1 - BTREE

Ta có các nhận xét sau:

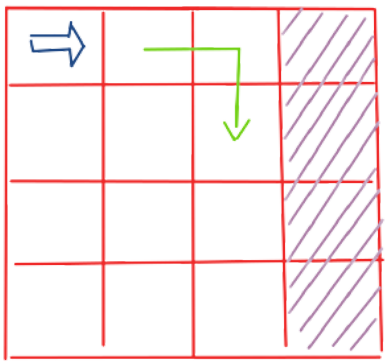
- Số cặp cạnh chéo nhau chính là số nghịch thế của hoán vị **P** thỏa mãn **P(i)** ở cây 2 có cạnh nối với **i** ở cây 1 (**i ≤ 2^d**).
- Mỗi nút trên cây tương ứng với một đoạn trên mảng và phép đổi chỗ hai nút con của một nút tương ứng với việc đổi chỗ hai nửa của đoạn tương ứng.
- Việc đổi chỗ hai nút con của một nút không ảnh hưởng đến các nút khác.

Từ đây ta có thuật toán đơn giản là với mỗi nút không phải nút lá ta thử đổi chỗ hai nút con của nó và lấy kết quả tốt hơn. Có thể sử dụng **BIT** để tính số nghịch thế, độ phức tạp là **O(2^d*d^2).**

2 - WGAME

Ta sẽ sử dụng quy hoạch động cho bài này. Trạng thái quy hoạch động của chúng ta cần lưu thông tin về các ô mà ta còn đi được, vị trí và hướng của chúng ta hiện tại, phần so khớp với xâu **P**.

Ta nhận thấy các ô ta còn đi đến được luôn tạo thành 1 hình chữ nhật, bởi vì lúc ta rẽ phải thì luôn có 1 hình chữ nhật bị cắt đi khỏi các ô hiện tại (như hình vẽ).



Từ đây ta có thể rằng chỉ cần lưu 4 biên là đủ để quản lý các ô còn đi được, để tối ưu bộ nhớ thì ta luôn cho ô đang đứng hiện tại nằm ở góc của hình chữ nhật, lúc này ta chỉ cần lưu vị trí hiện tại và 2 biên đối diện.

Về phần so khớp xâu ta có thể làm giống như thuật toán **KMP,** lưu lại phần hậu tố dài nhất mà bằng với tiền tố của **P.**

Tổng hợp lại, trạng thái quy hoạch động bao gồm 6 tham số như sau:

- **L R B T** : Lưu các hàng (cột) bị giới hạn về 4 phía (**L**, **R ≤ N**; **B**, **T ≤ M**), trong đó ô đang đứng hiện tại nằm ở một góc của hình chữ nhật.
- **U** : phần hậu tố dài nhất đang khớp với một tiền tố của **P** (**U ≤ |P|**).
- **V**: hướng đang đi (**V < 4**).

Để chuyển trạng thái, ta for số bước mà ta sẽ đi tiếp trước khi rẽ phải. Ta cài thêm một mảng **nxt[U][c]** là độ dài phần khớp sau khi thêm ký tự **c** nếu phần khớp hiện tại là **U** để tính lại phần khớp nhanh chóng, mảng này có thể được tính trong **O(|P| * 26)**:

```
int k = P.size();
for (int i = 0, j = 0; i < k; i++) {
   j = nxt[j][P[i] - 'A'];
   nxt[i][P[i] - 'A'] = i + 1;
   for (int c = 0; c < 26; c++)
       nxt[i + 1][c] = nxt[j][c];
}
for (int c = 0; c < 26; c++)
   nxt[k][c] = k;
```

Đối với bài này ta nên sử dụng đệ quy có nhớ để cài đặt được dễ dàng.

3 - PZZ

Subtask 1: **N ≤ 3**
Ở đây bảng chỉ có 9 ô nên số trạng thái tối đa là **9! = 362880** khá nhỏ. Ta có thể dùng thuật toán **BFS** để giải quyết subtask này.

Subtask 2: **N ≤ 5**
Hướng thực hiện của chúng ta là tìm cách xếp các hàng trên và cột trái để đưa thu gọn bảng thành 3 x 3 và đưa về subtask trên.

Một hướng giải có thể tham khảo như sau:
Ta sẽ sắp các hàng trên trước rồi đến các cột trái.
Với mỗi hàng (cột) ta sắp lần lượt theo mô hình (coi ô **N * N** là ô trống):

. . . . 1 -> . . . 1 2 -> … -> 1 2 … **N**

Từ . . . 1 2 -> . . 1 2 3 ta có thể làm như sau:

- Đưa số 3 xuống ngay dưới số 2, đảm bảo ô trống không đi vào những ô đã xếp.

. . . 1 2
. . . . 3

- Đưa ô trống đến ô đầu tiên của hàng đang xếp rồi đẩy nó đến ô cuối cùng của hàng, sau đó đưa số 3 lên.

  . . 1 2 *
  3

- Làm tương tự với phần còn lại.

Ta cần cài đặt 2 thao tác sau:

- Đưa ô trống đến một vị trí khác trong lúc đó không được thay đổi các ô đã xếp trước đó, điều này có thể làm được bằng cách **BFS** hoặc **DFS** từ vị trí ô trống.
- Đưa một số đến một vị trí khác, điều này làm được bằng cách liên tục đưa ô trống đến bên cạnh số đó và đổi chỗ.