اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

8-6 مسالهکوتاه‌ترین مسیر از یک رأس

8-6-1**مقدمه:**

در یک گراف وزن‌دار کوتاه‌ترین مسیر از رأس s به رأس v یک مسیر جهت‌دار از s به v است به صورتی که هیچ مسیر دیگری با وزن کمتر بین این دو رأس وجود نداشته باشد. در میان کاربردهای مستقیم یافتن کوتاه‌ترین مسیر از قبیل مسیریابی در نقشه‌ها و یا شبکه‌های کامپیوتری کابردهایی نیز وجود دارد که به نظر می‌رسد که نمی‌توان ارتباطی بین آنها و تئوری گراف‌ها برقرار کرد. از میان آنها می‌توان به تقسیم بندی وظایف در یک پروژه و یا موضوع آربیتراژ در علم اقتصاد اشاره کرد.

در مسائل کوتاه‌ترین مسیر ۴ حالت برای بررسی داریم:

1) کوتاه‌ترین مسیر را از رأس مبدا به سایر رأسها می‌خواهیم.

single source shortest path

2) کوتاه‌ترین مسیر را از همه رأس ها به رأس مقصد می‌خواهیم.

single target shortest path

3) کوتاه‌ترین مسیر را بین دو رأس مبدا و مقصد می‌خواهیم.

single pair shortest path

4) کوتاه‌ترین مسیر را بین هر دو رأس در گراف می‌خواهیم.

all pairs shortest path

حالت های اول و دومدرواقع یک مساله هستند.برای تبدیل حالت دوم به حالت اول کافی است کوتاه‌ترین‌مسیر از رأس مقصد را به سایر رأس ها به دست بیاوریم و در نهایت جهت مسیر انتخاب شده را برعکس کنیم. در گراف جهت دار باید قبل از هر کاری ابتدا جهت همه یال های گراف را برعکس کرده و بعد الگوریتم کوتاه‌ترین مسیر از رأس مبدأ را استفاده کنیم. همچنین در نهایت، با برعکس کردن یال های مسیر یافت شده، یال های دیگر گراف را نیز برعکس کنیم.

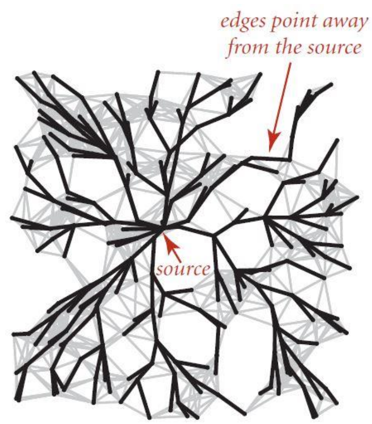
جواب مسأله در حالت اول یک درخت به نام درخت *کوتاه‌ترین مسیرها*[1] خواهد بود. به طور کلی در یک گراف وزن‌دار با انتخاب یک راس مانند s، درخت کوتاه‌ترین مسیرها از s، یک زیرگراف شامل s و تمام رئوس قابل

---

[1] Shortest-paths tree (SPT)

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

دسترس از s است که یک درخت ریشه‌دار را تشکیل می‌دهد و هر مسیر موجود در آن که از ریشه شروع می‌شود، کوتاه‌ترین مسیر از ریشه در گراف است. این درخت همواره وجود دارد. شکل زیر درخت کوتاه‌ترین مسیرها را در یک گراف با ۲۵۰ رأس مشخص کرده است.



الگوریتم های مساله کوتاه‌ترین مسیر:

الگوریتم دیکسترا: کوتاه‌ترین مسیر بین دو رأس (حالت ۳)

الگوریتم بلمن-فورد: کوتاه‌ترین مسیر از رأس مبدأ در حالتی که یال‌ها می‌توانند وزن منفی هم داشته باشند.

الگوریتم جستجوی *A : با کمک روش‌های ابتکاریِ جستجو، مسألهٔیافتنکوتاه‌ترینمسیربیندورأساتسریعمی-بخشد.

الگوریتم فلوید-وارشال:کوتاه‌ترین مسیر بین هر دو رأس (حالت ۴)

الگوریتم جانسون:کوتاه‌ترین مسیر بین هر دو رأس (حالت ۴)

در این قسمت به بررسی حالت سوم یعنی کوتاه‌ترین مسیر بین دو رأس می‌پردازیم.

در این مسائل حالت کلی گراف جهت داروزن دار به ما داده می‌شود؛می‌دانیم که در حالت خاص گراف های بدون وزن می‌توان از الگوریتم BFS برای یافتن کوتاه‌ترین مسیر استفاده کرد.

گراف G=(E,V) را در نظر می‌گیریم که در آن وزن یال‌ها به عدد حقیقی R نگاشت می‌شود.

$$W: E \to R$$

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

وزن مسیرP که به صورت دنباله ای از رأس‌ها به صورت $P = <V_0, V_1, …, V_k>$ است، برابراستبا مجموع وزن یال‌های سازنده مسیر.

لم 1 :

فرض کنید یک گراف جهت دار وزن دار G=(V,E) داریم که وزن آن دارای این تابع است:

$$W: E \rightarrow R$$

فرض کنید مسیر Pکوتاه‌ترین مسیر بین دو رأس دلخواه باشد. رأسهای این مسیر را V_1 تا V_k می‌نامیم:

$$P = < V_1, V_2, \dots, V_k >$$

آنگاه برای هر i و j به صورت :

$$1 \leq i \leq j \leq k$$

اگر P_{ij}زیرمسیری از P_{1K}و بین دو رأسi و j باشد:

$$P_{ij} = < V_i, V_{i+1}, \dots, V_j >$$

P_{ij}کوتاه‌ترین‌مسیر بین دو رأسi و j خواهد بود.

اثبات لم1: مسیر Pرا به صورت جمع سه زیرمسیر در نظر می‌گیریم که وزن آن به صورت زیر محاسبه می‌شود:

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk}).$$

حالا فرض می کنیم که یک مسیر مانند P'_{ij} بین دو رأسV_i و V_j وجود دارد که وزن آن کمتر از وزن مسیر P_{ij}است.

$$w(p'_{ij}) < w(p_{ij}).$$

بنابراین وزن مسیر P با احتساب زیرمسیر P'_{ij} به جای P_{ij} به این صورت محاسبه می شود:

$$w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$$

که کمتر از وزن مسیر Pاست و این با فرض اولیه در تناقض است.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

8-6-2**الگوریتم دایکسترا:**

الگوریتم دایکسترا کوتاه‌ترین مسیر از یک رأس مشخص به سایر رأس‌ها یا رأس مقصد را به ما می‌دهد. و با رسیدن به رأس مقصد الگوریتم متوقف می‌شود. در صورتی که گراف دارای یال با وزن منفی باشد، باید از الگوریتم‌های دیگری مانند الگوریتم بلمن–فورد استفاده کرد.

در این روش برای هر رأس یک زیروند یا اندیس در نظر گرفته می‌شود. این اندیس در واقع فاصله رأس مبدأ تا آن رأس را در هر مرحله اجرای الگوریتم مشخص می‌کند.

نحوه اجرای الگوریتم:

1) انتخاب رأس مبدا
2) یک مجموعهS از رأس‌های گراف در نظر می‌گیریم که در ابتدای اجرای الگوریتم تهی است.با پیشرفت الگوریتم این مجموعه شامل رأس‌هایی می‌شود که کوتاه‌ترین مسیر به آن‌ها تا آن مرحله یافت شده است.
3) رأس مبدأ را با اندیس صفر و بقیه رأس ها را با اندیس ∞ مشخص می کنیم و رأس مبدأ را رأس *منتخب* می‌نامیم.
4) شروع به حساب کردن اندیسی جدید برایرأس‌های مجاور رأس *منتخب*که خارج مجموعه S هستند، می‌کنیم؛ به این ترتیببه همه رأس های مجاور رأس *منتخب*رفته و اندیس جدید را برای هرکدام از این رأس ها مانند u به این صورت محاسبه می‌کنیم : وزن یال بین رأس *منتخب* و u+ اندیس رأس*منتخب*. از بین این مقدار جدید و اندیس قدیمی، عدد کوچک‌تر را بر می‌گزینیم و مقدار اندیس را به‌روز می-کنیم. همچنین رأس *منتخب* را وارد مجموعه S می‌کنیم.
5) از بین رأس های خارج از مجموعه Sرأس با کم‌ترین اندیس را رأس *منتخب* می‌نامیم.
6) اگر رأس مقصد را به عنوان رأس *منتخب* دیدیم، الگوریتم را خاتمه می‌دهیم وگرنه الگوریتم را از مرحله 4 ادامه می‌دهیم.

در پایان اندیس رأس مقصد نشان دهنده ی وزن کوتاه‌ترین مسیر از مبدا به مقصد است.

پیچیدگی زمانی:

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

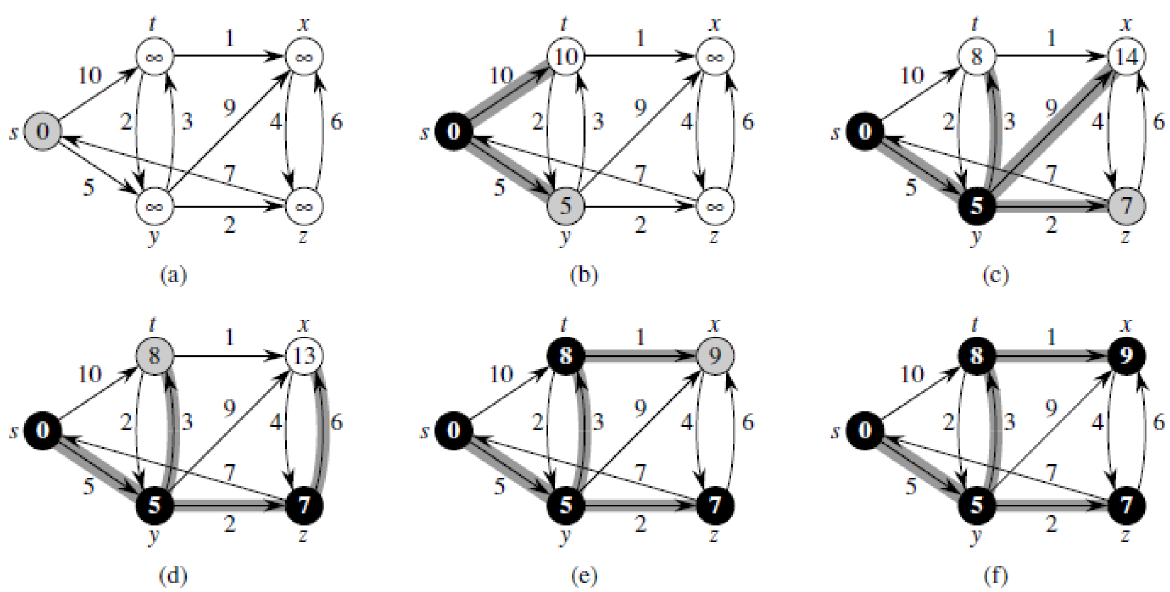
در صورت پیاده‌سازی با آرایه یا لیست پیوندی، پیچیدگی زمانی برابر $O(|V|^2 + |E|)$ خواهد بود. برای گراف‌های پراکنده، از لیست مجاورت برای نگهداری گراف استفاده می‌کنیم که پیچیدگی زمانی آن برابر است با: $O((|V| + |E|)\log|V|)$.

در شبه کد زیر می‌توانیم نحوه ی اجرای الگوریتم دایکسترا را مشاهده کنیم:

```
DIJKSTRA(G,w, s)
        INITIALIZE-SINGLE-SOURCE(G, s)
        S ← Ø
        Q ← V[G]
        while Q  != Ø
                do  u ← EXTRACT-MIN(Q)
                S ← S ∪ {u}
                for each vertex v ∈Adj[u]
                        do RELAX(u, v,w)
```

برای اینکه راحت تر الگوریتم دایکسترا را درک کنیم از مثال کتاب مقدمه‌ای بر الگوریتم‌ها تألیف Cormen، Leiserson،Rivest و Stein استفاده می‌کنیم :

مثال: گرافی با 5 رأس به شکل زیر داریم ، می‌خواهیم کوتاه‌ترین مسیر از رأسs به سایر رأس‌ها را به کمک الگوریتم دایکسترابه دست آوریم.



شکل1

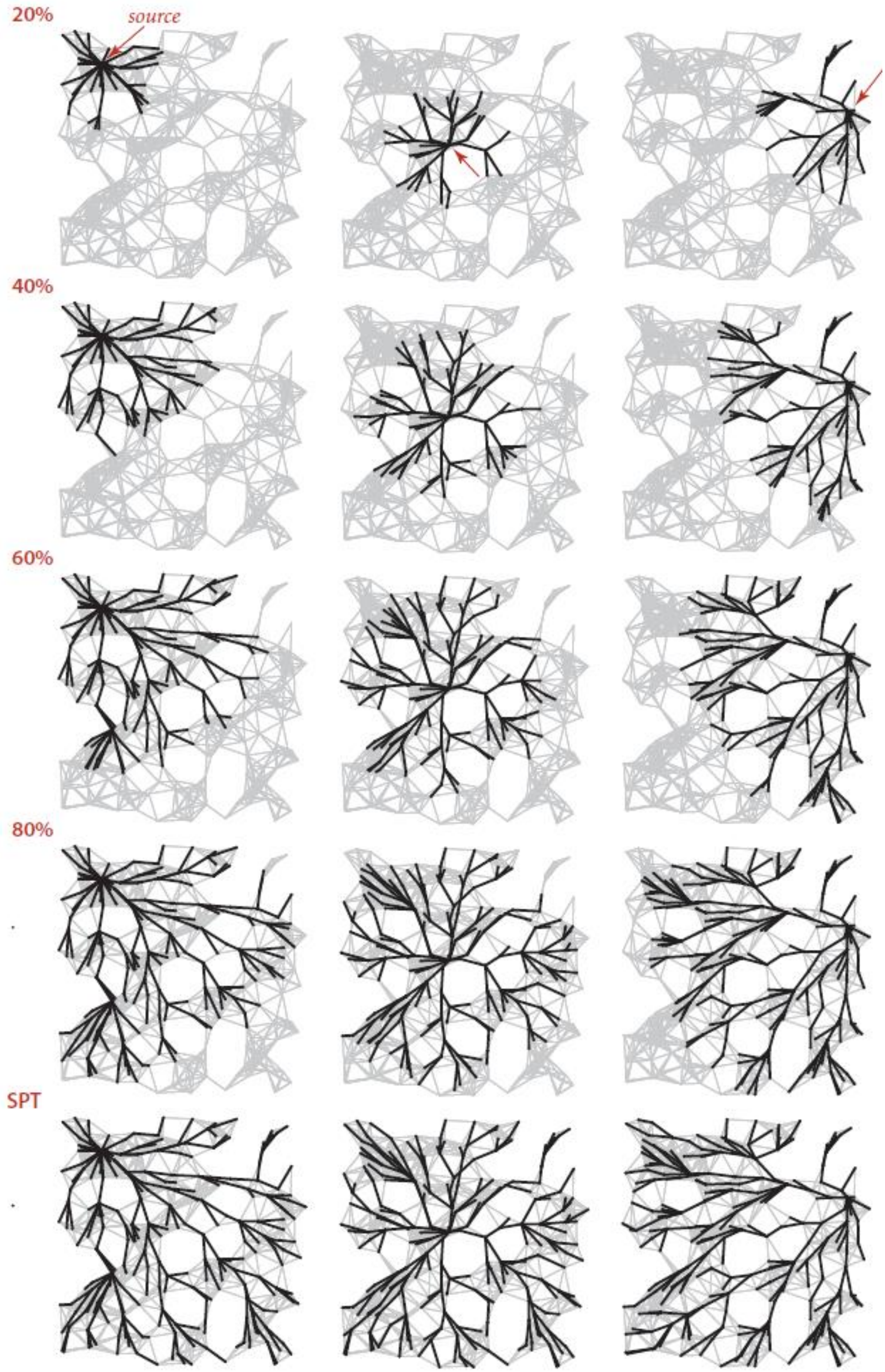
اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

دقت کنید که هر رأسی که به رنگ مشکی در می‌آید به منزله این است که وارد مجموعه S شده است، رأسی که خاکستری رنگ است، در حال بررسی و به روز شدن هستند و سایر رأس‌ها در صف کم اولویت قرار دارند.(مجموعه Q = V – S)

قسمتa : گراف ابتدایی است که عدد داخل هر رأس را ∞ قراردادیم تا در مراحل بعدی به تدریج کوتاه‌ترین مسیر از s تا آن رأس‌ها را بیابیم.

قسمت های b-f: نحوه ی اجرای الگوریتم را نشان می‌دهد، از رأسی که شروع کرده‌ایم به سایر رأسها می‌رویم، به این ترتیب عدد درون رأس ها به‌روز می‌شود . سپس از بین رأس هایی که در مجموعه Q قرار دارند آن رأسی که کمترین عدد را داراست انتخاب می‌کنیم و روی آن رأسمراحل 4تا 6 را که در قسمت نحوه ی اجرای الگوریتم گفته شد، تکرار می‌کنیم.

شکل زیر اجرای الگوریتم دایکسترا را در یک گراف با ۲۵۰ راس با شروع از یه مبدا مختلف بررسی می‌کند:


20%
source
40%
60%
80%
SPT

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

8-6-3 **الگوریتم بلمن‌فورد:**

دیدیم که الگوریتم دایکسترا مسأله کوتاه‌ترین مسیر را در گراف‌هایی که یال منفی دارند، حل نمی کند . اما این الگوریتم مسأله کوتاه‌ترین مسیر را برای گراف‌هایی که دارای یال منفی هستند نیز حل می کند.

توجه شود که پیچیدگی زمانی الگوریتم دایکسترا کمتر از بلمن‌فورد است، بنابراین در مواقعی که گراف دارای یال منفی نیست، کاربرد بیشتری دارد.

لازم به ذکر است که اگر گراف، دوری با وزن منفی داشته باشد که از مبدا قابل دست‌یابی باشد ، مسأله کوتاه‌ترین مسیر جوابی ندارد چون پیمایش مداوم آن دور،همواره وزنی کم‌تر ایجاد می‌کند و مقدار کوتاه‌ترین فاصله برای بعضی از رأس‌ها وجود نخواهد داشت. الگوریتم بلمن‌فورد راه حلی هم برای فهمیدن وجود دور منفی در گراف ارائه می‌دهد.

ساختار اصلی این الگوریتم مشابه الگوریتم دایکسترا است.

نحوه اجرای الگوریتم:

الگوریتم به صورت $|v| - 1$ بار به‌روز کردن آرایه‌ای به نام d خواهد بود. آرایه dبه این شکل تعریف می شود که برای هر رأسv مقدار d_v در آخرین مرحله i ام برابر کوتاه‌ترین مسیر از مبدا به v است(البته با این شرط که تعداد یال‌های این مسیرحداکثرi باشد). بنابراین در پایان مرحله $|v| - 1$ام d_vبرابر با کوتاه‌ترین مسیر از مبدا به v است.

در واقع مقدار همه درایه های d در هر بار اجرای الگوریتم به‌روز می‌شوند. این مقدارها، برای تعدادی از رأس‌ها واقعی و برای تعدادی دیگر غیر واقعی بوده ولی همواره در حال نزدیک شدن به مقدار واقعی خود است. در اولین به‌روز کردن آرایه، مقدار d برای رأس‌هایی که کوتاه‌ترین مسیر از رأس مبدأ تا آن‌ها شامل یک یال است، واقعی خواهند شد. در دومین به‌روز کردن آرایه، رأس هایی که با کوتاه‌ترین مسیر از مبدأ تا آن‌ها شامل دو یال است، مقدار واقعی در آرایه d خواهند یافت و به همین ترتیب در $|v| - 1$ امین به‌روز کردن آرایه، مقدار d برای اندیس‌هایی که کوتاه‌ترین فاصله آن‌ها از مبدأ شامل $|v| - 1$یال است، مقدار واقعی خواهد داشت. از آنجایی که کوتاه‌ترین فاصله از رأس مبدأ تا همه رئوس حداکثر از $|v| - 1$ رأس دیگر خواهد گذشت،همه مقادیر dمقدار واقعی خواهند بود.

اساس کار این الگوریتم،ریلکس کردن یال‌ها در هر مرحله است.

(تعریف ریلکس کردن: اگرd_v> d_u+w(u,v) بود، آنگاهd_v = d_u+w(u,v).)

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

حالا اگرریلکس کردن را برای بار |v|ام هم انجام دادیم و باز دیدیم که آرایهd تغییر کرد نتیجه می‌گیریم که گراف دارای دور منفی است. این یک مزیت برای الگوریتم بلمن‌فورد است که اگر گراف دارای دور منفی باشد ، آن را تشخیص می دهد.

پیچیدگی زمانی:

دیدیم که|v|-1 مرحله در اجرای الگوریتم داشتیم که در هر مرحله E عملیات بر روی یال‌ها انجام می‌شود ، به این ترتیب پیچیدگی زمانی الگوریتم بلمن‌فورد(|v||E|)o است.

شبه کد زیر نشان دهنده ی نحوه ی پیاده سازی این الگوریتم است که اگر گراف دارای دوری با وزن منفی باشد false بر می‌گرداند در غیر این صورت true برمی‌گرداند و دنباله dرا که معرف کوتاه‌ترین مسیر است مقداردهی می‌کند:

```
Begin
  for all vertexes w do
          d_w= ∞
          d_s= 0
   For  i=1 to |v|-1 do
      For all edge (u,v) in E do
                     If  d_u+w(u,v) <d_v
d_v = d_u+w(u,v)
for all edge (u,v) in E do
             If  d_u+w(u,v) <d_v
                   Return false
Return true
End
```

از خصوصیات بلمن‌فورد می‌توان به موارد زیر اشاره کرد:

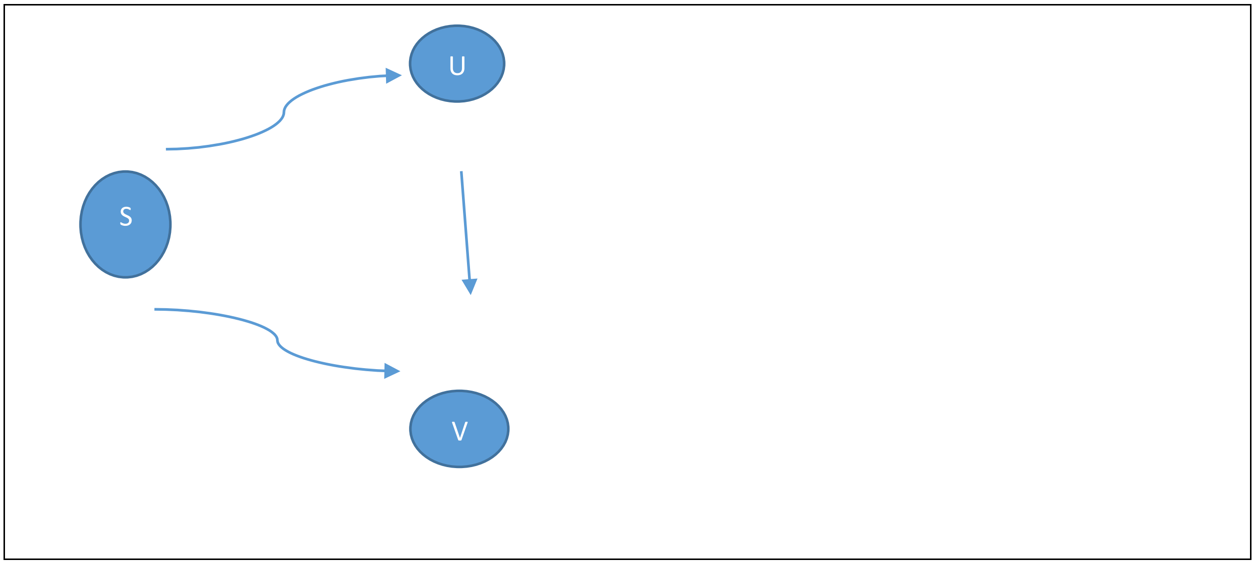
$\delta(s,v)$: طول کوتاه‌ترین مسیر بین s,v

1-خاصیت نامساوی مثلثی : $\delta(s,u)+w(u,v)\geq\delta(s,v)$

2-همواره $\delta(s,u)$به ازای هر u آنگاه $d[u] \geq \delta(s,u)$

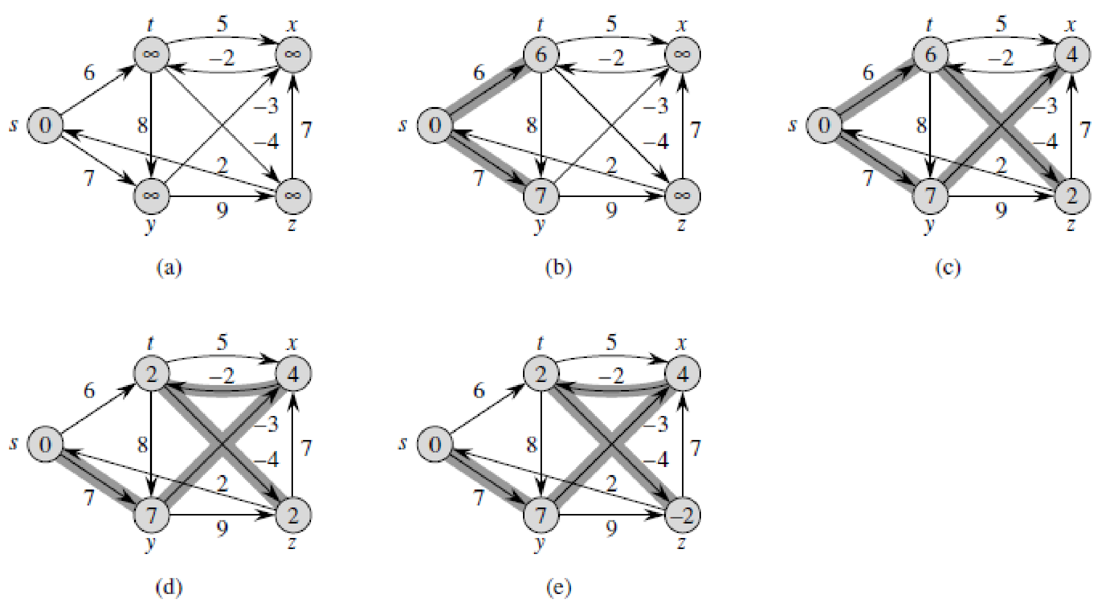
3-اگر از s بهu مسیری وجود نداشته باشد آنگاه$\delta(s,u)$ $d[u] = \infty$ =

4- فرض کنید از S به u مسیری وجود داشته باشد و از u به v یالی باشد و این مسیر از S بهvکوتاه‌ترین مسیر باشد. و در لحظه $d[u]= \delta(s,u)$ باشد آنگاه در لحظه بعدی حتما $d[v] = \delta(s,v)$ است.



برای روشن تر شدن این الگوریتم از مثال زیر که از کتاب مقدمه‌ای بر الگوریتم‌ها گرفته شده، بهره می‌بریم:

مثال: گرافی با 5 رأس را در نظر بگیرید می‌خواهیم کوتاه‌ترین مسیر از s به z را به کمک الگوریتم بلمن‌فورد به دست آوریم.



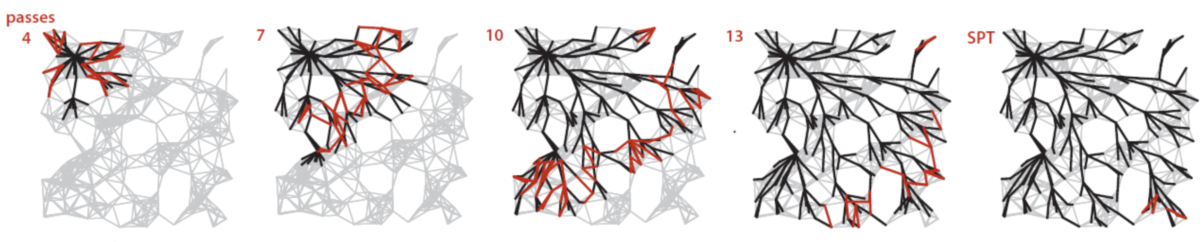
شکل2

شکل کاملا واضح است بنابرین ما به توضیحاتی مختصر بسنده می‌کنیم:

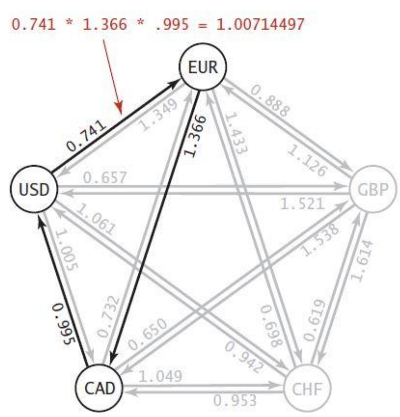
مقادیرd در هررأس نوشته شده و یال‌هایی که هاشور خورده اند معرف مسیری هستند که به کمک آن این اعداد به دست آمده‌اند.

قسمت a)گراف قبل از اجرای الگوریتم است و قسمت های b-e) گراف در حین اجرای الگوریتم را نشان می‌دهد که در قسمت eبر روی هر رأسکوتاه‌ترین‌مسیر از s به آن ها نوشته شده است.

شکل زیر اجرای بلمن-فورد روی یک گراف با ۲۵۰ راس را نشان می‌دهد. یال‌های قرمز رنگ در صف قرار دارند و هنوز به SPT اضافه نشده‌اند.



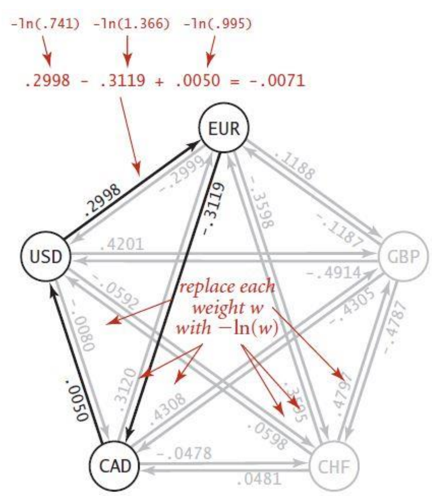
از کاربردهای الگوریتم بلمن-فورد می‌توان در تشخیص امکان آربیتراژ در بازارهای مالی اشاره کرد. آربیتراژ عبارت است از کسب سود از طریق اختلاف قیمت در دو بازار مختلف. به عنوان مثال گراف زیر را درنظر بگیرید.



هر رأس از این گراف نشان‌دهنده‌ی یک واحد پول و وزن هر یال نشانگر نرخ تبدیل دو رأس متصل به آن است. به عنوان مثال با هزار دلار آمریکا می‌توان ۷۴۱ یورو خرید. شما با ۷۴۱ یوروی موجود می‌توانید ۷۴۱*۱,۳۶۶

معادل ۱۰۱۲,۲۰۶ دلار کانادا بخرید. حال اگر شما این ۱۰۲۴,۲۰۶ دلار کانادا را به دلار آمریکا تبدیل کنید، حدود ۷ دلار سود خواهید کرد. زیرا $1012.206 \times 0.995 = 1007.14497$ و $1007.14497 - 1000 = 7.14497$. این مقدار ممکن زیاد به نظر نرسد ولی یک دلال پول ممکن است ۱ میلیون دلار در دقیقه وارد این چرخه کرده و دقیقه‌ای ۷ هزار دلار سود کند. این یک مثال از فرصت آربیتراژ بود که امکان سودهای کلان در بازارهای سرمایه را به افراد مختلف می‌دهد. این پدیده در بازارهای دیگر مانند بورس نیز اتفاق می‌افتد. اما این مسأله را چگونه می‌توان به مسأله‌ی کوتاه‌ترین مسیر و الگوریتم بلمن-فورد ارتباط داد؟

می‌توان گفت که مسأله‌ی آربیتراژ معادل مسأله‌ی یافتن یک دور منفی در یک گراف وزن‌دار است. برای حل مسأله از گراف تبدیل نرخ‌ها یک رونوشت تهیه می‌کنیم با این تفاوت که وزن یال‌ها برابر منفی لگاریتم وزن یال‌های گراف اصلی است. این تبدیل برای محاسبه‌ی وزن مسیرها مناسب است،زیرا درحقیقت وزن یال‌های موجود در مسیر را باید در یکدیگر ضرب کنیم. به عنوان مثال وزن یک مسیر با یال‌های w1, w2, w3, w4, w5, ... برابر $-\ln(w1)-\ln(w2)-\ln(w3)-\ln(w4)-\ln(w5)-...$ خواهد شد که معادل $-\ln(w1*w2*w3*w4*w5*...)$ است. حال در این رونوشت تهیه شده با استفاده از الگوریتم بلمن-فورد دور با وزن منفی را پیدا می‌کنیم. این دور یک فرصت آربیتراژ است.



4-6-8 – مسأله‌ی کوتاه‌ترین مسیر در گراف جهت‌دار بدون حلقه

در dagهای وزن‌دار با کمک مرتب‌سازی توپولوژیکی می‌توان کوتاه‌ترین مسیر را با هزینه‌ی زمانی $\Theta(V+E)$ محاسبه کرد. استفاده از این الگوریتم به جای استفاده از دایکسترا در dagها علاوه بر اینکه کوتاه‌ترین مسیر از یک مبدأ را در زمان خطی حل می‌کند، با وجود یال‌های منفی نیز دچار مشکل نخواهد شد.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

این الگوریتم با مرتب‌سازی توپولوژیکی dag شروع می‌شود. اگر dag یک مسیر از رأس u به رأس v داشته باشد، u در مرتب‌سازی زودتر از v ظاهر می‌شود. سپس براساس ترتیب توپولوژیکی رئوس، یال‌های خارج شونده از هر رأس را ریلکس می‌کنیم. شبه کد زیر مربوط به این الگوریتم است.

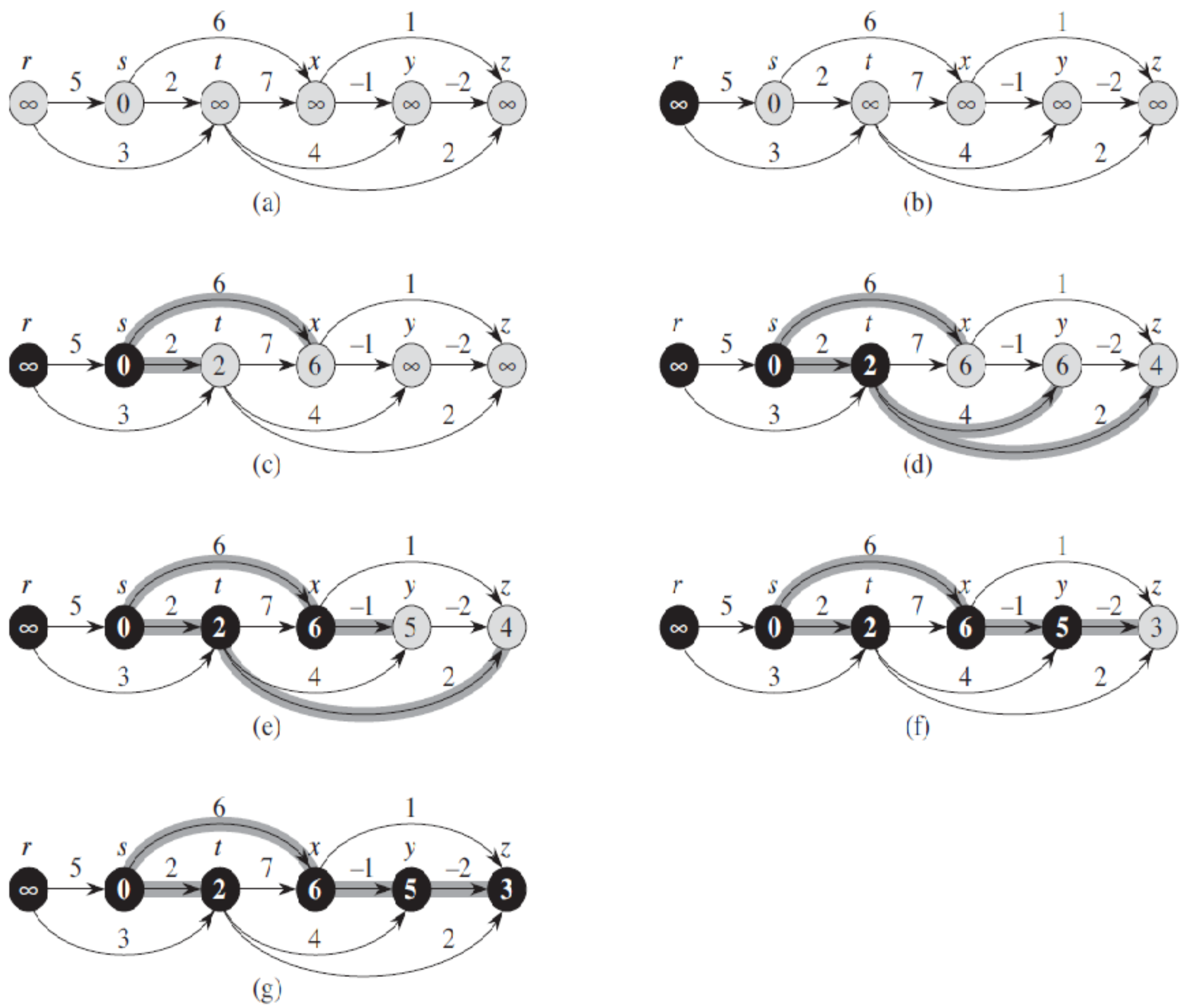
```
Begin
 Topologically sort the vertices of DAG
 Initialize single source
 For each vertex u, taken in topologically sorted order
   For each edge that leaves u do
     Relax edge
End
```

قضیه: به وسیله‌ی ریلکس کردن رئوس به ترتیب توپولوژیکی در یک dag می‌توان مسأله‌ی کوتاه‌ترین رأس از یک مبدأ را با هزینه‌ی زمانی V+E حل کرد.

هر یال مانند <v,w> دقیقا یک‌بار ریلکس می‌شود و با ریلکس شدن آن، نامساوی distTo[w] <= distTo[v] + e.weigt() برقرارخواهد شد. این نامساوی تا زمان اتمام اجرای الگوریتم برقرار خواهد ماند، زیرا distTo[v] تغییر نخواهد (چون بخاطر ترتیب توپولوژیکی بعد از ریلکس شدن یال‌های خروجی از v ، راس v دیگر آپدیت نخواهد شد). همچنین مقدار distTo[w] نیز تنها می‌تواند کاهش یابد، زیرا راس w تنها در صورتی آپدیت می‌شود که مقدار distTo[w] کمتر شود. بنابراین با اضافه شدن رأس‌هایی که از مبدأ قابل دسترس هستند، شرط بهینه بودن کوتاه‌ترین مسیر برقرار خواهد ماند.

بدست آوردن هزینه‌ی زمانی الگوریتم نیز به سادگی صورت می‌پذیرد. به یاد داریم که مرتب‌سازی توپولوژیکی با هزینه‌ی V+E صورت می‌گیرد. حلقه‌ای که وظیفه‌ی ریلکس کردن یال‌ها را برعهده دارد نیز با همین هزینه اجرا می‌شود. زیرا این حلقه هر یال را نهایتا یک‌بار ریلکس کرده و روی هر رأس نیز تنها یک‌بار اجرا می‌شود. پس هزینه‌ی نهایی اجرای الگورتیم V+E خواهد بود.

مثال زیر نحوه‌ی اجرای الگوریتم را در یک dag نشان می‌دهد:



با کمک این الگوریتم بدست آوردن طولانی‌ترین مسیر از یک رأس در یک dag نیز به سادگی قابل حل است. برای حل این مسأبه ابتدا یک رونوشت از dag مورد نظر تهیه می‌کنیم با این تفاوت که وزن یال‌ها در این رونوشت منفی وزن یال‌ها در گراف اصلی است. می‌توان گفت که کوتاه‌ترین مسیر در این رونوشت معادل طولانی‌ترین مسیر در گراف اصلی است. پس هزینه‌ی زمانی اجرای این الگوریتم نیز V+E خواهد بود.