گراف ها

Graphs

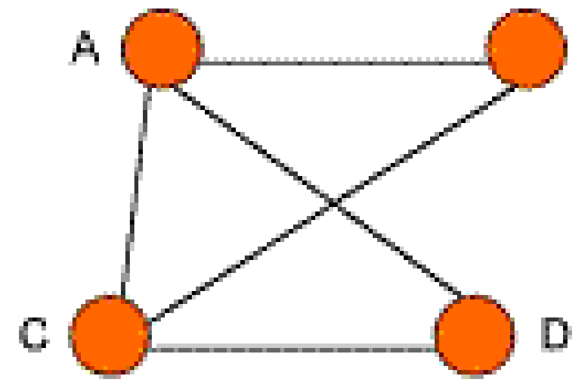
گراف

- یک گراف شامل دو مجموعه است:
  - مجموعه غیر تهی از گره ها یا رئوس (Vertex)
  - مجموعه ای از یال ها (Edge) که راس ها را به هم متصل می کنند.
- مثال: می توان شهر های یک کشور را رئوس و جاده های بین آن ها را یال های یک گراف تصور کرد.
- به هر راس یا هر یال گراف نامی اختصاص داده می شود.
- یک گراف تهی null graph گرافی است که تنها شامل راس است و مجموعه یال های آن تهی است یعنی یالی ندارد.

گراف جهت دار و غیر جهت دار

- یک گراف می تواند به دو شکل جهت دار directed یا غیرجهت دار undirected باشد.
- یک گراف جهت دار گرافی است که جهت هر یال در آن تعیین شده است.
- در گراف جهت دار ترتیب رئوس در هر یال اهمیت دارد و یال ها با پیکان هائی از راس ابتدا به راس انتها رسم می شوند.
- در گراف غیرجهت دار می توان در هر دو جهت بین راس ها حرکت کرد و ترتیب راس های یال اهمیت ندارد.

گراف جهت دار و غیر جهت دار



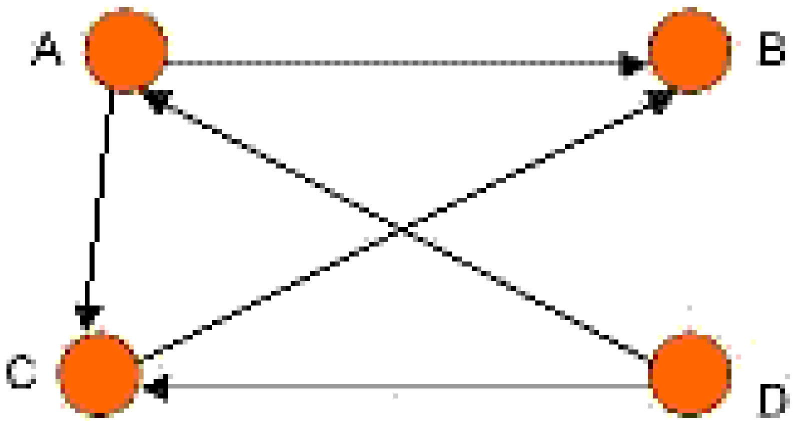
1) گراف بدون جهت

گراف شامل دو مجموعه است $G = \{V, E\}$

مجموعه ای از رئوس $V = \{A, B, C, D\}$

$E = \{ \{A,B\}, \{A, C\}, \{A, D\}, \{C, B\}, \{C, D\} \}$

مجموعه ای از مجموعک های دو عضوی به عنوان یال ها



2) گراف جهتدار

گراف شامل دو مجموعه است $G = \{V, E\}$

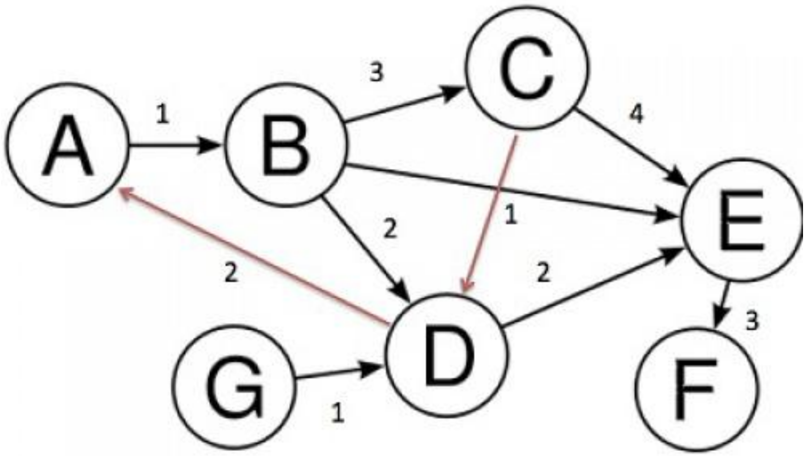
مجموعه ای از رئوس $V = \{A, B, C, D\}$

$E = \{ (A,B), (A, C), (D, A), (C, B), (D, C) \}$

مجموعه ای از زوج های مرتب به عنوان یال ها که عضو اول شروع و عضو دوم انتهای یال می باشد

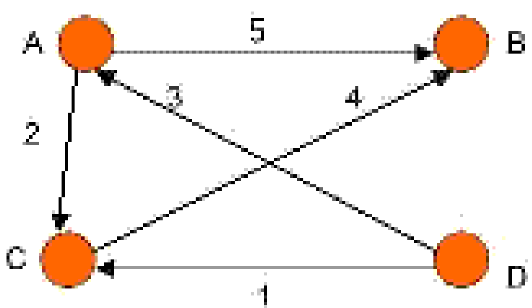
گراف وزن دار

- یال های گراف می توانند وزن دار weighted یا بدون وزن unweighted باشند.
- گرافی که یال های آن وزن باشد گراف وزن‌دار نامیده می شود.
- وزن می تواند نشان دهنده هزینه، مسافت، زمان یا هر مشخصه دیگری از یال باشد.

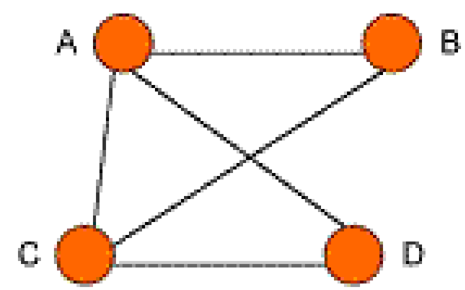


مجاورت در گراف

- هر یال بوسیله یک جفت راس مشخص می شود.
- دو راسی که توسط یک یال به هم متصل می شوند را رئوس مجاور adjacent
- یال را یک لبه تلاقی incident دو آن رو راس می نامند.
- به طور مثال در هر دو گراف ۲و ۱، دو گره A, B مجاور هم هستند، زیرا به وسیله یک یال بهم متصل شده اند.



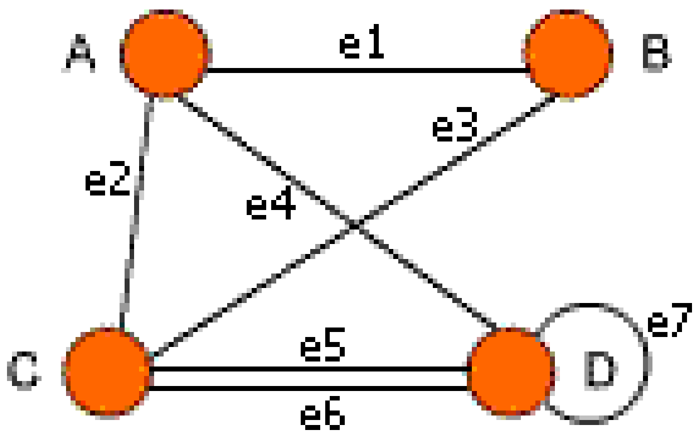
2) گراف جهتدار



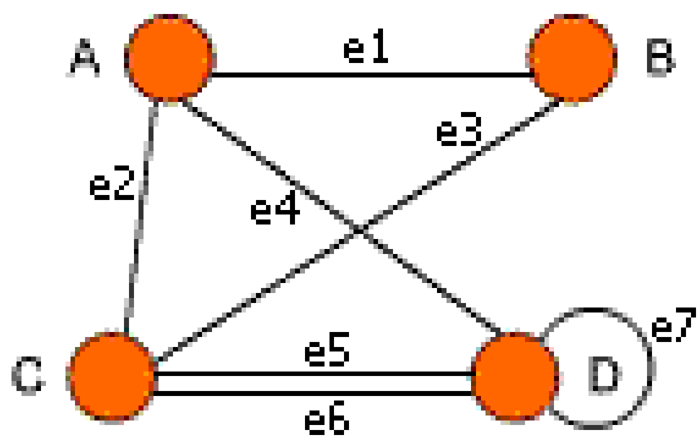
1) گراف بدون جهت

حلقه در گراف

- یک حلقه loop، یالی است که یک راس را به خودش متصل می کند.
- به عبارت دیگر راس ابتدا و انتهایش یکسان باشد.
- در شکل زیر یال e7 یک حلقه روی راس D است



یال های موازی در گراف

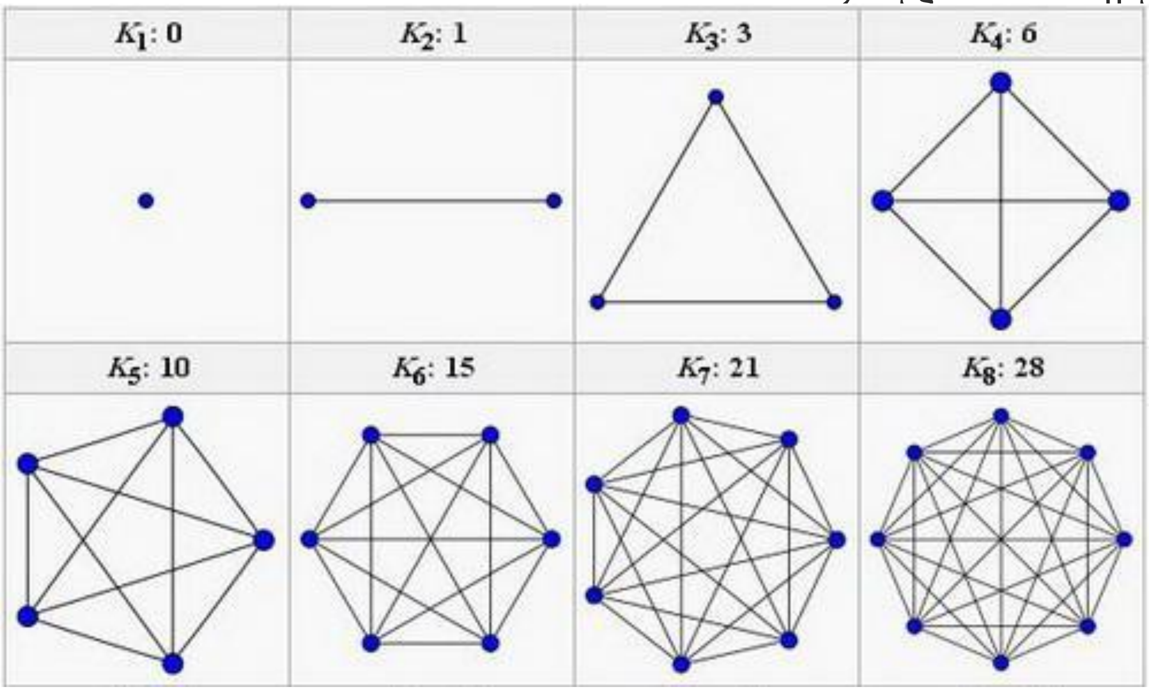
- یال های موازی parallel edges یا چندگانه:
  - یال‌هایی هستند که رئوس یکسان را بهم مرتبط می‌کنند.
- گرافی که دارای یال های موازی باشد را گراف چندگانه multigraph می نامند.
- مثال: در گراف چندگانه زیر یال های e5 و e6 که رئوس C و D را به هم متصل می کنند موازی هستند
- 

گراف ساده

- گراف بدون یال موازی و بدون حلقه را گراف ساده simple graph می‌نامند.
- گراف جهتدار را وقتی ساده می گویند که یال موازی نداشته باشد.
- حداکثر تعداد یال‌ها در یک گراف جهتدار با n راس برابر است با $n×(n-1)$
- حداکثر تعداد یال‌ها در یک گراف غیرجهتدار با n راس برابر است با $n×(n-1)/2$

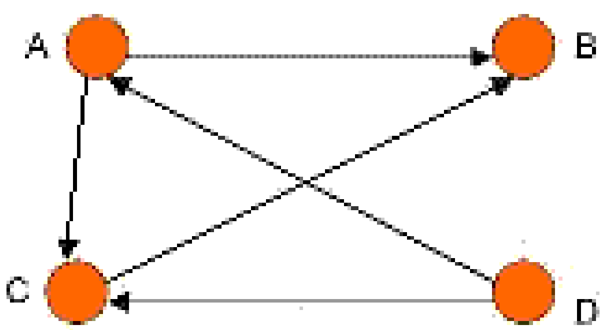
گراف کامل

- یک گراف کامل complete graph گراف ساده ای است که هر جفت راس آن مجاور باشند.
- یعنی هر از راس به کلیه راس های دیگر یالی وجود داشته باشد.
- به چنین گرافی K_n گفته می شود.



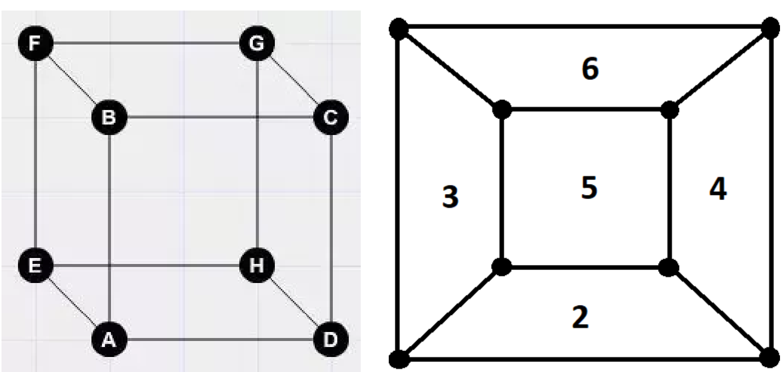
درجه گراف

- درجه degree هر راس: تعداد یال های متلاقی با آن راس می باشد.
- در گراف جهتدار
  - درجه ورودی indegree یک راس: تعداد یال هائی است که به آن راس وارد شده اند
  - درجه خروجی outdegree یک راس: تعداد یآل هائی است که از آن راس خارج شده اند.
- مثال: در گراف زیر درجه خروجی راس A دو و درجه ورودی آن یک است
- درجه گراف برابر درجه ماکزیمم رئوس گراف است.

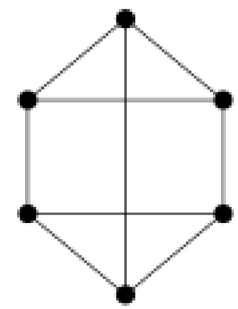


گراف منتظم

- گرافی که کلیه راس های آن از یک درجه باشد گراف منتظم regular graph نامیده می شود.
- گراف مکعب گراف منتظم درجه 3 است.
- دریک گراف مجموع درجات کلیه رئوس همواره عددی زوج است



گراف مکعب



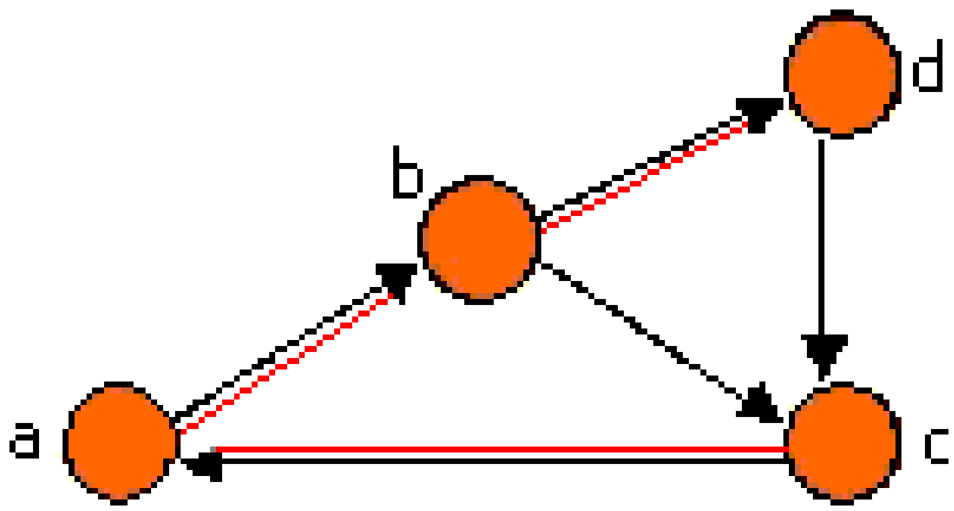
گراف ۳-منتظم

مسیر در گراف

- یک مسیر path در گراف: یک گذر از راس های متوالی در امتداد یک سری از یال ها است.
- راس انتهای یک یال راس ابتدای یال بعدی در توالی محسوب می شود.
- طول مسیر تعداد یال های مسیر است که در طول مسیر طی می شود.
- یک مسیر با طول n دارای n+1 راس و n یال است.
- در یگ گراف وزن دار طول مسیر برابر مجموع وزن‌های یال‌های مسیر است.
- دوراس را متصل reachable می‌گویند اگر مسیری بین آنها وجود داشته باشد.
- یک مسیر simple path ساده مسیری است که همه رئوس آن بجز احتمالا راس شروع و پایان تکراری نباشد.

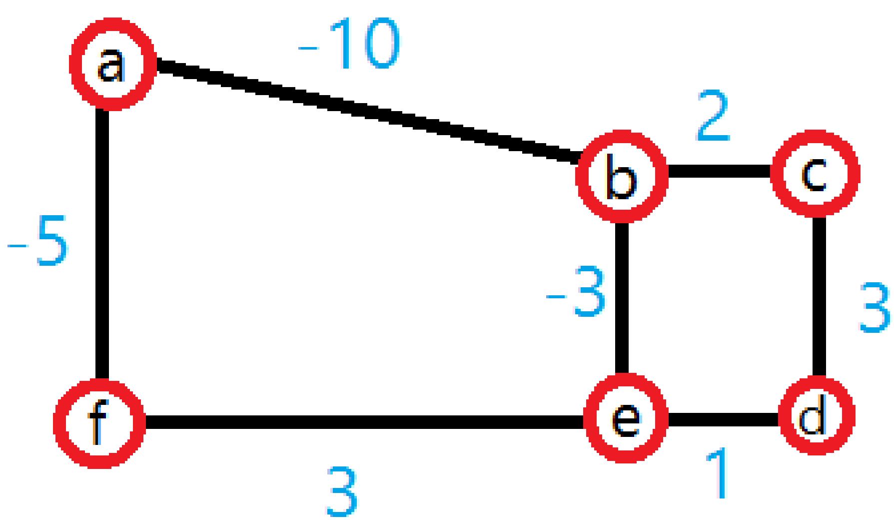
مسیر در گراف

- مثال. در شکل زیر یک مسیر نشان داده شده است که از راس Cآغاز و به راس Dختم می شود.
- با خطوط قرمز رنگ نشان داده شده است.



دور در گراف

- یک دور cycle: مسیر ساده ای است که راس شروع و پایانی آن یکی باشد.
- گراف ساده جهتداری که دارای دور نیست را غیر مدور acyclic می نامند.
- یک دور در گراف ساده بدون جهت حداقل شامل سه یال متفاوت است که هیچ راسی در آن تکراری نیست بجز راس شروع و پایان.
- گراف زیر دارای سه دور می باشد



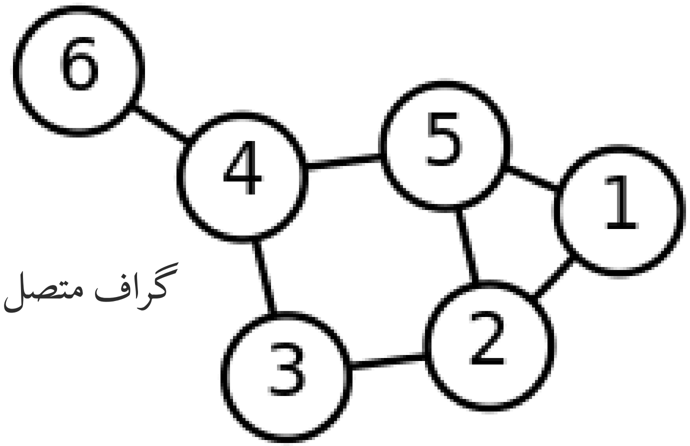
دور a-b-e-f-a
دور a-b-c-d-e-f-a منفی هستند.
دور b-c-d-e-b مثبت است.

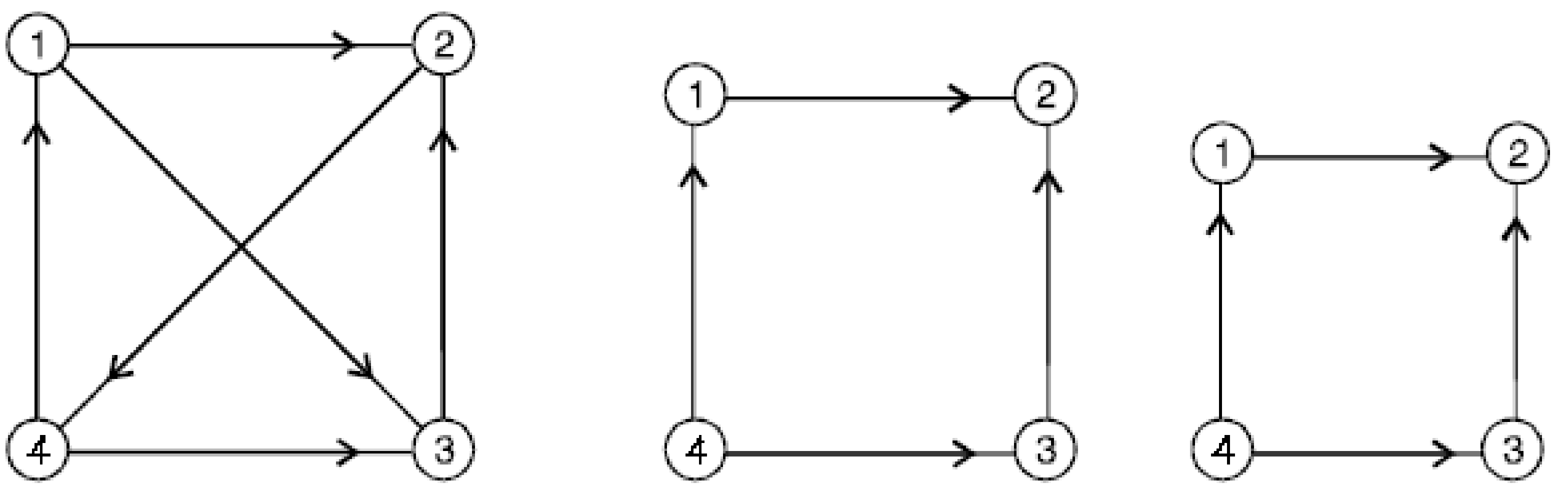
اتصال در گراف

- یک گراف غیرجهتدار، متصل یا همبند connected گفته می شود، اگر مسیری بین هر جفت راس آن وجود داشته باشد. یعنی هر دو راس آن متصل باشند.
- در گراف جهتدار چون جهت باید درنظر گرفته شود اتصال پیچیده تر است. ممکن است راس a به b متصل باشد ولی مسیری از راس b به a وجود نداشته باشد.
- در گراف جهتدار ساده سه حالت برای اتصال وجود دارد:
  - متصل ضعیف weakly connected یک گراف متصل ضعیف گرافی است که اگر جهت گراف ندیده گرفته شود متصل است.
  - متصل یکطرفه unilaterally connected یک گراف متصل یکطرفه گرافی است که حداقل یک راس آن به هر راس دیگری متصل باشد.
  - متصل قوی strongly connected یک گراف متصل قوی گرافی است که هر جفت راس متصل باشد.

اتصال در گراف

گراف متصل بدون جهت





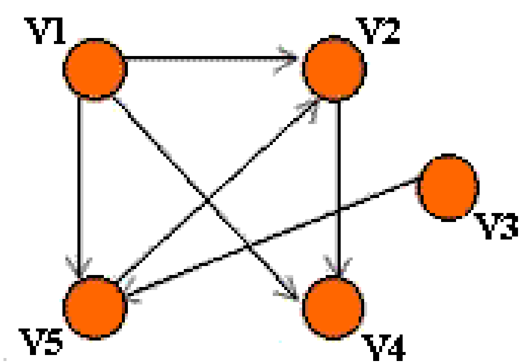
(a) Strongly connected (b) Weakly connected (c) Unilateraly connected

نمایش گراف

- راه های متعددی برای نمایش گراف در کامپیوتر وجود دارد دو ساختمان داده پایه ای که برای نمایش گراف استفاده می شوند:
  - ماتریس مجاورت Adjacency Matrix
  - لیست مجاورتی Adjacency List

نمایش گراف – ماتریس مجاورت

- ماتریس مجاورت adjacency matrix گراف G با n راس (که رئوس آن به ترتیب از v1 تا vn نامگذاری شده است)، یک ماتریس بیتی n×n با نام A است که در آن:
  - درایه a_{ij} برابر با 1 است اگر یالی از vi به vj وجود داشته باشد
  - درایه a_{ij} برابر با 0 است اگر یالی از vi به vj وجود نداشته باشد
  - ماتریس مجاورت برای یک گراف بدون جهت متقارن است.



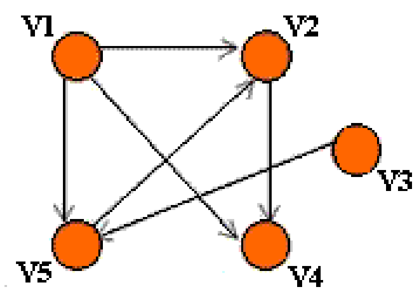
گراف

$$
\begin{array}{c|ccccc}
 & V1 & V2 & V3 & V4 & V5 \\
\hline
V1 & 0 & 1 & 0 & 1 & 1 \\
V2 & 0 & 0 & 0 & 1 & 0 \\
V3 & 0 & 0 & 0 & 0 & 1 \\
V4 & 0 & 0 & 0 & 0 & 0 \\
V5 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

نمایش گراف به فرم ماتریس مجاورت

نمایش گراف - ماتریس مجاورت

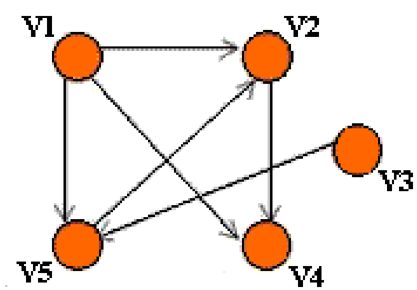
- در یک گراف غیر جهتدار درجه راس vi برابر با مجموع عناصر سطر iام در ماتریس مجاورت است.
- در یک گراف جهتدار درجه خروجی راس vi برابر مجموع عناصر سطر iام و درجه ورودی آن برابر مجموع عناصر ستون iام در ماتریس مجاورت است.
- فضای مورد نیاز در روش مانریس مجاورت برای نمایش یک گراف با مجموعه رئوس Vبرابر $O(|V|^2)$ است.
- اگر تعداد یال‌های گراف کم باشد می‌توان آنرا به صورت ماتریس اسپارس نمایش داد.



$$\begin{array}{c} \\ V1 \\ V2 \\ V3 \\ V4 \\ V5 \end{array} \begin{array}{c} \begin{array}{ccccc} V1 & V2 & V3 & V4 & V5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

نمایش گراف – ماتریس مجاورت

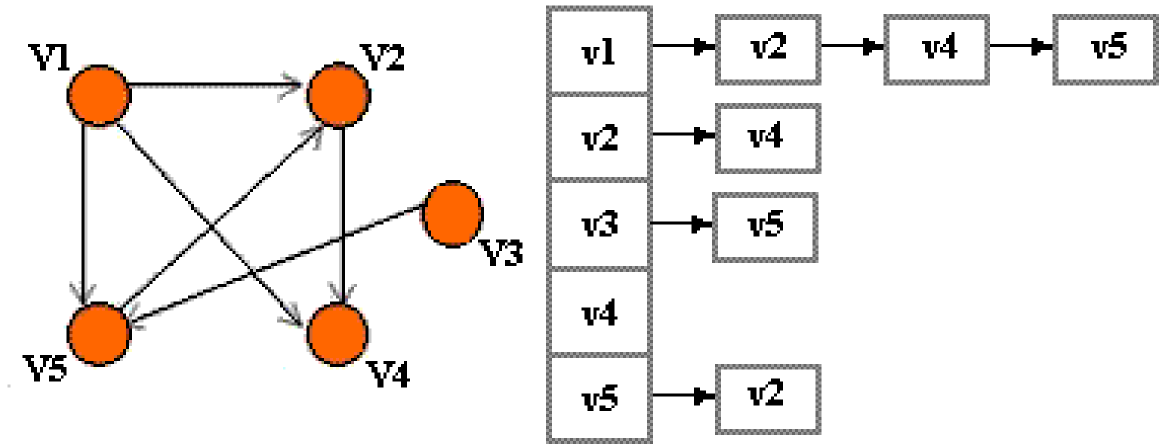
- مزیت اصلی نمایش ماتریسی در این است که محاسبه مسیر ها و دورها بسادگی توسط عملیات ماتریسی قابل انجام است.
- مجاورت بین دو راس با پیچیدگی زمان $O(1)$ تعیین می شود و اجازه رسم حلقه در گراف را می دهد.
- اشکال آن این است که از جنبه ظاهری گراف دور است و خواصی که بسادگی در شکل گراف نمایان است توسط ماتریس به سختی قابل رویت است.
- علاوه بر این ماتریس همجواری یال‌های موازی را نشان نمی‌دهد.



$$
\begin{array}{c c}
 & \begin{array}{ccccc} V1 & V2 & V3 & V4 & V5 \end{array} \\
\begin{array}{c} V1 \\ V2 \\ V3 \\ V4 \\ V5 \end{array} &
\begin{bmatrix}
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

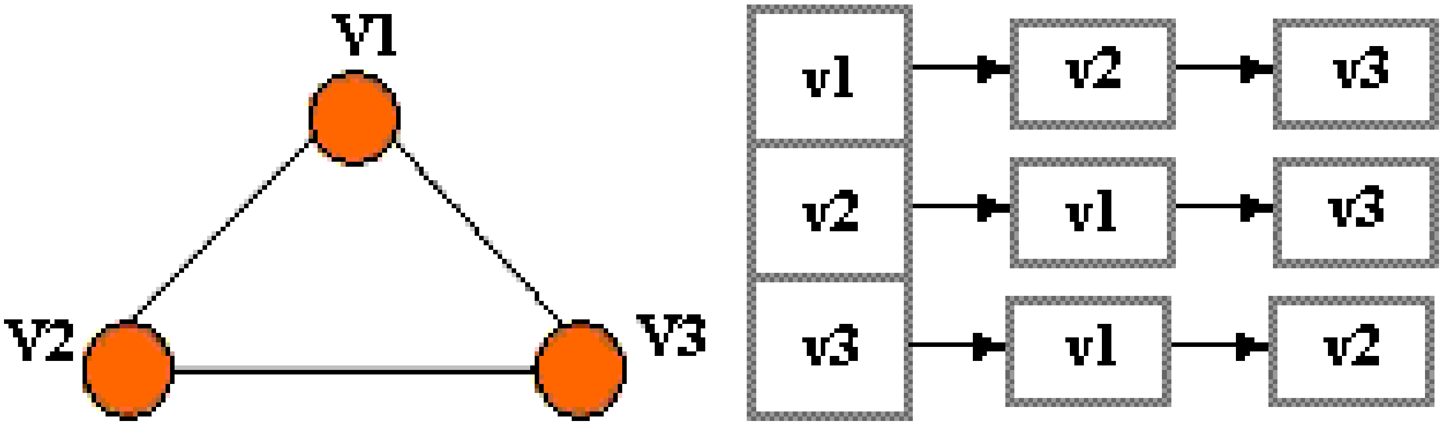
نمایش گراف - لیست مجاورت

- لیست مجاورتی adjacency list فرم دیگر نمایش گراف در کامپیوتر است.
- این ساختمان داده شامل لیستی از کلیه رئوس گراف است.
- برای هر راس یک لیست پیوندی وجود دارد که گره های آن رئوس مجاور راس را دربر می گیرند.
- به عبارت دیگر لیست i حاوی رئوسی است که مجاور راس viاست.



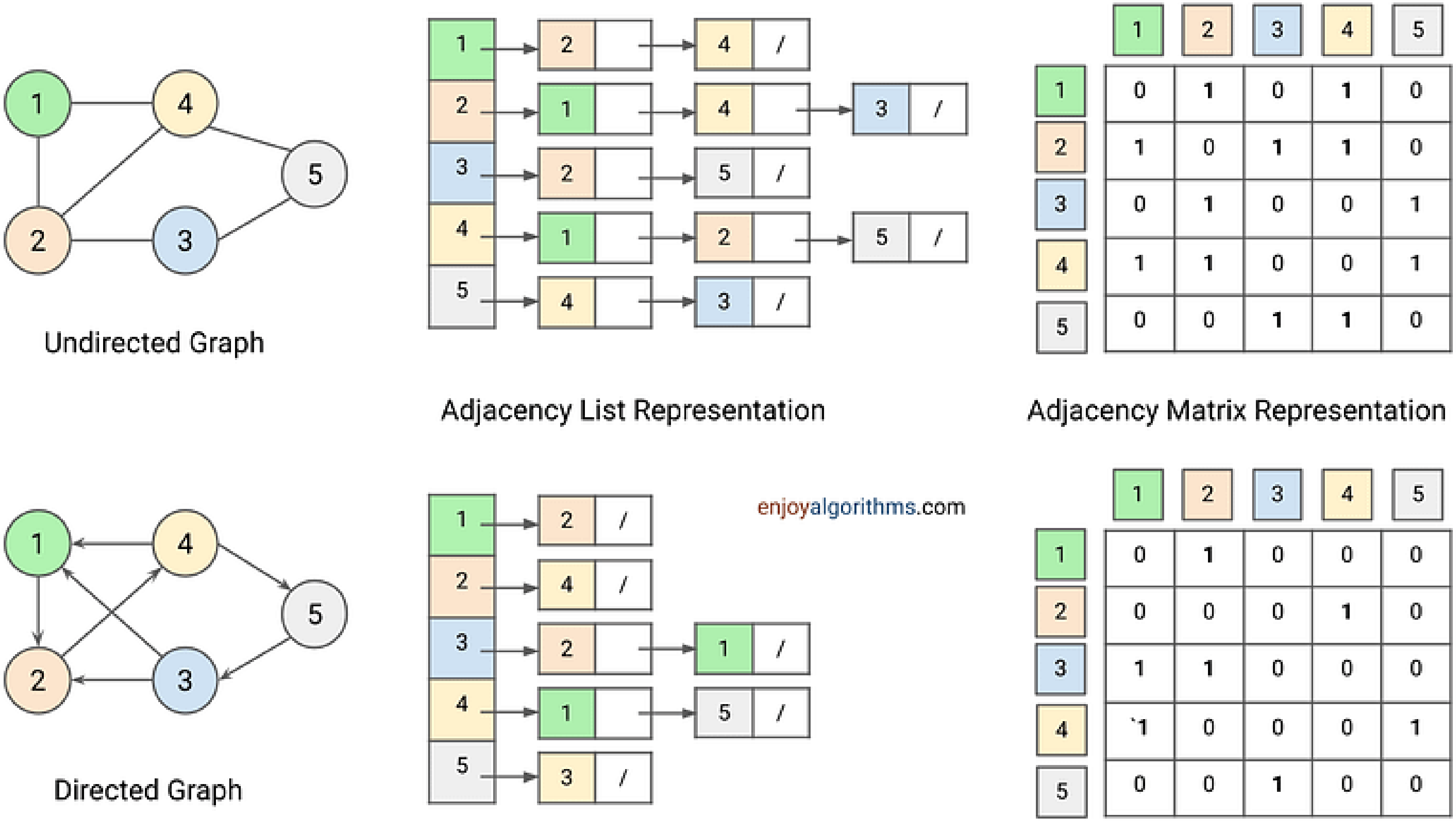
نمایش گراف – لیست مجاورت

- مثال : گراف غیر جهتدار زیر را درنظر بگیرید.
- لیست مجاورتی آن به صورت زیر است.



نمایش گراف - لیست مجاورت

- درجه هر راس در یک گراف غیر جهت دار با شمارش تعداد گره‌های لیست پیوندی مربوط به راس در لیست مجاورتی تعیین می‌شود.
- در یک گراف جهت دار، درجه خروجی هر راس با شمارش تعداد گره‌های لیست پیوندی مربوط به آن بدست می آید.
- اگر تعداد یال‌ها در گراف کم باشد این روش بهتر از ماتریس مجاورتی است.
- فضای مورد نیاز برای نمایش یک گراف با مجموعه رئوس V و مجموعه یال های E به روش لیست مجاورت برابر $O(|E|+|V|)$ می‌باشد.
- لیست مجاورتی به روشنی طبیعت مجاورتی رئوس گراف را نشان می دهد و اغلب زمانی استفاده می شود که گراف دارای تعداد یال های نسبتا متعادلی باشد.

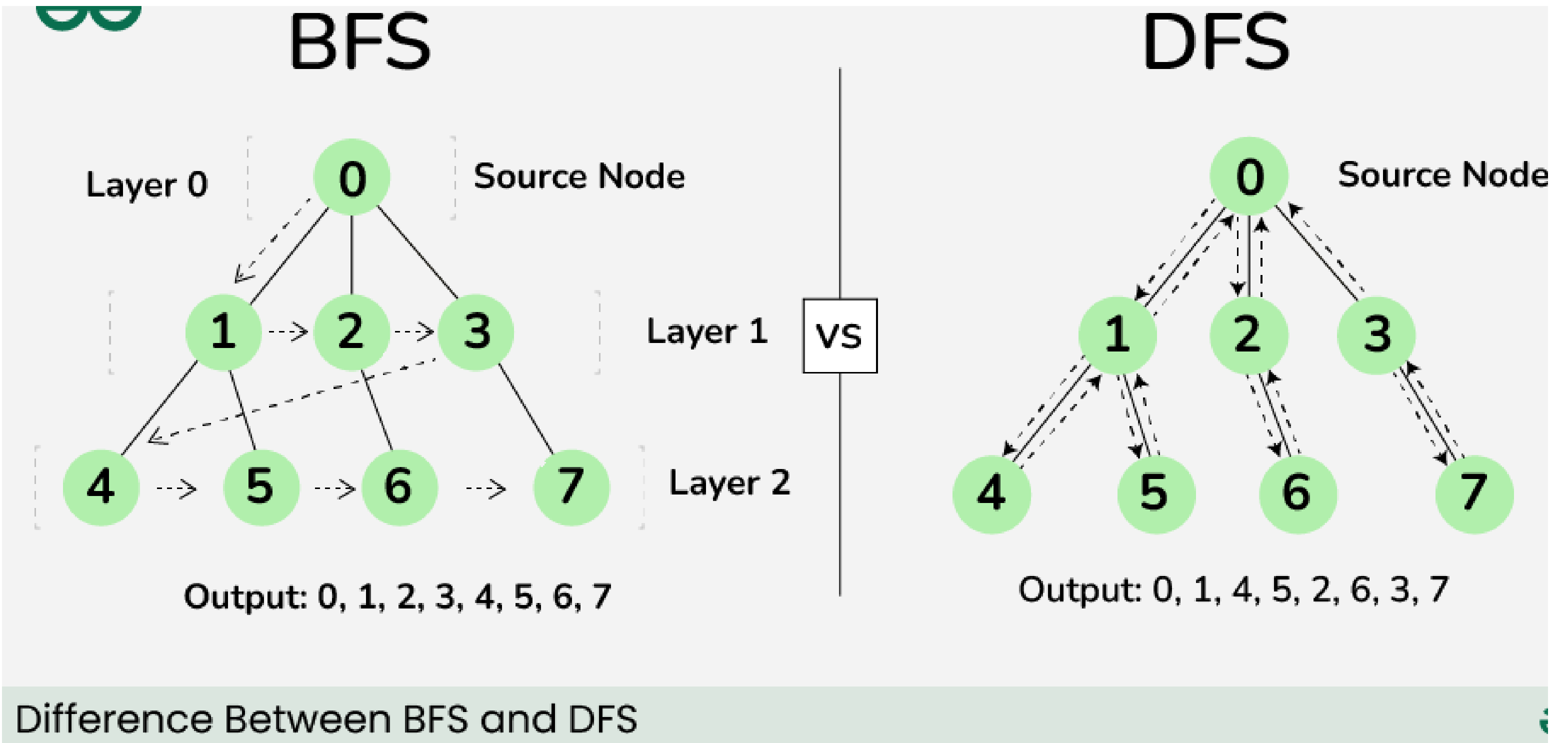

Undirected Graph
Adjacency List Representation
Adjacency Matrix Representation
enjoyalgorithms.com
Directed Graph

پیمایش گراف

- هدف از پیمایش این است که کلیه رئوسی که از طریق یک راس قابل دسترس هستند را بدست آوریم.
- دو روش معروف برای پیمایش وجود دارد:
  - جستجوی اول عمق Deep First Search
  - جستجوی اول سطح Breadth First Search

پیمایش گراف

- دو روش معروف برای پیمایش وجود دارد:
  - جستجوی اول عمق Deep First Search
  - جستجوی اول سطح Breadth First Search



پیمایش گراف- جستجوی اول عمق

- در جستجوی اول عمق پیمایش از یک راس آغاز می شود.
- هر راس که پردازش یا اصطلاحا ملاقات می شود یکی از رئوس مجاور آن که قبلا ملاقات نشده است انتخاب می شود و پیمایش با ملاقات راس مجاور آن ادامه پیدا می کند.
- اگر راس مجاوری وجود نداشت که قبلا ملاقات نشده باشد یک سطح به عقب بر میگردد.

پیمایش گراف- جستجوی اول عمق

شبه کد - بازگشتی

```
DFS (int v)
{
      int w
      Visited[v]:=1
      For (each vertex w adjacent to v)
         If (not visited[w]) then
            DFS(w)
         End if
      End For
}
```

برای گراف G با n راس و m یال، مرتبه اجرائی الگوریتم:

- وقتی گراف توسط ماتریس مجاورتی نمایش داده شده باشد

  O(n2)

- اگر از لیست مجاورتی استفاده شود O(m)

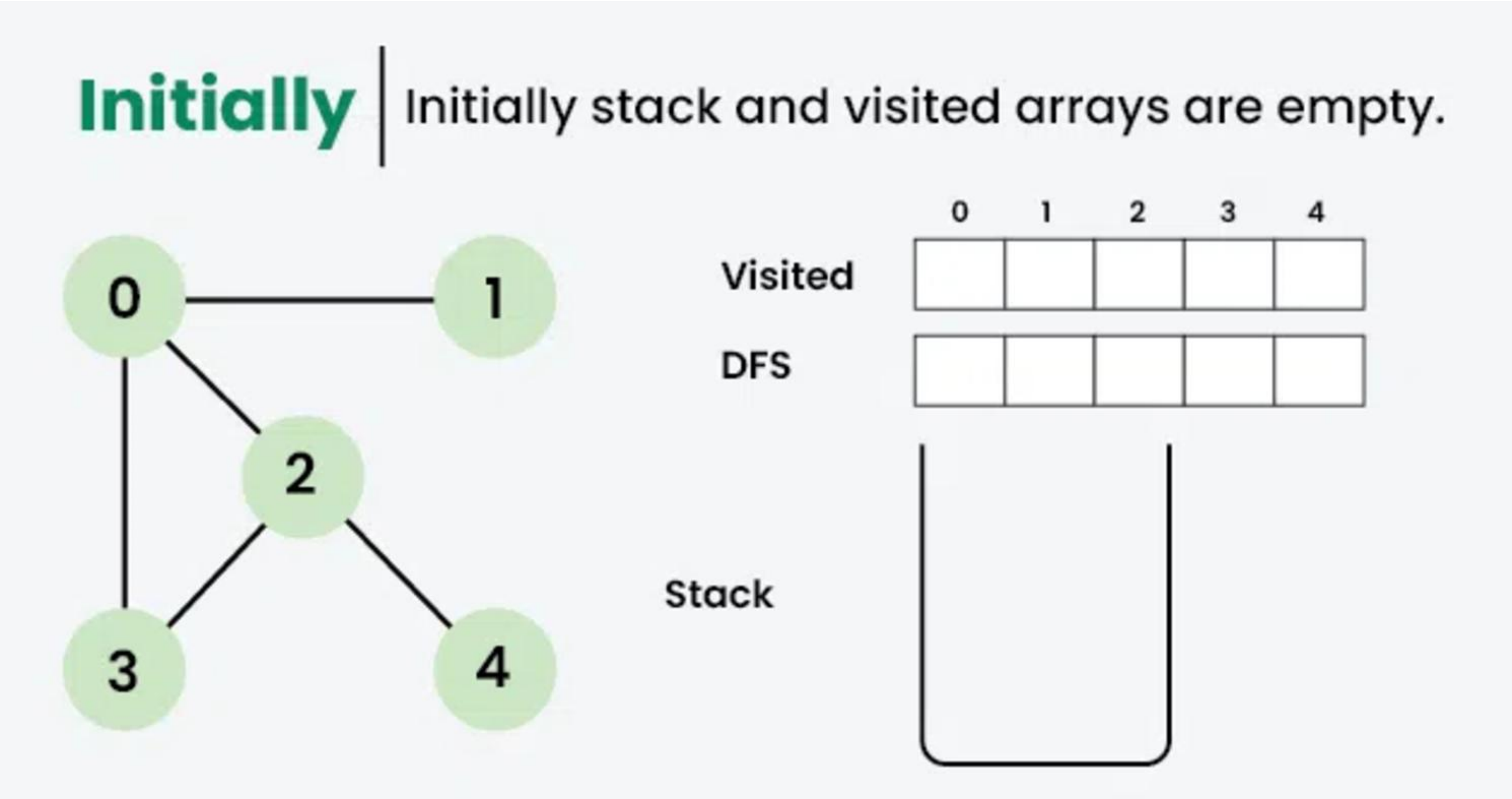
پیمایش گراف- جستجوی اول عمق

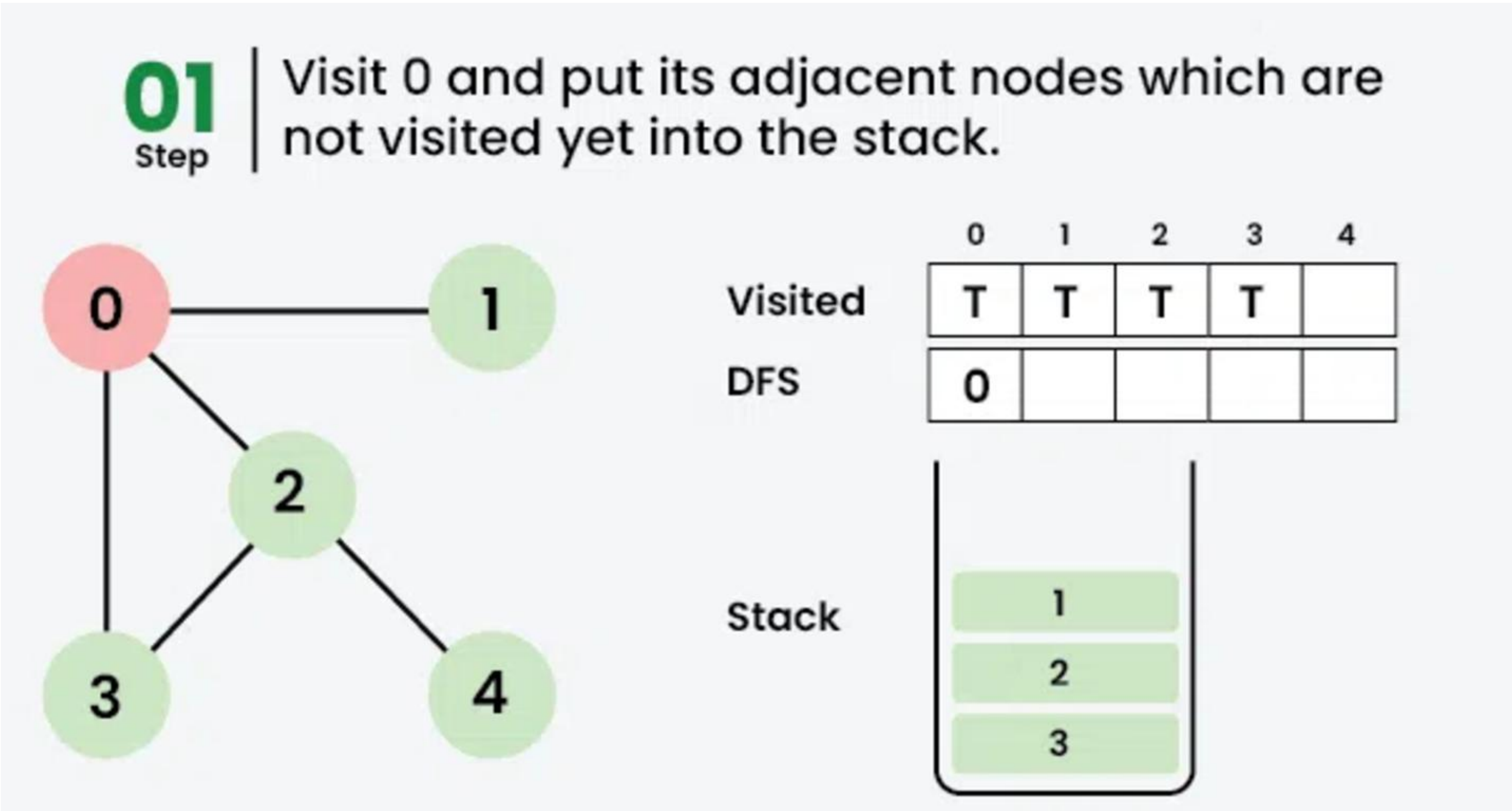
شبه کد - غیر بازگشتی

```
DFS(s):
  Initialize S to be a stack with one element s
  While S is not empty
    Take a node u from S
    If Explored[u] = false then
      Set Explored[u] = true
      For each edge (u, v) incident to u
        Add v to the stack S
      Endfor
    Endif
  Endwhile
```

برای گراف G با n راس و m یال، مرتبه اجرائی الگوریتم:

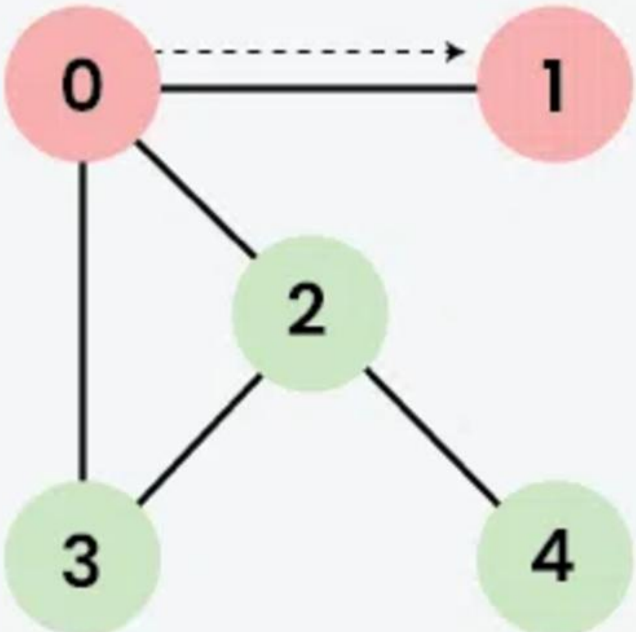
- وقتی گراف توسط ماتریس مجاورتی نمایش داده شده باشد
  O(n2)
- اگر از لیست مجاورتی استفاده شود O(m)


Initially
Initially stack and visited arrays are empty.
0
1
2
3
4
Visited
DFS
Stack
0
1
2
3
4

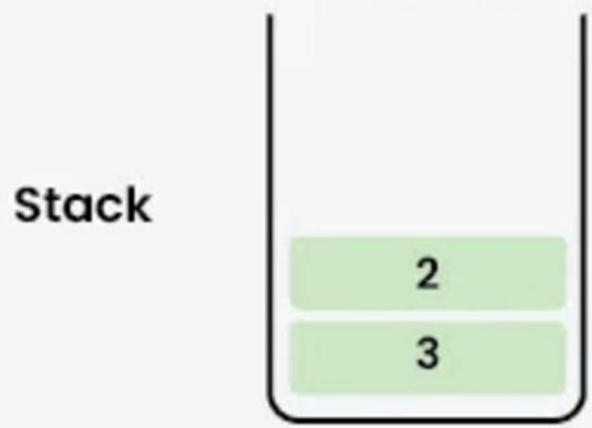

01
Step
Visit 0 and put its adjacent nodes which are not visited yet into the stack.
0
1
2
3
4
Visited
T
T
T
T
DFS
0
Stack
1
2
3

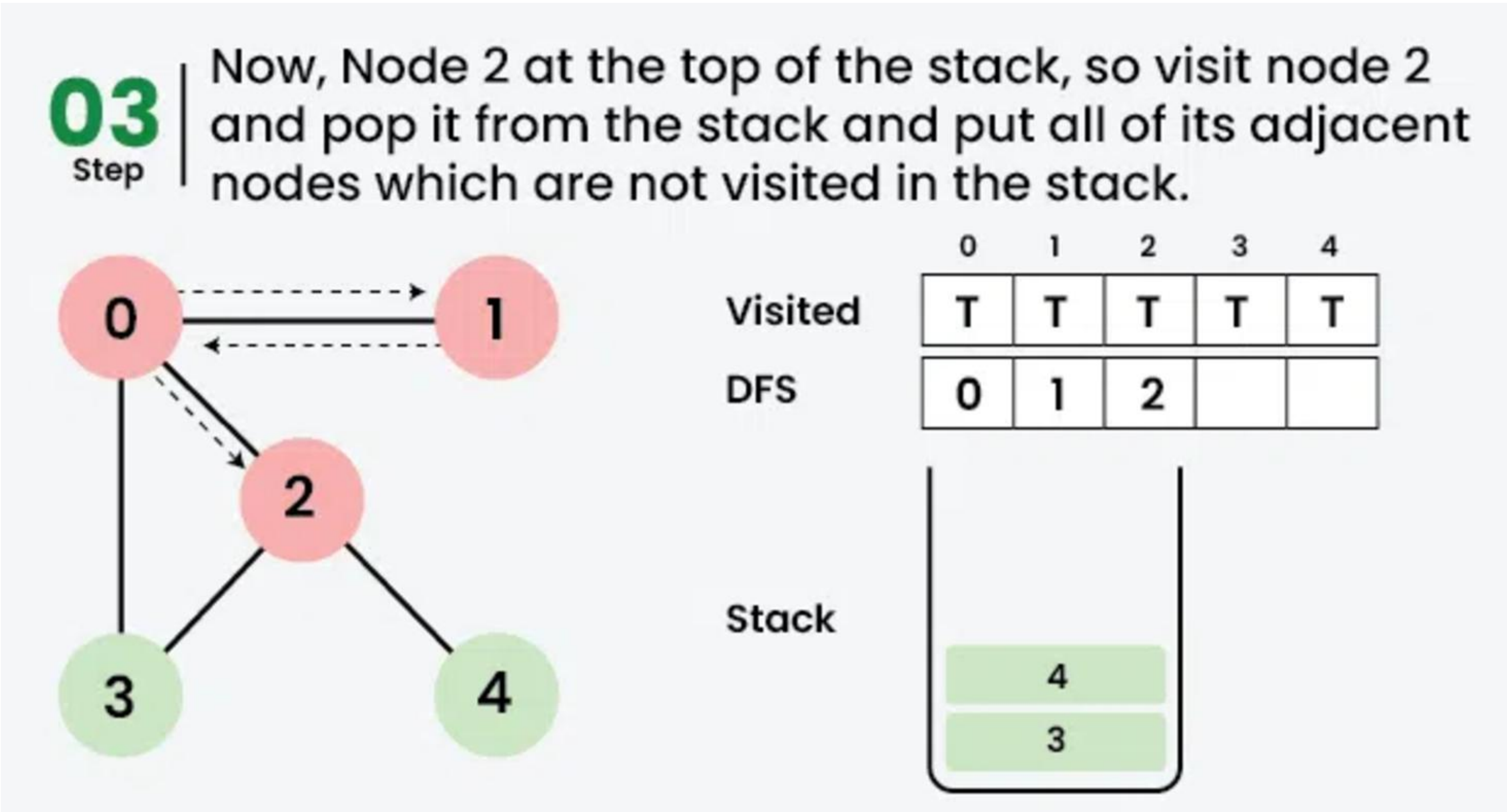
02 Step

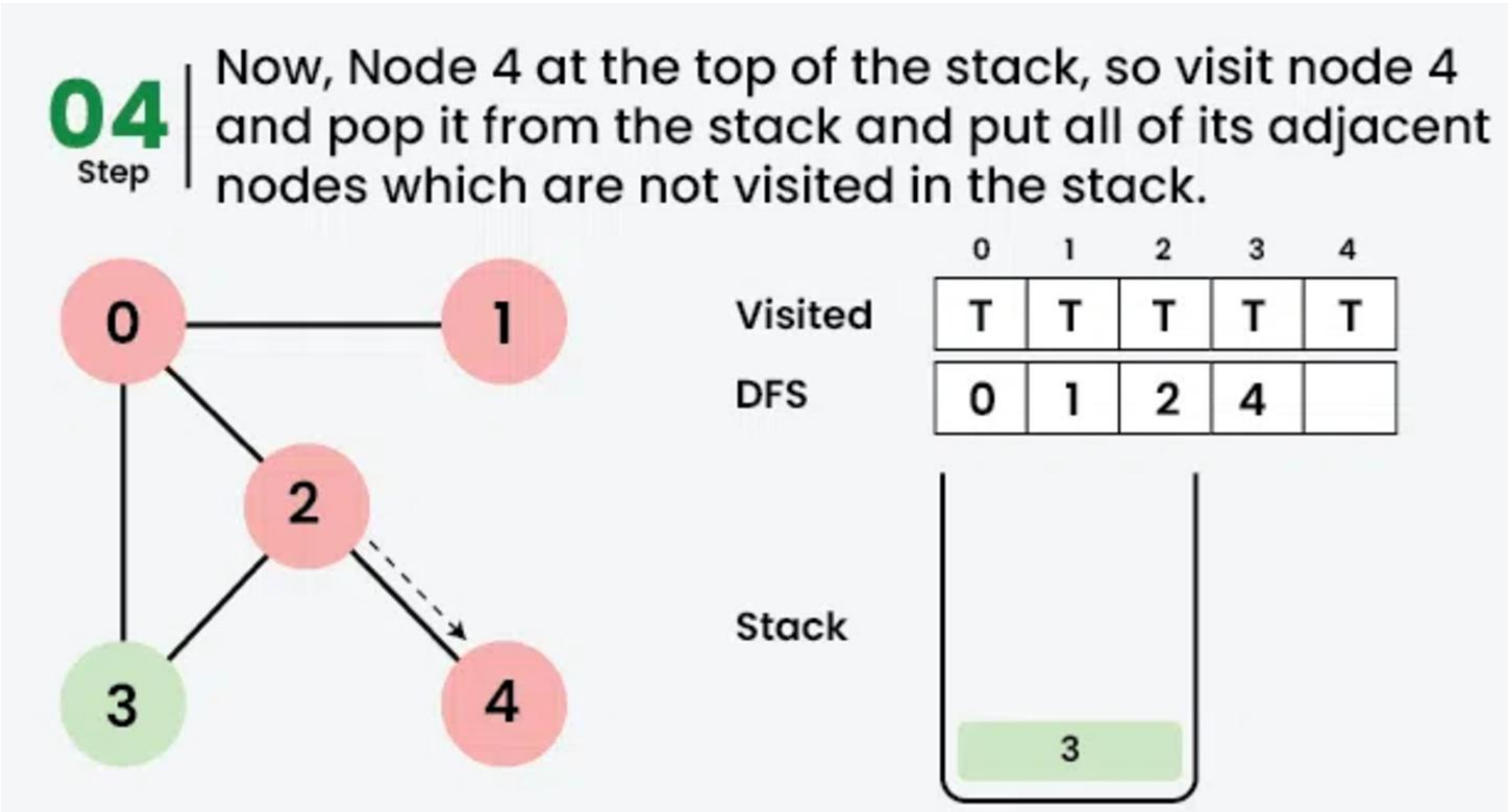
Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



	0	1	2	3	4
Visited	T	T	T	T	
DFS	0	1			

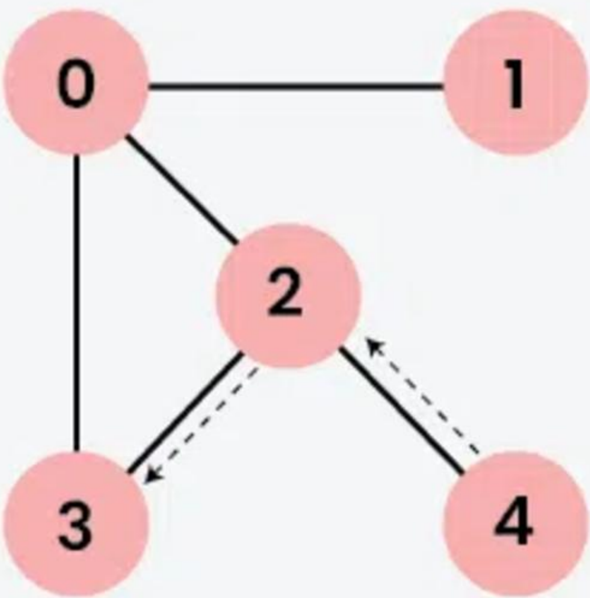



03
Step
Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.
0
1
2
3
4
Visited
T T T T T
DFS
0 1 2
Stack
4
3


04
Step
Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.
0
1
2
3
4
Visited
T T T T T
DFS
0 1 2 4
Stack
3

05 Step

Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.

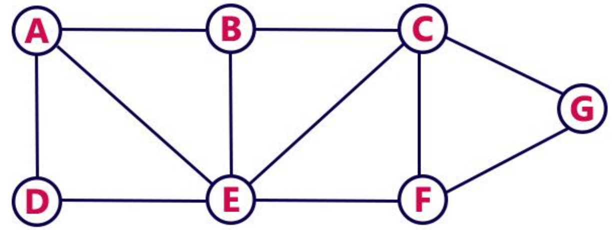


	0	1	2	3	4
Visited	T	T	T	T	T
DFS	0	1	2	4	3

Stack

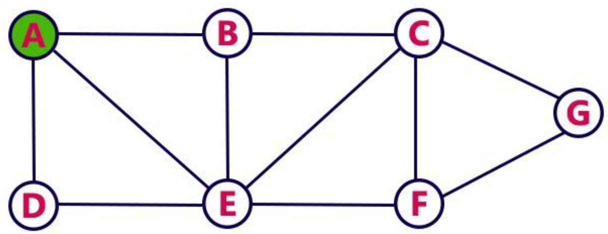
Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

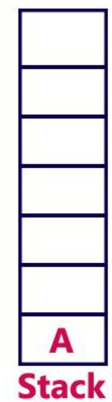
پیمایش گراف- جستجوی اول عمق



Step 1:

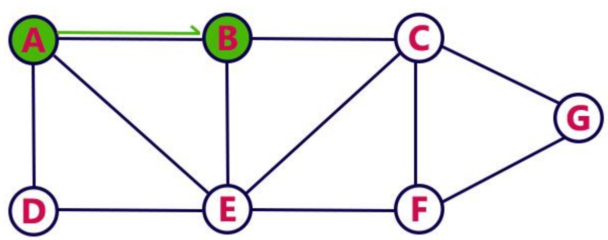
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

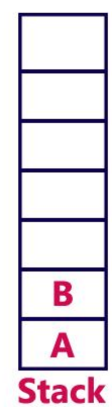




Step 2:

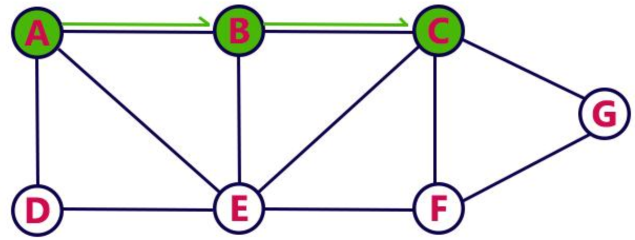
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

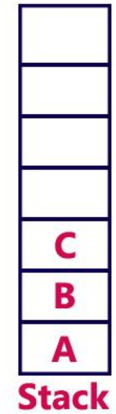




Step 3:

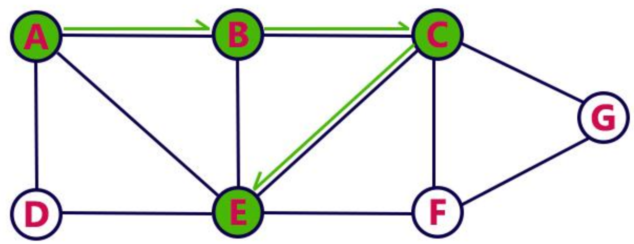
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

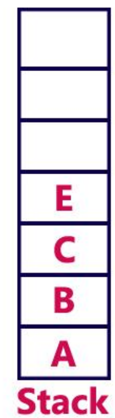




Step 4:

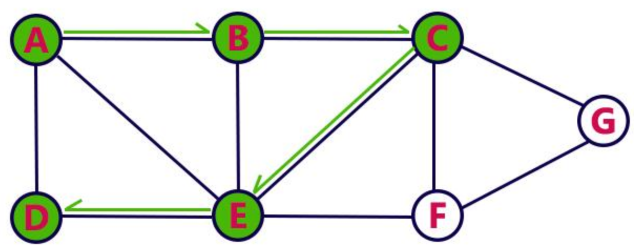
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack





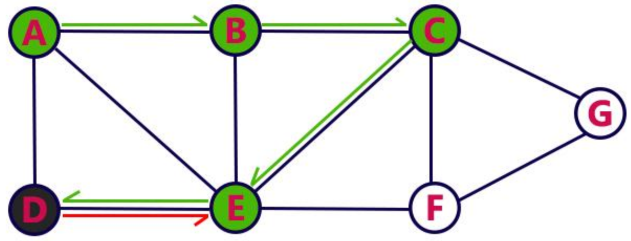
Step 5:

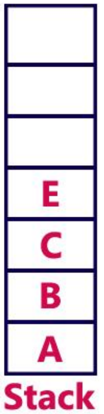
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



Step 6:

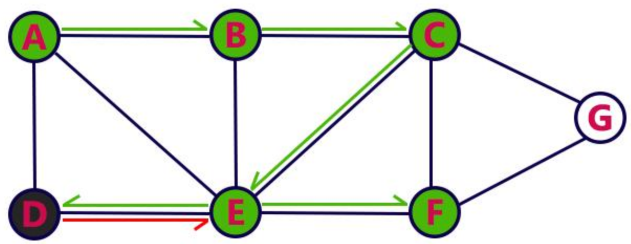
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.

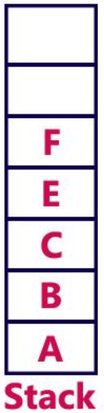




Step 7:

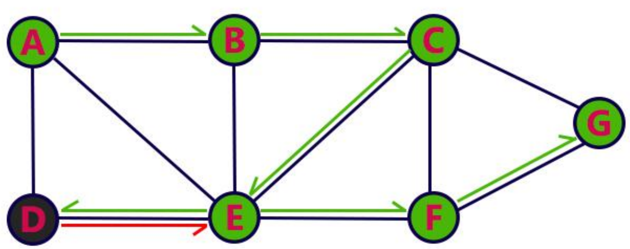
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

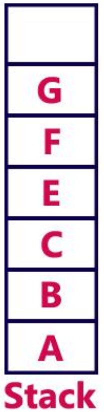




Step 8:

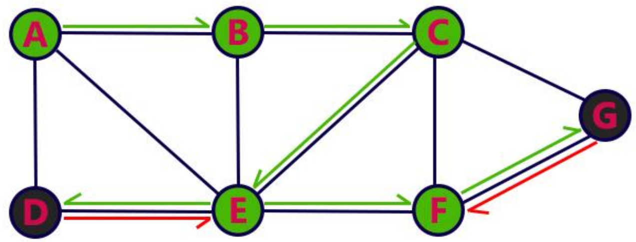
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

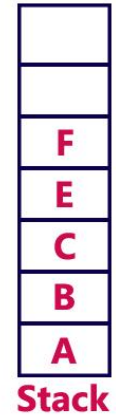




Step 9:

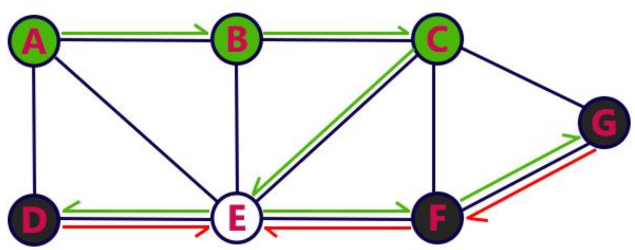
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.

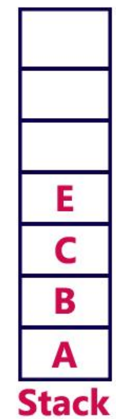




Step 10:

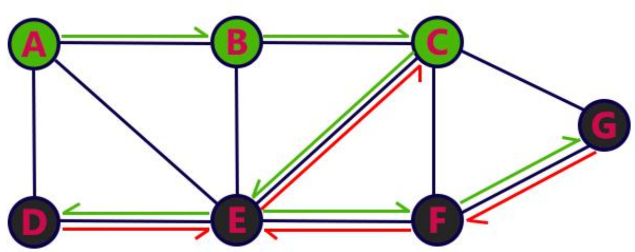
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

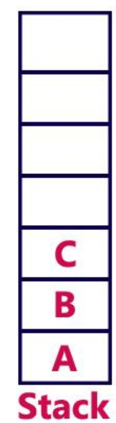




Step 11:

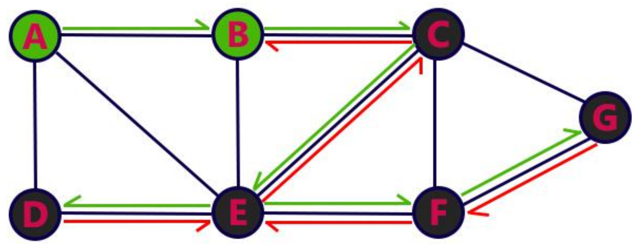
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

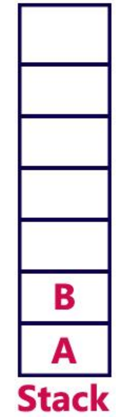




Step 12:

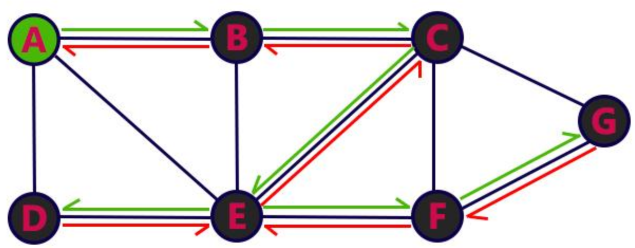
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.

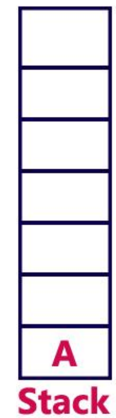




Step 13:

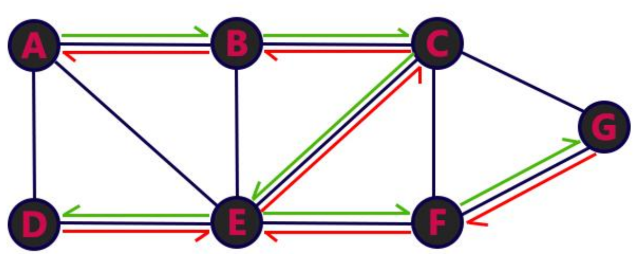
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

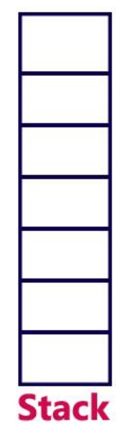




Step 14:

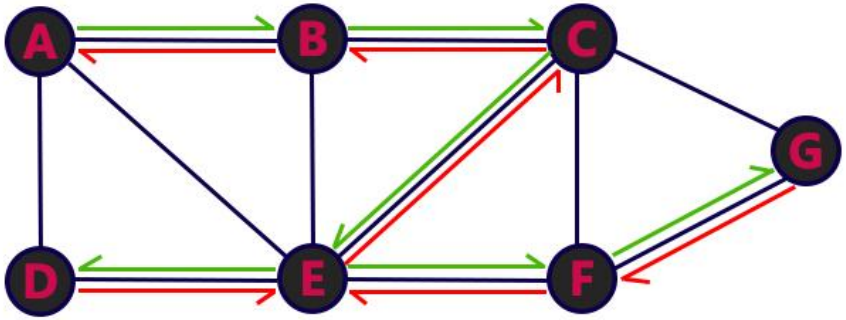
- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.

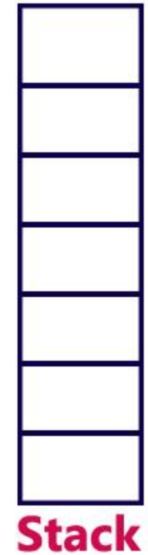




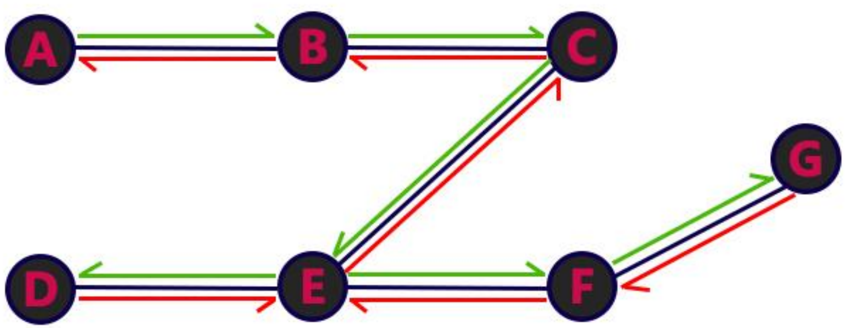
Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.





- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



پیمایش گراف- جستجوی اول سطح

- در جستجوی اول سطح پیمایش از یک راس آغاز می شود.
- آن راس و کلیه رئوس مجاورش ملاقات می شود سپس پیمایش از راس مجاور ادامه پیدا می کند.
- الگوریتم جستجوی اول سطح به صورت زیر است.
  - آرایه Visited برای تعیین رئوس ملاقات شده بکار می رود.
  - از یک صف برای نگه داشتن رئوس مجاور استفاده می شود.
  - هر بار که راسی ملاقات می شود کلیه رئوس مجاور آن در صف اضافه می شود.
  - پیمایش از راسی که از صف برداشته می شود ادامه پیدا می کند.

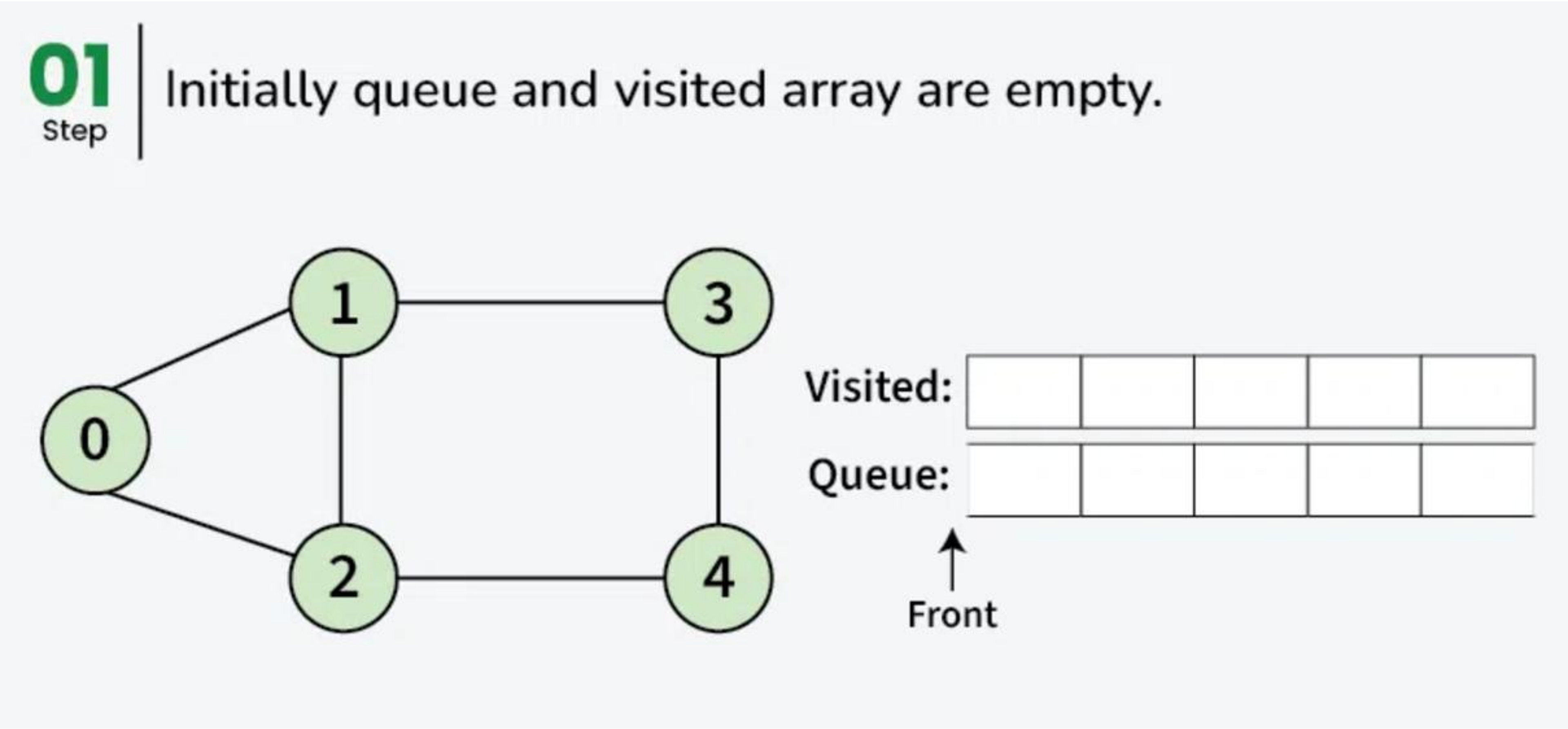
پیمایش گراف - جستجوی اول سطح

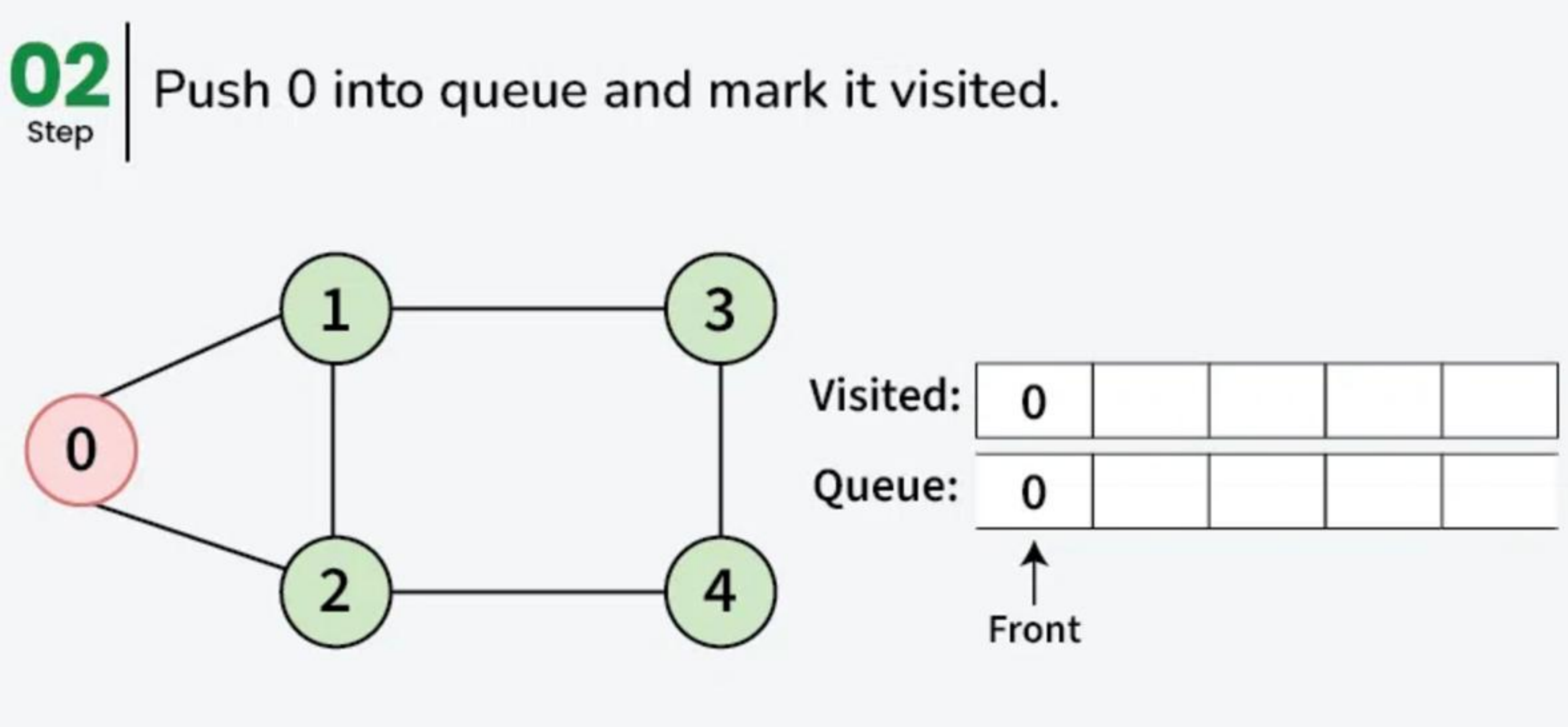
شبه کد - غیر بازگشتی

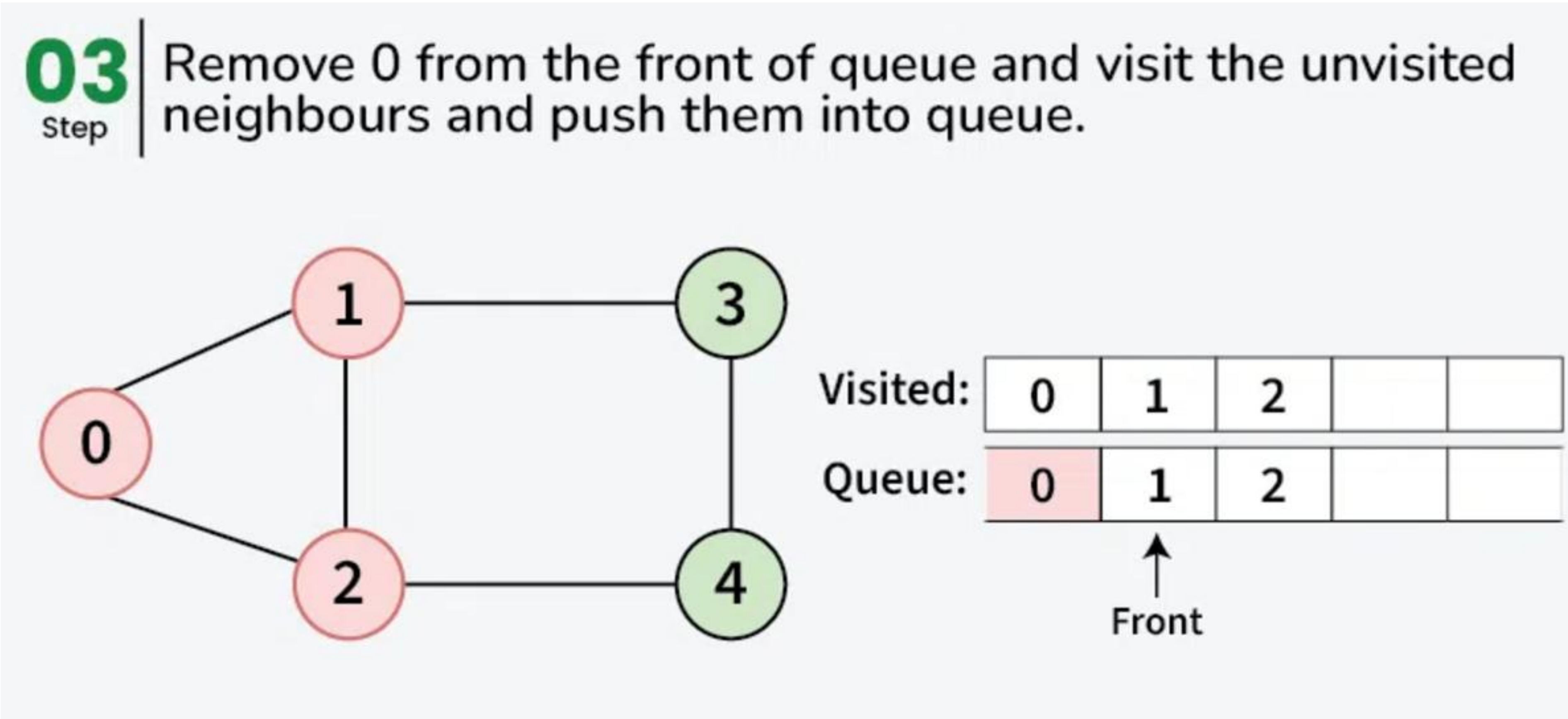
```
BFS(s)
   Mark s "discovered"
   queue = {s}
   while queue not empty
      u = remove_first(queue)
      for each edge {u,x}
         if (x is undiscovered)
            mark x discovered
            append x on queue
      mark u fully explored
```

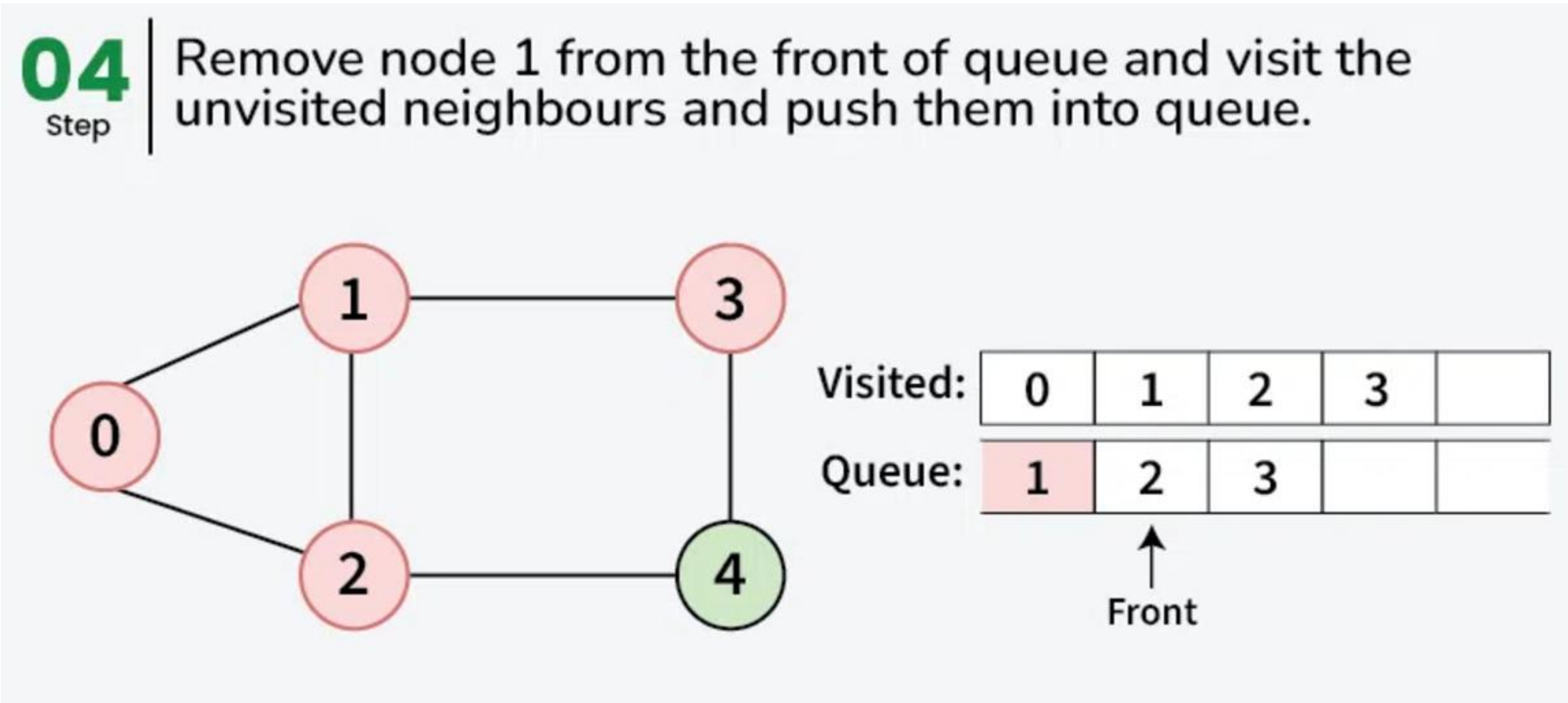
برای گراف G با n راس و m یال ، مرتبه اجرائی الگوریتم:

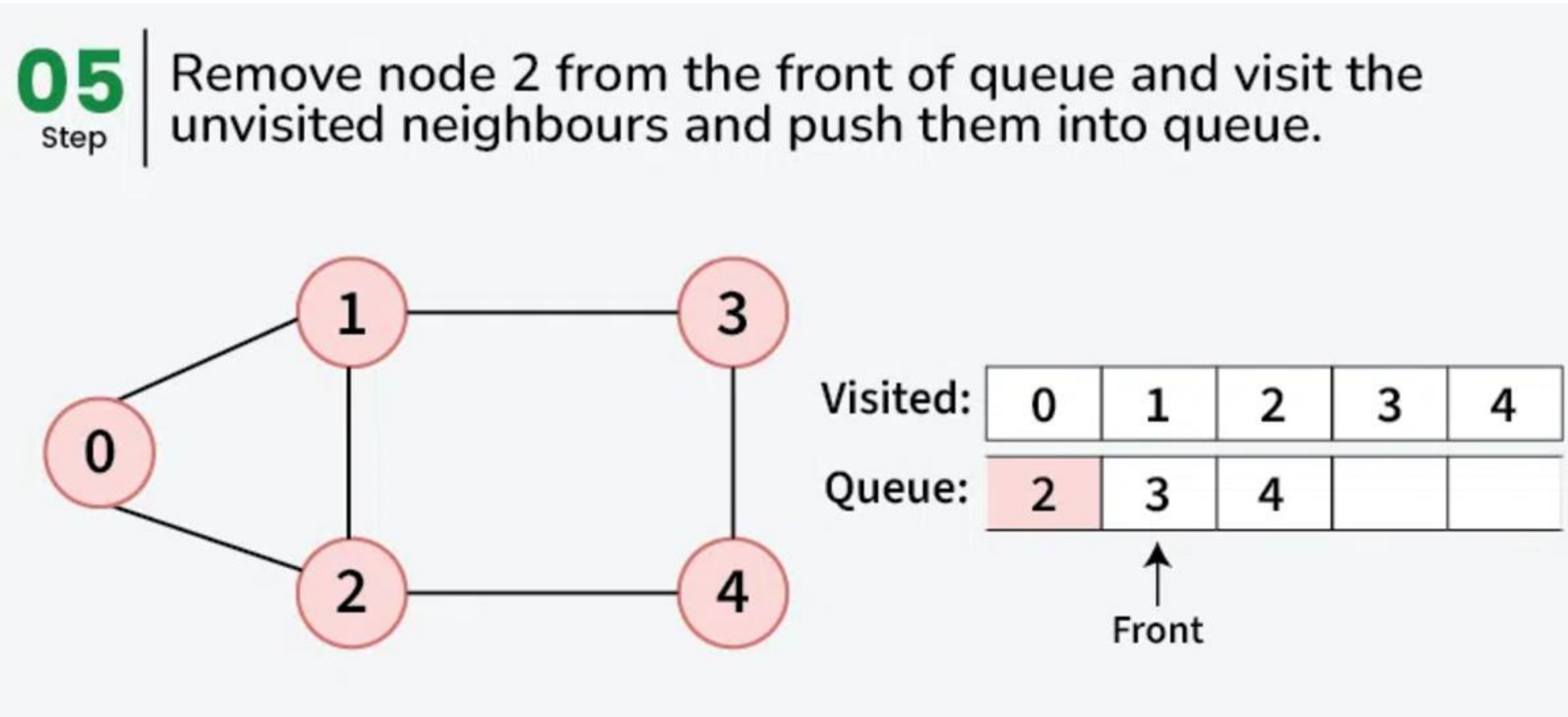
- وقتی گراف توسط ماتریس مجاورتی نمایش داده شده باشد $O(n^2)$
- اگر از لیست مجاورتی استفاده شود $O(m)$ است.

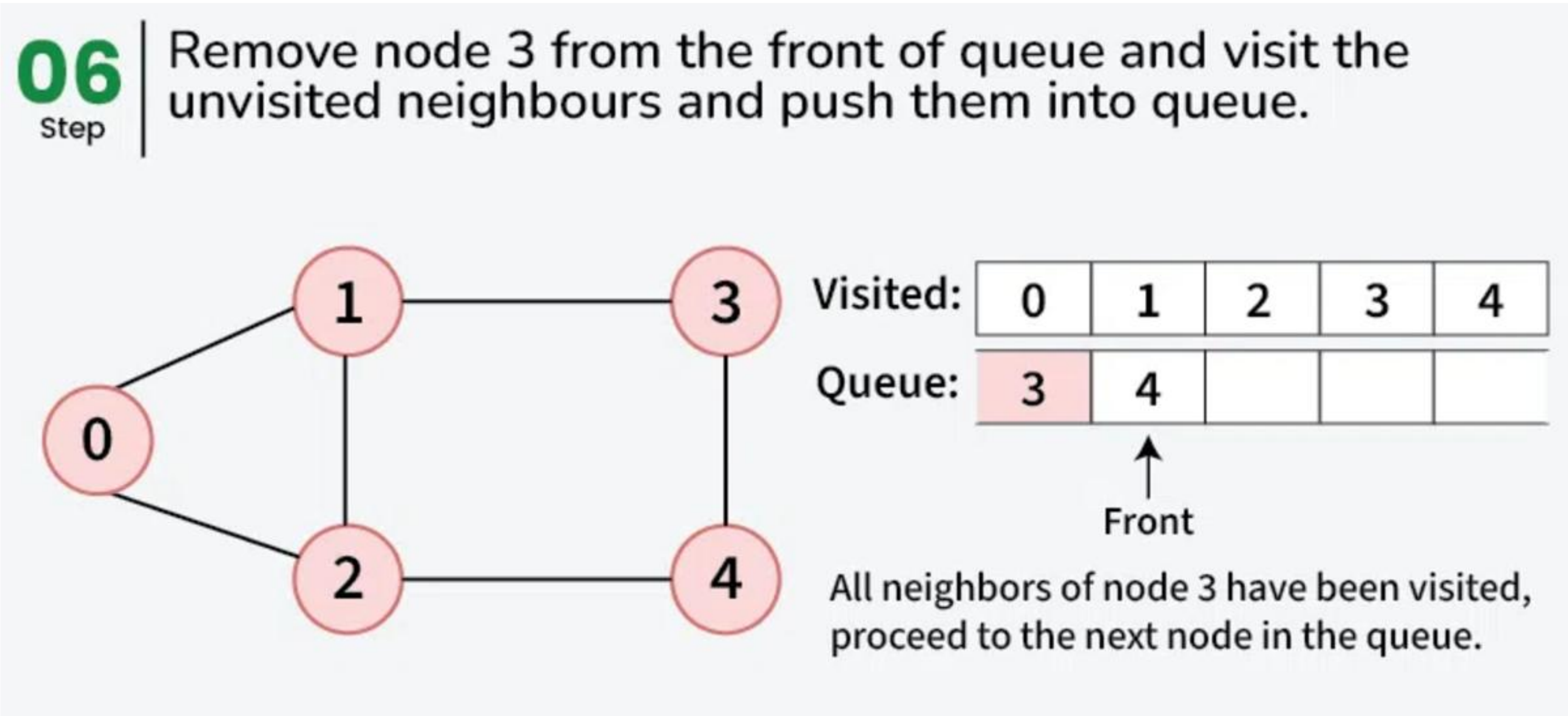

01
Step
Initially queue and visited array are empty.
1
3
0
2
4
Visited:
Queue:
Front

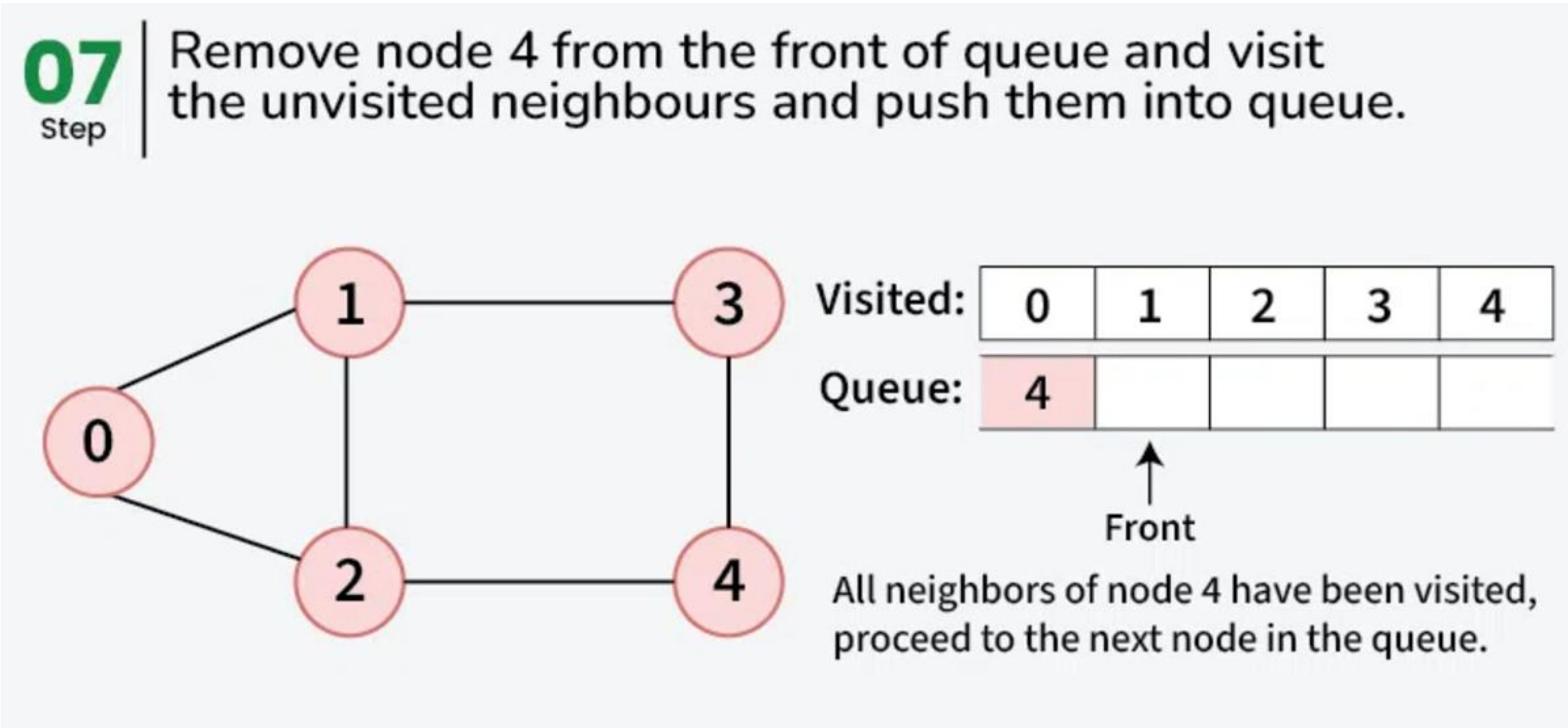

02
Step
Push 0 into queue and mark it visited.
1
3
0
2
4
Visited:
0
Queue:
0
Front


03
Step
Remove 0 from the front of queue and visit the unvisited neighbours and push them into queue.
0
1
2
3
4
Visited: 0 1 2
Queue: 0 1 2
Front


04
Step
Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.
0
1
2
3
4
Visited: 0 1 2 3
Queue: 1 2 3
Front


05
Step
Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.
0
1
2
3
4
Visited:
0
1
2
3
4
Queue:
2
3
4
Front

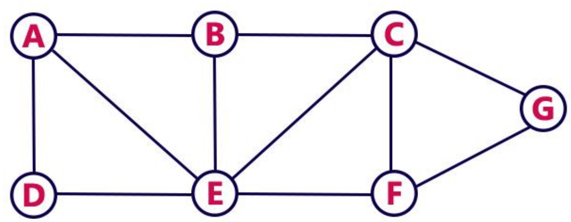

06
Step
Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.
0
1
2
3
4
Visited:
0 1 2 3 4
Queue:
3 4
Front
All neighbors of node 3 have been visited, proceed to the next node in the queue.


07
Step
Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.
1
3
0
2
4
Visited:
0 1 2 3 4
Queue:
4
Front
All neighbors of node 4 have been visited, proceed to the next node in the queue.

پیمایش گراف- جستجوی اول سطح

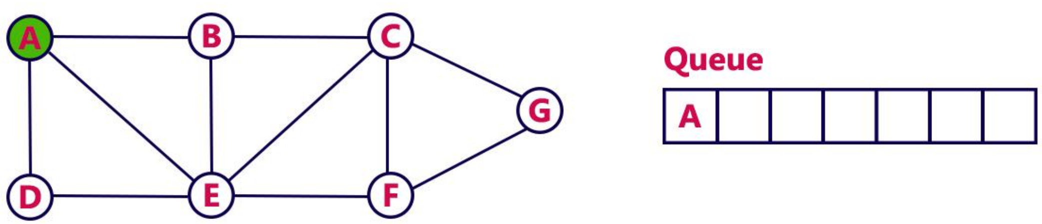
مثال۱:

Consider the following example graph to perform BFS traversal



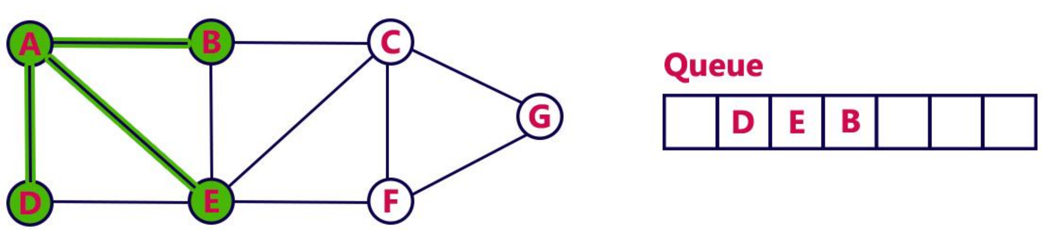
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



Step 2:

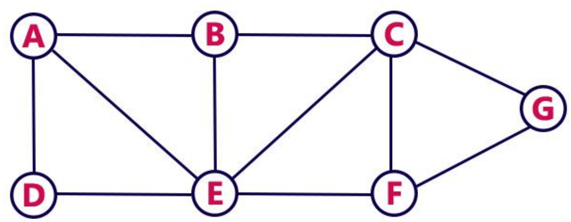
- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..



پیمایش گراف- جستجوی اول سطح

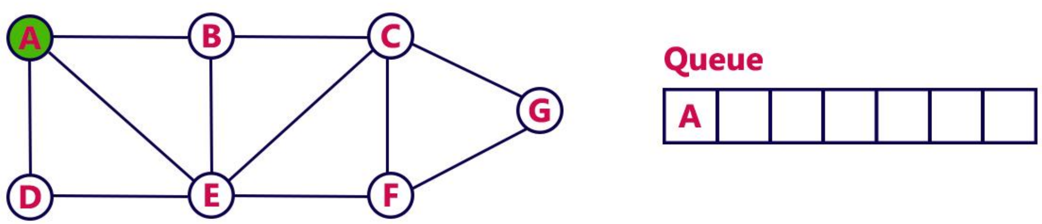
مثال۱:

Consider the following example graph to perform BFS traversal



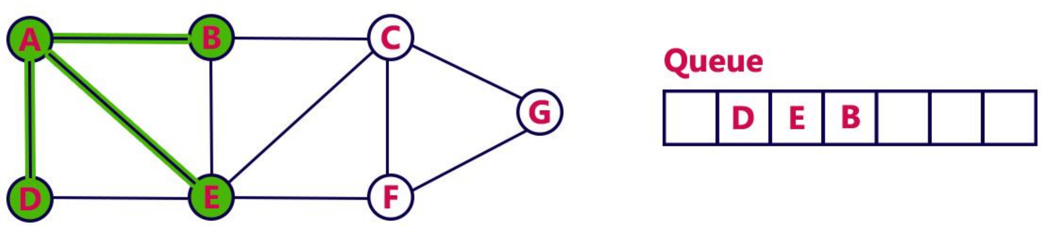
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



Step 2:

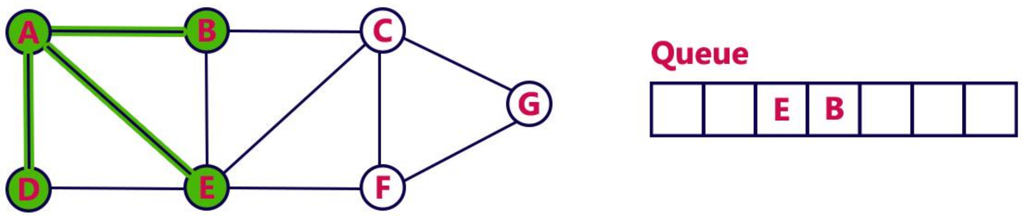
- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..



پیمایش گراف- جستجوی اول سطح

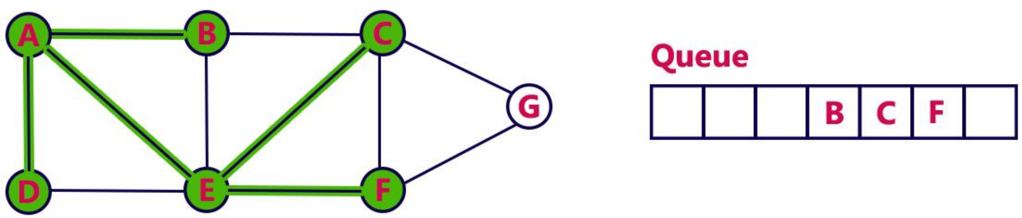
Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.



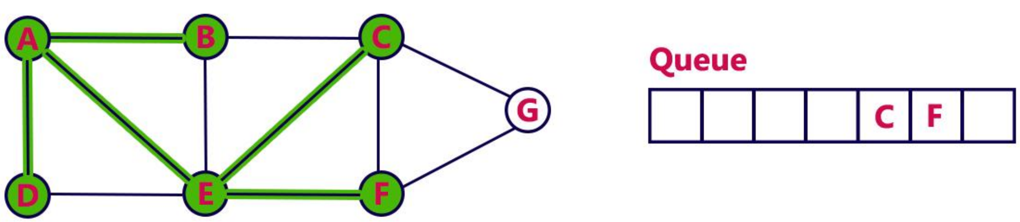
Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.



Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

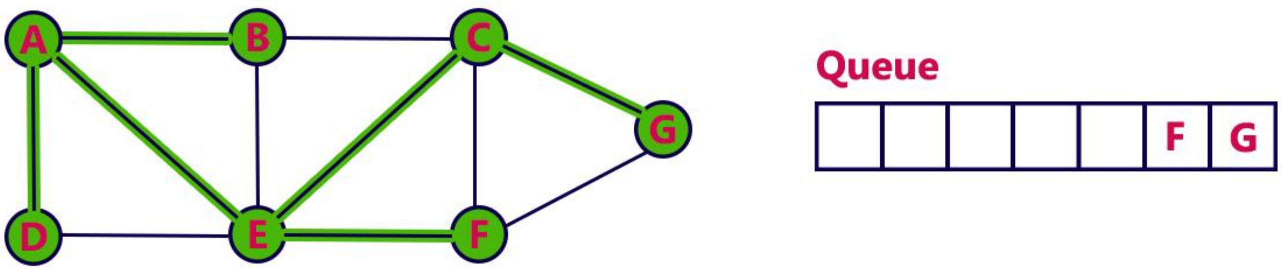


پیمایش گراف- جستجوی اول سطح

مثال۱:

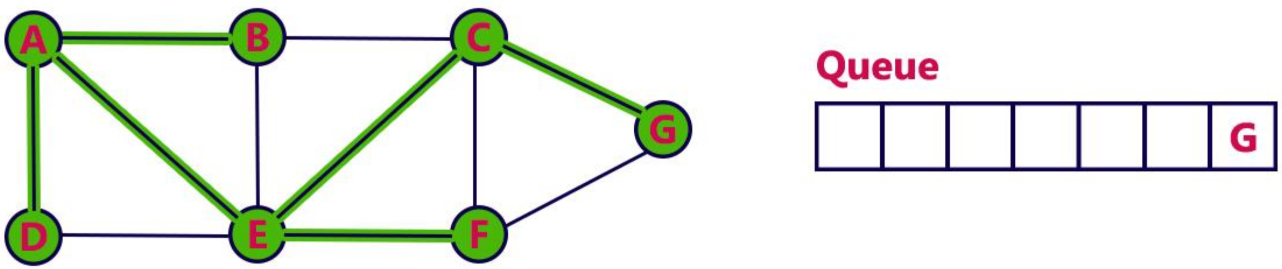
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

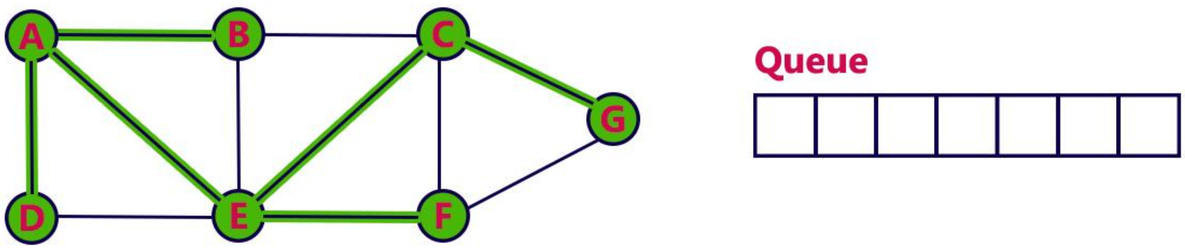


پیمایش گراف- جستجوی اول سطح

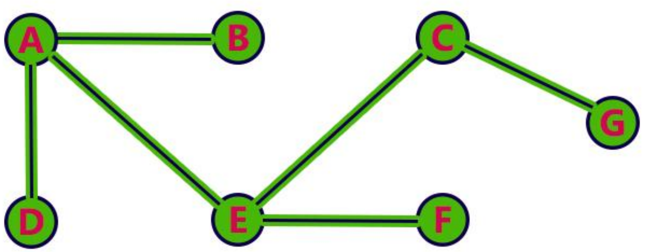
مثال۱:

Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.

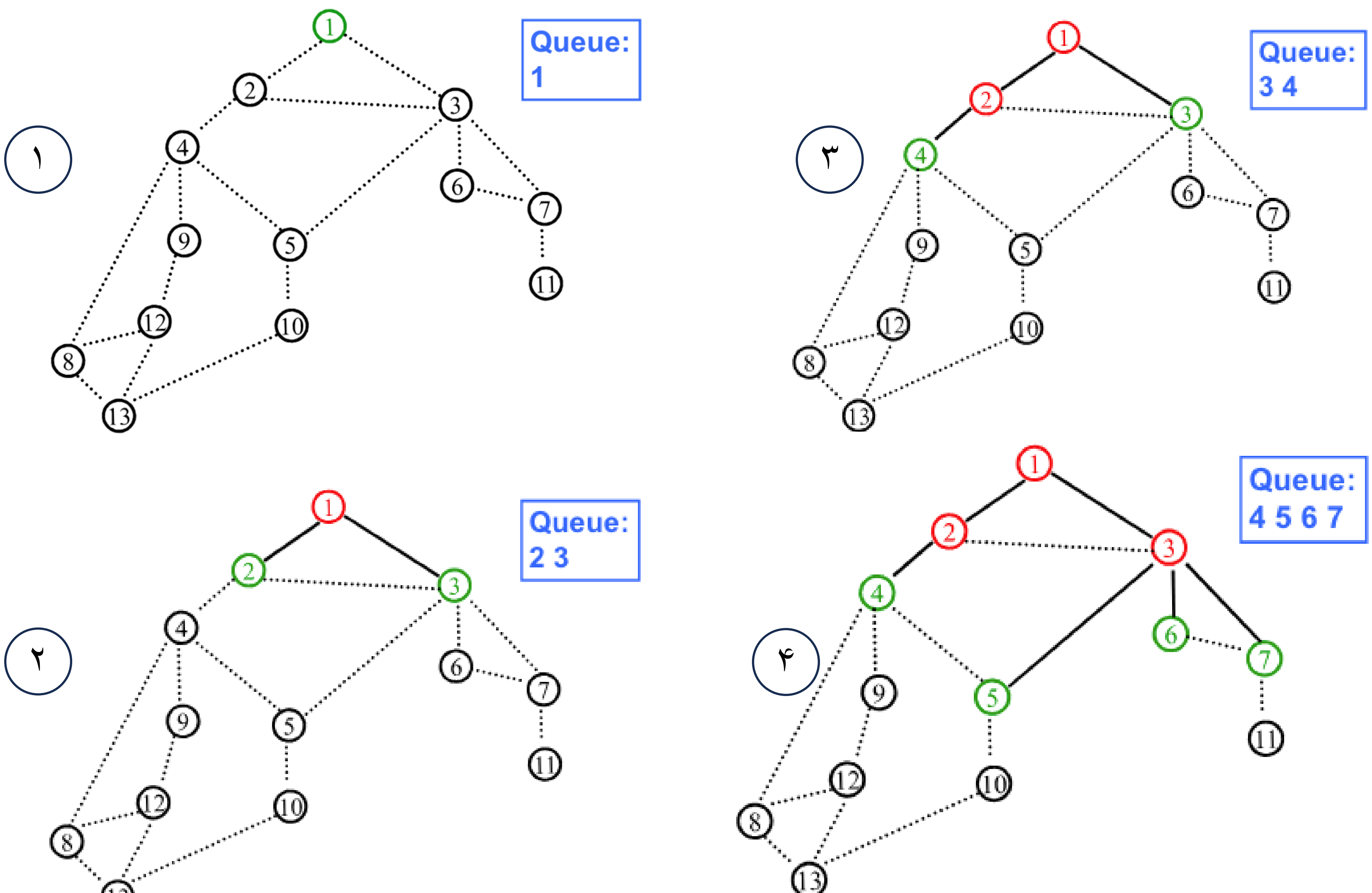


- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



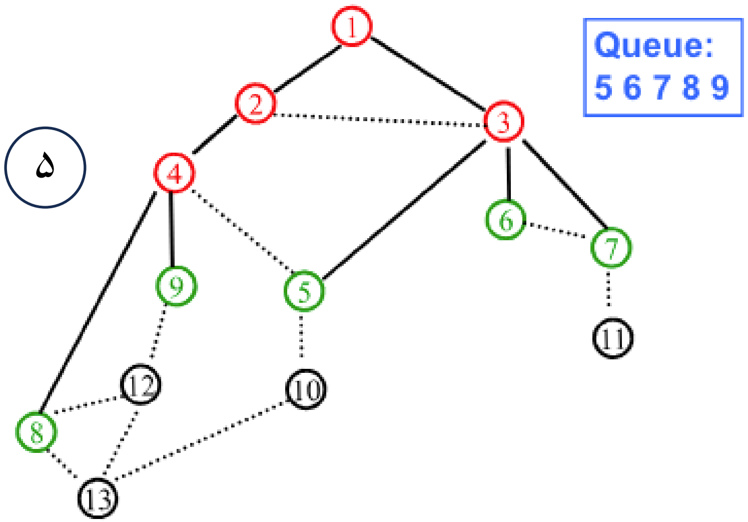
پیمایش گراف- جستجوی اول سطح

مثال۲:

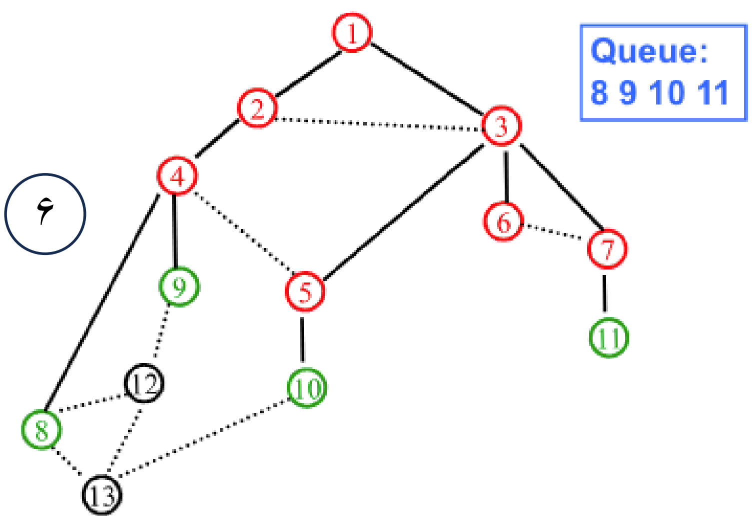


پیمایش گراف- جستجوی اول سطح

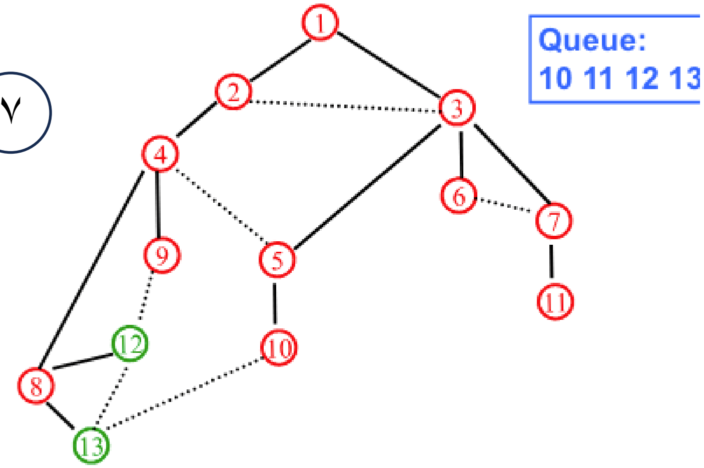
۵



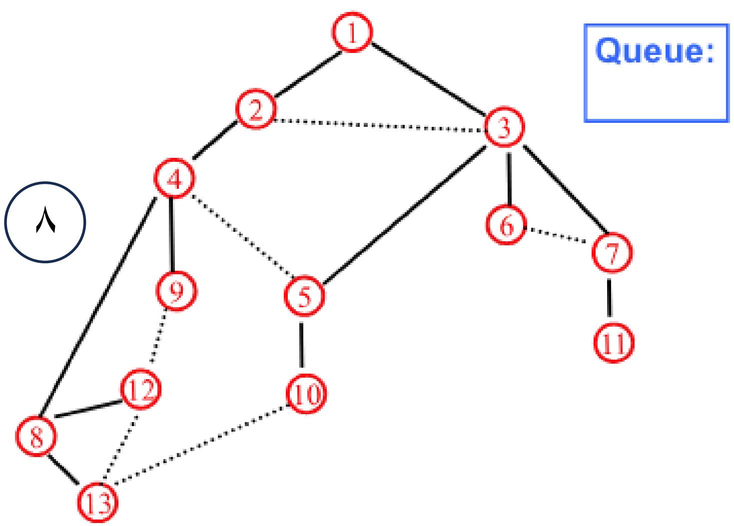
۶



۷

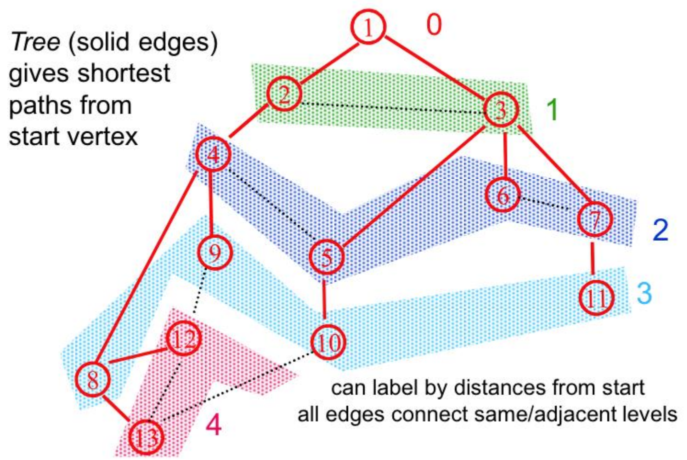


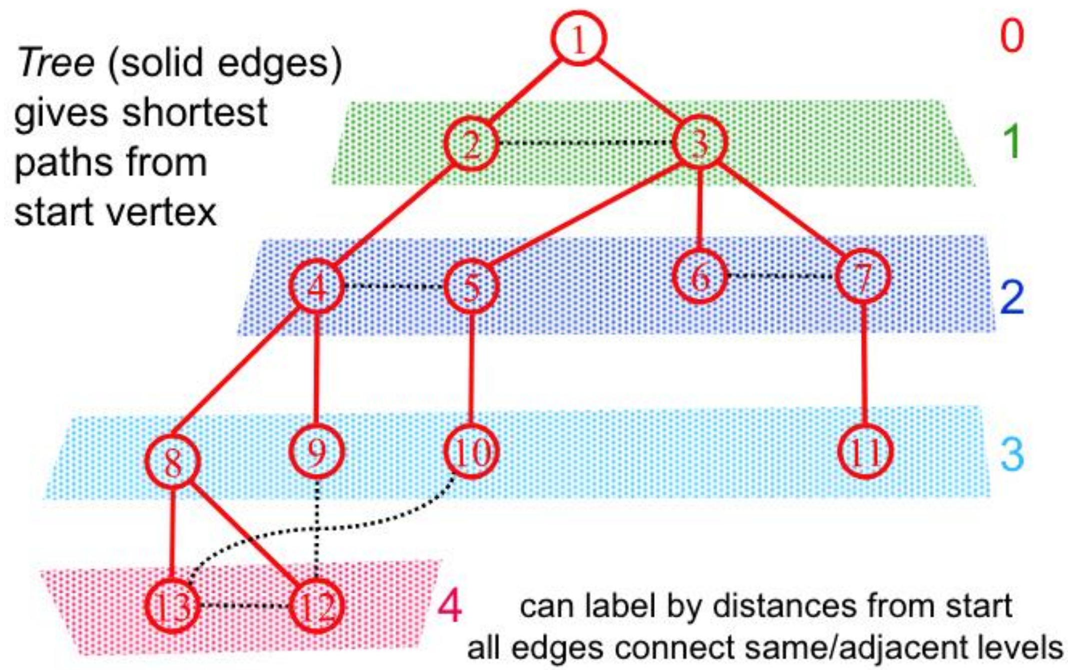
۸



پیمایش گراف- جستجوی اول سطح

در نهایت این پیمایش یک درخت جستجو با کمترین فاصله به هر نود را بر می گرداند.
هر سطح از فرزندان سطح قبلی به دست می آیند.



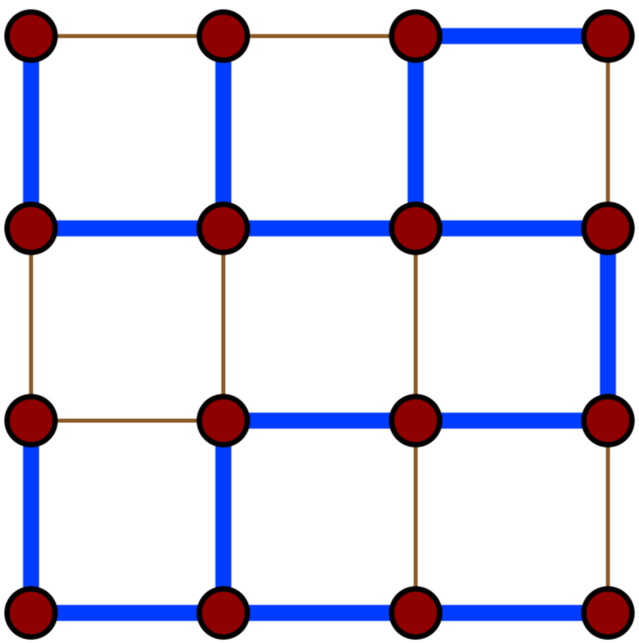


درخت پوشا

- در نظریه گراف، یک درخت پوشا با نام T از گراف همبند و بدون جهت G، درختی است که شامل تمام رئوس و حداقل برخی یال‌ها می‌باشد.
- به بیان ساده‌تر می‌توان گفت، درخت پوشا از گراف G، درختی است که مجموعه‌ای از یال‌ها را شامل می‌شود در حالی که تمام رئوس را پوشش می‌دهد.
- در واقع تمام رئوس G در درخت پوشا وجود دارند به شرطی که هیچ دوری ایجاد نشود و درخت همبند (متصل) نیز باشد.
- درخت پوشای گراف همبند G را می‌توان اینگونه نیز تعریف کرد:
  - مجموعه‌ای حداکثری از یال‌های G که هیچ دوری در آن وجود ندارد، و یا
  - مجموعه‌ای حداقلی از یال‌های G که همهٔ رئوس را به یکدیگر متصل می‌کند.

درخت پوشا

- در شکل زیر گراف متصل G (رنگ قرمز) به یک درخت (رنگ آبی) تبدیل شده است.
- درخت تمامی رئوس را شامل می باشد، اما فقط برخی از یال ها وجود دارند.



یک درخت پوشا، یال‌های آبی رنگ تشکیل یک درخت پوشا می‌دهند.

- پیمایش‌های DFS و BFS هر کدام یک درخت پوشا تولید می کنند.

درخت پوشا مینیمم

- درخت پوشای مینیمم Minimum Spanning Tree از گراف وزن دار G، درخت پوشایی است که مجموع وزن های آن حداقل باشد.
- برای بدست آوردن درخت پوشای حداقل دو الگوریتم الگوریتم کروسکال Kruskal و الگوریتم پریم Prim وجود دارد.

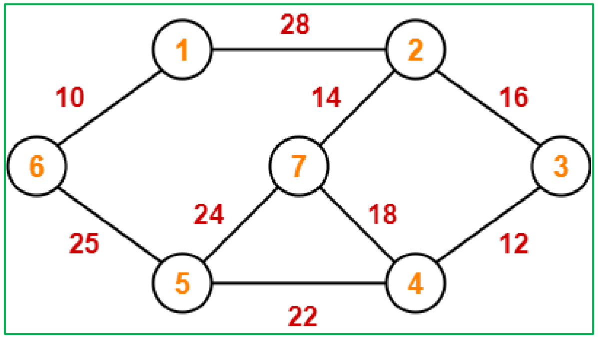
درخت پوشا مینیمم- الگوریتم کروسکال

- گراف G با n راس را در نظر بگیرید. به صورت زیر عمل می کند:

1. تمام یال ها را به طور صعودی بر حسب وزن مرتب کنید.
2. درخت T را متشکل از گره های G بدون یال را ایجاد کنید.
3. عملیات زیر را n-1 بار تکرار کنید:
4. یک یال با حداقل وزن را به درخت T اضافه کنید به طوریکه حلقه ایجاد نشود.

- گاهی چند یال دارای یک وزن هستند، در این حالت ترتیب یال هایی که انتخاب می شوند مهم نیست. درخت های پوشای حداقل مختلفی ممکن است حاصل شود اما مجموع وزن آنها همیشه یکسان و حداقل می شود.

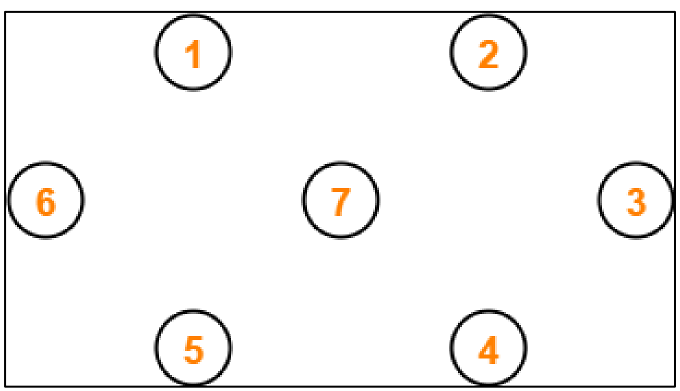
الگوریتم کروسکال



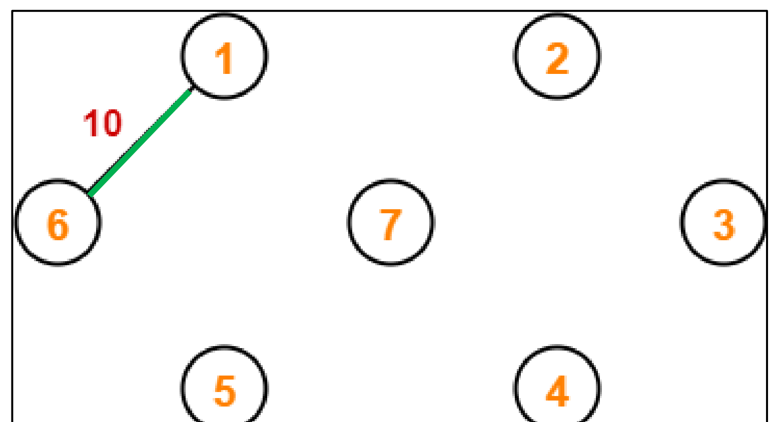
وزن	رتبه
۱۰	۱
۱۲	۲
۱۴	۳
۱۶	۴
۱۸	۵
۲۲	۶
۲۴	۷
۲۵	۸
۲۸	۹

❑ Kruskal

- در هر مرحله یال با وزن کمتر انتخاب می شوند
- به شرطی که دور پیش نیاید

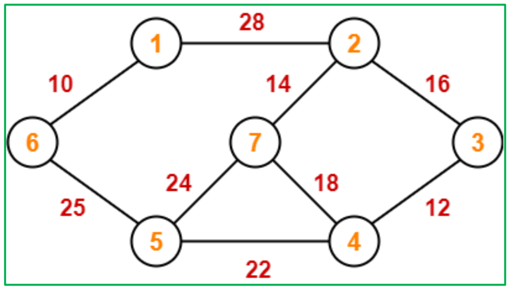


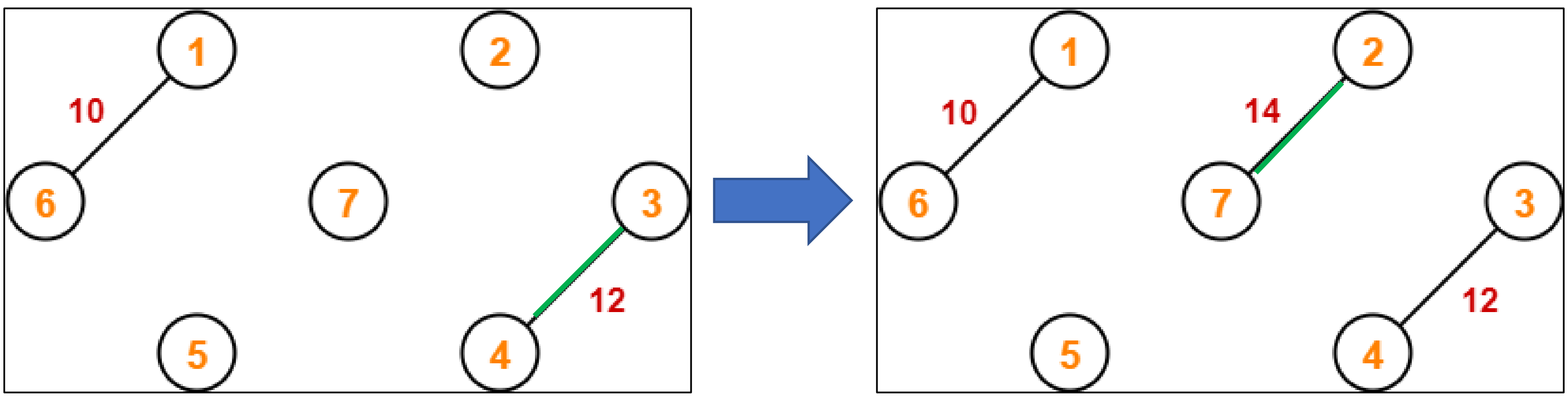
یال با وزن ۱۰
اضافه می شود



الگوریتم کروسکال

❑Kruskal



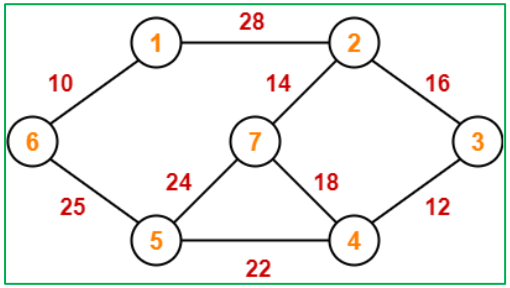


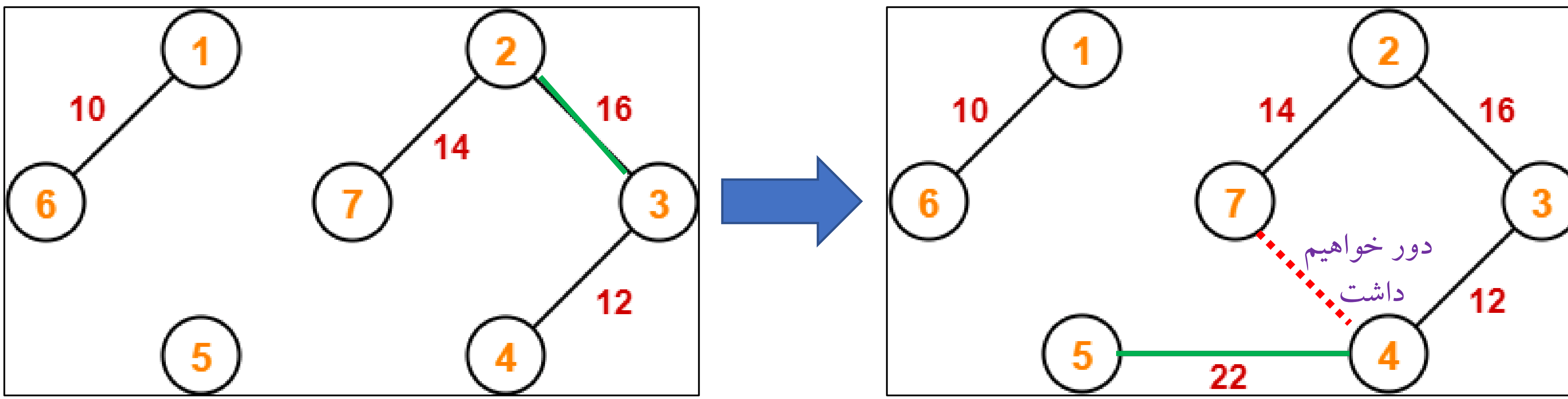
یال با وزن ۱۲
اضافه می شود

یال با وزن ۱۴
اضافه می شود

الگوریتم کروسکال

- Kruskal





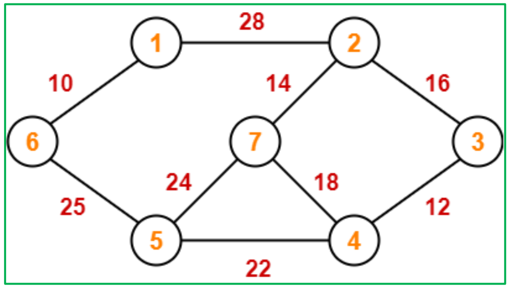
یال با وزن ۱۶ اضافه می شود

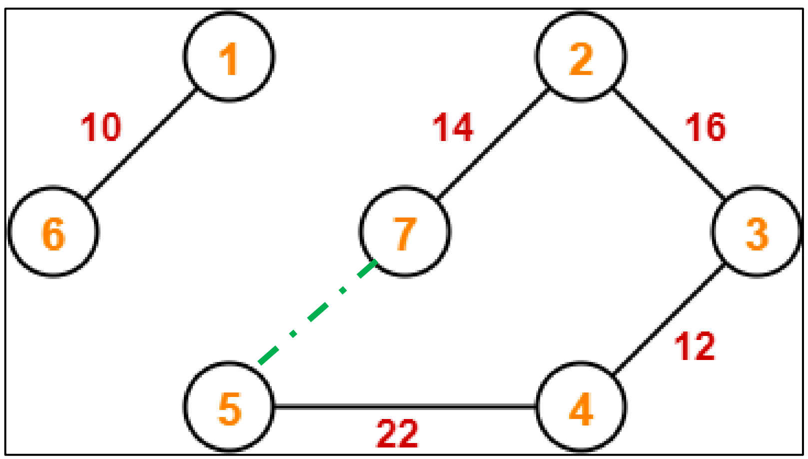
ابتدا یال با وزن ۱۸ اضافه می شود، دور داریم، پس حذف می شود

بعد یال با وزن ۲۲ اضافه می شود، دور نداریم

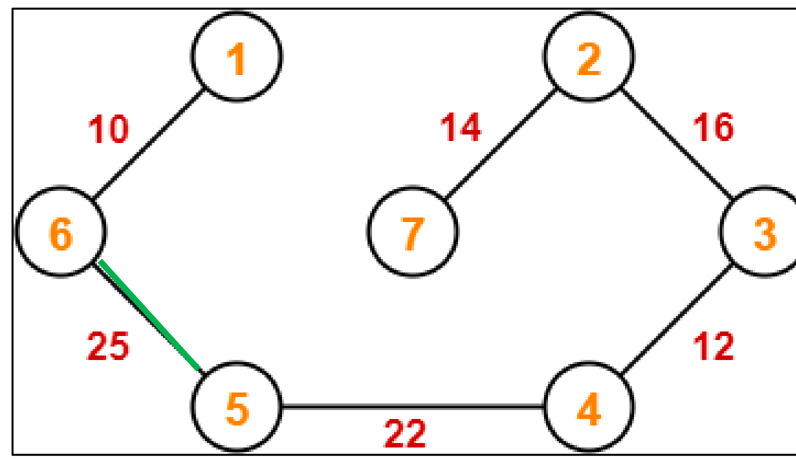
الگوریتم کروسکال

❑ Kruskal





اگر یال با وزن ۲۴ اضافه
شود دور خواهیم داشت



یال با وزن ۲۵ اضافه شود
دور نخواهیم داشت

درخت پوشای مینیمم – الگوریتم پریم

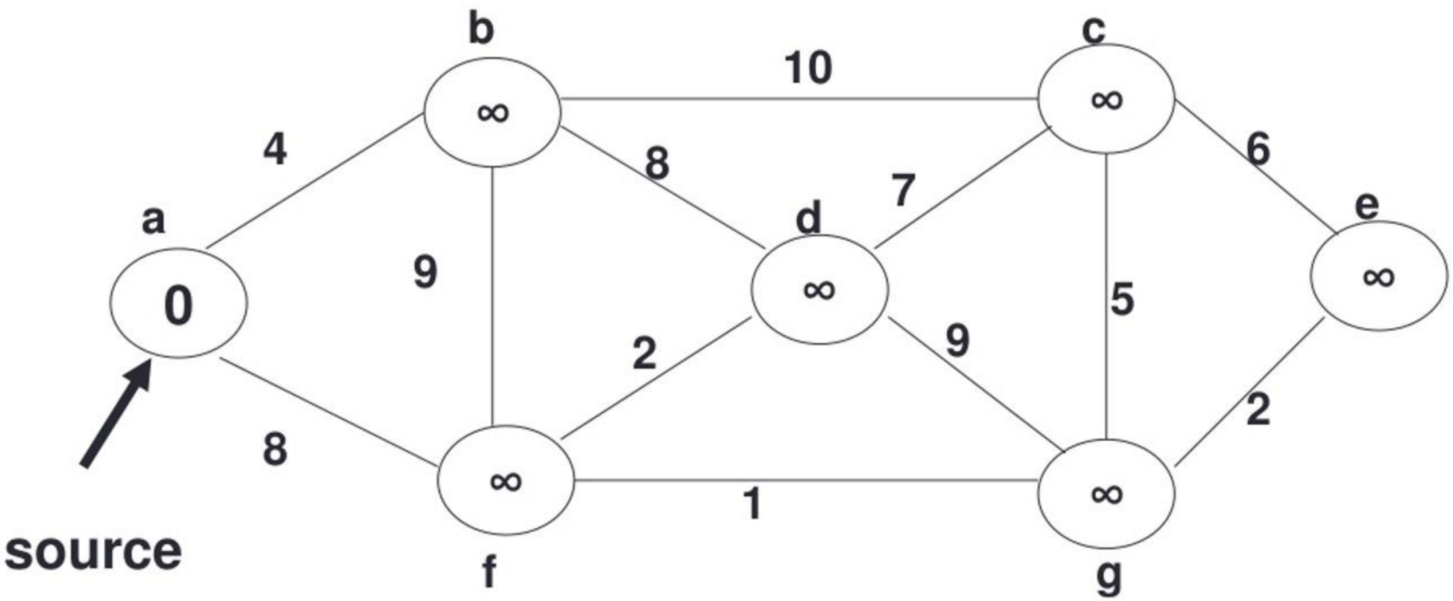
- گراف G با n راس را در نظر بگیرید. الگوریتم پریم به صورت زیر عمل می کند:

1. درخت تهی Tرا ایجاد کنید.
2. راس v (دلخواه یا داده شده) از گراف را انتخاب کرده و به درخت اضافه کنید.
3. عملیات زیر را تکرار کنید تا کلیه راس های گراف به درخت T اضافه شوند:

۴. یالی که با حداقل وزن به رئوس T متصل است را پیدا کنید. یال و راس متصل به آن را به درخت T اضافه کنید به طوریکه حلقه ایجاد نشود.

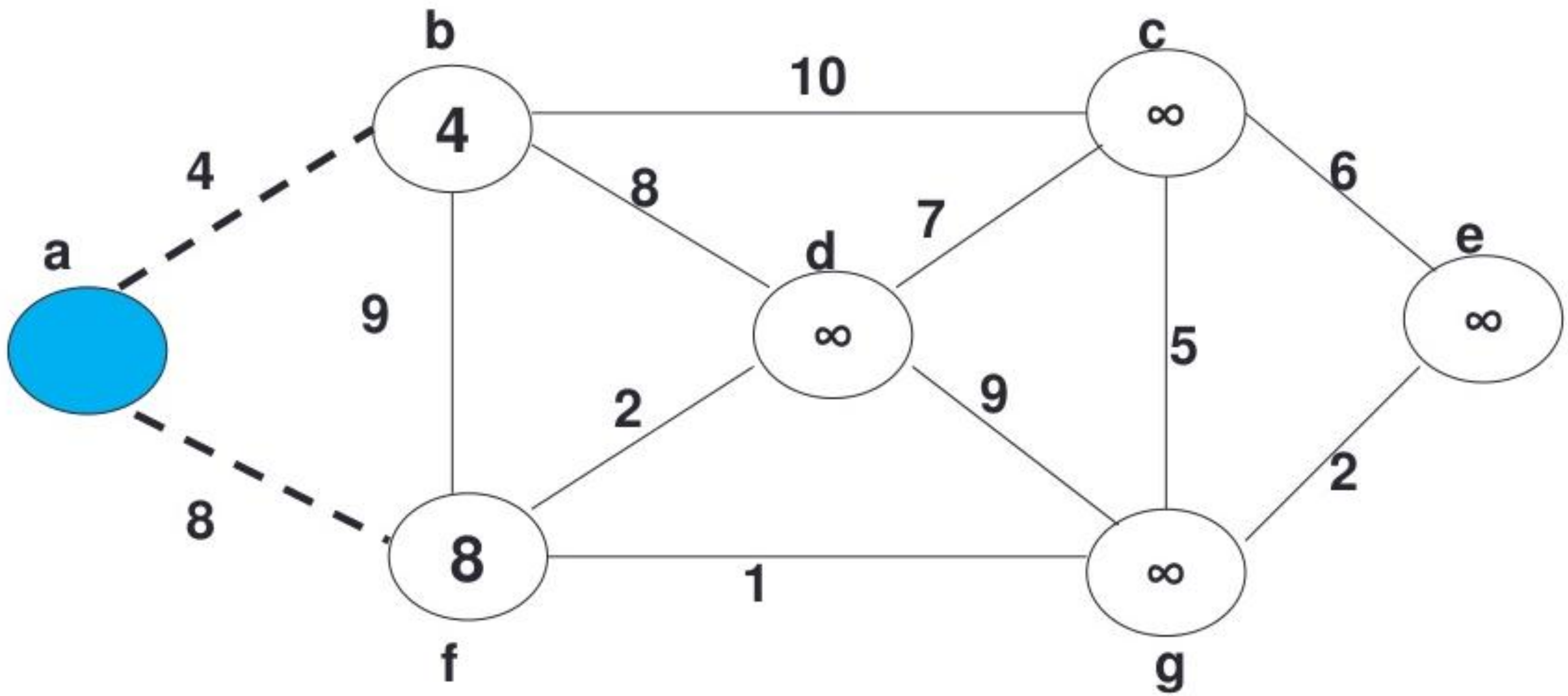
پیچیدگی زمانی الگوریتم $O(mn)$می شود، که m تعداد یال ها و n تعداد رئوس گراف G است.

درخت پوشای مینیمم – الگوریتم پریم



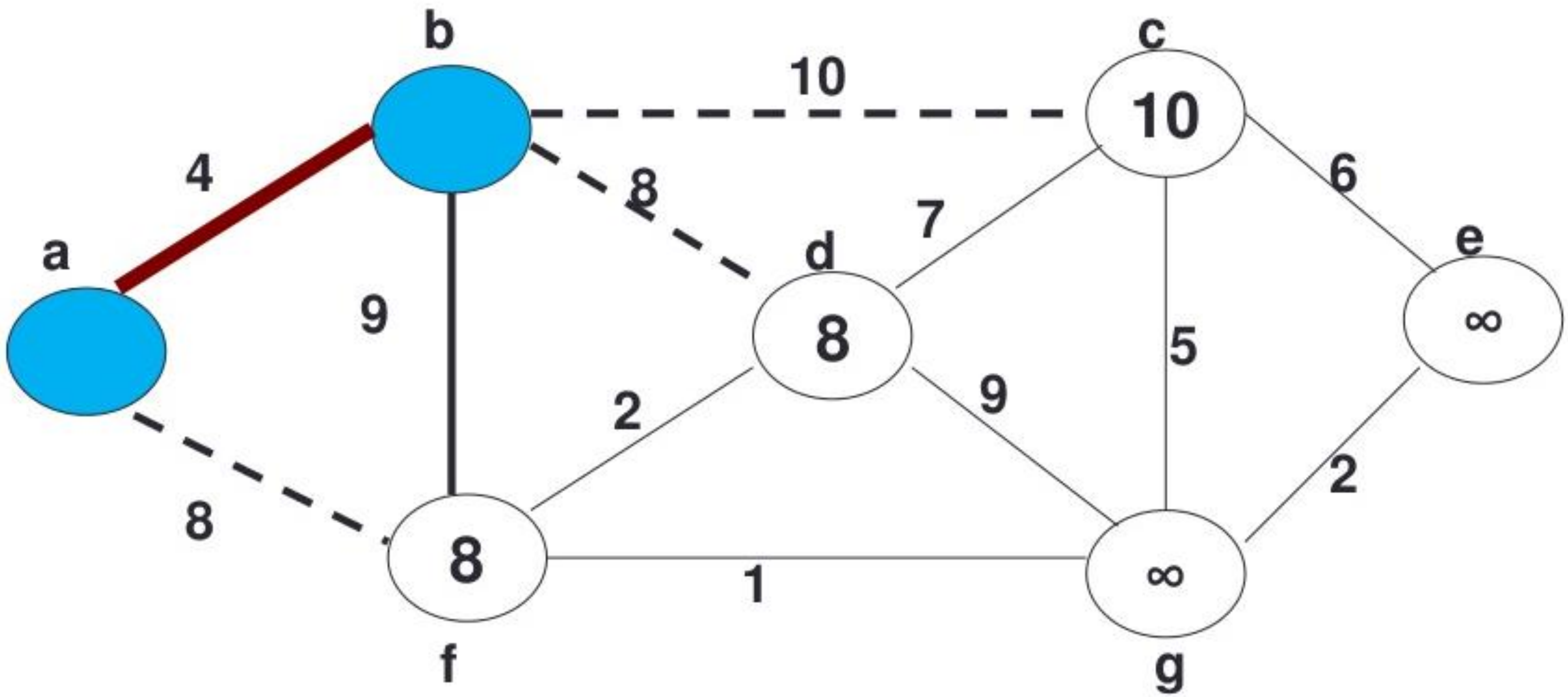
الگوریتم پریم

- در واقع با اضافه شدن هر نود (گره)، یال های متصل به آن هم به عنوان کاندیداهای ممکن به یال های کاندید قبلی اضافه می شود. (که با خط چین نمایش داده شده است)
- حال از بین تمام این یال ها، یال با کمترین هزینه انتخاب و به درخت اضافه می شود.



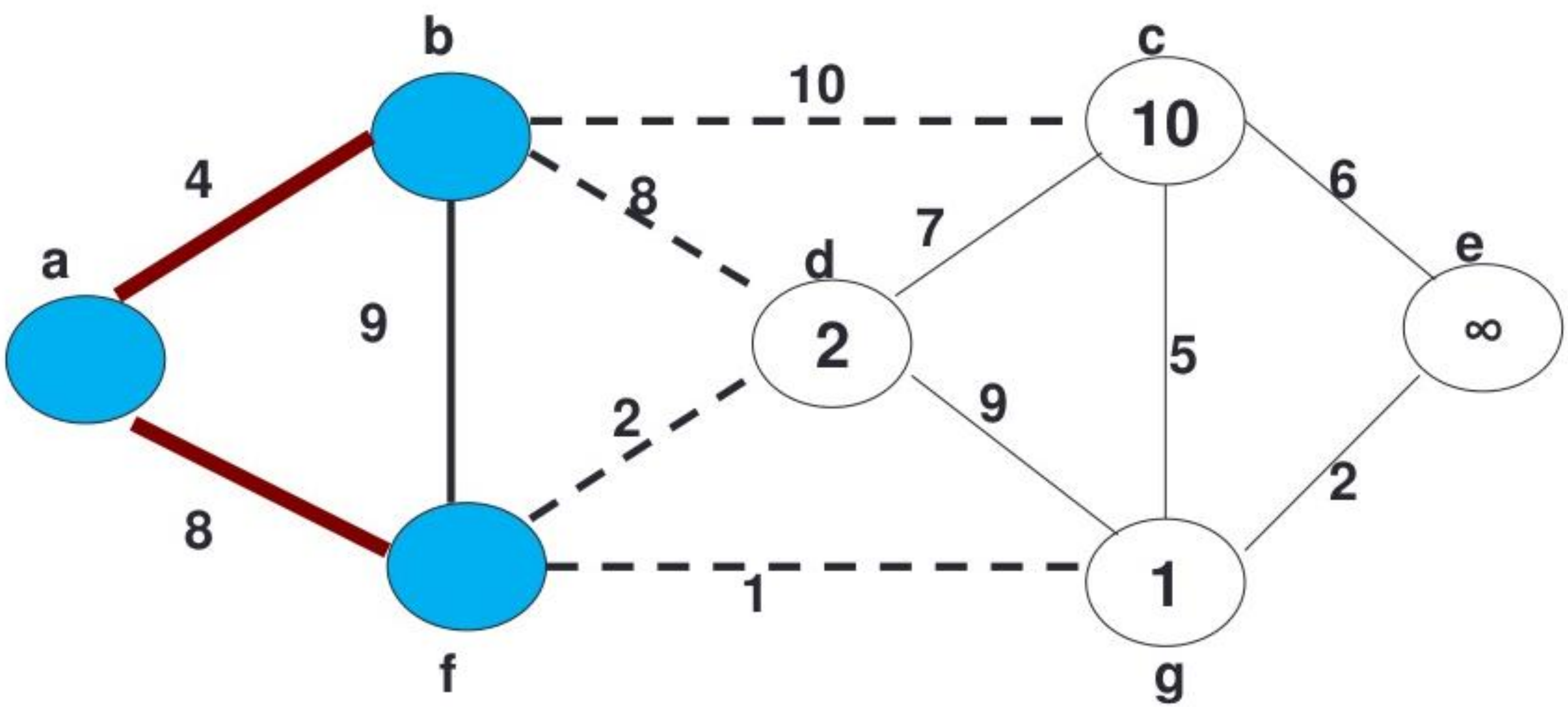
الگوریتم پریم

- حال با اضافه شدن نود b، یال های متصل به آن (با وزن های ۸ ، ۱۰) هم به عنوان کاندیداهای ممکن به یال های کاندید قبلی اضافه می شود
- حال از بین تمام این یال ها، یال با کمترین هزینه انتخاب و به درخت اضافه می شود.



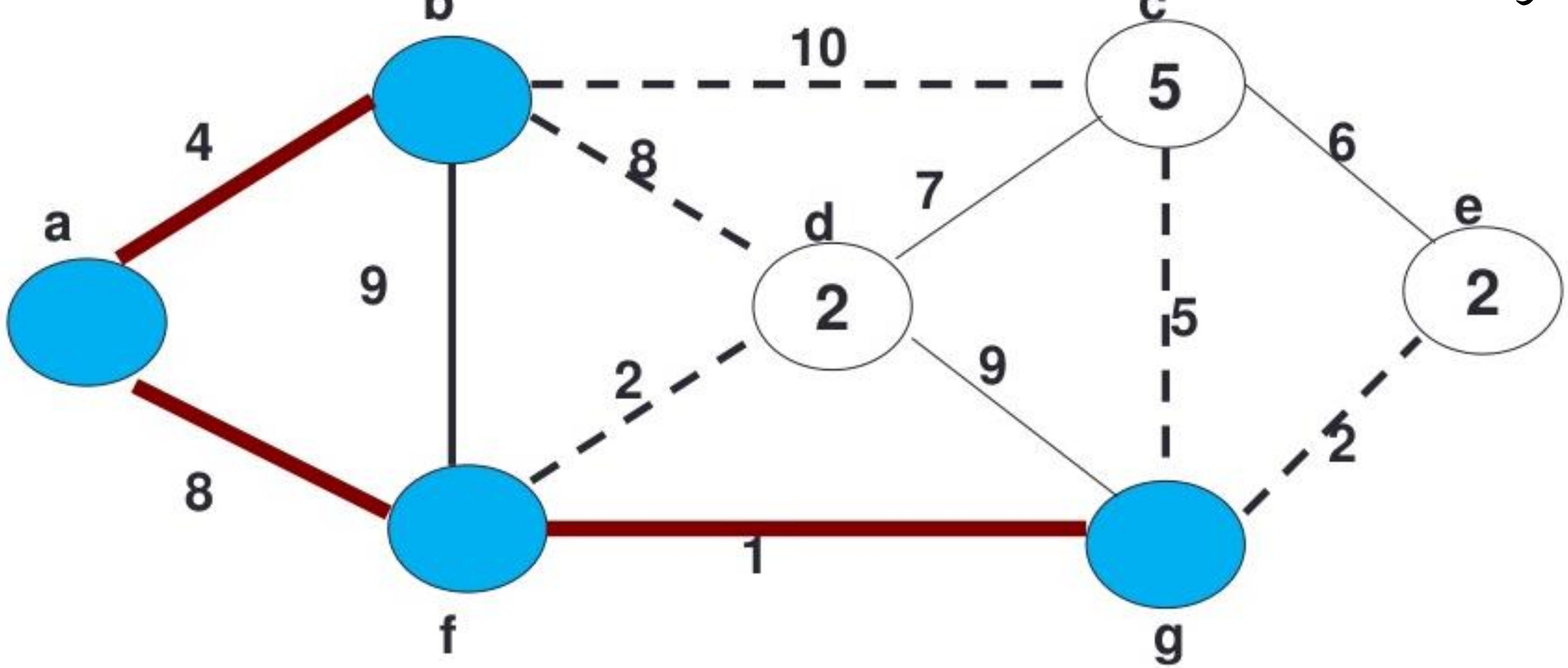
الگوریتم پریم

- حال با اضافه شدن نود f، یال های متصل به آن (با وزن های ۱ ، ۲) هم به عنوان کاندیداهای ممکن به یال های کاندید قبلی اضافه می شود
- حال از بین تمام این یال ها، یال با کمترین هزینه انتخاب (یال با وزن ۱) به درخت اضافه می شود.

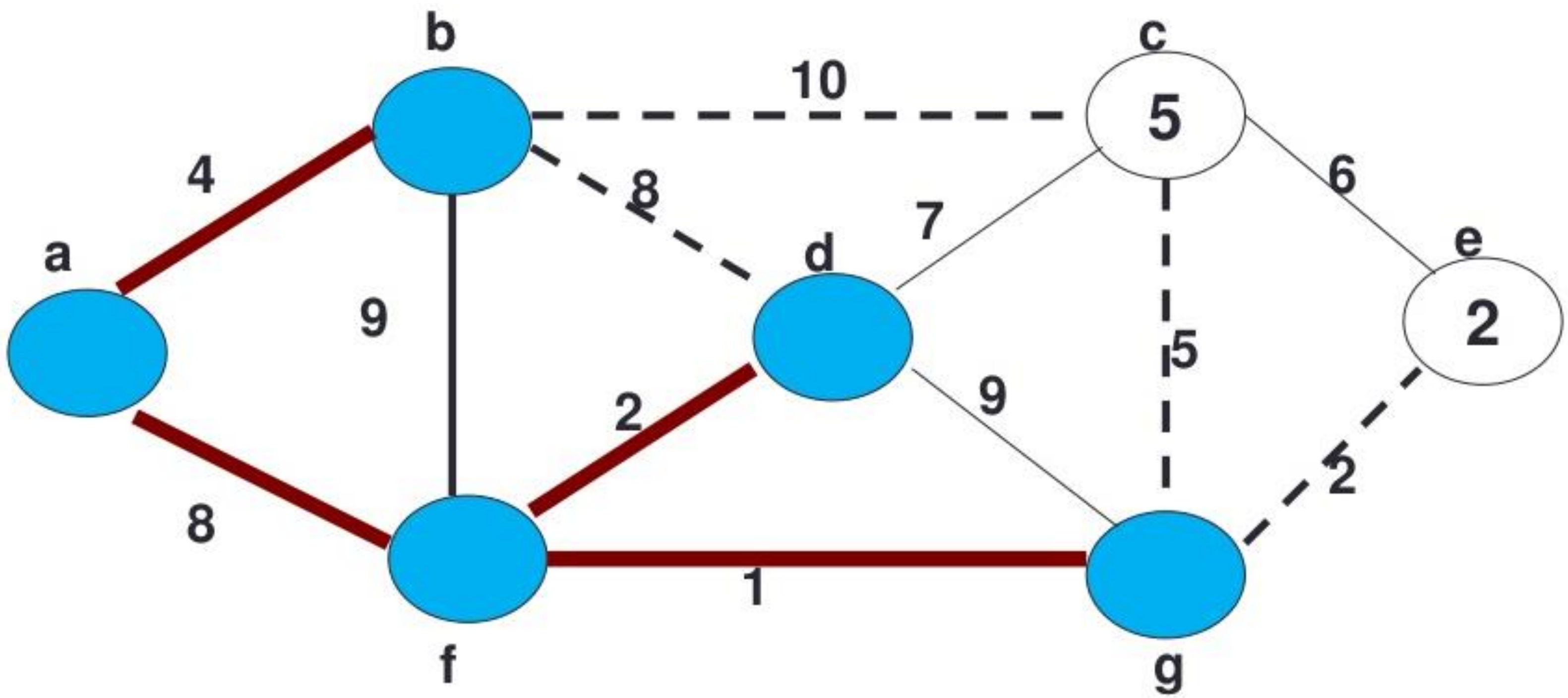


الگوریتم پریم

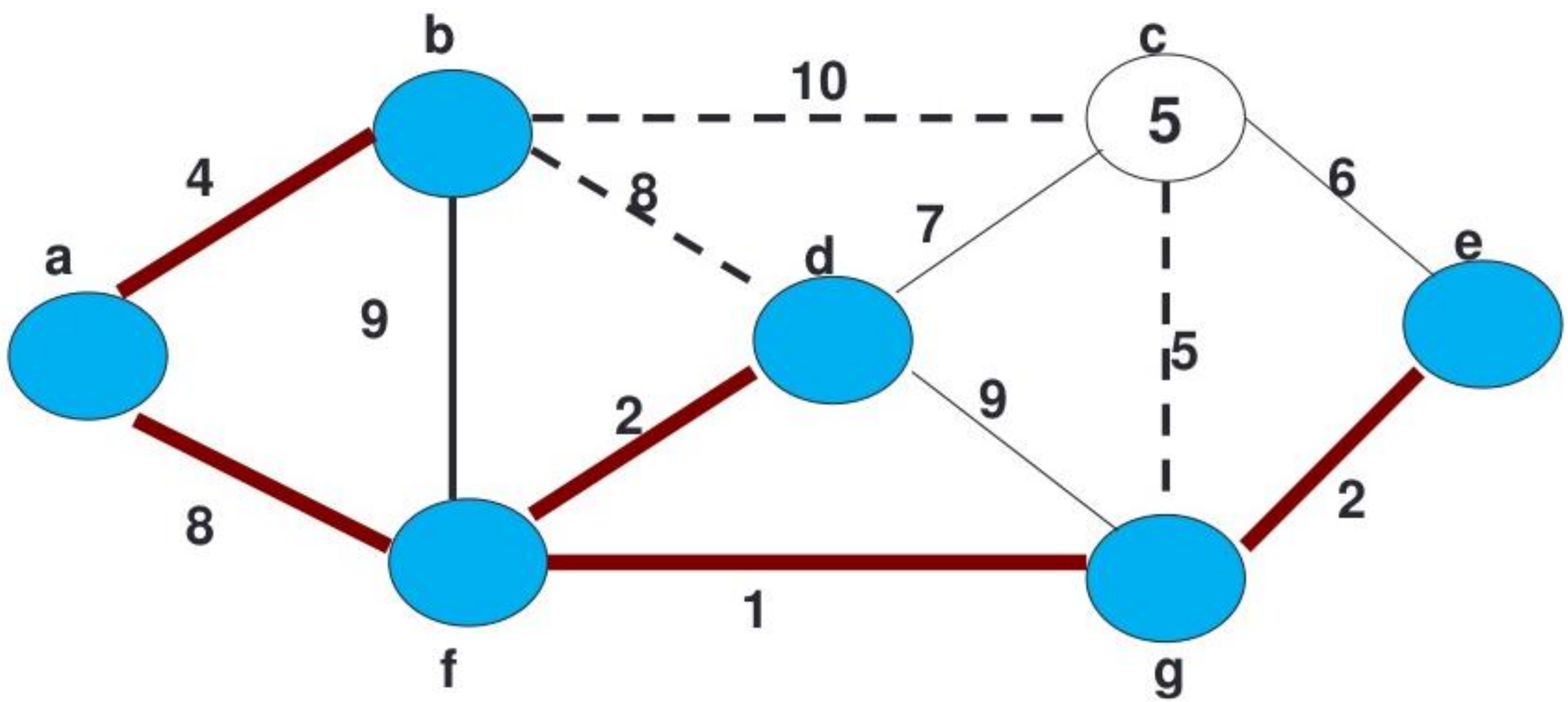
- حال با اضافه شدن نود g، یال های متصل به آن (با وزن های ۲ ، ۵ و ۹) هم به عنوان کاندیداهای ممکن به یال های کاندید قبلی اضافه می شود
- حال از بین تمام این یال ها، یال با کمترین هزینه انتخاب (یال با وزن ۲) به درخت اضافه می شود.



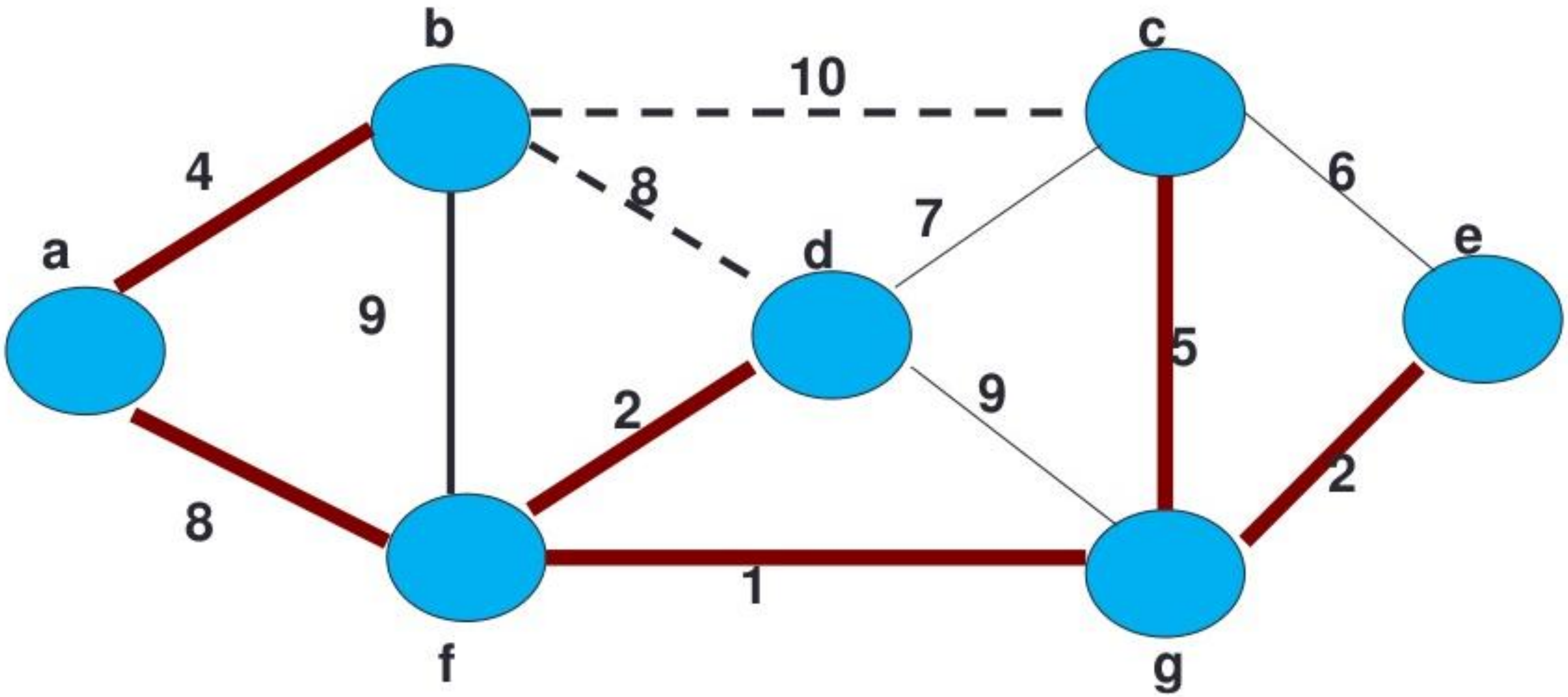
الگوریتم پریم



الگوریتم پریم



الگوریتم پریم



Minimum Cost=22