

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN
HỌC PHẦN: TOÁN RỜI RẠC
TÊN BÀI TẬP LỚN: BÀI THI GIỮA KỲ

Sinh viên thực hiện	Khóa	Lớp	Mã sinh viên
Trịnh Cẩm Nhung	13	DCCNTT13.10.13	20222446
Mạc Tiến Thành Đạt	13	DCCNTT13.10.13	20222402
Nguyễn Ngọc Doanh	13	DCCNTT13.10.13	20222539
Nguyễn Thị Hồng Duyên	13	DCCNTT13.10.13	20222441

Bắc Ninh, tháng 05 năm 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN: TOÁN RỜI RẠC

Nhóm: 4

TÊN (BÀI TẬP LỚN): BÀI THI GIỮA KÌ

STT	Sinh viên thực hiện	Khóa	Lớp	Mã sinh viên	Điểm bằng số	Điểm bằng chữ
1	Trịnh Cẩm Nhung	13	DCCNTT13.10.13	20222446		
2	Mạc Tiến Thành Đạt	13	DCCNTT13.10.13	20222402		
3	Nguyễn Ngọc Doanh	13	DCCNTT13.10.13	20222539		
4	Nguyễn Thị Hồng Duyên	13	DCCNTT13.10.13	20222441		

CÁN BỘ CHẤM 1

(Ký và ghi rõ họ tên)

CÁN BỘ CHẤM 2

(Ký và ghi rõ họ tên)

Bắc Ninh, tháng 05 năm 2023

MỤC LỤC

TOÁN RỜI RẠC

DANH MỤC CÁC TỪ VIẾT TẮT	5
PHÂN CÔNG CÔNG VIỆC	6
Phần 1. Lý thuyết.....	7
1.1 Quan hệ.....	7
1.1.1 Quan hệ hai ngôi	7
1.1.2 Quan hệ tương ứng, Lớp tương đương.	11
1.1.3 Quan hệ thứ tự. Biểu đồ Hasse.	13
1.2 Hàm số	15
1.2.1 Khái niệm đại số boole	15
1.2.2 Hàm số Boole.....	17
1.2.3 Mạch logic	19
1.2.4 Cực tiểu hoá các mạch logic	21
Phần 2. Thực hành.....	24
2.1 Thuật toán BFS.....	24
1.1.1 Khái niệm	24
1.1.2 Mô tả	24
1.1.3 Tính độ phức tạp.....	24
1.1.4 Kiểm nghiệm thuật toán bằng ví dụ thực tế:	25
1.1.5 Cài đặt.....	28
2.2 Thuật toán DFS.....	31
2.2.1 Khái niệm	31

2.2.2 Mô tả	31
2.2.3 Tính độ phức tạp.....	31
2.2.4 Kiểm nghiệm thuật toán bằng ví dụ thực tế	32
2.2.5 Cài đặt.....	34
2.3 Tự chọn	38
2.3.1 Thuật toán Kruskal xây dựng cây khung nhỏ nhất	38
2.3.2 Thuật toán Floyd	45

DANH MỤC CÁC TỪ VIẾT TẮT

STT	Chữ viết tắt	Giải thích
1	DFS	Depth-first search
2	BFS	Breadth First Search
3		

PHÂN CÔNG CÔNG VIỆC

Họ và tên	Mã sinh viên	Nội dung
Nguyễn Thị Hồng Duyên	20222441	Lý thuyết
Nguyễn Ngọc Doanh	20222539	Thuật toán BFS
Mạc Tiến Thành Đạt	20222402	Thuật toán DFS
Trịnh Cẩm Nhung	20222446	Tự chọn, code, tổng hợp báo cáo

Phần 1. Lý thuyết

1.1 Quan hệ

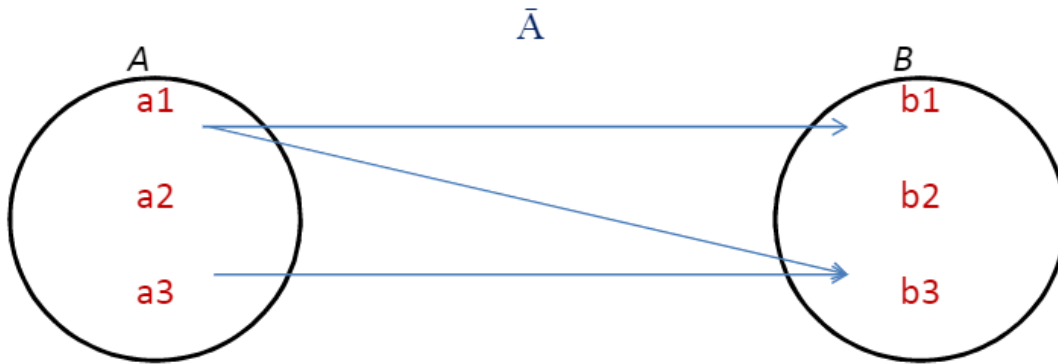
1.1.1 Quan hệ hai ngôi

- a. Định nghĩa: Cho hai tập A, B . Ta gọi tập R là một quan hệ hai ngôi từ A và B nếu $R \subseteq A \times B$.

Nếu $(a, b) \in R$ thì ta nói a có quan hệ R với b và ký hiệu $a R b$; ngược lại nếu $(a, b) \notin R$ thì ta ký hiệu $a \bar{R} b$.

Khi $A = B$, ta gọi R là quan hệ hai ngôi trên A .

Ví dụ:



$$R = \{ (a1, b1), (a1, b3), (a3, b3) \}$$

- b. Các tính chất của quan hệ.

Định nghĩa: giả sử R là một quan hệ hai ngôi trên tập A .

❖ Ta nói quan hệ R có tính phản xạ nếu và chỉ nếu :

$$A R a, \forall a \in A.$$

Ví dụ : Trên tập $A = \{1, 2, 3, 4\}$ quan hệ

$$R1 = \{ (1,1), (1,2), (2,1), (2,2), (3,4), (4,1), (4,4) \} \text{ không phản xạ vì } (3,3) \notin R1$$

$R_2 = \{(1,1), (1,2), (1,4), (2,2), (3,3), (4,1), (4,4)\}$ phản xạ vì $(1,1), (2,2), (3,3), (4,4) \in R_2$

Ví dụ

- Quan hệ \leq trên Z phản xạ vì $a \leq a, \forall a \in Z$.
- Quan hệ $>$ trên Z không phản xạ vì 1 không lớn hơn 1.
- Quan hệ “ $|$ ” (“ước số”) trên Z^+ là phản xạ vì mọi số nguyên dương a là ước của chính nó.

❖ Ta nói quan hệ R có tính đối xứng nếu và chỉ nếu

$$a R b \Rightarrow b R a, \forall a, b \in A$$

❖ Ta nói quan hệ R có tính phản xứng nếu và chỉ nếu

$$a R b \wedge b R a \Rightarrow a=b, \forall a, b \in A$$

Ví dụ :

- Quan hệ $R_1 = \{(1,1), (1,2), (2,1)\}$ trên tập $A = \{1, 2, 3, 4\}$ là đối xứng
- Quan hệ \leq trên Z không đối xứng, tuy nhiên nó phản xứng vì $(a \leq b) \wedge$

$$(b \leq a) \Rightarrow (a = b)$$

- Quan hệ “ $|$ ” (“ước số”) trên Z^+ không đối xứng, tuy nhiên nó có tính phản xứng vì $(a | b) \wedge (b | a) \Rightarrow (a = b)$.

❖ Ta nói quan hệ R có tính bắc cầu (truyền) nếu và chỉ nếu

$$(a R b \wedge b R c) \Rightarrow a R c, \quad \forall a, b, c \in A$$

c. Biểu diễn quan hệ

Định nghĩa: Cho R là quan hệ từ $A = \{1, 2, 3, 4\}$ đến $B = \{u, v, w\}$,

$$R = \{(1,u), (1,v), (2,w), (3,w), (4,u)\}$$

Khi đó R có thể biểu diễn như sau

	u	v	w
1	1	1	0
2	0	0	1
3	0	0	1
4	1	0	0



Đây là ma trận cấp 4×3 biểu diễn cho quan hệ R.

Cho R là quan hệ từ $A = \{a_1, a_2, \dots, a_m\}$ đến $B = \{b_1, b_2, \dots, b_n\}$. Ma trận biểu diễn của R là ma trận cấp $m \times n$ $M_R = [m_{ij}]_{m \times n}$ xác định bởi

$$m_{ij} = \begin{cases} 0 & \text{nếu } (a_i, b_j) \notin R \\ 1 & \text{nếu } (a_i, b_j) \in R \end{cases}$$

Ví dụ: cho R là quan hệ từ $A = \{1, 2, 3\}$ đến $B = \{1, 2\}$: $a R b \Leftrightarrow a > b$. Khi đó ma trận biểu diễn của R là:

	1	2
1	0	0
2	1	0
3	1	1

$$m_{ij} = \begin{cases} 1 & \text{nếu } (a_i, b_j) \in R \\ 0 & \text{nếu } (a_i, b_j) \notin R \end{cases}$$

Ví dụ. Cho R là quan hệ từ $A = \{a_1, a_2, a_3\}$ đến $B = \{b_1, b_2, b_3, b_4, b_5\}$ được biểu diễn bởi ma trận

$$\mathbf{M}_R = \begin{matrix} & \begin{matrix} b_1 & b_2 & b_3 & b_4 & b_5 \end{matrix} \\ \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \end{matrix}$$

Khi đó R gồm các cặp:

$\{(a_1, b_2), (a_2, b_1), (a_2, b_3), (a_2, b_4), (a_3, b_1), (a_3, b_3), (a_3, b_5)\}$

- Cho R là quan hệ trên tập A, khi đó M_R là ma trận vuông.
- R là phản xạ nếu tất cả các phần tử trên đường chéo của M_R đều bằng 1: $m_{ii} = 1$ với mọi i

	U	V	W
U	1	1	0
V	0	1	1
W	0	0	1

- R đối xứng nếu M_R là đối xứng

$$m_{ij} = m_{ji} \text{ với mọi } i, j$$

	U	V	W
U	1	0	1
V	0	0	1
W	1	1	0

- R là phản xứng nếu M_R thỏa mãn

$$m_{ij} = 0 \text{ hoặc } m_{ji} = 0 \text{ nếu } i \neq j$$

	U	V	W
U	1	0	1
V	0	0	0
W	0	1	1

1.1.2 Quan hệ tương ứng, Lớp tương đương.

Định nghĩa. Quan hệ R trên tập A được gọi là tương đương nếu nó có tính chất phản xạ, đối xứng và bắc cầu :

Ví dụ. Quan hệ R trên các chuỗi ký tự xác định bởi aRb nếu a và b có cùng độ dài. Khi đó R là quan hệ tương đương.

Ví dụ. Cho R là quan hệ trên \mathbb{Z} sao cho aRb nếu $a - b$ nguyên. Khi đó R là quan hệ tương đương

Nhớ lại rằng nếu a và b là các số nguyên thì a được gọi là chia hết cho b, hoặc a là bội của b, hoặc b là ước của a, hoặc b chia hết a nếu tồn tại số nguyên k sao cho $a = kb$

Ví dụ. Cho m là số nguyên dương và R là quan hệ trên \mathbb{Z} sao cho aRb khi và chỉ khi $a - b$ chia hết cho m thì R là một quan hệ tương đương

- Mỗi quan hệ rõ ràng là phản xạ và đối xứng.
- Cho a, b, c là các số nguyên sao cho $a - b$ và $b - c$ lần lượt là đều chia hết cho m thì $a - c = a - b + b - c$ cũng chia hết cho m Do đó R là bắc cầu
- Mỗi quan hệ này được gọi là đồng dạng modulo m và chúng tôi viết

$$a \equiv b \pmod{m} \text{ thay vì } aRb$$

❖ Lớp tương đương

Định nghĩa. Cho R là quan hệ tương đương trên A và phần tử $a \in A$. Lớp tương đương chứa a được ký hiệu bởi $[a]_R$ hoặc $[a]$ là tập

$$[a]_R = \{b \in A \mid b R a\}$$

Ví dụ. Tìm các lớp tương đương modulo 8 chứa 0 và 1?

Giải. Lớp tương đương modulo 8 chứa 0 gồm tất cả các số nguyên a chia hết cho 8.

Do đó

$$[0]_8 = \{\dots, -16, -8, 0, 8, 16, \dots\}$$

Tương tự

$$[1]_8 = \{a \mid a \text{ chia 8 dư } 1\}$$

$$=\{\dots, -15, -7, 1, 9, 17, \dots\}$$

Chú ý. Trong ví dụ cuối, các lớp tương đương $[0]_8$ và $[1]_8$ là rời nhau. Tổng quát, chúng ta có

Định lý: Cho R là quan hệ tương đương trên tập A và $a, b \in A$, Khi đó

$$(i) \ a R b \text{ nếu } [a]_R = [b]_R$$

$$(ii) \ [a]_R \neq [b]_R \text{ nếu } [a]_R \cap [b]_R = \emptyset$$

Chú ý. Các lớp tương đương theo một quan hệ tương đương trên A tạo nên một phân hoạch trên A , nghĩa là chúng chia tập A thành các tập con rời nhau.

Note. Cho $\{A_1, A_2, \dots\}$ là phân hoạch A thành các tập con không rỗng, rời nhau. Khi đó có duy nhất quan hệ tương đương trên A sao cho mỗi A_i là một lớp tương đương.

Thật vậy với mỗi $a, b \in A$, ta đặt $a R b$ nếu có tập con A_i sao cho $a, b \in A_i$.

Dễ dàng chứng minh rằng R là quan hệ tương đương trên A và $[a]_R = A_i$ nếu $a \in A_i$

Ví dụ. Cho m là số nguyên dương, khi đó có m lớp đồng dư modulo m là $[0]_m, [1]_m, \dots, [m-1]_m$.

Chúng lập thành phân hoạch của \mathbb{Z} thành các tập con rời nhau.

Chú ý rằng

$$[0]_m = [m]_m = [2m]_m = \dots [1]_m = [m+1]_m = [2m+1]_m = \dots$$

.....

$$[m-1]_m = [2m-1]_m = [3m-1]_m = \dots$$

Mỗi lớp tương đương này được gọi là số nguyên modulo m .

Tập hợp các số nguyên modulo m được ký hiệu bởi Z_m

$$Z_m = \{ [0]_m, [1]_m, \dots, [m-1]_m \}$$

1.1.3 Quan hệ thứ tự. Biểu đồ Hesse.

- a) Định nghĩa : Quan hệ R trên tập A là quan hệ thứ tự (thứ tự) nếu nó có tính chất phản xạ, phản xứng và bắc cầu.

Người ta thường ký hiệu quan hệ thứ tự bởi $<$ Cặp $(A, <)$ được gọi là tập sắp thứ tự hay poset.

Phản xạ : $a < a$

Phản ứng : $(a < b) \wedge (b < a) \rightarrow (a = b)$

Bắc cầu : $(a < b) \wedge (b < c) \rightarrow (a < c)$

Ví dụ. Quan hệ ước số “ $|$ ” trên tập số nguyên dương là quan hệ thứ tự, nghĩa là $(Z^+, |)$ là poset

Phản xạ : có, $x | x$ vì $x = 1 \cdot x$

Bắc cầu: có $a | b$ nghĩa là $b = ka$, $b | c$ nghĩa là $c = jb$.

$$\text{khi đó } c = j(ka) = jka: a|c$$

Phản ứng : có $a|b$ nghĩa là $b = ka$, $b|a$ nghĩa là $a = jb$.

$$\text{khi đó } a = jka$$

$$\text{suy ra } j = k = 1, \text{ nghĩa là } a = b$$

Định nghĩa. Các phần tử a và b của poset $(S, <)$ gọi là so sánh được nếu $a < b$ hay $b < a$

Trái lại thì ta nói a và b không so sánh được.

Cho $(S, <)$, π nếu hai phần tử tùy ý của S đều so sánh được với nhau thì ta gọi nó là tập sắp thứ tự toàn phần.

Ta cũng nói rằng $<$ là thứ tự toàn phần hay thứ tự tuyến tính trên S

Ví dụ. Quan hệ " \leq " trên tập số nguyên dương là thứ tự toàn phần.

Ví dụ. Quan hệ ước số " $|$ " trên tập hợp số nguyên dương không là thứ tự toàn phần, vì các số 5 và 7 là không so sánh được.

b) Biểu đồ Hasse

Mỗi poset có thể biểu diễn bởi đồ thị đặc biệt ta gọi là biểu đồ Hasse.

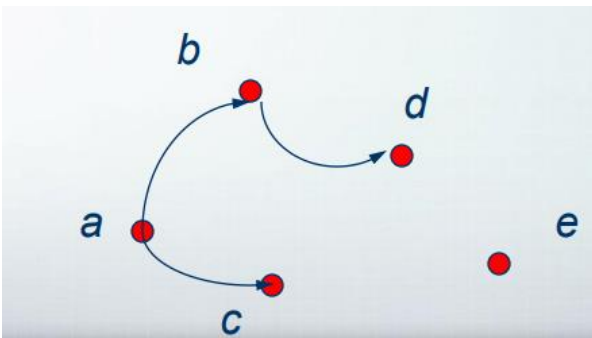
Để định nghĩa biểu đồ Hasse chúng ta cần các khái niệm phần tử trội và trội trực tiếp.

Định nghĩa: Phần tử b trong poset $(S, <)$ được gọi là phần tử trội của phần tử a trong S nếu $a < b$. Chúng ta cũng nói rằng a là được trội bởi b . Phần tử b được gọi là trội trực tiếp của a nếu b là trội của a , và không tồn tại trội c sao cho

$$a < c < b, a \neq c \neq b$$

Ta định nghĩa Biểu đồ Hasse của poset $(S, <)$ là đồ thị:

- + Mỗi phần tử của S được biểu diễn bởi một điểm trên mặt phẳng.
- + Nếu b là trội trực tiếp của a thì vẽ một cung đi từ a đến b .



$$a < b < d, a < c$$

1.2 Hàm số

1.2.1 Khái niệm đại số boole

Tập hợp khác rỗng S cùng với các phép toán ký hiệu nhân ($.$), cộng ($+$), lấy bù ($'$) được gọi là một đại số Boole nếu các tiên đề sau đây được thoả mãn với mọi $a, b, c \in S$.

1. Tính giao hoán:

a) $a.b = b.a$,

b) $a+b = b+a$.

2. Tính kết hợp:

a) $(a.b).c = a.(b.c)$,

b) $(a+b)+c = a+(b+c)$.

3. Tính phân phối:

a) $a.(b+c) = (a.b)+(a.c)$,

b) $a+(b.c) = (a+b).(a+c)$.

4. Tồn tại phần tử trung hoà: Tồn tại hai phần tử khác nhau của S , ký hiệu là 1 và 0 sao cho:

a) $a.1 = 1.a = a$,

b) $a+0 = 0+a = a$.

1 gọi là phần tử trung hoà của phép $.$ và 0 gọi là phần tử trung hoà của phép $+$.

5. Tồn tại phần tử bù: Với mọi $a \in S$, tồn tại duy nhất phần tử $a' \in S$ sao cho:

a) $a.a' = a'.a = 0$,

b) $a+a' = a'+a = 1$.

a' gọi là phần tử bù của a .

Chú ý: Trước hết cần lưu ý điều quan trọng sau đây: các tiên đề của đại số Boole được xếp theo từng cặp a) và b). Từ mỗi tiên đề a), nếu ta thay . bởi +, thay + bởi ., thay 1 bởi 0 và thay 0 bởi 1 thì ta được tiên đề b) tương ứng.

Ta gọi cặp tiên đề a), b) là đối ngẫu của nhau. Do đó nếu ta chứng minh được một định lý trong đại số Boole thì ta có ngay một định lý khác, đối ngẫu của nó, bằng cách thay . và 1 tương ứng bởi + và 0 (và ngược lại). Ta có:

Quy tắc đối ngẫu: Đối ngẫu của một định lý là một định lý.

Định lý:

6. (Tính nuốt)

a) $a.0 = 0$,

b) $a+1 = 1$

7. (Tính lũy đẳng)

a) $a.a = a$,

b) $a+a = a$.

8. (Hệ thức De Morgan)

a) $(a.b)' = a'+b'$,

b) $(a+b)' = a'.b'$.

9. (Hệ thức bù kép)

$(a')' = a$.

10. a) $1' = 0$,

b) $0' = 1$.

11. (Tính hút)

a) $a.(a+b) = a$,

b) $a+(a.b) = a$.

Chú ý: Hệ tiên đề của đại số Boole nêu ra ở đây không phải là một hệ tối thiểu.

1.2.2 Hàm số Boole

Định nghĩa: Ký hiệu $B = \{0, 1\}$ và $B_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in B, 1 \leq i \leq n\}$, ở đây B và B_n là các đại số Boole. Biến x được gọi là một biến Boole nếu nó nhận các giá trị chỉ từ B .

Một hàm từ B_n vào B được gọi là một hàm Boole (hay hàm đại số logic) bậc n . Các hàm Boole cũng có thể được biểu diễn bằng cách dùng các biểu thức được tạo bởi các biến và các phép toán Boole (xem Bảng 1 trong Thí dụ 1). Các biểu thức Boole với các biến x_1, x_2, \dots, x_n được định nghĩa bằng đệ quy như sau:

- $0, 1, x_1, x_2, \dots, x_n$ là các biểu thức Boole.

- Nếu P và Q là các biểu thức Boole thì P, PQ và $P+Q$ cũng là các biểu thức Boole.

Mỗi một biểu thức Boole biểu diễn một hàm Boole. Các giá trị của hàm này nhận được bằng cách thay 0 và 1 cho các biến trong biểu thức đó.

Hai hàm n biến F và G được gọi là bằng nhau nếu $F(a_1, a_2, \dots, a_n) = G(a_1, a_2, \dots, a_n)$ với mọi $a_1, a_2, \dots, a_n \in B$. Hai biểu thức Boole khác nhau biểu diễn cùng một hàm Boole được gọi là tương đương. Phần bù của hàm Boole F là hàm \bar{F} với

$\bar{F}(x_1, x_2, \dots, x_n) = \overline{F(x_1, x_2, \dots, x_n)}$. Giả sử F và G là các hàm Boole bậc n . Tổng Boole $F+G$ và tích Boole FG được định nghĩa bởi:

$$(F+G)(x_1, x_2, \dots, x_n) = F(x_1, x_2, \dots, x_n) + G(x_1, x_2, \dots, x_n),$$

$$(FG)(x_1, x_2, \dots, x_n) = F(x_1, x_2, \dots, x_n)G(x_1, x_2, \dots, x_n).$$

Định nghĩa: Cho x là một biến Boole và $\sigma \in B$. Ký hiệu:

$$x^\sigma = \begin{cases} x & \text{khi } \sigma = 1 \\ \bar{x} & \text{khi } \sigma = 0 \end{cases}$$

Dễ thấy rằng $x^\sigma = 1 \Leftrightarrow x = \sigma$. Với mỗi hàm Boole F bậc n , ký hiệu:

$$T_F = \{(x_1, x_2, \dots, x_n) \in B_n \mid F(x_1, x_2, \dots, x_n) = 1\}$$

Và gọi nó là tập đặc trưng của hàm F . Khi đó ta có:

$$T_F = \overline{T_F}, T_{F+G} = T_F \cup T_G, T_{FG} = T_F \cap T_G$$

Cho n biến Boole x_1, x_2, \dots, x_n . Một biểu thức dạng:

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \dots x_{i_k}^{\sigma_k}$$

trong đó $\sigma_1, \sigma_2, \dots, \sigma_k \in B, 1 \leq i_1 < i_2 < \dots < i_k \leq n$ được gọi là một hội sơ cấp của n biến x_1, x_2, \dots, x_n . Số các biến xuất hiện trong một hội sơ cấp được gọi là hạng của của hội sơ cấp đó.

Cho F là một hàm Boole bậc n . Nếu F được biểu diễn dưới dạng tổng (tuyến) của một số hội sơ cấp khác nhau của n biến thì biểu diễn đó được gọi là dạng tổng (tuyến) chuẩn tắc của F . Dạng tổng (tuyến) chuẩn tắc hoàn toàn là dạng chuẩn tắc duy nhất của F mà trong đó các hội sơ cấp đều có hạng n .

Mệnh đề: Mọi hàm Boole F bậc n đều có thể biểu diễn dưới dạng:

$$F(x_1, x_2, \dots, x_n) = \sum x_1^{\sigma_1} \dots x_i^{\sigma_i} F(\sigma_1, \dots, \sigma_i, x_{i+1}, \dots, x_n) \quad (1)$$

trong đó i là số tự nhiên bất kỳ, $1 \leq i \leq n$.

Hệ quả: Mọi hàm Boole F bậc n đều có thể được khai triển theo một biến x_i :

$$F(x_1, x_2, \dots, x_n) = \overline{x_i} F(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) + x_i F(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$

Cho $i=n$ trong mệnh đề trên và bỏ đi các nhân tử bằng 1 trong tích, các số hạng bằng 0 trong tổng, ta được hệ quả sau:

Mọi hàm Boole F bậc n đều có thể được khai triển dưới dạng:

$$F(x_1, x_2, \dots, x_n) = \sum x_1^{\sigma_1} \dots x_i^{\sigma_i} n$$

$$\sigma_1, \sigma_2, \dots, \sigma_k \in T_F$$

Chú ý: Từ Hệ quả 8.2.5, ta suy ra rằng mọi hàm Boole đều có thể biểu diễn dưới dạng tổng (tuyến) chuẩn tắc hoàn toàn. Như vậy mọi hàm Boole đều có thể biểu diễn bằng một biểu thức Boole chỉ chứa ba phép tích (hội), tổng (tuyến), bù (phủ định). Ta nói rằng hệ {tích, tổng, bù} là đầy đủ. Bằng đối ngẫu, ta có thể chứng minh một kết quả tương tự

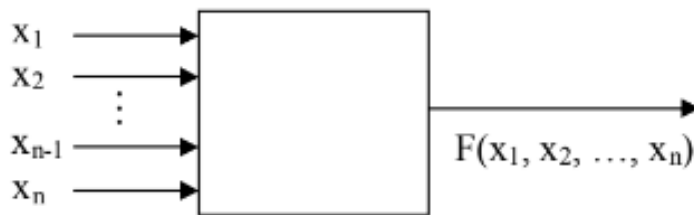
bằng việc thay tích bởi tổng và ngược lại, từ đó dẫn tới việc biểu diễn F qua một tích các tổng. Biểu diễn này được gọi là dạng tích (hội) chuẩn tắc hoàn toàn của F :

$$F(x_1, x_2, \dots, x_n) = \prod x_1^{\sigma_1} \dots x_i^{\sigma_i} n$$

$$\sigma_1, \sigma_2, \dots, \sigma_k \in T_F$$

1.2.3 Mạch logic

1.2.3.1 Cổng logic



Xét một thiết bị như hình trên, có một số đường vào (dẫn tín hiệu vào) và chỉ có một đường ra (phát tín hiệu ra). Giả sử các tín hiệu vào x_1, x_2, \dots, x_n (ta gọi là đầu vào hay input) cũng như tín hiệu ra F (đầu ra hay output) đều chỉ có hai trạng thái khác nhau, tức là mang một bit thông tin, mà ta ký hiệu là 0 và 1.

Ta gọi một thiết bị với các đầu vào và đầu ra mang giá trị 0, 1 như vậy là một mạch logic.

Đầu ra của một mạch logic là một hàm Boole F của các đầu vào x_1, x_2, \dots, x_n . Ta nói mạch logic trong hình trên thực hiện hàm F .

Các mạch logic được tạo thành từ một số mạch cơ sở, gọi là cổng logic. Các cổng logic sau đây thực hiện các hàm phủ định, hội và tuyển.

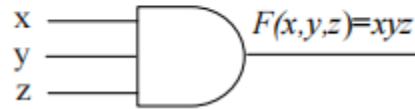
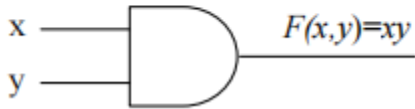
1. Cổng NOT: Cổng NOT thực hiện hàm phủ định. Cổng chỉ có một đầu vào. Đầu ra $F(x)$ là phủ định của đầu vào x .

$$F(x) = \bar{x} = \begin{cases} 0 & \text{khi } x = 1, \\ 1 & \text{khi } x = 0. \end{cases}$$



2. Cổng AND: Cổng AND thực hiện hàm hội. Đầu ra $F(x,y)$ là hội (tích) của các đầu vào.

$$F(x, y) = xy = \begin{cases} 1 & \text{khi } x = y = 1, \\ 0 & \text{trong các trường hợp khác.} \end{cases}$$



3. Cổng OR: Cổng OR thực hiện hàm tuyển (tổng). Đầu ra $F(x,y)$ là tuyển (tổng) của các đầu vào.

$$F(x, y) = x + y = \begin{cases} 1 & \text{khi } x = 1 \text{ hay } y = 1, \\ 0 & \text{khi } x = y = 0. \end{cases}$$



1.2.3.2 Mạch logic

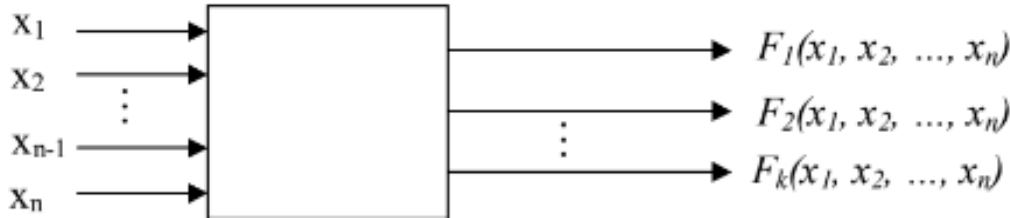
1. Tổ hợp các cổng: Các cổng logic có thể lắp ghép để được những mạch logic thực hiện các hàm Boole phức tạp hơn. Như ta đã biết rằng một hàm Boole bất kỳ có thể biểu diễn bằng một biểu thức chỉ chứa các phép $-$, $..$, $+$. Từ đó suy ra rằng có thể lắp ghép thích hợp các cổng NOT, AND, OR để được một mạch logic thực hiện một hàm Boole bất kỳ.

$$F(x,y) = x \oplus y = \begin{cases} 0 & \text{khi } x = y \\ 1 & \text{khi } x \neq y \end{cases}$$

Mạch logic này là một cổng logic, gọi là cổng XOR, được vẽ như hình dưới đây:



2. Mạch cộng: Nhiều bài toán đòi hỏi phải xây dựng những mạch logic có nhiều đường ra, cho các đầu ra F_1, F_2, \dots, F_k là các hàm Boole của các đầu vào x_1, x_2, \dots, x_n



1.2.4 Cực tiểu hoá các mạch logic

1.2.4.1 Bản đồ Karnaugh

Để làm giảm số các số hạng trong một biểu thức Boole biểu diễn một mạch, ta cần phải tìm các số hạng để tổ hợp lại. Có một phương pháp đồ thị, gọi là bản đồ Karnaugh, được dùng để tìm các số hạng tổ hợp được đối với các hàm Boole có số biến tương đối nhỏ. Phương pháp mà ta mô tả dưới đây đã được Maurice Karnaugh đưa ra vào năm 1953. Phương pháp này dựa trên một công trình trước đó của E.W. Veitch. Các bản đồ Karnaugh cho ta một phương pháp trực quan để rút gọn các khai triển tổng các tích, nhưng chúng không thích hợp với việc cơ khí hoá quá trình này. Trước hết, ta sẽ minh hoạ cách dùng các bản đồ Karnaugh để rút gọn biểu thức của các hàm Boole hai biến.

Có bốn hội sơ cấp khác nhau trong khai triển tổng các tích của một hàm Boole có hai biến x và y . Một bản đồ Karnaugh đối với một hàm Boole hai biến này gồm bốn ô vuông, trong đó hình vu biểu diễn hội sơ cấp có mặt trong khai triển được ghi số 1. Các hình ô được gọi là kề nhau nếu các hội sơ cấp mà chúng biểu diễn chỉ khác nhau một biến.

	Y		\bar{y}
X	xy	$X\bar{y}$	
\bar{x}	$\bar{x}y$	$\bar{x}\bar{y}$	

1.2.4.2 Phương pháp Quine-McCluskey

Cho hai hàm Boole F và G bậc n . Ta nói G là một nguyên nhân của F nếu $T_G \subset T_F$, nghĩa là $G \Rightarrow F$ là một hằng đúng.

Dễ thấy rằng mỗi hội sơ cấp trong một dạng tổng chuẩn tắc của F là một nguyên nhân của F . Hội sơ cấp A của F được gọi là một nguyên nhân nguyên tố của F nếu trong A xoá đi một biến thì hội nhận được không còn là nguyên nhân của F .

Nếu F_1, F_2, \dots, F_k là các nguyên nhân của F thì $T_{F_i} \subset T_F$. Khi đó

$$\sum_{i=1}^{T_k} = \bigcup_{i=1}^k T_{F_i} \subset T_F \text{ Do đó } \sum_{i=1}^{T_k} T_{F_i} \text{ là nguyên nhân của } F.$$

Cho S là một hệ các nguyên nhân của F . Ta nói rằng hệ S là đầy đủ đối với F nếu: $F = \sum_{G \in S} G$, nghĩa là $T_F = \bigcup_{G \in S} T_G$

Dạng tổng chuẩn tắc $F = \sum_{G \in S} G$ được gọi là dạng tổng chuẩn tắc thu gọn của F .

1.2.4.3 Phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc thu gọn:

Giả sử F là một hàm Boole n biến x_1, x_2, \dots, x_n . Mỗi hội sơ cấp của n biến đó được biểu diễn bằng một dãy n ký hiệu trong bảng $\{0, 1, -\}$ theo quy ước: ký tự thứ i là 1 hay 0 nếu x_i có mặt trong hội sơ cấp là bình thường hay với dấu phủ định, còn nếu x_i không có mặt thì ký tự này là $-$. Rõ ràng các hội sơ cấp chỉ có thể dán được với nhau bằng phép dán $Ax + A\bar{x} = A$ nếu chúng là kề nhau.

Thuật toán được tiến hành như sau: Lập một bảng gồm nhiều cột để ghi các kết quả dán. Sau đó lần lượt thực hiện các bước sau:

Bước 1: Viết vào cột thứ nhất các biểu diễn của các nguyên nhân hạng n của hàm Boole F . Các biểu diễn được chia thành từng nhóm, các biểu diễn trong mỗi nhóm có số các ký hiệu 1 bằng nhau và các nhóm xếp theo thứ tự số các ký hiệu 1 tăng dần.

Bước 2: Lần lượt thực hiện tất cả các phép dán các biểu diễn trong nhóm i với các biểu diễn trong nhóm $i+1$ ($i=1, 2, \dots$). Biểu diễn nào tham gia ít nhất một phép dán sẽ được ghi nhận một dấu $*$ bên cạnh. Kết quả dán được ghi vào cột tiếp theo.

Bước 3: Lập lại Bước 2 cho cột kế tiếp cho đến khi không thu thêm được cột nào mới. Khi đó tất cả các biểu diễn không có dấu * sẽ cho ta tất cả các nguyên nhân nguyên tố của F.

1.2.4.4 Phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc tối thiểu:

Sau khi tìm được dạng tổng chuẩn tắc thu gọn của hàm Boole F, nghĩa là tìm được tất cả các nguyên nhân nguyên tố của nó, ta tiếp tục phương pháp Quine-McCluskey tìm dạng tổng chuẩn tắc tối thiểu (cực tiểu) của F như sau.

Lập một bảng chữ nhật, mỗi cột ứng với một cấu tạo đơn vị của F (mỗi cấu tạo đơn vị là một hội sơ cấp hạng n trong dạng tổng chuẩn tắc hoàn toàn của F) và mỗi dòng ứng với một nguyên nhân nguyên tố của F. Tại ô (i, j), ta đánh dấu cộng (+) nếu nguyên nhân nguyên tố ở dòng i là một phần con của cấu tạo đơn vị ở cột j. Ta cũng nói rằng khi đó nguyên nhân nguyên tố i là phủ cấu tạo đơn vị j. Một hệ S các nguyên nhân nguyên tố của F được gọi là phủ hàm F nếu mọi cấu tạo đơn vị của F đều được phủ ít nhất bởi một thành viên của hệ. Dễ thấy rằng nếu hệ S là phủ hàm F thì nó là đầy đủ, nghĩa là tổng của các thành viên trong S là bằng F.

Một nguyên nhân nguyên tố được gọi là cốt yếu nếu thiếu nó thì một hệ các nguyên nhân nguyên tố không thể phủ hàm F. Các nguyên nhân nguyên tố cốt yếu được tìm như sau: tại những cột chỉ có duy nhất một dấu +, xem dấu + đó thuộc dòng nào thì dòng đó ứng với một nguyên nhân nguyên tố cốt yếu. Việc lựa chọn các nguyên nhân nguyên tố trên bảng đã đánh dấu, để được một dạng tổng chuẩn tắc tối thiểu, có thể tiến hành theo các bước sau.

Bước 1: Phát hiện tất cả các nguyên nhân nguyên tố cốt yếu.

Bước 2: Xóa tất cả các cột được phủ bởi các nguyên nhân nguyên tố cốt yếu.

Bước 3: Trong bảng còn lại, xóa nốt những dòng không còn dấu + và sau đó nếu có hai cột giống nhau thì xóa bớt một cột.

Bước 4: Sau các bước trên, tìm một hệ S các nguyên nhân nguyên tố với số biến ít nhất phủ các cột còn lại.

Tổng của các nguyên nhân nguyên tố cốt yếu và các nguyên nhân nguyên tố trong hệ S sẽ là dạng tổng chuẩn tắc tối thiểu của hàm F.

Các bước 1, 2, 3 có tác dụng rút gọn bảng trước khi lựa chọn. Độ phức tạp chủ yếu nằm ở Bước 4. Tình huống tốt nhất là mọi nguyên nhân nguyên tố đều là cốt yếu. Trường hợp này không phải lựa chọn gì và hàm F có duy nhất một dạng tổng chuẩn tắc tối thiểu cũng chính là dạng tổng chuẩn tắc thu gọn. Tình huống xấu nhất là không có nguyên nhân nguyên tố nào là cốt yếu. Trường hợp này ta phải lựa chọn toàn bộ bảng.

Phần 2. Thực hành

2.1 Thuật toán BFS

1.1.1 Khái niệm

Thuật toán BFS (Breadth-First Search) là một thuật toán trong trí tuệ nhân tạo, được sử dụng để vẽ biểu đồ thị dữ liệu hoặc tìm kiếm cấu trúc cây hoặc duyệt. Hình thức đầy đủ của BFS là tìm kiếm theo chiều rộng.

1.1.2 Mô tả

Bước 1: Xuất phát từ đỉnh số 5 trong đồ thị cho trước.

Bước 2: Xử lý đỉnh này và đánh dấu để không xử lý lần sau.

Bước 3: Đưa tất cả các đỉnh kề với nó vào danh sách xử lý và lần lượt xử lý các đỉnh kề với đỉnh đang xét

Bước 4: Quay lại B2 cho đến khi không còn đỉnh trong danh sách.

1.1.3 Tính độ phức tạp

Phức tạp thời gian: $O(|V| + |E|) = O(b^d)$

Phức tạp dữ liệu: $O(|V|) = O(b^d)$

1.1.4 Kiểm nghiệm thuật toán bằng ví dụ thực tế:

Cho đồ thị $G = \langle V, E \rangle$ được biểu diễn dưới dạng ma trận kề như bên dưới. Hãy áp dụng thuật toán BFS() duyệt tất cả các đỉnh kề với đỉnh số 5.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	0	0	1	1	0	0	0	0	0	0	0
2	1	0	0	0	1	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	1	0	0	0	1	0	0
4	0	0	1	0	1	0	1	1	0	0	1	0	0
5	1	1	0	1	0	1	0	0	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0	0	0
7	0	0	1	1	0	0	0	1	0	0	0	0	0
8	0	0	0	1	0	0	1	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	1	0	1	1
10	0	0	0	0	0	0	0	0	1	0	0	1	1
11	0	0	1	1	0	0	0	1	0	0	0	1	0
12	0	0	0	0	0	0	0	0	1	1	1	0	1
13	0	0	0	0	0	0	0	0	1	1	0	1	0

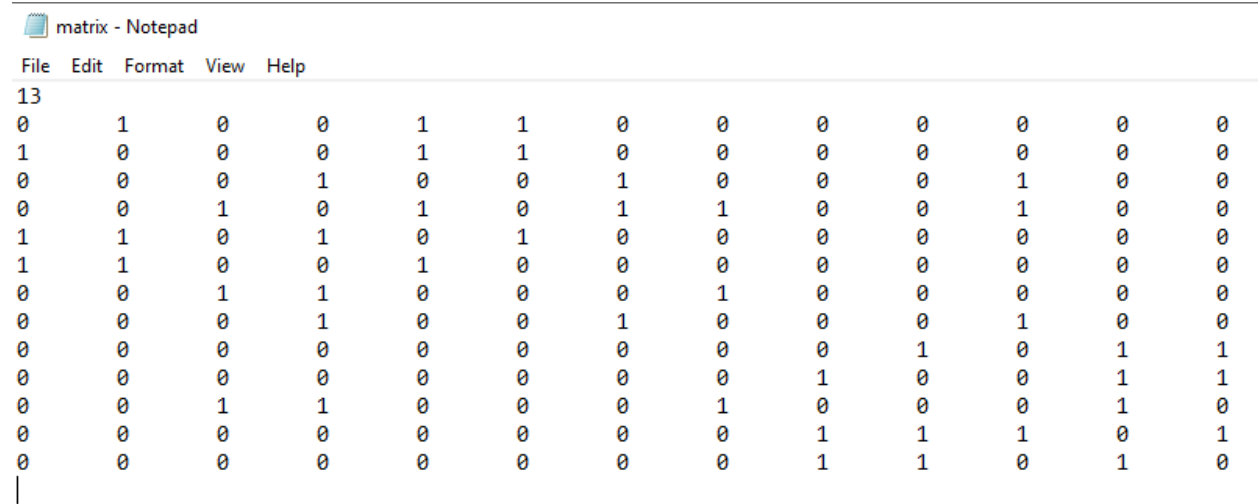
- Với $u = 5$ ta có bảng sau:

Bước	Các đỉnh trong hàng đợi	Các đỉnh được duyệt
1	5	\emptyset
2	1,2,4,6	5
3	2,4,6	5,1
4	4,6	5,1,2
5	6,3,7,8,11	5,1,2,4
6	3,7,8,11	5,1,2,4,6
7	7,8,11	5,1,2,4,6,3
8	8,11	5,1,2,4,6,3,7
9	11	5,1,2,4,6,3,7,8
10	12	5,1,2,4,6,3,7,8,11
11	9,10,13	5,1,2,4,6,3,7,8,11,12
12	10,13	5,1,2,4,6,3,7,8,11,12,9
13	13	5,1,2,4,6,3,7,8,11,12,9,10
14	\emptyset	5,1,2,4,6,3,7,8,11,12,9,10,13

Vậy $\text{BFS}(5) = 5,1,2,4,6,3,7,8,11,12,9,10,13$

1.1.5 Cài đặt

Tạo một file ma trận kề để input



```
matrix - Notepad
File Edit Format View Help
13
0 1 0 0 1 1 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 1 0 0
0 0 1 0 1 0 1 1 0 0 1 0 0
1 1 0 1 0 1 0 0 0 0 0 0 0
1 1 0 0 1 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 1 1
0 0 0 0 0 0 0 0 1 0 0 1 1
0 0 1 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 1 1 1 0 1
0 0 0 0 0 0 0 0 1 1 0 1 0
```

CODE thuật toán BFS

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

int n;
int matrix[1001][1001];
bool visited[1001];

void Init()
{
    ifstream file("matrix.txt", ios::in);
    file >> n;
```

```
for(int i = 1; i <= n; i++)
{
    for(int j = 1; j <= n; j++)
    {
        file >> matrix[i][j];
    }
    visited[i] = false;
}

void BFS(int u)
{
    cout << "BFS(" << u << "): ";
    queue<int> Q;
    Q.push(u);
    visited[u] = true;
    while(!Q.empty())
    {
        int s = Q.front();
        cout << s << " ";
        Q.pop();
        for(int t = 1; t <= n; t++)
        {
            if(visited[t] == false && matrix[s][t] == 1)
```

```
        {
            Q.push(t);
            visited[t] = true;
        }
    }
}

int main()
{
    int u;
    Init();
    cout << "-----TOAN ROI RAC-----\n";
    cout << "Duyet do thi theo chieu rong BFS\n";
    cout << "Chon dinh bat dau: ";
    cin >> u;
    BFS(u);
    cout << "\n-----";
    cout << "\n-----@Code by Trinh Cam Nhung-----";
    return 0;
}
```

Demo chương trình thuật toán BFS

```
"C:\Users\User\Documents\BAI TAP CHUNG\BFS1\BFS1\BFS.exe"
-----TOAN ROI RAC-----
Duyet do thi theo chieu rong BFS
Chon dinh bat dau: 5
BFS(5): 5 1 2 4 6 3 7 8 11 12 9 10 13
-----
-----@Code by Trinh Cam Nhung-----
Process returned 0 (0x0)   execution time : 4.675 s
Press any key to continue.
_
```

2.2 Thuật toán DFS

2.2.1 Khái niệm

Thuật toán DFS (*Depth-first search*) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

2.2.2 Mô tả

Bước 1: Xuất phát từ đỉnh 5 cho trước.

Bước 2: Xử lý đỉnh này và đánh dấu để không xử lý lần sau.

Bước 3: Lấy đỉnh đầu tiên từ ngăn xếp, nếu đỉnh đó có đỉnh kề chưa được thăm, thăm đỉnh đó bằng cách đánh dấu đã thăm và đưa vào ngăn xếp.

Bước 4: Quay lại B2 cho đến khi không còn đỉnh trong danh sách.

2.2.3 Tính độ phức tạp

Độ phức tạp thời gian: $O(|V| + |E|)$

Độ phức tạp dữ liệu: $O(|V|)$

2.2.4 Kiểm nghiệm thuật toán bằng ví dụ thực tế

Cho đồ thị $G = \langle V, E \rangle$ được biểu diễn dưới dạng ma trận kề như bên dưới. Hãy áp dụng thuật toán BFS() duyệt tất cả các đỉnh kề với đỉnh số 5.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	0	0	1	1	0	0	0	0	0	0	0
2	1	0	0	0	1	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	1	0	0	0	1	0	0
4	0	0	1	0	1	0	1	1	0	0	1	0	0
5	1	1	0	1	0	1	0	0	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0	0	0
7	0	0	1	1	0	0	0	1	0	0	0	0	0
8	0	0	0	1	0	0	1	0	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0	1	0	1	1
10	0	0	0	0	0	0	0	0	1	0	0	1	1
11	0	0	1	1	0	0	0	1	0	0	0	1	0
12	0	0	0	0	0	0	0	0	1	1	1	0	1
13	0	0	0	0	0	0	0	0	1	1	0	1	0

Với $u = 5$ ta có bảng sau:

Các đỉnh trong hàng chờ	Các đỉnh được duyệt	Các đỉnh chưa được duyệt
DFS(5)	5	1,2,3,4,6,7,8,9,10,11,12,13
DFS(1)	5,1	2,3,4,6,7,8,9,10,11,12,13
DFS(2)	5,1,2	3,4,6,7,8,9,10,11,12,13
DFS(6)	5,1,2,6	3,4,7,8,9,10,11,12,13
DFS(4)	5,1,2,6,4	3,7,8,9,10,11,12,13
DFS(3)	5,1,2,6,4,3	7,8,9,10,11,12,13
DFS(7)	5,1,2,6,4,3,7	8,9,10,11,12,13
DFS(8)	5,1,2,6,4,3,7,8	9,10,11,12,13
DFS(11)	5,1,2,6,4,3,7,8,11	9,10,12,13
DFS(12)	5,1,2,6,4,3,7,8,11,12	9,10,13
DFS(9)	5,1,2,6,4,3,7,8,11,12,9	10,13
DFS(10)	5,1,2,6,4,3,7,8,11,12,9,10	13
DFS(13)	5,1,2,6,4,3,7,8,11,12,9,10,13	\emptyset

2.2.5 Cài đặt

Tạo một file ma trận kề để input

matrix - Notepad

File Edit Format View Help

13

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

CODE Thuật toán DFS

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

int n; // so dinh cua do thi.
int matrix[1001][1001];
bool visited[1001];

void Init()
{
    ifstream file("matrix.txt", ios::in);
    file >> n;
    for(int i = 1; i <= n; i++)
    {
```

```
for(int j = 1; j <= n; j++)
{
    file >> matrix[i][j];
}
visited[i] = false;
}
}

void DFS(int u)
{
    cout << "DFS(" << u << "): ";
    stack<int> st;
    st.push(u);
    visited[u] = true;
    cout << u << " ";
    while(!st.empty())
    {
        int v = st.top();
        st.pop();
        for(int t = 1; t <= n; t++)
        {
            if(visited[t] == false && matrix[v][t] == 1)
            {
                cout << t << " ";
                visited[t] = true;
            }
        }
    }
}
```

```
        st.push(v);
        st.push(t);
        break;
    }
}
}
}

int main()
{
    int s; // dinh bat dau.
    Init();
    cout << "-----TOAN ROI RAC-----\n";
    cout << "Duyet do thi theo chieu sau DFS\n";
    cout << "Chon dinh bat dau: ";
    cin >> s;
    DFS(s);
    cout << "\n-----";
    cout << "\n-----@Code by Trinh Cam Nhung-----";
    return 0;
}
```

Demo chương trình thuật toán DFS

"C:\Users\User\Documents\BAI TAP CHUNG\DFS1\DFS1\DFS.exe"

```
-----TOAN ROI RAC-----  
Duyet do thi theo chieu sau DFS  
Chon dinh bat dau: 5  
DFS(5): 5 1 2 6 4 3 7 8 11 12 9 10 13  
-----  
-----@Code by Trinh Cam Nhung-----  
Process returned 0 (0x0)   execution time : 2.519 s  
Press any key to continue.
```

2.3 Tự chọn

2.3.1 Thuật toán Kruskal xây dựng cây khung nhỏ nhất

2.3.1.1 Khái niệm

Cây khung của đồ thị: cho $G = \langle V, E \rangle$ là đồ thị vô hướng. T là đồ thị con khung của G .

Nếu T là một cây thì T được gọi là cây khung của đồ thị G

Cây khung nhỏ nhất: cho G là đồ thị có trọng số. Cây khung nhỏ nhất (cây khung tối thiểu) nếu nó có trọng lượng nhỏ nhất

Thuật toán Kruskal là một thuật toán trong lý thuyết đồ thị để tìm cây bao trùm nhỏ nhất của một đồ thị liên thông có trọng số. Nói cách khác, nó tìm một tập hợp các cạnh tạo thành một cây chứa tất cả các đỉnh của đồ thị và có tổng trọng số các cạnh là nhỏ nhất. Thuật toán Kruskal là một ví dụ của thuật toán tham lam.

Thuật toán này xuất bản lần đầu tiên năm 1956, bởi Joseph Kruskal

2.3.1.2 Mô tả

Thuật toán sẽ xây dựng tập cạnh T của cây khung nhỏ nhất $H = \langle V, T \rangle$ theo từng bước sau:

Bước 1: Sắp xếp các cạnh của đồ thị G theo thứ tự tăng dần của trọng số cạnh

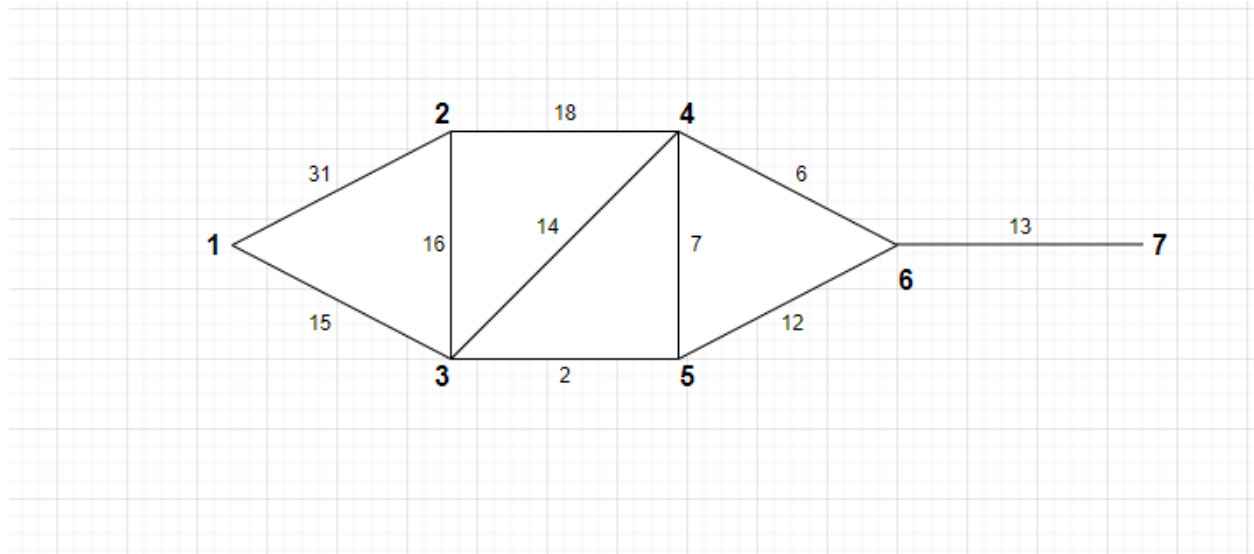
Bước 2: Xuất phát từ tập cạnh $T = \emptyset$, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách các cạnh đã sắp xếp, từ cạnh có trọng số nhỏ nhất đến cạnh có trọng số lớn nhất để tìm ra

cạnh mà khi bổ sung nó vào T không tạo thành chu trình trong tập các cạnh đã được bổ sung vào trước đó.

Bước 3: Thuật toán sẽ kết thúc khi thu được tập T gồm $n-1$ cạnh

2.3.1.3 Kiểm nghiệm thuật toán bằng ví dụ thực tế

Tìm cây khung nhỏ nhất của đồ thị sau bằng thuật toán Kruskal



Khởi tạo: đặt $T = \emptyset$, sắp xếp các cạnh đồ thị theo không gian có dãy

$(3,5); (4,6); (4,5); (5,6); (6,7); (4,3); (1,3); (2,3); (2,4); (1,2)$

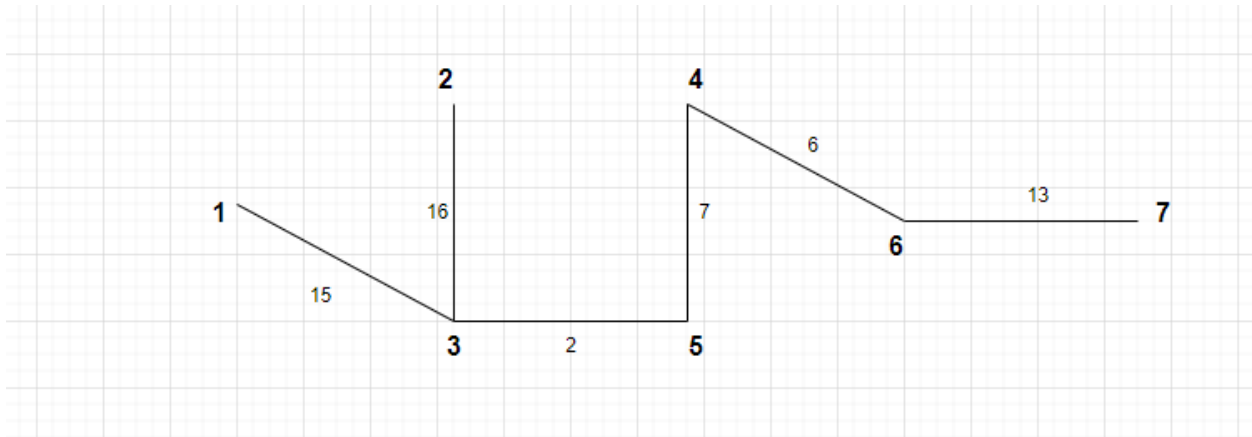
Dãy độ dài tương ứng:

2,6,7,12,13,14,15,16,18,31

Ở ba lần lặp đầu bổ sung vào tập T cạnh $(3,5); (4,6); (4,5)$. Rõ ràng thêm cạnh $(5,6)$ vào T thì nó sẽ tạo với hai cạnh $(4,5); (4,6)$ một chu trình nên loại hai cạnh này. Tiếp theo ta bổ sung cạnh $(6,7)$ vào T. Nhận thấy khi thêm cạnh $(4,3)$ vào dãy cũng sẽ tạo thêm một chu trình nên loại cạnh này. Tiếp theo ta bổ sung cạnh $(1,3); (2,3)$ vào T. Hai cạnh còn lại là $(2,4)$ và $(1,2)$ khi thêm vào cũng tạo thành một chu trình nên loại. Sau cùng ta thu được tập T gồm 6 cạnh:

$T = (3,5); (4,6); (4,5); (6,7); (1,3); (2,3)$

T chính là tập cạnh của cây khung nhỏ nhất



Độ dài cây khung nhỏ nhất bằng: $2+7+6+13+15+16=59$

2.3.1.4 Cài đặt

*Tạo file input bao gồm ba cột: điểm đầu, điểm cuối, trọng số

input - Notepad

Tệp Soạn thảo Định dạng Xem Trợ giúp

```
7 10
1 2 31
1 3 15
2 3 16
2 4 18
3 4 14
3 5 2
4 5 7
4 6 6
5 6 12
6 7 13
```

CODE xử lý thuật toán kruskal

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
struct edge{
```



```
int u, v;

int w;

};

const int maxn = 1001;

int n, m;

int parent[maxn], sz[maxn];

vector<edge> canh;

void make_set(){
    for(int i = 1; i <= n; i++){
        parent[i] = i;
        sz[i] = 1;
    }
}

int find(int v){
    if(v == parent[v]) return v;
    return parent[v] = find(parent[v]);
}

bool Union(int a, int b){
```

```
a = find(a);

b = find(b);

if(a == b) return false; // không thể gộp a, b vào với nhau

if(sz[a] < sz[b]) swap(a, b);

parent[b] = a;

sz[a] += sz[b];

return true;
}

void inp(){
    cin >> n >> m;

    for(int i = 0; i < m; i++){
        int x, y, w; cin >> x >> y >> w;

        edge e;

        e.u = x; e.v = y; e.w = w;

        canh.push_back(e);
    }
}

bool cmp(edge a, edge b){
    return a.w < b.w;
}
```

```
void kruskal(){
    //Tao cay khung cuc tieu rong
    vector<edge> mst;
    int d = 0;

    //Sort danh sach canh theo chieu dai tang dan
    sort(canh.begin(), canh.end(), cmp);

    //Buoc 3 lap
    for(int i = 0; i < m; i++){
        if(mst.size() == n - 1) break;

        edge e = canh[i]; // chon canh e la canh nho nhat
        if(Union(e.u, e.v)){
            mst.push_back(e);
            d += e.w;
        }
    }

    //Tra ve ket qua
    if(mst.size() != n - 1){
        cout << "Do thi khong lien thong !\n";
    }
    else{
```

```
        cout << "MST :" << d << endl;

        for(auto it : mst){

            cout << it.u << " " << it.v << " " << it.w << endl;

        }

    }

}

int main(){

    #ifndef ONLINE_JUDGE

    freopen("input.txt","r",stdin);

    freopen("output.txt","w",stdout);

    #endif // ONLINE_JUDGE

    inp();

    make_set();

    kruskal();

}
```

Sau khi run chương trình sẽ xuất ra một file bao gồm đường đi ngắn nhất, các cặp trước sau

output - Notepad

Tệp Soạn thảo Định dạng Xem Trợ giúp

MST : 59

3 5 2

4 6 6

4 5 7

6 7 13

1 3 15

2 3 16

2.3.2 Thuật toán Floyd

2.3.2.1 Khái niệm

Thuật toán Floyd–Warshall còn được gọi là thuật toán Floyd được Robert Floyd tìm ra năm 1962. thuật toán Floyd là một thuật toán giải quyết bài toán đường đi ngắn nhất trong một đồ thị có hướng dựa trên khái niệm các Đỉnh Trung Gian.

2.3.2.2 Mô tả

Từ ma trận trọng số $A[u,v]$, $u,v \in V$, ta tìm cận trên $d[v]$ của khoảng cách từ s đến tất cả các đỉnh $v \in V$. Mỗi khi phát hiện thấy $d[u] + A[u,v] < d[v]$ thì cận trên $d[v]$ sẽ được làm tốt lên bằng cách gán $d[v] = d[u] + A[u, v]$. Quá trình sẽ kết thúc khi nào ta không thể làm tốt hơn lên được bất kỳ cận trên nào, khi đó $d[v]$ sẽ cho ta giá trị ngắn nhất từ đỉnh s đến đỉnh v . Giá trị $d[v]$ được gọi là nhãn của đỉnh v .

2.3.2.3 Kiểm nghiệm bài toán bằng ví dụ thực tế

Tìm đường đi ngắn nhất từ đỉnh A đến đỉnh I trên đồ thị hình sau:

Bước 1. Gán cho nhãn đỉnh A là 0;

Bước 2. Trong số các cạnh (cung) xuất phát từ A, ta chọn cạnh có độ dài nhỏ nhất, sau đó gán nhãn cho đỉnh đó bằng nhãn của đỉnh A cộng với độ dài cạnh tương ứng. Ta chọn được đỉnh C có trọng số $AC = 5$, nhãn $d[C] = 0 + 5 = 5$.

Bước 3. Tiếp đó, trong số các cạnh (cung) đi từ một đỉnh có nhãn là A hoặc C tới một đỉnh chưa được gán nhãn, ta chọn cạnh sao cho nhãn của đỉnh cộng với trọng số cạnh tương ứng là nhỏ nhất gán cho nhãn của đỉnh cuối của cạnh. Như vậy, ta lần lượt gán được các nhãn như sau: $d[B] = 6$ vì $d[B] < d[C] + |CB| = 5 + 4$; $d[E] = 8$; Tiếp tục làm như vậy cho tới khi đỉnh I được gán nhãn đó chính là độ dài đường đi ngắn nhất từ A đến I. Thực chất, nhãn

của mỗi đỉnh chính là đường đi ngắn nhất từ đỉnh nguồn tới nó. Quá trình có thể được mô tả như trong bảng dưới đây.

Bước	Đỉnh được gán nhãn	Nhãn các đỉnh	Đỉnh đã dùng để gán nhãn
Khởi tạo	A	0	A
1	C	$0+5=5$	A
2	B	$0+6=6$	A
3	E	$0+8=8$	A
4	D	$5+4=9$	C
5	F	$5+7=13$	B
6	H	$8+6=14$	E
7	G	$9+6=15$	D
8	I	$15+3=18$	G

Như vậy, độ dài đường đi ngắn nhất từ A đến I là 18. Đường đi ngắn nhất từ A đến I qua các đỉnh: A-> C-> D -> G -> I.

2.3.2.4 Cài đặt

*Tạo file nhập vào chương trình có tên FLOYD.IN

```
FLOYD - Notepad
Tệp  Soạn thảo  Định dạng  Xem  Trợ giúp
b
1 9
0 6 5 0 8 0 0 0 0
6 0 4 0 0 7 0 0 0
5 4 0 4 0 0 0 0 0
0 0 4 0 4 0 6 0 0
8 0 0 4 0 0 0 6 0
0 7 0 0 0 0 5 0 6
0 0 0 6 0 5 0 4 3
0 0 0 0 6 0 4 0 5
0 0 0 0 0 6 3 5 0
```

CODE chương trình thuật toán FLOYD

```
#include<iostream>

#include<conio.h>

using namespace std;

#define MAX 10000

#define TRUE 1

#define FALSE 0

int A[50][50]; // ma trận trọng số của đồ thị.

int D[50][50]; // là mảng chứa các giá trị khoảng cách ngắn nhất từ i đến j

int S[50][50]; // S là mảng chứa giá trị phần tử ngay sau của i trên đường đi ngắn nhất
từ i->j */

int n; // số đỉnh của đồ thị

int u; // đỉnh đầu của đường đi

int v; // đỉnh cuối của đường đi

void Init(void){

freopen("FLOYD.IN", "r", stdin);

cin>>n>>u>>v;
```

```
cout<<"Số đỉnh của đồ thị: "<< n <<endl;
```

```
//nhập ma trận trọng số
```

```
for (int i = 1; i <= n; i++){
```

```
    for (int j = 1; j <= n; j++){
```

```
        cin>>A[i][j];
```

```
        if (i != j && A[i][j] == 0)
```

```
            A[i][j] = MAX;
```

```
    }
```

```
}
```



```
}

void Result(void){

    if (D[u][v] >= MAX) {

        cout<<"Khong co duong di";

    }

    else {

        cout<<"Duong di ngan nhat tu "<<(char)(u+'A'-1)<<" den "<<(char)(v + 'A' -1)<<
        " co do dai la "<<D[u][v]<<endl;

        cout<<"Duong di: "<<(char)(u + 'A' - 1);//in đỉnh đầu dưới dạng char.

        while (u != v) {
```

```
cout<<"->"<<(char)(S[u][v] + 'A' - 1); //in ra đường đi dưới dạng char.
```

```
u = S[u][v];
```

```
}
```

```
}
```

```
}
```

```
void Floy(void){
```

```
int i, j, k, found;
```

```
for (i = 1; i <= n; i++){
```

```
for (j = 1; j <= n; j++){
```

```
D[i][j] = A[i][j];
```

```
if (D[i][j] == MAX) S[i][j] = 0;
```

```
else S[i][j] = j;
```

```
}
```

```
}
```

/* Mang D[i,j] la mang chua cac gia tri khoan cach ngan nhat tu i den j

Mang S la mang chua gia tri phan tu ngay sau cua i tren duong di

ngan nhat tu i->j */

```
for (k = 1; k <= n; k++){
```

```
    for (i = 1; i <= n; i++){
```

```
for (j = 1; j <= n; j++){  
  
    if (D[i][k] != MAX && D[i][j] > (D[i][k] + D[k][j])){  
  
        // Tim D[i,j] nho nhat co the co  
  
        D[i][j] = D[i][k] + D[k][j];  
  
        S[i][j] = S[i][k];  
  
        //ung voi no la gia tri cua phan tu ngay sau i  
  
    }  
  
}  
  
}
```

```
}

int main(void){

    Init();

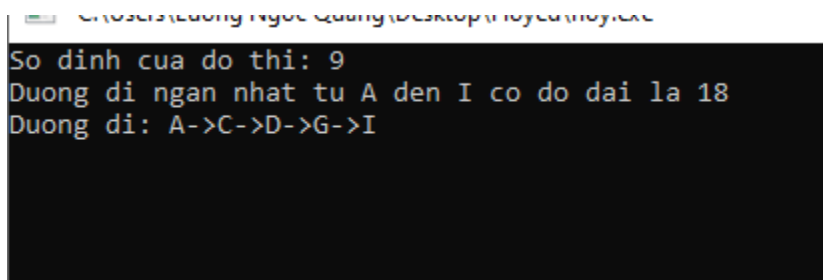
    Floy();

    Result();

    getch();

}
```

Demo chương trình



```
C:\Users\Duong Ngoc Quang\Desktop\toyca\toyca.exe
So dinh cua do thi: 9
Duong di ngan nhat tu A den I co do dai la 18
Duong di: A->C->D->G->I
```