Stefan Kurtz **Universität Hamburg**
Abteilung für Genominformatik **Zentrum für Bioinformatik**

Grundlagen der Sequenzanalyse
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 31.01.2023

Aufgabe 13.1 (10 Punkte)

Implementieren Sie den in der Vorlesung vorgestellten Neighbor-Joining Algorithmus in Python3. Damit Sie sich auf den Kern der Aufgabenstellung konzentrieren können, finden Sie in den Materialien die folgenden Python-Module:

`dist_matrix.py` implementiert eine Klasse `DistanceMatrix`, die es erlaubt, eine Distanzmatrix in einem Standardformat einzulesen und die darin spezifizierten Taxon-Namen und paarweisen Distanzen zu verwalten. Insbesondere implementiert die Klasse eine Methode `num_of_taxa`, die die Anzahl der Taxa liefert, sowie eine Methode `taxon_name`, die für das $i$te Taxon seinen Namen als String zurückliefert. Durch die Methode `distance` erhält man für zwei Taxon-Nummern i und j die Distanz der entsprechenden Taxa.

`nejo_node.py` enthält eine Klasse `NJnode` zur Repräsentation der Knoten des durch den Neighbor-Joining Algorithmus berechneten Phylogenetischen Baumes. Jede Instanz dieser Klasse enthält die folgenden Instanz-Variablen:

- `self.taxon_name` enthält für alle Knoten, die ein Taxon repräsentieren, den entsprechenden Taxon-Namen. Diese Knoten werden Blätter des Phylogenetischen Baums. Der Taxon-Name wird als Parameter der Methode `__init__` übergeben. Für alle anderen Knoten, die im Algorithmus innere Knoten des Phylogenetischen Baums werden, hat `self.taxon_name` den Wert `None`. Auf diese Instanz-Variable darf auch außerhalb der Klasse lesend und schreibend zugegriffen werden.
- `self._idnum` enthält eine eindeutige Identifikationsnummer des Knotens, der als Index für den Zugriff auf die Distanzmatrizen verwendet wird. Diese Identifikationsnummer wird automatisch über die Klassen-Variable `NJnode._idnum` gesetzt. Zum Lesen dieses Wertes verwenden Sie die Methode `idnum()`.
- `self._rvalue` speichert für den Knoten mit der Identifikationsnummer i aus der Menge L den Wert r_i, siehe Beschreibung des Algorithmus aus der Vorlesung. Zum Schreiben und Lesen dieses Wertes gibt es die Methoden `rvalue_set` und `rvalue`.

`nejo.py` enthält das Hauptprogramm mit einem Optionsparser. Nach dessen Aufruf wird eine Instanz der Klasse `DistanceMatrix` und eine Instanz der Klasse `NeighborJoining` erzeugt. Dann werden durch Aufruf der Funktion `enum_tree_edge` die Kanten des Phylogenetischen Baumes aufgezählt und diese mit einer Funktion `format_edge(precision,parent,children,d2c)` in eine String-Repräsentation mit 5 bzw. 3-Werten konvertiert, bevor sie ausgegeben werden. In der String-Repräsentation sind die einzelnen Werte durch einen Tabulator separiert. Ein String der Form

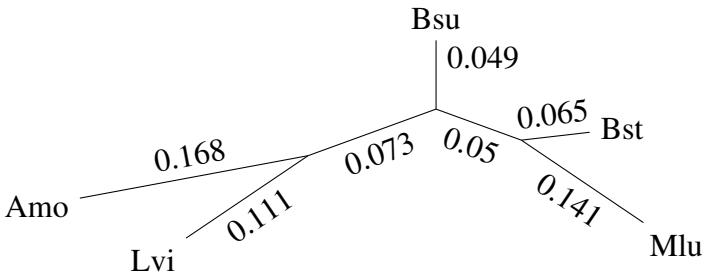
```
p c1 c2 d1 d2
```

repräsentiert einen Elternknoten `p` (spezifiziert durch den Parameter `parent`) mit den Kindknoten `c1` und `c2` (aus der Liste `children` der Länge 2), deren Distanz zu ihrem Elternknoten `d1` bzw.

d2 ist. d1 und d2 sind die beiden Elemente aus der Liste d2c. Nur in der letzten Zeile der Ausgabe steht ein String der Form

```
n1 n2 d
```

wobei n1 und n2 Knoten sind und d ihre Distanz im Phylogenetischen Baum. Blätter werden jeweils durch den Namen des Taxons dargestellt, das sie repräsentieren. Innere Knoten werden durch den entsprechenden idnum-Wert repräsentiert, also einer ganzen Zahl größer gleich der Anzahl der Taxa. Der folgende in der Vorlesung betrachtete Phylogenetische Baum



wird z.B. durch die folgenden Zeilen dargestellt:

```
5       Amo       Lvi       0.168       0.111
6       Mlu       Bst       0.141       0.065
7       5         Bsu       0.073       0.049
6       7         0.050
```

Das Hauptprogramm nejo.py erhält als einziges Argument den Namen der Datei mit der Distanzmatrix und liefert eine Ausgabe im obigen Format zurück.

Nachdem Sie sich mit den genannten Modulen vertraut gemacht haben, implementieren Sie nun in einer Datei nejo_class.py die Klasse NeighborJoining mit den folgenden Methoden bzw. Instanz-Variablen:

- Die Methode __init__(self,dm) erhält beim Aufruf genau einen Parameter dm. Das ist eine Instanz der Klasse DistanceMatrix. Diese Methode initialisiert eine eigene Distanzmatrix, die gross genug ist, um für alle Paare von Blättern und inneren Knoten, die im Algorithmus erzeugt werden, jeweils die Distanzwerte zu speichern. Überlegen Sie sich daher, wieviele innere Knoten m der Phylogenetische Baum für $n \geq 2$ Taxa enthält. Es muss dann durch den Aufruf der entsprechenden numpy-Methode (siehe dist_matrix.py) eine eigene $(n + m) \times (n + m)$-Matrix erzeugt werden. Diese wird zunächst für alle Paare von Taxa mit den Distanzwerten aus dm initialisiert. 2 Punkte
- Neben der $(n+m) \times (n+m)$-Matrix muss diese Klasse noch die Menge L speichern. Diese implementieren Sie als Liste self._Lset von Instanzen der Klasse NJnode. Am Anfang des Algorithmus (also am Ende von __init__) enthält diese Liste alle Blätter des Baumes.
- Die Methode distance_set(self,node1,node2,dist) setzt in der $(n + m) \times (n + m)$-Matrix den Distanzwert der beiden Knoten node1 und node2 auf dist. node1 und node2 sind dabei Instanzen der Klasse NJnode. 1 Punkt

- Die Methode `distance(self,node1,node2)` liefert entsprechend der eigenen Distanz-Matrix den aktuellen Distanzwert der beiden Knoten `node1` und `node2`. Diese sind jeweils Instanzen der Klasse `NJnode`. 1 Punkt
- Die Methode `update_Rtab(self)` berechnet für alle Knoten aus `self._Lset` den r-Wert und speichert ihn mit der Methode `rvalue_set` in diesem Knoten. 1 Punkt
- Die Methode `next_children_indexes(self)` liefert über eine **`return`**-Anweisung ein Paar (i, j) von Indexwerten von Knoten in `self._Lset`, so dass der aktuelle Distanzwert dieser Knoten (entsprechend der eigenen Distanz-Matrix) abzüglich der Summe ihrer r-Werte minimal ist. Sie brauchen also in ihrer Implementierung keine eigene Matrix N. Stattdessen zählen Sie alle Paare (i, j) mit $j < i$ in lexikographischer Reihenfolge auf und minimieren über die genannte Differenz des Distanz-Wertes und der r-Werte. Falls es mehrere Indexpaare mit dem gleichen Minimum gibt, soll immer das erste minimale in der Aufzählung gefundene Indexpaar zurückgeliefert werden. Es ist wichtig, hier mit den Indexwerten bzgl. der Liste `self._Lset` zu arbeiten, da man dadurch sehr einfach mit der passenden Methode für Listen die beiden entsprechenden Knoten aus `self._Lset` löschen kann. 1 Punkt
- Die Methode `enum_tree_edges(self)` implementiert nun die Iteration im Neighbor-Joining Algorithmus und liefert mit `yield`-Anweisungen die Kanten des Phylogenetischen Baums, und zwar jeweils als Tripel von drei Werten:
  - den im aktuellen Schritt neu erzeugten Elternknoten (bezeichnet durch k im Pseudocode des Algorithmus),
  - eine Liste der Länge 2 mit den beiden Kindknoten des neu erzeugten Elternknotens (bezeichnet durch i und j im Pseudocode des Algorithmus)
  - eine Liste der Länge 2 mit den Distanzen des Elternknotens zu den beiden Kindknoten.

  Im Terminationsschritt wird das Tripel `(None, self._Lset, [finaldist])` zurückgeliefert, wobei `self._Lset` aus genau zwei Elementen besteht und `finaldist` die Distanz dieser Elemente in der berechneten eigenen Distanz-Matrix ist. 2 Punkte

In den Materialien finden Sie Dateien mit Distanzmatrizen sowie mit den erwarteten Ergebnissen. Durch `make test` verifizieren Sie, dass Ihre Implementierung korrekt funktioniert. Die Distanzmatrix in `ln_example.mat` ist die aus dem Beispiel in der Vorlesung und eignet sich daher sehr gut zum Debugging Ihrer Implementierung. Die Datei `ebola10.mat` enthält eine Distanzmatrix für 10 unterschiedliche Ebola-Genome, die jeweils sehr eng miteinander verwandt sind.

Die oben angegebenen Punktezahlen summieren sich zu 8. Für die beiden erfolgreich bestandenen Tests gibt es jeweils einen weiteren Punkt.

Bitte die Lösungen zu diesen Aufgaben bis zum 05.02.2023 um 22:00 Uhr an gsa@zbh.uni-hamburg.de schicken.