

CHƯƠNG 2. CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Đặt bài toán:

Input: Đồ thị $G = (V, E)$ gồm n đỉnh, m cạnh;

Một đỉnh $u \in G$;

Output: Thứ tự thăm các đỉnh $v \in G$ bắt đầu từ đỉnh u ;

cuu duong than cong . com

cuu duong than cong . com

2.1 Thuật toán tìm kiếm theo chiều sâu (Depth - first Search)

2.1.1 Giới thiệu thuật toán

- Bước khởi tạo: Tất cả các đỉnh $v \in G$ chưa được xét ($vs[v]=0$);
- Bước 1: Tìm kiếm theo chiều sâu bắt đầu từ $v = u$ bằng cách thăm v và đánh dấu v được xét ($vs[v] = 1$);
- Bước 2: Chọn một đỉnh t kề với v và chưa được xét;
- Bước 3: Nếu chọn được t thì quay lại bước 1 với t đóng vai trò u ;
- Bước 4: Nếu không chọn được t thì quay lại bước 2 và đỉnh đóng vai trò v là đỉnh i có thứ tự duyệt ngay trước v ;
- Bước 5: Nếu tất cả các đỉnh kề của u đều đã được xét thì dừng;

2.1.2 Mô tả thuật toán

Thuật toán: DfsDequy(u){
 Thăm(u);
 vs[u] = 1;
 for v ∈ ke(u) do
 if (vs[v] = 0) DfsDequy(v);
 }

cuu duong than cong . com

2.1.3 Cài đặt và kiểm nghiệm thuật toán

Cài đặt 1: (Đệ quy)

```
// G cho bởi ma trận kề a[i][j]
int a[100][100], vs[100], n, u;
void DfsDequy(int u)
{ int v;
  cout << u << " ";
  vs[u]= 1;
  for (v= 1; v<=n; v++)
    if (vs[v]==0 && a[u][v]==1) DfsDequy(v);
}
```

Cài đặt 2: (Sử dụng ngăn xếp)

```
// G cho bởi ma trận kề a[i][j]
int vs[100], n, u, s[100];
void DfsNx(int u)
{ int top = 1; s[top] = u; vs[u] = 1;
  while (top > 0){
  int v = s[top];
  for (int i= 1; i<=n; i++) { int ok = 1;
    if (vs[i]==0 && a[v][i]==1) {
    top++; s[top] = i; vs[i] = 1; int ok = 0;
    break;    }
    if (ok) top--;
  }
}
```

2.2 Thuật toán tìm kiếm theo chiều rộng (Breadth - first Search)

2.2.1 Giới thiệu thuật toán

- Bước khởi tạo: Tất cả các đỉnh $v \in G$ chưa được xét ($vs[v]=0$);
- Bước 1 : Xây dựng hàng đợi q bắt đầu từ u và đánh dấu u đã xét ($vs[u]=1$);
- Bước 2 (lặp): Nếu q rỗng thì kết thúc. Ngược lại, lấy v ra khỏi hàng đợi và thăm v ;
- Bước 3: Đưa vào hàng đợi tất cả các đỉnh i kề với v và chưa được xét, đánh dấu i đã xét ($vs[i]=1$);
- Bước 4: Quay lại bước 2.

2.2.2 Mô tả thuật toán

Thuật toán: Bfs(u){

q = \emptyset ;

<Đưa u vào q>;

vs[u] = 1;

while q $\neq \emptyset$ { <Lấy v ra khỏi q>; Thăm(v);

for i \in ke(v) do

if (vs[i] = 0)

{<Đưa i vào q>; vs[i] = 1;

}

}

}

2.2.3 Cài đặt và kiểm nghiệm thuật toán

```
// G cho bởi ma trận kề a[i][j]
int vs[100], n, u, q[100];
void bfs(int u)
{ int v, dq = cq = 1; q[cq] = u; vs[u] = 1;
  while (dq <= cq){
    v = q[dq]; dq++; cout << v << " ";
    for (int i= 1; i<=n; i++)
      if (vs[i]==0 && a[v][i]==1) {
        cq++; q[cq] = i; vs[i] = 1;
      }
  }
}
```


2.3 Ứng dụng của các thuật toán tìm kiếm trên đồ thị

2.3.1 Duyệt tất cả các đỉnh của đồ thị

Input: Đồ thị $G = (V, E)$ gồm n đỉnh, m cạnh;

Output: Thứ tự thăm tất cả các đỉnh $v \in G$ bắt đầu từ đỉnh 1;

cuu duong than cong . com

cuu duong than cong . com

Giải thuật 1: Duyệt tất cả các đỉnh của đồ thị;

- Bước khởi tạo: Tất cả các đỉnh $v \in G$ chưa được thăm ($vs[v]=0$);
- Bước 1: Nếu tất cả các đỉnh đều được thăm thì kết thúc;
- Bước 2: Chọn $v \in G$ chưa được thăm (bắt đầu từ $v = 1$);
- Bước 3: Duyệt theo chiều sâu/chiều rộng (DFS/BFS) bắt đầu từ v ;
- Bước 4: Quay lại bước 1;

cuu duong than cong . com

cuu duong than cong . com

Cài đặt 1 (Sử dụng DFS):

```
void Duyệt1(){int v;  
    for (v = 1; v <= n; v++) vs[v] = 0;  
    for (v = 1; v <= n; v++)  
        if (vs[v] == 0) dfsDequy(v);  
}
```

cuu duong than cong . com

cuu duong than cong . com

Cài đặt 2 (Sử dụng BFS):

```
void Duyet2(){int v;  
    for (v = 1; v <= n; v++) vs[v] = 0;  
    for (v = 1; v <= n; v++)  
        if (vs[v] == 0) Bfs(v);  
}
```

cuu duong than cong . com

cuu duong than cong . com

2.3.2 Tính liên thông của đồ thị

1) Đường đi trên đồ thị

Khái niệm

- Đường đi độ dài k từ u tới $v \in G$ là dãy các đỉnh x_0, x_1, \dots, x_k , trong đó $x_0 = u, x_n = v$ và $(x_{i-1}, x_i) \in E, 1 \leq i \leq k$.
- Đường đi là chu trình nếu bắt đầu và kết thúc tại cùng một đỉnh, tức là $u = v$.
- Đường đi hoặc chu trình là đơn nếu không chứa một cạnh quá một lần.
- Đường đi là đường đi sơ cấp nếu đi qua các đỉnh không quá một lần, trừ đỉnh đầu và đỉnh cuối.
- Đường đi sơ cấp có đỉnh đầu và đỉnh cuối trùng nhau được là chu trình sơ cấp.

Đếm đường đi giữa các đỉnh

Định lý. Cho G là đồ thị với ma trận kề A gồm n đỉnh đánh số $1, 2, \dots, n$. Số các đường đi khác nhau độ dài r từ i đến j , r nguyên dương, bằng giá trị của phần tử (i, j) của ma trận A^r .

Chứng minh.

Theo định nghĩa, a_{ij} tại vị trí (i, j) của ma trận A chính là số đường đi độ dài 1 từ đỉnh i đến đỉnh $j \Rightarrow$ với $r = 1$ định lý đúng.

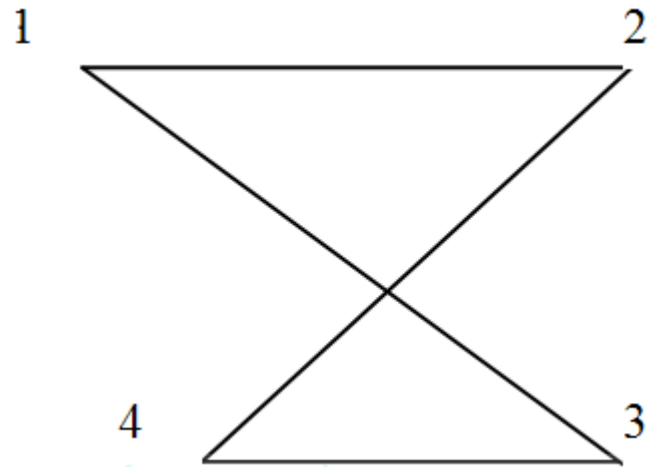
Giả sử định lý đúng với $r = k \Rightarrow A^k(i, j)$ là số đường đi khác nhau có độ dài k từ i đến j .

Chứng minh định lý đúng với $r = k+1$. Vì $A^{k+1} = A^k A \Rightarrow A^k(i, j) = b_{i1}a_{1j} + b_{i2}a_{2j} + \dots + b_{in}a_{nj}$, trong đó $b_{ih} = A^k(i, h)$. Theo giả thiết quy nạp, b_{ih} là số đường đi độ dài k từ i đến h .

Đường đi độ dài $k+1$ từ i đến j sẽ được tạo bởi đường đi độ dài k từ i đến đỉnh trung gian h và một cạnh từ h đến j .

Theo quy tắc nhân, số các đường đi như thế là tích của số đường đi độ dài k từ i đến h tức là b_{ih} và số các cạnh từ h tới j , tức là a_{hj} . Khi cộng các tích này lại theo tất cả các đỉnh trung gian ta được kết quả mong muốn.

Ví dụ. Có bao nhiêu đường đi độ dài 4 từ đỉnh 1 tới đỉnh 4 trong đồ thị G dưới đây:



Giải:

Có ma trận kề của G:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \Rightarrow A^4 = \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix} \Rightarrow \text{có 8 đường đi từ đỉnh 1 đến 4.}$$

2) Tìm đường đi trên đồ thị bắt đầu từ đỉnh u

Input: Đồ thị $G = (V, E)$ gồm n đỉnh, m cạnh;

Một đỉnh $u \in G$;

Output: Đường đi từ đỉnh u đến các đỉnh $v \in G$;

cuu duong than cong . com

cuu duong than cong . com

Giải thuật 2: Tìm đường đi từ u đến các đỉnh của đồ thị;

//pr[v] là đỉnh trước của v trên đường đi từ u đến v

- Bước khởi tạo: Tất cả $v \in G$ chưa được thăm ($vs[v]=0$), $pr[v]=0$;
- Bước 1: Duyệt theo chiều sâu/chiều rộng (DFS/BFS) bắt đầu từ u ;
- Bước 2: Tại mỗi đỉnh $v \in G$ (trừ u) được duyệt đến cập nhật $pr[v]$;
- Bước 3: (Trả lại kết quả) Xét $v \in G$ (trừ u):
Nếu $vs[v]=0 \Rightarrow$ không có đường đi từ u đến v ;
Nếu $vs[v]=1 \Rightarrow$ xuất đường đi từ u đến v ;

cuu duong than cong . com

Cài đặt 1 (Sử dụng DFS):

// G cho bởi ma trận kề a[i][j]

```
int vs[100], n, u, pr[100];
```

```
void DfsDequy(int u) { int v;
```

```
    vs[u]= 1;
```

```
    for (v= 1; v<=n; v++)
```

```
        if (vs[v]==0 && a[u][v]==1){ pr[v] = u; DfsDequy(v); }
```

```
    }
```

cuu duong than cong . com

Cài đặt 2 (Sử dụng BFS):

```
int vs[100], n, u, q[100], pr[100];
void bfs(int u) {
    int v, dq = cq = 1;
    q[cq] = u; vs[u] = 1; pr[u] = 0;
    while (dq <= cq){
        v = q[dq]; dq++;
        for (int i= 1; i<=n; i++)
            if (vs[i]==0 && a[v][i]==1) {
                cq++; q[cq] = i; vs[i] = 1; pr[i] = v;
            }
    }
}
```

2) Tính liên thông trong đồ thị vô hướng

Định nghĩa . Một đồ thị vô hướng *liên thông* \Leftrightarrow có đường đi giữa hai đỉnh bất kỳ.

- Đồ thị G không liên thông là hợp các đồ thị con liên thông, không có đỉnh chung gọi là các *thành phần liên thông* của G .
- Đỉnh $v \in G$ là *đỉnh cắt* hay *đỉnh khớp*, *đỉnh trụ* \Leftrightarrow xóa v và các cạnh liên thuộc sẽ tạo ra một đồ thị con có nhiều thành phần liên thông hơn G .
- Cạnh $e \in G$ là *cạnh cắt* hay *cầu* \Leftrightarrow khi xóa e sẽ được đồ thị con có nhiều thành phần liên thông hơn G .

cuu duong than cong . com

Thuật toán kiểm tra tính liên thông của đồ thị vô hướng

Giải thuật 3: Kiểm tra tính liên thông của đồ thị vô hướng.

Input: Đồ thị vô hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề $a[i][j]$;

Output: Giá trị 1 nếu G liên thông, giá trị 0 nếu G không liên thông;

cuu duong than cong . com

cuu duong than cong . com

Bước 1: Sử dụng giải thuật duyệt theo chiều sâu (hoặc chiều rộng) bắt đầu từ đỉnh 1.

Bước 2: Tính số lượng k các đỉnh được duyệt.

Bước 3: Nếu $k = n$ xuất 1; nếu $k < n$ xuất 0.

cuu duong than cong . com

cuu duong than cong . com

Cài đặt 1: Sử dụng DFS

```
// G cho bởi ma trận kề a[i][j]
int vs[100], n, k;
void DfsDequy(int u){ int v;
    vs[u]= 1; k++;
    for (v = 1; v <=n; v++)
        if (vs[v]==0 && a[u][v]==1) dfsDequy(v);
}
int It() {int v;
    for (v= 1; v<= n; v++) vs[v]= 0;
    k= 0;
    DfsDequy(1);
    if (k < n) return(0) else return(1);
}
```

Sử dụng DFS

```
// G cho bởi ma trận kề a[i][j]
int vs[100], n;
void dfsDequy(int u){ int v;
    vs[u]= 1;
    for (v = 1; v <=n; v++)
        if (vs[v]==0 && a[u][v]==1) dfsDequy(v);
}
int lt() {int v;
    for (v= 1; v<= n; v++) vs[v]= 0;
    dfsDequy(1);
    for (v= 1; v<= n; v++)
        if (vs[v] == 0) return(0);
    return(1);
}
```


Cài đặt 2: Sử dụng BFS

```
// G cho bởi ma trận kề a[i][j]
int vs[100], n, k;
void bfs(int u) { int q[100], dq, cq, i;
    dq= 1; cq= 1; q[cq]= u; vs[u]= 1; k++ ;
    while (dq <= cq)
    { int v= q[dq] ; dq++ ;
    for (i= 1; i<=n; i++)
        if (vs[i]==0 && a[u][i]==1) {cq++; q[cq]= i; vs[i]= 1; k++}
    }
}
int lt(){int i;
    for (i= 1; i<= n; i++) vs[i]= 0;
    k= 0;
    bfs(1);
    if (k < n) return(0) else return(1);
}
```

3) Tính liên thông trong đồ thị có hướng

Định nghĩa.

- Đồ thị có hướng G là *liên thông mạnh* \Leftrightarrow có đường giữa hai đỉnh bất kỳ $u, v \in G$.
- Đồ thị có hướng G là *liên thông yếu* \Leftrightarrow đồ thị vô hướng nền là liên thông \Rightarrow đồ thị liên thông mạnh thì cũng liên thông yếu.
- Một đồ thị có hướng G không liên thông mạnh là hợp các đồ thị con có hướng liên thông mạnh, không có đỉnh chung gọi là các *thành phần liên thông mạnh* của G .

cuu duong than cong . com

Thuật toán kiểm tra tính liên thông của đồ thị có hướng

Giải thuật 4: Kiểm tra tính liên thông của đồ thị có hướng.

Input: Đồ thị có hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề $a[i][j]$;

Output: Giá trị 1 nếu G liên thông mạnh, giá trị 2 nếu G không liên thông mạnh nhưng liên thông yếu, giá trị trong trường hợp còn lại;

cuu duong than cong . com

cuu duong than cong . com

Bước khởi tạo: $i = 1$;

Bước 1: Sử dụng giải thuật duyệt theo chiều sâu (hoặc chiều rộng) bắt đầu từ đỉnh i ;

Bước 2: Tính số lượng k các đỉnh được duyệt;

Bước 3: Nếu $k < n$ chuyển bước 5; nếu $k = n$ thì chuyển bước 4;

Bước 4: Nếu $i = n$ thì xuất 1, nếu $i < n$ thì $i = i + 1$ và quay lại bước 1;

Bước 5: Sử dụng giải thuật duyệt theo chiều sâu (hoặc chiều rộng) bắt đầu từ đỉnh 1 khi coi các cạnh của đồ thị là vô hướng;

Bước 6: Tính số lượng k các đỉnh được duyệt;

Bước 7: Nếu $k < n$ xuất 0; nếu $k = n$ thì xuất 2;

cuu duong than cong . com

Cài đặt

// G cho bởi ma trận kề a[i][j]

```
int vs[100], n, k;
```

```
void dfs1(int v)
```

```
{ int i;
```

```
  vs[v]= 1; k++;
```

```
  for (i= 1; i<=n; i++) if (vs[i]==0 && a[v][i]==1) dfs1(i);
```

```
}
```

```
void dfs2(int v)
```

```
{ int i;
```

```
  vs[v]= 1; k++;
```

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

```
    if ((vs[i]==0 && (a[v][i]==1 || a[i][v]))) dfs2(i);
```

```
}
```

cuu duong than cong . com

cuu duong than cong . com

```
int lt()
{int i;
  for (i= 1; i<= n; i++)
    { for (int v= 1; v<= n; v++) vs[v]= 0;
      k= 0;  dfs1(i);
      if (k < n) {
        for (int v= 1; v<= n; v++) vs[v]= 0;
        k = 0; dfs2(1);
        if (k < n) return(0) else return(2);
      }
      return(1);
    }
}
```

cuu duong than cong . com

4) Giải thuật tìm các thành phần liên thông của đồ thị

Giải thuật 5: Tìm các thành phần liên thông của đồ thị vô hướng.

Input: Đồ thị vô hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề.

Output: Số k các thành phần liên thông của G ; Mỗi đỉnh $u \in G$ được gán nhãn theo số thứ tự của thành phần liên thông chứa u .

Bước khởi tạo: $k = 0$; $lt[u] = 0$ với mọi đỉnh u ;

Bước 1: Nếu mọi đỉnh u đều có $lt[u] > 0$ thì chuyển bước 3, ngược lại chọn đỉnh u có $lt[u] = 0$;

Bước 2: $k = k + 1$, duyệt theo chiều sâu (hoặc theo chiều rộng) bắt đầu từ u và gán cho các đỉnh i được duyệt tới $lt[i] = k$; quay lại bước 1;

Bước 3: Xuất k và $lt[u]$ với mọi đỉnh u ;

cuu duong than cong . com

Giải thuật duyệt theo chiều sâu:

```
void dfs(int u, int k)
```

```
{ lt[u]= k;
```

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

```
    if (lt[i]==0 && a[u,i]==1) dfs(i, k);}
```

```
int tplt()
```

```
{ for (int i= 1; i<= n; i++) lt[i]= 0;
```

```
k= 0;
```

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

```
  if (lt[i] == 0)
```

```
    {k= k + 1;
```

```
      dfs(i, k); }
```

```
return(k);
```

```
}
```

cuu duong than cong . com

cuu duong than cong . com

Giải thuật duyệt theo chiều rộng:

```
void bfs(int v, int k)
{
    int q[100], dq, cq, u;
    dq = 1; cq = 1; q[cq] = v; lt[v] = k;
    while (dq <= cq)
    {
        u = q[dq]; dq++;
        for (int i = 1; i <= n; i++)
            if (lt[i] == 0 && a[u][i] == 1)
                { cq++; q[cq] = u; lt[u] = k; }
    }
}
```

```
int tplt()
{
    for (int i = 1; i <= n; i++) lt[i] = 0;
    k = 0;
    for (i = 1; i <= n; i++)
        if (lt[i] == 0)
            { k = k + 1;
              bfs(i, k); }
    return(k);
}
```

Giải thuật 6: Tìm các thành phần liên thông mạnh của đồ thị có hướng.

Input: Đồ thị có hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề.

Output: Số k các thành phần liên thông mạnh của G ; Mỗi đỉnh $u \in G$ được gán nhãn theo số thứ tự của thành phần liên thông mạnh chứa u .

Giải thuật duyệt theo chiều sâu:

```
int a[100][100], lt[100], vs[100];
```

```
int tt[100], n, k, t;
```

```
void dfsx(int u)
```

```
{ int i;
```

```
  t++; vs[u]= 1; tt[t]= u;
```

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

```
    if (lt[i] == 0 && a[u][i] == 1 && vs[i]==0)
```

```
      dfsx(i);
```

```
}
```

```
void dfsn(int v)
```

```
{ int i;
```

```
  lt[u]= k;
```

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

```
    if (lt[i]==0 && a[u][i]==1) dfsn(i);
```

```
}
```

```
int tplt()
```

```
{ int i, j;
```

```
  for (i= 1; i<= n; i++) lt[i]= 0;
```

```
  k= 0;
```

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

```
    if (lt[i] == 0)
```

```
      { for (j=1; j<= n; j++) vs[j]= 0;
```

```
        t= 0; dfsx(i);
```

```
        while (t > 0)
```

```
          { u= tt[t]; t-- ;
```

```
            if (lt[u]==0)
```

```
              { k++;
```

```
                dfsn(u); }
```

```
            } }
```

```
  return k;
```

```
}
```

5) Giải thuật tìm các đỉnh trụ của đồ thị

Input: Đồ thị vô hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề $a[i][j]$;

Output: Các đỉnh trụ của G ;

Giải thuật 7: Tìm đỉnh trụ của đồ thị vô hướng G ;

Bước khởi tạo: Tìm số k thành phần liên thông của G ;

Bước 1: Xét mọi đỉnh $u \in G$:

1.1: Bỏ u và các cạnh liên thuộc u và tính số thành phần liên thông l ;

1.2. Nếu $l > k$ thì ghi nhận lại u là đỉnh trụ;

1.3. Trả lại u và các cạnh liên thuộc u ;

Bước 2: Xuất danh sách các đỉnh trụ;

6) Giải thuật tìm các cạnh cầu của đồ thị

Input: Đồ thị vô hướng $G = (V, E)$ gồm n đỉnh cho bởi ma trận kề $a[i][j]$;

Output: Các cạnh cầu của G ;

Giải thuật 8: Tìm cạnh cầu của đồ thị vô hướng G ;

Bước khởi tạo: Tìm số k thành phần liên thông của G ;

Bước 1: Xét mọi cạnh $e \in G$:

1.1: Bỏ cạnh e và tính số thành phần liên thông l ;

1.2. Nếu $l > k$ thì ghi nhận lại e là cạnh cầu;

1.3. Trả lại cạnh e ;

Bước 2: Xuất danh sách các cạnh cầu;