[6pkt.] Zadanie 1.

Szablon rozwiązania:	`zad1.py`
Pierwszy próg złożoności:	$O(n^2)$
Drugi próg złożoności:	$O(n \log n)$

Dany jest zbiór przedziałów domkniętych $I = \{[a_1, b_1], \ldots, [a_n, b_n]\}$ gdzie każdy przedział zaczyna się i kończy na liczbie naturalnej (wliczając 0). Dane są także dwie liczby naturalne x i y. Dwa przedziały można skleić (czyli zamienić na przedział będący ich sumą mnogościową) jeśli mają dokładnie jeden punkt wspólny. Jeśli pewne przedziały można posklejać tak, że powstaje z nich przedział $[x, y]$ to mówimy, że są przydatne. Proszę napisać funkcję:

```
def intuse( I, x, y )
```

która zwraca listę numerów wszystkich przydatnych przedziałów. Zbiór I jest reprezentowana jako lista par opisujących przedziały. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Przykład. Dla danych:

```
I = [ (3,4), (2,5), (1,3), (4,6), (1,4) ]
#       0      1      2      3      4
x = 1
y = 6
```

prawidłowym wynikiem wywołania `intuse(I, x, y)` jest dowolna permutacja listy `[0,2,3,4]`.

[6pkt.] Zadanie 3.

Szablon rozwiązania:	`zad3.py`
Pierwszy próg złożoności:	$O(n + T)$, gdzie T to łączna liczba wciśnięć przełączników; na potrzeby analizy należy przyjąć, że $T = \Omega(m \log n)$.
Drugi próg złożoności:	$O(m \log n)$

Dane są lampki o numerach od 0 do $n - 1$. Każda z nich może świecić na zielono, czerwono, lub niebiesko i ma jeden przełącznik, który zmienia jej kolor (z zielonego na czerwony, z czerwonego na niebieski i z niebieskiego na zielony). Początkowo wszystkie lampki świecą na zielono. Operacja (a, b) oznacza "wciśnięcie przełącznika na każdej z lampkek o numerach od a do b". Wykonanych będzie m operacji. Proszę napisać funkcję:

```
def lamps( n, L )
```

która mając daną liczbę n lampek oraz listę L operacji (wykonywanych w podanej kolejności) zwraca ile maksymalnie lampek świeciło się na niebiesko (lampki są liczone na początku i po wykonaniu każdej operacji)

Przykład. Wywołanie:

```
lamps( 8, [(0,4),(2,6)] )
```

powinno zwrócić liczbę 3. Początkowo wszystkie lampki (o numerach od 0 do 7) świecą się na zielono. Następnie lampki o numerach od 0 do 4 zmieniają kolor na czerwony. Po ostatniej operacji lampki o numerach od 2 do 4 zmieniają kolor na niebieski, a lampki 5 i 6 zmieniają kolor na czerwony.

[6pkt.] Zadanie 2.

Szablon rozwiązania:	`zad2.py`
Pierwszy próg złożoności:	$O(n^2)$
Drugi próg złożoności:	$O(n)$

Dane jest drzewo T zawierające n wierzchołków. Każda krawędź e drzewa ma wagę $w(e) \in \mathbb{N}$ oraz unikalny identyfikator $id(e) \in \mathbb{N}$. Wagą drzewa jest sumę wag jego krawędzi. Proszę napisać funkcję:

```
def balance( T ):
  ...
```

która zwraca identyfikator takiej krawędzi e drzewa, że usunięcie e dzieli drzewo na takie dwa, których różnica wag jest minimalna. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Reprezentacja drzewa. Drzewo reprezentowane jest przy pomocy węzłów typu `Node`:

```
class Node:
  def __init__( self ):   # stwórz węzeł drzewa
    self.edges   = []     # lista węzłów do których są krawędzie
    self.weights = []     # lista wag krawędzi
    self.ids     = []     # lista identyfikatorów krawędzi

  def addEdge( self, x, w, id ): # dodaj krawędź z tego węzła do węzła x
    self.edges.append( x )       # o wadze w i identyfikatorze id
    self.weights.append( w )
    self.ids.append( id )
```

Pole `edges` zawiera listę obiektów typu `Node`. Pola `edges`, `weights` oraz `ids` to listy równej długości. Należy założyć, że drzewo ma conajmniej jedną krawędź. Dopuszczalne jest dopisywanie własnych pól do `Node`.

Przykład. Rozważmy poniższe drzewo:

```
A = Node()
B = Node()
C = Node()
D = Node()
E = Node()
A.addEdge(B, 6 , 1 )
A.addEdge(C, 10, 2 )
B.addEdge(D, 5 , 3 )
B.addEdge(E, 4 , 4 )
```

Wywołanie `balance(A)` powinno zwrócić liczbę 1, czyli identyfikator krawędzi z węzła A do B o wadze 6. Usunięcie jej dzieli nasze drzewo na dwie części, o wagach 10 (krawędź z A do C) oraz 9 (drzewo z korzeniem B i krawędziamy do D i E o wagach 4 i 5).