یافتن کوتاه‌ترین مسیرها

سید ناصر رضوی www.snrazavi.ir
۱۳۹۵

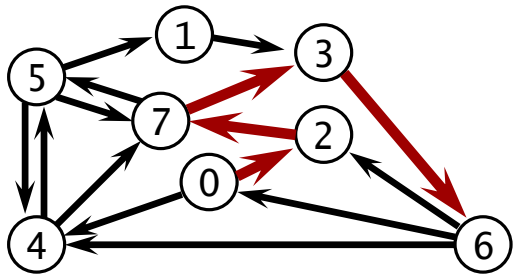
فهرست مطالب

کوتاه‌ترین مسیرها در گراف‌های جهت‌دار و وزن‌دار

- مسئله. با داشتن یک گراف جهت‌دار وزن‌دار، کوتاه‌ترین مسیر بین دو رأس **s** و **t** را پیدا کن.

گراف جهت‌دار وزن‌دار

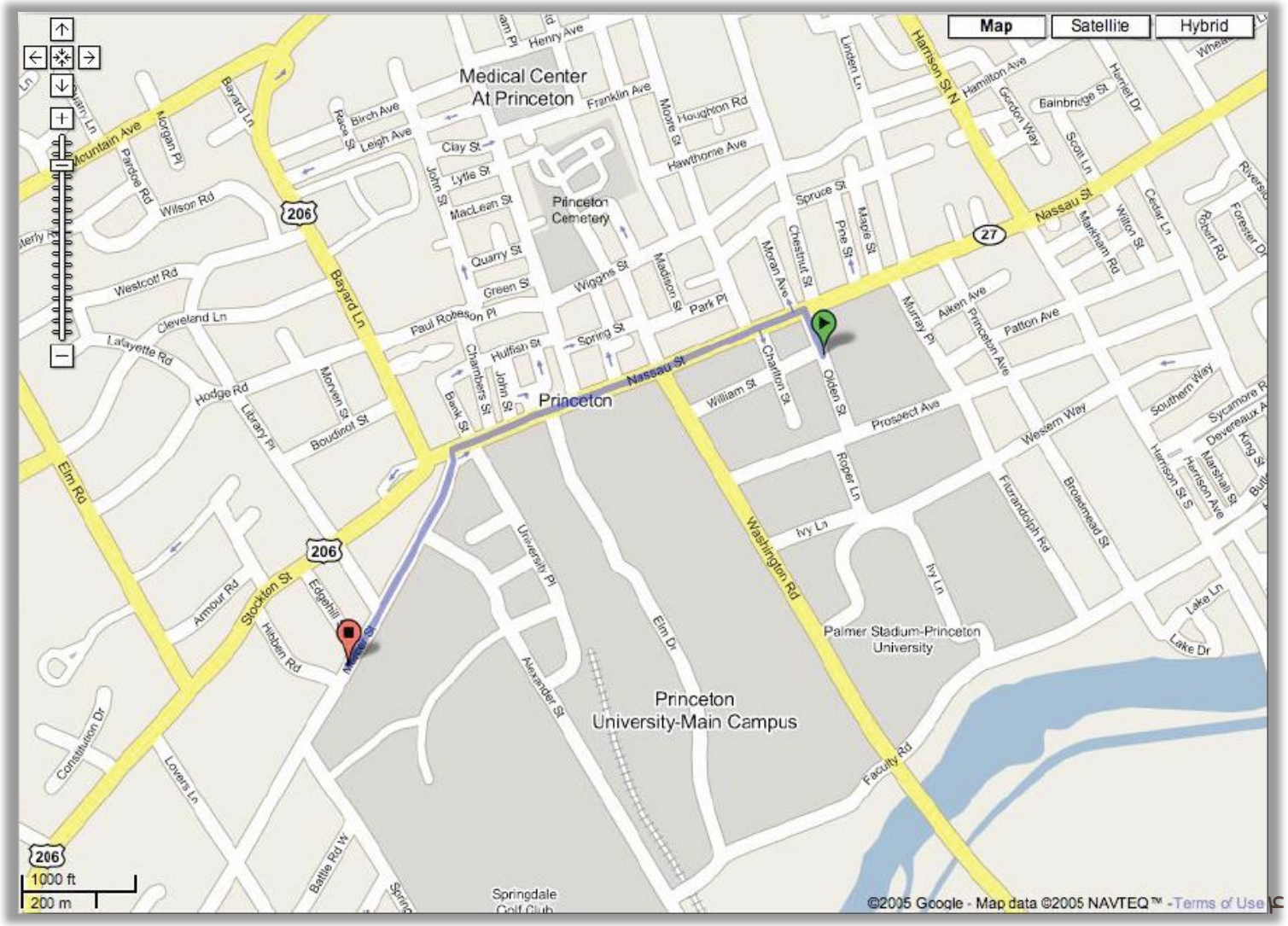
```
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```



کوتاه‌ترین مسیر از 0 به 6

```
0->2  0.26
2->7  0.34
7->3  0.39
3->6  0.52
```

نقشه‌های گوگل



ناوبری خودرو: سامانه‌ی مکان‌یابی جهانی

کاربردهای کوتاه‌ترین مسیر

http://en.wikipedia.org/wiki/Seam_carving

- ناوبری خودرو
- ناوبری روبات
- روش مسیر بحرانی
- مسیریابی در نقشه
- برنامه‌ریزی ترافیک شهری
- مسیریابی پیام‌های مخابراتی
- پروتکل‌های مسیریابی شبکه
- تغییر هوشمندانه‌ی اندازه تصویر
- سودجویی در معاملات ارزی
- و بسیاری از مسائل دیگر ...

گونه‌های مختلف مسئله‌ی کوتاه‌ترین مسیر

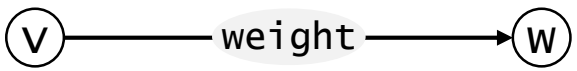
- کدام رئوس؟
  - تک مبدأ-تک مقصد: کوتاه‌ترین مسیر از یک رأس به یک رأس دیگر
  - تک مبدأ: کوتاه‌ترین مسیر از یک رأس به تمام رئوس دیگر
  - همه زوج رئوس: کوتاه‌ترین مسیر بین همه زوج رئوس
- محدودیت بر روی وزن یالها؟
  - وزن های غیر منفی
  - وزن های دلخواه
  - وزن های اقلیدسی
- دور؟
  - بدون «دور جهت‌دار»
  - بدون «دور منفی»

واسطها

واسط یال جهت‌دار وزن‌دار

- واسط کلاس یال جهت‌دار وزن‌دار.

```
public class DirectedEdge
                DirectedEdge(int v, int w, double weight)     ایجاد یال جهت‌دار v -> w
           int  from()                                        برگرداندن رأس ابتدایی v
           int  to(int v)                                     برگرداندن رأس انتهایی w
        double  weight()                                      برگرداندن وزن یال
        String  toString()                                    نمایش یال به صورت رشته ای
```



یال جهت‌دار وزن‌دار: پیاده‌سازی جاوا


```
public class DirectedEdge
{
    private final int v, w;
    private final double weight;

    public DirectedEdge(int v, int w, double weight)
    {
       this.v = v;
       this.w = w;
       this.weight = weight;
    }

    public int from()
    {  return v;  }

    public int to()
    {  return w;  }


    public double weight()
    {  return weight;  }

}
```

سازنده

برگرداندن رأس ابتدایی

برگرداندن رأس انتهایی

برگرداندن وزن یال

واسط گراف جهت‌دار وزن‌دار

public class	EdgeWeightedDigraph	
	EdgeWeightedDigraph(int V)	ایجاد یک گراف تهی با **V** رأس
	EdgeWeightedDigraph(In in)	ایجاد یک گراف از جریان ورودی
void	addEdge(DirectedEdge e)	افزودن یال **e** به گراف
Iterable<DirectedEdge>	adj(int v)	برگرداندن یالهای خروجی از رأس **v**
Iterable<DirectedEdge>	edges()	برگرداندن همه یالهای گراف
int	V()	برگرداندن تعداد رئوس
int	E()	برگرداندن تعداد یالها
String	toString()	نمایش گراف وزن‌دار به صورت یک رشته

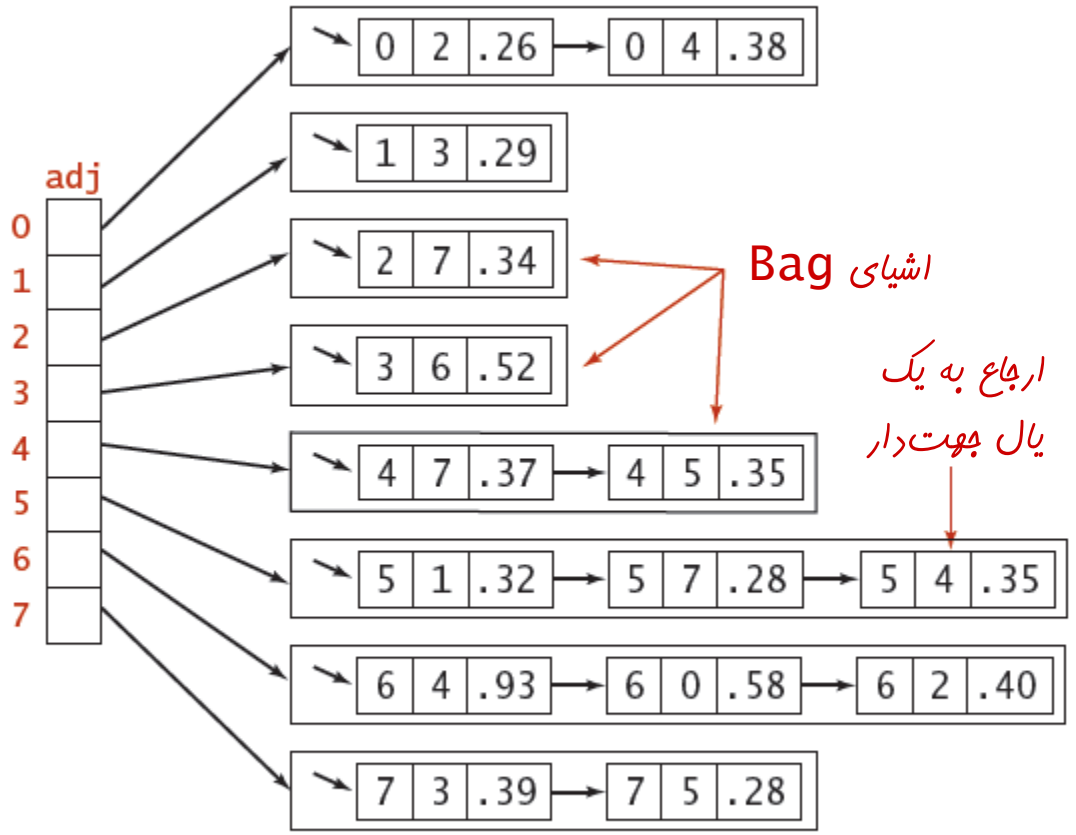
قرارداده‌ها. در این پیاده‌سازی داشتن طوقه و یالهای موازی مجاز است.

گراف جهت‌دار وزن‌دار: نمایش لیست همسایگی

tinyEWD.txt

V → 8
E → 15

```
4 5  0.35
5 4  0.35
4 7  0.37
5 7  0.28
7 5  0.28
5 1  0.32
0 4  0.38
0 2  0.26
7 3  0.39
1 3  0.29
2 7  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```



گراف جهت‌دار وزن‌دار: پیاده‌سازی جاوا


```
public class EdgeWeightedDigraph
{
    private final int V;
    private final Bag<DirectedEdge>[] adj;

    public EdgeWeightedDigraph(int V)
    {
       this.V = V;
       adj = (Bag<DirectedEdge>[]) new Bag[V];
       for (int v = 0; v < V; v++)
           adj[v] = new Bag<DirectedEdge>();
   }

    public void addEdge(DirectedEdge e)
    {
       int v = e.from();
       adj[v].add(e);
 }

    public Iterable<DirectedEdge> adj(int v)
    {  return adj[v];  }

}
```

سازنده

اضافه کردن یال e تنها به لیست همسایگی v

واسط کلاس کوتاه‌ترین مسیرهای تک مبدأ

- هدف. یافتن کوتاه‌ترین مسیرها از رأس مبدأ **s** به تمام رئوس دیگر.

public class	SP	
	SP(EdgeWeightedDigraph G, int s)	محاسبه کوتاه‌ترین مسیرها از **s**
double	distTo(int v)	طول کوتاه‌ترین مسیر از **s** به **v**
Iterable<DirectedEdge>	pathTo(int v)	کوتاه‌ترین مسیر از **s** به **v**
boolean	hasPathTo(int v)	آیا از **s** به **v** مسیر وجود دارد؟

```
SP sp = new SP(G, s);
for (int v = 0; v < G.V(); v++)
{
    StdOut.printf("%d to %d (%.2f): ", s, v, sp.distTo(v));
    for (DirectedEdge e : sp.pathTo(v))
        StdOut.println(e + "   ");
   StdOut.println();
}
```

واسط کلاس کوتاه‌ترین مسیرهای تک مبدأ

- هدف. یافتن کوتاه‌ترین مسیرها از رأس مبدأ **s** به تمام رئوس دیگر.

public class	SP	
	SP(EdgeWeightedDigraph G, int s)	محاسبه کوتاه‌ترین مسیرها از **s**
double	distTo(int v)	طول کوتاه‌ترین مسیر از **s** به **v**
Iterable<DirectedEdge>	pathTo(int v)	کوتاه‌ترین مسیر از **s** به **v**
boolean	hasPathTo(int v)	آیا از **s** به **v** مسیر وجود دارد؟

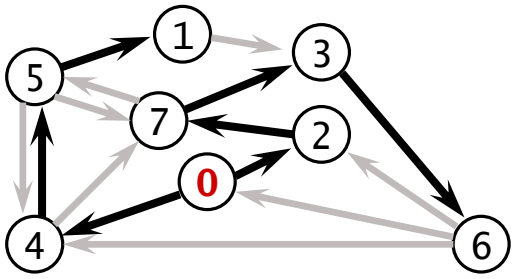
```
% java SP tinyEWD.txt 0
0 to 0 (0.00):
0 to 1 (1.05): 0->4 0.38  4->5 0.35  5->1 0.32
0 to 2 (0.26): 0->2 0.26
0 to 3 (0.99): 0->2 0.26  2->7 0.34  7->3 0.39
0 to 4 (0.38): 0->4 0.38
0 to 5 (0.73): 0->4 0.38  4->5 0.35
0 to 6 (1.51): 0->2 0.26  2->7 0.34  7->3 0.39  3->6 0.52
0 to 7 (0.60): 0->2 0.26  2->7 0.34
```

ویژگی‌های کوتاه‌ترین مسیر

ساختمان داده برای مسئله‌ی کوتاه‌ترین مسیر تک مبدأ

- هدف. یافتن کوتاه‌ترین مسیر از رأس s به تمام رئوس دیگر.
- مشاهده. یک درخت کوتاه‌ترین مسیر (SPT) وجود دارد. چرا؟
- نتیجه. درخت کوتاه‌ترین مسیر را با استفاده از دو آرایه می‌توان نمایش داد:
  - `distTo[v]` = طول کوتاه‌ترین مسیر از s به v.
  - `edgeTo[v]` = آخرین یال در کوتاه‌ترین مسیر از s به v.

v	distTo[]	edgeTo[]
0	0.00	null
1	1.05	5->1
2	0.26	0->2
3	0.97	7->3
5	0.38	0->4
4	0.73	4->5
6	1.49	3->6
7	0.60	2->7



درخت کوتاه‌ترین مسیر از s

ساختمان داده ها برای مسئله کوتاه‌ترین مسیر تک مبدأ

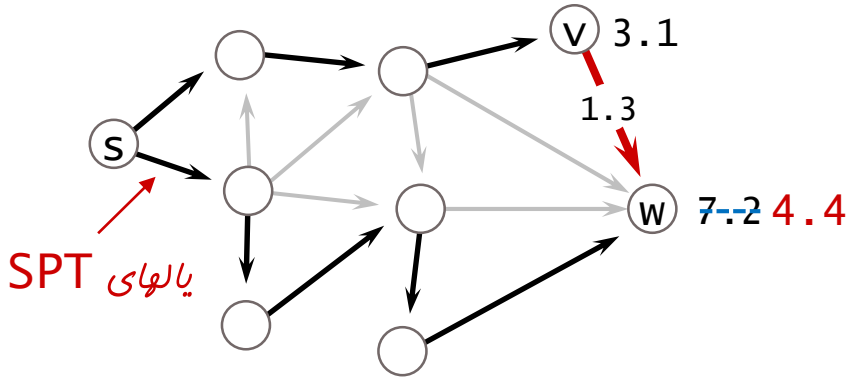
- هدف. یافتن کوتاه‌ترین مسیر از رأس s به تمام رئوس دیگر.
- مشاهده. یک درخت کوتاه‌ترین مسیر (SPT) وجود دارد. چرا؟
- نتیجه. درخت کوتاه‌ترین مسیر را با استفاده از دو آرایه می‌توان نمایش داد:
  - `distTo[v]` = طول کوتاه‌ترین مسیر از s به v.
  - `edgeTo[v]` = آخرین یال در کوتاه‌ترین مسیر از s به v.

```
public double distTo(int v)
{   return distTo[v];   }

public Iterable<DirectedEdge> pathTo(int v)
{
    if (!hasPathTo(v)) return null;
    Stack<DirectedEdge> path = new Stack<DirectedEdge>();
    for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()])
        path.push(e);
    return path;
}
```

راحت‌سازی یال (relaxation)

- راحت‌سازی یال e = v->w.
  - distTo[v] برابر است با طول کوتاه‌ترین مسیر فعلی از s به v
  - distTo[w] برابر است با طول کوتاه‌ترین مسیر فعلی از s به w
  - edgeTo[w] برابر است با آخرین یال در کوتاه‌ترین مسیر فعلی از s به w
  - اگر یال v -> w مسیر کوتاه‌تری به رأس w ایجاد می‌کند،
    هر دو مقدار distTo[w] و edgeTo[w] را به روز رسانی کن.

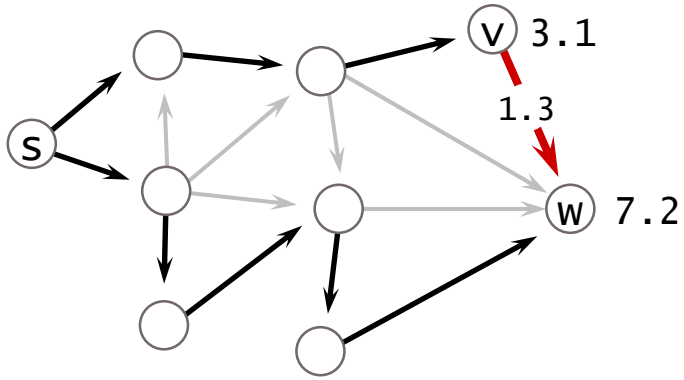


راحت‌سازی موفقیت آمیز یال v->w

راحت‌سازی یال (relaxation)

- راحت‌سازی یال e = v->w.
  - distTo[v] برابر است با طول کوتاه‌ترین مسیر فعلی از s به v
  - distTo[w] برابر است با طول کوتاه‌ترین مسیر فعلی از s به w
  - edgeTo[w] برابر است با آخرین یال در کوتاه‌ترین مسیر فعلی از s به w
  - اگر یال v -> w مسیر کوتاه‌تری به رأس w ایجاد می‌کند،

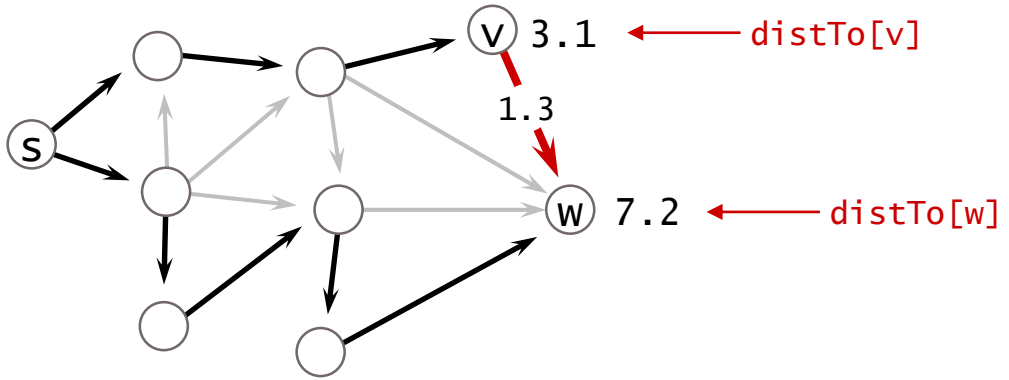
    هر دو مقدار distTo[w] و edgeTo[w] را به روز رسانی کن.




```
private void relax(DirectedEdge e) {
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight())
    {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
    }
}
```

شرط بهینگی کوتاه‌ترین مسیر

- گزاره. فرض کنید G یک گراف جهت‌دار وزن‌دار باشد.
- در این صورت مقادیر distTo[] بیانگر طول کوتاه‌ترین مسیرها از s هستند اگر و فقط اگر:
  - به ازای هر رأس v، مقدار distTo[v] بیانگر طول یک مسیر از s به v باشد.
  - به ازای هر یال e = v->w، داشته باشیم: distTo[w] <= distTo[v] + e.weight()
- اثبات. [شرط لازم]
  - فرض کنید به ازای یال e = v->w داشته باشیم distTo[w] > distTo[v] + e.weight().
  - در این صورت یال e مسیری از s به w ایجاد می‌کند که طول آن کوتاه‌تر از distTo[w] است.



شرط بهینگی کوتاه‌ترین مسیر

- گزاره. فرض کنید G یک گراف جهت‌دار وزن‌دار باشد.

  در این صورت مقادیر distTo[] بیانگر طول کوتاه‌ترین مسیرها از s هستند اگر و فقط اگر:
  - به ازای هر رأس v، مقدار distTo[v] بیانگر طول یک مسیر از s به v باشد.
  - به ازای هر یال e = v->w، داشته باشیم: distTo[w] <= distTo[v] + e.weight()
- اثبات. [شرط کافی]
  - فرض کنید $s = v_0 \to v_1 \to \ldots \to v_k = w$ کوتاه‌ترین مسیر از s به w باشد. آنگاه:

$$distTo[v_k] <= distTo[v_{k-1}] + e_k.weight()$$
$$distTo[v_{k-1}] <= distTo[v_{k-2}] + e_{k-1}.weight()$$
$$\ldots$$
$$distTo[v_1] <= distTo[v_0] + e_1.weight()$$

در نتیجه

$$distTo[w] = distTo[v_k] <= e_1.weight() + e_1.weight() + \ldots + e_k.weight()$$

بنابراین distTo[w] بیانگر طول کوتاه‌ترین از s به w است.

الگوریتم عمومی کوتاه‌ترین مسیر

Generic Algorithm (to compute SPT from s)

Initialize `distTo[s] = 0` and `distTo[v] = ∞` for all other vertices.

Repeat `until optimality conditions are satisfied`:
- Relax any edge

- گزاره. الگوریتم عمومی درخت کوتاه‌ترین مسیر را (در صورت وجود) محاسبه می‌کند.
- طرح اثبات.
  - در طول اجرا، `distTo[v]` طول یک مسیر ساده از `s` به `v` است
  - هر راحت‌سازی موفق، مقدار `distTo[v]` را کاهش می‌دهد.
  - مقدار `distTo[v]` حداکثر می‌تواند تعداد دفعات محدودی کاهش یابد.

الگوریتم عمومی کوتاه‌ترین مسیر

Generic Algorithm (to compute SPT from s)

Initialize `distTo[s] = 0` and `distTo[v] = ∞` for all other vertices.

Repeat `until optimality conditions are satisfied`:
- Relax any edge

- پیاده سازی کارا. کدام یال باید `relax` شود؟
- مثال ۱. الگوریتم دایکسترا (وزن یالها غیرمنفی)
- مثال ۲. الگوریتم مرتب‌سازی توپولوژیکی (بدون دورهای جهت‌دار)
- مثال ۳. الگوریتم بلمن - فورد (بدون دور منفی)

الگوریتم دایکسترا

چند نقل قول از دایکسترا

« فقط کاری را انجام بده که توان انجامش را داری. »

« کامپیوترها به عنوان یک ابزار چیزی بیش از یک موج گذرا بر سطح فرهنگ ما خواهند بود. کامپیوترها در ظرفیتی که برای چالش‌های ذهنی ایجاد می‌کنند، در تاریخ فرهنگی بشر بی‌سابقه هستند.»

« استفاده از کوبول مغز را تباه می‌کند؛ بنابراین تدریس آن باید به عنوان نوعی قانون شکنی تلقی شود. »

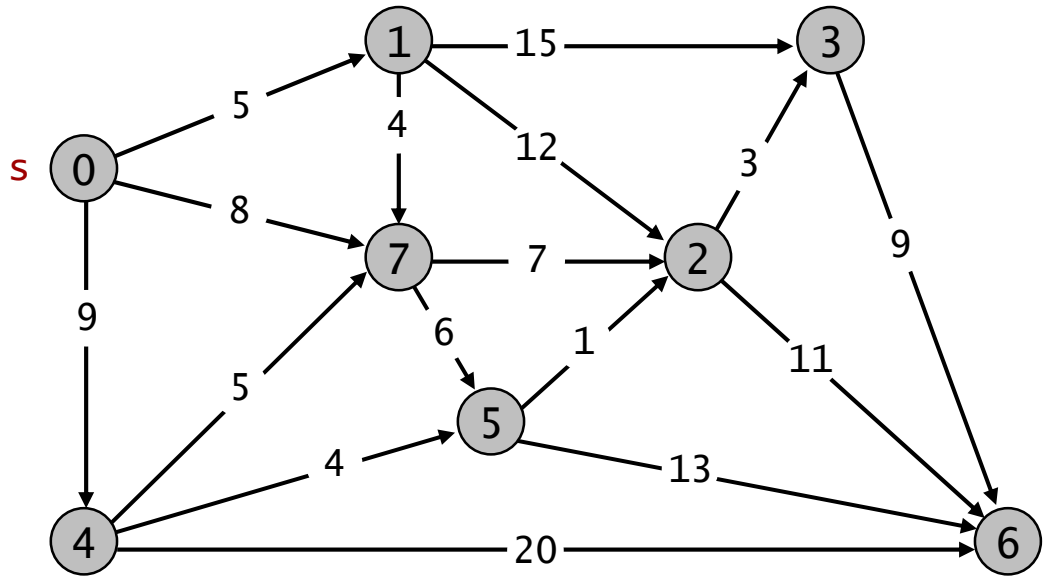
« تدریس برنامه‌نویسی خوب به دانشجویانی که قبلاً در معرض بیسیک بوده‌اند در عمل غیرممکن است: به عنوان برنامه‌نویسان بالقوه ذهن آنها چنان فلج شده که دیگر امیدی به بازسازی آن نیست. »

« ای پی ال اشتباهی است که به کمال خود رسیده است. ای پی ال زبان برنامه‌نویسی آینده با استفاده از روش‌های برنامه‌نویسی گذشته است. این زبان نسل تازه‌ای از خطاهای برنامه‌نویسی را به راه خواهد انداخت. »

ادسخر دایکسترا
جایزه تورینگ ۱۹۷۲

الگوریتم دایکسترا: اجرای نمایشی

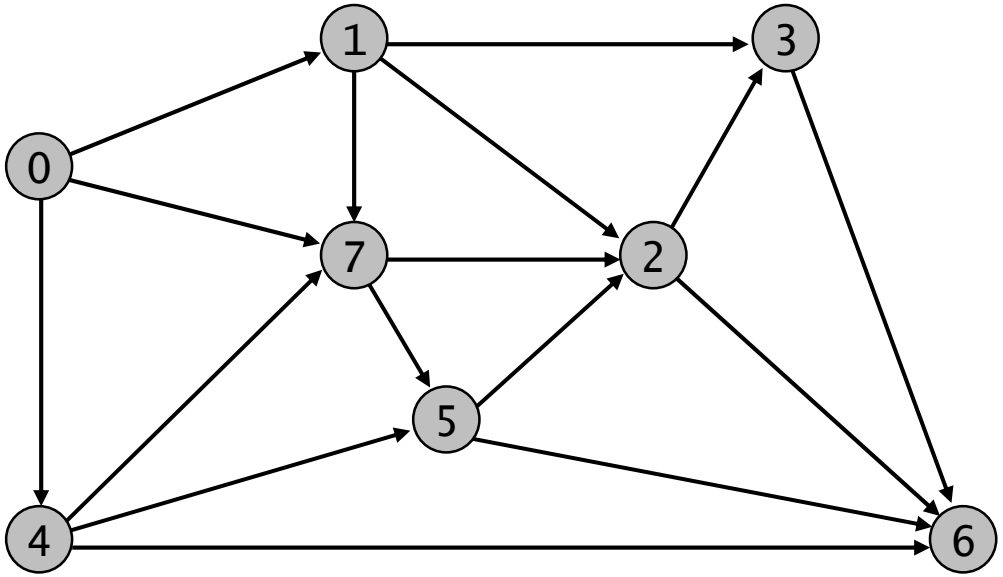
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.




```
0->1   5.0
0->4   9.0
0->7   8.0
1->2  12.0
1->3  15.0
1->7   4.0
2->3   3.0
2->6  11.0
3->6   9.0
4->5   4.0
4->6  20.0
4->7   5.0
5->2   1.0
5->6  13.0
7->5   6.0
7->2   7.0
```

الگوریتم دایکسترا: اجرای نمایشی

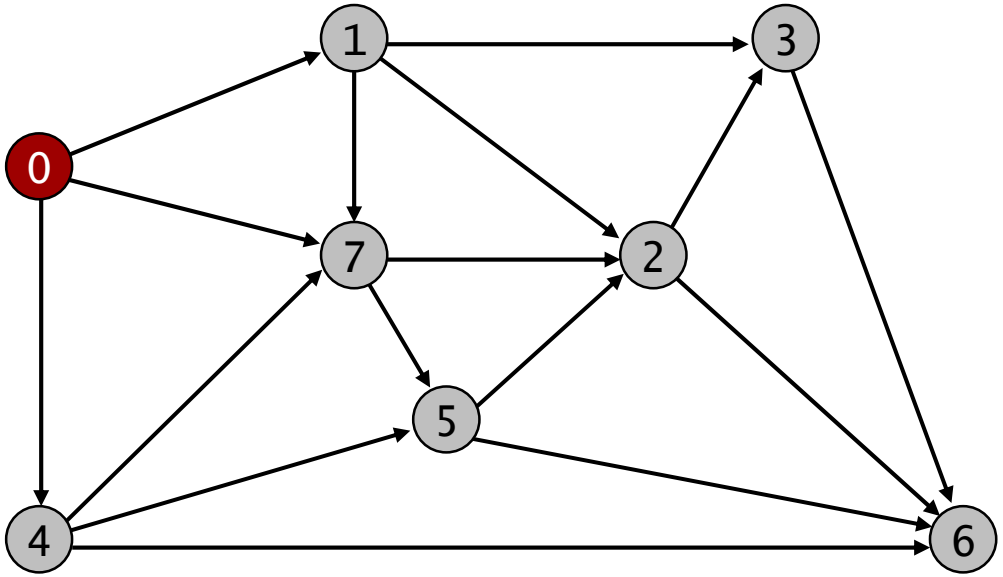
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
5		
4		
6		
7		

الگوریتم دایکسترا: اجرای نمایشی

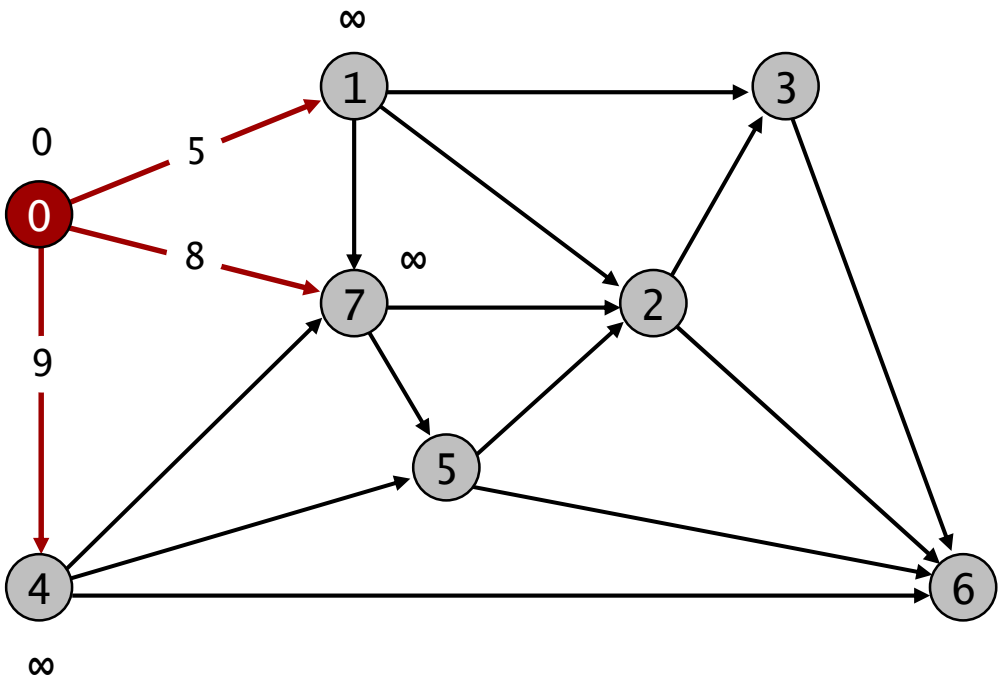
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
→ 0	0.0	-
1		
2		
3		
5		
4		
6		
7		

الگوریتم دایکسترا: اجرای نمایشی

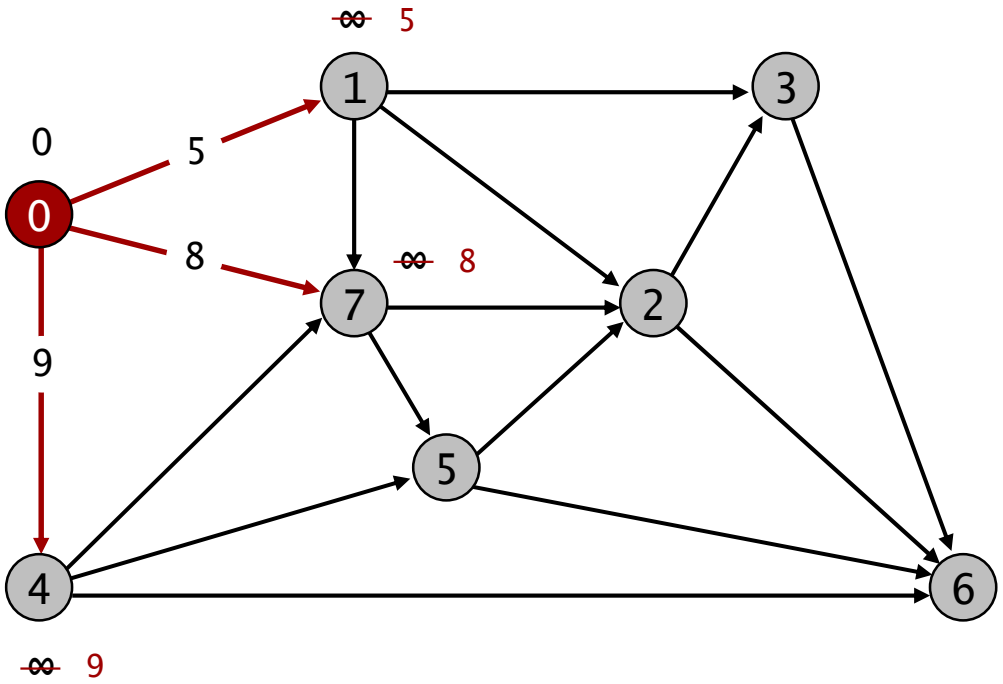
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
→ 0	0.0	-
1		
2		
3		
5		
4		
6		
7		

الگوریتم دایکسترا: اجرای نمایشی

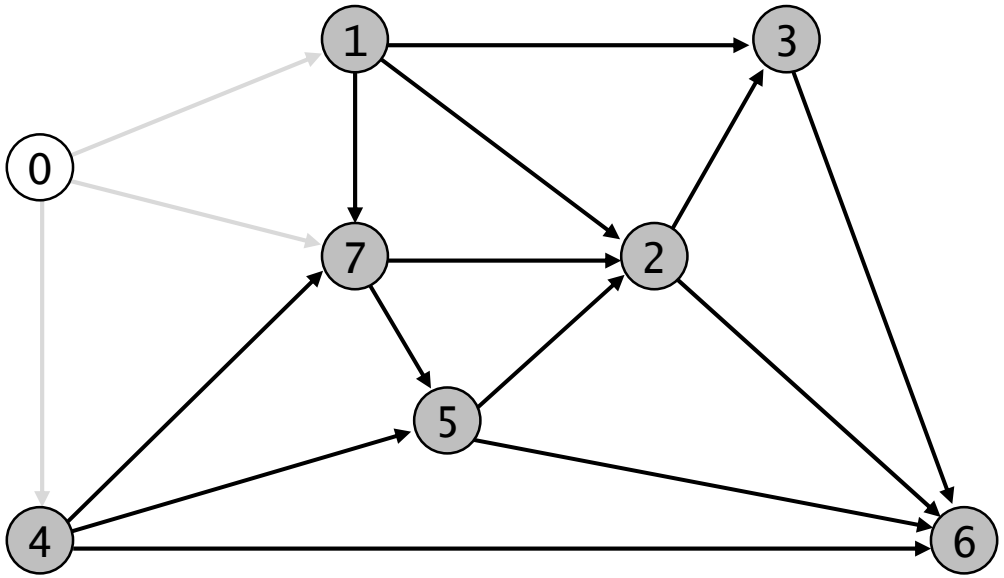
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
→ 0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

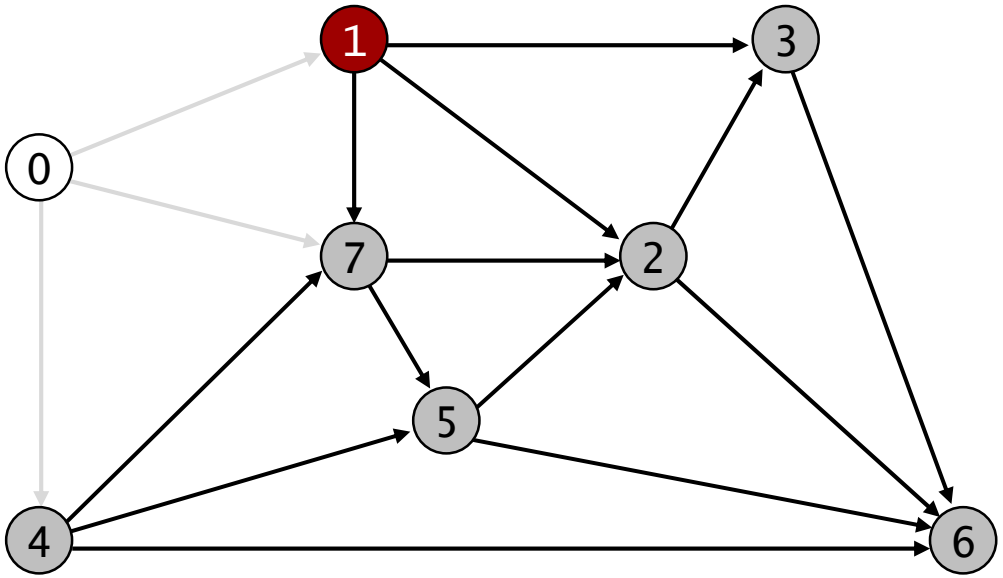
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

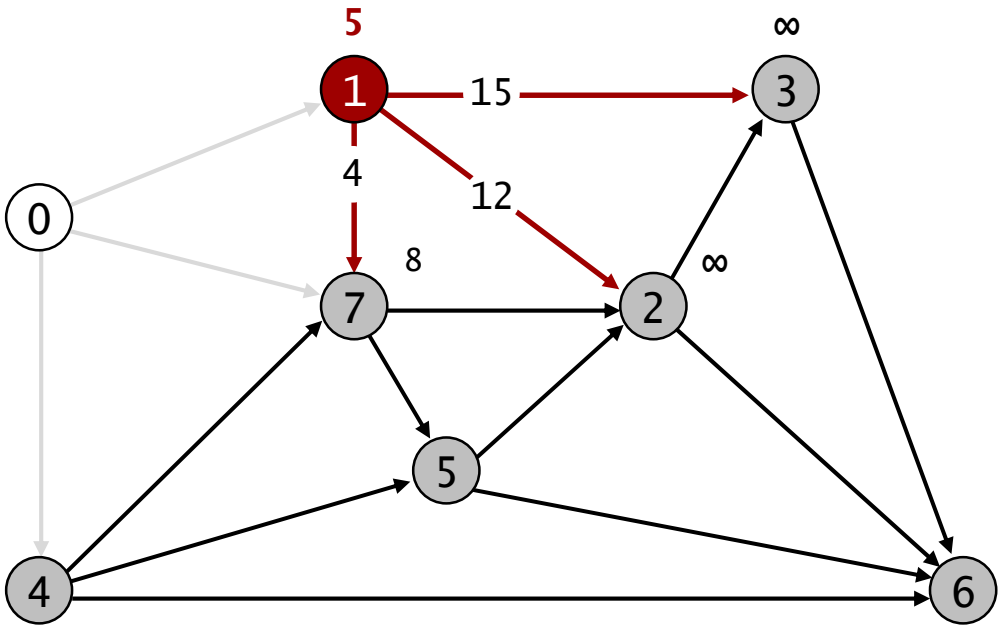
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

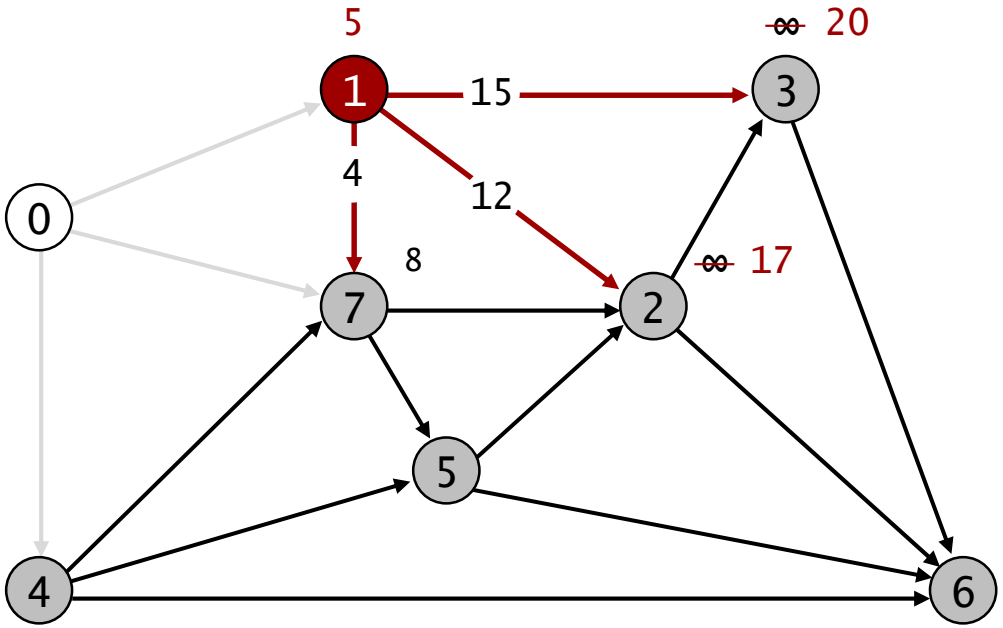
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

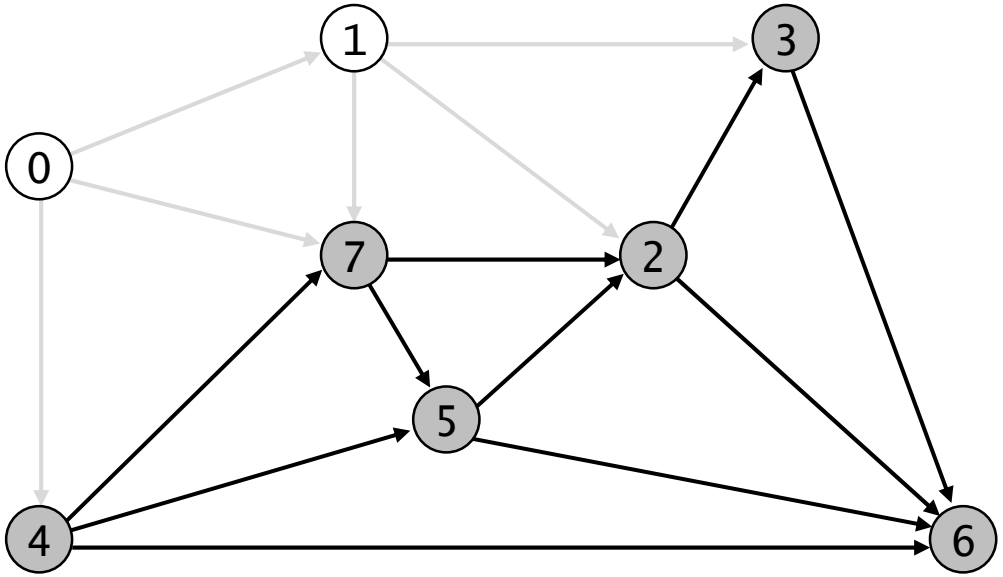
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

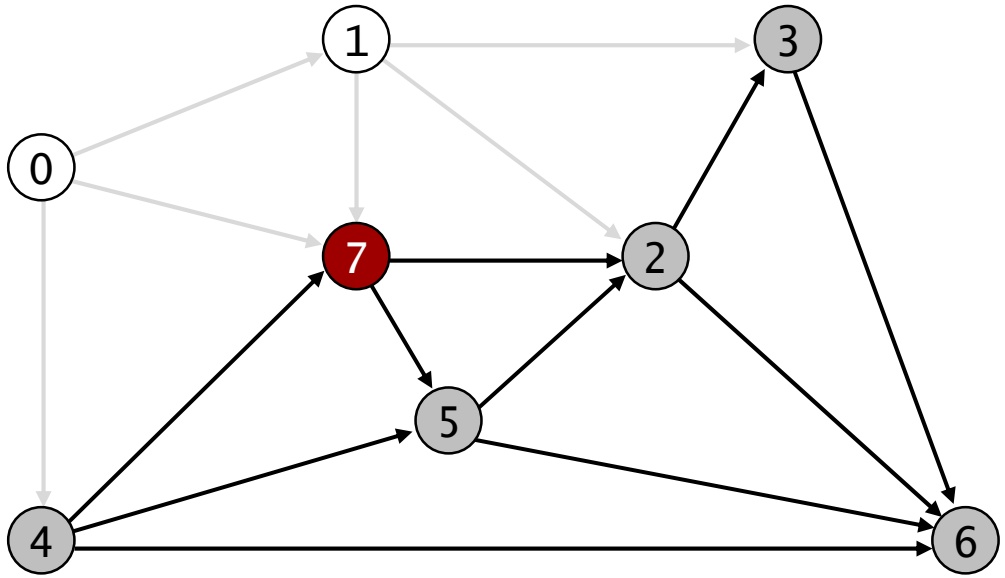
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

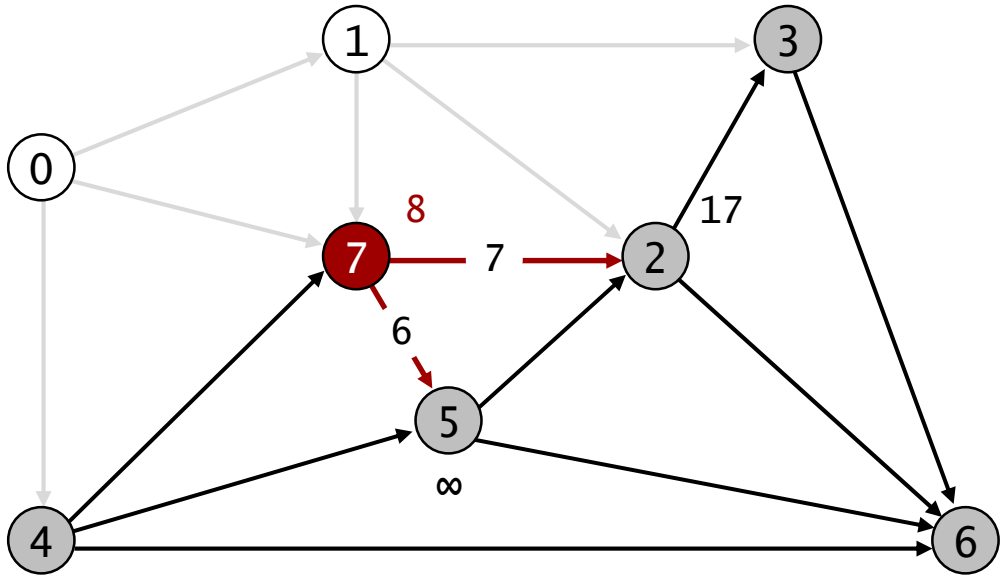
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	9.0	0->4
→ 6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

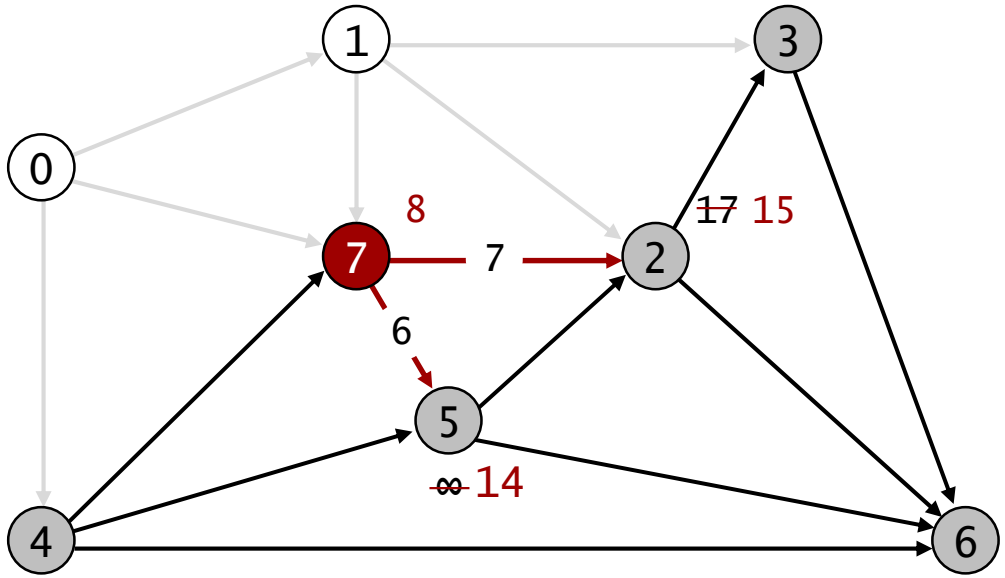
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	9.0	0->4
→ 6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

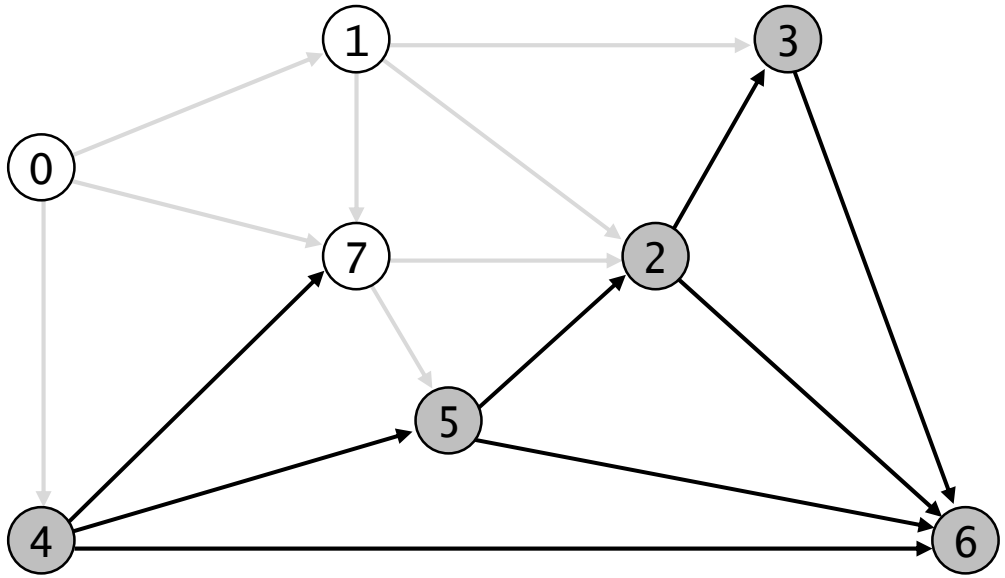
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	~~15.0~~	7->2
3	~~20.0~~	1->3
5	14.0	7->5
4	9.0	0->4
→ 6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

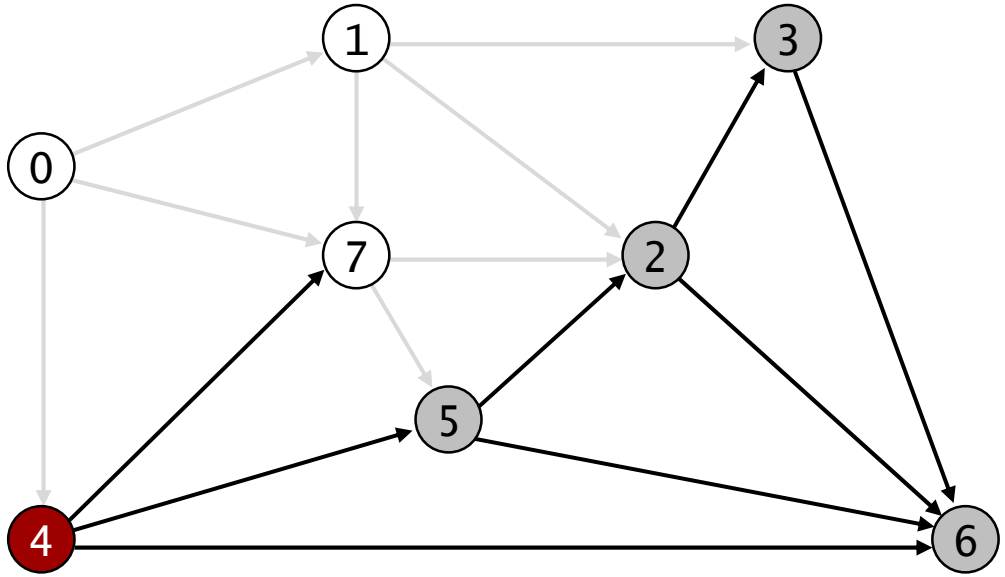
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	14.0	7->5
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

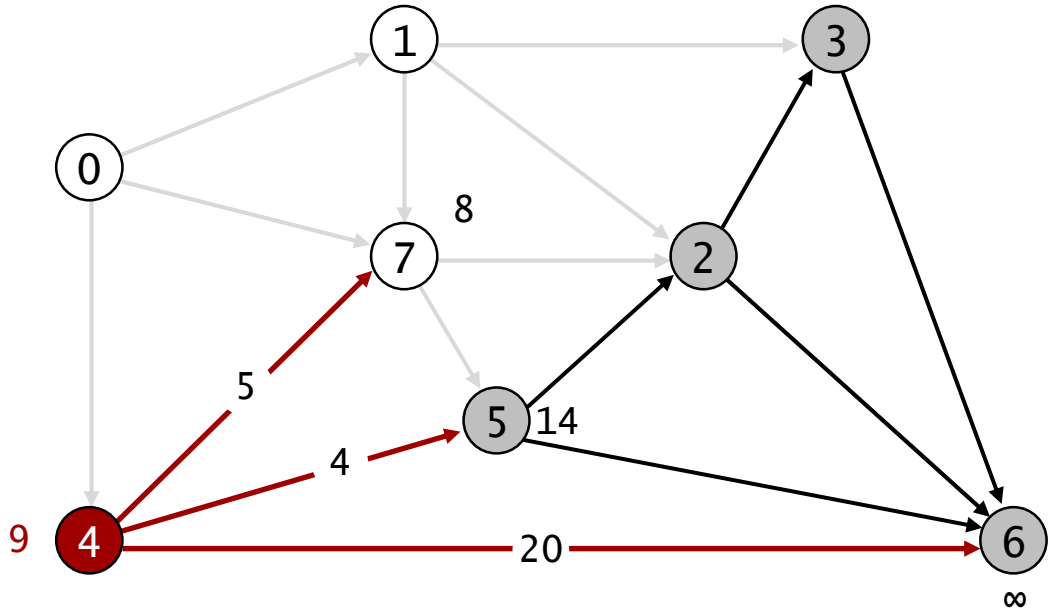
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
→ 5	14.0	7->5
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

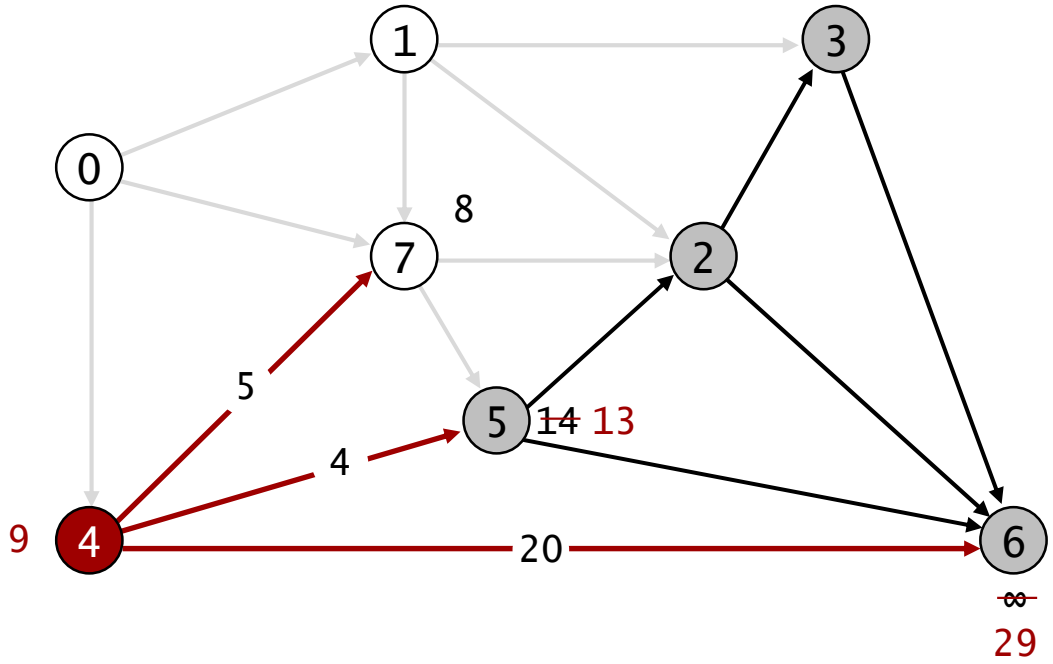
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
→ 5	14.0	7->5
4	9.0	0->4
6		
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

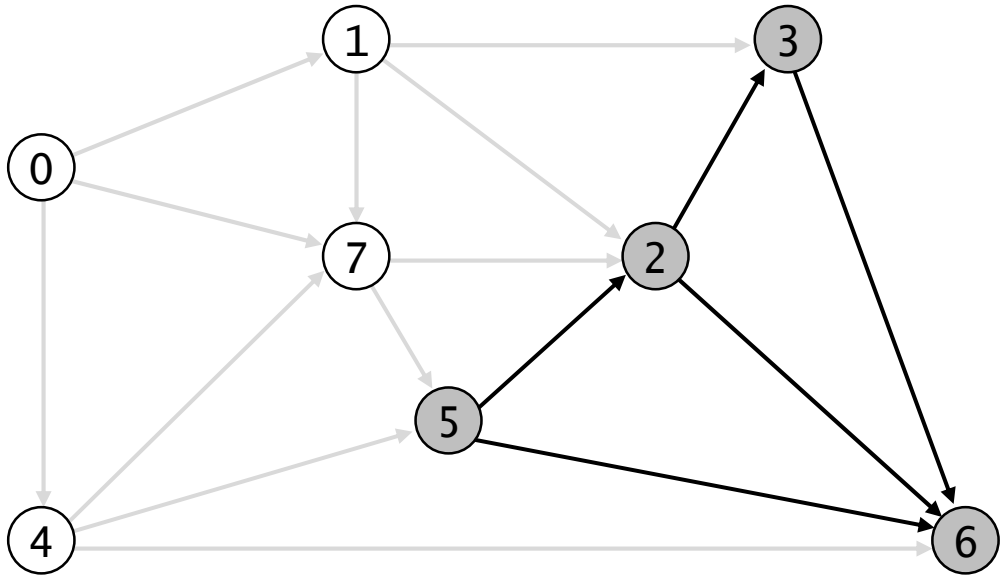
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	~~20.0~~	1->3
→ 5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

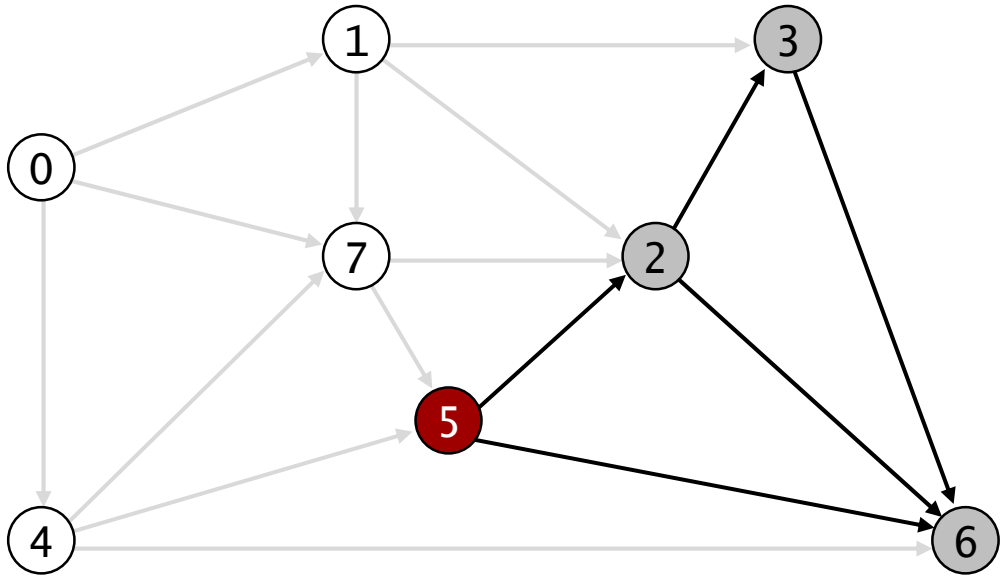
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

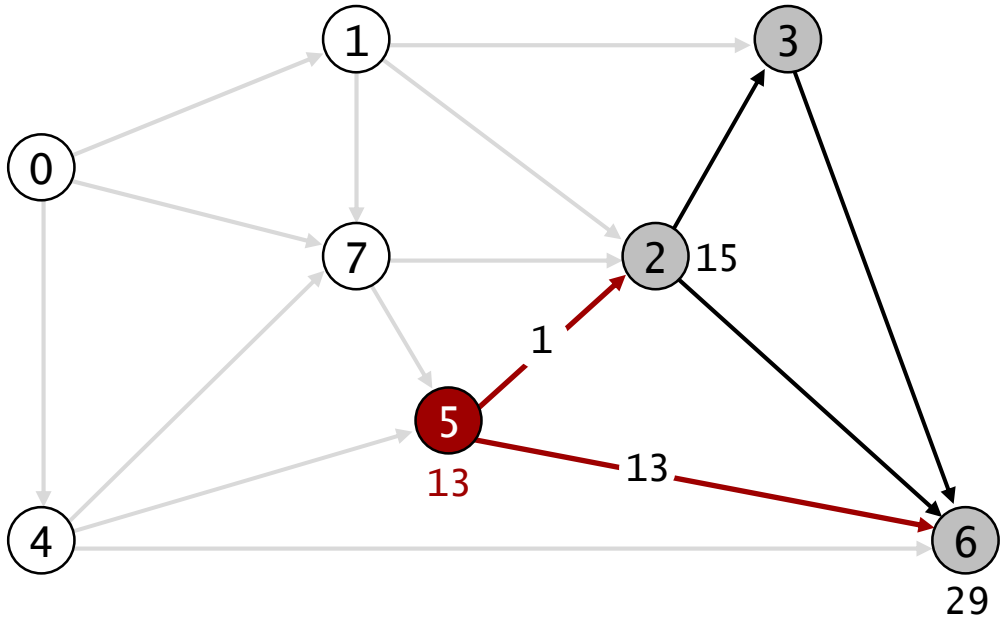
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

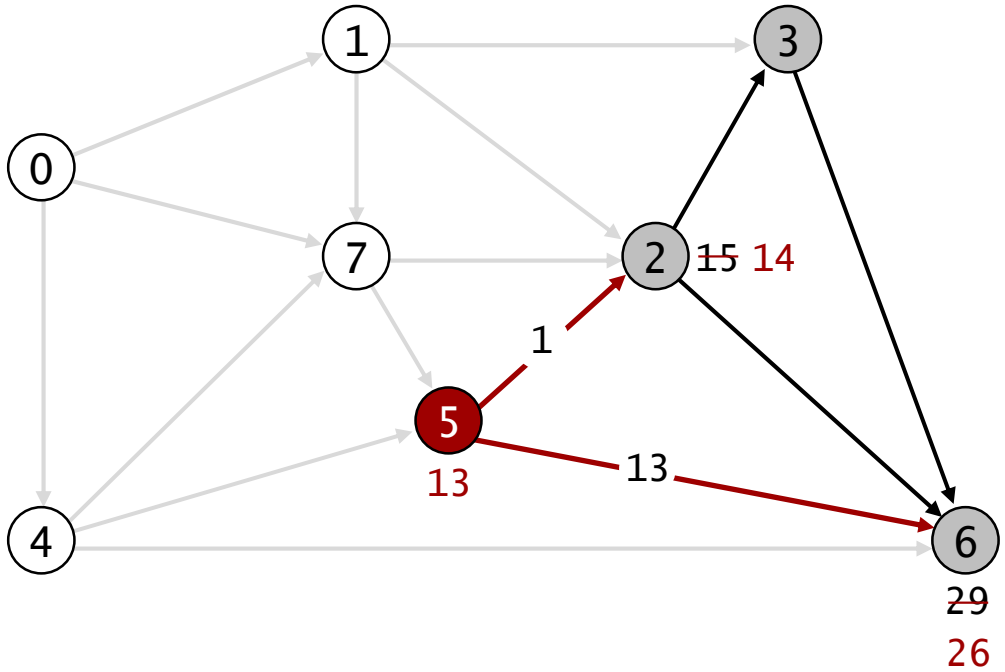
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

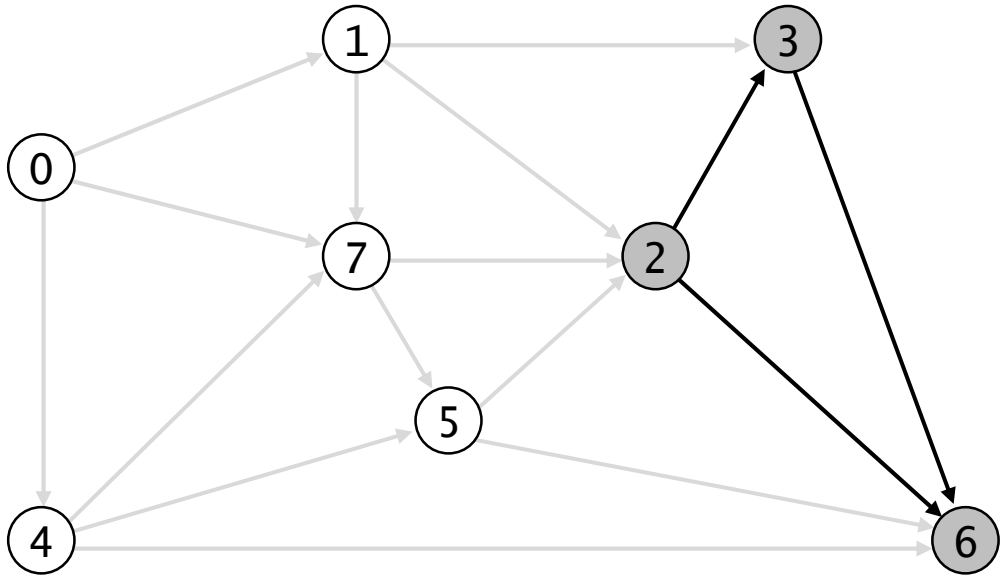
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

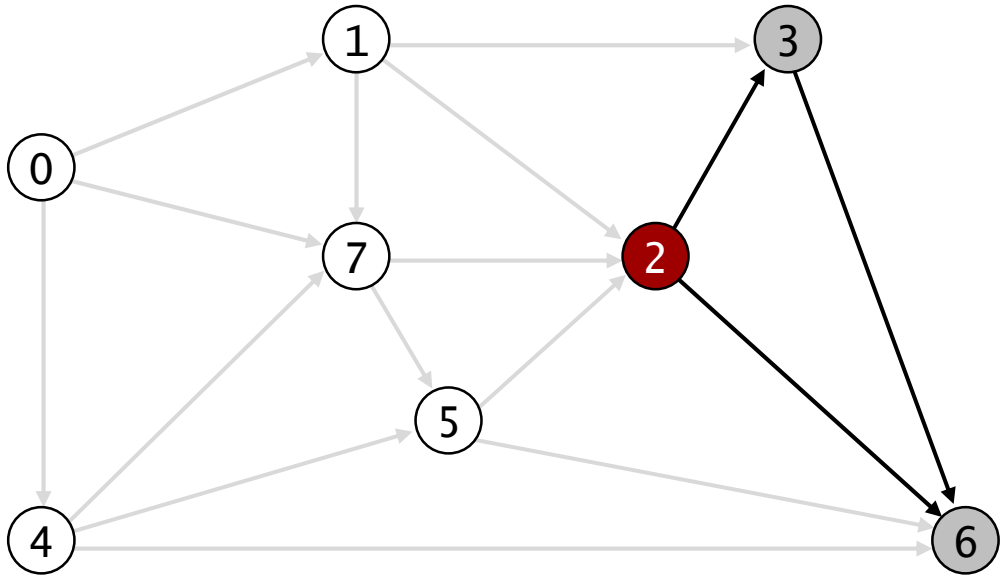
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس `s` در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

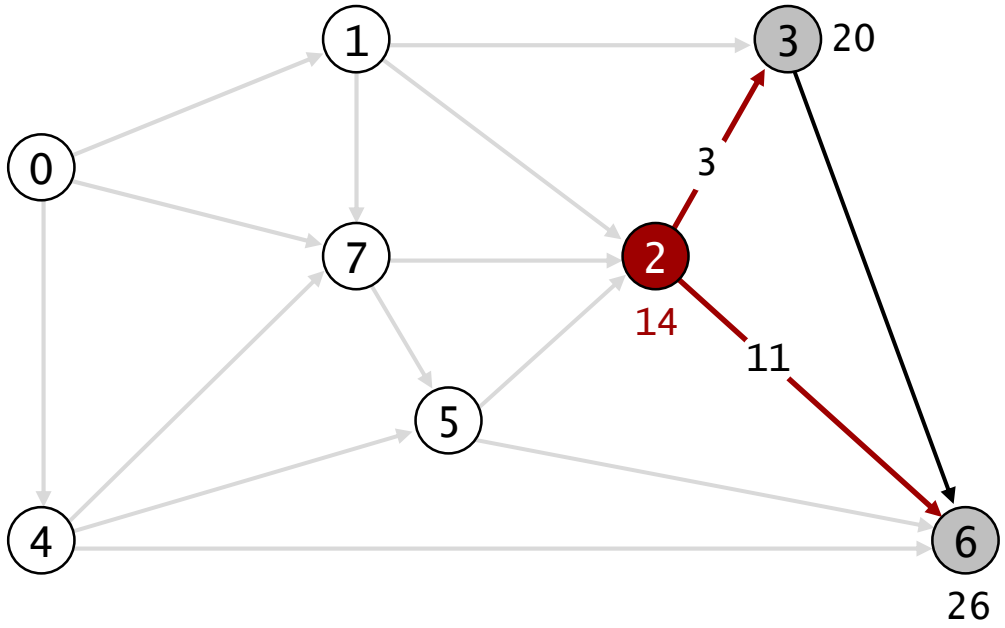
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
→ 2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

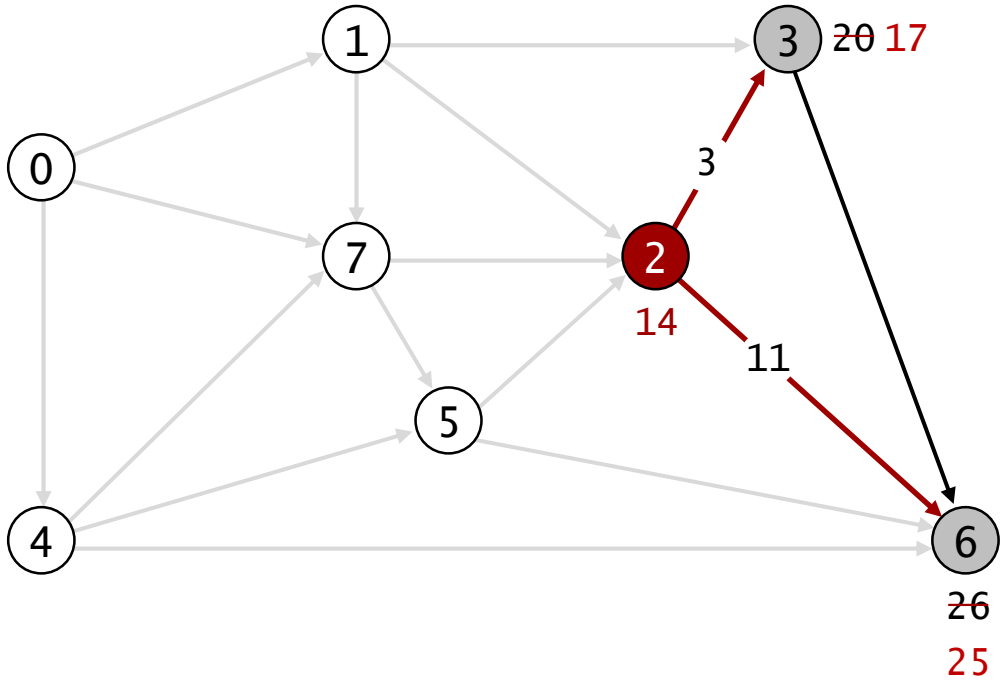
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
→ 2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

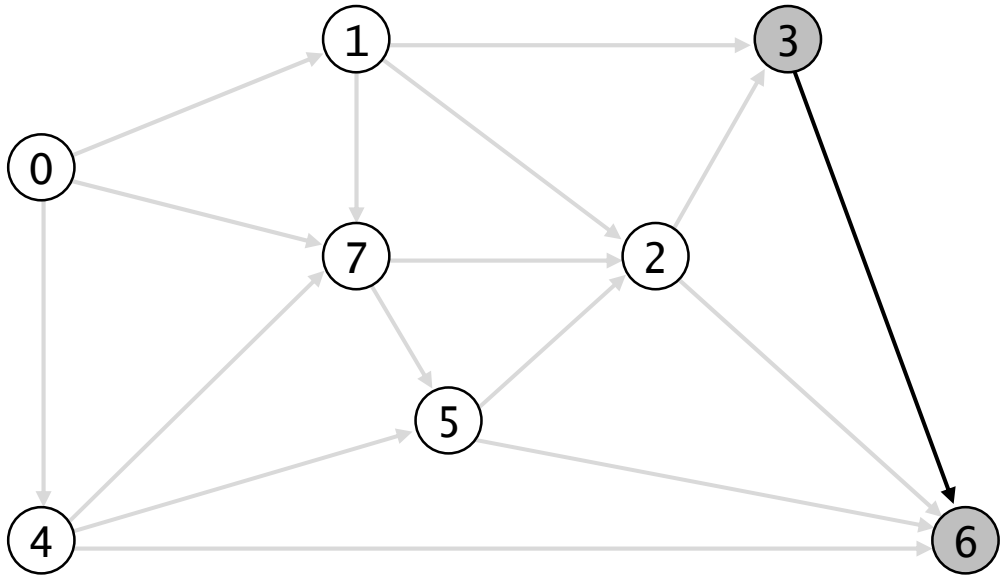
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

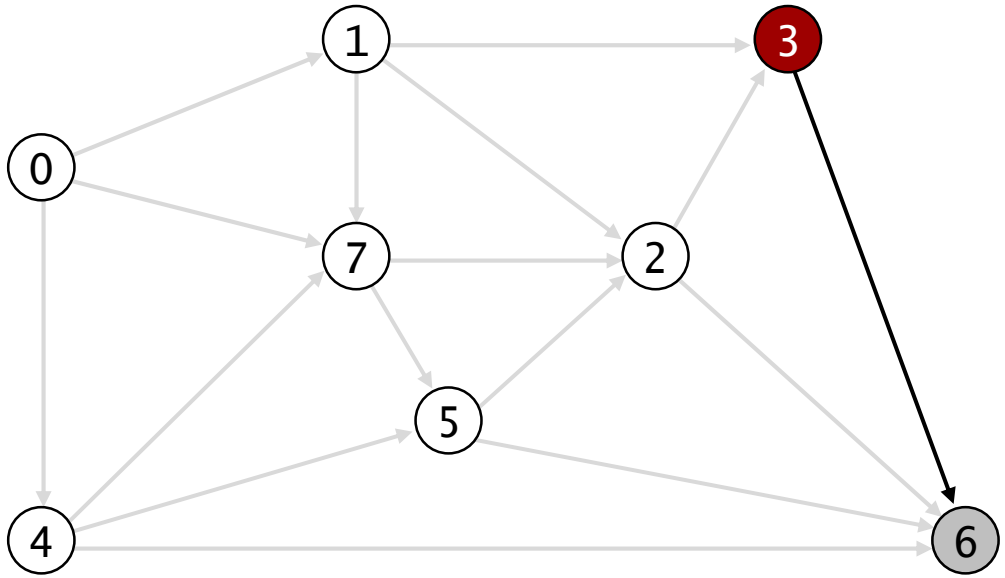
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

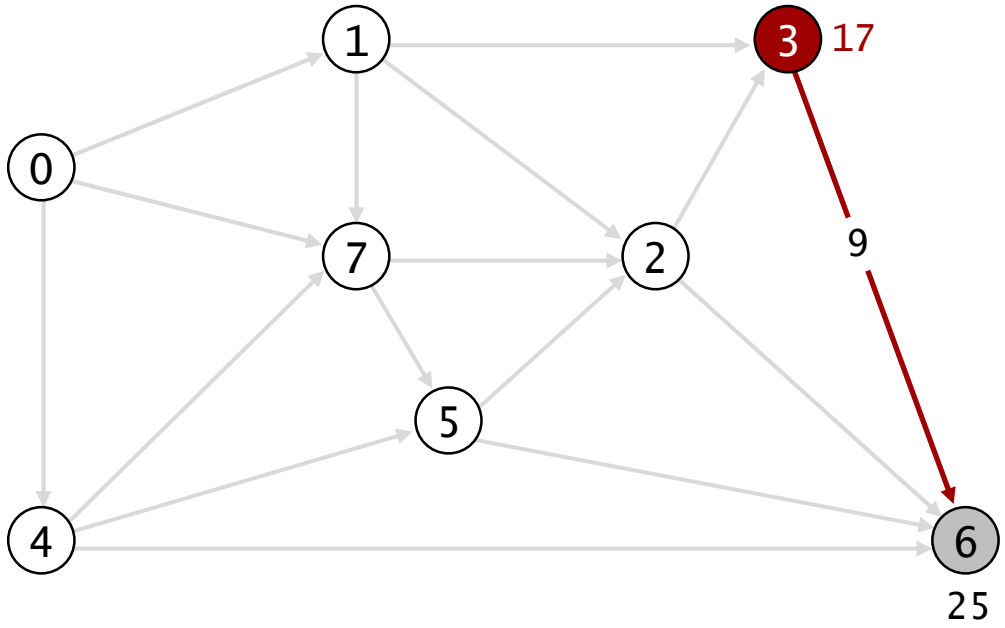
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

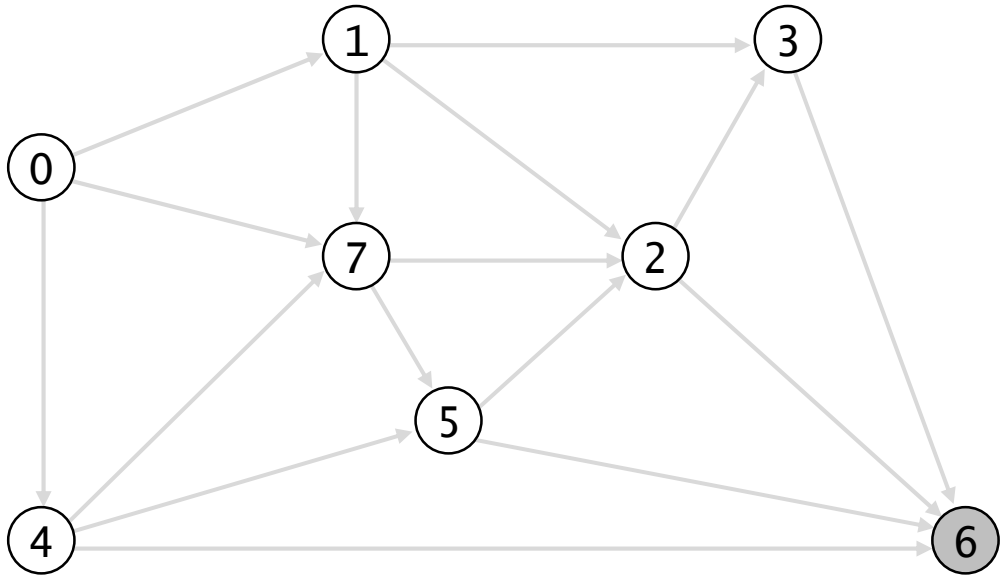
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

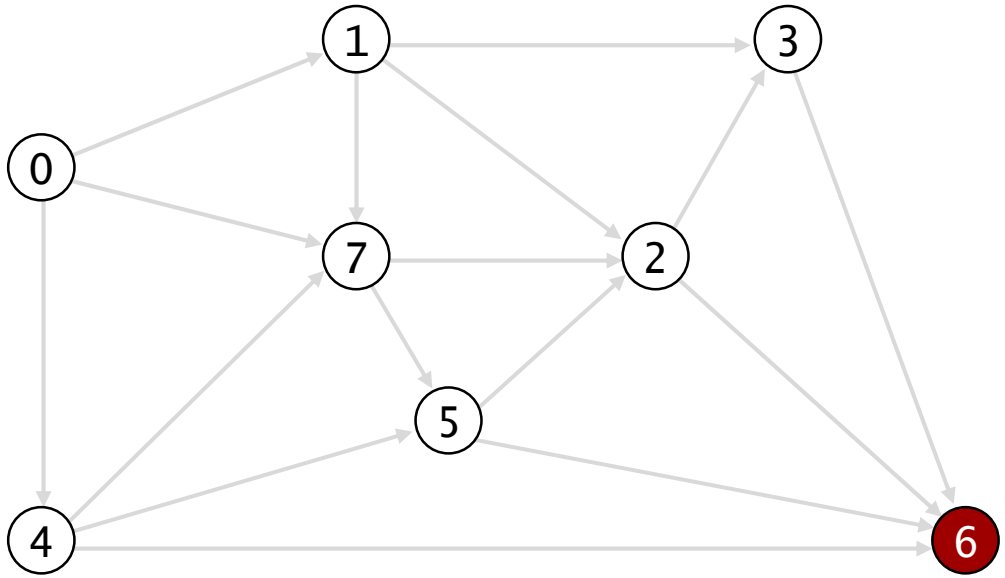
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس `s` در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

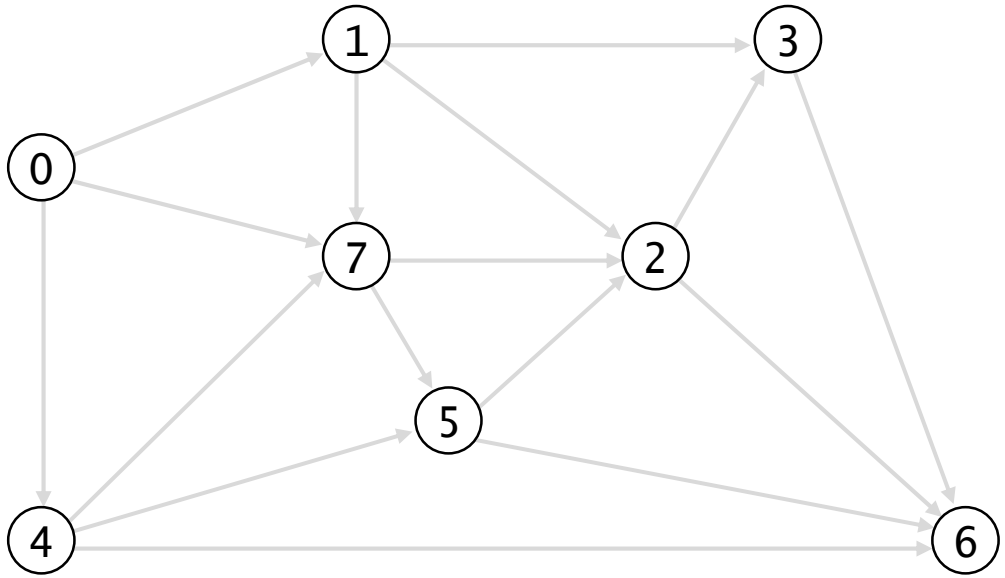
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



	v	distTo[]	edgeTo[]
	0	0.0	-
	1	5.0	0->1
	2	14.0	5->2
	3	17.0	2->3
	5	13.0	4->5
→	4	9.0	0->4
	6	25.0	2->6
	7	8.0	0->7

الگوریتم دایکسترا: اجرای نمایشی

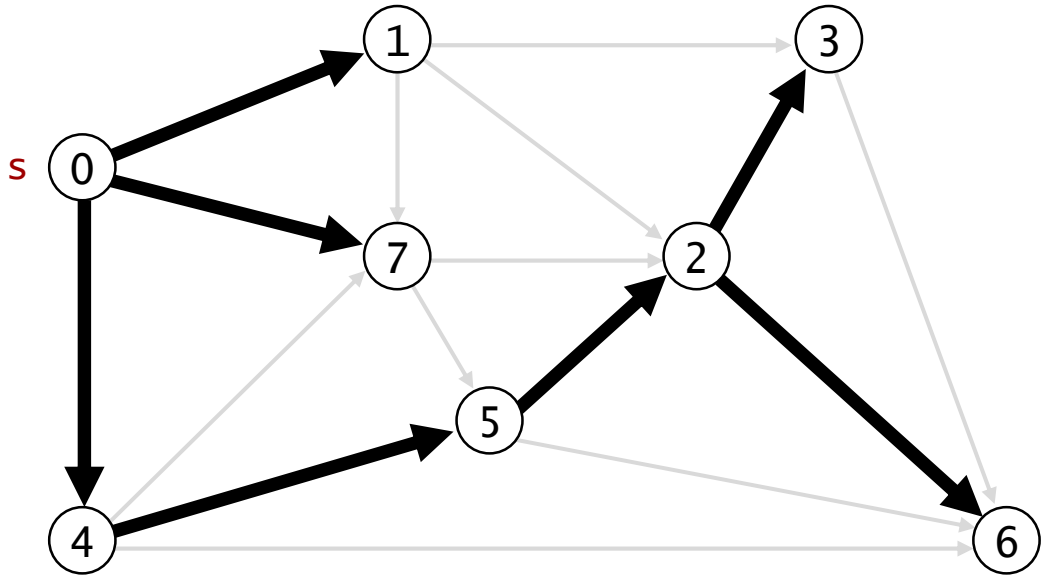
- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس `s` در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

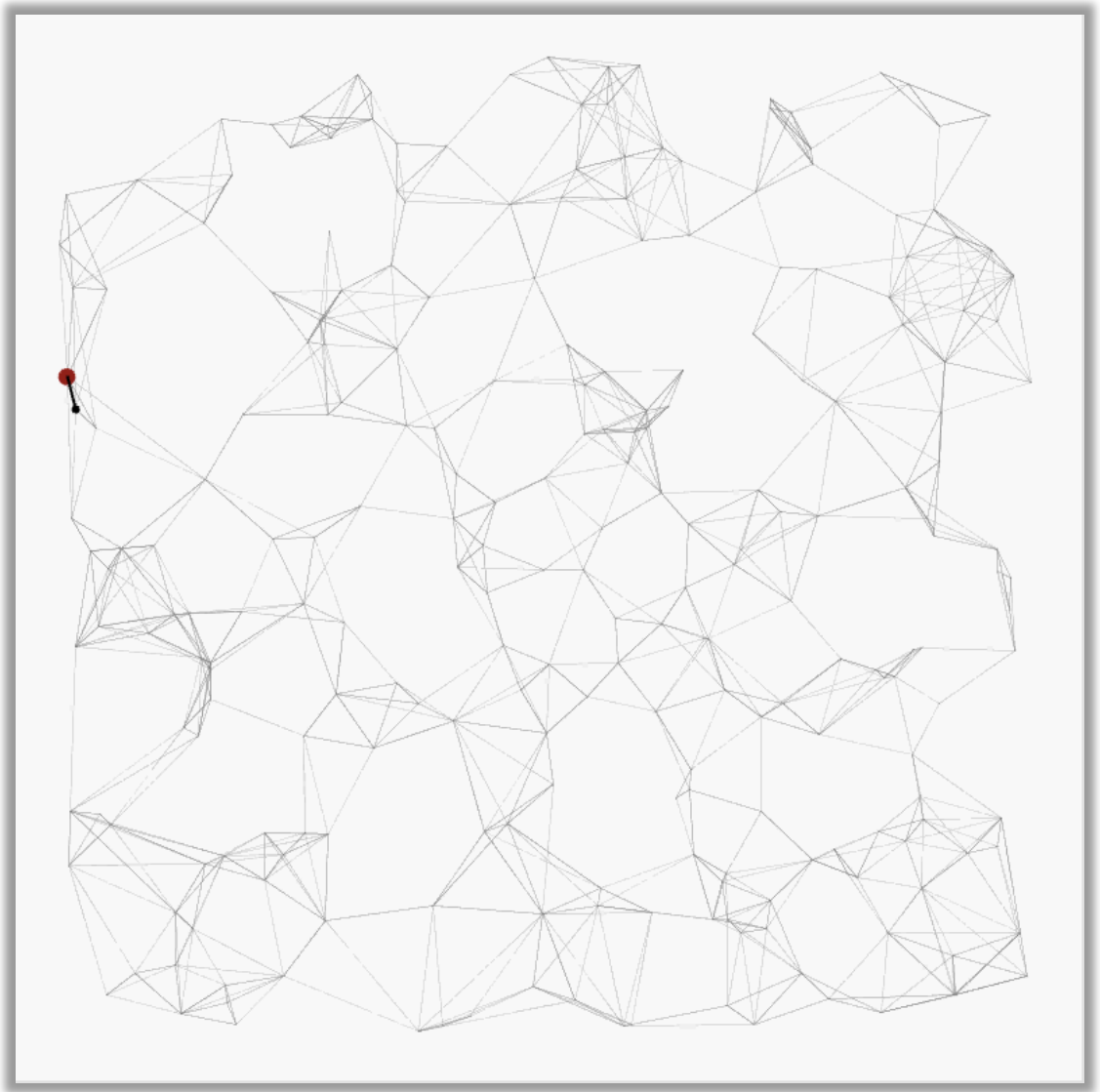
الگوریتم دایکسترا: اجرای نمایشی

- رئوس را به ترتیب صعودی فاصله‌ی آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یالهای خروجی از آن را راحت‌سازی کن.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

الگوریتم دایکسترا: اجرا



الگوریتم دایکسترا: اثبات درستی

- گزاره. الگوریتم دایجسترا در هر گراف جهت‌دار وزن‌دار بدون وزن های منفی درخت کوتاه‌ترین مسیر را محاسبه می‌کند.

- اثبات.
  - هر یال `e = v->w` دقیقاً یک بار راحت‌سازی می‌شود (در هنگام راحت‌سازی v)، که باعث می‌شود `distTo[w] ≤ distTo[v] + e.weight()`.
  - نامساوی بالا تا انتهای اجرای الگوریتم برقرار می‌ماند، زیرا:
    - `distTo[w]` هرگز افزایش نمی‌یابد،
    - `distTo[v]` نیز از این به بعد تغییر نمی‌کند،
  - در نتیجه، پس از خاتمه اجرای الگوریتم، شرط بهینگی کوتاه‌ترین مسیرها برقرار خواهد بود.

الگوریتم دایکسترا: پیاده‌سازی


```
public class DijkstraSP
{
    private DirectedEdge[] edgeTo;
    private double[] distTo;
    private IndexMinPQ<Double> pq;

    public DijkstraSP(EdgeWeightedDigraph G, int s)
    {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];
        pq = new IndexMinPQ<Double>(G.V());

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        pq.insert(s, 0.0);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();
            for (DirectedEdge e : G.adj(v))
                relax(e);
        }
    }
```

الگوریتم دایکسترا: پیاده‌سازی


```
    private void relax(DirectedEdge e)
    {
        int v = e.from(), w = e.to();
        if (distTo[w] > distTo[v] + e.weight())
        {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
            else                pq.insert     (w, distTo[w]);
        }
    }

}
```

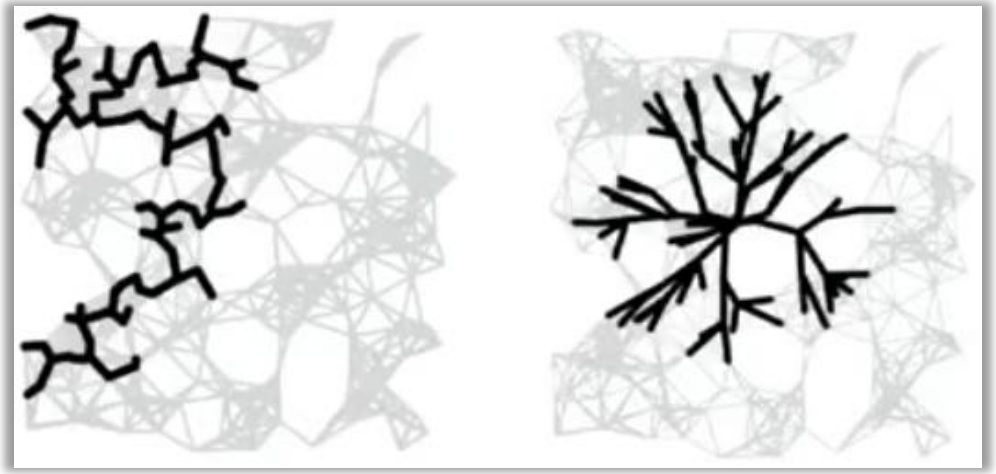
محاسبه درخت پوشا در گراف‌ها

الگوریتم دایکسترا آشنا به نظر می‌رسد؟

- الگوریتم پریم همان الگوریتم دایکسترا است.
- هر دو الگوریتم به خانواده‌ای از الگوریتم‌ها تعلق دارند که کارشان محاسبه‌ی درخت پوشا است.

تفاوت اصلی. قانون استفاده شده به منظور انتخاب رأس بعدی!

- پریم: نزدیک‌ترین رأس به درخت (از طریق یک یال بدون جهت)
- دایکسترا: نزدیک‌ترین رأس به مبدأ (از طریق یک یال جهت‌دار)



الگوریتم دایکسترا: زمان اجرا

- زمان اجرای الگوریتم دایکسترا بستگی به نحوه پیاده‌سازی صف اولویت دارد:
  - V مرتبه درج، V مرتبه حذف کوچک‌ترین، E مرتبه کاهش کلید

پیاده‌سازی	درج	حذف کوچک‌ترین	کاهش کلید	زمان اجرا
آرایه	1	V	1	V^2
هرم دودویی	$\log V$	$\log V$	$\log V$	$E \log V$
هرم d طرفه	$d \log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
هرم فیبوناچی	1 †	$\log V$ †	1 †	$E + V \log V$

† هزینه‌ی سرشکن شده

- جمع بندی.
  - پیاده‌سازی با آرایه برای گراف‌های شلوغ بهینه است.
  - هرم دودویی برای گراف‌های خلوت بسیار سریع‌تر است.
  - هرم فیبوناچی به لحاظ نظری بهترین است، اما در عمل ارزش پیاده‌سازی ندارد.

گراف جهت‌دار بدون دور (DAG)

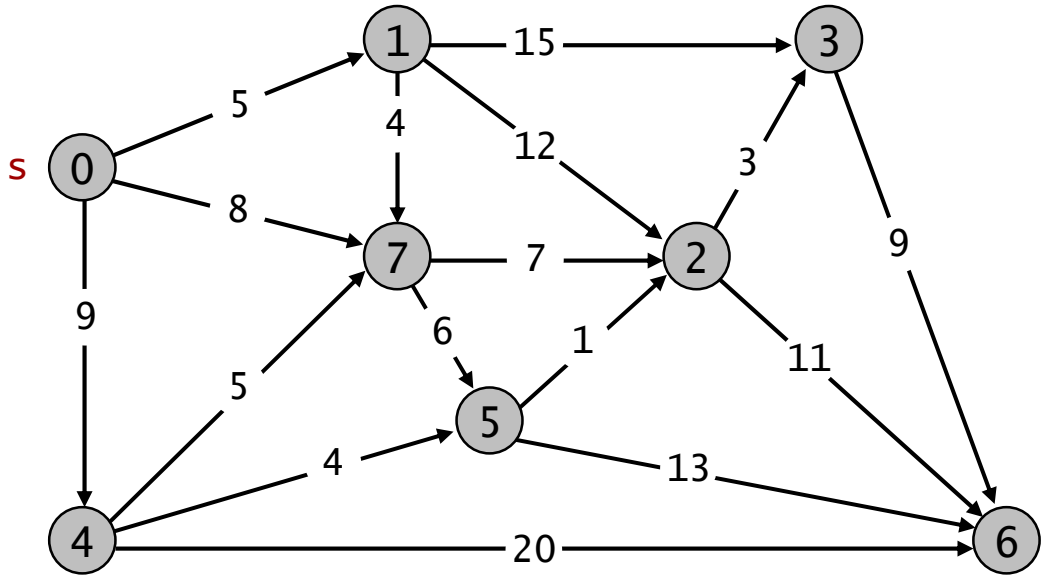
گراف جهت‌دار بدون دور

- س. فرض کنید یک گراف وزن‌دار جهت‌دار شامل هیچ دوری نباشد. آیا محاسبه‌ی کوتاه‌ترین مسیرها در چنین گرافی نسبت به گراف‌های جهت‌دار عمومی ساده‌تر است؟

- ج. بله

گراف جهت‌دار بدون دور: اجرای نمایشی

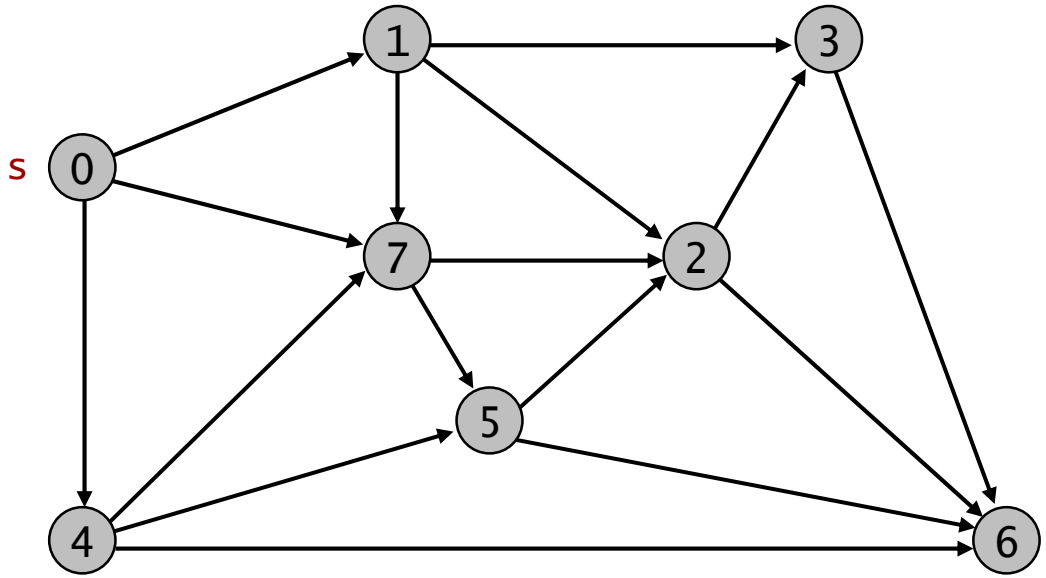
- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.




```
0->1   5.0
0->4   9.0
0->7   8.0
1->2  12.0
1->3  15.0
1->7   4.0
2->3   3.0
2->6  11.0
3->6   9.0
4->5   4.0
4->6  20.0
4->7   5.0
5->2   1.0
5->6  13.0
7->5   6.0
7->2   7.0
```

گراف جهت‌دار بدون دور: اجرای نمایشی

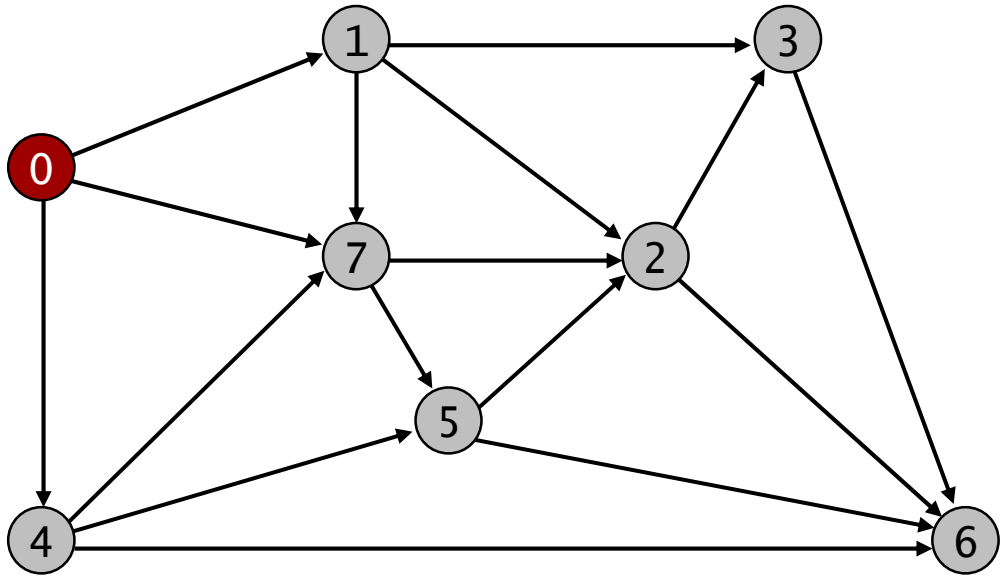
- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



ترتیب توپولوژیکی **0 1 4 7 5 2 3 6**

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

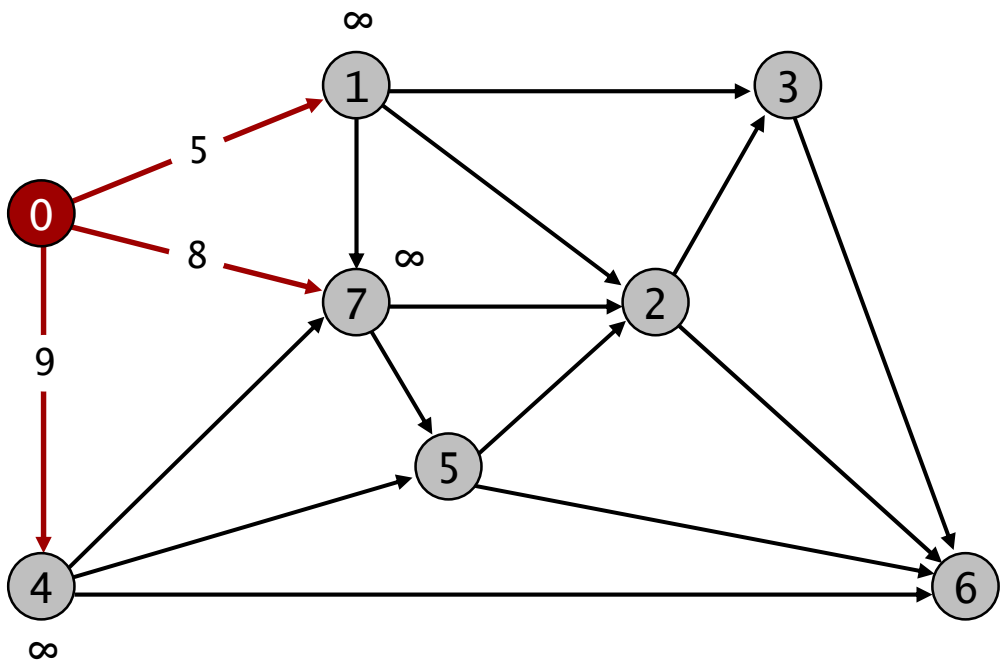


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
5		
4		
6		
7		

گراف جهت‌دار بدون دور: اجرای نمایشی

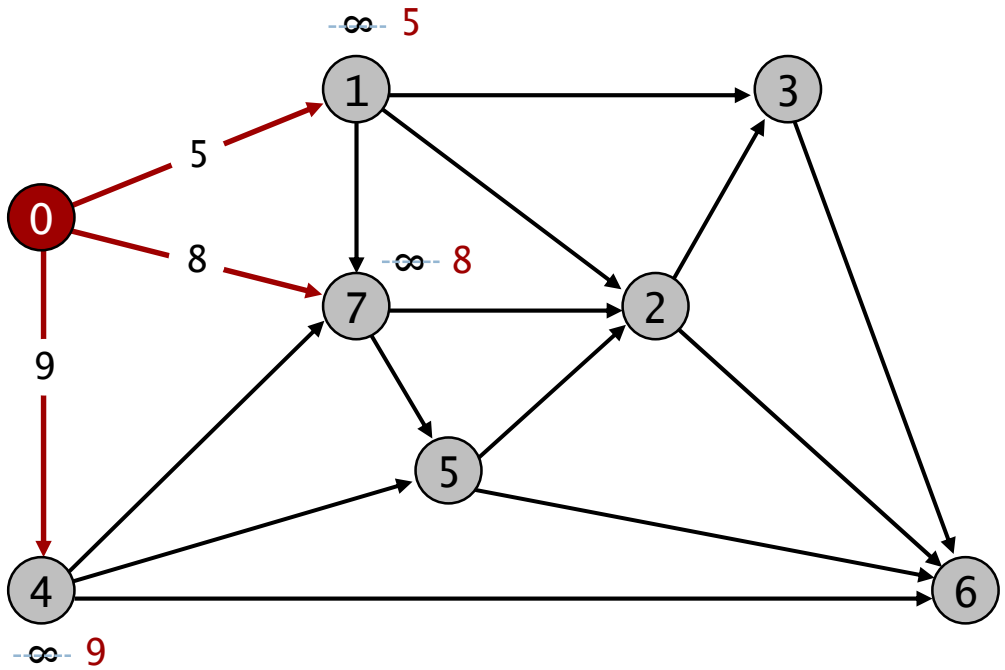
- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
5		
4		
6		
7		

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

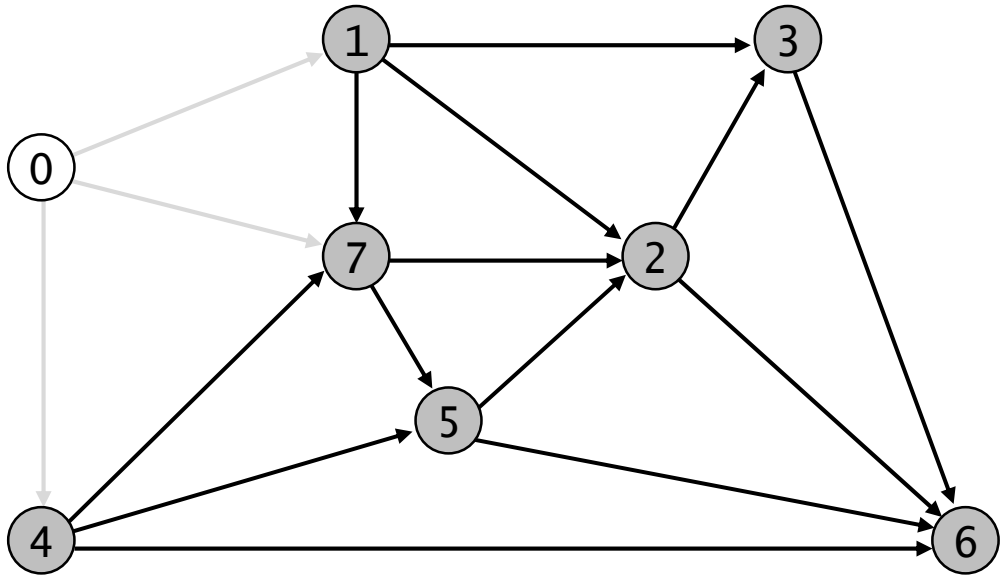


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



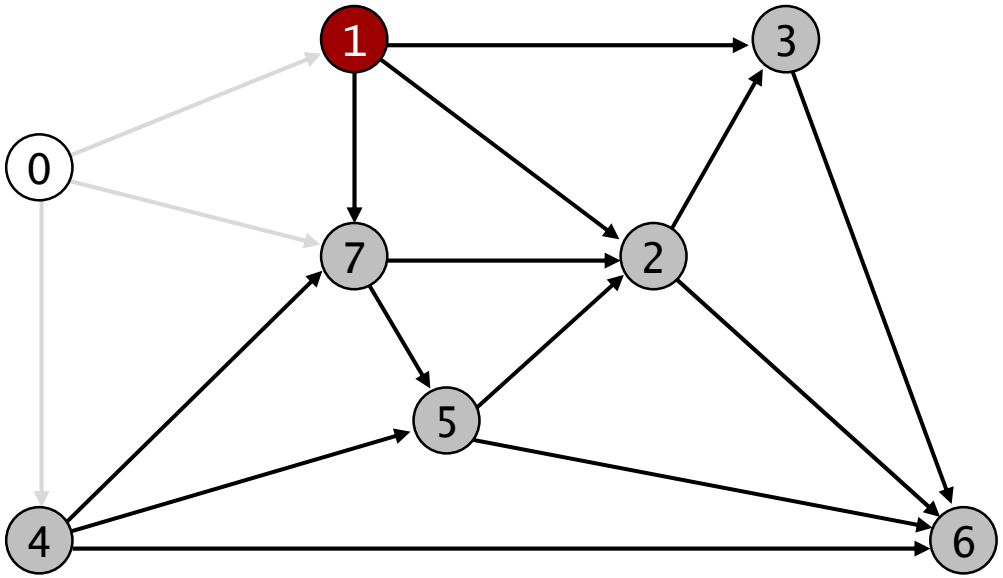
↓

0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

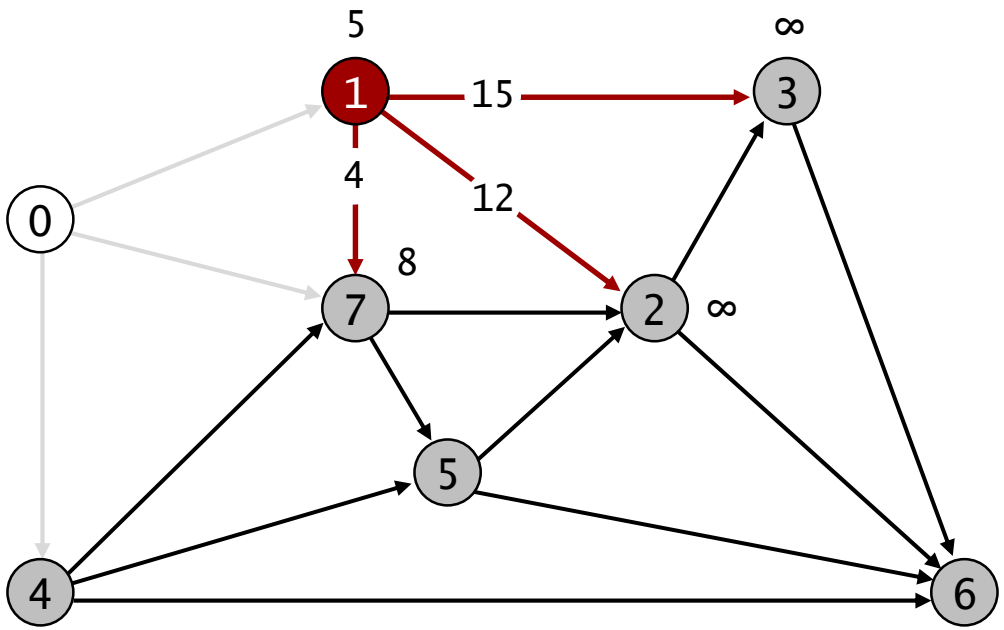


	0	1	4	7	5	2	3	6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

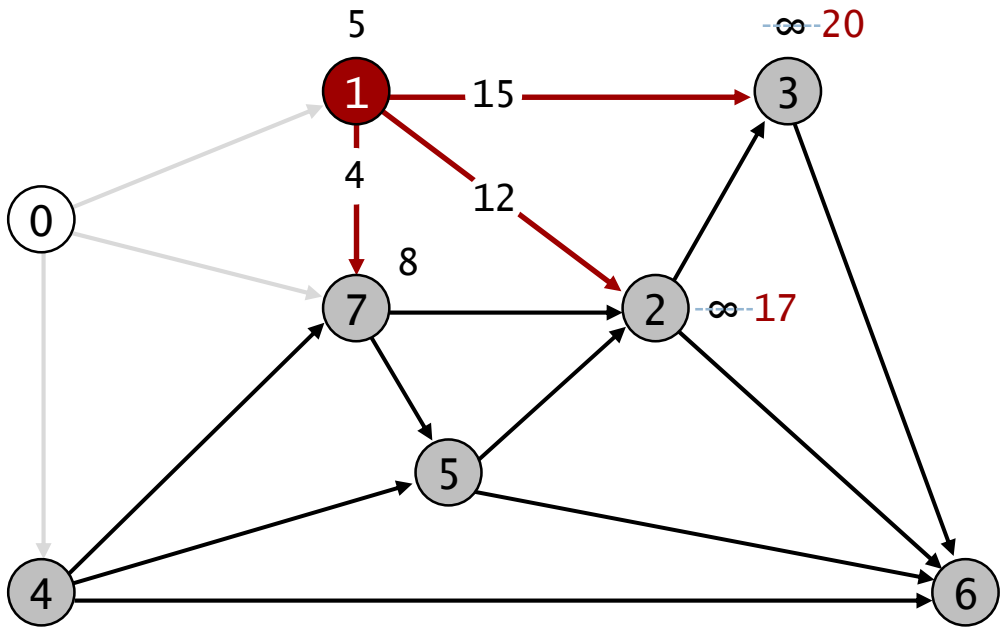
- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
5		
4	9.0	0->4
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

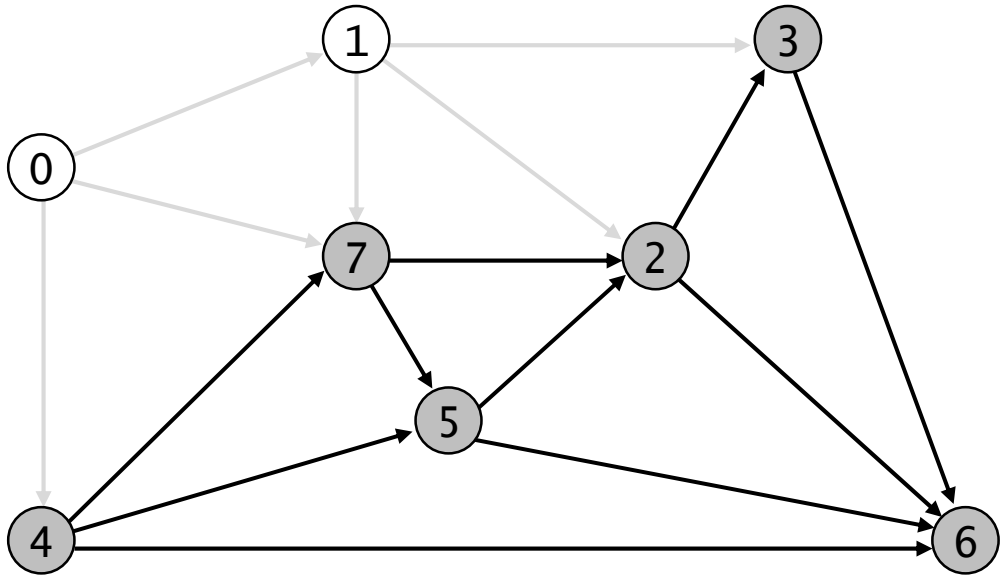
- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



	0	1	4	7	5	2	3	6
	v	distTo[]		edgeTo[]				
	0	0.0		-				
→	1	5.0		0->1				
	2	17.0		1->2				
	3	20.0		1->3				
	5							
	4	9.0		0->4				
	6							
	7	8.0		0->7				

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

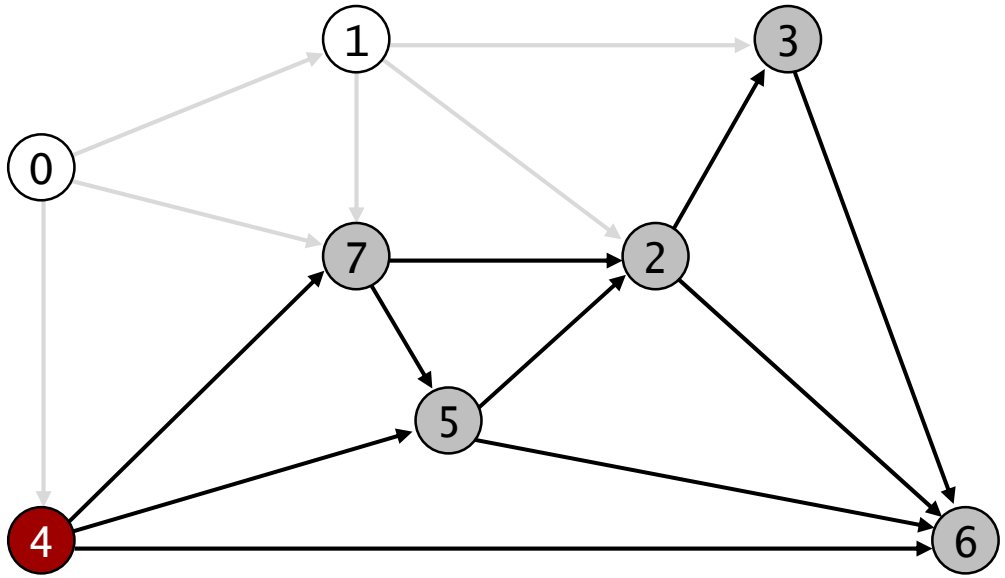


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	9.0	0->4
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

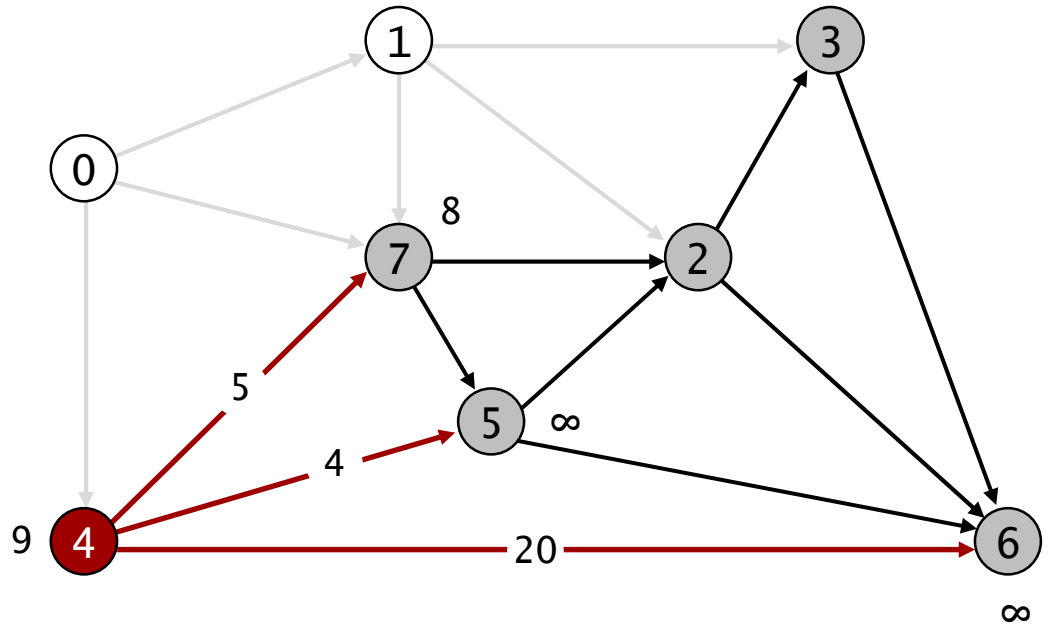


0 1 **4** 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	**9.0**	**0->4**
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

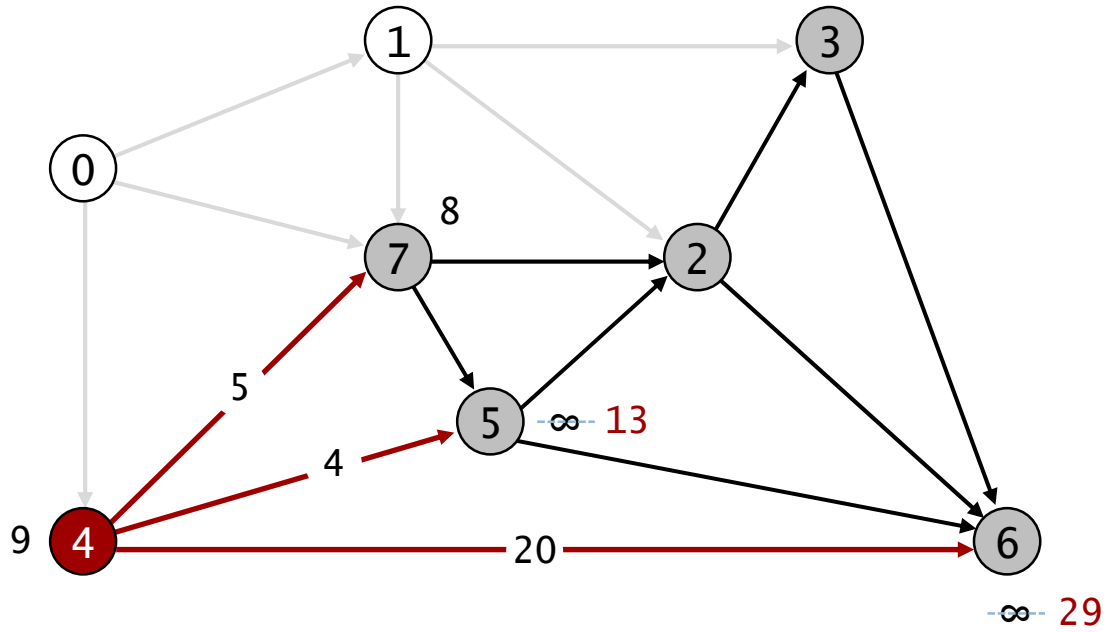


0 1 **4** 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5		
4	**9.0**	**0->4**
6		
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

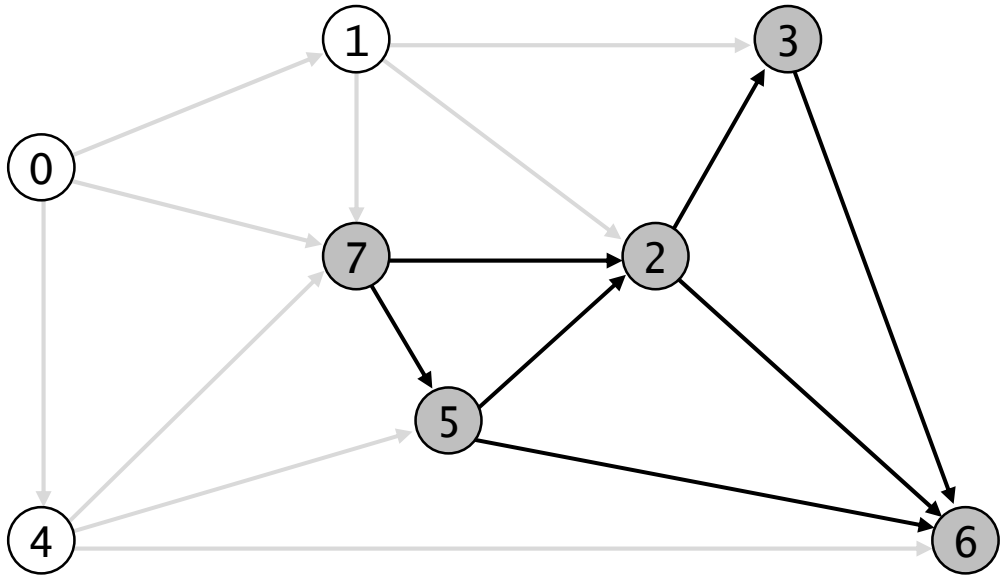


0 1 **4** 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

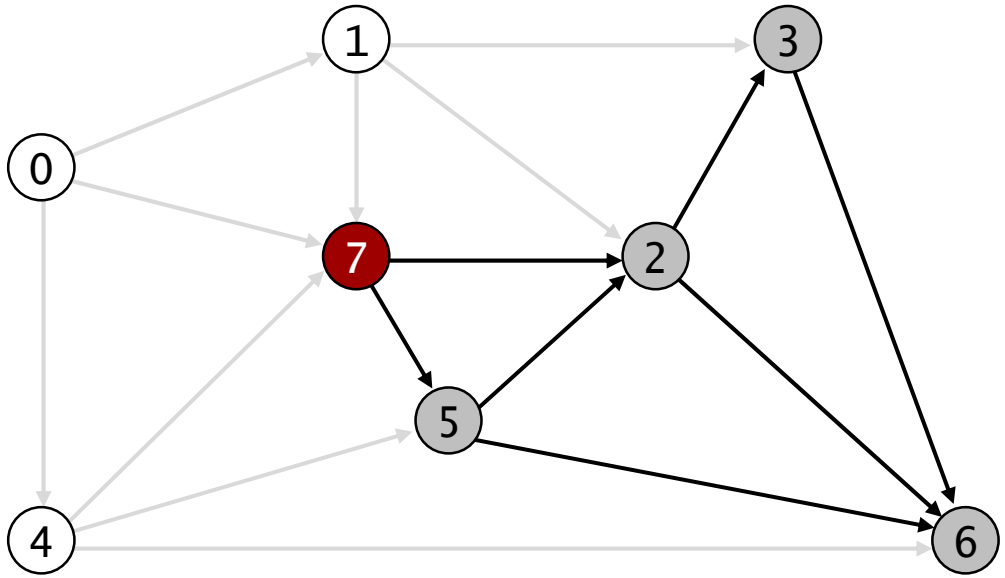


0 1 4 **7** **5** **2** **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

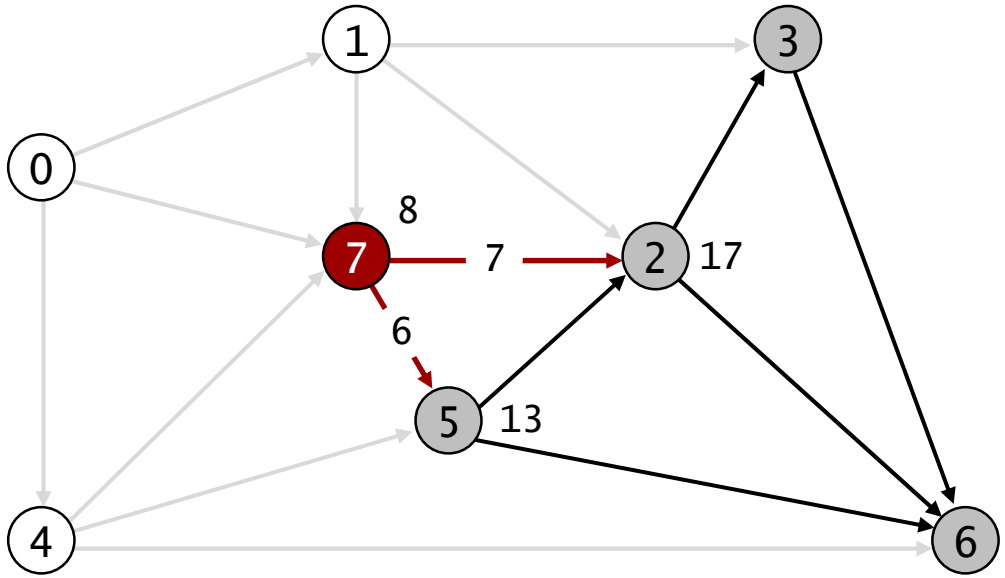


0 1 4 **7** **5** **2** **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
→ 6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

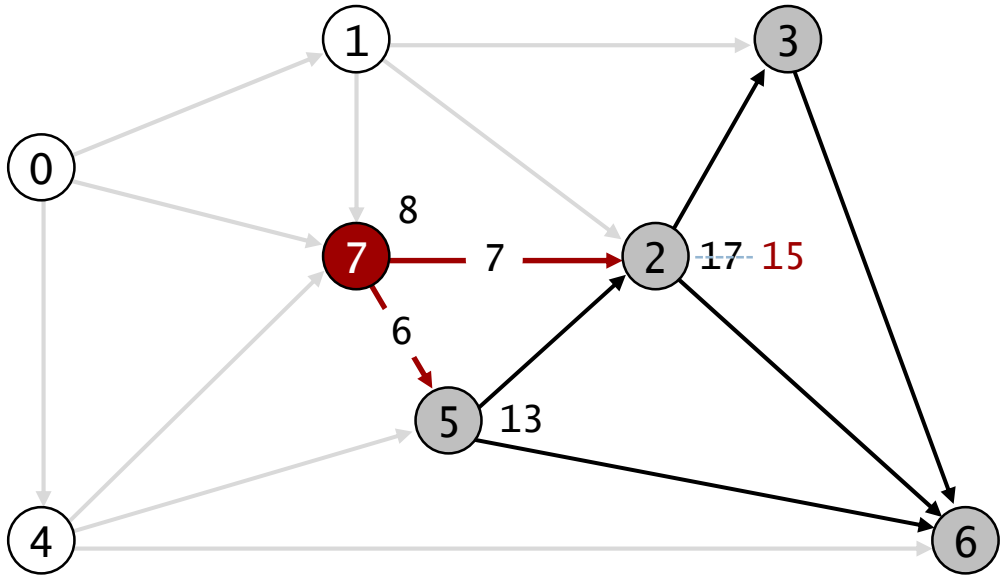


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
→ 6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

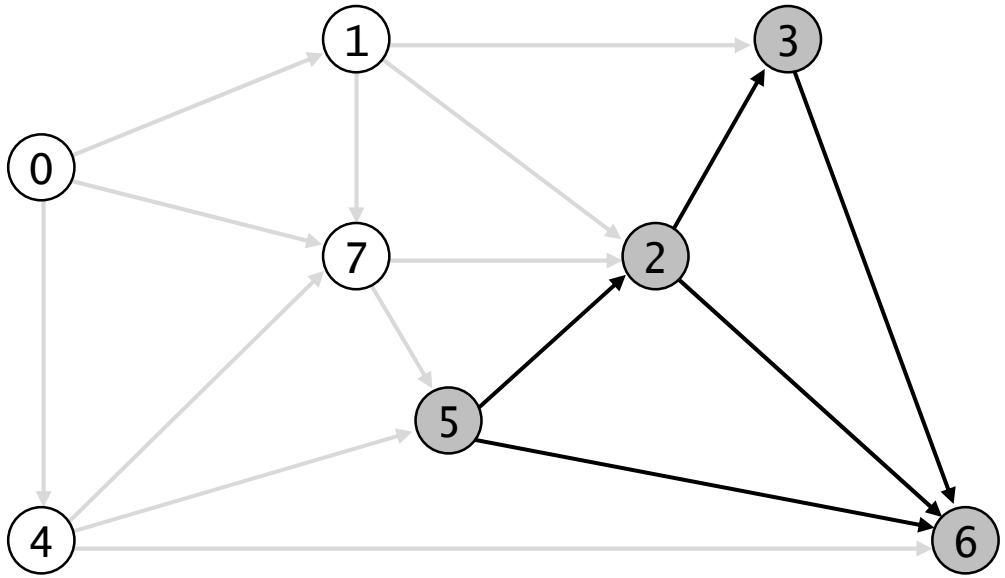


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
→ 6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

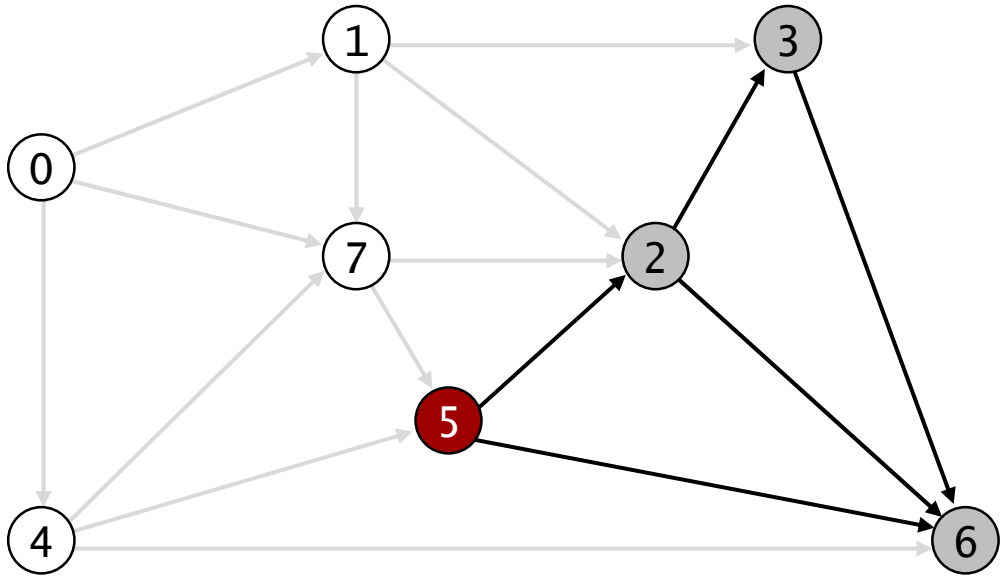


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

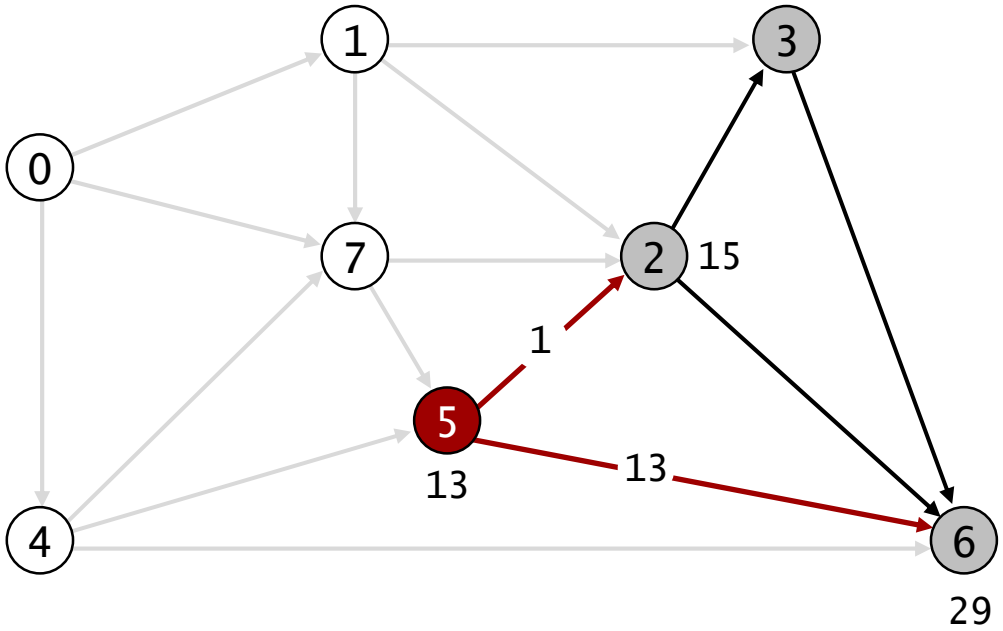


0 1 4 7 **5** **2** **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

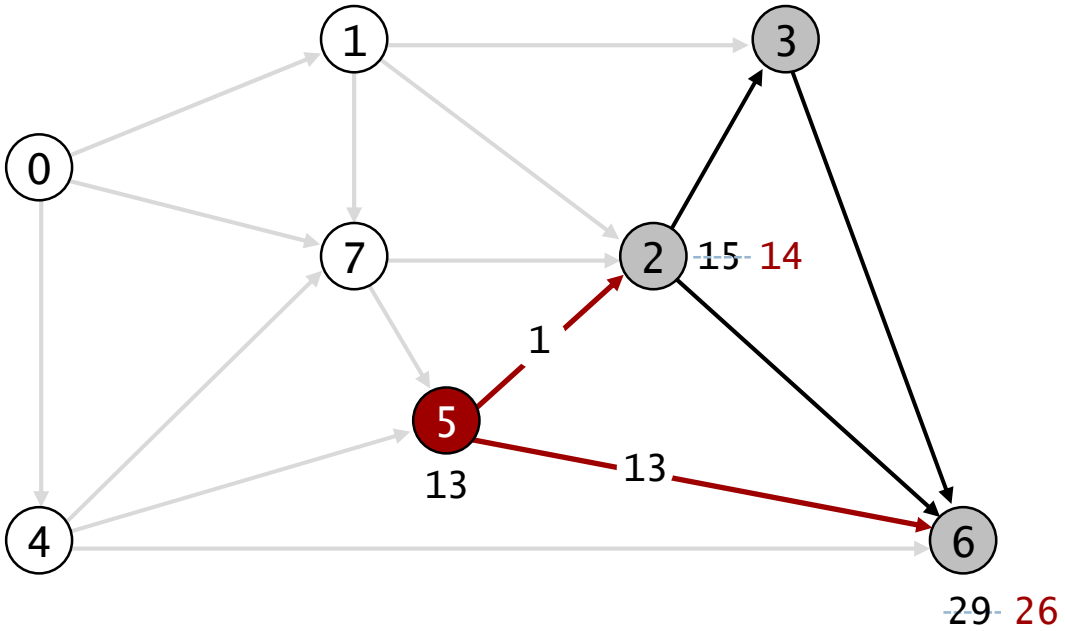


0 1 4 7 **5** **2** **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	15.0	7->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	29.0	4->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

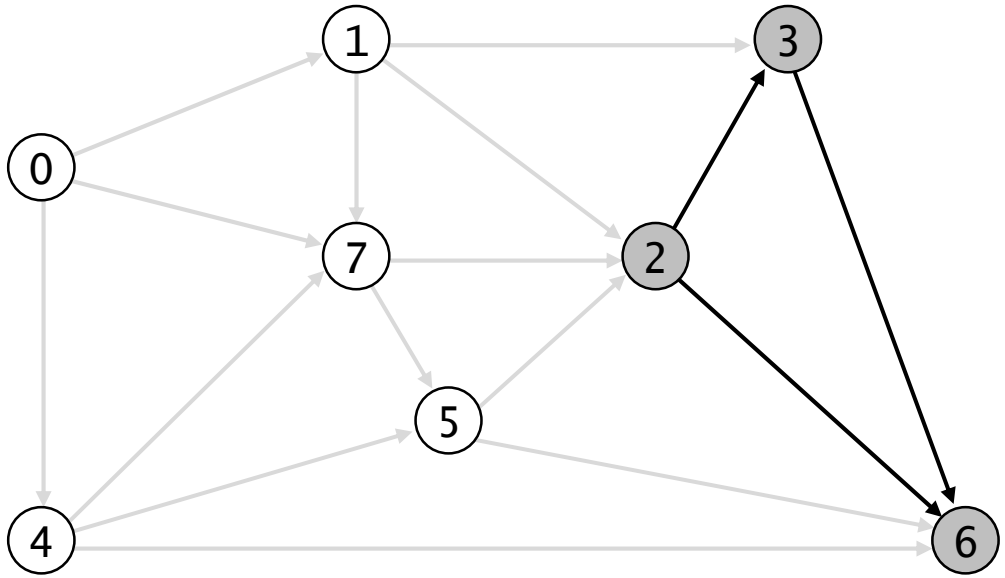


0 1 4 7 **5** **2** **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	~~14.0~~	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

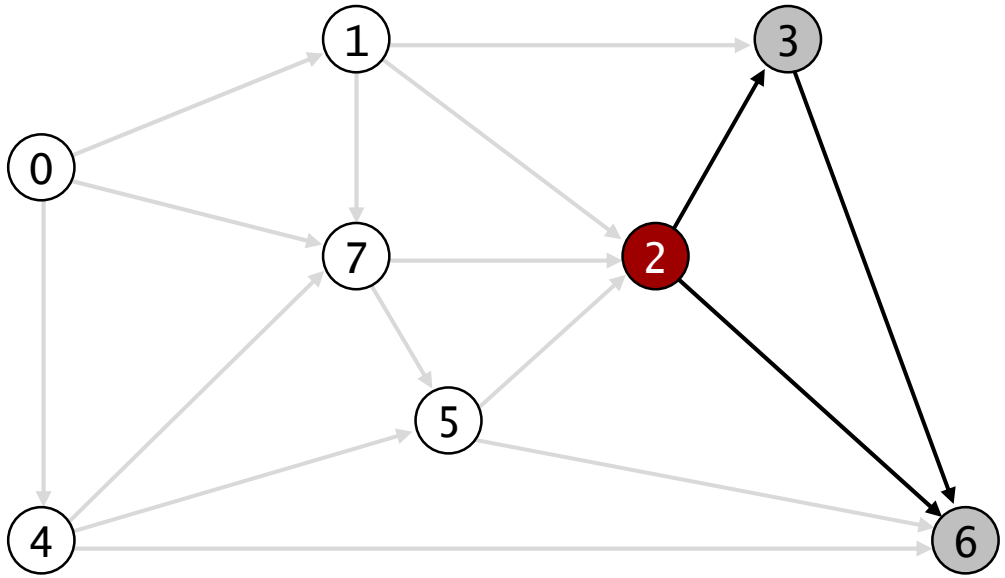


0 1 4 7 5 **2 3 6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

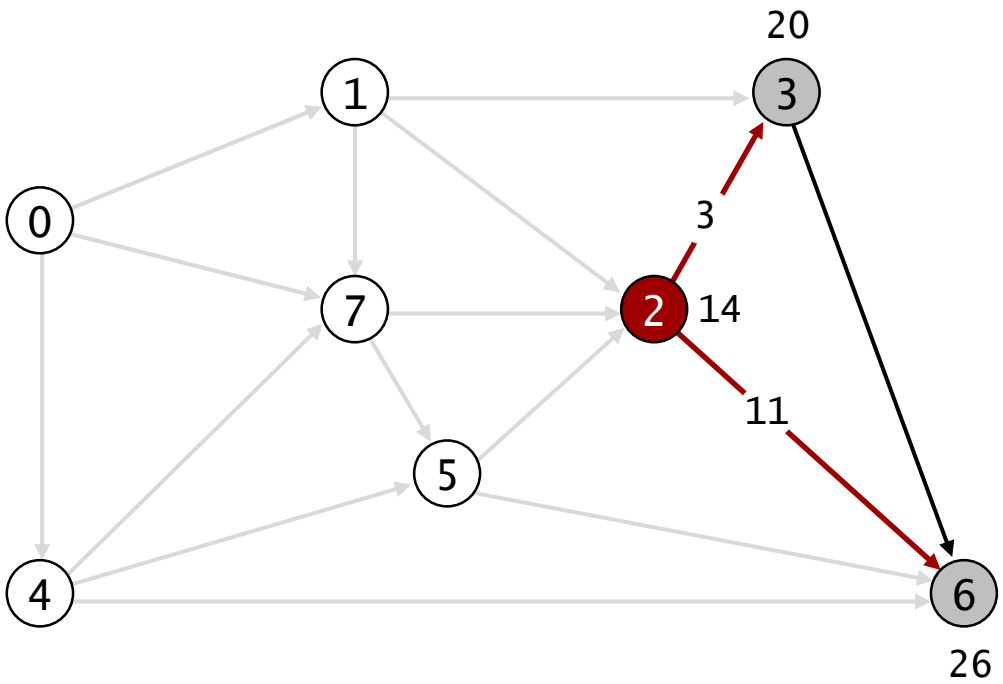


0 1 4 7 5 **2** 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	**14.0**	**5->2**
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

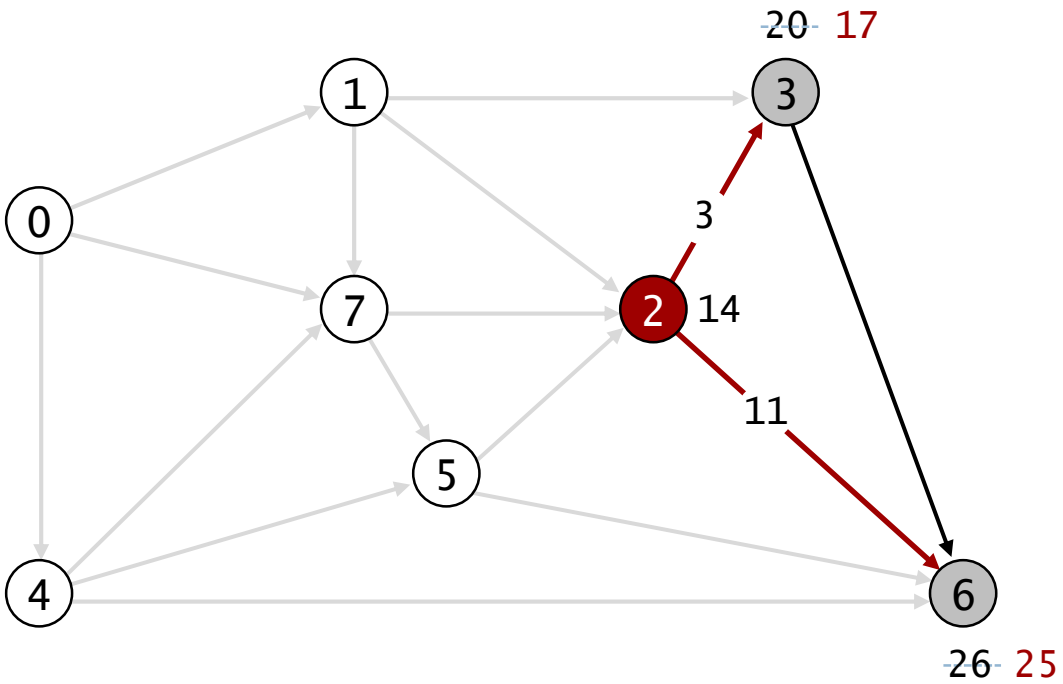


	0	1	4	7	5	2	3	6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
5	13.0	4->5
4	9.0	0->4
6	26.0	5->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

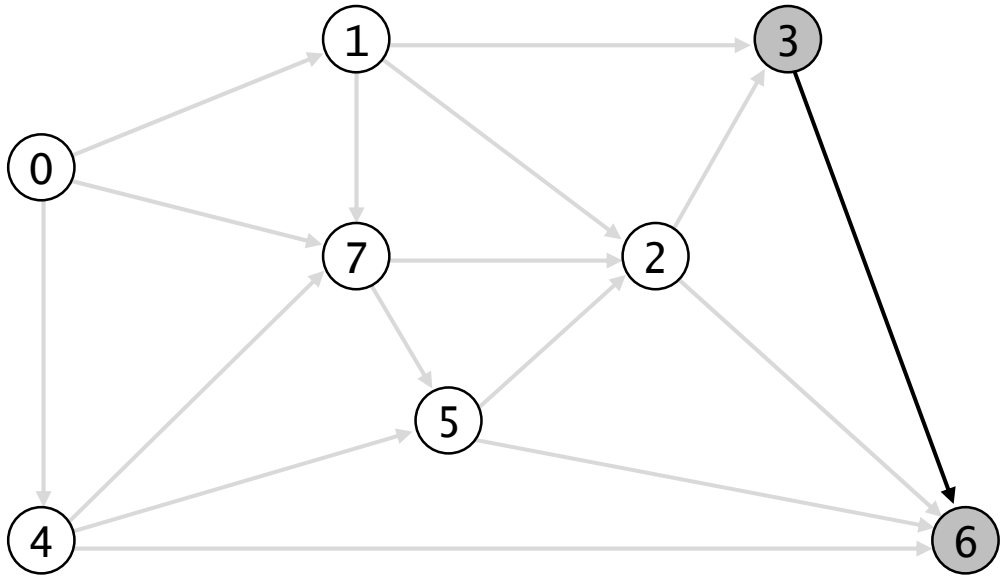


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

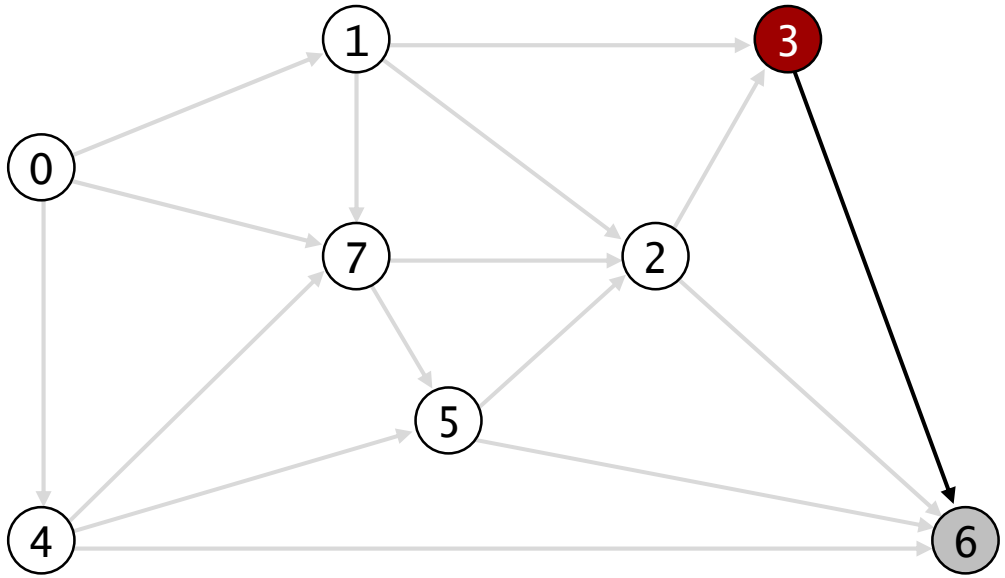


0 1 4 7 5 2 **3** **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

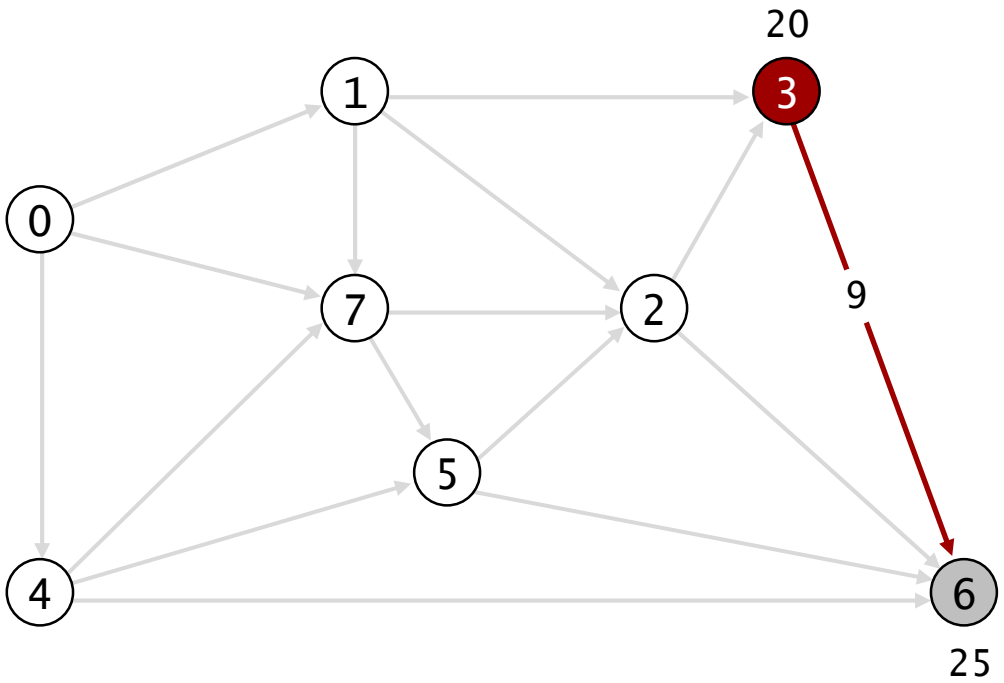


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

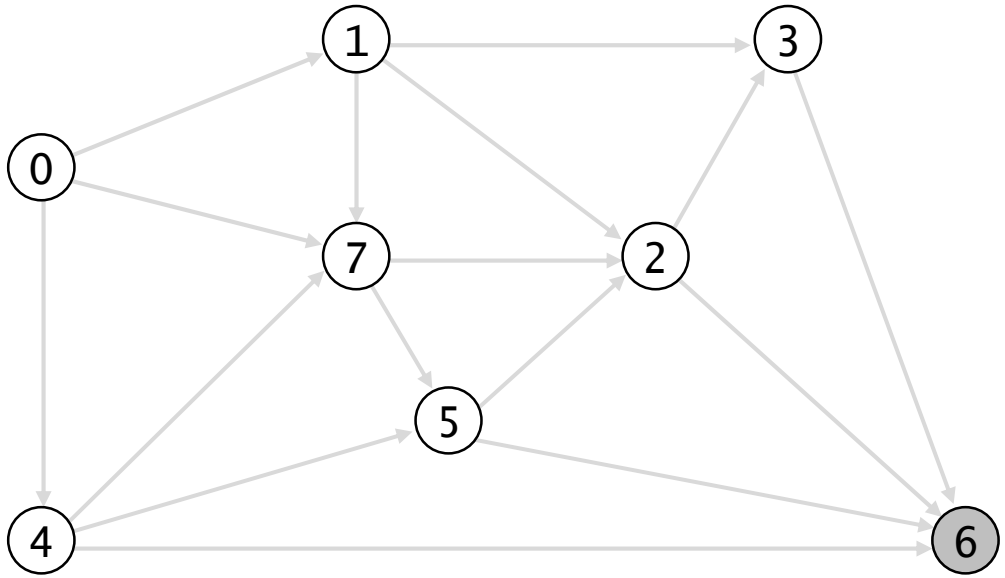


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

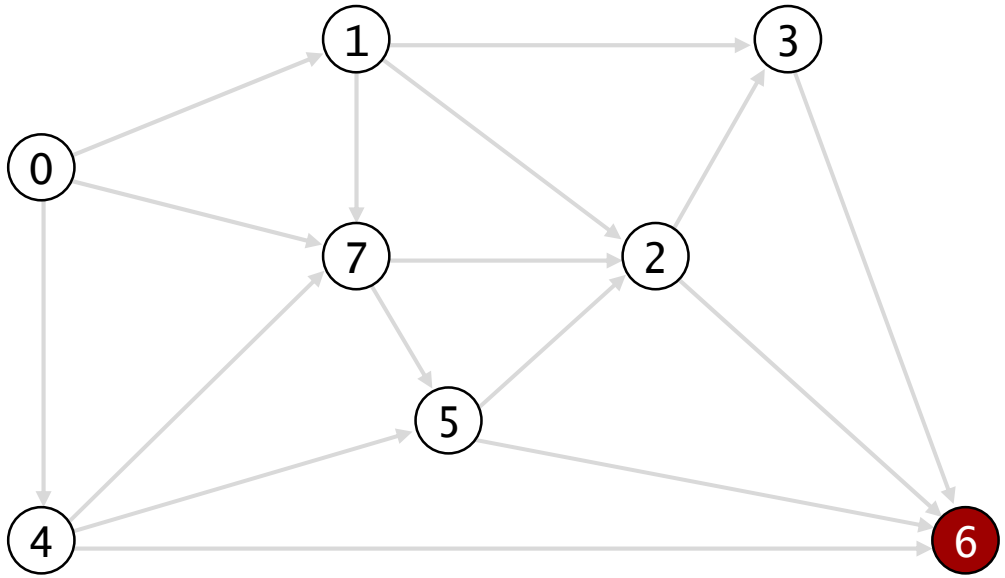


0 1 4 7 5 2 3 **6**

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

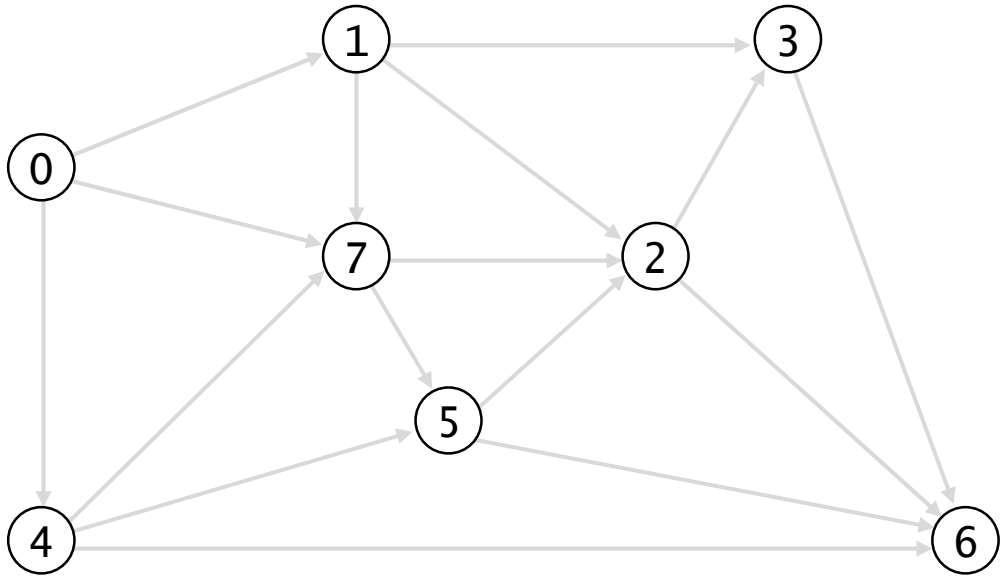


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
→ 4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.

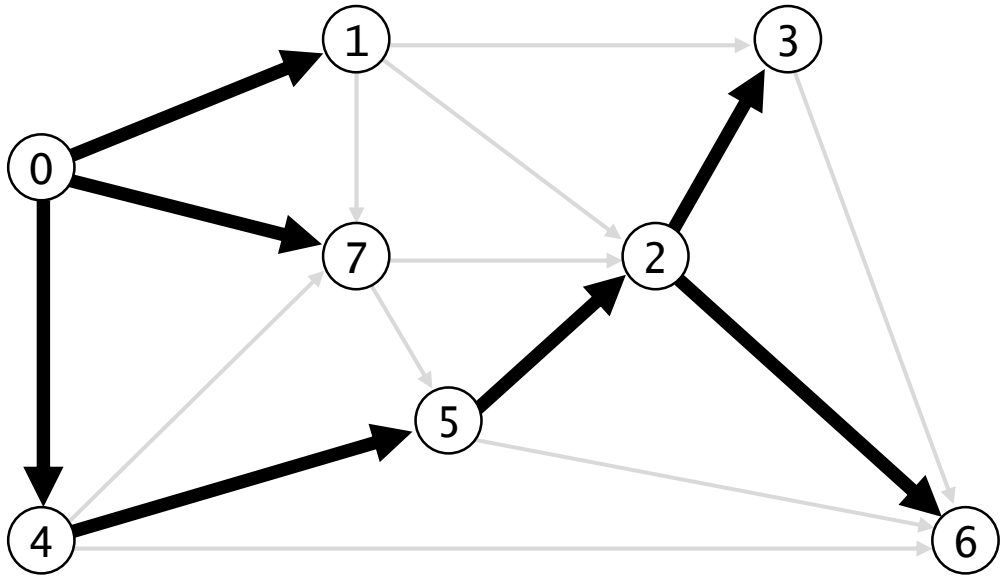


0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

گراف جهت‌دار بدون دور: اجرای نمایشی

- رئوس را به ترتیب توپولوژیکی در نظر بگیرید.
- در هر مرحله تمام یالهای خروجی از رأس انتخاب شده را راحت‌سازی کنید.



0 1 4 7 5 2 3 6

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
5	13.0	4->5
4	9.0	0->4
6	25.0	2->6
7	8.0	0->7

کوتاه‌ترین مسیرها در گراف جهت‌دار بدون دور: زمان اجرا

- گزاره. الگوریتم مرتب‌سازی توپولوژیکی در هر گراف جهت‌دار بدون دور، درخت کوتاه‌ترین مسیر را در زمانی متناسب با $E + V$ محاسبه می‌کند.

وزن یالها می توانند منفی باشند

- اثبات.
  - هر یال `v->w` = `e` دقیقاً یک بار راحت‌سازی می‌شود (در هنگام راحت‌سازی v)، که باعث می‌شود `distTo[w] ≤ distTo[v] + e.weight()`.
  - نامساوی بالا تا انتهای اجرای الگوریتم برقرار می‌ماند، زیرا:
    - `distTo[w]` هرگز افزایش نمی‌یابد،
    - `distTo[v]` نیز از این به بعد تغییر نمی‌کند،
  - در نتیجه پس از خاتمه‌ی اجرای الگوریتم، شرط بهینگی کوتاه‌ترین مسیرها برقرار خواهد بود.

کوتاه‌ترین مسیرها در گراف جهت‌دار بدون دور: پیاده‌سازی


```
public class AcyclicSP
{
    private DirectedEdge[] edgeTo;
    private double[] distTo;

    public AcyclicSP((EdgeWeightedDigraph G, int s)
    {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;
        Topological topological = new Topological(G);
        for (int v : topological.order())
            for (DirectedEdge e : G.adj(v))
                relax(e);

    }
}
```

تغییر هوشمندانه اندازه تصویر

- رگه‌گیری. [آویدان و شامیر] تغییر اندازه یک تصویر بدون تخریب آن به منظور نمایش در تلفن‌های همراه و مرورگرهای وب.

تغییر هوشمندانه اندازه تصویر

- رگه‌گیری. [آویدان و شامیر] تغییر اندازه یک تصویر بدون تخریب آن به منظور نمایش در تلفن‌های همراه و مرورگرهای وب.

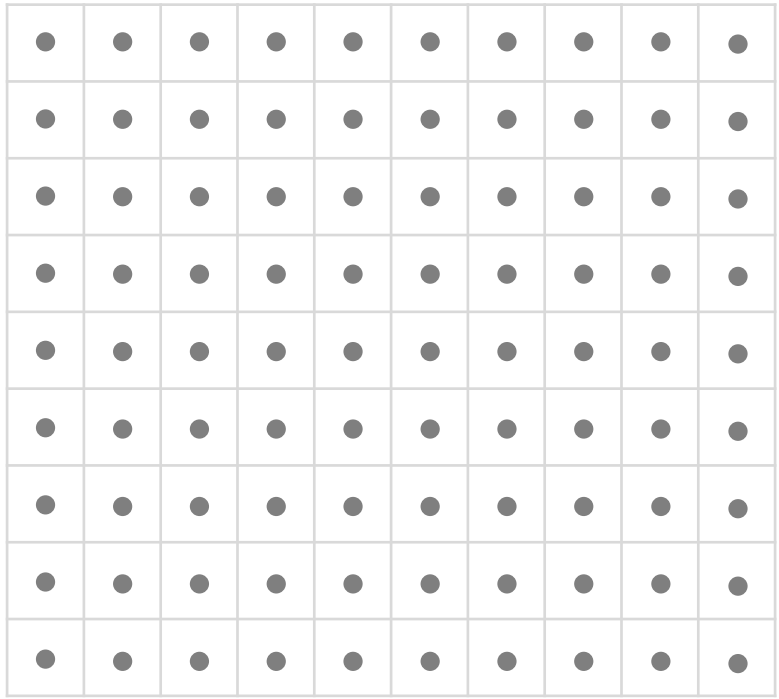
www.youtube.com/watch?v=6NcIJXTlugc

تغییر هوشمندانه اندازه تصویر

- برای یافتن رگه‌های عمودی:
  - DAG توری: رأس = پیکسل؛ یال = از پیکسل به سه پیکسل همسایه‌ی پایینی؛
  - وزن پیکسل = یک تابع انرژی از هشت پیکسل همسایه؛
  - رگه = یک کوتاه‌ترین مسیر از بالا به پایین.

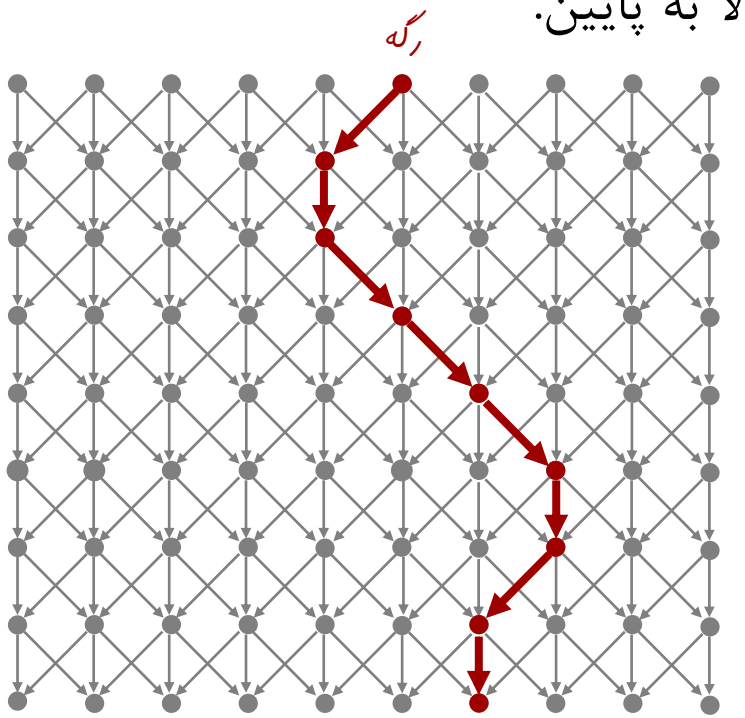
تغییر هوشمندانه اندازه تصویر

- برای یافتن رگه‌های عمودی:
  - DAG توری: رأس = پیکسل؛ یال = از پیکسل به سه پیکسل همسایه‌ی پایینی؛
  - وزن پیکسل = یک تابع انرژی از هشت پیکسل همسایه؛
  - رگه = یک کوتاه‌ترین مسیر از بالا به پایین.



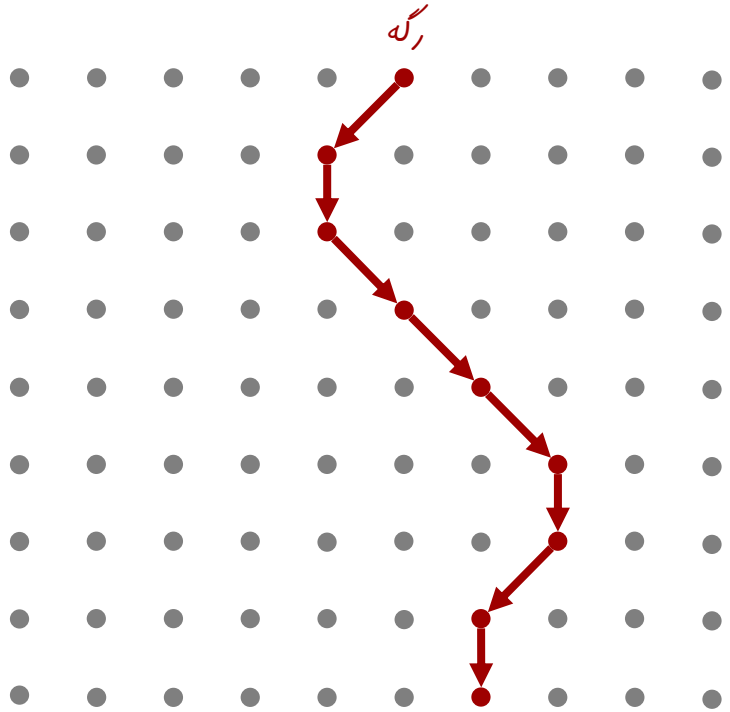
تغییر هوشمندانه اندازه تصویر

- برای یافتن رگه‌های عمودی:
  - DAG توری: رأس = پیکسل؛ یال = از پیکسل به سه پیکسل همسایه‌ی پایینی؛
  - وزن پیکسل = یک تابع انرژی از هشت پیکسل همسایه؛
  - رگه = یک کوتاه‌ترین مسیر از بالا به پایین.



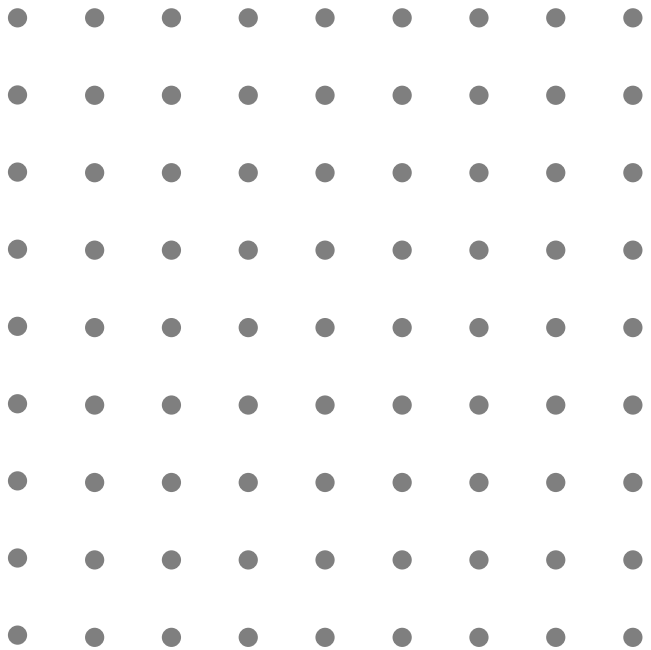
تغییر هوشمندانه اندازه تصویر

- برای حذف رگه‌های عمودی:
  - پیکسل‌های متعلق به رگه را حذف کن (هر سطر یک پیکسل)



تغییر هوشمندانه اندازه تصویر

- برای حذف رگه‌های عمودی:
  - پیکسل‌های متعلق به رگه را حذف کن (هر سطر یک پیکسل)



محاسبه طولانی ترین مسیرها در DAG

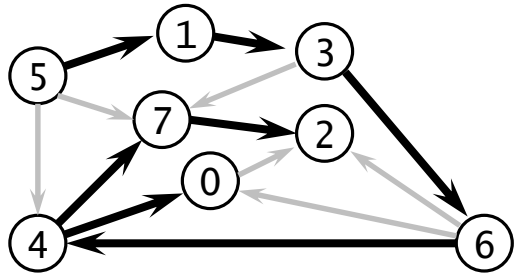
- تبدیل به مسئله‌ی محاسبه‌ی کوتاه‌ترین مسیرها.
  - وزن تمام یالها را منفی کن
  - کوتاهترین مسیرها را محاسبه کن
  - در مسیرهای محاسبه شده وزنها را منفی کن

ورودی کوتاه‌ترین مسیر

```
5->4 -0.35
4->7 -0.37
5->7 -0.28
5->1 -0.32
4->0 -0.38
0->2 -0.26
3->7 -0.39
1->3 -0.29
7->2 -0.34
6->2 -0.40
3->6 -0.52
6->0 -0.58
6->4 -0.93
```

ورودی طولانی‌ترین مسیر

```
5->4  0.35
4->7  0.37
5->7  0.28
5->1  0.32
4->0  0.38
0->2  0.26
3->7  0.39
1->3  0.29
7->2  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```

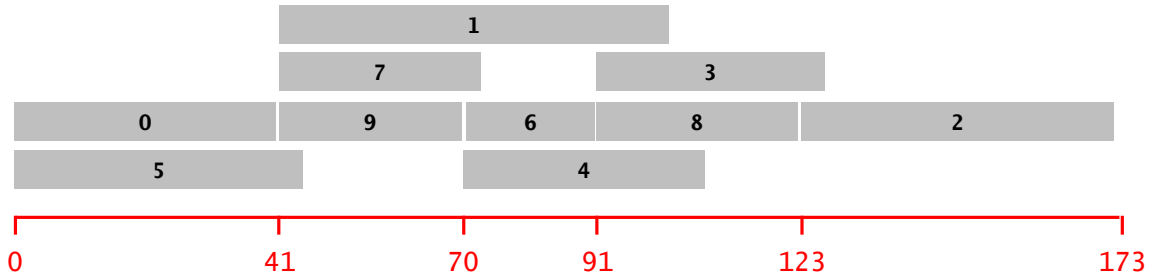


- نکته کلیدی. الگوریتم مرتب‌سازی توپولوژیکی حتی با وجود یال با وزن منفی درست کار می‌کند.

محاسبه‌ی طولانی‌ترین مسیرها در DAG: کاربرد

- زمانبندی موازی کارها. یک مجموعه از کارها به همراه مدت زمان انجام هر یک و همچنین اولویت آنها داده شده است. با رعایت اولویت‌های داده شده، این کارها را به گونه‌ای زمانبندی کنید که زمان اتمام کارها به حداقل برسد.

job	*duration*	*must complete before*		
0	41.0	1	7	9
1	51.0	2		
2	50.0			
3	36.0			
4	38.0			
5	45.0			
6	21.0	3	8	
7	32.0	3	8	
8	32.0	2		
9	29.0	4	6	

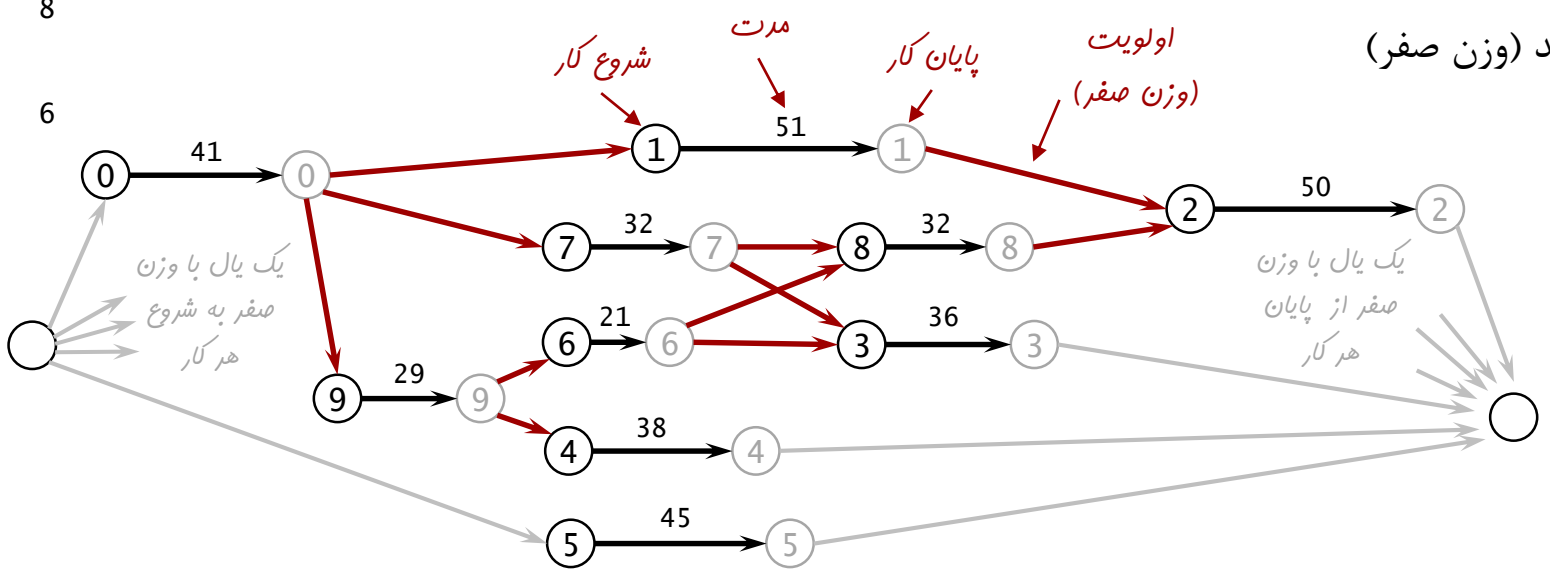


راه‌حل زمان‌بندی موازی کارها

روش مسیر بحرانی

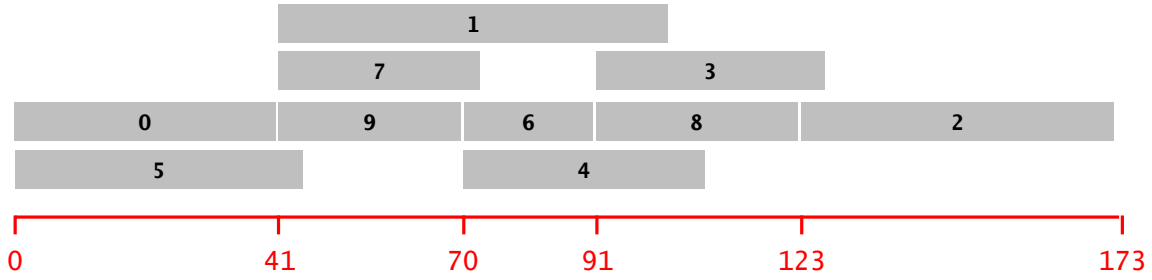
job	duration	must complete before		
0	41.0	1	7	9
1	51.0	2		
2	50.0			
3	36.0			
4	38.0			
5	45.0			
6	21.0	3	8	
7	32.0	3	8	
8	32.0	2		
9	29.0	4	6	

- روش مسیر بحرانی. یک DAG وزن‌دار به صورت زیر ایجاد کن:
  - رأس منبع و مقصد
  - دو رأس به ازای هر کار (شروع و پایان کار)
  - سه یال به ازای هر کار:
    - شروع به پایان (وزن برابر با مدت انجام کار)
    - منبع به شروع (وزن صفر)
    - پایان به مقصد (وزن صفر)

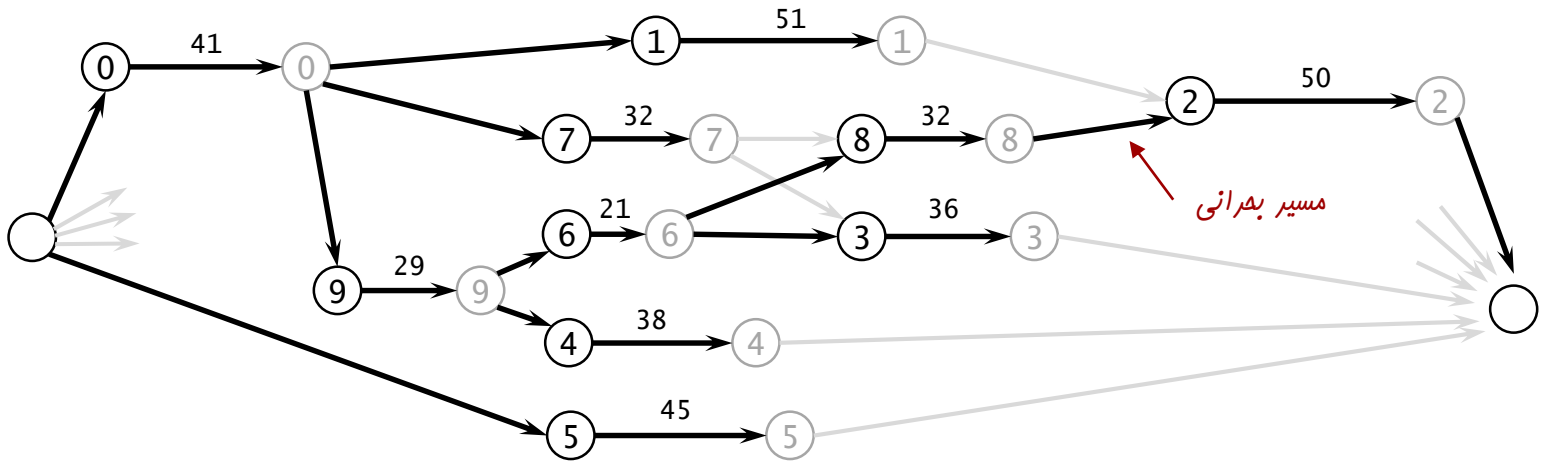


روش مسیر بحرانی

- روش مسیر بحرانی. از طولانی‌ترین مسیر از منبع برای زمانبندی کارها استفاده کن.



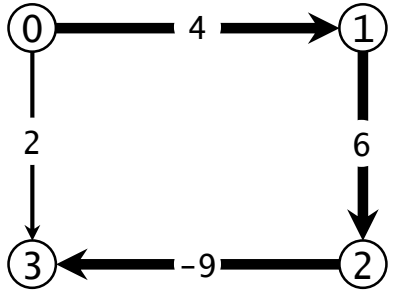
راه حل زمان‌بندی موازی کارها



یالهای با وزن منفی

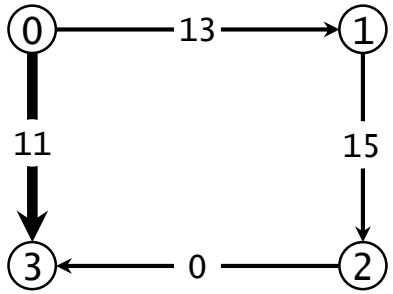
کوتاه‌ترین مسیرها با وجود یالهای با وزن منفی

- دایکسترا. برای گراف‌هایی که دارای یالهایی با وزن منفی هستند کار نمی‌کند.



دایکسترا بلافاصله پس از رأس 0 رأس 3 را انتخاب می‌کند.
اما کوتاه‌ترین مسیر از 0 به 3 مسیر 3->2->1->0 است.

- وزن‌دهی مجدد. اضافه کردن یک مقدار ثابت به وزن همه‌ی یالها کار نمی‌کند.



اضافه کردن 9 به وزن همه یالها کوتاه‌ترین مسیر را از
3->2->1->0 به 3->0 تغییر می دهد.

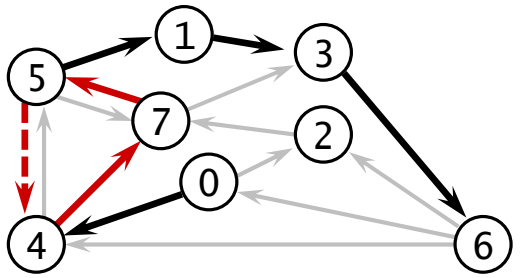
- نتیجه گیری. به یک الگوریتم متفاوت نیاز داریم.

کوتاه‌ترین مسیرها در گراف‌های جهت‌دار و وزن‌دار

- تعریف. یک دور منفی یک دور جهت‌دار است که مجموع وزن یالهای آن منفی است.

گراف جهت‌دار وزن‌دار

```
4->5  0.35
5->4 -0.66
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```



دور منفی (0.28 + 0.37 + -0.66)

```
5->4->7->5
```

کوتاه‌ترین مسیر از 0 به 6

```
0->4->7->5->4->7->5 ... -> 1->3->6
```

- گزاره. یک درخت کوتاه‌ترین مسیر وجود دارد اگر و فقط اگر دور منفی وجود نداشته باشد.

الگوریتم بلمن-فورد

Bellman-Ford Algorithm

Initialize `distTo[s] = 0` and `distTo[v] = ∞` for all other vertices

Repeat V times:
- Relax each edge

```
for (int pass = 0; pass < G.V(); pass++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
         relax(e);
```

گذر i (همه‌ی یالها را relax کن)

الگوریتم بلمن-فورد

Bellman-Ford Algorithm

Initialize `distTo[s] = 0` and `distTo[v] = ∞` for all other vertices

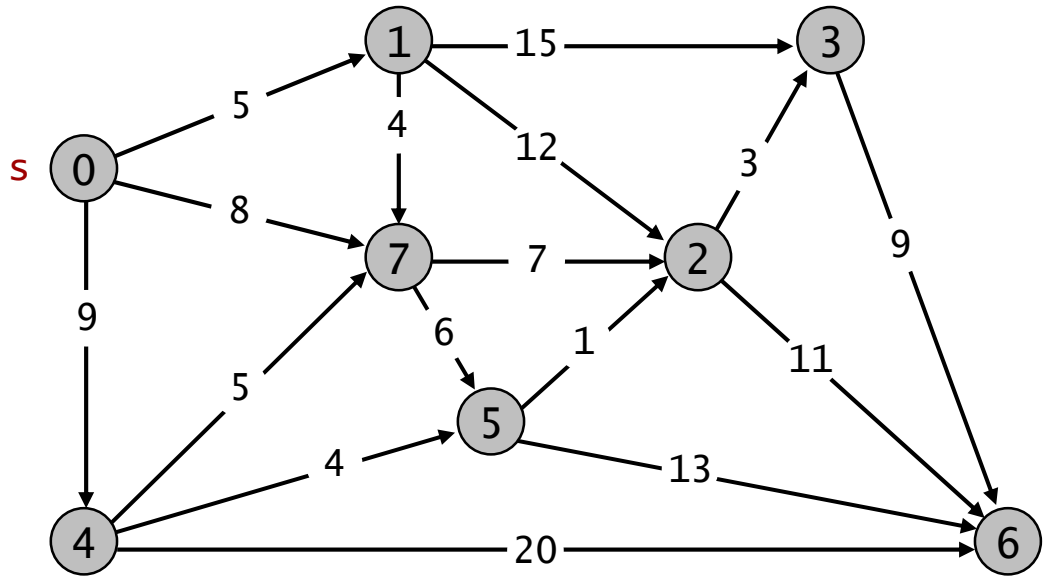
Repeat V times:
- Relax each edge

```
for (int pass = 0; pass < G.V(); pass++)
   for (DirectedEdge e : G.edges())
         relax(e);
```

گذر i (همه‌ی یالها را relax کن)

الگوریتم بلمن-فورد: اجرای نمایشی

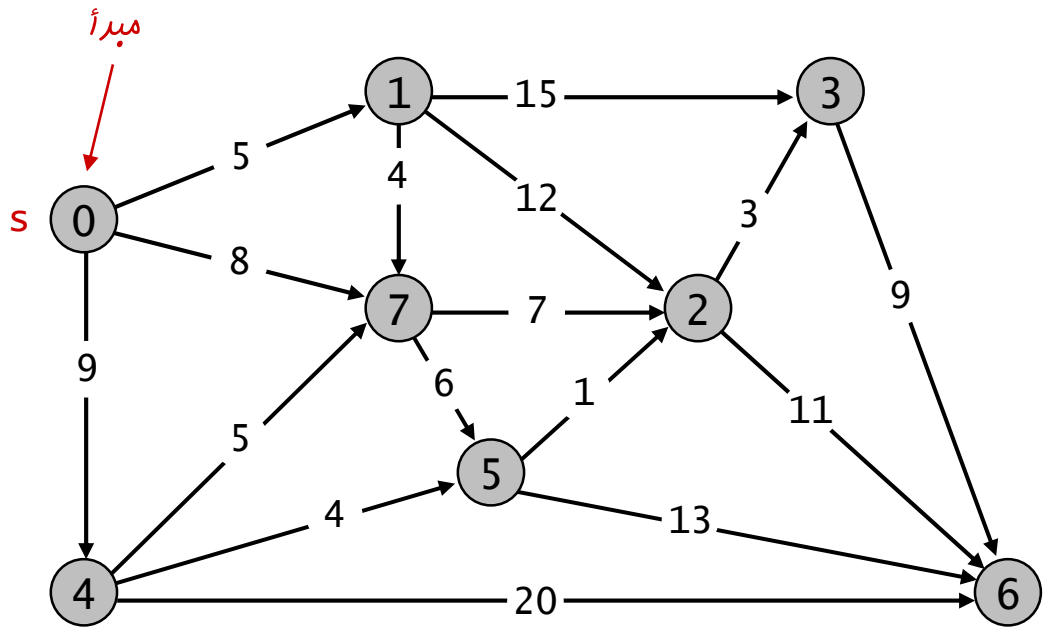
۷ مرتبه همه‌ی یالها را relax کن.




```
0->1    5.0
0->4    9.0
0->7    8.0
1->2   12.0
1->3   15.0
1->7    4.0
2->3    3.0
2->6   11.0
3->6    9.0
4->5    4.0
4->6   20.0
4->7    5.0
5->2    1.0
5->6   13.0
7->5    6.0
7->2    7.0
```

الگوریتم بلمن-فورد: اجرای نمایشی

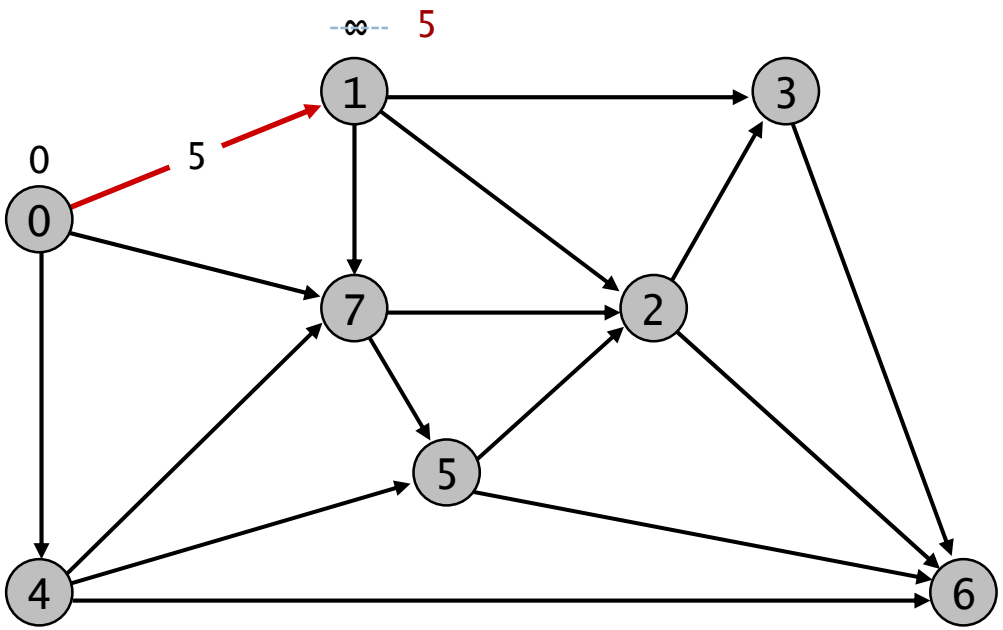
مقداردهی اولیه.



v	distTo[]	edgeTo[]
0	0.0	-
1	∞	-
2	∞	-
3	∞	-
4	∞	-
5	∞	-
6	∞	-
7	∞	-

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



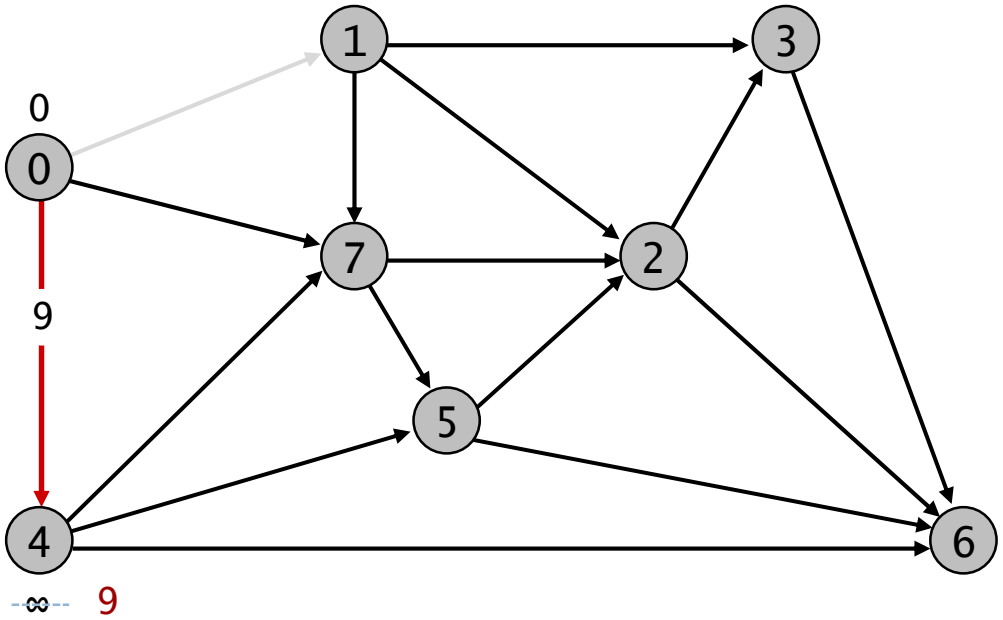
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
4		
5		
6		
7		

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



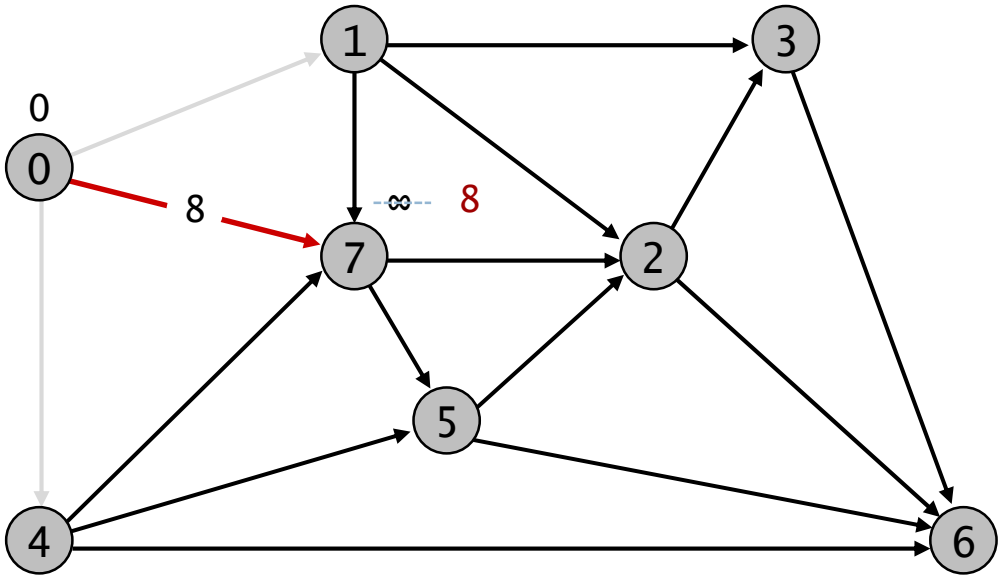
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
4	9.0	0->4
5		
6		
7		

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



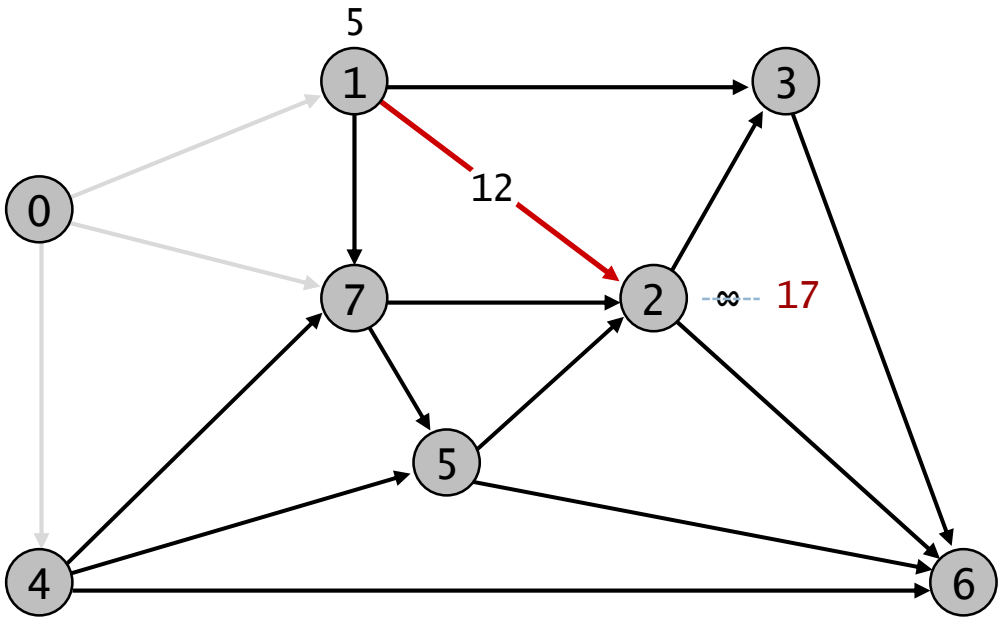
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2		
3		
4	9.0	0->4
5		
6		
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



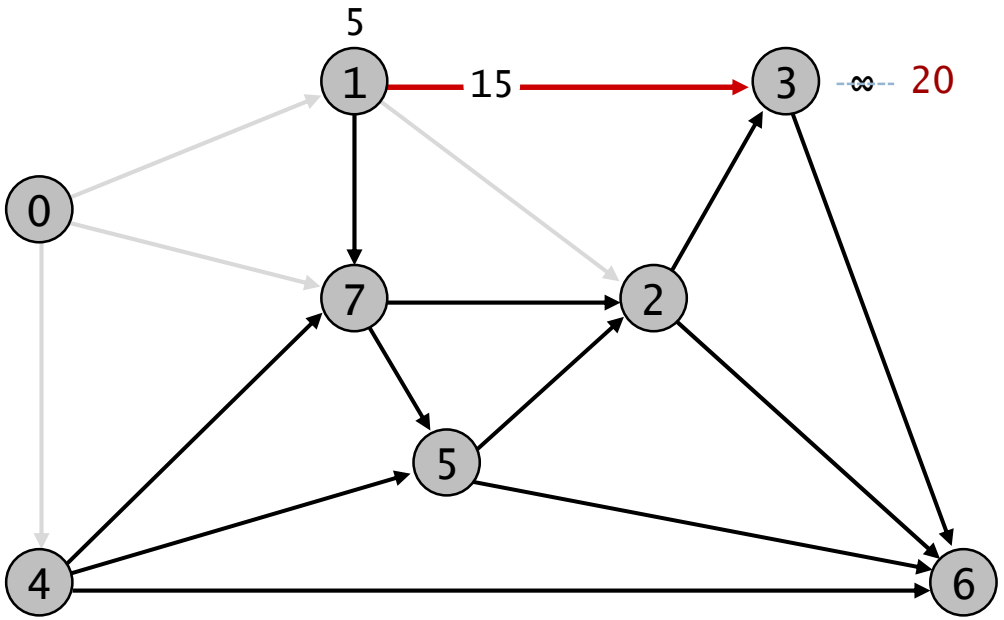
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3		
4	9.0	0->4
5		
6		
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



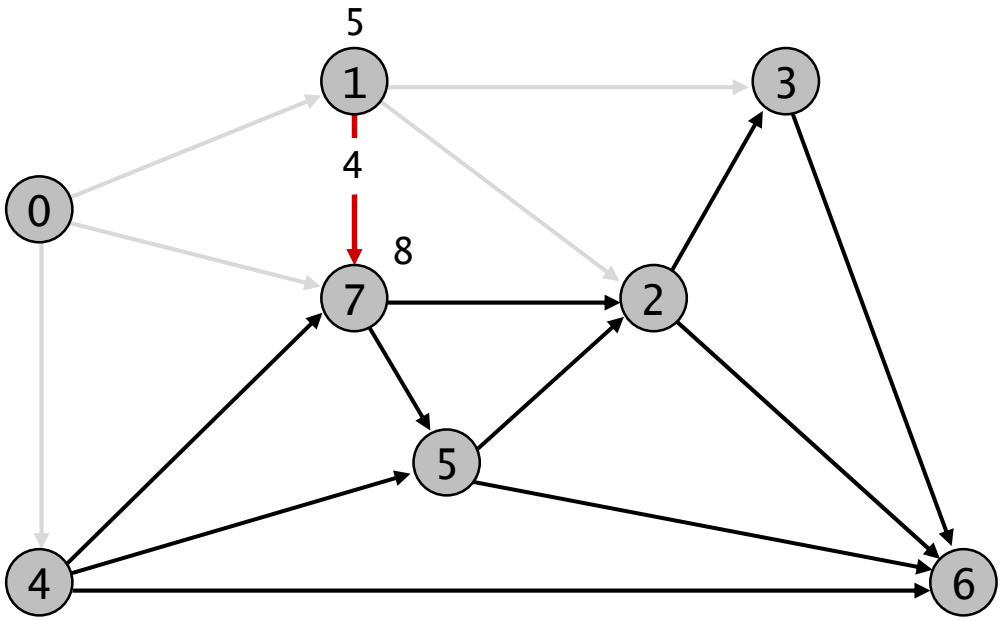
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5		
6		
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



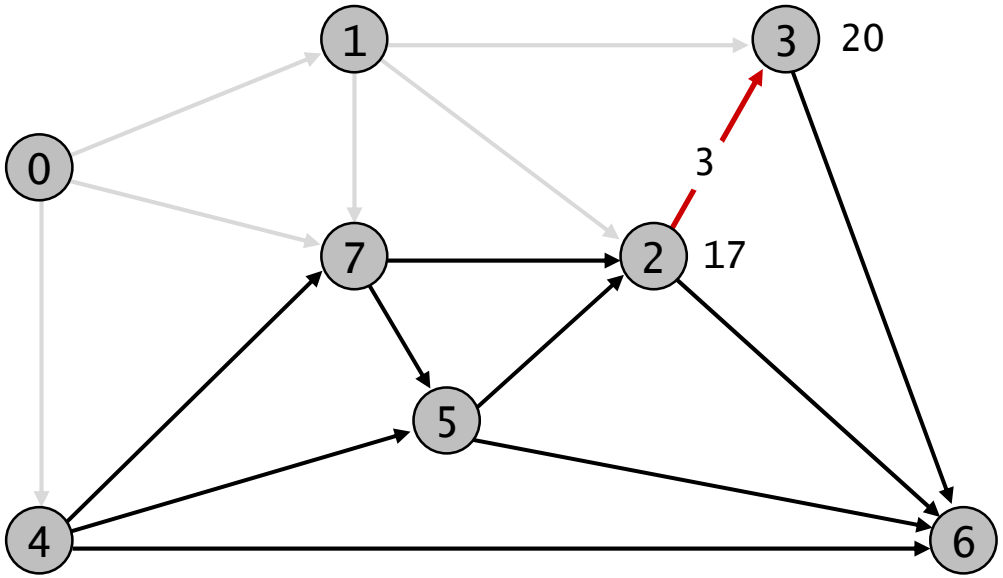
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5		
6		
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



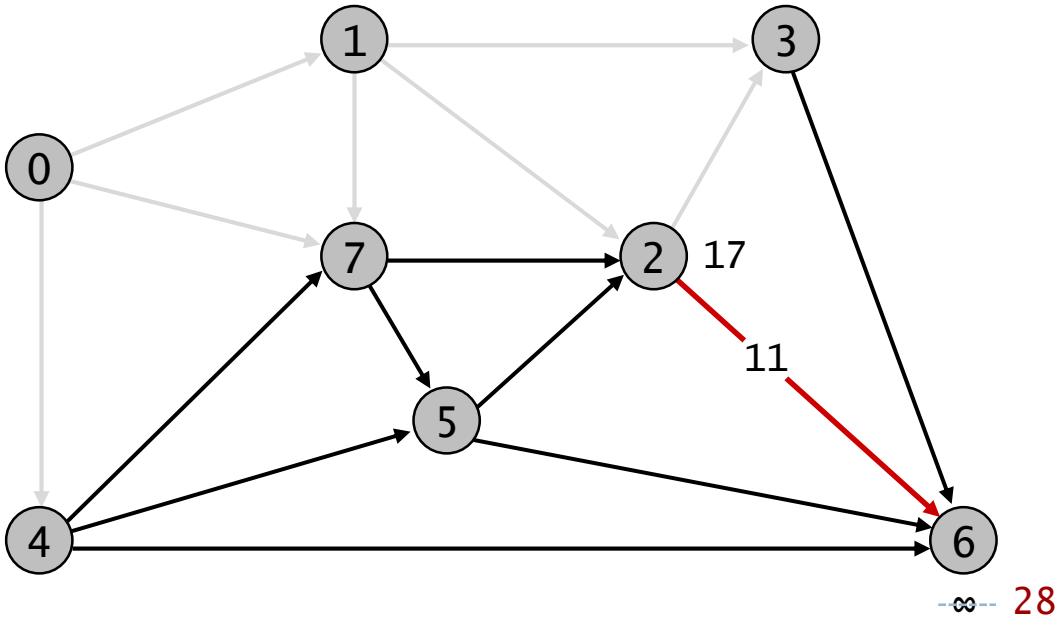
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5		
6		
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



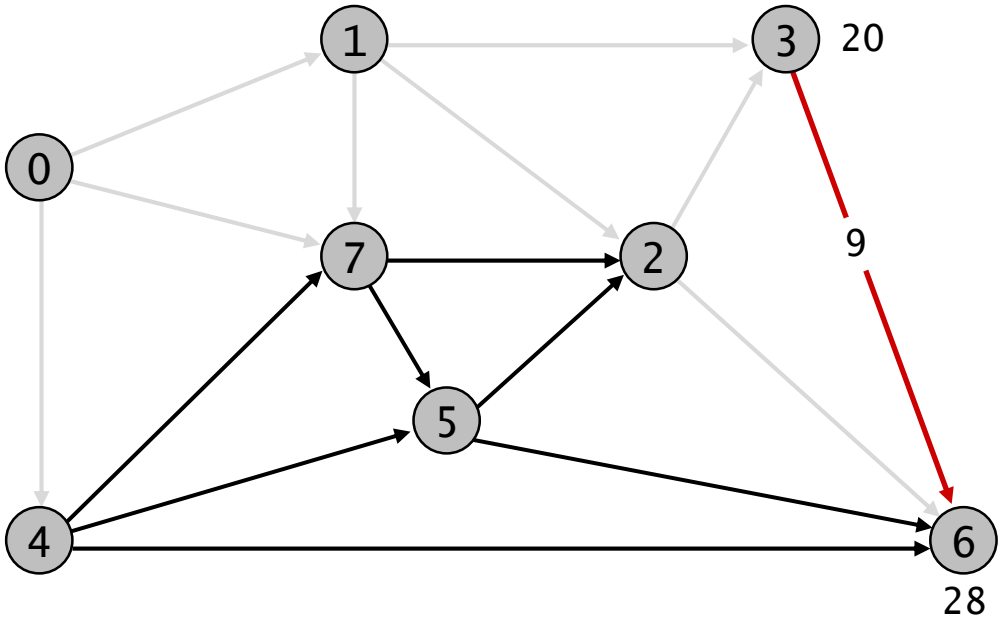
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5		
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



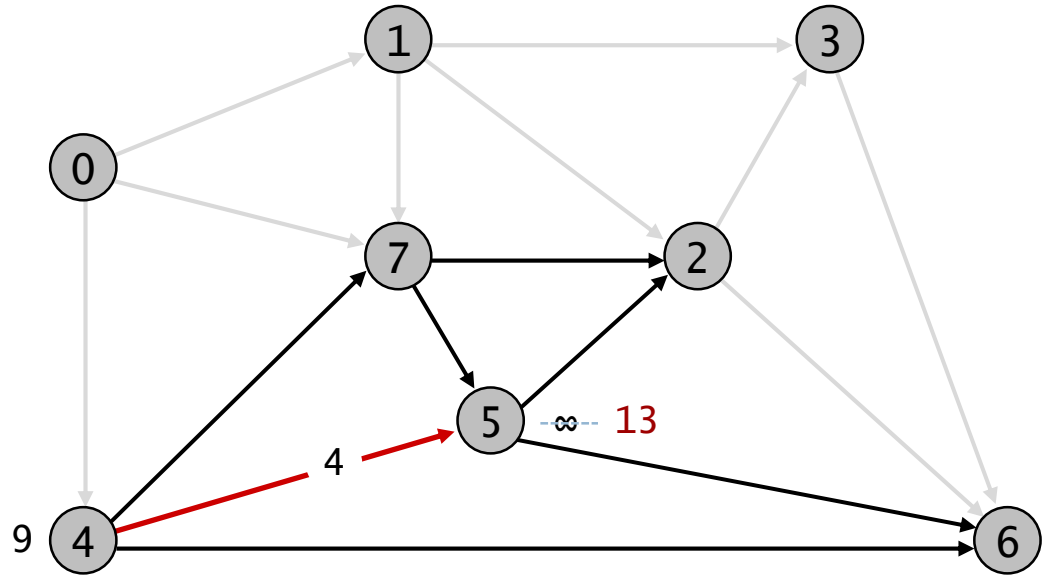
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5		
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



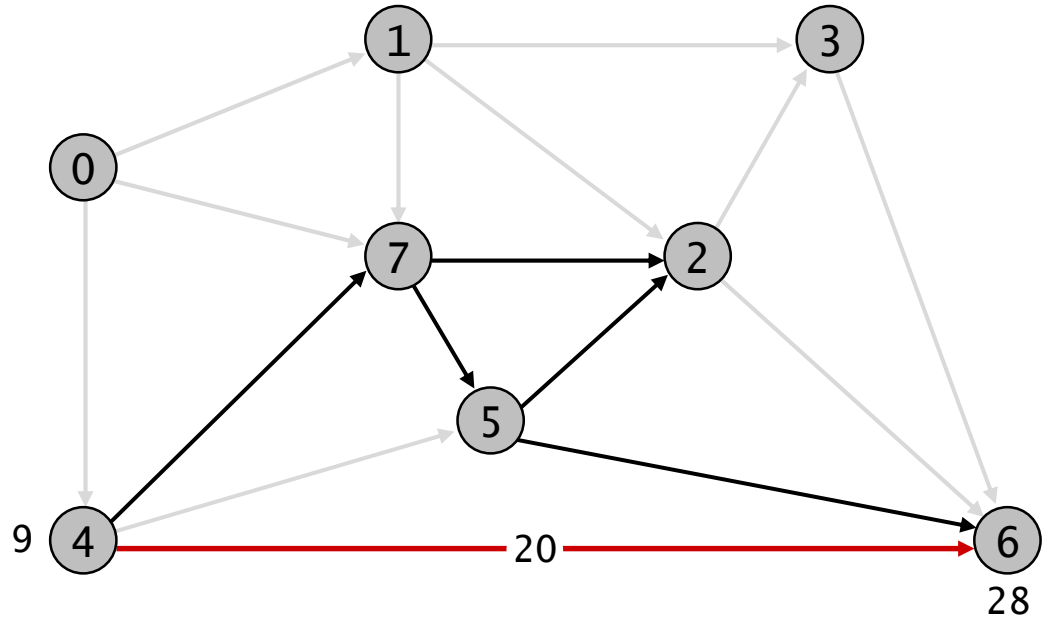
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



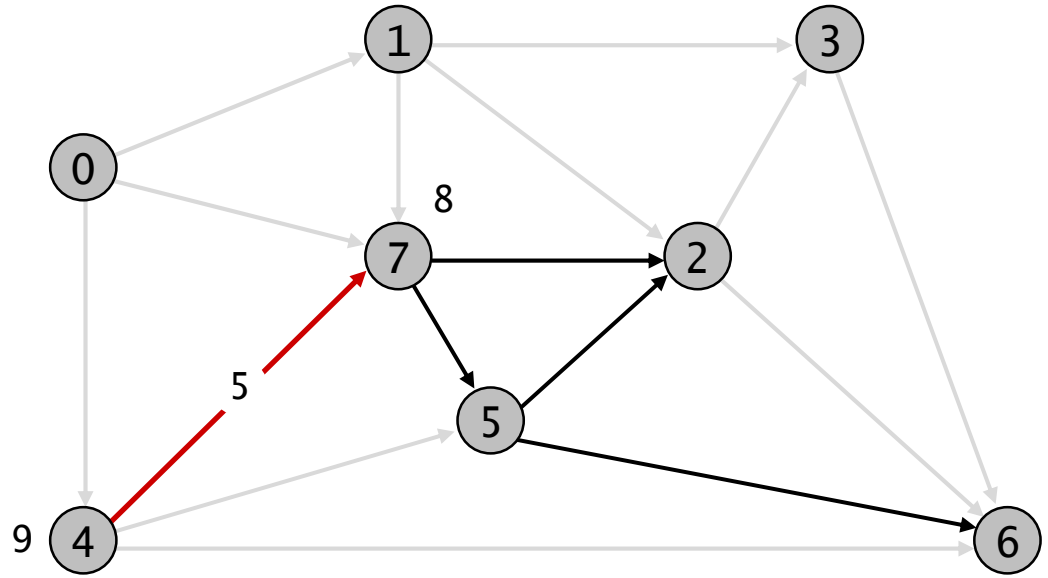
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



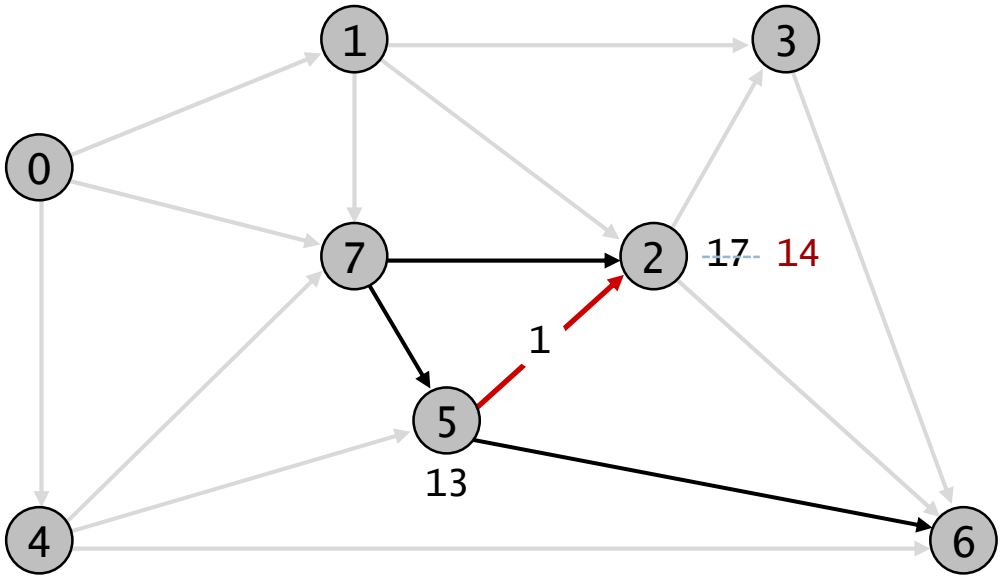
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	17.0	1->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



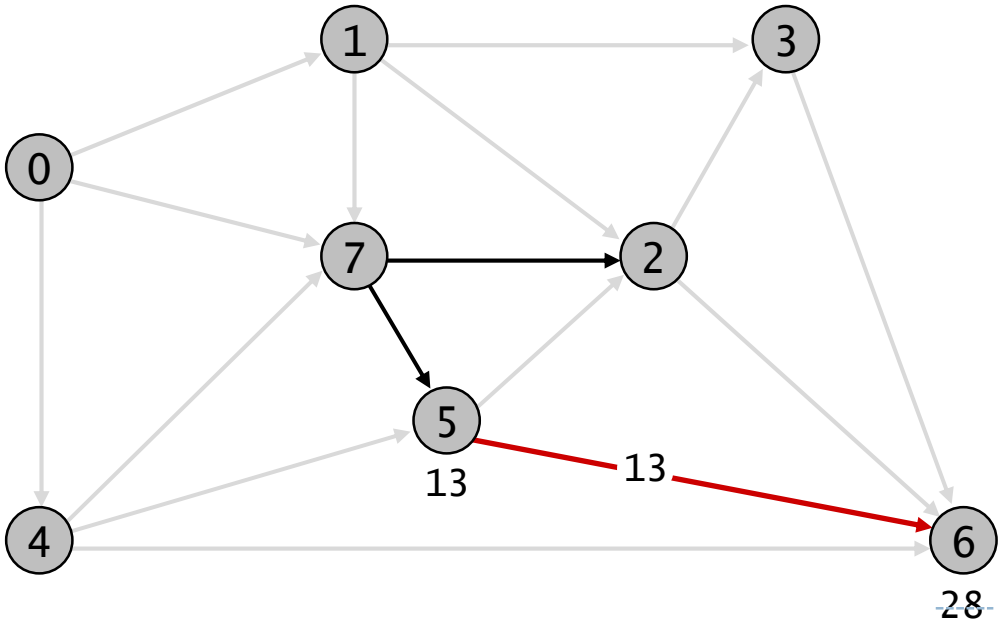
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	28.0	2->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



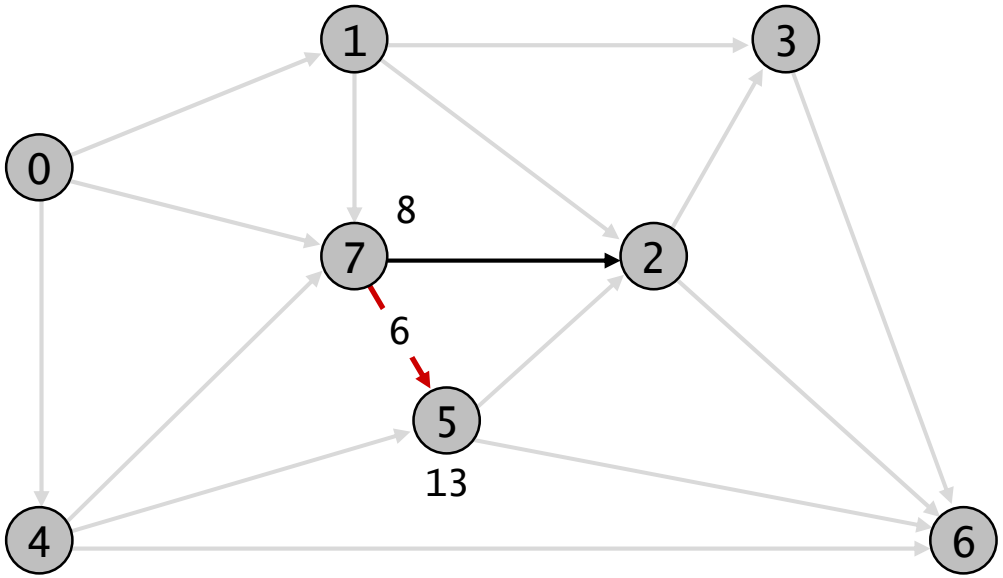
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



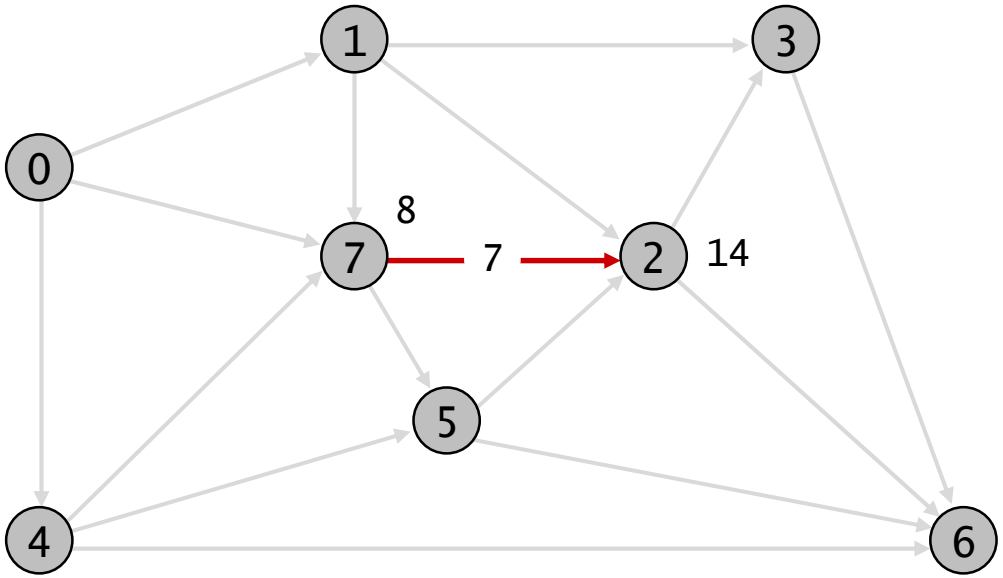
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



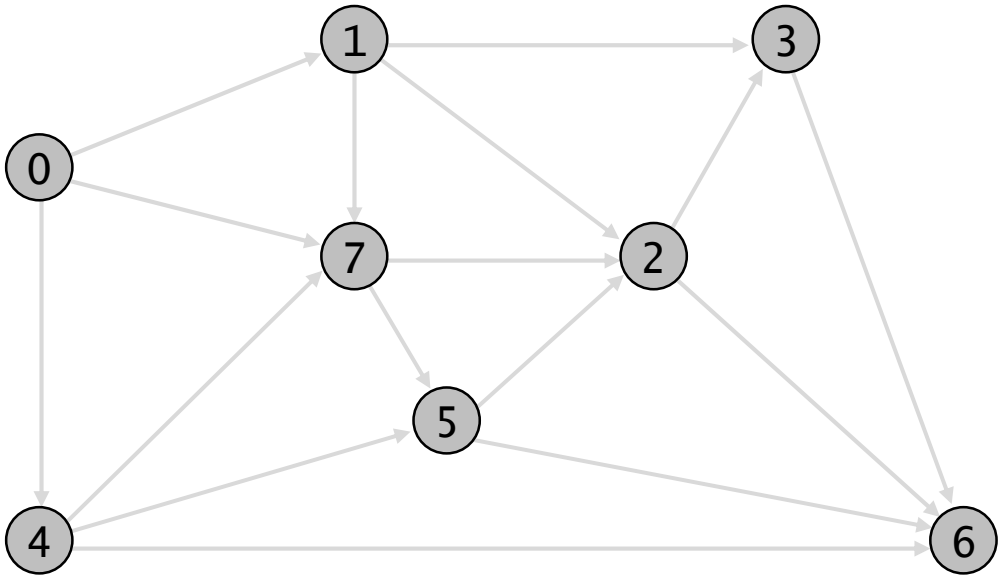
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



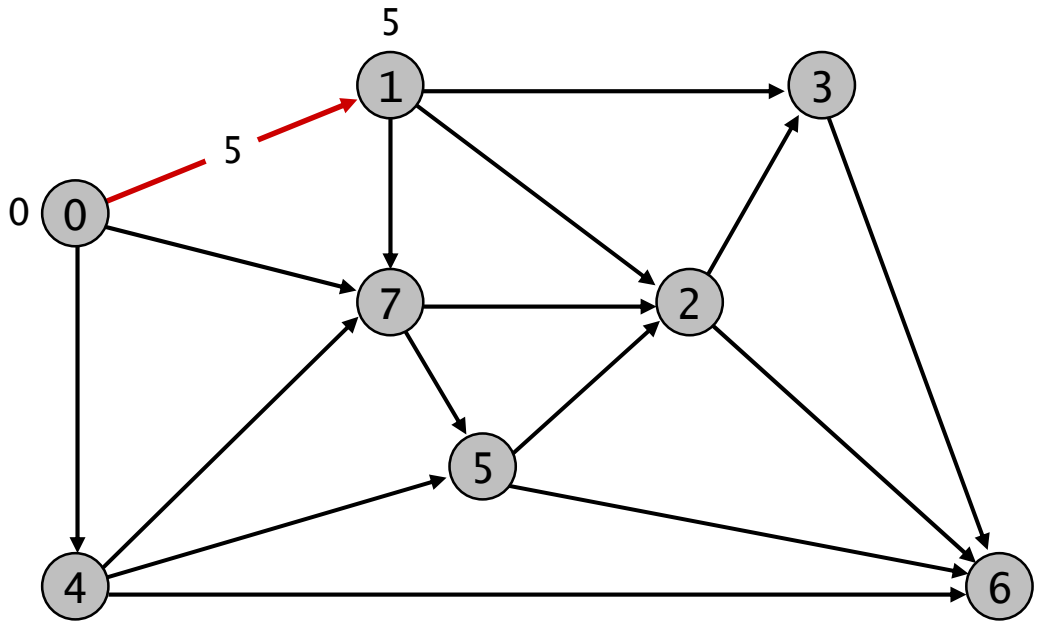
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 0

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



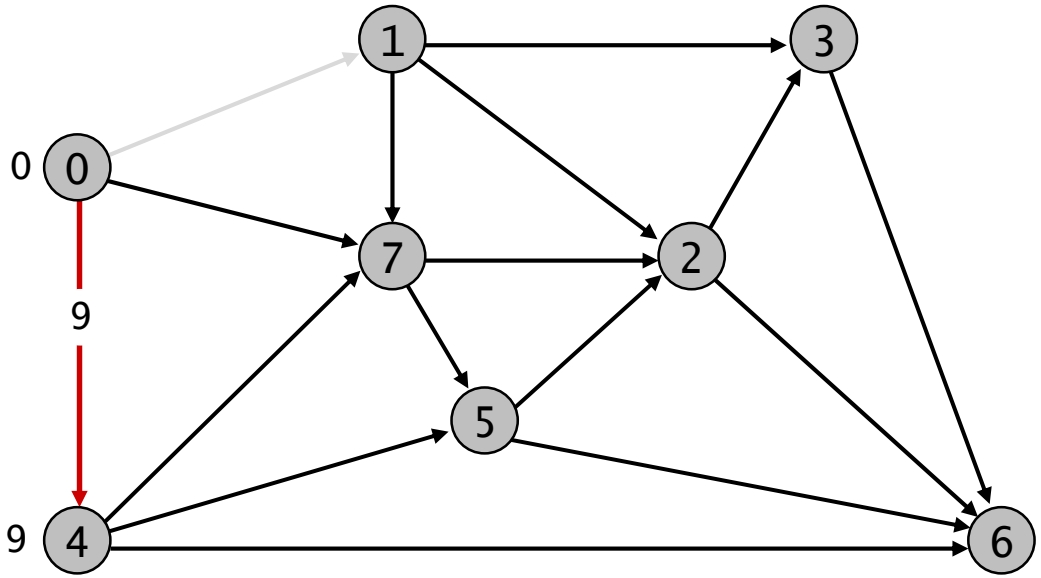
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

۷ مرتبه همه‌ی یالها را relax کن.



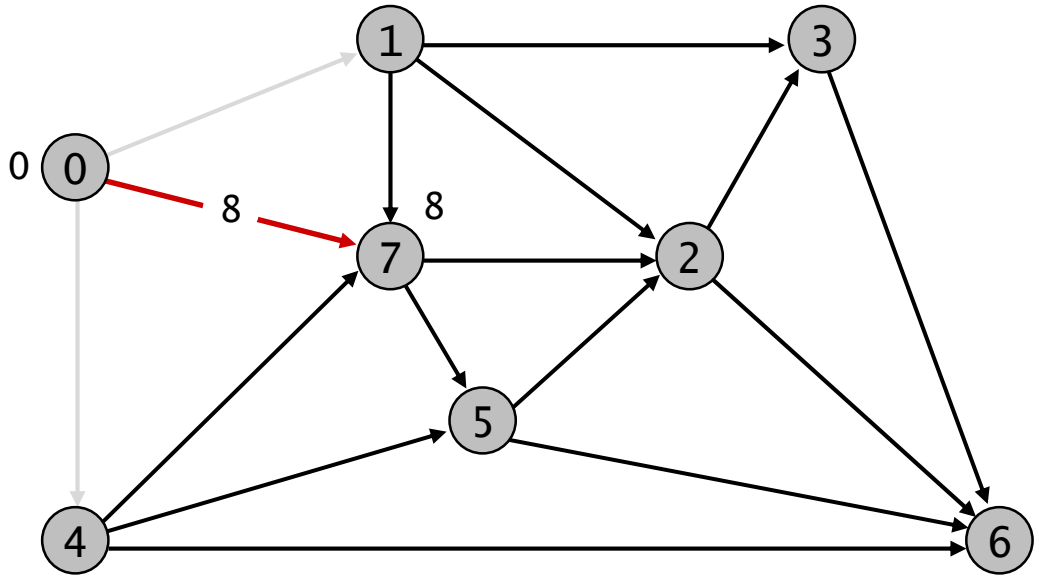
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



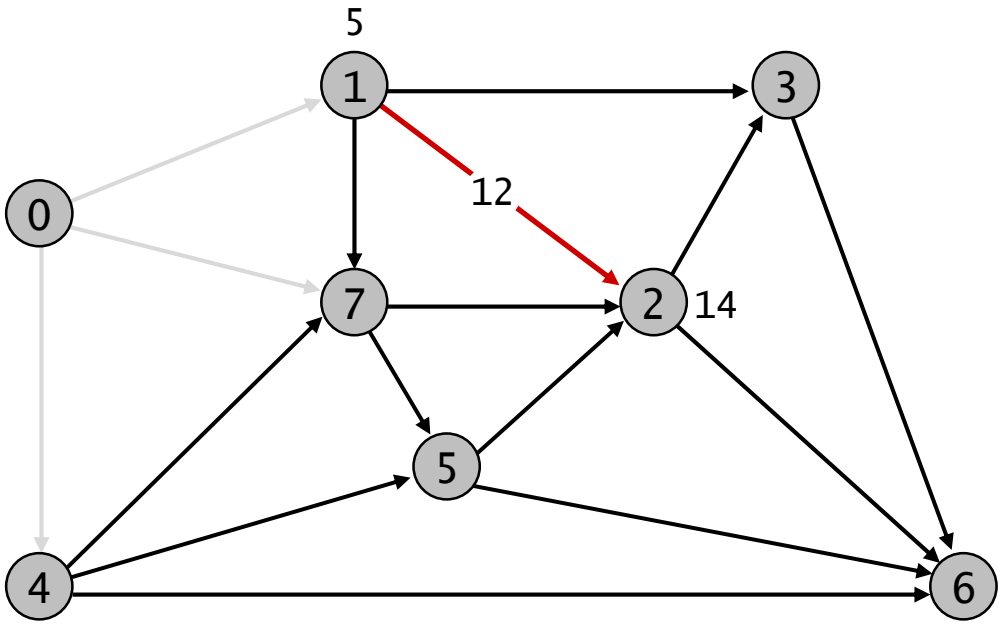
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



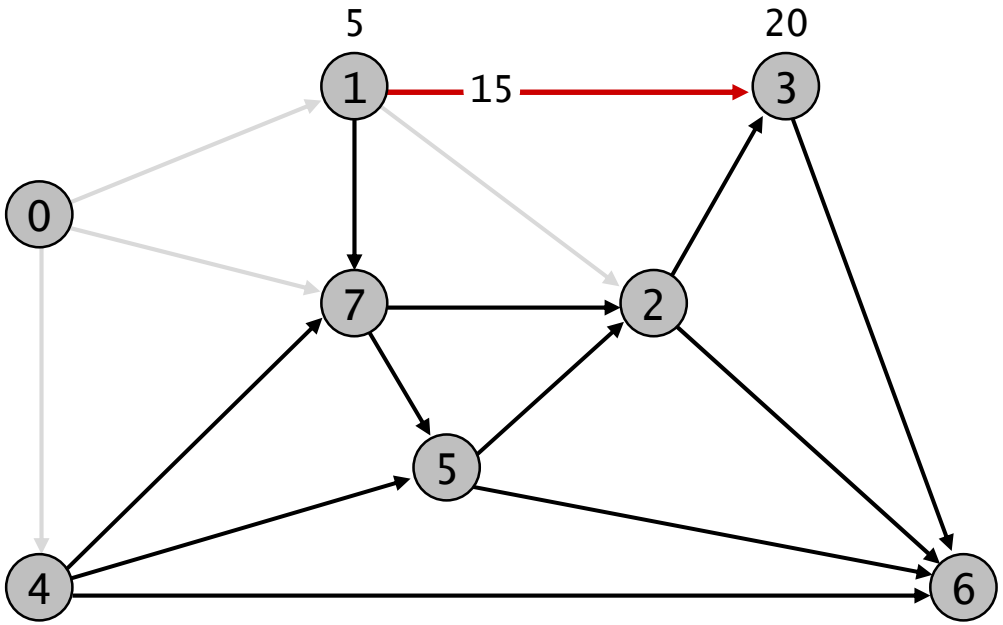
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



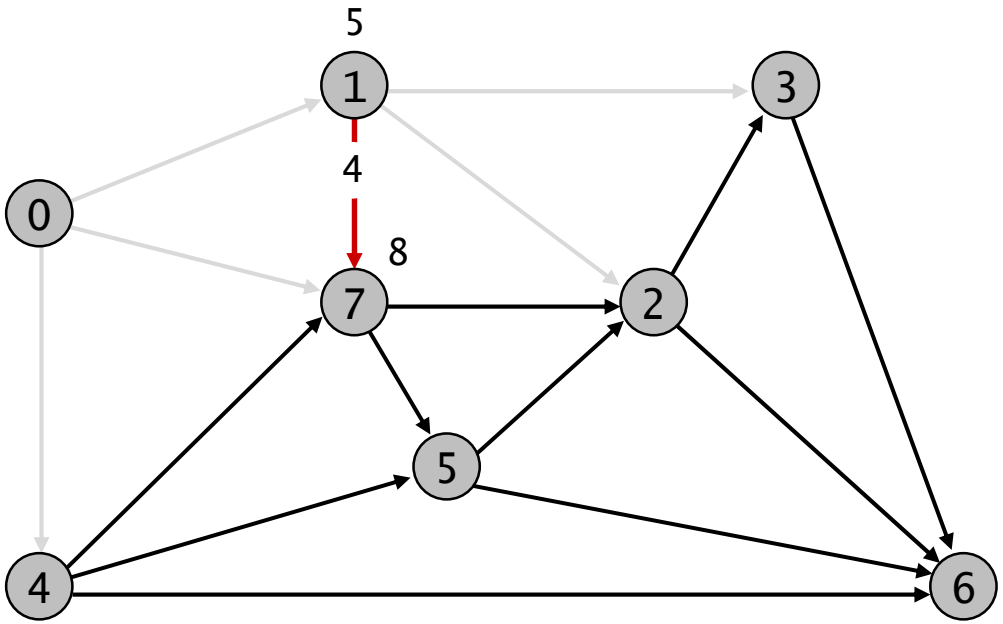
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



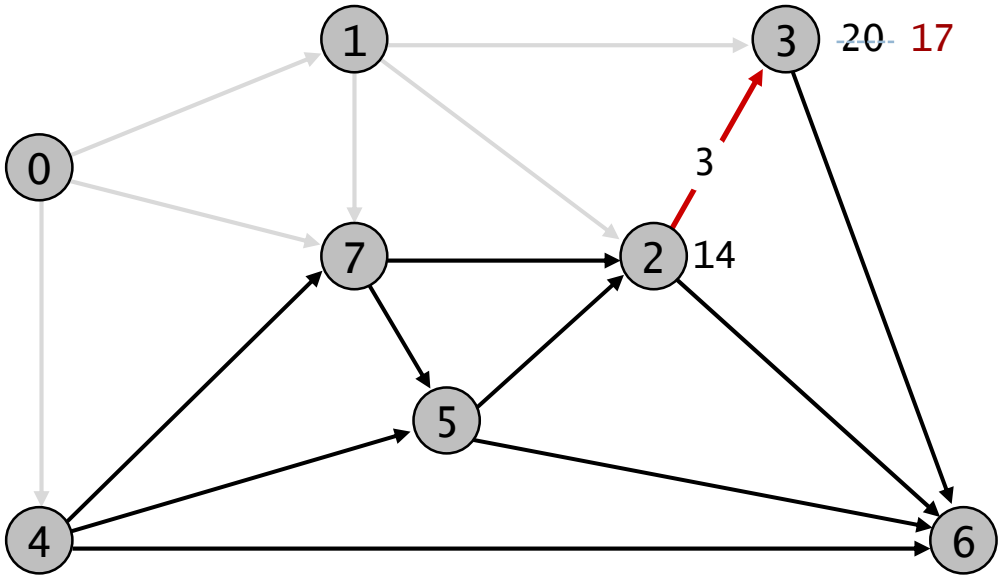
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	20.0	1->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



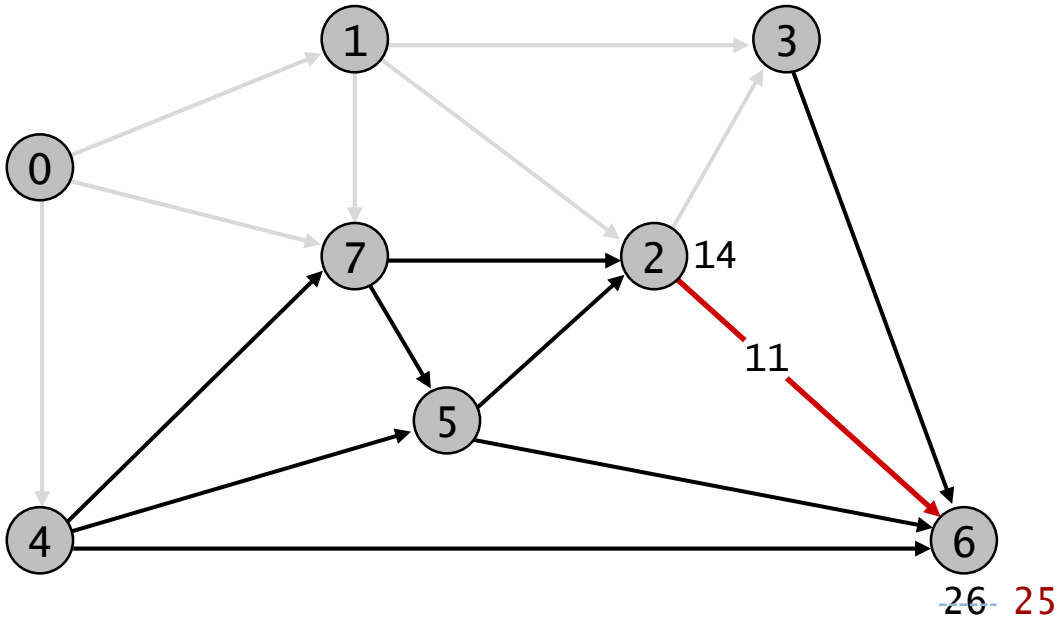
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	26.0	5->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



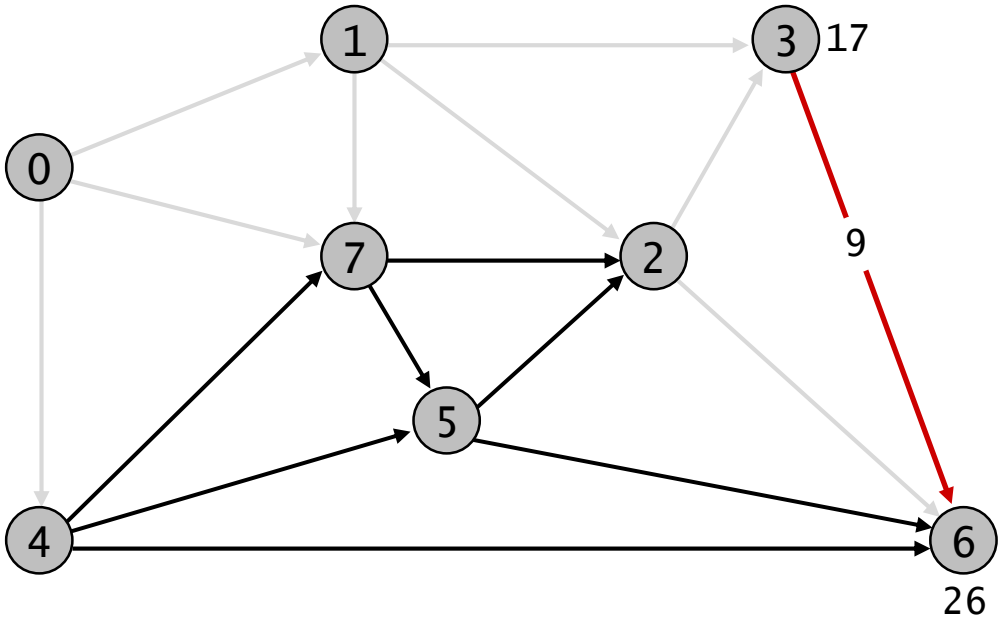
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



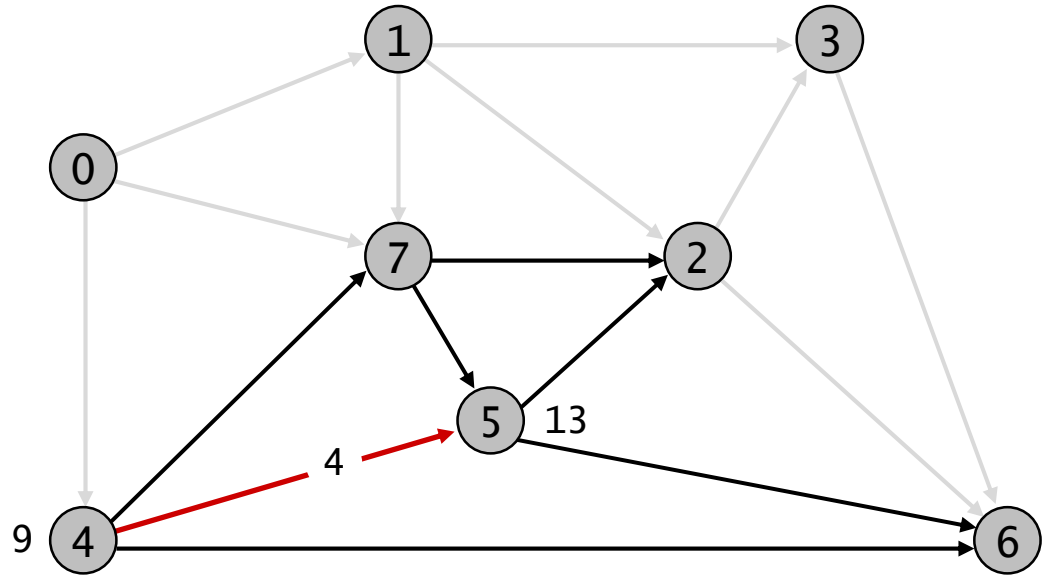
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



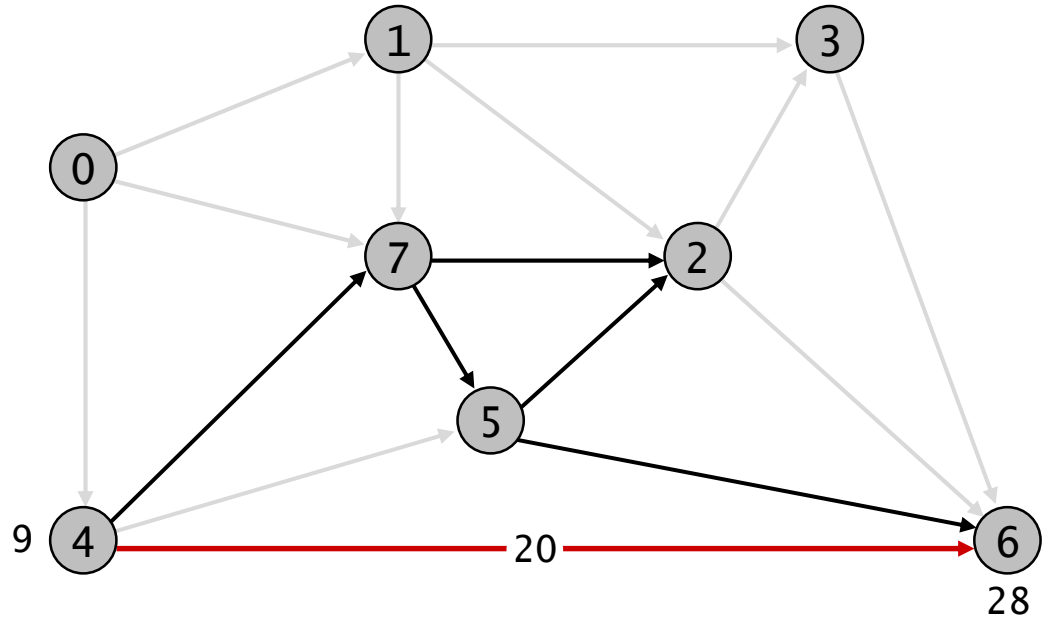
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



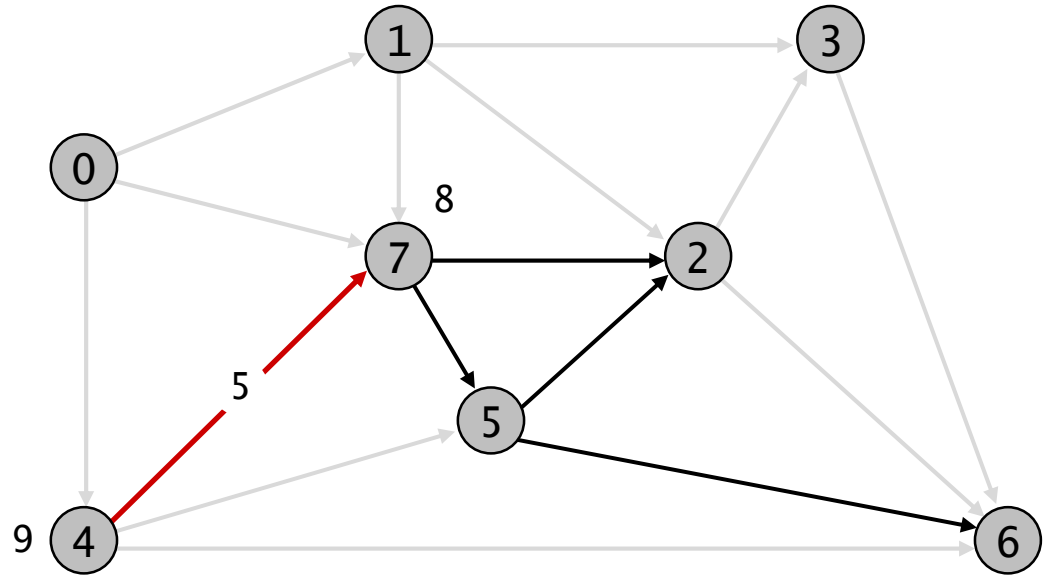
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



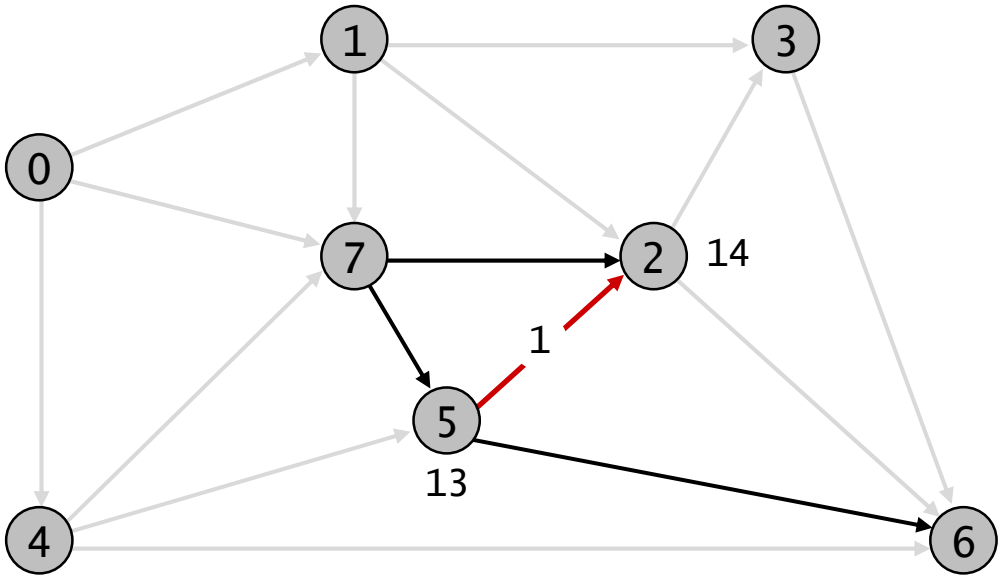
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



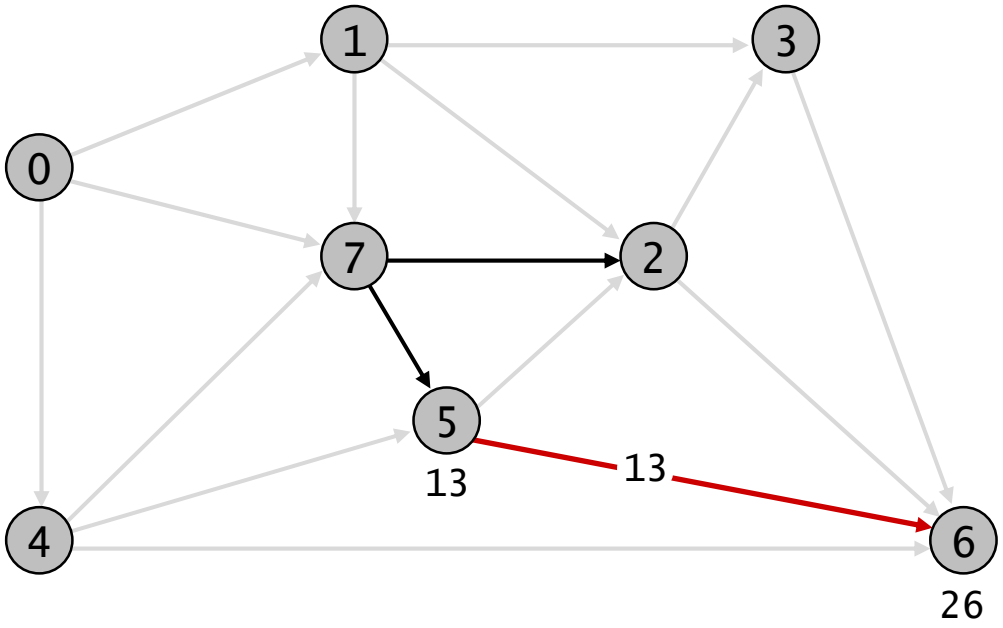
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



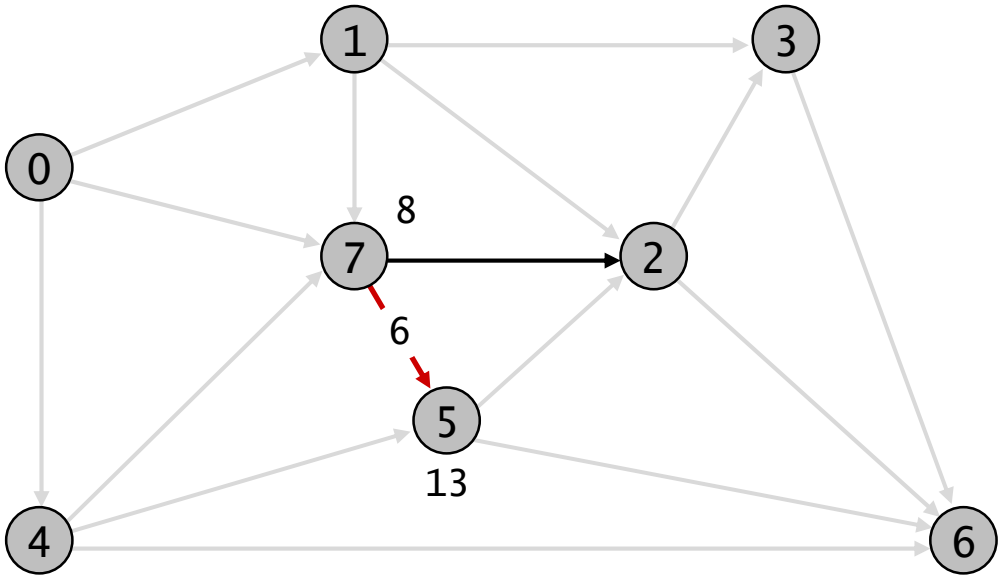
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



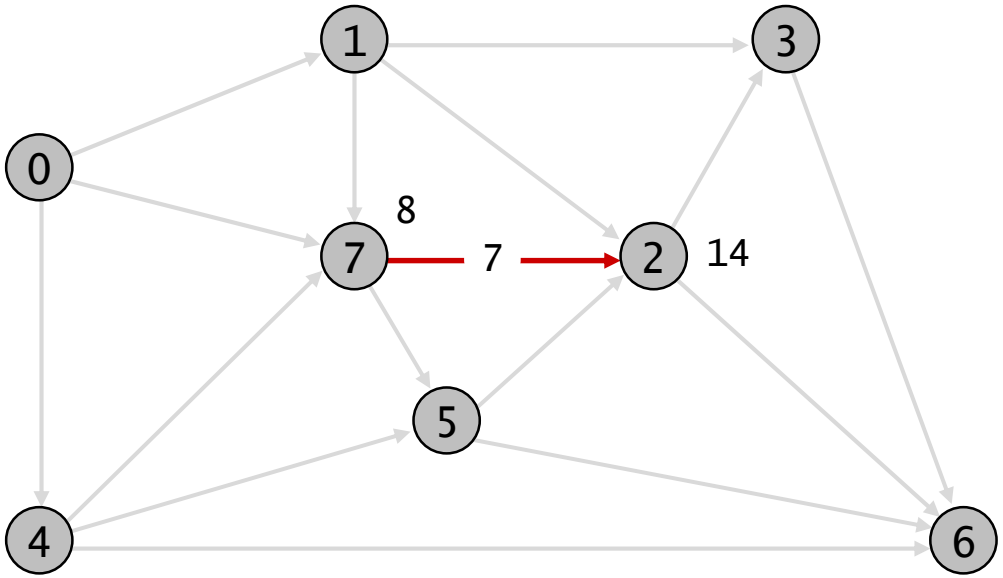
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.



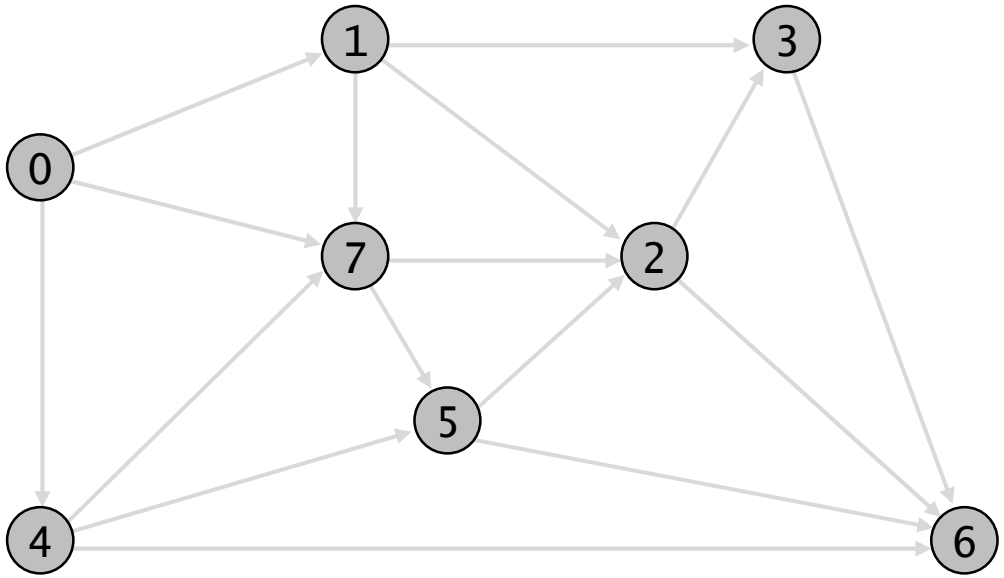
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذر 1

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.

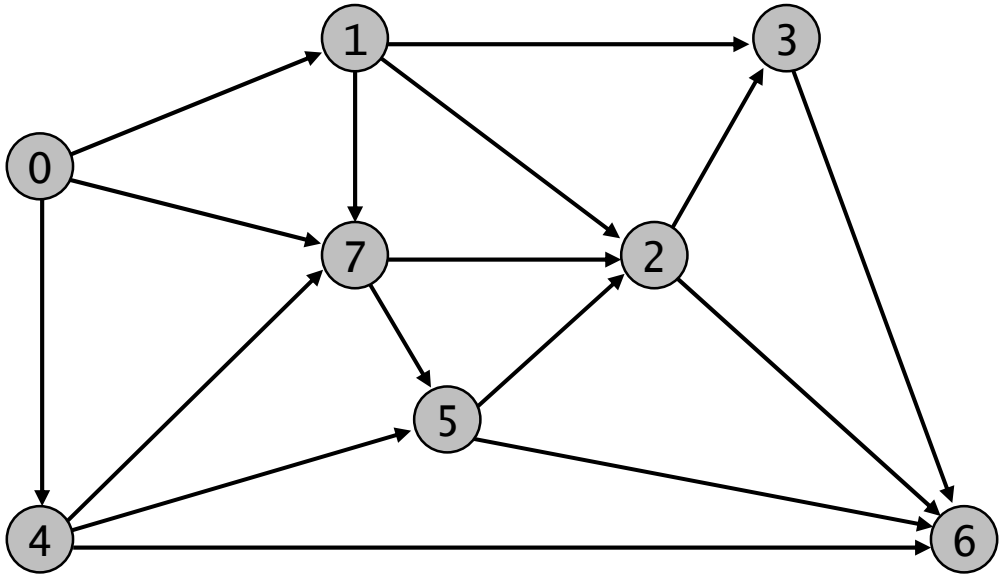


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

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

الگوریتم بلمن-فورد: اجرای نمایشی

V مرتبه همه‌ی یالها را relax کن.

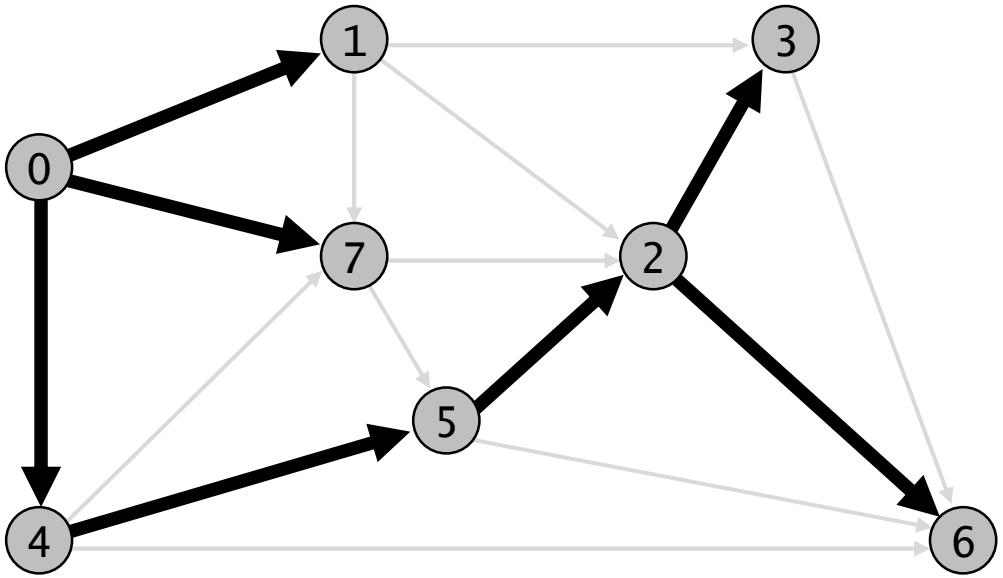


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

گذرهای ۲، ۳، ۴ و ... هیچ تغییری ایجاد نمی کنند

الگوریتم بلمن-فورد: اجرای نمایشی

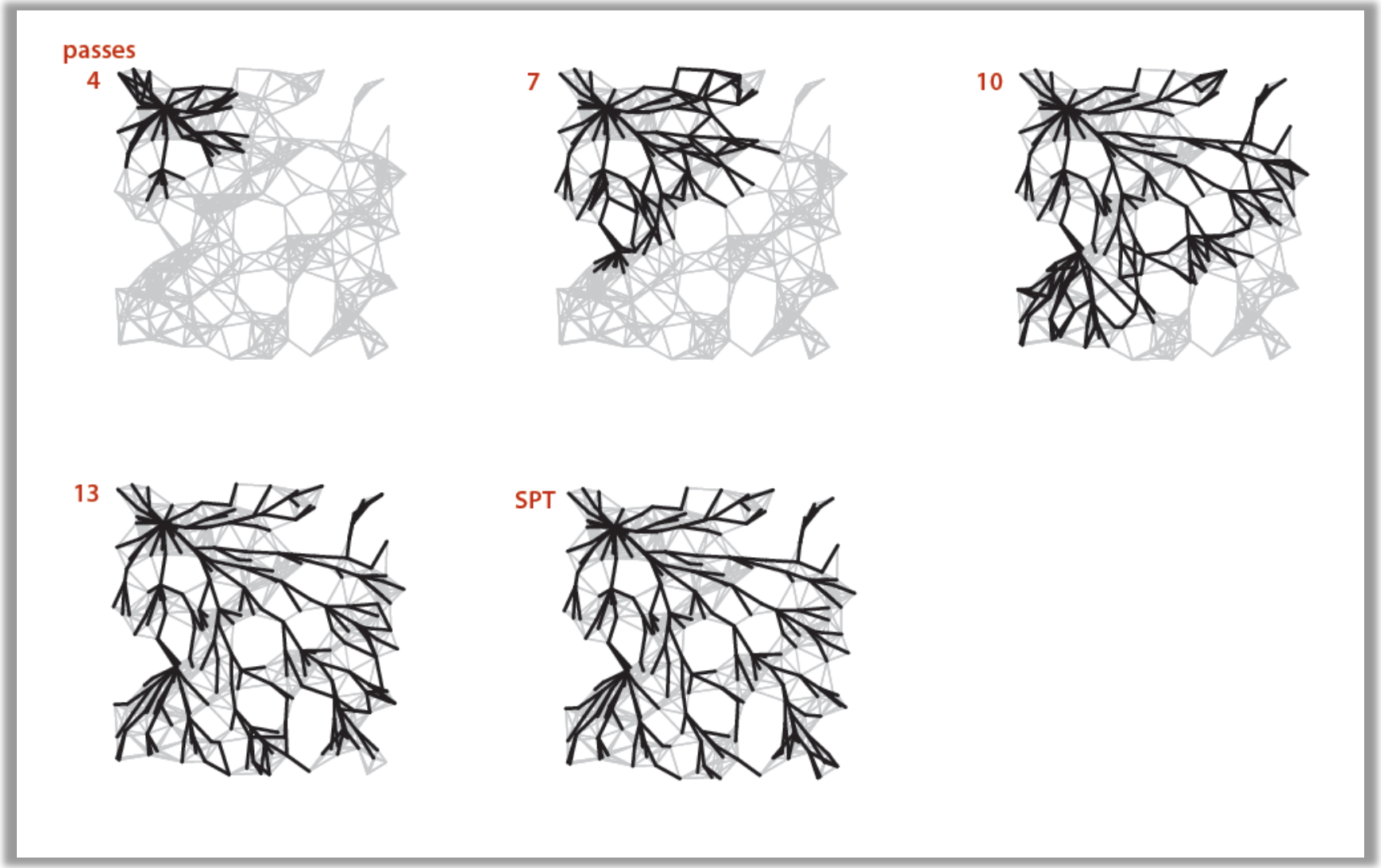
V مرتبه همه‌ی یالها را relax کن.



درخت کوتاه‌ترین مسیرها از رأس 0

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0->1
2	14.0	5->2
3	17.0	2->3
4	9.0	0->4
5	13.0	4->5
6	25.0	2->6
7	8.0	0->7

الگوریتم بلمن-فورد: اجرا



الگوریتم بلمن-فورد: تحلیل

Bellman-Ford Algorithm

Initialize `distTo[s] = 0` and `distTo[v] = ∞` for all other vertices

Repeat V times:
- **Relax each edge**

- گزاره. الگوریتم بلمن-فورد در هر گراف بدون دور منفی، درخت کوتاه‌ترین مسیر را در زمانی متناسب با $E \times V$ محاسبه می‌کند.

- ایده‌ی اثبات. پس از گذر `i`، کوتاه‌ترین مسیرهایی که حداکثر شامل `i` یال هستند محاسبه شده‌اند.

الگوریتم بلمن-فورد: بهبود زمان اجرا

- مشاهده. اگر در طول گذر `i` مقدار `distTo[v]` تغییر نکند، در گذر `i + 1` نیازی به راحت‌سازی یالهای خروجی از رأس `v` نیست.

- پیاده سازی با صف. رئوسی را که مقدار `distTo[]` برایشان تغییر کرده، در یک صف ذخیره کن.

مراقب باشید که از هر رأس تنها یک کپی در صف ذخیره کنید (چرا؟)

- تحلیل:
  - زمان اجرا هنوز در بدترین حالت متناسب با $V \times E$ است.
  - اما در عمل معمولا بسیار سریع‌تر است.

الگوریتم بلمن-فورد: پیاده‌سازی


```
public class BellmanFordSP  {

    private double[] distTo;
    private DirectedEdge[] edgeTo;
    private boolean[] onQ;
    private Queue<Integer> queue;

    public BellmanFordSP(EdgeWeightedDigraph G, int s)  {
        distTo = new double[G.V()];
        edgeTo = new DirectedEdge[G.V()];
        onQ = new boolean[G.V()];
        queue = new Queue<Integer>();

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        queue.enqueue(s);
        while(!queue.isEmpty()) {
            int v = queue.dequeue();
            onQ[v] = false;
            for (DirectedEdge e : G.adj(v)) relax(e);
        }

   }
```

الگوریتم بلمن-فورد: پیاده‌سازی


```
    private void relax(DirectedEdge e)
    {
        int v = e.from(), w = e.to();
        if (distTo[w] > distTo[v] + e.weight())
        {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (!onQ[w])
            {
                queue.enqueue(w);
                onQ[w] = true;
            }
    }
}
```

کوتاه‌ترین مسیر تک مبدأ: خلاصه

الگوریتم	محدودیت	معمولا	بدترین حالت	حافظه اضافی
مرتب‌سازی توپولوژیکی	بدون دور جهت‌دار	E + V	E + V	V
دایکسترا (هرم دودویی)	بدون وزنهای منفی	E log V	E log V	V
بلمن-فورد	بدون دور منفی	E V	E V	V
بلمن-فورد (مبتنی بر صف)		E + V	E V	V

- ملاحظه‌ی ۱. وجود دور جهت‌دار باعث سخت‌تر شدن مسئله می‌شود.
- ملاحظه‌ی ۲. وجود وزنهای منفی مسئله را باز هم سخت‌تر می‌کند.
- ملاحظه‌ی ۳. وجود دورهای منفی مسئله را غیر قابل حل می‌کند.

یافتن دور منفی

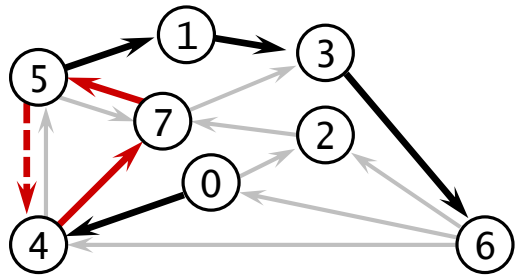
- دور منفی. دو متد به واسط کلاس SP اضافه کنید.

```
boolean  hasNegativeCycle()                    آیا دور منفی وجود دارد؟

Iterable<DirectedEdge>  negativeCycle()        برگرداندن دور منفی دسترس پذیر از مبدأ
```

گراف جهت‌دار وزن‌دار

```
4->5  0.35
5->4 -0.66
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```

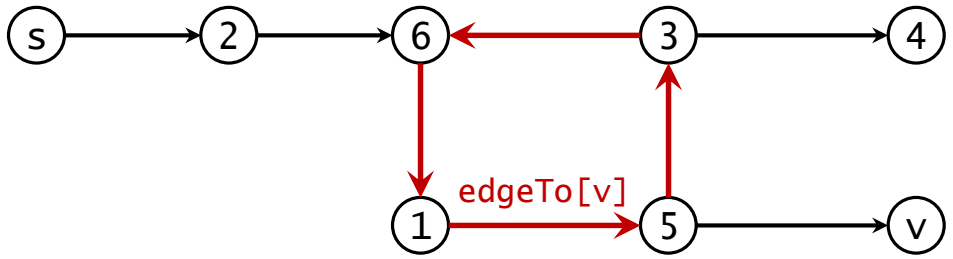


دور منفی (0.28 + 0.37 + -0.66)

5->4->7->5

یافتن دور منفی

- مشاهده. در صورت وجود دور منفی، الگوریتم بلمن-فورد در یک حلقه‌ی نامتناهی گیر می‌افتد.



- گزاره. اگر رأسی مانند v در گذر V به روزرسانی شود، یک دور منفی وجود دارد. [که می‌توان با استفاده از مقادیر edgeTo[] آن را پیدا نمود]

- در عمل. وجود دور منفی دفعات بیشتری بررسی می‌شود.

کاربرد دور منفی: معاملات ارزی (آربیتراژ)

- مسئله. با داشتن جدول نرخ تبدیل ارز، آیا امکان انجام یک معامله‌ی پر سود وجود دارد؟

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.350	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.620	1	0.953
CAD	0.995	0.732	0.650	1.049	1

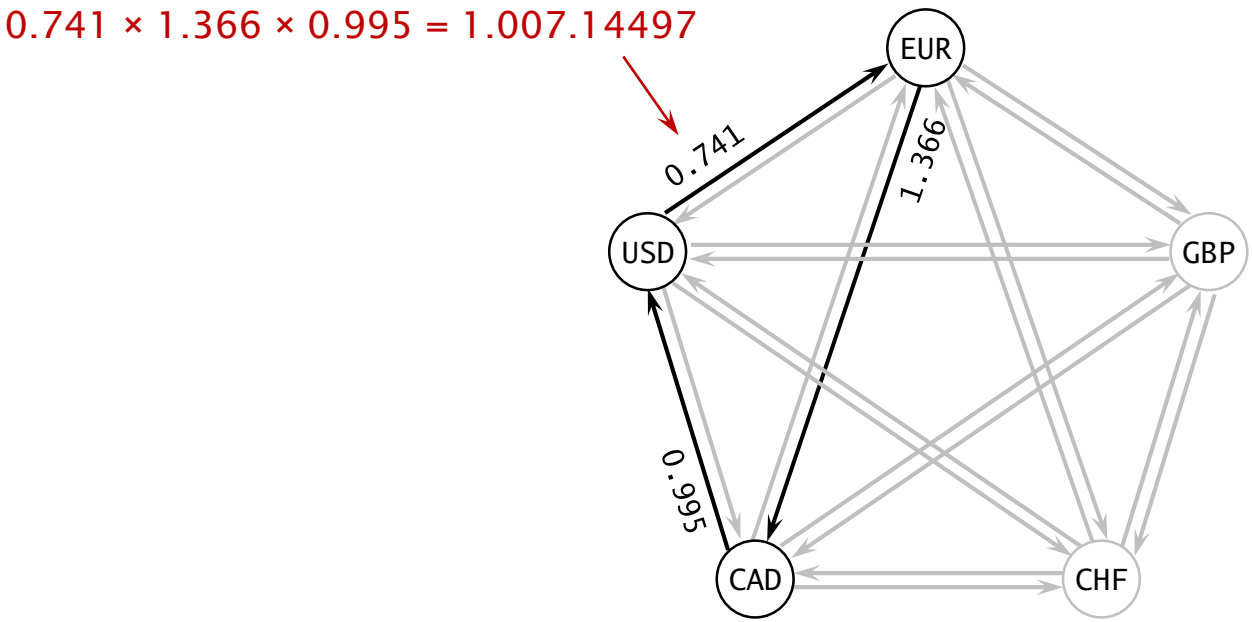
- مثال.

1000 USD -> 741 EUR -> 1012.206 CAD -> 1007.14497 USD

1000 × 0.741 × 1.366 × 0.995 = 1.007.14497

کاربرد دور منفی: معاملات ارزی (آربیتراژ)

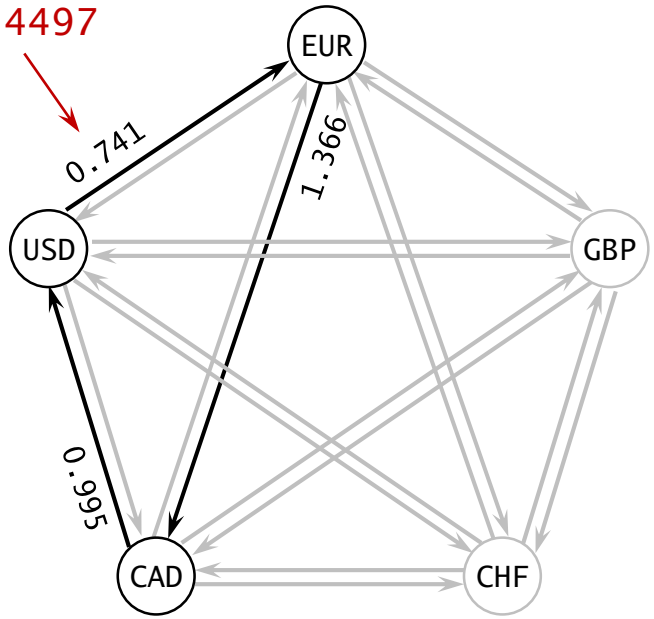
- گراف تبدیل ارز.
  - رأس = ارز؛ یال = معامله [وزن برابر با نرخ تبدیل]
  - یک دور جهت‌دار پیدا کن به گونه ای که حاصل‌ضرب وزن یالهای آن بزرگ‌تر از یک شود.



کاربرد دور منفی: معاملات ارزی (آربیتراژ)

- تبدیل مسئله به مسئله‌ی یافتن دور منفی با لگاریتم گرفتن از وزن یالها.
  - اگر وزن یالی برابر با w است، آن را با $-\ln w$ جایگزین کن.
  - با این کار ضرب تبدیل به جمع می‌شود.
  - یک دور جهت‌دار پیدا کن به گونه‌ای که مجموع وزن یالهای آن کوچک‌تر از صفر شود.

$0.741 \times 1.366 \times 0.995 = 1.007.14497$



کوتاه‌ترین مسیرها: خلاصه

- الگوریتم دایکسترا.
  - زمان نزدیک به خطی هنگامی که وزن یالها غیرمنفی باشند.
- گراف‌های بدون دور جهت‌دار.
  - کاربردهای عملی فراوان
  - سریع‌تر از الگوریتم دایکسترا
  - وزنهای منفی مشکلی ایجاد نمی‌کنند.
- وزنهای منفی و دورهای منفی.
  - کاربردهای عملی فراوان
  - در صورت عدم وجود دور منفی، می‌توان از الگوریتم بلمن-فورد استفاده نمود.
  - در صورت وجود دور منفی، می‌توان آن را با الگوریتم بلمن-فورد پیدا نمود.