ISSN: 2745-3673
http://www.Rcsj.ir

مجله‌ی علمی پژوهش در علوم رایانه (سال چهارم)
شماره ۱۶ / زمستان ۱۳۹۸ / ص ۶۵- ۵۵

بررسی الگوریتم‌های موازی برای جستجوی اول عمق گراف

حامد بابایی[1]، جابر کریم پور[2]، امید جوانشیر[3]

[1] مربی، گروه مهندسی کامپیوتر، دانشگاه آزاد واحد اهر، اهر.

[2] دانشیار، گروه علوم کامپیوتر، دانشکده ریاضی، دانشگاه تبریز، تبریز

[3] گروه مهندسی کامپیوتر، دانشگاه آزاد واحد اهر، اهر.

نام نویسنده مسئول:

حامد بابایی

چکیده

بررسی وجود الگوریتم‌های موازی برای مسـأله‌ی DFS در گراف‌ها منجربه ایجاد تفکرات متعددی در این زمینه شده است. اولین الگوریتم برای DFS را Tarjan و Tarjan –Hopcroft ارائه نمودند، و دیگران نیز سعی در ارائه‌ی الگوریتم‌های موازی در این حوزه کوشیدند. در این مقاله ابتدا بررسی ذاتا" ترتیبی الگوریتم DFS توسط REIF صورت می‌پذیرد و سپس بررسی الگوریتم NC برای گراف‌های مسطح توسط Smith و انتها نیز بررسی الگوریتم RNC توسط Anderson & Aggarwal مورد تحقیق قرارمی‌گیرد. البته در انتهای مقاله نتایج بررسی هر کدام از الگوریتم‌های موازی برای جستجوی اول عمق گراف به تفکیک ارائه شده است.

واژگان کلیدی: الگوریتم موازی، جستجوی اول عمق گراف، مسائل NP-کامل و مسائل P-کامل.

ISSN: 2745-3673
http://www.Rcsj.ir

مقدمه

ابتدا در این بخش به توضیحات مقدماتی در مورد گراف و مفهوم پیمایش DFS می‌پردازیم. یک گراف G = (V, E) با مجموعه‌ی رئوس در V={1,2,…,n} و مجموعه‌ی لبه‌ها در E می‌باشد و مفهوم جهت‌دار بودن گراف G یعنی یال‌ها بر اساس ضرب دکارتی به صورت زوج‌های مرتب قرار می‌گیرند، {(v,u),(u,v)}∈E و مفهوم غیر جهتار بودن گراف G یعنی یال‌ها به صورت زوج‌های غیر مرتب می‌باشند، {v,u}={u,v} ∈E. پیاده‌سازی گراف G با لیست‌های مجاورتی برای افزایش کارایی الگوریتم DFS از مرتبه‌ی O(n+e) می‌باشد که در اینجا |V|=n و |E|=e است. ساختمان داده‌ای که از DFS پشتیبانی می‌کند پشته است و DFS پیمایش گراف G را از ریشه شروع کرده و تا زمانی که پشته خالی نشده و کلیه‌ی رئوس گراف ملاقات نشده‌اند کار جستجو ادامه بدهیم. در زیر الگوریتم DFS به صورت بازگشتی برای پیمایش گراف G ارائه شده است [1, 2, 3, 4, 5, 6, 7].

```
procedure DFS(v)
begin
        mark v as visited;
        while there is an unmarked vertex w
                adjacent to v do
                add (v, w) to T;
                DFS(w)
        end { while }
end { DFS }
```

حال هدف کاهش زمان اجرای الگوریتم فوق به صورت پایین‌تر از زمان خطی و اجرای آن الگوریتم بر روی چندین پردازنده در حالت موازی می‌باشد.

الگوریتم موازی اول عمق گراف REIF

این بخش نزدیکی زیادی با روش حالت سریال دارد چون در اینجا اثبات ذاتا" ترتیبی الگوریتم DFS توسط REIF مورد نظر خواهد بود.

این بخش گواه می‌کند که DFS به صورت موازی دارای مرتبه‌ی زمانی و فضایی قطعی معادل (log n)^C (C > 0)، نمی باشد. البته DFS-ORDER و UDFS-ORDER می‌توانند توسط یک ماشین تورینگ قطعی با فضای (log n)^o(1) پذیرش شوند. مفهوم DFS-ORDER، همان DFSای است که بر روی ماشین RAM به صورت سریال انجام می‌شود و دارای زمان O(n+e) و پیاده‌سازی شده با لیست مجاورتی است، و UDFS-ORDER که به یک گراف جهت‌دار با ریشه‌ی معلوم و رئوس u,v∈V که رأس u قبل از v ملاقات شده است اشاره دارد. از مدل‌های محاسباتی می‌توان به مدارات بولین اشاره نمود [1, 2].

در [1, 2] یک مدار بولین a با n ورودی برای یک گراف غیر دوار جهت‌دار متناهی با گره‌های برچسب شده می‌باشد که n گره ورودی و یک گره خروجی دارد. گره‌های ورودی با متغیرها و مکمل (منفی) متغیرها برچسب زده می‌شوند و گره‌های دیگر با یک عملیات بولین همچون (AND) ∧ و (OR) ∨ بر چسب خورده و نیز حداقل یک مسیر از گره ورودی به گره خروجی وجود دارد. fan-in در a بزرگترین درجه‌ی هر گره می‌باشد، اندازه‌ی a تعداد گیت‌ها را شامل می‌شود و عمق a که طول بزرگترین مسیر از گره ورودی به گره خروجی می‌باشد. Σ یک الفبای متناهی است و L زبانی است که $L \subseteq \Sigma^n$ می‌باشد و یک مدار بولین C زمانی زبان L را می‌پذیرد که $w \in \Sigma^n$ باشد و مدار C خروجی‌های ۱ را بر روی wهایی که $w \in L$ باشند را تولید کند. اندازه‌ی پیچیدگی مدار برای زبان L کوچکترین اندازه‌ی مداری است که زبان L را می‌پذیرد. برای زبان $L^* \subseteq \Sigma^*$ اندازه‌ی پیچیدگی مدار L^* یک تابع f به صورت f(n) است که اندازه‌ی پیچیدگی مدار $L^n = L^* \cap \Sigma^n$ می‌باشد و ما نیاز به مدارات بولین $< \alpha >$ داریم تا بتواند رشته‌هایی از یک طول خاص را بپذیرد.

نکته: یک مدار از خانواده‌ی $< \alpha_n >$ اگر n معلوم باشد پس یک ماشین تورینگ قطعی می‌تواند $< \alpha_n >$ را در فضای O(log n) تولید کرده و شرایط را ارضاء کند.

L و 'L زبان‌هایی‌اند که بر روی الفبای متناهی Σ تعریف می‌شوند. در $L \leq \log L'$ که 'L فضای لگاریتمی کاهش پذیر به L می‌باشد و برای یک تابعی چون f دو شرط: ۱- برای هر $w \in \Sigma^*$ و $w \in L'$ آنگاه f(w)=L می‌باشد و ۲- اگر f یک تابع قابل محاسباتی در فضای لگاریتمیک باشد توسط ماشین تورینگ قطعی پذیرش می‌شود.

ISSN: 2745-3673
http://www.Rcsj.ir

کلاس P: کلاسی که زبان‌هایی در زمان چند جمله‌ای را توسط ماشین‌های تورینگ قطعی می‌پذیرد. زبان L به عنوان P- کامل است زمانی که L∈P باشد و برای هر L' ∈P باید شرط $L \leq \log L'$ برقرار باشد.

نکته: قابلیت کاهش‌پذیری فضای لگاریتمی یک رابطه‌ی تعدی می‌باشد که به صورت: اگر L∈P و L' ∈P پس آنگاه $L \leq \log L'$.

تعریف مدار بولین:

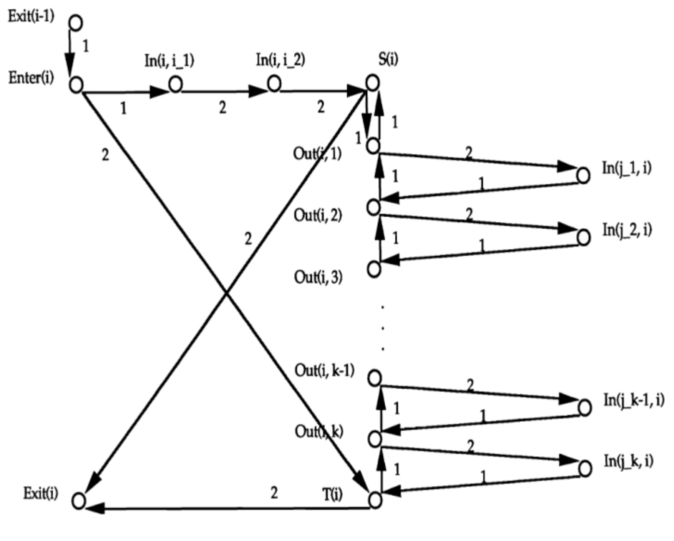
B= $(B_0, B_1, \dots, B_n)$ که برای هر B_i مقدار true و false یا یک عبارت عملیاتی بولین به صورت $op(B_{i_1}, B_{i_2})$ که $i_1, i_2 < i$ می‌باشد. $value(B_i) = B_i$ اگر B_i یک مقدار true یا false را داشته باشد و برای $B_i = (B_{i_1}, B_{i_2})$ که به $value(B_i) = op(value(B_{i_1}), value(B_{i_2}))$ تبدیل می‌گردد و در انتها $value(B) = value(B_n)$ خواهد بود.

قضیه ۱: مسأله‌ی مقدار مدار یک مسأله‌ی P-کامل است (یک مدار بولین B وجود دارد که باید value(B)=true بررسی گردد).

گزاره‌ای که قضیه‌ی فوق را اثبات می‌کند، اگر مدارات فقط به عملیات بولین محدود شده باشند پس مسأله مقدار مدار به همان صورت P-کامل باقی خواهد ماند که $B_0 = true$ و $B_i = \backsim (B_{i_1} \vee B_{i_2})$ (تابع NOR و NAND به عنوان توابع کامل می‌باشند) و $1 \leq i \leq n$ می‌باشد. (در شکل ۱ بر روی گراف جهت‌دار G_i قضایا و گزاره‌های گفته شده همگی اعمال می‌گردد تا درخت DFS در نهایت از روی این گراف استخراج گردد)

اثبات: آیا DFS-ORDER در کلاس P-کامل می‌باشد؟

کلیه‌ی پیش فرض‌های قبلی و نیز $j_1, j_2, \dots, j_k > i$ وجود دارند که به عنوان زیر دنباله‌های عملیات بولین $B_{j_1}, B_{j_2}, \dots, B_{j_k}$ می‌باشند. گراف جهت دار G_i با $V_i = \begin{Bmatrix} Enter(i), In(i,i_1), In(i,i_2), S(i), Our(i,1), \\ \dots, Out(i,u), T(i), Exit(i) \end{Bmatrix}$ و لبه‌هایی که در شکل ۱ می‌باشد وجود دارند که به صورت زیر نشان داده می‌شود.



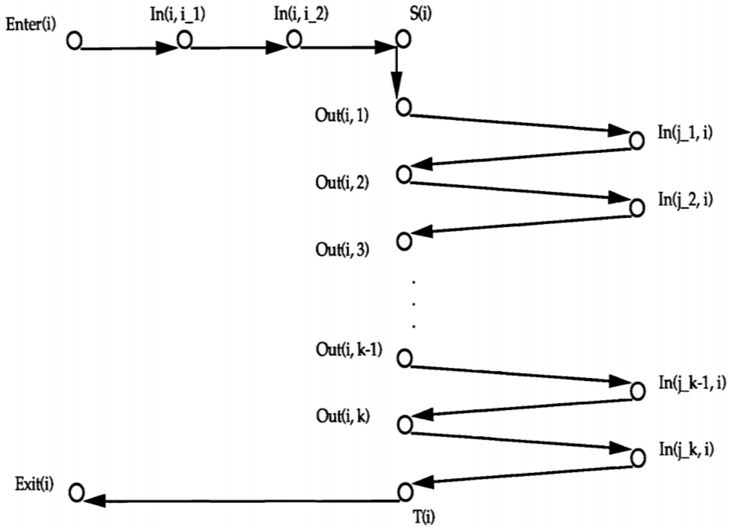
شکل ۱: گراف جهت دار G_i برای عملیات بولین $B_i = \backsim (B_{i_1} \vee B_{i_2})$ در B، زمانی که B_i در عملیات $B_{j_1}, B_{j_2}, \dots, B_{j_k}$ متعاقبا رخ می‌دهد.

حال انجام عمل پیمایش DFS بر روی G_i که از رأس ENTER(i) شروع می‌گردد و فرض می‌کنیم که هیچ رأسی در G_i قبلا" ملاقات نشده است. شکل ۲ نشان دهنده‌ی نتیجه‌ی پیمایش DFS را معلوم می‌کند و البته اگر رئوس $In(i, i_1),\ In(i, i_2)$ قبلا" ملاقات شده باشند و هیچ رأس دیگری از G_i ملاقات نشده باشد پیمایش از رأس ENTER(i) شروع گردد پس شکل ۳ نشان دهنده‌ی نتایج

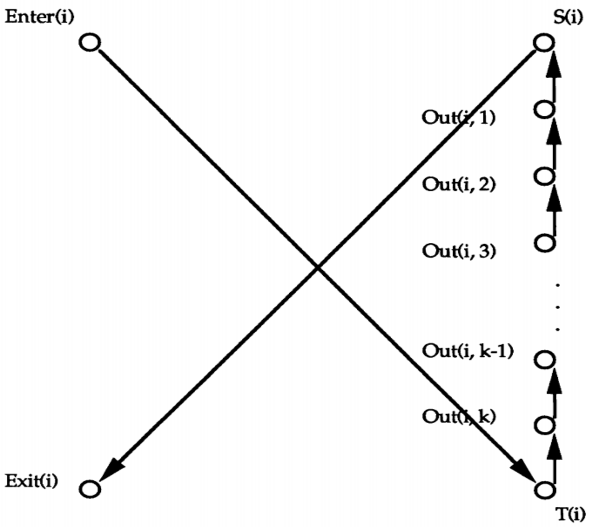
مجله‌ی علمی پژوهش در علوم رایانه ، شماره ۱۶، زمستان ۱۳۹۸، ص ۶۵- ۵۵
ISSN: 2745-3673
http://www.Rcsj.ir

پیمایش DFS به صورت لبه‌های درخت DFS خواهد بود. درهر یک از دو مورد یا همه‌ی رئوس از G_i ملاقات شده است مگر رأس $In(i,i_2)$ و یا اگر i<n باشد پس DFS از رأس ENTER(i+1) شروع می‌گردد.

توجه: در شکل ۱ لبه‌ی (ENTER(i-1), ENTER(i)) ما بین دو گراف G_i و G_{i-1} می‌باشد و 2k لبه مابین گراف $G_{j_1},\dots,G_{j_k}$ و G_i وجود دارد و G گرافی است شامل اجتماع کلیه‌ی گراف‌های $G_1,\dots,G_n$ و اتصال لبه‌ها به صورت گفته شده (ریشه‌ی گراف G در EXIT(0) قرار دارد). شکل‌های ۲ و ۳ هر دو نشان دهنده‌ی نتیجه‌ی پیمایش گراف G_i می باشند که به صورت زیر نشان داده می‌شوند.



شکل ۲: لبه‌های درخت DFS برای G_i ، با فرض اینکه قبلا هیچ کدام از $In(i,i_1)$ یا $In(i,i_2)$ ملاقات نشده‌اند.



شکل ۳: لبه‌های درخت DFS برای G_i ، با فرض اینکه قبلا هر کدام از $In(i,i_1)$ یا $In(i,i_2)$ و یا هر دو ملاقات شده‌اند.

لم ۱: S(n) قبل از T(n) در گراف G ملاقات می‌شود اگر value(B)=true باشد.

اثبات: i عددی ثابت و $1\le i\le n$ می‌باشد. اگر رأس ENTER(i) در ابتدا ملاقات شده باشد پس فرض استقراء را برای هر i' که $1\le i'\le i$ است داریم: ۱-اگر $value(B_{i'})=true$ پس لبه‌های درخت DFS در $G_{i'}$ همانند شکل ۲ توصیف می‌شود و در غیر اینصورت ۲- لبه‌های درخت DFS از $G_{i'}$ همانند شکل ۳ خواهد بود.

مجله‌ی علمی پژوهش در علوم رایانه ، شماره ۱۶، زمستان ۱۳۹۸، ص ۶۵- ۵۵
ISSN: 2745-3673
http://www.Rcsj.ir

نتیجه: درخت DFS ایجاد شده شامل یک تک مسیر از لبه‌های درخت DFS است که رأس شروع در EXIT(0) و خاتمه در ENTER(i) می‌باشد.

توجه: اگر $B_i = \backsim (B_{i_1} \vee B_{i_2})$ باشد $value(B_{i_1}) = value(B_{i_2}) = false$ پس فرضیه‌ی استقراء برای $In(i, i_1)$ و $In(i, i_2)$ که لبه‌های درخت DFSای هستند که از قبل ملاقات نشده‌اند و به صورت شکل ۲ توصیف می‌گردد و اگر $value(B_{i_1}) \vee value(B_{i_2}) = true$ باشد پس هر یک از دو رأس $In(i, i_1)$ یا $In(i, i_2)$ قبلا" ملاقات شده‌اند و لبه‌های درخت DFS به صورت شکل ۳ ملاقات می‌گردند.

نتیجه: ساختن گراف جهت‌دار G برای مدار B به وسیله‌ی ماشین تورینگ قطعی با فضای O(log n) و زمان O(log n) در مدل محاسباتی PRAM صورت می‌گیرد.

قضیه ۲: P-کامل بودن DFS-ORDER,UDFS-ORDER:

اثبات: اگر G=(V,E) یک گراف جهت دار به صورت گفته شده باشد و گراف $G' = (V, E')$ یک زیر گراف غیر جهت دار از G باشد پس می‌توان لبه‌های غیرجهت‌دار {u,v}∈E را برای هر لبه‌ی (u,v) ∈E یا (v,u) جایگزین نمود.

برای هر رأس v∈V، $E_{v_{out}}$ و $E_{v_{in}}$ وجود دارند که در G' بعد از ملاقات v ابتدا لبه‌هایی را که در $E_{v_{out}}$ هستند را ملاقات می‌کنیم و این دقیقا" همان ترتیب ملاقات در G می‌باشد و سپس لبه‌هایی که در $E_{v_{in}}$ هستند را ملاقات می‌کنیم که می‌تواند با هر ترتیب ممکن صورت پذیرد. البته ریشه‌ی G' در EXIT(0) می‌باشد.

لم ۲: S(n) قبل از T(n) در G' ملاقات می‌شود اگر value(B)=true باشد.

اثبات: این لم به اثبات P-کامل بودن UDFS-ORDER اشاره می‌کند و اثبات این گفته نیز به P-کامل بودن DFS-ORDER اشاره دارد.

الگوریتم موازی NC اول عمق گراف Smith

این بخش الگوریتم NC را که smith برای گراف‌های مسطح غیر جهت دار ارائه داده است بررسی می‌کند. این الگوریتم اولین و سریعترین الگوریتم موازی ارائه شده می‌باشد [3].

تعاریف و قضایای ضروری

گزاره ۱: در [3]، G یک گراف متصل با اجزای دو اتصالی G_i و یک رأس مشخص x. برای هر G_i خود x_i نقطه‌ی جداساز G_i متصل به x توسط کوتاه‌ترین مسیر ممکن می‌باشد. اگر T_i یک درخت DFS از G_i با ریشه‌ی x_i باشد پس اجتماع T_i یک درخت DFS برای G خواهد بود که در r ریشه دارد. اگر G یک گراف باشد و P یک مسیر ساده در G باشد به طوری که G-P یک اجتماع متمایز از اجزای $\{G_i\}$ می‌باشد و یکی به آخر از P که به نام ریشه می‌باشد. e یک لبه در G است و e (لبه‌هایی است که مورد تماس مسیر P می‌باشند) لبه‌های مورد تماس در P است اگر یک رأسی که دقیقا" در e است در P نیز باشد. e لبه‌ی غیرضروری برای G_i است اگر e مورد تماس P بوده باشد (اما در اینجا لبه‌ی دیگری چون e' در G_i وجود دارد که آن مورد تماس P در یک نقطه‌ی دورتر از ریشه نسبت به نقطه‌ای که e مورد تماس P قرار گرفته است خواهد بود) در غیر این صورت e یال ضروری برای G_i می‌باشد. کاهش G به واسطه‌ی مسیرهای موجود در P می‌باشد که منجربه ایجاد یک گراف G' می‌شود که از حذف کلیه‌ی یال‌های غیر ضروری G به وجود می‌آید. (اجزای G-P و عدم حذف رئوس انتهایی) G' شامل اجتماعی از P است که یال‌های ضروری از G در بر می‌گیرد و اجزای $\{G_i\}$ گراف که نتیجه‌ی حذف کلیه‌ی رئوس P از G می‌باشد.

گزاره ی ۲: e_i یک لبه‌ی ضروری برای G_i است که G_i را به P وصل می‌کند و T_i یک درخت DFS از G_i است که در رأس پایانی e_i که در G_i می‌باشد ریشه دارد. اجتماع P که $\{e_i\}$ را به وجود می‌آورد و $\{T_i\}$ یک درخت DFS از G می‌باشد.

مجلـه‌ی علمی پژوهش در علوم رایانه ، شماره ۱۶، زمستان ۱۳۹۸، ص ۶۵- ۵۵

ISSN: 2745-3673
http://www.Rcsj.ir

اثبات: اگر $v_1 < v_2$ پس v_2 جد v_1 در T می‌باشد. سپس T یک درخت DFS است اگر برای هر لبه‌ی (v,u)∈E که u<v یا v<u باشد. e یک لبه در G است اگر e به دو رأس u,v در درخت T_i متصل باشد پس آن‌ها قابل مقایسه‌اند چون T_i درخت DFS مفروض می باشد. e نمی‌تواند دو درخت T_i و T_j را به هم وصل کند چون بر خلاف فرض می‌باشد (آنها شامل اجزای متمایز از G-P می‌باشند). $e = \{v_{T_i}, v_P\}$ یک درخت T_i را به P متصل می‌کند. e برای G_i غیر ضروری است – رئوس پایانی قابل مقایسه‌اند چون هر رأس در T_i یک نواده از v_P است و اگر e برای G_i ضروری باشد پس رئوس پایانی قابل مقایسه‌اند چون لبه‌های e و e_i هر دو مورد تماس P در $v_{P'}$ می‌باشند. (رأس دیگری از e_i یک نواده از $v_{P'}$ است و v_{T_i} یک نواده از رأس دیگ ر e_i می‌باشد)

G یک گراف متصل است و G_s شامل همه‌ی یال‌هایی در G است که دارای سیکل می‌باشند و G_t شامل همه‌ی یال‌های دیگر G می‌باشد که شامل سیکل نمی‌باشد. پس G به G_s و G_t تجزیه می‌شود که این کار در زمان $O(\log^2 n)$ به صورت موازی صورت می گیرد.

قضیه ۱: اگر G یک گراف مسطح باشد گراف دو اتصالی با n رأس را داریم پس یک الگوریتم موازی برای یافتن سیکل ساده در G با خاصیت آنکه زیرگراف‌های داخلی و خارجی دارای تعداد رئوس کوچکتر یا مساوی 2n/3 را داشته باشند. این الگوریتم در زمان $O(\log^3 n)$ اجرا می‌گردد و تعداد پردازنده‌ها معادل با $O(n^4)$ می‌باشد.

توصیف الگوریتم

ورودی: یک گراف مسطح غیرجهت‌دار G و یک رأس r به عنوان مدخل ورودی (ریشه) می‌باشد.

خروجی: یک درخت DFS برای گراف G که در r ریشه دارد.

روش :

۱- یک گراف مسطح دردسترس با زمان $O(\log^2 n)$.

۲- یافتن اجزای دو اتصالی با زمان $O(\log^2 n)$.

۳- یافتن G_s و G_t و افزودن G_t به T و الگوریتمی که بر روی G_s کار می‌کند.

۴- برای هر جزء دو اتصالی از G_s به طور موازی فرآیندهای زیر صورت می‌گیرند:

- برای C_s (اجزایی از G_s) یک سیکل افرازبندی را می‌یابیم که با زمان $O(\log^2 n)$ انجام می‌گیرد.
- یافتن یک مسیر متصل رأس ورودی به سیکل افرازبندی که با زمان $O(\log^2 n)$ انجام می‌گیرد.

الف). یافتن درخت پوشای C_s.

ب). درخت پوشا با رأس ورودی (که آن ریشه است) هدایت می‌شود.

ج). حذف همه‌ی شاخه‌های درخت پوشا که آن خارج از سیکل افراز بندی جهت‌دار می‌باشد.

د). انتخاب یک شاخه از درخت پوشای جهت‌دار که آن وارد سیکل افرازبندی شده و یک علامت را به سوی ریشه پخش می‌کند.این مسیری است که آن ریشه را به سیکل افرازبندی متصل می‌کند.

– حذف یک لبه از سیکل افرازبندی که آن مجاور با نقطه‌ای که مکان مسیر پیدا شده می‌باشد و مسیر P به T اضافه می‌گردد. این کار در زمان O(1) صورت می‌گیرد.

- یافتن مجموعه‌ی $\{C_i\}$ از اجزای متصل C-P که این کار نیز در زمان $O(\log^2 n)$ صورت می‌گیرد.

۵- برای هر C_i مجموعه‌ای از کلیه‌ی لبه‌هایی که با P در تماس‌اند را پیدا کرده و یال‌های ضروری را تعیین می‌کنیم که این کار در زمان O(log n) صورت می‌گیرد.

۶- انتخاب هر یک از یال های ضروری.

۷- مراحل ۳ و۴ را برای هر C_i به کار گرفته می‌گردد که v_i به عنوان مدخل (ورودی) مورد استفاده قرار می‌گیرد.

ISSN: 2745-3673
http://www.Rcsj.ir

هر یک از اجزایی که در مرحله‌ی ۴ پیدا می‌شوند رئوس‌شان کوچکتر یا مساوی 2n/3 می‌باشد.

تکرار مراحل ۳ و ۴ که در زمان O(log n) برابر انجام می‌گیرد و برای یک محدوده‌ی زمانی کلی به صورت $O(\log^3 n)$ می‌باشد.

الگوریتم موازی RNC اول عمق گراف Anderson & Aggarwal

این بخش در مورد بررسی الگوریتم RNC برای DFS گراف‌های غیر جهت‌دار عمومی می‌باشد که توسط Anderson & Aggarwal ارائه شده است [4, 5, 6, 7].

تعاریف: یک مسیر در گراف G=(V,E) که یک مجموعه‌ی مرتب شده از رئوس مجزا به صورت $p = p_1, p_2, \dots, p_k$ و لبه‌هایی به صورت $(p_i, p_{i+1}) \in E$ که $1 \le i \le k$ خواهد بود. Q یک مجموعه‌ای برای جداسازی رئوس با مسیرهای متمایز می‌باشد اگر بزرگترین جزء متصل از V-Q اندازه‌ای بیش از n/2 را داشته باشد. یک قطعه‌ی اولیه از گراف G یک زیر درخت ریشه دار 'T می‌باشد که درخت T را بسط می‌دهد.

مرور الگوریتم

یک گراف G با ریشه‌ی معلوم r وجود دارد و ساخت یک قطعه‌ی اولیه‌ی 'T که ریشه‌ای به نام r را دارد به طوری که بزرگترین جزء متصل از 'V- T اندازه‌ای بیش از n/2 را دارد.

اجزاء متصل از 'V- T که $C_1, C_2, \dots, C_p$ هستند و برای هر C_i یک رأس یکتا $x_i \in T'$ با بیشترین عمق که مجاور با برخی رئوس y_i از C_i می‌باشد برای هر C_i باید این دو رأس به صورت موازی پیدا نمود. ساختن یک درخت DFS برای C_i که در y_i ریشه دارد به صورت بازگشتی انجام شده و سپس درخت‌های ساخته شده به هم متصل شده و به 'T با یک لبه از x_i به y_i وصل می‌شوند.

نکته: اندازه‌ی مسأله‌ی بازگشتی بیش از نصف اندازه‌ی مسأله‌ی اصلی می‌باشد پس بنابراین زمان اجرای الگوریتم O(log n) می‌باشد که این همان زمان مورد نیاز برای ساختن 'T می‌باشد.

برای ساختن یک قطعه‌ی اولیه ابتدا یک جداساز به نام Q را باید داشته باشیم که در آن تعداد مسیرها کوچکتر از یک عدد ثابت می-باشد و این قسمت زمانبرترین قسمت الگوریتم می‌باشد.

ساختن یک جداکننده:

ساختن مجموعه‌ای از مسیرهایی با رئوس متمایز $Q = \{q_1, q_2, \dots, q_k\}$ (k کوچکتر از یک ثابت عددی می‌باشد) و بزرگترین جزء از V-Q که اندازه‌ای بیش از n/2 را دارد. یک روال به نام Reduce(Q) را ایجاد می‌کنیم که تعداد مسیرهای تقلیل یافته Q به واسطه‌ی یک کسر ثابت را تا زمانی که خاصیت تفکیک کنندگی حفظ شود نشان می‌دهد. مقدار اولیه‌ی Q فقط V می‌باشد و در هر بار فراخوانی Reduce که 1/12 مسیرها در Q کنار گذاشته می‌شوند پس فراخوانی در زمان O(log n) انجام می شود (و باید مطمئن بود که Q شامل بیش از ۱۱ مسیر را شامل نمی‌شود)

کاهش تعداد مسیرها

ایده: یافتن مجموعه‌ای از مسیرها با رئوس مجزا مابین جفت مسیرها در Q می‌باشد و سپس ادغام جفت مسیرها باهم دیگر.

مسیرها در Q به دو مجموعه‌ی L و S تقسیم می‌شوند که L مسیر بلند و S مسیر کوتاه می‌باشد .

یافتن مجموعه‌ای از مسیرهای P با رئوس مجزا و توجه به یک مسیری چون p∈P که p به یک مسیر $l \in L$ و یک مسیرs∈S متصل می‌باشد.

اگر نقاط پایانی از p، x و y باشند پس $l = l'xl''$ و $s = s'ys''$ خواهند بود. حال p برای ادغام s و l استفاده می‌گردد.

اگر $|s'| \ge |s''|$ پس l را توسط $l'xs'$ و s را توسط s'' جایگزین می کنیم در غیر اینصورت l توسط $l'xs''$ و s را توسط s' جایگزین می گردد و l'' در هر دو حالت حذف می‌گردد. (طول s حداقل تا نصف کاهش داده شده و می‌توان s را به طور کامل ادغام نمود تا s از S حذف گردد)

توجه: ادغام جفت‌های s و l به صورت موازی صورت می‌گیرد تا تعداد مسیرها در Q با یک کسر ثابت کاهش داده شود.

مجلـه‌ی علمی پژوهش در علوم رایانه ، شماره ۱۶، زمستان ۱۳۹۸، ص ۶۵- ۵۵
ISSN: 2745-3673
http://www.Rcsj.ir

P باید دو محدودیت را ارضاء کند که شامل:

۱- p باید تا حد امکان بزرگ باشد و ۲- طول قطعات حذف شده نیز تا حد امکان باید بزرگ باشند.

دو موردی که در اینجا می‌توانست منجربه راه اشتباه شوند عبارت‌اند از:

۱- حذف یک قطعه ممکن بود باعث ترکیب دو جزء متصل از V-Q گردد و خاصیت جداسازی نقض شود.

۲- ممکن بود مجموعه‌ای از مسیرهایی با رئوس متمایز را به حد کافی بزرگ نیستند را پیدا نمود (در واقع کاهش طول مسیرها در S صورت می‌گیرد).

اگر در الگوریتم DFS موارد فوق رخ ندهد پس ابتدا |Q|=K و نیز در شروع کار K/4 مسیر در L و 3K/4 مسیر در S قرار می‌گیرد و سپس یک مجموعه‌ای از مسیرهایی با رئوس مجزا پیدا می‌کنیم که $P = \{p_1, p_2, \dots, p_\alpha\}$ ایجاد می‌گردد. هدف از محدودیت ۱ یعنی تعداد مسیرهای پیدا شده نباید کمتر از K/12 باشد پس فرض همان |P| > K/12 می‌باشد (می‌توان طول حداقل K/12 مسیر(ها) را در S به نصف کاهش داد) یافتن و ادغام مسیرها 9log n برابر بزرگتر قبل از خالی شدن S خواهد بود و مراحل تا زمانی که 11K/4 =< |Q| است ادامه می‌یابد.

مورد ۱: $\hat{L}$ مسیرهایی در L است که در P متصل نشده‌اند و $\hat{S}$ مسیرهایی در S است که در P متصل نشده‌اند. $\boldsymbol{L^*}$ یک مجموعه‌ای از قطعات مسیر در L می‌باشد و T=V-Q-P که مجموعه‌ای از رئوسی است که روی هیچ مسیری نمی‌باشد.

لم ۱: اگر بزرگترین جزء متصل $T \cup L^*$ اندازه‌ای دارای حداقل n/2 باشد پس بزرگترین جزء متصل $T \cup (S - \ddot{S})$ اندازه‌ای کوچکتر از n/2 را خواهد داشت.

اثبات: تا زمانی که Q یک جدا کننده برای G باشد بزرگترین جزء متصل T اندازه‌اش بزرگتر از n/2 می‌باشد (نمی‌توان یک مسیر از هر رأس در $\boldsymbol{L^*}$ به هر رأس در $S \cup \hat{S}$ داشت که آن (رئوس) از رئوس T استفاده کنند) زیر گراف استنتاج شده بر روی $T \cup L^* \cup (S - \ddot{S})$ باید شامل حداقل دو جزء متصل باشد هر یک از دو جزء شامل رئوسی از$\boldsymbol{L^*}$ یا شامل رئوسی از $(S - \ddot{S})$ را که در کل اندازه‌اش بزرگتر از n/2 می‌باشد را پیدا می‌کند.

توجه: تا زمانی که $\boldsymbol{L^*}$ و $(S - \ddot{S})$ در یک جزء متفاوت‌اند، بزرگترین جزء از $T \cup L^*$ اندازه‌اش حداقل n/2 خواهد بود پس اندازه‌ی اجزاء $(S - \ddot{S})$ که در $T \cup L^* \cup (S - \ddot{S})$ می‌باشد باید بزرگتر از n/2 باشد.

نتیجه: بزرگترین جزء متصل از $T \cup (S - \hat{S})$ دارای اندازه‌ای بزرگتر از n/2 را دارد.

اگر مورد ۱ (تا زمانی که P محدودیت‌های لیست شده را ارضاء کند می‌تواند تعداد مسیرها را در Q توسط یک کسر ثابت کاهش داد) رخ بدهد می‌توان مسیرها را در $(S - \ddot{S})$ به T اضافه نمود و تعداد کل مسیرها در Q را به واسطه‌ی یک کسر ثابت کاهش داد. اگر مورد ۲ رخ دهد می‌توان همانند مورد ۱ هر یک از دو مسیر $L - \hat{L}$ یا $(S - \ddot{S})$ را بدون نقض خاصیت جداسازی حذف کرد که نتیجه‌اش کاهش تعدادی از مسیرها در Q به واسطه‌ی یک کسر ثابت می‌باشد.

یافتن مجموعه‌ای از مسیرهای متمایز

اگر s و l معلوم باشند پس نیاز به یافتن یک مجموعه‌ای از مسیرها با رأس متمایز به صورت $P = \{p_1, p_2, \dots, p_\alpha\}$ می‌باشیم که α تا حد امکان بزرگ می‌باشد. زمانی که P را برای ادغام s و l استفاده می‌کنیم طول مسیرهایی که حذف شده‌اند تاحد امکان کوچک می‌باشند. کاهش مسأله به دو مسأله‌ی matching عمومی صورت می‌پذیرد.

لم ۲: مسأله‌ی یافتن بزرگترین مجموعه از مسیرهای متمایز که می‌تواند matching کامل با وزن مینیمم را در برخی از گراف‌های G' پیدا کند کاهش داده می‌شود (وزن هر لبه ۰ یا ۱ می‌باشد)

لم ۳: مسأله‌ی یافتن کوچکترین مجموعه از مسیرهای متمایز که می‌تواند اندازه‌ی مشخصی را برای مسأله‌ای که یک matching کامل با حداقل وزن را دارد پیدا کرده (آن هم در گرافی با بیش از 2n رأس و لبه‌هایی با وزن بیش از n) و کاهش دهد.

مجله‌ی علمی پژوهش در علوم رایانه ، شماره ۱۶، زمستان ۱۳۹۸، ص ۶۵- ۵۵
ISSN: 2745-3673
http://www.Rcsj.ir

نکته: می‌توان بزرگترین matching را با هزینه‌ی مینیمم در دو مرحله یافت. کاهش در لم ۳ برای یافتن اندازه‌یmatchingای بود که می‌خواستیم و کاهش در لم ۴ برای یافتن خود matching بود. زمان هر دو کاهش در هر دو لم O(log n) بوده و تعداد پردازنده‌ها معادل با n^2 می‌باشد.

ساختن قطعه اولیه:

جدا کننده‌ی کوچک Q یکبار ایجاد می‌گردد و باید برای ایجاد قطعه‌ی اولیه 'T با ریشه‌ی r استفاده شود به طوری که بزرگترین جزء متصل از 'V- T دارای اندازه‌ای بیش از n/2 را داشته باشد.

در ابتدا ریشه‌ی درخت 'T در r است. یک مسیر q∈Q را به طور مکرر انتخاب کرده و'T را گسترش می‌دهیم که 'T محتویاتش حداقل نصف q است (تا زمانی که تعداد مسیرها در Q به صورت یک مقدار ثابت و بیش از ۱۱ باشد ادامه می‌یابد) می‌توان عملیات را به صورت تکراری انجام داد و با معلوم بودن q کوچکترین رأس را در 'T می‌یابیم که از این رأس به q مسیری باشد.

اگر یک رأس مانند x را بیابیم و مسیری از x به یک رأس y∈q را رسم کنیم مانند $q = q'yq''$ می‌باشد. اگر q' حداقل به بزرگی q'' باشد و بتوان مسیر pyq' را به 'T افزود و q را با q'' جایگزین نمود، و در غیراینصورت می‌توان مسیر pyq'' را به 'T افزود و q را با q' جایگزین نمود. هر یک از دو مورد فوق را با کاهش طول q به حداقل نصف می توان انجام داد و تکرار این فرآیند در بیش از 11log n برابر صورت می‌گیرد.

برای نشان دادن بسط 'T به یک درخت DFS کافی است نشان دهیم که هیچ مسیری ما بین شاخه‌های مجزای 'T که کلیه‌ی رئوس داخلی‌اش در 'V- T است وجود ندارد.

آغاز بسط 'T از کوچکترین رأس صورت می‌گیرد و بزرگترین جزء متصل از 'V- T اندازه‌ای به بزرگی n/2 دارد چون 'T شامل کلیه‌ی مسیرهای موجود در Q می‌باشد.

شبه کد مورد نظر برای الگوریتم فوق به صورت زیر ارائه شده است.

```
procedure DFS(G, r)
T' ← Initial-Segment(G, T);
for each connected component C of G - T' do
Recursively compute a DFS tree for C;
Add this tree to T';
end { for }
end
procedure Initial-Segment(G, T )
Q ← v;
while |Q| > 11 do
Reduce(Q);
end { while )
Build the initial segment from Q;
end
procedure Reduce(Q)
K ← |Q| ;
Divide Q into two sets L and S, where |L|=K/4 and |S|=3K/4;
while |Q| > K/12 do
Find mincost disjoint paths
P = {p_1, p_2, ..., p_α} between L and S;
if α < K/12 > then
if Icc(T U (S - Ŝ)) < n/2 then
Q ← L U Ŝ U P ;
else
Q ← S U L̂ U P ;
return
else if lcc(T U L*) > n/2 then
```

ISSN: 2745-3673
http://www.Rcsj.ir

Q $\leftarrow$ L U $\hat{S}$ U P ;
return
else
Extend the paths of $\hat{L}$. Suppose p joins 1 and s, x and y are the endpoints of p
and $l = l'xl''$, x = s'ys". If $|s'| \geq |s''|$ then $l \leftarrow l'ps'$ and $s \leftarrow s''$;
otherwise, $l \leftarrow l'ps''$ and $s \leftarrow s'$. In both cases, l'' is discarded.
end { while }
end

زمان اجرای DFS برابر O(log n)– زمان اجرای Initial-Segment برابر با O(log n)– زمان اجرای reduce برابر با O(log n)– پس زمان مورد نیاز برای یافتن مسیرهای متمایز با حداقل‌ترین هزینه برابر با $O(\log^3 n)$ می‌باشد. چون مرحله‌ی آخر الگوریتم به دو مسأله‌ی matching تقسیم می‌گردد و بهینه‌ترین الگوریتم matching نیز دارای زمانی معادل با $O(\log^2 n)$ می‌باشد. پس در کل الگوریتم فوق دارای زمانی معادل با $O(\log^5 n)$ خواهد بود.

نتیجه‌گیری

در این مقاله نتایج بررسی هر سه الگوریتم برای یافتن الگوریتم موازی جستجوی اول عمق گراف در ادامه ارائه شده است. یافتن درخت DFS برای یک گراف در زمان کمتر از زمان خطی O(n) به وسیله الگوریتم Reif امکان پذیر نمی‌باشد. الگوریتم موازی که Smith برای DFS یک گراف ارائه کرده است دارای مرتبه‌ی زمانی $O(\log^3 n)$ می‌باشد و الگوریتمی که توسط Anderson & Aggarwal برای DFS یک گراف ارائه کردند دارای مرتبه زمانی $O(\log^5 n)$ می‌باشد.

ISSN: 2745-3673
http://www.Rcsj.ir

منابع و مراجع

[1] Reif, J, H. (1985) Depth-first search is inherently sequential. Information Processing Letters, 20:229-234.

[2] Reif, J, H. (1984) On synchronous parallel computations with independent probabilistic choice.SIAM Journal on Computing, 13(1):46-56.

[3] Smith, J, R. (1986) Parallel algorithms for depth-first searches I. planar graphs. SIAM Journal on Computing, 15(3):814-830, August 1986.

[4] Aggarwal, A. (1988) Anderson, R. J., A random NC algorithm for depth first search. Combinatorica, 8(1):1-12.

[5] Aggarwal, A. (1990) Anderson, R. J., Kao, M. Y., Parallel depth-first search in general directed graphs. SIAM Journal on Computing, 19(2):397-409.

[6] Anderson, R, J. (1986) A parallel algorithm for depth-first search. Extended Abstract, Mathematical Science Research Institute.

[7] Anderson, R, J. (1987) A parallel algorithm for the maximal path problem. Combinatorica, 7(4):315- 326.