```cpp
//
// Created by Nikolay Yakovets on 2018-02-01.
//

#include "SimpleGraph.h"
#include "SimpleEstimator.h"
#include <list>
#include <cmath>


SimpleEstimator::SimpleEstimator(std::shared_ptr<SimpleGraph> &g){
    graph = g;
}



void SimpleEstimator::prepare() {

    label_count.resize(graph->getNoLabels());
    start_end_set_counts.resize(graph->getNoLabels());

    // count label freq.
    for (int i = 0; i < graph->getNoVertices(); i++) {
        for (auto labelTarget : graph->adj[i]) {
            auto label = labelTarget.first;
            label_count[label]++;
        }
    }


    start_end_set_counts = compute_in_end_counts();

}


cardStat SimpleEstimator::estimate(RPQTree *q) {

    std::vector<std::pair<uint32_t,bool>> query_list;
    reduceQuery(q, query_list);

    // perform your estimation here
    uint32_t length = (uint32_t)query_list.size();
    if(length == 0) {
        return cardStat{0,0,0};
    }

    bool inverse_start = query_list[0].second;
    uint32_t label_start = query_list[0].first;

    double Vry;
    if(!inverse_start) {
        Vry = start_end_set_counts[label_start].second;
    } else {
        Vry = start_end_set_counts[label_start].first;
    }

    double Tr = label_count[label_start];

    if(length == 1) {
        return cardStat{0, (uint32_t) label_count[label_start],0};
    }

    for(int i = 1; i < length; i++) {
        double Vsy;
        double Ts = label_count[query_list[i].first];
        if(query_list[i].second) {
            Vsy = start_end_set_counts[query_list[i].first].second;
        } else {
            Vsy = start_end_set_counts[query_list[i].first].first;
        }
        double join_size = std::min(Tr*(Ts/Vsy),Ts*(Tr/Vry));
        if(query_list[i].second) {
            Vry = start_end_set_counts[query_list[i].first].first;
        } else {
            Vry = start_end_set_counts[query_list[i].first].second;
        }
        Vry = std::min(Vry,join_size);
        Tr = join_size;
    }
    return cardStat{ 0, (uint32_t) Tr, 0};
}

void SimpleEstimator::reduceQuery(RPQTree *q, std::vector<std::pair<uint32_t, bool>> &parsedQuery) {
    if (q->isLeaf()) {
        parsedQuery.emplace_back(std::make_pair((uint32_t) std::stol(q->data.substr(0, q->data.size()-1)),q->data.substr(q->data.size()-1
    }
    else if (q->isConcat()){
        reduceQuery(q->left, parsedQuery);
        reduceQuery(q->right, parsedQuery);
    }
}

std::vector<std::pair<uint32_t , uint32_t >> SimpleEstimator::compute_in_end_counts(){
    std::vector<std::pair<uint32_t, uint32_t >> res;
    res.resize(graph->getNoLabels());
//    std::unordered_set<uint32_t > inSet;
//    std::unordered_set<uint32_t > outSet;
    std::unordered_set<uint32_t > cSet;
    for(uint32_t i = 0; i < graph->getNoLabels(); i++){
        for(uint32_t j = 0; j < graph->getNoVertices(); j++) {
            for (auto edge: graph->adj[j]) {
```

```
                if (edge.first == i) {
//                    outSet.insert(edge.second);
//                    inSet.insert(j);
                    cSet.insert(edge.second);
                }
            }
        }
//        res[i].first = (uint32_t)inSet.size();
//        inSet.clear();
        res[i].first = (uint32_t) cSet.size();
        cSet.clear();
        for(uint32_t j = 0; j < graph->getNoVertices(); j++) {
            for (auto edge: graph->adj[j]) {
                if (edge.first == i) {
//                    outSet.insert(edge.second);
//                    inSet.insert(j);
                    cSet.insert(j);
                }
            }
        }
//        res[i].second = (uint32_t)outSet.size();
//        outSet.clear();
        res[i].second = (uint32_t) cSet.size();
        cSet.clear();
    }
    return res;
}
```