جستجو در درختها و گرافها

مطالب مورد بحث

- روشهای پیمایش درختهای دودویی
- روشهای پیمایش گرافها

روشهای پیمایش درختهای دودویی

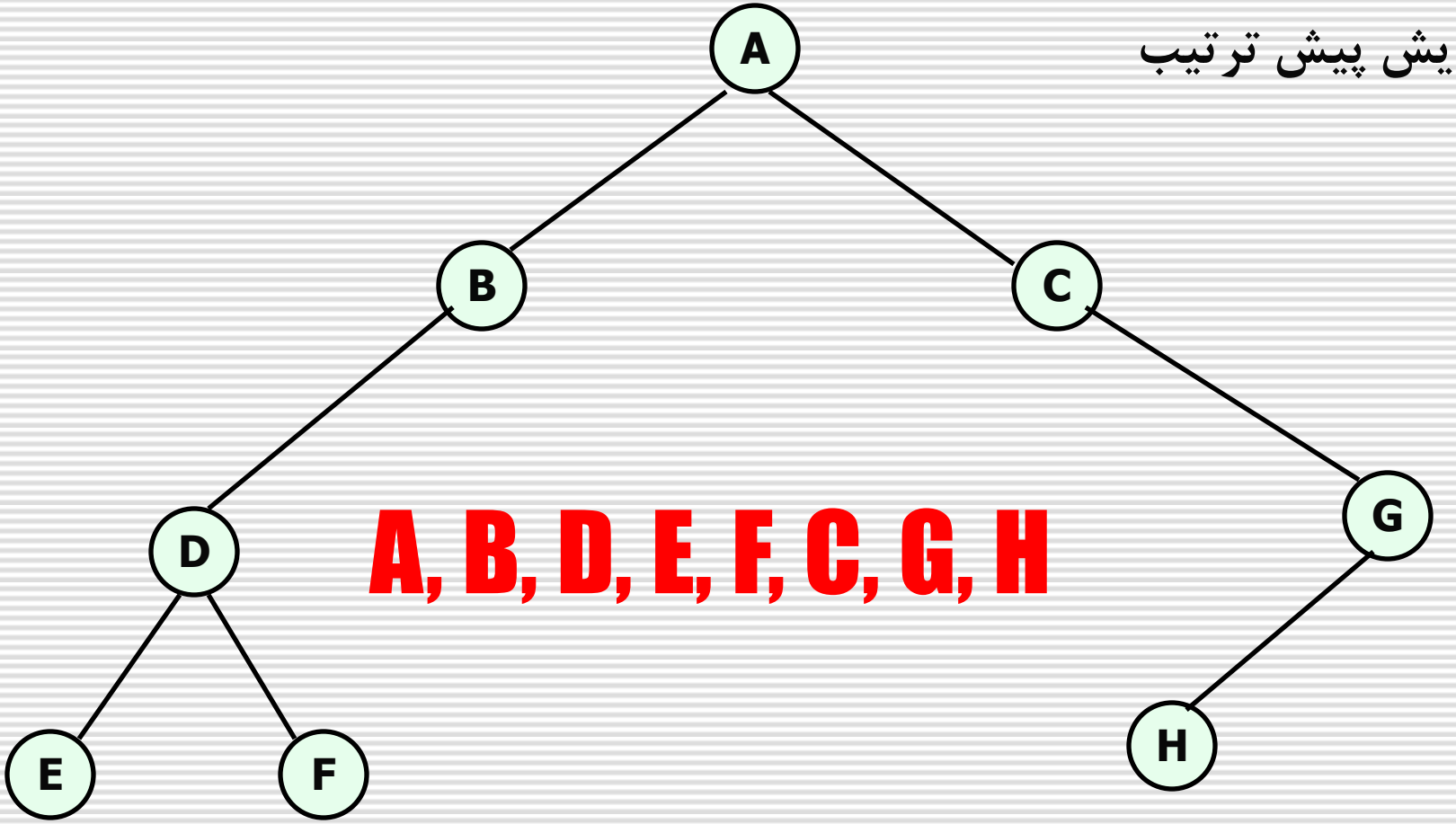
- درخت دودویی: درختی که درجه هر گره آن حداکثر برابر ۲ باشد. یعنی هر گره آن حداکثر دارای ۲ فرزند باشد.
- پیمایش درخت دودویی: روشی سیستماتیک برای ملاقات تمام گره های درخت دودویی
- سه روش عمده
  - پیمایش پیش ترتیب (preorder traversal)
  - پیمایش میان ترتیب (inorder traversal)
  - پیمایش پس ترتیب (postorder traversal)
  - سه روش فوق روشهای پیمایش چپ به راست هستند. یعنی زیردرخت سمت چپ یک گره نسبت به زیردرخت سمت راست آن گره، برای ملاقات اولویت دارد.

روشهای پیمایش درختهای دودویی

- پیمایش پیش ترتیب درخت T
  - اگر درخت T تهی است، کار تمام است. در غیر اینصورت
  - ابتدا ریشه درخت T را ملاقات می کنیم.
  - سپس زیردرخت سمت چپ درخت T را با استفاده از همین الگوریتم بطور پیش ترتیب پیمایش می کنیم.
  - سپس زیردرخت سمت راست درخت T را با استفاده از همین الگوریتم بطور پیش ترتیب پیمایش می کنیم.

روشهای پیمایش درختهای دودویی

پیمایش پیش ترتیب



روشهای پیمایش درختهای دودویی

```
void  preorder(binaryTree T) {
   if(T is not empty) {
         visit(T.root);
         preorder(T.leftSubtree);
         preorder(T.rightSubtree);
   }
}
```

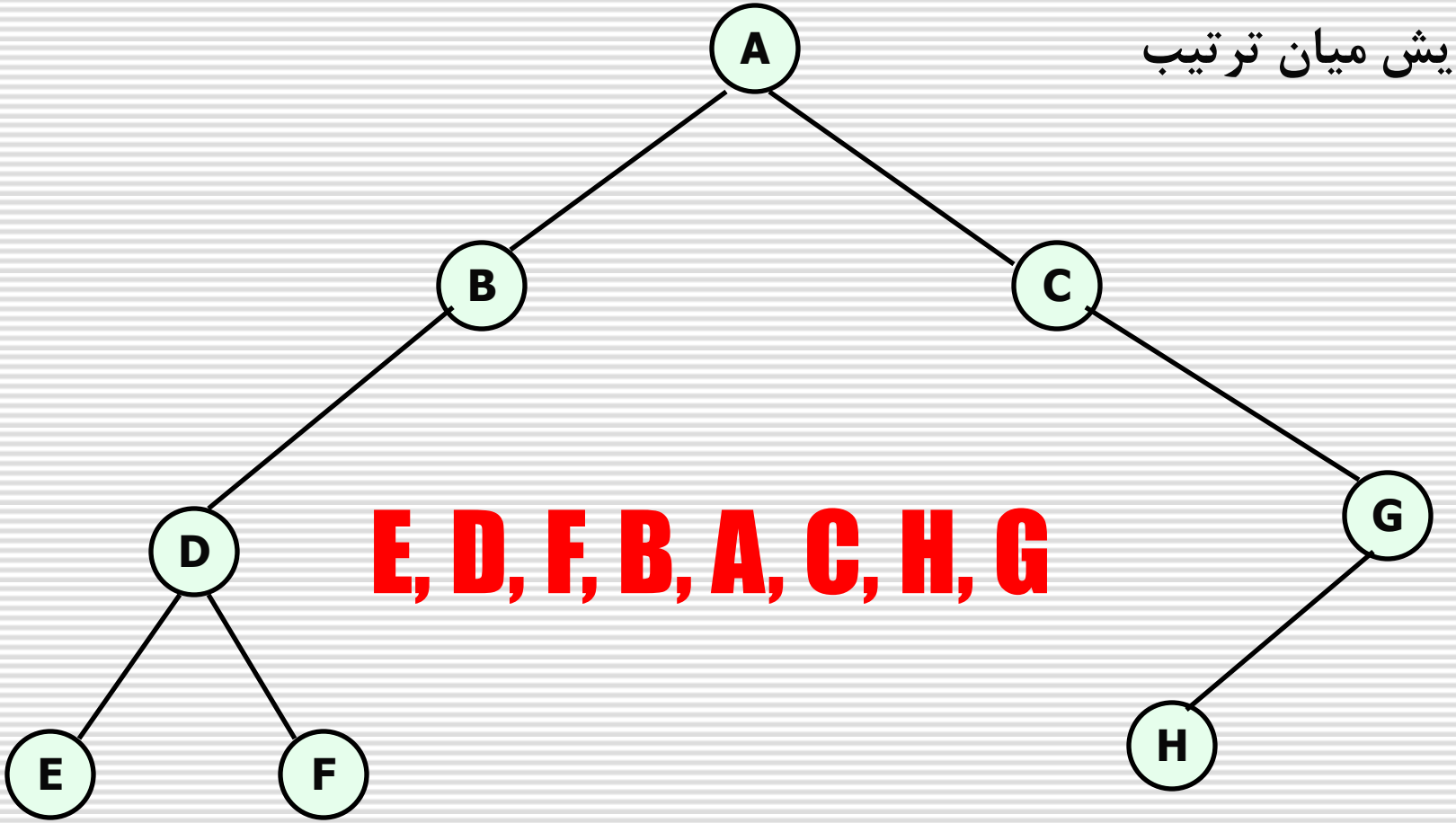
مرتبه زمانی؟

روشهای پیمایش درختهای دودویی

- پیمایش میان ترتیب درخت T
  - اگر درخت T تهی است، کار تمام است. در غیر اینصورت
  - زیردرخت سمت چپ درخت T را با استفاده از همین الگوریم بطور میان ترتیب پیمایش می کنیم.
  - سپس ریشه درخت T را ملاقات می کنیم.
  - سپس زیردرخت سمت راست درخت T را با استفاده از همین الگوریتم بطور میان ترتیب پیمایش می کنیم.

روشهای پیمایش درختهای دودویی

پیمایش میان ترتیب



روشهای پیمایش درختهای دودویی

```
void  inorder(binaryTree T) {
    if(T is not empty) {
        inorder(T.leftSubtree);
        visit(T.root);
        inorder(T.rightSubtree);
    }
}
```

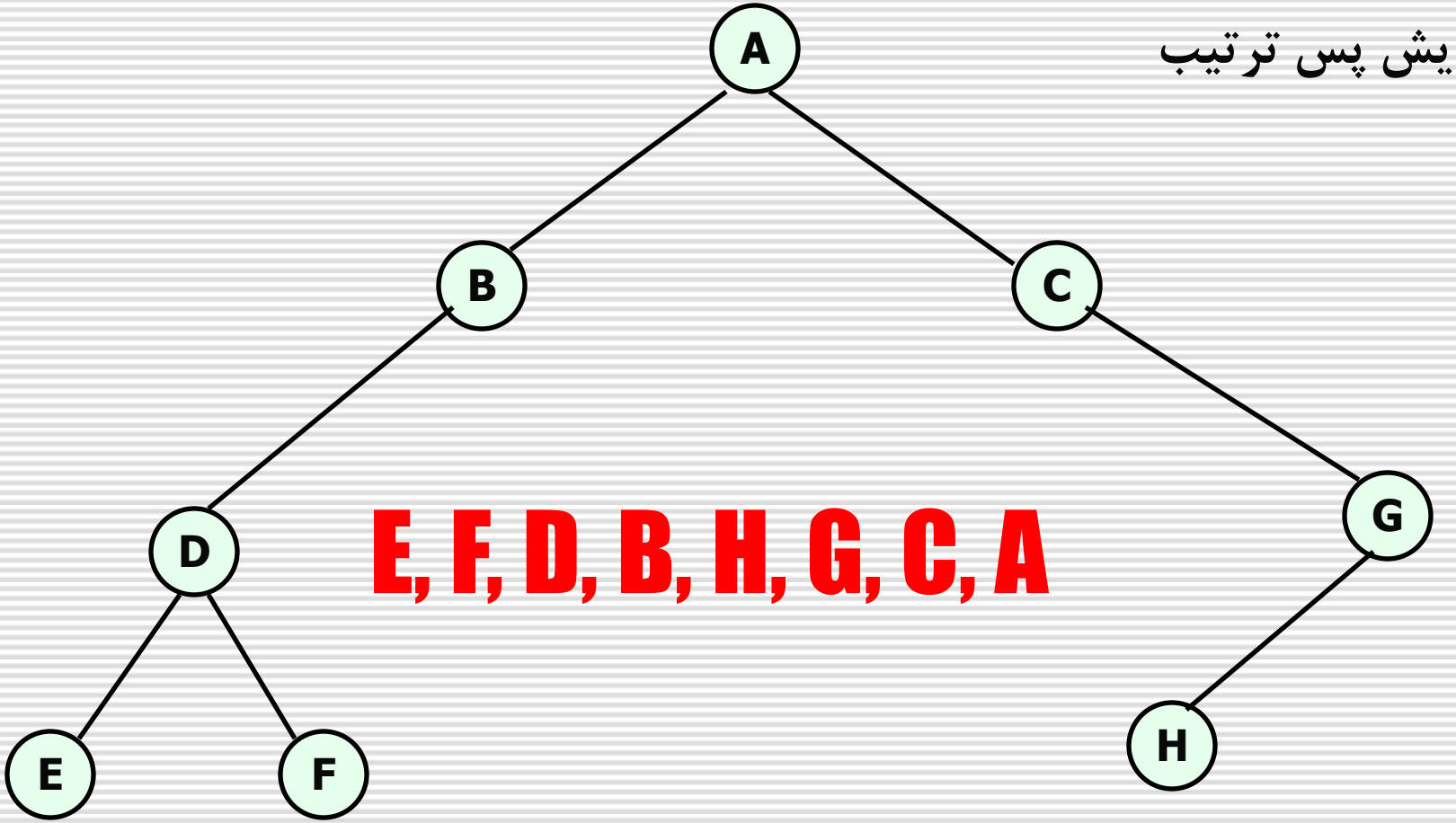
مرتبه زمانی؟

روشهای پیمایش درختهای دودویی

- پیمایش پس ترتیب درخت T
  - اگر درخت T تهی است، کار تمام است. در غیر اینصورت
  - زیردرخت سمت چپ درخت T را با استفاده از همین الگوریم بطور پس ترتیب پیمایش می کنیم.
  - سپس زیردرخت سمت راست درخت T را با استفاده از همین الگوریتم بطور پس ترتیب پیمایش می کنیم.
  - سپس ریشه درخت T را ملاقات می کنیم.

روشهای پیمایش درختهای دودویی

پیمایش پس ترتیب



روشهای پیمایش درختهای دودویی

```
void  postorder(binaryTree T) {
    if(T is not empty) {
        postorder(T.leftSubtree);
        postorder(T.rightSubtree);
        visit(T.root);
    }
}
```

مرتبه زمانی

روش های پیمایش گرافها

- گراف G=(V, E) داده شده است. می خواهیم از یک گره آغاز و کلیه گره های قابل دسترس را ملاقات کنیم.
- دو روش
  - جستجو به ترتیب پهنا (سطح-اول)(BFS: Breadth First Search)
  - جستجو به ترتیب عمق (عمق-اول) (DFS: Depth First Search)
- پیش از پرداختن به این موضوع، مقدماتی در مورد نحوه نمایش گرافها (graph representation) ذکر می کنیم.

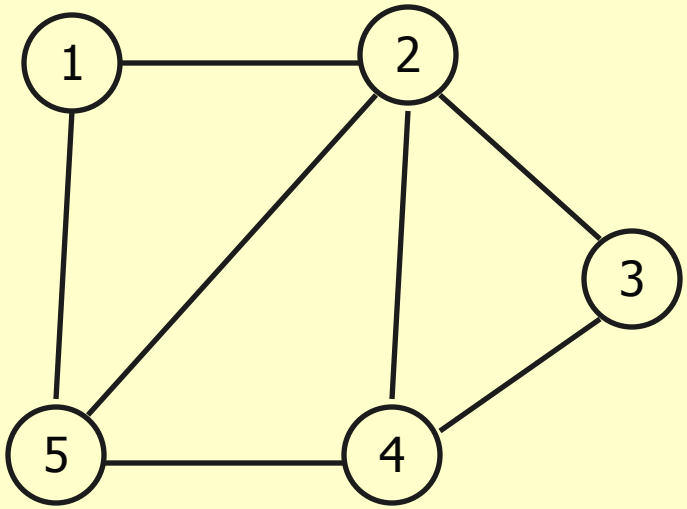
روشهای نمایش گرافها

- دو روش عمده
  - لیست مجاورت: Adjacency List
  - ماتریس مجاورت: Adjacency Matrix

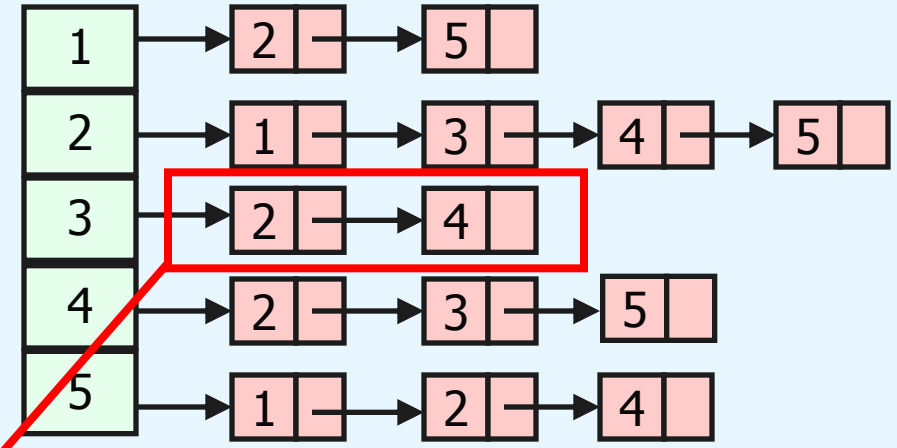
روشهای نمایش گرافها

- لیست مجاورت
  - لیست مجاورت گراف G=(V,E) که |V|=n، شامل
  - یک آرایه n عنصری (به ازای هر گره گراف یک عنصر)
  - عنصر i ام این آرایه، شامل لیستی از گره های مجاور گره i در گراف می باشد.

لیست مجاورت



گراف بدون جهت G=(V,E)

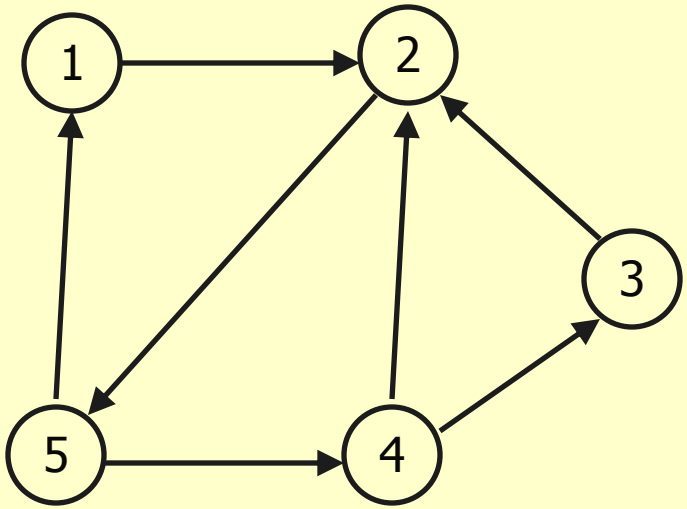


نمایش گراف **G** با استفاده از لیست مجاورت

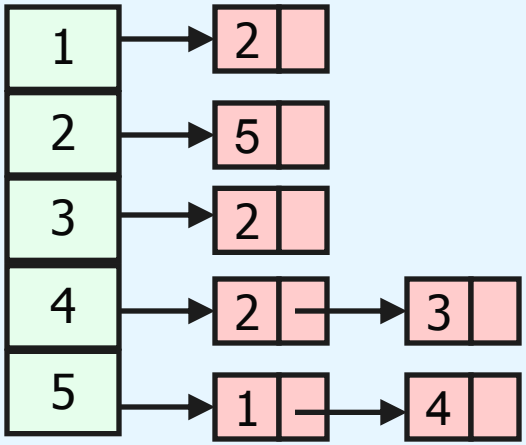
میزان فضای لازم برای نمایش لیست مجاورت گراف؟

لیست مجاورت گره **3**

لیست مجاورت



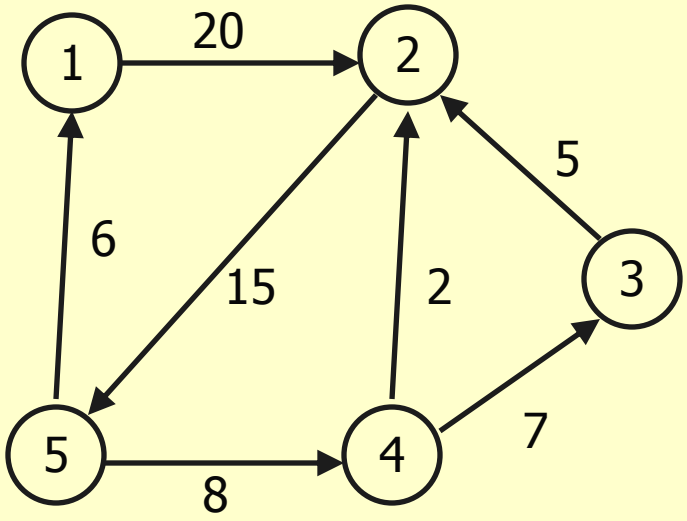
گراف جهتدار G=(V,E)



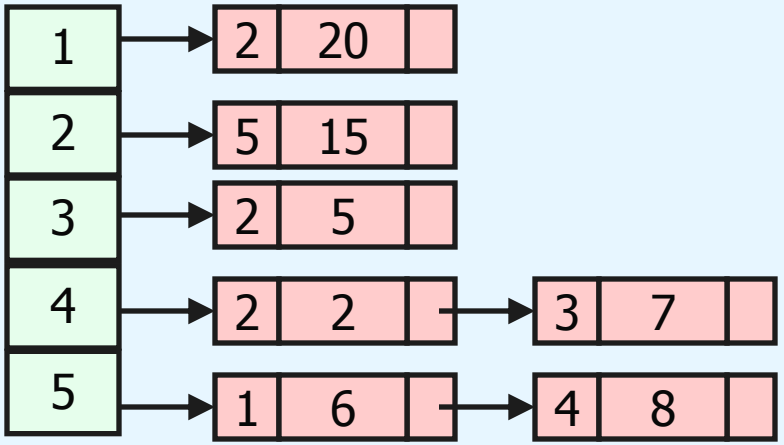
نمایش گراف **G** با استفاده از لیست مجاورت

میزان فضای لازم برای نمایش لیست مجاورت گراف؟

لیست مجاورت



گراف وزندار و جهت دار G=(V,E)

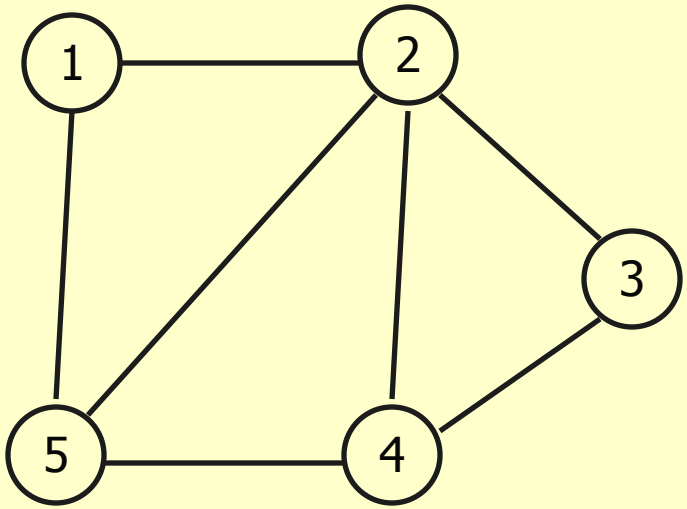


نمایش گراف **G** با استفاده از لیست مجاورت

روشهای نمایش گرافها

- ماتریس مجاورت گراف G=(V,E) که n=|V|، عبارت است از یک آرایه دو بعدی دارای n سطر و n ستون
- مقدار عنصر [i][j]
  - اگر گراف وزندار باشد، آنگاه
    - اگر گراف جهت دار باشد، برابر است با وزن یالی که گره i را به گره j وصل می کند. (اگر چنین یالی وجود نداشت برابر است با صفر)
    - اگر گراف بدون جهت باشد، برابر است با وزن یال بین گره های i و j (اگر چنین یالی وجود نداشت برابر است با صفر)
  - اگر گراف بدون وزن باشد، آنگاه
    - اگر گراف جهت دار باشد، چنانچه از گره i به گره j یالی وجود داشته باشد، برابر است با ۱ و چنانچه یالی وجود نداشته باشد برابر است با صفر.
    - اگر گراف بدون جهت باشد، چنانچه بین گره i و گره j یالی وجود داشته باشد برابر است با ۱ و در غیر اینصورت برابر است با صفر.

ماتریس مجاورت



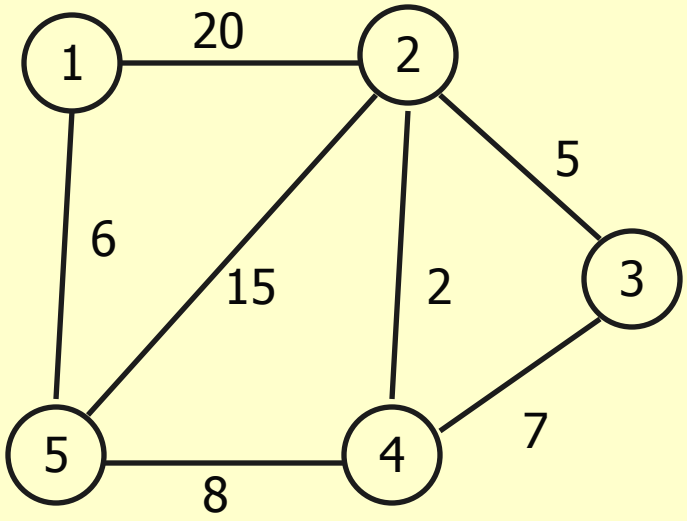
گراف بدون وزن و بدون جهت

G=(V,E)

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

نمایش گراف **G** با استفاده ازماتریس مجاورت

ماتریس مجاورت



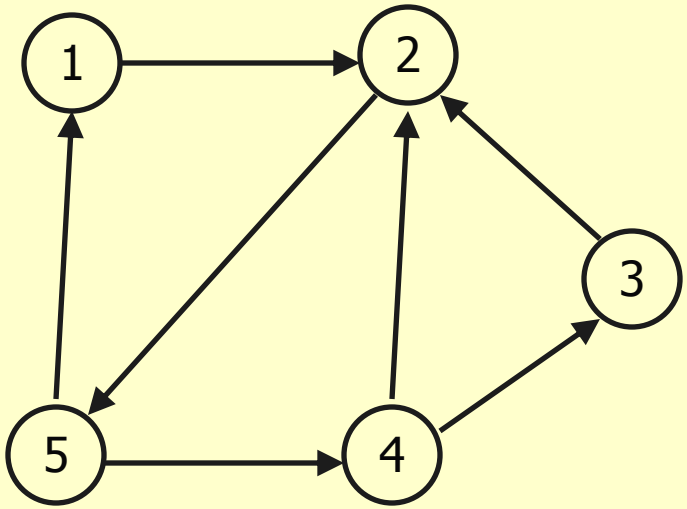
گراف وزندار و بدون جهت

G=(V,E)

	1	2	3	4	5
1	0	20	0	0	6
2	20	0	5	2	15
3	0	5	0	7	0
4	0	2	7	0	8
5	6	15	0	8	0

نمایش گراف **G** با استفاده ازماتریس مجاورت

ماتریس مجاورت



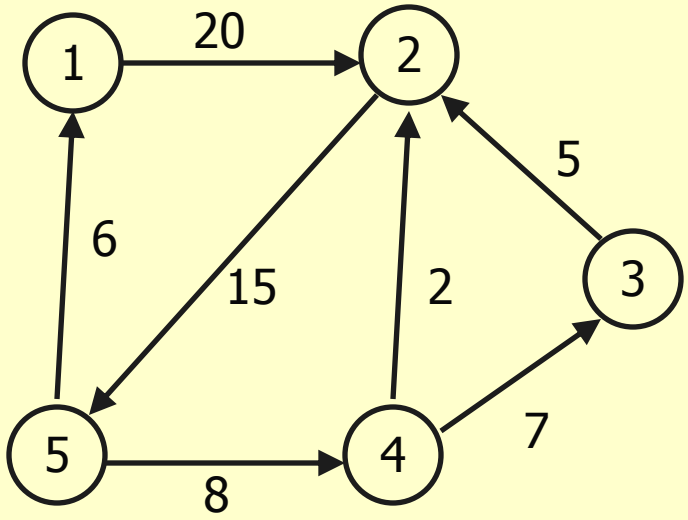
گراف بدون وزن و جهتدار

G=(V,E)

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

نمایش گراف **G** با استفاده ازماتریس مجاورت

ماتریس مجاورت



گراف وزندار و جهت دار

G=(V,E)

	1	2	3	4	5
1	0	20	0	0	0
2	0	0	0	0	15
3	0	5	0	0	0
4	0	2	7	0	0
5	6	0	0	8	0

نمایش گراف **G** با استفاده ازماتریس مجاورت

نکات

- از نظر میزان مصرف حافظه (space)
  - لیست مجاورت چون متناسب با تعداد یالهای گراف، فضا می گیرد. اما ماتریس مجاورت برای یک گراف که n گره دارد، در هر حالتی، یک آرایه n * n است و به تعداد یالهای موجود ربطی ندارد. در نتیجه برای گرافهای spars، استفاده از لیست مجاورت بهتر است.
- از نظر پیچیدگی مکانی (space complexity)
  - لیست مجاورت: $O(|V| + |E|)$
  - ماتریس مجاورت: $O(|V|^2)$
  - $|V|$ یعنی تعداد گره های گراف
  - $|E|$ یعنی تعداد یالهای گراف

نکات

- برای تشخیص اینکه آیا بین دو گره i و j گراف یالی وجود دارد یا خیر (تشخیص مجاورت دو گره)
  - با استفاده از ماتریس مجاورت، فقط به یک lookup نیاز دارد باید عنصر [i][j] ماتریس مجاورت را بررسی کنیم که با O(1) قابل انجام است.
  - با استفاده از لیست مجاورت، باید لیست مجاورت گره i را بررسی کنیم که این کار با O(n) انجام می شود که n تعداد گره های گراف است.
  - در نتیجه از نظر تشخیص مجاورت دو گره، ماتریس مجاورت کاراتر است.
- به دلیل مشابه، بروزرسانی عناصر ماتریس مجاورت ساده تر از بروزرسانی عناصر لیست مجاورت است.

نکات

- اگر گراف وزن دار باشد، می توان از همان ماتریس مجاورت برای ذخیره وزن یالها نیز استفاده کرد، در این صورت بجای استفاده از دو مقدار 0 و 1، از وزن خود یالها برای مقداردهی عناصر ماتریس مجاورت استفاده می کنیم.
- اما اگر بخواهیم گراف وزن داری را با لیست مجاورت نمایش دهیم باید به هر یک از گره های لیست مجاورت فیلد جدید بیافزاییم تا وزن یال را در خود ذخیره کند.

نکات

- اگر گراف وزن دار نباشد، می توان در پیاده سازی ماتریس مجاورت، به هر عنصر ماتریس، یک بیت تخصیص داد. در نتیجه میزان فضای گرفته شده کاهش می یابد و حجم ماتریس عملا فشرده می شود. چنین عملی برای لیست مجاورت قابل انجام نیست.
- پیاده سازی ماتریس مجاورت نسبت به لیست مجاورت ساده تر است.

نکات

- اگر برای **یک گره** بخواهیم گره های مجاورش را پیدا کنیم،
  - در لیست مجاورت، باید کل لیست مجاورت آن عنصر را پیمایش کنیم که این کار با $O(|V|)$ قابل انجام است.
- اگر بخواهیم برای **کل گره ها**، گره های مجاورشان را پیدا کنیم، باید کل لیست مجاورت را پیمایش کنیم.
  - لیست مجاورت شامل $|E|$ یال (برای گراف جهت دار) و یا $2*|E|$ یال (برای گراف بدون جهت) می باشد.
  - در نتیجه اگر بخواهیم برای تمام گره های گراف، گره های مجاورشان را پیدا کنیم، این کار با استفاده از لیست مجاورت، با $O(|E|)$ قابل انجام است.

نکات

- اگر برای **یک گره** بخواهیم گره های مجاورش را پیدا کنیم،
  - در ماتریس مجاورت، باید کل سطر مربوط به آن عنصر را پیمایش کنیم که این کار با $O(|V|)$ قابل انجام است.
- اگر بخواهیم برای **کل گره ها**، گره های مجاورشان را پیدا کنیم، باید کل ماتریس مجاورت (اگر گراف جهتدار باشد) یا نصف ماتریس مجاورت (اگر گراف بدون جهت باشد) را پیمایش کنیم.
  - در نتیجه اگر بخواهیم برای تمام گره های گراف، گره های مجاورشان را پیدا کنیم، این کار با استفاده از ماتریس مجاورت، با $O(|V|^2)$ قابل انجام است.

جستجو به ترتیب سطح اول: BFS

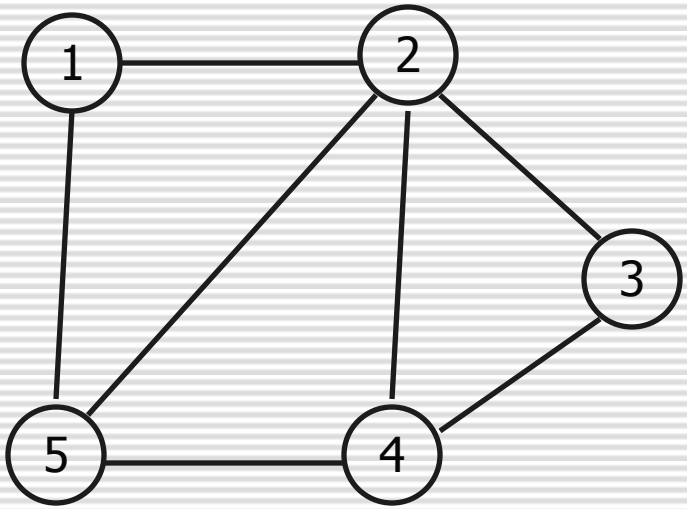
- گراف G=(V,E) و یک گره متمایز گراف را بعنوان نقطه شروع جستجو داریم (گره s). جستجوی به ترتیب سطح این گراف، بطور سیستماتیک، تمام گره هایی را که از گره s قابل دسترس هستند را پیدا می کند.
- منظور از سیستماتیک: گره های قابل دسترس از s، بر حسب فاصله شان تا گره s، به ترتیب از نزدیکترین تا دورترین گره، ملاقات می شوند. البته منظور از فاصله، تعداد یالهای موجود درمسیر s به گره می باشد.

جستجو به ترتیب سطح اول: BFS

- اگر سطح k را بعنوان گره هایی در نظر بگیریم که فاصله شان تا گره s، برابر k یال است، آنگاه BFS، ابتدا تمام گره های سطح ۱ را ملاقات می کند. سپس تمام گره های سطح ۲ را ملاقات می کند و ...
- نحوه انجام کار:
  - با یک گره، کار را شروع کن. این گره را ملاقات کن. سپس تمام گره های مجاور آن گره، که تا کنون ملاقات نشده اند، را ابتدا ملاقات و سپس در یک صف قرار بده.
  - تا زمانی که صف، تهی نشده است، عنصر سر صف را خارج کن و بعنوان گره شروع، همین روال را با آن اجرا کن.

جستجو به ترتیب سطح اول: BFS

- مثال: گره شروع: گره ۱



جستجو به ترتیب سطح اول: BFS

```
void BFS(vertex source) {
    visited[source] = true;
    for ( all vertices w adjacent to source) {
        if ( ! visited[w] ) {
            visited[w] = true;
            Enqueue(w);
        }
    }
    newSource = Dequeue();
    if(newSource!=null) BFS(newSource);
}
```

نسخه بازگشتی

جستجو به ترتیب سطح اول: BFS

```
void BFS(vertex source) {
   while(source!=null) {
       visited[source] = true;
       for ( all vertices w adjacent to source) {
         if ( ! visited[w] ) {
                  visited[w] = true;
                  Enqueue(w);
         }
       }
       source = Dequeue();
   }
}
```

نسخه غیر بازگشتی

جستجو به ترتیب سطح اول: BFS

اگر گراف متصل نبود چطور؟

```
void generalBFS(vertex source) {
    for(int i=0; i<|V| ; i++ ) {
        if( ! visited( source_i ) ) {
            BFS(source_i);
        }
    }
}
```

جستجو به ترتیب سطح اول: BFS

- مرتبه زمانی
  - هر گره دقیقا یک بار در صف درج می شود. درج در صف با $O(1)$ انجام می شود.
  - گراف دارای $|V|$ گره است پس درج تمام گره ها با $O(|V|)$ انجام می شود.
  - هر گره دقیقا یک بار از صف خارج می شود. خارج کردن یک گره از صف با $O(1)$ انجام می شود.
  - گراف دارای $|V|$ گره است. در نتیجه خارج کردن تمام گره ها از صف، با $O(|V|)$ انجام می شود.

جستجو به ترتیب سطح اول: BFS

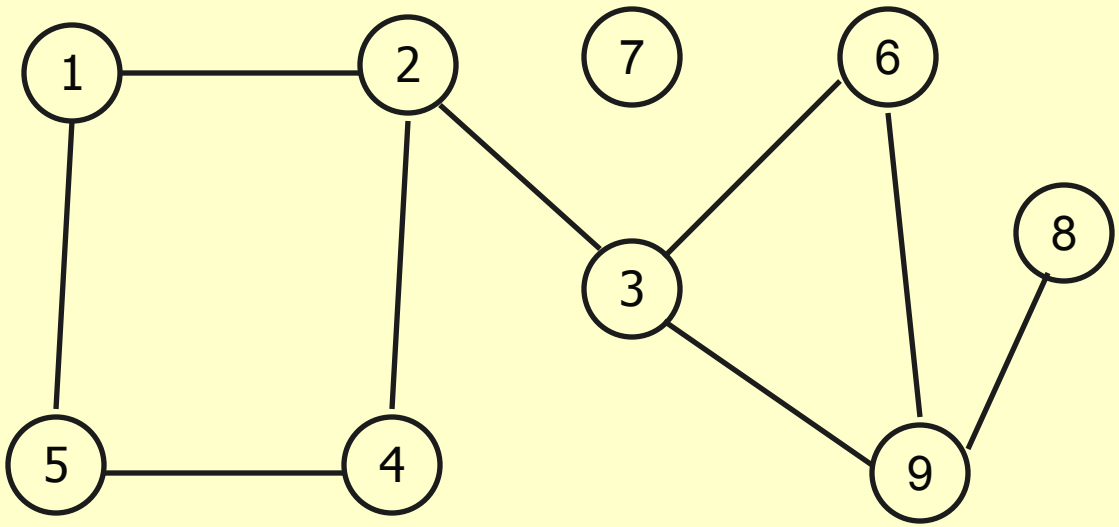
- ادامه مرتبه زمانی
  - علاوه بر ورود هر گره به صف و خروج هر گره از صف، کاری که صورت می گیرد، تشخیص گره های مجاور هر یک از گره ها می باشد. یعنی برای هر یک از گره های گراف باید گره های مجاورش را پیدا کنیم.
  - اگر از لیست مجاورت برای نمایش گراف استفاده شده باشد، این کار با $O(|E|)$ قابل انجام است.
  - اگر از ماتریس مجاورت برای نمایش گراف استفاده شده باشد، این کار با $O(|V|^2)$ قابل انجام است.
- در نتیجه
- اگر از لیست مجاورت استفاده شده باشد، مرتبه زمانی BFS، برابر $O(|V| + |E|)$ می باشد.
- اگر از ماتریس مجاورت استفاده شده باشد، مرتبه زمانی BFS، برابر $O(|V|2)$ می باشد.

جستجو به ترتیب عمق اول: DFS

- در جستجو به ترتیب عمق،
  - از یک گره بعنوان مبدا جستجو، کار را شروع می کنیم و سپس در هر مرحله، از بین همسایگان ملاقات نشده گره جاری، یک گره را انتخاب کرده، عمل جستجو را از آن گره ادامه می دهیم. اگر در یک مرحله، گره جاری، هیچ گره مجاور ملاقات نشده ای نداشت، به گره قبل برمیگردیم و گره بعدی را از بین گره های مجاور ملاقات نشده آن گره، انتخاب می کنیم (backtracking)
- ملاقات گره ها در امتداد یک شاخه تا انتهای آن ادامه می یابد و با پایان یافتن شاخه، شاخه های مجاور بررسی می شوند.

جستجو به ترتیب عمق اول: DFS

گراف بدون جهت G=(V,E)



یک جواب حاصل از جستجوی عمقی گراف G: نقطه شروع: گره 1

1 – 2 – 3 – 6 – 9 – 8 – 4 – 5 – 7

البته این جواب یکتا نمی باشد. یک جواب دیگر:

1 – 5 – 4 – 2 – 3 – 9 – 5 – 8 - 7

جستجو به ترتیب عمق اول: DFS

```
void DFS(vertex source) {
    visited[source] = true;
    u = source;
    for ( all vertices w adjacent to u) {
        if (! visited[w] ) {
            DFS(w);
        }
    }
}
```

جستجو به ترتیب عمق اول: DFS

- مرتبه زمانی
  - هر گره دقیقا یک بار ملاقات می شود.
    - ملاقات شدن یک گره را می توان بمنزله چاپ مقدار آن در خروجی، در نظر گرفت.
    - ملاقات شده هر گره با $O(1)$ انجام می شود.
  - ملاقات تمام گره ها با $O(|V|)$ قابل انجام است.
  - علاوه بر ملاقات هر گره، کاری که صورت می گیرد، آنستکه برای هر گره، باید گره های مجاورش تشخیص داده شود.
    - اگر از لیست مجاورت استفاده شده باشد، این کار با $O(|E|)$ قابل انجام است.
    - اگر از ماتریس مجاورت استفاده شده باشد، این کار با $O(|V|^2)$ قابل انجام است.

جستجو به ترتیب عمق اول: DFS

- در نتیجه، مرتبه زمانی DFS
  - اگر از لیست مجاورت استفاده شده باشد: $O(|V| + |E|)$
  - اگر از ماتریس مجاورت استفاده شده باشد: $O(|V|^2)$

جستجو به ترتیب عمق اول: DFS

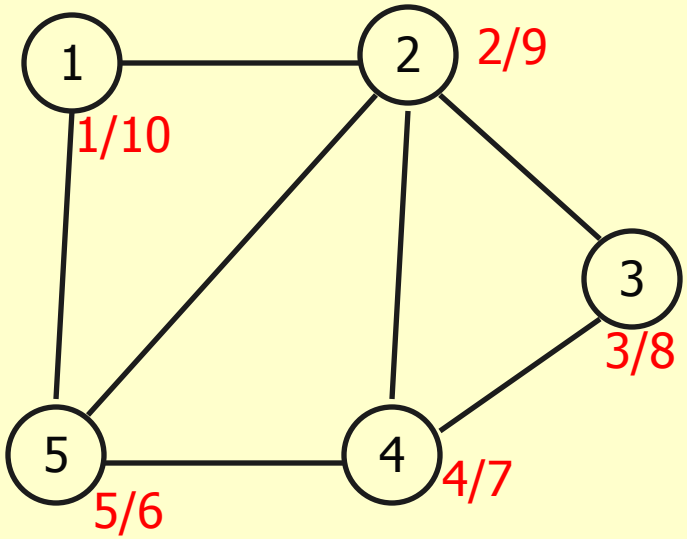
- دو مفهومی که در ادامه به آنها نیاز داریم
  - زمان اکتشاف: discovery time
    - زمانی که یک گره برای اولین بار ملاقات می شود.
  - زمان پایان: finish time
    - زمانی که بررسی و بسط گره های مجاور (مستقیم یا باواسطه) یک گره، تمام می شود.
- بسادگی می توان الگوریتم DFS را طوری تغییر داد که زمان پایان و زمان اکتشاف گره ها را هم محاسبه نماید.

جستجو به ترتیب عمق اول: DFS

```
void DFS(vertex source) {
    visited[source] = true;
    discTime[source] = ++globalCounter;
    u = source;
    for ( all vertices w adjacent to u) {
        if (! visited[w] ) {
            DFS(w);
        }
    }
    finishTime[source]  = ++globalCount;
}
```

تغییراتی که در الگوریتم DFS داده شد تا زمان اکتشاف و زمان پایان گره ها را هم محاسبه نماید.

جستجو به ترتیب عمق اول: DFS



گراف بدون جهت G=(V,E)

نتیجه جستجوی عمقی گراف روبرو (مبدا جستجو گره **1** می باشد)

1-2-3-4-5

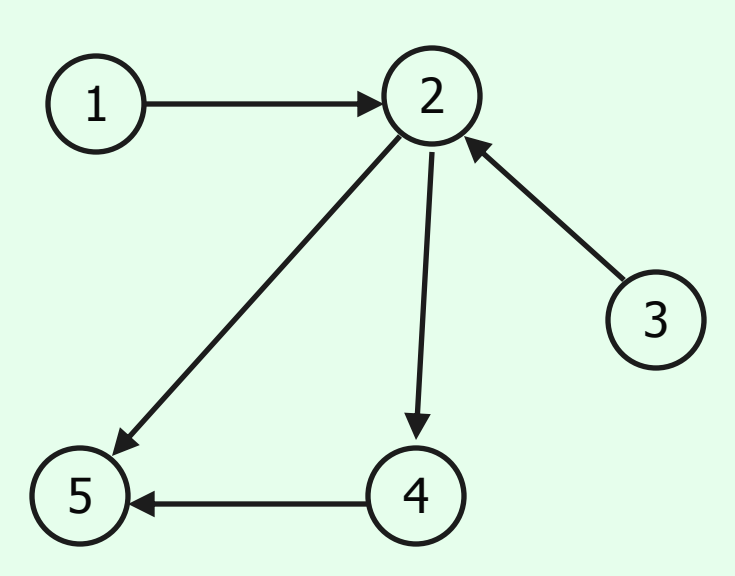
جستجو به ترتیب عمق اول: DFS

- دو نمونه از کاربردهای DFS
  - مرتب سازی توپولوژیکی گراف
  - یافتن مولفه های قویا همبند یک گراف

مرتب سازی توپولوژیکی گراف

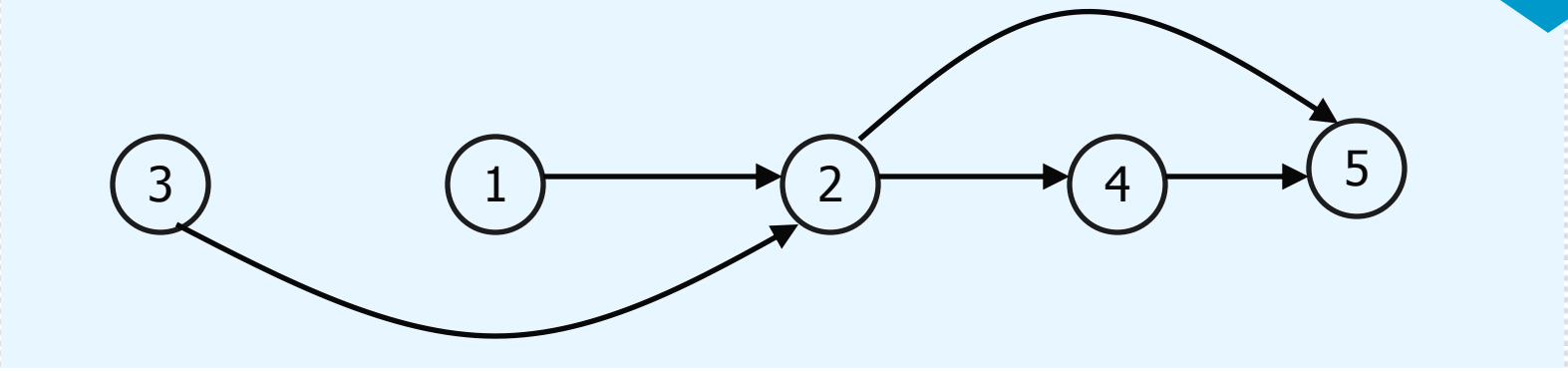
- گراف G=(V, E) را که یک DAG می باشد در نظر بگیرید.
  - DAG: Directed Acyclic Graph
  - گراف جهت دار بدون حلقه
- نتیجه مرتب سازی توپولوژیکی گراف G، یک ترتیب خطی از تمام گره های گراف G است بطوریکه برای هر یال <u,v> در این گراف، در ترتیب مذکور، گره u قبل از گره v آمده باشد.
  - می توان اینطور تصور کرد که می خواهیم تمام گره های گراف را روی یک خط افق قرار دهیم طوری که اگر یالهای گراف را رسم کنیم، تمام یالها از سمت چپ به راست باشند.
- DAG ها در بسیاری از کاربردها برای نمایش ترتیب و اولویت رویدادها استفاده می شوند.

مرتب سازی توپولوژیکی گراف



گراف جهت دار بدون حلقه G

نتیجه مرتب سازی توپولوژیکی گراف **G**
(لزوما یکتا نمی باشد)



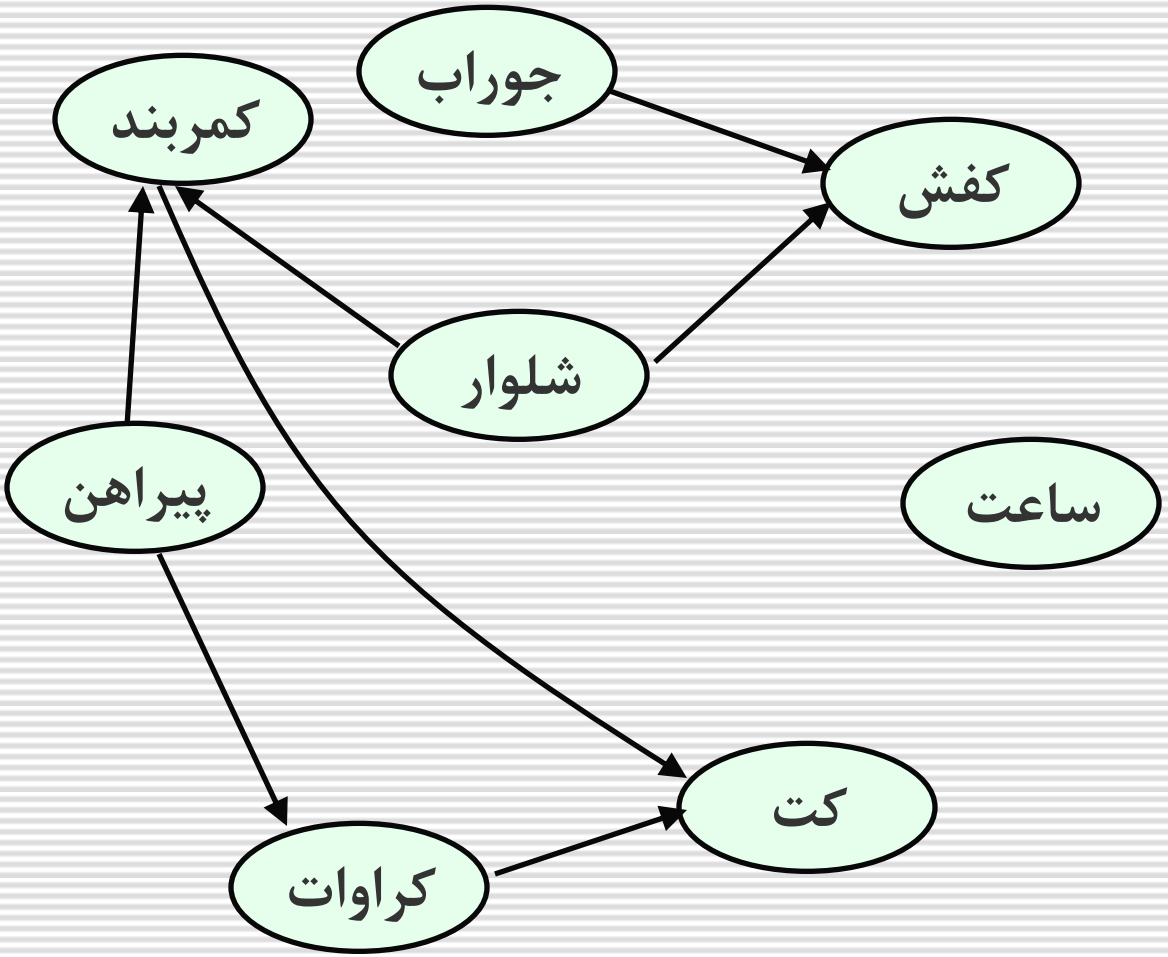
مرتب سازی توپولوژیکی گراف

- الگوریتم مرتب سازی توپولوژیکی
  - مرحله اول: با استفاده از یک DFS، گراف را پیمایش کن. در ضمن این کار، زمان پایان هر یک از گره ها را نیز محاسبه کن.
  - مرحله دوم: گره ها را به ترتیب نزولی زمان پایانشان، در انتهای یک لیست درج کن.
  - مرحله سوم: این لیست را بعنوان نتیجه مرتب سازی توپولوژیکی گراف بازگردان.

توجه: می توان مرحله دوم را در مرحله اول ادغام نمود.

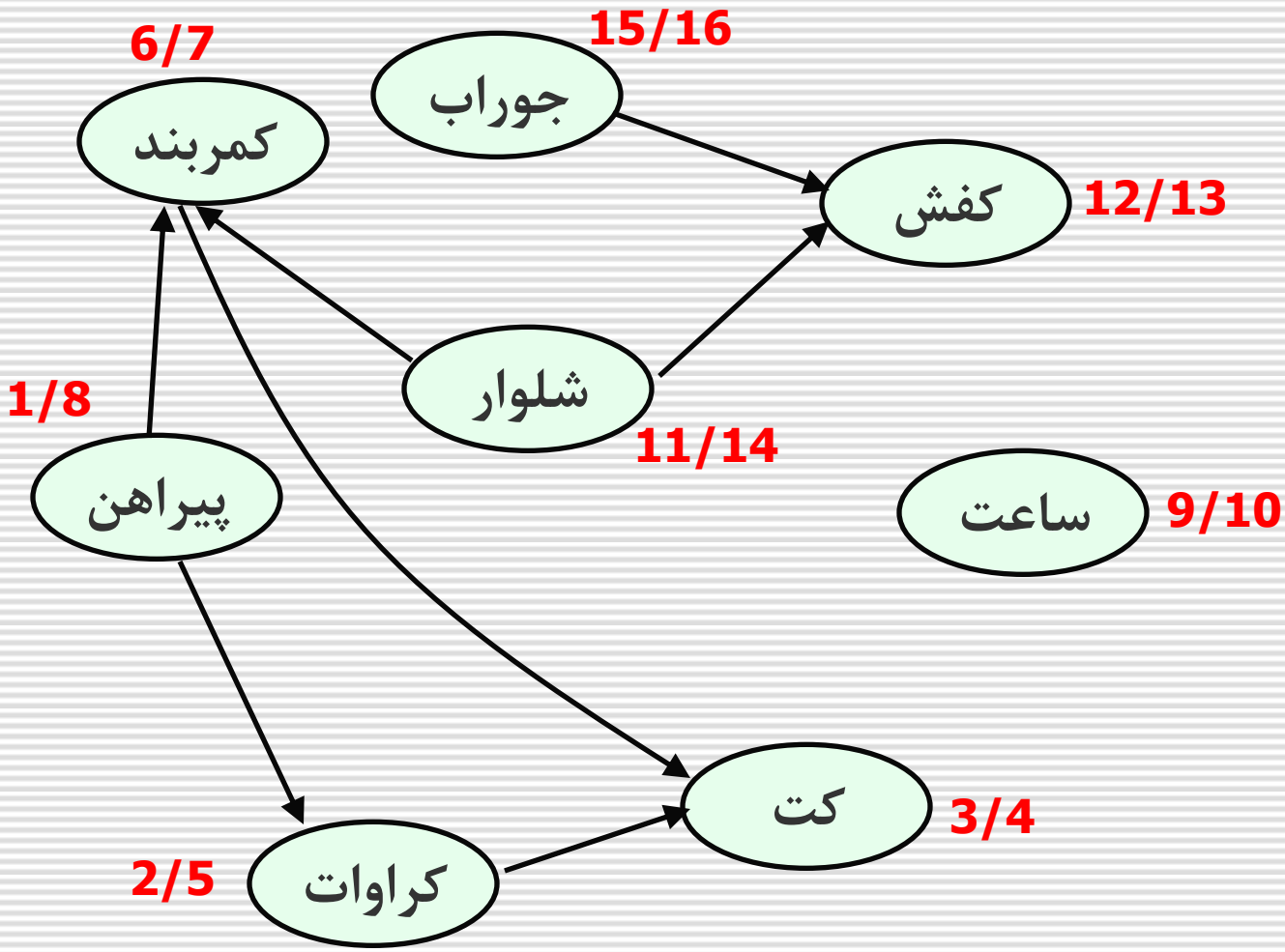
مرتب سازی توپولوژیکی گراف

- مثال:



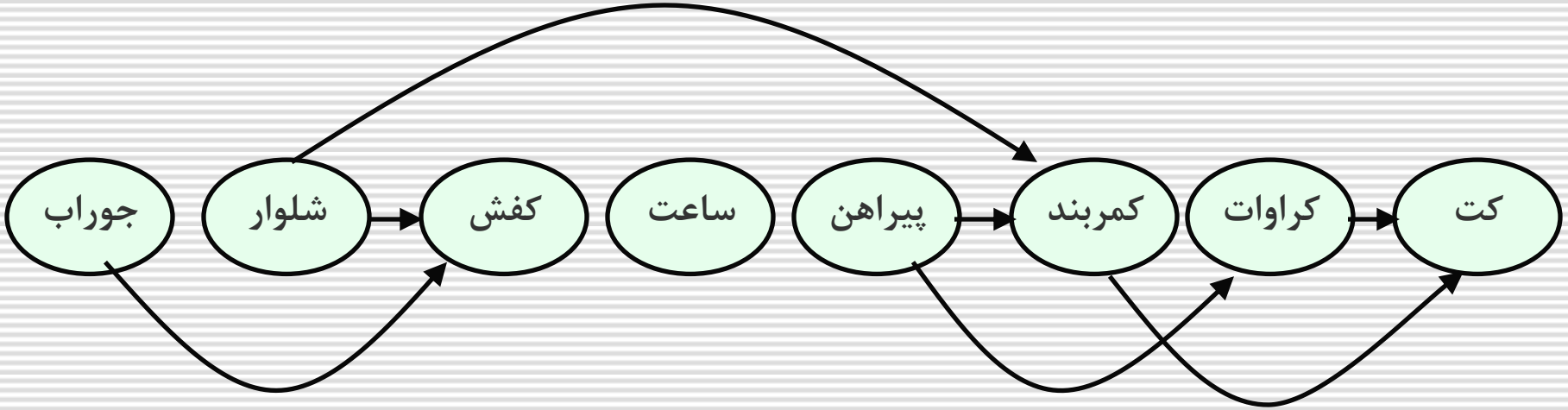
مرتب سازی توپولوژیکی گراف

- مثال:



مرتب سازی توپولوژیکی گراف

- مثال:



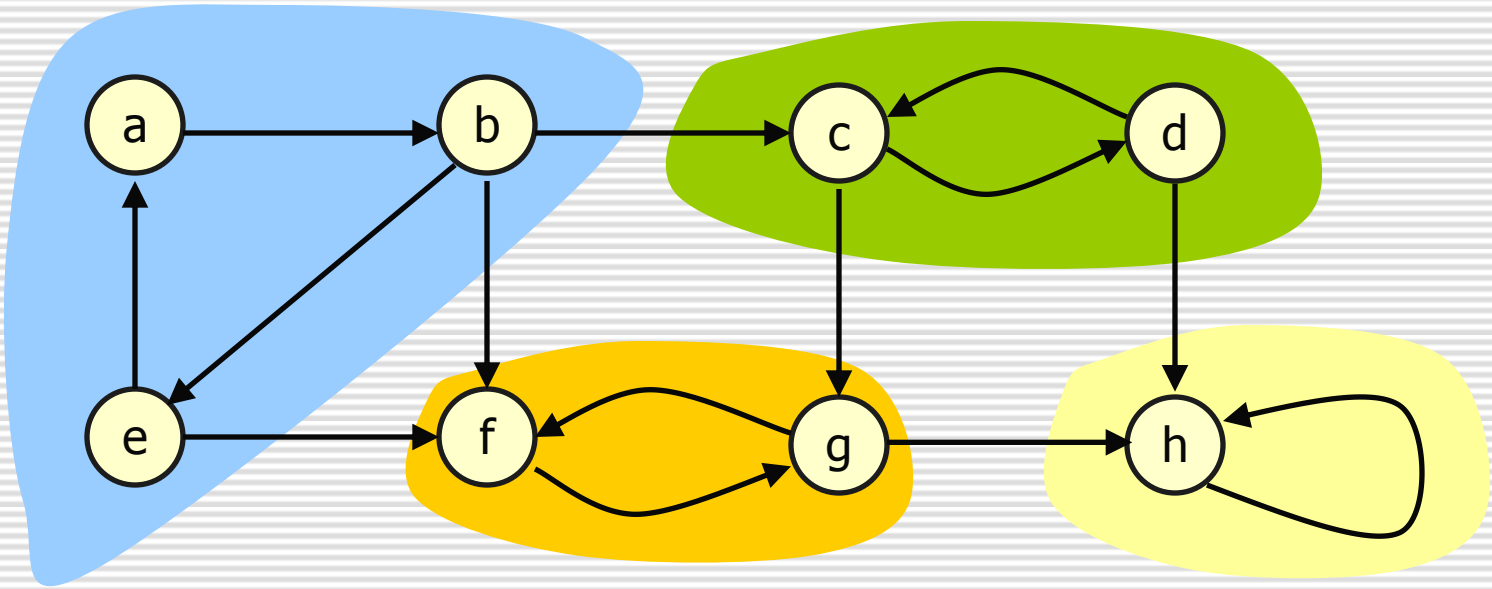
مرتب سازی توپولوژیکی گراف

- مرتبه زمانی ؟

یافتن مولفه های قویا همبند یک گراف

- گراف جهت دار G=(V,E)
- بزرگترین زیرمجموعه V به نام C که
  - برای هر زوج گره متمایز u و v از مجموعه C، یک مسیر از u به v و همچنین یک مسیر از v به u در گراف G باشد.
  - به C یک مولفه قویا همبند گراف G گفته می شود.
  - Strongly connected component

یافتن مولفه های قویا همبند یک گراف



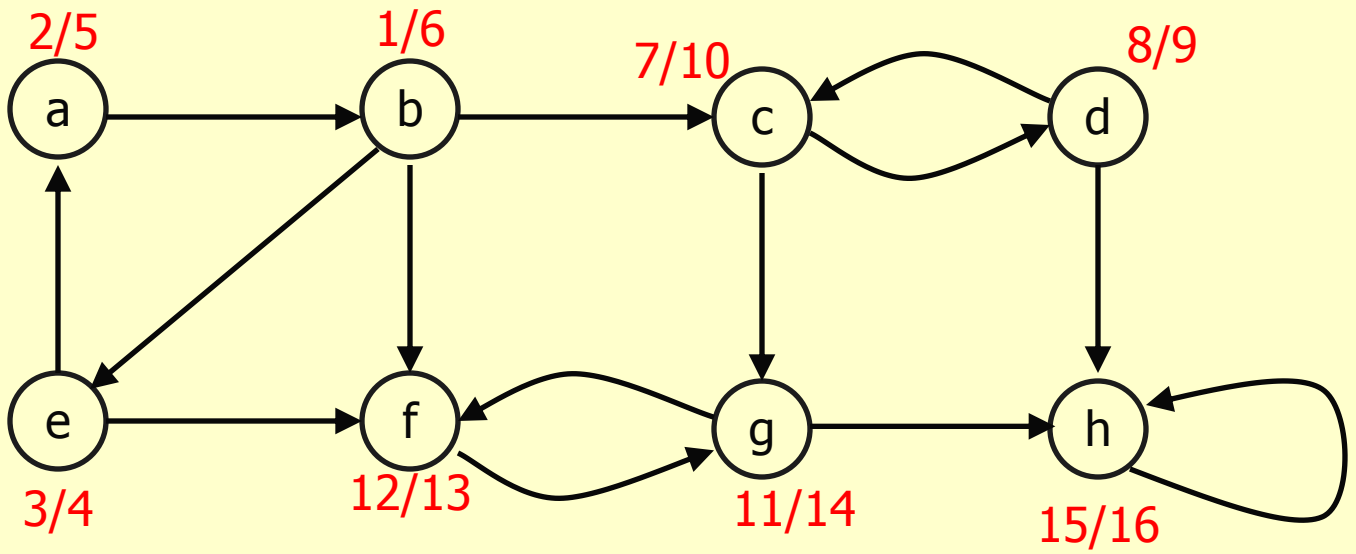
گراف فوق دارای ۴ مولفه قویا همبند می باشد.

می خواهیم الگوریتمی ارائه کنیم که برای گراف G، مولفه های قویا همبند آن را تشخیص دهد.

یافتن مولفه های قویا همبند یک گراف

- الگوریتم
- مرحله اول: گراف G را با DFS پیمایش کن و در ضمن این پیمایش، زمان پایان هر گره را نیز مشخص کن.
- مرحله دوم: از روی گراف G، گراف G^T را بدست آور.
- مرحله سوم: گراف G^T را با DFS پیمایش کن، اما در ضمن انجام این کار، هربار که لازم بود نقطه شروع جستجو را انتخاب کنی، گره ها را بر حسب ترتیب نزولی زمان پایان (که در مرحله اول محاسبه شد) انتخاب کن. هر بار که نیاز به تعیین نقطه مبدا بود، گره های ملاقات شده قبلی را بعنوان یک مولفه قویا همبند، به خروجی بفرست.

یافتن مولفه های قویا همبند یک گراف



مولفه های قویا همبند گراف فوق:

مولفه اول: **b, a, e** مولفه دوم: **c, d**

مولفه سوم: **g, f** مولفه چهارم: **h**

یافتن مولفه های قویا همبند یک گراف

- مرتبه زمانی؟