1) One of the most important operations that is implemented by speech recognition devices is *POS tagging*. It consists in tagging each part of a sentence with the role of that term in the sentence (e.g., noun, verb, or modal).

   To this aim, data analysis algorithms working on huge sets of data are used to produce *Emission* and *Transition* Probabilities.

   Given two roles **R1** (e.g., noun) and **R2** (e.g., verb) the transition probability **p(R1, R2)** is the probability that a term of type **R1** is followed by a term of type **R2**.

   Instead, given a term **T** and a role **R**, the emission probability **q(T, R)** is the probability that the term **T** is playing the role **R**.

   For example, we may have the following transition matrix

	Noun	**Modal**	**Verb**	**End**
Start	3/4	1/4	0	0
Noun	1/9	3/9	1/9	4/9
Modal	1/4	0	3/4	0
Verb	1	0	0	0

   meaning that the first term of the sentence with probability **3/4** is a noun and with probability **1/4** is a modal. After a noun there is another noun with probability **1/9**, there is a modal with probability **3/9**; there is a verb with probability **1/9**, and the sentence ends with probability **4/9**. After a modal, instead, with probability **1/4** there is a noun, and with probability **3/4** there is a verb. Finally, after a verb there is a noun with probability **1**.

   Similarly, we may have the following emission probabilities

	Noun	**Modal**	**Verb**
Will	1/4	3/4	0
Mary	1	0	0
Spot	1/2	0	1/2
Jane	1	0	0

   meaning that 'Will' is a noun with probability **1/4** and a modal with probability **3/4**, 'Spot' is a noun with probability **1/2** and a verb with probability **1/2**, and 'Mary' and 'Jane' are surely nouns.

Once the speech recognition device receives the sentence "Will Mary spot Jane?", it may assign different roles to each term.

For example, it may assign roles Noun - Noun - Verb - Noun or roles Modal - Noun - Verb - Noun. Each assignment has a probability to be the right one depending on the transition and emission probabilities.

For example, according to the transition and emission probabilities given above, the the probability that the first assignment of roles is the right one is equal to

3/4 (the sentence starts with a noun) * **1/4** ('Will' is a noun) * **1/9** (a noun is followed by another noun) * **1** ('Mary' is a noun) * **1/9** (a noun is followed by a verb) * **1/2** ('Spot' is a verb) * **1** (a verb is followed by a noun) * **1** ('Jane' is a noun) * **4/9** (the sentence ends with a noun) = **1/1944.**

The second assignment of roles, instead, is likely to be the right one with probability

1/4 (the sentence starts with a modal) * **3/4** ('Will' is a modal) * **1/4** (a modal is followed by a noun) * **1** ('Mary' is a noun) * **1/9** (a noun is followed by a verb) * **1/2** ('Spot' is a verb) * **1** (a verb is followed by a noun) * **1** ('Jane' is a noun) * **4/9** (the sentence ends with a noun) = **1/864**

Hence, we can conclude that the second assignment is more likely to be the right one.

Design a function **pos_tagging(R, S, T, E)** that takes in input

- a tuple **R** of roles,
- a tuple **S** of strings,
- a dictionary **T** whose keys are the roles in **R** plus the special role **Start** and values are dictionaries **T[r]** such that
    - the keys of **T[r]** are the roles in **R** plus the special role **End**
    - the values of **T[r]** are the transition probabilities between **r** and the corresponding role defined by the key
- a dictionary **E** whose keys are the strings in **S** and value are dictionaries **E[s]** such that
    - the keys of **E[s]** are the roles in **R**
    - the values in **E[s]** are the emission probabilities between **s** and the corresponding role defined by the key

The function returns a dictionary whose keys are the words in **S** and the values are the roles assigned to these words, so that the selected assignment is the one of maximum likelihood.

The test file provided with the homework provides some examples of data on which your code can be tested. Notice that in grading your projects they will be run against data different from the test one. The 15 projects that will return the correct result against this new data in the shortest time will receive a bonus point.

2) You are asked to test several speech recognition devices in order to choose the one giving the right performances. Some preliminary tests on these devices have been run, but more massive tests are necessary. Unfortunately, these further tests are very expensive and time-consuming, and thus we would like to choose a small subset of "more relevant" devices and we would like to run tests only on the selected devices. Clearly, we would like to select devices to test so that the best device is included in the test.

   Specifically, we are given **N** devices that have been preliminarily tested for **D** days. For each device and for each day, we collected the average number of times in which the device has correctly recognized the sentence given in input.

   We tested devices for different lengths of the sentence, going from 3-term sentences up to **X**-term sentences. For example if **N=3** and **X=4**, we could have collected that device 1 recognized in average 38 3-term sentences and 35 4-term sentences for each day, while device 2 and device 3 have performances respectively of 40 on 3-term sentences and 29 on 4-term sentences, and of 37 on 3-term sentences and 34 on 4-term sentences.

   Note that, given the data for two devices **A** and **B**, it is possible that the performances of one device, say **A**, are strictly better than the performances of the other device, say **B**, on all possible sentence lengths (as is the case between device 1 and device 3 in the example above). In this case we say that **A** *dominates* **B** and we could eliminate **B** from our test.

   Anyway, it is not sufficient to select only those devices that are not dominated by any remaining device. Indeed, we are also required to provide for each selected device a *fallback* alternative (a device that should be tested in place of the selected one in the case that the latter is not available at the testing time). For this reason, we need to partition the **N** devices in subsets such that each subset enjoys the following *no-interleaving property*: if **A** and **B** are included in the same subset, then either **A** dominates **B** or vice versa. In this way, for each subset it is possible to define a rank of the devices in the subset such that the device with rank **r** dominates all devices with rank **r' > r**, and thus the device with rank **r** can be safely chosen as fallback device in case all the devices with rank **r' < r** are unavailable.

   For example, for the setting above we may group device 1 and device 3 in the same subset, and device 2 in another subset.

   As another example consider **N = 5** devices, a maximum length of sentences **X = 7** and the following collected data:

	3-term sentences	**4-term sentences**	**5-term sentences**	**6-term sentences**	**7-term sentences**
Device 1	100	99	85	77	63
Device 2	101	88	82	75	60

Device 3	98	89	84	76	61
Device 4	110	65	65	67	80
Device 5	95	80	80	63	60

Then, we can partition the devices in three subsets: the first subset is championed by Device 1 followed by Device 3, and then by Device 5; the second subset contains only Device 2; the third subset contains only Device 4.

However, it is not hard to check that it is impossible to partition devices in less than three classes. Indeed Device 2 is not dominated and does not dominate each of the remaining devices (since in all cases it is not better either on 3-term or on 7-term sentences and it is better either on 3-term or on 4-term sentences), and similarly Device 4 is not dominated and does not dominate each of the remaining devices.

Design a class **DeviceSelection** with the following interface:

- **DeviceSelection(N, X, data)**, where **N** is a tuple of strings identifying the devices, **X** is an integer, and **data** is a dictionary whose keys are the elements of **N**, and whose values are tuples of **X-2** elements describing the performances of the corresponding device over sentences from 3-term to **X**-term;
- **countDevices()**, that returns the minimum number **C** of devices for which we need to run the expensive tests. That is, **C** is the number of subsets in which the devices are partitioned so that every subset satisfies the non-interleaving property;
- **nextDevice(i)**, that takes in input an integer **i** between **0** and **C-1**, and returns the string identifying the device with highest rank in the **i**-th subset that has been not returned before, or **None** if no further device exists (e.g., the first call of **nextDevice(0)** returns the device with the highest rank in the first subset, i.e., the one that dominates all the remaining devices in this subset, the second call returns the device with the second highest rank, and so on). The method throws an exception if the value in input is not in the range **[0, C-1]**.

The test file provided with the homework provides some examples of data on which your code can be tested. Notice that in grading your projects they will be run against data different from the test one. The 15 projects that will return the correct result against this new data in the shortest time will receive a bonus point.

HINT: it may be useful to represent devices on a bipartite graph with **N** nodes on each side, with an edge between node **u** and node **v** only if **u** represents a device that dominates the device represented by **v**.