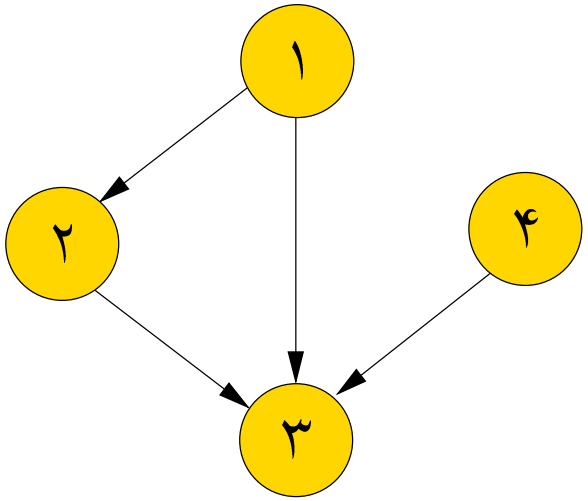
نمایش گراف‌ها (Graph Representation)

$G = (V, E)$ **که** V **مجموعه‌ی رئوس و** $E \subseteq V \times V$ **مجموعه‌ی یال‌هاست.**

اگر G **هم‌بند باشد،** $|E| \geq |V| - ۱$ **پس** $\lg E = \Theta(\lg |E|)$



نمایش گراف‌ها (ادامه)

گراف جهت‌دار (directed) (dirgaphs) **یا بدون جهت** (undirected)

پیاده‌سازی گراف‌ها

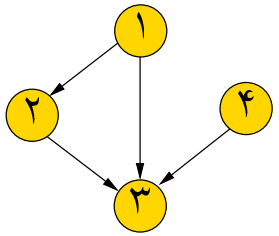
فرض: $V = \{۱, ۲, \ldots, n\}$

۱) ماتریس مجاورت (adjacency matrix)

$A[۱..n, ۱..n]$

$$A[i,j] = \begin{cases} ۱ & \text{if } (i,j) \in E \\ ۰ & \text{if } (i,j) \notin E \end{cases}$$

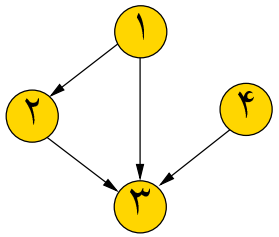
پیاده‌سازی گراف‌ها (ماتریس مجاورت)



i \ j	۱	۲	۳	۴
۱	0	1	1	0
۲	0	0	1	0
۳	0	0	0	0
۴	0	0	1	0

حافظه: $\Theta(V^2)$، **نمایش شلوغ** (dense representation)

نمایش گراف‌های خلوت



۲) لیست مجاورت (adjacency list)**: برای هر** $v \in V$ **رئوس مجاور** v **در** $adj[v]$

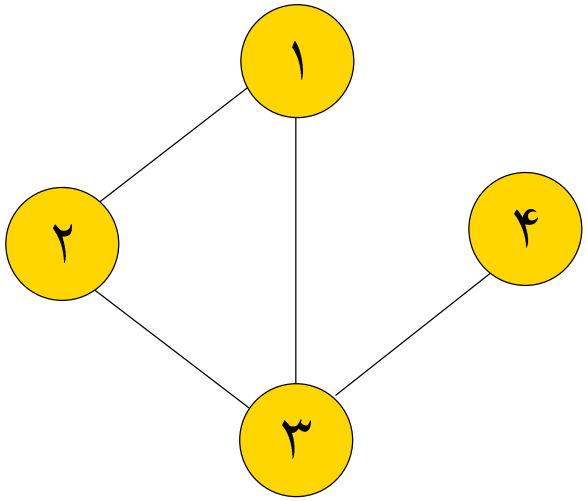
$$Adj[۱] = \{۲, ۳\}$$
$$Adj[۲] = \{۳\}$$
$$Adj[۳] = \{\}$$
$$Adj[۴] = \{۳\}$$

نمایش گراف‌های خلوت (ادامه)

(درجه‌ی خروجی) $|Adj[v]| = outdergee(v)$ **در گراف‌های جهت دار**

(درجه) $|Adj[v]| = dergee(v)$ **در گراف‌های عادی (بدون جهت دار)**

لم دست‌دادن (handshaking Lemma)



$$\sum deg(v) = ۲|E|$$

پس لیست مجاورت $\Theta(E+V)$ حافظه مصرف می‌کند.

جست‌وجوی گراف‌ها (Graph Searching Algorithms)

- بررسی همه‌ی رأس‌ها و یال‌های گراف $G = (V, E)$
- گراف می‌تواند جهت‌دار یا عادی باشد
- فرض می‌کنیم که گراف با لیست مجاورت پیاده‌سازی شده است

جست‌وجوی گراف‌ها

مثال‌ها:

- **جست‌وجوی عمق‌اول** (Depth-first Search (DFS))
- **جست‌وجوی سطح‌اول** (Breadth-first Search (DFS))

کاربردها: کامپالرها، گرافیک کامپیوتری، حل مازها، هوش مصنوعی

جست‌وجوی عمق‌اول (DFS)

ورودی: گراف G که ممکن است جهت‌دار یا عادی باشد

$time$: **یک متغیر سراسری به‌عنوان برچسب زمانی**

```
DFS (G)
  1  for each vertex u ∈ V[G]
  2      do color[u] ← white
  3  time ← 0
  4  for each vertex u ∈ V[G]
  5      do if color[u] = white
  6           then DFS-Visit (u)
```

جست‌وجوی عمق‌اول (ادامه)

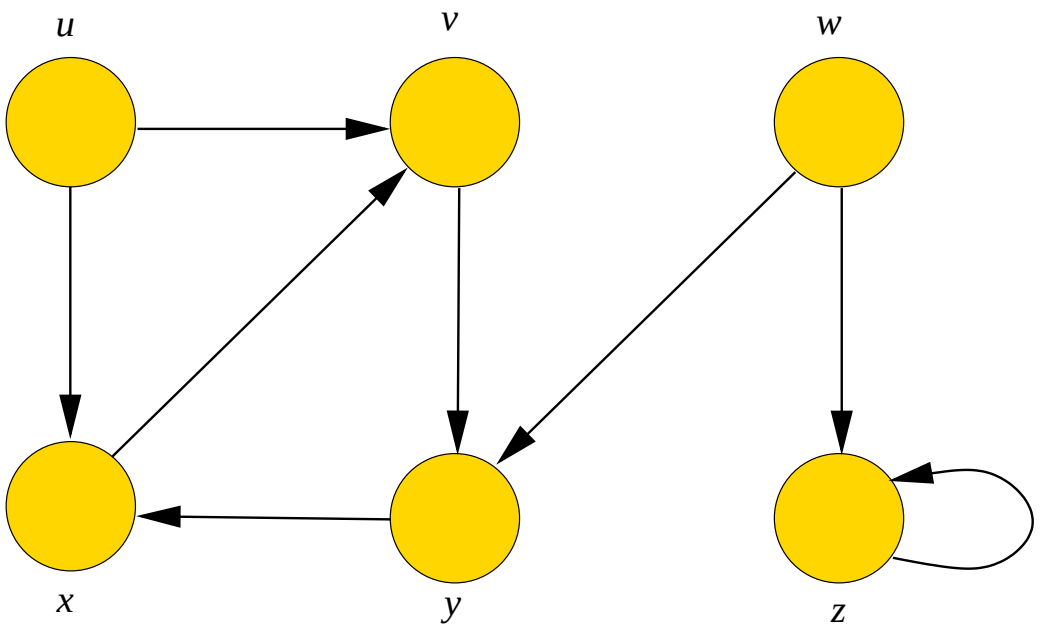
```
DFS-Visit (u)
 1  color[u] ← Gray          ▷ رأس سفید ملاقات شد
 2  d[u] ← time              ▷ زمان ملاقات را مشخص کن
 3  time ← time + 1          ▷ حرکت ساعت
 4  for each v ∈ Adj[u]
 5      do if color[v] = white
 6          then DFS-Visit (v)
 7  color[u] ← Black         ▷ اتمام ملاقات: رنگ سیاه
 8  f[u] ← time              ▷ زمان ختم ملاقات
 9  time ← time + 1          ▷ حرکت ساعت
```

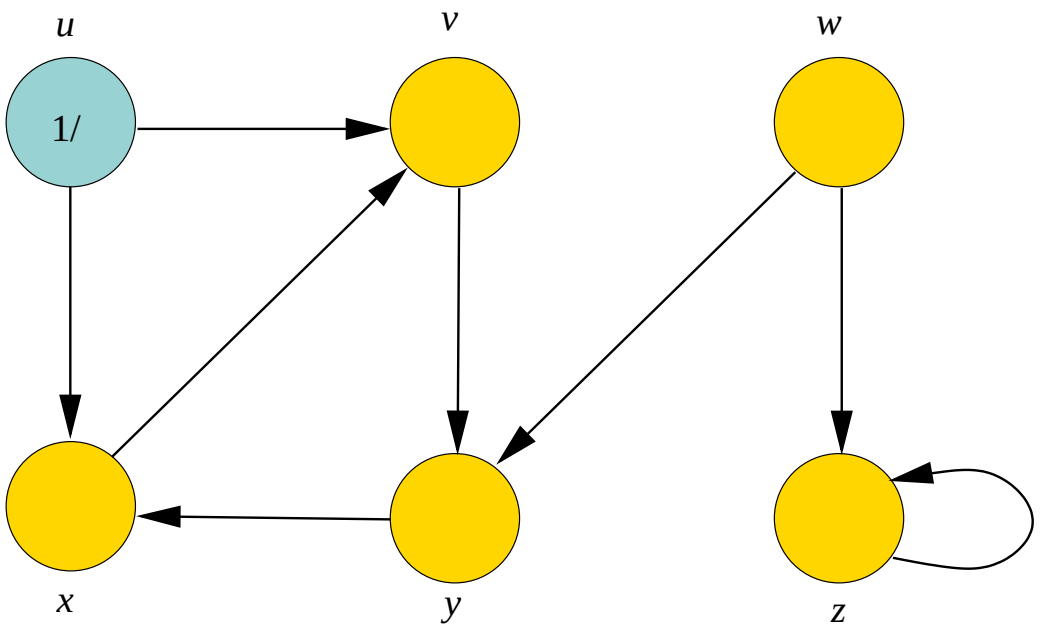
چرا درست کار می‌کند؟

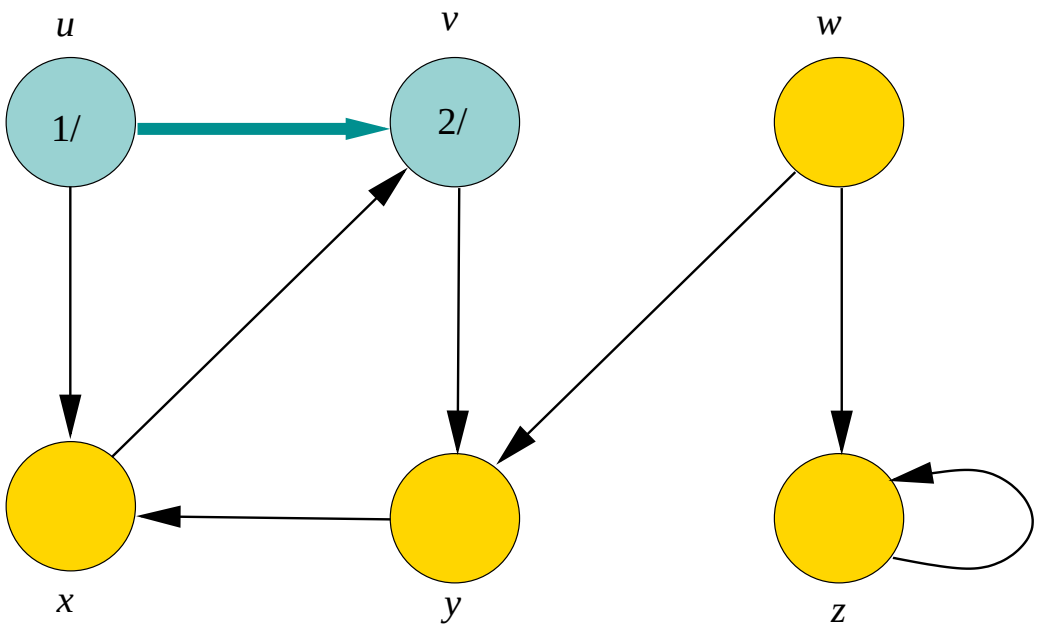
جست‌وجوی عمق‌اول: تحلیل

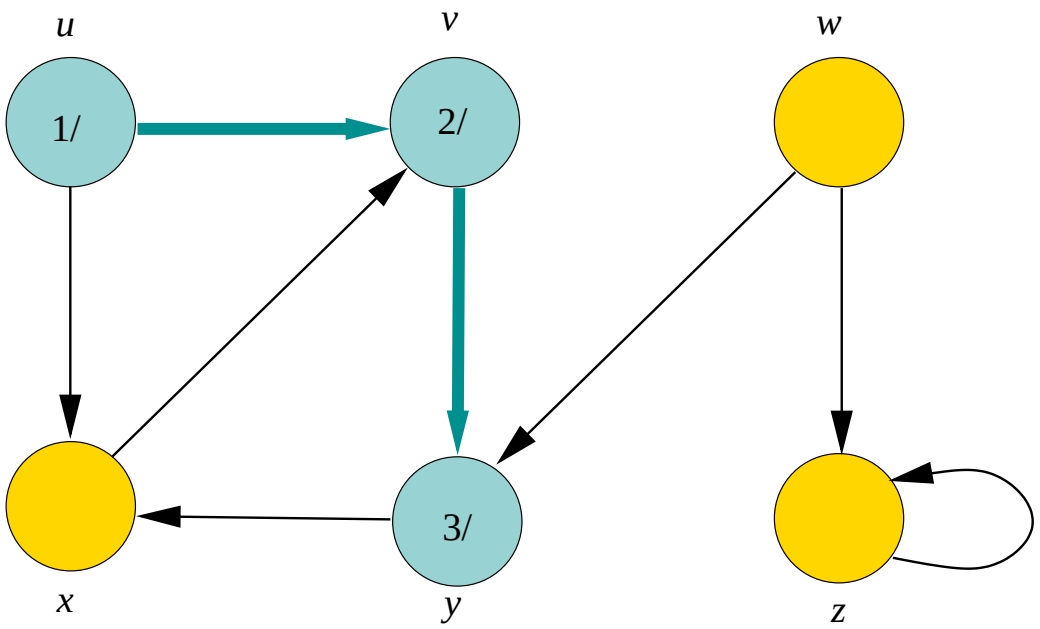
با توجه به تغییر رنگ‌ها هر رأس دقیقاً یک بار ملاقات می‌شود و اگر گراف جهت‌دار باشد هر یال یک‌بار و اگر عادی باشد دو بار ملاقات می‌شود.

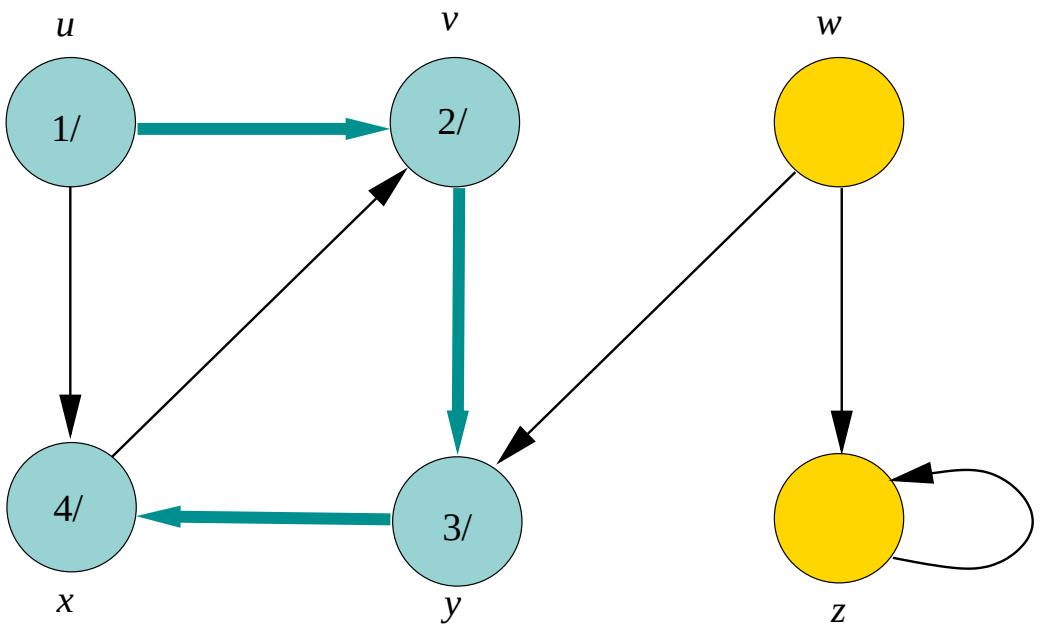
پس الگوریتم از $\Theta(V+E)$ (خطی) است.

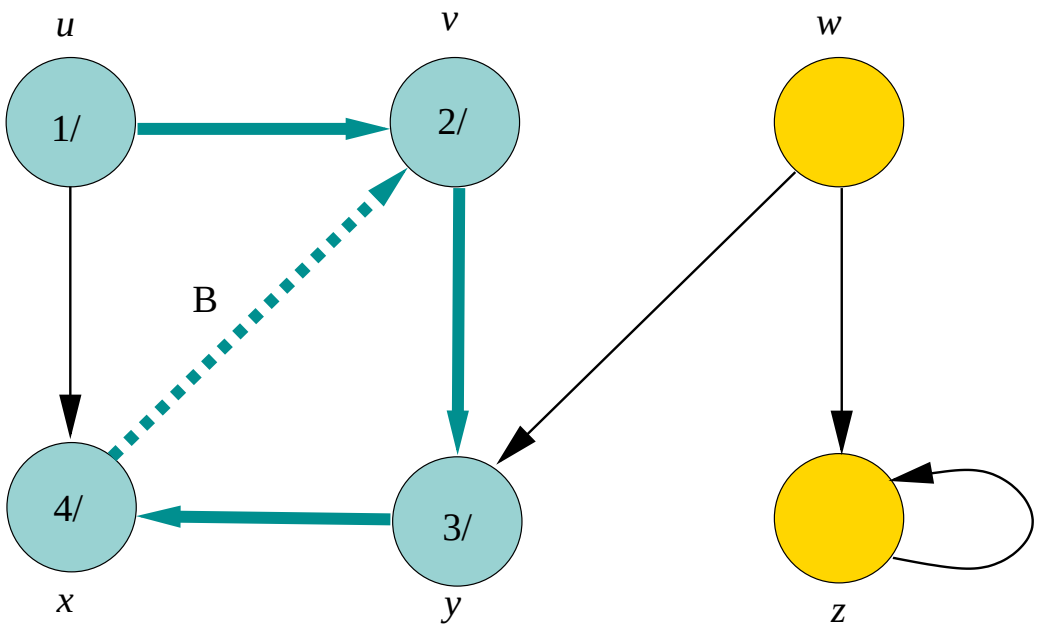

u
v
w
x
y
z

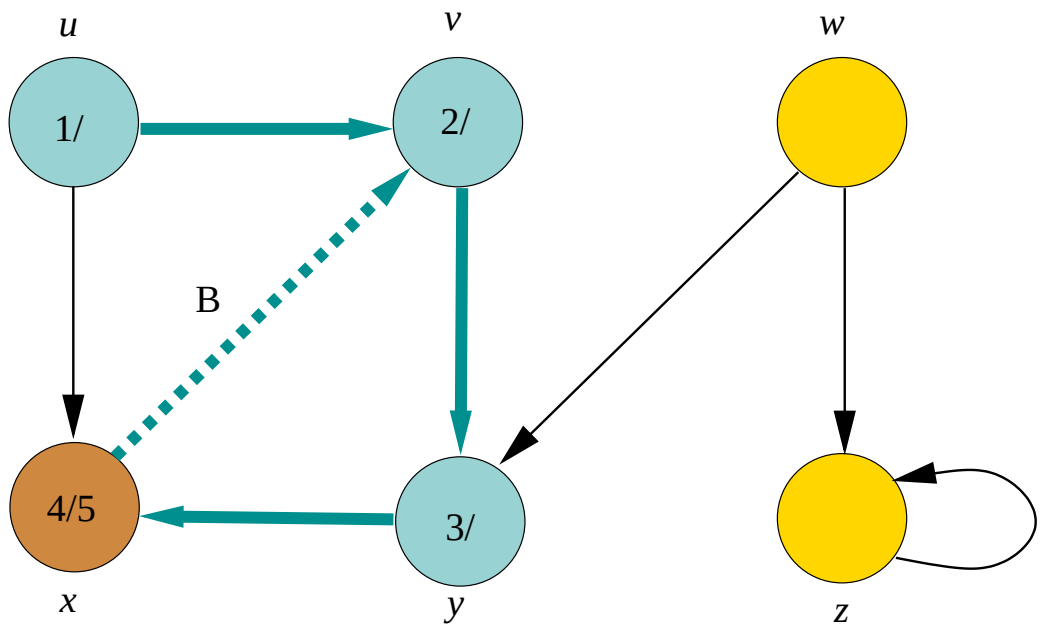

u
v
w
1/
x
y
z

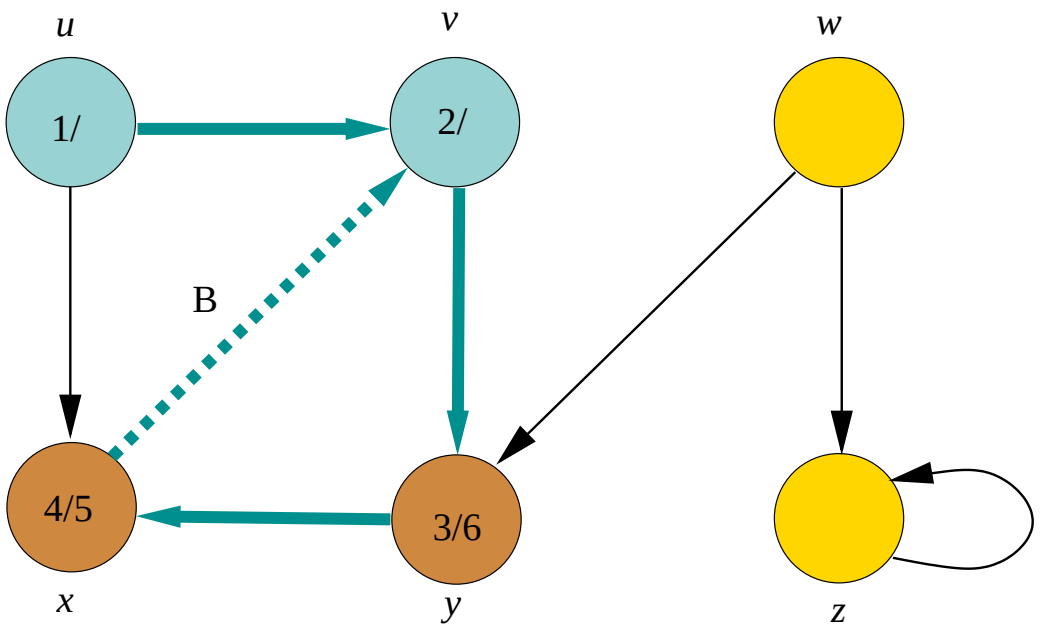

u
v
w
1/
2/
x
y
z

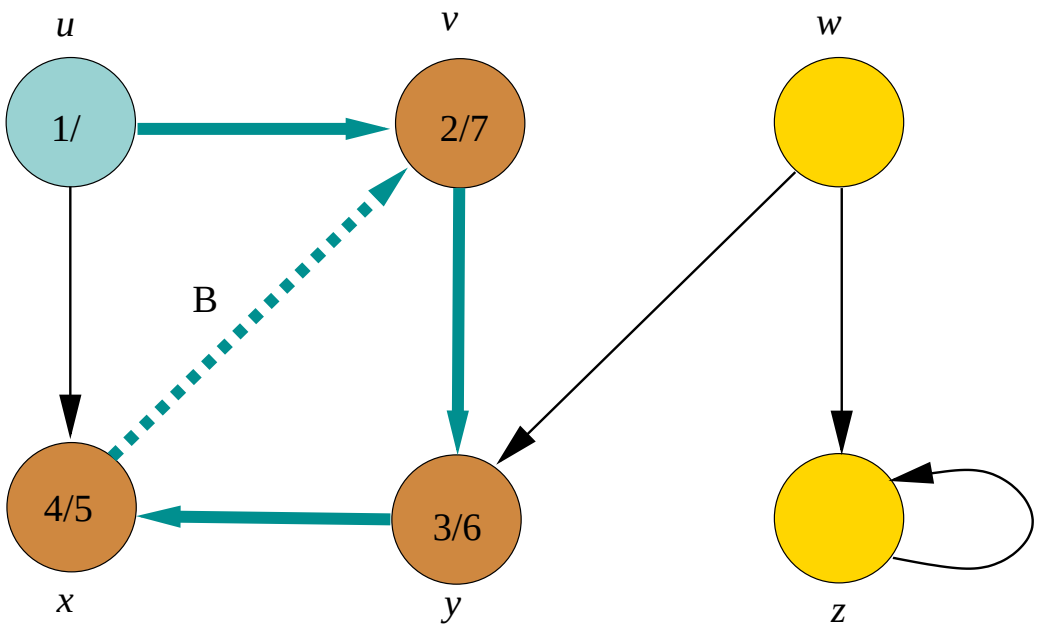

u
v
w
1/
2/
3/
x
y
z

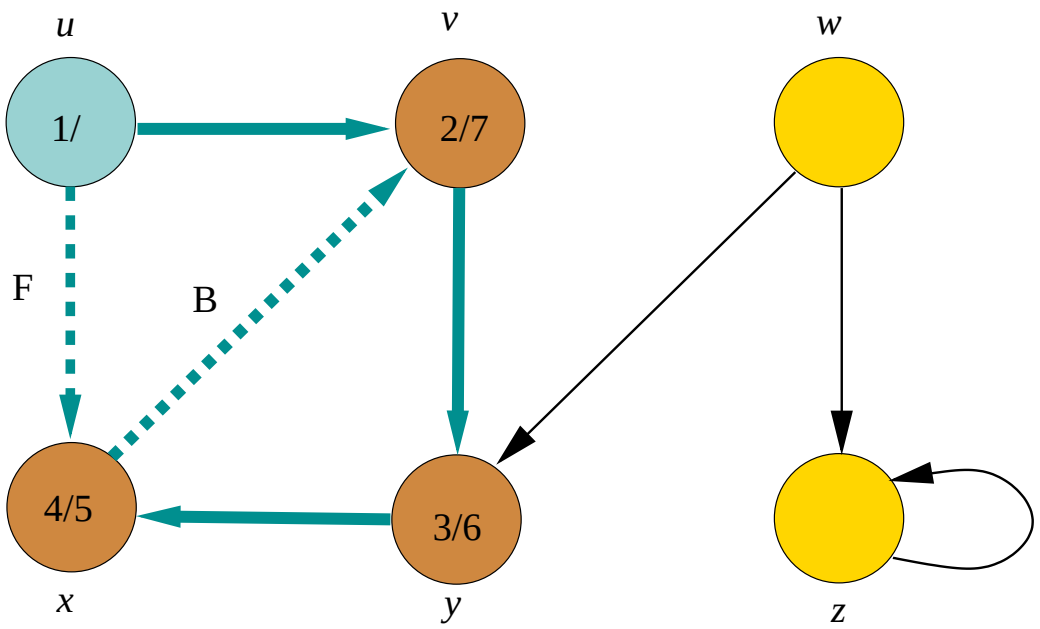

u
v
w
1/
2/
4/
3/
x
y
z

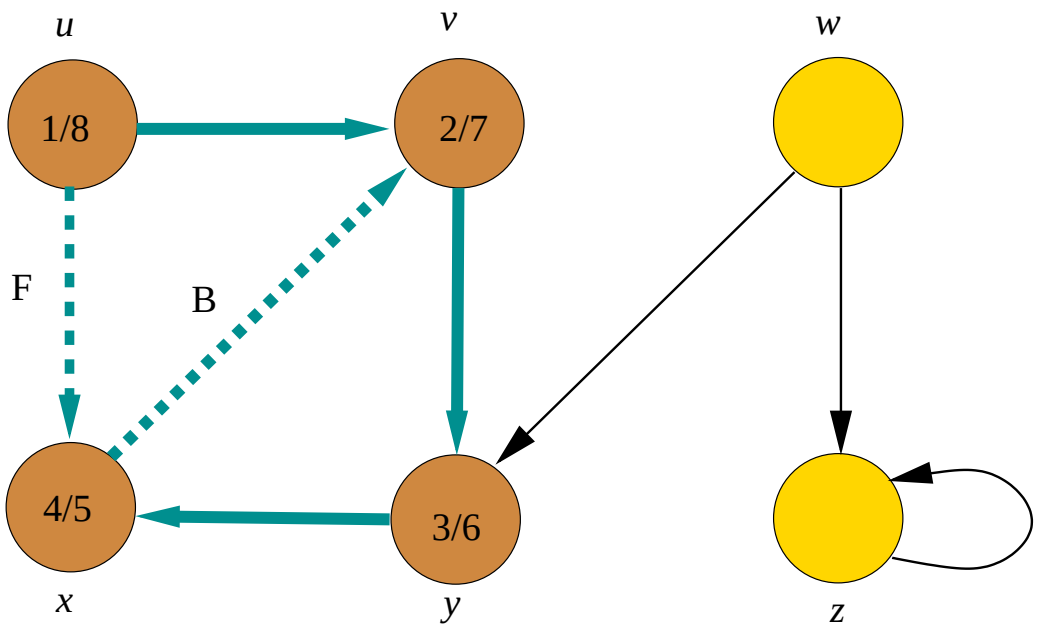

u
v
w
1/
2/
B
4/
3/
x
y
z

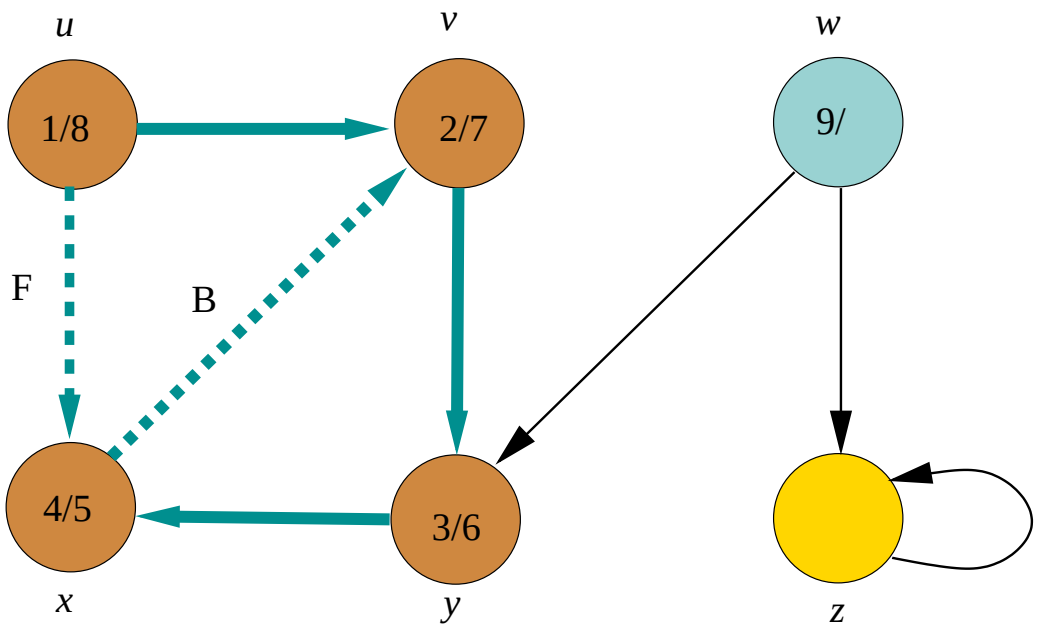

u
v
w
1/
2/
B
4/5
3/
x
y
z

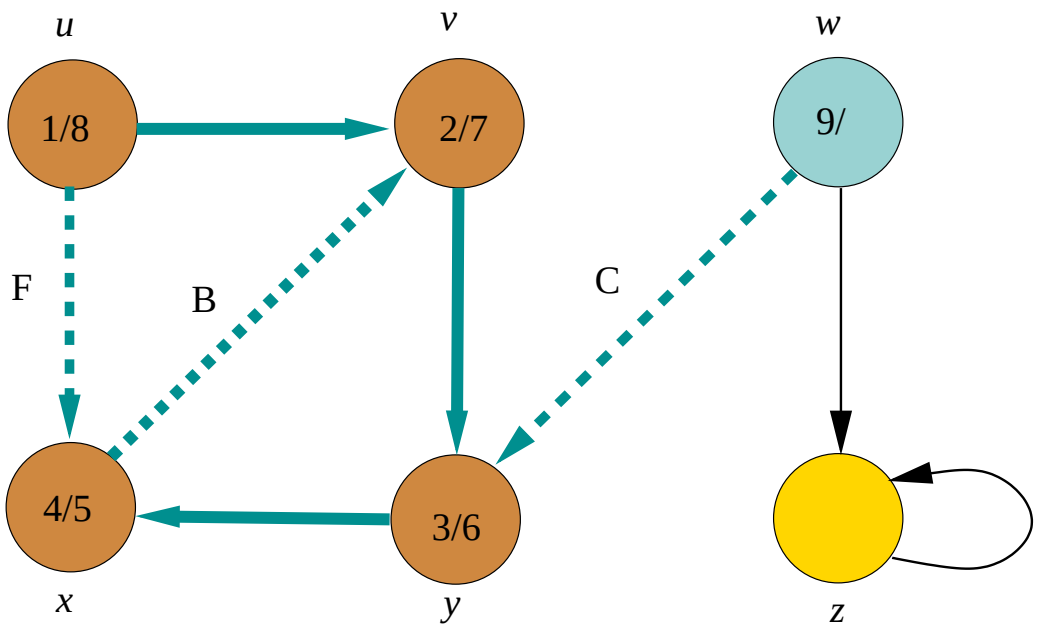

u
v
w
1/
2/
B
4/5
3/6
x
y
z

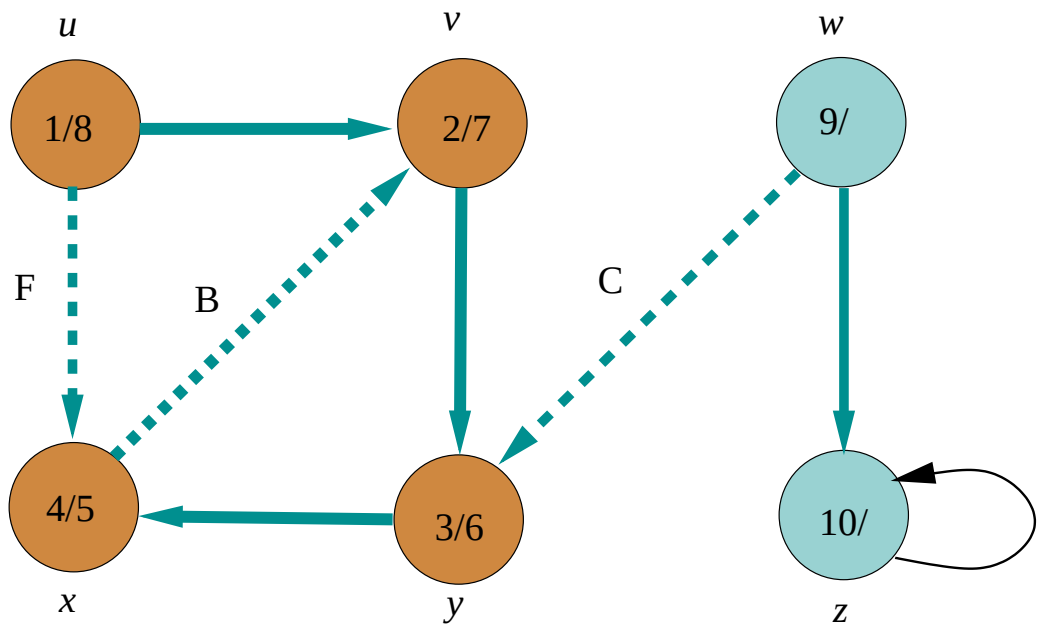

u
v
w
1/
2/7
B
4/5
3/6
x
y
z

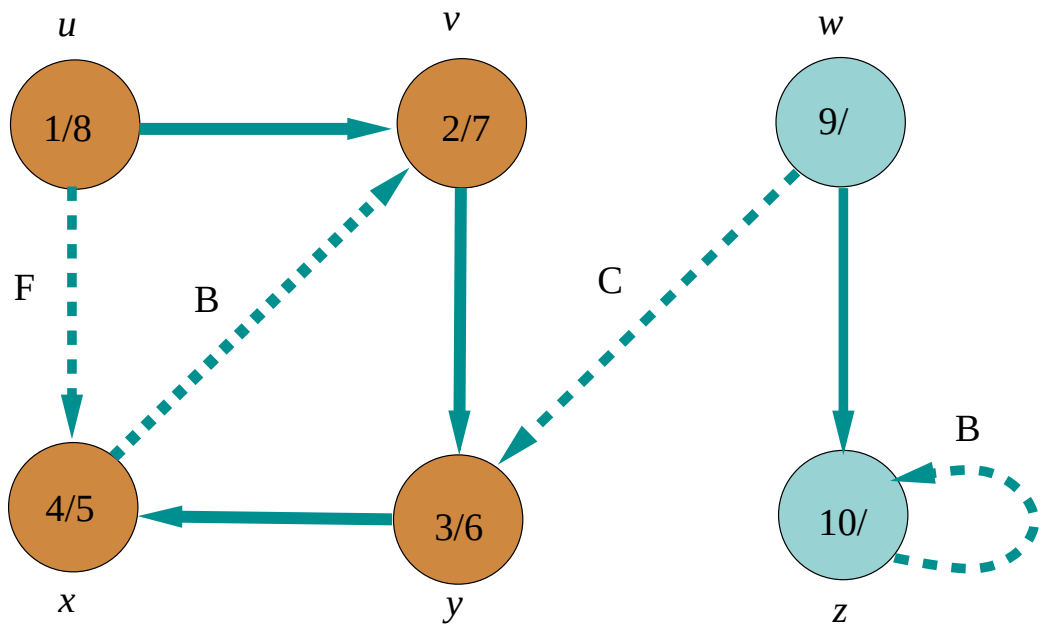

u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z

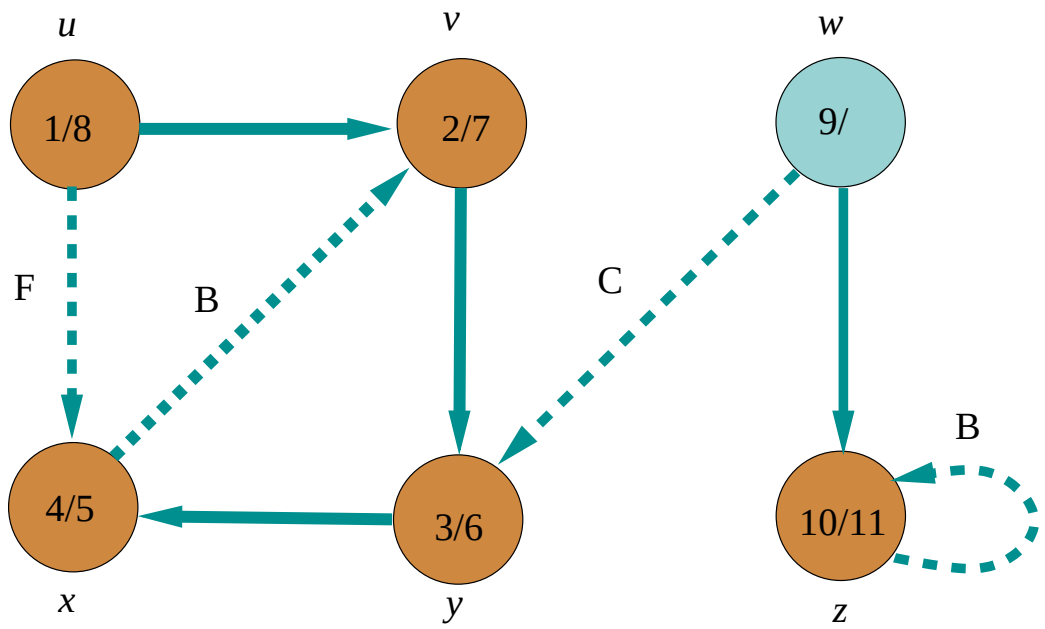

u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z

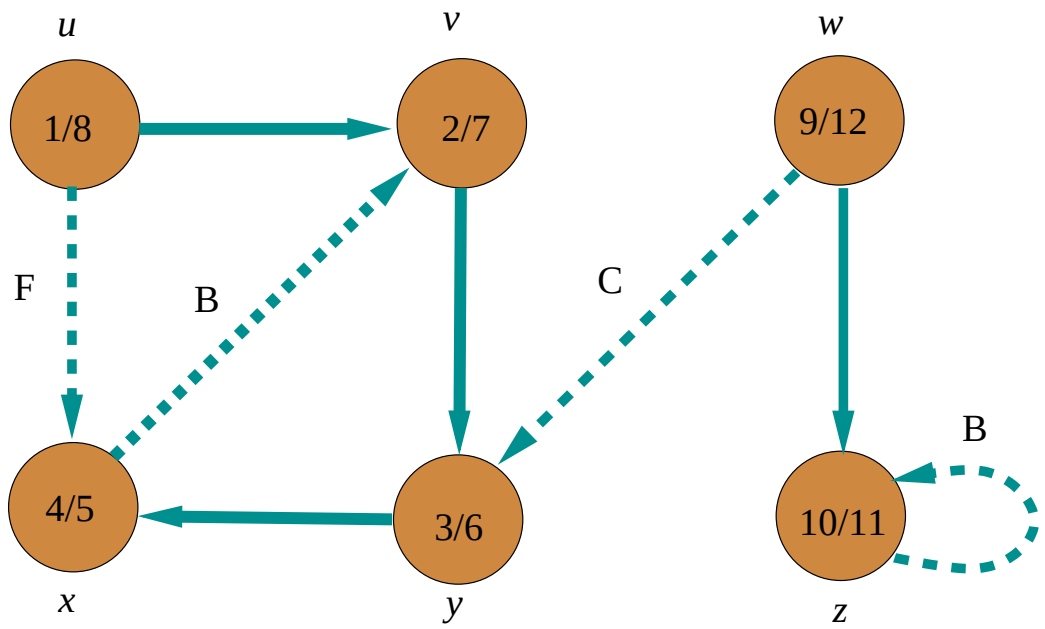

u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
B
x
y
z


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/11
x
y
z


u
v
w
1/8
2/7
9/12
F
B
C
B
4/5
3/6
10/11
x
y
z

جست‌وجوی عمق‌اول: برچسب‌ها

- **زمان شروع ملاقات:** $d[u]$
- **زمان ختم ملاقات:** $f[u]$

$$d[u] \leq f[u]$$

جست‌وجوی عمق‌اول: برچسب‌ها

رنگ یک رأس

- سفید: قبل از ملاقات
- خاکستری: در مدت ملاقات
- سیاه: پس از ملاقات

رأس‌های خاکستری تشکیل یک زنجیره‌ی خطی می‌دهند (پشته‌ی فراخوانی بازگشتی)

جست‌وجوی عمق‌اول: پرانتزگذاری

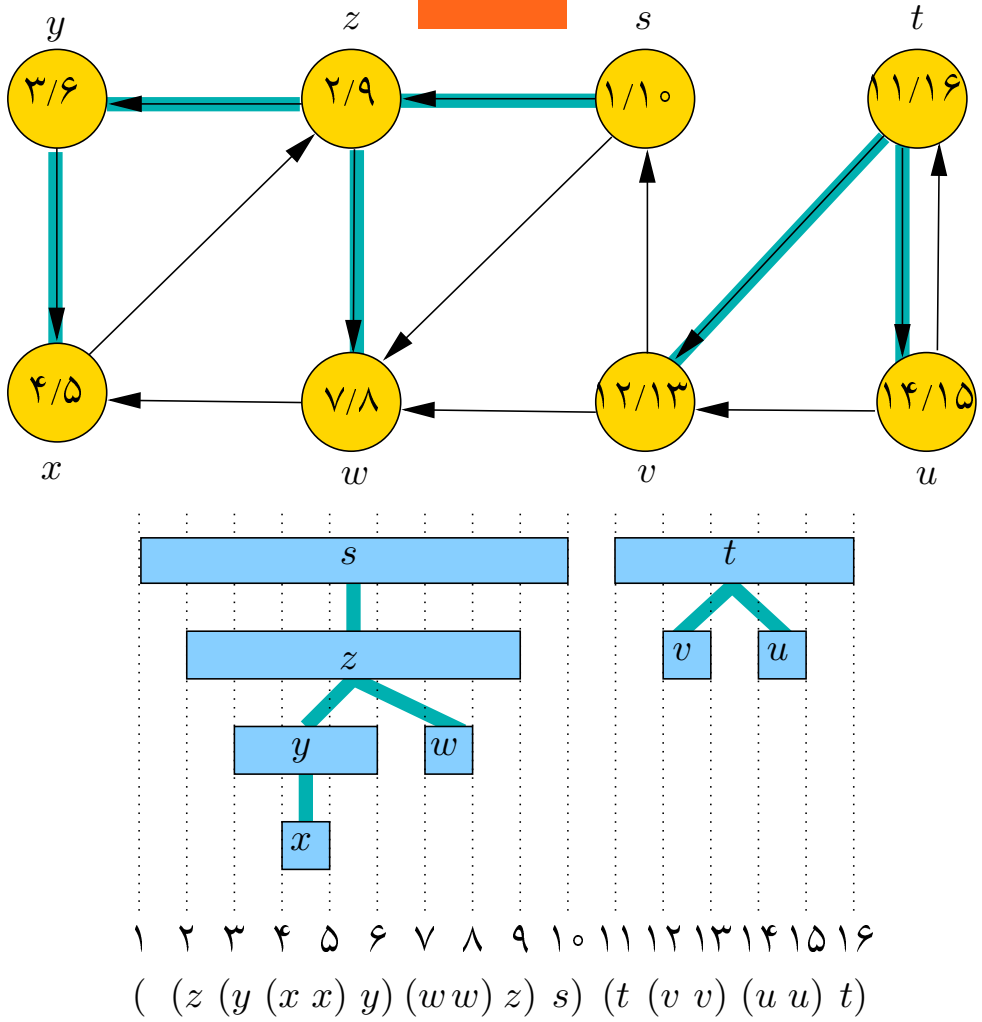
ساختار پرانتزها

- **شروع ملاقات u با "(u"**
- **ختم ملاقات u با "u)"**

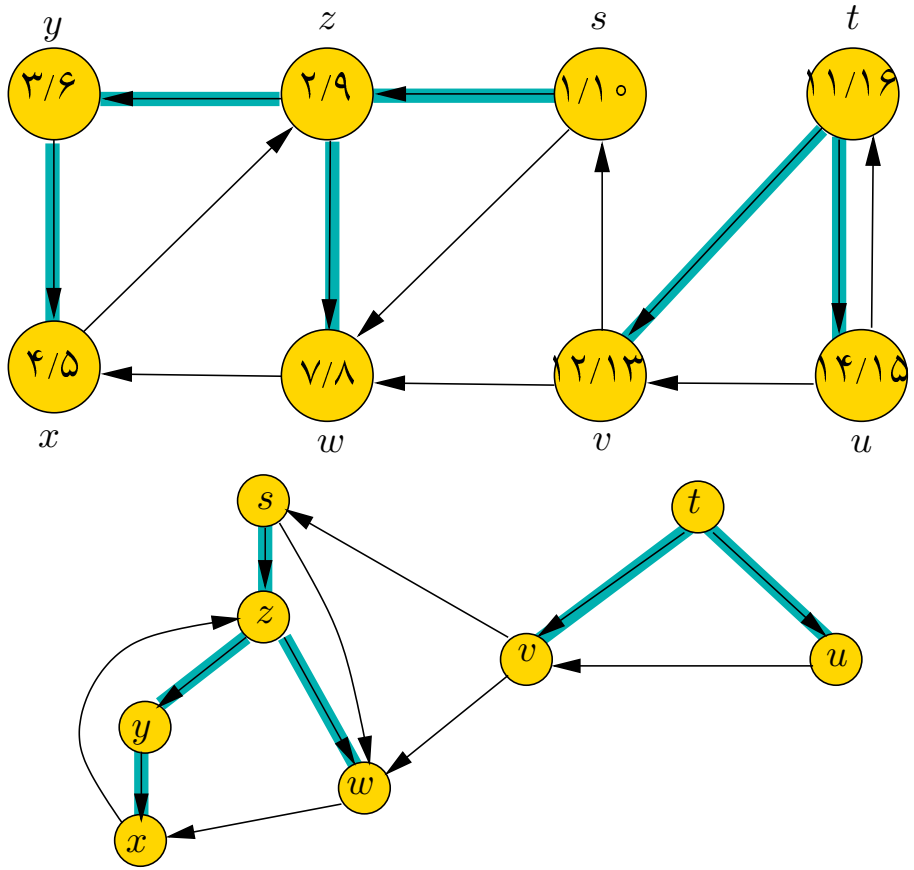
عبارت حاصل کاملاً پرانتزگذاری شده است.

اثبات: ساده

مثال


y
z
s
t
۳/۶
۲/۹
۱/۱۰
۱۱/۱۶
۴/۵
۷/۸
۱۲/۱۳
۱۴/۱۵
x
w
v
u
s
t
z
v
u
y
w
x
۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۳ ۱۴ ۱۵ ۱۶
((z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

دسته‌بندی یال‌ها



- **یال درختی** (tree edge): (از رأس خاکستری به رأس سفید)
  - **یک رأس سفید جدید پیدا می‌کند**
  - **یال‌های درختی یک جنگل فراگیر** (spanning forest) **است**
- **یال پس‌سو** (back edge): (از رأس خاکستری به رأس خاکستری)
  - **از اولاد به جد**
- **یال پیش‌سو** (forward edge): (از رأس خاکستری به رأس سیاه)
  - **از جد به اولاد** (یالی از درخت نیست)
- **یال چپ‌سو** (cross edge): (از رأس خاکستری به رأس سیاه)
  - **بقیه‌ی یال‌ها، بین گره‌های دو زیر درخت**

دسته‌بندی یال‌ها: نکات

رابطه‌های اولاد/جد برای یال‌های درختی تعریف می‌شوند.
یال‌های درختی و پس‌سو مهم هستند.
اکثر الگوریتم‌ها بین یال‌های پیش‌سو و چپ‌سو فرقی قایل نیستند.

تمرین: چه‌گونه یال‌های پیش‌سو را از چپ‌سو تفکیک کنیم؟

جست‌وجوی عمق‌اول: لم

لم: در جست‌وجوی عمق‌اول گراف‌های بدون جهت، هر یال یا درختی است و یا پس‌سو.

اثبات: برهان خلف

تمرین

شرط لازم و کافی برای این‌که یک گراف بدون جهت دور داشته باشد آن است که DFS یال پس‌سو داشته باشد.

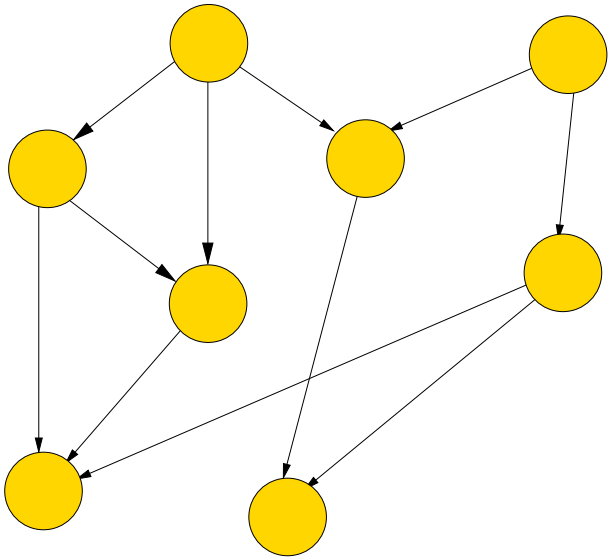
اثبات:

گراف دور داشته باشد: $\Longleftarrow$ DFS **یال پس‌سو دارد**

DFS **یال پس‌سو دارد:** $\Longleftarrow$ **گراف دور دارد**

تشخیص این کار در $O(V)$ **است (نه در** $O(V+E)$**)** | **چرا؟**

گراف جهت‌دار بدون دور (Directed Acyclic Graph: DAG)



رابطه‌ی پیش‌نیازی را مدل می‌کند و در بسیاری کاربردها به‌کار می‌رود: اجرای موازی کارها (ترتیب توپولوژیکی)

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد: $\Longleftarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد: $\Longleftarrow$ DFS یال پس‌سو دارد

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد: $\Longleftarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد: $\Longleftarrow$ DFS یال پس‌سو دارد

فرض: v اولین رأسی است که در دور C ملاقات می‌شود، و u رأس پدر او باشد،

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد $\Longrightarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد $\Longrightarrow$ DFS یال پس‌سو دارد

فرض: v اولین رأسی است که در دور C ملاقات می‌شود، و u رأس پدر او باشد،

– در زمان ملاقات v بقیه‌ی رأس‌های C سفید هستند.

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد: $\Longleftarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد: $\Longleftarrow$ DFS یال پس‌سو دارد

فرض: v اولین رأسی است که در دور C ملاقات می‌شود، و u رأس پدر او باشد،

– در زمان ملاقات v بقیه‌ی رأس‌های C سفید هستند.

– ملاقات v تمام نمی‌شود تا همه‌ی رأس‌های سفیدی قابل دست‌رسی از v سیاه شوند.

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد: $\Longleftarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد: $\Longleftarrow$ DFS یال پس‌سو دارد

فرض: v اولین رأسی است که در دور C ملاقات می‌شود، و u رأس پدر او باشد،

– در زمان ملاقات v بقیه‌ی رأس‌های C سفید هستند.

– ملاقات v تمام نمی‌شود تا همه‌ی رأس‌های سفیدی قابل دست‌رسی از v سیاه شوند.

$\Longleftarrow$ (u, v) یال پس‌سوست.

تشخیص یک گراف جهت‌دار بدون دور

شرط لازم و کافی برای این‌که یک گراف جهت‌دار دور داشته باشد آنست که DFS یال پس‌سو داشته باشد.

$\Longleftarrow$: DFS یال پس‌سو دارد: $\Longrightarrow$ گراف دور دارد

$\Longrightarrow$: گراف دور داشته باشد: $\Longrightarrow$ DFS یال پس‌سو دارد

فرض: v اولین رأسی است که در دور C ملاقات می‌شود، و u رأس پدر او باشد،

– در زمان ملاقات v بقیه‌ی رأس‌های C سفید هستند.

– ملاقات v تمام نمی‌شود تا همه‌ی رأس‌های سفیدی قابل دست‌رسی از v سیاه شوند.

$\Longleftarrow$ (u, v) یال پس‌سوست.

تشخیص این کار در $O(V + E)$ است (نه $O(V)$). چرا؟

مرتب‌سازی توپولوژیکی (Topological Sort)

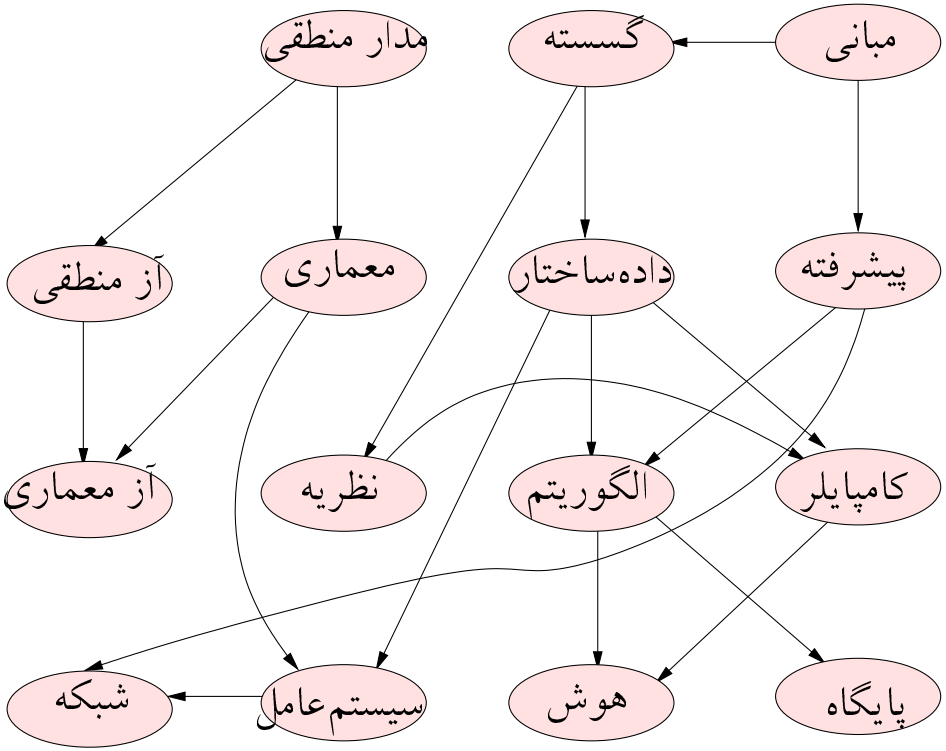
□ ورودی: یک DAG

□ خروجی: ترتیبی از رأس‌ها به‌طوری که

• برای هر یال (u, v) در گراف، u قبل از v بیاید.

اگر گراف دور داشته باشد، ترتیب توپولوژیکی امکان‌پذیر نیست.

مرتب‌سازی توپولوژیکی: مثال



الگوریتم برای مرتب‌سازی توپولوژیکی

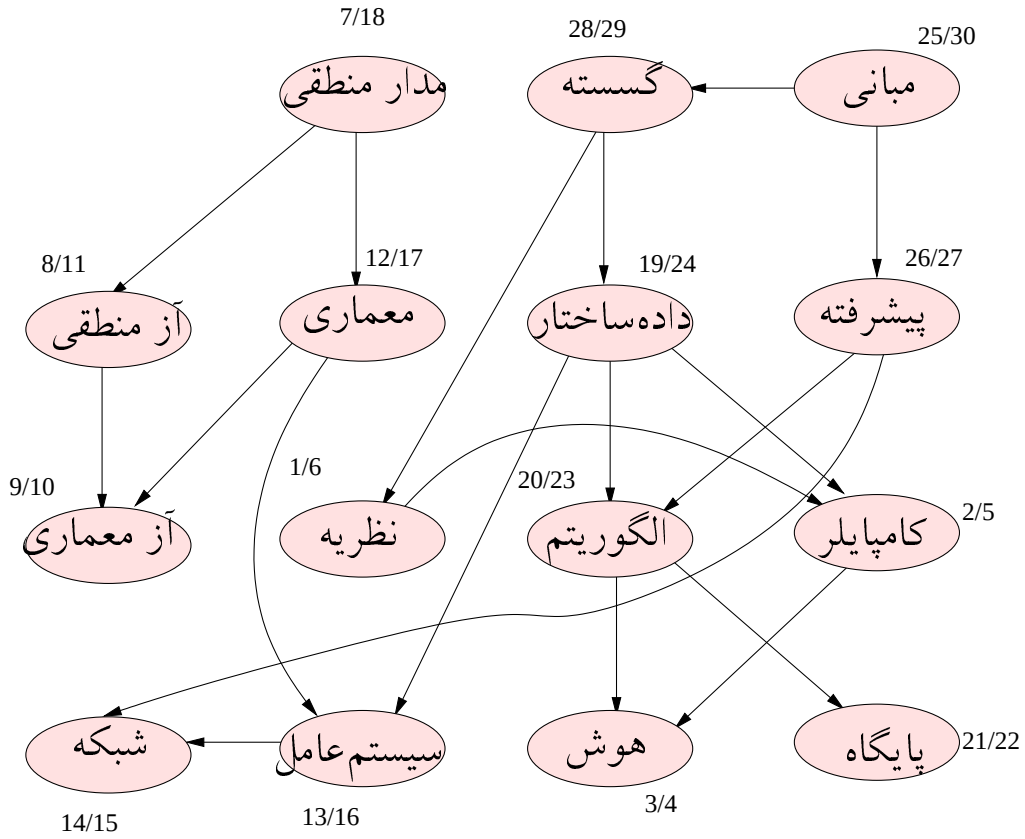
TOPOLOGICAL-SORT (G)
1 Call DFS(G) and compute $f[u]$ for all $u \in V$
2 in DFS, as each vertex finishes, insert it onto the front of a list L
3 **return** L

تحلیل:

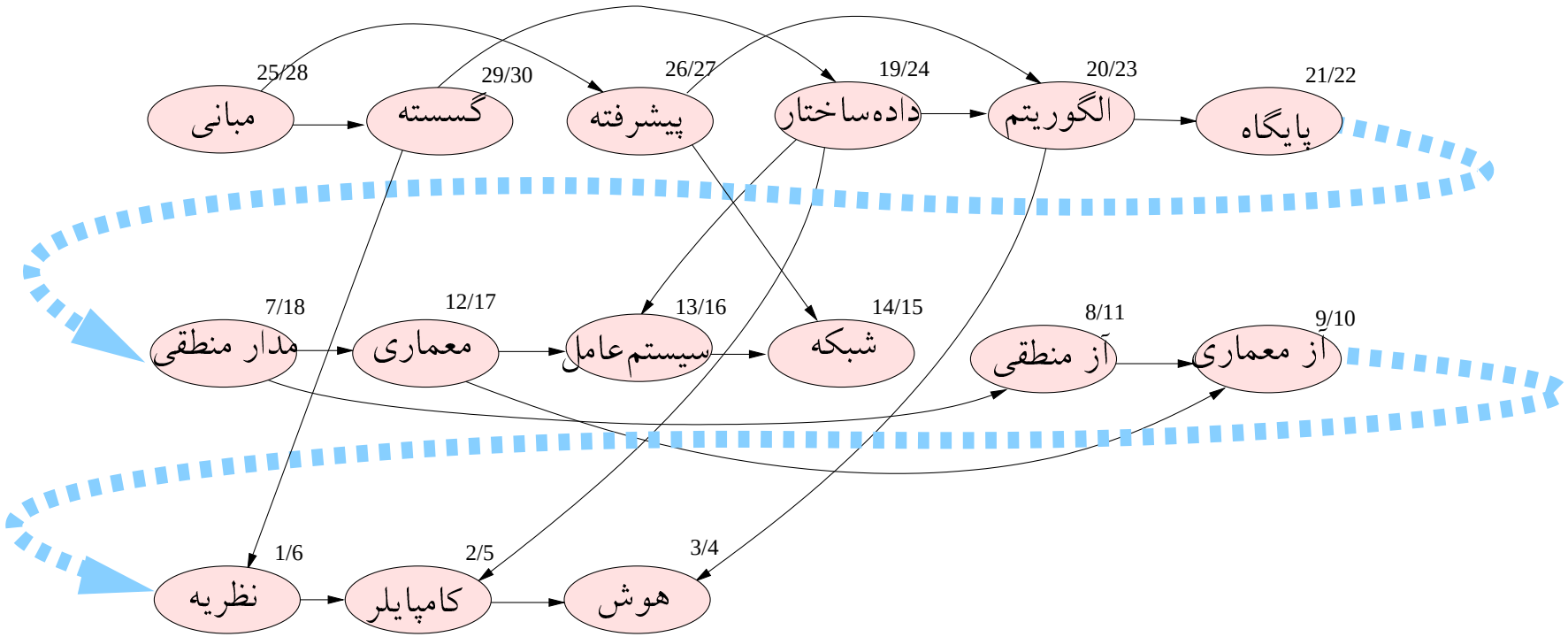
$O(V+E)$ **برای** DFS

$O(V+E) \Longleftarrow$ **برای درج در ابتدای لیست** $O(V)$

مثال درس‌ها


7/18
مدار منطقی
28/29
گسسته
25/30
مبانی
8/11
آز منطقی
12/17
معماری
19/24
داده‌ساختار
26/27
پیشرفته
9/10
آز معماری
1/6
نظریه
20/23
الگوریتم
2/5
کامپایلر
شبکه
سیستم‌عامل
هوش
پایگاه
21/22
14/15
13/16
3/4

ترتیب توپولوژیکی



مرتب‌سازی توپولوژیکی: اثبات درستی

ادعا: $(u,v) \in E \implies f[u] > f[v]$

اثبات: هنگامی که (u,v) ملاقات می‌شود، u خاکستری است

- **اگر v خاکستری باشد، (u,v) پس‌سو است $\implies$ دور (تناقض!)**
- **اگر v سفید باشد، v فرزند u می‌شود** $\implies f[u] > f[v]$
- **اگر v سیاه باشد،** $\implies f[u] > f[v]$

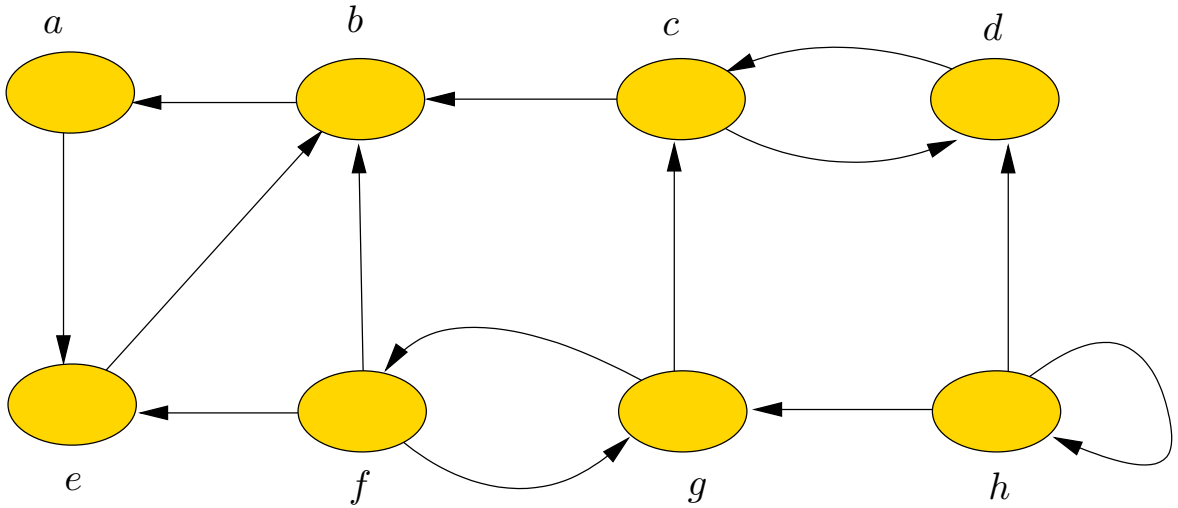
اجزاء قویاً هم‌بند (Strongly Connected Components)

در یک گراف جهت‌دار $G = (V, E)$ رابطه‌ی R را بر روی V تعریف می‌کنیم:

uRv اگگر $u \rightsquigarrow v$ و $v \rightsquigarrow u$

R یک رابطه‌ی هم‌ارزی $\Longleftarrow$ V به مجموعه‌هایی افراز می‌شود که آن‌ها را **اجزاء قویاً هم‌بند** می‌گوییم.

اجزاء قویاً هم‌بند: مثال

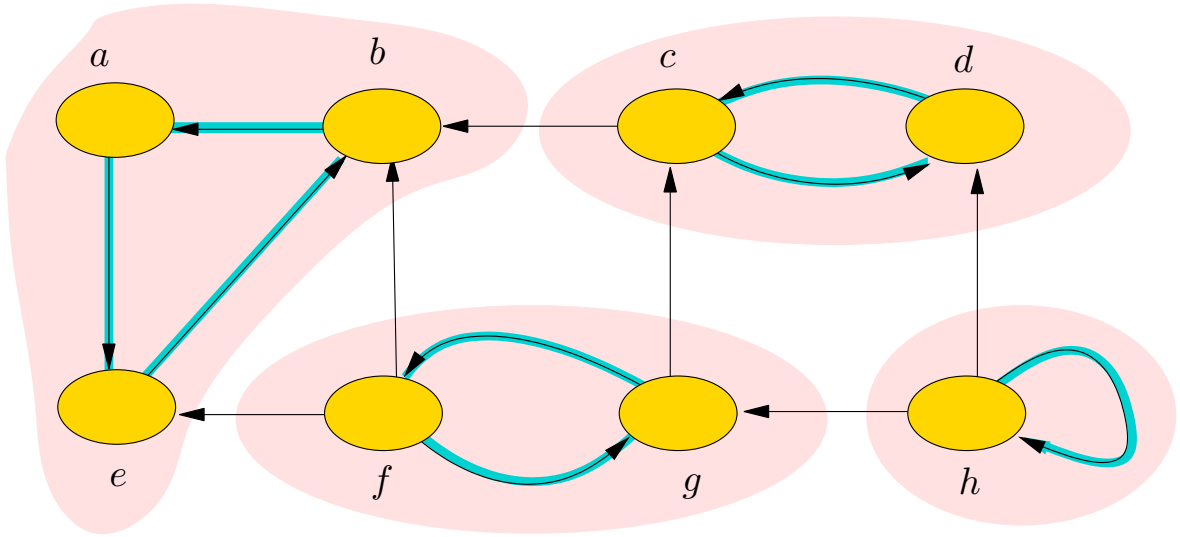


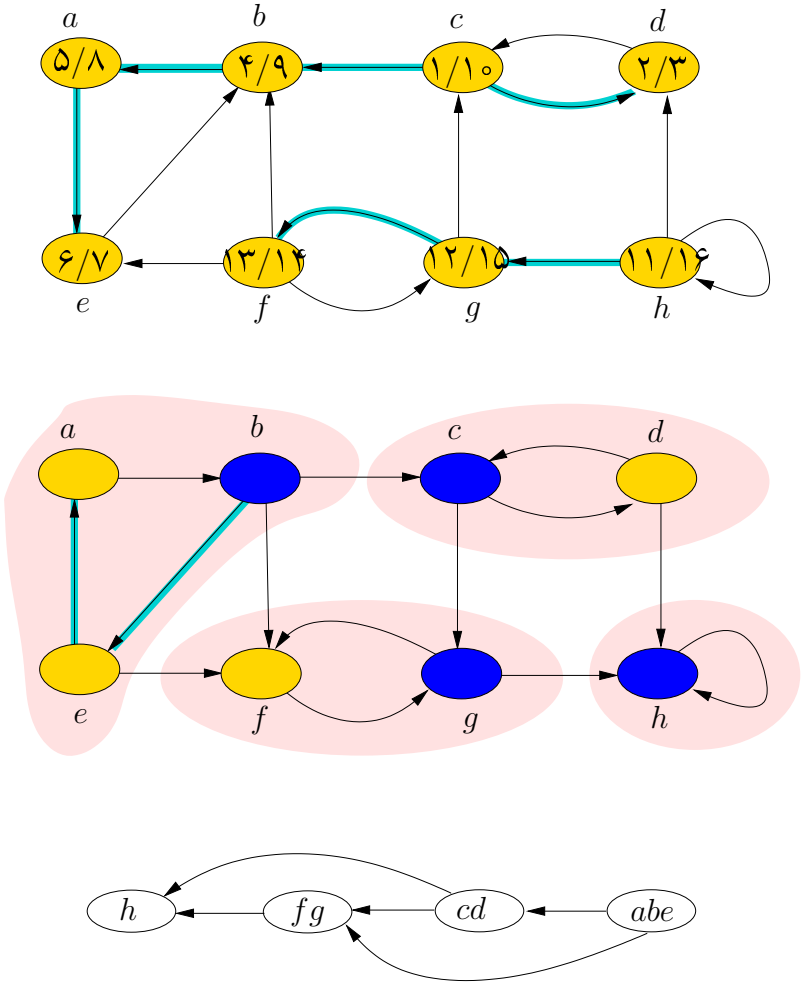
اجزاء قویاً هم‌بند: الگوریتم

STRONGLY-CONNECTED-COMPONENTS (G)
1 Call DFS(G) and compute $f[u]$ for all $u \in V$
2 Compute G^T
3 Call DFS(G^T), but in the main loop of DFS
consider the vertices in order of decreasing $f[u]$
4 Output the vertices of each tree in DFS-forest of Step 3 as one SCC

تحلیل: $O(V + E)$

مثال




a
b
c
d
۵/۸
۴/۹
۱/۱۰
۲/۳
۶/۷
۱۳/۱۴
۱۲/۱۵
۱۱/۱۶
e
f
g
h
a
b
c
d
e
f
g
h
h
fg
cd
abe

اجزاء قویاً هم‌بند: اثبات درستی الگوریتم

لم: vRu **اگگر** u **و** v **هردو در یک درخت از جنگل عمق‌اول** G **باشند**

$\Longleftarrow$ **: فرض:** uRv

اگر در $\mathrm{DFS}(G^T)$ **مثلاً** u **زودتر ملاقات شود، حتماً** v **قبل از پایان ملاقات** u **ملاقات می‌شود**

$\Longrightarrow$: u و v هردو در یک درخت از جنگل عمق‌اول G به‌نام T هستند

فرض کنید x ریشه‌ی T است $\Longleftarrow$ $f[x] > f[u]$ و $f[x] > f[v]$

$\Longleftarrow$ $x \rightsquigarrow u$ در G^T $\Longleftarrow$ $u \rightsquigarrow x$ در G

اثبات می‌کنیم که در $\mathrm{DFS}(G)$، $d[x] < d[u]$

اگر نباشد، یعنی $d[u] < d[x]$،

چون داریم $u \rightsquigarrow x$ در G، $\Longleftarrow$ $f[x] < f[u]$ (تناقض!)

$\Longleftarrow$ در $\mathrm{DFS}(G)$ داریم، $d[x] < d[u] < f[u] < f[x]$ $\Longleftarrow$ $x \rightsquigarrow u$ در G $\Longleftarrow$ uRx

با همین استدلال، uRx $\Longleftarrow$ uRv

جست‌وجوی عمق‌اول (غیربازگشتی)

```
DFS(G)
 1  for each vertex u ∈ V[G]
 2      do color[u] ← white
 3  time ← 0
 4  for each vertex u ∈ V[G]
 5      do if color[u] = white
 6           then NR-DFS (u)
```

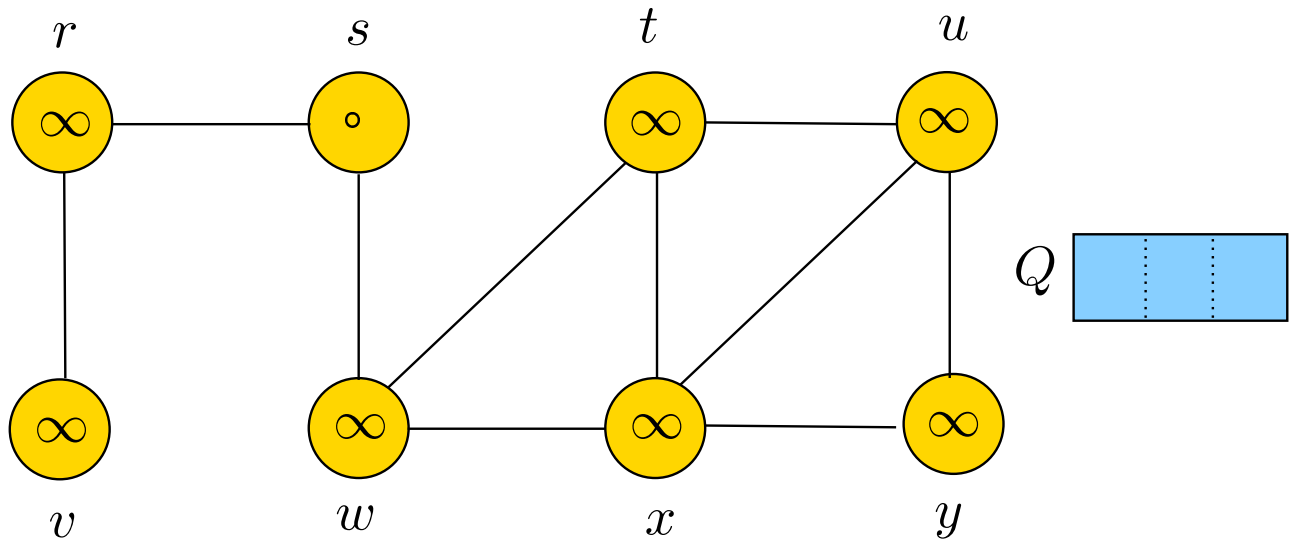
جست‌وجوی عمق‌اول غیر بازگشتی (ادامه)

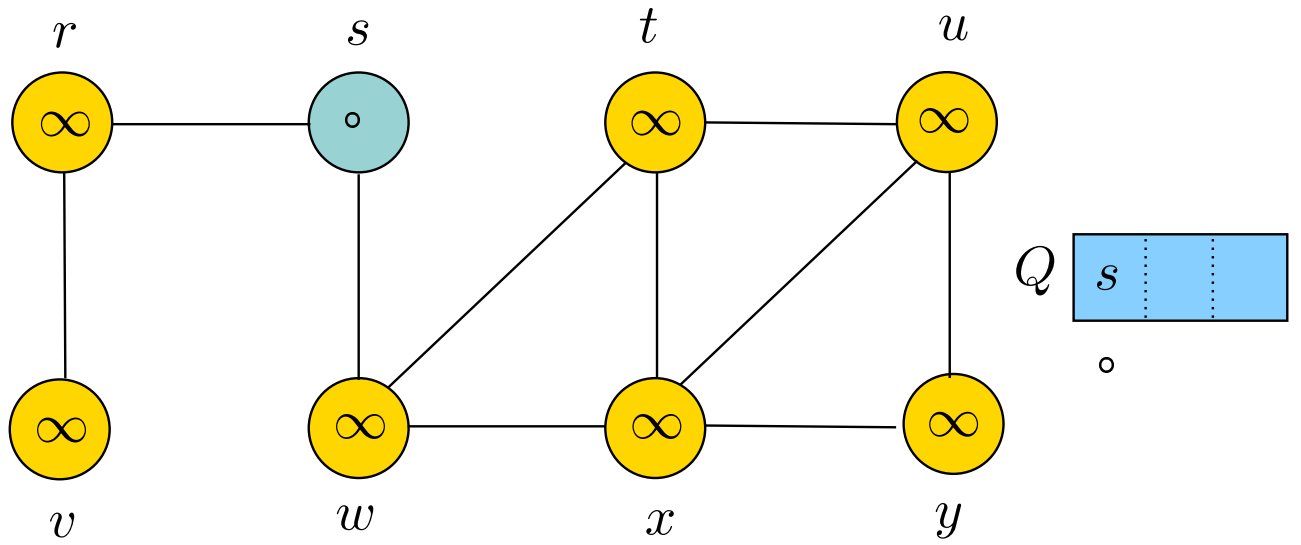
```
NR-DFS(u)
 1  color[u] ← Gray                ▷ رأس سفید ملاقات شد
 2  PUSH(S, u)
 3  while S ≠ φ
 4        do u ← POP (S)
 5           for each v ∈ Adj[u]
 6              do if color[v] = white
 7                    then PUSH (S, v)
 8                         color[v] ← Gray
 9           color[u] ← Black      ▷ اتمام ملاقات: رنگ سیاه
```

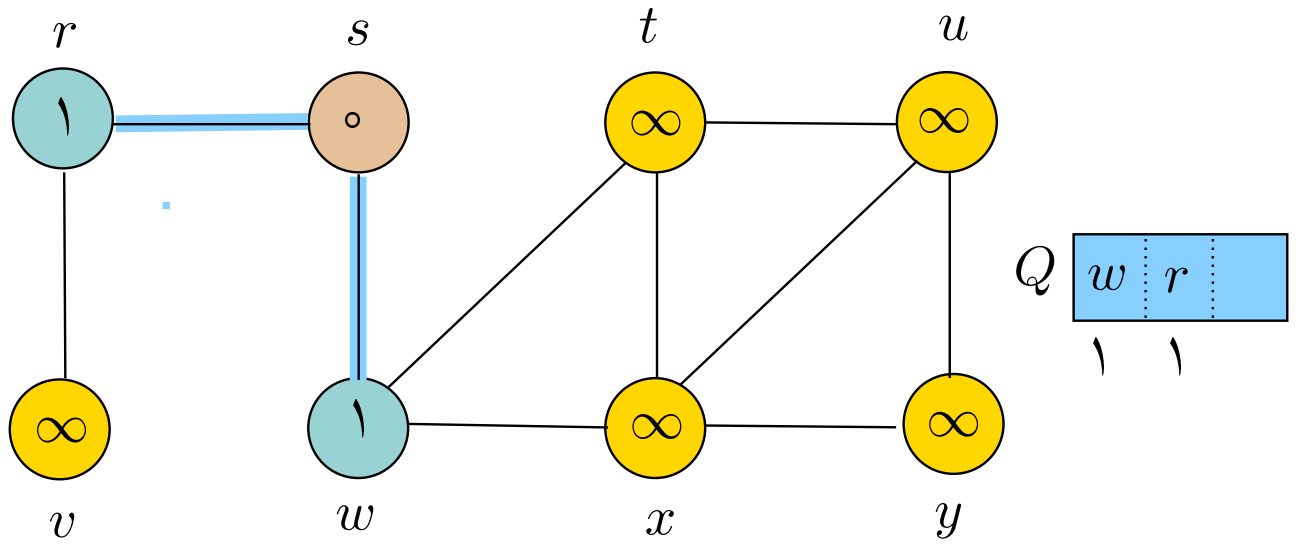
جست‌وجوی سطح‌اول (Breadth-First-Search)

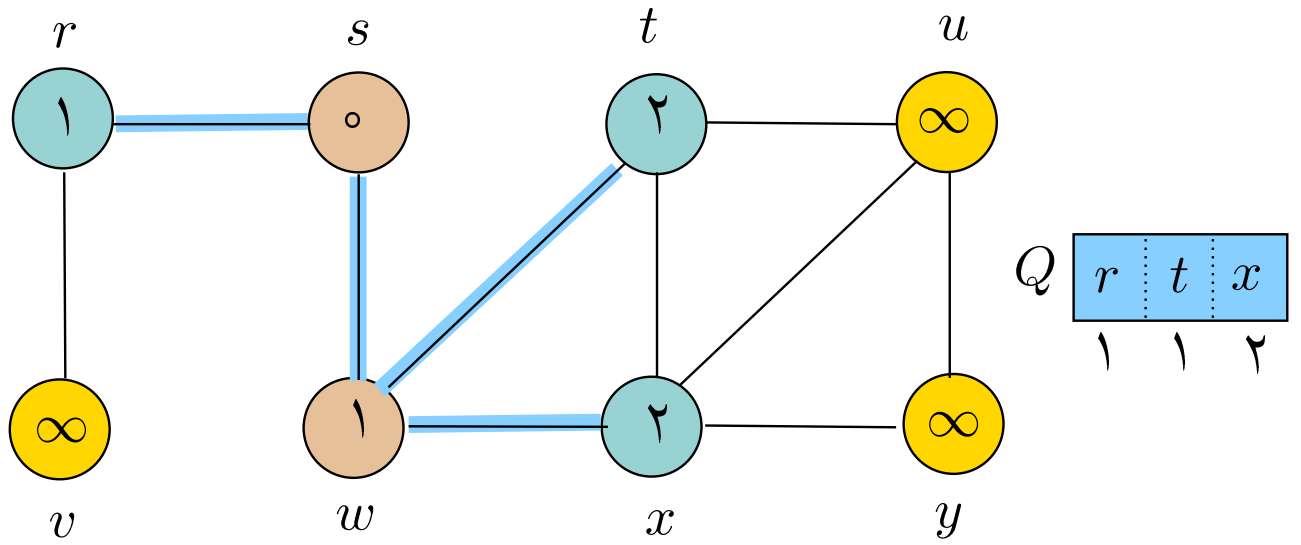
```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2      do color[u] ← white
 3         d[u] ← ∞; π[u] ← null
 4  color[s] ← Gray
 5  d[s] ← 0; π[s] ← null
 6  Q ← {s}
 7  while Q ≠ φ
 8        do u ← DEQUEUE(Q)
 9           for each v ∈ Adj[u]
10               do if color[v] = white
11                     then color[v] ← Gray
12                          d[v] ← d[u] + 1
13                          π[v] ← u
14                          ENQUEUE(Q, v)
15           color[u] ← Black
```

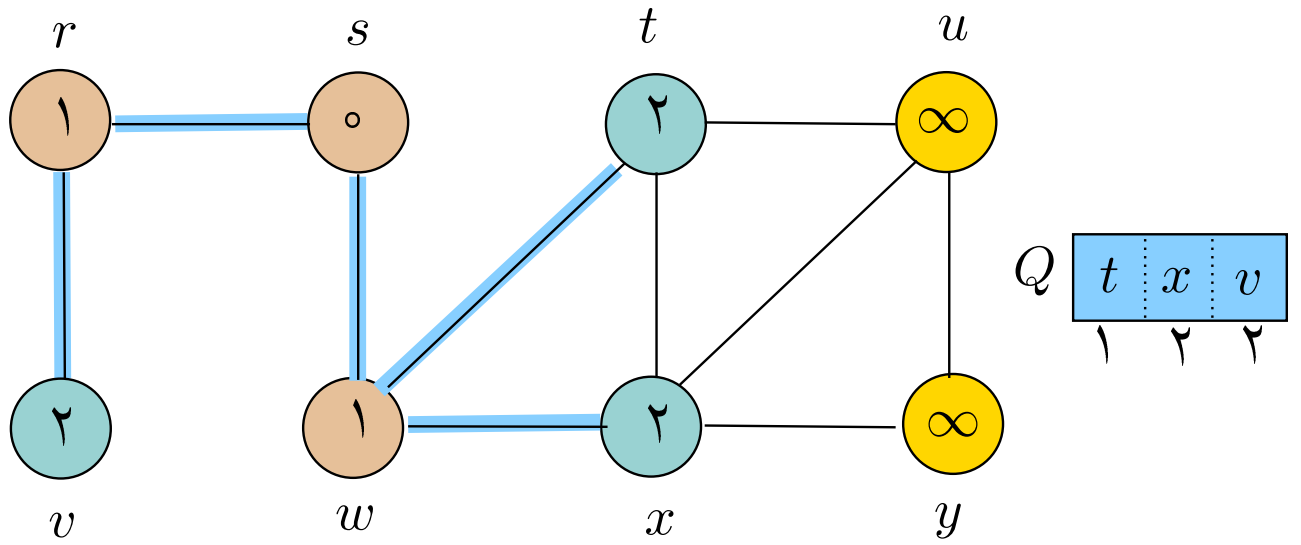
جست‌وجوی سطح‌اول: مثال

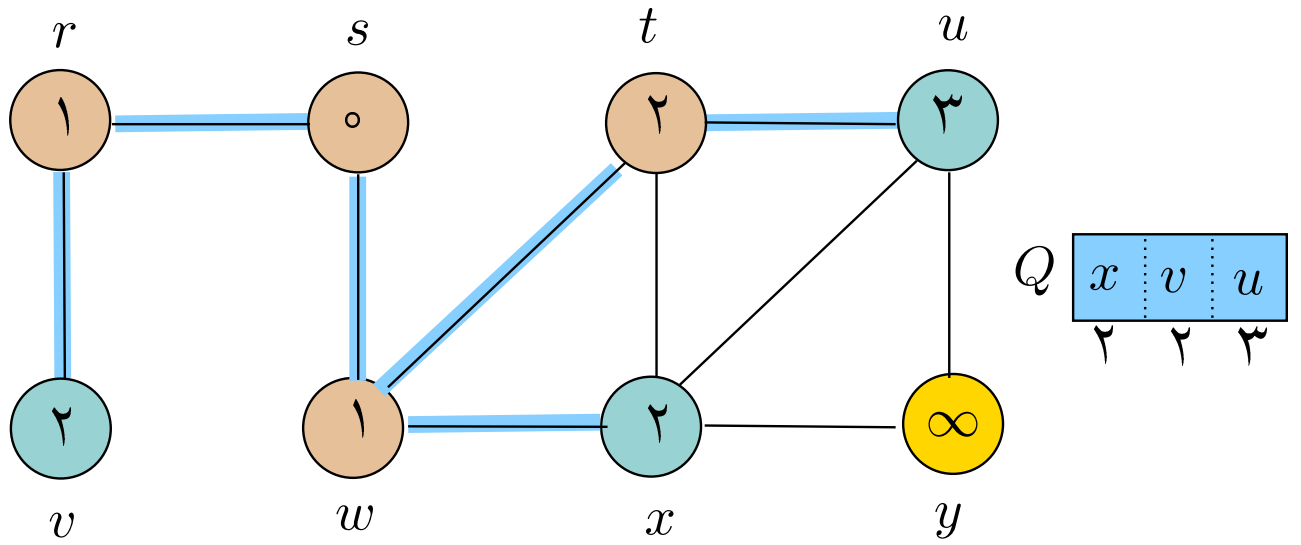

r
s
t
u
∞
0
∞
∞
∞
∞
∞
∞
Q
v
w
x
y

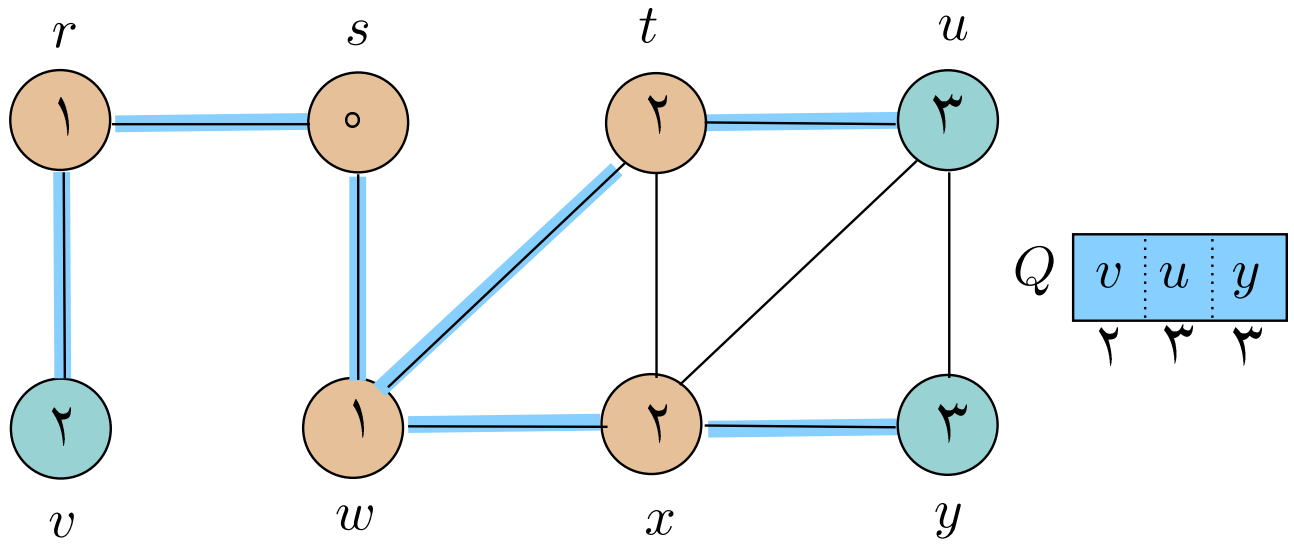

r
s
t
u
∞
0
∞
∞
∞
∞
∞
∞
v
w
x
y
Q
s
0

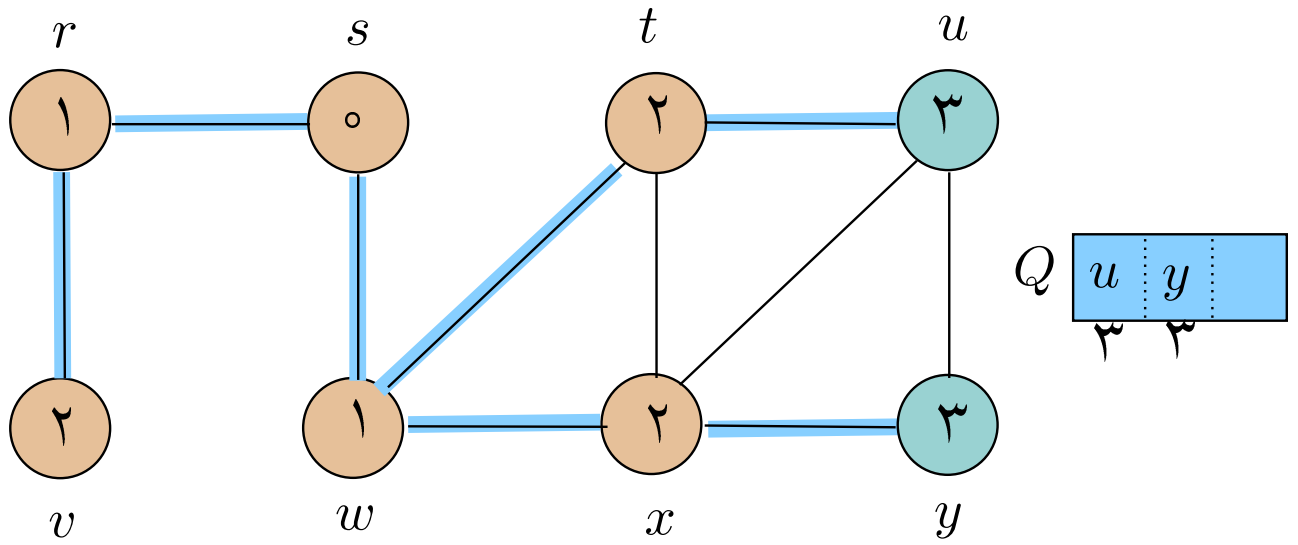

r
s
t
u
۱
۰
∞
∞
∞
۱
∞
∞
v
w
x
y
Q
w
r
۱
۱

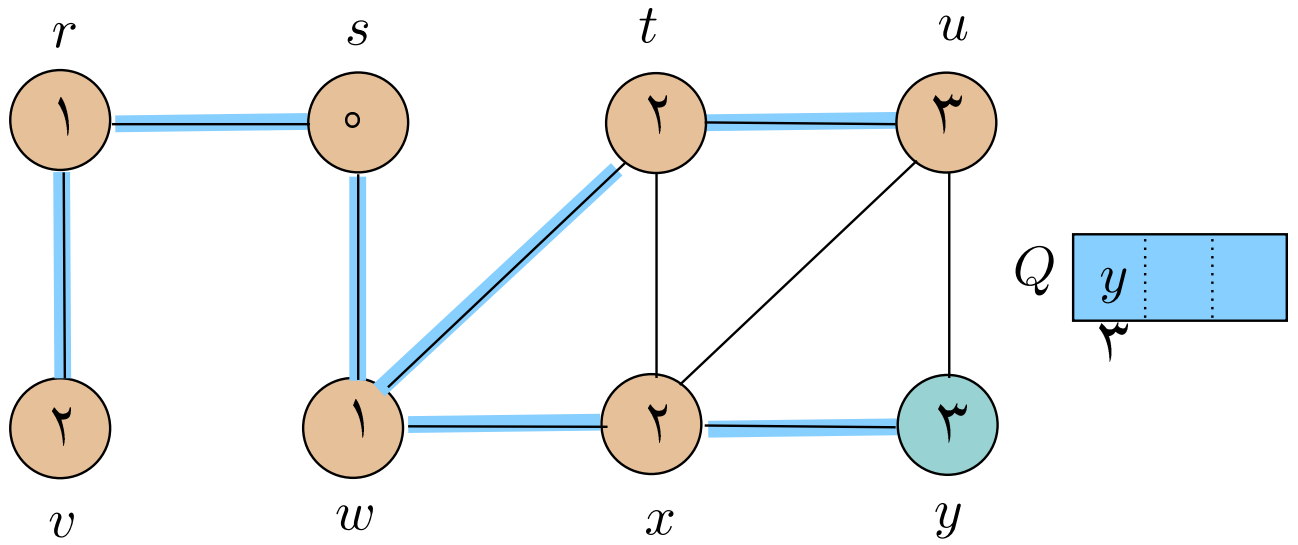

r
s
t
u
۱
۰
۲
∞
∞
۱
۲
∞
v
w
x
y
Q
r
t
x
۱
۱
۲

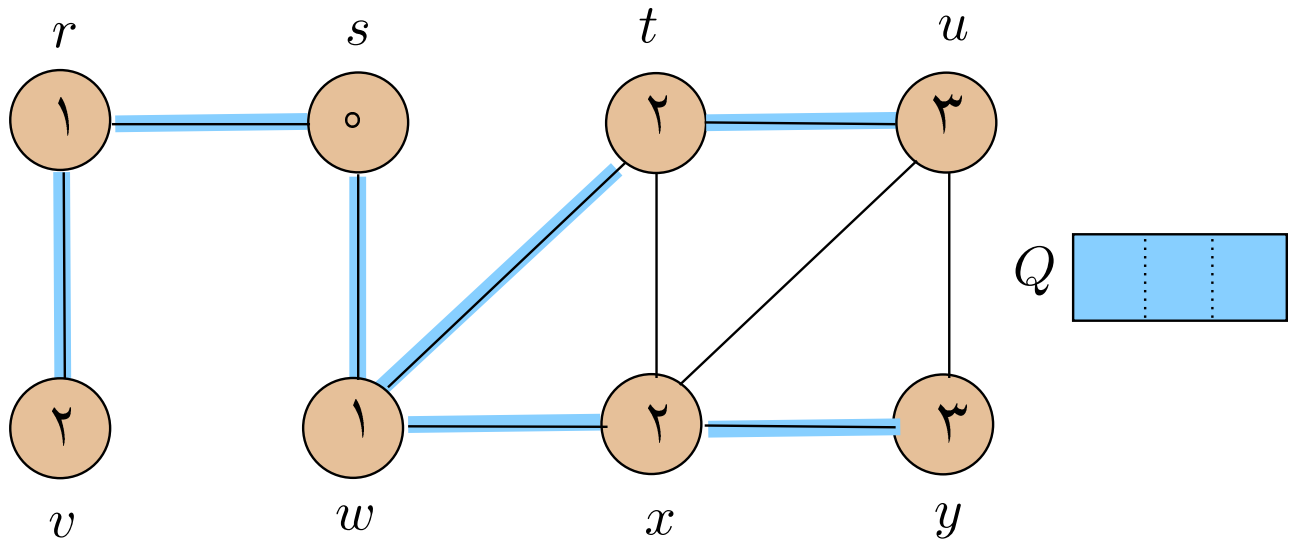

r
s
t
u
۱
۰
۲
∞
۲
۱
۲
∞
v
w
x
y
Q
t
x
v
۱
۲
۲


r
s
t
u
۱
۰
۲
۳
۲
۱
۲
∞
v
w
x
y
Q
x
v
u
۲
۲
۳


r
s
t
u
۱
۰
۲
۳
۲
۱
۲
۳
v
w
x
y
Q
v
u
y
۲
۳
۳


r
s
t
u
۱
۰
۲
۳
۲
۱
۲
۳
v
w
x
y
Q
u
y
۳
۳


r
s
t
u
۱
۰
۲
۳
۲
۱
۲
۳
v
w
x
y
Q
y
۳


r
s
t
u
۱
۰
۲
۳
Q
۲
۱
۲
۳
v
w
x
y

کوتاه‌تریه مسیرها

گراف جهت‌دار و وزن‌دار $G = (V, E)$

وزن: یک تابع $W : E \to \Re$

وزن یک مسیر وزن‌دار $p = v_1 \to v_2 \ldots \to v_k$ **برابر** $w(p) = \sum_{i=i}^{k-1} w(v_i, v_{i+1})$

کوتاه‌ترین مسیر: یک مسیر با کم‌ترین وزن

جست‌وجوی سطح‌اول و کوتاه‌ترین مسیرها

BFS کوتاه‌ترین مسیرها را از نظر تعداد یال (یا اگر وزن همه‌ی یال‌ها برابر باشد) به‌دست می‌آورد.

چرا؟

انواع مسئله‌های کوتاه‌ترین مسیر

- **یافتن همه‌ی کوتاه‌ترین مسیرها از یک مبداء**
  (Single Source Shortest Paths (SSSP))
- **یافتن همه‌ی کوتاه‌ترین مسیرها بین هر زوج رأس**
  (All Pairs Shortest Paths (APSP))
- **یافتن کوتاه‌ترین مسیر بین دو رأس**
  (Single Pair Shortest Paths (SPSP))

نمایش کوتاه‌ترن مسیر

برای هر رأس v، رأس $\pi[v]$ به‌عنوان پدر v در کوتاه‌ترین مسیر ذخیره می‌شود.

$$\rightsquigarrow \pi[v] \longrightarrow v \rightsquigarrow$$

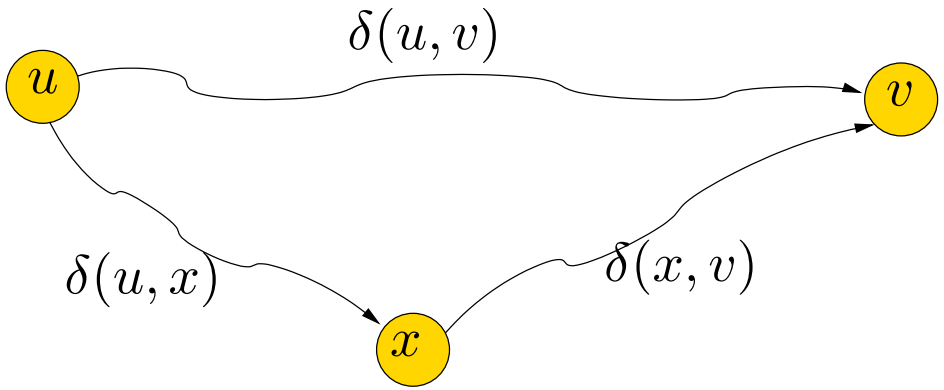
کوتاه‌ترین مسیرها: زیرمسئله‌ی بهینه

زیر مسیرهای یک مسیر بهینه خود مسیرهای بهینه هستند.

کوتاه‌ترین مسیرها: نامعادله‌ی مثلثی

اگر $\delta(u,v)$ طول کوتاه‌ترین مسیر از u به v باشد،

$$\delta(u,v) \leq \delta(u,x) + \delta(x,v)$$



کوتاه‌ترین مسیرها

آیا مسئله «خوش تعریف» است؟

کوتاه‌ترین مسیرها

آیا مسئله «خوش تعریف» است؟

بله، حتا با وزن‌های منفی، اگر دور منفی نداشته باشیم

الگوریتم بلمن فورد

همه‌ی کوتاه‌ترین مسیرها (APSP) از یک مبدأ s به بقیه‌ی رأس‌ها را پیدا می‌کند.

الگوریتم‌های مختلف برای این مسئله:

- برای هر $v \in V$ یک مقدار $d[v]$ که اندازه‌ی کوتاه‌ترین مسیر از s به v تا آن‌زمان است را نگه می‌دارد. این مقدار «روزآمد» می‌شود.
- خود مسیر را به‌راحتی می‌توان ساخت.

مقدار دهی اولیه برای کوتاه‌ترین مسیرها از یک مبداء

```
SINGLE-SOURCE-INITIALIZATION (G, s)
 1  for each v ∈ V
 2      do d[v] ← ∞
 3         π[v] ← null
 4  d[s] ← 0
```

مرحله‌ی تخفیف (Relax)

```
RELAX (u, v, w)
  1  if d[v] > d[u] + w(u, v)
  2    then d[v] ← d[u] + w(u, v)
  3         π[v] ← u
```

الگوریتم بلمن فورد

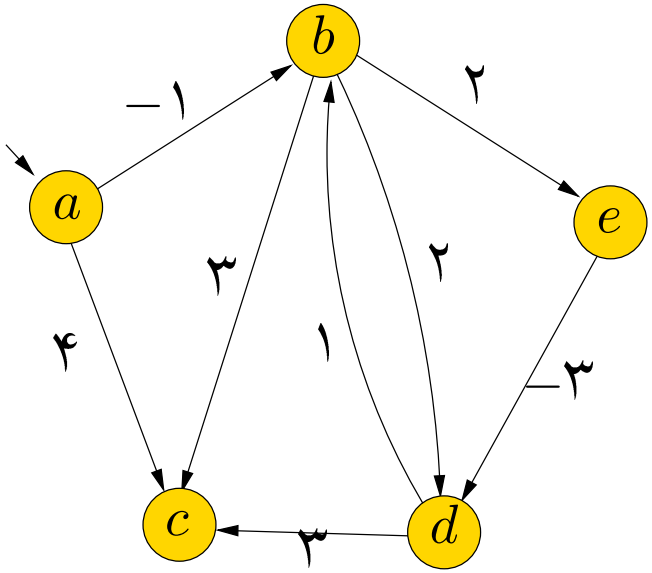
```
BELLMAN-FORD(G, w, s)
 1  SINGLE-SOURCE-INITIALIZATION(G, s)
 2  for i ← 1 to |V| − 1
 3      do for each edge (u, v) ∈ E
 4             do RELAX(u, v, w)
 5
 6  for each edge (u, v) ∈ E
 7      do if d[v] > d[u] + w(u, v)
 8            then No Solution!
```

الگوریتم بلمن فورد (ادامه)

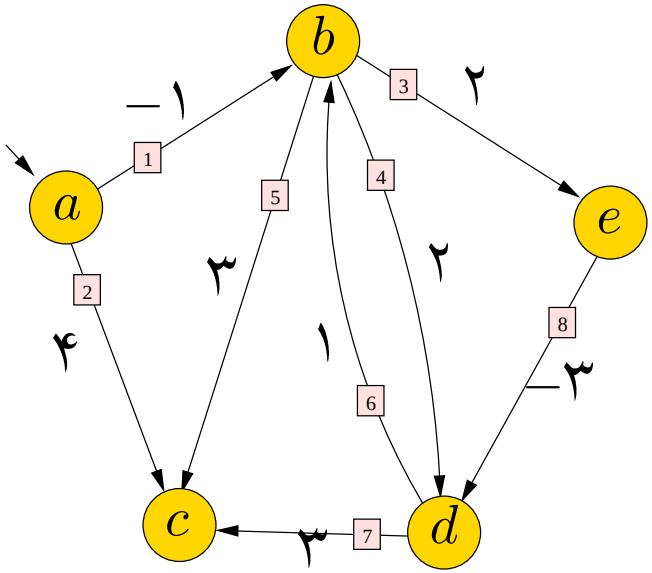
سه مرحله‌ی مهم

- **مقداردهی اولیه** (Initialization)
- **تخفیف مقدار** (Relaxation)
- **آزمون این‌که جواب دارد یا خیر.**

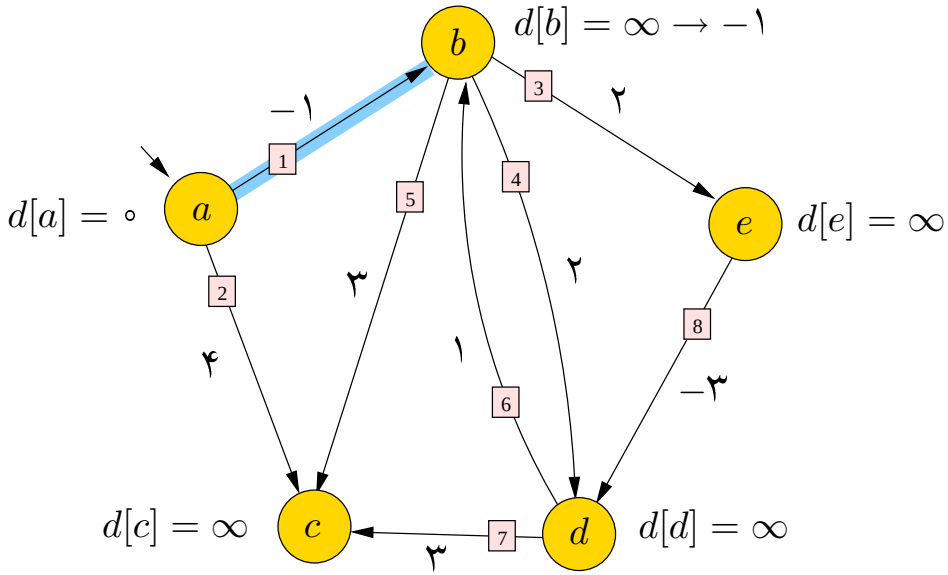
الگوریتم بلمن فورد (مثال)



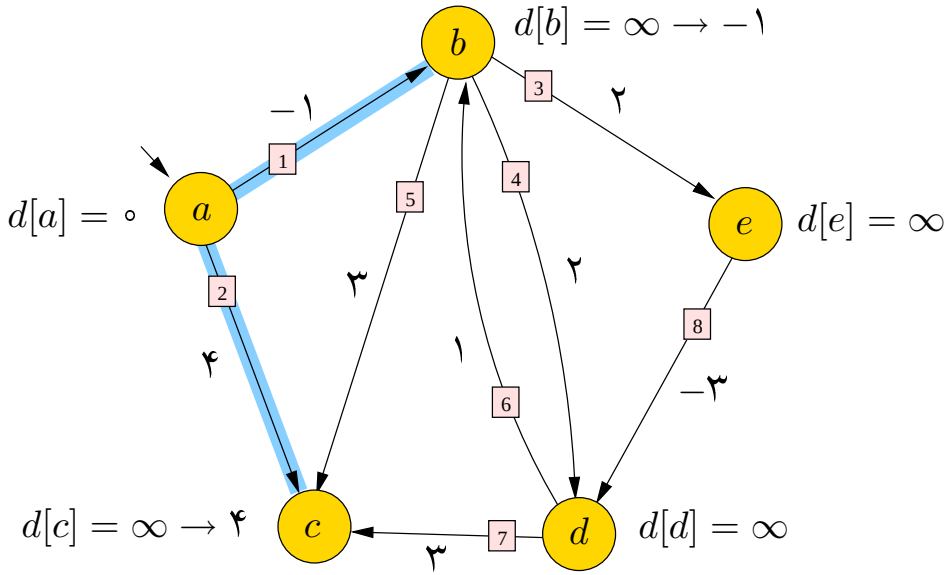
ترتیب یال‌ها



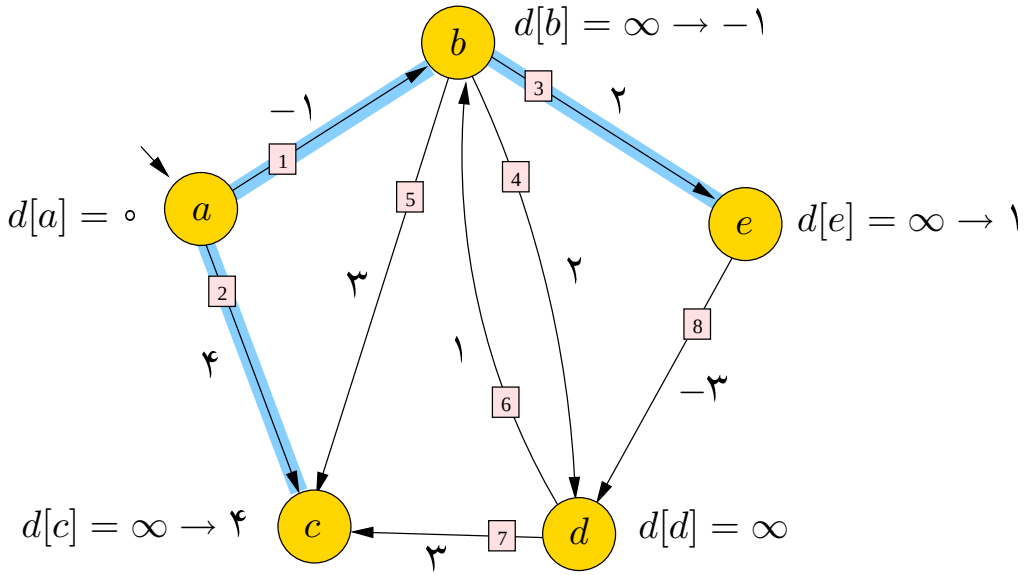
اجرای الگوریتم



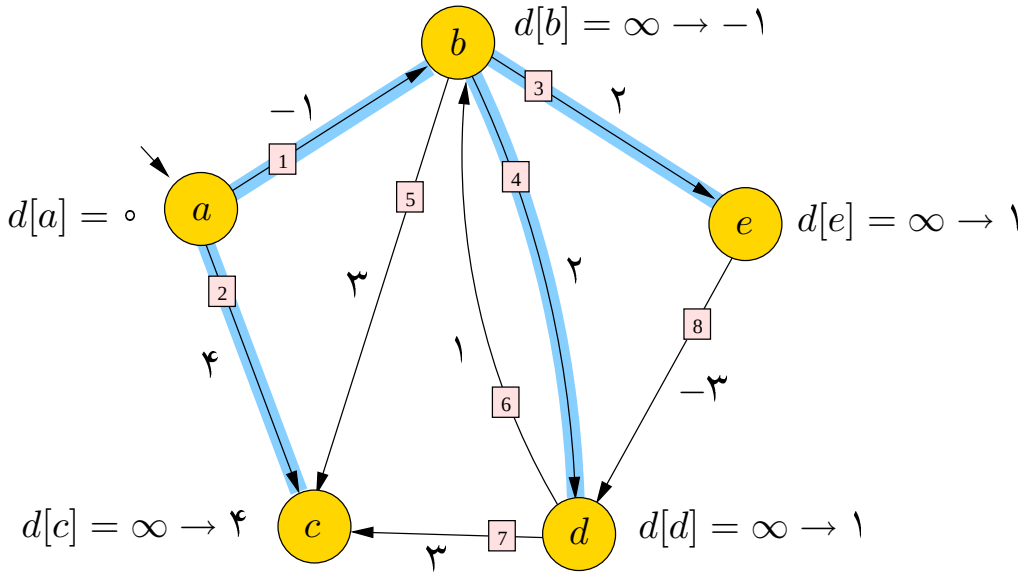
اجرای الگوریتم



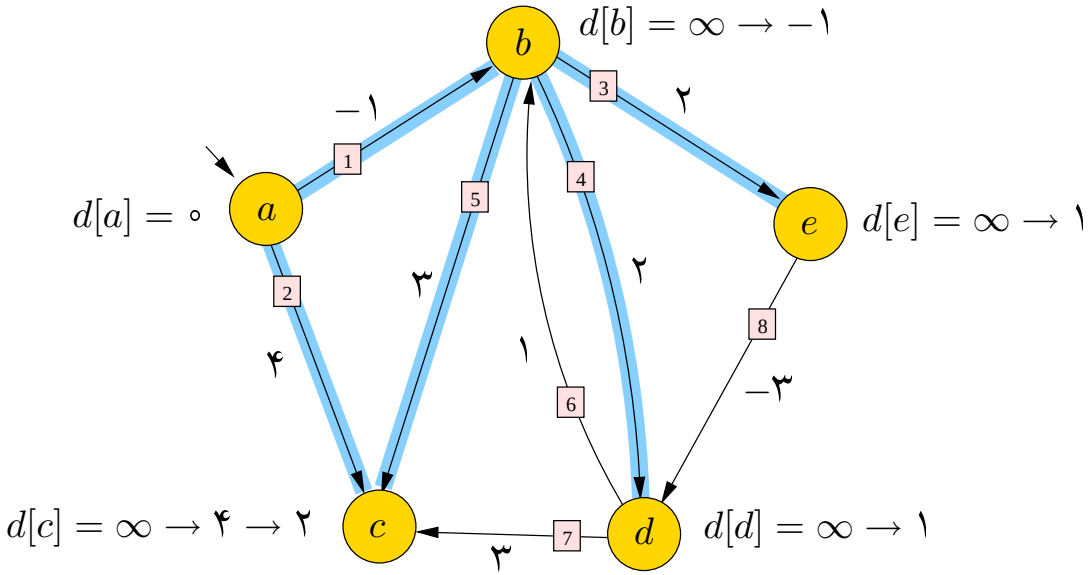
اجرای الگوریتم



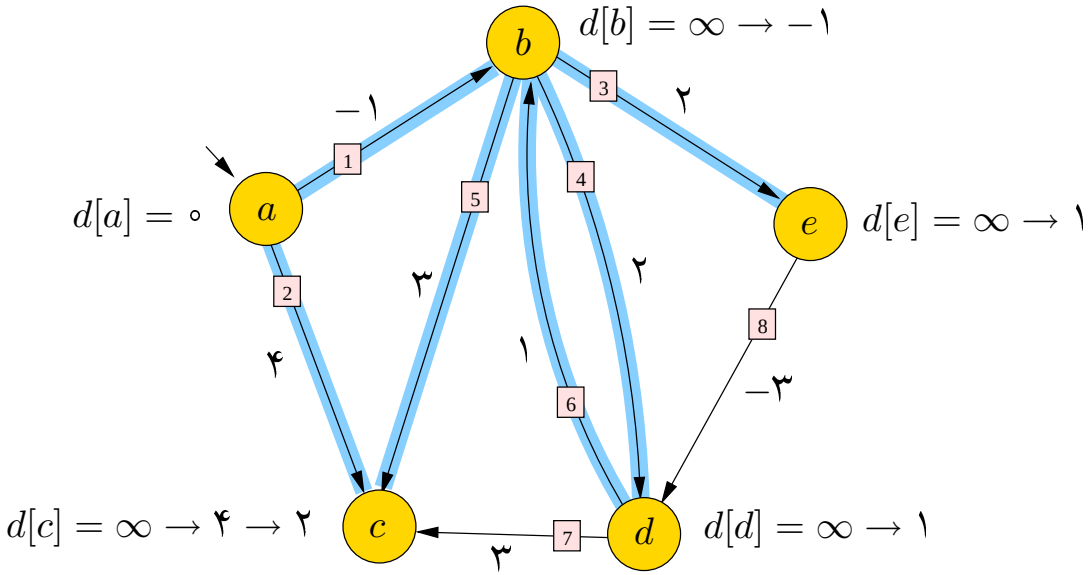
اجرای الگوریتم



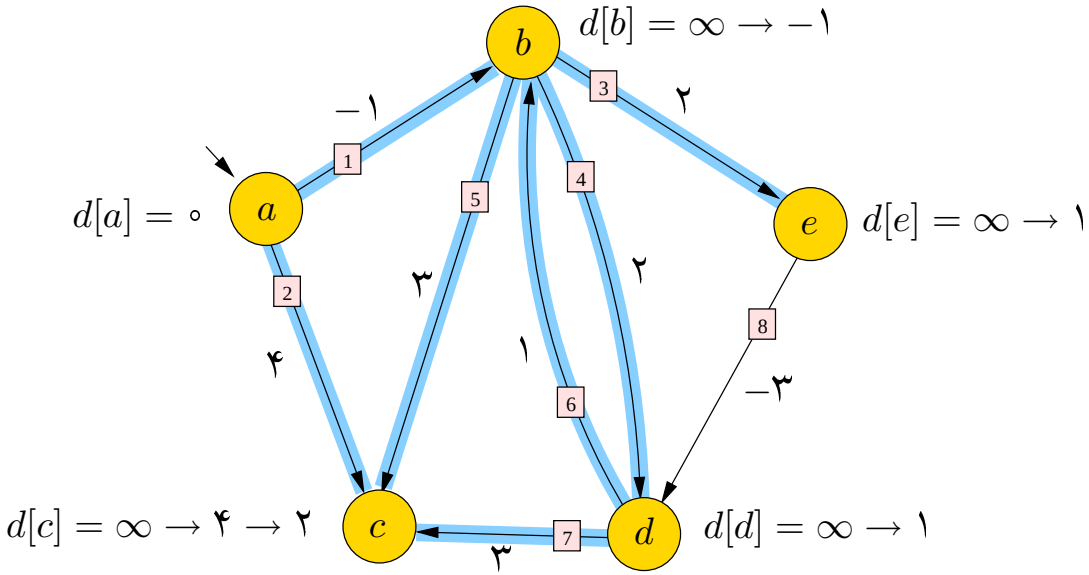
اجرای الگوریتم



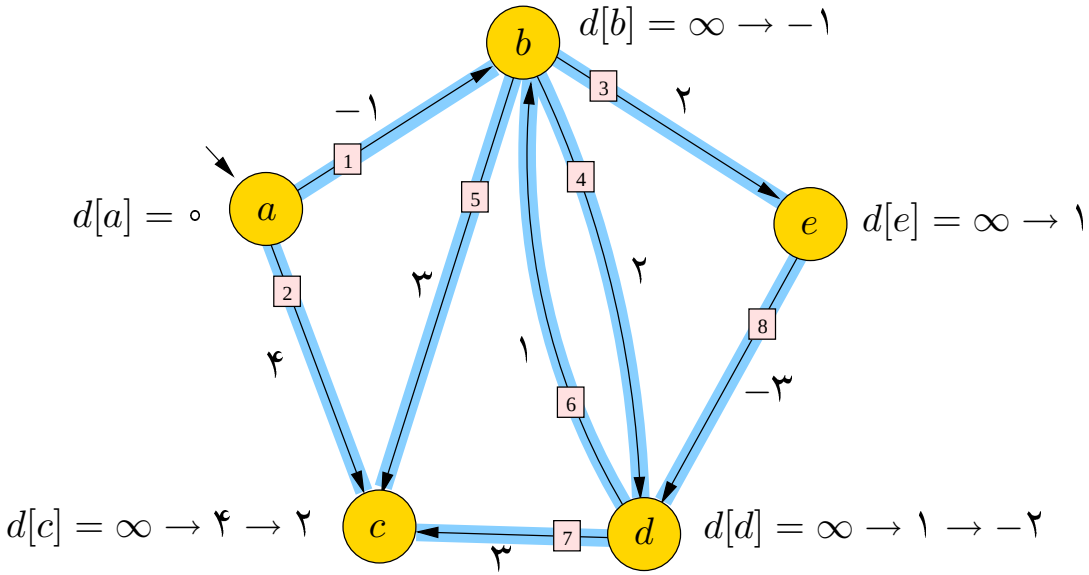
اجرای الگوریتم



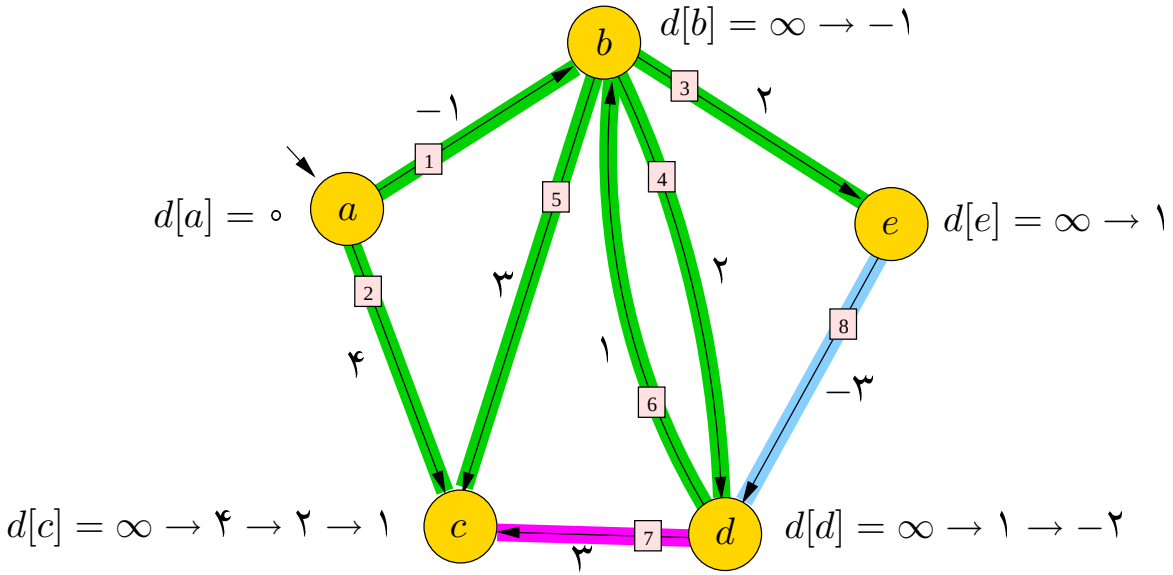
اجرای الگوریتم



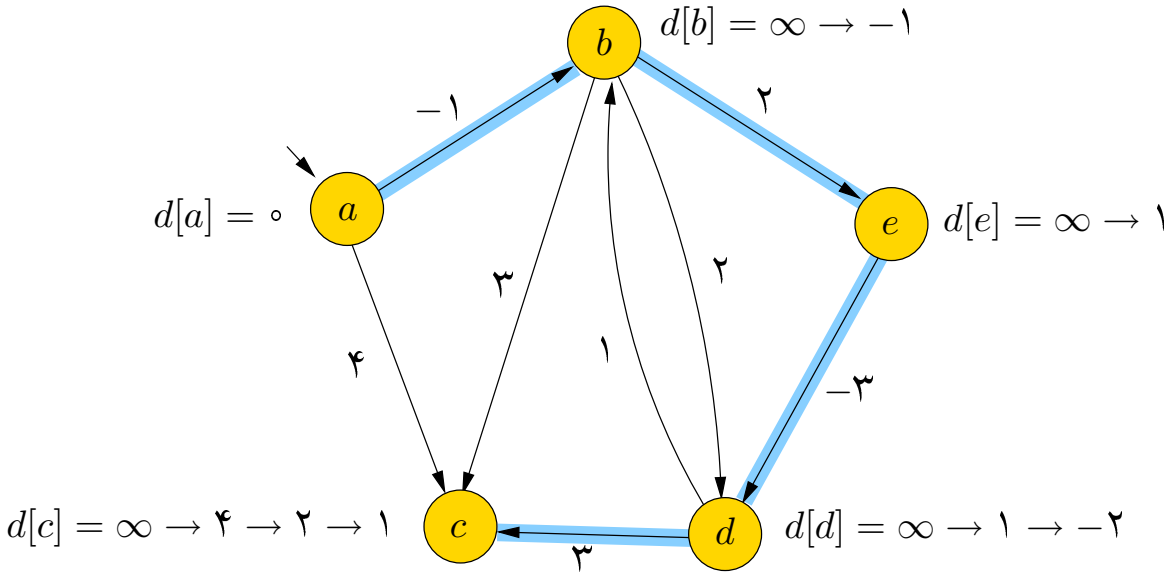
اجرای الگوریتم



اجرای الگوریتم



اجرای الگوریتم (پایان)



الگوریتم بلمن فورد (چرا درست است؟)

الگوریتم بلمن فورد (چرا درست است؟)

لم: همیشه $d[v] \geq \delta(s, v)$

اثبات:

- **در ابتدای الگوریتم این مطلب درست است.**
- **(برهان خلف) فرض کنید** v **اولین رأسی است که** $d[v] < \delta(s, v)$ **و** u **رأسی است که با فرمول** $d[v] = d[u] + w(u, v)$ **موجب این کار شده است.**

در آن‌صورت داریم

$$\begin{aligned} d[v] &< \delta(s,v) \\ &\leq \delta(s,u) + \delta(u,v) \qquad \text{نامعادله‌ی مثلثی} \\ &\leq \delta(s,u) + w(u,v) \\ &\leq d[u] + w(u,v) \end{aligned}$$

تناقض!

هنگامی‌که $d[v]$ برابر $\delta(s,v)$ می‌شود دیگر تغییر نمی‌کند؟ چرا؟

اثبات درستی الگوریتم بلمن فورد (ادامه)

لم: الگوریتم بلمن فورد درست است. یعنی پس از $|V| - ۱$ مرحله، مقدار d نهایی است.

اثبات:

اثبات درستی الگوریتم بلمن فورد (ادامه)

لم: الگوریتم بلمن فورد درست است. یعنی پس از $|V| - ۱$ مرحله، مقدار d نهایی است.

اثبات: دور منفی نداریم،

یک کوتاه‌ترین مسیر: $s \longrightarrow v_۱ \longrightarrow v_۲ \longrightarrow v_۳ \longrightarrow \ldots \longrightarrow v_{i-۱} \longrightarrow v_i \ldots \longrightarrow v_k$

با استقرا نشان می‌دهیم که در انتهای مرحله‌ی i ام $d[v_i]$ نهایی است.

پایه: $d[s]$.

اگر $d[v_{i-۱}]$ درست باشد، در انتهای مرحله‌ی i ام $d[v_i]$ درست خواهد بود.

چرا پس از $|V| - ۱$ مرحله مقدار d نهایی است؟

الگوریتم بلمن فورد: تحلیل

در $\Theta(VE)$

با این الگوریتم می‌توان وجود دور منفی در گراف را تشخیص داد.

کوتاه‌ترین مسیرها از یک مبداء در DAG

اگر گراف DAG باشد، می‌توان در $\Theta(E+V)$ این کار را انجام داد.

کوتاه‌ترین مسیرها از یک مبداء در DAG

اگر گراف DAG **باشد، می‌توان در** $\Theta(E+V)$ **این کار را انجام داد.**

DAG-SSSP (G, w, s)
1 do topological sort on G
2 SINGLE-SOURCE-INITIALIZATION(G, s)
3 **for** each $u \in V$ in the topological sort order
4 **do for** each $v \in Adj[u]$
5 **do** RELAX(u, v, w)

کوتاه‌ترین مسیرها از یک مبداء در DAG (چرا درست است؟)

استقرا بر روی ترتیب عناصر در مرتبه‌ی توپولوژیکی

طولانی‌ترین مسیرها از یک مبداء در DAG

مانند بلمن-فورد در $O(VE)$

طولانی‌ترین مسیرها از یک مبداء در DAG

مانند بلمن-فورد در $O(VE)$

می‌توان در $\Theta(E+V)$ این کار را انجام داد.

طولانی‌ترین مسیرها از یک مبداء در DAG

مانند بلمن-فورد در $O(VE)$

می‌توان در $\Theta(E+V)$ این کار را انجام داد.

DAG-SSSP (G, w, s)
1 **for** each $u \in V$ in the topological sort order
2 **do for** each $v \in Adj[u]$
3 **do** $longest[v] \leftarrow \max\{longest[v], longest[v] + w(u, v)\}$

ادگار دایکسترا

Edsger Wybe Dijkstra (May 11, 1930 - August 6, 2002)

الگوریتم دایکسترا (Dijkstra's Algorithm)

برای مسئله‌ی SSSP

- وزن یال‌ها غیر منفی است. (با این شرط از بلمن فورد بهتر است)
- مانند جست‌وجوی سطح‌اول است. (اگر وزن همه‌ی یال‌ها برابر باشد، البته BFS بهتر است)
- از یک صف اولویت برای ذخیره‌ی d ها استفاده می‌کند.

الگوریتم دایکسترا (کلیات)

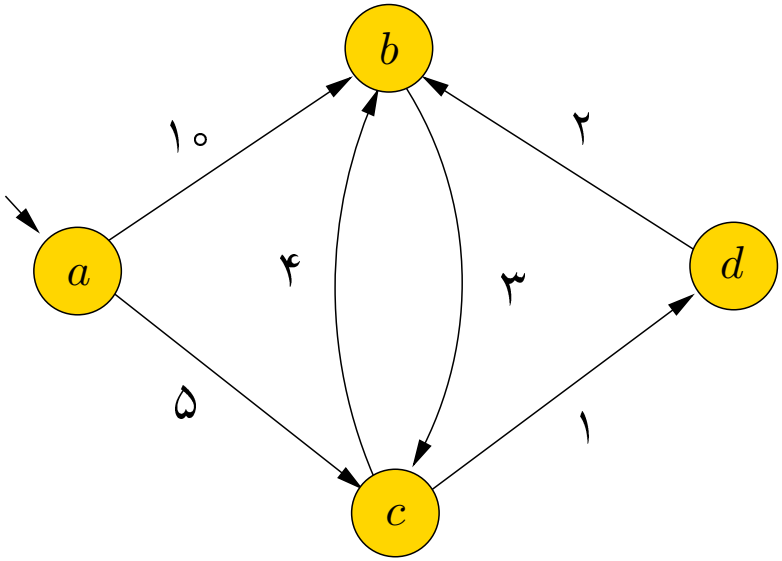
- یک مجموعه‌ی S (در ابتدا تهی) نگه می‌دارد که شامل رأس‌هایی است که کوتاه‌ترین مسیر از s به آن‌ها یافت و نهایی شده است.
- بقیه‌ی رأس‌ها با مقدار d ای که تا کنون دارند، در مجموعه‌ی Q هستند.
- هر بار رأس $u \in Q$ که کم‌ترین d را دارد انتخاب و به مجموعه‌ی S اضافه می‌کند. این انتخاب نهایی است.
- d رأس‌های موجود در Q که مجاور u هستند را روزآمد می‌کند (یال‌های متصل به u relax می‌شوند)

الگویتم حریصانه است.

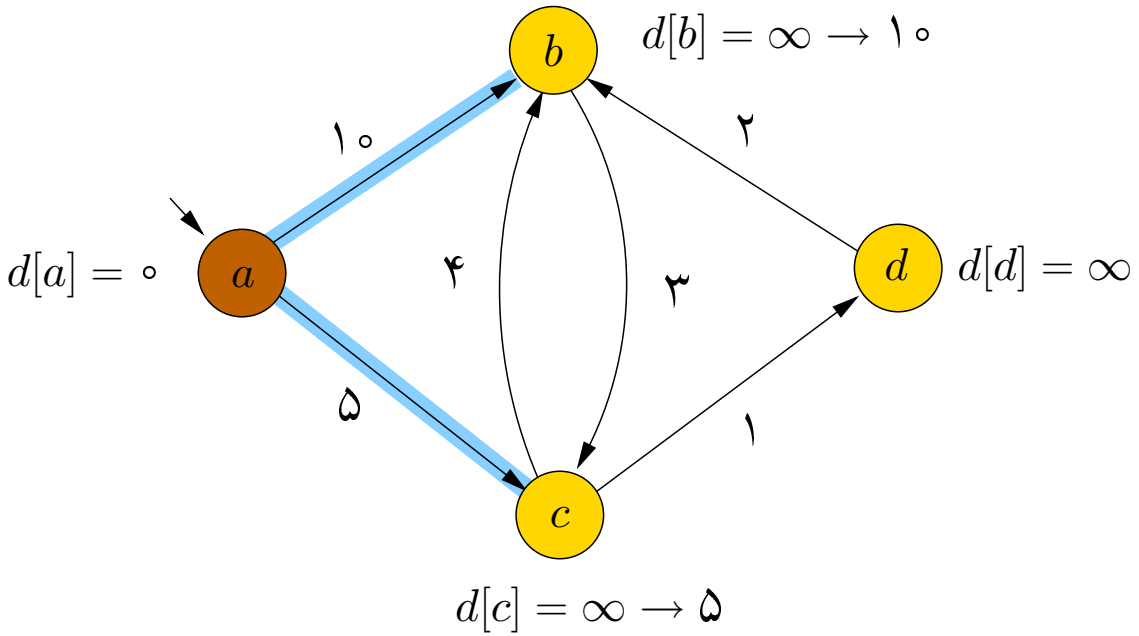
الگوریتم دایکسترا

```
DIJKSTRA (G, w, s)
 1  SINGLE-SOURCE-INITIALIZATION(G, s)
 2  S ← φ
 3  Q ← V
 4  while Q ≠ φ
 5        do u ← EXTRACT-MIN(Q)
 6           S ← S ∪ {u}
 7           for each v ∈ Adj[u]
 8              do RELAX(u, v, w)
```

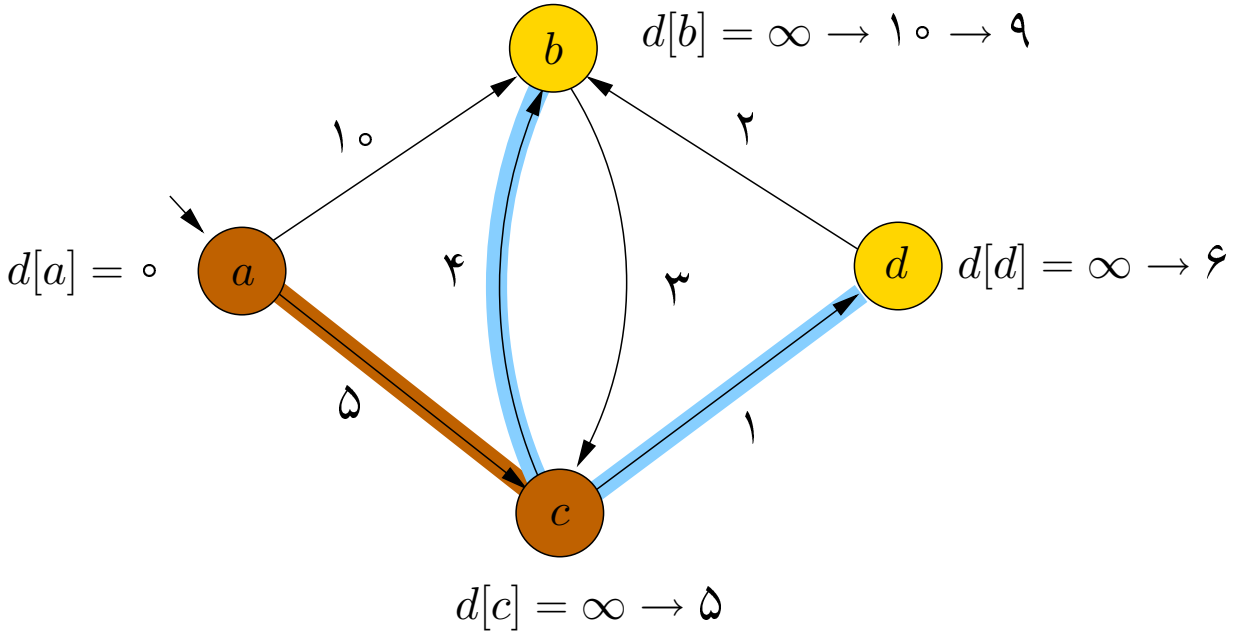
الگوریتم دایکسترا: مثال



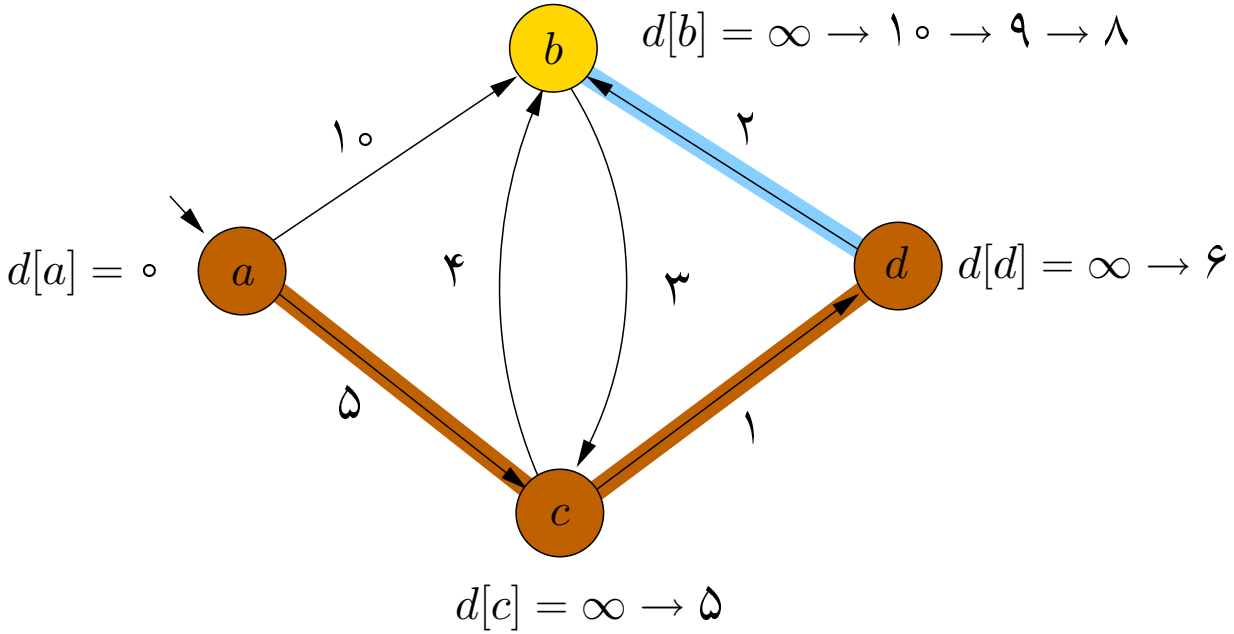
اجرای الگوریتم دایکسترا



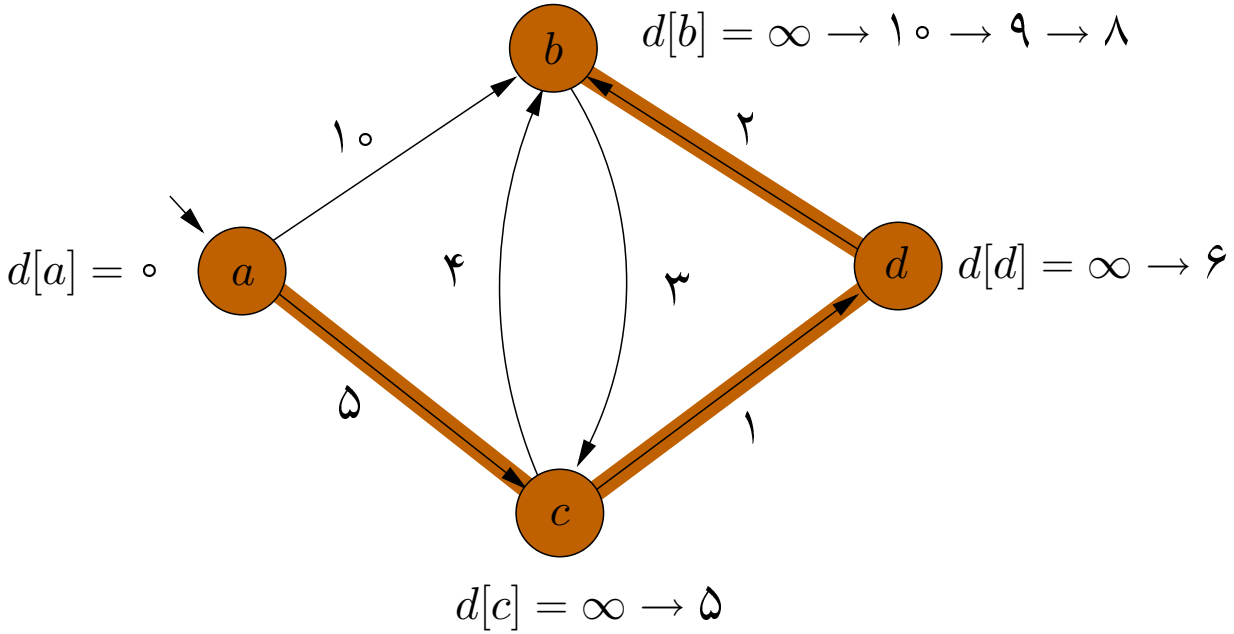
اجرای الگوریتم دایکسترا



اجرای الگوریتم دایکسترا



اجرای الگوریتم دایکسترا



الگوریتم دایکسترا: تحلیل

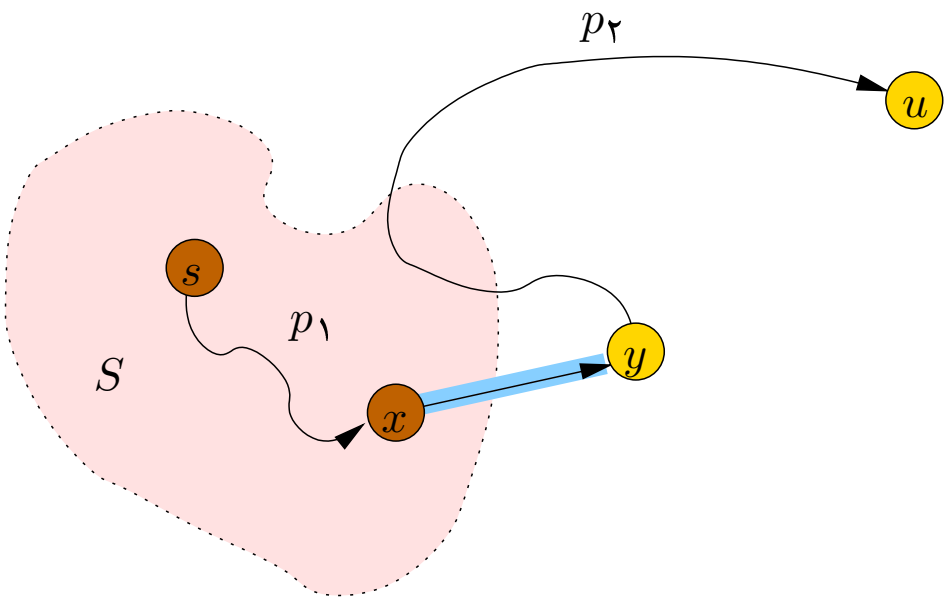
$|V|$ بار Extract-Min

$|E|$ بار Decrease-Key

Q	T(Extract-Min)	T(Decrease-Key)	Dijkstra time
array	$\Theta(V)$	$\Theta(۱)$	$\Theta(V^۲)$
binary heap	$\Theta(\lg V)$	$\Theta(\lg V)$	$\Theta((V+E)\lg V)$
Fibonacci heap	$\Theta(\lg V)$	$\Theta(۱)$	$\Theta(V\lg V+E)$

الگوریتم دایکسترا: اثبات درستی

ادعا: اگر u به S اضافه شود، $d[u] = \delta(s, u)$



الگوریتم دایکسترا: اثبات درستی (ادامه)

می‌دانیم که همواره $d[u] \geq \delta(s,u)$

فرض: u اولین رأسی است که وارد S می‌شود و $d[u] > \delta(s,u)$

فرض کنید p کوتاه‌ترین مسیر واقعی به u است.

فرض کنید y اولین رأس روی p است که در S نیست.

الگوریتم دایکسترا: اثبات درستی (ادامه)

بدیهی است که $\delta(s,u) \geq \delta(s,y) = d[y]$ **چون وزن منفی نداریم.**

پس $d[u] > \delta(s,u) \geq d[y]$

پس باید y **انتخاب می‌شد که وارد** S **شود نه** u**. تناقض!**