ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

CẤU TRÚC RỜI RẠC CHO KHMT

TRAVELING SALESMAN PROBLEM

GVHD: MAI XUÂN TOÀN
SV thực hiện: Nguyễn Tấn Phát-2352888

Hồ Chí Minh, 6/2024

1. Giới thiệu

Bài toán người bán hàng (Travelling Salesman Problem - TSP) là một bài toán NP-khó thuộc thể loại tối ưu rời rạc hay tổ hợp được nghiên cứu trong vận trù học hoặc lý thuyết khoa học máy tính. Bài toán được phát biểu như sau. Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất thăm mỗi thành phố đúng một lần - theo Wikipedia.

Mục đích của báo cáo này là để hiểu về phương pháp tiếp cận Dynamic Programming with Bitmasking để giải quyết bài toán TSP.

2. Định nghĩa vấn đề

Bài toán được thể hiện bằng ma trận. Trong đó các thành phố là các đỉnh và khoảng cách giữa chúng là trọng số các cạnh. Mục tiêu là xác định chu trình Hamilton ngắn nhất để giải quyết bài toán.

3. Cách tiếp cận

Cách tiếp cận chính của bài toán là Dynamic Programming with Bitmasking.

Dynamic Programming with Bitmasking là một phương pháp phổ biến và hiệu quả để giải quyết vấn đề này bằng cách sử dụng kỹ thuật quy hoạch động (Dynamic Programming) kết hợp với việc sử dụng bitmask. Đối với bài toán TSP Bitmasking được sử dụng để lưu trữ trạng các đỉnh đã được ghé qua.

Dynamic Programming: Để giải quyết bài toán TSP, tính toán các đoạn đường có thể có từ một đỉnh xuất phát đến mọi đỉnh khác, đi qua tất cả các đỉnh và quay lại đỉnh xuất phát. Dynamic Programming giải quyết bài toán này bằng cách lưu lại và sử dụng lại các đoạn đường ngắn nhất.

Bitmasking: Được sử dụng để lưu trữ trạng thái của các đỉnh đã được đi qua trong quá trình duyệt qua các đoạn đường. Bằng cách sử dụng bitmask để đánh dấu các đỉnh đã đi qua, dễ dàng kiểm tra và cập nhật trạng thái của các đỉnh.

Hàm tsp:

```
int tsp(int graph[20][20], int pos, int visited, vector<
    vector<int>>& dp, vector<vector<int>>& parent, int
    numVertices) {
    if (visited == ((1 << numVertices) - 1)) {
        return graph[pos][0] == 0 ? INF : graph[pos][0];
            point
    }

    if (dp[pos][visited] != -1) {
        return dp[pos][visited];
    }

    int ans = INF;
    for (int city = 0; city < numVertices; city++) {
        if ((visited & (1 << city)) == 0 && graph[pos][city
            ] != 0) {
            int newAns = graph[pos][city] + tsp(graph, city
                , visited | (1 << city), dp, parent,
                numVertices);
            if (newAns < ans) {
                ans = newAns;
                parent[pos][visited] = city; // Save the
                    parent (next city) in the path
            }
        }
    }

    return dp[pos][visited] = ans;
}
```

Listing 1: Hàm tsp

Hàm tsp (Travelling Salesman Problem) dùng để tìm khoảng cách thấp nhất của quá trình đi qua tất cả các đỉnh của đồ thị bắt đầu từ một đỉnh cụ thể và quay trở lại đỉnh xuất phát.

Điều kiện: Nếu tất cả các đỉnh đã được ghé (visited == ((1 « numVertices) - 1)), hàm sẽ kiểm tra xem có thể quay lại đỉnh xuất phát hay không (graph[pos][0]). Nếu quay lại đỉnh xuất phát, trả về khoảng cách quay lại, nếu không thì trả về INF.

Duyệt các đỉnh: Nếu mỗi đỉnh chưa đi qua và có cạnh nối từ đỉnh hiện tại (graph[pos][city] != 0), tính khoảng cách mới (newAns) bằng cách tính đệ quy tới đỉnh tiếp theo (tsp(graph, city, visited | (1 « city), dp, parent, numVertices)).

Cập nhật kết quả: Nếu khoảng cách mới nhỏ hơn khoảng cách hiện

tại (newAns $<$ ans), cập nhật khoảng cách và lưu lại đỉnh mới đó.

Hàm Traveling:

```
string Traveling(int graph[20][20], int numVertices, char
    startVertex) {
    vector<vector<int>> dp(numVertices, vector<int>(1 <<
        numVertices, -1));
    vector<vector<int>> parent(numVertices, vector<int>(1
        << numVertices, -1));
    int start = startVertex - 'A';
    int cost = tsp(graph, start, 1 << start, dp, parent,
        numVertices);

    vector<int> path;
    int visited = 1 << start;
    int pos = start;
    path.push_back(pos);

    while (true) {
        int nextPos = parent[pos][visited];
        if (nextPos == -1) break;
        pos = nextPos;
        visited |= (1 << pos);
        path.push_back(pos);
    }
    path.push_back(start);

    // Convert the path to a string
    stringstream path_stream;
    for (int i : path) {
        path_stream << static_cast<char>('A' + i) << " ";
    }

    return path_stream.str();
}
```

Listing 2: Hàm Traveling

Hàm Traveling để khởi tạo và giải quyết bài toán TSP, sau đó tạo và trả về chuỗi biểu diễn đoạn đường tối ưu.

Khởi tạo: Tạo bảng dp và parent để lưu lại đỉnh tiếp theo.

Tính khoảng cách: Gọi hàm tsp để tính khoảng cách từ đỉnh bắt đầu.

Tạo lại đoạn đường: Dùng bảng parent để tái tạo đoạn đường từ đỉnh bắt đầu, qua các đỉnh theo thứ tự lưu trong bảng parent, và quay lại đỉnh bắt đầu.

Chuyển đổi thành chuỗi: Chuyển đổi đoạn đường (dạng vector các đỉnh) thành chuỗi ký tự, mỗi ký tự là một đỉnh.

4. Kết quả

Sử dụng testcase mẫu của trường để kiểm tra kết quả:

Testcase 1:

Input: Số đỉnh là 12, đỉnh bắt đầu là A, partile là 0.1

```
1
2  int main() {
3      int G[20][20];
4      ifstream fin("inMat1.txt");
5      int n = 12;
6
7      for (int i = 0; i < n; i++) {
8          for (int j = 0; j < n; j++) {
9              fin >> G[i][j];
10         }
11     }
12
13     string output = Traveling(G, n, 'A');
14     int cost = pathlength(G, n, output);
15     double partile = 0.1;
16
17     if ((204 - 204 * partile) <= cost && cost <= (204 + 204
           * partile)) {
18         cout << "your output is at least 90% correct" <<
              endl;
19     }
20     fin.close();
21     return 0;
22 }
```

Listing 3: Testcase 1

Output: your output is at least 90% correct

Testcase 2:

Input: Số đỉnh là 12, đỉnh bắt đầu là A, partile là 0.0

```
1
2  int main() {
3      int G[20][20];
4      ifstream fin("inMat1.txt");
5      int n = 12;
6
7      for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
            fin >> G[i][j];
        }
    }

    string output = Traveling(G, n, 'A');
    int cost = pathlength(G, n, output);
    double partile = 0.0;

    if ((204 - 204 * partile) <= cost && cost <= (204 + 204
        * partile)) {
        cout << "your output is correct" << endl;
    }
    fin.close();
    return 0;
}
```

Listing 4: Testcase 2

Output: your output is correct

5. Kết luận

Báo cáo này giải quyết bài toán người bán hàng (Travelling Salesman Problem - TSP) bằng cách sử dụng phương pháp tiếp cận Dynamic Programming with Bitmasking. Sử dụng Dynamic Programming kết hợp với Bitmasking là một trong những cách tốt để giải quyết bài toán TSP, giúp tối ưu hóa thời gian và không gian bộ nhớ đối với một bài toán có nhiều trường hợp như TSP.