دانشگاه صنعتی شریف
دانشکده ریاضی

گزارش سمینار مباحث پیشرفته در الگوریتم‌ها

عنوان:

الگوریتم Baruvka برای یافتن درخت پوشای کمینه

نگارش:
علی توسلی
۴۰۲۲۰۱۰۹۹

استاد درس:
دکتر زارعی

بهمن ۱۴۰۲

فهرست مطالب

۱- پیشگفتار

الگوریتم Baruvka [۱] قدیمی‌ترین الگوریتم یافتن درخت پوشای کمینه می‌باشد که اولین بار در سال ۱۹۲۶ منتشر شد که روشی بود برای ساختن یک شبکه برق بهینه در شهر مارویای جمهوری چک. در سال ۱۹۳۸ این الگوریتم توسط Choquet بازیابی شد. این الگوریتم چند بار دیگر در سال‌های مختلف بازیابی شد که آخرین بار آن در سال ۱۹۶۵ بود توسط Sollin که در نهایت هم این الگوریتم به همین نام معروف شد.

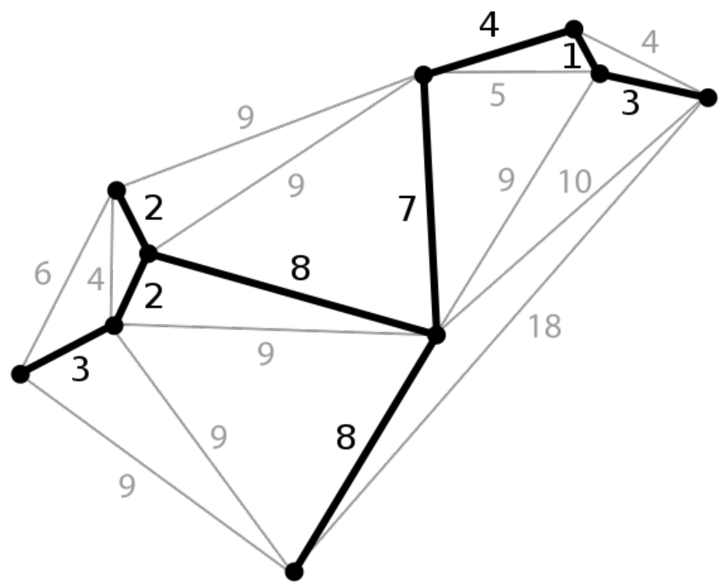
از کاربردهای این الگوریتم می‌توان به کاربرد آن در حوزه پردازش موازی اشاره کرد.

۲- شرح مساله

مساله یافتن درخت پوشای کمینه از مسائل معروف و مهم در علوم کامپیوتر می‌باشد. معروف‌ترین الگوریتم‌های این مساله، Kruskal [۲] و Prim [۳] می‌باشند.

مساله به این صورت است که یک گراف وزن‌دار بدون جهت داده شده. به دنبال یافتن یک زیردرخت از این گراف هستیم که مجموع وزن یال‌های آن بین همه درخت‌های ممکن کمینه باشد. می‌توان گراف را ناهمبند در نظر گرفت، در این صورت به دنبال یافتن یک جنگل پوشای کمینه هستیم.

شکل زیر درخت پوشای کمینه برای یک گراف ساده همبند را نشان می‌دهد. یال‌های پررنگ یال‌های این درخت هستند:



۳- الگوریتم

فرض کنید V مجموعه راس‌های گراف باشد و E مجموعه یال‌های گراف.

الگوریتم به این صورت عمل می‌کند که ابتدا هر راس را در مولفه همبندی خودش تعریف می‌کند. در ابتدا مجموعه یال‌های برگزیده شده برای درخت پاسخ مجموعه تهی است و به صورت دوره‌ای این مجموعه را کامل می‌کند. در هر مرحله، تعداد مولفه همبندی در گراف موجود است که این مولفه‌ها توسط یال‌های برگزیده شده تشکیل شده‌اند. در ابتدای هر مرحله، به صورت همزمان، از هر مولفه همبندی، کوچک‌ترین یالی که از آن خارج می‌شود را انتخاب می‌کند. سپس این یال‌ها را به مجموعه یال‌های برگزیده اضافه می‌کنیم. این کار را آنقدر انجام می‌دهیم تا به یک درخت پوشا برسیم.

در ادامه ثابت می‌کنیم این درخت پوشا، کمینه نیز می‌باشد.

۳-۱- جزئیات الگوریتم

هنگامی که برای یک مولفه، یال کمینه که از آن خارج می‌شود را انتخاب می‌کنیم، ممکن است چند یال باشند که این خاصیت را داشته باشند. حال سوالی که پیش می‌آید این است که از بین این‌ها، کدام یال را باید برگزید؟

پاسخ: به این صورت عمل می‌کنیم که مولفه‌ها را شماره‌گذاری می‌کنیم، اگر چند یال بودند که وزن کمینه را داشتند، آن یالی را انتخاب می‌کنیم که مولفه مقصد آن کوچک‌ترین شماره را داشته باشد. اگر چند یال با خاصیت فوق بودند، یالی را انتخاب می‌کنیم که شماره راس آن کمینه باشد.

۳-۲- تحلیل زمانی و حافظه الگوریتم

به هر عملیات پیدا کردن یال کمینه برای مولفه‌ها، یک گام می‌گوئیم.

لم ۱: تعداد گام‌ها از مرتبه $O(lg|v|)$ است.

برهان: در هر گام، تعداد مولفه‌ها حداقل نصف می‌شود (ممکن است در برخی موارد، کمتر از نصف هم بشود). از آنجا که در وضعیت ابتدایی تعداد $|V|$ مولفه داریم و در حالت انتهایی تنها یک مولفه، بنابراین حداکثر $O(lg|V|)$ گام خواهیم داشت.

برای هر گام (پیدا کردن کمینه یال خروجی از هر مولفه)، کافی است یک DFS بزنیم که از مرتبه $O(|V| + |E|)$ می‌باشد.

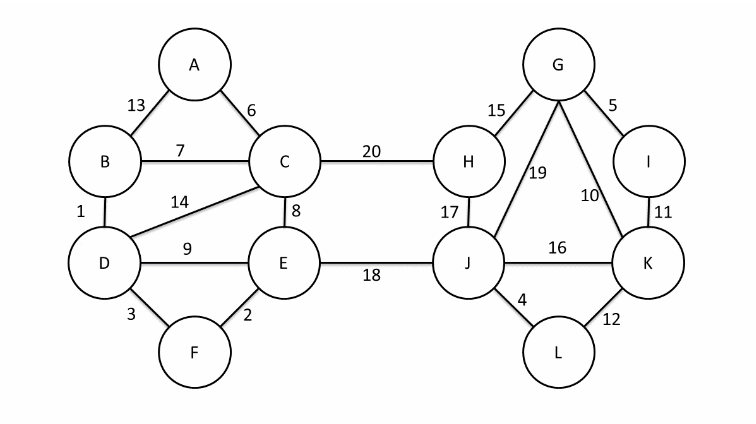
بنابراین در مجموع، پیچیدگی زمانی از مرتبه $O((|V| + |E|) \times lg|V|)$ خواهد بود.

از نظر حافظه، فقط یال‌ها را ذخیره کردیم و در هر مرحله برای هر مولفه یال کمینه آن را نگه داشتیم. بنابراین پیچیدگی حافظه برابر خواهد بود با: $O(|E| + |V|)$.

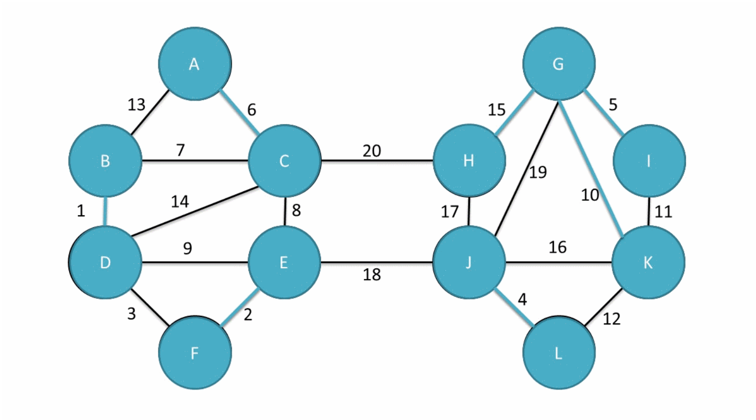
۴- مثالی از اجرای الگوریتم

گراف زیر را در نظر بگیرید. می‌خواهیم الگوریتم را روی آن اجرا کنیم. هر عکس نمایانگر یک گام از اجرای الگوریتم یعنی پیدا کردن کمینه یال برای هر مولفه می‌باشد.

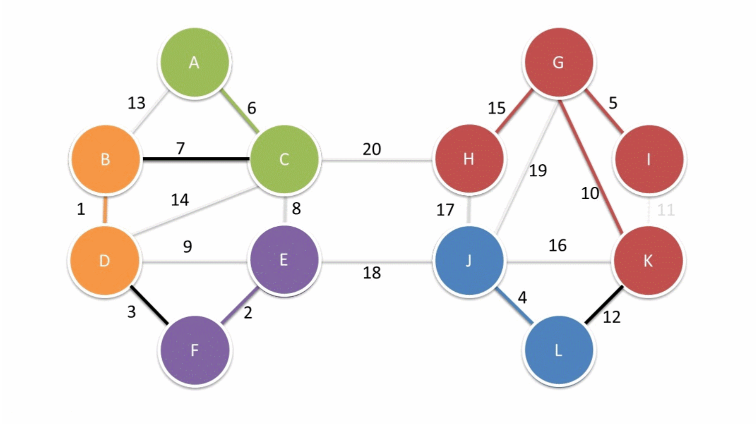
گراف ورودی:



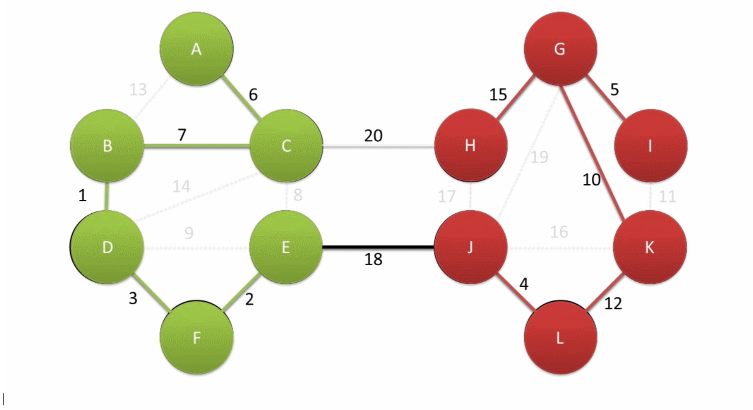
بعد از اجرای گام اول:



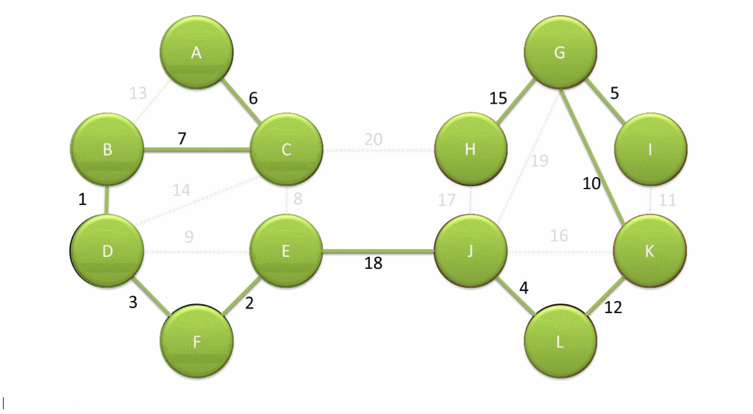
بعد از اجرای گام دوم:



بعد از اجرای گام سوم:



درخت نهایی:



۵- پیاده‌سازی الگوریتم

```
Input: A weighted undirected graph G = (V, E)
Output: F, a minimum spanning forest of G

Initialize a forest F to (V, E') where E' = {}

completed := false
while not completed do
    Find the connected components of F and assign to each vertex its component
    Initialize the cheapest edge for each component to "None"
    for each edge uv in E, where u and v are in different components of F:
        let wx be the cheapest edge for the component of u
        if is-preferred-over(uv, wx) then
            Set uv as the cheapest edge for the component of u
        let yz be the cheapest edge for the component of v
        if is-preferred-over(uv, yz) then
            Set uv as the cheapest edge for the component of v
    if all components have cheapest edge set to "None" then
        completed := true
    else
        completed := false
        for each component whose cheapest edge is not "None" do
            Add its cheapest edge to E'

function is-preferred-over(edge1, edge2) is
    return (edge2 is "None") or
            (weight(edge1) < weight(edge2)) or
            (weight(edge1) = weight(edge2) and tie-breaking-rule(edge1, edge2))

function tie-breaking-rule(edge1, edge2) is
    The tie-breaking rule; returns true if and only if edge1
    is preferred over edge2 in the case of a tie.
```

۶- درستی الگوریتم

لم ۲: در هیچ گامی دور ایجاد نمی‌شود (و بنابراین مستقیم نتیجه می‌شود که در نهایت هم دور ساخته نمی‌شود).

برهان: فرض کنید در یک گام یک دور ایجاد شود. ابتدا یک فرض اضافه در نظر می‌گیریم و در نهایت ثابت می‌کنیم که این فرض تاثیری نداشته. فرض کنید وزن تمام یال‌ها در گراف اولیه متفاوت باشد. از آنجا که یال‌ها را بین مولفه‌ها قرار می‌دهیم، کافی است مولفه‌ها را به عنوان super node در نظر بگیریم. فرض کنید یک دور ایجاد شده که دنباله آن به این صورت باشد: $v_1, v_2, ..., v_k, v_1$. که v_iها متمایز هستند و هر کدام نماینده یک مولفه هستند. از آنجا که وزن یال‌ها متمایز است، داریم:

$$w(v_1v_2) < w(v_kv_1)$$
$$w(v_2v_3) < w(v_1v_2)$$
$$w(v_3v_4) < w(v_2v_3)$$
$$...$$
$$w(v_kv_1) < w(v_{k-1}v_k)$$

که از این نامساوی‌ها نتیجه میگیریم:

$$w(v_kv_1) < w(v_kv_1)$$

که یک تناقض است. بنابراین در هیچ‌کدام از گام‌ها دور به وجود نمی‌آید. توجه کنید که یک فرض اضافه داشتیم که وزن تمام یال‌ها متمایز است. حال اگر این فرض را حذف کنیم، حتی اگر یکی از تساوی‌ها هم بیشتر اکید باشد باز هم می‌توان همین نتیجه را گرفت. تنها حالت بد این است که همه تساوی‌ها = باشند. در این صورت با توجه به پیاده‌سازی الگوریتم که در صورت مساوی بودن دو یال، یال با شماره مولفه کمتر را انتخاب می‌کرد به تناقض می‌رسیم.

از آنجا که از هر مولفه تنها یک یال رسم می‌شود، بنابراین یال v_1v_2 باید از مولفه یک انتخاب شده باشد، $w(v_1v_2) <$ v_2v_3 باید از مولفه دو باشد و ... $w(v_1v_2) < w(v)$

لم ۳: درخت نهایی دارای وزن کمینه در بین تمامی درخت‌هاست.

برهان: ثابت می‌کنیم بعد از هر گام، یک درخت پوشای کمینه وجود دارد که یال‌های فعلی، زیر مجموعه آن باشند.

فرض کنید در یک مرحله، یال)uv(بین دو راس u و v برگزیده شده. میخواهیم ثابت کنیم یک درخت پوشای کمینه وجود دارد که این یال بعلاوه یال‌های پیشین را دارد. این کار را به صورت استقرایی انجام می‌دهیم. فرض کنید مجموعه یال‌های برگزیده قبل از اضافه کردن این یال، T باشد. طبق فرض استقرا می‌دانیم یک درخت پوشای کمینه وجود دارد که T زیرمجموعه‌ی آن باشد. اکنون برای اثبات حکم استقرا، کافی است ثابت کنیم یک درخت پوشای کمینه وجود دارد که T + uv زیر مجموعه آن است. برهان خلف می‌زنیم. فرض کنید هیچ درخت پوشای کمینه نباشد که این یال‌ها زیرمجموعه آن باشند. فرض کنید این‌گونه نباشد. یک درخت پوشای کمینه که T زیرمجموعه آن است را M بگیرید. می‌دانیم یال uv در M نیست. پس یک یال دیگری باید باشد که دو مولفه‌ای که u و v در آن هستند را متصل کنید. این یال را xy بگیرید. حال یال xy را به M اضافه کنید. واضح است که یک دور ساخته می‌شود. از طرفی، طبق روشی که در الگوریتم داشتیم، اندازه یال xy

بیشتر از اندازه uv است، زیرا در غیر اینصورت این یال انتخاب می‌شد. بنابراین بعد از اضافه کردن یال uv، یک دور ایجاد می‌شود که یال xy هم در آن است. حال کافی است یال xy را حذف کنیم. واضح است که اندازه درخت کمتر شد و این تناقض است. زیرا فرض کرده بودیم که M مجموع وزن کمینه را دارد.

حال اگر دقت کنیم، می‌بینیم که فرض متمایز بودن وزن یال‌ها لازم نبود زیرا می‌توان به یال‌ها مقادیری اضافه کرد که وزن آن‌ها دو به دو فرق کند. اینگونه روند الگوریتم تغییر نمی‌کند ولی اثبات ما کارا خواهد بود. بنابراین لم ثابت شد.

۶-۱- جمع‌بندی

در لم دوم ثابت کردیم هیچ‌گاه دور ایجاد نمی‌شود. از طرفی، در هر گام تعداد مولفه‌های همبندی حداقل یک واحد کم می‌شود. بنابراین بعد از تعدادی گام حتما به یک درخت می‌رسیم (جایی که تنها یک مولفه همبندی می‌ماند). از طرفی در لم سوم ثابت کردیم که درختی که بدست می‌آید دارای کمینه وزن است. بنابراین ثابت کردیم که این الگوریتم در نهایت درخت پوشای کمینه را خروجی می‌دهد.

۷- الگوریتم‌های مشابه

همانطور که در بخش اول گفتیم، دو الگوریتم دیگر برای یافتن درخت پوشای کمینه وجود دارند. در جدول زیر به مقایسه این الگوریتم‌ها می‌پردازیم:

Algorithm	Time Complexity	Implementation Complexity	Publish Day
Baruvka	$O(E\ log\ V)$	Simple (DFS)	1926
Kruskal	$O(E\ log\ E)$	Medium (DSU)	1956
Prim	$O(V + V\ log\ V)$	Complex (Fibonacci heap)	1957

مراجع

[1] L. Huang, Dai, “Equipping the barzilai–borwein method with the two dimensional quadratic termination property,” *SIAM*, 2021.
[2] G. Zhou, Dai, “Gradient methods with adaptive step-sizes,” *Comput. Optim Appl.*, 2006.
[3] Y. Fletcher, Teo, “On the barzilai–borwein method, in optimization and control with applications,” *Springer, Boston*, 2005.