**Descrição do algoritmo**

O BLAST (Basic Local Alignment Search Tool) é um dos algoritmos mais utilizados atualmente para pesquisas em bases de dados de sequências biológicas. O objetivo principal do BLAST é identificar alinhamentos locais de alta qualidade entre uma sequência de consulta (query) e as sequências presentes em uma base de dados.

O algoritmo funciona por meio de uma estratégia baseada em "palavras" curtas (seeds), que são subsequências extraídas da query. Essas seeds são comparadas com subsequências das sequências na base de dados para encontrar correspondências exatas ou quase exatas. Após identificar essas correspondências iniciais, o BLAST expande essas regiões em ambas as direções para formar alinhamentos maiores. O processo de expansão continua até que a pontuação do alinhamento caia abaixo de um valor pré-definido, conhecido como threshold.

Na implementação simplificada do BLAST apresentada aqui, o processo foi dividido em quatro etapas principais, cada uma realizada por uma função específica:

1. **query\_map**: Esta função realiza o mapeamento da query, gerando as seeds (subsequências) e suas posições na query.
2. **hits**: Identifica as correspondências entre as seeds da query e as subsequências da sequência alvo na base de dados.
3. **extend\_hit**: Expande as correspondências identificadas pela função hits em ambas as direções, até que a pontuação do alinhamento caia abaixo do threshold.
4. **best\_hit**: Avalia todos os alinhamentos expandidos e seleciona aquele com a melhor pontuação, representando o melhor alinhamento encontrado entre a query e a sequência da base de dados.

**Projeto de baixo nível**

**Função *query\_map***

Esta função é responsável por dividir a sequência de consulta (query) em subsequências menores, chamadas de "**seeds**” (ou palavras), com tamanho definido pelo parâmetro **w** que representa a janela de procura. Cada seed gerada é armazenada como uma chave em um dicionário, onde o valor associado a cada chave é uma lista das posições em que essa seed ocorre na sequência de consulta. Essa estrutura de dados será usada nas etapas seguintes para identificar possíveis correspondências entre a query e as sequências da base de dados.

**Função *hits***

A função hits utiliza o dicionário gerado pela função query\_map e uma sequência alvo para identificar correspondências (hits) entre as palavras da query e a sequência alvo. Para cada chave no dicionário, a função encontra todas as posições em que esta aparece na sequência alvo e associa essas posições às posições correspondentes na query. O resultado final é uma lista de tuplos, onde cada tuplo contém dois valores: o índice da seed na query e o índice correspondente na sequência alvo.

**Função *extend\_hit***

Esta função recebe a query, a sequência alvo, o hit da função hits (um par de índices iniciais na query e na sequência alvo), e o parâmetro w. A função tenta estender o hit para os dois lados (esquerda e direita), desde que a proporção de correspondências seja de pelo menos 50% do tamanho da extensão.

A função devolve um tuplo contendo:

1. O índice inicial do hit estendido na query;
2. O índice inicial do hit estendido na sequência alvo;
3. O tamanho total do alinhamento estendido;
4. O número total de correspondências.

**Função *best\_hit***

A função best\_hit combina as etapas anteriores para identificar o melhor alinhamento global entre a query e a sequência alvo. Ela utiliza as seeds geradas pela função query\_map, identifica hits iniciais com a função hits, e expande esses hits com a função extend\_hit. Entre todos os alinhamentos obtidos, o melhor é escolhido com base na maior pontuação global, que corresponde ao maior número de correspondências corretas no alinhamento estendido.

**Projeto de baixo nível**

**Função *query\_map***

**Objetivo**

A função query\_map tem como objetivo criar um dicionário que associa subsequências (ou "seeds") da query às posições em que essas subsequências aparecem na query.

**Entradas**

1. **query** (str): A sequência de consulta (query), que será dividida em subsequências.
2. **window\_size** (int): O tamanho da janela de procura, ou seja, o comprimento de cada subsequência (seed) gerada.

**Saída**

**dict\_seeds** (defaultdict): Um dicionário onde:

* As **chaves** são todas as subsequências (seeds) de tamanho window\_size encontradas na query.
* Os **valores** são listas com as posições iniciais em que cada seed aparece na query.

**Passos Internos**

1. Criar um dicionário vazio (usando defaultdict) para armazenar as seeds e suas posições.
2. Iterar sobre a sequência seq utilizando uma janela deslizante de tamanho window\_size.
3. Para cada subsequência extraída:
   * Adicionar a subsequência como chave no dicionário.
   * Adicionar a posição inicial da subsequência à lista correspondente no dicionário.
4. Devolver o dicionário resultoante

**Pseudocódigo**

**Função *get\_al\_positions***

FUNÇÃO query\_map(seq, window\_size):

Criar dicionário dict\_seeds como defaultdict(lista)

PARA cada posição i na query até (len(query) - window\_size + 1):

Extrair subsequência query[i : i + window\_size]

Adicionar i na lista associada à subsequência no dicionário dict\_seeds

Devolver dict\_seeds

**Objetivo**

A função get\_all\_positions localiza todas as posições iniciais de uma subsequência específica em uma sequência maior. É uma função auxiliar usada para encontrar correspondências exatas.

A função **get\_all\_positions** é fundamental para identificar correspondências exatas entre uma subsequência da query e a sequência alvo. Ela é usada pela função **hits** para criar pares de correspondência.

**Entradas**

1. **subseq** (str): A subsequência que se deseja localizar na sequência alvo.
2. **seq** (str): A sequência alvo onde será feita a busca.

**Saída**

* **positions** (list[int]): Uma lista contendo todas as posições iniciais em que a subsequência ocorre na sequência alvo.

**Passos Internos**

1. Definir uma lista vazia para armazenar as posições encontradas.
2. Iterar sobre a sequência alvo com base no comprimento da subsequência.
3. Comparar cada subsequência extraída com subseq:
   * Se houver correspondência, armazenar a posição inicial.
4. Retornar a lista de posições encontradas.

**Pseudocódigo**

**Função *hits***

FUNÇÃO get\_all\_positions(subseq, seq):

Criar lista vazia positions

PARA cada posição i em seq até (len(seq) - len(subseq) + 1):

Se seq[i : i + len(subseq)] for igual a subseq:

Adicionar i à lista positions

Devolver a lista positions

9

**Objetivo**  
Esta função identifica todos os pares de posições correspondentes entre as subsequências (seeds) da query e a sequência alvo. Cada seed gerada pela função query\_map é comparada com todas as suas ocorrências na sequência alvo, gerando uma lista de tuplos que representam os alinhamentos iniciais (hits).

**Entradas**

1. qm (defaultdict): O dicionário gerado pela função query\_map, onde:
   * As **chaves** são as seeds extraídas da query.
   * Os **valores** são listas de posições dessas seeds na query.
2. seq (str): A sequência alvo onde os hits serão buscados.

**Saídas**

* list[tuple[int, int]]:  
  Uma lista de tuplos, onde cada tuplo contém:
  + A posição inicial da seed na query (P1).
  + A posição inicial correspondente da mesma seed na sequência alvo (P2).

**Passos Internos**

1. Definir uma lista vazia res para armazenar os hits encontrados.
2. Iterar sobre cada seed (**chave**) e sua lista de posições (**valores**) no dicionário qm.
   * Para cada posição da seed na query (P1), buscar todas as posições da mesma seed na sequência alvo (P2) usando a função get\_all\_positions.
   * Para cada par de posições (P1, P2), adicionar o tuplo correspondente à lista res.
3. Retornar a lista res contendo todos os hits encontrados.

**Pseudocódigo**

FUNÇÃO hits(qm, seq):

hits\_l ← lista vazia

PARA cada seed Q em qm:

PARA cada posição P1 em qm[Q]: # posições da seed na query

PARA cada posição P2 obtida por get\_all\_positions(Q, seq): # posições da seed

Adicionar o tuplo (P1, P2) à lista hits\_l

Devolver a lista hits\_l

**Função *extend\_hit***

**Objetivo**  
Expandir um hit inicial (alinhamento local) entre a query e a sequência alvo. A extensão ocorre em ambas as direções (esquerda e direita), enquanto a proporção de correspondências (matches) permanece acima de 50%.

**Entradas**

1. query (str): A sequência de consulta (query).
2. seq (str): A sequência alvo.
3. hit (tuple[int, int]): Um tuplo representando um alinhamento inicial (hit):
   * O índice inicial da seed na query.
   * O índice inicial correspondente na sequência alvo.
4. w (int): O tamanho da seed usada no alinhamento inicial.

**Saídas**

* tuple[int, int, int, int]: Um tuplo contendo:
  + O índice inicial do alinhamento estendido na query.
  + O índice inicial do alinhamento estendido na sequência alvo.
  + O comprimento total do alinhamento estendido.
  + O número total de correspondências (matches) no alinhamento.

**Passos Internos**

1. **Definição**:
   * Calcular os índices inicial (query\_i, hit\_i) e final (query\_f, seq\_f) do hit na query e na sequência alvo.
   * Estabelecer o número inicial de correspondências como o tamanho da seed (w).
   * Determinar se é possível expandir para a esquerda e direita verificando os limites das sequências.
2. **Extensão à Esquerda**:
   * Enquanto houver espaço para expandir para a esquerda e a proporção de matches for suficiente (≥ 50%):
     + Diminuir os índices iniciais (query\_i e hit\_i).
     + Aumentar o comprimento do alinhamento.
     + Comparar os caracteres da query e da sequência alvo nas novas posições:
       - Aumentar o número de matches se os caracteres forem iguais.
   * Atualizar a condição de paragem.
3. **Extensão à Direita**:
   * Enquanto houver espaço para expandir para a direita e a proporção de matches for suficiente (≥ 50%):
     + Aumentar os índices finais (query\_f e seq\_f).
     + Aumentar o comprimento do alinhamento.
     + Comparar os caracteres da query e da sequência alvo nas novas posições:
       - Aumentar o número de matches se os caracteres forem iguais.
   * Atualizar a condição de paragem.
4. Devolver o resultado como um tuplo: os índices iniciais estendidos (query\_i, hit\_i), o comprimento do alinhamento estendido, e o número total de correspondências.

**Pseudocódigo**

FUNÇÃO extend\_hit(query, seq, hit, w):

Definir query\_i e hit\_i como os índices iniciais do hit na query e na sequência alvo, respetivamente.

Definir query\_f e seq\_f como os índices finais do hit na query e na sequência alvo, respetivamente.".

correspondencias ← w

extensao ← w

Enquanto for possível expandir para a esquerda E correspondencias ≥ extensao / 2:

Diminuir query\_i e hit\_i.

Aumentar extensao.

Se query[query\_i] == seq[hit\_i]:

Aumentar correspondencias.

Enquanto for possível expandir para a direita E correspondencias ≥ extensao / 2:

Aumentar query\_f e seq\_f.

Aumentar extensao.

Se query[query\_f] == seq[seq\_f]:

Aumentar correspondencias.

Devolver (query\_i, hit\_i, query\_f - query\_i + 1, correspondencias)

**Função *best\_hit***

**Objetivo**

Identificar o melhor alinhamento entre uma query e uma sequência alvo. A função utiliza os resultados das funções query\_map, hits e extend\_hit para encontrar e avaliar alinhamentos locais. O melhor alinhamento é aquele com o maior número de correspondências (matches).

**Entradas**

1. **query** (str): A sequência de consulta, que representa o padrão a ser alinhado.
2. **seq** (str): A sequência alvo, onde buscamos os melhores alinhamentos.
3. **w** (int): O tamanho da janela de procura, que determina o comprimento das seeds iniciais geradas.

**Saídas**

**tuple[int, int, int, int]**: Um tuplo contendo:

1. **Índice inicial na query**: A posição inicial do melhor alinhamento na query.
2. **Índice inicial na sequência**: A posição inicial do melhor alinhamento na sequência alvo.
3. **Tamanho do alinhamento**: O comprimento do melhor alinhamento encontrado.
4. **Número de correspondências**: A contagem de matches corretos no melhor alinhamento.

**Passos Internos**

**1. Gerar o mapa de seeds**

* A função utiliza query\_map para gerar um dicionário que mapeia as subsequências (seeds) da query e suas posições.
  + **Entrada**: A query e o tamanho da janela (w).
  + **Saída**: Um dicionário contendo cada seed como chave e as posições correspondentes como valores.

**2. Encontrar hits iniciais**

* Com o dicionário gerado, a função chama hits para identificar todos os pares de posições correspondentes entre as seeds da query e a sequência alvo.
  + **Entrada**: O dicionário de seeds e a sequência alvo.
  + **Saída**: Uma lista de tuplos, onde cada tuplo contém a posição de uma seed na query e na sequência alvo.

**3. Avaliar cada hit**

* Para cada hit da lista, a função chama extend\_hit para expandi-lo, calculando o alinhamento completo.
  + **Entrada**: A query, a sequência alvo, o hit inicial e o tamanho da janela.
  + **Saída**: Um tuplo contendo os índices iniciais, o tamanho do alinhamento e o número total de correspondências.

**4. Escolher o melhor hit**

* A função utiliza a função embutida max para selecionar o alinhamento com o maior número de correspondências.
  + **Critério de desempate**: Se dois hits tiverem o mesmo número de correspondências, o primeiro hit encontrado é priorizado.
  + **Entrada**: A lista de hits estendidos.
  + **Saída**: O melhor alinhamento representado por um tuplo com as informações do índice inicial, tamanho do alinhamento e número de matches.

**Resumo do Funcionamento**

A função best\_hit combina as etapas principais do algoritmo BLAST:

1. Gera seeds da query (query\_map).
2. Encontra hits iniciais entre a query e a sequência alvo (hits).
3. Expande cada hit para maximizar o alinhamento (extend\_hit).
4. Seleciona o melhor alinhamento com base no número de correspondências.

Esta função sintetiza todo o processo e retorna o resultado final do alinhamento, que pode ser utilizado para análise biológica detalhada.