# Practical session 5

Implementing NJ

# Problems in translating the problem into classes

- "*Within the zoo, various animal types such as lions, chimps, rhinos, and snakes are present, with the possibility of acquiring additional types in the future. For each type of animal in the zoo, some variation is observed in the number of eyes and color of the skin. The feeding schedule and specific dietary preferences vary for each animal. Lions and snakes are fed meat, while chimps and rhinos consume vegetables. The animal keeper ensures that each day, the appropriate type of food is provided to each animal at the designated feeding time*. **Create a program that simulates each day this process**."

# Problems in translating the problem into classes

- First, we need to identify the classes (usually names).
- Second, we need to identify the relationships between classes (redundancies indicate a common root).
- Third, we need to identify the atributes (usually adjectives).
- Fourth, we need to identify actions (usually verbs).

**Classes**

# Problems in translating the problem into classes

- “*Within the zoo, various animal types such as lions, chimps, rhinos, and snakes are present, with the possibility of acquiring additional types in the future. For each type of animal in the zoo, some variation is observed in the number of eyes and color of the skin. The feeding schedule and specific dietary preferences vary for each animal. Lions and snakes are fed meat, while chimps and rhinos consume vegetables. The animal keeper ensures that each day, the appropriate type of food is provided to each animal at the designated feeding time*. **Create a program that simulates each day this process**.”

**Attributes'**

# Problems in translating the problem into classes

- “*Within the zoo, various animal types such as lions, chimps, rhinos, and snakes are present, with the possibility of acquiring additional types in the future. For each type of animal in the zoo, some variation is observed in the number of eyes and color of the skin. The feeding schedule and specific dietary preferences vary for each animal. Lions and snakes are fed meat, while chimps and rhinos consume vegetables. The animal keeper ensures that each day, the appropriate type of food is provided to each animal at the designated feeding time*. **Create a program that simulates each day this process**.”

**Actions?**

# Problems in translating the problem into classes

- “*Within the zoo, various animal types such as lions, chimps, rhinos, and snakes are present, with the possibility of acquiring additional types in the future. For each type of animal in the zoo, some variation is observed in the number of eyes and color of the skin. The feeding schedule and specific dietary preferences vary for each animal. Lions and snakes are fed meat, while chimps and rhinos consume vegetables. The animal keeper ensures that each day, the appropriate type of food is provided to each animal at the designated feeding time*. **Create a program that simulates each day this process**.”

# pseudocode

# Problems in translating the problem into classes

- Prior conditions: we have created for each type of animal as many animals as the present in the zoo. For example:
- pumba = Lion(number_of_eyes, color)
- nala = Lion(number_of_eyes, color)
- …
- Put each type of animal in its own list
- Put each list into a list_of_animals
- arnau = ZooKeeper()
- for hour in 1:24 DO:
  - for list_of_animals DO:
    - for animals in list DO:
      - if(animal.get_time_feeding()==hour)
      - arnau.feed(animal)

# Exam

```
def estimate_emission_probabilities(PI, x):
    # Get the different states of the sequence of hidden states PI
    states_in_PI = []
    # iterate over the sequence of PI. For each state, check if it is in the list of states_in_PI
    # This loop is looking for the unique states in the sequence
    for i in self.PI:
        # add to the list if it is not already in the list of states
        if i not in states_in_PI:
            # add the state in the states_in_PI list
            states_in_PI.append(i)

    # Get the different categories that can be found in the vector x
    categories_in_x = {}
    # Iterate over each position of x
    for i in x:
        # if the category is not in the list of categories, add it
        if i not in categories_in_x:
            # add the category to the list of categories
            categories_in_x .append(i)

    # a dictionary of dictionaries to store, for each state, the emission probabilities
    e = []

    # iterate over all the states of PI
    for key in states_in_PI:
        # create a new dictionary to store the categories of that state
        t = []
        # for each unique category
        for cat in categories_in_x:
            # initialize the counter at 0
            t[cat] = 0
        # add to the dictionary e the dictionary with the frequencies of the different categories
        e[key] = t

    # iterate over each position
    for i in range(len(x)):
        # current state at position i
        state = PI[i]
        # current category at position i
        category = x[i]
        # update the dictionary of dictionaries at state and category
        e[state][category] = e[state][category] + 1

    #
    for state in states_in_PI:
        #
        T = 0
        #
        for cat in categories_in_x:
            #
            T = T + e[state][cat]
        #
        for cat in categories_in_x:
            #
            e[state][cat] = int(e[state][cat]/float(T))
```

# Exercise 2

```
'''
Exercise 2. (2 Pnt) Assume an exponential probability distribution with cdf = 1 - exp(-l*x).
Use the inverse transform sampling technique to create a function called "sample"
that takes l as parameter to generate numbers x randomly sampled
from the probability distribution.

Tip: Use math.log and random.random(). Comment each line of code
'''
def sample(l):
    u = random.random()
    # log(1-cdf) = -l*x; x = -log(1-cdf)/l
    return -math.log(1-u)/l
```

# Exercise 2

```
'''
Exercise 3. (2 Pnt) Fix (five errors) and comment the code. WHAT IS THE PURPOSE OF THIS CODE? '''

def prob_x_and_pi(prior_prob, transition_prob, emission_prob, x, pi):
    # Initialize the probability with the prior at state in position i-1
    p = prior_prob[pi[0]] # prior of the hidden state
    # counter to store the state of previous position.
    si = pi[0]
    #
    for ii in range(len(pi)):
        #
        p = p*emission_prob[pi[ii]][x[ii]]*transition_prob[si][pi[ii]] # move from state si to pi and emission from state pii and x
        #
        si = pi[ii]
    #
    return p # return p
```

*Exercise 4 (2 Pnt)*

*Reconstruct phylogenetic tree from the following distance matrix using UPGMA approach:*

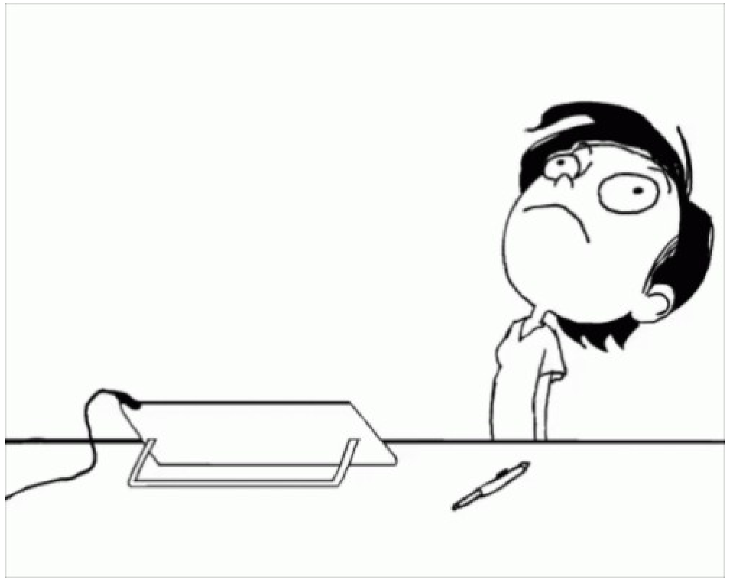
| OTUS | A | B | (CD) | E | F |
|---|---|---|---|---|---|
| A | 0 | 6 | 29 | 24 | 30 |
| B | 6 | 0 | 31 | 26 | 28 |
| (CD) | 29 | 31 | 0 | 32 | 15 |
| E | 24 | 26 | 32 | 0 | 30 |
| F | 30 | 28 | 15 | 30 | 0 |

*What would be the topology of this tree? (Parentheses indicate the order of grouping):*
*a) ((EAB(CDF)); b) ((EA(B))(CD)F); c) (((AB)(CD)F)E); d) (E(AB))((CD)F)*
*If in the previous exercise dF(CD) – d(CD)=9, what is the distance of the both taxons C and D to their most recent common ancestor?*
*a) 3,0 ; b) 1,5 ; c) 1,0 ; d) 2,0*

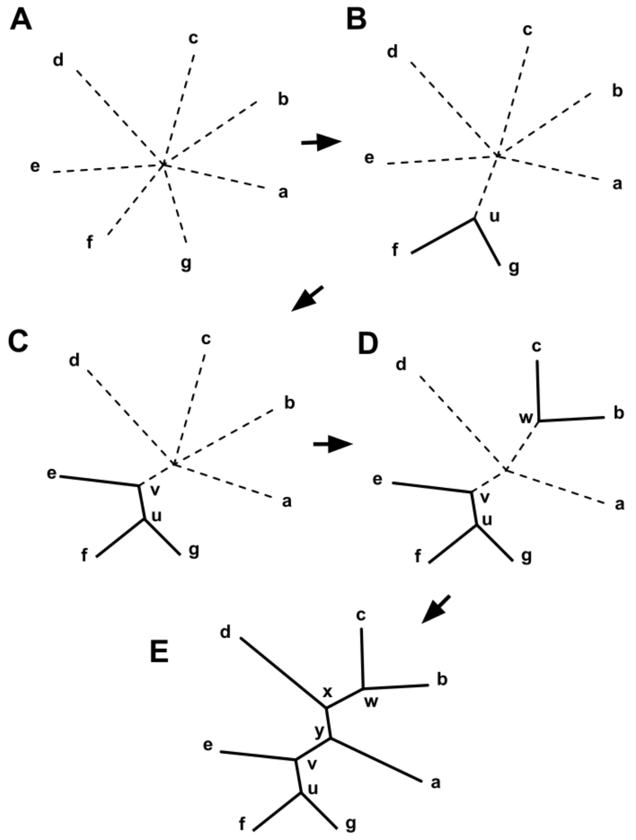
```
'''        (((A:3.0,B:3.0):1.5,E:1.5):14.75,
           ((CD):7.5,F:7.5):14.75)
```

```
'''
Exercise 5 (2 Pnt).

Write in pseudocode (NO NEED TO BE PYTHON FRIENDLY) an algorithm for
evolving in forward a sequence
 of n nucleotides for x generations given a transition matrix M between
nucleotides.

'''
```

# Stuff going on during the exam…



# Implement the NJ algorithm




From Wikipedia

# Implement the NJ algorithm

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$$

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 5 | 9 | 9 | 8 |
| b | 5 | 0 | 10 | 10 | 9 |
| c | 9 | 10 | 0 | 8 | 7 |
| d | 9 | 10 | 8 | 0 | 3 |
| e | 8 | 9 | 7 | 3 | 0 |

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | | −50 | −38 | −34 | −34 |
| b | −50 | | −38 | −34 | −34 |
| c | −38 | −38 | | −40 | −40 |
| d | −34 | −34 | −40 | | −48 |
| e | −34 | −34 | −40 | −48 | |

From Wikipedia

# Implement the NJ algorithm

$$\delta(a,u) = \frac{1}{2}d(a,b) + \frac{1}{2(5-2)}\left[\sum_{k=1}^{5} d(a,k) - \sum_{k=1}^{5} d(b,k)\right] \quad = \frac{5}{2} + \frac{31-34}{6} = 2$$

$$\delta(b,u) = d(a,b) - \delta(a,u) \quad = 5 - 2 = 3$$

| | a | b | c | d | e |
|---|---|---|---|---|---|
| **a** | 0 | 5 | 9 | 9 | 8 |
| **b** | 5 | 0 | 10 | 10 | 9 |
| **c** | 9 | 10 | 0 | 8 | 7 |
| **d** | 9 | 10 | 8 | 0 | 3 |
| **e** | 8 | 9 | 7 | 3 | 0 |

$$\delta(f,u) = \frac{1}{2}d(f,g) + \frac{1}{2(n-2)}\left[\sum_{k=1}^{n} d(f,k) - \sum_{k=1}^{n} d(g,k)\right]$$

$$\delta(g,u) = d(f,g) - \delta(f,u)$$

$$d(u,k) = \frac{1}{2}[d(f,k) + d(g,k) - d(f,g)]$$

$$d(u,c) = \frac{1}{2}[d(a,c) + d(b,c) - d(a,b)] = \frac{9+10-5}{2} = 7$$

$$d(u,d) = \frac{1}{2}[d(a,d) + d(b,d) - d(a,b)] = \frac{9+10-5}{2} = 7$$

$$d(u,e) = \frac{1}{2}[d(a,e) + d(b,e) - d(a,b)] = \frac{8+9-5}{2} = 6$$

| | u | c | d | e |
|---|---|---|---|---|
| **u** | 0 | **7** | **7** | **6** |
| **c** | **7** | 0 | *8* | *7* |
| **d** | **7** | *8* | 0 | *3* |
| **e** | **6** | *7* | *3* | 0 |

From Wikipedia