

گزارشکار پروژه‌ی کارخانه فرشبافی

طراحی الگوریتم‌ها

استاد راهنما: دکتر پیمان ادیبی

اعضای گروه:

امیر فیض

محمدکاظم هرندی

بهار1402

فهرست مطالب

[طراحی 2](#_Toc136005215)

[طراحی فرش‌های جدید 2](#_Toc136005216)

[توضیحات الگوریتم 2](#_Toc136005217)

[تحلیل مرتبه زمانی 2](#_Toc136005218)

[تحلیل مرتبه حافظه 3](#_Toc136005219)

[نمونه‌ی ورودی 3](#_Toc136005220)

[نمونه‌خروجی 3](#_Toc136005221)

[فروش 5](#_Toc136005222)

[جستجو براساس طرح نقشه 5](#_Toc136005223)

[خرید براساس میزان پول 6](#_Toc136005224)

[مسیر یابی به نزدیک‌ترین فروشگاه 7](#_Toc136005225)

[تحلیل مرتبه زمانی 7](#_Toc136005226)

[تحلیل مرتبه حافظه 8](#_Toc136005227)

[نمونه‌ی ورودی 8](#_Toc136005228)

[نمونه‌ی خروجی 9](#_Toc136005229)

# طراحی

## طراحی فرش‌های جدید

برای پاسخگویی به این قسمت از پروژه از الگوریتم رنگ‌آمیزی گراف[[1]](#footnote-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

1. ابتدا یک آرایه‌ی نتیجه (result) با طول V تعریف می‌شود و تمام مقادیر آن به -1 اولیه می‌شوند. این آرایه، برای نگه‌داری رنگ‌های اختصاص داده شده به گره‌ها استفاده می‌شود.
2. رنگ اول به گره اول اختصاص داده می‌شود (result[0] = 0).
3. آرایه‌ای موقتی به نام available با طول V تعریف می‌شود و اولیه تمام مقادیر آن به true می‌شوند. این آرایه، برای نشان دادن رنگ‌های موجود برای اختصاص به گره‌های مجاور استفاده می‌شود.
4. 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.
5. 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 2 تنها یک عملیات است و زمان O(1) را دارد.
* مرحله‌ی 3 نیز یک عملیات است و زمان O(V) را دارد.
* مرحله‌ی 4 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 4 از مرتبه O(V + E) است.
* مرحله‌ی 5 شامل یک حلقه است که V بار تکرار می‌شود و زمان O(V) را دارد.

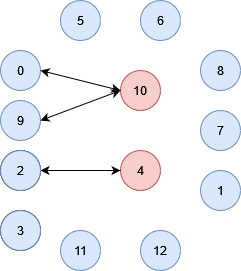
بنابراین، مجموع زمان اجرایی الگوریتم برابر است با O(V^2 + VE)، که مرتبه‌ی زمانی خطی نسبت به تعداد گره‌ها و یال‌ها است.

### تحلیل مرتبه حافظه

1. آرایه result با طول V برای ذخیره رنگ‌های اختصاص داده شده به گره‌ها ایجاد می‌شود. این آرایه حافظه O(V) را اشغال می‌کند.
2. 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.
3. 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



1 - وروردی چهارم کد

2 - خروجی چهارم کد

### نمونه‌خروجی

# فروش

## جستجو براساس طرح نقشه

## خرید براساس میزان پول

## مسیر یابی به نزدیک‌ترین فروشگاه

برای پاسخ گویی به مسئله‌ی مسیریابی به نزدیکترین فروشگاه، از الگوریتم دایکسترا[[2]](#footnote-2) 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:

1. 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 (.
2. 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.
3. 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.

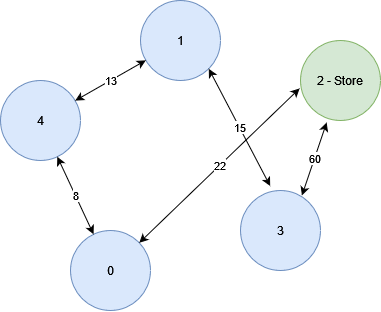
### تحلیل مرتبه حافظه

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:

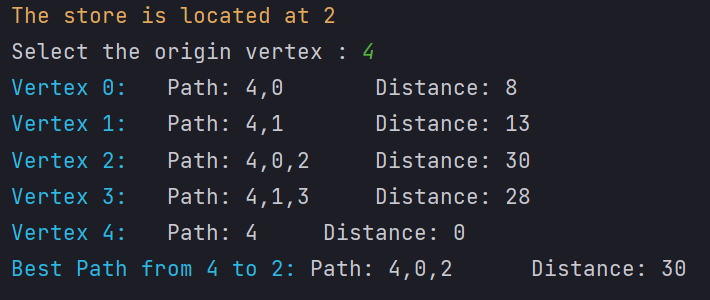
1. Map<Vertex, Integer> distance: این نگاشت [[3]](#footnote-3)برای ذخیره فاصله‌ها بین رئوس استفاده می‌شود. حافظه مصرفی این نگاشت به اندازه تعداد رئوس گراف است، بنابراین مصرف حافظه این بخش برابر با O(V) است.
2. Map<Vertex, Vertex> previous: این نگاشت برای ذخیره رئوس قبلی در مسیرهای کوتاه‌ترین فاصله استفاده می‌شود. مصرف حافظه این نگاشت نیز به اندازه تعداد رئوس گراف است، بنابراین مصرف حافظه این بخش نیز برابر با O(V) است.
3. 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.
4. 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



### نمونه‌ی خروجی



1. Graph Coloring Algorithm [↑](#footnote-ref-1)
2. Dijkstra [↑](#footnote-ref-2)
3. Map [↑](#footnote-ref-3)