ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG ĐIỆN – ĐIỆN TỬ

BÁO CÁO

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Đề tài

Giải thuật Backtracking và ứng dụng

GIẢNG VIÊN HƯỚNG DẪN: TS TẠ THỊ KIM HUỆ

SINH VIÊN THỰC HIỆN: TRẦN HOÀI NAM

MSSV: 20224412

LỚP: CTTT HỆ THỐNG NHÚNG THÔNG MINH VÀ IOT K67

MÃ LỚP: 152455

Hà Nội, 01/2024

LỜI NÓI ĐẦU

Trong thời đại công nghệ thông tin phát triển vượt bậc, các cấu trúc dữ liệu và giải thuật đã trở thành nền tảng quan trọng trong việc giải quyết các bài toán thực tế, từ lập trình cơ bản đến các lĩnh vực phức tạp như trí tuệ nhân tạo, xử lý dữ liệu lớn và tối ưu hóa hệ thống. Môn học **Cấu trúc Dữ liệu và Giải thuật** cung cấp cho sinh viên những kiến thức căn bản và kỹ năng cần thiết để tổ chức, xử lý dữ liệu cũng như áp dụng các thuật toán nhằm giải quyết hiệu quả những bài toán thực tiễn.

Báo cáo này tập trung nghiên cứu về giải thuật **Backtracking** – một phương pháp tìm kiếm và thử sai mạnh mẽ, được sử dụng rộng rãi để giải quyết các bài toán tổ hợp, tối ưu hóa và tìm kiếm. Điển hình là bài toán "Knight’s Tour" (Hành trình của quân Mã trên bàn cờ vua), một bài toán cổ điển nhưng có tính ứng dụng cao trong các lĩnh vực như lập trình trò chơi, mô phỏng và nghiên cứu trí tuệ nhân tạo.

Trong báo cáo, tôi sẽ trình bày chi tiết cách tiếp cận bài toán, phân tích giải thuật Backtracking được sử dụng, và các bước thực hiện thuật toán để giải quyết vấn đề. Đồng thời, tôi cũng sẽ phân tích độ phức tạp của thuật toán và đề xuất hướng tối ưu hóa nhằm cải thiện hiệu suất. Báo cáo không chỉ mang ý nghĩa học thuật mà còn minh họa rõ ràng tính ứng dụng của giải thuật Backtracking trong việc giải quyết các bài toán phức tạp.

Cuối cùng, tôi xin gửi lời cảm ơn sâu sắc đến giảng viên **TS. Tạ Thị Kim Huệ**, người đã hướng dẫn tận tình và tạo điều kiện thuận lợi để tôi hoàn thành bài báo cáo này. Tôi rất mong nhận được sự góp ý từ quý thầy cô và các bạn để cải thiện và hoàn thiện báo cáo tốt hơn.

MỤC LỤC

BÀI TOÁN:

In ra tất cả các hành trình có thể của quân Mã trên bàn cờ vua

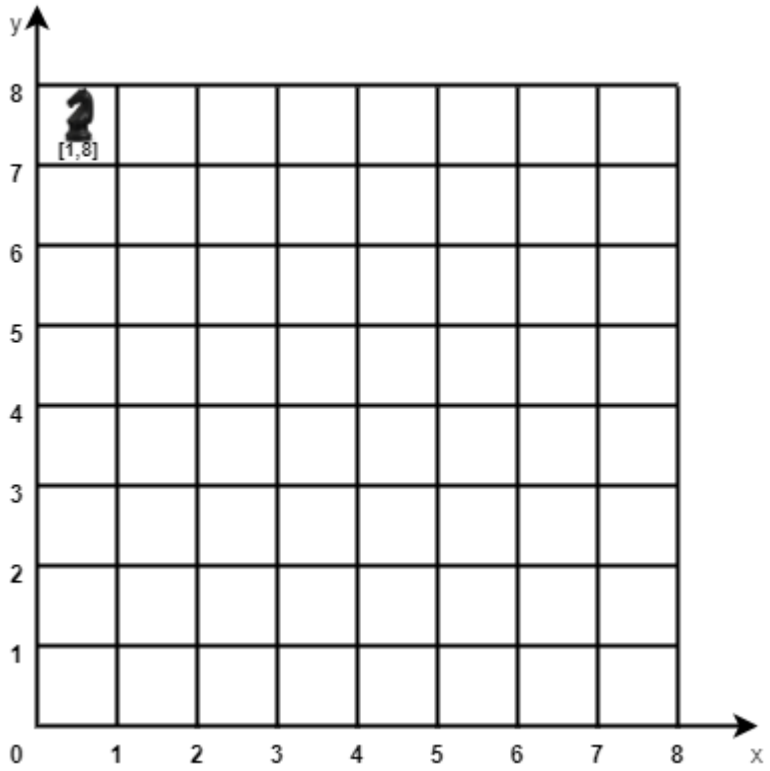
I. Mô tả bài toán

1. Mục đích

- Tìm đường đi cho quân mã để nó đi hết tất cả các vị trí trên bàn cờ vua sao cho các vị trí nó đi qua chỉ đúng một lần.

2. Đầu vào

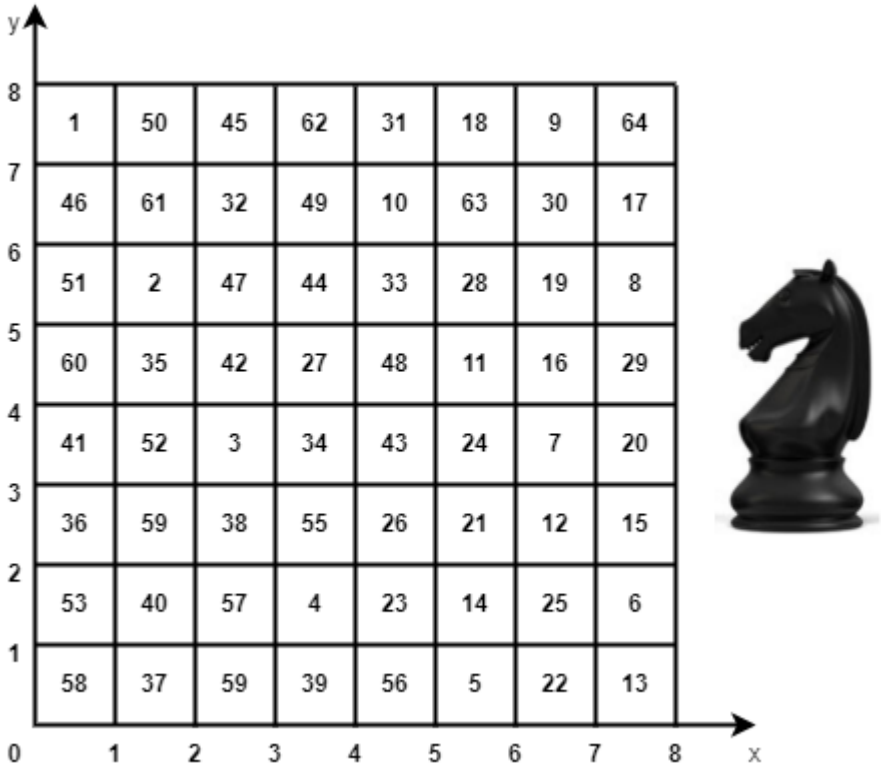
- Vị trí bất kì trên bàn cờ



- **Phân tích:**
  - Tại hình trên, ví dụ con mã xuất phát từ vị trí có toạ độ [1,8].
  - Cách di chuyển của con mã: Đầu tiên di chuyển ngang hoặc dọc 2 ô, tiếp theo di chuyển vuông góc với hướng còn lại có thể trái hoặc phải (lên hoặc xuống).

3. Đầu ra

- In ra thứ tự vị trí mà con mã đã đi qua và không được trùng vị trí nào.



- Phân tích: Bảng trên là 1 trong những kết quả về đường đi của quân mã khi đi hết bàn cờ.

II. Ý tưởng thuật toán

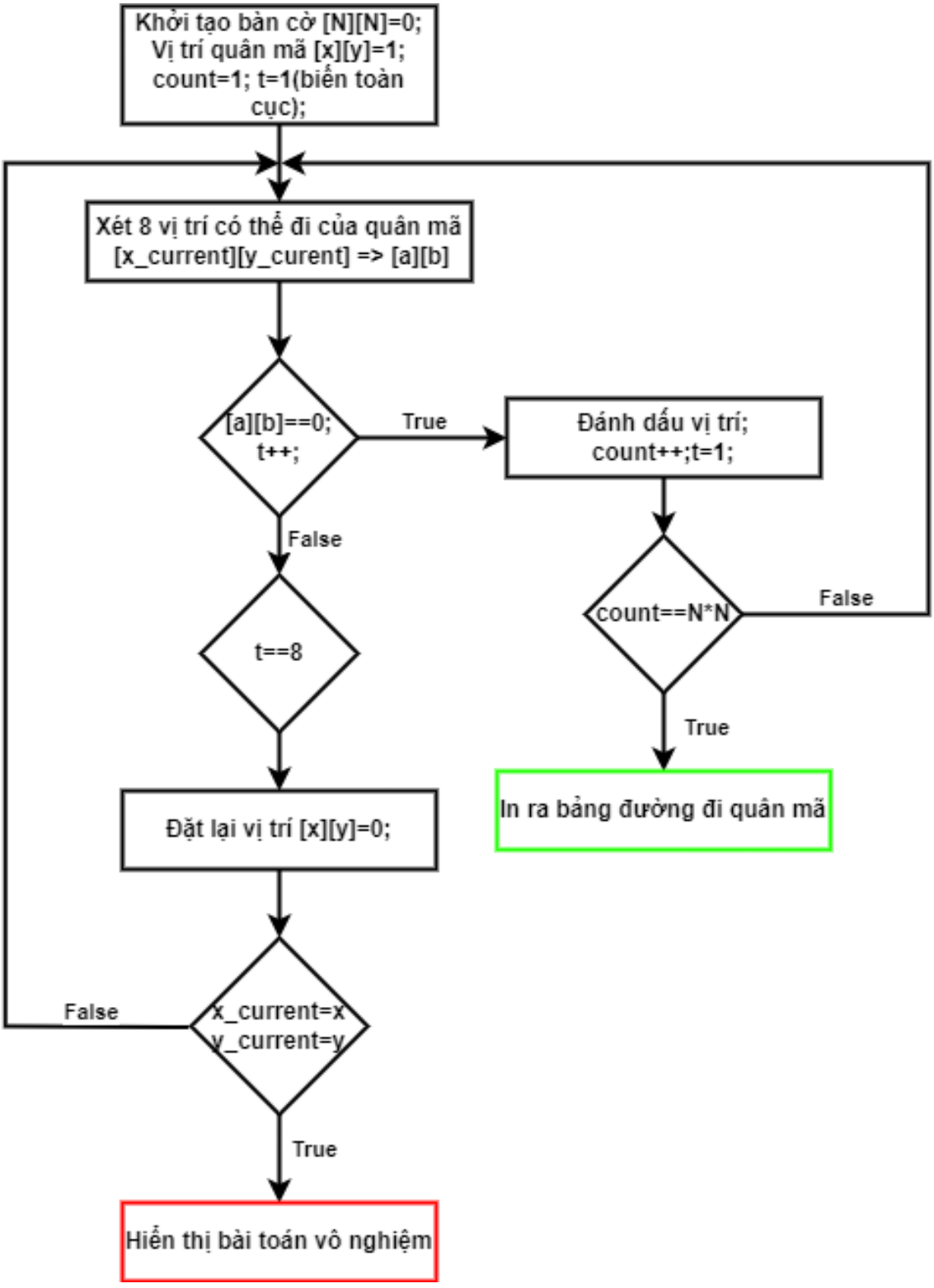
1. Nguyên tắc hoạt động

- Có 8 cách di chuyển cho quân mã khi nó ở vị (x,y): (x+2, y+1), (x+2, y-1), (x-2, y+1), (x-2, y-1),(x+1, y+2), (x+1, y-2), (x-1, y+2), (x-1, y-2). Lưu ý kiểm tra điều kiện $1 \leq x \leq 8; 1 \leq y \leq 8$.
- Di chuyển quân mã theo quy luật của nó, nếu không thoã mãn điều kiện thì quay lại tìm hướng đi khác.
- Mỗi lần di chuyển xong, đánh số thứ tự vào bảng Oxy.

2. Thuật toán

- Khởi tạo bàn cờ board[N][N] với tất cả giá trị là 0 (biểu thị các ô chưa đi qua), và vị trí ban đầu cho quân mã là (stx,sty)
- Với mỗi bước đi kiểm tra 8 trường hợp có thể từ ô hiện tại (x,y):
  - Nếu hợp lệ cập nhật vị trí đó so với thứ tự bước đi hiện tại và gọi đệ quy tiếp tục cho bước tiếp theo.
  - Nếu không hợp lệ thử lại bước khác
- Nếu không tìm được hành trình đầy đủ (vẫn còn ô "0" mà không thể tìm được đường đi tiếp theo cho quân mã) quay lui và đặt lại giá trị ô đó là "0".
- Kết thúc khi đi qua tất cả các ô (N x N), in ra bàn cờ. Trường hợp không tìm được hành trình, thông báo không có lời giải.

3. Lưu đồ thuật toán



III. Mô tả code

```cpp
#include <iostream>
using namespace std;
// Mảng chi tiết 8 nước đi của quân mã
int row[] = {2, 1, -1, -2, -2, -1, 1, 2};
int col[] = {1, 2, 2, 1, -1, -2, -2, -1};
// Hàm kiểm tra nước đi hợp lệ trên bàn cờ
bool isValid(int x, int y, int N) {
    return (x >= 0 && y >= 0 && x < N && y < N);
}
// Hàm quay lui thực hiện tìm tất cả các hành trình của quân mã
void knightTour(int **visited, int x_current, int y_current, int count, int N, int &totalSolutions) {
// Đánh dấu ô hiện tại đã được ghé thăm
   visited[x_current][y_current] = count;
   count++;
   // Nếu tất cả các ô đã được ghé thăm, in hành trình
   if (count == N * N + 1) {
       totalSolutions++;
       cout << "Hanh trinh " << totalSolutions << ":\n";
       // In bàn cờ theo chiều từ dưới lên trên và từ trái sang phải
       for (int i = 0; i < N; i++) {  // Duyệt từ dòng dưới cùng lên trên
           for (int j = 0; j < N; j++) {
               cout << visited[i][j] << " ";  // Các ô được in từ trái sang phải
           }
           cout << endl;
       }
       cout << endl;
       // Quay lui trước khi trả về
       visited[x_current][y_current] = 0;
       return;
   }
   // Duyệt các nước đi
   for (int t = 0; t < 8; t++) {
       int newX = x_current + row[t];
       int newY = y_current + col[t];
       // Nếu vị trí mới hợp lệ và chưa ghé thăm
       if (isValid(newX, newY, N) && visited[newX][newY] == 0) {
           knightTour(visited, newX, newY, count, N, totalSolutions);
       }
   }
// Quay lui: Nếu không có hành trình hợp lệ, đánh dấu lại ô hiện tại là chưa ghé thăm
```

```
41.    visited[x_current][y_current] = 0;
42.}
43.int main() {
44.    int N;
45.    cout << "Nhap kich thuoc ban co N: ";
46.    cin >> N;
47.    int startX, startY;
48.    cout << "Nhap toa do bat dau cua quan ma (x y): ";
49.    cin >> startX >> startY;
50.    // Điều chỉnh tọa độ sao cho (1,1) là góc dưới bên trái
51.    startX = N - startX;  // Dòng tăng từ dưới lên
52.    startY = startY - 1;  // Cột tăng từ trái sang phải
53.    // Kiểm tra vị trí bắt đầu có hợp lệ không
54.    if (!isValid(startX, startY, N)) {
55.        cout << "Toa do khong hop le! Vui long chay lai chuong trinh va nhp
   toa do hop le." << endl;
56.        return 1;
57.    }
58.    // Cấp phát mảng 2 chiều động cho bàn cờ
59.    int **visited = new int *[N];
60.    for (int i = 0; i < N; i++) {
61.        visited[i] = new int[N];
62.        for (int j = 0; j < N; j++) {
63.            visited[i][j] = 0;  // Khởi tạo tất cả các ô là chưa ghé thă
64.        }
65.    }
66.    int count = 1;  // Số lượng ô đã ghé thăm
67.    int totalSolutions = 0;
68.
69.    // Bắt đầu từ vị trí xuất phát do người dùng nhập
70.    knightTour(visited, startX, startY, count, N, totalSolutions);
71.    // In ra tổng số hành trình hợp lệ
72.    if (totalSolutions == 0) {
73.        cout << "Khong tim thay hanh trinh hop le." << endl;
74.    } else {
75.        cout << "Tong so hanh trinh hop le: " << totalSolutions << endl;
76.     }
77.     // Giải phóng bộ nhớ
78.     for (int i = 0; i < N; i++) {
79.         delete[] visited[i];
80.     }
81.     delete[] visited;
82.     return 0;
83.}
```

IV. Kiểm thử và nhận xét

1. Kiểm thử

- **Case 1:**
  - **Input:**

    Xét đầu vào là bàn cờ 5x5 và vị trí xuất phát của quân mã là [1][1]
  - **Output:**

    Sẽ có 304 cách đi cho quân mã nếu xuất phát từ vị trí này.

```
Hanh trinh 304:
3 12 7 18 25
6 17 4 13 8
11 2 19 24 21
16 5 22 9 14
1 10 15 20 23

Tong so hanh trinh hop le: 304
```

  - **Nhận xét:**

    Với bàn cờ 5x5, đã có rất nhiều cách để cho quân mã xuất phát và đi hết bàn cờ mà mỗi ô chỉ đi 1 lần.

- **Case 2:**
  - **Input:**

    Xét đầu vào là bàn cờ 4x4 và vị trí xuất phát của quân mã là [2][2].
  - **Output:**

    Không tìm thấy đường đi cho trường hợp này.

```
Nhap kich thuoc ban co N: 4
Nhap toa do bat dau cua quan ma (x y): 2 2
Khong tim thay hanh trinh hop le.
```

- **Nhận xét:**

  Phạm vi bàn cờ đã được thu hẹp 4x4, dẫn đến việc di chuyển toàn bộ mặt bàn cờ mà mỗi ô chỉ đi một lần là không thể.

2. Nhận xét:

- **Độ phức tạp của thuật toán:**
  - **Thời gian:**

    Thuật toán thử 8 khả năng di chuyển từ vị trí hiện tại. Với kích thước bàn cờ N x N thì độ phức tạp sẽ là: $O(8^{N*N})$
  - **Không gian:**

    Sử dụng mảng 2D để theo dõi các ô ⇒ Bộ nhớ chiếm $O(N^2)$
- **Ý nghĩa thực tiễn:**
  - Bài toán Knigh's Tour là bài toán kinh điển về thuật toán Backtracking, phương pháp giải quyết các bài toán tìm kiếm tất cả giải pháp và tối ưu hoá trong không gian tìm kiếm lớn.
- **Hạn chế:**
  - Độ phức tạp của bài toán khá cao khi N tăng lên.
  - Thuật toán tìm kiếm theo tuần tự nên mất nhiều thời gian.
  - Đối với bàn cờ nhỏ (1x1), bài toán có thể không có lời giải hợp lệ, thuật toán quay lui sẽ không dừng lại hợp lý hoặc không có thông báo rõ về trường hợp này.

TÀI LIỆU THAM KHẢO

[1] Bollwarm, "500+ Data Structures and Algorithms practice problems,"

Available online: https://github.com/bollwarm/DataStructuresAlgorithms?tab=readme-ov-file

[2] GeeksforGeeks, "Backtracking Algorithms,"

Available online: https://www.geeksforgeeks.org/backtracking-algorithms/

[3] Programiz, "Backtracking Algorithm,"

Available online: https://www.programiz.com/dsa/backtracking-algorithm