[CENG 315 ALL Sections] Algorithms

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - 571 - Computer Engineering
      - CENG 223 All Sections
      - CENG 223 Section 2
      - CENG 315 ALL Sections
        - Participants
        - Competencies
        - Grades
        - General
        - October 2 - October 8
        - October 9 - October 15
        - October 15 - October 22
        - October 23 - October 29
        - October 30 - November 5
        - November 6 - November 12
        - November 13 - November 19
          - THE3
            - Description
            - Submission view
        - November 20 - November 26
        - November 27 - December 3
        - December 3 - December 10
        - December 11 - December 17
        - December 18 - December 24
        - December 25 - December 31
        - January 1 - January 7
        - January 8 - January 14
      - CENG 315 Section 1

Description | Submission view

Grade

Reviewed on Thursday, November 30, 2023, 5:32 PM by Automatic grade
Grade: 100.00 / 100.00

Assessment report [-]
[+]**Output of make**
[+]**For input 01:**
[+]**For input 02:**
[+]**For input 03:**
[+]**For input 04:**
[+]**For input 05:**
[+]**For input 06:**
[+]**For input 07:**
[+]**For input 08:**
[+]**For input 09:**
[+]**For input 10:**
[+]**For input 11:**
[+]**For input 12:**
[+]**For input 13:**
[+]**For input 14:**
[+]**For input 15:**
[+]**For input 16:**
[+]**For input 17:**
[+]**For input 18:**
[+]**For input 19:**
[+]**For input 20:**

Submitted on Saturday, November 18, 2023, 6:46 PM (Download)

the3.cpp

```cpp
#include "the3.h"

// do not add extra libraries, but you can define helper functions below.
int count_glob = 0;
int compare(char a, char b, int mismatch, int match) {
  if (a == b)
    return match;
  else
    return -mismatch;
}

/*
PART 1
you are expected to call recursive_alignment (as the name suggests) recursively to find an alignment.
initial call_count value given to you will be 0.
you should check if call_count >= 1000000, if so, set possible_alignment string to "STACK LIMIT REACHED", return INT_MIN (or anything - it will not be chec
*/

int recursive_alignment(std::string sequence_A, std::string sequence_B, int gap, int mismatch, int match, std::string &possible_alignment, int call_count)
    count_glob++;
    int highest_alignment_score;
    call_count = count_glob;
    if (call_count >= 1000000) {
        possible_alignment = "STACK LIMIT REACHED";
        return INT_MIN;
    }

    if (sequence_A.empty()) {
        possible_alignment += std::string(sequence_B.length(), '_');
        return -gap * sequence_B.length();
    }

    if (sequence_B.empty()) {
        possible_alignment += std::string(sequence_A.length(), '.');
        return -gap * sequence_A.length();
    }

    int score_match_mismatch ;
    if(sequence_A.back() == sequence_B.back())
        score_match_mismatch = match + recursive_alignment(sequence_A.substr(0, sequence_A.length() - 1), sequence_B.substr(0, sequence_B.length() - 1), ga
    else
        score_match_mismatch = -mismatch + recursive_alignment(sequence_A.substr(0, sequence_A.length() - 1), sequence_B.substr(0, sequence_B.length() - 1)

    std::string alignment_match_mismatch;
    if(sequence_A.back() == sequence_B.back())
        alignment_match_mismatch = possible_alignment + sequence_A.back();
    else
        alignment_match_mismatch = possible_alignment + '!';


    std::string temp_align_A;
    int score_gap_A = -gap + recursive_alignment(sequence_A, sequence_B.substr(0, sequence_B.length() - 1), gap, mismatch, match, temp_align_A, call_count)
    temp_align_A += '_';

    std::string temp_align_B;
    int score_gap_B = -gap + recursive_alignment(sequence_A.substr(0, sequence_A.length() - 1), sequence_B, gap, mismatch, match, temp_align_B, call_count)
    temp_align_B += '.';
    if (count_glob > 1000000)
    {
        possible_alignment = "STACK LIMIT REACHED";
        return INT_MIN;
    }

    if (score_match_mismatch >= score_gap_A && score_match_mismatch >= score_gap_B)
    {
        possible_alignment = alignment_match_mismatch;
        highest_alignment_score = score_match_mismatch;
    }
    else if (score_gap_A >= score_gap_B)
    {
        possible_alignment = temp_align_A;
        highest_alignment_score = score_gap_A;
    }
    else
    {
        possible_alignment = temp_align_B;
        highest_alignment_score = score_gap_B;
    }

    return highest_alignment_score;
}



/*
PART 2
you are expected to create a dynamic programming table to find the highest alignment score.
then you will need to reconstruct a possible alignment string from the table.
*/
int dp_table_alignment(std::string sequence_A, std::string sequence_B, int gap, int mismatch, int match, std::string &possible_alignment){

    int** dp_table;
    dp_table = new int* [sequence_A.length()+1];
    for (int i = 0; i < sequence_A.length()+1; i++) {
      dp_table [i] = new int [sequence_B.length()+1];
    }
    int highest_alignment_score;

    for(int i = 0; i <= sequence_A.length(); i++)
        dp_table[i][0] = -i * gap;
    for(int j = 0; j <= sequence_B.length(); j++)
        dp_table[0][j] = -j * gap;

    for(int i = 1; i <= sequence_A.length(); i++)
    {
        for(int j = 1; j <= sequence_B.length(); j++)
        {
            int temp_score;
            if(sequence_A[i-1] == sequence_B[j-1])
                temp_score = match;
            else
                temp_score = -mismatch;

            dp_table[i][j] = std::max({dp_table[i-1][j-1] + temp_score, dp_table[i-1][j] - gap, dp_table[i][j-1] - gap});
        }
    }

    int i = sequence_A.length(), j = sequence_B.length();
    while(i > 0 && j > 0)
    {
        if(dp_table[i][j] == dp_table[i-1][j-1] + ((sequence_A[i-1] == sequence_B[j-1]) ? match : -mismatch))
        {
            if(sequence_A[i-1] == sequence_B[j-1])
                possible_alignment = sequence_A[i-1] + possible_alignment;
            else
                possible_alignment = "!" + possible_alignment;
            i--;j--;
        }
        else if(dp_table[i][j] == dp_table[i-1][j] - gap)
        {
            possible_alignment = "." + possible_alignment;
            i--;
        } else
        {
            possible_alignment = "_" + possible_alignment;
            j--;
        }
    }

    for( ; i > 0; i--)
        possible_alignment = "." + possible_alignment;

    for( ; j > 0; j--)
        possible_alignment = "_" + possible_alignment;

    highest_alignment_score = dp_table[sequence_A.length()][sequence_B.length()];
    return highest_alignment_score;
}

```

test.cpp

```cpp
// this file is for you for testing purposes, it will not be included in evaluation.

#include <iostream>
#include <fstream>
#include "the3.h"

void file_input(std::string& sequence_A, std::string& sequence_B, int& gap, int& mismatch, int& match){
    std::string file_name = "inp09.txt"; // inp01-inp10 are available.
    std::ifstream infile (file_name);
    if(!infile.is_open()){
        std::cout << "Input file cannot be opened" << std::endl;
        std::cout << "File name: " << file_name << std::endl;
        return;
    }
    infile >> sequence_A;
    infile >> sequence_B;
    infile >> gap;
    infile >> mismatch;
    infile >> match;
    return;
}

void test(){
    std::string sequence_A;
    std::string sequence_B;
    int gap, mismatch, match, highest_alignment_score_p1, highest_alignment_score_p2;
    std::string  possible_alignment_p1;
    std::string  possible_alignment_p2;
    int call_count = 0;

    file_input(sequence_A,sequence_B,gap,mismatch,match);
    std::cout << "Sequence A: " << sequence_A << std::endl <<
                 "Sequence B: " << sequence_B << std::endl <<
                 "gap: " << gap << std::endl <<
                 "mismatch: " << mismatch << std::endl <<
                 "match: " << match << std::endl;
    std::cout << "PART 1:" << std::endl;
    highest_alignment_score_p1 = recursive_alignment(sequence_A,sequence_B,gap,mismatch,match,possible_alignment_p1,call_count);
    std::cout << "Highest match score: " << highest_alignment_score_p1 << std::endl <<
                 "Possible alignment: " << std::endl;
    std::cout << possible_alignment_p1 << std::endl;

    std::cout << "PART 2:" << std::endl;
    highest_alignment_score_p2 = dp_table_alignment(sequence_A,sequence_B,gap,mismatch,match,possible_alignment_p2);
    std::cout << "Highest match score: " << highest_alignment_score_p2 << std::endl <<
                "Possible alignment: " << std::endl;
    std::cout << possible_alignment_p2 << std::endl;
    return;
}

int main(){
    test();
    return 0;
}

```

VPL