# جواب تکلیف سری سوم

**طراحی الگوریتم**
**دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۳**

۱. به تعداد $n$ نفر در یک صف ایستاده‌اند. به هر نفر یک عدد متمایز داده شده است که فقط خودش از آن مطلع است. می‌خواهیم شخصی را پیدا کنیم که عددش از همسایه‌هایش بیشتر باشد. نشان دهید با پرسیدن $O(\log n)$ سوال می‌توانیم شخصی با این وضعیت را پیدا کنیم.

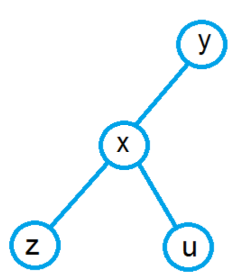
از نفر وسط صف $x$ و نفر سمت راستش $y$ و نفر سمت چپش $z$ می‌پرسیم. چهار حالت داریم:

- $z < x > y$
  در این حالت نفر وسط را به عنوان جواب گزارش می‌کنیم.
- $z < x < y$
  در این حالت الگوریتم را با افراد سمت راست $x$ ادامه می‌دهیم.
- $z > x > y$
  در این حالت الگوریتم را با افراد سمت چپ $x$ ادامه می‌دهیم.
- $z > x < y$
  در این حالت تفاوتی نمی‌کند. الگوریتم را می‌توانیم با هر دو طرف ادامه دهیم.

در هر صورت، با سه پرسش، یا شخص مورد نظر را پیدا می‌کنیم یا بازه مورد جستجو را نصف می‌کنیم. در نتیجه با $O(\log n)$ پرسش نفری با وضعیت مورد نظر را پیدا می‌کنیم.

۲. آیا می‌توانید نتیجه مسئله قبل را به حالتی که افراد راسهای یک درخت باینری هستند تعمیم دهید؟ اینجا دنبال فردی هستیم که عددش از همسایه‌هایش بیشتر باشد. اگر افراد رئوس یک درخت (غیر باینری) باشند چطور؟ دقت کنید اینجا فقط می‌خواهیم با کمترین تعداد پرسش فرد مورد نظر را پیدا کنیم (زمان اجرای الگوریتم ملاک نیست.)

برای درخت باینری می‌توانیم همان نتیجه صف را بدست آوریم. از یک ویژگی جالب درختان باینری استفاده می‌کنیم. در یک درخت باینری با $n$ راس همواره یک راس $x$ وجود دارد که زیردرختی که ریشه‌اش $x$ است اندازه‌اش (تعداد رئوسش) حداقل $n/3$ و حداکثر $\frac{2n+1}{3}$ است. فرض کنید $x$ چنین راسی باشد. اگر پدر این راس $y$ باشد و دو فرزند چپ و راستش هم (در صورت وجود) $z$ و $u$ باشند، از این ۴ راس عددشان را می‌پرسیم.



- اگر عدد $x$ از بقیه بزرگتر بود. راس $x$ را به عنوان جواب گزارش می‌کنیم.
- اگر عدد $x$ از عدد پدرش کوچکتر بود، زیردرخت با ریشه $x$ را از درخت حذف می‌کنیم و با بقیه ادامه می‌دهیم.
- در غیر اینصورت اگر $x < u$ آنگاه الگوریتم را با زیردرخت با ریشه $u$ ادامه می‌دهیم.
- در غیر اینصورت اگر $x < z$ آنگاه مشابه حالت قبل الگوریتم را با زیردرخت با ریشه $z$ ادامه می‌دهیم.

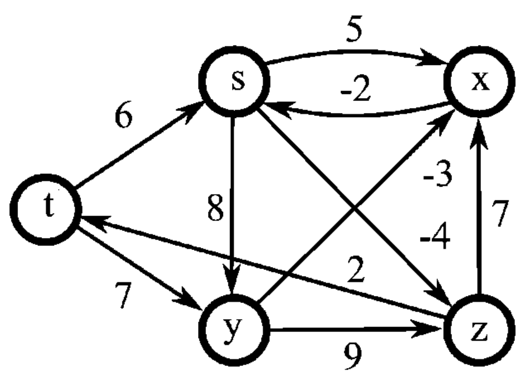
در هر حالت اندازه درختی که باقی می‌ماند حداکثر $\frac{2n+1}{3}$ است. در نتیجه الگوریتم بعد از $O(\log n)$ پرسش خاتمه می‌یابد.

برای درخت غیر باینری، برای مثال اگر درخت یک ستاره باشد (راسی در وسط با درجه $n-1$) حالتی وجود دارد که $n$ سوال باید پرسیده شود.

۳. در کلاس به این نکته اشاره شد در پیاده‌سازی الگوریتم بلمن فورد نیازی به نگه‌داری همه ستونها نداریم. در واقع کافی است که فقط یک آرایه را نگه داریم که به معنی فاصله کنونی تا راس مقصد است. در طی اجرای الگوریتم $d[u]$ همواره یک کران بالا برای فاصله راس $u$ تا راس مقصد $t$ است. در انتهای الگوریتم $d[u]$ دقیقا برابر فاصله $u$ از راس مقصد خواهد شد. شیوه بروزرسانی $d[u]$ ها در الگوریتم زیر نشان داده شده است. توجه کنید اینجا $w(u,v)$ طول یال $(u,v)$ را نشان می‌دهد که ممکن است منفی باشد. دقت کنید در یک حرکت زیرکانه، $d[v]$ را فقط در صورتی بروزرسانی می‌کنیم که یال خروجی $(v,u)$ موجود باشد بطوریکه $d[u]$ در گذر قبلی بروزرسانی شده باشد. این تعداد چک ها را کمتر می‌کند. اگر در یک گذر هیچ بروزرسانی انجام نشد، الگوریتم خاتمه می‌یابد.

آرایه first[] هم، مشابه آنچه در کلاس گفته شد، برای بازسازی کوتاهترین مسیر نگهداری می‌شود.

الگوریتم صفحه بعد را برای گراف داده شده اجرا کنید. برای مثال داده شده، چند بار آرایه $d$ بروزرسانی می‌شود؟ بعد از چند گذر الگوریتم خاتمه می‌یابد؟ جواب کوتاهترین مسیر را با استفاده از آرایه first بدست آورید.



BELLMAN–FORD–MOORE$(V, E, w, t)$

---

FOREACH node $v \in V$ : $d[v] \leftarrow \infty$

first$[v] \leftarrow null$

$d[t] \leftarrow 0$

FOR $i = 1$ TO $n - 1$

    FOREACH node $u \in V$ :

        IF ($d[u]$ was updated in previous pass)

            FOREACH edge $(v, u) \in E$ :

                IF $(d[v] > d[u] + w(v, u))$

                    $d[v] \leftarrow d[u] + w(v, u)$

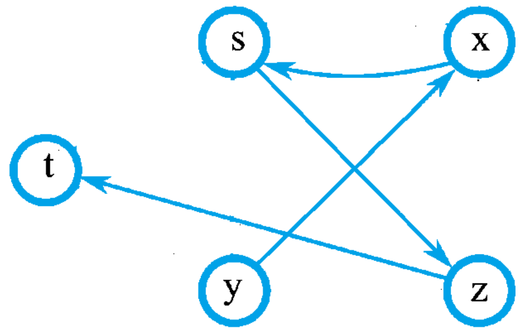
                    first$[v] \leftarrow u$

    IF (no $d[]$ value changed in pass $i$) STOP.

تعداد بروزرسانی‌های آرایه $d$ بستگی به ترتیب بررسی راسها در حلقه دوم دارد. اگر ترتیب الفبایی را فرض بگیریم و خط اول الگوریتم را در نظر نگیریم، تعداد بروزرسانی‌ها، 6 بار است. الگوریتم بعد از $n - 1 = 4$ گذر خاتمه می‌یابد.

| s | t | x | y | z |
|---|---|---|---|---|
| $\infty$ | $\underline{0}$ | $\infty$ | $\infty$ | $\infty$ |
| $\infty$ | 0 | $\infty$ | $\infty$ | $\underline{2}$ |
| $\underline{-2}$ | 0 | $\infty$ | $\underline{11}$ | 2 |
| -2 | 0 | $\underline{-4}$ | 11 | 2 |
| -2 | 0 | -4 | $\underline{-7}$ | 2 |

گراف first در زیر نشان داده شده است.




۴. الگوریتم فلوید_وارشال با استفاده از رابطه بازگشتی زیر طول کوتاهترین مسیر بین همه زوج رئوس را پیدا می‌کند.

$$\begin{aligned} ShortestPath(i,j,k) = \min\{&ShortestPath(i,j,k-1), \\ &ShortestPath(i,k,k-1) + ShortestPath(k,j,k-1)\} \end{aligned}$$

اینجا $ShortestPath(i,j,k)$ به معنی طول کوتاهترین مسیر از $i$ به $j$ است که فقط از مجموعه رئوس $\{1,\cdots,k\}$ استفاده می‌کند. در نهایت طول کوتاهترین از $i$ به $j$ برابر با درایه $ShortestPath(i,j,n)$ خواهد بود.

- توضیح دهید که چرا اگر گراف ورودی دور منفی داشته باشد، آنگاه برای حداقل یک $i$ داریم:

$$ShortestPath(i,i,n) < 0$$

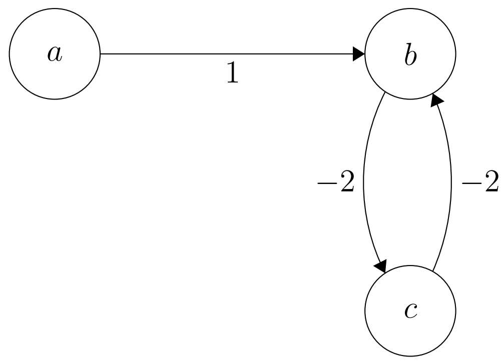
فرض کنید $P_{ij}$ یک مسیر ساده از $i$ به $j$ باشد. فرض کنید $k$ بزرگترین راس در مسیر $P_{ij}$ باشد. با استقرا روی $k$ می‌توان ثابت کرد که $ShortestPath(i,j,k)$ از طول مسیر $P_{ij}$ بیشتر نیست. همچنین بدیهی است، مقدار $ShortestPath(i,j,k)$ با افزایش $k$ بیشتر نمی‌شود.

حال فرض کنید راس $i$ در یک دور منفی باشد و $b$ بزرگترین راس در این دور باشد. برای مسیر $P_{ib}$ بنا به مشاهده بالا $ShortestPath(i,b,b-1)$ از طول مسیر $P_{ib}$ بیشتر نیست. همچنین $ShortestPath(b,i,b-1)$ از طول مسیر $P_{bi}$ بیشتر نیست. همچنین داریم:

$$ShortestPath(i,i,b) \leq ShortestPath(i,b,b-1) + ShortestPath(b,i,b-1) < 0$$

این گزاره گفته شده را ثابت می‌کند.

- الگوریتم فلوید وارشال را برای مثال زیر اجرا کنید. طول کوتاهترین مسیر بدست آمده برای زوج رئوس را بنویسید.




با فرض $a < b < c$، داریم

| $k = 0$ | a | b | c |
|---|---|---|---|
| a | 0 | 1 | $\infty$ |
| b | $\infty$ | 0 | -2 |
| c | $\infty$ | -2 | 0 |

| $k = a$ | a | b | c |
|---|---|---|---|
| a | 0 | 1 | $\infty$ |
| b | $\infty$ | 0 | -2 |
| c | $\infty$ | -2 | 0 |

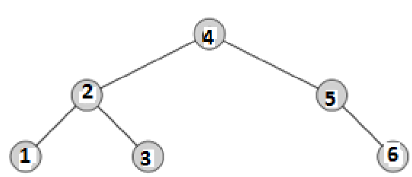
| $k = b$ | a | b | c |
|---|---|---|---|
| a | 0 | 1 | -1 |
| b | $\infty$ | 0 | -2 |
| c | $\infty$ | -2 | -4 |

| $k = c$ | a | b | c |
|---|---|---|---|
| a | 0 | -3 | -5 |
| b | $\infty$ | -4 | -6 |
| c | $\infty$ | -6 | -8 |

۵. با استفاده از تکنیک برنامه ریزی پویا، درخت BST بهینه برای یک دنباله دسترسی به طول $m$ را پیدا کنید. زمان اجرای الگوریتم شما چقدر است؟ فرض کنید که درخت شامل عناصر 1 تا $n$ است. برای مثال وقتی $n = 6$ یک دنباله دسترسی می‌تواند بصورت زیر باشد.

$$S = 2, 5, 5, 6, 1, 3, 3, 3, 5$$

زمان دسترسی به عنصر $i$ برابر با عمق $i$ در درخت است. برای مثال اگر درخت باینری بصورت زیر باشد، مجموع زمان دسترسی برای دنباله بالا برابر است با




$$accesstime(2) + 3accesstime(5) + accesstime(6) + accesstime(1) + 3accesstime(3) = 14$$

توجه کنید می‌خواهیم درختی بسازیم که مجموع زمان دسترسی با توجه به دنباله داده شده مینیمم شود.

فرض کنید

$$x_1 < x_2 < \cdots < x_n$$

عناصر متمایز داخل دنباله ورودی باشند که به ترتیب از کوچک به بزرگ مرتب شده‌اند. تعریف می‌کنیم $f(x_i)$ تعداد تکرار عنصر $x_i$ در دنباله ورودی. همچنین تعریف می‌کنیم $OPT(i,j)$ هزینه درخت بهینه برای زیرمجموعه عناصر

$$\{x_i, \cdots, x_j\}$$

تعریف می‌کنیم

$$F_{i,j} = \sum_{t=i}^{j} f(x_t).$$

اگر $x_k$ را در ریشه درخت قرار دهیم داریم:

$$OPT(1,n) = \min_{k \in \{1,\cdots,n\}} \{OPT(1,k-1) + OPT(k+1,n) + F_{1,n}\}$$

وقتی

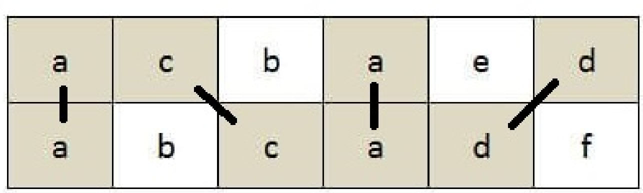
$$OPT(i,j) = 0 \quad \text{برای} \quad i > j$$

بطور کلی

$$OPT(i,j) = \min_{k \in \{i,\cdots,j\}} \{OPT(i,k-1) + OPT(k+1,n) + F_{i,j}\}$$

با توجه به اینکه جدول $OPT$ دو بعدی است و هر خانه از جدول را می‌توان در زمان $O(n)$ محاسبه کرد، پس زمان پر کردن جدول $O(n^3)$ است. اگر طول دنباله $m$ باشد، مرتب سازی دنباله و محاسبه $f(x)$ ها در زمان $O(m \log n)$ قابل انجام است. مقادیر $F_{i,j}$ هم در زمان $O(n^2)$ قابل محاسبه است. پس در کل زمان اجرای الگوریتم $O(m \log n + n^3)$ است.

۶. توضیح دهید که چگونه می‌توان طولانی ترین زیر دنباله مشترک میان دو دنباله $S$ و $T$ را با استفاده از راه حلی که برای مسئله همترازسازی دنباله‌ها در کلاس ارائه کردیم محاسبه کنیم؟ دقت کنید یک زیردنباله لزوما دنباله‌ای پشت سر هم از عناصر نیست.




کافی است هزینه همتراز کردن دو کاراکتر متفاوت را بینهایت قرار دهیم (عددی خیلی بزرگ) و همچنین هزینه همتراز کردن دو کاراکتر یکسان را صفر قرار دهیم. همچنین هزینه همتراز نکردن را هم 1 قرار می‌دهیم. جواب مسئله طولانی ترین زیردنباله مشترک خواهد بود.

۷. در کلاس دیدیم که با استفاده از آرایه دوبعدی $OPT(i,j)$ می‌توانیم هزینه همترازسازی بهینه بین دو دنباله $S$ و $T$ را پیدا کنیم. نشان دهید که چگونه می‌توان از مسئله کوتاهترین مسیر در گراف استفاده کرد و همترازسازی بهینه را از آرایه دو بعدی $OPT$ استخراج کرد.

به شکل زیر توجه کنید. تصویر خود گویای مطلب است. برای توضیحات بیشتر، صفحه ۲۸۳ کتاب مرجع را ببینید.