1-A) DESARROLLO CON INANICIÓN → Usamos un lock para garantizar la exclusión mútua

Monitor

```
def __init__()
    lock = Lock()
    CN_pasando : int = 0
    CS_pasando : int = 0
    P_pasando : int = 0

    permiso_C : VC = True
    permiso_P : VC = True
```

2) A)

INV:

$0 \leq CN\_pasando \leq NCARS$

$0 \leq CS\_pasando \leq NCARS$

$0 \leq P\_pasando \leq NPED$

$CN\_pasando > 0 \Rightarrow CS\_pas = 0 \wedge P\_pas = 0$

$CS\_pasando > 0 \Rightarrow CN\_pas = 0 \wedge P\_pas = 0$

$P\_pasando > 0 \Rightarrow CN\_pas = 0 \wedge CS\_pas = 0$

```
def wants_enter_CN()
    lock.acquire()
    permiso_C.wait_for(puedePasar(CN))
    CN_pasando += 1
    lock.release()

def leaves_car_CN()
    lock.acquire()
    CN_pasando -= 1
    permiso_C.notify_all()
    permiso_P.notify_all()
    lock.release()

def wants_enter_P()
    lock.acquire()
    permiso_P.wait_for(puedePasarP)
    P_pasando += 1
    lock.release()

def wants_enter_CS()
    lock.acquire()
    permiso_C.wait_for(puedePasarCS)
    CS_pasando += 1
    lock.release()

def leaves_car_CS()
    lock.acquire()
    CS_pasando -= 1
    permiso_C.notify_all()
    permiso_P.notify_all()
    lock.release()

def leaves_P()
    lock.acquire()
    P_pasando -= 1
    permiso_C.notify_all()
    permiso_P.notify_all()
    lock.release()
```

Hay inanición porque 2 de ellos podrían "pasarse" cuando van a salir y entrar solo ellos 2, sufriendo el otro inanición. Para solucionarlo, podemos añadir turnos que reparta el paso de forma justa, e independiente. Para ello, añadimos variables compartidas para saber cuántos están esperando de cada tipo, indicando que quieren pasar (como para el productor consumidor) y otra variable compartida que lleve el nº de vehículos/peatones que pasan seguidos, para limitar el nº, y poder cambiar el turno de forma justa.

(1) B) DESARROLLO SIN INANICIÓN.

El turno se gestionará de forma que como máximo pueden pasar MAX_CN, MAX_CS y MAX_P seguidos. Una vez llegado al límite el turno cambia al siguiente que tenga elementos esperando, siguiendo el orden: CN → CS → P → CN. Si se ha llegado al MAX_x pero no hay nadie más esperando en el puente, el contador de seguidos se reinicia y pueden seguir pasando.

Monitor

```
def __init__():
    lock = Lock()

    CN_pasando : int = 0
    CS_pasando : int = 0
    P_pasando : int = 0

    CN_seguidos : int = 0
    CS_seguidos : int = 0
    P_seguidos : int = 0

    CN_esperando : int = 0
    CS_esperando : int = 0
    P_esperando : int = 0

    turno : int = 0   (0, 1, 2)

    permiso_C : VC = True
    permiso_P : VC = True
```

(2) B)

"Puente por turnos.py" (DEMO EN EL CÓDIGO)

Invariante

$0 \leq CN\_pasando \leq NCARS$

$0 \leq CS\_pasando \leq NCARS$

$0 \leq P\_pasando \leq NPED$

$0 \leq CN\_seguidos \leq MAX\_SEG\_CN$

$0 \leq CS\_seguidos \leq MAX\_SEG\_CS$

$0 \leq P\_seguidos \leq MAX\_SEG\_P$

$0 \leq CN\_esperando \leq NCARS$

$0 \leq CS\_esperando \leq NCARS$

$0 \leq P\_esperando \leq NPED$

$0 \leq turno \leq 2$

$CN\_pas > 0 \Rightarrow CS\_pas = 0 \wedge P\_pas = 0 \wedge tur = 0$

$CS\_pas > 0 \Rightarrow CN\_pas = 0 \wedge P\_pas = 0 \wedge tur = 1$

$P\_pas > 0 \Rightarrow CN\_pas = 0 \wedge CS\_pas = 0 \wedge tur = 2$

```
def wants_enter_CN():
    lock.acquire()
    CN_esperando += 1
    permisoC.wait_for(puedenPasar CN)
    CN_esperando -= 1
    CN_pasando += 1
    CN_seguidos += 1
    si CN_seguidos == MAX_SEG_CN V CN_esperando = 0:
        si CS_esperando > 0 → turno = 1
        sino y P_esperando > 0 → turno = 2
        CN_seguidos = 0
    lock.release()
```

```
def leaves_CN():
    lock.acquire()
    CN_pasando -= 1
    permiso_C.notify_all()
    permiso_P.notify_all()
    lock.release()
```

def y

```
def wants_enter_CS()
    lock.acquire()
    CS_esperando += 1
    permiso_C.wait_for(pueden_pasar_CS)
    CS_esperando -= 1
    CS_pasando += 1
    CS_seguidos += 1
    si CS_seguidos == MAX_SEG_CS ∨ CS_esperando = 0:
        si P_esperando > 0 → turno = 2
        si no y CN_esp > 0 → turno = 0
    lock.release
```

```
def leaves_CS()
    lock.acquire
    CS_pasando -= 1
    permiso_C.notifyall()
    permiso_P.notifyall()
    lock.release()
```

```
def wants_enter_P()
    lock.acquire()
    P_esperando += 1
    permisoP.wait_for(pueden_pasar_P)
    P_esperando -= 1
    P_pasando += 1
    P_seguidos += 1
    si P_seguidos = MAX_SEG_P ∨ P_esperando = 0:
        si CN_esp > 0 → turno = 0
        si no y CS_esp > 0 → turno = 1
    lock.release()
```

```
def leaves_P()
    lock.acquire
    P_pasando -= 1
    permiso_C.notify_all()
    permiso_P.notify_all()
    lock.release()
```

```
def pueden_pasar_CN()
    si CS_esperando = 0 ∧ P_esperando = 0 → turno = 0
        (No hay nadie esperando a entrar)
    return CS_pasando = 0 ∧ P_pasando = 0 ∧ turno = 0
```

```
def pueden_pasar_CS()
    si CN_esperando > 0 ∧ P_esperando = 0 → turno = 1
    return CN_pasando = 0 ∧ P_pasando = 0 ∧ turno = 1
```

```
def pueden_pasar_P()
    si CN_esperando = 0 ∧ CS_esperando = 0 → turno = 2
    return CN_pasando = 0 ∧ CS_pasando = 0 ∧ turno = 2
```

③ Versión con inanición y sin inanición
Demo que el puente es seguro (no hay 2 grupos cruzando a la vez)

Tanto los coches del norte, como los coches del sur, como los peatones esperan (wait_for) a que se cumpla que no haya nadie de los otros 2 cruzando y sea su turno (métodos ← puedenPasarCN, puedenPasarCS, puedenPasarP). Si alguno de los otros estuviera cruzando devolvería False, luego no podrían pasar. Cuando el último salga del puente, avisará al resto con los "notify_all", y volverán a comprobar si pueden pasar, luego el puente el seguro.

④ Demo que no hay Deadlocks

B) Sin inanición:
Para que ocurriera un Deadlock tendrían que estar todos los que no hayan terminado atascados en su wait, pero eso no puede ser ya que tendría que darse al mismo tiempo:

$$CN\_pasando = 0 \land CS\_pasando = 0 \land P\_pasando = 0 \land \underbrace{\neg turno = 0 \land \neg turno = 1 \land \neg turno = 2}_{(!!)}$$

con $CN\_esperando > 0 \lor CS\_esperando > 0 \lor P\_esperando > 0$, ya que si no habrían terminado.

El turno siempre será de alguien por como está definido, y si solo hay un tipo esperando a entrar, cambia a él automáticamente (en el if del "puedenPasarX").

A) Con inanición tampoco hay deadlock ya que aunque no haya turnos, si queda alguien por pasar, como los otros 2 no están pasando, tendrá permiso para entrar.

⑤ ↘

⑤ Demo ausencia de inanicion

A) En "PUENTE BÁSICO (con inanicion)" sí hay inanicion, ya que uno de los tipos podría quedarse aislado si los otros dos se intercambian el paso uno con otro, lo mismo pasaría con la justicia, en esta versión NO la hay, ya que si no dejan de llegar de un tipo, los otros 2 no pasarían nunca

B) En la versión final con turnos ("PUENTE POR TURNOS") NO hay inanicion ya que la propia gestión del turno lo impide. Se establecen 3 constantes:

MAX-SEGUIDOS_CN, MAX_SEGUIDOS_CS y MAX-SEGUIDOS_P, y cuando pasen esa cantidad de elementos seguidos de un mismo tipo, el turno cambiará al siguiente que tenga elementos esperando a entrar, siguiendo el orden Coches Norte → Coches Sur → Peatones → Coches Norte, y garantizando la justicia

⑥ Implementación
- PUENTE BÁSICO (con inanicion).py
- PUENTE POR TURNOS.py