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

KHOA KHOA HỌC & KĨ THUẬT MÁY TÍNH

CẤU TRÚC RỜI RẠC CHO KHOA HỌC MÁY TÍNH - CO1007

BÁO CÁO BÀI TẬP LỚN

BELLMAN - FORD ALGORITHM

Sinh viên:
Nguyễn Đức Thịnh (2313284)

Giảng viên:
TS. Mai Xuân Toàn

Tp. Hồ Chí Minh, Tháng 06/2024

Mục lục

§1 Bài toán người bán hàng (Traveling Salesman Problem)

Bài toán người bán hàng (TSP - Traveling Salesman Problem) là một trong những bài toán kinh điển của khoa học máy tính và toán học. Bài toán này xuất phát từ một tình huống thực tế đơn giản: một người bán hàng muốn ghé thăm mỗi thành phố trong một tập hợp các thành phố, chỉ đường ngắn nhất có thể, rồi quay trở về thành phố xuất phát. Mặc dù tình huống này có vẻ đơn giản, nhưng bài toán TSP là một bài toán NP-khó, có nghĩa là không có thuật toán hiệu quả để giải quyết tất cả các trường hợp trong một thời gian hợp lý.

Bài toán TSP có thể được mô tả như sau: Cho trước một tập hợp các thành phố và khoảng cách giữa mỗi cặp thành phố, bài toán yêu cầu tìm một đường đi qua tất cả các thành phố sao cho mỗi thành phố chỉ được ghé qua một lần và chi phí (hoặc khoảng cách) của đường đi là nhỏ nhất.

Bài toán TSP có ứng dụng rộng rãi trong thực tế, từ việc lập lịch vận chuyển hàng hóa, lập kế hoạch du lịch đến các ứng dụng trong lĩnh vực công nghệ, như việc tối ưu hóa mạng lưới điện, định tuyến trong mạng máy tính, và nhiều ứng dụng khác.

Mặc dù không có thuật toán hiệu quả để giải quyết bài toán TSP cho tất cả các trường hợp, nhưng đã có nhiều phương pháp được phát triển để giải quyết các trường hợp cụ thể hoặc để tìm ra các giải pháp gần đúng. Điều này làm cho bài toán TSP trở thành một điểm nóng trong nghiên cứu thuật toán và tối ưu hóa.

§2 Giới thiệu

§2.1 Định nghĩa Đệ quy

Đệ quy là kỹ thuật để một hàm tự gọi lại chính nó. Kỹ thuật này cung cấp một cách để chia các vấn đề phức tạp thành các vấn đề đơn giản hơn, dễ giải quyết hơn.

```
void recursion(n){
    //code
    recursion(n + 1) //calling self function
}
```

§2.2 Các dạng Đệ quy

Chúng ta có thể sử dụng các dạng đệ quy khác nhau trong một số ngữ cảnh và vấn đề nhất định. Chúng ta có thể liệt kê một số dạng như sau:

1. **Chia để trị (Divide and conquer):** Đệ quy thường được sử dụng trong các thuật toán chia để trị, nơi một vấn đề được chia thành các vấn đề con nhỏ hơn được giải quyết đệ quy. Các giải pháp này sau đó được kết hợp để giải quyết vấn đề ban đầu.

2. **Xử lí tuần tự (Sequential Processing):** Trong một số tình huống mà bạn cần đảm bảo một chuỗi các thao tác xảy ra tuần tự (ví dụ, khi xử lý các tác vụ không đồng bộ), đệ quy có thể giúp quản lý các quá trình tuần tự này một cách trực quan hơn so với các vòng lặp.

3. **Quay lui (Backtracking):** Đệ quy thường được sử dụng trong các thuật toán quay lui, nơi giải pháp được xây dựng từng bước bằng cách thực hiện một loạt các

lựa chọn. Nếu một lựa chọn dẫn đến ngõ cụt, thuật toán sẽ quay lui về trạng thái trước đó và thử lựa chọn khác.

4. **Cấu trúc dạng cây (Tree-like structures):** Các thuật toán đệ quy rất phù hợp cho các vấn đề liên quan đến cấu trúc dạng cây, chẳng hạn như cây nhị phân, nơi mỗi nút có các nút con. Duyệt hoặc tìm kiếm các cấu trúc như vậy có thể được thực hiện hiệu quả bằng cách sử dụng đệ quy. Các thuật toán đệ quy cũng rất phù hợp với một số loại cấu trúc dữ liệu nhất định, chẳng hạn như đồ thị.

Bây giờ chúng ta sẽ xem xét dạng đệ quy sẽ được sử dụng trong bài toán Traveling Salesmen Problem là Quay lui (Backtracking).

§2.3 Đệ quy quay lui

Đệ quy quay lui là một kỹ thuật được sử dụng trong các thuật toán để tìm kiếm hệ thống các giải pháp cho một vấn đề bằng cách khám phá các khả năng khác nhau và quay lui khi đạt đến một ngõ cụt. Nó liên quan đến việc thử các lựa chọn một cách đệ quy và hoàn tác những lựa chọn đó nếu chúng không dẫn đến một giải pháp. Đệ quy quay lui thường được sử dụng trong các vấn đề liên quan đến việc tìm tất cả các giải pháp có thể, như các câu đố, các vấn đề thoả mãn ràng buộc và các vấn đề tối ưu hóa. Dưới đây là ví dụ về bài toán N - Hậu sử dụng Đệ quy quay lui.

```
#include <iostream>
#include <vector>
using namespace std;

bool isSafe(vector<vector<int>>& board, int row, int col, int n) {
    for (int i = 0; i < row; i++) {
        if (board[i][col] == 1) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (board[i][j] == 1) {
            return false;
        }
    }
    return true;
}
bool solveNQueens(vector<vector<int>>& board, int row, int n) {
    if (row == n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cout << board[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl;
        return true;
    }
    bool res = false;
    for (int col = 0; col < n; col++) {
```

```
        if (isSafe(board, row, col, n)) {
            board[row][col] = 1;
            res = solveNQueens(board, row + 1, n) || res;
            board[row][col] = 0;
        }
    }
    return res;
}
```

§3 Thuật toán Traveling Salesmen Problem

§3.1 Mô tả thuật toán

Để giải bài toán TSP bằng đệ quy quay lui, chúng ta có thể tuân theo các bước sau:

1. **Xác định bản đồ và khoảng cách:** Đầu tiên, xác định bản đồ các thành phố và khoảng cách giữa chúng. Điều này có thể được biểu diễn dưới dạng một ma trận hoặc một đồ thị. Xác định điểm xuất phát: Chọn một thành phố bất kỳ làm điểm xuất phát cho hành trình.

2. **Tạo một hành trình ban đầu:** Bắt đầu từ điểm xuất phát, thêm thành phố tiếp theo vào hành trình. Điều này có thể thực hiện bằng cách duyệt qua tất cả các thành phố có thể đến từ vị trí hiện tại và chọn thành phố có khoảng cách nhỏ nhất.

3. **Kiểm tra điều kiện dừng:** Kiểm tra xem đã thăm tất cả các thành phố chưa. Nếu đã thăm tất cả các thành phố, kiểm tra xem có một đường đi từ thành phố hiện tại trở lại thành phố xuất phát không. Nếu có, đây là một lời giải.

4. **Đệ quy quay lui:** Nếu chưa thăm hết tất cả các thành phố, thử thêm mỗi thành phố chưa được thăm vào hành trình và tiếp tục đệ quy từ đó.

5. **Quay lui:** Nếu không có đường đi nào từ thành phố hiện tại để hoàn thành hành trình, quay lại bước trước đó và thử một lựa chọn khác.

6. **Lặp lại quá trình:** Lặp lại quá trình cho đến khi tìm thấy một hành trình hoặc kiểm tra tất cả các khả năng mà không tìm thấy lời giải.

7. **In kết quả:** Nếu tìm thấy một hành trình, in ra hành trình đó. Nếu không, thông báo rằng không có lời giải.

§3.2 Chương trình C++

```
#include<iostream>
using namespace std;

const int MAX = 100;

int visit[MAX];
int path[MAX];
int best_path[MAX];
int min_cost;

void tspUtil(int graph[20][20], int cur_pos, int n, int count, int cost
    , int start) {
```

```
    if (count == n && graph[cur_pos][start]) {
        if (cost + graph[cur_pos][start] < min_cost) {
            min_cost = cost + graph[cur_pos][start];
            for (int i = 0; i < n; ++i) {
                best_path[i] = path[i];
            }
            best_path[n] = start;
        }
        return;
    }
    for (int i = 0; i < n; ++i) {
        if (!visit[i] && graph[cur_pos][i]) {
            visit[i] = 1;
            path[count] = i;

            tspUtil(graph, i, n, count + 1, cost + graph[cur_pos][i],
   start);

            visit[i] = 0;
        }
    }
}
string Traveling(int graph[20][20], int n, char start_vertex) {
    int start = start_vertex - 'A';
    string result;
    min_cost = 1e9;
    for (int i = 0; i < n; ++i) {
        visit[i] = 0;
        path[i] = -1;
        best_path[i] = -1;
    }
    visit[start] = 1;
    path[0] = start;

    tspUtil(graph, start, n, 1, 0, start);

    for (int i = 0; i <= n; ++i) {
        result += char(best_path[i] + 'A');
        if (i < n) {
            result += " ";
        }
    }
    return result;
}
```

Tài liệu tham khảo

- *Recursion*, https://en.wikipedia.org/wiki/Recursion_(computer_science)
- *Travelling salesman problem*, https://en.wikipedia.org/wiki/Travelling_salesman_problem
- *Backtracking*, https://en.wikipedia.org/wiki/Backtracking
- *Eight queens puzzle*, https://en.wikipedia.org/wiki/Eight_queens_puzzle
- *Bellman–Ford algorithm*, https://en.wikipedia.org/wiki/Bellman–Ford_algorithm