گزارشکار پروژه‌ی کارخانه فرشبافی

طراحی الگوریتم‌ها

استاد راهنما: دکتر پیمان ادیبی

اعضای گروه:

امیر فیض

محمدکاظم هرندی

بهار۱۴۰۲

فهرست مطالب

طراحی

طراحی فرش‌های جدید

برای پاسخگویی به این قسمت از پروژه از الگوریتم رنگ‌آمیزی گراف[1] استفاده کرده‌ایم. این الگوریتم برای تخصیص رنگ به گره‌های یک گراف به نحوی که هیچ دو گره مجاوری همرنگ نباشند، استفاده می‌شود. بعد از انتخاب این بخش از منوی برنامه یک لیست از نمونه گراف‌های (فرش) موجود در فایل موجود به کاربر نشان داده می‌شود. در این مرحله کاربر یکی از این نمونه هارا انتخاب می‌کند سپس این الگوریتم مشخص می‌کند که هر یک از راس‌های گراف باید با چه رنگی، رنگ آمیزی بشوند.

توضیحات الگوریتم

۱. ابتدا یک آرایه‌ی نتیجه (result) با طول V تعریف می‌شود و تمام مقادیر آن به -۱ مقداردهی اولیه می‌شوند. این آرایه، برای نگه‌داری رنگ‌های اختصاص داده شده به گره‌ها استفاده می‌شود.
۲. رنگ اول به گره اول اختصاص داده می‌شود.(result[0] = 0)
۳. آرایه‌ای موقتی به نام available با طول V تعریف می‌شود و اولیه تمام مقادیر آن به true مقداردهی اولیه می‌شوند. این آرایه، برای نشان دادن رنگ‌های موجود برای اختصاص به گره‌های مجاور استفاده می‌شود.
٤. برای هر گره از دومین گره به بعد، مراحل زیر را انجام می‌دهیم:
   - بررسی تمام گره‌های مجاور به u و در صورتی که رنگی به آن‌ها اختصاص داده شده باشد، آن رنگ را در آرایه available به عنوان رنگ موجود علامت می‌زنیم.
   - در آرایه available ، اولین رنگ موجود را پیدا کرده و به عنوان رنگ گره u اختصاص می‌دهیم.
   - مقادیر آرایه available را برای مرحله‌ی بعدی بازنشانی می‌کنیم.
۵. در نهایت، نتیجه به صورت "Vertex u ---> Color c" چاپ می‌شود که نشان‌دهنده‌ی رنگ اختصاص داده شده به هر گره است.

تحلیل مرتبه زمانی

- مرحله‌ی ۲ تنها یک عملیات است و زمان O(1) را دارد.
- مرحله‌ی ۳ نیز یک عملیات است و زمان O(V) را دارد.
- مرحله‌ی ٤ شامل دو حلقه است. حلقه بیرونی V-1 بار تکرار می‌شود و حلقه درونی ممکن است تا V-1 بار تکرار شود. برای هر گره، در حد بدترین حالت، تعداد یال‌های آن گره است. بنابراین، تعداد کل عملیات‌های انجام شده در مرحله‌ی ٤ از مرتبه O(V + E) است.
- مرحله‌ی ۵ شامل یک حلقه است که V بار تکرار می‌شود و زمان O(V) را دارد.

[1] Graph Coloring Algorithm

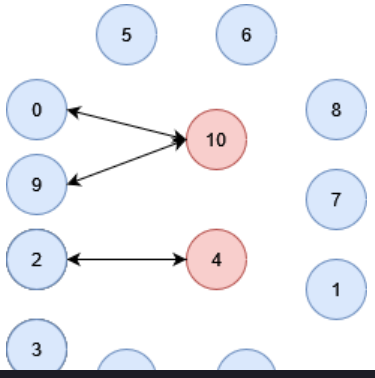
بنابراین، مجموع زمان اجرایی الگوریتم برابر است با O(V^2 + VE) ، که مرتبه‌ی زمانی خطی نسبت به تعداد گره‌ها و یال‌ها است.

تحلیل مرتبه حافظه

۱. آرایه result با طول V برای ذخیره رنگ‌های اختصاص داده شده به گره‌ها ایجاد می‌شود. این آرایه حافظه O(V) را اشغال می‌کند.
۲. آرایه available با طول V برای نشان دادن رنگ‌های موجود برای اختصاص به گره‌های مجاور تعریف می‌شود. این آرایه نیز حافظه O(V) را اشغال می‌کند.
۳. از مقدار حافظه برای متغیرهای محلی صرف نظر می‌کنیم زیرا فضای ثابت است و تأثیری در تحلیل حافظه ندارد.

بنابراین، مجموع حافظه مصرفی توسط الگوریتم شامل آرایه result و آرایه available است که در مجموع O(V) حافظه را اشغال می‌کنند.

نمونه‌ی ورودی



نمونه‌خروجی

```
Vertex 0 --->  Color 0
Vertex 1 --->  Color 0
Vertex 2 --->  Color 0
Vertex 3 --->  Color 0
Vertex 4 --->  Color 1
Vertex 5 --->  Color 0
Vertex 6 --->  Color 0
Vertex 7 --->  Color 0
Vertex 8 --->  Color 0
Vertex 9 --->  Color 0
Vertex 10 --->  Color 1
Vertex 11 --->  Color 0
Vertex 12 --->  Color 0
```

۲ - خروجی کد

فروش

جستجو براساس طرح نقشه

در این بخش از الگوریتم خاصی استفاده نشده و صرفا خانه های آرایه با یکدیگر مقایسه شده‌اند.

در ابتدا با انتخاب این گزینه از منو لیستی از فرش‌های موجود به کاربر نشان داده می‌شود. کاربر با انتخاب فرش مورد نظر خود، سه مورد از نزدیکترین فرش‌های مشابه به فرش مورد نظر کاربر، نمایش داده می‌شود.

برای این بخش از دو متد استفاده شده که آنهارا تحلیل خواهیم کرد.

تحلیل مرتبه زمانی

تابع differencePercentage :

- این تابع یک حلقه forبا طول carpet1.length دارد که بر اساس اندیس‌های آرایه carpet1حرکت می‌کند.
- در هر مرحله، یک شرط if اجرا می‌شود و در صورت برقراری آن، متغیر counter افزایش می‌یابد.
- در نهایت، مقدار متغیر counterبه عنوان خروجی تابع برگردانده می‌شود.

زمان اجرای این تابع به طول آرایه carpet1 بستگی دارد و به صورت مستقیم در طول حلقه for تعیین می‌شود. پس زمان اجرای این تابع برابر با O(carpet1.length) است.

تابع chooseMostSimilars :

- ابتدا ۳ متغیر imax1، imax2وimax3به همراه ۳ متغیر max1، max2وmax3را ایجاد می‌کند و به صفر مقداردهی اولیه می‌شوند.
- سپس یک آرایه به نام results با طول carpet2.length تعریف می‌شود.
- یک حلقه for با طول carpet2.length وجود دارد که بر اساس اندیس‌های آرایه carpet2 حرکت می‌کند.
- در هر مرحله، اگر مقدار اندیس فعلی برابر با nنباشد، تابع differencePercentage با پارامترهای carpet1وcarpet2 فراخوانی می‌شود و مقدار برگشتی آن در مکان متناظر آرایه resultsقرار می‌گیرد.
- سپس یک حلقه forدیگر با طول carpet2.length وجود دارد که بر اساس اندیس‌های آرایه results حرکت می‌کند.
- در هر مرحله، مقدار اندیس فعلی با متغیرهای max1، max2وmax3مقایسه می‌شود و در صورت بزرگ‌تر بودن، مقادیر متغیرها و متغیرهای مربوطه به اندیس‌ها به روزرسانی می‌شوند.
- در نهایت، یک آرایه answer تعریف می‌شود که شامل مقادیر imax3، imax2وimax1 است و به علاوه سه خروجی نمایش داده می‌شوند.
- در نهایت، آرایه answer به عنوان خروجی تابع برگردانده می‌شود.

زمان اجرای این تابع برابر با O(carpet2.length) است، زیرا دو حلقه for اصلی طول carpet2.length را پیمایش می‌کنند.

در نتیجه، زمان اجرای کل کد برابر با زمان اجرای تابع chooseMostSimilars است که O(carpet2.length) است.

تحلیل مرتبه حافظه

تابع differencePercentage :

- از نظر حافظه، تابع differencePercentage فقط یک متغیر counter را استفاده می‌کند که مقدار آن در طول حلقه تغییر می‌کند. بنابراین، حافظه مورد نیاز برای اجرای این تابع ثابت است و به مرتبه O(1) است.

تابع chooseMostSimilars :

- در این تابع، ابتدا ۶ متغیر به نام‌های imax1، imax2، imax3، max1، max2و max3 تعریف می‌شوند که در ابتدا به صفر مقداردهی اولیه می‌شوند. همچنین، یک آرایه به نامresults با طول carpet2.length تعریف می‌شود.
- بر اساس اندازه ورودی carpet2، تعداد متغیرها و طول آرایه results متغیر است.
- به طور کلی، می‌توان گفت تعداد متغیرها و حافظه مورد نیاز در این تابع به مرتبه ورودی carpet2.length است.

بنابراین، مرتبه حافظه کل کد برابر با O(carpet2.length) است.

خرید براساس میزان پول

در این بخش از مسئله از الگوریتم کوله پشتی[1] استفاده شده است. این مسئله به دو روش متفاوت پاسخ داده شده است:

- ثابت بودن مقدار بودجه برای خرید فرش
- ثابت بودن وزن کوله پشتی برای خرید فرش

به توضیح بخش ثابت بودن مقدار بودجه خواهیم پرداخت. در منوی اصلی برنامه پس از انتخاب این بخش لیستی از فرش‌های موجود در فروشگاه به نمایش در می‌آید و در مرحله‌ی بعدی مقدار بودجه از کاربر دریافت می‌شود و بیشترین تعداد فرش قابل خرید همراه‌با قیمت آن فرش به نمایش در می‌آید.

توضیحات الگوریتم

در این مسئله، فرض می‌شود که ما یک کوله پشتی با ظرفیت محدود و یک مجموعه از اشیاء با ارزش و حجم داریم. هدف شما انتخابی از اشیاء است که مقدار کل ارزش آنها را ماکزیمم کنید، در حالی که حجم کل اشیاء انتخاب شده بیشتر از ظرفیت کوله پشتی نشود. الگوریتم کوله پشتی به صورت یک الگوریتم برنامه‌ریزی پویا عمل می‌کند. ابتدا یک جدول به اندازه تعداد اشیاء و ظرفیت کوله پشتی ایجاد می‌شود. سپس در هر سلول از جدول، به ترتیب اشیاء را در نظر می‌گیریم و تصمیم می‌گیریم آیا آن اشیاء را به کوله پشتی اضافه کنیم یا نه. با استفاده از روابط و قوانین برنامه‌ریزی پویا، ارزش کل و حجم کل هر خانه از جدول محاسبه می‌شود.

در نهایت، با مقایسه مقادیر به دست آمده در سطر آخر جدول، مقدار بیشترین ارزش کل و شامل شدن اشیاء مربوطه در کوله پشتی برای بهینه‌سازی مشخص می‌شود.

تحلیل مرتبه زمانی

- مرتب کردن آرایه با استفاده از تابع ()sort در زمان O(n log n) انجام می‌شود، که n تعداد عناصر آرایه است.
- پیمایش آرایه با حلقه foreach نیز در زمان O(n) انجام می‌شود .بنابراین، زمان اجرای کل الگوریتم به طور کلی به O(n log n) خواهد بود، که n تعداد عناصر آرایه قیمت‌ها است.

تحلیل مرتبه حافظه

برای تحلیل مرتبه حافظه الگوریتم، نیازمند بررسی میزان فضایی است که توسط الگوریتم در هر لحظه اشغال می‌شود. از آرایه prices برای نگهداری قیمت‌ها استفاده می‌شود. اندازه این آرایه برابر با تعداد عناصر ورودی است و بسته به ورودی متغیر است. بنابراین، مصرف حافظه برای آرایه prices از O(n) خواهد بود.

در مجموع، مصرف حافظه الگوریتم متناسب با تعداد عناصر آرایه ورودی است و به صورت O(n) است، که nتعداد عناصر آرایه قیمت‌ها است.

[1] Knapsack Algorithm

نمونه ورودی

Carpet0:	value: 845	Weigh: 768	Carpet1:	value: 120	Weigh: 971
Carpet2:	value: 980	Weigh: 328	Carpet3:	value: 295	Weigh: 486
Carpet4:	value: 713	Weigh: 599	Carpet5	:value: 302	Weigh: 572
Carpet6:	value: 541	Weigh: 436	Carpet7:	value: 604	Weigh: 540
Carpet8:	value: 558	Weigh: 786	Carpet9:	value: 551	Weigh: 860

نمونه خروجی

```
Enter your maximum price
1300
You can buy these carpets.
Carpet 0 with price 120
Carpet 1 with price 295
Carpet 2 with price 302
Carpet 3 with price 541
You can buy  4
```

۳ - خروجی با بودجه ۱۳۰۰

مسیر یابی به نزدیک‌ترین فروشگاه

برای پاسخ گویی به مسئله‌ی مسیریابی به نزدیکترین فروشگاه، از الگوریتم دایکسترا[1] استفاده کردیم. ابتدا روش استفاده را توضیح و در ادامه به تحلیل کد می‌پردازیم. در منوی برنامه با انتخاب پیدا کردن بهترین مسیر به فروشگاه وارد این بخش می‌شویم. در این قسمت الگوریتمی برای ساخت گراف‌های تصادفی و پاسخ به آن گراف تصادفی و یک گراف شامل پنج راس و نه یال وجود دارد. در ابتدا گراف تولید و راسی که در آن وجود داریم را به عنوان مبدا از ورودی دریافت می‌کند، سپس از آن راس بهترین مسیرها را به تمام رئوس دیگر و بهترین مسیر را به فروشگاه‌ها نمایش خواهد داد.

این الگوریتم، یک الگوریتم برای پیدا کردن کوتاه‌ترین مسیر بین یک رأس مبدأ و سایر رئوس در یک گراف است. حلقه اصلی الگوریتم تا زمانی ادامه پیدا می‌کند که صف حاوی رئوسی که باید بررسی شوند خالی شود. زمان اجرای این حلقه بر اساس تعداد رئوس و یال‌های گراف تغییر می‌کند. دستورات داخل حلقه برای هر رأس فعلی، همسایگان آن را بررسی کرده و در صورتی که فاصله جدید کمتر از فاصله قبلی باشد، فاصله و رأس قبلی را به‌روزرسانی می‌کند. سپس همسایه مورد نظر را از صف حذف و با فاصله به‌روزرسانی شده به صف اضافه می‌کند. پس از اجرای حلقه، نزدیک‌ترین مسیرها و فاصله آنها به رئوس دیگر در گراف نمایش داده می‌شوند.

تحلیل مرتبه زمانی

برای تحلیل مرتبه‌ی زمانی تمام قسمت‌های اصلی کد را بررسی می‌کنیم:

۱. مقداردهی اولیه: این بخش شامل مقداردهی اولیه فاصله‌ها و صف است. مقداردهی اولیه فاصله‌ها از اندازه بی‌نهایت استفاده می‌کند که به‌طور مستقیم زمانی را مصرف نمی‌کند. اما مقداردهی اولیه صف با توجه به تعداد رئوس گراف، زمانی به مرتبه $O(V)$ است.(V نشان‌دهنده تعداد رئوس است.)
۲. حلقه اصلی: زمان اجرای این حلقه بستگی به تعداد رئوس و یال‌های گراف دارد. این حلقه برای هر رأس فعلی در صف، همسایگان آن را بررسی کرده و در صورت لزوم فاصله و رأس قبلی را به‌روزرسانی می‌کند. زمانی که همسایگان را بررسی می‌کند، به ازای هر یال، عملیاتی ثابت انجام می‌دهد. پس زمان اجرای این حلقه به مرتبه تعداد یال‌ها است.
۳. نمایش نزدیک‌ترین مسیرها: این بخش شامل حلقه‌ای است که برای هر رأس در گراف، مسیر کوتاه‌ترین فاصله و فاصله را نمایش می‌دهد. این حلقه نیز به ازای هر رأس، عملیاتی ثابت انجام می‌دهد. پس زمان اجرای این حلقه به مرتبه تعداد رئوس است.

با توجه به موارد فوق، زمان اجرای کل الگوریتم دایکسترا متناسب با مجموع تعداد رئوس و یال‌ها در گراف است. به طور خلاصه، زمان اجرای الگوریتم به صورت معمول به مرتبه $O((V + E)\log V)$ است، که V تعداد رئوس و E تعداد یال‌های گراف است.

[1] Dijkstra

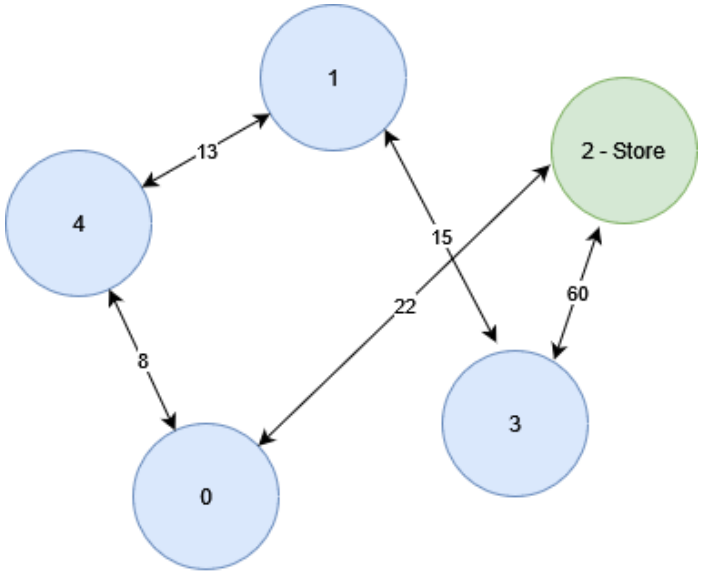
تحلیل مرتبه حافظه

برای تحلیل حافظه در این کد، نیازمند بررسی مصرف حافظه توسط متغیرها و ساختارهای داده مورد استفاده هستیم. در ادامه، مصرف حافظه برخی از اجزای این کد را بررسی می‌کنیم:

۱. Map<Vertex, Integer> distance: این نگاشت[1] برای ذخیره فاصله‌ها بین رئوس استفاده می‌شود. حافظه مصرفی این نگاشت به اندازه تعداد رئوس گراف است، بنابراین مصرف حافظه این بخش برابر با O(V)است.
۲. Map<Vertex, Vertex> previous: این نگاشت برای ذخیره رئوس قبلی در مسیرهای کوتاه‌ترین فاصله استفاده می‌شود. مصرف حافظه این نگاشت نیز به اندازه تعداد رئوس گراف است، بنابراین مصرف حافظه این بخش نیز برابر با O(V) است.
۳. PriorityQueue<Vertex> queue: این صف برای نگهداری رئوسی استفاده می‌شود که باید بررسی شوند. حافظه مصرفی این صف به اندازه تعداد رئوس گراف است و ممکن است در حالت بدترین (وقتی که تمام رئوس در صف قرار دارند) به O(V) برسد.
٤. List<Vertex> path: این لیست برای ذخیره مسیر کوتاه‌ترین فاصله استفاده می‌شود. مصرف حافظه این لیست برابر با طول مسیر کوتاه‌ترین فاصله است که در حالت بدترین می‌تواند به V برسد.

با توجه به موارد فوق، مجموع مصرف حافظه این الگوریتم به صورت کلی به اندازه تعداد رئوس گراف است و در حالت بدترین به O(V) می‌رسد.

نمونه‌ی ورودی



٤ - گراف موجود در فایل

[1] Map

نمونه‌ی خروجی

```
The store is located at 2
Select the origin vertex : 4
Vertex 0:   Path: 4,0       Distance: 8
Vertex 1:   Path: 4,1       Distance: 13
Vertex 2:   Path: 4,0,2     Distance: 30
Vertex 3:   Path: 4,1,3     Distance: 28
Vertex 4:   Path: 4     Distance: 0
Best Path from 4 to 2: Path: 4,0,2      Distance: 30
```

۵- خروجی برای راس چهارم

محتوای اضافه شده به پروژه

در این قسمت قابلیت‌های اضافه‌ی این پروژه را شرح خواهیم داد:

- یک الگوریتم اضافه برای مسئله‌ی محدودیت وزن فرش‌ها استفاده شده است که اضافه بر مطالب پروژه است.
- در بخش پیدا کردن بهترین مسیر، ابتدا بهترین مسیر از راس انتخابی به تمامی رئوس نمایش داده شده و در انتها بهترین مسیر به فروشگاه را نمایش خواهد داد.
- این پروژه کمترین میزان دریافت ورودی از کاربر را داراست. برای انجام این کار اکثر منابع و ورودی‌های مورد نیاز را به صورت تصادفی ساخته و در فایل‌های موجود در پروژه نوشته و موقع نیاز آنها را از فایل می‌خوانیم.
- به علت کمبود وقت از محتوای گرافیکی برنامه صرف نظر شد اما برای ساده نبودن صفحه‌ی کنسول از نوشتن با رنگ‌های متفاوت و قالبی زیبا به زیبایی پروژه کمک کرده ایم.
- در هر مرحله از روند انجام کار گزارش انجام روزانه نوشته شده است که محتوای مربوطه در فایل های پروژه موجود است.
- تمامی مراحل انجام کار به صورتی کاملا واضح، موجود و در مخزن گیت‌هاب[1] نگه داری می‌شود.
- از نظر کدنویسی سعی شده است که تمامی قوانین مربوط به کدنویسی تمیز، خوانا بودن و .... را رعایت کنیم.

[1] GitHub

منابع

1.Foundation of algorithms, 5th ed, 2015

2. https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm

3. https://www.geeksforgeeks.org/java-program-for-dynamic-programming-set-10-0-1-knapsack-problem

لینک دسترسی به محتوای کامل پروژه

https://github.com/amir-feiz/AlgorithmProject