

دانشگاه شهید بهشتی کرمان

شش: گراف

ساختمان داده ها و الگوریتم

مدرس: دکتر نجمه منصوری

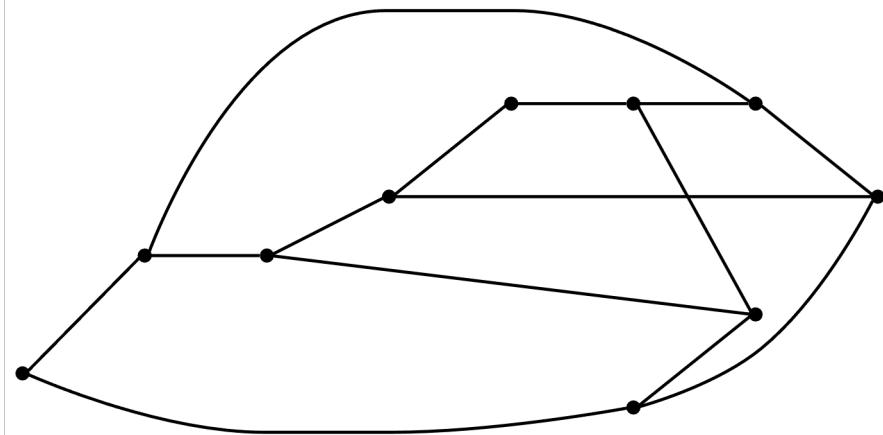
نگارنده: سجاد هاشمیان

گراف

هرگراف $G(V, E)$ شامل دو مجموعه V و E است که :

V مجموعه‌ای متناهی و ناتهی از رئوس می‌باشد.

E مجموعه‌ای متناهی و نه لزوماً ناتهی از یال‌ها می‌باشد.

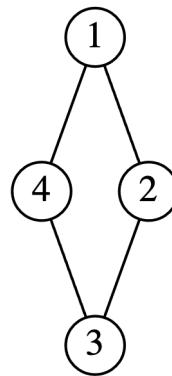


گراف بدون جهت

در یک گراف بدون جهت یال از راس v به راس u همان یال از راس u به راس v است و به دو صورت (u, v) یا (v, u) نمایش داده می‌شود.

$$V(G) = \{1, 2, 3, 4\}$$

$$E(G) = \{(1,2), (2,3), (1,4), (3,4)\}$$



در واقع ترتیب رئوس در بیان یک یال مهم نبوده و زوج رئوس، زوج مرتب نیستند. به عبارت بهتر $(u, v) = (v, u)$

گراف جهت دار (Digraph)

در یک گراف جهت دار هر یال از u به v با زوج مرتب (u,v) نمایش داده می شود.

$$V(G) = \{1, 2, 3\}$$

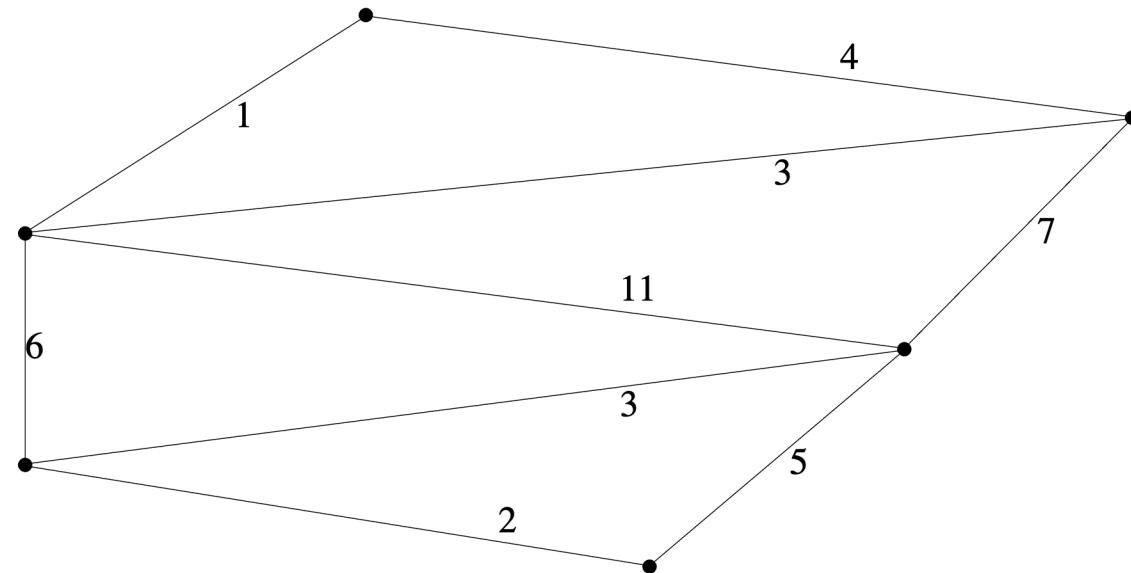
$$E(G) = \{(1,2), (2,1), (2,3)\}$$



بنابراین (v,u) و (u,v) دو یال متفاوت را نمایش می دهند و به عبارت بهتر $(u,v) \neq (v,u)$.

گراف وزن دار

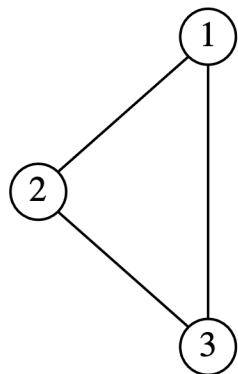
گراف G را برجسب دار یا وزن در می گویند اگر اطلاعاتی به یال های آن نسبت داده شود؛ که به اطلاعات نسبت داده شده به یال ها، وزن یا هزینه یال می گویند. به عبارتی یال های گراف $G(V,E)$ با سه تابی (u,v,w) نشان داده می شوند، که u و v دو سر یال و w وزن یال است.



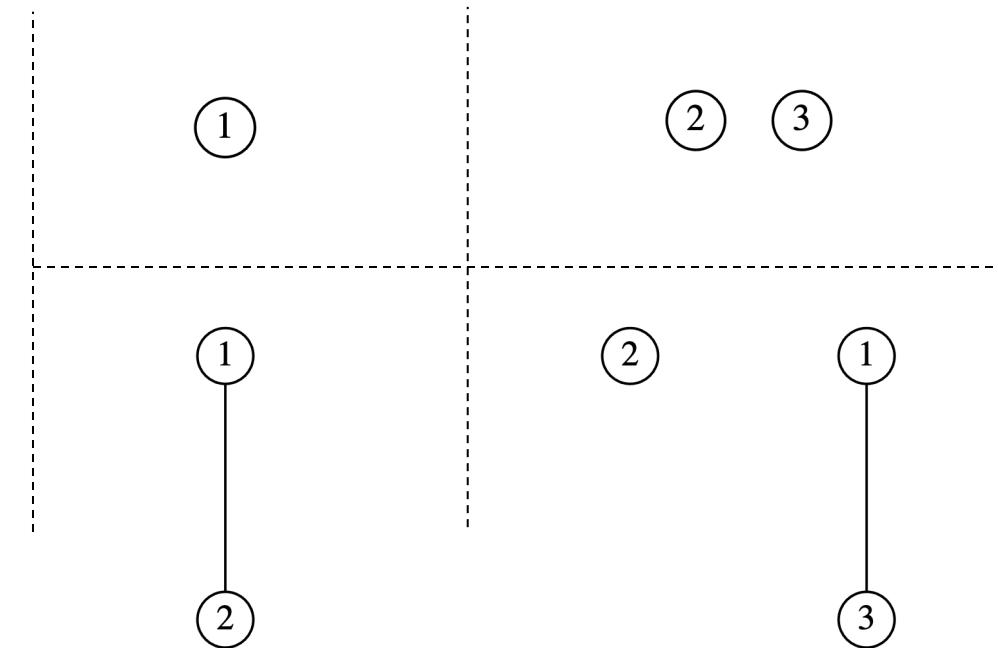
زیرگراف

به گراف $H(V, E)$ زیرگراف $G(V, E)$ گوییم اگر $V(H) \subseteq V(G)$ و $E(H) \subseteq E(G)$ باشد.

: گراف G



: زیرگراف های G

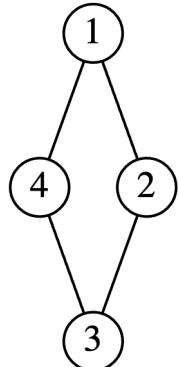


همسایگی یا مجاورت (Adjacency)

دو گره u, v را در یک گراف مجاور (همسایه) گوییم هرگاه یال مستقیمی بین آنها وجود داشته باشد. به عبارتی:

- گراف بدون جهت: دو گره u, v مجاور (همسایه) هستند هرگاه یال (u, v) وجود داشته باشد.
- در گراف جهت دار: هرگاه یال (u, v) وجود داشته باشد یعنی یال مستقیمی از u به v وجود داشته باشد، آنگاه گوییم راس u مجاور است با راس v .

گره ۱ و ۲ مجاور هستند.
گره ۴ و ۳ نیز مجاور هستند.
گره ۳ و ۱ مجاور نیستند.

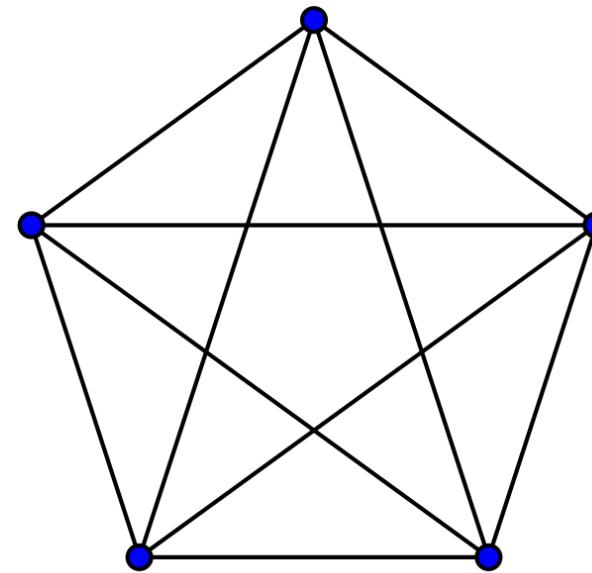
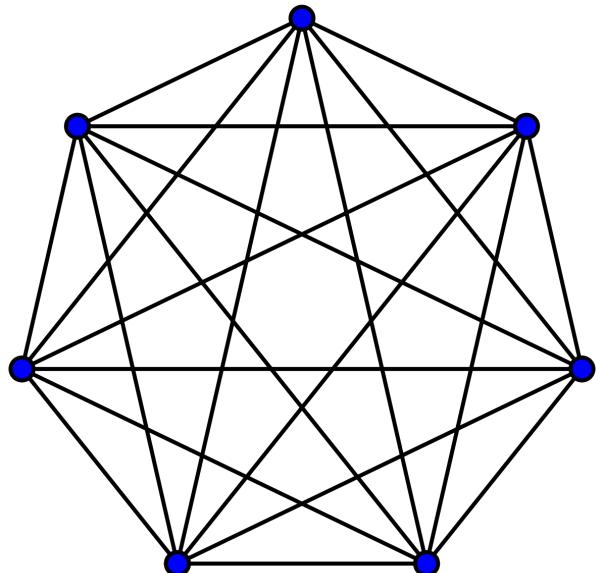


گره ۲ مجاور هر دو راس ۱ و ۳ است.
گره ۳ مجاورت راسی ندارد.



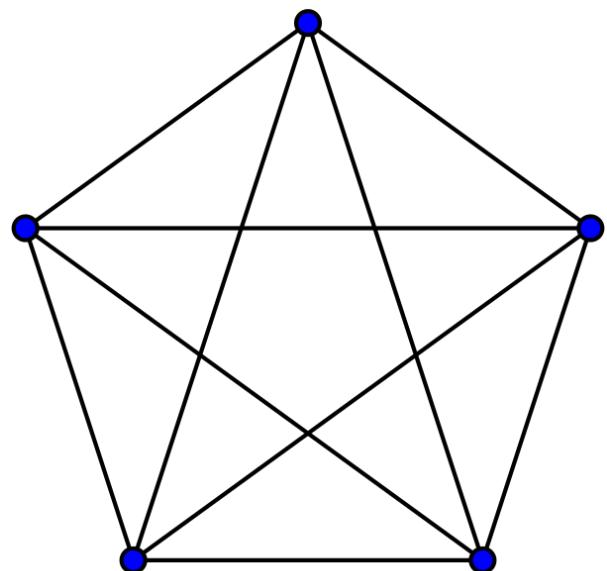
گراف کامل

به گراف G کامل می‌گوییم اگر هر دو راس آن مجاور باشند.



گراف کامل

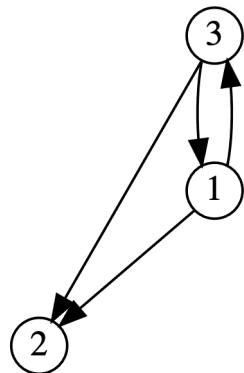
گراف بدون جهت G کامل است اگر و فقط :



- هر دو راس دلخواه، مجاور(همسایه) باشند.
- درجه هر کدام از رؤوس $n-1$ باشد.
- تعداد لبه ها $\frac{n(n-1)}{2}$ باشد.

مسیر (Path)

هر دنباله از رئوس پشت سرهم و متفاوت که دو راس را به هم متصل می‌کنند مسیر نامیده می‌شود؛ تعداد یال‌های موجود در یک مسیر را طول مسیر می‌گویند.

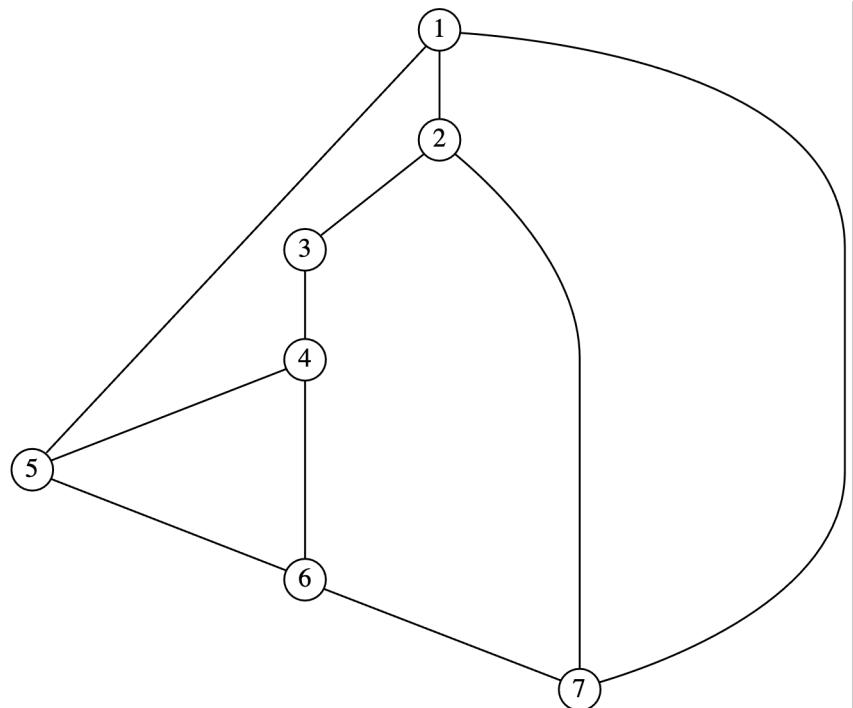


دنباله $\langle 1,2 \rangle$ و $\langle 3,1 \rangle$ مسیرهایی به طول ۱ را نشان می‌دهند.

دنباله $\langle 1,3,2 \rangle$ و $\langle 3,1,2 \rangle$ مسیرهایی به طول ۲ را نشان می‌دهند.

دور (Cycle)

به هر مسیر با راس ابتدایی و انتهایی برابر یک دور می‌گوییم.



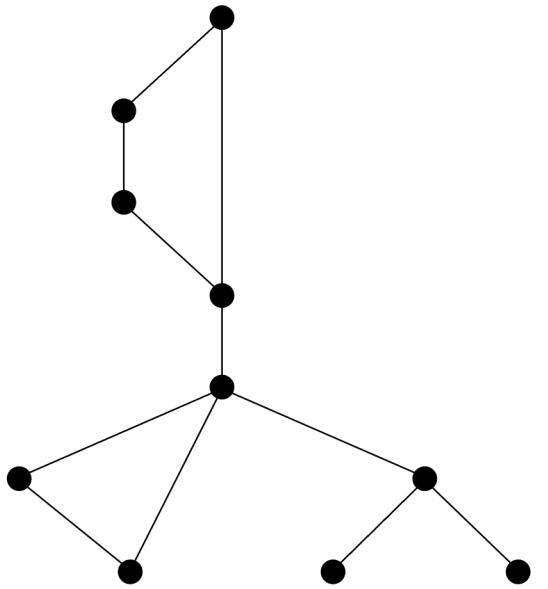
دنباله $\langle 5,4,6,4 \rangle$ و $\langle 1,2,7,1 \rangle$ دورهایی به طول ۳ را نشان می‌دهند.

دنباله $\langle 2,3,4,5,7,3 \rangle$ و $\langle 1,2,3,4,5,1 \rangle$ دورهایی به طول ۵ را نشان می‌دهند.

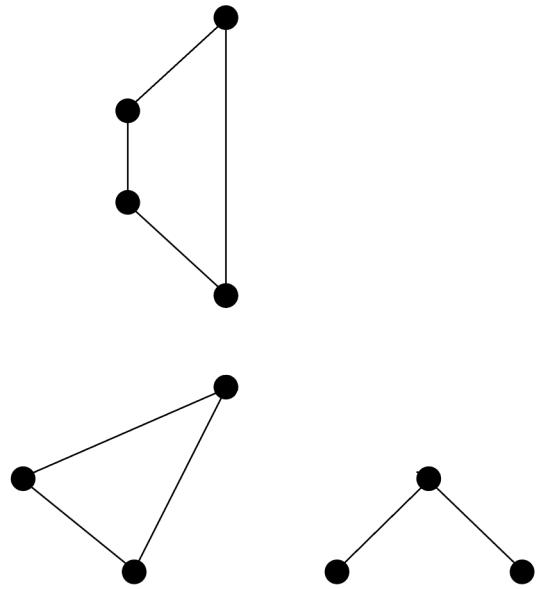
دنباله $\langle 1,2,3,4,5,6,7,1 \rangle$ یک دور شامل همه رئوس و در نتیجه به طول ۶ را نشان می‌دهد.

گراف همبند

گرافی را همبند گوییم هرگاه برای هر دو راس u و v ، حداقل یک مسیر از u به v و بلعکس موجود باشد.



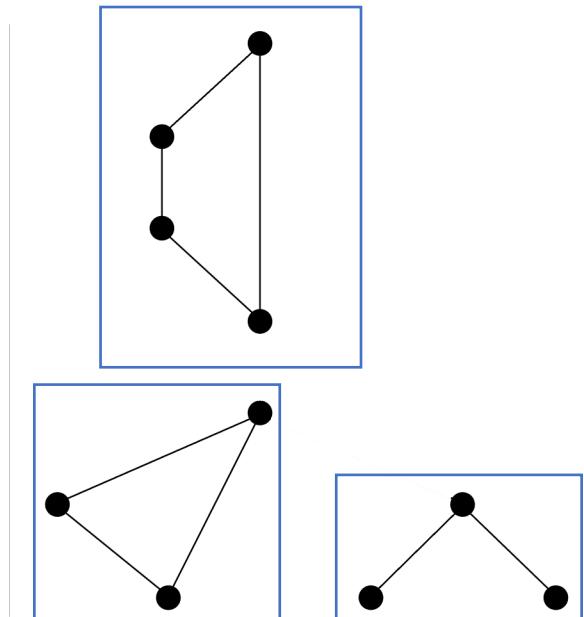
گراف همبند



گراف غیر همبند

مولفه همبندی

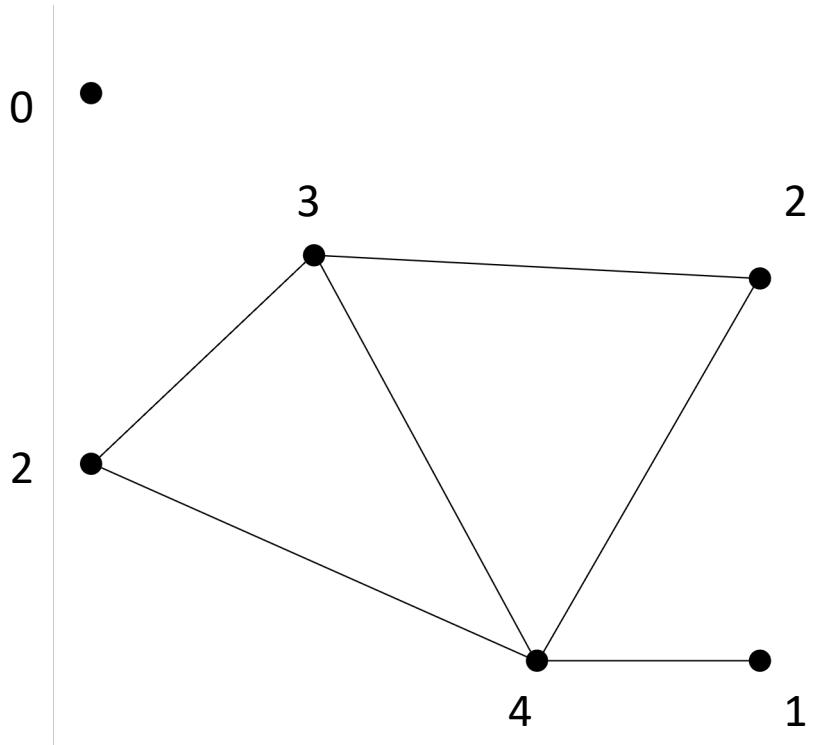
یک زیر گراف مانند H از G یک مؤلفه همبندی برای G است اگر و فقط اگر بین هر دو راس در H دست کم یک مسیر وجود داشته باشد و با افزودن هر راس (و یا یال) دیگری از G به H این خاصیت از بین برود. به عبارت دیگر هر زیر گراف بیشینه و همبند از G یک مؤلفه همبندی G است.



یک گراف با ۳ مولفه همبندی

درجه رئوس

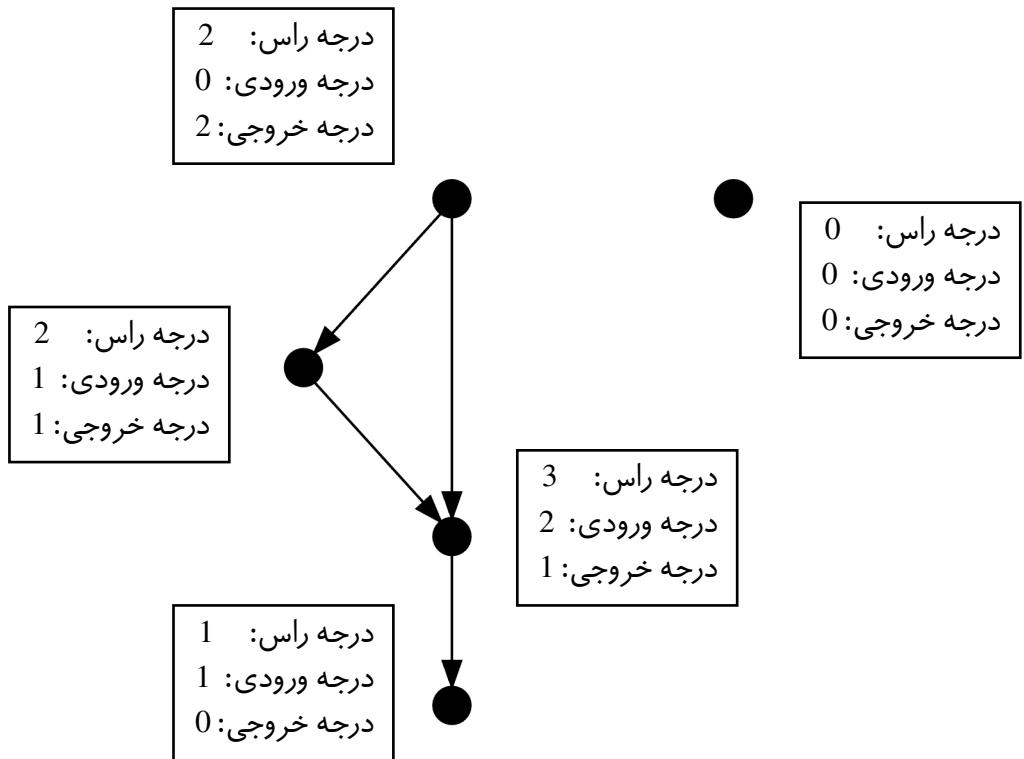
درجه یک راس از گراف بدون جهت تعداد یال‌های متقابی با آن راس است. به عبارت بهتر درجه هر راس از گراف تعداد راس‌های مجاور به آن را نشان می‌دهد.



درجہ رئوس

در هر گراف جهت دار درجه یک راس برابر است با (درجه ورودی + درجه خروجی) که:

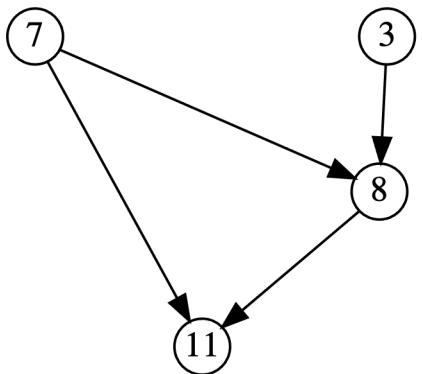
- درجه ورودی: تعداد یال هایی که به آن راس وارد می شوند.
- درجه خروجی: تعداد یال هایی که از آن راس خارج می شوند.



تعداد یال ها

قضیه. در هر گراف با n راس و e یال، اگر $d(v)$ درجه راس v باشد، آنگاه تعداد یال های گراف برابر است با:

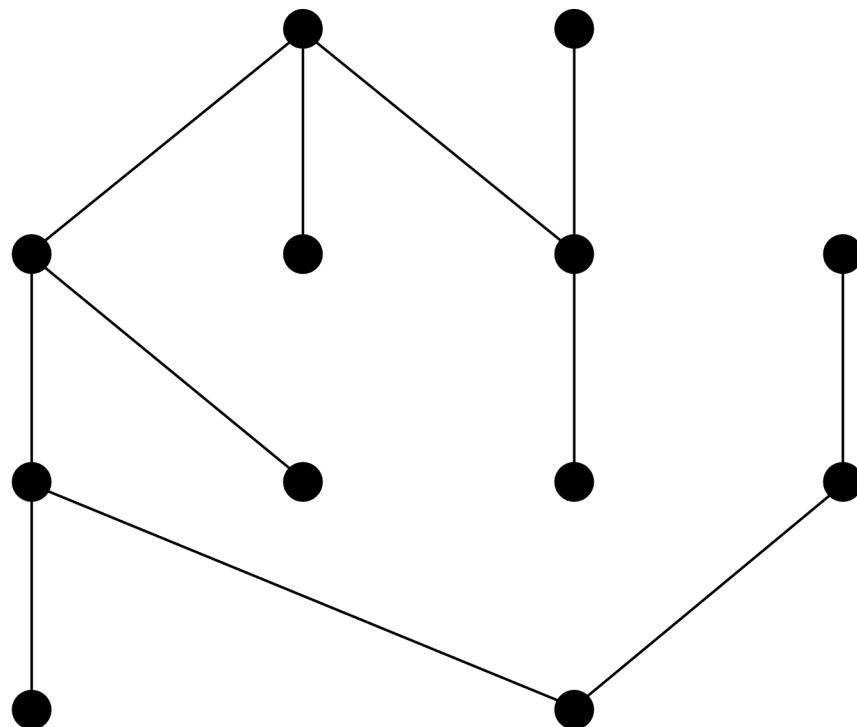
$$e = \frac{\sum d(v)}{2} \implies 2e = \sum_{v \in V(G)} d(v)$$



$$e = \frac{1 + 3 + 2 + 2}{2} = 4$$

درخت

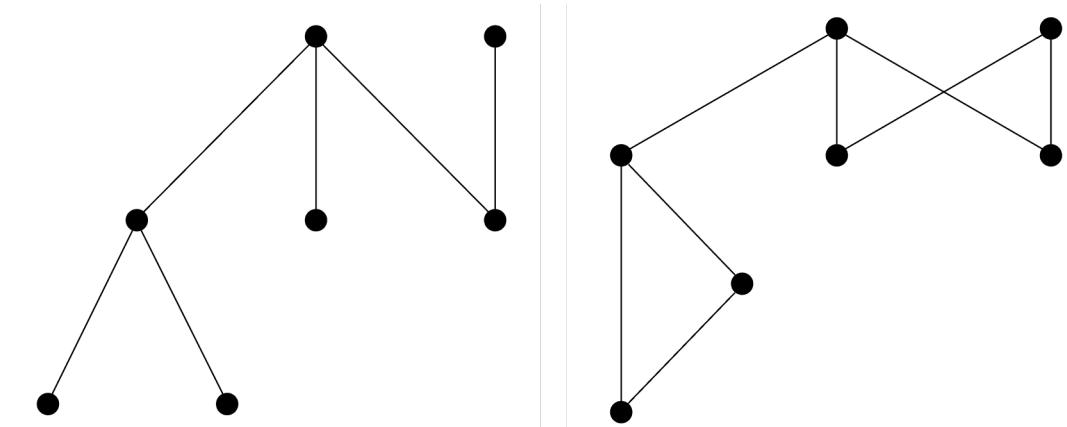
گراف بدون جهت T را یک درخت گوییم اگر هیچ دوری نداشته باشد.



درخت

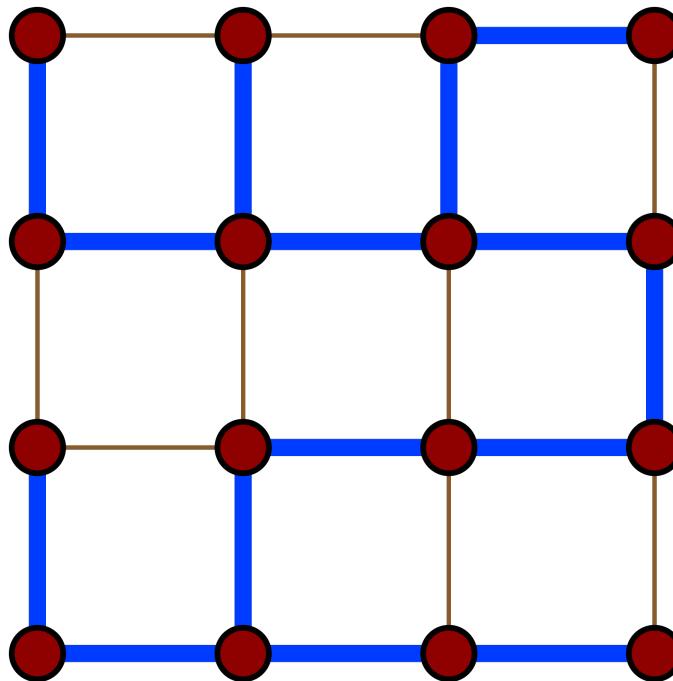
قضیه. برای گراف بدون جهت T با n راس، موارد زیر یکسان و معادل هستند:

- T یک درخت می‌باشد.
- T گرافی همبند است، اما اگر هر یک از یال‌ها حذف گردد، گراف حاصل همبند نمی‌باشد.
- برای هر راس مجزا مانند u و v از T تنها یک مسیر ساده از u به v وجود دارد.
- T فاقد دور بوده و دارای $n-1$ یال می‌باشد.



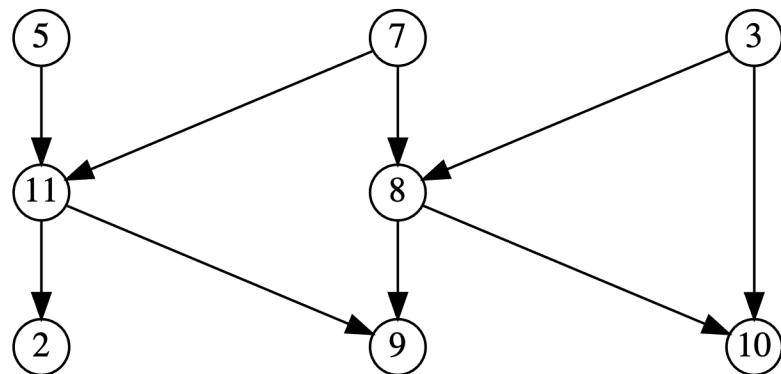
درخت پوشای

یک درخت پوشای T از گراف همبند و بدون جهت G ، درختی است که شامل تمام رئوس و مجموعه‌ای از یال‌ها می‌باشد.



گراف جهت دار بدون دور (DAG)

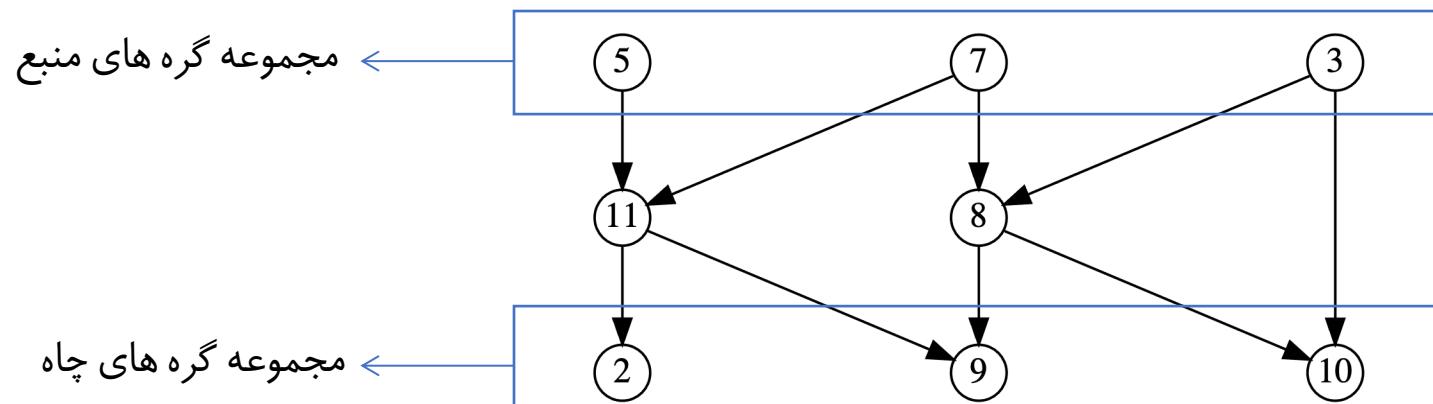
گراف جهت دار غیر مدور (Directed Acyclic Graph) یک گراف جهت دار است که هیچ دوری ندارد.



گره های چاه و منبع

راس منبع (source) : به راسی که درجه ورودی آن ۰ باشد اطلاق می شود.

راس چاه (sink) : به راسی که درجه خروجی آن ۰ باشد گفته می شود.

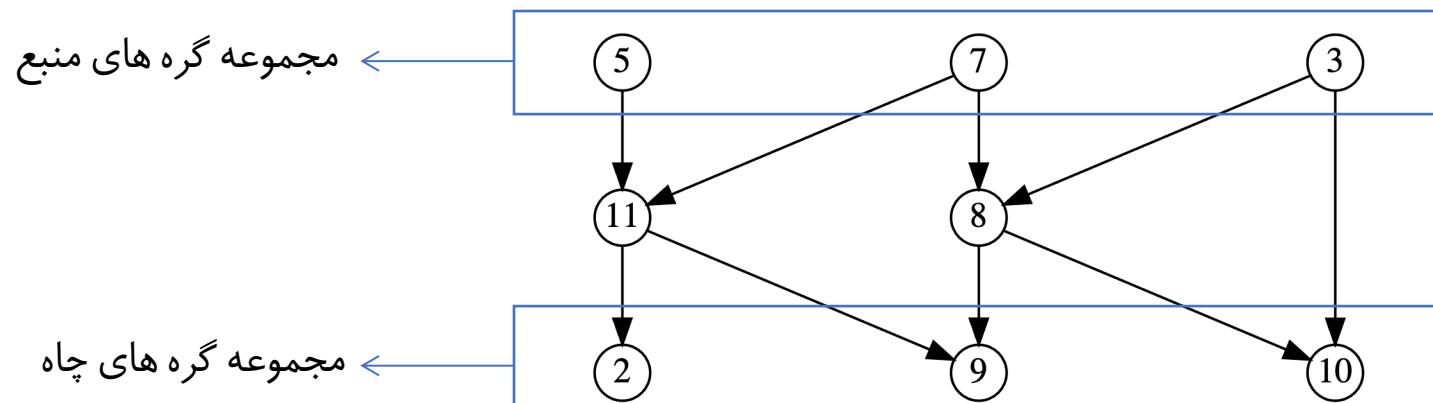


گره های چاه و منبع

راس منبع (source): به راسی که درجه ورودی آن 0 باشد اطلاق می‌شود.

راس چاه (sink): به راسی که درجه خروجی آن 0 باشد گفته می‌شود.

سوال) در چه صورتی لزوماً این گره ها وجود دارند؟



نمایش گراف

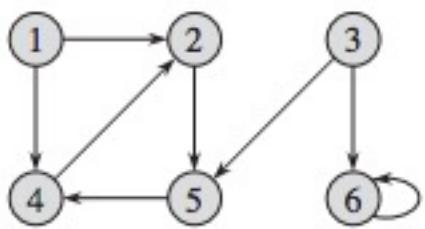
نمایش گراف در واقع همان روش‌های ذخیره‌سازی گراف در کامپیوتر است، به عبارت دیگر با توجه به محدودیت‌هایی که در کامپیوتر داریم، نمی‌توانیم شکل گراف را همان‌طور که در کاغذ می‌کشیم نشان دهیم، یا با گفتن ویژگی‌های تصویری آن را به راحتی ذخیره و شبیه سازی کنیم. از این‌رو روش‌های مختلفی برای اینکار بوجود آمده است.

شماره‌گذاری

اولین کاری که برای ذخیره سازی گراف نیاز داریم، شماره‌گذاری رئوس است. یعنی به هر راس یک شماره نسبت دهیم تا بتوانیم بین آن‌ها تمایز قائل شویم. از این‌رو گراف‌های یک شکل، نمایشی متفاوت دارند. چون آرایه‌ها و آدرس‌ها در کامپیوتر از صفر شروع می‌شوند معمولاً شماره‌گذاری را از صفر آغاز می‌کنند.

ماتریس مجاوٽ

```
for e in E(G):
    adj[e.u][e.v] = e.w
    adj[e.v][e.u] = e.w
```



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

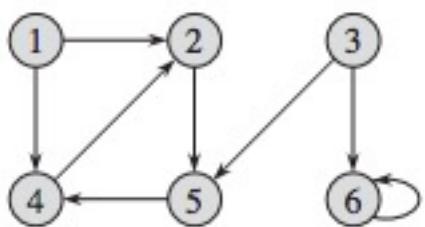
در واقع یک جدول دو بعدی از درایه‌ها است که طول سطر و ستون آن برابر تعداد راس‌های گراف است.

ابتدا راس‌ها را شماره گذاری می‌کنیم. حال در درایه‌ی سطر i ام و ستون j ام آن اگر از راس شماره i به j یال نبود، صفر می‌گذاریم؛ اگر یال بود وزن آن را و اگر گراف وزن‌دار نبود، ۱ می‌گذاریم.

همچنین اگر گراف بدون جهت باشد، این را برای قرینه آن هم انجام می‌دهیم یعنی این‌بار همین کار را از سطر ز به ستون i انجام می‌دهیم.

ماتریس مجاوٽ

```
for e in E(G):
    adj[e.u][e.v] = e.w
    adj[e.v][e.u] = e.w
```



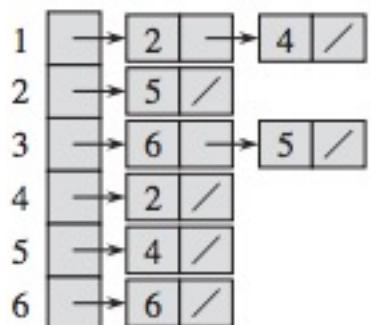
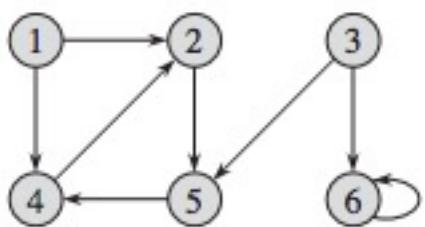
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

در این شیوه ذخیره سازی چون تعداد سطرها و ستون‌ها برابر تعداد رئوس است پس به فضای حافظه‌ای از $O(n^2)$ نیاز داریم که n تعداد رأس‌ها است.

اگرچه ممکن است کمی زیاد بنظر بیاید، اما خوبی این شیوه در این است که با $O(1)$ می‌توان از وزن یال بین دو راس در صورت وجود اطلاع یافت که در شیوه‌های دیگر این ویژگی را نداریم.

لیست مجاورت

```
for e in E(G):
    adj[e.u].append([e.v, w])
    adj[e.v].append([e.u, w])
```



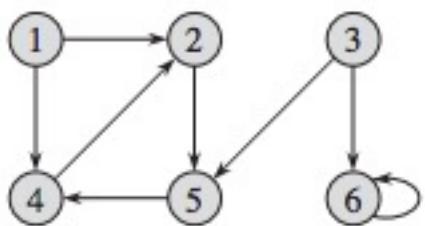
لیست مجاورت، در واقع لیستی است که به ازای هر راس، لیستی از مجموعه یال‌های خروجی در گراف‌های جهت‌دار را نگه می‌داریم.

پس فضای حافظه‌ی ما به تعداد یال‌ها وابسته است؛ با کمی تفکر می‌توان دریافت که فضای مصرفی در گراف‌های بی‌جهت و جهت‌دار به ترتیب دوباره و برابر تعداد یال‌هاست.

از آنجایی که برای هر راس تعداد یال‌های متفاوتی را نگه می‌داریم، می‌توانیم به ازای هر راس یک لیست پیوندی بگیریم.

لیست مجاورت

```
for e in E(G):
    adj[e.u].append([e.v, w])
    adj[e.v].append([e.u, w])
```



1	→	2	→	4	/
2	→	5	/		
3	→	6	→	5	/
4	→	2	/		
5	→	4	/		
6	→	6	/		

با توجه به مطالب گفته شده، مرتبه حافظه‌ای مورد نظر برای اینکار از $O(n+e)$ است که برای گراف‌های تنک، فضای بھینه و کمی است.

اما برای گراف‌های شلوغ، ماتریس مجاورت بهتر است، چراکه در این حالت با وجود فضای مصرفی‌ای به اندازه ماتریس مجاورت، همچنان برای فهمیدن وجود یال بین دو راس v و u در بدترین حالت باید کل لیست را بگردیم که یعنی از $O(n)$ زمان مصرف می‌کنیم.

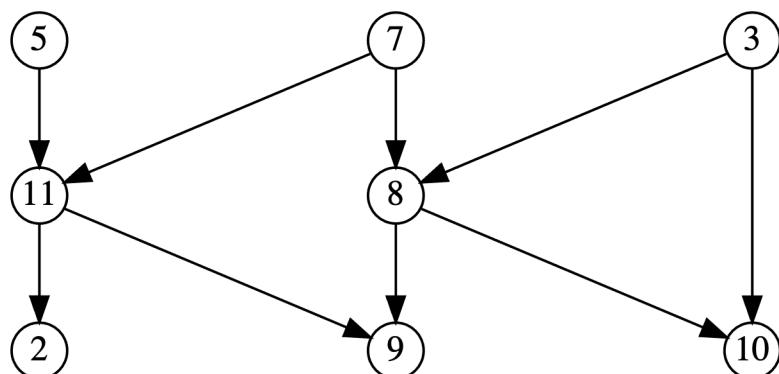
مسئله. مرتب سازی توپولوژی

مرتب سازی توپولوژیک، مرتب سازی رئوس یک گراف جهت دار بدون طوقه و بدون دور است به طوری که هر راس قبل از رئوسی می‌اید که به آنها یال خروجی داده است.

مرتب سازی قابل قبول 5, 7, 3, 11, 2, 8, 9, 10

مرتب سازی قابل قبول 5, 7, 3, 11, 8, 2, 9, 10

مرتب سازی غیر قابل قبول 5, 7, 3, 11, 2, **9**, 8, 10



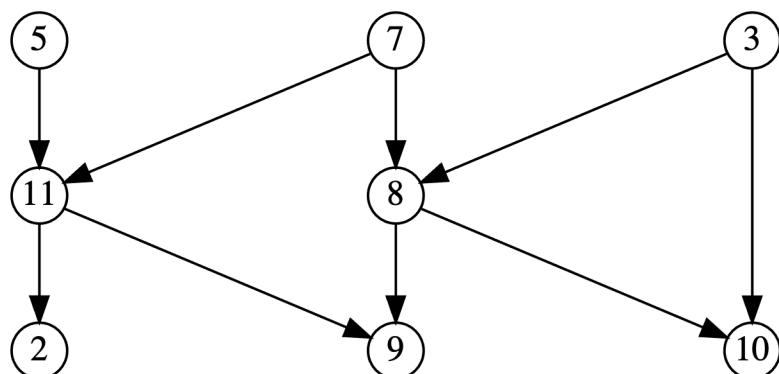
مسئله. مرتب سازی توپولوژی

مرتب سازی توپولوژیک، مرتب سازی رئوس یک گراف جهت دار بدون طوقه و بدون دور است به طوری که هر راس قبل از رئوسی می‌اید که به آنها یال خروجی داده است.

مرتب سازی قابل قبول 5, 7, 3, 11, 2, 8, 9, 10

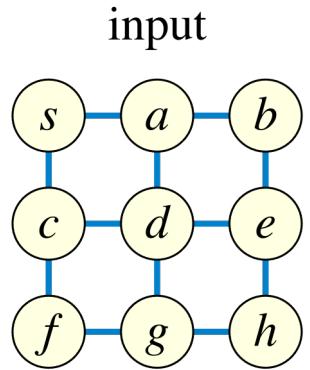
مرتب سازی قابل قبول 5, 7, 3, 11, 8, 2, 9, 10

مرتب سازی غیر قابل قبول 5, 7, 3, 11, 2, 9, 8, 10

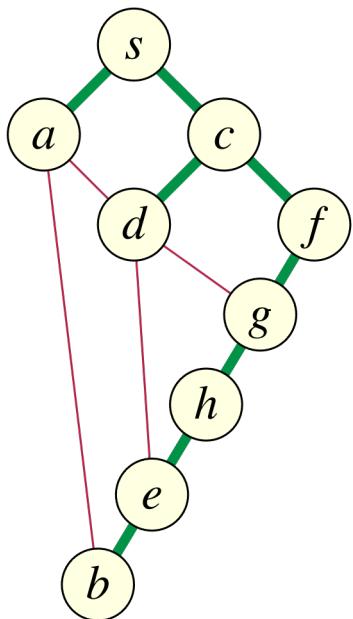


پاسخ: پیمایش سطح اول!

پیمایش گراف‌ها



stack traversal



پیمایش گراف (Graph traversal) در حالت کلی به معنی گذشتن و دیدن راس‌های گراف است و معمولاً پیمایش از طریق یال‌های موجود در گراف انجام می‌شود.

معمولاً پیمایش گراف به تنها بی ارزش خاصی ندارد و هدف از پیمایش، محاسبه یا پیدا کردن چیز دیگری است.

برای مثال شاید با نگاه کردن به یک گراف بتوانید خواص آن را پیدا کنید و مسئله را حل کنید؛ اما در کامپیوتر به دلیل محدودیت‌های همین راحتی نمی‌توان این کار را کرد. الگوریتم‌های پیمایش در حرکت روی گراف و پیدا کردن خواص آن به ما کمک می‌کنند.

پیمایش گراف‌ها

انواع پیمایش:

- پیمایش سطح اول (BFS): به کمک ساختار صفحه‌نشانی
- پیمایش عمق اول (DFS): به کمک ساختار پشته

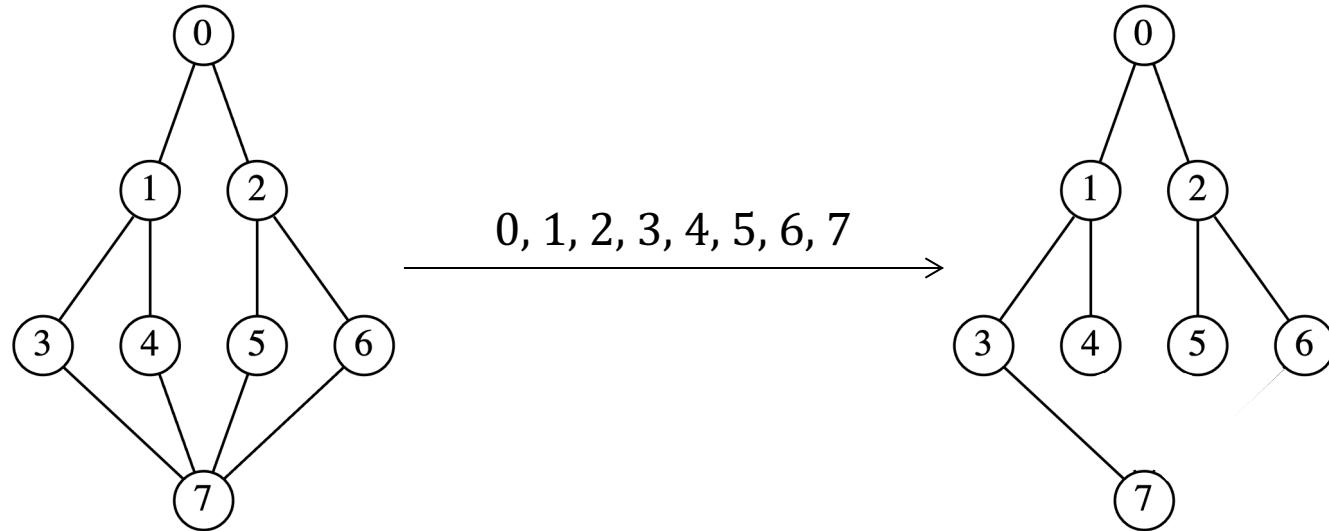
مرتبه اجرایی:

- اگر گراف G توسط ماتریس مجاورت ارائه شود، آن‌گاه زمان لازم برای پیمایش $O(n^2)$ است.
- اگر گراف G توسط لیست مجاورت ارائه شود، آن‌گاه زمان لازم برای پیمایش $O(n+e)$ است.
- از آنجا که e حداقل از $O(n^2)$ است (گراف کامل) پس در بدترین حالت زمان اجرایی برابر با $O(n^2)$ است.

جستجو سطح اول

Breath First Search

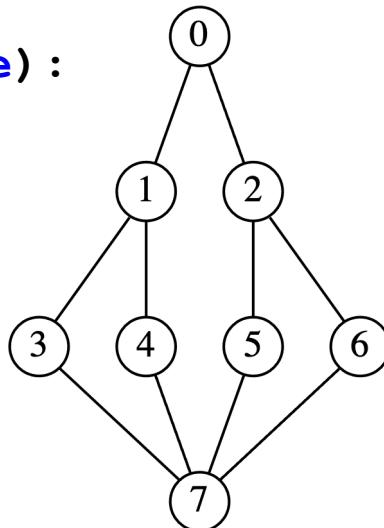
۱. راس شروع را انتخاب کرده و به **صف ملاقات اضافه** می‌کنیم (بیش فرض راس ۰).
۲. تا زمانی که **صف ملاقات خالی** نشده:
۳. برای هر راس ملاقات شده، تمام رئوس همسایه آن را که تا به حال ملاقات نشده‌اند را، به **صف ملاقات اضافه** می‌کنیم.



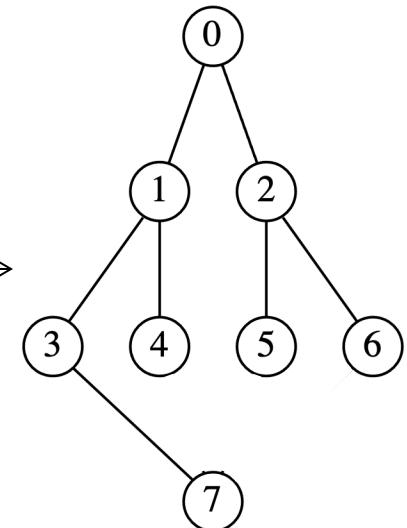
Breath First Search

جستجو سطح اول

```
def BFS(v) :  
    Q = empty Queue  
    Q.push(v)  
    visited[v]=True  
    while(len(Q)!=0):  
        v=Q.pop()  
        for u in adj[v]:  
            if(visited[u]==False):  
                Q.push(u)  
                visited[u]=True  
    return
```

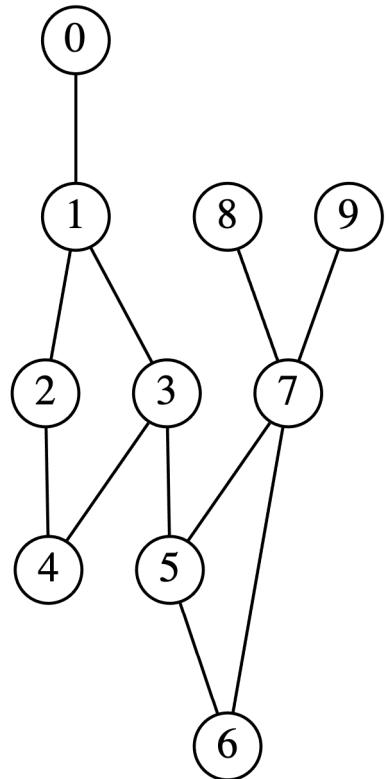


0, 1, 2, 3, 4, 5, 6, 7



مثال

ترتیب ملاقات گره‌ها در پیمایش سطح اول گراف رو برو از چپ به راست کدام است؟



0, 1, 3, 2, 4, 5, 6, 7, 8, 9 (۱)

0, 1, 3, 2, 4, 5, 7, 6, 8, 9 (۲)

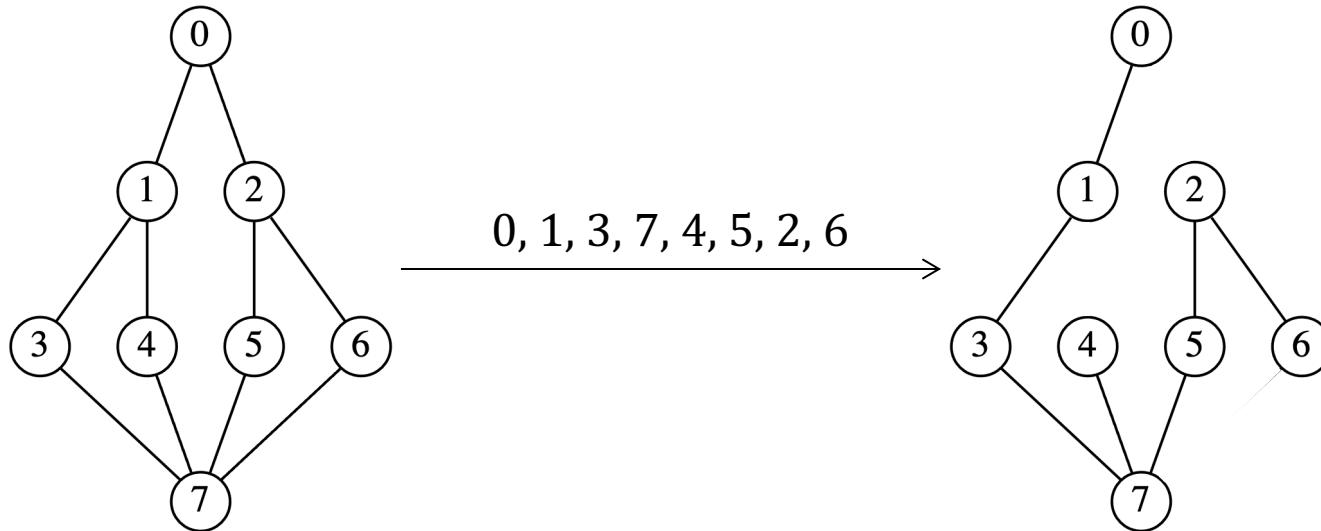
0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (۳)

0, 1, 3, 2, 4, 5, 7, 6, 9, 8 (۴)

جستجو عمق اول

Depth First Search

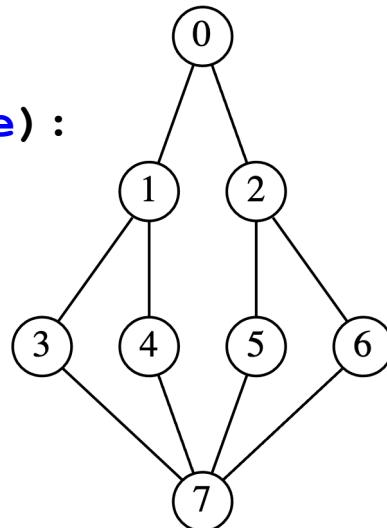
۱. راس شروع را انتخاب کرده و به پیشته ملاقات اضافه می‌کنیم (پیش فرض راس ۰).
۲. تا زمانی که پیشته ملاقات خالی نشده:
۳. برای هر راس ملاقات شده، تمام رئوس همسایه آن را که تا به حال ملاقات نشده‌اند را، به پیشته ملاقات اضافه می‌کنیم.



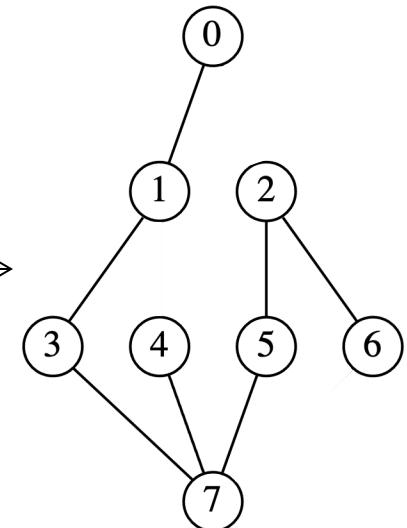
Depth First Search

جستجو عميق اول

```
def DFS(v):
    S = empty Stack ←
    S.push(v)
    visited[v]=True
    while(len(S)!=0):
        v=S.pop()
        → visited[u]=True
        for u in adj[v]:
            if(visited[u]==False):
                S.push(u)
    return
```

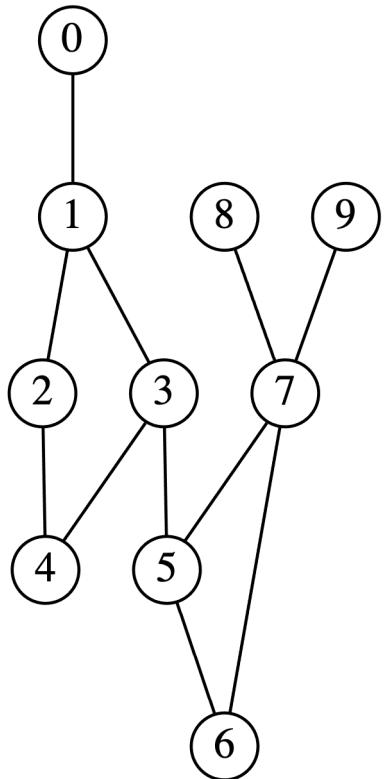


0, 1, 3, 7, 4, 5, 2, 6



مثال

ترتیب ملاقات گره‌ها در پیمایش عمق اول گراف رو برو از چپ به راست کدام است؟



۰, ۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸, ۹ (۱)

۰, ۱, ۲, ۴, ۳, ۵, ۷, ۶, ۸, ۹ (۲)

۰, ۱, ۲, ۳, ۴, ۵, ۶, ۷, ۹, ۸ (۳)

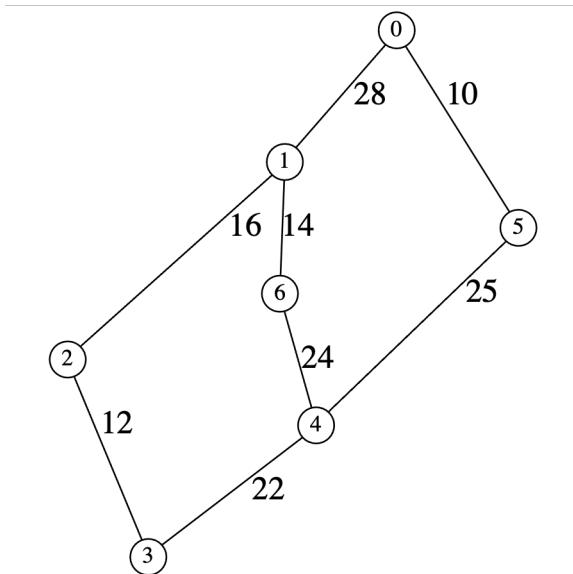
۰, ۱, ۳, ۲, ۴, ۵, ۷, ۶, ۸, ۹ (۴)

مسئله: درخت پوشای کمینه

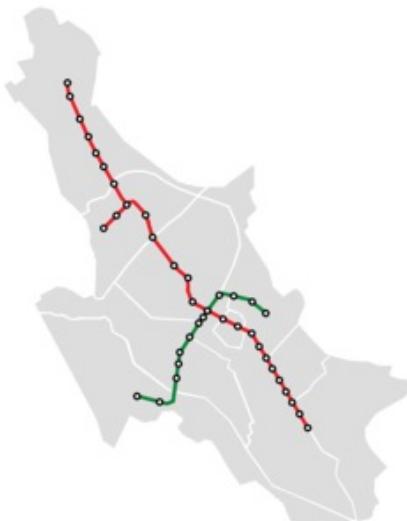
(MST : Minimum Spanning Tree)

پیماش‌های هر گراف وزن دار منجر به تولید درختان پوشای $n-1$ یال و با هزینه‌های متفاوت می‌شود؛ بنابراین مسئله اینگونه ایجاد می‌شود:

سوال. فرض کنید به ما یک گراف وزن دار داده شده و از ما خواسته شده زیردرخت فراگیری از آن را پیدا کنیم که مجموع وزن یال‌های آن کمینه باشد.



مسئله: درخت پوشای مینیموم (MST : Minimum Spanning Tree)



به عنوان مثال فرض کنید یک شبکه راه آهن که تعدادی شهر را به یکدیگر متصل می‌کند در دست احداث است می‌خواهیم با داشتن هزینه C_{ij} مربوط به احداث خط مستقیم بین شهرهای v_i و v_j شبکه را طوری طراحی کنیم که مجموع هزینه‌های ساخت به کمترین مقدار خود برسد.

با در نظر گرفتن هر شهر به عنوان یک راس از گراف وزن دار با وزن‌های $w(v_i, v_j) = C_{ij}$ مسئله به یافتن یک زیر گراف فراگیر همبند با کمترین وزن در یک گراف منجر می‌شود.

الگوریتم کراسکال (Kruskal)

در این روش، درخت پوشای کمینه T ، یال به یال ساخته می‌شود تا دقیقاً $n-1$ یال برای آن انتخاب شود:

- .1 تمام یال‌ها را به ترتیب صعودی وزن‌شان مرتب کن.
- .2 کمینه یال بررسی نشده، یعنی یال (u,v) را انتخاب کن.
- .3 اگر اضافه کردن آن به T دور ایجاد نمی‌کند، آن به T اضافه کن.
- .4 اگر هنوز $n-1$ یال انتخاب نشده به شماره ۳ برو.
- .5 پایان.

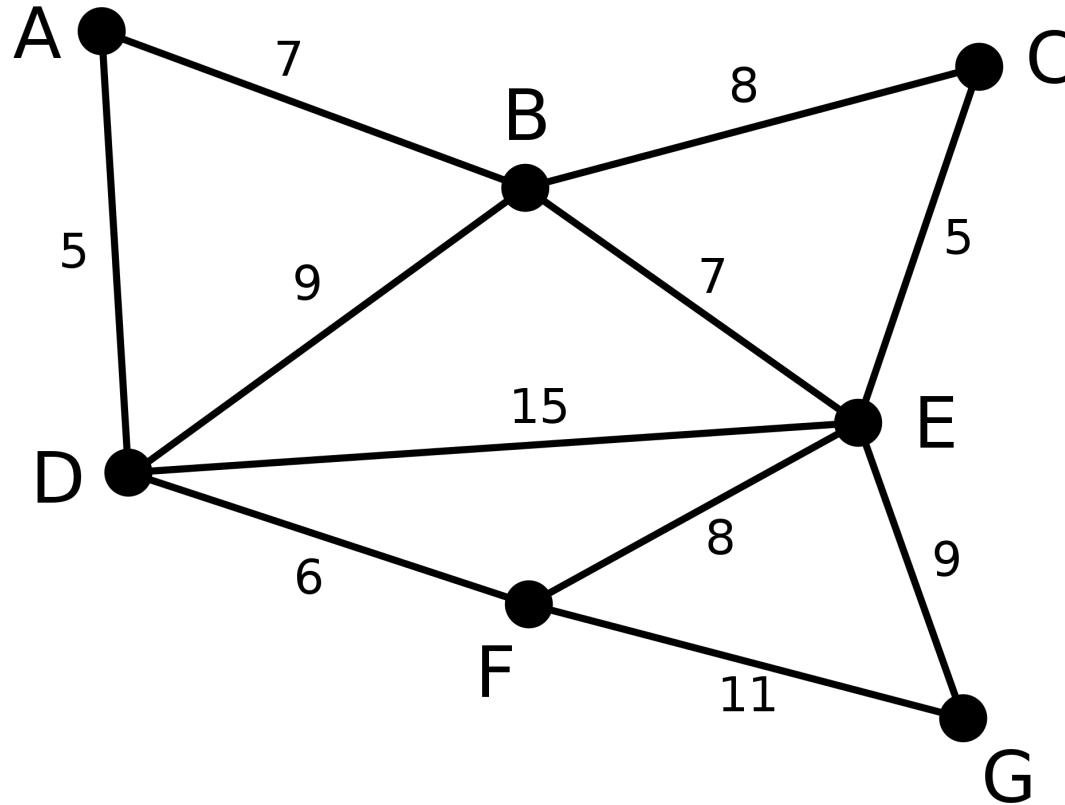
الگوریتم کراسکال (Kruskal)

در این روش، درخت پوشای کمینه T ، یال به یال ساخته می‌شود تا دقیقاً $n-1$ یال برای آن انتخاب شود:

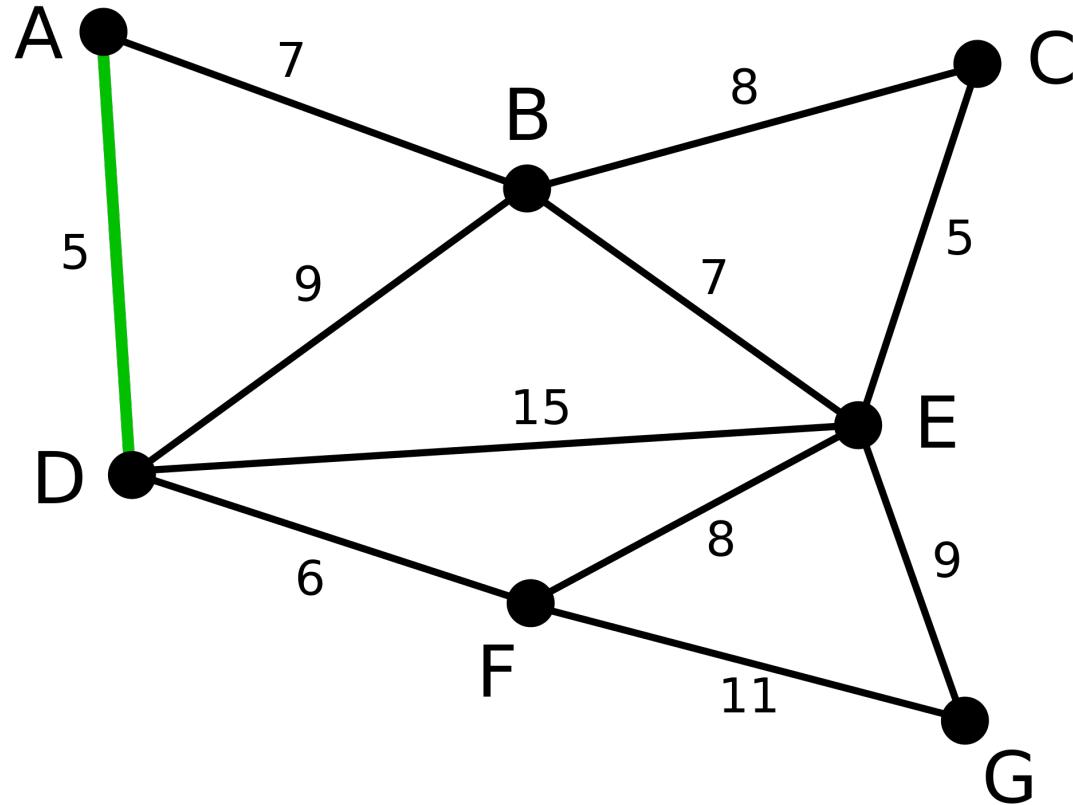
- .1 تمام یال‌ها را به ترتیب صعودی وزن‌شان مرتب کن.
- .2 کمینه یال بررسی نشده، یعنی یال (u,v) را انتخاب کن.
- .3 اگر اضافه کردن آن به T دور ایجاد نمی‌کند، آن به T اضافه کن.
- .4 اگر هنوز $n-1$ یال انتخاب نشده به شماره ۳ برو.
- .5 پایان.

صحت درستی جواب این الگوریتم؟
مرتبه اجرایی آن؟

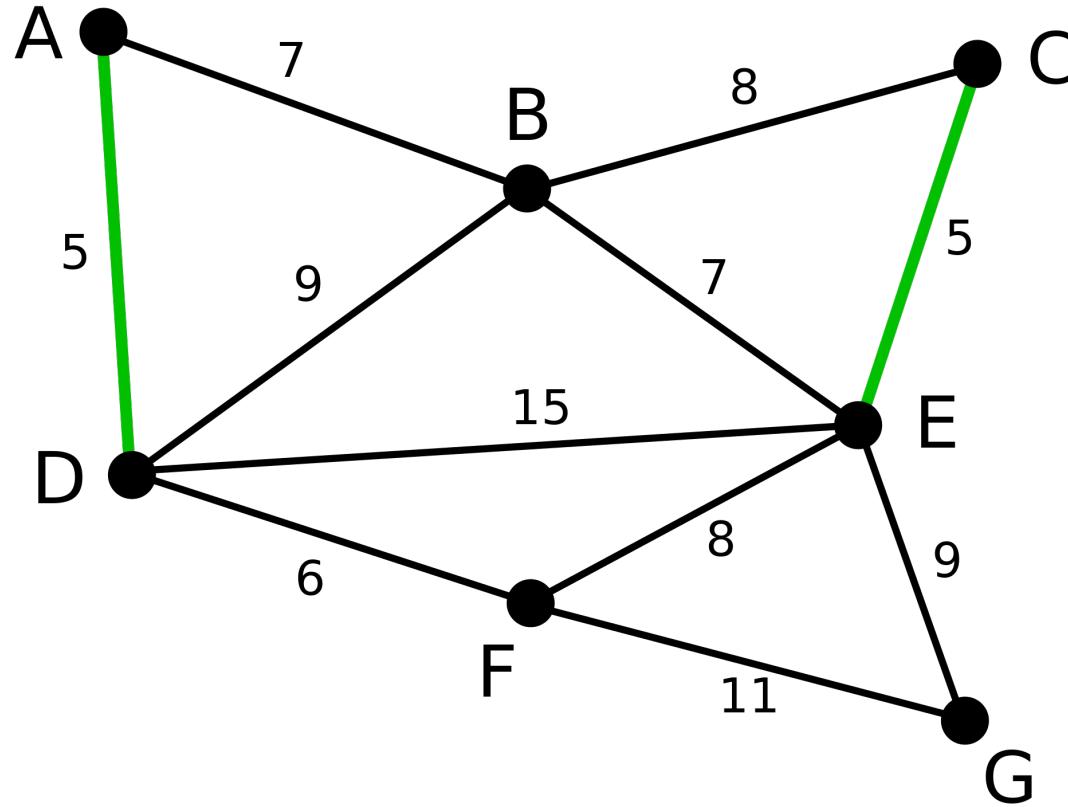
الگوريتم کراسکال (Kruskal)



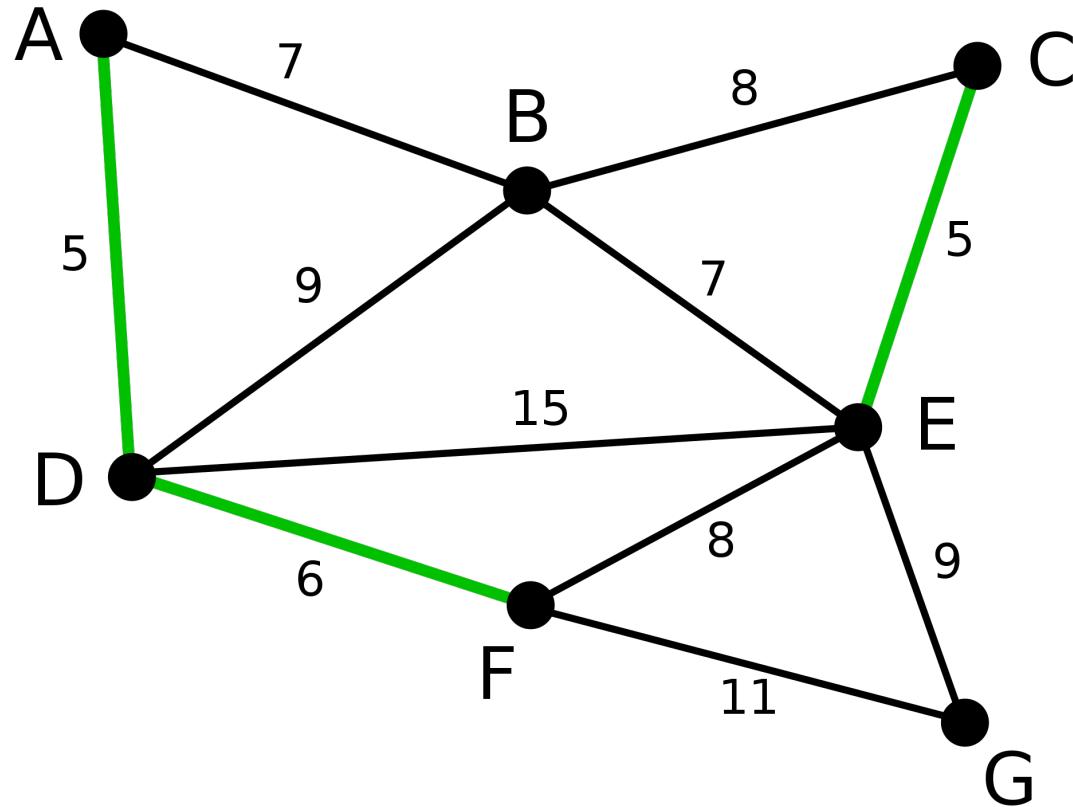
الگوريتم کراسکال (Kruskal)



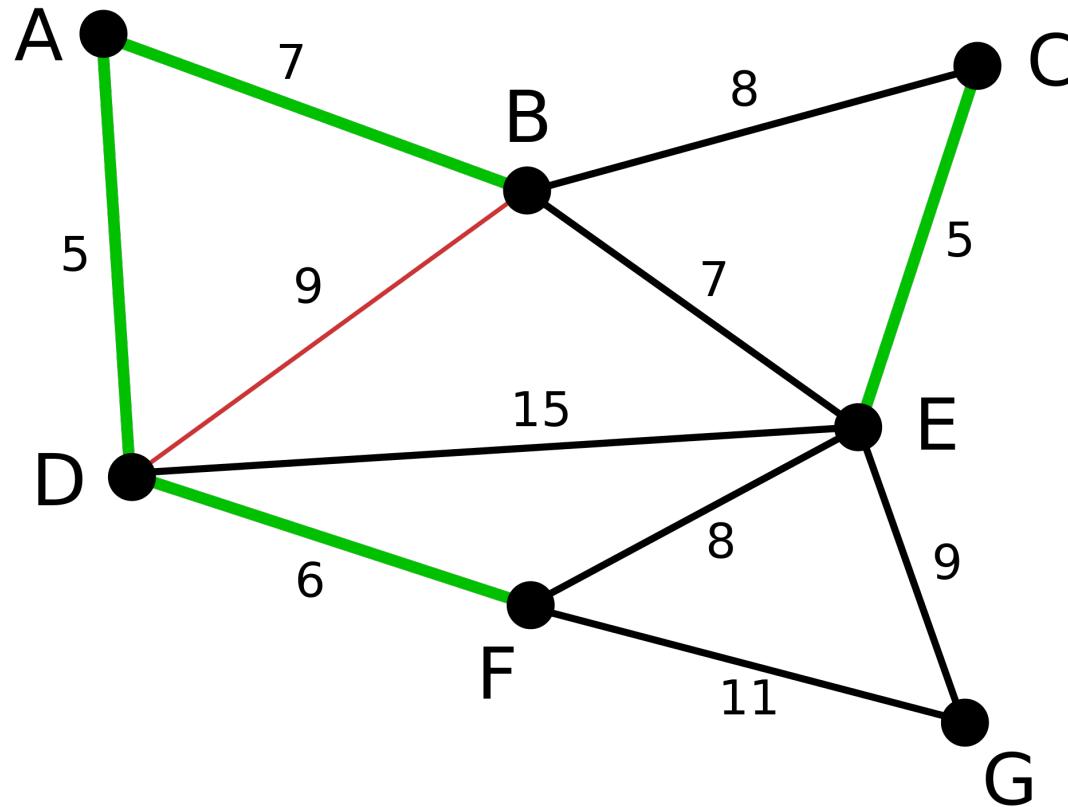
الگوریتم کراسکال (Kruskal)



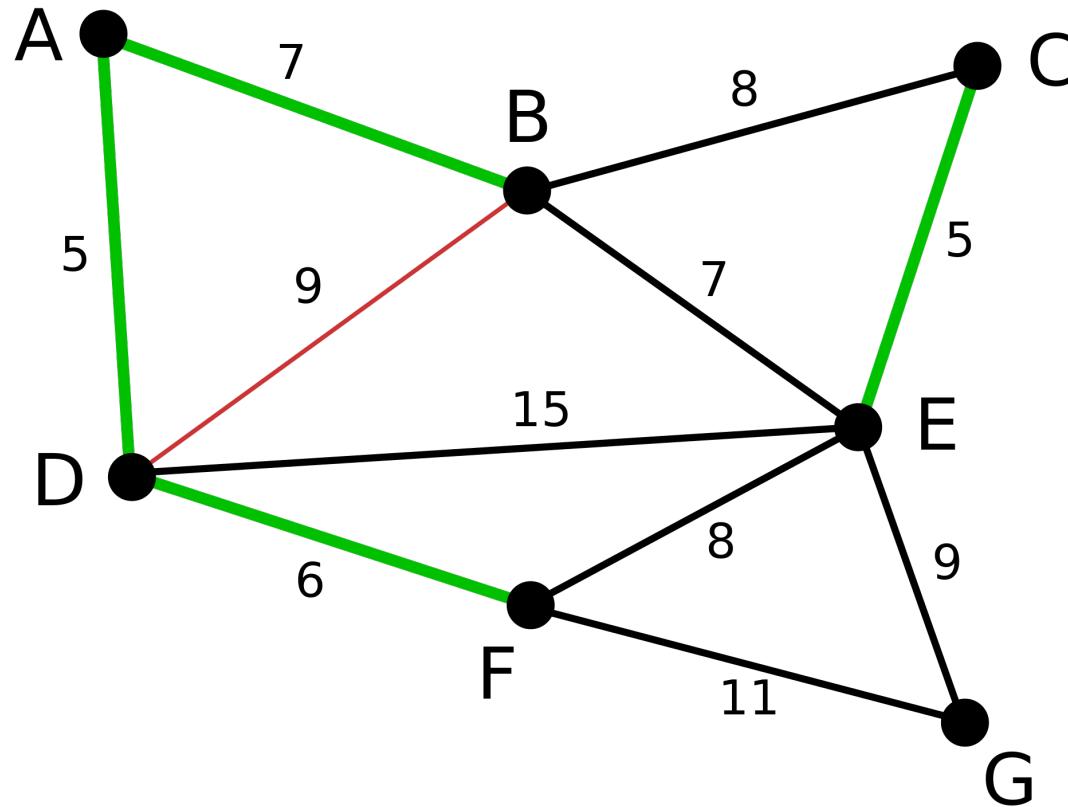
الگوريتم کراسکال (Kruskal)



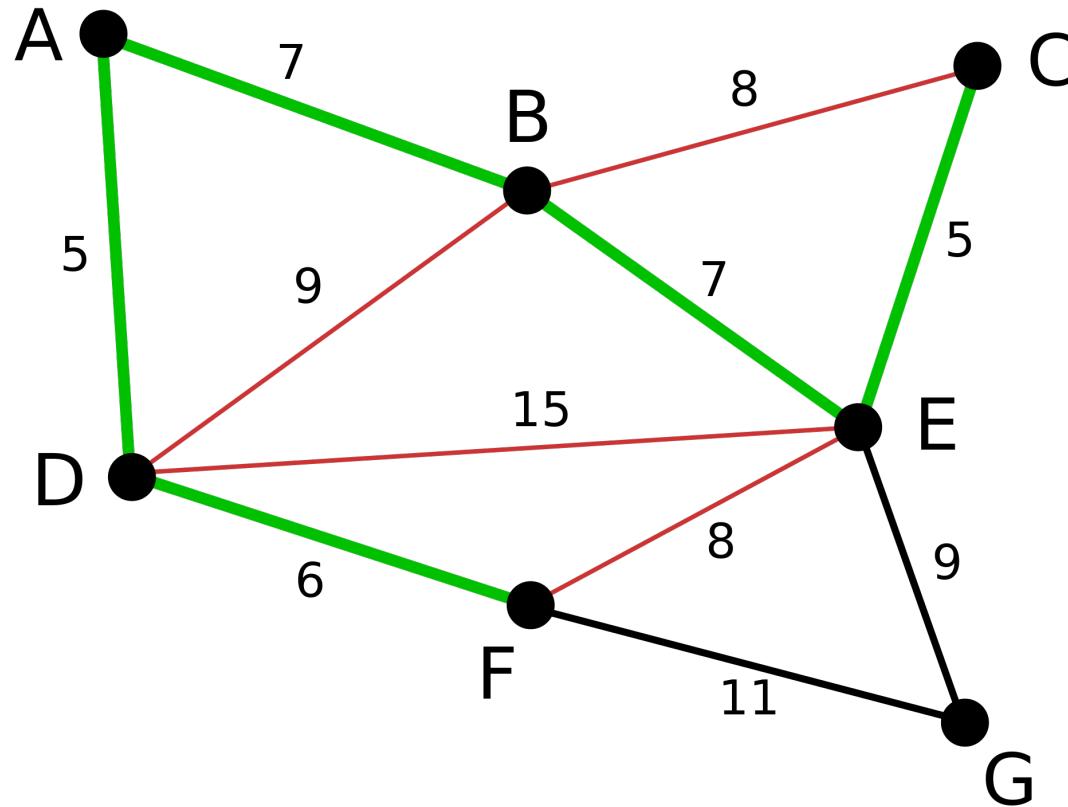
الگوریتم کراسکال (Kruskal)



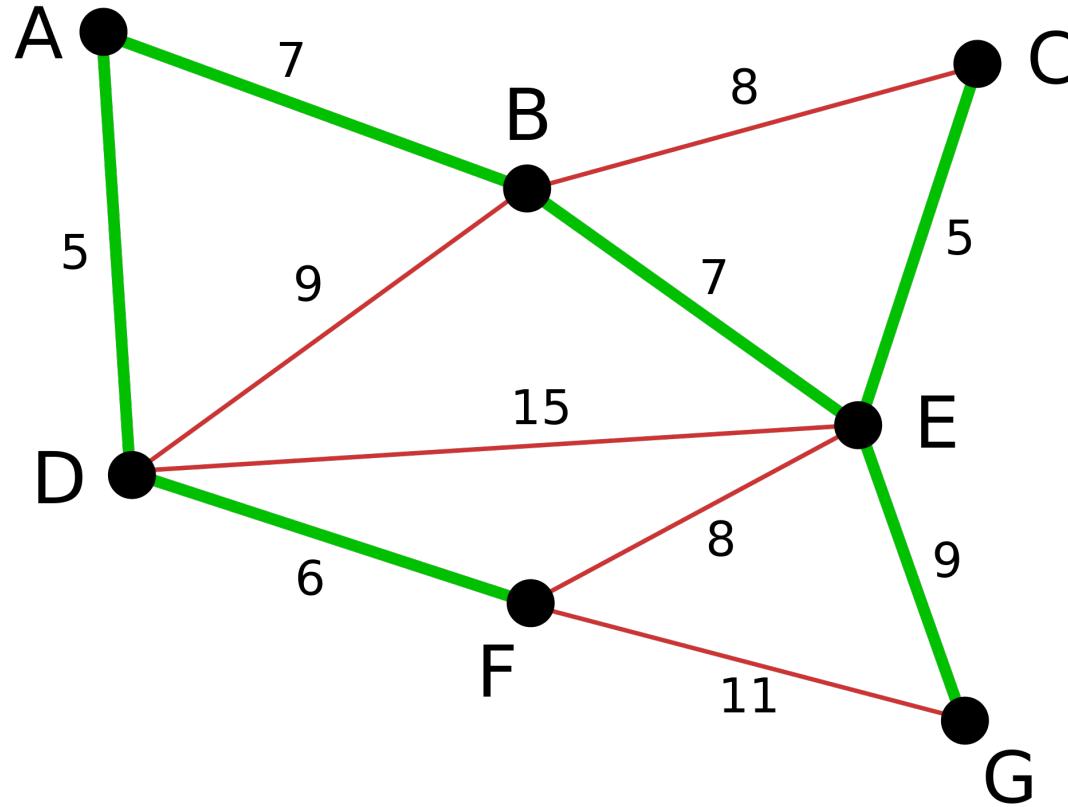
الگوریتم کراسکال (Kruskal)



الگوريتم کراسکال (Kruskal)



الگوريتم کراسکال (Kruskal)



الگوریتم پرایم (Prim)

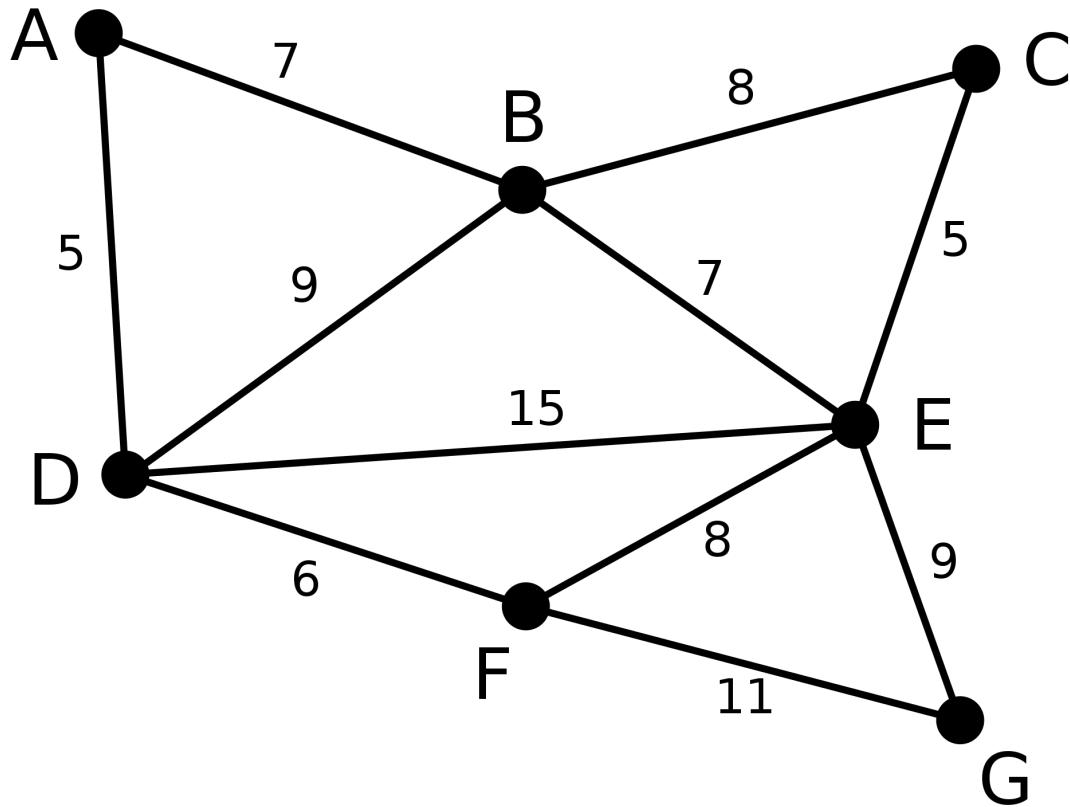
الگوریتم پرایم با یک درخت مانند T که تنها شامل یک راس دلخواه از گراف اصلی است، شروع می‌کند. سپس یک یال با کمترین هزینه به گونه‌ای انتخاب می‌شود که:

- یکی از رئوسش در درخت T باشد.
- وزن آن کمینه باشد.

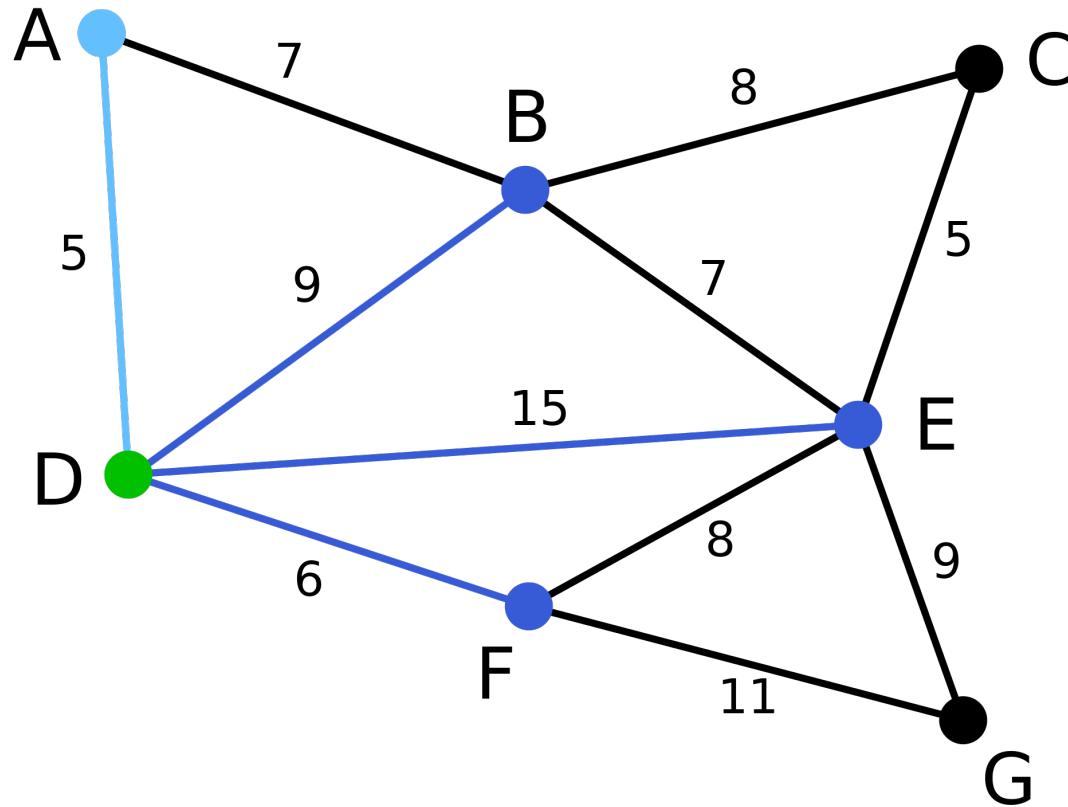
این عمل را تا زمانی که T شامل $n-1$ یال باشد، ادامه می‌دهیم.

توجه: برای اطمینان از این که یال‌های اضافه شده تشکیل یک دور نمی‌دهند، در هر مرحله یالی مانند (u,v) با هزینه مینیمم را به گونه‌ای انتخاب می‌کنیم که دقیقاً یکی از رئوس u یا v در T وجود داشته باشد.

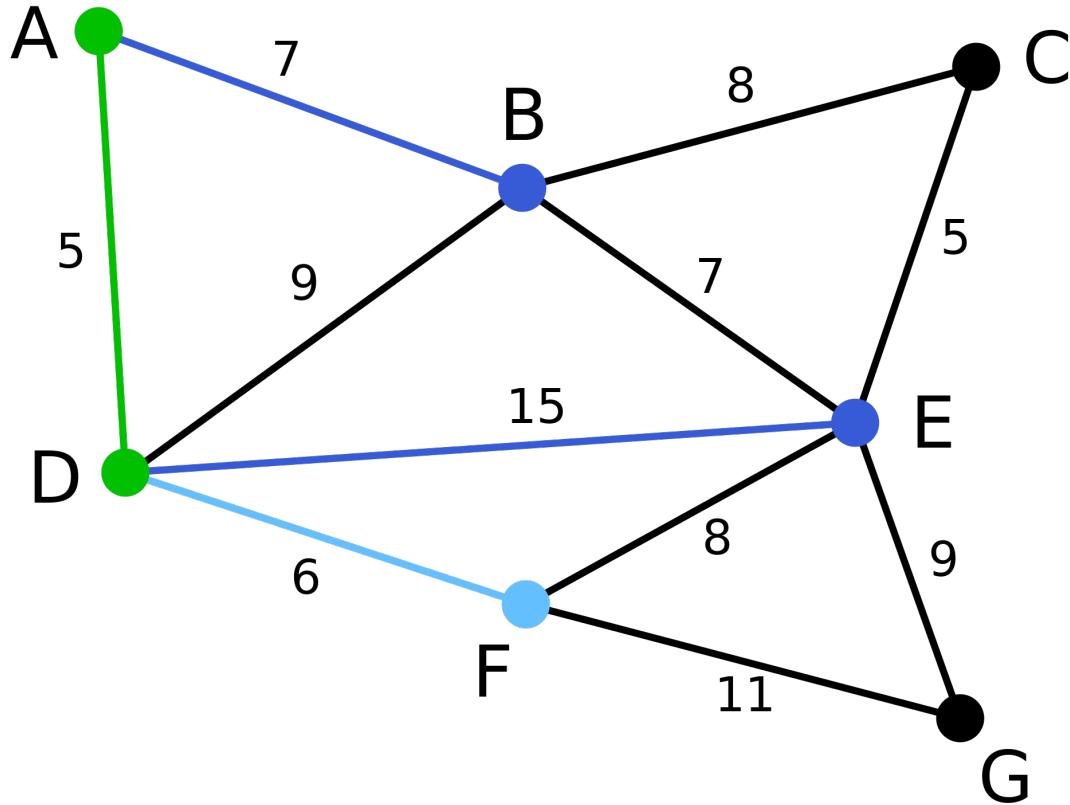
الگوریتم پریم (Prim)



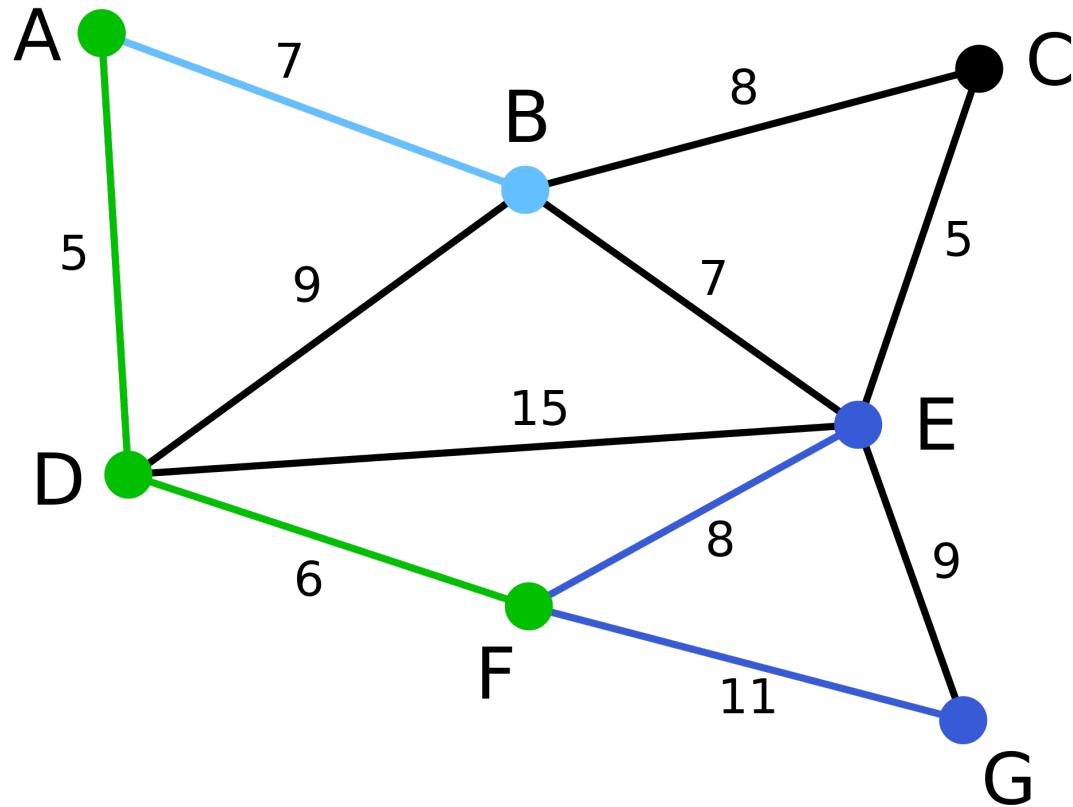
الگوریتم پریم (Prim)



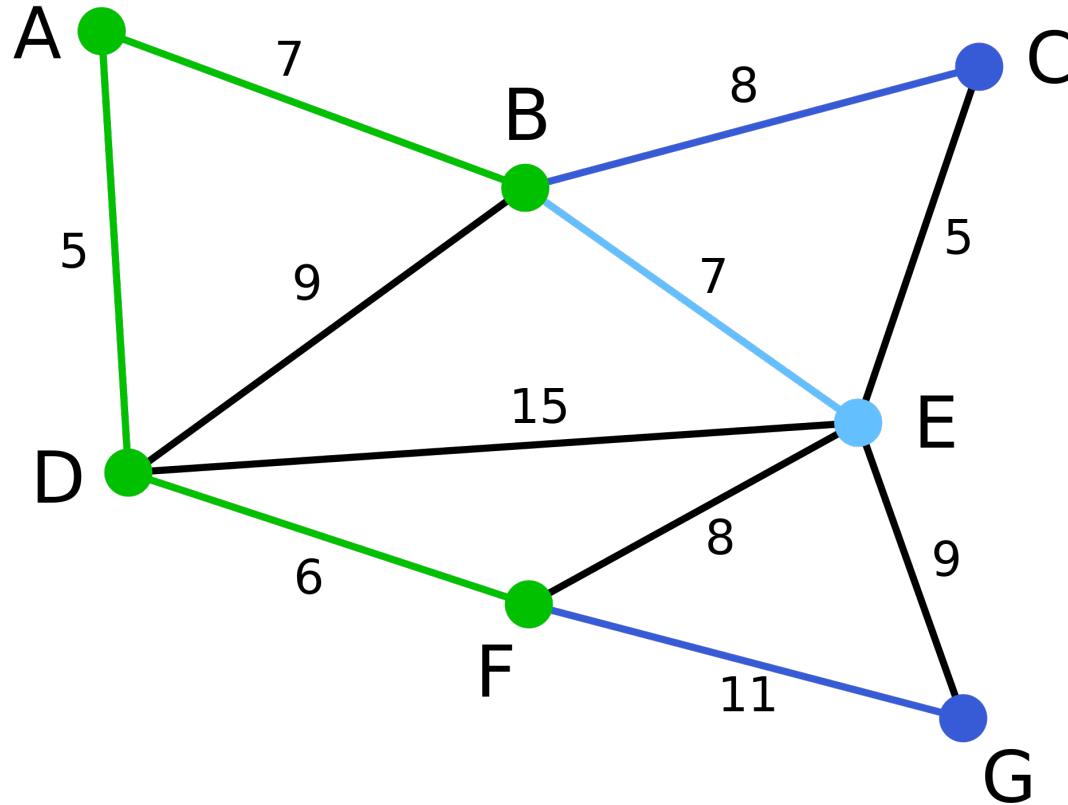
الگوریتم پریم (Prim)



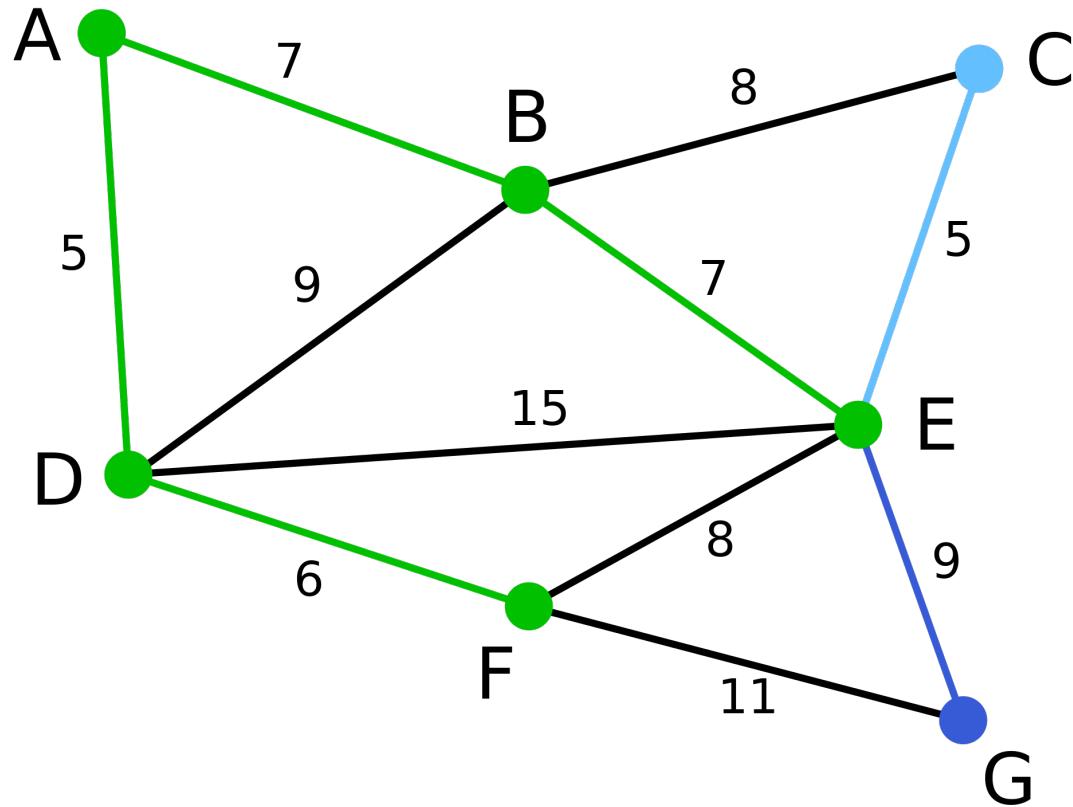
الگوریتم پریم (Prim)



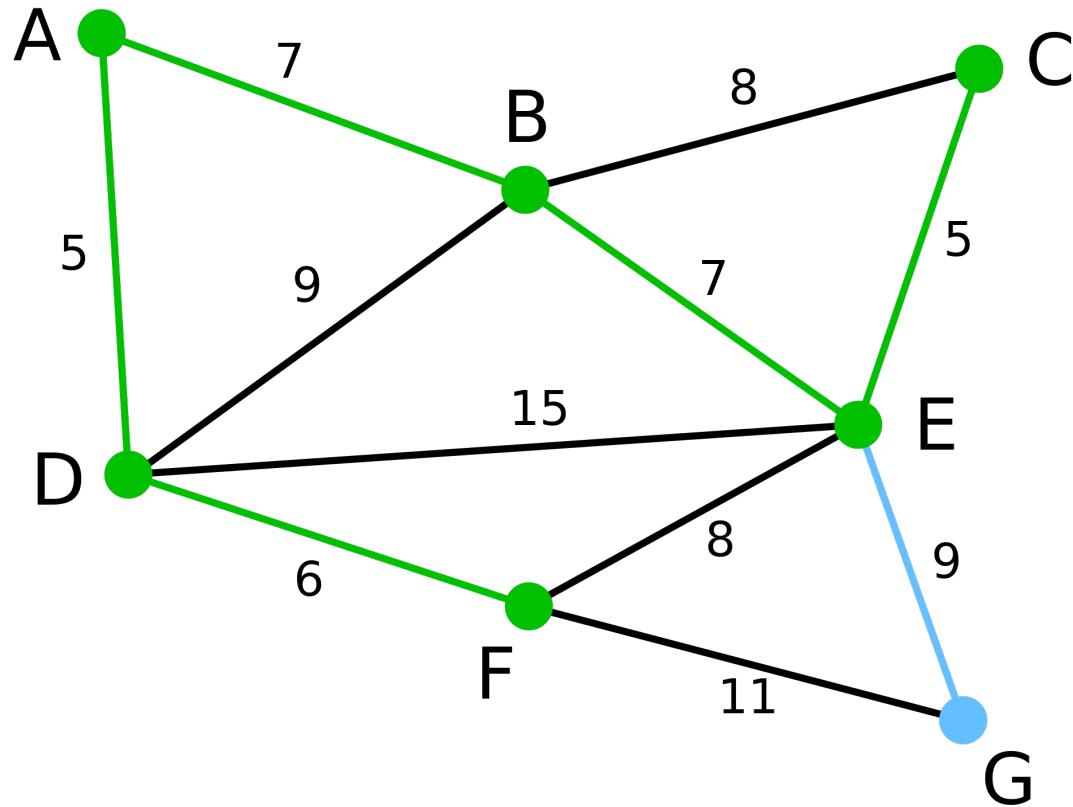
الگوریتم پریم (Prim)



الگوریتم پریم (Prim)



الگوریتم پریم (Prim)



الگوریتم پریم (Prim)

