دانشکده علوم ریاضی
دانشگاه صنعتی شریف

الگوریتم‌های خلاصه‌سازی برای مه‌داده

محمدهادی فروغمنداعرابی
پاییز ۱۳۹۹

خلاصه‌سازی گراف

جلسه هشتم

نگارنده: فاطمه کرمانی

۱ خلاصه‌سازی حافظه در الگوریتم پیدا کردن جنگل فراگیر

هدف در این بخش خلاصه‌سازی حافظه‌ی مورد نیاز برای اجرای الگوریتم‌های مختلف روی گراف لازم است. در این جلسه، الگوریتم یافتن جنگل فراگیر روی یک گراف بررسی می‌شود. جنگل فراگیر بزرگ‌ترین زیرگراف یک گراف است که دوری ندارد. مولفه‌های هم‌بندی جنگل فراگیر همان مولفه‌های هم‌بندی گراف است. این مسئله، تعدادی زیرمسئله دارد که در ادامه به آن‌ها می‌پردازیم.

۱.۱ بازیابی بردار kـ تنک

تعریف ۱.

به بردار $x \in \mathbb{R}^n$، kـ تنک می‌گوییم اگر $|\operatorname{support}(x)| < k$. $\operatorname{support}(x) \subseteq [n]$ برابر است با مجموعه‌ی اندیس‌های i که $x_i \neq 0$.

زیرمسئله‌ی بازیابی kـ تنک برای xهای kـ تنک، مقدار دقیق x را برمی‌گرداند (اندیس‌های $\operatorname{support}(x)$ و مقادیرشان) و در بقیه‌ی حالات می‌تواند جواب دل‌خواه دهد.

می‌توان نشان داد این مسئله به صورت قطعی با $O(k\log(nM))$ بیت حل می‌شود، اگر ورودی‌های مسئله از M بزرگ‌تر نشوند و مقادیر آپدیت‌ها اعداد صحیح هستند. یک روش معمول برای این کار این است که از یک مپ [۱] استفاده کنیم. در این صورت به اندازه‌ی خانه‌های غیر صفر بیت حافظه نیاز داریم. در این حالت فرض کنید تعداد بیت‌های ناصفر در طول زمان کم و زیاد بشوند. در روش معمول اگر یک بار این تعداد، از k بیش‌تر شود الگوریتم دیگر درست جواب نخواهد داد حتا اگر بعد از آن با به‌روزرسانی‌های بعدی این تعداد کم‌تر از k بشود.

[۱] ترجمه‌ش چی می‌شه؟

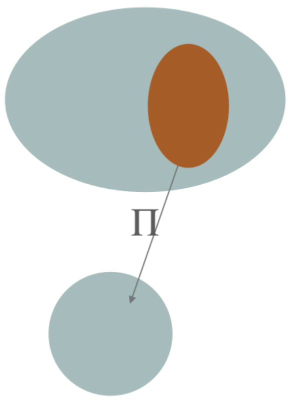
شکل ۱

هدف این است که ساختمان داده‌ای بسازیم که در مواقعی که در طول انجام به‌روزرسانی‌ها هرگاه آرایه‌ی لحظه‌ای kـ تنک بود، جواب درست را با احتمال ۱ درست برگرداند و در صورتی که kـ تنک نبود، جواب دل‌خواه برگرداند.

ایده‌ی طراحی این ساختمان داده این است که به جای خود آرایه‌ی x، یک خلاصه‌ساز خطی از آن را نگه داریم. این خلاصه‌ساز را Π می‌نامیم. به جای x، Πx را ذخیره کنیم. ابعاد Π، $m \times n$ است و $m \ll n$.

ویژگی‌ای که Π باید داشته باشد این است که قابل بازیابی باشد. برای این منظور، اگر در شکل ۴ بیضی بزرگ‌تر $\mathbb{R}^n$، بیضی کوچک‌تر $\mathbb{R}^m$ و بیضی نارنجی رنگ بردارهای kـ تنک باشد، Π هیچ دو عضو نارنجی را به یک موجود مپ نکند. به عبارت دیگر یکی از دو شرط معادل زیر را داشته باشیم:

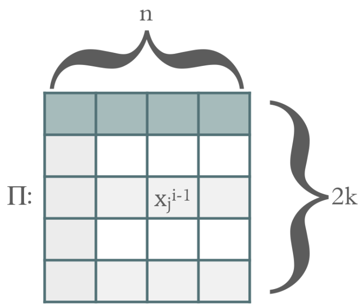
- برای هر دو x, y kـ تنک، $\Pi x \neq \Pi y$
- برای هر z $2k$ـ تنک، $\Pi z \neq 0$



شکل ۲

توضیح: دو شرط بالا معادلند چون اگر $x - y$ را برای x و y که هر دو kـ تنک هستند را در نظر بگیرید، حاصل یک بردار $2k$ـ تنک است. همچنین شرط دوم به این معنی است که هیچ ترکیب خطی شامل $2k$ تا از ستون‌های Π نباید صفر بشود. برای ساختن چنین Π ای یک کاندید مناسب، مطابق شکل ۳ است. در ادامه نشان می‌دهیم که ماتریس متشکل از انتخاب هر $2k$ ستون از Π، ناتبهگون است. نکته: اگر به جای $\mathbb{R}^n$، F_p که میدان اعداد اول pتایی است می‌گرفتیم برای یافتن وارون کار بسیار ساده‌تری در پیش داریم. در این حالت می‌توانیم تمام حالت‌های F_p^m را بگردیم و ببینیم کدام یکی جواب مورد نظر ماست. پس فرض می‌کنیم اعدادمان اعضای F_p هستند و $n < p$ است و همچنین اعدادی که به عنوان ورودی داریم از p کوچک‌تر است.

$x_1, \dots, x_n \in F_p$ در نظر می‌گیریم و تعریف می‌کنیم برای هر i, j $j \in [n]$, $i \in [2k]$، $\Pi_{i,j} = x_j^{i-1}$. (یک گزینه‌ی منطقی برای $x_1, \dots, x_n$، $1, 2, \dots, n$ است.)



شکل ۳

گزاره ۲. دترمینان ماتریس واندرموند [2] فرض کنید $A \in F^{r\times r}$ را تعریف کنیم $A_{i,j} = x_j^{i-1}$ برای $i,j \in [r]$ برای یک میدان F. آنگاه داریم:

$$\det(A) = \prod_{1\leq i<j\leq r} (x_i - x_j)$$

طبق گزاره‌ی ۲ هر $2k$ ستون از Π را که انتخاب کنیم دترمینان ناصفر دارد.

برای بازیابی kـ تنک، به جای بردار x، Πx را ذخیره می‌کنیم. برای به‌روزرسانی، فرض کنیم درایه‌ی iام x را به اندازه ی Δ می‌خواهیم تغییر دهیم. در این صورت حاصل برابر است با $\Pi x + \Pi^i \Delta$. Π^i ستون iام Π است.

برای بازیابی x، چون تعداد بردارهای kـ تنک خیلی زیاد نیست، می‌توانیم همه‌ی آن‌ها را چک کنیم به این صورت که پیدا کنیم که ضرب کدام یک از آن‌ها در ماتریس Π برابر مقدار ذخیره شده هست که آن را به عنوان جواب برگردانیم. این الگوریتم به صورت یکتا جواب درست را به دست می‌آورد چون تبدیل Π یک به یک است.

تحلیل حافظه

حافظه‌ی مورد نیاز $2k \log p$ است.

پس زیر مسئله‌ی اول مورد نیاز را بررسی کردیم. در بخش بعد، زیرمسئله‌ی دومی که با آن برخورد می‌کنیم را بیان می‌کنیم.

۲.۱ پیدا کردن support

تعریف مسئله

فرض کنید برای میدان F می‌خواهیم دو عملیات زیر را با حافظه‌ی کم و به صورت احتمالی داشته باشیم:

- update() به روزرسانی ($\Delta = \pm 1$)
- query() $i \in \text{support}(x)$ را برگرداند

ایده‌ی اصلی این است که $\log n$ ساختمان داده در نظر بگیریم و هر خانه‌ی حافظه را به یکی از این ساختمان داده‌ها متناظر کنیم. به طوری که احتمال این‌که یک خانه‌ی حافظه را به یک ساختمان داده متناظر کنیم به طور نمایی از احتمال این‌که به ساختمان داده‌ی بعدی متناظر کنیم بیش‌تر باشد. به عبارت دقیق‌تر اگر تابع تناظر این دو را درهم‌ساز h در نظر بگیریم داشته باشیم $\mathbb{P}(h(i) = j) = \frac{1}{2^j}$. برای این ساختمان داده‌ها یک عدد ثابتی در نظر می‌گیریم که در صورتی که تعداد اعدادی که به آن‌ها متناظر می‌کنیم کمتر از آن عدد ثابت باشد، آن ساختمان داده خوب کار می‌کند. در غیر این صورت عدد دل‌خواه خروجی می‌دهد.

با توجه به این‌که در هر ساختمان داده، به صورت حدودی نصف قبلی خانه‌ی حافظه وارد می‌شود، یکی از این ساختمان‌داده‌ها پیدا می‌شود که در بازه‌ی مناسب (تعداد ناصفر اما کم‌تر از آن عدد ثابت فرض شده) ورودی دارد.

الگوریتم

- شروع
  - تابع درهم‌ساز نمایی h را تعریف می‌کنیم. $pr(h(i) = j) = \frac{1}{2^j}$.

[2] Vandermonde Matrix

– A_j را برای هر $j \in [\log n]$ یک الگوریتم بازیابی kـ تنک که در بخش قبل داشتیم قرار می‌دهیم.

* $k = C\log(1/\delta)$

• به‌روزرسانی

– $\text{update}(i,\Delta) = A_{h(i)}.update(i,\Delta)$

• جواب

– اولین بزرگ‌ترین j که A_j خالی نبود: یکی از اعضای A_j.

1	A_1
2	A_2
...	...
log n	$A_{\log n}$

شکل ۴

۱.۲.۱ تحلیل الگوریتم

در صورتی که $x = 0$ در این صورت تمام ساختمان داده‌ها صفر برمی‌گردانند و پاسخ درست است. اگر $|\text{support}(x)| \le k$ باشد، در بدترین حالت تمام اعضای x در یک ساختمان داده قرار می‌گیرند در این حالت نیز جواب درست را برمی‌گرداند.

پس حالتی را در نظر می‌گیریم که $t := |\text{support}(x)| \le k$. $T_j =:$ تعداد x_iهایی که در A_j قرار گرفته‌اند. امید ریاضی T_j را داریم $\mathbb{E}T_j = t/2^j$. j^* را طوری انتخاب می‌کنیم که داشته باشیم برای c مناسب $1 < c < C$، $c\log(1/\delta) \le t/2^{j^*} < 2c\log(1/\delta)$. j^* با احتمال خوبی اندیس همان ساختمان داده‌ای است که جواب درست را به ما خواهد داد.

دو اتفاق خوب و تضمین کننده وجود دارد برای این‌که j^* پاسخ درست باشد.

• $\mathcal{E}_1 : \max_{j \ge j^*} T_j \le k$

• $\mathcal{E}_2 : T_{j^*} \ge 1$

پس اگر احتمال این‌که این دو اتفاق بیفتد بالا باشد، در این صورت الگوریتم مطلوب است. احتمال اتفاق نیفتادن $\mathcal{E}_2$ و $\mathcal{E}_1$ را حساب می‌کنیم و طبق کران تجمعی [3] احتمال اتفاق افتادن هر دو را با کران بالای جمع حاصل تخمین می‌زنیم.

$$\mathbb{P}(\neg\mathcal{E}_1) \le \sum_{j=j^*}^{\log n} \mathbb{P}(T_j > k) = \sum_{j=j^*}^{\log n} \mathbb{P}(T_j > (k2^j/t).\mathbb{E}T_j) < (k2^j/t)^{C'k} \quad (1)$$

که تساوی با ضرب و تقسیم کردن در $\mathbb{E} = t/2^j$ برقرار است و نامساوی آخر طبق کران چرنوف با $\lambda > 2e - 1$ اتفاق می‌افتد:

گزاره ۳. کران چرنوف

$$\mathbb{P}(X > (1+\lambda)\mu) < \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^\mu < \lambda^{-\Omega(\lambda\mu)} < \sqrt{1/(1+\lambda)}^{-\Omega(\lambda\mu}$$

از آن‌جایی که داشتیم $k = C\log(1/\delta)$ و $c\log(1/\delta) \le t/2^{j^*} \le 2c\log(1/\delta)$ در نتیجه $C/2c < k2^{j^*}/t \le C/c$ پس در واقع اندازه‌ی $k2^{j^*}$ بزرگ است. برای δ کوچک و توان به اندازه‌ی کافی بزرگ داریم

$$(k2^j/t)^{C'k} < (C/2c)^{-C'k} < \delta/2 \quad (2)$$

[3] union bound

که با حاصل نامساوی ۱ در کل نتیجه می‌دهد

$$\mathbb{P}(\neg\mathcal{E}_1) < \delta/2$$

از استدلال‌های بالا نتیجه می‌گیریم احتمال این‌که $\mathcal{E}$ اتفاق نیفتد زیاد نیست.

حال احتمال رخداد $\mathcal{E}$ را بررسی می‌کنیم. می‌دانیم به طور متوسط چند تا از ورودی‌ها در ساختمان داده‌ی iام قرار می‌گیرد احتمال قرار گیری یکی از ورودی‌ها در j^* برابر $1/2^{j^*}$ است. بنابراین احتمال این‌که هیچ ورودی‌ای نداشته باشد برابر است با

$$\mathbb{P}(\neg\mathcal{E}_2) = (1 - 1/2^{j^*})^t < \exp(-t/2^{j^*}) < \delta^c < \delta/2 \tag{3}$$

که مجددا در این نتیجه‌گیری از $c\log(1/\delta) \leq t/2^{j^*} \leq 2c\log(1/\delta)$ استفاده کردیم و شرط کوچک بودن δ به اندازه‌ی کافی.

از ۲ و ۳ و استفاده از کران اجتماع احتمال بد بودن کم‌تر از δ است.

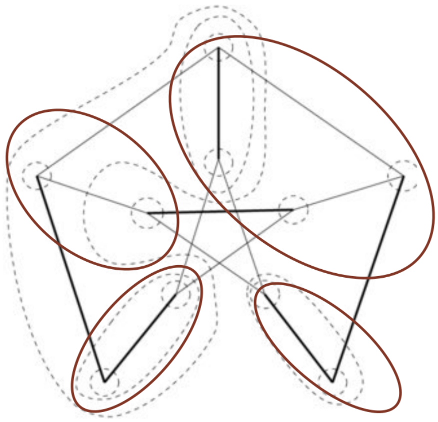
حافظه‌ی مورد نیاز

الگوریتم با $O(\log(1/\delta)\log^2 n)$ بیت حافظه اجرا می‌شود زیرا $\log n$ تا بازیابی kـتنک استفاده می‌شود که هر کدام یک بردار $2k$ تایی استفاده می‌کنند $2k \simeq \log(1/\delta)$ و حافظه هر کدام $\log p \simeq \log n$ بیت است.

۲ مسئله‌ی جنگل فراگیر

الگوریتمی برای پیدا کردن یک جنگل فراگیر در یک گراف متغیر [4] است.

- ورودی: یک گراف
- خروجی: جنگل فراگیر (با احتمال $(1 - 1/n^b)$)



شکل ۵

از ساختمان داده‌ای که در قسمت قبل معرفی کردیم استفاده می‌کنیم برای حل این مسئله.
عملیات ساختمان داده‌ی مورد نظر:

- اضافه‌ی یال
- حذف یال
- یک زیرجنگل فراگیر را برگردان

[4] dynamic

الگوریتم غیر جریانی

قبل از بررسی الگوریتم در حالت کلی، فرض می‌کنیم گراف غیر متغیر است. افرازی از راس‌ها را ذخیره می‌کنیم. در ابتدا هر راس در کلاس هم‌ارزی جداگانه‌ای قرار داد. هر بار یک یال خروجی از هر مجموعه را انتخاب می‌کنیم و دو مجموعه‌ی دو سر یال‌ها را با هم ادغام می‌کنیم و یال‌های انتخاب شده را در جنگل حاصل قرار می‌دهیم. هر بار تعداد مجموعه‌های افراز نصف می‌شود. پس بعد از $\log n$ مرحله یک جنگل فراگیر داریم. برای یک مثال از این الگوریتم شکل ۵

۱.۲ روش خلاصه‌سازی AGM

برای هر راس u یک بردار x_u تعریف می‌کنیم. این بردار نشان می‌دهد که راس u به چه راس‌هایی یال دارد.

- اگر یال v–u وجود داشت آن‌گاه

$$f(x) = \begin{cases} ۱, & u < v \text{ اگر} \\ -۱, & v < u \text{ اگر} \end{cases}$$

- اگر یال v–u وجود نداشت $x_u[v] = ۰$

برای مجوعه‌ی A که زیر مجموعه‌ای از راس‌های گراف است تعریف می‌کنیم $X_A := \sum_{u\in A} x_u$. ویژگی X_A این است که اگر دو سر یک یال در A قرار بگیرد، جمع اعداد متناظر دو سر یال با هم ساده می‌شوند. طبق این ویژگی، $\text{support}(X_A)$ یال‌هایی را مشخص می‌کند که بیرون مجموعه‌ی A هستند. پس برای هر کدام از مجموعه‌های A، با اجرای یک بار الگوریتم $\text{support-find}(X_A)$ می‌توانیم طبق توضیحات الگوریتم که در بالا ارائه شد، یک یال پیدا کنیم و مجموعه‌های دو سر یال را با هم ادغام کنیم و در صورتی که با اضافه کردن یالی دوری ایجاد نمی‌شد آن یال را به مجموعه‌ی یال‌های جنگل فراگیر مورد نظر اضافه می‌کنیم. و X_A را به روز رسانی می‌کنیم.

- A ها را برای شروع تک راس‌های گراف قرار بده.
- برای هر X_A یک بار $\text{support-find}(X_A)$ را فراخوانی کن و برای هر مجموعه یک یال پیدا کن. با k و $\delta < ۱/(Rn^{۱+b})$
- برای R بار
  - برای هر مجموعه‌ی A، یک یال را پیدا کن.
  - برای هر یال: دو مجموعه‌ی A و B که دو سر یال هستند را ادغام کن.
  - $X_A + X_B = X_{A\cup B}$
  - $\text{support-find}()$ های دو مجموعه را ادغام کن.

تحلیل حافظه

برای این الگوریتم R بار از support-find استفاده می‌کنیم. برای ذخیره‌ی اطلاعات مربوط به گراف درsupport-find، n بیت حافظه نیاز دارد. در کل این الگوریتم از $O(nR\log^۳ n)$ بیت حافظه استفاده می‌کند.