به نام خدا

گزارش تمرین سری ششم AP

محمدمهدی مالوردی

9723079

استاد جهانشاهی

برنامه های پیاده سازی شده در qn.h به صورت زیر می باشد:

include > C q1.h

```cpp
#ifndef Q1_H
#define Q1_H

#include <iostream>
#include <cmath>
#include <functional>



namespace q1 {

    template<typename T = double, typename F = std::function<T(T)>>
    inline T gradient_descent(T initial_value, T step_size, F gradient_func = F()) {
        T limit{step_size / 50};
        size_t Max_lim{1000};
        size_t Counter{0};
        T slope{1.0};
        T currentP{initial_value};
        T nextP{currentP + step_size};
        int dir{0};
        if(gradient_func(nextP) - gradient_func(currentP) < 0){
            dir = 1;
        }
        else{
            dir = -1;
        }
        while ((Counter < Max_lim) && (slope > limit)) {
            nextP = currentP + dir * step_size;
            T currentOut{gradient_func(currentP)};
            T nextOut = gradient_func(nextP);
            T gradient{(currentOut - nextOut) / step_size};
            slope = pow(gradient, 2);
            currentP = nextP;
            Counter++;
        }

        return currentP;
    }

}

#endif //Q1_H
```

پیاده سازی سوال اول

برای پیاده سازی سوال اول ابتدا الگوریتم gradient descent را به صورت خیلی ساده پیاده سازی کردیم. نهایت تکرار الگوریتم را 1000 و عدد همگرایی را برابر step_size/50 قرار می دهیم. باید حواسمان به جهت حرکت نیز باشد که پایین ترین نقطه رو پیدا کنیم، برای همین متغیری به نام dir تعریف کردیم

که یا +1 است یا -1 و این را حاصل تفریق تابع در دو نقطه پشت هم مطابق کد نشان می دهد. همچنین شیب معادله را هم از مقدار 1 شروع کرده و طبق فرمول داخل کد محاسبه می کنیم.

نکته خیلی مهم در همه سوالات این است که چون include<qn> ها در چند فایل استفاده شده، باید اول هر تابع از کلمه کلیدی inline استفاده کنیم که به تکرار این ها ارور نگیرد.

همچنین باید کد را برای همه توابع قابل استفاده کنیم که طبق STL :

```
template<typename T = double, typename F = std::function<T(T)>>
    inline T gradient_descent(T initial_value, T step_size, F gradient_func =
F())
```

از دو خط بالا استفاده می کنیم.

include > C q2.h

```cpp
#ifndef Q2_H
#define Q2_H

#include <fstream>
#include <regex>
#include <iostream>

namespace q2 {
    struct Patient {
        std::string name;
        size_t age{};
        size_t smokes{};
        size_t area_q{};
        size_t alkhol{};
    };
    inline std::string left_trim(const std::string &str) {
        return std::regex_replace(str, std::regex("^\\s+"), std::string(""));
    }
    inline std::string right_trim(const std::string &str) {
        return std::regex_replace(str, std::regex("\\s+$"), std::string(""));
    }
    inline std::string trim(const std::string &str) {
        return left_trim(right_trim(str));
    }


    inline std::vector<Patient> read_file(const std::string &filename) {

        std::fstream stream("../resources/" + filename);
        std::string line;
        stream >> line >> line;
        std::vector<std::string> elements;
        std::vector<Patient> patients;

        while (stream >> line) {
            if (elements.size() == 6)
                elements.clear();
            std::string element{};
            for (auto data: line) {
                if (data == ',') {
                    if (!trim(element).empty()){
                        elements.push_back(trim(element));
```

```
                    elements.push_back(trim(element));
                }
                element = "";
            }
            else {
                element += data;
            }
        }
        if (!trim(element).empty()){
            elements.push_back(trim(element));
        }
        if (elements.size() < 6){
            continue;
        }
        Patient pat = {
                elements[0] + " " + elements[1],
                static_cast<size_t>(std::stoi(elements[2])),
                static_cast<size_t>(std::stoi(elements[3])),
                static_cast<size_t>(std::stoi(elements[4])),
                static_cast<size_t>(std::stoi(elements[5])),
        };
        patients.push_back(pat);
    }

    return patients;
}

inline void sort(std::vector<Patient> &pat) {
    std::ranges::sort(pat, std::greater<>{},
                      [](const Patient &person) {
                          return (3 * person.age + 5 * person.smokes + 2 * person.area_q + 4 * person.alkhol);
                      });
}
}
#endif //Q2_H
```

پیاده سازی سوال دوم

در این تابع به وسیله توابع trim، اسپیس های احتمالی دو طرف داده های را از بین می بریم. سپس دو خط اول که مهم نیست را رد می کنیم. داده ها را متناسب با این که بینشان ، است جداسازی می کنیم و پس از ریختن آن ها درون elements به کمک static_cast پارامترهایمان را درون pat می ریزیم. همچنین دو پارامتر string اول چون مربوط به name هستند به هم می چسبانیم.

در نهایت به کمک std::ranges::sort مرتب سازی گفته شده را انجام می دهیم.

include > C q3.h

```cpp
#ifndef Q3_H
#define Q3_H

#include <queue>

namespace q3 {
    struct Flight {
        std::string flight_number;
        size_t duration{};
        size_t connections{};
        size_t connection_minutess{};
        size_t price{};
    };

    inline auto cmp_func = [](const Flight &f1, const Flight &f2) {
        return f1.duration + f1.connection_minutess + 3 * f1.price > f2.duration + f2.connection_minutess + 3 * f2.price;
    };
    inline std::string left_trim(const std::string &str) {
        return std::regex_replace(str, std::regex("^\\s+"), std::string(""));
    }
    inline std::string right_trim(const std::string &str) {
        return std::regex_replace(str, std::regex("\\s+$"), std::string(""));
    }
    inline std::string trim(const std::string &str) {
        return left_trim(right_trim(str));
    }


    inline std::priority_queue<Flight, std::vector<Flight>, decltype(cmp_func)> gather_flights(const std::string &filename) {

        std::fstream stream("../resources/" + filename);
        std::string line;

        std::vector<std::string> elements;
        std::priority_queue<Flight, std::vector<Flight>, decltype(cmp_func)> flights(cmp_func);
        while (stream >> line) {
            if (elements.size() == 6){
                elements.clear();
            }
            std::string element{};
            size_t data{};
            while (data < line.size()) {
```

```
        while (data < line.size()) {
            if (line[data] == '-') {
                if (!trim(element).empty()){
                    elements.push_back(trim(element));
                }
                element = "";
            }
             else {
                element += line[data];
            }
            data++;
        }
        if (!trim(element).empty()){
            elements.push_back(trim(element));
        }
        if (elements.size() < 6){
            continue;
        }

        size_t sect{};
        while (sect < elements.size()) {
            size_t minutes{};
            std::string str{};
            size_t letter{elements[sect].find(':') + 1};
            while (letter < elements[sect].size()) {
                if (elements[sect][letter] == 'h') {
                    minutes += 60 * std::stoi(str);
                    str = {};
                }
                else if (elements[sect][letter] == 'm') {
                    minutes += std::stoi(str);
                    str = {};
                }
                 else if (elements[sect][letter] == ',') {
                    str = {};
                }
                 else{
                    str += elements[sect][letter];
```

```
                    str += elements[sect][letter];
                }
                letter++;
            }
            if (str.empty()){
                elements[sect] = std::to_string(minutes);
            }
            else{
                elements[sect] = str;
            }
            sect++;
        }

        Flight flight = {
                elements[1],
                static_cast<size_t>(std::stoi(elements[2])),
                static_cast<size_t>(std::stoi(elements[3])),
                static_cast<size_t>(std::stoi(elements[4])),
                static_cast<size_t>(std::stoi(elements[5])),
        };
        flights.push(flight);
    }

    return flights;
}

}


#endif //Q3_H
```

پیاده سازی سوال سوم

پیاده سازی این سوال شباهت زیادی به سوال قبلی دارد. برای مقایسه پروازها تابع زیر را پیاده سازی می کنیم:

```
inline auto cmp_func = [](const Flight &f1, const Flight &f2) {
        return f1.duration + f1.connection_minutess + 3 * f1.price > f2.duration
+ f2.connection_minutess + 3 * f2.price;
    };
```

در این تابع، مرتب سازی را به کمک while های تو در تو و بدون for انجام می دهیم. همچنین باید دقت کنیم که همه زمان ها باید بر مبنای دقیقه باشند، برای همین همه ساعت ها را به دقیقه تبدیل کردیم.

include > C q4.h

```cpp
#ifndef Q4_H
#define Q4_H

#include <functional>
#include <numeric>

namespace q4 {
    struct Vector2D {
        double x{};
        double y{};
    };

    struct Sensor {
        Vector2D pos;
        double accuracy{};
    };


    inline Vector2D kalman_filter(std::vector<Sensor> sensors) {
        double accumulate_accuaracies{};
        double approximation_of_x{};
        double approximation_of_y{};
        Sensor sns{std::accumulate(sensors.begin(), sensors.end(), Sensor{{0, 0}, 0},
                        [&accumulate_accuaracies](const Sensor &first, const Sensor &second) -> Sensor {
                            accumulate_accuaracies += first.accuracy + second.accuracy;
                            return {accumulate_accuaracies};
                        })};



        Sensor kalman{std::accumulate(sensors.begin(), sensors.end(), Sensor{{0, 0}, 0},
                                [accumulate_accuaracies, &approximation_of_x, &approximation_of_y](const Sensor &first, const Sensor &second) -> Sensor {
                                    approximation_of_x += second.pos.x * (second.accuracy / accumulate_accuaracies);
                                    approximation_of_y += second.pos.y * (second.accuracy / accumulate_accuaracies);
                                    return {{approximation_of_x, approximation_of_y}};
                                })};

        return kalman.pos;
    }
}

#endif //Q4_H
```

پیاده سازی سوال چهارم

برای پیاده سازی این سوال از std::accumulate استفاده می کنیم و آن را مطابق شکل به کار می بریم. به کمک آن به راحتی می توان دقت ها را جمع و محاسبه کرد و بهترین تقریب برای موقعیت خواسته شده را به دست آورد.

لینک گیت هاب:

https://github.com/mahdimld/AP-HW06