

# **Code documentation - Sphinx**

*Release 31/05/2023*

## **Algoritmos Avançados para a Bioinformática**

May 30, 2023



|          |                                   |           |
|----------|-----------------------------------|-----------|
| <b>1</b> | <b>Code Documentation</b>         | <b>1</b>  |
| 1.1      | package_aab.src package . . . . . | 1         |
|          | <b>Python Module Index</b>        | <b>15</b> |
|          | <b>Index</b>                      | <b>17</b> |



---

## Code Documentation

---

### 1.1 package\_aab.src package

#### 1.1.1 Submodules

#### 1.1.2 package\_aab.src.Automata module

**class** package\_aab.src.Automata.**Automata** ( *alphabet: str, pattern: str* )

Bases: object

Class implemented for the search of patterns in nucleotide sequences. This pattern search method processes a sequence in a linear way, reading from left to right. The search started at the first state of the “machine” and at the first character of the text, at each step of the search the next character of the sequence is considered, the next state is searched state and moves to a new state. The number of states is equal to the length of the pattern plus one. The pattern is found when the pattern being searched for is equal to the length of the pattern entered.

**applyNextState** ( *seq: str* ) → list

Method that returns a list of all next states. :param seq: sequence entered :return: list of next states

**buildTransitionTable** ( *pattern: str* )

Method that constructs the transition table. The transition table returns the next state of the automata machine from the current state and previous states. :param pattern: pattern to search in the sequence

**nextState** ( *current: int, char: str* ) → int

Method that returns the next state. :param current: current state :param char: character of the pattern to search for :return: the next state

**patternSeqPosition** ( *seq: str* ) → list

Method that returns the list of positions where an occurrence of the pattern in the sequence starts. :param: sequence entered :return: list of the positions where an occurrence of the pattern in the sequence starts.

**printAutomata** ( )

Method that prints the results of the pattern search using the automata. :return : automata printed

package\_aab.src.Automata.**overlap** ( *seq1: str, seq2: str* ) → int

Method that overlaps two sequences and checks for matching :param seq1: first string :param seq2: second string :return: last position of the smallest string that matches

### 1.1.3 package\_aab.src.BWT module

**class** package\_aab.src.BWT.**BWT** ( seq='', buildsufarray=False, sa=None )

Bases: object

Class for implementing the Burrows-Hweeler algorithm. The algorithm is useful for compressing large sequences, thus reducing their space. Moreover, it is possible to find patterns in the compression format efficiently.

**build\_bwt** ( text, buildsufarray=False ) → str

Method that generates rotations of the text and sorts them to construct the BWT string.  
:param text: :param buildsufarray: :return:

**bw\_matching** ( pattern=<class 'str'> ) → List[int]

Method for searching for patterns from BWT sequence :param pattern: pattern to search  
:return: list of matches found

**bw\_matching\_pos** ( patt ) → List

Method that searches for matching patterns :param patt: pattern to search :return: list of matches found

**check\_seqs** ( )

Method that checks if the sequence introduced is a string :return:

**static findithocc** ( le, elem, index )

Method that allows the discovered of the i-ésima occurrence of an symbol in the list :param le:  
:param elem: :param index: :return: -1 in the cases where doesn't occur

**get\_first\_col** ( ) → List[str]

Method that recovers the first column of BWT sequence. The first column correspond the BWT sequence sorted. :return: first column of Matrix M

**inverse\_bwt** ( ) → str

Method recovers the original sequence from its BWT. :return: string of the original sequence

**last\_to\_first** ( ) → List

Creates a list mapping each character in the BWT sequence to its corresponding index in the first column. :return: list of indices

**set\_bwt** ( bw )

Method that sets the BWT string directly for a BTW instance. It is useful in scenarios where we have a precomputed BWT string and want to assign it to the bwt attribute of the BWT instance without recomputing it using the build\_bwt method. :param bw: :return:

### 1.1.4 package\_aab.src.BoyerMoore module

**class** package\_aab.src.BoyerMoore.**BoyerMoore** ( alphabet=<class 'str'>, pattern=<class 'str'> )

Bases: object

**process\_bcr** ( )

Bad Character rule implementation. This rule is used to skip over characters in the text that do not match the corresponding character in the pattern. It precomputes the last occurrence of each character in the pattern and uses this information to determine the number of positions to shift the pattern relative to the text when there is a mismatch. Method where a dictionary is created with all possible symbols (occ) as keys, and the values define the rightmost position at which the symbol appears in the pattern (-1 means it does not occur). This allows you to quickly calculate the number of positions to follow to search according to the mismatch in the pattern (value for the symbol in the dictionary). Note that

this value can be negative, meaning that the rule in this case is not useful and is ignored in the next iteration.

**process\_gsr** ( )

Good Suffix rule implementation. Calculates the value of the f list (the length of the longest proper suffix that matches the suffix of the pattern), and s list (the length of the longest proper suffix that matches the prefix of the pattern).

**search\_pattern** ( *text=<class 'str'>* ) → List[int]

This method allows to find a pattern in a given text, based on the object of the class that contains the pattern and its alphabet. :param text: string of the text where we want to look for our pattern :return: list with the indexes where the pattern starts

## 1.1.5 package\_aab.src.EAMotifs module

## 1.1.6 package\_aab.src.EvolAlgorithm module

## 1.1.7 package\_aab.src.Indiv module

**class** package\_aab.src.Indiv.**Indiv** ( *size, genes=[], lb=0, ub=1* )

Bases: object

# The Indiv class represents an individual in a genetic algorithm. It contains methods for initialization, mutation, crossover, and fitness evaluation.

**crossover** ( *indiv2* ) → tuple

**getFitness** ( ) → Union[int, float]

**getGenes** ( ) → list

**initRandom** ( *size: int* ) → None

**mutation** ( ) → None

**one\_pt\_crossover** ( *indiv2* ) → tuple

**setFitness** ( *fit* ) → None

**class** package\_aab.src.Indiv.**IndivInt** ( *size: int, genes: list = [], lb: int = 0, ub: int = 1* )

Bases: [Indiv](#)

**initRandom** ( *size: int* ) → None

Method that generates a list of genes of the individual (random int numbers between upper and lower bounds) :param size: number of genes to generate :return:

**mutation** ( )

**class** package\_aab.src.Indiv.**IndivReal** ( *size: int, genes: list = [], lb: float = 0.0, ub: float = 1.0* )

Bases: [Indiv](#)

**initRandom** ( *size: int* ) → None

Method that generates a list of genes of the individual with random values in the interval [lb, ub]. :param size: :return:

**mutation** ( ) → None

Method for real representations that alters a single gene (mutation) The mutation method performs a mutation operation on one randomly selected gene in the individual. :return:

## 1.1.8 package\_aab.src.MetabolicNetwork module

## 1.1.9 package\_aab.src.MotifFinding module

**class** package\_aab.src.MotifFinding.**MotifFinding** ( *size: int = 8, seqs=None* )

Bases: object

Class implemented for searching for recurrent patterns in a biological sequence, which can be DNA or proteins. The pattern to look for can be an exact sequence or a degenerate consensus in which there are ambiguous characters.

**branchAndBound** ( ) → Optional[Union[list, List[int]]]

Method that implements to iterate over the initial position of the list of solutions to find the best solution motif. :return: the vector of starting positions s (representative of the best motif found)

**bypass** ( *s: list* ) → list

Method implemented to bypass the condition explained above. :param s: index list of the motifs in the entered sequences. :return: list of indexes with zeros in the non-corresponding positions.

**createMotifFromIndexes** ( *indexes: list* )

Method that implements probabilistic motifs of type MyMotif. :param indexes: list of the indexes of the initial positions of the motif of the sub-sequences used to create the motif. :return:

**exhaustiveSearch** ( ) → list

Method implemented to find the motif with the vector of best scores. This method allows to derive the profile and the consensus sequence. :return: vector of best scores

**gibbs** ( *num\_iterations: int* ) → list

Method that implements the Gibbs Sampling algorithm, by choosing new segments at random (increasing the possibilities of converging to a correct solution). The algorithm starts with a set of randomly selected motif positions, and then in each iteration, it chooses one sequence at random, removes it from consideration, and recomputes the motif position based on the remaining sequences. This process continues for a fixed number of iterations, after which the algorithm returns the best set of motif positions found so far. :param num\_iterations: number of iterations :return: list of best scores

**heuristicConsensus** ( ) → list

'''Method that computes the heuristic consensus algorithm: - Considering only the first two sequences, choose the initial positions s1 and s2 that give a better score. - For each of the following sequences, iteratively, choose the best starting position in the sequence, in order

to maximize the score.

**Returns** list of starting positions s (representative of the best motif found)

**heuristicStochastic** ( ) → list

Method that computes the heuristic stochastic consensus algorithm, using the most likely segments to adjust starting positions to achieve the best profile (motif) :return: the vector of starting positions s (representative of the best motif found)

**nextSol** ( *s: list* ) → list

Auxiliary method (to exhaustiveSearch) that gives the next vector of starting positions. Method implemented to iterate over all the possible values of the motif position in the entered sequence. :param s: list of the motif indexes in the entered sequence. :return: list of all the possible values of the position of the motif in the entered sequence.



**nextVertex** (*s: list*) → list

Method implemented to find the next vertex. :param s: list of indices of the motifs in the entered sequences. :return: list of next vertices

**readFile** (*file: str, type: str*) → None

Method that reads the sequence file. param file: sequence file. param type: type of the sequences.

**roulette** (*f: list*) → int

Method implemented to simulate examples of a roulette wheel. The probability of choosing a certain position is proportional to your score. :param f: list of positions :return: chosen value

**score** (*s: list*) → int

Method implemented for the scoring function. The scoring function iterates over all motif positions and determines the maximum value of the motif score. :param s: list of motif indices in the entered sequences. :return: maximum score of the motif score.

**scoreMult** (*s*)

Method implemented for the multiplication of the maximum motif scores. The scoreMult() method provides a way to quantify the importance of each occurrence of a motif by calculating the maximum score for each position in the motif, and then multiplying these scores together to get an overall score. :param s: list of the motif indexes in the introduced sequences. :return: multiplication of maximum scores.

**seqSize** (*i: int*) → int

Method that returns the length of the sequence. :param i: index of the sequence in the sequence list. :return: length of the sequence with index i from the sequence list.

package\_aab.src.MotifFinding.**random** ( ) → x in the interval [0, 1).

package\_aab.src.MotifFinding.**test1\_1** ( )

package\_aab.src.MotifFinding.**test1\_2** ( )

package\_aab.src.MotifFinding.**test1\_3** ( )

package\_aab.src.MotifFinding.**test1\_4** ( )

package\_aab.src.MotifFinding.**test1\_5** ( )

package\_aab.src.MotifFinding.**test2** ( )

### 1.1.10 package\_aab.src.MyGraph module

**class** package\_aab.src.MyGraph.**MyGraph** (*g: dict = {}*)

Bases: object

Class for implementation of graphs. Class MyGraph is a parent class of different ones such as: MetabolicNetwork, OverlapGraph and DeBruijnGraph. This class essentially builds graphs with a given dictionary of values or strings.

**add\_edge** (*o: str, d: str*)

Method that add edge to the graph; if vertices do not exist, they are added to the graph :param o: vertice do arco :param d: vertice do arco :return:

**add\_vertex** (*v: str*)

Method that adds a vertex to the graph and tests if vertex exists or not, adding if True :param v: element to add in the graph

**all\_clustering\_coefs** ( ) → dict

Method that calculates all coefficients :return: dictionary of coefficients of each node where the key corresponds to each node and each coefficient

**all\_degrees** ( *deg\_type*='inout' ) → dict

Method that computes the degree (of a given type) for all nodes.

"Deg\_type" can be "in", "out", or "inout"

Parameters **deg\_type** – tipo de grau (entrada, saída ou ambos)

Returns returns the degrees

**betweenness centrality** ( *node* ) → float

**check\_balanced\_graph** ( ) → bool

Checking method if the graph is balanced return:value of "False" or "True", if the graph is unbalanced or balanced, respectively

**check\_balanced\_node** ( *node* ) → bool

Method to check if a node is balanced :param node: node :return: value of "False" or "True", if the node's input degree is not equal/or is equal to the node's output degree, respectively

**check\_if\_hamiltonian\_path** ( *p: list* ) → bool

Method checking if path is Hamiltonian :param p: path :return: value of "False" or "True" if the Hamiltonian path is invalid or valid, respectively.

**check\_if\_valid\_path** ( *p: list* ) → bool

Method of checking if path is correct :param p:path :return: value of "False" or "True" if path is invalid or valid, respectively.

**check\_nearly\_balanced\_graph** ( )

Method to check if the graph is semi-balanced :return: returns none (if the graph is not semi-balanced) or the tuples

**closeness centrality** ( *node: str* )

Method of averaging approach to distances travelled between affected nodes :param node: node :return: returns the average distance between affected nodes

**clustering\_coef** ( *v* ) → float

Clustering coefficient calculation method, to measure the extent to which each node is embedded in a cohesive group :param v: node :return: clustering coefficient

**degree** ( *v: str* ) → int

Method that obtains the number of degree. Represents the number adjacentes nodes of the given one :param v: node :return: returns degree of node v

**distance** ( *s, d* )

Method that calculates the distance between two nodes, s and d, in a graph represented as an adjacency list :param s: node s where the path begins :param d: node d where the path ends :return: returns the distance between the nodes s and d

**eulerian\_cycle** ( )

Eulerian cycle passes through all arcs of the examining graph once, returning to the starting node :return: value of "None" (if not an Eulerian cycle) or the list res (cycle list)

**eulerian\_path** ( )

Eulerian path method, if existent :return: returns the path

**get\_adjacents** ( *v: str* ) → list

Method that obtains adjacents of the given element :param v: node :return: returns the list of nodes adjacent to node v

**get\_edges** ( ) → list

Method that gets the edges in the graph as a list of tuples (origin, destination) v represent the origin d represent the destination

**get\_nodes** ( ) → list

Method that obtain list of nodes in the graph

**get\_predecessors** ( *v: str* ) → list

Method that obtains predecessors of the given element :param v: node :return: returns a list of predecessor nodes of node v

**get\_successors** ( *v: str* )

Method that obtains the successors of the given element :param v: node :return: returns a list of successor nodes of node v

**has\_cycle** ( ) → bool

Method that checks if the graph has a loop, that is if the path is closed :return: value of "False" or "True" if the path is not closed or closed, respectively.

**highest\_closeness** ( *top=10* ) → list

**highest\_degrees** ( *all\_deg=None, deg\_type='inout', top=10* ) → list

Method that calculates higher degrees param all\_deg: input and output grades, or both param deg\_type: degree type (input, output or both) :return: list of the highest degrees

**in\_degree** ( *v: str* ) → int

Method that obtains number of in degree. Represents the number of predecessors this node of the graph possesses :param v: nodo :return: returns degree of entry from node

**is\_connected** ( ) → bool

Method for checking whether it is connected or not :return: value of "False" or "True", if not connected or connected, respectively

**mean\_clustering\_coef** ( ) → float

Method of the global average of coefficients :return: the value of the average of all coefficients

**mean\_clustering\_perdegree** ( *deg\_type='inout'* ) → dict

Method that calculates values for the average of coefficients for all nodes :param deg\_type: type of degree (input, output or both) :return: returns the dictionary of nodes and their coefficient

**mean\_degree** ( *deg\_type='inout'* ) → dict

Method to calculate the average degrees :param deg\_type: type of degree (input, output or both) :return: returns the average degrees

**mean\_distances** ( )

Method that calculates the cverage distance dethod for Each Node :return: returns the average distance and the proportion of nodes reachable

**node\_has\_cycle** ( *v* )

Method to check if the node has a cycle, that is if it begins and ends in the same node :param v: node :return: value of "False" or "True" if the node has no cycle or has a cycle, respectively

**out\_degree** ( *v*: *str* ) → int

Method that obtains the number of out degree. Represents the number of successors/ramifications this node of the graph possesses :param v: node :return: returns degree of exit from node

**print\_graph** ( )

Method that prints the content of the graph as adjacency list

**prob\_degree** ( *deg\_type*='inout' ) → dict

Method for calculating degree probability :param deg\_type: type of degree (input, output or both) :return: returns the probability of degrees

**reachable\_bfs** ( *v* )

Method of nodes reachable from v starts with the source node, then visits all its successors, followed by its successors until all possible nodes are visited :param v: starting node :return: returns list of nodes reachable from v

**reachable\_dfs** ( *v* )

Method of nodes reachable from v, from left to right (in depth) starts with the source node, explores its first successor, then its first successor until no further exploration is possible, then returns to explore more alternatives. :param v: node :return: returns list of nodes reachable from v

**reachable\_with\_dist** ( *s*: *str* )

Method that returns a list of the reachable nodes from the given "s" and respective distance needed :param s: node s :return: list of tuples of the reachable nodes and distance: (Node, Distance)

**search\_hamiltonian\_path** ( )

" Hamiltonian pathfinding method :return: if p is different from None returns p otherwise returns None

**search\_hamiltonian\_path\_from\_node** ( *start*: *int* )

Hamiltonian pathfinding method on the graph :param start: initial node :return: Hamiltonian path in list

**shortest\_path** ( *s*, *d* )

Method that returns the shortest path between s and d (list of nodes it passes through) :param s: node s, node where the path begins :param d: node d, node where the path ends :return: return the shortest path, list of nodes it passes through

**size** ( )

Method that obtains the size of the graph: number of nodes and number of edges

`package_aab.src.MyGraph.is_in_tuple_list ( tl, val ) → bool`

### 1.1.11 package\_aab.src.MyGraph1 module

**class** `package_aab.src.MyGraph1.MyGraph` ( *g*: *dict* = {} )

Bases: object

**add\_edge** ( *o*: *Union*[*str*, *int*, *float*], *d*: *Union*[*str*, *int*, *float*] )

Add edge to the graph; if vertices do not exist, they are added to the graph :param o: represents the origin (starting) vertex :param d: represents the destination (ending) vertex of the edge

**add\_vertex** ( *v*: *Union*[*str*, *int*, *float*] )

Add a vertex to the graph; tests if vertex exists not adding if it does

**degree** ( *v: Union[str, int, float]* ) → int

Method that calculates degree of node v (all adjacent nodes either forerunners or successors)

**distance** ( *s: Union[str, int, float], d: Union[str, int, float]* )

Method that calculates the distance between two nodes, s and d, in a graph represented as an adjacency list :param s: node s where the path begins :param d: node d where the path ends :return: returns the distance between the nodes s and d

**get\_adjacents** ( *v: Union[str, int, float]* ) → list

Method to obtain adjacents of the given element

**get\_edges** ( ) → list

Method returning the list of arcs v represent the origin node d represent the destination node

**get\_nodes** ( ) → list

Method that returns the list of nodes in the graph

**get\_predecessors** ( *v: Union[str, int, float]* ) → list

Method to obtain predecessors of the given element

**get\_successors** ( *v: Union[str, int, float]* ) → list

Method for obtain successors of the given element

**has\_cycle** ( )

**in\_degree** ( *v: Union[str, int, float]* ) → int

Method to obtain the number of in degree. Represents the number of predecessors this node of the graph possesses

**node\_has\_cycle** ( *v: Union[str, int, float]* ) → bool

Method that verifies if a node has a cycle, i.e. if it starts and ends at the same node

**out\_degree** ( *v: Union[str, int, float]* ) → int

Method to obtain the number of out degree. Represents the number of successors/ramifications this node of the graph possesses

**print\_graph** ( )

Prints the content of the graph as adjacency list

**reachable\_bfs** ( *v: Union[str, int, float]* ) → list

Method of nodes reachable from v starts with the source node, then visits all its successors, followed by its successors until all possible nodes are visited :param v: starting node :return: returns list of nodes reachable from v

**reachable\_dfs** ( *v: Union[str, int, float]* ) → list

Method of nodes reachable from v, from left to right (in depth) starts with the source node, explores its first successor, then its first successor until no further exploration is possible, then returns to explore more alternatives. :param v: node :return: returns list of nodes reachable from v

**reachable\_with\_dist** ( *s: Union[str, int, float]* ) → list

Method that returns a list of the reachable nodes from the given "s" and respective distance needed :param s: node s :return: list of tuples of the reachable nodes and distance: (Node, Distance)

**shortest\_path** ( *s: Union[str, int, float], d: Union[str, int, float]* )

Method that returns the shortest path between s and d (list of nodes it passes through)

:param s: node s, node where the path begins :param d: node d, node where the path ends  
:return: return the shortest path, list of nodes it passes through

**size** ( ) → tuple

Method that returns the size of the graph : number of nodes, number of edges

package\_aab.src.MyGraph1.**is\_in\_tuple\_list** ( *tl, val* )

package\_aab.src.MyGraph1.**test1** ( )

package\_aab.src.MyGraph1.**test2** ( )

package\_aab.src.MyGraph1.**test3** ( )

package\_aab.src.MyGraph1.**test4** ( )

package\_aab.src.MyGraph1.**test5** ( )

### 1.1.12 package\_aab.src.MyMotifs module

**class** package\_aab.src.MyMotifs.**MyMotifs** ( *lseqs: list = [], pwm: list = [], alphabet: Optional[str] = None* )

Bases: object

Class that presents the methods that allow the manipulation and search of recurring patterns (motifs) in biology sequences and also do PWM.

**consensus** ( ) → str

Method that generates a consensus sequence. Consensus sequences store the most conserved characters in each position of the pattern, that is, the highest value of each column of the count matrix. :return: string of the consensus sequence.

**createPWM** ( ) → None

Method that creates the probabilistic matrix. PWMs are probabilistic representations of the characters in biological sequences. It calculates the probability of nucleotide i being found at position j.

**doCounts** ( )

Method that implements the counting matrices. :return: counts matrix

**maskedConsensus** ( ) → str

Method that generates the consensus sequence that is obtained with the characters that have an incidence higher than 50%. :return: string of the consensus sequence with incidence higher than 50%.

**mostProbableSeq** ( *seq: str* ) → int

Method implemented to determine the sub-sequence with the highest probability of matching the pattern being searched. :param seq: sequence introduced :return: the index of the sub-sequence with higher probability of matching the pattern in search.

**probAllPositions** ( *seq: str* ) → list

Method implemented to calculate the probability of finding patterns in longer sequences and calculate the probability of the pattern occurring at each character in the sequence, ie, of occurring at each sub-sequence of the pattern size. For example, if the pattern size is 3 and the sequence is "ATCGTACG", the method would calculate the probability of the pattern occurring at each of the sub-sequences of size 3, which are "ATC", "TCG", "CGT", "GTA", "TAC", "ACG". :param seq: sequence introduced :return: list of probabilities of the pattern occurring at each subsequence

**probabSeq** (*seq: str*) → float

Method that calculates the probability of a pattern being found in a sequence. :param seq: sequence introduced :return: the probability of a pattern being found in the sequence

`package_aab.src.MyMotifs.createMatZeros` (*line\_num: int, col\_num: int*) → list

Method that creates the matrix of zeros. The matrix of zeros is used as a base to store the counts or probabilities of each character in each position of the biological sequences that are being analyzed. :param line\_num: number of lines in the matrix :param col\_num: number of columns in the matrix :return: matrix of zeros

`package_aab.src.MyMotifs.printMat` (*mat: list*)

Method to prints the matrix :param mat: matrix

### 1.1.13 package\_aab.src.MySeq module

**class** `package_aab.src.MySeq.MySeq` (*seq: str, seq\_type: str*)

Bases: object

Class presenting the methods that allow the manipulation of DNA, RNA and protein sequences.

**allProtein** ( )

Method that constructs of a list with all the existing and possible proteins in the sequence. The sequence can be proteic or DNA Only executes in case the object is a DNA or AMINO sequence :return: DNA or amino sequence

**alphabet** ( )

Method that verifies the type of sequence :return: returns the possible characters of the sequence.

**codonTranslate** (*cod: str*) → str

Method that translates codons into their respective amino acids. :param cod: codon to look for in the codon table :return: amino acid sequence

**largestOrfProtein** ( )

Method to get the bigger protein in the sequence :return: string with the bigger protein

**longestProteinSeq** ( )

Method searching for the longest protein sequence :return: longest proteic sequence

**orfs** ( )

Method that determines the open reading frames (ORF), i.e. the sequences between the initiation codon and the STOP codon. It generates six reading frames of the DNA sequence and the reverse complement. :return: returns ORF's

**printSeq** ( )

Method that prints the complete sequence :return: string of the complete sequence

**reverseComplement** ( )

Method that inverts and complements the sequence Only executes in case the object is a DNA sequence :return: DNA sequence

**rnaCodon** ( ) → list

Method that searches for the codons in the sequence. :returns: list of codons

**seqTranslation** (*initial\_pos: int = 0*)

Method that performs the translation of the sequence :param initial\_pos: determines the initial position of the reading sequence :return:

**transcription ( )**

Method that returns the transcript sequence. Replacing of the thymine base 'T' for the uracil base 'U' Only executes in case the object is a DNA sequence :return: RNA sequence

**valida ( )**

Method that checks if all the characters in sequence are present in the alphabet defined :return:

**validateSeqRE ( )** → bool

Method that validates the sequence according to the type of characters present through regular expressions. :return: value True or False if the sequences are valid or invalid, respectively

### 1.1.14 package\_aab.src.Popul module

### 1.1.15 package\_aab.src.SuffixTree module

**class** package\_aab.src.SuffixTree.**SuffixTree** ( seqs: str )

Bases: object

Class that creates a suffix tree of a pattern that will be searched in a sequence. It allows pre-processing a target sequence, making its search more efficient. It is the solution to pre-process very large sequences, discovering which trees contain a given pattern, finding out the longest common substring in a set of sequences and calculating the maximum overlap of a set of sequences.

**add\_node** ( origin, symbol: str, leaf\_num=- 1 ) → None

Method that adds nodes to the tree. :param origin: current node :param symbol: character referring to the node to be added :param leaf\_num: leaf number (-1 is default, correspond to an internal node)

**add\_suffix** ( p, suf\_num )

Method that adds suffix to the tree. :param p: pattern :param suf\_num: suffix number for leaves or -1 for non-leaves

**check\_seqs** ( )

Method that verifies if the sequence introduced is a string

**find\_pattern** ( pattern )

Method that looks for patterns (trie) starting from the root until it reaches the end node or fails the search. :param pattern: pattern to search for :return: list of sheets below a given node or none (if search fails)

**get\_leafes\_below** ( node )

Auxiliary method to collect all the leafes below a given node. param node: node from which the information of the sheets below this one are searched. :return: list of sheets below a given node.

**print\_tree** ( )

Method that prints the tree structure (dictionary)

**suffix\_tree\_from\_seq** ( text )

Method that creates the suffix tree by adding a symbol ("\$\$") at the end of the sequence and calls the previous method for each suffix in the sequence -> no suffix will be prefixed with another suffix. :param text: sequence that will be added to the tree

package\_aab.src.SuffixTree.**test** ( )



### 1.1.16 package\_aab.src.Trie module

**class** package\_aab.src.Trie.**Trie** ( seqs: str )

Bases: object

Class responsible for implementing the prefix tree that allows pre-processing of a set of patterns. The symbols of a given alphabet are associated with the arcs of a tree. The trie is built from a set of patterns, starting from the root node and iterating each pattern, adding the necessary nodes so that the tree contains the path from the root to the leaf, representing the pattern.

**check\_seqs** ( )

Method that verifies if the sequence is a string

**insert** ( seq: str ) → None

Method that inserts the sequences into the trie :param seq: sequences introduced :return:

**insert\_sequence** ( seq ) → None

Method that inserts a single sequence into the trie. :param seq: sequence introduced :return:

**matches** ( seq ) → bool

Method that verifies if a given sequence matches any sequence present in the trie :param seq: sequence introduced :return: booleano

**print\_trie** ( )

Method that prints the tree structure (dictionary)

### 1.1.17 package\_aab.src.debruijn module

**class** package\_aab.src.debruijn.**DeBruijnGraph** ( frags: list )

Bases: MyGraph

Class implemented to represent Bruijn graphs. These represent the fragments (k-mers) as arcs of the graph and the nodes as sequences of size k-1, corresponding to prefixes or suffixes of the fragments. This class is a subclass of MyGraph and thus inherits all the methods defined in it.

**add\_edge** ( o: str, d: str )

Method that adds the arc (o,d) to the graph, checking that it doesn't already exist there  
param o: arc vertex param d: vertex of arc

**create\_deBruijn\_graph** ( frags: list )

Method implementing the creation of a DeBruijn graph :param frags: a set of sequences

**in\_degree** ( v: str ) → int

Method that calculates the degree of entry of node v :param v: node :return: returns the node's degree of entry

**seq\_from\_path** ( path: list ) → str

Method that gets the sequence from the constructed path. param path: path of the list graph :return: returns the sequence represented by the constructed path

package\_aab.src.debruijn.**composition** ( k: int, seq: str ) → list

Method that recovers the original sequence, giving as input values the obtained sequence and the value of k param k: size of the fragments param seq: sequence obtained :return: returns the original sequence in list

package\_aab.src.debruijn.**prefix** ( seq: str ) → str

Method that gets the sequence prefix obtained in the "seq\_from\_path" method. :param seq: sequence represented by the constructed path :return: returns the sequence prefix seq, which corresponds to the sequence except the last character

`package_aab.src.debrujin.suffix(seq: str) → str`

Method that gets the sequence suffix obtained in the “seq\_from\_path” method. :param seq: sequence represented by the constructed path :return: returns the sequence suffix seq, which corresponds to the sequence except the first character

### 1.1.18 Module contents

- `genindex`
- `modindex`
- `search`

## p

### package\_aab

- package\_aab.src, [14](#)
- package\_aab.src.Automata, [1](#)
- package\_aab.src.BoyerMoore, [2](#)
- package\_aab.src.BWT, [2](#)
- package\_aab.src.debrujin, [13](#)
- package\_aab.src.Indiv, [3](#)
- package\_aab.src.MotifFinding, [4](#)
- package\_aab.src.MyGraph, [5](#)
- package\_aab.src.MyGraph1, [8](#)
- package\_aab.src.MyMotifs, [10](#)
- package\_aab.src.MySeq, [11](#)
- package\_aab.src.SuffixTree, [12](#)
- package\_aab.src.Trie, [13](#)



**A**

`add_edge()` (package\_aab.src.debruijn.DeBruijnGraph method), 13  
`add_edge()` (package\_aab.src.MyGraph.MyGraph method), 5  
`add_edge()` (package\_aab.src.MyGraph1.MyGraph method), 8  
`add_node()` (package\_aab.src.SuffixTree.SuffixTree method), 12  
`add_suffix()` (package\_aab.src.SuffixTree.SuffixTree method), 12  
`add_vertex()` (package\_aab.src.MyGraph.MyGraph method), 5  
`add_vertex()` (package\_aab.src.MyGraph1.MyGraph method), 8  
`all_clustering_coefs()` (package\_aab.src.MyGraph.MyGraph method), 6  
`all_degrees()` (package\_aab.src.MyGraph.MyGraph method), 6  
`allProtein()` (package\_aab.src.MySeq.MySeq method), 11  
`alphabet()` (package\_aab.src.MySeq.MySeq method), 11  
`applyNextState()` (package\_aab.src.Automata.Automata method), 1  
`Automata` (class in package\_aab.src.Automata), 1

**B**

`betweenness_centrality()` (package\_aab.src.MyGraph.MyGraph method), 6  
`BoyerMoore` (class in package\_aab.src.BoyerMoore), 2  
`branchAndBound()` (package\_aab.src.MotifFinding.MotifFinding method), 4  
`build_bwt()` (package\_aab.src.BWT.BWT method), 2  
`buildTransitionTable()` (package\_aab.src.Automata.Automata method), 1  
`bw_matching()` (package\_aab.src.BWT.BWT method), 2

`bw_matching_pos()` (package\_aab.src.BWT.BWT method), 2  
`BWT` (class in package\_aab.src.BWT), 2  
`bypass()` (package\_aab.src.MotifFinding.MotifFinding method), 4

**C**

`check_balanced_graph()` (package\_aab.src.MyGraph.MyGraph method), 6  
`check_balanced_node()` (package\_aab.src.MyGraph.MyGraph method), 6  
`check_if_hamiltonian_path()` (package\_aab.src.MyGraph.MyGraph method), 6  
`check_if_valid_path()` (package\_aab.src.MyGraph.MyGraph method), 6  
`check_nearly_balanced_graph()` (package\_aab.src.MyGraph.MyGraph method), 6  
`check_seqs()` (package\_aab.src.BWT.BWT method), 2  
`check_seqs()` (package\_aab.src.SuffixTree.SuffixTree method), 12  
`check_seqs()` (package\_aab.src.Trie.Trie method), 13  
`closeness_centrality()` (package\_aab.src.MyGraph.MyGraph method), 6  
`clustering_coef()` (package\_aab.src.MyGraph.MyGraph method), 6  
`codonTranslate()` (package\_aab.src.MySeq.MySeq method), 11  
`composition()` (in module package\_aab.src.debruijn), 13  
`consensus()` (package\_aab.src.MyMotifs.MyMotifs method), 10  
`create_deBruijn_graph()` (package\_aab.src.debruijn.DeBruijnGraph method), 13  
`createMatZeros()` (in module package\_aab.src.MyMotifs), 11  
`createMotifFromIndexes()` (package\_aab.src.MotifFinding.MotifFinding method), 4  
`createPWM()` (package\_aab.src.MyMotifs.MyMotifs method), 10

crossover() (package\_aab.src.Indiv.Indiv method), 3

## D

DeBruijnGraph (class in package\_aab.src.debruijn), 13

degree() (package\_aab.src.MyGraph.MyGraph method), 6

degree() (package\_aab.src.MyGraph1.MyGraph method), 9

distance() (package\_aab.src.MyGraph.MyGraph method), 6

distance() (package\_aab.src.MyGraph1.MyGraph method), 9

doCounts() (package\_aab.src.MyMotifs.MyMotifs method), 10

## E

eulerian\_cycle() (package\_aab.src.MyGraph.MyGraph method), 6

eulerian\_path() (package\_aab.src.MyGraph.MyGraph method), 6

exhaustiveSearch() (package\_aab.src.MotifFinding.MotifFinding method), 4

## F

find\_pattern() (package\_aab.src.SuffixTree.SuffixTree method), 12

findithocc() (package\_aab.src.BWT.BWT static method), 2

## G

get\_adjacents() (package\_aab.src.MyGraph.MyGraph method), 7

get\_adjacents() (package\_aab.src.MyGraph1.MyGraph method), 9

get\_edges() (package\_aab.src.MyGraph.MyGraph method), 7

get\_edges() (package\_aab.src.MyGraph1.MyGraph method), 9

get\_first\_col() (package\_aab.src.BWT.BWT method), 2

get\_leafes\_below() (package\_aab.src.SuffixTree.SuffixTree method), 12

get\_nodes() (package\_aab.src.MyGraph.MyGraph method), 7

get\_nodes() (package\_aab.src.MyGraph1.MyGraph method), 9

get\_predecessors() (package\_aab.src.MyGraph.MyGraph method), 7

get\_predecessors() (package\_aab.src.MyGraph1.MyGraph method), 9

get\_successors() (package\_aab.src.MyGraph.MyGraph method), 7

get\_successors() (package\_aab.src.MyGraph1.MyGraph method), 9

getFitness() (package\_aab.src.Indiv.Indiv method), 3

getGenes() (package\_aab.src.Indiv.Indiv method), 3

gibbs() (package\_aab.src.MotifFinding.MotifFinding method), 4

## H

has\_cycle() (package\_aab.src.MyGraph.MyGraph method), 7

has\_cycle() (package\_aab.src.MyGraph1.MyGraph method), 9

heuristicConsensus() (package\_aab.src.MotifFinding.MotifFinding method), 4

heuristicStochastic() (package\_aab.src.MotifFinding.MotifFinding method), 4

highest\_closeness() (package\_aab.src.MyGraph.MyGraph method), 7

highest\_degrees() (package\_aab.src.MyGraph.MyGraph method), 7

## I

in\_degree() (package\_aab.src.debruijn.DeBruijnGraph method), 13

in\_degree() (package\_aab.src.MyGraph.MyGraph method), 7

in\_degree() (package\_aab.src.MyGraph1.MyGraph method), 9

Indiv (class in package\_aab.src.Indiv), 3

IndivInt (class in package\_aab.src.Indiv), 3

IndivReal (class in package\_aab.src.Indiv), 3

initRandom() (package\_aab.src.Indiv.Indiv method), 3

initRandom() (package\_aab.src.Indiv.IndivInt method), 3

initRandom() (package\_aab.src.Indiv.IndivReal method), 3

insert() (package\_aab.src.Trie.Trie method), 13

insert\_sequence() (package\_aab.src.Trie.Trie method), 13

inverse\_bwt() (package\_aab.src.BWT.BWT method), 2

is\_connected() (package\_aab.src.MyGraph.MyGraph method), 7

is\_in\_tuple\_list() (in module package\_aab.src.MyGraph), 8

is\_in\_tuple\_list() (in module package\_aab.src.MyGraph1), 10

## L

largestOrfProtein() (package\_aab.src.MySeq.MySeq method), 11

last\_to\_first() (package\_aab.src.BWT.BWT method), 2

longestProteinSeq() (package\_aab.src.MySeq.MySeq method), 11

## M

maskedConsensus() (package\_aab.src.MyMotifs.MyMotifs method), 10

matches() (package\_aab.src.Trie.Trie method), 13

mean\_clustering\_coef() (package\_aab.src.MyGraph.MyGraph method), 7

mean\_clustering\_perdegree() (package\_aab.src.MyGraph.MyGraph method), 7

mean\_degree() (package\_aab.src.MyGraph.MyGraph method), 7

mean\_distances() (package\_aab.src.MyGraph.MyGraph method), 7

module

- package\_aab.src, 14
- package\_aab.src.Automata, 1
- package\_aab.src.BoyerMoore, 2
- package\_aab.src.BWT, 2
- package\_aab.src.debrujin, 13
- package\_aab.src.Indiv, 3
- package\_aab.src.MotifFinding, 4
- package\_aab.src.MyGraph, 5
- package\_aab.src.MyGraph1, 8
- package\_aab.src.MyMotifs, 10
- package\_aab.src.MySeq, 11
- package\_aab.src.SuffixTree, 12
- package\_aab.src.Trie, 13

mostProbableSeq() (package\_aab.src.MyMotifs.MyMotifs method), 10

MotifFinding (class in package\_aab.src.MotifFinding), 4

mutation() (package\_aab.src.Indiv.Indiv method), 3

mutation() (package\_aab.src.Indiv.IndivInt method), 3

mutation() (package\_aab.src.Indiv.IndivReal method), 3

MyGraph (class in package\_aab.src.MyGraph), 5

MyGraph (class in package\_aab.src.MyGraph1), 8

MyMotifs (class in package\_aab.src.MyMotifs), 10

MySeq (class in package\_aab.src.MySeq), 11

## N

nextSol() (package\_aab.src.MotifFinding.MotifFinding method), 4

nextState() (package\_aab.src.Automata.Automata method), 1

nextVertex() (package\_aab.src.MotifFinding.MotifFinding method), 5

node\_has\_cycle() (package\_aab.src.MyGraph.MyGraph method), 7

node\_has\_cycle() (package\_aab.src.MyGraph1.MyGraph method), 9

## O

one\_pt\_crossover() (package\_aab.src.Indiv.Indiv method), 3

orfs() (package\_aab.src.MySeq.MySeq method), 11

out\_degree() (package\_aab.src.MyGraph.MyGraph method), 8

out\_degree() (package\_aab.src.MyGraph1.MyGraph method), 9

overlap() (in module package\_aab.src.Automata), 1

## P

package\_aab.src

- module, 14

package\_aab.src.Automata

- module, 1

package\_aab.src.BoyerMoore

- module, 2

package\_aab.src.BWT

- module, 2

package\_aab.src.debrujin

- module, 13

package\_aab.src.Indiv

- module, 3

package\_aab.src.MotifFinding

- module, 4

package\_aab.src.MyGraph

- module, 5

package\_aab.src.MyGraph1

- module, 8

package\_aab.src.MyMotifs

- module, 10

package\_aab.src.MySeq

- module, 11

package\_aab.src.SuffixTree

- module, 12

package\_aab.src.Trie

- module, 13

patternSeqPosition() (package\_aab.src.Automata.Automata method), 1

prefix() (in module package\_aab.src.debrujin), 13

print\_graph() (package\_aab.src.MyGraph.MyGraph method), 8

print\_graph() (package\_aab.src.MyGraph1.MyGraph method), 9

print\_tree() (package\_aab.src.SuffixTree.SuffixTree method), 12

print\_trie() (package\_aab.src.Trie.Trie method), 13

printAutomata() (package\_aab.src.Automata.Automata method), 1

printMat() (in module package\_aab.src.MyMotifs), 11

printSeq() (package\_aab.src.MySeq.MySeq

method), 11  
prob\_degree() (package\_aab.src.MyGraph.MyGraph method), 8  
probabSeq() (package\_aab.src.MyMotifs.MyMotifs method), 11  
probAllPositions() (package\_aab.src.MyMotifs.MyMotifs method), 10  
process\_bcr() (package\_aab.src.BoyerMoore.BoyerMoore method), 2  
process\_gsr() (package\_aab.src.BoyerMoore.BoyerMoore method), 3

## R

random() (in module package\_aab.src.MotifFinding), 5  
reachable\_bfs() (package\_aab.src.MyGraph.MyGraph method), 8  
reachable\_bfs() (package\_aab.src.MyGraph1.MyGraph method), 9  
reachable\_dfs() (package\_aab.src.MyGraph.MyGraph method), 8  
reachable\_dfs() (package\_aab.src.MyGraph1.MyGraph method), 9  
reachable\_with\_dist() (package\_aab.src.MyGraph.MyGraph method), 8  
reachable\_with\_dist() (package\_aab.src.MyGraph1.MyGraph method), 9  
readFile() (package\_aab.src.MotifFinding.MotifFinding method), 5  
reverseComplement() (package\_aab.src.MySeq.MySeq method), 11  
rnaCodon() (package\_aab.src.MySeq.MySeq method), 11  
roulette() (package\_aab.src.MotifFinding.MotifFinding method), 5

## S

score() (package\_aab.src.MotifFinding.MotifFinding method), 5  
scoreMult() (package\_aab.src.MotifFinding.MotifFinding method), 5  
search\_hamiltonian\_path() (package\_aab.src.MyGraph.MyGraph method), 8  
search\_hamiltonian\_path\_from\_node() (package\_aab.src.MyGraph.MyGraph method), 8  
search\_pattern() (package\_aab.src.BoyerMoore.BoyerMoore method), 3  
seq\_from\_path() (package\_aab.src.debruijn.DeBruijnGraph method), 13  
seqSize() (package\_aab.src.MotifFinding.MotifFinding method), 5  
seqTranslation() (package\_aab.src.MySeq.MySeq method), 11  
set\_bwt() (package\_aab.src.BWT.BWT method), 2  
setFitness() (package\_aab.src.Indiv.Indiv

method), 3  
shortest\_path() (package\_aab.src.MyGraph.MyGraph method), 8  
shortest\_path() (package\_aab.src.MyGraph1.MyGraph method), 9  
size() (package\_aab.src.MyGraph.MyGraph method), 8  
size() (package\_aab.src.MyGraph1.MyGraph method), 10  
suffix() (in module package\_aab.src.debruijn), 14  
suffix\_tree\_from\_seq() (package\_aab.src.SuffixTree.SuffixTree method), 12  
SuffixTree (class in package\_aab.src.SuffixTree), 12

## T

test() (in module package\_aab.src.SuffixTree), 12  
test1() (in module package\_aab.src.MyGraph1), 10  
test1\_1() (in module package\_aab.src.MotifFinding), 5  
test1\_2() (in module package\_aab.src.MotifFinding), 5  
test1\_3() (in module package\_aab.src.MotifFinding), 5  
test1\_4() (in module package\_aab.src.MotifFinding), 5  
test1\_5() (in module package\_aab.src.MotifFinding), 5  
test2() (in module package\_aab.src.MotifFinding), 5  
test2() (in module package\_aab.src.MyGraph1), 10  
test3() (in module package\_aab.src.MyGraph1), 10  
test4() (in module package\_aab.src.MyGraph1), 10  
test5() (in module package\_aab.src.MyGraph1), 10  
transcription() (package\_aab.src.MySeq.MySeq method), 12  
Trie (class in package\_aab.src.Trie), 13

## V

valida() (package\_aab.src.MySeq.MySeq method), 12  
validateSeqRE() (package\_aab.src.MySeq.MySeq method), 12