[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
        - November 20 - November 26
        - November 27 - December 3
        - December 3 - December 10
          - THE6
            - Description
            - Submission view
        - 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 Saturday, December 16, 2023, 5:51 PM by Automatic grade
Grade: 100.00 / 100.00

Assessment report [-]
[-]**Output of make**
g++ -c -o sol6.o sol6.cpp
[-]**For input 01:**
lowest_total_energy: correct
lowest_energy_structure: correct
chain_size: correct
longest_atom_chain: correct
[-]**For input 02:**
lowest_total_energy: correct
lowest_energy_structure: correct
chain_size: correct
longest_atom_chain: correct
[-]**For input 03:**
lowest_total_energy: correct
lowest_energy_structure: correct
chain_size: correct
longest_atom_chain: correct
[-]**For input 04:**
lowest_total_energy: correct
lowest_energy_structure: correct
chain_size: correct
longest_atom_chain: correct
[-]**For input 05:**
lowest_total_energy: correct
lowest_energy_structure: correct
chain_size: correct
longest_atom_chain: correct

Submitted on Saturday, December 16, 2023, 5:51 PM (Download)

the6.cpp

```cpp
#include "the6.h"

// do not add extra libraries here
#define INF 1000000000

int find_structure(std::vector< std::vector<std::pair<int,int> > >& bond_energies, std::vector< std::vector<std::pair<int,int> > >& lowest_energy_structure)
    int num_nodes = bond_energies.size();
    lowest_energy_structure.clear();
    lowest_energy_structure.resize(num_nodes);

    std::vector<int> key(num_nodes, INF);
    std::vector<int> parent(num_nodes, -1);
    std::vector<bool> inMST(num_nodes, false);

    std::priority_queue< std::pair<int, int>, std::vector <std::pair<int, int>> , std::greater<std::pair<int, int>> > pq;

    key[0] = 0;
    pq.push(std::make_pair(0, 0));

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        inMST[u] = true;

        for (auto x : bond_energies[u]) {
            int v = x.first;
            int weight = x.second;

            if (inMST[v] == false && key[v] > weight) {
                key[v] = weight;
                pq.push(std::make_pair(key[v], v));
                parent[v] = u;
            }
        }
    }

    int total_cost = 0;
    for (int i = 1; i < num_nodes; ++i) {
        lowest_energy_structure[parent[i]].push_back(std::make_pair(i, key[i]));
        lowest_energy_structure[i].push_back(std::make_pair(parent[i], key[i]));
        total_cost += key[i];
    }

    return total_cost;
}

void dfs(int node, int depth, int& max_depth, int& max_node, std::vector< std::vector<std::pair<int,int> > > &adjacency_list_in, std::vector<bool>& visited)
    visited[node] = true;
    if (depth > max_depth)
        {
```

```
        max_depth = depth;
        max_node = node;
    }
    for (auto& neighbour : adjacency_list_in[node])
    {
        if (!visited[neighbour.first])
            {
            dfs(neighbour.first, depth + 1, max_depth, max_node, adjacency_list_in, visited);
        }
    }
}

int find_longest_chain(std::vector< std::vector<std::pair<int,int> > >& molecule_structure, std::vector<int>& chain){
    int n = molecule_structure.size();
    std::vector<bool> visited(n, false);
    int max_depth = -1;
    int max_node = -1;
    dfs(0, 0, max_depth, max_node, molecule_structure, visited);
    std::fill(visited.begin(), visited.end(), false);
    max_depth = -1;
    dfs(max_node, 0, max_depth, max_node, molecule_structure, visited);
    chain.resize(max_depth + 1);
    chain[max_depth] = max_node;
    for (int i = max_depth - 1; i >= 0; --i) {
        for (auto& neighbour : molecule_structure[max_node])
        {
            if (visited[neighbour.first])
            {
                chain[i] = neighbour.first;
                max_node = neighbour.first;
                break;
            }
        }
    }
    return chain.size();
}

```

test.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_structure){
    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_structure)

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


#endif //THE6_THE6_H
```

VPL