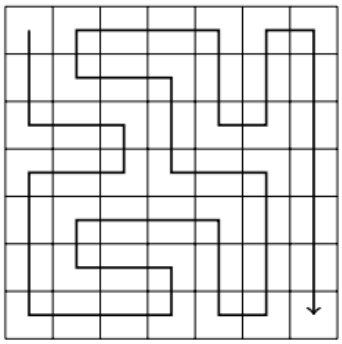
Đếm số đường đi Hamilton trong lưới $n \times n$

Ngày 2 tháng 10 năm 2025

Giới thiệu

Hãy xem xét bài toán tính số lượng đường đi trong một lưới $n \times n$ từ góc trên bên trái đến góc dưới bên phải sao cho đường đi đó đi qua **mỗi ô đúng một lần**. Ví dụ, trong lưới 7×7, có 111712 đường đi như vậy. Một ví dụ về một đường đi như sau (chỉ minh hoạ):



Hình 1: Đường đi trong lưới 7×7 (minh hoạ).

Chúng ta tập trung vào trường hợp 7×7, vì mức độ khó của nó phù hợp với nhu cầu của chúng ta. Chúng ta bắt đầu với một thuật toán *backtracking* cơ bản, sau đó từng bước tối ưu hoá bằng các quan sát về cách có thể *prune* (cắt tỉa) quá trình tìm kiếm. Sau mỗi tối ưu hoá, ta đo thời gian chạy và số lượng *recursive calls* để thấy rõ tác động của từng tối ưu lên hiệu quả tìm kiếm.

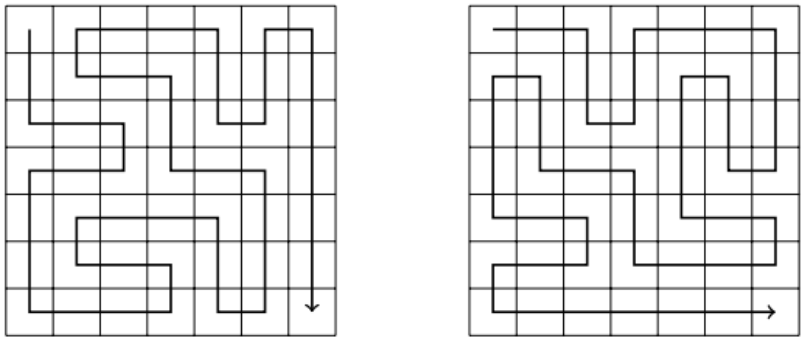
1 Thuật toán cơ bản

Phiên bản đầu tiên của thuật toán không chứa bất kỳ tối ưu hoá nào. Ta đơn giản dùng *backtracking* để sinh tất cả các đường đi có thể từ góc trên bên trái đến góc dưới bên phải và đếm số lượng đường đi như vậy.

- **Running time**: 483 seconds.
- **Number of recursive calls**: 76 billion.

2 Optimization 1

Trong bất kỳ lời giải nào, bước đầu tiên luôn là đi xuống một ô hoặc sang phải một ô. Luôn tồn tại hai đường đi đối xứng nhau qua đường chéo của lưới sau bước đầu tiên (*symmetry*). Vì vậy, ta có thể quy ước luôn đi xuống (hoặc sang phải) ở bước đầu, rồi cuối cùng nhân đôi số lượng lời giải.

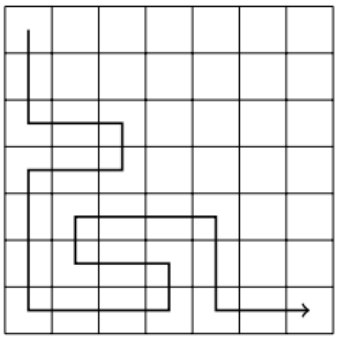


Hình 2: Hai đường đi đối xứng qua đường chéo (minh hoạ).

- **Running time**: 244 seconds.
- **Number of recursive calls**: 38 billion.

3 Optimization 2

Nếu đường đi đến ô góc dưới bên phải *trước* khi đã đi qua tất cả các ô khác trong lưới, thì rõ ràng không thể hoàn thành lời giải. Khi phát hiện trường hợp này, ta có thể dừng tìm kiếm ngay lập tức.

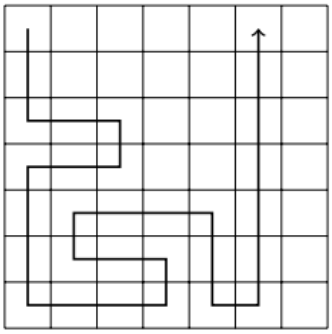


Hình 3: Đến đích quá sớm nên không thể hoàn thành (minh hoạ).

- **Running time**: 119 seconds.
- **Number of recursive calls**: 20 billion.

4 Optimization 3

Nếu đường đi chạm vào tường và có thể rẽ trái *hoặc* rẽ phải, lưới sẽ bị chia thành hai phần riêng biệt chứa các ô chưa được đi qua. Trong trường hợp này, không thể đi hết tất cả các ô nên có thể dừng tìm kiếm.

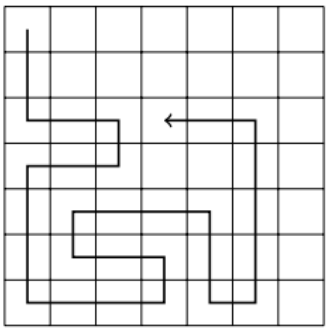


Hình 4: Chạm tường và có thể rẽ trái/phải ⇒ chia vùng (minh hoạ).

- **Running time**: 1.8 seconds.
- **Number of recursive calls**: 221 million.

5 Optimization 4

Tổng quát hoá ý tưởng của Optimization 3: nếu đường đi không thể tiếp tục tiến lên phía trước nhưng có thể rẽ trái *hoặc* rẽ phải, lưới sẽ bị chia thành hai phần riêng biệt và cả hai đều chứa các ô chưa được đi qua. Rõ ràng ta không thể đi qua tất cả các ô nữa, nên có thể dừng tìm kiếm.



Hình 5: Không thể tiến lên, chỉ có thể rẽ trái/phải ⇒ chia vùng (minh hoạ).

- **Running time**: 0.6 seconds.
- **Number of recursive calls**: 69 million.