Báo cáo chương trình tìm đường A* bằng ngôn ngữ C++

Ngày 18 tháng 06 năm 2025

Mục lục

1 Giới thiệu

Chương trình được viết bằng ngôn ngữ C++ để giải bài toán tìm đường đi ngắn nhất từ thành phố xuất phát đến thành phố đích sử dụng thuật toán A* (A-star). Dữ liệu bao gồm danh sách các thành phố, các cạnh kết nối giữa chúng và giá trị heuristic (ước lượng chi phí từ thành phố đến đích).

2 Nguyên lý giải thuật A*

Thuật toán A* là một thuật toán tìm đường tối ưu trong đồ thị, sử dụng hàm đánh giá $f(n) = g(n) + h(n)$, trong đó:

- $g(n)$: chi phí từ điểm bắt đầu đến nút n.
- $h(n)$: ước lượng chi phí từ n đến đích (heuristic).
- $f(n)$: tổng chi phí ước tính của con đường đi qua n đến đích.

Thuật toán sử dụng hàng đợi ưu tiên để chọn đỉnh có $f(n)$ nhỏ nhất và cập nhật đường đi ngắn nhất tìm được cho đến khi đến đích.

3 Cấu trúc chương trình

- **Edge**: struct lưu thông tin thành phố lân cận và chi phí đi đến.
- **graph**: biểu diễn đồ thị dưới dạng danh sách kề.
- **heuristic**: ánh xạ thành phố với giá trị heuristic tương ứng.
- **Node**: struct lưu thông tin một đỉnh đang xét trong thuật toán.
- **a_start(start, goal)**: hàm chính thực thi thuật toán A*.

4 Giải thích mã nguồn C++

- `taiFileGraph(filename)`: Đọc dữ liệu cạnh từ file:

```
void taiFileGraph(const string& filename){
    ifstream file(filename);
    string city1, city2;
    int cost;
    while(file >> city1 >> city2 >> cost){
        graph[city1].push_back({city2, cost});
        graph[city2].push_back({city1, cost});
    }
}
```

- `taiFileHeuristic(filename)`: Đọc giá trị heuristic từ file:

```
void taiFileHeuristic(const string& filename){
    ifstream file(filename);
    string city;
    int h;
    while(file >> city >> h){
        heuristic[city] = h;
    }
}
```

- **Struct Node**: Dùng trong hàng đợi ưu tiên:

```
struct Node {
    string city;
    int cost;
    int totalCost;
    bool operator>(const Node& other) const {
        return totalCost > other.totalCost;
    }
};
```

- `a_start(start, goal)`: Thuật toán A* tìm đường:

```
void a_start(string start, string goal) {
    priority_queue<Node, vector<Node>, greater<Node>> openSet;
    unordered_map<string,int> gScore;
    unordered_map<string,string> cameFrom;
    set<string> visited;

    for (const auto& node : graph) gScore[node.first] = INT_MAX;
    gScore[start] = 0;
    openSet.push({start, 0, heuristic[start]});

    while (!openSet.empty()) {
        Node current = openSet.top(); openSet.pop();
        if (current.city == goal) {
            vector<string> path;
            for (string at = goal; !at.empty(); at = cameFrom[at])
                path.push_back(at);
            reverse(path.begin(), path.end());
            cout << "A* Path: ";
            for (size_t i = 0; i < path.size(); ++i)
                cout << path[i] << (i != path.size() - 1 ? "->" : "\\n"
            cout << "Total cost: " << gScore[goal] << endl;
            return;
        }
        if (visited.count(current.city)) continue;
        visited.insert(current.city);

        for (const Edge& edge : graph[current.city]) {
            int tentative_g = gScore[current.city] + edge.cost;
            if (tentative_g < gScore[edge.neighbor]) {
                cameFrom[edge.neighbor] = current.city;
                gScore[edge.neighbor] = tentative_g;
                int f = tentative_g + heuristic[edge.neighbor];
                openSet.push({edge.neighbor, tentative_g, f});
            }
        }
    }
    cout << "No path found.\\n";
}
```

5 Kết luận

Chương trình tìm đường sử dụng thuật toán A* với dữ liệu từ file. A* đảm bảo tìm ra đường đi ngắn nhất nếu hàm heuristic là tối ưu (không đánh giá quá thấp chi phí còn lại). Thuật toán hoạt động hiệu quả với dữ liệu có giới hạn và cấu trúc đồ thị rõ ràng.