موضوع:

الگوریتم بلمن فورد

نام و نام خانوادگی:

مسعود کوکب

درس:

برنامه ریزی پویا

نام استاد:

دکتر محمد فرشی

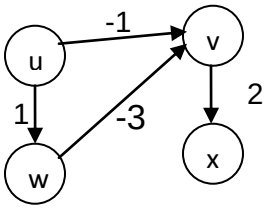
الگوریتم Bellman-Ford مساله ی کوتاهترین مسیرهای تک مبدا را در گراف های وزن دار جهت دار حل میکند.در حالت کلی وزن یالها میتواند منفی هم باشد.

نکته:در گراف نباید دوری با وزن منفی وجود داشته باشد .چون دور با وزن منفی باعث می شود دیگر کوتاهترین مسیر بین دو راس بدون معنی باشد .زیرا میتوان با هر بار دور زدن در این دور وزن مسیر بین دو یال مورد نظر را کاهش داد.با تکرار این عمل تا بی نهایت میتوان به وزن ∞- رسید که این منطقی نیست.

ورودی این الگوریتم یک گراف وزن دار جهت دار مانند G=(V,E) با یک راس مبدا مانند v و یک آرایه ی دوبعدی است که وزن یال های موجود بین دوراس را نشان میدهد،درصورتی که یالی بین دو راس نباشد مقدار ∞ را بر میگرداند.این آرایه را با اسم w می شناسیم.

یک راه حل برای یافتن کوتاه ترین مسیر بین دو راس در یک گراف به این صورت ا ست که برای رفتن به هر راس از راسی که قرار داریم یالی با کمترین وزن موجود را برداریم .که این روش همان روش حریصانه است و با یک مثال نقض ثابت میکنیم که به جواب درست نمیرسد.

مثال نقض:



در گراف روبرو کوتاه ترین مسیر بین دو راس u و x را

طبق الگوریتم حریصانه بدست می آوریم.چون ابتدا در دراس u هستیم ،

پس باید ابتدا یال با وزن -1 را انتخاب کنیم و به راس v برویم.سپس در راس v باید مستقیما به x برویم.که وزن یال موجو 2 است.پس وزن مسیر برابر 1 میشود.اما اگر ابتدا از u به w برویم سپس از w به v و سپس به x برویم مسیری با وزن 0 بدست می آید.پس راه حل حریصانه به جواب صحیح نمیرسد.

بررسی زیر ساختار بهینه:

فرض کنید در یک گراف کوتاه ترین مسیر بین u و v موجود باشد.فرض کنید این مسیر شامل راسی مانند x باشد.ادعا میکنیم مسیر u به x دارای وزن مینیمم است .با تکنیک cut-and-paste به سادگی قابل اثبات است.فرض میکنیم مسیر بین u و x را در جواب بهینه با p نمایش می دهیم،فرضکنید مسیری مانند q وجود دارد که وزن آن نسبت به p کمتر باشد.پس مسیر p را در جواب بهینه بو میداریم و مسیر q را جایگزین آن میکنیم.پس وزن مسیر بین u و v کمتر از مقدار بهینه میشود که این با بهینه بودن مسیر اولیه در تناقض است.

ایجاد ی رابطه ی بازگشتی:

وقتی هیچ دوری با وزن منفی نداریم،بین هر دوراس از یک گراف n-راسی تنها یک مسیر کوتاهتر وجود دارد که حداکثر n-1 یال دارد.برای مشاهده ی این مطلب، توجه کنید که مسیری که بیشتر از n-1 یال دارد حداقل یک راس آن تکراری است یعنی دور وجود دارد.حذف دورهای مسیر،مسیر دیگوی با همان منبع و مقصد را نتیجه می دهد .این مسیر بدون دور است و طول آن از مسیر اولیه بیشتر نیست،چون حداقل وزن دورهای حذف شده صفر است .ما میتوانیم این مشاهدات را برای حداکثر (ماکزیمم) تعداد یال های کوتاهتذین مسیر از راس منبع به تمام رئوس باقی مانده گراف استفاده کنیم.

فرض کنید dist[l][u] طول کوتاهترین مسیر از راس منبع v به راس u باشد.همانطور که قبلا تذکر داده شد، وقتی دوری با وزن منفی نداریم ، میتوانیم جست و جو برای یافتن کوتاهترین مسیرها را به مسیرهایی با حداکثر n-1 یال محدود کنیم . بنابراین dist[n-1][u] طول کوتاهترین مسیر از v به u است.

پس هدف ما در اینجا محاسبه ی dist[n-1][u] برای همه ی مقادیر u است.این کار با استفاده از روش برنامه ریزی پویا می تواند انجام شود.

1.اگر کوتاهترین مسیر از u به v با حداکثر k یال (k>1) ، بیشتر از k-1 یال نداشته باشد آنگاه dist[k][u]= dist[k-1][u] است.

2.اگر کوتاهترین مسیر از u به v با حداکثر k یال،(k>1)،دقیقا k یال داشته باشد،این مسیر از یک کوتاهترین مسیر v به یک راس j و به دنبال آن یال <j,u> ساخته شده است.مسیر u به j،k-1 یال دارد و طول آن dist[k-1][j] است.همه رئوس i به طوری که یال <i,u> در گراف موجود باشد،انتخاب ها وجایگزین هایی برای j هستند.چون ما کوتاهترین مسیر را میخواهیم،مقدار i که dist[k-1][i] را حداقل(مینیمم) می کند، مقدار درست برای j است.

از این مشاهدات رابطه بازگشتی برای dist حاصل میشود

$$dist[k][u]=\min\{ dist[k-1][u], \min_i\{ dist[k-1][i]+w[i][u]\}\}$$

این رابطه بازگشتی میتواند برای محاسبه ی dist[k] از dist[k-1] به ازای k=2,3,…,n-1 استفاده شود.

ارائه ی یک الگوریتم و ساخت یک جواب بهینه:

برای ساخت یک جواب بهینه از یک آرایه کمکی به نام A استفاده میکنیم.در این آرایه مقدار هر درایه ،عنصر واسطه ی قبلی برای رسیدن به مبدا را نشان میدهد.

```
Belmanford(v,w,dist,n)
{
    for i=1 to n do
        dist[i]=w[v][i];
    for k=2 to n-1 do
        for each u such that u!=v and u has at least one incoming edge
do
            for each <i,u> in the graph do
                if dist[u]>dist[i]+w[i][u] then
                    dist[u]=dist[i]+w[i][u];
                    A[i][u]=i;
}
```

تابعی برای ساخت یک جواب بهینه:

نام این تابع را Print می نامیم.

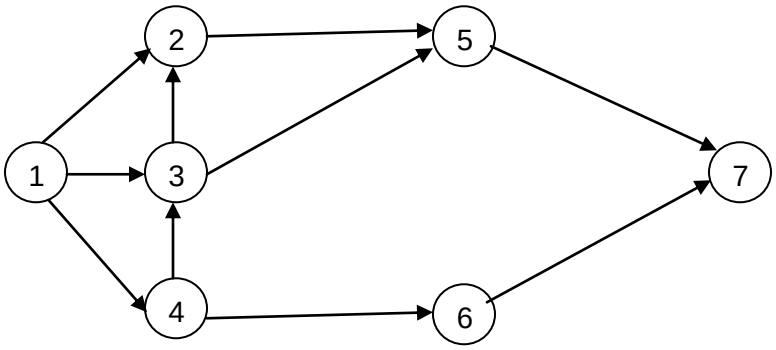
```
    Print(A,x)
{
    if  A[x]==v
        print 'v';
        return;
```

```
        else
                Print(A,A[x]);
                print'x';
}
```

پیچیدگی زمانی الگوریتم :

هر تکرار حلقه ی for با استفاده از ماتریس همجواری ($o(n^2)$ زمان میگیرد.پیچیدگی کل وقتی ماتریس های همجواری استفاده شود برابر ($o(n^3)$ می شود.پیچیدگی الگوریتم کوتاه ترین مسیر را میتوان با توجه به این نکته که اگر هیچکدام از مقادیر dist در یک بار تکرار حلقه for تغییر نکند در تکرار بعدی هم تغییر نمیکنند،میتوان کاهش داد.این حلقه میتواند دوباره بازنویسی شود ،تا بعد از n-1 تکرار یا بعد از اولین تکرار که در آن هیچ یک از مقادیر dist تغییر نکنند،خاتمه یابد.

مثال:



در شکل فوق گرافی با هفت راس موجود است.هدف محاسبه ی آرایه ی dist است.این آرایه را با استفاده از رابطه ی بازگشتی و الگوریتم پر میکنیم.dist[k][1] برابر صفر است.زایرا 1 راس منبع است.همچنین dist[1][2]=6 ،dist[2][3]=5 و dist[1][4]=5 زیرا یالهایی از یک به این گره ها وجود دارد.فاصله dist[1][ ] برای گره های 5،6 و 7 برابر ∞ است،زیرا هیچ یالی از این رئوس به 1 وجود ندارد.

$$dist[2][2]=\min\{ dist[1][2], \min_i\{ dist[1][i]+w[i][2]\}\}$$

=min{6,0+6,5-2,5+∞,∞+∞,∞+∞,∞+∞}=3

دراینجا جمله های 0+2،5-6،5+∞،∞+∞،∞+∞،∞+∞ به ترتیب متناظر با انتخاب i برابر1،3،4،5،6 و 7 است.بقیه ی عناصر به صورت مشابه محاسبه میشوند.

$$Dist=\begin{pmatrix} 0 & 6 & 5 & 5 & \infty & \infty & \infty \\ 0 & 3 & 3 & 5 & 5 & 4 & \infty \\ 0 & 1 & 3 & 5 & 2 & 4 & 7 \\ 0 & 1 & 3 & 5 & 0 & 4 & 5 \\ 0 & 1 & 3 & 5 & 0 & 4 & 3 \\ 0 & 1 & 3 & 5 & 0 & 4 & 3 \end{pmatrix}$$