دانشکده‌ی علوم ریاضی

آنالیز الگوریتم‌ها — ۱۳ اسفند ۹۱

جلسه‌ی ۸: الگوریتم دایکسترا

مدرّس: دکتر شهرام خزائی — *نگارنده: پردیس ملک‌زاده*

می‌خواهیم کوتاه‌ترین فاصله‌ی همه‌ی رأس‌ها از رأس دلخواه s در یک گراف را بیابیم. در صورتی که یال‌ها بدون وزن باشند می‌توانیم از الگوریتم BFS استفاده کنیم. اکنون مسأله‌ی پیدا کردن کوتاه‌ترین مسیر در گراف‌های وزن‌دار را در نظر بگیرید. گراف ساده‌ی $G = (V, E)$ که به هر یال e در آن وزن l_e داده شده است را در نظر بگیرید. گره‌ها می‌توانند نمایانگر شهرها و وزن یک یال می‌تواند نمایانگر فاصله‌ی بین دو شهر، هزینه‌ی رفت و آمد بین دو شهر و یا زمان رفت و آمد بین دو شهر باشد. برای هر رأس دلخواه v، فاصله‌ی v از s را $\mathrm{dist}[v]$ می‌نامیم. در حالت‌های خاص زیر مسأله را می‌توان به الگوریتم BFS کاهش داد.

- اگر $l_e = ۱$ برای هر $e \in E$ باشد با گراف مانند گراف بدون وزن برخورد کرده و با استفاده از الگوریتم BFS به جواب می‌رسیم.
- اگر وزن یال‌ها عدد صحیح باشند می‌توان با اضافه کردن رأس‌های dummy گراف G را به یک گراف بدون وزن تبدیل کرد: به این صورت که هر یال $e = (u, v)$ با طول l_e را با یک مسیر به طول l_e بین u و v جایگزین می‌کنیم. برای ایجاد چنین مسیری باید $l_e - ۱$ رأس dummy به گراف اضافه کنیم.

ایده اضافه کردن رأس dummy الگوریتم کارایی به ما نمی‌دهد. برای حل این مسأله در حالت کلی از الگوریتم دایکسترا[۱] استفاده می‌شود که در ادامه معرفی خواهد شد.

۱ الگوریتم دایکسترا

الگوریتم دایکسترا از الگوریتم‌های حریصانه می‌باشد. در این الگوریتم ابتدا مجموعه‌ی X را برابر $\{s\}$، $\mathrm{dist}[s]$ را برابر ۰ و dist بقیه‌ی رأس‌های گراف را برابر ∞ تعریف می‌کنیم. در هر مرحله همه‌ی یال‌های $e = (v, w)$ که یک سر آن‌ها در X $(v \in X)$ و سر دیگر آن‌ها در $V - X$ $(w \in V - X)$ است را در نظر گرفته، یالی را انتخاب می‌کنیم که کمیت $\mathrm{dist}[v] + l_{vw}$ را کمینه کند، w را به X می‌افزاییم و $\mathrm{dist}[w]$ را برابر $\mathrm{dist}[v] + l_{vw}$ قرار می‌دهیم.

شبه کد این الگوریتم را در زیر مشاهده می‌کنید:

[۱] Dijkstra

Algorithm 1 Algorithm: Dijkstra's Shortest Path Algorithm

```
function Dijkstra1(Graph G, Vertex s, length {l_e}_{e∈E})
    [assumes s is a vertex of G]
    for all v ∈ V do
        dist[v] ← ∞
    dist[s] ← 0
    X ← {s}
    while X ≠ V do
        Among all edges (v, w) ∈ E with v ∈ X and w ∉ X, pick the one that minimizes dist[v] + l_vw
         and call it (v′, w′)
        Add w′ to X
        dist[w′] ← dist[v′] + l_v′w′
    return dist[·]
```

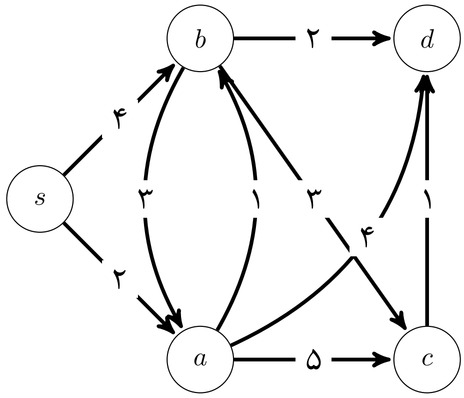
برای این که علاوه بر طول کوتاه‌ترین مسیر بین دو رأس، خود مسیر را نیز به دست آوریم، می‌توانیم در هر مرحله کوتاه‌ترین مسیر پیدا شده را به نحو مناسبی ذخیره کنیم، بدین منظور شبه کد الگوریتم دایکسترا را به صورت زیر بازنویسی می‌کنیم:

Algorithm 2 Algorithm: Dijkstra's Shortest Path Algorithm Edited To Find Path

```
function Dijkstra2(Graph G, Vertex s, length {l_e}_{e∈E})
    [assumes s is a vertex of G]
    for all v ∈ V do
        dist[v] ← ∞
        prev[v] ← null
    dist[s] ← 0
    X ← {s}
    while X ≠ V do
        Among all edges (v, w) ∈ E with v ∈ X and w ∉ X, pick the one that minimizes dist[v] + l_vw
         and call it (v′, w′)
        Add w′ to X
        dist[w′] ← dist[v′] + l_v′w′
        prev[w′] ← v′
    return dist[·], prev[·]
```

در این صورت مشروط بر این که $\mathrm{dist}[v] \neq \infty$ (یا معادلاً $\mathrm{prev}[v] \neq \mathsf{null}$) کوتاه‌ترین مسیر از s به v مسیر $v, v_۱, ..., v_{t-۱}, s$ است که $v_۰ = v$ و $\mathrm{prev}[v_{i-۱}] = v_i$ برای $i = ۱, ۲, ..., t$ که $v_t = s$.

مثال ۱ *الگوریتم دایکسترا را بر روی گراف جهت‌دار زیر اجرا می‌کنیم.*



جدول زیر مراحل اجرای الگوریتم را نشان میدهد:

dist	مرحله ۰	مرحله ۱	مرحله ۲	مرحله ۳	مرحله ۴
s	۰	۰	۰	۰	۰
a	∞	۲	۲	۲	۲
b	∞	∞	۳	۳	۳
c	∞	∞	∞	∞	۷
d	∞	∞	∞	۵	۵
X	$\{s\}$	$\{s,a\}$	$\{s,a,b\}$	$\{s,a,d,d\}$	$\{s,a,b,c,d\}$
(v',w')	$-$	sa	ab	bd	ac

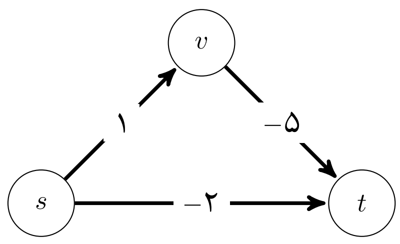
ابتدا $X=\{s\}$، $\mathrm{dist}[s]=۰$ و $\mathrm{dist}[a]=\mathrm{dist}[b]=\mathrm{dist}[c]=\mathrm{dist}[d]=\infty$ *است.*
در مرحله‌ی اول یال sa *انتخاب می‌شود،* a *به* X *اضافه می‌شود و* $\mathrm{dist}[a]$ *برابر ۲ می‌شود.*
در مرحله‌ی دوم یال ab *انتخاب می‌شود،* b *به* X *اضافه می‌شود و* $\mathrm{dist}[b]$ *برابر ۳ می‌شود.*
در مرحله‌ی سوم یال bd *انتخاب می‌شود،* d *به* X *اضافه می‌شود و* $\mathrm{dist}[d]$ *برابر ۵ می‌شود.*
در مرحله‌ی چهارم یال ac *انتخاب می‌شود،* c *به* X *اضافه می‌شود و* $\mathrm{dist}[c]$ *برابر ۷ می‌شود.*
پس از اتمام الگوریتم داریم:

$$\mathrm{dist}[s]=۰,\mathrm{dist}[a]=۲,\mathrm{dist}[b]=۳,\mathrm{dist}[c]=۷,\mathrm{dist}[d]=۵$$

۲ گراف‌های با طول منفی

الگوریتم دایکسترا تنها در صورتی که وزن یال‌ها اعداد مثبت باشند صحیح کار می‌کند و در صورتی که وزن یال‌ها منفی باشند ممکن است درست کار نکند.

مثال ۲ *در گراف زیر وزن تعدادی از یال‌ها منفی است. همان طور که مشاهده می‌کنید طول کوتاه‌ترین مسیر از* s *به* t *برابر ۴− است، در حالی که بر اساس الگوریتم دایکسترا طول کوتاه‌ترین مسیر از* s *به* t *برابر ۲− محاسبه می‌شود.*



ایده‌های زیر برای این که الگوریتم برای گراف‌های شامل یال‌هایی با طول منفی هم کار کند، به ذهن می‌رسد:

۱. برعکس کردن جهت یال‌ها با وزن منفی و ضرب وزن یال در -1

۲. اضافه کردن یک مقدار ثابت به وزن همه‌ی یال‌ها به طوری که وزن همه‌ی یال‌ها مثبت شود

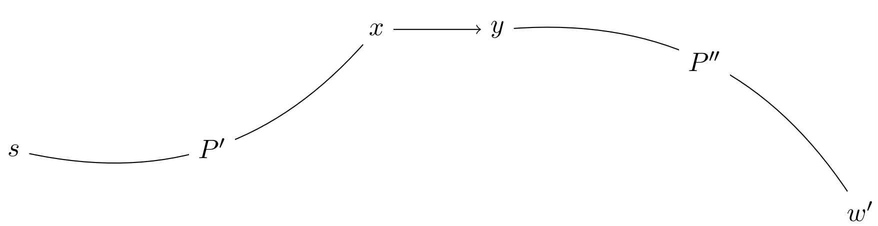
اما به راحتی می‌توان نشان داد که هیچ کدام از این ایده‌ها کار نمی‌کند. در این جلسه صرفاً خود رامحدود به گراف‌ها با طول نامنفی می‌کنیم و در آینده گراف‌های کلی را در نظر می‌گیریم.

۳ اثبات درستی الگوریتم دایکسترا

قضیه ۱ *اگر وزن یالها مثبت باشند، پس از توقف الگوریتم دایکسترا* $\mathrm{dist}[v]$ *طول کوتاه‌ترین مسیر از* s *به* v *در* G *است.*

برهان. X_k را مجموعه‌ی X بعد از k $(0 \leqslant k \leqslant |V|-1)$ مرحله اجرای الگوریتم دایکسترا روی گراف همبند G می‌نامیم. نشان می‌دهیم برای هر رأس دلخواه v در X_k $\mathrm{dist}[v]$ طول کوتاه‌ترین مسیر از s به v در G است.
با استقرا روی k حکم را ثابت می‌کنیم. پایه‌ی استقرا $k=0$ بدیهی است. فرض کنید حکم به ازای k برقرار است. می‌خواهیم درستی حکم را به ازای $k+1$ ثابت کنیم.
فرض کنید w'، $k+1$ امین رأسی است که به $X = X_k$ اضافه شده و در $k+1$ امین مرحله‌ی اجرای الگوریتم یال $v'w'$ انتخاب شده است. چون $X_{k+1} = X_k \cup \{w'\}$ و پس از مرحله $k+1$ ام تنها مقدار $\mathrm{dist}[w']$ تغییر می‌کند، کافی است نشان دهیم برای راس w'، کوتاه‌ترین فاصله رأس s از رأس w' برابر $\mathrm{dist}[v'] + l_{v'w'}$ است. کوتاه‌ترین مسیر از s به v' به علاوه‌ی یال $v'w'$ یک مسیر از s به w' به طول $\mathrm{dist}[v'] + l_{v'w'}$ است. ادعا می‌کنیم که این مسیر خود یک کوتاهترین مسیر از s به w' است. مسیر دلخواه P از رأس s به رأس w' را در نظر بگیرید. می‌خواهیم ثابت کنیم طول P از $\mathrm{dist}[v'] + l_{v'w'}$ کمتر نیست. اولین یال xy در مسیر P که یک سر آن درX_k $(x \in X_k)$ و سر دیگر آن خارج از X_k $(y \notin X_k)$ است را در نظر بگیرید. بخشی از مسیر P که بین s و x است را P' و بخشی که بین y و w' است را P'' می‌نامیم. داریم:

$$l_P = l_{P'} + l_{xy} + l_{P''}$$



سپس با توجه به مثبت بودن وزن یال‌ها داریم:

$$l_P \geqslant l_{P'} + l_{xy} \tag{۱}$$

چون $x \in X_k$ است، در نتیجه با توجه به فرض استقرا داریم:

$$l_{P'} \geqslant \text{dist}[x] \Rightarrow l_{P'} + l_{xy} \geqslant \text{dist}[x] + l_{xy} \tag{۲}$$

یال xy را در نظر بگیرید. یک سر این یال در X_k و سر دیگر آن خارج از X_k است. یک سر یال $v'w'$ نیز در X_k و سر دیگر آن خارج از X_k است. اما در مرحله k ام، الگوریتم دایکسترا یال $v'w'$ را انتخاب کرده است، پس داریم:

$$\text{dist}[x] + l_{xy} \geqslant \text{dist}[v'] + l_{v'w'} \tag{۳}$$

با توجه به نامساوی‌های ۱ و ۲ و ۳ داریم:

$$l_P \geqslant d(v') + l_{v'w'}$$

■

۴ پیچیدگی الگوریتم

در الگوریتم دایکسترا حلقه‌ی while $O(n)$ بار اجرا می‌شود و در هر بار اجرای حلقه‌ی while پیدا کردن رأس w' در زمان $O(m)$ انجام می‌شود. پس پیچیدگی این الگوریتم از مرتبه‌ی $O(mn)$ است.
برای کاهش پیچیدگی، الگوریتم را با استفاده از صف اولویت[۲] یا هرم[۳] پیاده‌سازی می‌کنیم که شبه کد آن را در زیر می‌بینید:

Algorithm 3 Algorithm: DIJKSTRA WITH PRIORITY QUEUE

```
function DIJKSTRA3(Graph G, Vertex s, length {l_e}_{e∈E})
    for all v ∈ V do
        dist[v] ← ∞
    dist[s] ← 0
    H ← MAKEQUEUE(V) [using dist-values as keys]
    while H ≠ ∅ do
        w = EXTRACTMIN(H)
        for all edges (w, u) ∈ E do
            if dist[u] > dist[w] + l_wu then
                dist[u] = dit[w] + l_wu
                CHANGEKEY(H, u)
    return dist[·]
```

صف اولویت یا هرم نوعی داده ساختار است. در این داده ساختار، داده‌ها در یک درخت کامل دودویی ذخیره می‌شوند. در این درخت هر سطح از چپ به راست پر می‌شود و هر سطح باید قبل از ورود داده‌ها به سطح بعدی پر شود. هم

[۲] priority queue
[۳] heap

چنین مقدار کلید هر گره کوچک‌تر یا مساوی مقدار کلید فرزندان آن گره است. در نتیجه ریشه‌ی درخت همواره شامل کوچک‌ترین داده است.
روی صف اولویت اعمال $\textsc{ChangeKey}$، $\textsc{ExtractMin}$ و $\textsc{Insert}$ تعریف می‌شوند که به صورت زیر کار می‌کنند.

- عمل $\textsc{Insert}$ داده‌ی جدید را در اولین مکان موجود در پایین درخت قرار می‌دهد و تا وقتی که از پدر خود کوچکتر است، با والدش جابه‌جا می‌شود. تعداد جابه‌جایی‌ها حداکثر برابر ارتفاع درخت یعنی $\lfloor \log_۲ n \rfloor$ خواهد بود (تعداد داده‌های درخت را n در نظر می‌گیریم).
- عمل $\textsc{ChangeKey}$ مشابه عمل $\textsc{Insert}$ است، با این تفاوت که داده‌ی مورد نظر از قبل در درخت است.
- عمل $\textsc{ExtractMin}$ ریشه‌ی درخت را حذف می‌کند و سمت راست‌ترین گره در پایین‌ترین سطح درخت را به جای آن قرار می‌دهد و تا وقتی که از هر یک از فرزندانش بزرگ‌تر است با فرزند کوچک‌ترش جابجا می‌شود.

در صف اولویت پیچیدگی همه‌ی اعمال $\textsc{ChangeKey}$، $\textsc{ExtractMin}$ و $\textsc{Insert}$ از مرتبه‌ی $O(\log n)$ می‌باشد و این اعمال به ترتیب به تعداد m، n و n در الگوریتم دایکسترا انجام می‌شوند. پس پیچیدگی این الگوریتم از مرتبه‌ی $O((n+m)\log n)$ می‌باشد که به صورت $(m \geqslant n-۱)$ ساده می‌شود.
دقت کنید که اگر بجای استفاده از داده ساختار هرم از یک فهرست پیوندی[۴] استفاده می‌کردیم (که عمل $\textsc{ExtractMin}$ را در $O(n)$ و اعمال $\textsc{Insert}$ و $\textsc{ChangeKey}$ را در $O(۱)$ انجام می‌دهد) به پیچیدگی $O(n^۲)$ دست می‌یافتیم.
می‌توانیم الگوریتم دایکسترا را با استفاده از صف اولویت d تایی پیاده‌سازی کنیم. داده ساختار صف اولویت d تایی مشابه داده ساختار صف اولویت است، با این تفاوت که هر گره d تا فرزند دارد. اعمال $\textsc{ChangeKey}$، $\textsc{ExtractMin}$ و $\textsc{Insert}$ در صف اولویت d تایی مشابه اعمال متناظر در صف اولویت هستند.
پیچیدگی اعمال $\textsc{ChangeKey}$، $\textsc{ExtractMin}$ و $\textsc{Insert}$ در صف اولویت d تایی از به ترتیب از مرتبه‌ی $O(\log_d n)$، $O(d\log_d n)$ و $O(\log_d n)$ است و این اعمال به ترتیب به تعداد m، n و n انجام می‌شوند. پس پیچیدگی الگوریتم در این پیاده‌سازی از مرتبه‌ی $O((nd+m)\log_d n)$ می‌باشد که به ازای $d \approx \frac{m}{n}$ بهینه است.

$$O((nd+m)\log_d n) = O((nd+m)\frac{\log n}{\log d}) = O(m\frac{\log n}{\log \frac{m}{n}}) = O(m\log_{\frac{m}{n}} n)$$

در صورتی که الگوریتم دایکسترا را با یک داده ساختار پیچیده‌تر به نام هرم فیبوناتچی[۵] پیاده‌سازی کنیم پیچیدگی آن به مرتبه‌ی $O(n\log n + m)$ کاهش می‌یابد.

[۴] linked list
[۵] Fibonacci heap