به نام خدا

ساختمان داده‌ها و الگوریتم‌ها (۴۰۲۵۴)

دانشگاه صنعتی شریف — تمرین پنجم

مدرس: دکتر مهدی صفرنژاد — انتشار: ۹ خرداد ۱۴۰۱

الگوریتم‌های گراف

سؤال ۱. برای مسئله جستجوی مینیمم بازه‌ای با الگوریتم با پیش‌پردازش خطی و زمان پاسخگویی $O(1)$ نشان دهید چگونه پیدا کردن پاسخ پایین‌ترین جد مشترک در درخت کارتزین می‌تواند به پاسخ مینیمم بازه‌ای منجر شود. آیا برعکس این موضوع نیز برقرار است؟

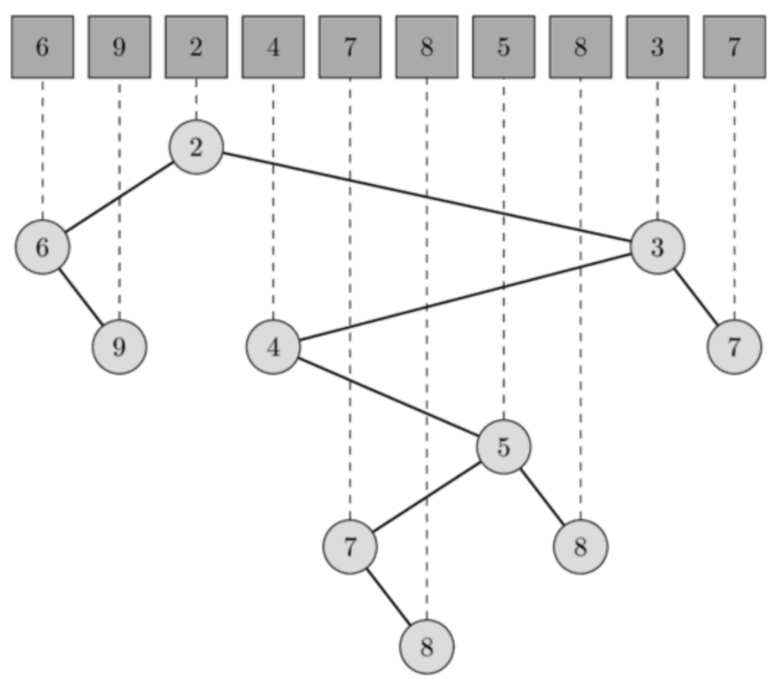
پاسخ:

فرض کنید $A = [1 \cdots N]$ آرایه اعدادی باشد که می‌خواهیم به RMQ های روی آن پاسخ دهیم.

برای این کار ابتدا درخت کارتزین آرایه A را می‌سازیم. نخست به تعریف درخت کارتزین و ویژگی‌های آن می‌پردازیم.

درخت کارتزین: درختی باینری است که دارای خاصیت هرم کمینه است، به گونه که پیمایش میان‌ترتیب (in-order traversal) آن خود آرایه اولیه را به همان ترتیبی که عناصرش قرار گرفته‌اند برگرداند.

در شکل زیر نمونه‌ای از یک آرایه و درخت کارتزین آن را می‌بینید.



حال کوئری $[l, r]$ را در نظر بگیرید. فرض کنید l' و r' به ترتیب رئوس متناظر با عناصر $A[l]$ و $A[r]$ در درخت کارتزین آرایه A باشند.

حال LCA رئوس l' و r' را درنظر بگیرید. میدانیم این رأس در پیمایش in-order بین فرزندان دو طرفش قرار می‌گیرد و از آنجایی که l' و r' در دو طرف آن هستند و درخت ما کارتزین است پس در آرایه اصلی LCA آن‌ها بینشان قرار گرفته‌است. از طرفی تمام رأس‌هایی که در in-order بین l' و r' می‌آیند باید فرزند یا نواده LCA باشد. پس به خاطر خاصیت هرم کمینه در درخت کارتزین،LCA از همه آن‌ها کوچک‌تر است. بنابراین مینیم بازه‌ای $[l, r]$، کوچک‌ترین جد مشترک آن‌ها در درخت کارتزین متناظر است.

با استفاده از الگوریتم زیر در زمان اجرای $O(N)$ درخت کارتزین آرایه A را می‌سازیم:

procedure CartesianTree($A[1 \cdots N]$)
//input sequence A with length N indexed from 1
// label the root node 1, with left child = right child = 0 (nonexistent)
    $root \leftarrow 1$
    $left[1] \leftarrow 0$
    $right[1] \leftarrow 0$
    $label[1] \leftarrow A[1]$
    for $i \in [2 \cdots N]$ **do**
        $last \leftarrow i - 1$
        $label[i] \leftarrow A[i]$
        $right[i] \leftarrow 0$
        while $label[last] > A[i] and last \neq root$ **do** last ← parent[last]
            if $label[last] > A[i]$ **then** // A[i] is the smallest element yet; make it new root
                $parent[root] \leftarrow i$
                $left[i] \leftarrow root$
                $root \leftarrow i$
            else if right[last] = 0 **then** // just insert it
                $right[last] \leftarrow i$
                $parent[i] \leftarrow last$
                $left[i] \leftarrow 0$
            else// reconfigure links
                $parent[right[last]] \leftarrow i$
                $left[i] \leftarrow right[last]$
                $right[last] \leftarrow i$
                $parent[i] \leftarrow last$

حال کافی است با استفاده از الگوریتمی مانند Farach-Colton and Bender پیش‌پردازش $O(N)$ و پاسخگویی $O(۱)$ با پیدا کردن LCA به پرسش RMQ پاسخ دهیم.

به همین شکل می‌توان با استفاده از پاسخ RMQ به پرسش LCA در یک درخت باینری جواب داد.

سؤال ۲. تن‌تن، خبرنگار جوان، به یک کشور پر از تبهکار سفر کرده است. در این کشور، شهرها از طریق جاده‌هایی با مسافت مشخص به یکدیگر متصل هستند و زمانی که رفتن از یک شهر به شهر دیگر طول می‌کشد برابر با مسافت جاده‌ی بین دو شهر است. تن‌تن که در شهر s قرار دارد، مدارکی بر علیه تبهکاران این کشور پیدا کرده است و می‌خواهد آنها را به دست دادستانی کل در شهر g برساند.

در این کشور T تبهکار در T شهر مختلف قرار دارند و می‌خواهند پیش از رسیدن تن‌تن به دادستانی او را بربایند. همچنین این تبهکاران در C تا از شهرها خودروهایی دارند که می‌تواند مسافت بین دو شهر را در نصف زمان عادی طی کند و اگر تبهکاران به این شهرها برسند می‌توانند برای ادامه‌ی مسیر از این خودروها استفاده کنند. در صورتی که در زمان رسیدن تن‌تن به یک شهر، حداقل یکی از تبهکاران پیش از او به آن شهر رسیده باشد، تن‌تن ربوده می‌شود.

اگر تن‌تن می‌تواند مستقل از مسیری که توسط تبهکاران طی می‌شود از دست آنها فرار کند و در زمان متناهی به شهر g برسد، کمترین مدت زمانی که برای این کار نیاز دارد و مسیری که باید طی کند را پیدا کنید و در غیر این صورت بگویید چنین مسیری وجود ندارد و تن‌تن ربوده می‌شود.

پاسخ:

فرض کنید کوتاه‌ترین فاصله تن‌تن تا مقصد از تمام کوتاه‌ترین فواصل دزدها تا مقصد کمتر باشد. حال فرض کنید تن‌تن در مسیر با کوتاه‌ترین فاصله به سمت مقصد حرکت می‌کند و یکی از دزدها می‌تواند در شهر X واقع در مسیر تن‌تن زودتر به تو برسد و او را بگیر. در این صورت اگر دزد از آن شهر در مسیری که تن‌تن به سمت مقصد حرکت می‌کرد راه خود را ادامه دهد، آنگاه فاصله‌اش تا مقصد از فاصله تن‌تن کمتر می‌شد(زیرا فاصله‌اش از X کمتر بوده است) که با فرض اولیه تناقض دارد. پس اگر اگر فاصله تن‌تن کوتاه‌تر باشد سالم به مقضد می‌رسد.

اگر فاصله یکی از دزدها کمتر باشد، می‌تواند زودتر به مقصد برسد و آن‌جا تن‌تن را بگیرد.

حال برای حل مسئله گراف متناظر را تشکیل می‌دهیم. یک بار از رأس متناظر با تن‌تن الگورینم دایسترا را اجرا می‌کنیم و کمترین فاصله تا مقصد را بدست می‌آوریم.

برای کمترین فاصله دزدها به دلیل وجود ماشین کار کمی پیچیده‌تر است. کاری که می‌نوانیم کنیم این است که گرافی به نصف وزن‌های گراف اولیه بسازیم و از رأس‌های دارای ماشین گراف دوم به گراف اول یال جهت‌دار با وزن صفر وصل کنیم و از مقصد در گراف دوم الگوریتم دایسترا را اجرا کنیم و هر گاه به رأس ماشین‌دار در گراف دوم رسیدیم، تنها به رأس‌های متصل به آن‌ها در گراف اول برویم. سپس در بین فواصل بدست آمده از دزدها کم‌ترین را پیدا کنیم.

االبته روش بهینه‌تر این است که به جای ساخت کامل گراف دوم تنها بخشی را بسازیم و با استفاده از داده‌ساختار صف حین اجرای الگوریتم دایسترا وجود گراف دوم را شبیه‌سازی کنیم.

سؤال ۳. مربع یک گراف جهتدار $G = (V, E)$ گراف $G^۲ = (V, E^۲)$ است؛ به گونه‌ای که $(u, v) \in E^۲$ اگر و تنها اگر G مسیری به طول حداکثر ۲ بین u و v را شامل شود. الگوریتمی بیهنه برای محاسبه $G^۲$ از G هم برای ماتریس مجاورت و هم برای لیست مجاورت G ارائه دهید و زمان اجرای آن را تحلیل کنید.

پاسخ:

فرض کنید ماتریس مجاورت A را داریم. سطر iام و ستون jام را در نظر بگیرید. اگر هم عنصر kام سطر i و هم عنصر kام ستون j هر دو یک باشند، از رأس i به رأس j مسیری به طول ۲ وجود دارد که از رأس k می‌گذرد. بنابراین اگر سطر i را در ستون j ضرب

داخلی کنیم،خاصل برابر با تعداد مسیرهای به طول ۲ از رأس i به رأس j خواهد بود.

بنابراین ماتریس A^2 ماتریسی است که درایه $[i,j]$ آن تعداد مسیرهای به طول ۲ از i به j است.

پس کافی است ماتریس A^2 را محاسبه کنیم و متناظر با درایه‌های ناصفر ماتریس‌های A و A^2 در گراف G^2 یال ایجاد کنیم. در حال حاضر بهینه‌ترین الگوریتم برای ضرب ماتریس Coppersmith-Windograd است که در زمان $O(|V|^{2.3728639})$ مربع ماتریس مجاورت گراف G را محاسبه می‌کند.

حال فرض کنید لیست مجاورت L را داریم.

ابتدا ترانهاده گراف جهت‌دار G را تعریف می‌کنیم: گراف ترانهاده گراف جهت‌دار G که با G^T نشان می‌دهیم، یک گراف جهت‌دار دیگر است با همان رئوس ولی یال‌هایی در جهت معکوس. به عبارت دیگر، اگر G شامل یال (u,v) باشد، ترانهاده آن شامل یال (v,u) است و برعکس.

حال از روی L گراف ترانهاده G را محاسبه می‌کنیم. ما ابتدا یک لیست مجاورت خالی از ترانهاده را حفظ نگه‌میداریم. سپس، هر لیست موجود در نمودار اصلی را اسکن می کنیم. اگر در لیست مربوط به راس v قرار داشته باشیم و u را به عنوان یک عنصر در لیست ببینیم، سپس یک عنصر از v را به لیست در گراف ترانهاده مربوط به راس u اضافه می کنیم. از آنجایی که این فقط به اسکن تمام لیست ها نیاز دارد، زمان اجرای آن $O(|V|+|E|)$ خواهد بود.

لیست مجاورت گراف G^T را با L^T نشان می‌دهیم. سپس، به لیست مجاورت اولیه خالی از G^2 عناصر را به این صورت اضافه می‌شود. همانطور که لیست هر رأس را اسکن می‌کنیم، مانند v ،و عنصری را می‌بینیم که به u می رود، u را به لیست مربوط به v اضافه می‌کنیم، اما u را نیز به لیست تمام عناصر در لیست v در G^T اضافه می‌کنیم.

از آنجایی که ممکن است در بدترین حالت پردازش هر یال O(|V|) زمان ببرد، در مجموع این روش در زمان $O(|E||V|+|V|)$ اجرا می‌شود.

سؤال ۴. گراف جهت‌دار $G=(V,E)$ را در نظر بگیرید. مجموعه SCC را مجموعه تمام مولفه‌های قویا همبند G تعریف می‌کنیم. حال گراف ساده‌ای را درنظر بگیرید که مجموعه رئوسش SCC باشد و بین دو رأس آن یک یال جهت‌دار وجود دارد،اگر و تنها اگر میان دو رأس موجود در این دو مولفه قویا همبند در گراف G یالی با جهت یکسان وجود داشته باشد. این گراف را گراف مولفه‌ای گراف G می‌نامیم.

الف) الگورتمی با پیچیدگی زمانی $O(|V|+|E|)$ ارائه دهید که گراف مولفه‌ای G محاسبه کند.

ب) گراف $G'=(V,E')$ را درنظر بگیرید که دارای ویژگی‌های زیر باشد:

—مولفه‌های همبندی یکسانی با G دارد.

—گراف مولفه‌ای یکسانی با G دارد.

—E' تا حد امکان کوچک باشد.(مینیمال باشد.)

الگوریتمی بهینه ارائه دهید که G' را محاسبه کند.

پاسخ:

۱.ابتدا مولفه‌های همبندی را بدست می‌آوریم. برای این کار می‌توان از الگوریتم Kosaraju استفاده کرد که در زمان $O(|V|+|E|)$ اجرا می‌شود. شبه‌کد این الگوریتم به صورت زیر است:

```
s ← None
order ← list[]
procedure DFSLoop(graph G(V, E), list seq)
    if seq is None then
        nodes ← G.V
    else
        nodes ← seq
    for each node ∈ nodes do
        if is not node.visited then
            s ← node
            DFS(G, node)

procedure DFS(graph G(V, E), node v)
    v.visited ← True
    v.leader ← s
    for each u ∈ G[v] do
        if is not u.visited then
            DFS(G, u)
        order ← [v] + order

procedure Kosaraju(graph G(V, E)
    DFSLoop(G^T, None)
    DFSLoop(G, order)
    leaders ← list[]
    for each nodeinG.V do
        leaders.add(node.leader)
```

برای هر رأس، یک متغیر SCC به آن می دهیم، به طوری که $v.SCC$ نشان دهنده مولفه قویا همبندی است که v در آن قرار دارد. سپس، برای هر یال (uv) در گراف اصلی، یک یال از $u.SCC$ به $v.SCC$ اضافه می‌کنیم، در صورتی که قبلاً وجود نداشته باشد. کل این فرآیند در زمان $O(|V| + |E|)$ انجام می‌شود.

۲. حال فرض کنید گراف مولفه‌ای G را ساخته‌ایم و به هر رأس گراف یک لیبل SCC داده‌ایم. همچنین یک لیست A از اعضای هر مولفه‌ای قویا همبند تشکیل داده‌ایم. به گونه‌ای که $A[i]$ لیست رأس‌های موجود در مولفه قویا همبند iام باشد.

حال یک بار dfs را روی گراف اجرا می‌کنیم و برای هر یالی که مشاهده می‌کنیم، بررسی می‌کنیم ایا دو مؤلفه همبندی را به هم وصل می‌کند یا خیر. اگر وصل نمی‌کرد آن را حذف می‌کنیم. همچنین اگر یال دیگر بود که قبل از آن مشاهده شده باشد

و همان دو مؤلفه را به هم وصل کند، آن را نگه می‌داریم و یال جدیدی که مشاهده کرده‌ایم را حذف می‌کنیم. این فرآیند در زمان $O(|V|+|E|)$ اجرا می‌شود. حال یال‌هایی که باقی‌مانده‌اند، یک مجموعه مینمال را تشکیل می‌دهند که مؤلفه‌های همبندی متفاوتی را به هم متصل می‌کنند. حال باید ابن یال‌ها را را به صورتی در مؤلفه‌های همبندی قرار دهیم که کمترین اندازه E' را داشته باشیم. کمترین تعداد یال لازم برای ساخت یک مؤلفه قویا همبند با n رأس، n یال است که تشکیل یک دور می‌دهند. برای هر مؤلفه همبندی فرض کنید $v_1, v_2, \cdots v_k$ رئوس موجود در آن باشد. این مجموعه‌ها را می‌توان از آرایه A بدست آورد. حال برای هر کدام از این مجموعه‌ها دور متشکل از این رئوس را تشکیل می‌دهیم. ابن فرآیند نیز در زمان $O(|V|+|E|)$ اجرا می‌شود.

سؤال ۵. در یک گراف جهت‌دار وزن‌دار $G=(V,E)$ برون‌مرکزی وزن‌دار $\epsilon(u)$ رأس $u \in V$ کوتاه‌ترین فاصله وزن‌دار تا دورترین رأس نسبت به آن است. به عبارتی $\epsilon(u) = \max\{\delta(u,v) | v \in V\}$. شعاع وزن‌دار $R(G)$ گراف وزن‌دار G، کوچک‌ترین برون‌مرکزی وزن‌دار تمام رأس‌های آن است. برای گراف G که دور جهت‌دار با وزن منفی ندارد،الگوریتمی با پیچیدگی زمانی $O(|V|^3)$ ارائه دهید که شعاع وزن‌دار آن را محاسبه کند.

پاسخ: یک راه ساده برای حل مسئله $|V|$ بار استفاده از الگوریتم بلمن‌ـفورد برای پیدا کردن کمترین فاصله میان تمام جفت رأس‌های گراف است. اما این فرآیند دارای زمان اجرای $O(|V| \times |V| \times |E|)$ که می‌تواند به $O(|V|^4)$ برسد. بنابراین در اینجا از الگوریتم دیگری به نام Johnson استفاده می‌کنیم.

procedure Johnson(graph G)
//create G'
 $G'.V \leftarrow G.V + s$
 $G'.E \leftarrow G.E$
 for each $u \in G.V$ **do**
  $G'.E \leftarrow G'.E \leftarrow (s,u)$
  $weight(s,u) \leftarrow 0$
 if Bellman-Ford(s) == $False$ **then return** "The input graph has a negative weight cycle"
 else
  for each vertex $v \in G'.V$ **do**
   $h(v) \leftarrow distance(s,v)$ computed by Bellman-Ford
  for edge $(u,v) \in G'.E$ **do**
   $weight'(u,v) \leftarrow weight(u,v) + h(u) - h(v)$
 $D \leftarrow newmatrixofdistancesinitializedtoinfinity$ **for** $vertex$u $\in G.V$ **do**
 run Dijkstra(G, weight', u) to compute distance'(u, v) for all v in G.V
 for each vertex $vinG.V$ **do**
  $D_{(u,v)} \leftarrow distance'(u,v) + h(v) - h(u)$
 return D

زمان اجرای این الگوریتم به دلیل یک بار اجرای بلمن‌ـفورد و $|V|$ بار اجرای دایسترا برابر است با $O(|V|^2 \log|V| + |V||E|)$.

که در بدترین حالت برابر است با $O(|V|^3)$.

حال میان تمام جفت رأس‌های گراف را داریم. برای هر رأس u در زمان $O(|V|)$ مقدار $\epsilon(u)$ را با ماکسیمم گرفتن از سطر u ام ماتریس D محاسبه می‌کنیم که در مجموع برای $|V|$ رأس در زمان $O(|V|^2)$ برای تمام رئوس بدست می‌آید. سپس در $O(|V|)$ از بین تمام $\epsilon(u)$ ها مینیمم می‌گیریم و $R(G)$ بدست می‌آید.

سؤال ۶. کیمیا به همراه خود m کوئری دارد که هر کدام از آنها یک مقدار $a_i \in \mathbb{N}$ دارند. او از سجاد می‌خواهد که یک آرایه‌ی nتایی تهیه کند و تا جایی که می‌تواند، کوئریهای کیمیا را در آرایه‌اش جا دهد. کیمیا دو شرط هم برای جا دادن کوئریها در آرایه دارد. اول این که در هر خانه‌ی آرایه‌ی سجاد، حداکثر یک کوئری قرار بگیرد، و دوم آن که اگر کوئری i در خانه‌ی jام آرایه‌ی سجاد قرار گرفته، حتما $a_i < j$ باشد. به سجاد کمک کنید تا با الگوریتمی با پیچیدگی زمانی $O(n + m\log n)$ بیشترین تعداد کوئری را در آرایه‌اش جا دهد.

پاسخ:

واضح است که بهتر است هر کوئری را در بیشینه خانه‌ی ممکن قرار داد، و تا زمانی که این ویژگی را داریم، فرقی نمی‌کند اگر یک کوئری را با کوئری دیگری که قبلا در آنجا بوده جابه‌جا کنیم. پس هر کوئری را در بیشینه جای ممکن در صورت امکان قرار می‌دهیم.

بازه‌های متوالی از کوئری‌های انجام شده را به صورت یک مجموعه می‌بینیم. همچنین، در نماینده مجموعه، کمینه مقدار موجود در مجموعه را نگه مي‌داریم.

برای هر کوئری جدید در صورتی که مجموعه‌ای برای a_i ساخته نشده باشد، این مجموعه را می‌سازیم. در صورتی که چنین مجموعه وجود داشته باشد، و کمینه آن m باشد، در صورت وجود، مجموعه $m - ۱$ را می‌سازیم و این کوئری را در خانه $m - ۱$ قرار می‌دهیم.

در ادامه مجموعه $m - ۱$ را در صورت وجود با مجموعه‌های $m - ۲$ و m اجتماع می‌گیریم تا تمام مجموعه‌ها بازه‌های متوالی پری را نشان دهند که در دو سرشان کوئری قرار نگرفته.

از n ساخت مجموعه و m اجتماع استفاده شده است. پس پیچیدگی زمانی از $O(n + m\log n)$ خواهد بود.

علاوه بر این می‌توان از انتهای آرایه به کمک یک صف شروع کرد، و کوئری‌هایی که مقدار آنها برابر خانه‌ی کنونی‌ست در صف قرار داد، سپس در صورت خالی نبودن صف، مقدار ابتدایی آن را در خانه آرایه قرار داد. پیچیدگی زمانی این الگوریتم از $O(n + m)$ خواهد بود.

سؤال ۷. گراف جهت‌دار وزن‌دار $G = (V, E)$ را درنظر بگیرید. فرض کنید این گراف همبند باشد. رأس مادر را رأسی می‌نامیم که از آن به تمام رأس‌های دیگر گراف راه باشد. الگوریتمی با پیچیدگی زمانی $O(|V| + |E|)$ ارائه دهید که در صورت وجود رأس مادر، یکی از آن‌ها را بیابد.

پاسخ:

روش بدیهی حل این مسئله اجرای DFS روی تمام رأس‌ها گراف است که در هر بار اجرا بررسی می‌کنیم می‌توان از یک رأس به تمام رأس‌های دیگر رفت یا خیر. اما زمان اجرای این روش از مرتبه $O(|V|(|V| + |E|))$ است.

اما می‌توان بهتر از این عمل کرد. از یک رأس دلخواه DFS را اجرا می‌کنیم و در هر مرحله که از یک رأس بکترک کردیم،آن رأس را در یک پشته push می‌کنیم. هر گاه که به رأس ابتدایی رسیدیم و دیگر نتوانستیم DFS را ادامه دهیم، از یک رأس دیگر که از آن

بازدید نکرده‌ایم مجددا DFS را به همین شکل اجرا می‌کنیم. و در این فرایند رئوس مشاهده شده را دیگر بازدید نمی‌کنیم. هر گاه از تمام راس‌های گراف بازدید شد فرآیند را متوقف می‌کنیم.

رأسی که در بالای استک قرار دارد یا رأس مادر است، یا در این گراف رأس مادر وجود ندارد.

برای اثبات گزاره فوق فرض کنید خلاف آن برقرار باشد، یعنی عنصر top پشته ایجاد شده رأس مادر نباشد،اما در بقیه پشته رأس M وجود داشته باشد که به تمام رأس‌های دیگر از جمله top مسیر دارد.(رأس مادر است)

حال فرض کنید از رأس top مسیری به M وجود نداشته باشد. از آنجایی که M در پشته از top پایین‌تر است،پس قبل از آن مشاهده شده است،اما چون M به top مسیر دارد و در اجرای DFS هنوز بازدید نشده است، پس وقتی از M فرآیند DFS را انجام می‌دهیم top قبل از M وارد پشته می‌شود. که تناقض است.

حال فرض کنید از top به M مسیر وجود داشته باشد، از آنجایی که M خود رأس مادر است، پس top هم رأس مادر خواهد بود. که با فرض اولیه در تناقض است؛ بنابراین ادعای ما ثابت می‌شود.

حال برای اینکه بررسی کنیم رأس top رأس مادر است، DFS را از آن اجرا می‌کنیم و اگر تمام رأس‌های گراف مشاهده شدند، رأس مادر خواهد بود،و در غیر این صورت خیر.

سؤال ۸. نشان دهید که با DFS می‌توان در گراف غیرجهت‌دار G تعداد مولفه‌های همبندی گراف یا k را به دست آورد و به هر رأس گراف G مانند v، عدد $v.cc$ را نسبت داد، به‌طوری‌که:

$$v.cc \in \mathbb{N}, ۱ \leq v.cc \leq k$$

$$v.cc = u.cc \iff u \; and \; v \; are \; in \; the \; same \; connected \; component$$

پاسخ:

الگوریتم DFS را به گونه‌ای تغییر می‌دهیم که برای ویزیت‌کردن رأس‌ها متد دومی تعریف می‌کنیم و تعداد مولفه‌های همبندی گراف را در متغیر $counter$ ذخیره می‌کنیم. الگوریتم DFS را به‌صورت زیر بازنویسی می‌کنیم:

```
procedure DFS(G)
    for each vertex u ∈ G.V do
        u.visited ← False
    counter ← 0
    for each vertex u ∈ G.V do
        if u.visited = false then
            counter ← counter + 1
            DFS-Visit(G, u, counter)

procedure DFS-Visit(G, u, counter)
    u.visited ← True
```

$u.cc \leftarrow counter$
for each vertex $v \in G.Adj[u]$ **do**
if $v.visited = False$ **then**
DFS-VISIT$(G, v, counter)$
=0

مشخص است که DFS متغیر $counter$ را هربار که DFS-Visit را صدا می‌کند افزایش می‌دهد. به این شکل همه رأس‌های یک درخت در جنگل DFS ما، مقدار cc یکسانی دارند. همچنین مقدار نهایی $counter$ برابر است با تعداد دفعاتی که DFS-Visit توسط DFS صدا شده است.

بنابراین، تنها باید نشان دهیم که رأس‌هایی که در یک‌بار صدازده‌شدن DFS-Visit توسط DFS ویزیت می‌شوند، لزوما در یک مولفه همبندی گراف G هستند و برعکس.

- همه رئوسی که در یک صدازده‌شدن DFS-Visit توسط DFS در یک مولفه همبندی‌اند: باتوجه‌به این‌که رئوسی که در یک صدازده‌شدن DFS-Visit توسط DFS ویزیت می‌شوند، از همسایگی دیگر رئوس انتخاب می‌شوند (حلقه for داخل DFS-Visit) پس بین همه آن‌ها مسیری وجود دارد و این رئوس متعلق به یک مولفه همبندی می‌باشند.
- همه رئوس یک مولفه همبندی در یک صدازده‌شدن DFS-Visit توسط DFS ویزیت می‌شوند: فرض کنید u اولین رأس در مولفه همبندی C باشد که ویزیت می‌شود، پس هنگامی که وارد DFS-Visit می‌شویم، بقیه رئوس مولفه همبندی C $visited$ برابر $False$ دارند. طبق $white\ path\ theory$ می‌دانیم که رأس v در درخت DFS گراف G فرزند u است اگر و تنها اگر از u به v مسیری شامل رأس‌های ویزیت‌نشده وجود داشته باشد. در این‌جا، این شرط برای u و تمام رأس‌های دیگر مولفه همبندی C برقرار است، بنابراین تمام رئوس یک مولفه همبندی درون یک درخت در جنگل DFS گراف G هستند و به‌همین‌ترتیب، در یک صدازده‌شدن DFS-Visit توسط DFS ویزیت می‌شوند.

سؤال ۹. فرض کنید یال (u, v) کم‌ترین وزن را بین تمام یال‌های گراف وزن‌دار غیرجهت‌دار G دارد. ثابت که (u, v) در MST گراف G حضور دارد.

پاسخ:

از برهان خلف استفاده می‌کنیم. فرض کنید T_1 در گراف G یک MST است و (u, v) در T_1 وجود ندارد. نشان می‌دهیم که T_1 نمی‌تواند MST باشد و درنتیجه MST گراف G در خود (u, v) را دارد.

به T_1 یال (u, v) را اضافه می‌کنیم. می‌دانیم در T_1 بین u و v مسیری وجود داشته است (چون T_1 یک $spanning\ tree$ است) و این‌که این مسیر (u, v) نیست (چون T_1 شامل (u, v) نیست) پس در گراف جدید، بین u و v دو مسیر وجود دارد. به‌سادگی نتیجه می‌گیریم در گراف جدید دوری شامل (u, v) وجود دارد. حال یالی از این دور که بیش‌ترین وزن را دارد را e می‌نامیم و حذف می‌کنیم. می‌دانیم این یال نمی‌تواند (u, v) باشد، زیرا این یال میان همه یال‌های گراف G کم‌ترین وزن را دارد. درخت جدید را T_2 می‌نامیم. می‌دانیم چون یال را از یک دور گراف حذف کرده‌ایم، هنوز درخت حاصل یک $spanning\ tree$ است. حال می‌نویسیم:

$$w(T_2) = w(T_1) + w((u, v)) - w(e),\ (u, v)\ is\ minimum\ edge\ in\ G \longrightarrow w(T_2) < w(T_1)$$

درحالی‌که T_1 باید MST گراف G باشد. این یعنی این‌که به تناقض رسیده‌ایم و MST گراف G باید رأس (u, v) را در خود داشته باشد.

سؤال ۱۰. در گراف وزن‌دار جهت‌دار $G = (V, E)$ که دوری با وزن منفی ندارد، کوتاه‌ترین مسیر هر رأس $v \in V$ از رأس *source* را پیدا می‌کنیم. فرض کنید ماکسیمم طول این مسیرها را برابر m قرار می‌دهیم. الگوریتم بلمن‌ــفورد را طوری تغییر دهید که تنها $m + 1$ بار iteration داشته باشد، حتی اگر مقدار m در ابتدا برای ما مشخص نباشد.

پاسخ:

حال که بیش‌ترین تعداد یال در مسیرها از رأس *source* برابر m است، طبق خاصیت *path relaxation* الگوریتم بلمن‌ــفورد، می‌دانیم که پس از m بار iteration، دیگر تمام مقادیر *distance* رئوس، به‌درستی محاسبه شده‌اند و دیگر تغییر نخواهند کرد. بنابراین با اضافه‌کردن یک متغیر به‌نام *changes* پیش از هر iteration بررسی می‌کنیم که آیا در iteration قبلی، تغییری ایجاد شد یا خیر. پس الگوریتم را به‌شکل زیر پیاده می‌کنیم:

procedure BELLMAN-FORD(G, w, s)
  $changes \leftarrow True$
  while $changes = True$ **do**
    $changes \leftarrow False$
    for each edge $(u, v) \in G.E$ **do** RELAX(u, v, w)

procedure RELAX(u, v, w)
  if $v.distance > u.distance + w(u, v)$ **then**
    $v.distance \leftarrow u.distance + w(u, v)$
    $changes \leftarrow True$

دقت کنید باتوجه‌به این‌که در گراف G دوری با وزن منفی وجود ندارد، می‌دانیم که هنگامی که در یک iteration در مقادیر *distance* رئوس گراف تغییری رخ ندهد، در iteration بعدی نیز رخ نخواهد داد.

سؤال ۱۱. فرض کنید $G = (V, E, w)$ گرافی وزن‌دار و جهت‌دار است. فرض کنید $c : V \longrightarrow \{r, g, b\}$ یک تابع است که به هر رأس مانند v، رنگ قرمز، سبز یا آبی را نسبت می‌دهد، اگر c یک رنگ باشد، V_c و E_c را به‌صورت زیر تعریف می‌کنیم:

- V_c مجموعه تمام رأس‌هایی است که رنگ c را دارند. یعنی $V_c = \{v \in V \mid c(v) = c\}$.
- E_c مجموعه یال‌هایی است که از رأس‌های V_c خارج می‌شوند. یعنی $E_c = \{(u, v) \in E \mid u \in V_c\}$.

فرض کنید گراف G و رنگ‌آمیزی c از خصوصیات زیر پیروی می‌کنند:

۱. تمام یال‌های گراف یا بین دو رأس هم‌رنگ‌اند یا رأسی قرمز را به رأسی سبز وصل می‌کنند، یا رأسی سبز را به رأسی آبی متصل می‌کنند.
۲. $|V_r| = |E_r| = O(|V|)$ و وزن همه یال‌های E_r برابر عدد طبیعی w_r است.
۳. $|V_g| = |E_g| = O\left(|V|^{0.99}\right)$ و وزن همه یال‌های E_g اعداد صحیح و نامنفی است.

۴. $|V_b| = |E_b| = O\left(\sqrt{|V|}\right)$ و وزن همه یال‌های E_b اعداد صحیح مثبت یا منفی است.

با داشتن گراف G، رنگ‌آمیزی c، رأس $s \in V_r$ و رأس $t \in V_b$، الگوریتمی با پیچیدگی زمانی $O\left(|V|\right)$ ارائه دهید که طول کوتاه‌ترین مسیر s به t را محاسبه کند.

پاسخ:

هر مسیر از s به t از سه بخش مجزا تشکیل شده‌است:

۱. یک مسیر، شامل یال‌هایی از E_r

۲. یک مسیر، شامل یال‌هایی از E_g

۳. یک مسیر (شاید خالی) شامل یال‌هایی از E_b

پس ما کوچک‌ترین مسیرها در گراف G را به‌صورت افزایشی محاسبه می‌کنیم، ابتدا با استفاده از یال‌های E_r سپس با یال‌هایی از E_g و درنهایت بابررسی یال‌های E_b.

قدم اول: گراف غیروزن‌دار $G' = (V', E')$ را می‌سازیم، به‌صورتی که $E' = E_r$ و V' شامل تمام رأس‌هایی است که در یال‌های E_r حضور دارند؛ یعنی $V' = \bigcup_{(u,v)\in E_r} \{u, v\}$. می‌دانیم که رأس‌های V' در مجموعه‌های V_r و V_g حضور دارند. برروی گراف G' با آغاز از رأس s یک BFS اجرا می‌کنیم تا فاصله‌ها بدون‌درنظرگرفتن وزن را بیابیم. حال کم‌ترین فاصله رأس s از هر رأس سبزرنگ g که در V' عضو باشد را از رابطه $\delta(s, g) = (g.distance)w_r$ به دست می‌آوریم. می‌دانیم اندازه G' از $O(|V|)$ است، بنابراین این قدم از پیچیدگی زمانی $O(|V|)$ است.

قدم دوم: حال گراف وزن‌دار $G'' = (V'', E'')$ را می‌سازیم. این گراف از دو بخش (متصل‌به‌هم)تشکیل شده‌است: ۱) رأس s و رأس‌های سبزرنگ $V_g \cap V'$ به‌همراه یال‌های وزن‌داری که در قدم اول محاسبه کرده‌ایم و ۲) یال‌های E_g و رأس‌های V_g. حال، تمام وزن‌ها در G'' مثبت‌اند بنابراین الگوریتم دایکسترا را اجرا می‌کنیم و به این شکل، کوتاه‌ترین فاصله‌ها در گراف G'' را نیز به دست می‌آوریم. به این ترتیب، فاصله محاسبه‌شده s از رئوس آبی‌رنگ $V'' \cap V_b$ برابر است با طول کوتاه‌ترین مسیر از s تا این رئوس، به‌شرطی‌که این مسیرها تنها از رئوس قرمز و سبز بگذرند. می‌دانیم اندازه G'' از $O(1 + |V_g| + |E_g|) = O(|V|^{0.99})$ است، بنابراین این قدم از پیچیدگی زمانی $O(|V|^{0.99} \log |V|^{0.99}) = O(|V|)$ است.

قدم سوم: حال گراف وزن‌دار $G''' = (V''', E''')$ را می‌سازیم. این گراف از دو بخش (متصل‌به‌هم) تشکیل شده‌است: ۱) رأس s و رأس‌های آبی‌رنگ $V'' \cap V_b$ به‌همراه یال‌های وزن‌داری که در قدم دوم محاسبه کرده‌ایم و ۲) یال‌های E_b و رأس‌های V_b. وزن‌های یال‌های G''' می‌توانند مثبت یا منفی باشند، بنابراین الگوریتم بلمن‌ــ‌فورد را با رأس مبدأ s روی گراف G''' اجرا می‌کنیم. حال، فاصله‌ای که از s تا t به دست می‌آوریم، برابر است با کوتاه‌ترین فاصله s تا t یا همان خواسته مسأله. می‌دانیم اندازه G''' از $O(1 + |V_b| + |E_b|) = O(\sqrt{|V|})$ است، بنابراین این قدم از پیچیدگی زمانی $O(\sqrt{|V|}\sqrt{|V|}) = O(|V|)$ است.

هر سه قدم این الگوریتم از $O(|V|)$ بوده‌اند، بنابراین کل الگوریتم نیز از $O(|V|)$ است.

سؤال ۱۲. قطر درخت $T = (V, E)$ را به‌صورت $max_{u,v\in V}\delta\left(u, v\right)$ تعریف می‌کنیم؛ یعنی ماکسیمم فاصله بین رأس‌های T. با فرض یکتابودن قطر، الگوریتمی بهینه برای محاسبه قطر درخت T ارائه دهید و پیچیدگی زمانی آن را تحلیل کنید.

پاسخ:

فرض کنید u و v دو طرف طولانی‌ترین مسیر درخت T باشند. فرض کنید s هر رأسی از درخت T باشد. نشان می‌دهیم که یک BFS که از s آغاز شود، رأس u یا v یا هردوی آن‌ها را به‌عنوان دورترین رأس T از رأس s نشان خواهد داد.

برای اثبات این مسأله از برهان خلف استفاده می‌کنیم. فرض کنید نتیجه اجرای BFS با شروع از s، رأس x را به‌عنوان دورترین رأس از s نشان می‌دهد و $x \neq u, v$. حال سه حالت مختلف را بررسی می‌کنیم:

۱. حالتی که s روی مسیر (u, v) باشد:

در این صورت دو مسیر (s, u) و (s, v) را در نظر می‌گیریم. می‌دانیم که $\delta(s, x) > \delta(s, u), \delta(s, v)$. بدون کاستن از کلیت مسأله فرض می‌کنیم $\delta(s, u) > \delta(s, v)$. دراین‌صورت مشاهده می‌کنیم که:

$$\delta(s, x) + \delta(s, u) > \delta(s, u) + \delta(s, v) \longrightarrow \delta(x, u) > \delta(u, v)$$

که این با فرض قطربودن (u, v) درتناقض است.

۲. حالتی که s روی مسیر (u, v) نباشد، اما مسیر (s, x) با مسیر (u, v) حداقل یک نقطه برخورد داشته باشد:

دراین‌صورت یکی از رئوس برخورد این دو مسیر را c می‌نامیم. حالا با منطقی مشابه منطق حالت اول پیش می‌رویم و بدون کاستن از کلیت مسأله فرض می‌کنیم $\delta(s, u) > \delta(s, v)$:

$$\delta(s, x) > \delta(s, v) \longrightarrow \delta(s, c) + \delta(c, x) > \delta(s, c) + \delta(c, v) \longrightarrow \delta(c, x) > \delta(c, v)$$

حال به شرایطی مانند حالت اول رسیدیم؛ پس به‌سادگی می‌نویسیم:

$$\delta(c, x) + \delta(c, u) > \delta(c, u) + \delta(c, v) \longrightarrow \delta(x, u) > \delta(u, v)$$

که این با فرض قطربودن (u, v) درتناقض است.

۳. حالتی که مسیر (s, x) با (u, v) نقطه برخوردی نداشته باشد:

رأس‌های a و b را به‌ترتیب روی مسیرهای (u, v) و (s, x) طوری انتخاب می‌کنیم که $\delta(a, b)$ مینیمم باشد. رأس a روی مسیر (u, v) است بنابراین در حالت اول اثباتمان صدق می‌کند. یعنی دورترین رأس درخت T از a یکی از رئوس u و v است. بدون کاستن از کلیت مسأله، فرض می‌کنیم این رأس u است. پس می‌توانیم بنویسیم $\delta(a, u) > \delta(a, x)$.
از طرف دیگر، می‌دانیم که دورترین رأس از s همان x بوده‌است پس می‌نویسیم:

$$\delta(s, x) > \delta(s, u) \longrightarrow \delta(s, b) + \delta(b, x) > \delta(s, b) + \delta(b, u) \longrightarrow \delta(b, x) > \delta(b, u)$$

از طرفی ما a و b را روی (u, v) و (s, x) به‌طوری انتخاب کرده‌بودیم که $\delta(a, b)$ مینیمم باشد؛ بنابراین می‌توانیم بنویسیم:

$$\delta(b, x) > \delta(b, u) \longrightarrow \delta(b, x) + \delta(a, b) > \delta(b, u) - \delta(a, b) \longrightarrow \delta(a, x) > \delta(a, u)$$

که این با رابطه $\delta(a, u) > \delta(a, x)$ که بالاتر به دست آورده‌ایم، درتناقض است.

پس یک BFS از هر رأسی در درخت T به ما u یا v را نشان خواهد داد. بدیهی‌ست که با تکرار BFS این‌بار، از رأس u یا v دورترین رأس به دست می‌آید که همان سر دیگر قطر است؛ زیرا اگر دورترین رأس رأس دیگری باشد، مسیری طولانی‌تر از قطر درخت T به دست می‌آید که با تعریف قطر درخت T درتناقض است.

می‌دانیم هر BFS از $O(|V|)$ است پس کل الگوریتم ما نیز از $O(۲|V|)$ یا همان $O(|V|)$ است.

موفق باشید