تکلیف سری سوم

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

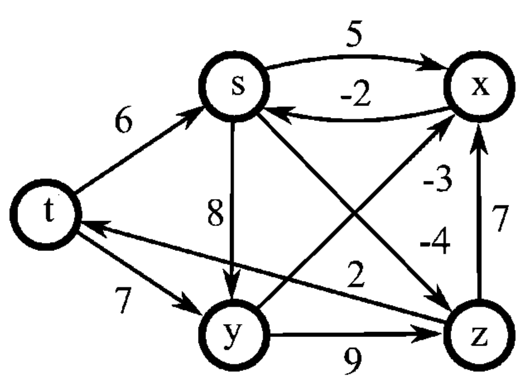
۱. به تعداد n نفر در یک صف ایستاده‌اند. به هر نفر یک عدد متمایز داده شده است که فقط خودش از آن مطلع است. می‌خواهیم شخصی را پیدا کنیم که عددش از همسایه‌هایش بیشتر باشد. نشان دهید با پرسیدن $O(\log n)$ سوال می‌توانیم شخصی با این وضعیت را پیدا کنیم.

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.)

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 ∈ V : d[v] ← ∞

first[v] ← null

d[t] ← 0

FOR i = 1 TO n − 1

    FOREACH node u ∈ V :

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

            FOREACH edge (v, u) ∈ E :

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

                    d[v] ← d[u] + w(v, u)

                    first[v] ← u

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

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

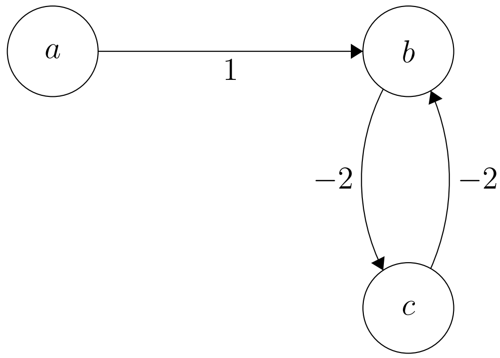
$$\begin{aligned} ShortestPath(i,j,k) = \min\{&ShortestPath(i,j,k-1),\\ &ShortestPath(i,k,k-1) + ShortestPath(k,j,k-1)\} \end{aligned}$$

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:

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

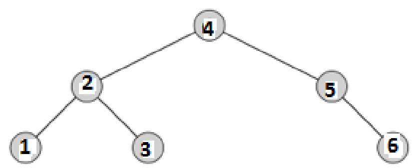
- الگوریتم فلوید وارشال را برای مثال زیر اجرا کنید. طول کوتاهترین مسیر که از الگوریتم برای زوج رئوس بدست آمده را بنویسید.



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.

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

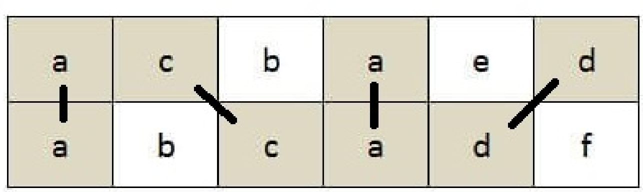
زمان دسترسی به عنصر i برابر با عمق i در درخت است. برای مثال اگر درخت باینری بصورت زیر باشد، مجموع زمان دسترسی برای دنباله بالا برابر است با



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

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

۶. توضیح دهید که چگونه می‌توان طولانی 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.



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.