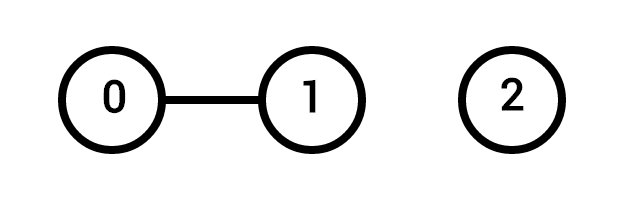
Cho một đồ thị vô hướng và chỉ số của một đỉnh bất kì

Hãy tính kích thước của [thành phần liên thông](https://vi.wikipedia.org/wiki/Th%C3%A0nh_ph%E1%BA%A7n_li%C3%AAn_th%C3%B4ng) có chứa đỉnh đó

**Ví dụ:**

* Với matrix = [[false, true, false],  
            [true, false, false],  
            [false, false, false]] và vertex = 0, thì kết quả dfsComponentSize(matrix,vertex) = 2.  
    
  Thành phần liên thông có chứa đỉnh 0 bao gồm 2 đỉnh là đỉnh 0 và 1 như hình dưới đây

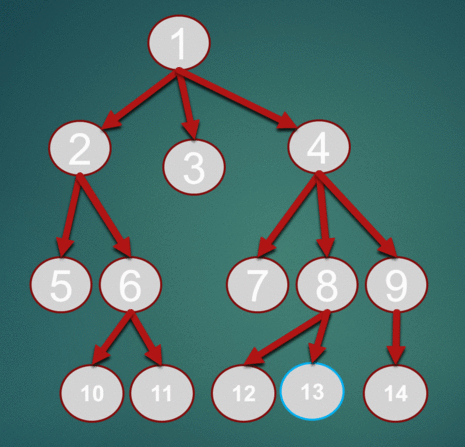


**Đầu vào/Đầu ra**

* **[Thời gian chạy] 0.5 giây**
* **[Đầu vào] array.array.boolean matrix**[Ma trận kề](https://vi.wikipedia.org/wiki/Ma_tr%E1%BA%ADn_k%E1%BB%81) của đồ thị vô hướng*Điều kiện:*3 ≤ matrix.length ≤ 10,  
  matrix[i].length = matrix.length.
* **[Đầu vào] integer vertex**  
  *Điều kiện:*  
  0 ≤ vertex < matrix.length.
* **[Đầu ra] integer**

**Lý thuyết :**

* Cách duyệt trên đồ thị :
  + với đồ thị có trọng số mọi cạnh bằng nhau (có thể coi là 1), ta dùng được BFS (Breadth First Search) và DFS (Depth First Search)
  + với đồ thị có trọng số không âm : Dijkstra, Floyd, Ford Bellman ... (bạn có thể tự tìm hiểu trên mạng hoặc làm đến phần advanced :3
* Cách hoạt động của BFS : Ta sẽ từ 1 đỉnh, đánh dấu những đỉnh kề, thăm mọi đỉnh ở tầng này rồi đi đến tầng tiếp theo, cứ thế cho đến khi thăm được cả thành phần liên thông.



* Cách triển khai : bạn cần 1 cấu trúc dữ liệu gọi là hàng đợi (ví dụ : queue in C++), và 1 mảng visit để xác định đỉnh đã thăm chưa   
  + Đầu tiên, ta bỏ vào queue đỉnh xuất phát
  + Với đỉnh u đầu tiên trong queue, ta thăm mọi đỉnh kề v, nếu v chưa thăm thì ghi lại giá trị visit[v], và bỏ v vào queue. Làm thế đến khi không còn đỉnh nào ở trong queue
  + Nếu chỉ cần biết là v đã thăm hay chưa, thì ta ghi lại visit[v] = 1. Còn nếu cần khoảng cách từ đỉnh xuất phát đến v, thì ta gán visit[v] = visit[u] + 1
* Tính đúng đắn :
  + Ta sẽ chứng minh theo đệ quy. Giả sử tầng thứ 1 đến i - 1 ta đã thăm hết. Xét đỉnh u ở tầng i, ta chỉ thăm được các đỉnh ở tầng kề trên hoặc kề dưới với u, tức tầng i - 1 hoặc i + 1. Mà tầng kề trên đã thăm hết, nên các đỉnh v chưa thăm kề u sẽ chỉ thuộc tầng i + 1. Vậy nên, sau khi duyệt hết các đỉnh thuộc tầng i, trong queue chỉ còn các đỉnh tầng i + 1. Cứ đệ quy thế, ta sẽ thấy : các đỉnh được ném vào queue lần lượt theo từng tầng, đúng như điều ta muốn ở BFS
  + 1 đỉnh u được ném vào queue tối đa 1 lần (vì sau đó visit[u] đã được sửa đổi, không đủ điều kiện để ném vào lần nữa). Vậy nên số đỉnh tối đa trong queue là n
  + Mỗi đỉnh u ta thăm mọi đỉnh kề v của nó, mà u chỉ xuất hiện tối đa 1 lần, nên ta thăm tối đa m cạnh
  + Vậy nên, *ĐPT : O(n + m)*
  + Code minh họa :

void BFS(vector<vector<bool>> matrix, int n, int StartVertex) {

int visit[n] = { 0 };

queue <int> Q;

visit[StartVertex] = 1, Q.push(StartVertex);

// bo dinh xuat phat vao, coi nhu tang 0

while (Q.size()) {

int u = Q.front(); Q.pop();

for (int v = 0; v < n; ++v)

if (matrix[u][v] && visit[v] == 0) // co canh (u, v), v chua tham

visit[v] = 1, Q.push(v);// danh dau v da tham, bo v vao queue

}

}