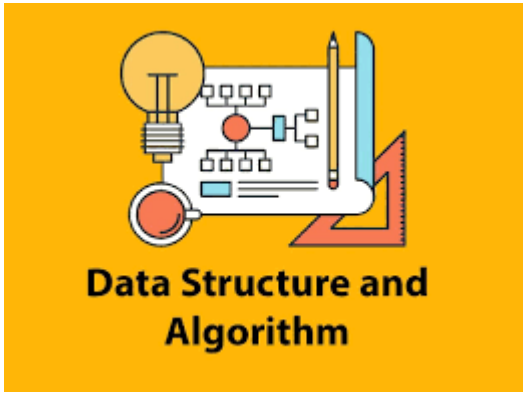
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

NGÀY 12: LÝ THUYẾT ĐỒ THỊ



Giảng viên: Th.S Bùi Văn Kiên

NỘI DUNG

- Giới thiệu về đồ thị
- Phương pháp biểu diễn đồ thị
- Thuật toán tìm kiếm theo chiều sâu DFS
- Thuật toán tìm kiếm theo chiều rộng BFS
- Áp dụng DFS và BFS

1. Giới thiệu về đồ thị

ĐỒ THỊ:

- G =<V, E>
- V là tập hợp hữu hạn được gọi là tập đỉnh (V = {1, 2,.., n})
- E là tập các cặp đỉnh trong V được gọi là tập cạnh.

- Đồ thị vô hướng: Các cạnh hai chiều
- Đơn đồ thị vô hướng:

Giữa hai đỉnh bất kỳ thuộc V có nhiều nhất một cạnh.

- Đa đồ thị vô hướng:

Tồn tại một cặp đỉnh trong V có nhiều hơn một cạnh nối.

- Giả đồ thị vô hướng. Có **khuyên**, tức là cạnh e =(u, u)

1. Giới thiệu về đồ thị

- Đơn đồ thị có hướng.
  - E là tập các cặp có thứ tự hai đỉnh thuộc V.
  - Có thể gọi là cạnh có hướng, hoặc cung

- Đa đồ thị có hướng. Có cạnh lặp (cung lặp) trên một cặp đỉnh.

1. Một số thuật ngữ trên đồ thị vô hướng

- **Đỉnh kề.** Hai đỉnh u và v của đồ thị vô hướng G = <V, E> được gọi là kề nhau nếu (u,v) là cạnh thuộc đồ thị G.

- **Bậc của đỉnh.** Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh xuất phát từ nó và ký hiệu là deg(v).

- **Đường đi từ u đến v:** dãy $x_0, x_1, \ldots, x_{n-1}, x_n$, trong đó n là số nguyên dương, $x_0 = u$, $x_n = v$, $(x_i, x_{i+1}) \in E$.

- **Chu trình:** Đường đi có đỉnh đầu trùng với đỉnh cuối.

1. Một số thuật ngữ trên đồ thị vô hướng

- **Tính liên thông.** Đồ thị vô hướng được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

- **Thành phần liên thông.** Đồ thị vô hướng liên thông thì số thành phần liên thông là 1. Đồ thị vô hướng không liên thông thì số liên thông của đồ thị là số các đồ thị con của nó liên thông.

- **Đỉnh trụ.** Đỉnh $u \in V$ được gọi là đỉnh trụ nếu loại bỏ u cùng với các cạnh nối với u làm tăng thành phần liên thông của đồ thị.

- **Cạnh cầu.** Cạnh $(u, v) \in E$ được gọi là cầu nếu loại bỏ (u, v) làm tăng thành phần liên thông của đồ thị.

1. Một số thuật ngữ trên đồ thị vô hướng

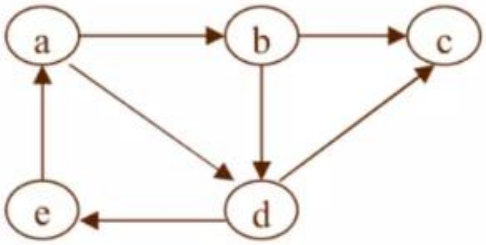
- **Cây:**
  - Đồ thị vô hướng có N đỉnh và đúng N-1 cạnh, không có chu trình.
  - Các đỉnh dưới cùng được gọi là một nút lá.

1. Một số thuật ngữ trên đồ thị có hướng

- **Bậc ra/vào của một đỉnh.**
- **Thành phần liên thông mạnh.** Một tập hợp đỉnh được gọi là liên thông mạnh nếu như luôn tìm được đường đi giữa 2 đỉnh bất kì.

$deg^+(a) = 2;\ deg^+(b) = 2;\ deg^+(c) = 0,$
$deg^+(d) = 2;\ deg^+(e) = 1.$

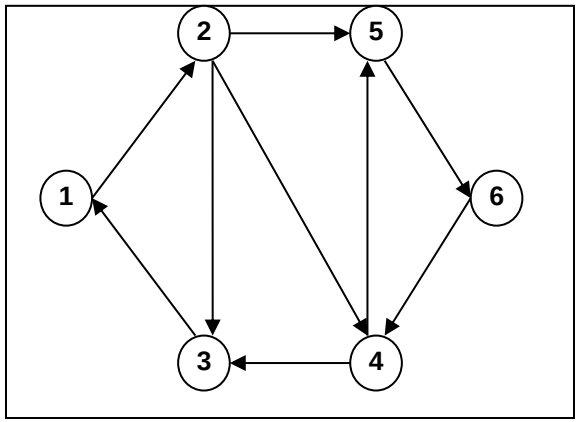
$deg^-(a) = 1;\ deg^-(b) = 1;\ deg^-(c) = 2;$
$deg^-(d) = 2,\ deg^-(e) = 1.$



2. Biểu diễn đồ thị

(1) Biểu diễn bằng ma trận kề

- Ưu điểm:
  - Đơn giản
  - Dễ dàng kiểm tra hai đỉnh kề nhau qua một phép so sánh.
- Nhược điểm
  - Tốn bộ nhớ
  - Không thể biểu diễn được với các đồ thị có số đỉnh lớn
  - Có thể có những phép so sánh không cần thiết khi đồ thị thưa.

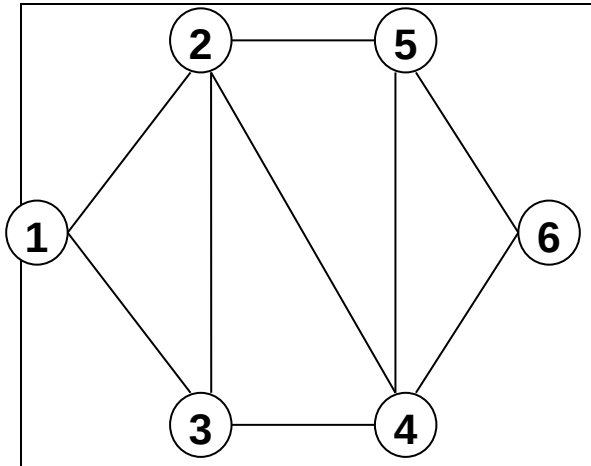


	i = 1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	1	1	1	0
3	1	0	0	0	0	0
4	0	0	1	0	1	0
5	0	0	0	0	0	1
6	0	0	0	1	0	0

2. Biểu diễn đồ thị

(2) Biểu diễn bằng danh sách cạnh

- Ưu điểm:
  - Trong trường hợp đồ thị thưa sẽ tiết kiệm được không gian nhớ;
  - Thuận lợi cho một số thuật toán chỉ quan tâm đến các cạnh của đồ thị.
- Nhược điểm
  - Khi cần duyệt các đỉnh kề với đỉnh u phải duyệt tất cả các cạnh của đồ thị nếu xử lý không khéo.

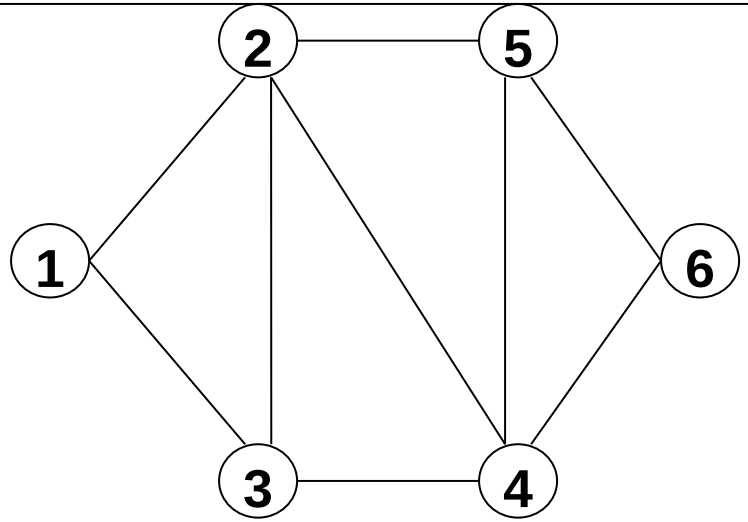


Đỉnh đầu	Đỉnh cuối
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6
5	6

2. Biểu diễn đồ thị

(3) Biểu diễn bằng danh sách kề: sử dụng vector/DSLK

- Ưu điểm:
  - Dễ dàng duyệt tất cả các đỉnh của một danh sách kề;
  - Thời gian duyệt đồ thị nhanh hơn.
- Nhược điểm
  - Khó cài đặt với các bài toán xử lý cạnh.



List(1) = {2, 3 }
List(2) = {1, 3, 4, 5 }
List(3) = {1, 2, 4 }
List(4) = {2, 3 , 5, 6}
List(5) = {2, 4, 6 }
List(6) = {4, 5 }

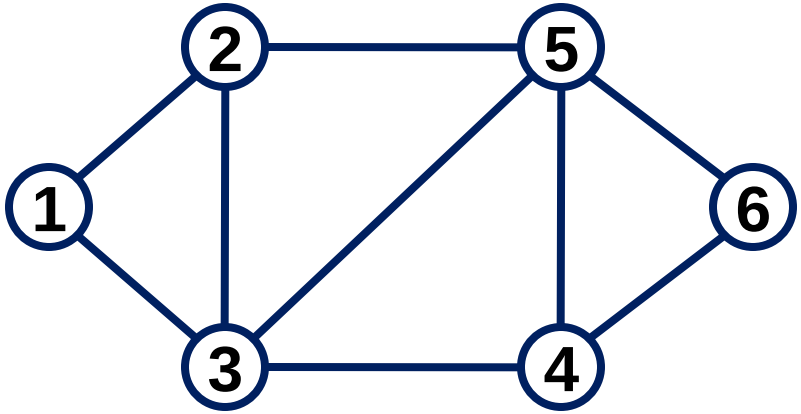
2. Biểu diễn đồ thị

Input từ đề bài: Danh sách cạnh

N = 6, M = 9

M dòng tiếp theo, mỗi dòng mô tả một cạnh của đồ thị.

6 9
1 2
1 3
2 3
2 5
3 4
3 5
4 5
4 6
5 6



Ma trận kề

6
0 1 1 0 0 0
1 0 1 0 1 0
1 1 0 0 1 0
0 1 1 0 1 1
0 1 1 1 0 1
0 0 0 1 1 0

Danh sách kề

List[1] = { 2 3 }
List[2] = { 1 3 5 }
List[3] = { 1 2 4 5 }
List[4] = { 3 5 6 }
List[5] = { 2 3 4 6 }
List[6] = { 4 5 }

2. Biểu diễn đồ thị

Đề bài: Cho danh sách cạnh với N đỉnh và M cạnh

- Chuyển input sang dạng ma trận kề

```
cin>>n>>m;
for(i=0; i<m; i++){
    cin>>u>>v;
    a[u][v] = 1;
    a[v][u] = 1;
}
```

- Chuyển input sang dạng danh sách kề, sử dụng vector

```
cin >> n >> m;
for(i=1; i<=m; i++){
    cin >> u >> v;
    List[u].push_back(v);
    List[v].push_back(u);
}
```

3. Tìm kiếm theo chiều sâu DFS

Cho G = <V, E>. Thuật toán tìm kiếm theo chiều sâu:

- Đánh dấu trạng thái các đỉnh
  - visited[u] = false ứng với trạng thái đỉnh u chưa được duyệt
  - visited[u] = true ứng với trạng thái đỉnh u đã được duyệt.
- Bắt đầu tại đỉnh tùy ý u $\in$ V ta thăm luôn đỉnh u.

Bật trạng thái visited[u] = true.

- Duyệt trên tập đỉnh Ke(u) = { v $\in$ V: (u, v) $\in$ E } và duyệt theo chiều sâu tại đỉnh v $\in$ Ke(u) đầu tiên chưa được duyệt.

3. Tìm kiếm theo chiều sâu DFS

- Thuật toán đệ quy DFS(u)

```
void DFS ( u) {
        visited[u] = true;
        for (v=1; v<=n; v++) {
                if ( v ∈ Ke(u) and visited[v] == false)
                        DFS(v);
        }
}
```

3. Tìm kiếm theo chiều sâu DFS

- DFS cài đặt bằng ma trận kề

```
void DFS(int u) {
    cout << u << endl;
    visited[u] = true;
    for(int v = 1; v <= n; v++) {
        if(a[u][v] == 1 && visited[v] == false) {
            visited[v] = true;
            DFS(v);
        }
    }
}
```

- DFS cài đặt bằng danh sách kề **DSA09004**

Danh sách kề

List[1] = { 2 3 }
List[2] = { 1 3 4 }
List[3] = { 1 2 4 5 }
List[4] = { 2 3 5 6 }
List[5] = { 3 4 6 }
List[6] = { 4 5 }

```
void DFS(int u) {
    cout << u << " ";
    visited[u] = true;
    for (int i = 0; i < List[u].size(); i++) {
        int v = List[u][i];
        if (visited[v] == false)
            DFS(v);
    }
}
```

4. Tìm kiếm theo chiều rộng BFS

```
Bước 1(Khởi tạo):
        Queue = ∅;                  // Khởi tạo hàng đợi rỗng
        Push(Queue, start);         // Đưa start vào hàng đợi
        visited[start] = true;      // Ghi nhận đỉnh đã xét
Bước 2 (Lặp):
    while (Queue not empty) do      //Lặp đến khi hàng đợi rỗng
        u = Pop(Queue);             //Lấy u ra khỏi hàng đợi
        <Thăm đỉnh u>;              //Thăm đỉnh u
        for each v ∈ Ke(u) do       //Lặp trên danh sách kề của u
            if (visited[v] == false) then //Nếu v chưa được duyệt
                Push(Queue, v);     //Đưa v vào hàng đợi
                visited[v] = true;  //Ghi nhận v đã xét
            EndIf;
        EndFor;
    EndWhile;
Bước 3 (Trả lại kết quả) : Return(<Tập đỉnh được duyệt>) ;
```

5. Ứng dụng DFS/BFS

Độ phức tạp thuật toán DFS(u), BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị:

- $O(n^2)$ trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với n là số đỉnh
- O(n.m) trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với n là số đỉnh của đồ thị, m là số cạnh.
- O(max(n, m)) trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.

5. Ứng dụng DFS/BFS

- **Duyệt tất cả các đỉnh của đồ thị.**
- **Xác định các thành phần liên thông của đồ thị.**
- **Tìm đường đi từ s đến t.**
- **Tính đường đi ngắn nhất từ s đến t.**
- Duyệt các đỉnh trụ của đồ thị.
- Duyệt các cạnh cầu của đồ thị.
- Xây dựng cây khung của đồ thị.
- Xác định tính liên thông mạnh.

5.1. Duyệt tất cả các đỉnh của đồ thị

- Display-Graph:

```
begin
 * Bước 1 (Khởi tạo):
        for (u=1; u<=n; u++) do {          //thiết lập tất cả các đỉnh đều chưa duyệt
              visited[u] = false;
        Endfor;
  * Bước 2 (Lặp):
        for (u =1; u <= n; u++)  do {      //lặp trên tập đỉnh V
              if (visited[u] == false) then
                          BFS (u);              //DFS(u);
              endif;
         endfor;
end.
```

5.2. Tìm số thành phần liên thông

Count connected component

* Bước 1 (Khởi tạo):

```
counter = 0;                    //thiết lập số thành phần liên thông ban đầu là 0
for (u=1; u<=n; u++) do { //thiết lập tất cả các đỉnh đều chưa duyệt
        visited[u] = false;
Endfor;
```

* Bước 2 (Lặp):

```
for (u =1; u <= n; u++)  do {       //lặp trên tập đỉnh V
        if (visited[u] == false) then
                counter ++;         //tăng thành phần liên thông
                <ghi nhận các đỉnh cùng một thành phần liên thông>;
                BFS (u);            //DFS(u);
        endif;
endfor;
```

Bước 3 (trả lại kết quả): Return counter;

5.2. Tìm số thành phần liên thông

DSA09011

SỐ LƯỢNG HÒN ĐẢO

Bài làm tốt nhất

Cho một bản đồ kích thước N x M được mô tả bằng ma trận A[][].A[i][j] = 1 có nghĩa vị trí (i, j) là nổi trên biển. 2 vị trí (i, j) và (x, y) được coi là liền nhau nếu như nó có chung đỉnh hoặc chung cạnh. Một hòn đảo là một tập hợp các điểm (i, j) mà A[i][j] = 1 và có thể di chuyển giữa hai điểm bất kì trong đó.

Nhiệm vụ của bạn là hãy đếm số lượng đảo xuất hiện trên bản đồ.

Input: Dòng đầu tiên là số lượng bộ test T (T ≤ 20).

Mỗi test bắt đầu bởi 2 số nguyên N và M (1 ≤ N, M ≤ 500).

N dòng tiếp theo, mỗi dòng gồm M số nguyên A[i][j].

Output: Với mỗi test, in ra số lượng hòn đảo tìm được.

5.3. Tìm đường đi từ s → t

Dùng DFS

* Bước 1 (Khởi tạo):

```
for (u=1; u<=n; u++) {        //thiết lập tất cả các đỉnh đều chưa duyệt
        visited[u] = false;
        parent[u] = -1;
}
```

* Bước 2 (Gọi đệ quy):

```
void DFS (u) {
        visited[u] = true;
        for (v=1; v <=n; v++) {
                if (a[u][v] == 1 and visited[v] == false) {
                        DFS(v);
                        parent[v] = u;
                }
        }
}
```

Bước 3 (trả lại kết quả): Truy vết đường đi dựa trên mảng parent[];

5.3. Tìm đường đi từ s → t

Dùng BFS

Dừng lại khi đến đích. Thêm bước truy vết từ mảng parent[]

5.4. Tính đường đi ngắn nhất từ s → t

Danh sách cạnh

N = 6, M = 9, ST = 1, DES = 6

6 9 1 6

1 2

1 3

2 3

2 5

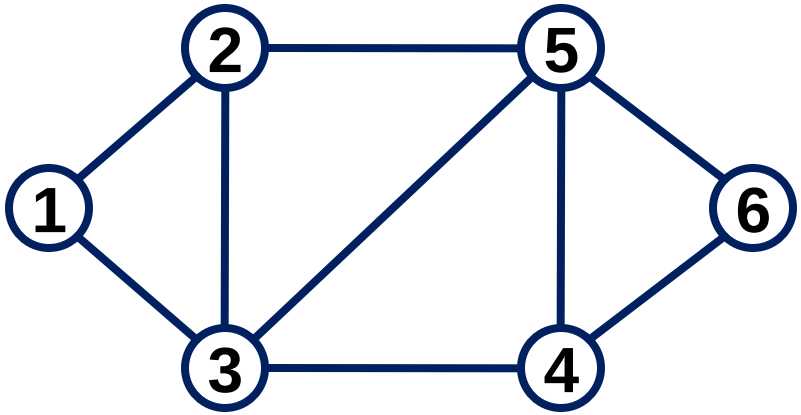
3 4

3 5

4 5

4 6

5 6



Output:

1→2→5→6

Nếu sử dụng BFS, đường đi tìm được luôn là đường đi ngắn nhất giữa s và t

5.4. Tính đường đi ngắn nhất từ s → t

Danh sách cạnh

N = 6, M = 9, ST = 1, DES = 6

6 9 1 6

1 2

1 3

2 3

2 5

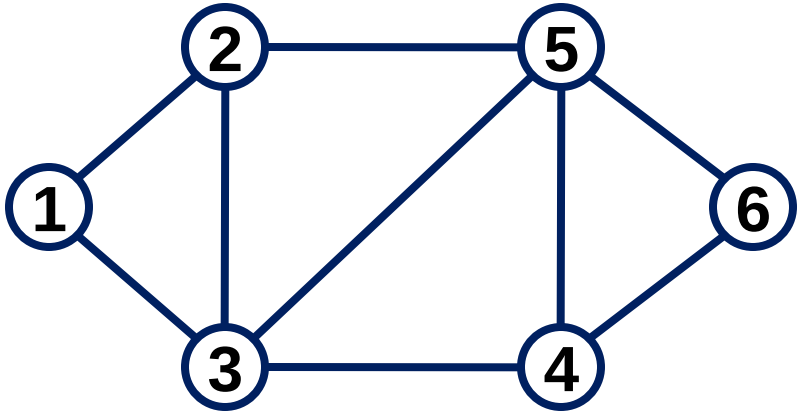
3 4

3 5

4 5

4 6

5 6



Output:

1→2→5→6

Cách tính: thêm mảng dist[] vào hàm BFS

Quy hoạch động: dist[v] = dist[u]+1

5.5 Ứng dụng BFS

- **Kỹ thuật "loang" BFS**
  - Bước 1: Đưa trạng thái xuất phát vào hàng đợi
  - Bước 2: Lặp đến khi hàng đợi rỗng hoặc gặp trạng thái đích
    - Xét trạng thái S ở đầu hàng đợi
    - Loại bỏ S ra khỏi hàng
    - Đưa các trạng thái đến được từ S vào hàng đợi

  (chú ý: nếu trạng thái đã từng có trong hàng đợi trước đó thì không thể đưa vào lần 2)
  - Bước 3: Kết luận kết quả
- Kỹ thuật BFS sẽ cho số bước chuyển trạng thái từ xuất phát đến đích là ít nhất.
- Vấn đề: đánh dấu trạng thái đã đi qua?

5.5 Ứng dụng BFS

Có 3 trường hợp:

- Trường hợp 1: Các trạng thái hoàn toàn phân biệt => Không cần đánh dấu
  - Các bài toán BFS nhị phân
- Trường hợp 2: Các trạng thái có thể đánh dấu bằng mảng (một chiều hoặc hai chiều).
  - Ví dụ: BFS trên đồ thị
- Trường hợp 3: Các trạng thái phức tạp hoặc cần tìm kiếm nhanh => Sử dụng mã hóa + đánh dấu bằng set hoặc map.

5.5 Ví dụ

Bài toán di chuyển trên bảng:

- Cho bảng số kích thước N x M, có các ô ST, EN, ., *
- Mỗi bước bạn có thể di chuyển trái, trên, phải, dưới nếu ô đó là dấu .
- * biểu diễn một ô bị block, không đi được
- Hãy tìm đường đi ngắn nhất từ ST → EN, in ra độ dài đường đi

(1,1) → (1, 2) → (1, 3) → (2, 3) → (3, 3) → (3, 4) → (3, 5) → (4, 5)

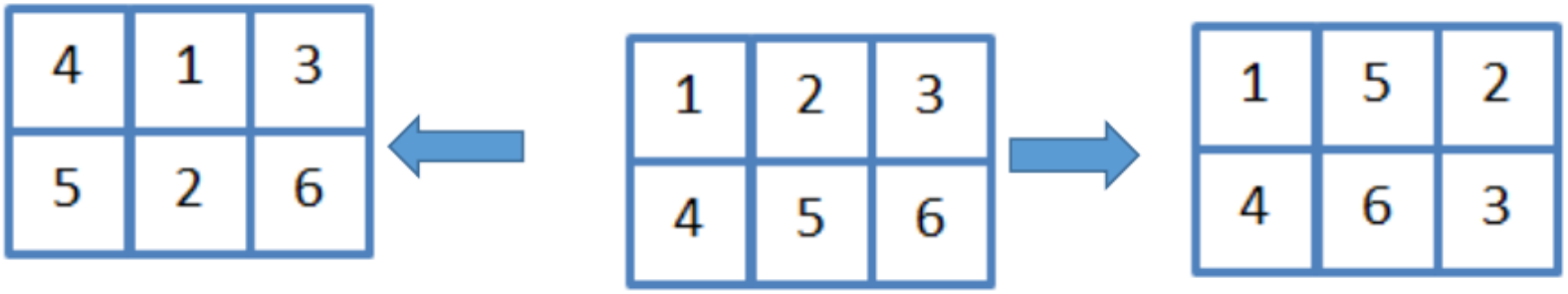
Input:
N M
ST.x ST.y EN.x EN.y
N xâu mô tả bảng

4 5
1 1 4 5
...*.
.*...
.....
...*.

5.6 Ví dụ

Bài toán quay hình vuông:

- Cho bảng gồm 6 miếng ghép ô vuông, có 2 phép xoay trái & phải



- In ra số bước ít nhất để di chuyển từ trạng thái ST → EN

Trạng thái xuất phát ST
Trạng thái kết thúc EN
1 2 3 4 5 6
4 1 2 6 5 3

Output = 2

https://code.ptit.edu.vn/student/question/DSA08029

QUESTIONS & ANSWERS