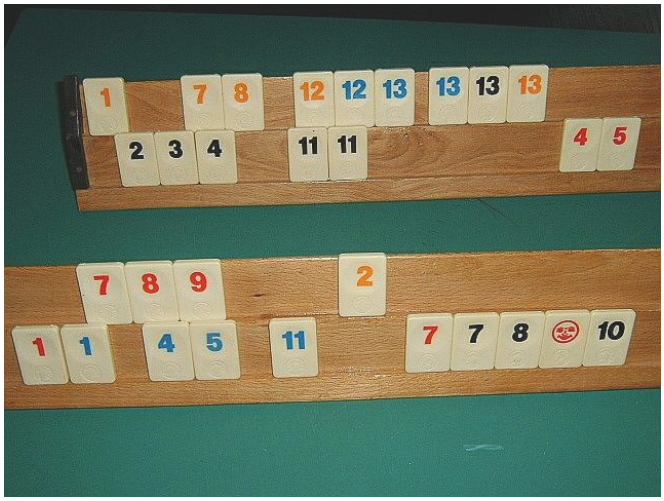
Rummikub

Historically, the Rummikub game was played with two sets of cards. Because this required quite some space on the table, modern versions of the game replaced cards by 106 smaller tiles: two sets of tiles that range in value from one to thirteen, in four colors (red, yellow, blue and black), and two joker tiles ($2 \times 13 \times 4 + 2 = 106$).

At the start of the game, the pool of tiles are shuffled together and spread out face down across the table so that the face of the tiles is not reveiled to the players. Each player draws and reveals one tile. The player whose tile has the highest number value will start the game. Tiles are returned to the pool, and players in turn collect 14 random tiles and arrange them on their racks. Play then begins with the starting player, and proceeds in a clockwise direction.



In the Rummikub game, players each have a rack (container) to store tiles, without revealing the face of the tiles to the other players.

For a player's first move, he must play a set with a value of at least 30 points. Point values are taken from the face value of each tile played, with the joker (if played) assuming the value of the tile it is being used in place of. The player may not use other players' tiles to make the initial 30 or more. A player's first move is known as the *initial meld*. If a player cannot make an initial meld, he must pick up a single tile from the pool and add it to his rack. Play then proceeds to the next player. Game play continues until a player has used all of the tiles in the rack, at which point he should call out "Rummikub" and is declared the winner.

Assignment

In this assignment we represent a **tile** in the Rummikub game as a string. This string starts with an integer from the range $[1, 13]$ that represents the value of the tile, followed by a single uppercase letter that represents its color: `R` (red), `Y` (yellow), `B` (blue) or `K` (black).

A **group of tiles** represents for example all tiles a player has on his rack, or an initial meld of tiles that is played on the table by a player. We use two distinct ways to represent a group of tiles: as a sequence (a list or a tuple) or as a dictionary.

The first way represents a group of tiles as a sequence (a list or a tuple) of tiles. Because the same tile might occur more than once in the group — we even allow the same tile to occur more than twice — it may happen that the same string occurs more than once in the sequence that

represents the group of tiles.

De second way represents a group of tiles as a dictionary that maps tiles onto the number of occurrences of tiles in the group. The string representation of the tiles is thus used as a key in the dictionary, and the number of occurrences as the corresponding values. All values in the dictionary are strictly positive. Tiles that do not occur in the group are never used as keys in the dictionary. Your task:

- Write a function group that takes a sequence (a list or a tuple) of strings that represents a group of tiles. The function must return a dictionary that represents the given group of tiles.
- Write a function play that takes two dictionaries. The first dictionary represent an initial meld that a player wants to put on the table. The second dictionary represents all tiles the player has on his rack. The function must return a Boolean value that indicates whether or not the player can put the initial meld on the table. The latter is the case if the player has all the tiles in the initial meld on his rack. If the play can put the initial meld on the table, the function must make sure that all tiles from the initial meld are removed from the dictionary that represents the tiles on the rack. If the player cannot put the initial meld on the table, the function may not modify the given dictionary that represents the tiles the player has on his rack.

Example

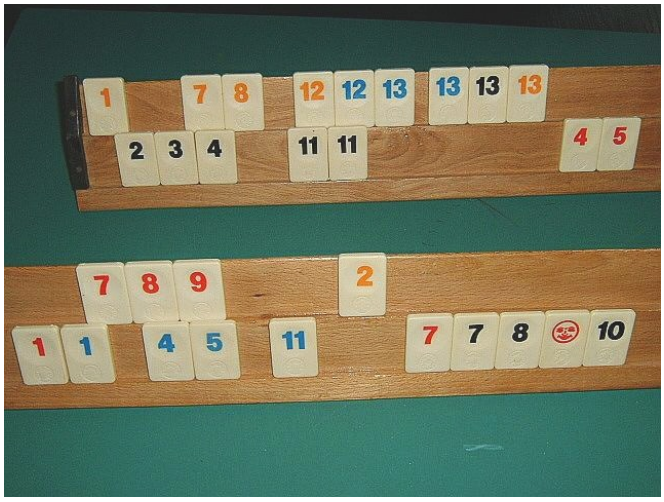
```
>>> group(['9Y', '7R', '9Y', '7R', '9Y', '7R', '9Y'])
{'9Y': 4, '7R': 3}
>>> group(['1Y', '7Y', '8Y', '12Y', '12B', '13B', '13B', '13Z', '13Y', '2Z', '3Z', '4Z', '11Z', '11Z', '4R', '5R'])
{'8Y': 1, '12Y': 1, '13B': 2, '3Z': 1, '5R': 1, '7Y': 1, '1Y': 1, '13Y': 1, '4Z': 1, '13Z': 1, '12B': 1, '2Z': 1, '4R': 1, '11Z': 2}

>>> meld = group(['13B', '13Z', '13Y'])
>>> rack = group(['1Y', '7Y', '8Y', '12Y', '12B', '13B', '13B', '13Z', '13Y', '2Z', '3Z', '4Z', '11Z', '11Z', '4R', '5R'])
>>> play(meld, rack)
True
>>> meld
{'13B': 1, '13Y': 1, '13Z': 1}
>>> rack
{'8Y': 1, '12Y': 1, '13B': 1, '3Z': 1, '5R': 1, '7Y': 1, '1Y': 1, '4Z': 1, '12B': 1, '2Z': 1, '4R': 1, '11Z': 2}

>>> meld = group(['12B', '12Z', '12Y'])
>>> play(meld, rack)
False
>>> rack
{'8Y': 1, '12Y': 1, '13B': 1, '3Z': 1, '5R': 1, '7Y': 1, '1Y': 1, '4Z': 1, '12B': 1, '2Z': 1, '4R': 1, '11Z': 2}
```

Rummikub is een gezelschapsspel dat oorspronkelijk werd gespeeld met twee sets kaarten. Omdat daarvoor vrij veel ruimte op de tafel nodig was, zijn de kaarten in moderne versies van het spel vervangen door 106 kleinere stenen: twee sets van de getallen 1 tot en met 13 in de kleuren rood-geel-blauw-zwart en twee jokers ($2 \times 13 \times 4 + 2 = 106$).

Bij aanvang van het spel worden alle stenen op hun kop op tafel gelegd, zodat hun waarde niet zichtbaar is. Deze stenen vormen de pot. Daarna neemt iedere speler 14 stenen uit de pot, en zet ze op zijn rekje. Om te bepalen wie mag beginnen, pakt elke speler een extra steen uit de pot en legt die zichtbaar op tafel. De speler wiens steen de hoogste waarde heeft mag beginnen.



Tijdens het spelen van Rummikub plaatsen de spelers stenen uit de pot op hun rekje. Hierbij zorgen ze ervoor dat de waarde van de stenen op hun rekje onzichtbaar blijft voor hun tegenspelers.

Het doel van het spel is om als eerste al je stenen op tafel te leggen door nieuwe rijtjes te vormen en open op tafel uit te leggen, of door rijtjes die open op tafel liggen aan te vullen. Hiervoor moeten de spelers tijdens het spel mogelijks bijkomende stenen uit de pot nemen en op hun rekje plaatsen.

Opgave

In deze opgave stellen we een **steen** van het spel Rummikub voor als een string. Deze string begint met een natuurlijk getal uit het interval $[1, 13]$ die de waarde van de steen voorstelt, gevolgd door één enkele hoofdletter die de kleur van de steen voorstelt: `R` (rood), `G` (geel), `B` (blauw) of `Z` (zwart).

Een **groep stenen** omschrijft bijvoorbeeld alle stenen die een speler op zijn rekje heeft staan, of alle stenen die een speler in één beurt uitlegt vanop zijn rekje naar de tafel. We gebruiken twee manieren om een groep stenen voor te stellen: als een reeks (lijst of tuple) of als een dictionary.

De eerste manier stelt een groep stenen voor als een reeks (een lijst of een tuple) van stenen. Omdat eenzelfde steen meerdere keren in een groep kan voorkomen — we laten hierbij zelfs toe dat eenzelfde steen meer dan twee keer voorkomt — kan het dus voorkomen dat dezelfde string meerdere keren voorkomt in de reeks die de groep stenen voorstelt.

De tweede manier stelt een groep stenen voor als een dictionary die stenen afbeeldt op het aantal voorkomens van die steen in de groep. De stringvoorstelling van de stenen wordt dus gebruikt als sleutel voor de dictionary, en het aantal voorkomens als de bijhorende waarde. Hierbij geldt steeds dat het aantal voorkomens minstens 1 is. Stenen die niet voorkomen in de groep worden nooit als sleutel gebruikt in de dictionary. Gevraagd wordt:

- Schrijf een functie `groep` waaraan een reeks (een lijst of een tuple) van strings moet doorgegeven worden die een groep stenen voorstelt. De functie moet een dictionary teruggeven die de gegeven groep stenen voorstelt.
- Schrijf een functie `uitleggen` waaraan twee dictionaries moeten doorgegeven worden. De eerste stelt een rijtje stenen voor die een speler wil uitleggen. De tweede stelt de stenen voor die de speler op zijn rekje heeft staan. De functie moet een Booleaanse waarde teruggeven die aangeeft of de speler het gegeven rijtje stenen kan uitleggen. Dat is het

geval als alle stenen van het rijtje wel degelijk op het gegeven rekje staan, rekening houdend met het aantal keer dat de stenen voorkomen in het rijtje en op het rekje. Indien de speler het gegeven rijtje kan uitleggen, dan moet de functie er ook voor zorgen dat die stenen weggehaald worden uit de dictionary die de stenen op het rekje voorstelt. Indien de speler het gegeven rijtje niet kan uitleggen, dan mag de functie de gegeven dictionary die de stenen op het rekje voorstelt niet wijzigen.

Voorbeeld

```
>>> groep(['1G', '7G', '8G', '12G', '12B', '13B', '13B', '13Z', '13G', '2Z', '3Z', '4Z', '11Z', '11Z', '4R', '5R'])
{'4R': 1, '8G': 1, '5R': 1, '11Z': 2, '4Z': 1, '12G': 1, '13G': 1, '13Z': 1, '1G': 1, '2Z': 1, '13B': 2, '7G': 1, '3Z': 1, '12B': 1}
>>> groep(['9G', '7R', '9G', '7R', '9G', '7R', '9G'])
{'9G': 4, '7R': 3}

>>> rijtje = groep(['13B', '13Z', '13G'])
>>> rekje = groep(['1G', '7G', '8G', '12G', '12B', '13B', '13B', '13Z', '13G', '2Z', '3Z', '4Z', '11Z', '11Z', '4R', '5R'])
>>> uitleggen(rijtje, rekje)
True
>>> rijtje
{'13B': 1, '13G': 1, '13Z': 1}
>>> rekje
{'8G': 1, '3Z': 1, '12G': 1, '7G': 1, '12B': 1, '2Z': 1, '11Z': 2, '1G': 1, '4Z': 1, '4R': 1, '5R': 1, '13B': 1}

>>> rijtje = groep(['12B', '12Z', '12G'])
>>> uitleggen(rijtje, rekje)
False
>>> rekje
{'8G': 1, '3Z': 1, '12G': 1, '7G': 1, '12B': 1, '2Z': 1, '11Z': 2, '1G': 1, '4Z': 1, '4R': 1, '5R': 1, '13B': 1}
```