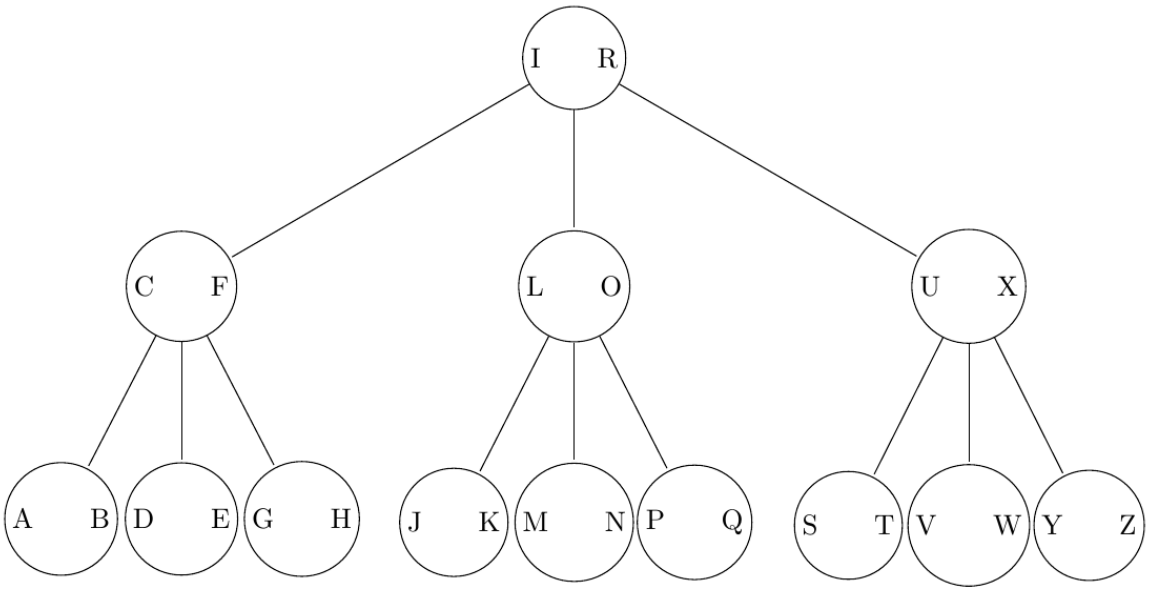
1. Given a **(2,8)-Tree T** saving **Currency** objects, an integer **k**, and two currency codes $\mathbf{c_1}$ and $\mathbf{c_2}$, a ***(k, c_1, c_2)-cover*** of **T** is a set of nodes of **T** that jointly contain at least **k** currencies whose code is between $\mathbf{c_1}$ and $\mathbf{c_2}$ (extremes included).

   E.g., let **T** be the following tree, **k=4**, $\mathbf{c_1}$ **= B** and $\mathbf{c_2}$ **= F**.



   The following are valid **(k, $\mathbf{c_1}$, $\mathbf{c_2}$)**-cover of **T**:
   - **(A, B), (C, F), (D, E)**
   - **(C, F), (D, E)**

   Instead, **(A, B), (D, E)** is not a valid **(k, $\mathbf{c_1}$, $\mathbf{c_2}$)**-cover of **T** because it covers only 3 currencies whose code is between $\mathbf{c_1}$ and $\mathbf{c_2}$.

   Implement a greedy algorithm that tries to compute the **(k, $\mathbf{c_1}$, $\mathbf{c_2}$)**-cover of **T** with the minimum number of nodes. The algorithm must return **None** if there are less than **k** currencies in the tree whose code is between $\mathbf{c_1}$ and $\mathbf{c_2}$.

   Evaluate the running time of the proposed algorithm.

   Does the algorithm always return the optimal solution? If not, can you bound the approximation ratio of the algorithm (you can both evaluate this bound with a formal analysis, or experimentally by testing the algorithm on a large number of trees).

2. Implement an algorithm that, given in input a **Currency** object and a float number **r** with at most two decimal points, returns

   - the number of different ways that value **r** can be achieved by using denominations of the given currency;
   - the list of different changes of the value **r** that can be achieved by using denominations of the given currency.

   E.g., for a currency whose denominations are **{0.1, 0.2, 0.5}** and **r=0.6**, the algorithm must return **5**, and the five different possible ways that can be used for changing 0.6, i.e., **0.5+0.1**, **3*0.2**, **2*0.2+2*0.1**, **0.2+4*0.1**, **6*0.1**.

   The algorithm must run in time order of **r** * the number of the denominations of the **Currency** object taken in input.

3. Given two currencies **u** and **v**, if the rate exchange among these currencies is **r(u, v)** then one unit of currency **u** is equivalent to $\mathbf{e^{-r(u, v)}}$ units of currency **v**. Note that the rate exchange of **(u, v)** is different from the rate exchange of **(v, u)**, i.e., **r(u, v) != r(v, u)**. Note also that the rate exchange **r(u, v)** can be positive (meaning that for exchanging one unit of **u** we need at most one unit of **v**) or negative (meaning that for exchanging one unit of **u** we need more than one unit of **v**).

   E.g., if we are given currencies with code **USD**, **EUR**, **GBP** and rate exchanges **r(USD, EUR)=0,3**, **r(EUR, GBP)=-0,31**, and **r(GBP, USD)=0,005**, then **1 USD** corresponds to **0,74 EUR**, **1 EUR** corresponds to **1,36 GBP**, **1 GBP** corresponds to **0,995 USD**. Thus, if **1 USD** is exchanged in **EUR** and then the obtained money is exchanged in **GBP**, than one achieves **1,006 GBP** = $\mathbf{e^{-0,01} = e^{-[r(USD, EUR)+r(EUR,GBP)]}}$.

   We say that an *arbitrage opportunity* for a given currency **s** exists if there exists a sequence of changes starting and terminating with **s** (i.e., a cycle) such that the total exchange rate of the sequence is negative. For example, with the values above, there exists an arbitrage opportunity for **USD**, since **r(USD, EUR) + r(EUR, GBP) + r(GBP, USD) = -0,005** (and, indeed, by exchanging **1 USD** one receives **1,02 USD**).

   Design an algorithm that takes in input
   - a set **C** of **Currency** objects such that, for every object **c** in **C**, the attribute **Change** contains the rate exchange of **c** against every other currency in **C** different from **c**;
   - a specific **Currency** object **s**.

   The algorithm must return **Null** if **s** is not in **C**. If **s** belongs to **C**, then the algorithm returns **False** if there is not an arbitrage opportunity for **s** within the given set of currencies, and otherwise it returns a cycle that witnesses the arbitrage opportunity for **s**.

4. Consider a set **C** of currencies. Differently from the previous point, here we allow that the rate exchange among two currencies **c, c'** in **C** could not exist (i.e., the attribute **Change** of the object corresponding to currency **c** does not contain an entry corresponding to currency **c'**). Moreover, we assume that if the rate exchange **r(c, c')** exists, then also **r(c', c)** exists and **r(c, c') = r(c', c) > 0**.

   We say that an *exchange tour* is a sequence of currencies that starts and ends with the same currency (i.e., a cycle) involving *all* currencies *once*. The rate of an exchange tour is given by the sum of the exchange rates of all the exchanges involved in the tour.

   E.g., if the set **C** contains currencies with code **USD**, **EUR**, **GBP, CNY, JPY** and rate exchanges
   - **r(USD, EUR)=r(EUR, USD)=0,30**
   - **r(EUR, GBP)=r(GBP, EUR)=0,31**
   - **r(USD, GBP)=r(GBP, USD)=0,09**
   - **r(GBP, CNY)=r(CNY, GBP)=0,05**
   - **r(EUR, CNY)=r(CNY, EUR)=0,87**
   - **r(USD, JPY)=r(JPY, USD)=0,43**
   - **r(JPY, CNY)=r(CNY, JPY)=0,11**

   then **(USD, EUR, GBP, CNY, JPY, USD)** is an exchange tour of rate **1,2**. Similarly, **(USD, JPY, CNY, EUR, GBP, USD)** is an exchange tour of rate **1,81**. Instead the following are not valid exchange tours:
   - **(USD, EUR, GBP, USD)** since there are currencies in **C**, namely **CNY** and **JPY**, that are not involved in this tour;

- **(USD, GBP, CNY, JPY, CNY, EUR, USD)** since there are currencies, namely **CNY**, that are involved more than once;
- **(USD, GBP, CNY, JPY, EUR, USD)** since it involves a pair of consecutive currencies, namely **(JPY, EUR)** for which the rate exchange has not been defined.

Design a local search algorithm that takes in input a set of **Currency** objects and looks for an exchange tour of minimal rate.

After the deadline, all proposed algorithms for the exchange tour computation will be tested on 5 secret inputs involving sets of 100 currencies. The 10 algorithms with the best performances, in terms of the quality of the produced solution, will be selected and ranked with respect to the time taken by the algorithm. The groups presenting the three fastest algorithms will be awarded with a bonus as follows: all the members of the first one will receive a bonus point; all the components of the second and third group will receive half a point.
Similarly, the 10 fastest algorithms (independently from the returned solution) will be selected and ranked with respect to the quality of the produced solution. The groups presenting the three best solutions will be awarded with a bonus as follows: all the members of the first one will receive a bonus point; all the components of the second and third group will receive half a point.