دانشکده‌ی علوم ریاضی

آنالیز الگوریتم‌ها ۶ اسفند ۹۱

جلسه‌ی ۶: الگوریتم‌های پیمایش گراف

مدرّس: دکتر شهرام خزائی *نگارنده: شایان میرجعفری*

۱ مقدمه

در این بخش با الگوریتم‌های BFS و DFS آشنا خواهیم شد که برای پیمایش در یک گراف به کار می‌روند. در جستجوی اول سطح[۱] (BFS) همان‌گونه که از اسم آن پیداست رئوس گراف به صورت سطح به سطح بررسی می‌شوند و هر رأس پس از پردازش به صف[۲] اضافه می‌شود اما در جستجوی اول عمق[۳] (DFS) رئوس به صورت عمقی بررسی می‌شوند، بدین معنا که تا آن‌جا که ممکن است به عمق بیشتر و بیشتر می‌رود و در مواجهه با بن‌بست عقب‌گرد می‌کند؛ هر رأس پس از پردازش کامل وارد پشته[۴] می‌شود.

۲ جستجوی اول سطح

۱.۲ چگونه کار می‌کند؟

الگوریتم از ریشه شروع می‌کند (در گراف‌ها و یا درخت‌های بدون ریشه رأس دلخواهی به عنوان ریشه انتخاب می‌شود) و آن را در سطح صفر قرار می‌دهد. سپس در هر مرحله همه‌ی همسایه‌های رئوس آخرین سطح دیده شده را که تا به حال دیده نشده‌اند بازدید می‌کند و آنها را در سطح بعدی می‌گذارد. این فرایند زمانی متوقف می‌شود که همه‌ی همسایه‌های رئوس آخرین سطح قبلاً دیده شده باشند.

همچنین در مسائلی که حل مسئله مستلزم یافتن رأس هدف با خصوصیات مشخصی است، جستجوی سطح اول به صورت غیرخلاق عمل می‌کند. بدین ترتیب که الگوریتم هر دفعه همه‌ی همسایه‌های یک رأس را بازدید کرده و سپس به سراغ رأس بعدی می‌رود و بنابراین گراف سطح به سطح پیمایش خواهد شد. این روند تا جایی ادامه می‌یابد که رأس هدف پیدا شود و یا احتمالاً همه‌ی گراف پیمایش شود.

از نقطه نظر عملی، برای پیاده‌سازی این الگوریتم از صف استفاده می‌شود. بدین ترتیب که در ابتدا ریشه در صف قرار می‌گیرد. سپس هر دفعه عنصر ابتدای صف بیرون کشیده شده، همسایگانش بررسی شده و هر همسایه‌ای که تا به حال دیده نشده باشد به انتهای صف اضافه می‌شود.

[۱] breadth first search
[۲] queue
[۳] depth first search
[۴] stack

۲.۲ پیاده‌سازی

Algorithm 1 Algorithm: Breadth First Search

```
function BFS(graph G, vertex s)
    [assumes G = (V, E) and s ∈ V]
    mark s explored, other nodes unexplored
    set dist(s) = 0 and dist(v) = ∞ for v ≠ s
    initialite the queue data structure Q with s
    while Q ≠ ∅ do
        remove the first node of Q, call it v
        for each edge (v, w) do
            if w unexplored then
                mark w explored
                add w to Q (at the end)
                dist(w) = dist(v) + 1
```

قضیه ۱ *پس از پایان الگوریتم، رأس v به عنوان یک رأس پیمایش شده نشان می‌شود[۵]، اگر و فقط اگر از رأس s قابل دسترس باشد.*

برهان. همانطور که از از الگوریتم پیداست می‌دانیم که اگر v نشان شود آن‌گاه مسیری از رأس s به v وجود دارد ، حال با استقراء روی طول مسیر داریم:
فرض می‌کنیم مسیر $v_۰, \ldots, v_t$ به طوریکه $v_۰ = s$ وجود دارد که تمامی رئوس این مسیر نشان شده‌اند.
پایه‌ی استقراء به ازای $t = ۰$ طبق فرض بالا برقرار است.
فرض کنید که مسیری از s به $v_{t-۱}$ وجود دارد به طوریکه رأس $v_{t-۱}$ توسط یالی مانند $(v\prime, v_{t-۱})$ پیدا شده است که $v\prime$ قبل از $v_{t-۱}$ نشان شده است ، حال طبق فرض گفته شده رأس v_t به واسطه‌ی یالی که قبلاً سر دیگر آن نشان شده است پیدا می‌شود و نشان می‌شود که فرض می‌شود $v_t = v$ پس مسیری از s به v وجود دارد.

حال فرض می‌کنیم مسیری از s به v وجود دارد ، نشان می‌دهیم که پس از پایان الگوریتم رأس v نشان می‌شود.
فرض خلف کنید که چنین نباشد ، اما چون طبق مراحل اجرای الگوریتم پس از آن‌که هر رأس توسط رأسی که قبلاً در مسیر قرار گرفته است نشان می‌شود پس این با پایان یافتن الگوریتم در تناقض است. پس فرض خلف باطل بوده و رأس v نشان می‌شود.

■

قضیه ۲ *پس از پایان الگوریتم، مقدار $\mathrm{dist}(v)$ کمترین فاصله رأس v از رأس s است.*

برهان. اگر رأس v از رأس s قابل دسترس نباشد، طبق قضیه ۱ پس از پایان الگوریتم به عنوان یک رأس پیمایش شده نشان نمی‌شود. چون مقدار $\mathrm{dist}(\cdot)$ برای یک رأس فقط وقتی تغییر می‌کند که به عنوان یک رأس پیمایش شده نشان شود، بنابراین مقدار $\mathrm{dist}(v)$ تا انتهای الگوریتم همان مقدار اولیه، یعنی ∞، باقی می‌ماند.
حال فرض کنید رأس v از رأس s قابل دسترس باشد. مجموعه همه رأس‌هایی را که از s قابل دسترس هستند و کمترین فاصله آنها از s برابر i است با L_i نشان دهید. نشان می‌دهیم $v \in L_i$ اگر و فقط اگر $\mathrm{dist}(v) = i$.
با استقراء روی طول مسیر داریم:
پایه‌ی استقراء به ازای $i = ۱$ بدیهی است.
فرض می‌کنیم برای هر $i - ۱$ داریم: $v\prime \in L_{i-۱} \Longleftrightarrow \mathrm{dist}(v\prime) = i - ۱$

[۵] marked explored

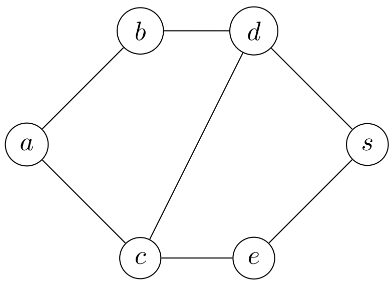
اگر در الگوریتم BFS ، v توسط v' به صف اضافه شود، آن‌گاه $v' \in L_{i-۱} \iff v \in L_i$
همچنین چون v' و v مجاورند پس $\mathrm{dist}(v) = i - ۱ + ۱ = i$ و حکم ثابت می‌شود.

■

۳.۲ پیچیدگی زمانی

هر رأس در صورتی وارد صف می‌شود که تا به حال دیده نشده باشد و بنابراین هر رأس قابل دسترسی از ریشه دقیقاً یک بار وارد صف خواهد شد. هر بار عمل ورود یک رأس به صف در $O(۱)$ انجام می‌شود و لذا با فرض همبند بودن گراف اعمال مربوط به صف $O(n)$ زمان را صرف خواهند کرد. همچنین هر یال در گراف بدون جهت دقیقاً دو بار و هر یال در گراف جهت‌دار دقیقاً یک بار پیمایش خواهند شد. بدین ترتیب با فرض همبندی گراف، پیچیدگی زمانی جستجوی سطح اول $O(n+m)$ خواهد بود. اگر گراف همبند نباشد، پیچیدگی $O(n_s + m_s)$ می‌باشد که n_s و m_s به‌ترتیب تعداد رئوس و تعداد یال‌هایی هستند که از s قابل دسترس می‌باشند.

مثال ۱ *جستجوی اول سطح از رأس* s



در این مثال داده‌ساختار صف به این‌صورت است که ابتدا رأس s نشان می‌شود و وارد صف می‌شود، سپس s از صف خارج شده و رأس‌های مجاور با آن در صورتی که نشان نشده باشند وارد صف می‌شوند که در اینجا رئوس d و e می‌باشند

e,d

سپس d از صف خارج می‌شود و همسایگان آن با شرط فوق وارد صف می‌شوند

c,b,e

این‌بار نوبت به رأس e میرسد و از صف خارج می‌شود اما در اینجا چون همسایه آن ، رأس c ، قبلاً توسط رأس دیگری نشان شده سراغ رأس بعدی می‌رویم

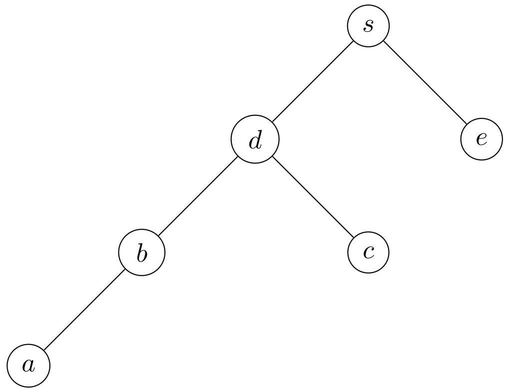
c,b

حال نوبت به رأس b می‌رسد و رأس a توسط آن وارد صف می‌شود

a,c

و این دو رأس باقی‌مانده در صف همانند رأس e که در بالا گفته شد فقط از صف خارج می‌شوند و صف خالی باقی می‌ماند.

درخت حاصل از این پیمایش



در این مثال با توجه به ساده بودن گراف هر رأس یک‌بار و هر یال ۲بار پیمایش می‌شوند.

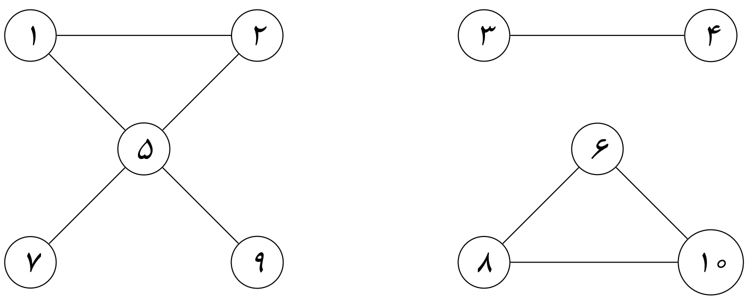
۴.۲ پیدا کردن مؤلفه‌های همبندی

مؤلفه‌های همبندی[۶] یک گراف را با استفاده از رابطه $\leftrightsquigarrow$ که به صورت زیر تعریف می‌شود تعریف می‌کنیم.

تعریف ۱ *در گراف بدون جهت G می‌گوییم رأس v با رأس w در رابطه است و با $v \leftrightsquigarrow w$ نشان می‌دهیم اگر و تنها اگر مسیری بین v و w در گراف G وجود داشته باشد.*

می‌توان نشان داد که رابطه $\leftrightsquigarrow$ یک رابطه هم ارزی است و بنابراین مجموعه رأس‌های گراف، V، را به کلاس‌های هم‌ارزی $V_1, \dots, V_t$ افراز می‌کند. همچنین می‌توان مجموعه یال‌های گراف، E، را به زیرمجموعه‌های $E_1, \dots, E_t$ افراز کرد به طوریکه E_i مجموعه همه یالهایی است که دو سر آن در V_i هستند. با این نمادگذاری، هر زیرگراف $G_i = (V_i, E_i)$ از G را یک مؤلفه همبندی گراف G می‌نامیم.

مثال ۲ *گراف زیر دارای سه مؤلفه همبندی است.*



۱.۴.۲ پیاده‌سازی

با اندکی تغییر در الگوریتم BFS می‌توان از آن به صورت زیر برای پیدا کردن مؤلفه‌های همبندی یک گراف استفاده کرد. پس از خاتمه الگوریتم تمام رأس‌هایی که در یک کلاس هم‌ارزی قرار دارند دارای $\text{leader}(\cdot)$ یکسان هستند. به وضوح پیچیدگی این الگوریتم از مرتبه $O(|V| + |E|)$ می‌باشد.

[۶]connected components

Algorithm 2 Algorithm: CONNECTED COMPONENTS

```
function CONNECTEDCOMPONENTS(graph G)
    [assumes G = (V, E)]
    initially all nodes unexplored
    for every s ∈ V do
        if s not yet explored then
            BFS(G, s)

function BFS(graph G, vertex s)
    [assumes G = (V, E) and s ∈ V]
    mark s explored
    set leader(s) = s
    initialite the queue data structure Q with s
    while Q ≠ ∅ do
        remove the first node of Q, call it v
        for each edge (v, w) do
            if w unexplored then
                mark w explored
                add w to Q (at the end)
                leader(w) = s
```

۳ جستجوی اول عمق

۱.۳ چگونه کار می‌کند؟

الگوریتم از ریشه شروع می‌کند (در گراف‌ها و یا درخت‌های بدون ریشه راس دلخواهی به عنوان ریشه انتخاب می‌شود) و در هر مرحله همسایه‌های رأس جاری را از طریق یال‌های خروجی آن رأس به ترتیب بررسی کرده و به محض روبه‌رو شدن با همسایه‌ای که قبلاً دیده نشده باشد، به صورت بازگشتی برای آن رأس به عنوان رأس جاری اجرا می‌شود. در صورتی که همه‌ی همسایه‌ها قبلاً دیده شده باشند، الگوریتم عقب‌گرد می‌کند و اجرای الگوریتم برای رأسی که از آن به رأس جاری رسیده‌ایم، ادامه می‌یابد. به عبارتی الگوریتم تا آنجا که ممکن است، به عمق بیشتر و بیشتر می‌رود و در مواجهه با بن بست عقب‌گرد می‌کند. این فرایند مادامی‌که همه‌ی رأس‌های قابل دستیابی از ریشه دیده شوند ادامه می‌یابد.

همچنین در مسائلی که حالات مختلف متناظر با رئوس یک گراف‌اند و حل مسئله مستلزم یافتن رأس هدف با خصوصیات مشخصی است، جستجوی عمق اول به صورت غیرخلاق عمل می‌کند. بدین‌ترتیب که هر دفعه الگوریتم به اولین همسایه‌ی یک رأس در گراف جستجو و در نتیجه هر دفعه به عمق بیشتر و بیشتر در گراف می‌رود تا به رأسی برسد که همه‌ی همسایگانش دیده شده‌اند که در حالت اخیر، الگوریتم به اولین رأسی بر می‌گردد که همسایه‌ی داشته باشد که هنوز دیده نشده باشد. این روند تا جایی ادامه می‌یابد که رأس هدف پیدا شود و یا احتمالاً همه‌ی گراف پیمایش شود.

از نقطه نظر عملی، برای اجرای الگوریتم، از یک پشته استفاده می‌شود. بدین ترتیب که هر بار با ورود به یک رأس دیده نشده، آن رأس را در پشته قرار می‌دهیم و هنگام عقب‌گرد رأس را از پشته حذف می‌کنیم. بنابراین در تمام طول الگوریتم اولین عنصر پشته رأس در حال بررسی است.

۲.۳ پیاده‌سازی

Algorithm 3 Algorithm: Depth First Search

function DFS(graph G, vertex s)
    let S be a stack data structure
    push s to S
    while $S \neq \emptyset$ **do**
        pop a node from S and call it v
        if v unexplored **then**
            mark v
            for each edge (v, w) **do**
                if w unexplored **then**
                    push w to S

این الگوریتم در زیر به صورت بازگشتی نیز ارائه می‌شود:

Algorithm 4 Algorithm: Recursive DFS

function RecursiveDFS(graph G, vertex s)
    [*assumes before the first call all nodes are unexplored*]
    mark s as explored
    for each edge (s, w) **do**
        RecursiveDFS(G, w)

۳.۳ پیچیدگی زمانی

مشخص است که الگوریتم، هر یال در گراف بدون‌جهت را دقیقاً دوبار (یک بار به بهنگام بررسی هر یک از دو انتها) و هر یال در گراف جهت‌دار را دقیقاً یک‌بار پیمایش می‌کند. همچنین هر رأس قابل دسترسی از ریشه دقیقاً یک‌بار بازدید خواهد شد. پس پیچیدگی زمانی جستجوی اول عمق نیز از $O(n_s + m_s)$ می‌باشد.

مثال ۳ *اجرای الگوریتم DFS روی گراف مثال ۱*

در اینجا ابتدا رأس s را در پشته push می‌کنیم سپس آن را pop و نشان کرده و رئوس مجاور با آن را به پشته push می‌کنیم

e
d

حال رأس e را که در بالای پشته قرار دارد را pop می‌کنیم و رئوس مجاور آن را در صورتی که نشان نشده بودند به پشته push می‌کنیم

c
d

حال به رأس c می‌رسیم و مراحل گفته شده را برای آن انجام می‌دهیم

a
d
d

حال به رأس a در بالای پشته می رسیم و رأس b را توسط آن به پشته $push$ می‌کنیم

b
d
d

در اینجا b در بالای پشته قرار دارد ، توسط آن رأس d را که هنوز نشان نشده را به پشته $push$ می‌کنیم

d
d
d

این‌بار رأس d در بالای پشته قرار دارد آن را pop و نشان کرده اما چون رأس‌های مجاور آن قبلاً نشان شده‌اند دیگر عنصری را به پشته $push$ نمی‌کنیم و تا خالی شدن پشته عمل pop را انجام می‌دهیم.

در آخر پشته خالی شده و همهٔ رئوس نشان می‌شوند.

۴.۳ مرتبه توپولوژیکی

تعریف ۲ *یک مرتبه توپولوژیکی برای گراف $G = (V, E)$ با n رأس یک تابع دو سویه $f : V \to \{1, ..., n\}$ است به طوری که $(v, w) \in E \Leftrightarrow f(v) < f(w)$.*

قضیه ۳ *یک گراف جهت‌دار دارای مرتبه توپولوژیکی است اگر و تنها اگر بدون دور باشد.*

تعریف ۳ *یک گراف بدون دور جهت‌دار، به اختصار DAG*[۷] *نامیده می‌شود. در یک DAG به هر رأس بدون یال ورودی، یک چشمه*[۸] *و به هر رأس بدون یال خروجی، یک چاهک*[۹] *می‌گوییم.*

لم ۴ *هر DAG دارای حداقل یک چشمه و حداقل یک چاهک است.*

برهان. در یک DAG مفروض فرض کنید که $v_1, \dots, v_t$ طولانی‌ترین مسر ممکن باشد. ادعا می‌کنیم که v_1 یک چشمه و v_t یک چاهک است. فرض خلف کنید که v_1 چشمه نباشد. بنابراین دارای یال ورودی (v_0, v_1) است که تناقض است زیرا در این صورت $v_0, v_1, \dots, v_t$ یک مسیر طولانی‌تر در گراف خواهد بود. لذا رأس v_1 یال ورودی ندارد و چشمه است. به همین ترتیب می‌توان استدلال کرد که v_t یک چاهک است. ■

در جلسه بعد الگوریتمی برای پیدا کردن مرتبه توپولوژیکی یک DAG ارائه خواهیم کرد.

[۷] directed acyclic graph
[۸] source
[۹] sink