گرافها-قسمت اول

سید مهدی وحیدی پور

دانشگاه کاشان دانشکده برق و کامپیوتر

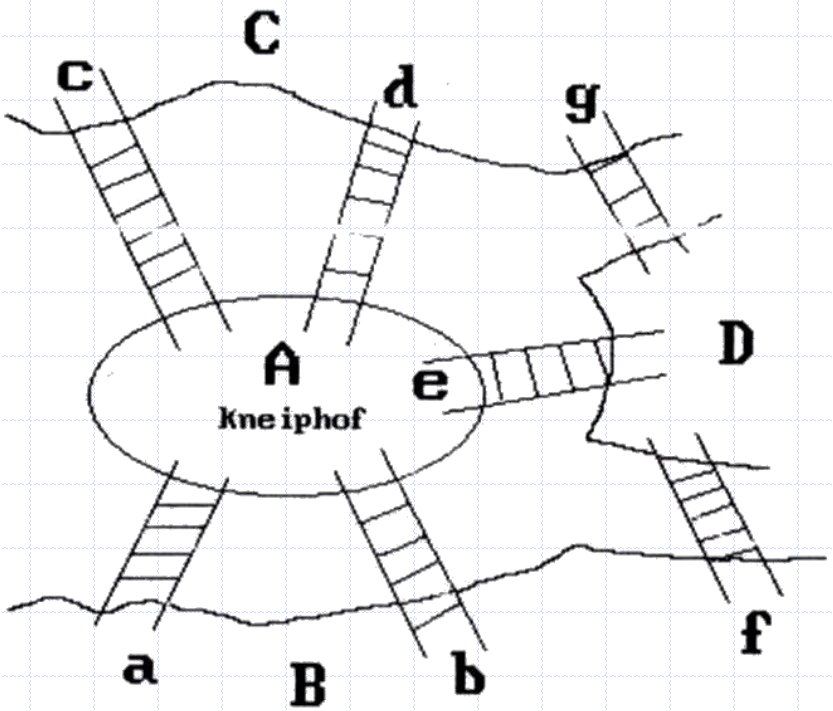
گراف ها

- نوع داده مجرد گراف
  - صورت های مختلف بازنمایی گراف
- عملیات روی گراف ها
  - جستجوی روی گرافها
    - جستجوی عمقی
    - جستجوی عرضی
  - مولفه های همبند
  - درختهای پوشا
  - مولفه های همبند دوطرفه
- درختهای پوشا با کمترین هزینه
- کوتاه ترین مسیر
  - بستار متعدی

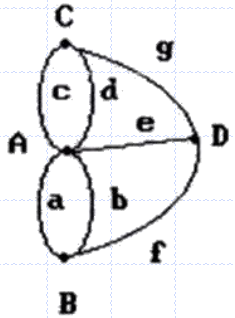
نوع داده مجرد گراف

- مقدمه

مثال : یک مساله گراف



(a)



(b)

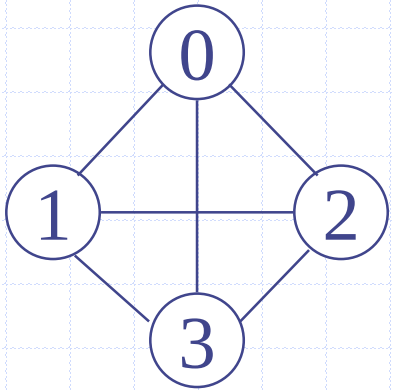
نوع داده مجرد گراف

تعریف

- یک گراف از دو مجموعه تشکیل می شود:
  - یک مجموعه محدود غیر تهی از رئوس V(G)
  - یک مجموعه (احیانا تهی) از یالها E(G)
- G(V,E) یک گراف را نشان می دهد
- یک گراف بدون جهت گرافی است که در آن ترتیب رئوس در یک یال اهمیت ندارد. $(v_0, v_1) = (v_1, v_0)$
- یک گراف جهت دار گرافی است که در آن یک یال متناظر با یک زوج مرتب از رئوس است $<v_0, v_1> != <v_1, v_0>$

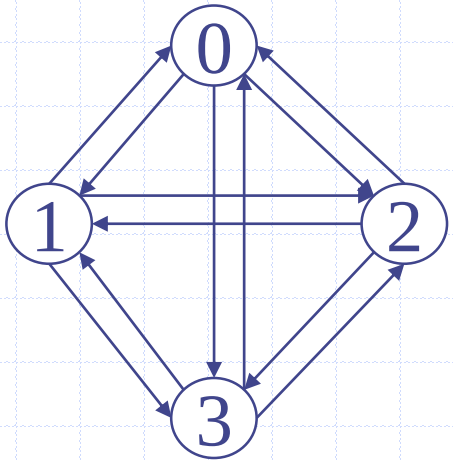
نوع داده مجرد گراف

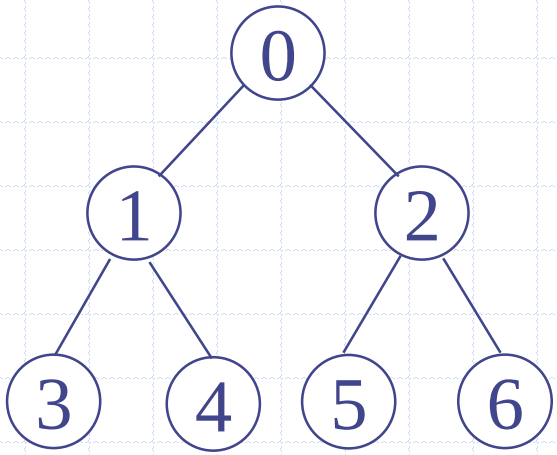
- گراف کامل : گرافی است که دارای حداکثر تعداد لبه باشد.
  - گراف بدون جهت کامل: $n(n-1)/2$ یال
  - گراف جهت دار کامل: $n(n-1)$ یال



G_1

گراف کامل





G_2

G_3

گراف غیر کامل

$V(G_1)=\{0,1,2,3\}$

$E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$

$V(G_2)=\{0,1,2,3,4,5,6\}$

$E(G_2)=\{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$

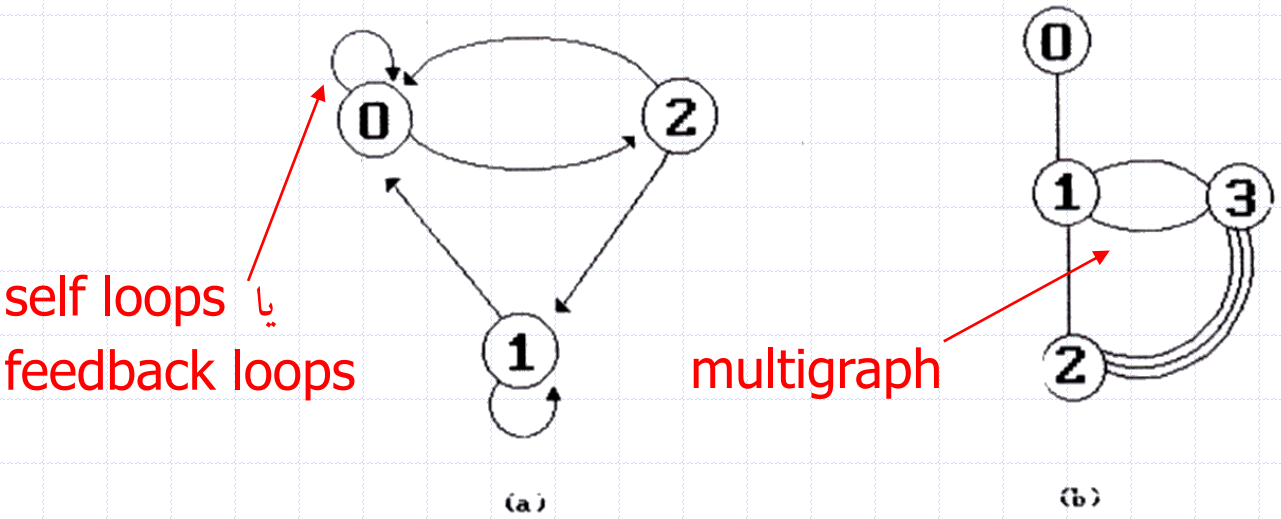
$V(G_3)=\{0,1,2\}$

$E(G_3)=\{<0,1>,<1,0>,<1,2>\}$

نوع داده مجرد گراف

- در یک گراف نمی توان یالی از یک راس به خودش داشت. چنین یالهایی *self loops* نامیده می شوند. با حذف این محدودیت *گراف خودیالی* به دست می آید.

- در یک گراف یک یال نمی تواند چند بار ظاهر شود. با حذف این محدودیت *گراف چندگانه* به دست می آید.



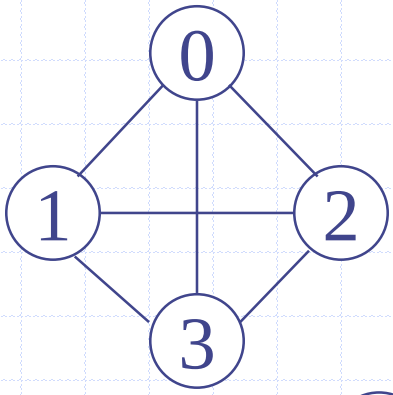
نوع داده مجرد گراف

- اگر (u,v)یک یال از یک گراف بدون جهت باشد
  - u و v مجاور هستند و یال (u,v) حاوی راس های u و v است.

- اگر <u,v> یک یال از یک گراف جهت دار باشد
  - راس u مجاور به راس v و راس v مجاور از راس u است. یال <u,v> حاوی راس های u و v است.

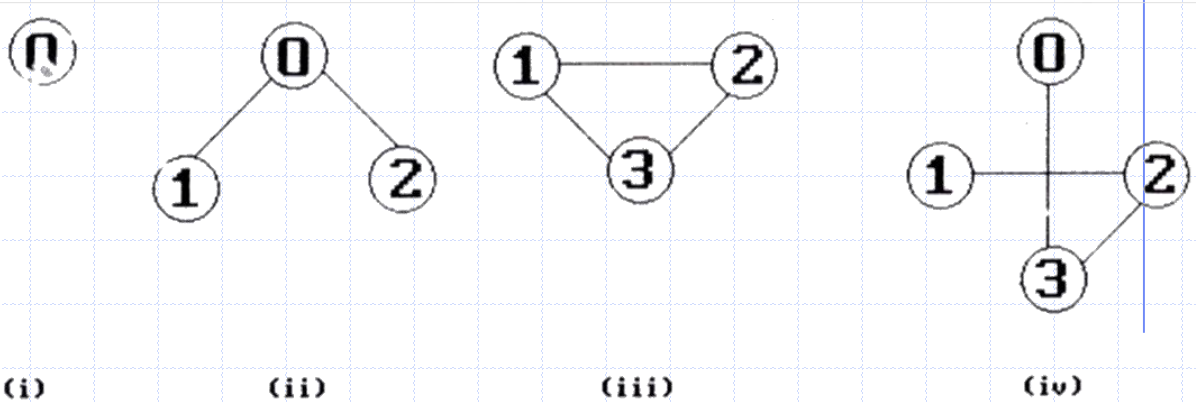
نوع داده مجرد گراف

- یک زیر گراف G گرافی است مانند `G به گونه ای که
- $V(G') \subseteq V(G)\ and\ E(G') \subseteq E(G)$.

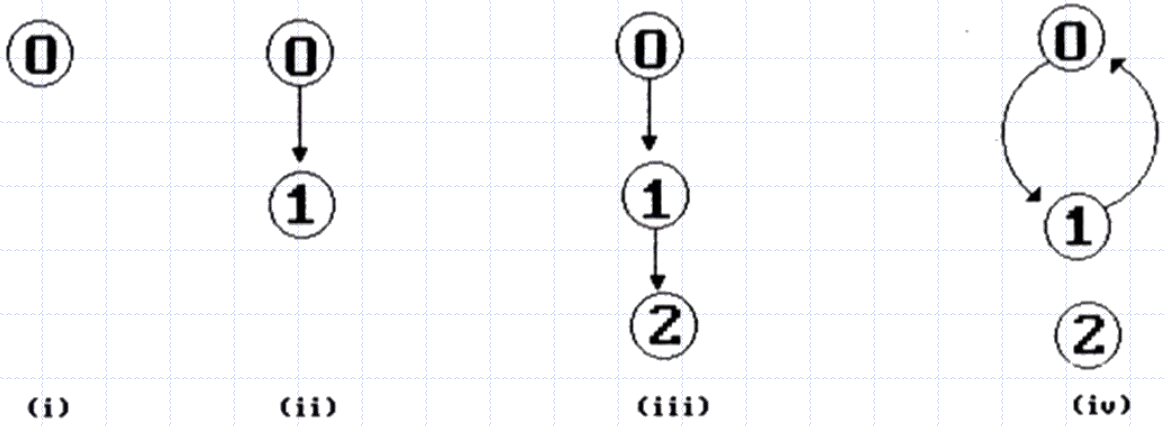


G_1

G_3



(a) Some of the subgraphs of G_1

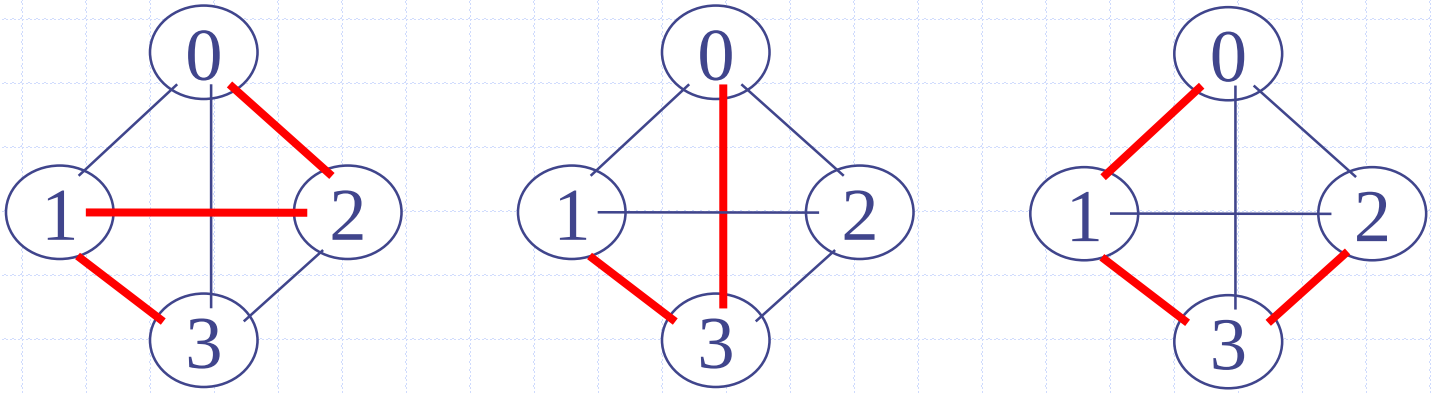


(b) Some of the subgraphs of G_3

نوع داده مجرد گراف

مسیر

- یک مسیر از راس v_p به راس v_q در گراف G دنباله ای از رئوس به صورت $v_p, v_{i1}, v_{i2}, \ldots, v_{in}, v_q$ است به نحوی که $(v_p, v_{i1}), (v_{i1}, v_{i2}), \ldots, (v_{in}, v_q)$ یالهای گراف G باشند.
  - مسیر (0, 2), (2, 1), (1, 3) به صورت 0, 2, 1, 3 نیز نشان داده می شود.
- طول یک مسیر به صورت تعداد لبه های درون آن تعریف می شود.

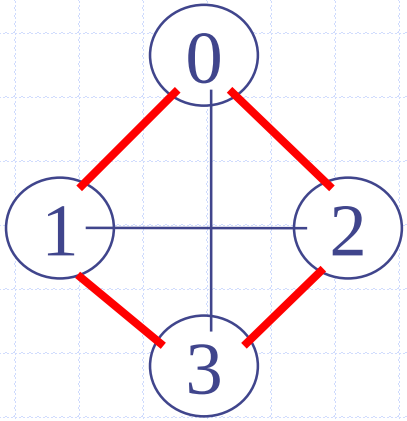


نوع داده مجرد گراف

- مسیر ساده و حلقه

مسیر ساده (جهت دار) مسیری است که در آن تمام راس ها متمایز باشند.

دور یا حلقه مسیر ساده ای است که راس اول و آخر آن یکی است.



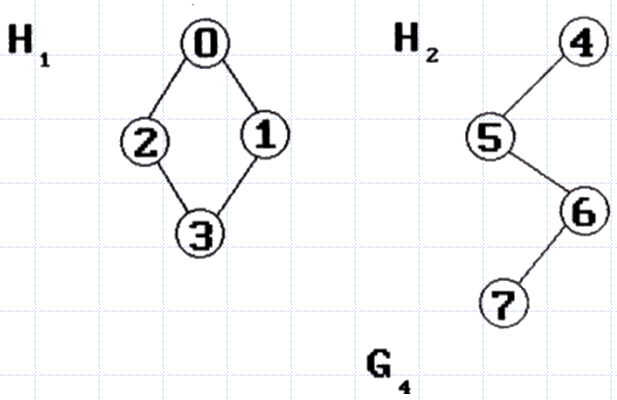
برای مثال 0،2،3،1،0 یک حلقه است.

نوع داده مجرد گراف

- گراف همبند
  - در یک گراف بدون جهت G دو راس u و v را همبند گویند اگر مسیری در G بین آنها وجود داشته باشد.
  - یک گراف بدون جهت را همبند گویند اگر و فقط اگر برای هر زوج راس u و v مسیری از u به v در G وجود داشته باشد.
- مولفه های همبند

مولفه همبند یک گراف بدون جهت بزرگترین زیر گراف همبند آن است

- درخت یک گراف همبند بدون دور است

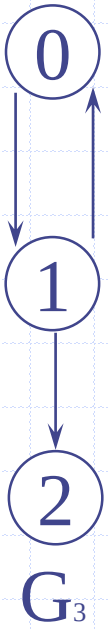


یک گراف با دو مولفه همبند

G4

نوع داده مجرد گراف

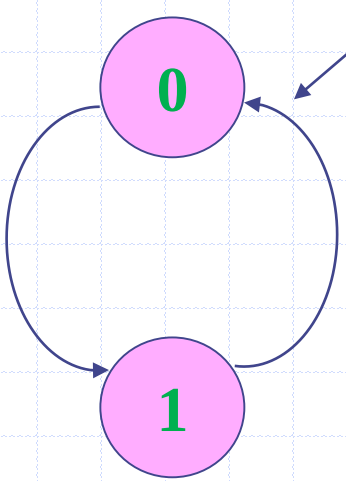
- مولفه همبند قوی
  - یک گراف جهت دار همبند قوی است اگر و فقط اگر برای هر زوج راس u و v یک مسیر جهت دار از u به v و همچنین از v به u وجود داشته باشد.
  - مولفه همبند قوی بزرگترین زیر گرافی است که همبند قوی باشد.

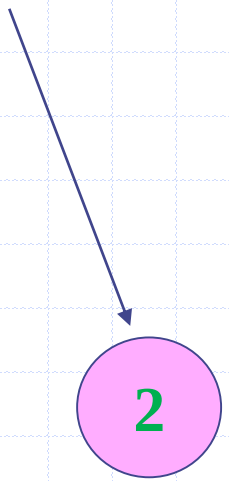


G_3

گراف همبند قوی نیست

مولفه همبند قوی

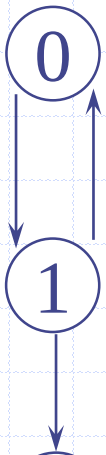




نوع داده مجرد گراف

- درجه
  - درجه یک راس، تعداد یالهایی است که با آن تلاقی دارند
- برای گراف جهت دار
  - درجه ورودی راس: تعداد یالهایی است که سر آنها به راس مذکور متصل باشد.
  - درجه خروجی راس: تعداد یالهایی است که ته آنها به راس مذکور متصل باشد.

گراف روبرو: درجه ورودی راس ۱ برابر ۱ ، درجه خروجی آن برابر ۲ ،درجه آن برابر ۳



G_3

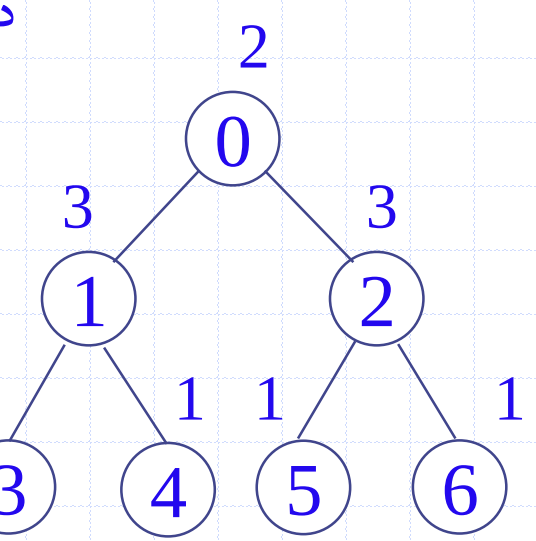
- اگر di درجه راس i در گراف G با n راس و e یال باشد، تعداد یالها برابر است با

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

نوع داده مجرد گراف

گراف جهت دار

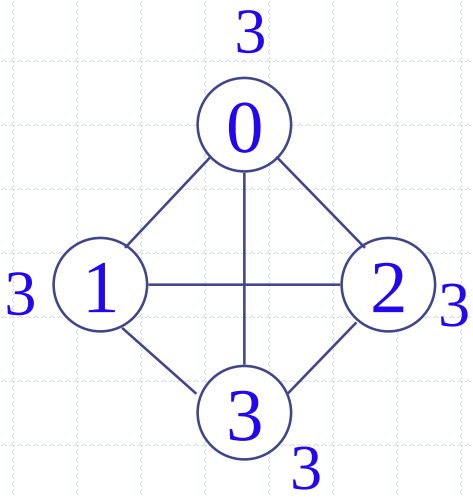
درجه ورودی و خروجی



G_3

گراف بدون جهت

درجه



G_1

G_2

نوع داده مجرد گراف

structure *Graph* is

objects: a nonempty set of vertices and a set of undirected edges, where each edge is a pair of vertices.

functions:

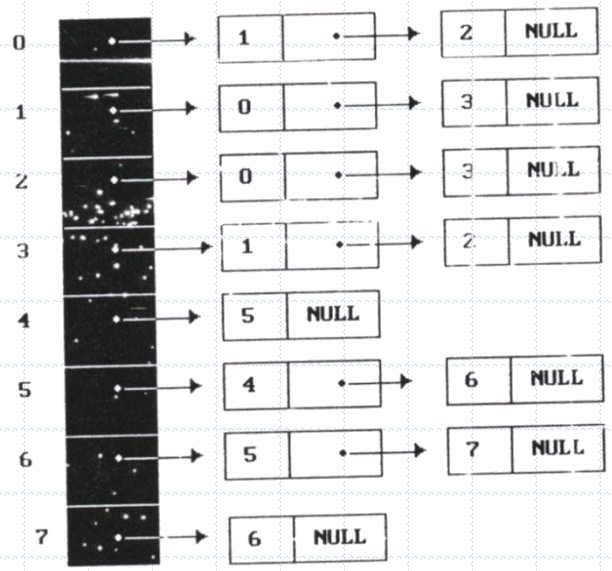
for all *graph* ∈ *Graph*, v, v_1, and v_2 ∈ *Vertices*

Graph Create()	::=	**return** an empty graph.
Graph InsertVertex(*graph*, v)	::=	**return** a graph with v inserted. v has no incident edges.
Graph InsertEdge(*graph*, v_1, v_2)	::=	**return** a graph with a new edge between v_1 and v_2.
Graph DeleteVertex(*graph*, v)	::=	**return** a graph in which v and all edges incident to it are removed.
Graph DeleteEdge(*graph*, v_1, v_2)	::=	**return** a graph in which the edge (v_1, v_2) is removed. Leave the incident nodes in the graph.
Boolean IsEmpty(*graph*)	::=	**if** (*graph* == empty graph) **return** *TRUE* **else return** *FALSE*.
List Adjacent(*graph*, v)	::=	**return** a list of all vertices that are adjacent to v.

بازنمایی گراف

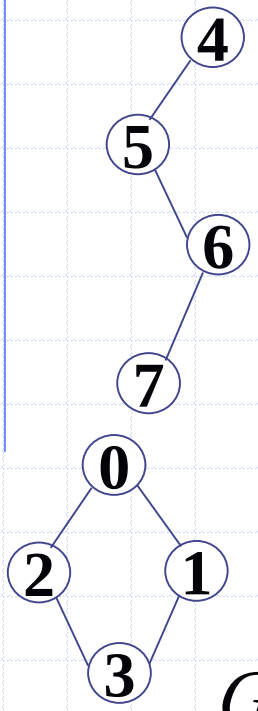
- ماتریس همسایگی
- لیست همسایگی

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



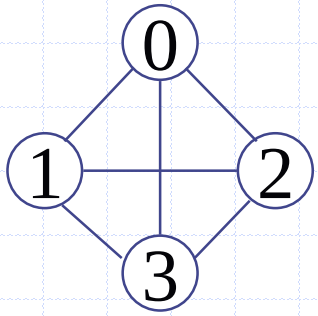
بازنمایی گراف

- ماتریس همسایگی
- برای یک گراف بدون جهت متقارن است ولی برای گراف جهت دار لزوما اینطور نیست



G_4

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



G_1

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

G_3

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

بازنمایی گراف

- ماتریس همسایگی
  - در یک گراف بدون جهت درجه هر راس i برابر است با مجموع عضوهای سطری آن

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

  - در یک گراف جهت دار مجموع عضوهای سطری درجه خروجی آن راس و مجموع عضوهای ستونی درجه ورودی آن راس است.

$$ind(vi) = \sum_{j=0}^{n-1} A[j,i] \qquad outd(vi) = \sum_{j=0}^{n-1} A[i,j]$$

  - پیچیدگی زمانی تشخیص تعداد یال گراف و یا تشخیص همبند بودن گراف
    - گراف بدون جهت $O(n^2/2)$
    - گراف جهت دار $O(n^2)$

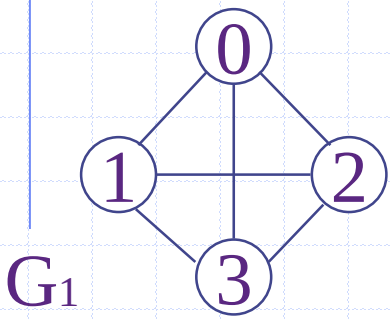
بازنمایی گراف

- لیست همسایگی
- برای یک گراف بدون جهت با n راس و e یال این بازنمایی به n گره سر و 2e گره لیست نیاز دارد.

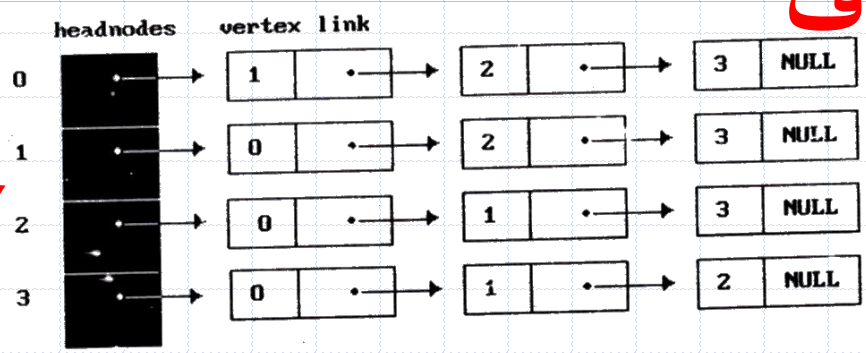
```
#define MAX_VERTICES 50 /*maximum number of vertices*/
typedef struct node *node_pointer;
typedef struct node {
        int vertex;
        struct node *link;
        };
node_pointer graph[MAX_VERTICES];
int n = 0; /* vertices currently in use */
```

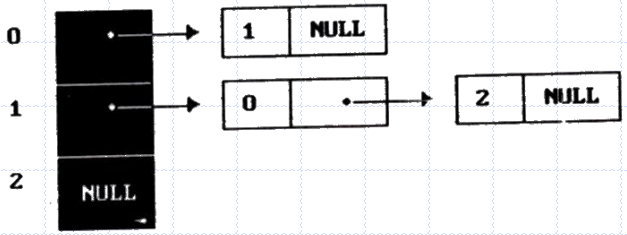
بازنمایی گراف

مثال



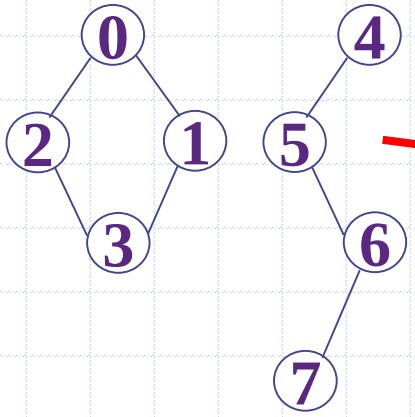
G_1

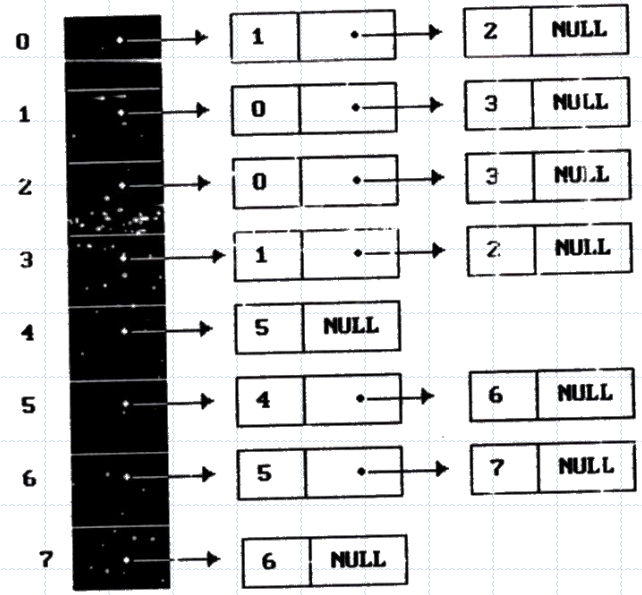




G_3

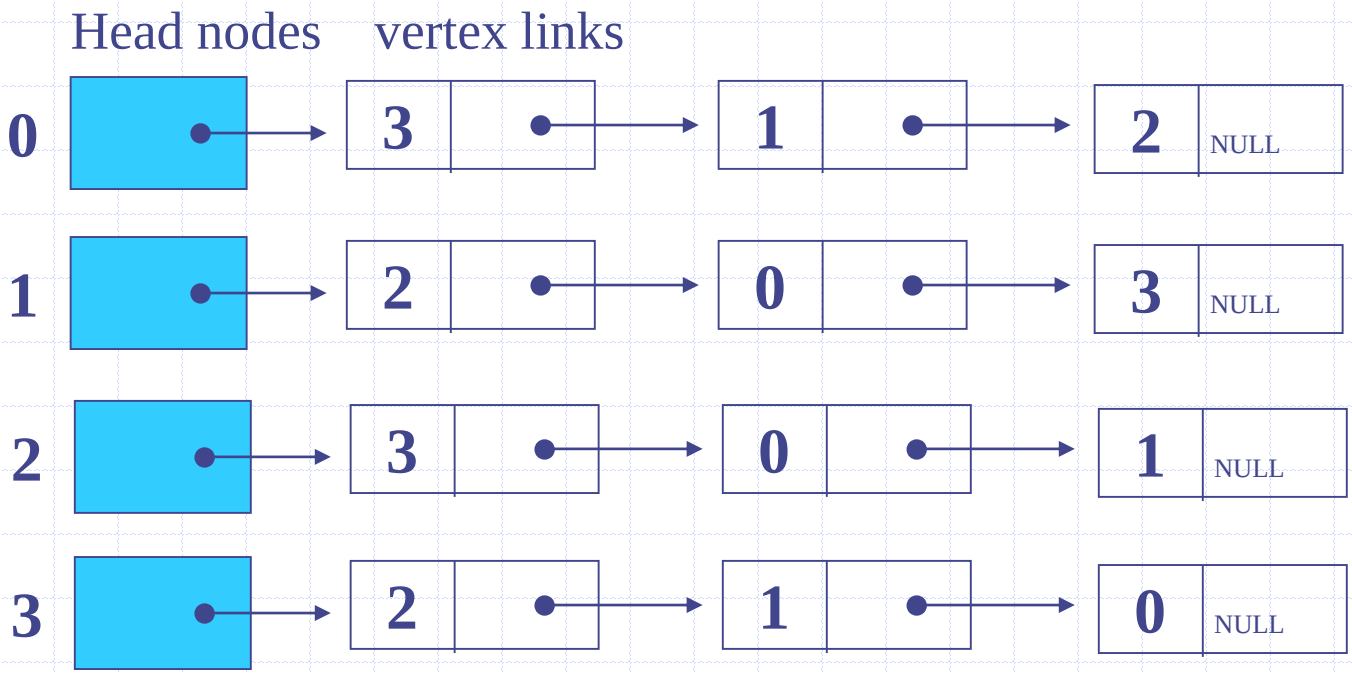
G_4

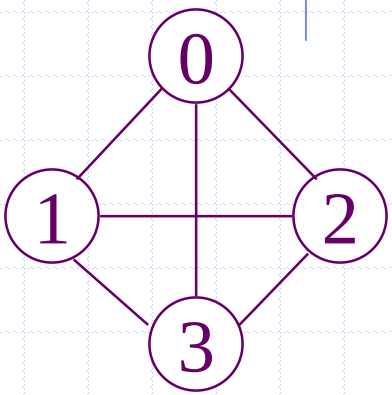




بازنمایی گراف

ترتیب اهمیتی ندارد





بازنمایی گراف

[0]	9		[12]	3	
[1]	11		[13]	0	2
[2]	13		[14]	3	
[3]	15		[15]	1	3
[4]	17		[16]	2	
[5]	18		[17]	5	4
[6]	20		[18]	4	5
[7]	22		[19]	6	
[8]	23		[20]	5	6
[9]	1	0	[21]	7	
[10]	2		[22]	6	7
[11]	0	1			

- نمایش ترتیبی از گراف
  - لیست همسایگی گراف را به صورت ترتیبی در یک ارایه قرار دهید.
  - $node[0] \sim node[n-1]$نقطه شروع لیست همسایگی راس $0 \leqq i < n$, i *را نشان می دهند.*

$$node[n] = n + 2e + 1$$

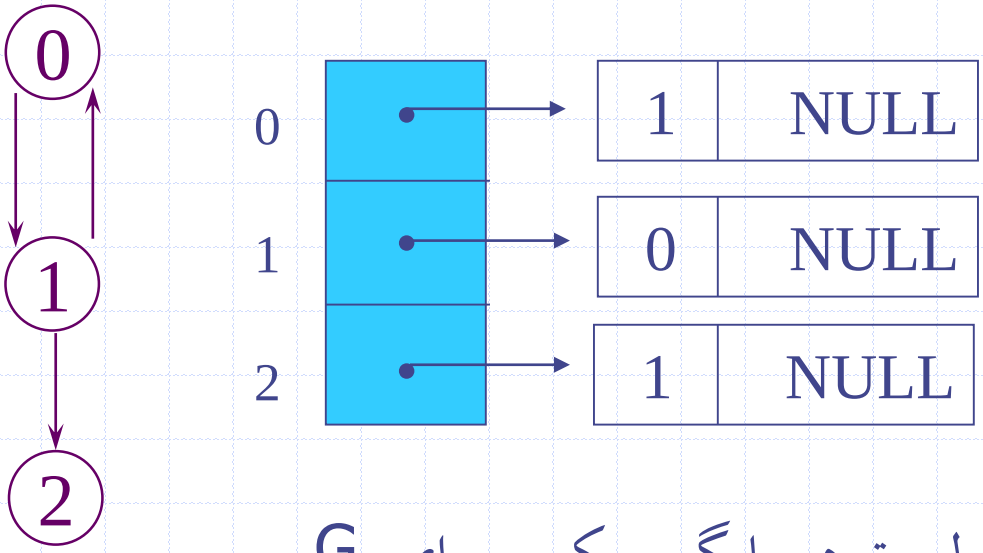
  - راسهای همسایه راس i در $node[node[i]]$تا $node[node[i+1]-1]$قرار دارند.
  - تعداد لبه G را می توان در زمان $O(n+e)$به دست آورد.

بازنمایی گراف

- پیچیدگی زمانی در بازنمایی با لیست همسایگی
  - درجه یک راس در یک گراف بدون جهت
    - تعداد گره در لیست همسایگی آن
  - تعداد لبه ها در گراف
    - در زمان O(n+e) تعیین می شود.
  - درجه خروجی یک راس در یک گراف جهت دار
    - تعداد گره در لیست همسایگی آن
  - درجه ورودی یک راس در یک گراف جهت دار
    - کل ساختار لیست همسایگی باید پیمایش شود.

بازنمایی گراف

- پیدا کردن درجه ورودی راس ها



لیست همسایگی معکوس برای G_3

بازنمایی گراف

- یال های وزن دار
  - یالهای یک گراف می توانند وزن دار باشند. این وزنها می تواند بیانگر
    - فاصله یک راس از راس دیگر باشد
    - هزینه انتقال از یک راس به راس مجاور باشد

- *ماتریس همسایگی:* adj_mat[i][j] می تواند وزن یال را بیان کند
- *لیست همسایگی*: یک فیلد وزن به گره ها اضافه می شود.

- یک گراف با یالهای وزن دار شبکه *network* نامیده می شود.

عملیات روی گراف ها

- پیمایش (جستجو)
  - جستجوی عمقی
  - جستجوی عرضی

Depth First Search (DFS): preorder traversal

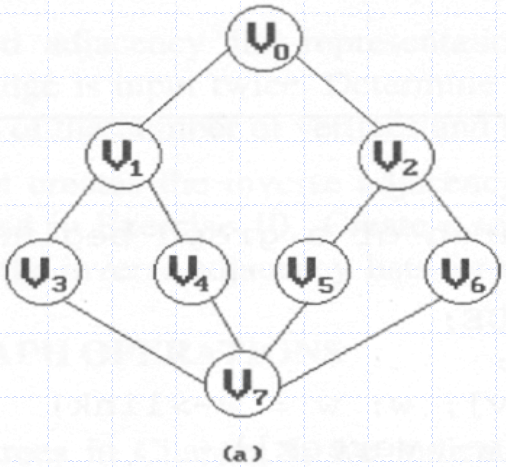
Breadth First Search (BFS): level order traversal

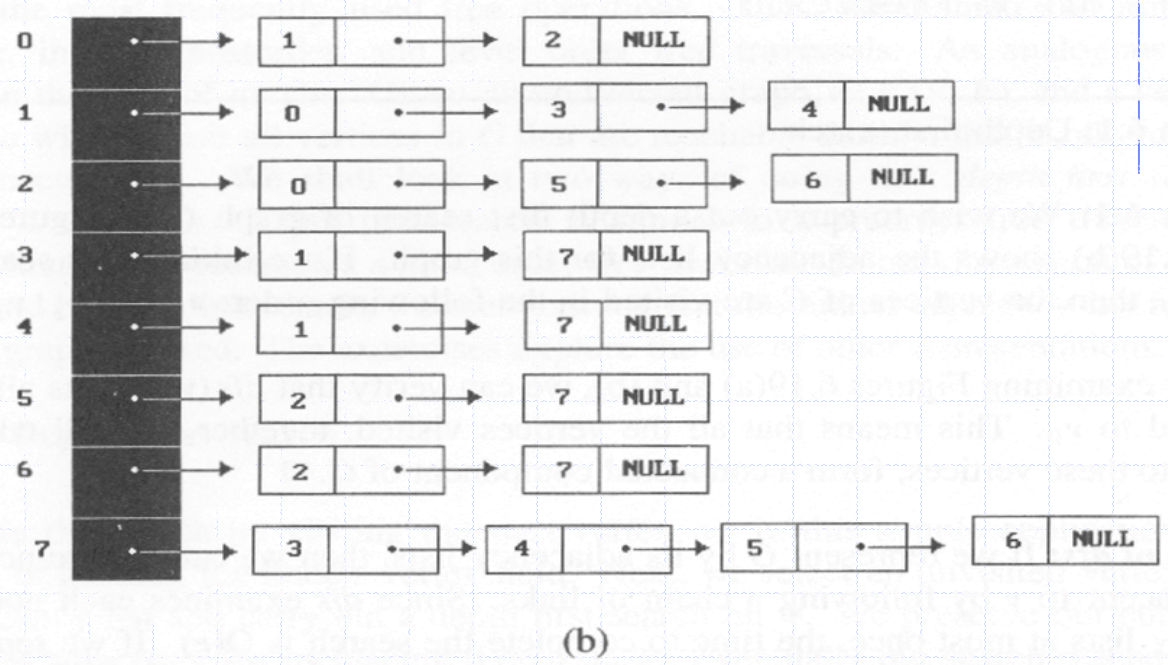
- درخت پوشا
- مولفه های همبند دوطرفه

عملیات روی گراف ها

- مثال پیمایش گراف با استفاده از لیست همسایگی

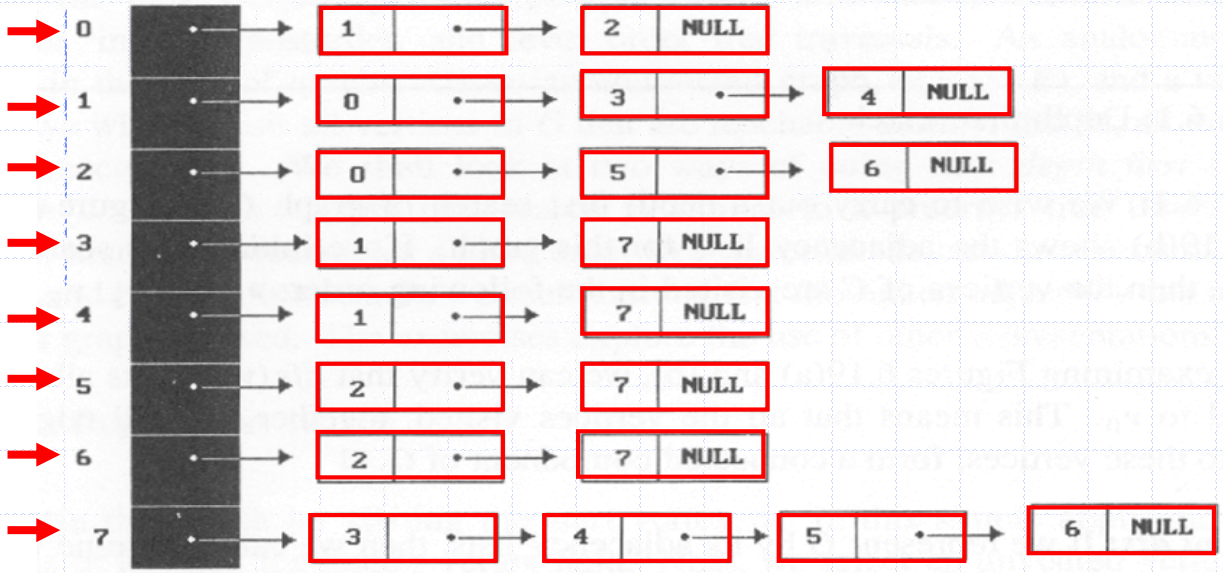
depth first search (DFS): $v_0, v_1, v_3, v_7, v_4, v_5, v_2, v_6$

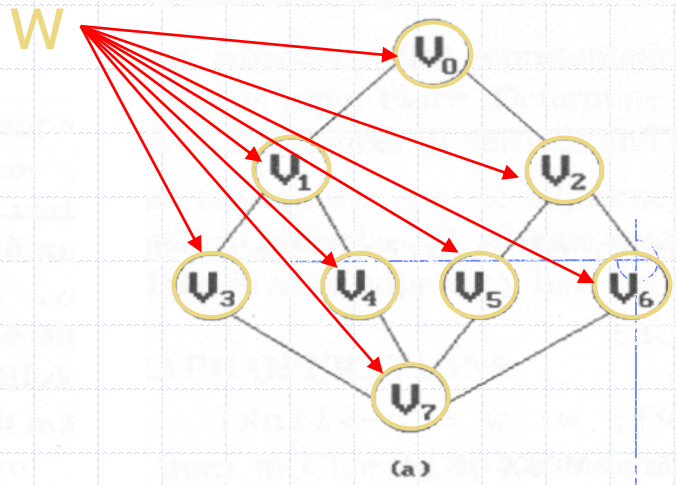




breadth first search (BFS): $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$

جستجوی عمقی





Data structure
adjacency list: O(e)
adjacency matrix: $O(n^2)$

```
void dfs(int v)
{
/* depth first search of a graph begi
   node_pointer w;
   visited[v] = TRUE;
   printf("%5d",v);
   for (w = graph[v]; w; w = w->link)
      if (!visited[w->vertex])
         dfs(w->vertex);
}
```

```
#define FALSE       0
#define TRUE        1
short int visited[MAX_VERTICES];
```

visited: [0] [1] [2] [3] [4] [5] [6] [7]

output: 0 1 3 7 4 5 2 6

عملیات روی گراف ها

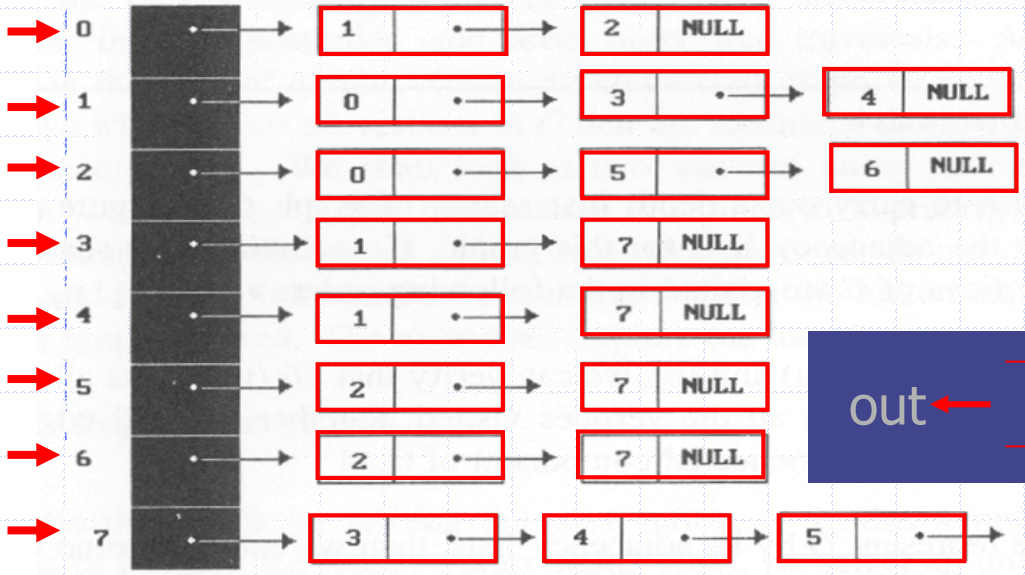
جستجوی عرضی

برای پیاده سازی نیاز به

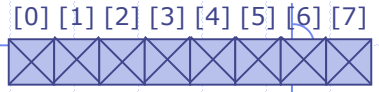
- به یک صف: که اعمال روی صف مشابه اعمال بیان شده در فصل ۴ است
- به یک ارایه سراسری visited: که در ابتدا به صفر مقدار دهی اولیه می شود.

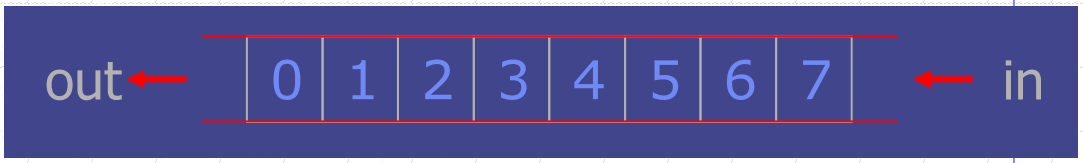
```
typedef struct queue *queue_pointer;
typedef struct queue {
        int vertex;
        queue_pointer link;
        };
void addq(queue_pointer *, queue_pointer *, int);
int deleteq(queue_pointer *);
```

جستجوی عرضی



visited:

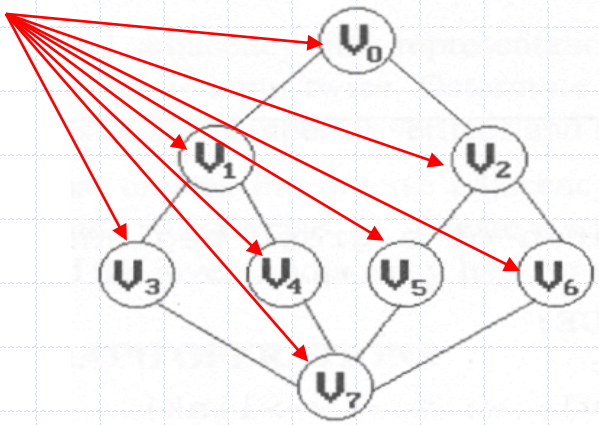




output: 0 1 2 3 4 5 6 7

adjacency list: O(e)
adjacency matrix: $O(n^2)$

W




```
node_pointer w;
queue_pointer front,rear;
front = rear = NULL; /* initialize queue */
printf("%5d",v);
visited[v] = TRUE;
addq(&front, &rear, v);
while (front) {
   v = deleteq(&front);
   for (w = graph[v]; w; w = w->link)
      if (!visited[w->vertex]) {
         printf("%5d", w->vertex);
         addq(&front,&rear,w->vertex);
         visited[w->vertex] = TRUE;
      }
   }
```

عملیات روی گراف ها

- مولفه های همبند
  - اگر G یک گراف بدون جهت باشد می توان تعیین کرد که آیا گراف همبند است یا نه.
    - یکی از دو تابع dfs یا bfs را احضار کنیم و سپس تعیین کنیم آیا راس ملاقات نشده ای وجود دارد یا نه.
  - مولفه های همبند یک گراف را می توان با احضارهای مکرر یکی از دو تابع bfs(v) یا dfs(v) تعیین کرد که در آن v راسی است که هنوز ملاقات نشده است.

```
void connected(void)
{
/* determine the connected components of a graph */
int i;
for (i = 0; i < n; i++)
   if(!visited[i]) {
      dfs(i);
      printf("\n");
   }
}
```

adjacency list: O(n+e)
adjacency matrix: $O(n^2)$

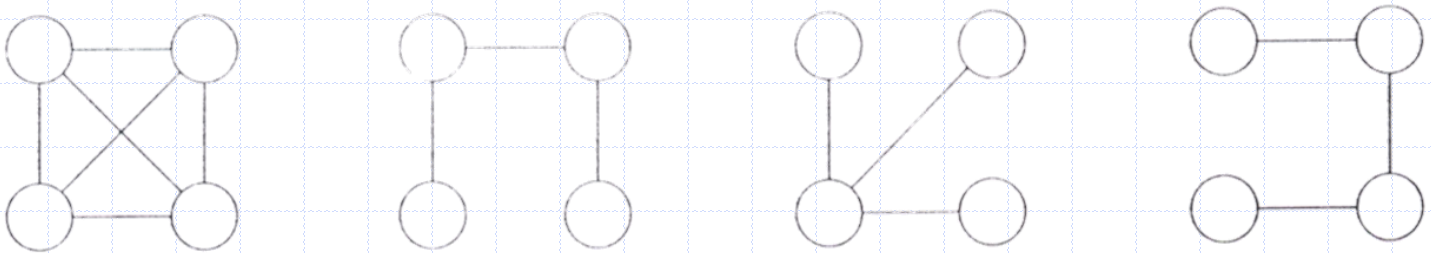
عملیات روی گراف ها

درخت های پوشا

- چنانچه G یک گراف همبند باشد، انگاه پیمایش آن به صورت جستجوی عمقی یا عرضی، با شروع از هر راس دلخواه تمام رئوس گراف G را ملاقات می کنند. در این حالت لبه های گراف G به دو قسمت تقسیم می شوند :
  - T(یعنی لبه های درخت) : مجموعه لبه های به کار رفته یا پیمایش شده در جریان جستجو می باشد.
  - N (یعنی لبه های غیر درخت) : مجموعه لبه های باقی مانده می باشد.
  - لبه های T تشکیل درختی را می دهند که شامل تمام راس های گراف G می باشد.

عملیات روی گراف ها

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



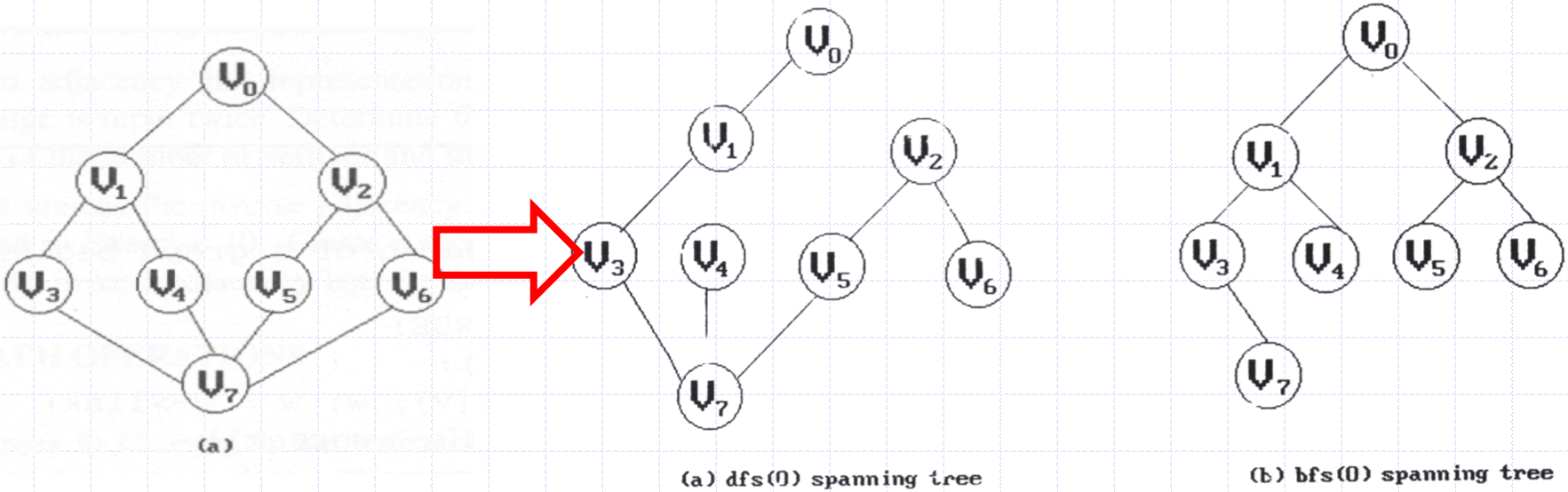
یک گراف کامل و سه درخت از درخت های پوشای آن

عملیات روی گراف ها

- درخت پوشا
- با استفاده از جستجوی عمقی یا جستجوی عرضی می توان درخت پوشا را ایجاد کرد

درخت حاصل از جستجوی عمقی درخت پوشای عمقی نام دارد

درخت حاصل از جستجوی عرضی درخت پوشای عرضی نام دارد



عملیات روی گراف ها

- درخت پوشای عمقی

```
Void dfs (int V)
{
/* depth first search of a graph beginning with vertex V .
*/
        T={};
        node_pointer w ;
        visited[V] = TRUE ;
        print f ( " %5d "  ،V ) ;
        for (w = graph[V] ; w ;  w = w-> link )
                if ( !Visited [w->vertex] ) {
                        dfs (w-> vertex ) ;
                        T=Tu { (v,w) }
                }
}
```

عملیات روی گراف ها

- درخت پوشای عرضی

```
Void bfs (int V)
{
/* breadth  search  of  a graph  beginning  with  vertex V . */
        T={};
        node_pointer  w ;
        queue_pointer  front ,  rear;
        front =  rear = NULL;     /* initialize queue*/
        print f ( " %5d "  ،V ) ;
        visited[V] = TRUE ;
        addq (&front,  &rear, V)
        while (front) {
                V= deleteq (&front) ;
                for (w = graph[V] ; w ;  w = w-> link ) {
                if ( !Visited [w->vertex] ) {
                        print f ( " %5d " , w-> vertex ) ;
                        addq (&front,  &rear, w-> vertex )
                        visited[w-> vertex ] = TRUE ;
                        T=Tυ { (v,w) }
                }
        }
}
```

عملیات روی گراف ها

- ویژگیهای درخت پوشا
  - یک درخت پوشا کوچکترین زیر گراف G′ از G است به طوری که V(G′) = V(G)
    - کوچکترین زیر گراف، زیر گرافی با حداقل تعداد یال تعریف می شود
  - هر گراف متصل با nراس ، 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.
  - اگر یال غیر درختی مانند (v, w) به یک درخت پوشا مانند T اضافه شود آنگاه یک دور ایجاد می شود.