



THUẬT TOÁN ỨNG DỤNG

Đồ thị



1. Khái niệm đồ thị
2. Biểu diễn đồ thị trong máy tính
3. Duyệt đồ thị
4. Đường đi ngắn nhất
5. Bài tập



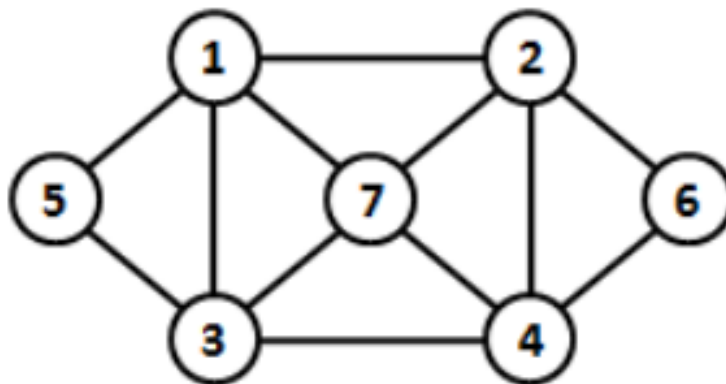
Phần 1

Khái niệm đồ thị

Khái niệm đồ thị



- Đồ thị = Sự trừu tượng hóa các đối tượng và các mối liên hệ giữa chúng trong thực tế
 - Đường đi giữa các thành phố
 - Đường nối mạng giữa các thiết bị kết nối
 - Đường điện trong khu vực
 - Mối quan hệ giữa các cá nhân trên mạng xã hội
- Các đối tượng = các đỉnh
- Các mối quan hệ, kết nối = các cạnh (cung)



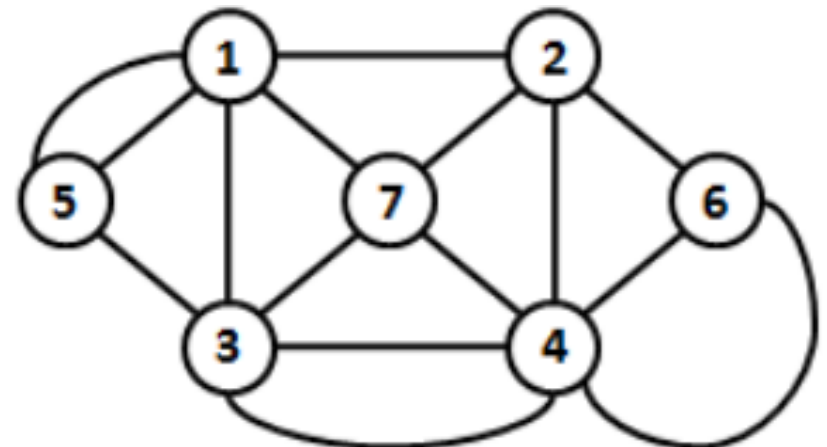
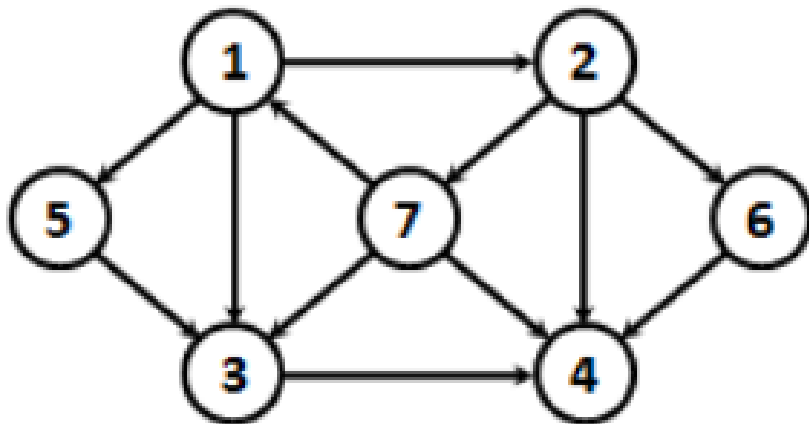


- Đồ thị = các đỉnh + các cung nối giữa chúng
 - $G = (V, E)$
 - $G = \text{Graph}$ (đồ thị)
 - $V = \text{Vertices}$ (đỉnh)
 - $E = \text{Edges}$ (cung)
- Tập V : tập các đỉnh, thường đánh số từ 1 đến n (hoặc từ 0 đến $n-1$)
- Tập E : tập các cung nối giữa hai đỉnh, một cung là một cặp (u, v) , có thể $u = v$
- Đồ thị có hướng: cung (u, v) và cung (v, u) không có mối liên hệ gì đặc biệt (thường nói gọi tắt là đồ thị)

Khái niệm đồ thị



- Đa đồ thị: giữa các cặp (u, v) có thể có nhiều hơn 1 cung nối chúng
- Đơn đồ thị: giữa các cặp (u, v) chỉ có tối đa 1 cung
- Đồ thị vô hướng: cung (u, v) và cung (v, u) là một, không phân biệt
 - Trường hợp này người ta dùng từ cạnh (u, v) để chỉ ý nghĩa (u, v) và (v, u) là tương đương



Độ đo về đỉnh, cung, cạnh



- Nếu có cạnh (u, v) thì hai đỉnh u và v được gọi là kề nhau (đỉnh liền kề)
- Cạnh $e = (u, v)$ gọi là liên thuộc hay phụ thuộc đỉnh u (và cả đỉnh v , đương nhiên)
- Bậc của đỉnh $v = \text{deg}(v) =$ số cạnh phụ thuộc vào $v =$ số đỉnh liền kề với v
 - Trong đồ thị vô hướng: số đỉnh bậc lẻ luôn chẵn
- Cung $e = (u, v)$: e gọi là cung ra khỏi u (và là cung đi vào v)
- Số cung ra của v là $\text{deg}^+(v)$, số cung vào v là $\text{deg}^-(v)$
 - Tổng các deg^+ và deg^- luôn bằng nhau (và bằng số cung)
- Cung (u, v) có thể có trọng số, khi đó G là đồ thị trọng số

Đường đi và chu trình



- Đường đi từ u đến v = bắt đầu từ u liên tiếp di chuyển qua các đỉnh kề để đến v
- Đường đi không tự cắt từ u đến v = quá trình di chuyển từ u đến v không thăm lại một đỉnh đã đi qua (thường nói về đường đi ta nói về đường đi không tự cắt)
- Chu trình = đường đi từ u trở về chính nó
- Một đường đi (chu trình) được gọi là đơn giản nếu nó không chứa những cạnh (cung) lặp
- Một đường đi (chu trình) được gọi là căn bản nếu nó không chứa những đỉnh lặp



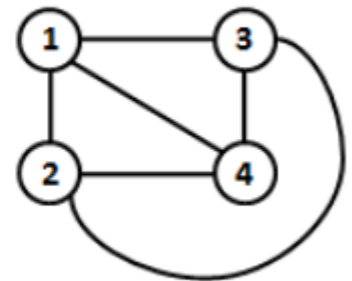
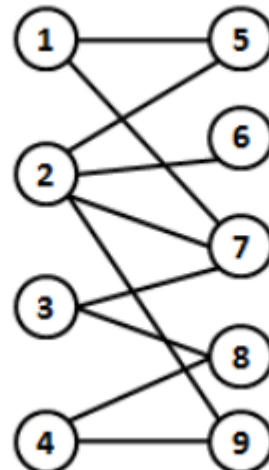
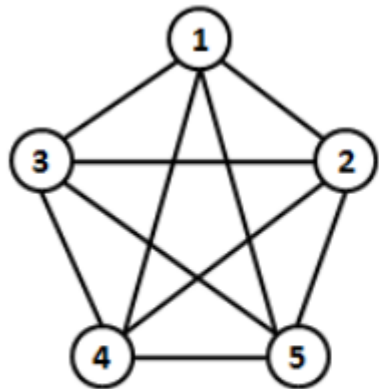
- Đồ thị vô hướng G : là đồ thị liên thông (connected graph) nếu mọi cặp đỉnh đều có đường đi đến nhau
- Đồ thị G : là đồ thị liên thông mạnh (strongly connected graph) nếu mọi cặp đỉnh đều có đường đi đến nhau
- Đồ thị G : là đồ thị liên thông yếu (weakly connected graph) nếu khi chuyển về vô hướng nó là đồ thị liên thông
- Đồ thị vô hướng G : là đồ thị đầy đủ (completed graph) nếu mọi cặp đỉnh kề nhau



- Đồ thị vô hướng G không liên thông có thể chia thành các đồ thị con liên thông, những đồ thị con này gọi là các thành phần liên thông (components)
- Một cạnh e được gọi là cầu nếu loại bỏ e khỏi G làm tăng số lượng thành phần liên thông của G
- Một đỉnh v được gọi là điểm khớp nếu loại bỏ nó khỏi G làm tăng số lượng thành phần liên thông của G

Một số loại đồ thị đặc biệt

- Đồ thị hoàn thiện (complete graphs) = đồ thị vô hướng và mọi cặp đỉnh đều có đường đi trực tiếp tới nhau
- Đồ thị hai phía (bipartite graphs) = đồ thị vô hướng tồn tại một phép chia các đỉnh thành 2 tập không có kết nối nội bộ nhưng có kết nối lẫn nhau
- Đồ thị phẳng (planar graphs) = đồ thị có thể được vẽ trên một mặt phẳng sao cho các cạnh chỉ giao với nhau tại các đỉnh chung





Phần 2

Biểu diễn đồ thị trong máy tính

Biểu diễn đồ thị trong máy tính

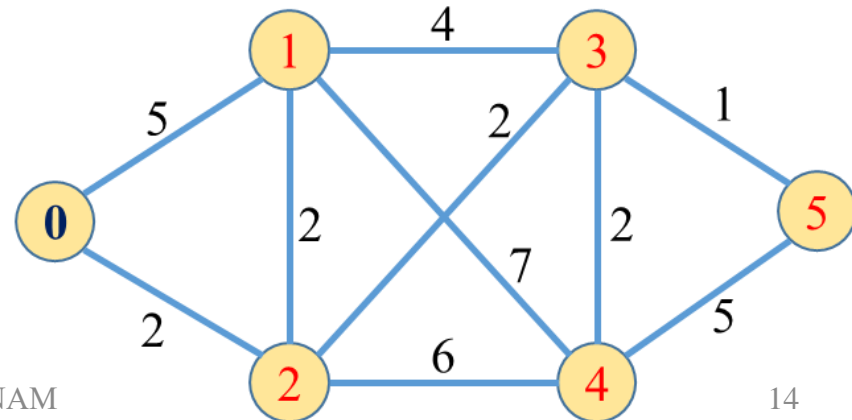


- Đánh số thứ tự các đỉnh: từ 1 đến n
 - Hoặc đánh số từ 0 đến $n-1$, tùy vào mục đích lập trình
 - Hầu như không có nhu cầu đặt tên đỉnh
 - Nhưng vài bài toán trong thực tế dữ liệu ở đỉnh khá quan trọng
- Như vậy dữ liệu về đỉnh chỉ có 1 biến (số lượng đỉnh)
- Biểu diễn dữ liệu về kết nối quan trọng hơn rất nhiều
- Trường hợp chỉ quan tâm đến kết nối:
 - Ma trận kề: ma trận 2 chiều (hoặc vector of vector), ô $[u][v]$ giá trị 0/1 (false/true) xác định cung (u, v) có kết nối hay không
 - Ma trận liên thuộc (incidence matrix): $[u][v] = -1$ nếu từ đỉnh u đi đến đỉnh v , $[u][v] = 1$ nếu từ đỉnh v đi đến đỉnh u , $[u][v] = 0$ nếu không có kết nối

Biểu diễn đồ thị trong máy tính



- Trường hợp chỉ quan tâm đến kết nối:
 - Danh sách kề: vector of vector, mỗi thành phần là một vector các đỉnh kề
 - Danh sách cung: vector chứa các cung (cặp đỉnh)
- Trường hợp quan tâm đến trọng số kết nối:
 - Ma trận trọng số: ma trận 2 chiều (hoặc vector of vector), $a[u][v]$ là trọng số của cung (u,v)
 - Ma trận trọng số của đồ thị vô hướng: $a[u][v] = a[v][u]$
 - Danh sách cung: vector chứa các cung, bao gồm cặp đỉnh và trọng số của cung





Phần 3

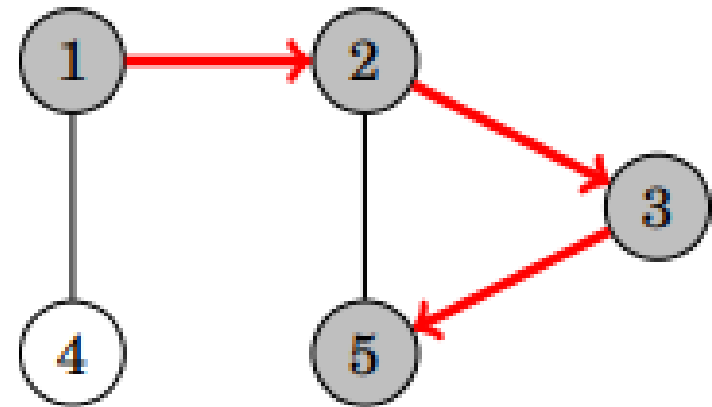
Duyệt đồ thị

Duyệt theo chiều sâu (dfs)



```
// ma trận kề  
bool g[100][100];  
// đánh dấu đã duyệt chưa, ban đầu đặt bằng false hết  
bool v[100];
```

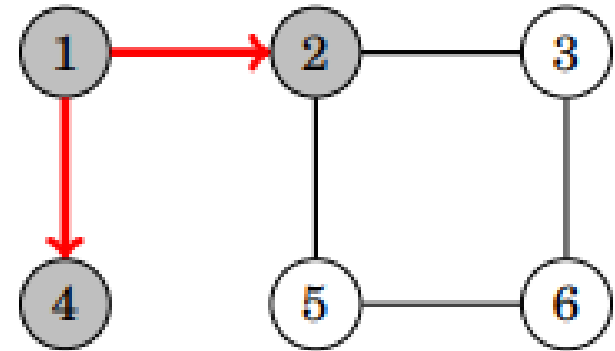
```
void dfs(int s) {  
    // đánh dấu đỉnh s đã được xử lý  
    v[s] = true;  
    // xử lý đỉnh s  
    dosmt(s);  
    // duyệt các đỉnh con  
    for (int i = 1; i <= n; i++)  
        if (g[s][i] && !v[i]) dfs(i);  
}
```



Duyệt theo chiều rộng (bfs)

```
bool g[100][100];    // ma trận kề
bool v[100];         // đánh dấu đã duyệt chưa
queue<int> q;         // lưu lại các đỉnh đã thăm
```

```
void bfs(int s) {
    v[s] = true;
    q.push(s);
    while (!q.empty()) {
        // lấy một đỉnh ra khỏi queue
        int s = q.front(); q.pop();
        // xử lý đỉnh s
        dosmt(s);
        // đẩy các đỉnh kề vào queue
        for (int i = 1; i <= n; i++)
            if (g[s][i] && !v[i]) {
                v[i] = true; q.push(i);
            }
    }
}
```



Ứng dụng của DFS và BFS



- DFS và BFS: tính toán những thành phần liên thông của một đồ thị cho trước
- BFS và DFS: Phát hiện chu trình trong một đồ thị vô hướng
- DFS: tính toán những thành phần liên thông mạnh của một đồ thị có hướng cho trước
- DFS: tính toán những cầu và điểm khớp của một đồ thị vô hướng liên thông
- DFS: sắp xếp topo trên một đồ thị có hướng không chu trình (DAG)

Ứng dụng của DFS và BFS



- BFS và DFS: Tìm đường đi dài nhất trên một cây
- BFS: Tìm đường đi ngắn nhất (chiều dài của một đường đi được định nghĩa là số lượng cạnh của đường đi đó)
- BFS: Kiểm tra liệu một đồ thị cho trước có là đồ thị hai phía không



Phần 4

Đường đi ngắn nhất

Thuật toán dijkstra

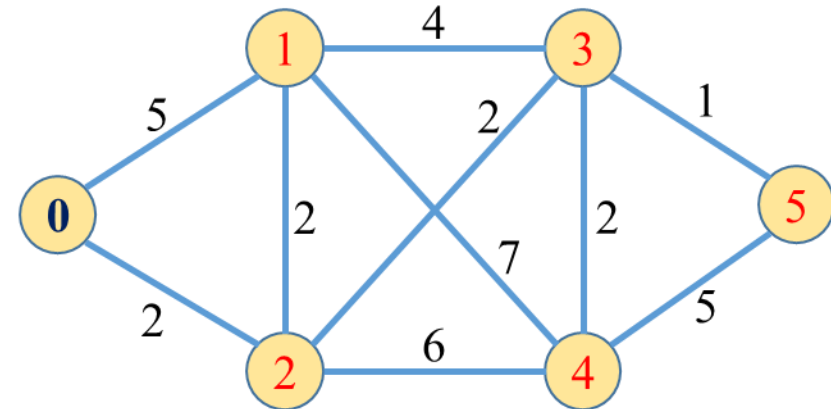


- Bài toán: Đồ thị G , có n đỉnh, có hướng. Không có cung nào có trọng số âm. Tìm đường đi ngắn nhất từ 0 đến 5.

- Ý tưởng:

- Giả thiết đường đi ngắn nhất là đường đi trực tiếp từ 0 đến các đỉnh còn lại:

- $0 \Rightarrow 1: 5$
- $0 \Rightarrow 2: 2$
- $0 \Rightarrow 3: \text{INF}$
- $0 \Rightarrow 4: \text{INF}$
- $0 \Rightarrow 5: \text{INF}$



- Thuật toán sẽ cố gắng giảm nhỏ các giá trị này xuống bằng cách đi vòng qua các đỉnh khác.

Thuật toán dijkstra



Đầu vào:

- Mảng $A[n][n]$ là trọng số các cung
- Tìm đường đi từ P đến Q

Dữ liệu dùng trong quá trình tính toán:

- Mảng $B[n]$: $B[I]$ lưu độ dài đường đi từ P đến I (bản chất là ta tìm $B[Q]$)
- Mảng $C[n]$: đánh dấu xem đỉnh I đã cố định chưa, đỉnh I cố định \sim đã tìm được đường đi min từ P đến I

Thuật toán dijkstra



Khởi tạo ban đầu (có thể có nhiều cách):

- Tất cả $B[i] = +\infty$, riêng $B[P] = 0$
- Cố định P ($C[P] = \text{true}$)
- Đỉnh cố định hiện tại $k = P$

Lặp đến khi cố định được Q ($C[Q]=\text{true}$):

- Thử đi vòng qua đỉnh k: duyệt mọi $B[i]$, nếu $B[i] > B[k]+A[k][i]$ thì cập nhật $B[i] = B[k]+A[k][i]$
- Chọn đỉnh cố định mới: $k = Q$
 - + Duyệt mọi $C[i]$, nếu $C[i]=\text{false}$ và $B[i] < B[k]$ thì cập nhật $k = i$
 - + $C[k] = \text{true}$

Thuật toán Floyd–Warshall



- Bài toán: Đồ thị G , có n đỉnh, có hướng. Không tồn tại chu trình nào có tổng trọng số âm. Tìm đường đi ngắn nhất của mọi cặp đỉnh (p, q)
- Ý tưởng: Nếu từ đường đi hiện tại từ p đến q không tốt bằng đi vòng qua đỉnh k (tức là đi từ p đến k rồi đi từ k đến q) thì ta thay thế đường đi hiện tại bằng đường đi vòng qua k

Thuật toán Floyd–Warshall



Đầu vào:

- Mảng $A[n][n]$ là trọng số các cung

Dữ liệu dùng trong quá trình tính toán:

- Mảng $B[n][n]$: ma trận đường đi ngắn nhất
- Mảng $C[n][n]$: ma trận đỉnh trung gian

Khởi tạo ban đầu (có thể có nhiều cách):

```
for (int i = 1; i <= n; i++)  
    for (int j = 1; j <= n; j++) {  
        B[i][j] = A[i][j];  
        C[i][j] = j;  
    }
```

Thuật toán Floyd–Warshall



Tối thiểu hóa ma trận đường đi (ma trận B):

```
for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (B[i][j] > B[i][k] + B[k][j]) {
                B[i][j] = B[i][k] + B[k][j];
                C[i][j] = C[i][k];
            }
```

Thuật toán Floyd–Warshall



In đường đi từ p đến q:

```
if (B[p][q] == INF) cout << "Không có đường đi từ p đến q";
else {
    cout << "Đường đi có độ dài: " << B[p][q] << endl;
    while (p != q) {
        cout << p << "-->";
        p = C[p][q];
    }
    cout << q << endl;
}
```



Phần 5

Bài tập



1. Một bản đồ trò chơi dạng lưới cỡ $M \times N$, một vài ô bị đánh dấu là bẫy không thể bước vào. Lưới đánh số các dòng từ 1 đến M và các cột từ 1 đến N . Nhân vật của bạn đứng ở ô (p, q) và chỉ có thể di chuyển sang các ô chung cạnh. Nếu có thể di chuyển tới một ô ở mép lưới thì nhân vật có thể thoát khỏi bản đồ. Hãy trả lời xem nhân vật của bạn có rời khỏi bản đồ được không?

INPUT:

- Dòng 1: 4 số nguyên M, N, P, Q
- M dòng tiếp theo: mỗi dòng là một string N kí tự, kí tự X đại diện cho ô có bẫy, kí tự "." đại diện cho ô bình thường.

OUTPUT: in ra YES nếu có thể thoát khỏi bản đồ, in ra NO nếu ngược lại.

2. Một khu hầm hình chữ nhật gồm $N \times M$ căn hầm. Các căn hầm được đánh số từ dòng từ 1 đến n theo chiều từ trên xuống dưới, đánh số cột từ 1 đến m theo chiều từ trái qua phải. Giữa hai căn hầm sát nhau có cửa thông nhau mà phải mất một thời gian nào đó mới có thể mở cửa để đi từ căn hầm này sang căn hầm kia. Sau khi bắt cóc công chúa, mục phù thủy giam nàng tại căn hầm cuối cùng $[n, m]$ - dòng n cột m . Chàng thợ sửa ống nước Super Mario đang ở căn hầm đầu tiên $[1, 1]$. Bạn hãy tìm các căn hầm mà Mario phải đi qua để giải cứu công chúa một cách nhanh nhất.



	1	1	1
9	9	9	1
	1	1	1
1	9	9	9
	1	1	1
			

Bài tập



INPUT:

- Dòng thứ nhất là hai số nguyên n và m ($1 \leq n, m \leq 700$)
- N dòng tiếp theo: mỗi dòng gồm $m-1$ số nguyên $a[i][j]$ là khoảng thời gian để mở cửa từ hầm $[i, j]$ sang hầm $[i, j+1]$
- $N-1$ dòng tiếp theo: mỗi dòng gồm m số nguyên $b[i][j]$ là khoảng thời gian để mở cửa từ hầm $[i, j]$ sang hầm $[i+1, j]$

	1	1	1
9	9	9	1
1	9	9	9
1	1	1	

OUTPUT: số nguyên xác định khoảng thời gian sớm nhất mà Mario có thể giải cứu công chúa.

3. Hãy chỉ ra cách dịch chuyển để từ hình A sang B hoặc thông báo không có cách dịch chuyển

