Exercise 10: Hierarchical clustering of the grain data

Step 1: Load the dataset *(done for you)*.

In [4]:
```
import pandas as pd

seeds_df = pd.read_csv('seeds-less-rows.csv')

# remove the grain species from the DataFrame, save for later
varieties = list(seeds_df.pop('grain_variety'))

# extract the measurements as a NumPy array
samples = seeds_df.values
```

Step 2: Import:

- `linkage` and `dendrogram` from `scipy.cluster.hierarchy`.
- `matplotlib.pyplot` as `plt`.

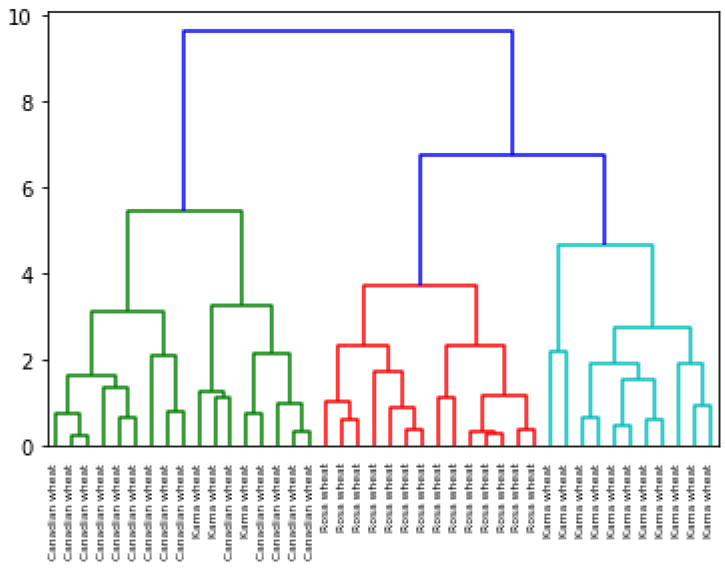
In [5]:
```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
```

Step 3: Perform hierarchical clustering on `samples` using the `linkage()` function with the `method='complete'` keyword argument. Assign the result to `mergings`.

In [31]:
```
mergings=linkage(samples,method='complete')
```

Step 4: Plot a dendrogram using the `dendrogram()` function on `mergings`, specifying the keyword arguments `labels=varieties`, `leaf_rotation=90`, and `leaf_font_size=6`. Remember to call `plt.show()` afterwards, to display your plot.

In [36]:
```
dendrogram(mergings,labels=varieties,leaf_rotation=90,leaf_font_size=6)
plt.show()
```



Exercise 11: Hierarchies of stocks

Previously, you used k-means clustering to cluster companies according to their stock price movements. This time, perform *hierarchical* clustering of the companies. You are given a NumPy array of price movements `movements`, where the rows correspond to companies, and a list of the company names `companies`.

SciPy hierarchical clustering doesn't fit into a sklearn pipeline, so you'll need to use the `normalize()` function from `sklearn.preprocessing` instead of `Normalizer`.

Step 1: Load the data *(written for you)*

In [4]:
```
import pandas as pd

fn = 'company-stock-movements-2010-2015-incl.csv'
stocks_df = pd.read_csv(fn, index_col=0)

companies = list(stocks_df.index)
movements = stocks_df.values
```

Step 2: Make the necessary imports:

- `normalize` from `sklearn.preprocessing`.
- `linkage` and `dendrogram` from `scipy.cluster.hierarchy`.
- `matplotlib.pyplot` as `plt`.

In [5]:
```
from sklearn.preprocessing import normalize
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
```

Step 3: Rescale the price movements for each stock by using the `normalize()` function on `movements`.

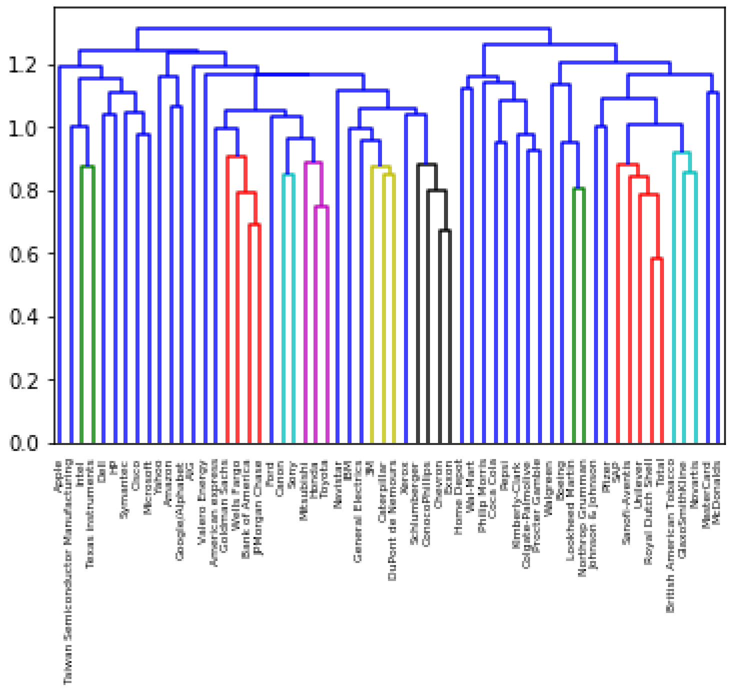
In [7]:
```
normalized_movements = normalize(movements)
```

Step 4: Apply the `linkage()` function to `normalized_movements`, using `'complete'` linkage, to calculate the hierarchical clustering. Assign the result to `mergings`.

In [9]:
```
mergings=linkage(normalized_movements,method='complete')
```

Step 5: Plot a dendrogram of the hierarchical clustering, using the list `companies` of company names as the `labels`. In addition, specify the `leaf_rotation=90`, and `leaf_font_size=10` keyword arguments as you did in the previous exercise.

In [11]:
```
dendrogram(mergings,labels=companies,leaf_rotation=90,leaf_font_size=6)
plt.show()
```



Exercise 12: Which clusters are closest?

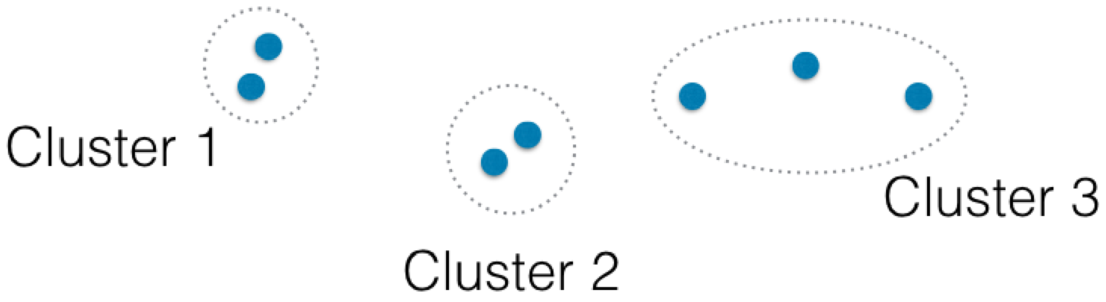
Let's compare now two different linkage methods:

- In **complete** linkage, the distance between clusters is the distance between the *furthest* points of the clusters.
- In **single** linkage, the distance between clusters is the distance between the *closest* points of the clusters.

Consider the three clusters in the following diagram, and answer the question below.

In [ ]:
```
# some magic for showing the image
from IPython.display import Image
Image("../images/cluster_linkage_riddle.png")
```

Out[ ]:



Question:

Which of the following statements are true?

A. In single linkage, cluster 3 is the closest to cluster 2.

B. In complete linkage, cluster 1 is the closest to cluster 2.

Answer: both A and B are true.

Exercise 13: Different linkage, different hierarchical clustering!

Now, perform a hierarchical clustering of the voting countries with `'single'` linkage, and compare the resulting dendrogram with the one in the video. Different linkage, different hierarchical clustering!

First, we need to do a little pre-processing to account for one of the Eurovision rules: countries are not allowed to vote for themselves.

Step 1: Load the DataFrame *(written for you)*

```
In [14]: import pandas as pd

         scores_df = pd.read_csv('eurovision-2016-televoting.csv', index_col=0)
         country_names = list(scores_df.index)
```

Step 2: Display the DataFrame, and have a look. Each row represents a country that *voted*, while each column represents a country that *performed*.

Notice the NaN ("not-a-number") values. These correspond to missing scores in the original CSV file. These scores are missing because countries that performed were not allowed to vote for themselves.

In [15]:
```
scores_df
```

Out[15]:

	Armenia	Australia	Austria	Azerbaijan	Belgium	Bulgaria	Croatia	Cypru
From country								
Albania	2.0	12.0	0.0	0.0	0.0	8.0	0.0	0.0
Armenia	NaN	0.0	4.0	0.0	0.0	0.0	0.0	6.0
Australia	0.0	NaN	3.0	0.0	12.0	10.0	0.0	0.0
Austria	0.0	3.0	NaN	0.0	0.0	5.0	0.0	0.0
Azerbaijan	0.0	2.0	0.0	NaN	0.0	8.0	0.0	0.0
Belarus	7.0	1.0	0.0	8.0	0.0	4.0	0.0	0.0
Belgium	7.0	4.0	3.0	0.0	NaN	5.0	0.0	0.0
Bosnia & Herzegovina	0.0	3.0	5.0	8.0	0.0	2.0	10.0	0.0
Bulgaria	8.0	5.0	4.0	1.0	0.0	NaN	0.0	7.0
Croatia	0.0	5.0	6.0	0.0	0.0	1.0	NaN	0.0
Cyprus	8.0	5.0	0.0	0.0	0.0	12.0	0.0	NaN
Czech Republic	8.0	1.0	4.0	6.0	0.0	5.0	0.0	0.0
Denmark	0.0	10.0	1.0	0.0	8.0	0.0	0.0	0.0
Estonia	0.0	4.0	6.0	0.0	0.0	0.0	0.0	3.0
F.Y.R. Macedonia	7.0	3.0	0.0	0.0	4.0	10.0	5.0	0.0
Finland	0.0	7.0	6.0	0.0	0.0	4.0	0.0	1.0
France	12.0	0.0	8.0	0.0	4.0	5.0	0.0	0.0
Georgia	12.0	1.0	0.0	7.0	0.0	3.0	0.0	0.0
Germany	2.0	5.0	7.0	0.0	0.0	4.0	0.0	0.0
Greece	8.0	5.0	1.0	0.0	0.0	7.0	0.0	12.0
Hungary	0.0	3.0	6.0	0.0	0.0	4.0	0.0	5.0
Iceland	0.0	8.0	2.0	1.0	3.0	0.0	0.0	0.0
Ireland	0.0	6.0	1.0	0.0	0.0	5.0	0.0	0.0
Israel	6.0	5.0	3.0	2.0	1.0	7.0	0.0	0.0
Italy	1.0	0.0	0.0	0.0	0.0	7.0	0.0	6.0
Latvia	0.0	6.0	4.0	3.0	0.0	1.0	0.0	0.0
Lithuania	0.0	5.0	1.0	0.0	0.0	0.0	0.0	0.0

	Armenia	Australia	Austria	Azerbaijan	Belgium	Bulgaria	Croatia	Cypru
From country								
Malta	0.0	12.0	0.0	6.0	0.0	8.0	0.0	0.0
Moldova	7.0	5.0	4.0	8.0	0.0	2.0	0.0	0.0
Montenegro	0.0	0.0	0.0	7.0	0.0	5.0	6.0	0.0
Norway	0.0	8.0	0.0	0.0	2.0	5.0	0.0	0.0
Poland	2.0	7.0	4.0	0.0	0.0	3.0	0.0	1.0
Russia	12.0	4.0	8.0	6.0	0.0	0.0	0.0	7.0
San Marino	2.0	5.0	0.0	0.0	0.0	3.0	0.0	0.0
Serbia	2.0	6.0	0.0	0.0	5.0	8.0	4.0	3.0
Slovenia	0.0	3.0	6.0	0.0	0.0	2.0	8.0	0.0
Spain	6.0	4.0	2.0	0.0	0.0	12.0	0.0	0.0
Sweden	0.0	12.0	5.0	0.0	0.0	4.0	0.0	0.0
Switzerland	0.0	1.0	10.0	0.0	0.0	0.0	0.0	0.0
The Netherlands	8.0	5.0	6.0	0.0	12.0	1.0	0.0	0.0
Ukraine	7.0	4.0	0.0	10.0	0.0	2.0	0.0	0.0
United Kingdom	0.0	6.0	0.0	0.0	0.0	8.0	0.0	2.0

42 rows × 26 columns

Step 3: Fill in the NaNs with the highest possible score (12) - we are assuming that countries would vote for themselves, if they had been allowed to do so. *(This bit written for you).*

```
In [16]: scores_df=scores_df.fillna(12)
```

Step 4: Import the `normalize` function from `sklearn.preprocessing`.

```
In [17]: from sklearn.preprocessing import normalize
```

Step 5: Apply the normalize function to `scores_df.values`, assigning the result to `samples`.

(Why do we need to normalize? Because now that the missing values are filled with 12 points, some countries (those that performed) given a greater total number of points when voting. The `normalize` function corrects for this.)

In [18]:
```
samples= normalize(scores_df)
```

Step 6: Import:

- `linkage` and `dendrogram` from `scipy.cluster.hierarchy`.
- `matplotlib.pyplot` as `plt`.

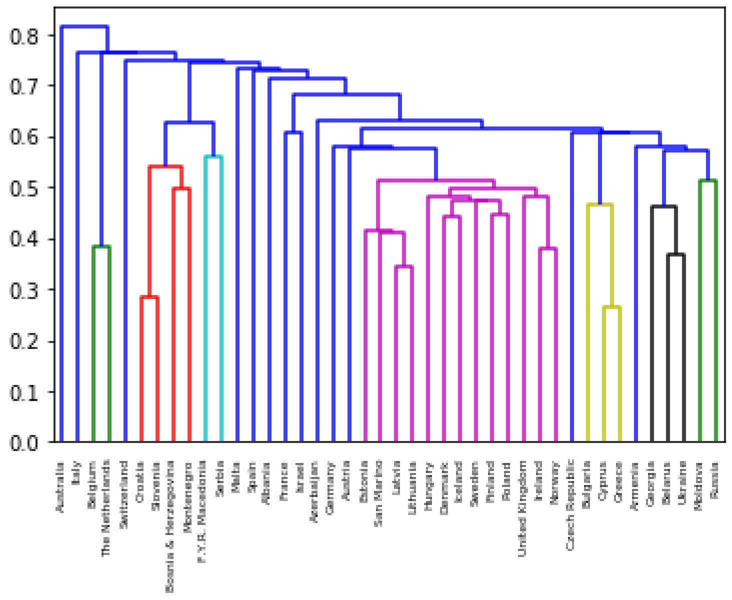
In [19]:
```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
```

Step 7: Perform hierarchical clustering on `samples` using the `linkage()` function with the `method='single'` keyword argument. Assign the result to `mergings`.

In [20]:
```
mergings=linkage(samples,method='single')
```

Step 8: Plot a dendrogram of the hierarchical clustering, using the list `country_names` as the `labels`. In addition, specify the `leaf_rotation=90`, and `leaf_font_size=6` keyword arguments as you have done earlier.

In [21]:
```
dendrogram(mergings,labels=country_names,leaf_rotation=90,leaf_font_size=6)
plt.show()
```



Step 9: Compare your dendrogram above to the one in the slides and notice that different linkage functions give different hierarchical clusterings.

Both the linkage functions we've considered, "complete" and "single", have advantages and disadvantages. In practice, just try both out, and see which dendrogram seems more sensible.

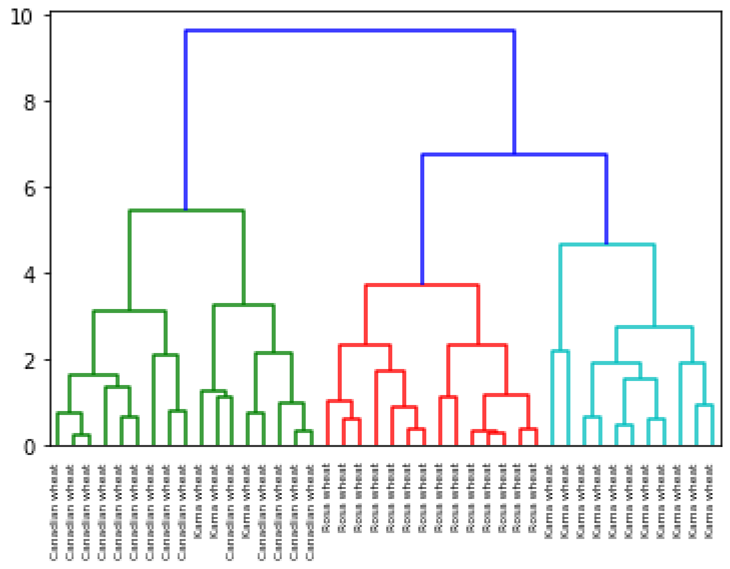
Exercise 14: Intermediate clusterings - how many clusters?

Consider the dendrogram below - it is the result of your hierarchical clustering of some of the grain samples.

Question: If the hierarchical clustering were stopped at height 6 on the dendrogram, how many clusters would there be?

Hint: Imagine a horizontal line at this height.

In [3]:
```
import pandas as pd

seeds_df = pd.read_csv('seeds-less-rows.csv')

# remove the grain species from the DataFrame, save for later
varieties = list(seeds_df.pop('grain_variety'))

# extract the measurements as a NumPy array
samples = seeds_df.values

from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

mergings = linkage(samples, method='complete')

dendrogram(mergings,
           labels=varieties,
           leaf_rotation=90,
           leaf_font_size=6,
)
plt.show()
```



Answer: 3

Exercise 15: Extracting the cluster labels

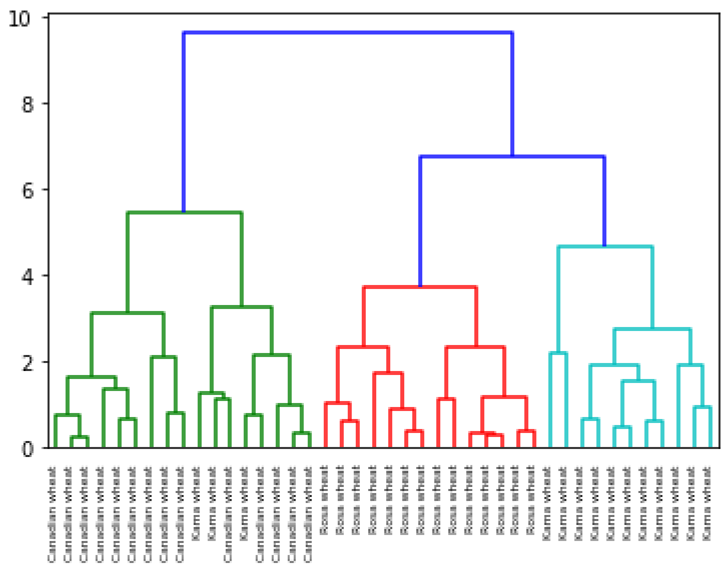
In the previous exercise, you saw that the intermediate clustering of the grain samples at height 6 has 3 clusters. Now, use the `fcluster()` function to extract the cluster labels for this intermediate clustering, and compare the labels with the grain varieties using a cross-tabulation.

Step 1: Load the dataset: *(written for you)*

```
In [1]: import pandas as pd

        seeds_df = pd.read_csv('seeds-less-rows.csv')

        # remove the grain species from the DataFrame, save for later
        varieties = list(seeds_df.pop('grain_variety'))

        # extract the measurements as a NumPy array
        samples = seeds_df.values
```

Step 2: Run the hierarchical clustering of the grain samples that you worked out earlier *(filled in here for you).*

```
In [2]: from scipy.cluster.hierarchy import linkage, dendrogram
        import matplotlib.pyplot as plt

        mergings = linkage(samples, method='complete')

        dendrogram(mergings,
                   labels=varieties,
                   leaf_rotation=90,
                   leaf_font_size=6,
        )
        plt.show()
```



Step 3: Import `fcluster` from `scipy.cluster.hierarchy`.

```
In [3]: from scipy.cluster.hierarchy import fcluster
```

Step 4: Obtain a flat clustering by using the `fcluster()` function on `mergings`. Specify a maximum height of `6` and the keyword argument `criterion='distance'`. Assign the result to `labels`.

```
In [7]: labels=fcluster(mergings,6,criterion='distance')
```

Step 5: Create a DataFrame `df` with two columns named `'labels'` and `'varieties'`, using `labels` and `varieties`, respectively, for the column values.

```
In [8]: df=pd.DataFrame({'labels':labels,'varieties':varieties})
```

Step 6: Create a cross-tabulation `ct` between `df['labels']` and `df['varieties']` to count the number of times each grain variety coincides with each cluster label.

```
In [12]: ct=pd.crosstab(df['labels'],df['varieties'])
```

Step 7: Display `ct` to see how your cluster labels correspond to the wheat varieties.

```
In [13]: ct
```

Out[13]:

varieties	**Canadian wheat**	**Kama wheat**	**Rosa wheat**
labels			
1	14	3	0
2	0	0	14
3	0	11	0