Chương 8. ĐỒ THỊ

Lập trình thi đấu

Đỗ Phan Thuận

Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

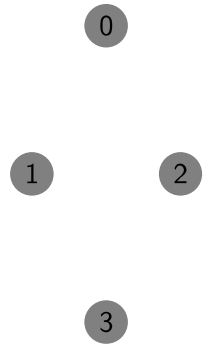
Ngày 5 tháng 1 năm 2016

Nội dung

Đồ thị là gì?

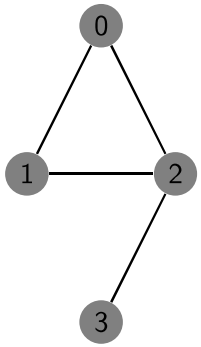
Đồ thị là gì?

- Đỉnh
  - Giao của các con đường
  - Máy tính
  - Các sàn trong nhà
  - Các đối tượng



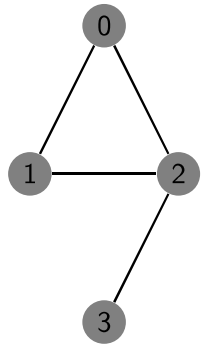
Đồ thị là gì?

- Đỉnh
  - Giao của các con đường
  - Máy tính
  - Các sàn trong nhà
  - Các đối tượng
- Cạnh
  - Con đường
  - Dây mạng
  - Thang bộ và thang máy
  - Mối quan hệ giữa các đối tượng



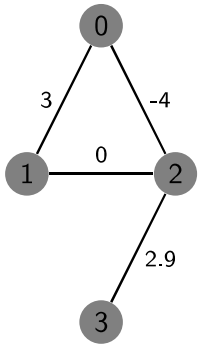
Các loại cạnh

- Không có trọng số



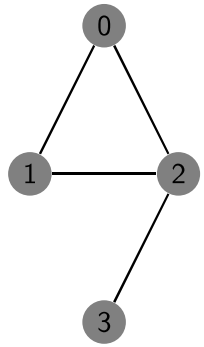
Các loại cạnh

- Không có trọng số hoặc Có trọng số



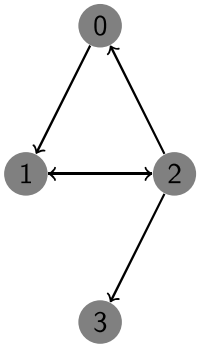
Các loại cạnh

- Không có trọng số hoặc Có trọng số
- Vô hướng



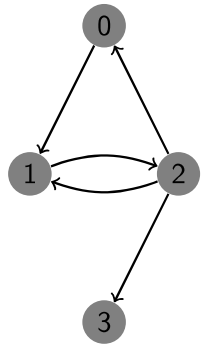
Các loại cạnh

- Không có trọng số hoặc Có trọng số
- Vô hướng or Có hướng

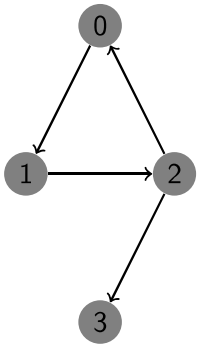


Các loại cạnh

- Không có trọng số hoặc Có trọng số
- Vô hướng or Có hướng

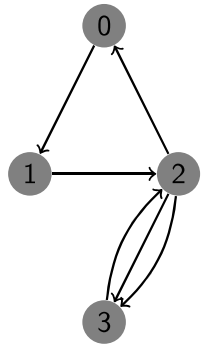


Đa đồ thị



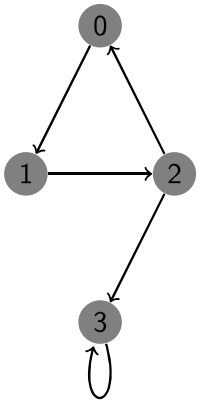
Đa đồ thị

- Cạnh lặp



Đa đồ thị

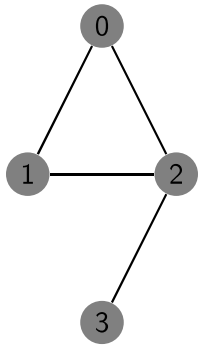
- Cạnh lặp
- Khuyên



Danh sách kề

```
0: 1, 2
1: 0, 2
2: 0, 1, 3
3: 2
```

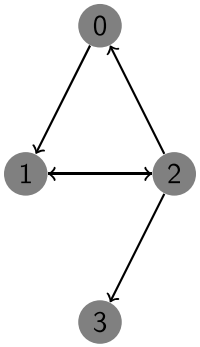
```
vector<int> adj[4];
adj[0].push_back(1);
adj[0].push_back(2);
adj[1].push_back(0);
adj[1].push_back(2);
adj[2].push_back(0);
adj[2].push_back(1);
adj[2].push_back(3);
adj[3].push_back(2);
```



Danh sách kề (có hướng)

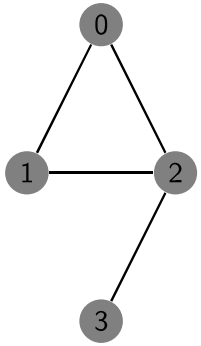
```
0: 1
1: 2
2: 0, 1, 3
3:
```

```
vector<int> adj[4];
adj[0].push_back(1);
adj[1].push_back(2);
adj[2].push_back(0);
adj[2].push_back(1);
adj[2].push_back(3);
```



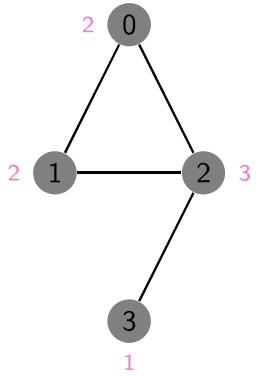
Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh:
  - Số lượng cạnh kề
  - Số lượng đỉnh kề



Một số tính chất trên đỉnh (đồ thị vô hướng)

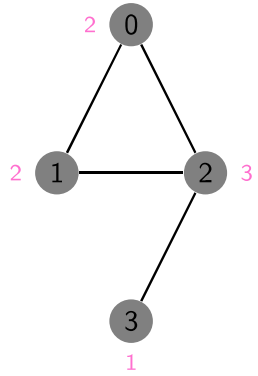
- Bậc của một đỉnh:
  - Số lượng cạnh kề
  - Số lượng đỉnh kề



Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh:
  - Số lượng cạnh kề
  - Số lượng đỉnh kề
- Bổ đề về những cái bắt tay (Handsaking)

$$\sum_{v \in V} \deg(v) = 2|V|$$

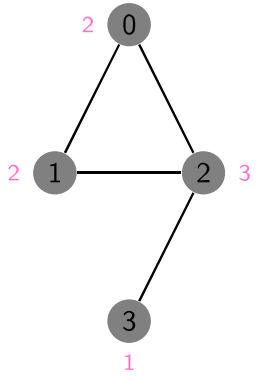


Một số tính chất trên đỉnh (đồ thị vô hướng)

- Bậc của một đỉnh:
  - Số lượng cạnh kề
  - Số lượng đỉnh kề
- Bổ đề về những cái bắt tay (Handsaking)

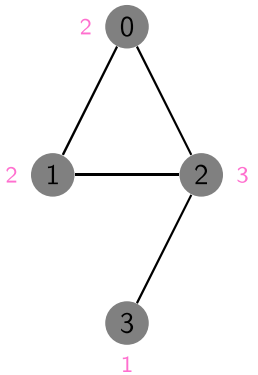
$$\sum_{v \in V} \deg(v) = 2|V|$$

$$2 + 2 + 3 + 1 = 2 \times 4$$



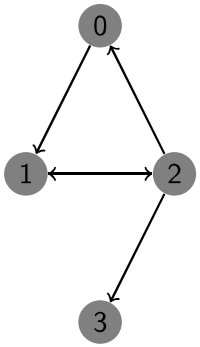
Một số tính chất trên đỉnh (đồ thị vô hướng)

```
0: 1, 2
1: 0, 2
2: 0, 1, 3
3: 2

adj[0].size() // 2
adj[1].size() // 2
adj[2].size() // 3
adj[3].size() // 1
```



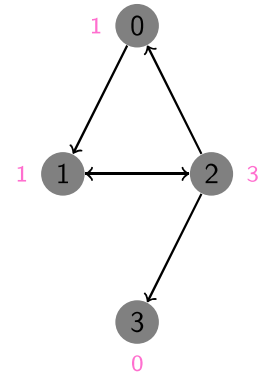
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh



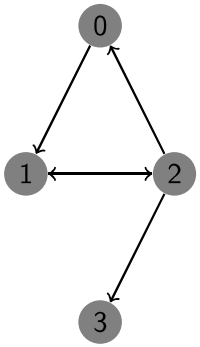
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh



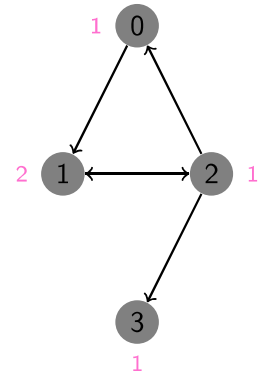
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
  - Số lượng cạnh đi vào đỉnh



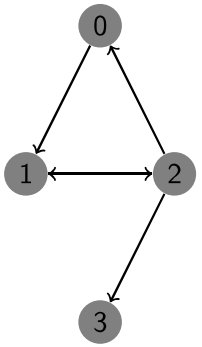
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
  - Số lượng cạnh đi vào đỉnh



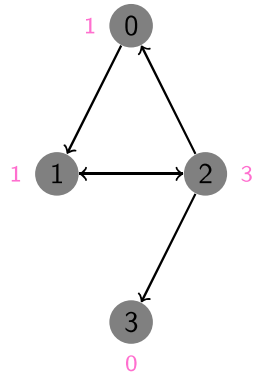
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
  - Số lượng cạnh đi vào đỉnh



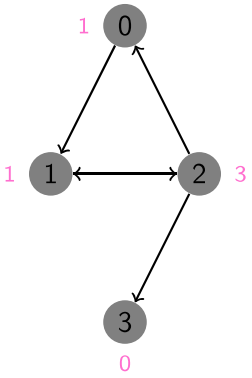
Một số tính chất trên đỉnh (đồ thị có hướng)

- Bán bậc ra của một đỉnh
  - Số lượng cạnh đi ra khỏi đỉnh
- Bán bậc vào của một đỉnh
  - Số lượng cạnh đi vào đỉnh



Danh sách kề (có hướng)

```
0: 1
1: 2
2: 0, 1, 3
3:

adj[0].size() // 1
adj[1].size() // 1
adj[2].size() // 3
adj[3].size() // 0
```



Đường đi

- Đường đi (Path / Walk / Trail):

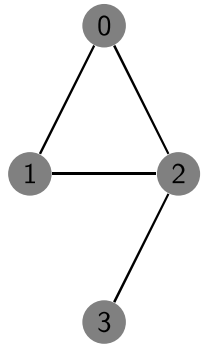
$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$

không lặp cạnh: $e_i = e_j \Rightarrow i = j$

$$\text{to}(e_i) = \text{from}(e_{i+1})$$



Đường đi

- Đường đi (Path / Walk / Trail):

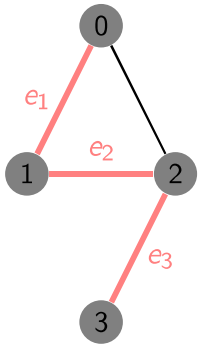
$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$

không lặp cạnh: $e_i = e_j \Rightarrow i = j$

$$\text{to}(e_i) = \text{from}(e_{i+1})$$



Đường đi

- Đường đi (Path / Walk / Trail):

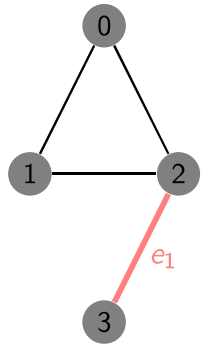
$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$

không lặp cạnh: $e_i = e_j \Rightarrow i = j$

$$\text{to}(e_i) = \text{from}(e_{i+1})$$



Đường đi

- Đường đi (Path / Walk / Trail):

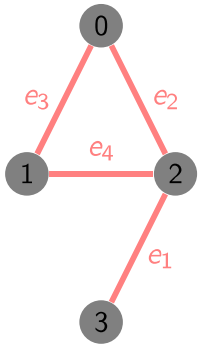
$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$

không lặp cạnh: $e_i = e_j \Rightarrow i = j$

$$\text{to}(e_i) = \text{from}(e_{i+1})$$



Chu trình

- Chu trình (Cycle / Circuit / Tour):

$$e_1 e_2 \dots e_k$$

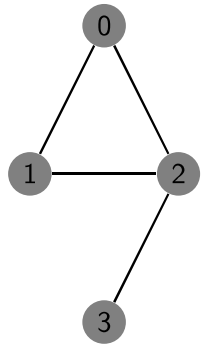
sao cho

$$e_i \in E$$

$$e_i = e_j \Rightarrow i = j$$

$$\text{to}(e_i) = \text{from}(e_{i+1})$$

$$\text{from}(e_1) = \text{to}(e_k)$$



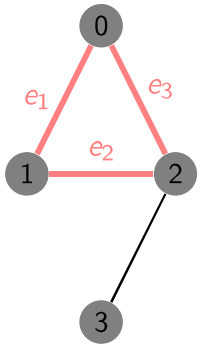
Chu trình

- Chu trình (Cycle / Circuit / Tour):

$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$
$$e_i = e_j \Rightarrow i = j$$
$$\text{to}(e_i) = \text{from}(e_{i+1})$$
$$\text{from}(e_1) = \text{to}(e_k)$$



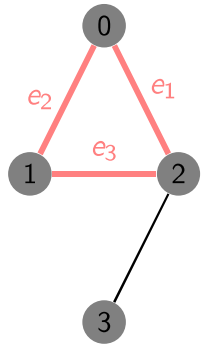
Chu trình

- Chu trình (Cycle / Circuit / Tour):

$$e_1 e_2 \dots e_k$$

sao cho

$$e_i \in E$$
$$e_i = e_j \Rightarrow i = j$$
$$\text{to}(e_i) = \text{from}(e_{i+1})$$
$$\text{from}(e_1) = \text{to}(e_k)$$



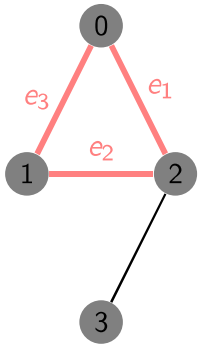
Chu trình

- Chu trình (Cycle / Circuit / Tour):

$$e_1 e_2 \dots e_k$$

sao cho

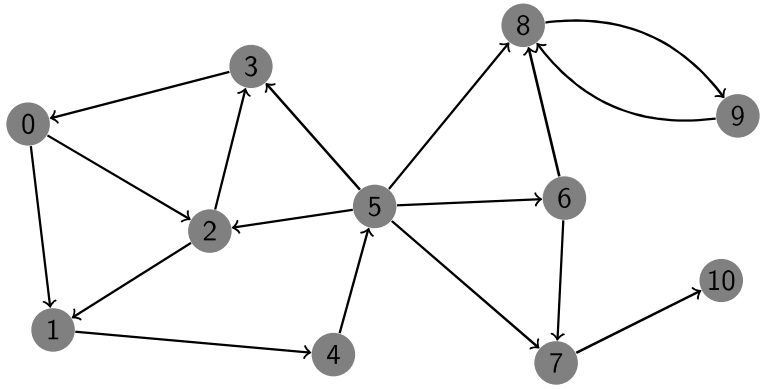
$$e_i \in E$$
$$e_i = e_j \Rightarrow i = j$$
$$\text{to}(e_i) = \text{from}(e_{i+1})$$
$$\text{from}(e_1) = \text{to}(e_k)$$



Tìm kiếm theo chiều sâu - DFS

- Cho đồ thị $G = (V, E)$ (có hướng hoặc vô hướng) và hai đỉnh u và v, hỏi có tồn tại một đường đi từ u đến v?
- Thuật toán tìm kiếm theo chiều sâu đưa ra được dduongf đi như vậy nếu tồn tại
- Thuật toán duyệt đồ thị ưu tiên theo chiều sâu, bắt đầu từ đỉnh xuất phát u
- Thực chất ta không cần chỉ rõ đỉnh v, vì ta có thể để thuật toán thăm tất cả các đỉnh có thể đến được từ u (mà vẫn cùng độ phức tạp)
- Vậy độ phức tạp của thuật toán là bao nhiêu?
- Mỗi đỉnh được thăm đúng một lần, và mỗi cạnh cũng được duyệt qua đúng một lần (nếu đến được)
- $O(n + m)$

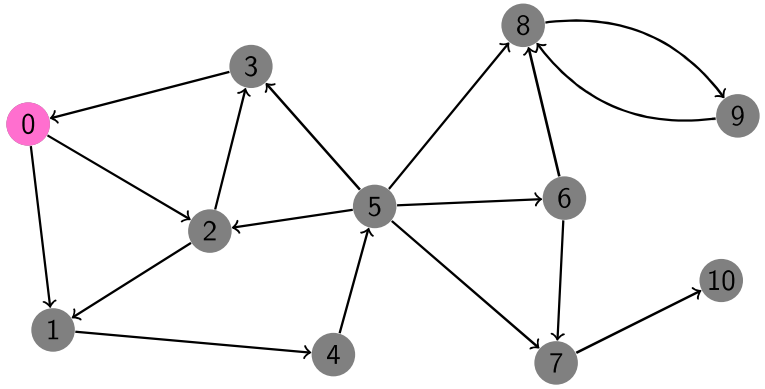
Tìm kiếm theo chiều sâu



Stack: |

	0	1	2	3	4	5	6	7	8	9	10
marked	0	0	0	0	0	0	0	0	0	0	0

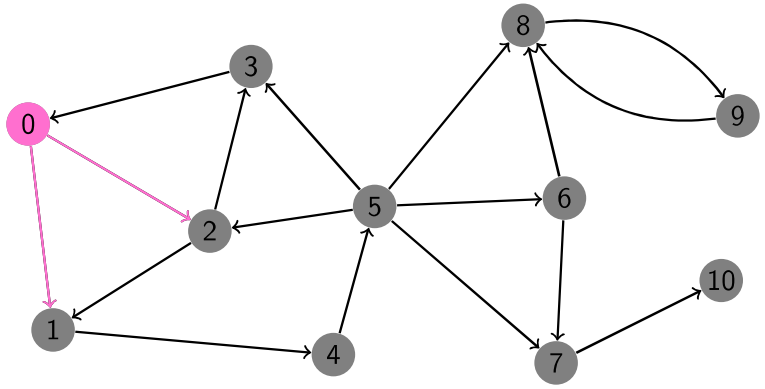
Tìm kiếm theo chiều sâu



Stack: 0 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	0	0	0	0	0	0	0	0	0	0

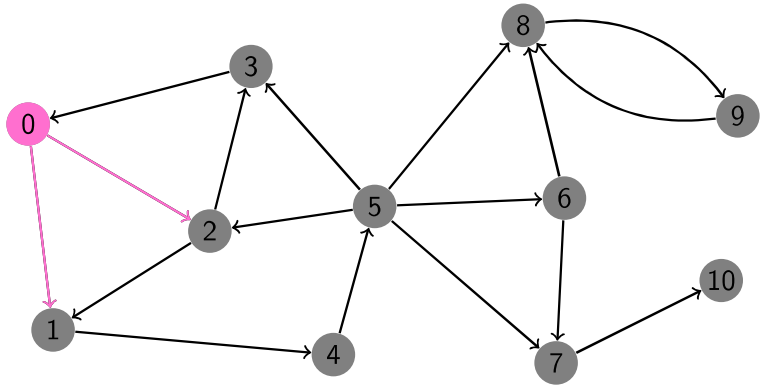
Tìm kiếm theo chiều sâu



Stack: 0 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	0	0	0	0	0	0	0	0	0	0

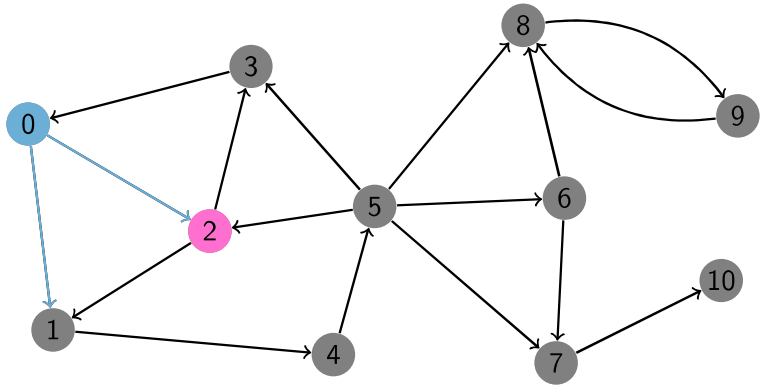
Tìm kiếm theo chiều sâu



Stack: 0 | 2 1

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

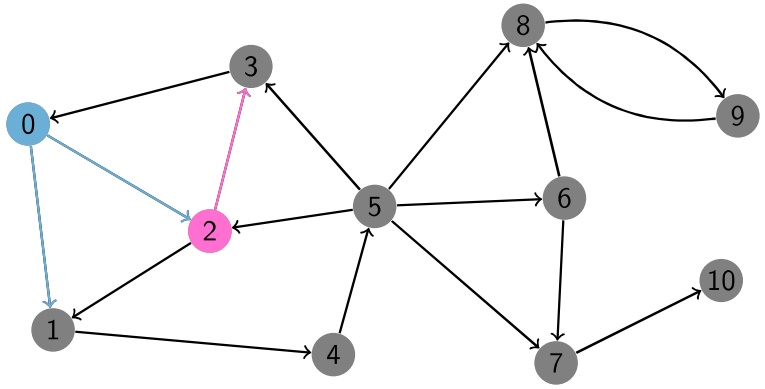
Tìm kiếm theo chiều sâu



Stack: 2 | 1

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

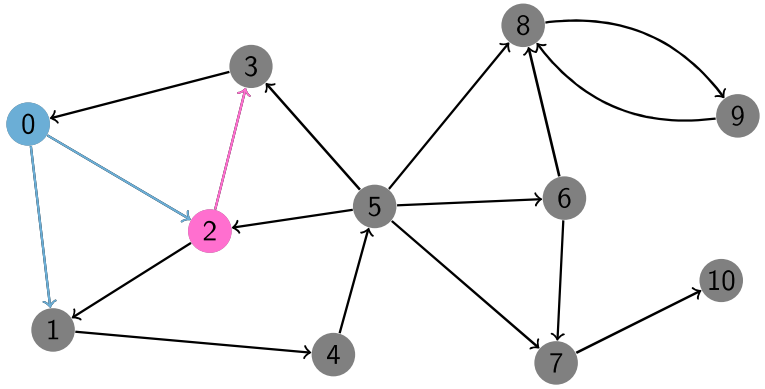
Tìm kiếm theo chiều sâu



Stack: 2 | 1

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

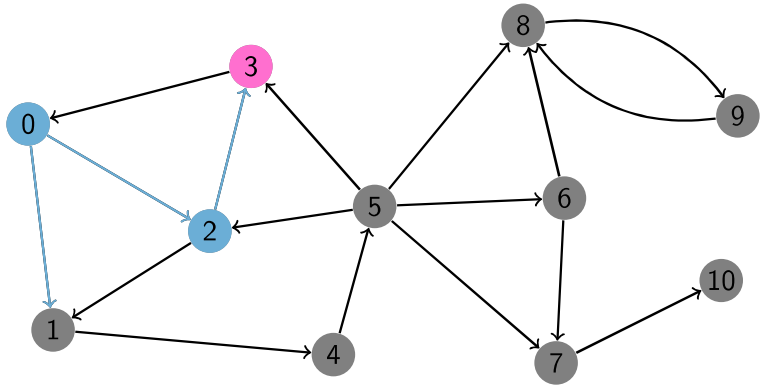
Tìm kiếm theo chiều sâu



Stack: 2 | 3 1

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	0	0	0	0	0	0	0

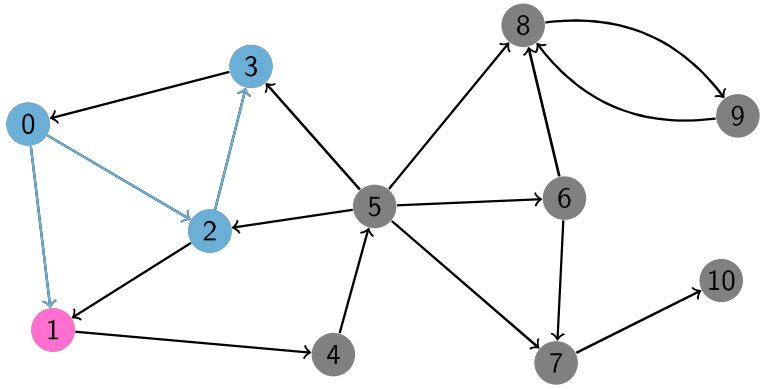
Tìm kiếm theo chiều sâu



Stack: 3 | 1

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	0	0	0	0	0	0	0

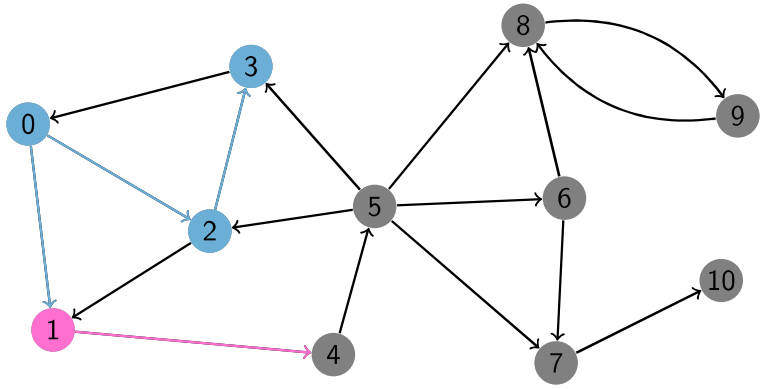
Tìm kiếm theo chiều sâu



Stack: 1 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	0	0	0	0	0	0	0

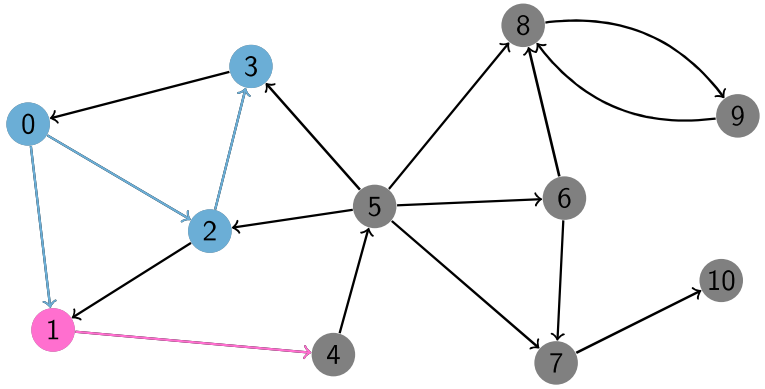
Tìm kiếm theo chiều sâu



Stack: 1 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	0	0	0	0	0	0	0

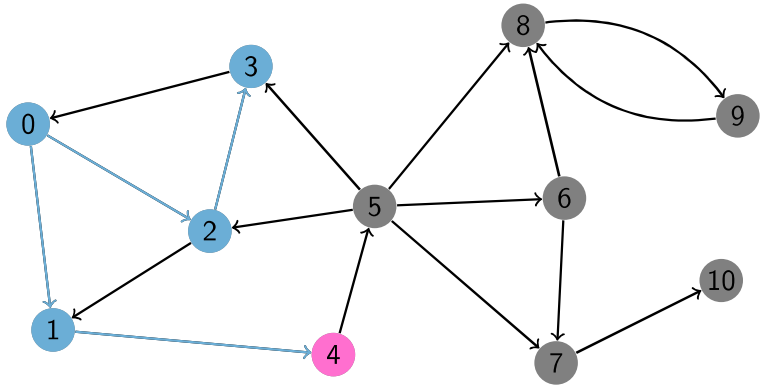
Tìm kiếm theo chiều sâu



Stack: 1 | 4

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

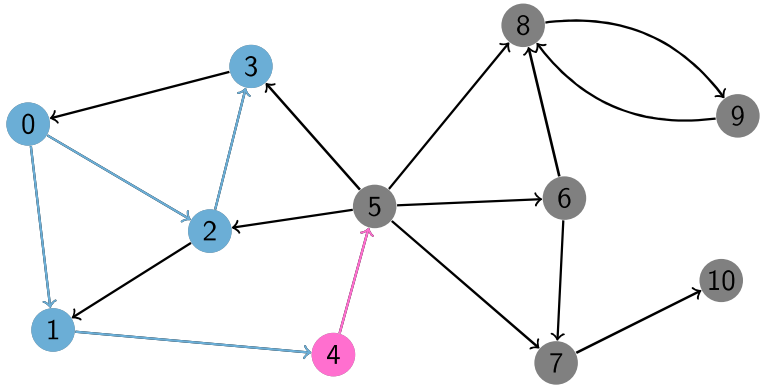
Tìm kiếm theo chiều sâu



Stack: 4 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

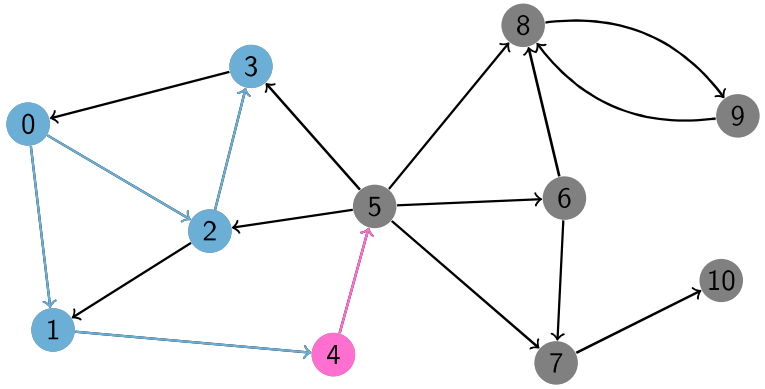
Tìm kiếm theo chiều sâu



Stack: 4 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

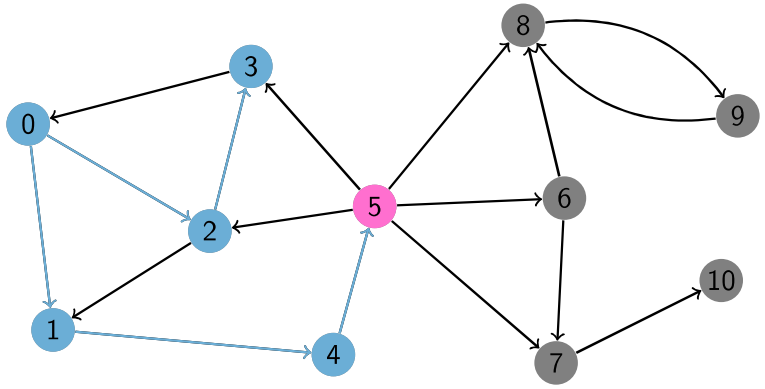
Tìm kiếm theo chiều sâu



Stack: 4 | 5

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

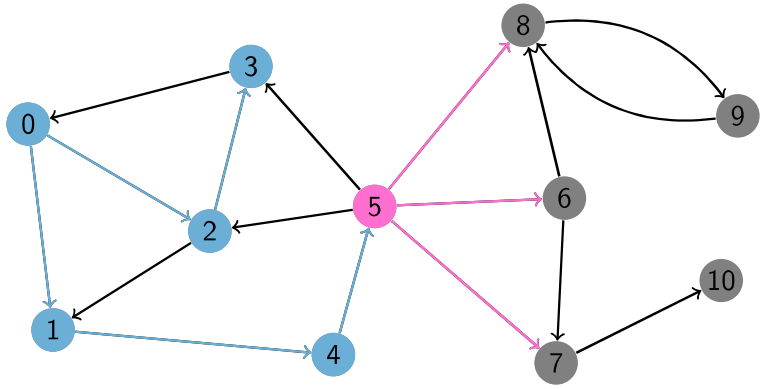
Tìm kiếm theo chiều sâu



Stack: 5 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

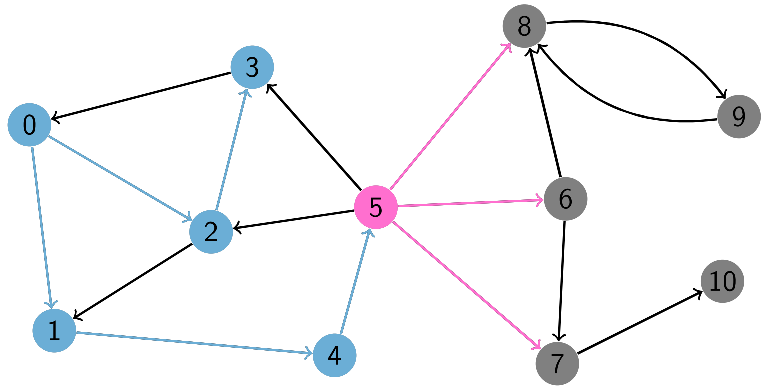
Tìm kiếm theo chiều sâu



Stack: 5 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

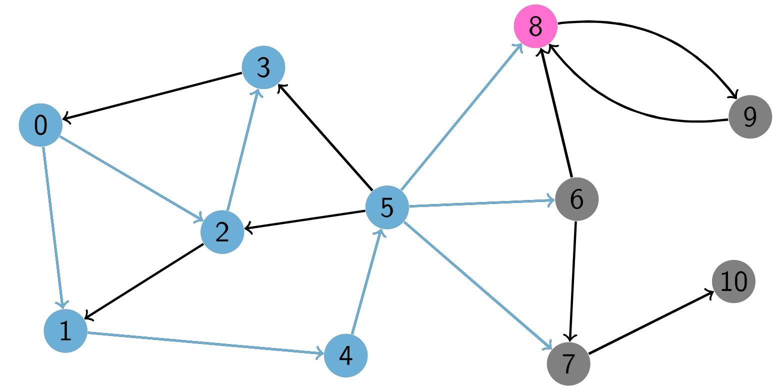
Tìm kiếm theo chiều sâu



Stack: 5 | 8 6 7

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

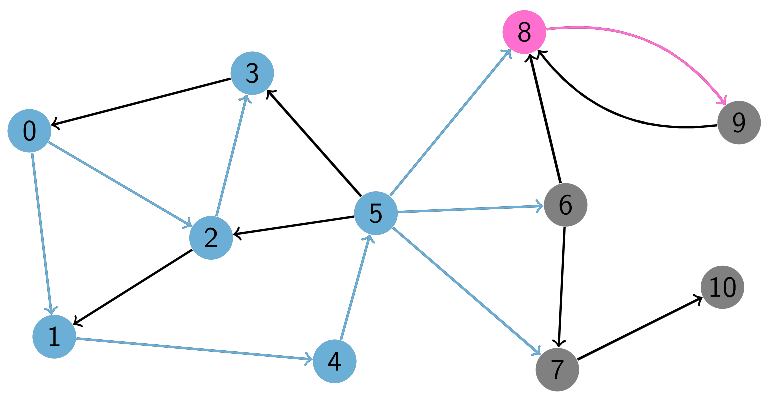
Tìm kiếm theo chiều sâu



Stack: 8 | 6 7

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

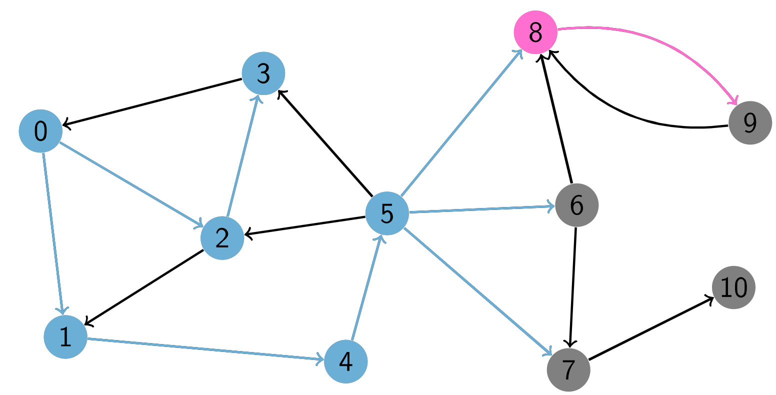
Tìm kiếm theo chiều sâu



Stack: 8 | 6 7

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

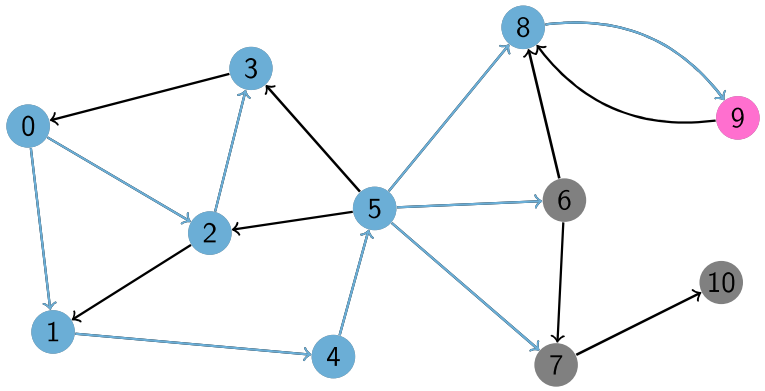
Tìm kiếm theo chiều sâu



Stack: 8 | 9 6 7

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	0

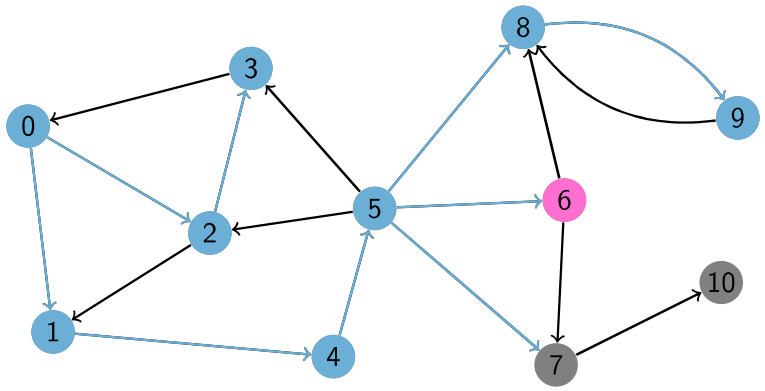
Tìm kiếm theo chiều sâu



Stack: 9 | 6 7

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	0

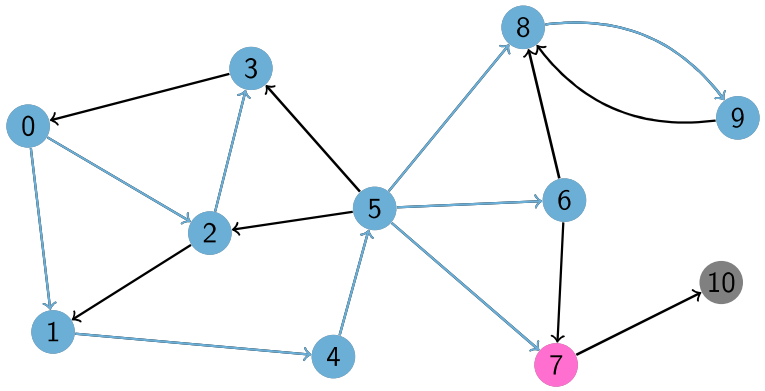
Tìm kiếm theo chiều sâu



Stack: 6 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	0

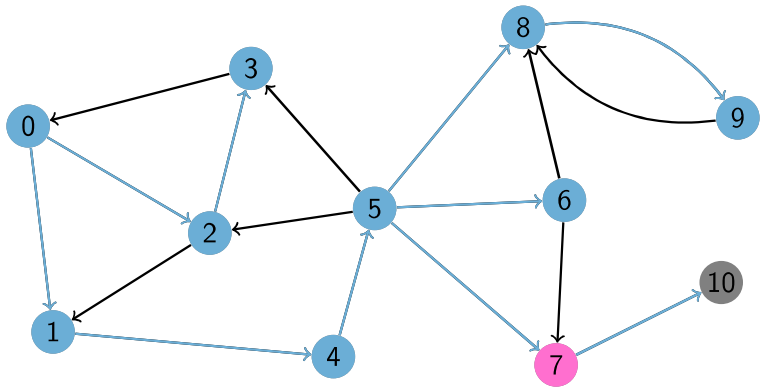
Tìm kiếm theo chiều sâu



Stack: 7 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	0

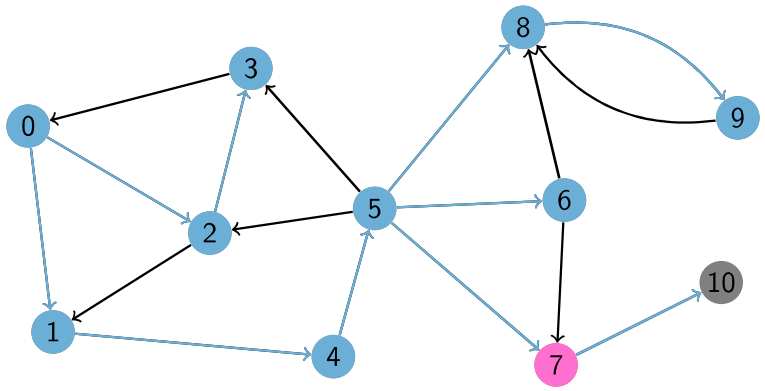
Tìm kiếm theo chiều sâu



Stack: 7 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	0

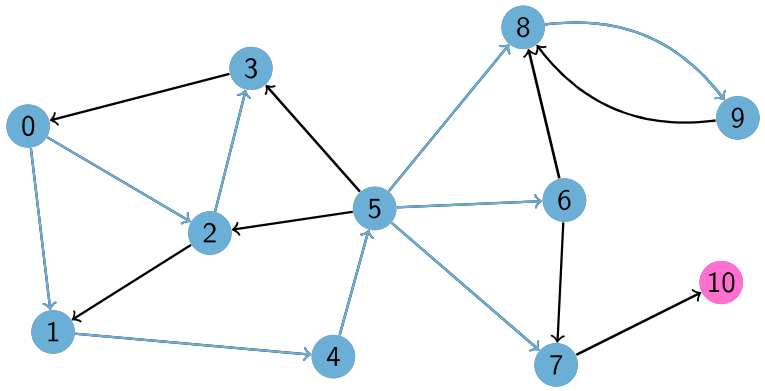
Tìm kiếm theo chiều sâu



Stack: 7 | 10

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

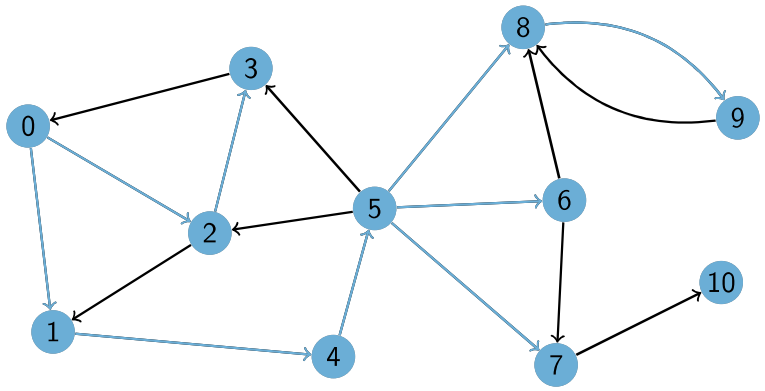
Tìm kiếm theo chiều sâu



Stack: 10 |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

Tìm kiếm theo chiều sâu



Stack: |

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

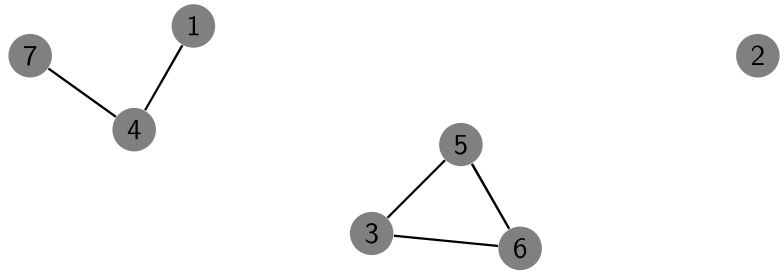
Tìm kiếm theo chiều sâu

```cpp
vector<int> adj[1000];
vector<bool> visited(1000, false);

void dfs(int u) {
    if (visited[u]) {
        return;
    }

    visited[u] = true;

    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        dfs(v);
    }
}
```

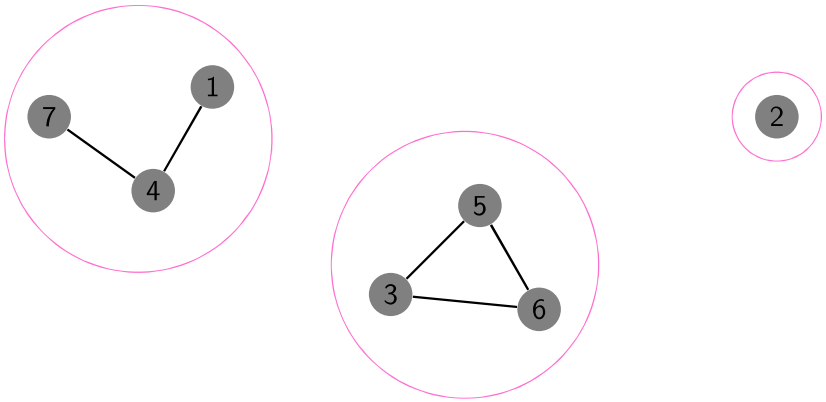
Thành phần liên thông

- Một đồ thị vô hướng có thể phân chia thành các thành phần liên thông (TPLT)
- Một TPLT là một tập con tối đa các đỉnh sao cho giữa hai đỉnh bất kỳ trong tập đều có đường đi giữa chúng
- Ta có thể giải quyết bài này bằng vài thuật toán khác nhau. Có thể sử dụng Union-Find để tìm các TPLT

Thành phần liên thông



Thành phần liên thông



Thành phần liên thông

- Cũng có thể sử dụng tìm kiếm theo chiều sâu để tìm các TPLT
- Lấy một đỉnh bất kỳ và gọi thuật toán tìm kiếm theo chiều sâu xuất phát từ đỉnh đó
- Tất cả các đỉnh đến được từ đỉnh xuất phát đó đều thuộc cùng một TPLT
- Lặp lại quá trình trên đến khi thu được toàn bộ các TPLTR
- Độ phức tạp $O(n+m)$

Thành phần liên thông

```
vector<int> adj[1000];
vector<int> component(1000, -1);

void find_component(int cur_comp, int u) {
    if (component[u] != -1) {
        return;
    }

    component[u] = cur_comp;

    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        find_component(cur_comp, v);
    }
}

int components = 0;
for (int u = 0; u < n; u++) {
    if (component[u] == -1) {
        find_component(components, u);
        components++;
    }
}
```


Cây DFS

- Khi gọi DFS từ một đỉnh nào đó, các vết tìm kiếm tạo thành một cây
- Khi duyệt từ một đỉnh đến một đỉnh khác chưa được thăm, cạnh duyệt qua đó gọi là *cạnh xuôi (forward edge)*
- Khi duyệt từ một đỉnh đến một đỉnh đã thăm trước đó rồi, cạnh duyệt qua đó gọi là *cạnh ngược (backward edge)*
- Tập các cạnh ngược tạo thành một cây

- *xem ví dụ*

Cây DFS

- Cây từ các cạnh xuôi, cùng với các cạnh ngược, chứa đựng rất nhiều thông tin về đồ thị ban đầu

- Ví dụ: một cạnh ngược luôn nằm trên một chu trình của đồ thị ban đầu
- Nếu không có cạnh ngược thì không có chu trình trong đồ thị ban đầu (nghĩa là loại đồ thị không có chu trình - acyclic graph)

Phân tích cây DFS

- Hãy quan sát kỹ hơn cây DFS

- Đầu tiên, hãy đánh số các đỉnh theo thứ tự duyệt của thuật toán DFS
- Với mỗi đỉnh, ta muốn biết số nhỏ nhất của một đỉnh đã được thăm khi đi xuống cây con có gốc tại đỉnh đỉnh đó

- Tại sao? Sau một lát..

- *see example*

Phân tích cây DFS

```
const int n = 1000;
vector<int> adj[n];
vector<int> low(n), num(n, -1);
int curnum = 0;

void analyze(int u, int p) {
    low[u] = num[u] = curnum++;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (v == p) continue;
        if (num[v] == -1) {
            analyze(v, u);
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], num[v]);
        }
    }
}

for (int u = 0; u < n; u++) {
    if (num[u] == -1) {
        analyze(u, -1);
    }
}
```


Phân tích cây DFS

- Độ phức tạp chỉ là $O(n+m)$, do chỉ gọi một lần DFS
- Bây giờ hãy xem một số ứng dụng của thuật toán này

Cầu

- Cho đồ thị không trọng số $G = V, E)$
- Không mất tính tổng quát, giả sử G liên thông (nghĩa là G là một TPLT lớn)
- Tìm một cạnh mà nếu loại bỏ cạnh đó ra khỏi G thì G mất tính liên thông
- Thuật toán trực tiếp: Thử loại bỏ từng cạnh một, và tính số TPLT thu được
- Cách này thiếu hiệu quả, $O(m(n+m))$

Cầu

- Bây giờ quan sát các giá trị tính được từ cây DFS
- Nhận thấy rằng một cạnh xuôi (u, v) là cầu khi và chỉ khi `low[v] >num[u]`
- Như vậy chỉ cần mở rộng thuật toán phân tích cây DFS ở trên để đưa ra toàn bộ cầu
- Độ phức tạp thuật toán vẫn chỉ là $O(n+m)$

Cầu

```
const int n = 1000;
vector<int>  adj[n], low(n), num(n, -1);
int curnum = 0;
vector<pair<int, int> > bridges;

void find_bridges(int u, int p) {
    low[u] = num[u] = curnum++;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (v == p) continue;
        if (num[v] == -1) {
            find_bridges(v, u);
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], num[v]);
        }
        if (low[v] > num[u]) {
            bridges.push_back(make_pair(u, v));
        }
    }
}
for (int u = 0; u < n; u++) {
    if (num[u] == -1)
        find_bridges(u, -1);
}
```


TPLT mạnh

- Ta đã biết cách tìm các TPLT trên đồ thị vô hướng
- Thế trên đồ thị có hướng thì sao?
- Các TPLT này có một chút khác biệt trên đồ thị có hướng, bởi vì nếu v đến được từ u thì không có nghĩa là u đến được từ v
- Tuy vậy, định nghĩa vẫn tương tự
- Một TPLT mạnh là một tập con tối đa các đỉnh sao cho giữa hai đỉnh bất kỳ trong tập luôn có đường đi từ đỉnh này đến đỉnh kia và ngược lại

TPLT mạnh

- Thuật toán tìm các TPLT ở trên không áp dụng được
- Thay vào đó ta có thể sử dụng cây DFS để tìm các TPLT mạnh này
- *xem ví dụ*

TPLT mạnh

```
vector<int> adj[100];
vector<int> low(100), num(100, -1);
vector<bool> incomp(100, false);
int curnum = 0;

stack<int> comp;

void scc(int u) {

    // scc code...
}

for (int i = 0; i < n; i++) {
    if (num[i] == -1) {
        scc(i);
    }
}
```

TPLT mạnh

```
void scc(int u) {
    comp.push(u);
    incomp[u] = true;
    low[u] = num[u] = curnum++;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (num[v] == -1) {
            scc(v);
            low[u] = min(low[u], low[v]);
        } else if (incomp[v]) {
            low[u] = min(low[u], num[v]);
        }
    }
    if (num[u] == low[u]) {
        printf("comp: ");
        while (true) {
            int cur = comp.top();
            comp.pop();
            incomp[cur] = false;
            printf("%d, ", cur);
            if (cur == u) break;
        }
        printf("\n");
    }
}
```


TPLT mạnh

- Độ phức tạp thuật toán?
- Cơ bản là chỉ dùng thuật toán phân tích cây DFS (có độ phức tạp $O(n+m)$), cộng thêm một vòng lặp để xây dựng TPLT mạnh
- Mỗi đỉnh chỉ thuộc một TPLT...
- Vì vậy độ phức tạp vẫn chỉ là $O(n+m)$

Bài toán ví dụ: Come and Go

- http://uva.onlinejudge.org/external/118/11838.html

Sắp xếp topo

- Có n công việc
- Mỗi công việc i có một danh sách các công việc cần phải hoàn thành trước khi bắt đầu công việc i
- Hãy tìm một trình tự mà ta có thể thực hiện toàn bộ các công việc
- có thể biểu diễn trên một đồ thị có hướng
  - Mỗi công việc là một đỉnh của đồ thị
  - Nếu công việc j phải hoàn thành trước công việc i, thì thêm một cạnh có hướng từ đỉnh i đến đỉnh j
- Lưu ý là không thể có một trình tự thỏa mãn nếu như đồ thị có chu trình
- Có thể sửa đổi thuật toán DFS để đưa ra một trình tự thỏa mãn trong thời gian $O(n+m)$, hoặc đưa ra không có trình tự thỏa mãn

Sắp xếp topo

```
vector<int> adj[1000];
vector<bool> visited(1000, false);
vector<int> order;

void topsort(int u) {
    if (visited[u])
        return;

    visited[u] = true;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        topsort(v);
    }

    order.push_back(u);
}

for (int u = 0; u < n; u++)
    topsort(u);
```

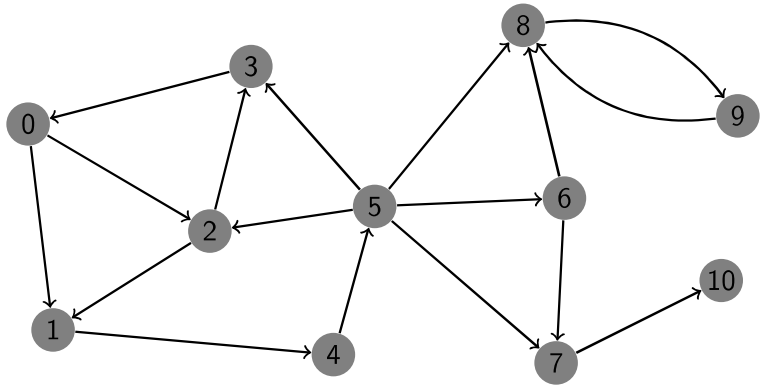

Bài toán ví dụ: Ordering Tasks

- http://uva.onlinejudge.org/external/103/10305.html

Tìm kiếm theo chiều rộng - BFS

- Thuật toán tìm kiếm theo chiều rộng chỉ khác DFS ở trình tự thăm các đỉnh
- BFS cho trình tự thăm tăng dần theo khoảng cách từ đỉnh xuất phát

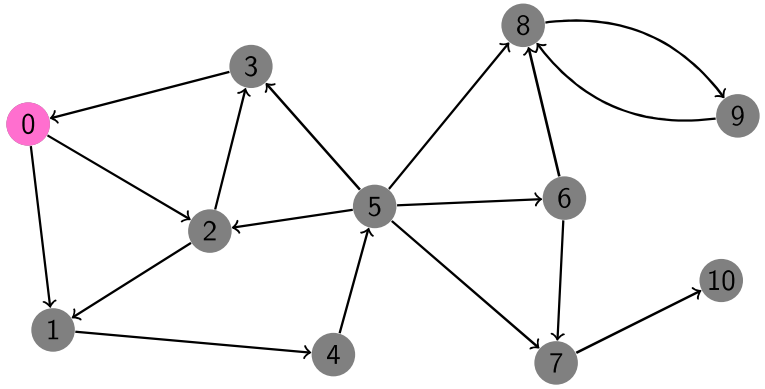
BFS



Queue:

	0	1	2	3	4	5	6	7	8	9	10
marked	0	0	0	0	0	0	0	0	0	0	0

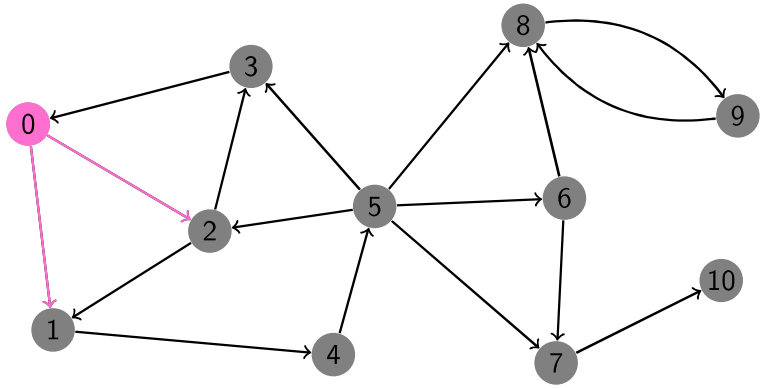
BFS



Queue: 0

	0	1	2	3	4	5	6	7	8	9	10
marked	1	0	0	0	0	0	0	0	0	0	0

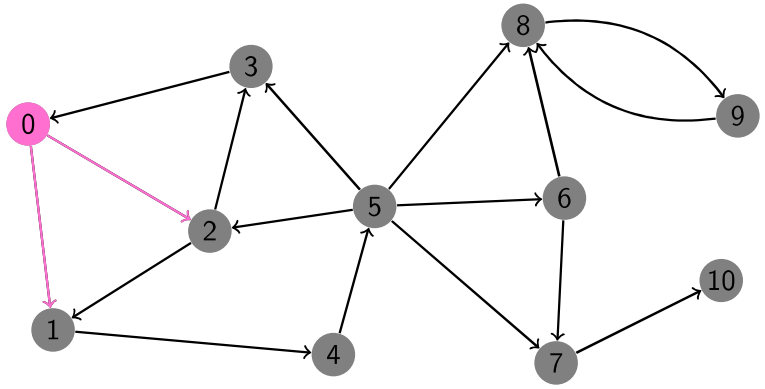
BFS



Queue: 0

	0	1	2	3	4	5	6	7	8	9	10
marked	1	0	0	0	0	0	0	0	0	0	0

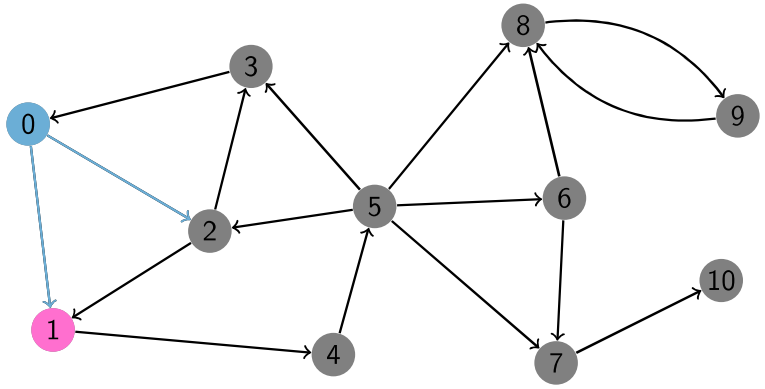
BFS



Queue: 0 1 2

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

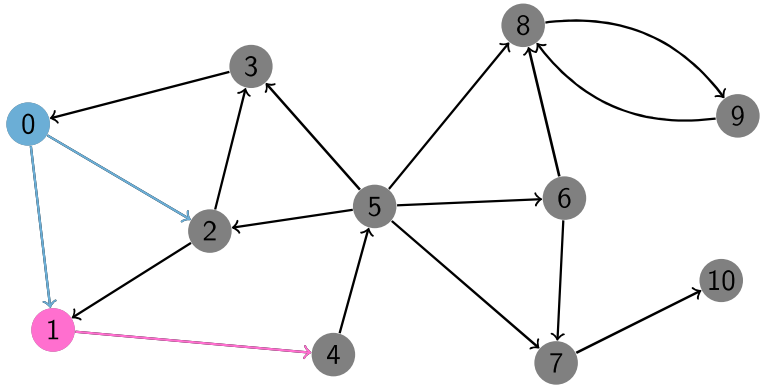
BFS



Queue: 1 2

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

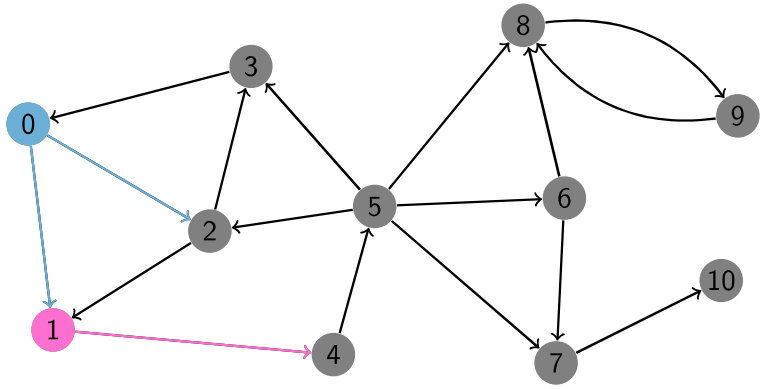
BFS



Queue: 1 2

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	0	0	0	0	0	0	0

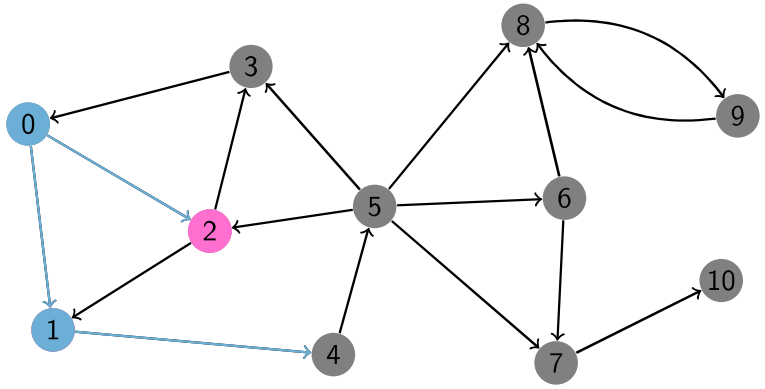
BFS



Queue: 1 2 4

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	1	0	0	0	0	0	0

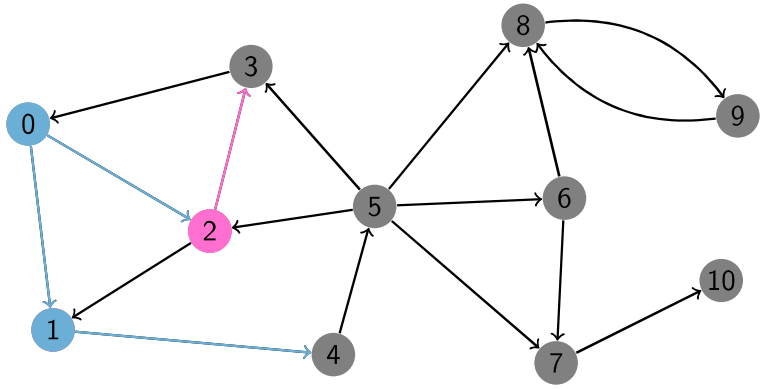
BFS



Queue: 2 4

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	1	0	0	0	0	0	0

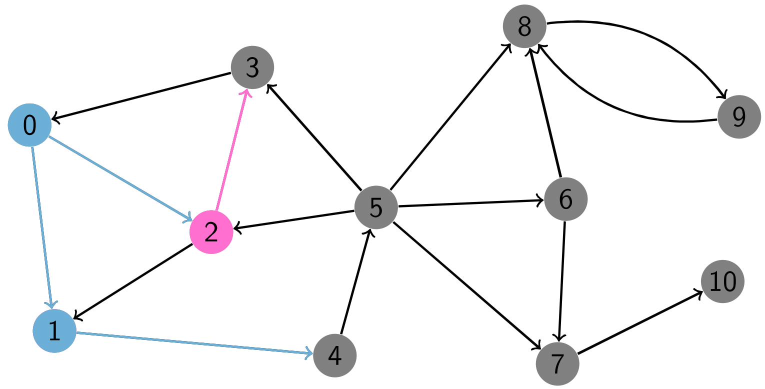
BFS



Queue: 2 4

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	0	1	0	0	0	0	0	0

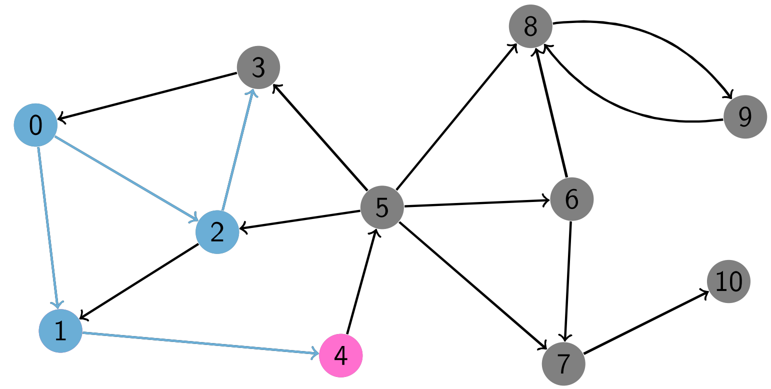
BFS



Queue: 2 4 3

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

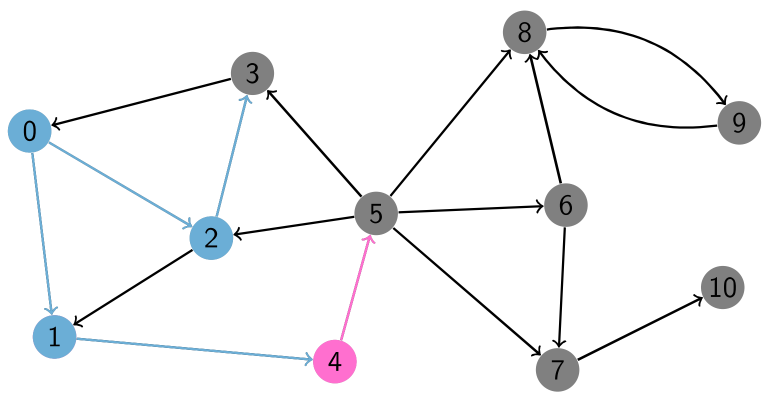
BFS



Queue: 4 3

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

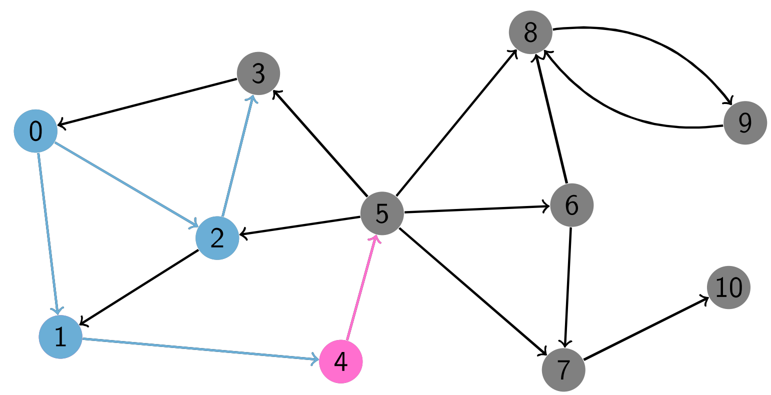
BFS



Queue: 4 3

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	0	0	0	0	0	0

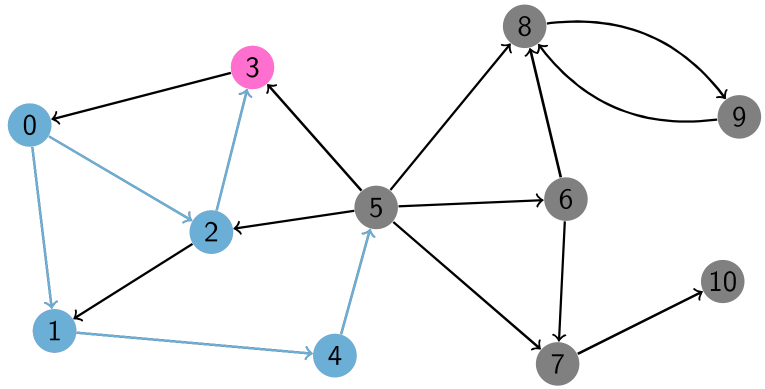
BFS



Queue: 4 3 5

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

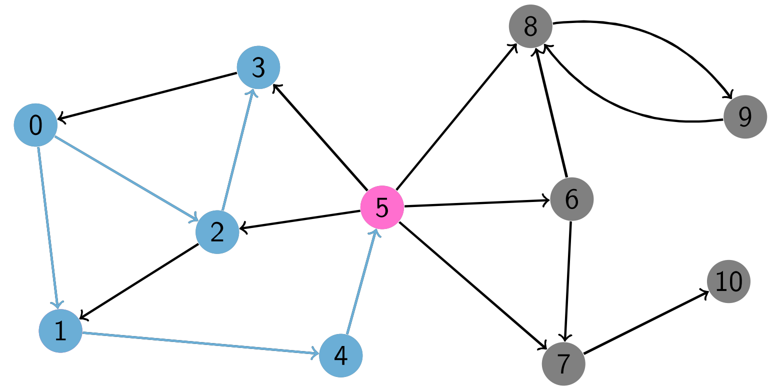
BFS



Queue: 3 5

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

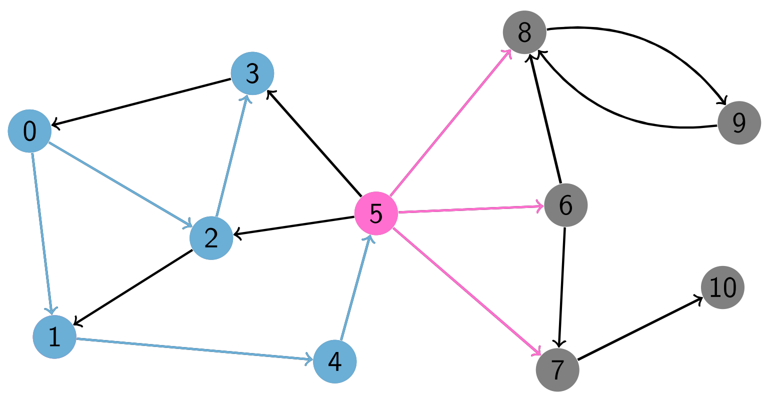
BFS



Queue: 5

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

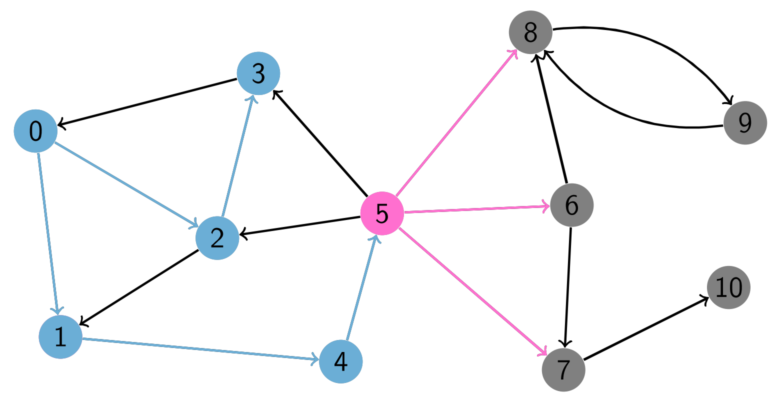
BFS



Queue: 5

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	0	0	0	0	0

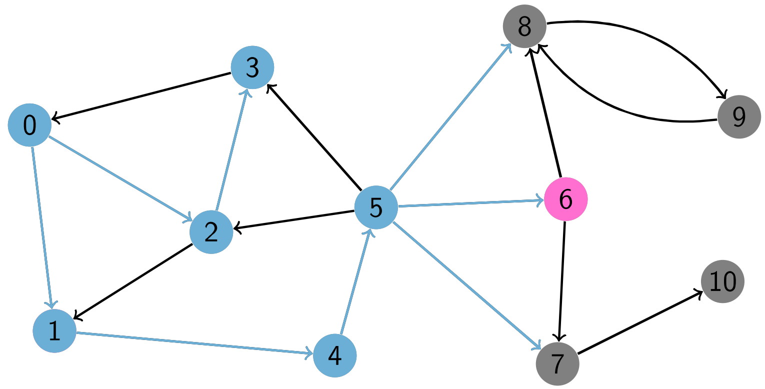
BFS



Queue: 5 6 7 8

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

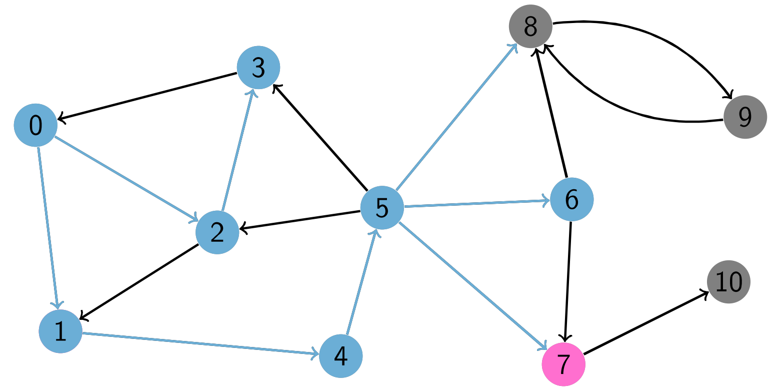
BFS



Queue: 6 7 8

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

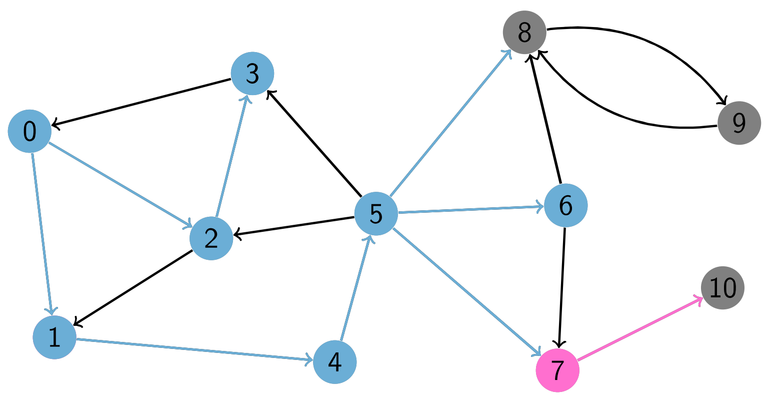
BFS



Queue: 7 8

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

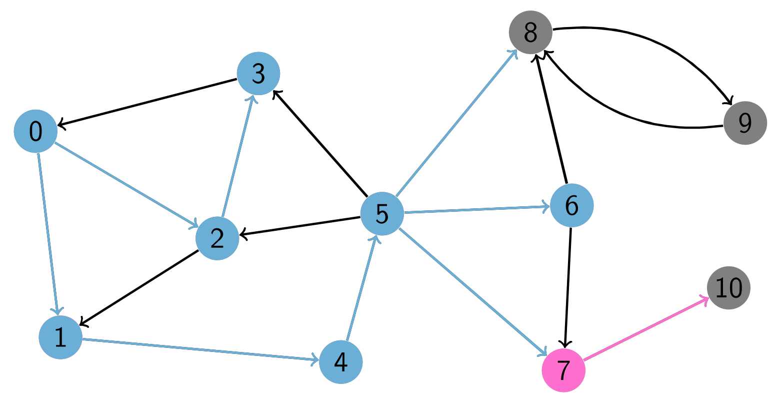
BFS



Queue: 7 8

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	0

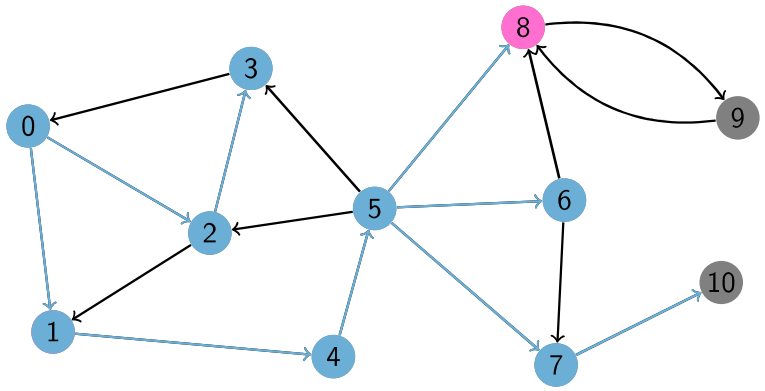
BFS



Queue: 7 8 10

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	1

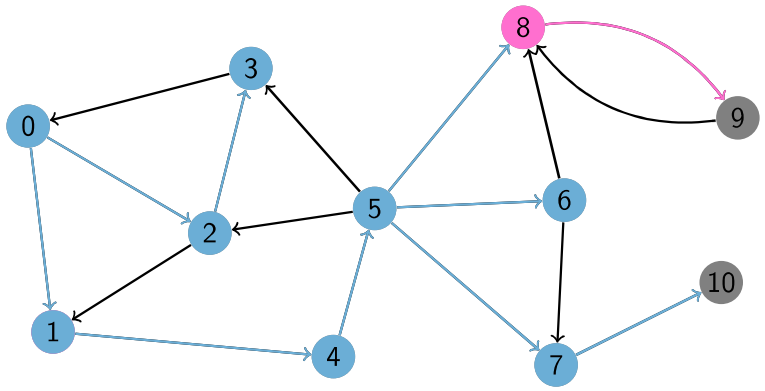
BFS



Queue: 8 10

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	1

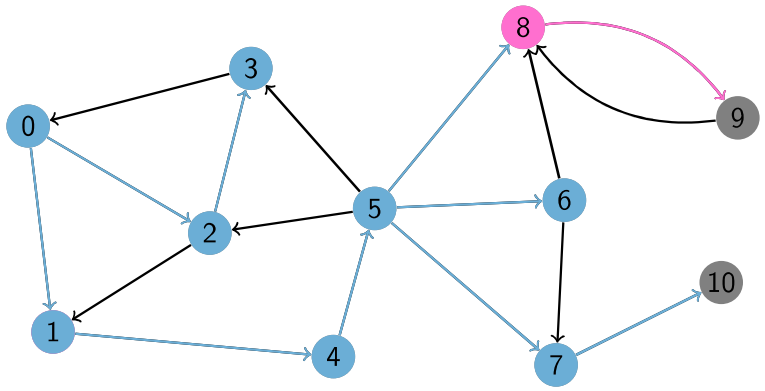
BFS



Queue: 8 10

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	0	1

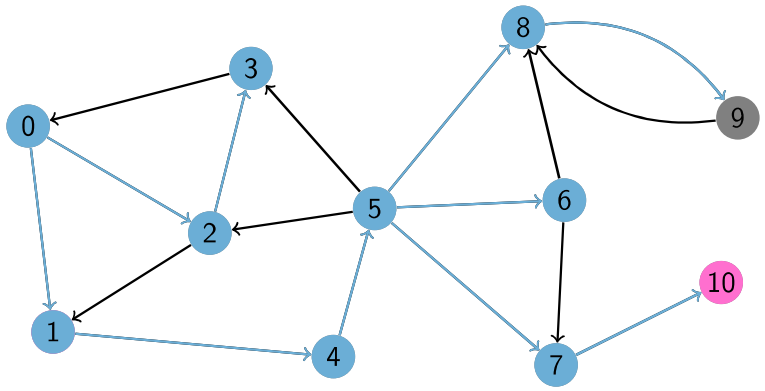
BFS



Queue: 8 10 9

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

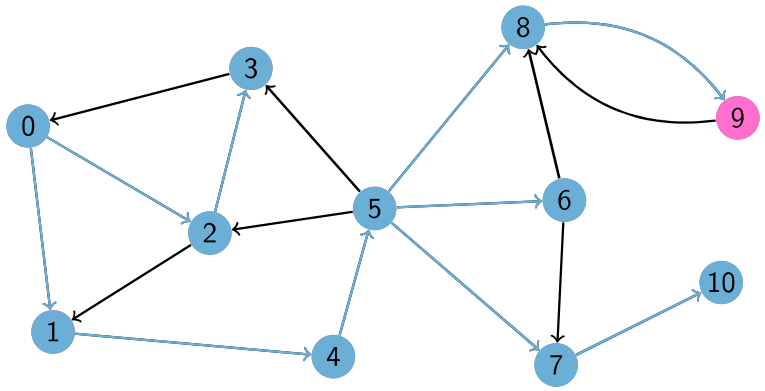
BFS



Queue: 10 9

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

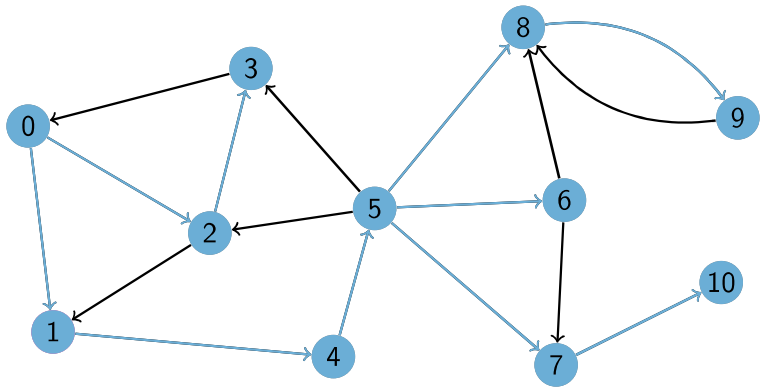
BFS



Queue: 9

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

BFS



Queue:

	0	1	2	3	4	5	6	7	8	9	10
marked	1	1	1	1	1	1	1	1	1	1	1

BFS

```
vector<int> adj[1000];
vector<bool> visited(1000, false);

queue<int> Q;
Q.push(start);
visited[start] = true;

while (!Q.empty()) {
    int u = Q.front(); Q.pop();

    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (!visited[v]) {
            Q.push(v);
            visited[v] = true;
        }
    }
}
```


Đường đi ngắn nhất trên đồ thị không trọng số

- Cho đồ thị không trọng số $G = (V, E)$, hãy tìm đường đi ngắn nhất từ đỉnh A đến đỉnh B
- Có nghĩa là tìm đường đi từ A đến B đi qua ít cạnh nhất
- BFS duyệt các đỉnh theo trình tự tăng dần khoảng cách từ đỉnh xuất phát
- Vì vậy chỉ cần gọi một lần BFS từ đỉnh A đến khi tìm thấy B
- Hoặc duyệt qua toàn bộ G, và ta có đường đi ngắn nhất từ A đến tất cả các đỉnh khác
- Độ phức tạp: $O(n + m)$

Đường đi ngắn nhất trên đồ thị không trọng số

```
vector<int> adj[1000];
vector<bool> dist(1000, -1);

queue<int> Q;
Q.push(A);
dist[A] = 0;

while (!Q.empty()) {
    int u = Q.front(); Q.pop();

    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (dist[v] == -1) {
            Q.push(v);
            dist[v] = 1 + dist[u];
        }
    }
}

printf("%d\n", dist[B]);
```