روش حریصانه

سید ناصر رضوی n.razavi@tabrizu.ac.ir
۱۳۹۵

فهرست مطالب

معرفی

- ایده. با یک مجموعه جواب تهی شروع کن
  - هر بار از بین عناصر باقیمانده بهترین عنصر را انتخاب کن [انتخاب حریصانه]
  - عنصر انتخاب شده را در صورت امکان به مجموعه جواب اضافه کن [بررسی امکان سنجی]
  - اگر مجموعه جواب بیانگر یک راه‌حل است، آن را برگردان و در غیر این صورت مراحل فوق را تکرار کن [بررسی راه حل]
- کاربرد اصلی. مسائل بهینه‌سازی
- بهینگی. نیاز به یک اثبات رسمی دارد!
  - آیا دنباله‌ای از انتخاب‌های بهینه محلی، به یک راه‌حل بهینه سراسری منجر می‌شود؟

مسئله باقیمانده پول

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۰

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۲۵

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۳۵

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۳۵

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۳۵

مسئله باقیمانده پول

- مثال ۱. پس دادن ۳۶ سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده. [مجموعه جواب]

مجموع: ۳۶

راه حل بهینه

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

[مجموعه جواب]

مجموع: ۰

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

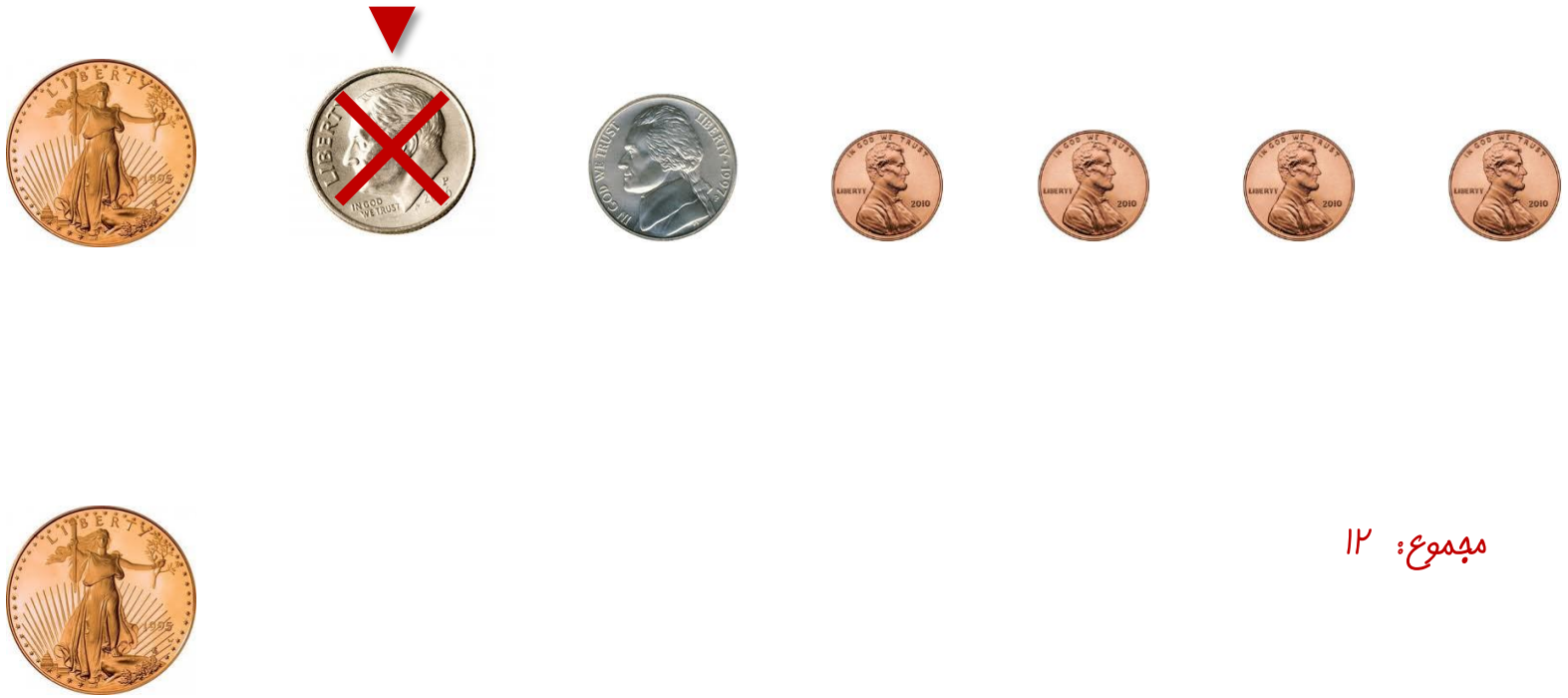
[مجموعه جواب]

مجموع: ۱۲

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.



- سکه‌های انتخاب شده.

[مجموعه جواب]

مجموع: ۱۲

مسئله باقیمانده پول

مجموع: ۱۲

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

[مجموعه جواب]

مجموع: ۱۳

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

[مجموعه جواب]

مجموع: ۱۴

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

[مجموعه جواب]

مجموع: ۱۵

مسئله باقیمانده پول

- مثال ۲. پس دادن ۱۶سنت با حداقل تعداد سکه‌ها

- سکه‌های موجود در صندوق.

- سکه‌های انتخاب شده.

[مجموعه جواب]

راه حل غیر بهینه

مجموع: ۱۶

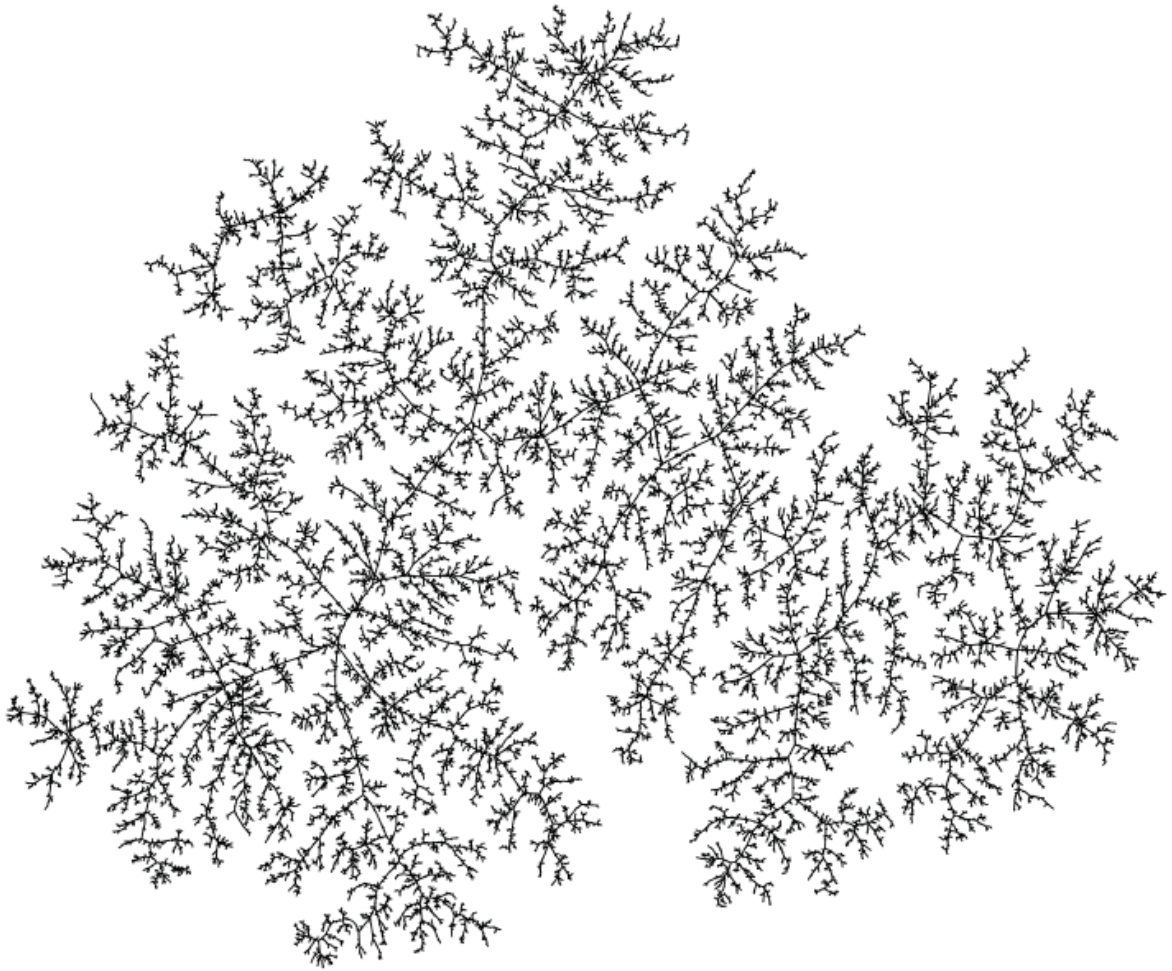
اجزای یک الگوریتم حریصانه

- انتخاب.
  - انتخاب عنصر بعدی که باید به مجموعه جواب اضافه گردد.
  - انتخاب بر اساس یک ملاک حریصانه انجام می‌شود؛ به این معنا که هر انتخاب در زمان خود بهینه است.
- امکان‌سنجی.
  - بررسی این که آیا اضافه کردن عنصر جدید به مجموعه جواب، امکان‌پذیر است یا خیر.
- بررسی راه‌حل.
  - بررسی این که آیا مجموعه جواب جدید، بیانگر یک راه‌حل است یا خیر.

۲۰ درخت پوشای کمینه

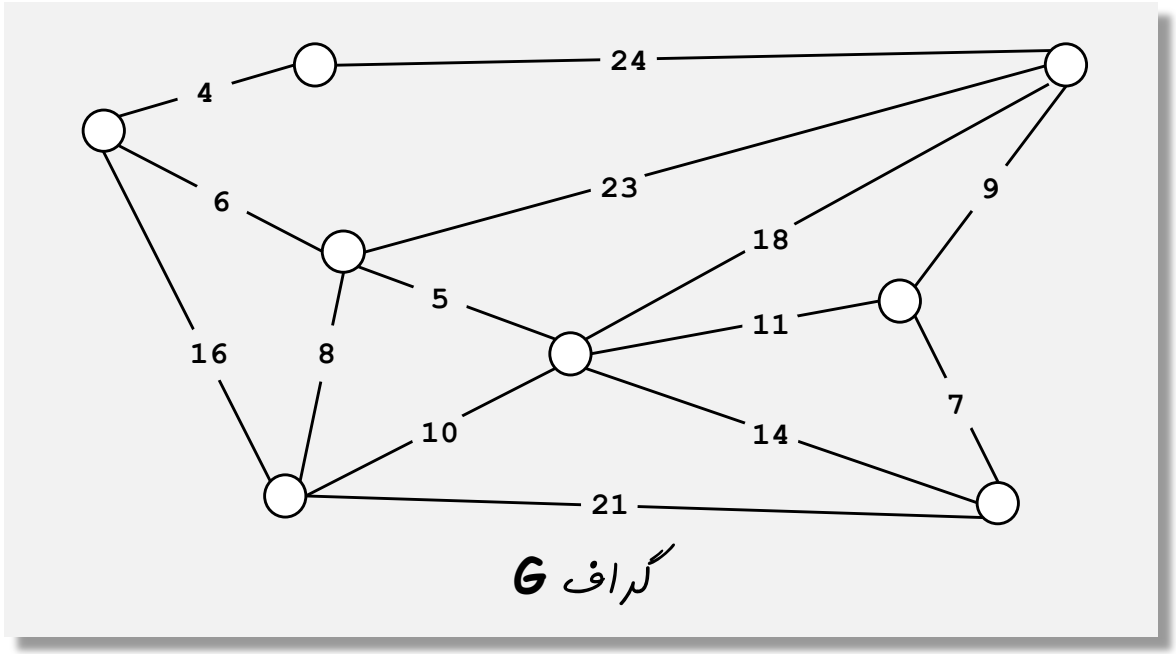
فهرست مطالب

- درخت پوشای کمینه.
  - معرفی
  - واسط گراف وزن‌دار
  - الگوریتم حریصانه
  - الگوریتم کروسکال
  - الگوریتم پریم



درخت پوشای کمینه

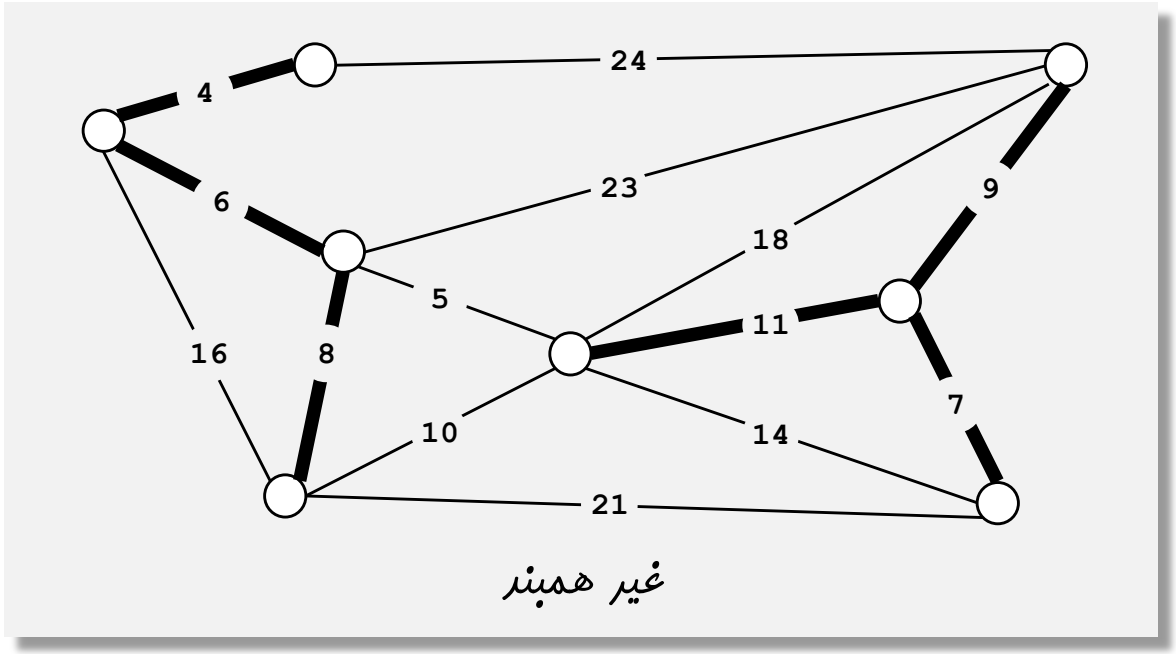
- ورودی. گراف بدون جهت و همبند G که وزن یال‌های آن مثبت هستند.
- تعریف. یک درخت پوشا از گراف G یک زیرگراف همبند و بدون دور از گراف G است.
- هدف. یافتن یک درخت پوشا با کمترین وزن



گراف G

درخت پوشای کمینه

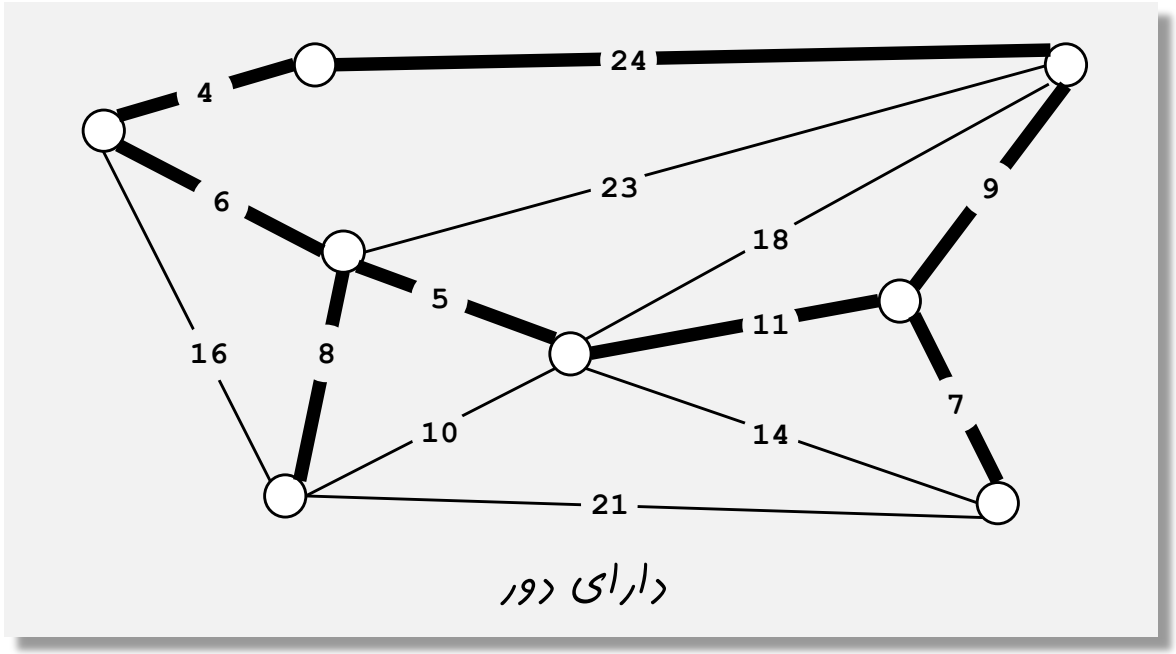
- ورودی. گراف بدون جهت و همبند G که وزن یال‌های آن مثبت هستند.
- تعریف. یک درخت پوشا از گراف G یک زیرگراف همبند و بدون دور از گراف G است.
- هدف. یافتن یک درخت پوشا با کمترین وزن



غیر همبند

درخت پوشای کمینه

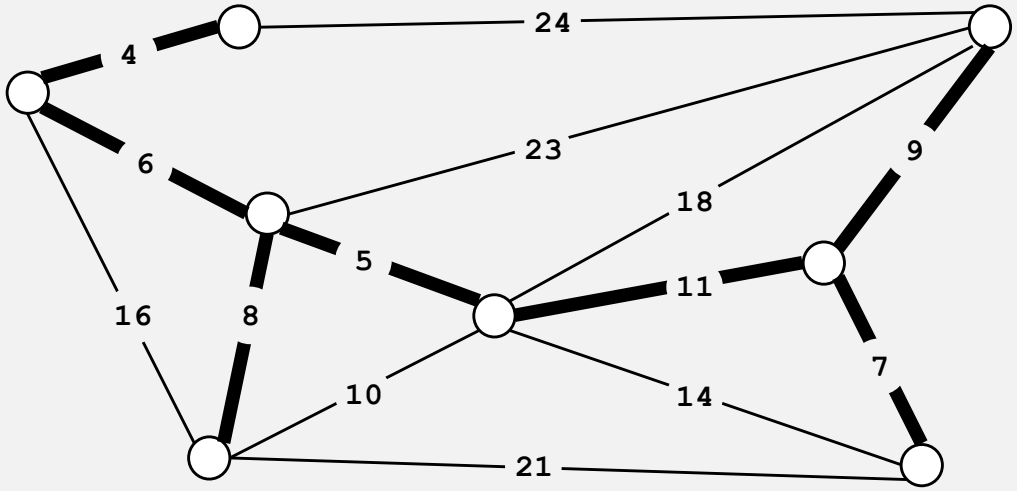
- ورودی. گراف بدون جهت و همبند G که وزن یال‌های آن مثبت هستند.
- تعریف. یک درخت پوشا از گراف G یک زیرگراف همبند و بدون دور از گراف G است.
- هدف. یافتن یک درخت پوشا با کمترین وزن



دارای دور

درخت پوشای کمینه

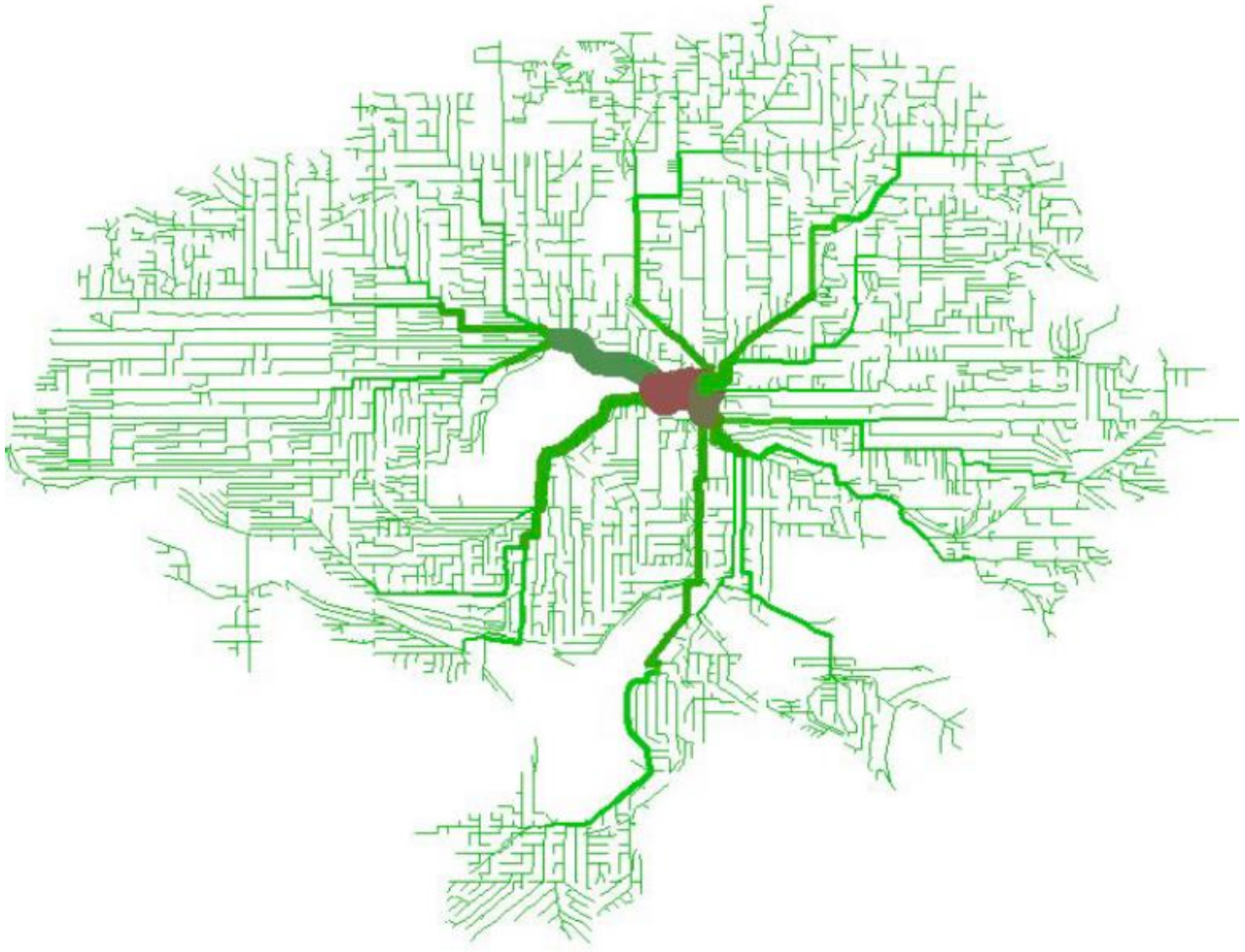
- ورودی. گراف بدون جهت و همبند G که وزن یال‌های آن مثبت هستند.
- تعریف. یک درخت پوشا از گراف G یک زیرگراف همبند و بدون دور از گراف G است.
- هدف. یافتن یک درخت پوشا با کمترین وزن



یک درخت پوشا با وزن ۵۰

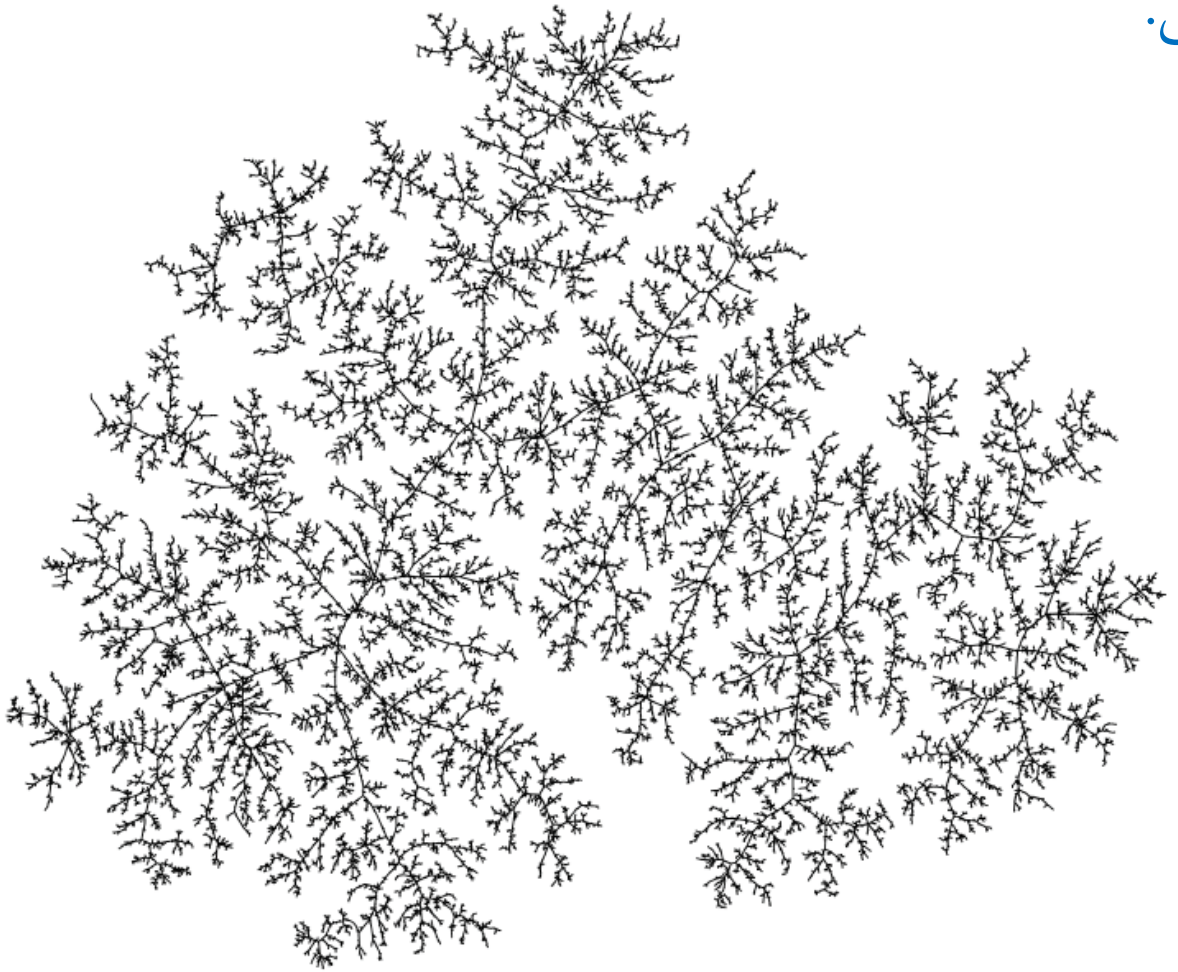
درخت پوشای کمینه

- درخت پوشای کمینه برای مسیرهای دوچرخه‌سواری در شهر سیاتل.



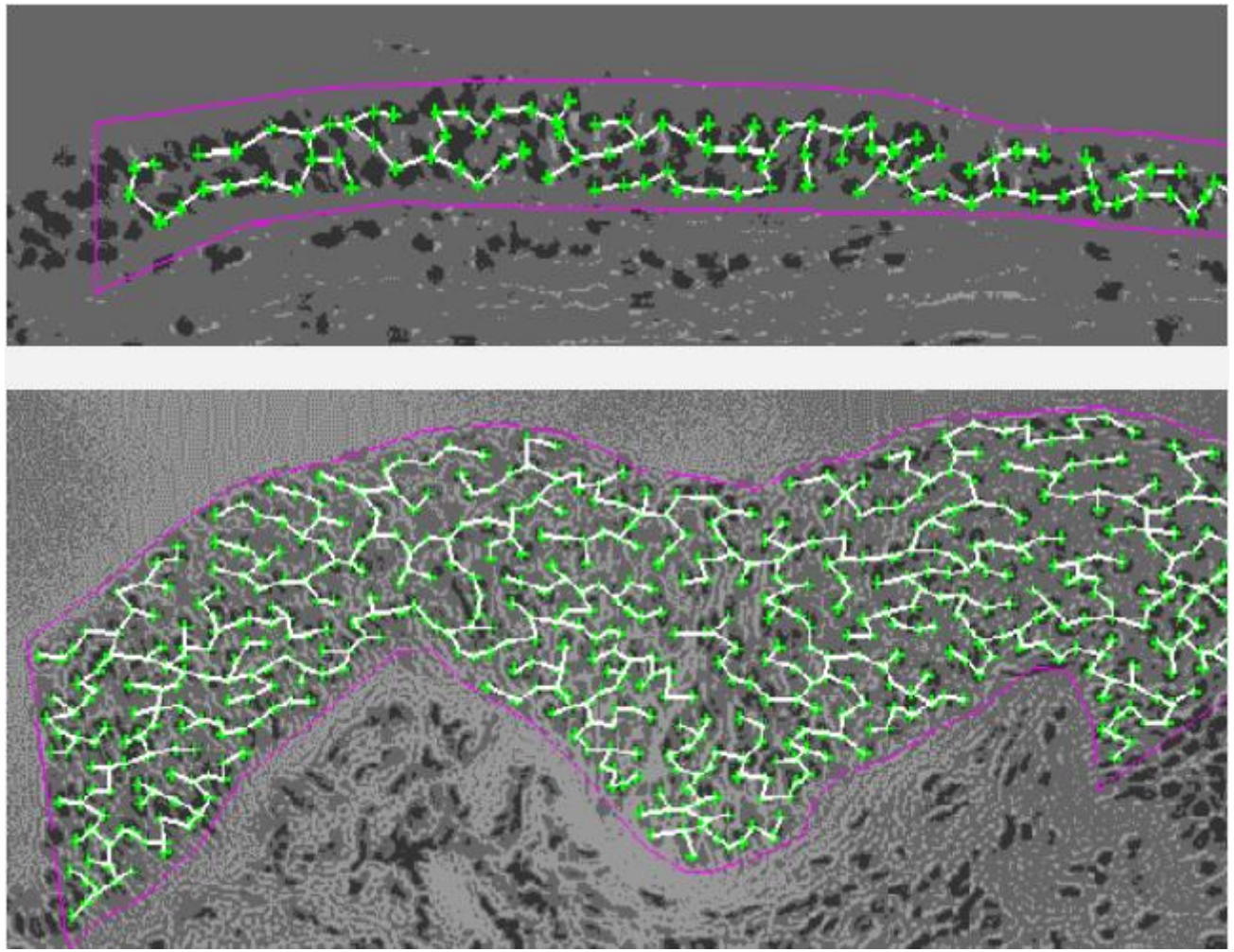
درخت پوشای کمینه

- درخت پوشای کمینه برای یک گراف تصادفی.



درخت پوشای کمینه

- درخت پوشای کمینه بیانگر آرایش هسته سلول‌ها در یک بافت پوستی برای پژوهش در مورد سرطان.



کاربردها

- برخی از کاربردهای درخت پوشای کمینه.
  - طراحی شبکه (ارتباطی، الکتریکی، کامپیوتری، کابلی، جاده‌ای)
  - الگوریتم‌های تقریبی برای مسائل ان-پی کامل (فروشنده دوره‌گرد)
  - یافتن شبکه راه‌ها در تصاویر هوایی و ماهواره‌ای
  - تشخیص چهره به صورت بلادرنگ
  - ...

۳۰ واسط گراف وزن‌دار

واسط یال وزن‌دار

- واسط کلاس یال وزن‌دار.

```
public class Edge implements Comparable<Edge>

            Edge(int v, int w, double weight)   ایجاد یال وزن دار v-w
       int  either()                            برگرداندن یکی از نقاط انتهایی
       int  other(int v)                        برگرداندن نقطه انتهایی دیگر
       int  compareTo(Edge that)                مقایسه این یال با یک یال دیگر
    double  weight()                            برگرداندن وزن یال
    String  toString()                          نمایش یال به صورت رشته ای
```

یال وزن‌دار: پیاده‌سازی جاوا


```
public class Edge implements Comparable<Edge>
{
   private final int v, w;
   private final double weight;

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

   public int either()
   {  return v;  }

   public int other(int vertex)
   {
      if (vertex == v) return w;
      else             return v;
   }

   public int compareTo(Edge that)
   {
      if      (this.weight < that.weight) return -1;
      else if (this.weight > that.weight) return +1;
      else                                return  0;
   }
}
```

سازنده

یکی از رئوس

رأس دیگر

مقایسه دو یال بر مبنای وزن آن ها

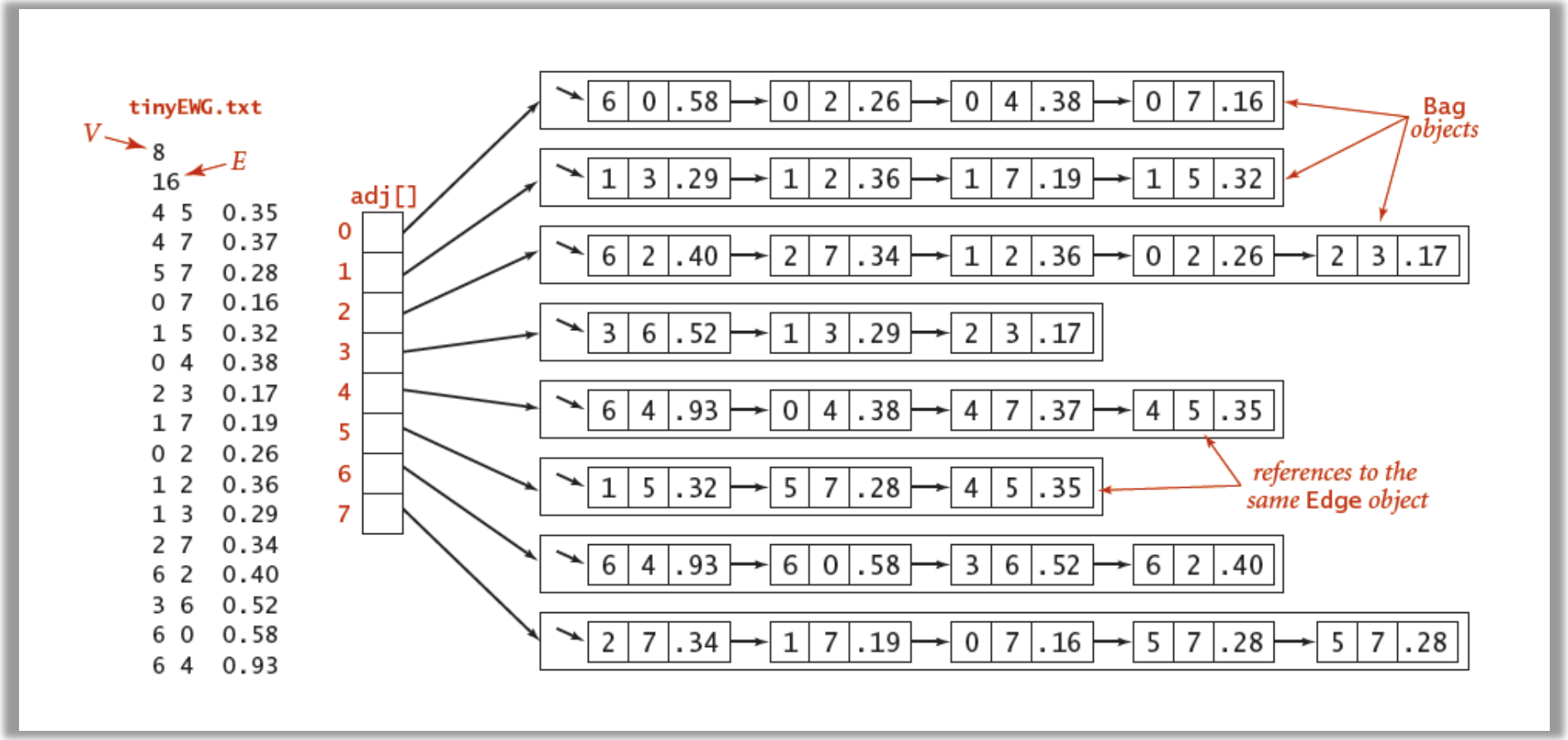
واسط گراف وزن‌دار

public class	EdgeWeightedGraph	
	EdgeWeightedGraph(int V)	ایجاد یک گراف تهی با V رأس
	EdgeWeightedGraph(In in)	ایجاد یک گراف از جریان ورودی
void	addEdge(Edge e)	افزودن یال **e** به گراف
Iterable<Edge>	adj(int v)	برگرداندن یال های متلاقی با رأس **v**
Iterable<Edge>	edges()	برگرداندن همه یال های گراف
int	V()	برگرداندن تعداد رئوس
int	E()	برگرداندن تعداد یال ها
String	toString()	نمایش گراف وزن دار به صورت یک رشته

قرارداد. در این پیاده‌سازی، داشتن طوقه و یال‌های موازی مجاز است.

گراف وزن‌دار: پیاده‌سازی لیست همسایگی

- به ازای هر رأس، یال‌های متلاقی با آن در یک لیست ذخیره می‌شوند.



گراف وزن‌دار: پیاده‌سازی جاوا


```
public class EdgeWeightedGraph
{
    private final int V;
    private final Bag<Edge>[] adj;

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

    public void addEdge(Edge e)
    {
       int v = e.either(), w = e.other(v);
       adj[v].add(e);
       adj[w].add(e);
   }

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

}
```

سازنده

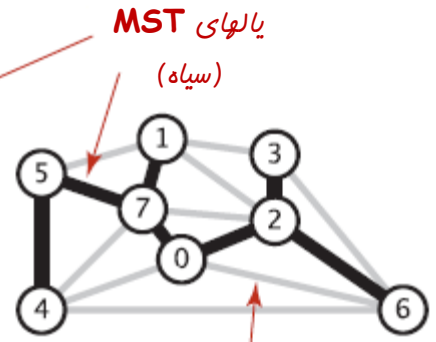
اضافه کردن یال به هر دو لیست همسایگی

واسط درخت پوشای کمینه

public class	MST	
	MST(EdgeWeightedGraph G)	سازنده
Iterable<Edge>	edges()	برگرداندن یال های درخت پوشای کمینه
double	weight()	برگرداندن وزن درخت پوشای کمینه

tinyEWG.txt

```
8
16
4 5  0.35
4 7  0.37
5 7  0.28
0 7  0.16
1 5  0.32
0 4  0.38
2 3  0.17
1 7  0.19
0 2  0.26
1 2  0.36
1 3  0.29
2 7  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```




```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

واسط درخت پوشای کمینه

public class	MST	
	MST(EdgeWeightedGraph G)	سازنده
Iterable<Edge>	edges()	برگرداندن یال های درخت پوشای کمینه
double	weight()	برگرداندن وزن درخت پوشای کمینه

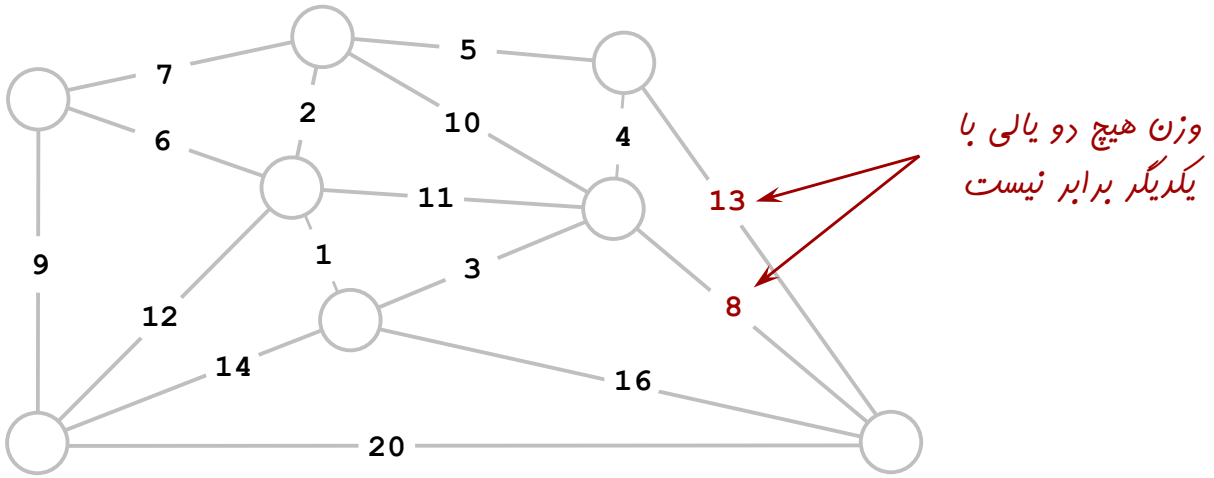
```
Public static void main(String[] args)
{
   In in = new In(args[0]);
   EdgeWeightedGraph G = new EdgeWeightedGraph(in);
   MST mst = new MST(G);
   for (Edge e : mst.edges())
      StdOut.println(e);
   StdOut.printf("%.2f\n", mst.weight());
}
```

```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

۳۸ الگوریتم حریصانه

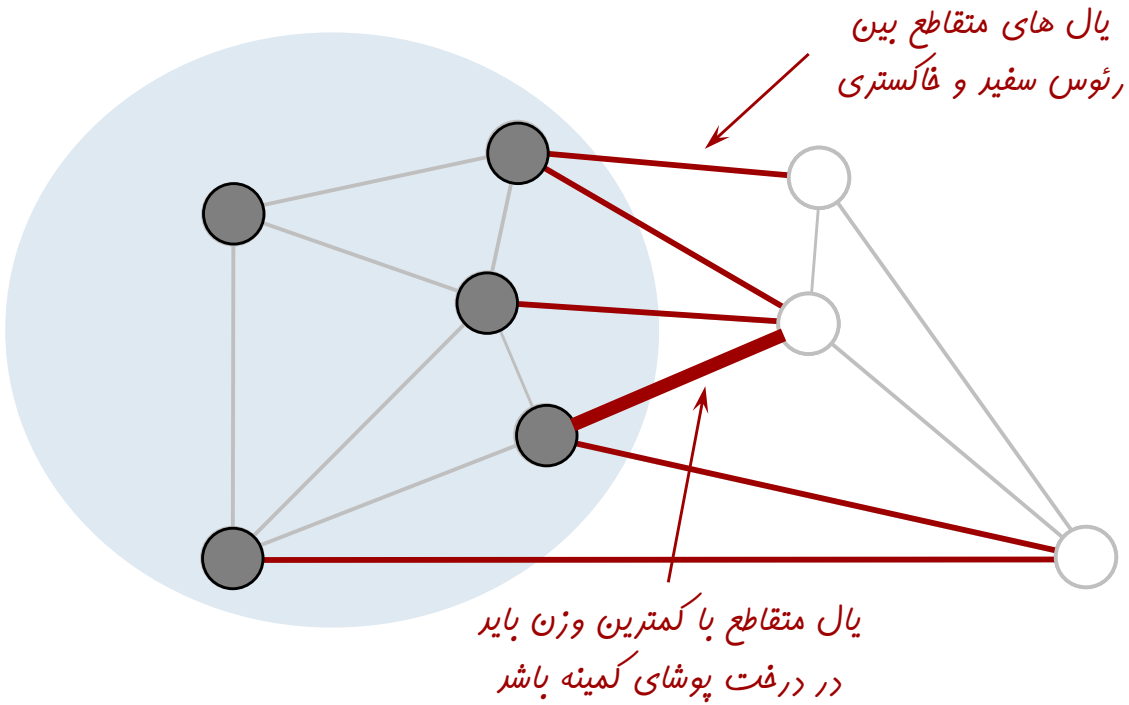
ویژگی برش

- فرضیات ساده کننده.
  - وزن یال‌ها متفاوتند.
  - گراف، همبند است.
- نتیجه. درخت پوشای کمینه وجود دارد و منحصر به فرد است.



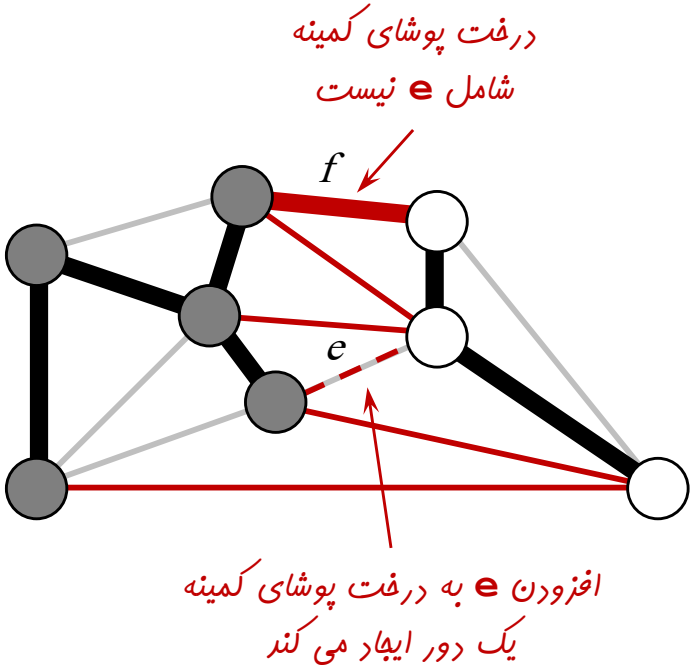
ویژگی برش

- تعریف. یک برش از گراف، رئوس آن گراف را به دو مجموعه (غیرتهی) افراز می‌کند.
- تعریف. یک یال متقاطع، یک رأس از یک مجموعه را به یک رأس از مجموعه دیگر متصل می‌کند.
- ویژگی برش. به ازای هر برش، یال متقاطع با کمترین وزن در درخت پوشای کمینه قرار دارد.



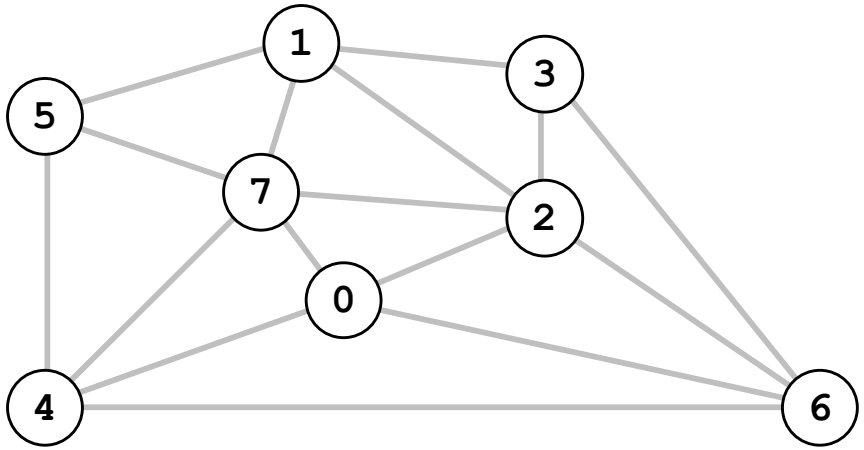
ویژگی برش: اثبات درستی

- تعریف. یک برش از گراف رئوس آن گراف را به دو مجموعه (غیرتهی) افراز می کند.
- تعریف. یک یال متقاطع یک رأس از یک مجموعه را به یک رأس از مجموعه دیگر متصل می کند.
- ویژگی برش. به ازای هر برش، یال متقاطع با کمترین وزن در درخت پوشای کمینه قرار دارد.
- اثبات. فرض کنید **e** یک یال متقاطع با کمترین وزن در یک برش باشد.
  - فرض کنید **e** در درخت پوشای کمینه نباشد.
  - افزودن **e** به درخت پوشای کمینه یک دور ایجاد می کند.
  - در این دور یک یال دیگر مانند **f** باید یک یال متقاطع باشد.
  - درخت حاصل از برداشتن **f** و افزودن **e** همچنان یک درخت پوشا است.
  - از آنجا که وزن **e** از وزن **f** کمتر است، این درخت پوشای جدید وزن کمتری دارد.
  - تناقض!



الگوریتم حریصانه: اجرای نمایشی

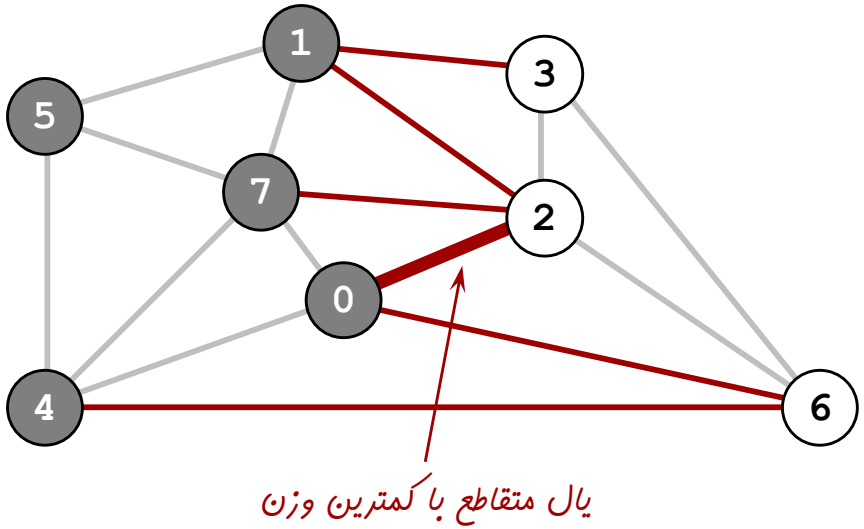
- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.



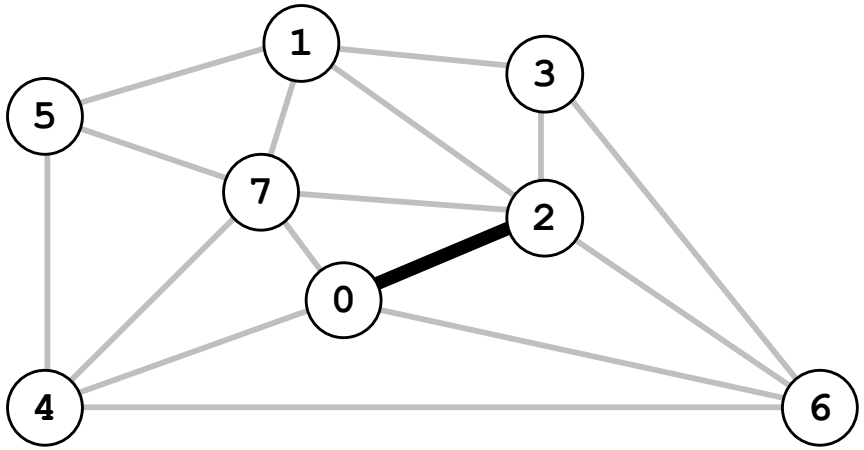
یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

0-2	0.26
1-3	0.29
2-7	0.34
1-2	0.36
6-0	0.58
6-4	0.93

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.

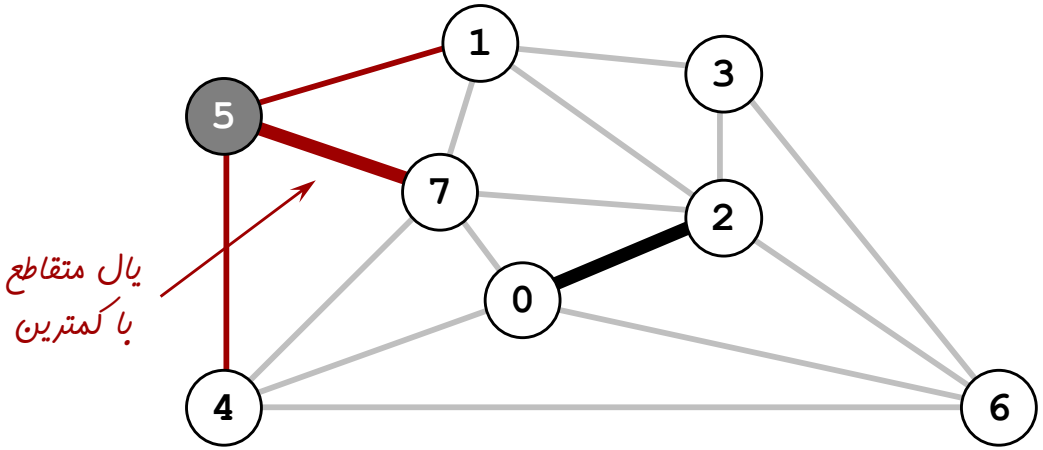


MST

0-2

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

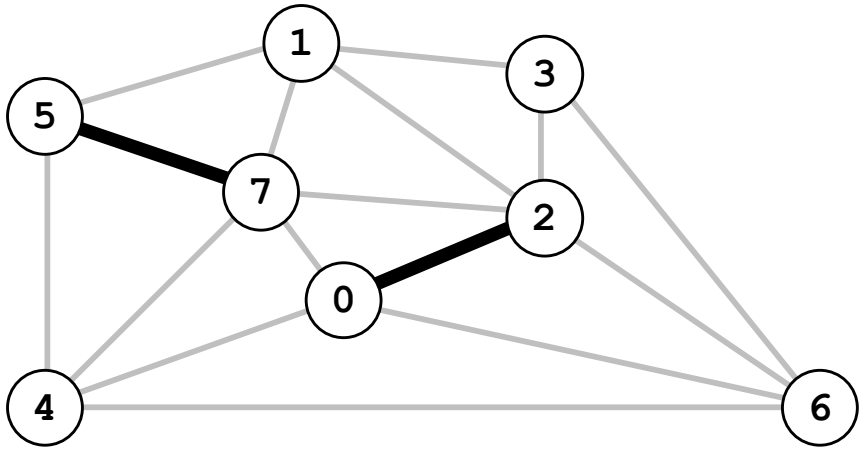
5-7	0.28
1-5	0.32
4-5	0.35

MST

0-2

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.

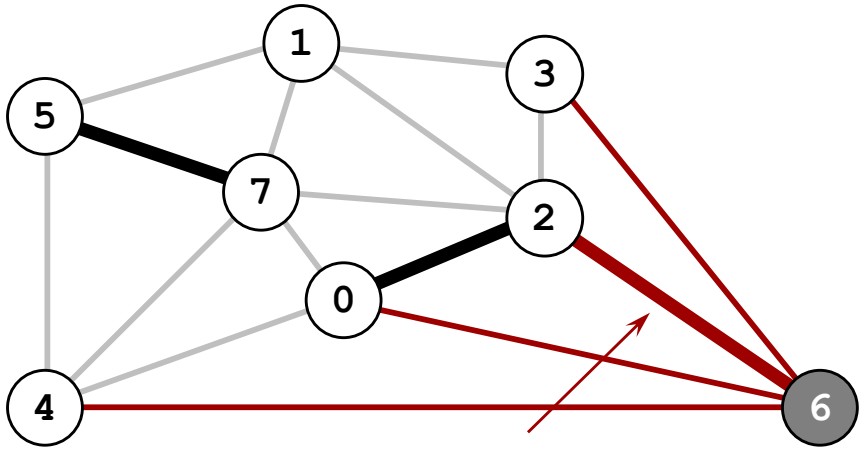


MST

0-2 5-7

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

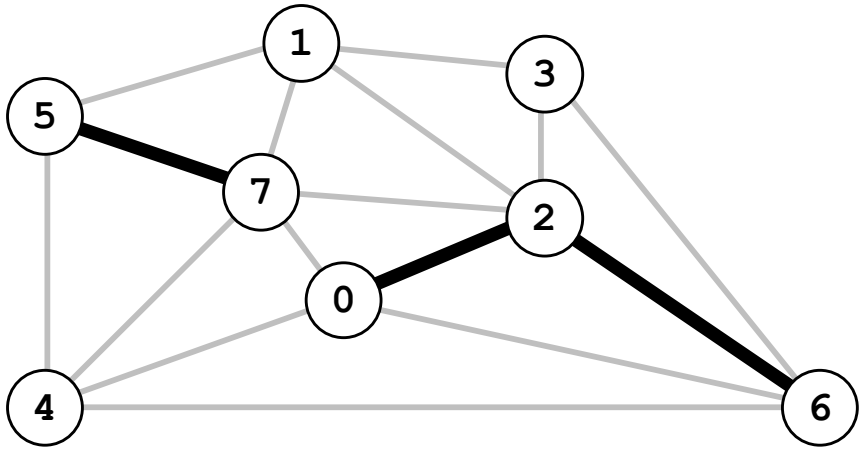
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

MST

0-2 5-7

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.

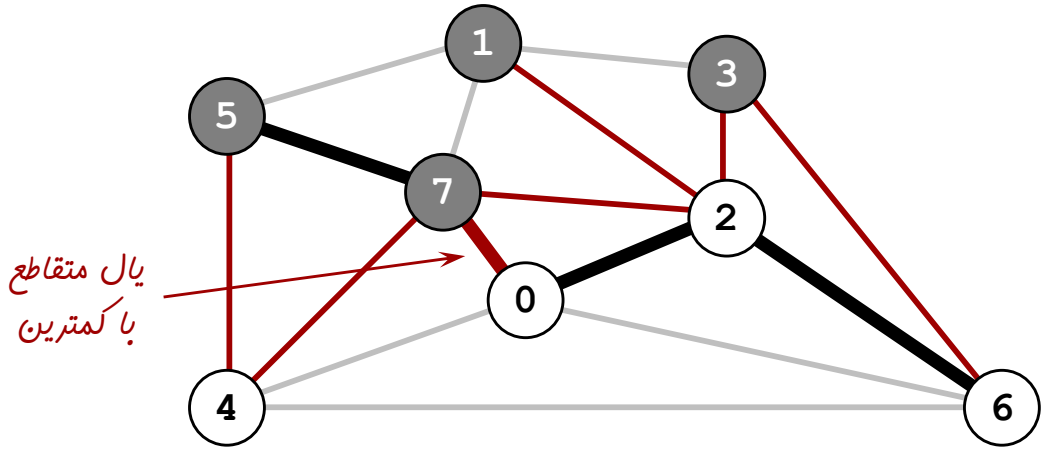


MST

0-2 5-7 6-2

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا **V - 1** یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

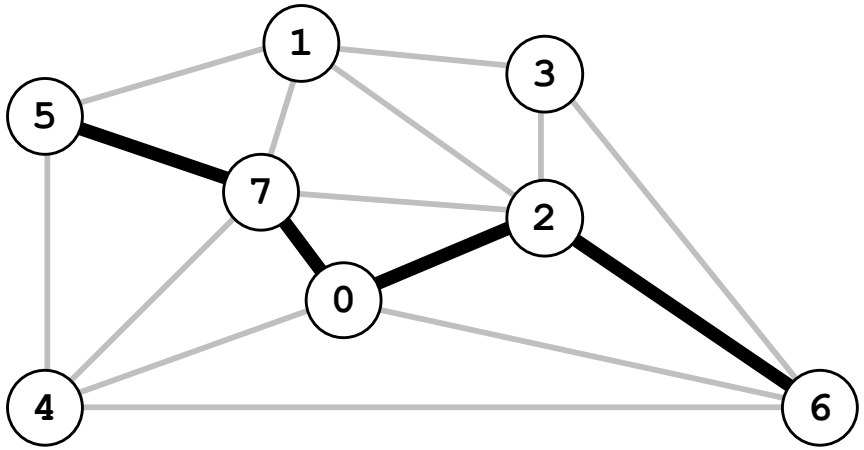
0-7	0.16
2-3	0.17
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
3-6	0.52

MST

0-2 5-7 6-2

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.

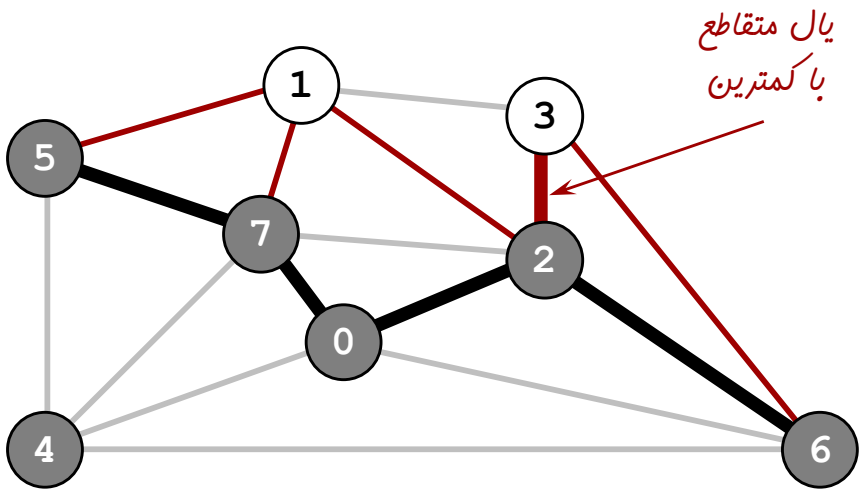


MST

0-2 5-7 6-2 0-7

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - \mathbf{1}$ یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

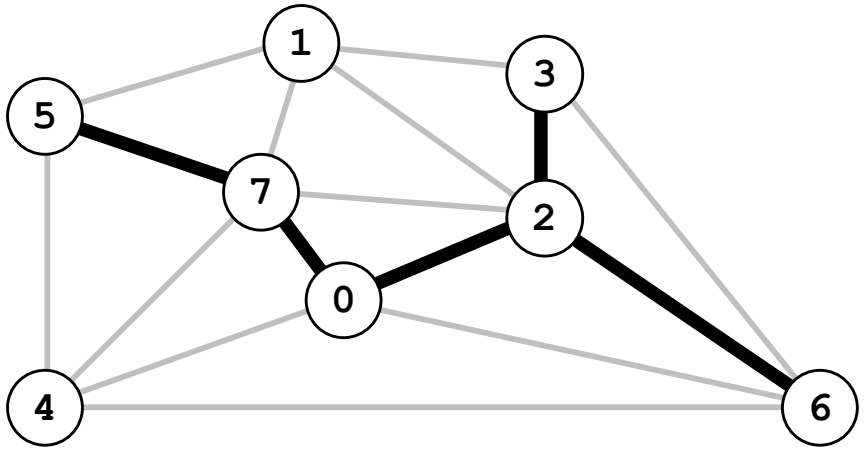
2-3	0.17
1-7	0.19
1-5	0.32
1-2	0.36
3-6	0.52

MST

0-2 5-7 6-2 0-7

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.

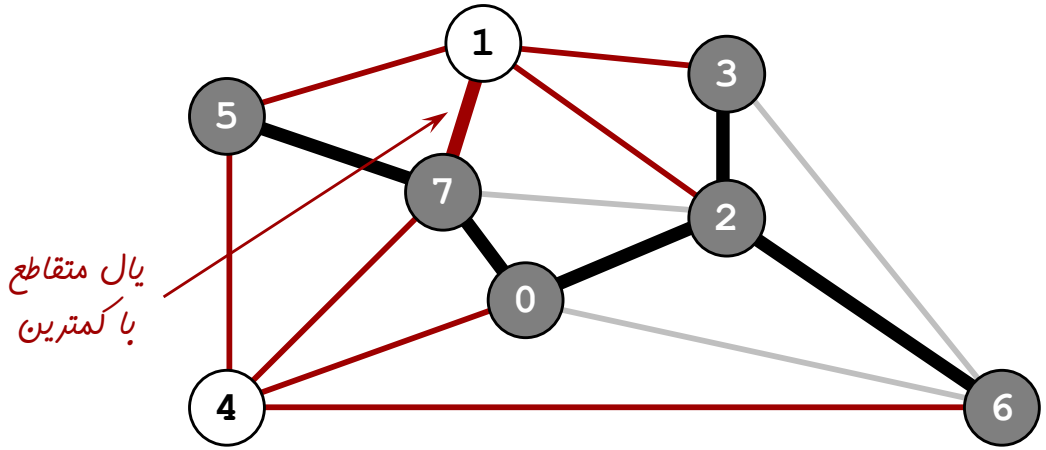


MST

0-2 5-7 6-2 0-7 2-3

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - \mathbf{1}$ یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

MST در

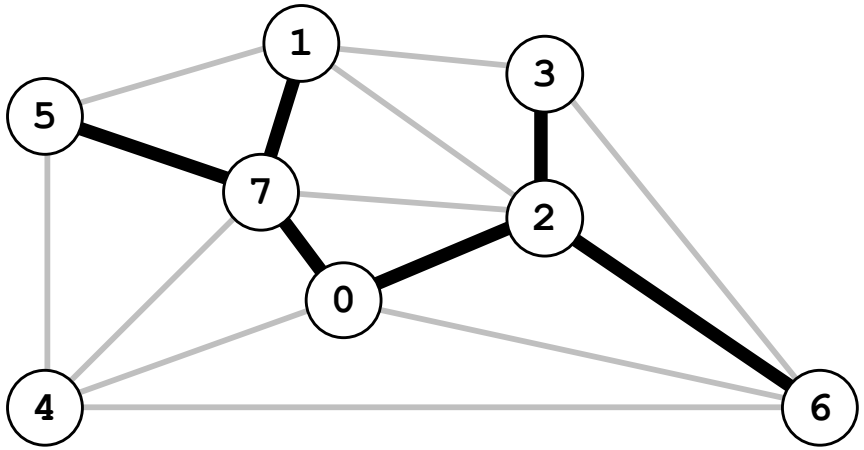
1-7	0.19
1-3	0.29
1-5	0.32
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-4	0.93

MST

0-2 5-7 6-2 0-7 2-3

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.

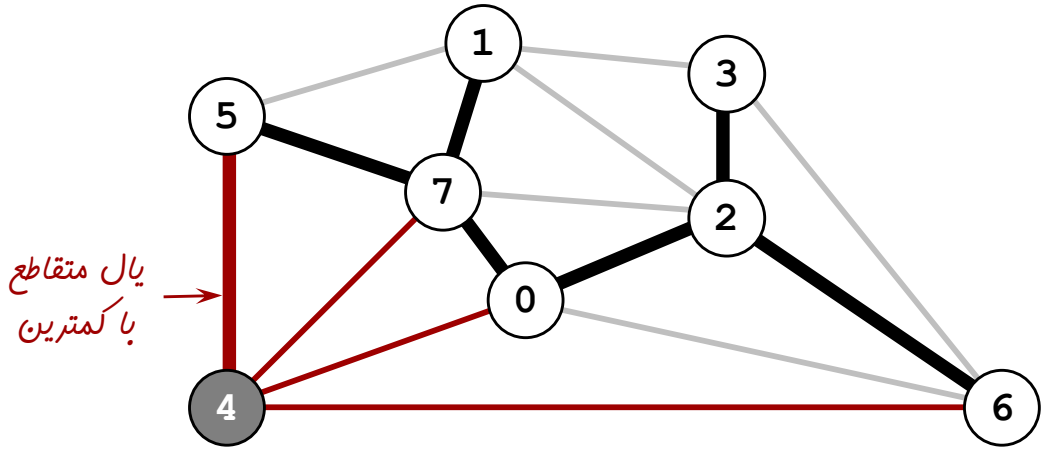


MST

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

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.



یال های متقاطع
(مرتب شده بر اساس وزن)

در MST →

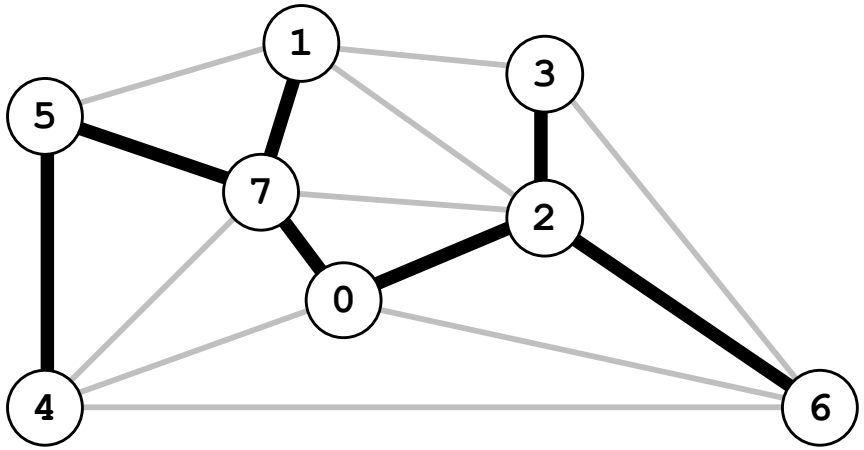
4-5	0.35
4-7	0.37
0-4	0.38
6-4	0.93

MST

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

الگوریتم حریصانه: اجرای نمایشی

- الگوریتم حریصانه.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V} - 1$ یال سیاه شوند.



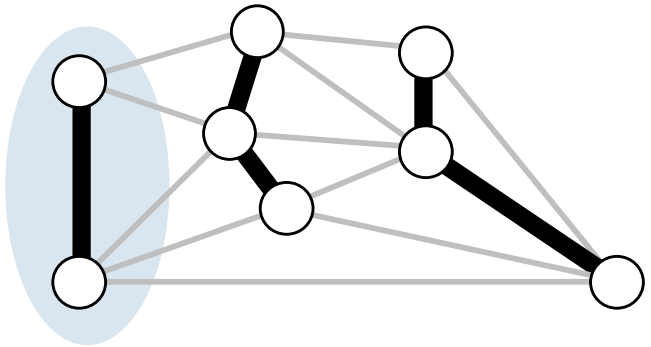
MST

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

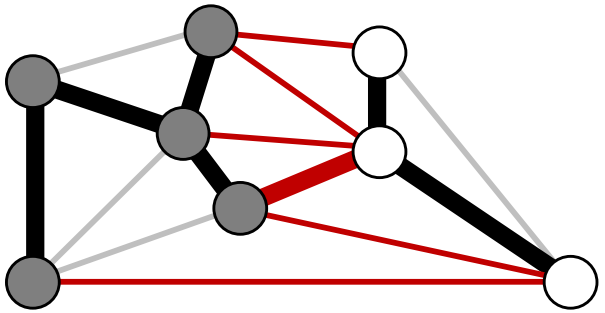
الگوریتم حریصانه: اثبات درستی

- گزاره. خروجی الگوریتم حریصانه یک درخت پوشای کمینه است.

- اثبات.
  - هر یالی که سیاه می شود به درخت پوشای کمینه تعلق دارد (بر اساس ویژگی برش).
  - اگر تعداد یال های سیاه کمتر از $\mathbf{V - 1}$ باشد، یک برش بدون یال متقاطع سیاه وجود دارد. (برشی را در نظر بگیرید که رئوس آن یک مولفه همبند تشکیل می دهند)



تعداد یال های سیاه کمتر از $\mathbf{V - 1}$



یک برش بدون یال متقاطع سیاه

الگوریتم حریصانه: پیاده‌سازی کارا

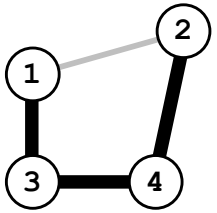
- گزاره. خروجی الگوریتم زیر، یک درخت پوشای کمینه است.
  - با یال‌هایی که در ابتدا همگی خاکستری هستند، شروع کن.
  - هر بار یک برش بدون یال متقاطع سیاه پیدا کن و یال با کمترین وزن آن را سیاه کن.
  - عمل بالا را آن قدر تکرار کن تا $\mathbf{V - 1}$ یال سیاه شوند.

- پیاده‌سازی کارا. انتخاب برش چگونه باشد؟ یال با وزن کمینه چگونه پیدا شود؟
  - مثال ۱. الگوریتم کروسکال [با ما باشید]
  - مثال ۲. الگوریتم پریم [با ما باشید]
  - مثال ۳. الگوریتم برووکا

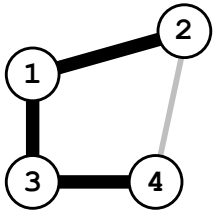
برداشتن فرضیات ساده کننده

- س. اگر وزن همه یال‌ها متفاوت نباشند، چه می‌شود؟
- ج. الگوریتم حریصانه با وجود یال‌هایی که دارای وزن برابر هستند، هنوز به درستی کار می‌کند!

(اثبات ارائه شده نیاز به اصلاح دارد)

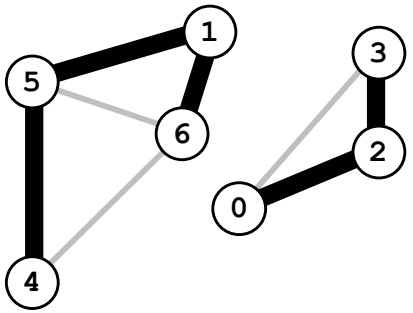



```
1   2   1.00
1   3   0.50
2   4   1.00
3   4   0.50
```




```
1   2   1.00
1   3   0.50
2   4   1.00
3   4   0.50
```

- س. اگر گراف همبند نباشد چه می‌شود؟
- ج. خروجی یک جنگل پوشای کمینه است = درخت پوشای کمینه برای هر مؤلفه.



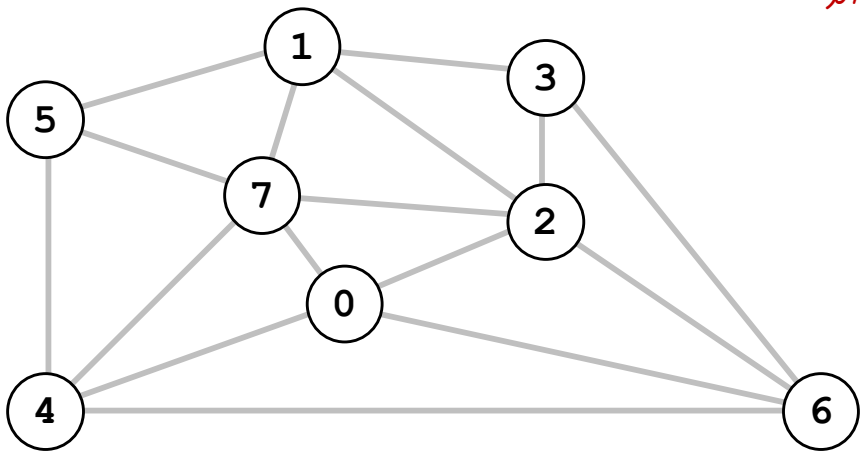

```
4   5   0.61
4   6   0.62
5   6   0.88
1   5   0.11
2   3   0.35
0   3   0.60
1   6   0.10
0   2   0.22
```

الگوریتم کروسکال

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

یال های گراف که بر اساس وزن مرتب شده اند

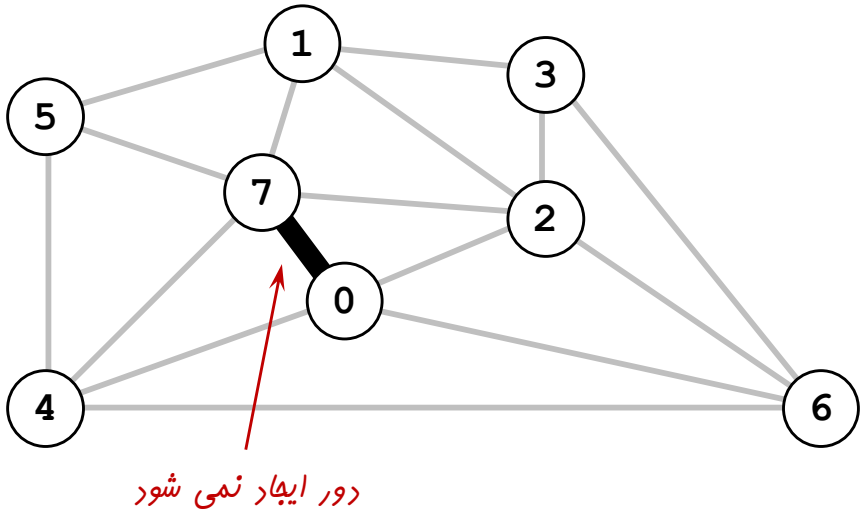



```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37
0-4   0.38
6-2   0.40
3-6   0.52
6-0   0.58
6-4   0.93
```

الگوریتم کروسکال: اجرای نمایشی

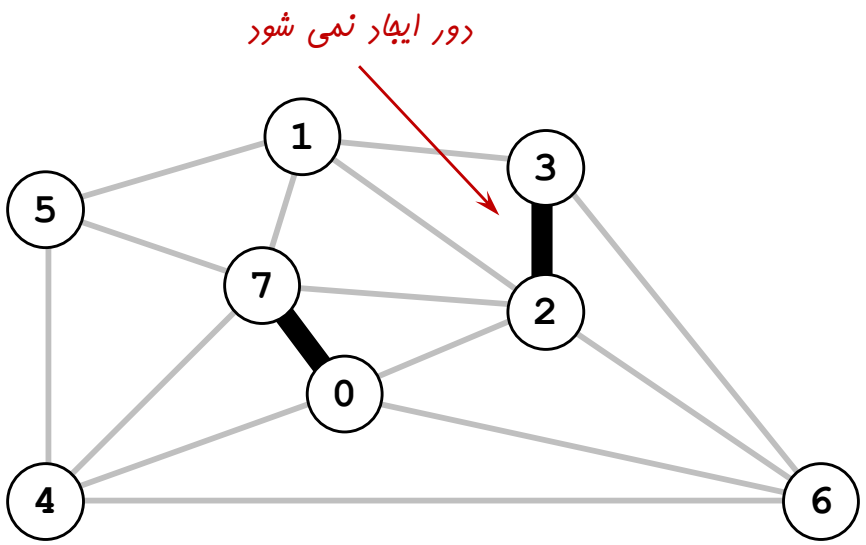
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

0-7 0.16 ⟵ در **MST**



الگوریتم کروسکال: اجرای نمایشی

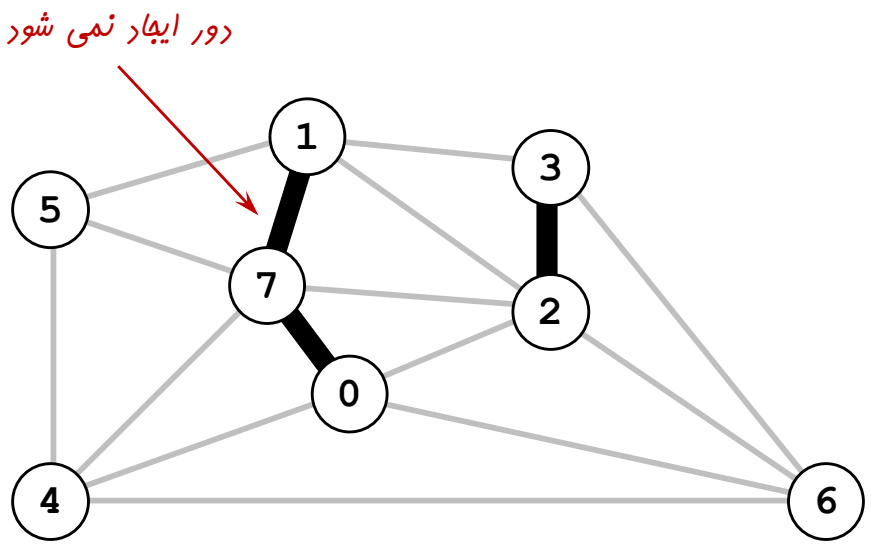
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.



MST در ⟵
0-7 0.16
2-3 0.17

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

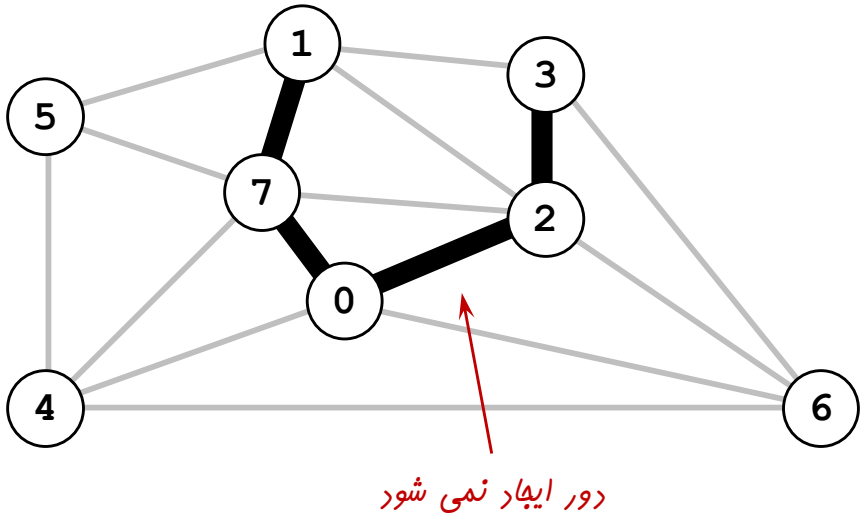


در MST →

```
0-7  0.16
2-3  0.17
1-7  0.19
```

الگوریتم کروسکال: اجرای نمایشی

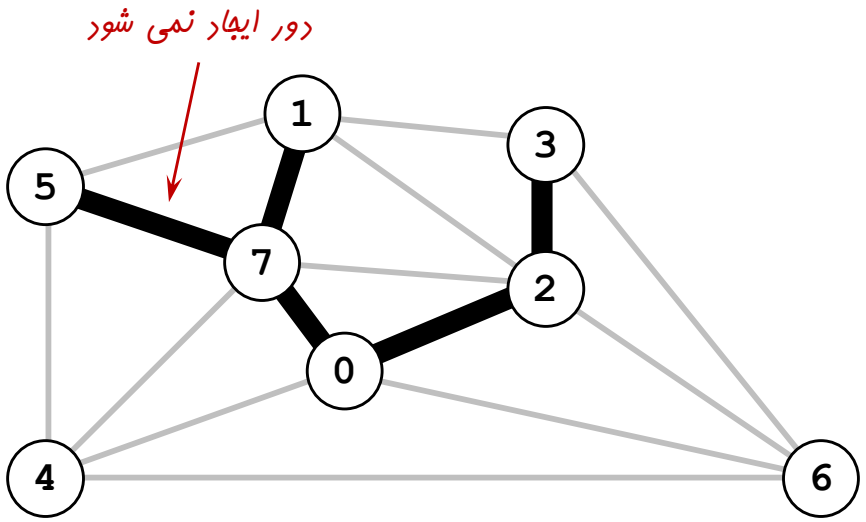
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26   ← در MST
```

الگوریتم کروسکال: اجرای نمایشی

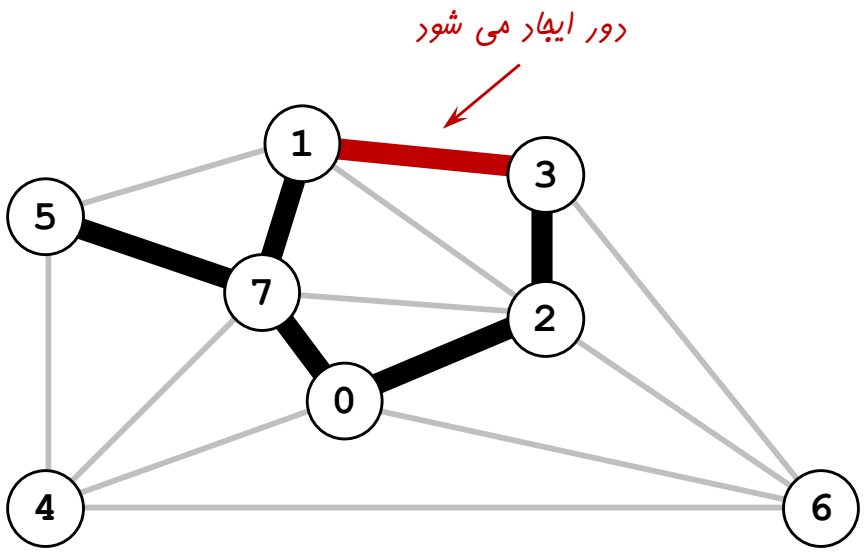
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28   ← در MST
```

الگوریتم کروسکال: اجرای نمایشی

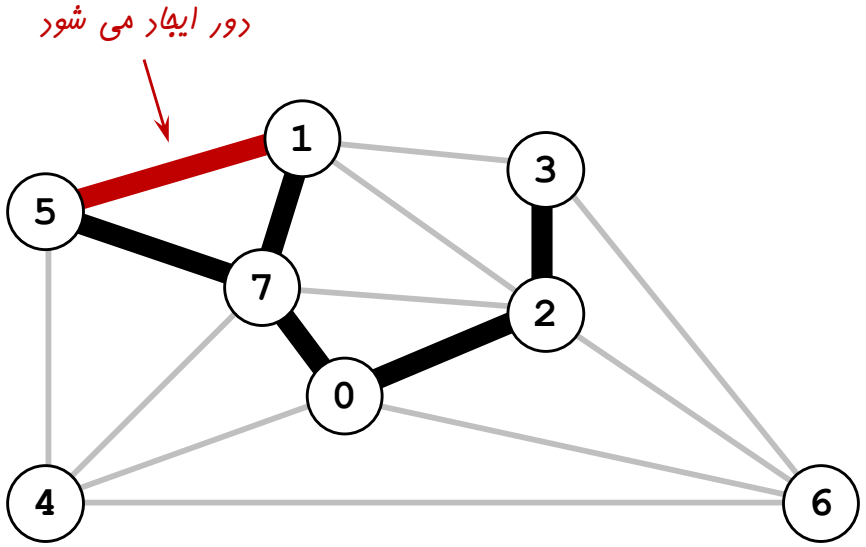
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

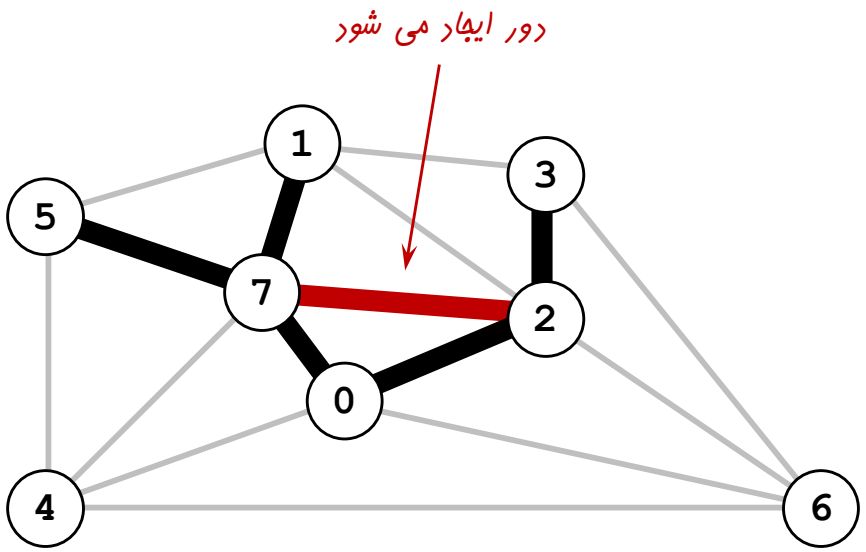
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

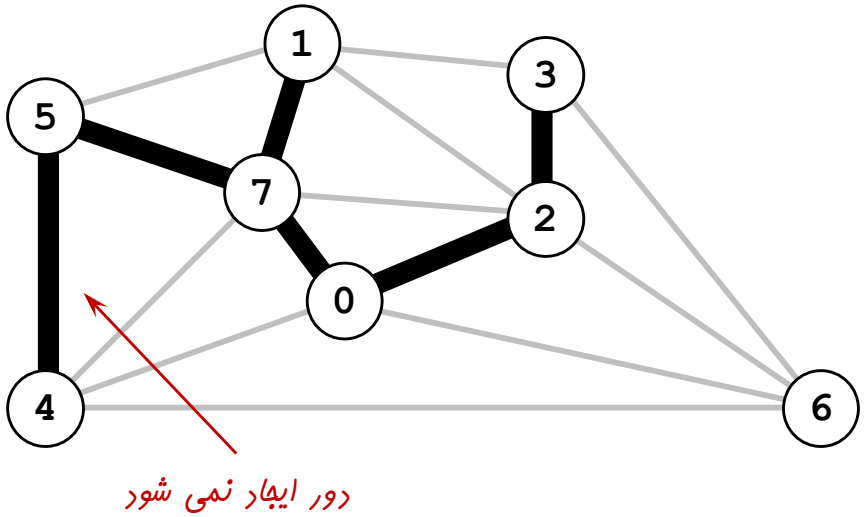
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

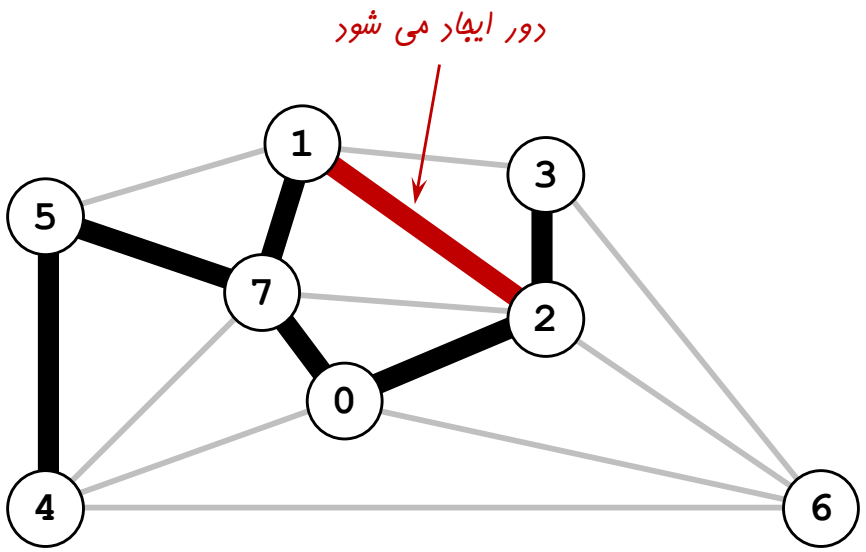
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.



	0-7	0.16
	2-3	0.17
	1-7	0.19
	0-2	0.26
	5-7	0.28
	1-3	0.29
	1-5	0.32
	2-7	0.34
در MST ←	4-5	0.35

الگوریتم کروسکال: اجرای نمایشی

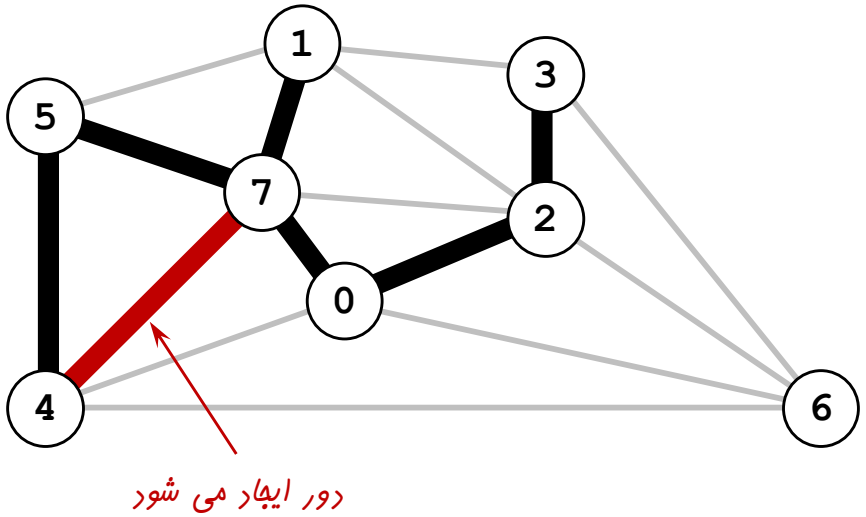
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

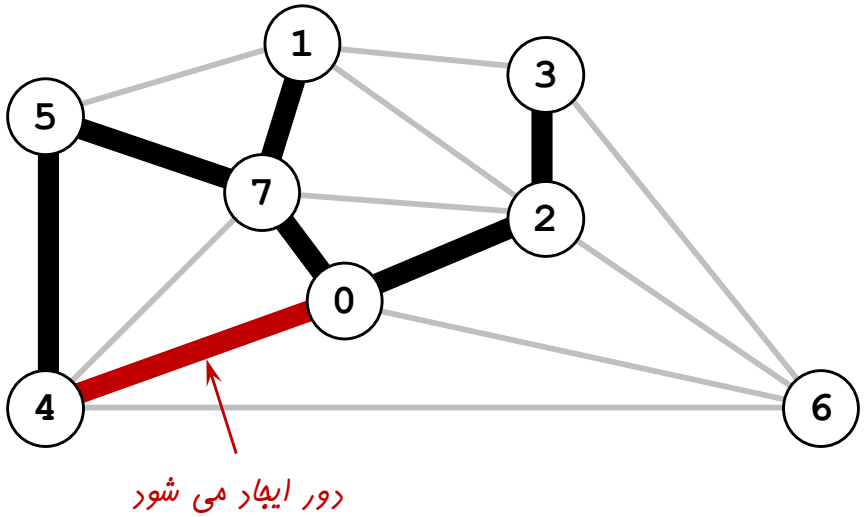
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

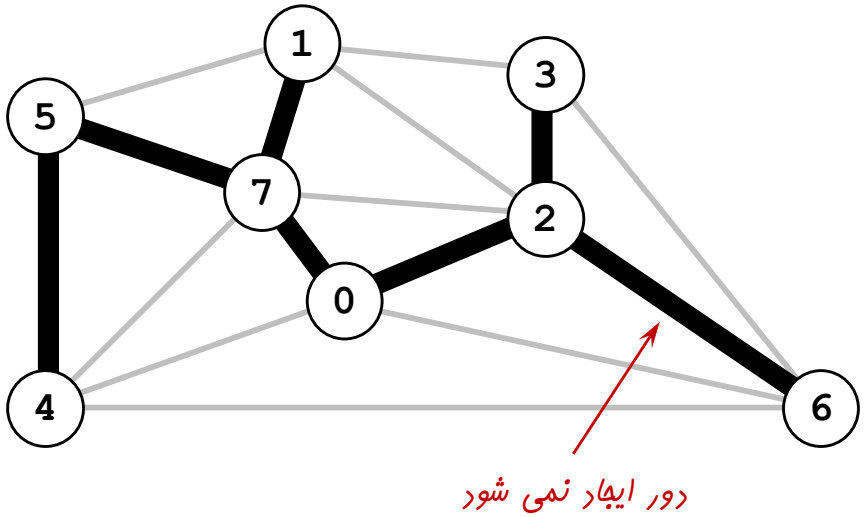
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.




```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37
0-4   0.38   → در MST نیست
```

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

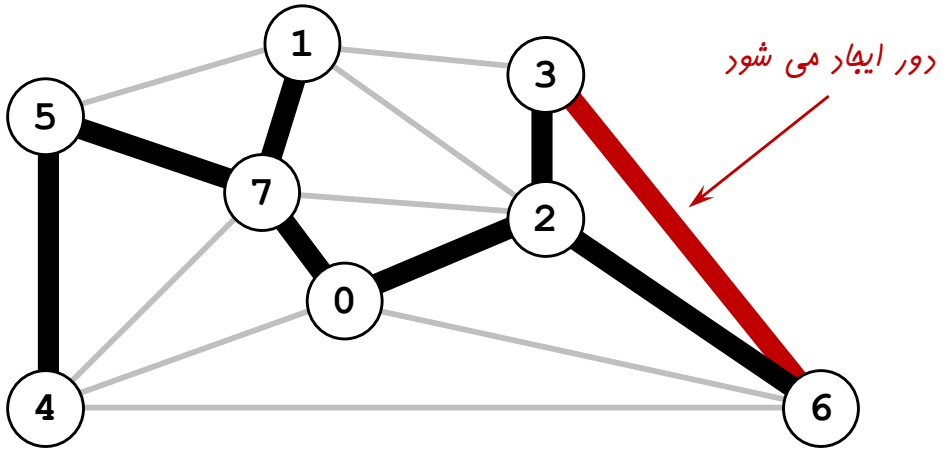


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40

در MST →

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

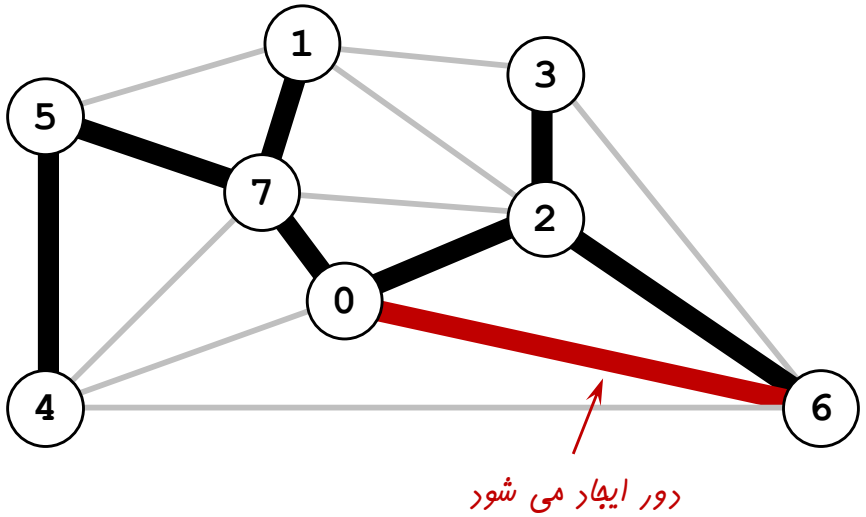


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52

→ در **MST** نیست

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

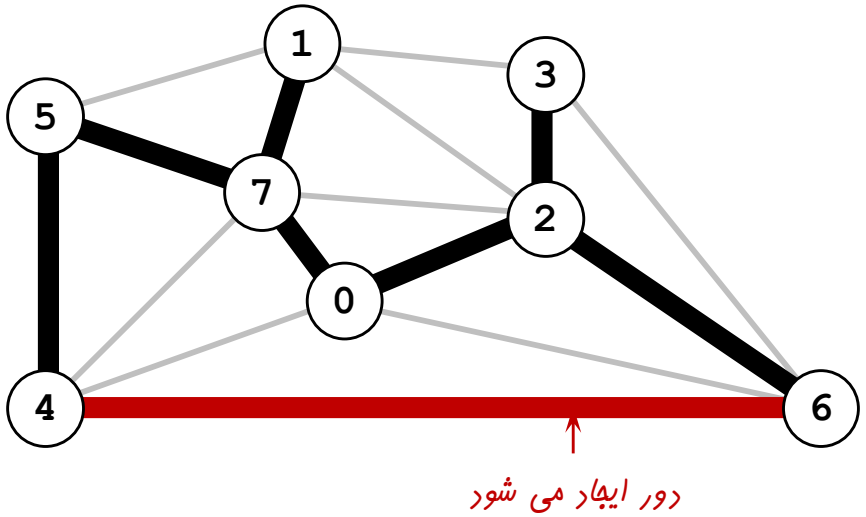


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58

→ در **MST** نیست

الگوریتم کروسکال: اجرای نمایشی

- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.

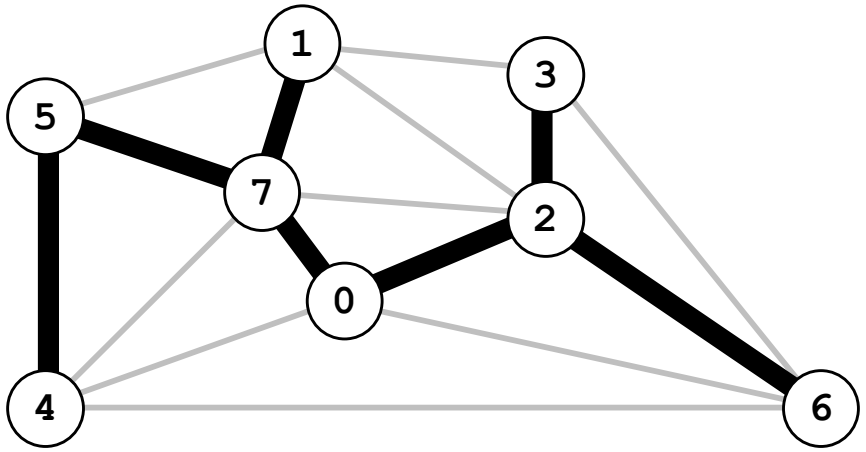


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

→ در **MST** نیست

الگوریتم کروسکال: اجرای نمایشی

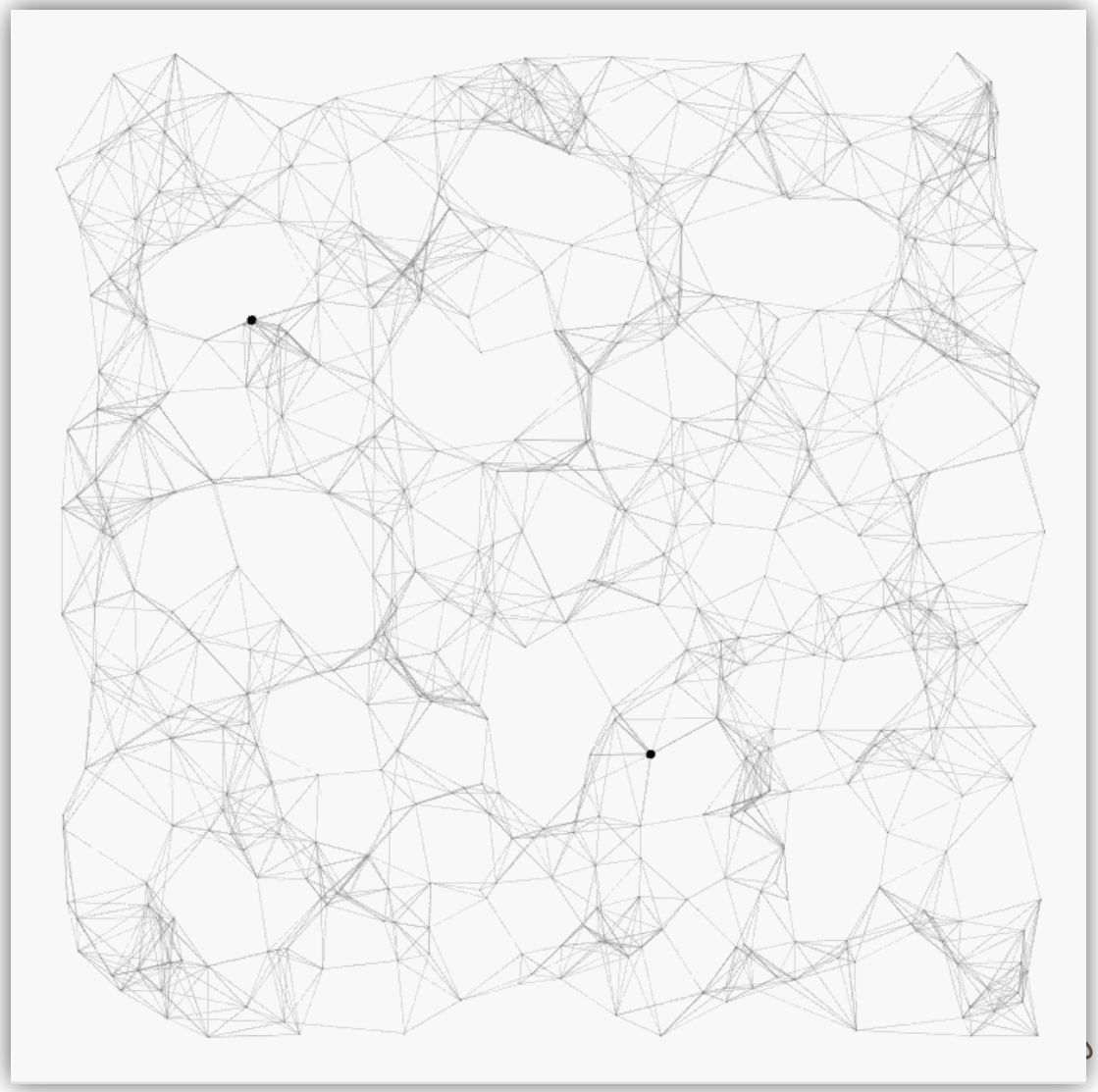
- الگوریتم کروسکال. [کروسکال، ۱۹۵۶]
  - یال‌ها را به ترتیب صعودی وزن در نظر بگیر.
  - کم‌وزن‌ترین یال باقیمانده را به درخت **T** اضافه کن؛ مگر آنکه انجام این کار باعث ایجاد دور شود.



درخت پوشای کمینه

```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

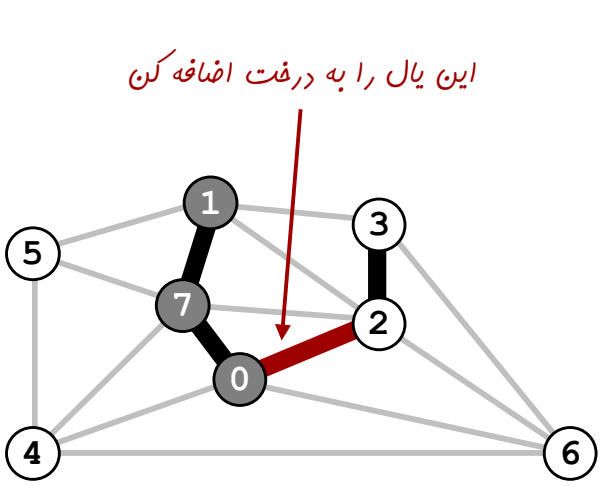
الگوریتم کروسکال: اجرای نمایشی

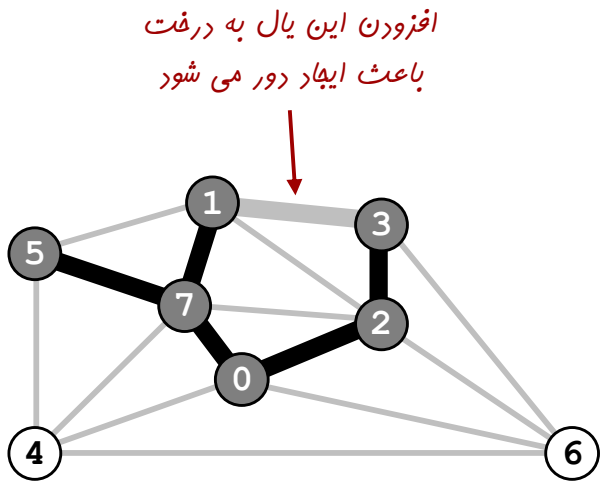


الگوریتم کروسکال: چالش پیاده سازی

- چالش. آیا افزودن یال **v-w** به درخت **T** باعث ایجاد دور می‌شود؟
- درجه سختی؟

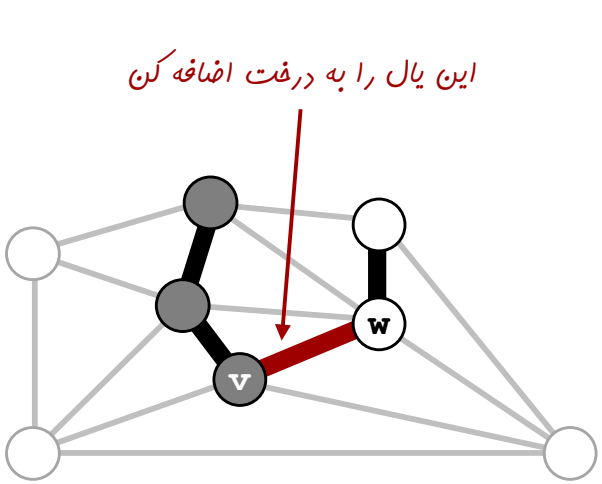
- E + V
- V ← *با شروع از رأس **v** الگوریتم **DFS** را اجرا کن، بررسی کن آیا رأس **w** قابل دسترسی است*
- log V
- log* V ← *با استفاده از ساختمان داده مجموعه های مجزا !*
- 1

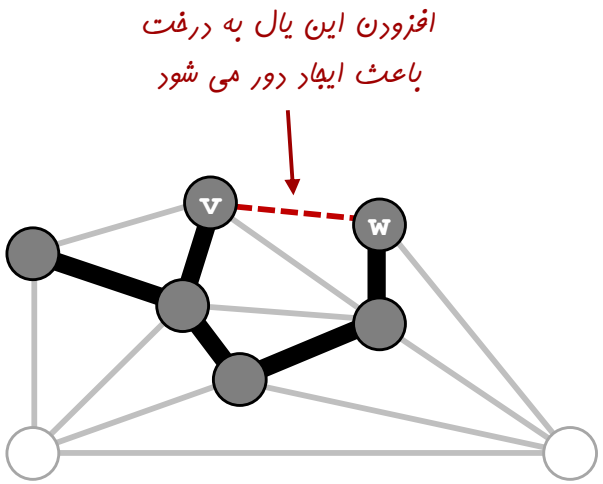




الگوریتم کروسکال: چالش پیاده‌سازی

- چالش. آیا افزودن یال **v-w** به درخت **T** باعث ایجاد دور می‌شود؟
- را ه‌حل کارا. استفاده از ساختمان داده مجموعه‌های مجزا
  - هر مؤلفه همبند در **T** را در یک مجموعه مجزا نگهداری کن.
  - اگر دو رأس **v** و **w** در یک مجموعه باشند، آنگاه افزودن یال **v-w** باعث ایجاد دور می‌شود.
  - پس از افزودن یال **v-w** به **T**، مجموعه‌های شامل دو رأس **v** و **w** را با یکدیگر ادغام کن.





الگوریتم کروسکال: پیاده سازی در جاوا


```
public class KruskalMST
{
    private Queue<Edge> mst = new Queue<Edge>();

    public KruskalMST (EdgeWeightedGraph G)
    {
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges())
            pq.insert(e);

        UF uf = new UF(G.V());
        while (!pq.isEmpty() && mst.size() < G.V() - 1)
        {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (!uf.connected(v, w))
            {
                uf.union(v, w);
                mst.enqueue(e);
            }
        }
    }

    public Iterable<Edge> edges() { return mst; }
}
```

الگوریتم کروسکال: زمان اجرا

- گزاره. الگوریتم کروسکال در بدترین حالت درخت پوشای کمینه را در زمانی متناسب با E log E محاسبه می‌کند.

- اثبات.

عمل	فراوانی	زمان هر اجرا
ایجاد صف اولویت	1	E
حذف کوچکترین	E	log E
اجتماع	V	log* V
بررسی متصل بودن	E	log* V

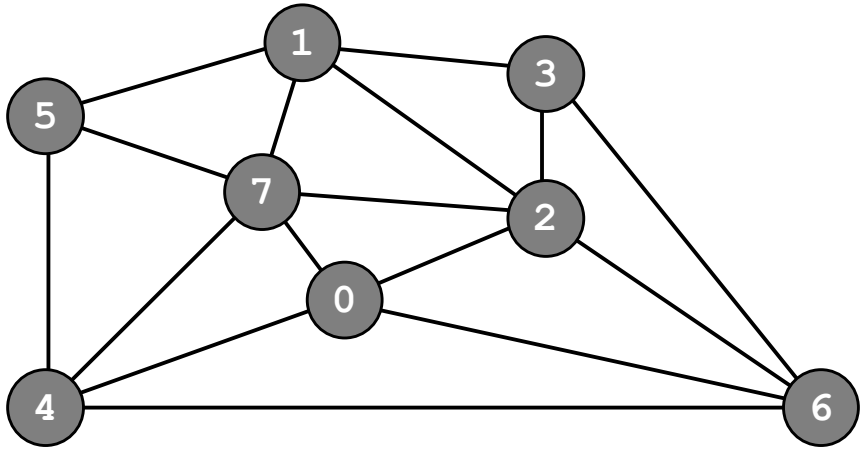
یادآوری: در دنیای واقعی log* V <= 5

- توجه. اگر یال‌ها از قبل مرتب باشند، مرتبه رشد برابر با E log* V خواهد بود.

الگوریتم پریم

الگوریتم پریم: اجرای نمایشی

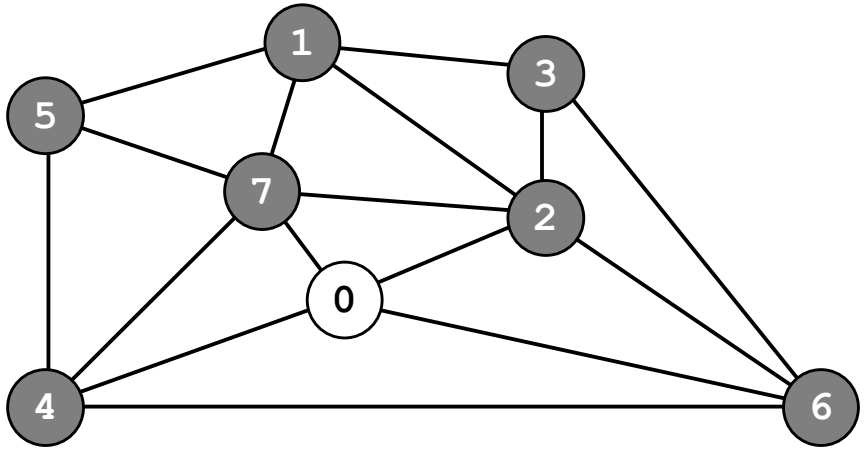
- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.




```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37
0-4   0.38
6-2   0.40
3-6   0.52
6-0   0.58
6-4   0.93
```

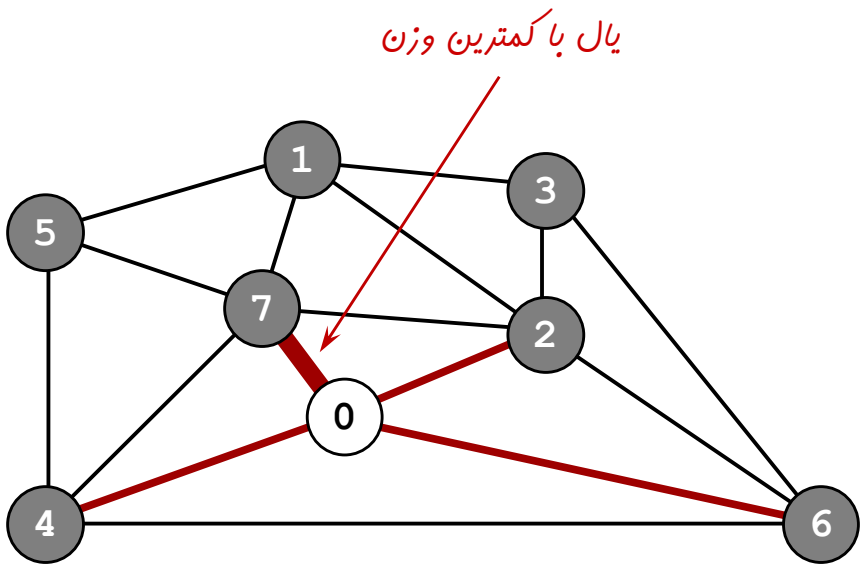
الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.



الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.



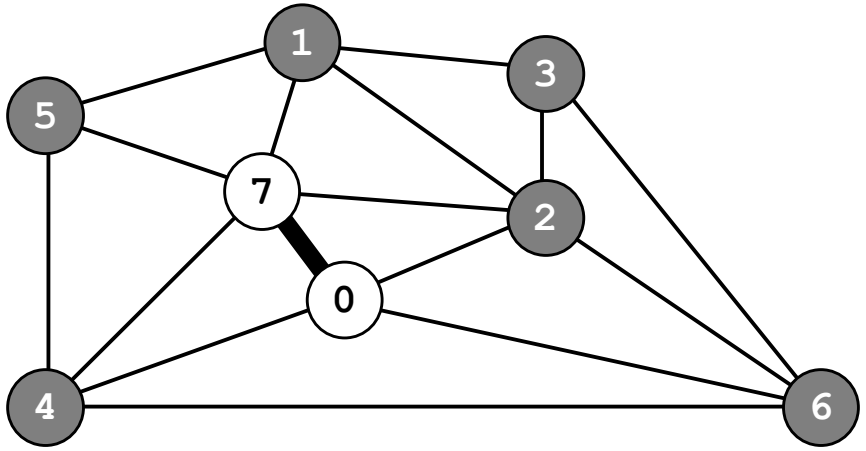
یال هایی که دقیقا یک
رأسشان در T است

MST در ⟶

0-7	0.16
0-2	0.26
0-4	0.38
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

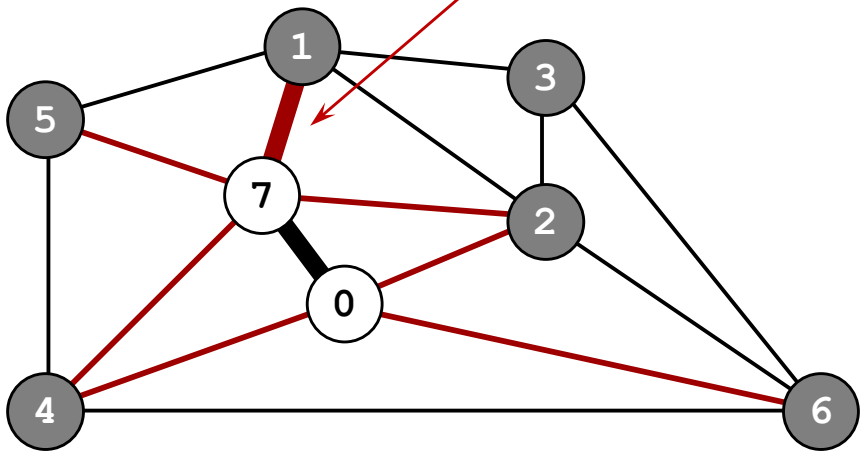


یال های MST 0-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7

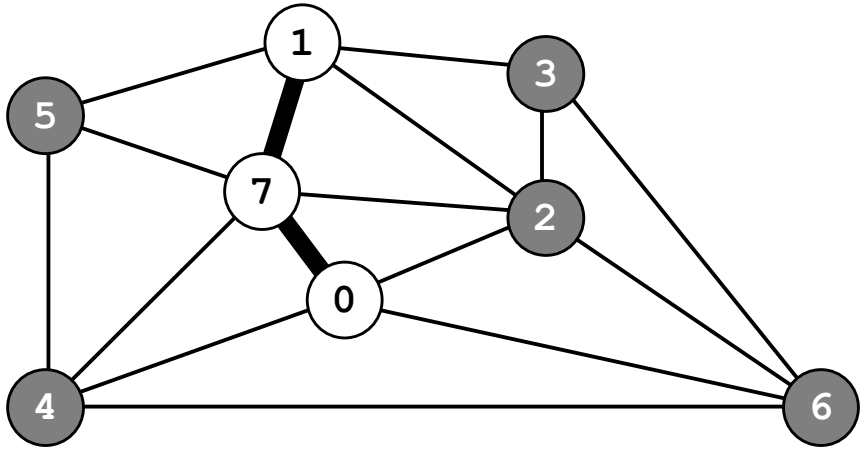
یال هایی که دقیقا یک رأسشان در T است

MST در

1-7	0.19
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

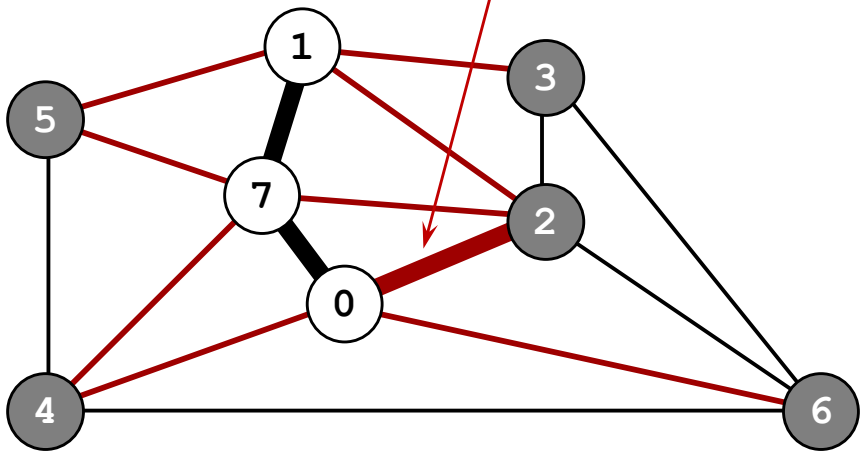


یال های MST 0-7 1-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7 1-7

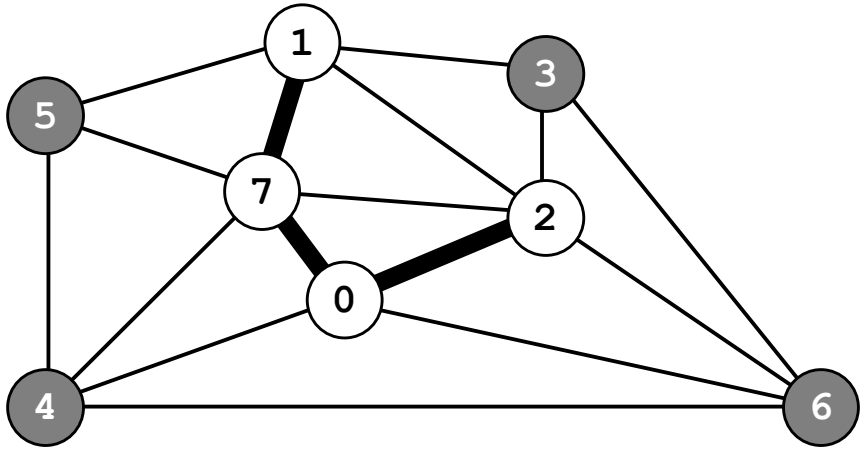
یال هایی که دقیقا یک رأسشان در T است

MST در ⟵

0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
1-2	0.36
4-7	0.37
0-4	0.38
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

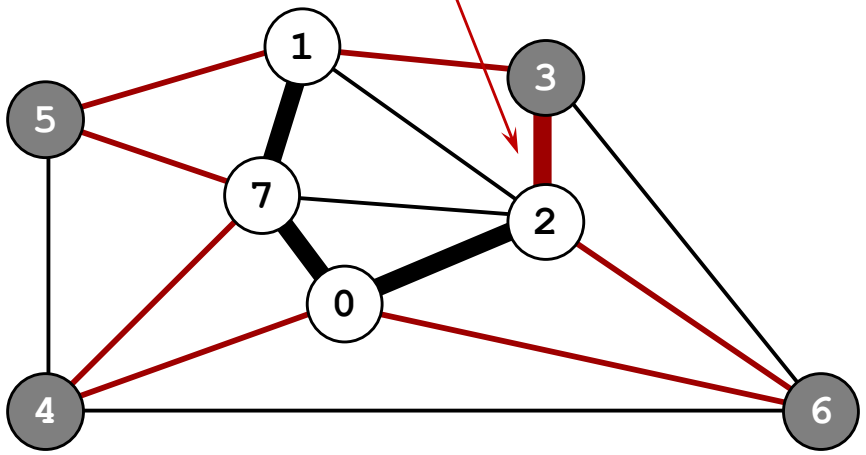


یال های **MST** 0-7 1-7 0-2

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7 1-7 0-2

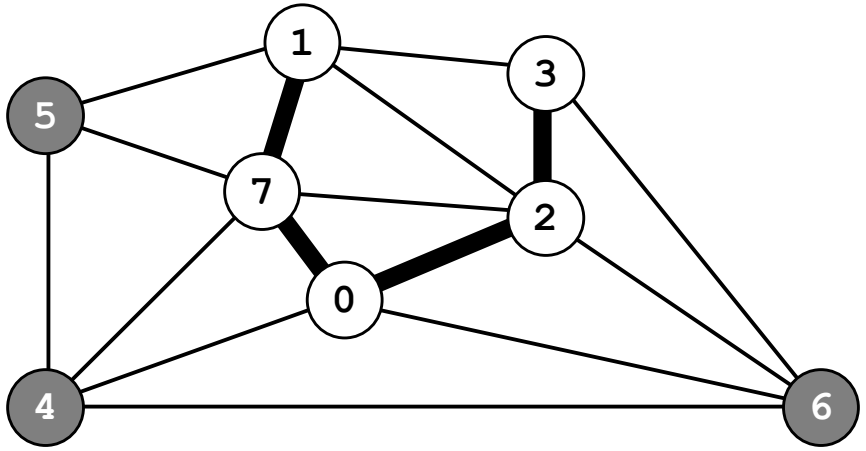
یال هایی که دقیقا یک رأسشان در T است

MST در ⟵

2-3	0.17
5-7	0.28
1-3	0.29
1-5	0.32
4-7	0.37
0-4	0.38
6-2	0.40
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

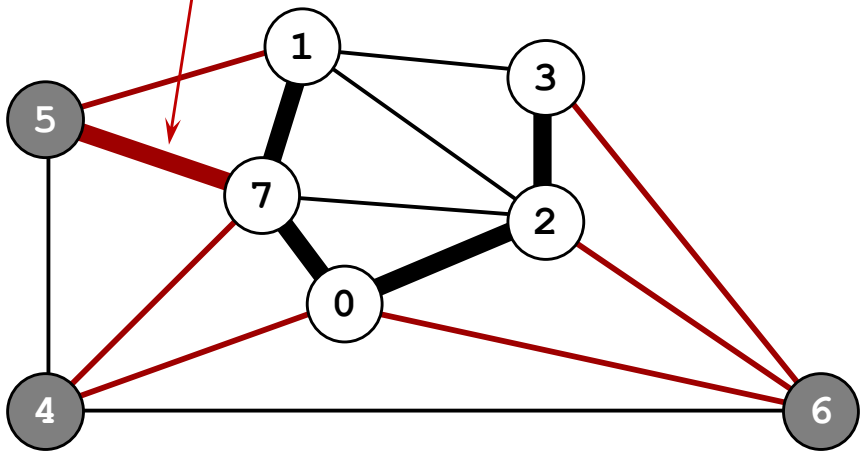


یال های MST 0-7 1-7 0-2 2-3

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7 1-7 0-2 2-3

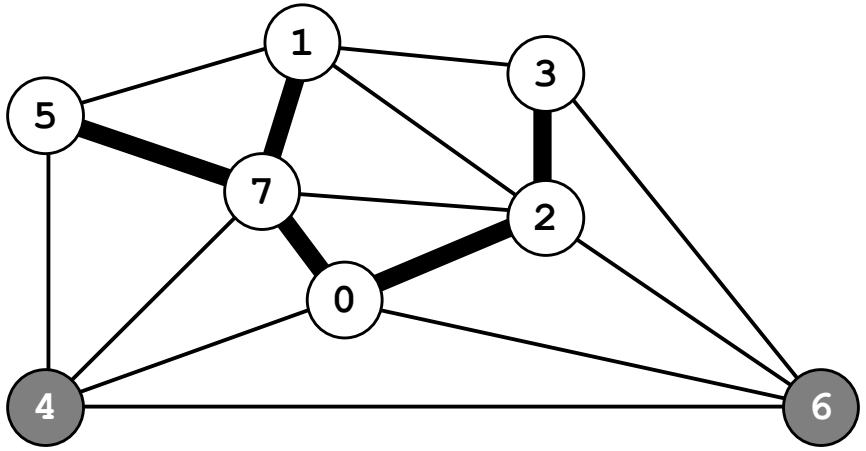
یال هایی که دقیقا یک رأسشان در T است

MST در →

5-7	0.28
1-5	0.32
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

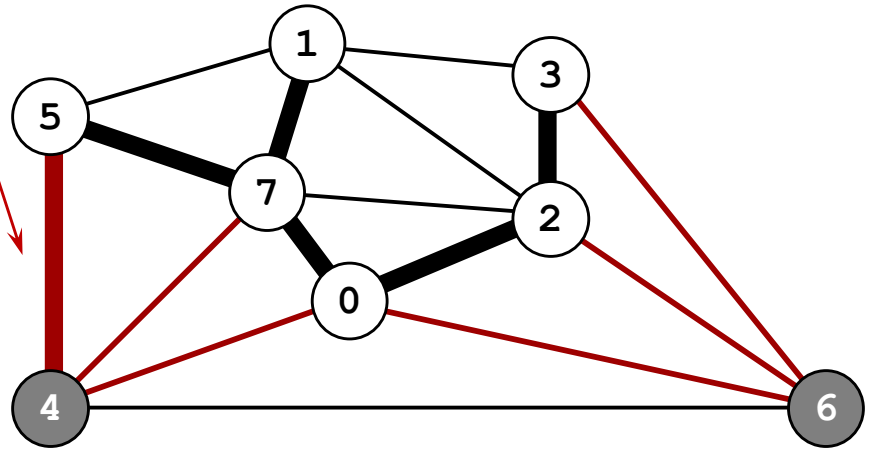


یال های MST 0-7 1-7 0-2 2-3 5-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7 1-7 0-2 2-3 5-7

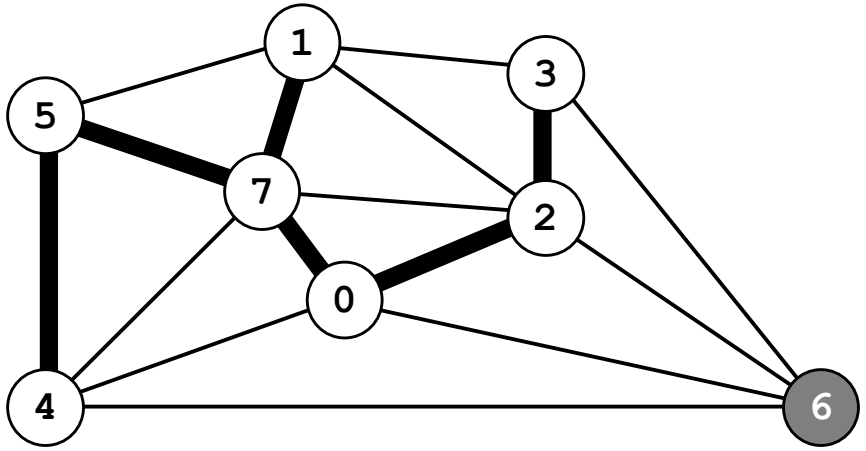
یال هایی که دقیقا یک رأسشان در T است

MST در →

4-5	0.35
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

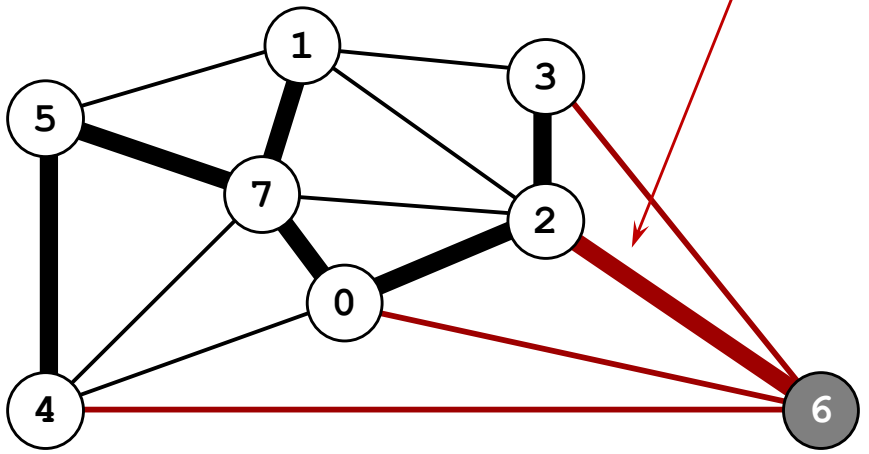


یال های MST 0-7 1-7 0-2 2-3 5-7 4-5

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.

یال با کمترین وزن



یال های MST 0-7 1-7 0-2 2-3 5-7 4-5

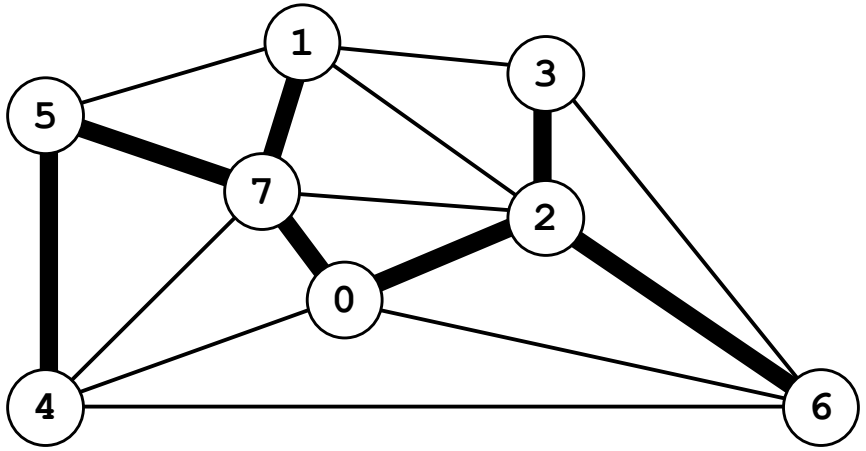
یال هایی که دقیقا یک رأسشان در T است

MST در →

6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

الگوریتم پریم: اجرای نمایشی

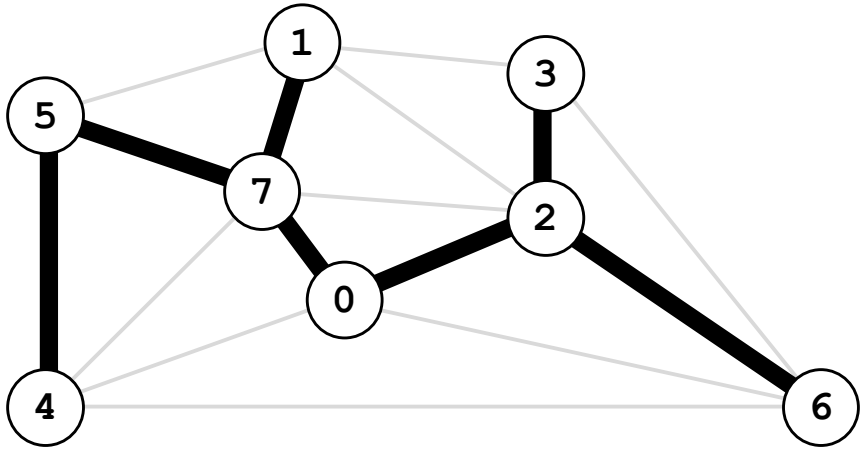
- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.



یال های MST 0-7 1-7 0-2 2-3 5-7 4-5 6-2

الگوریتم پریم: اجرای نمایشی

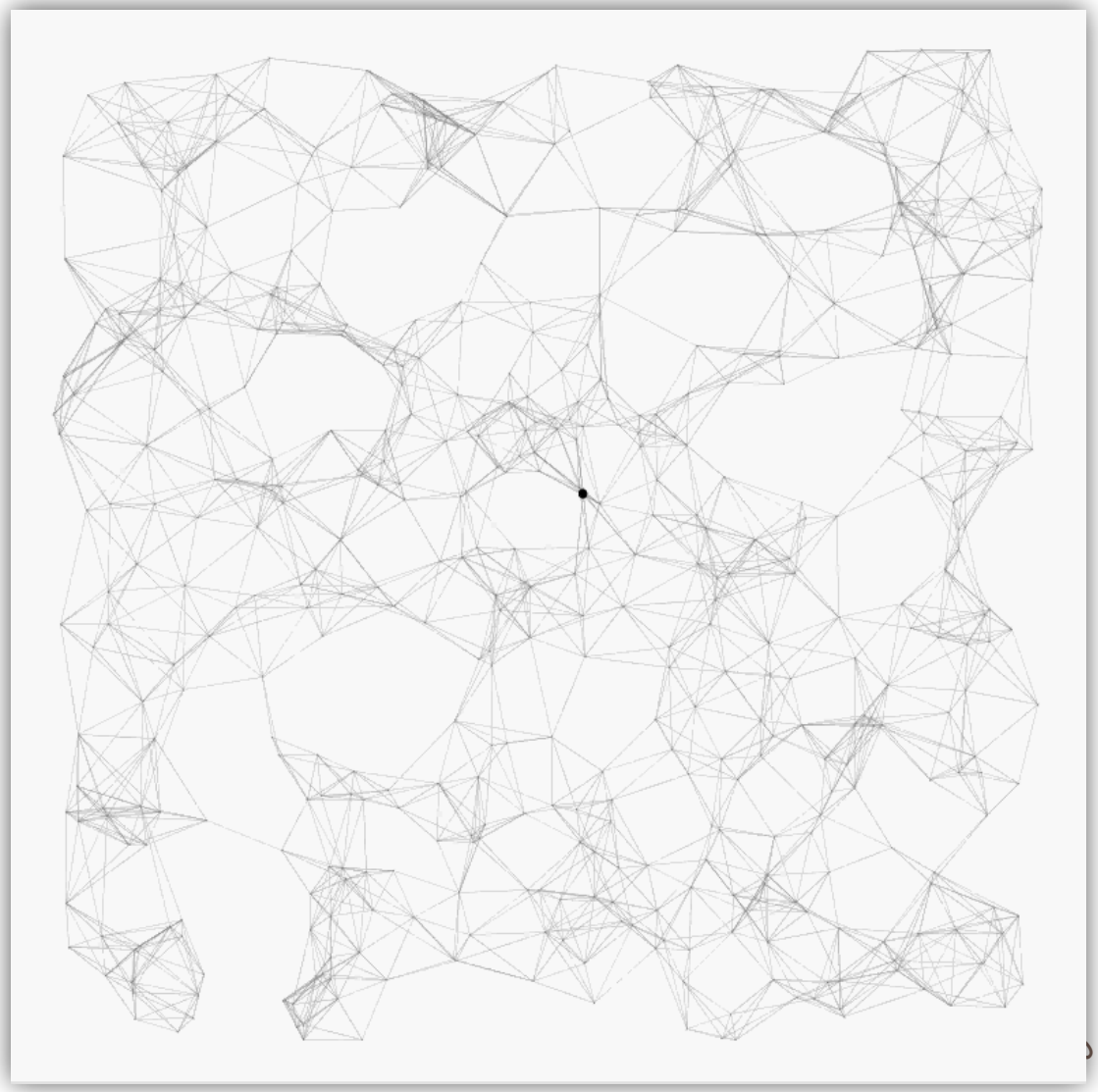
- الگوریتم پریم. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن.
  - در هر مرحله، یال با کمترین وزن را که دقیقاً یک رأس آن در T است، به T اضافه کن.



درخت پوشای کمینه

یال های MST 0-7 1-7 0-2 2-3 5-7 4-5 6-2

الگوریتم پریم: اجرا

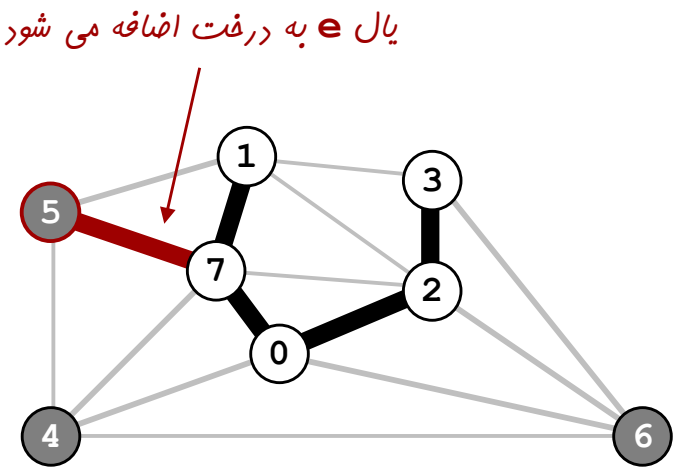


الگوریتم پریم: اثبات درستی

- گزاره. [یارنیک ۱۹۳۰، دیکسترا ۱۹۵۷، پریم ۱۹۵۹]

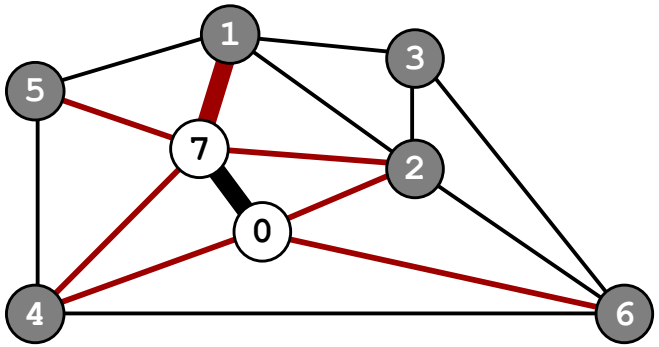
  خروجی الگوریتم پریم یک درخت پوشای کمینه است.

- اثبات. الگوریتم پریم یک حالت خاص از الگوریتم حریصانه محاسبه MST است.
  - فرض کنید e یالی باشد که یک رأس از درخت را به یک رأس دیگر که به درخت تعلق ندارد متصل می‌کند.
  - برش = مجموعه رئوس درخت.
  - هیچ یال متقاطعی سیاه نیست.
  - وزن هیچ یال متقاطعی از e کمتر نیست.



الگوریتم پریم: چالش های پیاده‌سازی

- چالش. یافتن یال با کمترین وزن که دقیقاً یک سرش در T باشد.
- درجه سختی.
  - E ← بررسی تمام یال ها
  - V
  - log E ← استفاده از یک صف اولویت
  - log* E
  - 1

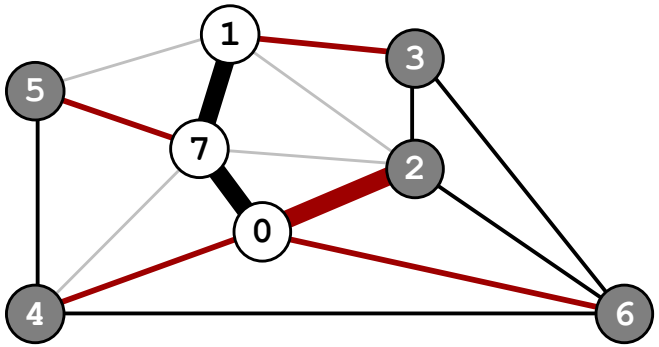


یال **1-7** یال با کمترین وزن است
که دقیقا یک سرش در **T** است

1-7	**0.19**
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

الگوریتم پریم: پیاده‌سازی مشتاق

- چالش. یافتن یال با کمترین وزن که دقیقاً یک سرش در T باشد.
- راه‌حل مشتاق. رئوسی را که به وسیله یک یال به T متصل هستند در یک صف اولویت نگهداری کن، به طوری که اولویت رأس v برابر است با وزن کوتاه‌ترین یالی که v را به T وصل می‌کند.
  - نزدیک‌ترین رأس (v) به درخت را از صف حذف و یال مرتبط با آن را (یال e = v-w) به درخت اضافه کن.
  - صف اولویت را با در نظر گرفتن همه یال‌های متلاقی با v (به صورت e = v-x) به روز رسانی کن
    - اگر x در صف اولویت قرار دارد، آن را نادیده بگیر
    - در غیر این صورت x را به صف اولویت اضافه کن
    - اگر v-x کوتاه‌ترین یالی باشد که x را به درخت وصل می‌کند، اولویت x را افزایش بده



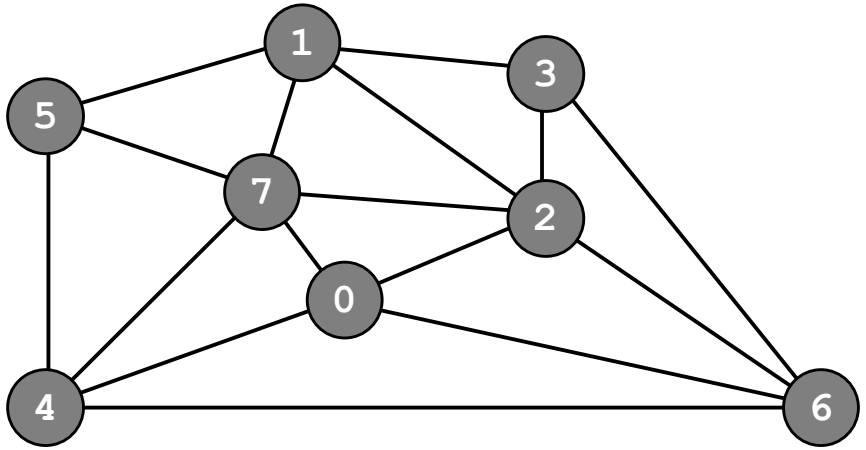

```
0
1   1-7   0.19
2   0-2   0.26
3   1-3   0.29
4   0-4   0.38
5   5-7   0.28
6   6-0   0.58
7   0-7   0.16
```

قرمز: در PQ

سیاه: در MST

الگوریتم پریم: اجرای نمایشی

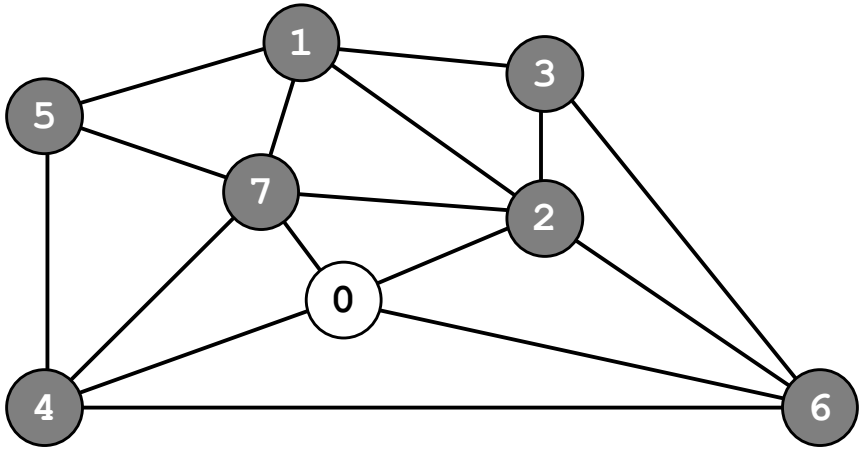
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

الگوریتم پریم: اجرای نمایشی

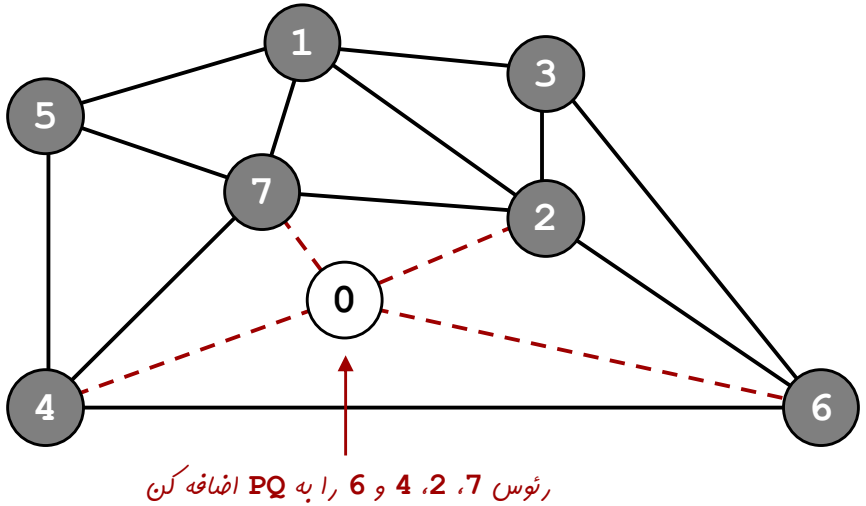
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

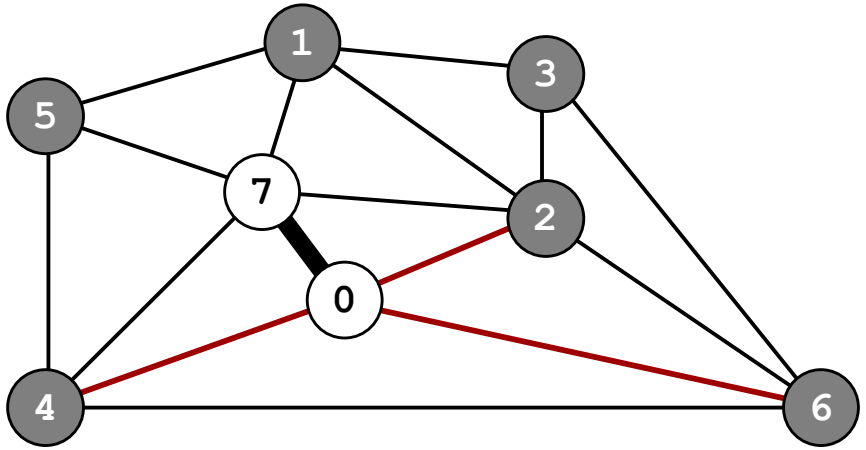


v	edgeTo[]	distTo[]
0	–	–
7	0-7	0.16
2	0-2	0.26
4	0-4	0.38
6	6-0	0.58

رئوس موجود در PQ
(مرتب بر اساس وزن)

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

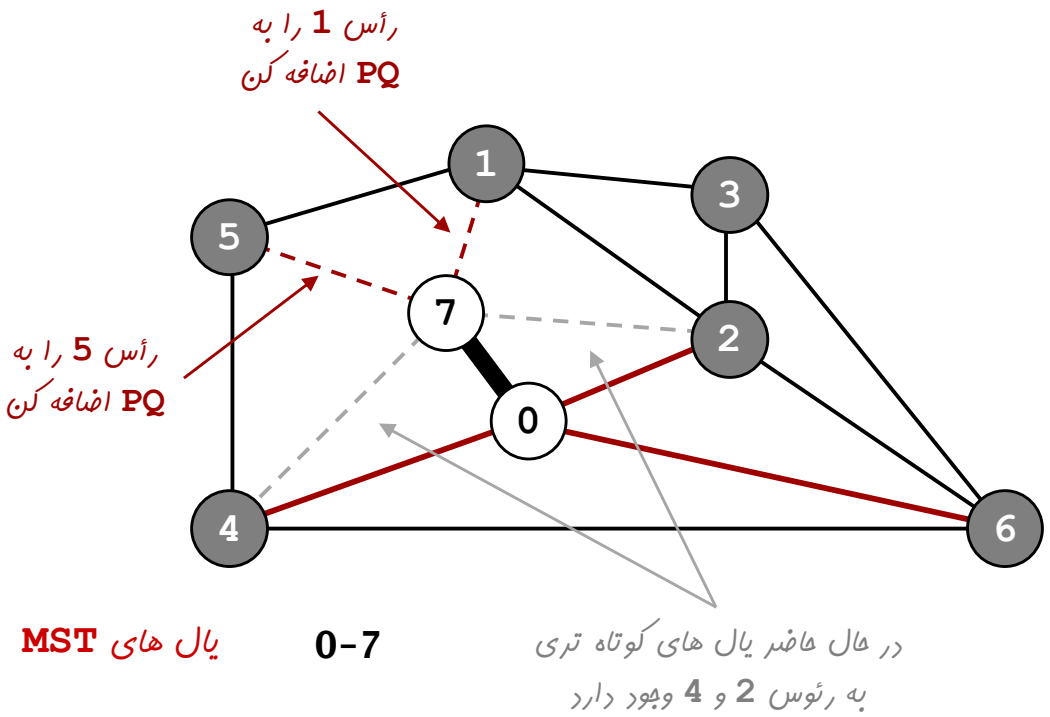


v	edgeTo[]	distTo[]
0	-	-
→ 7	0-7	0.16
2	0-2	0.26
4	0-4	0.38
6	6-0	0.58

یال های MST 0-7

الگوریتم پریم: اجرای نمایشی

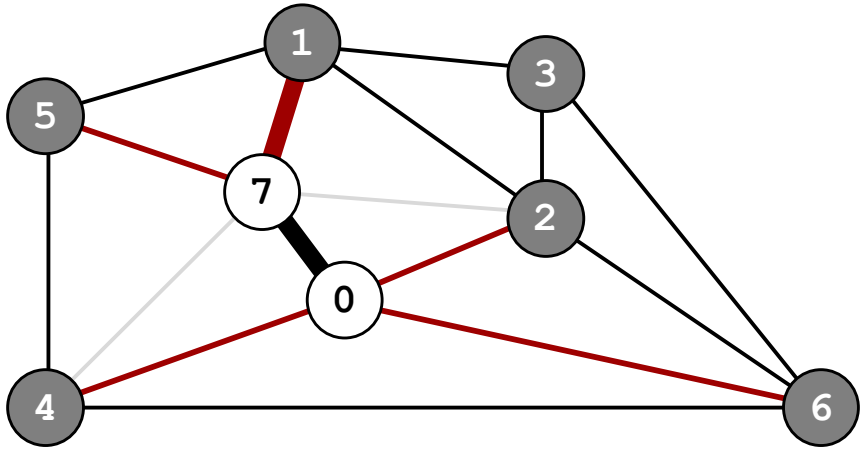
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-
→ 7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
5	5-7	0.28
4	0-4	0.38
6	6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

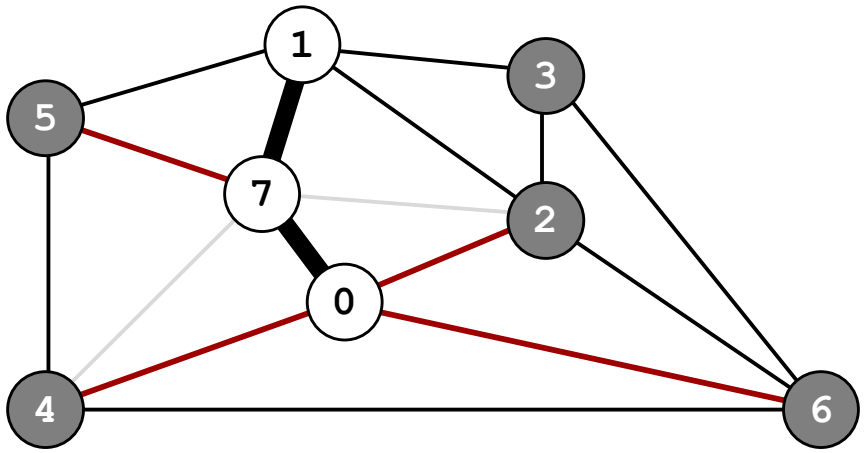


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
→ 1	1-7	0.19
2	0-2	0.26
5	5-7	0.28
4	0-4	0.38
6	6-0	0.58

یال های MST 0-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

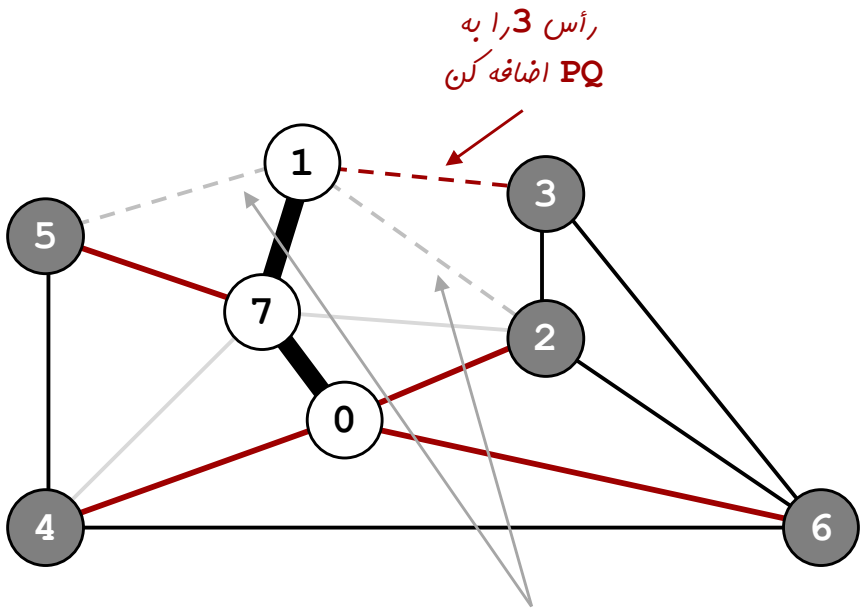


	v	edgeTo[]	distTo[]
	0	-	-
	7	0-7	0.16
→	1	1-7	0.19
	2	0-2	0.26
	5	5-7	0.28
	4	0-4	0.38
	6	6-0	0.58

یال های **MST** 0-7 1-7

الگوریتم پریم: اجرای نمایشی

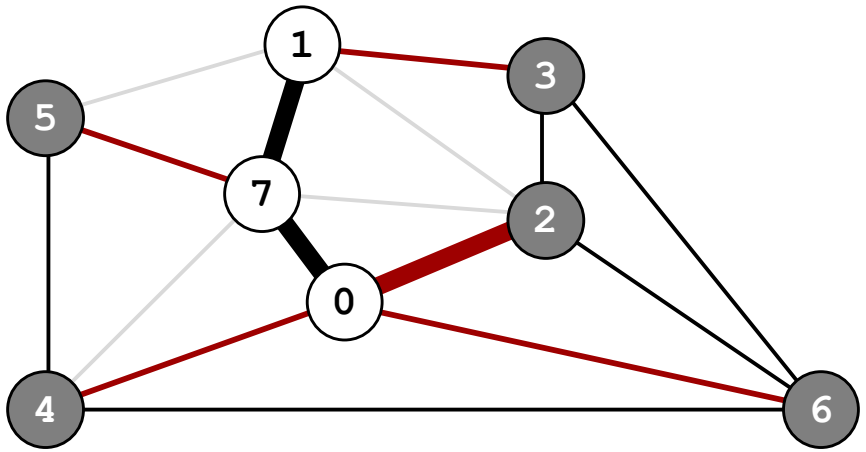
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
→ 1	1-7	0.19
2	0-2	0.26
5	5-7	0.28
(3)	1-3	0.29
4	0-4	0.38
6	6-0	0.58

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

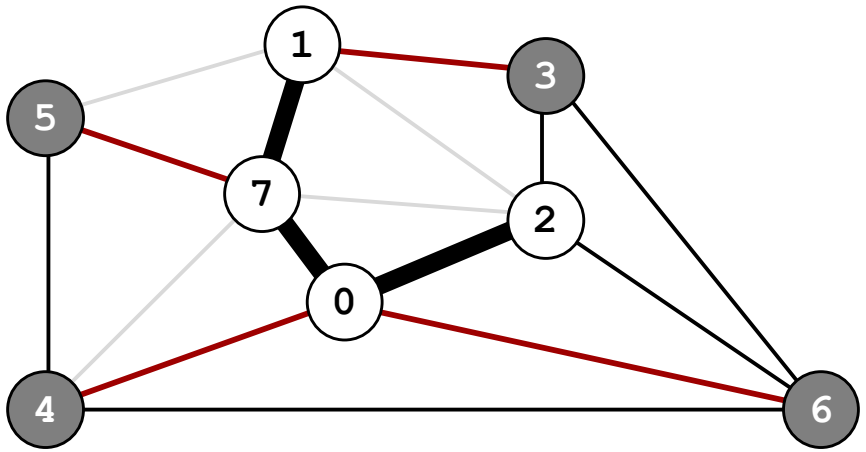


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
5	5-7	0.28
3	1-3	0.29
4	0-4	0.38
6	6-0	0.58

یال های **MST** 0-7 1-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

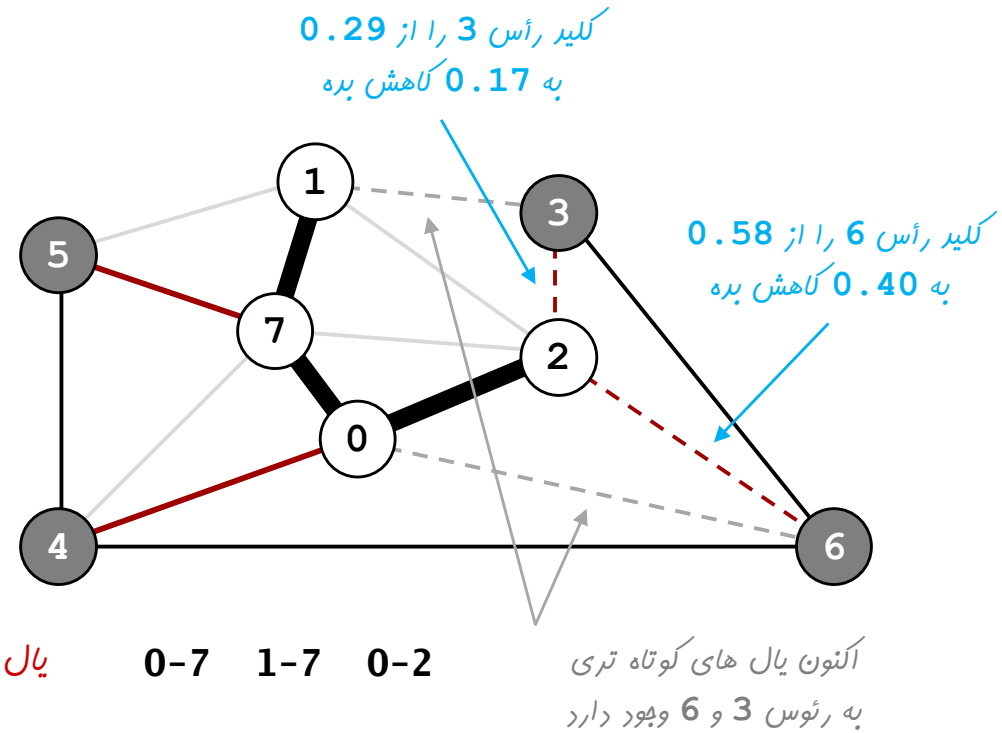


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
→ 2	0-2	0.26
5	5-7	0.28
3	1-3	0.29
4	0-4	0.38
6	6-0	0.58

یال های **MST** 0-7 1-7 0-2

الگوریتم پریم: اجرای نمایشی

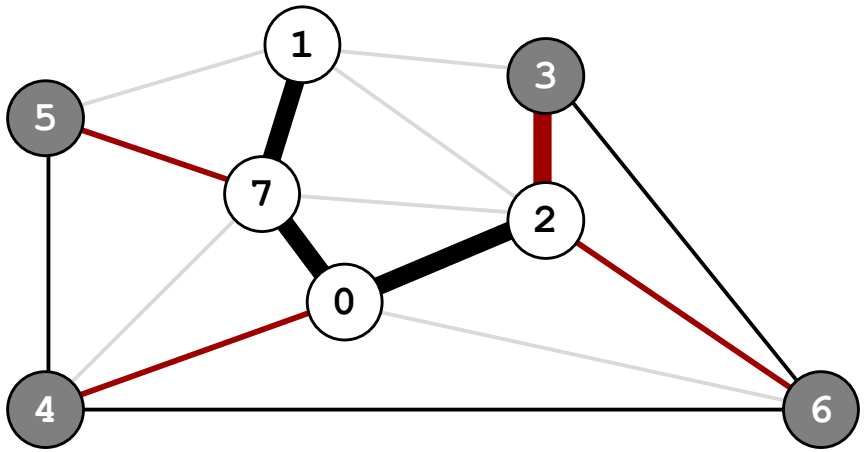
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
→ 2	0-2	0.26
	2-3	0.17
(3)	~~1-3~~	~~0.29~~
5	5-7	0.28
4	0-4	0.38
(6)	~~6-0~~	~~0.58~~
	6-2	0.40

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

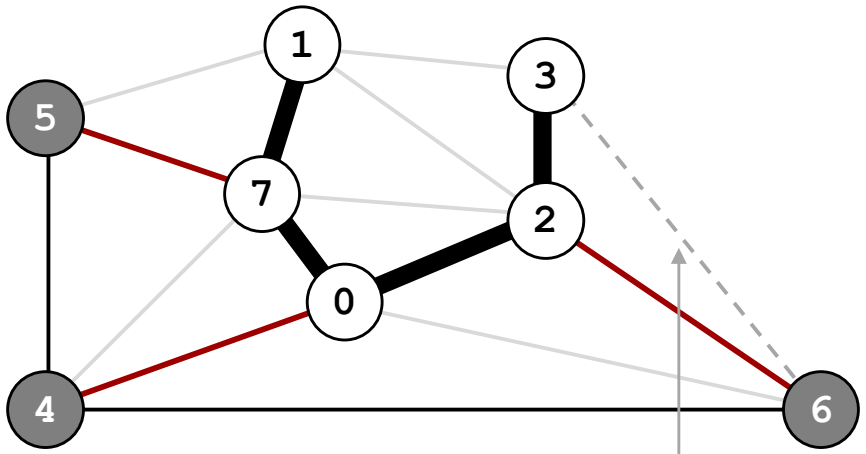


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
→ 3	2-3	0.17
5	5-7	0.28
4	0-4	0.38
6	6-2	.40

یال های **MST** 0-7 1-7 0-2

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



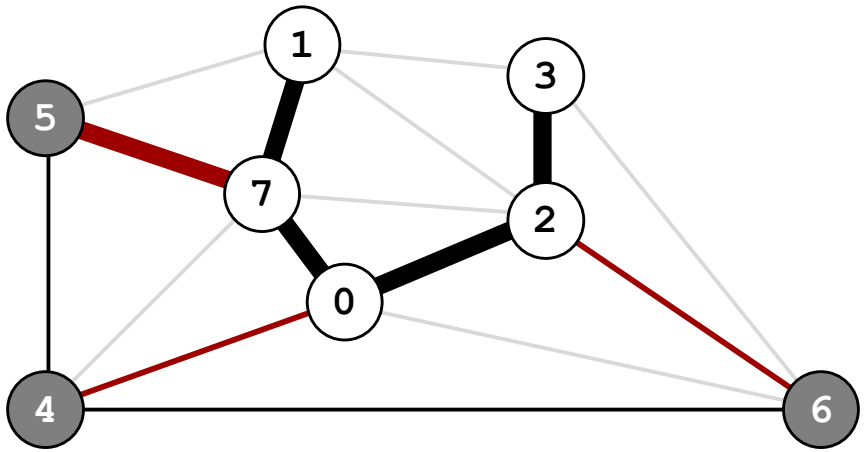
یال های MST 0-7 1-7 0-2 2-3

در حال حاضر یال کوتاه تری به رأس 6 وجود دارد

v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
→ 3	2-3	0.17
5	5-7	0.28
4	0-4	0.38
6	6-2	0.40

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

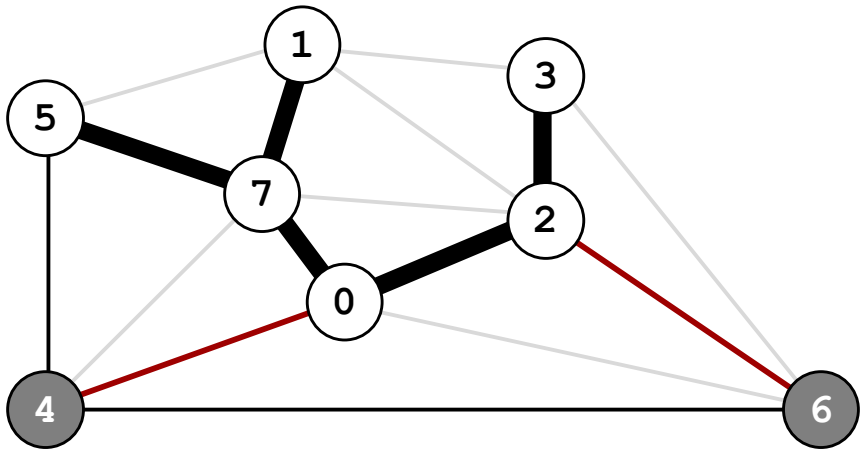


یال های MST 0-7 1-7 0-2 2-3

v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
→ 5	5-7	0.28
4	0-4	0.38
6	6-2	0.40

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

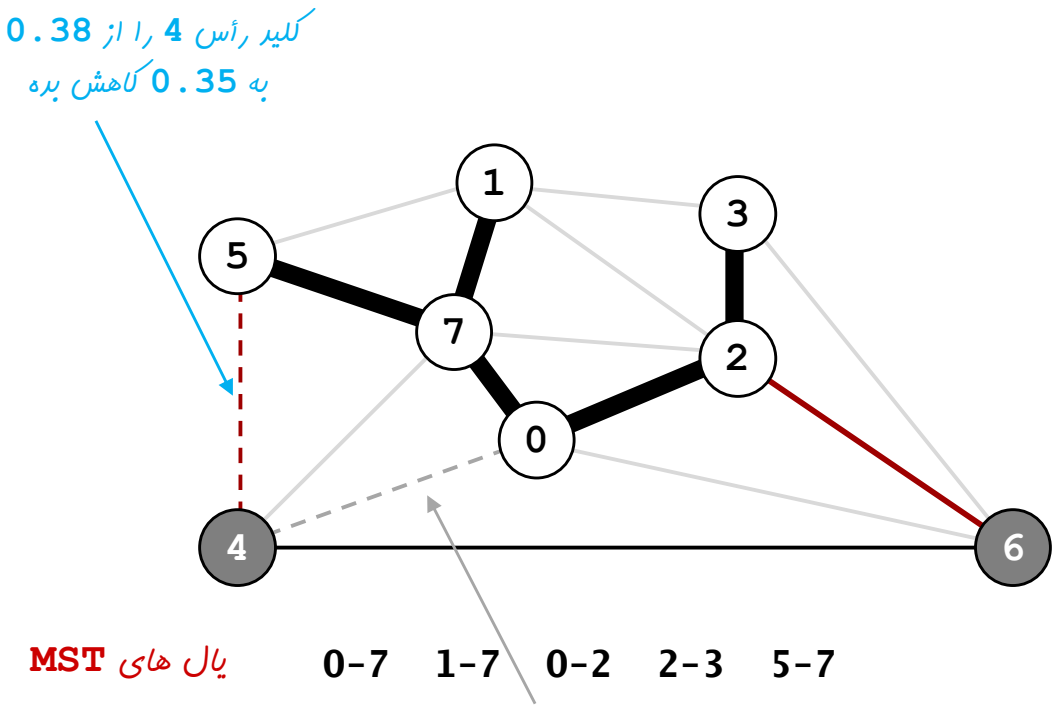


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
→ 5	5-7	0.28
4	0-4	0.38
6	6-2	0.40

یال های MST 0-7 1-7 0-2 2-3 5-7

الگوریتم پریم: اجرای نمایشی

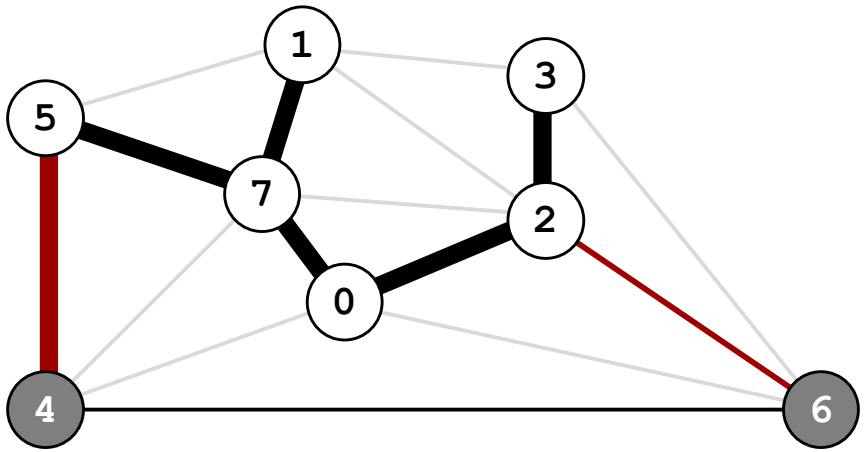
- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
→ 5	5-7	0.28
4	~~0-4~~ 4-5	~~0.38~~ 0.35
6	6-2	0.40

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

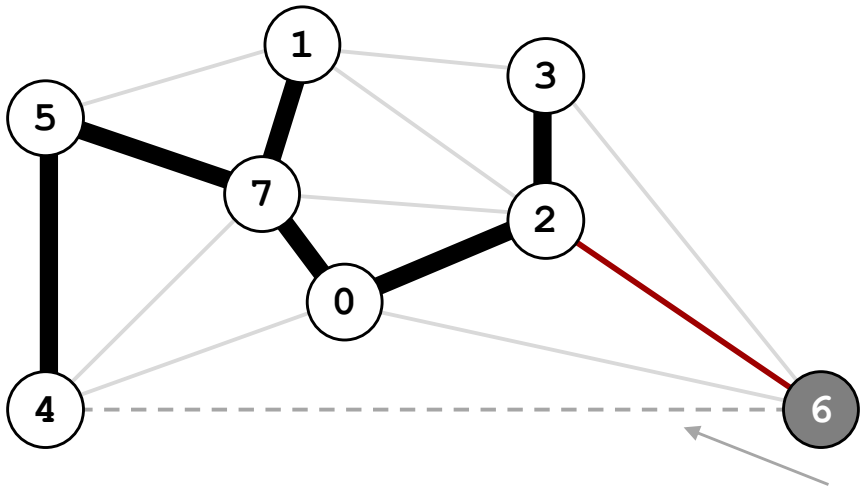


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
→ 4	4-5	0.35
6	6-2	0.40

یال های MST 0-7 1-7 0-2 2-3 5-7

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



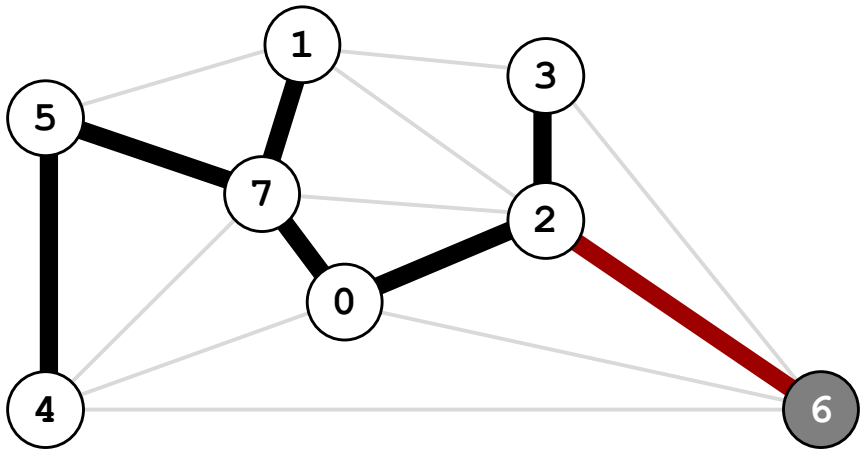
v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
→ 4	4-5	0.35
6	6-2	0.40

در حال حاضر یک یال کوتاه تر به رأس 4 وجود دارد

یال های **MST** 0-7 1-7 0-2 2-3 5-7 4-5

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

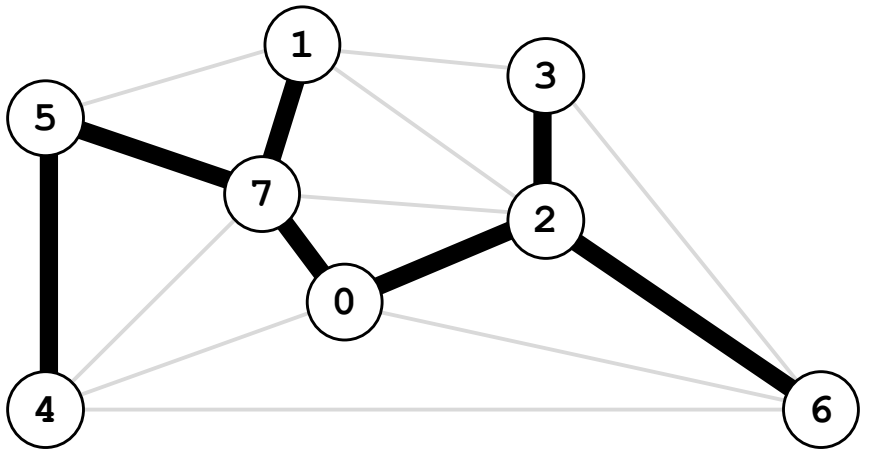


v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
4	4-5	0.35
→ 6	6-2	0.40

یال های **MST** 0-7 1-7 0-2 2-3 5-7 4-5

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال

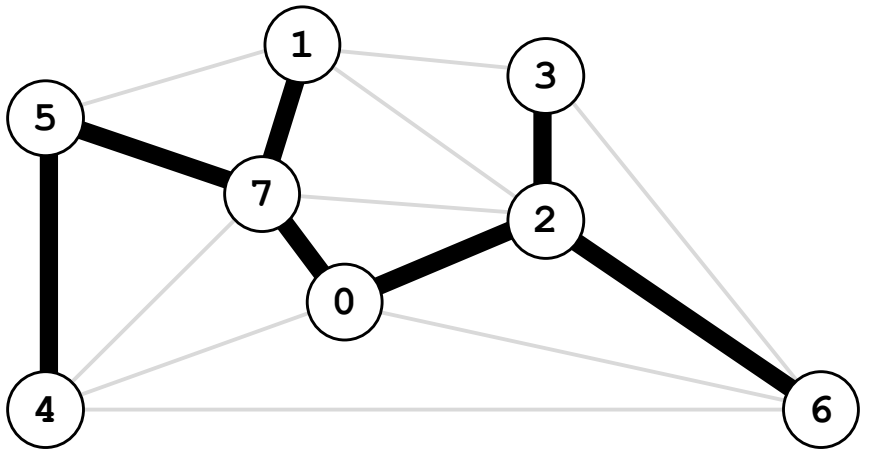


یال های MST 0-7 1-7 0-2 2-3 5-7 4-5 6-2

v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
4	4-5	0.35
→ 6	6-2	0.40

الگوریتم پریم: اجرای نمایشی

- الگوریتم پریم.
  - با رأس صفر شروع کن و درخت T را به طور حریصانه بزرگ کن
  - در هر مرحله، یال با کمترین وزن را که دقیقا یک رأس آن در T است، به T اضافه کن
  - تکرار مراحل فوق تا به دست آوردن V - 1 یال



v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
4	4-5	0.35
6	6-2	0.40

یال های **MST** 0-7 1-7 0-2 2-3 5-7 4-5 6-2

صف اولویت اندیس‌دار

- به هر کلید در صف اولویت یک اندیس بین 0 تا N - 1 نسبت بده
  - عملیات: حذف عنصر با کوچک‌ترین کلید و درج یک عنصر جدید
  - یک عمل جدید: کاهش کلید یک عنصر با مشخص کردن اندیس آن

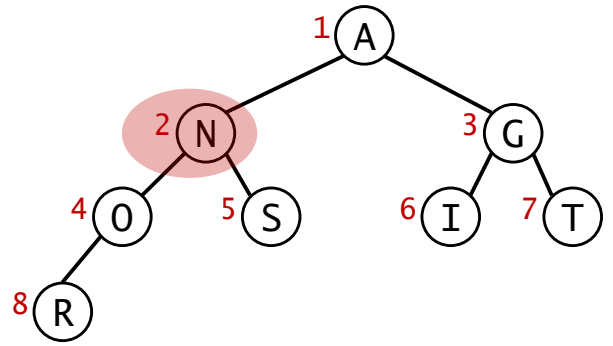
```
public class IndexMinPQ <Key extends Comparable<Key>>
```

	IndexMinPQ(int N)	ایجاد یک صف اولویت اندیس دار با اندیس های 0 تا N - 1
void	insert(int k, Key key)	نسبت دادن اندیس k به کلید key
void	decreaseKey(int k, Key key)	کاهش کلید مرتبط با اندیس k
boolean	contains(int k)	آیا اندیس k در صف اولویت وجود دارد؟
int	delMin()	حذف عنصر با کوچکترین کلید و برگرداندن اندیس مرتبط با آن
boolean	isEmpty()	آیا صف اولویت خالی است؟
int	size()	برگرداندن تعداد عناصر صف اولویت

پیاده‌سازی صف اولویت اندیس‌دار

- پیاده‌سازی.
  - با کدی مشابه با کد `MinPQ` شروع کن
  - برای پیاده‌سازی از سه آرایه `keys[]`، `pq[]` و `qp[]` استفاده کن به گونه‌ای که:
    - `Keys[i]` بیانگر اولویت `i` است
    - `pq[i]` بیانگر اندیس کلیدی است که در مکان `i` از هرم قرار دارد
    - `qp[i]` بیانگر مکان کلید با اندیس `i` در هرم است
  - به منظور پیاده‌سازی عمل `decreaseKey(k, key)` از دستور `swim(qp[k])` استفاده کن

i	0	1	2	3	4	5	6	7	8
keys[i]	A	S	O	R	T	I	N	G	-
pq[i]	-	0	6	7	2	1	5	4	3
qp[i]	1	5	4	8	7	6	2	3	-



الگوریتم پریم: پیاده‌سازی


```
public class PrimMST
{
    private boolean[] marked;          // true if v on tree
    private Edge[] edgeTo;             // shortest edge from tree vertex
    private double[] distTo;           // distTo[w] = edgeTo[w].weight()
    private IndexMinPQ<Double> pq;     // eligible crossing edges

    public PrimMST(EdgeWeightedGraph G)
    {
        marked = new boolean[G.V()];
        edgeTo = new Edge[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[0] = 0;

        pq.insert(0, 0.0);
        while (!pq.isEmpty())
            visit(G, pq.delMin());
    }
```

الگوریتم پریم: پیاده‌سازی


```
    private void visit(EdgeWeightedGraph G, int v)
    {
        marked[v] = true;
        for (Edge e : G.adj(v))
        {
            int w = e.other(v);
            if (marked[w]) continue;
            if (e.weight() < distTo[w])
            {
                edgeTo[w] = e;
                distTo[w] = e.weight();
                if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
                else                pq.insert     (w, distTo[w]);
            }
        }
    }


    public Iterable<Edge> edges() {  /* exercise */  }
    public double weight()        {  /* exercise */  }
}
```

الگوریتم پریم: زمان اجرا

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

پیاده سازی	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

† هزینه سرشکن شده

- جمع‌بندی.
  - پیاده‌سازی با آرایه برای گراف‌های شلوغ بهینه است.
  - هرم دودویی برای گراف‌های خلوت بسیار سریع تر است.
  - هرم فیبوناچی در تئوری بهترین است، اما ارزش پیاده‌سازی ندارد.

کوتاه‌ترین مسیر تک مبدأ

واسط یال جهت‌دار وزن‌دار

- واسط کلاس یال جهت‌دار وزن دار.

public class	DirectedEdge	
	DirectedEdge(int v, int w, double weight)	ایجاد یال جهت دار 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: پیاده سازی‌جاوا


```
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 یال های خروجی از رأس **v**
Iterable<DirectedEdge>	edges()	برگرداندن همه یال های گراف
int	V()	برگرداندن تعداد رئوس
int	E()	برگرداندن تعداد یال ها
String	toString()	نمایش گراف وزن دار به صورت یک رشته

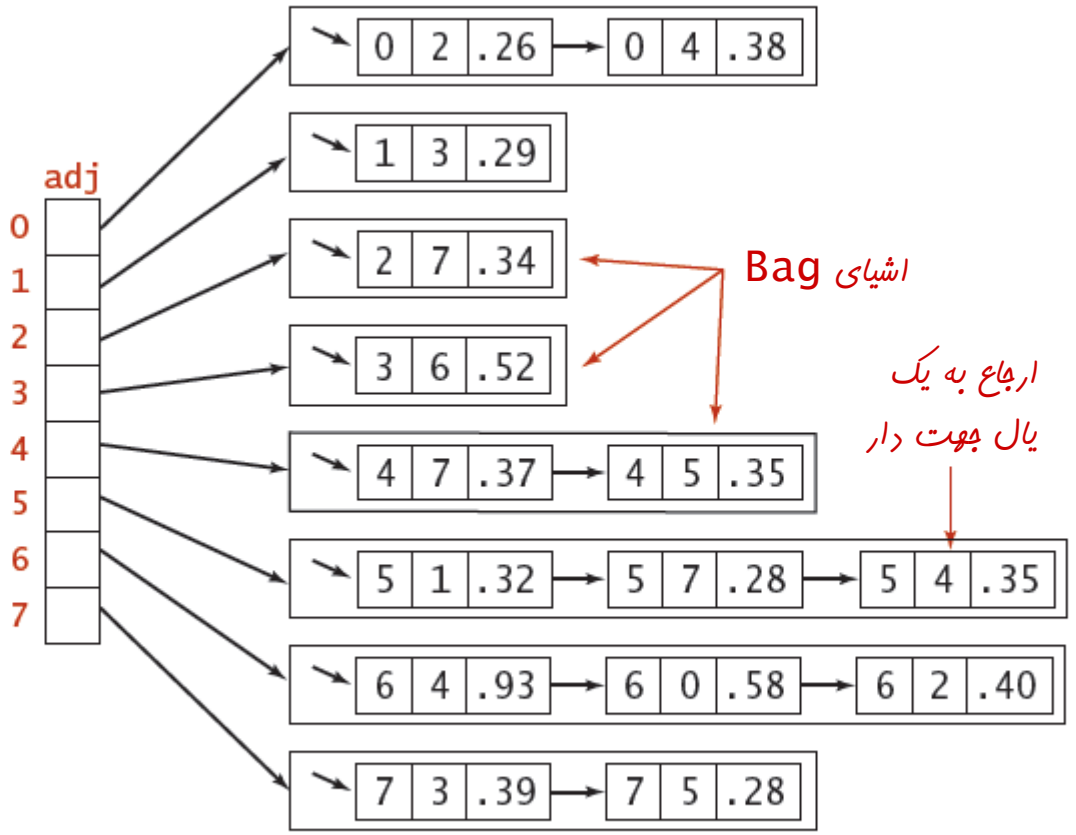
قراردادها. در این پیاده‌سازی داشتن طوقه و یال‌های موازی مجاز است.

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 کوتاه ترین مسیرها از **s**
double	distTo(int v)	طول کوتاه ترین مسیر از **s** به **v**
Iterable<DirectedEdge>	pathTo(int v)	کوتاه ترین 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.

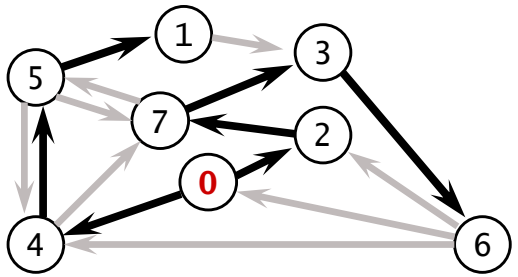
public class	SP	
	SP(EdgeWeightedDigraph G, int s)	محاسبه کوتاه ترین مسیرها از **s**
double	distTo(int v)	طول کوتاه ترین مسیر از **s** به **v**
Iterable<DirectedEdge>	pathTo(int v)	کوتاه ترین مسیر از **s** به **v**
boolean	hasPathTo(int v)	آیا از **s** به **v** مسیر وجود دارد؟

```
% 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 داده‌ها 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.



درخت کوتاه‌ترین مسیر از s

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

ساختمان داده‌ها برای مسئله کوتاه‌ترین مسیر تک مبدأ

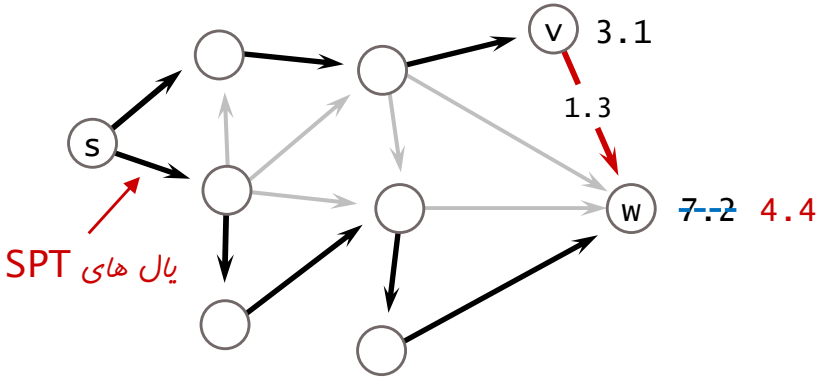
- هدف. یافتن کوتاه‌ترین مسیر از رأس s به تمام رئوس دیگر.
- مشاهده. یک درخت کوتاه‌ترین مسیر (SPT) وجود دارد. چرا؟
- نتیجه. درخت کوتاه‌ترین مسیر را با استفاده از دو آرایه می‌توان نمایش داد:
  - distTo[v] = طول کوتاه‌ترین مسیر از s به v.
  - edgeTo[v] = آخرین یال در کوتاه‌ترین مسیر از s به v.

```
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 می کند،
    هر دو مقدار distTo[w] و edgeTo[w] را به روز رسانی کن.

راحت‌سازی موفقیت آمیز یال v->w

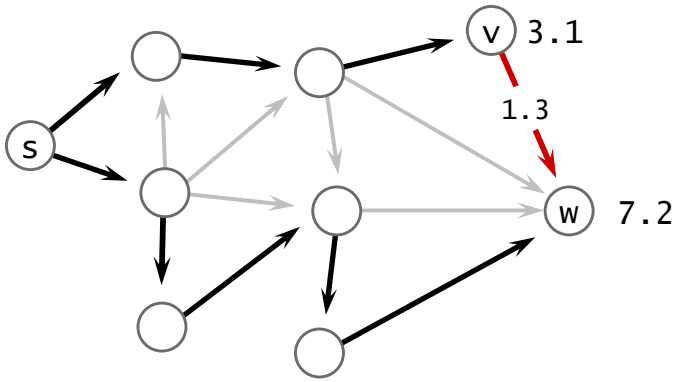


راحت‌سازی یال (relaxation)

- راحت‌سازی یال v->w = e.
  - 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 می کند،

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

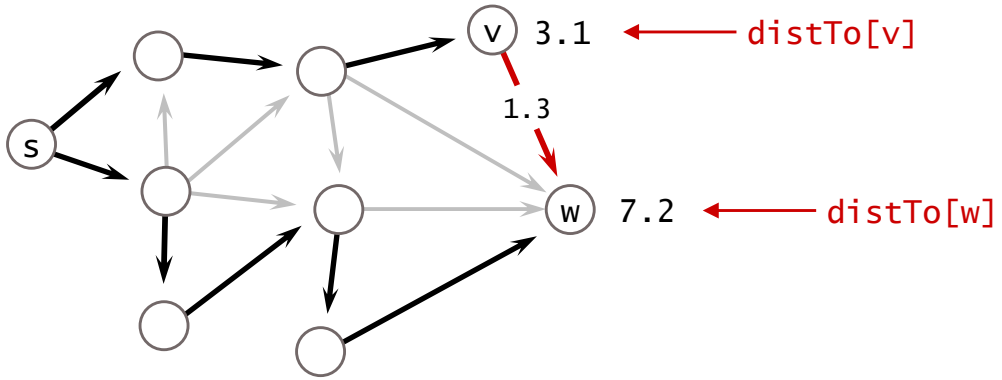
```
private void relax(DirectedEdge e)
{
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight())
    {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
    }
}
```



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 کوتاه ترین 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

- پیاده‌سازی کارا. کدام یال باید راحت‌سازی 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

ادسخر دایکسترا
جایزه تورینگ ۱۹۷۲

« 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.»

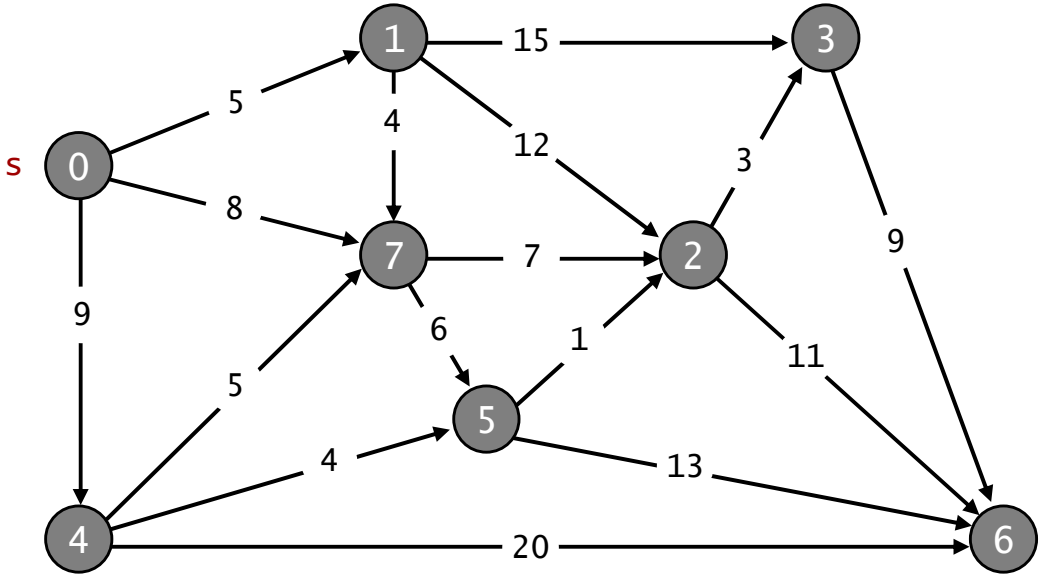
« استفاده از کوبول مغز را تباه می‌کند؛ بنابراین تدریس آن باید به عنوان نوعی قانون‌شکنی تلقی شود. »

« 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. »

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

الگوریتم دایکسترا: اجرای نمایشی

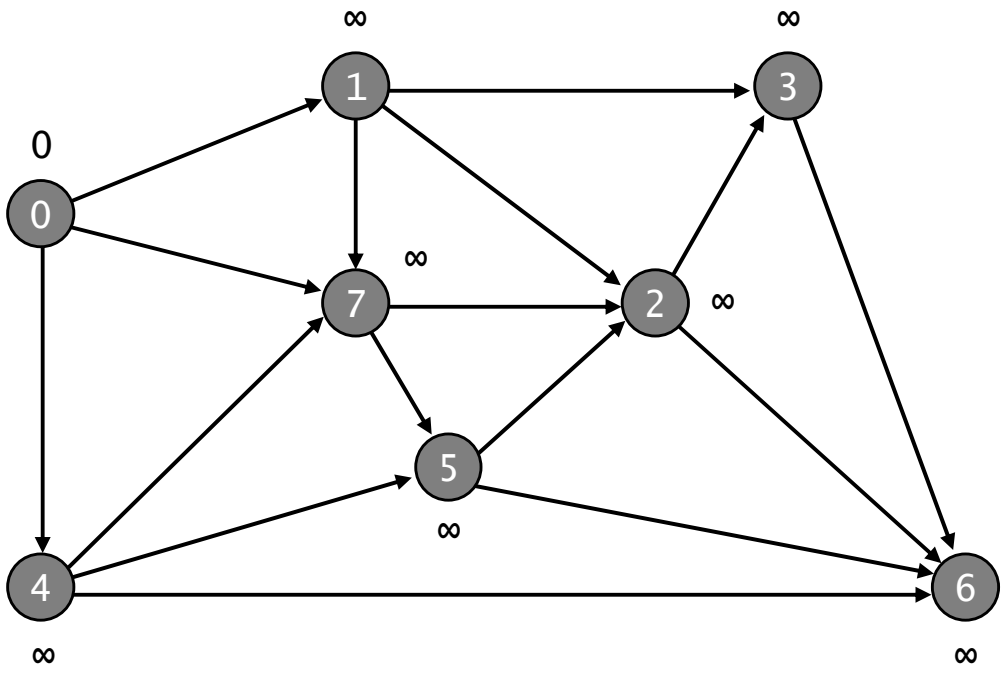
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.




```
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

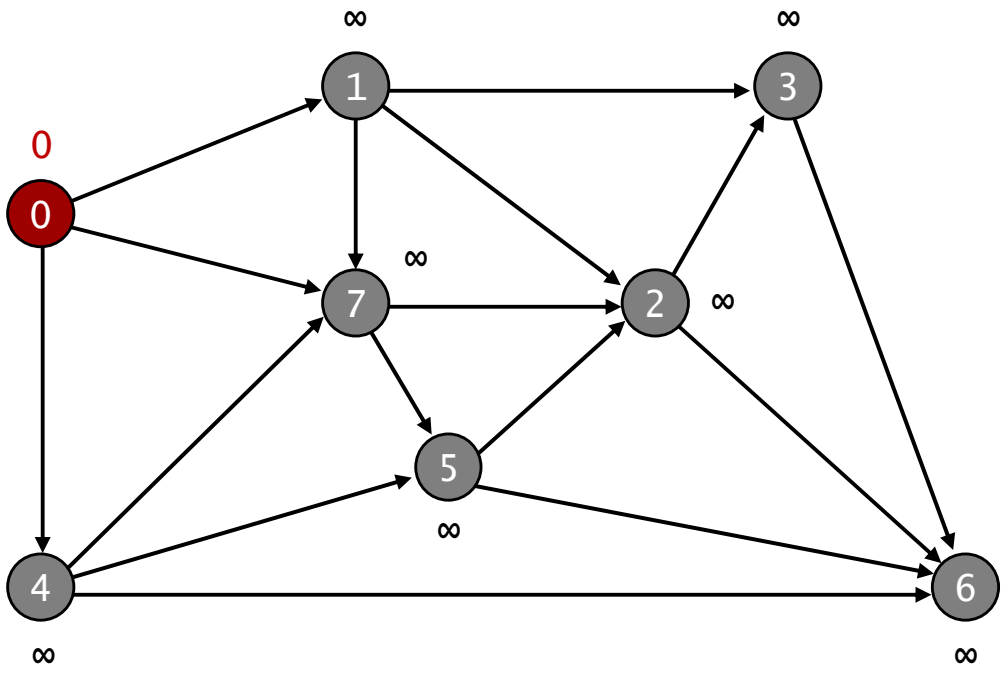
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

الگوریتم دایکسترا: اجرای نمایشی

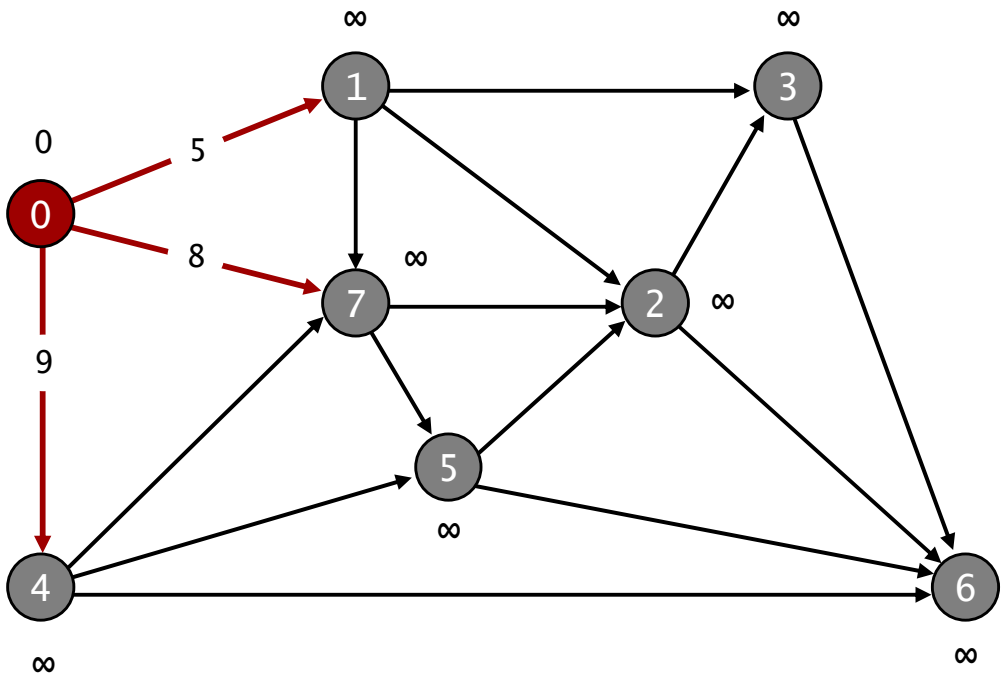
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

الگوریتم دایکسترا: اجرای نمایشی

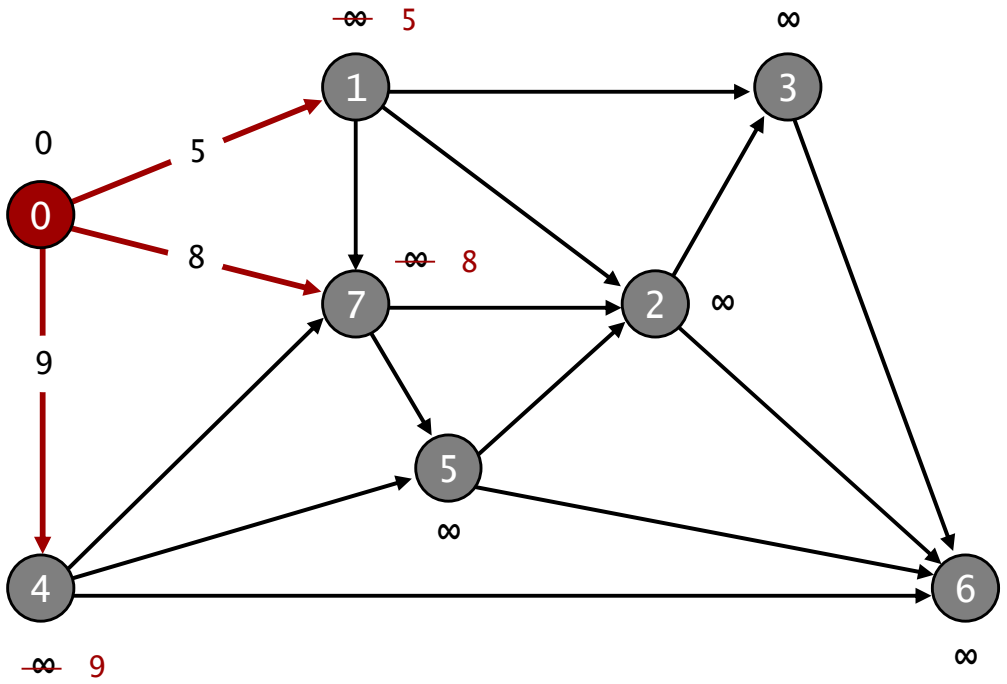
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

الگوریتم دایکسترا: اجرای نمایشی

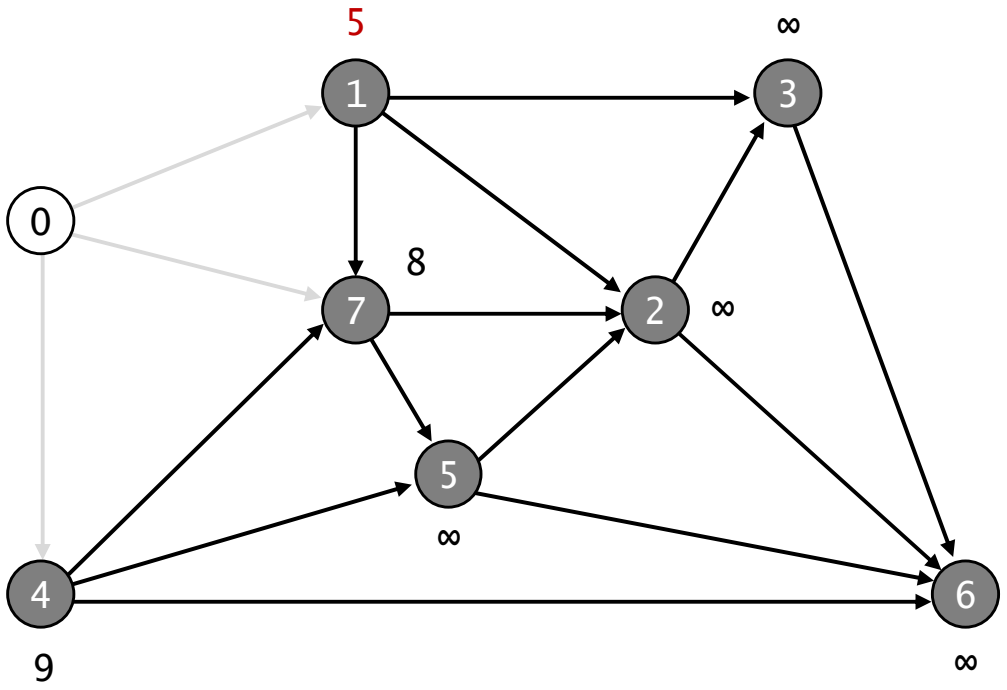
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

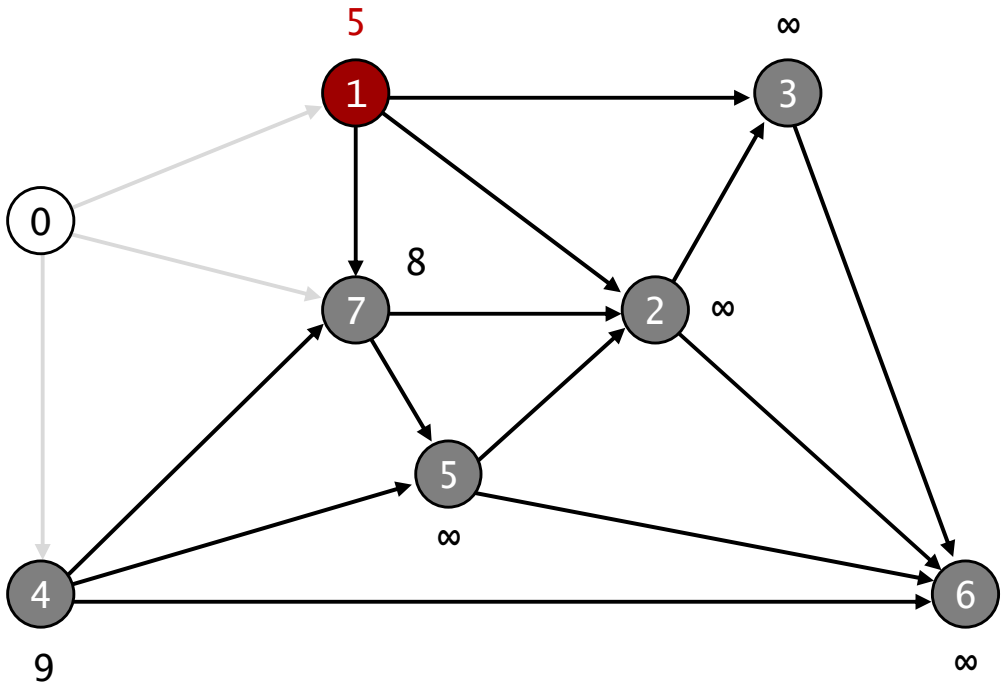
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

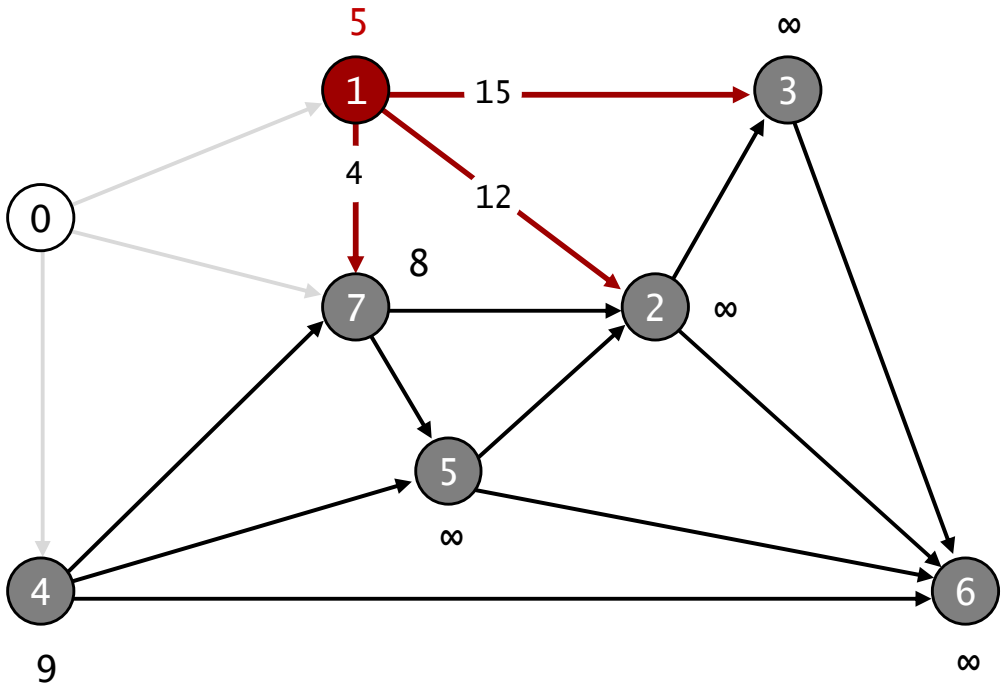
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

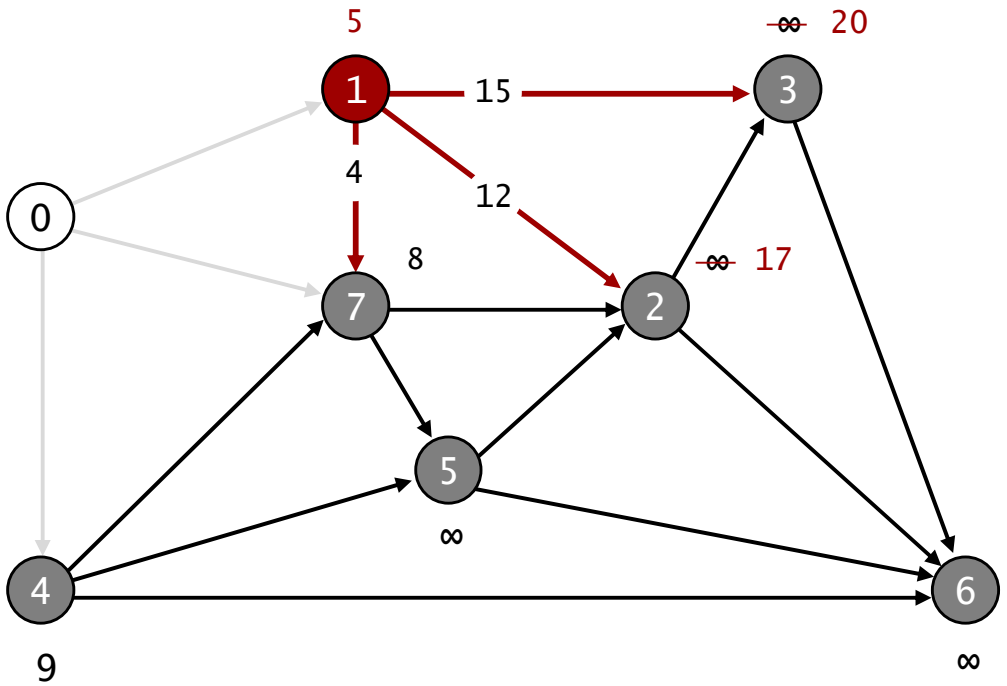
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

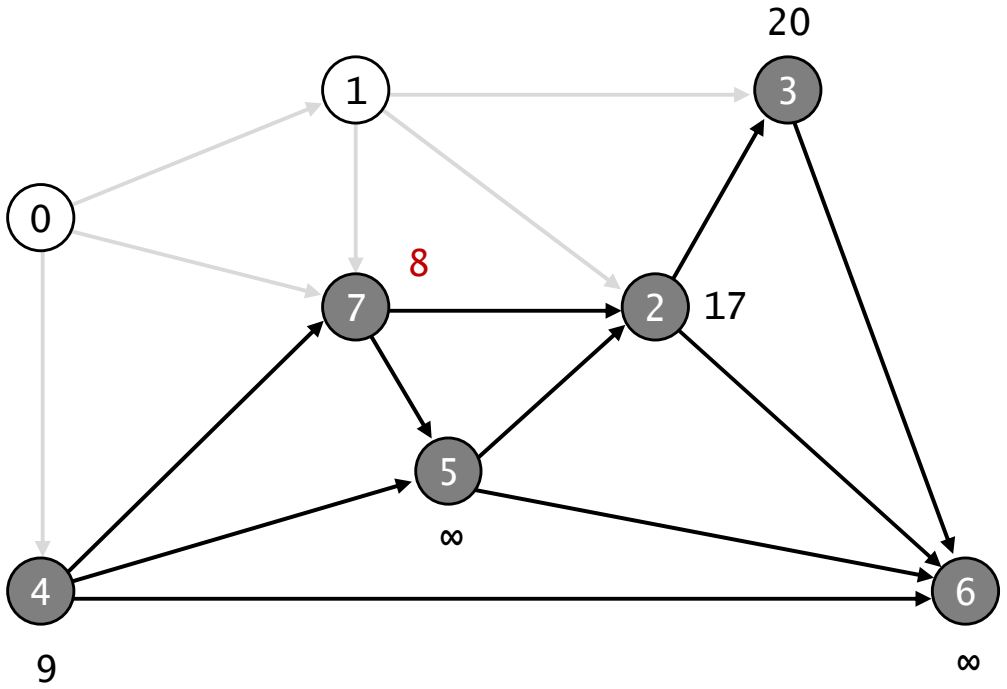
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

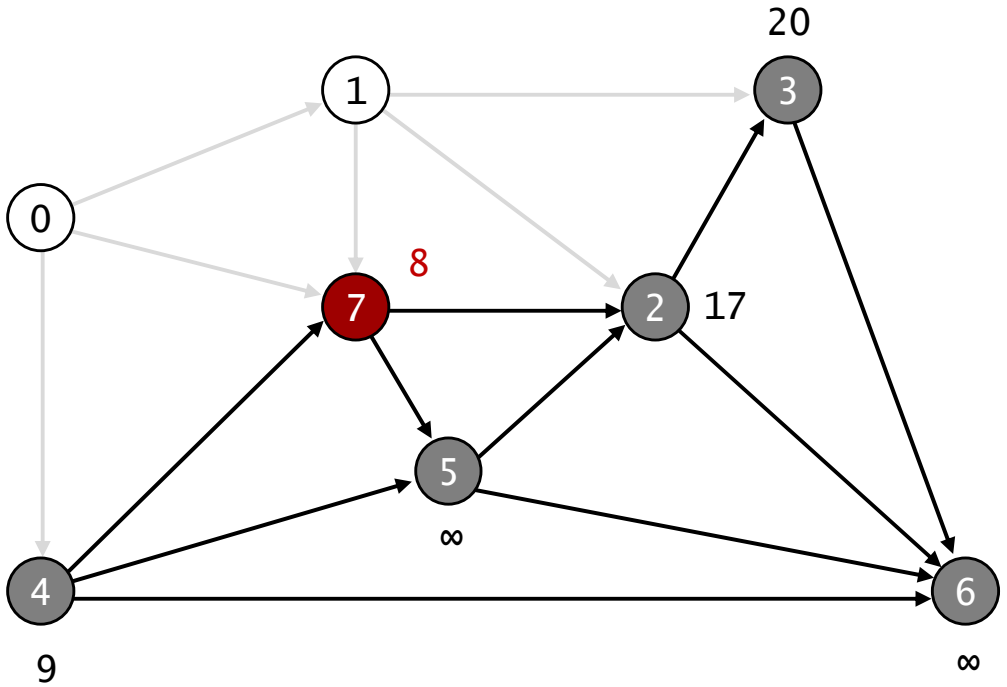
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

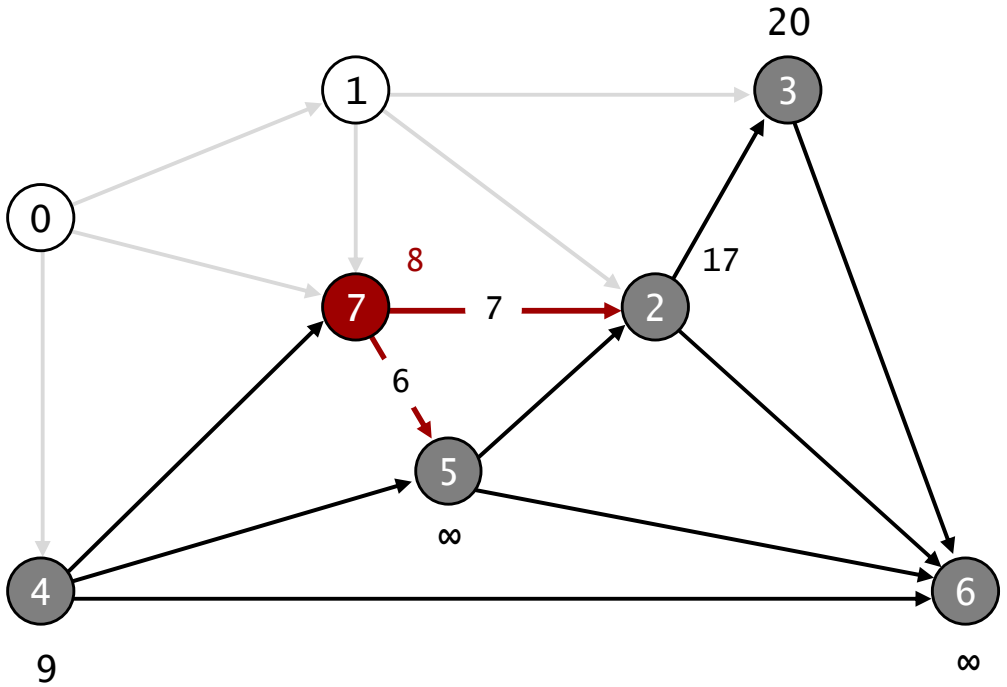
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

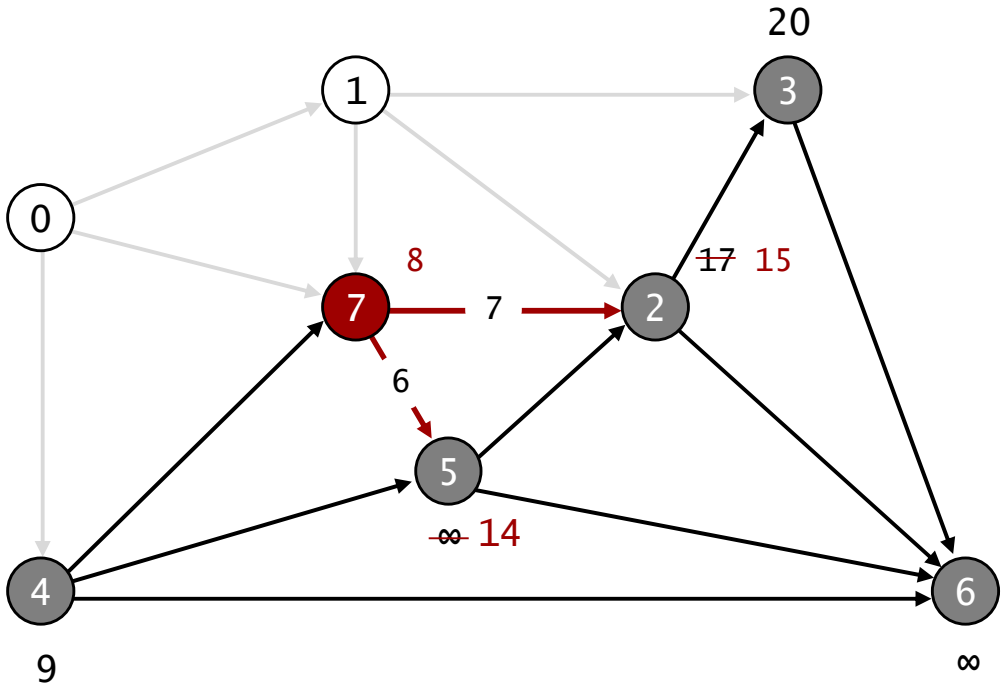
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

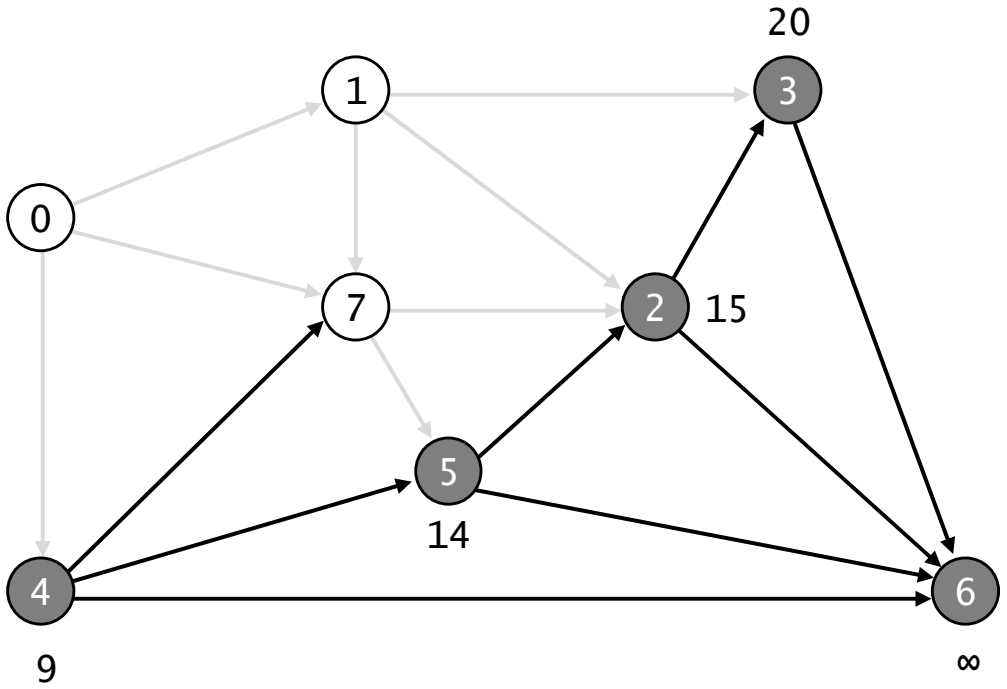
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

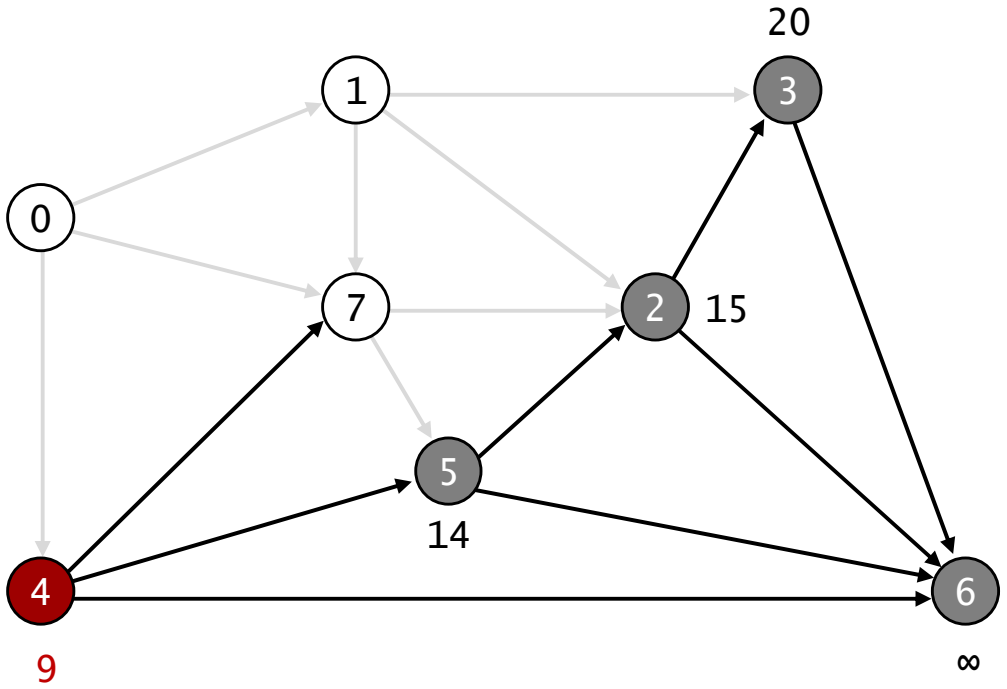
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

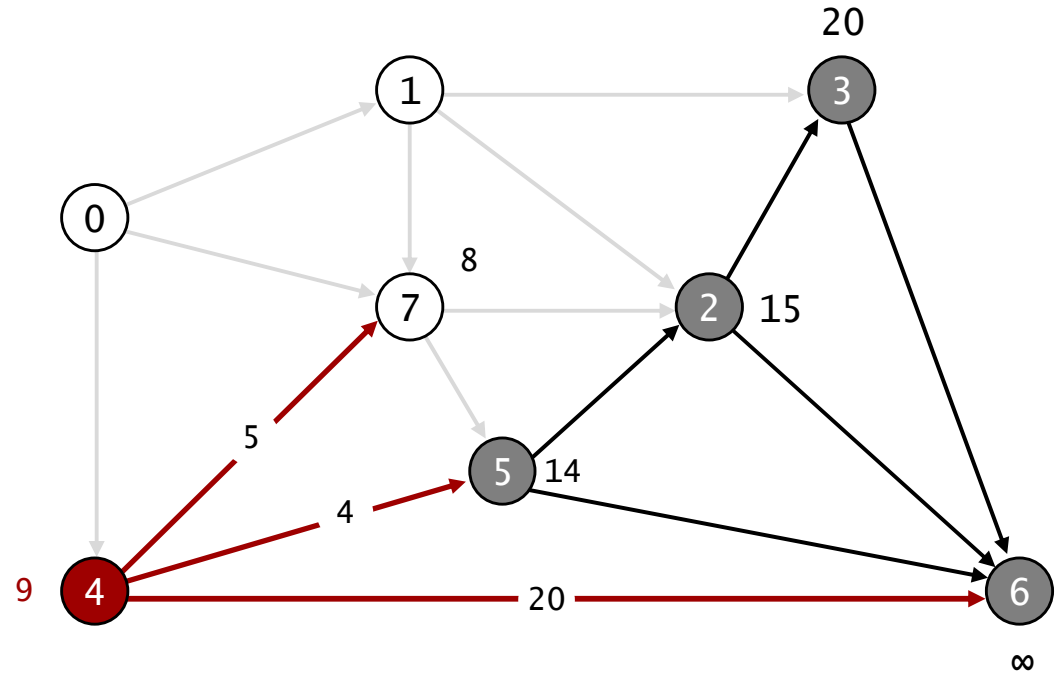
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

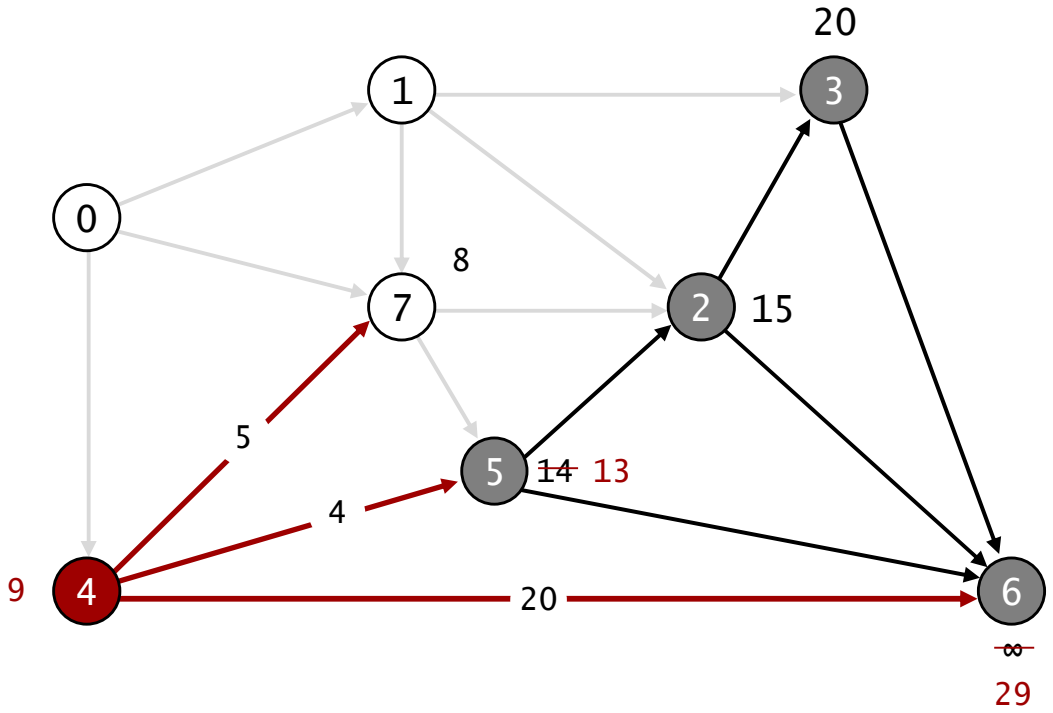
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

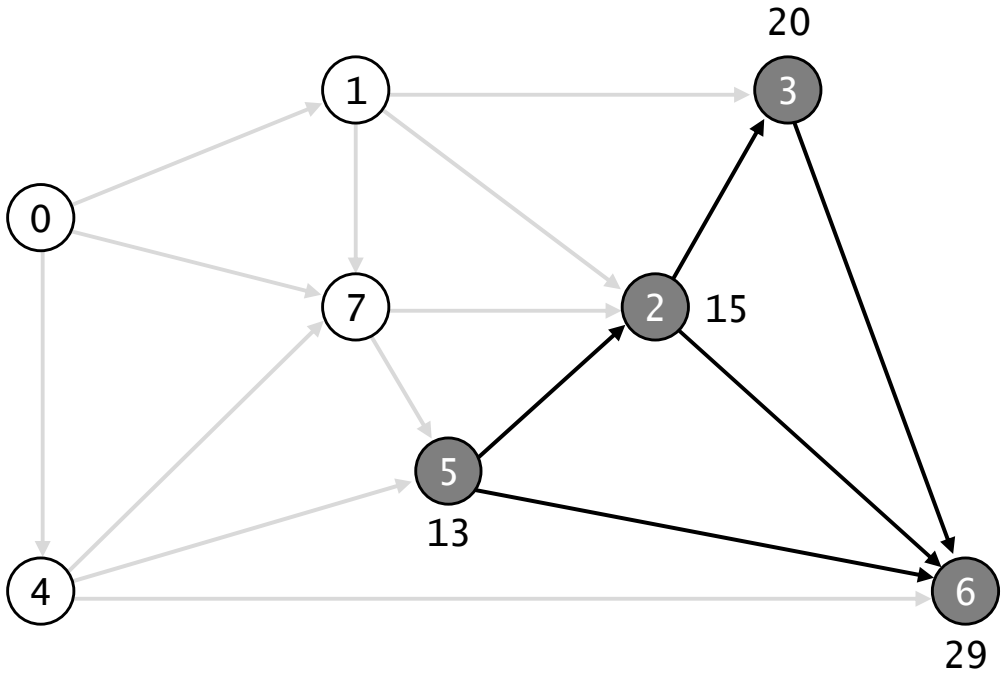
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

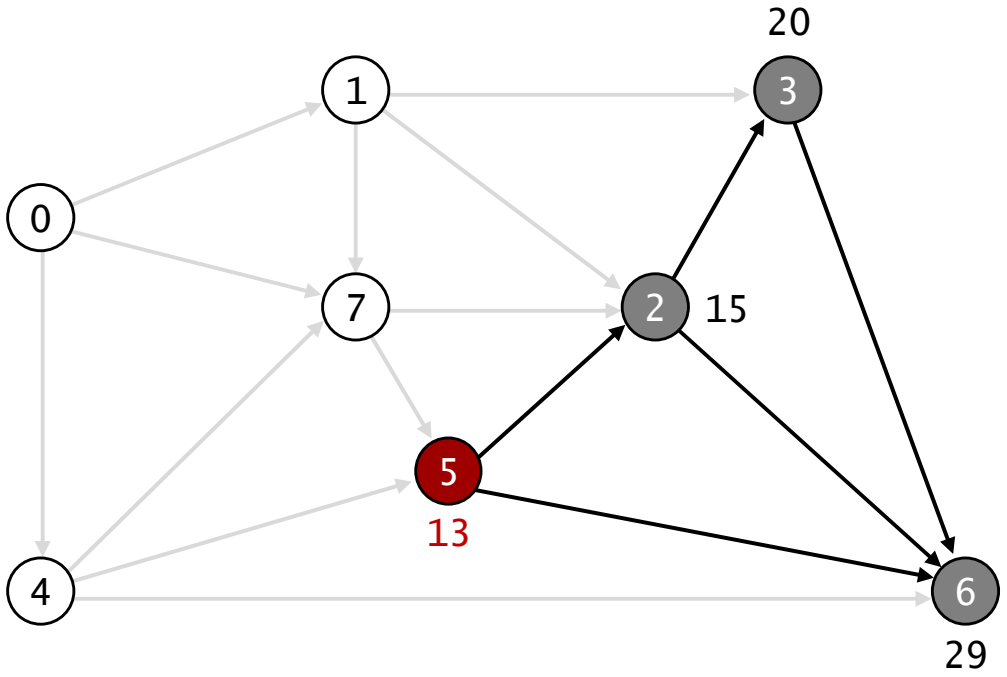
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

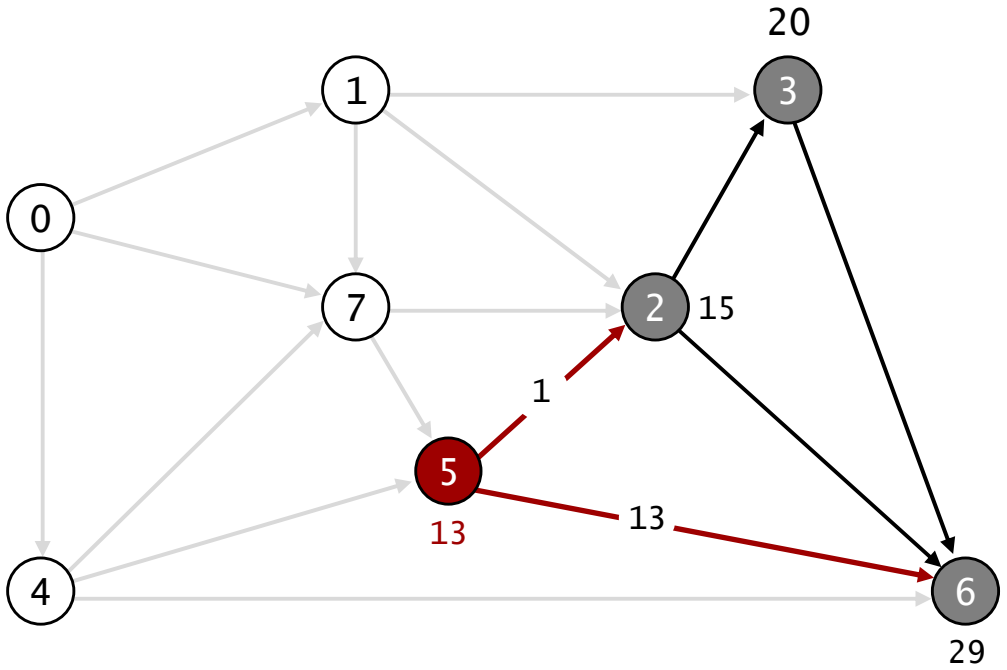
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

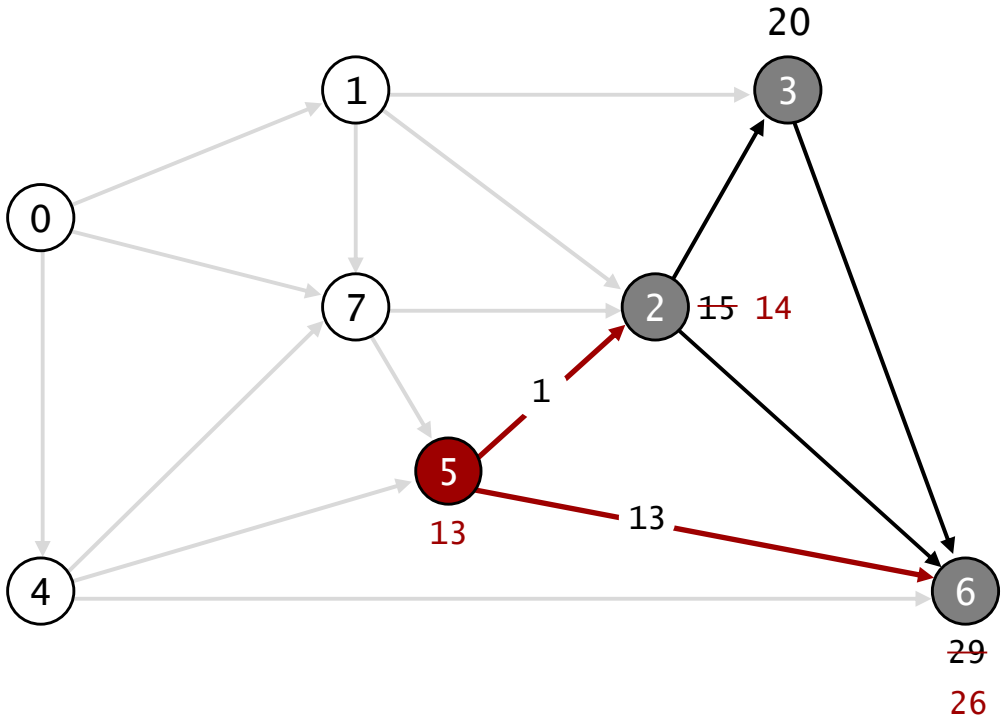
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

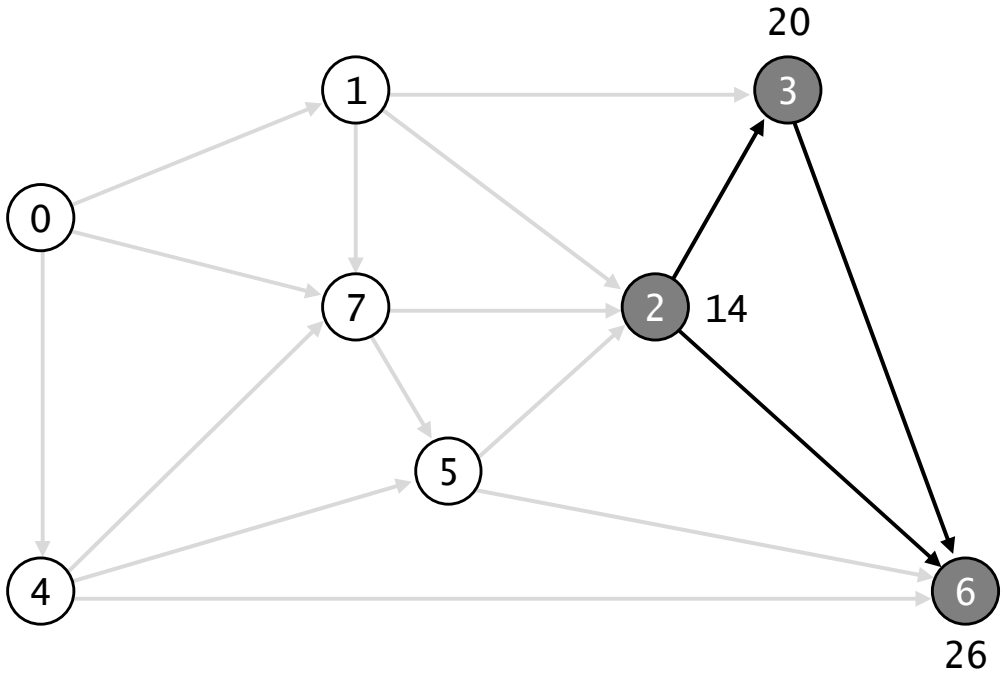
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

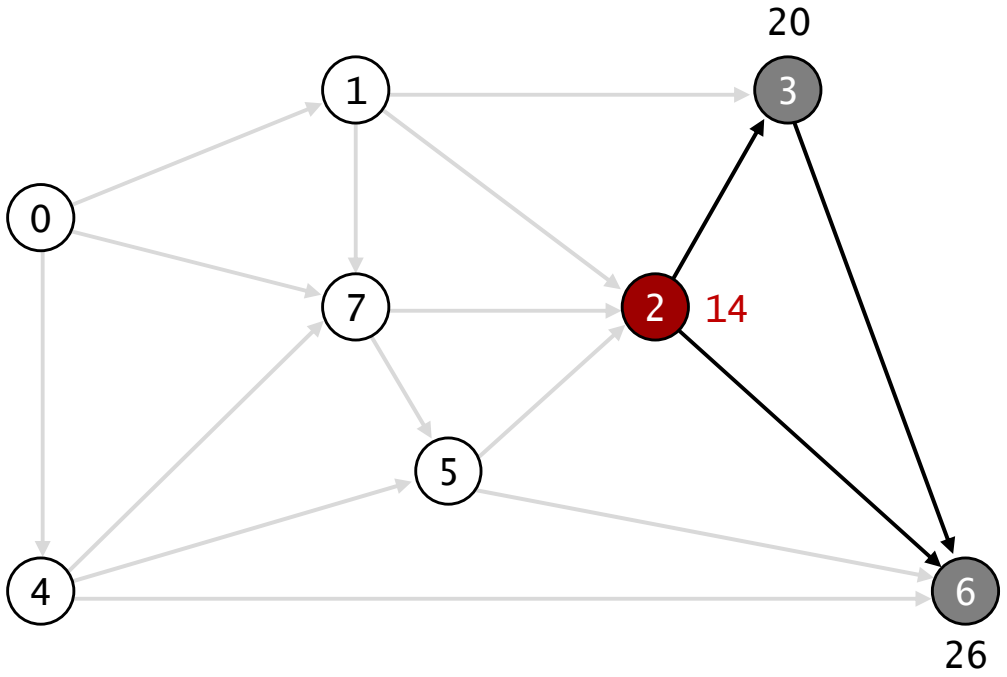
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

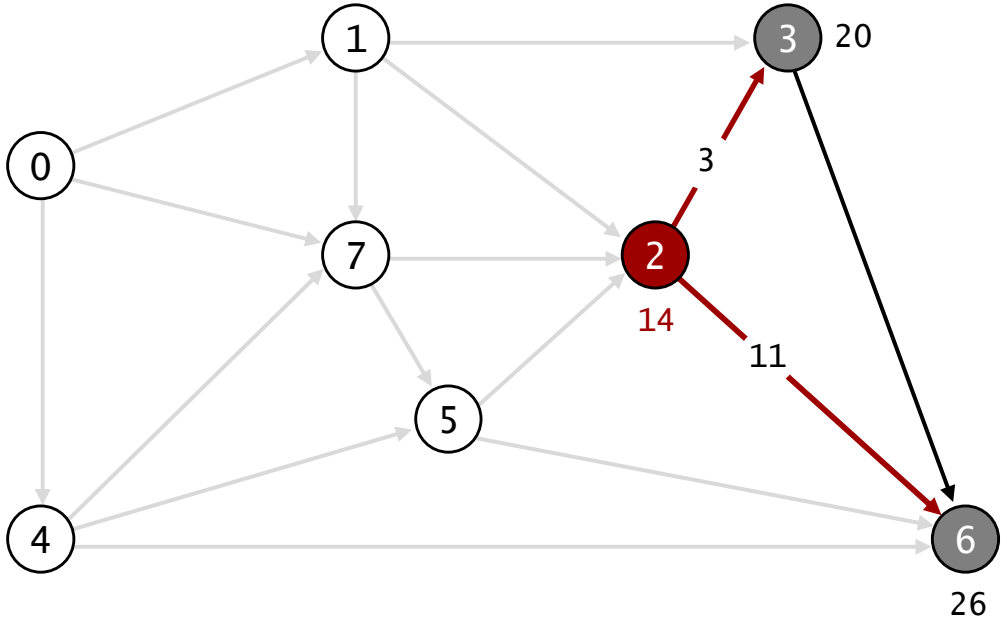
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

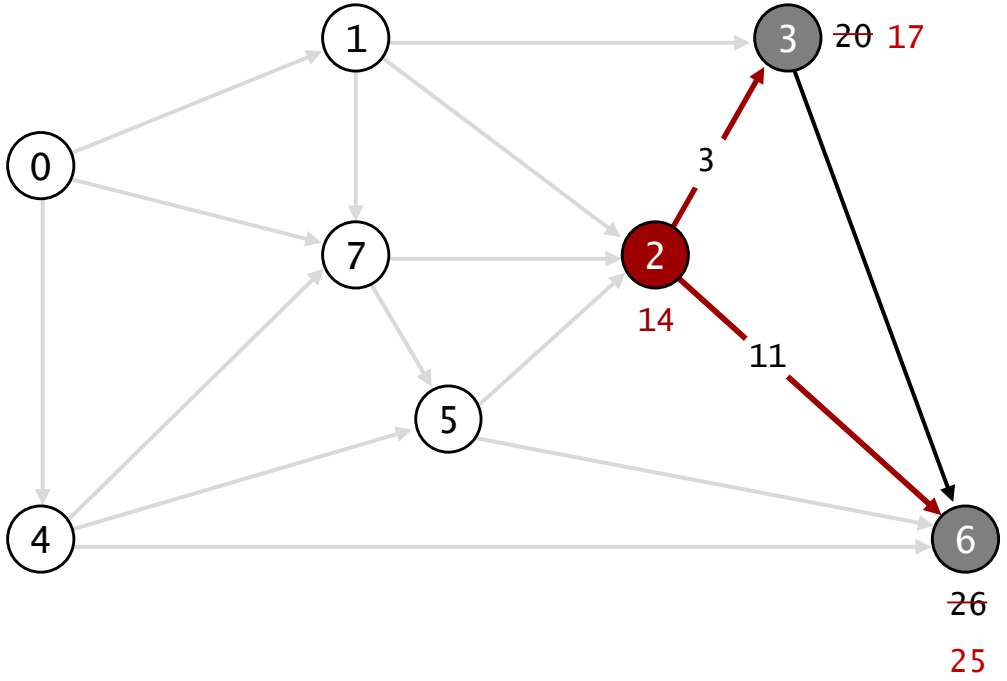
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

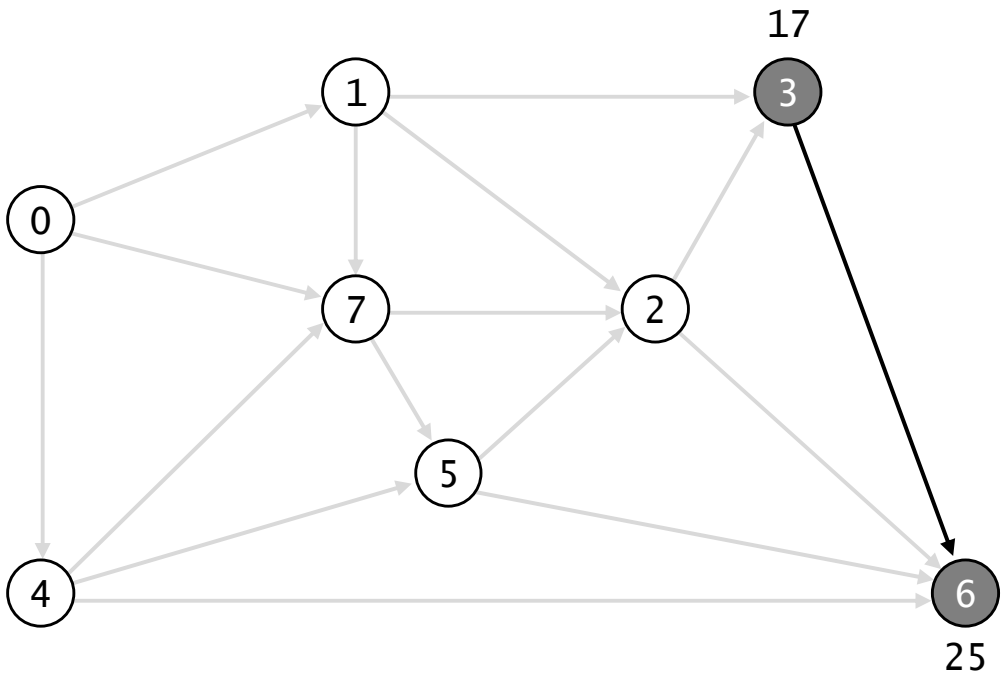
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

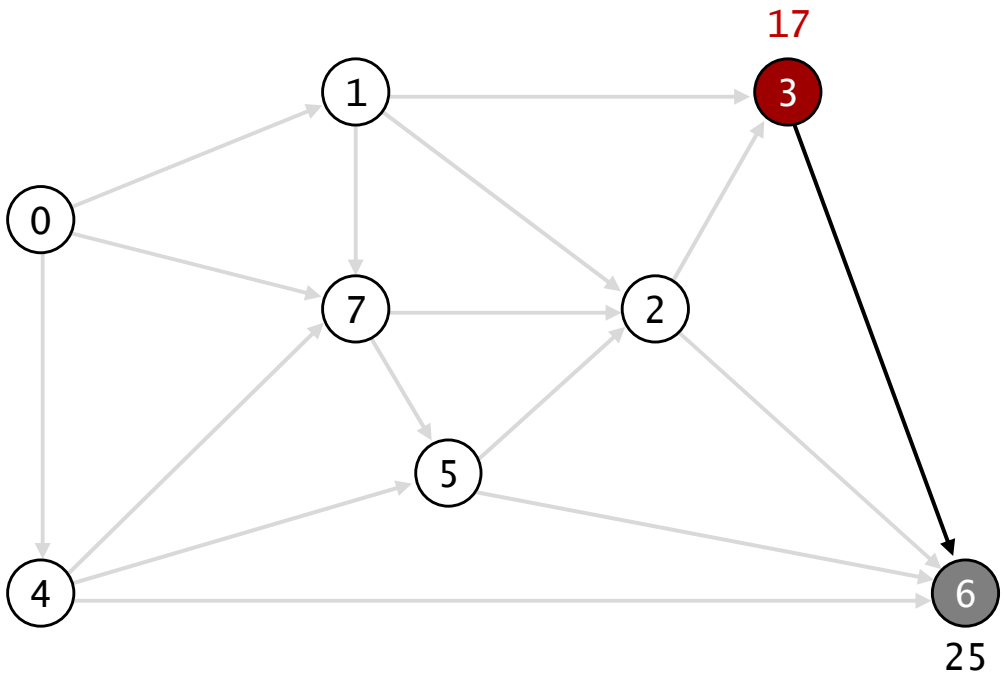
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

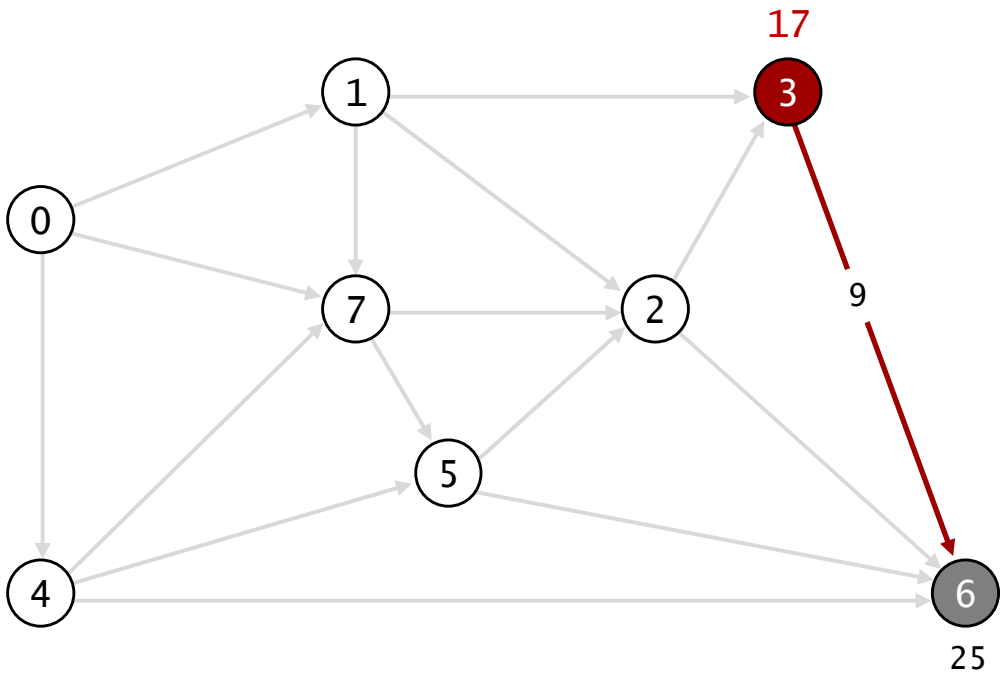
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

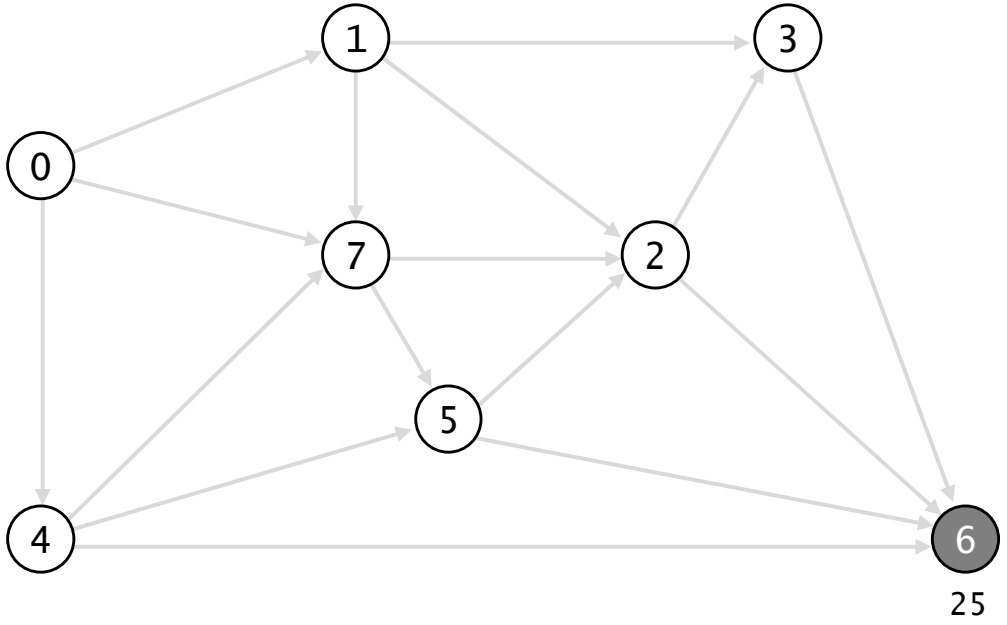
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

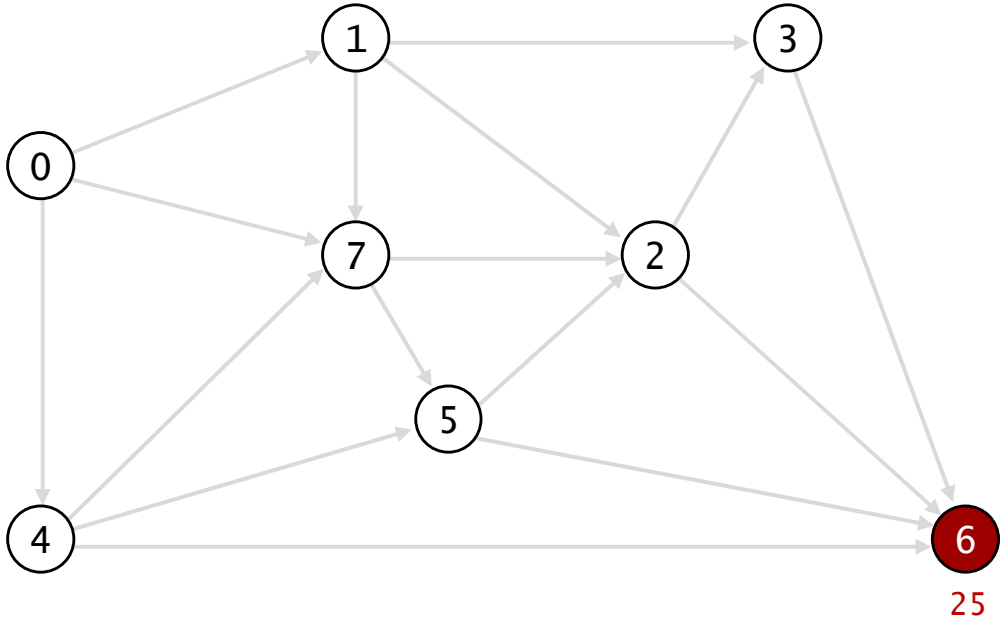
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

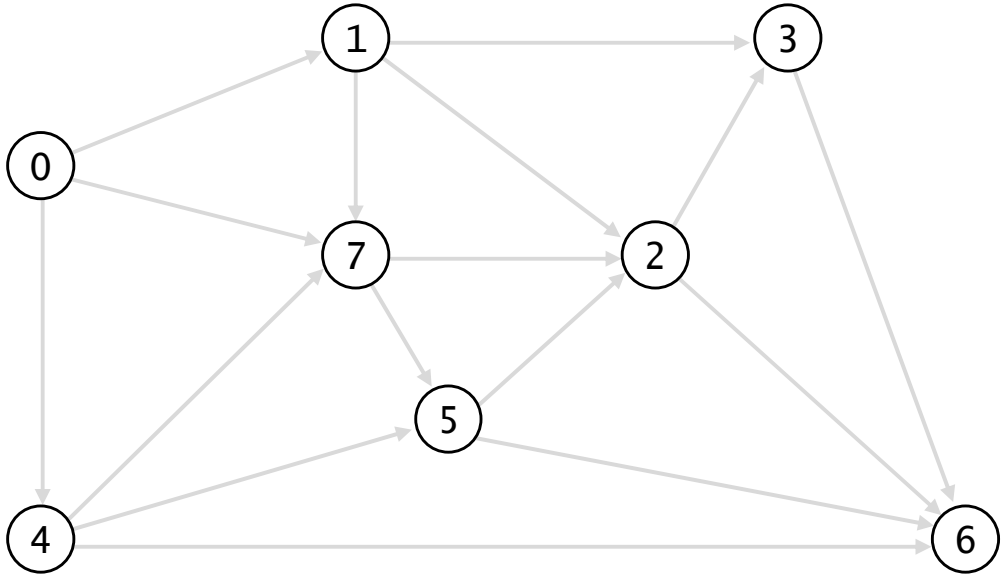
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

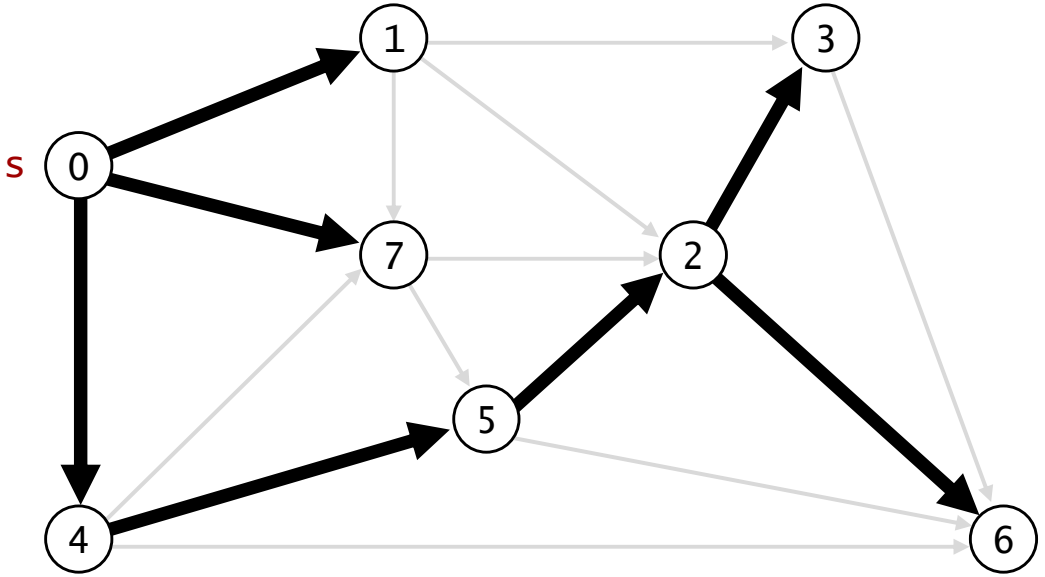
- رئوس را به ترتیب صعودی فاصله آنها تا رأس **s** در نظر بگیر.
- نزدیک‌ترین رأس را به درخت اضافه کن و تمام یال‌های خروجی از آن را راحت‌سازی کن.



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

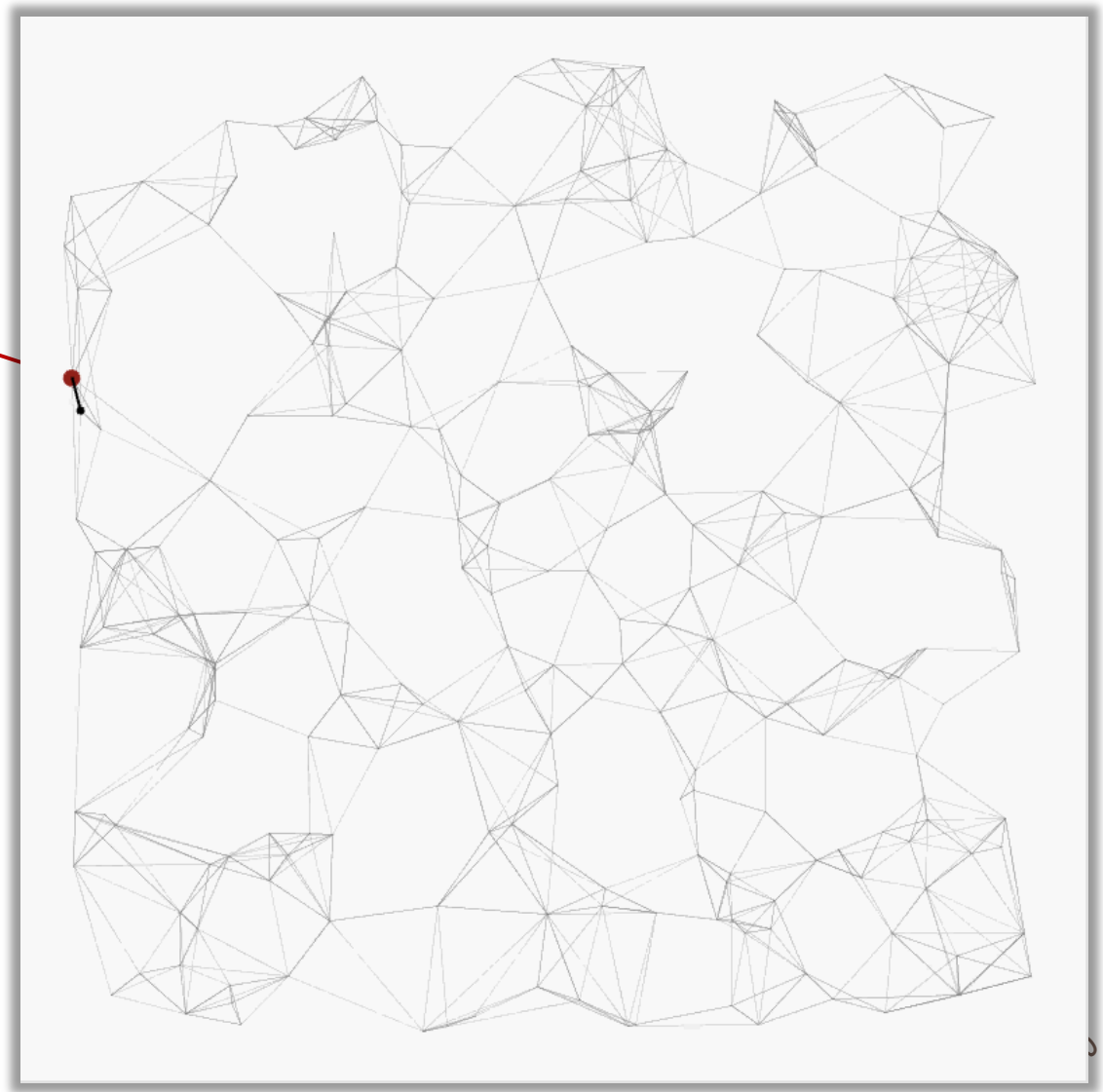
الگوریتم دایکسترا: اجرای نمایشی

- رئوس را به ترتیب صعودی فاصله 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 دایکسترا در هر گراف جهت‌دار وزن‌دار بدون وزن‌های 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:

$$\texttt{distTo[w]} \le \texttt{distTo[v]} + \texttt{e.weight()}$$

  - نامساوی فوق 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.

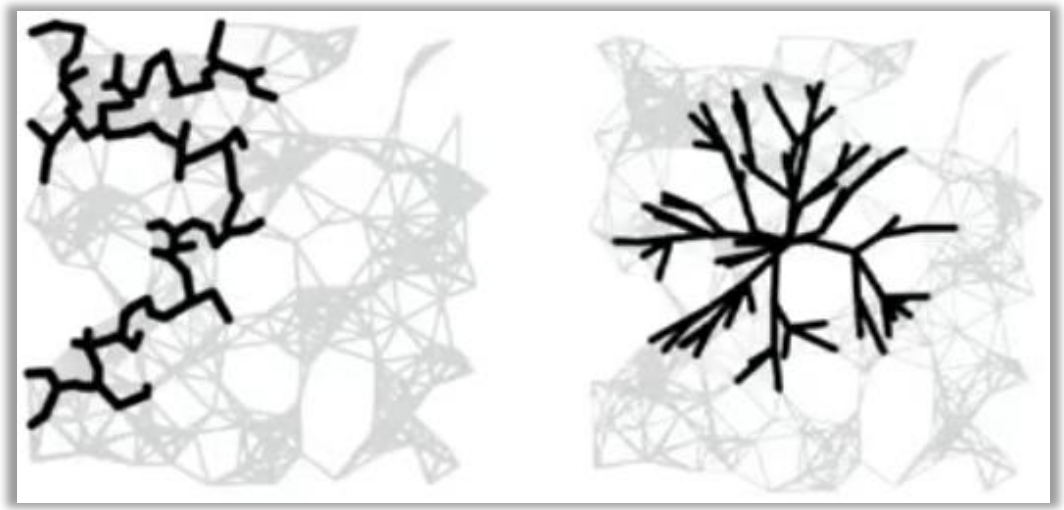
الگوریتم دایکسترا: پیاده‌سازی


```
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 محاسبه 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 شیوه پیاده‌سازی صف اولویت دارد:
  - V مرتبه درج، V مرتبه حذف کوچک ترین، 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

† هزینه سرشکن شده

- جمع‌بندی.
  - پیاده‌سازی با آرایه برای گراف‌های شلوغ بهینه است.
  - هرم دودویی برای گراف‌های خلوت بسیار سریع‌تر است.
  - هرم فیبوناچی در تئوری بهترین است، اما ارزش پیاده‌سازی ندارد.