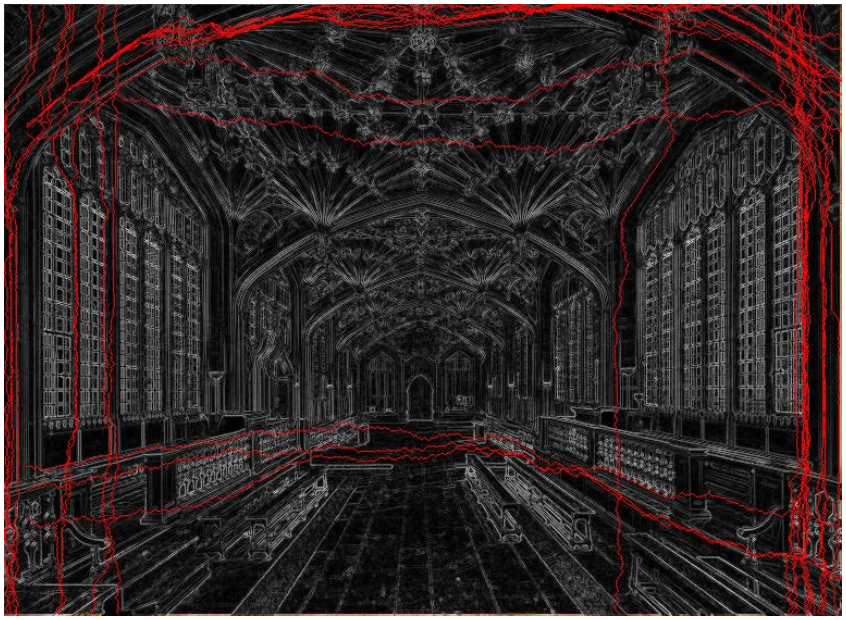
در نهایت برای ورودی بالا وقتی به این مرحله رسیدیم یعنی کار به پایان رسیده است:

```
9 vertical seam
10 vertical seam
11 vertical seam
12 vertical seam
13 vertical seam
14 vertical seam
15 vertical seam
16 vertical seam
17 vertical seam
18 vertical seam
19 vertical seam
20 vertical seam
1 horizontal seam
2 horizontal seam
3 horizontal seam
4 horizontal seam
5 horizontal seam
6 horizontal seam
7 horizontal seam
8 horizontal seam
9 horizontal seam
10 horizontal seam
11 horizontal seam
12 horizontal seam
13 horizontal seam
successfully saved in outputs folder in In_8
```

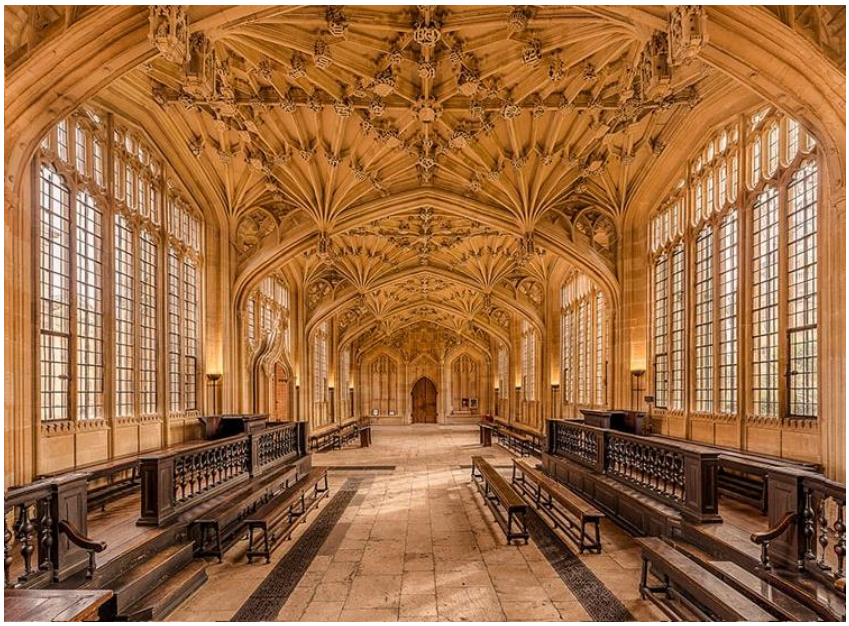
خروجی را می توانید در outputs و در دایرکتوری به اسم همان عکس اصلی مشاهده کنید. که شامل سه عکس است:



Energy photo که همان عکس معادل انرژی های محاسبه شده برای پیکسل ها می باشد.



Removed seams که درز های پاک شده را روی عکس قبلی نمایش می دهد. و مطابق ورودی ۱۳ تا افقی و ۲۰ تا عمودی هستند. این دو عکس ابعادی برابر با عکس اصلی دارند.



Result که عکس resize شده با ابعاد (13-763) * (20-770) یعنی 750*770 است.

عملکرد کد:

تابع **calculate energy**: این تابع بر اساس همان الگوریتم توضیح داده شده در داکیومنت پروژه یعنی dual gradint انرژی را برای یک پیکسل خاص (ورودی تابع)محاسبه میکند و عدد آن را بازمیگرداند.

این تابع صرفا تعدادی عملیات با O(1) انجام می دهد که تعدادشان ثابت است و لذا از O(1)است.

تابع **Create_Energy_picture**: کار این تابع ایجاد اولین عکس در خروجی است یعنی همان energy photo. بر این اساس که انرژی تک تک پیکسل ها که در main با فراخوانی تابع calculate_energy برای هر کدامشان خوانده شده را به عنوان RGB عکس جدیدی به آن می‌دهد. تا نقاط پر انرژی تر به نسبت انرژی شان سفید بشوند.

البته چون بازه ی انرژی ها از ۰ تا ۶۲۴.۵ می باشد ولی RGB بین ۰ تا ۲۵۵هستند، بیشترین انرژی را در main پیدا کرده و به این تابع می دهیم و تمام انرژی ها قبل از قرار گرفتن به عنوان سه مولفه ی رنگ هر پیکسل، در 255/maximum_energy ضرب میشوند و به این صورت بیشترین انرژی به عدد 255، scale می شود.

این تابع حلقه های تو در تو به اندازه طول و عرض عکس دارد چون ماتریس جدیدی را باید کاملا مقداردهی کند. که عملیات مقدار دهی آن از O(1) هستند و لذا در کل O(width*height) است.

تابع **Find_Vertical_seam**: کار این تابع پیدا کردن بهترین درز عمودی است. روشی که من در اینجا پیاده کردم نوعی الگوریتم حریصانه است. که از پیکسل بالا سمت چپ شروع می کند، و بین سه پیکسل پایینی اش کمترین را انتخاب می کند. سپس انرژی این ها را جمع میزند و همین کار را برای خانه ی جدید تکرار میکند و همینطور ادامه می دهد تا به پایین عکس برسد. مجموع انرژی ها را روی این درز نگه داشته ایم. حال همین کار را برای دومین خانه ردیف بالا انجام می دهیم و درز بهینه ی آن را پیدا می کنیم. و ... در نهایت بین تمام این درز ها، آنی که کمترین مجموع انرژی داشته است را به صورت لیستی از پیکسل ها برمیگردانیم.

این تابع هم تمام طول و عرض ماتریس انرژی ها (که همان اندازه عکس است) را می پیماید و مقایسه های مختلفی از O(1) را انجام می دهد. مینیمم درز هم در ابتدا با بی نهایت مقدار دهی شده و هر بار با O(1) پس از اتمام بررسی هر درز مقایسه می شود و محاسبه می شود. پس در کل این تابع هم O(width*height) است.

تابع **Remove_Seam**: در این تابع برای وجود نداشتن باگ موقع پاک کردن درز ها و خراب شدن اندازه های آرایه ها، ابتدا تمام ماتریس را به لیستی سه بعدی تبدیل میکنیم و با 1- کردن مقادیر خانه هایی که تابع قبلی تصمیم بر پاک کردنشان گرفته، آن ها را علامت گذاری میکنیم. سپس از ماتریس انرژی ها آن ها را پاک میکنیم و از عکس اصلی هم پاک میکنیم ولی در عکسی که میخواهیم نمایششان بدهیم به جای 1- کردن و سپس پاک کردن، صرفا آن را مقداری قرمز می گذاریم (عکس removed seams) و به همین خاطر ابعاد این عکس همان ابعاد عکس اصلی است. نکته ای که مهم است این است که باید از padding عکس هم پاک کنیم چون ماتریس عکس نمیتواند در سطر های مختلف طول مختلفی داشته باشد و باید یک پیکسل از آن هم پاک شود.

در نهایت باید عرض عکس را یکی کم کنیم و ماتریس عکس جدید را جایگزین قبلی کنیم(self.image) تا در صورت نیاز حاصل یا سیو شود و یا دوباره درز جدیدی در آن پیدا شود.

این تابع بستگی به وجود داشتن A (RGB بودن یا RGBA بودن عکس) به دو حالت تقسیم شده است که فرقی در الگوریتم نداشته و صرفا برای درستی کد و حفظ شفافیت است. در کل حلقه ای به طول remove_path که از تابع find_vertical_seam آمده داریم که به اندازهheight عکس می باشد.

پس تا اینجاO(height) را داریم. بعد تعدادی عملیات ساده مانند شرط ها و مقایسه ها و مقداردهی ها داریم که همگی O(1) هستند. ولی برای پاک کردن 1- های جایپذاری شده در remove_path، چه در ماتریس انرژی و چه در ماتریس عکس باید آن را بگردیم و لذا این هم حلقه های تو در تو دارد و O(width*height) می باشد.

مقداردهی های اولیه(قبل از main): که صرفا ورودی گرفتن از کاربر و ساختن دایرکتوری برای ذخیره عکس و بعضی مقدار دهی ها و تعریف متغیر هاست و همگی O(1) هستند و تاثیر خاصی ندارد.

تابع **main** : ابتدا در این تابع با استفاده از calculate_energy، ماتریسenergies را پر میکنیم و در همین حین ماکسیمم آن را هم که در create_energy_picture نیاز داشتیم می یابیم. بعد از ساختن energy photo با توابع توضیح داده شده از O(weight*width) ، باید به تعدادی که کاربر میخواهد درز عمودی پیدا کنیم و آن را پاک کنیم و دوباره پیدا کنیم و پاک کنیم و ... که می دانیم توابع پاک کردن و پیدا کردن آن O(width*height) است که چون به تعدادverticals بار تکرار می شود(عدد گرفته شده از کاربر برای تعداد ستون های عمودی که باید حذف شوند) پس تا اینجا O(verticals*width*height) را داریم.

ایده ای که من زدم تا از کد تکراری جلوگیری کنم این بود که پس از حذف کردن درز های عمودی(در صورتی که قرار است چیزی را حذف کنیم)، عکس را ۹۰ درجه بچرخانیم و سپس مشابه قبل درز های عمودی را بیابیم(که در واقع افقی ها هستند) و آن ها را پاک کنیم و بعد حاصل را ۲۷۰ درجه بچرخانیم تا در نهایت درز های افقی حذف شوند. پس بعد از سیو کردن و چرخاندن عکس حاصل شده، این بار horizontals دفعه توابع پیدا کردن و حذف کردن درز را فراخوانی میکنیم که O(horizontals*width*height) خواهد بود. و در نهایت حاصل عکس انرژی ها با درز های قرمز شده را در دایرکتوری گفته شده سیو میکنیم.

پس پیچیدگی زمانی کد در کل O((horizontals+verticals)*width*height) می باشد.

نکته ۱: به غیر از این الگوریتم حریصانه که من پیاده سازی کردم الگوریتم های دیگری هم وجود دارد مثلا با برنامه نویسی پویا که در اینجا توضیح داده شده است. اما از نظر زمانی فرق خاصی با حریصانه نداشت: https://avikdas.com/2019/07/29/improved-seam-carving-with-forward-energy.html

نکته ۲: ماتریس انرژی را می توان پس از حذف هر درز مجددا حساب کرد اما این کار بسیار زمانبر است در حالی که تاثیر چندانی در کیفیت عکس های خروجی نداشت(امتحان کرده ام) و کیفیت خروجی هم اکنون هم همانطور که در فولدر outputs مشاهده میکنید مطلوب است.

نکته ۳: به علت کم بودن محدودیت حجم در کوئرا، تصاویر خروجی من در لینک زیر است:

https://files.fm/u/bquawmggk#sign_up