HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Ngăn xếp, hàng đợi

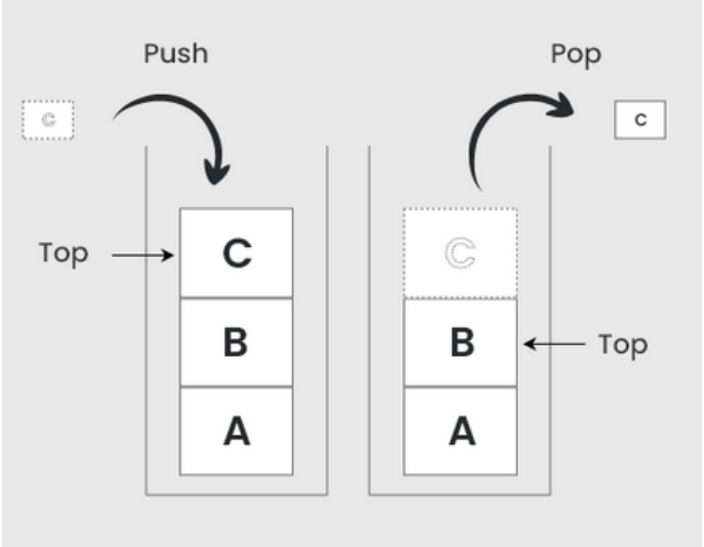
ONE LOVE. ONE FUTURE.

Nội dung

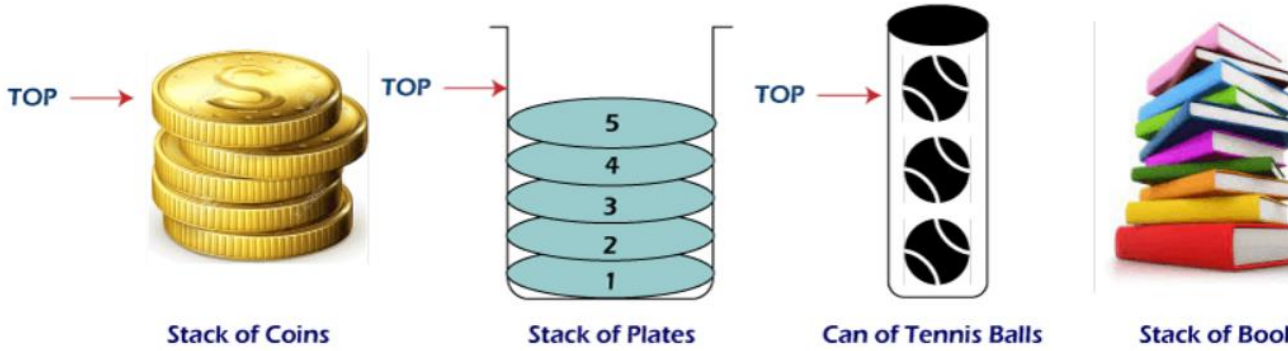
- **Ngăn xếp**
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung

Giới thiệu ngăn xếp

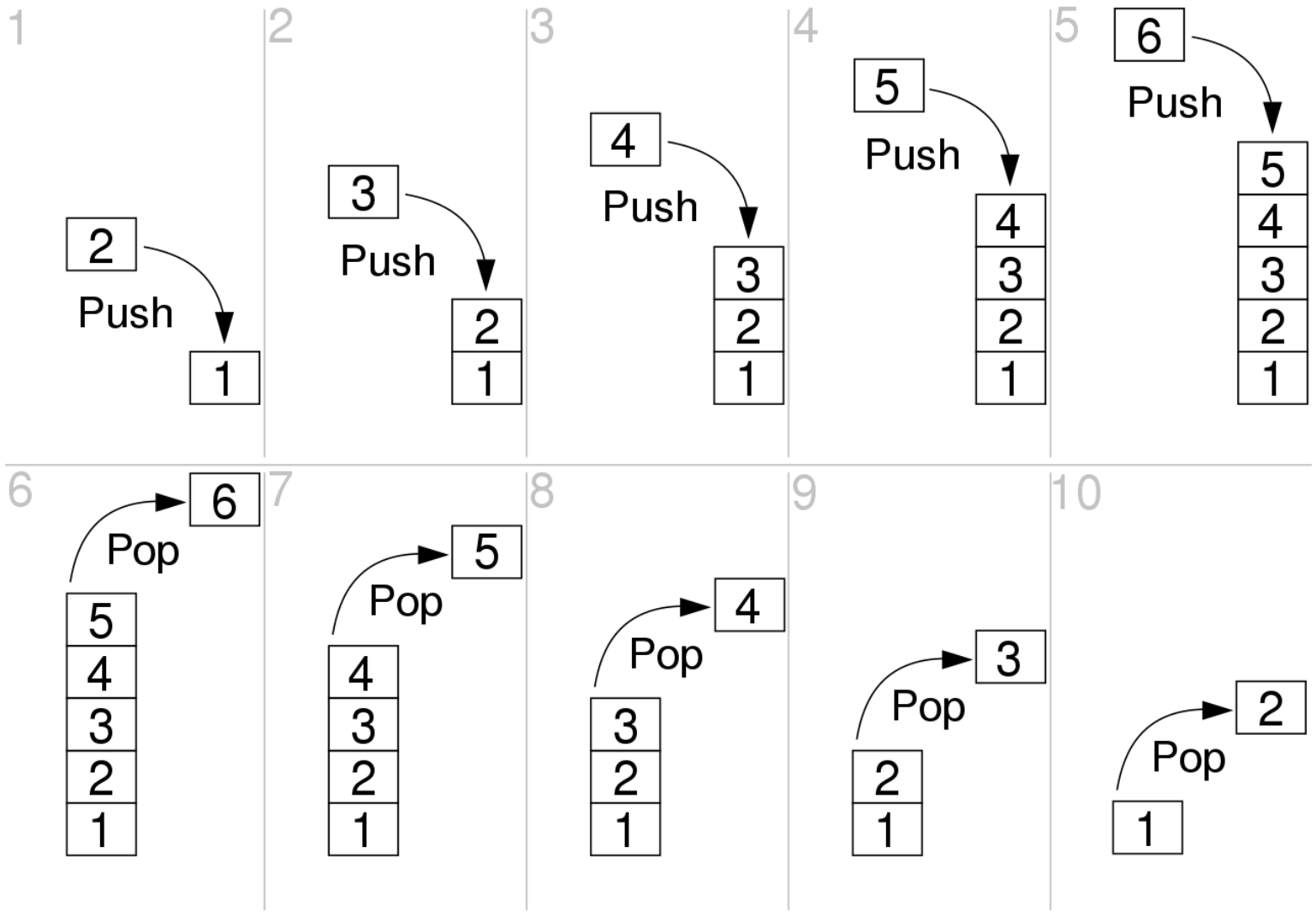
- Ngăn xếp là một cấu trúc dữ liệu tuyến tính (Linear Data Structure);
- Thao tác thêm mới và loại bỏ phần tử trên danh sách được thực hiện ở 1 đầu (đỉnh hay **top**) của danh sách theo nguyên tắc Last In First Out (phần tử cuối cùng được thêm vào ngăn xếp là phần tử đầu tiên bị loại bỏ).



- Thao tác cơ bản trên ngăn xếp *S*:
  - *Push(x, S)*: chèn 1 phần tử x vào ngăn xếp
  - *Pop(S)*: lấy ra 1 phần tử đầu ra khỏi ngăn xếp
  - *Top(S)*: truy cập phần tử ở đỉnh của ngăn xếp
  - *isEmpty(S)*: trả về true nếu ngăn xếp rỗng



Giới thiệu ngăn xếp



Nguồn: https://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Cấu trúc và hàm khởi tạo thành phần

```
typedef struct Node{
    char value;
    struct Node* next;
}Node;


Node*makeNode(char v){
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}
```

Khai báo struct cho một phần tử của ngăn xếp

Tạo một phần tử (node) của ngăn xếp

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Push

```
Node* push(Node * head, char v){
    Node * new_node = makeNode(v);
    if(head == NULL) return new_node;
    else{
        new_node->next = head;
        head = new_node;
        return head;
    }
}
```

Tạo một node mới

Nếu ngăn xếp rỗng, trả về phần tử mới tạo

Nếu ngăn xếp không rỗng, biến node mới tạo thành phần tử đầu danh sách

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Pop

```
Node* pop(Node* head){
    if(head == NULL) return head;
    Node* p = head;
    head = head->next;
    free(p);
    return head;
}
```

Nếu ngăn xếp rỗng, trả về rỗng

Nếu ngăn xếp không rỗng, xóa phần tử đầu của ngăn xếp và trả về phần tử đầu của ngăn xếp

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Top và isEmpty

```
char top(Node* head){
    return head->value;
}

bool isEmpty(Node* head){
    if(head == NULL) return true;

    return false;
}
```

Một số ứng dụng ngăn xếp

- **Ngăn xếp lời gọi hàm:** Ngăn xếp được sử dụng rộng rãi để quản lý cuộc gọi hàm và biến cục bộ trong ngôn ngữ lập trình. Khi một hàm được gọi, ngữ cảnh của nó (bao gồm đối số và biến cục bộ) được đẩy vào ngăn xếp. Khi hàm trả về, ngữ cảnh của nó được đẩy ra khỏi ngăn xếp, cho phép lồng ghép hàm một cách chính xác.
- **Cơ chế quay lại (Undo):** Ngăn xếp được sử dụng trong các ứng dụng yêu cầu tính năng Hoàn tác, như trình soạn thảo văn bản, phần mềm đồ họa hoặc hệ thống quản lý phiên bản. Mỗi hành động của người dùng có thể được đẩy vào ngăn xếp, và việc hoàn tác một hành động liên quan đến việc lấy nó ra khỏi ngăn xếp để hoàn ngược thay đổi.

Một số ứng dụng ngăn xếp

- **Thuật toán Backtracking:** Trong các thuật toán như tìm kiếm theo chiều sâu (DFS) và các thuật toán backtracking (**Ví dụ:** giải quyết các câu đố như vấn đề N-Queens hoặc Sudoku), một ngăn xếp có thể được sử dụng để theo dõi các nút hoặc trạng thái đã được thăm. Điều này cho phép dễ dàng quay lại để khám phá các lựa chọn thay thế khi cần.
- **Phân tích Biểu thức:** Ngăn xếp được sử dụng để phân tích và hiểu biểu thức trong các trình biên dịch và trình thông dịch. Chúng giúp duy trì thứ tự ưu tiên của các toán tử và đánh giá biểu thức một cách chính xác.

Một số ứng dụng ngăn xếp

- **Quản lý Bộ nhớ:** Ngăn xếp đóng vai trò quan trọng trong quản lý bộ nhớ trong các hệ thống máy tính. Chúng được sử dụng để quản lý ngăn xếp gọi hàm, nơi lưu trữ thông tin về cuộc gọi hàm và biến cục bộ. Ngăn xếp giúp phân bổ bộ nhớ cho cuộc gọi hàm và giải phóng nó khi các hàm trả về, ngăn chặn rò rỉ bộ nhớ.
- **Phân tích Biểu thức:** Ngăn xếp được sử dụng để phân tích và hiểu biểu thức trong các trình biên dịch và trình thông dịch. Chúng giúp duy trì thứ tự ưu tiên của các toán tử và đánh giá biểu thức một cách chính xác.

Nội dung

- Ngăn xếp
- **Bài tập: Kiểm tra tính cân xứng của dãy ngoặc**
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung

Bài toán kiểm tra tính cân xứng của dãy ngoặc

- Cho dãy dấu ngoặc E mà mỗi phần tử là một dấu ngoặc thuộc một trong các loại: (,), [,], {, }. Viết chương trình kiểm tra xem dãy ngoặc đó có cân xứng (balanced) hay không?
- Ví dụ:
  - ( ) [{ } ([ ])]: cân xứng
  - ( ) [{ } ([ ] }]: không cân xứng

Bài toán kiểm tra tính cân xứng của dãy ngoặc

- Dữ liệu đầu vào:
  - Một dòng duy nhất chứa xâu ký tự thể hiện dãy dấu ngoặc
- Kết quả đầu ra:
  - Ghi 1 nếu dãy dấu ngoặc là cân xứng và ghi 0, nếu dãy dấu ngoặc không cân xứng

stdin	stdout
( ) [{ } ([ ])]	1

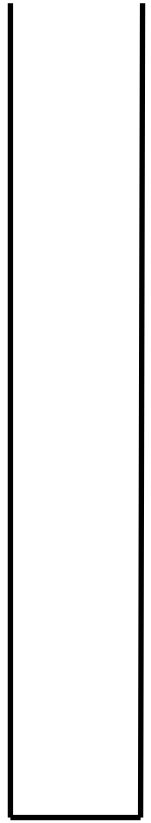
stdin	stdout
( ) [{ } ([ ] }]	0

Bài toán kiểm tra tính cân xứng của dãy ngoặc

- **Thuật toán**
  - Khởi tạo một ngăn xếp rỗng S
  - Duyệt dãy dấu ngoặc từ trái qua phải
    - Nếu gặp ngoặc mở A thì đưa ngoặc mở đó vào S
    - Nếu gặp ngoặc đóng B
      - Nếu S rỗng thì kết luận dãy ngoặc E không cân xứng
      - Nếu S không rỗng
        - Lấy một ngoặc mở A khỏi ngăn xếp S
        - Nếu A và B không cân xứng (ngoặc mở và đóng khác loại) thì kết luận E không cân xứng
  - Kết thúc duyệt, nếu S không rỗng thì kết luận E không cân xứng, ngược lại thì E là cân xứng

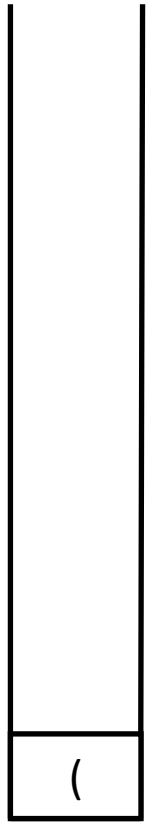
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Khởi tạo ngăn xếp rỗng, thực hiện duyệt dãy ngoặc từ trái qua phải



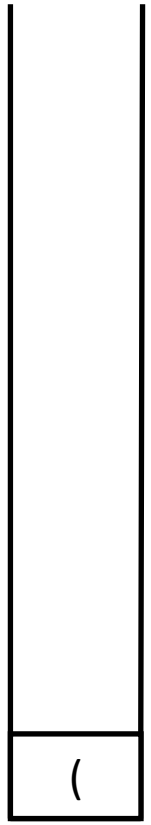
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp



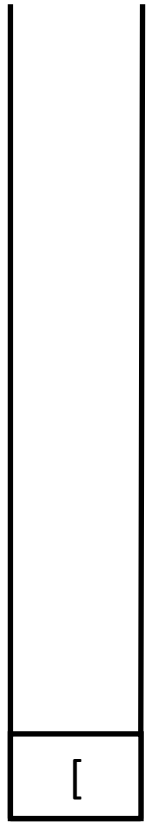
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp
  - Bước 2: Xét ngoặc tiếp theo là ngoặc đóng “)” → lấy ngoặc mở ra khỏi ngăn xếp và so khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt



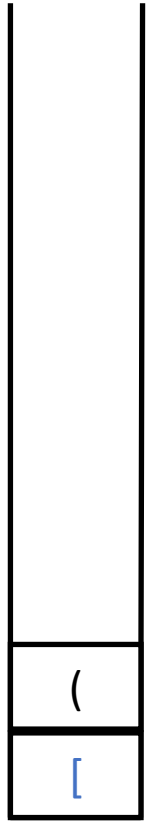
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp
  - Bước 2: Xét ngoặc tiếp theo là ngoặc đóng “)” → lấy ngoặc mở ra khỏi ngăn xếp và so khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 3: Gặp ngoặc mở “[“ → đưa ngoặc mở này vào ngăn xếp



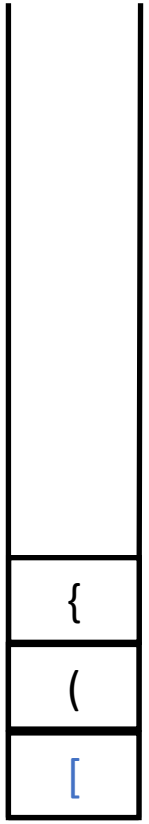
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở vào ngăn xếp
  - Bước 2: Xét ngoặc tiếp theo là ngoặc đóng ")" → lấy ngoặc mở ra khỏi ngăn xếp và so khớp "(")" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 3: Gặp ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
  - Bước 4: Gặp ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp



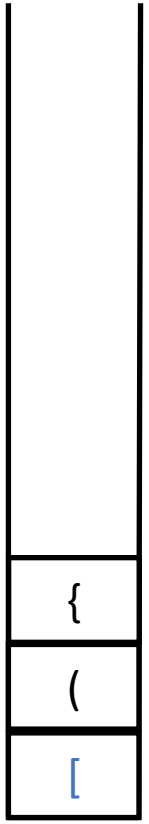
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở vào ngăn xếp
  - Bước 2: Xét ngoặc tiếp theo là ngoặc đóng ")" → lấy ngoặc mở ra khỏi ngăn xếp và so khớp "(")" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 3: Gặp ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
  - Bước 4: Gặp ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp



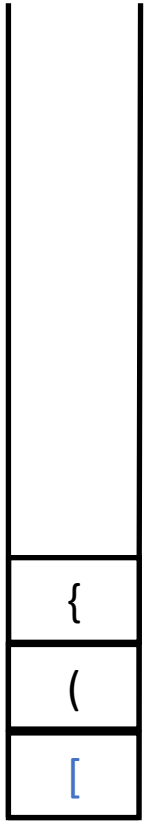
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 1: Xét ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở vào ngăn xếp
  - Bước 2: Xét ngoặc tiếp theo là ngoặc đóng ")" → lấy ngoặc mở ra khỏi ngăn xếp và so khớp "(")" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 3: Gặp ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
  - Bước 4: Gặp ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp



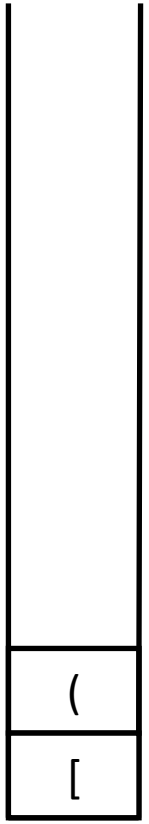
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp
    - So khớp "{" "}" → kết quả là đúng nên ta tiếp tục duyệt



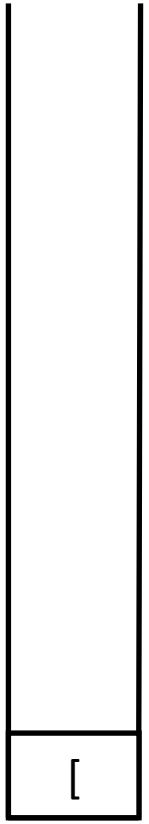
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp
    - So khớp "{" "}" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 7: Gặp ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp



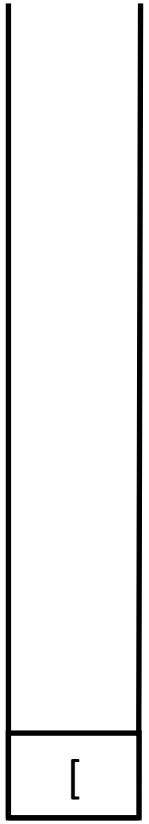
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp
    - So khớp "{" "}" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 7: Gặp ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả là đúng nên ta tiếp tục duyệt



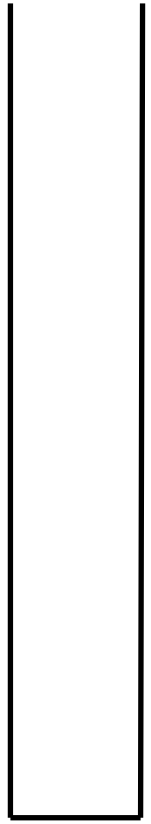
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp
    - So khớp "{" "}" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 7: Gặp ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 8: Gặp ngoặc đóng "]" → lấy 1 ngoặc mở là "[" ra khỏi ngăn xếp



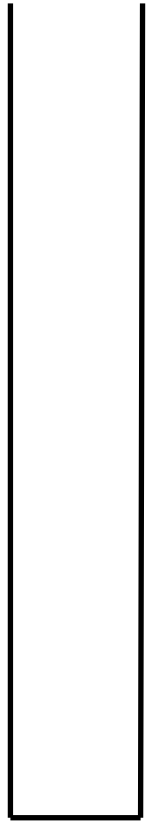
Minh họa

- **Minh họa số 1:** dãy ngoặc ( ) [({ })]
  - Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
  - Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp
    - So khớp "{" "}" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 7: Gặp ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả là đúng nên ta tiếp tục duyệt
  - Bước 8: Gặp ngoặc đóng "]" → lấy 1 ngoặc mở là "[" ra khỏi ngăn xếp
    - So khớp "[" "]" → lúc này dãy ngoặc đã duyệt xong và ngăn xếp rỗng, nên kết quả dãy ngoặc này là cân xứng

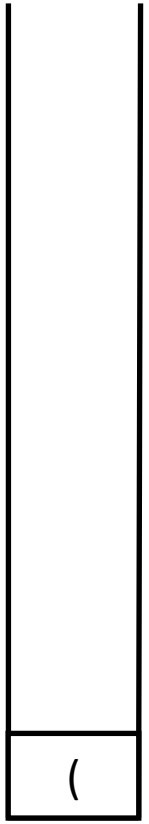


Minh họa

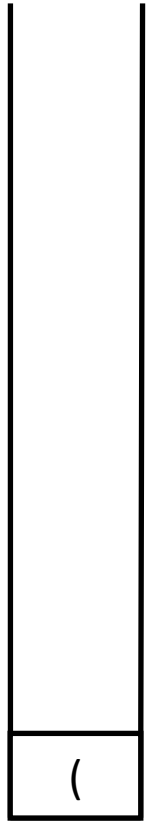
- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Khởi tạo ngăn xếp rỗng và duyệt dãy ngoặc từ trái qua phải



- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp

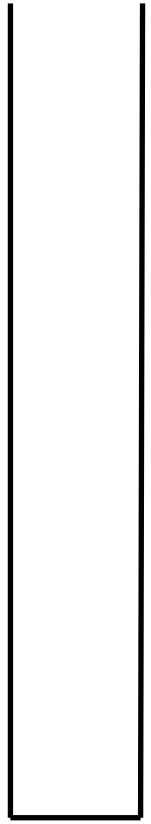


- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp



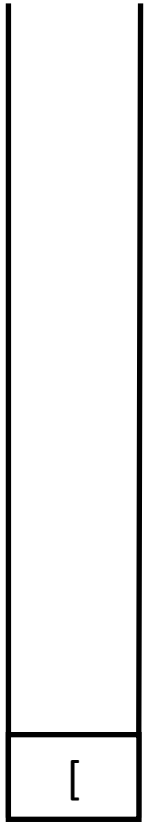
Lato

- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
  - Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
    - So khớp “(“ “)” → kết quả đúng nên ta duyệt tiếp



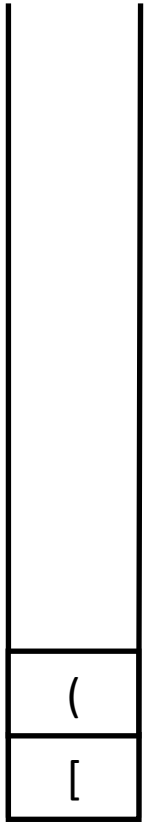
Minh họa

- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả đúng nên ta duyệt tiếp
  - Bước 3: Gặp ngoặc tiếp theo là ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp



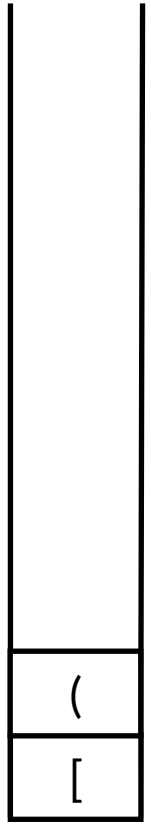
Minh họa

- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả đúng nên ta duyệt tiếp
  - Bước 3: Gặp ngoặc tiếp theo là ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp



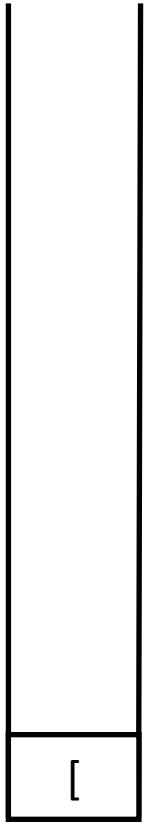
Minh họa

- **Minh họa số 2:** dãy ngoặc ( ) [(} })]
  - Bước 1: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng ")" → lấy 1 ngoặc mở là "(" ra khỏi ngăn xếp
    - So khớp "(" ")" → kết quả đúng nên ta duyệt tiếp
  - Bước 3: Gặp ngoặc tiếp theo là ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
  - Bước 4: Gặp ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
  - Bước 5: Gặp ngoặc tiếp theo là ngoặc đóng "}" → lấy 1 ngoặc mở là ngoặc "(" ra



Minh họa

- Bước 4: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 5: Gặp ngoặc tiếp theo là ngoặc đóng “}” → lấy 1 ngoặc mở là ngoặc “(“ ra
  - So khớp “(“ “}” → kết quả so khớp là sai nên ta kết luận dãy ngoặc đã cho không cân xứng



Cài đặt thuật toán

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct Node{
        char c;
        struct Node* next;
}Node;
Node* top;
char s[1000001];
Node* makeNode(char c){
        Node* p=(Node*)malloc(sizeof(Node));
        p->c = c; p->next = NULL; return p;
}
```

Cài đặt thuật toán

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct Node{
        char c;
        struct Node* next;
}Node;
Node* top;
char s[1000001];
Node* makeNode(char c){
        Node* p=(Node*)malloc(sizeof(Node));
        p->c = c; p->next = NULL; return p;
}
```

```
void push(char c){
        Node* p = makeNode(c);
        p->next = top; top = p;
}
char pop(){
        if(top == NULL) return ' ';
        Node* tmp = top; top = top->next;
        char res = tmp->c;
        free(tmp);
        return res;
}
```

Cài đặt thuật toán

```
int match(char a, int b){
  if(a == '(' && b == ')') return 1;
  if(a == '{' && b == '}') return 1;
  if(a == '[' && b == ']') return 1;
  return 0;
}
```

Cài đặt thuật toán

```
int match(char a, int b){
  if(a == '(' && b == ')') return 1;
  if(a == '{' && b == '}') return 1;
  if(a == '[' && b == ']') return 1;
  return 0;
}
```

```
int check(char* s){
  for(int i = 0; i < strlen(s); i++){
    if(s[i] == '(' || s[i] == '{' || s[i] == '[')
      push(s[i]);
    else{
      if(top==NULL) return 0;
      char o = pop();
      if(!match(o,s[i])) return 0;
    }
  }
  return top == NULL;
}
```

Cài đặt thuật toán

```
int match(char a, int b){
  if(a == '(' && b == ')') return 1;
  if(a == '{' && b == '}') return 1;
  if(a == '[' && b == ']') return 1;
  return 0;
}
```

```
int main(){
  scanf("%s",s);
  int res = check(s);
  printf("%d",res);
  return 0;
}
```

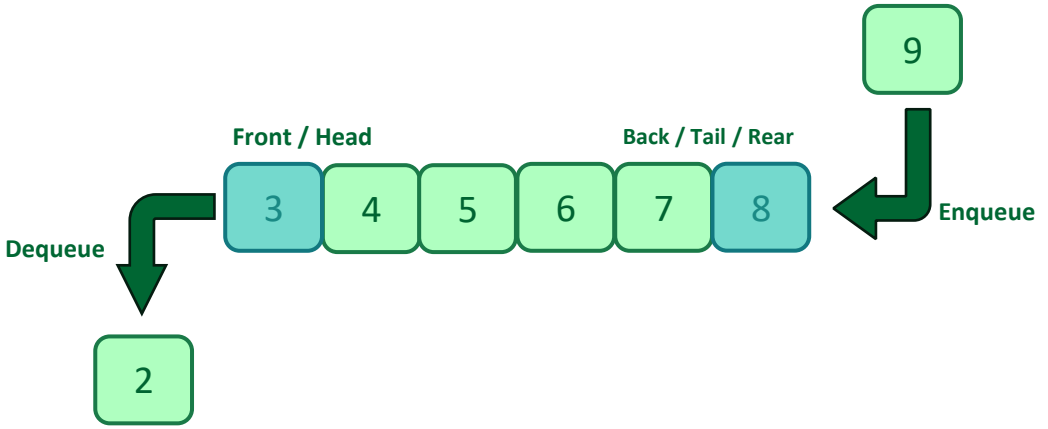
```
int check(char* s){
  for(int i = 0; i < strlen(s); i++){
    if(s[i] == '(' || s[i] == '{' || s[i] == '[')
      push(s[i]);
    else{
      if(top==NULL) return 0;
      char o = pop();
      if(!match(o,s[i])) return 0;
    }
  }
  return top == NULL;
}
```

Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- **Hàng đợi**
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung

Giới thiệu hàng đợi

- Hàng đợi là một cấu trúc dữ liệu tuyến tính có hai đầu, head và tail
- Thao tác thêm mới phần tử được thực hiện ở tail
  Thao tác loại bỏ phần tử được thực hiện ở head
- Nguyên tắc hoạt động: FIFO – First In First Out
- Thao tác cơ bản trên hàng đợi Q:
  - enqueue(x, Q): chèn 1 phần tử x vào hàng đợi
  - dequeue(Q): lấy ra 1 phần tử khỏi hàng đợi
  - isEmpty(Q): trả về true nếu hàng đợi rỗng



Ý tưởng hoạt động của hàng đợi

Nguồn: https://www.geeksforgeeks.org/queue-data-structure/

Minh họa hoạt động của hàng đợi

Thao tác	Trạng thái hàng đợi	Phần tử trả về nếu có
Create an empty queue	Empty	-
Enqueue 1	1	-
Enqueue 2	1, 2	-
Enqueue 3	1, 2, 3	-
Dequeue	2, 3	1
Enqueue 4	2, 3, 4	-
Enqueue 5	2, 3, 4, 5	-
Dequeue	3, 4, 5	2
Dequeue	4, 5	3
Check if empty	4, 5	0

Cài đặt hàng đợi sử dụng danh sách liên kết đơn

- Cấu trúc và hàm khởi tạo thành phần

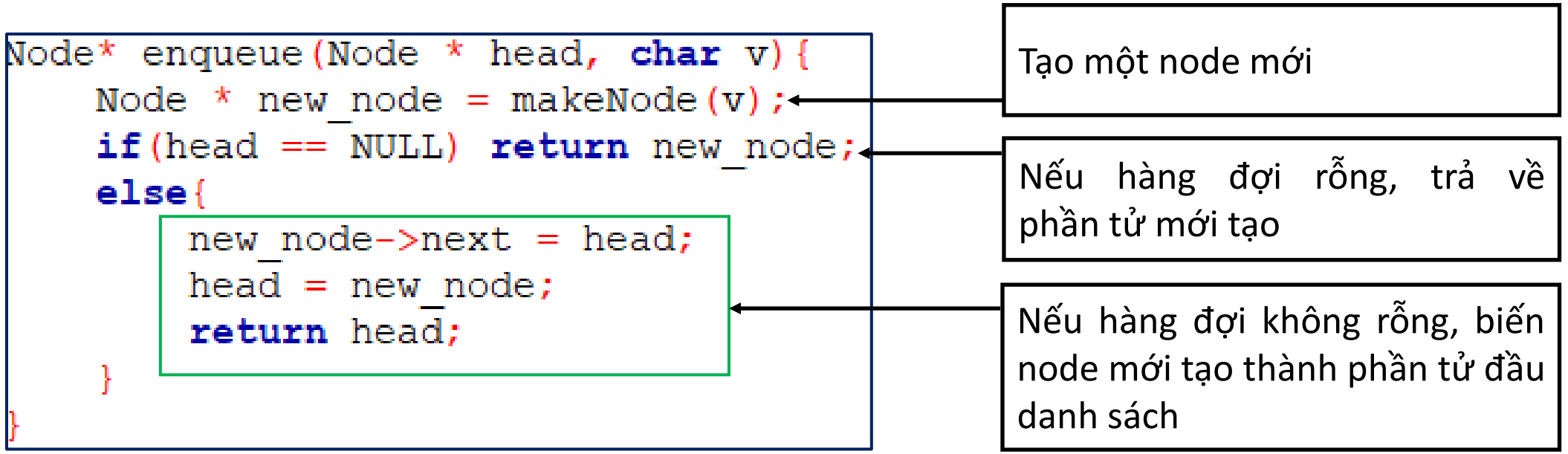
```
typedef struct Node{
    char value;
    struct Node* next;
}Node;


Node*makeNode(char v){
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}
```

Khai báo struct cho một phần tử của hàng đợi

Tạo một phần tử (node) của hàng đợi

Cài đặt hàng đợi sử dụng danh sách liên kết đơn

- Thao tác Enqueue



Cài đặt hàng đợi sử dụng danh sách liên kết đơn

- Thao tác Dequeue

```
char dequeue(Node** head){
    if(*head == NULL) return NULL;
    Node* p, *q;
    p = *head;
    char ans;

    if(p->next == NULL){
        ans = p->value;
        free(p);
        *head = NULL;
        return ans;
    }

    while(p->next->next != NULL){
        p = p->next;
    }
    q = p->next;
    p->next = NULL;
    ans = q->value;
    free(q);

    return ans;
}
```

Nếu hàng đợi rỗng, trả về rỗng

Nếu hàng đợi chỉ có một phần tử

Nếu hàng đợi có nhiều hơn một phần tử, lấy phần tử cuối cùng của danh sách

Một số ứng dụng của hàng đợi

- **Tìm kiếm theo chiều rộng (BFS):** BFS là một thuật toán để duyệt hoặc tìm kiếm trong cấu trúc dữ liệu cây và đồ thị. Nó sử dụng một hàng đợi để khám phá các nút theo từng cấp độ, làm cho nó trở thành công cụ quan trọng để giải quyết các vấn đề liên quan đến đồ thị.
- **Lập lịch công việc (Task scheduling)**: Hàng đợi được sử dụng trong các hệ điều hành để lên lịch cho các tác vụ hoặc tiến trình để thực thi. Các tác vụ được đặt vào hàng đợi và hệ điều hành thực hiện chúng theo thứ tự chúng được thêm vào, đảm bảo sự công bằng trong phân phối tài nguyên.

Một số ứng dụng của hàng đợi

- **Xử lý yêu cầu trên máy chủ web (Web server request handling):** Các máy chủ web sử dụng hàng đợi để quản lý các yêu cầu HTTP đến. Mỗi yêu cầu đến được đặt vào hàng đợi và xử lý bởi các luồng hoặc tiến trình làm việc, cho phép máy chủ xử lý nhiều yêu cầu cùng một lúc
- **Đệm trong các hoạt động I/O (Buffering in I/O operations):** Hàng đợi được sử dụng để đệm dữ liệu trong các hoạt động I/O. Ví dụ, khi dữ liệu được đọc từ tệp hoặc ổ đĩa mạng, thường được đặt vào hàng đợi trước khi xử lý để làm dịu sự biến đổi trong tốc độ nhận dữ liệu.

Một số ứng dụng của hàng đợi

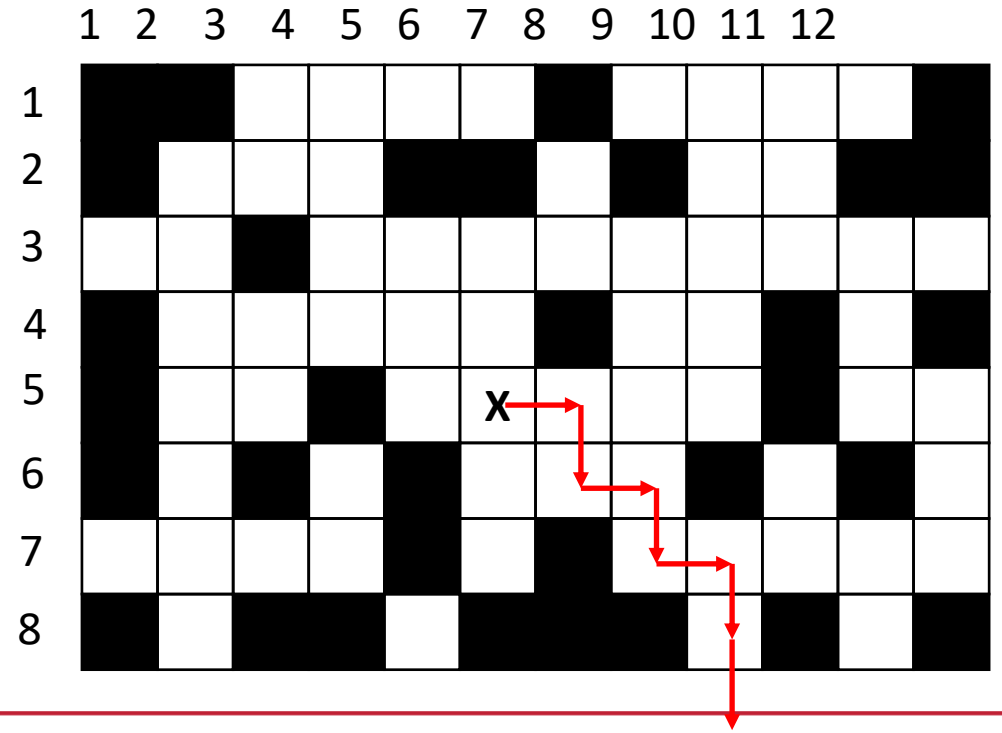
- **Quản lý tác vụ trong ứng dụng đa luồng (Task management in multithreading):** Trong các ứng dụng đa luồng, hàng đợi có thể được sử dụng để quản lý các tác vụ cần được thực thi đồng thời. Các luồng thực hiện bóc tác vụ từ hàng đợi và thực hiện công việc tương ứng.
- **Xử lý đơn đặt hàng trong kho (Order fulfillment in warehouses):** Trong quản lý logistics và kho lưu trữ, hàng đợi có thể được sử dụng để quản lý quy trình thực hiện đơn đặt hàng. Đơn đặt hàng được đặt vào hàng đợi để lựa chọn, đóng gói và giao hàng để đảm bảo xử lý hiệu quả.

Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- **Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung**

Đề bài

- **Đề bài:** Một mê cung hình chữ nhật được biểu diễn bởi 0-1 ma trận NxM trong đó A[i,j] = 1 thể hiện ô (i,j) là tường gạch và A[i,j] = 0 thể hiện ô (i,j) là ô trống, có thể di chuyển vào. Từ 1 ô trống, ta có thể di chuyển sang 1 trong 4 ô lân cận (lên trên, xuống dưới, sang trái, sang phải) nếu ô đó là ô trống.
  - Xuất phát từ 1 ô trống trong mê cung, hãy tìm đường ngắn nhất thoát ra khỏi mê cung



Đề bài

- Dữ liệu đầu vào:
  - Dòng 1: ghi 4 số nguyên dương n, m, r, c trong đó n và m tương ứng là số hàng và cột của ma trận A (1 <= n,m <= 999) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
  - Dòng i+1 (i=1,...,n): ghi dòng thứ i của ma trận A
- Kết quả đầu ra:
  - Ghi ra số bước cần di chuyển ngắn nhất để thoát ra khỏi mê cung, hoặc ghi giá trị -1 nếu không tìm thấy đường đi nào thoát ra mê cung.

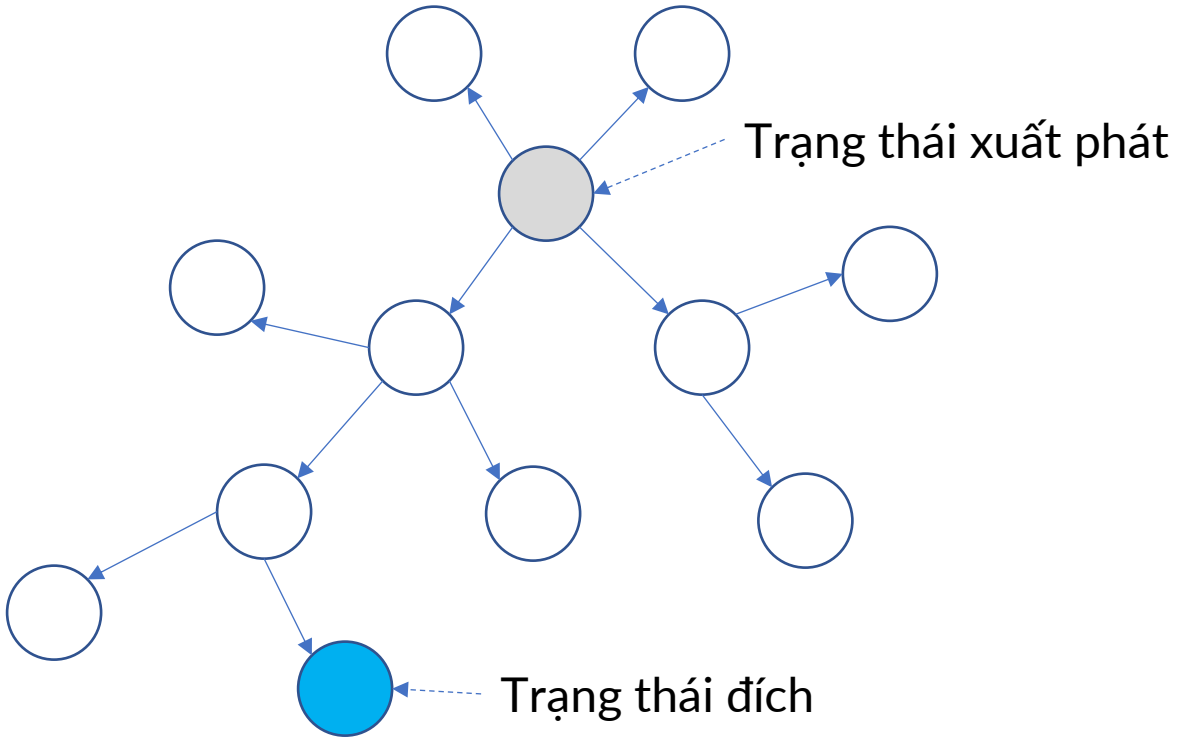
Đề bài

- Dữ liệu đầu vào:
  - Dòng 1: ghi 4 số nguyên dương n, m, r, c trong đó n và m tương ứng là số hàng và cột của ma trận A (1 <= n,m <= 999) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
  - Dòng i+1 (i=1,...,n): ghi dòng thứ i của ma trận A
- Kết quả đầu ra:
  - Ghi giá số bước cần di chuyển ngắn nhất để thoát ra khỏi mê cung, hoặc ghi giá trị -1 nếu không tìm thấy đường đi nào thoát ra mê cung.

stdin	stdout
8 12 5 6 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1	7

Thuật toán

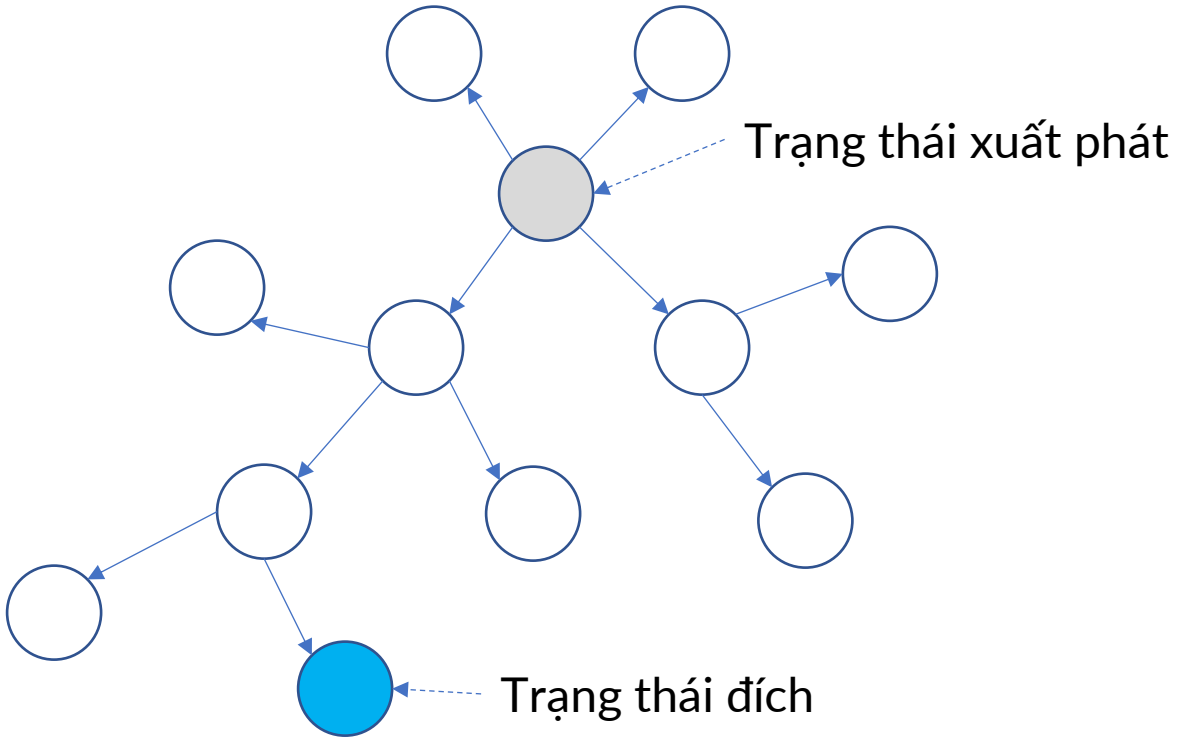
- Thuật toán tìm kiếm theo chiều rộng (loang) tìm đường đi ngắn nhất trên mô hình chuyển trạng thái:
  - Trạng thái đầu
  - Trạng thái đích
  - Mỗi trạng thái *s* sẽ có 1 tập *N*(*s*) các trạng thái lân cận



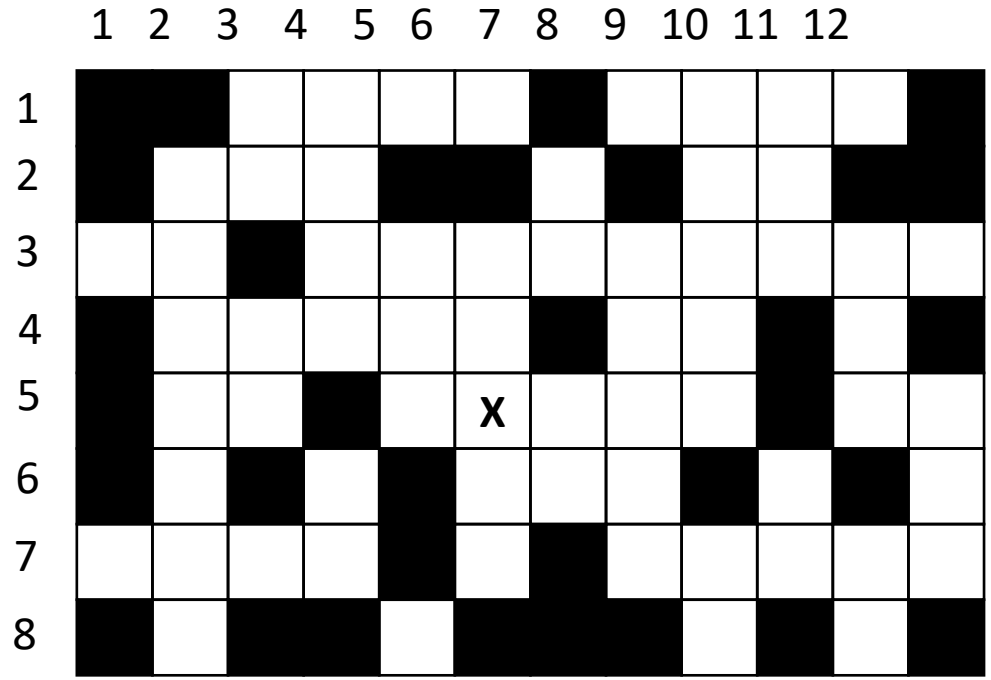
Thuật toán

- Thuật toán tìm kiếm theo chiều rộng (loang) tìm đường đi ngắn nhất trên mô hình chuyển trạng thái:

```
findPath(s0, N){
  Khởi tạo hàng đợi Queue
  Queue.PUSH(s0);
  visited[s0] = true;
  while Queue not empty do{
    s = Queue.POP();
    for x ∈ N(s) do{
       if not visited[x] and check(x) then{
            if target(x) then
               return x;
            else{
               Queue.PUSH(x);
               visited[x] = true;
            }
       }
    }
  }
}
```



Minh họa



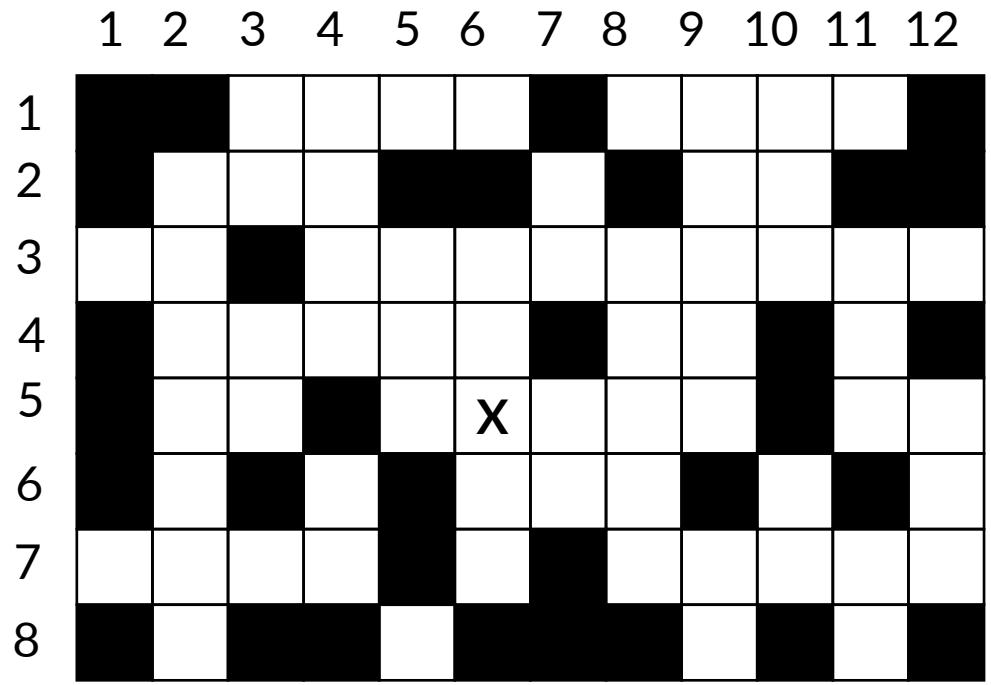
Khởi tạo hàng đợi Q rỗng

Đưa trạng thái (5,6) vào Q

	(5,6)	

Minh họa

- **Minh họa**



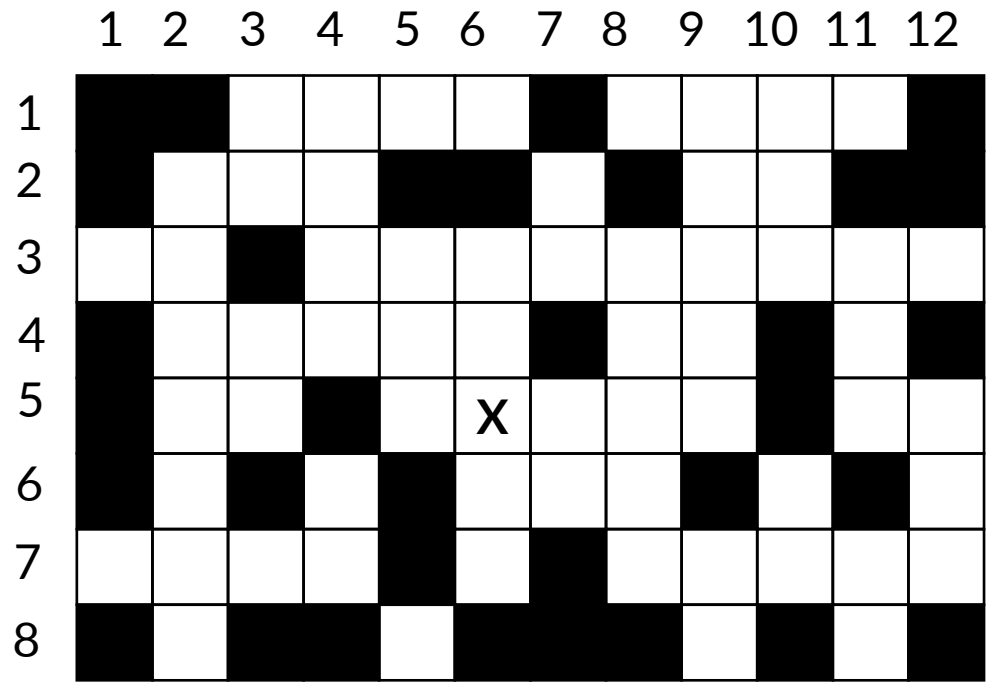
Lấy (5,6) ra khỏi Q

Đưa trạng thái (5,7), (5, 5), (4, 6), (6,6) vào Q

	(5,6)	(5,7)	(5,5)	(4,6)	(6,6)	

Minh họa

- **Minh họa**



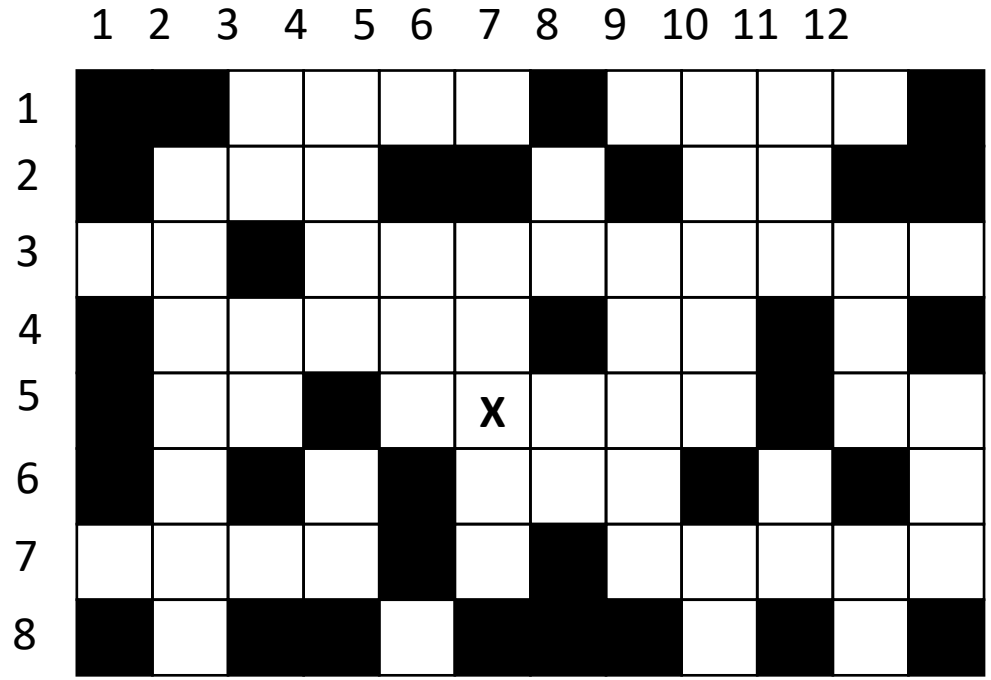
Lấy (5,7) ra khỏi Q

Đưa trạng thái (6,7), (5, 8) vào Q

(5,7)	(5,5)	(4,6)	(6,6)	(6,7)	(5,8)

Minh họa

- **Minh họa**



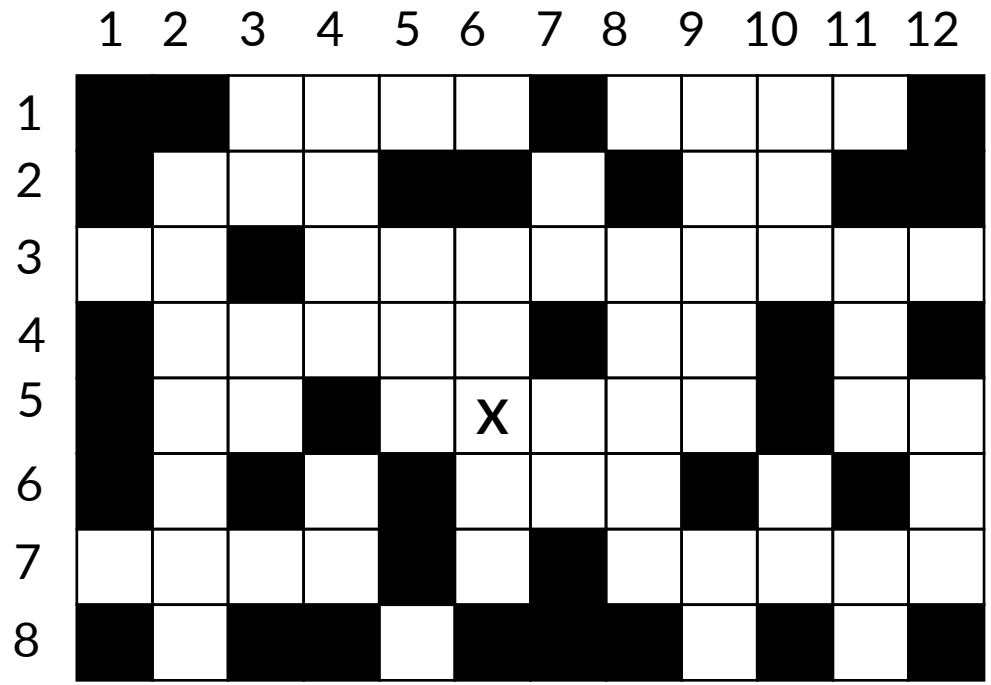
Lấy (5,5) ra khỏi Q

Đưa trạng thái (4,5) vào Q

	~~(5,5)~~	(4,6)	(6,6)	(6,7)	(5,8)	(4,5)	

Minh họa

- **Minh họa**

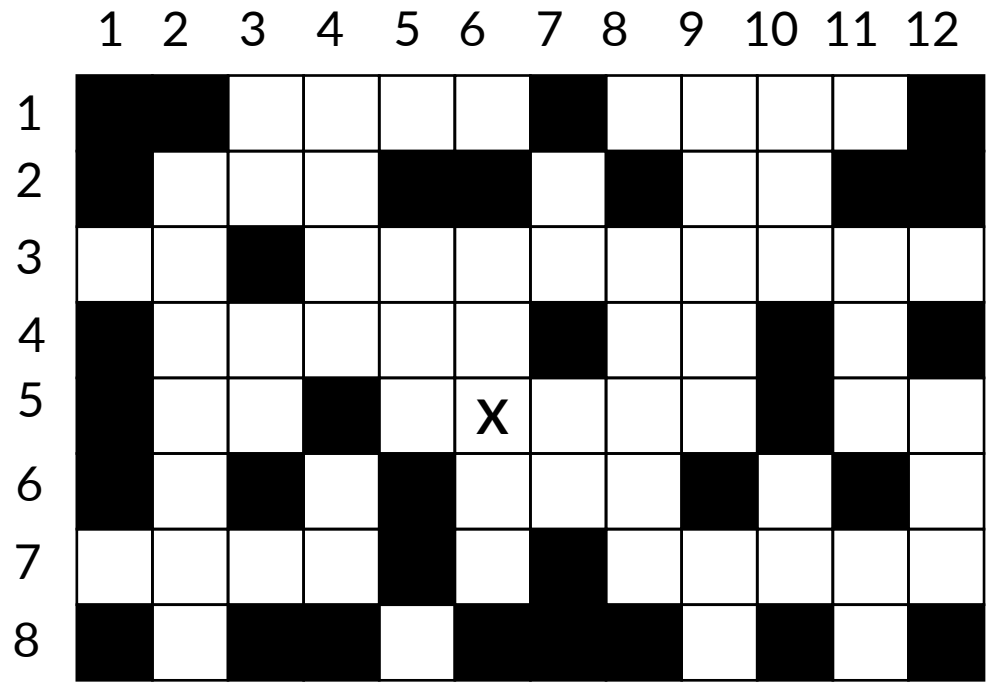


Lấy (4,6) ra khỏi Q

Đưa trạng thái (3,6) vào Q

	(4,6)	(6,6)	(6,7)	(5,8)	(4,5)	(3,6)	

Minh họa

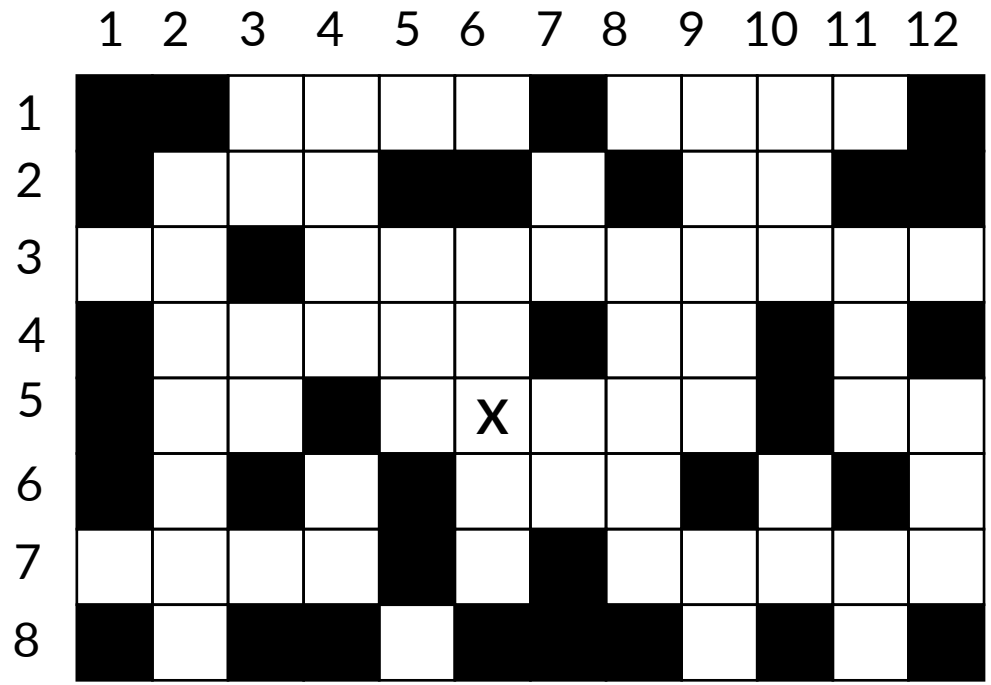


Lấy (6,6) ra khỏi Q

Đưa trạng thái (7,6) vào Q

	(6,6)	(6,7)	(5,8)	(4,5)	(3,6)	(7,6)	

Minh họa

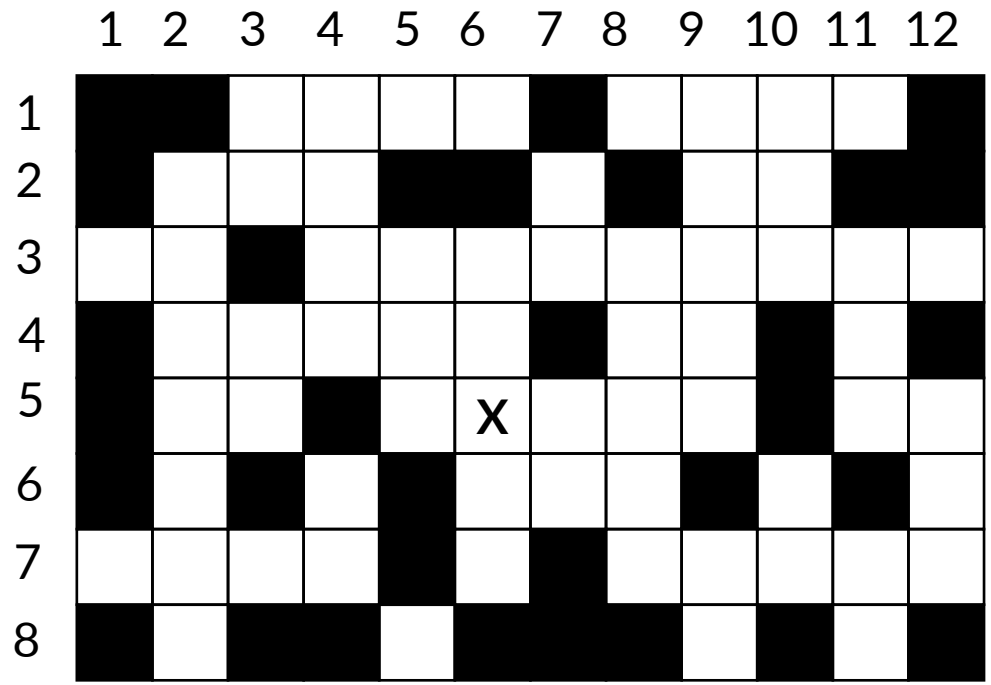


Lấy (6,7) ra khỏi Q

Đưa trạng thái (6,8) vào Q

	~~(6,7)~~	(5,8)	(4,5)	(3,6)	(7,6)	(6,8)	

Minh họa

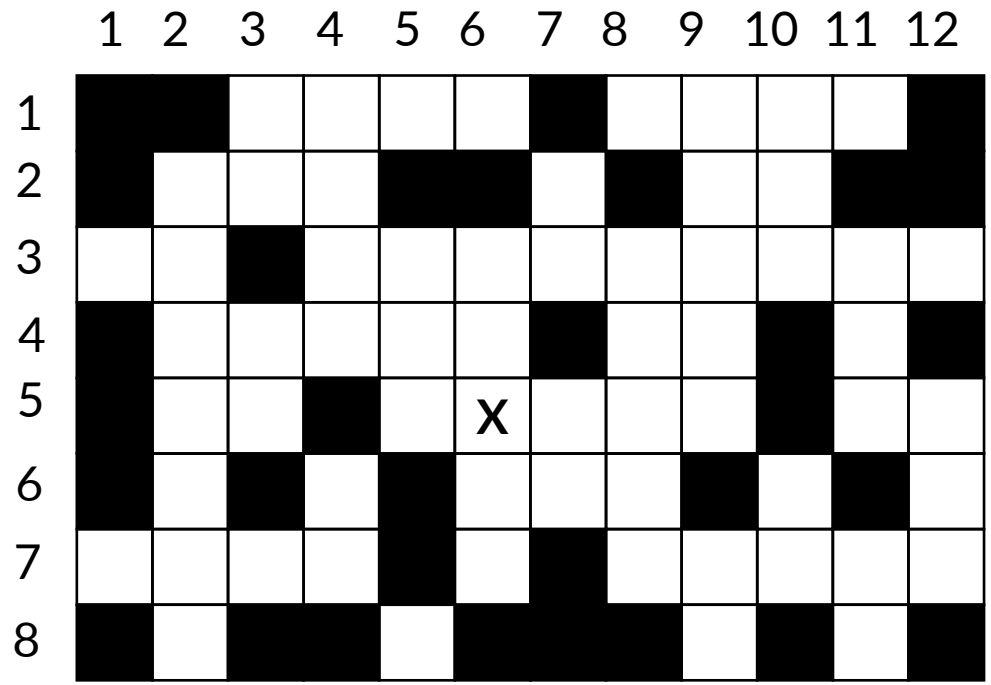


Lấy (5,8) ra khỏi Q

Đưa trạng thái (5,9) và(4,8) vào Q

	~~(5,8)~~	(4,5)	(3,6)	(7,6)	(6,8)	(5,9)	(4,8)	

Minh họa

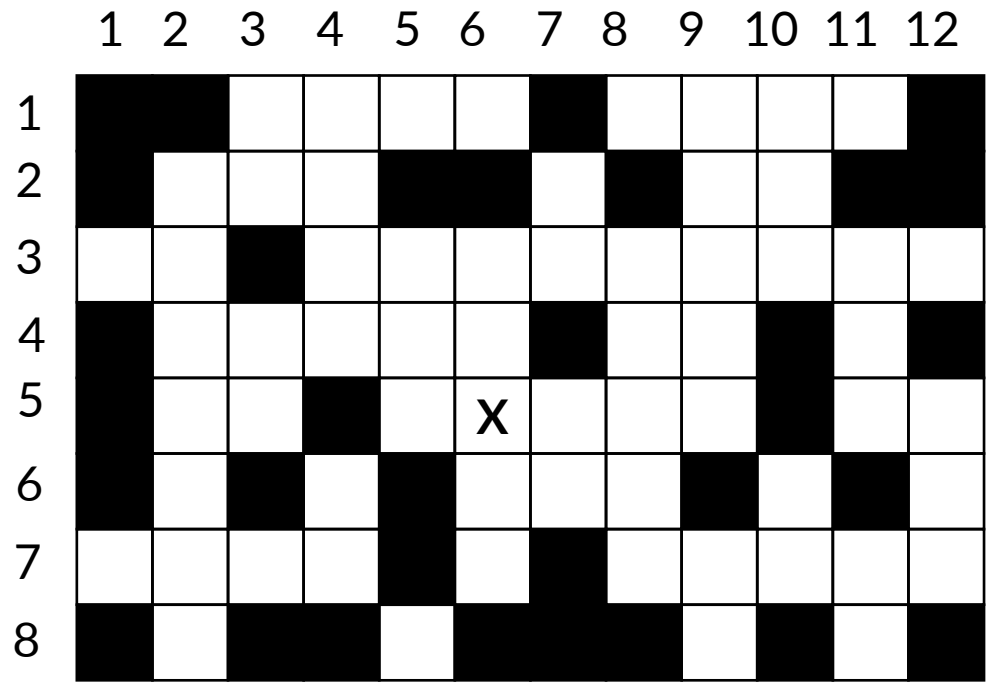


Lấy (4,5) ra khỏi Q

Đưa trạng thái (4,4) và(3,5) vào Q

	~~(4,5)~~	(3,6)	(7,6)	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)	

Minh họa

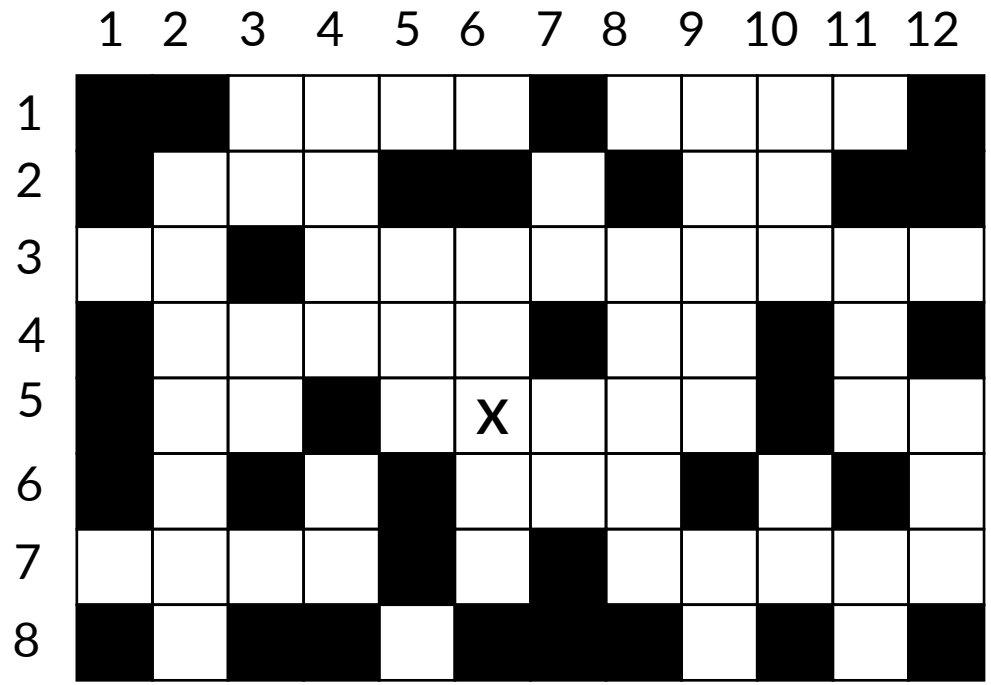


Lấy (3,6) ra khỏi Q

Đưa trạng thái (3,7) vào Q

	~~(3,6)~~	(7,6)	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)	(3,7)	

Minh họa

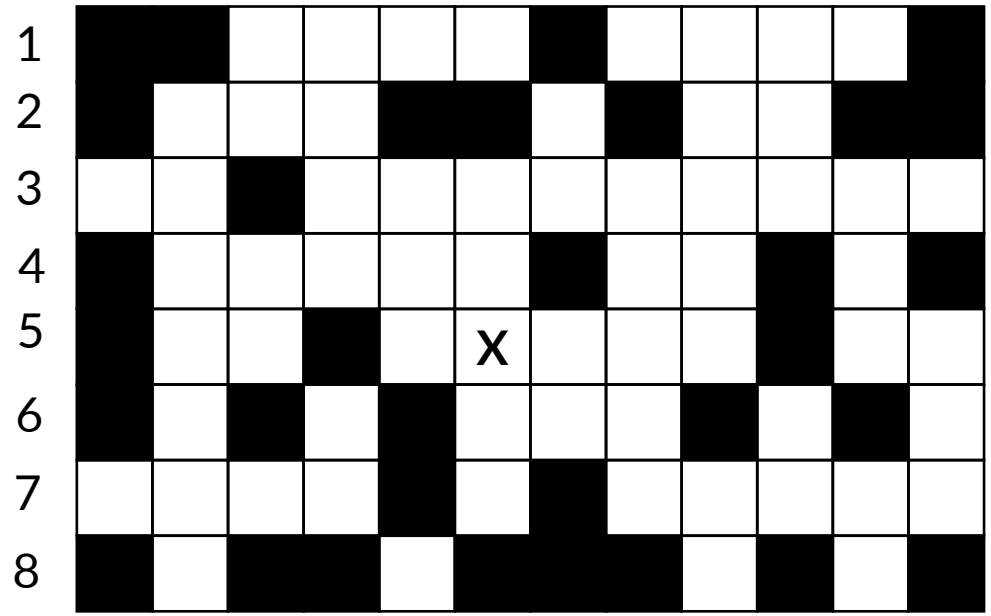


Lấy (7,6) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	(7,6)	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)	(3,7)	

Minh họa

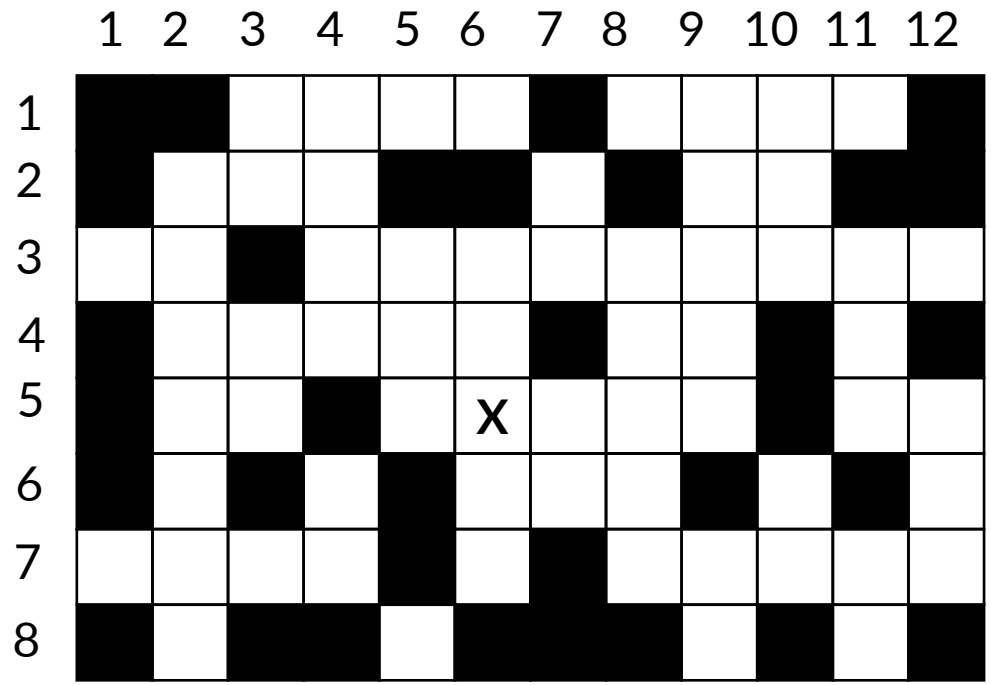


Lấy (6,8) ra khỏi Q

Đưa được trạng thái (7,8) vào Q

	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)	(3,7)	(7,8)	

Minh họa

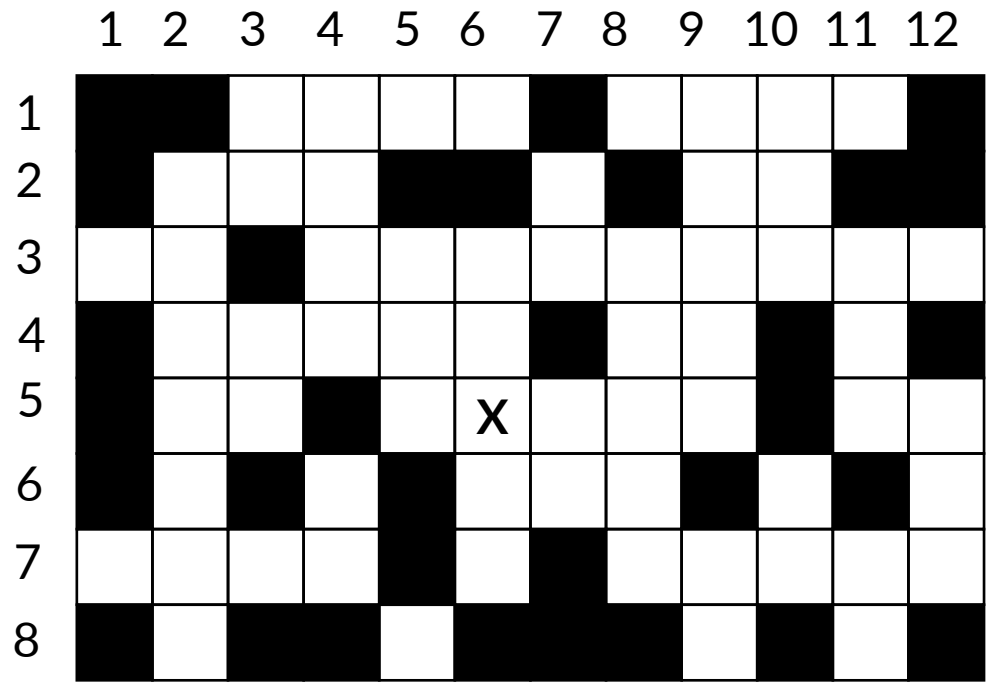


Lấy (5,9) ra khỏi Q

Đưa được trạng thái (4,9) vào Q

	~~(5,9)~~	(4,8)	(4,4)	(3,5)	(3,7)	(7,8)	(4,9)	

Minh họa

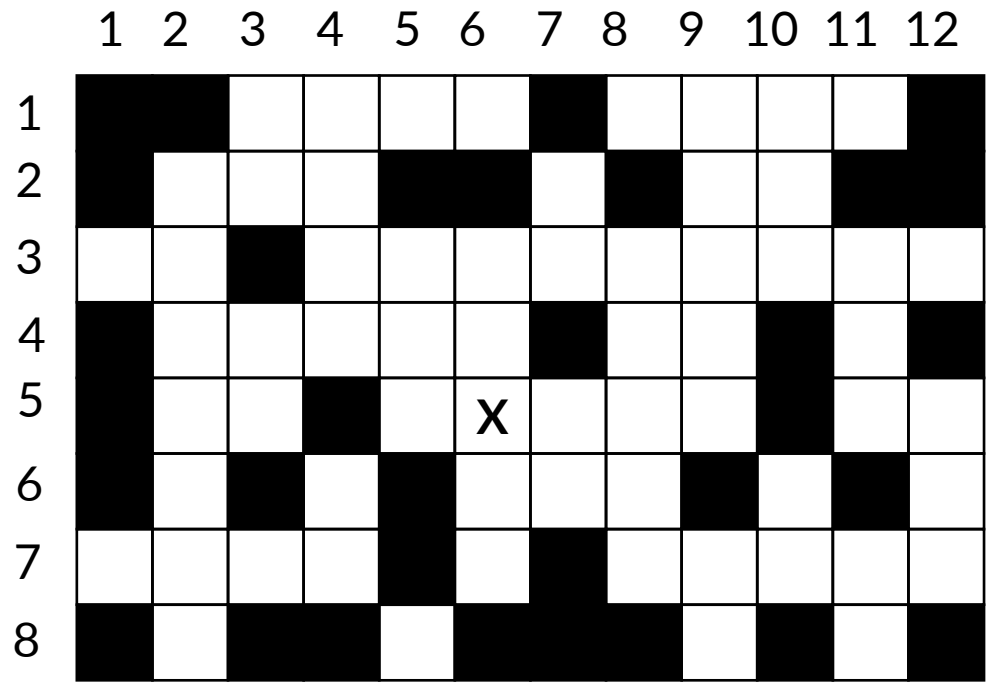


Lấy (4,8) ra khỏi Q

Đưa được trạng thái (3,8) vào Q

	(4,8)	(4,4)	(3,5)	(3,7)	(7,8)	(4,9)	(3,8)	

Minh họa

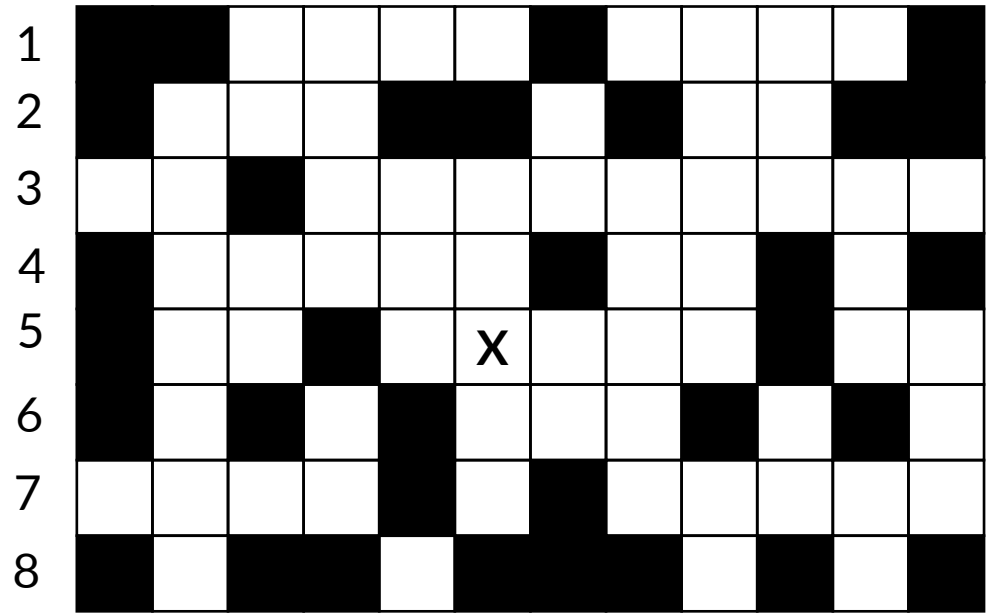


Lấy (4,4) ra khỏi Q

Đưa được trạng thái (4,3), (3,4) vào Q

	(4,4)	(3,5)	(3,7)	(7,8)	(4,9)	(3,8)	(4,3)	(3,4)	

Minh họa

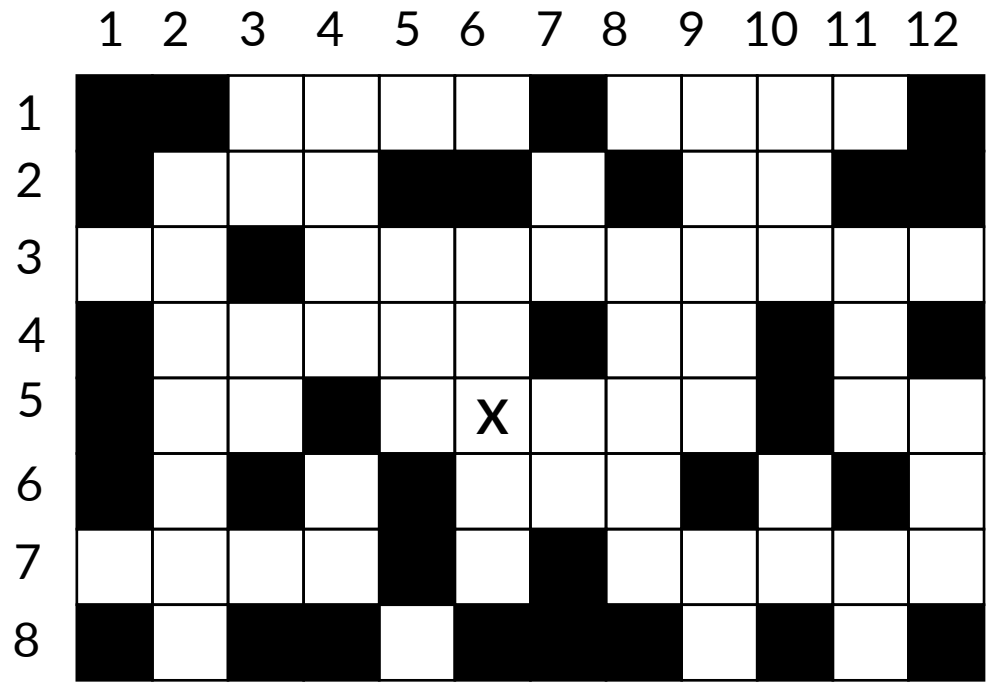


Lấy (3,5) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	~~(3,5)~~	(3,7)	(7,8)	(4,9)	(3,8)	(4,3)	(3,4)	

Minh họa

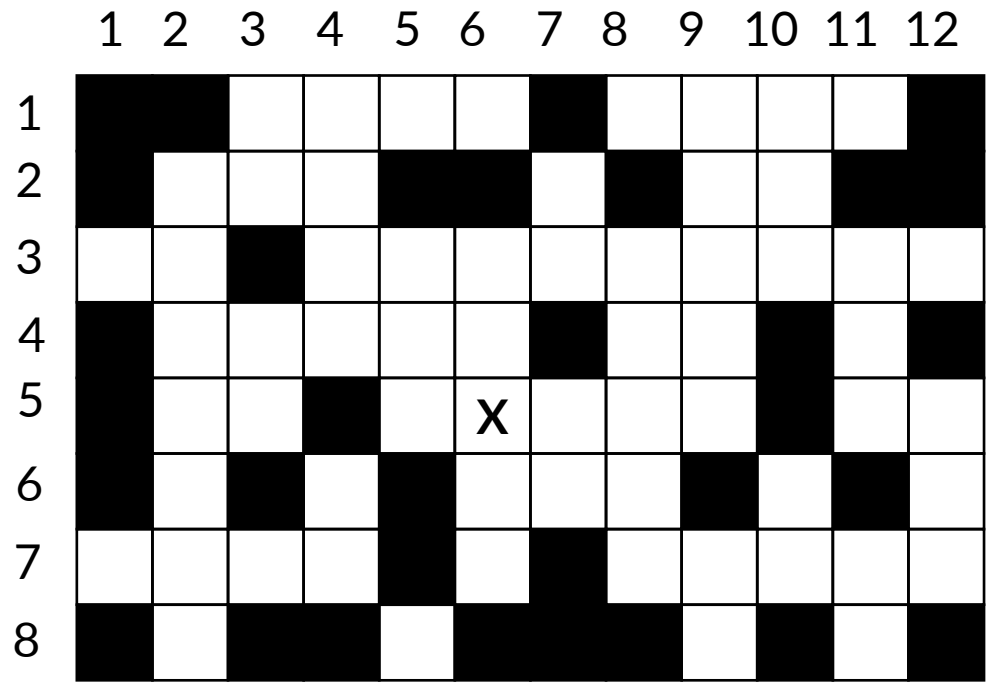


Lấy (3,7) ra khỏi Q

Đưa được trạng thái mới (2,7) vào Q

	~~(3,7)~~	(7,8)	(4,9)	(3,8)	(4,3)	(3,4)	(2,7)	

Minh họa

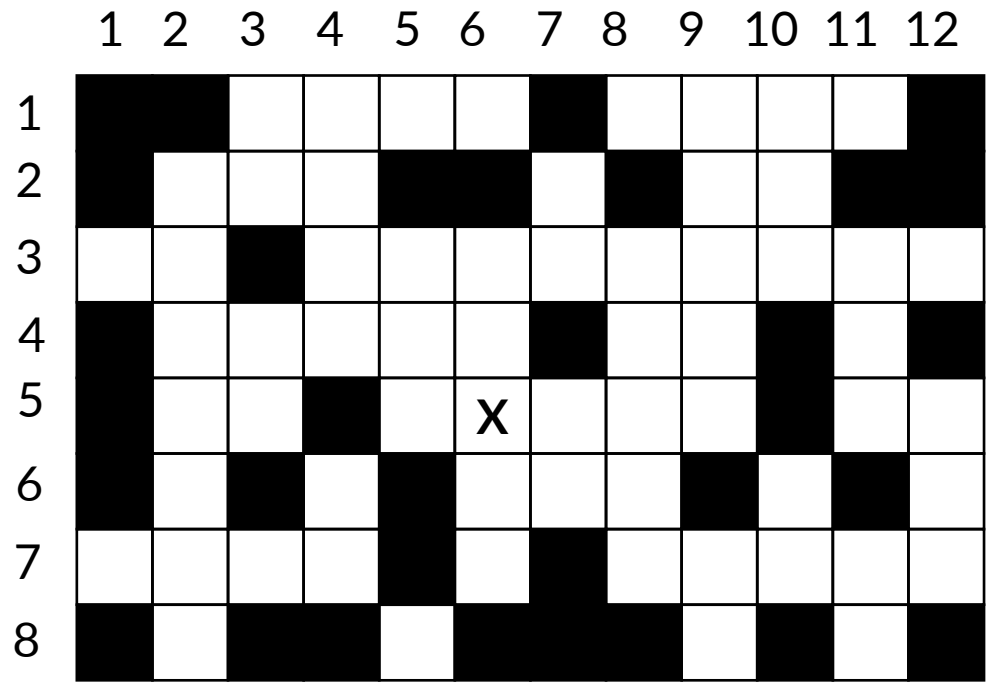


Lấy (7,8) ra khỏi Q

Đưa được trạng thái mới (7,9) vào Q

	~~(7,8)~~	(4,9)	(3,8)	(4,3)	(3,4)	(2,7)	(7,9)	

Minh họa

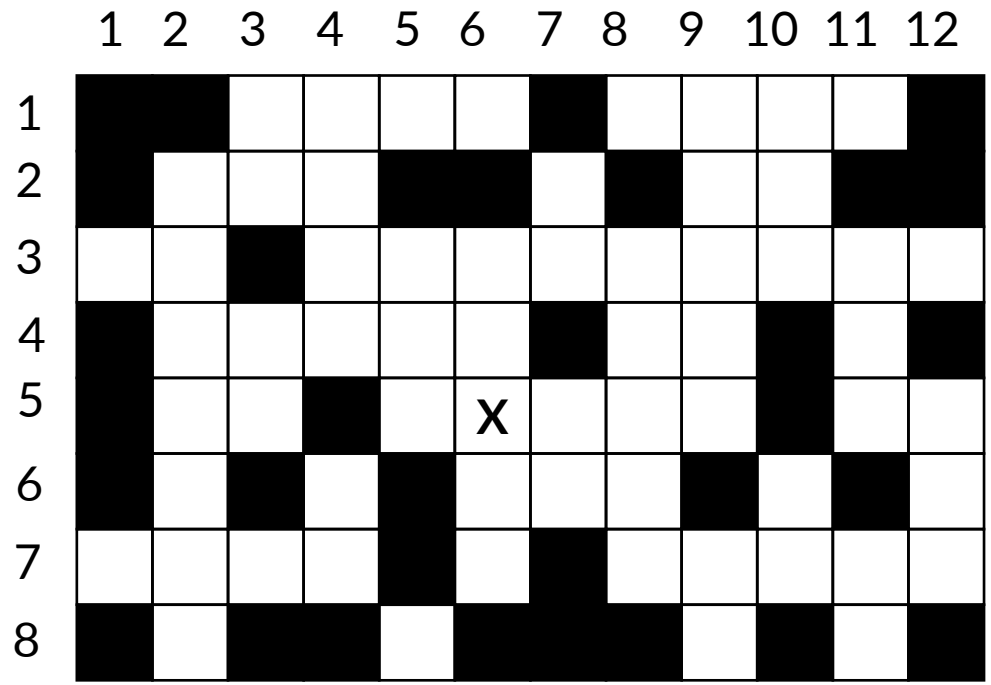


Lấy (4,9) ra khỏi Q

Đưa được trạng thái mới (3,9) vào Q

	(4,9)	(3,8)	(4,3)	(3,4)	(2,7)	(7,9)	(3,9)	

Minh họa



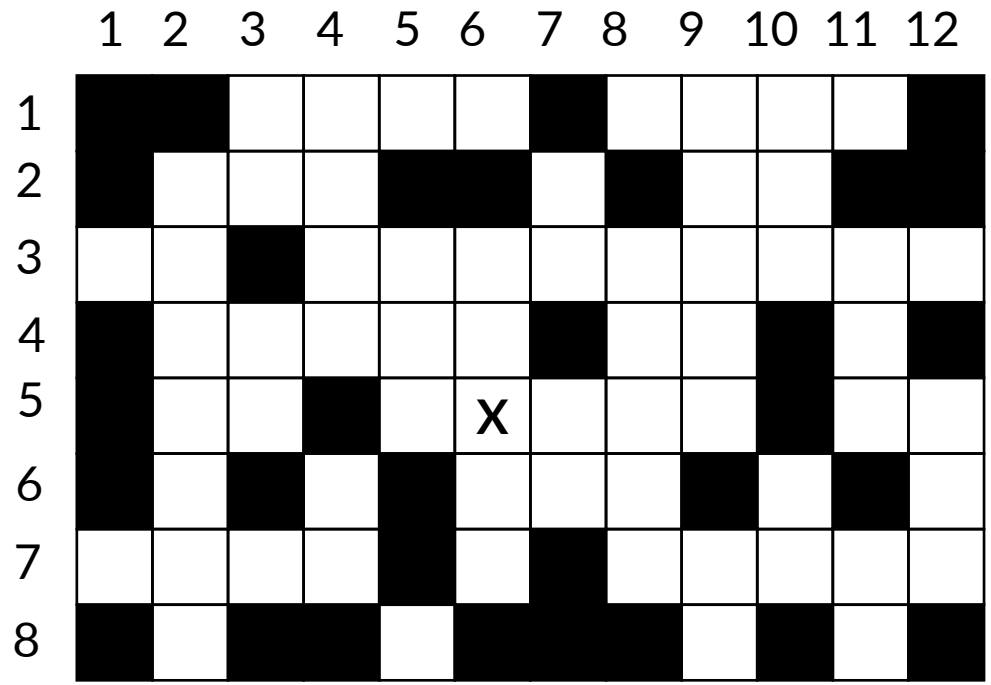
Lấy (3,8) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	~~(3,8)~~	(4,3)	(3,4)	(2,7)	(7,9)	(3,9)	

Minh họa

- **Minh họa**

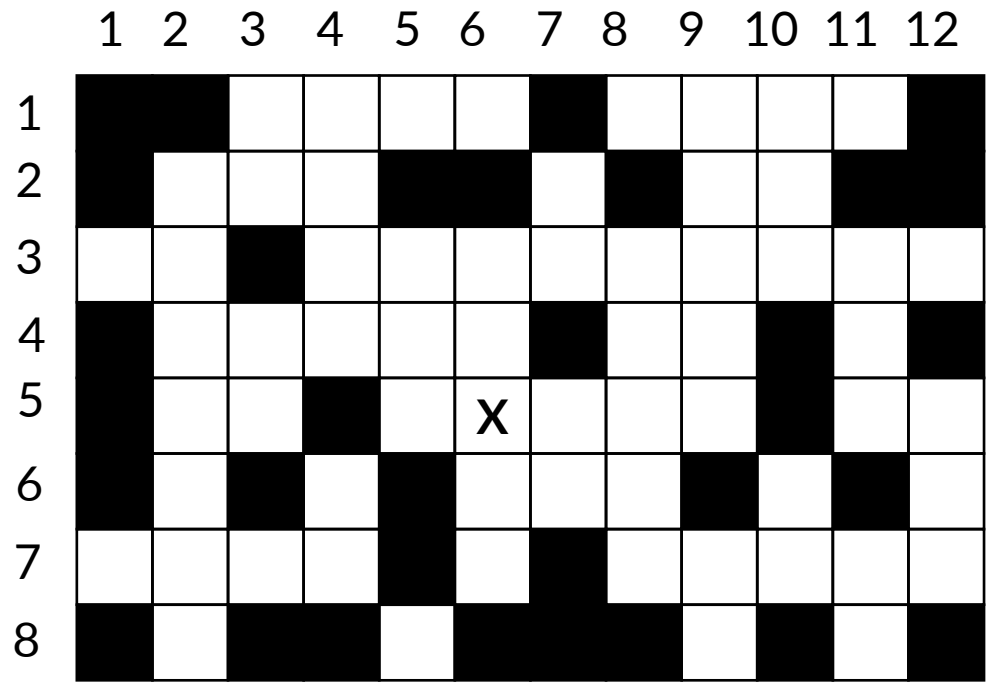


Lấy (4,3) ra khỏi Q

Đưa được trạng thái mới (4,2), (5,3) vào Q

	(4,3)	(3,4)	(2,7)	(7,9)	(3,9)	(4,2)	(5,3)	

Minh họa

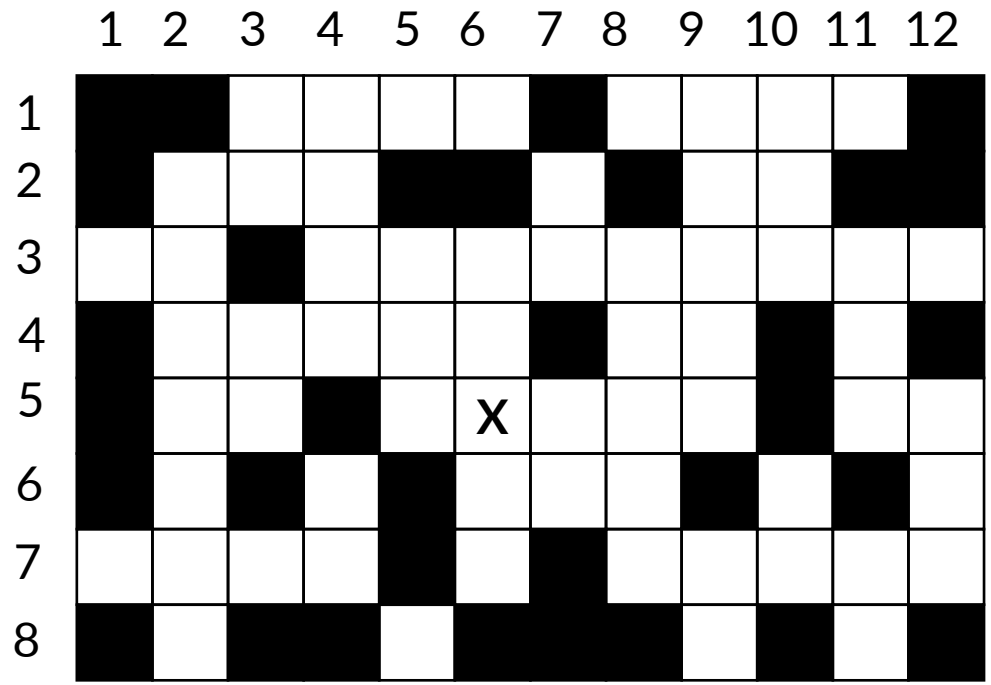


Lấy (3,4) ra khỏi Q

Đưa được trạng thái mới (2,4) vào Q

~~(3,4)~~	(2,7)	(7,9)	(3,9)	(4,2)	(5,3)	(2,4)

Minh họa

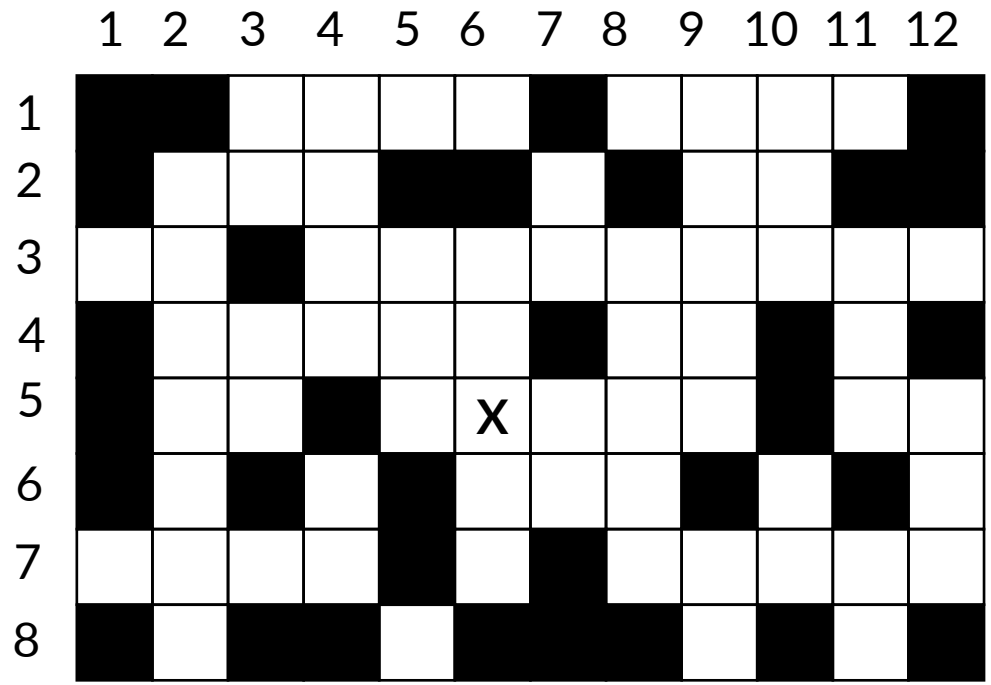


Lấy (2,7) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	~~(2,7)~~	(7,9)	(3,9)	(4,2)	(5,3)	(2,4)	

Minh họa

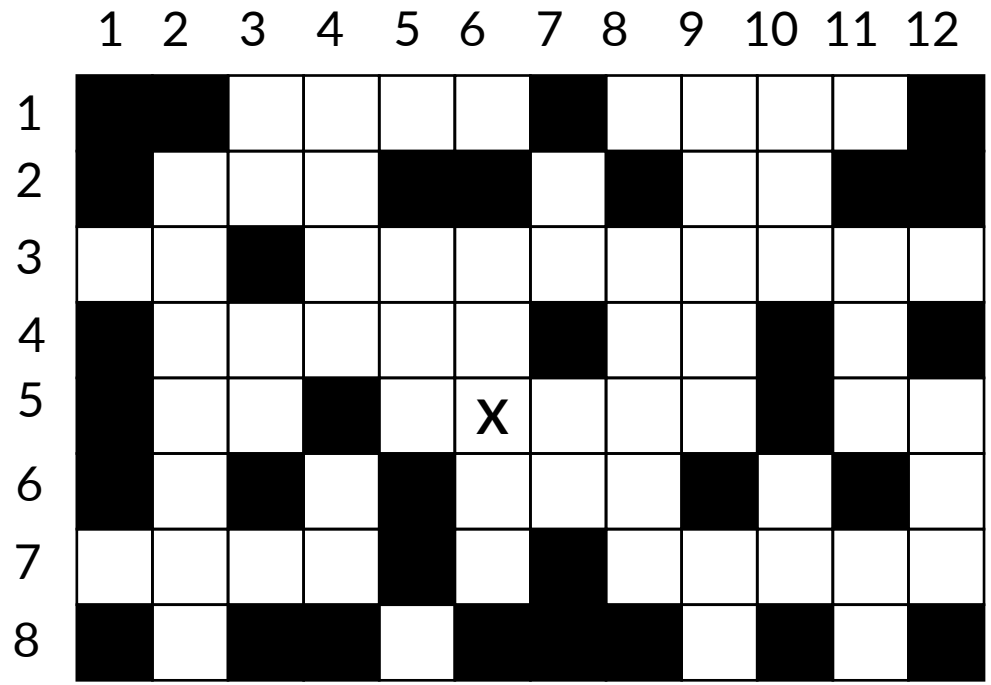


Lấy (7,9) ra khỏi Q

Đưa được trạng thái mới (8,9), (7,10) vào Q

	~~(7,9)~~	(3,9)	(4,2)	(5,3)	(2,4)	(8,9)	(7,10)	

Minh họa

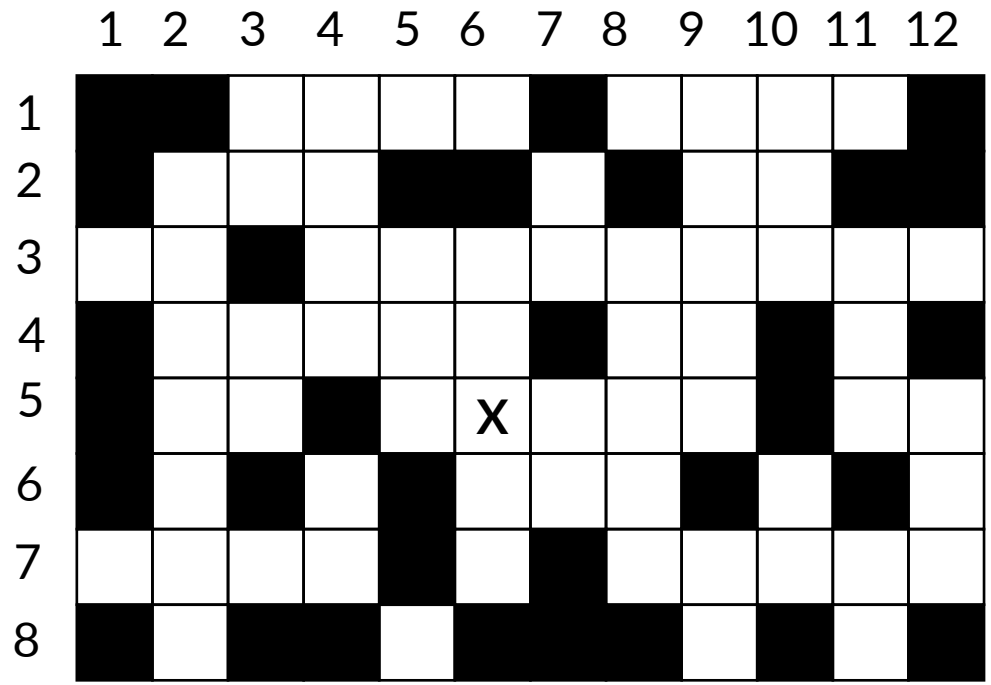


Lấy (3,9) ra khỏi Q

Đưa được trạng thái mới (2,9), (3,10) vào Q

	(3,9)	(4,2)	(5,3)	(2,4)	(8,9)	(7,10)	(2,9)	(3,10)	

Minh họa

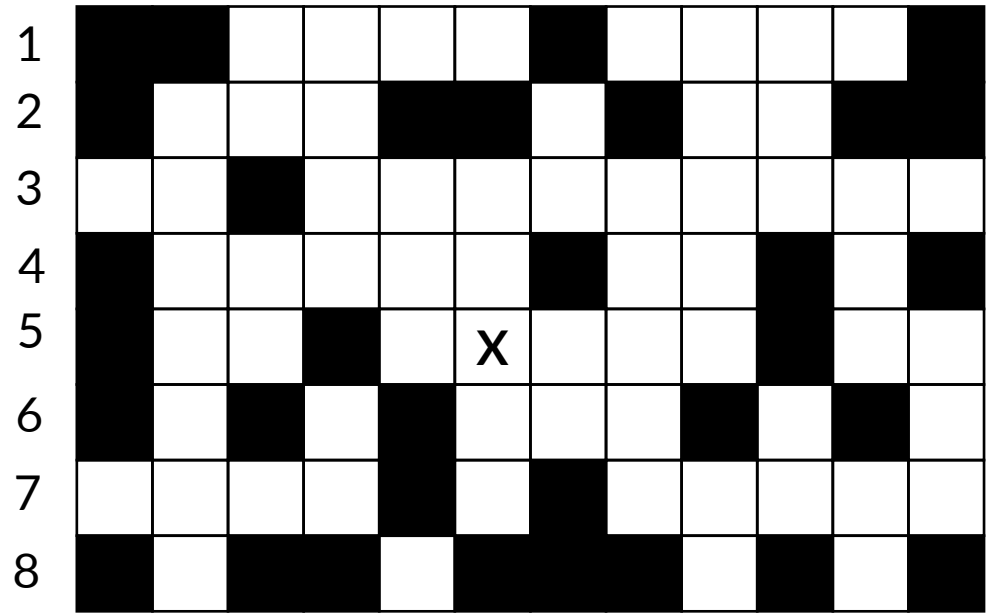


Lấy (4,2) ra khỏi Q

Đưa được trạng thái mới (3,2), (5,2) vào Q

	~~(4,2)~~	(5,3)	(2,4)	(8,9)	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	

Minh họa

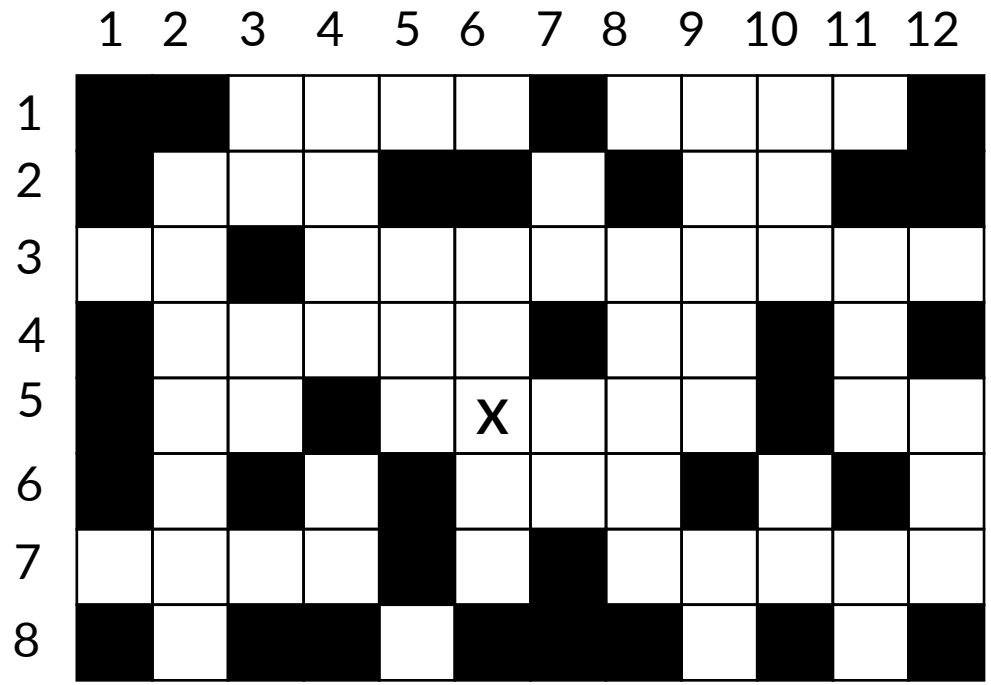


Lấy (5,3) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	~~(5,3)~~	(2,4)	(8,9)	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	

Minh họa



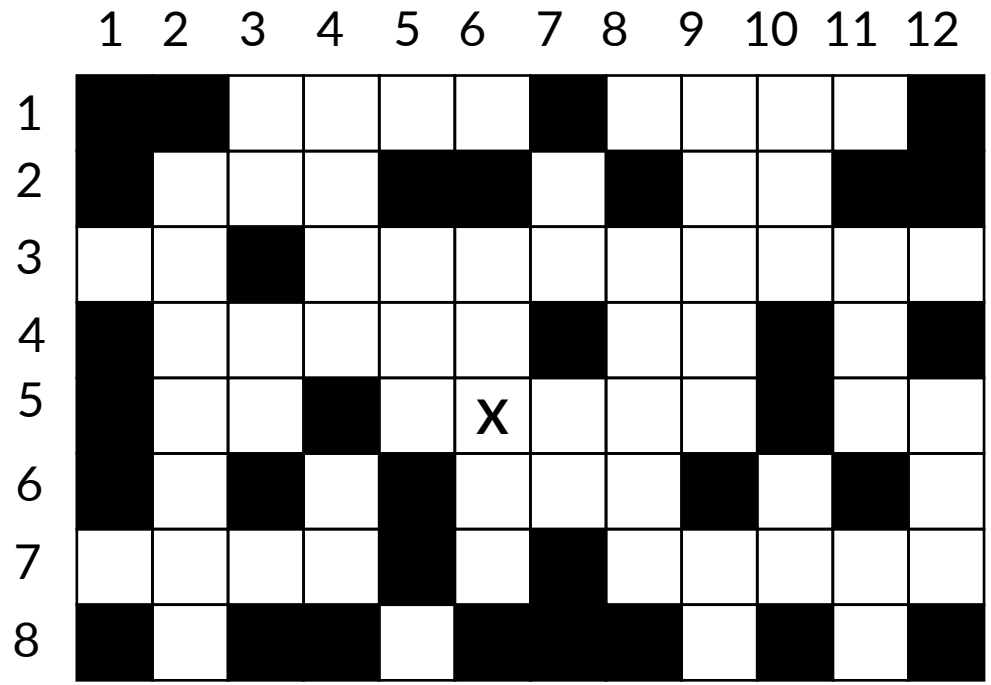
Lấy (2,4) ra khỏi Q

Đưa được trạng thái mới (1,4) vào Q

	~~(2,4)~~	(8,9)	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	(1,4)	

Minh họa

- **Minh họa**



Lấy (8,9) ra khỏi Q

Sinh ra được trạng thái đích (9,9) ứng với vị trí ngoài mê cung

	~~(8,9)~~	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	(1,4)	

Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
        int row;
        int col;
        int step;
        struct Node* next;
}Node;
```

Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
        int row;
        int col;
        int step;
        struct Node* next;
}Node;
```

```
int n,m;
int A[1000][1000];
int startRow, startCol;
int visited[1000][1000];
Node* first;
Node* last;
int dr[4] = {0,0,1,-1};
int dc[4] = {1,-1,0,0};
```

Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
        int row;
        int col;
        int step;
        struct Node* next;
}Node;
```

```
int n,m;
int A[1000][1000];
int startRow, startCol;
int visited[1000][1000];
Node* first;
Node* last;
int dr[4] = {0,0,1,-1};
int dc[4] = {1,-1,0,0};
```

```
Node* makeNode(int r,int c, int step){
  Node* p = (Node*)malloc(sizeof(Node));
  p->row = r; p->col = c;
  p->step = step; p->next = NULL;
  return p;
}
```

Cài đặt thuật toán

```
int isEmpty(){
  return first == NULL && last == NULL;
}
void push(Node* p){
  if(isEmpty(){
    first = p; last = p;  return;}
  last->next = p; last = p;
}
Node* pop(){
  if(isEmpty()) return NULL;
  Node* tmp = first; first = first->next;
  if(first == NULL) last = NULL;
  return tmp;
}
```

Cài đặt thuật toán

```
int isEmpty(){
  return first == NULL && last == NULL;
}
void push(Node* p){
  if(isEmpty(){
    first = p; last = p;  return;}
  last->next = p; last = p;
}
Node* pop(){
  if(isEmpty()) return NULL;
  Node* tmp = first; first = first->next;
  if(first == NULL) last = NULL;
  return tmp;
}
```

```
void input(){
  scanf("%d %d %d %d",&n,&m,&sr,&sc);
  for(int i = 1; i <= n; i++)
   for(int j = 1; j <= m; j++)
     scanf("%d",&A[i][j]);
}
```

Cài đặt thuật toán

```
int isEmpty(){
  return first == NULL && last == NULL;
}
void push(Node* p){
  if(isEmpty(){
    first = p; last = p;  return;}
  last->next = p; last = p;
}
Node* pop(){
  if(isEmpty()) return NULL;
  Node* tmp = first; first = first->next;
  if(first == NULL) last = NULL;
  return tmp;
}
```

```
void input(){
  scanf("%d %d %d %d",&n,&m,&sr,&sc);
  for(int i = 1; i <= n; i++)
   for(int j = 1; j <= m; j++)
     scanf("%d",&A[i][j]);
}
```

```
int targetState(Node* s){
  return (s->row < 1 ||
    s->row > n || s->col < 1
      || s->col > m);
}
```

Cài đặt thuật toán

```
void init(){
  first = NULL; last = NULL;
  for(int i = 0; i <= 1000; i++)
   for(int j = 0; j <= 1000; j++)
    visited[i][j] = 0;
  Node* startState = makeNode(sr,sc,0);
  push(startState); visited[sr][sc] = 1;
}
```

Cài đặt thuật toán

```
void init(){
  first = NULL; last = NULL;
  for(int i = 0; i <= 1000; i++)
   for(int j = 0; j <= 1000; j++)
    visited[i][j] = 0;
  Node* startState = makeNode(sr,sc,0);
  push(startState); visited[sr][sc] = 1;
}
```

```
int solve(){
 init();
 while(!isEmpty()){
  Node* s = pop();
  for(int k = 0; k < 4; k++){
  int nr = s->row + dr[k];int nc= s->col + dc[k];
    if(visited[nr][nc]==0&& A[nr][nc]==0){
      Node* ns = makeNode(nr, nc, s->step + 1);
      push(ns);visited[nr][nc] = 1;
      if(targetState(ns)) return ns->step;
    }
   }
  }
  return -1;
}
```

Cài đặt thuật toán

```
void init(){
  first = NULL; last = NULL;
  for(int i = 0; i <= 1000; i++)
   for(int j = 0; j <= 1000; j++)
    visited[i][j] = 0;
  Node* startState = makeNode(sr,sc,0);
  push(startState); visited[sr][sc] = 1;
}
```

```
int main(){
  input();
  int res = solve();
  printf("%d",res);
  return 0;
}
```

```
int solve(){
 init();
 while(!isEmpty()){
  Node* s = pop();
  for(int k = 0; k < 4; k++){
  int nr = s->row + dr[k];int nc= s->col + dc[k];
    if(visited[nr][nc]==0&& A[nr][nc]==0){
      Node* ns = makeNode(nr, nc, s->step + 1);
      push(ns);visited[nr][nc] = 1;
      if(targetState(ns)) return ns->step;
    }
   }
  }
  return -1;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !