CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

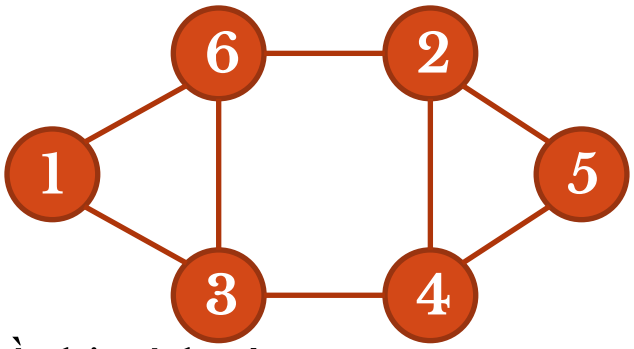
ĐỒ THỊ VÀ ỨNG DỤNG

Đồ thị và ứng dụng

- Khái niệm và định nghĩa
- Biểu diễn đồ thị
- Duyệt đồ thị
- Đồ thị Euler và đồ thị Hamilton
- Cây khung nhỏ nhất của đồ thị
- Đường đi ngắn nhất trên đồ thị

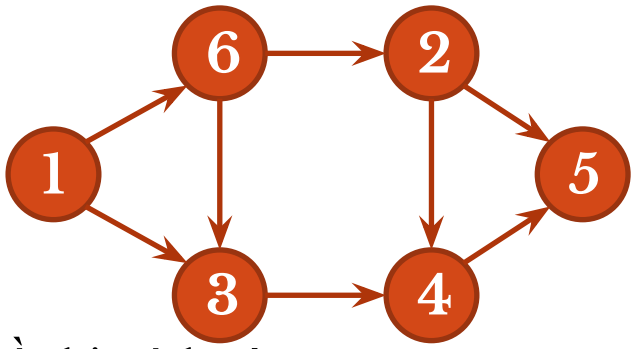
Khái niệm và định nghĩa

- Đồ thị là một đối tượng toán học mô hình hoá các thực thể và các liên kết giữa các thực thể đó
- Đồ thị $G = (V, E)$ trong đó V là tập đỉnh và E là tập cạnh (cung)
- Với mỗi $(u, v) \in E$: ta nói v kề với u



Đồ thị vô hướng

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



Đồ thị vô hướng

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

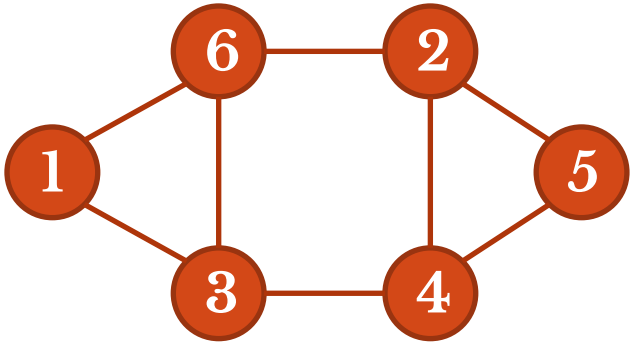
Khái niệm và định nghĩa

- Bậc của một đỉnh trên đồ thị vô hướng là số đỉnh kề với đỉnh đó:

$$deg(v) = \#\{u \mid (u, v) \in E\}$$

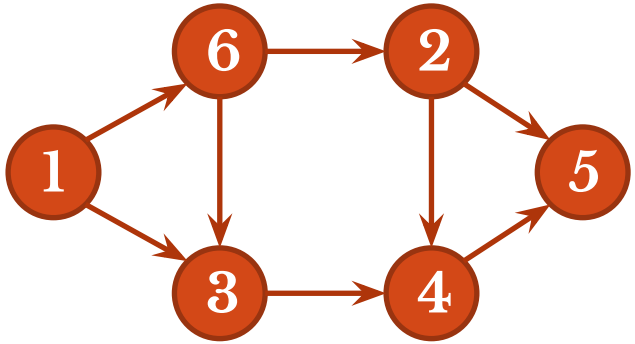
- Bán bậc vào (ra) của một đỉnh trên đồ thị có hướng là số cung đi vào (ra) đỉnh đó:

$$deg^-(v) = \#\{u \mid (u, v) \in E\},\ deg^+(v) = \#\{u \mid (v, u) \in E\}$$



đồ thị vô hướng

$deg(1) = 2,\ deg(4) = 3$

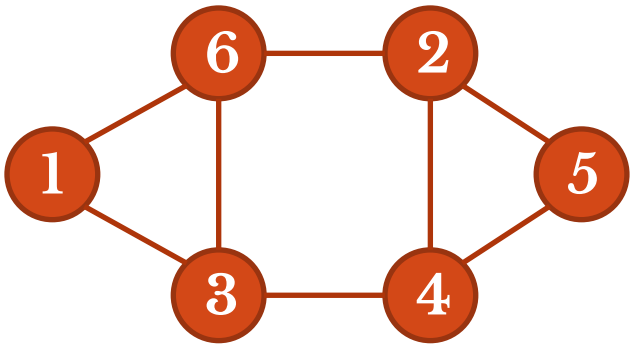


đồ thị có hướng

$deg^-(1) = 0,\ deg^+(1) = 2$

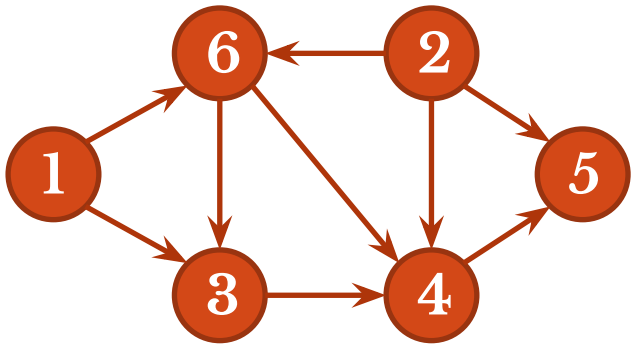
Khái niệm và định nghĩa

- Cho đồ thị $G=(V, E)$ và 2 đỉnh $s, t \in V$, đường đi từ s đến t trên G là dãy $s = x_0, x_1, \ldots, x_k = t$ trong đó $(x_i, x_{i+1}) \in E$, với $\forall\ i = 0, 1, \ldots, k\text{-}1$



Đường đi từ 1 đến 5:

- 1, 3, 4, 5
- 1, 6, 2, 5

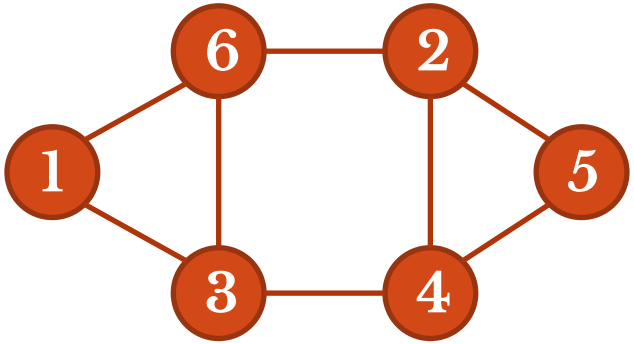


Đường đi từ 1 đến 5:

- 1, 3, 4, 5
- 1, 6, 4, 5

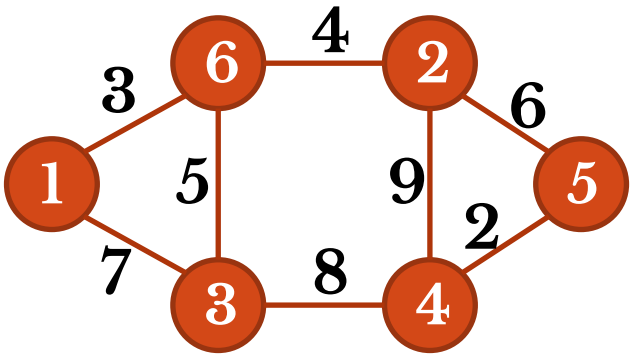
Biểu diễn đồ thị

- Ma trận kề



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

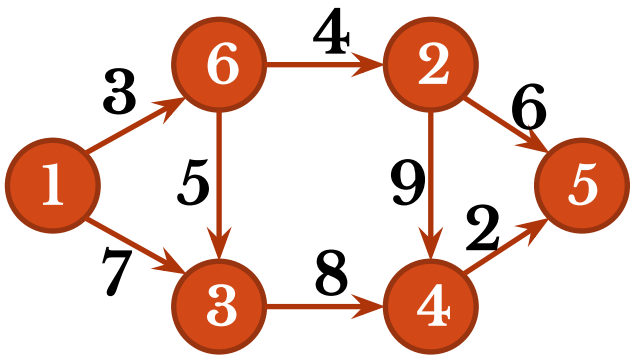
- Ma trận trọng số



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

Biểu diễn đồ thị

- Danh sách kề
  - Với mỗi $v \in V$, $A(v)$ là tập các bộ (v, u, w) trong đó w là trọng số của cung (v, u)
  - $A(1) = \{(1, 6, 3), (1, 3, 7)\}$
  - $A(2) = \{(2, 4, 9), (2, 5, 6)\}$
  - $A(3) = \{(3, 4, 8)\}$
  - $A(4) = \{(4, 5, 2)\}$
  - $A(5) = \{\}$
  - $A(6) = \{(6, 3, 5), (6, 2, 4)\}$



Duyệt đồ thị

- Duyệt các đỉnh của đồ thị theo một thứ tự nào đó
- Các đỉnh được duyệt (thăm) đúng 1 lần
- Hai phương pháp cơ bản:
  - Duyệt theo chiều sâu
  - Duyệt theo chiều rộng

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

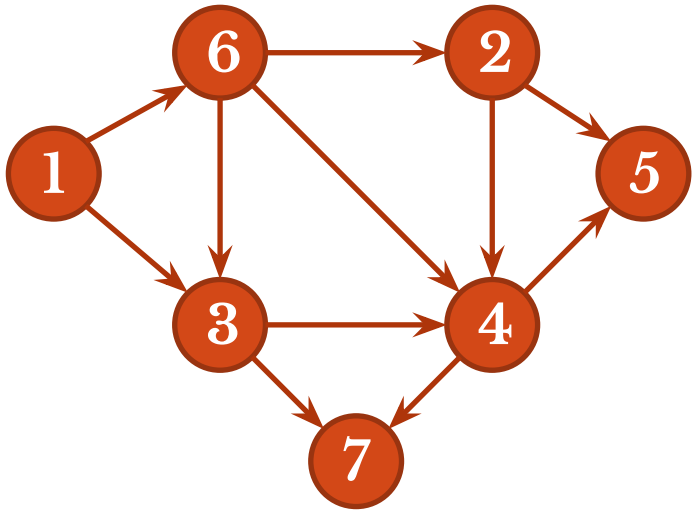
- DFS(u): duyệt theo chiều sâu bắt đầu từ đỉnh u
  - Nếu tồn tại đỉnh v trong danh sách kề của u chưa được thăm thì tiến hành thăm v và gọi DFS(v)
  - Nếu tất cả các đỉnh kề với u đã được thăm thì DFS quay trở lại đỉnh x mà từ đó thăm u để tiến hành thăm các đỉnh khác kề với x mà chưa được thăm. Lúc này đỉnh u được gọi là *đã duyệt xong*
- Cấu trúc dữ liệu: với mỗi đỉnh v của đồ thị
  - $p(v)$: là đỉnh từ đó thăm v
  - $d(v)$: thời điểm v được thăm nhưng chưa duyệt xong
  - $f(v)$: thời điểm đỉnh v đã được duyệt xong
  - color(v)
    - WHITE: chưa thăm
    - GRAY: đã được thăm nhưng chưa duyệt xong
    - BLACK: đã duyệt xong

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

```
DFS(u) {
  t = t + 1;
  d(u) = t;
  color(u) = GRAY;
  foreach (đỉnh v kề với u) {
    if(color(v) = WHITE) {
      p(v) = u;
      DFS(v);
    }
  }
  t = t + 1;
  f(u) = t;
  color(u) = BLACK;
}
```

```
DFS() {
  foreach (đỉnh u thuộc V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  foreach(đỉnh u thuộc V) {
    if(color(u) = WHITE) {
      DFS(u);
    }
  }
}
```

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS



đỉnh	1	2	3	4	5	6	7
d							
f							
p	-	-	-	-	-	-	-
color	W	W	W	W	W	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

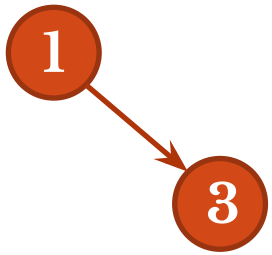
DFS(1)

1

đỉnh	1	2	3	4	5	6	7
d	1						
f							
p	-	-	-	-	-	-	-
color	G	W	W	W	W	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

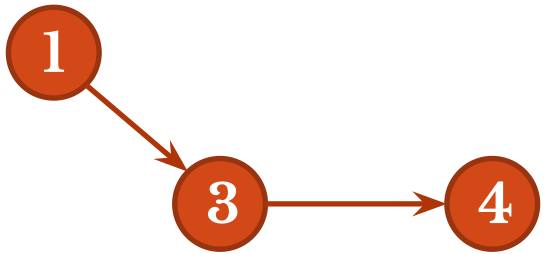
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2				
f							
p	-	-	1	-	-	-	-
color	G	W	G	W	W	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

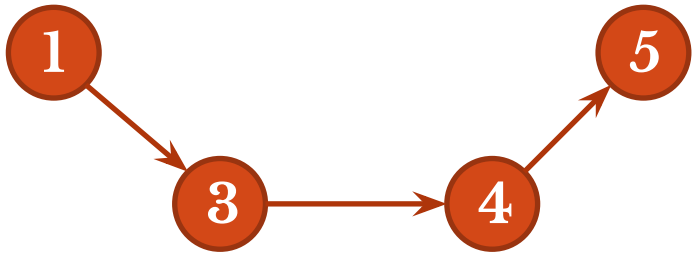
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3			
f							
p	-	-	1	3	-	-	-
color	G	W	G	G	W	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

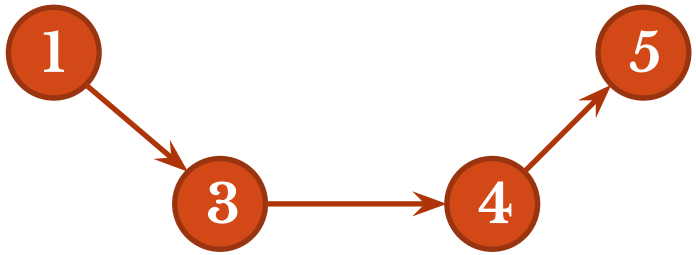
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		
f							
p	-	-	1	3	4	-	-
color	G	W	G	G	G	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

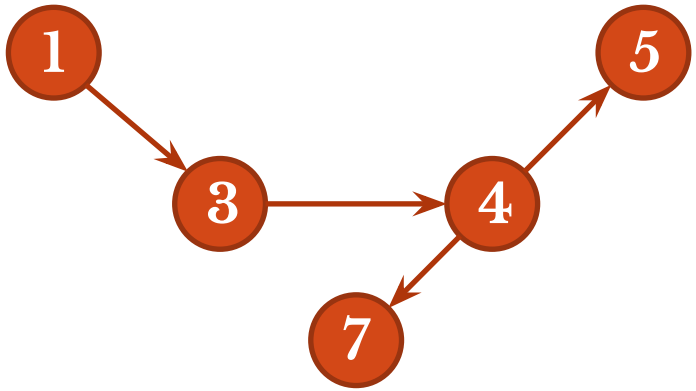
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		
f					5		
p	-	-	1	3	4	-	-
color	G	W	G	G	B	W	W

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

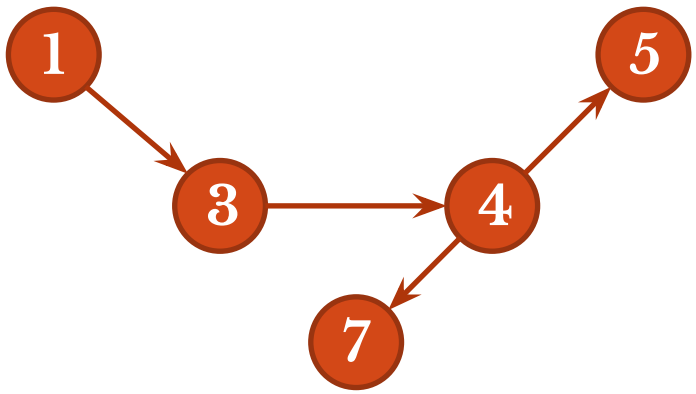
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	G

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

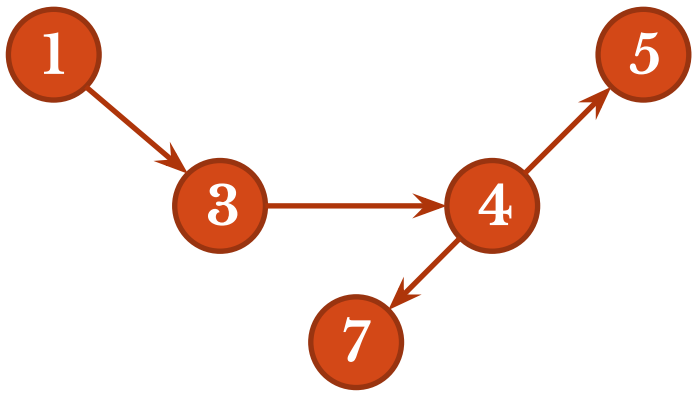
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f					5		7
p	-	-	1	3	4	-	4
color	G	W	G	G	B	W	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

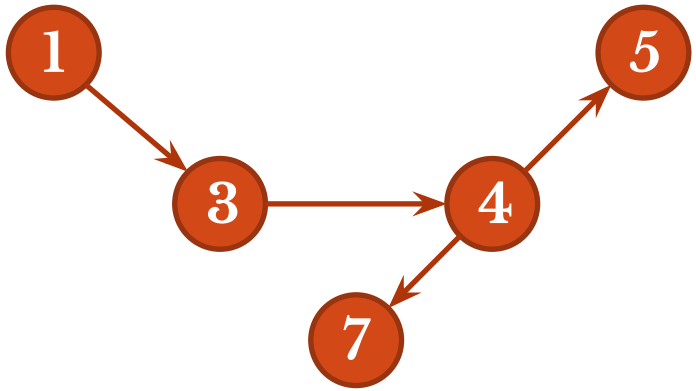
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f				8	5		7
p	-	-	1	3	4	-	4
color	G	W	G	B	B	W	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

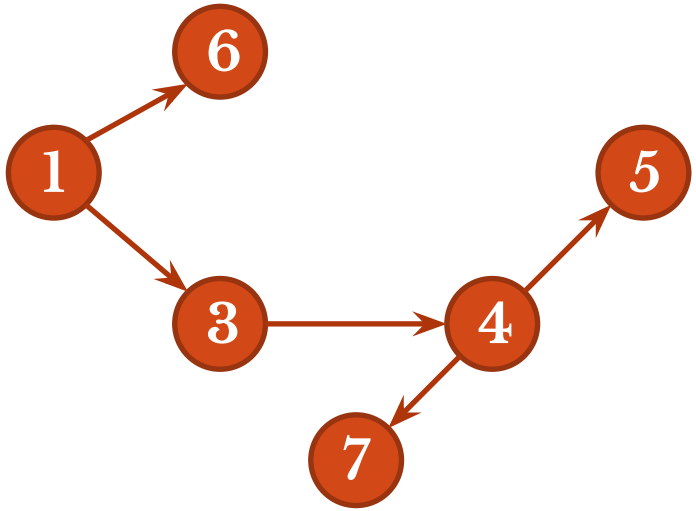
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4		6
f			9	8	5		7
p	-	-	1	3	4	-	4
color	G	W	B	B	B	W	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

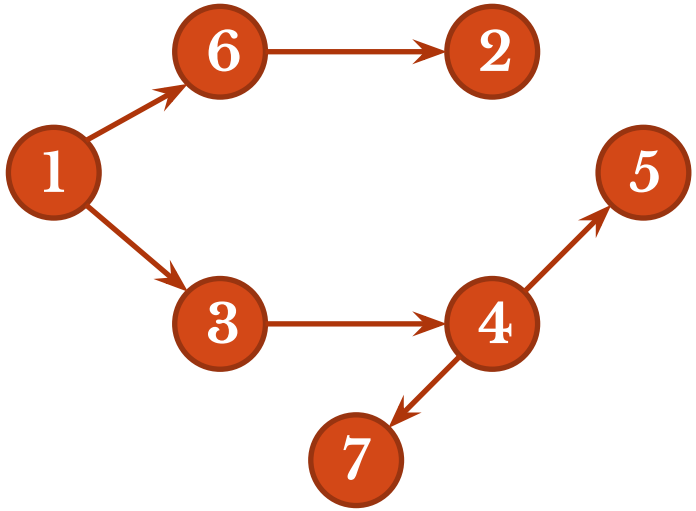
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1		2	3	4	10	6
f			9	8	5		7
p	-	-	1	3	4	1	4
color	G	W	B	B	B	G	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

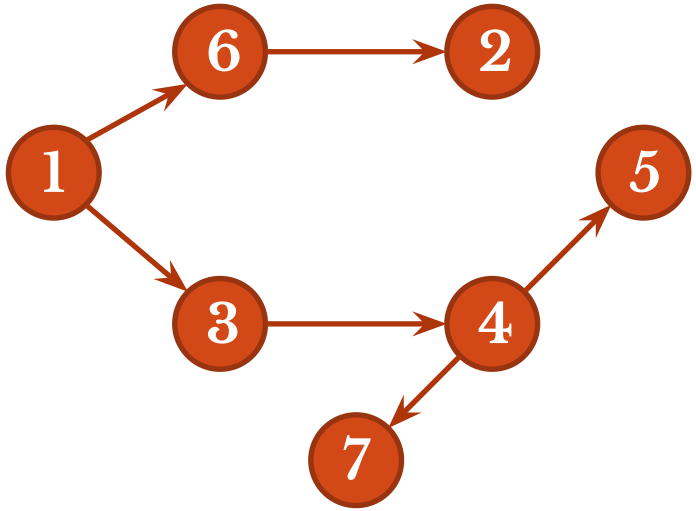
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f			9	8	5		7
p	-	6	1	3	4	1	4
color	G	G	B	B	B	G	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

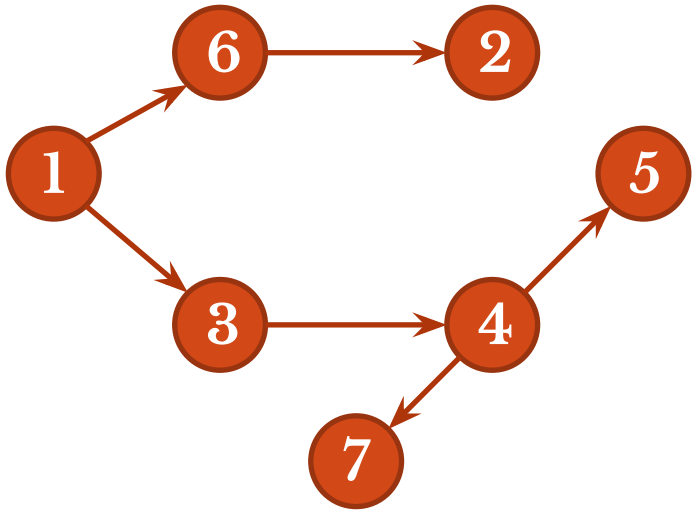
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5		7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	G	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

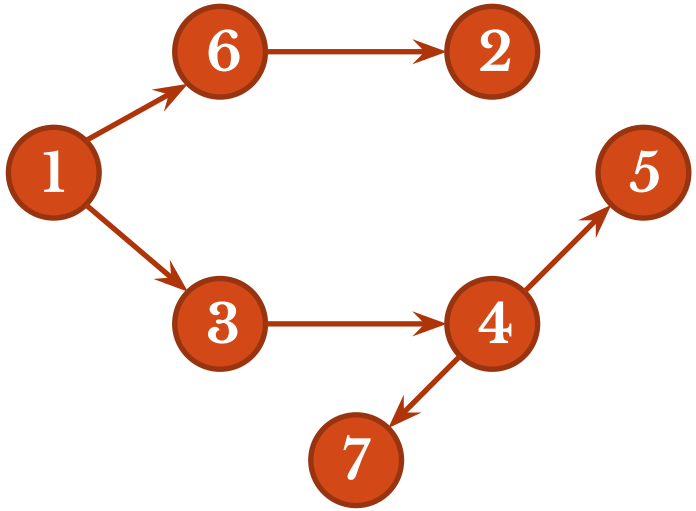
DFS(1)



đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f		12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	G	B	B	B	B	B	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

DFS(1)



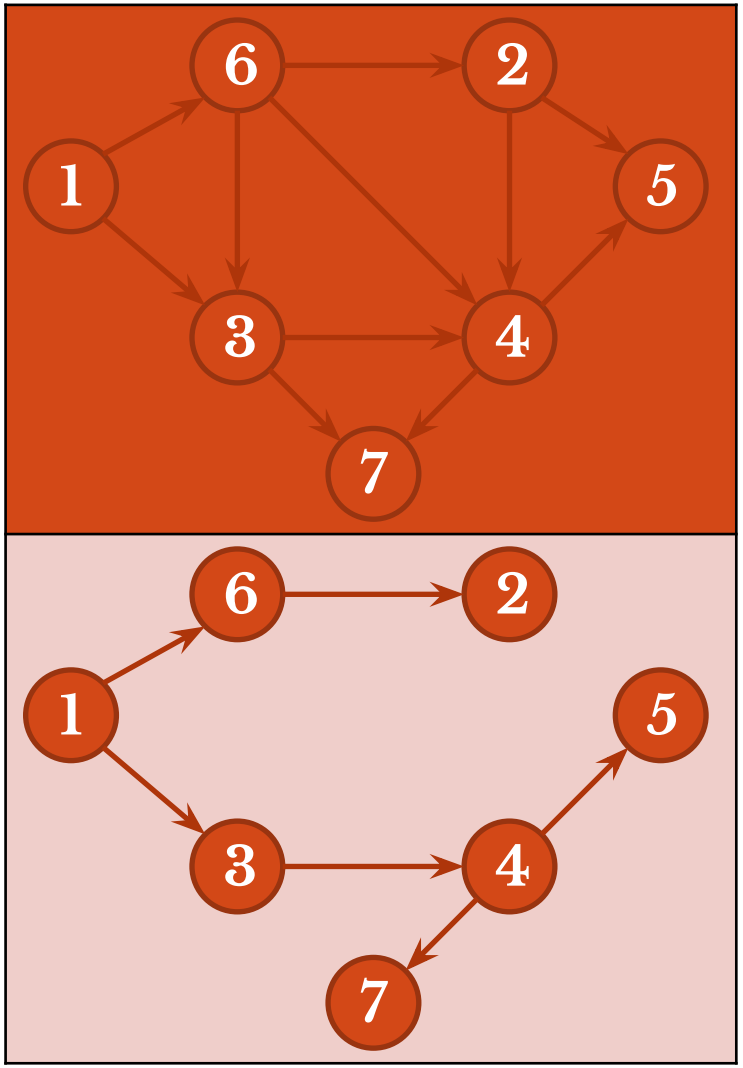
đỉnh	1	2	3	4	5	6	7
d	1	11	2	3	4	10	6
f	14	12	9	8	5	13	7
p	-	6	1	3	4	1	4
color	B	B	B	B	B	B	B

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

- Kết quả DFS trên đồ thị sẽ cho 1 rừng, bao gồm các cây DFS
- Phân loại cạnh
  - Cạnh của cây (tree edge): (u,v) là cạnh của cây nếu v được thăm từ u
  - Cạnh ngược (back edge): (u,v) là cạnh ngược nếu v là tổ tiên của u trong cây DFS
  - Cạnh thuận (forward edge): (u,v) là cạnh thuận nếu u là tổ tiên của v trong cây DFS
  - Cạnh ngang (crossing edge): các cạnh còn lại

Duyệt đồ thị theo chiều sâu
Depth First Search - DFS

- Phân loại cạnh
  - Cạnh của cây: (1, 6), (1, 3), (6, 2), (3, 4), (4, 5), (4, 7)
  - Cạnh ngược:
  - Cạnh thuận: (3, 7)
  - Cạnh ngang: (6, 3), (6, 4), (2, 4), (2,5)



Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

- BFS(u): duyệt theo chiều rộng xuất phát từ đỉnh u
  - Thăm các đỉnh u
  - Thăm các đỉnh kề với u mà chưa được thăm (gọi là các đỉnh mức 1)
  - Thăm các đỉnh kề với các đỉnh mức 1 mà chưa được thăm (gọi là các đỉnh mức 2)
  - Thăm các đỉnh kề với các đỉnh mức 2 mà chưa được thăm (gọi là các đỉnh mức 3)
  - . . .
- Sử dụng cấu trúc hàng đợi (queue) để cài đặt

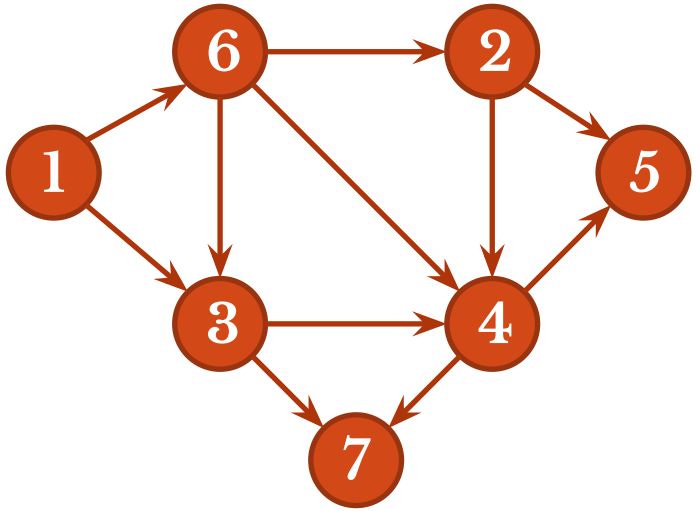
Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

```
BFS(u) {
  d(u) = 0;
  khởi tạo hàng đợi Q;
  enqueue(Q,u);
  color(u) = GRAY;
  while(Q khác rỗng) {
    v = dequeue(Q);
    foreach(x kề với v) {
      if(color(x) = WHITE){
        d(x) = d(v) + 1;
        color(x) = GRAY;
        enqueue(Q,x);
      }
    }
  }
}
```

```
BFS() {
  foreach (đỉnh u thuộc V) {
    color(u) = WHITE;
    p(u) = NIL;
  }
  foreach(đỉnh u thuộc V) {
    if(color(u) = WHITE) {
       BFS(u);
    }
  }
}
```

Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

BFS(1)



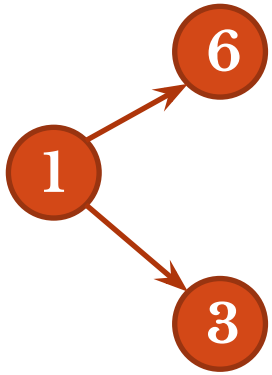
Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

BFS(1)

1

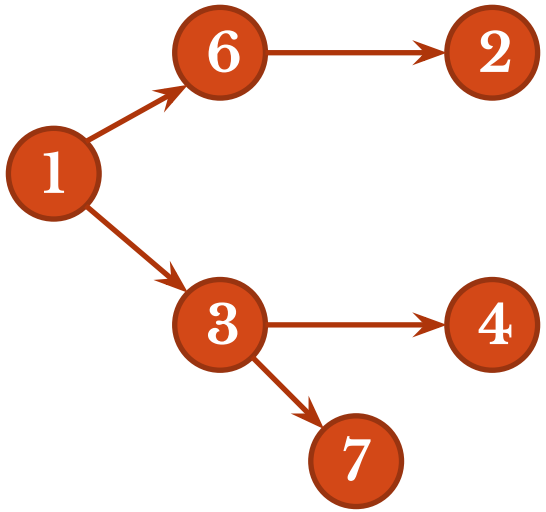
Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

BFS(1)



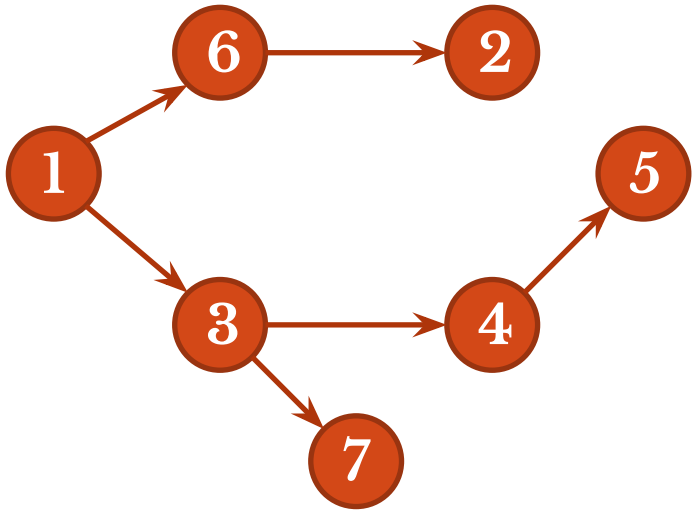
Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

BFS(1)



Duyệt đồ thị theo chiều rộng
Breadth First Search - BFS

BFS(1)

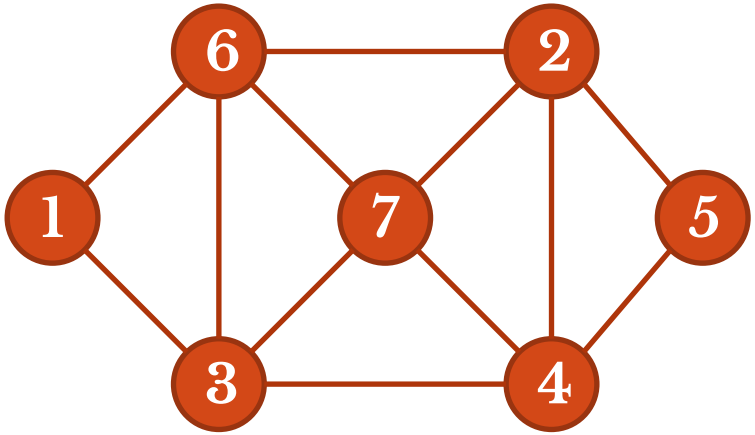


Ứng dụng

- Sắp xếp môn học: cho tập N môn học 1, 2, 3…, N. Tập P thể hiện điều kiện tiên quyết bao gồm các cặp 2 môn học i và j cho biết môn i phải học trước môn j. Hãy đưa ra dãy thứ tự các môn học sao cho với mỗi cặp $<i, j>$ trong điều kiện tiên quyết thì môn i phải đứng trước môn j trong dãy đó.

Đồ thị Euler và đồ thị Hamilton

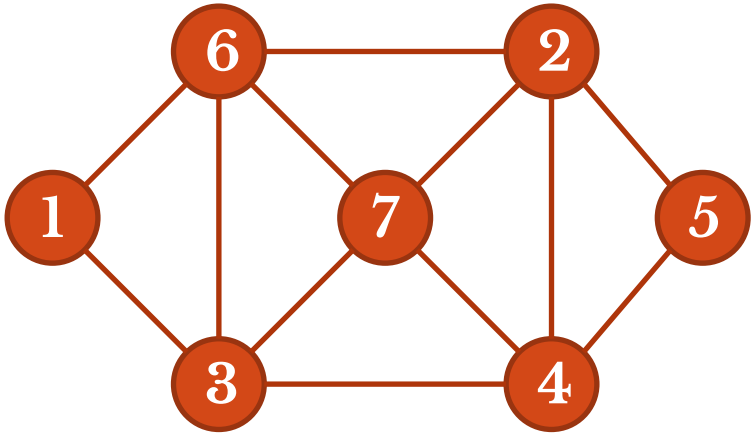
- Cho đồ thị vô hướng G = (V, E)
  - Chu trình Euler trên G là chu trình đi qua tất cả các cạnh, mỗi cạnh đúng 1 lần.
  - Chu trình Hamilton là chu trình đi qua tất cả các đỉnh, mỗi đỉnh đúng 1 lần (trừ đỉnh xuất phát)
- Đồ thị chứa chu trình Euler được gọi là đồ thị Euler
- Đồ thị chứa chu trình Hamilton được gọi là đồ thị Hamilton



- Chu trình Euler: 1, 6, 3, 7, 6, 2, 5, 4, 2, 7, 4, 3, 1
- Chu trình Hamilton: 1, 6, 2, 5, 4, 7, 3, 1

Đồ thị Euler và đồ thị Hamilton

- Định lí: Đồ thị vô hướng là đồ thị Euler khi và chỉ khi nó là đồ thị liên thông trong đó các đỉnh có bậc là số chẵn
- Định lí: Đồ thị vô hướng n đỉnh trong đó mỗi đỉnh có bậc lớn hơn hoặc bằng $n/2$ là đồ thị Hamilton



Đồ thị Euler và đồ thị Hamilton

- Thuật toán tìm chu trình Euler, sử dụng ngăn xếp
- Đồ thị đầu vào $G = (V, E)$ trong đó $A(x)$ là tập các đỉnh kề với đỉnh x

```
euler(G = (V, E)) {
  khởi tạo stack S, CE;
  chọn v là một đỉnh bất kỳ thuộc V;
  push(S,v);
  while(S khác rỗng) {
    x là nút ở đỉnh của S;
    if(A(x) khác rỗng) {
      chọn y là một đỉnh bất kỳ ∈ A(x);
      push(S,y);
      loại bỏ cạnh (x,y) khỏi G;
    }else{
      x = pop(S); push(CE,x);
    }
  }
  thứ tự các đỉnh trong CE tạo chu trình euler
}
```

Đồ thị Euler và đồ thị Hamilton

- Thuật toán tìm chu trình Hamilton, sử dụng duyệt đệ quy quay lui
- Đồ thị đầu vào $G = (V, E)$ trong đó
  - $V = \{1, 2, \ldots, n\}$
  - $A(v)$ là tập các đỉnh kề với đỉnh v
- Mô hình hoá: mảng $x[1..n]$
- Mảng đánh dấu mark[v] = true nếu đỉnh v đã được dùng và mark[v] = false, ngược lại

```
TRY(k) {
  for(v ∈ A(x[k-1]) {
    if(not mark[v]) {
      x[k] = v;
      mark[v] = true;
      if(k == n) {
        if(v ∈ A(x[1]) {
          ghi nhận chu trình Hamilton trong x;
        }
      }else{
        TRY(k+1);
      }
      mark[v] = false;
    }
  }
}
```

Cây khung nhỏ nhất trên đồ thị

- Cho đồ thị vô hướng liên thông $G = (V, E, w)$.
  - Mỗi cạnh $(u, v) \in E$ có $w(u, v)$ là trọng số
  - Nếu $(u, v) \notin E$ thì $w(u, v) = \infty$
- Cây khung là đồ thị vô hướng, liên thông và không có chu trình chứa tất cả các đỉnh của G.
- Cây $T = (V, F)$ trong đó $F \subseteq E$ gọi là một cây khung của G
  - Trọng số $w(T) = \sum_{e \in F} w(e)$
- Tìm cây khung của G có trọng số nhỏ nhất
- Ứng dụng
  - Thiết kế mạng truyền thông

Cây khung nhỏ nhất trên đồ thị

- Thuật toán KRUSKAL tìm cây khung nhỏ nhất
- Ý tưởng chính
  - Sắp xếp các cạnh theo thứ tự không giảm của trọng số□ danh sách L
  - Xuất phát cây khung T rỗng
  - Duyệt danh sách L từ trái qua phải, với mỗi cạnh e, thực hiện bổ sung vào T nếu không tạo ra chu trình

Cây khung nhỏ nhất trên đồ thị

- Thuật toán PRIM tìm cây khung nhỏ nhất
- Ý tưởng chính
  - Xây dựng cây bằng cách kết nạp lần lượt các cạnh vào cây T theo chiến lược tham lam
  - Ban đầu, tập đỉnh V_T của cây chỉ gồm 1 đỉnh được chọn ngẫu nhiên
  - Mỗi bước, chọn đỉnh (chưa thuộc V_T) gần T nhất để kết nạp đỉnh và cạnh vào cây T
- Cấu trúc dữ liệu
  - Với mỗi đỉnh $v \notin V_T$
    - $d(v)$ là khoảng cách từ v đến V_T:

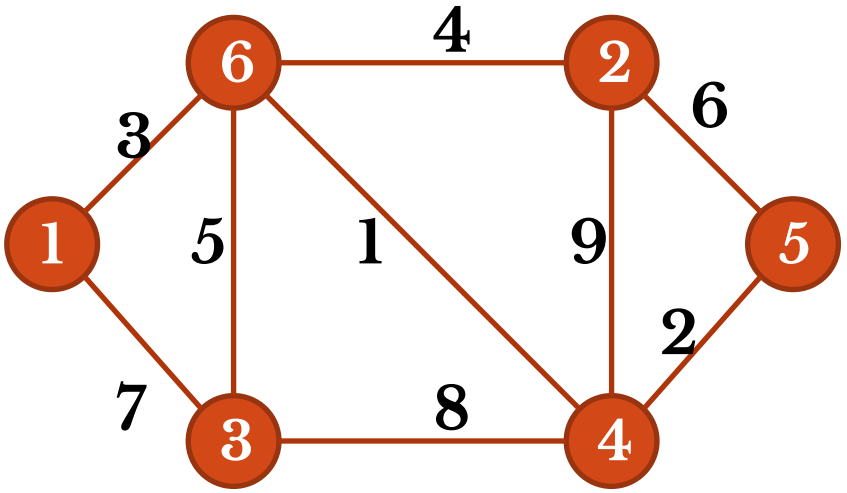
$$d(v) = \min\{w(v, u) \mid u \in V_T, (u, v) \in E\}$$

    - *near(v)*: đỉnh $\in V_T$ có $w(v, \text{near}(v)) = d(v)$;

Cây khung nhỏ nhất trên đồ thị

```
PRIM(G = (V,E,w)) {
  chọn s là một đỉnh nào đó của V;
  for(v ∈ V) {
     d(v) = w(s,v); near(v) = s;
  }
  E_T = {}; V_T = {s};
  while(|V_T| ≠ |V|) {
    v = chọn đỉnh ∈ V \ V_T có d(v) nhỏ nhất;
    V_T = V_T∪{v}; E_T = E_T ∪ {(v, near(v))};
    for(x ∈ V \ V_T) {
      if(d(x) > w(x,v) {
        d(x) = w(x,v);
        near(x) = v;
      }
    }
  }
  return (V_T, E_T);
}
```

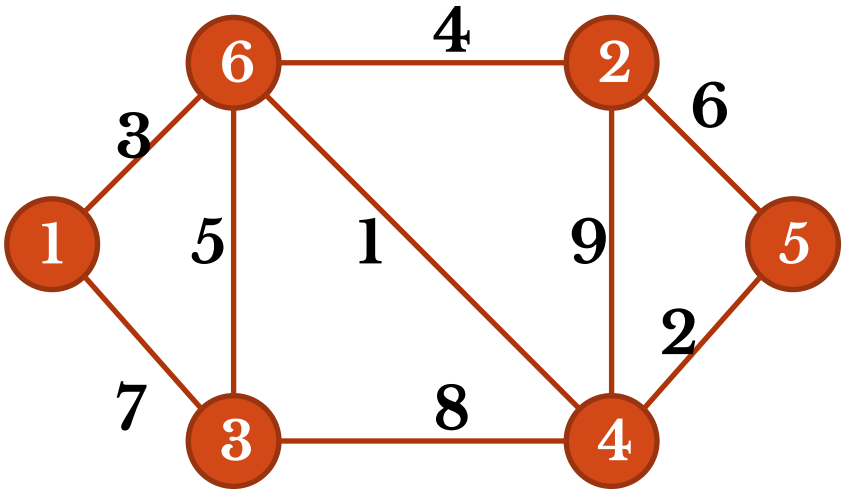
Cây khung nhỏ nhất trên đồ thị



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), \text{near}(v))$
- Đỉnh xuất phát $s = 1$

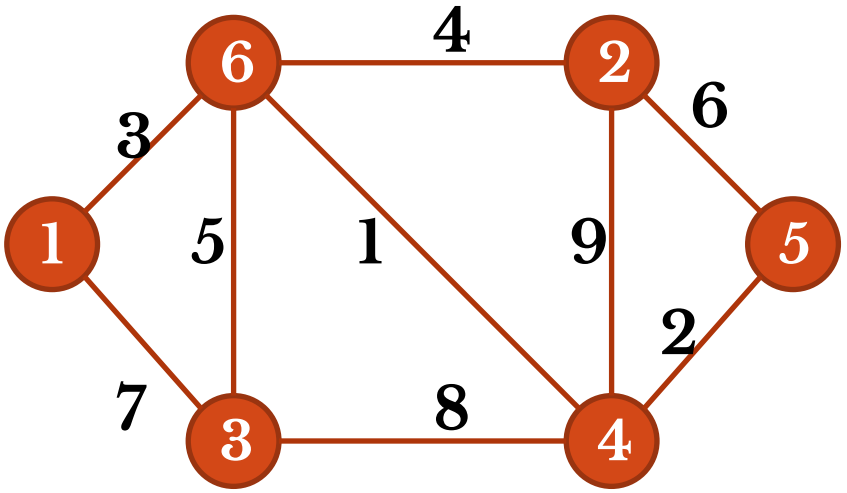
	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1)	
Bước 1	-						
Bước 2	-						
Bước 3	-						
Bước 4	-						
Bước 5	-						

Cây khung nhỏ nhất trên đồ thị



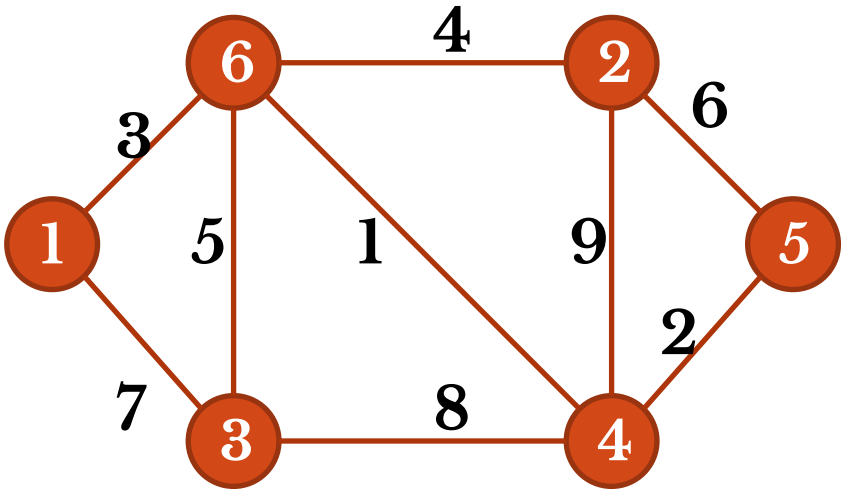
	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Bước 1	-	(4,6)	(5,6)	(1,6)	(∞,1)	-	
Bước 2	-					-	
Bước 3	-					-	
Bước 4	-					-	
Bước 5	-					-	

Cây khung nhỏ nhất trên đồ thị



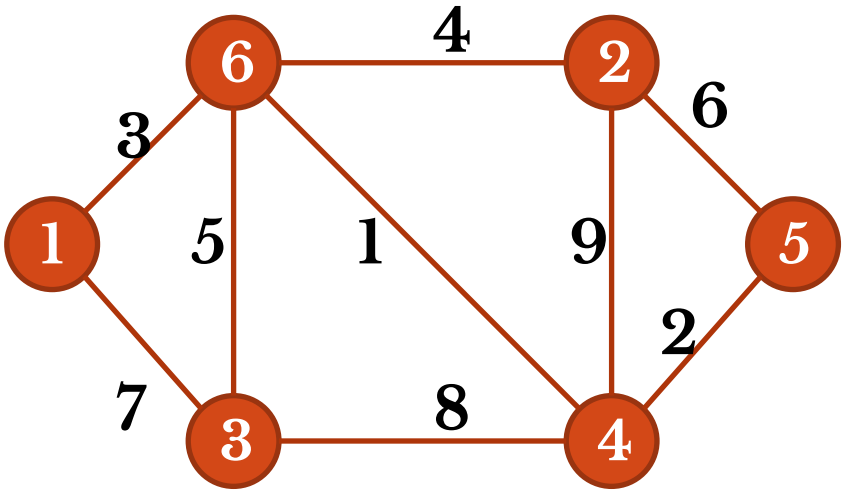
	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Bước 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6), (4,6)
Bước 2	-	(4,6)	(5,6)	-	(2,4)	-	
Bước 3	-			-		-	
Bước 4	-			-		-	
Bước 5	-			-		-	

Cây khung nhỏ nhất trên đồ thị



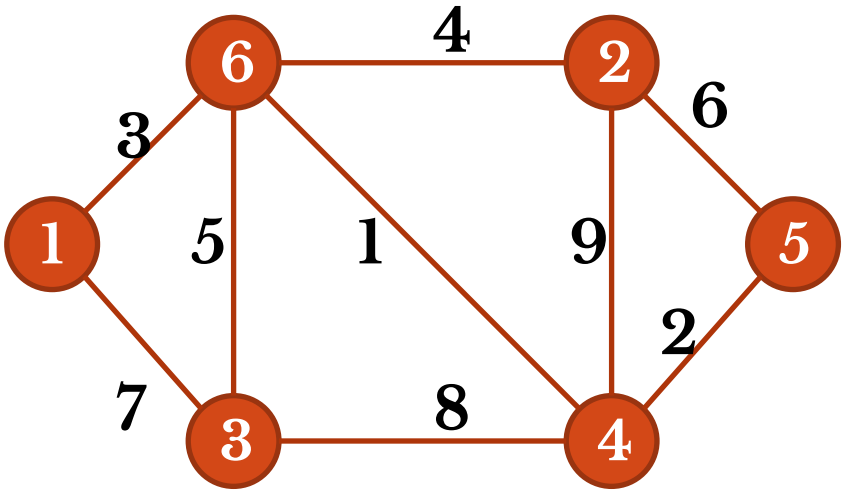
	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Bước 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6),(4,6)
Bước 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6),(4,6),(4,5)
Bước 3	-	(4,6)	(5,6)	-	-	-	
Bước 4	-			-	-	-	
Bước 5	-			-	-	-	

Cây khung nhỏ nhất trên đồ thị



	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *	(1,6)
Bước 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6),(4,6)
Bước 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6),(4,6),(4,5)
Bước 3	-	(4,6) *	(5,6)	-	-	-	(1,6),(4,6),(4,5),(2,6)
Bước 4	-	-	(5,6)	-	-	-	
Bước 5	-	-		-	-	-	

Cây khung nhỏ nhất trên đồ thị



	1	2	3	4	5	6	E_T
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1)	
Bước 1	-	(4,6)	(5,6)	(1,6) *	(∞,1)	-	(1,6)
Bước 2	-	(4,6)	(5,6)	-	(2,4) *	-	(1,6), (4,6)
Bước 3	-	(4,6) *	(5,6)	-	-	-	(1,6), (4,6), (4,5)
Bước 4	-	-	(5,6) *	-	-	-	(1,6), (4,6), (4,5), (2,6)
Bước 5	-	-	-	-	-	-	(1,6), (4,6), (4,5), (2,6), (3,6)

Cây khung nhỏ nhất trên đồ thị
Cài đặt thuật toán PRIM

```
#include <stdio.h>
#include <set>
#include <map>
#include <stack>
using namespace std;
struct Arc{
     int nod;
     int w;
};
set<int> V;// set of nodes
map<int, set<Arc*> > A;// A[v] is the set of adjacent arcs of v

// data structure for prim
map<int, int> d;
map<int, int> near;
set<int> S;
```

Cây khung nhỏ nhất trên đồ thị
Cài đặt thuật toán PRIM

```
int findMin(){
  // find a node v of NonFixed having minimum d[v]
  int min = 1000000;
  int v_min = -1;
  for(set<int>::iterator p = S.begin(); p != S.end(); p++){
    int v = *p;
    if(d[v] < min){
      min = d[v];
      v_min = v;
    }
  }
  return v_min;
}
```

Cây khung nhỏ nhất trên đồ thị
Cài đặt thuật toán PRIM

```
void prim(int s){
  // initialization
  for(set<int>::iterator pi = V.begin(); pi != V.end(); pi++){
    int x = *pi;  d[x] = 100000000;
  }
  d[s] = 0;
  for(set<Arc*>::iterator ps = A[s].begin(); ps != A[s].end(); ps++){
    Arc* a = *ps;
    int x = a->nod;  int w = a->w;
    d[x] = w; near[x] = s;
  }
  for(set<int>::iterator pi = V.begin(); pi != V.end(); pi++){
    int v = *pi;
    if(v != s)
      S.insert(v);
  }
  . . .
```

Cây khung nhỏ nhất trên đồ thị
Cài đặt thuật toán PRIM

```
  //LOOP
  while(S.size() > 0){
    int v = findMin();
    printf("select edge (%d,%d) with d = %d\n",v,near[v],d[v]);
    S.erase(v);
    // update d of non-fixed nodes
    for(set<Arc*>::iterator pv = A[v].begin(); pv != A[v].end(); pv++){
      Arc* a = *pv;
      int x = a->nod;
      int w = a->w;
      if(d[x] > w){
        d[x] = w;
        near[x] = v;
      }
    }
  }
}
```

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

- Cho đồ thị trọng số $G = (V, E, w)$.
  - Mỗi cạnh $(u, v) \in E$ có $w(u, v)$ là trọng số không âm
  - Nếu $(u, v) \notin E$ thì $w(u, v) = \infty$
- Cho s là một đỉnh thuộc V, tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại

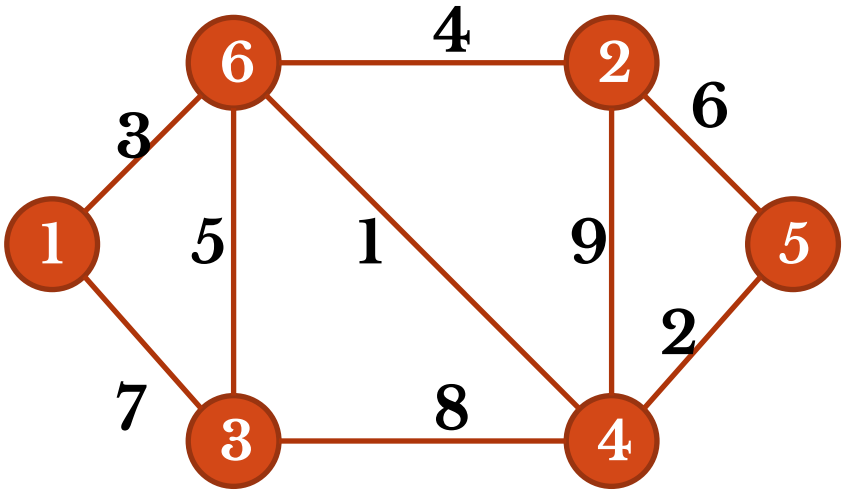
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

- Ý tưởng thuật toán Dijkstra:
  - Với mỗi đỉnh $v \in V$, duy trì:
    - $\mathcal{P}(v)$ là đường đi cận trên của đường đi ngắn nhất từ s đến v
    - $d(v)$: trọng số của $\mathcal{P}(v)$
    - $p(v)$: đỉnh trước đỉnh v trên $\mathcal{P}(v)$
  - Khởi tạo
    - $\mathcal{P}(v) = \langle s, v \rangle$, $d(v) = w(s,v)$, $p(v) = s$
  - Làm tốt cận trên
    - Mỗi khi phát hiện có đỉnh u sao cho $d(v) > d(u) + w(u, v)$ thì cập nhật
      - $d(v) = d(u) + w(u, v)$
      - $p(v) = u$

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
Dijkstra(G = (V,E,w)) {
  for(v ∈ V) {
     d(v) = w(s,v); p(v) = s;
  }
  S = V \ {s};
  while(S ≠ {}) {
    u = chọn đỉnh ∈ S có d(u) nhỏ nhất;
    S = S \ {u};
    for(v ∈ S) {
      if(d(v) > d(u) + w(u,v) {
        d(v) = d(u) + w(u,v);
        p(v) = u;
      }
    }
  }
}
```

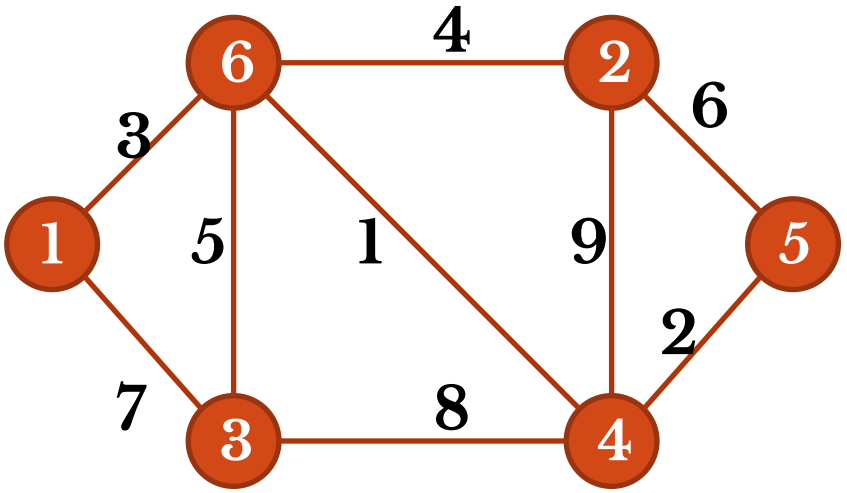
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	**(0,1)**	**(∞,1)**	**(7, 1)**	**(∞, 1)**	**(∞, 1)**	**(3,1)**
Bước 1	-					
Bước 2	-					
Bước 3	-					
Bước 4	-					
Bước 5	-					

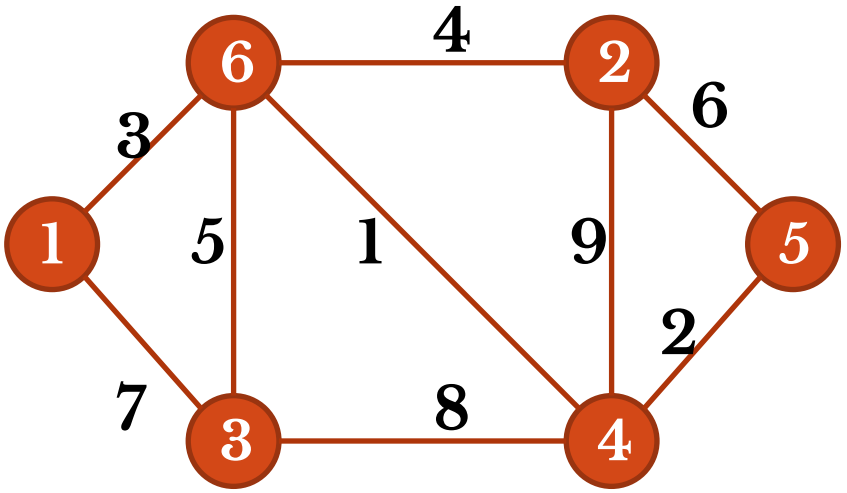
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Bước 1	-	(7,6)	(7,1)	(4,6)	(∞,1)	-
Bước 2	-					-
Bước 3	-					-
Bước 4	-					-
Bước 5	-					-

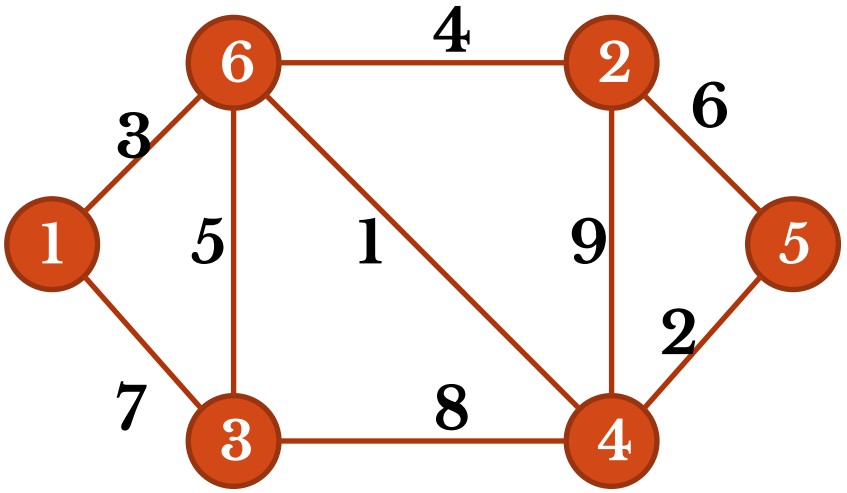
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Bước 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Bước 2	-	(7,6)	(7,1)	-	(6, 4)	-
Bước 3	-			-		-
Bước 4	-			-		-
Bước 5	-			-		-

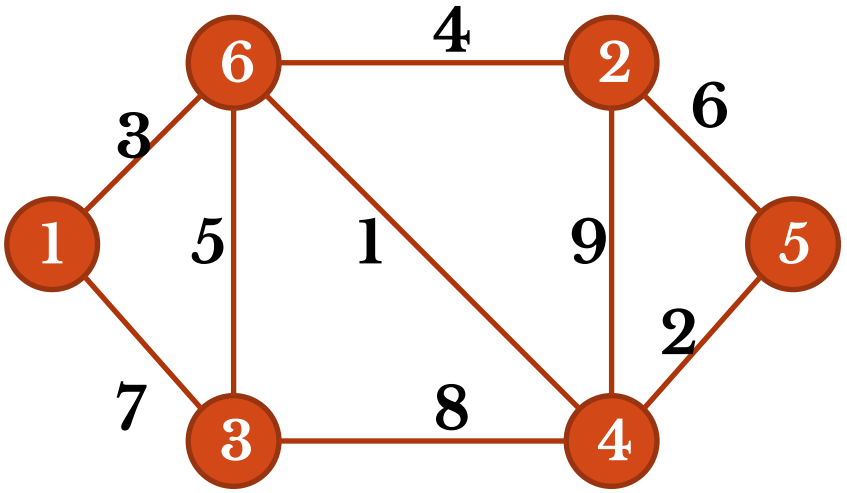
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Bước 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Bước 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Bước 3	-	(7,6)	(7,1)	-	-	-
Bước 4	-			-	-	-
Bước 5	-			-	-	-

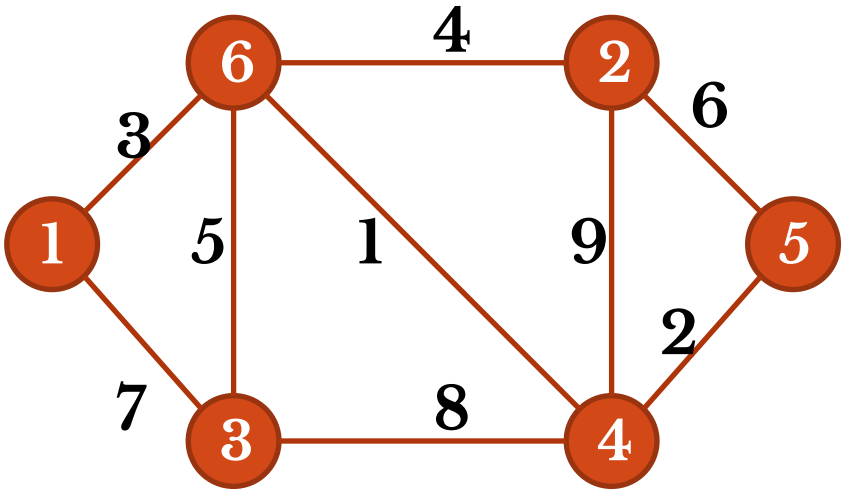
Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Bước 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Bước 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Bước 3	-	(7,6)	(7,1) *	-	-	-
Bước 4	-	(7,6)	-	-	-	-
Bước 5	-		-	-	-	-

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra



- Mỗi ô ứng với đỉnh v của bảng chứa nhãn $(d(v), p(v))$
- Đỉnh xuất phát $s = 1$

	1	2	3	4	5	6
Khởi tạo	(0,1)	(∞,1)	(7, 1)	(∞, 1)	(∞, 1)	(3,1) *
Bước 1	-	(7,6)	(7,1)	(4,6) *	(∞,1)	-
Bước 2	-	(7,6)	(7,1)	-	(6, 4) *	-
Bước 3	-	(7,6)	(7,1) *	-	-	-
Bước 4	-	(7,6) *	-	-	-	-
Bước 5	-	-	-	-	-	-

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
#include <stdio.h>
#include <set>
#include <map>
#include <stack>
using namespace std;
struct Arc{
  int nod;
  int w;
};

set<int> V;// set of nodes
map<int, set<Arc*> > A;// A[v] is the set of adjacent arcs of v

// data strcuture for dijkstra
map<int, int> d;
map<int, int> p;
set<int> S;
```

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
int findMin(){
  // find a node v of NonFixed having minimum d[v]
  int min = 1000000;
  int v_min = -1;
  for(set<int>::iterator p = S.begin(); p != S.end(); p++){
    int v = *p;
    if(d[v] < min){
      min = d[v];
      v_min = v;
    }
  }
  return v_min;
}
```

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
void dijkstra(int s){
  // initialization
  for(set<int>::iterator pi = V.begin(); pi != V.end(); pi++){
    int x = *pi;  d[x] = 100000000;
  }
  d[s] = 0;
  for(set<Arc*>::iterator ps = A[s].begin(); ps != A[s].end(); ps++){
    Arc* a = *ps;
    int x = a->nod;  int w = a->w;
    d[x] = w; p[x] = s;
  }
  for(set<int>::iterator pi = V.begin(); pi != V.end(); pi++){
    int v = *pi;
    if(v != s)
      S.insert(v);
  }
  . . .
```

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
  //LOOP
  while(S.size() > 0){
    int v = findMin();
    S.erase(v);
    // update label of nodes in S
    for(set<Arc*>::iterator pv = A[v].begin(); pv != A[v].end(); pv++){
      Arc* a = *pv;
      int x = a->nod;
      int w = a->w;
      if(d[x] > d[v] + w){
        d[x] = d[v] + w;
        p[x] = v;
      }
    }
  }
}
```

Đường đi ngắn nhất trên đồ thị
Thuật toán Dijkstra

```
void printPath(int s, int v){
  stack<int> S;
  int x = v;
  while(x != s){
    S.push(x);
    x = p[x];
  }
  printf("%d ",s);
  while(!S.empty()){
    int  x = S.top(); S.pop();
    printf("%d ",x);
  }
  printf("\n");
}
```