Procura de motifs

Algoritmos exato (Branch-and-Bound), heurístico (tipo consensus) e heurístico (tipo estocástico)

Implementando Branch-and-Bound

Exemplo de sequencias para alinhamento:

seqs = [“cattgccc”, “aaattggt”, “tttattgt”, “ggattgaa”]

Definimos que o tamanho do motif é de 4.

Se alinhamos:

```
c a t t g c c c
a a a t t g g t
t t t a t t g t
g g a t t g a a
```

Podemos representar os motifs nas sequencias com uma lista das primeiras posições do motif na sequência:

first_pos_seq = [1, 2, 3, 2]

Implementando Branch-and-Bound

```
class MotifFinding:

        # BRANCH AND BOUND

        def bypass (self, s):

                …

        def nextVertex (self, s):

                …

        def branchAndBound (self):

                …
```

bypass – recebe uma lista das primeiras posições do motif nas sequências.
Retorna o input sem os ramos (posições) do prefixo a descartar.

nextVertex – recebe uma lista das primeiras posições do motif nas sequências.
Retorna o input com o seguinte ramo a ser avaliado com scoring ou então passa para a seguinte arvore de prefixos (faz **bypass**).

branchAndBound - retorna uma lista das primeiras posições do motif nas sequências com melhor scoring.

Implementando Branch-and-Bound

```
def bypass (self, s):
        pos =  last position of s
        loop while
                pos is bigger or equal to 0;
                pos in S is equal to (seqs size on pos – motif size) :
                        decrease pos by 1 in each loop.
        if pos is lower than 0:
                return None
        else:
                res = list from 0 to pos of elements from s
                append pos element s incremented by 1 to res
                return res
```

```
def nextVertex (self, s):
        if len of s is smaller than size of seqs:
                res = copy of s
                append 0 to res
                return res
        else:
                return bypass list
```

Implementando Branch-and-Bound

```
def branchAndBound (self, s):
    define best_score equal to -1
    define best_motif equal to None
    define size with len of seqs
    define s with a list of zeros with len size
    loop while s is not none:
        if len of s is lower than size:
            define max_s _score equal to score of s + ( size – len of s) * size of motif
            if max_s_score is less than best_score:
                s equal to bypass of s
                (move s to next tree and discard leafs of this tree)
            else:
                s equal to next vertex of s
                (move s to next leaf of the tree)
        else:
            define score with score of s
            if score is bigger than best_score:
                define best_score equal to score
                define best_motif with copy of s
                define s with next vertex of s
                (move s to next leaf of the tree)
    return best_motif
```

Implementando algoritmo heurístico (tipo Consensus)

```
class MotifFinding:

        # Consensus (heuristic)

        def heuristicConsensus (self):

                ...
```

heuristicConsensus – retorna uma lista das primeiras posições do motif nas sequências com o melhor scoring obtido a partir das duas primeiras sequencias (fixas) com o melhor scoring.

Implementando algoritmo heurístico (tipo Consensus)

```
def heuristicConsensus (self):
        define s with a list of zeros with len size
        define max_score with -1
        for index i in range of (sequence size at 0 – motif size):
                for index j in range of (sequence size at 1 – motif size):
                        define partial_s equal to [I,j]
                        define score equal to score of partial_s
                        if score is bigger than max_score:
                                define max_score with score
                                set first two elements of s with partial_s elements
        for index k in range of 2 to size of seqs:
                define max_score with -1
                define partial_s with res elements until k + a zero in the end the array
                 for index i in range of (sequence size at k – motif size):
                        define partial_s at position k is equal to i index
                        define score equal to score of partial_s
                        if score is bigger than max_score:
                                define max_score equal to score
                                define s at position k equal to i
        return s
```

Implementando algoritmo estocástico (tipo Consensus)

```
class MotifFinding:

        # Consensus (heuristic)

        def heuristicStochastic (self):

                ...
```

heuristicStochastic – retorna uma lista das primeiras posições do motif nas sequências com o melhor scoring obtido de uma lista gerada aleatoriamente das primeiras posições do motif.

Implementando algoritmo estocástico (tipo Consensus)

```
def heuristicStochastic (self):
        define s with a list of zeros with len size
        for index k in range of len of s:
                define s at positon k with random int
                        from 0 until of (sequence size at k index – motif size)
        define profile with create motif from indexes from s
        define best_score with -1
        define best_s with None
        loop while score multi of s is bigger than best_score:
                define best_score with score multi of s
                define best_s with copy of s
                for index k in range of len of s:
                        define s at position k with most probable seq
                        with the profile from sequence at k position
                define profile with create motif from indexes from s
        return best_s
```

Implementando Gibbs Sampling

```
class MotifFinding:

        # Gibbs sampling

        def gibbs (self, numits):
                …
```

gibbs – retorna uma lista das primeiras posições do motif nas sequências com o melhor scoring obtido de uma lista gerada aleatoriamente das primeiras posições do motif que posteriormente foram iteradas de forma ao score aumentar ao máximo.

Implementando Gibbs Sampling

```
def gibbs (self, numits):
        define s list
        for k index in range of len of seqs:
                append to s a random int
                        from 0 until of (sequence size at k index – motif size)
        define best_s with copy of s
        define best_score with score multi of s
        for _ in range of numits:
                define seq_index with random int from 0 to len of seqs – 1
                define seq_selected from seqs with seq_index
                pop from s the seq_index
                define removed from pop of seqs using the seq_index
                define profile with s using create Motif From Indexes
                reinsert the removed seq in seqs with the seq_index position
                define r with all prob positons from profile using the seq_selected
                define pos with roulette method using r
                insert in s at seq_index position the pos
                define score with the multi score from s
                if score is bigger than best_score:
                        define best_score equal to score
                        define best_s with copy of s
        return best_s
```