天津大学

2020-2021 学年第一学期
《高级算法》结课作业

题　　目：	Pair Wised Sequence Alignment using Suffix Tree
学　　院：	智能与计算学部
专　　业：	计算机科学与技术专业
教　　师：	宫秀军老师
学　　号：	2020244002
姓　　名：	刘冬冬
邮　　箱：	3463264078@qq.com

2021 年　1 月 12　日

Abstract

[Abstract]: With the continuous deepening of bioinformatics research, a large amount of bioinformatics has been produced and is still in constant flow production. If the processing of this information can be speeded up, the mystery of life can be better revealed to get a promising future.

For this reason, researchers use indexing techniques to improve the processing of sequences. For the smaller sequence, the suffix tree is undoubtedly a good choice, but there is a ″memory bottleneck″ problem that makes it unsuitable for long genes sequence; suffix array is another good choice, but it occupies more storage space than suffix tree occupies storage space. Although time is less, but the performance of the suffix array is much lower than that of the suffix tree; the search based on and although the citation structure can be used for fast matching, it cannot do anything about gene sequences with low similarity.

In this paper, based on the deep study of the classical suffix tree[1], we present Binary-Tree based on a suffix tree index structure. Binary-Tree restores all short biology sequences, which divided by Longest common sequence[2] which is calculated using suffix-tree. The Binary-Tree based algorithm solve the “memory bottleneck” that the classical suffix tree exists, and then in order to construct a suffix tree, we sort the short biology sequences by alphabetical order and according to the biggest public prefix of the two adjacent biology sequences confirm the branch point. During constructing the suffix tree, and align all the sequence in one time, through experiment, we can see that the length of sequence data is two long, that we can’ t align sequence in one time, considering computer requirements, such as the memory, and computing time. So we use suffix tree to find the longest common sequence and use the longest common sequence find aforementioned to split original sequence and construct the binary tree. After

constructing the binary tree, we using the middle order algorithm to go over the whole tree to get the alignment sequence and the alignment cost. And for each node, we use Needleman-Wunsch algorithm[3] to align partial sequence and calculate the score using dynamic algorithm.

Introduction

[Introduction]:

Bioinformatics is not only one of the most important frontiers of life sciences today, but also one of the core areas of science. With the deepening of bioinformatics research and the development of the human genome blueprint Completed, the amount of biological information has increased sharply, especially the number of gene sequences has increased exponentially, with the amount of gene, it is particularly important to retrieve the required information quickly and efficiently in the sequence. The most typical feature of a gene sequence database is that it has an extremely large data size. For example, GenBank[4], a database established by the National Biotechnology Information Center is increasing exponentially. The number of nucleotide bases doubles every month.

Sequence alignment aim to discover maximally similar (or identical) amino acid or identical nitrogenous base positions across the aligned query set of sequences. Sequence wised alignment is an essential preprocess to many bioinformatics analyses, including homology modeling, secondary structure prediction, phylogenetic reconstruction, and protein structure, and function prediction. Sequence alignment is visualized as two-dimensional matrix in which the rows are the individual sequences and the columns are maximally similar or identical amino acid or nitrogenous base positions arranged to correspond by inserting gap characters in appropriate positions. Sequence alignment can provide a wealth of information about the structure or function relationships within a set of sequences, such as the evolutionary conservation of functionally or structurally important sites, and conserved hydrophobicity patterns in precise regions.

Methods

[Methods]:

Question: Given genome sequences of mycobacterium tuberculosis (MTB) 35 spices, we are required to design an algorithm to make alignment between any two of spices using suffix tree methods, and the requirements are as follows:

- Presenting the principle, workflow and codes of the algorithm;
- Implementing the algorithm using c/c++ or python(can modify existing algorithms to meet your needs);
- Reporting your results for 35*34 experiments including spend time, used memory of suffix tree, aligned scores and the final alighment;
- Comparing your results with BLAST and FASTA algorithm.

Suffix tree pairwise alignment: the basis of our method is a data structure known as a suffix tree, which is a compressed (Aho and Corasick, 1975)[5] containing all the suffixes of a given text as keys, and positions in the text as values, suffix trees allow particularly fast implementations of many important string operations(Baeza-Yates and Gonnet, 1996)[6].And the definition is that, the tree has exactly n+1 leaves numbered from 0 to n. Except for the root, every internal node has at least two children. Each edge is labeled with a nonempty substring of S, No two edge starting out of a node can have string labels beginning with the same character. The string obtained by concatenating all the string labels found on the path from the root to leaf I spells out a suffix, a simple substring that begins at any position in S and extends to the end of S. The efficient algorithms given by Ukkonen can construct a suffix tree in linear time with O(n) computational space from a string, where n is the length of the string. This methods also reduces suffix tree construction to O(n) time, for constant-size alphabets, and O(nlogn) in general.

Original methods proposed in the work[7], given two sequences which are needed to be aligned, we call the sequence S1 and sequence S2 for convenience, it is assumed that the sequence to be compared are highly similar. Therefore, there are many common substrings in both S1 and S2. The alignment process consists of four steps, which is show in figure 1.

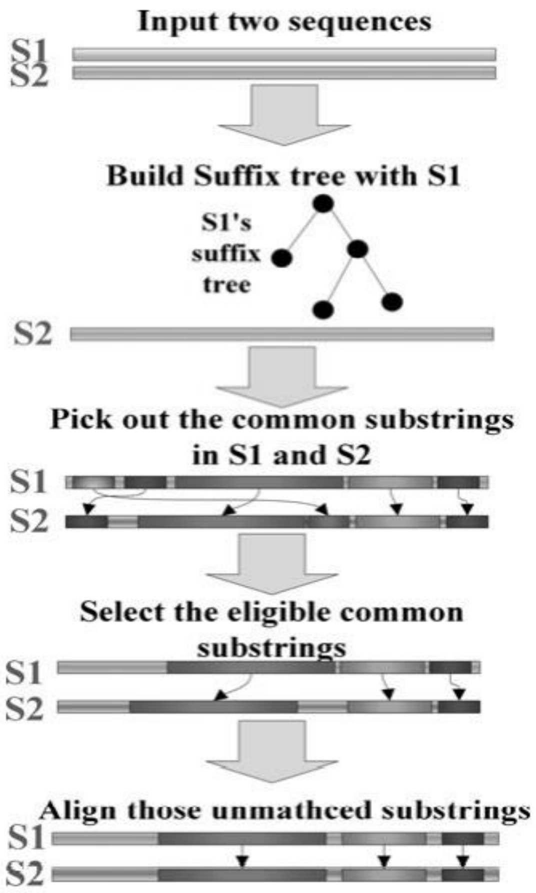
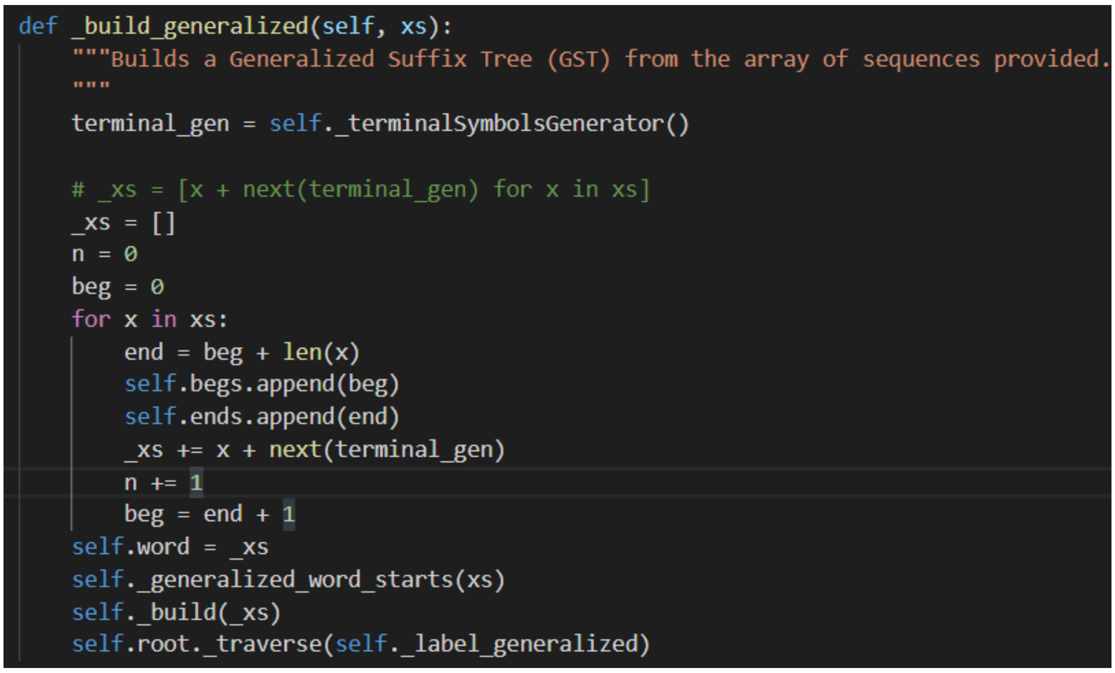

Input two sequences
S1
S2
Build Suffix tree with S1
S1's suffix tree
S2
Pick out the common substrings in S1 and S2
S1
S2
Select the eligible common substrings
S1
S2
Align those unmathced substrings
S1
S2

Figure 1: Alignment

The alignment mainly contain four steps, which are as follows:

1. Build a suffix tree from one sequence, here S11 is assumed as the chosen one, and the tree’ s name is tree-S1;

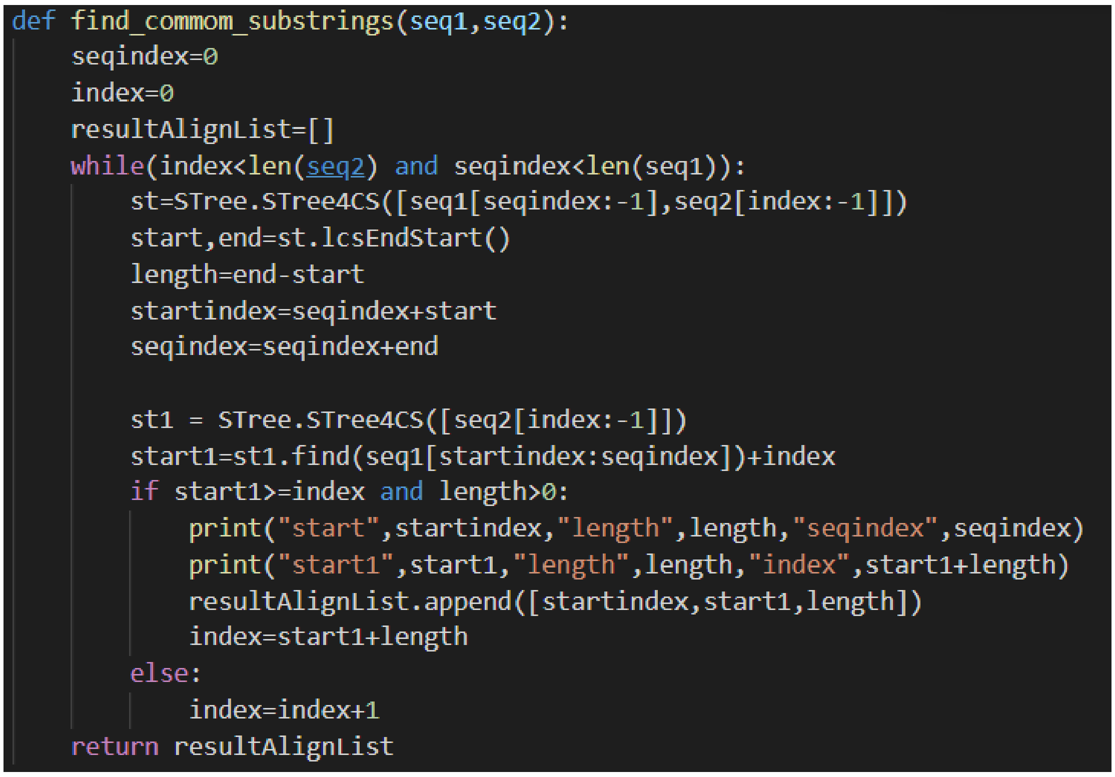


```
def _build_generalized(self, xs):
    """Builds a Generalized Suffix Tree (GST) from the array of sequences provided.
    """
    terminal_gen = self._terminalSymbolsGenerator()

    # _xs = [x + next(terminal_gen) for x in xs]
    _xs = []
    n = 0
    beg = 0
    for x in xs:
        end = beg + len(x)
        self.begs.append(beg)
        self.ends.append(end)
        _xs += x + next(terminal_gen)
        n += 1
        beg = end + 1
    self.word = _xs
    self._generalized_word_starts(xs)
    self._build(_xs)
    self.root._traverse(self._label_generalized)
```

Figure 2: Suffix-Trees Construction Code

2. Pick out common sequences.



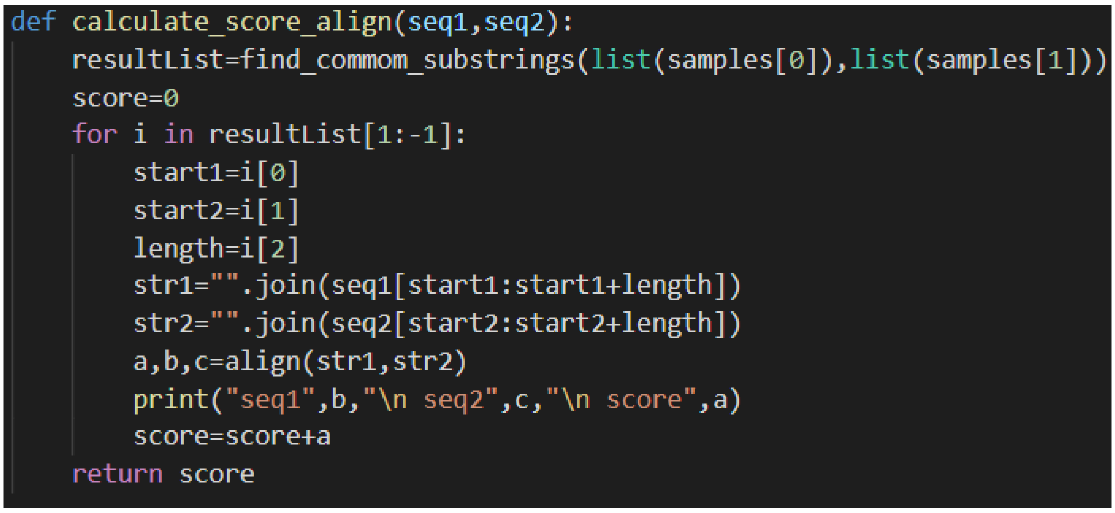
```
def find_commom_substrings(seq1,seq2):
    seqindex=0
    index=0
    resultAlignList=[]
    while(index<len(seq2) and seqindex<len(seq1)):
        st=STree.STree4CS([seq1[seqindex:-1],seq2[index:-1]])
        start,end=st.lcsEndStart()
        length=end-start
        startindex=seqindex+start
        seqindex=seqindex+end

        st1 = STree.STree4CS([seq2[index:-1]])
        start1=st1.find(seq1[startindex:seqindex])+index
        if start1>=index and length>0:
            print("start",startindex,"length",length,"seqindex",seqindex)
            print("start1",start1,"length",length,"index",start1+length)
            resultAlignList.append([startindex,start1,length])
            index=start1+length
        else:
            index=index+1
    return resultAlignList
```

Figure 3: Common Sequence Picking Code

3. Pick out common strings in S1 and S2 with sequence S2 and Tree-S1 to form two common substring sets;

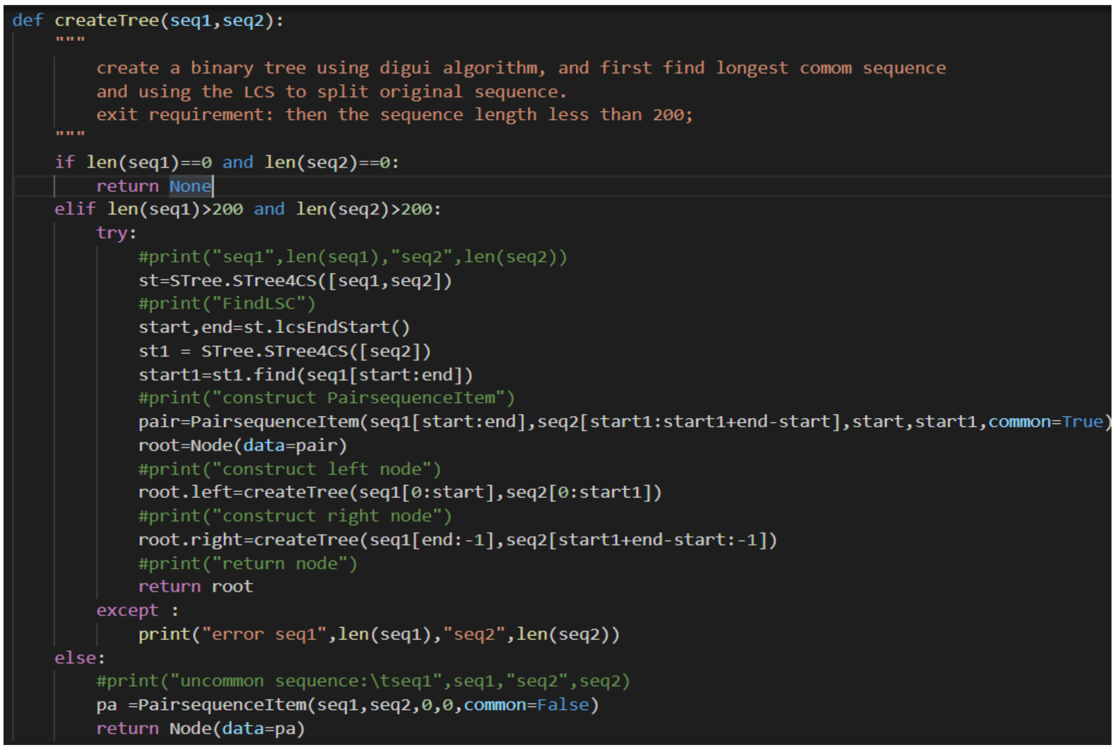
4. Aligns the remaining unmatched substrings between S1&S2 , all substrings are concatenated to form the aligned sequence. And then calculate the score between two sequence. The aligned score id defined by sore(X,Y)=\sum_{i=0}^{n-1}(d(X[i],Y[i]);



```python
def calculate_score_align(seq1,seq2):
    resultList=find_commom_substrings(list(samples[0]),list(samples[1]))
    score=0
    for i in resultList[1:-1]:
        start1=i[0]
        start2=i[1]
        length=i[2]
        str1="".join(seq1[start1:start1+length])
        str2="".join(seq2[start2:start2+length])
        a,b,c=align(str1,str2)
        print("seq1",b,"\n seq2",c,"\n score",a)
        score=score+a
    return score
```

Figure 4: Alignment and Score Calculation

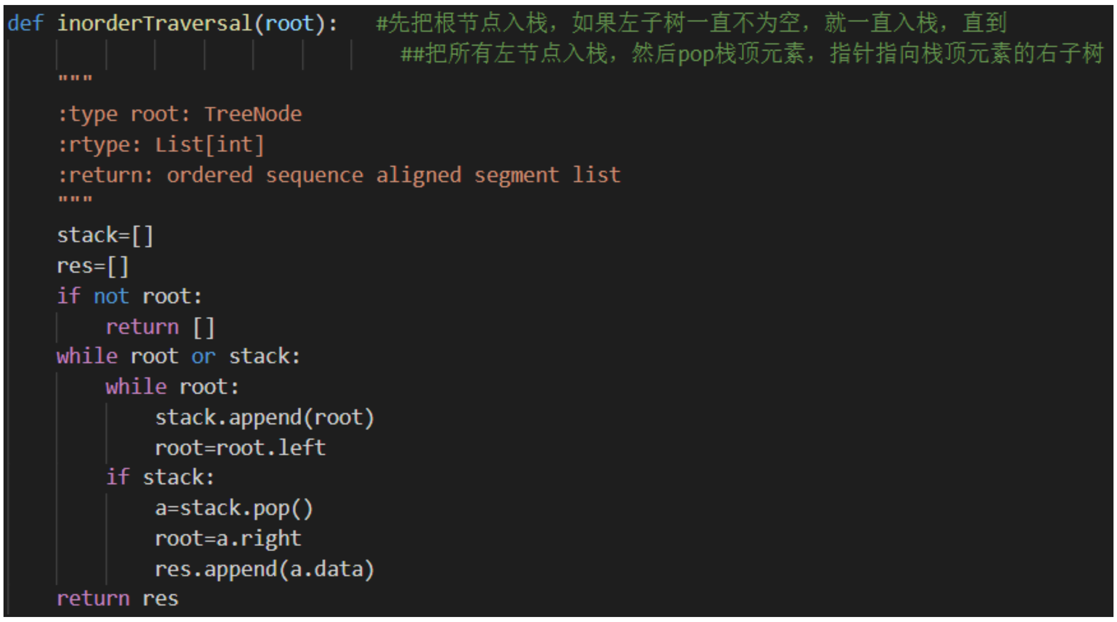
But during experiment, we find the sequence is too long, more than 20000 char, which takes a lot of time to computing, so we improve the algorithm using the binary tree based on the suffix tree. The main idea is that we use suffix tree to find the longest common sequence in the two sequence, and use the sequence aforementioned to split origin sequence to construct binary tree. For the binary tree, each tree node is restore the sub common sequence or two uncommon sequence, for uncommon sequence, we use need Needleman-Wunsch, a kind of dynamic algorithm to calculate the alignment sequence score and get the



```python
def createTree(seq1,seq2):
    """
        create a binary tree using digui algorithm, and first find longest comom sequence
        and using the LCS to split original sequence.
        exit requirement: then the sequence length less than 200;
    """
    if len(seq1)==0 and len(seq2)==0:
        return None
    elif len(seq1)>200 and len(seq2)>200:
        try:
            #print("seq1",len(seq1),"seq2",len(seq2))
            st=STree.STree4CS([seq1,seq2])
            #print("FindLSC")
            start,end=st.lcsEndStart()
            st1 = STree.STree4CS([seq2])
            start1=st1.find(seq1[start:end])
            #print("construct PairsequenceItem")
            pair=PairsequenceItem(seq1[start:end],seq2[start1:start1+end-start],start,start1,common=True)
            root=Node(data=pair)
            #print("construct left node")
            root.left=createTree(seq1[0:start],seq2[0:start1])
            #print("construct right node")
            root.right=createTree(seq1[end:-1],seq2[start1+end-start:-1])
            #print("return node")
            return root
        except :
            print("error seq1",len(seq1),"seq2",len(seq2))
    else:
        #print("uncommon sequence:\tseq1",seq1,"seq2",seq2)
        pa =PairsequenceItem(seq1,seq2,0,0,common=False)
        return Node(data=pa)
```

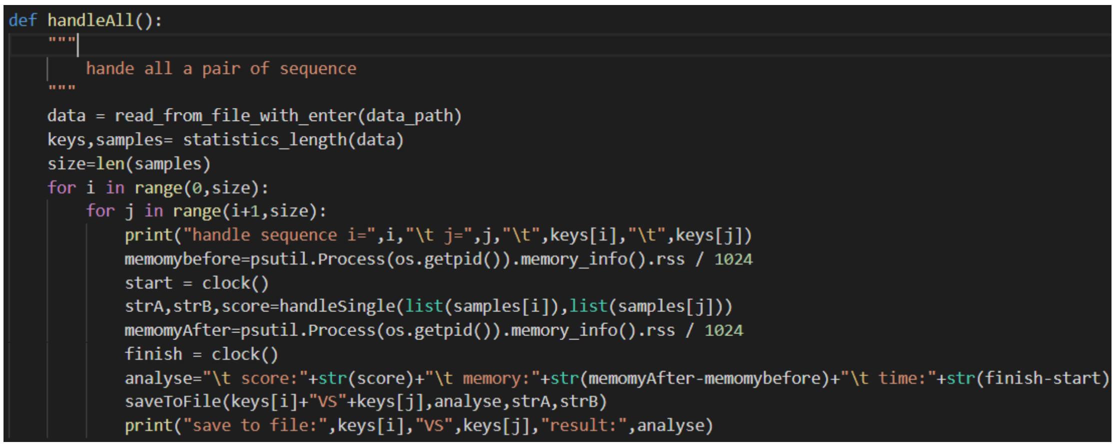
Figure 5: Binary Tree Construction

two alignment sequence. Then we using a middle order algorithm to go over the whole tree to get the alignment sequence and the alignment cost. The proposed methods takes much little time and memory to get the results.



```python
def inorderTraversal(root):   #先把根节点入栈，如果左子树一直不为空，就一直入栈，直到
                                ##把所有左节点入栈，然后pop栈顶元素，指针指向栈顶元素的右子树
    """
    :type root: TreeNode
    :rtype: List[int]
    :return: ordered sequence aligned segment list
    """
    stack=[]
    res=[]
    if not root:
        return []
    while root or stack:
        while root:
            stack.append(root)
            root=root.left
        if stack:
            a=stack.pop()
            root=a.right
            res.append(a.data)
    return res
```

Figure 6: Binary Tree Reverse



```python
def handleAll():
    """
        hande all a pair of sequence
    """
    data = read_from_file_with_enter(data_path)
    keys,samples= statistics_length(data)
    size=len(samples)
    for i in range(0,size):
        for j in range(i+1,size):
            print("handle sequence i=",i,"\t j=",j,"\t",keys[i],"\t",keys[j])
            memomybefore=psutil.Process(os.getpid()).memory_info().rss / 1024
            start = clock()
            strA,strB,score=handleSingle(list(samples[i]),list(samples[j]))
            memomyAfter=psutil.Process(os.getpid()).memory_info().rss / 1024
            finish = clock()
            analyse="\t score:"+str(score)+"\t memory:"+str(memomyAfter-memomybefore)+"\t time:"+str(finish-start)
            saveToFile(keys[i]+"VS"+keys[j],analyse,strA,strB)
            print("save to file:",keys[i],"VS",keys[j],"result:",analyse)
```

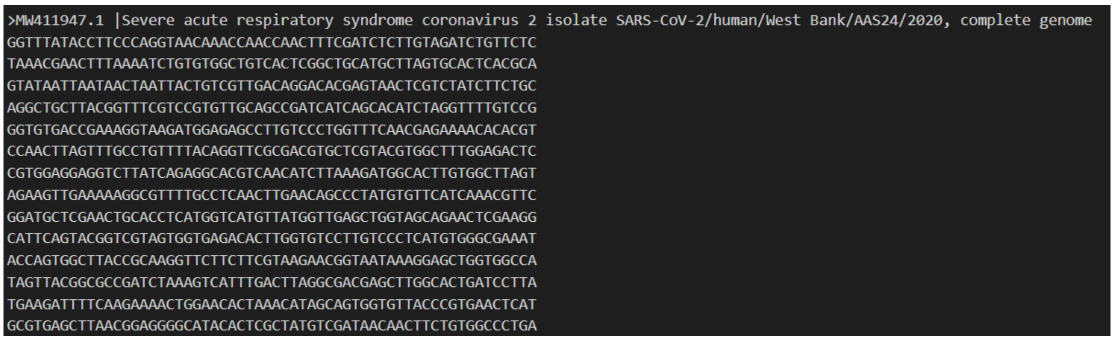
Figure 7: Comparison between each other

Results

[Results]: Here in these section, we will first introduce the experiment environments such as the computer parameters, the data description and the result of the code for 35*34 experiments including spend time, used memory of suffix tree, alignment results and the final alignment, as well as the comparison between BLAST and FASTA algorithm.

Computer settings: processor: Inter(R) Core(TM) i7-9750H CPU @2.60GHz 2.59GHz; memory: 16.0GB; GPU: NVIDIA GeForce GTX 1650。

Data description: the data is provided by teacher, thirty strains of SARS cov-2, which is contained in fasta format file showed in figure 8, and the data about thirty strains of SARS cov-2 distribution is showed in the figure 9, from the figure 9 , the average count of the dna sequence is about 30000.



```
>MW411947.1 |Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/West Bank/AAS24/2020, complete genome
GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTC
TAAACGAACTTTAAAATCTGTGTGGCTGTCACTCGGCTGCATGCTTAGTGCACTCACGCA
GTATAATTAATAACTAATTACTGTCGTTGACAGGACACGAGTAACTCGTCTATCTTCTGC
AGGCTGCTTACGGTTTCGTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTTTGTCCG
GGTGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACACACGT
CCAACTTAGTTTGCCTGTTTTACAGGTTCGCGACGTGCTCGTACGTGGCTTTGGAGACTC
CGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCTTAAAGATGGCACTTGTGGCTTAGT
AGAAGTTGAAAAAGGCGTTTTGCCTCAACTTGAACAGCCCTATGTGTTCATCAAACGTTC
GGATGCTCGAACTGCACCTCATGGTCATGTTATGGTTGAGCTGGTAGCAGAACTCGAAGG
CATTCAGTACGGTCGTAGTGGTGAGACACTTGGTGTCCTTGTCCCTCATGTGGGCGAAAT
ACCAGTGGCTTACCGCAAGGTTCTTCTTCGTAAGAACGGTAATAAAGGAGCTGGTGGCCA
TAGTTACGGCGCCGATCTAAAGTCATTTGACTTAGGCGACGAGCTTGGCACTGATCCTTA
TGAAGATTTTCAAGAAAACTGGAACACTAAACATAGCAGTGGTGTTACCCGTGAACTCAT
GCGTGAGCTTAACGGAGGGGCATACACTCGCTATGTCGATAACAACTTCTGTGGCCCTGA
```

Figure 8: Data Structure

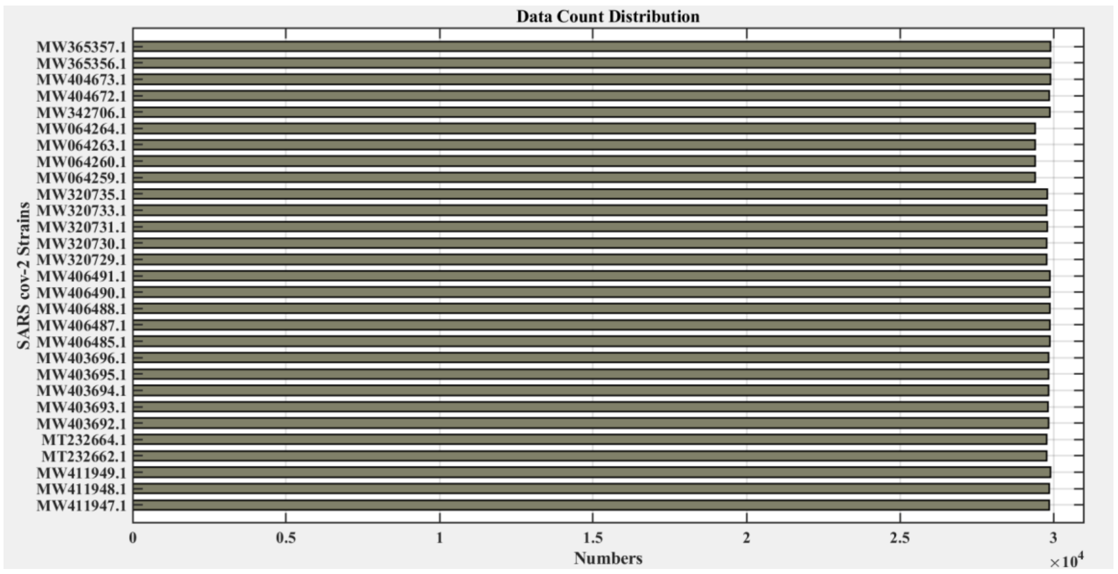
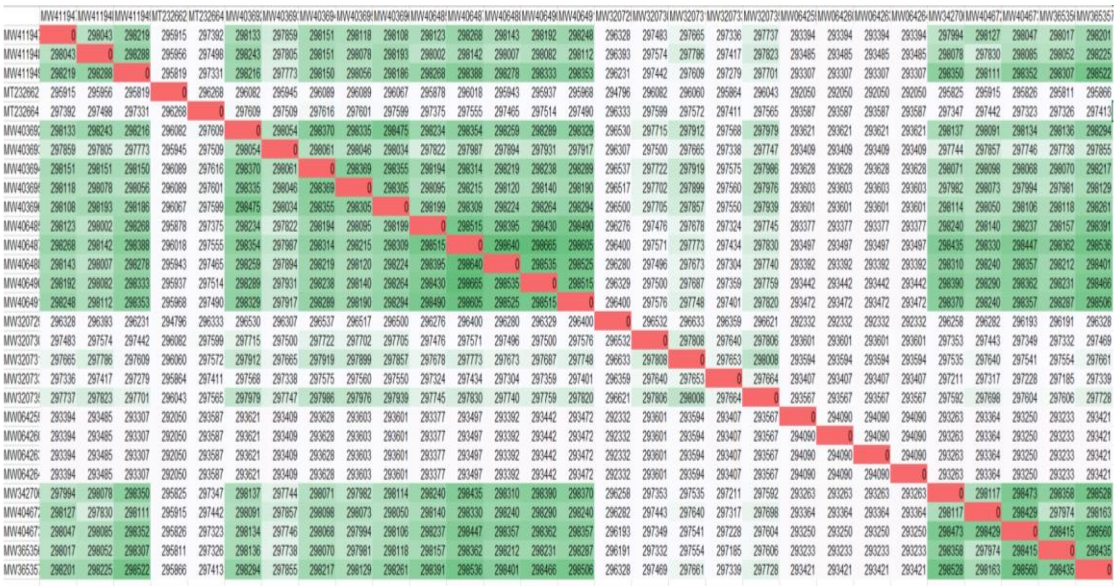


Figure 9: Data Number Distribution

Alignment Score: using score calculation metrics provide in the assignment, sore(X,Y)=\sum_{i=0}^{n-1}(d(X[i],Y[i])); If X[i] and Y[i] both are not gaps, then d(X[i],Y[i]) is defined by the substitution matrix in table 1; If X[i] or Y[i] is a gap, then d(X[i],Y[i])=-4 for opening gap, d(X[i],Y[i])=-1 for extending gap. For each sequence, we are supposed to calculate following 29 sequence, align each pair of them and calculate the aligned sequence score. Here the 30*30 matrix score are showed in the figure 10;

	A	C	G	T
A	10	-5	0	-5
C	-5	10	-5	0
G	0	-5	10	-5
T	-5	0	-5	10

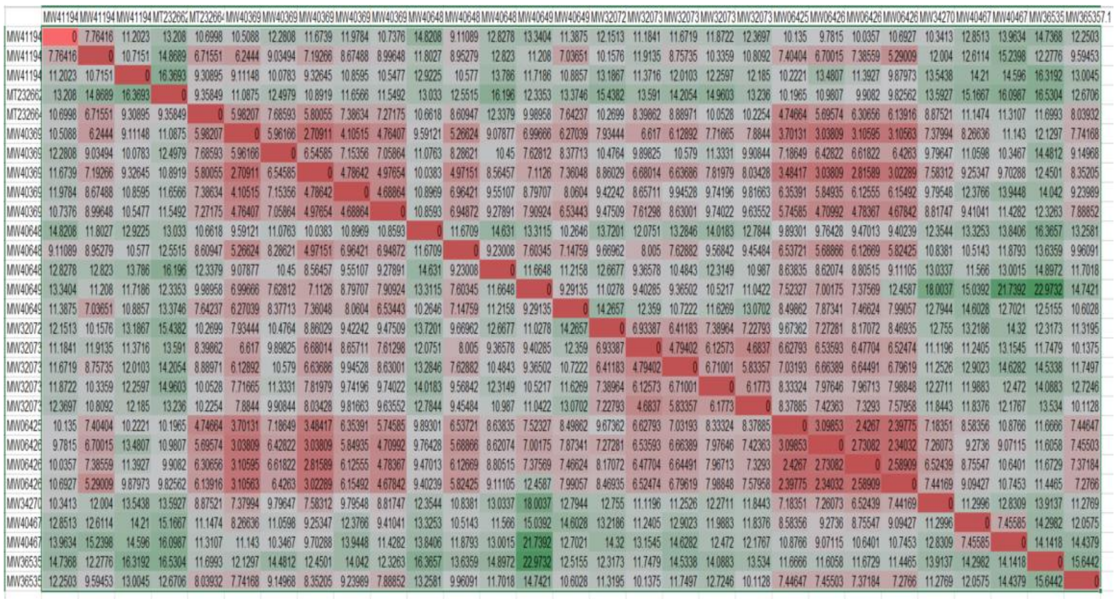
Table 1



	MW41194	MW41194	MW41194	MT232662	MT232664	MW40369	MW40369	MW40369	MW40369	MW40369	MW40648	MW40648	MW40648	MW40649	MW40649	MW32072	MW32073	MW32073	MW32073	MW32073	MW06425	MW06426	MW06426	MW06426	MW34270	MW40467	MW40467	MW36535	MW365357
MW41194	0	298043	298219	295915	297392	298133	297859	298151	298118	298108	298123	298268	298143	298192	298248	296328	297483	297665	297336	297737	293394	293394	293394	293394	297994	298127	298047	298017	298201
MW41194	298043	0	298288	295956	297498	298243	297805	298151	298078	298193	298002	298142	298007	298082	298112	296393	297574	297786	297417	297823	293485	293485	293485	293485	298078	297830	298085	298052	298225
MW41194	298219	298288	0	295819	297331	298216	297773	298150	298056	298186	298268	298388	298278	298333	298353	296231	297442	297609	297279	297701	293307	293307	293307	293307	298350	298111	298352	298307	298522
MT232662	295915	295956	295819	0	296268	296082	295945	296089	296089	296067	295878	296018	295943	295937	295968	294796	296082	296060	295864	296043	292050	292050	292050	292050	295825	295915	295826	295811	295866
MT232664	297392	297498	297331	296268	0	297609	297509	297616	297601	297599	297375	297555	297465	297514	297490	296333	297599	297572	297411	297565	293587	293587	293587	293587	297347	297442	297323	297326	297413
MW40369	298133	298243	298216	296082	297609	0	298054	298370	298335	298475	298234	298354	298259	298289	298329	296530	297715	297912	297568	297979	293621	293621	293621	293621	298137	298091	298134	298136	298294
MW40369	297859	297805	297773	295945	297509	298054	0	298061	298046	298034	297822	297987	297894	297931	297917	296307	297500	297665	297338	297747	293409	293409	293409	293409	297744	297857	297746	297738	297855
MW40369	298151	298151	298150	296089	297616	298370	298061	0	298369	298355	298194	298314	298219	298238	298289	296537	297722	297919	297575	297986	293628	293628	293628	293628	298071	298098	298068	298070	298217
MW40369	298118	298078	298056	296089	297601	298335	298046	298369	0	298305	298095	298215	298120	298140	298190	296517	297702	297899	297560	297976	293603	293603	293603	293603	297982	298073	297994	297981	298129
MW40369	298108	298193	298186	296067	297599	298475	298034	298355	298305	0	298199	298309	298224	298264	298294	296500	297705	297857	297550	297939	293601	293601	293601	293601	298114	298050	298106	298118	298261
MW40648	298123	298002	298268	295878	297375	298234	297822	298194	298095	298199	0	298515	298395	298430	298490	296276	297476	297678	297324	297745	293377	293377	293377	293377	298240	298140	298237	298157	298391
MW40648	298268	298142	298388	296018	297555	298354	297987	298314	298215	298309	298515	0	298640	298665	298605	296400	297571	297773	297434	297830	293497	293497	293497	293497	298435	298330	298447	298362	298536
MW40648	298143	298007	298278	295943	297465	298259	297894	298219	298120	298224	298395	298640	0	298535	298525	296280	297496	297673	297304	297740	293392	293392	293392	293392	298310	298240	298357	298212	298401
MW40649	298192	298082	298333	295937	297514	298289	297931	298238	298140	298264	298430	298665	298535	0	298515	296329	297500	297687	297359	297759	293442	293442	293442	293442	298390	298290	298362	298231	298466
MW40649	298248	298112	298353	295968	297490	298329	297917	298289	298190	298294	298490	298605	298525	298515	0	296400	297576	297748	297401	297820	293472	293472	293472	293472	298370	298240	298357	298287	298506
MW32072	296328	296393	296231	294796	296333	296530	296307	296537	296517	296500	296276	296400	296280	296329	296400	0	296532	296633	296359	296621	292332	292332	292332	292332	296258	296282	296193	296191	296328
MW32073	297483	297574	297442	296082	297599	297715	297500	297722	297702	297705	297476	297571	297496	297500	297576	296532	0	297808	297640	297806	293601	293601	293601	293601	297353	297443	297349	297332	297469
MW32073	297665	297786	297609	296060	297572	297912	297665	297919	297899	297857	297678	297773	297673	297687	297748	296633	297808	0	297653	298008	293594	293594	293594	293594	297535	297640	297541	297554	297661
MW32073	297336	297417	297279	295864	297411	297568	297338	297575	297560	297550	297324	297434	297304	297359	297401	296359	297640	297653	0	297664	293407	293407	293407	293407	297211	297317	297228	297185	297339
MW32073	297737	297823	297701	296043	297565	297979	297747	297986	297976	297939	297745	297830	297740	297759	297820	296621	297806	298008	297664	0	293567	293567	293567	293567	297592	297698	297604	297606	297728
MW06425	293394	293485	293307	292050	293587	293621	293409	293628	293603	293601	293377	293497	293392	293442	293472	292332	293601	293594	293407	293567	0	294090	294090	294090	293263	293364	293250	293233	293421
MW06426	293394	293485	293307	292050	293587	293621	293409	293628	293603	293601	293377	293497	293392	293442	293472	292332	293601	293594	293407	293567	294090	0	294090	294090	293263	293364	293250	293233	293421
MW06426	293394	293485	293307	292050	293587	293621	293409	293628	293603	293601	293377	293497	293392	293442	293472	292332	293601	293594	293407	293567	294090	294090	0	294090	293263	293364	293250	293233	293421
MW06426	293394	293485	293307	292050	293587	293621	293409	293628	293603	293601	293377	293497	293392	293442	293472	292332	293601	293594	293407	293567	294090	294090	294090	0	293263	293364	293250	293233	293421
MW34270	297994	298078	298350	295825	297347	298137	297744	298071	297982	298114	298240	298435	298310	298390	298370	296258	297353	297535	297211	297592	293263	293263	293263	293263	0	298117	298473	298358	298528
MW40467	298127	297830	298111	295915	297442	298091	297857	298098	298073	298050	298140	298330	298240	298290	298240	296282	297443	297640	297317	297698	293364	293364	293364	293364	298117	0	298429	297974	298163
MW40467	298047	298085	298352	295826	297323	298134	297746	298068	297994	298106	298237	298447	298357	298362	298357	296193	297349	297541	297228	297604	293250	293250	293250	293250	298473	298429	0	298415	298560
MW36535	298017	298052	298307	295811	297326	298136	297738	298070	297981	298118	298157	298362	298212	298231	298287	296191	297332	297554	297185	297606	293233	293233	293233	293233	298358	297974	298415	0	298435
MW36535	298201	298225	298522	295866	297413	298294	297855	298217	298129	298261	298391	298536	298401	298466	298506	296328	297469	297661	297339	297728	293421	293421	293421	293421	298528	298163	298560	298435	0

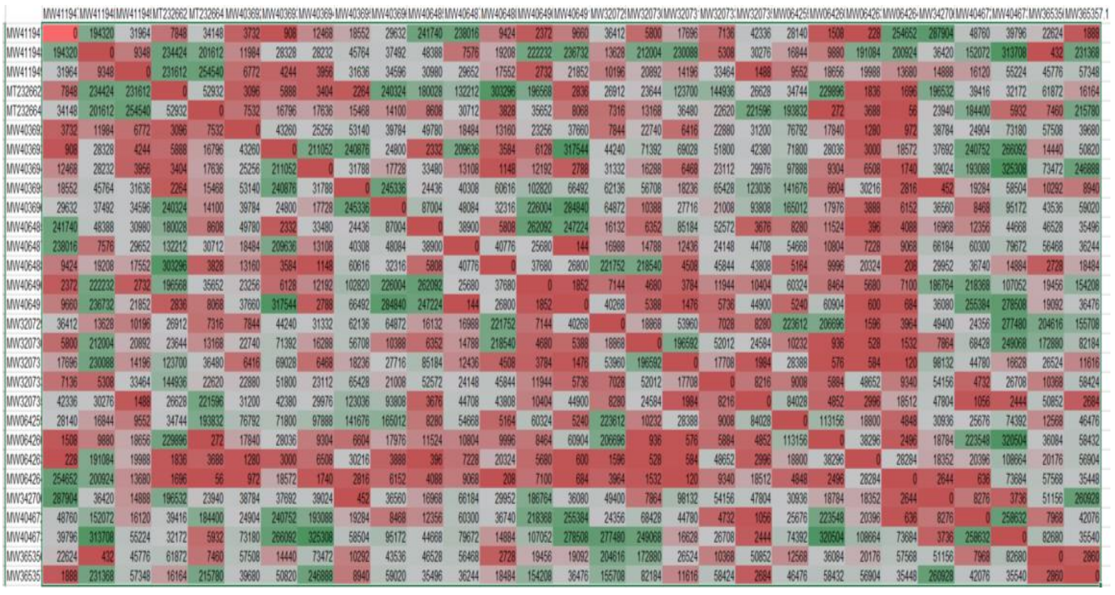
Figure 10: Aligned Score Matrix

Process Monitor: for each aligned score matrix calculation progress, we use clock() function and psutil module provided by python api to get the information about the code runtime conditions. And the time matrix is showed in the figure 11, and the memory matrix is showed in the figure 12. And the aligned sequence is partial showed in the figure 13, due to the size.



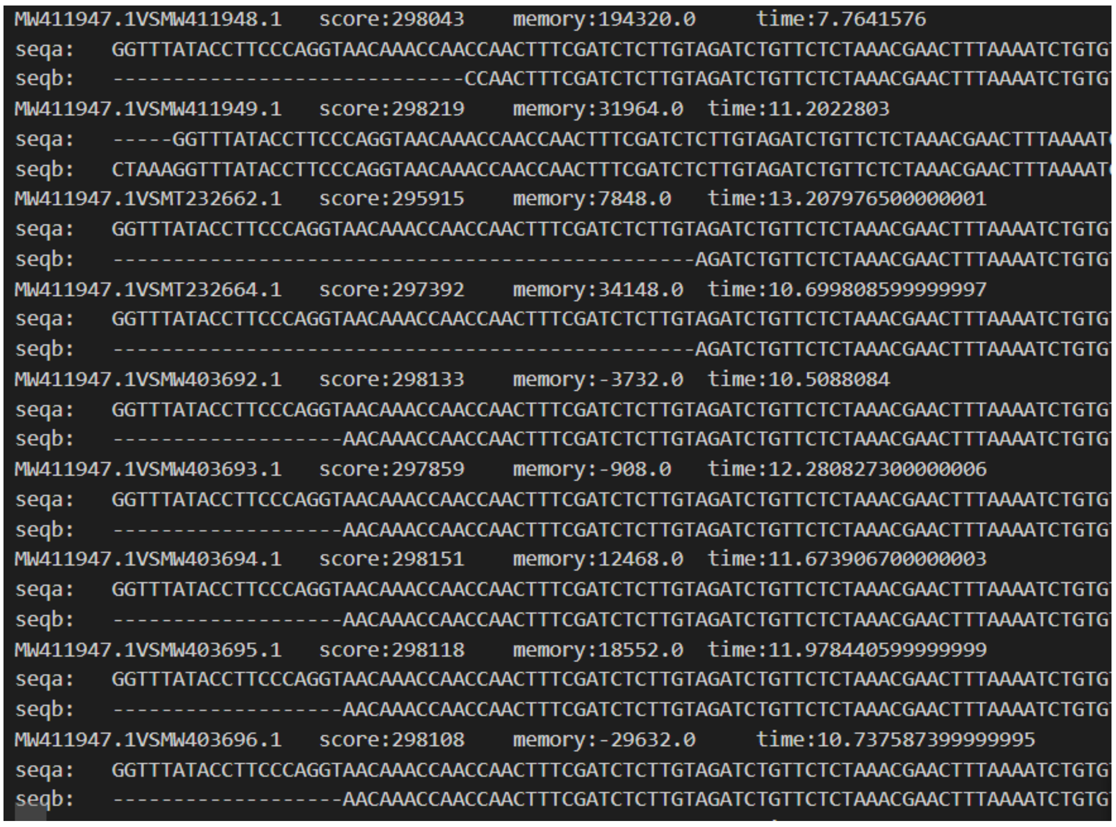
	MW41194	MW41194	MW41194	MT23266	MT23266	MW40369	MW40369	MW40369	MW40369	MW40369	MW40648	MW40648	MW40648	MW40649	MW40649	MW32072	MW32073	MW32073	MW32073	MW32073	MW06425	MW06426	MW06426	MW06426	MW34270	MW40467	MW40467	MW36535	MW365357.1
MW41194	0	7.76416	11.2023	13.208	10.6998	10.5088	12.2808	11.6739	11.9784	10.7376	14.8208	9.11089	12.8278	13.3404	11.3875	12.1513	11.1841	11.6719	11.8722	12.3697	10.135	9.7815	10.0357	10.6927	10.3413	12.8513	13.9634	14.7368	12.2503
MW41194	7.76416	0	10.7151	14.8689	6.71551	6.2444	9.03494	7.19266	8.67488	8.99648	11.8027	8.95279	12.823	11.208	7.03651	10.1576	11.9135	8.75735	10.3359	10.8092	7.40404	6.70015	7.38559	5.29009	12.004	12.6114	15.2398	12.2776	9.59453
MW41194	11.2023	10.7151	0	16.3693	9.30895	9.11148	10.0783	9.32645	10.8595	10.5477	12.9225	10.577	13.786	11.7186	10.8857	13.1867	11.3716	12.0103	12.2597	12.185	10.2221	13.4807	11.3927	9.87973	13.5438	14.21	14.596	16.3192	13.0045
MT23266	13.208	14.8689	16.3693	0	9.35849	11.0875	12.4979	10.8919	11.6566	11.5492	13.033	12.5515	16.196	12.3353	13.3746	15.4382	13.591	14.2054	14.9603	13.236	10.1965	10.9807	9.9082	9.82562	13.5927	15.1667	16.0987	16.5304	12.6706
MT23266	10.6998	6.71551	9.30895	9.35849	0	5.98207	7.68593	5.80055	7.38634	7.27175	10.6618	8.60947	12.3379	9.98958	7.64237	10.2699	8.39862	8.88971	10.0528	10.2254	4.74664	5.69574	6.30656	6.13916	8.87521	11.1474	11.3107	11.6993	8.03932
MW40369	10.5088	6.2444	9.11148	11.0875	5.98207	0	5.96166	2.70911	4.10515	4.76407	9.59121	5.26624	9.07877	6.99666	6.27039	7.93444	6.617	6.12892	7.71665	7.8844	3.70131	3.03809	3.10595	3.10563	7.37994	8.26636	11.143	12.1297	7.74168
MW40369	12.2808	9.03494	10.0783	12.4979	7.68593	5.96166	0	6.54585	7.15356	7.05864	11.0763	8.28621	10.45	7.62812	8.37713	10.4764	9.89825	10.579	11.3331	9.90844	7.18649	6.42822	6.61822	6.4263	9.79647	11.0598	10.3467	14.4812	9.14968
MW40369	11.6739	7.19266	9.32645	10.8919	5.80055	2.70911	6.54585	0	4.78642	4.97654	10.0383	4.97151	8.56457	7.1126	7.36048	8.86029	6.68014	6.63686	7.81979	8.03428	3.48417	3.03809	2.81589	3.02289	7.58312	9.25347	9.70288	12.4501	8.35205
MW40369	11.9784	8.67488	10.8595	11.6566	7.38634	4.10515	7.15356	4.78642	0	4.68864	10.8969	6.96421	9.55107	8.79707	8.0604	9.42242	8.65711	9.94528	9.74196	9.81663	6.35391	5.84935	6.12555	6.15492	9.79548	12.3766	13.9448	14.042	9.23989
MW40369	10.7376	8.99648	10.5477	11.5492	7.27175	4.76407	7.05864	4.97654	4.68864	0	10.8593	6.94872	9.27891	7.90924	6.53443	9.47509	7.61298	8.63001	9.74022	9.63552	5.74585	4.70992	4.78367	4.67842	8.81747	9.41041	11.4282	12.3263	7.88852
MW40648	14.8208	11.8027	12.9225	13.033	10.6618	9.59121	11.0763	10.0383	10.8969	10.8593	0	11.6709	14.631	13.3115	10.2646	13.7201	12.0751	13.2846	14.0183	12.7844	9.89301	9.76428	9.47013	9.40239	12.3544	13.3253	13.8406	16.3657	13.2581
MW40648	9.11089	8.95279	10.577	12.5515	8.60947	5.26624	8.28621	4.97151	6.96421	6.94872	11.6709	0	9.23008	7.60345	7.14759	9.66962	8.005	7.62882	9.56842	9.45484	6.53721	5.68866	6.12669	5.82425	10.8381	10.5143	11.8793	13.6359	9.96091
MW40648	12.8278	12.823	13.786	16.196	12.3379	9.07877	10.45	8.56457	9.55107	9.27891	14.631	9.23008	0	11.6648	11.2158	12.6677	9.36578	10.4843	12.3149	10.987	8.63835	8.62074	8.80515	9.11105	13.0337	11.566	13.0015	14.8972	11.7018
MW40649	13.3404	11.208	11.7186	12.3353	9.98958	6.99666	7.62812	7.1126	8.79707	7.90924	13.3115	7.60345	11.6648	0	9.29135	11.0278	9.40285	9.36502	10.5217	11.0422	7.52327	7.00175	7.37569	12.4587	18.0037	15.0392	21.7392	22.9732	14.7421
MW40649	11.3875	7.03651	10.8857	13.3746	7.64237	6.27039	8.37713	7.36048	8.0604	6.53443	10.2646	7.14759	11.2158	9.29135	0	14.2657	12.359	10.7222	11.6269	13.0702	8.49862	7.87341	7.46624	7.99057	12.7944	14.6028	12.7021	12.5155	10.6028
MW32072	12.1513	10.1576	13.1867	15.4382	10.2699	7.93444	10.4764	8.86029	9.42242	9.47509	13.7201	9.66962	12.6677	11.0278	14.2657	0	6.93387	6.41183	7.38964	7.22793	9.67362	7.27281	8.17072	8.46935	12.755	13.2186	14.32	12.3173	11.3195
MW32073	11.1841	11.9135	11.3716	13.591	8.39862	6.617	9.89825	6.68014	8.65711	7.61298	12.0751	8.005	9.36578	9.40285	12.359	6.93387	0	4.79402	6.12573	4.6837	6.62793	6.53593	6.47704	6.52474	11.1196	11.2405	13.1545	11.7479	10.1375
MW32073	11.6719	8.75735	12.0103	14.2054	8.88971	6.12892	10.579	6.63686	9.94528	8.63001	13.2846	7.62882	10.4843	9.36502	10.7222	6.41183	4.79402	0	6.71001	5.83357	7.03193	6.66389	6.64491	6.79619	11.2526	12.9023	14.6282	14.5338	11.7497
MW32073	11.8722	10.3359	12.2597	14.9603	10.0528	7.71665	11.3331	7.81979	9.74196	9.74022	14.0183	9.56842	12.3149	10.5217	11.6269	7.38964	6.12573	6.71001	0	6.1773	8.33324	7.97646	7.96713	7.98848	12.2711	11.9883	12.472	14.0883	12.7246
MW32073	12.3697	10.8092	12.185	13.236	10.2254	7.8844	9.90844	8.03428	9.81663	9.63552	12.7844	9.45484	10.987	11.0422	13.0702	7.22793	4.6837	5.83357	6.1773	0	8.37885	7.42363	7.3293	7.57958	11.8443	11.8376	12.1767	13.534	10.1128
MW06425	10.135	7.40404	10.2221	10.1965	4.74664	3.70131	7.18649	3.48417	6.35391	5.74585	9.89301	6.53721	8.63835	7.52327	8.49862	9.67362	6.62793	7.03193	8.33324	8.37885	0	3.09853	2.4267	2.39775	7.18351	8.58356	10.8766	11.6666	7.44647
MW06426	9.7815	6.70015	13.4807	10.9807	5.69574	3.03809	6.42822	3.03809	5.84935	4.70992	9.76428	5.68866	8.62074	7.00175	7.87341	7.27281	6.53593	6.66389	7.97646	7.42363	3.09853	0	2.73082	2.34032	7.26073	9.2736	9.07115	11.6058	7.45503
MW06426	10.0357	7.38559	11.3927	9.9082	6.30656	3.10595	6.61822	2.81589	6.12555	4.78367	9.47013	6.12669	8.80515	7.37569	7.46624	8.17072	6.47704	6.64491	7.96713	7.3293	2.4267	2.73082	0	2.58909	6.52439	8.75547	10.6401	11.6729	7.37184
MW06426	10.6927	5.29009	9.87973	9.82562	6.13916	3.10563	6.4263	3.02289	6.15492	4.67842	9.40239	5.82425	9.11105	12.4587	7.99057	8.46935	6.52474	6.79619	7.98848	7.57958	2.39775	2.34032	2.58909	0	7.44169	9.09427	10.7453	11.4465	7.2766
MW34270	10.3413	12.004	13.5438	13.5927	8.87521	7.37994	9.79647	7.58312	9.79548	8.81747	12.3544	10.8381	13.0337	18.0037	12.7944	12.755	11.1196	11.2526	12.2711	11.8443	7.18351	7.26073	6.52439	7.44169	0	11.2996	12.8309	13.9137	11.2769
MW40467	12.8513	12.6114	14.21	15.1667	11.1474	8.26636	11.0598	9.25347	12.3766	9.41041	13.3253	10.5143	11.566	15.0392	14.6028	13.2186	11.2405	12.9023	11.9883	11.8376	8.58356	9.2736	8.75547	9.09427	11.2996	0	7.45585	14.2982	12.0575
MW40467	13.9634	15.2398	14.596	16.0987	11.3107	11.143	10.3467	9.70288	13.9448	11.4282	13.8406	11.8793	13.0015	21.7392	12.7021	14.32	13.1545	14.6282	12.472	12.1767	10.8766	9.07115	10.6401	10.7453	12.8309	7.45585	0	14.1418	14.4379
MW36535	14.7368	12.2776	16.3192	16.5304	11.6993	12.1297	14.4812	12.4501	14.042	12.3263	16.3657	13.6359	14.8972	22.9732	12.5155	12.3173	11.7479	14.5338	14.0883	13.534	11.6666	11.6058	11.6729	11.4465	13.9137	14.2982	14.1418	0	15.6442
MW36535	12.2503	9.59453	13.0045	12.6706	8.03932	7.74168	9.14968	8.35205	9.23989	7.88852	13.2581	9.96091	11.7018	14.7421	10.6028	11.3195	10.1375	11.7497	12.7246	10.1128	7.44647	7.45503	7.37184	7.2766	11.2769	12.0575	14.4379	15.6442	0

Figure 11: Time Matrix



	MW41194	MW41194	MW41194	MT232662	MT232664	MW40369	MW40369	MW40369	MW40369	MW40369	MW40648	MW40648	MW40648	MW40649	MW40649	MW32072	MW32073	MW32073	MW32073	MW32073	MW06425	MW06426	MW06426	MW06426	MW34270	MW40467	MW40467	MW36535	MW365357.1
MW41194	0	194320	31964	7848	34148	3732	908	12468	18552	29632	241740	238016	9424	2372	9660	36412	5800	17696	7136	42336	28140	1508	228	254652	287904	48760	39796	22624	1888
MW41194	194320	0	9348	234424	201612	11984	28328	28232	45764	37492	48388	7576	19208	222232	236732	13628	212004	230088	5308	30276	16844	9880	191084	200924	36420	152072	313708	432	231368
MW41194	31964	9348	0	231612	254540	6772	4244	3956	31636	34596	30980	29652	17552	2732	21852	10196	20892	14196	33464	1488	9552	18656	19988	13680	14888	16120	55224	45776	57348
MT232662	7848	234424	231612	0	52932	3096	5888	3404	2264	240324	180028	132212	303296	196568	2836	26912	23644	123700	144936	26628	34744	229896	1836	1696	196532	39416	32172	61872	16164
MT232664	34148	201612	254540	52932	0	7532	16796	17636	15468	14100	8608	30712	3828	35652	8068	7316	13168	36480	22620	221596	193832	272	3688	56	23940	184400	5932	7460	215780
MW40369	3732	11984	6772	3096	7532	0	43260	25256	53140	39784	49780	18484	13160	23256	37660	7844	22740	6416	22880	31200	76792	17840	1280	972	38784	24904	73180	57508	39680
MW40369	908	28328	4244	5888	16796	43260	0	211052	240876	24800	2332	209636	3584	6128	317544	44240	71392	69028	51800	42380	71800	28036	3000	18572	37692	240752	266092	14440	50820
MW40369	12468	28232	3956	3404	17636	25256	211052	0	31788	17728	33480	13108	1148	12192	2788	31332	16288	6468	23112	29976	97888	9304	6508	1740	39024	193088	325308	73472	246888
MW40369	18552	45764	31636	2264	15468	53140	240876	31788	0	245336	24436	40308	60616	102820	66492	62136	56708	18236	65428	123036	141676	6604	30216	2816	452	19284	58504	10292	8940
MW40369	29632	37492	34596	240324	14100	39784	24800	17728	245336	0	87004	48084	32316	226004	284840	64872	10388	27716	21008	93808	165012	17976	3888	6152	36560	8468	95172	43536	59020
MW40648	241740	48388	30980	180028	8608	49780	2332	33480	24436	87004	0	38900	5808	262092	247224	16132	6352	85184	52572	3676	8280	11524	396	4088	16968	12356	44668	46528	35496
MW40648	238016	7576	29652	132212	30712	18484	209636	13108	40308	48084	38900	0	40776	25680	144	16988	14788	12436	24148	44708	54668	10804	7228	9068	66184	60300	79672	56468	36244
MW40648	9424	19208	17552	303296	3828	13160	3584	1148	60616	32316	5808	40776	0	37680	26800	221752	218540	4508	45844	43808	5164	9996	20324	208	29952	36740	14884	2728	18484
MW40649	2372	222232	2732	196568	35652	23256	6128	12192	102820	226004	262092	25680	37680	0	1852	7144	4680	3784	11944	10404	60324	8464	5680	7100	186764	218368	107052	19456	154208
MW40649	9660	236732	21852	2836	8068	37660	317544	2788	66492	284840	247224	144	26800	1852	0	40268	5388	1476	5736	44900	5240	60904	600	684	36080	255384	278508	19092	36476
MW32072	36412	13628	10196	26912	7316	7844	44240	31332	62136	64872	16132	16988	221752	7144	40268	0	18868	53960	7028	8280	223612	206696	1596	3964	49400	24356	277480	204616	155708
MW32073	5800	212004	20892	23644	13168	22740	71392	16288	56708	10388	6352	14788	218540	4680	5388	18868	0	196592	52012	24584	10232	936	528	1532	7864	68428	249068	172880	82184
MW32073	17696	230088	14196	123700	36480	6416	69028	6468	18236	27716	85184	12436	4508	3784	1476	53960	196592	0	17708	1984	28388	576	584	120	98132	44780	16628	26524	11616
MW32073	7136	5308	33464	144936	22620	22880	51800	23112	65428	21008	52572	24148	45844	11944	5736	7028	52012	17708	0	8216	9008	5884	48652	9340	54156	4732	26708	10368	58424
MW32073	42336	30276	1488	26628	221596	31200	42380	29976	123036	93808	3676	44708	43808	10404	44900	8280	24584	1984	8216	0	84028	4852	2996	18512	47804	1056	2444	50852	2684
MW06425	28140	16844	9552	34744	193832	76792	71800	97888	141676	165012	8280	54668	5164	60324	5240	223612	10232	28388	9008	84028	0	113156	18800	4848	30936	25676	74392	12568	46476
MW06426	1508	9880	18656	229896	272	17840	28036	9304	6604	17976	11524	10804	9996	8464	60904	206696	936	576	5884	4852	113156	0	38296	2496	18784	223548	320504	36084	58432
MW06426	228	191084	19988	1836	3688	1280	3000	6508	30216	3888	396	7228	20324	5680	600	1596	528	584	48652	2996	18800	38296	0	28284	18352	20396	108664	20176	56904
MW06426	254652	200924	13680	1696	56	972	18572	1740	2816	6152	4088	9068	208	7100	684	3964	1532	120	9340	18512	4848	2496	28284	0	2644	636	73684	57568	35448
MW34270	287904	36420	14888	196532	23940	38784	37692	39024	452	36560	16968	66184	29952	186764	36080	49400	7864	98132	54156	47804	30936	18784	18352	2644	0	8276	3736	51156	260928
MW40467	48760	152072	16120	39416	184400	24904	240752	193088	19284	8468	12356	60300	36740	218368	255384	24356	68428	44780	4732	1056	25676	223548	20396	636	8276	0	258632	7968	42076
MW40467	39796	313708	55224	32172	5932	73180	266092	325308	58504	95172	44668	79672	14884	107052	278508	277480	249068	16628	26708	2444	74392	320504	108664	73684	3736	258632	0	82680	35540
MW36535	22624	432	45776	61872	7460	57508	14440	73472	10292	43536	46528	56468	2728	19456	19092	204616	172880	26524	10368	50852	12568	36084	20176	57568	51156	7968	82680	0	2860
MW36535	1888	231368	57348	16164	215780	39680	50820	246888	8940	59020	35496	36244	18484	154208	36476	155708	82184	11616	58424	2684	46476	58432	56904	35448	260928	42076	35540	2860	0

Figure 12: Memory Matrix



```
MW411947.1VSMW411948.1    score:298043    memory:194320.0     time:7.7641576
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -----------------------------CCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW411949.1    score:298219    memory:31964.0  time:11.2022803
seqa:   -----GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAAT
seqb:   CTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAAT
MW411947.1VSMT232662.1    score:295915    memory:7848.0   time:13.207976500000001
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   ------------------------------------------------AGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMT232664.1    score:297392    memory:34148.0  time:10.699808599999997
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   ------------------------------------------------AGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW403692.1    score:298133    memory:-3732.0  time:10.5088084
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -------------------AACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW403693.1    score:297859    memory:-908.0   time:12.280827300000006
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -------------------AACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW403694.1    score:298151    memory:12468.0  time:11.673906700000003
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -------------------AACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW403695.1    score:298118    memory:18552.0  time:11.978440599999999
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -------------------AACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
MW411947.1VSMW403696.1    score:298108    memory:-29632.0     time:10.737587399999995
seqa:   GGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
seqb:   -------------------AACAAACCAACCAACTTTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTG
```

Figure 13: Aligned Sequence

Comparison with BLAST and FASTA algorithm: BLASTN and BLASTP use a rolling window to break down a query sequence into words and word synonyms that form a search set. At least two words or synonyms in the search set must match a target sequence in the database, for that sequence to be reported in the results. The BLAST algorithm is showed in the figure 14, and the Fasta algorithm mainly has five steps, which is showed as follows:

1. Identify common k-words between I and J;
2. Score diagonals with k-word matches, identify 10 best diagonals;
3. Rescore initial regions with a substitution score matrix;
4. Join initial regions using gaps, penalize for gaps;
5. Perform gynamic programming to find final alignments.

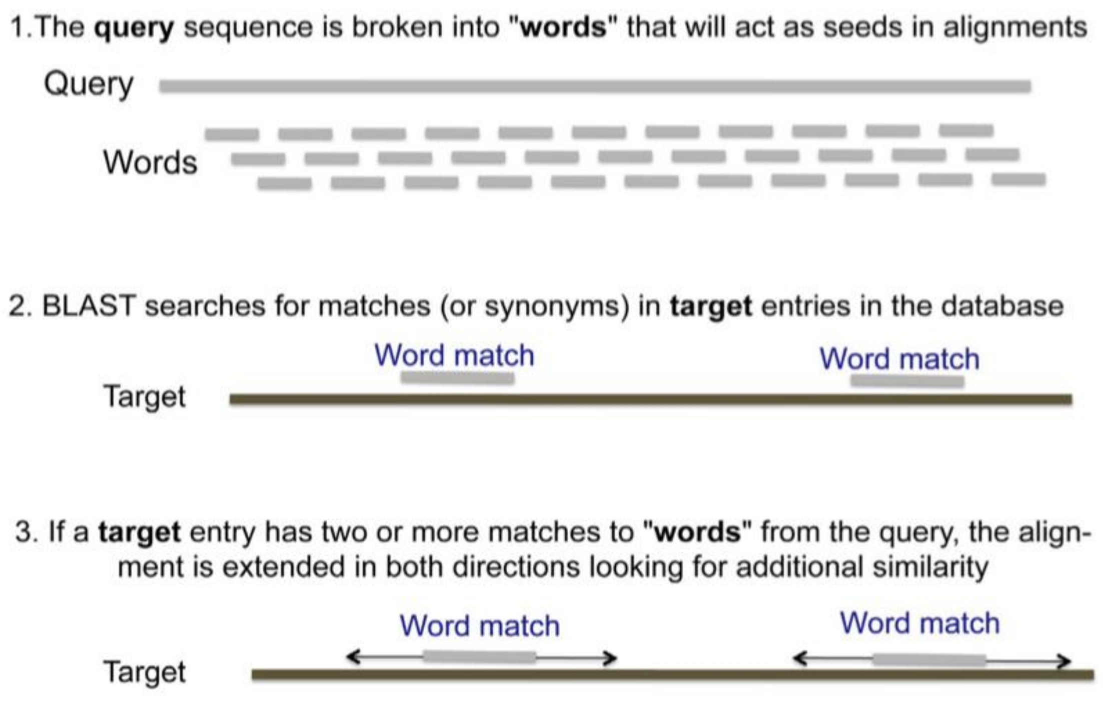

1.The query sequence is broken into "words" that will act as seeds in alignments
Query
Words
2. BLAST searches for matches (or synonyms) in target entries in the database
Word match
Word match
Target
3. If a target entry has two or more matches to "words" from the query, the alignment is extended in both directions looking for additional similarity
Word match
Word match
Target

Figure 14: Blast Algorithm workflow

Conclusion

[Conclusion]：Based on the deep study of the classical suffix tree, we present Binary-Tree based on a suffix tree index structure. Binary-Tree restores all short biology sequences, which divided by Longest common sequence which is calculated using suffix-tree. The Binary-Tree based algorithm solve the “memory bottleneck” that the classical suffix tree exists, and then in order to construct a suffix tree, we sort the short biology sequences by alphabetical order and according to the biggest public prefix of the two adjacent biology sequences confirm the branch point. During constructing the suffix tree, and align all the sequence in one time, through experiment, we can see that the length of sequence data is two long, that we can’t align sequence in one time, considering computer requirements, such as the memory, and computing time. So we use suffix tree to find the longest common sequence and use the longest common sequence find aforementioned to split original sequence and construct the binary tree. After constructing the binary tree, we using the middle order algorithm to go over the whole tree to get the alignment sequence and the alignment cost. And for each node, we use Needleman-Wunsch algorithm to align partial sequence and calculate the score using dynamic algorithm.

During solve the task the teacher assigned, I lean something about DNA sequence matching through the internet and am very amazed by the beautiful data structure, which is of so much power to solve the problem in reality. By solving the question, I also improve my coding ability as well as algorithm understanding and constructing ability. Thanks for sincerely teaching.

References

[1] McCreight, Edward M. "A space-economical suffix tree construction algorithm." Journal of the ACM (JACM) 23, no. 2 (1976): 262-272.
[2] Bergroth, Lasse, Harri Hakonen, and Timo Raita. "A survey of longest common subsequence algorithms." In Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000, pp. 39-48. IEEE, 2000.
[3] Likic, Vladimir. "The Needleman-Wunsch algorithm for sequence alignment." Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne (2008): 1-46.
[4] Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J. and Sayers, E.W., 2010. GenBank. Nucleic acids research, 39(suppl_1), pp.D32-D37.
[5] Aho, Alfred V., and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." Communications of the ACM 18, no. 6 (1975): 333-340.
[6] Baeza-Yates, Ricardo A., and Gaston H. Gonnet. "Fast text searching for regular expressions or automaton searching on tries." Journal of the ACM (JACM) 43, no. 6 (1996): 915-936.
[7] Su, Wenhe, Xiangke Liao, Yutong Lu, Quan Zou, and Shaoliang Peng. "Multiple sequence alignment based on a suffix tree and center-star strategy: a linear method for multiple nucleotide sequence alignment on spark parallel framework." Journal of Computational Biology 24, no. 12 (2017): 1230-1242.