درخت پوشای کمینه

مقدمه

طراحی مدارات الکترونیکی اغلب اوقات نیازمند به وصل کردن قطعات مدار به صورت الکتریکی با سیم‌کشی کردن بین آن‌ها می‌باشد. برای اینکه مثلا n قطعه‌ی الکترونیکی را به هم وصل کنیم، نیازمند n-1 سیم هستیم. از میان تمام حالات سیم‌کشی بین قطعات، آن حالتی که به کم‌ترین میزان سیم نیاز دارد مورد علاقه‌ی ما می-باشد. ما می‌توانیم مسأله‌ی سیم‌کشی مدارات را با یک گراف بدون جهت همبند مثل G = (V,E) مدل کنیم، که در اینجا V مجموعه‌ی رئوس گراف و E مجموعه‌ی یال‌های گراف است و برای هر یال (u,v) € E به w(u,v) وزن یال uv می گوییم. بدین صورت مسأله‌ی ما تبدیل می شود به پیدا کردن یک زیر مجموعه مثل T از گراف E به طوریکه تمام رئوس گراف را به هم وصل کند، دور نداشته باشد و همچنین w(T) $=\sum_{(u,v)\in T} w(u,v)$ کمترین مقدار خود را داشته باشد. چون T دور ندارد پس حتماً یک درخت است و چون تمام رئوس را پوشش می دهد پس T یک درخت پوشاست. از طرفی می‌خواهیم در بین تمام درخت‌های پوشای ممکن، T کمترین وزن را داشته باشد پس می‌توانیم به مسأله‌ی پیدا کردن زیرمجموعه‌ی T مسأله‌ی یافتن درخت پوشای کمینه نیز بگوییم.

در این فصل ما دو الگوریتم را برای پیدا کردن درخت پوشای کمینه بررسی می‌کنیم: الگوریتم کراسکال[۱] و الگوریتم پریم[۲]. هر کدام از این دو الگوریتم دارای هزینه‌ی زمانی O(ElgV) هستند. اگر در پیاده‌سازی آن‌ها از ساختمان داده‌ی پشته(هیپ) ی باینری[۳] استفاده کنیم. اگر از پشته‌ی فیبوناتچی[۴] استفاده کنیم می‌توانیم هزینه ی این دو را به O(E + VlgV) کاهش دهیم که همان‌طور که می‌بینید اگر |E| خیلی از |V| بزرگتر باشد، شاهد بهبود زیادی در هزینه خواهیم بود. این دو الگوریتم از نوع حریصانه[۵] هستند. هر مرحله از یک الگوریتم حریصانه از بین چند انتخاب ممکن، گزینه‌ای را انتخاب می‌کند که در آن لحظه بهترین باشد. در بخش بعدی روشی عمومی را معرفی می‌کنیم که برای پیدا کردن درخت پوشای کمینه در هر مرحله یک یال به درخت مورد نظر اضافه می‌کند. دو الگوریتمی هم که بعداً معرفی می‌کنیم بر پایه ی همین روش عمل می کنند.

ساختن یک درخت پوشای کمینه

فرض کنید که یک گراف همبند بدون جهت مانند G = (V,E) با یک تابع مثل w: E→ ℝ داریم و می خواهیم درخت پوشای کمینه را برای این گراف بدست آوریم. این استراتژی حریصانه از یک روش عمومی بهره

---

۱ Kruskal
۲ Prim
۳ Binary Heap
۴ Fibonacci Heap
۵ Greedy

می‌برد که در هر مرحله یک یال به درخت پوشای کمینه‌ی مورد نظر ما اضافه می‌کند. در این روش یک مجموعه از یال‌ها به نام A داریم که در آن A یک زیرمجموعه از یکی از درخت‌های پوشای کمینه‌ی ممکن است. در هر مرحله ما یالی مثل (u,v) را که می توانیم به A اضافه کنیم بدون اینکه این شرط را بهم زده باشیم که $A \cup \{(u,v)\}$ هم زیر مجموعه‌ی درخت پوشای کمینه باشد، پیدا می کنیم. به چنین یالی یک یال امن[۶] برای A می گوییم. شبه کد این روش را در زیر می بینید:

GENERIC_MST(G, w)

A = 0;

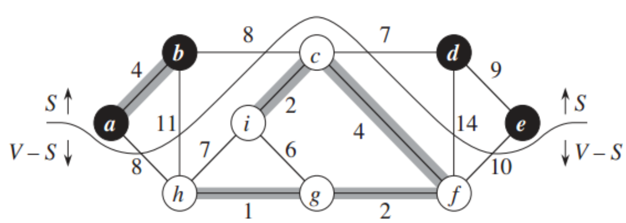
while A does not form a spanning tree

find and edge (u,v) that is safe for A

$A = A \cup \{(u,v)\}$

return A

مشکل ما یافتن یال امن برای A است. قضیه‌ای که در آخر این بخش داریم روش پیدا کردن یال امن را به ما نشان می دهد. اما قبل از آن نیازمند به چند تعریف هستیم. یک برش[۷] (S, V-S) از گراف بدون جهت G = (V, E) یک تقسیم مجموعه ی V به دو مجموعه است. شکل ۱ نمونه ی یک برش را نشان می دهد:



شکل ۱ – برش C = (S, V-S) . رؤوس سیاه در دسته ی S و رؤوس سفید در دسته یV-S قرار دارند.

[۶] Safe Edge
[۷] Cut

می گوییم یک یال یک برش را قطع می کند اگر یکی از نقاط پایانش در S و دیگری در V-S باشد. می گوییم یک برش به یک مجموعه از رئوس مثل A احترام می گذارد اگر تمام رئوس A در یک طرف برش قرار گرفته باشند. یک یال را یال سبک[۸] می نامیم اگر از تمام یال هایی که یک برش را قطع می کنند وزن کمتری داشته باشد. توجه کنید که ممکن است بیش از یک یال سبک یک برش را قطع کنند. قضیه ی زیر روش پیدا کردن یال امن را به ما می گوید:

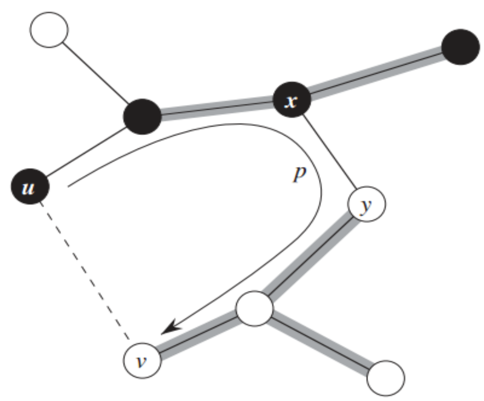
قضیه۱: فرض کنید G یک گراف همبند بدون جهت باشد با تابع مقدار حقیقی وزن w. فرض کنید A یک زیر مجموعه از E باشد که در یک درخت پوشای کمینه برای G مشمول شده باشد. همینطور فرض کنید (S, V-S) یک برش از گراف G باشد که به A احترام می‌گذارد و (u,v) یک یال سبک باشد که (S, V-S) را قطع می کند. در این صورت یال (u,v) یک یال امن برای A می باشد.

اثبات: فرض کنید درخت پوشای کمینه T که A را شامل شده دارای یال (u,v) که یک یال سبک است، نباشد. در این صورت ما درخت دیگری پیدا می کنیم که از T وزن کمتری داشته باشد. در درخت T یال (u,v) به همراه مسیر ساده‌ی p بین این دو رأس در T یک دور می سازد. از طرفی چون u و v در دو طرف مخالف برش (S, V-S) هستند، حداقل یک یال از مسیر p برش (S, V-S) را قطع می‌کند. فرض کنید آن یال (x,y) باشد. از طرفی (x,y) عضو A نیست چون برش را قطع کرده. حال اگر یال (x,y) از T را حذف کرده و یال (u,v) را اضافه کنیم به درختی جدید مانند T’ می رسیم که داریم: $T' = T - \{(x,y)\} \cup \{(u,v)\}$. با یک محاسبه ی ساده داریم:

$$w(T') = w(T) - w(x,y) + w(u,v) \ \leq w(T)$$

که به وضوح یک تناقض است. پس قضیه اثبات شد.

[۸] Light Edge



شکل ۲ – اثبات قضیه ی ۱. یال های رنگ شده یال هایی هستند که جزو درخت پوشای کمینه هستند.

در واقع در متد عمومی یک جنگل داریم مانند $G_A = (V, A)$ داریم که هر عضو آن یک درخت است. در هر مرحله دو تا از بخش های G_A به هم وصل می‌شوند و تشکیل یک درخت می‌دهند. در شروع کار G_A یک دارای $|V|$ رأس یا همان $|V|$ بخش(درخت یک رأسی) می‌باشد. ضمناً توجه به این نکته ضروری است که در روش عمومی، در هر مرحله مجموعه ی A همیشه بدون دور باقی می‌ماند. بنابراین کافیست که حلقه ی این متد $|V|-1$ بار اجرا شود.

قضیه ۲: در گراف $G = (V,E)$ فرض کنید $C = (V_C, E_C)$ یک بخش همبند(درخت) در جنگل $G_A = (V, A)$ باشد. اگر یال (u,v) یک یال سبک باشد که C را به بخش های دیگر G_A وصل کند در این صورت (u,v) یک یال امن برای A است.

اثبات: برش $(V_C , V-V_C)$ به A احترام می گذارد و (u,v) یک یال سبک برای این برش است. پس طبق قضیه ی شماره ی ۱ یال (u,v) برای A امن است.

الگوریتم های کراسکال و پریم

دو الگوریتم پریم و کراسکال بر پایه ی متد عمومی عمل می‌کنند. هر دو الگوریتم از یک روش خاص برای پیدا کردن یال امن استفاده می‌کنند. در الگوریتم کراسکال مجموعه ی A یک جنگل است که رئوس آن تمام رئوس گراف داده شده می‌باشد. یال امن برای مجموعه‌ی A همیشه کمترین(کم وزن ترین) یال در گراف است که دو بخش جدا را به هم وصل می کند. در الگوریتم پریم مجموعه ی A یک درخت است یال امن اضافه شده به مجموعه ی A همیشه کم‌وزن‌ترین یال است که درخت را به یک رأس غیر از درخت وصل می‌کند.

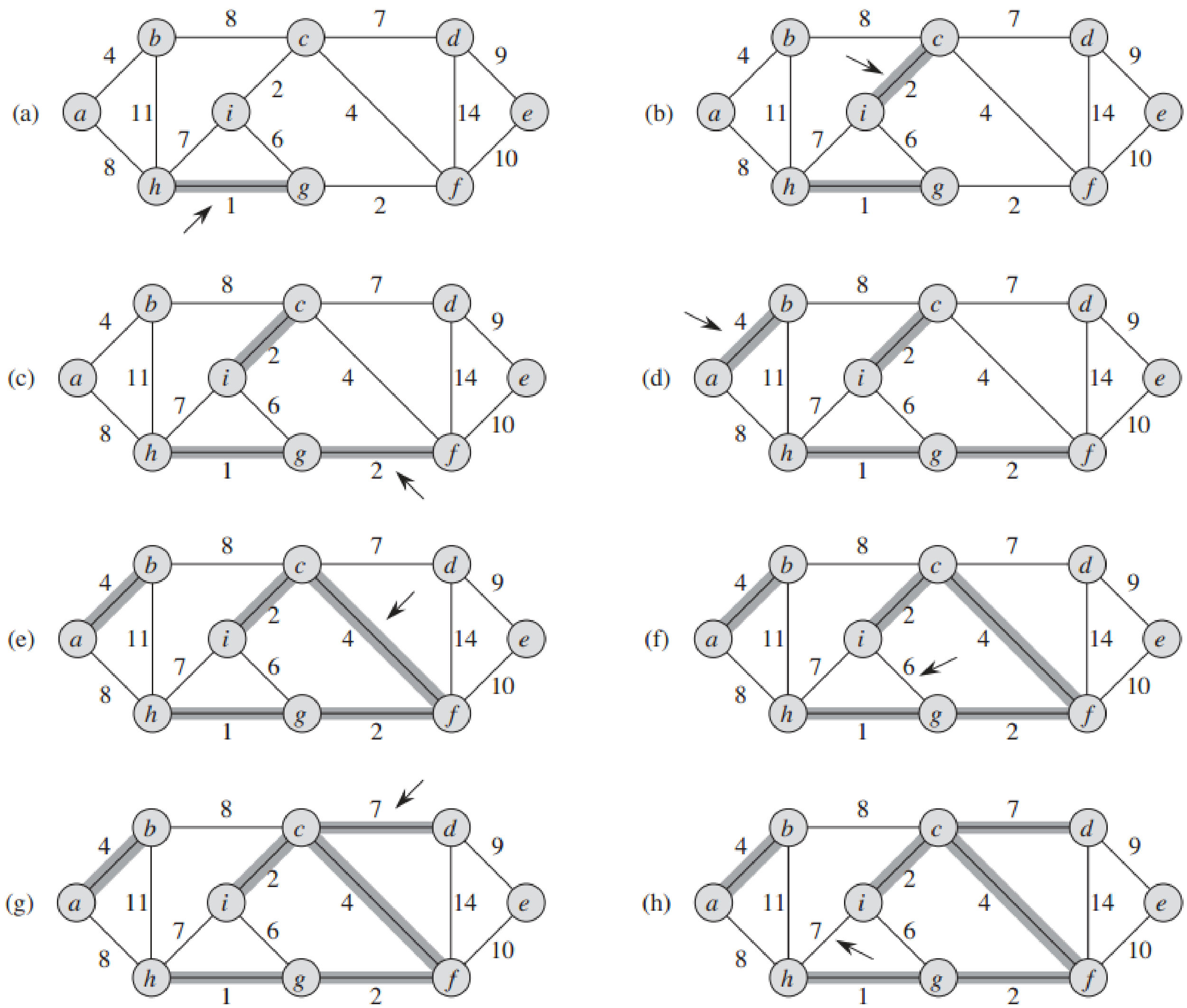
الگوریتم کراسکال:

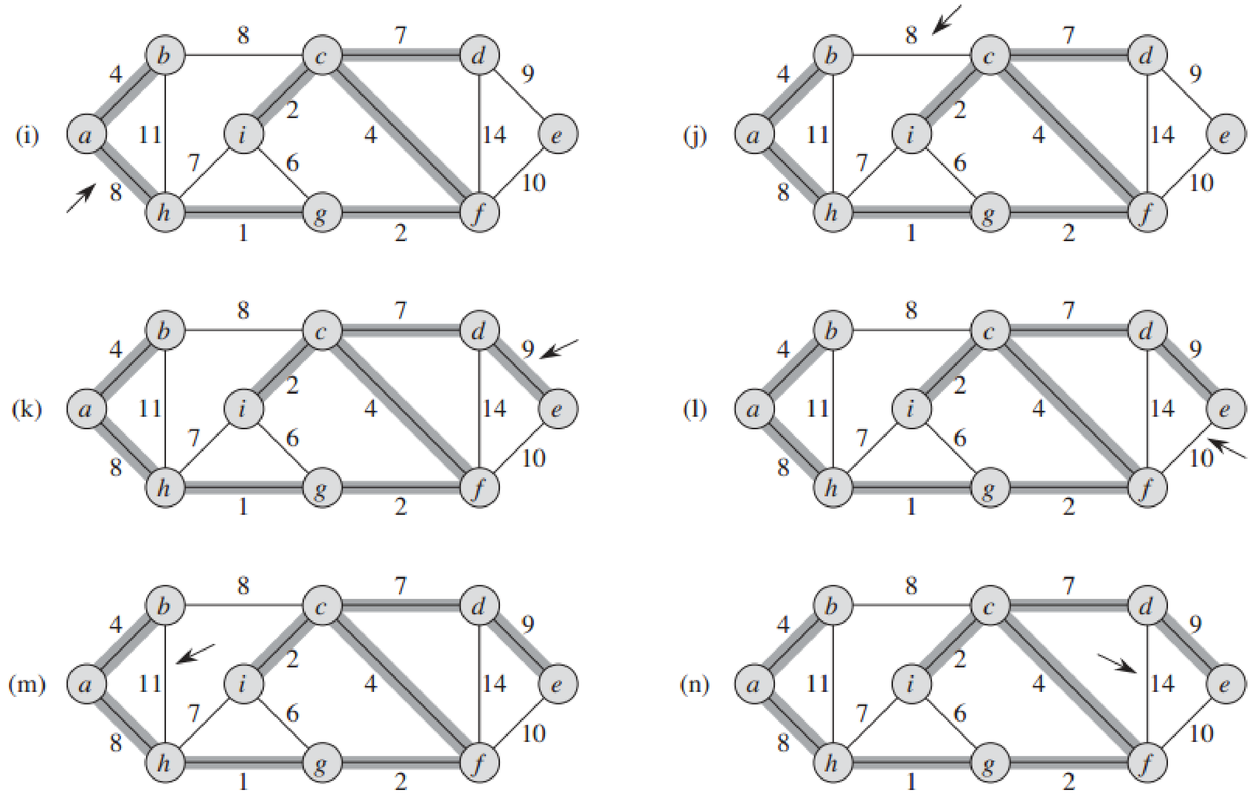
الگوریتم کراسکال یک یال امن پیدا می کند تا به یک جنگل در حال رشد اضافه کند. این یال را از بین تمام یال‌هایی که هر دو درختی در جنگل را به هم وصل می‌کنند پیدا می‌کند، یالی که وزن آن کمترین نیز باشد. مثلاً اگر یک یال سبک مثل (u,v) دو بخش C_1 و C_2 را به هم وصل کند طبق قضیه‌ی قبل یال (u,v) یک یال امن برای C_1 یا C_2 محسوب می شود.

```
MST-KRUSKAL(G, w)
    A = 0
    for each vertex v ϵ G.V
        MAKE-SET(v)
    sort the edges of G.E into nondecreasing order by weight w
    for each edge(u, v) ϵ G.E taken in nondecreasing order by weight
        ifFIND-SET(u) != FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u,v)
    return A
```

خط ۳-۱ مقدار دهی اولیه‌ی مجموعه ی A به تهی و ساختن |V| درخت است که هر کدام یک رأس دارند. حلقه ی for در خط‌های ۸-۵ یال‌ها را به ترتیب صعودی بر اساس وزن‌شان بررسی می‌کند. حلقه برای هر یال (u,v) بررسی می‌کند که آیا در انتهای u و v در یک درخت قرار دارند یا نه. اگر جواب مثبت است در آن صورت یال (u,v) را نمی‌توان بدون ایجاد دور در گراف به جنگل اضافه کرد و یال را رد می‌کنیم. در غیر این صورت دو رأس در دو درخت متفاوتند. سپس خط ۷ یال (u,v) را به مجموعه‌ی A اضافه می‌کند و خط ۸ رؤوس دو درخت را ادغام کرده و به یک درخت تبدیل می‌کند. شکل زیر مراحل اجرای الگوریتم را نشان می‌دهد:

درخت پوشای کمینه





شکل ۳- مراحل اجرای الگوریتم کراسکال. یال‌های هاشور خورده به جنگل در حال رشد A تعلق دارند. در هر مرحله فلش به یالی که در حال بررسی است اشاره می‌کند.

برای محاسبه‌ی زمان اجرای الگوریتم کراسکال از خط اول الگوریتم شروع می‌کنیم. مقداردهی اولیه‌ی A در خط ۱ هزینه‌ی $O(1)$ دارد و هزینه‌ی مرتب کردن یال‌ها بر اساس وزن، $O(ElgV)$ می‌باشد(هزینه‌ی |V| بار فراخوانی MAKE-SET را بعداً حساب می‌کنیم). حلقه‌ی for در خط ۸-۵ $O(E)$ بار عمل‌های FIND-SET و UNION را بر روی مجموعه‌های جدای جنگل انجام می‌دهد. به همراه |V| بار عمل MAKE-SET این اعمال هزینه‌ی کلی $O((V+E)\alpha(V))$ می‌برند، بطوریکه α یک تابع با شیب صعودی بسیار کم است. چون ما فرض کرده ایم که G همبند است پس $|E| \geq |V|-1$, بنابراین اعمال بالا زمان $O\big(E\alpha(V)\big)$ می برند. علاوه بر این چون $\alpha(|V|) = O(lgV) = O(lgE)$ زمان کلی الگوریتم کراسکال $O(ElgE)$ خواهد شد. با توجه به اینکه $E < |V|^2$ داریم $lgE = O(lgV)$ پس می‌توانیم رابطه هزینه الگوریتم کراسکال را اینگونه بنویسیم: $O(ElgV)$.

الگوریتم پریم:

همانند الگوریتم کراسکال، الگوریتم پریم نیز حالت خاصی از متد عمومی درخت پوشای کمینه است. در الگوریتم پریم درخت از یک رأس به نام ریشه شروع می شود و رشد می‌کند تا جایی که تمام رئوس گراف را پوشش دهد. هر مرحله یک یال سبک که A را به یک رأس تنها وصل می‌کند، به درخت A وصل می‌شود. رأس تنها رأسی است که در درخت A وجود نداشته باشد. این الگوریتم تنها یال‌هایی را که برای A امن هستند به A اضافه می کند پس وقتی الگوریتم پایان می‌یابد درخت A همان درخت پوشای کمینه‌ی می‌باشد. برای اینکه الگوریتم پریم با کارایی بالا اجرا شود، در هر مرحله از الگوریتم تمام رئوسی که در درخت ما نیستند در یک صف اولویت کمینه مانند Q بر اساس مقدار کلیدشان نگه‌داری می‌شوند. برای هر رأس مانند v، عبارت v.key وزن یال با کمترین وزن در بین یال‌های متصل به v را نشان می دهد. اگر $v.key = \infty$ یعنی چنین یالی وجود ندارد. همینطور عبارت $v.\pi$ پدر رأس v را در درخت نشان می‌دهد.

MST-PRIM(G, w, r)

foreach $u \epsilon G.V$

$\quad u.key = \infty$

$\quad u.\pi = null$

$r.key = 0$

$Q = G.V$

WhileQ != 0

$\quad$ u = EXTRACT-MIN(Q)

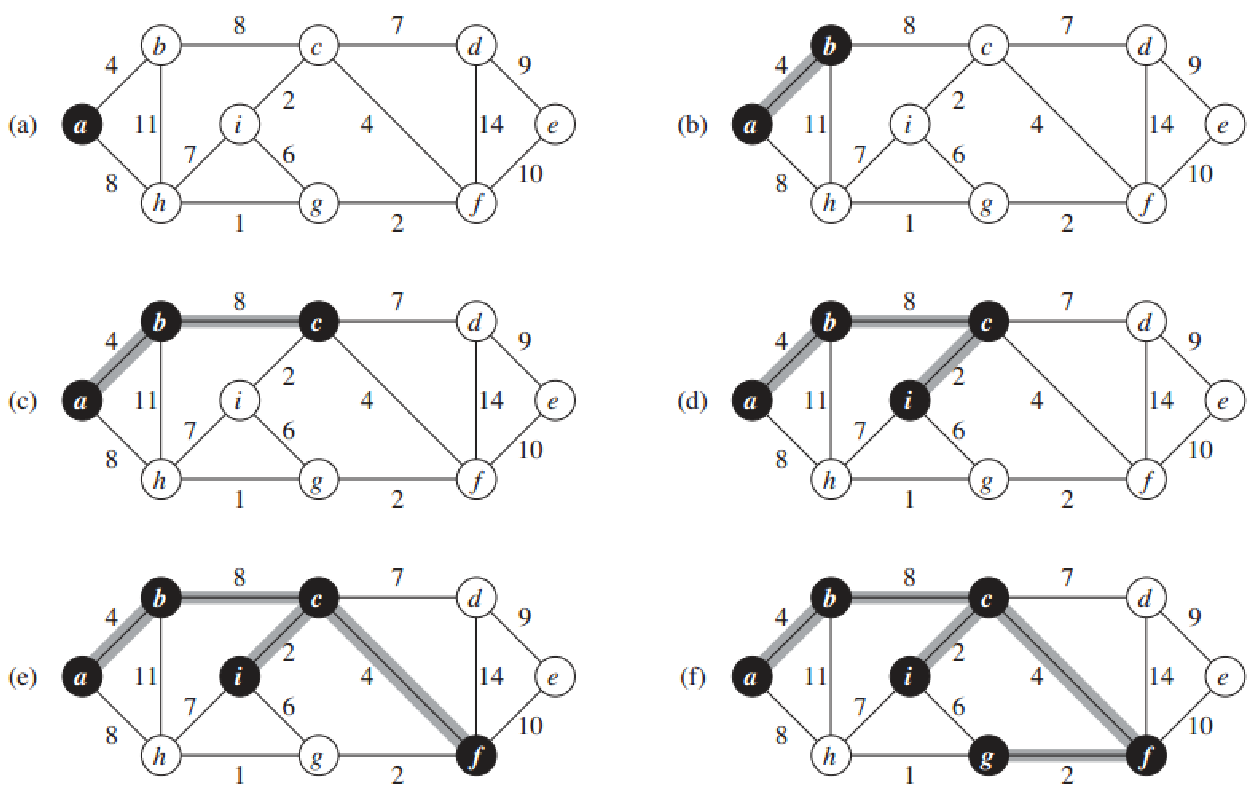
$\quad$ **for** each $v \epsilon G.Adj[u]$

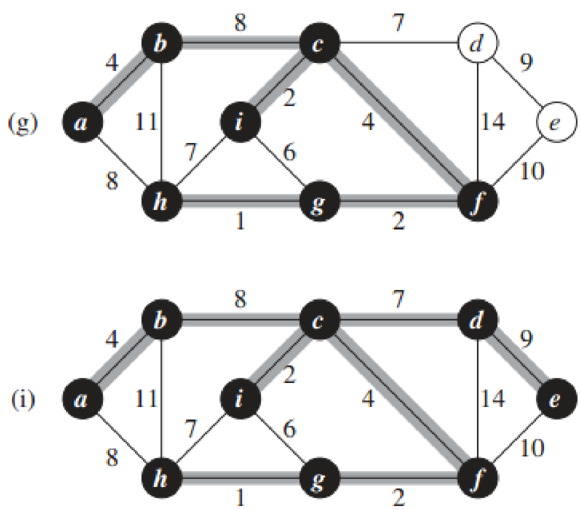
$\quad\quad$ **if**$v\epsilon Q$ and $w(u,v) < v.key$

$\quad\quad\quad v.\pi = u$

$\quad\quad\quad v.key = w(u,v)$

در شبه کد خط‌های ۵-۱ مقدار key هر رأس را برابر ∞ قرار می‌دهد(البته به جز ریشه که مقدار key آن برابر صفر قرار می گیرد تا همیشه به عنوان اولین راس خارج شود)، پدر هر رأس را null قرار می دهد و صف اولویت Q تمام رئوس را شامل می‌شود. خط ۷ یک رأس مانند $u \epsilon Q$ را مشخص می‌کند که روی یک یال سبک که برش (V-Q, Q) را قطع می‌کند واقع است. سپس u را از Q حذف کرده به V-Q اضافه می‌کند و این یعنی اضافه کردن یال $(u, u.\pi)$ به مجموعه A . حلقه‌ی for در خطوط ۱۱-۸ عبارات key و π را برای تمام رؤوس متصل به u که در درخت نیستند، به روز می‌کند.





شکل ۴- مراحل اجرای الگوریتم پریم. رأس a ریشه است. رؤوس سیاه و یال های هاشور خورده در درخت در حال رشد هستند.

زمان اجرای الگوریتم پریم بستگی به آن دارد که چگونه صف اولویت کمینه را پیاده‌سازی کنیم. اگر Q را با پشته‌ی مینیمم باینری پیاده‌سازی کنیم می‌توانیم با استفاده از تابع BUILD-MIN-HEAP خطوط ۵-۱ را در $O(V)$ اجرا کنیم. حلقه ی while , |V| بار اجرا می شود و چون هر عملیات EXTRACT-MIN هزینه ی $O(lgV)$ دارد هزینه‌ی کل فراخوانی های EXTRACT-MIN برابر $O(VlgV)$ خواهد شد. حلقه‌ی for در خطوط ۱۱-۸ $O(E)$ بار اجرا می‌شود زیرا جمع تعداد همه‌ی لیست‌های مجاورت 2|E| است. داخل حلقه‌ی for، برای فهمیدن عضویت هر راس در Q، یک بیت برای هر راس نگه می‌داریم و وقتی یک راس از Q خارج می‌شود، آن بیت را به روز می‌کنیم. در خط ۱۱ نیاز به اجرای عمل DECREASE-KEY روی پشته ی مینیمم دارد که هزینه‌ی $O(lgV)$ دارد که بنابراین کل هزینه برای الگوریتم پریم $O(VlgV + ElgV) = O(ElgV)$ خواهد بود که با الگوریتم کراسکال برابر است. البته اگر از پشته‌ی فیبوناتچی استفاده می‌کردیم می‌توانستیم الگوریتم را با هزینه‌ی $O(E + VlgV)$ انجام دهیم چون هزینه‌ی سرشکن آن برای DECREASE-KEY , $O(1)$ می باشد.

چند تمرین

درخت پوشای کمینه

یک . فرض کنید تمام وزن‌های یال‌ها در گراف، اعداد صحیح از یک تا $|V|$ باشند. الگوریتم Prim را چقدر می‌توانید سریع‌تر کنید با دانستن این موضوع؟ اگر وزن‌ها اعداد صحیح از ۱ تا W که W یک عدد ثابت باشد، باشند چه؟

دو. فرض کنید یک درخت پوشای کمینه برای گراف G بدست آمده است. اگر یک راس به همراه یال‌های متصلش به گراف اضافه شود، کمترین هزینه برای به روز کردن درخت پوشای کمینه‌ی بدست آمده چقدر است؟

سه. دومین درخت پوشای کمینه در یک درخت، درختی است که بعد از درخت پوشای کمینه، نسبت به سایر درخت‌های پوشای گراف کمینه باشد. موارد زیر را برای گراف بدون جهت و همبند G که وزن یال‌هایش متمایز می‌باشد و تعداد یال‌هایش از تعداد راس‌هایش بیشتر می‌باشد پاسخ دهید:

الف. نشان دهید که درخت پوشای کمینه‌ی این گراف منحصر به فرد است ولی دومین درخت پوشای کمینه‌ی آن لزوما منحصر به فرد نیست.

ب. نشان دهید اگر T درخت پوشای کمینه‌ی گراف G باشد، آنگاه یال‌های $(u, v) \in T$ و $(x, y) \in G - T$ وجود دارند به طوریکه $T' = T - \{(x, y)\} \cup \{(u, v)\}$ دومین درخت پوشای کمینه‌ی گراف G باشد.

ج. بهترین هزینه‌ی پیدا کردن دومین درخت پوشای کمینه‌ی گراف G چقدر می‌باشد؟