HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

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

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

ĐỒ THỊ (PHẦN 1)

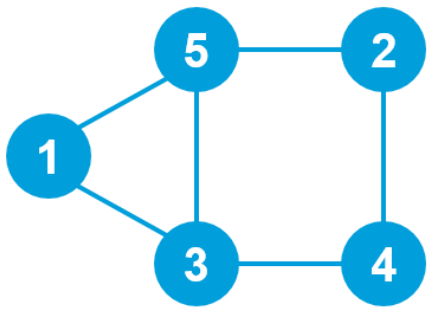
ONE LOVE. ONE FUTURE.

NỘI DUNG

- Nhắc lại khái niệm cơ bản về đồ thị
- Cấu trúc dữ liệu biểu diễn đồ thị
- Duyệt đồ thị
- Tìm thành phần liên thông
- Kiểm tra đồ thị hai phía

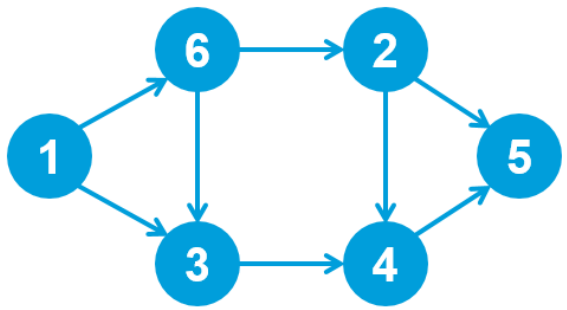
NHẮC LẠI KHÁI NIỆM CƠ BẢN VỀ ĐỒ THỊ

- Đồ thị là cấu trúc bao gồm các thực thể (còn gọi là đỉnh) và liên kết giữa các thực thể (còn gọi là cạnh hoặc cung)
- Ký hiệu đồ thị $G = (V, E)$, trong đó V là tập đỉnh và E là tập cạnh (cung)
- $(u,v) \in E$: ta nói v kề



Đồ thị vô hướng

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{(1, 3), (1,5), (2, 4), (2, 5), (3, 4), (3, 5)\}$



Đồ thị có hướng

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 3), (1,6), (2, 4), (2, 5), (6, 2), (3, 4), (6, 3), (4, 5)\}$

NHẮC LẠI KHÁI NIỆM CƠ BẢN VỀ ĐỒ THỊ

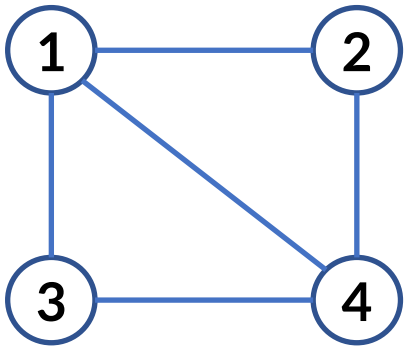
- Cho $G = (V, E)$ là một đồ thị
  - Đường đi từ đỉnh s đến đỉnh t trên đồ thị là dãy các đỉnh $v_1, v_2, \ldots, v_k$ trong đó
    - $s = v_1, t = v_k$
    - $(v_i, v_{i+1}) \in E$
  - Chu trình là đường đi trong đó đỉnh đầu và cuối trùng nhau
- Đồ thị vô hướng G được gọi là liên thông nếu giữa 2 đỉnh bất kỳ của G luôn có đường đi giữa chúng

CẤU TRÚC DỮ LIỆU BIỂU DIỄN ĐỒ THỊ

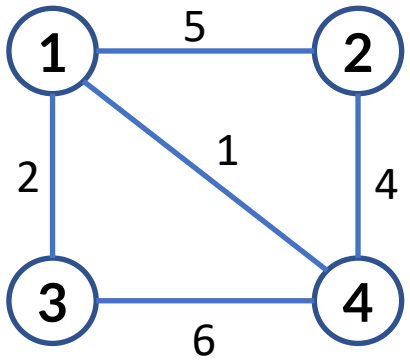
- Ma trận kề, ma trận trọng số
- Danh sách kề
- Danh sách cạnh

CẤU TRÚC DỮ LIỆU BIỂU DIỄN ĐỒ THỊ

- Ma trận kề, ma trận trọng số



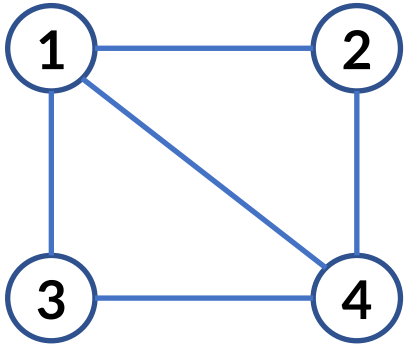
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0



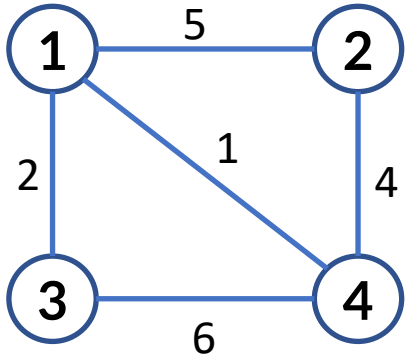
	1	2	3	4
1	0	5	2	1
2	5	0	0	4
3	2	0	0	6
4	1	4	6	0

CẤU TRÚC DỮ LIỆU BIỂU DIỄN ĐỒ THỊ

- Danh sách kề
  - A[v]: danh sách các đỉnh (hoặc cạnh/cung đối với đồ thị trọng số) kề với đỉnh v



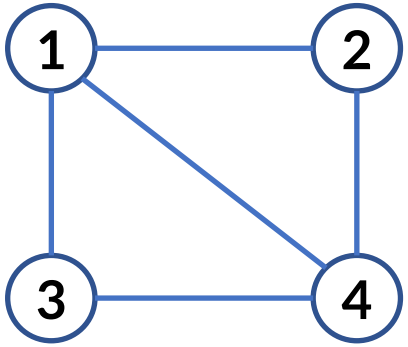
- A[1] = [2, 3, 4]
- A[2] = [1, 4]
- A[3] = [1, 4]
- A[4] = [1, 2, 3]



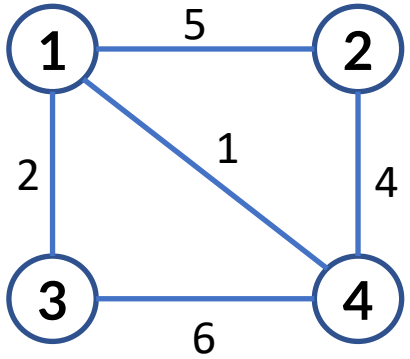
- A[1] = [(1,2,5), (1,3,2), (1,4,1)]
- A[2] = [(1,2,5), (2,4,4)]
- A[3] = [(1,3,2), (3,4,6)]
- A[4] = [(1,4,1), (2,4,4), (3,4,6)]

CẤU TRÚC DỮ LIỆU BIỂU DIỄN ĐỒ THỊ

- **Danh sách cạnh**
  - **E: danh sách các cạnh/cung của đồ thị**



Danh sách cạnh: E = [(1,2), (1,3), (1,4), (2, 4), (3,4)]



Danh sách cạnh kèm trọng số trên cạnh
E = [(1,2, 5), (1,3, 2), (1,4, 1), (2, 4, 4), (3,4, 6)]

DUYỆT ĐỒ THỊ

- Duyệt đồ thị theo chiều sâu: thăm các đỉnh của đồ thị, mỗi đỉnh đúng 1 lần
- A[v]: danh sách các đỉnh kề với v

```
DFS(u, A) {// duyệt theo chiều sâu từ đỉnh u
  visited[u] = true; //Thăm đỉnh u;
  for v in A[u] do {
     if visited[v] = false then {
        DFS(v, A);
     }
  }
}
```

```
DFS(G = (V, A)){ // duyệt theo chiều sâu trên G
  for v in V do visited[v] = false;
  for v in V do {
    if visited[v] = false then {
       DFS(v, A);
    }
  }
}
```

DUYỆT ĐỒ THỊ

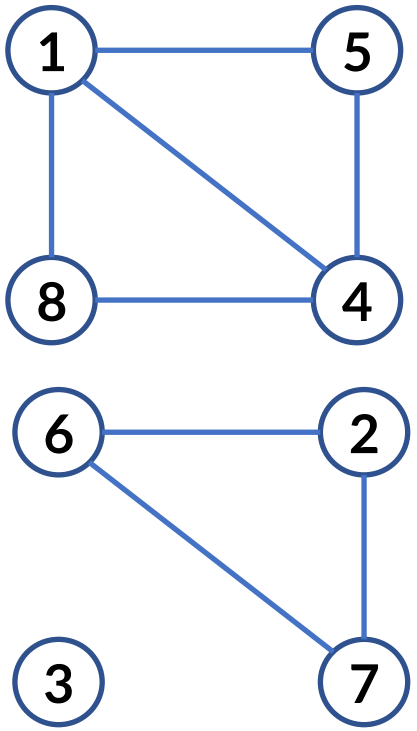
- Duyệt đồ thị theo chiều rộng: thăm các đỉnh của đồ thị, mỗi đỉnh đúng 1 lần
- A[v]: danh sách các đỉnh kề với v

```
BFS(G = (V, A)){ // duyệt theo chiều rộng trên G
  for v in V do visited[v] = false;
  for v in V do {
    if visited[v] = false then {
       BFS(v, A);
    }
  }
}
```

```
BFS(u, A) {// duyệt theo chiều rộng từ đỉnh u
  Q = hàng đợi rỗng;
  Q.push(u);  visited[u] = true; // thăm đỉnh u
  while Q not empty do {
    v = Q.pop();
    for x in A[v] do
      if visited[x] = false then {
         Q.push(x); visited[x] = true; // thăm x
      }
  }
}
```

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

- Mô tả bài toán
  - Cho đồ thị vô hướng G = (V, A) trong đó
    - V là tập đỉnh
    - A là cấu trúc danh sách kề: A[v] là danh sách các đỉnh kề với v
  - Cần tìm các thành phần liên thông của G
- Thuật toán:
  - Áp dụng duyệt theo chiều sâu (hoặc duyệt theo chiều rộng) trên G
  - DFS(u) cho phép thăm tất cả các đỉnh thuộc cùng thành phần liên thông với u



TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

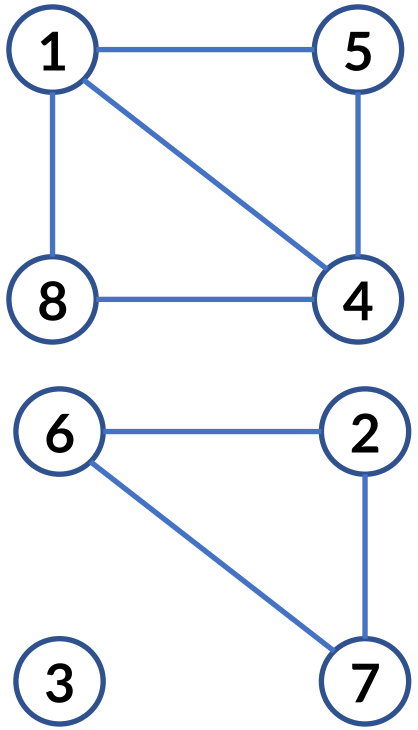
- nbCC: số thành phần liên thông của G
- C[v]: chỉ số (chạy từ 1 đến nbCC) của thành phần liên thông chứa đỉnh v

```
DFS(u, A) {// duyệt theo chiều sâu từ đỉnh u
  visited[u] = true; //Thăm đỉnh u;
  C[u] = nbCC;
  for v in A[u] do {
     if visited[v] = false then {
        DFS(v, A);
     }
  }
}
```

```
DFS(G = (V, A)){ // duyệt theo chiều sâu trên G
  for v in V do visited[v] = false;
  nbCC = 0;
  for v in V do {
    if visited[v] = false then {
       nbCC = nbCC + 1;
       DFS(v, A);
    }
  }
}
```

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

- Minh họa với ngôn ngữ C
- Dữ liệu
  - Dòng 1: chứa 2 số nguyên dương n và m tương ứng là số đỉnh và số cạnh
  - Dòng *i* + 1 (*i* = 1, 2, …, *m*): chứa 2 số nguyên dương u và v là 2 đầu mút của cạnh thứ *i*
- Kết quả
  - Hiển thị danh sách các đỉnh của mỗi thành phần liên thông tìm được trên 1 dòng



stdin	stdout
8 8 1 4 1 5 1 8 2 6 2 7 4 5 4 8 6 7	C[1]: 1 4 5 8 C[2]: 2 6 7 C[3]: 3

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

```
#include <stdio.h>
#define N 100001
typedef struct Node{
  int id;
  struct Node* next;
}Node;
int n,m; // so dinh va so canh cua G
Node* A[N]; // A[v]: con tro den dau DS ke
int nbCC; // so thanh phan lien thong
int C[N]; // C[v]: chi so thanh phan lien
          // thong chua v
```

```
Node* makeNode(int id){
        Node* p = (Node*)malloc(sizeof(Node));
        p->id = id; p->next = NULL;
        return p;
}
Node* insert(int id, Node* h){
        Node* p = makeNode(id);
        p->next = h;
        return p;
}
```

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

```
void input(){
  scanf("%d %d",&n,&m);
  for(int v = 1; v <= n; v++) A[v] = NULL;
  for(int i = 1; i <= m; i++){
    int u,v; scanf("%d%d",&u,&v);
    A[u] = insert(v,A[u]);
    A[v] = insert(u,A[v]);
  }
}
```

```
void DFS(int u){
  C[u] = nbCC;
  for(Node* p = A[u]; p != NULL; p = p->next){
    int v = p->id;
    if(C[v] == -1){
      DFS(v);
    }
  }
}
```

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

```
void DFSG(){
  for(int u = 1; u <= n; u++) C[u] = -1;
  nbCC = 0;
  for(int u = 1; u <= n; u++){
    if(C[u] == -1){
      nbCC = nbCC + 1;
      DFS(u);
    }
  }
}
```

```
void printCC(){
  for(int k = 1; k <= nbCC; k++){
    printf("C[%d]: ",k);
    for(int v = 1; v <= n; v++){
      if(C[v] == k) printf("%d ",v);
    }
    printf("\n");
  }
}
```

TÌM THÀNH PHẦN LIÊN THÔNG CỦA ĐỒ THỊ

```
int main(){
  input();
  DFSG();
  printCC();
  return 0;
}
```

KIỂM TRA ĐỒ THỊ HAI PHÍA

- Mô tả bài toán
  - Cho đồ thị vô hướng G = (V, A) trong đó
    - V là tập đỉnh
    - A là cấu trúc danh sách kề: A[v] là danh sách các đỉnh kề với v
  - Kiểm tra xem G có phải là đồ thị hai phía hay không?

KIỂM TRA ĐỒ THỊ HAI PHÍA

- Thuật toán:
  - Áp dụng duyệt theo chiều rộng trên G
  - d[v]: mức (độ dài đường đi từ đỉnh xuất phát đến v trong BFS) của đỉnh v
  - BFS(u): thăm tất cả các đỉnh cùng thành phần liên thông với u
    - Nếu thành phần liên thông chứa u là đồ thị hai phía thì các đỉnh v có d[v] chẵn sẽ ở thuộc phía chứa u, còn các đỉnh v có d[v] lẻ sẽ thuộc phía còn lại
    - Nếu phát hiện cạnh (u,v) có d[u] và d[v] cùng tính chẵn lẻ thì đồ thị đã cho không phải là đồ thị hai phía

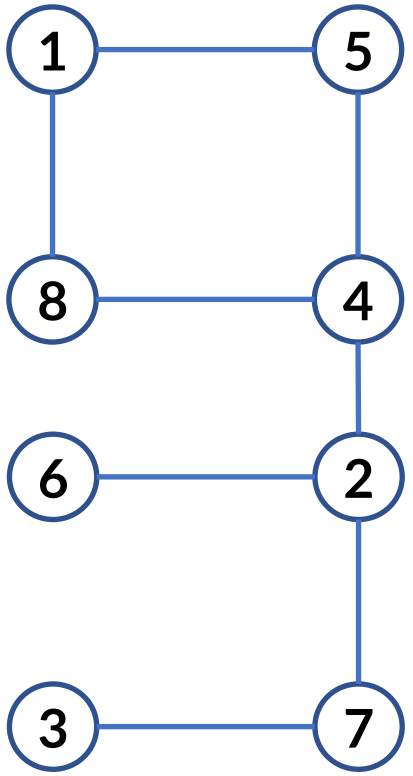
KIỂM TRA ĐỒ THỊ HAI PHÍA – MÃ GIẢ

```
BFS(u) {
  Q = empty queue;  Q.push(u);  d[u] = 0;
  while Q not empty do {
    v = pop();
    for x in A[v] do {
      if(d[x] > -1){
         if  d[v] + d[x]  is even then return false;
      }
      else {  d[x] = d[v] + 1;   Q. push(x);  }
    }
  }
  return true;
}
```

```
solve(){
  for u = 1 to n do  d[u] = -1;
  for u = 1 to n do if(d[u] = -1){
    if(BFS(u) = false) {
       return false;
    }
  }
  return true;
}
```

KIỂM TRA ĐỒ THỊ HAI PHÍA - CODE

- Minh họa với ngôn ngữ C
- Dữ liệu
  - Dòng 1: chứa 2 số nguyên dương n và m tương ứng là số đỉnh và số cạnh
  - Dòng *i* + 1 (*i* = 1, 2, …, *m*): chứa 2 số nguyên dương u và v là 2 đầu mút của cạnh thứ *I*
- Kết quả
  - Ghi ra 1 nếu đồ thị là hai phía và ghi 0, ngược lại



stdin	stdout
8 8 1 5 1 8 2 4 2 6 2 7 3 7 4 5 4 8	1

KIỂM TRA ĐỒ THỊ HAI PHÍA - CODE

```
#include <stdio.h>
#include <stdlib.h>
#define N 100001
typedef struct Node{
  int id;
  struct Node* next;
}Node;
Node* makeNode(int id){
  Node* p = (Node*)malloc(sizeof(Node));
  p->id = id; p->next = NULL;
  return p;
}
```

```
Node* head;
Node* tail;
void initQueue(){
  head = NULL; tail = NULL;
}
int queueEmpty(){
  return head == NULL && tail == NULL;
}
void push(int id){
  Node* p = makeNode(id);
  if(queueEmpty()){ head = p; tail = p; }
  else { tail->next = p; tail = p; }
}
```

KIỂM TRA ĐỒ THỊ HAI PHÍA

```
int pop(){
  if(queueEmpty()) return -1;
  int r = head->id;  Node* tmp = head;
  head = head->next;
  if(head == NULL) tail = NULL;
  free(tmp);
  return r;
}
Node* add(int id, Node* h){
  Node* p = makeNode(id);
  p->next = h;  return p;
}
```

```
int n,m;
Node* A[N];
int d[N];// d[v] is the level of d
void input(){
  scanf("%d %d",&n,&m);
  for(int v = 1; v <= n; v++) A[v] = NULL;
  for(int i = 1; i <= m; i++){
     int u,v;
     scanf("%d%d",&u,&v);
     A[u] = add(v,A[u]);  A[v] = add(u,A[v]);
  }
}
```

KIỂM TRA ĐỒ THỊ HAI PHÍA

```
int BFS(int u){
  initQueue();  push(u);  d[u] = 0;
  while(!queueEmpty()){
    int v = pop();
    for(Node* p = A[v]; p != NULL; p = p->next){
       int x = p->id;
       if(d[x] > -1){ if(d[v] % 2 == d[x] % 2) return 0;  }
       else{  d[x] = d[v] + 1;   push(x);  }
    }
  }
  return 1;
}
```

```
void solve(){
  for(int v = 1; v <= n; v++)  d[v] = -1;
  int ans = 1;
  for(int v= 1; v <= n; v++) if(d[v]== -1){
    if(!BFS(v)){  ans = 0; break;  }
  }
  printf("%d",ans);
}
int main(){
  input();
  solve();
  return 0;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !