طراحی و تحلیل الگوریتم ها

مدرس: سعدون عزیزی

s.azizi@uok.ac.ir

گروه مهندسی کامپیوتر

نیم سال دوم ۹۷-۹۶

الگوریتم‌های گراف

- **الگوریتم‌های اولیه گراف**
- درخت‌های پوشای کمینه
- کوتاهترین مسیرهای با مبدأ یکسان
- کوتاهترین مسیرها بین هر دو رأس

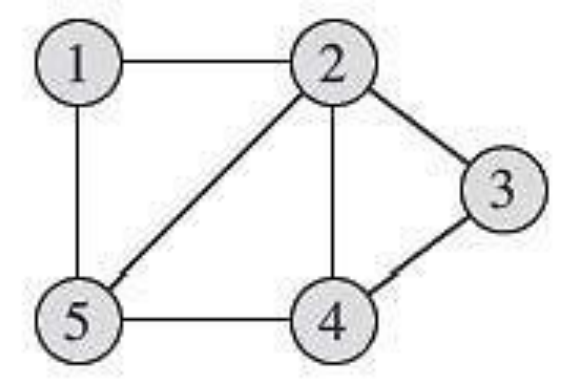
کاربردهای گراف

- شبکه‌های کامپیوتری
- شبکه‌های اجتماعی
- صفحات وب
- طراحی مدارهای الکتریکی
- زمانبندی کارها
- نقشه‌ها
- ...

تعریف گراف

تعریف

گراف G به صورت یک زوج مرتب G=(V,E) قابل تعریف است که در آن V مجموعه‌ای متناهی از رأس‌ها (گره‌ها) و E مجموعه‌های متناهی از یال‌ها (لینک‌ها) است که هر یال دو رأس دلخواه را به هم وصل می‌کند



مثال:

- مجموعه رئوس: V={1,2,3,4,5}
- مجموعه یال‌ها: E={(1,2),(1,5),(2,3),(2,4),(2,5)}

نمایش گراف‌ها

- ماتریس مجاورت

  - این نمایش از گراف G=(V,E)، به صورت یک ماتریس $A=(a_{ij})$ با اندازه |V|×|V| است که در آن

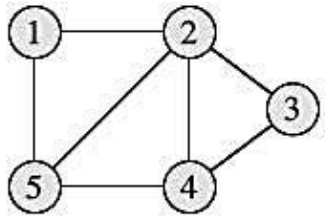
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E\ , \\ 0 & \text{otherwise}\ . \end{cases}$$

- لیست مجاورت

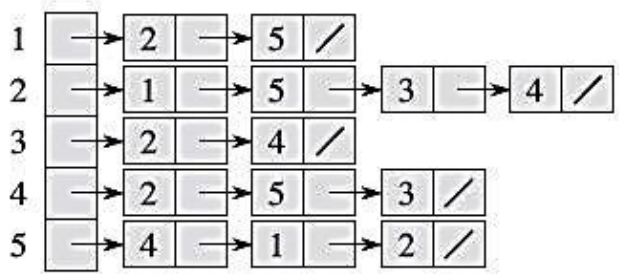
  - این نمایش از گراف G=(V,E)، شامل یک آرایه (که آن را Adj می‌نامیم) از لیست‌های پیوندی با اندازه V است؛ یک لیست برای هر رأس. برای هر $u \in V$، لیست Adj[u] شامل تمام رأس‌هایی مانند v است که $(u,v) \in E$.

نمایش گراف‌ها (مثال)

- گراف بدون جهت



(a)

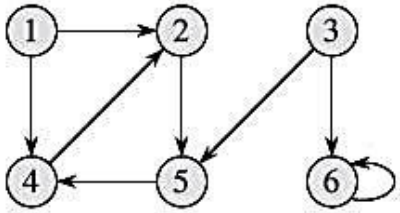


(b)

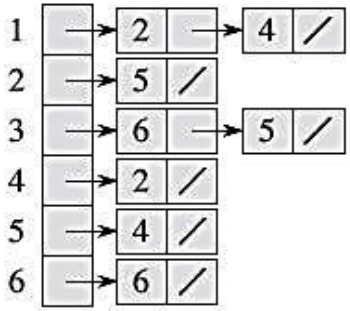
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

- گراف جهت‌دار



(a)



(b)

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

(c)

نمایش گراف‌ها (مقایسه)

نوع نمایش	وجود یال بین دو رأس	لیست یال‌ها	لیست همسایه‌ها
ماتریس مجاورت	O(1)	$O(\|V\|^2)$	O(|V|)
لیست مجاورت	O(deg(v))	O(|E|)	O(deg(v))

جستجوی سطح اول

- جستجوی سطح اول (Breadth-First Search - BFS)

- الگوریتم از یک رأس مشخص مانند s در گراف G=(V,E) شروع کرده و رأس‌ها را پیمایش می‌کند تا تمام رأس‌های قابل دسترس از s را «کشف» کند.
- این الگوریتم فاصله (کمترین تعداد رأس میانی) هر رأس از s را مشخص می‌کند
- همچنین یک «درخت سطح اول» با ریشه‌ی s تشکیل می‌دهد که شامل تمام رأس‌های قابل دسترس از s می‌باشد
- این الگوریتم برای هر دو نوع گراف جهت‌دار و بدون جهت قابل استفاده است
- این الگوریتم قبل از کشف هر رأس با فاصله k+1 از s، تمام رأس‌های با فاصله k از s را کشف می‌کند (علت نامگذاری الگوریتم)

جستجوی سطح اول

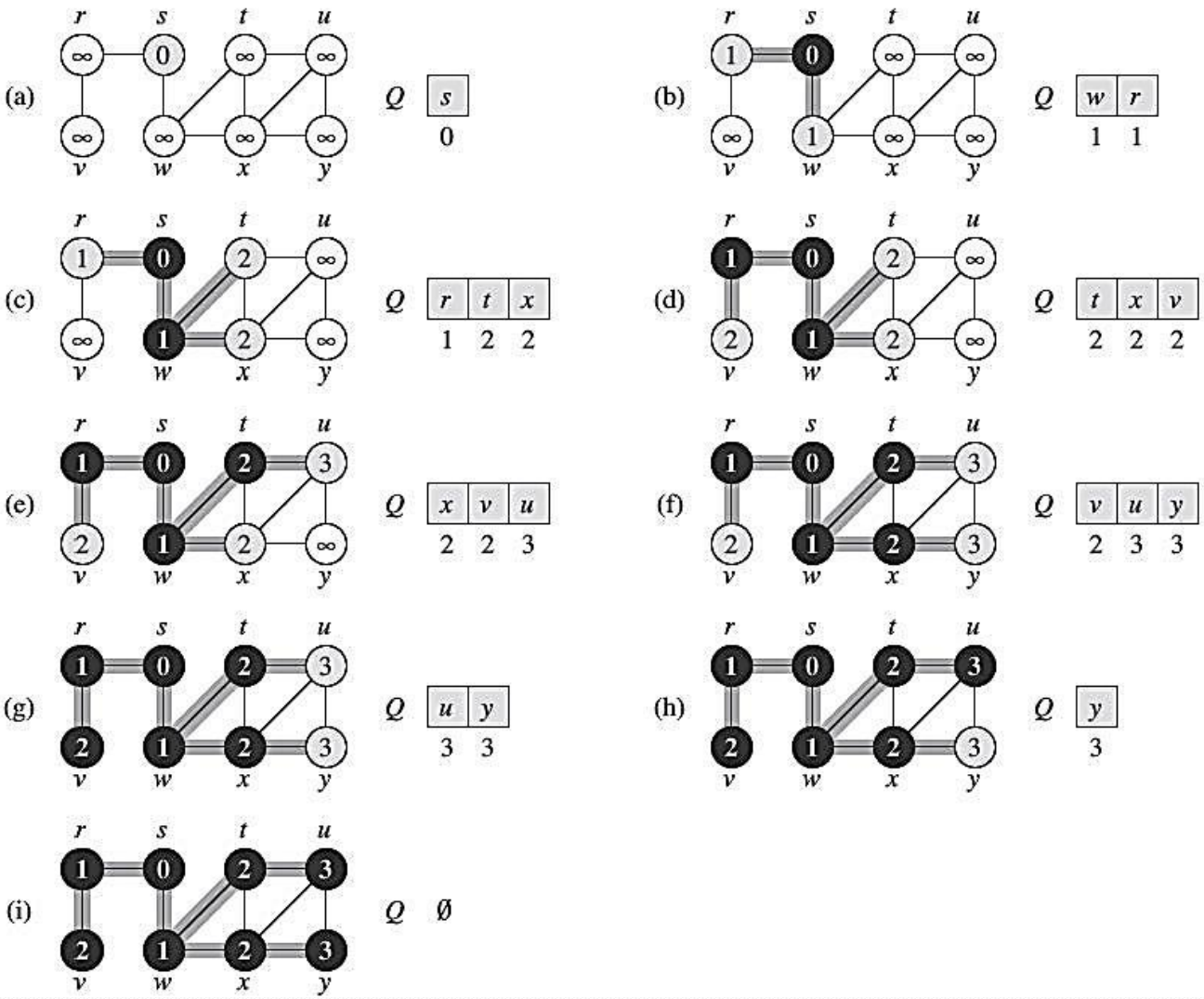
برای هر راس $v \in V$، الگوریتم اطلاعات زیر را در مورد آن نگه می‌دارد:

- v.color:
  - سفید: اگر راس v هنوز کشف نشده باشد
  - خاکستری: اگر راس v کشف شده باشد اما همسایه‌هایش هنوز کشف نشده باشند
  - سیاه: اگر راس v به همراه تمام همسایه‌هایش کشف شده باشند
- v.d: تعداد یال‌ها از راس شروع s تا راس v (فاصله ریشه تا راس v)
- $v.\pi$: پدر راس v در درخت جستجو

جستجوی سطح اول: الگوریتم

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

جستجوی سطح اول: مثال



جستجوی سطح اول: زمان اجرا

- زمان اجرای الگوریتم BFS از مرتبه $O(V+E)$ است. (چرا؟)

جستجوی عمق اول

- استراتژی جستجوی عمق اول این است که تا حد ممکن «عمیق‌تر» در گراف پیش برود
- در هر لحظه، راس‌های همسایه‌ی آخرین راس کشف شده مورد بررسی قرار می‌گیرند. اگر آن راس همسایه‌ای داشت که هنوز کشف نشده بود، همین کار در مورد همان راس کشف نشده انجام می‌شود
- وقتی تمام همسایه‌های راس v کشف شدند، الگوریتم به راسی باز می‌گردد که v از طریق آن کشف شده بود
- این عملیات آن قدر تکرار می‌شود تا تمام راس‌های قابل دسترس از راس شروع کشف شوند

جستجوی عمق اول

برای هر راس $v \in V$، الگوریتم اطلاعات زیر را در مورد آن نگه می‌دارد:

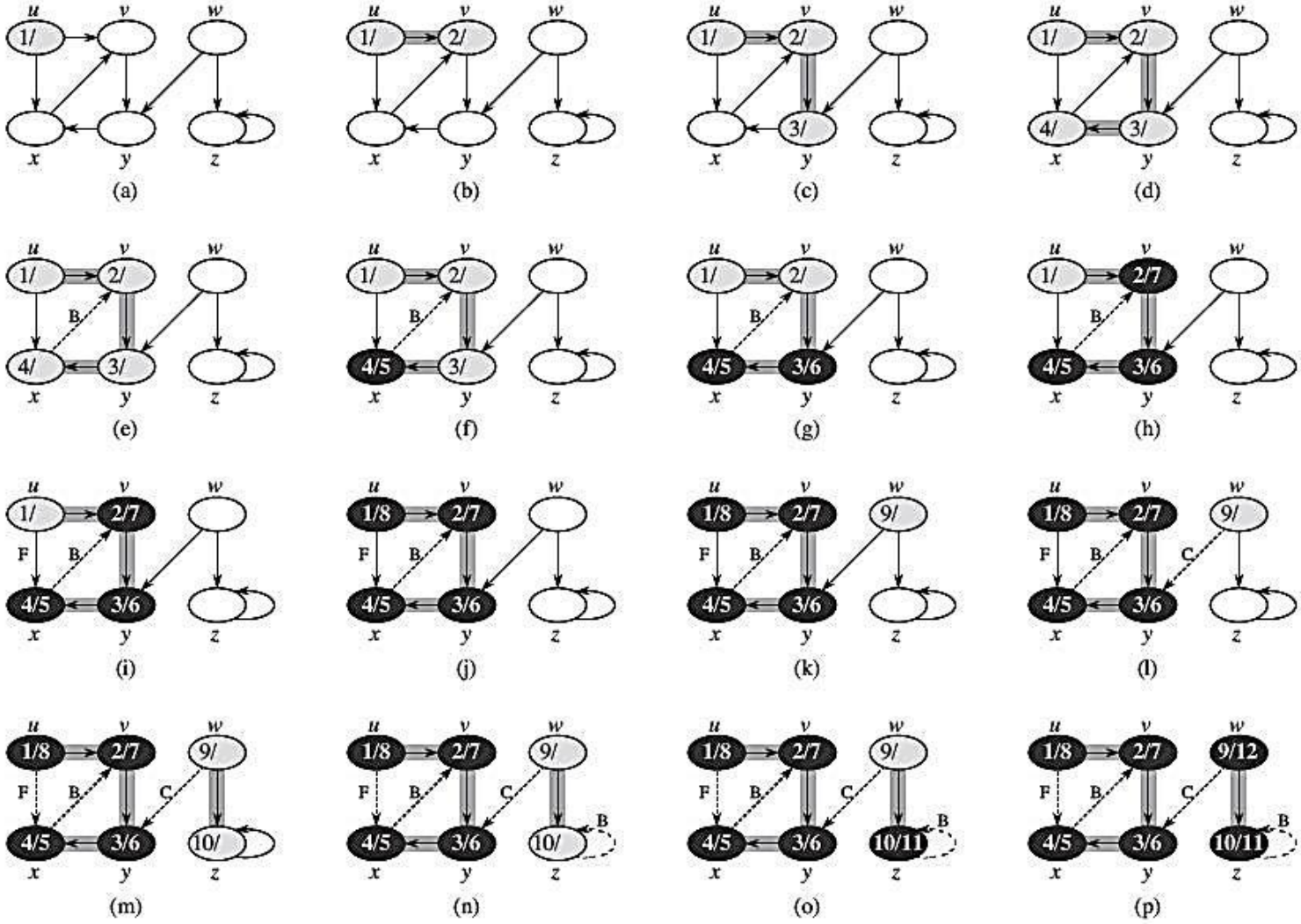
- v.color: همانند BFS
- برای هر راس v دو زمان نگه‌داری می‌شود:
  - v.d: زمان کشف (خاکستری شدن) راس v
  - v.f: زمان کامل شدن (سیاه شده) راس v
- $v.\pi$: پدر راس v در درخت جستجو

جستجوی عمق اول: الگوریتم

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1                  // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]            // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK                  // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

جستجوی عمق اول: مثال



جستجوی عمق اول: زمان اجرا

- زمان اجرای الگوریتم DFS از مرتبه $O(V+E)$ است. (چرا؟)

جستجوی عمق اول: دسته‌بندی یال‌ها

- یال‌های درختی (Tree edge): یال‌های درون جنگل عمق اول هستند. یال (u,v) یک یال درختی است اگر v ابتدا با بررسی یال (u,v) کشف شود.
- یال‌های عقبی (Back edge): یال‌هایی مانند (u,v) هستند که راس u را به راس جدش v در درخت اول عمق وصل می‌کند.
- یال‌های جلویی (Forward edge): یال‌های غیردرختی مانند (u,v) هستند که راس u را به راس نوه‌اش v وصل می‌کند.
- یال‌های ضربدری (Cross edge): تمام یال‌های دیگر را می‌گویند.

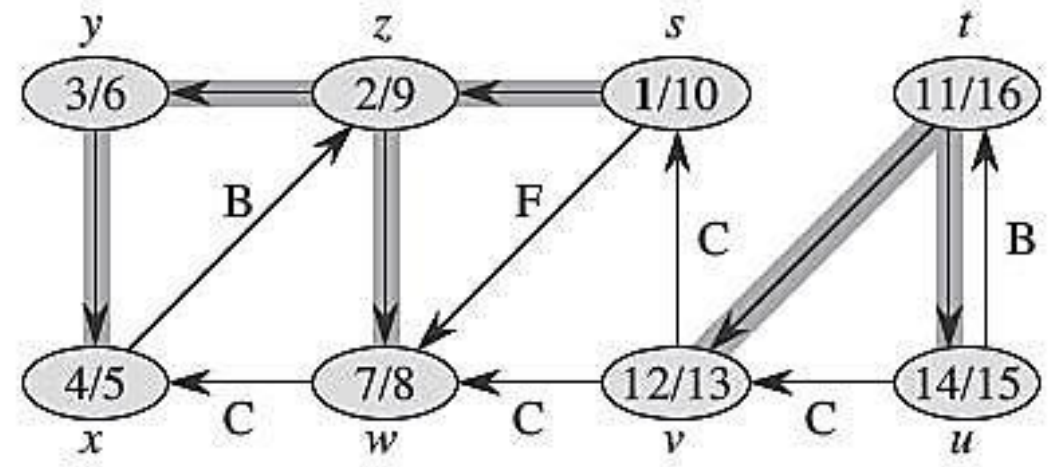
جستجوی عمق اول: دسته‌بندی یال‌ها

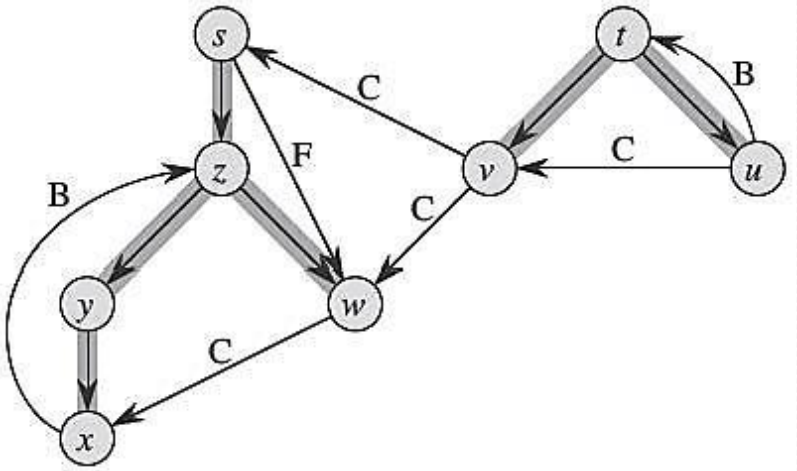
در الگوریتم DFS وقتی برای اولین بار یال (u,v) را بررسی می‌کنیم، رنگ راس v اطلاعاتی در مورد آن یال به ما می‌دهد:

- رنگ سفید نشان‌دهنده یک یال درختی است
- رنگ خاکستری نشان‌دهنده یک یال عقبی است
- رنگ سیاه نشان‌دهنده یک یال جلویی یا ضربدری است

- اگر u.d<v.d آنگاه (u,v) یک یال جلویی است و اگر u.d>v.d آنگاه (u,v) یک یال ضربدری است (چرا؟)

جستجوی عمق اول: دسته‌بندی یال‌ها (مثال)





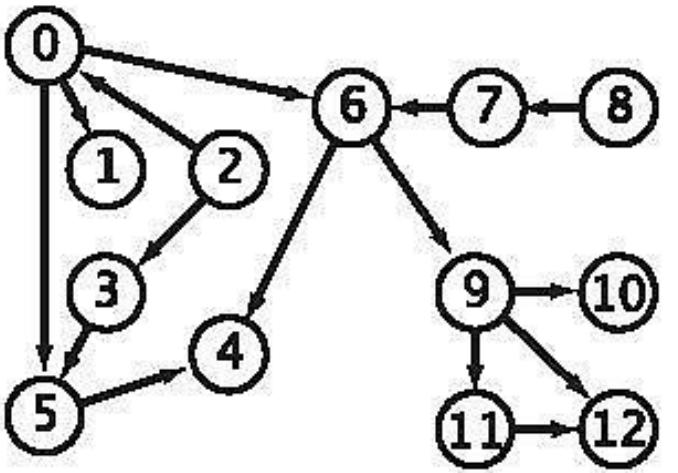
مرتب‌سازی توپولوژیکی

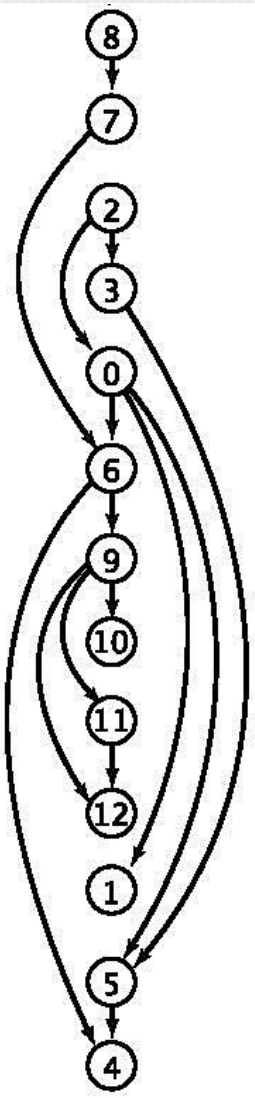
تعریف

یک مرتب‌سازی توپولوژیکی (Topological Sort) روی گراف G=(V,E) ترتیبی خطی است برای تمام رأس‌های آن به طوری که اگر G شامل یک یال (u,v) باشد، آنگاه در این ترتیب u قبل از v ظاهر می‌شود.

- اگر گراف دور داشته باشد، آنگاه چنین ترتیب خطی ممکن نیست.
- می‌توان از جستجوی عمق اول برای انجام مرتب‌سازی توپولوژیکی روی یک گراف جهت‌دار بدون دور (DAG) استفاده کرد.

مرتب‌سازی توپولوژیکی: مثال





مرتب‌سازی توپولوژیکی: الگوریتم

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

- مرتب‌سازی توپولوژیکی را می‌توان در زمان O(V+E) انجام داد.

مرتب‌سازی توپولوژیکی: اثبات درستی

لم: یک گراف جهت‌دار G بدون دور است اگر و فقط اگر جستجوی عمق اول G هیچ یال عقبی تولید نکند.

قضیه: TOPOLOGICAL-SORT(G) یک مرتب‌سازی توپولوژیکی برای گراف جهت‌دار بدون دور G تولید می‌کند.

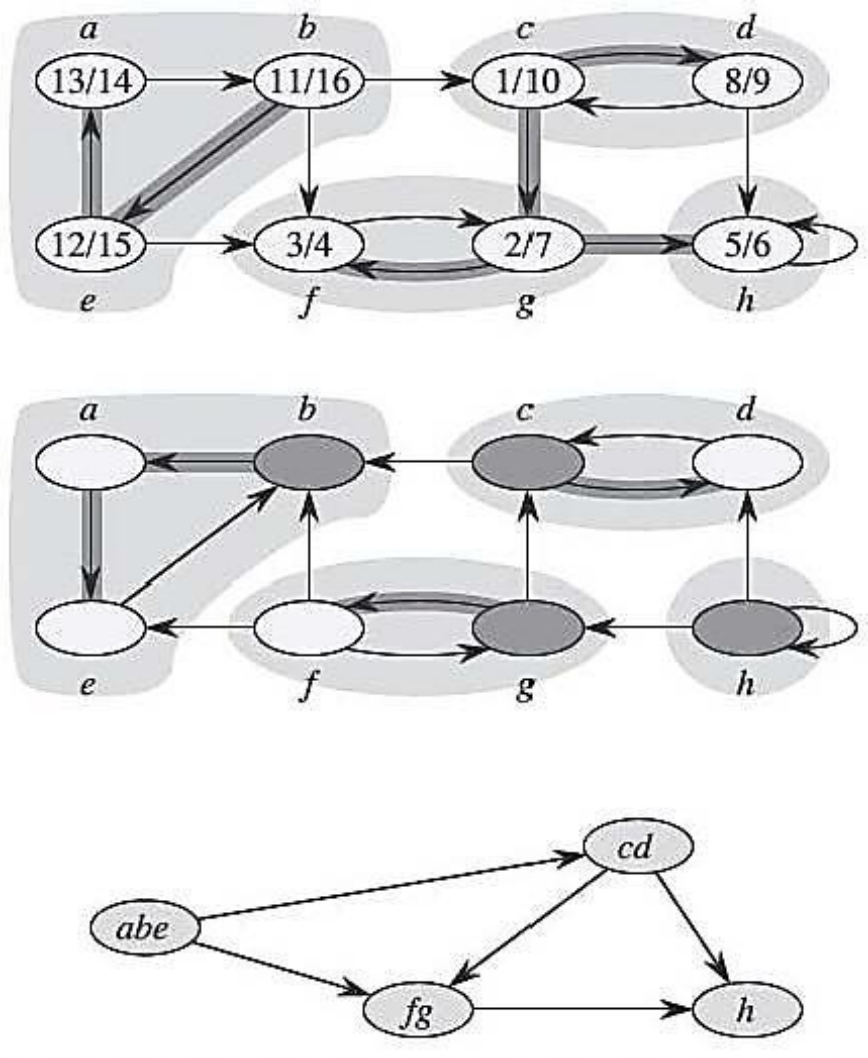
مؤلفه‌های قویاً همبند

تعریف

- یک مؤلفه‌ی قویاً همبند یک گراف $G=(V,E)$، یک مجموعه‌ی ماکسیمال مانند $C \subseteq V$ از رأس‌ها است به طوری که برای هر زوج u و v در C داشته باشیم $u \rightsquigarrow v$ و $v \rightsquigarrow u$؛ یعنی رأس‌های u و v از یکدیگر قابل دسترس باشند.
- ترانهاده گراف $G=(V,E)$ عبارت است از گراف $G^T=(V,E^T)$ جایی که $E^T=\{(u,v): (v,u) \in E\}$

- با داشتن نمایش لیست مجاورت گراف G، گراف G^T را می‌توان در زمان $O(V+E)$ ساخت.
- مؤلفه‌های قویاً همبند گراف G و G^T دقیقاً یکسان هستند: u و v در G از یکدیگر قابل دسترس هستند اگر و فقط اگر در G^T هم از یکدیگر قابل دسترس باشند.

مؤلفه‌های قویاً همبند: مثال



مؤلفه‌های قویاً همبند: الگوریتم

الگوریتم خطی زیر، با زمان اجرای $O(V+E)$، با استفاده از دو جستجوی عمق اول، یکی بر روی G و دیگری بر روی G^T، مؤلفه‌های قویاً همبند را در گراف جهت‌دار $G=(V,E)$ محاسبه می‌کند.

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^T), but in the main loop of DFS, consider the vertices
in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
separate strongly connected component