Фінальний звіт

Підсумовування результатів автоматизованого процесу генерації, тестування та оптимізації C++ алгоритму для виявлення та корекції аномальних GPS-координат. Процес виконувався за допомогою Python-скрипта, що взаємодіяв з моделлю ШІ(ChatGPT) для послідовного покращення алгоритму протягом 15 успішних ітерацій.

Статистика ітерацій

|  |  |  |  |
| --- | --- | --- | --- |
| Ітерація № | Статус успіху | Час виконання | Статус компіляції |
| 1 | Успіх | 0.3476 сек. | Успіх |
| 2 | Збій | - | Збій |
| 3 | Успіх | 0.0422 сек. | Успіх |
| 4 | Успіх | 0.0369 сек. | Успіх |
| 5 | Успіх | 0.0405 сек. | Успіх |
| 6 | Збій | - | Збій |
| 7 | Збій | - | Збій |
| 8 | Успіх | 0.0384 сек. | Успіх |
| 9 | Збій | - | Збій |
| 10 | Успіх | 0.0351 сек. | Успіх |
| 11 | Успіх | 0.0360 сек. | Успіх |
| 12 | Успіх | 0.0376 сек. | Успіх |
| 13 | Збій | - | Збій |
| 14 | Успіх | 0.0349 сек. | Успіх |
| 15 | Збій | - | Збій |
| 16 | Успіх | 0.0360 сек. | Успіх |
| 17 | Успіх | 0.0363 сек. | Успіх |
| 18 | Успіх | 0.0359 сек. | Успіх |
| 19 | Успіх | 0.0350 сек. | Успіх |
| 20 | Успіх | 0.0359 сек. | Успіх |
| 21 | Збій | - | Збій |
| 22 | Збій | - | Успіх |
| 23 | Успіх | 0.0381 сек. | Успіх |
| 24 | Збій | - | Успіх |
| 25 | Збій | - | Успіх |
| 26 | Успіх | 0.0347 сек. | Успіх |
| 27 | Успіх | 0.0357 сек. | Успіх |

Найбільш ефективний алгоритм(Ітерація №26) мав середню швидкість виконання 0.0347 сек.

Алгоритм ітерації №26

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iostream>

#include <vector>

#include <nlohmann/json.hpp>

using json = nlohmann::json;

struct Coordinate {

double lat;

double lon;

long long time;

};

void from\_json(const json& j, Coordinate& c) {

j.at("lat").get\_to(c.lat);

j.at("lon").get\_to(c.lon);

j.at("time").get\_to(c.time);

}

void to\_json(json& j, const Coordinate& c) {

j = json{ {"lat", c.lat}, {"lon", c.lon}, {"time", c.time} };

}

double haversine\_distance(double lat1, double lon1, double lat2, double lon2) {

constexpr double R = 6371000.0; // Earth radius in meters

double phi1 = lat1 \* M\_PI / 180.0;

double phi2 = lat2 \* M\_PI / 180.0;

double delta\_phi = (lat2 - lat1) \* M\_PI / 180.0;

double delta\_lambda = (lon2 - lon1) \* M\_PI / 180.0;

double a = sin(delta\_phi/2) \* sin(delta\_phi/2) +

cos(phi1) \* cos(phi2) \*

sin(delta\_lambda/2) \* sin(delta\_lambda/2);

double c = 2 \* atan2(sqrt(a), sqrt(1-a));

return R \* c;

}

json process\_coordinates(const json& input) {

std::vector<Coordinate> coordinates = input.get<std::vector<Coordinate>>();

json output;

std::vector<Coordinate> corrected\_points;

int anomalies\_detected = 0;

int anomalies\_corrected = 0;

for (size\_t i = 0; i < coordinates.size(); ++i) {

if (i > 0 && i < coordinates.size() - 1) {

double distance\_prev = haversine\_distance(coordinates[i].lat, coordinates[i].lon, coordinates[i - 1].lat, coordinates[i - 1].lon);

double distance\_next = haversine\_distance(coordinates[i].lat, coordinates[i].lon, coordinates[i + 1].lat, coordinates[i + 1].lon);

double speed\_prev = distance\_prev / (coordinates[i].time - coordinates[i - 1].time);

double speed\_next = distance\_next / (coordinates[i + 1].time - coordinates[i].time);

if (speed\_prev > 200.0 || speed\_next > 200.0) {

anomalies\_detected++;

// Linear interpolation if previous and next points are valid

Coordinate interpolated\_point;

interpolated\_point.lat = (coordinates[i - 1].lat + coordinates[i + 1].lat) / 2;

interpolated\_point.lon = (coordinates[i - 1].lon + coordinates[i + 1].lon) / 2;

interpolated\_point.time = coordinates[i].time;

corrected\_points.push\_back(interpolated\_point);

anomalies\_corrected++;

} else {

corrected\_points.push\_back(coordinates[i]);

}

} else {

corrected\_points.push\_back(coordinates[i]);

}

}

output["corrected\_points"] = corrected\_points;

output["anomalies\_detected"] = anomalies\_detected;

output["anomalies\_corrected"] = anomalies\_corrected;

return output;

}

int main() {

// Example usage

json input = R"([

{"lat": 49588396, "lon": 34569212, "time": 1746025730},

{"lat": 49588400, "lon": 34569220, "time": 1746025740},

{"lat": 49588410, "lon": 34569230, "time": 1746025750}

])"\_json;

json output = process\_coordinates(input);

std::cout << output.dump(4) << std::endl;

return 0;

}

Висновок

Завдяки ітеративному процесу взаємодії з ШІ, вдалося досягти помітного покращення продуктивності C++ алгоритму. Середній час виконання найкращого алгоритму (0.0347 сек.) значно нижчий, ніж результати першої ітерації (0.3476 сек.), що свідчить про значну ефективність застосованого підходу автоматизованої оптимізації.