فصل ششم: گراف ها

اهداف

- آشنایی با گراف
- ماتریس مجاورتی
- جستجوی گراف

فصل ششم : گراف ها

هر گراف G شامل دو مجموعه V و E است :

V : مجموعه محدود و غیرتهی از رئوس است

E : مجموعه ای محدود و احتمالا غیرتهی از لبه ها می باشد.

V(G) و E(G) : مجموعه رئوس و لبه های گراف G را نمایش می دهند.

برای نمایش گراف هم می توانیم بنویسیم G=(V، E)

۱-۶ گراف ها

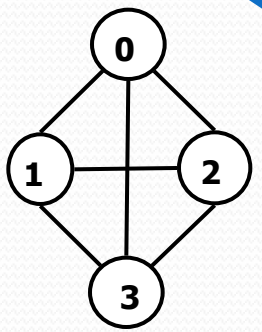
در یک گراف بدون جهت، زوج رئوس ، زوج مرتب نیستند ، بنابراین زوج های (v_1, v_0) و (v_0, v_1) با هم یکسانند.

در یک گراف جهت دار هر لبه با زوج مرتب $< v_i, v_j >$ نمایش داده می شود. v_j انتها و v_i ابتدای لبه هستند. بنابراین $< v_i, v_j >$ و $< v_j, v_i >$ دو لبه متفاوت را نمایش می دهند.

۱-۶ گراف ها

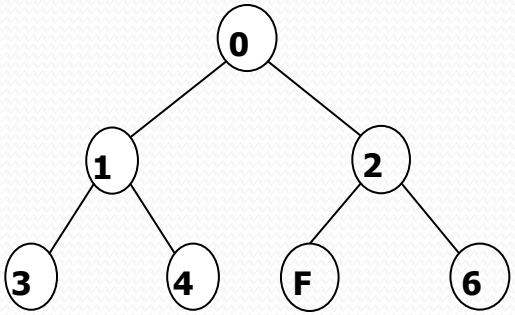
- برای گراف بدون جهت ، لبه ها به صورت خطوط یا منحنی نمایش داده می شوند.
- برای گراف های جهت دار لبه ها به صورت فلش هایی که از انتها به ابتدا رسم شده است ، ارایه می گردند.

مثال

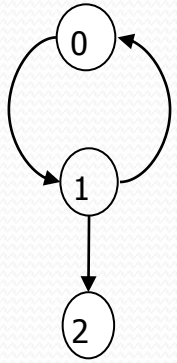


G_1

بدون جهت



G_2



G_3

جهت دار

۱-۶ محدودیت های گراف ها

- محدودیت های زیر بر روی گراف ها اعمال می شود :
- یک گراف فاقد لبه ای از یک راس مانند i ، به خودش می باشد. این مطلب بدین معنی است که لبه (v_i , v_i) غیر معتبر می باشد.
- یک گراف فاقد رویداد چندگانه از یک لبه می باشد.

گراف کامل : گراف کامل گرافی است که دارای حداکثر تعداد لبه باشد.

۱-۶ گراف ها

برای یک گراف بدون جهت با **n** راس ، حداکثر تعداد لبه ها ، تعداد متمایز و غیرمرتب زوج های (v_i, v_j) ، **i ≠ j** می باشد. این تعداد برابر است با :

n(n-1) / 2

اگر گراف جهت داری با **n** گره وجود داشته باشد، بیشترین تعداد لبه های آن برابر است با :

n(n-1)

۱-۶ گراف ها

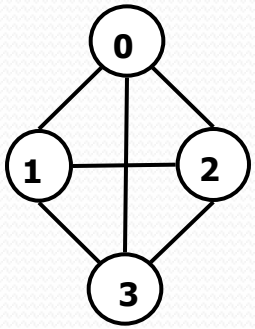
اگر (v_0, v_1) یک لبه در گراف بدون جهت باشد ، می گوییم رئوس v_0 و v_1 دو راس مجاور و لبه (v_i, v_j) یک لبه متلاقی روی v_i و v_j است .

یک زیر گراف G، گرافی است مانند G' به نحوی که $V(G') \subseteq V(G)$ و $E(G') \subseteq E(G)$ باشد.

۱-۶ گراف ها

- طول یک مسیر تعداد لبه های موجود در آن است.
- مسیر ساده ، مسیری است که همه رئوس آن به جز اولی و آخری مجزا باشند.
- حلقه یا سیکل ، یک مسیرساده است که اولین و آخرین راس آن یکی باشد.

برای مثال **0،2،1،0** یک حلقه در G_1 است.



G_1

۱-۶ گراف ها

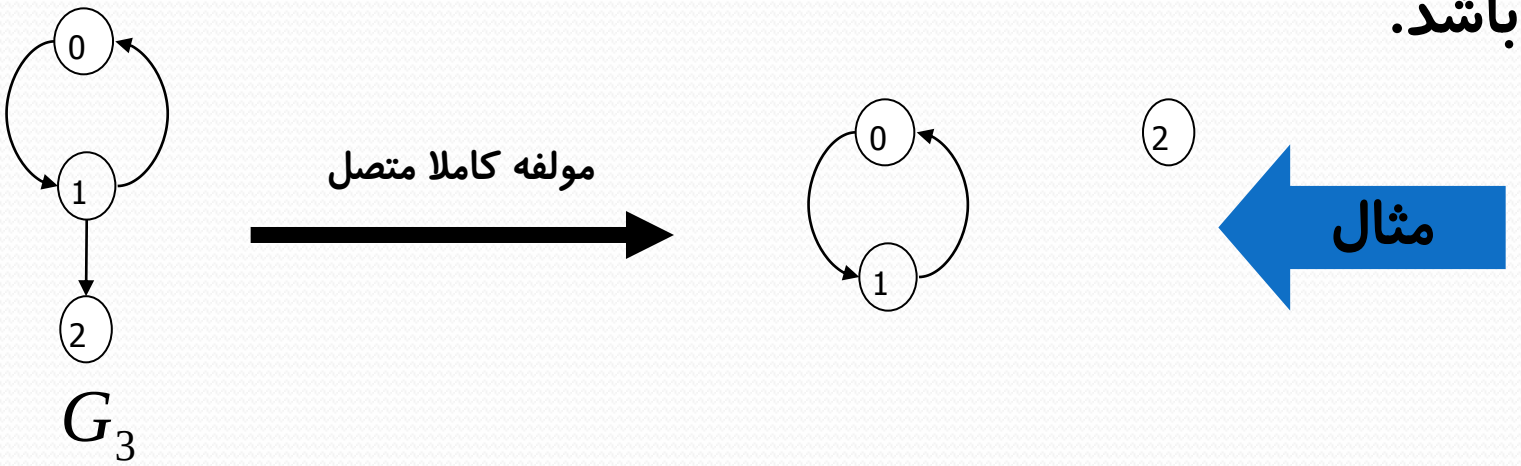
در گراف بدون جهت مانند **G** ، دو راس v_0 و v_1 را **متصل** می گویند، اگر مسیری در **G** از v_0 به v_1 وجود داشته باشد.
یک گراف بدون جهت را **متصل** می نامیم اگر برای هر زوج راس v_j, v_i در **V(G)** ،مسیری از v_i به v_j در **G** وجود داشته باشد.

یک مولفه اتصال یا به طور ساده تر یک مولفه ، در گراف بدون جهت ، بزرگترین زیرگراف متصل آن است.

۱-۶ گراف ها

یک گراف جهت دار کاملا متصل نامیده می شود ، اگر برای هر زوج از رئوس v_i, v_j در V(G) ، مسیری جهت دار از v_i به v_j و همچنین از v_j به v_i وجود داشته باشد.

یک مولفه کاملا متصل ، بزرگترین زیرگرافی است که کاملا متصل باشد.



۱-۶ گراف ها

- درجه یک راس تعداد لبه های متلاقی با آن راس است .

اگر در گراف **G** با **n** راس ،d_iدرجه راس **i** و **e** تعداد لبه ها باشد ، به آسانی می توان دید که تعداد لبه ها برابر است با :

$$e = (\sum_{0}^{n-1} d_i) / 2$$

۱-۶ نمایش گراف

نمایش گراف ها به سه صورت است :

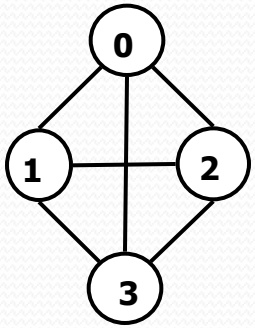
- ماتریس مجاورتی
- لیست های مجاورتی
- لیست های چندگانه

۱-۶ ماتریس مجاورتی

فرض کنید $G=(V,E)$ یک گراف با n راس باشد ، $n \geq 1$، ماتریس مجاورتی گراف G یک آرایه دوبعدی $n \times n$ به نام adj_mat می باشد. اگر لبه (v_i, v_j) (برای گراف جهت دار $<v_i, v_j>$) در $E(G)$ باشد ، آنگاه adj_mat[i] [j]= 0 خواهد بود.

ماتریس مجاورتی برای یک گراف بدون جهت متقارن است زیرا لبه (v_i, v_j) در $E(G)$ خواهد بود ، اگر و تنها اگر لبه (v_i, v_j) نیز در $E(G)$ باشد

۶-۱ ماتریس مجاورتی (مثال)



G_1

ماتریس مجاورتی

	0	1	2	3
0	0	1	1	1
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0

G_1

۱-۶ ماتریس مجاورتی

برای گراف بدون جهت ، درجه هر راس مانند i مجموع عناصر سطری آن است :

$$\sum_{j=0}^{n-1} adj\_mat[i][j]$$

برای یک گراف جهت دار ، مجموع سطری ، درجه خارجه و مجموع ستونی ، درجه وارده خواهد بود.

۶-۱ لیست های مجاورتی

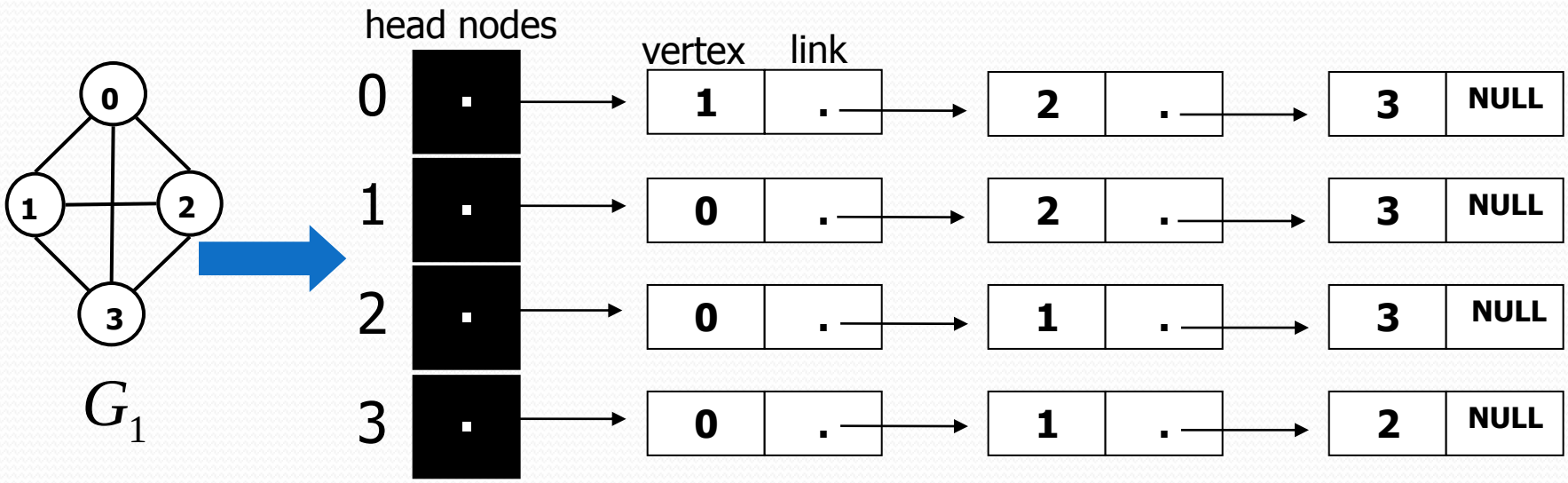
با این نمایش ، **n** سطر ماتریس مجاورتی در **n** لیست پیوندی قرار می گیرد. برای هرراس از گراف **G** ، یک لیست وجود دارد.

هر گره حداقل دو فیلد دارد : راس و اتصال

در هر لیست مشخصی مانند **i** ، گره های لیست حاوی رئوس مجاور از راس **i** می باشند.

در یک گراف بدون جهت با **n** راس و **e** لبه ، **n** گره **head** و **2e** گره لیست دارد. هر گره لیست دو فیلد لازم دارد.

۶-۱ لیست های مجاورتی(مثال)



۶-۱ لیست های مجاورتی

- درجه هر راس در یک گراف بدون جهت را می توان به سادگی با شمارش تعداد گره های آن در لیست مجاورتی تعیین کرد.
- اگر تعداد رئوس گراف **G** برابر با **n** باشد ، تعداد کل لبه ها، در زمان **O(n + e)** تعیین می شود.

۱-۶ یال وزن دار (لبه های وزنی)

- در ماتریس مجاورتی ، کمیت **1** که نشان دهنده وجود یک لبه است را با وزن یک لبه تعویض می کنیم. در لیست های مجاورتی و لیست های مجاورتی چندگانه ، فیلد **weight** به ساختار گره اضافه می شود.

- گرافی که لبه هایش دارای وزن باشد ، **شبکه** نامیده می شود.

۲-۶ اعمال ابتدایی گراف

با توجه به گراف بدون جهت G(V, E) و راس v از V(G) میخواهیم رئوسی از G را که از v قابل دسترسی هستند، به دست آوریم (یعنی همه رئوس متصل به v) . برای این کار دو روش وجود دارد :

- جستجوی عمقی: روش عمقی تا حدودی شبیه پیمایش preorder یک درخت است.

- جستجوی ردیفی (سطحی) : این روش تا حدودی پیمایش ترتیب سطحی درخت را مجسم می کند.

در پیمایش های عمقی و ردیفی ، فرض می کنیم که برای نمایش گراف ها از لیست مجاورتی پیوندی استفاده شده است.

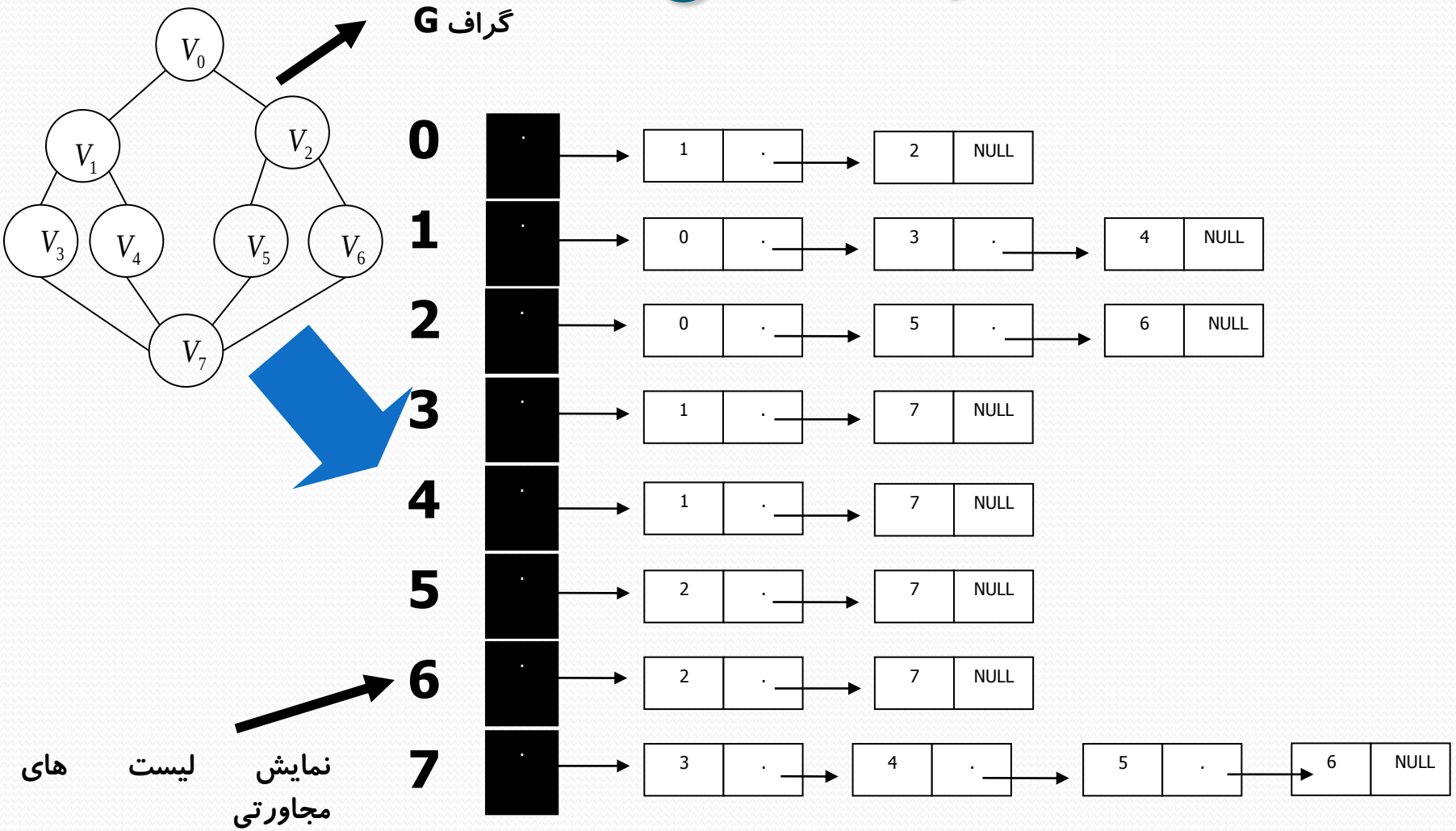
۲-۶ جستجوی عمقی
Depth-first search (DFS)

- در آغاز راس **V** را ملاقات می کنیم.
- بعد راسی مانند **W** را که قبلا ملاقات نشده و مجاور به **V** است را انتخاب کرده و روش جستجوی عمقی را با **W** دنبال می کنیم.
- موقعیت جاری راس **V** در لیست مجاورتی با قرار دادن آن در یک پشته صورت می گیرد.
- در نهایت ، جستجو به راسی مانند **u** خواهد رسید که فاقد هر گونه راس غیرملاقات شده در لیست مجاورتی باشد.
- در این مرحله راسی از پشته انتخاب و و حذف شده و فرآیند فوق به همین صورت تا زمانی که پشته خالی نشده ادامه پیدا می کند.
- بر اساس این روش ، رئوس ملاقات شده، خارج شده و رئوس ملاقات نشده ، داخل پشته قرار می گیرند. جستجو زمانی پایان می پذیرد که پشته تهی باشد.

برنامه ۱-۶

```
void Graph::DFS()        //Driver
{
        visited = new Boolean[n];
        for(int j=0 ; j<n ; j++)
                visited[i] = FALSE;
        DFS(0);
        delete [] visited;
}
void Graph::DFS(int  v)
{
        visited[v] = TRUE ;
        print f ( " %5d "  ،V ) ;
        for ( each vertex w adjacent to v)
                if ( !visited [w] )
                        DFS (w ) ;
}
```

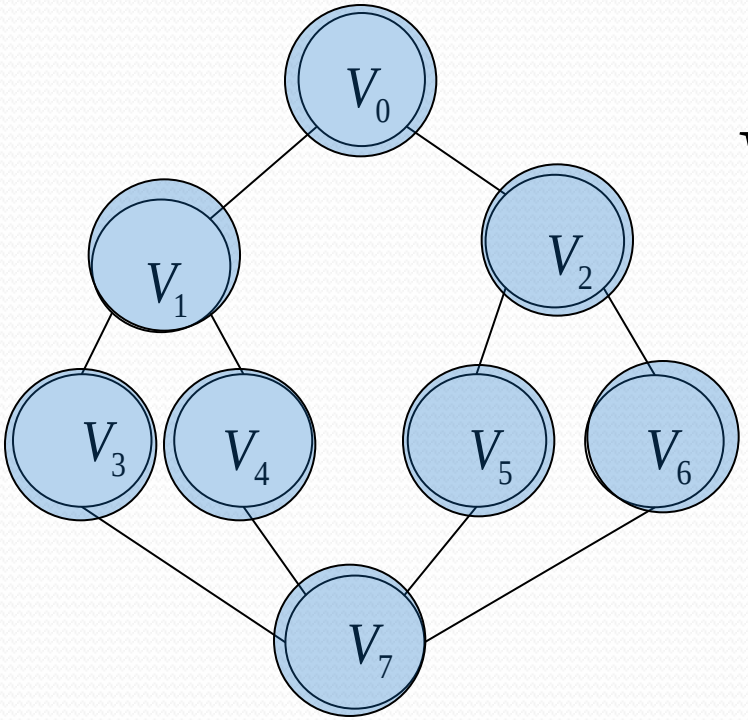
۲-۶ جستجوی عمقی(مثال)



ممکن است برای ذخیره سازی گراف از ماتریس مجاورتی استفاده شده باشد

۲-۶ جستجوی عمقی(مثال)

اگر روش جستجوی عمقی را از راس v_0 آغاز کنیم ، رئوس گراف **G** به ترتیب زیر ملاقات می شوند.



$$v_0, v_1, v_3, v_7, v_4, v_5, v_2, v_6$$

۲-۶ تحلیل DFS

اگر **G** توسط ماتریس مجاورتی نمایش داده شود ، زمان لازم برای تعیین همه رئوس مجاور به **v** ، **O(n)** است. از آنجا که حداکثر **n** راس مشاهده می شود ، کل زمان $O(n^2)$ خواهد شد.

۲-۶ جستجوی ردیفی
breadth-first search (BFS)

- پیمایش را با راس **V** شروع نموده ، پس از ملاقات راس مزبور ، آنرا علامت گذاری می کنیم.
- اول هر یک از رئوس مجاور به راس **V** را در لیست مجاورتی ملاقات میکنیم.
- زمانی که همه رئوس مجاور با راس **V** را ملاقات کردیم ، تمام رئوس ملاقات نشده که مجاور با اولین راس مجاور با **V** در لیست مجاورتی است را ملاقات می کنیم.
- برای انجام این کار مادامیکه هر راس ملاقات می شود ، آنرا در یک صف قرار میدهیم.
- هنگامی که لیست مجاورتی تمام شد، راسی را از صف حذف و با تست هر یک از رئوس در لیست مجاورتی این فرآیند را ادامه می دهیم.
- رئوس ملاقات نشده، ملاقات و سپس در صف قرار می گیرند.
- رئوس ملاقات شده نادیده گرفته می شوند.
- جستجو هنگامی که صف تهی گردد ، خاتمه می یابد.

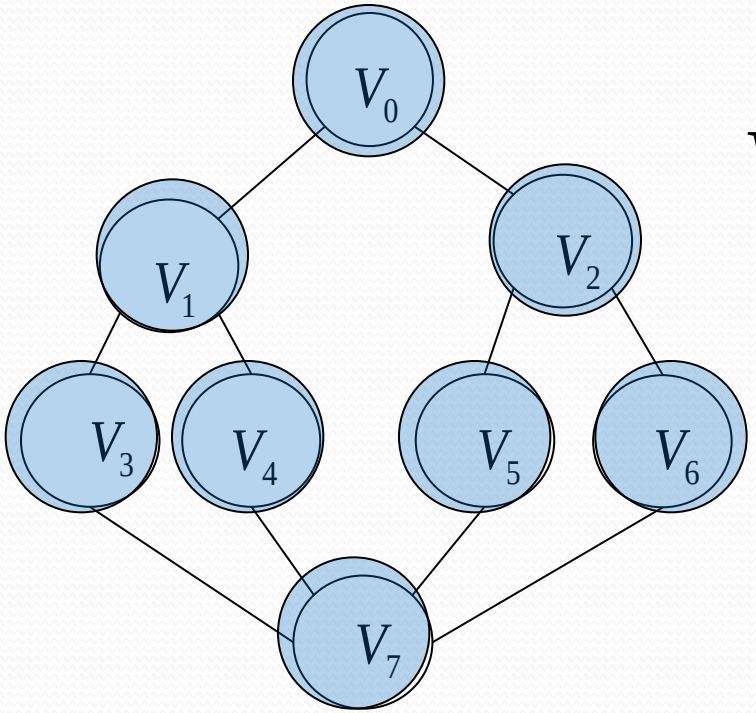
برنامه ۲-۶

```
void Graph::BFS(int  v)
{
        visited = new Boolean[n];
        for (int j=0 ; j<n ; j++)
                visited[i] = FALSE;
        Queue<int> q;
        q.Add(v);
        while (!q.IsEmpty())
        {
                q.Delete(v);
                print f ( " %5d "  ،v) ;
                for ( all vertex w adjacent to v)
                        if ( !visited [w] )
                        {

                          q.Add(w);
                          visited[w] = TRUE ;
                         }
        }
        delete [] visited;
}
```

۲-۶ جستجوی سطحی(مثال)

اگر روش جستجوی سطحی را از راس v_0 آغاز کنیم ،
رئوس گراف **G** به ترتیب زیر ملاقات می شوند.



$$v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$$

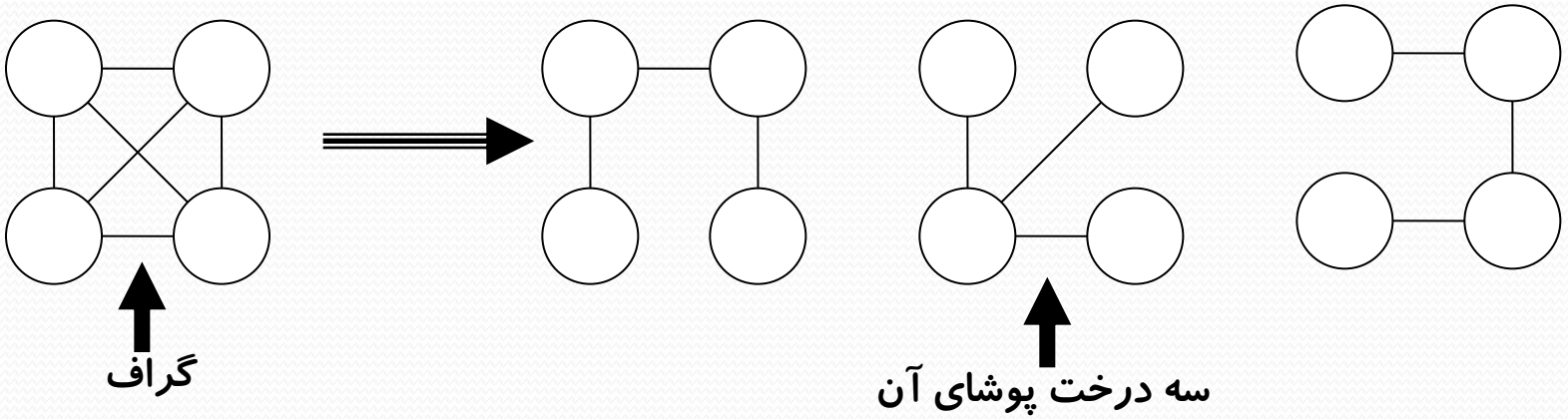
محاسبه مولفه های همبند

- به بخشهای مجزای گراف که با یکدیگر اتصال نداشته باشند مولفه های گراف میگویند و مولفه های یک گراف با یکدیگر هیچ گونه اتصالی ندارند.
- با استفاده از پیمایش در عمق میتوانیم مولفه های همبند گراف را بدست آوریم.

```
void Graph::Components()
{
        visited = new Boolean[n];
        for(int j=0 ; j<n ; j++)
                visited[i] = FALSE;
        for(j=0 ; j<n ; j++)
                if (! visited[j])
                {
                        DFS(j);
                        OutputNewComponent();
                }
        delete [] visited;
}
```

۲-۶ درخت های پوشا

تعریف درخت پوشا : درختی که تعدادی از لبه ها و تمام رئوس **G** را در بر دارد ، درخت پوشا نامیده می شود :



۲-۶ درخت های پوشا

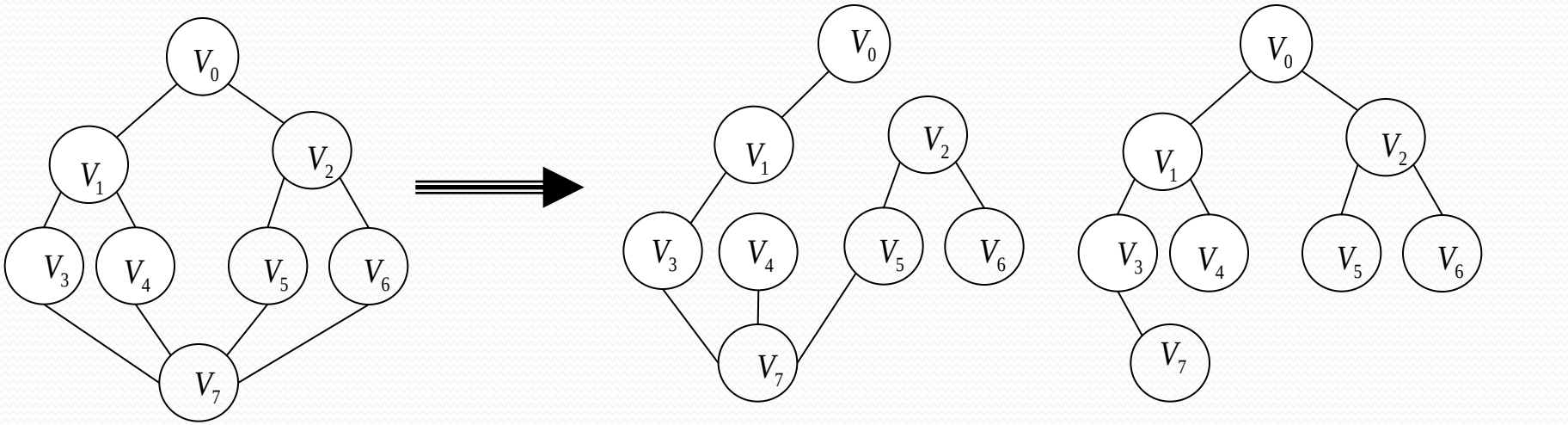
چنانچه گراف **G** متصل باشد ، پیمایش های جستجوی عمقی یا جستجوی ردیفی ، تمام رئوس گراف **G** را ملاقات می کنند.

در این حالت با اعمال هر یک از پیمایش ها لبه های گراف **G** به دو قسمت تقسیم می شوند :

- **T**(برای لبه های درخت) : مجموعه لبه های به کار رفته یا پیموده شده در جستجو می باشد.
- **N** (برای لبه های غیر درخت) : مجموعه لبه های باقی مانده می باشد.

۲-۶ درخت های پوشا

- درخت پوشایی که از فراخوانی **dfs** به دست می آید را درخت پوشای عمقی می نامند.
- چنانچه از روش **bfs** استفاده شود ، درخت پوشای حاصل را درخت پوشای عرضی (سطحی یا ردیفی) می نامند.



درخت پوشای **dfs** درخت پوشای **bfs**

۲-۶ درخت های پوشا

- مقصود از زیر گراف حداقل ، یعنی زیرگرافی که تعداد لبه هایش کمترین باشد.
- هر گراف متصل با n راس ، بایستی حداقل n-1 لبه داشته باشد و همه گراف ها متصل با n-1 لبه ، درخت هستند.
- درخت پوشا دارای n-1 لبه می باشد.
- ایجاد زیرگراف های حداقل ، کاربردهای متعددی در طراحی شبکه های ارتباطی دارد.
- **مثال** : اگر رئوس گراف G نماینده شهرها و لبه ها معرف جاده های ارتباطی بین شهرها بلشد ، آنگاه حداقل تعداد خطوط مورد نیاز برای اتصال n شهر به یکدیگر n-1 خواهد بود.