ساختمان داده‌ها

فصل ششم

گرافها

E-mail: Hadi.khademi@gmail.com

مهر ماه ۹۴- دانشگاه علم و فرهنگ

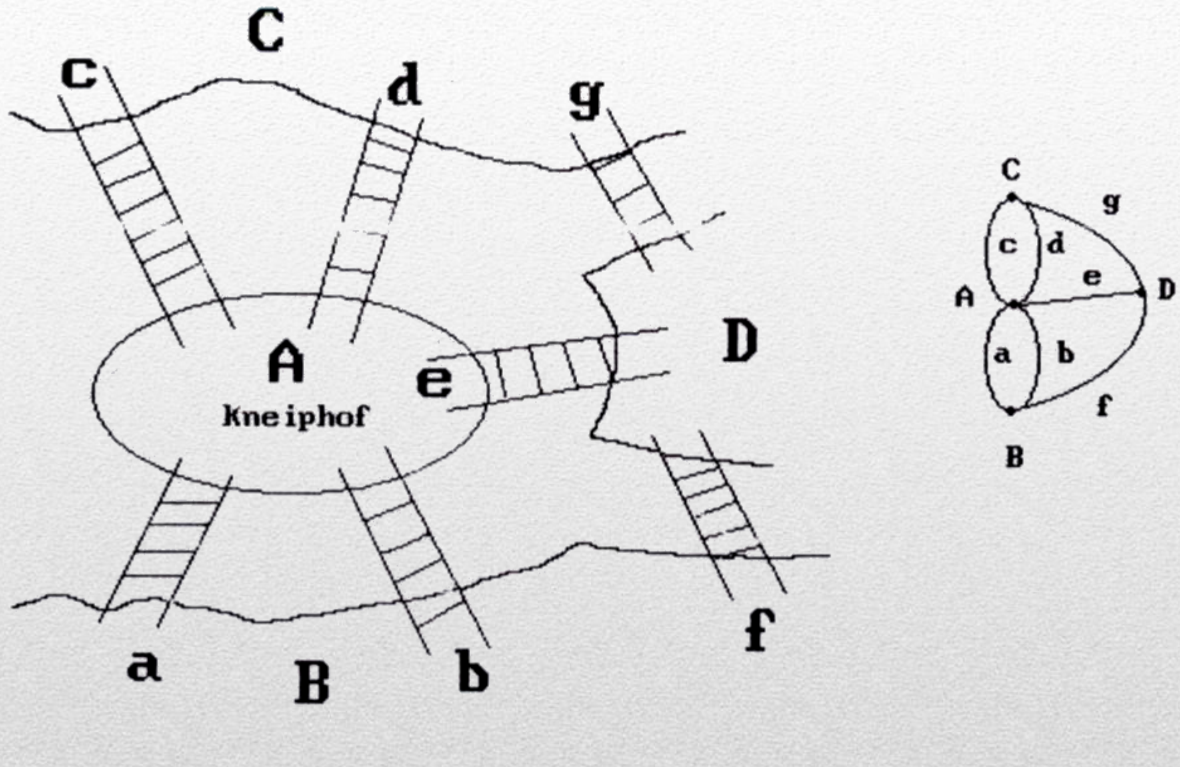
- نوع داده مجرد گراف
  - صورت های مختلف بازنمایی گراف
- عملیات روی گراف ها
  - جستجوی روی گرافها
    - جستجوی عمقی
    - جستجوی عرضی
  - مولفه های همبند
  - درختهای پوشا
  - مولفه های همبند دوطرفه
- درختهای پوشا با کمترین هزینه
- کوتاه ترین مسیر
  - بستار متعدی

گرافها

E-mail: Hadi.khademi@gmail.com

- **مقدمه**

مثال : یک مساله گراف



(a) (b)

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

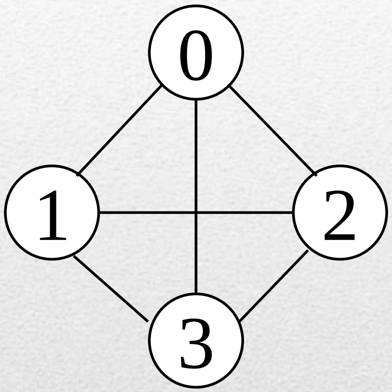
- **تعریف**
- یک گراف از دو مجموعه تشکیل می شود:
  - یک مجموعه محدود غیر تهی از رئوس V(G)
  - یک مجموعه (احیانا تهی) از یالها E(G)
- G(V,E) یک گراف را نشان می دهد
- یک گراف بدون جهت گرافی است که در آن ترتیب رئوس در یک یال اهمیت ندارد. (v0, v1) = (v1,v0)
- یک گراف جهت دار گرافی است که در آن یک یال متناظر با یک زوج مرتب از رئوس است <v0, v1> != <v1, v0>

ابتدا ⟶ انتها

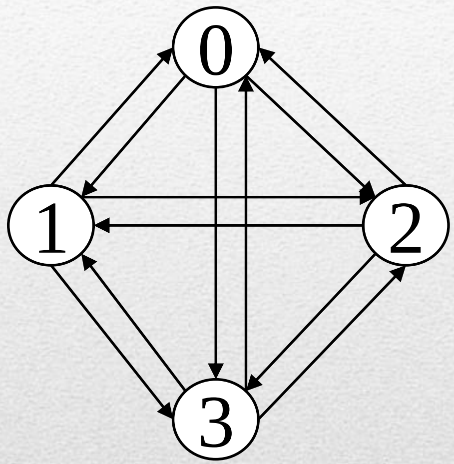
نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

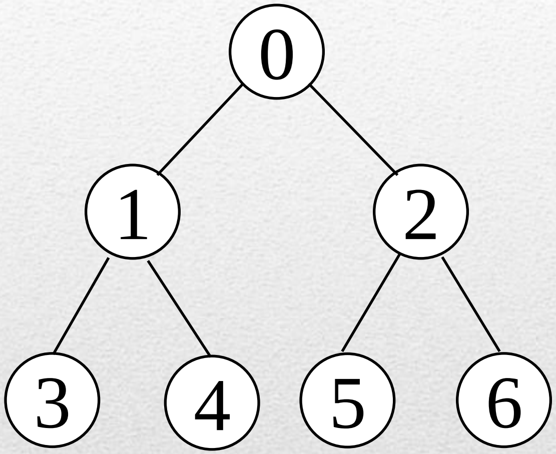
- **گراف کامل : گرافی است که دارای حداکثر تعداد لبه باشد.**
  - گراف بدون جهت کامل: $n(n-1)/2$ یال
  - گراف جهت دار کامل: $n(n-1)$ یال



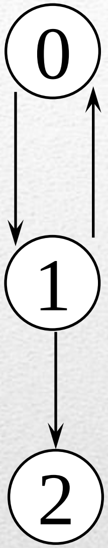
G_1



گراف کامل



G_2



G_3

گراف غیر کامل

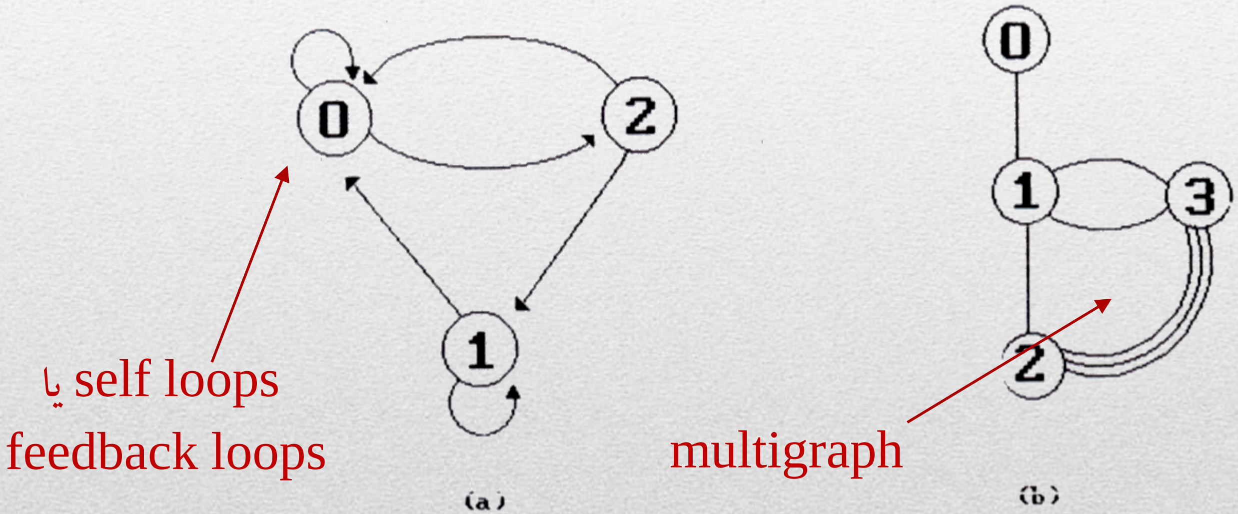
V(G1)={0,1,2,3} E(G1)={(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)}
V(G2)={0,1,2,3,4,5,6} E(G2)={(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)}
V(G3)={0,1,2} E(G3)={<0,1>,<1,0>,<1,2>}

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

- 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.

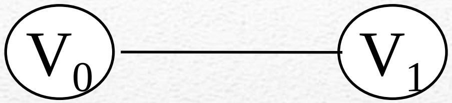
- در یک گراف یک یال نمی تواند چند بار ظاهر شود. با حذف این محدودیت گراف چندگانه به دست می آید.



نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

- **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.



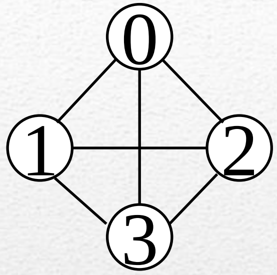
- **اگر <u,v> یک یال از یک گراف جهت دار باشد**
  - راس u مجاور به راس v و راس v مجاور از راس u است. یال <u,v> حاوی راس های u و v است.

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

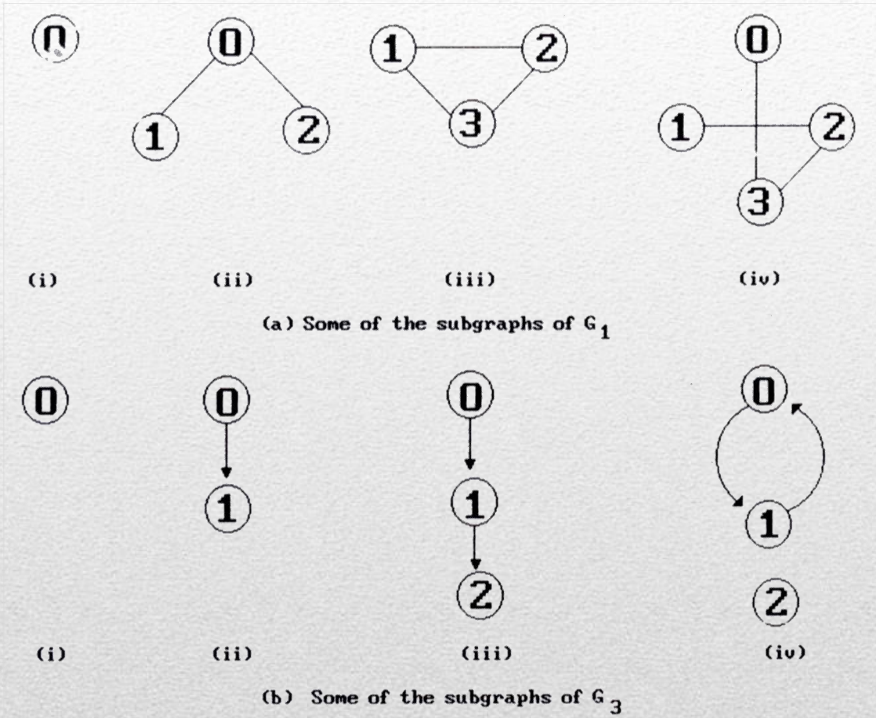
نوع داده مجرد گراف

- 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

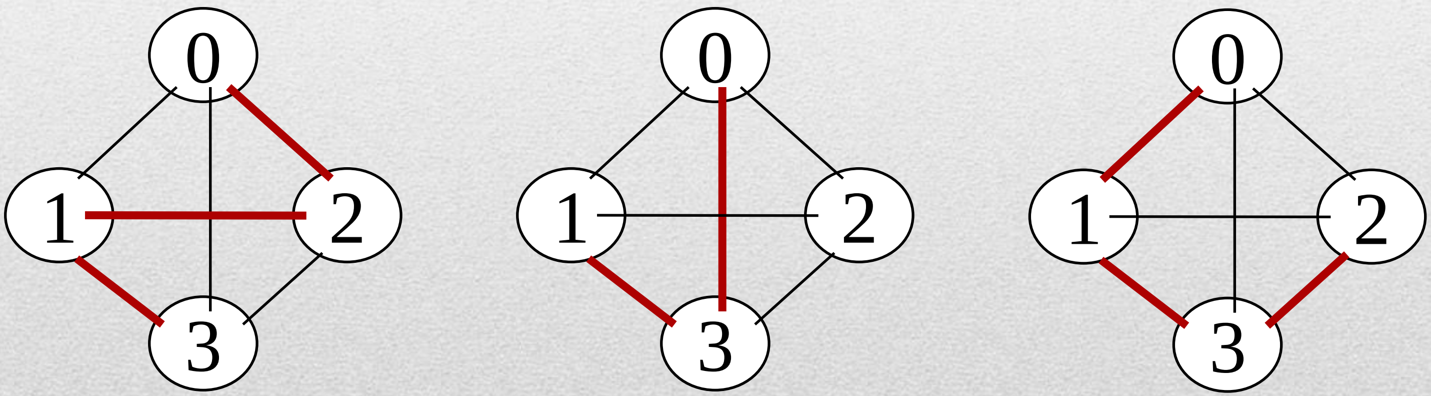
G_3



E-mail: Hadi.khademi@gmail.com

نوع داده مجرد گراف

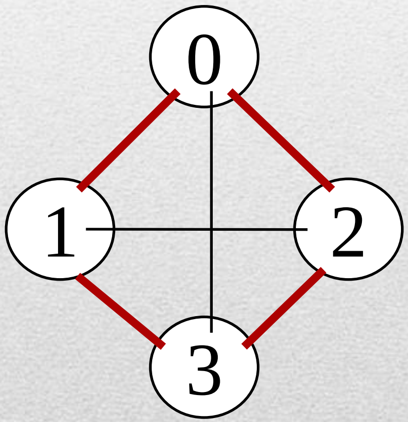
- **مسیر**
  - یک مسیر از راس v_p به راس v_q در گراف G دنباله ای از رئوس به صورت $v_p, vi_1, vi_2, ..., vi_n, v_q$ است به نحوی که
  - $(v_p, vi_1), (vi_1, vi_2), ..., (vi_n, vq)$ یالهای گراف G باشند.
    - مسیر $(0, 2), (2, 1), (1, 3)$ به صورت $0, 2, 1, 3$ نیز نشان داده می شود.
  - طول یک مسیر به صورت تعداد لبه های درون آن تعریف می شود.



E-mail: Hadi.khademi@gmail.com

- **مسیر ساده و حلقه**
- مسیر ساده (جهت دار) مسیری است که در آن تمام راس ها احتمالا به جز راس اول و آخر متمایز باشند.

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

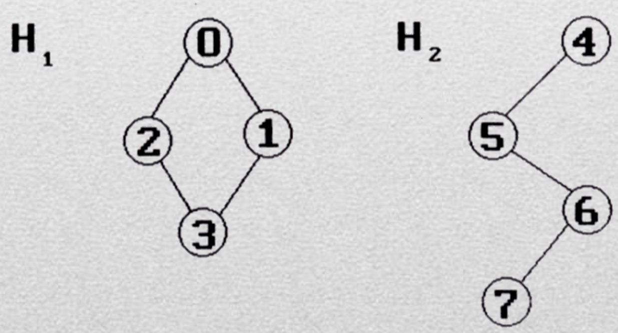


برای مثال 0،1،3،2،0 یک حلقه است.

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

- **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**



G_4

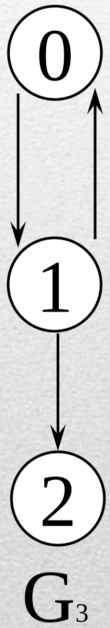
یک گراف با دو مولفه همبند

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

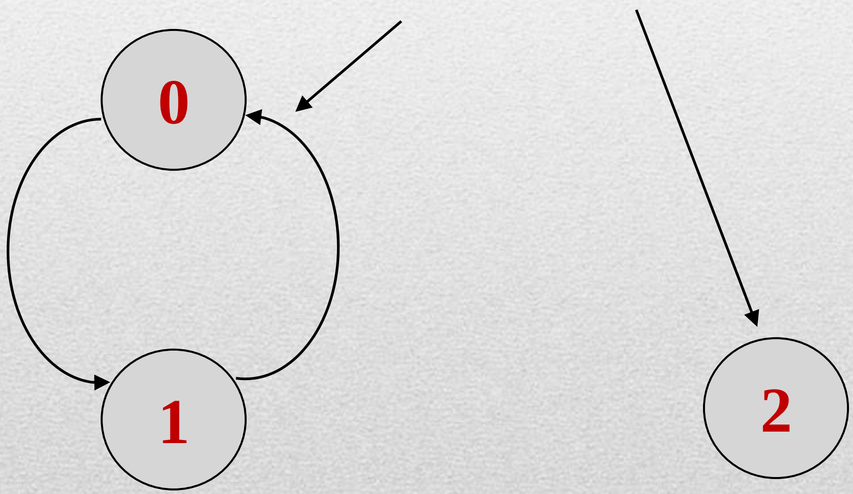
▪ مولفه همبند قوی

- 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.



گراف همبند قوی نیست

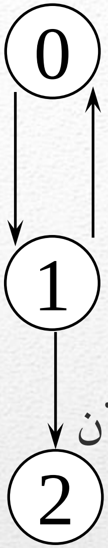
مولفه همبند قوی



نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

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

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

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

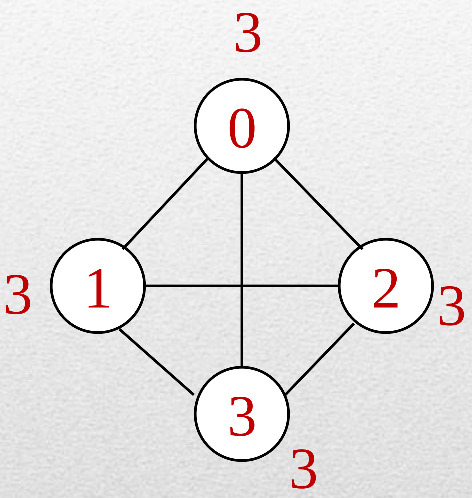
نوع داده مجرد گراف

- گراف جهت دار

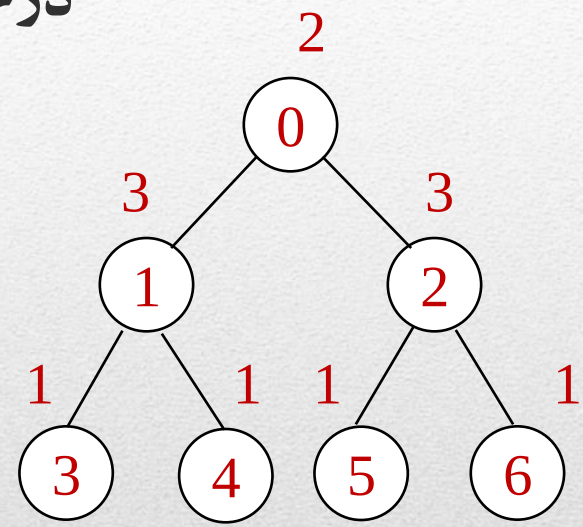
درجه ورودی و خروجی

گراف بدون جهت

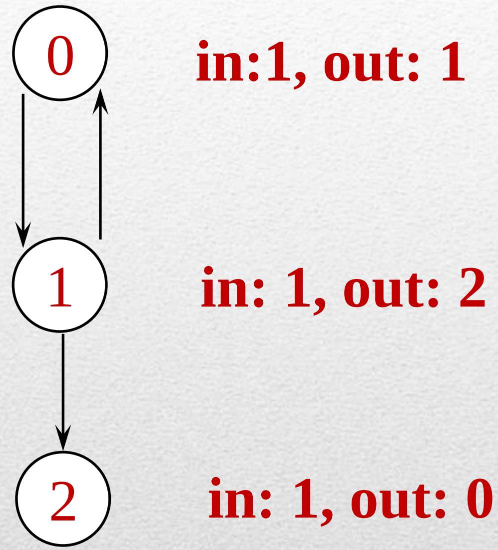
درجه



G1



G2



G3

E-mail: Hadi.khademi@gmail.com

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 $graph \in Graph$, v, v_1, and $v_2 \in Vertices$

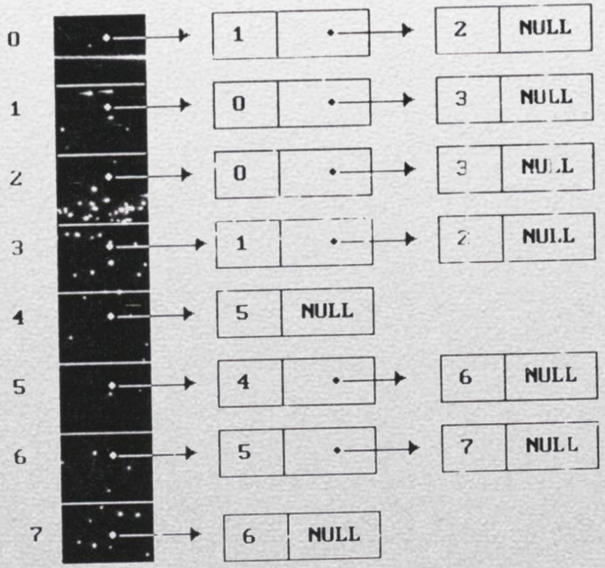
Graph Create()	::=	**return** an empty graph.
Graph InsertVertex(*graph*, v)	::=	**return** a graph with v inserted. v has no incident edges.
Graph InsertEdge(*graph*, v_1, v_2)	::=	**return** a graph with a new edge between v_1 and v_2.
Graph DeleteVertex(*graph*, v)	::=	**return** a graph in which 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.

نوع داده مجرد گراف

E-mail: Hadi.khademi@gmail.com

$$
\begin{array}{c|cccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

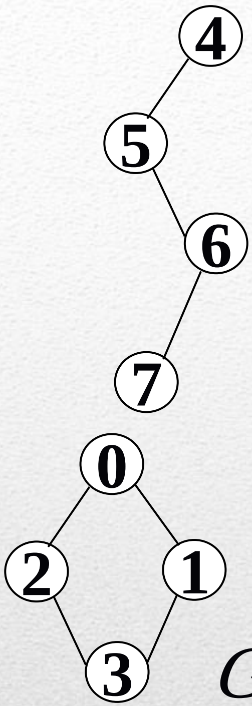
- ماتریس همسایگی
- لیست همسایگی



بازنمایی گراف

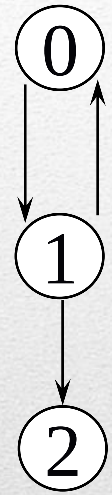
E-mail: Hadi.khademi@gmail.com

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



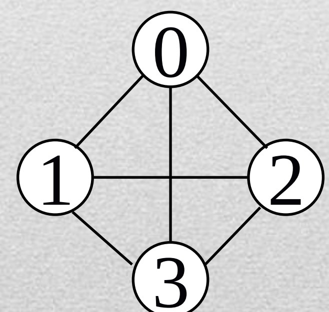
G_4

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



G_3

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



G_1

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

بازنمایی گراف

E-mail: Hadi.khademi@gmail.com

- **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.

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

  - 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

E-mail: Hadi.khademi@gmail.com

- **لیست همسایگی**

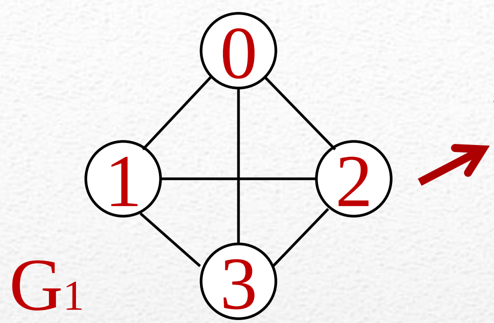
برای یک گراف بدون جهت با n راس و e یال این بازنمایی به n گره سر و 2e گره لیست نیاز دارد.

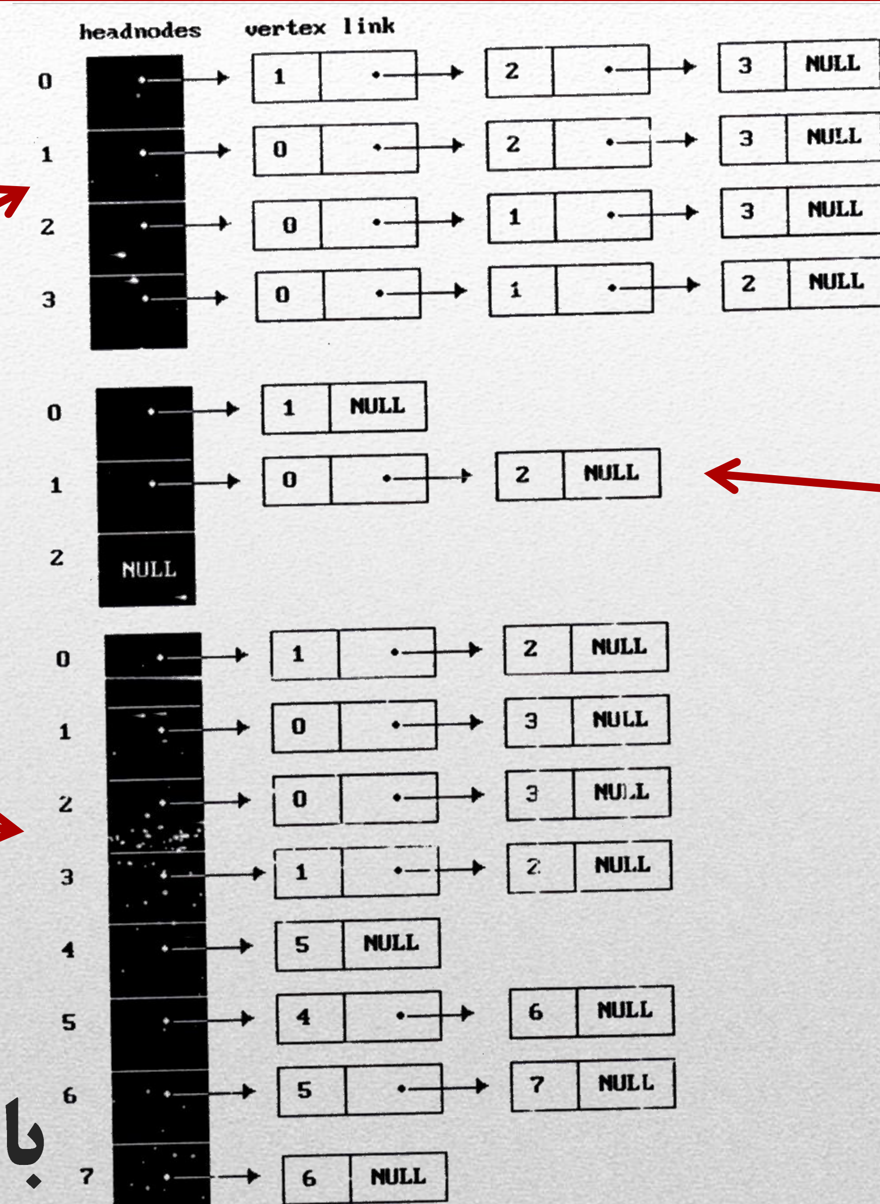
```
#define MAX_VERTICES 50 /*maximum number of vertices*/
typedef struct node *node_pointer;
typedef struct node {
        int vertex;
        struct node *link;
        };
node_pointer graph[MAX_VERTICES];
int n = 0; /* vertices currently in use */
```

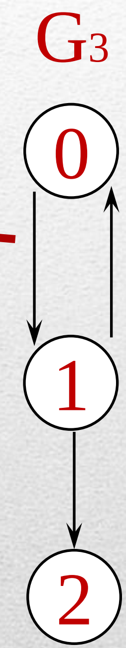
بازنمایی گراف

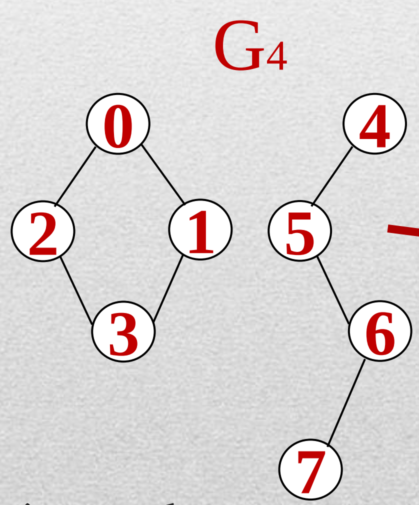
E-mail: Hadi.khademi@gmail.com

- مثال





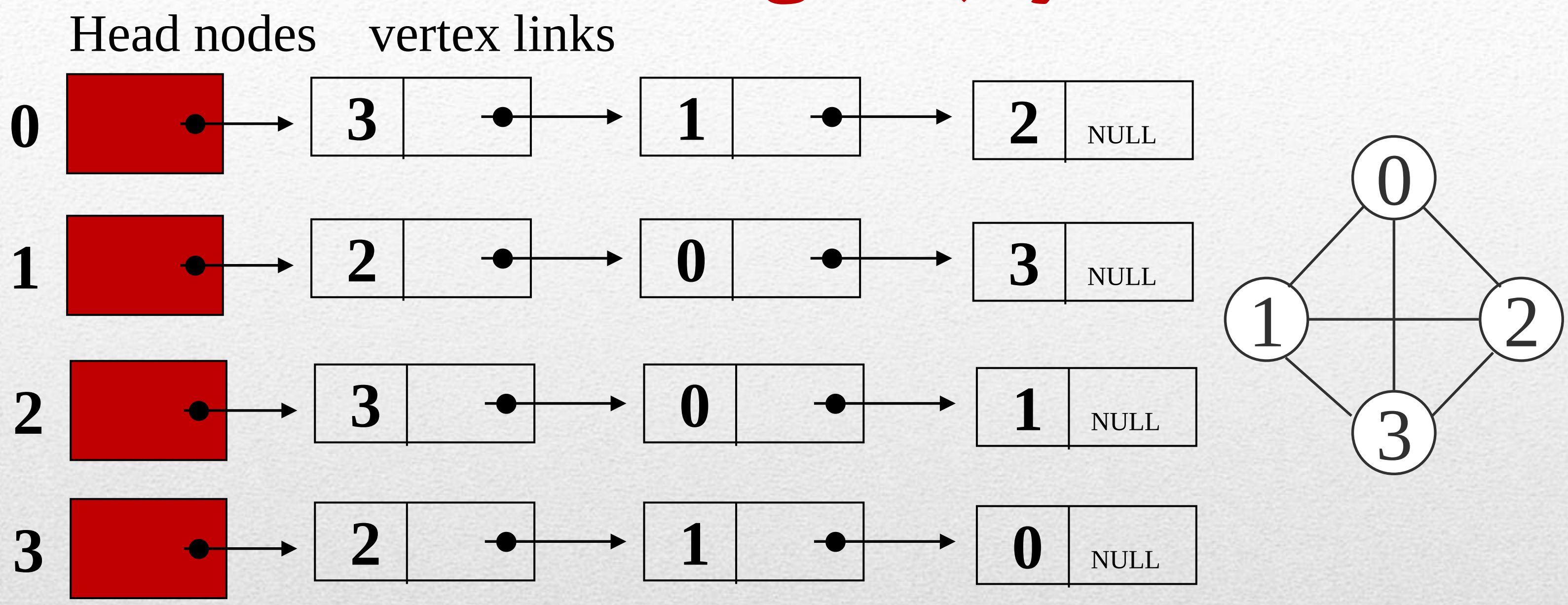




بازنمایی گراف

E-mail: Hadi.khademi@gmail.com

ترتیب اهمیتی ندارد



بازنمایی گراف

E-mail: Hadi.khademi@gmail.com

- **پیمایش (جستجو)**
  - جستجوی عمقی
  - جستجوی عرضی

Depth First Search (DFS): preorder traversal

Breadth First Search (BFS): level order traversal

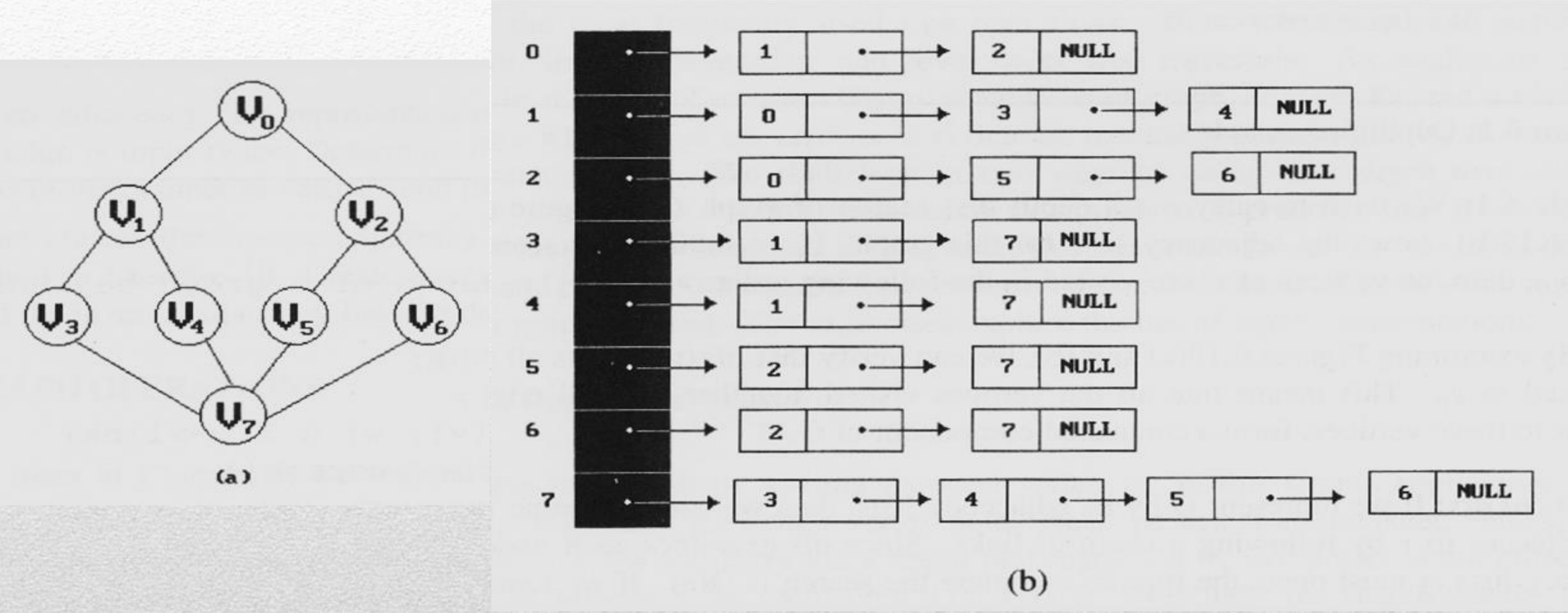
- **درخت پوشا**
- **مولفه های همبند دوطرفه**

عملیات روی گراف ها

E-mail: Hadi.khademi@gmail.com

▪ مثال پیمایش گراف با استفاده از لیست همسایگی

depth first search (DFS): v0, v1, v3, v7, v4, v5, v2, v6



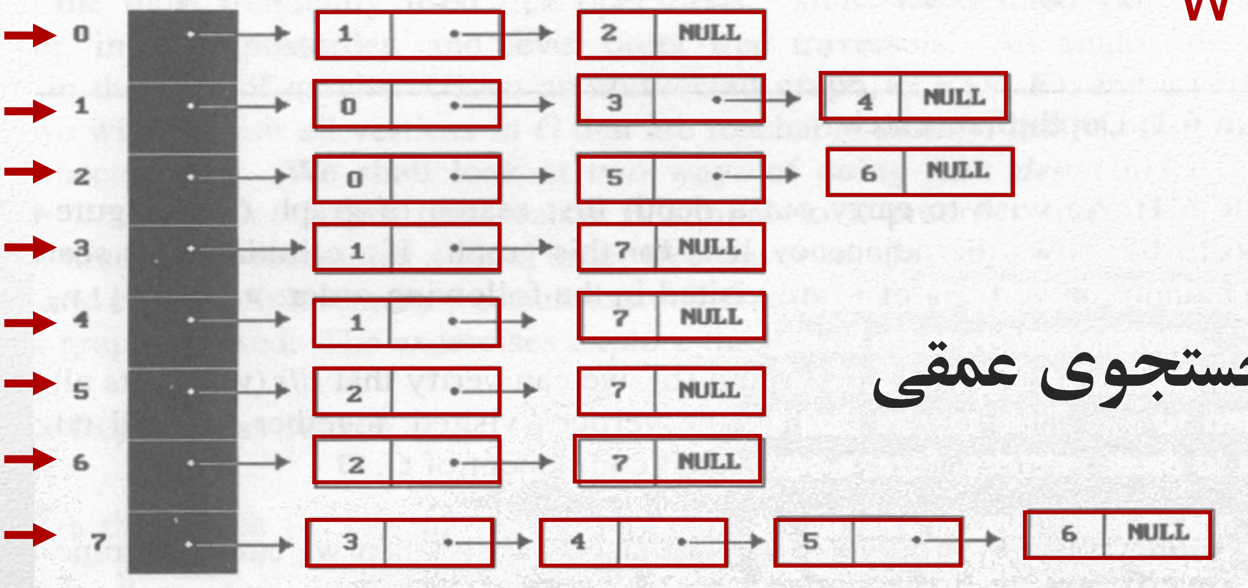
(a) (b)

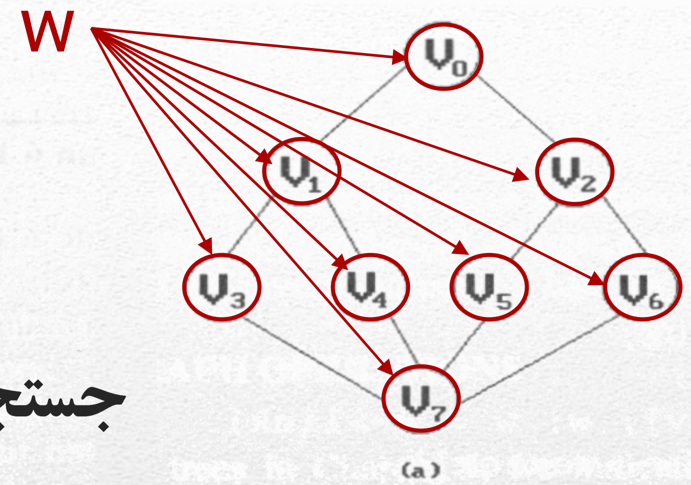
breadth first search (BFS): v0, v1, v2, v3, v4, v5, v6, v7

عملیات روی گراف ها

E-mail: Hadi.khademi@gmail.com

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

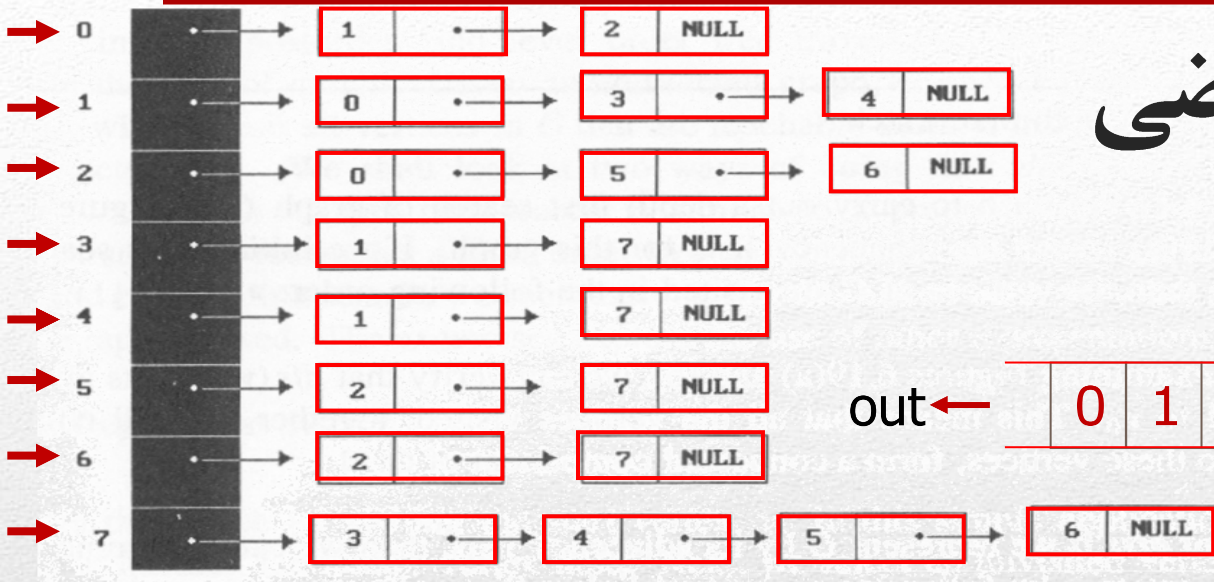
E-mail: Hadi.khademi@gmail.com

- **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 *);
```


E-mail: Hadi.khademi@gmail.com

جستجوی عرضی



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

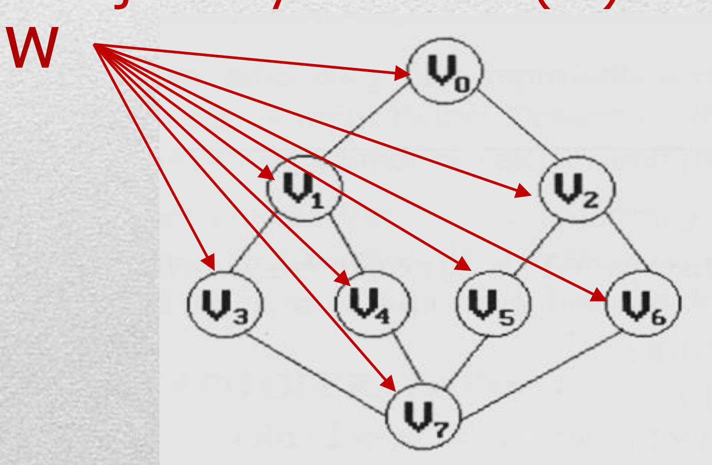
visited: ☒☒☒☒☒☒☒☒

out ← | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ← in

output: 0 1 2 3 4 5 6 7

adjacency list: O(e)

adjacency matrix: $O(n^2)$



(a)

```
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;
      }
   }
```

E-mail: Hadi.khademi@gmail.com

- **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

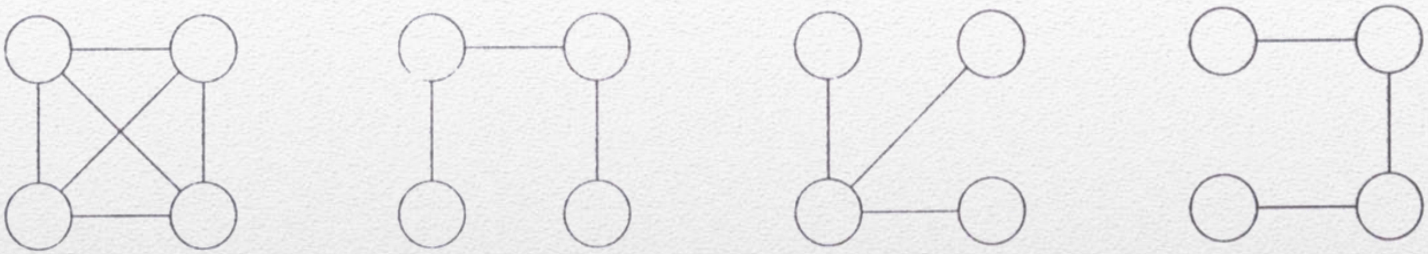
E-mail: Hadi.khademi@gmail.com

- **درخت های پوشا**
- چنانچه G یک گراف همبند باشد، آنگاه 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

E-mail: Hadi.khademi@gmail.com

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

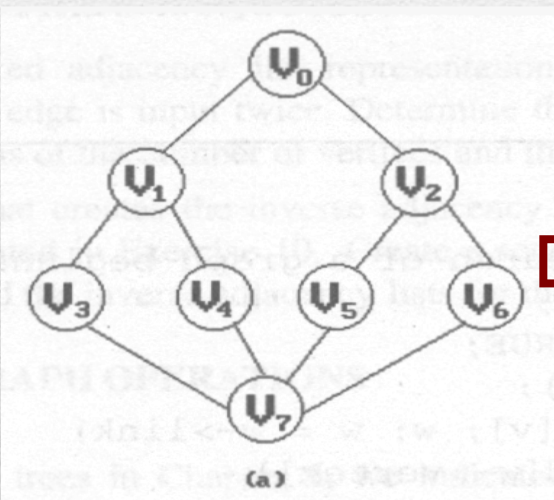


یک گراف کامل و سه درخت از درخت های پوشای آن

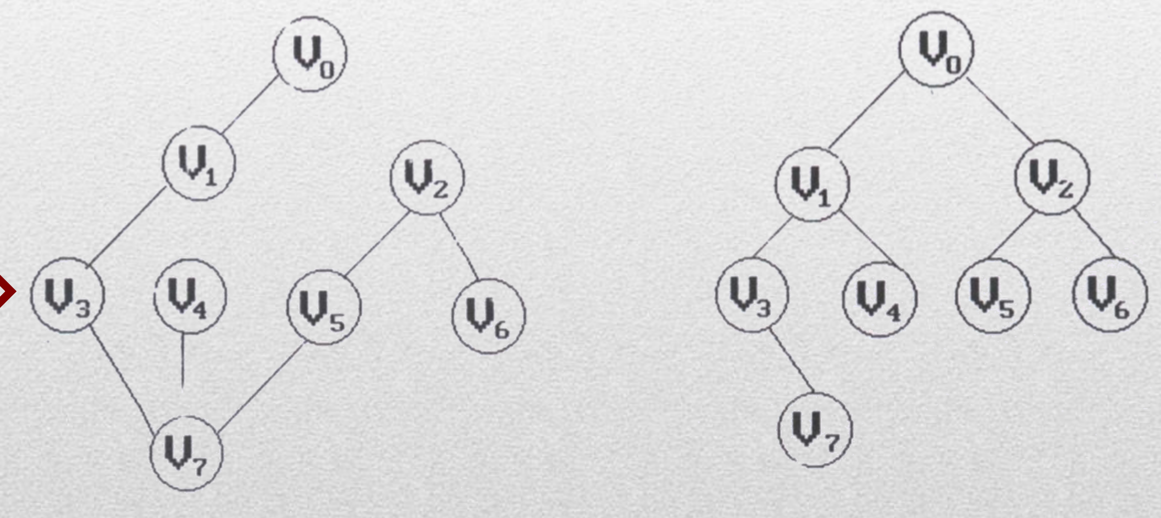
عملیات روی گراف ها

E-mail: Hadi.khademi@gmail.com

- **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



(a)



(a) dfs(0) spanning tree

(b) bfs(0) spanning tree

عملیات روی گراف ها

E-mail: Hadi.khademi@gmail.com

▪ 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=Tυ { (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

E-mail: Hadi.khademi@gmail.com

- **ویژگیهای درخت پوشا**
- یک درخت پوشا کوچکترین زیر گراف G’ از G است به طوری که V(G’) = V(G)
  - کوچکترین زیر گراف، زیر گرافي با حداقل تعداد یال تعریف مي شود
- 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.
- اگر یال غیر درختی مانند (v, w) به یک درخت پوشا مانند T اضافه شود آنگاه یک دور ایجاد می شود.

عملیات روی گراف ها

E-mail: Hadi.khademi@gmail.com

درختهای پوشا با کمترین هزینه

- درخت پوشای با کمترین هزینه، درخت پوشایی است که کمترین هزینه را دارد.
- سه الگوریتم مختلف برای به دست آوردن درخت پوشا با کمترین هزینه از یک گراف بدون جهت وجود دارد.
  - الگوریتم Kruskal
  - الگوریتم Prim
  - الگوریتم Sollin
- همه این الگوریتم ها از روش طراحی حریصانه (greedy method) استفاده می کنند.

E-mail: Hadi.khademi@gmail.com

- **راهکار حریصانه**
  - در هر مرحله بهترین تصمیم را بر اساس اطلاعات موجود در آن لحظه اتخاذ می کنیم. معمولا اتخاذ تصمیم در هر مرحله بر اساس کمترین هزینه یا بیشترین سود استوار است.
    - برای ایجاد درخت های پوشا با کمترین هزینه از معیار کمترین هزینه استفاده می کنیم.
  - در مراحل بعدی نمی توانیم تصمیم های قبل را عوض کنیم بنابراین باید مطمئن شویم این تصمیم منجر به راه حل معتبر می شود.
  - یک راه حل معتبر بر اساس قیدهای بیان شده در مساله تعیین می شود.
  - راه حل ما باید در شرایط زیر صدق کند:
  - تنها باید از یالهای گراف استفاده کند.
  - باید دقیقا از n-1 یال استفاده کند.
  - از یالهایی که دور ایجاد می کنند نمی توان استفاده کرد.

درختهای پوشا با کمترین هزینه

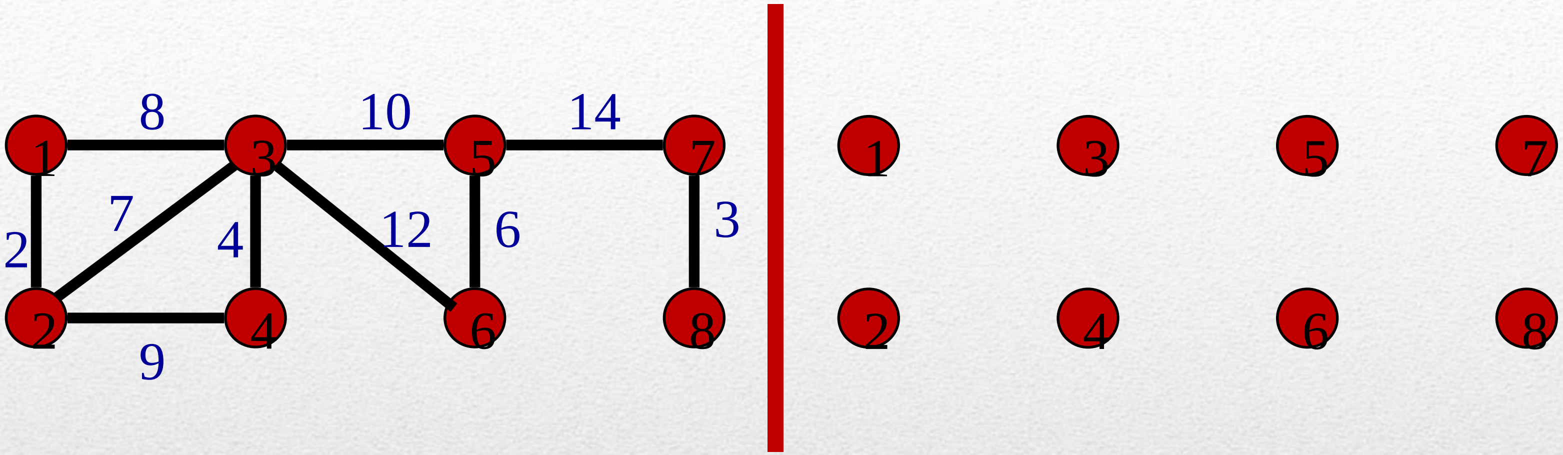
E-mail: Hadi.khademi@gmail.com

- **الگوریتم Kruskal**
  - درخت پوشا با کمترین هزینه را با اضافه کردن یک یال در هر مرحله می سازد.
  - یال ها برای اضافه شدن به T به ترتیب غیر نزولی هزینه شان انتخاب می شوند.
  - یک یال به T اضافه می شود مشروط به اینکه با یالهایی که قبلا در T بوده اند دور تشکیل ندهد.
  - از آنجا که گراف G همبند است و $n > 0$ راس دارد دقیقا n-1 یال برای اضافه شدن به T انتخاب می شود.

  - قضیه : اگر G یک گراف همبند بدون جهت باشد آنگاه الگوریتم Kruskal یک درخت پوشا با کمترین هزینه را تولید می کند.
  - Time complexity: O(e log e)

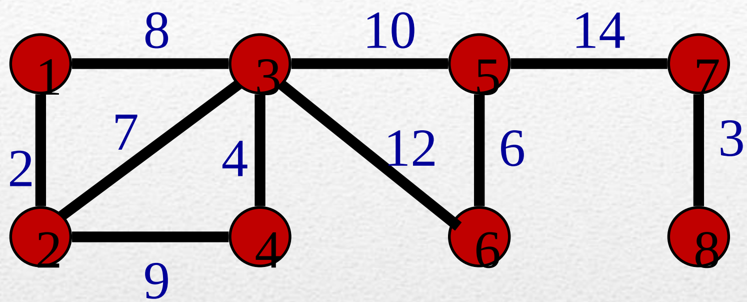
درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

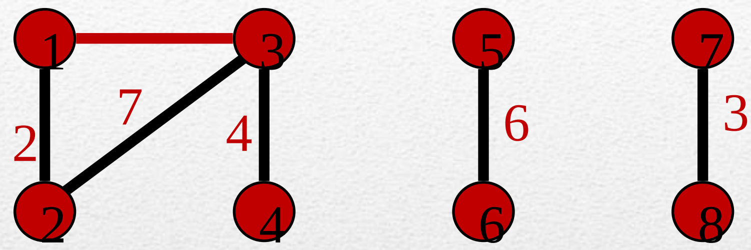
E-mail: Hadi.khademi@gmail.com



- با یک جنگل که هیچ یالی ندارد شروع کنید
- لبه ها را به ترتیب صعودی وزن آنها انتخاب کنید.
- لبه (1,2) انتخاب شده و به جنگل اضافه می شود.

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

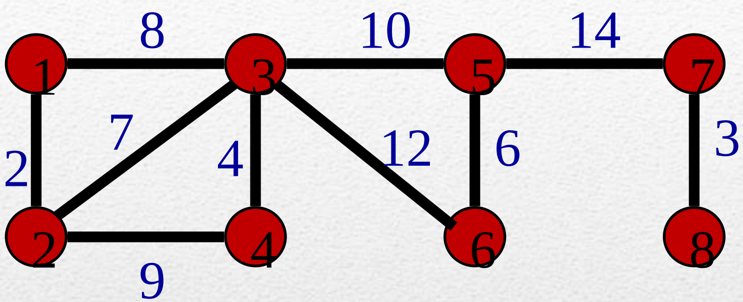


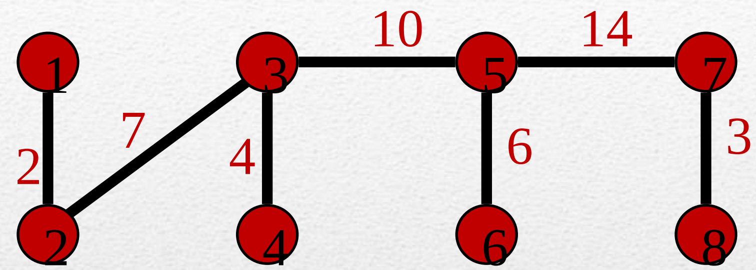


- در مرحله بعد لبه (7,8) انتخاب شده و اضافه می شود.
- در مرحله بعد لبه (3,4) انتخاب شده و اضافه می شود.
- در مرحله بعد لبه (5,6) انتخاب شده و اضافه می شود.
- در مرحله بعد لبه (2,3) انتخاب شده و اضافه می شود.
- در مرحله بعد لبه (1,3) انتخاب شده و به دلیل آنکه دور ایجاد می کند حذف می شود.

درختهای پوشا با کمترین هزینه (الگوریتم **Kruskal**)

E-mail: Hadi.khademi@gmail.com

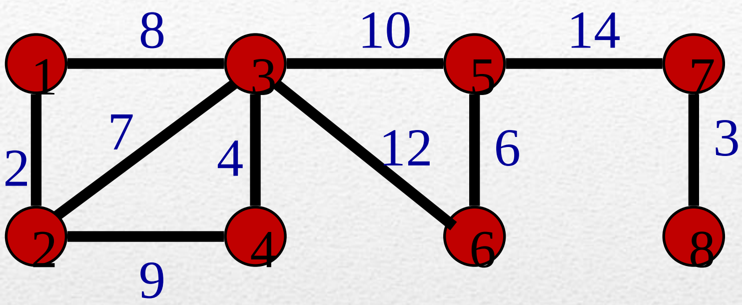


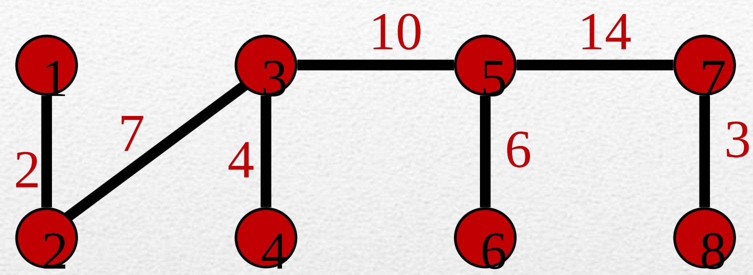


- در مرحله بعد لبه (2,4) انتخاب شده و به دلیل آنکه دور ایجاد می کند حذف می شود.
- در مرحله بعد لبه (3,5) انتخاب شده و اضافه می شود.
- در مرحله بعد لبه (3,6) انتخاب شده و به دلیل آنکه دور ایجاد می کند حذف می شود.
- در مرحله بعد لبه (5,7) انتخاب شده و اضافه می شود.

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

E-mail: Hadi.khademi@gmail.com





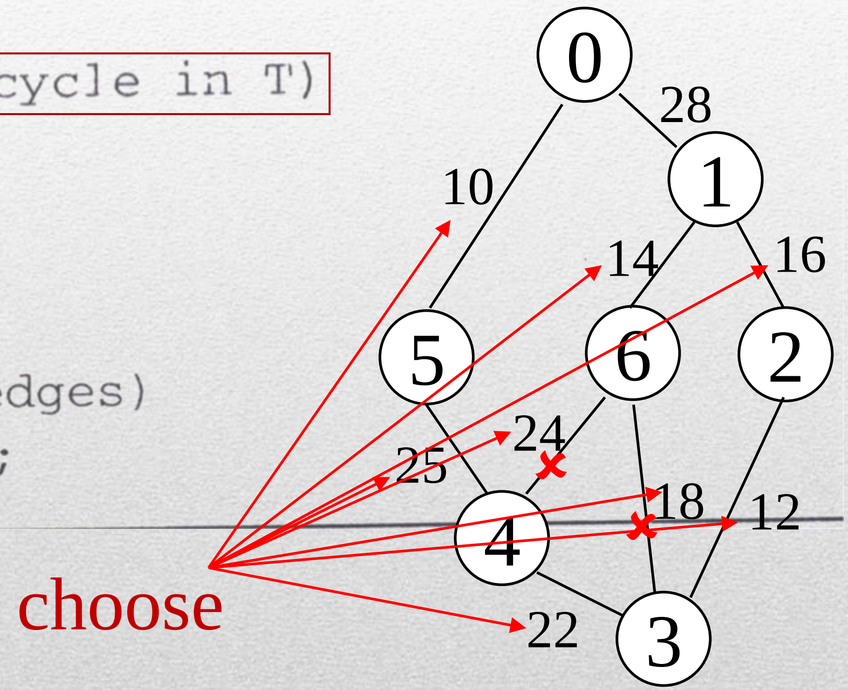
- n-1 یال انتخاب شد و هیچ دوری ایجاد نگردید.
- بنابراین ما یک درخت پوشا به دست آوردیم.
- هزینه این درخت 46 است.
- در صورتی که هزینه یالهای مختلف درخت متفاوت باشد درخت پوشای با کمترین هزینه یکتا است.

درختهای پوشا با کمترین هزینه (الگوریتم **Kruskal**)

E-mail: Hadi.khademi@gmail.com

▪ شبه کد الگوریتم Kruskal

```
T = {};
while (T contains less than n-1 edges && E is not empty) {
   choose a least cost edge (v,w) from E;
   delete (v,w) from E;
   if ((v,w) does not create a cycle in T)
      add (v,w) to T;
   else
      discard (v,w);
}
if (T contains fewer than n-1 edges)
   printf("No spanning tree\n");
```



E-mail: Hadi.khademi@gmail.com

- **ساختمان داده برای الگوریتم Kruskal**
- **مجموعه لبه ها E**
- **عملیات:**
  - آیا E تهی است؟
  - انتخاب و حذف یال با کمترین هزینه
- **می توان یالها را به ترتیب غیر نزولی در زمان O(e log e) مرتب کرد**
- **می توان از یک min heap برای نگهداری یالها استفاده کرد.**
  - ساختن اولیه heap : O(e)
  - تعیین و حذف یال با کمترین هزینه O(log e)

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

E-mail: Hadi.khademi@gmail.com

- **ساختمان داده برای الگوریتم Kruskal**
- **مجموعه یالهای انتخاب شده T**
- **عملیات:**
  - آیا T، n-1 یال دارد؟
  - آیا اضافه کردن یال (v,w) به T دور ایجاد می کند ؟
  - یک یال به T اضافه کنید.

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

E-mail: Hadi.khademi@gmail.com

- **ساختمان داده برای الگوریتم Kruskal**
  - نگهداری مجموعه یالهای انتخاب شده T در یک آرایه
  - آیا T، n-1 یال دارد؟
  - تعداد یالها را در ارایه بررسی می کنیم: $O(1)$

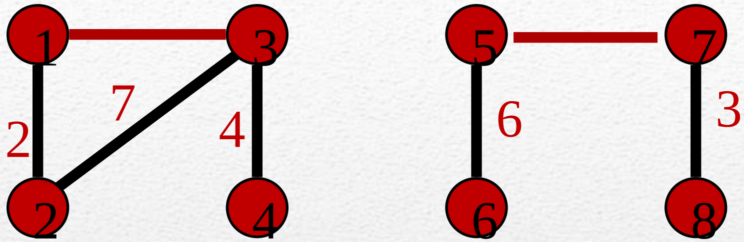
  - آیا اضافه کردن یال (v,w) به T دور ایجاد می کند ؟
  - آسان نیست

  - یک یال به T اضافه کنید.
  - آن را به انتهای آرایه اضافه می کنیم: $O(1)$

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

E-mail: Hadi.khademi@gmail.com

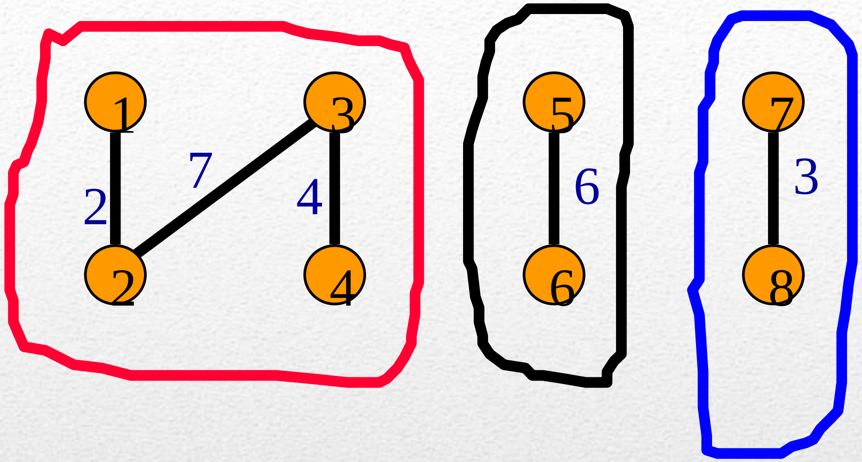
- ساختمان داده برای الگوریتم **Kruskal**



- آیا اضافه کردن یال (v, w) به T دور ایجاد می کند؟
- اجزای T در هر لحظه درخت هستند.
- وقتی v و w در یک جزء باشند اضافه کردن یال (v,w) دور ایجاد می کند.
- وقتی که v و w در دو جزء جدا باشند اضافه کردن یال (v,w) دور ایجاد نمی کند.

E-mail: Hadi.khademi@gmail.com

- ساختمان داده برای الگوریتم Kruskal



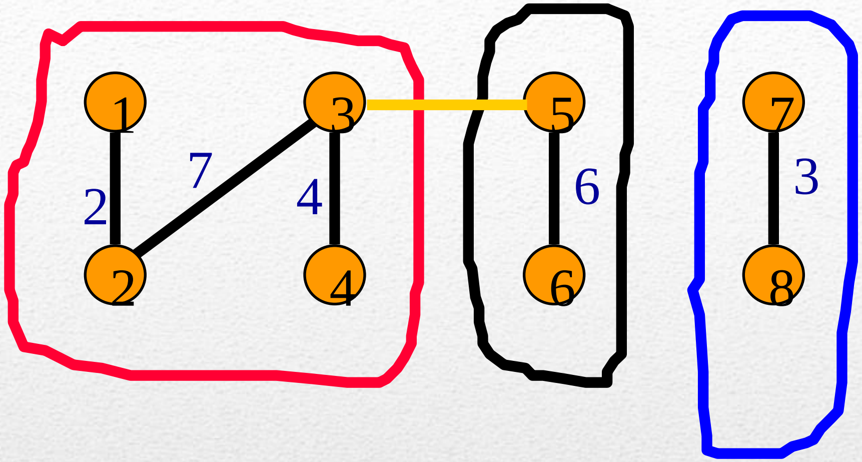
- هر جزء T با رئوس درون آن مشخص می شود.
- هر جزء را با مجموعه رئوس آن نشان می دهیم

{1, 2, 3, 4}, {5, 6}, {7, 8}

- دو راس در یک جزء هستند اگر درون یک مجموعه باشند.

E-mail: Hadi.khademi@gmail.com

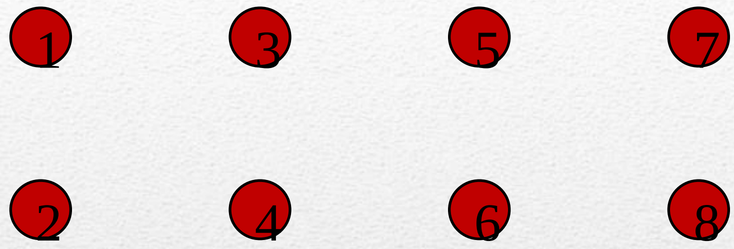
- ساختمان داده برای الگوریتم **Kruskal**



- هنگامی که یال (v,w) به T اضافه می شود، دو جزئی که شامل v و w بوده اند با هم ترکیب می شوند تا یک جزء را تشکیل دهند.
- چنانچه مجموعه ها برای بازنمایی اجزای T به کار روند، مجموعه های شامل v و w را با هم اجتماع می گیریم.
- {1, 2, 3, 4} + {5, 6} => {1, 2, 3, 4, 5, 6}

E-mail: Hadi.khademi@gmail.com

- ساختمان داده برای الگوریتم **Kruskal**
- در ابتدا T تهی است.



- مجموعه ها در ابتدا به صورت:
  - {1} {2} {3} {4} {5} {6} {7} {8}
- آیا اضافه کردن یال (v,w) به T دور ایجاد می کند ؟ اگر دور ایجاد نمی کند آن را اضافه کنید.

```
s1 = Find(v); s2 = Find(w);
if (s1 != s2) Union(s1, s2);
```

درختهای پوشا با کمترین هزینه (الگوریتم **Kruskal**)

E-mail: Hadi.khademi@gmail.com

- **ساختمان داده برای الگوریتم Kruskal**

- استفاده از توابع سریع برای پیاده سازی مجموعه ها
  - مقدار دهی اولیه:

$O(n)$

  - حداکثر 2e عمل find و n-1 عمل اجتماع:

خیلی نزدیک به $O(n+e)$

  - عملیات روی min heap برای انتخاب یالها به ترتیب غیر نزولی

$O(e \log e)$

  - پیچیدگی زمانی الگوریتم Kruskal :

$O(n + e \log e)$

درختهای پوشا با کمترین هزینه (الگوریتم Kruskal)

E-mail: Hadi.khademi@gmail.com

- **الگوریتم Prim**
  - درخت پوشا با کمترین هزینه را با اضافه کردن یک یال در هر مرحله می سازد.
  - یال ها برای اضافه شدن به T به ترتیب غیر نزولی هزینه شان انتخاب می شوند.
  - در تمام مراحل الگوریتم مجموعه یالهای انتخاب شده یک درخت را تشکیل می دهند.
  - توجه در الگوریتم Kruskal مجموعه یالهای انتخاب شده در هر مرحله یک جنگل را می سازند.

درختهای پوشا با کمترین هزینه (الگوریتم Prim)

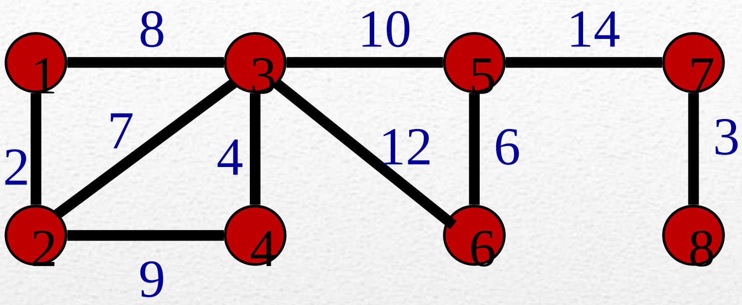
E-mail: Hadi.khademi@gmail.com

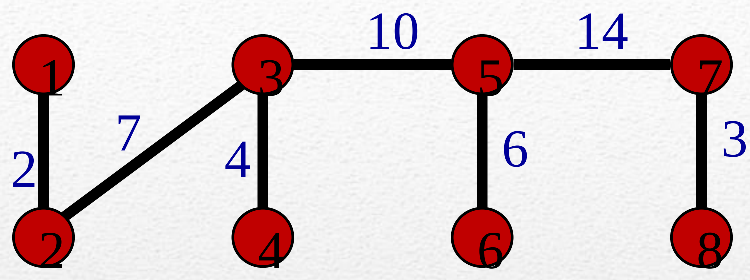
- **الگوریتم Prim**
  - در ابتدا T تنها شامل یک راس است.
  - یال (u,v) با حداقل هزینه را به گونه ای می یابیم که با اضافه شدن آن به T ، T درخت باقی بماند.
  - یال (u,v) به گو نه ای است که تنها یکی از رئوس u و یا v در T باشد.
  - اضافه کردن یال را ادامه می دهیم تا T شامل n-1 یال شود.

- **الگوریتم Prim برای هر گراف بدون جهت همبند درخت پوشا با کمترین هزینه را پیدا می کند .**

درختهای پوشا با کمترین هزینه (الگوریتم Prim)

E-mail: Hadi.khademi@gmail.com





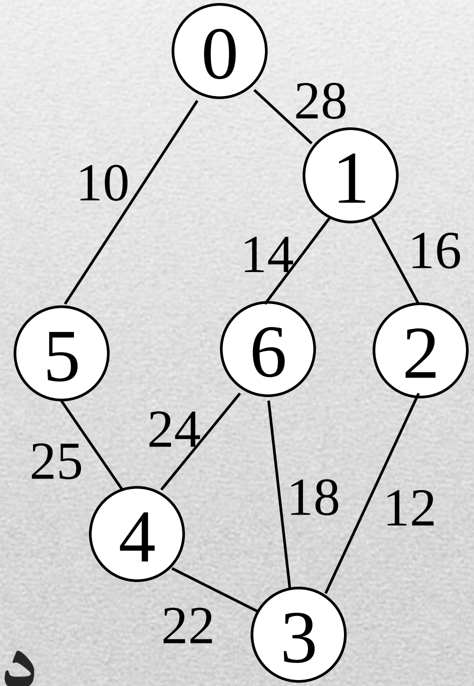
- با هر راس دلخواهی شروع کنید
- یک درخت با دو راس از طریق اضافه کردن کم هزینه ترین یال ایجاد کنید.
- یک درخت با سه راس از طریق اضافه کردن کم هزینه ترین یال ایجاد کنید.
- درهر لحظه یک یال به درخت اضافه کنید تا درختی با n-1 یال به دست بیاید (اگر درخت n-1 یال داشته باشد شامل همه ی n راس خواهد بود).

درختهای پوشا با کمترین هزینه (الگوریتم Prim)

E-mail: Hadi.khademi@gmail.com

▪ شبه کد الگوریتم Prim

```
T = {};
TV = {0}; /* start with vertex 0 and no edges */
while (T contains fewer than n-1 edges) {
   let (u, v) be a least cost edge such that u ∈ TV and
   v ∉ TV;
   if (there is no such edge)
     break;
   add v to TV;
   add (u, v) to T;
}
if (T contains fewer than n-1 edges)
   printf("No spanning tree\n");
```



درختهای پوشا با کمترین هزینه (الگوریتم Prim)

- می توان ثابت کرد که همه ی روشهای ذکر شده منجر به درخت پوشای کمترین هزینه می شوند.
- سریع ترین روش، روش Prim است
  - O(n2) با استفاده از پیاده سازی مشابه الگوریتم Dijkstra برای کوتاهترین مسیر
  - O(e + n log n) با استفاده از Fibonacci heap
- الگوریتم Kruskal از درختهای union-find برای رسیدن به پیچیدگی O(n + e log e) استفاده می کند

روشهای ایجاد درخت پوشا با کمترین هزینه

E-mail: Hadi.khademi@gmail.com