

MATH 578A Programming Assignment #3

Haifeng Chen

■ Summary

Progressive multiple global sequence alignment algorithm combines the most similar pair of sequences or clusters in each step. For each cluster, it need find the maximum alignment score and delete the score with the merged clusters. Red-black tree get the maximum element in constant time and delete element in $O(\log(n))$ time. The time complexity of my program is $O(r^2 \log(r)m^2)$, r is the number of sequences, m is the length of sequences in the result alignment. In the second example, my program took 1.26 seconds.

■ Algorithm

Progressive multiple global sequence alignment algorithm

Input: r sequences

Output: r sequences alignment

lfPro[i] array, profile of cluster i

cluster[i] array, all the string in cluster i

setDist[i] multiple set, alignment scores between cluster i and all other clusters, this set is descending order, implemented by red-black tree

Dist[i][j] store the alignment score between cluster i and j

Indicator[i] when Indicator[i] equals to 1, cluster i isn't merged to another cluster, otherwise cluster[i] have been merged to another cluster

- (1) Assign each string in one single cluster
- (2) Compute profile for each cluster and store in lfPro[i]
- (3) Using pairwise alignment algorithm to calculate the optimal alignment score for each pair of cluster i and j , and store them in setDist[i], setDist[j], Dist[i][j] and Dist[j][i]
- (4) Assign all Indicator[i] to 1

- (5) Find the maximum score in $\text{setDist}[i][0]$ for all $\text{Indicator}[i]$ is not 0, get the pair $k1$ and $k2$, and the alignment score between cluster $k1$ and $k2$ is maximum
- (6) Delete the score $\text{Dist}[k1][k2]$ in $\text{setDist}[k1]$ and $\text{setDist}[k2]$
- (7) Merge cluster $k2$ to $k1$, set $\text{Indicator}[k2]$ is 0
- (8) Re-compute profile for cluster $k1$
- (9) Using pairwise alignment algorithm to calculate the optimal alignment score for cluster $k1$ to all other clusters that are not merged, store them in $\text{setDist}[i]$, $\text{setDist}[k1]$, and $\text{Dist}[k1][i]$, $\text{Dist}[i][k1]$
- (10) Go to step(5) until there is only one cluster which is not merged

▪ Time Complexity

In the beginning, the pairwise alignment of each pair takes $O(r^2n^2)$ time, r is the number of sequences, n is the length of each sequence. It takes $r - 1$ steps to merge the r sequence to only one cluster. In each step, it should find the maximum alignment score between each pair. For $\text{setDist}[i]$, it can get the maximum score to cluster i in constant time, and it takes $O(r)$ time to find the maximum score between all the r sequences. In addition, it should delete the element in $\text{setDist}[i]$ which is the alignment score between cluster i and cluster $k1, k2$. Multiple set is implemented by red-black tree which get the maximum value in constant time and delete the element in $\log(r)$ time. Thus, delete all the alignment score takes $r\log(r)$ time.

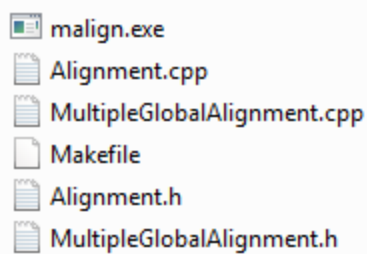
At last, it should update the alignment score to cluster $k1$, it takes $O(r)$ time. So the total time complexity is $O(r^2n^2 + (r - 1)r\log(r)m^2)$, here, m is the length of the sequence when all the sequence merged together. Therefore, the time complexity is $O(r^2\log(r)m^2)$. Figure 1 also shows the running time is linearly related to $r^2\log(r)m^2$.

▪ Programming Tips

- (1) In the very beginning of the algorithm, it calculates the alignment scores of all pairs of clusters. Actually, each cluster just has only one sequence and it do not need to using profile. I use pairwise alignment algorithm to get the alignment score of each pair.

- (2) In the program, the most time-consuming part is computing the score of column i of one cluster and column j of another cluster. In every pairwise alignment, it should call this function $3mn$ times. So this function need more efficient.
- (3) The pairwise alignment algorithm function was called many times. If every time the program assigned space for array s and array l , it took a lot time. Thus, if the program assign space for s and l only once in the beginning, it will save time.

■ Code and executable file



■ Compiler

MinGW GCC 4.7.2

```
>g++ -o mgalign.exe Alignment.cpp MultipleGlobalAlignment.cpp
```

■ Running Environment

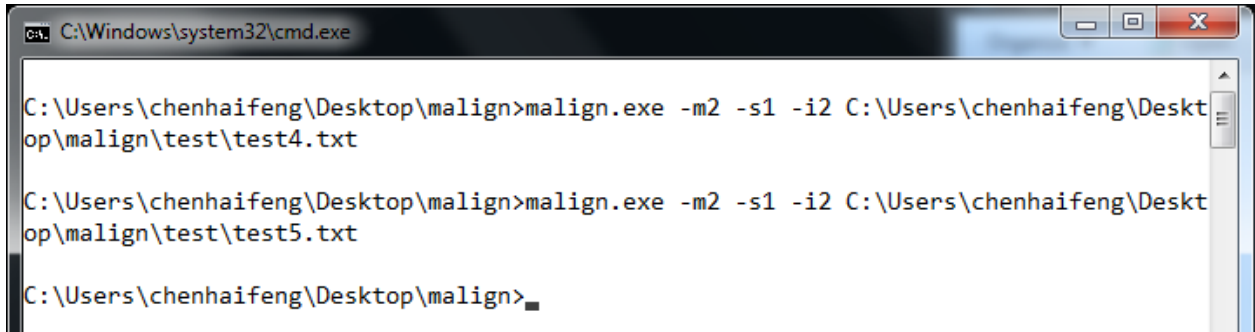
Windows 7 Professional 32-bit

Processor: Inter(R) Core(TM) 2 Duo CPU T6570 @2.10GHz 2.10GHz

Installed memory (RAM): 2.00 GB

■ Results of the two test examples

The first example took about 0 second, and the second example took 1.264 seconds.



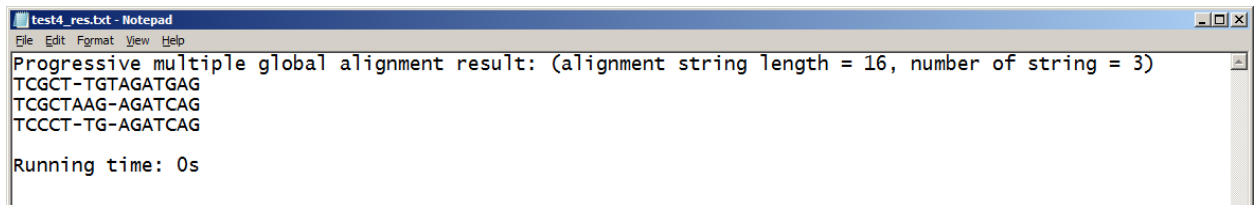
```
C:\Windows\system32\cmd.exe

C:\Users\chenhaifeng\Desktop\malign>malign.exe -m2 -s1 -i2 C:\Users\chenhaifeng\Desktop\malign\test\test4.txt

C:\Users\chenhaifeng\Desktop\malign>malign.exe -m2 -s1 -i2 C:\Users\chenhaifeng\Desktop\malign\test\test5.txt

C:\Users\chenhaifeng\Desktop\malign>
```

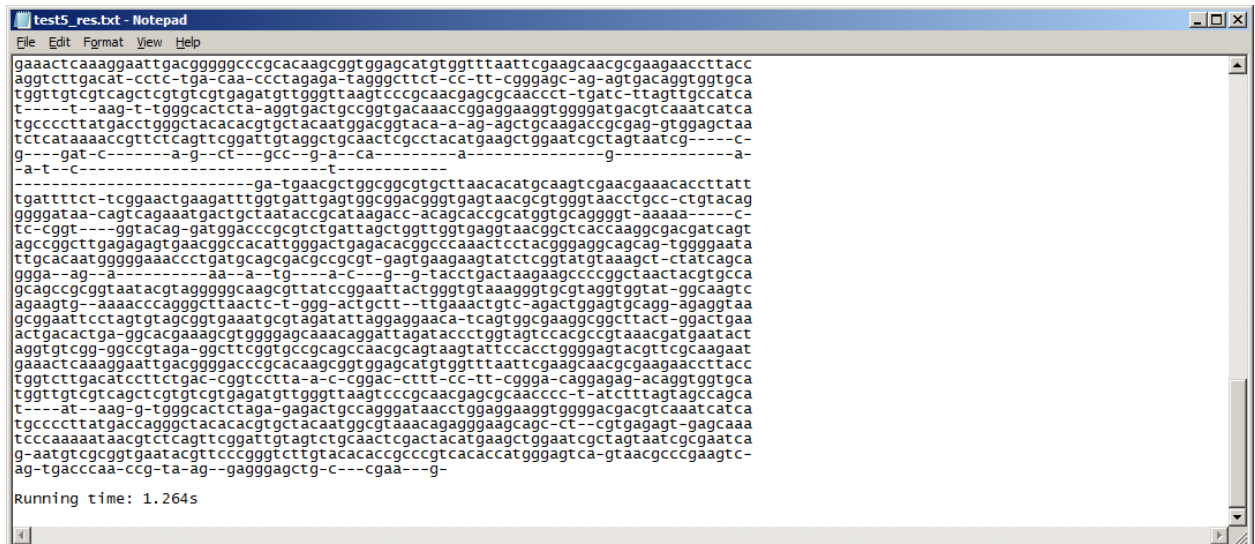
To see the completed result, please go to the attached files
|test_result|test4_res.txt and |test_result|test5_res.txt.



```
test4_res.txt - Notepad
File Edit Format View Help

Progressive multiple global alignment result: (alignment string length = 16, number of string = 3)
TCGCT-TGATAGTAG
TCGCTAAG-AGATCAG
TCCT-TG-AGATCAG

Running time: 0s
```



```
test5_res.txt - Notepad
File Edit Format View Help

gaaactcaaaggaattgacggggccgcacaaagcgggtgagcatgtggtttaattcgaagcaacgcgaagaaccttacc
aggtcttgacat-cctc-tga-caa-ccctagaga-tagggcttct-cc-tt-cgggagc-ag-agtgacaggtggtgca
tgggtgtcgtcagctcgtgctgagatgttgggttaagtcgcccaacgagcgcaacccct-tgac-tttagttgccatca
t-----aa-g-t-tgggcaactca-aggtgactgccggtgacaaacggaggaaggtggggatgacgtcaaatcatca
tgcccttatgactgggtacacacgtgctacaatggagcgtaca-a-ag-agctgcaagaccgcgag-gtggagctaa
tctcataaaaacggttctcagttcggattgtaggctgcaactcgctacatgaagctggaatcgctagtaaatcg-----c-
g-----gat-c-----a-g-----ct-----gcc--g--a--ca-----a-----g-----a-----
-a-t--C-----t-----ga-tgaacgctggcggcgtgcttaacacatgcaagtcgaacgaaacaccttatt
tgattttct-tcggaactgaagatttgggtgattgagtgccgagcgggtgagtacgcgtgggttaacctgcc-ctgtacag
ggggataa-cagtcagaaatgactgctaataaccgcataagacc-acagcaccgcagtggtgcagggt-aaaaa-----c-
tc-cggt----ggtacag-gatggaccgcgctctgattagctggttgggtgaggtaacggctcaccaggcgacgatcagt
agccggcttgagagagtgaaacgacacattgggactgagacacggcccaactcctacgggagcagcag-tggggaata
ttgcacaatggggaacccctgatgcagcagcgcgcgt-gagtgaagaagtatctcggtatgtaagct-ctatcagca
ggga--ag--a-----aa--a--tg-----a-c---g--g-tacctgactaagaagcccgctaacctacgtgcc
gcagccggttaacgtagggggcaagcgttatccggaattactgggtgtaaggggtgctaggtggtat-ggcaagtc
agaagtg-aaaaccagggttctaactc-t-ggg-actgctt--ttgaaactgtc-agactggagtgagg-agaggtaa
gcggaattcctagtgagcgtgaaatgcgtagatattaggaggaaca-tcagtgccgaagcggcttact-ggactgaa
actgacactga-ggcacgaagcgtggggagcaaacaggattagataccctggtagtccacgcgtaaacgatgaatact
aggtgtcgg-ggcccgtaga-ggcttcggtgccgacgcaacgcagtaagattccacctggggagtagcttcgcaagaat
gaaactcaaaggaattgacggggaccgcacaaagcgggtgagcatgtggtttaattcgaagcaacgcgaagaaccttacc
tgggtcttgacatccttctgac-cggtcctta-a-c-cggac-cttt-cc-tt-cggga-caggagag-acaggtggtgca
tgggtgtcgtcagctcgtgctgagatgttgggttaagtcgcccaacgagcgcaacccc-t-acttttagtagccagca
t-----at--aag-g-tgggcaactcaga-gagactgcccagggataacctggaggaaggtggggacgacgtcaaatcatca
tgcccttatgacagaggtacacacgtgctacaatggcgtaaacagaggaagcagc-ct--cgtgagagt-gagcaaa
tcccaaaaataacgtctcagttcggattgtagtcgcaactcgactacatgaagctggaatcgctagtaaatcgcaatca
g-aatgtcgcggtgaatcgttccgggtcttgcacacaccccgctcacacctgggagtc-gtaacgcccgaagtc-
ag-tgacccaa-cg-ta-ag--gagggagctg-c---cgaa---g-
```

■ Runtime Comparison

I randomly generated 23 datasets to test the runtime of malign.exe. The lengths of the sequences are the same in each test data. The results are shown in Table 1, and Figure 1 is the curve of running time with respect to $r^2 \log(r)m^2$.

Table 1 comparison between running time and length and number of sequences

Test	original length (n)	# of sequences (r)	result length (m)	$r^2 \log(r)m^2$	Time (s)
1	5	29	6	1.47E+05	0
2	10	51	21	6.51E+06	0.015
3	20	90	57	1.71E+08	0.124
4	30	85	78	2.82E+08	0.249
5	50	38	100	7.58E+07	0.124
6	80	68	196	1.08E+09	0.982
7	100	23	191	8.73E+07	0.172
8	120	48	286	1.05E+09	1.138
9	150	19	263	1.06E+08	0.296
10	300	74	790	2.12E+10	16.474
11	500	36	1,110	8.26E+09	11.154
12	800	93	2,268	2.91E+11	184.610
13	1,200	98	3,443	7.53E+11	463.321
14	1,300	21	2,324	1.05E+10	23.758
15	1,500	26	2,930	2.73E+10	51.792
16	1,800	45	4,119	1.89E+11	215.077
17	2,000	64	5,068	6.31E+11	542.288
18	2,400	9	3,457	3.07E+09	13.728
19	2,500	81	6,887	1.97E+12	1,359.980
20	3,000	83	8,310	3.03E+12	2,057.690
21	4,000	65	10,252	2.67E+12	2,239.110
22	5,000	63	12,715	3.84E+12	3,258.780
23	7,000	18	12,266	2.03E+11	497.718

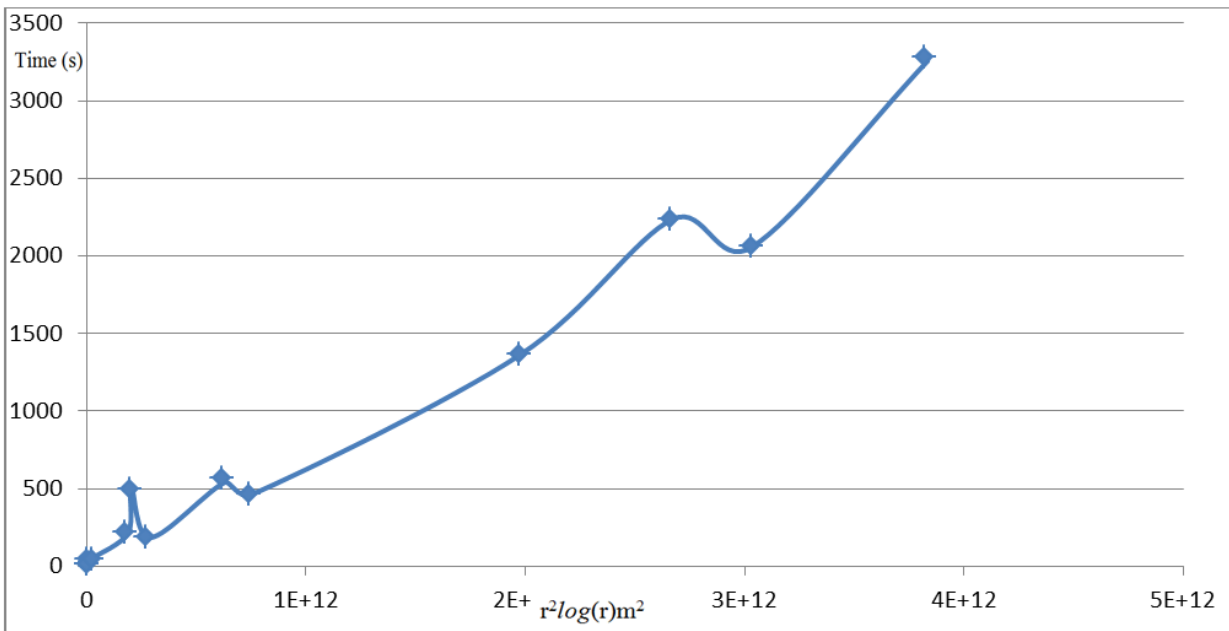


Figure 1 Running time with respect to $r^2 \log(r)m^2$

■ Reference

Introduction to Information Retrieval

By *Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze*

Chapter 17 Time complexity of Hierarchical Clustering

Website: <http://nlp.stanford.edu/IR-book/html/htmledition/time-complexity-of-hac-1.html>

```

//Source Code
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Alignment.h

#ifndef ALIGNMENT_H_
#define ALIGNMENT_H_

#include "MultipleGlobalAlignment.h"

#include <vector>
#include <string>
#include <cstdio>
#include <fstream>
#include <iostream>

using namespace std;
using namespace spaceMultipleGlobalAlignment;

const string ErrorInfo =
    "Please input correct parameters! For example:\n\
    >galign.exe -m2 -s1 -i2 seq1.txt seq2.txt\n\
    scoring function with +2 for match, -1 for mismatch and -2
for indels\
    to align two DNA sequences in seq1.txt and seq2.txt.\n";

int checkFilePath(const string & strPath)
{
    ifstream fin(strPath.c_str());
    int r = 0;
    if (!fin.good())
        r = 0;
    else
        r = 1;
    fin.close();
    return r;
}
#endif /* ALIGNMENT_H_ */

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Alignment.cpp

#include "Alignment.h"

int main(int argc, char *argv[])

```



```

* MATH 578A Homework3
* Multiple Global Alignment Algorithm
* MultipleGlobalAlignment.cpp
*
* Author: Haifeng Chen
* Contact: haifengc at usc dot edu

* Compiler:
* (1) Ubuntu 12.10 32-bit, g++ (Ubuntu/Linaro 4.7.2-2ubuntu1) 4.7.2
* (2) Windows 7 32-bit, MinGW GCC 4.7.2

* Created on: Mar 9, 2013
*/

```

```

#ifndef MULTIPLEGLOBALALIGNMENT_H_
#define MULTIPLEGLOBALALIGNMENT_H_

#include <set>
#include <map>
#include <ctime>
#include <cmath>
#include <limits>
#include <vector>
#include <string>
#include <cstdio>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include <sstream>
#include <iostream>

using namespace std;

namespace spaceMultipleGlobalAlignment
{

#define I (i - 1)
#define J (j - 1)
#define P (p - 1)
#define Q (q - 1)
#define DIAG ('a')
#define UP ('b')
#define LEFT ('c')
#define lf_inf (std::numeric_limits<double>::max())

struct multisetcmp

```

```

{
    bool operator()(const pair<double, int> & a,
                    const pair<double, int> & b) const
    {
        return a.first > b.first;
    }
};

enum MatchLabel
{
    MATCH = 0, MISMATCH = 1, INDEL = 2, TWOindel = 3
};

enum Nucleotide
{
    A = 0, C = 1, G = 2, T = 3, d = 4
};

const char alphabet[5] = { 'A', 'C', 'G', 'T', '-' };

class CMultipleGlobalAlignment
{
public:
    CMultipleGlobalAlignment(const string & file, const vector<int> &
weight);
    ~CMultipleGlobalAlignment();
    void runMultipleGlobalAlignment();
private:
    string filePath;
    vector<string> seqs;
    vector<int> w;

    vector<vector<double> > s;
    //vector<vector<double> > epq; //the profile value gotten from
Pro1(i) and Pro2(j)
    vector<vector<char> > l;

    /* Dist stores the distance from i to j, which is n×n matrix.
    *
    * We should use a data structure, which can insert, delete and
    * find element in log(n) time. Red-black tree insert, delete
    * and find emelmetn in log(n) time.
    *
    * In STL, set, map, multiset, multimap are implemented by red-
    black tree.

```

```

    * Here, we don't need to change the value of Distance, we just
need to
    * delete or insert, so we use multiset.
    * reference http://nlp.stanford.edu/IR-
book/html/htmledition/time-complexity-of-hac-1.html
    */
    vector<multiset<pair<double, int>, multisetcmp> > setDist;
    vector<vector<double> > Dist;
    vector<int> Indicator; // tag[i] marks whether sequence i has
been clustered.
    vector<vector<string> > cluster; // the sequence id in each
cluster
    vector<vector<int> > clusterID;
    vector<vector<vector<double> > > clusterProfile;

    map<int, string> mapres;

    clock_t TimeStart, TimeEnd;
    double memory;

    void outPutCluster();
    void resultDisplay();
    void multipleGlobalAlign();
    void stringReverse(string & str);
    int readString(const string & strPath);
    void outputfastaFormat(const string & str);
    MatchLabel charMatch(const char & a, const char & b);
    void outputfastaFormat(ofstream & fout, const string & str);
    pair<double, char> max(const double & s1, const double & s2,
        const double & s3);
    void outputResultString(const string & str, const int & start,
        const int & end);
    void setMulitiStrAlignProfile(const vector<string> & str,
        vector<vector<double> > & lfPro);
    void setMulitiStrAlignProfile(const string & str,
        vector<vector<double> > & lfPro);
    double computeProfileIJ(const vector<vector<double> > & lfPro,
        const int & si);
    double computeProfileIJ(const vector<vector<double> > & lfPro1,
        const int & si, const vector<vector<double> > & lfPro2,
        const int & sj);
    double globalAlignScore(const vector<string> & str1,
        const vector<string> & str2, const
vector<vector<double> > & lfPro1,
        const vector<vector<double> > & lfPro2);
    double globalAlignScore(const string & U, const string & V);

```

```

        void globalAlignPath(vector<string> & str1, vector<string> & str2,
                             const vector<vector<double> > & lfPro1,
                             const vector<vector<double> > & lfPro2);

};

}

#endif /* MULTIPLEGLOBALALIGNMENT_H_ */

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//MultipleGlobalAlignment.cpp

#include "MultipleGlobalAlignment.h"

using namespace spaceMultipleGlobalAlignment;

CMultipleGlobalAlignment::CMultipleGlobalAlignment(const string & file,
                                                    const vector<int> & weight)
{
    filePath = file;
    w = weight;
    memory = 0.0;

    TimeStart = 0.0;
    TimeEnd = 0.0;
}

CMultipleGlobalAlignment::~CMultipleGlobalAlignment()
{
}

int CMultipleGlobalAlignment::readString(const string & strPath)
{
    ifstream fin(strPath.c_str());
    if (!fin.good()) {
        printf("Cannot open the file: %s.\n", strPath.c_str());
        fin.close();
        return 0;
    }
    bool vaild = true;
    string strTmp, strIn;
    strIn.clear();

```

```

seqs.clear();
while (!fin.eof()) {
    getline(fin, strTmp, '\n');
    if (strTmp.size() == 0)
        continue;
    for (int i = 0; i < (int) strTmp.size(); i++) {
        if (strTmp[i] == '>') {
            if (strIn.size() != 0) {
                seqs.push_back(strIn);
                strIn.clear();
            }
            break;
        }
        if (strTmp[i] != 'A' && strTmp[i] != 'C' &&
strTmp[i] != 'G'
                                && strTmp[i] != 'T' && strTmp[i] != 'N' &&
strTmp[i] != 'a'
                                && strTmp[i] != 'c' && strTmp[i] != 'g' &&
strTmp[i] != 't'
                                && strTmp[i] != 'n') {
            vaild = false;
        } else {
            strIn += strTmp[i];
        }
    }
    if (strIn.size() != 0) {
        seqs.push_back(strIn);
        strIn.clear();
    }
    if (vaild == false) {
        printf(
                                "The DNA file %s contains characters which is not
a, c, g, t or n. \n
                                These characters have been deleted.\n",
                                strPath.c_str());
    }
    fin.close();
    return 1;
}

inline void CMultipleGlobalAlignment::stringReverse(string & str)
{
    int n = str.size();
    char c;

```

```

    for (int i = 0; i < n / 2; i++) {
        c = str[i];
        str[i] = str[n - i - 1];
        str[n - i - 1] = c;
    }
}

inline MatchLabel CMultipleGlobalAlignment::charMatch(const char & a,
    const char & b)
{
    if (a == '-' && b == '-')
        return TWOindel;
    else if (a == '-' || b == '-')
        return INDEL;
    else if (a == b)
        return MATCH;
    else
        return MISMATCH;
}

inline pair<double, char> CMultipleGlobalAlignment::max(const double &
s1,
    const double & s2, const double & s3)
{
    /*if two of them are equal, then there are more than one optimal
    path*/
    if (s1 >= s2) {
        if (s1 >= s3)
            return pair<double, char>(s1, DIAG);
        else
            return pair<double, char>(s3, LEFT);
    } else {
        if (s2 >= s3)
            return pair<double, char>(s2, UP);
        else
            return pair<double, char>(s3, LEFT);
    }
}

void CMultipleGlobalAlignment::outputfastaFormat(const string & str)
{
    for (size_t t = 0; t < str.size(); t++) {
        if (t % 80 == 0 && t != 0) {
            printf("\n");
        }
        printf("%c", str[t]);
    }
}

```

```

    }
    printf("\n");
}

void CMultipleGlobalAlignment::outputfastaFormat(ofstream & fout,
    const string & str)
{
    for (size_t t = 0; t < str.size(); t++) {
        if (t % 80 == 0 && t != 0) {
            fout << endl;
        }
        fout << str[t];
    }
    fout << endl;
}

void CMultipleGlobalAlignment::resultDisplay()
{
    string strfile = filePath;
    size_t pos = strfile.find(".txt");
    strfile = strfile.substr(0, pos);

    char chr[100];
    sprintf(chr, "%s_res.txt", strfile.c_str());

    ofstream fout(chr);
    size_t length = 0;
    for (size_t i = 0; i < cluster.size(); i++) {
        if (Indicator[i] && cluster[i].size() != 0) {
            if (cluster[i].size() > 0)
                length = cluster[i][0].size();
            for (size_t j = 0; j < cluster[i].size(); j++) {
                mapres[clusterID[i][j]] = cluster[i][j];
            }
        }
    }

    fout << "Progressive multiple global alignment result: "
        << "(alignment string length = " << length
        << ", number of string = " << mapres.size() << ")" <<
endl;

    for (map<int, string>::iterator it = mapres.begin(); it !=
mapres.end();
        it++) {
        outputfastaFormat(fout, it->second);
    }
}

```

```

    }
    fout << endl;
    fout << "Running time: " << (double) (TimeEnd - TimeStart) /
CLOCKS_PER_SEC
        << "s" << endl;
    fout.close();
}

void CMultipleGlobalAlignment::setMulitiStrAlignProfile(
    const vector<string> & str, vector<vector<double> > & lfPro)
{
    /* Time and Space complexity
    * Time O(5n)
    * Space O(5n)
    * */
    if (str.size() == 0)
        return;
    lfPro.resize(5);
    size_t n = str[0].size();
    vector<double> lfv(n);
    for (size_t t = 0; t < lfPro.size(); t++) {
        lfPro[t] = lfv;
    }
    lfv.clear();
    size_t r = str.size();
    for (size_t i = 0; i < n; i++) {
        int cnt[5] = { 0 };
        for (size_t j = 0; j < r; j++) {
            if (str[j][i] == 'a' || str[j][i] == 'A')
                cnt[A]++;
            else if (str[j][i] == 'c' || str[j][i] == 'C')
                cnt[C]++;
            else if (str[j][i] == 'g' || str[j][i] == 'G')
                cnt[G]++;
            else if (str[j][i] == 't' || str[j][i] == 'T')
                cnt[T]++;
            else if (str[j][i] == '-')
                cnt[d]++;
        }
        for (int j = 0; j < 5; j++) {
            if (cnt[j] == 0)
                lfPro[j][i] = 0.0;
            else
                lfPro[j][i] = (double) cnt[j] / (double)
str.size();
        }
    }
}

```



```

    }
}

void CMultipleGlobalAlignment::setMulitiStrAlignProfile(const string &
    str,
    vector<vector<double> > & lfPro)
{
    vector<string> sstr;
    sstr.push_back(str);
    setMulitiStrAlignProfile(sstr, lfPro);
}

inline double CMultipleGlobalAlignment::computeProfileIJ(
    const vector<vector<double> > & lfPro1, const int & si,
    const vector<vector<double> > & lfPro2, const int & sj)
{
    /* Time and Space complexity
    * Time O(25)
    * no extra Space
    * */
    double sum = 0.0;

    double s = 0.0;
    for (size_t i = 0; i < 4; i++) {
        s += lfPro1[i][si] * lfPro2[i][sj];
    }
    sum += s * w[MATCH];

    s = 0.0;
    for (size_t i = 0; i < 4; i++) {
        s += lfPro1[i][si]
            * (lfPro2[0][sj] + lfPro2[1][sj] + lfPro2[2][sj]
+ lfPro2[3][sj]
            - lfPro2[i][sj]);
    }
    sum += s * w[MISMATCH];

    s = 0.0;
    s += lfPro1[4][si] * (1 - lfPro2[4][sj]);
    s += lfPro2[4][sj] * (1 - lfPro1[4][si]);
    sum += s * w[INDEL];

    return sum;
}

inline double CMultipleGlobalAlignment::computeProfileIJ(

```

```

        const vector<vector<double> > & lfPro, const int & si)
{
    double sum = 0.0;
    sum = lfPro[0][si] + lfPro[1][si] + lfPro[2][si] + lfPro[3][si];
    return sum * w[INDEL];
}

double CMultipleGlobalAlignment::globalAlignScore(const string & U,
        const string & V)
{
    /* Time and Space complexity
    * Time O(mn)
    * Space O(mn)
    * here we can use linear space pairwise alignment
    * actually, we can use pairwiseGlobalAlignScore(const
vector<string> & str1...)
    * to calculate the score of two string, but this function will
save time.
    */
    size_t n = U.size(), m = V.size();

    s.resize(n + 1);

    for (size_t i = 0; i <= n; i++) {
        s[i].resize(m + 1, 0);
    }

    s[0][0] = 0;
    for (size_t j = 1; j <= m; j++) {
        s[0][j] = s[0][j - 1] + w[INDEL];
    }

    for (size_t i = 1; i <= n; i++) {
        s[i][0] = s[i - 1][0] + w[INDEL];
        for (size_t j = 1; j <= m; j++) {
            double s1, s2, s3;
            s1 = s[i - 1][j - 1] + w[charMatch(U[i], V[j])];
            s2 = s[i - 1][j] + w[INDEL];
            s3 = s[i][j - 1] + w[INDEL];
            pair<double, char> charMatchResult = max(s1, s2, s3);
            s[i][j] = charMatchResult.first;
        }
    }

    return s[n][m];
}

```

```

double CMultipleGlobalAlignment::globalAlignScore(const vector<string>
& str1,
            const vector<string> & str2, const vector<vector<double> > &
lfPro1,
            const vector<vector<double> > & lfPro2)
{
    /* Time and Space complexity
    * Time O(25mn)
    * Space O(mn)
    * here we can use linear space pairwise alignment
    * */
    if (str1.size() == 1 && str2.size() == 1) {
        string U = str1[0];
        string V = str2[0];
        return globalAlignScore(U, V);
    }

    size_t n = str1[0].size(), m = str2[0].size();

    s.resize(n + 1);

    for (size_t i = 0; i <= n; i++) {
        s[i].resize(m + 1, 0);
    }

    s[0][0] = 0;
    for (size_t j = 1; j <= m; j++) {
        s[0][j] = s[0][j - 1] + computeProfileIJ(lfPro2, J);
    }

    for (size_t i = 1; i <= n; i++) {
        s[i][0] = s[i - 1][0] + computeProfileIJ(lfPro1, I);
        for (size_t j = 1; j <= m; j++) {
            double s1, s2, s3;
            s1 = s[i - 1][j - 1] + computeProfileIJ(lfPro1, I,
lfPro2, J);
            s2 = s[i - 1][j] + computeProfileIJ(lfPro1, I);
            s3 = s[i][j - 1] + computeProfileIJ(lfPro2, J);
            pair<double, char> charMatchResult = max(s1, s2, s3);
            s[i][j] = charMatchResult.first;
        }
    }

    return s[n][m];
}

```

```

void CMultipleGlobalAlignment::globalAlignPath(vector<string> & str1,
        vector<string> & str2, const vector<vector<double> > &
lfPro1,
        const vector<vector<double> > & lfPro2)
{
    /*pairwiseGlobalAlignScore pairwiseGlobalAlignPath have much
common code,
    * but I use two different function, and don't use one to call
another, because
    * for pairwiseGlobalAlignScore, we just need to calculate score,
we do not
    * need to store the array l. we can use linear space pairwise
alignment.
    * this will save some space;
    */
    /* Time and Space complexity
    * Time O(mn)
    * Space O(2mn)
    */
    size_t n = str1[0].size(), m = str2[0].size();

    s.resize(n + 1);
    l.resize(n + 1);
    for (size_t i = 0; i <= n; i++) {
        s[i].resize(m + 1, 0);
        l[i].resize(m + 1, LEFT);
    }

    s[0][0] = 0;
    for (size_t j = 1; j <= m; j++) {
        s[0][j] = s[0][j - 1] + computeProfileIJ(lfPro2, J);
        l[0][j] = LEFT;
    }
    for (size_t i = 1; i <= n; i++) {
        s[i][0] = s[i - 1][0] + computeProfileIJ(lfPro1, I);
        l[i][0] = UP;
        for (size_t j = 1; j <= m; j++) {
            double s1, s2, s3;
            s1 = s[i - 1][j - 1] + computeProfileIJ(lfPro1, I,
lfPro2, J);
            s2 = s[i - 1][j] + computeProfileIJ(lfPro1, I);
            s3 = s[i][j - 1] + computeProfileIJ(lfPro2, J);
            pair<double, char> charMatchResult = max(s1, s2, s3);
            s[i][j] = charMatchResult.first;
            l[i][j] = charMatchResult.second;
        }
    }
}

```

```

    }
}
int p = n, q = m;
vector<string> rstr1(str1.size());
vector<string> rstr2(str2.size());
while (p >= 0 && q >= 0 && (p + q != 0)) { // trace back from
s[n][m] to s[0][0]
    if (l[p][q] == DIAG) {
        for (size_t t = 0; t < str1.size(); t++) {
            rstr1[t].push_back(str1[t][P]);
        }
        for (size_t t = 0; t < str2.size(); t++) {
            rstr2[t].push_back(str2[t][Q]);
        }
        p = p - 1;
        q = q - 1;
    } else if (l[p][q] == UP) {
        for (size_t t = 0; t < str1.size(); t++) {
            rstr1[t].push_back(str1[t][P]);
        }
        for (size_t t = 0; t < str2.size(); t++) {
            rstr2[t].push_back('-');
        }
        p = p - 1;
    } else if (l[p][q] == LEFT) {
        for (size_t t = 0; t < str1.size(); t++) {
            rstr1[t].push_back('-');
        }
        for (size_t t = 0; t < str2.size(); t++) {
            rstr2[t].push_back(str2[t][Q]);
        }
        q = q - 1;
    }
}
for (size_t t = 0; t < str1.size(); t++) {
    stringReverse(rstr1[t]);
}
for (size_t t = 0; t < str2.size(); t++) {
    stringReverse(rstr2[t]);
}
str1 = rstr1;
str2 = rstr2;
}

void CMultipleGlobalAlignment::multipleGlobalAlign()
{

```

```

/* Time and Space complexity
 * Time  $O(r^2 * (\underline{mn} + 2\log(r)) + r(r + \underline{mn} + r * (4\log(r) + \underline{mn})))$ 
 * Space  $O(\underline{mn})$ 
 */
size_t r = seqs.size();
setDist.resize(r);
Dist.resize(r);
for (size_t t = 0; t < Dist.size(); t++) {
    Dist[t].resize(r);
}
Indicator.resize(r, 1);
cluster.resize(r);
clusterID.resize(r);
clusterProfile.resize(r);
clusterProfile.resize(r);
vector<vector<double>> lfPro;
for (size_t i = 0; i < r; i++) {
    Indicator[i] = 1;
    cluster[i].push_back(seqs[i]);
    clusterID[i].push_back(i);
    setMulitiStrAlignProfile(seqs[i], lfPro);
    clusterProfile[i] = lfPro;
}
seqs.clear();
for (size_t i = 0; i < r; i++) {
    for (size_t j = 0; j < i; j++) {
        double score = globalAlignScore(cluster[i], cluster[j],
                                           clusterProfile[i], clusterProfile[j]);
        Dist[i][j] = score;
        Dist[j][i] = score;
        setDist[i].insert(pair<double, int>(score, j));
        setDist[j].insert(pair<double, int>(score, i));
    }
}
for (size_t t = 1; t <= r - 1; t++) {
    double maxdis = -1f_inf;
    size_t maxIndex = 0;
    for (size_t i = 0; i < r; i++) {
        if (Indicator[i] && setDist[i].begin()->first > maxdis)
        {
            maxdis = setDist[i].begin()->first;
            maxIndex = i;
        }
    }
    size_t k1 = maxIndex;

```

```

size_t k2 = setDist[k1].begin()->second;
/*merge k2 to k1*/
setDist[k1].clear();
setDist[k2].clear();
/*
 * merge the strings in cluster1 and cluster2,
 * and after this, they have the same length
 */
globalAlignPath(cluster[k1], cluster[k2], clusterProfile[k1],
                clusterProfile[k2]);
for (size_t j = 0; j < cluster[k2].size(); j++) {
    cluster[k1].push_back(cluster[k2][j]);
    clusterID[k1].push_back(clusterID[k2][j]);
}
Indicator[k2] = 0;
cluster[k2].clear();
clusterID[k2].clear();
clusterProfile[k2].clear();
setMulitiStrAlignProfile(cluster[k1], lfPro);
clusterProfile[k1] = lfPro;
for (size_t j = 0; j < r; j++) {
    if (Indicator[j] && j != k1) {
        setDist[j].erase(pair<double, int>(Dist[j][k1],
k1));
        setDist[j].erase(pair<double, int>(Dist[j][k2],
k2));

        Dist[j][k1] = globalAlignScore(cluster[k1],
cluster[j],
                clusterProfile[k1], clusterProfile[j]);
        Dist[k1][j] = Dist[j][k1];

        setDist[j].insert(pair<double, int>(Dist[j][k1],
k1));
        setDist[k1].insert(pair<double, int>(Dist[k1][j],
j));
    }
}
}

}

void CMultipleGlobalAlignment::runMultipleGlobalAlignment()
{
    time_t rawtime;
    time(&rawtime);
    if (!readString(filePath))

```

```
        return;

    TimeStart = clock();
    multipleGlobalAlign();
    TimeEnd = clock();
    resultDisplay();
}
```