

CHƯƠNG 2: BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

Contents

2.1 Phương pháp ma trận kề	26
2.1.1 Ma trận liên thuộc.....	28
2.1.2 Ma trận trọng số.....	28
2.1.3 Các đặc điểm của phương pháp ma trận kề.....	29
2.1.4 Ưu & nhược điểm của ma trận kề	29
2.2 Phương pháp danh sách kề.....	29
2.2.1 Biểu diễn danh sách kề dựa vào mảng	30
2.2.2 Biểu diễn danh sách kề bằng danh sách liên kết	31
2.2.3 Storage of adjacency list.....	32
2.2.4 Ưu & nhược của danh sách kề.....	33
2.2.5 Danh sách kề so với Ma trận kề	33
2.3 Phương pháp danh sách cạnh/ cung	33
2.3.1 Danh sách cạnh của đồ thị vô hướng.....	33
2.3.2 Danh sách cung của đồ thị có hướng.....	34
2.3.3 02 cách cài đặt danh sách cạnh/ cung.....	34
2.3.4 Ưu & nhược của danh sách cạnh/ cung	34
2.4 So sánh các phương pháp biểu diễn.....	35
2.5 Bài tập & Code cài đặt.....	35
2.5.1 Ma trận kề.....	35
2.5.2 Danh sách kề & Danh sách cạnh	42

Trong cuộc sống có nhiều cách để biểu diễn đồ thị: Lời nói, câu văn, hình vẽ... Tuy nhiên các cách này chỉ đủ để con người hiểu, còn máy tính thì không (Nếu cho văn bản mô tả đồ thị thì máy tính chưa thể biểu diễn đồ thị bằng hình vẽ, và ngược lại).

Như vậy để máy tính có thể hiểu được đồ thị, thì ta cần biểu diễn bằng các CẤU TRÚC DỮ LIỆU, và cần vận dụng các cấu trúc đó vào việc viết mã một cách TỔNG QUÁT.

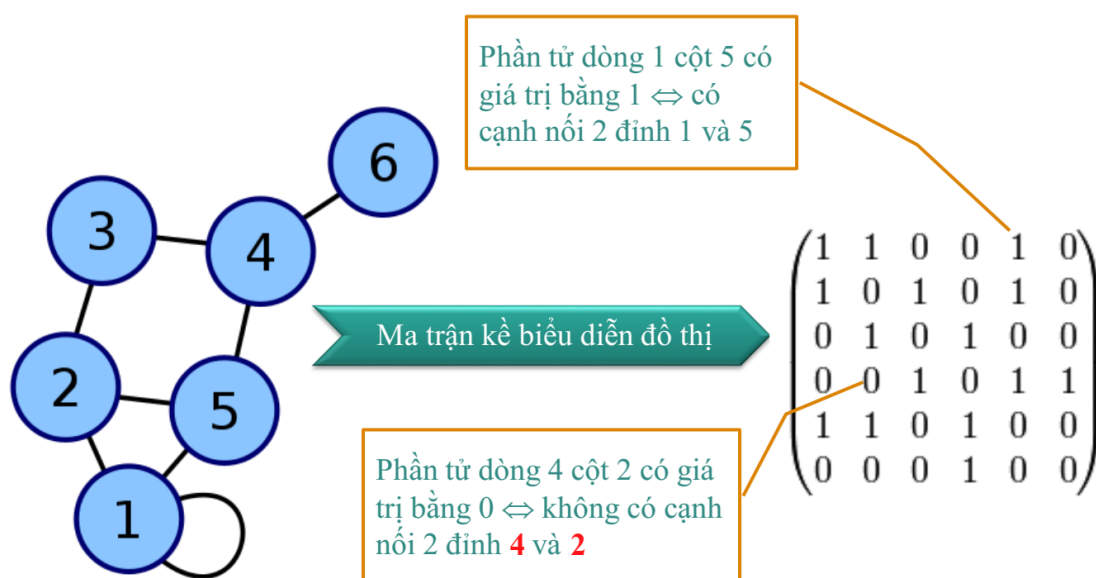
2.1 Phương pháp ma trận kề

- Xét đồ thị $G = (V, E)$, giả sử tập V gồm n đỉnh và được sắp thứ tự $V = v_1, v_2, \dots, v_n$, tập E gồm m cạnh và được sắp thứ tự $E = e_1, e_2, \dots, e_m$.

- Ma trận kề (*adjacency matrix*) A (hoặc A_G) của G , đối với danh sách các đỉnh này, là ma trận $n \times n$, trong đó **đỉnh v_i được biểu thị bằng dòng i và cột i và phần tử a_{ij} biểu thị số cạnh giữa v_i và v_j .**

- Nếu G là đơn đồ thị, ma trận kề A chứa: 1 là phần tử thứ (i, j) của ma trận khi v_i và v_j kề nhau, và 0 là ngược lại. Nói cách khác, ma trận kề của nó là $A = [a_{ij}]$ thì:

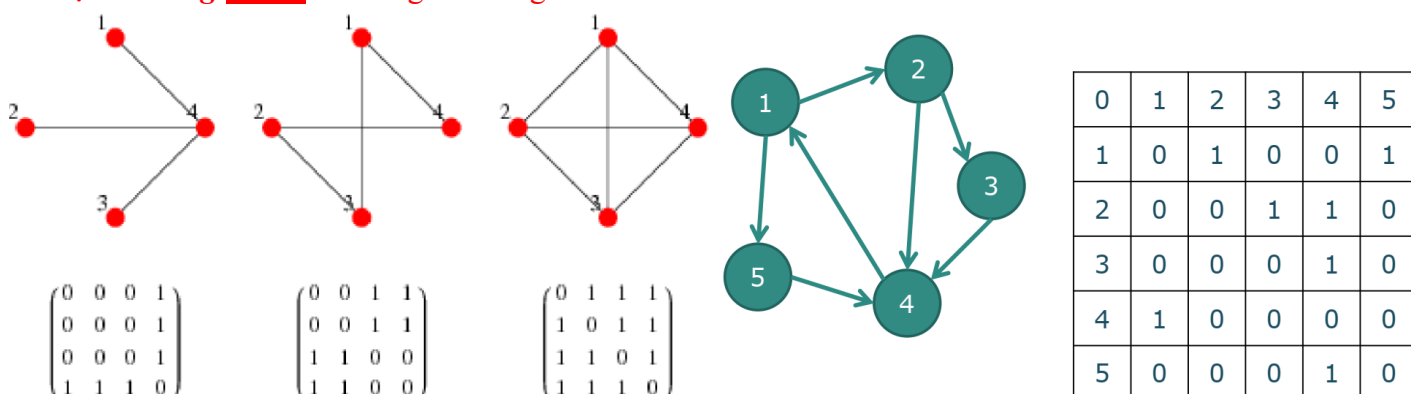
$$a_{ij} = \begin{cases} 1 & \text{nếu } \{v_i, v_j\} \text{ là một cạnh của } G \\ 0 & \text{ngược lại} \end{cases}$$



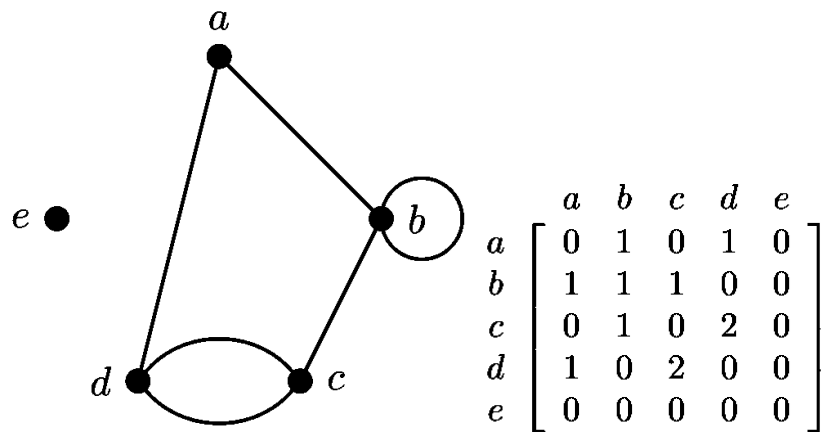
- Lưu ý rằng ma trận kề của đồ thị dựa trên thứ tự được chọn cho các đỉnh. Do đó, có thể có tới $n!$ các ma trận kề khác nhau cho đồ thị có n đỉnh, vì có $n!$ thứ tự khác nhau của n đỉnh.

- Một đơn đồ thị không có vòng lặp nên mỗi phần tử a_{ii} , $i = 1, 2, 3, \dots, n$, là 0 \rightarrow Trong trường hợp đồ thị đơn hữu hạn, ma trận kề là một ma trận $(0,1)$ với các giá trị 0 nằm trên đường chéo chính.

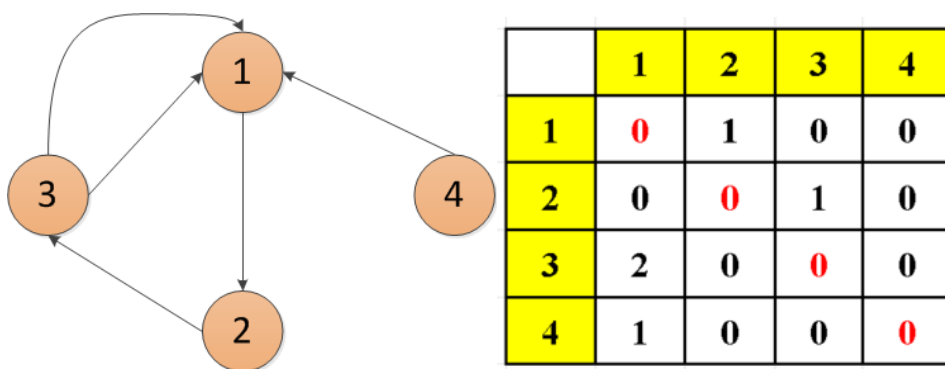
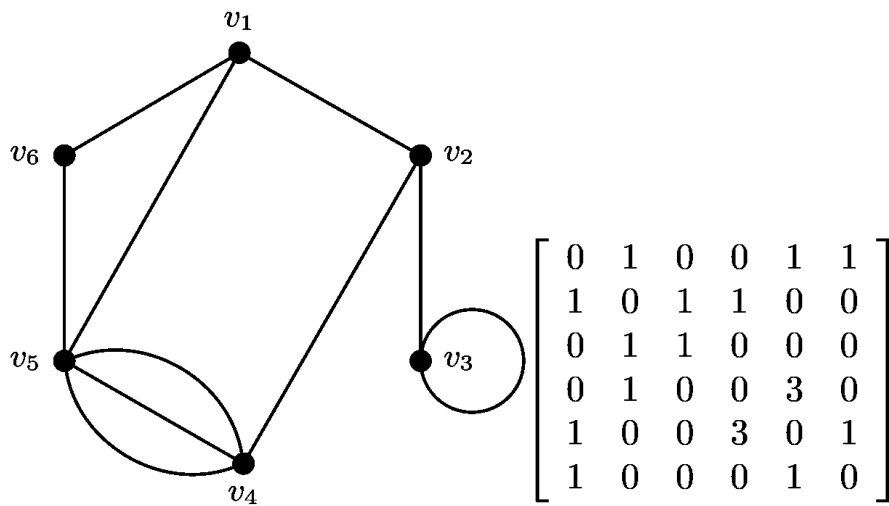
- Nếu đồ thị là vô hướng, ma trận kề là ma trận đối xứng (qua đường chéo chính), $a[i,j] = a[j,i]$. Ma trận kề của đồ thị có hướng có thể là không đối xứng.



- Khi có nhiều cạnh nối cùng một cặp đỉnh v_i và v_j , hoặc có nhiều vòng lặp ở cùng một đỉnh, thì ma trận kề không còn là ma trận 0-1, vì phần tử thứ (i, j) của ma trận này bằng với số cạnh liên kết với $\{v_i, v_j\}$.
- Tất cả các đồ thị **vô hướng**, bao gồm **đa đồ thị** và **giả đồ thị**, đều có ma trận kề **đối xứng**.
- Ma trận kề cũng có thể được sử dụng để biểu diễn đồ thị vô hướng có vòng và có nhiều cạnh. Vòng lặp tại đỉnh v_i được biểu thị bằng số 1 tại vị trí thứ (i, i) của ma trận kề.



Lưu ý rằng phần tử $(2; 2)$ biểu thị vòng lặp tại b và các phần tử $(3; 4)$ và $(4; 3)$ chỉ ra rằng có hai cạnh giữa c và d. Cột & hàng e có toàn số 0 vì e là đỉnh cô lập.

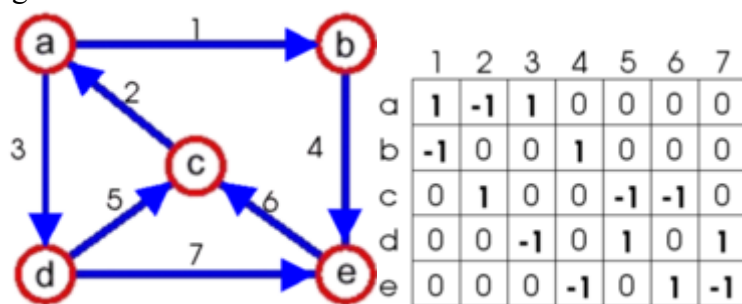


- Đối với đồ thị thưa (ma trận thưa), nghĩa là đồ thị có ít cạnh, người ta thường chọn dùng danh sách kề hơn do nó chiếm ít bộ nhớ hơn. Ma trận liên thuộc là một biểu diễn ma trận khác cho đồ thị.

2.1.1 Ma trận liên thuộc

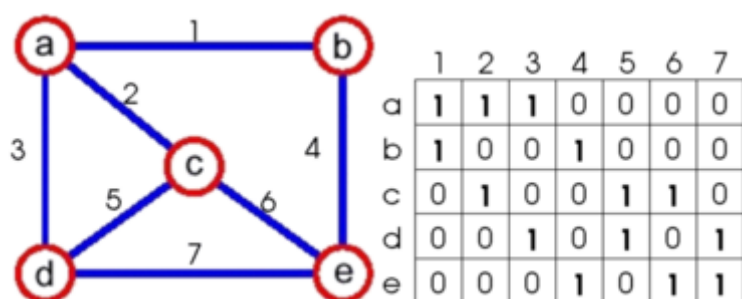
- **Có hướng:** Nếu G là đồ thị có hướng *không có khuyên*, ma trận liên thuộc (hay liên kết đỉnh cạnh) của đồ thị G , ký hiệu $A(G)$, là ma trận $n*m$ (n : số đỉnh, m : số cạnh) được định nghĩa là $A = (A_{ij})$ với quy ước:

- * $A_{ij} = 1$ nếu cạnh j hướng ra khỏi đỉnh i
- * $A_{ij} = -1$ nếu cạnh j hướng vào đỉnh i .
- * $A_{ij} = 0$ nếu cạnh j không kề đỉnh i .



- **Vô hướng:** Nếu G là đồ thị vô hướng *không có khuyên*, ma trận liên thuộc (hay liên kết đỉnh cạnh) của đồ thị G , ký hiệu $A(G)$, là ma trận $n*m$ (n : số đỉnh, m : số cạnh) được định nghĩa là $A = (A_{ij})$ với quy ước:

- * $A_{ij} = 1$ nếu đỉnh i kề với cạnh j .
- * $A_{ij} = 0$ nếu ngược lại.

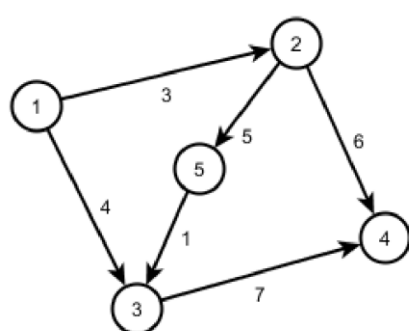


2.1.2 Ma trận trọng số

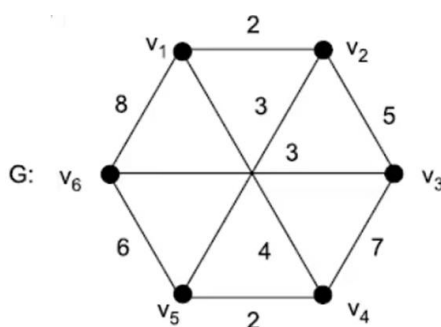
- Đồ thị có trọng số $G(V, E, W)$ là đồ thị mà mỗi cạnh/ cung (u, v) được gán tương ứng với một số thực, kí hiệu là $w(u, v)$ hoặc $c(u, v)$.

- Ma trận kề biểu diễn đồ thị có trọng số gọi là **ma trận trọng số** → **Vẫn giữ các tính chất của ma trận kề.**

- Gọi C là ma trận trọng số của đồ thị G : $C_{ij} = \begin{cases} \text{trọng số, nếu } (v_i, v_j) \in E \\ 0 \text{ hoặc } \infty \text{ hoặc } -\infty \text{ (tùy theo quy ước), nếu } (v_i, v_j) \notin E \end{cases}$



∞	3	4	∞	∞
∞	∞	∞	6	5
∞	∞	∞	7	∞
∞	∞	∞	∞	∞
∞	∞	1	∞	∞



$$A(G) = \begin{pmatrix} 0 & 2 & 0 & 4 & 0 & 8 \\ 2 & 0 & 5 & 0 & 3 & 0 \\ 0 & 5 & 0 & 7 & 0 & 3 \\ 4 & 0 & 7 & 0 & 2 & 0 \\ 0 & 3 & 0 & 2 & 0 & 6 \\ 8 & 0 & 3 & 0 & 6 & 0 \end{pmatrix}$$

Tuy nhiên nếu dùng số 0 trong trường hợp *không tồn tại cạnh/ cung* giữa 2 đỉnh đang xét & một cạnh khác có *trọng số bằng 0* thì sẽ gây ra nhầm lẫn.

Ví dụ:

	0	7	<u>0</u>
	7	0	0
	<u>0</u>	0	0

→ Vì vậy nên dùng ∞ để đảm bảo tính nhất quán.
Ý nghĩa ∞ : Không có cạnh nối = Có cạnh nối nhưng cạnh đó dài vô cực.

2.1.3 Các đặc điểm của phương pháp ma trận kề

+ **Đồ thị vô hướng**

- Ma trận kề đối xứng (qua đường chéo chính).
- Tổng các phần tử trên dòng (cột) của ma trận bằng bậc của đỉnh tương ứng.
- Tổng các phần tử trên ma trận bằng 2 lần số cạnh.

+ **Đồ thị có hướng**

• Tổng các phần tử trên **dòng** của ma trận bằng **bán bậc ra (outdeg)** của đỉnh tương ứng (*cung từ đỉnh đó đến đỉnh kia có không → đi ra*).

• Tổng các phần tử trên **cột** của ma trận bằng **bán bậc vào (indeg)** của đỉnh tương ứng (*cung từ đỉnh kia đến nó có không → đi vào*).

- Tổng các phần tử của ma trận bằng số cạnh.

2.1.4 Ưu & nhược điểm của ma trận kề

• **Ưu điểm**

- Đơn giản, dễ cài đặt trên máy tính
- Sử dụng một mảng hai chiều để biểu diễn ma trận kề
- Dễ dàng **kiểm tra được hai đỉnh u, v có kề với nhau hay không** trong $O(1)$ bằng việc kiểm tra đúng một phép so sánh ($a[u][v] \neq 0$?)

• **Nhược điểm**

- Lãng phí bộ nhớ: bất kể số cạnh nhiều hay ít ta cần **n^2** đơn vị bộ nhớ để biểu diễn
- Không thể biểu diễn được với các đồ thị *có số đỉnh lớn*
- Để xem xét đỉnh u có những đỉnh kề nào cần mất **n** phép so sánh kể cả đỉnh u là đỉnh cô lập hoặc đỉnh treo.

2.2 Phương pháp danh sách kề

- Danh sách kề là danh sách biểu diễn tất cả các **cạnh hoặc cung** trong một đồ thị.

+ Nếu đồ thị vô hướng, mỗi phần tử của danh sách là **một cặp hai đỉnh là hai đầu của cạnh** tương ứng.

+ Nếu đồ thị có hướng, mỗi phần tử là một cặp có thứ tự gồm **hai đỉnh là đỉnh đầu và đỉnh cuối của cung** tương ứng.

- The adjacency list is an array $L[0...n-1]$ of lists, where n is the number of vertices in the graph.

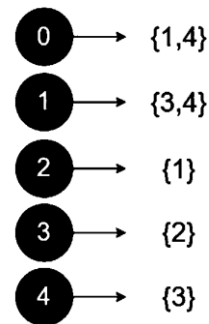
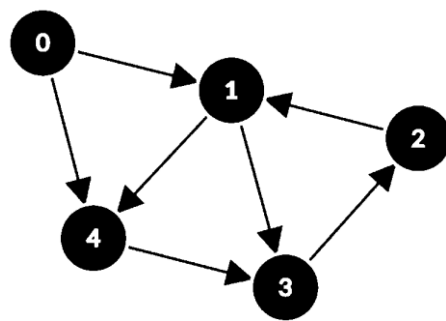
- Each array entry is indexed by the vertex id (as with adjacency matrix)
- The list $L[i]$ stores the ids of the vertices adjacent to i

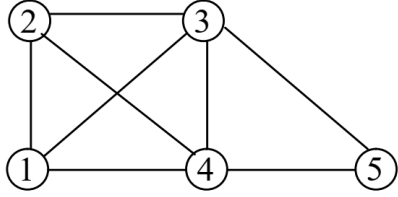
- Tương ứng với mỗi đỉnh v của đồ thị, ta có tương ứng một danh sách để lưu các đỉnh kề với nó.

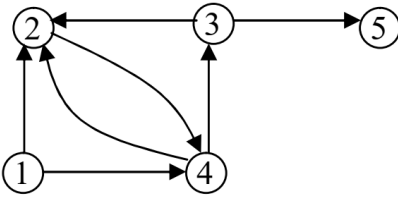
- Thông thường, danh sách kề không coi trọng thứ tự giữa các cạnh.

- Danh sách: *mảng 1 chiều, hoặc danh sách liên kết*.

Ví dụ:



 G ₁	Danh sách	Đỉnh kề
	1	2, 3, 4
	2	1, 3, 4
	3	1, 2, 4, 5
	4	1, 2, 3, 5
	5	3, 4

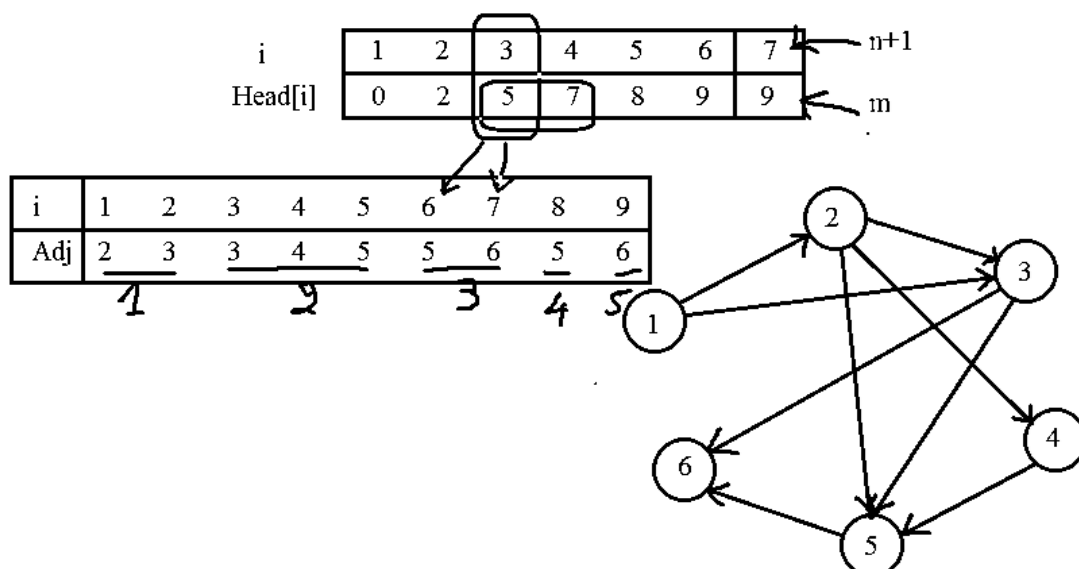
 G ₂	Danh sách	Đỉnh kề
	1	2, 4
	2	4
	3	2, 5
	4	2, 3
	5	∅

2.2.1 Biểu diễn danh sách kề dựa vào mảng

- Mảng được chia thành n đoạn. Để biết một đoạn thuộc mảng bắt đầu từ phần tử nào đến phần tử nào ta sử dụng một **mảng khác** dùng để lưu trữ vị trí các phần tử bắt đầu và kết thúc của đoạn.
- Có thể dùng một mảng các vector: `vector<int> adj[1001];`
- Hoặc dùng một vector các vector: `vector<vector<int>> adj;`

Cách 1:

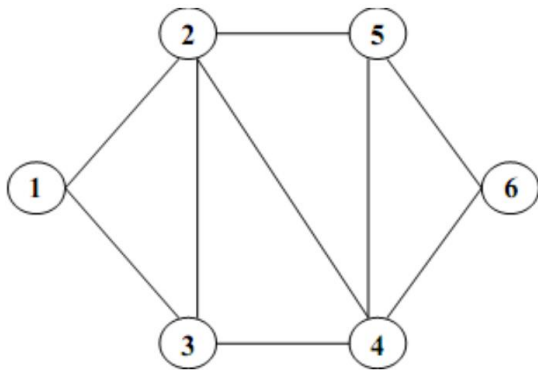
Ví dụ 1: <https://kienthuc24h.com/danh-sach-ke-c-ly-thuyet-do-thi/>



Gọi $Head[i]$ là chỉ số kết thúc của đoạn quản lý đỉnh kề của $i-1$. Hay $Head[i]+1$ là chỉ số bắt đầu của đoạn quản lý đỉnh kề của i .

Như vậy, chỉ số của đoạn chứa các đỉnh kề của i là từ **$Head[i]+1$ đến $Head[i+1]$**

Ví dụ 2:



$$Ke(1) = \{ 2, 3 \}.$$

$$Ke(2) = \{ 1, 3, 4, 5 \}.$$

$$Ke(3) = \{ 1, 2, 4 \}.$$

$$Ke(4) = \{ 2, 3, 5, 6 \}.$$

$$Ke(5) = \{ 2, 4, 6 \}.$$

$$Ke(6) = \{ 4, 5 \}.$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A[i]=?	2	3	1	3	4	5	1	2	4	2	3	5	6	2	4	6	4	5
	Đoạn 1		Đoạn 2			Đoạn 3			Đoạn 4			Đoạn 5			Đoạn 6			

$$VT[6] = \{ 0, 2, 6, 9, 13, 16, 18 \}.$$

Cách 2:

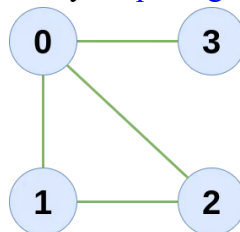
- Tạo mảng 2 chiều, 1 chiều lưu danh sách các đỉnh, chiều còn lại lưu danh sách các đỉnh kề tương ứng.
- Mỗi lần danh sách của đỉnh kề được cập nhật, bậc của đỉnh đó được cập nhật và lưu vào một mảng mới.
- Mỗi danh sách kề được kết thúc bằng -1 để đánh dấu việc kết thúc danh sách kề của một đỉnh.

(Chi tiết xem phần 2.5)

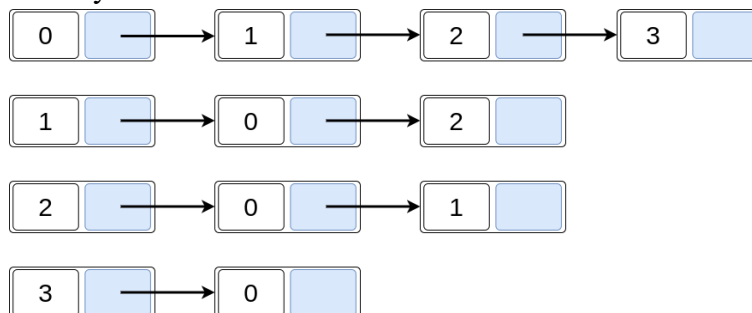
2.2.2 Biểu diễn danh sách kề bằng danh sách liên kết

- Biểu diễn một đồ thị (graph) dưới dạng **một mảng các danh sách liên kết**. Trong đó, chỉ số mảng đại diện cho đỉnh của đồ thị và các phần tử trong danh sách liên kết của đỉnh đó là các đỉnh có kết nối với đỉnh đó.

Ví dụ 1: Lấy ví dụ với một đồ thị vô hướng dưới đây: <https://nguyenvanhieu.vn/danh-sach-ke-adjacency-list/>



Với đồ thị phía trên, chúng ta có thể biểu diễn nó vào bộ nhớ máy tính dưới dạng một mảng các danh sách liên kết như hình vẽ dưới đây:

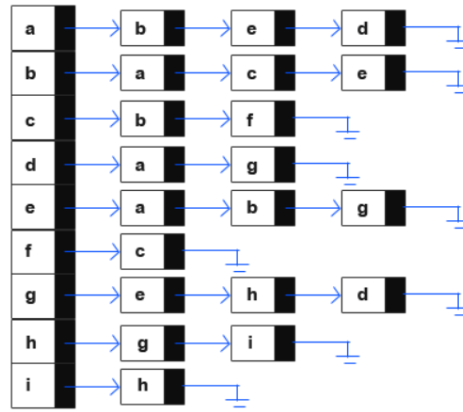
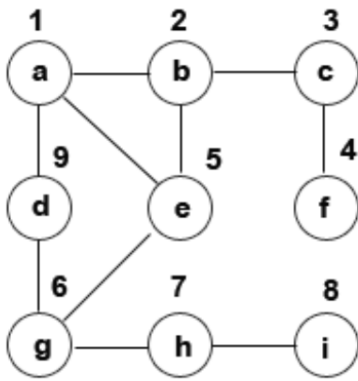


Đồ thị có 4 đỉnh (0, 1, 2 và 3). Do đó, chúng ta cũng sẽ có 4 danh sách liên kết cho 4 đỉnh đó. Trong mỗi danh sách liên kết là các Node đại diện cho các đỉnh có liên kết với Node head.

- Danh sách liên kết cho **đỉnh 0 (head)** có 3 Node phía sau lần lượt là (1, 2 và 3) thể hiện rằng các cặp đỉnh (0, 1), (0,2) và (0, 3) có kết nối.
- Tương tự, danh sách liên kết của **đỉnh 2 (head)** có 2 Node phía sau lần lượt là (0 và 2) thể hiện rằng các cặp đỉnh (2, 0) và (2,1) có kết nối.

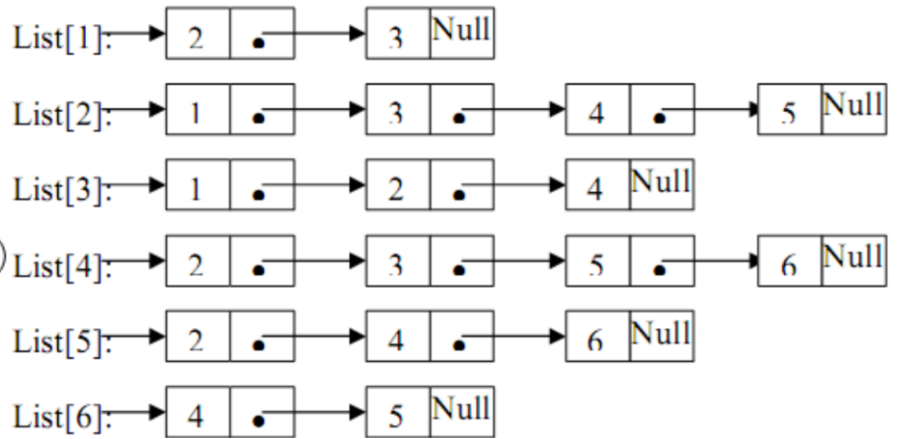
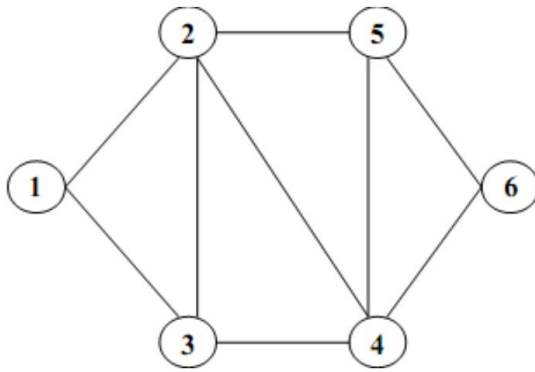
• Tránh nhầm lẫn đỉnh 0 kề đỉnh 1, đỉnh 1 kề đỉnh 2, đỉnh 2 kề đỉnh 3... khi nhìn vào DSLK.

Ví dụ 2:

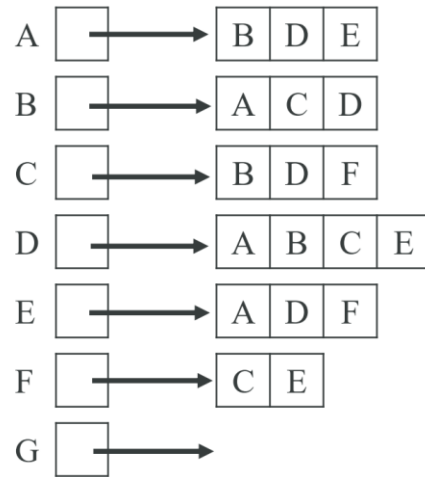
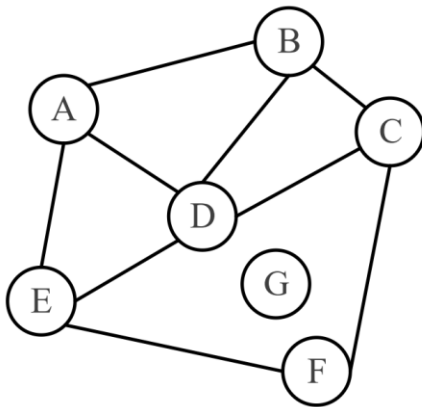


Ví dụ 3:

• Với mỗi đỉnh $u \in V$, ta biểu diễn mỗi danh sách kề của đỉnh bằng một danh sách liên kết List(u).



Ví dụ 4:



2.2.3 Storage of adjacency list

• The array takes up $O(n)$ space.

• Define degree of v , $\deg(v)$, to be the number of edges incident to v . Then, the total space to store the graph is proportional to: $\sum_{\text{vertex } v} \deg(v)$

• An edge $e = \{u, v\}$ of the graph contributes (góp) a count of 1 to $\deg(u)$ and contributes a count 1 to $\deg(v)$.

• Therefore, $\sum_{\text{vertex } v} \deg(v) = 2m$, where m is the total number of edges

• In all, the adjacency list takes up $O(n + m)$ space.

• If $m = O(n^2)$, both adjacent matrix and adjacent lists use $O(n^2)$ space.

• If $m = O(n)$, adjacent list outperforms (vượt trội hơn) adjacent matrix

. However, one cannot tell in $O(1)$ time whether two vertices are connected.

2.2.4 Ưu & nhược của danh sách kề

• Ưu điểm:

- Dễ dàng duyệt tất cả các đỉnh của một danh sách kề;
- Dễ dàng duyệt các cạnh của đồ thị trong mỗi danh sách kề;
- Tối ưu về phương pháp biểu diễn.

• Nhược điểm: Khó khăn cho người đọc có kỹ năng lập trình yếu.

2.2.5 Danh sách kề so với Ma trận kề

+ Danh sách lân cận

- . Nhỏ gọn hơn ma trận kề nếu đồ thị có ít cạnh.
- . Cần nhiều thời gian hơn để tìm xem một cạnh có tồn tại không.

+ Ma trận kề

- . Luôn yêu cầu không gian $n^2 \rightarrow$ Có thể lãng phí rất nhiều không gian nếu số cạnh thưa thớt.
- . Có thể nhanh chóng tìm thấy nếu một cạnh tồn tại.

2.3 Phương pháp danh sách cạnh/ cung

- Trong trường hợp đồ thị thưa (đồ thị có số cạnh $m < 6n$), người ta thường biểu diễn đồ thị dưới dạng danh sách cạnh bằng cách liệt kê tất cả các cạnh của đồ thị trong một danh sách, mỗi phần tử của danh sách là một cặp (u, v) tương ứng với một cạnh của đồ thị.

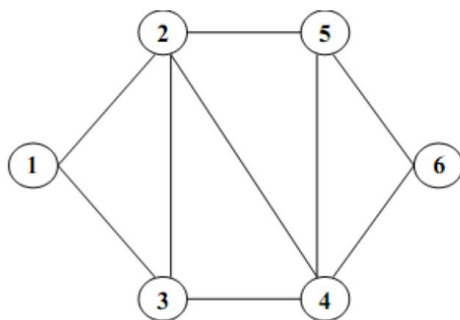
(Trong trường hợp đồ thị có hướng thì mỗi cặp (u, v) tương ứng với một cung, u là đỉnh đầu và v là đỉnh cuối của cung).

- Danh sách được lưu trong bộ nhớ dưới dạng mảng hoặc danh sách liên kết.

2.3.1 Danh sách cạnh của đồ thị vô hướng

- + Chỉ cần liệt kê các cạnh (u, v) mà không cần liệt kê cạnh (v, u) .
- + Nên liệt kê các cạnh theo thứ tự tăng dần của đỉnh đầu mỗi cạnh \rightarrow Đỉnh đầu nhỏ hơn đỉnh cuối mỗi cạnh.
- + Số cạnh có giá trị u thuộc cả vế phải và vế trái của danh sách cạnh là bậc của đỉnh u .

Ví dụ 1:



<u>Đỉnh đầu</u>	<u>Đỉnh cuối</u>
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6
5	6

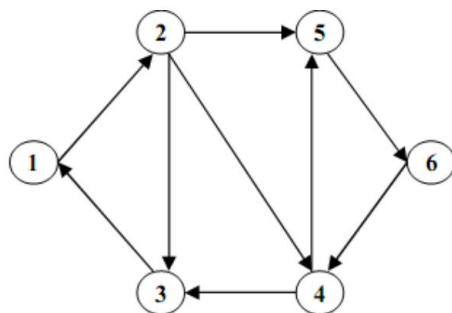
Ví dụ 2:

	<u>Đỉnh đầu</u>	<u>Đỉnh cuối</u>	<u>Trọng số</u>
	1	2	4
	2	3	5
	2	4	-3
	3	4	2

2.3.2 Danh sách cung của đồ thị có hướng

- + Mỗi cạnh là bộ có tính đến thứ tự các đỉnh (Đặc biệt chú ý đến hướng của các cạnh)
- + Đỉnh đầu không nhất thiết phải nhỏ hơn đỉnh cuối mỗi cạnh.
- + Số cạnh có giá trị u thuộc cả về phải các cạnh là $\deg^+(u)$.
- + Số cạnh có giá trị u thuộc cả về trái các cạnh là $\deg^-(u)$.

Ví dụ 1:

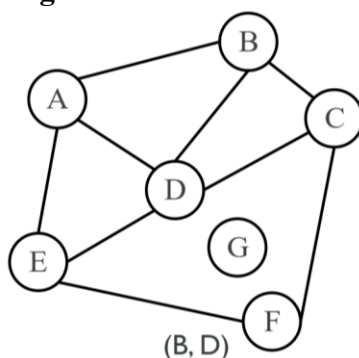


Đỉnh đầu	Đỉnh Cuối
1	2
2	3
2	4
2	5
3	1
4	3
4	5
5	6
6	4

Ví dụ 2:

	Đỉnh đầu	Đỉnh cuối	Trọng số
	2	1	1
	4	2	2
	3	2	3
	3	4	4

2.3.3 02 cách cài đặt danh sách cạnh/ cung



Using Array	0	1	2	3	4	5	6	7	8
	(A, B)	(A, D)	(A, E)	(B, C)	(B, D)	(D, C)	(D, E)	(F, E)	(F, C)



2.3.4 Ưu & nhược của danh sách cạnh/ cung

- Ưu điểm của danh sách cạnh:

- + Trong trường hợp đồ thị thưa ($m < 6n$), biểu diễn bằng danh sách cạnh tiết kiệm được không gian nhớ, bởi nó chỉ cần $2m$ ô nhớ để lưu danh sách cạnh;
- + Thuận lợi cho một số thuật toán chỉ quan tâm đến các cạnh của đồ thị.
- + Trong một số trường hợp, ta phải xét tất cả các cạnh của đồ thị thì cài đặt trên danh sách cạnh làm cho việc duyệt các cạnh dễ dàng hơn. (Thuật toán Kruskal chẳng hạn)

- Nhược điểm của danh sách cạnh:

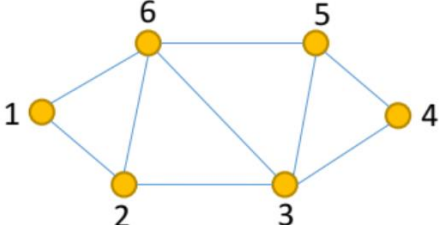
Khi ta cần duyệt tất cả các đỉnh kề với đỉnh v nào đó của đồ thị, thì chẳng có cách nào khác là phải duyệt tất cả các cạnh, lọc ra những cạnh có chứa đỉnh v và xét đỉnh còn lại. Điều đó khá tốn thời gian trong trường hợp đồ thị dày (nhiều cạnh) & làm cho thuật toán có chi phí tính toán cao.

2.4 So sánh các phương pháp biểu diễn

Nội dung so sánh	Ma trận kề	Danh sách kề	Danh sách cạnh / cung
Dung lượng bộ nhớ	$n \times n$	Phụ thuộc số đỉnh kề của mỗi đỉnh	Phụ thuộc số cạnh/cung
Độ phức tạp tính toán	Phụ thuộc vào cấp của ma trận		
Ưu điểm	Cài đặt đơn giản	Hỗ trợ tốt các thuật toán trên đồ thị	Cài đặt đơn giản
Nhược điểm	Không phù hợp với những đồ thị “thưa”, khi đó phần lớn các phần tử của ma trận bằng 0	Dư thừa dữ liệu khi biểu diễn đồ thị vô hướng	Không phù hợp với những đồ thị “màu”, khi đó số cạnh / cung rất lớn

2.5 Bài tập & Code cài đặt

2.5.1 Ma trận kề

	INPUT.INP	Các đỉnh của đồ thị G: 1 2 3 4 5 6
	6 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 0 OUTPUT.OUT 6 2 3 4 2 3 4	Các cạnh của đồ thị G: (1, 2) (1, 6) (2, 3) (2, 6) (3, 4) (3, 5) (3, 6) (4, 5) (5, 6) Bạc của các đỉnh: Đỉnh 1: Bạc = 2 Đỉnh 2: Bạc = 3 Đỉnh 3: Bạc = 4 Đỉnh 4: Bạc = 2 Đỉnh 5: Bạc = 3 Đỉnh 6: Bạc = 4

* Code nhập tay:

```
#include <iostream>
#include <vector>
using namespace std;

//Hàm nhập ma trận kề
void v_input(vector<vector<int>>& adjacencyMatrix, int n) {
    //bản chất ma trận kề chỉ có n phần tử, mỗi phần tử chứa một vector kích thước n
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> adjacencyMatrix[i][j];
        }
    }
}
```

```

//Hàm tính bậc của đồ thị
vector<int>cal_Degree(vector<vector<int>>adjacencyMatrix, int n) {
    //Tạo một vector lưu bậc của các đỉnh, sau khi xong hàm thì trả về vector này -> Kiểu trả
    về là vector
    vector<int>Degree(n, 0);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (adjacencyMatrix[i][j] == 1) Degree[i]++;
        }
    }
    return Degree;
}

//Hàm in bậc của đồ thị
void printDegree(vector<int>Degree, int n) {
    for (int i = 0; i < n; i++) {
        cout << "Đỉnh " << i+1 << ": " << Degree[i] << endl;
    }
}

//Hàm in các cạnh đồ thị = in chỉ số của ma trận nếu phần tử tại đó bằng 1
void printEdge(vector<vector<int>>adjacencyMatrix, int n) {
    vector<vector<int>>temp = adjacencyMatrix; //tạo vector tạm
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (temp[i][j] == 1) {
                if (temp[i][j] == temp[j][i]) { //tránh cạnh trùng do đồ thị vô hướng
                    cout << "(" << i+1 << ", " << j+1 << ")" << endl;
                    temp[j][i] = 0;
                }
            }
        }
    }
}

//Hàm in các đỉnh của đồ thị.
void printVertex(int n) {
    for (int i = 0; i < n; i++) {
        cout << i+1 << " ";
    }
    cout << endl;
}

int main() {
    int n;
    cout << "Nhập số đỉnh: "; cin >> n;
    cout << "=====\n";
    cout << "Nhập ma trận cạnh " << n << "x" << n << ": " << endl;

    vector<vector<int>>adjacencyMatrix(n, vector<int>(n));
    //Tạo một ma trận có n hàng, mỗi hàng lại là một vector có n phần tử → Ma trận vuông NxN
    //Với tất cả phần tử được khởi tạo mặc định bằng 0.
    v_input(adjacencyMatrix, n);

    cout << "=====\n";
    cout << "Danh sách cạnh của đồ thị: \n";
    printEdge(adjacencyMatrix, n);

    cout << "=====\n";
    cout << "Danh sách đỉnh của đồ thị: \n";
}

```

```

    printVertex(n);

    cout << "=====\n";
    cout << "Danh sach bac cua do thi: \n";
    printDegree(cal_Degree(adjacencyMatrix, n), n);

    return 0;
}

```

Kết quả:

Nhap so dinh: 6

=====

Nhap ma tran cap 6x6:

```

0 1 0 0 0 1
1 0 1 0 0 1
0 1 0 1 1 1
0 0 1 0 1 0
0 0 1 1 0 1
1 1 1 0 1 0

```

=====

Danh sach canh cua do thi:

```

(1, 2)
(1, 6)
(2, 3)
(2, 6)
(3, 4)
(3, 5)
(3, 6)
(4, 5)
(5, 6)

```

=====

Danh sach dinh cua do thi:

```

1 2 3 4 5 6

```

=====

Danh sach bac cua do thi:

```

Dinh 1: 2
Dinh 2: 3
Dinh 3: 4
Dinh 4: 2
Dinh 5: 3
Dinh 6: 4

```

**** Code đưa KO vào file***

```

#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

// Hàm tính bậc của các đỉnh trong đồ thị
vector<int> calculateDegrees(const vector<vector<int>>& adjacencyMatrix) {
    int n = adjacencyMatrix.size();
    vector<int> degrees(n, 0);
}

```

```

for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if (adjacencyMatrix[i][j] == 1) {
            degrees[i]++;
        }
    }
}

return degrees;
}

int main() {
    ifstream inFile("E:\\BacDoThiVoHuong.INP");
    ofstream outFile("E:\\BacDoThiVoHuong.OUT");

    if (!inFile.is_open() || !outFile.is_open()) {
        cerr << "Cannot open input or output file!" << endl;
        return 1;
    }

    int n;
    inFile >> n;

    vector<vector<int>> adjacencyMatrix(n, vector<int>(n));
    //Tạo một ma trận có n hàng, mỗi hàng lại là một vector có n phần tử → Ma trận vuông NxN
    //Với tất cả phần tử được khởi tạo mặc định bằng 0.

    // Đọc ma trận kề từ file
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            inFile >> adjacencyMatrix[i][j];
        }
    }

    // Tính bậc của các đỉnh
    vector<int> degrees = calculateDegrees(adjacencyMatrix);

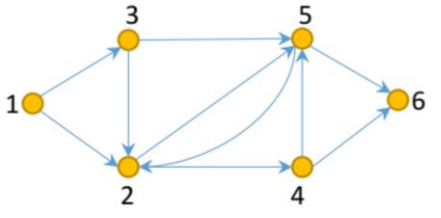
    // In số đỉnh của đồ thị ra file
    outFile << n << endl;

    // In bậc của các đỉnh ra file
    for (int i = 0; i < n; ++i) {
        outFile << degrees[i] << " ";
    }

    // Đóng các file
    inFile.close();
    outFile.close();

    return 0;
}

```

	OUTPUT.OUT 6 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0	Các đỉnh của đồ thị G: 1 2 3 4 5 6 Các cạnh của đồ thị G: (1, 2) (1, 3) (2, 4) (2, 5) (3, 2) (3, 5) (4, 5) (4, 6) (5, 2) (5, 6) Bạc vào và bạc ra của các đỉnh: Đỉnh 1: Bạc vào = 0, Bạc ra = 2 Đỉnh 2: Bạc vào = 3, Bạc ra = 2 Đỉnh 3: Bạc vào = 1, Bạc ra = 2 Đỉnh 4: Bạc vào = 1, Bạc ra = 2 Đỉnh 5: Bạc vào = 3, Bạc ra = 2 Đỉnh 6: Bạc vào = 2, Bạc ra = 0
---	--	---

*** Code nhập tay:**

```

#include <iostream>
#include <vector>
using namespace std;

//Hàm nhập ma trận kề
void v_input(vector<vector<int>>& adjacencyMatrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> adjacencyMatrix[i][j];
        }
    }
}

//Hàm tính bậc của đồ thị
vector<pair<int, int>> cal_Degree(vector<vector<int>>adjacencyMatrix, int n) {
    //Tạo một vector lưu bậc của các đỉnh, sau khi xong hàm thì trả về vector này -> Kiểu trả
    về là vector
    vector<pair<int, int>>Degree(n, { 0, 0 });

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (adjacencyMatrix[i][j] == 1) Degree[i].first++; //bán bậc ra = dòng
            if (adjacencyMatrix[j][i] == 1) Degree[i].second++; //bán bậc vào = cột
        }
    }
    return Degree;
}

//Hàm in bậc của đồ thị
void printDegree(vector<pair<int, int>>Degree, int n) {
    for (int i = 0; i < n; i++) {
        cout << "Đỉnh " << i+1 << ": Bạc ra: " << Degree[i].first << "; Bạc vào: " <<
        Degree[i].second << endl;
    }
}

```



```

//Hàm in các cạnh đồ thị = in chỉ số của ma trận nếu phần tử tại đó bằng 1
void printEdge(vector<vector<int>>adjacencyMatrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (adjacencyMatrix[i][j] == 1) {
                cout << "(" << i + 1 << ", " << j + 1 << ")" << endl;
            }
        }
    }
}

//Hàm in các đỉnh của đồ thị.
void printVertex(int n) {
    for (int i = 0; i < n; i++) {
        cout << i+1 << " ";
    }
    cout << endl;
}

int main() {
    int n;
    cout << "Nhap so dinh: "; cin >> n;
    cout << "=====\n";
    cout << "Nhap ma tran cap " << n << "x" << n << ": " << endl;

    vector<vector<int>>adjacencyMatrix(n, vector<int>(n));

    v_input(adjacencyMatrix, n);

    cout << "=====\n";
    cout << "Danh sach canh cua do thi: \n";
    printEdge(adjacencyMatrix, n);

    cout << "=====\n";
    cout << "Danh sach dinh cua do thi: \n";
    printVertex(n);

    cout << "=====\n";
    cout << "Danh sach bac cua do thi: \n";
    printDegree(cal_Degree(adjacencyMatrix, n), n);

    return 0;
}

```

Kết quả:

Nhap so dinh: 6

=====

Nhap ma tran cap 6x6:

```

0 1 1 0 0 0
0 0 0 1 1 0
0 1 0 0 1 0
0 0 0 0 1 1
0 1 0 0 0 1
0 0 0 0 0 0

```

=====

Danh sach canh cua do thi:

(1, 2)

(1, 3)

(2, 4)
(2, 5)
(3, 2)
(3, 5)
(4, 5)
(4, 6)
(5, 2)
(5, 6)

=====

Danh sach dinh cua do thi:

1 2 3 4 5 6

=====

Danh sach bac cua do thi:

Dinh 1: Bac ra: 2; Bac vao: 0

Dinh 2: Bac ra: 2; Bac vao: 3

Dinh 3: Bac ra: 2; Bac vao: 1

Dinh 4: Bac ra: 2; Bac vao: 1

Dinh 5: Bac ra: 2; Bac vao: 3

Dinh 6: Bac ra: 0; Bac vao: 2

*** Code lưu KQ vào file:**

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

// Hàm tính bậc vào và bậc ra của các đỉnh trong đồ thị
vector<pair<int, int>> calculateDegrees(const vector<vector<int>>& adjacencyMatrix) {
    int n = adjacencyMatrix.size();
    vector<pair<int, int>> degrees(n, {0, 0});

    for (int i = 0; i < n; ++i) {
        // Tính bậc vào
        for (int j = 0; j < n; ++j) {
            if (adjacencyMatrix[j][i] == 1) {
                degrees[i].first++;
            }
        }

        // Tính bậc ra
        for (int j = 0; j < n; ++j) {
            if (adjacencyMatrix[i][j] == 1) {
                degrees[i].second++;
            }
        }
    }

    return degrees;
}

int main() {
    ifstream inFile("E:\\BacVaoRa.INP");
    ofstream outFile("E:\\BacVaoRa.OUT");

    if (!inFile.is_open() || !outFile.is_open()) {
        cerr << "Cannot open input or output file!" << endl;
        return 1;
    }
}
```

```

}

int n;
inFile >> n;

vector<vector<int>> adjacencyMatrix(n, vector<int>(n));

// Đọc ma trận kề từ file
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        inFile >> adjacencyMatrix[i][j];
    }
}

// Tính bậc vào và bậc ra của các đỉnh
vector<pair<int, int>> degrees = calculateDegrees(adjacencyMatrix);

// In số đỉnh của đồ thị ra file
outFile << n << endl;

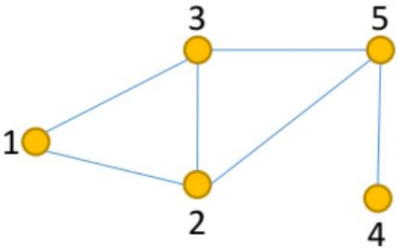
// In bậc vào và bậc ra của các đỉnh ra file
for (int i = 0; i < n; ++i) {
    outFile << degrees[i].first << " " << degrees[i].second << endl;
}

// Đóng các file
inFile.close();
outFile.close();

return 0;
}

```

2.5.2 Danh sách kề & Danh sách cạnh

	<u>Danh sách kề</u>	<u>Danh sách cạnh</u>
 <p>OUTPUT.OUT</p> <p>5 2 3 3 1 3</p>	<p>INPUT.INP</p> <p>5 2 3 -1 1 3 5 -1 1 2 5 -1 5 -1 2 3 4 -1</p>	<p>INPUT.INP</p> <p>5 6 1 2 1 3 2 3 2 5 3 5 4 5</p>

* Danh sách kề:

- Code nhập tay, dùng vector 2 chiều, pair để lưu danh sách cạnh:

```

#include <iostream>
#include <vector>
#include <utility>
#include <set>

using namespace std;

//Hàm nhập danh sách đỉnh kề
void InputAdjacencyList(vector<vector<int>>&AdjList, vector<int>&Degree, const int n) {
    for (int i = 0; i < n; i++) {
        cout << "Nhập ds ke cua dinh thu " << i + 1 << ": (ket thuc ds bang -1) ";
        int adjvertex;

```

```

        while (cin >> adjvertex && adjvertex != -1) {
            AdjList[i].push_back(adjvertex); // Lưu đỉnh kề (giả sử đỉnh nhập từ 1 đến n)
            Degree[i]++;
        }
    }
}

//In danh sách kề đã nhập
void PrintAdjList(const vector<vector<int>>>adjList, const int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < adjList[i].size(); j++) {
            cout << adjList[i][j] << " ";
        }
        cout << endl;
    }
}

//Hàm in bậc của mỗi đỉnh
void PrintDegree(const vector<int>Degree, const int n) {
    for (int i = 0; i < n; i++) {
        cout << "Đỉnh " << i + 1 << ": " << Degree[i] << endl;
    }
}

void PrintEdge(const vector<vector<int>>>AdjList, const int n) {

    vector<pair<int, int>>>edgeList;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < AdjList[i].size(); j++) {
            if (i+1 <= AdjList[i][j]) {
                //Điều kiện để 1,2 in mà 2,1 không cần in nữa: (đỉnh) i+1 <= (đỉnh)
                trong ds kề
                /*vd:
                * i + 1 = 1, ds kề đỉnh 1 = 2, 3 --> cạnh: 1,2; 1,3
                * i + 1 = 2, ds kề đỉnh 2 = 1, 3, 5 --> cạnh 2,1; 2,3; 2,5
                Do cạnh 2,1 đã in rồi (1,2) thì ta thấy i <= thành phần đỉnh kề (1 <=
                2, mà 2>1 là không thỏa ĐK == đã in rồi)*/
                edgeList.push_back({ i+1, AdjList[i][j] });
            }
        }
    }

    for (int i = 0; i < edgeList.size(); i++) {
        cout << "(" << edgeList[i].first << ", " << edgeList[i].second << ")" << endl;
    }
}

//In các đỉnh
void printVertex(const int n) {
    for (int i = 0; i < n; i++) {
        cout << i + 1 << " ";
    }
    cout << endl;
}

int main() {
    int n; //số đỉnh của đồ thị
    cout << "Nhập số đỉnh của đồ thị: ";
    cin >> n;

    cout << "=====\n";
    cout << "Nhập danh sách kề: " << endl;
}

```

```

vector<vector<int>>>adjacencyMatrix(n);
vector<int>Degree(n, 0);

InputAdjacencyList(adjacencyMatrix, Degree, n);

cout << "=====\n";
cout << "In danh sach ke: " << endl;
PrintAdjList(adjacencyMatrix, n);

cout << "=====\n";
cout << "Danh sach canh cua do thi: \n";
PrintEdge(adjacencyMatrix, n);

cout << "=====\n";
cout << "Danh sach dinh cua do thi: \n";
printVertex(n);

cout << "=====\n";
cout << "Danh sach bac cua do thi: \n";
PrintDegree(Degree, n);

return 0;
}

```

Kết quả:

Nhap so dinh cua do thi: 5

=====

Nhap danh sach ke:

Nhap ds ke cua dinh thu 1: (ket thuc ds bang -1) 2 3 -1

Nhap ds ke cua dinh thu 2: (ket thuc ds bang -1) 1 3 5 -1

Nhap ds ke cua dinh thu 3: (ket thuc ds bang -1) 1 2 5 -1

Nhap ds ke cua dinh thu 4: (ket thuc ds bang -1) 5 -1

Nhap ds ke cua dinh thu 5: (ket thuc ds bang -1) 2 3 4 -1

=====

In danh sach ke:

2 3

1 3 5

1 2 5

5

2 3 4

=====

Danh sach canh cua do thi:

(1, 2)

(1, 3)

(2, 3)

(2, 5)

(3, 5)

(4, 5)

=====

Danh sach dinh cua do thi:

1 2 3 4 5

=====

Danh sach bac cua do thi:

Dinh 1: 2
Dinh 2: 3
Dinh 3: 3
Dinh 4: 1
Dinh 5: 3

- Code nhập file, dùng mảng động 2 chiều:

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

int main() {
    ifstream infile("E:\\undirected_adjacency_list.txt"); // Mở file để đọc

    int n; // Số đỉnh
    infile >> n; // Đọc số đỉnh từ dòng đầu tiên của file

    // Tạo mảng để lưu danh sách kề
    int **adjacency_list = new int*[n];
    int *degrees = new int[n](); // Mảng lưu trữ bậc của mỗi đỉnh, được khởi tạo với giá trị 0

    // Đọc danh sách kề từ file
    for (int i = 0; i < n; ++i) {
        int neighbor;
        adjacency_list[i] = new int[n]; // Mảng lưu trữ các đỉnh kề
        int count = 0;
        while (true) {
            infile >> neighbor;
            if (neighbor == -1) break; // Kết thúc danh sách kề của một đỉnh
            adjacency_list[i][count++] = neighbor;
            // Tăng bậc của đỉnh kề lên 1
            degrees[i]++;
        }
        // Đánh dấu kết thúc danh sách kề
        adjacency_list[i][count] = -1;
    }

    infile.close(); // Đóng file sau khi đọc xong

    // Mở file để ghi kết quả
    ofstream outfile("E:\\out_undirected_adjacency_list.txt");
    if (!outfile.is_open()) {
        cerr << "Không thể tạo file đầu ra.";
        return 1;
    }

    // Ghi số đỉnh
    outfile << n << endl;

    // Ghi dãy bậc
    for (int i = 0; i < n; ++i) {
        outfile << degrees[i];
        if (i != n - 1) {
            outfile << " ";
        }
    }

    // Đóng tệp
```

```

outfile.close();

// Giải phóng bộ nhớ
for (int i = 0; i < n; ++i) {
    delete[] adjacency_list[i];
}
delete[] adjacency_list;
delete[] degrees;

return 0;
}

```

*** Danh sách cạnh (Code nhập tay):**

```

#include <iostream>
#include <utility>
#include <vector>

using namespace std;

//Nhập danh sách cạnh
void InputEdgeList(vector<pair<int, int>>&EdgeList, int m) {
    int f, s; //biến tạm lưu cạnh là 2 đỉnh kề nhau

    for (int i = 0; i < m; i++) {
        cin >> f >> s;
        EdgeList[i] = make_pair(f, s);
    }
}

//Tính bậc của đồ thị
vector<int>Cal_Degree(vector<pair<int, int>>EdgeList, int n, int m) {
    vector <pair<int, int>>EdgeList2;
    for (int i = 0; i < m; i++) {
        pair<int, int>temp;
        temp = make_pair(EdgeList[i].second, EdgeList[i].first);
        EdgeList2.push_back(temp);
        EdgeList2.push_back(EdgeList[i]);
    }

    vector<int>Degree(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < EdgeList2.size(); j++) {
            if ((i + 1) == EdgeList2[j].first) {
                Degree[i]++;
            }
        }
    }
    return Degree;
}

//Hàm in bậc của đồ thị
void printDegree(vector<int>Degree, int n) {
    for (int i = 0; i < n; i++) {
        cout << "Đỉnh " << i + 1 << ": " << Degree[i] << endl;
    }
}

//Hàm in các đỉnh của đồ thị.
void printVertex(int n) {
    for (int i = 0; i < n; i++) {
        cout << i + 1 << " ";
    }
}

```



```

    }
    cout << endl;
}

int main() {
    cout << "Nhap lan luot so dinh & so canh: ";
    int n, m; // n = số đỉnh, m = số cạnh
    cin >> n >> m;

    cout << "=====\n";
    cout << "Nhap danh sach canh: " << endl;

    vector<pair<int, int>>EdgeList(m);

    InputEdgeList(EdgeList, m);

    cout << "=====\n";
    cout << "Danh sach dinh cua do thi: \n";
    printVertex(n);

    cout << "=====\n";
    cout << "Danh sach bac cua do thi: \n";
    printDegree(Cal_Degree(EdgeList, n, m), n);

    return 0;
}

```

Kết quả:

Nhap lan luot so dinh & so canh: 5 6

=====

Nhap danh sach canh:

1 2
1 3
2 3
2 5
3 5
4 5

=====

Danh sach dinh cua do thi:

1 2 3 4 5

=====

Danh sach bac cua do thi:

Dinh 1: 2
Dinh 2: 3
Dinh 3: 3
Dinh 4: 1
Dinh 5: 3