ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CẤU TRÚC RỜI RẠC

GIẢI BÀI TOÁN NGƯỜI DU LỊCH

Tên Sinh Viên; Hồng Phi Trường

MSSV: 2313703

Lớp: L02

Ho Chi Minh City, November 2024

Mục lục

1 Mở đầu

Bài toán Người du lịch (Travelling Salesman Problem) là một trong những bài toán kinh điển và khó trong tin học. Bài toán người du lịch (TSP) đặt ra câu hỏi sau: Cho một danh sách các thành phố và khoảng cách giữa mỗi cặp thành phố, con đường ngắn nhất có thể ghé thăm mỗi thành phố đúng một lần và quay trở lại thành phố ban đầu là bao nhiêu? Đây là một bài toán NP khó trong tối ưu hóa tổ hợp, quan trọng trong nghiên cứu hoạt động và khoa học máy tính lý thuyết. Trong bài toán này em sử dụng phương pháp quy hoạch động để giải quyết.

2 Giải thuật Held-Karp (Dynamic Programming)

2.1 Ý tưởng thuật toán

Có nhiều cách giải quyết bài toán TSP như quay lui và xét nhánh cận. Tuy nhiên, với số lượng đỉnh lớn, các phương pháp này không tối ưu do gọi đệ quy quá nhiều lần, dẫn đến thời gian xử lý lâu. Thuật toán Held-Karp dựa trên quy hoạch động (Dynamic Programming) là một trong những phương pháp hiệu quả để giải quyết bài toán này.

2.2 Các bước thuật toán

1. **Bước 1:** Nhập đầu vào dưới dạng ma trận kề, trong đó giá trị của phần tử i, j là trọng số từ đỉnh i đến đỉnh j.
2. **Bước 2:** Tạo hai vector 2 chiều:
   - `dp`: lưu trữ chi phí nhỏ nhất để thăm tất cả các đỉnh trong trạng thái hiện tại.
   - `parent`: lưu thông tin đường đi để truy vết.

   Kích thước của cả hai vector là $2^n \times n$.
3. **Bước 3:** Khởi tạo chi phí tại thành phố ban đầu bằng 0. Duyệt qua tất cả các trạng thái `mask` (2^n trạng thái).
4. **Bước 4:** Với mỗi trạng thái `mask`, kiểm tra các đỉnh u, v:
   - Nếu u đã được thăm, v chưa thăm, và có đường đi từ u đến v, xem xét cập nhật trạng thái.
   - Nếu chi phí mới (`newCost`) nhỏ hơn chi phí hiện tại, cập nhật bảng `dp` và lưu thông tin đỉnh trước đó vào `parent`.
5. **Bước 5:** Khi đã thăm tất cả các đỉnh (`mask = (1 << n) - 1`), tính chi phí quay về đỉnh xuất phát. Tạo biến `last` để phục vụ việc truy vết ngược lại đường đi.
6. **Bước 6:** Sử dụng bảng `parent` để truy vết đường đi ngược từ `last` về đỉnh xuất phát:
   - Tạo vòng lặp, thêm đỉnh hiện tại vào vector `path`.
   - Cập nhật đỉnh hiện tại, loại bỏ đỉnh hiện tại khỏi `mask`, đến khi lấy hết tất cả các đỉnh.
   - Đảo ngược chuỗi `path` để có thứ tự đúng.
7. **Bước 7:** Trong hàm `Travelling`, cập nhật thêm đỉnh đầu tiên, chuyển các trọng số 0 hoặc âm thành 99999, chuyển path thành chuỗi ký tự và trả về kết quả.

3 Đánh giá

Thuật toán Held-Karp có độ phức tạp thời gian $O(2^n \times n)$ và độ phức tạp không gian $O(2^n \times n)$, hiệu quả hơn so với phương pháp quay lui trong trường hợp số đỉnh lớn. Tuy nhiên việc triển khai code khá phức tạp và rắc rối so với quay lui nhánh cận, ngoài ra còn nhiều thuật toán khác để giải quyết bài toán trên.

4 Code

tsm.cpp > ...

```cpp
#include "tsm.h"

void DP(int G[][30], int n, int start, vector<int> &path) {
    vector<vector<int>> dp(1 << n, vector<int>(n, 2147483647));
    vector<vector<int>> parent(1 << n, vector<int>(n, -1));

    dp[1 << start][start] = 0;

    for (int mask = 0; mask < (1 << n); ++mask) {
        for (int u = 0; u < n; ++u) {
            if (mask & (1 << u)) {
                for (int v = 0; v < n; ++v) {
                    if (!(mask & (1 << v)) && G[u][v] != 99999 && dp[mask][u] != 2147483647) {
                        int newCost = dp[mask][u] + G[u][v];
                        if (newCost < dp[mask | (1 << v)][v]) {
                            dp[mask | (1 << v)][v] = newCost;
                            parent[mask | (1 << v)][v] = u;
                        }
                    }
                }
            }
        }
    }

    int minCost = 2147483647, last = -1;
    for (int i = 0; i < n; ++i) {
        if (i != start && G[i][start] != 99999 && dp[(1 << n) - 1][i] != 2147483647) {
            int cost = dp[(1 << n) - 1][i] + G[i][start];
            if (cost <= minCost) {
                minCost = cost;
                last = i;
            }
        }
    }

    int mask = (1 << n) - 1, current = last;
    while (current != -1) {
        path.push_back(current);
        int temp = parent[mask][current];
        mask &= ~(1 << current);
        current = temp;
    }
    reverse(path.begin(), path.end());
}

string Traveling(int G[][30], int n, char startVertex) {
```

Hình 1: Hàm DP

```
string Traveling(int G[][30], int n, char startVertex) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (G[i][j] <= 0) G[i][j] = 99999;
        }
    }

    int start = startVertex - 'A';
    vector<int> path;
    DP(G, n, start, path);

    string kq;
    for (int i =0; i<path.size();i++) {
        kq += char('A' + path[i]);
        kq += " ";
    }
    kq += startVertex;
    return kq;
}
```

Hình 2: Hàm Travelling