طراحی الگوریتم: پیشرفتهای روشمند

حسین رادمرد

۲ اردیبهشت ۱۴۰۳

۱.۰ مسئله ۸۳ رنگ کردن یک گراف دو رنگی

در این مسئله، مانند موارد بعدی، حل مسئله هدف ماست و بهینه بودن جواب برای ما حائز اهمیت نیست. دراینجا، ما الگوریتمی را برای رنگ کردن گراف با دردست داشتن تنها دو رنگ، بررسی میکنیم. نسخهی کلیتر(رنگ آمیزی با هر تعداد رنگ) در مسئلهی ۵۹، صفحهی ۲۳۶ بررسی میشود. گرچه، نسخهی فعلی بسیار بهینهتر است. در ادامه، این بحث بسیار به موضوع ''گرافهای پرکاربرد: گرافهای دوبخشی'' مرتبط است.

گراف همبند بدون جهت G = N، (V)ناتهی) به ما داده شدهست. ما قصد داریم با رنگهای سیاه و سفید گراف را رنگ آمیزی کنیم به گونهای که هیچ دو راس همسایهای دارای رنگ یکسانی نباشند. چنین گرافی را گراف دورنگی مینامیم. الگوریتم حریصانه ای که ما قصد ساخت آن را برای این منظور داریم به پیمایش سطری گراف ها مرتبط است که آن را در ابتدا بررسی کردیم.

چنین گرافی را گراف دورنگی مینامیم. الگوریتم حریصانه ای که ما قصد ساخت آن را برای این منظور داریم به پیمایش سطری گراف ها مرتبط است که آن را در ابتدا بررسی کردیم.

پیمایش سطری گراف: یادآوری

معرفی

اول از همه، اجازه دهید مفاهیم ''فاصلهی میان دو راس'' و ''پیمایش سطری'' را برای گرافهای همبندِ بدون جهت تعریف کنیم.

تعریف ۱۰ (فاصلهی میان دو راس): در نظر میگیریم، G = N، (V) یک گراف همبند بدون جهت و s و s' دو راس این گراف باشند. طول کوتاهترین مسیر میان s و s' را فاصلهی میان s و s' گویند.

تعریف ۱۱ (جستجوی سطری): فرض کنیم G یک گراف همبند و بدون جهت و s یکی از راس های آن باشد. هر فرایندی که با افزایش فاصلهها از راس s با راس های گراف G برخورد میکند به عنوان پیمایش سطری گراف G از s شناخته میشود.

از نمودار (b) در شکل ۸.۷ صفحهی ۳۶۳، میتوانیم نتیجه بگیریم که لیست a، b، c، d، e، f، g، h با پیمایش سطری با شروع از راس a مطابقت دارد. و همین مطلب برای لیست a، c، b، d، e، h، f، g نیز صدق میکند.

تصویر ۸.۷ – یک مثال از گراف. تصویر (a) گرافی را نمایش میدهد که مثالی از حالت مسئله را نشان میدهد. تصویر (b) کوتاهترین مسیر راس a را تا هر راس گراف با خطوط پررنگ نشان میدهد. در تصویر (b)، عددی که در هر راس مشخص است در واقع فاصلهی آن راس تا راس a است.

حلقه بدون تغییر

ما علاقه داریم یک الگوریتم بدون تغییر بسازیم؛ یک الگوریتم حریصانه؛ و اینگونه خود را محدود میکنیم. برای جستجوی حلقه بدون تغییر، ادامه ی این ساز و کار به خواننده واگذار میشود. اکنون تصور میکنیم قسمتی از کار انجام شده ست(بخش ۳، صفحه ۹۳ را ببینید). به این ترتیب، برای یک گراف جزئی G′ = (N′، V′) (زیرگراف G القا شده با مجموعه رئوس ،N′ شامل رئوس ابتدایی)، لیستی تشکیل شده از پیمایش سطری G′ با شروع از s داریم. عموما* این لیست، CLOSE نامیده میشود. پیشرفت این روند شامل گستردن این لیست با افزودن رئوسی است که در CLOSE نیستند و تا جای ممکن به s نزدیکند.

از آنجایی که هر راسی که در CLOSE حضور نداشته باشد، یک کاندید احتمالی برای انقال به CLOSE است، در غیاب بقیه ی مفروضات، پیشرفت ممکن اما به همان نسبت هزینه بر است. پیشنهاد میکنیم که نسخه اول این ثابت را با اضافه کردن یک ساختمان داده بهبود ببخشید. ساختمان داده OPEN شامل تمام رئوسی ست که در CLOSE حضور نداشته و کاندید این موضوع هستند که همسایه حذاقل یکی از رئوس CLOSE هستند. بیشین*، OPEN به عنوان یک لیست اولویت با مدیریت برروی فاصله ی عناصرش از s بوجود می آید، این موضوع به این دلیل است که عنصری که باید به لیست CLOSE منتقل شود باید نزدیک ترین به s باشد.

بعدها میبینیم که نسخه ساده شده یک لیست اولویت نیز امکان پذیر است. برای ماندگاری این نسخه جدید از ثابت*، بهینه است که سر OPEN را به انتهای لیست CLOSE منتقل کنیم، و – به عنوان همتای تقویت ثابت** – برای معرفی همسایگان "جدید" عنصر منتقل شده به ،OPEN عنصرهایی که نه در OPEN نه در CLOSE هستند(این یک انتخاب حریصانه است).

با این حال، با توجه به عنصری e در ،OPEN پرسیدن مستقیم درباره وجود یا عدم وجود یکی از همسایگان آن در OPEN یا CLOSE می تواند پرهزینه باشد. راه حل بهتر شامل تقویت (جدید) با گزاره زیر است: از نظر رنگ آمیزی آینده، یک "رنگ" به هر راس گراف اختصاص می یابد، سفید اگر راس در OPEN یا CLOSE باشد، و در غیر این صورت خاکستری (در واقع، در اینجا، دو رنگ نقش مقادیر بولین را بازی می کنند). به شرطی که دسترسی مستقیم به رئوس امکان پذیر باشد، به روز رسانی OPEN آسان تر می شود. در پیشرفت، حفظ این مکمل ناوردا با رنگ آمیزی هر راسی که به OPEN منتقل می شود به رنگ سفید حاصل می شود.

بیایید به استراتژی مدیریت صف OPEN بازگردیم. آیا می توان به جای صف اولویت دار از یک صف ساده FIFO (نگاه کنید به بخش ۸.۱، صفحه ۳۲) استفاده کرد؟ در این صورت، مدیریت OPEN به طور قابل توجهی ساده می شود. برای انجام این کار، زمانی که رأس e از OPEN خارج می شود تا به CLOSE ملحق شود، همسایگان e که نامزد ورود به OPEN هستند باید فاصله ای بیشتر یا مساوی با تمام عناصر موجود در OPEN داشته باشند، که این امر امکان داشتن یک صف مرتب را فراهم می کند. این بدان معناست که اگر e در فاصله k از s باشد، سایر عناصر OPEN در فاصله k یا (k + ۱) از s قرار دارند، زیرا همسایگان "خاکستری" e در فاصله (k + ۱) از s قرار دارند. این فرض را به ناوردا اضافه می کنیم. خواننده دعوت می شود بررسی کند که آیا این موضوع با راه‌اندازی حلقه واقعاً برقرار شده است. همچنان باید ثابت کرد که با پیشرفت حفظ می شود. در نهایت، ما ناوردای زیر را پیشنهاد می کنیم که از چهار بند تشکیل شده است.

۱. CLOSE is the FIFO queue whose content represents a "breadth-first traversal" of the sub-graph of G induced by the vertices present in CLOSE.

۲. OPEN is the FIFO queue of the vertices that are neighbors of those present in CLOSE. The set intersection of OPEN and CLOSE is empty.

۳. If the head of OPEN contains a vertex whose distance to s is k، then other elements of OPEN are at the distance k or (k + ۱) from s.

in colored are OPEN in or CLOSE in either present vertices the G، graph the In .۴
grey. in are others white،

شکل ۹.۷، صفحه ۳۶۶، مراحل مختلف ''کاوش اولویت عرضی'' گراف شکل ۸.۷، صفحه ۳۶۳ را نشان می دهد. در هر نمودار شکل، رئوس موجود در CLOSE به صورت خطوط خاکستری ظاهر می شوند، رئوس موجود در OPEN با خطوط دوتایی هستند. فواصل فقط به عنوان یادآوری ذکر شده است، الگوریتم از آنها استفاده نمی کند. بیایید به عنوان مثال در مورد مرحله ای که از طرحواره (e) منجر به طرحواره (f) می شود، نظر بدهیم. در نمودار (e)، CLOSE شامل لیست ''کاوش اولویت عرضی'' زیرگراف القا شده توسط رئوس a، b، c و d است. راس e، سر صف OPEN، به انتهای CLOSE منتقل می شود. کدام همسایه های e قصد دارند به OPEN ملحق شوند؟ c و b از قبل در CLOSE هستند، ربطی به آنها ندارد. g از قبل در OPEN است، تحت تأثیر قرار نمی گیرد. تنها راس باقی مانده h است که به OPEN ملحق می شود و به رنگ سفید رنگ آمیزی می شود.

ساختمان داده ها

دو نوع ساختار داده در این الگوریتم استفاده می شود. نوع اول، صف های FIFO، در صفحه ۳۲ توضیح داده شده است. نوع دوم مربوط به نوعی ''رنگ آمیزی'' گراف ها می باشد.

ساختار داده‌ای به نام نمودار رنگی بدون جهت نیاز به رنگ‌آمیزی رئوس نمودار، دسترسی به رنگ آن‌ها و بررسی لیست همسایه‌ها وجود دارد، بنابراین تعاریف زیر (مجموعه رنگ‌ها فرض می‌شود که تعریف شده است): • رویه ColorGr(G، s، col): عملیاتی که راس s از G را با استفاده از رنگ col رنگ‌آمیزی می‌کند. • تابع WhichColorGr(G، s) نتیجه :Colors تابعی که رنگ راس s از G را برمی‌گرداند. • رویه OpenNeighborsGr(G، s): عملیاتی که بررسی لیست همسایه‌های راس s از G را آغاز می‌کند. • تابع EndListNeighborsGr(G، s) نتیجه :B تابعی که صحیح را تحویل می‌دهد اگر بررسی لیست همسایه‌های راس s از G به پایان رسیده باشد. • رویه ReadNeighborsGr(G، s، s′): عملیاتی که هویت راسی که ''زیر سرخوان'' لیست همسایه‌های s قرار دارد را در s′ ذخیره می‌کند، سپس سرخوان را یک موقعیت به جلو حرکت می‌دهد. برای این کاربرد، از نظر بیان الگوریتم و کارایی، بهترین تکمیل، نمایش با لیست مجاورت است (مشابه نمایش نمودارهای جهت‌دار در شکل ۳.۱ صفحه ۲۳ را ببینید). بنابراین نمودار به عنوان یک ''سه‌تایی'' تعریف می‌شود G = (N، V، R) (که در آن R متناظر با رنگ‌ها (در حال حاضر ''سفید'' و ''خاکستری'') است. الگوریتم علاوه بر نمودار G، این الگوریتم از متغیرهای cv (راس جاری) و neighb برای مرور لیست همسایه‌ها استفاده می‌کند.