Dijkstra es mi pastor

Nada me falta

Víctor Terrón
@pyctor


Python Kung Fu

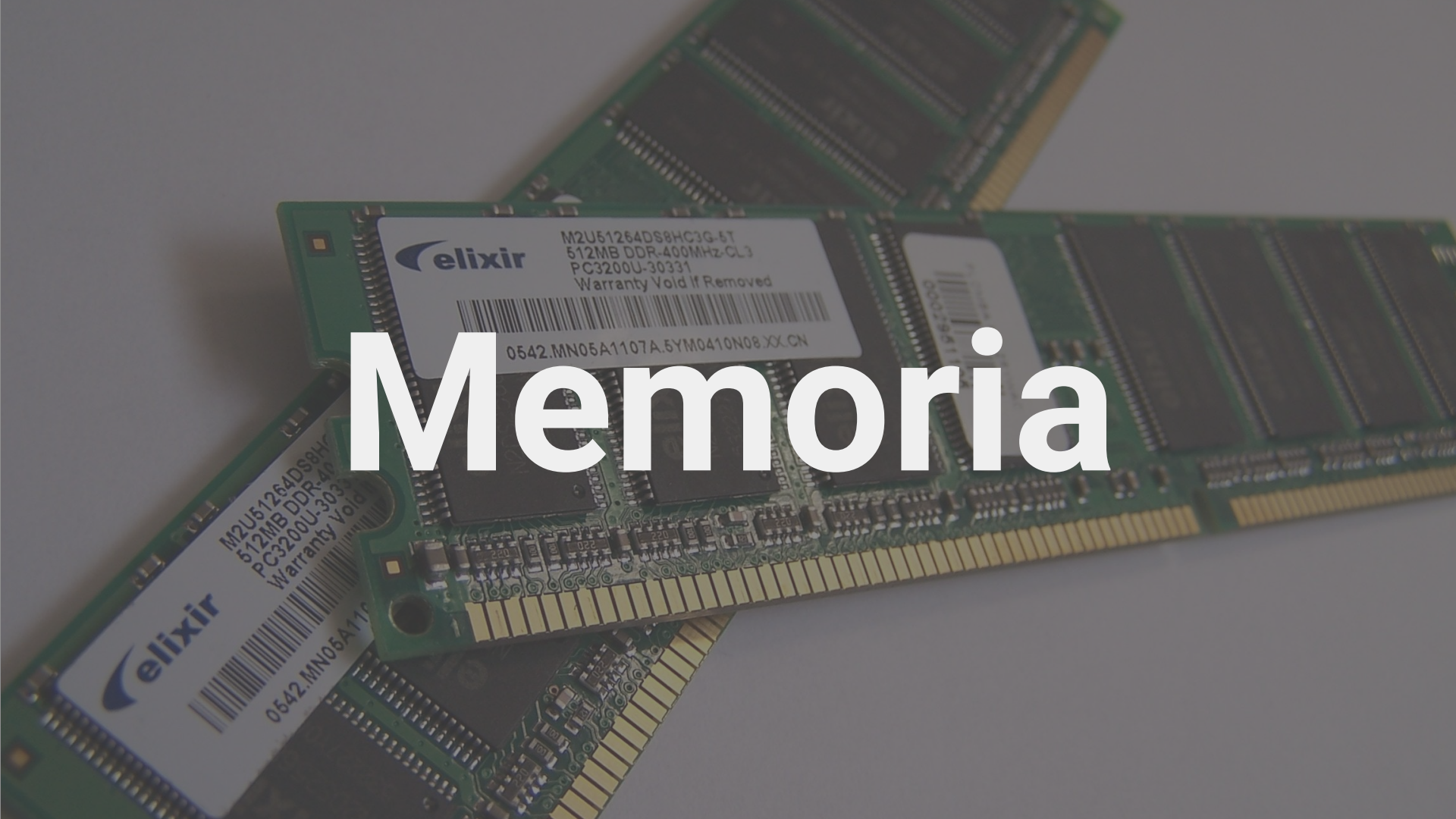
“¿Cómo determinas si una cadena de texto es una **permutación** de otra?”


He visto cosas que no
creeríais

```
def is_permutation(word, another):
    for letter in word:
        if word.count(letter) != another.count(letter):
            return False
    return True



>>> print(is_permutation("amor", "roma"))
True
```


The Hitchhiker's Guide to
Big O

Tiempo


elixir
M2U51264DS8HC3G-5T
512MB DDR-400MHz-CL3
PC3200U-30331
Warranty Void If Removed
0542.MN05A1107A.5YM0410N08.XX.CN
elixir
M2U51264DS8HC
512MB DDR-40
PC3200U-303
Warranty Void
0542.MN05A11

Memoria

Número áureo

¿A qué es proporcional?


IBM
IBM
IBM

```
>>> letras = u'abcdefghijklmnñopqrstuvwxyz'
>>> letras[5]
u'f'


>>> len(letras)
27
```

```
>>> letras = u'abcdefghijklmnñopqrstuvwxyz'
>>> letras[5]
u'f'
```

O(1)

Constante

```
>>> len(letras)
27
```

```
>>> "we dance whene'er we're able".count('e')
8


>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
>>> "we dance whene'er we're able".count('e')
8
```

O(n)

Lineal

```
>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
numeros = range(10)
parejas = []
for x in numeros:
    for y in numeros:
        parejas.append((x, y))
```

```
numeros = range(10)
parejas = []
for x in numeros:
    for y in numeros:
        parejas.append((x, y))
```

$O(n^2)$

Cuadrático

```
def fib(n):
    """ Return the n-th Fibonacci number."""
    if n in {0, 1}:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

>>> fib(10)
89
```

```
def fib(n):
    """ Return the n-th Fibonacci number."""
    if n in {0, 1}:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

>>> fib(10)
89
```

$O(2^n)$

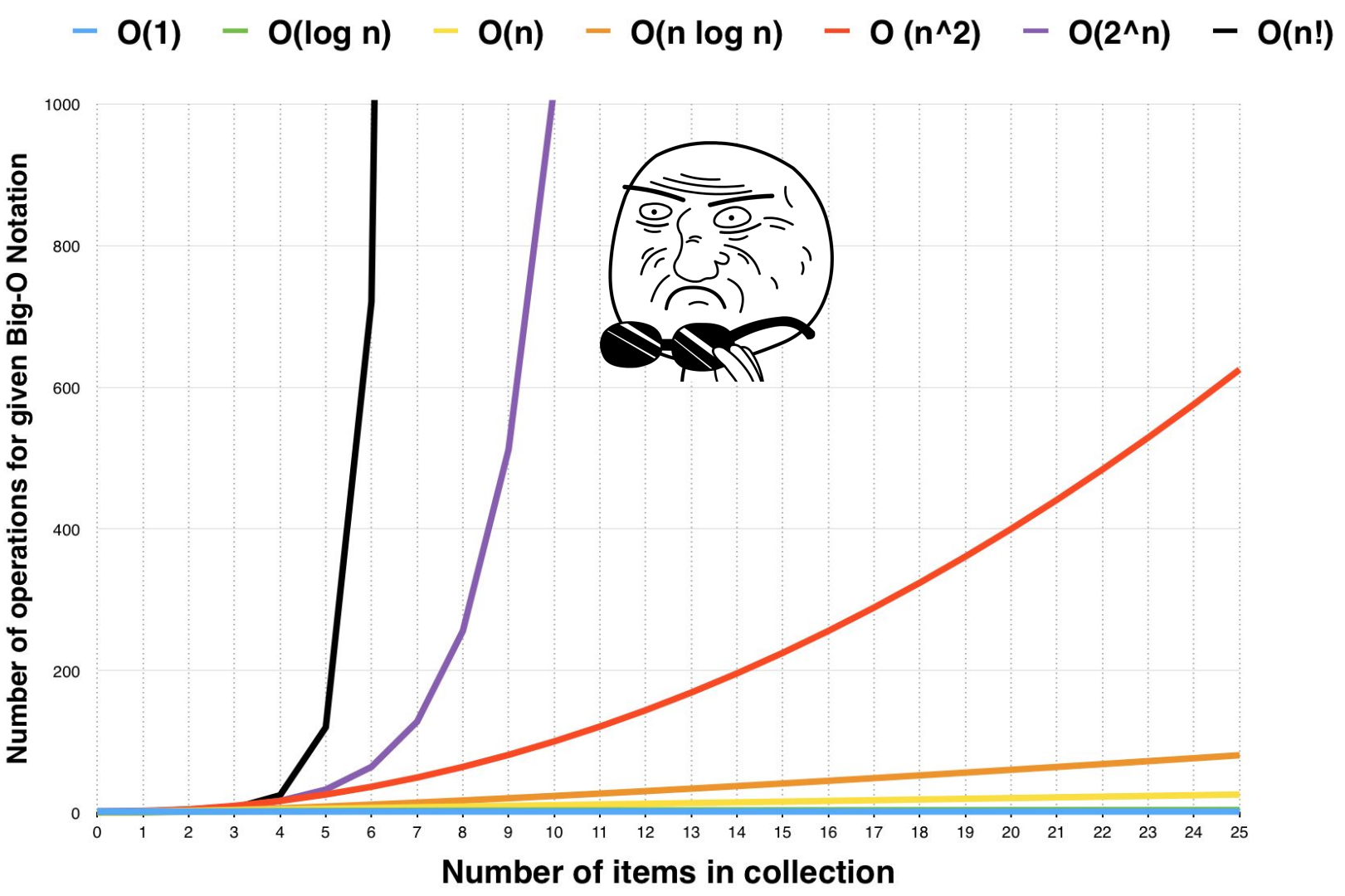
Exponencial

```
>>> numeros = range(1, int(1e18))
>>> bisect.bisect(numeros, 5988)
5988
```

```
>>> numeros = range(1, int(1e18))
>>> bisect.bisect(numeros, 5988)
5988
```

O(log n)

Logarítmico


O(1)
O(log n)
O(n)
O(n log n)
O (n^2)
O(2^n)
O(n!)
Number of operations for given Big-O Notation
0
200
400
600
800
1000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Number of items in collection


5586459
5632 03 753
066789 8127
2710339
4616 914 68
210103 1888
4410 9 36
1440107

Grado

Nos quedamos con el **monomio** de mayor grado

$$O(n^2 + n + \log n) =$$

$$\mathbf{O(n^2)}$$

Predomina cuando $\mathbf{n \to +\infty}$

Coeficientes

Eliminamos los coeficientes — son sólo una **constante**

$$O(5\,n) = O(n)$$

“¿Cómo determinas si una cadena de texto es una **permutación** de otra?”

```
def is_permutation(word, another):
    for letter in word:
        if word.count(letter) != another.count(letter):
            return False
    return True



>>> print(is_permutation("amor", "roma"))
True
```

```
def is_permutation(word, another):
    for letter in word:
        if word.count(letter) != another.count(letter):
            return False
    return True
```

$O(n^2)$

```
>>> print(is_permutation("amor", "roma"))
True
```


Mucho que aprender todavía tienes

```python
def is_permutation(word, another):

    word_counter = {}
    for letter in word:
        if letter in word_counter:
            word_counter[letter] += 1
        else:
            word_counter[letter] = 1

    # Hacemos lo mismo para 'another'...

    return word_counter == another_counter
```

```
def is_permutation(word, another):

    word_counter = {}
    for letter in word:
        if letter in word_counter:
            word_counter[letter] += 1
        else:
            word_counter[letter] = 1

    # Hacemos lo mismo para 'another'...

    return word_counter == another_counter
```

O(n)

defaultdict

```
>>> counter = collections.defaultdict(int)
>>> counter['a'] = 15
>>> print(counter['a'])
15
>>> print(counter['b'])
0
>>> counter['c'] += 1
>>> print(counter['c'])
1
```

Devuelve cero, no **KeyError**

```python
import collections

def is_permutation(word, another):

    word_counter = collections.defaultdict(int)
    for letter in word:
        word_counter[letter] += 1

    # Hacemos lo mismo para 'another'...

    return word_counter == another_counter
```


Aún mejor

Counter

```
>>> counter = collections.Counter('abracadabra')
>>> counter
Counter({'a': 5, 'r': 2, 'b': 2, 'c': 1, 'd': 1})
>>> print(counter['a'])
5
>>> print(counter['x'])
0
```

Devuelve cero, no **KeyError**

```python
import collections

def is_permutation(word, another):

    word_counter    = collections.Counter(word)
    another_counter = collections.Counter(another)
    return word_counter == another_counter
```

En una única línea...

```python
from collections import Counter

def is_permutation(word, another):
    return Counter(word) == Counter(another)
```

```
def is_permutation(word, another):
    return sorted(word) == sorted(another)
```

```
def is_permutation(word, another):
    return sorted(word) == sorted(another)
```

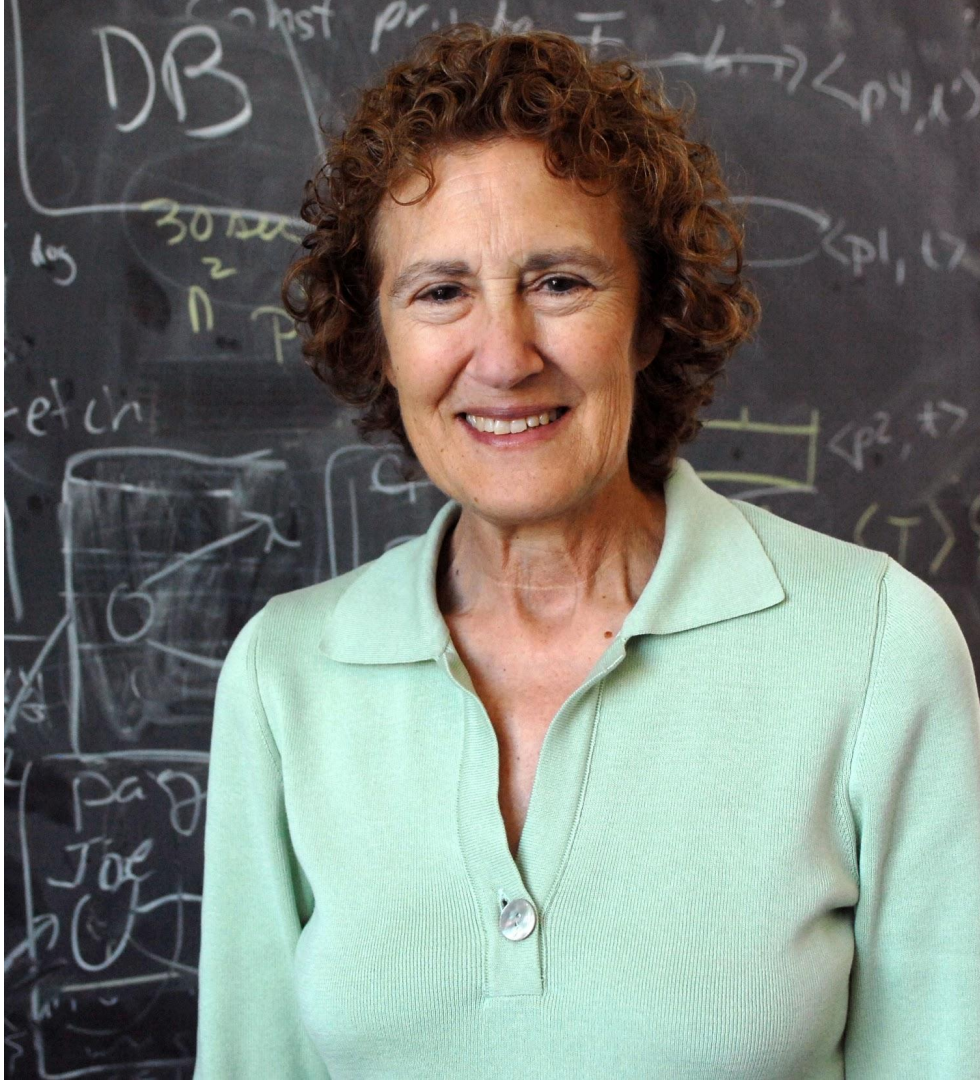
O(n log n)

Más Pythónica

¡Más lenta!

> You never need optimal performance, **you need good-enough performance.** [...] Programmers are far too hung up with performance

— Barbara Liskov




Oh, y hablando de
Big O

```
import itertools

def is_permutation(word, another):

    for permutation in itertools.permutations(word):
        if permutation == tuple(another):
            return True
    return False
```

```
import itertools

def is_permutation(word, another):

    for permutation in itertools.permutations(word):
        if permutation == tuple(another):
            return True
    return False
```

O(n!)

Explosión combinatoria

```
>>> itertools.permutations("recluta")
<itertools.permutations object at 0x7fdbe3e75d70>
>>> len(list(itertools.permutations("recluta")))
5040
>>> len("supercalifragilisticoespialidoso")
32
>>> math.factorial(_)
2.63131e+35    # billions and billions...
```

Objeto **iterador**

```
sorted(x) == sorted(y)
```

Es **más** sencillo

¿Por qué nos gusta Python?

Los argumentos de por qué Python **no mola** ya nos los conocemos...

- ¡El sangrado obligatorio!
- ¡El tipado dinámico!
- ¡El parámetro **self** explícito!
- ¡Es muy, muy, *muy* lento!
- ....

> We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**

— Donald Knuth

simplicity.

“**It's easier to ask forgiveness** than it is to get permission”

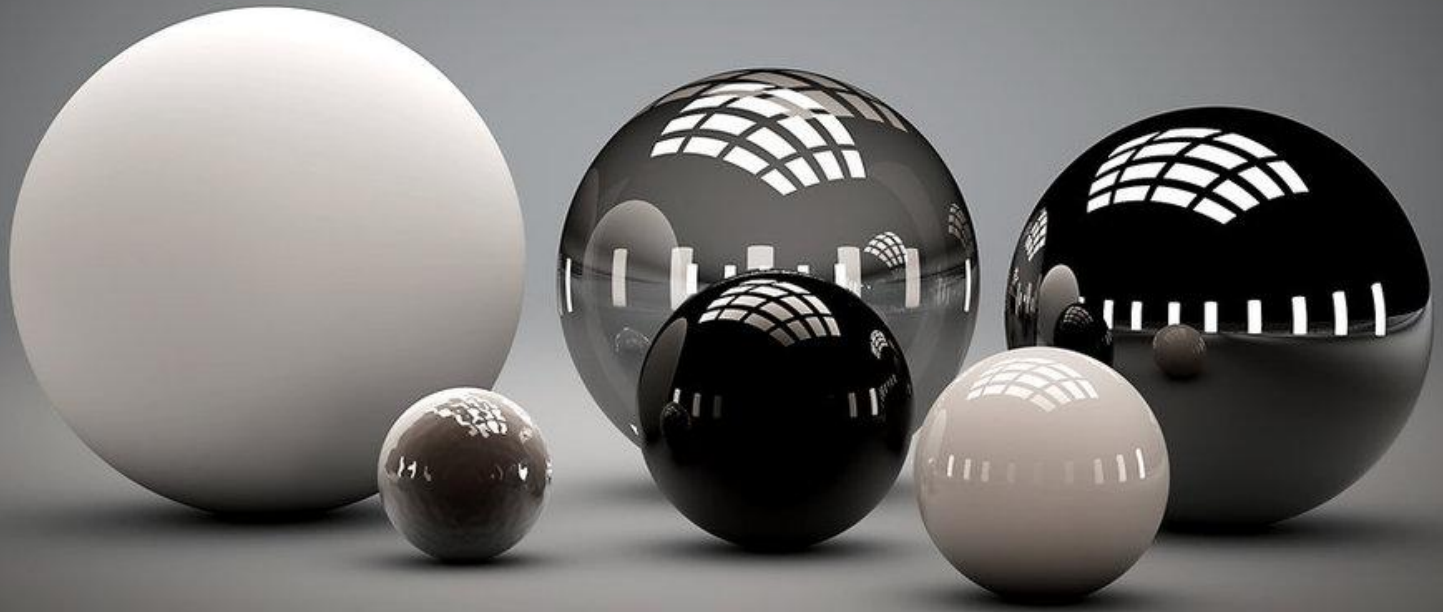
— Grace Hopper

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
[...]
There should be one-- and preferably only one --
obvious way to do it.
```

“**Simplicity** is prerequisite for reliability”

— Edsger W. Dijkstra

Heap



```
n = 5
numeros = [2, 4, 6, 9]
numeros.append(n)              # O(1)
numeros.sort()                 # O(n log n)
siguiente = numeros.pop(0)     # O(n)

>>> print(siguiente)
2
>>> print(numeros)
[4, 5, 6, 9]
```

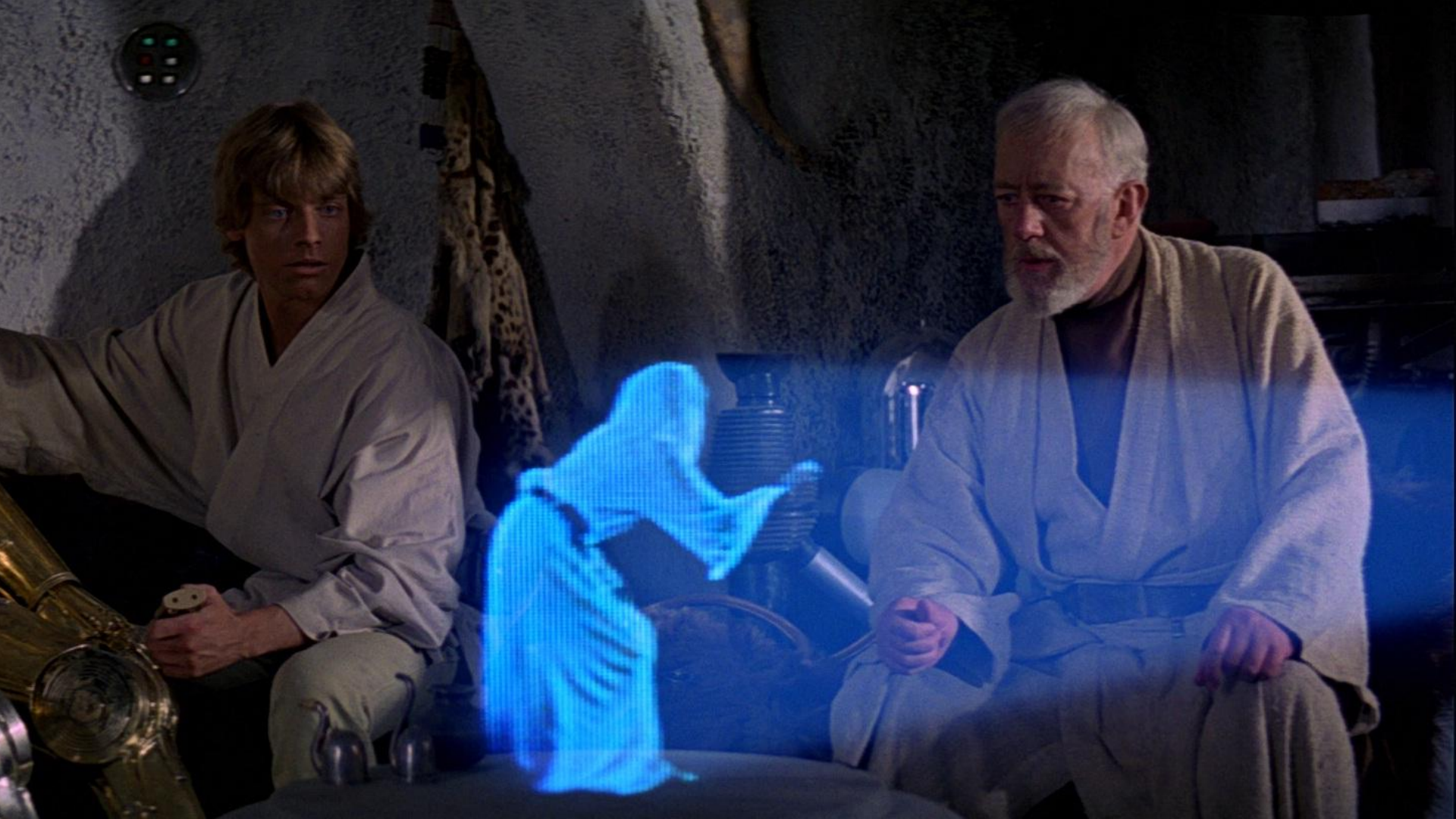
```
n = 5
numeros = [2, 4, 6, 9]
numeros.append(n)           # O(1)
numeros.sort()              # O(n log n)
siguiente = numeros.pop(0)  # O(n)

>>> print(siguiente)
2
>>> print(numeros)
[4, 5, 6, 9]
```

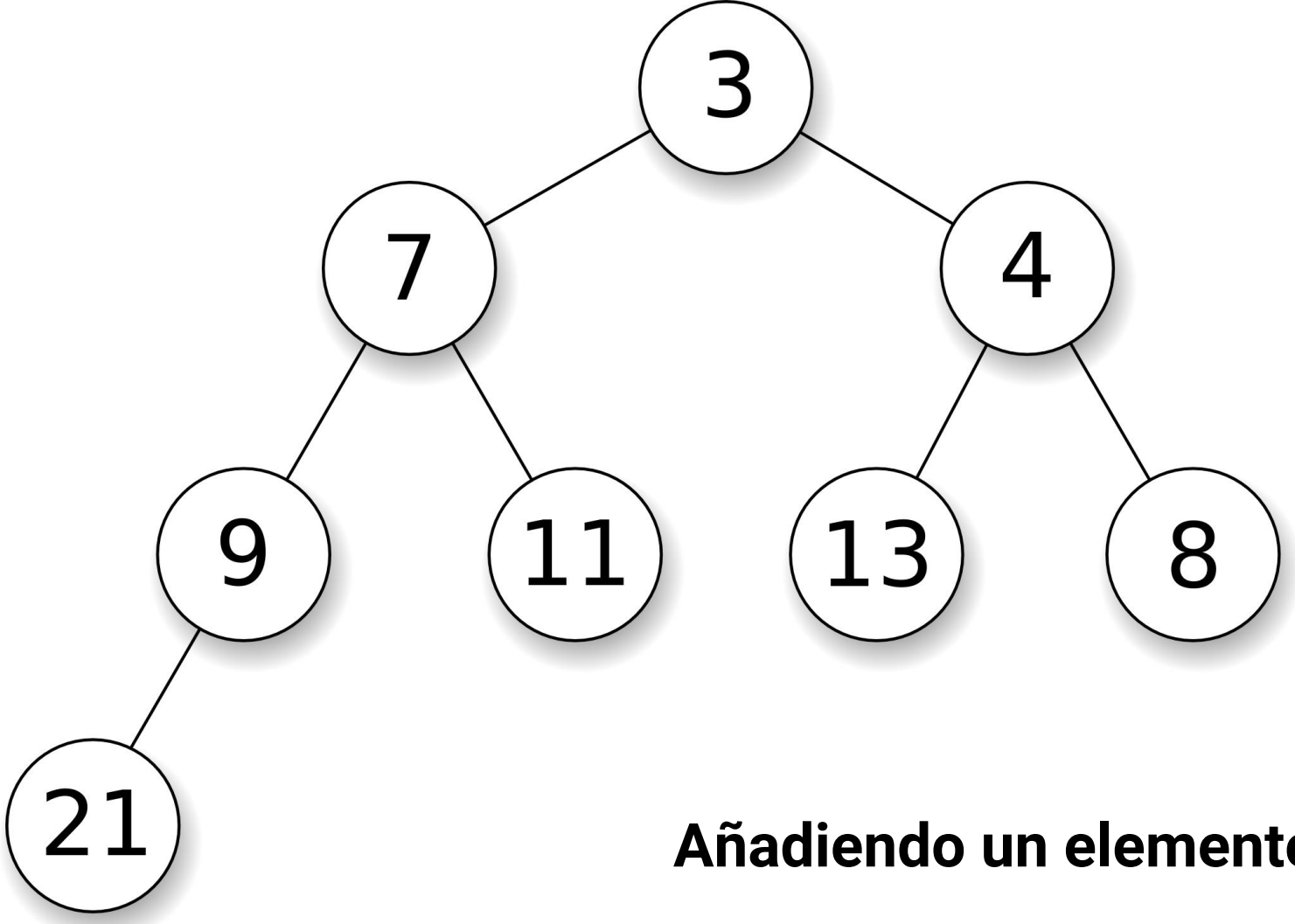
O(n log n)

```
import bisect

n = 5
numeros = [2, 4, 6, 9]
index = bisect.bisect(numeros, n) # O(log n)
numeros.insert(index, n)          # O(n)
siguiente = numeros.pop(0)        # O(n)

>>> siguiente
2
>>> numeros
[4, 5, 6, 9]
```

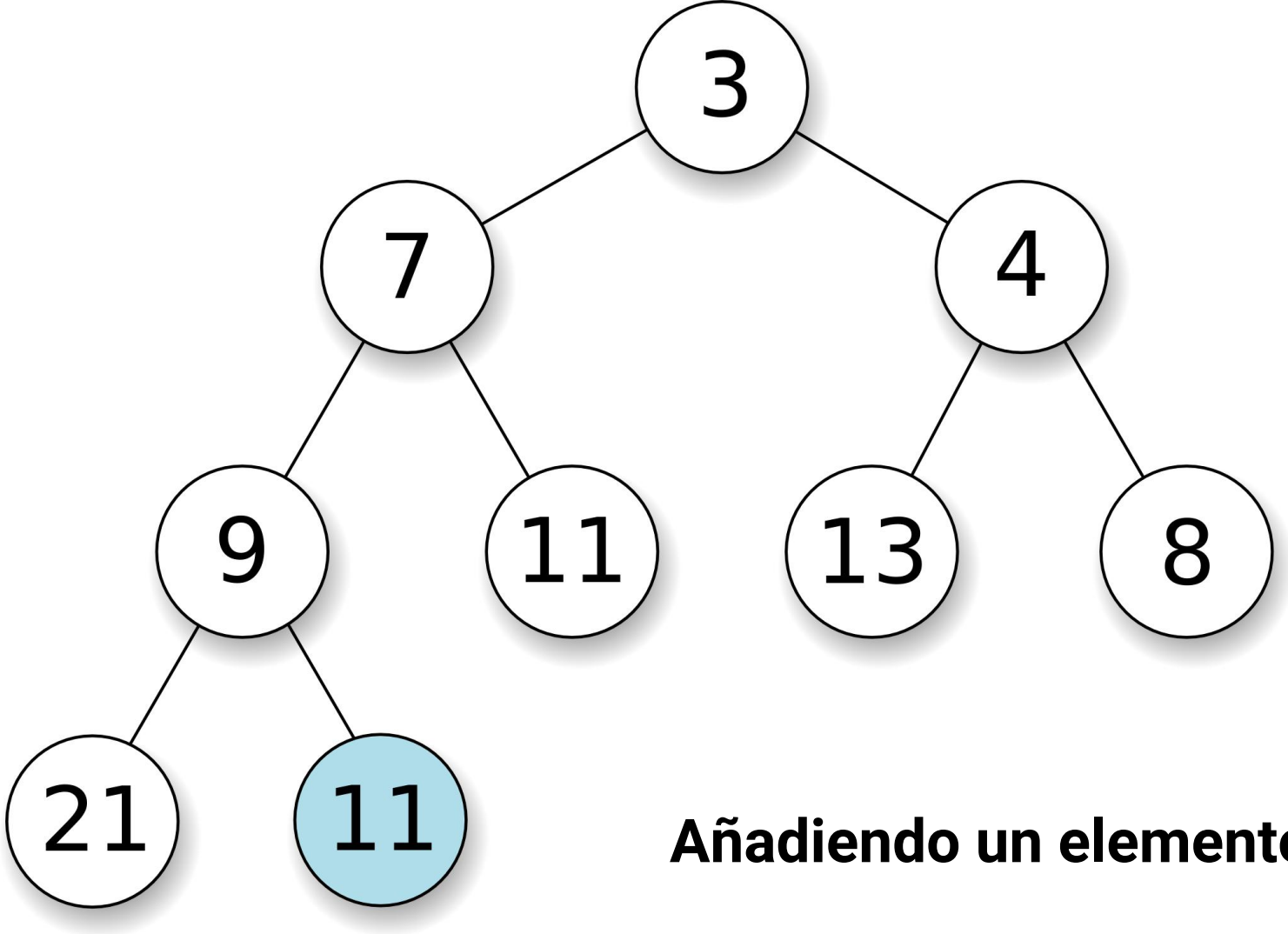
```python
import bisect

n = 5
numeros = [2, 4, 6, 9]
index = bisect.bisect(numeros, n) # O(log n)
numeros.insert(index, n)          # O(n)
siguiente = numeros.pop(0)        # O(n)

>>> siguiente
2
>>> numeros
[4, 5, 6, 9]
```

O(n)

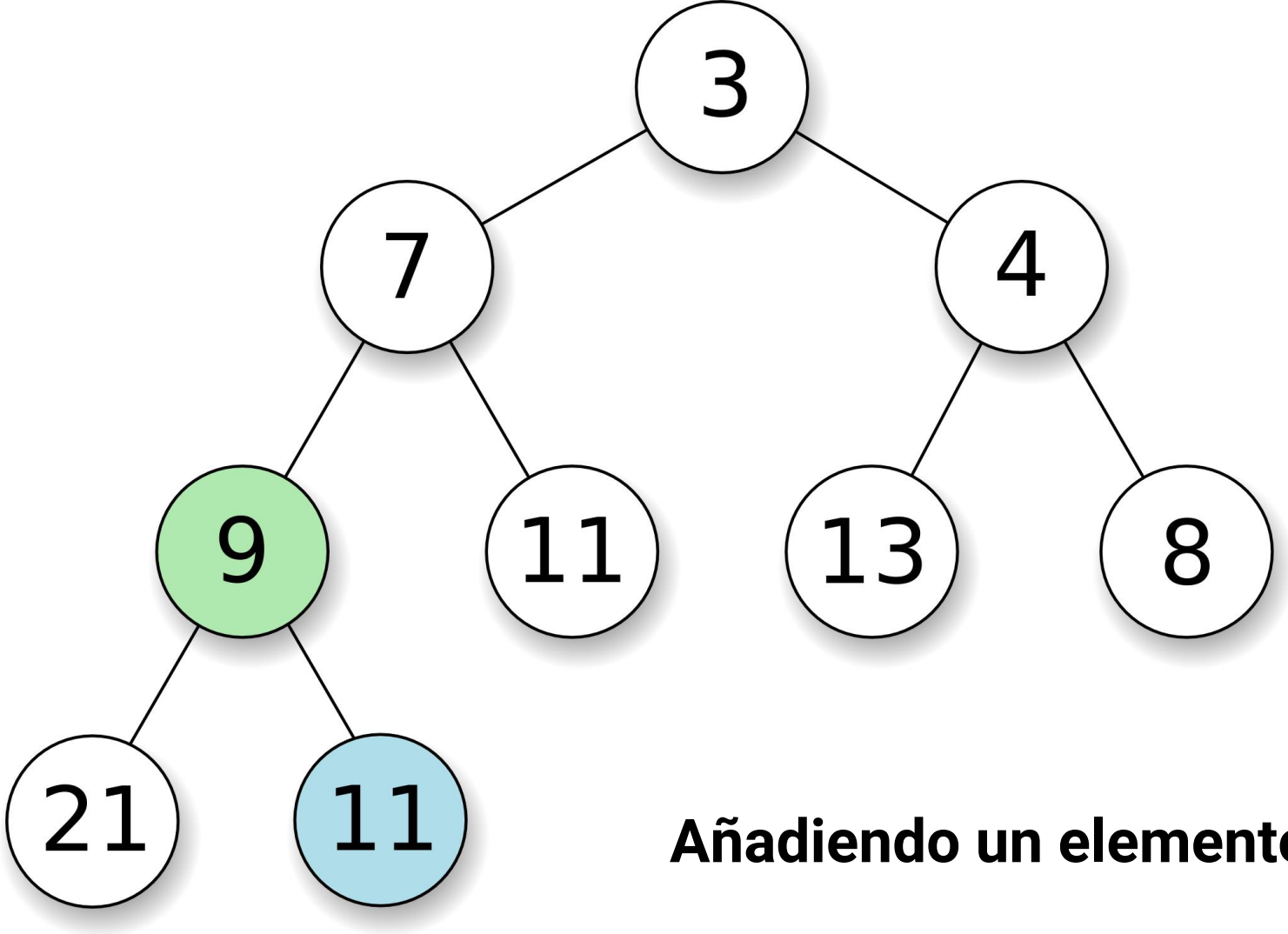




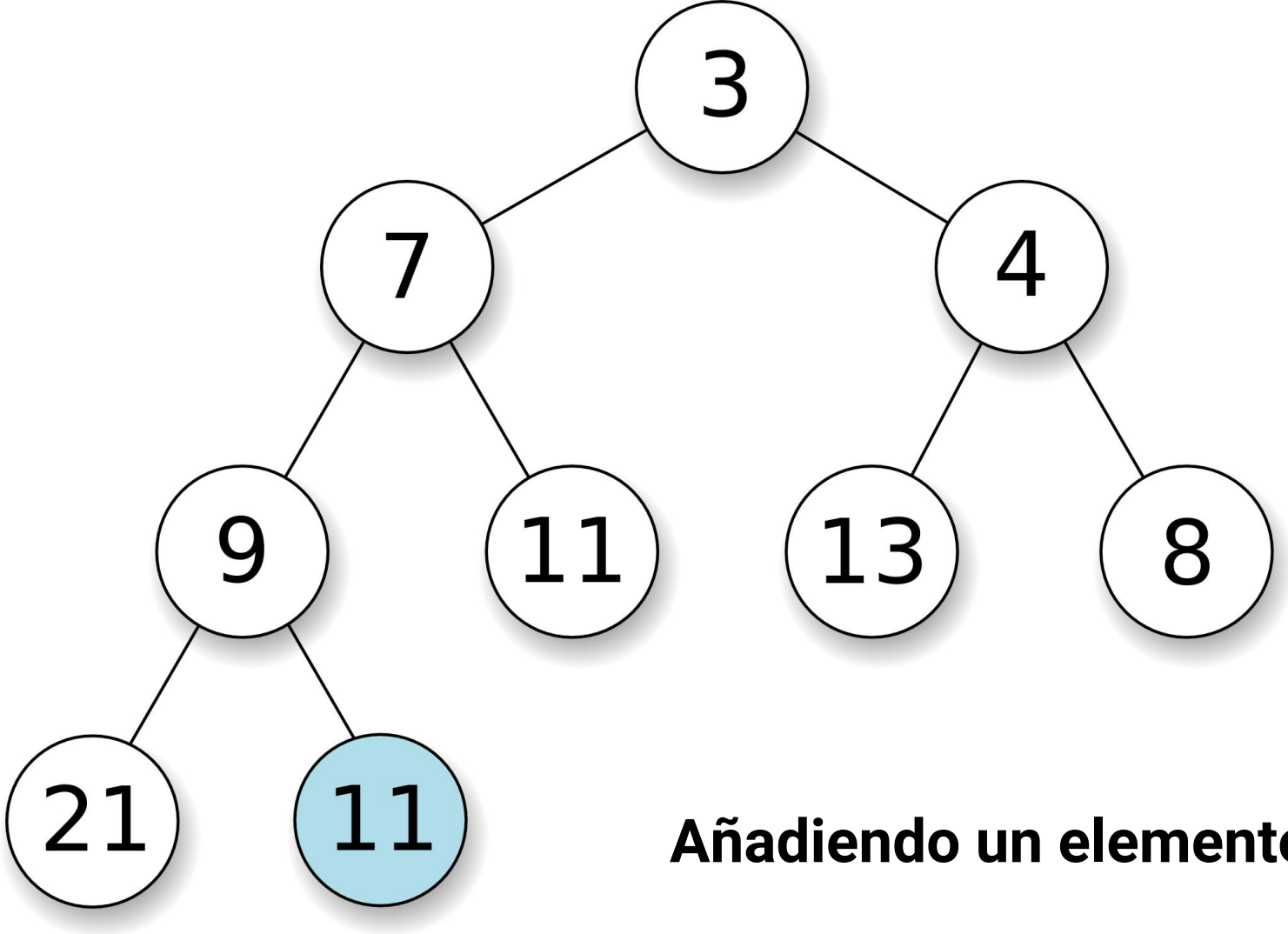
Añadiendo un elemento (I)



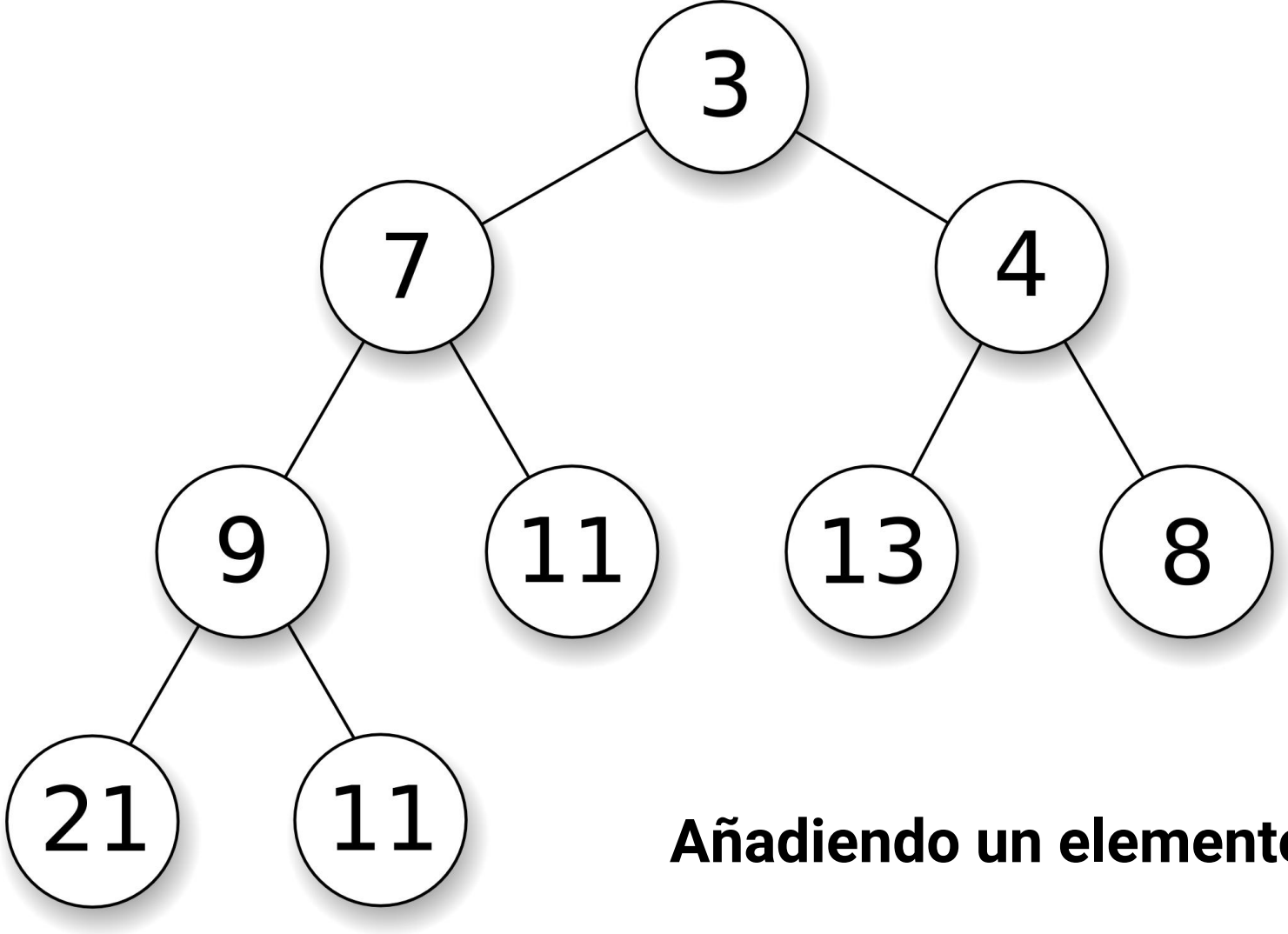
Añadiendo un elemento (I)



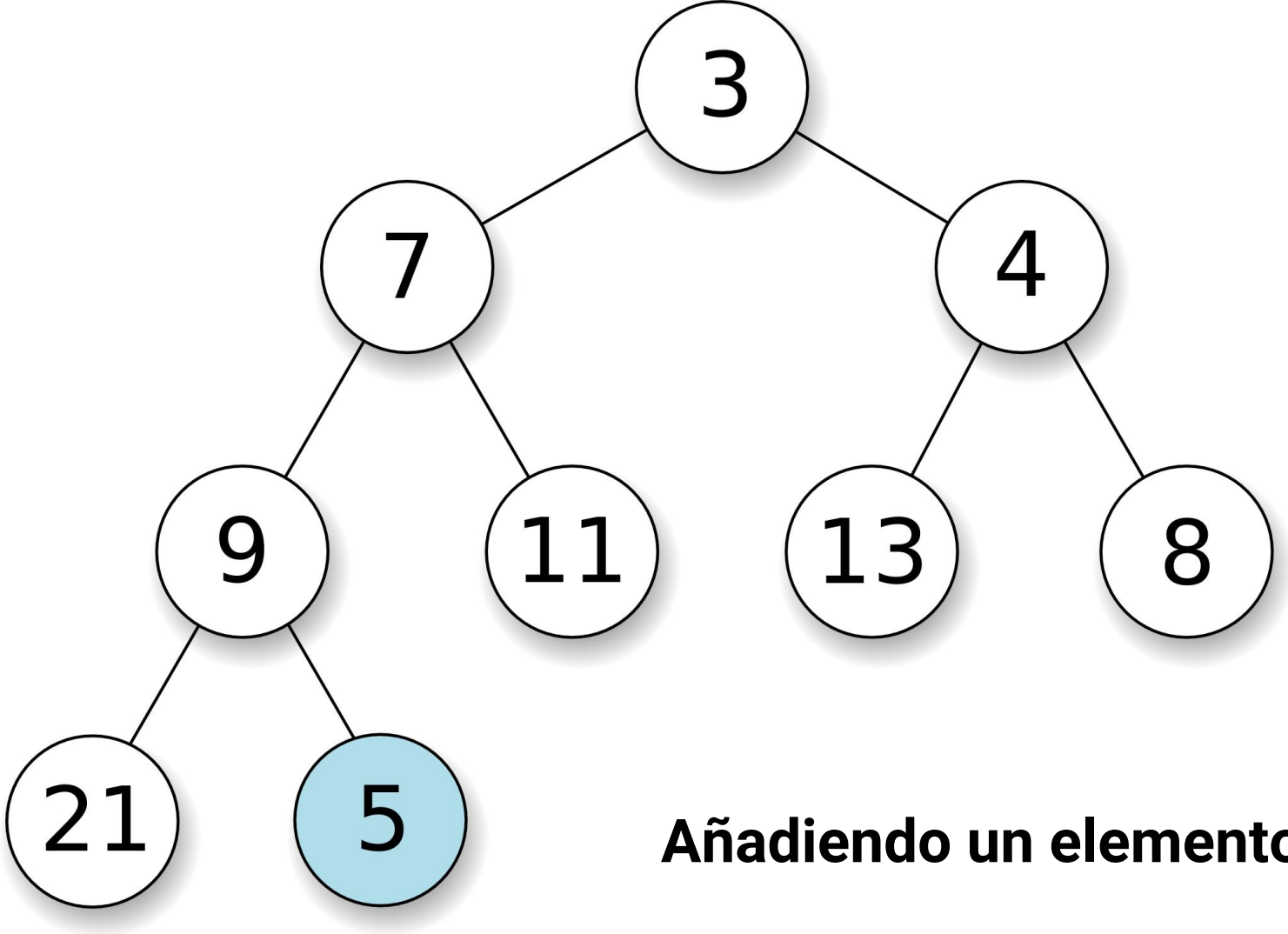
Añadiendo un elemento (I)



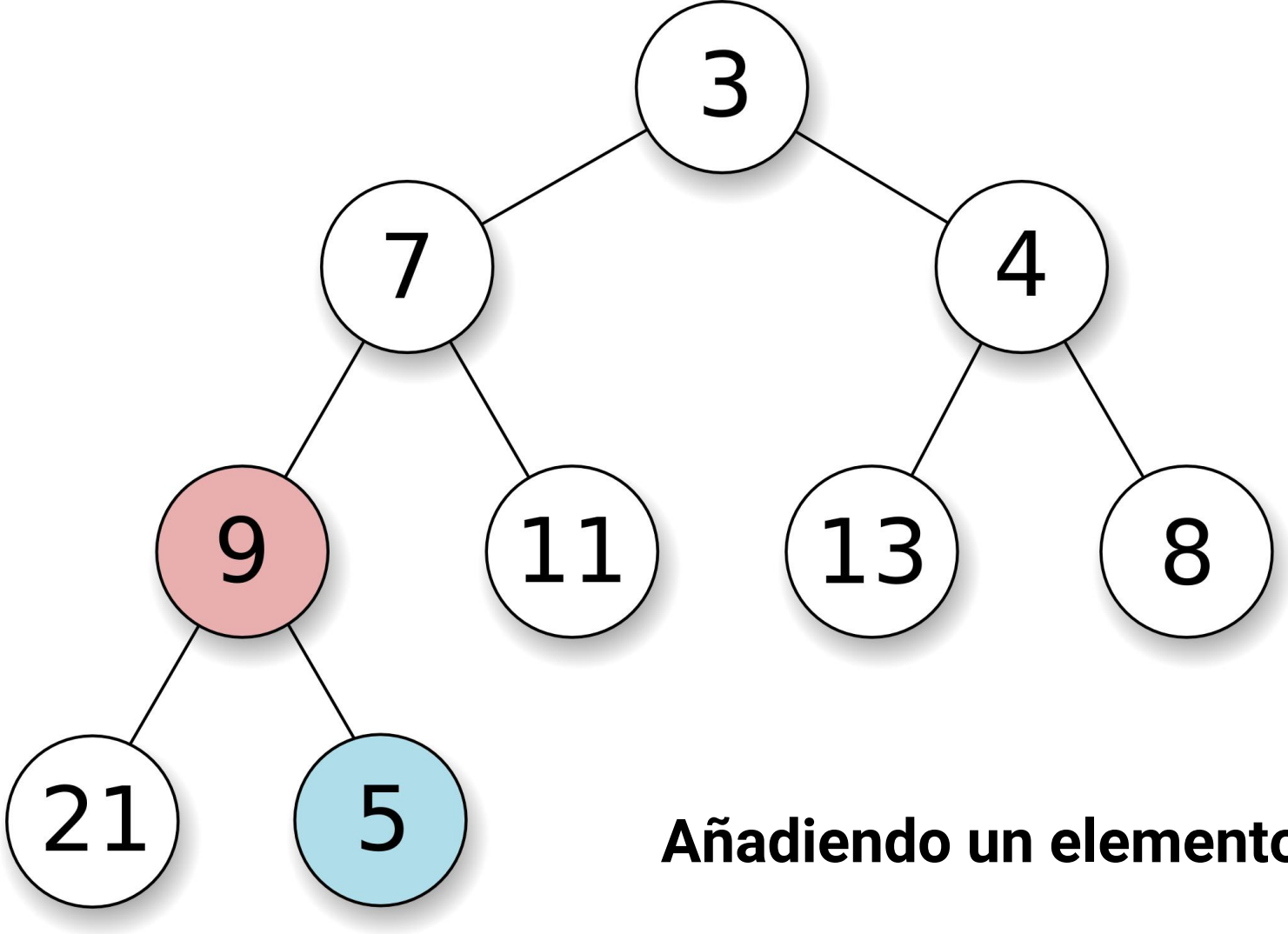
Añadiendo un elemento (I)



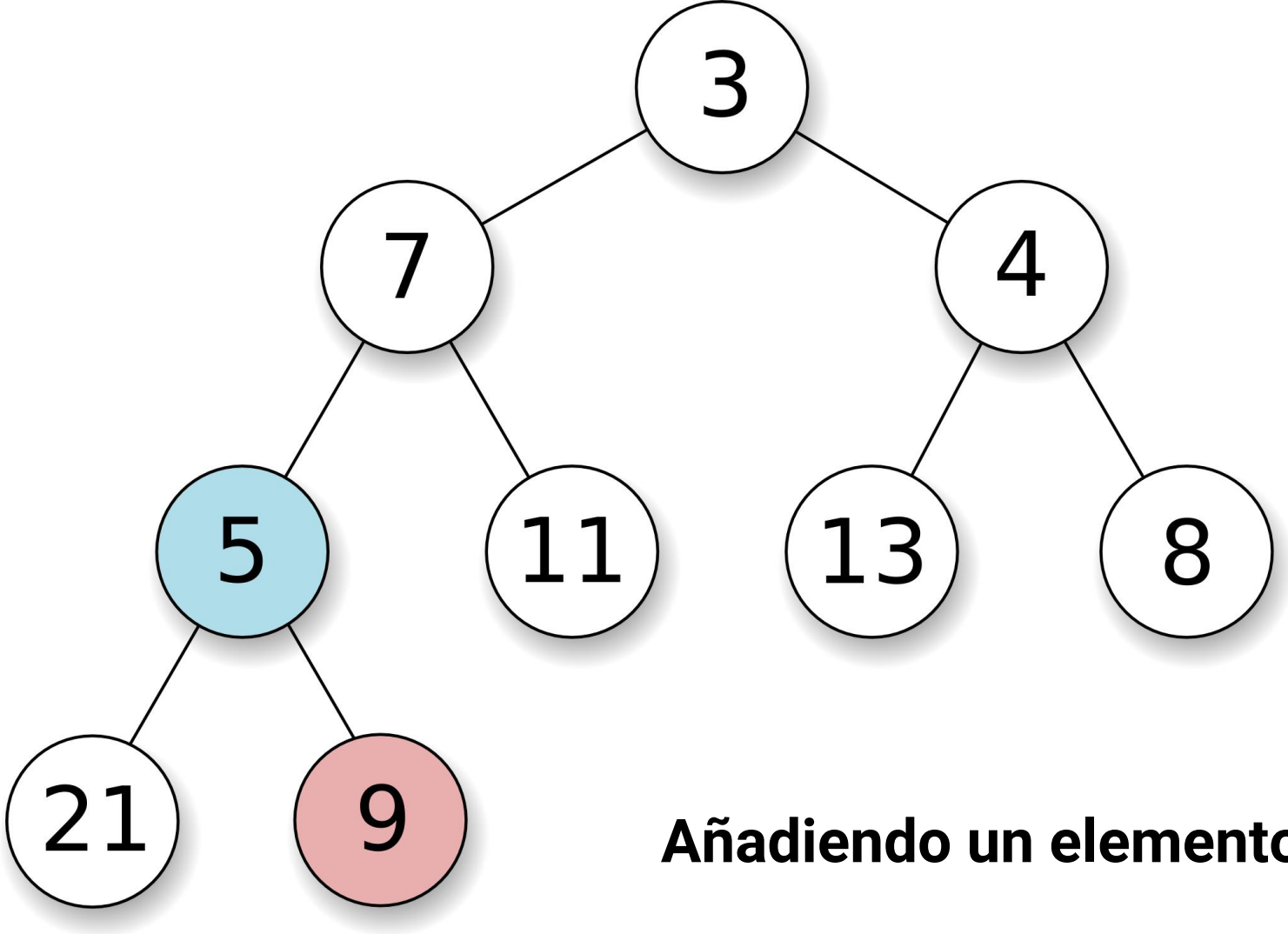
Añadiendo un elemento (I)



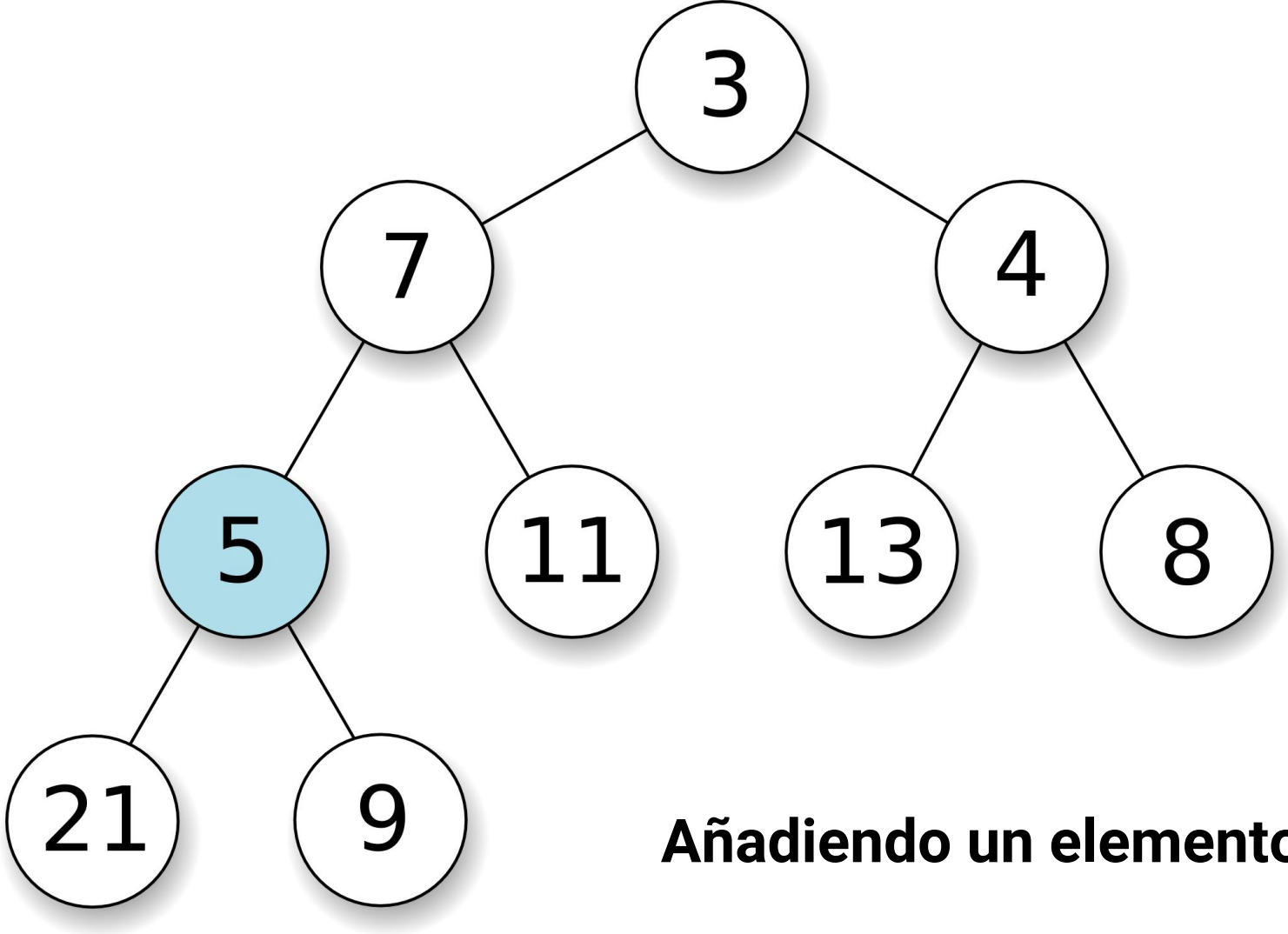
Añadiendo un elemento (II)



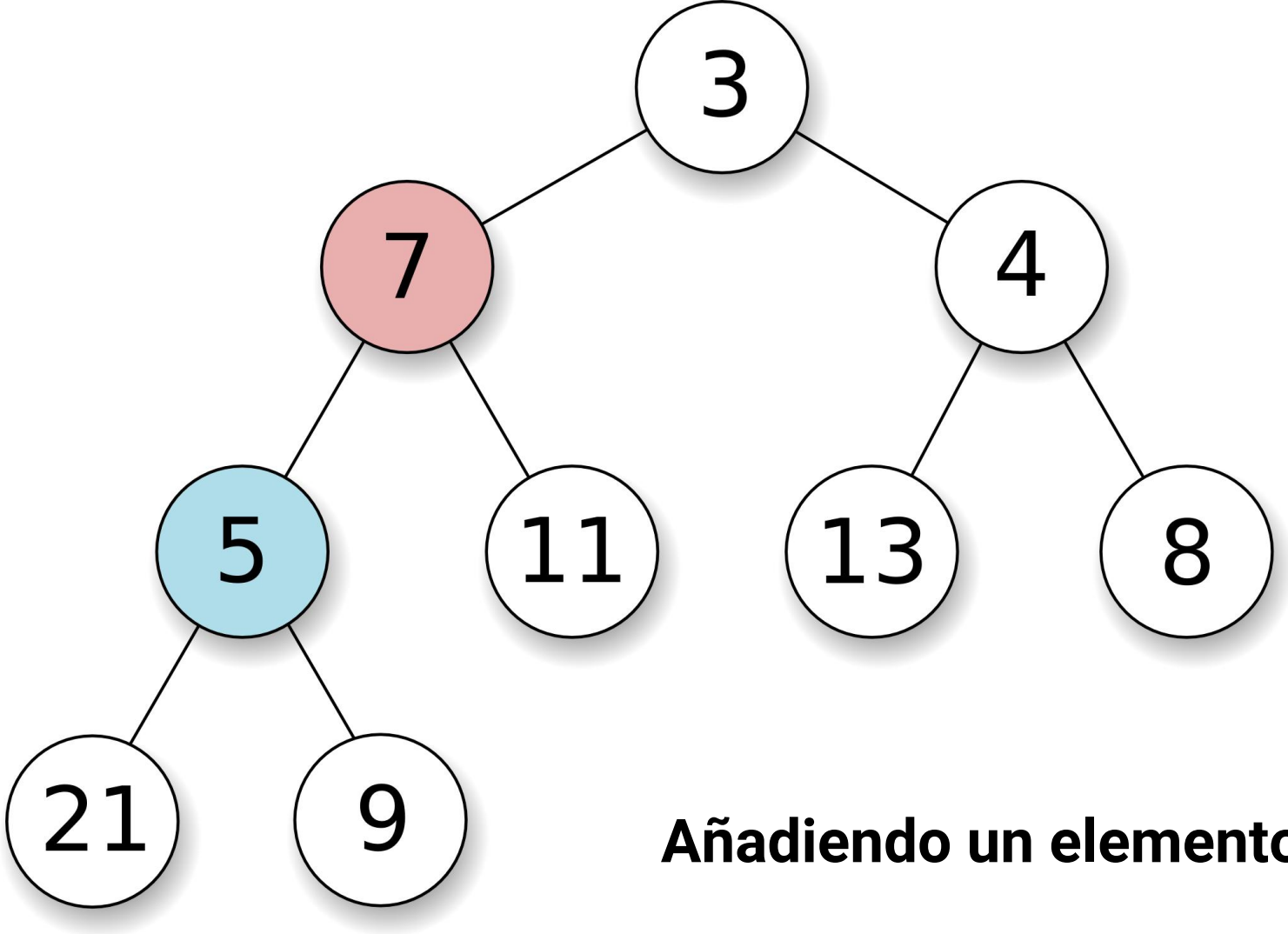
Añadiendo un elemento (II)



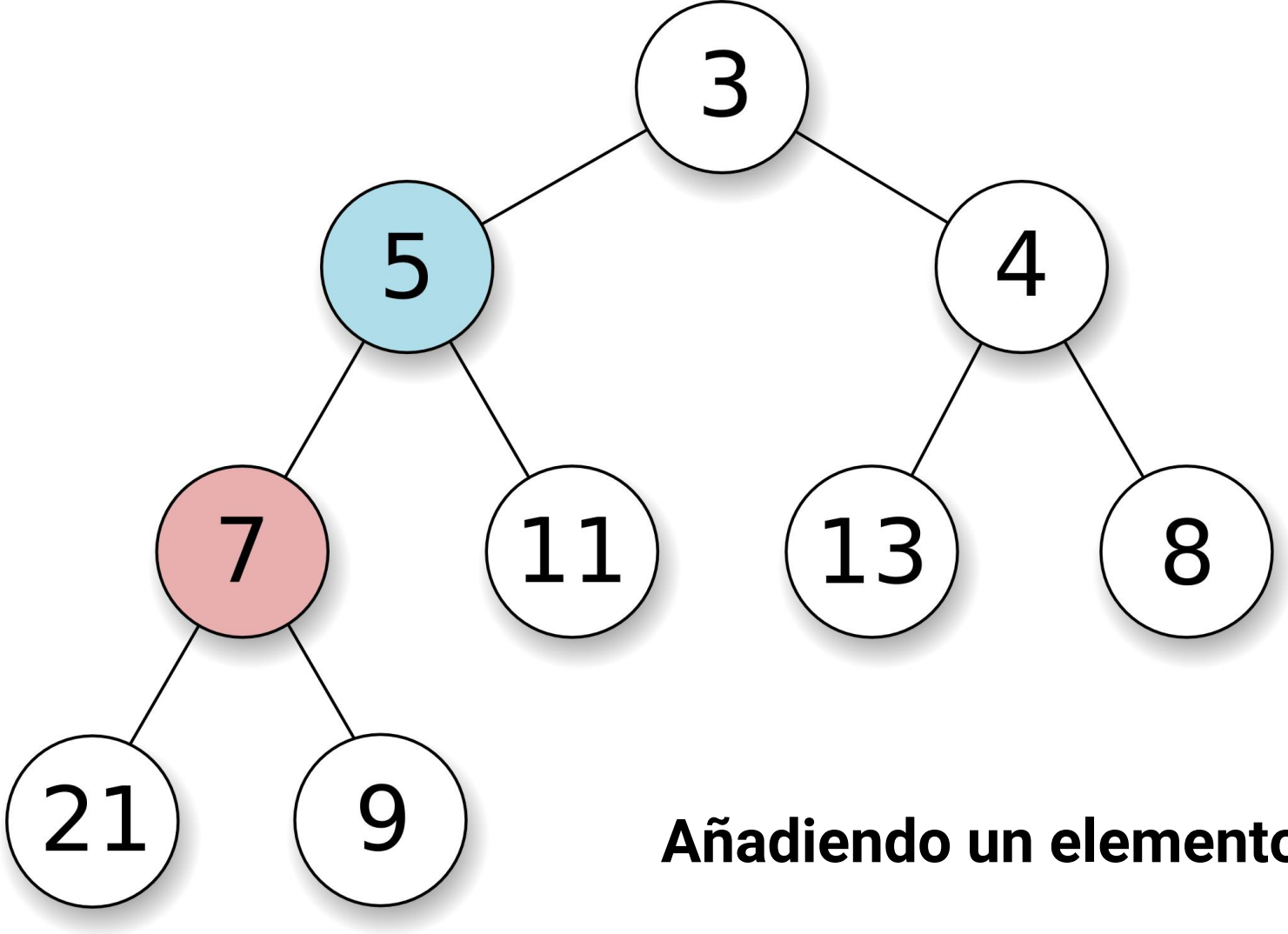
Añadiendo un elemento (II)



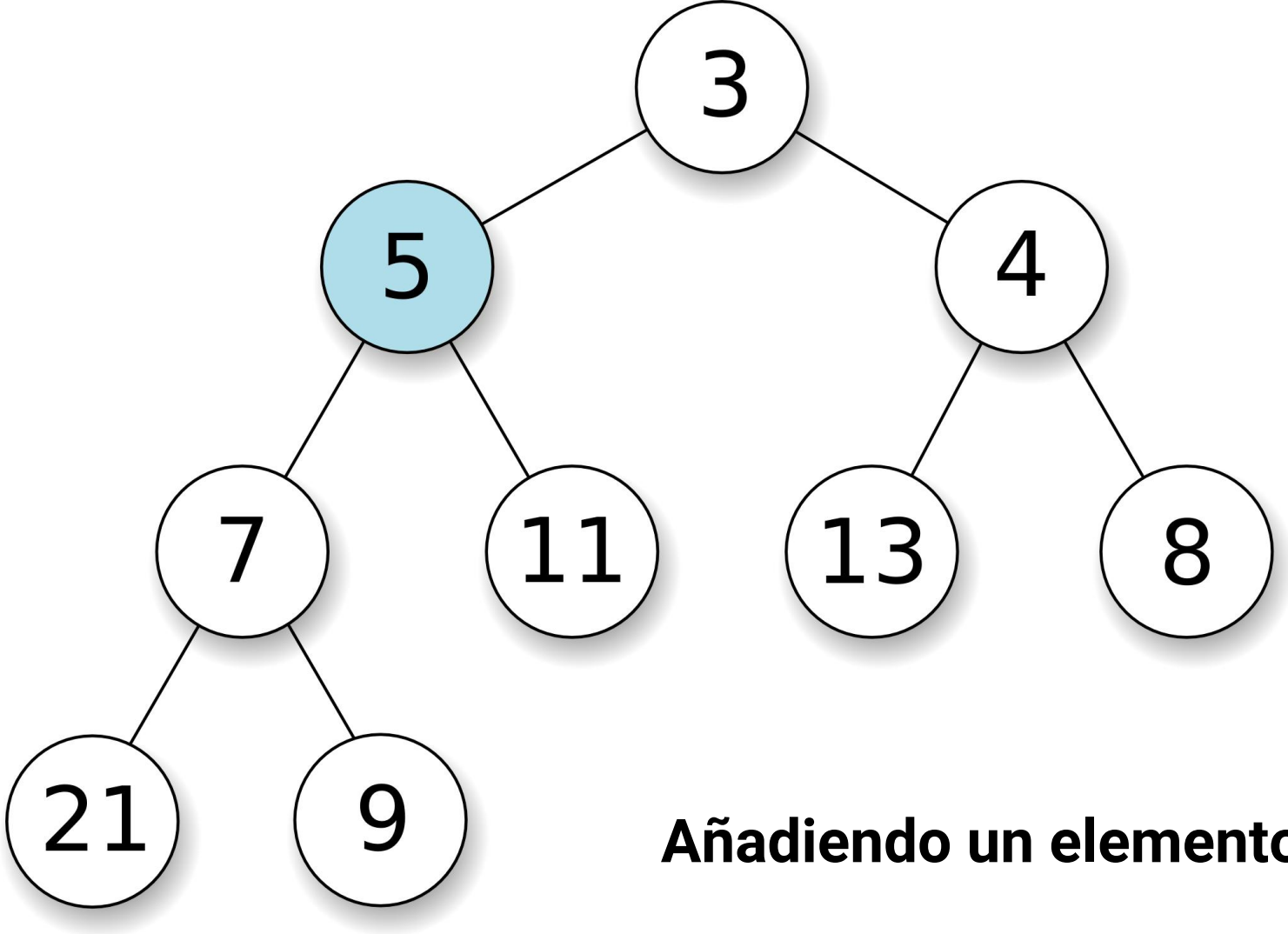
Añadiendo un elemento (II)



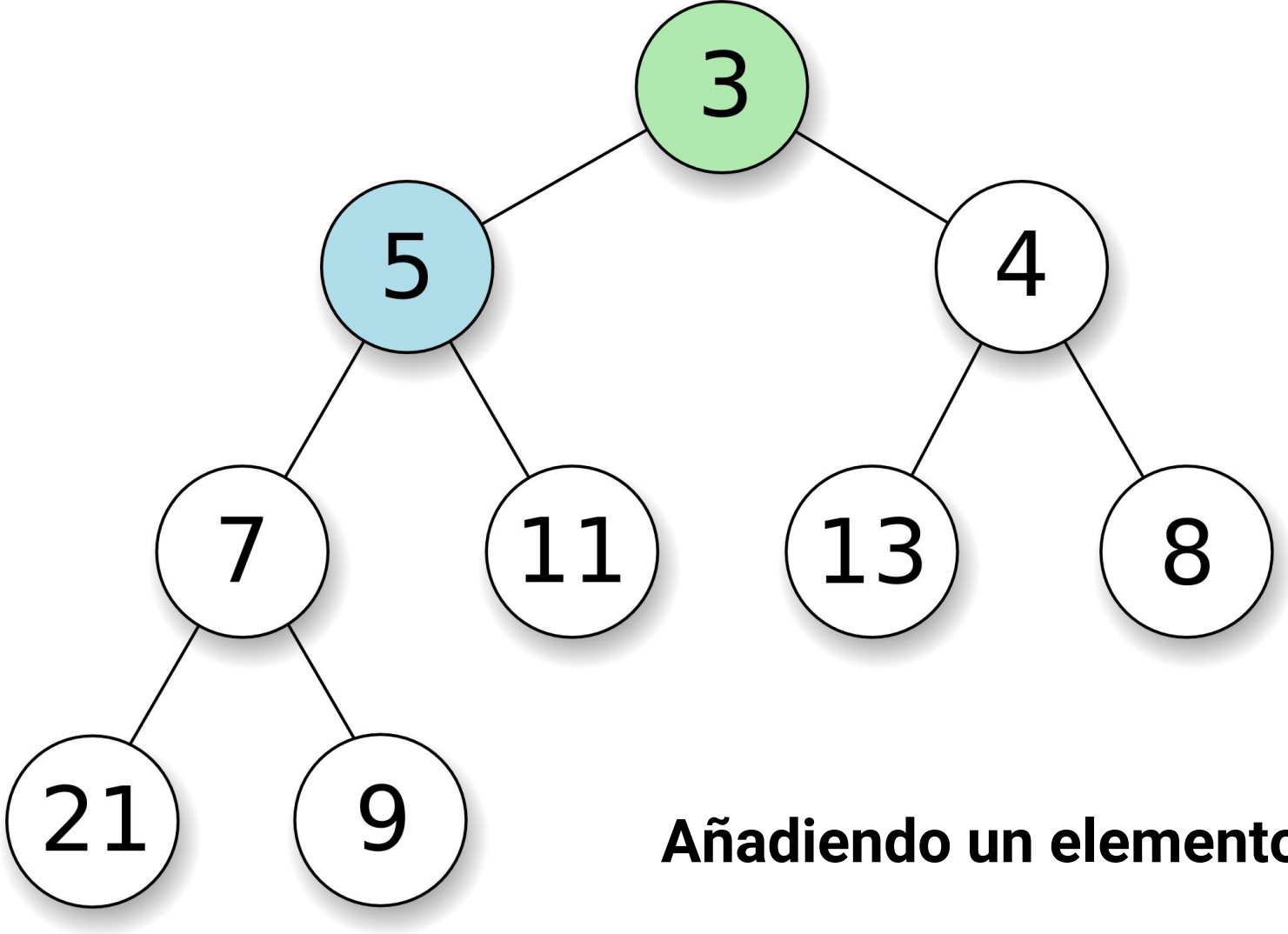
Añadiendo un elemento (II)



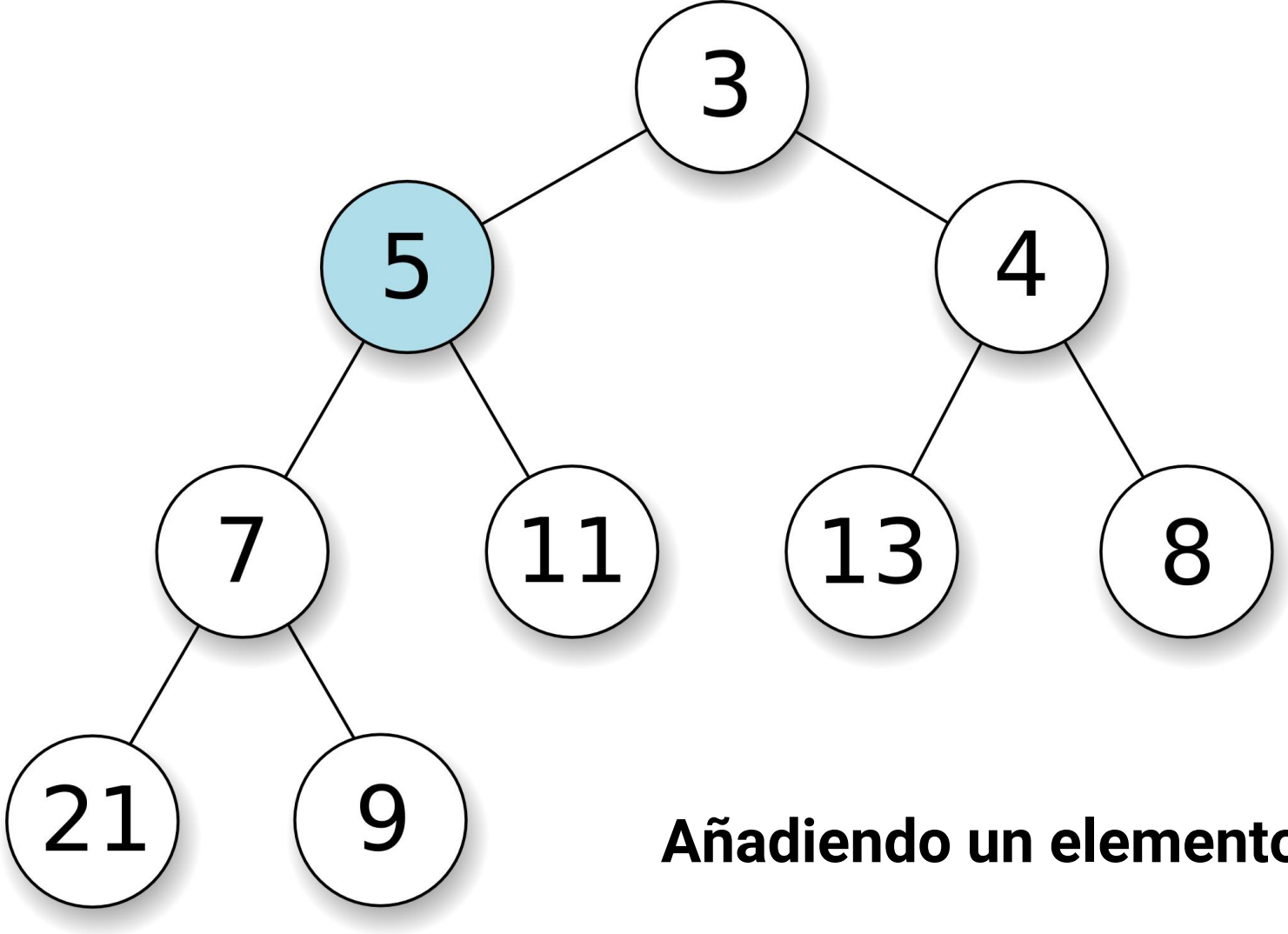
Añadiendo un elemento (II)



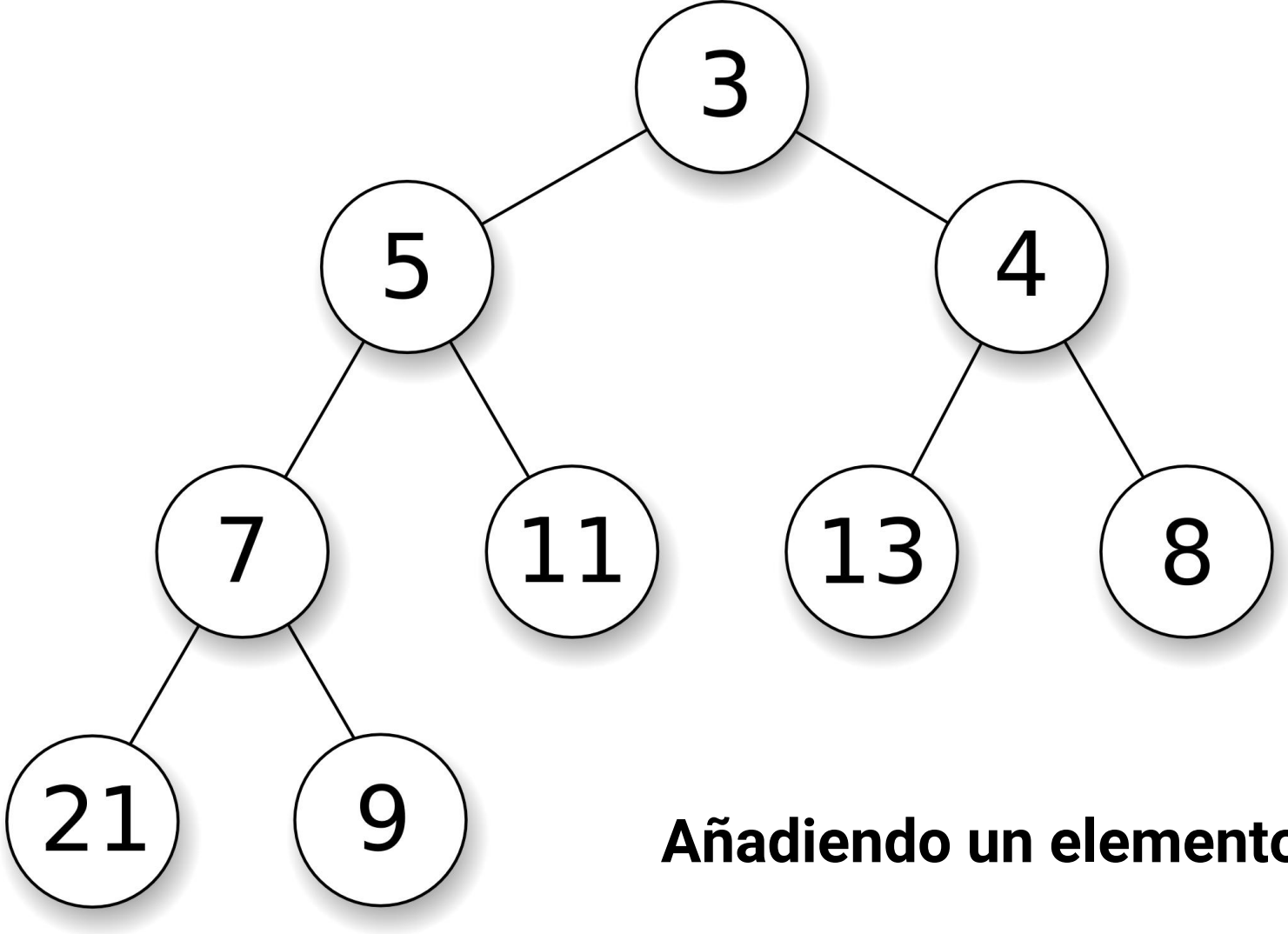
Añadiendo un elemento (II)



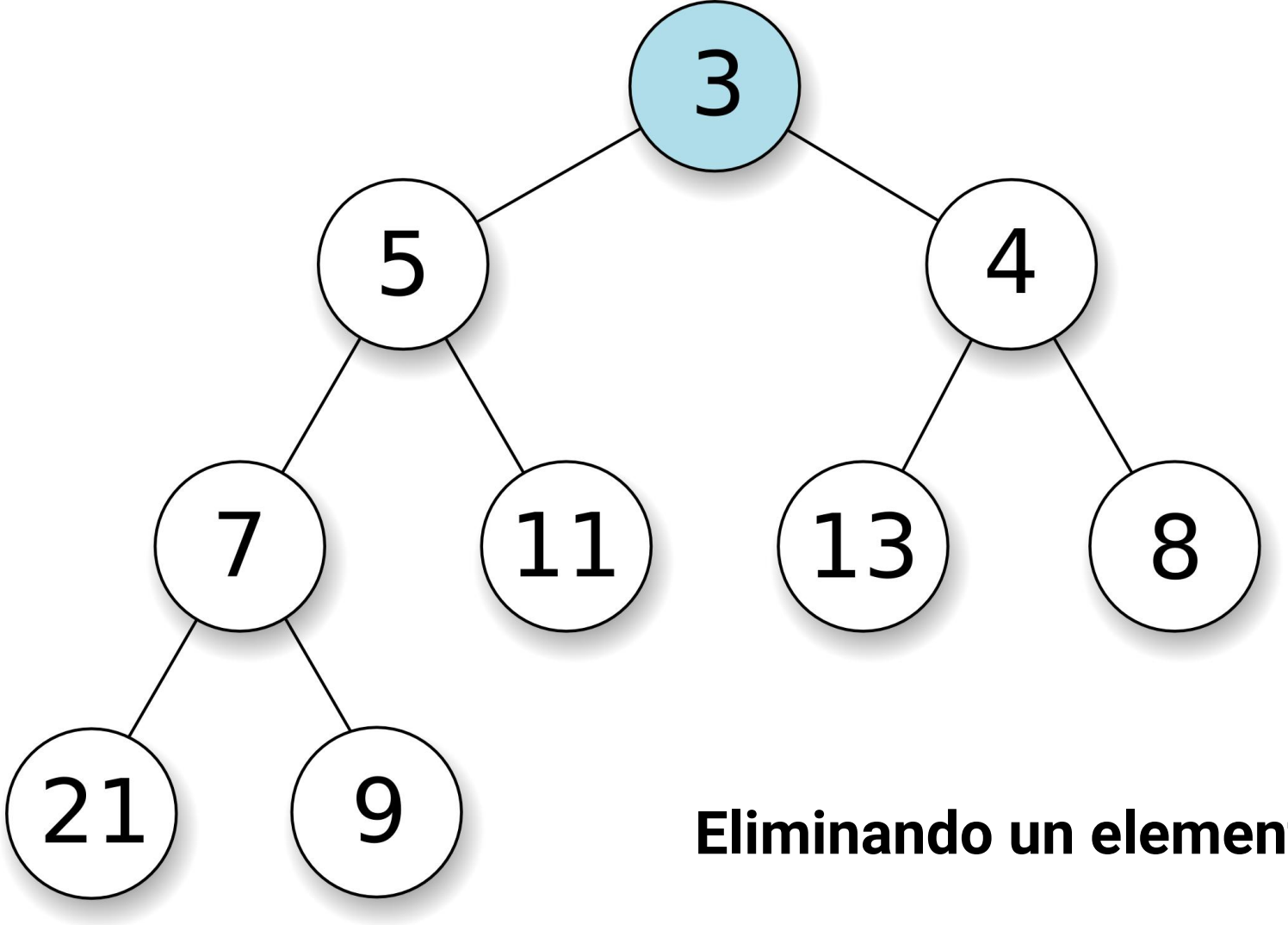
Añadiendo un elemento (II)



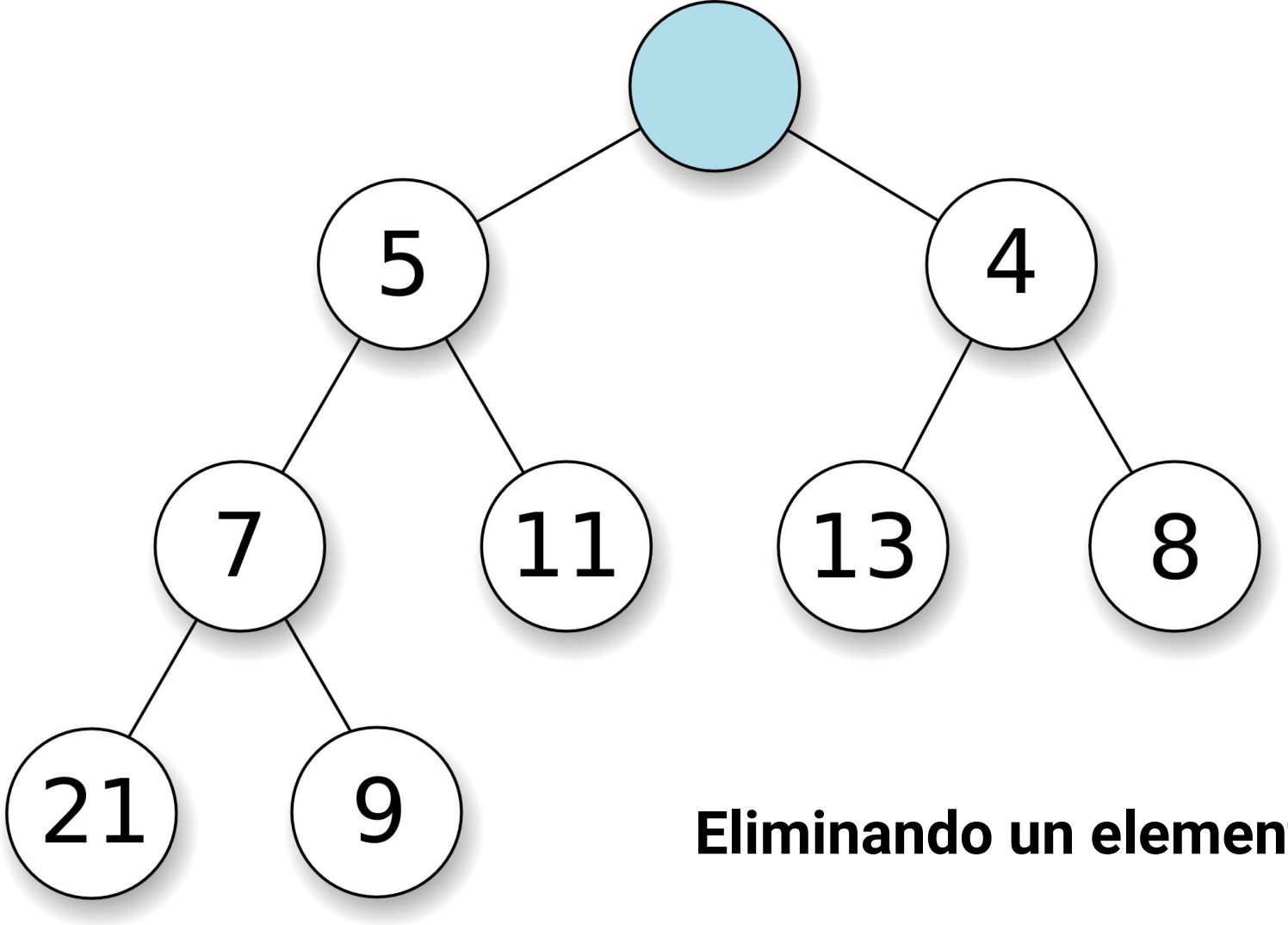
Añadiendo un elemento (II)



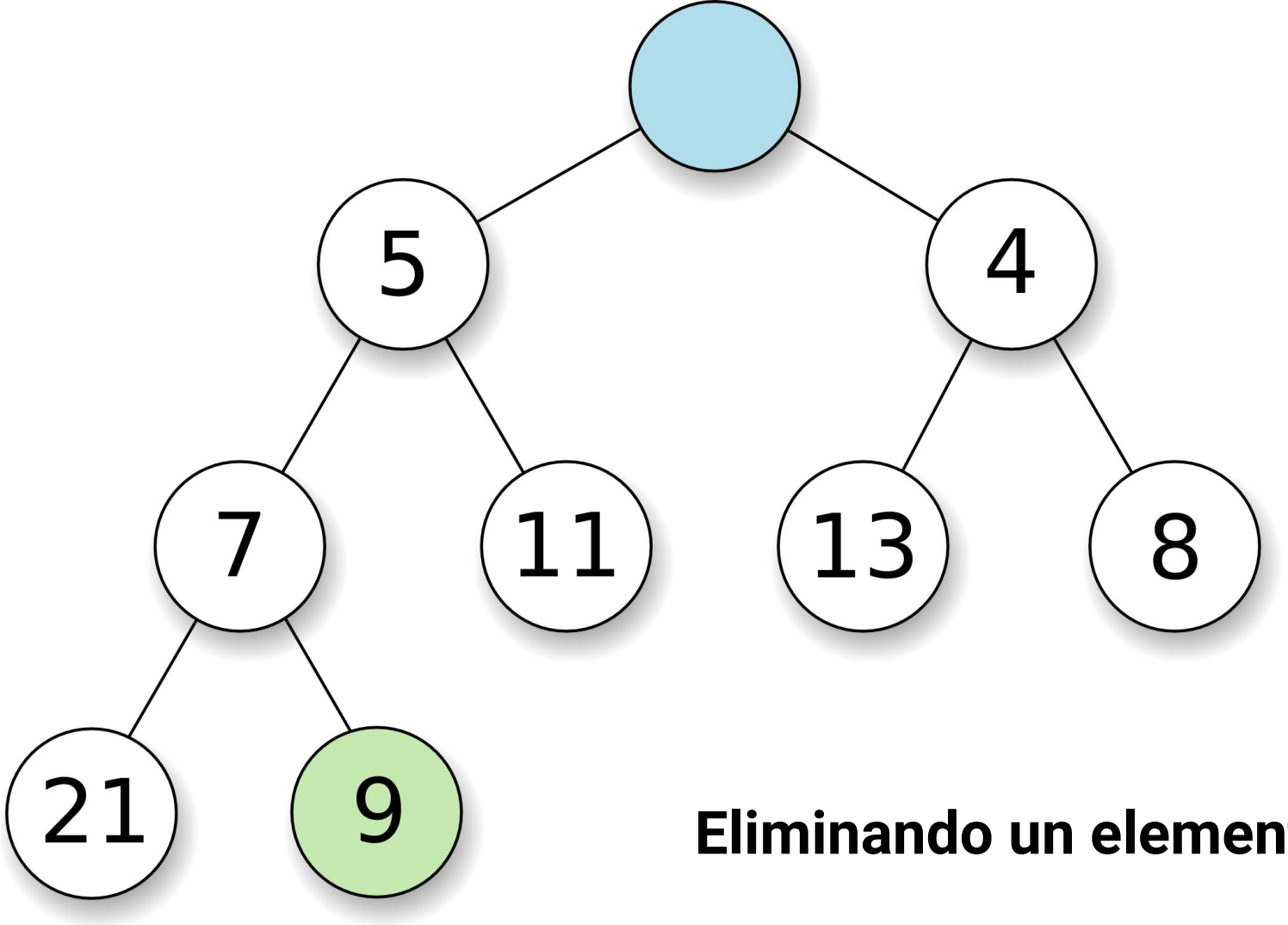
Añadiendo un elemento (II)



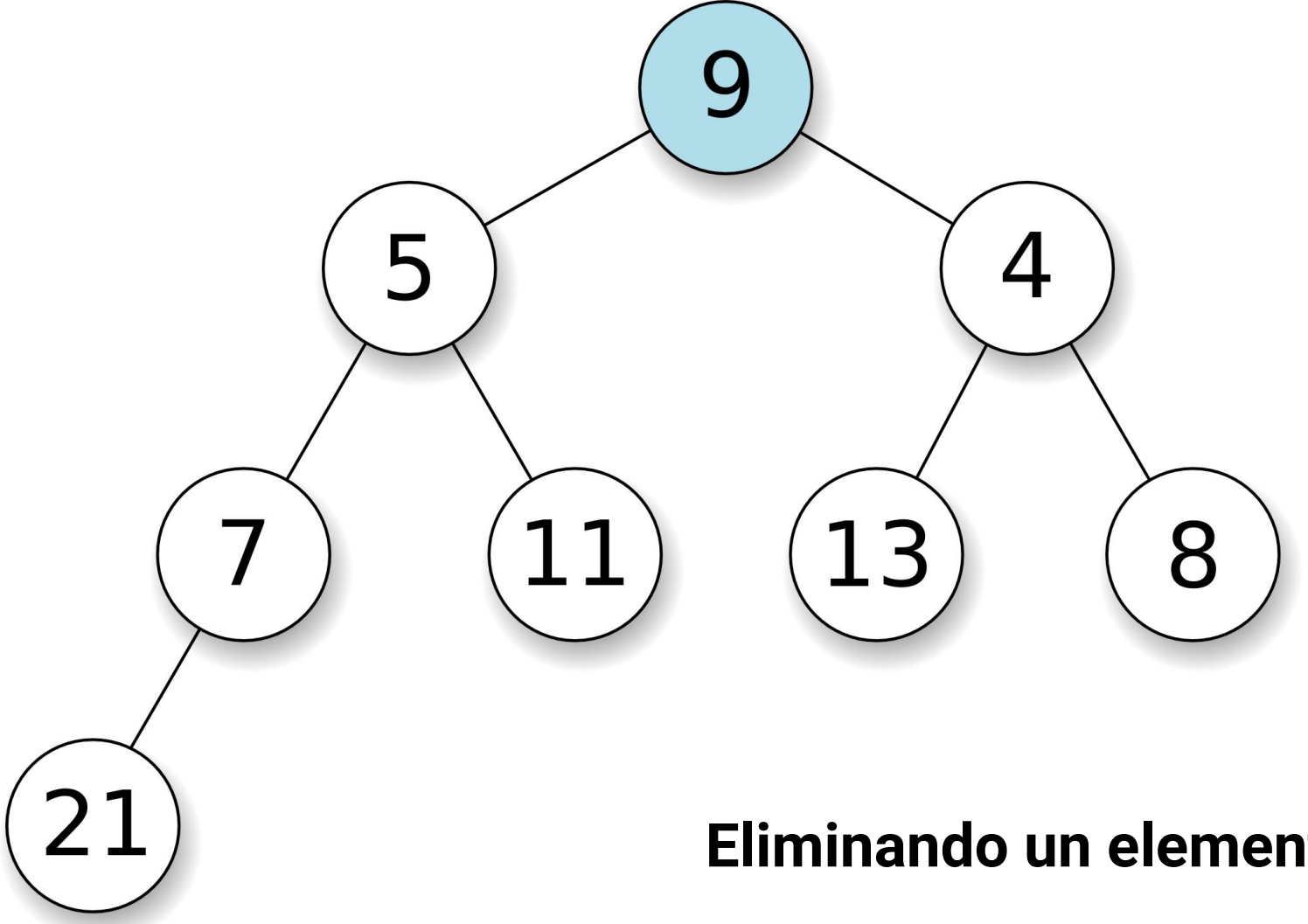
Eliminando un elemento



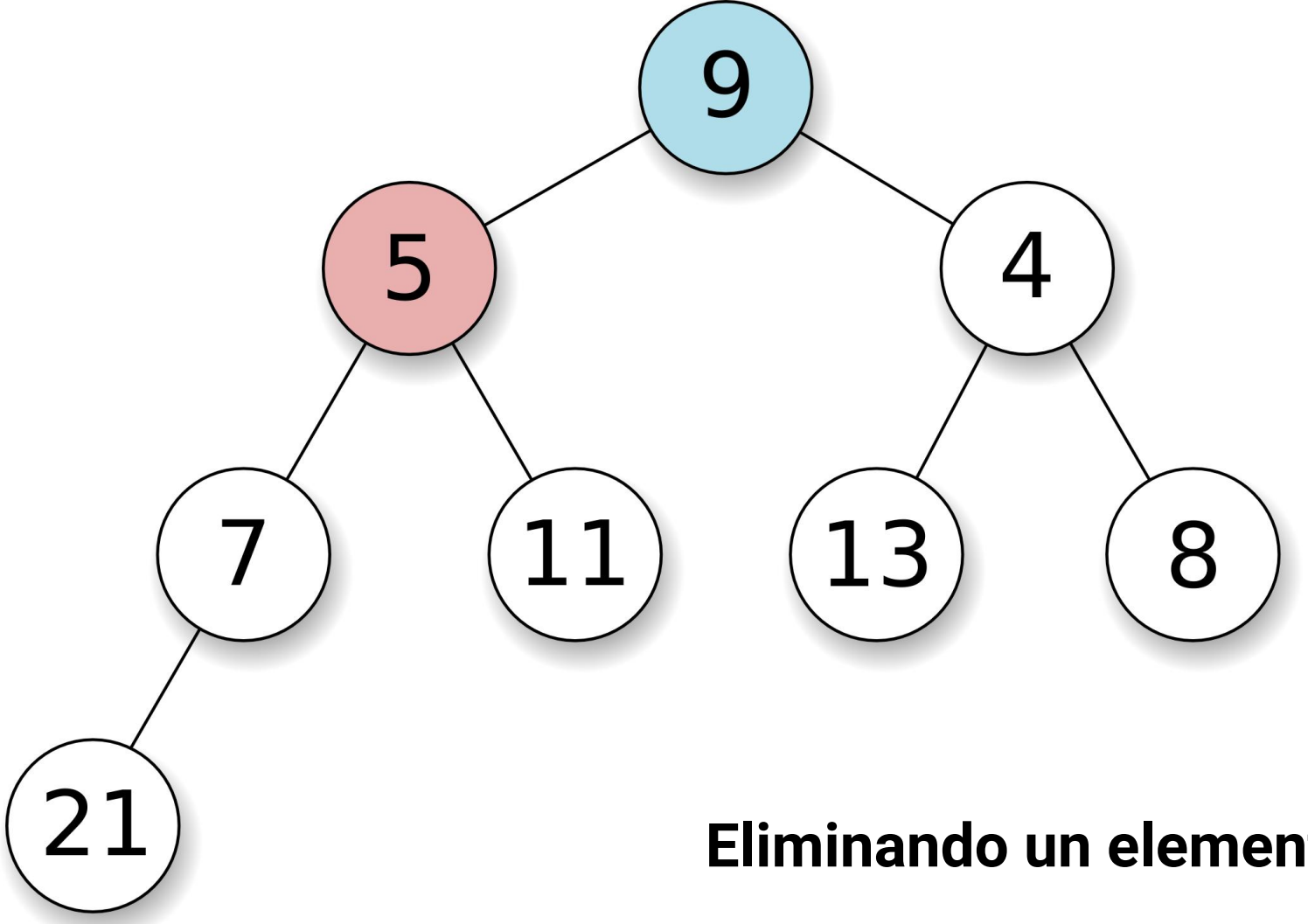
Eliminando un elemento



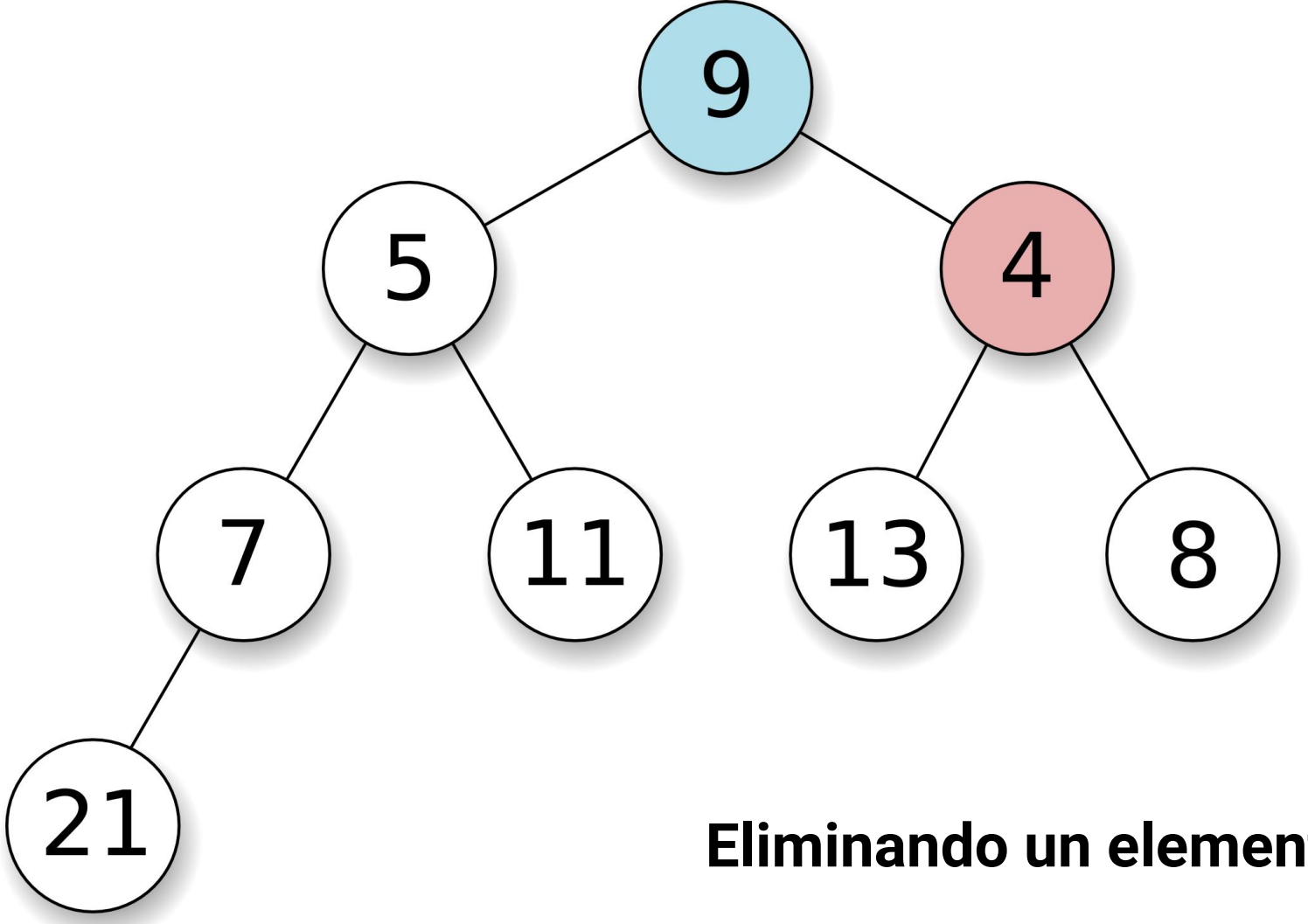
Eliminando un elemento



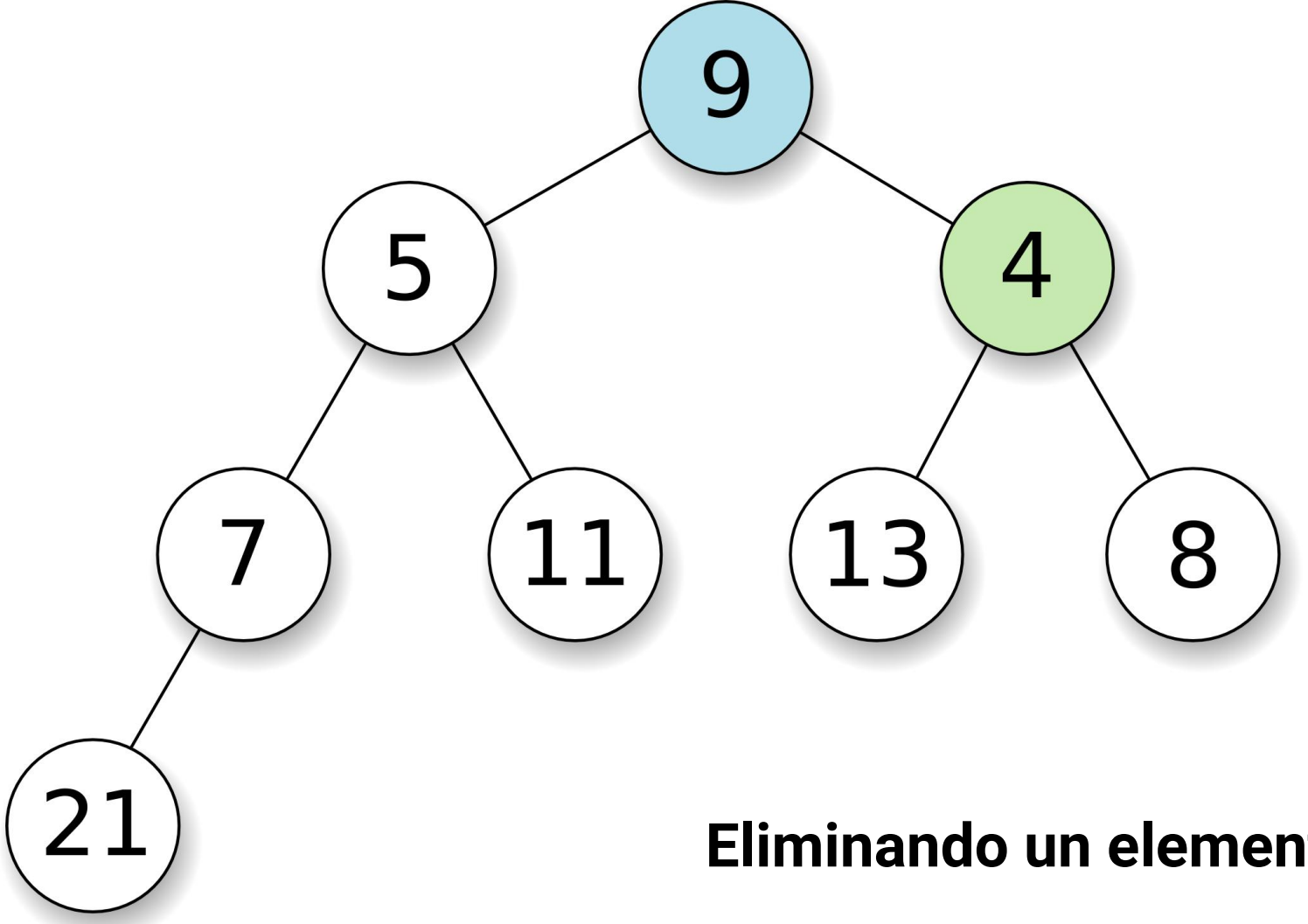
Eliminando un elemento



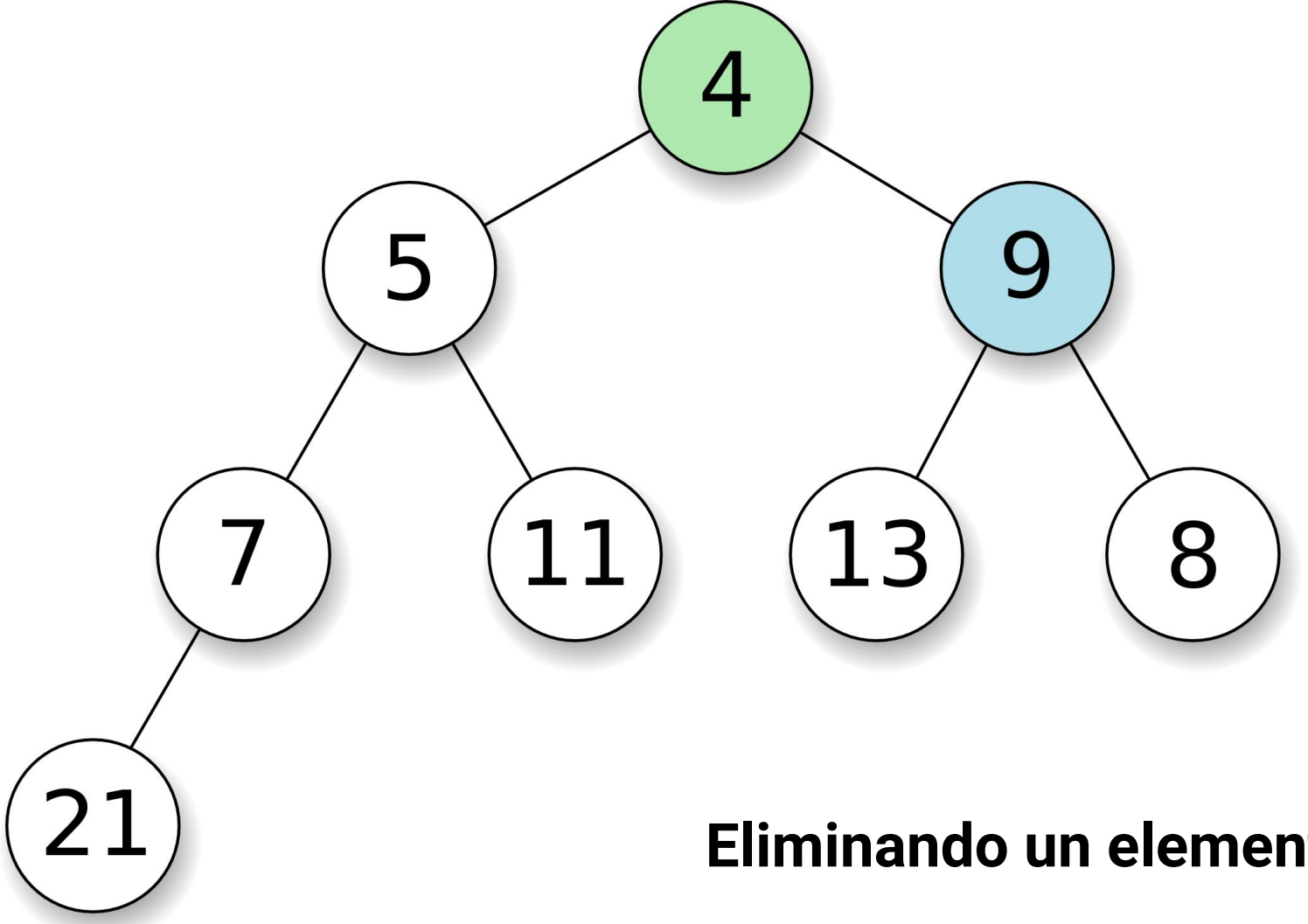
Eliminando un elemento



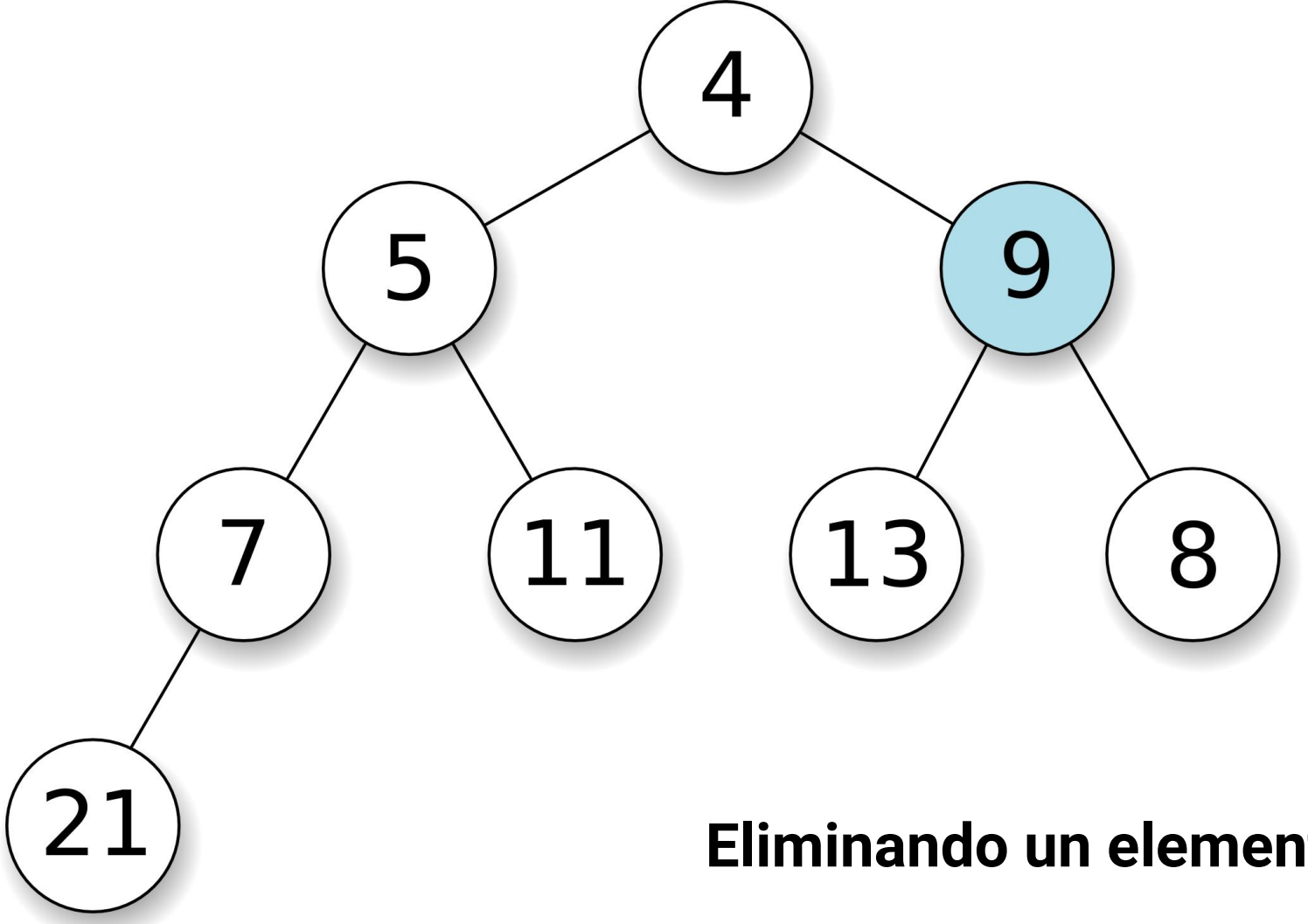
Eliminando un elemento



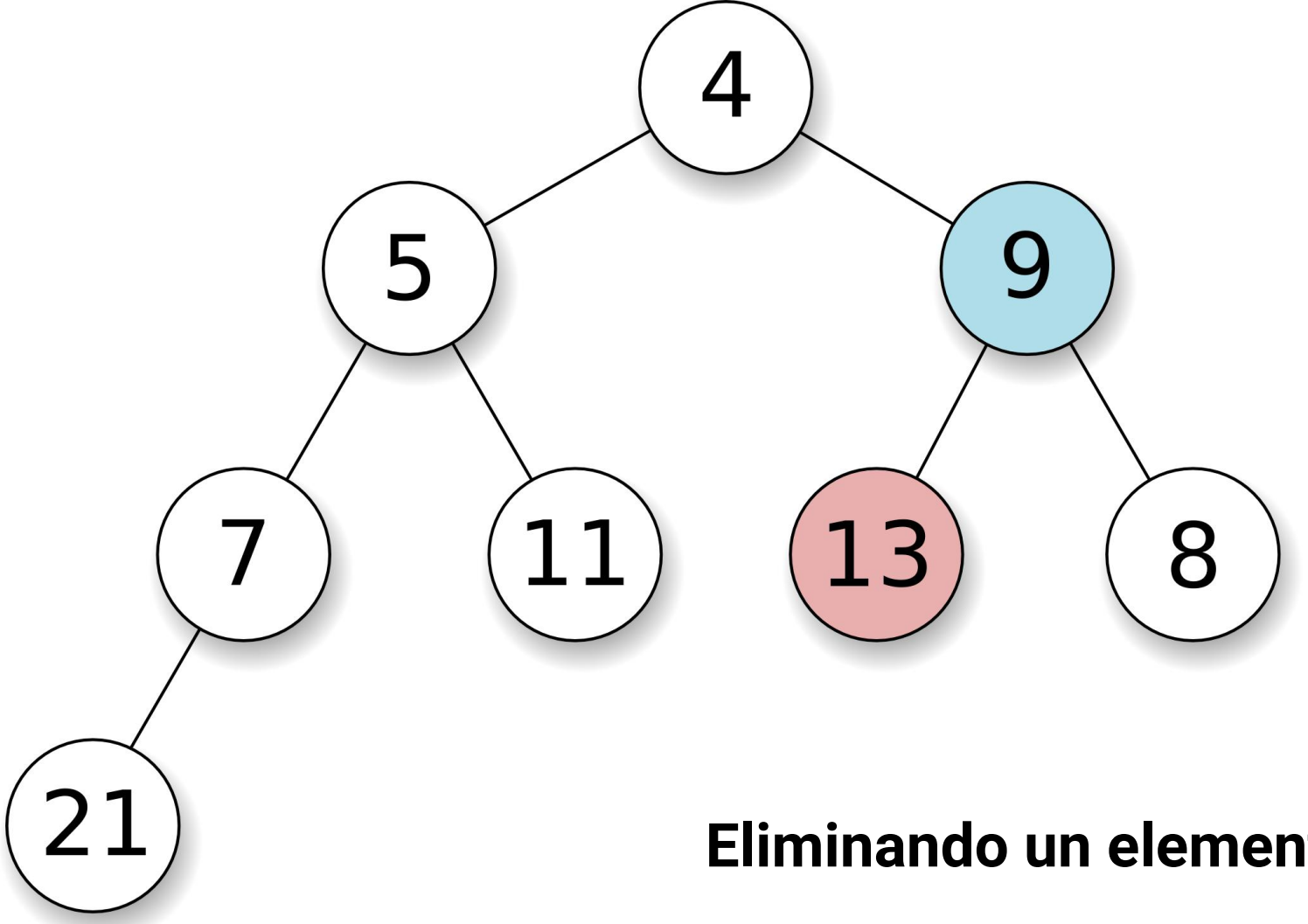
Eliminando un elemento



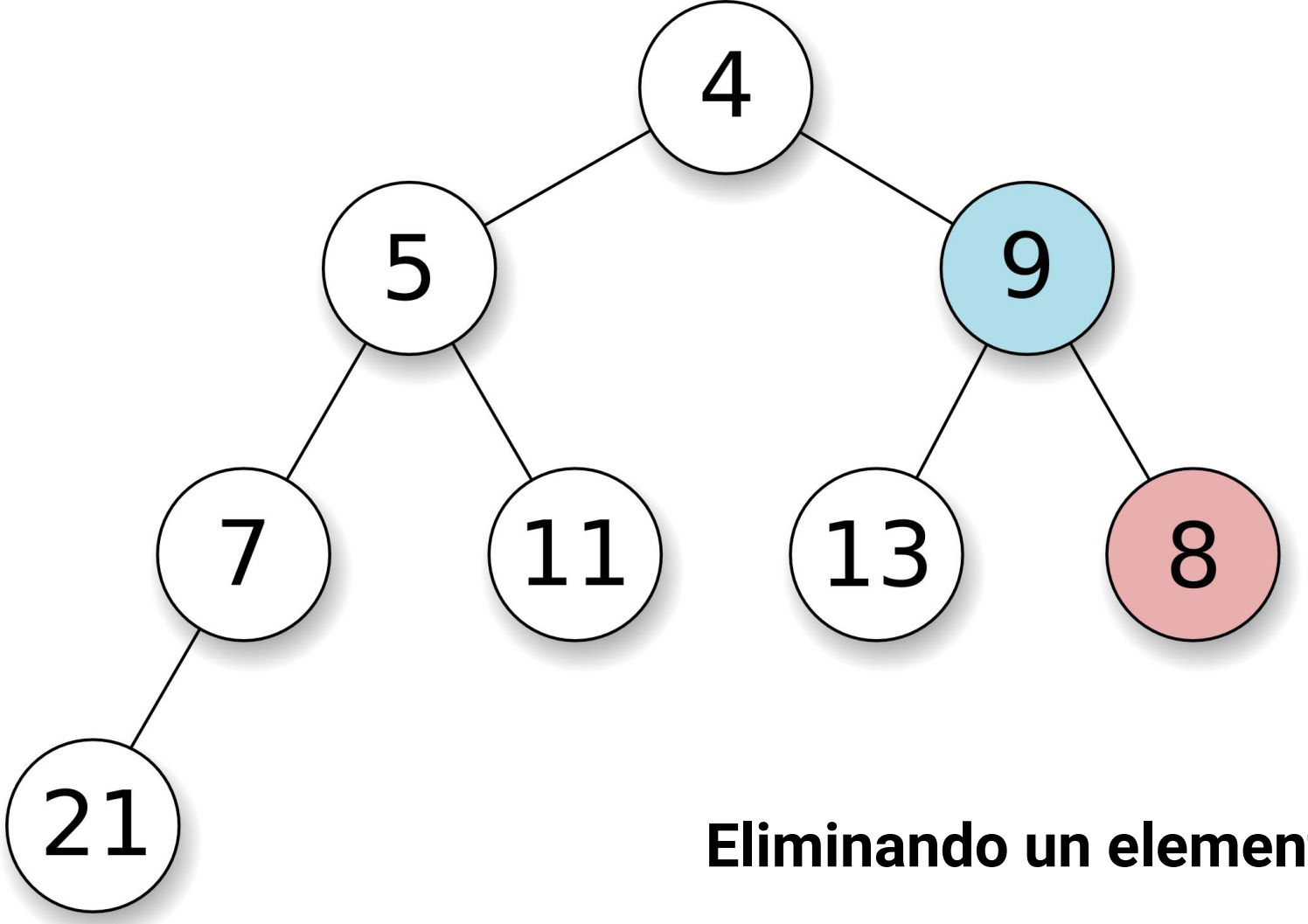
Eliminando un elemento



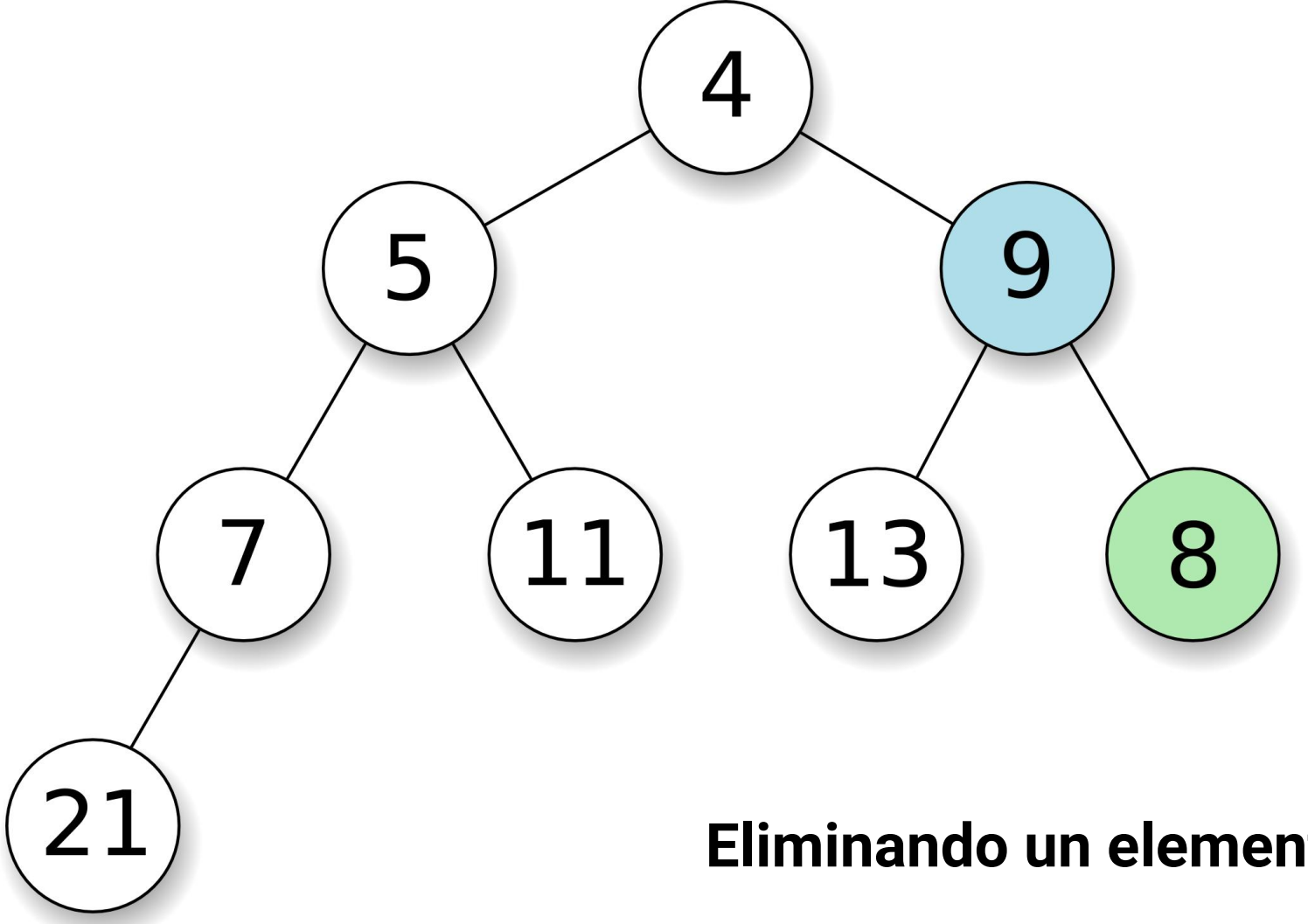
Eliminando un elemento



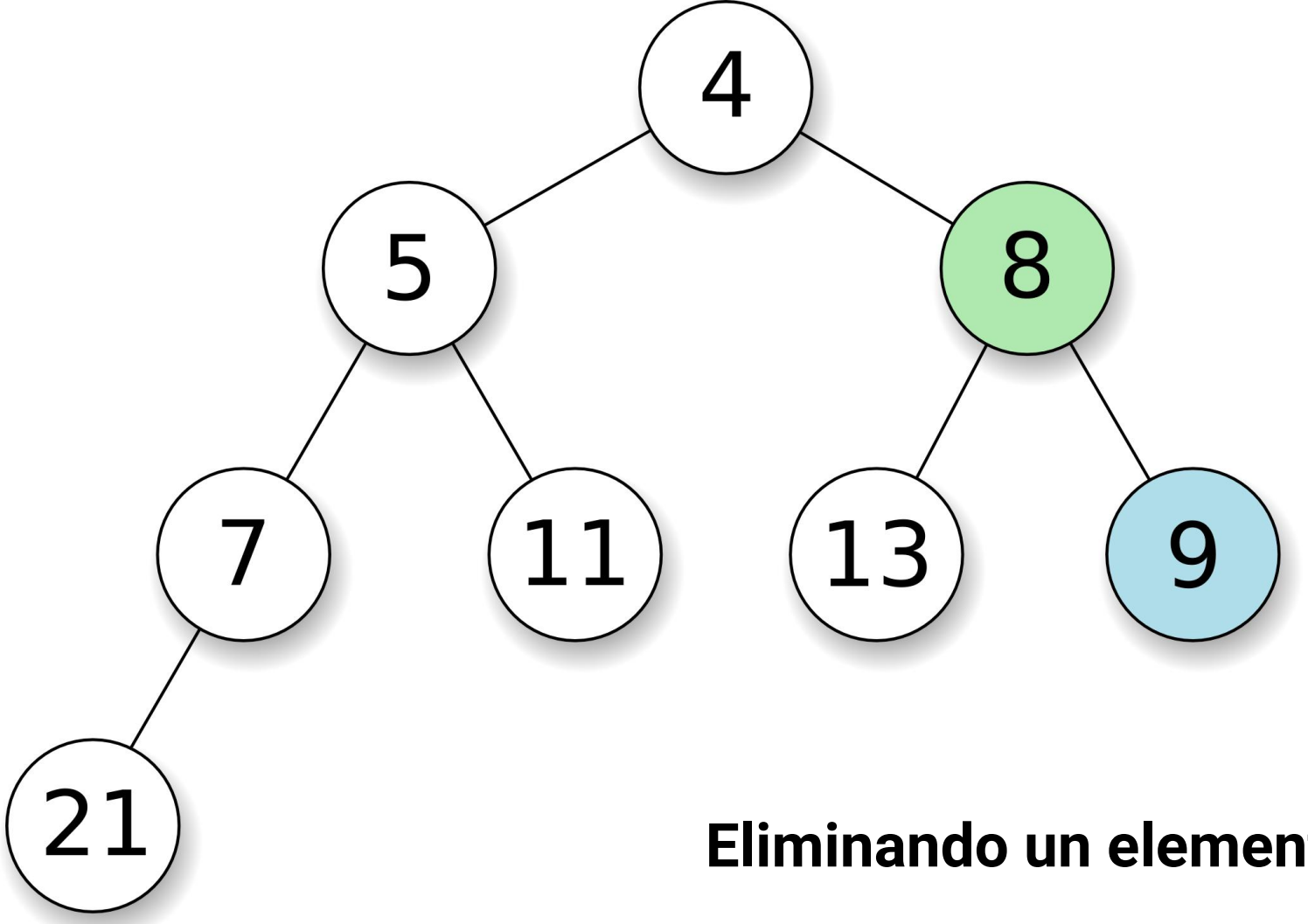
Eliminando un elemento



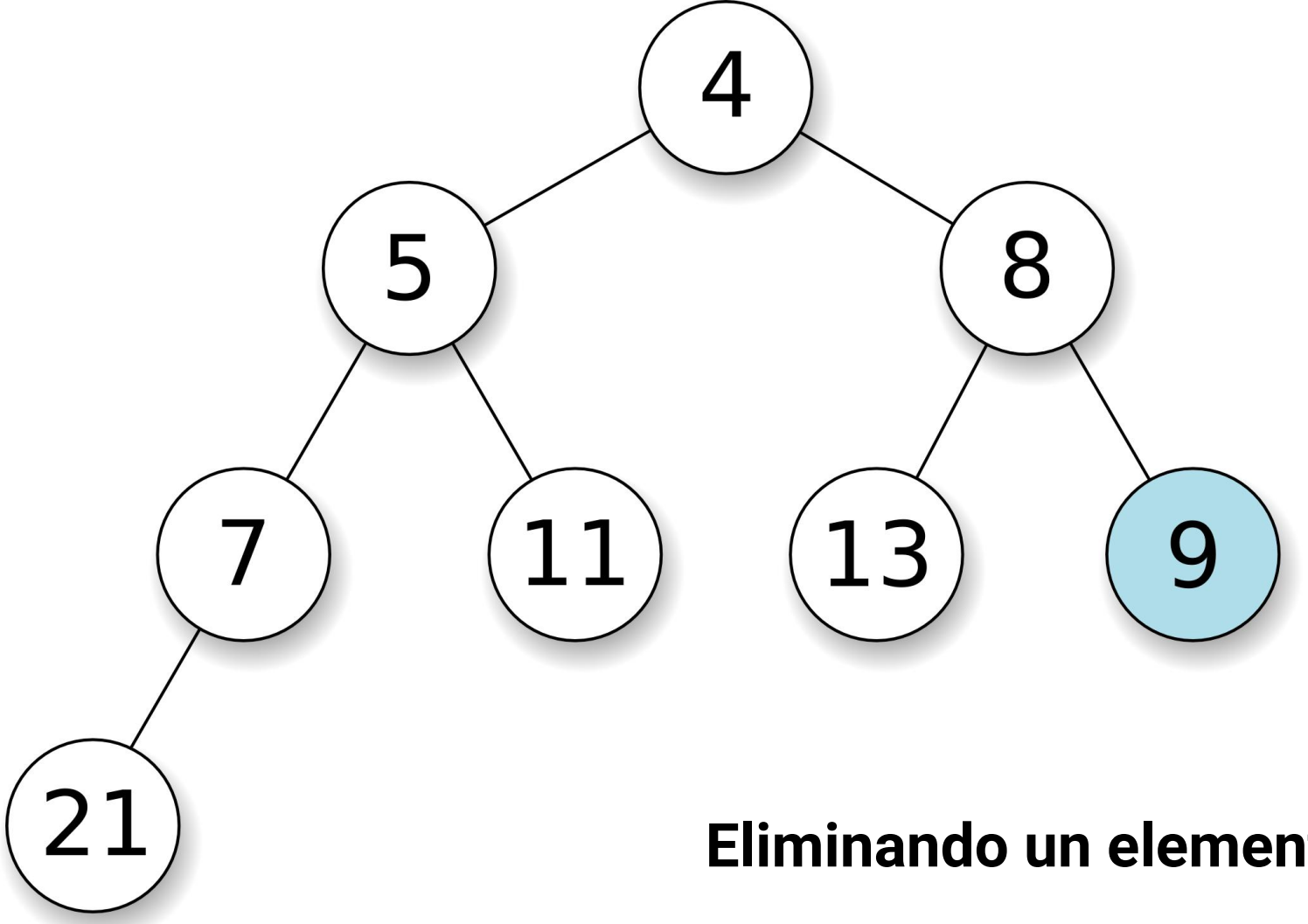
Eliminando un elemento



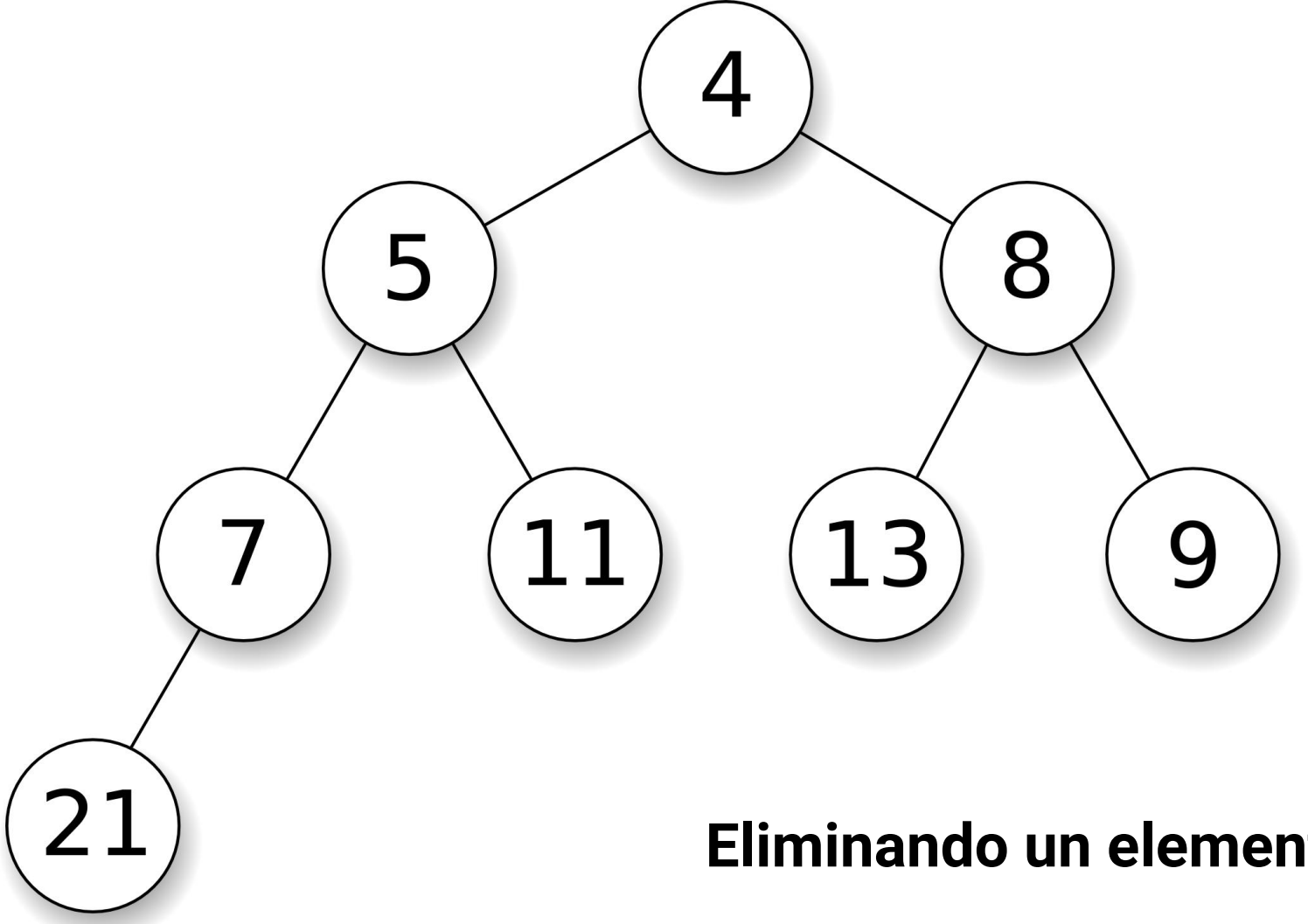
Eliminando un elemento



Eliminando un elemento



Eliminando un elemento



Eliminando un elemento


O(log n)

```
import heapq

h = []
heapq.heappush(h, 4)
heapq.heappush(h, 2)
heapq.heappush(h, 7)

>>> heapq.heappop(h)
2
```

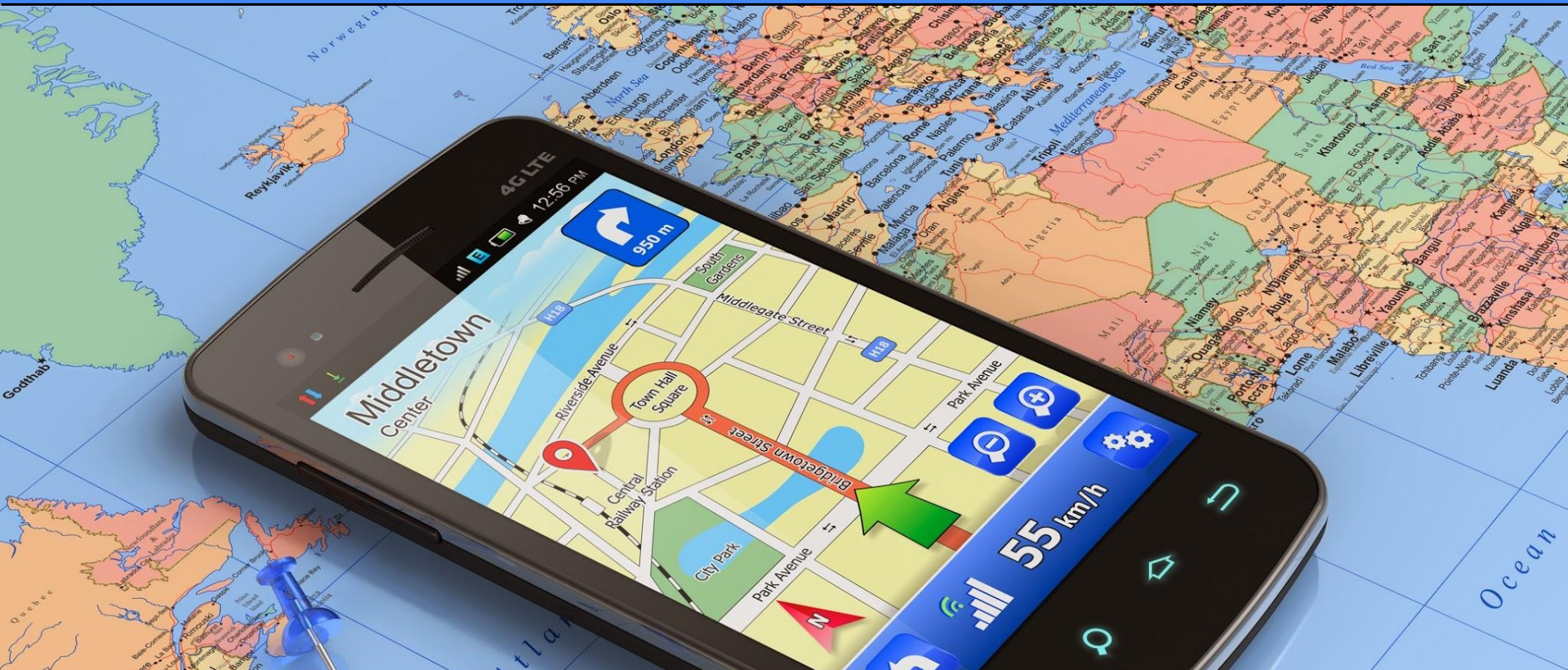
```
import heapq

class Heap(object):
    """A min-heap heap."""

    def __init__(self):
        self._values = []

    def push(self, value):
        """Push the value item onto the heap."""
        heapq.heappush(self._values, value)

    def pop(self):
        """Pop and return the smallest item from the heap."""
        return heapq.heappop(self._values)
```

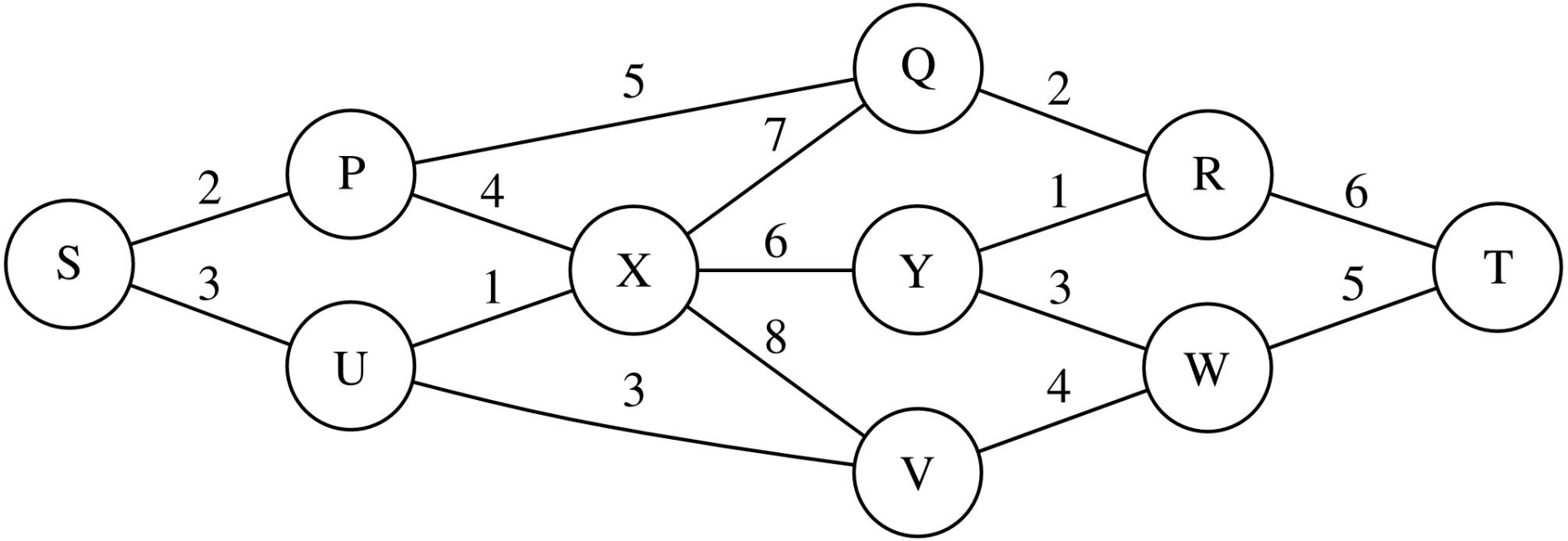
```
heap = Heap()

heap.push(4)
heap.push(2)
heap.push(9)

>>> heap.pop()
2
>>> heap.pop()
4
```

```
import collections

Distancia = collections.namedtuple('Distancia', 'km destino')

heap = Heap()
heap.push(Distancia(349, 'Barcelona'))
heap.push(Distancia(302, 'Madrid'))
heap.push(Distancia(959, 'Santiago de Compostela'))

>>> heap.pop()
Distancia(km=302, destino='Madrid')
```

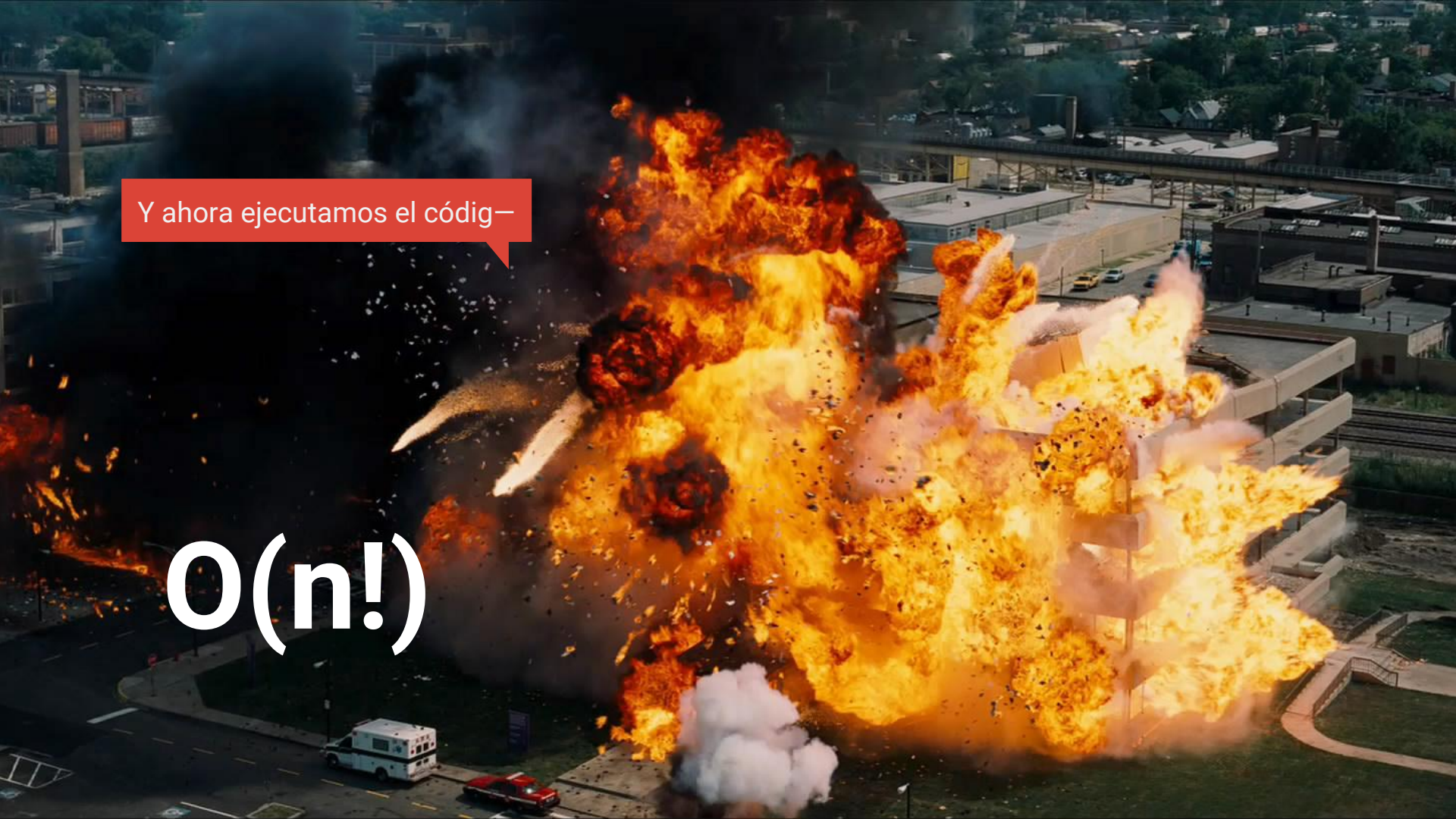
```
dist1 = Distancia(302, 'Madrid')
dist2 = Distancia(349, 'Barcelona')

>>> min(dist1, dist2)
Distancia(km=302, destino='Madrid')
```

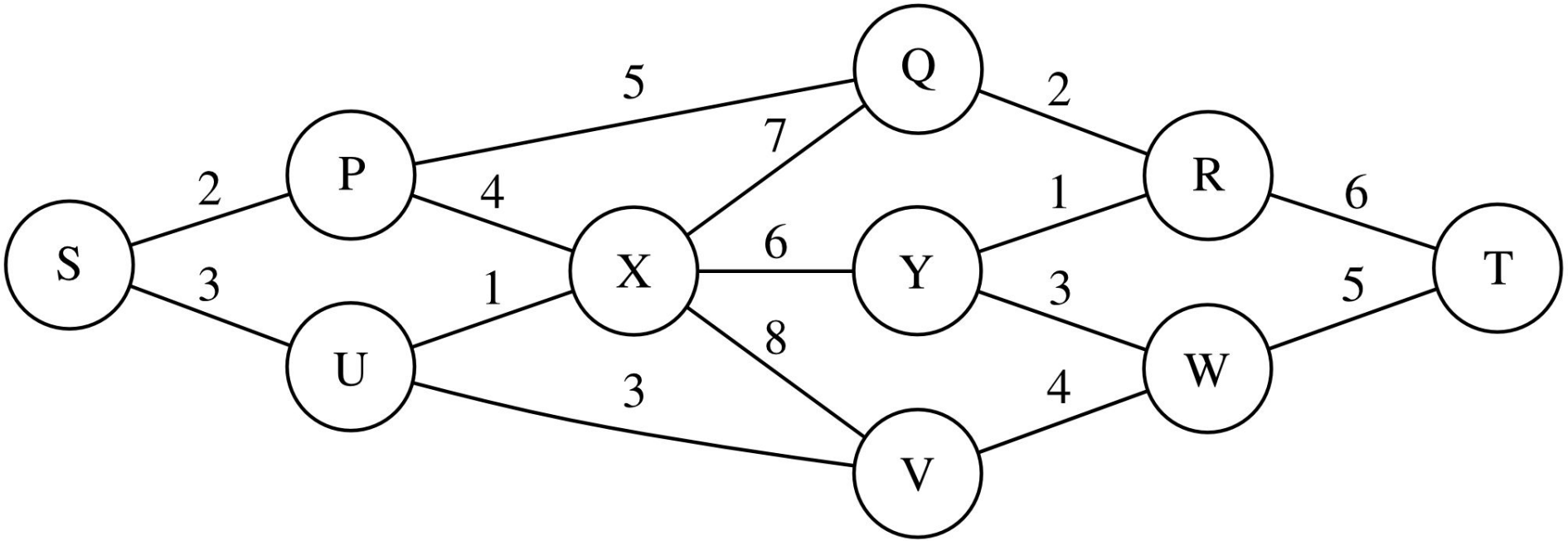
Algoritmo de Dijkstra

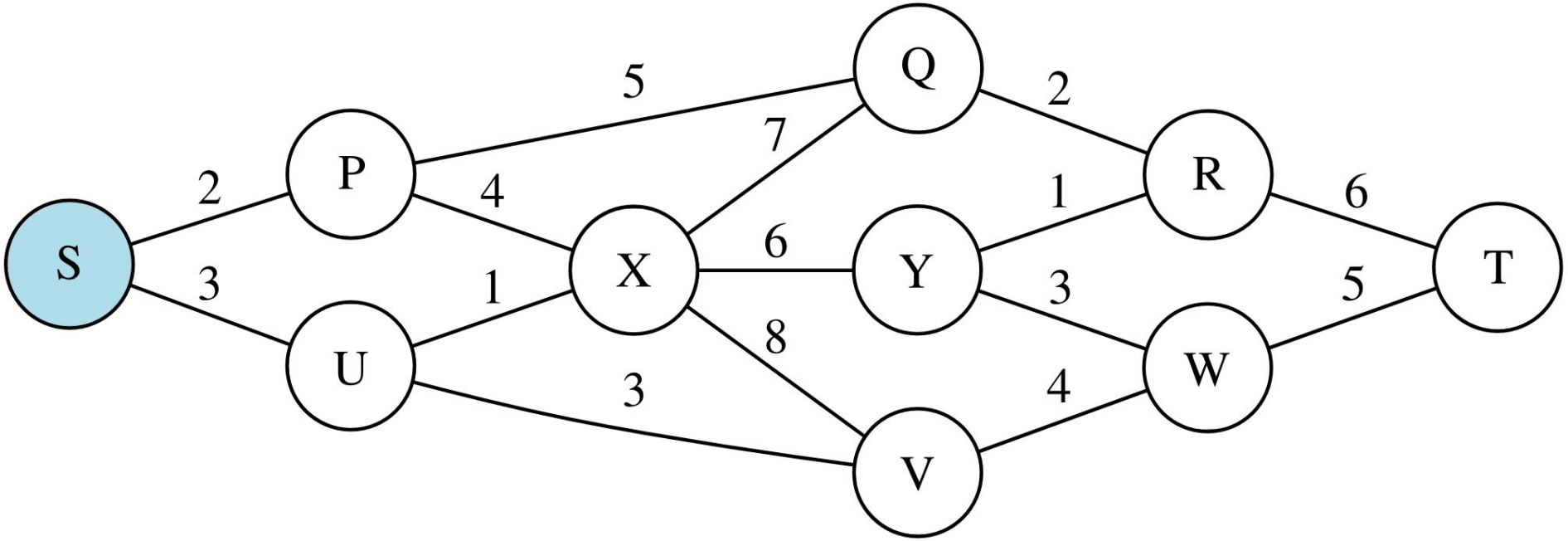


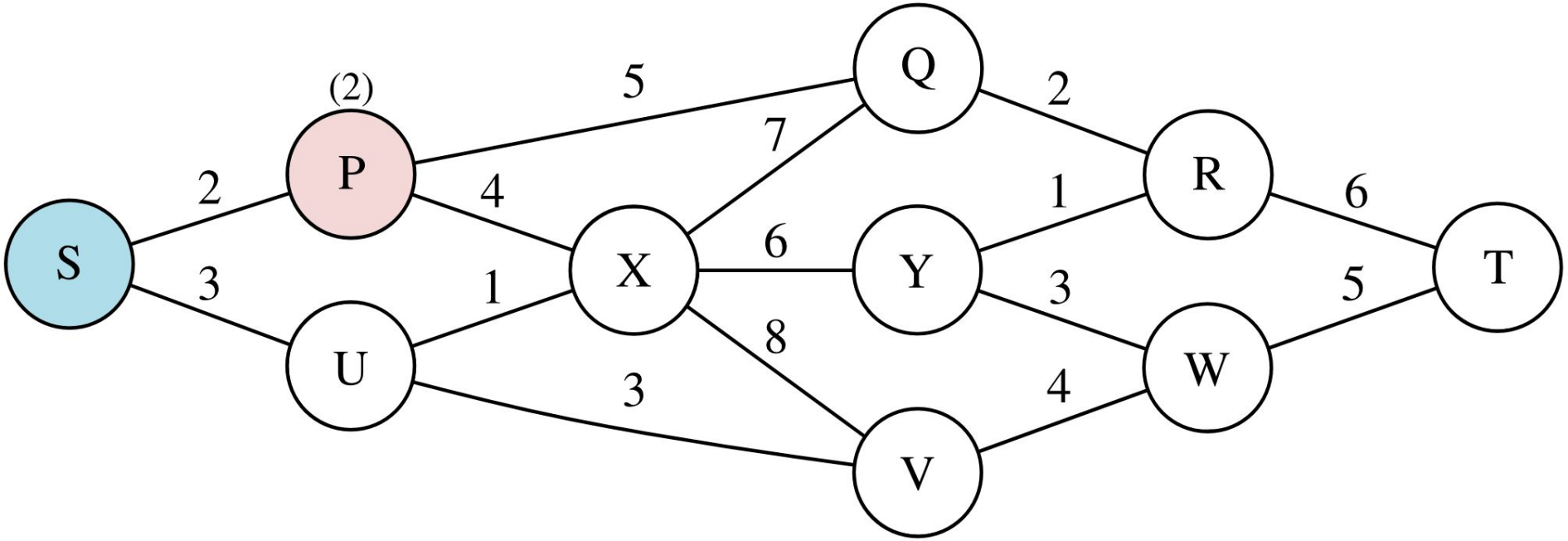


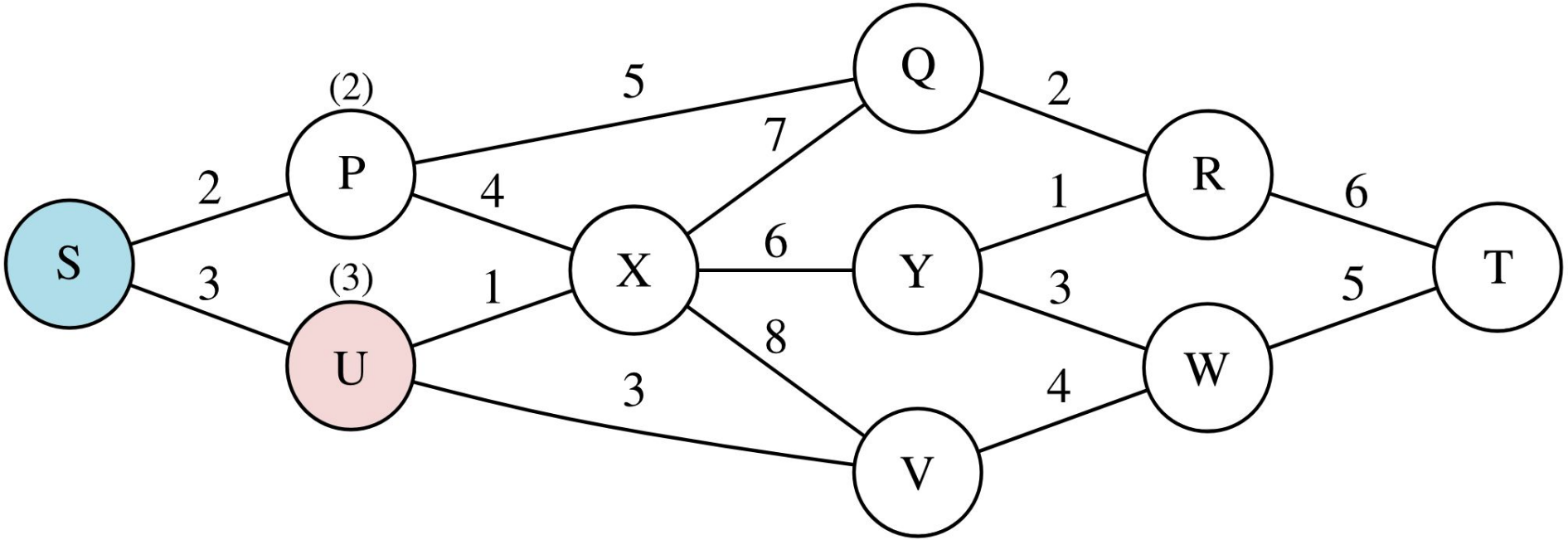
¿Fuerza bruta?

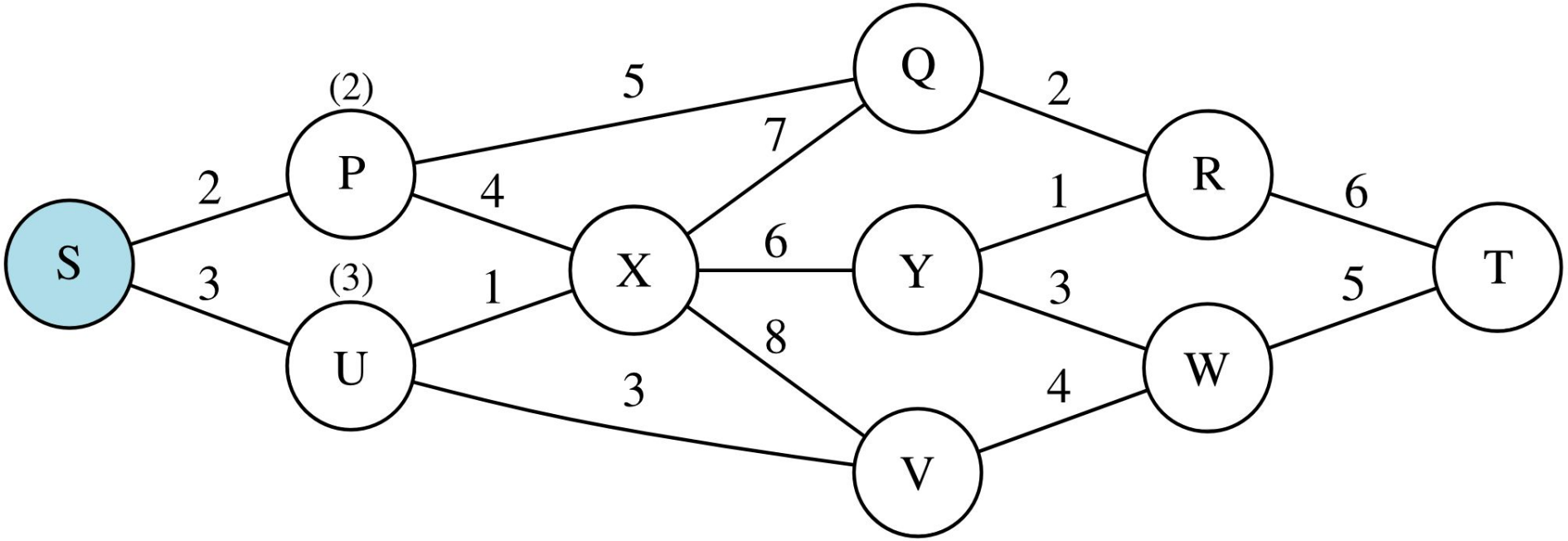

Y ahora ejecutamos el códig—
O(n!)

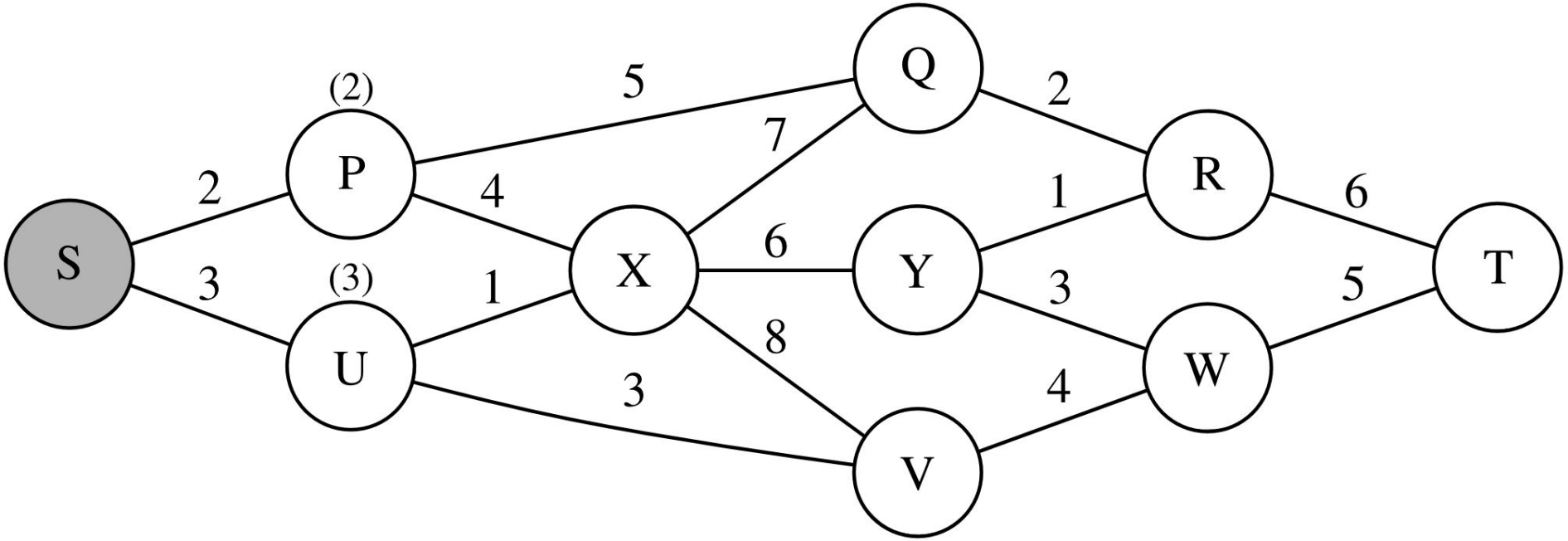

S
P
U
X
Q
Y
V
R
W
T
2
3
5
4
7
1
6
8
2
1
3
6
5
4

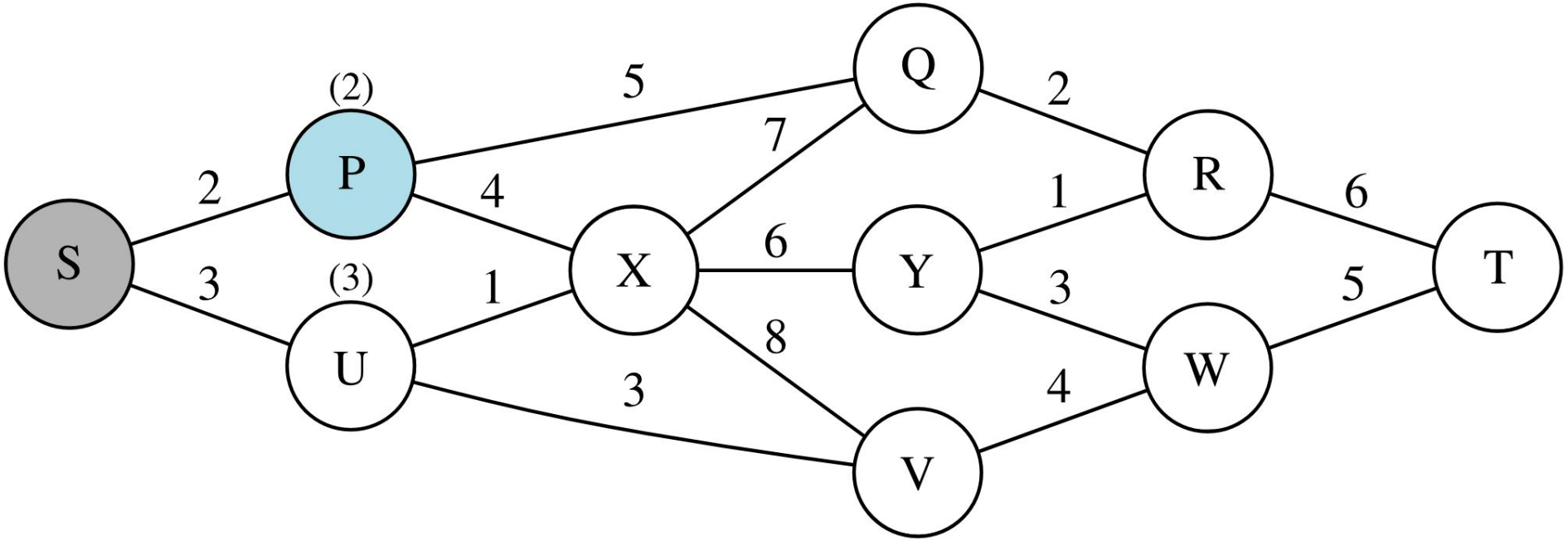

S
P
U
X
Q
Y
V
R
W
T
2
3
5
4
7
1
6
8
2
1
3
6
5
4

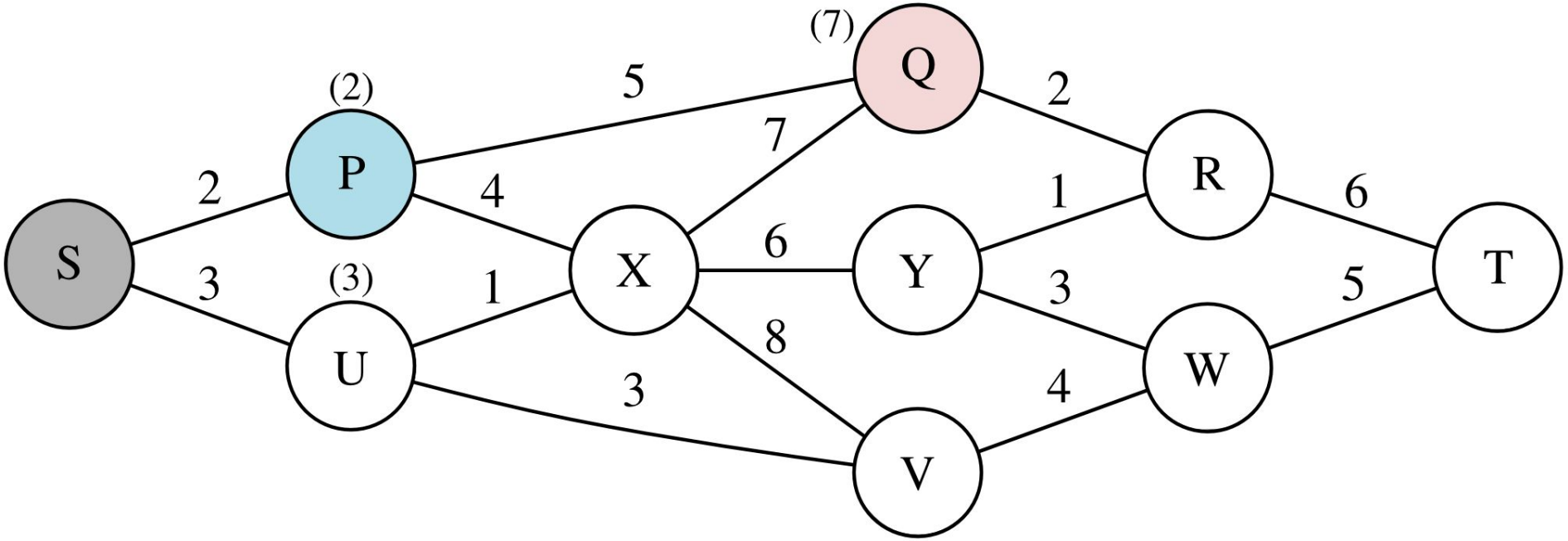

S
P
(2)
U
X
Q
Y
V
R
W
T
2
3
5
4
7
1
6
8
2
1
3
4
6
5

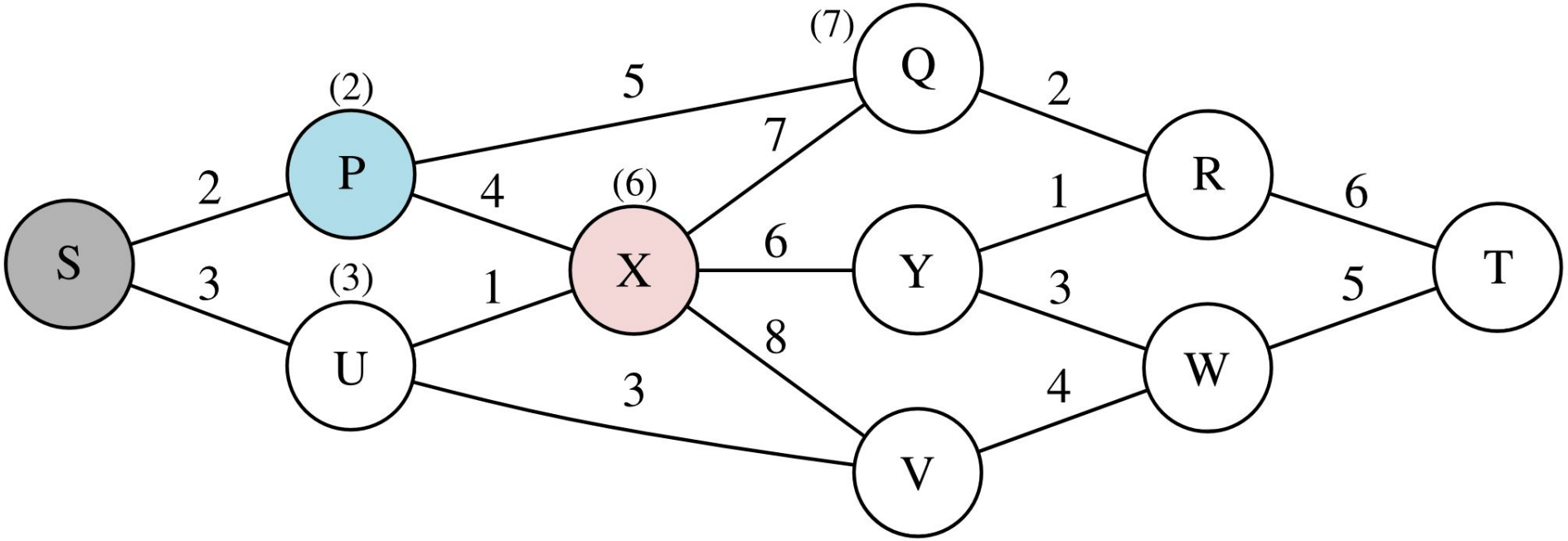

S
P
(2)
U
(3)
X
Q
Y
R
W
V
T
2
3
5
4
1
7
6
8
3
2
1
3
4
6
5

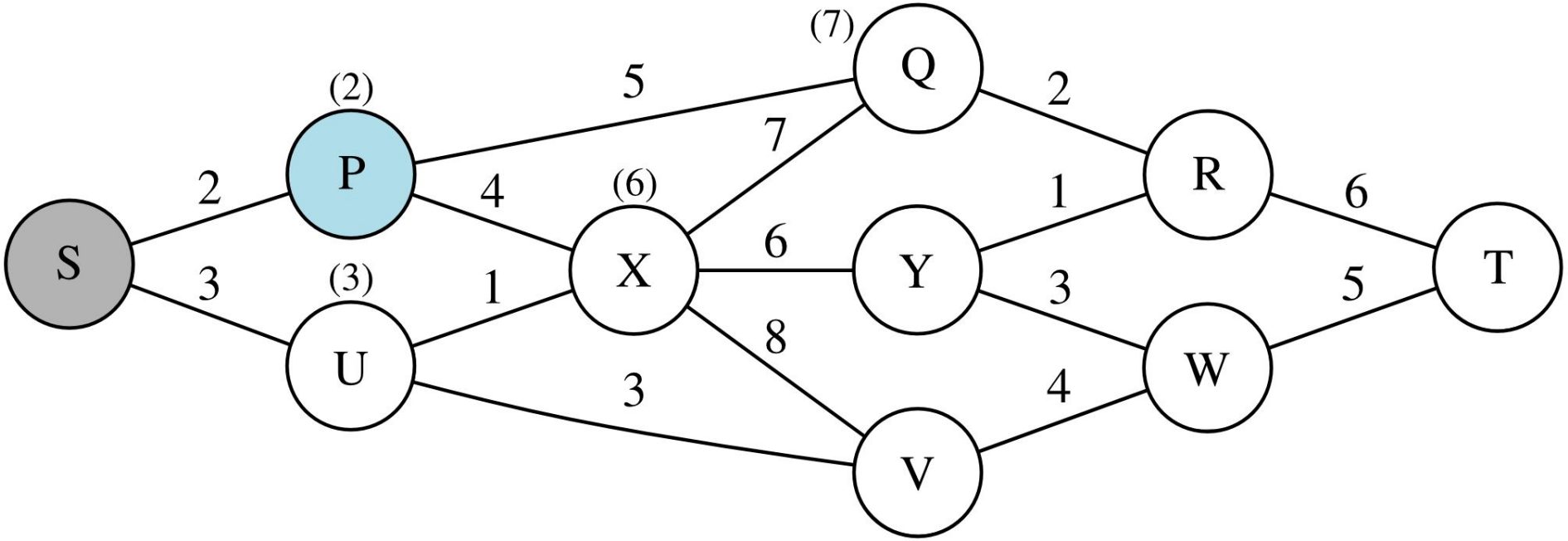

S
P
(2)
U
(3)
X
Q
Y
R
W
V
T
2
3
5
4
1
7
6
8
3
2
1
3
4
6
5

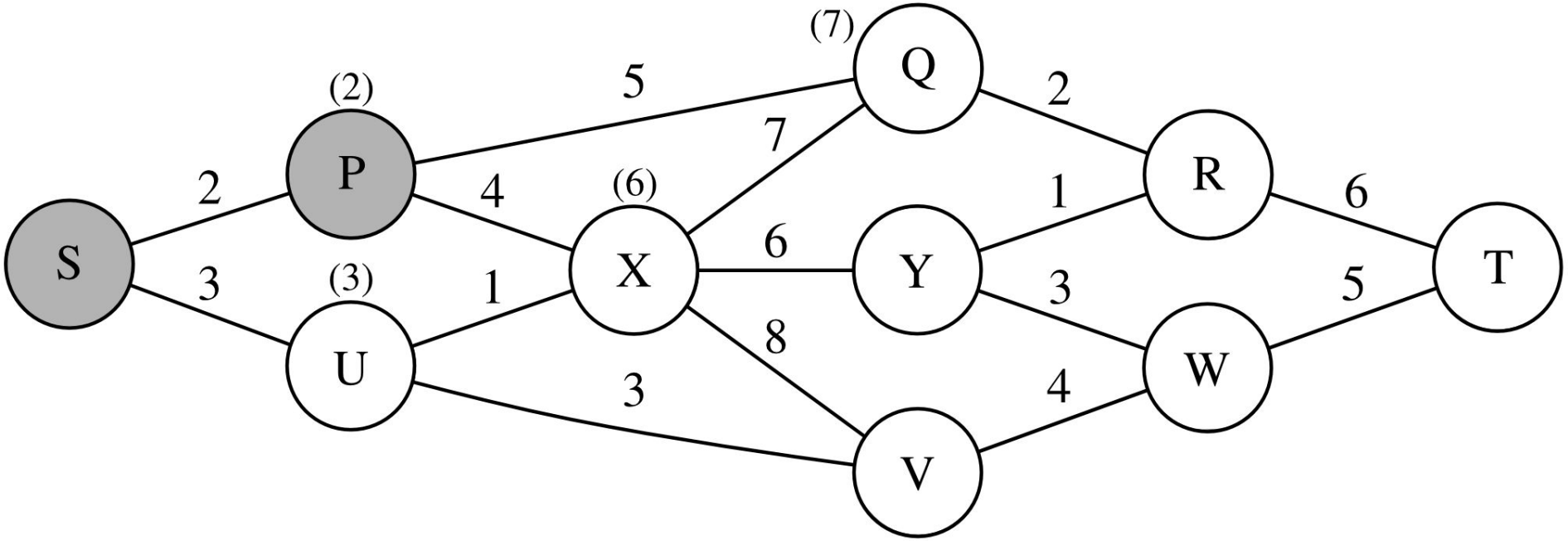

S
P
(2)
U
(3)
X
Q
Y
R
W
V
T
2
3
5
4
1
7
6
8
3
2
1
3
4
6
5

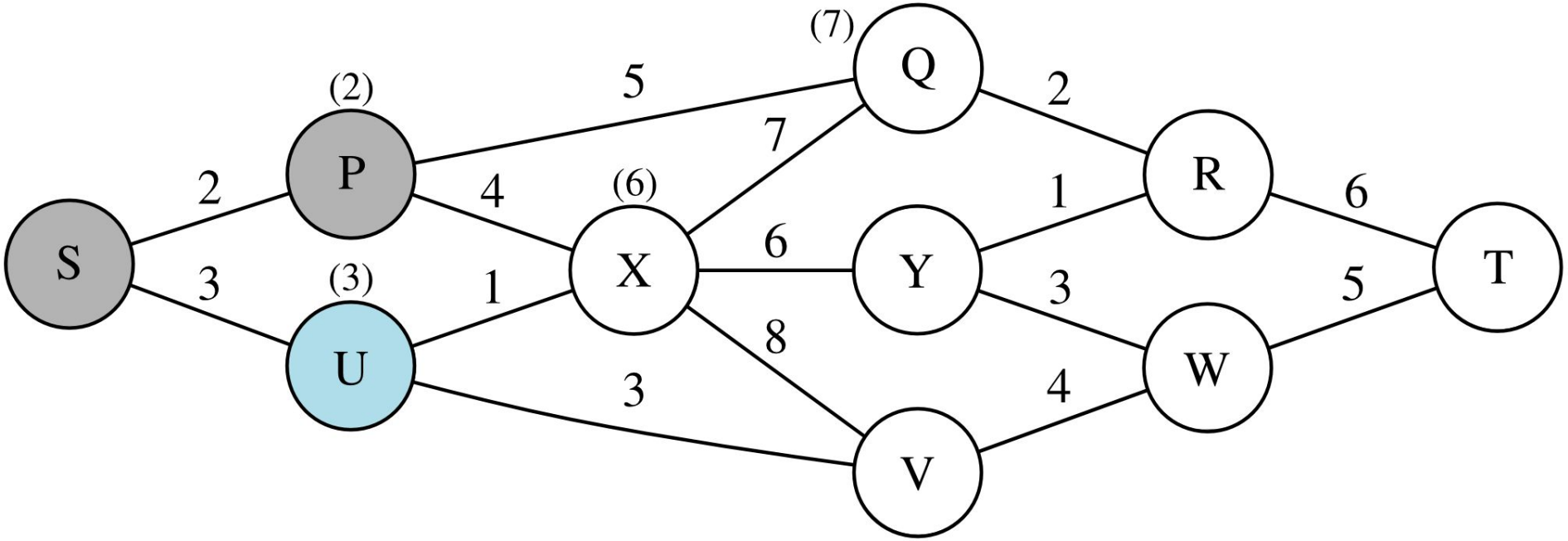

S
P
(2)
U
(3)
X
Q
Y
R
W
V
T
2
3
5
4
1
7
6
8
3
2
1
3
4
6
5

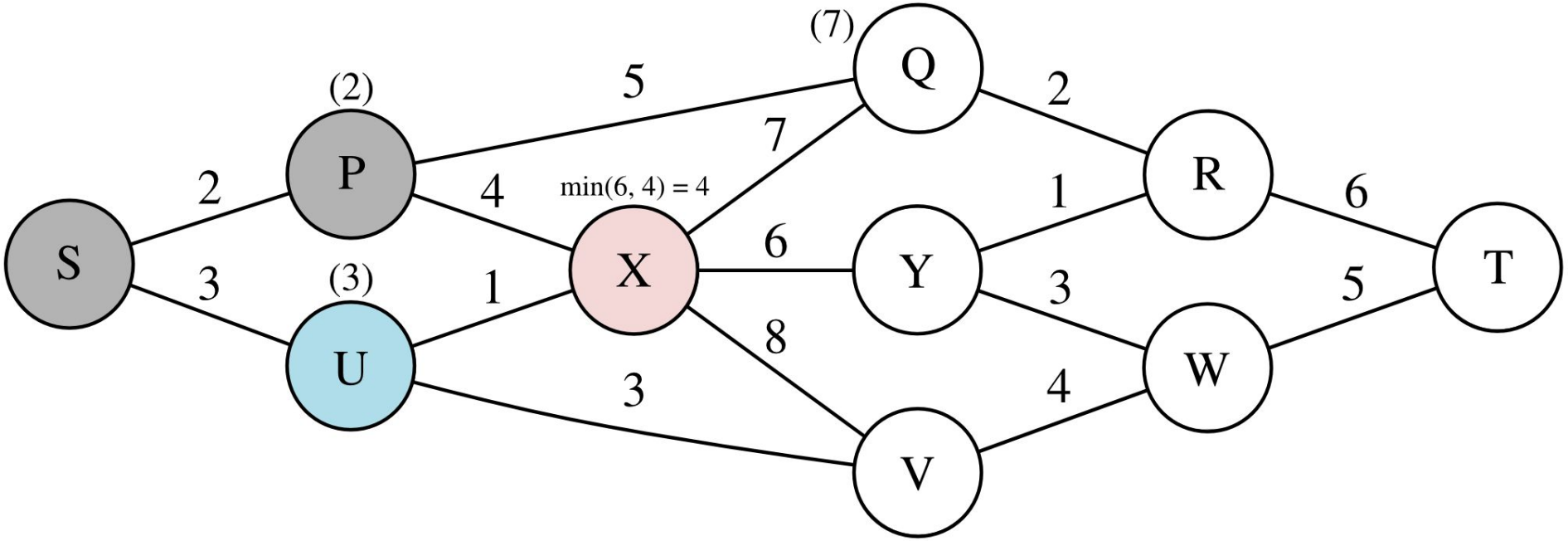

S
P
(2)
U
(3)
X
Q
(7)
Y
R
W
V
T
2
3
4
1
5
7
6
8
3
1
2
3
4
6
5

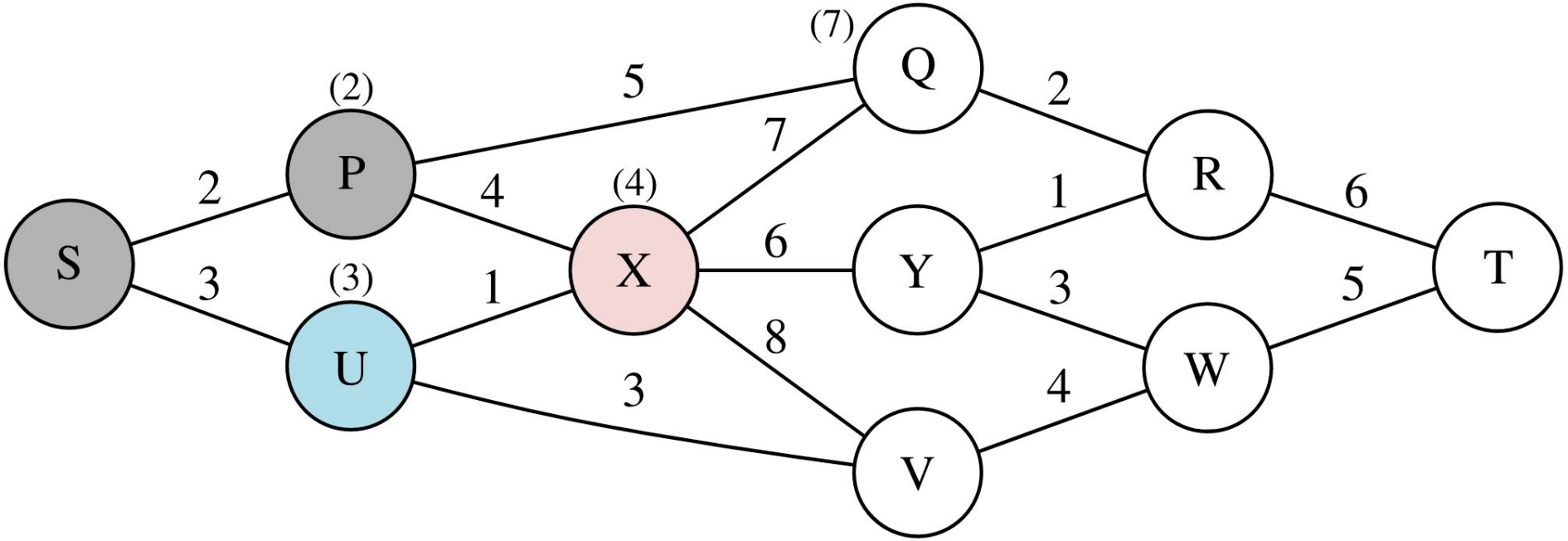

S
P
(2)
U
(3)
X
(6)
Q
(7)
Y
R
W
V
T
2
3
5
4
7
1
6
8
3
2
1
3
4
6
5

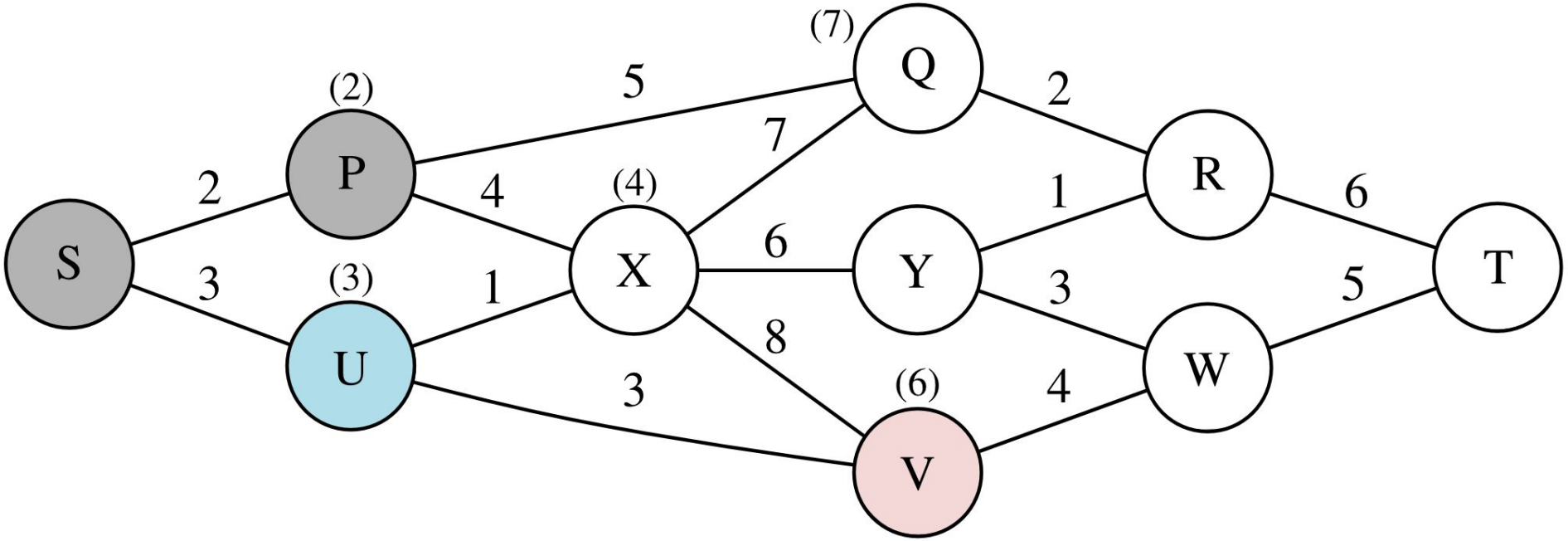

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(6)
(7)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

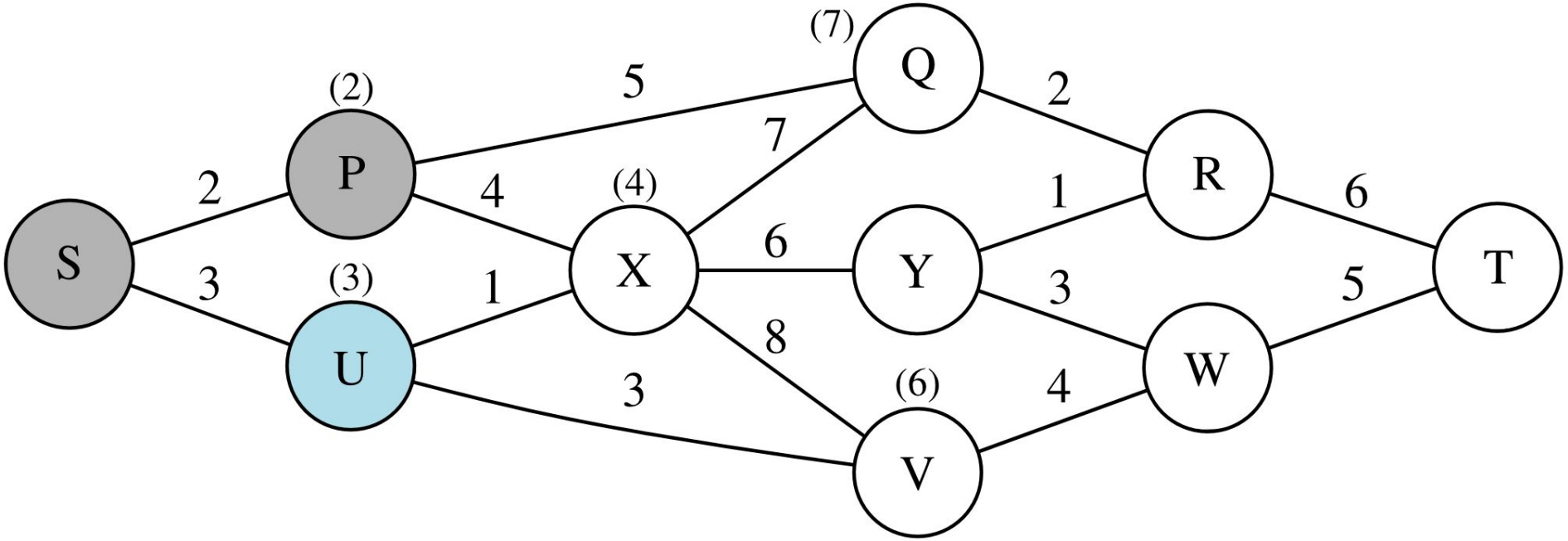

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(6)
(7)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

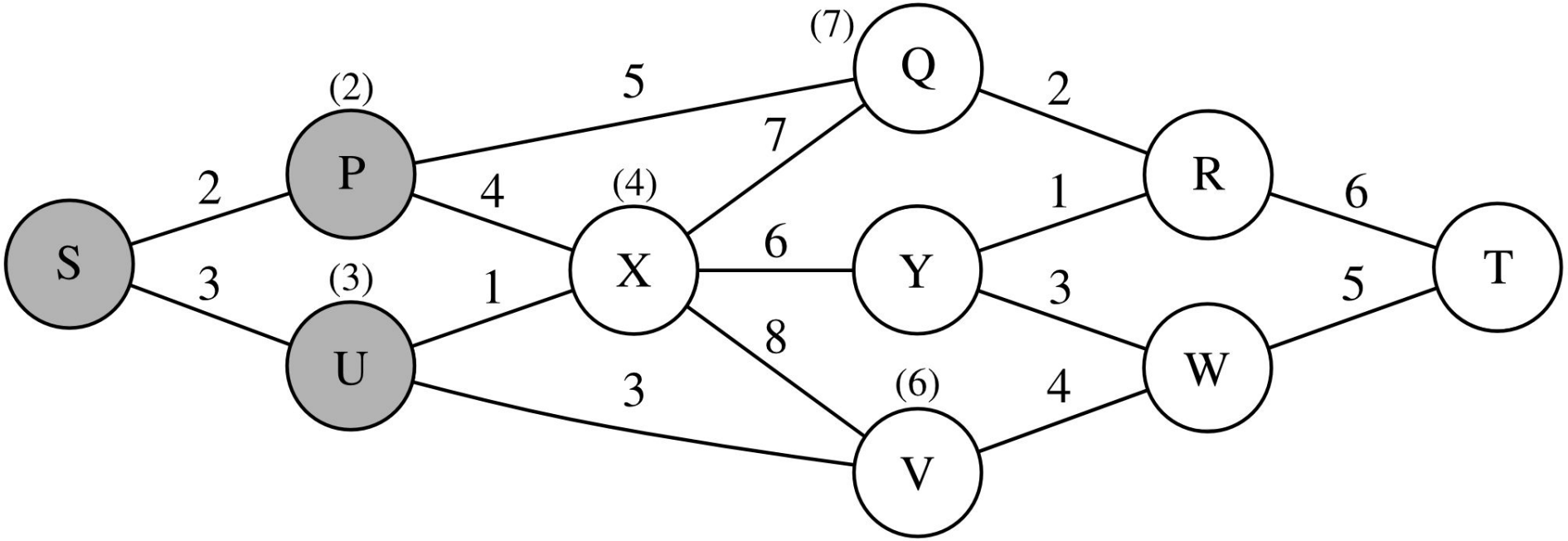

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(6)
(7)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

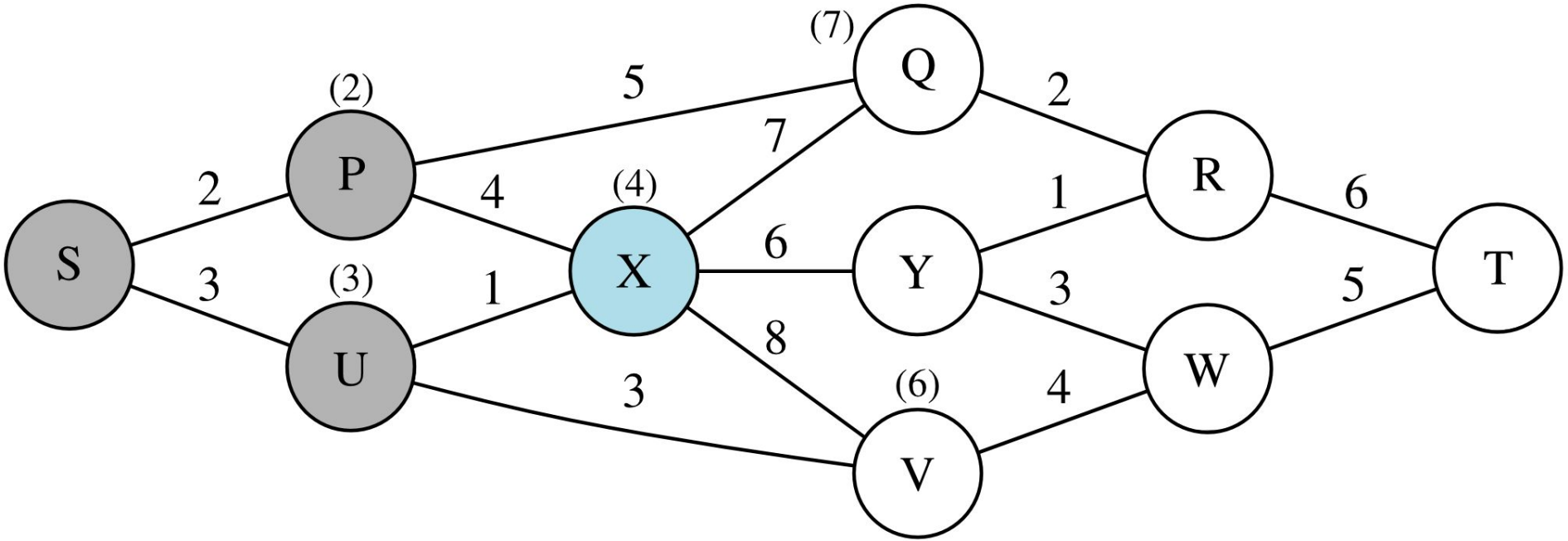

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(7)
2
3
5
4
7
1
6
8
3
2
1
3
4
6
5
min(6, 4) = 4

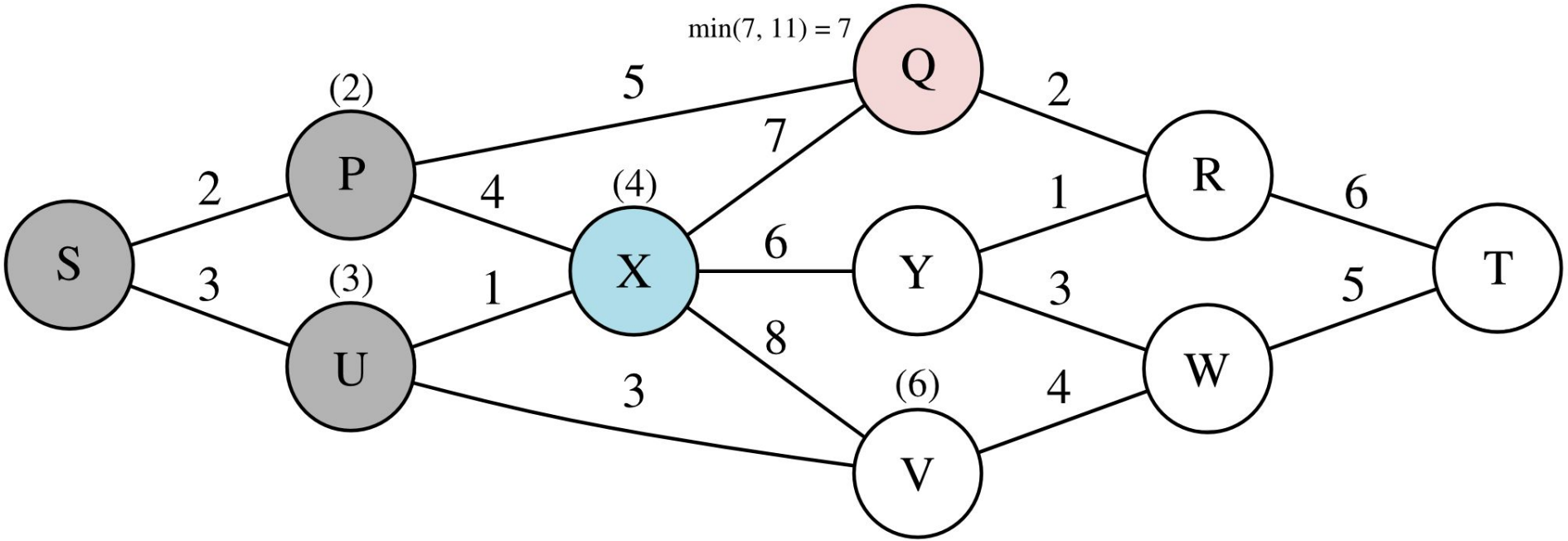

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

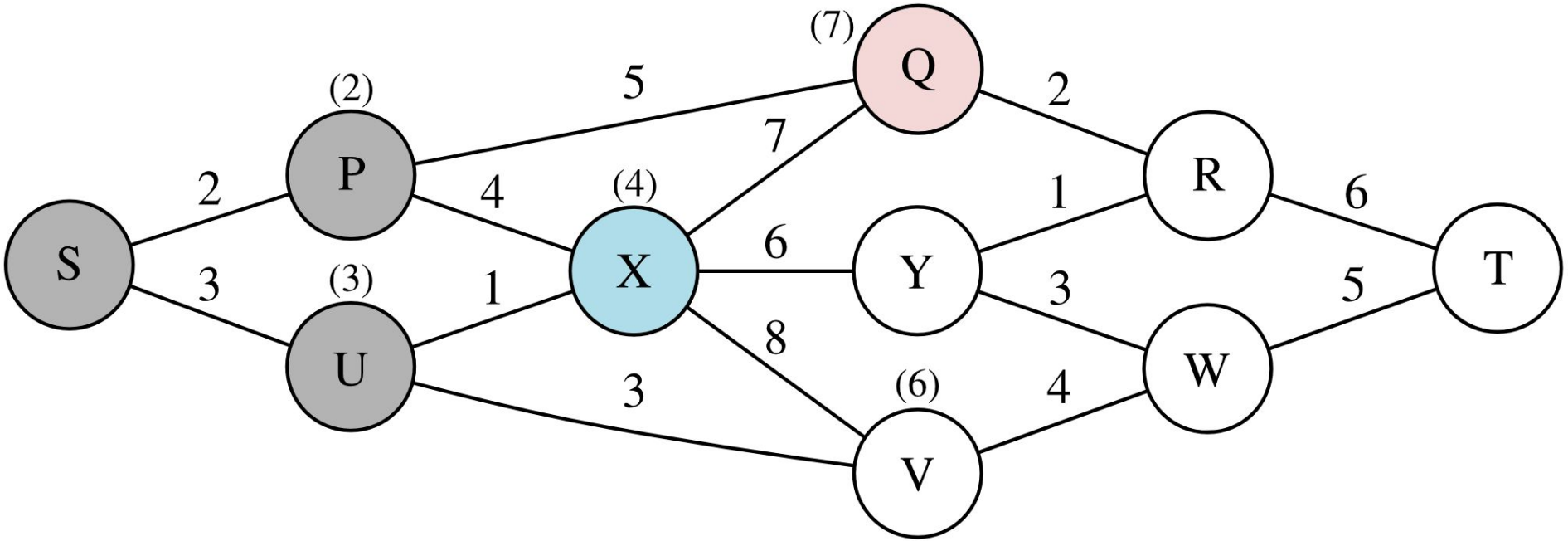

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

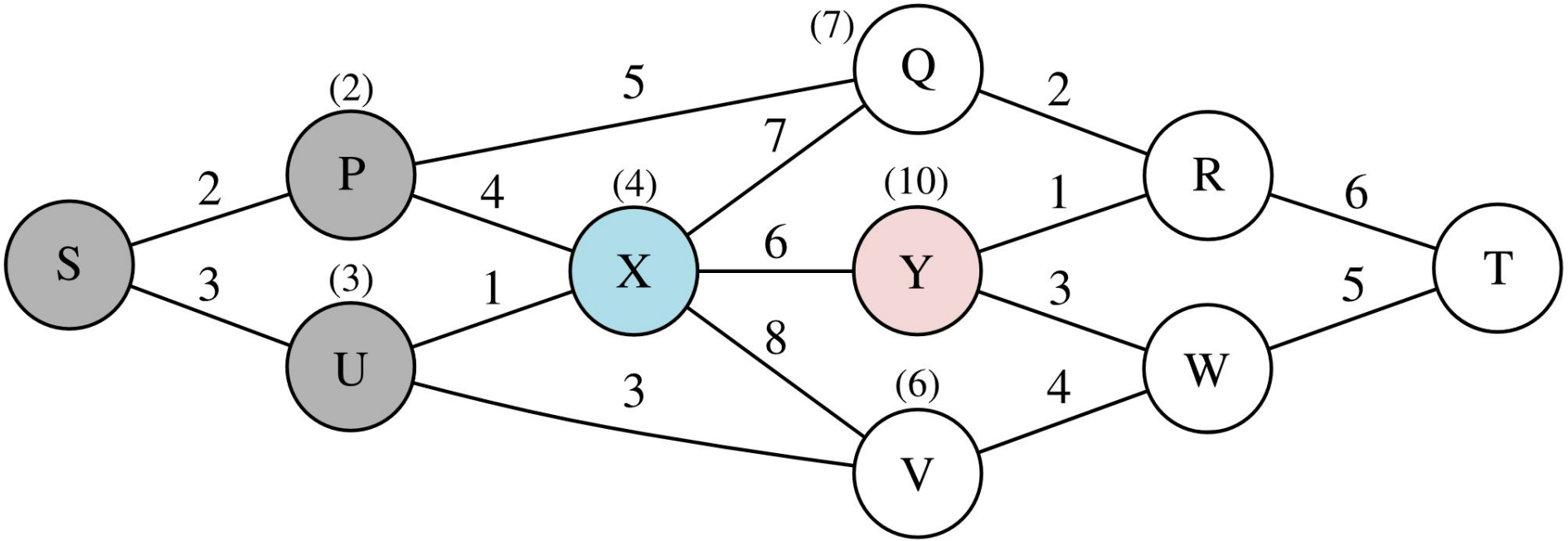

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

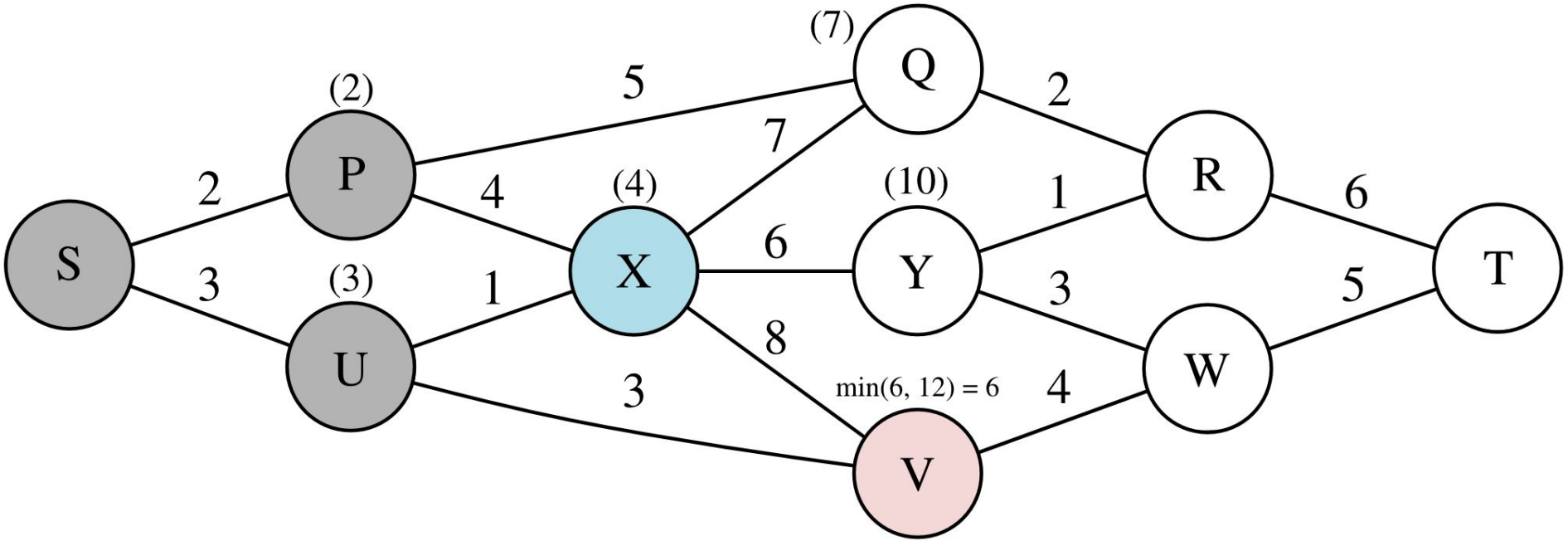

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

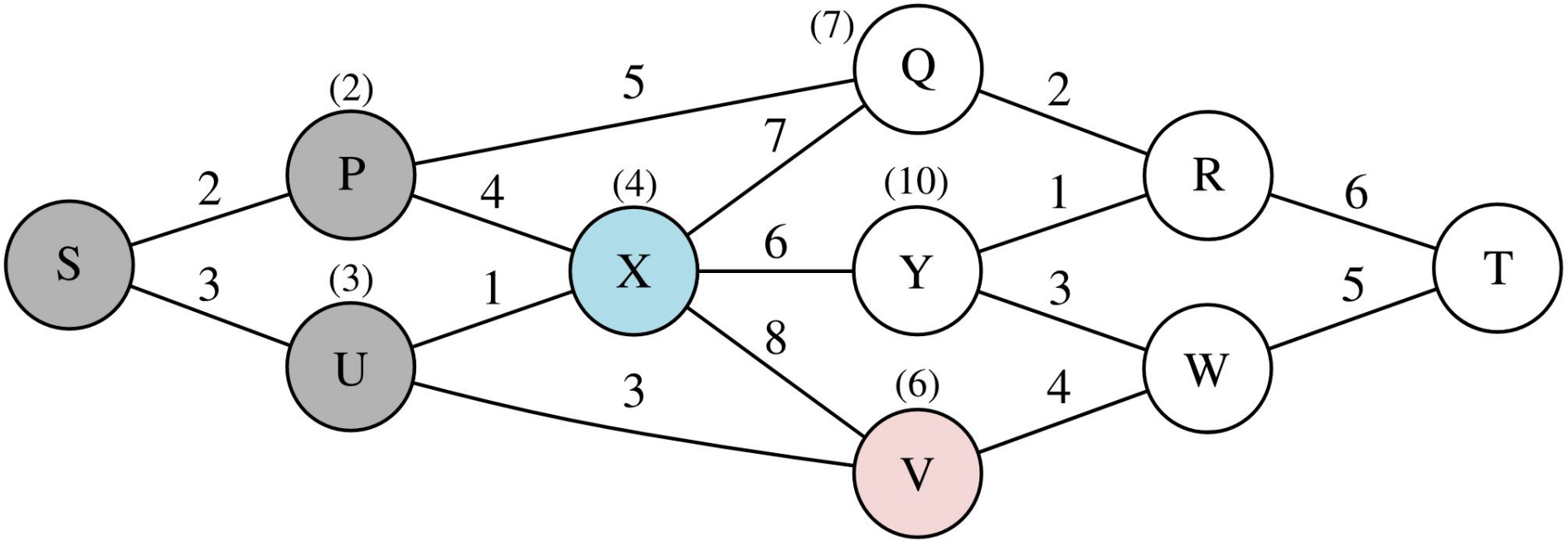

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5

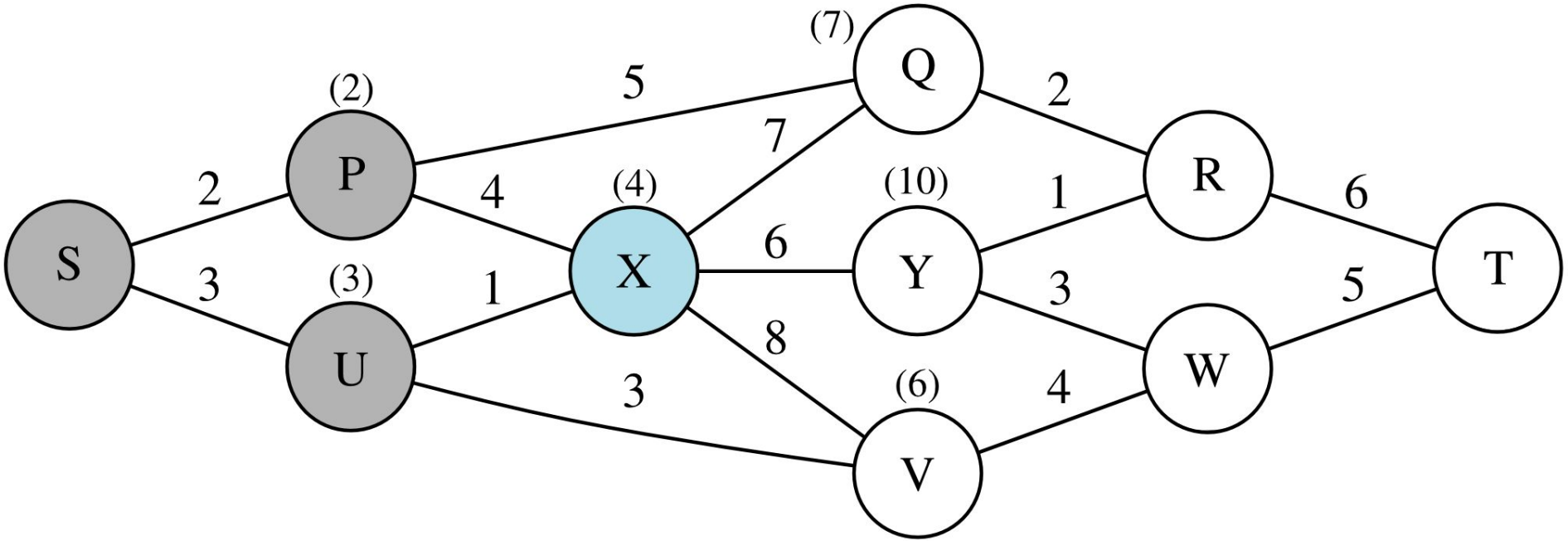

min(7, 11) = 7
Q
(2)
P
5
2
7
2
R
S
2
4
(4)
1
6
3
(3)
1
X
6
Y
T
3
5
U
8
W
3
(6)
4
V

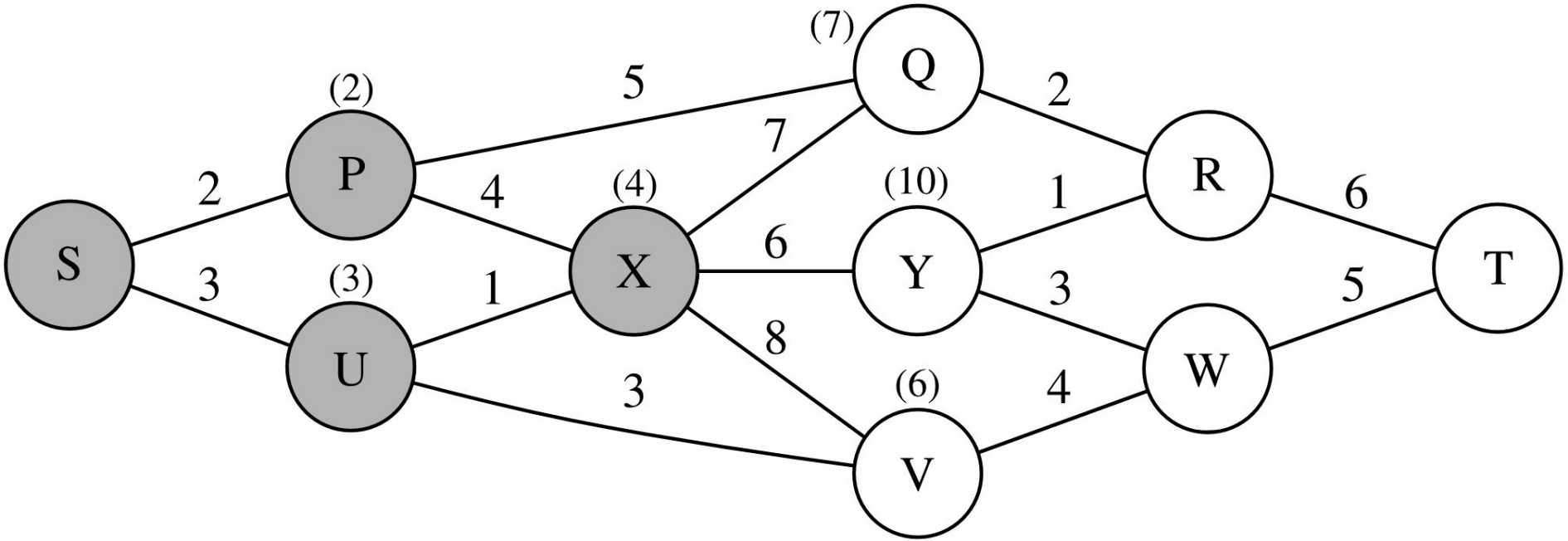

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

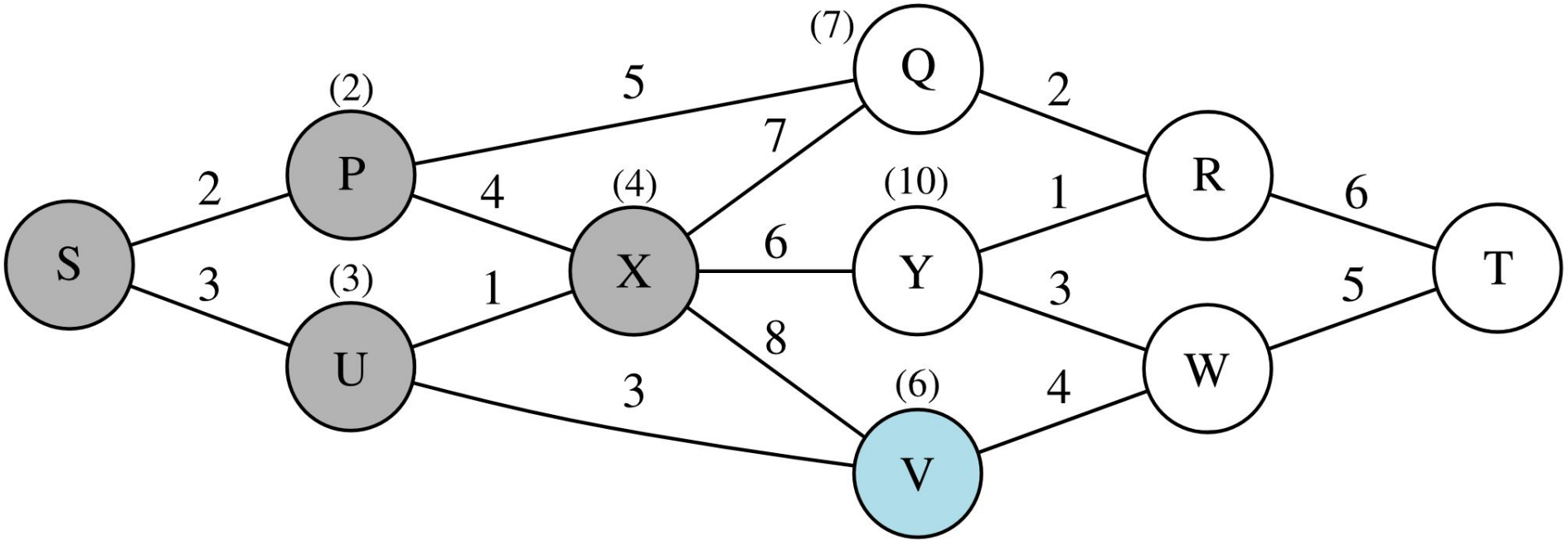

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

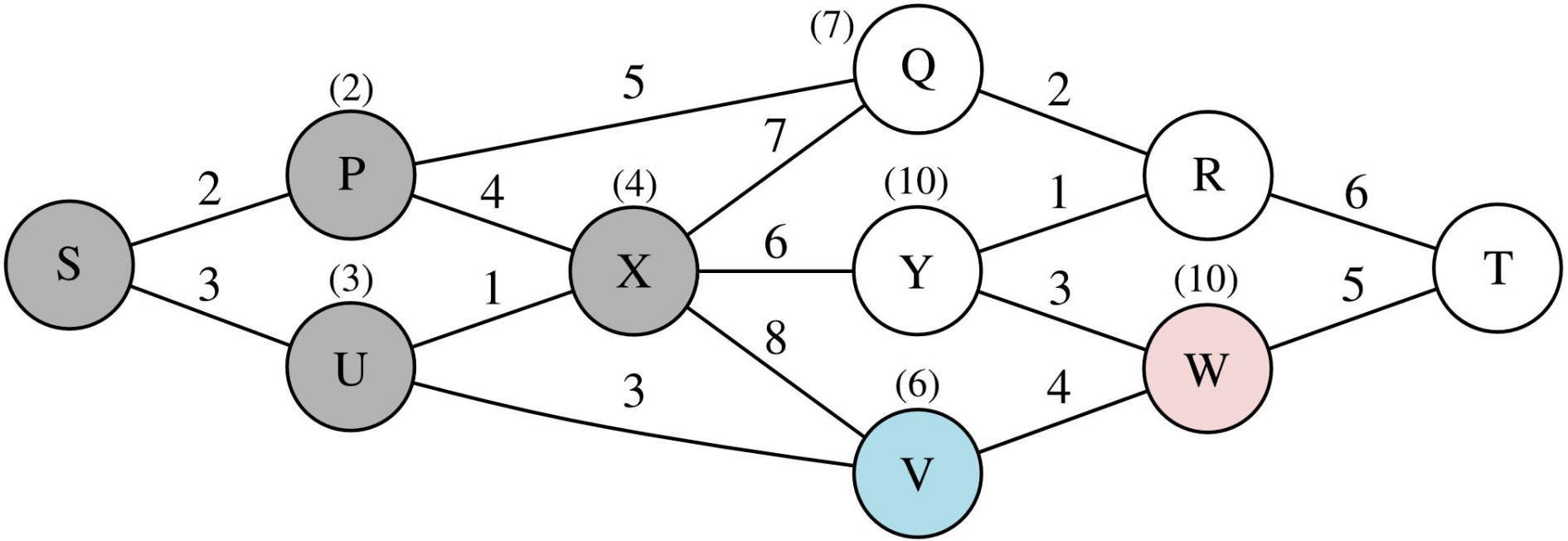

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
2
3
4
1
5
7
6
8
3
2
1
3
4
6
5
min(6, 12) = 6

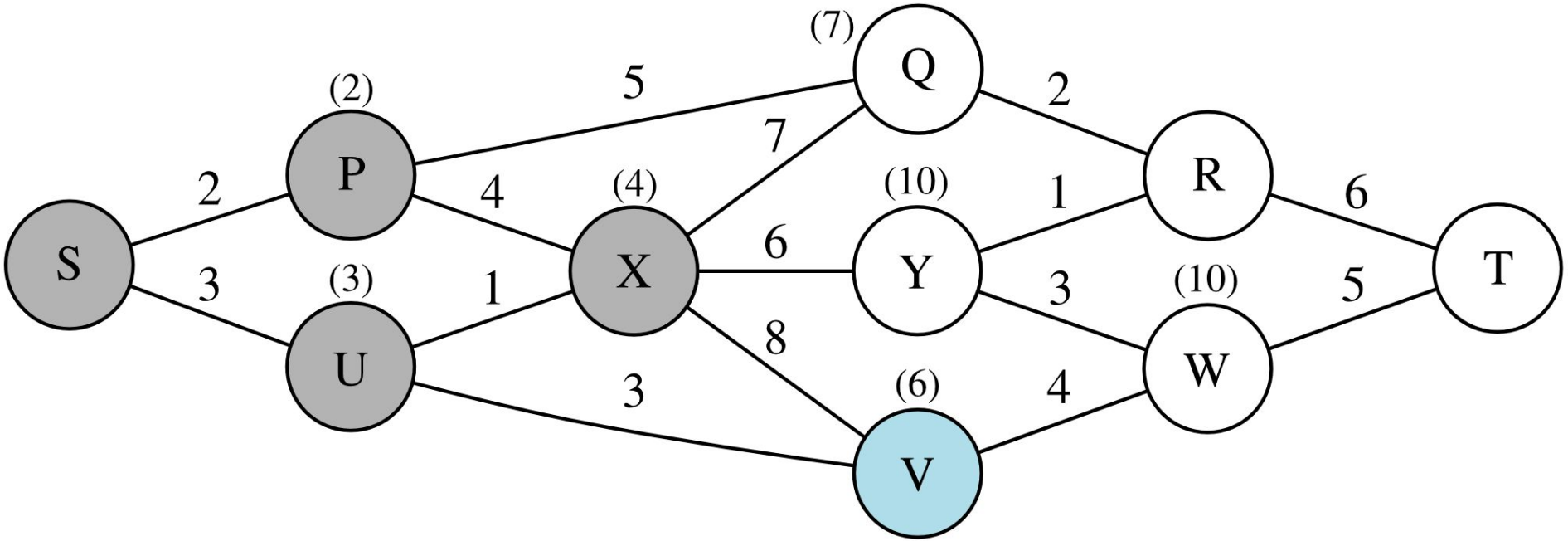

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

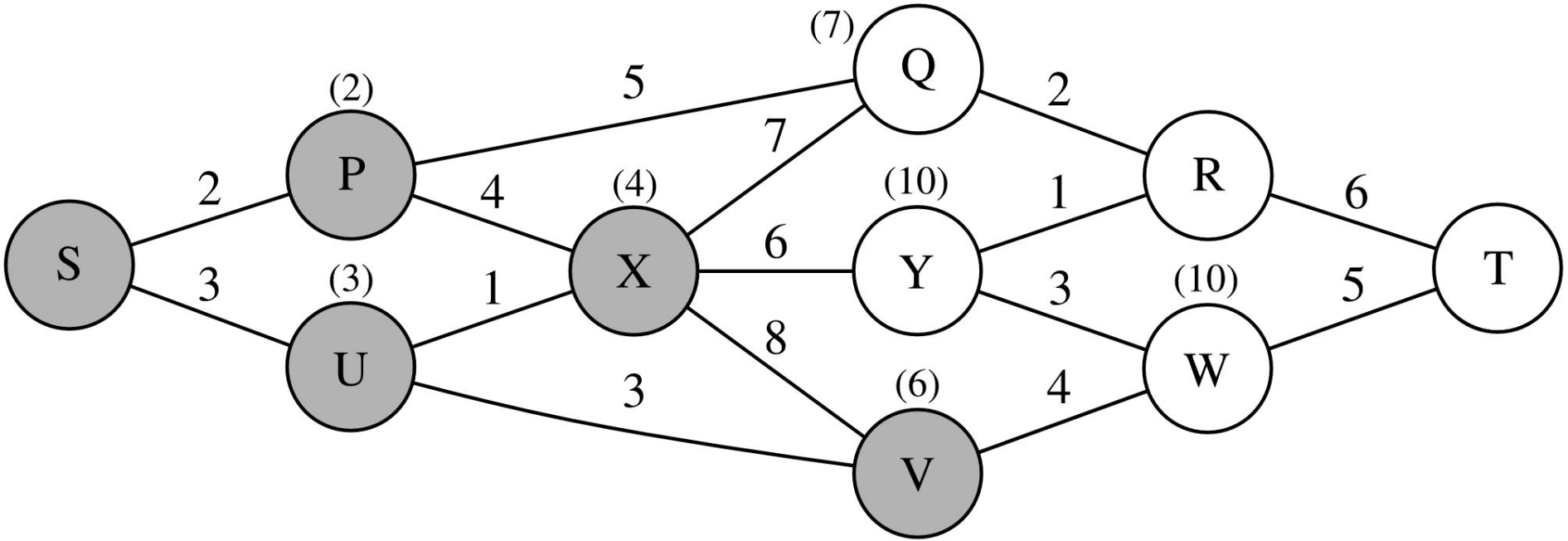

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

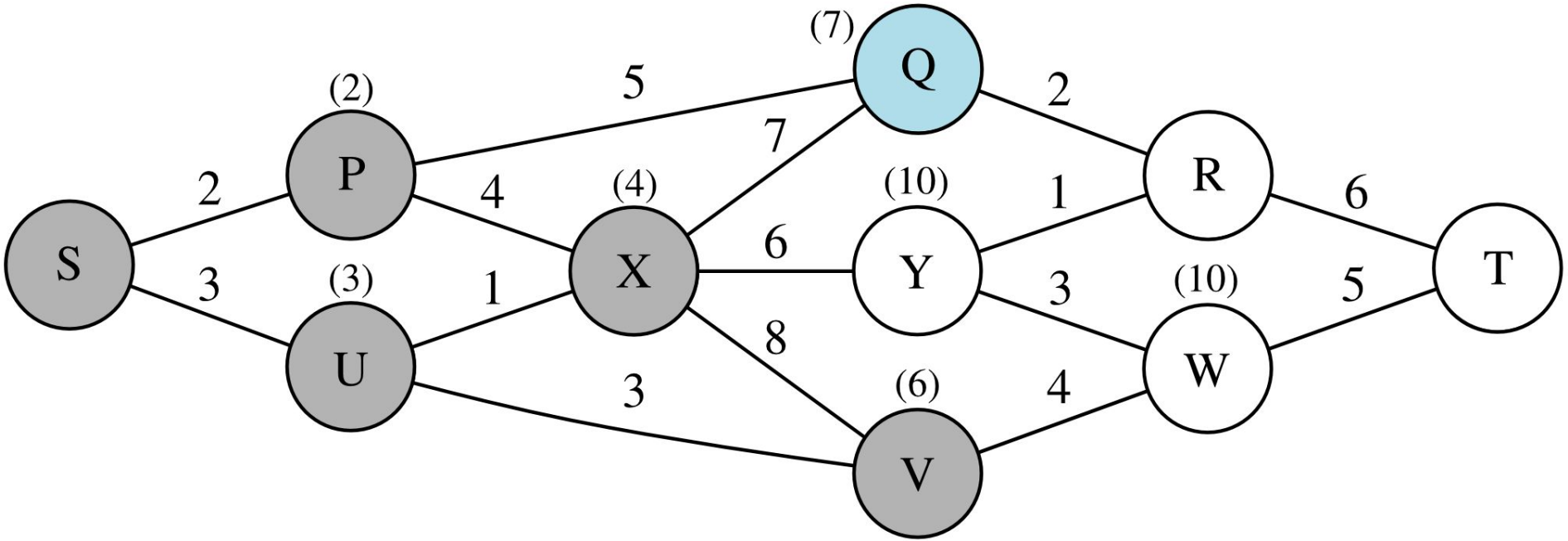

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

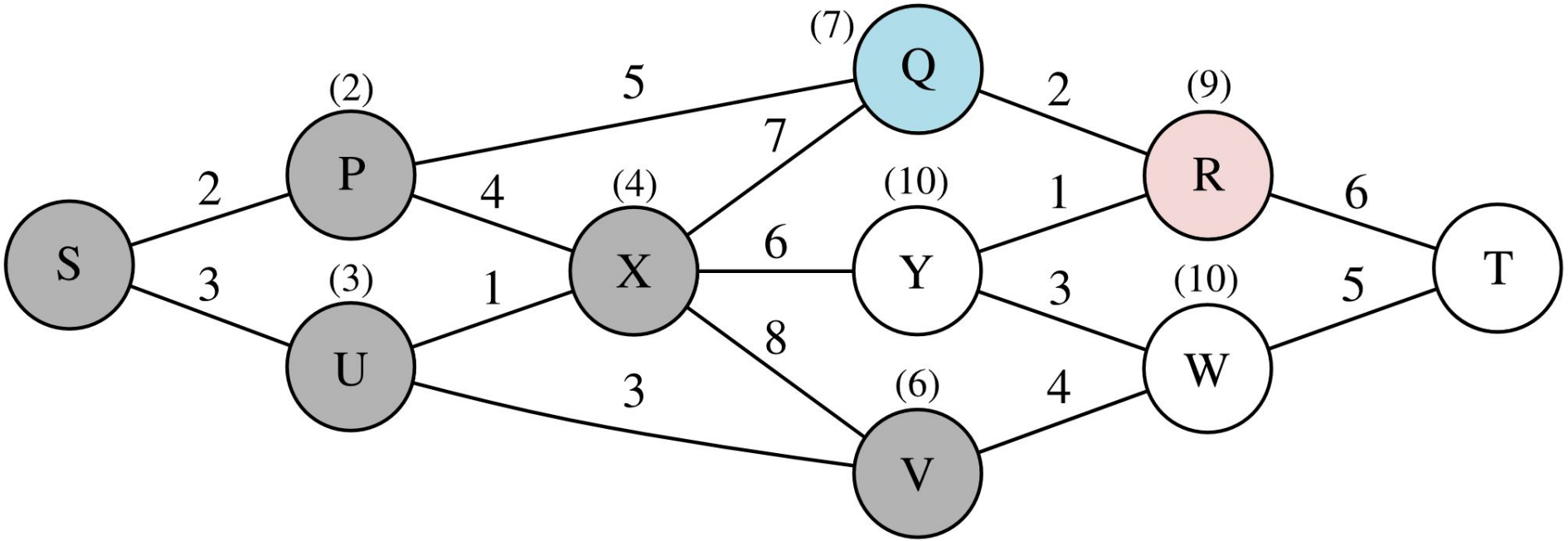

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

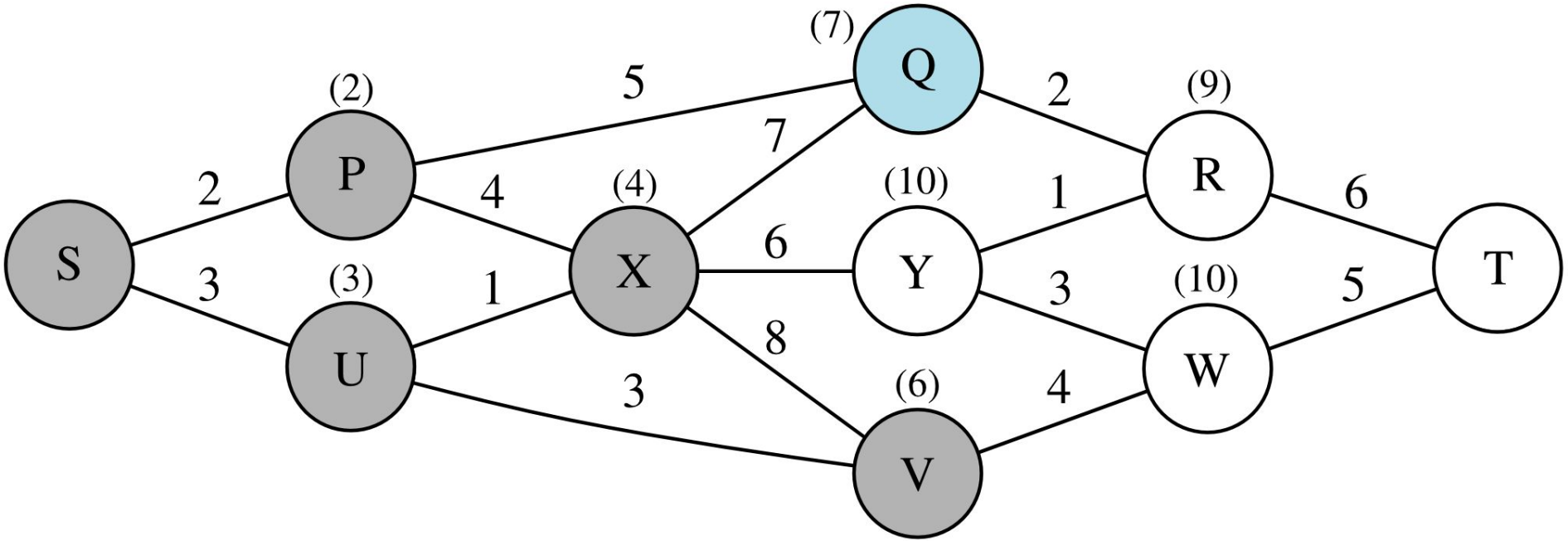

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(10)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

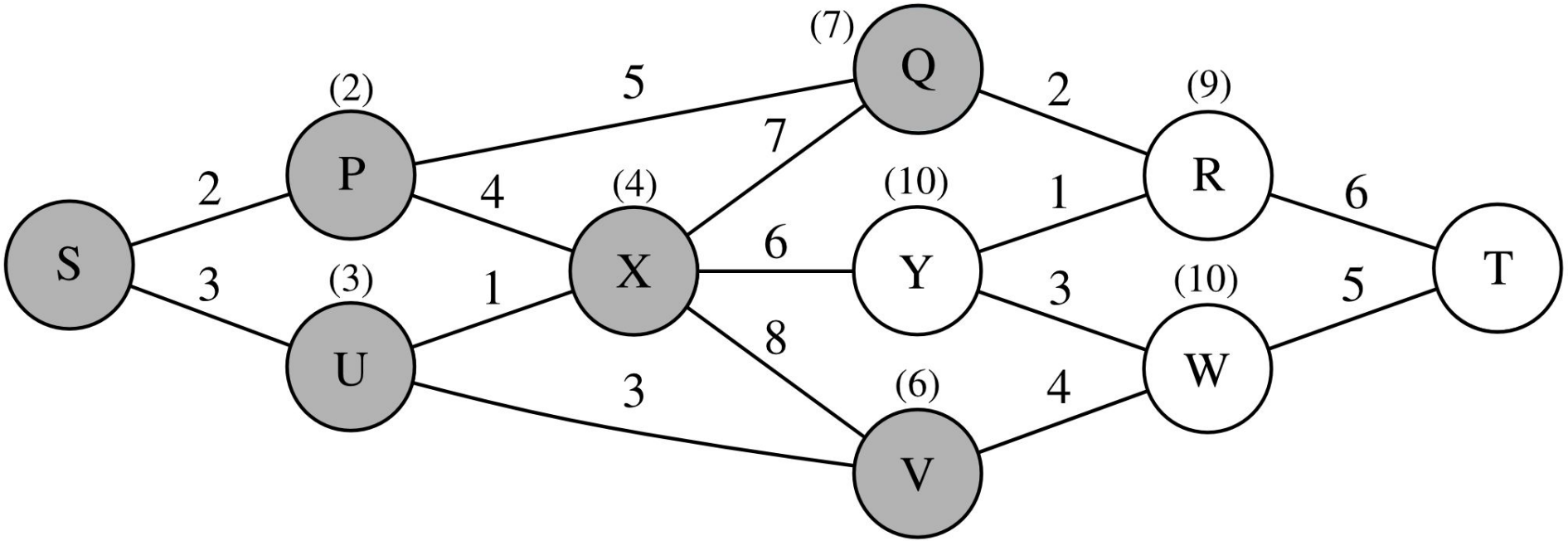

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(10)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

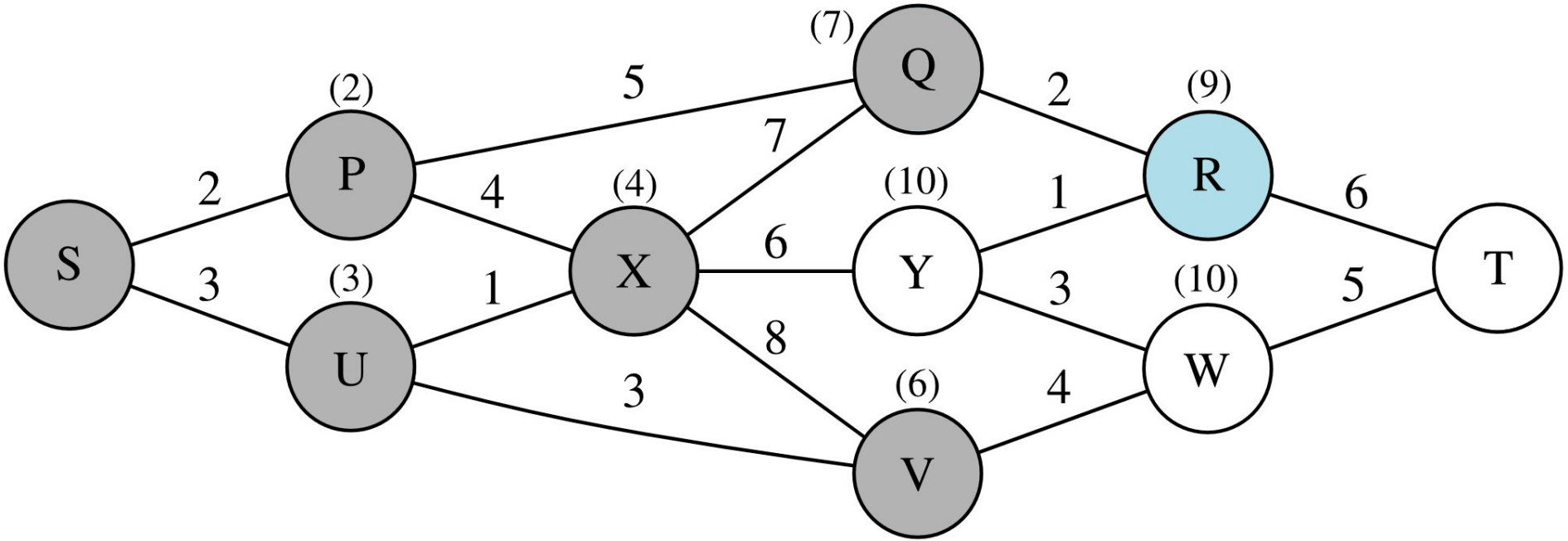

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(10)
2
3
5
4
7
1
6
8
3
2
1
3
4
6
5

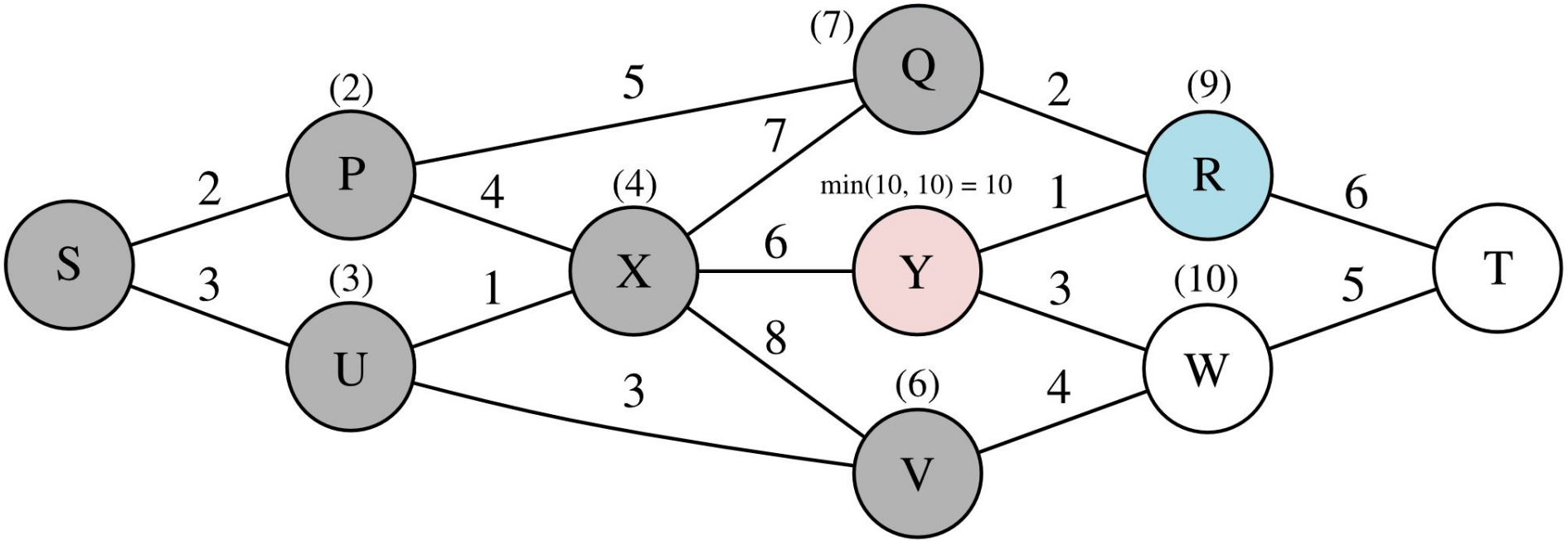

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(10)
2
3
4
5
1
7
6
8
3
1
3
4
2
6
5

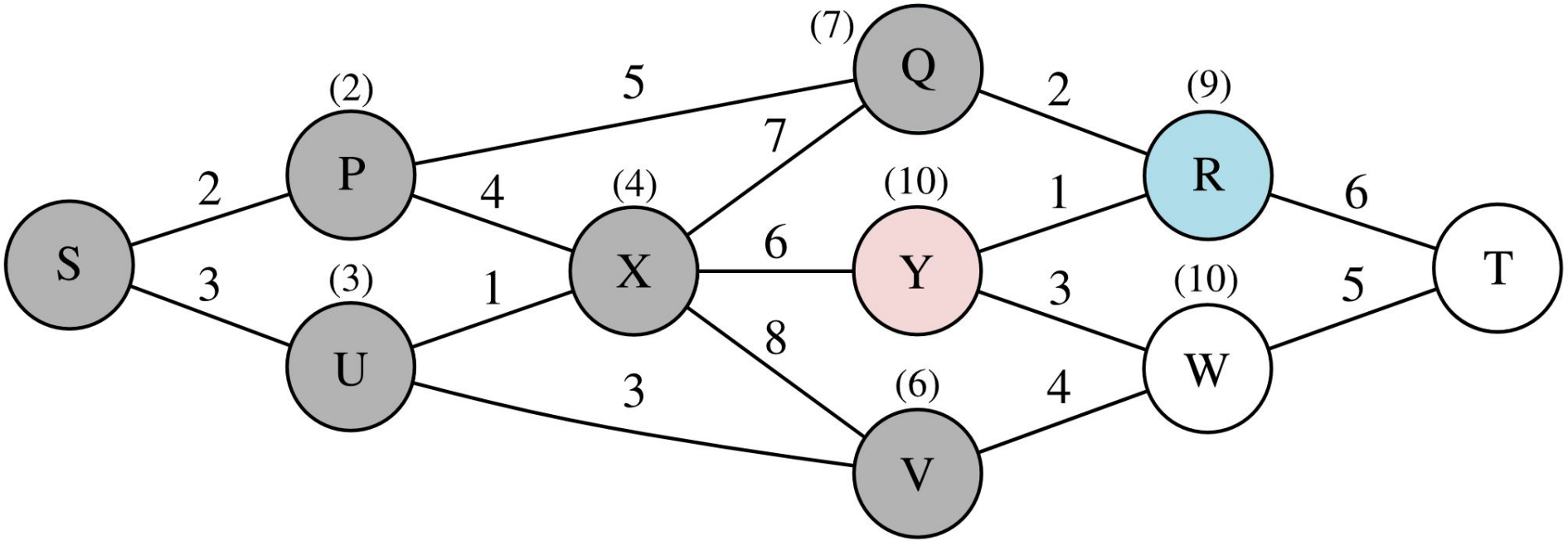

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

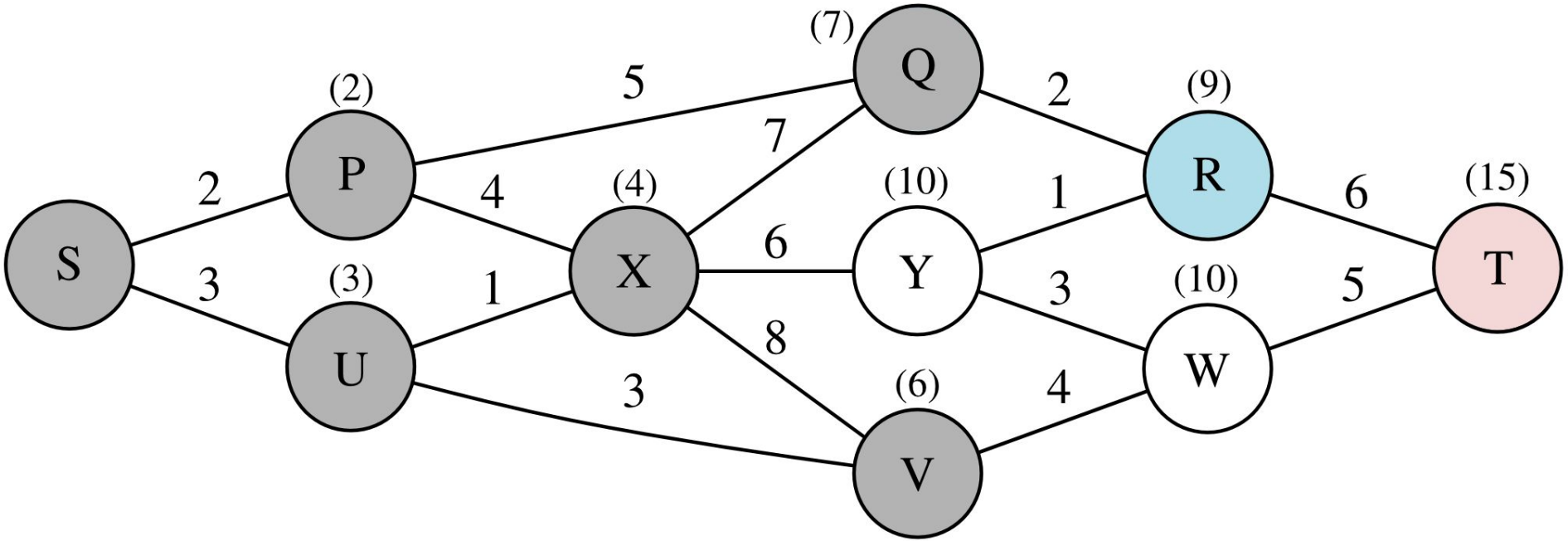

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

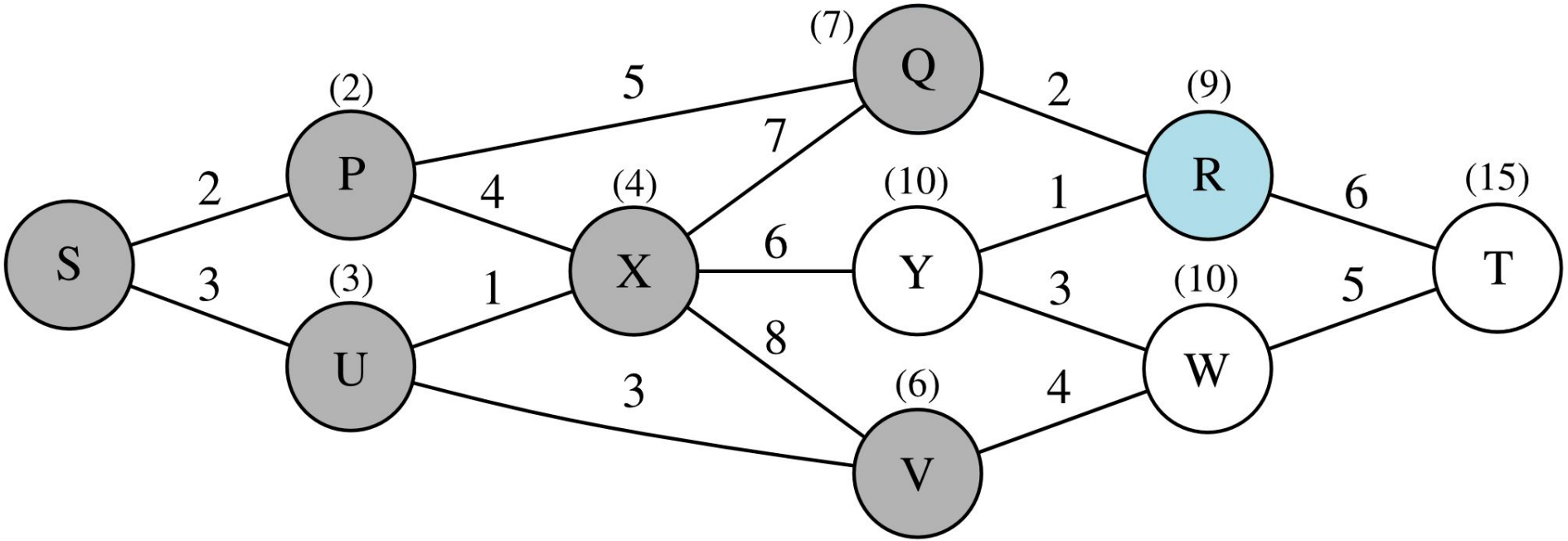

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

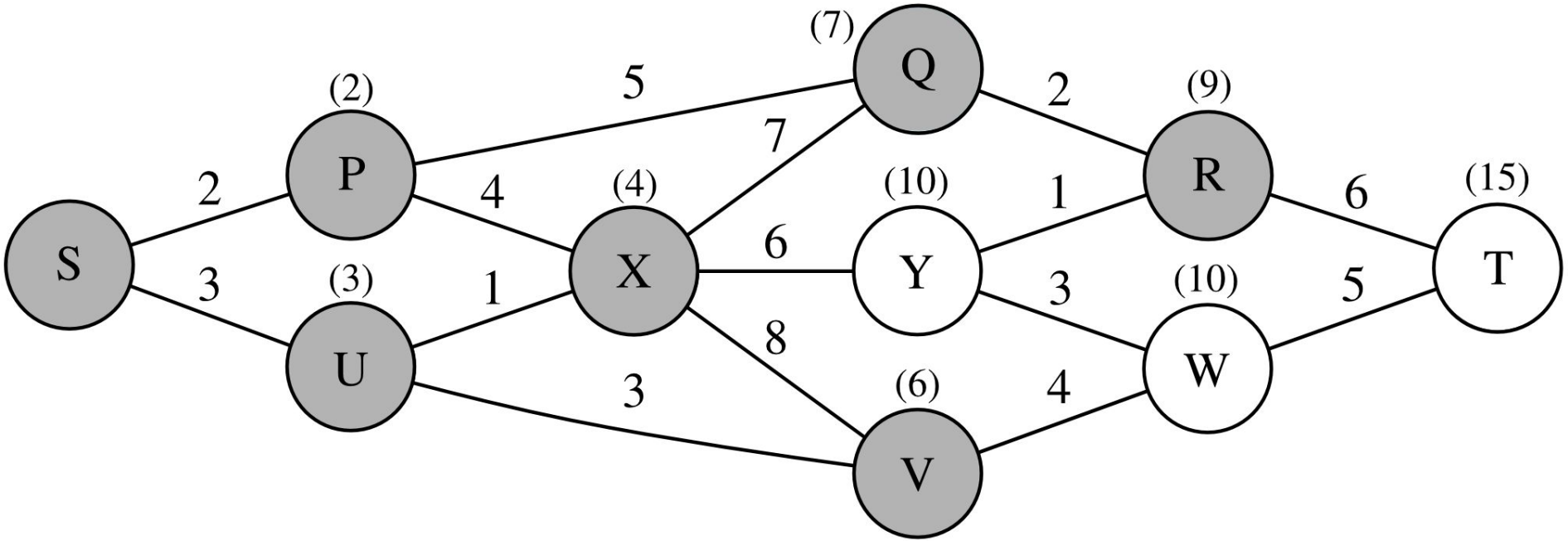

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
2
3
5
4
1
7
6
8
3
2
1
3
4
6
5

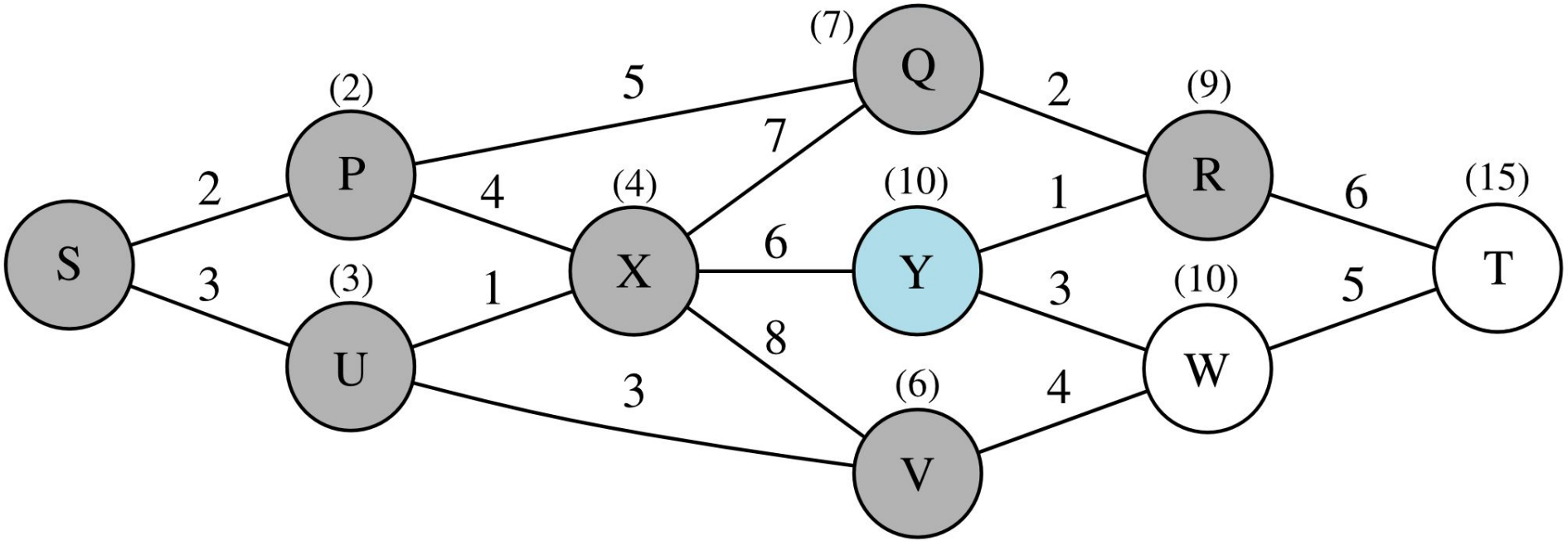

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(6)
(9)
(10)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5
min(10, 10) = 10

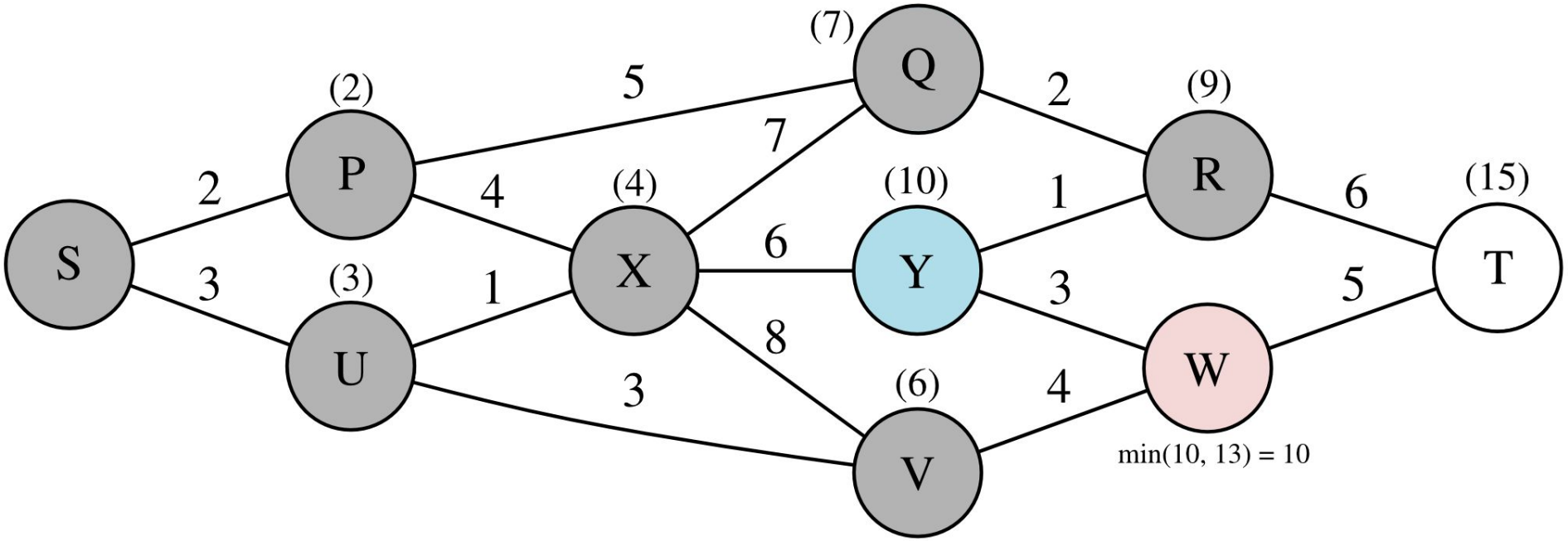

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

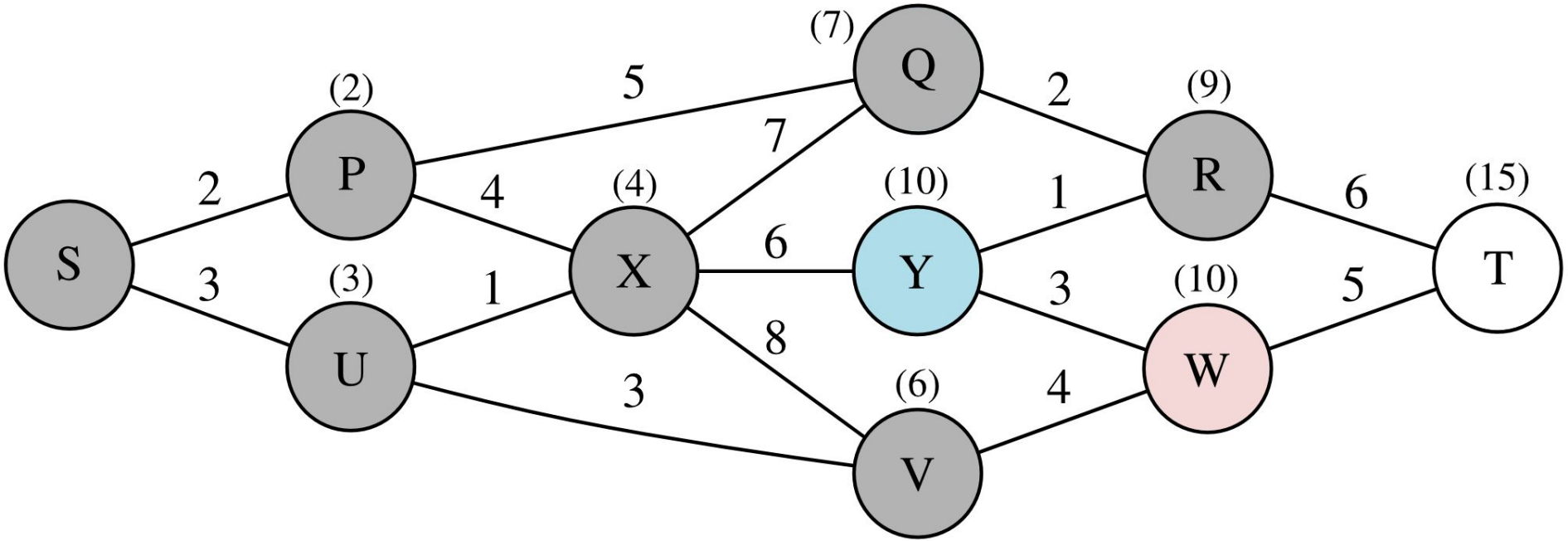

S
P
(2)
U
(3)
X
(4)
Q
(7)
Y
(10)
V
(6)
R
(9)
W
(10)
T
(15)
2
3
5
4
7
1
6
8
3
2
1
6
3
4
5

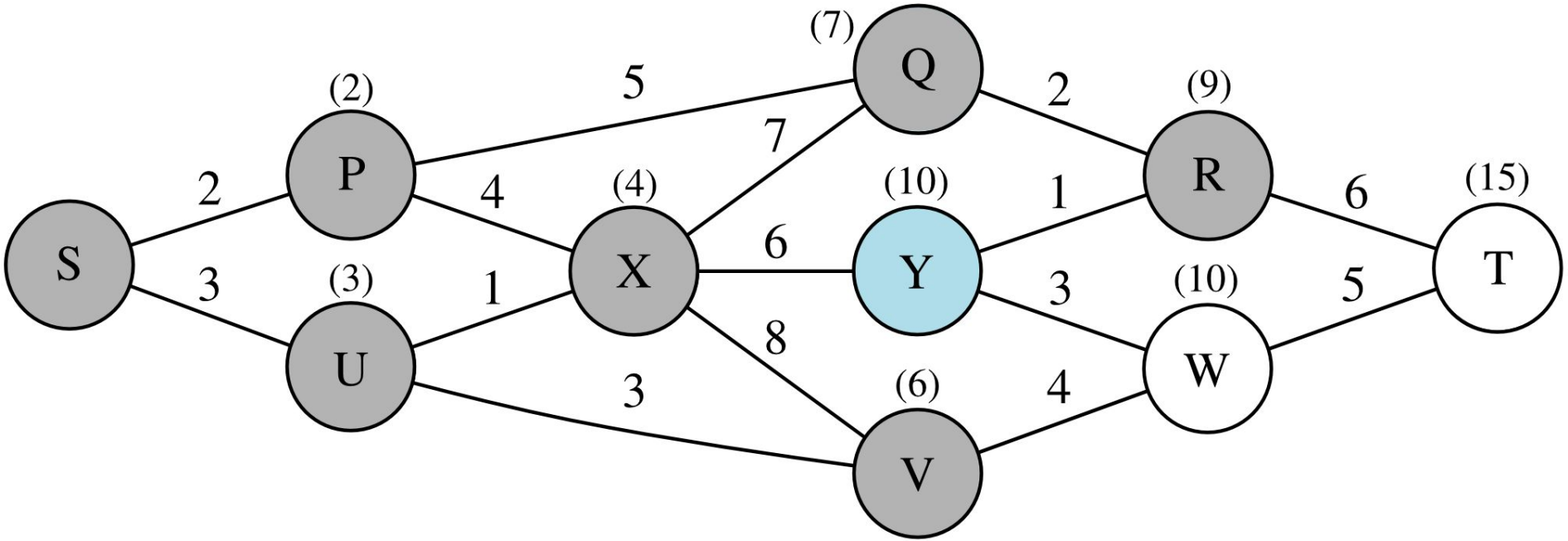

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

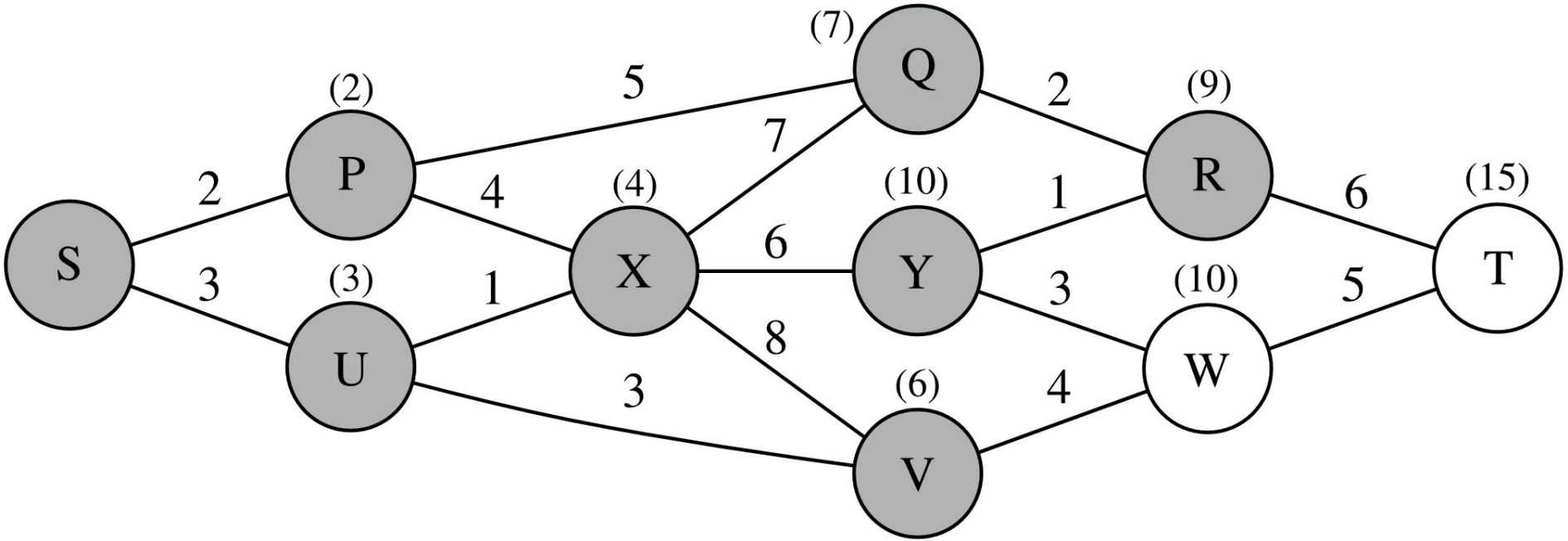

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

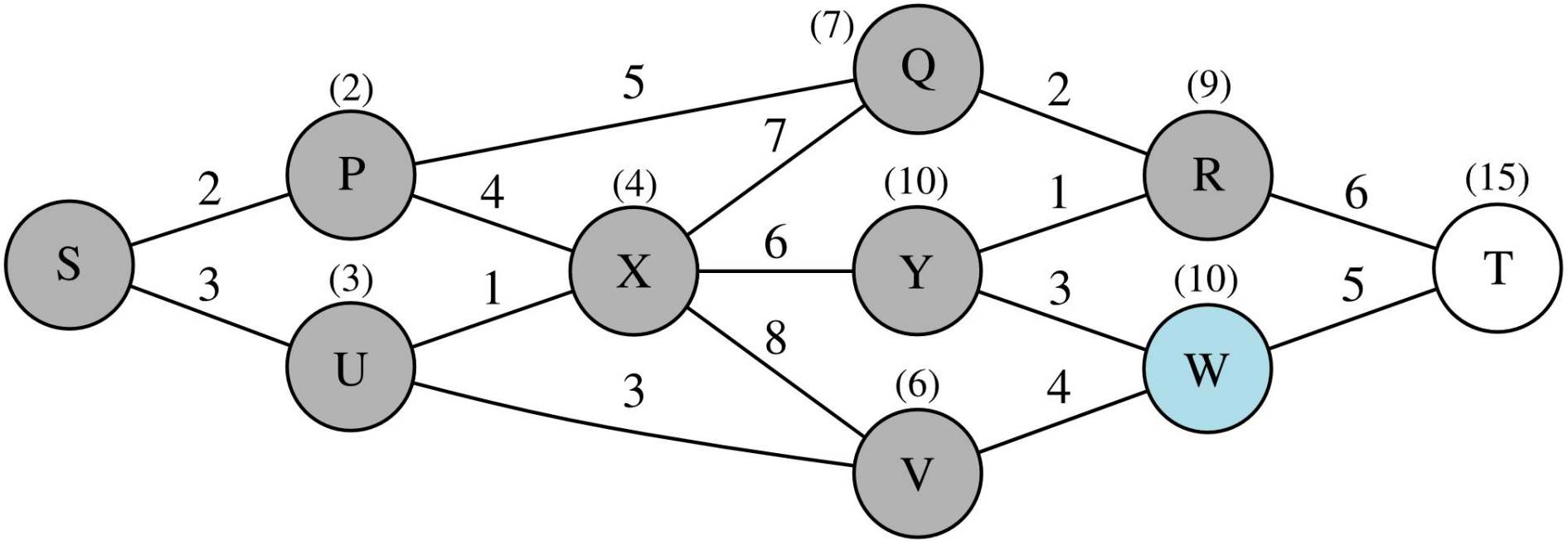

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

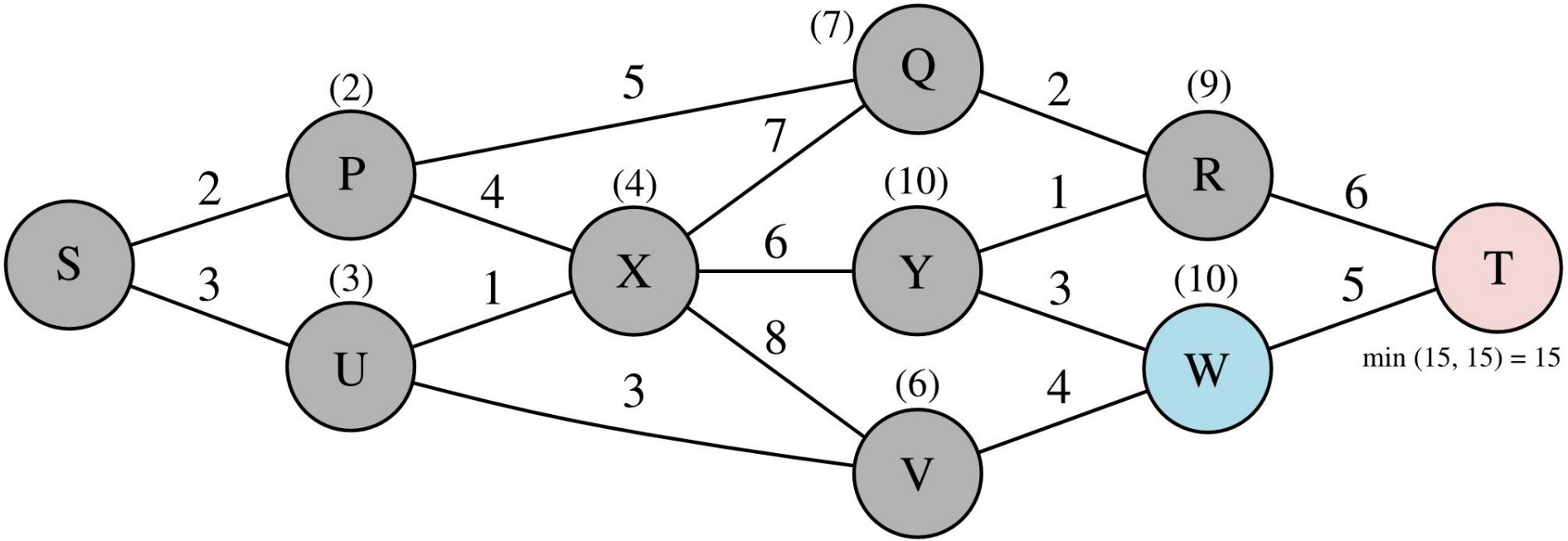

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5
min(10, 13) = 10

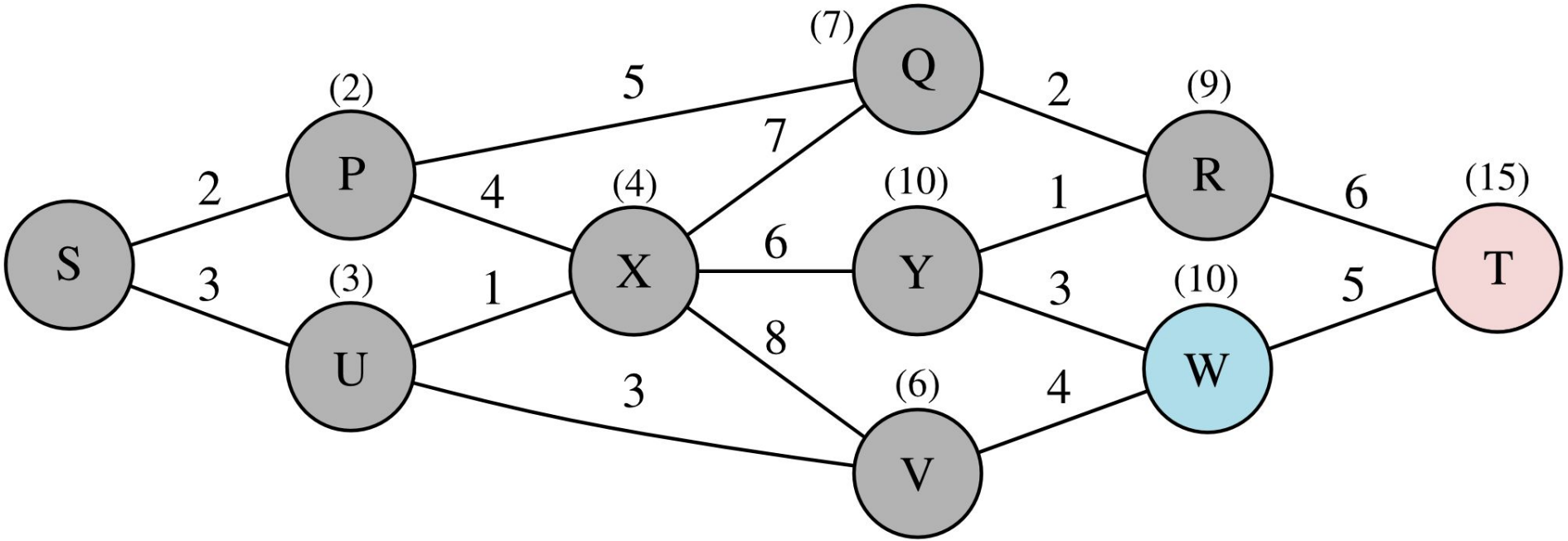

S
P
(2)
U
(3)
X
(4)
Q
(7)
Y
(10)
V
(6)
R
(9)
W
(10)
T
(15)
2
3
5
4
7
1
6
8
3
2
1
6
3
4
5

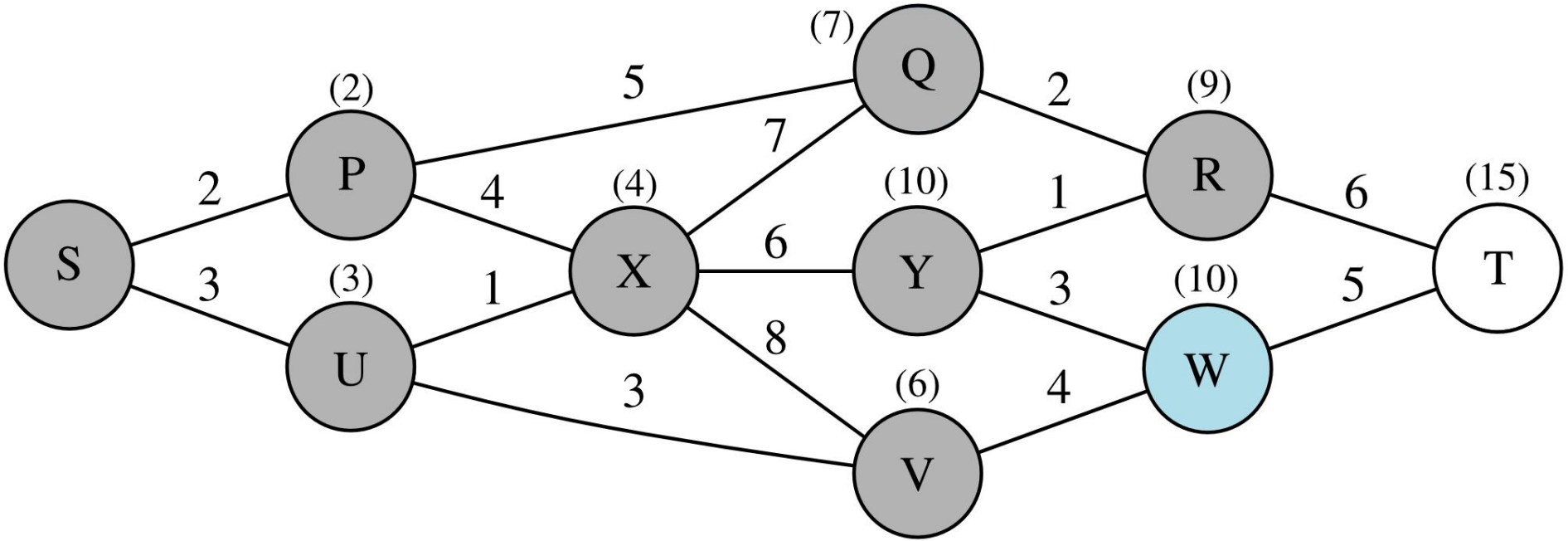

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5

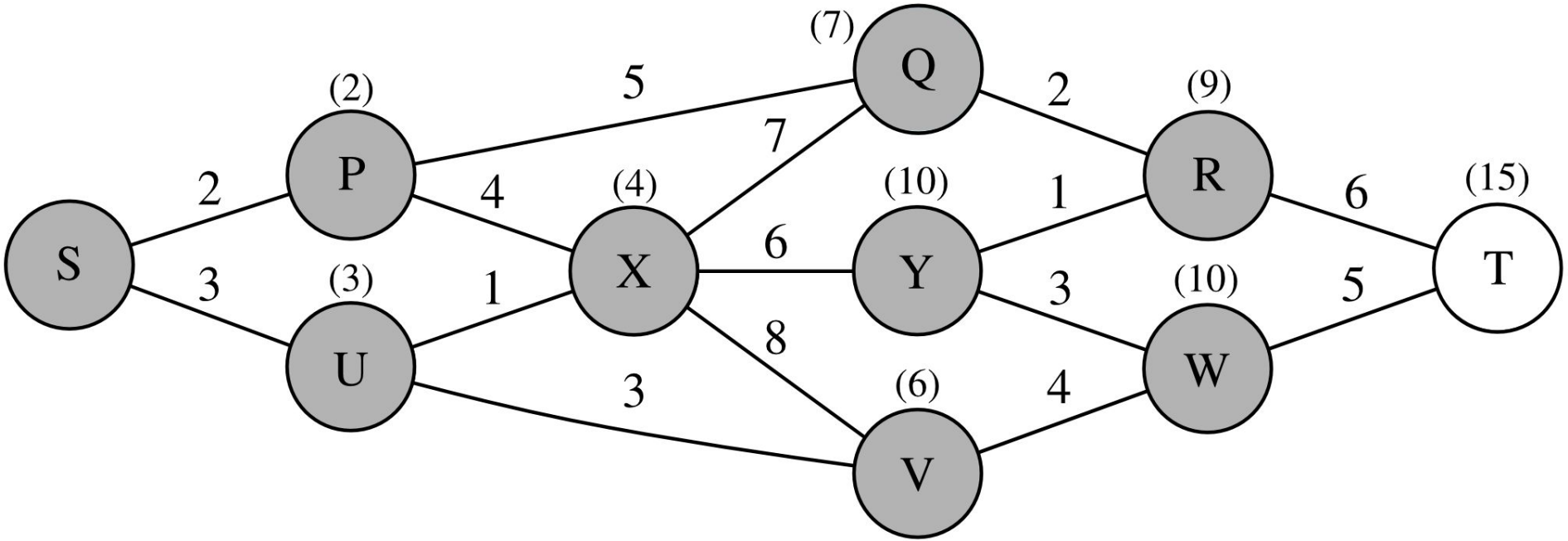

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

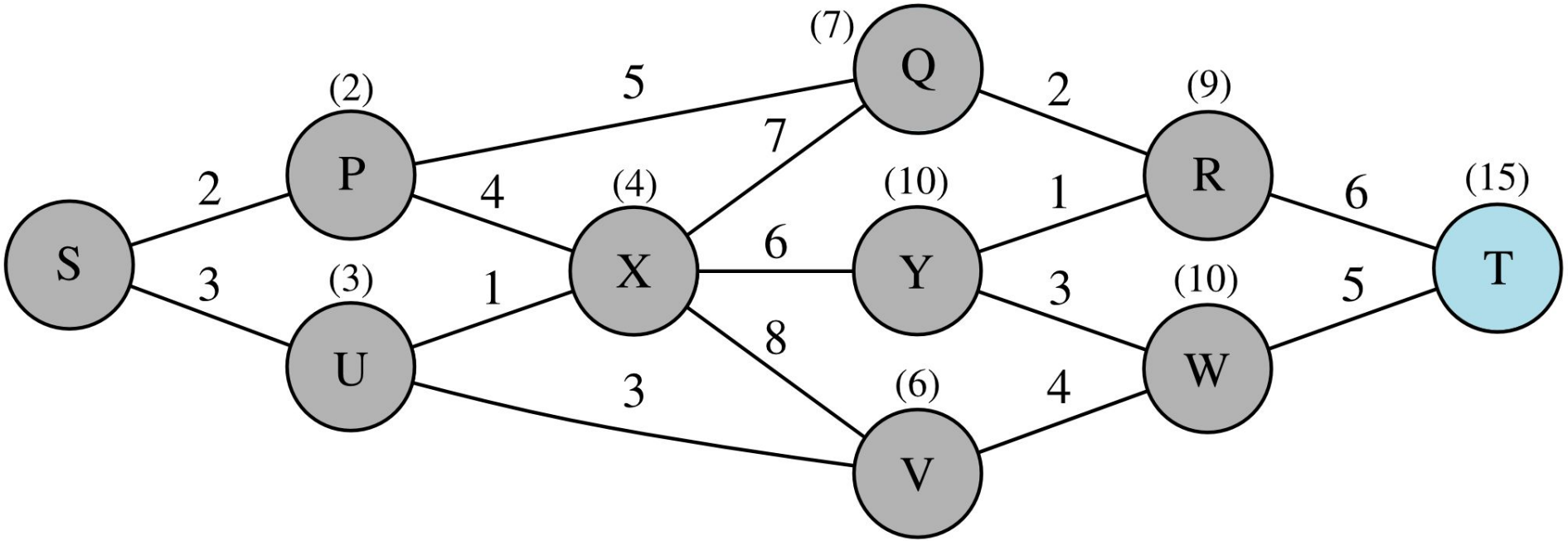

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

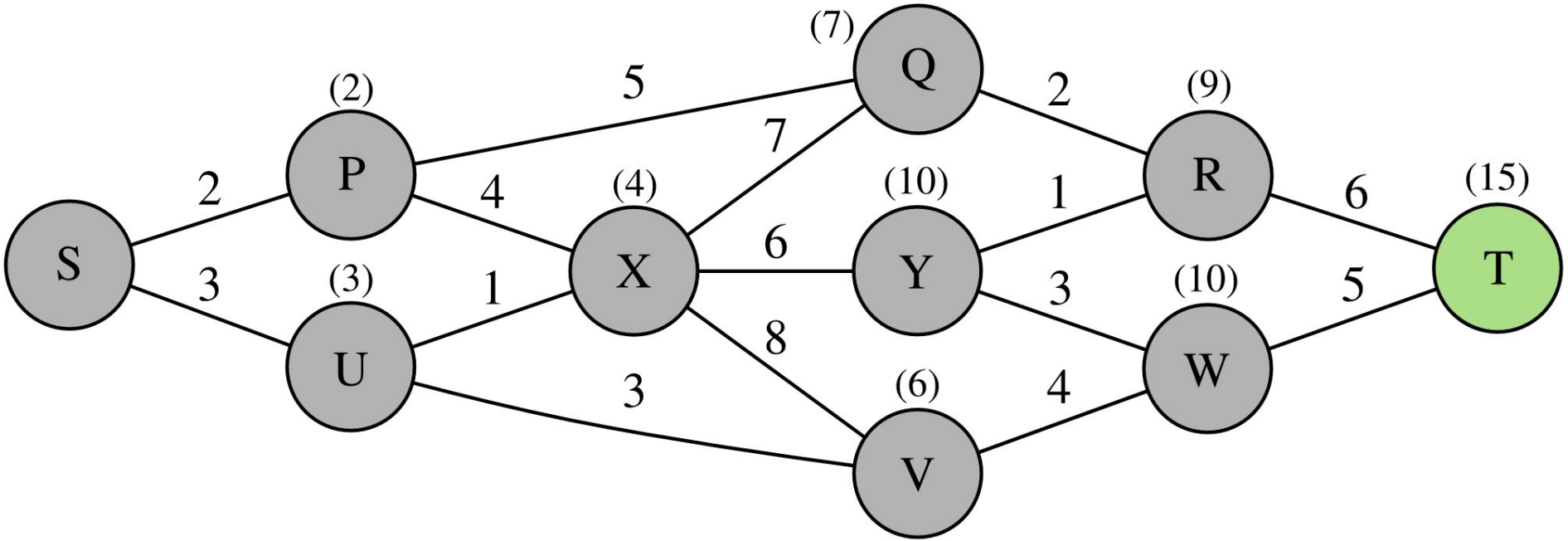

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(9)
(10)
(6)
2
3
5
4
7
1
6
8
3
2
1
3
4
6
5
min (15, 15) = 15

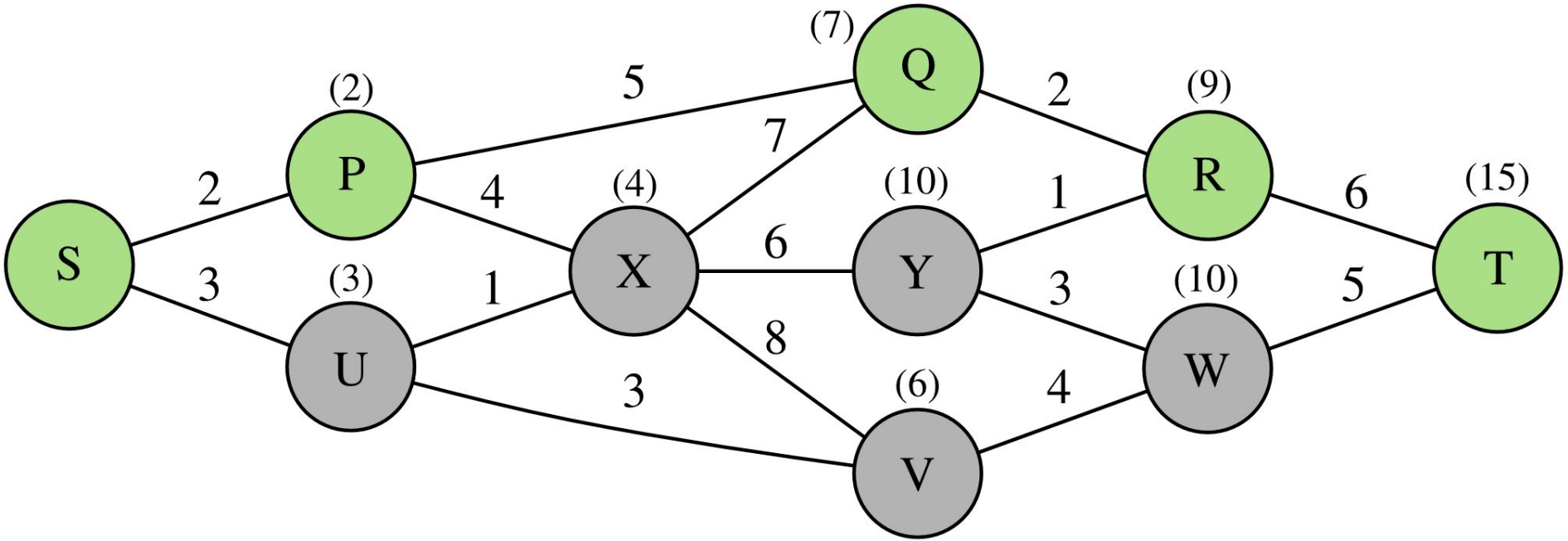

S
P
(2)
U
(3)
X
(4)
Q
(7)
Y
(10)
V
(6)
R
(9)
W
(10)
T
(15)
2
3
5
4
7
1
6
8
3
2
1
6
3
4
6
5

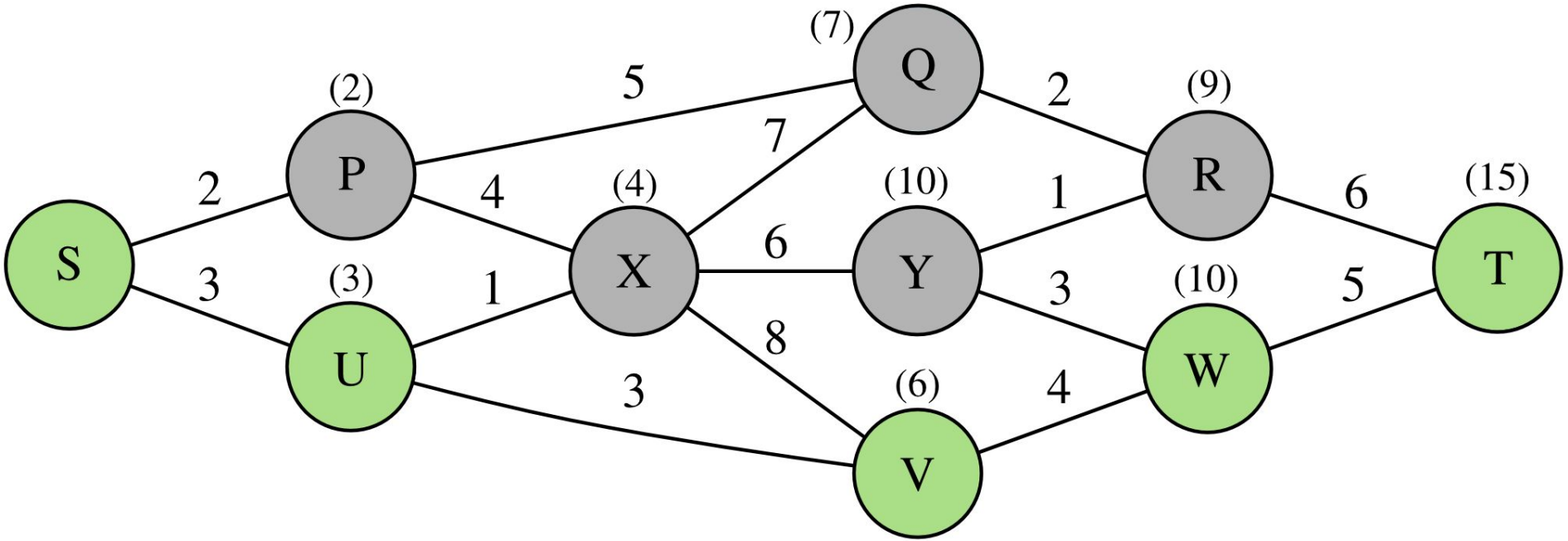

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5

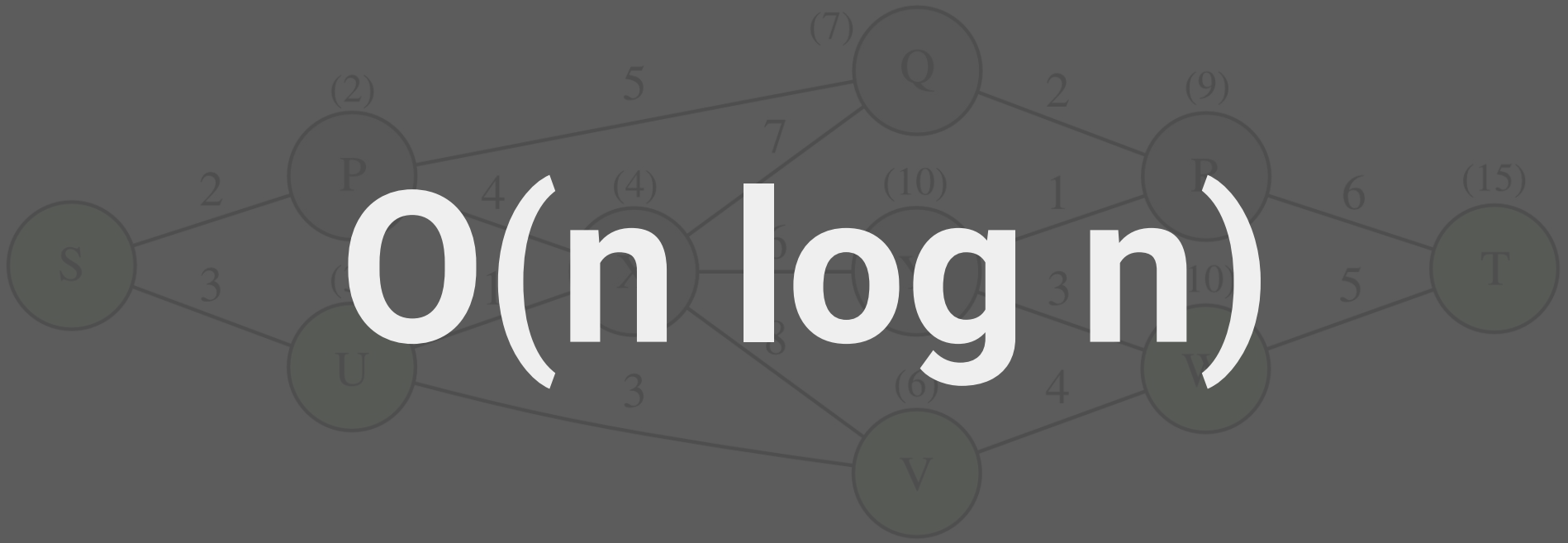

S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
1
7
6
8
3
2
1
3
4
6
5


S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5


S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5


S
P
(2)
U
(3)
X
(4)
Q
(7)
Y
(10)
V
(6)
R
(9)
W
(10)
T
(15)
2
3
5
4
7
1
6
8
3
2
1
6
3
4
5


S
P
U
X
Q
Y
V
R
W
T
(2)
(3)
(4)
(7)
(10)
(6)
(9)
(10)
(15)
2
3
4
5
7
1
6
8
3
2
1
3
4
6
5



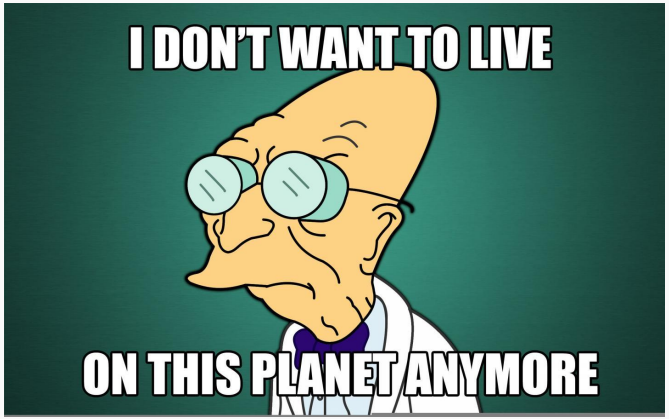
O(n log n)

Pseudocode [edit]

In the following algorithm, the code `u ← vertex in Q with min dist[u]`, searches for the vertex *u* in the vertex set *Q* that has the least `dist[u]` value. `length(u, v)` returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes *u* and *v*. The variable *alt* on line 19 is the length of the path from the root node to the neighbor node *v* if it were to go through *u*. If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path. The `prev` array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

```
 1  function Dijkstra(Graph, source):
 2
 3      create vertex set Q
 4
 5      for each vertex v in Graph:             // Initialization
 6          dist[v] ← INFINITY                  // Unknown distance from source to v
 7          prev[v] ← UNDEFINED                 // Previous node in optimal path from source
 8          add v to Q                          // All nodes initially in Q (unvisited nodes)
 9
10      dist[source] ← 0                        // Distance from source to source
11
12      while Q is not empty:
13          u ← vertex in Q with min dist[u]    // Source node will be selected first
14          remove u from Q
15
16          for each neighbor v of u:           // where v is still in Q.
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]:               // A shorter path to v has been found
19                  dist[v] ← alt
20                  prev[v] ← u
21
22      return dist[], prev[]
```

```
from collections import namedtuple, queue
from pprint import pprint as pp


inf = float('inf')
Edge = namedtuple('Edge', 'start, end, cost')

class Graph():
    def __init__(self, edges):
        self.edges = edges2 = [Edge(*edge) for edge in edges]
        self.vertices = set(sum(([e.start, e.end] for e in edges2), []))

    def dijkstra(self, source, dest):
        assert source in self.vertices
        dist = {vertex: inf for vertex in self.vertices}
        previous = {vertex: None for vertex in self.vertices}
        dist[source] = 0
        q = self.vertices.copy()
        neighbours = {vertex: set() for vertex in self.vertices}
        for start, end, cost in self.edges:
            neighbours[start].add((end, cost))
        #pp(neighbours)

        while q:
            u = min(q, key=lambda vertex: dist[vertex])
            q.remove(u)
            if dist[u] == inf or u == dest:
                break
            for v, cost in neighbours[u]:
                alt = dist[u] + cost
                if alt < dist[v]:                                    # Relax (u,v,a)
                    dist[v] = alt
                    previous[v] = u
        #pp(previous)
        s, u = deque(), dest
        while previous[u]:
            s.pushleft(u)
            u = previous[u]
        s.pushleft(u)
        return s
```



PYCON 2016

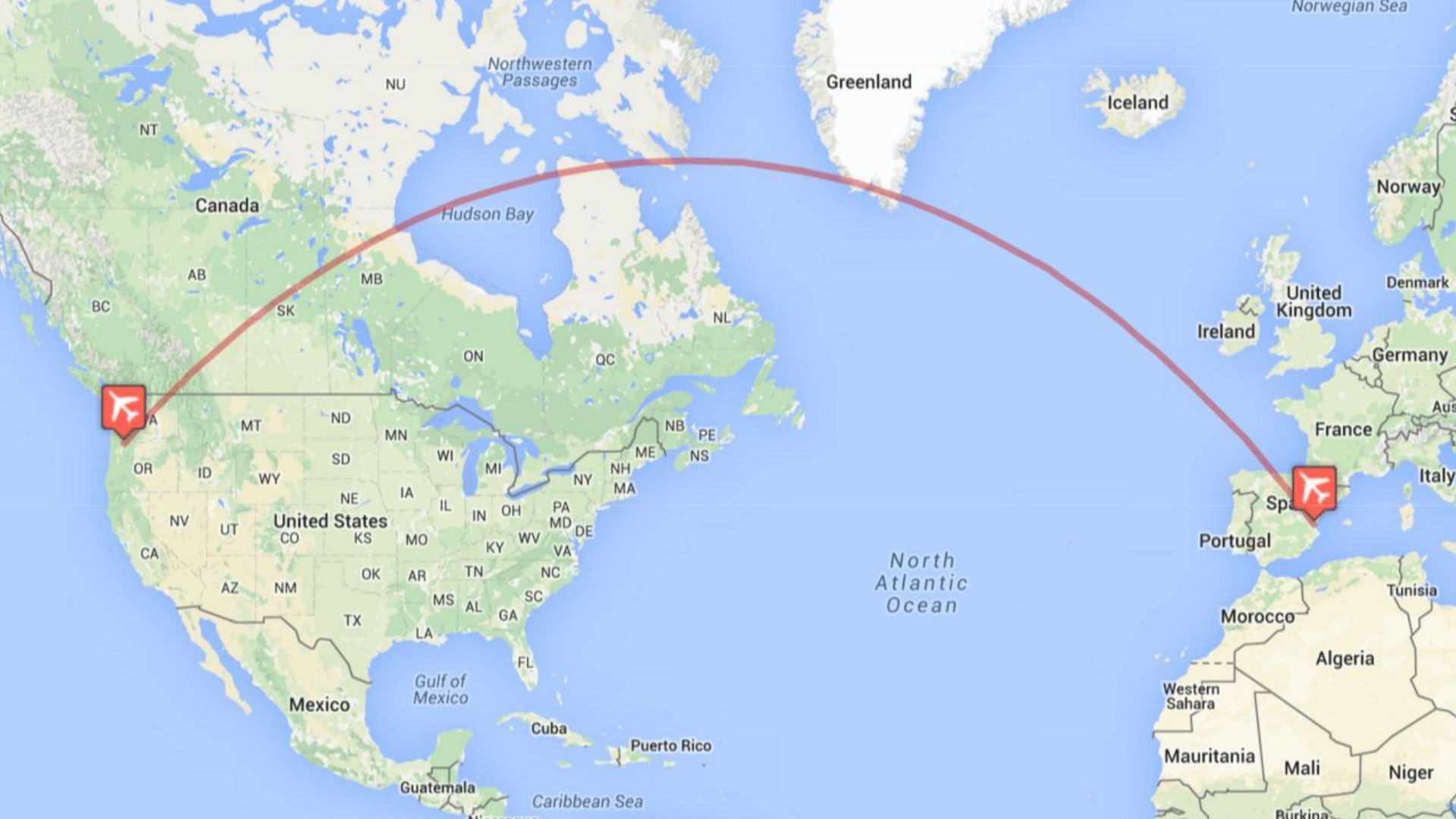
BY THE PYTHON COMMUNITY

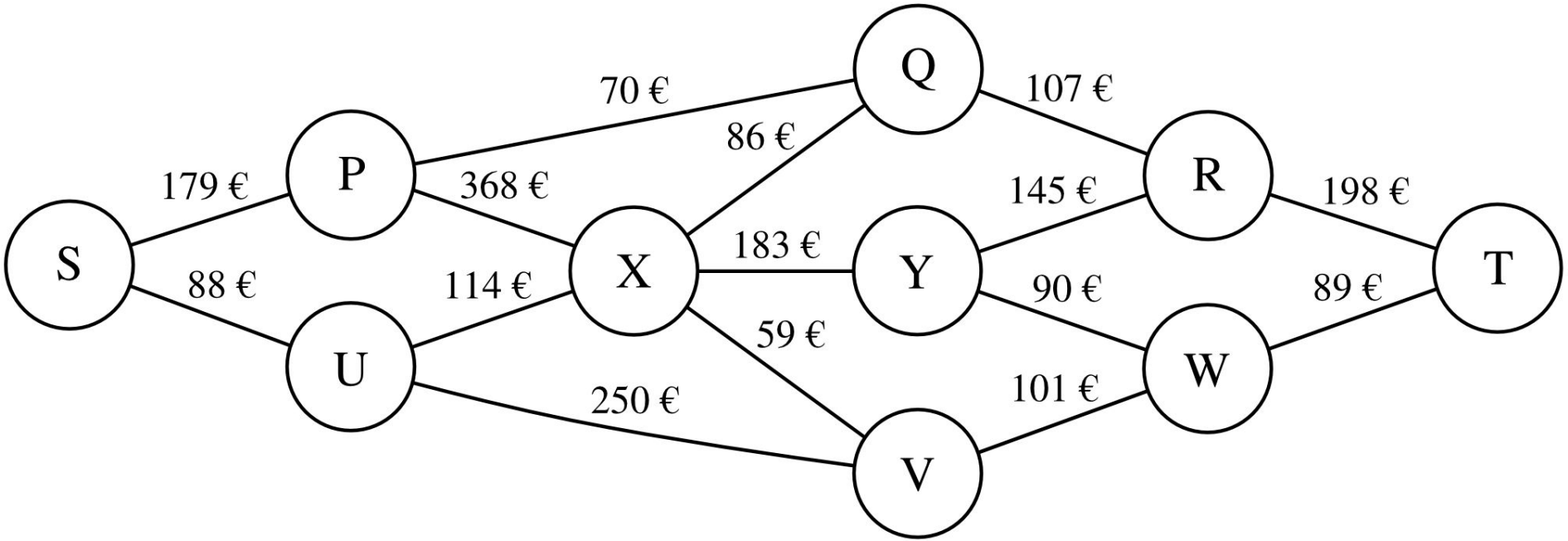
ROSE  CITY

FOR THE PYTHON COMMUNITY

PORTLAND, OREGON

MAY 28TH - JUNE 5TH

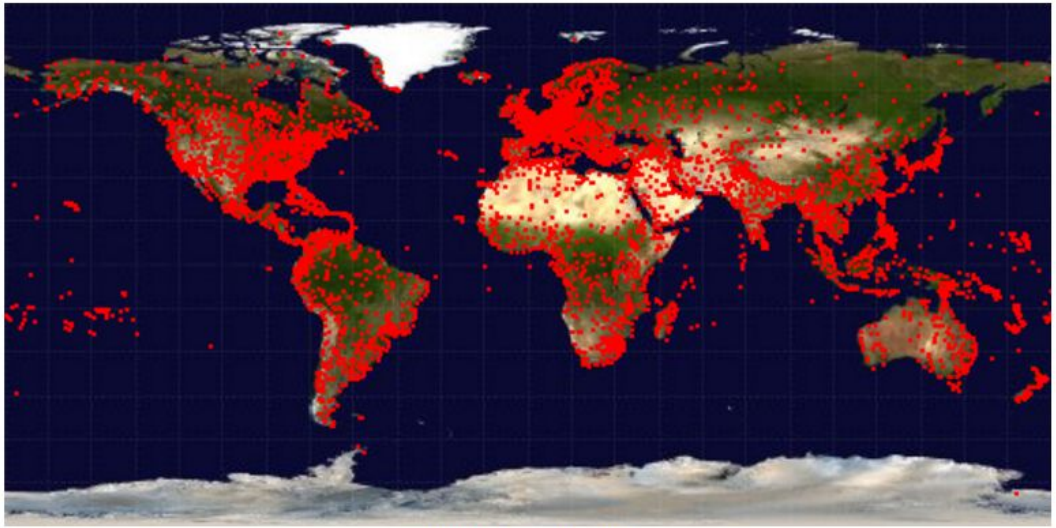

Norwegian Sea
Northwestern Passages
NU
Greenland
Iceland
NT
Canada
Hudson Bay
Norway
AB
MB
Denmark
BC
SK
United Kingdom
NL
Ireland
ON
QC
Germany
MT
ND
MN
NB
PE
France
SD
WI
ME
NS
OR
ID
WY
MI
NH
NY
Italy
NE
IA
MA
IL
IN
OH
PA
NV
UT
United States
CO
KS
MO
MD
DE
Portugal
CA
KY
WV
VA
North Atlantic Ocean
OK
AR
TN
NC
Tunisia
AZ
NM
MS
AL
SC
Morocco
TX
GA
LA
Algeria
FL
Gulf of Mexico
Western Sahara
Mexico
Cuba
Puerto Rico
Mauritania
Mali
Niger
Guatemala
Caribbean Sea


S
P
U
X
Q
Y
V
R
W
T
179 €
88 €
70 €
368 €
86 €
114 €
183 €
59 €
250 €
107 €
145 €
90 €
101 €
198 €
89 €

Airport, airline and route data

OpenFlights.org

Airport database



(click to enlarge)

As of January 2012, the OpenFlights Airports Database contains **6977** airports spanning the globe, as shown in the map above. Each entry contains the following information:

Airport ID	Unique OpenFlights identifier for this airport.
Name	Name of airport. May or may not contain the **City** name.
City	Main city served by airport. May be spelled differently from **Name**.
Country	Country or territory where airport is located.
IATA/FAA	3-letter FAA code, for airports located in **Country** "United States of America". 3-letter IATA code, for all other airports.

airports.dat

```
1,"Goroka","Goroka","Papua New Guinea","GKA","AYGA",-6.081689,145.391881,5282,10,"U","Pacific/Port_Moresby"
2,"Madang","Madang","Papua New Guinea","MAG","AYMD",-5.207083,145.7887,20,10,"U","Pacific/Port_Moresby"
3,"Mount Hagen","Mount Hagen","Papua New Guinea","HGU","AYMH",-5.826789,144.295861,5388,10,"U","Pacific/Port_Moresby"
4,"Nadzab","Nadzab","Papua New Guinea","LAE","AYNZ",-6.569828,146.726242,239,10,"U","Pacific/Port_Moresby"
5,"Port Moresby Jacksons Intl","Port Moresby","Papua New Guinea","POM","AYPY",-9.443383,147.22005,146,10,"U","Pacific/Port_
6,"Wewak Intl","Wewak","Papua New Guinea","WWK","AYWK",-3.583828,143.669186,19,10,"U","Pacific/Port_Moresby"
7,"Narsarsuaq","Narssarssuaq","Greenland","UAK","BGBW",61.160517,-45.425978,112,-3,"E","America/Godthab"
8,"Nuuk","Godthaab","Greenland","GOH","BGGH",64.190922,-51.678064,283,-3,"E","America/Godthab"
9,"Sondre Stromfjord","Sondrestrom","Greenland","SFJ","BGSF",67.016969,-50.689325,165,-3,"E","America/Godthab"
10,"Thule Air Base","Thule","Greenland","THU","BGTL",76.531203,-68.703161,251,-4,"E","America/Thule"
11,"Akureyri","Akureyri","Iceland","AEY","BIAR",65.659994,-18.072703,6,0,"N","Atlantic/Reykjavik"
12,"Egilsstadir","Egilsstadir","Iceland","EGS","BIEG",65.283333,-14.401389,76,0,"N","Atlantic/Reykjavik"
13,"Hornafjordur","Hofn","Iceland","HFN","BIHN",64.295556,-15.227222,24,0,"N","Atlantic/Reykjavik"
14,"Husavik","Husavik","Iceland","HZK","BIHU",65.952328,-17.425978,48,0,"N","Atlantic/Reykjavik"
15,"Isafjordur","Isafjordur","Iceland","IFJ","BIIS",66.058056,-23.135278,8,0,"N","Atlantic/Reykjavik"
16,"Keflavik International Airport","Keflavik","Iceland","KEF","BIKF",63.985,-22.605556,171,0,"N","Atlantic/Reykjavik"
17,"Patreksfjordur","Patreksfjordur","Iceland","PFJ","BIPA",65.555833,-23.965,11,0,"N","Atlantic/Reykjavik"
18,"Reykjavik","Reykjavik","Iceland","RKV","BIRK",64.13,-21.940556,48,0,"N","Atlantic/Reykjavik"
19,"Siglufjordur","Siglufjordur","Iceland","SIJ","BISI",66.133333,-18.916667,10,0,"N","Atlantic/Reykjavik"
20,"Vestmannaeyjar","Vestmannaeyjar","Iceland","VEY","BIVM",63.424303,-20.278875,326,0,"N","Atlantic/Reykjavik"
21,"Sault Ste Marie","Sault Sainte Marie","Canada","YAM","CYAM",46.485001,-84.509445,630,-5,"A","America/Toronto"
22,"Winnipeg St Andrews","Winnipeg","Canada","YAV","CYAV",50.056389,-97.0325,760,-6,"A","America/Winnipeg"
23,"Shearwater","Halifax","Canada","YAW","CYAW",44.639721,-63.499444,167,-4,"A","America/Halifax"
24,"St Anthony","St. Anthony","Canada","YAY","CYAY",51.391944,-56.083056,108,-3.5,"A","America/St_Johns"
25,"Tofino","Tofino","Canada","YAZ","CYAZ",49.082222,-125.7725,80,-8,"A","America/Vancouver"
26,"Kugaaruk","Pelly Bay","Canada","YBB","CYBB",68.534444,-89.808056,56,-7,"A","America/Edmonton"
27,"Baie Comeau","Baie Comeau","Canada","YBC","CYBC",49.1325,-68.204444,71,-5,"A","America/Toronto"
28,"Bagotville","Bagotville","Canada","YBG","CYBG",48.330555,-70.996391,522,-5,"A","America/Toronto"
29,"Baker Lake","Baker Lake","Canada","YBK","CYBK",64.298889,-96.077778,59,-6,"A","America/Winnipeg"
30,"Campbell River","Campbell River","Canada","YBL","CYBL",49.950832,-125.270833,346,-8,"A","America/Vancouver"
31,"Brandon Muni","Brandon","Canada","YBR","CYBR",49.91,-99.951944,1343,-6,"A","America/Winnipeg"
32,"Cambridge Bay","Cambridge Bay","Canada","YCB","CYCB",69.108055,-105.138333,90,-7,"A","America/Edmonton"
33,"Nanaimo","Nanaimo","Canada","YCD","CYCD",49.052333,-123.870167,93,-8,"A","America/Vancouver"
34,"Castlegar","Castlegar","Canada","YCG","CYCG",49.296389,-117.6325,1624,-8,"A","America/Vancouver"
```

```
Airport = namedtuple('Airport', 'code name country latitude longitude')

def get_airports(path='airports.dat'):
    """Return a generator that yields Airport objects."""

    with open(path, 'rt') as fd:
        reader = csv.reader(fd)
        for row in reader:
            name      = row[1]
            country   = row[3]
            code      = row[4]          # IATA code (eg, 'BCN' for Barcelona)
            latitude  = float(row[6]) # Decimal degrees; negative is South.
            longitude = float(row[7]) # Decimal degrees; negative is West.
            yield Airport(code, name, country, latitude, longitude)
```

```
# Make it possible to easily look up airports by IATA code.
AIRPORTS = {airport.code : airport for airport in get_airports()}


>>> AIRPORTS['VLC']
Airport(code='VLC', name='Valencia', country='Spain', latitude=39.
489314, longitude=-0.481625)
>>> AIRPORTS['PDX']
Airport(code='PDX', name='Portland Intl', country='United States',
latitude=45.588722, longitude=-122.5975)
```

2B,410,ASF,2966,KZN,2990,,0,CR2
2B,410,ASF,2966,MRV,2962,,0,CR2
2B,410,CEK,2968,KZN,2990,,0,CR2
2B,410,CEK,2968,OVB,4078,,0,CR2
2B,410,DME,4029,KZN,2990,,0,CR2
2B,410,DME,4029,NBC,6969,,0,CR2
2B,410,DME,4029,UUA,6160,,0,CR2
2B,410,EGO,6156,KGD,2952,,0,CR2
2B,410,EGO,6156,KZN,2990,,0,CR2
2B,410,GYD,2922,NBC,6969,,0,CR2
2B,410,KGD,2952,EGO,6156,,0,CR2
2B,410,KZN,2990,AER,2965,,0,CR2
2B,410,KZN,2990,ASF,2966,,0,CR2
2B,410,KZN,2990,CEK,2968,,0,CR2
2B,410,KZN,2990,DME,4029,,0,CR2
2B,410,KZN,2990,EGO,6156,,0,CR2
2B,410,KZN,2990,LED,2948,,0,CR2
2B,410,KZN,2990,SVX,2975,,0,CR2
2B,410,LED,2948,KZN,2990,,0,CR2
2B,410,LED,2948,NBC,6969,,0,CR2
2B,410,LED,2948,UUA,6160,,0,CR2
2B,410,MRV,2962,ASF,2966,,0,CR2
2B,410,NBC,6969,DME,4029,,0,CR2
2B,410,NBC,6969,GYD,2922,,0,CR2
2B,410,NBC,6969,LED,2948,,0,CR2
2B,410,NBC,6969,SVX,2975,,0,CR2

```
Flight = collections.namedtuple('Flight', 'origin destination')

def get_flights(path='flights.dat'):
    """Return a generator that yields direct Flight objects."""

    with open(path, 'rt') as fd:
        reader = csv.reader(fd)
        for row in reader:
            origin      = row[2]      # IATA code of source ...
            destination = row[4]      # ... and destination airport.
            nstops      = int(row[7]) # Number of stops; zero for direct.
            if not nstops:
                yield Flight(origin, destination)
```

```
>>> valencia = AIRPORTS['VLC']
>>> for flight in get_flights():
...     if flight.origin == valencia.code:
...         print(flight)
...
Flight(origin='VLC', destination='PMI')
Flight(origin='VLC', destination='VIE')
Flight(origin='VLC', destination='FRA')
Flight(origin='VLC', destination='CDG')
Flight(origin='VLC', destination='CMN')
Flight(origin='VLC', destination='MAD')
Flight(origin='VLC', destination='FCO')
Flight(origin='VLC', destination='PMI')
[...]
```

```
class Graph(object):
    """ A hash-table implementation of an undirected graph."""

    def __init__(self):
        # Map each node to a set of nodes connected to it
        self._neighbors = collections.defaultdict(set)

    def connect(self, node1, node2):
        self._neighbors[node1].add(node2)
        self._neighbors[node2].add(node1)

    def neighbors(self, node):
        return self._neighbors[node]
```

```
g = Graph()
g.connect('A', 'B')
g.connect('A', 'C')
g.connect('B', 'C')


>>> g.neighbors('A')
{'C', 'B'}
```

```
def get_world_graph():
    """Return a populated Graph object with real airports and routes."""

    world = Graph()
    for flight in get_flights():
        origin      = AIRPORTS[flight.origin]
        destination = AIRPORTS[flight.destination]
        world.connect(origin, destination)
    return world
```

```
>>> world = get_world_graph()
>>> valencia = AIRPORTS['VLC']
>>> for destino in world.neighbors(valencia):
...     print(destino)
...
```

```
Airport(code='STN', name='Stansted', country='United Kingdom', latitude=51.885, longitude=0.235)
Airport(code='PRG', name='Ruzyne', country='Czech Republic', latitude=50.100833, longitude=14.26)
Airport(code='LGW', name='Gatwick', country='United Kingdom', latitude=51.148056, longitude=-0.190278)
Airport(code='VIE', name='Schwechat', country='Austria', latitude=48.110278, longitude=16.569722)
Airport(code='ORY', name='Orly', country='France', latitude=48.725278, longitude=2.359444)
Airport(code='NTE', name='Nantes Atlantique', country='France', latitude=47.153189, longitude=-1.610725)
Airport(code='SCQ', name='Santiago', country='Spain', latitude=42.896333, longitude=-8.415144)
Airport(code='IEV', name='Zhuliany Intl', country='Ukraine', latitude=50.401694, longitude=30.449697)
[...]
```

```python
@staticmethod
@functools.lru_cache()
def get_price(origin, destination, euros_per_km=0.1):
    """Return the cheapest flight without stops."""

    # Haversine distance, in kilometers
    point1 = origin.latitude, origin.longitude,
    point2 = destination.latitude, destination.longitude
    distance = haversine.haversine(point1, point2)
    return distance * euros_per_km
```

```
>>> world = Graph.load()
>>> valencia = AIRPORTS['VLC']
>>> heathrow = AIRPORTS['LHR']
>>> world.get_price(valencia, heathrow)
133.30263823905867
```

```
Route = namedtuple('Stop', 'price airport')

def dijkstra(self, origin, destination):
    """Use Dijkstra's algorithm to find the cheapest path."""

    routes = Heap()
    for neighbor in self.neighbors(origin):
        price = self.get_price(origin, neighbor)
        routes.push(Route(price=price, airport=neighbor))

    visited = set()
    visited.add(origin)
```

```
while True:

    # Find the nearest yet-to-visit airport
    price, airport = routes.pop()
    if airport in visited:
        continue

    # We have arrived! Wo-hoo!
    if airport is destination:
        return price
```

```
# Tentative distances to all the unvisited neighbors
 for neighbor in self.neighbors(airport):
     if neighbor not in visited:
         # Total spent so far plus the price of getting there
         new_price = price + self.get_price(airport, neighbor)
         routes.push(Route(new_price, neighbor))

 visited.add(airport)
```

```
def dijkstra(self, origin, destination):
    """Use Dijkstra's algorithm to find the cheapest path."""

    routes = Heap()
    for neighbor in self.neighbors(origin):
        price = self.get_price(origin, neighbor)
        routes.push(Route(price=price, airport=neighbor))

    visited = set()
    visited.add(origin)

    while True:

        # Find the nearest yet-to-visit airport
        price, airport = routes.pop()
        if airport in visited:
            continue

        # We have arrived! Wo-hoo!
        if airport is destination:
            return price

        # Tentative distances to all the unvisited neighbors
        for neighbor in self.neighbors(airport):
            if neighbor not in visited:
                # Total spent so far plus the price of getting there
                new_price = price + self.get_price(airport, neighbor)
                routes.push(Route(new_price, neighbor))

        visited.add(airport)
```

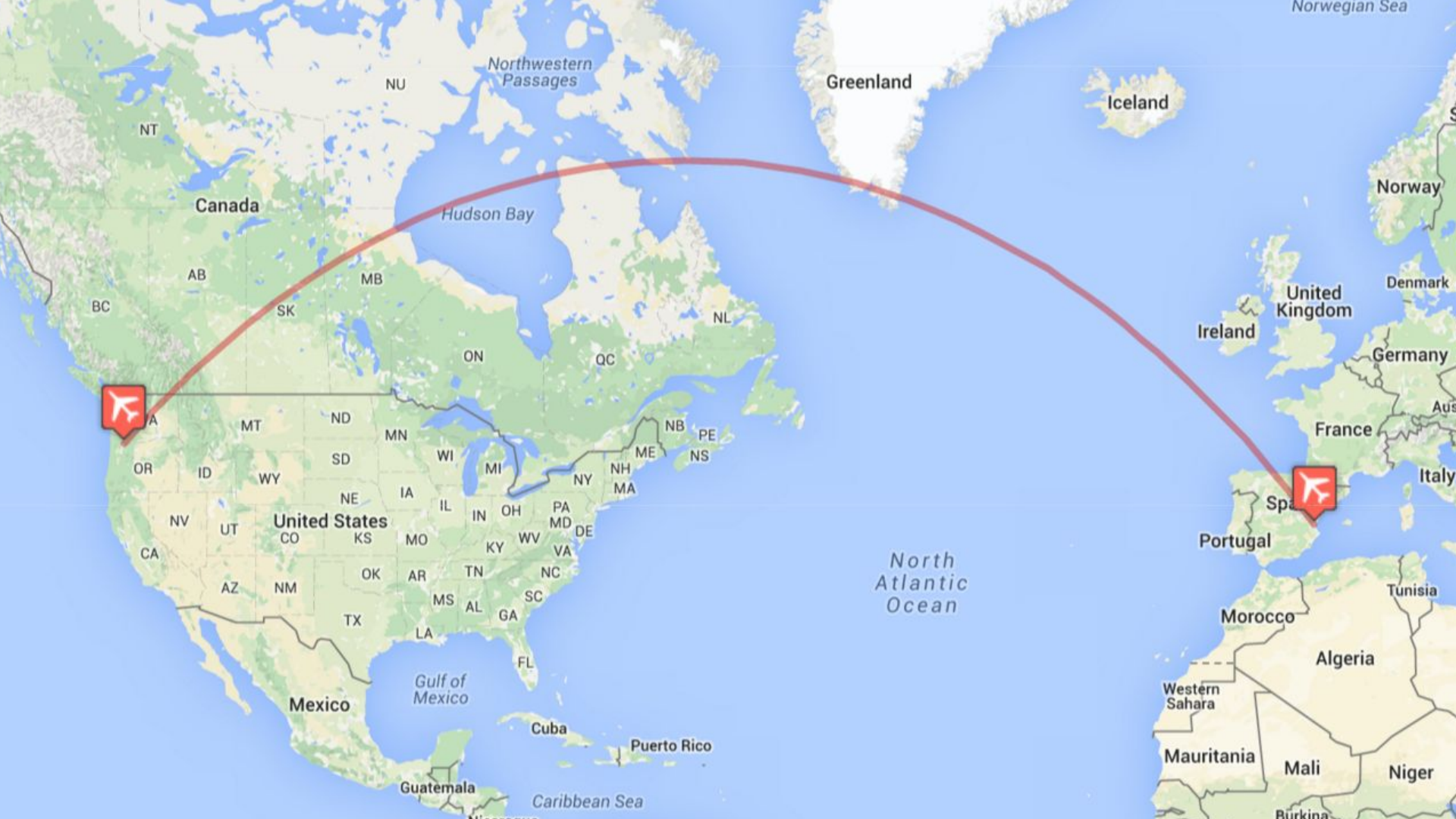
```
world = get_world_graph()
valencia = AIRPORTS['VLC']
portland = AIRPORTS['PDX']
price = world.dijkstra(valencia, portland)

>>> print(price, '€')
917.0652085335274 €
```

```
Route   = collections.namedtuple('Route'  , 'price path')

[...]

routes = Heap()
for neighbor in self.neighbors(origin):
    price = self.get_price(origin, neighbor)
    routes.push(Route(price=price, path=[neighbor]))

[...]

# Find the nearest yet-to-visit airport
price, path = routes.pop()
airport = path[-1]
if airport in visited:
    continue
```

```
# Tentative distances to all the unvisited neighbors
for neighbor in self.neighbors(airport):
    if neighbor not in visited:
        # Total spent so far plus the price of getting there
        new_price = price + self.get_price(airport, neighbor)
        new_path  = path  + [neighbor]
        routes.push(Route(new_price, new_path))
```

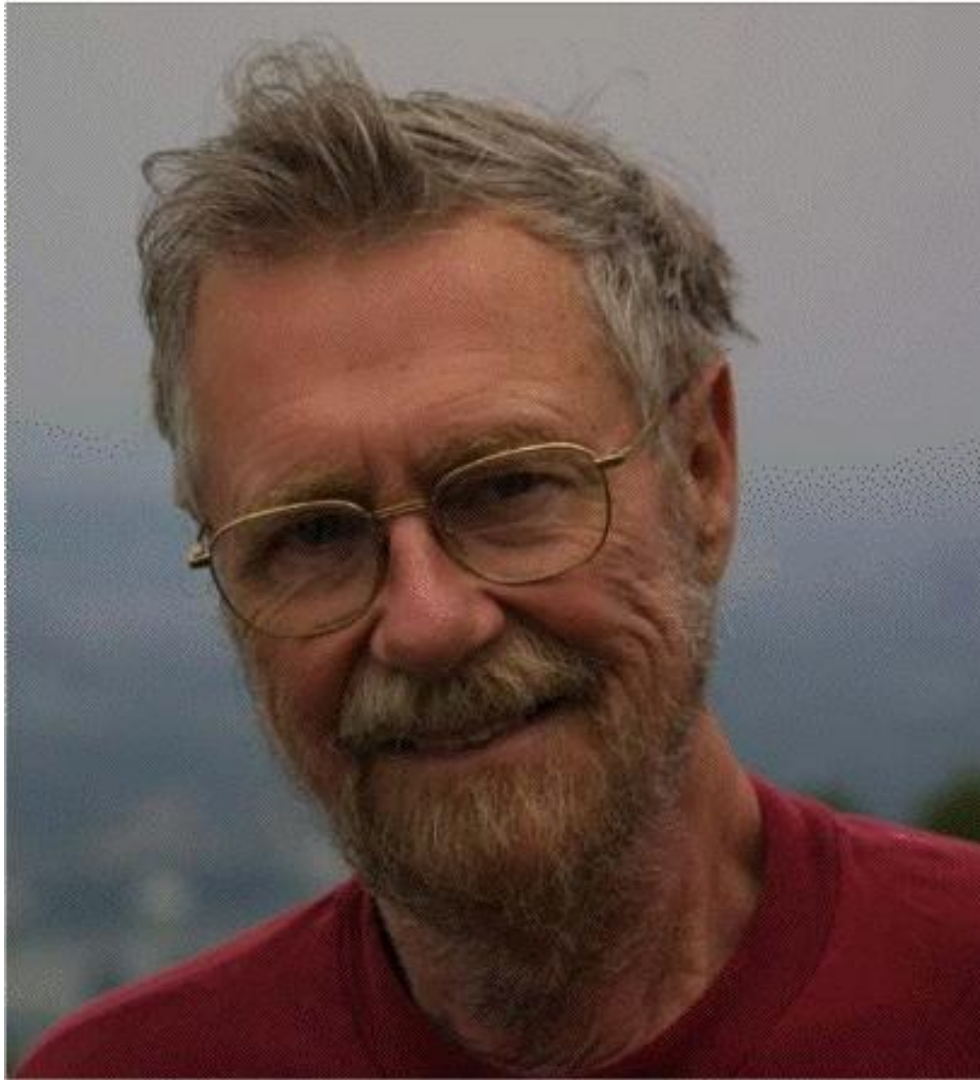
```
world = Graph.load()
valencia = AIRPORTS['VLC']
portland = AIRPORTS['PDX']
distance, path = world.dijkstra(valencia, portland)
for index, airport in enumerate(path):
    print(index, '|', airport)
print(distance, '€')
```

```
0 | Airport(code='BRS', name='Bristol', country='United Kingdom', [...]
1 | Airport(code='KEF', name='Keflavik International Airport', country='Iceland', [...]
2 | Airport(code='SEA', name='Seattle Tacoma Intl', country='United States', [...]
3 | Airport(code='PDX', name='Portland Intl', country='United States', [...]
917.0652085335274 €
```


Norwegian Sea
Northwestern Passages
NU
Greenland
Iceland
NT
Canada
Hudson Bay
Norway
AB
MB
Denmark
BC
SK
United Kingdom
NL
Ireland
ON
QC
Germany
MT
ND
MN
NB
PE
France
SD
WI
ME
NS
OR
ID
WY
MI
NH
NY
Italy
NE
IA
MA
IL
IN
OH
PA
NV
UT
United States
CO
KS
MO
MD
DE
Portugal
CA
KY
WV
VA
North Atlantic Ocean
OK
AR
TN
NC
Tunisia
AZ
NM
MS
AL
SC
Morocco
TX
GA
LA
Algeria
FL
Gulf of Mexico
Western Sahara
Mexico
Cuba
Puerto Rico
Mauritania
Mali
Niger
Guatemala
Caribbean Sea

> **Elegance is not a dispensable luxury** but a factor that decides between success and failure

— Edsger W. Dijkstra



```
>>> x = 2
>>> (x << 8) + (x << 6)
640
```

> The fastest algorithm can frequently be replaced by one that is **almost as fast and much easier to understand**

— Douglas W. Jones

Image Credits

- Shepherd in Romania, by stabmixer.
- Shaolin Monk Wallpaper.
- Ralph Wiggum, The Simpsons.
- Blade Runner, "Tears in Rain" monologue (1982)
- The Hitchhiker's Guide to the Galaxy (2005)
- Intel Devil's Canyon
- Elixir 512MB DDR RAM.
- The Creation of Adam, Michelangelo.
- IBM AS400 mainframe with console.
- Mother of God meme.
- Big-O Complexity Chart, by Eric Rowell.
- A Beautiful Mind (2001)
- Shaolin master and student.
- Yoda (Star Wars)
- Thumbs and Ammo (Pulp Fiction)
- Sisyphus, by Chewu.
- Happy Woman.
- Anaconda HD Wallpaper.
- Turtle Wallpaper.
- Barbara Liskov (Photo: Donna Coveney)
- Lightning Wallpaper.
- Walking Away From Explosion.
- The Python Logo.
- Haters Gonna Hate meme.
- Donald Knuth (Photo: Rob Becker)
- Simplicity Wallpaper, by fbailo.
- Commodore Grace M. Hopper, USN (covered)
- Yin-yang symbol.
- Edsger Wybe Dijkstra (Photo: Hamilton Richards)
- Wallpaper: boules, élégant, la réflexion, la forme Fonds d'écran.
- Princess Leia hologram (Star Wars)
- Min-heap example based on Wikipedia's example of a max-heap.
- Awesome Smiley Face Wallpaper.
- Touchscreen smartphone with GPS navigation on world map.
- The Dark Knight (2008)
- Dijkstra's algorithm example based on Wikipedia's.
- Screenshot of Wikipedia's article on Dijkstra's Algorithm.
- Screenshot of Rosetta Code's page on Dijkstra's Algorithm.
- I Don't Want to Live on This Planet Anymore meme.
- Screenshot of PyConUS 2016 website.
- Screenshot of OpenFlights.org website.
- Yuki Tsubota: Not Bad!
- Edsger W. Dijkstra.
- Douglas Jones (Photo: Tim Schoon)