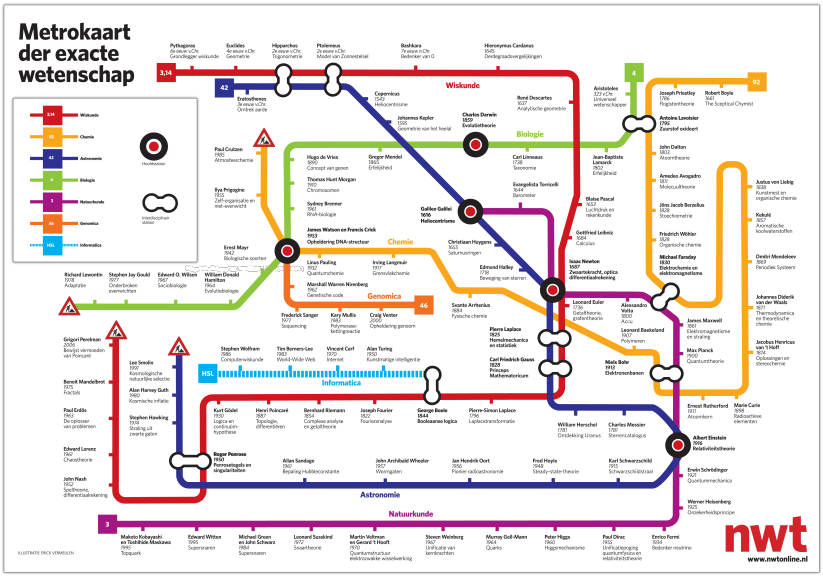
Subway map of the exact sciences

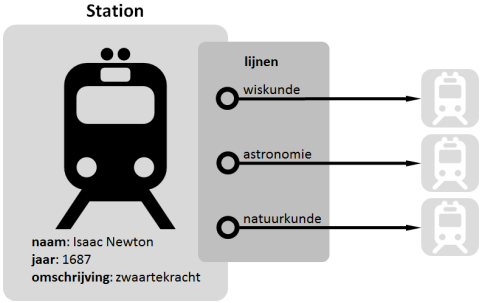
In its 2011 November issue, NWT Magazine published the *Underground map of Science* below. The challenge of this assignment consists of adding new data types to Python that allow to construct, expand and question such underground maps. An underground map is built from a number of underground lines that link different stations in a set order The traffic on an underground line goes in only one direction, and for this assignment, underground lines never form a loop. Some stations can be on multiple underground lines, so that it is possible to transfer from one line to another.



Underground map of Science

Assignment

Define a class Station with which the underground stations can be represented on a map. Every underground station possesses the properties name, year and description, that each represent a string (the year can also be described as for example: 6th century BC). From every station you can go to zero or more other metro stations, but each of these outgoing connections are on a different underground line. To represent these outgoing connections, every station possesses a property lines that refers to a dictionary. This dictionary prints the underground lines that leave from a station, by portraying each name of a line (a string; here always the name of a science) on the following line.



Schematic representation of the objects of the class Station.

The objects of the class Station must possess at least the following methods:

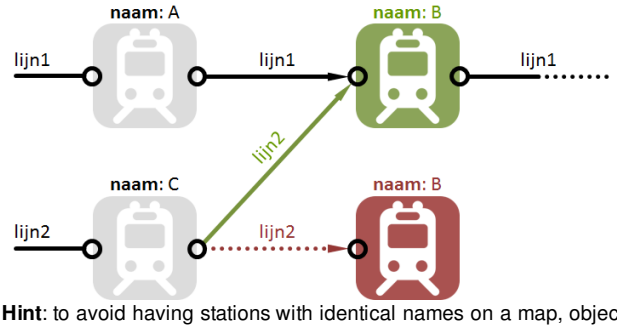
- An initializing method __init__ with at least three parameters: name, year and description. Three strings must be given to these parameters, that each must be appointed to the properties of the same name of the newly constructed object. Furthermore, every newly constructed object should also possess a property lines that initially refers to an empty dictionary.
- A method __eq__ that can be used to test whether two underground stations are equal. Two stations are only seen as equal if the values of their properties name, year and description are equal. Do note that the values of the property lines has no influence on the equality of underground stations.
- A method __repr__ that prints a string representation of the underground stationd of the format Station(name=*name*, year=year, description=description). Here, *name*, *year* and *description* must respectively be filled out with the string representation of the properties name, year and description of the object.
- A method __str__ that prints the string representation of the underground station in the format *name* (*year*, *description*). Here, *name*, *year* and *description* must respectively be filled out with the value of the properties name, year and description of the object.
- A method link that can be used to link an underground station on a given underground line with another station (an object of the class Station). The name of the underground line and the other underground station with which the link is made, must be given to the method as arguments. The method must raise an AssertionError with the message station is already linked with line *name* if the station already has an outgoing connection for the given underground line. Here, instead of *name* in the message, the name of the underground line should appear.
- A method next to which the name of an underground line must be given. The method must print the station with which the underground station is linked on the given line. If the station on the given line is not connected with any other stations, the method should print the value None.

Define a class UndergroundMap with which underground maps can be represented. Underground maps are built from a number of underground lines that connect various stations in a set order. Every underground map possesses a property lines, that refers to a dictionary. This dictionary portrays the name (a string; here always the name of a science) for every underground line on the map on the begin station of the underground line (an object of the class Station). The objects of the class UndergroundMap must at least contain the following methods:

- An initializing method __init__ that makes sure that every newly constructed underground map has a property lines that initially refers to an empty dictionary.
- A method beginstation that prints the begin station (an object from the class Station) of a given underground line. The name of the underground line must

be given to the function as an argument. If the underground map does not contain a line of the given name, the method should print the value None.

- A method terminal that prints the terminal (an object of the class Station) of a given underground line. The name of the line must be given to the function as an argument. If the underground map does not contain a line of the given name, the method must print the value None.
- A method expand which can be used to expand a given underground line with a new station. The name of the line and the new station (an object of the class Station) must be given to the method as arguments. Expanding an underground line is done by linking the terminal of the line with the new station. If the underground map does not yet contain a line with the given name, the new station should be the begin station of the line. But be careful: a map may never contain two stations with the same name. If the map already has a station with the same name of the new station, the terminal of the given line must be linked with the station that already exists. Suppose the situation below where line1 already has a station with the name B, and we want to expand line2 (with terminal C) with a station named B. Then station C must be linked with the existing station (indicated in green) and no new station should be added to the map (indicated in red).



Hint: to avoid having stations with identical names on a map, objects of the class UndergroundMap can keep a dictionary, that portrays the names of the stations on the map on the corresponding objects of the class Station.

- A method addStations to which a location of a text file must be given. Every line of this file must contain the four following fields, separated by a single tab: *i*) the name of a science, *ii*) the name of a scientist, *iii*) a year and *iv*) a short description of important accomplishments of the scientist. The method must expand for every line indicated with the underground line with the name of a science, with a station of which the name, the year and the description are based on the last three fields. These expansions must happen in the order in which the lines occur in the file.
- Make sure that the initializing method has an optional parameter stations, to which the location of a text file can be given. If a value is given to this parameter, the map must already be expanded with the order and description of the stations like they are in the text file in the initialization. You do this of course by calling the method addStations.

Make sure you use the methods you have already implemented optimally when implementing the classes Station and UndergroundMap.

Example (class Station)

```
>>> station1 = Station('James Watson', '1953', 'clarification DNA-structure')
>>> station1.name
'James Watson'
>>> station1.year
'1953'
>>> station1.description
'clarification DNA-structure'
>>> station1.lines
{}
>>> station1 == station1
True
>>> station1
Station(name='James Watson', year='1953', description='clarification DNA-structure')

>>> station2 = Station(name='Ernst Mayr', year='1942', description='biological species')
>>> print(station2)
Ernst Mayr (1942, biological species)
>>> station1 == station2
False

>>> station1.link('biology', station2)
>>> station1.lines
{'biology': Station(name='Ernst Mayr', year='1942', description='biological species')}
>>> station1.next('biology')
Station(name='Ernst Mayr', jaar='1942', description='biological species')
>>> print(station1.next('chemistry'))
None

>>> station3 = Station('Marshall Warren Nirenberg', '1962', 'genetic code')
>>> station4 = Station('Ilya Prigogine', '1955', 'self-organization and non-balance')
>>> station1.link('chemistry', station3)
>>> station1.link('chemistry', station4)
Traceback (most recent call last):
AssertionError: station is already linked with line chemistry
>>> station1.link('genomics', station4)
>>> station1.lines
{'biologie': Station(name='Ernst Mayr', yearr='1942', description='biological species'), 'genomics': Station(name='Ilya Prigogine', year='1955', description='self-organization and non-balance'), 'cher
>>> print(station1.next('chemistry'))
Marshall Warren Nirenberg (1962, genetic code)
>>> print(station1.next('genomics'))
Ilya Prigogine (1955, self-organization and non-balance)
```

Example (class Metromap)

In the example session below we assume that the text file milestones.txt is situated in the current directory.

```
>>> metromap = Metrokaart()
>>> metromap.expand('genomics', Station('James Watson', '1953', 'clarification DNA-structure'))
>>> metromap.expand('genomics', Station('Marshall Warren Nirenberg', '1962', 'genetic code'))
>>> metromap.expand('genomics', Station(name='Frederick Sanger', year='1977', description='sequencing'))
>>> metromap.beginstation('genomics')
Station(naam='James Watson', year='1953', description='clarification DNA-structure')
```

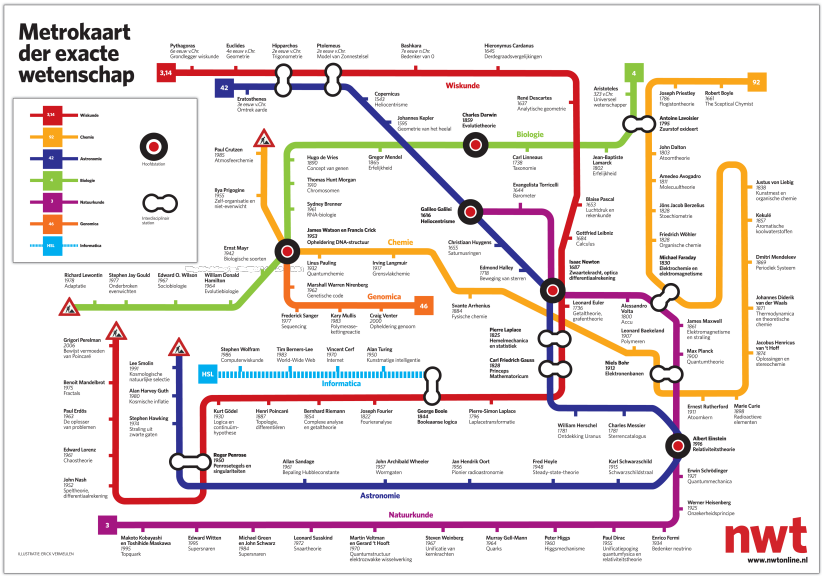
```
>>> metromap.terminal('genomics')
Station(name='Frederick Sanger', year='1977', description='sequencing')
>>> metromap.expand('chemistry', Station('Robert Boyle', '1661', 'The Sceptical Chymist'))
>>> metromap.beginstation('chemistry')
Station(name='Robert Boyle', year='1661', description='The Sceptical Chymist')
>>> metromap.terminal('chemistry')
Station(name='Robert Boyle', year='1661', description='The Sceptical Chymist')

>>> metromap2 = Undergroundmetromap('milestones.txt')
>>> metromap2.beginstation('chemistry')
Station(name='Robert Boyle', year='1661', description='The Sceptical Chymist')
>>> metromap2.terminal('chemistry')
Station(name='Paul Crutzen', year='1985', description='atmospheric chemistry')

>>> station = metromap2.beginstation('genomics')
>>> station
Station(name='James Watson', year='1953', description='clarification DNA-structure')
>>> station.next('biology')
Station(name='Ernst Mayr', year='1942', description='biological species')
>>> print(station.next('genomics'))
Marshall Warren Nirenberg (1962, genetic code)
>>> print(station.next('chemistry'))
Ilya Prigogine (1955, self-organization and non-balance)
>>> station.next('genomics').next('genomics')
Station(name='Frederick Sanger', year='1977', description='sequencing')

>>> metromap2.expand('chemistry', Station('Martin Karplus', '2013', 'computational chemistry'))
>>> metromap2.expand('informatics', Station('Martin Karplus', '2013', 'computational chemistry'))
>>> station = metromap2.terminal('chemistry')
>>> station
Station(name='Martin Karplus', year='2013', description='computational chemistry')
>>> metromap2.expand('chemistry', Station('Michael Levitt', '2014', 'computational chemistry'))
>>> metromap2.expand('informatics', Station('Arieh Warshel', '2014', 'computational chemistry'))
>>> station.lines
{'informatics': Station(name='Arieh Warshel', year='2014', description='computational chemistry'), 'chemistry': Station(name='Michael Levitt', year='2014', description='computational chemistry')}
```

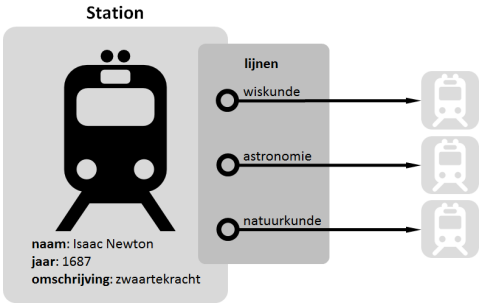
In zijn Novembernummer van 2011 publiceerde NWT Magazine onderstaande *Metrokaart der exacte wetenschap*. De uitdaging van deze opgave bestaat erin om nieuwe gegevenstypes aan Python toe te voegen die toelaten om dergelijke metrokaarten op te stellen, uit te breiden en te bevragen. Een metrokaart is opgebouwd uit een aantal metrolijnen die verschillende metrostations in een vaste volgorde met elkaar verbinden. Het verkeer op een metrolijn verloopt dus in één richting, en bij deze opgave vormen metrolijnen ook nooit een lus. Sommige metrostations kunnen wel op verschillende metrolijnen liggen, zodat daar van de ene op de andere lijn kan overgestapt worden.



Metrokaart der exacte wetenschappen

Opgave

Definieer een klasse `Station` waarmee de metrostations op een metrokaart kunnen voorgesteld worden. Elk metrostation beschikt over de eigenschappen `naam`, `jaar` en `omschrijving`, die telkens verwijzen naar een string (het jaartal kan ook beschreven worden door bv. `6e eeuw v. Chr.`). Vanuit elk metrostation kan je vertrekken naar nul of meer andere metrostations, maar elk van deze uitgaande verbindingen liggen op een verschillende metrolijn. Om deze uitgaande verbindingen voor te stellen, beschikt elk metrostation ook over een eigenschap `lijnen` die verwijst naar een dictionary. Deze dictionary geeft de metrolijnen aan die vertrekken vanuit het station, door telkens de naam van een metrolijn (een string; hier steeds de naam van een exacte wetenschap) af te beelden op het volgende metrostation op die lijn.



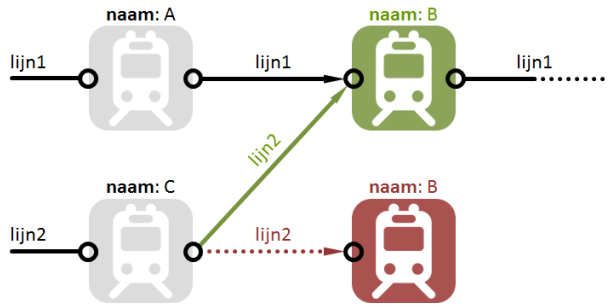
Schematische voorstelling van de objecten van de klasse Station.

De objecten van de klasse Station moeten verder minstens over de volgende methoden beschikken:

- Een initialisatiemethode __init__ met drie parameters: naam, jaar en omschrijving. Aan deze parameters moeten drie strings doorgegeven worden, die telkens moeten toegekend worden aan de gelijknamige eigenschappen van het nieuw aangemaakt object. Voorts moet elk nieuw aangemaakt object ook beschikken over een eigenschap lijnen die initieel verwijst naar een lege dictionary.
- Een methode __eq__ die kan gebruikt worden om te testen of twee metrostations gelijk zijn. Twee metrostations worden enkel als gelijk beschouwd als de waarden van hun eigenschappen naam, jaar en omschrijving gelijk zijn. Merk dus op dat de waarde van de eigenschap lijnen geen invloed heeft op de gelijkheid van metrostations.
- Een methode __repr__ die een stringvoorstelling van het metrostation teruggeeft volgens het formaat Station(naam=*naam*, jaar=*jaar*, omschrijving=*omschrijving*). Hierbij moeten *naam*, *jaar* en *omschrijving* respectievelijk ingevuld worden met de stringvoorstelling van de eigenschappen naam, jaar en omschrijving van het object.
- Een methode __str__ die een stringvoorstelling van het metrostation teruggeeft volgens het formaat *naam* (*jaar*, *omschrijving*). Hierbij moeten *naam*, *jaar* en *omschrijving* respectievelijk ingevuld worden met de waarde van de eigenschappen naam, jaar en omschrijving van het object.
- Een methode verbind die kan gebruikt worden om het metrostation op een gegeven metrolijn te verbinden met een ander metrostation (een object van de klasse Station). De naam van de metrolijn en het andere metrostation waarmee de verbinding moet gemaakt worden, moeten als argumenten aan de methode doorgegeven worden. De methode moet een AssertionError opwerpen met de boodschap station is reeds verbonden op lijn *naam* indien het metrostation reeds een uitgaande verbinding heeft voor de opgegeven metrolijn. Hierbij moet op de plaats van *naam* in de boodschap uiteraard de naam van de metrolijn ingevuld worden.
- Een methode volgende waaraan de naam van een metrolijn moet doorgegeven worden. De methode moet het metrostation teruggeven waarmee het metrostation verbonden is op de aangegeven metrolijn. Indien het metrostation op de gegeven metrolijn niet verbonden is met een ander metrostation, dan moet de methode de waarde None teruggeven.

Definieer een klasse Metrokaart waarmee metrokaarten kunnen voorgesteld worden. Metrokaarten zijn opgebouwd uit een aantal metrolijnen die verschillende metrostations in een vaste volgorde met elkaar verbinden. Elke metrokaart beschikt over een eigenschap lijnen, die verwijst naar een dictionary. Deze dictionary beeldt voor elke metrolijn op de metrokaart de naam van de lijn (een string; hier steeds de naam van een exacte wetenschap) af op het beginstation van de metrolijn (een object van de klasse Station). De objecten van de klasse Metrokaart moeten verder minstens over de volgende methoden beschikken:

- Een initialisatiemethode __init__ die ervoor moet zorgen dat elke nieuw aangemaakte metrokaart een eigenschap lijnen heeft die initieel verwijst naar een lege dictionary.
- Een methode beginstation die het beginstation (een object van de klasse Station) van een gegeven metrolijn teruggeeft. De naam van de metrolijn moet als argument aan de functie doorgegeven worden. Indien de metrokaart geen metrolijn heeft met de opgegeven naam, dan moet de methode de waarde None teruggeven.
- Een methode eindstation die het eindstation (een object van de klasse Station) van een gegeven metrolijn teruggeeft. De naam van de metrolijn moet als argument aan de functie doorgegeven worden. Indien de metrokaart geen metrolijn heeft met de opgegeven naam, dan moet de methode de waarde None teruggeven.
- Een methode uitbreiden waarmee een gegeven metrolijn kan uitgebreid worden met een nieuw metrostation. De naam van de metrolijn en het nieuwe metrostation (een object van de klasse Station) moeten als argumenten aan de methode doorgegeven worden. Het uitbreiden van een metrolijn gebeurt door het eindstation van de metrolijn te verbinden met het nieuwe metrostation. Indien de metrokaart nog geen metrolijn bevatte met de opgegeven naam, dan moet het nieuwe metrostation het beginstation van de metrolijn worden. Maar let op: een metrokaart mag nooit twee metrostations bevatten met dezelfde naam. Indien de metrokaart reeds een metrostation heeft met dezelfde naam als het nieuwe metrostation, dan moet het eindstation van de gegeven metrolijn verbonden worden met het bestaande station.Veronderstel onderstaande situatie waarbij op lijn1 reeds een metrostation voorkomt met naam B, en we lijn2 (met eindstation C) willen uitbreiden met een metrostation met naam B. Dan moet metrostation C verbonden worden met het bestaande metrostation (hier aangeven in het groen) en moet er geen nieuw metrostation aan de metrokaart toegevoegd worden (hier aangegeven in het rood).



Hint: om te vermijden dat er stations met dezelfde naam voorkomen op een metrokaart, kunnen objecten van de klassen Metrokaart een dictionary bijhouden, die de naam van de stations op de kaart afbeeldt op de corresponderende objecten van de klasse Station.

- Een methode stationsToevoegen waaraan de locatie van een tekstbestand moet doorgegeven worden. Elke regel van dit bestand moet de volgende vier velden bevatten, telkens van elkaar gescheiden door één enkele tab: *i*) de naam van een exacte wetenschap, *ii*) de naam van een wetenschapper, *iii*) een jaaraanduiding en *iv*) een korte omschrijving van de belangrijkste verwezenlijkingen van de wetenschapper. De methode moet voor elke regel de metrolijn aangeduid met de naam van de exacte wetenschap uitbreiden met een metrostation waarvan de naam, het jaar en de omschrijving gebaseerd zijn op de laatste drie velden. Deze uitbreidingen moeten gebeuren in de volgorde waarin de regels in het bestand voorkomen.
- Zorg ervoor dat de initialisatiemethode een optionele parameter stations heeft, waaraan de locatie van een tekstbestand kan doorgegeven worden. Als er een waarde doorgegeven wordt voor deze parameter, dan moet bij initialisatie de metrokaart reeds uitgebreid worden met de volgorde en

beschrijving van de metrostations zoals ze in het tekstbestand vervat zitten. Dit doe je uiteraard door hiervoor de methode `stationsToevoegen` aan te roepen.

Zorg er bij de implementatie van de klassen `Station` en `Metrokaart` voor dat je telkens optimaal gebruik maakt van de methoden die je reeds eerder geïmplementeerd hebt.

Voorbeeld (klasse `Station`)

```
>>> station1 = Station('James Watson', '1953', 'opheldering DNA-structuur')
>>> station1.naam
'James Watson'
>>> station1.jaar
'1953'
>>> station1.omschrijving
'opheldering DNA-structuur'
>>> station1.lijnen
{}
>>> station1 == station1
True
>>> station1
Station(naam='James Watson', jaar='1953', omschrijving='opheldering DNA-structuur')

>>> station2 = Station(naam='Ernst Mayr', jaar='1942', omschrijving='biologische soorten')
>>> print(station2)
Ernst Mayr (1942, biologische soorten)
>>> station1 == station2
False

>>> station1.verbind('biologie', station2)
>>> station1.lijnen
{'biologie': Station(naam='Ernst Mayr', jaar='1942', omschrijving='biologische soorten')}
>>> station1.volgende('biologie')
Station(naam='Ernst Mayr', jaar='1942', omschrijving='biologische soorten')
>>> print(station1.volgende('chemie'))
None

>>> station3 = Station('Marshall Warren Nirenberg', '1962', 'genetische code')
>>> station4 = Station('Ilya Prigogine', '1955', 'zelf-organisatie en niet-evenwicht')
>>> station1.verbind('chemie', station3)
>>> station1.verbind('chemie', station4)
Traceback (most recent call last):
AssertionError: station is reeds verbonden op lijn chemie
>>> station1.verbind('genomica', station4)
>>> station1.lijnen
{'biologie': Station(naam='Ernst Mayr', jaar='1942', omschrijving='biologische soorten'), 'genomica': Station(naam='Ilya Prigogine', jaar='1955', omschrijving='zelf-organisatie en niet-evenwicht'), 'c
>>> print(station1.volgende('chemie'))
Marshall Warren Nirenberg (1962, genetische code)
>>> print(station1.volgende('genomica'))
Ilya Prigogine (1955, zelf-organisatie en niet-evenwicht)
```

Voorbeeld (klasse `Metrokaart`)

Bij onderstaande voorbeeldsessie gaan we ervan uit dat het tekstbestand mijlpalen.txt zich in de huidige directory bevindt.

```
>>> kaart = Metrokaart()
>>> kaart.uitbreiden('genomica', Station('James Watson', '1953', 'opheldering DNA-structuur'))
>>> kaart.uitbreiden('genomica', Station('Marshall Warren Nirenberg', '1962', 'genetische code'))
>>> kaart.uitbreiden('genomica', Station(naam='Frederick Sanger', jaar='1977', omschrijving='sequencing'))
>>> kaart.beginstation('genomica')
Station(naam='James Watson', jaar='1953', omschrijving='opheldering DNA-structuur')
>>> kaart.eindstation('genomica')
Station(naam='Frederick Sanger', jaar='1977', omschrijving='sequencing')
>>> kaart.uitbreiden('chemie', Station('Robert Boyle', '1661', 'The Sceptical Chymist'))
>>> kaart.beginstation('chemie')
Station(naam='Robert Boyle', jaar='1661', omschrijving='The Sceptical Chymist')
>>> kaart.eindstation('chemie')
Station(naam='Robert Boyle', jaar='1661', omschrijving='The Sceptical Chymist')

>>> kaart2 = Metrokaart('mijlpalen.txt')
>>> kaart2.beginstation('chemie')
Station(naam='Robert Boyle', jaar='1661', omschrijving='The Sceptical Chymist')
>>> kaart2.eindstation('chemie')
Station(naam='Paul Crutzen', jaar='1985', omschrijving='atmosfeerchemie')

>>> station = kaart2.beginstation('genomica')
>>> station
Station(naam='James Watson', jaar='1953', omschrijving='opheldering DNA-structuur')
>>> station.volgende('biologie')
Station(naam='Ernst Mayr', jaar='1942', omschrijving='biologische soorten')
>>> print(station.volgende('genomica'))
Marshall Warren Nirenberg (1962, genetische code)
>>> print(station.volgende('chemie'))
Ilya Prigogine (1955, zelf-organisatie en niet-evenwicht)
>>> station.volgende('genomica').volgende('genomica')
Station(naam='Frederick Sanger', jaar='1977', omschrijving='sequencing')

>>> kaart2.uitbreiden('chemie', Station('Martin Karplus', '2013', 'computationele chemie'))
>>> kaart2.uitbreiden('informatica', Station('Martin Karplus', '2013', 'computationele chemie'))
>>> station = kaart2.eindstation('chemie')
>>> station
Station(naam='Martin Karplus', jaar='2013', omschrijving='computationele chemie')
>>> kaart2.uitbreiden('chemie', Station('Michael Levitt', '2014', 'computationele chemie'))
>>> kaart2.uitbreiden('informatica', Station('Arieh Warshel', '2014', 'computationele chemie'))
>>> station.lijnen
```

{'informatica': Station(naam='Arieh Warshel', jaar='2014', omschrijving='computationele chemie'), 'chemie': Station(naam='Michael Levitt', jaar='2014', omschrijving='computationele chemie')}