LECTURE 03

BACKTRACKING

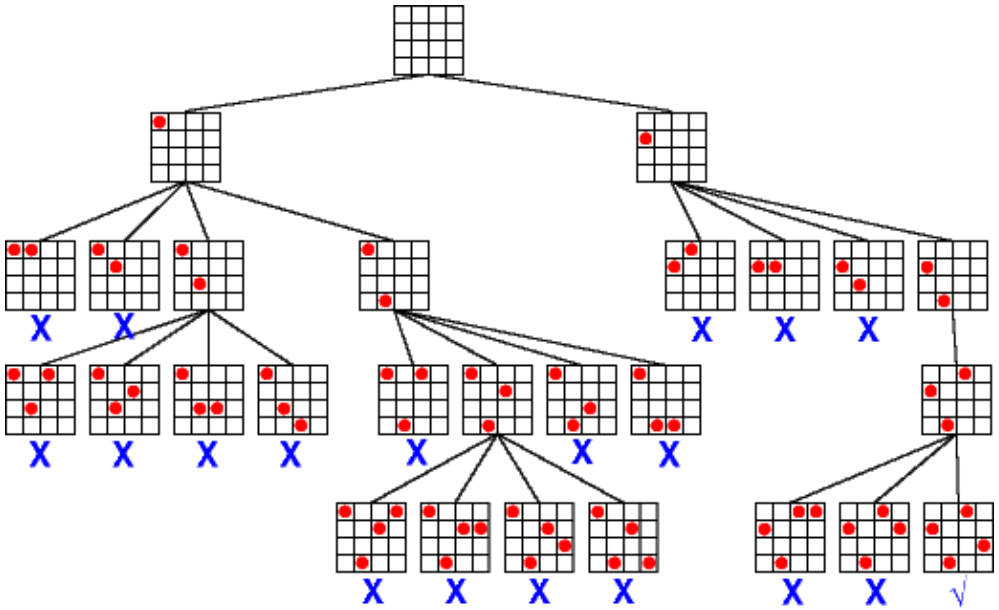
Big-O Coding

Website: www.bigocoding.com

Giới thiệu tổng quan

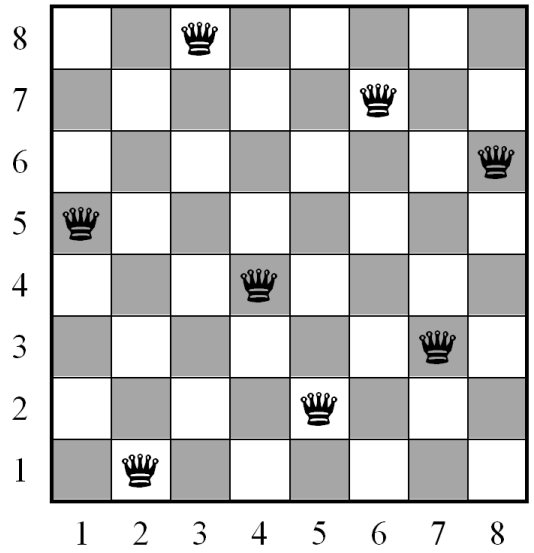
Backtracking (quay lui): là kĩ thuật thiết kế giải thuật dựa trên phương pháp đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, cho đến khi nào đi hết tất cả các trường hợp, rồi lại quay lại trường hợp trước đó để tìm tiếp lời giải theo hướng khác.

Quá trình đó cứ lặp đi lặp lại, cho đến khi nào đáp ứng hết yêu cầu bài toán thì sẽ dừng lại và xuất kết quả.



Bài toán N-Queen

N-Queen (bài toán N hậu): Trên bàn cờ vua có kích thước NxN (N hàng và N cột). Tìm cách đặt N quân hậu lên bàn cờ sao cho không quân nào được ăn quân nào.

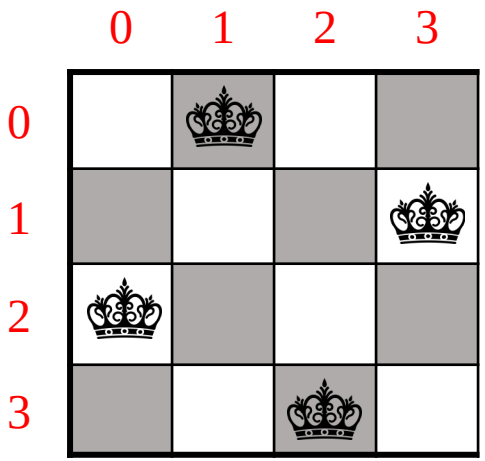


** **Lưu ý:** Quân hậu có thể ăn các quân cùng hàng, cùng cột, hoặc cùng đường chéo chính, đường chéo phụ.*

Bài toán 4-Queen

Bài toán minh họa: Bàn cờ có kích thước N = 4 (4x4).

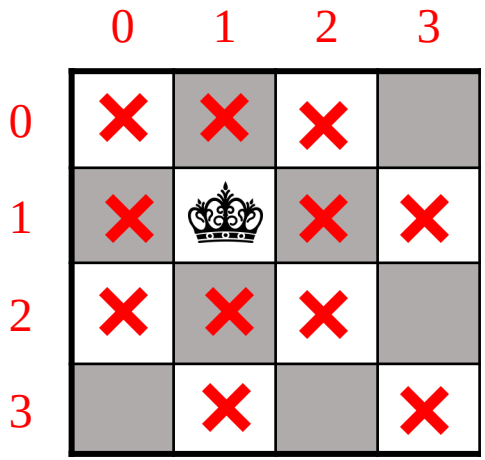
Làm thế nào để đặt 4 quân hậu lên bàn cờ này.



Phân tích bài toán

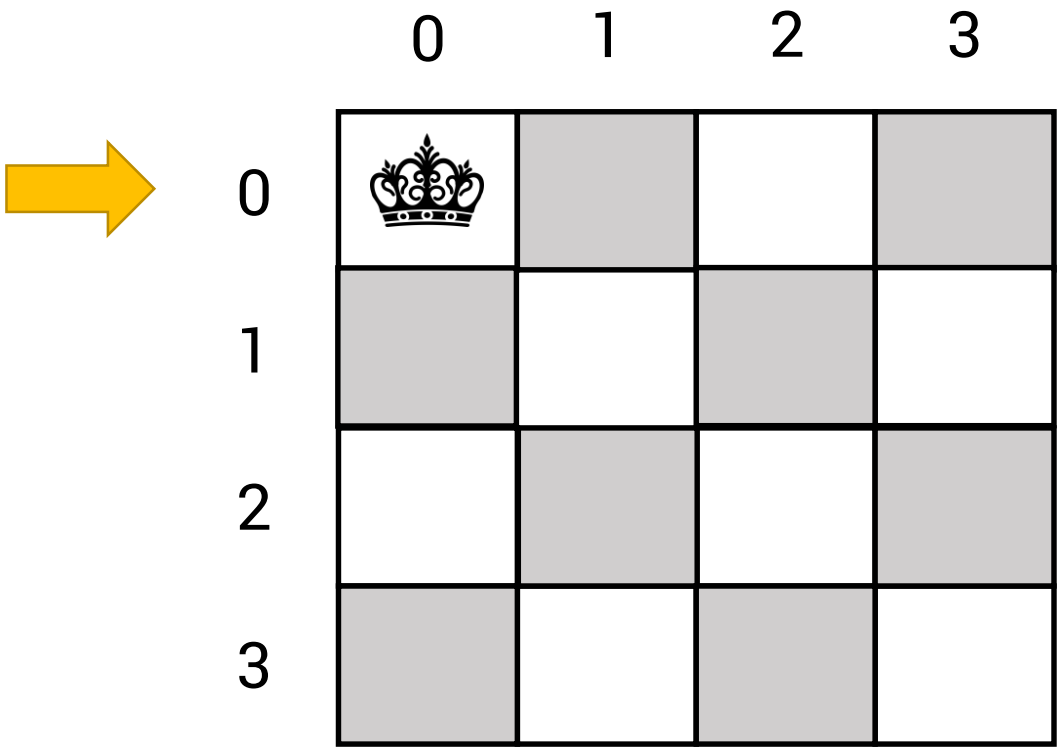
Khi quân hậu được đặt vào một vị trí (1, 1)trên bàn cờ thì:

- **Dòng:** toàn bộ **dòng 1** sẽ không được phép đặt quân nào khác.
- **Cột:** toàn bộ **cột 1** sẽ không được phép đặt quân nào khác.
- **Chéo chính:** ô **(0, 0), (2, 2), (3, 3)** sẽ không được đặt quân nào khác.
- **Chéo phụ:** ô **(0, 2), (2, 0)** sẽ không được đặt quân nào khác.



Bước 1: Đặt quân Hậu dòng 0

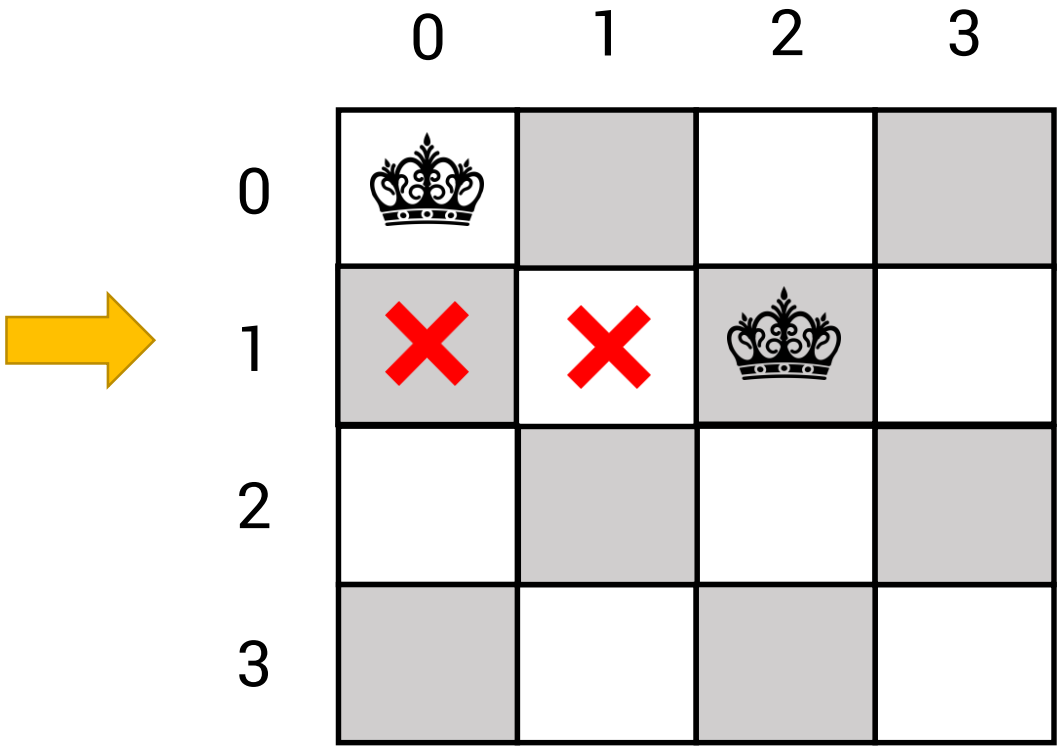
Đặt quân Hậu ô (0, 0). Dòng 0 sẽ không còn được đặt bất kì quân Hậu nào khác.



➔ Đệ quy xuống dòng 1.

Bước 2: Gọi đệ quy dòng 1

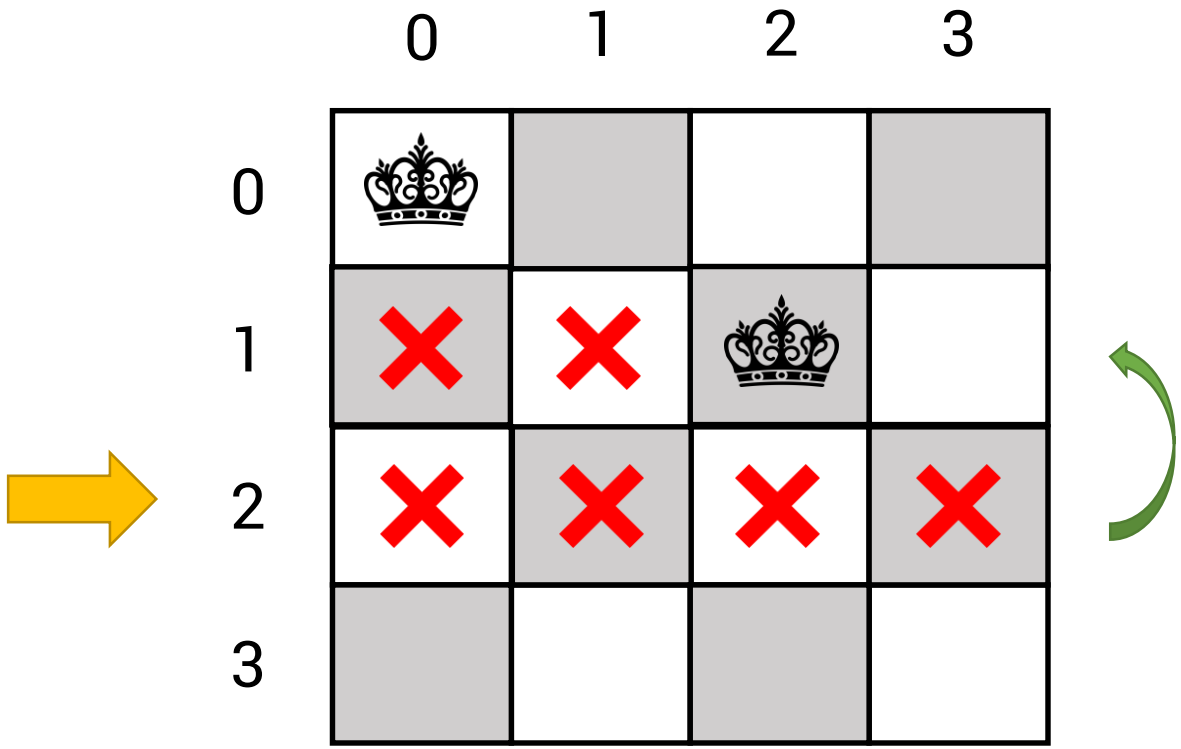
Ô (1, 0) và ô (1, 1) không thể đặt quân Hậu. Do đó đặt quân Hậu ô (1, 2).



➔ Đệ quy xuống dòng 2.

Bước 3: Gọi đệ quy dòng 2

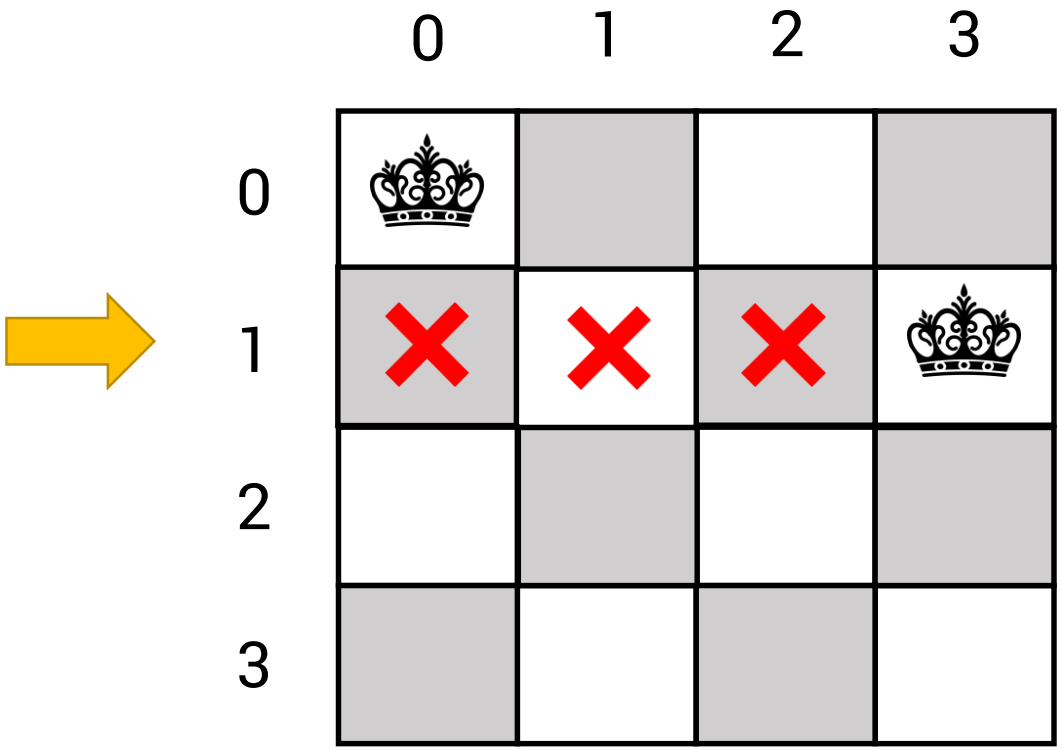
Không có vị trí nào có thể đặt quân Hậu ở dòng 2.



➔ Quay ngược lại trạng thái dòng 1 (**backtracking**)

Bước 4: Quay lui về dòng 1

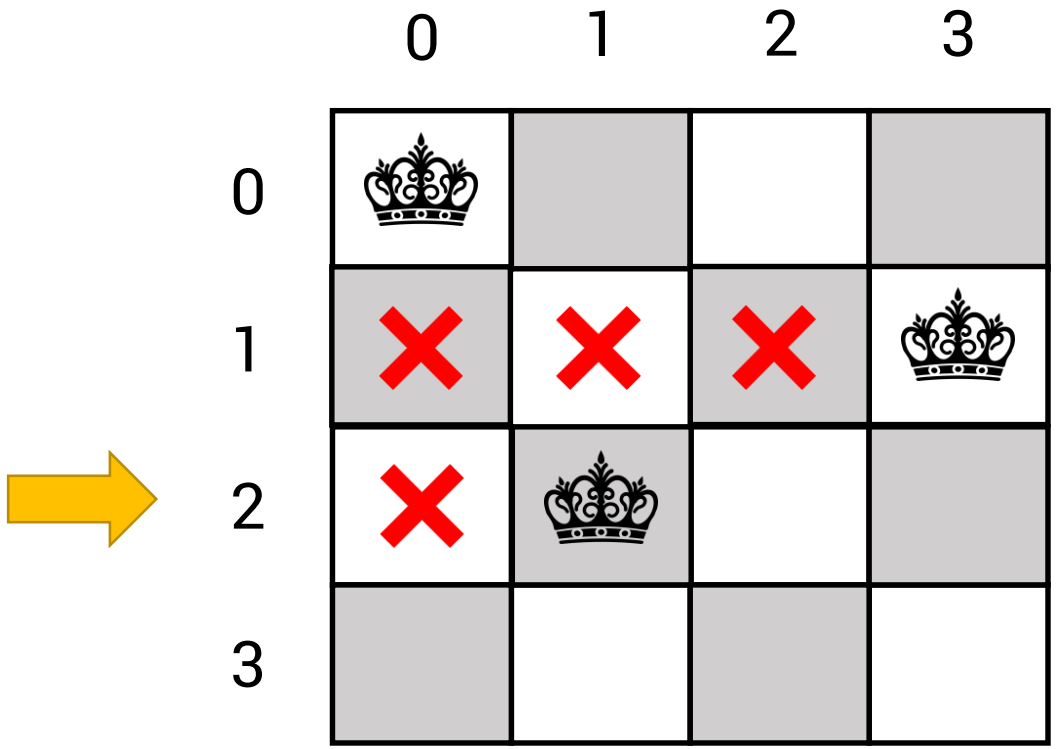
Ô (1, 2) không thể đặt quân Hậu. Do đó thử đặt quân Hậu ô (1, 3).



➔ Đệ quy xuống dòng 2.

Bước 5: Gọi đệ quy dòng 2

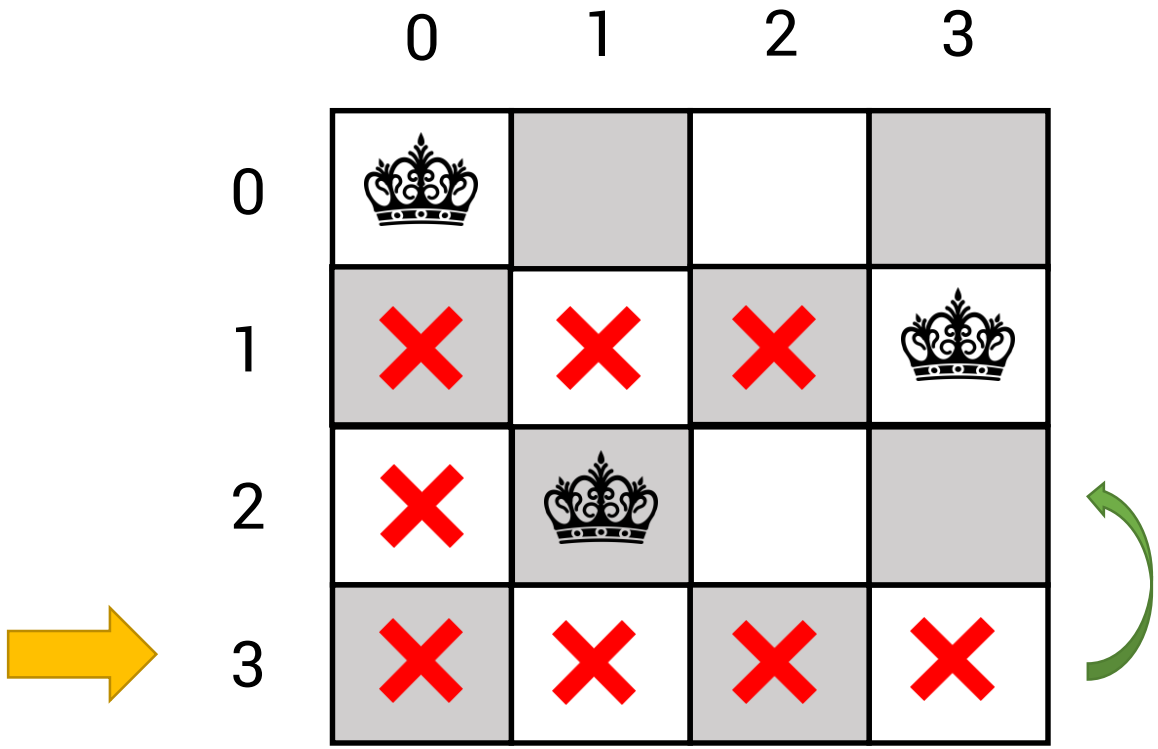
Ô (2, 0) không thể đặt quân Hậu. Đặt thử quân Hậu ô (2, 1).



➔ Gọi đệ quy dòng 3.

Bước 6: Gọi đệ quy dòng 3

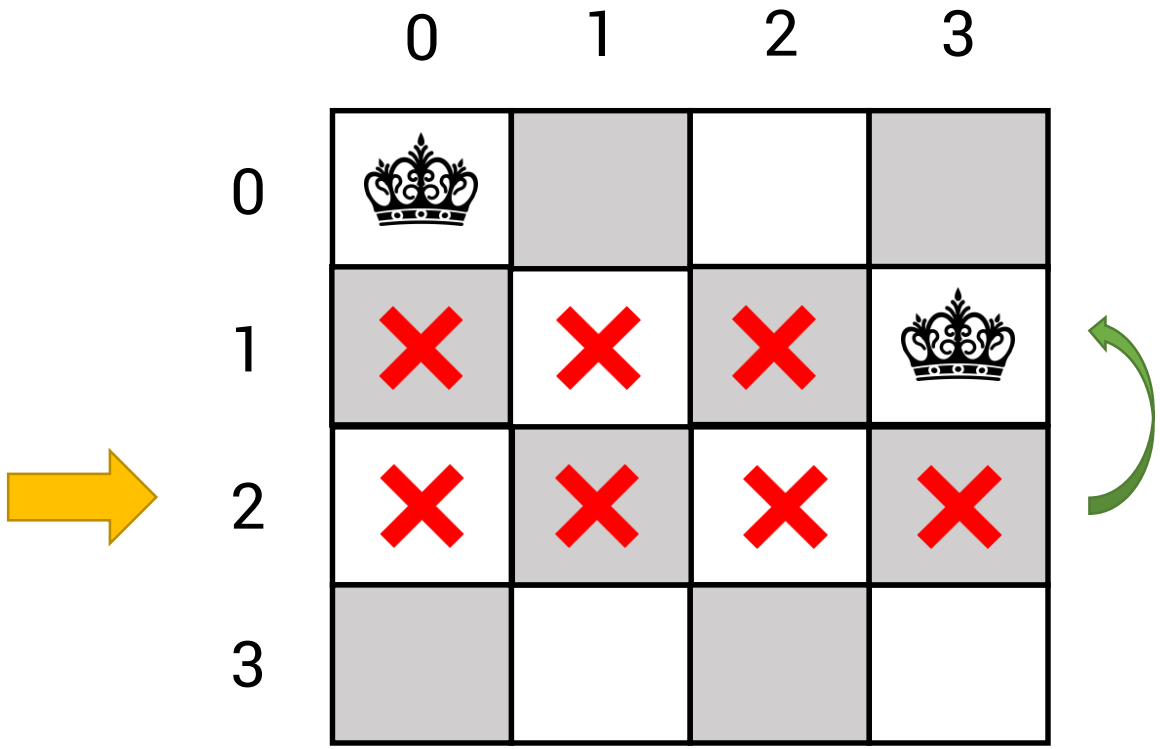
Không thể đặt quân Hậu ở dòng 3.



➔ Quay ngược lại trạng thái dòng 2 (**backtracking**)

Bước 7: Quay lui dòng 2

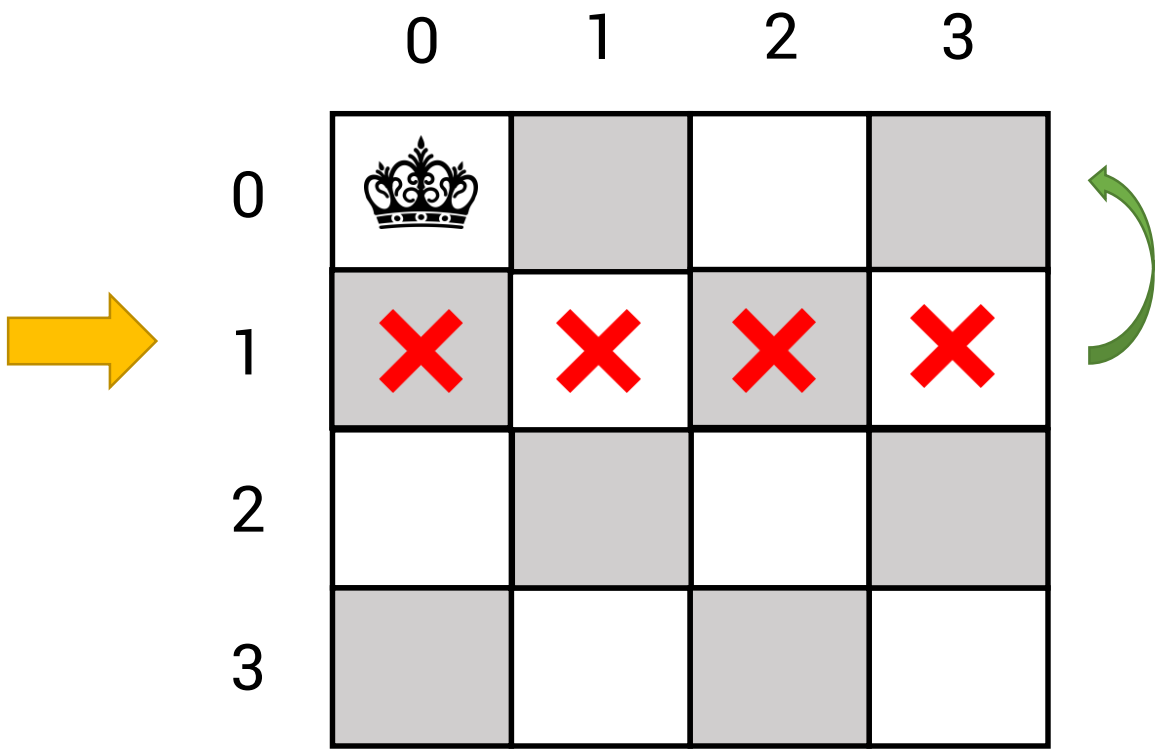
Không thể đặt quân Hậu ở dòng 2.



➔ Quay ngược lại trạng thái dòng 1 (**backtracking**)

Bước 8: Quay lui dòng 1

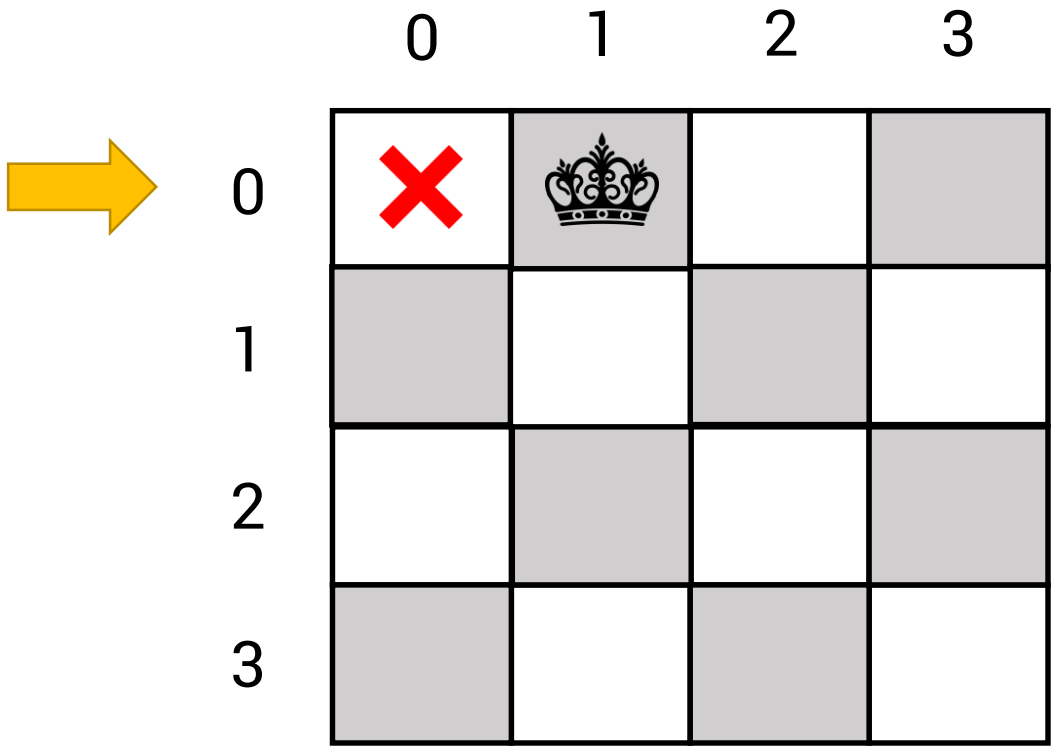
Không thể đặt quân Hậu ở dòng 1.



➔ Quay ngược lại trạng thái dòng 0 (**backtracking**)

Bước 9: Quay lui dòng 0

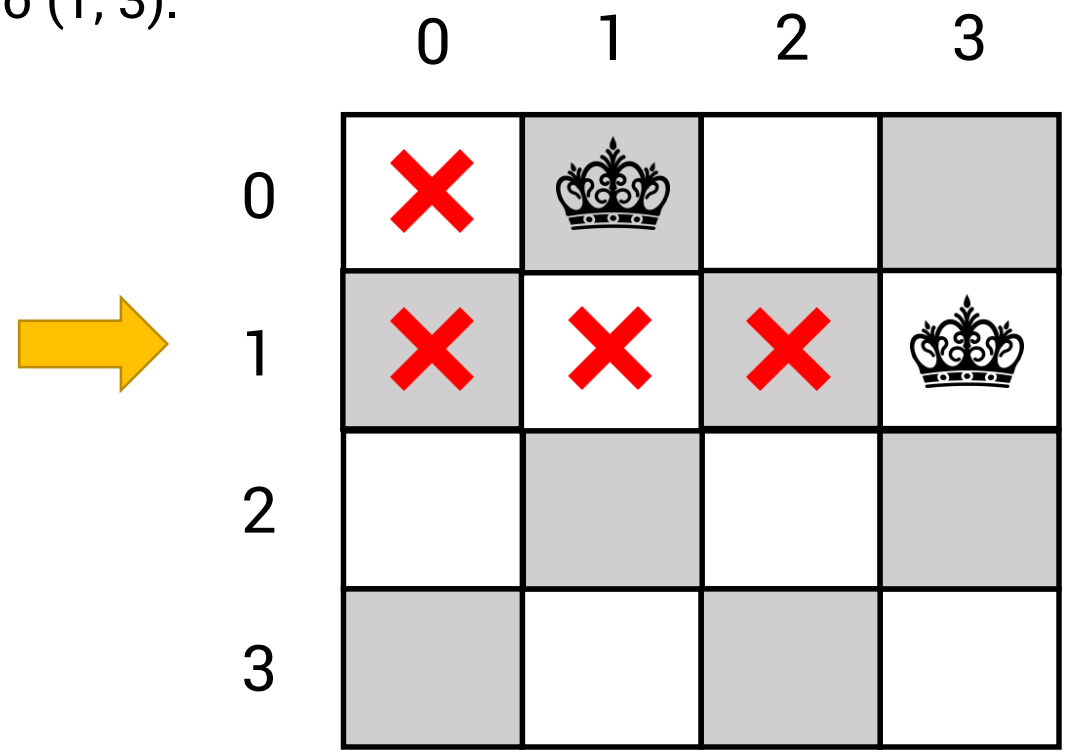
Thử đặt quân Hậu ô (1, 1).



➔ Đệ quy dòng 1.

Bước 10: Đệ quy dòng 1

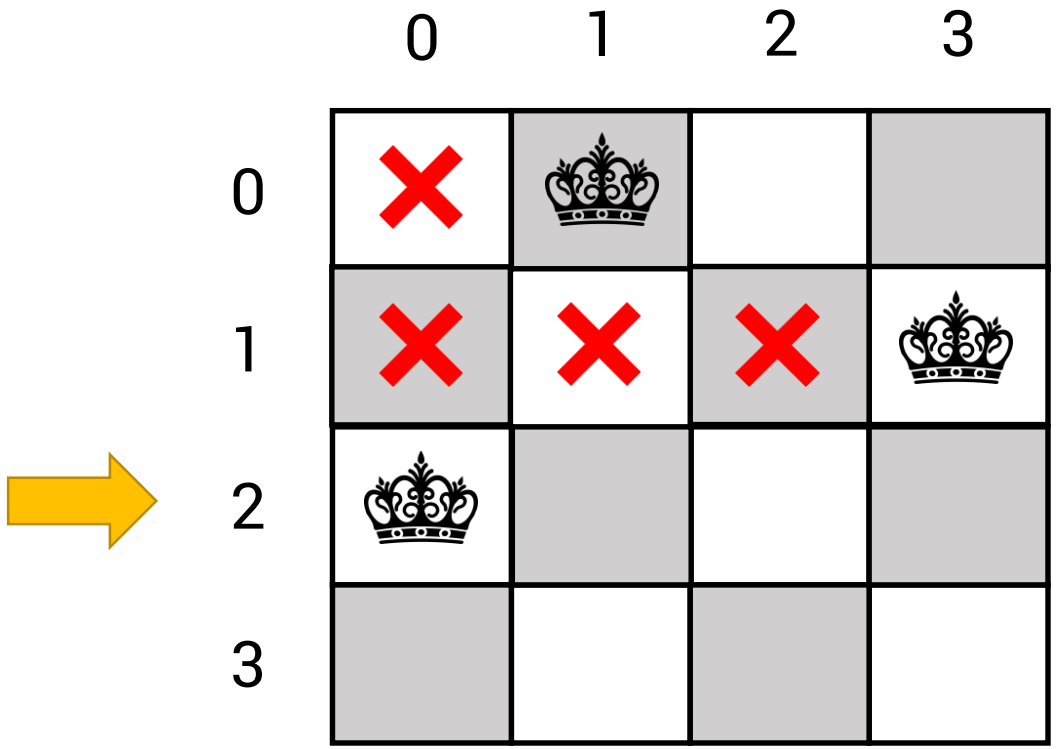
Không thể đặt quân Hậu ô (1, 0), ô (1, 1), và ô (1, 2). Thử đặt quân Hậu ở ô (1, 3).



➔ Đệ quy dòng 2.

Bước 11: Đệ quy dòng 2

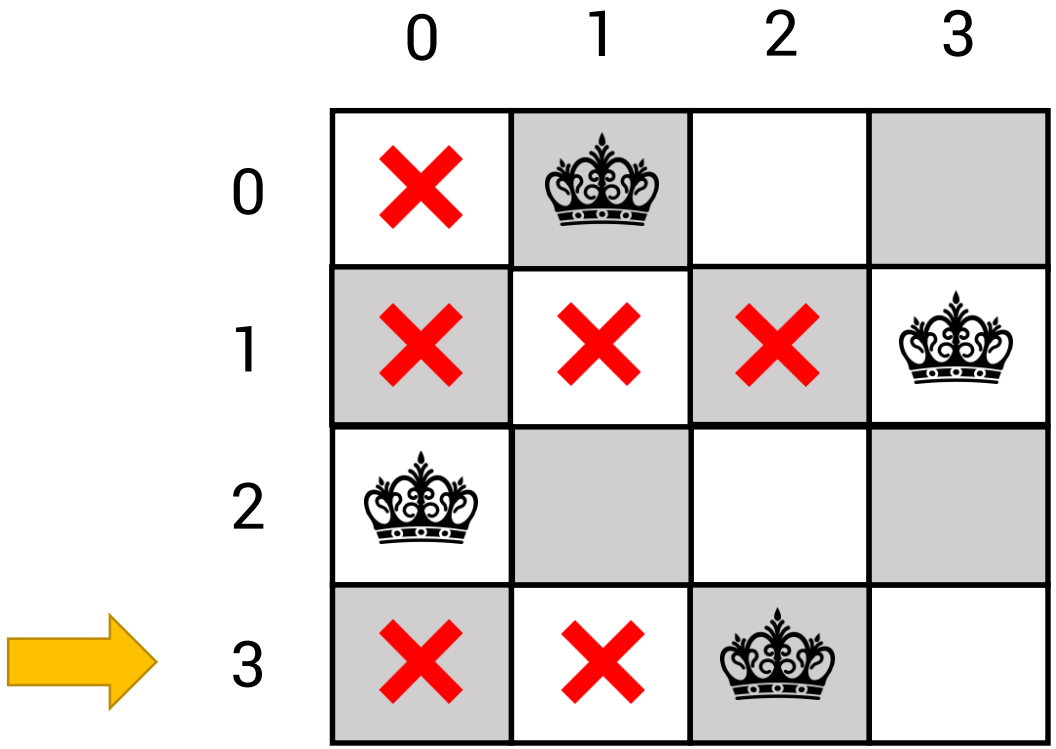
Thử đặt quân Hậu ô (2, 0).



➔ Đệ quy dòng 3.

Bước 12: Đệ quy dòng 3

Thử đặt quân Hậu ô (3, 2).

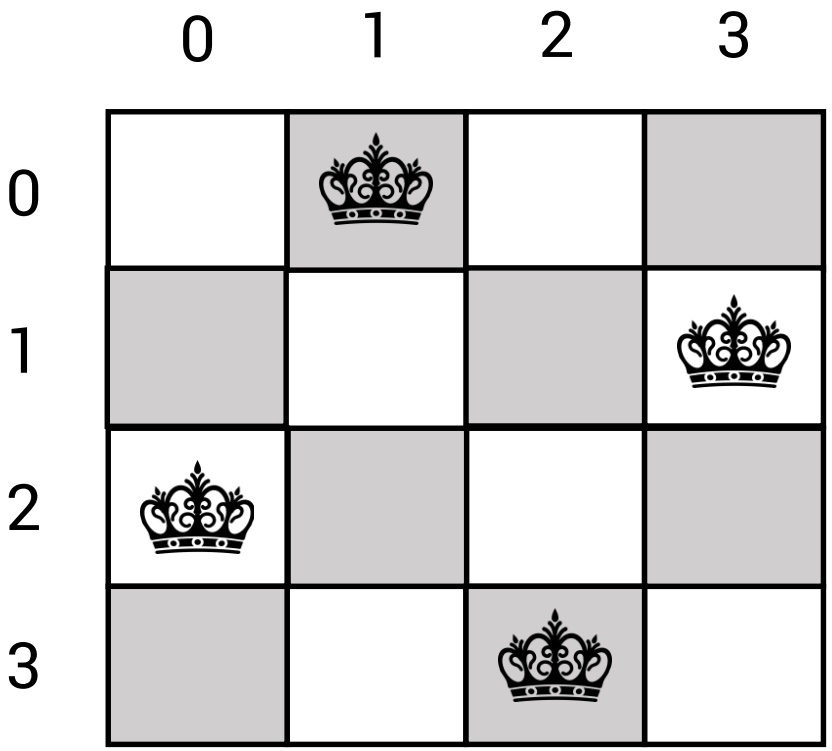


➔ Dừng thuật toán, đã tìm ra được 1 đáp án.

Xuất một kết quả của bài toán

Một đáp án đúng của bài toán là đặt quân Hậu tại các ô:

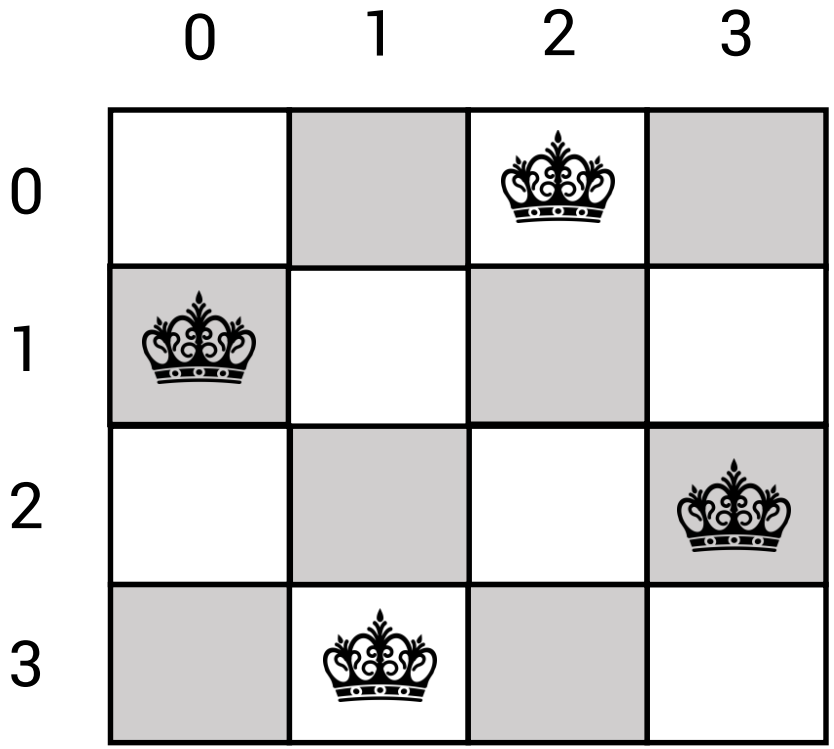
- (0, 1)
- (1, 3)
- (2, 0)
- (3, 2)



Một kết quả khác của bài toán

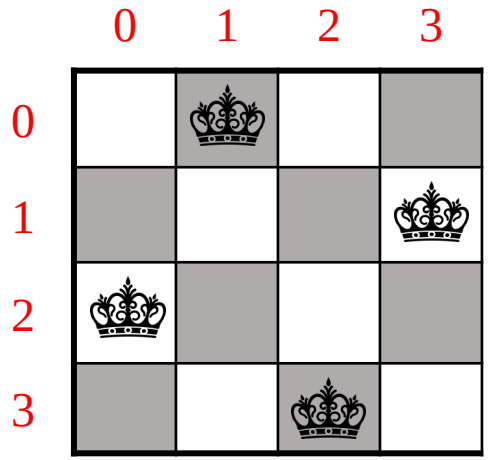
Ta có thể đặt quân Hậu tại các ô:

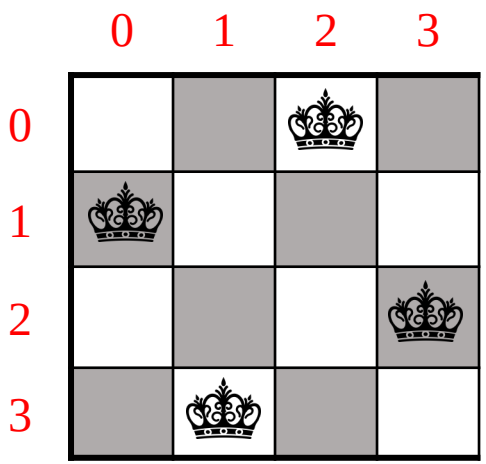
- (0, 2)
- (1, 0)
- (2, 3)
- (3, 1)



Kết quả cuối cùng

Như vậy sau khi chạy backtracking thì ta được 2 kết quả đúng với yêu cầu đề bài:





Time Complexity: O(N!)

Source Code N-Queen

```cpp
#include <iostream>
using namespace std;
#define N 4
int board[N][N];
void printSolution()
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            cout << " " << board[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}
```

Source Code N-Queen

```
15. bool check(int board[N][N], int row, int col)
16. {
17.     //check Vertical
18.     for (int i = 0; i < row; i++)
19.         if (board[i][col])
20.             return false;
21.     //check Main diagonal
22.     for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
23.         if (board[i][j])
24.             return false;
25.     //check Secondary diagonal
26.     for (int i = row, j = col; j < N && i >= 0; i--, j++)
27.         if (board[i][j])
28.             return false;
29.     return true;
30. }
```

Source Code N-Queen

```
31. bool NQueen(int board[N][N], int row)
32. {
33.     if (row == N)
34.     {
35.         printSolution();
36.         return true;
37.     }
38.     for (int j = 0; j < N; j++)
39.         //Check if the queen can be placed on chessboard
40.         if (check(board, row, j) == true) {
41.             //Place this queen in board[row][j]
42.             board[row][j] = 1;
43.             NQueen(board, row + 1);
44.             //Backtracking
45.             board[row][j] = 0;
46.         }
47.     return false;
48. }
```

Source Code N-Queen

```
48. int main()
49. {
50.     NQueen(board, 0);
51.     return 0;
52. }
```

Source Code N-Queen

```
1.  N = 4
2.  board = [[0] * N for i in range(N)]


3.  def printSolution():
4.      for i in range(N):
5.          for j in range(N):
6.              print(board[i][j], end = ' ')
7.          print()
8.      print()
```

Source Code N-Queen

```
9.  def check(board, row, col):
10.     # check Vertical
11.     for i in range(row):
12.         if board[i][col]:
13.             return False
14.     # check Main diagonal
15.     i = row
16.     j  = col
17.     while i >= 0 and j >= 0:
18.         if board[i][j]:
19.             return False
20.         i -= 1
21.         j -= 1
```

Source Code N-Queen

```
22.         # check Secondary diagonal
23.         i = row
24.         j = col
25.         while j < N and i >= 0:
26.             if board[i][j]:
27.                 return False
28.             i -= 1
29.             j += 1
30.         return True
```

Source Code N-Queen

```
31. def NQueen(board, row):
32.     if row == N:
33.         printSolution()
34.             return True
35.     for j in range(N):
36.         if check(board, row, j) == True:
37.             board[row][j] = 1
38.             NQueen(board, row + 1)
39.             board[row][j] = 0
40.     return False
41. if __name__ == "__main__":
42.     NQueen(board, 0)
```

Source Code N-Queen

```
1.  public class Main {
2.      private static int N = 4;
3.      private static int[][] board = new int[N][N];

4.      private static void printSolution() {
5.          for (int i = 0; i < N; i++) {
6.              for (int j = 0; j < N; j++)
7.                  System.out.printf(" %d ", board[i][j]);
8.              System.out.println();
9.          }
10.         System.out.println();
11.     }
```

Source Code N-Queen

```
12.     private static boolean check(int[][] board, int row, int col) {
13.         //check Vertical
14.         for (int i = 0; i < row; i++) {
15.             if (board[i][col] != 0)
16.                 return false;
17.         }
18.         //check Main diagonal
19.         for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
20.             if (board[i][j] != 0)
21.                 return false;
22.         }
23.         //check Secondary diagonal
24.         for (int i = row, j = col; j < N && i >= 0; i--, j++) {
25.             if (board[i][j] != 0)
26.                 return false;
27.         }
28.         return true;
29.     }
```

Source Code N-Queen

```
30.     private static boolean NQueen(int[][] board, int row) {
31.         if (row == N) {
32.             printSolution();
33.             return true;
34.         }
35.         for (int j = 0; j < N; j++) {
36.             //Check if the queen can be placed on chessboard
37.             if (check(board, row, j) == true) {
38.                 //Place this queen in board[row][j]
39.                 board[row][j] = 1;
40.                 NQueen(board, row + 1);
41.                 //Backtracking
42.                 board[row][j] = 0;
43.             }
44.         }
45.         return false;
46.     }
```

Source Code N-Queen

```
47.     public static void main(String[] args) {
48.         NQueen(board, 0);
49.         return;
50.     }
51. }
```

Permutations Of String

Permutations Of String (hoán vị chuỗi): Cho một chuỗi các ký tự, hãy tìm tất cả các hoán vị của chuỗi đã cho.

Ví dụ: Cho chuỗi “ABCD” tìm tất cả các hoán vị của chuỗi đã cho.

A	B	C	D

Một số hoán vị của chuỗi ABCD.

A	B	D	C

D	B	C	A

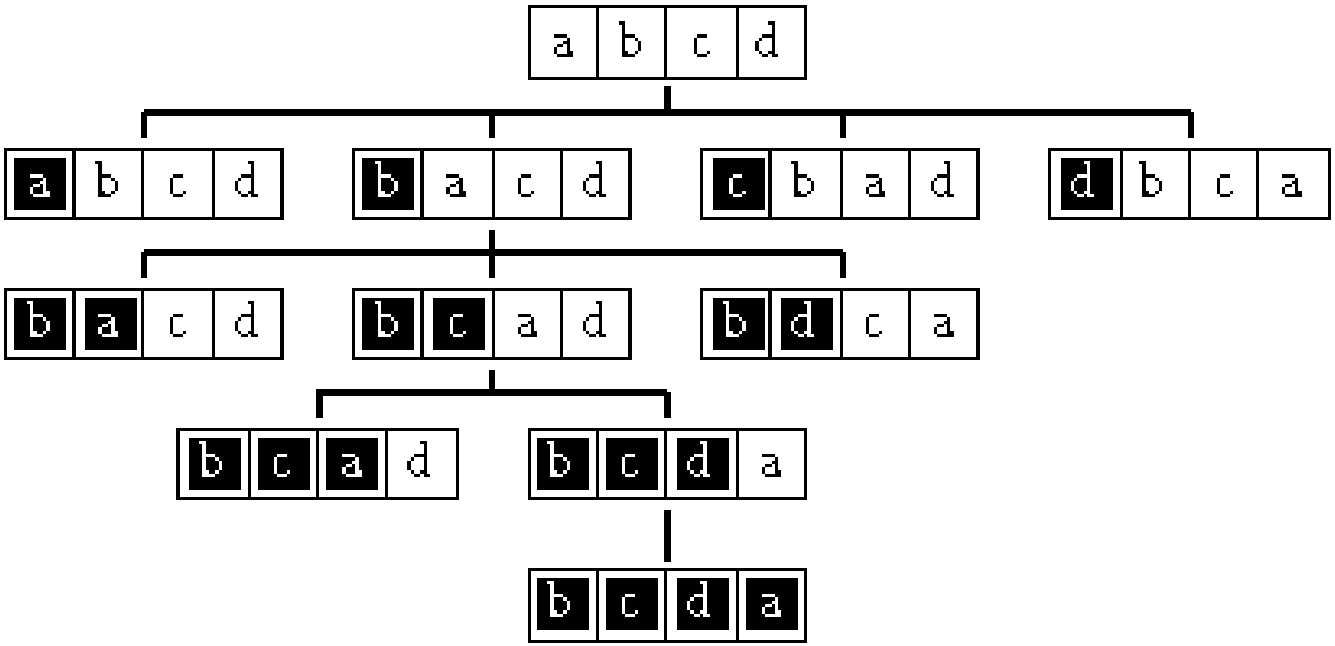
B	C	A	D

C	D	A	B

Ý tưởng cách giải quyết

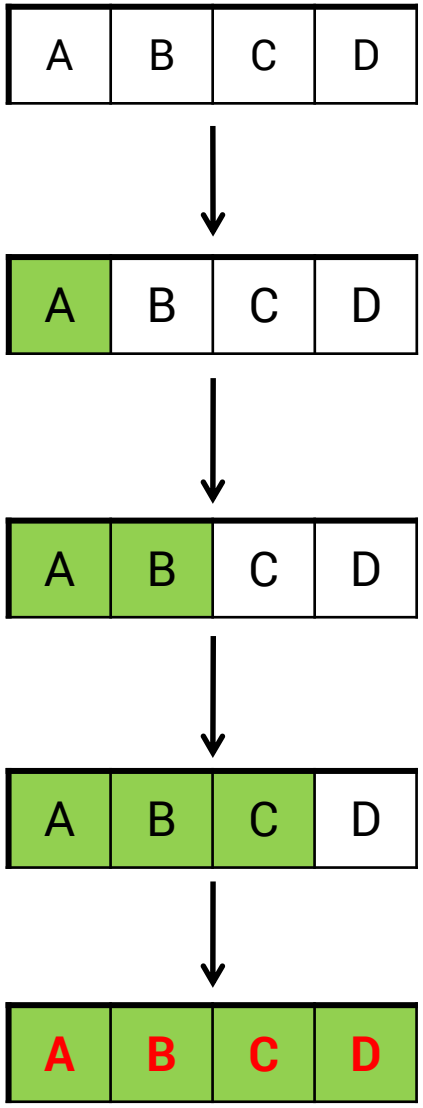
Lần lượt cố định các ký tự ở từng vị trí trong chuỗi ban đầu, để tạo ra các chuỗi con. Cho đến khi cố định được ký tự nào nữa thì chuỗi đó là chuỗi kết quả.

Quay lại bước trước để tìm cố định ký tự ở vị trí khác và tạo ra chuỗi con mới. Lặp đi lặp lại quá trình này cho đến khi tìm được tất cả các chuỗi.



Time Complexity: O(N*N!)

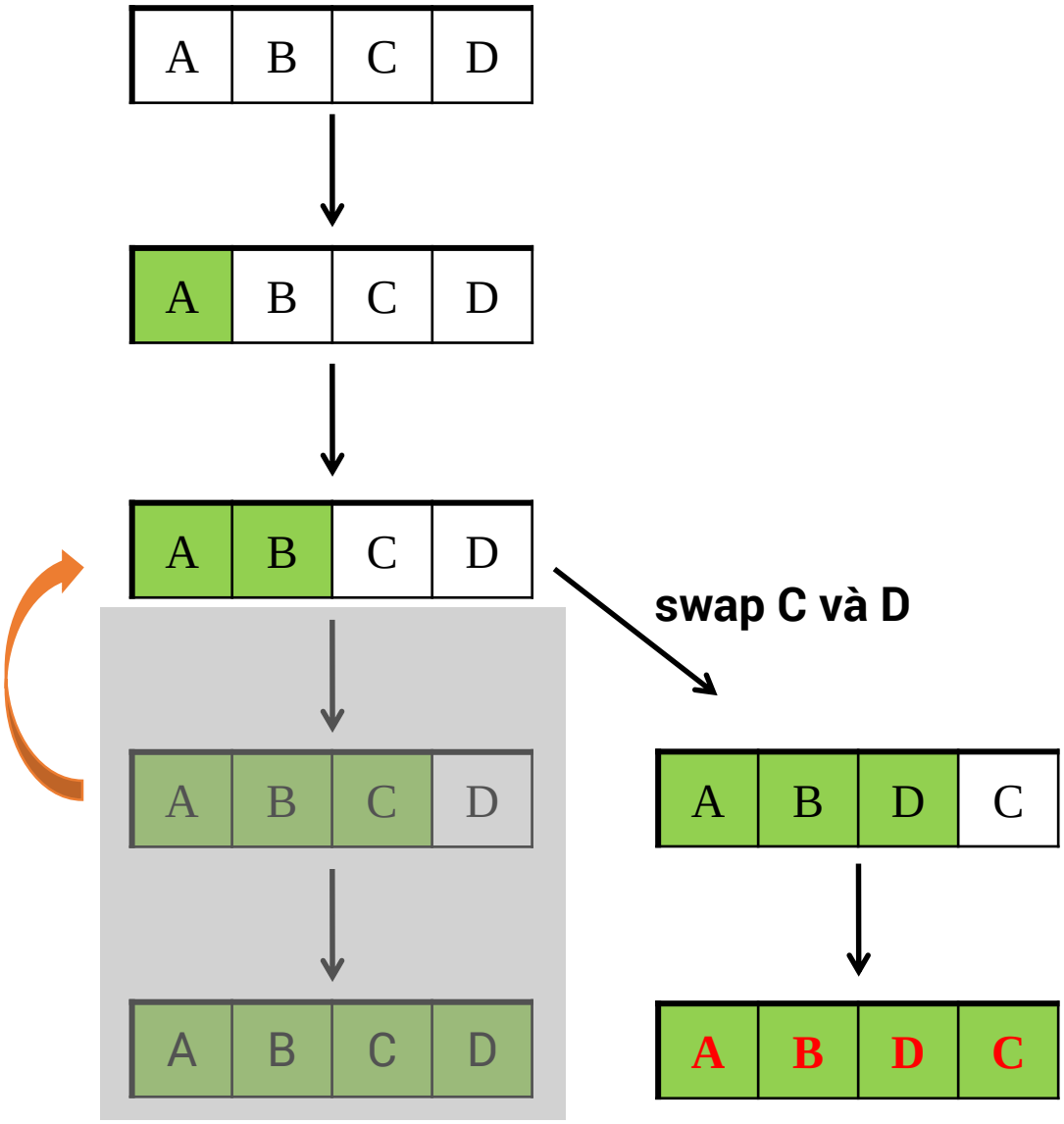
Bước 1: chạy thuật toán lần 1



Kết quả:

- **ABCD**

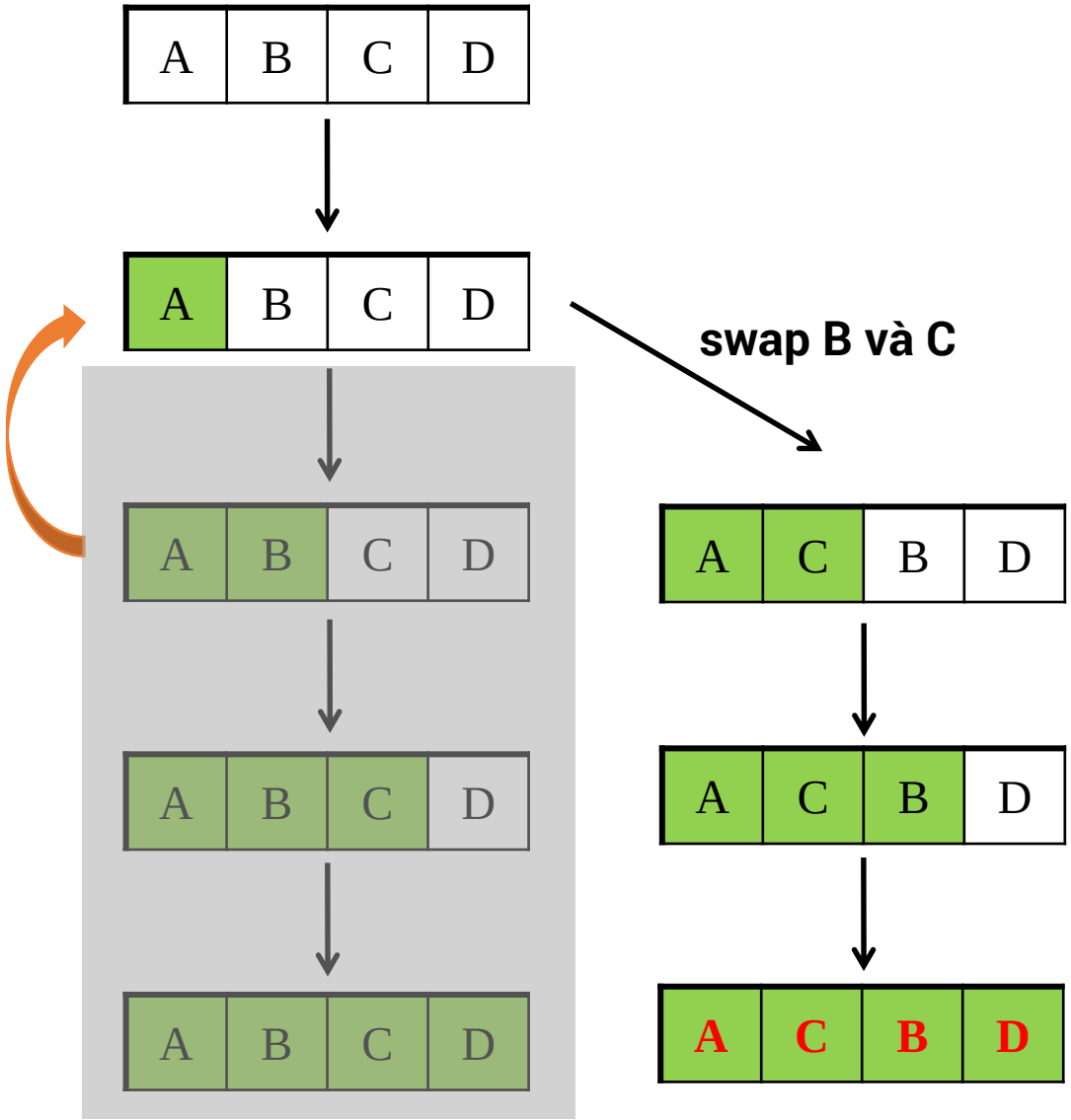
Bước 2: chạy thuật toán lần 2



Kết quả:

- ABCD, **ABDC**

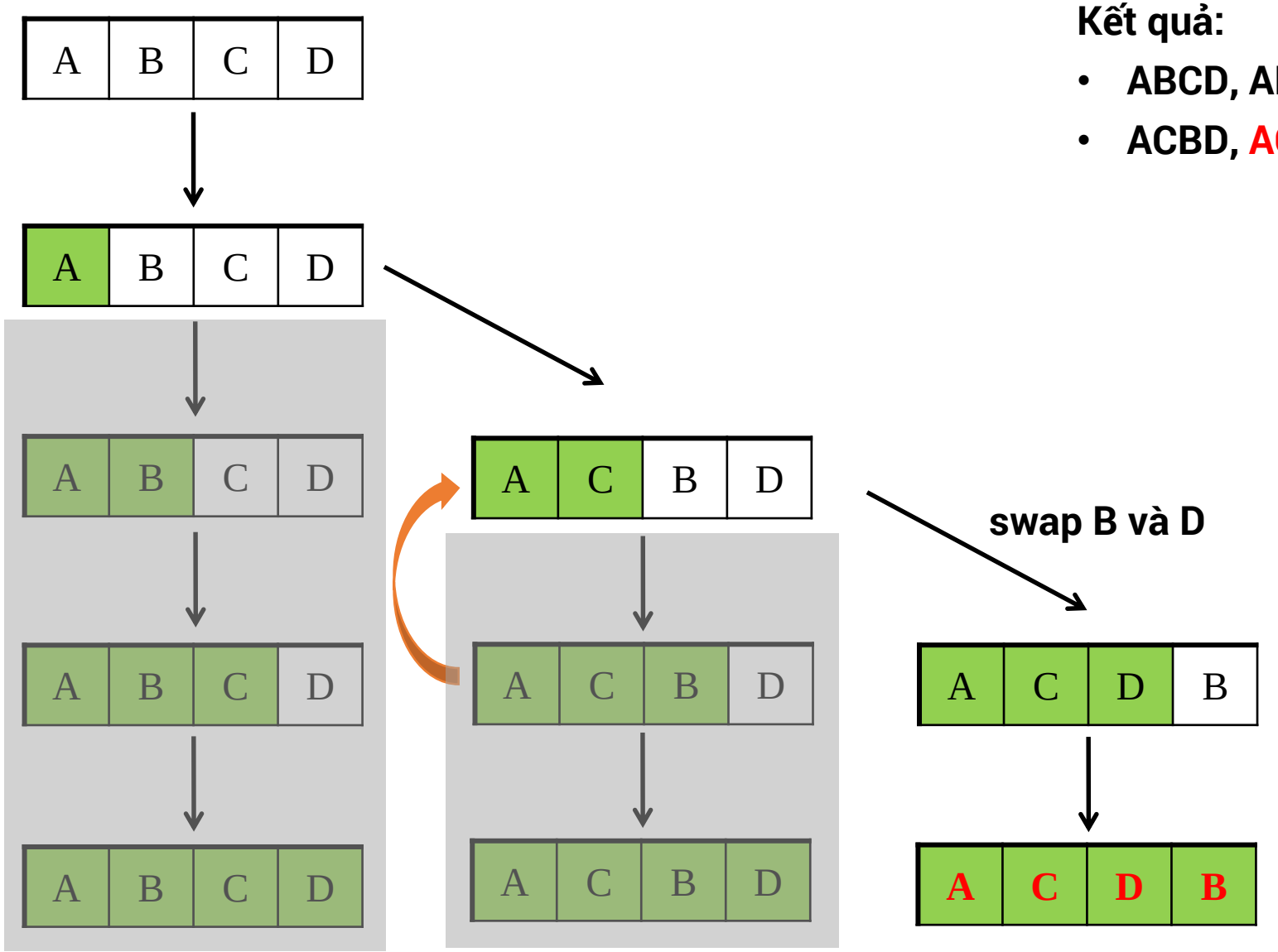
Bước 3: chạy thuật toán lần 3



Kết quả:

- ABCD, ABDC
- **ACBD**

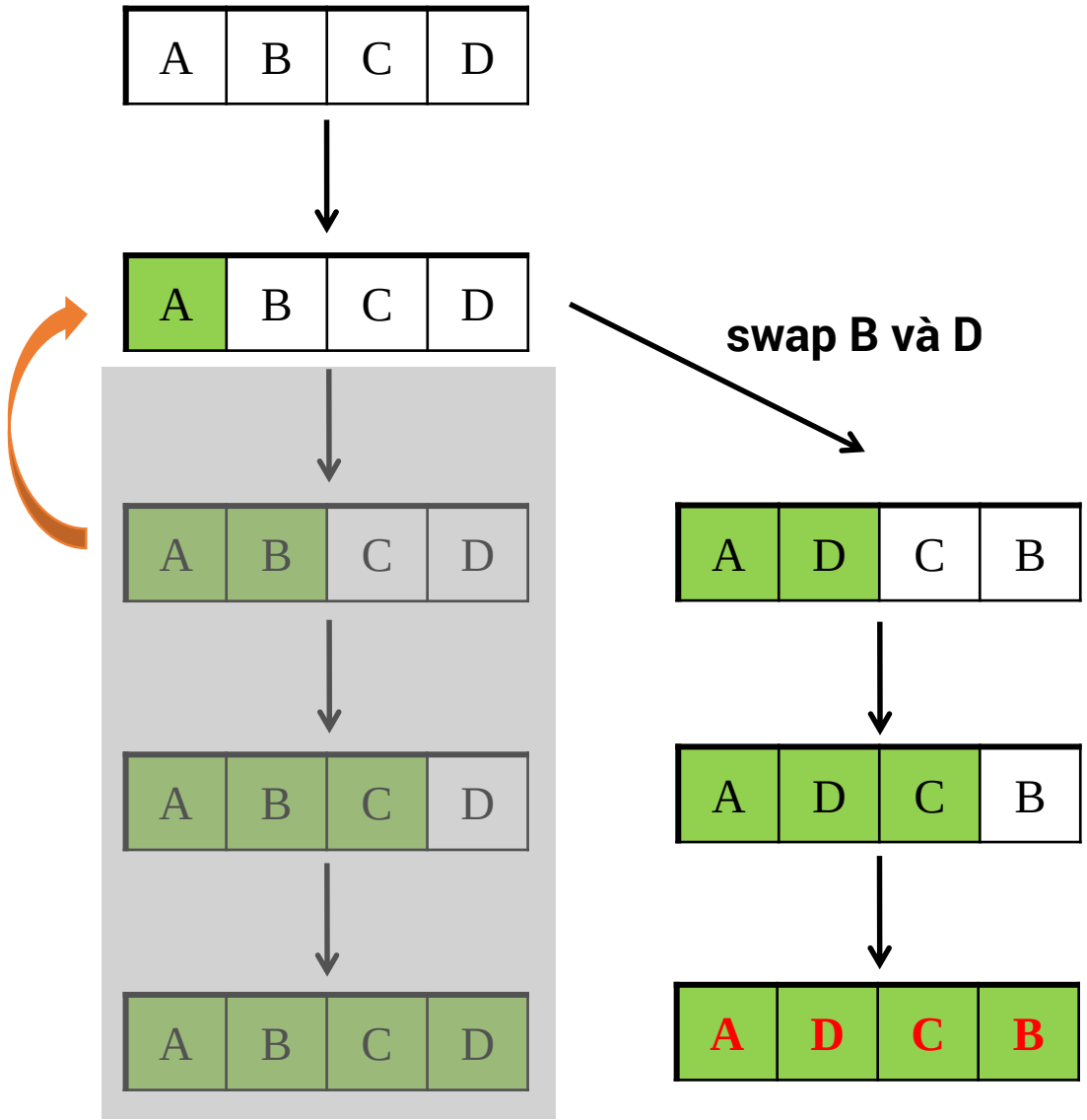
Bước 4: chạy thuật toán lần 4



Kết quả:

- **ABCD, ABDC**
- **ACBD, ACDB**

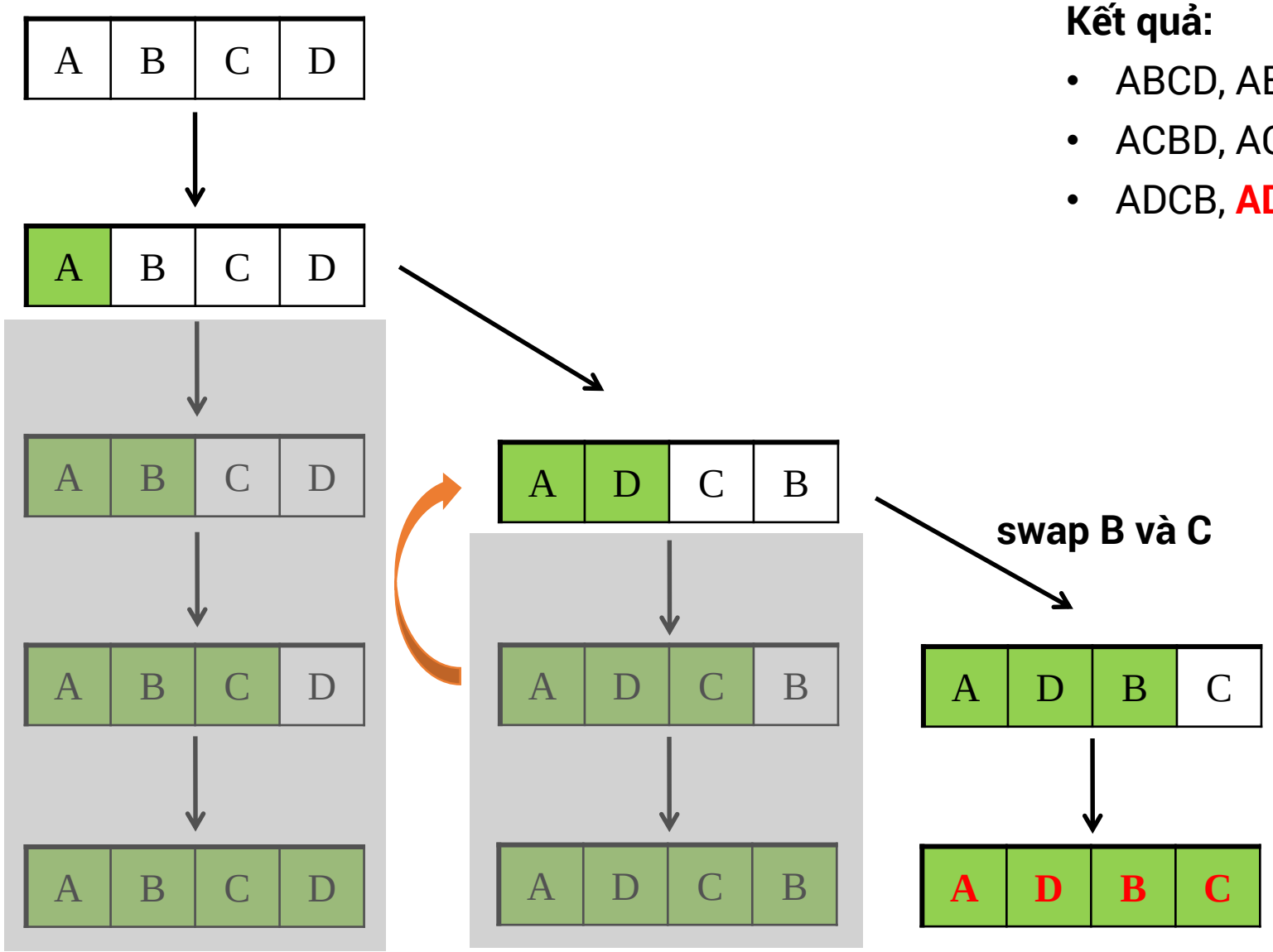
Bước 5: chạy thuật toán lần 5



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- **ADCB**

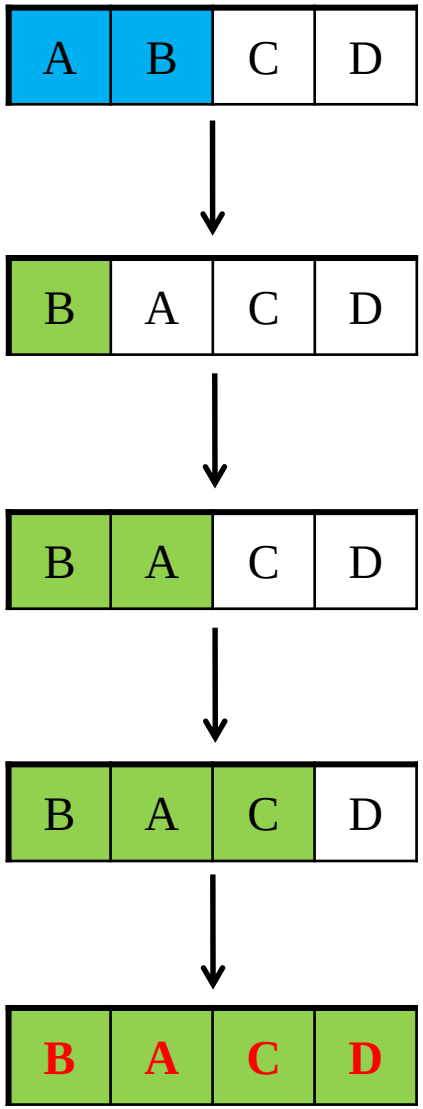
Bước 6: chạy thuật toán lần 6



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- ADCB, **ADBC**

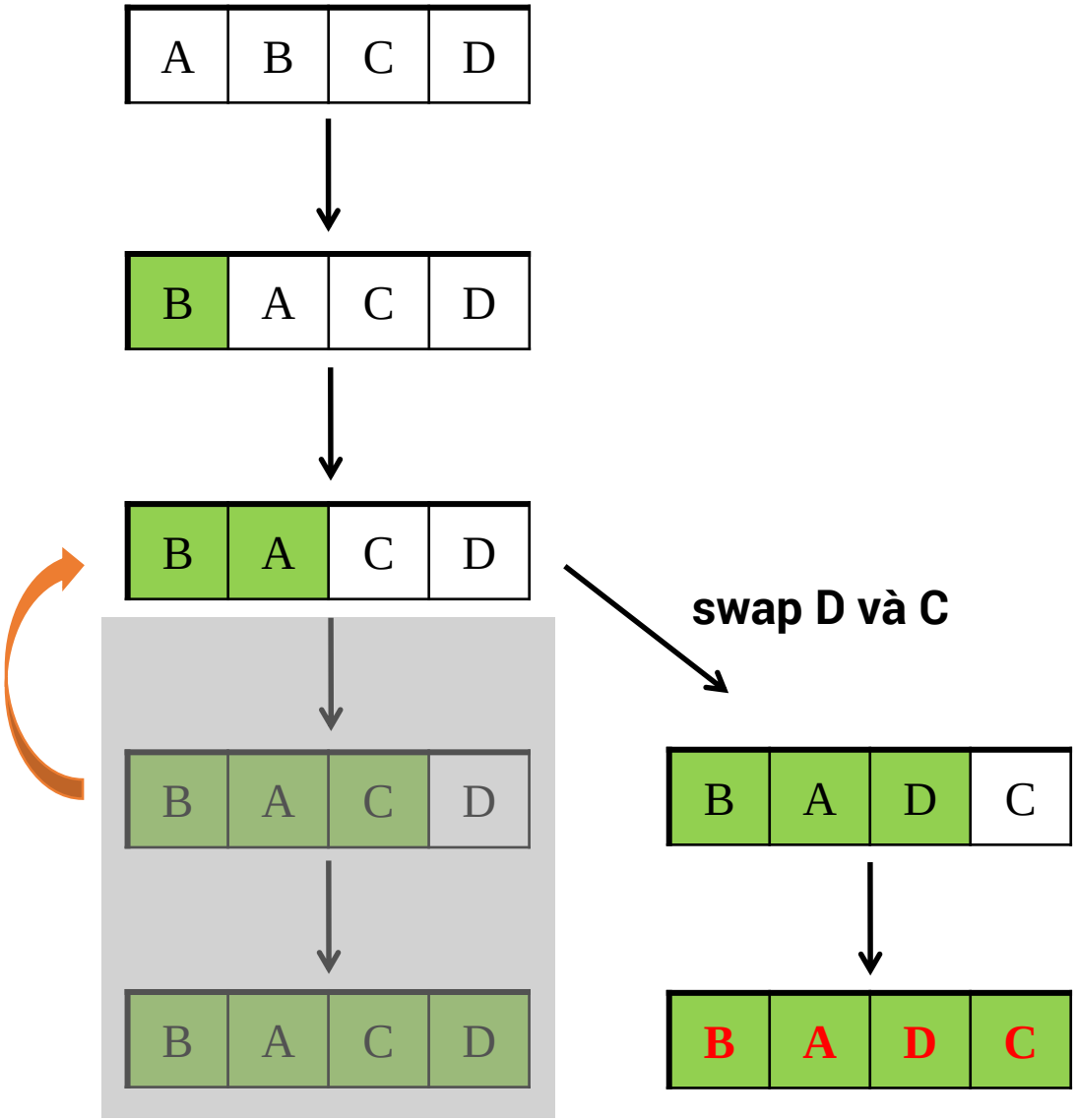
Bước 7: chạy thuật toán lần 7



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- ADCB, ADBC
- **BACD**

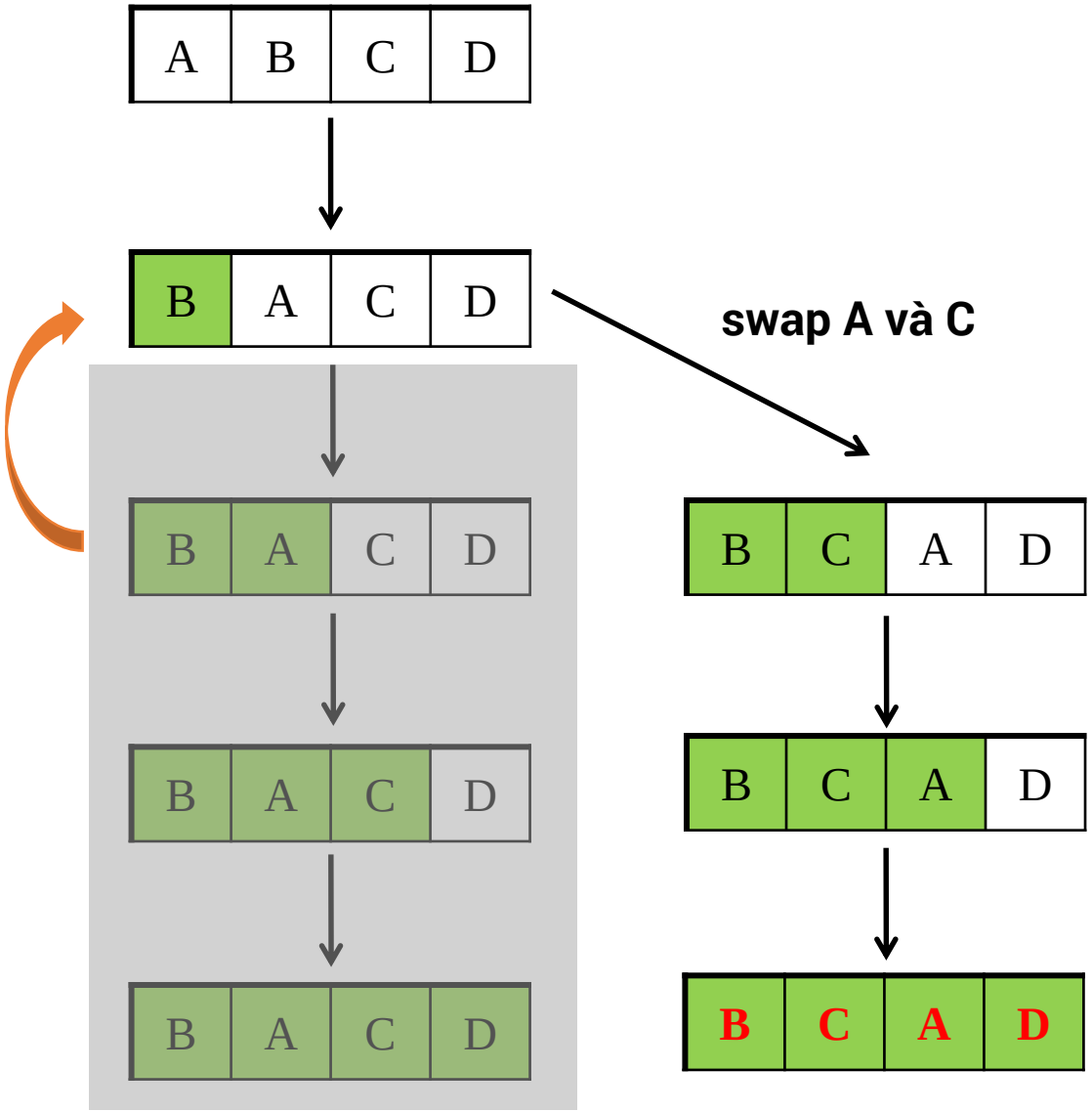
Bước 8: chạy thuật toán lần 8



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- ADCB, ADBC
- BACD, **BADC**

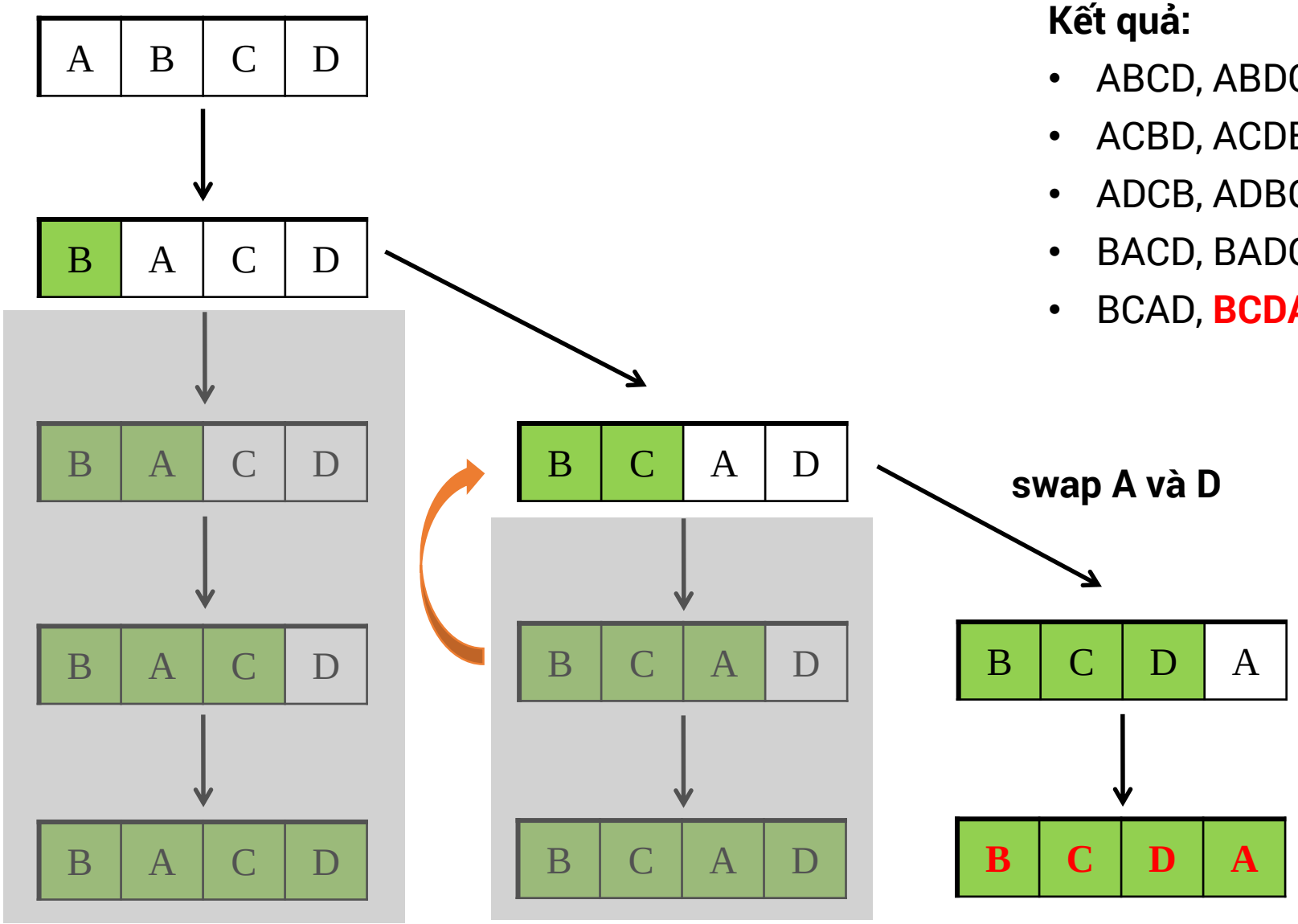
Bước 9: chạy thuật toán lần 9



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- ADCB, ADBC
- BACD, BADC
- **BCAD**

Bước 10: chạy thuật toán lần 10



Kết quả:

- ABCD, ABDC
- ACBD, ACDB
- ADCB, ADBC
- BACD, BADC
- BCAD, **BCDA**

TIẾP TỤC CHẠY BACKTRACKING CHO CÁC TRƯỜNG HỢP CÒN LẠI

Kết quả bài toán

Cho chuỗi “ABCD” tìm tất cả các hoán vị của chuỗi đã cho.

A	B	C	D

Chuỗi ABCD có 4 ký tự (n = 4) như vậy sẽ có tổng cộng 24 hoán vị có thể có của chuỗi ABCD.

ABCD	BACD	CBAD	DBCA
ABDC	BADC	CBDA	DBAC
ACBD	BCAD	CABD	DCBA
ACDB	BCDA	CADB	DCAB
ADCB	BDCA	CDAB	DACB
ADBC	BDAC	CDBA	DABC

Source Code Permutations Of String

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
void permutation(string s, int l, int r)
{
    if (l == r)
        cout << s << endl;
    else
    {
        for (int i = l; i < r; i++)
        {
            swap(s[l], s[i]);
            permutation(s, l + 1, r);
            swap(s[l], s[i]);
        }
    }
}
```

Source Code Permutations Of String

```
19. int main()
20. {
21.     string s("ABCD");
22.     permutation(s, 0, s.length());
23.     return 0;
24. }
```

Source Code Permutations Of String

```
1.  def permutation(s, l, r):
2.      if l == r:
3.          print(''.join(s))
4.      else:
5.          for i in range(l, r):
6.              s[l], s[i] = s[i], s[l]
7.              permutation(s, l + 1, r)
8.              s[l], s[i] = s[i], s[l]
9.
10. if __name__ == "__main__":
11.     s = list("ABCD")
12.     permutation(s, 0, len(s))
```

Source Code Permutations Of String

```java
import java.lang.StringBuilder;
public class Main {
    private static void permutation(StringBuilder s, int l, int r) {
        if (l == r)
            System.out.println(s);
        else {
            for (int i = l; i < r; i++) {
                char temp = s.charAt(l);
                s.setCharAt(l, s.charAt(i));
                s.setCharAt(i, temp);
                permutation(s, l + 1, r);
                temp = s.charAt(l);
                s.setCharAt(l, s.charAt(i));
                s.setCharAt(i, temp);
            }
        }
    }
```

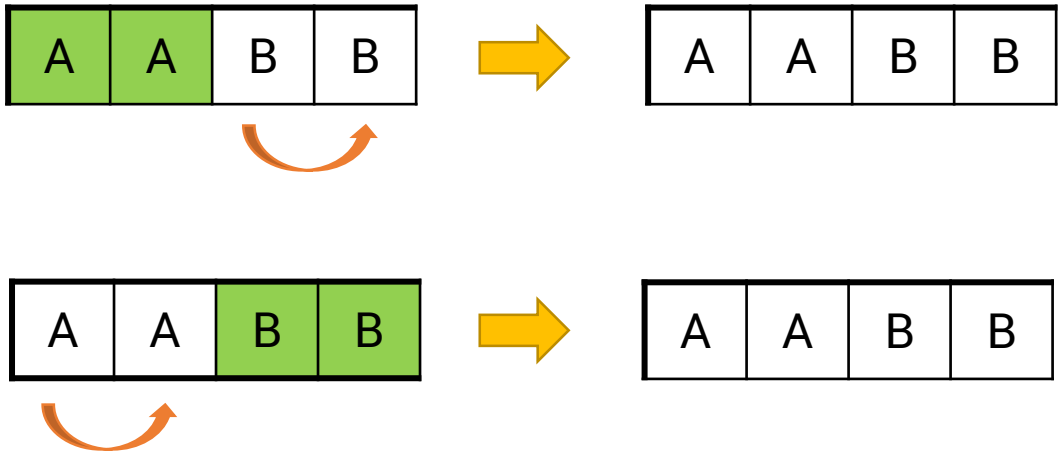
Source Code Permutations Of String

```
18.     public static void main(String[] args) {
19.         StringBuilder s = new StringBuilder("ABCD");
20.         permutation(s, 0, s.length());
21.         return;
22.     }
23. }
```

Distinct Permutations Of String

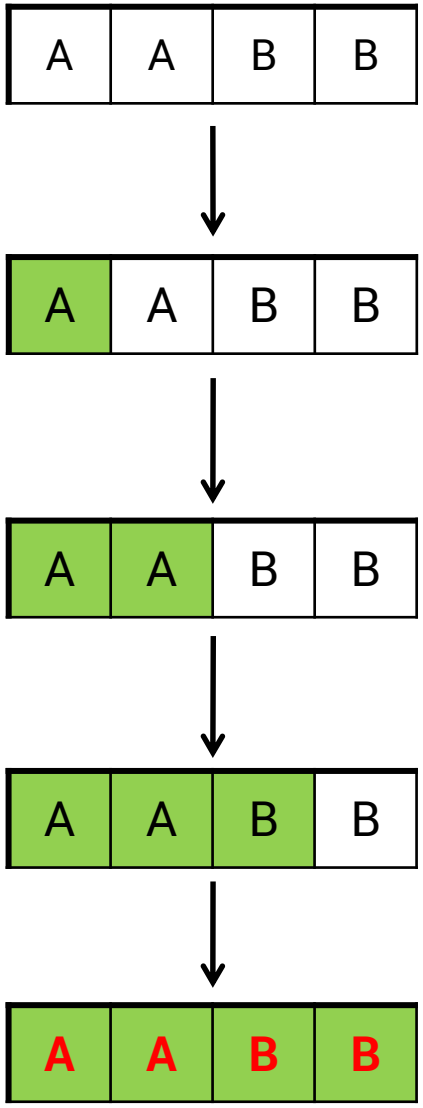
Distinct Permutations Of String (Hoán vị chuỗi không trùng lặp): Nếu trong trường hợp chuỗi có nhiều ký tự giống nhau khả năng sẽ gặp sự trùng lặp.

Ví dụ: Cho chuỗi “AABB” tìm tất cả các hoán vị của chuỗi đã cho.



Time Complexity: O(N*N*N!)

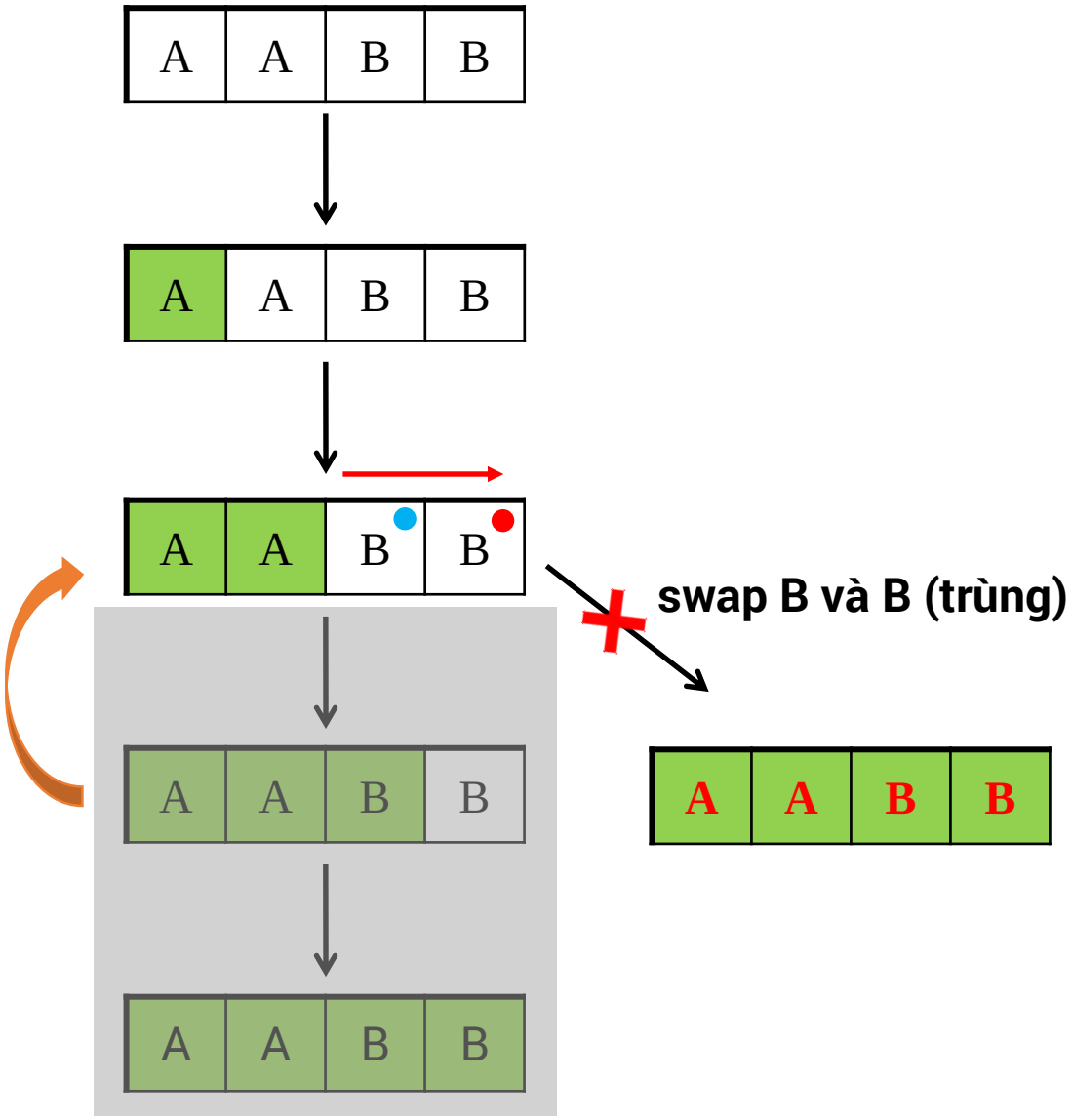
Bước 1: chạy thuật toán lần 1



Kết quả:

- **AABB**

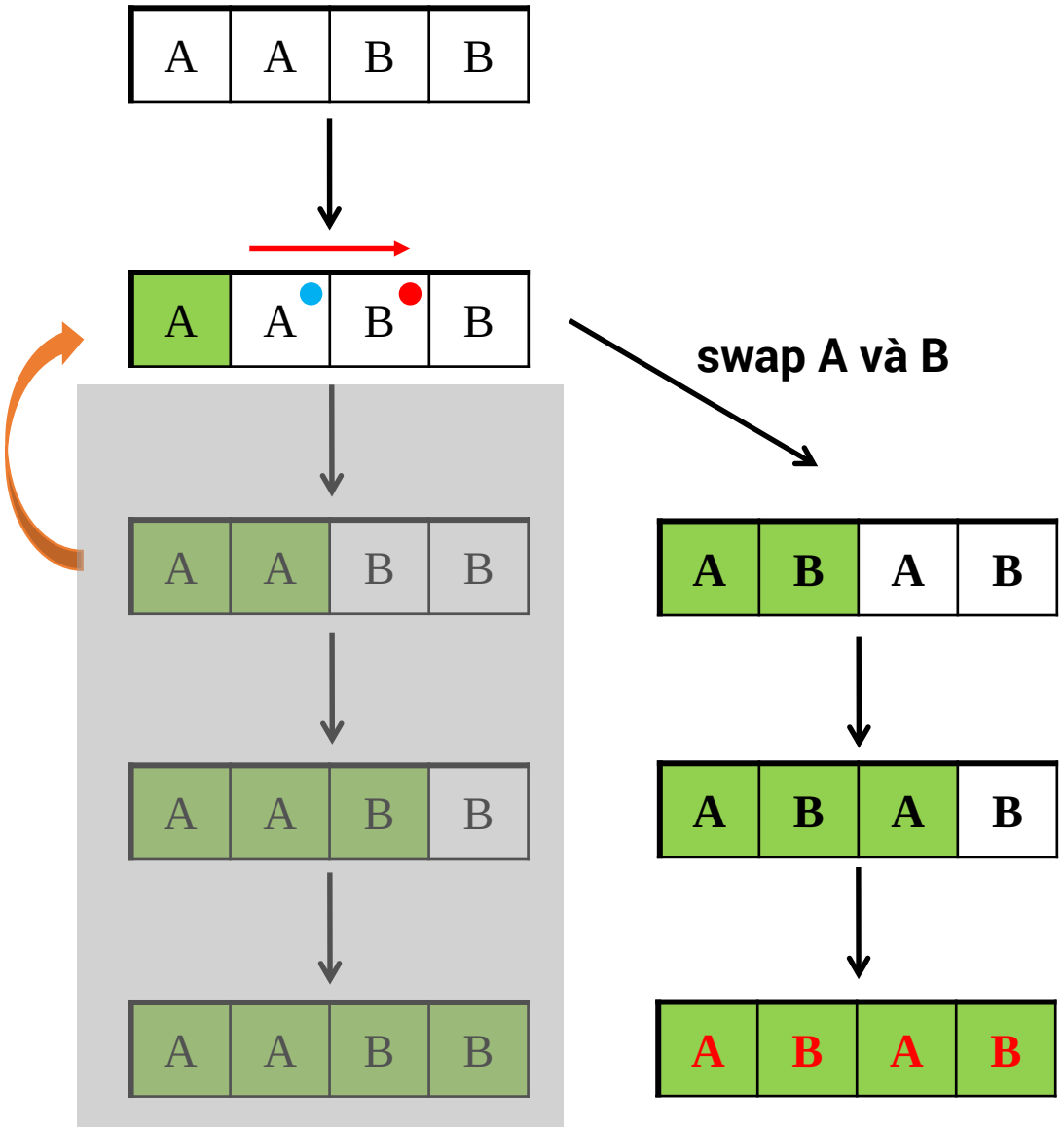
Bước 2: chạy thuật toán lần 2



Kết quả:

- AABB

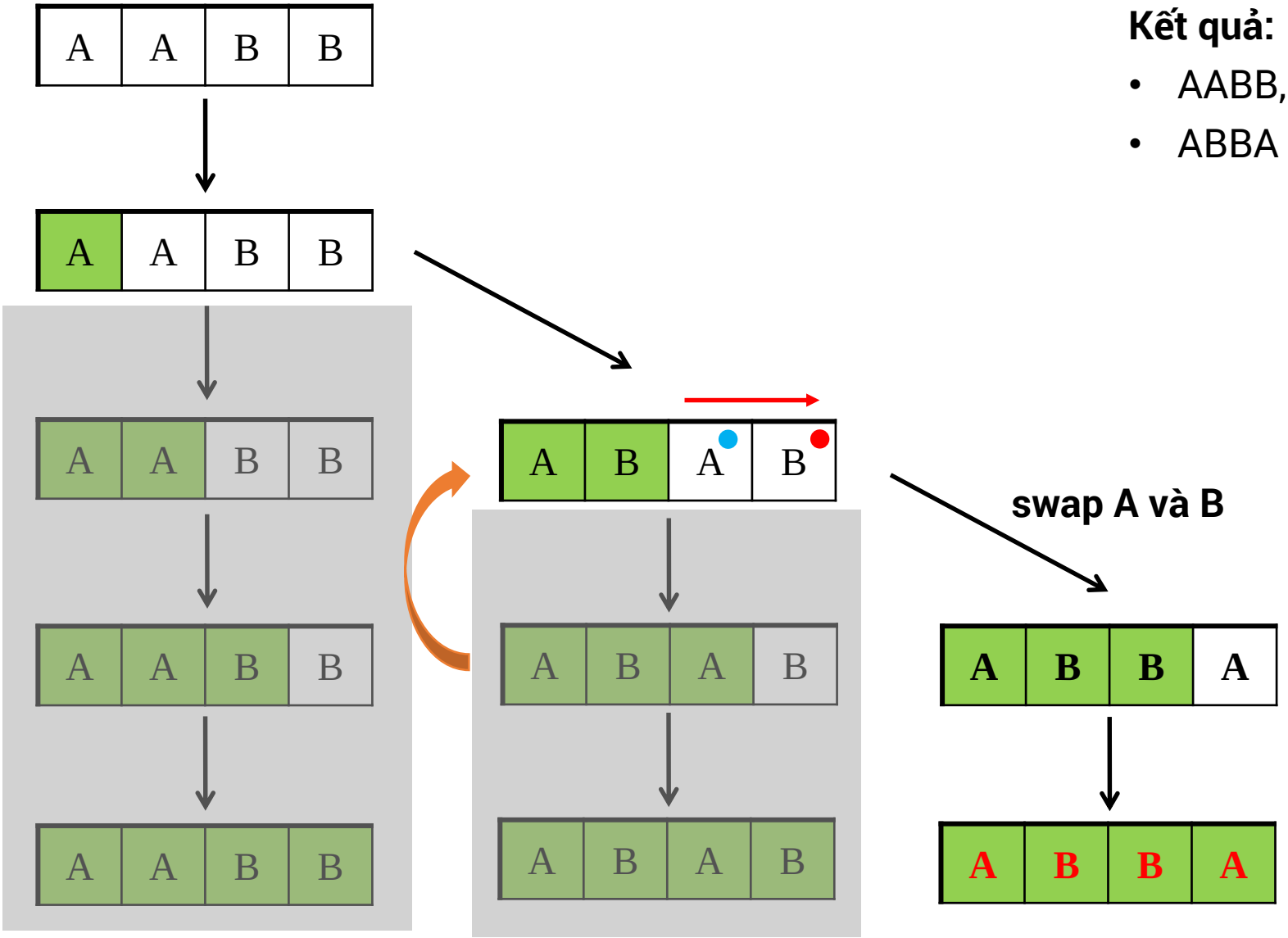
Bước 3: chạy thuật toán lần 3



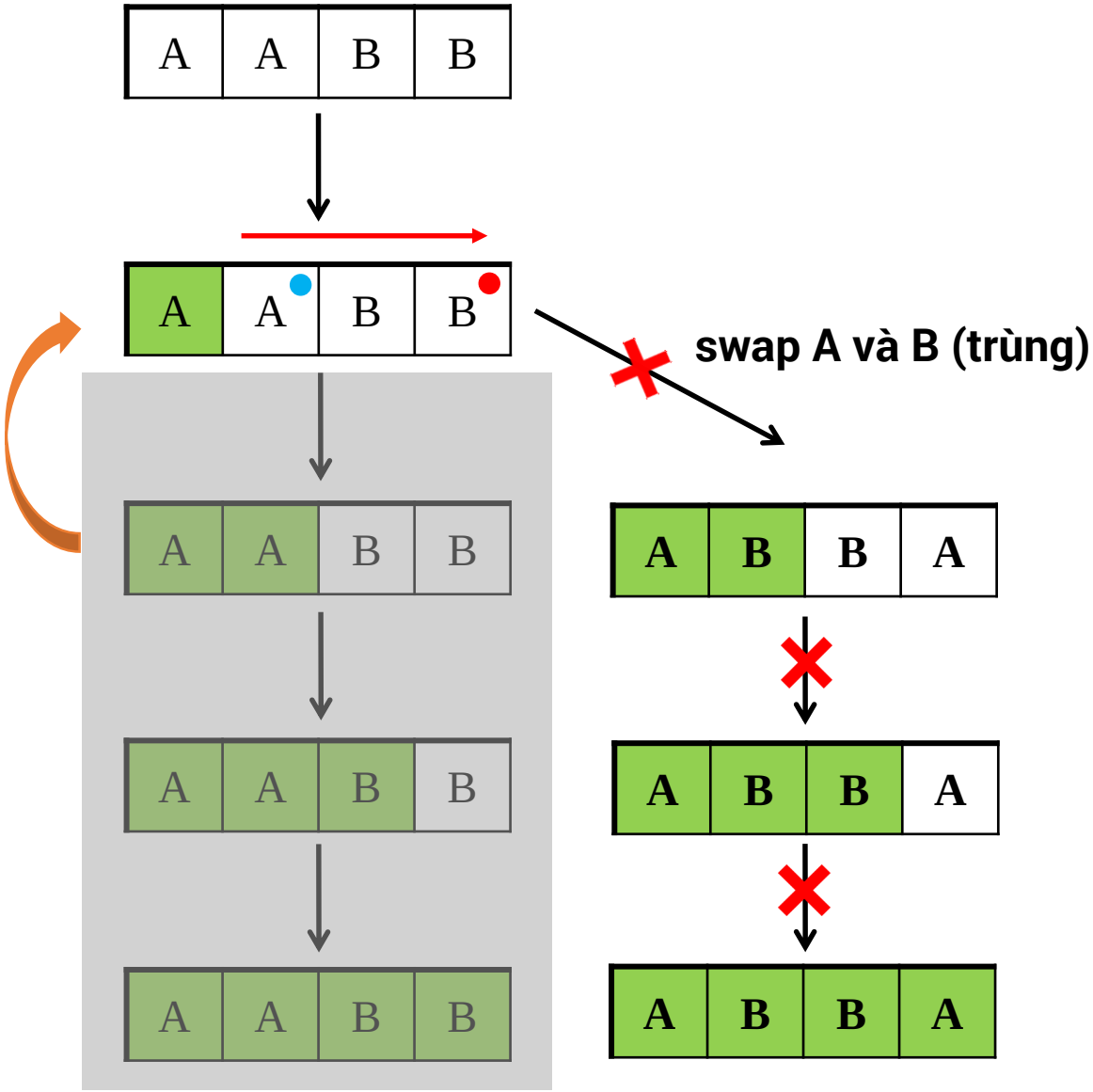
Kết quả:

- AABB, **ABAB**

Bước 4: chạy thuật toán lần 4



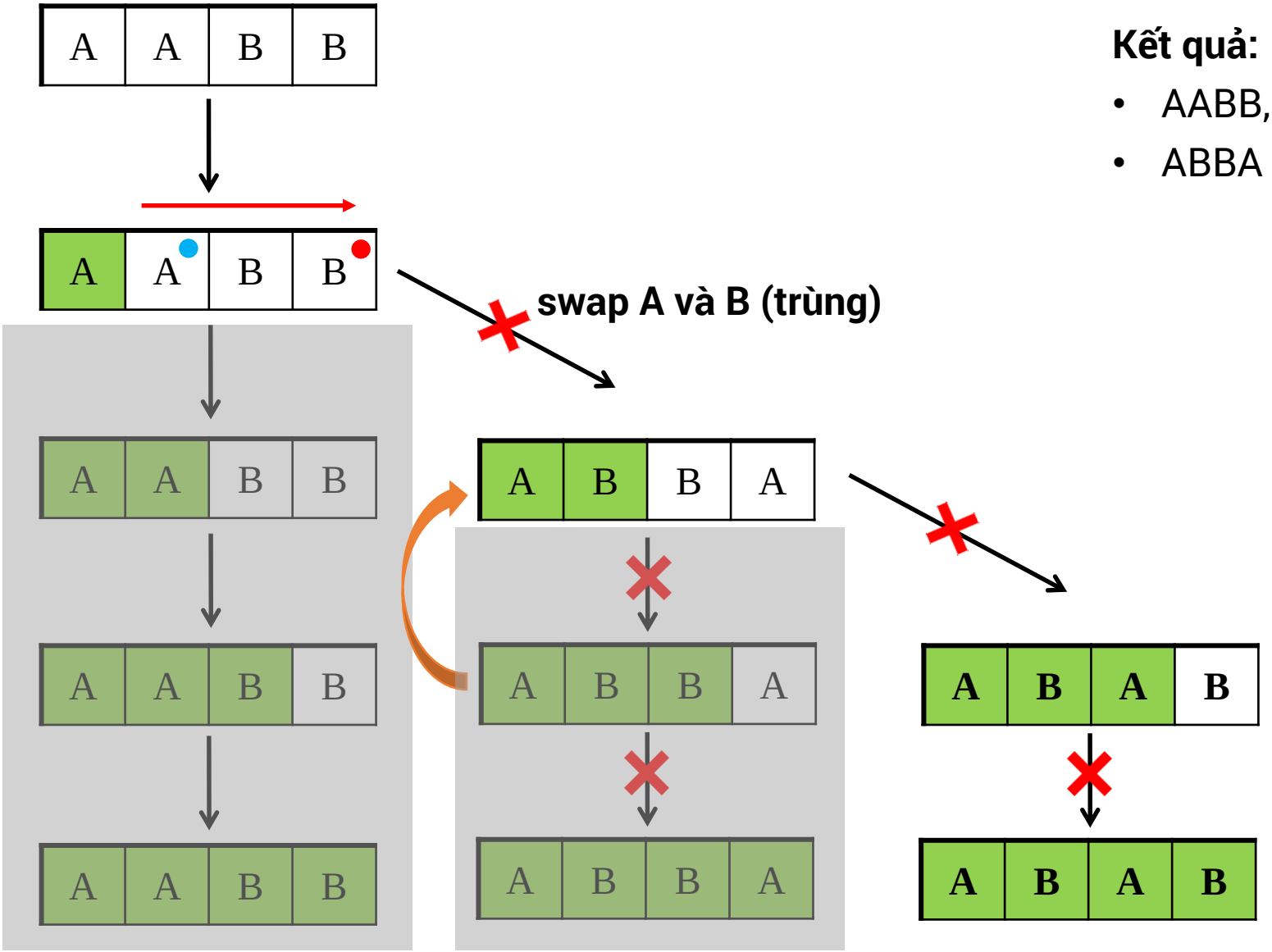
Bước 5: chạy thuật toán lần 5



Kết quả:

- AABB, ABAB
- ABBA

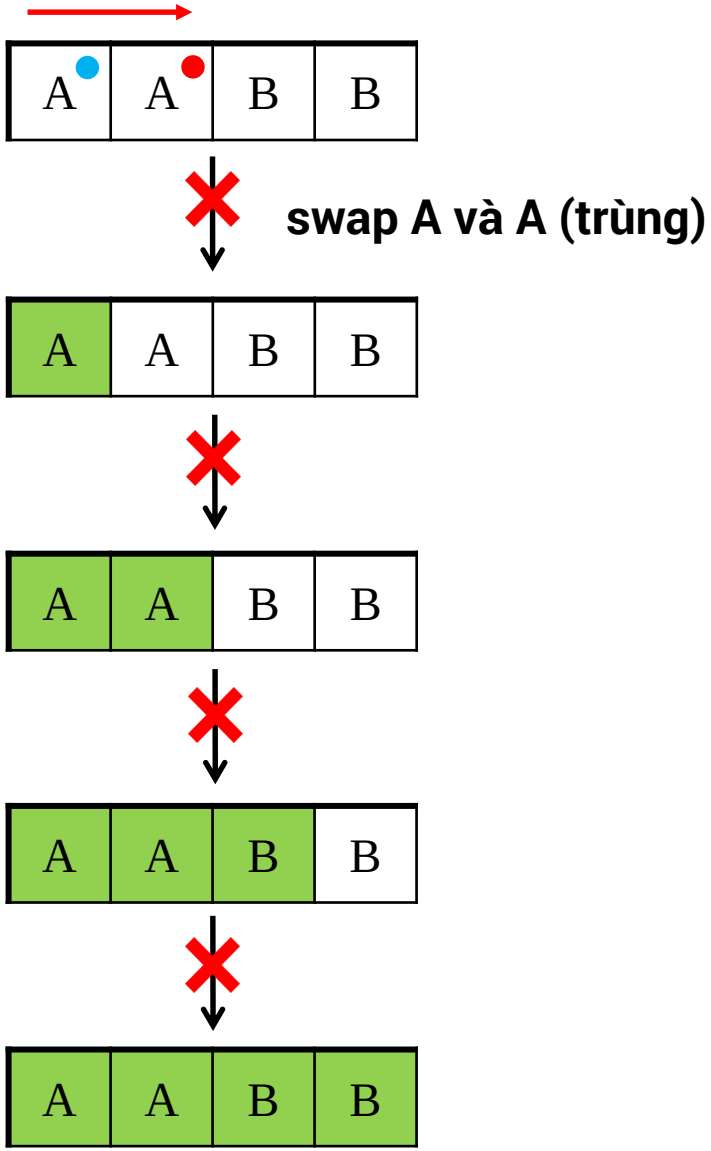
Bước 6: chạy thuật toán lần 6



Kết quả:

- AABB, ABAB
- ABBA

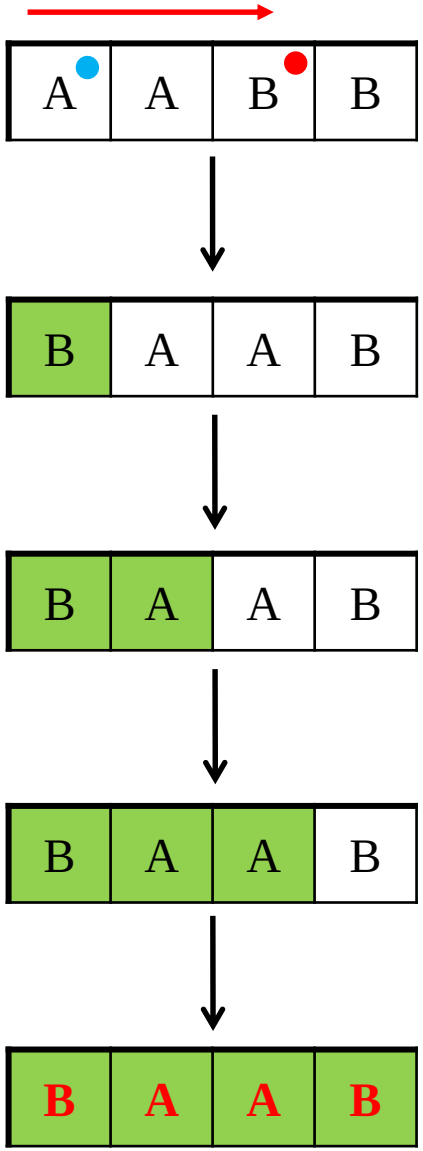
Bước 7: chạy thuật toán lần 7



Kết quả:

- AABB, ABAB
- ABBA

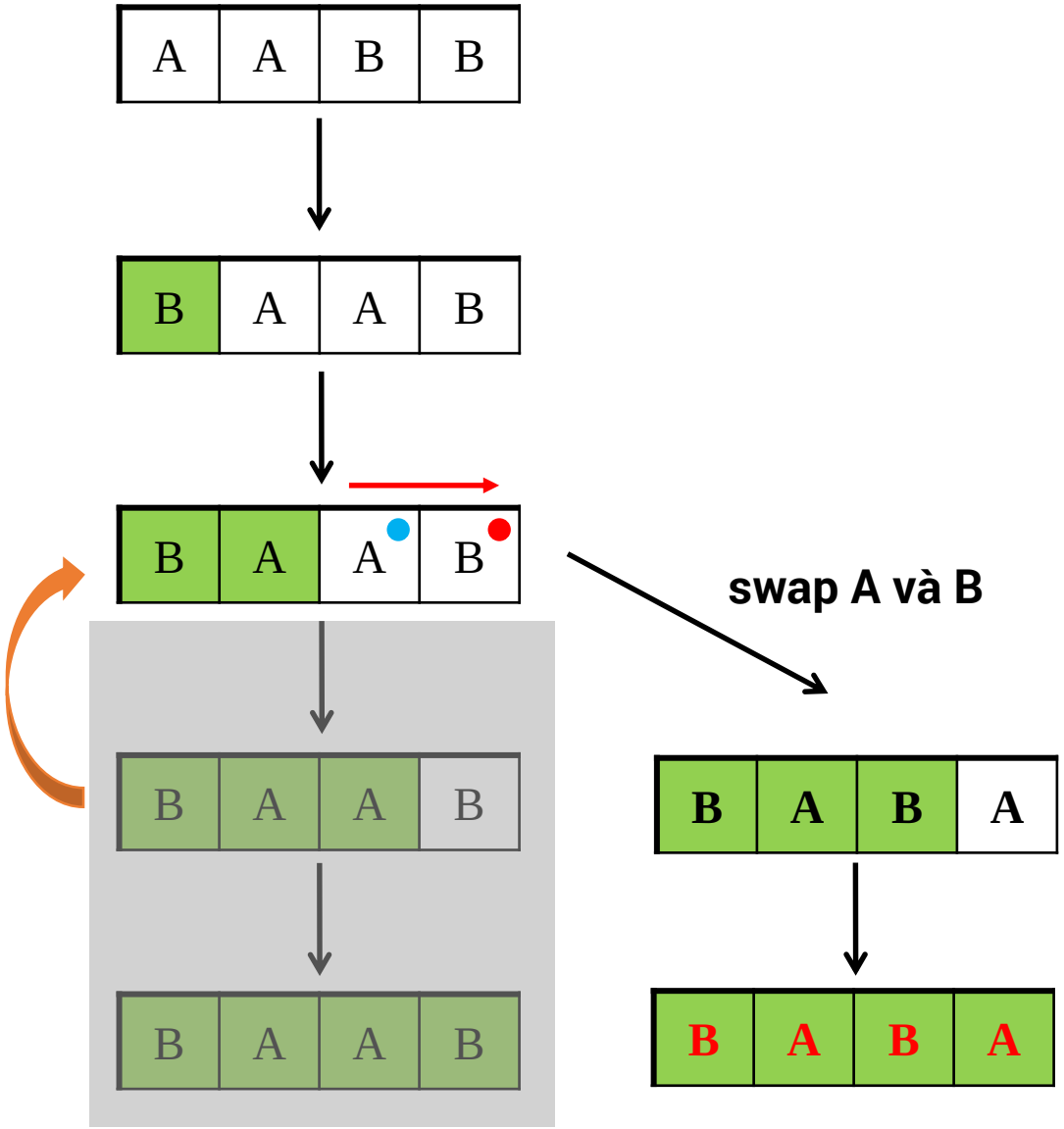
Bước 8: chạy thuật toán lần 8



Kết quả:

- AABB, ABAB
- ABBA, **BAAB**

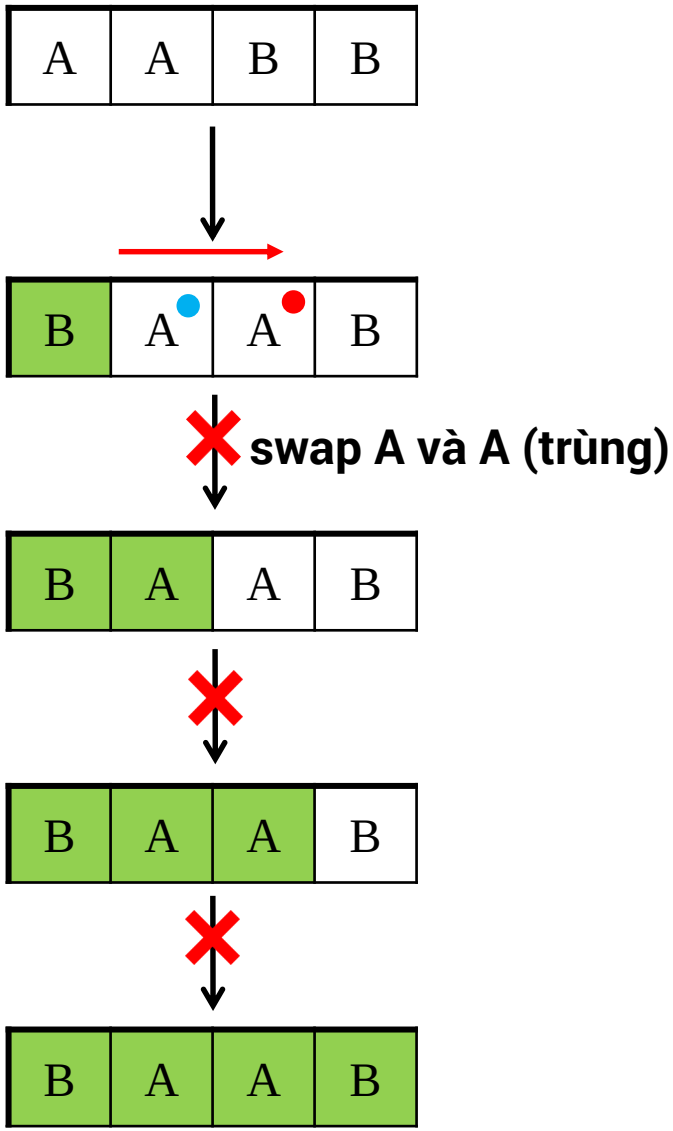
Bước 9: chạy thuật toán lần 9



Kết quả:

- AABB, ABAB
- ABBA, BAAB
- **BABA**

Bước 10: chạy thuật toán lần 10



Kết quả:

- AABB, ABAB
- ABBA, BAAB
- BABA

TIẾP TỤC CHẠY BACKTRACKING CHO CÁC TRƯỜNG HỢP CÒN LẠI

Kết quả bài toán

Cho chuỗi “AABB” tìm tất cả các hoán vị của chuỗi đã cho.

A	A	B	B

Chuỗi AABB có 4 ký tự (n = 4) như vậy sẽ có tổng cộng 6 hoán vị có thể có của chuỗi AABB.

AABB

ABAB

ABBA

BAAB

BABA

BBAA

Source Code Distinct Permutations Of String

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

bool shouldSwap(string s, int start, int end)
{
    for (int i = start; i < end; i++)
        if (s[i] == s[end])
            return false;
    return true;
}
```

Source Code Distinct Permutations Of String

```
11. void distinctPermutations(string s, int l, int r)
12. {
13.     if (l >= r)
14.     {
15.         cout << s << endl;
16.         return;
17.     }
18.     for (int i = l; i < r; i++)
19.     {
20.         bool check = shouldSwap(s, l, i);
21.         if (check==true)
22.         {
23.             swap(s[l], s[i]);
24.             distinctPermutations(s, l + 1, r);
25.             swap(s[l], s[i]);
26.         }
27.     }
28. }
```

Source Code Distinct Permutations Of String

```
11. int main()
12. {
13.     string s = "AABB";
14.     distinctPermutations(s, 0, s.length());
15.     return 0;
16. }
```

Source Code Distinct Permutations Of String

```
1.  def shouldSwap(s, start, end):
2.      for i in range(start, end):
3.          if s[i] == s[end]:
4.              return False
5.      return True

6.  def distinctPermutations(s, l, r):
7.      if l >= r:
8.          print(''.join(s))
9.          return
10.     for i in range(l, r):
11.         check = shouldSwap(s, l, i)
12.         if check == True:
13.             s[l], s[i] = s[i], s[l]
14.             distinctPermutations(s, l + 1, r)
15.             s[l], s[i] = s[i], s[l]
16.   if __name__ == "__main__":
17.     s = list("AABB")
18.     distinctPermutations(s, 0, len(s))
```

Source Code Distinct Permutations Of String

```
1.  import java.lang.StringBuilder;


2.  public class Main {
3.      private static boolean shouldSwap(StringBuilder s, int start, int end){

4.          for (int i = start; i < end; i++)
5.              if (s.charAt(i) == s.charAt(end))
6.                  return false;
7.          return true;
8.      }
```

Source Code Distinct Permutations Of String

```
9.      private static void distinctPermutations(StringBuilder s, int l, int r) {
10.         if (l >= r)
11.             System.out.println(s);
12.         else {
13.             for (int i = l; i < r; i++) {
14.                 boolean check = shouldSwap(s, l, i);
15.                 if (check) {
16.                     char temp = s.charAt(l);
17.                     s.setCharAt(l, s.charAt(i));
18.                     s.setCharAt(i, temp);
19.                     distinctPermutations(s, l + 1, r);
20.                     temp = s.charAt(l);
21.                     s.setCharAt(l, s.charAt(i));
22.                     s.setCharAt(i, temp);
23.                 }
24.             }
25.         }
26.     }
```

Source Code Distinct Permutations Of String

```
27.     public static void main(String[] args) {
28.         StringBuilder s = new StringBuilder("AABB");
29.         distinctPermutations(s, 0, s.length());
30.         return;
31.     }
32. }
```

Nhận xét

Ưu điểm:

- Dễ dàng cài đặt bằng phương pháp đệ quy.
- Các trạng thái được lưu trong stack nên được sử dụng bất kỳ thời điểm nào.

Khuyết điểm:

- Gọi đệ quy nhiều lần, độ phức tạp sẽ rất lớn.
- Yêu cầu dụng lượng vùng nhớ lớn vì phải lưu trữ tất cả trạng thái cần có.

Hỏi đáp