Contents

Homework 5: Trial and Error

Bùi Khánh Duy - 20001898

I. Lí thuyết phương pháp

Trình bày (ngắn gọn) ý tưởng và lược đồ tổng quát của 1 trong 3 phương pháp:

- Vét cạn (Brute Force/Exhaustive);
- Quay lui (Backtracking);
- Nhánh cận (Branch and Bound).

1. Vét cạn (Brute Force/Exhaustive):

Ý tưởng: Vét cạn là phương pháp duyệt tất cả các trường hợp có thể xảy ra để tìm kiếm giải pháp cho bài toán. Phương pháp này thường được sử dụng trong các bài toán có số lượng dữ liệu nhỏ và không yêu cầu tính toán phức tạp.

Lược đồ tổng quát:

1. Khởi tạo giá trị ban đầu.
2. Duyệt tất cả các trường hợp có thể xảy ra bằng vòng lặp hoặc đệ quy.
3. Kiểm tra điều kiện để xác định giải pháp.
4. Nếu tìm thấy giải pháp, trả về kết quả và kết thúc thuật toán.
5. Nếu không tìm thấy giải pháp, quay lại bước 2 và tiếp tục duyệt các trường hợp khác.

2. Quay lui (Backtracking):

Ý tưởng: Quay lui là phương pháp tìm kiếm giải pháp cho các bài toán kết hợp với việc lựa chọn và loại bỏ các phương án. Phương pháp này thường được sử dụng trong các bài toán tìm kiếm đường đi, sắp xếp, chia nhỏ và thống kê.

Lược đồ tổng quát:

1. Khởi tạo giá trị ban đầu.
2. Lựa chọn một phương án và thực hiện nó.
3. Kiểm tra điều kiện để xác định giải pháp.
4. Nếu tìm thấy giải pháp, trả về kết quả và kết thúc thuật toán.
5. Nếu không tìm thấy giải pháp, quay lại bước 2, loại bỏ phương án vừa chọn và thử phương án khác.

3. Nhánh cận (Branch and Bound):

Ý tưởng: Nhánh cận là phương pháp tìm kiếm giải pháp cho các bài toán tối ưu. Phương pháp này tìm kiếm các giải pháp tiềm năng và loại bỏ những giải pháp không cần thiết trước khi tìm kiếm các giải pháp khác.

Lược đồ tổng quát:

1. Khởi tạo giá trị ban đầu.
2. Tìm kiếm các giải pháp tiềm năng và tính toán giá trị của chúng.
3. Loại bỏ giải pháp không cần thiết và chỉ giữ lại những giải pháp tiềm năng tốt nhất.
4. Tạo các nhánh mới cho các giải pháp tiềm năng tốt nhất và tiếp tục tìm kiếm giải pháp trong các nhánh này.
5. Kiểm tra điều kiện để xác định giải pháp tối ưu.
6. Nếu tìm thấy giải pháp tối ưu, trả về kết quả và kết thúc thuật toán.
7. Nếu không tìm thấy giải pháp tối ưu, quay lại bước 3 và tiếp tục loại bỏ giải pháp không cần thiết.

II. Lập trình

Khớp xâu

Sử dụng thuật toán **Rabin-Karp**

Bài toán

Cho 2 xâu: xâu cần tìm s và đoạn văn bản t. Kiểm tra xem s có xuất hiện trong t hay không? Nếu có, liệt kê tất cả các vị trí của nó với độ phức tạp thời gian $O(|s| + |t|)$

Ý tưởng của thuật toán:

Sử dụng hàm băm (hash) cho xâu s. Tính giá trị của hàm băm với tất cả các tiền tố của xâu t. Sau đó có thể kiểu tra s với các xâu con có độ dài $|s|$ trong thời gian $O(1)$, tổng cộng sẽ mất $O(|t|)$.

Tổng độ phức tạp thời gian là $O(|t| + |s|)$.

Code (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> hashing(T + 1, 0);
    for (int i = 0; i < T; i++)
        hashing[i+1] = (hashing[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurrences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (hashing[i+S] + m - hashing[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurrences.push_back(i);
    }
    return occurrences;
}
```

Dãy nhị phân

Liệt kê các dãy nhị phân có độ dài n.

Có 2 cách code:

C1: Đệ quy

```python
def generateBinary(n, current):
    if n == 0:
        result.append(current)
        return
    for i in range(2):
        generateBinary(n-1, current + str(i))

def binary_list(n):
        generateBinary(n, "")
        return result
result = []
```

```
binary_list(5);
print(result)
```

Kết quả

```
['00000', '00001', '00010', '00011', '00100', '00101', '00110', '00111', '01000', '01001', '01010', '01
```

C2: Khử đệ quy

```
def binary_list2(n):
    result = []
    for i in range(1<<n):
        result.append(bin(i)[2:].zfill(n))
    return result
print(binary_list2(5))
```

```
['00000', '00001', '00010', '00011', '00100', '00101', '00110', '00111', '01000', '01001', '01010', '01
```

Cả 2 đều có độ phức tạp thuật toán $O(2^n)$

III. Đặt bài toán, thiết kế, phân tích và triển khai thuật toán

Lưu ý: Ở đây sẽ dùng ngôn ngữ C++

Bài toán

Hãy chỉ ra một cách đi của con mã để xuất phát từ một ô cờ (x, y) nào đó $(x, y = \overline{1...8})$ có thể đi qua tất cả các ô cờ, mỗi ô duy nhất một lần.

Knight's tour

Tên tiếng anh: **Knight tour**

Tên tiếng việt: Mã đi tuần.

Mô tả bài toán

Cho bàn cờ $N \times N$ với quân mã đặt ở 1 vị trí bất kì (để cho đơn giản, quân mã được đặt ở vị trí $1, 1$). Quân mã cần di chuyển trên bàn cờ theo đúng luật, với mỗi ô được đi qua đúng 1 lần. In ra thứ tự của các ô được đi qua.

```
Input :
N = 8
Output:
0  59  38  33  30  17   8  63
37  34  31  60   9  62  29  16
58   1  36  39  32  27  18   7
35  48  41  26  61  10  15  28
```

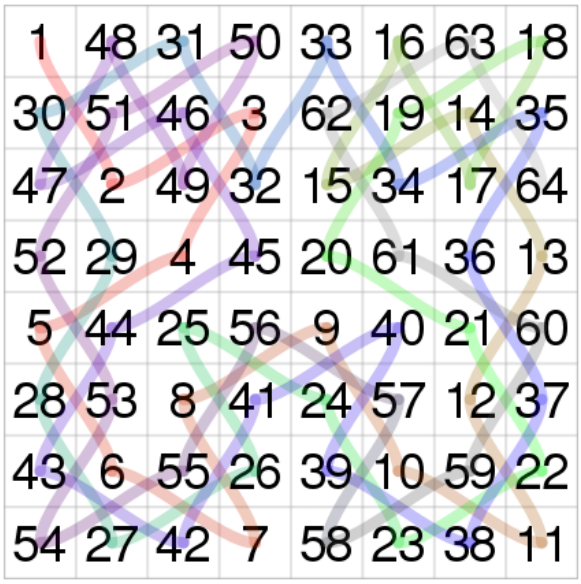

Figure 1: Một đáp án giải cho bài toán này (bàn cờ 8x8)

```
42  57   2  49  40  23   6  19
47  50  45  54  25  20  11  14
56  43  52   3  22  13  24   5
51  46  55  44  53   4  21  12
```

Phân tích

Quân mã ở đỉnh (i, j) có thể di chuyển đến 8 đỉnh khác

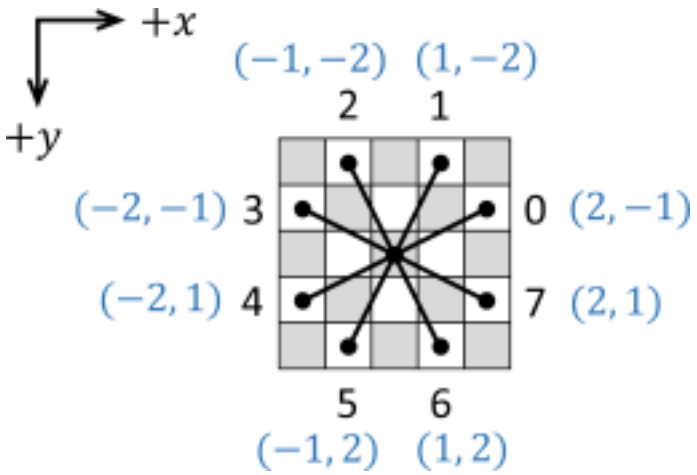

Figure 2: Các hướng của quân mã

Từ đó ta có thể định nghĩa trước các cặp (dx, dy) tương ứng với 8 hướng có thể đi

```
int dx[] = {-2, -2, -1, -1, 1, 1, 2, 2};
int dy[] = {-1, 1, -2, 2, -2, 2, -1, 1};
```

Các hướng giải quyết

Backtracking

Ý tưởng giải bằng phương pháp này khá đơn giản, có các bước như sau

1. Thêm 1 đỉnh vào tập lời giải, đánh dấu lại đỉnh này đã đi qua và đi đến một trong 8 hướng còn lại
2. Nếu đỉnh hiện tại không đi đủ thành 1 vòng ~ nghiệm sai quay lại để giải theo hướng khác, đồng thời cũng xoá các trạng thái của đường đi đang có.
3. Sau khi duyệt hết tất cả các trường hợp, nếu không có kết quả In ra màn hình `Not found!`, ngược lại in ra ma trận có dạng như ở ví dụ

```
#include<vector>
#include<utility>
#include<iostream>
using namespace std;
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int N;
int dx[] = {-2, -2, -1, -1, 1, 1, 2, 2};
int dy[] = {-1, 1, -2, 2, -2, 2, -1, 1};

pair<int, int> operator+ (const pair<int, int>& a, const pair<int, int>& b) {
    return {a.first + b.first, a.second + b.second};
}
bool operator== (const pair<int, int>& a, const pair<int, int>& b) {
    return a.first == b.first && a.second == b.second;
}
vector<pair<int, int>> path, ans;
// vector<vector<pair<int, int>>> ans;
vector<vector<bool>> check;
void solve();
void out() {
    ans = path;
}
bool found = false;
bool inside(pair<int, int> x) {
    return 0 <= x.first && x.first < N && 0 <= x.second && x.second < N;
}
void backtrack(pair<int, int> pos) {
    if (found) return;
    for(int i =0; i < 8; ++i) {
        pair<int, int> curr = pos+make_pair(dx[i], dy[i]);
        if (inside(curr)) {
            if (!check[curr.first][curr.second]) {
                check[curr.first][curr.second] = 1;
                path.push_back(curr);
                backtrack(curr);
                check[curr.first][curr.second] = 0;
```

```
                path.pop_back( );
            }
            else if (curr == make_pair(0, 0) && path.size() == N*N) {
                out();
                found = true;
            }
        }
    }
}
int main()
{
    IOS;
    solve();
}
void solve() {
    cin >> N;
    check.resize(N, vector<bool>(N, 0));
    check[0][0] = 1;
    path.push_back({0, 0});
    backtrack({0, 0});
    cout << ans.size() <<"\n";
    // print(ans);
    if (!found) {
        cout << "Not found!";
    } else {
        if (ans.size() != 0) {
            vector<vector<int>> grid(N, vector<int>(N));
            int i = 0;
            for(auto &it: ans) {
                grid[it.first][it.second] = i++;
            }
            for(auto &it: grid) {
                for(auto &it2: it) {
                    cout << it2 << " ";
                }
                cout << "\n";
            }
        }
    }

}
```

Phương pháp này có thể giải cho bàn cờ 6×6, kết quả thu được như sau:

```
0 7 4 19 2 9
5 18 1 8 29 20
14 35 6 3 10 31
17 24 15 30 21 28
34 13 26 23 32 11
25 16 33 12 27 22
```

Các trường hợp < 6 đều ra `Not found!`

Tuy nhiên, với trường hợp bàn cờ 8×8 thì chương trình chạy rất lâu.

Hãy phân tích độ phức tạp của thuật toán này. Bỏ qua các bước chi tiết như tạo mảng, copy mảng v.v...

Bàn cờ $N \times N$ có N^2 ô, và ở mỗi vị trí, ta có tối đa 8 nước đi được. Vậy nên độ phức tạp thời gian là $O(8^{N^2})$

Ngoài ra độ phức tạp không gian là $O(N^2)$

Ở trường hợp $N = 8 \quad 8^{N^2} = 6.2771017354 \times 10^{57}$ Rất rất lớn. Phương pháp này không đáp ứng đủ.

Thuật toán Warnsdorff

Nhìn sâu vào bài toán mã đi tuần, đây không chỉ đơn giản là bài toán duyệt thông thường. Nó có yếu tố đồ thị ở trong đấy. Việc sử dụng lý thuyết đồ thị cũng là thành phần chính cho cách giải này.

Có 2 quy tắc như sau:

1. Điểm xuất phát có thể là vị trí bất kì trên bàn cờ.
2. Quân mã luôn di chuyển đến ô cạnh nó (theo luật), chưa từng đi qua với số bậc nhỏ nhất.

(Số bậc của 1 đỉnh là số lượng những đỉnh có thể đi được từ nó và cũng chưa đi qua)

Các bước giải như sau:

1. Đặt điểm xuất phát (0, 0) (hoặc bất kì)
2. Đánh dấu là đỉnh đã đi qua, có giá trị 0.
3. Thực hiện các thao tác sau từ giá trị 2 đến $N \times N - 1$:
   - Gọi S là đỉnh đi được từ đỉnh P
   - Đánh dấu P đã đi qua
   - Đặt $S = P$
4. Trả về kết quả là bàn cờ đã được đánh các ô.

```cpp
#include <bits/stdc++.h>
using namespace std;
int N = 8;

const int dx[8] = {1,1,2,2,-1,-1,-2,-2};
const int dy[8] = {2,-2,1,-1,2,-2,1,-1};

bool inside(int x, int y)
{
    return ((x >= 0 && y >= 0) && (x < N && y < N));
}

bool isValid(int a[], int x, int y)
{
    return (inside(x, y)) && (a[y*N+x] < 0);
}

int getDegree(int a[], int x, int y)
{
    int count = 0;
    for (int i = 0; i < N; ++i)
        if (isValid(a, (x + dx[i]), (y + dy[i])))
            count++;
```

```
    return count;
}

bool nextMove(int a[], int *x, int *y)
{
    int min_deg_idx = -1, c, min_deg = (N+1), next_x, next_y;

    int start = rand()%N;
    for (int count = 0; count < N; ++count)
    {
        int i = (start + count)%N;
        next_x = *x + dx[i];
        next_y = *y + dy[i];
        if ((isValid(a, next_x, next_y)) && (c = getDegree(a, next_x, next_y)) < min_deg)
        {
            min_deg_idx = i;
            min_deg = c;
        }
    }

    if (min_deg_idx == -1)
        return false;

    next_x = *x + dx[min_deg_idx];
    next_y = *y + dy[min_deg_idx];

    a[next_y*N + next_x] = a[(*y)*N + (*x)]+1;

    *x = next_x;
    *y = next_y;

    return true;
}

void print(int a[])
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
            cout << a[j*N+i] << " \n"[j == N-1];
    }
}

bool neighbour(int x, int y, int xx, int yy)
{
    for (int i = 0; i < N; ++i)
        if (((x+dx[i]) == xx)&&((y + dy[i]) == yy))
            return true;

    return false;
}
int loopCount = 0;
bool found = 0;
```

```
bool findClosedTour()
{
    int a[N*N];
    for (int i = 0; i< N*N; ++i)
        a[i] = -1;

    int sx = 0;
    int sy = 0;

    int x = sx, y = sy;
    a[y*N+x] = 0; // Mark first move.

    for (int i = 0; i < N*N-1; ++i)
        if (nextMove(a, &x, &y) == 0)
            return false;

    if (!neighbour(x, y, sx, sy))
        return false;

    found = true;
    print(a);
    return true;
}

int main()
{
    srand(time(NULL));

    // While we don't get a solution
    while (!findClosedTour() || loopCount++ > 1000)
    {
    ;
    }
    if (!found) cout << "Not found!";
    return 0;
}
```

Output:

```
(base) [09:36:01] [~/Documents/mycodes/Classes/Algorithm_Design_and_Analysis/week5] [main ]    ./a
0 15 60 33 2 17 20 35
59 32 1 16 51 34 3 18
14 63 48 61 44 19 36 21
31 58 43 52 47 50 41 4
56 13 62 49 42 45 22 37
27 30 57 46 53 40 5 8
12 55 28 25 10 7 38 23
29 26 11 54 39 24 9 6
[09:36:02] [cost 0.630s] ./a
```

Về bản chất, đây là thuật toán sử dụng phương pháp Heuristic bằng cách đánh giá đỉnh có thể đi tiếp bằng số bậc nhỏ nhất. Ngoài ra còn áp dụng những `random()` ở những đoạn nhất định.

Độ phức tạp thời gian: $O(N^2 \log(N))$

Độ phức tạp không gian: $O(N^2)$

Chia để trị

Việc giải bằng phương pháp này được lấy cảm hứng từ paper: "An efficient algorithm for the Knight's tour problem"

Thuật toán Warnsdorff có tốc độ chạy rất nhanh với $N \geq 6$, nhưng vì là 1 thuật toán Heuristic nên không phải lúc nào cũng cho ra kết quả. Theo paper A SIMPLE ALGORITHM FOR KNIGHT'S TOURS của "Sam Ganzfried: "[…]The algorithm produces a successful tour over 85% of the time on most boards with m less than 50, and it succeeds over 50% of the time on most boards with N less than 100. However, for $N > 200$ the success rate is less than 5%, and for $N > 325$ there were no successes at all. These observations suggest that the success rate of Warnsdorff's random algorithm rapidly goes to 0 as N increases." Tạm dịch: Thuật toán có tỉ lệ thành công 85% với $N \leq 50$ và chỉ còn 50% với bàn cờ có kích cỡ nhỏ hơn 100. Nhưng với kích cỡ $N > 325$ thì tỉ lệ = 0%. Vậy nên tỉ lệ thành công của thuật toán Warnsdorff đi về 0 khi N tăng lên. Minh hoạ qua hình ảnh sau:

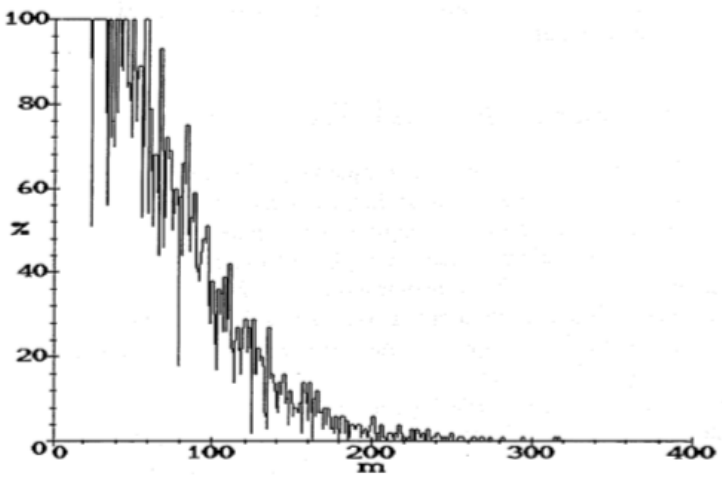

Figure 3: Hình 1.

Theorem 2.1. For all even $n \geq 6$ there exists a structured knight's tour on an $n \times n$ and an $n \times (n+2)$ board. Such a tour can be constructed in time $O(n^2)$.

Định lý 2.1 khẳng định sẽ luôn tồn tại đường đi cho con mã trong bàn cờ chẵn với kích cỡ $N \geq 6$, có thể được dựng trong $O(N^2)$.

Để thực hiện, ta sẽ tách bàn cờ lớn về những bàn cờ nhỏ hơn, cho đến khi về trạng thái cơ sở: bàn cờ có kết quả được tính từ trước. Đó là trạng thái 8×8. Ta cũng sẽ chỉ quan tâm tới các bàn cờ là bội số của 8.

Như đã nói ở trên, khi có đáp án cho trạng thái cơ sở, ta có thể giải được với $N = 8k$, $k \in \mathbb{N}$. Bằng cách chia nhỏ thành 4 bàn cờ, và đệ quy cho đến khi $N = 8$.

Không mất tính tổng quát, xét vị trí xuất phát ở góc dưới cùng bên trái (phần D của hình 3(b)). Một khi nước đi chạm rìa trên cùng, thay vì đi tiếp đến đỉnh D (vẫn ở hình 3(b)) và hoàn thành nước đi, ta đi tiếp sang đỉnh E (hình 3(c)) và xử lý ở phần bàn ở góc trên trái. Cứ thế từ E sang F, rồi F sang G… Và hoàn thành chuyến đi.

Đây là hình ảnh minh hoạ cho phương pháp này

Lưu ý rằng mỗi khi chuyến đi bị đóng lại như trường hợp cơ sở. Tính đóng của bàn cờ cho phép chúng ta di chuyển sang các góc khác cho đến khi hết bàn.

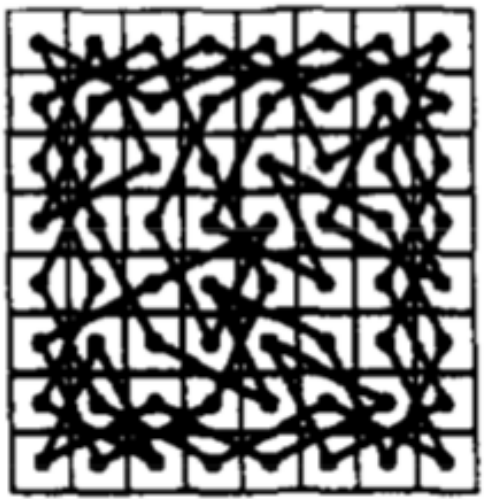

Figure 4: Hình 2.

Phần code ở trong file đính kèm bài, ở đây sẽ đưa ra các output của chương trình:

```
n = 8
board = Chessboard(n,n)

print(board.GetRows())
print(board.GetColumns())

print("Building knight path list...")
board.FindPathList()
print("Building knight tour...")
print()
board.FindTour()
board.PrintTour()

n = 128
board = Chessboard(n,n)

print("Building knight path list...")
board.FindPathList()
print("Building knight tour...")
print()
board.FindTour()
board.PrintTour()

print("Path length:", len(board.GetPathList()))
print("Tot. squares:", board.GetColumns() * board.GetRows())

board.CheckTour()
```

Output

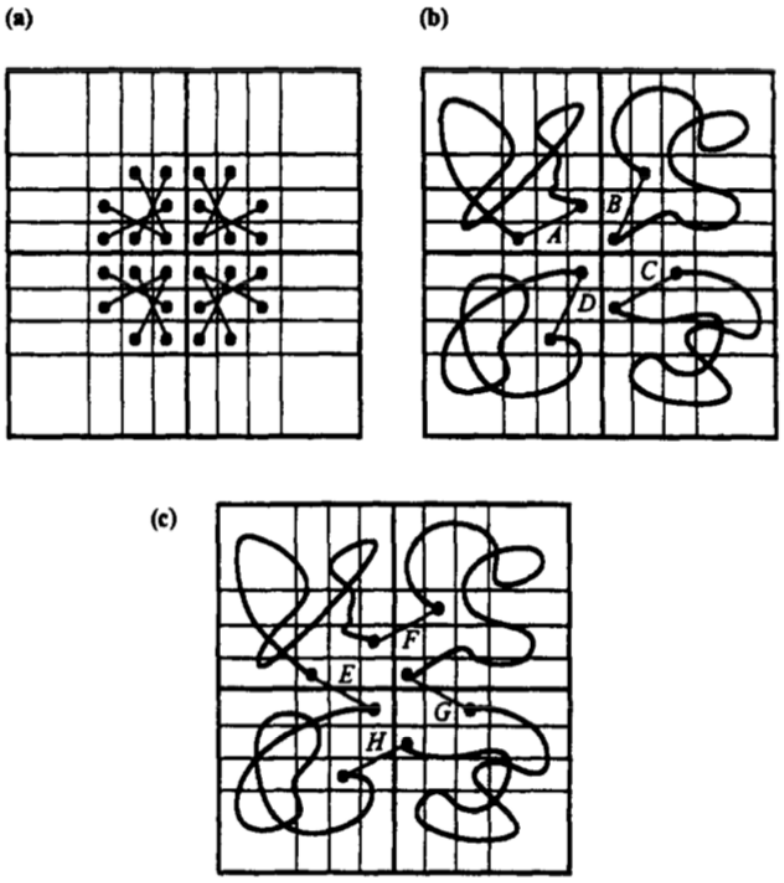

Fig. 3. How to combine four structured knight's tours into one: (a) the moves at the inside corners, (b) the edges A, B, C, D to be deleted, and (c) the replacement edges E, F, G, H.

Figure 5: Hình 3.

```
(base) [10:27:19] [~/Documents/mycodes]    /opt/homebrew/bin/python3 /Users/delus/Documents/mycodes/Clas
8
8
Building knight path list...
Building knight tour...

[[26  7 42 11 28 31 56 13]
 [43 10 27 32 55 12 17 30]
 [ 6 25  8 41 52 29 14 57]
 [ 9 44 33 54 35 16 51 18]
 [24  5 36 47 40 53 58 15]
 [37  2 45 34 61 48 19 50]
 [ 4 23 64 39 46 21 62 59]
 [ 1 38  3 22 63 60 49 20]]
Building knight path list...
Building knight tour...

[[ 3954  3973  3938 ...  7891  7916  7873]
 [ 3937  3970  3953 ...  7872  7877  7890]
 [ 3974  3955  3972 ...  7889  7874  7917]
 ...
 [16357     2 16365 ... 12118 12153 12120]
 [    4 16343 16384 ... 12155 12132 12129]
 [    1 16358     3 ... 12130 12119 12154]]
Path length: 16384
Tot. squares: 16384
Complete:        True
Legal:           True
[10:27:20] [cost 0.624s] /opt/homebrew/bin/python3 /Users/delus/Documents/mycodes/Classes/Algorithm_Des
```

Với $n = 128$ chỉ mất ≈ 0.63s (code thuật toán **Warnsdorff** cũng mất thời gian tương tự nhưng chỉ giải được $n = 8$).

Neural Network

Trong quá trình nghiên cứu để làm bài tập, em có đọc qua được những bài viết về giải bài toán này bằng cách dùng Mạng Neural. Phương pháp này giúp giải những bài toán có kích thước bàn cờ N lớn. Ở đây là link tham khảo chứ không đi sâu vào

https://avinayak.github.io/programming/algorithm/2022/04/01/solving-knights-tour-using-a-neural-network.html