Algorithm 1: BFA Algorithm

```
Input: Multiple protein sequences
Output: Identical sequence repeats
Def CollectAllSequence(path):
    *...Read the file from given path...*
    List=[]                                              *..vector of strings..*
    *...Extract individual sequences and append it into the list...*
    return List
Def FindingIdenticalRepeat(List of sequences):
    n=Total no of sequences
    max_length=max(length of sequences)
    step_length=5
    *...to store repeats...*
    collect_motifs=[]                                    *..vector of strings..*
    *...to store number of occurrences of repeat...*
    collect_motifs_frequency=[]                          *..vector of integers..*
    while step_length ≤ max_length do
        splitted=[]
        for i in range of n do
            if length of i^th sequence ≥ step_length then
                split=[splitting of i^th sequence by equal step_length characters]
                splitted.append(split)
        joined=[]
        for i in range size of splitted do
            key=splitted[i]
            for j in range (size of key) do
                joined.append(j)
        B=list(set(joined))
        if size of B < size of joined then
            *...Repeat is present...*
            for j in range size of B do
                motif=b[j]
                value=Joined.count(motif)
                if value > 1 then
                    collect_motif.append(motif)
                    collect_motifs_frequency.append(value)
            else
                *...Do nothing...*
        step_length+=1
    return collect_motifs, collect_motifs_frequency
Function Main:
    path=*...path to .fasta or .txt file of protein sequences...*
    List_of_sequences=CollectAllSequence(path)
    A,B=FindingIdenticalRepeat(List_of_sequences)
    Identical_Repeats=A
    Repeat_Frequency=B
    return Identical_Repeats, Repeat_Frequency
```

Algorithm 2: STBA : Suffix Tree Based Approach

Input: Multiple protein sequences

Output: Identical sequence repeats

Def `CollectAllSequen(`*path*`):`

    ...Read the file from given path...
    List=[]
    ...Extract individual sequences and append it into the list...
    return List

Def `JoinSequence(`*sequences, spl_char*`):`

    CombinedSeq=¨
    ...CombinedSeq is a empty string...
    for *i in range number of sequences* **do**
        CombinedSeq+=sequences[i]+spl_char[i]
    return CombinedSeq

Class `Node`**(object):**

    ...Construct the Node ...

Class `Edge`**(object):**

    ...Construct the Edge ...

Class `SuffixTree`**(object):**

    ...Construct the SuffixTree using Ukkonen's algorithm ...

Def `FindingIdenticalRepeat(`*path, spl_char*`):`

    sequences=**CollectAllSequence**(path)
    CombinedSeq=**JoinSequence**(sequences,spl_char)
    ...Instantiation of SuffixTree...
    ...Constructing SuffixTree using CombineSeq...
    ...Finding All Internal nodes of SuffixTree...
    Count_Of_Leaves= *...Count total number of leaves joined with each internal node...*
    BFS_path= *...applying **Breadth First Search (BFS)** to reach each **Internal Node** from the **root**...*
    ...Store BFS path and count of leaves associated with each internal node...
    motifs=[BFS_path]
    motif_frequency=[Count_Of_Leaves]
    return motifs, motif_frequency

Function `Main`**:**

    path=*...path to **.fasta** or **.txt** file of protein sequences...*
    motifs,motif_frequency=**FindingIdenticalRepeat**(path)
    return motifs,motif_frequency

Algorithm 3: RKBM: Rabin-Karp Linear Time pattern searching based method

```
Input: Multiple protein sequences
Output: Identical sequence repeats
Def CollectAllSequence(path):
    *...Extract individual sequences and append it into the list...*
    List=[]                                  *...This is list of sequences...*
    return List
Def HashNumeric(Amino_Acids, spl_char, max_num):
    num=max_num+3
    HashValue={}
    *...HashValue is a dictionary or hash table consist of numbers assigned
     corresponding to each amino acid and special characters present in Joined protein
     sequences...*
    HashValue={}
    for i in range all type of Amino_Acids do
        HashValue[Amino_Acids[i]]=ord(Amino_Acids[i])
        *...ord gives ASCII CODE for characters of Amino_Acids...*
    for i in range total spl_char do
        HashValue[spl_char[i]]=num
        num+=1
    return HashValue
Def JoinSequences(sequences, spl_char):
    CombinedSeq=¨ ¨                  *...CombinedSeq is a empty string...*
    for i in range number of sequences do
        CombinedSeq+=sequences[i]+spl_char[i]
    return CombinedSeq
Def RabinKarpA(MidValue, nums, p, l, S):
    sl_wd_hash=0
    *...sl_wd_hash stands for sliding window hash value for each slide of window...*
    for i in range MidValue do
        sl_wd_hash=(sl_wd_hash × l+nums[i])%p
    Hashes={sl_wd_hash}
    index=-1
    MaxValue=pow(l, MidValue, p)
    for i in range MidValue to length of S do
        sl_wd_hash=(l× sl_wd_hash-nums[i-MidValue]× MaxValue +nums[i])%p
        if sl_wd_hash is in Hashes then
            index=i+1-MidValue
            Hashes.add(sl_wd_hash)
    return index
```

```
Def RabinKarpB(step, nums, l, S):
    p=2^63 − 1
    sl_wd_hash=0
    for i in range step do
        sl_wd_hash=(sl_wd_hash × l+nums[i])%p
    hashes=[sl_wd_hash]
    MaxValue=pow(l,step,p)
    for i in range (step to length of S) do
        sl_wd_hash=(l× sl_wd_hash-nums[i-step]×MaxValue +nums[i])%p
        hashes.append(sl_wd_hash)
    hash_1={} hash_2={} hash_3={}
    for i in range size of hashes do
        if hashes[i] not in hash_1 then
            hash_1[hashes[i]]=1
            hash_2[hashes[i]]=[i]
        else
            hash_1[hashes[i]]+=1 indexes.append(i)
            hash_2[hashes[i]]=indexes
    for key in hash_1 do
        if hash_1[key] > 1 then
            hash_3[key]=[hash_1[key] ,hash_2[key], step]
    return hash_3
Def LongestDuplicatedRepeats(S, l):
    p=2^63 − 1
    start=0
    end=(size_of_S -1)
    coded=HashNumeric(Amino_Acids, spl_char, max_num)
    nums=[]
    for c in S do
        nums.append(coded[c]-50)
    while start ≤ end do
        MidValue=(start+end) // 2
        index=RabinKarpA(MidValue, nums, p, l, S)
        if index = -1 then
            end=MidValue -1
        else
            begin=index
            start=MidValue + 1
    MaxLen=begin+start-1-begin
    dict=RabinKarpB(MaxLen, nums, l, S)
    return dict, MaxLen
```

Def `NewCombineSeqCreation(`*CombineSeq,Longest_Repeats, remove_index*`):`

...This function will remove all occurrences of repeats (except one) from old CombineSeq and create new CombineSeq by joining residual sequences by spl_char or 3 special characters distinct combinations...

...Keeping one Longest_Repeat in CombineSeq will help to preserve info related to it in joined sequence...

return CombineSeq

Def `IdenticalRepeatsRecursion(`*CombineSeq, pattern_length, dict*`):`

while *pattern_length > 4* **do**

...Collect the Repeat...

...Create New CombineSeq...

CombineSeq=**NewCombineSeqCreation**(CombineSeq,Longest_Repeats, remove_index)

...find Longest Duplicated repeats in New CombineSeq...

dict, MaxLen=**LongestDuplicatedRepeats**(S, l)

pattern_length=MaxLen

...Repeat the above process...

IdenticalRepeats=**IdenticalRepeatsRecursion**(CombineSeq, pattern_length, dict)

return IdenticalRepeats

Function `Main` **():**

Amino_Acids='ARNDCQEGHILKMFPSTWYVX'

spl_char=[,#,&,{...so on]

...spl_char is collection of unique special characters which is not element of amino acids...

...It will be used to combined multiple protein sequences...

path= *...path to **.fasta** or **.txt** file of protein sequences...*

sequences=**CollectAllSequence**(path)

CombineSeq=**JoinSequences**(sequences, spl_char)

l=21+number_of_sequences

dict, pattern_length=**LongestDuplicatedRepeats**(CombineSeq, l)

IdenticalRepeats=**IdenticalRepeatsRecursion**(CombineSeq, pattern_length, dict)