[CENG 315 ALL Sections] Algorithms

THE6

Available from: Saturday, December 16, 2023, 12:00 PM
Due date: Sunday, December 17, 2023, 11:59 PM
Requested files: the6.cpp, test.cpp, the6.h (Download)
Type of work: Individual work

You, an AI hater, recently learned that an AI model discovered new molecule structures equivalent to 800 years' worth of knowledge[1][2], and you are extremely annoyed by this news. While on an anger rant, you claim that you can write a basic program to find new optimal structures by yourself. Your chemist friends dare you to do so. They also offer to check the validity of your program, if you also write them another program that helps with their project.

You come up with an idea to find new molecule structures: If we have a graph where each vertex is an atom and each edge is the energy of the bond between the two atoms, maybe a new possible molecule structure can be found by selecting the edges with lowest energy bonds, without creating a cycle. Your chemist friends tell you this is most definitely a wrong approach, but you insist on trying.

Additionally, your chemist friends need a program to find the longest chain in a given molecule, so you agree to also write that program for them.

Problem

This exam consists of two parts, graded independently.

> PART1

You need to complete the function ***find_stucture()*** which **returns the total bond energy** of the found molecule:

```
int find_structure(std::vector< std::vector<std::pair<int,int> > > &bond_energies,
std::vector< std::vector<std::pair<int,int> > > &lowest_energy_structure);
```

- ***bond_energies***: a bidirectional graph represented by an adjacency list, where the vertices **(v)** are atoms, edges are bonds, and weights **(w)** of edges are the energies of the bonds. In other words, *an entry in the adjacency list for vertex v0 is <v1, w>, which represents an edge between v0-v1 with a weight of w.*
- ***lowest_energy_structure***: the found molecule structure represented as an adjacency list, with the same format as ***bond_energies***. You are expected to find the structure and assign it to this argument.
- *There is at most one bond with two atoms with only one weight value*, i.e. there is at most one undirected edge between any two vertices in the graph, meaning at most one weight value for each pair. *Undirectionality is shown in the adjacency list by adding two mirror entries for each edge* for ease of implementation.
- Vertices are represented as integers starting from 0, and the **maximum number of vertices in the graph is 1000**.
- Weights are represented as integers starting from 1, and the **maximum weight value is 100**.
- ***return value* is the total bond energy of the *lowest_energy_structure*.**

> PART2

You need to complete the function ***find_longest_chain()*** which **returns the number of the atoms in the longest chain** of the given molecule.

```
int find_longest_chain(std::vector< std::vector<std::pair<int,int> > > &molecule_structure,
std::vector<int> chain);
```

- ***molecule_structure***: a graph with the same representation and limits as the *bond_energies and lowest_energy_structure arguments of PART 1*. ***Additionally, it is guaranteed that the structure is a connected, acyclic, undirected graph, with all edges having the***

same weight (i.e. weights are not important). Undirectionality is shown in the adjacency list by adding two mirror entries for each edge for ease of implementation.

- ***chain*:** a vector of integers, where each integer maps to a vertex ID in the found longest chain. The vector should follow the order of the chain. The validity of the chain will be checked by the tester to see if consecutive vertices in the vector have edges between them or not.
- ***return value* is the *total number of atoms in the longest chain* of the molecule. The longest chain is defined as the count of vertices on the path between the two farthest vertices in the graph, including the start and end vertices.**

Constraints and Hints:

- When deciding between edges with the same weights connecting v to two different vertices *u_1* and *u_2*, **always favor the *u_i* with a smaller index**.
- Similarly, if you need to decide between two edges (*u1,u3,w1)* and *(u2,u4,w2)* where *w1=w2*, you should again **favor the edge with the smaller *u_i***, meaning (u1,u3,w1).
- There are **no self-loops**, meaning there is no edge such that *(u_i, u_j, w)* where *i=j*.

Evaluation:

- After your exam, black-box evaluation will be carried out. You will get full points if you return the correct order or strongly connected components. The grade you see in the VPL contains 50% of your final grade. We will evaluate your grades with different inputs after the end of the exam.

  Grade distribution is as follows:
- 30% part1 return value ***lowest_total_energy***
- 30% part1 ***lowest_energy_structure***
- 10% part2 return value ***longest_chain_size***
- 30% part2 ***chain***

Example IO:

1)

> PART 1

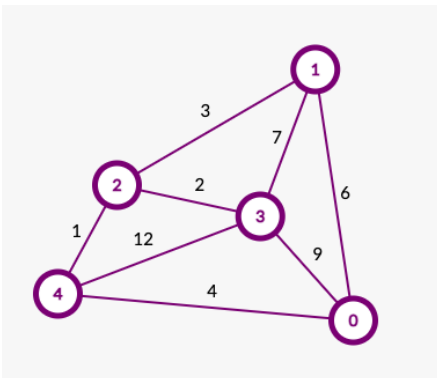

fig 1: bond_energies graph

bond_energies:

0	{ (1,6) (3,9) (4,4) }
1	{ (0,6) (2,3) (3,7) }
2	{ (1,3) (3,2) (4,1) }
3	{ (0,9) (1,7) (2,2) (4,12) }
4	{ (0,4) (2,1) (3,12) }

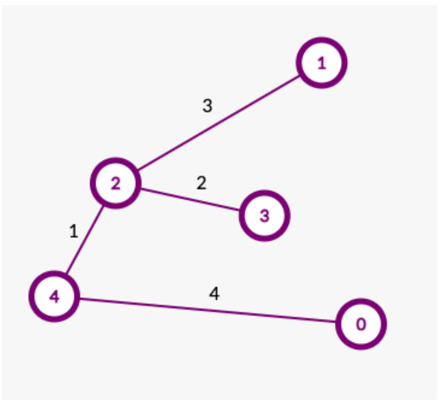

fig 2: lowest_energy_structure

lowest_energy_structure found:

0	{ (4,4) }
1	{ (2,3) }
2	{ (1,3) (4,1) (3,2) }
3	{ (2,2) }
4	{ (2,1) (0,4) }

return value ***lowest_total_energy***: 10

> PART 2

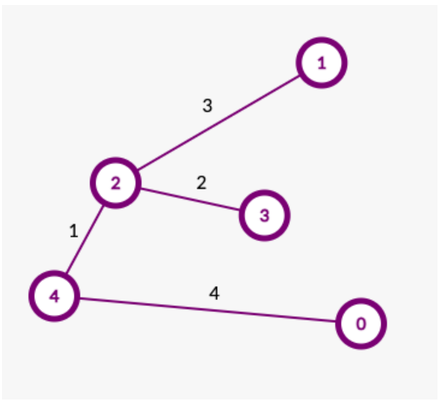

fig3: molecule_structure

note: *for the first i/o, input of part2 is the same as the output of part1.*
this is not guaranteed to be the case for all outputs. please check other i/o.

molecule_structure:

0	{ (4,4) }
1	{ (2,3) }
2	{ (1,3) (3,2) (4,1) }
3	{ (2,2) }
4	{ (0,4) (2,1) }

chain:
(3)-(2)-(4)-(0) and alternatively, (1)-(2)-(4)-(0)

return value ***longest_chain_size***: 4

Specifications:

- There are 2 tasks to be solved in **36 hours** in this take-home exam.
- You will implement your solutions in ***the6.cpp*** file.
- You are free to add other functions to *the6.cpp*
- **Do not change** the first line of the6.cpp, which is #include "the6.h"
- Some libraries are included in "the6.h" for your convenience, you can use them freely.
- **Do not change** the arguments and the return value of the functions ***find_stucture()*** and ***find_longest_chain()*** in the file the6.cpp
- **Do not include** any other library or write include anywhere in your the6.cpp file (not even in comments).
- You are given ***test.cpp*** file to test your work on **ODTUClass** or your **locale**. You can, and you are, encouraged to modify this file to add different test cases.
- If you want to test your work and see your outputs you can compile your work on your locale as:

```
>g++ test.cpp the6.cpp -Wall -std=c++11 -o test

> ./test
```

- You can test your the6.cpp on the virtual lab environment. If you click **run**, your function will be compiled and **executed with test.cpp**. If you click **evaluate**, you will get **feedback** for your current work, and your work will be **temporarily graded** for a limited number of inputs.
- The grade you see in lab is not your final grade, **your code will be reevaluated with different inputs** after the exam.

The system has the following limits **[NEEDS CHECK & CHANGE]:**

- a maximum execution time of 8 seconds (your program needs to return in less than a second per test case on average - each test case has 2 parts)
- a 1 GB maximum memory limit,
- an execution file size of 4M.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity, but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.

*not relevant to the question:
[1] https://deepmind.google/discover/blog/millions-of-new-materials-discovered-with-deep-learning/
[2] https://www.nature.com/articles/s41586-023-06735-9

Requested files

the6.cpp

```
#include "the6.h"

// do not add extra libraries here

int find_structure(std::vector< std::vector<std::pair<int,int> > >& bond_energies, std::vector< std::vector<std::pair<int,int> > >& lowest_energy
    int lowest_total_energy = 0;

    return lowest_total_energy;
}
int find_longest_chain(std::vector< std::vector<std::pair<int,int> > >& molecule_structure, std::vector<int>& chain){
    int longest_chain_size = 0;
    return longest_chain_size;
}

```

test.cpp

```cpp
#include <iostream>
#include <fstream>
#include "the6.h"


void print_adj_list(std::vector< std::vector< std::pair<int,int> > >& adj_list) {
    int N = adj_list.size();
    if (N == 0) {
        std::cout << "list is empty!" << std::endl;
        return;
    }

    for (int v=0;v<N;v++) {
        std::cout << v << "\t{";
        for (auto p : adj_list[v]) {
            std::cout << " (" << p.first << "," << p.second << ")";
        }
        std::cout << " }\n";
    }
    return;
}

// you can use this if you want to print the adj list as a matrix
void print_adj_list_as_matrix(std::vector< std::vector< std::pair<int,int> > >& adj_list) {
    int N = adj_list.size();
    if (N == 0) {
        std::cout << "list is empty!" << std::endl;
        return;
    }
    int** matrix;
    matrix = new int*[N];
    for(int temp=0; temp < N; temp++) matrix[temp] = new int[N];
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            matrix[i][j] = -1; // no edge
        }
    }

    for (int i=0; i<N; i++){
        for (std::pair<int,int> x: adj_list[i]) {
            matrix[i][x.first] = x.second;
        }
    }

    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            if (matrix[i][j] == -1) std::cout << "- ";
            else std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    for(int i=0; i<N; i++) delete[] matrix[i];
    delete[] matrix;
    return;
}

void read_from_file(std::vector< std::vector<std::pair<int,int> > >& bond_energies, std::vector< std::vector<std::pair<int,int> > >& molecule_st
    char addr[]= "inp01.txt"; // 01-05 are available
    std::ifstream infile (addr);
    if (!infile.is_open()){
        std::cout << "File \'"<< addr
                  << "\' can not be opened. Make sure that this file exists." << std::endl;
        return;
    }

    int V_p1, E_p1, V_p2, E_p2;

    infile >> V_p1 >> E_p1;
    bond_energies.resize(V_p1);
    for (int l=0; l<E_p1; l++) {
        int v1, v2, w;
        infile >> v1 >> v2 >> w;
        bond_energies[v1].push_back(std::make_pair(v2,w));
        bond_energies[v2].push_back(std::make_pair(v1,w));
    }

    infile >> V_p2 >> E_p2;
    molecule_structure.resize(V_p2);
    for (int l=0; l<E_p2; l++) {
        int v1, v2, w;
        infile >> v1 >> v2 >> w;
        molecule_structure[v1].push_back(std::make_pair(v2,w));
        molecule_structure[v2].push_back(std::make_pair(v1,w));
    }


    infile.close();
}

int main(){
    std::vector< std::vector< std::pair<int,int> > > bond_energies;
    std::vector< std::vector< std::pair<int,int> > > molecule_structure;
    std::vector< std::vector< std::pair<int,int> > > lowest_energy_structure;
    std::vector<int> chain;
    int longest_chain_size, lowest_total_energy;

    read_from_file(bond_energies, molecule_structure);
    lowest_energy_structure.resize(bond_energies.size());

    lowest_total_energy = find_structure(bond_energies, lowest_energy_structure);
    std::cout << "PART 1: " << std::endl << "Bond energy graph:" << std::endl;
    print_adj_list(bond_energies);
    std::cout << "Graph of the lowest energy structure found:" << std::endl;
    print_adj_list(lowest_energy_structure);
    std::cout << "Total energy of the lowest energy structure: " << lowest_total_energy << std::endl;

    longest_chain_size = find_longest_chain(molecule_structure,chain);
    std::cout << "PART 2: " << std::endl << "Molecule structure graph:" << std::endl;
    print_adj_list(molecule_structure);
    std::cout << "Atom count in longest chain: " << longest_chain_size << std::endl;
    std::cout << "Longest chain:" << std::endl;
    std::cout << "(" << chain[0] << ")";
    for (int i=1; i<chain.size(); i++) {
        std::cout << " - (" << chain[i] << ")";
    }

    return 0;
}
```

the6.h

```
#ifndef THE6_THE6_H
#define THE6_THE6_H
#include <vector>
#include <utility>
#include <queue>
#include <stack>
#include <climits>
#include <algorithm>

//updating this file will not change the execution in the VPL

int find_structure(std::vector< std::vector<std::pair<int,int> > > &bond_energies, std::vector< std::vector<std::pair<int,int> > > &lowest_energy

int find_longest_chain(std::vector< std::vector<std::pair<int,int> > > &molecule_structure, std::vector<int>& chain);


#endif //THE6_THE6_H
```

VPL