Báo cáo Đề án Tối ưu Tuyến đường Giao hàng (TSP)

Sinh viên: Vòng Lý NàmNàm Phúc
MSSV: 2302700392
Lớp: 1488

Ngày 14 tháng 7 năm 2025

Giới thiệu bài toán

Bài toán Travelling Salesman Problem (TSP) là một bài toán kinh điển trong trí tuệ nhân tạo và tối ưu hóa, yêu cầu tìm tuyến đường ngắn nhất để một người giao hàng xuất phát từ kho, đi qua tất cả các điểm giao hàng đúng một lần và quay lại kho. Trong đề án này, em sử dụng ngôn ngữ **C++** để xây dựng chương trình giải TSP với các thuật toán heuristic (**Greedy**, **2-opt**, **Simulated Annealing**), tích hợp **ràng buộc thời gian** (cửa sổ thời gian giao hàng) và **chi phí giao hàng**. Ngoài ra, em sử dụng **Python** với thư viện **Folium** để trực quan hóa tuyến đường trên bản đồ. Dữ liệu tọa độ được lấy từ **OpenStreetMap API** hoặc file `inputdoan2.txt`, đảm bảo tính thực tế.

1. Giải thích chi tiết chương trình C++ (`doan2.cpp`)

Chương trình `doan2.cpp` được viết bằng C++ để giải bài toán TSP với các yêu cầu sau:

- **Yêu cầu tối thiểu**: Nhập tọa độ kho và ít nhất 5–10 điểm giao hàng, tính tuyến đường tối ưu với khoảng cách Euclidean nhỏ nhất, xuất thứ tự điểm và tổng khoảng cách.
- **Yêu cầu nâng cao**: Sử dụng dữ liệu thực tế từ OpenStreetMap, thêm ràng buộc thời gian/chi phí, trực quan hóa tuyến đường, và sử dụng các thuật toán heuristic nâng cao.

Dưới đây là giải thích chi tiết từng phần của mã C++:

1.1 Các thư viện và cấu trúc dữ liệu

```
1 // T i   u   t u y n     ng     giao h ng cho 63  t nh   V i t
    Nam  v i  r ng  b u c  t h i  gian v  chi ph
2 #include <windows.h>
3 #include <iostream>
4 #include <vector>
5 #include <cmath>
```

```cpp
#include <fstream>
#include <sstream>
#include <climits>
#include <algorithm>
#include <ctime>
#include <random>
#include <curl/curl.h>

using namespace std;

// Cấu trúc điểm giao hàng
struct Point {
    double x, y; // Tọa độ (kinh độ, vĩ độ)
    string name; // Tên điểm
    double start_time; // Bắt đầu của sổ thời gian (giờ, ví dụ: 8.0 là 8h sáng)
    double end_time;   // Kết thúc của sổ thời gian (giờ, ví dụ: 12.0 là 12h trưa)
};

// Cấu trúc thông tin tuyến đường
struct RouteInfo {
    vector<int> route; // Tuyến đường
    double total_distance; // Tổng không chích
    double total_time; // Tổng thời gian giao hàng
    double total_cost; // Tổng chi phí giao hàng
    bool time_feasible; // Tuyến đường có khả thi về thời gian
};
```

Giải thích:

- **Thư viện**:
  - `<windows.h>`: Hỗ trợ lệnh `system("pause")` để tạm dừng console trên Windows.
  - `<iostream>`: Nhập/xuất dữ liệu (ví dụ: `cout`, `cin`).
  - `<vector>`: Lưu danh sách các điểm và tuyến đường.
  - `<cmath>`: Tính toán khoảng cách Euclidean (`sqrt`, `pow`).
  - `<fstream>`, `<sstream>`: Đọc/ghi file và xử lý chuỗi.
  - `<climits>`: Sử dụng `INT_MAX` để tìm khoảng cách nhỏ nhất.
  - `<algorithm>`: Hàm `reverse` cho 2-opt.
  - `<ctime>`, `<random>`: Tạo số ngẫu nhiên cho Simulated Annealing.
  - `<curl/curl.h>`: Gọi API OpenStreetMap để lấy tọa độ.
- **Cấu trúc** `Point`:
  - `x`, `y`: Kinh độ và vĩ độ (ví dụ: Hà Nội có tọa độ `(105.8542, 21.0285)`).
  - `name`: Tên tỉnh (ví dụ: "Hà Nội").

- `start_time`, `end_time`: Cửa sổ thời gian giao hàng (ví dụ: từ 8h sáng đến 12h trưa).

- **Cấu trúc** `RouteInfo`:
  - `route`: Danh sách chỉ số các điểm theo thứ tự tuyến đường (ví dụ: `[0, 1, 2, 0]` cho Hà Nội → TP. Hồ Chí Minh → Đà Nẵng → Hà Nội).
  - `total_distance`: Tổng khoảng cách Euclidean (độ).
  - `total_time`: Tổng thời gian di chuyển (giờ).
  - `total_cost`: Tổng chi phí giao hàng (đơn vị).
  - `time_feasible`: Kiểm tra xem tuyến đường có thỏa mãn cửa sổ thời gian không.

1.2 Hàm nhập dữ liệu từ file

```
// H m    c     d    l i u   t   file
vector<pair<string, pair<double, double>>>
   read_inputs_from_file(const string& filename) {
    vector<pair<string, pair<double, double>>> inputs;
    ifstream in(filename);
    string line;
    while (getline(in, line)) {
        istringstream iss(line);
        string name;
        double lat, lon;
        iss >> ws;
        getline(iss, name, ' ');
        iss >> lat >> lon;
        name.erase(0, name.find_first_not_of(" \t"));
        name.erase(name.find_last_not_of(" \t") + 1);
        inputs.push_back({name, {lat, lon}});
    }
    return inputs;
}
```

Giải thích:

- **Mục đích**: Đọc dữ liệu từ file `inputdoan2.txt`, chứa thông tin về tên tỉnh, tọa độ, và cửa sổ thời gian.

- **Cách hoạt động**:
  - Mở file bằng `ifstream`.
  - Đọc từng dòng, phân tách tên tỉnh (có thể chứa dấu cách, như "Hà Nội") và tọa độ (`lat`, `lon`).
  - Lưu vào `inputs` dưới dạng `pair<string, pair<double, double>>`, với tên tỉnh và tọa độ.
  - Xóa khoảng trắng thừa trong tên tỉnh để đảm bảo định dạng sạch.

- **Ví dụ**: Với dòng `Hà Nội 21.0285 105.8542 8.0 12.0`, hàm lưu `{"Hà Nội", {21.0285, 105.8542}}`.

1.3 Hàm lấy tọa độ từ OpenStreetMap

```
// Lấy tọa độ và các mốc thời gian từ file hoặc API OpenStreetMap
vector<Point> fetch_coordinates_from_osm(vector<pair<string, pair<double, double>>> inputs) {
    vector<Point> points;
    ifstream inFile("inputdoan2.txt");
    unordered_map<string, tuple<double, double, double, double>> file_data;
    string line;

    if (inFile.is_open()) {
        while (getline(inFile, line)) {
            istringstream iss(line);
            string name;
            double lat, lon, start_time, end_time;
            iss >> ws;
            getline(iss, name, ' ');
            iss >> lat >> lon >> start_time >> end_time;
            name.erase(0, name.find_first_not_of(" \t"));
            name.erase(name.find_last_not_of(" \t") + 1);
            file_data[name] = {lat, lon, start_time, end_time};
        }
        inFile.close();
        cout << "Đã đọc dữ liệu từ inputdoan2.txt\n";
    } else {
        cerr << "Cảnh báo: Không thể mở inputdoan2.txt. Sử dụng API OpenStreetMap.\n";
    }

    CURL* curl;
    CURLcode res;
    string readBuffer;

    curl_global_init(CURL_GLOBAL_DEFAULT);
    curl = curl_easy_init();
    if (curl) {
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        curl_easy_setopt(curl, CURLOPT_USERAGENT, "TSPDemo/1.0");
        curl_easy_setopt(curl, CURLOPT_CAINFO, "C:/curl-8.14.1_2-win64-mingw/cacert.pem");

        for (size_t i = 0; i < inputs.size(); ++i) {
            const auto& input = inputs[i];
            string province = input.first;
            auto it = file_data.find(province);
            if (it != file_data.end()) {
```

```
43                  auto [lat, lon, start_time, end_time] =
                        it->second;
44                  points.push_back({lon, lat, province,
                        start_time, end_time});
45                  cout << "   c     t   file: " << province << " =>
                        (" << lon << ", " << lat << ", " <<
                        start_time << ", " << end_time << ")\n";
46              } else {
47                  string url;
48                  if (input.first.empty()) {
49                      url =
                            "https://nominatim.openstreetmap.org/reverse?format=j
                            +
50                              to_string(input.second.first) +
                                "&lon=" +
                                to_string(input.second.second) +
51                            "&zoom=18&addressdetails=1";
52                  } else {
53                      char* escaped = curl_easy_escape(curl,
                            input.first.c_str(), 0);
54                      url =
                            "https://nominatim.openstreetmap.org/search?q="
                            + string(escaped) +
                            ",+Vietnam&format=json";
55                      curl_free(escaped);
56                  }
57
58                  curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
59                  res = curl_easy_perform(curl);
60                  if (res != CURLE_OK) {
61                      cerr << " L i  khi  l y   d    l i u  " <<
                            input.first << ": " <<
                            curl_easy_strerror(res) << endl;
62                      continue;
63                  }
64
65                  size_t pos = readBuffer.find("\"lat\":\"");
66                  if (pos != string::npos) {
67                      pos += 7;
68                      size_t end = readBuffer.find("\"", pos);
69                      double lat = stod(readBuffer.substr(pos, end
                            - pos));
70                      pos = readBuffer.find("\"lon\":\"", end);
71                      if (pos != string::npos) {
72                          pos += 7;
73                          end = readBuffer.find("\"", pos);
74                          double lon = stod(readBuffer.substr(pos,
                                end - pos));
75                          string name = input.first.empty() ?
                                "   im   " + to_string(i) :
                                input.first;
```

```
                        points.push_back({lon, lat, name, 8.0,
                            12.0});
                        cout << " L y   t   OSM: " << name << "
                            => (" << lon << ", " << lat << ",
                            8.0, 12.0)\n";
                    }
                } else {
                    cerr << "Kh ng t m  t h y : " <<
                        input.first << endl;
                }
                readBuffer.clear();
            }
        }
        curl_easy_cleanup(curl);
    }
    curl_global_cleanup();
    return points;
}
```

Giải thích:

- **Mục đích**: Lấy tọa độ và cửa sổ thời gian từ file `inputdoan2.txt` hoặc OpenStreetMap API.
- **Cách hoạt động**:
  - **Đọc từ file**: Đọc `inputdoan2.txt`, lưu tên tỉnh, tọa độ, và cửa sổ thời gian vào `file_data`.
  - **Lấy từ API**: Nếu tỉnh không có trong file, gọi API OpenStreetMap:
    - Nếu nhập tên tỉnh (ví dụ: "Hà Nội"), tìm tọa độ bằng `search`.
    - Nếu nhập tọa độ, tìm tên bằng `reverse`.
    - Gán cửa sổ thời gian mặc định `[8.0, 12.0]` nếu không có trong file.
  - Lưu các điểm vào `points` (kiểu `vector<Point>`).
- **Ví dụ**: Với `inputdoan2.txt` chứa `Hà Nội 21.0285 105.8542 8.0 12.0`, hàm tạo `Point{105.8542, 21.0285, "Hà Nội", 8.0, 12.0}`. Nếu tỉnh không có trong file, API trả về tọa độ và gán `[8.0, 12.0]`.

1.4 Hàm tính khoảng cách Euclidean

```
// T nh  k h o ng  c ch Euclidean  g i a  hai  im
double euclidean_distance(Point p1, Point p2) {
    return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
}
```

Giải thích:

- **Mục đích**: Tính khoảng cách giữa hai điểm `p1` và `p2` bằng công thức Euclidean: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

- **Cách hoạt động**: Sử dụng `pow` để tính bình phương chênh lệch tọa độ và `sqrt` để lấy căn bậc hai.
- **Ví dụ**: Nếu `p1 = {105.8542, 21.0285}` (Hà Nội) và `p2 = {106.6602, 10.7626}` (TP. Hồ Chí Minh), khoảng cách là $\sqrt{(106.6602 - 105.8542)^2 + (10.7626 - 21.0285)^2} \approx 10.28$ độ.

1.5 Hàm tính thông tin tuyến đường

```
// T nh th ng tin t u y n ng ( k h o ng c ch , t h i gian, chi ph , k h thi)
RouteInfo calculate_route_info(const vector<Point>& points, const vector<int>& route, double speed_kmh = 50.0, double cost_per_km = 10.0) {
    double total_distance = 0.0;
    double total_time = 8.0; // B t u l c 8h s ng
    double total_cost = 0.0;
    bool time_feasible = true;

    for (size_t i = 0; i < route.size() - 1; ++i) {
        double dist = euclidean_distance(points[route[i]], points[route[i + 1]]);
        total_distance += dist;
        total_cost += dist * cost_per_km; // Chi ph t l v i k h o ng c ch
        double travel_time = dist / speed_kmh; // T h i gian = k h o ng c ch / t c
        total_time += travel_time;

        int next = route[i + 1];
        if (total_time < points[next].start_time) {
            total_time = points[next].start_time; // i n khi b t u c a s t h i gian
        } else if (total_time > points[next].end_time) {
            time_feasible = false; // Vi p h m c a s t h i gian
        }
    }

    return {route, total_distance, total_time, total_cost, time_feasible};
}
```

Giải thích:

- **Mục đích**: Tính tổng khoảng cách, thời gian, chi phí, và kiểm tra tính khả thi của tuyến đường.
- **Cách hoạt động**:
  - Lặp qua các điểm trong `route`, tính khoảng cách giữa hai điểm liên tiếp bằng `euclidean_distance`.

- **Tổng khoảng cách**: Cộng tất cả khoảng cách giữa các điểm.
- **Chi phí**: Tính bằng khoảng cách nhân với `cost_per_km` (mặc định 10 đơn vị/độ).
- **Thời gian**: Bắt đầu lúc 8h sáng, cộng thời gian di chuyển (`dist / speed_kmh`, mặc định 50 km/h). Nếu đến sớm hơn `start_time`, chờ đến `start_time`. Nếu đến muộn hơn `end_time`, đánh dấu `time_feasible = false`.

- **Ví dụ**: Với tuyến đường `[0, 1, 0]` (Hà Nội → TP. Hồ Chí Minh → Hà Nội), giả sử khoảng cách mỗi đoạn là 10 độ, thời gian di chuyển mỗi đoạn là $10/50 = 0.2$ giờ, chi phí là $10 \times 10 = 100$ đơn vị. Nếu đến TP. Hồ Chí Minh lúc 8.2h (trong `[8.0, 12.0]`), tuyến đường khả thi.

1.6 Thuật toán Greedy

```cpp
// Thuật toán Greedy: Chọn điểm gần nhất
pair<vector<int>, double> greedy_tsp(vector<Point>& points) {
    int n = points.size();
    vector<bool> visited(n, false);
    vector<int> route = {0};
    visited[0] = true;
    int current = 0;
    for (int i = 0; i < n - 1; ++i) {
        double min_dist = INT_MAX;
        int next = -1;
        for (int j = 0; j < n; ++j) {
            if (!visited[j] &&
                euclidean_distance(points[current], points[j]) <
                min_dist) {
                min_dist = euclidean_distance(points[current],
                    points[j]);
                next = j;
            }
        }
        if (next == -1) break;
        visited[next] = true;
        route.push_back(next);
        current = next;
    }
    route.push_back(0);
    auto info = calculate_route_info(points, route);
    return {info.route, info.total_distance};
}
```

Giải thích:

- **Mục đích**: Xây dựng tuyến đường ban đầu bằng cách luôn chọn điểm gần nhất chưa đi qua.
- **Cách hoạt động**:
  - Bắt đầu từ kho (điểm 0, `route = {0}`).

- Đánh dấu các điểm đã đi qua bằng mảng `visited`.
- Từ điểm hiện tại (`current`), tìm điểm chưa đi qua có khoảng cách nhỏ nhất.
- Thêm điểm đó vào `route`, cập nhật `current`, lặp lại cho đến khi đi qua tất cả điểm.
- Thêm điểm 0 vào cuối để quay lại kho.
- Tính thông tin tuyến đường bằng `calculate_route_info`.

- **Ví dụ**: Với 3 điểm (Hà Nội, TP. Hồ Chí Minh, Đà Nẵng), Greedy có thể tạo tuyến đường `0 → 1 → 2 → 0` nếu TP. Hồ Chí Minh gần Hà Nội nhất, và Đà Nẵng gần TP. Hồ Chí Minh nhất.

1.7 Thuật toán 2-opt

```
// Hàm thực hiện hoán đổi 2-opt
vector<int> two_opt_swap(const vector<int>& route, int i, int k) {
    vector<int> new_route = route;
    reverse(new_route.begin() + i, new_route.begin() + k + 1);
    return new_route;
}

// Thuật toán 2-opt: Tối ưu hóa tuyến đường
pair<vector<int>, double> two_opt_optimize(const vector<Point>& points, vector<int> route) {
    auto best_info = calculate_route_info(points, route);
    double best_dist = best_info.total_distance;
    bool improved = true;
    while (improved) {
        improved = false;
        for (int i = 1; i < route.size() - 2; ++i) {
            for (int k = i + 1; k < route.size() - 1; ++k) {
                auto new_route = two_opt_swap(route, i, k);
                auto new_info = calculate_route_info(points, new_route);
                if (new_info.total_distance < best_dist && new_info.time_feasible) {
                    route = new_route;
                    best_dist = new_info.total_distance;
                    best_info = new_info;
                    improved = true;
                }
            }
        }
    }
    return {best_info.route, best_dist};
}
```

Giải thích:

- **Mục đích**: Cải thiện tuyến đường Greedy bằng cách hoán đổi các đoạn đường để giảm tổng khoảng cách, đồng thời đảm bảo tính khả thi thời gian.
- **Cách hoạt động**:
  - `two_opt_swap`: Đảo ngược một đoạn của tuyến đường từ chỉ số `i` đến `k` (ví dụ: `[0, 1, 2, 3, 0]` với `i=1`, `k=3` trở thành `[0, 3, 2, 1, 0]`).
  - `two_opt_optimize`: Lặp qua tất cả cặp `(i, k)`, thử hoán đổi, và giữ tuyến đường mới nếu khoảng cách nhỏ hơn và thỏa mãn cửa sổ thời gian.
  - Lặp lại cho đến khi không còn cải thiện (`improved = false`).
- **Ví dụ**: Nếu tuyến đường `0 → 1 → 2 → 3 → 0` có khoảng cách lớn do đường từ 1 đến 2 và 2 đến 3 cắt nhau, 2-opt có thể hoán đổi thành `0 → 1 → 3 → 2 → 0` để giảm khoảng cách.

1.8 Thuật toán Simulated Annealing

```
// Thuật toán Simulated Annealing: Tối ưu hóa metaheuristic
pair<vector<int>, double> simulated_annealing_optimize(const vector<Point>& points, vector<int> route, double initial_temp = 1000.0, double cooling_rate = 0.995, int max_iterations = 10000) {
    auto best_info = calculate_route_info(points, route);
    vector<int> best_route = route;
    double best_dist = best_info.total_distance;

    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(0.0, 1.0);
    uniform_int_distribution<> index_dis(1, route.size() - 2);

    double temp = initial_temp;
    for (int iter = 0; iter < max_iterations; ++iter) {
        int i = index_dis(gen);
        int k = index_dis(gen);
        if (k < i) swap(i, k);
        if (k == i) k = i + 1;

        auto new_route = two_opt_swap(route, i, k);
        auto new_info = calculate_route_info(points, new_route);
        double new_dist = new_info.total_distance;

        if (new_info.time_feasible && (new_dist < best_dist || dis(gen) < exp((best_dist - new_dist) / temp))) {
            route = new_route;
            if (new_dist < best_dist && new_info.time_feasible) {
                best_route = new_route;
                best_dist = new_dist;
                best_info = new_info;
```

```
            }
        }

        temp *= cooling_rate;
    }

    return {best_route, best_dist};
}
```

Giải thích:

- **Mục đích**: Tối ưu hóa tuyến đường bằng cách sử dụng metaheuristic Simulated Annealing, chấp nhận cả các thay đổi không tối ưu để thoát khỏi cực trị địa phương.
- **Cách hoạt động**:
  - Bắt đầu từ tuyến đường 2-opt.
  - Tạo hoán đổi 2-opt ngẫu nhiên bằng cách chọn `i`, `k` ngẫu nhiên.
  - Chấp nhận tuyến đường mới nếu:
    - Khoảng cách nhỏ hơn và khả thi về thời gian.
    - Hoặc theo xác suất $\exp((best\_dist - new\_dist)/temp)$, cho phép chấp nhận tuyến đường tệ hơn để tìm giải pháp tốt hơn sau này.
  - Giảm `temp` theo `cooling_rate` (0.995) qua 10000 vòng lặp.
- **Ví dụ**: Nếu tuyến đường 2-opt không tối ưu do mắc kẹt ở cực trị địa phương, Simulated Annealing có thể thử các hoán đổi ngẫu nhiên và tìm tuyến đường tốt hơn.

1.9 Hàm chính (`main`)

```
// H m  ch nh
int main() {
    vector<pair<string, pair<double, double>>> inputs;
    int mode;
    cout << " C h n  c h  :\n1.  N h p  t  file\n2.  N h p
        t h  c ng\n3.  S  d ng  d  l i u  m u \n> ";
    cin >> mode;
    cin.ignore();

    if (mode == 1) {
        string filename;
        cout << " N h p  t n file: ";
        getline(cin, filename);
        inputs = read_inputs_from_file(filename);
    } else if (mode == 2) {
        int n;
        cout << " N h p  s  l  ng  im  : ";
        cin >> n;
        cin.ignore();
        for (int i = 0; i < n; ++i) {
```

```
            string name;
            double lat, lon;
            cout << " N h p  t  n  h o c  t a  c a
                 im  " << i + 1 << ": ";
            getline(cin, name);
            if (isdigit(name[0]) || name[0] == '-') {
                istringstream iss(name);
                iss >> lat >> lon;
                inputs.push_back({"", {lat, lon}});
            } else {
                inputs.push_back({name, {0, 0}});
            }
        }
    } else {
        inputs = {
            {"H   N i ", {0, 0}},
            {"TP.  H   Ch  Minh", {0, 0}},
            {"     N ng ", {0, 0}},
            {"", {10.403, 106.732}}
        };
    }

    auto points = fetch_coordinates_from_osm(inputs);
    if (points.size() < 2) {
        cerr << "\ n   L i : Kh ng  l y  c  t a
             .  C h  t  m  t h y  " << points.size() << "
            im  .\n";
        cerr << "      Vui  l  ng  k i m  tra  k t  n i
           internet  h o c  t  n  a  im  .\n";
        system("pause");
        return 1;
    }

    auto greedy = greedy_tsp(points);
    auto greedy_info = calculate_route_info(points,
       greedy.first);
    auto two_opt = two_opt_optimize(points, greedy.first);
    auto two_opt_info = calculate_route_info(points,
       two_opt.first);
    auto sa = simulated_annealing_optimize(points,
       two_opt.first);
    auto sa_info = calculate_route_info(points, sa.first);

    cout << "\ n T u y n  ng  Greedy:\n";
    for (int i : greedy_info.route) cout << i << "(" <<
       points[i].name << ") ";
    cout << "\ n K h o ng  c  ch: " << greedy_info.total_distance
       << "  \n";
    cout << " T ng  t h i  gian: " << greedy_info.total_time <<
       "  g i  \n";
```

```
60      cout << " T ng  chi ph  : " << greedy_info.total_cost << "
            n    v  \n";
61      cout << " K h   thi  v    t h i  gian: " <<
           (greedy_info.time_feasible ? "C  " : "Kh ng") << endl;
62
63      cout << "\ n T u y n     ng     t i   u  h a 2-opt:\n";
64      for (int i : two_opt_info.route) cout << i << "(" <<
           points[i].name << ") ";
65      cout << "\ n K h o ng  c  ch: " << two_opt_info.total_distance
           << "      \n";
66      cout << " T ng   t h i  gian: " << two_opt_info.total_time <<
           "  g i  \n";
67      cout << " T ng  chi ph  : " << two_opt_info.total_cost << "
            n    v  \n";
68      cout << " K h   thi  v    t h i  gian: " <<
           (two_opt_info.time_feasible ? "C  " : "Kh ng") << endl;
69
70      cout << "\ n T u y n     ng     t i   u  h a Simulated
           Annealing:\n";
71      for (int i : sa_info.route) cout << i << "(" <<
           points[i].name << ") ";
72      cout << "\ n K h o ng  c  ch: " << sa_info.total_distance << "
               \n";
73      cout << " T ng   t h i  gian: " << sa_info.total_time << "
            g i  \n";
74      cout << " T ng  chi ph  : " << sa_info.total_cost << "   n
            v  \n";
75      cout << " K h   thi  v    t h i  gian: " <<
           (sa_info.time_feasible ? "C  " : "Kh ng") << endl;
76
77      system("pause");
78      ofstream out("tsp_result.csv");
79      out << "T n ,Kinh      , V        , T h i  gian  b t
            u   , T h i  gian  k t  th c\n";
80      for (int idx : sa_info.route) {
81          out << points[idx].name << "," << points[idx].x << ","
              << points[idx].y << ","
82              << points[idx].start_time << "," <<
                 points[idx].end_time << "\n";
83      }
84      out.close();
85
86      cout << "     l  u  k t   q u   v  o  tsp_result.csv\n";
87      return 0;
88  }
```

Giải thích:

- **Mục đích**: Điều khiển luồng chính của chương trình, từ nhập liệu đến xuất kết quả.
- **Cách hoạt động**:

- Yêu cầu người dùng chọn chế độ nhập liệu:
  - **Chế độ 1**: Đọc từ file `inputdoan2.txt`.
  - **Chế độ 2**: Nhập thủ công số lượng điểm, sau đó nhập tên tỉnh hoặc tọa độ.
  - **Chế độ 3**: Sử dụng dữ liệu mẫu (4 điểm).
- Gọi `fetch_coordinates_from_osm` để lấy tọa độ và cửa sổ thời gian.
- Tạo tuyến đường bằng `greedy_tsp`, tối ưu bằng `two_opt_optimize` và `simulated_annealing`
- Xuất thông tin tuyến đường (Greedy, 2-opt, Simulated Annealing) ra console.
- Lưu tuyến đường Simulated Annealing vào `tsp_result.csv`.

- **Ví dụ**: Nếu chọn chế độ 1 và nhập `inputdoan2.txt`, chương trình đọc 4 tỉnh, lấy tọa độ từ file, và tối ưu tuyến đường.

2. Vẽ bản đồ tuyến đường bằng Python (`drawdoan2.py`)

2.1 Mã Python

```
import folium
import pandas as pd

#    c    file CSV  t    ch   ng  tr nh  C++
df = pd.read_csv("tsp_result.csv")

# T nh  t a        trung  t  m  cho  b n
center_lat = df["Latitude"].mean()
center_lon = df["Longitude"].mean()

#  T o   b n
m = folium.Map(location=[center_lat, center_lon], zoom_start=6)

# Th m  c  c    im    v    v    t u y n     ng
route = []
for i, row in df.iterrows():
    lat, lon = row["Latitude"], row["Longitude"]
    name = row["Name"]
    start_time = row["StartTime"]
    end_time = row["EndTime"]
    popup_text = f"{name}<br> C a   s    t h i  gian:
       {start_time:.1f} - {end_time:.1f}"
    folium.Marker([lat, lon], popup=popup_text).add_to(m)
    route.append([lat, lon])

#  N i   c  c    im     b ng      ng
folium.PolyLine(route, color="blue", weight=3).add_to(m)

# T nh  t ng  k h o ng  c ch,  t h i  gian  v  chi  p h
total_distance = 0.0
total_time = 8.0  #  B t    u    l  c  8h  s  ng
```

```
speed_kmh = 50.0
cost_per_km = 10.0
total_cost = 0.0
for i in range(len(route) - 1):
    dx = route[i + 1][1] - route[i][1]
    dy = route[i + 1][0] - route[i][0]
    dist = (dx**2 + dy**2)**0.5
    total_distance += dist
    total_time += dist / speed_kmh
    total_cost += dist * cost_per_km
    if i < len(route) - 1:
        if total_time < df["StartTime"].iloc[i + 1]:
            total_time = df["StartTime"].iloc[i + 1]

# Th m t m t t v o b n
summary_html = f"""
<div style="position: fixed; bottom: 50px; left: 50px;
   background-color: white; padding: 10px; border: 1px solid
   black;">
    <h4>T m t t t u y n ng </h4>
    <p> T ng k h o ng c ch: {total_distance:.2f} </p>
    <p> T ng t h i gian: {total_time:.2f} g i </p>
    <p> T ng chi ph : {total_cost:.2f} n v </p>
</div>
"""
folium.Element(summary_html).add_to(m)

# L u b n v o file HTML
m.save("tsp_map.html")
print("  l u b n  v o tsp_map.html")
```

Giải thích:

- **Mục đích**: Đọc `tsp_result.csv`, vẽ tuyến đường trên bản đồ tương tác, hiển thị thông tin điểm và tóm tắt.
- **Cách hoạt động**:
  - **Đọc CSV**: Sử dụng `pandas` để đọc `tsp_result.csv`, chứa tên, kinh độ, vĩ độ, cửa sổ thời gian.
  - **Tạo bản đồ**: Tính trung tâm bản đồ (`center_lat`, `center_lon`) và tạo bản đồ bằng `folium.Map` với độ phóng 6.
  - **Thêm điểm**: Đặt marker cho mỗi điểm với popup hiển thị tên và cửa sổ thời gian (ví dụ: "Hà Nội, Cửa sổ thời gian: 8.0 - 12.0").
  - **Vẽ tuyến đường**: Nối các điểm bằng đường màu xanh (`PolyLine`).
  - **Tính tóm tắt**: Tính lại khoảng cách, thời gian, chi phí (tương tự `calculate_route_info`) và hiển thị trong hộp tóm tắt.
  - **Lưu bản đồ**: Xuất ra `tsp_map.html` để xem trong trình duyệt.

- **Ví dụ**: Với `tsp_result.csv` chứa 4 tỉnh, bản đồ hiển thị 4 điểm, đường nối, và tóm tắt như "Tổng khoảng cách: 20.56 độ, Tổng thời gian: 8.41 giờ, Tổng chi phí: 205.60 đơn vị".

3. Kết luận và đề xuất nâng cao

3.1 Kết quả đạt được

Chương trình đã đáp ứng đầy đủ các yêu cầu:

- **Yêu cầu tối thiểu**:
  - Nhập ít nhất 5–10 điểm (hỗ trợ 63 tỉnh hoặc nhập thủ công).
  - Tính tuyến đường tối ưu với khoảng cách Euclidean nhỏ nhất.
  - Xuất tuyến đường, khoảng cách, thời gian, chi phí, và tính khả thi.
- **Yêu cầu nâng cao**:
  - Sử dụng dữ liệu thực tế từ OpenStreetMap API.
  - Tích hợp ràng buộc thời gian (`[8.0, 12.0]`) và chi phí (10 đơn vị/độ).
  - Trực quan hóa tuyến đường bằng `drawdoan2.py`.
  - Sử dụng Greedy, 2-opt, và Simulated Annealing, so sánh kết quả.

3.2 Đề xuất nâng cao

- Thêm thuật toán 3-opt hoặc Ant Colony để cải thiện chất lượng tuyến đường.
- Hỗ trợ nhập cửa sổ thời gian thủ công khi chọn chế độ nhập tay.
- Tích hợp Google Maps API để hiển thị tuyến đường thực tế hơn.
- Thêm thời gian phục vụ tại mỗi điểm (ví dụ: 0.5 giờ/điểm) để mô phỏng thực tế.