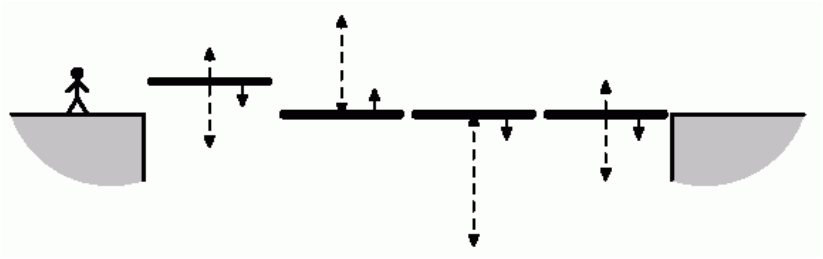
Turbo elevator

In this game, the aim is to get to the other side of the crevice by means of a turbo elevator. This consists of a number of horizontal platforms that were placed one after the other. The platforms vertically move up and down, and operate independently. You can only jump to the next platform, if it is situated on the same height as the platform you are on at the moment.



Assignment

An initial level and an initial direction (going up, going down or standing still) is set for every platform at the beginning of the game. Furthermore, a lowest and highest level is set for every platform, these can not be changed afterwards. With every next time step, the platform moves one level up or down, or it stands still and stays in the same position. If the platform has reached the lowest (resp. highest) level, it will move back up (resp. down) from the next time step. A platform for which the lowest level is equal to the highest level, or for which was indicated that it is immobile, stays in the same position with every time step. Write a class `Platform` that can be used to model such platforms. For this, an object of the class `Platform` must contain the following methods:

- An initializing method `__init__` with which the configuration of a platform can be set. To this method 4 whole numbers must be given as an argument: *i*) initial level, *ii*) initial direction, *iii*) lowest level and *iv*) highest level. The last two arguments are optional, and are equated with the initial level if it is not given. The values of the four arguments must respectively be appointed to the attributes `position`, `direction`, `lowest` and `highest` of the newly made object of the class `Platform`. For the configuration of a platform, The lowest level may not be higher than the highest level, and the initial level must be situated in between the lowest and highest level (boundaries included). Look at the example below to see how the initializing method must react if these conditions are not met. The direction in which the platform initially moves, is represented by the values -1 (down), 0 (standing still) or 1 (up). See to it that these values can be symbolically represented by the variables `Platform.DOWN`, `Platform.STILL` and `Platform.UP`.
- A method `next` to simulate that the elevator moves one level up or down or stays still within the period of one time step. This method must adjust the value of the attribute `position` according to the description from the introduction.

```
>>> platform = Platform(0, Platform.OP, -1, 1)
>>> platform.position
0
>>> for steps in range(5):
...     platform.next()
...     print(platform.position)
1
0
```

```
-1
0
1
>>> platform = Platform(1, Platform.DOWN, 1, 1)
>>> for steps in range(3):
...     print(platform.position)
...     platform.next()
1
1
1
>>> platform = Platform(3, Platform.STILL)
>>> for steps in range(3):
...     print(platform.position)
...     platform.next()
3
3
3
>>> platform = Platform(-3, Platform.UP, -1, 1)
Traceback (most recent call last):
AssertionError: invalid configuration
```

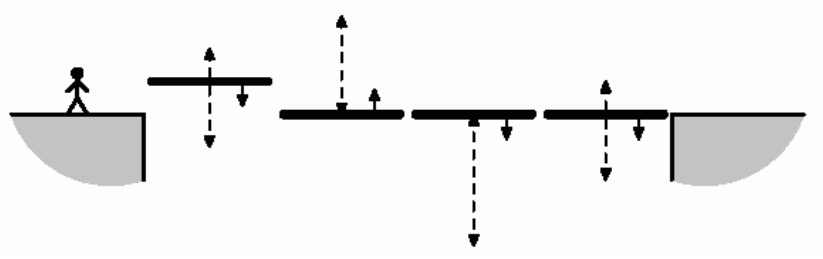
Now, also write a class TurboElevator, of which every object represents a turbo elevator that is built from a number of platforms. The first and last platform represent the starting point and the other side of the crevice, but do not have to be on the same level, and also do not have to stand still. To make an object of the class TurboElevator, no arguments must be given. In the beginning the turbo elevator does not have any platforms yet, but an extra platform can be added at any time. Decide for yourself if it is necessary to provide an __init__ method for the class TurboElevator, and how it should be implemented. Objects of the class TurboElevator must however contain the following methods:

- A method add with which a new platform (an object of the class Platform) can be added to the turbo elevator.
- A method timesteps that prints after how many time steps the last platform of the turbo elevator was reached. When calling this method, the time is set to zero, and one always starts from the first platform with every time step, first all platforms of the turbo elevator either move a position up or down, or they stand still, conform the method next of the class Platform. If, after that, the platform on which we are situated is on the same level as the next platform, we go to the next platform. The method timesteps must, according to the procedure above, also gradually adjust the condition of the platforms of the turbo elevator. Not that, because of this, it is possible that with every next call of the method time steps, a different value is printed, even though nothing has changed in the configuration of the turbo elevator. It is also possible that with a certain configuration of a turbo elevator, it is impossible to reach the other side. Therefore, the method timesteps must print the value None, if the last platform is not reached after 1000 time steps.

```
>>> turboElevator = TurboElevator()
>>> turboElevator.add(Platform(0, Platform.STILL))
>>> turboElevator.add(Platform(2, Platform.DOWN, -2, 2))
>>> turboElevator.add(Platform(0, Platform.UP, 0, 4))
>>> turboElevator.add(Platform(0, Platform.DOWN, -4, 0))
>>> turboElevator.add(Platform(0, Platform.DOWN, -2, 4))
>>> turboElevator.add(Platform(0, Platform.STILL))
>>> turboElevator.timesteps()
16
>>> turbo elevator.add(Platform(4, Platform.OP, 2, 8))
```

```
>>> print(turboElevator.timesteps())
None
```

Bij dit spelletje is het de bedoeling om aan de overkant van een kloof te geraken met behulp van een turbolift. Deze bestaat uit een aantal horizontale platformen die aansluitend achter elkaar opgesteld worden. De platformen bewegen verticaal op en neer, en werken onafhankelijk van elkaar. Je kunt enkel naar het volgende platform springen, wanneer dat zich op dezelfde hoogte bevindt als het platform waarop je momenteel staat.



Opgave

Voor elk platform wordt bij aanvang een startniveau ingesteld en een richting waarin het platform zich initieel beweegt: naar boven, naar beneden of stilstaan. Daarnaast wordt voor het platform ook een laagste en een hoogste niveau ingesteld, en deze kunnen achteraf niet meer gewijzigd worden. Bij elke volgende tijdsstap beweegt het platform één niveau naar boven of naar beneden, of blijft het stilstaan. Als het platform het laagste (resp. hoogste) niveau bereikt heeft, dan beweegt het vanaf de volgende tijdsstap terug naar boven (resp. naar beneden). Een platform waarvoor het laagste niveau gelijk is aan het hoogste niveau, of waarvoor bij aanvang werd opgegeven dat het stilstaat, blijft bij elke tijdsstap stilstaan. Definieer een klasse `Platform` die kan gebruikt worden om dergelijke platformen te modelleren. Hiervoor moet een object van de klasse `Platform` over de volgende methoden beschikken:

- Een initialisatiemethode `__init__` waarmee de configuratie van een platform kan ingesteld worden. Aan deze methode moeten vier gehele getallen als argument doorgegeven worden: *i*) startniveau, *ii*) startrichting, *iii*) laagste niveau en *iv*) hoogste niveau. De laatste twee argumenten zijn optioneel, en worden gelijkgesteld aan het startniveau indien ze niet worden doorgegeven. De waarden van de vier argumenten moeten respectievelijk toegekend worden aan de attributen `stand`, `richting`, `laagste` en `hoogste` van het nieuw aangemaakt object van de klasse `Platform`. Voor de configuratie van een platform moet gelden dat het laagste niveau niet hoger ligt dan het hoogste niveau, en dat het startniveau tussen het laagste en het hoogste niveau gelegen is (grenzen inbegrepen). Bekijk onderstaand voorbeeld om te zien hoe de initialisatiemethode moet reageren als niet aan deze voorwaarden voldaan is. De richting waarin het platform initieel beweegt, wordt voorgesteld door de waarden -1 (naar beneden), 0 (stilstand) of 1 (naar boven). Zorg er voor dat deze waarden symbolisch kunnen voorgesteld worden door de variabelen `Platform.NEER`, `Platform.STIL` en `Platform.OP`.
- Een methode `volgende` waarmee gesimuleerd wordt dat de lift binnen de periode van één tijdsstap, één niveau naar boven of naar beneden beweegt, of blijft stilstaan. Deze methode moet de waarde van het attribuut `stand` aanpassen volgens de beschrijving uit de inleiding.

```
>>> platform = Platform(0, Platform.OP, -1, 1)
>>> platform.stand
0
```

```
>>> for stappen in range(5):
...     platform.volgende()
...     print(platform.stand)
1
0
-1
0
1
>>> platform = Platform(1, Platform.NEER, 1, 1)
>>> for stappen in range(3):
...     print(platform.stand)
...     platform.volgende()
1
1
1
>>> platform = Platform(3, Platform.STIL)
>>> for stappen in range(3):
...     print(platform.stand)
...     platform.volgende()
3
3
3
>>> platform = Platform(-3, Platform.OP, -1, 1)
Traceback (most recent call last):
AssertionError: ongeldige configuratie
```

Definieer ook nog een klasse TurboLift, waarvan elk object een turbolift voorstelt die is opgebouwd uit een aantal platformen. Het eerste en laatste platform stellen hierbij het vertrekpunt en de overkant van de kloof voor, maar hoeven niet op hetzelfde niveau te staan, en moeten ook niet noodzakelijk stilstaan. Voor het aanmaken van een object van de klasse TurboLift moeten geen argumenten opgegeven worden. Bij aanvang heeft een turbolift nog geen platformen, maar er kan te allen tijde achteraan een extra platform toegevoegd worden. Beslis zelf of het nodig is om voor de klasse TurboLift een __init__ methode te voorzien, en hoe die dan moet geïmplementeerd worden. Objecten van de klasse TurboLift moeten wel verplicht de volgende methoden hebben:

- Een methode voegtoe waarmee achteraan de turbolift een nieuw platform (een object van de klasse Platform) kan toegevoegd worden.
- Een methode tijdsstappen die teruggeeft na hoeveel tijdsstappen men het laatste platform van de turbolift bereikt. Bij het aanroepen van deze methode wordt de tijd op nul gezet, en vertrekt men vanaf het eerste platform. Bij elke tijdsstap bewegen eerste alle platformen van de turbolift één niveau naar boven of naar beneden, of blijven ze stilstaan, conform de methode volgende van de klasse Platform. Als daarna het platform waarop men zich bevindt op hetzelfde niveau staat als het volgende platform, dan stapt men over naar het volgende platform. De methode tijdsstappen moet volgens bovenstaande procedure ook gaandeweg de toestand van de platformen van de turbolift aanpassen. Merk op dat het hierdoor mogelijk is dat bij elke volgende aanroep van de methode tijdsstappen een verschillende waarde wordt teruggegeven, ook al wordt aan de configuratie van de turbolift niets gewijzigd. Het kan ook zijn dat het met een gegeven configuratie van een turbolift onmogelijk is om de overkant te bereiken. Daarom moet de methode tijdsstappen de waarde None teruggeven, indien het laatste platform nog niet bereikt is na 1000 tijdsstappen.

```
>>> turbolift = TurboLift()
>>> turbolift.voegtoe(Platform(0, Platform.STIL))
>>> turbolift.voegtoe(Platform(2, Platform.NEER, -2, 2))
```

```
>>> turbolift.voegtoe(Platform(0, Platform.OP, 0, 4))
>>> turbolift.voegtoe(Platform(0, Platform.NEER, -4, 0))
>>> turbolift.voegtoe(Platform(0, Platform.NEER, -2, 4))
>>> turbolift.voegtoe(Platform(0, Platform.STIL))
>>> turbolift.tijdsstappen()
16
>>> turbolift.voegtoe(Platform(4, Platform.OP, 2, 8))
>>> print(turbolift.tijdsstappen())
None
```