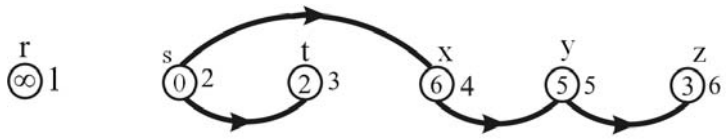
در شکل زیر گراف G_π که یک درخت کوتاهترین مسیرها می‌باشد نشان داده شده است. مقدار داخل هر رأس v برابر $d[v]$ است که این مقدار در انتها برابر $\delta(s,v)$ خواهد شد. شماره کنار هر رأس نشان دهنده ترتیبی است که رئوس بر حسب ترتیب توپولوژیکی در خط سوم (حلقه for) انتخاب و پردازش می‌شوند:



ترتیب پردازش رئوس طبق ترتیب توپولوژی از چپ به راست عبارتست از:

r, s, t, x, y, z

➢**قضیه:** اگر $G=(V,E)$ یک گراف جهت‌دار وزن دار با رأس مبداء s باشد که شامل هیچ دوری نیز نباشد آنگاه بعد از پایان رویه DAG-SHORTEST-PATH برای همه رئوس v در V داریم: $d[v]=\delta(s,v)$ و گراف G_π یک درخت کوتاهترین مسیرها خواهد بود.

کاربرد الگوریتم DAG-SHORTEST-PATH در PERT CHART

در یک PERT CHART، یال‌ها نشان دهنده کارهایی (jobs) هستند که باید انجام شوند و وزن یال‌ها نشان دهنده زمان‌های مورد نیاز برای اجرای کارهای خاص می‌باشند.

اگر یال (u,v) وارد رأس v شود و یال (v,x) از رأس v خارج شود، آنگاه کار (u,v) باید پیش از کار (v,x)انجام شود. یک مسیر در این DAG، نشان دهنده یک دنباله از کارهایی است که باید در ترتیب خاصی انجام شود.

یک مسیر بحرانی (critical path) طولانی‌ترین مسیر گذرنده از این DAG متناظر با طولانی ترین زمان اجرای یک دنباله مرتب از کارها می‌باشد. وزن یک مسیر بحرانی، یک کران پایین برای کل زمان اجرای همه کارها می‌باشد. یک مسیر بحرانی را می‌توان به صورت‌های زیر پیدا کرد:

۱) منفی‌کردن وزن یال‌ها و اجرای DAG-SHORTEST-PATH

۲) اجرای الگوریتم DAG-SHORTEST-PATH با اصلاح آن به این ترتیب که در خط (۲) الگوریتم INITIALIZE-SINGLE-SOURCE به جای ∞، $-\infty$ قرار داده و در رویه RELAX، $<$ را با $>$عوض نماییم.

الگوریتم دایکسترا برای یافتن کوتاه‌ترین مسیر از مبداء واحد

این الگوریتم مسأله کوتاهترین مسیرها از یک مبداء منفرد را نسبت به یک گراف که در آن همه یال‌ها دارای وزن غیر منفی باشد حل می‌کند. بنابراین برای هر $(u,v)\in E$ داریم: $W(u,v)\geq 0$

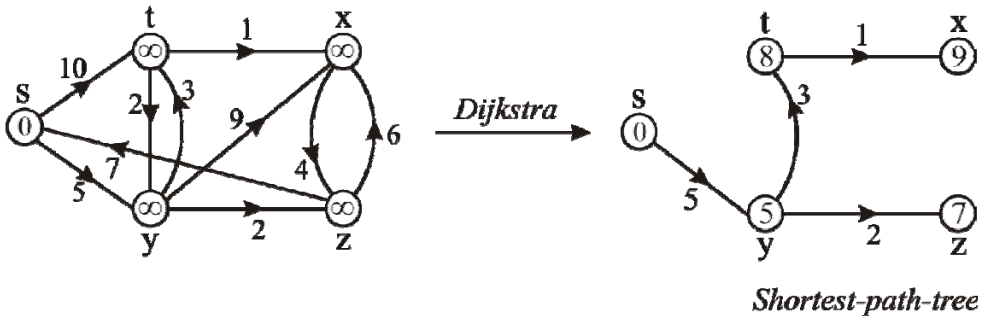
یک پیاده‌سازی خوب از الگوریتم دایکسترا باعث می شود که زمان اجرای آن از الگوریتم بلمن- فورد بهتر شود.

این الگوریتم یک مجموعه S حاوی رئوسی را که تا به حال کوتاهترین مسیر از مبدأ s به آنها به دست آمده را نگهداری می‌کند. الگوریتم به طور متوالی رأس u، $u\in V-S$ را با می نیمم مقدار تخمین کوتاهترین مسیر $(d[u])$ انتخاب و u را به S اضافه کرده و سپس همه یال‌هایی را که از u خارج شده‌اند را relax می‌کند.

برای نگهداری مقادیر d، از صف اولویت دار Q استفاده شده است.

DIJKSTRA(G, W, S)
۱) INITIALIZE-SINGLE-SOURCE(G,S)
۲) $S \leftarrow \varnothing$
۳) $Q \leftarrow V[G]$
۴) WHILE $Q \neq \varnothing$
۵) do Extract-MIN(Q)
۶) $S \leftarrow S \cup \{u\}$
۷) for each vertex $v \in Adj[u]$
۸) do RELAX(u,v,w)

مثال:



آنالیز زمانی الگوریتم Dijkstra

زمان اجرای الگوریتم به پیاده سازی صف اولویت دار Q بستگی دارد:

۱- اگر Q به صورت یک آرایه عادی در نظر گرفته شود زمان اجرای الگوریتم عبارتست از:

$$O(|V|^2 + |E|) = O(|V|^2)$$

۲- اگر گراف بحد کافی خلوت (sparse) باشد و در حالت خاص $|E| = O(\frac{|V|^2}{\log|V|})$ آنگاه پیاده سازی صف اولویت دار با یک $\min - heap$ عملی است و در این صورت زمان اجرا عبارتست از:

$$O\big((|V|+|E|)\log|V|\big)$$

چنانچه همه رئوس از s (مبداء) قابل دسترسی باشند این زمان می تواند به $O(|E|\log|V|)$ کاهش یابد که همان اصلاح شده پیاده‌سازی ذکر شده با زمان $O(|V|^2)$ می‌باشد.

۳- چنانچه Q با یک Fibonacci heap پیاده سازی شود، زمان اجرا عبارتست از:

$$O(|V|\log|V| + |E|)$$

فصل پنجم

روش بازگشت به عقب (BT)

عناوین اصلی

- مسأله n- وزیر
- محدودیت‌های صریح
- محدودیت‌های ضمنی
- زمانبندی پردازنده
- جنگل‌های مجموعه‌های مجزا