Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 23: Ejercicios Examen (parte 2)

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

17 de Abril, 2015

Examen

Temario examen:

- Control de Flujo.
- Funciones.
- Strings.
- Lectura y Escritura de Archivos.
- Listas.
- Búsqueda y Ordenamiento.
- Programación Orientada a Objetos.
- Recursión.
- Backtracking.

Examen

Hoy:

- Control de Flujo.
- Funciones.
- Strings.
- Lectura y Escritura de Archivos.
- Listas.
- Búsqueda y Ordenamiento.
- Programación Orientada a Objetos.
- Recursión.
- Backtracking.

POO

Un banco te pide diseñar un programa que permita manejar las cuentas bancarias de sus clientes. Para ellos es fundamental llevar un registro de todas las transacciones realizadas por un cliente (los montos depositados y retirados). Cada cliente puede tener varias cuentas bancarias, y cada cuenta bancaria debe tener una lista de transacciones realizadas sobre ella (dinero que entra y sale).

POO

Para ello implemente las siguientes clases (con sus respectivos constructores):

Clase	Atributos	Métodos
Movimiento	- fecha - monto	
CuentaBancaria	- balance - movimientos	+ retirar(monto) + depositar(monto)
Persona	- nombre - rut - cuentas	+ agregar_cuenta() + retirar(id, monto) + depositar(id, monto) + mostrar_detalle()

POO

a) Implementa el constructor de la clase `Movimiento`, que inicialice sus atributos `monto` y `fecha`. Para la fecha utiliza la función `datetime.now()` de la librería `datetime`.

POO

a) Implementa el constructor de la clase `Movimiento`, que inicialice sus atributos `monto` y `fecha`. Para la fecha utiliza la función `datetime.now()` de la librería `datetime`.

```
1 import datetime
2
3 class Movimiento:
4   def __init__(self,monto):
5     self.fecha = datetime.datetime.now()
6     self.monto = monto
```

POO

b) Implementa la clase `CuentaBancaria`:

- Atributos:
  - `balance`: Dinero actual en la cuenta.
  - `movimientos`: Lista con los movimientos.
- Métodos:
  - `retirar(monto)`: Usuario agrega monto a su cuenta.
  - `depositar(monto)`: Usuario retira monto de su cuenta.

POO

```
8  class CuentaBancaria:
9    def __init__(self):
10     self.balance = 0
11     self.movimientos = []
12
13   def retirar(self,monto):
14     self.balance -= monto
15     self.movimientos.append(Movimiento(-monto))
16
17   def depositar(self,monto):
18     self.balance += monto
19     self.movimientos.append(Movimiento(monto))
```

POO

c) Implementa la clase `Cliente`:

- Atributos:
  - `nombre`
  - `rut`
  - `cuentas`: Lista de cuentas.
- Métodos:
  - `agregar_cuenta()`: Agrega una nueva cuenta vacía al cliente.
  - `retirar(id,monto)`: Retira `monto` de la cuenta `id`.
  - `depositar(id,monto)`: Deposita `monto` en la cuenta `id`.
  - `mostrar_detalle()`: Muestra en consola los datos del cliente (nombre y rut), y el detalle de cada una de sus cuentas (todas sus transacciones y el balance actual).

POO

```
22 class Cliente:
23   def __init__(self,nombre,rut):
24     self.nombre = nombre
25     self.rut =rut
26     self.cuentas = []
27
28   def agregar_cuenta(self):
29     self.cuentas.append(CuentaBancaria())
30
31   def retirar(self,num_cuenta, monto):
32     self.cuentas[num_cuenta].retirar(monto)
33
34   def depositar(self,num_cuenta, monto):
35     self.cuentas[num_cuenta].depositar(monto)
```

POO

```
37    def mostrar_detalle_cuentas(self):
38      print("Titular:",self.nombre)
39      print("Rut:",self.rut)
40      for i in range(len(self.cuentas)):
41        print("\nCuenta:",i,"----------")
42        for m in self.cuentas[i].movimientos:
43          print(m.fecha,m.monto)
44        print("Balance:",self.cuentas[i].balance)
```

POO

Finalmente:

- Crea el cliente Aldo Verri (RUT: 3.000.000-0).
- Agregale 3 cuentas.
- En la cuenta `0` agregar $100.000.
- En la cuenta `2` agregar $200.000.
- Retirar de todas las cuentas $10.000.000.
- Muestra el detalle del cliente luego de las transacciones realizadas.

POO

```
46 c = Cliente("Aldo Verry","3.000.000-0")
47 for i in range(3):
48   c.agregar_cuenta()
49 c.depositar(0,100000)
50 c.depositar(2,200000)
51 for i in range(3):
52   c.retirar(i,10000000)
53 c.mostrar_detalle_cuentas()
```

Recursión

Implementa una función recursiva para obtener el número máximo de una lista.

Recursión

Implementa una función recursiva para obtener el número máximo de una lista.

```
1 def maximo(l):
2   if(len(l) == 1):
3     return l[0]
4   m = maximo(l[1:])
5   if(l[0] > m):
6     return l[0]
7   else:
8     return m
```

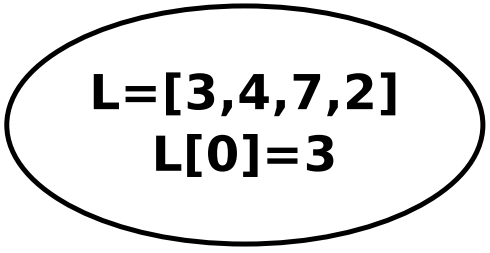
Recursión

Implementa una función recursiva para obtener el número máximo de una lista.

```
1 def maximo(l):
2   if(len(l) == 1):
3     return l[0]
4   m = maximo(l[1:])
5   if(l[0] > m):
6     return l[0]
7   else:
8     return m
```

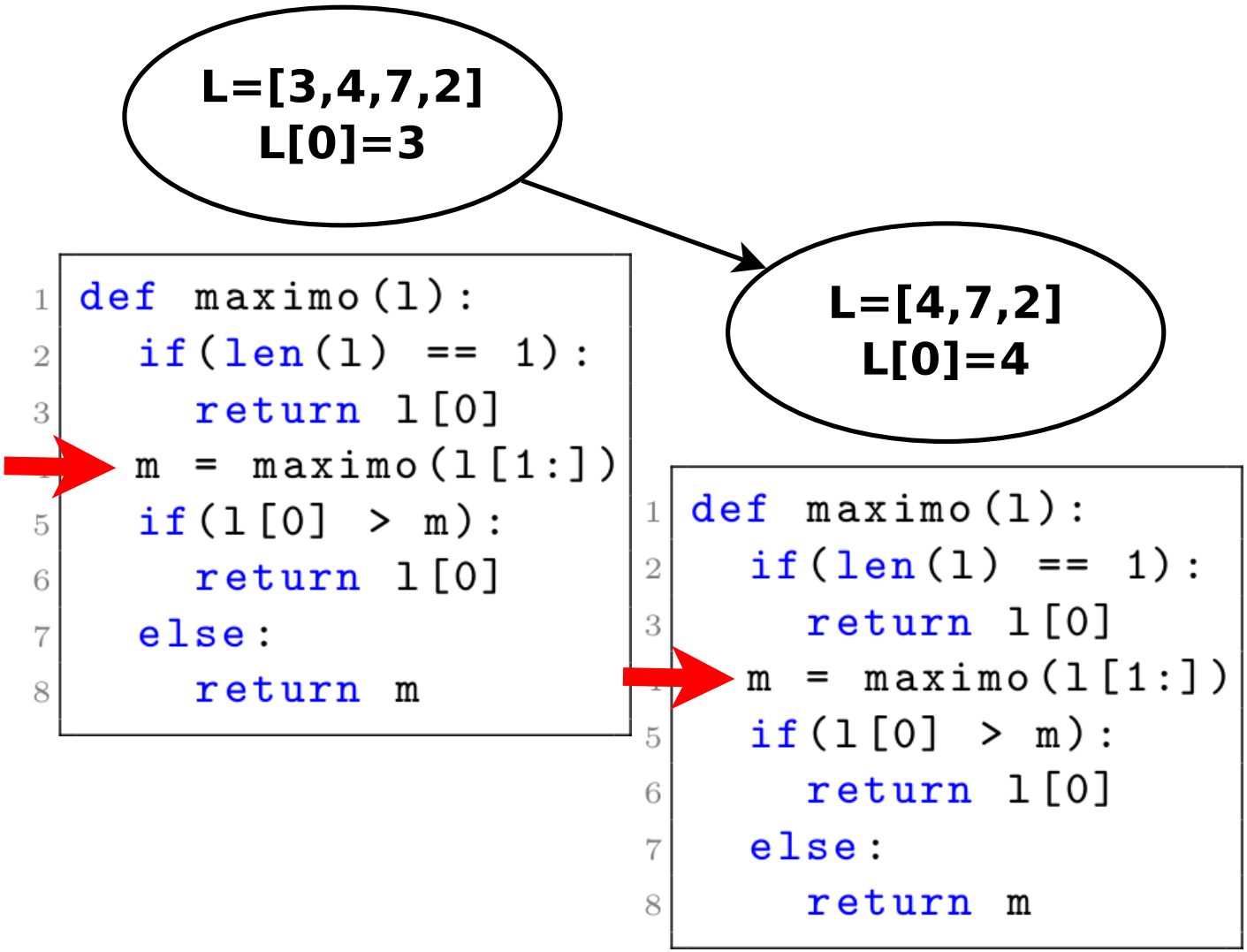
Ejemplo: `maximo([3,4,7,2])`

Recursión

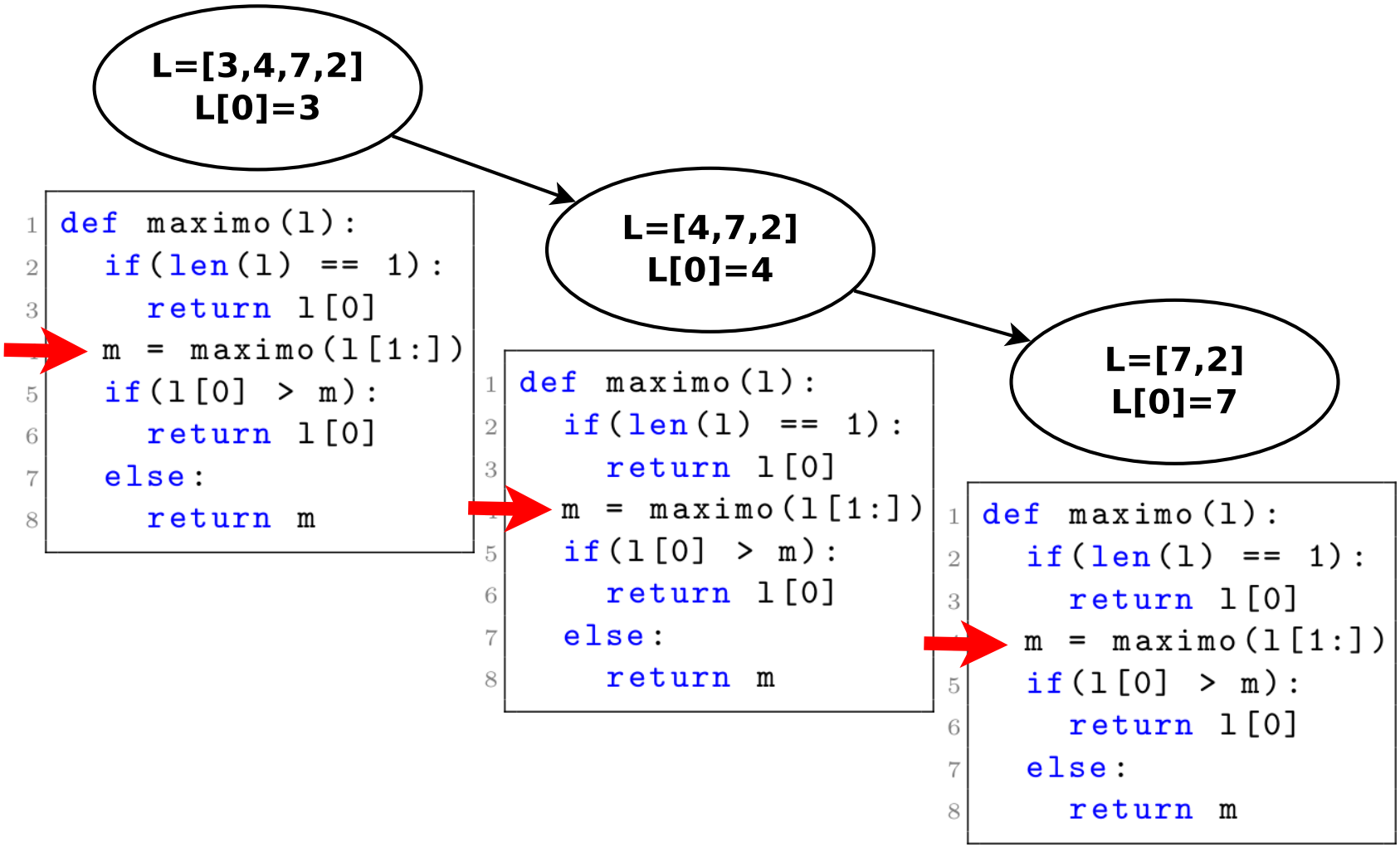



```
1  def maximo(l):
2    if(len(l) == 1):
3      return l[0]
4    m = maximo(l[1:])
5    if(l[0] > m):
6      return l[0]
7    else:
8      return m
```

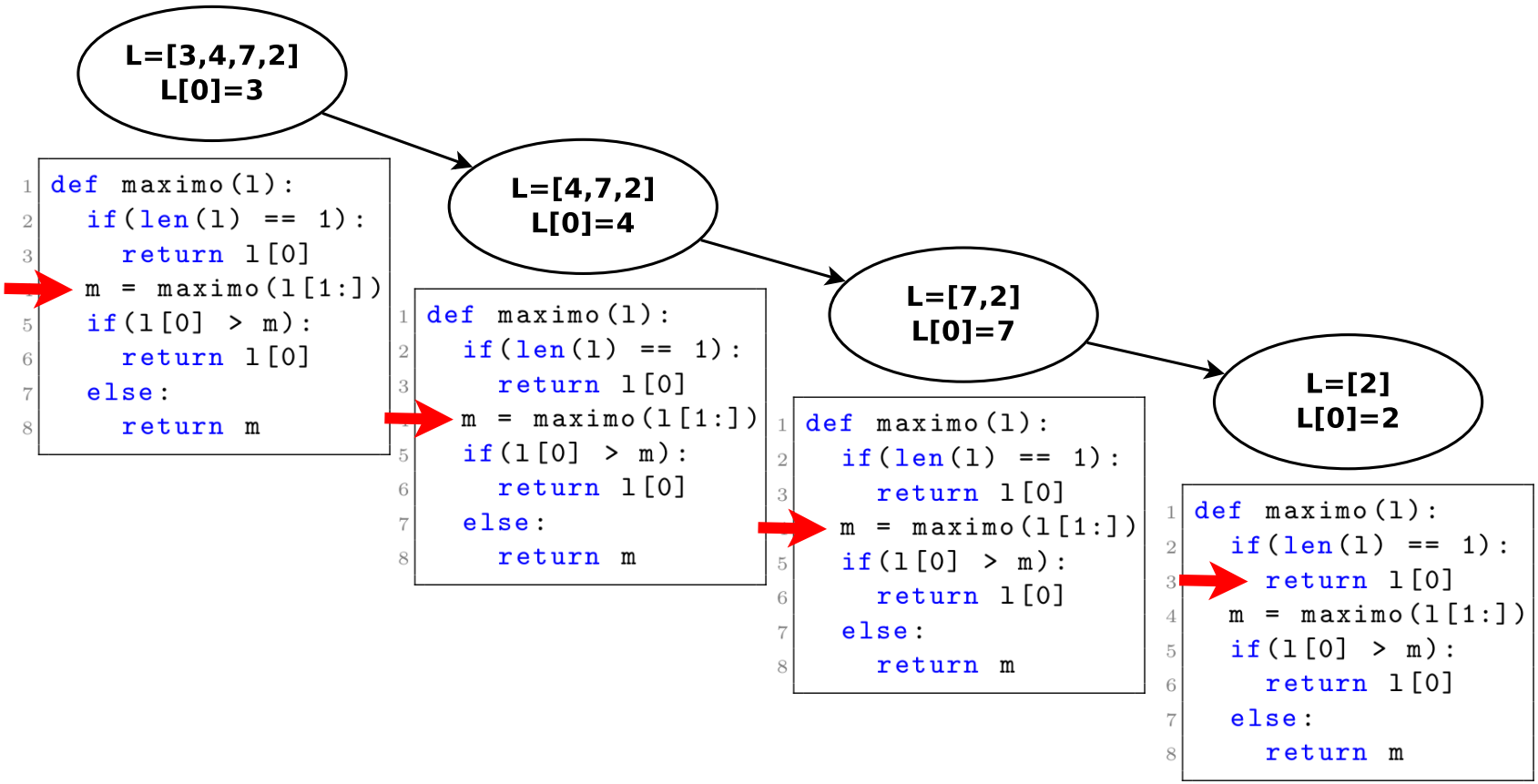
Recursión


L=[3,4,7,2]
L[0]=3
L=[4,7,2]
L[0]=4
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m

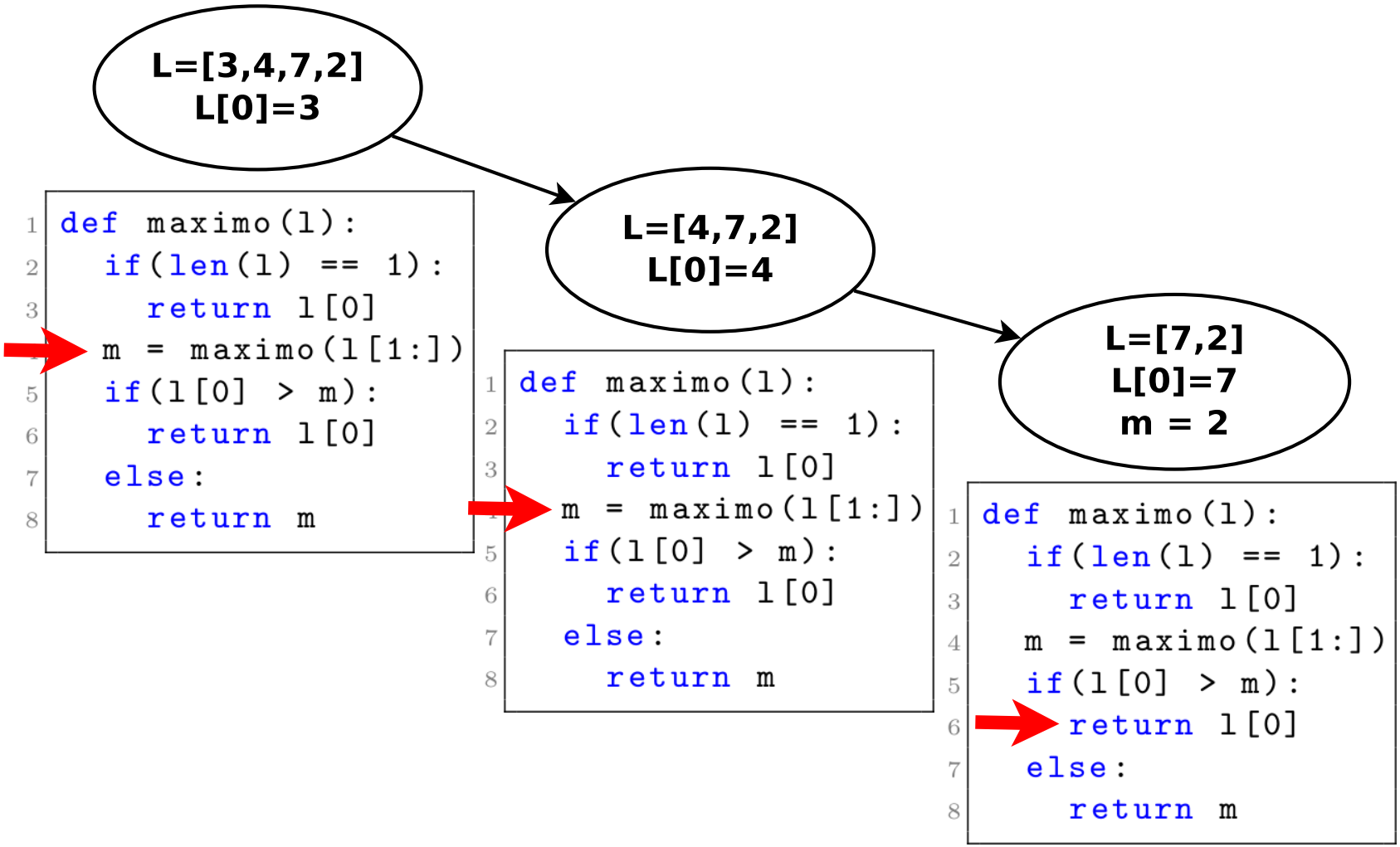
Recursión


L=[3,4,7,2]
L[0]=3
L=[4,7,2]
L[0]=4
L=[7,2]
L[0]=7
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m

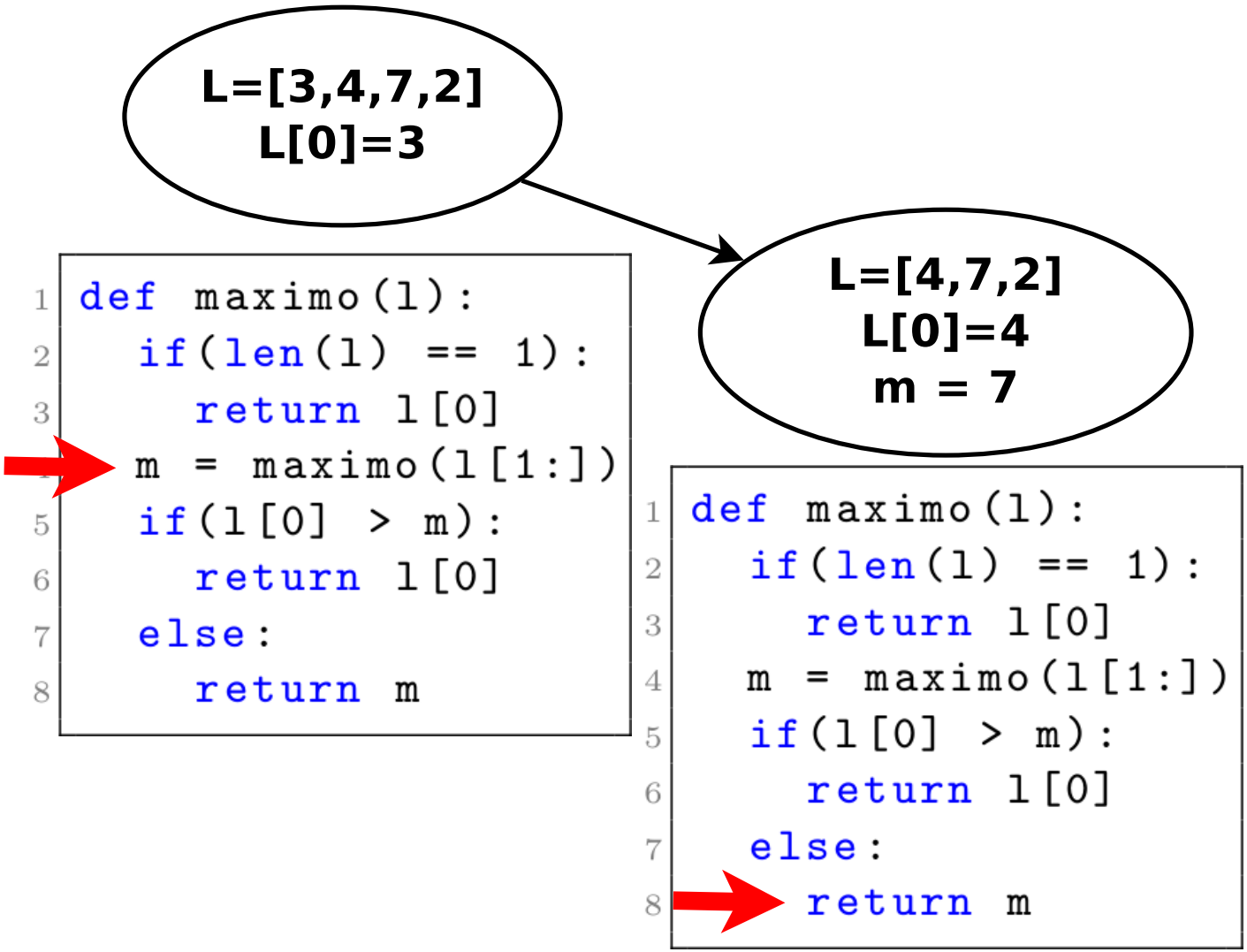
Recursión


L=[3,4,7,2]
L[0]=3
L=[4,7,2]
L[0]=4
L=[7,2]
L[0]=7
L=[2]
L[0]=2
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m

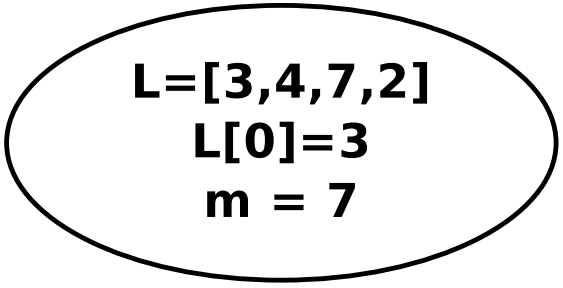
Recursión


L=[3,4,7,2]
L[0]=3
L=[4,7,2]
L[0]=4
L=[7,2]
L[0]=7
m = 2
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m
def maximo(l):
if(len(l) == 1):
return l[0]
m = maximo(l[1:])
if(l[0] > m):
return l[0]
else:
return m

Recursión


L=[3,4,7,2]
L[0]=3
L=[4,7,2]
L[0]=4
m = 7
1 def maximo(l):
2 if(len(l) == 1):
3 return l[0]
4 m = maximo(l[1:])
5 if(l[0] > m):
6 return l[0]
7 else:
8 return m
1 def maximo(l):
2 if(len(l) == 1):
3 return l[0]
4 m = maximo(l[1:])
5 if(l[0] > m):
6 return l[0]
7 else:
8 return m

Recursión




```
1  def maximo(l):
2      if(len(l) == 1):
3          return l[0]
4      m = maximo(l[1:])
5      if(l[0] > m):
6          return l[0]
7      else:
8          return m
```

Recursión

Implementa la búsqueda binaria en forma recursiva.

Input:

- `l`: Lista ordenada de números.
- `n`: Número particular.

Retorno: `True` ssi `n` se encuentra en `l`.

Recursión

Implementa la búsqueda binaria en forma recursiva.

Input:

- `l`: Lista ordenada de números.
- `n`: Número particular.

Retorno: `True` ssi `n` se encuentra en `l`.

Algoritmo:

- Ver valor central `i`.
- Si `n == l[i]`: Retornar **True**.
- Si `n < l[i]`: Buscar en mitad izquierda de `l`.
- Si `n > l[i]`: Buscar en mitad derecha de `l`.

Recursión

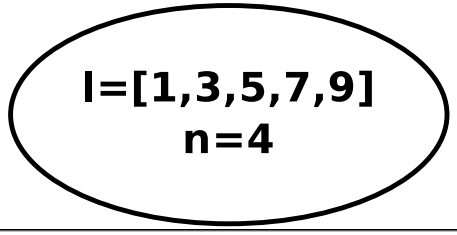
```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
10     return bb(l[i+1:],n)
```

Recursión

```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
10     return bb(l[i+1:],n)
```

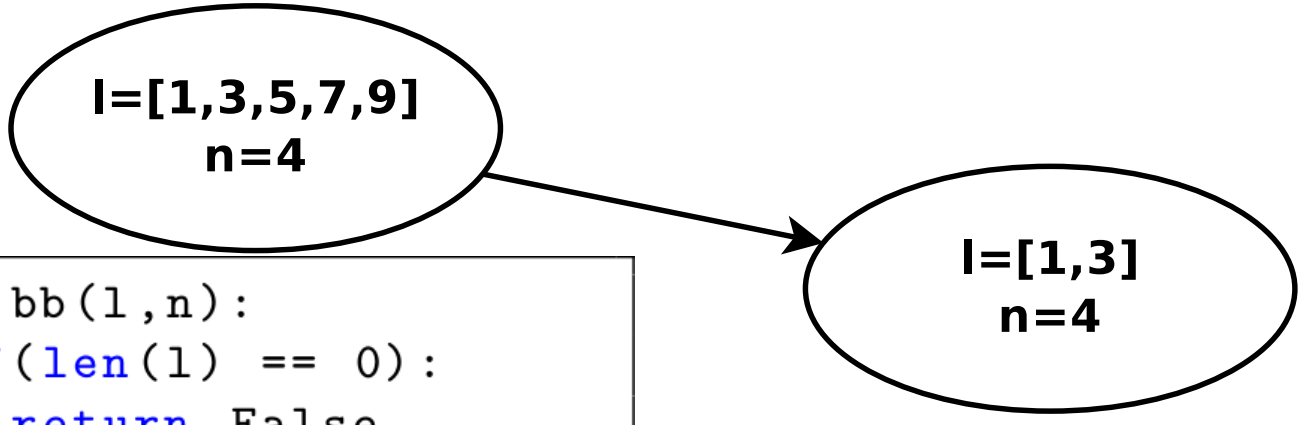
Ejemplo: `bb([1,3,5,7,9],4)`

Recursión




```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
0      return bb(l[i+1:],n)
```

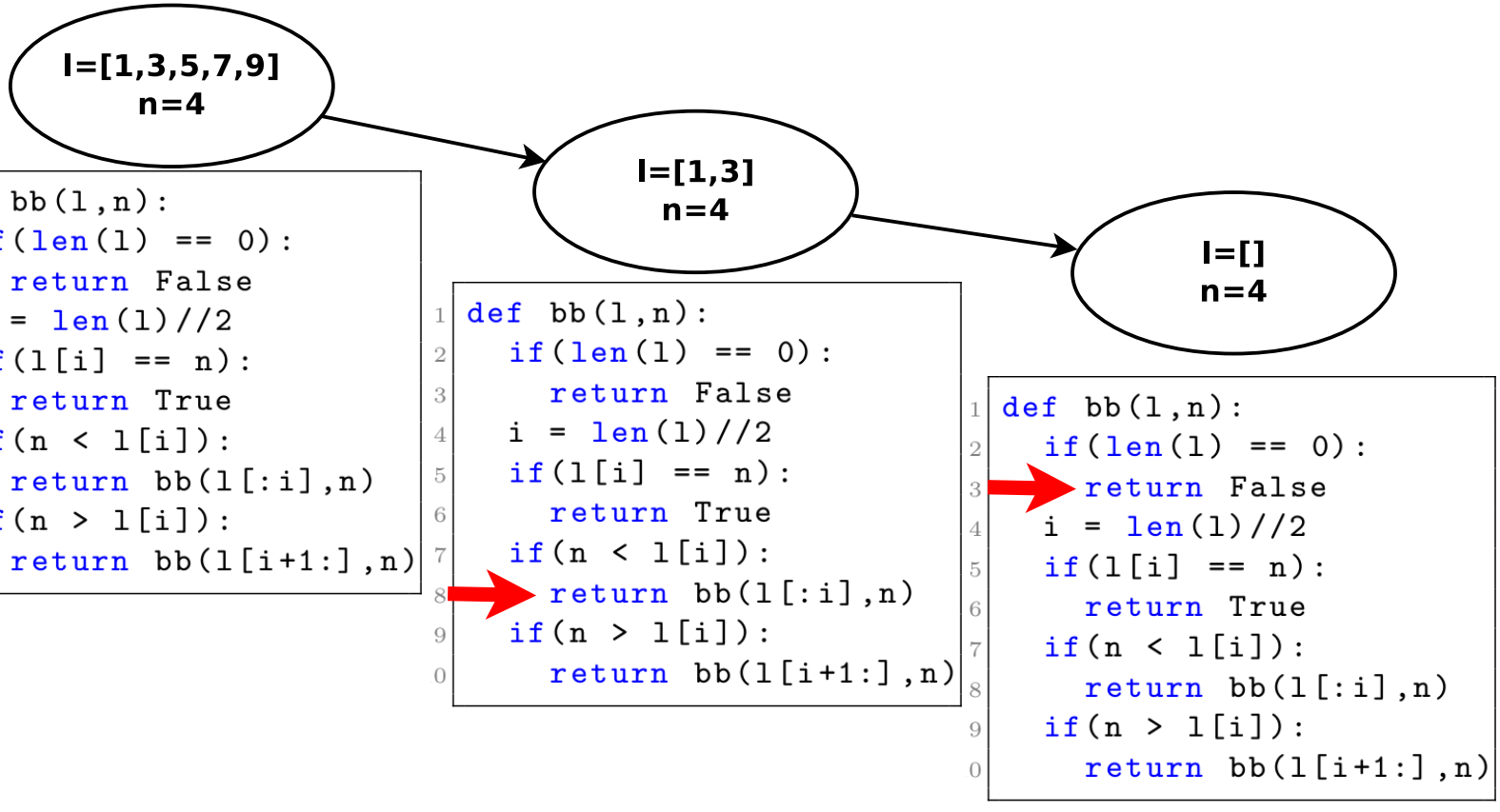
Recursión



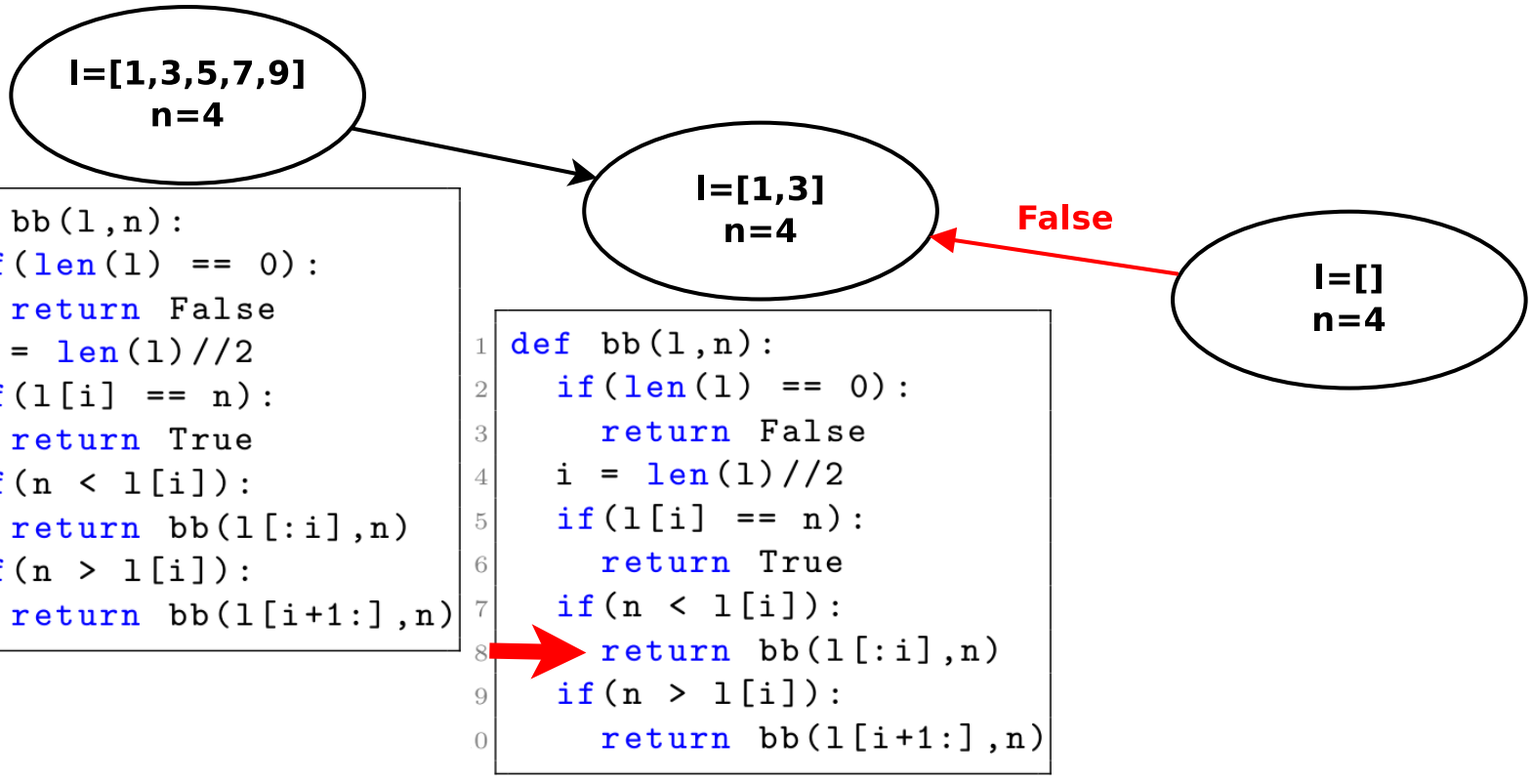

```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
0      return bb(l[i+1:],n)
```

```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
0      return bb(l[i+1:],n)
```

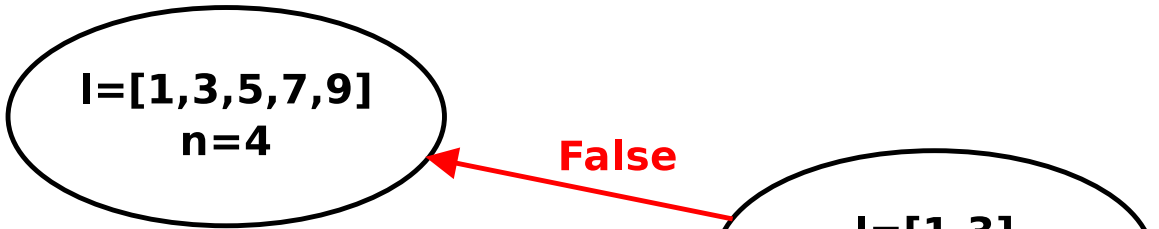
Recursión


l=[1,3,5,7,9]
n=4
l=[1,3]
n=4
l=[]
n=4
def bb(l,n):
if(len(l) == 0):
return False
i = len(l)//2
if(l[i] == n):
return True
if(n < l[i]):
return bb(l[:i],n)
if(n > l[i]):
return bb(l[i+1:],n)

Recursión

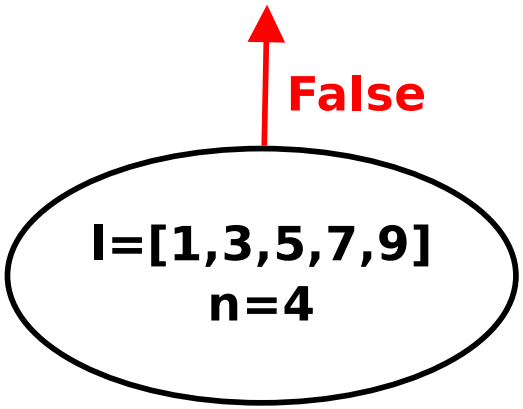

l=[1,3,5,7,9]
n=4
l=[1,3]
n=4
False
l=[]
n=4
def bb(l,n):
if(len(l) == 0):
return False
i = len(l)//2
if(l[i] == n):
return True
if(n < l[i]):
return bb(l[:i],n)
if(n > l[i]):
return bb(l[i+1:],n)
def bb(l,n):
if(len(l) == 0):
return False
i = len(l)//2
if(l[i] == n):
return True
if(n < l[i]):
return bb(l[:i],n)
if(n > l[i]):
return bb(l[i+1:],n)

Recursión




```
1  def bb(l,n):
2    if(len(l) == 0):
3      return False
4    i = len(l)//2
5    if(l[i] == n):
6      return True
7    if(n < l[i]):
8      return bb(l[:i],n)
9    if(n > l[i]):
0      return bb(l[i+1:],n)
```

Recursión



Recursión

Problema de la mochila 0/1

Se tiene una mochila que aguanta un peso máximo `v` en ella, y una lista de objetos con pesos y beneficios conocidos. Encontrar la combinación de objetos que debería meter en la mochila tal que la suma de sus beneficios sea máxima, y la suma de sus pesos sea $\leq$ `v`.

Recursión

Ejemplo:

- `L = [[1,2],[1.5,1],[0.5,3]]`, donde L[i][0] es el peso del objeto `L[i]`, y L[i][1] es su beneficio.
- `v = 2`.

Opciones:

Items	Peso	Beneficio
0	1	2
1	1.5	1
2	0.5	3
1, 2	2	4
0, 2	1.5	5

Recursión

Idea solución: En cada paso decido si llevo o no el item cero de la lista.

Recursión

Idea solución: En cada paso decido si llevo o no el item cero de la lista.

Caso base: Lista vacía o volumen negativo de la mochila.

Recursión

Idea solución: En cada paso decido si llevo o no el item cero de la lista.

Caso base: Lista vacía o volumen negativo de la mochila.
Llamado recursivo: Retornar máximo entre llevar L[0] y no llevarlo.

Recursión

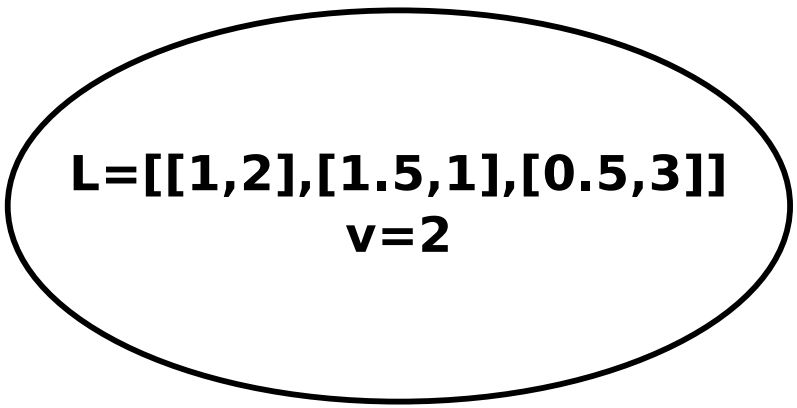
Idea solución: En cada paso decido si llevo o no el item cero de la lista.

Caso base: Lista vacía o volumen negativo de la mochila.
Llamado recursivo: Retornar máximo entre llevar L[0] y no llevarlo.

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

Recursión

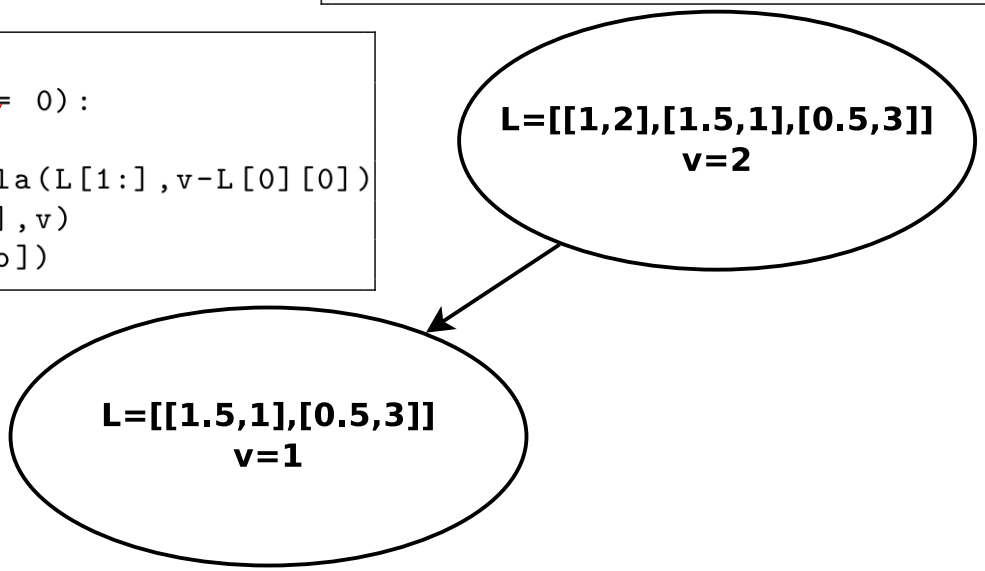
```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



Recursión

```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2

```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=1

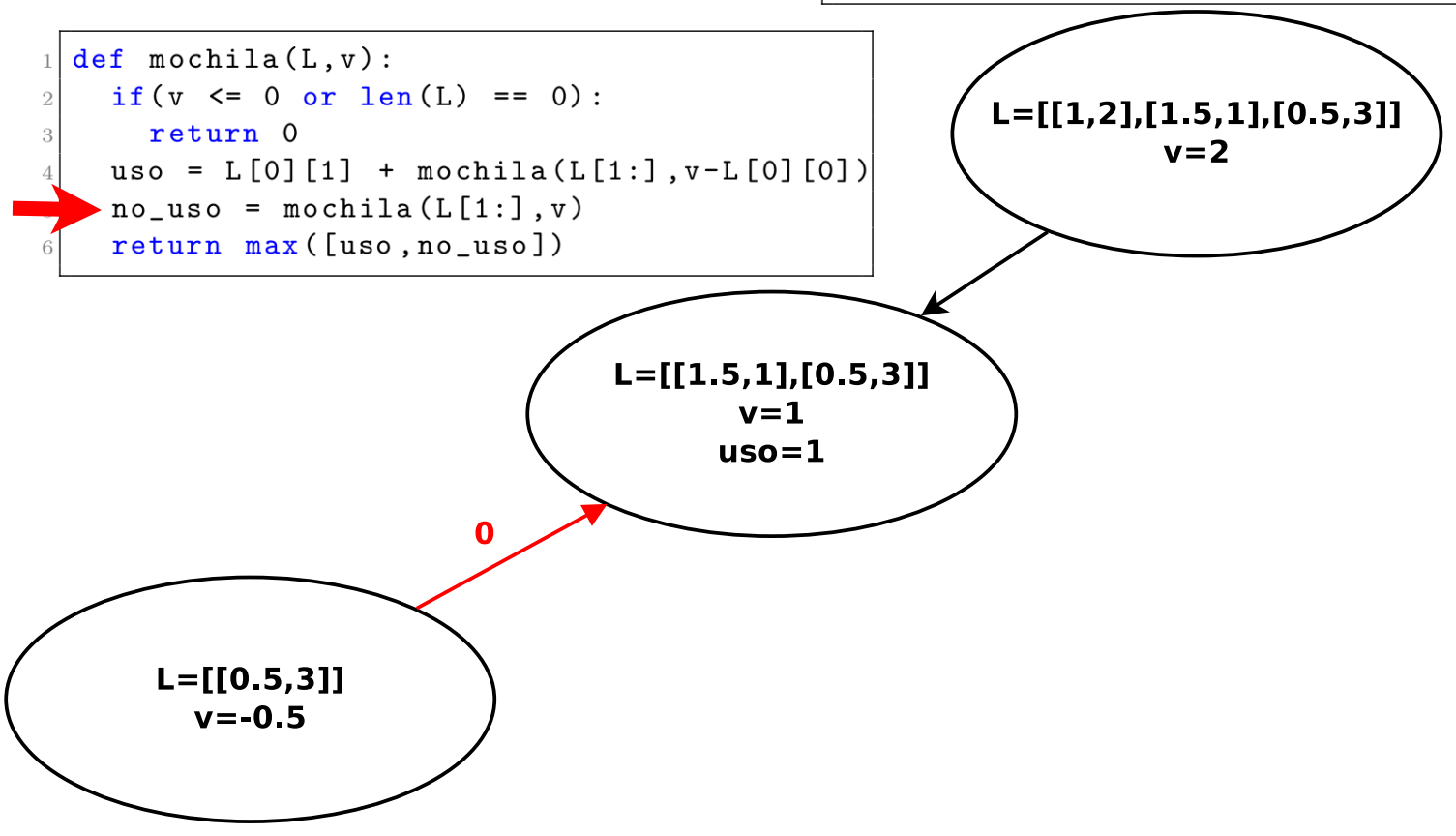
```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```

L=[[0.5,3]]
v=-0.5

Recursión

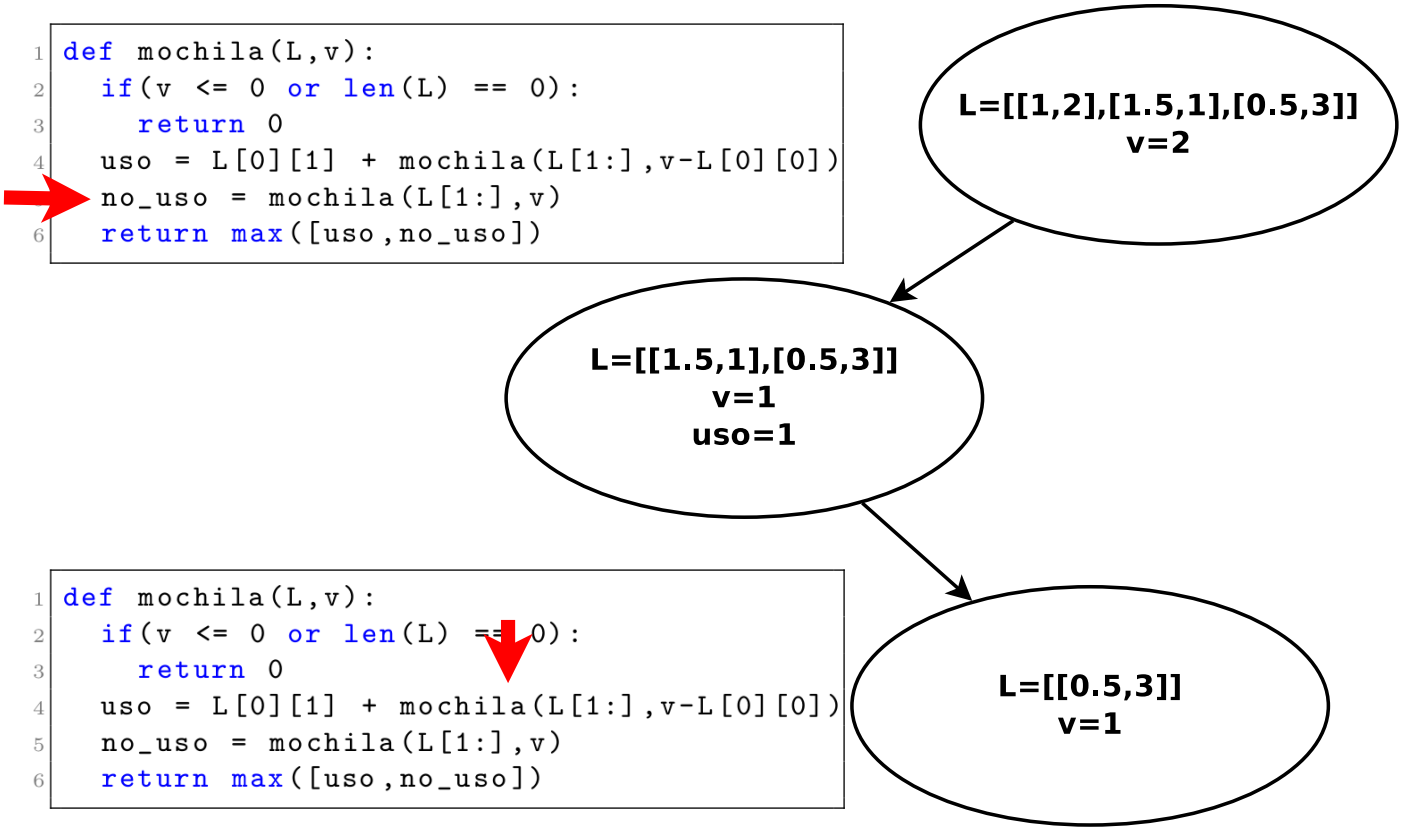
```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



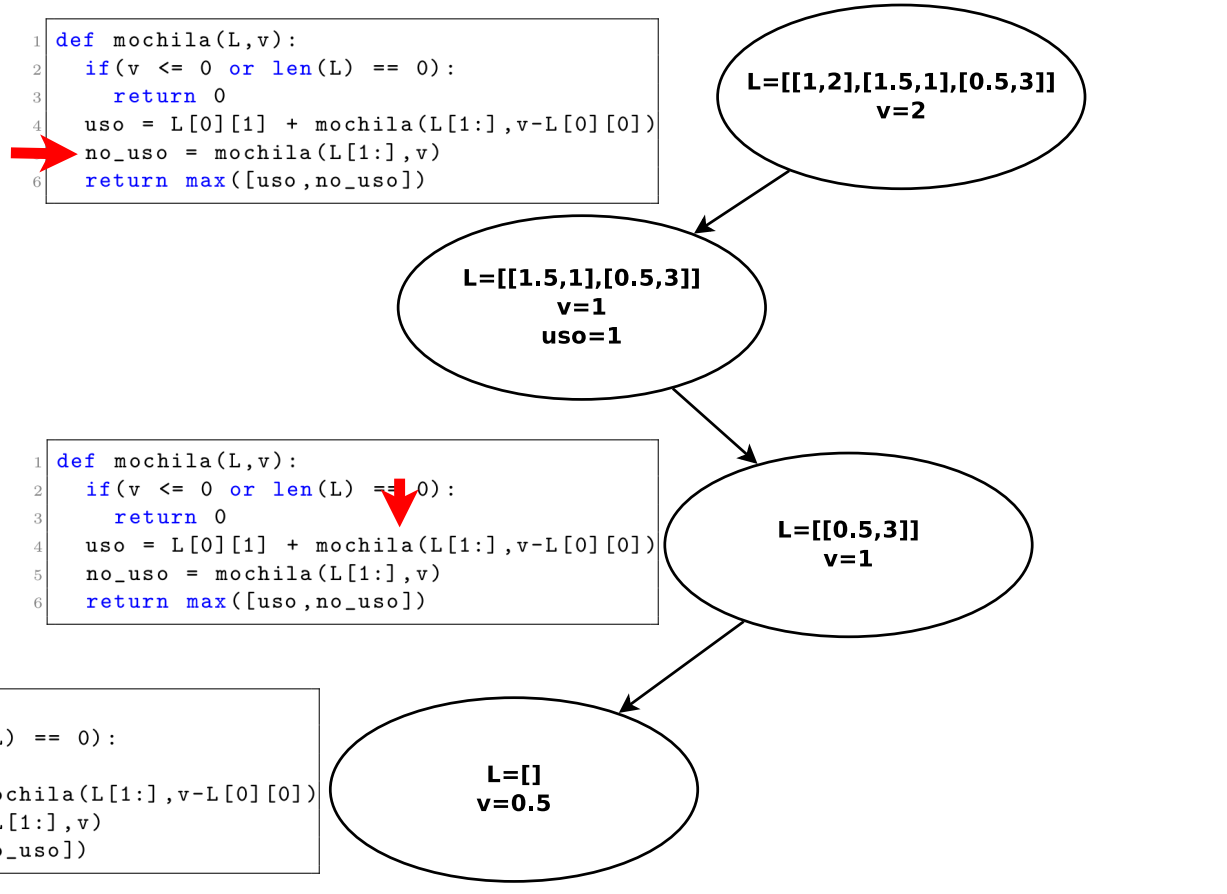
Recursión

```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```



Recursión

```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```



Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2

L=[[1.5,1],[0.5,3]]
v=1
uso=1

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

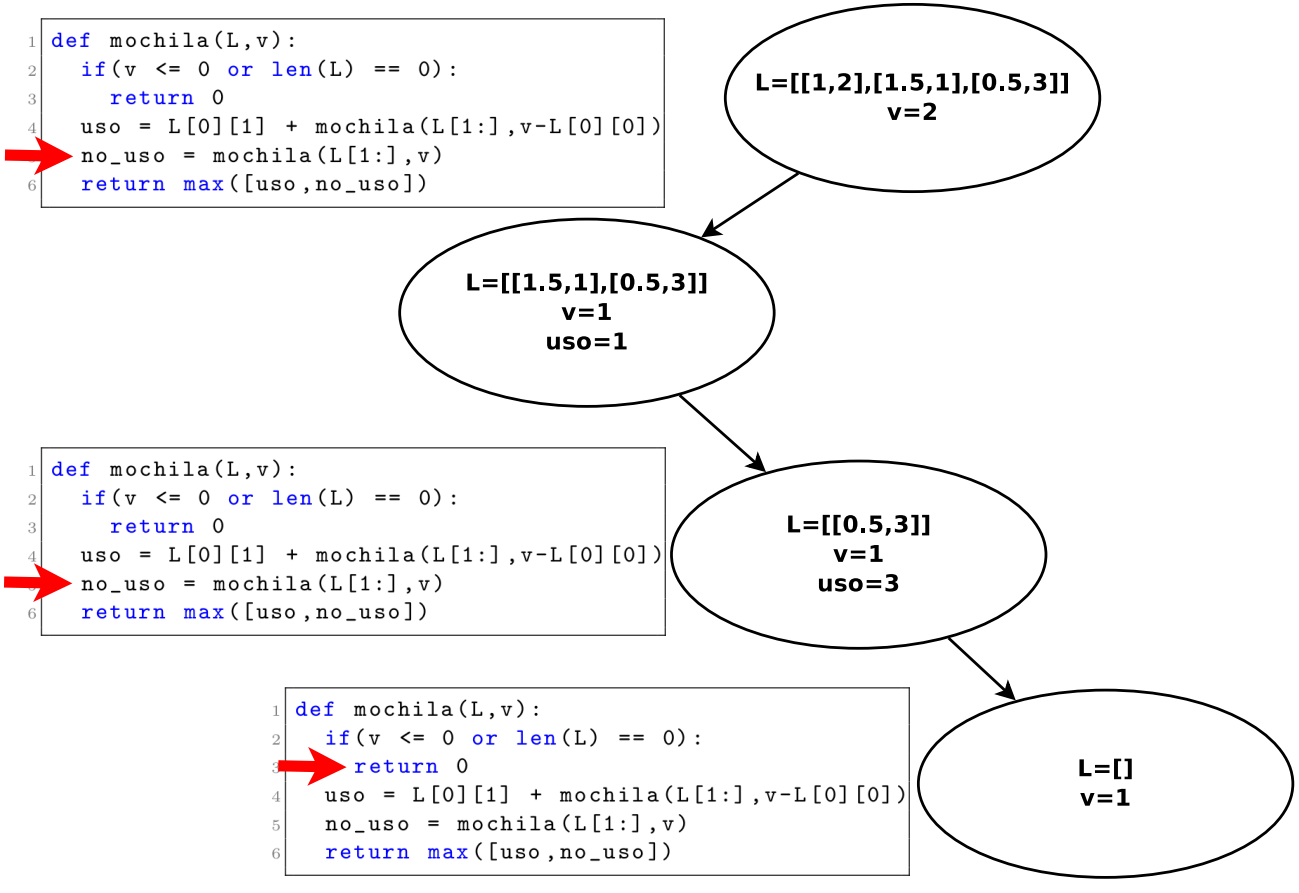
L=[[0.5,3]]
v=1
uso=3

0

L=[]
v=0.5

Recursión

```python
def mochila(L,v):
  if(v <= 0 or len(L) == 0):
    return 0
  uso = L[0][1] + mochila(L[1:],v-L[0][0])
  no_uso = mochila(L[1:],v)
  return max([uso,no_uso])
```



Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2

L=[[1.5,1],[0.5,3]]
v=1
uso=1

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[0.5,3]]
v=1
uso=3
no_uso=0

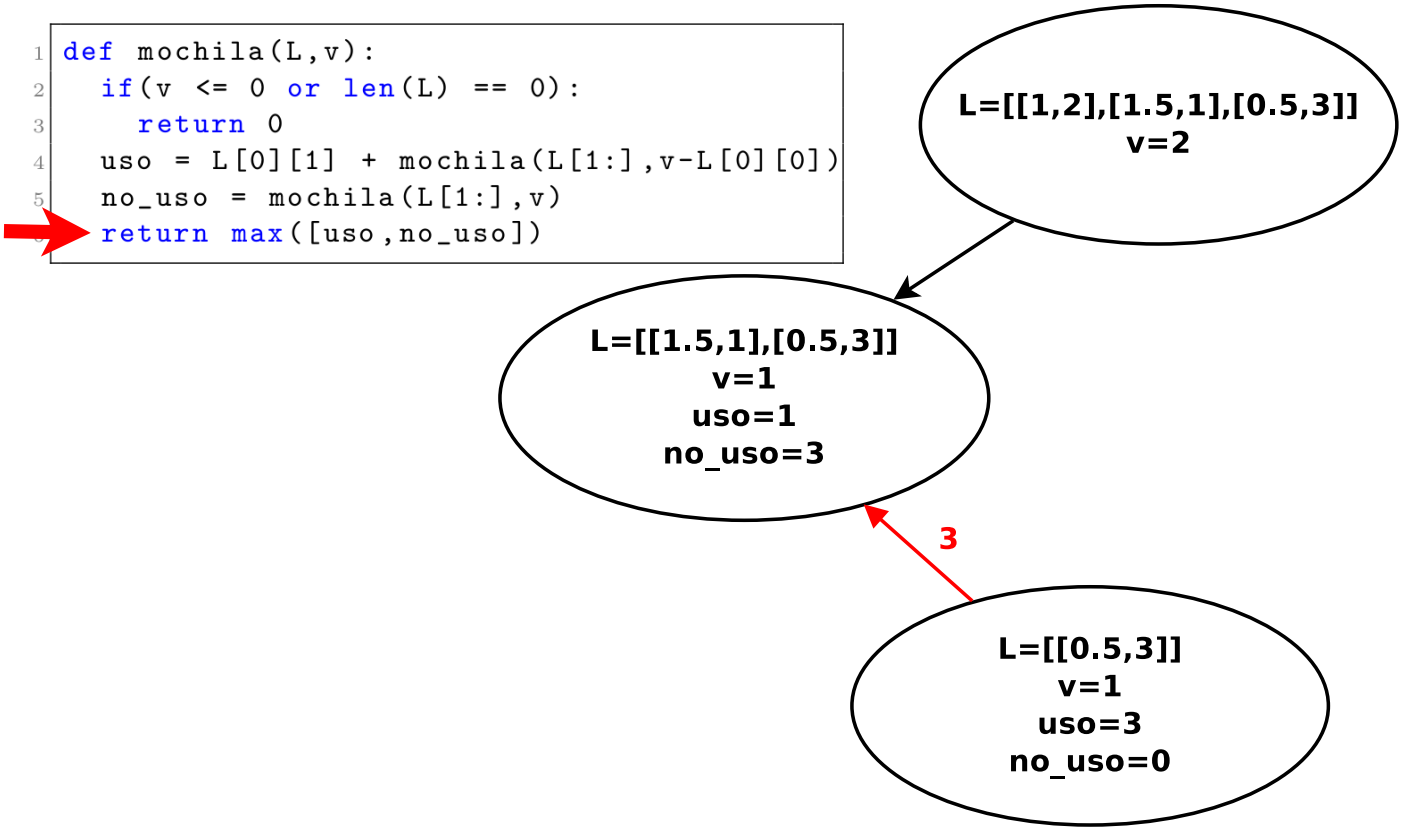
0

L=[]
v=1

Recursión

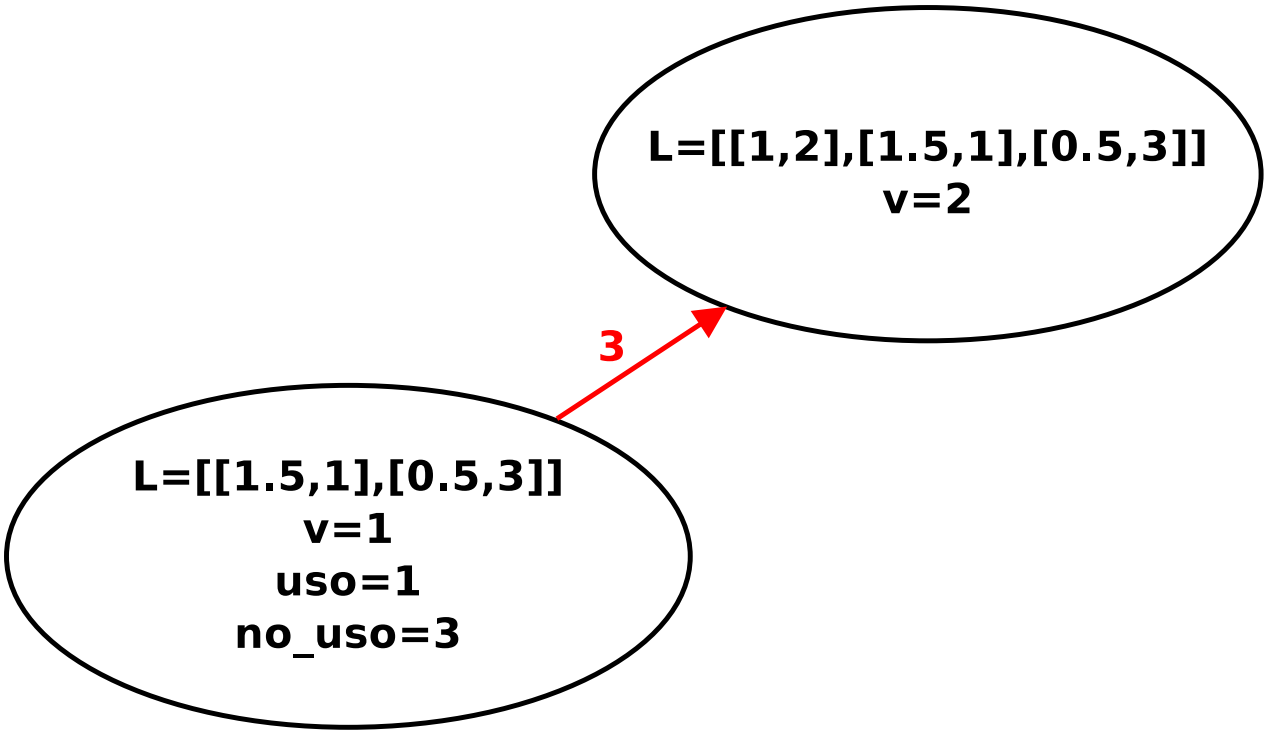
```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



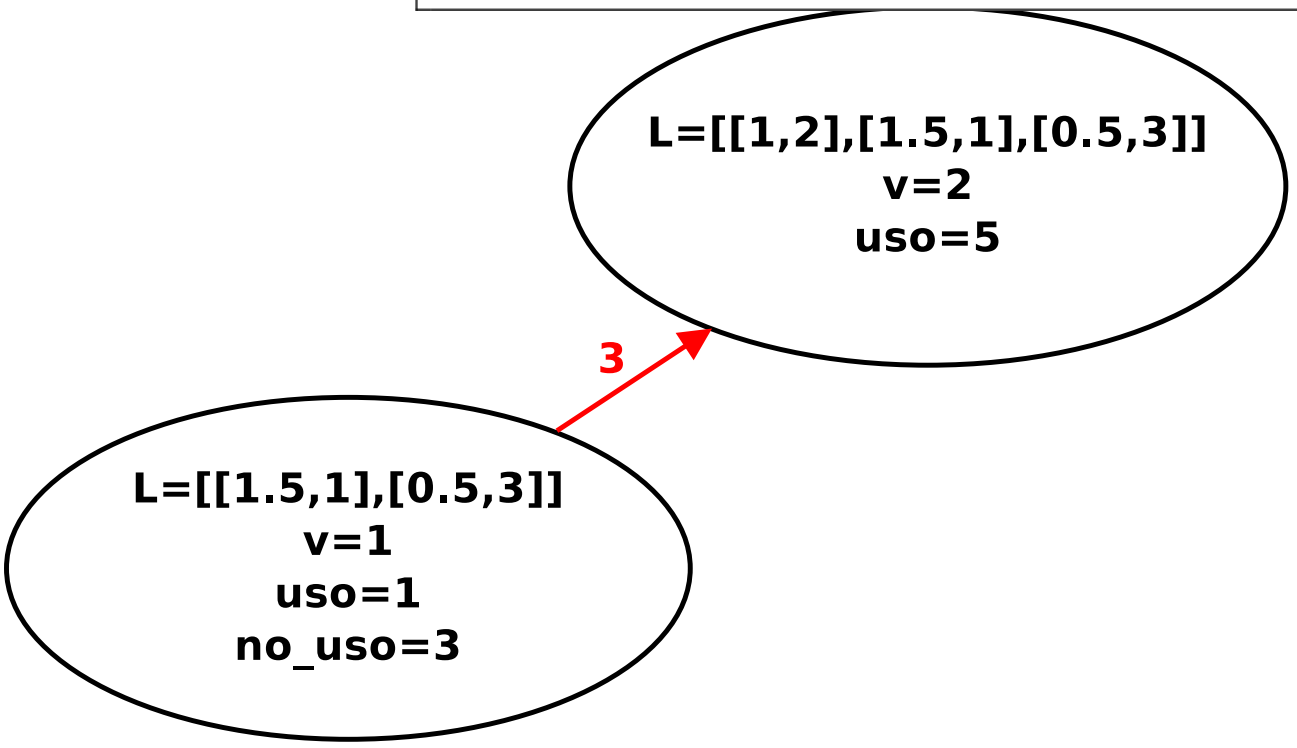
Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



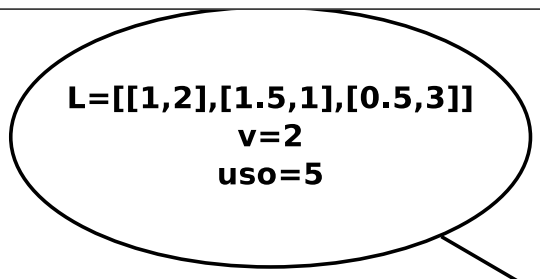
Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

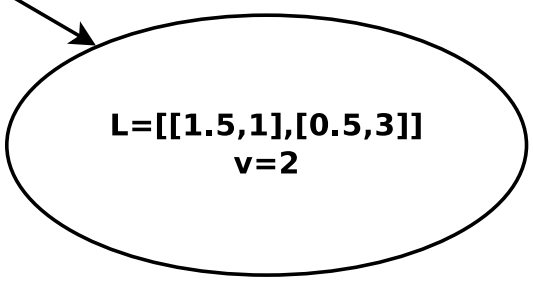


Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```




```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5

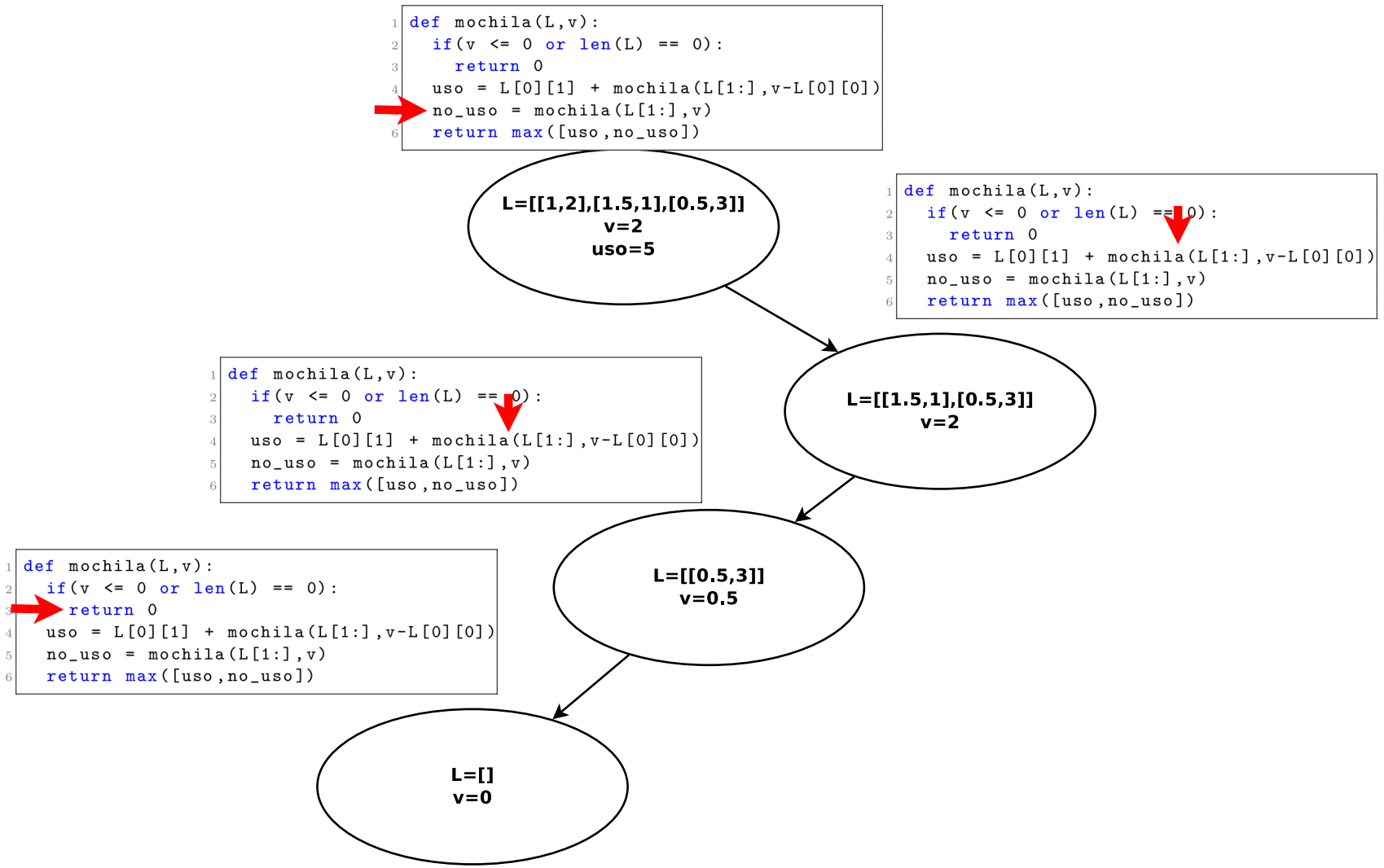
```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=2

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

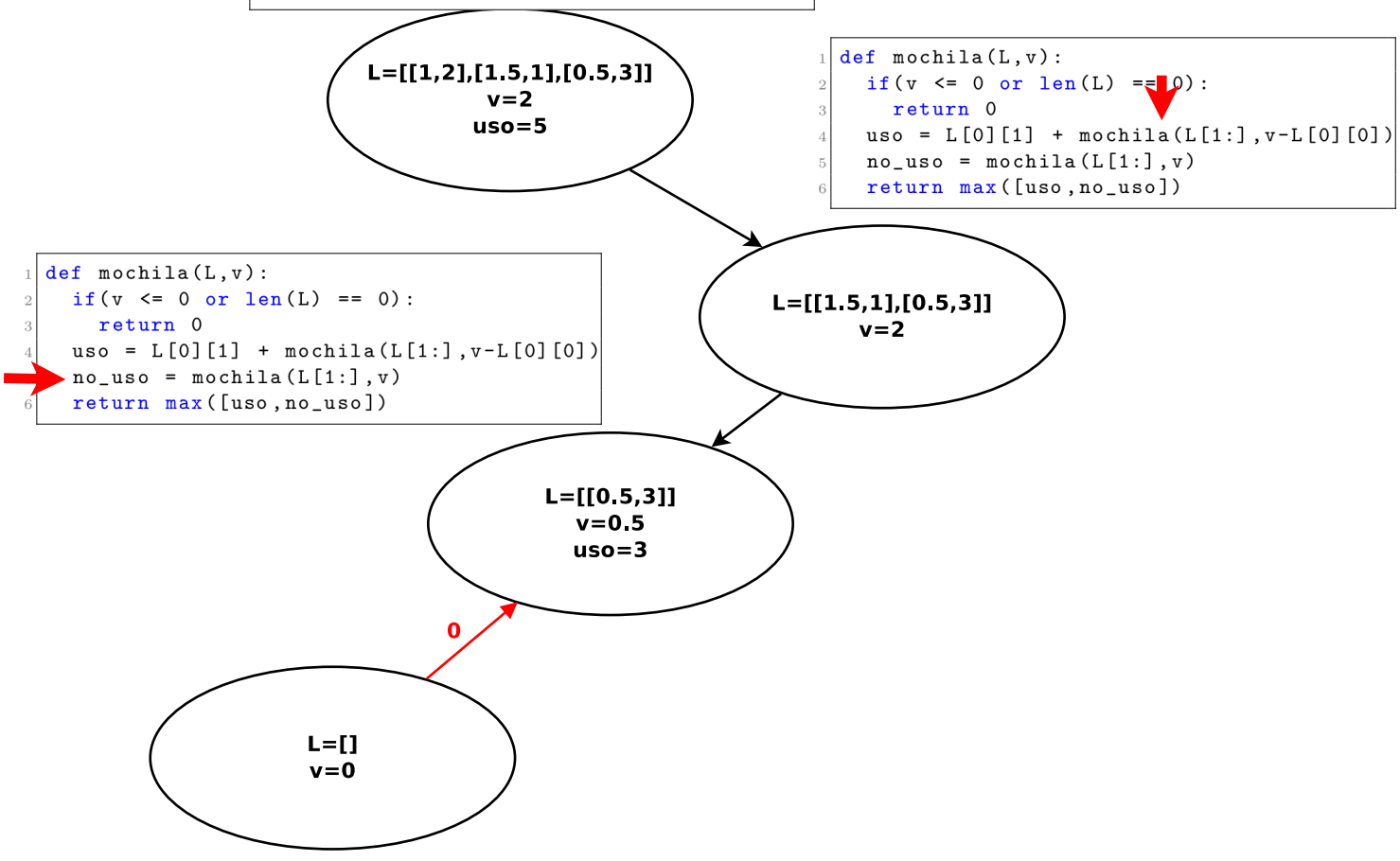
L=[[0.5,3]]
v=0.5

Recursión

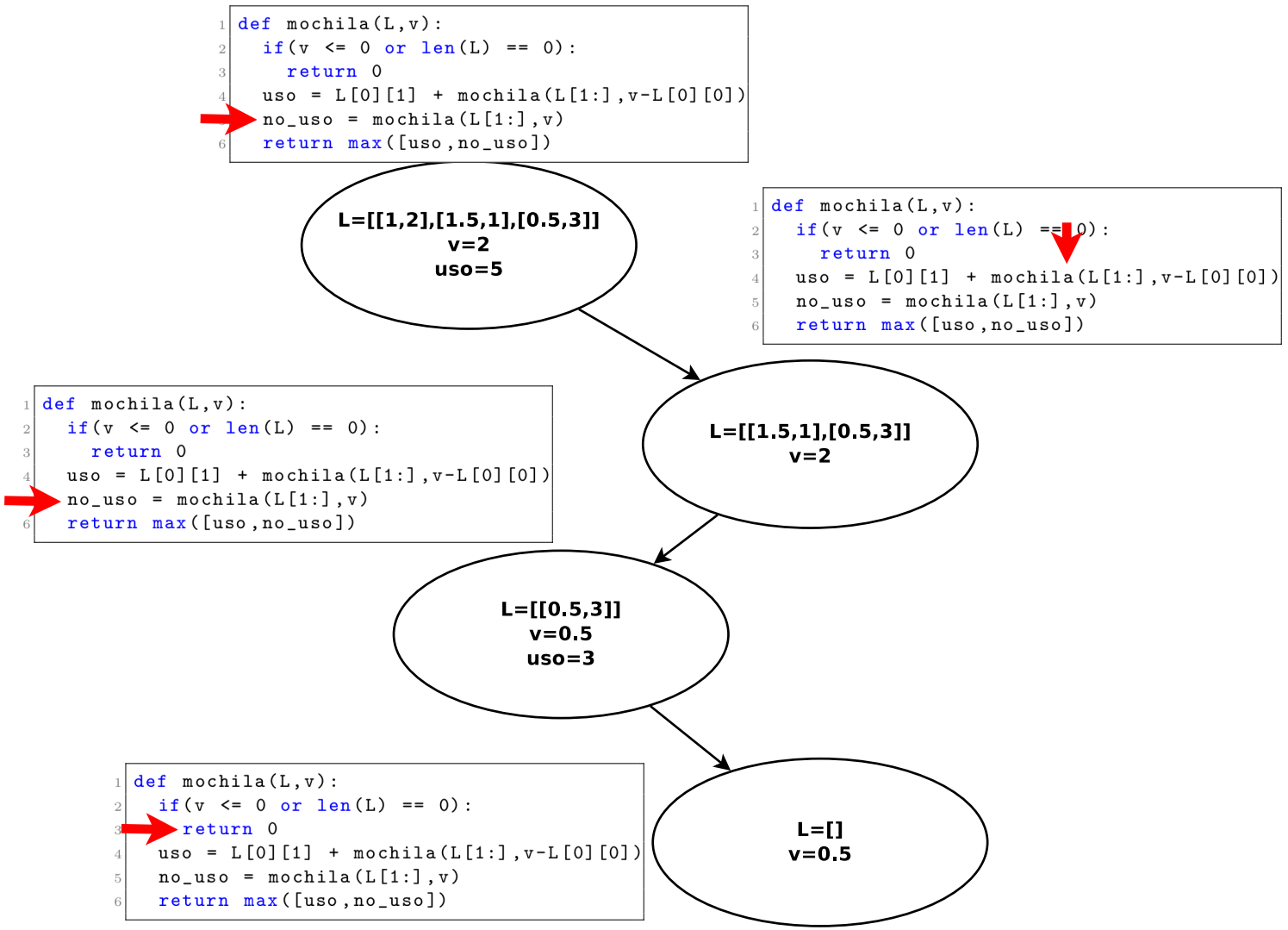

def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5
L=[[1.5,1],[0.5,3]]
v=2
L=[[0.5,3]]
v=0.5
L=[]
v=0

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



Recursión


def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5
L=[[1.5,1],[0.5,3]]
v=2
L=[[0.5,3]]
v=0.5
uso=3
L=[]
v=0.5

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=2

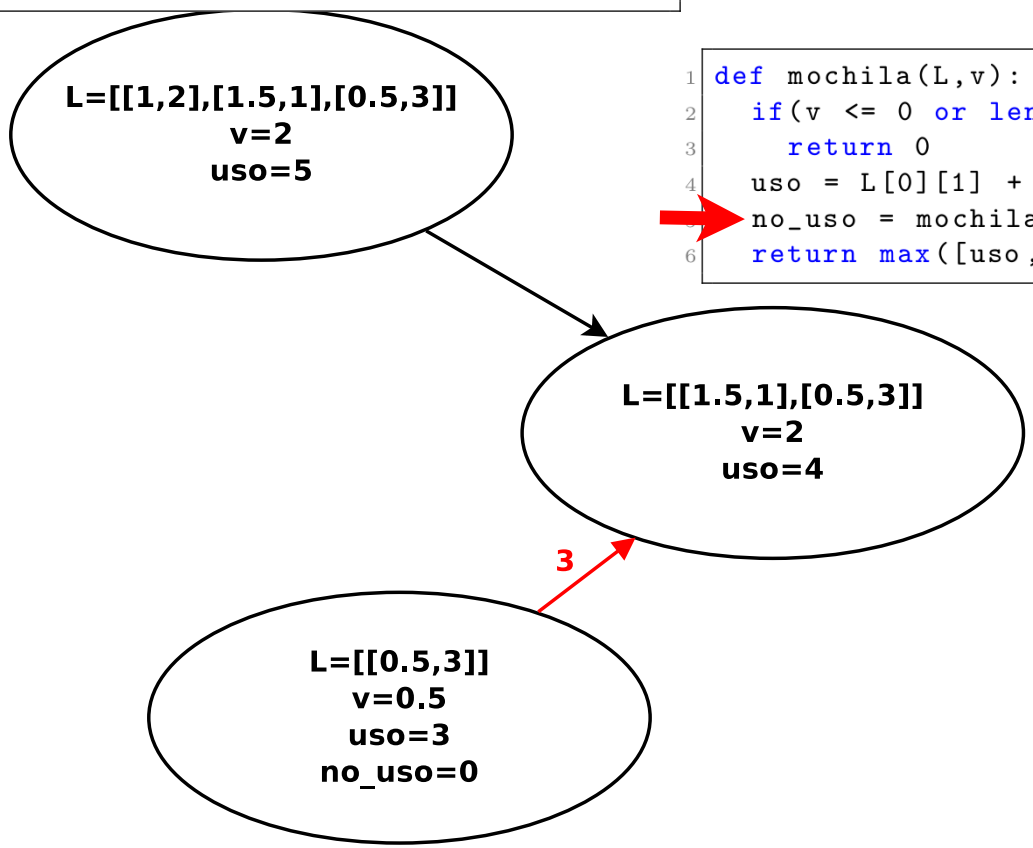
L=[[0.5,3]]
v=0.5
uso=3
no_uso=0

0

L=[]
v=0.5

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```




```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=2
uso=4

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[0.5,3]]
v=2

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=2
uso=4

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[0.5,3]]
v=2

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[]
v=1.5

Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[1.5,1],[0.5,3]]
v=2
uso=4

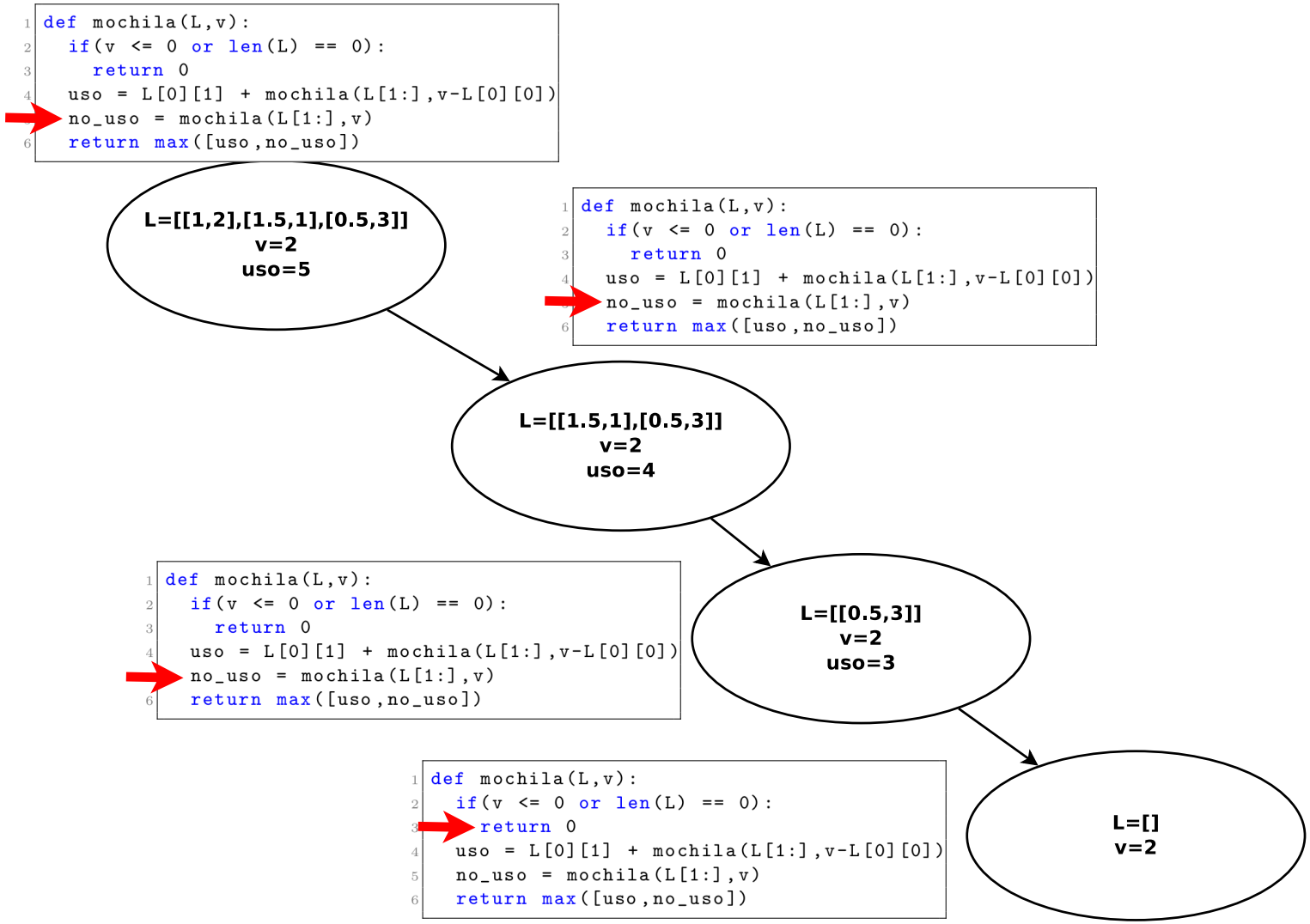
```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

L=[[0.5,3]]
v=2
uso=3

0

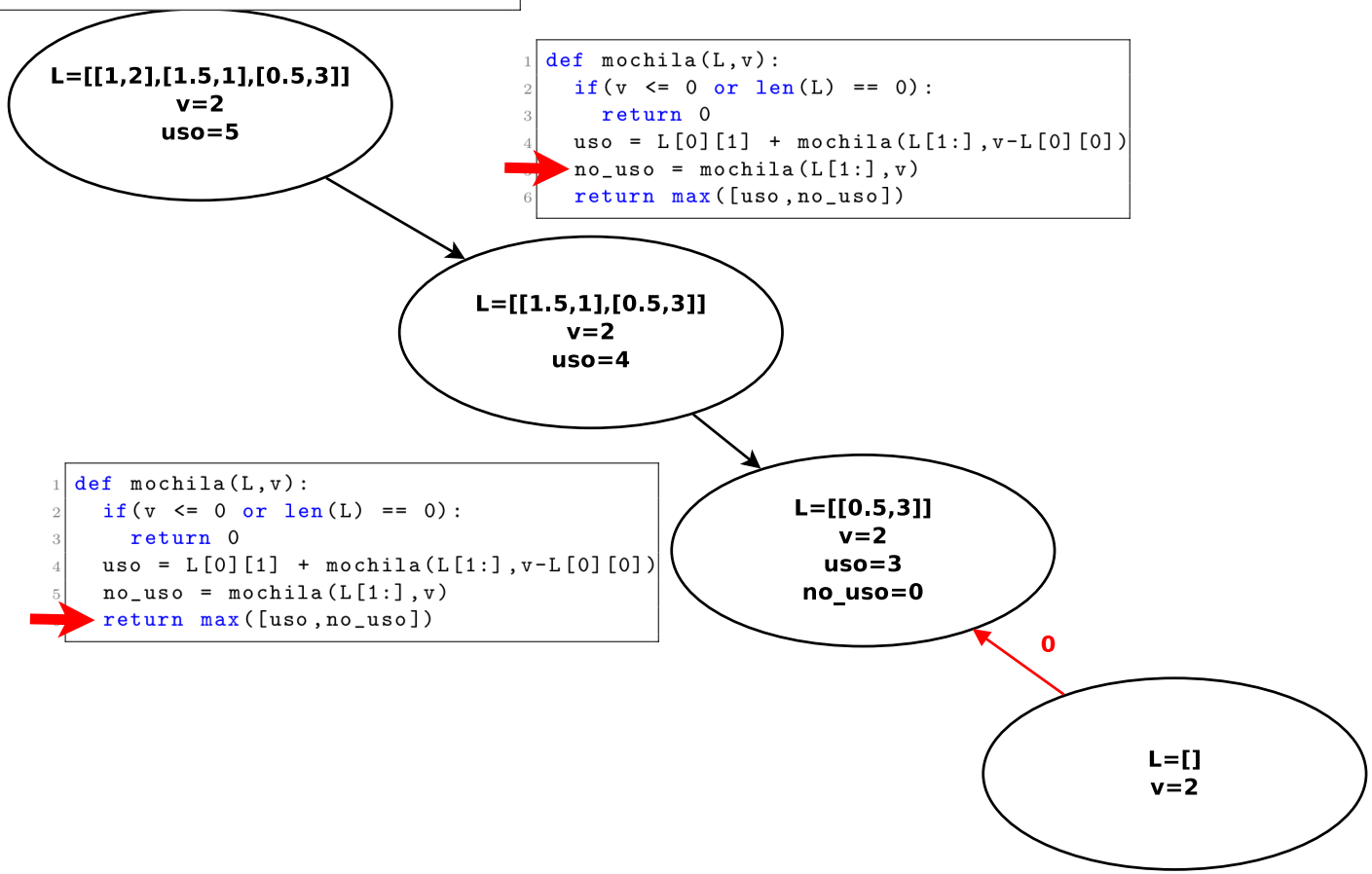
L=[]
v=1.5

Recursión


def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5
def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[[1.5,1],[0.5,3]]
v=2
uso=4
def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[[0.5,3]]
v=2
uso=3
def mochila(L,v):
if(v <= 0 or len(L) == 0):
return 0
uso = L[0][1] + mochila(L[1:],v-L[0][0])
no_uso = mochila(L[1:],v)
return max([uso,no_uso])
L=[]
v=2

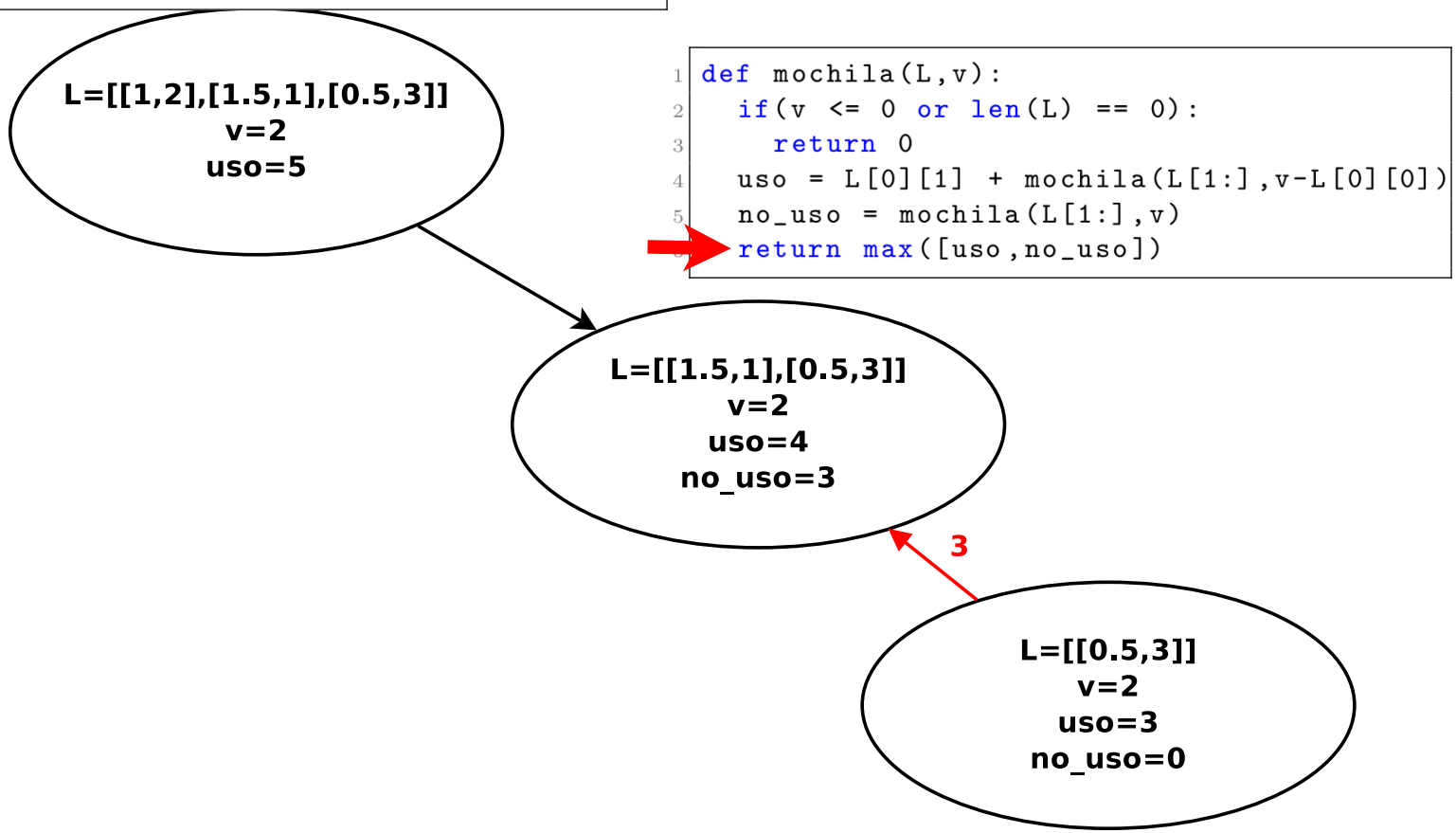
Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```



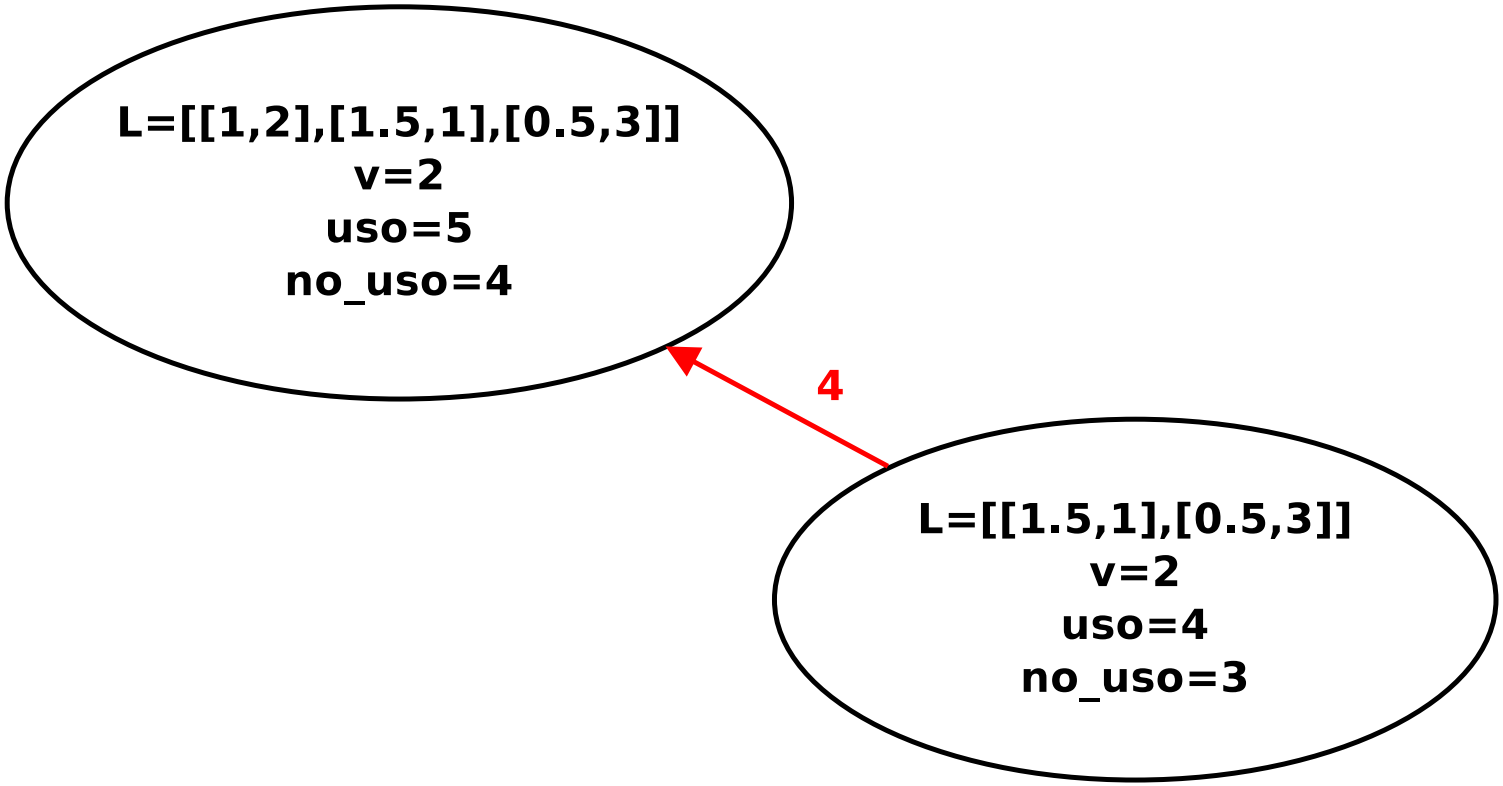
Recursión

```
1 def mochila(L,v):
2   if(v <= 0 or len(L) == 0):
3     return 0
4   uso = L[0][1] + mochila(L[1:],v-L[0][0])
5   no_uso = mochila(L[1:],v)
6   return max([uso,no_uso])
```

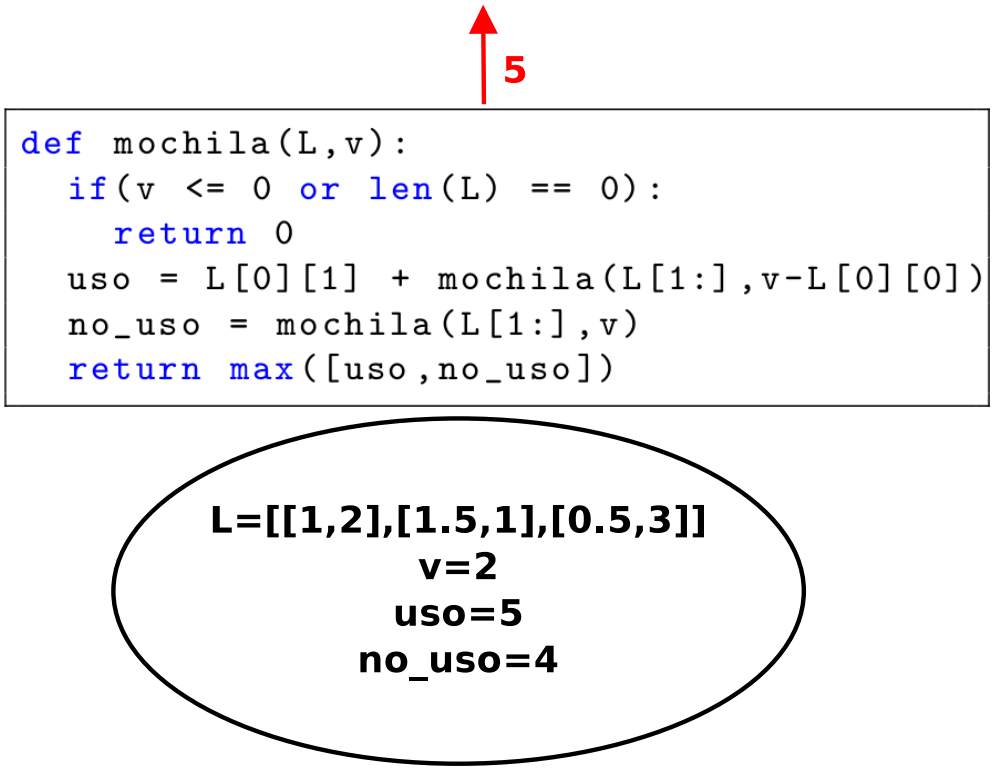


Recursión

```
1  def mochila(L,v):
2    if(v <= 0 or len(L) == 0):
3      return 0
4    uso = L[0][1] + mochila(L[1:],v-L[0][0])
5    no_uso = mochila(L[1:],v)
6    return max([uso,no_uso])
```



Recursión


5
1 def mochila(L,v):
2 if(v <= 0 or len(L) == 0):
3 return 0
4 uso = L[0][1] + mochila(L[1:],v-L[0][0])
5 no_uso = mochila(L[1:],v)
6 return max([uso,no_uso])
L=[[1,2],[1.5,1],[0.5,3]]
v=2
uso=5
no_uso=4

Nos vemos!

Eso fue todo, que les vaya bien en el examen :)