طراحی الگوریتم

جزوه چهارم - الگوریتم‌های گراف

الگوریتم‌های گراف مجموعه‌ای از تکنیک‌ها هستند که برای تحلیل و حل مسائلی که با گراف‌ها مدل می‌شوند به کار می‌روند. گراف‌ها از مجموعه‌ای از گره‌ها (رأس‌ها) و یال‌ها (اتصال‌ها) تشکیل شده‌اند و در نمایش سیستم‌های پیچیده مانند شبکه‌های اجتماعی، جاده‌ها، شبکه‌های کامپیوتری و روابط منطقی بسیار کاربرد دارند. این الگوریتم‌ها برای مسائلی همچون پیدا کردن کوتاه‌ترین مسیر (الگوریتم Dijkstra و *A)، جستجوی گره‌ها یا ارتباطات (DFS و BFS) یافتن مولفه‌های متصل، تعیین درخت پوشای کمینه (الگوریتم‌های Prim و Kruskal) و محاسبه بیشترین جریان شبکه (الگوریتم Ford-Fulkerson) استفاده می‌شوند.

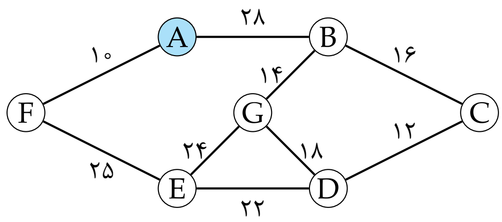
کاربردهای عملی این الگوریتم‌ها شامل سیستم‌های مسیریابی GPS برای پیدا کردن بهترین مسیر، تحلیل شبکه‌های اجتماعی برای شناسایی افراد تأثیرگذار یا کشف اجتماعات، طراحی شبکه‌های کامپیوتری برای بهینه‌سازی انتقال داده و حتی در هوش مصنوعی و یادگیری ماشین برای تحلیل گراف‌های دانش یا مدل‌های توصیه‌گر است. همچنین در علوم زیستی برای تحلیل شبکه‌های زیستی، در اقتصاد برای بررسی شبکه‌های مالی و حتی در مهندسی برای طراحی مدارهای الکتریکی کاربرد دارند.

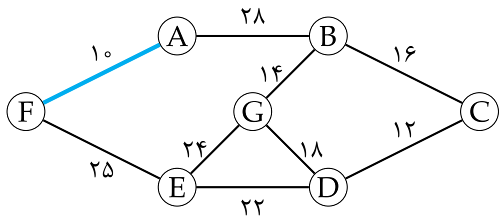
ویژگی کلیدی این الگوریتم‌ها این است که برای انواع گراف‌ها (جهت‌دار، غیرجهت‌دار وزن‌دار، بدون وزن) طراحی شده‌اند و بسته به پیچیدگی مسأله و ساختار گراف، سرعت و دقت مختلفی دارند. از سوی دیگر، با پیشرفت تکنولوژی، استفاده از این الگوریتم‌ها در تحلیل داده‌های بزرگ و شبکه‌های عظیم رشد چشمگیری داشته است.

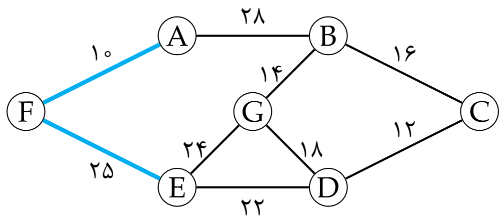
۱. الگوریتم‌های یافتن درخت پوشای کمینه

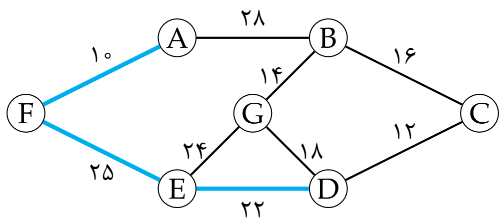
الگوریتم پریم

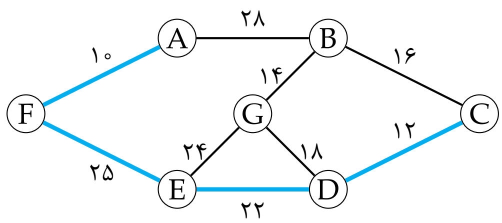
الگوریتم پریم، الگوریتمی برای پیدا کردن درخت پوشای کمینه یک گراف وزن‌دار است.
شرح: در روش پریم ابتدا یک راس را انتخاب می‌کنیم. سپس کم وزن‌ترین یال‌های این راس را به عنوان یک یال درخت انتخاب می‌کنیم. حال بین یال های این دو راس کم‌وزن‌ترین یال را پیدا کرده و به عنوان عضوی از درخت انتخاب می‌کنیم. حال بین یال‌های این سه راس کم‌وزن‌ترین را انتخاب می‌کنیم. همینطور ادامه می‌دهیم تا درخت کامل شود. باید توجه داشت که یالی که هر بار اضافه می‌کنیم دور ایجاد نکند یا به عبارت دیگر یک سر آن در بین راس‌های غیر اتخاب شده باشد. در ادامه یک مثال برای اجرای این الگوریتم را می‌بینید.

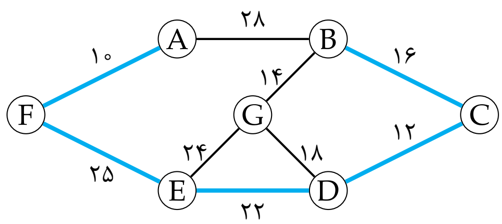

۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲

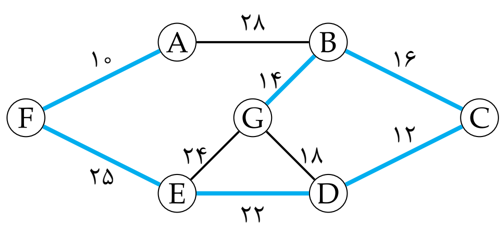

۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲


۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲


۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲


۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲


۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲


۲۸
A
B
۱۰
۱۴
۱۶
F
G
C
۲۴
۱۸
۱۲
۲۵
E
D
۲۲

صحت: ادعایی ارائه می‌دهیم که صحت الگوریتم را در هر مرحله ثابت می‌کند.

ادعا: اگر گراف را به دو مجموعه V و V' از راس‌ها افراز کنیم، کم‌وزن‌ترین یال بین یال‌هایی که یک طرفشان در مجموعه V و طرف دیگرشان در مجموعه V' است باید جزء درخت پوشای کمینه باشد (اگر چند یال با کم‌ترین وزن وجود داشت، باید دقیقا یکی از آن‌ها جزء درخت پوشای کمینه باشد).

اثبات: اگر یالی بین این دو مجموعه انتخاب نشود، گراف همبند نمی‌شود، اگر ۲ یال یا بیشتر انتخاب شود، با فرض همبندی در هر دو مجموعه دور ایجاد خواهد شد، و اگر یالی که کمترین وزن را ندارد انتخاب شود، می‌توان با عوض کردن آن با کم‌وزن‌ترین ویژگی‌های درخت را حفظ کرد و مجموع وزن‌ها را کاهش داد، پس کم‌وزن‌ترین انتخاب می‌شود.

شبه کد:

Algorithm ۱ Prim's algorithm

Input: A graph G with vertices and edges، and a source node.
Output: Minimum Spanning Tree (MST).
Initialize a priority queue (min-heap).
Initialize an empty set MST (to store the edges of the minimum spanning tree).
Initialize a set `visited` to track visited nodes.
Add the source node to the priority queue with a cost of 0.
while the priority queue is not empty **do**
    Extract the node u with the smallest cost from the priority queue.
    if u is already in `visited` **then**
        continue.
    end if
    Add u to `visited`.
    Add the edge leading to u to MST (if u is not the source).
    for each neighbor v of u **do**
        if v is not in `visited` **then**
            Add v to the priority queue with the cost of the edge (u, v).
        end if
    end for
end while
return MST

پیچیدگی الگوریتم: یک روش خوب برای بهینه کردن الگوریتم نگه داشتن کم‌ترین فاصله‌ی هر راس تا راس‌های انتخابیست. وقتی یک راس به مجموعه‌ی ما اضافه شد. فاصله‌ی بقیه راس‌ها را به روز می‌کنیم. در این صورت هر بار برای پیدا کردن راس نزدیک‌تر و به روز رسانی $O(n)$ عملیات انجام می‌دهیم و چون n بار این کار انجام می‌دهیم، پیچیدگی کل الگوریتم $O(n^2)$ می‌شود. اگر فاصله‌هر راس تا نزدیک‌ترین راس از بین راس‌هایی که در مجموعه قرار گرفته‌اند را در داه‌ساختاری مناسب ذخیره کنیم می‌توانیم پیچیدگی الگوریتم را بهتر کنیم. اگر این داده‌ساختار هرم کمینه باشد، چون حذف و اضافه از $O(logn)$ است و به ازای یال حداکثر یک بار عملیات اضافه کردن رخ می‌دهد و چون قطعا تعداد عملیات حذف کم‌تر از تعداد عملیات اضافه کردن است پیچیدگی الگوریتم به $O(mlogn + n)$ تغییر می‌کند که برای حالاتی که تعداد یال‌ها از $\theta(\frac{n^2}{logn})$ کم‌تر باشد پیچیدگی کل کاهش پیدا می‌کند. حال اگر از هرم فیبوناچی استفاده کنیم پیچیدگی به $O(m + n)$ کاهش پیدا می‌کند.

الگوریتم کروسکال

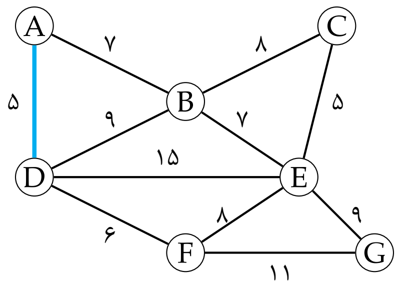
الگوریتم کروسکال، الگوریتم برای پیدا کردن درخت پوشای کمینه یک گراف وزن‌دار است. این الگوریتم بر خلاف الگوریتم پریم لزوما اجزایی که در حین اجرا جزء درخت پوشای کمینه تشخیص می‌دهد همبند نیستند و تنها تضمین می‌کند که در پایان این شرط برقرار است.

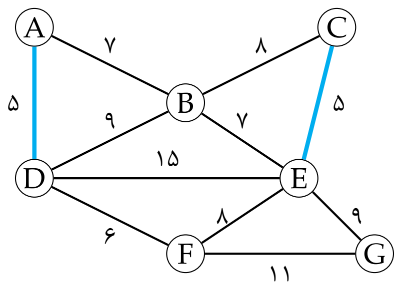
شرح: در این الگوریتم، یال‌ها را بر اساس وزن به صورت صعودی مرتب می‌کنیم. سپس از اولین یال شروع می‌کنیم و هر یالی که با یال‌هایی که قبلا انتخاب کردیم دور نمی‌سازد را انتخاب می‌کنیم. تا جایی که درخت تشکیل شود.

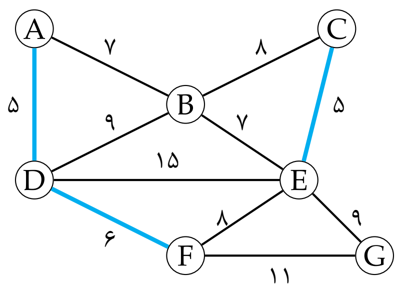
اما چگونه متوجه شویم که یال جدید با یال‌های انتخابی قبلی دور نمی‌سازد؟ کافیست گرافی که تا کنون از یال‌های انتخابی تشکیل شده را به مؤلفه‌های همبندی تقسیم کنیم و اگر یال جدید هر دو سر آن در یک مؤلفه بود، یال جدید دور ایجاد می‌کند و در غیر اینصورت اضافه کردن یال جدید مشکلی ندارد.

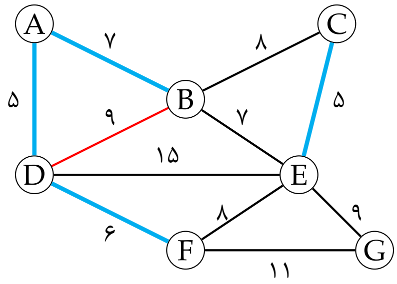
هر راس متغیری همراه خود دارد که شماره مولفه‌ی همبندیش را نشان می‌دهد. در ابتدا هر راس، خود یک مؤلفه‌ی همبندیست، و وقتی یک یال بین دو مؤلفه ارتباط ایجاد می‌کند باید شماره مولفه‌ی همبندی هر دو گروه یکی شود.

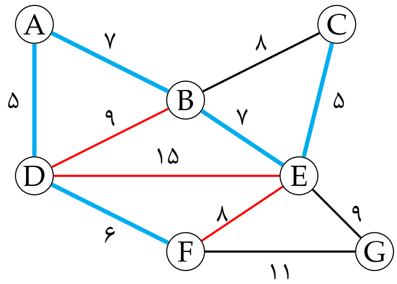
در شکل زیر یک مثال برای اجرای این الگوریتم را می‌بینید. توجه کنید یال‌های انتخاب شده تنها در آخر کار یک مجموعه‌ی همبند تشکیل می‌دهند ولی در الگوریتم پریم همیشه در طول ساخت درخت، مجموعه ما همبند بود.

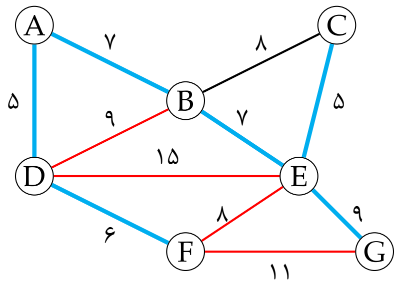












صحت: به برهان خلف فرض کنید درخت پوشای کمینه‌ی T وجود دارد که مجموع وزن یال‌های ساخته شده توسط آن کمتر از درخت تولید شده توسط الگوریتم کروسکال به نام G باشد. کم‌وزن‌ترین یالی در که عضو T است ولی عضو G نیست را انتخاب کنید. این یال حتما توسط الگوریتم کروسکال انتخاب می‌شد مگر اینکه با یال‌های قبلی دور بسازد و چون تمام یال های سبک تر از این یال در هر دو درخت وجود دارد به معنی آن است که در درخت T دور وجود دارد که تناقض است.

شبه کد:

Algorithm ۲ Kruskal's algorithm

Input: A graph G with vertices and edges.
Output: Minimum Spanning Tree (MST).
Initialize MST as an empty set.
Sort all edges in G by their weight in non-decreasing order.
Create a disjoint-set for all vertices.
for each edge (u, v) in the sorted edge list **do**
    if `find(u)` $\neq$ `find(v)` **then** ▷ Check if u and v belong to different sets
        Add edge (u, v) to MST.
        `Union` the sets of u and v.
    end if
end for
return MST

پیچیدگی الگوریتم: مرتب کردن یال ها و بررسی یال‌ها از $O(m+mlogn)$ است که برابر $O(mlogn)$ می‌باشد و هربار اتصال دو مولفه از $O(n)$ است، که چون اتصال $n-1$ بار انجام می‌شود، پیچیدگی الگوریتم $O(mlogn+n^2)$ می‌شود.

۲. الگوریتم‌های یافتن کوتاه‌ترین مسیر

الگوریتم دایکسترا

الگوریتم دایکسترا راه‌کاری برای پیدا کردن کم‌وزن مسیر از رأس مشخص آغاز به بقیه رئوس در گراف جهت‌دار و وزن‌دار (با وزن‌های مثبت) می‌دهد. وزن یک مسیر در گراف وزن‌دار برابر مجموع وزن یال‌های آن است. جهت‌دار نبودن یال‌ها هم مشکلی ایجاد نمی‌کند و می‌توان برای یال‌های غیر جهت‌دار دو یال فرض کرد.

شرح: فرض کنید $1 \leq s \leq n$ که در آن s رأس آغاز است و فرض کنید:

$$dist(s) = 0$$

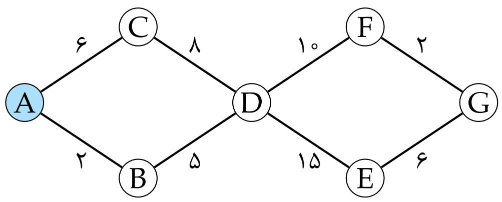
و به ازای هر $v \neq s$:

$$dist(v) = \infty$$

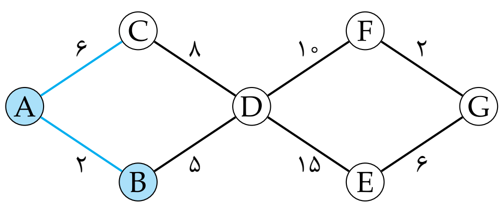
فرض کنید مجموعه‌ی T برابر رئوسی باشد که تا کنون کم وزن‌ترین مسیر آن‌ها را پیدا کرده‌ایم. این الگوریتم در هر مرحله نزدیک‌ترین رأس به s را که تا کنون به مجموعه‌ی T اضافه نشده را انتخاب می‌کند (مثلا x) و آن را به مجموعه‌ی T اضافه می‌کند و فاصله‌ی دیگر رأس‌ها را با توجه به فاصله‌ی x بروز می‌کند. به ازای هر رأس v خارج T:

$$dist(v) = min(dist(v), dist(x) + w(x, v))$$

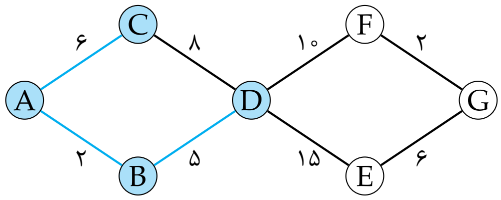
که در آن $w(x, v)$ برابر وزن یال بین x و v است. این الگوریتم در هر مرحله رأسی را که کوتاه‌ترین فاصله‌ی آن تا s پیدا شده است را به T اضافه می‌کند، زیرا کوتاه ترین مسیر این رأس از یکی از رأس‌های T می‌گذرد که در مراحل قبلی فاصله آن‌ها بدست آمده و دیگر رئوس را بروز کرده‌اند.
در صورت منفی بودن یال‌ها فرض اینکه در هر مرحله رأسی که کوتاه‌ترین مسیر آن پیدا شده اضافه می‌شود زیر سوال می‌رود. زیرا این رأس می‌تواند بدون استفاده از رأس‌های T و یال‌های منفی مسیری کوتاه‌تر به s پیدا کند.
در ادامه یک مثال برای اجرای این الگوریتم را می‌بینید.



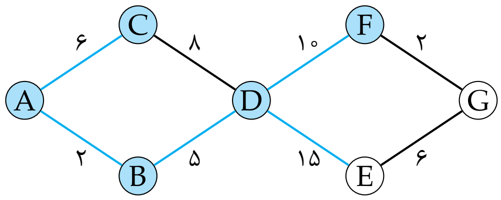
Vertex	A	B	C	D	E	F	G
Visited	✓	×	×	×	×	×	×
Distance	0	∞	∞	∞	∞	∞	∞



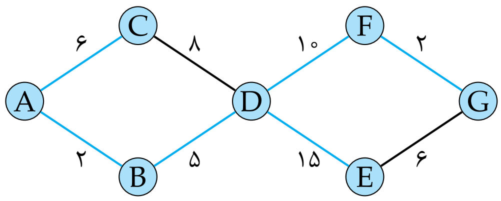
Vertex	A	B	C	D	E	F	G
Visited	✓	✓	×	×	×	×	×
Distance	0	2	∞	∞	∞	∞	∞



Vertex	A	B	C	D	E	F	G
Visited	✓	✓	✓	✓	×	×	×
Distance	0	2	6	7	∞	∞	∞



Vertex	A	B	C	D	E	F	G
Visited	✓	✓	✓	✓	×	✓	×
Distance	0	2	6	7	∞	17	∞



Vertex	A	B	C	D	E	F	G
Visited	✓	✓	✓	✓	✓	✓	✓
Distance	0	2	6	7	22	17	19

شبه کد:

Algorithm ۳ Dijkstra's algorithm

```
Input: A graph G with vertices، edges and a start vertex s.
Output: Shortest distance to all vertices from s.
Initialize Graph.
dist[s] = 0
for each vertex v in vertices do
    if v ≠ s then
        dist[v] = ∞
    end if
end for
while Size(T) ≠ n do
    u =: vertex not in T with min dist[u]
    add u to T
    for each neighbor v of u do
        dist[v] = min(dist[v], dist[u] + w[u][v])
    end for
end while
```

پیچیدگی الگوریتم: به ازای هر رأس باید روند بالا را طی کنیم. یعنی دنبال آن بگردیم و همه‌ی همسایه‌های آن را پیمایش کنیم. پس پیچیدگی زمانی برنامه از $O(n \times n) = O(n^2)$ است. هر چند می‌توان با استفاده از داده ساختار هرم پیاده‌سازی از $O(e \times \log(n))$ ارائه داد.

الگوریتم بلمن_فورد

الگوریتم بلمن_فورد راه‌کاری برای پیدا کردن کم‌وزن‌ترین مسیر از رأس مشخص آغاز به بقیه رئوس در گراف جهت‌دار و وزن‌دار می‌دهد. وزن یک مسیر در گراف وزن‌دار برابر مجموع وزن یال‌های آن است. جهت‌دار نبودن یال‌ها هم مشکلی ایجاد نمی‌کند و می‌توان برای یال‌های غیر جهت‌دار دو یال فرض کرد.

یکی از مزیت های این الگوریتم نسبت به الگوریتم دایکسترا توانایی اجرا شدن روی گراف‌ها با یال منفی است.
کاربردها: این الگوریتم علاوه بر پیدا کردن کوتاه‌ترین مسیر به پیدا کردن دور منفی در گراف‌ها کمک می‌کند. بنابراین استفاده‌های بیشتری از الگوریتم‌های مشابه می‌تواند داشته باشد.
شرح: فرض کنید $1 \leq s \leq n$ که در آن s رأس آغاز است و فرض کنید:

$$dist(s) = 0$$

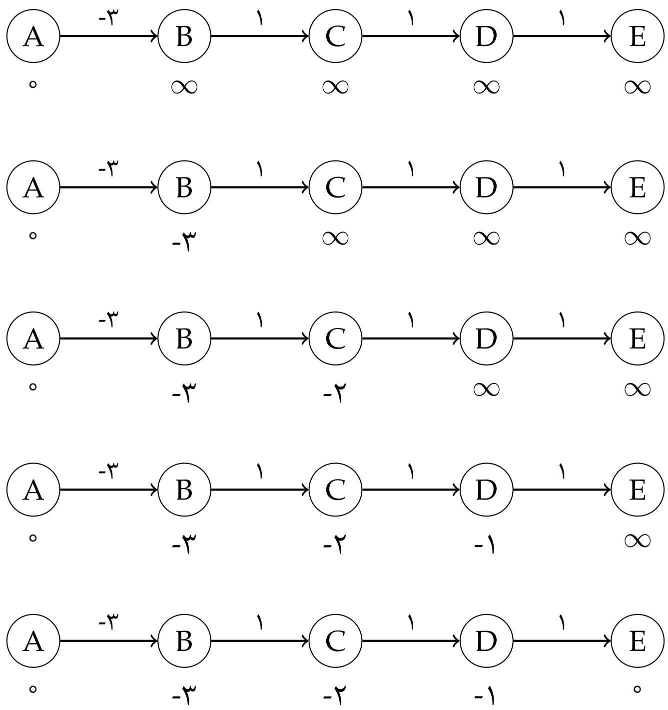
و به ازای هر $v \neq s$:

$$dist(v) = \infty$$

این الگوریتم به تعداد یکی کمتر از رأس ها در هر مرحله روی همه‌ی یال‌ها عملیات زیر را انجام می‌دهد:

$$dist(u) = min(dist(u), dist(v) + w(v, u))$$

در واقع فاصله دو سر یال را با توجه به وزن آن تصحیح می‌کند. به این عملیات در اصطلاح $Relax$ کردن یال‌ها می‌گویند.
صحت: درستی این الگوریتم را به استقرا ثابت می‌کنیم. فرض کنید این الگوریتم تا i ــامین بار اجرا شدن کوتاه‌ترین فاصله تمامی رئوسی که کم‌وزن‌ترین مسیر آن‌ها حداکثر i یال دارد را پیدا کند. پایه استقرا: در مرحله صفر ــام رأس آغاز فاصله‌اش صفر است؛ پس درست است. گام استقرا: هر یک از رأس‌هایی که کوتاه‌ترین مسیرشان حداکثر $i + 1$ یال دارد آخرین یالشان حتما به یک رأسی است که در مرحله قبلی فاصله‌شان پیدا شده است(در واقع حداکثر از i یال استفاده می‌کنند). پس بعد از $Relax$ کردن یال ها برای بار $i + 1$ ــام جواب تمامی رأس‌هایی که کوتاه‌ترین مسیرشان $i + 1$ یالی است را پیدا کرده‌ایم. پس درستی الگوریتم ثابت می‌شود.
در ادامه یک مثال برای اجرای این الگوریتم را می‌بینید.



شبه کد:

Algorithm ۴ Bellman-Ford's algorithm

Input: A graph G with vertices، edges and a start vertex s.
Output: Shortest distance to all vertices from s.
Initialize Graph.
for each vertex v in vertices **do**
  if $v \neq s$ **then**
    $dist[v] = \infty$
  end if
end for
Relax edges repeatedly.
for i from 1 to $size(vertices) - 1$ **do**
  for each edge (u, v) with weight w in edges **do**
    if $dist[u] + w < dist[v]$ **then**
      $dist[v] = dist[u] + w$
    end if
  end for
end for

پیچیدگی الگوریتم: به ازای هر رأس باید روند بالا را طی کنیم. یعنی دنبال آن بگردیم و همه‌ی همسایه‌های آن را پیمایش کنیم. پس پیچیدگی زمانی برنامه از $O(n \times n) = O(n^2)$ است. هر چند می‌توان با استفاده از داده ساختار هرم پیاده‌سازی از $O(e \times \log(n))$ ارائه داد.

الگوریتم فلوید_وارشال

فرض کنید یک گراف جهت‌دار وزن‌دار با مجموعه رئوس $\{1, 2, ..., n\}$ داریم. اگر گراف بدون جهت باشد، می‌توان از هر یال، 2 یال جهت‌دار ساخت و الگوریتم را اجرا کرد. وزن یال‌های گراف می‌تواند مثبت یا منفی باشد، اما گراف نباید دور با مجموع وزن منفی داشته باشد.
وزن یک مسیر را، مجموع وزن یال‌های آن مسیر می‌نامیم. فاصله‌ی بین دو رأس را وزن کوتاه‌ترین (کم‌وزن‌ترین) مسیر بین آن‌ها در نظر می‌گیریم. می‌خواهیم به ازای هر دو رأس از این گراف، فاصله‌ی بین آن دو رأس را پیدا کنیم. الگوریتم فلوید_وارشال، راه حلی برای این مسئله ارائه می‌کند.
شرح: فرض کنید $1 \leq i, j, k \leq n$ باشد. تمام مسیرهایی را از رأس i به رأس j در نظر بگیرید که رأس‌های میانی مسیر، فقط بتوانند از رأس‌های $\{1, 2, ..., k\}$ باشند. مقدار $dist(i, j, k)$ را وزن کوتاه‌ترین (کم‌وزن‌ترین) مسیر در بین این مسیرها می‌گوییم. می‌خواهیم مقادیر $dist$ را به ازای i, j, k های مختلف پیدا کنیم. برای $k = 0$ می‌دانیم:

$$dist(i, i, 0) = 0$$

و اگر i به j یال با وزن w داشته باشد:

$$dist(i, j, 0) = w$$

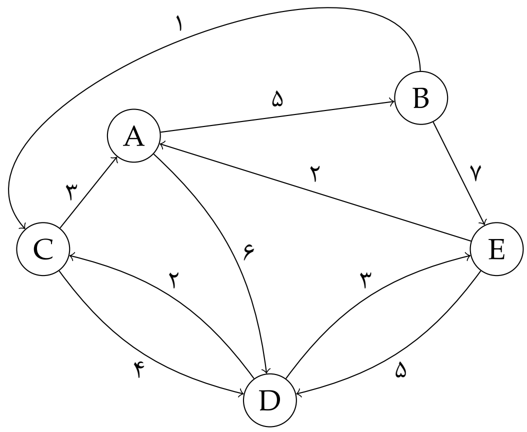
و اگر i به j یال نداشته باشد:

$$dist(i, j, 0) = \infty$$

حال مقادیر $dist$ را به ازای $k \geq 1$ حساب می‌کنیم. فرض کنید به ازای یک k، تمام مقادیر $dist(i,j,k)$ را به ازای i,j های مختلف، می‌دانیم. با این فرض، می‌خواهیم تمام مقادیر $dist(i,j,k+1)$ را به ازای i,j های مختلف بیابیم. دو رأس a,b را در نظر بگیرید. می‌خواهیم $dist(a,b,k+1)$ را بیابیم. مسیرهایی که باید برای محاسبه‌ی $dist(a,b,k+1)$ در نظر گرفته شوند، دو حالت دارند؛ یا شامل رأس میانی $k+1$ نیستند یا شامل رأس میانی $k+1$ هستند. کم‌وزن‌ترین مسیری که شامل رأس میانی $k+1$ نیست، وزن $dist(a,b,k)$ را دارد و کم‌وزن‌ترین مسیری که شامل رأس میانی $k+1$ است، وزن $dist(a,k+1,k)+dist(k+1,b,k)$ را دارد (زیرا باید ابتدا از a به $k+1$ برویم و سپس به b برویم.) پس:

$$dist(a,b,k+1) = min(dist(a,b,k), dist(a,k+1,k) + dist(k+1,b,k))$$

در ادامه یک مثال برای اجرای این الگوریتم را می‌بینید.



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

$$A_1 = \begin{bmatrix} 0 & 5 & \infty & 6 & \infty \\ \infty & & & & \\ 3 & & & & \\ \infty & & & & \\ 2 & & & & \end{bmatrix} \qquad A_1 = \begin{bmatrix} 0 & 5 & \infty & 6 & \infty \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & \infty & 5 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} & 5 & & & \\ \infty & 0 & 1 & \infty & 7 \\ & 8 & & & \\ & \infty & & & \\ & 7 & & & \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 5 & 6 & 6 & 12 \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & 15 \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} & & 6 & & \\ & & 1 & & \\ 3 & 8 & 0 & 4 & 15 \\ & & 2 & & \\ & & 8 & & \end{bmatrix} \qquad A_3 = \begin{bmatrix} 0 & 5 & 6 & 6 & 12 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 15 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} & & & 6 & \\ & & & 5 & \\ & & & 4 & \\ 5 & 10 & 2 & 0 & 3 \\ & & & 5 & \end{bmatrix} \qquad A_4 = \begin{bmatrix} 0 & 5 & 6 & 6 & 9 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 7 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 7 & 5 & 0 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} & & & & 9 \\ & & & & 7 \\ & & & & 7 \\ & & & & 3 \\ 2 & 7 & 7 & 5 & 0 \end{bmatrix} \qquad A_5 = \begin{bmatrix} 0 & 5 & 6 & 6 & 9 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 7 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 7 & 5 & 0 \end{bmatrix}$$

شبه کد:

Algorithm ۵ Floyd-Warshall's algorithm

Input: A graph G with vertices and weighted edges.
Output: Shortest paths between all pairs of vertices.
Let dist be a $3D$ array of minimum distances initialized to ∞
for i from 1 to n **do**
　$dist[i][i][0] = 0$
end for
for each edge from vertex a to vertex b **do**
　$dist[a][b][0] = w$　　$\triangleright$ w is the weight of u, v edge
end for
for k from 1 to n **do**
　for i from 1 to n **do**
　　for j from 1 to n **do**
　　　$dist[i][j][k] = min(dist[i][j][k-1], dist[i][k][k-1] + dist[k][j][k-1]$
　　end for
　end for
end for

پیچیدگی الگوریتم: به ازای هر k باید روند بالا را طی کنیم. به ازای هر k نیز باید $dist$ بین هر ۲ رأس را محاسبه کنیم. پس پیچیدگی زمانی برنامه از $O(n \times n^2) = O(n^3)$ است. برای حافظه نیز یک آرایه‌ی ۳ بعدی از $O(n^3)$ نیاز داریم. پس حافظه نیز از $O(n^3)$ است. هر چند جلوتر راه‌کاری برای کم‌کردن حافظه ارائه می‌شود.
با کمی دقت در می‌یابیم اگر بعد سوم حافظه را نیز در نظر نگیریم، برنامه به درستی کار می‌کند. پس حافظه را می‌توان از $O(n^2)$ در نظر گرفت.

۳. مسیر شبکه

در مسیریابی اینترنت، تأخیرهایی در خطوط و روترها وجود دارد که تأخیر روترها بسیار قابل توجه‌تر است. فرض کنید علاوه بر داشتن طول خطوط (یال‌ها) $\{l_e : e \in E\}$، گراف شبکه همچنین هزینه‌هایی برای روترها (رأس‌ها) $\{c_v : v \in V\}$ دارد. هزینه یک مسیر را به صورت مجموع طول خطوط آن مسیر به‌علاوه هزینه تمامی روترهای موجود در آن (شامل روترهای ابتدا و انتها) تعریف می‌کنیم. الگوریتمی کارآمد برای مسئله زیر ارائه دهید.

- **ورودی:** یک گراف شبکه جهت‌دار $G = (V, E)$؛ طول خطوط شبکه (l_e) و هزینه روترهای شبکه (c_v)؛ یک رأس شروع $s \in V$.
- **خروجی:** یک آرایه dist به گونه‌ای که برای هر رأس u، مقدار $\mathrm{dist[u]}$ هزینه کوتاه‌ترین مسیر از s به u باشد. توجه کنید که $\mathrm{dist[s]} = c_s$ است.

پاسخ :

الگوریتم دایکسترا را پیاده‌سازی می‌کنیم. فرض کنیم مجموعه‌های S و $V-S$ را داریم (S شامل همه رئوسی که کوتاه‌ترین مسیر به آن‌ها از رأس s پیدا شده و $V-S$ شامل باقی رئوس). ابتدا مجموعه S خالی است و $dist[s] = c_s$ و به ازای باقی رئوس $dist[v] = \infty$ است. در گام‌های بعد عمل زیر را تکرار می‌کنیم:

رأس v با کم‌ترین مقدار $dist[v]$ را از مجموعه $V - S$ انتخاب می‌کنیم، سپس این رأس را به مجموعه S اضافه می‌کنیم. در نهایت آرایه $dist[v]$ را به این صورت به‌روزرسانی می‌کنیم:

به ازای هر رأس v در $V - S$، $dist[v] = min(dist[v], dist[u] + w(u, v) + c_v)$

زمانی که همه رئوس در مجموعه S باشند، کوتاه‌ترین مسیر از s به همه آن‌ها در آرایه $dist$ وجود دارد.

۴. درخت پوشای کمینه

نشان دهید که برای گرافی با وزن‌های متمایز برای یال‌ها، یک درخت پوشای کمینه یکتا وجود دارد.

پاسخ :

با برهان خلف ثابت می‌کنیم:

فرض کنیم برای چنین گرافی حداقل دو درخت پوشای کمینه وجود داشته باشد. این دو درخت را A و B می‌نامیم. حال یال‌های درخت‌های A و B را به ترتیب وزن مرتب می‌کنیم. اولین جایی که یال‌های دو درخت متفاوت می‌شوند را در نظر می‌گیریم (برای فرض یال n ام). فرض کنیم این یال در درخت A e_1 و در درخت B e_2 نام دارد، همچنین وزن e_1 کمتر از وزن e_2 است (به دلیل تمایز وزن‌های گراف اصلی). حال در صورتی که یال e_1 را به درخت B اضافه کنیم، یک دور در این درخت تشکیل می‌شود. یال‌های دیگر این دور را در نظر می‌گیریم، در صورتی که همه این یال‌ها در درخت A نیز بودند، در آن یک دور تشکیل می‌شد. در نتیجه حداقل یکی از این یال‌ها در درخت A نبوده‌است. این یال را e_3 می‌نامیم. می‌دانیم وزن e_3 بیشتر از وزن e_1 است به این دلیل که e_1 کمترین وزن بین یال‌هایی را دارد که فقط در یکی از درخت‌ها موجود است. حال اگر یال e_3 را از درخت B حذف کرده و به

جای آن یال e_1 را در درخت بگذاریم، مجموع وزن یال‌های این درخت کمتر می‌شود. پس توانستیم به درختی برسیم که وزن‌های کمتری از درخت پوشای کمینه B دارد که این موضوع تناقض است.

۵. کوتاه‌ترین مسیر

آیا کوتاه‌ترین مسیر بین هر دو رأس در درخت پوشای کمینه $T = (V, E')$ از یک گراف $G = (V, E)$ که گرافی متصل وزن‌دار و بدون جهت است، کوتاه‌ترین مسیر بین همان دو رأس در G است؟ ثابت کنید. فرض کنید وزن تمامی یال‌های G یکتا و بزرگتر از صفر هستند.

پاسخ :

با یک مثال نقض ثابت میکنیم که حکم درست نیست. گراف کامل K_n را در نظر بگیرید که اگر وزن هر یال را با w_i نشان دهیم، $1.5 <= w_i < 2$ است. حال میدانیم در درخت پوشای کمینه دو راس وجود دارند که با یک یال مستقیم به هم وصل نیستند(در غیر این صورت یعنی همه یال‌ها وجود دارند و درخت نمیشد)، این بدین معناست که حداقل دو یال میان آن‌ها وجود دارد و چون وزن هر یال حداقل 1.5 است، پس وزن کوتاه ترین مسیر بین آن ها حداقل 3 است که میدانیم این مقدار در گراف اصلی عددی بین 1.5 و 2 است که تناقض است.

۶. تعداد کوتاه‌ترین مسیر

اغلب اوقات کوتاه‌ترین مسیر بین دو رأس یک گراف یکتا نیست. الگوریتم دایکسترا را به گونه‌ای تغییر دهید که علاوه بر محاسبه کوتاه‌ترین مسیرها، تعداد کوتاه‌ترین مسیرهای متمایز را از یک رأس شروع s به تمامی رأس‌ها نیز پیدا کند. فرض کنید گراف دارای وزن‌های مثبت است.

پاسخ :

یک آرایه به سایز N (تعداد رئوس) به نام $paths[N]$ تعریف میکنیم و مقدار اولیه تمام خانه های آن را صفر می گذاریم بجز خانه s و مقدار $paths[s] = 1$ میگذاریم. حال الگوریتم را اینطور تغییر می دهیم که وقتی

$dist[v] = dist[u] + w(u, v)$ بود، $paths[v] += paths[u]$ میکنیم که این یعنی مسیر دیگری با طول کمینه فعلی پیدا کرده ایم که در این صورت باید تعداد کوتاه ترین مسیر های s تا u را به v اضافه کنیم. حالا اگر $dist[v] > dist[u] + w(u, v)$ بود، یعنی مسیر کوتاه تری پیدا کرده ایم و اینجا مقداری قبلی $paths[v]$ بی اعتبار می شود چون تعداد مسیر های با طول بیشتر را ذخیره کرده بود . الان که مسیر کوتاه تری داریم، آن را بصورت $paths[v] = paths[u]$ مقدار دهی میکنیم. در نهایت در $paths[i]$ تعداد مسیر های متمایز با کوتاه ترین طول از s به i ذخیره میشود.

۷. دور منفی

الگوریتم‌های بلمن–فورد و فلوید وارشال در صورتی که گراف ورودی شامل یک دور منفی باشد، چگونه رفتار می‌کنند؟

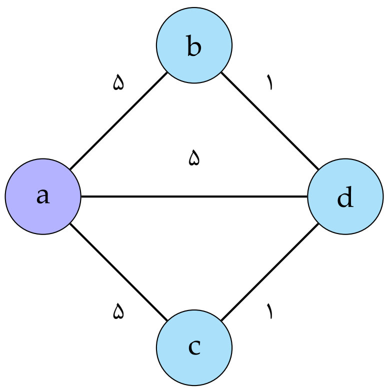
پاسخ :

در الگوریتم بلمن–فورد در مرحله i ام تمام کوتاه ترین مسیر هایی که حداکثر i یال دارند، بدرستی آپدیت و پیدا شده اند. ما فرآیند ریلکس کردن یال ها را $N - 1$ بار انجام میدهیم زیرا طولانی ترین مسیر حداکثر $N - 1$ یال

دارد و اگر یک دور دیگر عملیات ریلکس کردن را انجام دهیم، در صورتی که دور منفی نداشته باشیم، باید ارایه جواب ثابت بماند. در غیر این صورت اگر مقدار یک خانه به مقدار کمتری تغییر کند، یعنی مسیری به طول حداقل N با وزن کمتر یافت شده که چون مسیر ساده به طول N نداریم، این بدین معناست که در این مسیر یک دور طی شده است و از آنجایی که وزن این مسیر کمتر شده است پس یعنی این دور منفی بوده است و در حقیقت هر چند بار که این دور را طی کنیم وزن مسیر ما کمتر و کمتر خواهد شد و بدین شکل آن را شناسایی میکنیم. در الگوریتم فلوید وارشال نیز کافیست در انتهای الگوریتم قطر ماتریس جواب را چک کنیم یعنی $dist[i][i]$. مقدار این خانه ها در ابتدا صفر است و اگر در یک راس در یک دور منفی قرار نداشته باشد، مقدار اولیه صفر آن تغییر نمیکند. اگر یکی از این مقادیر مثلا برای راس i منفی شود، یعنی یک مسیر به شکل $i, k, ..., i$ وجود داشته که یعنی این مسیر تشکیل یک دور منفی داده است.

۸. دایکسترا یا پریم؟

با فرض این که a رأس مبدا است، الگوریتم دایکسترا و همچنین پریم را روی گراف زیر اجرا کنید. آیا الگوریتم دایکسترا درخت کمینه پوشا را پیدا می‌کند؟



پاسخ :

با اجرای الگوریتم دایکسترا در همان دور اول که روی همسایه های a پیمایش میکنیم فاصله 5 برای هر سه راس b، c، d ست میشود و در مراحل بعد تغییری نمیکند. با اجرای الگوریتم پریم ابتدا یکی از یال های متصل به a با وزن 5 انتخاب می شود و سپس دو یال با وزن 1 انتخاب می شوند. که با نتیجه الگوریتم دایکسترا متفاوت است پس جواب به این صورت است که الگوریتم دایکسترا درخت کمینه پوشا را پیدا نمی کند.