NGUYỄN HỮU ĐIỂN

THUẬT TOÁN VÀ LẬP TRÌNH

QUYỂN 5B
LÝ THUYẾT ĐỒ THỊ VÀ THUẬT TOÁN

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

NGUYỄN HỮU ĐIỂN

THUẬT TOÁN VÀ LẬP TRÌNH

QUYỂN 5

LÝ THUYẾT ĐỒ THỊ VÀ THUẬT TOÁN

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

LỜI NÓI ĐẦU

Những năm trước khi lập trình VieTeX tôi toàn dùng C/C++ thu thập tài liệu nhiều nhưng không có thời gian để viết lại. Nay muốn viết lại thì sức khỏe không ổn định. Tôi đã cố gắng gom lại thành các tập lập trình theo chủ đề. Nội dung mỗi thuật toán bắt đầu từ lý thuyết đến lập trình bằng C/C++.

Cuốn sách viết ra không dành riêng cho các bạn học tin học, mà các bạ học toán, thầy cô giáo, các bạn thích tìm hiểu về thuật toán. Cũng như tôi bắt đầu có biết gì về lập trình đâu, tự học và chăm chỉ là thành công thôi. Tôi dùng trình biên dịch Dev-C++: https://www.bloodshed.net/
Hiện nay Dev-C++ cải tiến rất nhiều và chạy tốt với môi trường unicode. Những ví dụ trong tài liệu các bạn chép thẳng vào soạn thảo và biên dịch không cần cấu hình trình biên dịch.

Tôi đã làm các quyển sách:

1. Thuật toán và số học.
2. Thuật toán và dữ liệu.
3. Thuật toán xắp xếp
4. Thuật toán tìm kiếm
5. Thuật toán đồ thị,
6. Thuật toán quay lui
7. Thuật toán chia để trị
8. Thuật toán động
9. Thuật toán tham
10. Thuật toán nén
11. Một số đề thi Olympic Tin học.

Cuốn sách dành cho học sinh phổ thông yêu toán, học sinh khá giỏi môn toán, các thầy cô giáo, sinh viên đại học ngành toán, ngành tin học và những người yêu thích Toán - Tin. Trong biên soạn không thể tránh khỏi sai sót và nhầm lẫn mong bạn đọc cho ý kiến.

Hà Nội, ngày 25 tháng 2 năm 2022

Nguyễn Hữu Điển

NHỮNG KÝ HIỆU

Trong cuốn sách này ta dùng những kí hiệu với các ý nghĩa xác định trong bảng dưới đây:

$\mathbb{N}$	tập hợp số tự nhiên
$\mathbb{N}^*$	tập hợp số tự nhiên khác 0
$\mathbb{Z}$	tập hợp số nguyên
$\mathbb{Q}$	tập hợp số hữu tỉ
$\mathbb{R}$	tập hợp số thực
$\mathbb{C}$	tập hợp số phức
$\equiv$	dấu đồng dư
∞	dương vô cùng (tương đương với $+\infty$)
$-\infty$	âm vô cùng
$\emptyset$	tập hợp rỗng
C_m^k	tổ hợp chập k của m phần tử
$\vdots$	phép chia hết
$\not\vdots$	không chia hết
$UCLN$	ước số chung lớn nhất
$BCNN$	bội số chung nhỏ nhất
deg	bậc của đa thức
IMO	International Mathematics Olympiad
APMO	Asian Pacific Mathematics Olympiad

NỘI DUNG

DANH SÁCH CÁC HÌNH

DANH SÁCH CÁC BẢNG

Danh sách chương trình

CHƯƠNG 5
LÝ THUYẾT ĐỒ THỊ VÀ THUẬT TOÁN

5.5. Tính bắc cầu và cách xây dựng. Sắp xếp cấu trúc liên kết

Ở phần đầu của chương, chúng ta nhận thấy rằng các tập hợp các đối tượng có quan hệ xác định giữa chúng có thể được biểu diễn bằng một đồ thị.

Một ví dụ đơn giản như vậy là tổ chức thứ bậc trong một tập đoàn. Mọi nhân viên trong đó đều là cấp dưới trực tiếp của người khác - những mối quan hệ như vậy có thể được biểu thị bằng một cái cây. Tuy nhiên, chúng thường phức tạp hơn. Ví dụ, ai đó có thể có nhiều hơn một "sếp" trực tiếp, hoặc có một chu trình. Trong những trường hợp như vậy cần sử dụng đồ thị có định hướng. Các bài toán có thể thực hiện là kiểm tra xem A có phải là cấp dưới của B (trực tiếp hay gián tiếp) hay không, hoặc liệt kê mọi người theo thứ bậc, tức là. đối với mỗi cặp (A, B), nếu A là cấp dưới của B, thì A phải lùi xa trong danh sách hơn B.

Ví dụ thứ hai, hãy cho một bể cá và chúng ta muốn mua cá. Người ta biết loài nào có thể cùng tồn tại và loài nào không thể. Một bài toán khả thi là tìm ra số lượng tối đa các loài cá khác nhau có thể sống cùng nhau.

Các bài toán chúng ta sẽ xem xét trong đoạn này giải quyết các ví dụ được liệt kê, cũng như nhiều bài toán thực tế khác.

5.5.1. Đóng bắc cầu. Thuật toán Worschal

Trong **??** chúng ta đã sử dụng thuật toán Worschal khi chúng ta muốn tìm ra các cặp đỉnh trong đồ thị được nối với nhau bằng một đường dẫn. Từ ma trận lân cận A, chúng ta thu được ma trận khả năng truy xuất A'. Chúng ta đã nói rằng đồ thị G' với ma trận lân cận A' được gọi là *đóng bắc cầu* của G.

Trên đây chúng ta đã xem xét bài toán sắp xếp nhân viên trong tổng công ty theo hệ thống cấp bậc. Chúng ta hãy xây dựng một đồ thị có định hướng (và có thể theo chu trình) tương ứng $G(V, E)$

như sau: cạnh $(i, j) \in E$ tồn tại đỉnh i là trưởng trực tiếp của j. Nếu chúng ta tìm thấy bao đóng bắc cầu $G'(V', E')$ của G, thì $(i, j) \in E'$ tức là ta trưởng (trực tiếp hoặc gián tiếp) của j.

Có thể tìm thấy một số ứng dụng đóng chuyển tiếp khác. Như một bài tập đơn giản, chúng ta để người đọc xem xét cách nó có thể được áp dụng trong bài toán sau: Một tập hợp các sự kiện và một quan hệ được đã cho, cho biết sự kiện nào dẫn trực tiếp đến sự hiện thực của một sự kiện khác. Đối với một sự kiện t, chúng ta muốn tìm tập hợp các sự kiện S không thể xảy ra trướct. Ví dụ, để tìm vô số tất cả các sự kiện không thể xảy ra trước khi chúng ta đến tuổi trưởng thành.

Vai trò chính trong các bài toán được xem xét được đóng bởi tính chu trình trong đồ thị. Trong các ví dụ có thể được biểu diễn bằng đồ thị xoay chiều, việc tìm kiếm các quan hệ "gián tiếp" được giải quyết hiệu quả hơn nhiều (xem 5.5.6 - Sắp xếp theo cấu trúc liên kết). Chúng ta sẽ gọi lại thuật toán của Worschal từ **??**:

```
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    if (A[i][k])
      for (j = 0; j < n; j++)
        if (A[k][j])
          A[i][j] = 1;
```

Định lý 5.7 (Worschal)**.** *Việc kiểm tra ba đỉnh (i, j, k) thông qua ba* `for` *các chu trình được lồng vào nhau theo cách được trình bày ở trên đảm bảo việc tìm ra chính xác sự đóng bắc cầu của đồ thị [Brassard, Bratley - 1996].*

Thoạt nhìn, định lý cuối cùng không có gì mới. Nó là một biến thể từ trên xuống (xem Chương 8) của lược đồ đệ quy trực quan tương ứng để tìm cách đóng bắc cầu:

$$A_k[i][j] = \begin{cases} A[i][j], & k = 0; \\ \min\{A_{k-1}[i][j], A_{k-1}[i][k] + A_{k-1}[k][j]\}, & k > 0. \end{cases}$$

Điều quan trọng là cần lưu ý thứ tự của các chu trình. Ví dụ: nếu thay vì phân đoạn ở trên, chúng ta thực thi:

```
for (i = 0; i < n; i++)
  for (k = 0; k < n; k++) if (A[i][k])
    ......
```

chúng ta sẽ nhận được một thuật toán mà trong trường hợp chung sẽ không tìm thấy điểm đóng bắc cầu chính xác của một đồ thị tùy ý. (Tại sao?)

Bài tập

▷ **5.41.** Chứng minh định lý Worschal.

▷ **5.42.** Chứng minh rằng các chu trình không thể đảo ngược. Nêu một đồ thị ví dụ trong đó chuyển vị dẫn đến bài toán.

5.5.2. Định hướng bắc cầu

Định nghĩa 5.18. Một đồ thị vô hướng $G(V, E)$ được cho. Định hướng các cạnh của nó được gọi là việc đã cho một thứ tự cho mỗi cạnh không định hướng (i, j) của E, tức là đồ thị thay đổi từ không định hướng sang có định hướng.

Định nghĩa 5.19. Cho một đồ thị có hướng $G(V, E)$, trong đó với mọi ba đỉnh $i, j, k \in V$ được thỏa mãn: Nếu $(i, k) \in E$ và $(k, j) \in E$, thì nó tuân theo cạnh $(i, j) \in E$. Khi đó G được gọi là bắc cầu.

Định nghĩa 5.20. Một đồ thị không có hướng $G(V, E)$ được gọi là có hướng chuyển tiếp nếu có thể định hướng các cạnh của nó theo cách mà đồ thị mới thu được có tính bắc cầu. Nếu không, đồ thị được gọi là không thể đọc chuyển tiếp.

Chúng ta sẽ xem xét bài toán định hướng bắc cầu của một đồ thị.

Thuật toán của Pnueli, Lempel và Ivena

Chúng ta sẽ giả định rằng đồ thị vô hướng đã cho được liên thông (nếu không, thì thuật toán được áp dụng tuần tự cho từng thành phần liên thông của nó).

Theo $i - j$, chúng ta có nghĩa là các đỉnh $i \in V$ và $j \in V$ được nối với nhau bởi một cạnh không có định hướng.

Theo $i \to j$, chúng ta có nghĩa là một cạnh được định hướng từ i đến j; với $i \leftarrow j$, chúng ta có nghĩa là một cạnh được định hướng từ j đến i, và với $i \neq j$ là không có cạnh giữa i và j.

Chúng ta sẽ sử dụng hai quy tắc sau (đối với $i, j, k \in V$):

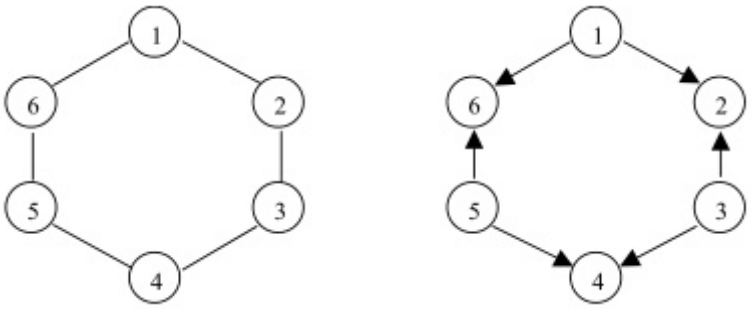
Quy tắc R1. Nếu thỏa mãn các điều kiện $i \to j, j - k, i \neq k$ thì ta định hướng cạnh $j - k : k \to j$.

Quy tắc R2. Nếu thỏa mãn các điều kiện $i \to j, i - k, j \neq k$ thì ta định hướng cạnh $i - k : i \to k$.

Thuật toán

1. Chúng ta chọn một cạnh không định hướng ngẫu nhiên và cung cấp cho nó một số định hướng. Chúng ta đánh dấu cạnh được định hướng như đã xem xét để nếu chúng ta quay lại bước này lần thứ hai, chúng ta không chọn cùng một cạnh.
2. Chúng ta áp dụng các quy tắc **R1** và **R2**, càng nhiều càng tốt, cho mỗi cạnh được định hướng:
   Chúng ta hãy xem xét cạnh $i \to j$. Khi đó với mỗi đỉnh $k \in V$ kề với j ta chuyển sang trường hợp 2.1), và với mọi đỉnh k kề i - sang trường hợp 2.2).
   2.1). Hãy nhìn vào cạnh (j, k)
   a. (theo quy tắc $R1$) Nếu $i \neq k$ và $j - k$ được thỏa mãn, thì chúng ta định hướng (j, k) là $j \leftarrow k$
   b. (mâu thuẫn với quy tắc $R1$) Nếu thỏa mãn $i \neq k$ và $j \to k$ thì đồ thị có tính bắc cầu không định hướng và kết thúc.
   2.2). Xem xét cạnh (i, k)
   a. (theo quy tắc $R2$) Nếu $j \neq k$ và $i - k$ được thỏa mãn, thì chúng ta định hướng (i, k) là $i \to k$.
   b. (mâu thuẫn với quy tắc $R2$) Nếu $j \neq k$ và $i \leftarrow k$ được thỏa mãn thì đồ thị là không thể chuyển đổi được và chúng ta kết thúc.
3. Kiểm tra xem tất cả các cạnh đều được định hướng. Nếu đúng như vậy, thì chúng ta đã có được hướng bắc cầu của đồ thị và chúng ta đã hoàn thành. Ngược lại, nếu không phải tất cả các cạnh đều được định hướng, chúng ta xóa các cạnh được định hướng khỏi đồ thị và quay lại bước 1).

Một ví dụ về định hướng bắc cầu được thảo luận trong Hình 5.22.



Hình 5.22. Định hướng bắc cầu

Trong cách triển khai bên dưới, đồ thị được biểu diễn bằng ma trận lân cận $A[][]$. Nếu cạnh không định hướng (i, j) từ đồ thị thì $A[i][j] == 1$ và $A[j][i] == 1$. Ngược lại, giá trị của các phần tử này bằng 0. Khi chúng ta định hướng cạnh $i - j$ là $i \to j$, chúng ta gán $A[i][j] = 2$ và $A[j][i] = -2$ in ma trận. Vì ở bước 3) tất cả các cạnh định hướng đều bị xóa khỏi thuật toán, chúng ta phải lưu ý lưu chúng theo một cách nào đó, để sau khi hoàn thành thuật toán, có thể in được đồ thị có hướng chuyển tiếp. Với mục đích này, chúng ta sẽ sử dụng thêm hai giá trị: đối với tất cả các phần tử $A[i][j] == 2$ (và tương ứng là $A[j][i] == -2$), chúng ta sẽ gán $A[i][j] = -3$ và $A[j][i] = -4$. Do đó, việc kiểm tra xem cạnh (i, j) có phải từ đồ thị hay không sẽ là nếu $(A[i][j] > 0)$.

Sau đây là mã nguồn của chương trình:

Chương trình 5.14. Định hướng bắc cầu (513trans-or.c)

```
#include <stdio.h>
/* Số đỉnh cực đại có thể của đồ thị */
#define MAXN 150
/* S đỉnh đồ thị cho */
const unsigned n = 6;
/*Ma trận cạnh ef của đồ thị*/
int A[MAXN][MAXN] = {
 { 0, 1, 0, 0, 0, 1 },
 { 1, 0, 1, 0, 0, 0 },
 { 0, 1, 0, 1, 0, 0 },
 { 0, 0, 1, 0, 1, 0 },
 { 0, 0, 0, 1, 0, 1 },
```

```
 { 1, 0, 0, 0, 1, 0 }
};

/* // Ví dụ cho đồ thị không định hướng
  const unsigned n = 5;
  int A[MAXN][MAXN] = {
   { 0, 1, 0, 0, 1},
   { 1, 0, 1, 0, 0},
   { 0, 1, 0, 1, 0},
   { 0, 0, 1, 0, 1},
   { 1, 0, 0, 1, 0}};
*/

char trOrient(void)
{ /* tìm số cạnh trong đồ thị*/
  unsigned i, j, k, r, tr = 0;
  char flag;
  for (i = 0; i < n - 1; i++)
    for (j = i + 1; j < n; j++)
      if (A[i][j]) tr++;
  r = 0;
  do {
    for (i = 0; i < n; i++) { /* bước 1 - định hướng c (i,j) */
      for (j = 0; j < n; j++)
        if (1 == A[i][j]) {
          A[i][j] = 2;
          A[j][i] = -2;
          break;
        }
    if (j<n) break;
  }

  /* Áp dụng quy tắc 1) và 2), đến khi nào có thể*/
  do {
    flag = 0;
    for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
        if (2 == A[i][j]) {
          for (k = 0; k < n; k++) {
            if (i != k && j != k) {
```

```
                if (0 == A[i][k] || A[i][k] < -2) { /*trường hợp 2.1)*/
                  /* a) -> đồ thị không bắc cầu định hướng*/
                  if (2 == A[j][k]) return 1;
                  /* b) -> định hướng cạnh (j,k) */
                  if (1 == A[j][k])
                    { A[k][j] = 2; A[j][k] = -2; flag = 1; }
                }
                if (0 == A[j][k] || A[j][k] < -2) { /*trường hợp 2.2)*/
                  /* a) -> đồ thị không b cầu định hướng*/
                  if (2 == A[k][i]) return 1;
                  /* b) -> cạnh định hướng (i,k) */
                  if (1 == A[i][k]) {
                    A[i][k] = 2;
                    A[k][i] = -2;
                   flag = 1;
                  }
                }
              }
            }
          }
        }
      }
    } while (flag);

  /*bước 3-loại các cạnh định hướng trong đồ thị*/
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (2 == A[i][j]) {
        A[i][j] = -3; A[j][i] = -4; r++;
      }
  } while (r < tr);
  /* lặp lại cho đến khi tất cả cạnh của đồ thị là định hướng*/
  return 0;
}

void printGraph(void)
{ unsigned i, j;
  printf("Định hướng bắc cầu là: \n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
```

```
        if (-3 == A[i][j]) printf(" 1");
          else (-4 == A[i][j]) ? printf(" -1") : printf(" 0");
     printf("\n");
    }
}

int main() {
  if (trOrient())
    printf("Đồ thị không định hướng bắc cầu! \n");
  else
    printGraph();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Định hướng bắc cầu là:
 0  1  0  0  0  1
-1  0 -1  0  0  0
 0  1  0  1  0  0
 0  0 -1  0 -1  0
 0  0  0  1  0  1
-1  0  0  0 -1  0
```

Bài tập

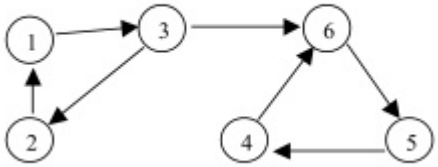
▷ **5.43.** Hãy chứng minh rằng thuật toán của Pnueli, Lempel và Ivena hoạt động chính xác.

5.5.3. Giảm bắc cầu

Định nghĩa 5.21. Một đồ thị định hướng $G(V, E)$ với ma trận khả năng truy cập W được đã cho. Một đồ thị $G'(V, E')$, $E' \subseteq E$, với số cạnh tối thiểu mà ma trận tiếp cận của nó lại là W, được gọi là giảm bắc cầu của G.

Ví dụ, một trong những thể hiện trong Hình 5.23 (bên phải) đồ thị là sự giảm bắc cầu của bất kỳ đồ thị định hướng nào có chứa đồ thị đã cho và có ma trận khả năng truy cập được hiển thị ở bên trái.

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



Hình 5.23. Giảm bắc cầu

Chúng ta sẽ xem xét bài toán tìm một giảm bắc cầu của một đồ thị có định hướng $G(V, E)$ cho trước với ma trận khả năng phản ứng W.

Thuật toán nghịch đảo của Worschal

Ta khởi tạo ma trận $A[][]$ với các giá trị như đã cho trong ma trận $W[][]$. Chúng ta loại bỏ liên tiếp các cạnh khỏi $A[][]$ cho đến khi chúng ta thu được độ giảm bắc cầu cần thiết (ở cuối thuật toán $A[][]$ sẽ là ma trận lân cận cho đồ thị cần thiết với số cạnh tối thiểu).

Với mỗi $k = 1, 2, ..., n$ ta tìm được hai đỉnh i và j sao cho:

- có một cạnh (i, k), tức là $A[i][k] == 1$.
- có một đường đi từ k đến j, tức là $W[k][j] == 1$.
- Nếu hai điểm trên gặp nhau và tồn tại cạnh $(i, j)(A[i][j] == 1)$ thì nó bị loại bỏ vì nó thừa.

Thuật toán được thực hiện với ba chu trình lồng nhau và như với thuật toán Worschal, chu trình tính bằng k phải là chu trình bên ngoài nhất:

```
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    if (A[i][k])
      for (j = 0; j < n; j++)
        if (A[i][j] && W[k][j]) A[i][j] = 0;
```

Bài tập

▷ **5.44.** Điều gì sẽ xảy ra nếu điều kiện $E' \subseteq E$ bị loại bỏ khỏi định nghĩa về giảm bắc cầu, tức là. Tìm một đồ thị có số cạnh nhỏ nhất (không nhất thiết phải là tập con của các cạnh đã cho) trên ma trận

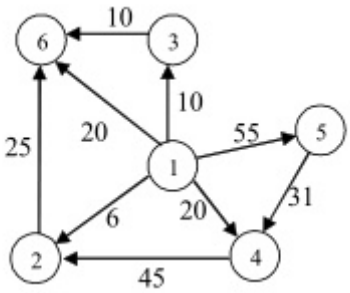
tiếp cận đã cho? Liệu thuật toán nghịch đảo của Worschal có hoạt động chính xác trong trường hợp này không?

5.5.4. Kiểm soát công ty

Bài toán: Một đồ thị có hướng trọng số $G(V, E)$ được cho với trọng số của các cạnh của số tự nhiên từ 0 đến 100. Ta nói rằng đỉnh iđiều khiển j nếu tồn tại một cạnh $(i, j), f(i, j) > 50$ hoặc có nhiều đỉnh $v_1, v_2, ..., v_k$ do i điều khiển và có các cạnh $(v_1, j), (v_2, j), ..., (v_k, j)$ và $f(v_1, j) + f(v_2, j) + \cdots + f(v_k, j) > 50$. Bài toán là tìm tất cả các đỉnh do m điều khiển trên một đỉnh m cho trước.

Bài toán mô tả một tình huống thực tế trong đó các đỉnh của đồ thị đại diện cho các công ty và mỗi cạnh (i, j) cho thấy phần trăm cổ phần mà công ty i sở hữu từ công ty j.

Dữ liệu đầu vào mẫu cho bài toán được thể hiện trong Hình 5.5.4 trong Hình, Công ty 1 điều khiển 2, 4 và 5.



Hình 5.24. Đồ thị có trọng số định hướng.

Bài toán có thể được giải quyết bằng cách điều chỉnh thuật toán Worshal để đóng bắc cầu. Chúng ta sẽ giới thiệu một mảng *control*[], trong đó chúng ta sẽ giữ tỷ lệ sở hữu của công ty i với các công ty khác. Lúc đầu, chúng ta khởi tạo mảng với trọng số của các cạnh từ i đến các đỉnh khác. Mỗi khi một công ty mới do ta kiểm soát xuất hiện, chúng ta sẽ thêm phần trăm mà công ty này kiểm soát từ những công ty khác vào mảng *control*[]. Một công ty mới do i kiểm soát xuất hiện khi tỷ lệ phần trăm ở vị trí *control*[] trở nên trên 50. Độ phức tạp của thuật toán được mô tả là $\Theta(n^3)$.

Chương trình 5.15. Kiểm soát công ty (514company.c)

```
#include <stdio.h>
/* Số lượng công ty lớn nhất có thể (các đỉnh trong đồ thị) */
#define MAXN 150
/* Số ương công ty (số đỉnh trong đồ thị) */
const unsigned n = 6;
const unsigned m = 1; /* Tìm những công ty nào kiểm soát 1 công ty *
    /
/* Ma trận kề của đồ thị*/
const unsigned A[MAXN][MAXN] = {
  { 0, 6, 10, 20, 55, 20 },
  { 0, 0, 0, 0, 0, 25 },
  { 0, 0, 0, 0, 0, 10 },
  { 0, 45, 0, 0, 0, 0 },
  { 0, 0, 0, 31, 0, 0 },
  { 0, 0, 0, 0, 0, 0 }
};

unsigned control[MAXN];
char used[MAXN];

void addControls(void)
{ unsigned i, j;
  for (i = 0; i < n; i++)
  /*công ty i kiểm soát, ta thêm tác động của nó vào những người c
    ủa m */
    if (control[i] > 50 && !used[i]) {
     for (j = 0; j < n; j++)
        control[j] += A[i][j];
      used[i] = 1;
    }
  }

void solve(void)
{ unsigned i;
  for (i = 0; i < n; i++) {
    control[i] = 0;
    used[i] = 0;
  }
  for (i = 0; i < n; i++) control[i] = A[m-1][i];
```

```
  for (i = 0; i < n; i++) addControls();
}

void printResult(void) {
  unsigned i;
  printf("Công ty %u kiểm soát những công ty sau đây: \n", m);
  for (i = 0; i < n; i++)
    if (control[i] > 50)
      printf("%u: %3u% \n", i, control[i]);
  printf("\n");
}

int main() {
  solve();
  printResult();
}
```

Kết quả thực hiện chương trình:

```
Công ty 1 kiểm soát các công ty sau:
1: 53%
3: 51%
4: 55%
```

5.5.5. Bài tập

▷ **5.45.** Có thể giải bài toán với độ phức tạp nhỏ hơn $\Theta(n^3)$ không?

5.5.6. Sắp xếp theo cấu trúc liên kết

Định nghĩa 5.22. *Sắp xếp tôpô* của một đồ thị xoay chiều có định hướng $G(V, E)$ được gọi là danh sách có thứ tự tuyến tính Z gồm các đỉnh của nó, mà nó được thỏa mãn: Với mỗi hai đỉnh$i, j \in V$, nếu có một đường đi từ i đến j, thì đỉnh i phải đứng trước j trong Z.

Trong thực tế, danh sách Z hiếm khi có thể được xây dựng theo một cách duy nhất.

Định nghĩa 5.23. Khi có nhiều hơn một bậc của đỉnh trong Z, tập hợp tất cả các danh sách Z có thể có được gọi là sắp xếp tôpô đầy đủ.

Mệnh đề 5.1. *Một đồ thị mạch hở có định hướng $G(V,E)$ được cho. Chỉ có một danh sách Z, là cách sắp xếp theo cấu trúc liên kết của G, tức là G được liên thông yếu.*

Ví dụ 1: Có một hệ thống cấp bậc nghiêm ngặt trong một tu viện ở Tây Tạng: mỗi tu sĩ có một hoặc nhiều cấp trên trực tiếp (người chịu trách nhiệm về những vi phạm của mình), và những người không có ông chủ được coi là cấp trên của họ. Các nghi lễ hiến tế được thực hiện hàng năm, và một số nhà sư phải được chọn để ném xuống "vực thẳm của vô cùng." Các nhà sư luôn được chọn để hiến tế trong số những người đứng thấp nhất trong hệ thống cấp bậc của tu viện. Tuy nhiên, thống đốc của tu viện đã không tốt với lý thuyết về số lượng và đã tìm thấy một lập trình viên để xác định những nhà sư q đó nên được hy sinh. Rõ ràng, bài toán là tìm cách sắp xếp tôpô của đồ thị và hy sinh q tu sĩ cuối cùng từ danh sách kết quả **Z**.

Ví dụ 2: Một ví dụ khác là sách giáo khoa toán. Các định lý, định nghĩa, tiên đề khác được sử dụng trong phần chứng minh của mỗi định lý. Sách giáo khoa nên được viết theo một cách nhất quán - "được sắp xếp theo cấu trúc liên kết" và mỗi định lý sẽ chỉ trích dẫn những điều đã được định nghĩa/chứng minh trước đó.

Vì không cho phép tính chu trình trong các đồ thị được coi là có định hướng, nên luôn có ít nhất một đỉnh trong chúng không có giá trị tiền nhiệm. (Tại sao?) Một đỉnh như vậy (lưu ý rằng có thể có nhiều hơn một) sẽ chiếm một vị trí hàng đầu trong sắc lệnh. Theo kết quả của kết luận vừa được đã cho, chúng ta nhận được

Thuật toán 1 để sắp xếp topo

1) Khởi tạo Z dưới dạng danh sách trống.

2) Chọn đỉnh i không có đỉnh trước và thêm nó vào cuối danh sách Z. Loại trừ i khỏi đồ thị, cũng như tất cả các đường liên quan đến nó.

3) Lặp lại bước 2) cho đến khi không còn đỉnh.

Ghi chú:

1. Nếu ở bước 2) có nhiều hơn một đỉnh không có tiền nhiệm thì ta chọn tùy ý. (Tại sao?)

2. Nếu ở bước 2) không có đỉnh nào không có đỉnh trước và có

nhiều đỉnh hơn trong đồ thị, thì đồ thị đó là tuần hoàn, điều này mâu thuẫn với điều kiện và do đó không có sắp xếp tôpô. Điều ngược lại cũng đúng: nếu đồ thị là tuần hoàn, thì tại một số bước của thuật toán, chắc chắn chúng ta sẽ thấy mình trong tình huống được mô tả. (Tại sao?)

Việc thực hiện thuật toán này là hiệu quả nhất nếu chúng ta biểu diễn đồ thị thông qua danh sách những người kế nhiệm (hoặc danh sách những người đi trước). Chúng ta sẽ giữ nguyên số i của các bậc trước cho mỗi đỉnh $i \in V$. Tất cả num i có thể được tìm thấy với tổng độ phức tạp $\Theta(n+m)$ bằng cách duyệt qua đồ thị. Do đó, sau khi tìm được những người thừa kế $i_1, i_2, ..., i_k$ trên mỗi đỉnh i, chúng ta tăng thêm một $num_{i_1}, num_{i_2}, ..., num_{i_k}$. Tiếp theo, ở mỗi bước, chúng ta chọn đỉnh j, với $num_j = 0$. Thêm nó vào cuối danh sách Z, và trừ một đơn vị num_j cho mỗi đỉnh kế tiếp j của đỉnh đã chọn. Tổng số bước sẽ là n (để biểu diễn bằng ma trận lân cận). Do đó độ phức tạp của thuật toán trở thành) $\Theta(n^2)$.

Độ phức tạp có thể được giảm xuống $\Theta(m+n)$ như sau:

1) Ban đầu ta đưa vào Z tất cả các đỉnh có số bậc trước bằng 0. Chúng ta sẽ giới thiệu một con trỏ trỏ đến phần tử chưa được xét cuối cùng của Z (ở đầu con trỏ trỏ đến phần tử đầu tiên của Z).

2) Xét đỉnh i mà con trỏ trỏ tới. Đối với mỗi kế thừa của nó j, chúng ta giảm num_j đi một. Nếu bất kỳ num_j nào trở thành 0, thêm j vào cuối Z. Di chuyển con trỏ đến phần tử tiếp theo của Z và lặp lại bước 2).

Nếu đồ thị không chứa vòng lặp, sau tối đa n lần lặp lại của 2) tất cả các đỉnh của đồ thị sẽ nằm trong Z và nó sẽ được sắp xếp theo cấu trúc liên kết.

Thuật toán 2 để sắp xếp tôpô không đầy đủ

Thuật toán thứ hai chúng ta sẽ xem xét có cùng độ phức tạp $\Theta(m+n)$ như thuật toán 1. Ở một mức độ nào đó, nó "đối lập" với thuật toán trước: ở mỗi bước, chúng ta sẽ tìm kiếm một đỉnh không có người thừa kế và do đó chúng ta sẽ nhận được sự sắp xếp tôpô của đồ thị ngược lại. Việc triển khai rất dễ dàng và thanh lịch, sử dụng mặt trái của phép đệ quy trong chức năng thu thập thông tin theo chiều sâu:

```
void DFS(unsigned i)
{ unsigned k;
  used[i] = 1;
  for (k = 0; k < n; k++)
    if (A[i][k] && !used[k])
      DFS(k);
  printf("%u ", i + 1);
}
```

Có thể thấy, sửa đổi duy nhất của *DFS* tiêu chuẩn của **??** là nơi in trên đầu *i*. Điều này được thực hiện sau khi tất cả các lệnh gọi đệ quy đã được thực hiện, điều này đảm bảo rằng tất cả các đỉnh có thể đạt được từ đỉnh hiện tại đã được xem và in.

Trong hàm chính, chúng ta sẽ chạy $DFS(i)$ cho mỗi đỉnh i, mà chúng ta chưa xét đến và không có đỉnh nào trước đó. Điều kiện cuối cùng là cần thiết cho hoạt động chính xác của thuật toán và chúng ta sẽ kiểm tra nó bằng cách sử dụng array Used[] , được giới thiệu để đánh dấu các đỉnh đã ghé thăm. Sau đây là mã nguồn của chương trình:

Chương trình 5.16. Sắp xếp tô pô (515topsort.c)

```
#include <stdio.h>
/*Số lượng lớn có thể trong đồ thị */
#define MAXN 200
/* Số đỉnh trong đồ thị*/
const unsigned n = 5;
/* Ma trận cạnh kề của đồ thị*/
const char A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 0 },
  { 0, 0, 1, 0, 1 },
  { 0, 0, 0, 1, 0 },
  { 0, 0, 0, 0, 0 },
  { 0, 0, 1, 0, 0 }
};

char used[MAXN];

/* sửa đổi DFS */
void DFS(unsigned i)
```

```
{ unsigned k;
  used[i] = 1;
  for (k = 0; k < n; k++)
    if (A[i][k] && !used[k])
      DFS(k);
  printf("%u ", i + 1);
}

int main() {
  unsigned i;
  /* khởi tạo */
  for (i = 0; i < n; i++) used[i] = 0;
  printf("Sắp xếp tô pô (theo chiều ngược lại): \n");
  for (i = 0; i < n; i++)
    if (!used[i])
      DFS(i);
  printf("\n");
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Sắp xếp theo topo (theo thứ tự ngược lại):
4 3 5 2 1
```

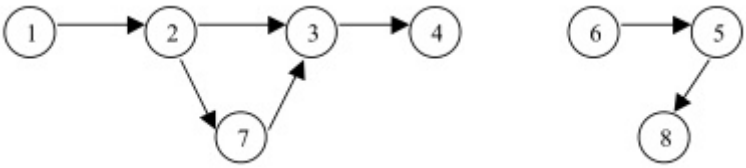
Bài tập

▷ **5.46.** Hãy chứng minh Mệnh đề 5.25.

▷ **5.47.** Để chứng minh rằng nếu trong bước 2 của thuật toán 1 có nhiều hơn một đỉnh không có đỉnh trước, chúng ta có thể tự do chọn bất kỳ đỉnh nào trong số chúng.

5.5.7. Sắp xếp tô pô đầy đủ

Xem xét đồ thị trong Hình 5.25. Một loại tôpô cho G có thể là $Z = (1,2,7,3,4,6,5,8)$ hoặc $Z = (6,1,2,5,7,3,8,4)$. Sử dụng chương trình để tìm cách sắp xếp tôpô đầy đủ (mà chúng ta sẽ trình bày sau), có thể tính được rằng số lượng của tất cả các danh sách Z khác nhau là 56. (Tại sao?)



Hình 5.25. Sắp xếp tô pô đầy đủ

Trong giây lát, chúng ta sẽ quay lại thuật toán 1 của 5.5.6. Khi chúng ta tìm kiếm sắp xếp theo cấu trúc liên kết, ở mỗi cấp độ, chúng ta chọn một đỉnh mà không có bậc trước, và khi có một số - chúng ta chọn bất kỳ. Các quyết định khác nhau xuất phát từ sự lựa chọn không xác định. Để tìm tập hợp tất cả các danh sách Z có thể có, chúng ta sẽ áp dụng tính năng cạn kiệt hoàn toàn. Vì vậy, khi có nhiều hơn một lựa chọn để chọn một đỉnh, chúng ta sẽ thử tất cả chúng lần lượt (rõ ràng là mỗi lựa chọn đều dẫn đến một quyết định). Trong phần triển khai bên dưới, điều này được thực hiện bởi hàm $fullTopSort()$. Khi xem một đỉnh, chúng ta sẽ xóa nó khỏi đồ thị và bắt đầu đệ quy $fullTopSort()$ với đồ thị đã sửa đổi. Sau khi quay trở lại từ đệ quy, chúng ta khôi phục đồ thị và lặp lại tương tự với đỉnh có thể tiếp theo. Phần dưới cùng của đệ quy là khi danh sách Z chứa n đỉnh, tại đây chúng ta in ra nghiệm tìm được. Lược đồ của hàm đệ quy chính như sau:

```
fullTopSort(count) {
  if (count == n) {
    <Tìm thấy sắp xếp tô pô => In nó ra>;
    return;
  }
  for (<mỗi đỉnh vi không có tiền nhiệm>)
    <xóa bỏ vi của đồ thị>;
    fullTopSort(count + 1); /*đệ quy và trả về trước */
    <khôi phục vi in ra>;
  }
}
```

Chương trình ầy đủ

Chương trình 5.17. Sắp xếp tô pô đầy đủ (516topsortf.c)

```
#include <stdio.h>
#define MAXN 200 /* số đỉnh lớn nhất*/
/* Số đỉnh của đồ thị */
const unsigned n = 8;
/* Ma trận cạnh kề */
char A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 1, 0, 0, 0, 1, 0 },
  { 0, 0, 0, 1, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 1 },
  { 0, 0, 0, 0, 1, 0, 0, 0 },
  { 0, 0, 1, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0 }
};

char used[MAXN];
unsigned topsort[MAXN], total = 0;

void printSort(void)
{ unsigned i;
  printf("Số sắp xếp tô pô %u: ", ++total);
  for (i = 0; i < n; i++) printf("%u ", topsort[i] + 1);
  printf("\n");
}

void fullTopSort(unsigned count)
{ unsigned i, j, k, saved[MAXN];
  if (count == n) { printSort(); return; }
  /* Tìm tất cả các đỉnh không có đỉnh trước nó*/
  for (i = 0; i < n; i++) {
    if (!used[i]) {
      for (j = 0; j < n; j++)
        if (A[j][i]) break;
        if (j == n) {
          for (k = 0; k < n; k++) {
            saved[k] = A[i][k]; A[i][k] = 0;
          }
        used[i] = 1;
        topsort[count] = i;
```

```
            fullTopSort(count + 1); /* hồi quy */
            used[i] = 0;
            for (k = 0; k < n; k++) A[i][k] = saved[k];
          }
        }
      }
}

int main() {
  unsigned i;
  for (i = 0; i < n; i++) used[i] = 0;
  fullTopSort(0);
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Số sắp xếp tô pô 1: 1 2 6 5 7 3 4 8
Số sắp xếp tô pô 2: 1 2 6 5 7 3 8 4
Số sắp xếp tô pô 3: 1 2 6 5 7 8 3 4
Số sắp xếp tô pô 4: 1 2 6 5 8 7 3 4
Số sắp xếp tô pô 5: 1 2 6 7 3 4 5 8
...
Số sắp xếp tô pô 49: 6 1 5 2 8 7 3 4
Số sắp xếp tô pô 50: 6 1 5 8 2 7 3 4
Số sắp xếp tô pô 51: 6 5 1 2 7 3 4 8
Số sắp xếp tô pô 52: 6 5 1 2 7 3 8 4
Số sắp xếp tô pô 53: 6 5 1 2 7 8 3 4
Số sắp xếp tô pô 54: 6 5 1 2 8 7 3 4
Số sắp xếp tô pô 55: 6 5 1 8 2 7 3 4
Số sắp xếp tô pô 56: 6 5 8 1 2 7 3 4
```

Bài tập

▷ **5.48.** Tính toán phân tích số lượng sắp xếp tôpô khác nhau của đồ thị trong Hình 5.5.7.

5.5.8. Bổ sung một đồ thị xoay chiều thành một đồ thị được liên kết yếu

Bài toán: Cho đồ thị mạch hở có hướng $G(V, E)$. Ta đang tìm đồ thị $G'(V, E')$ sao cho:

- $E \subseteq E'$
- Chỉ có một danh sách Z, là một cách sắp xếp theo topo của G'.

Nói cách khác, chúng ta đang tìm một tập hợp các cạnh để thêm vào Gsao cho có một danh sách$Z$ của G' được sắp xếp theo cấu trúc liên kết duy nhất.

Bài toán cũng có thể được đặt như sau: Bổ sung tập hợp các cạnh của một đồ thị xoay chiều có định hướng sao cho nó trở nên liên kết yếu trong khi vẫn còn xoay chiều.

Kết luận sau đây có thể được rút ra từ việc định nghĩa lại bài toán: sau khi thực hiện đóng bắc cầu trên đồ thị, ma trận lân cận A sẽ có dạng như sau:

(*) Mọi $i \neq j$ có một trong hai phần tử $A[i][j]$ và $A[j][i]$ bằng một, và còn lại - bằng không.

Thuật toán 1

Thuật toán đầu tiên chúng ta sẽ đề xuất để giải quyết bài toán như sau:

1) Chúng ta thực hiện đóng bắc cầu và, nếu có $i, j \in V, i \neq j$, mà $A[i][j] == 0$ và $A[j][i] == 0$, thì chúng ta gán $A[i][j] = 1$

2) Chúng ta thực hiện 1), trong khi (*) đúng với mọi i, j nghĩa là số các đơn vị 1 là $\frac{n^2 - n}{2}$.

Thuật toán trực quan này có độ phức tạp $\Theta(n^4)$ và không thể áp dụng trong thực tế.

Thuật toán 2

Chúng ta hãy xem xét một cách sắp xếp tôpô của đồ thị $Z = (v_{i1}, v_{i2}, ..., v_{in})$. Việc hoàn thiện G' được thực hiện theo sơ đồ sau:

```
for (j = 0; j < n-1; j++)
  for (k = j+1; k < n; k++)
    if (<không tồn tại (v ij ,v ik )>) /*thêm cạnh (v ij ,v ik ); */
```

Thuật toán này rõ ràng là hiệu quả hơn *Thuật toán 1* và có độ phức tạp $\Theta(n^2)$.

Bài tập

▷ **5.49.** Tìm độ phức tạp của *Thuật toán 1*.

▷ **5.50.** 2. Cho đồ thị mạch hở $G(V, E)$. Ta tìm đồ thị $G'(V, E')$ sao cho:

- $E \subseteq E'$
- Chỉ có một danh sách Z', đó là một cách sắp xếp theo tôpô của G'.
- E' chứa một số lượng cạnh tối thiểu.

Độ phức tạp tốt nhất có thể đạt được là gì?

5.5.9. Xây dựng đồ thị của các đỉnh đã cho

Bài toán: Đã cho bậc của các đỉnh của một đồ thị vô hướng. Xây dựng đồ thị (tức là xác định tập các cạnh của nó). Có thể là bài toán không có lời giải, tức là không có đồ thị với các bậc đã cho.

Lời giải: Chúng ta sẽ giải quyết bài toán trên cơ sở quan sát sau: chúng ta hãy xem xét một đỉnh tùy ý của đồ thị, ký hiệu nó bằng i, và bậc của nó bằng $d(i)$. Rõ ràng, để có lời giải thì phải có ít nhất $d(i)$ đỉnh có bậc lớn hơn hoặc bằng một đỉnh kề với i. Chúng ta có thể nối chúng và giảm độ của chúng đi một, và đặt độ của đỉnh i là 0. Chúng ta tiếp tục càng lâu càng tốt.

chỉ còn làm rõ một cách chúng ta sẽ chọn các đỉnh (điều này không thể được thực hiện một cách ngẫu nhiên - tại sao?). Như vậy, ở mỗi bước ta sẽ chọn đỉnh có bậc cao nhất và nối lại với các đỉnh có bậc cao nhất.

Chúng ta cung cấp việc triển khai thuật toán được mô tả cho người đọc như một bài tập dễ dàng. Việc triển khai đơn giản nhất sẽ có độ phức tạp $\Theta(n^2.\log_2 n)$: n bước, trong mỗi bước chúng ta sắp xếp các mức độ. Tất nhiên, việc sắp xếp là không cần thiết, bởi vì việc sắp xếp lại, sau khi giảm các bậc đi 1, có thể được thực hiện với độ phức tạp tuyến tính (như thế nào?), Và do đó chúng ta nhận được tổng độ phức tạp $\Theta(n^2)$.

Bài tập

▷ **5.51.** Liệu thuật toán được mô tả có hoạt động chính xác không nếu chúng ta xem xét một đồ thị là đa đồ thị?

▷ **5.52.** Thuật toán được mô tả sẽ hoạt động chính xác nếu có các vòng lặp trong đồ thị?

▷ **5.53.** Hãy chứng tỏ rằng thứ tự các đỉnh được coi là quan trọng.

▷ **5.54.** Đề xuất một cách để sắp xếp lại các đỉnh bằng độ phức tạp tuyến tính.

▷ **5.55.** Có thể giải bài toán với độ phức tạp $\Theta(m+n)$ không? Và với bài toán nhỏ hơn?

5.6. Khả năng tiếp cận và liên thông

Các bài toán về khả năng tiếp cận và liên thông được sử dụng trong máy tính, truyền thông, mạng đường bộ và các mạng khác.

Ví dụ: Một mạng máy tính được đã cho trong đó có một liên thông trực tiếp giữa một số máy tính. Nếu chúng ta trình bày mạng dưới dạng một đồ thị có định hướng, các tác vụ sau có thể được quan tâm:

- *Liên thông mạng:* Để xác định xem có một liên thông giữa hai máy tính, dù là trực tiếp hay gián tiếp, tức là. đi qua các máy tính khác. Để giải quyết bài toán, cần phải kiểm tra xem đồ thị có được liên thông hay không.
- *Tìm một cụm:* Xác định số lượng máy tính được liên thông lớn nhất.
- *Ổn định mạng:* Nếu tất cả các máy tính trong hệ thống được liên thông, thì bài toán tiếp theo có thể là tìm các máy tính chia sẻ hoặc liên thông chia sẻ, tức là sao cho nếu bị loại bỏ, các cặp máy tính không còn được liên thông nữa sẽ xuất hiện.

Những bài toán này và các bài toán khác sẽ được thảo luận trong phần này và được giải quyết với sự trợ giúp của các thuật toán từ lý thuyết đồ thị.

5.6.1. Các thành phần liên thông

Bài toán: Một đồ thị không định hướng được đã cho. Kiểm tra xem nó đã được liên thông chưa, tức là. cho dù có một đường đi giữa mỗi trong số hai đỉnh của nó. Nếu nó không được liên thông, hãy tìm tất cả các thành phần được liên thông của nó, tức là. tất cả các đồ thị con được liên thông tối đa của nó.

Thuật toán

Bài toán có thể được giải quyết bằng cách duyệt đồ thị, ví dụ: theo chiều sâu:

1) Chúng ta bắt đầu từ một đỉnh ngẫu nhiên và đánh dấu tất cả các đỉnh mà chúng ta coi khi thu thập thông tin là thuộc một thành phần được liên thông. Nếu nó chứa tất cả các đỉnh của đồ thị, thì nó được liên thông.

2) Nếu các đỉnh yêu cầu còn lại, điều đó có nghĩa là đồ thị không được liên thông. Chúng ta bắt đầu thu thập thông tin mới từ một đỉnh không được kiểm soát và xây dựng thành phần được liên thông thứ hai. Lặp lại bước 2) cho đến khi còn lại các đỉnh cần thiết.

Độ phức tạp của thuật toán là $\Theta(n+m)$.

Việc triển khai bao gồm một sửa đổi đơn giản của chức năng thu thập thông tin theo độ sâu. Sau đây là mã nguồn của chương trình.

Chương trình 5.18. Số thành phần liên thông (517strcon1.c)

```
#include <stdio.h>
/* Số đỉnh lớn nhất trong đồ thị  */
#define MAXN 200
/* Số đỉnh  trong đồ thị  */
const unsigned n = 6;
/* Ma trận kề trong đồ thị*/
const char A[MAXN][MAXN] = {
  { 0, 1, 1, 0, 0, 0 },
  { 1, 0, 1, 0, 0, 0 },
  { 1, 1, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 1, 1 },
  { 0, 0, 0, 1, 0, 1 },
  { 0, 0, 0, 1, 1, 0 }
};
```

```
char used[MAXN];

/*Sửa đổi DFS */
void DFS(unsigned i)
{ unsigned k;
  used[i] = 1;
  printf("%u ", i + 1);
  for (k = 0; k < n; k++)
    if (A[i][k] && !used[k]) DFS(k);
}

int main() {
  unsigned i, comp;
  /* Khởi tạo*/
  for (i = 0; i < n; i++) used[i] = 0;
  printf("\nĐây là tất cả thành phần liên thông: \n");
  comp = 0;
  for (i = 0; i < n; i++)
    if (!used[i]) {
      comp++;
      printf("{ ");
      DFS(i);
      printf("}\n");
    }
  if (1 == comp)
    printf("Đồ thị liên thông.\n");
  else
    printf("Số thành phần liên thông trong đồ thị: %d \n", comp);
  return 0;
}
```

Kết quả thực hiện chương trình:

```
{1 2 3}
{4 5 6}
Số thành phần được liên thông trong đồ thị: 2
```

Bài tập

▷ **5.56.** Hãy triển khai một thuật toán để tìm các thành phần được liên thông của một đồ thị có trọng số chiều rộng.

5.6.2. Các thành phần liên kết mạnh trong đồ thị định hướng

Chúng ta sẽ nhớ lại định nghĩa 5.10: đồ thị định hướng $G(V, E)$ được gọi là liên thông mạnh nếu có một đường đi từ i đến j và từ j đến i, với mỗi $i \neq j, i, j \in V$. Nếu đồ thị không được liên thông mạnh, chúng ta quan tâm đến tất cả các thành phần của liên thông mạnh (tất cả các đồ thị con được liên thông mạnh lớn nhất của G).

Các thuật toán chúng ta sẽ xem xét một lần nữa là sự thích ứng của thu thập thông tin sâu, nhưng ở đây mọi thứ không tầm thường như trong một đồ thị không định hướng.

Thuật toán 1

1) Ta chọn một đỉnh $i \in V$ tùy ý.

2) Chúng ta thực hiện $DFS(i)$ và tìm tập các đỉnh R có thể truy cập được từ i.

3) Chúng ta tạo thành một đồ thị "đảo ngược" $G'(V, E')$: hướng của tất cả các cạnh mà chúng "đảo ngược", tức là cạnh $(j, k) \in E'$ khi và chỉ khi $(k, j) \in E$.

4) Chúng ta thực hiện thu thập thông tin theo chiều sâu từ đỉnh i trong G'- vì vậy chúng ta tìm thấy tập hợp các đỉnh Q, có thể đạt được từ i trong G' (và tương ứng đạt đến i trong G).

5) Giao điểm của R với Q cho một thành phần liên kết mạnh.

6) Loại trừ thành phần này khỏi đồ thị và nếu có nhiều đỉnh hơn, hãy lặp lại bước 1).

Thuật toán được mô tả có độ phức tạp $\Theta(n.(N + m))$. Thuật toán thứ hai chúng ta sẽ trình bày có độ phức tạp $\Theta(m + n)$.

Thuật toán 2

Sửa đổi đầu tiên của chức năng thu thập thông tin theo độ sâu mà chúng ta sẽ thực hiện là đánh số các đỉnh: mỗi đỉnh sẽ nhận được một số phải lớn hơn số kế nhiệm của nó trong quá trình thu thập thông tin. Chiến lược đánh số này (được gọi là postnum trong tiếng

Anh) được thực hiện bằng cách thêm dòng sau vào hàm `DFS(i)` sau lệnh gọi đệ quy:

```
postnum [i] = count ++; ,
```

trong đó count là bộ đếm đánh số (chúng ta đã đi qua bao nhiêu đỉnh cho đến nay).

Sau đây là phần thực của thuật toán:

1) Thực hiện thu thập thông tin theo độ sâu trong G, đánh số các đỉnh theo chiến lược postnum được mô tả ở trên.

2) Chúng ta tạo thành một đồ thị $G(V, E')$, tương tự như G: sự khác biệt là hướng của tất cả các cạnh trong nó là "đảo ngược", tức là cạnh $(j, i) \in E'$ khi và chỉ khi $(i, j) \in E$.

3) Thực hiện thu thập thông tin theo độ sâu trong cột G'. Chúng ta bắt đầu tìm kiếm từ đỉnh w này, mà $postnum[w]$ có giá trị lớn nhất. Tất cả các đỉnh đạt được trong quá trình thu thập thông tin đều thuộc cùng một thành phần được liên thông mạnh. Nếu tìm kiếm không đi qua tất cả các đỉnh, thì đồ thị không được liên thông mạnh và chúng ta chọn đỉnh ban đầu thứ hai là đỉnh trong số các đỉnh không được kiểm tra mà giá trị của postnum là cao nhất. Bước này được lặp lại cho đến khi chúng ta đi xung quanh tất cả các đỉnh.

Mã nguồn của C sau đây. Điểm thú vị nhất trong đó là cách tiếp cận mà chúng ta sử dụng để thực hiện bước 2) của thuật toán. Vì sẽ cực kỳ kém hiệu quả nếu sử dụng một ma trận lân cận khác cho G', chúng ta đã sử dụng hàm thu thập thông tin theo độ sâu thứ hai (`backDFS`), hàm này xem xét các cạnh của đồ thị G ở dạng "đảo ngược", như thể đang làm việc với G'. Lưu ý rằng cách tiếp cận này sẽ không thể thực hiện được với mọi bản trình bày của cột: ví dụ: với danh sách những người thừa kế.

Chương trình 5.19. Các thành phần liên thông mạnh (518strconn.c)

```
#include <stdio.h>
const N = 10; /* số đỉnh và ma trận kề của đồ thị*/
int A[N][N] = {
  { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 1, 1, 0, 0, 0, 0, 0, 0 },
  { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 1, 0, 1, 0, 0, 0 },
```

```
    { 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },
    { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
    { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 }
};

int used[N];
int postnum[N], count = 0;

/* Duy theo chiều sâu trong khi vẫn giữ nguyên số*/
void DFS(int i)
{ int j;
  used[i] = 1;
  for (j = 0; j < N; j++)
    if (!used[j] && A[i][j]) DFS(j);
  postnum[i] = count++;
}

/* Duyệt theo chiều sâu đồ thị G' */
void backDFS(int i)
{ int j;
  printf("%d ", i + 1);
  count++;
  used[i] = 1;
  for (j = 0; j < N; j++)
    if (!used[j] && A[j][i]) {
      backDFS(j);
    }
}

/*Tìm thành phần liên thông mạnh trong đồ thị*/
void strongComponents(void)
{ int i;
  for (i = 0; i < N; i++) used[i] = 0;
  while (count < N - 1) {
    for (i = 0; i < N; i++)
      if (!used[i]) DFS(i);
  }
```

```
  for (i = 0; i < N; i++) used[i] = 0;
  count = 0;
  while (count < N - 1) {
    int max = -1, maxv = -1;
    for (i = 0; i < N; i++)
      if (!used[i] && postnum[i] > max) {
        max = postnum[i];
        maxv = i;
      }
    printf("{ ");
    backDFS(maxv);
    printf("}\n");
  }
}

int main() {
  printf("Những thành ph liên thông mạnh là:\n");
  strongComponents();
}
```

Kết quả thực hiện chương trình:

```
Các thành phần có liên quan chặt chẽ trong đồ thị là:
{1 3 2 6 5 4}
{7 10 9 8}
```

Bài tập

▷ **5.57.** Chứng minh rằng độ phức tạp của thuật toán 1 là $\Theta(n.(N + m))$.
(Gợi ý: Để chứng minh rằng trường hợp xấu nhất đối với thuật toán là một đồ thị có hướng xoay chiều với đúng $m = n.(N-1)/2$ cạnh. Trong trường hợp này, các thành phần của liên thông mạnh là n và duyệt theo chiều sâu cho mỗi thành phần có độ phức tạp $\Theta(m+n)$.)

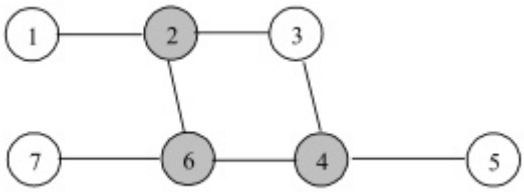
▷ **5.58.** Thuật toán kiểm tra xem một đồ thị có định hướng có được *liên thông yếu* hay không và để tìm tất cả các *thành phần được liên thông yếu* của nó là gì? Lưu ý rằng, không giống như liên thông mạnh, phá vỡ đồ thị của các thành phần không giao nhau, hai thành phần được liên thông yếu khác nhau có thể bao gồm các đỉnh giống nhau. Độ phức tạp của thuật toán của bạn là gì?

5.6.3. Điểm phân chia trong một đồ thị không định hướng

Định nghĩa 5.24. Cho một đồ thị vô hướng được liên thông. *Một điểm phân chia* trong đồ thị được gọi là đỉnh, sau khi loại bỏ điểm đó (cũng như tất cả các cạnh liên quan đến nó) thì đồ thị không còn được liên thông.

Định nghĩa 5.25. Một đồ thị liên thông không định hướng được gọi là hai liên thông nếu không có điểm phân chia. Điều này cũng có nghĩa là nó vẫn được liên thông sau khi loại bỏ bất kỳ đỉnh nào.

Đối với đồ thị liên thông trong Hình 5.26 tách các đỉnh 2, 4 và 6. Ví dụ, nếu chúng ta loại bỏ đỉnh 6, chúng ta nhận được hai thành phần của liên thông: $\{1, 2, 3, 4, 5\}$ và $\{7\}$.



Hình 5.26. Điểm phân tách trong đồ thị

Nếu chúng ta sử dụng trực tiếp định nghĩa, chúng ta có thể đạt được độ phức tạp $\Theta(n.(M+n))$: chúng ta sẽ sử dụng thuật toán để kiểm tra xem một đồ thị không định hướng có được liên thông hay không. Mỗi đỉnh, sau khi loại bỏ mà đồ thị không còn được liên thông, là một điểm phân chia.

Thuật toán trên rất dễ thực hiện, nhưng không phải là hiệu quả nhất.

Mệnh đề 5.2. *Đỉnh $k \in V$ là điểm phân chia khi và chỉ khi có hai đỉnh khác i và j $(i, j \in V)$ mà mỗi đỉnh giữa chúng đi qua k.*

Mệnh đề 5.3. *Gọi $G(V, E)$ là một đồ thị không có hướng được liên thông và $T(V, D)$ là một cây bao trùm của G, được xây dựng bằng cách duyệt theo chiều sâu. Đỉnh $k \in V$ sẽ là điểm phân chia nếu và chỉ khi tồn tại các đỉnh $i, j \in V$ sao cho thỏa mãn các điều kiện đồng thời:*

1) $(k, i) \in D$

2) $j \neq k$

3) j không phải là kế vị của i trong T

4) $lowest[i] \geq prenum[k]$ *trong đó:*

- $prenum[k]$ *là số nhận được bởi đỉnh k, khi đánh số sơ bộ của các đỉnh (tức là số của các đỉnh được thu thập thông tin nhận được ở đầu hàm thu thập thông tin trước khi gọi đệ quy)*
- $lowest[i]$ *nhận được ít nhất từ:*
  a) $prenum[i]$
  b) $prenum[w]$ *cho mỗi đỉnh w sao cho* $(i, w) \in E$ *và* $(i, w) \in D$.
  c) $lowest[w]$ *cho mỗi người thừa kế w của i trong D.*

Dựa trên Mệnh đề 2, chúng ta sẽ tìm thấy các điểm phân chia trong một đồ thị không định hướng:

Thuật toán

1) Chúng ta thực hiện thu thập thông tin theo chiều sâu từ bất kỳ đỉnh nào của G. Gọi T là cây bao phủ thu được kết quả là. Đối với mỗi đỉnh i với $prenum[i]$, chúng ta đã chỉ ra số *prenum* của đỉnh thu được trong quá trình thu thập thông tin.

2) Chúng ta đi xung quanh cây T trong LDK (xem). Đối với mỗi đỉnh i, mà chúng ta xem xét, chúng ta tính $lowest[i]$ theo cách được mô tả trong Mệnh đề 2.

3) Các điểm phân chia được xác định như sau:

3.1) Gốc của cây T là điểm phân chia khi và chỉ khi có nhiều hơn một người thừa kế.

3.2) Đỉnh i khác với gốc của T là điểm phân chia nếu và chỉ khi i có một kế trực tiếp x mà đỉnh $lowest[x] \geq prenum[i]$ đang giữ.

Độ phức tạp của thuật toán này, khi trình bày cột với danh sách các phần tử kế tiếp, là $\Theta(n + m)$. (Tại sao?) Tuy nhiên, nhận thức của chúng ta là $\Theta(n^2)$.

Trong cách triển khai được đề xuất dưới đây, chúng ta đã sử dụng biểu diễn của đồ thị với ma trận lân cận. Chúng ta sẽ không sử dụng một mảng bổ sung cho cây phủ T: để chỉ ra rằng một cạnh tham gia vào T, chúng ta sẽ sử dụng giá trị bổ sung $+2$ trong ma trận lân cận $A[][]$

Chương trình 5.20. Các điểm phân tách trong đồ thị (519artic.c)

```
#include <stdio.h>
#define MAXN 150 /*Số đỉnh cực đại trong đồ thị*/
/* Số đỉnh đồ thị */
const unsigned n = 7;
/* Ma trận kề */
char A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 0, 0, 0 },
  { 1, 0, 1, 0, 0, 1, 0 },
  { 0, 1, 0, 1, 0, 1, 0 },
  { 0, 0, 1, 0, 1, 1, 0 },
  { 0, 0, 0, 1, 0, 0, 0 },
  { 0, 1, 1, 1, 0, 0, 1 },
  { 0, 0, 0, 0, 0, 1, 0 }
};

unsigned prenum[MAXN], lowest[MAXN], cN;
unsigned min(unsigned a, unsigned b) { return (a < b) ? a : b; }

void DFS(unsigned i)
{ unsigned j;
  prenum[i] = ++cN;
  for (j = 0; j < n; j++)
    if (A[i][j] && !prenum[j]) {
      A[i][j] = 2; /* Xây dựng cây phủ T */
      DFS(j);
    }
}

/* Duyệt cây theo postorder */
void postOrder(unsigned i)
{ unsigned j;
  for (j = 0; j < n; j++)
    if (2 == A[i][j]) postOrder(j);
  lowest[i] = prenum[i];
  for (j = 0; j < n; j++)
    if (1 == A[i][j]) lowest[i] = min(lowest[i], prenum[j]);
  for (j = 0; j < n; j++)
    if (2 == A[i][j]) lowest[i] = min(lowest[i], lowest[j]);
}
```

```
void findArticPoints(void)
{ unsigned artPoints[MAXN], i, j, count;
  for (i = 0; i < n; i++) {
    prenum[i] = 0; lowest[i] = 0; artPoints[i] = 0;
  }
  cN = 0;
  DFS(0);
  for (i = 0; i < n; i++)
    if (0 == prenum[i]) {
      printf("Đồ thị không liên thông - \n");
      return;
    }
  postOrder(0);

  /* kiểm tra 3.1) */
  count = 0;
  for (i = 0; i < n; i++)
    if (2 == A[0][i]) count++;
  if (count > 1) artPoints[0] = 1;

  /* sử dụng bước 3.2) */
  for (i = 1; i < n; i++) {
    for (j = 0; j < n; j++)
      if (2 == A[i][j] && lowest[j] >= prenum[i]) break;
    if (j < n) artPoints[i] = 1;
  }

  printf(" Những điểm tách trong đồ thị là:\n");
  for (i = 0; i < n; i++)
    if (artPoints[i]) printf("%u ", i + 1);
  printf("\n");
}

int main(){
  findArticPoints();
  return 0;
}
```

Kết quả thực hiện chương trình:

Các điểm phân chia trong cột là:
2 4 6

Bài tập

▷ **5.59.** Chứng minh mệnh đề 1 và mệnh đề 2.

5.6.4. $k-$liên thông của đồ thị không định hướng

Định nghĩa 5.26. Đồ thị không định hướng được gọi là *k-liên thông đối với các đỉnh* nếu nó được liên thông và vẫn liên thông sau khi loại bỏ $k-1$ ngẫu nhiên đỉnh.

Định nghĩa 5.27. Đồ thị không định hướng được gọi là *k-liên thông đối với các cạnh* nếu nó được liên thông và vẫn liên thông sau khi loại bỏ $k-1$ ngẫu nhiên cạnh.

Cho đồ thị $G(V,E)$. Chúng ta sẽ xem xét các bài toán sau:

- Tìm k nhỏ nhất mà trong đó G không được k-liên thông theo đỉnh.
- Tìm k nhỏ nhất mà G không được k-liên thông theo các cạnh.

Chúng ta sẽ phác thảo cách hai bài toán trên có thể được quyết định bằng cách sử dụng bài toán của luồng tối đa.

k-liên thông theo các cạnh

Tầm quan trọng của luồng tối đa (có dung lượng của tất cả các bộ phận cạnh) giữa nguồn i và người tiêu dùng j bằng số cạnh tối thiểu cần thiết để phân chia hai đỉnh để chúng vẫn ở trong các thành phần liên thông khác nhau (tại sao?) . Do đó, độ k tối thiểu bằng với giá trị tối thiểu của các luồng tối đa giữa đỉnh bất kỳ i và đỉnh $n-1$ đỉnh còn lại của đồ thị.

k-liên thông về các đỉnh

Định lý 5.8 (Meninger). *Đồ thị k-liên thông theo đỉnh khi và chỉ khi mỗi cặp các đỉnh được liên thông bởi k đường không cắt nhau (không trùng các đỉnh trung gian).*

Thuật toán để kiểm tra k-liên thông về đỉnh:

1. Xây dựng số lượng $G'(V', E')$ với thuộc tính sau: Trong mỗi tập hợp các đường không giao nhau theo đỉnh trong G tương ứng với tập hợp các đường không giao nhau theo cạnh trong G'.
   Việc xây dựng được thực hiện như sau:
   - Trên mỗi đỉnh $i \in V$ tương ứng hai đỉnh $i_1 \in V'$ và $i_2 \in V'$ và cạnh $(i_1, i_2) \in E'$.
   - Trên mỗi cạnh $(i, j) \in E$ tương ứng các cạnh $(i_1, j_2) \in E$ và $(i_2, j_1) \in E'$.
2. Thông qua một luồng tối đa trong G', chúng ta kiểm tra xem có k biết đường dẫn không giao nhau nào (theo cạnh trong chúng các cạnh liên quan):
   (a) Nếu chúng tồn tại, thì suy ra đồ thị k-liên kết: Thuộc tính của những con đường không được chuyển từ các cạnh G' vào các đỉnh trong G. (Tại sao?) Áp dụng định lý của Meninger cho những đường không giao nhau trên G và chúng ta nhận được kết quả tìm kiếm.
   (b) Nếu chúng không tồn tại, đồ thị không là k-liên thông.

Bài tập

▷ **5.60.** Hãy chứng minh rằng tính chất của những con đường không giao nhau chuyển từ các cạnh trong G' vào các đỉnh trong G.

▷ **5.61.** Sửa đổi thuật toán để kiểm tra xem đồ thị có k-liên thông với các đỉnh sao cho với tìm k-liên thông tối đa của đồ thị.

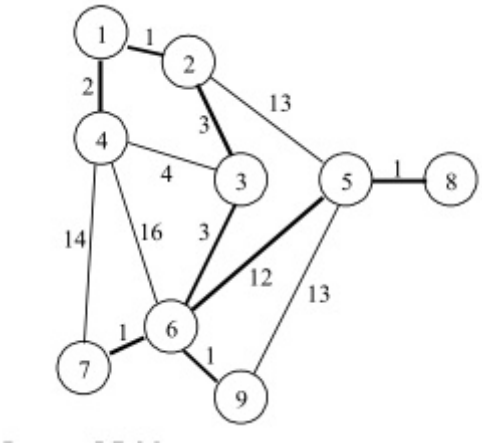
5.7. Các tập hợp con tối ưu và các tâm của đồ thị

5.7.1. Cây bao phủ tối thiểu

Để tưởng tượng một nhóm các hòn đảo, chúng ta muốn kết nối với những cây cầu theo cách mà nó có thể đạt được từ mọi hòn đảo đến nhau trong nhóm. Vì việc xây dựng cầu có liên quan đến một số chi phí nhất định, mục tiêu chúng ta sẽ là vô số cây cầu mà chúng ta xây dựng, có giá tối thiểu. Chúng ta sẽ cho rằng đối với mỗi cặp đảo chúng ta biết nếu nó có thể được kết nối trực tiếp với một cây

cầu và chi phí bao nhiêu.

Trình bày theo lý thuyết về đồ thị, bài toán trên sẽ trông như thế này: một số lượng không bị kéo $G(V, E)$ trong đó đỉnh của tập hợp V là độc lập và cạnh (i, j) tồn tại sao cho có thể xây dựng một cây đường dẫn giữa các đỉnh độc lập i và j. Trọng số $f(i, j)$ xác định giá cả có thể của nó. Tìm bao phủ (bao phủ) $T(V, D)$ của G với trọng số tối thiểu của tổng trọng số các cạnh có trong D.



Hình 5.27. Cây bao phủ nhỏ nhất trong đồ thị

Cây bao trùm T với tính chất như vậy được gọi là *cây phủ tối thiểu*. Chúng ta xem xét hai thuật toán phổ biến (xem Chương 9) giải bài toán đã cho.

- Thuật toán Kruskal

Xem xét đồ thị không hướng trọng số $G(V, E)$.

Thuật toán của Kruskal để tìm cây phủ tối thiểu

1) Tạo n bộ, như trong tập hợp thứ i đặt đặt đỉnh thứ i của đồ thị.

2) Sắp xếp các cạnh của cột theo thứ tự tăng dần (tính theo trọng số của chúng).

3) Tạo một cây rỗng $T(V, \varnothing)$. Sau khi hoàn thành thuật toán T tìm được cây bao phủ tối thiểu.

4) Thực hiện $(n-1)$ lần. Thêm cạnh $(i,j) \in E$ cho T, sao cho hoàn thành:

- Trọng số $f(i,j)$ có thể nhỏ nhất có thể (có một danh sách được sắp xếp, trọng lượng của các cạnh của đồ thị)
- Các đỉnh i và j được đặt trong các bộ khác nhau, sau mỗi bổ sung (i,j) lấy hợp các tập hợp, trong đó có i và j.

Lưu ý: Mỗi cạnh được nhìn thấy nhiều nhất một lần. Thật vậy, nếu bất kỳ bước nào và j nào trong cùng một bộ, họ sẽ vẫn còn "cùng nhau" tất cả các cách: thuật toán chỉ đơn vị nhưng không phá vỡ. Do đó, nếu một cạnh một lần là "không phù hợp", nó sẽ là thích ứng và mỗi lần lặp tiếp theo của bước 4). Quá trình xây dựng cây kết thúc tại việc tích hợp vào $n-1$ (mỗi chu trình tiếp theo sẽ đóng lại, tại sao?).

Xem xét cách thuật toán cho ví dụ trong Hình 5.7. Ban đầu, chín đầu cột được đặt trong chín bộ khác nhau (các thành phần):

{1} {2} {3} {4} {5} {6} {7} {8} {9}

Danh sách các cạnh được sắp xếp có được trong bước 2) là như sau:

(1, 2) = 1; (5, 8) = 1; (7, 6) = 1; (6, 9) = 1; (1, 4) = 2; (2, 3) = 3; (3, 6) = 3;(3, 4) = 4; (5, 6) = 12; (5, 9) = 13; (2, 5) = 13; (4, 7) = 14; (4, 6) = 14;

Chúng ta chọn cạnh đầu tiên (1, 2) và kết hợp các tập hợp, trong đó nằm là đỉnh 1 và 2. Chúng ta có được {1, 2} {3} {4} {5} {6} {7} { 8 } {9}.

Chúng ta chọn cạnh lớn nhất tiếp theo `(5, 8)`. Các tập hợp là: `{1, 2} {3} {4} {5, 8} {6} {7} {9}`.

Bên cạnh cây che phủ, chúng ta liên tiếp bao gồm `(7, 6)`, `(6, 9)`, `(1, 4)`, `(2, 3)`, `(3, 6)` và chúng ta nhận được: `{1, 2, 3, 4, 6, 7, 9} {5, 8}`.

Chúng ta bỏ qua cạnh `(3, 4)`, vì các đỉnh 3 và 4 đã nằm trong cùng một tập hợp. Chúng ta lấy cạnh tiếp theo `(5, 6)`, nơi tất cả các đỉnh của đồ thị thuộc một tập hợp chung. Lớp gỗ che phủ tối thiểu được xây dựng và bao gồm các cạnh `(1, 2)`, `(1, 4)`, `(2, 3)`, `(3, 6)`, `(6, 7)`, `(6, 9)`, `(6, 5) ))`, `(5, 8)`.

Hãy tính độ phức tạp của thuật toán. Các bước cần thiết để khởi tạo bộ là $\Theta(n)$. Hơn nữa, độ phức tạp được xác định bởi hai điều. Đầu tiên là cách chúng ta sẽ thực hiện việc sắp xếp ở bước 2). Nếu chúng ta sử dụng sắp xếp theo hình chóp (xem **??**), Độ phức tạp của sắp xếp trong trường hợp xấu nhất sẽ là $\Theta(m.\log_2 m)$. Yếu tố quyết định thứ hai là cách xác minh rằng hai đỉnh thuộc cùng một tập hợp sẽ được thực hiện (và việc hợp nhất các tập hợp trong mỗi lựa chọn cạnh ở bước 4). Chúng ta sẽ sử dụng các thành phần của kết nối (những thành phần này đã được thảo luận khi xem xét các cách trình bày đồ thị - 5.2.). Đối với mỗi thành phần của kết nối, chúng ta xây dựng một cây với gốc của bất kỳ phần nào trên cùng của nó. Khi chúng ta thêm một cạnh mới vào đồ thị, chúng ta sẽ kết hợp hai cây tương ứng với hai thành phần kết nối (ví dụ, thêm một cạnh trong đó gốc của cây này kế thừa gốc của cây kia). Việc kiểm tra xem hai đỉnh có trong cùng một thành phần kết nối hay không được thực hiện bằng cách so sánh các gốc của cây tương ứng với các thành phần này (nghĩa là các rễ này có phải là cùng một đỉnh hay không). Tiếp cận gốc của cây (bắt đầu từ một chiếc lá) có log độ phức tạp $\Theta(\log_2 n)$ và vì chúng ta thực hiện $n-1$ bước nên tổng độ phức tạp của thuật toán sẽ là: $\Theta(m.\log_2 m + n.\log_2 n)$.

Trong việc thực hiện C, đồ thị được biểu diễn bằng danh sách các cạnh trong mảng `S[]` với `M` phần tử (bằng số cạnh trong đồ thị). `S[i].x, S[i].y, S[i].Weight` cho thấy rằng cạnh `(S[i .x, S[i].y)` có trọng lượng `S[i] .weight`. Với sắp xếp hàm `(A, 0, M)`, chúng ta sắp xếp các cạnh của đồ thị theo trọng số của chúng theo thứ tự tăng dần. Chúng ta sẽ triển khai các phép toán với các thành phần của kết nối và các cây tương ứng của chúng như sau: Chúng ta sẽ giới thiệu một mảng `prev[]`, trong đó `prev[i]` là một đỉnh của cùng một tập hợp mà i thuộc về, và là đỉnh của nó (trong bộ này, không phải trong D!). Nếu đỉnh i là gốc của cây tương ứng, thì `prev[i] == 0`. Sự kết hợp của hai tập hợp được thực hiện bằng cách thực thi `prev[r2] = r1`, trong đó `r1` và `r2` là gốc của các cây mà chúng ta hợp nhất. Xác định gốc của cây có đỉnh `i` như sau:

```
root = i;
while (NO_PARENT != prev[root]) root = prev[root];
```

Ngoài ra, sau mỗi lần tìm kiếm như vậy, chúng ta sẽ thực hiện co

đường. Nó được thực hiện để giảm tổng chiều sâu của cây với chi phí thu thập thêm một lần thu thập thông tin từ i đến gốc. Sau mỗi lần tìm kiếm, các phần tử trên đường dẫn từ i đến gốc sẽ kế thừa gốc:

```
while (i != root) {
  savei = i;
  i = prev[i];
  prev[savei] = root;
}
```

Với nhiều yếu tố hơn, chúng ta có thể cải thiện nhiều hơn nữa bằng cách chia sẻ khả năng cân bằng gỗ và co ngót đường. Tuy nhiên, việc cân bằng đòi hỏi một lượng bộ nhớ bổ sung theo thứ tự của tổng số n phần tử của tập hợp, và ở các giá trị thấp hơn của n không mang lại nhiều lợi ích so với việc thu nhỏ đường mà không cân bằng. Do đó, trong quá trình thực hiện thuật toán Kruskal, chúng ta sẽ sử dụng hàm trên để thu nhỏ đường không cân bằng. Rõ ràng, số lượng các phép toán tìm kiếm và phép nối tỷ lệ với tổng số n phần tử của các tập hợp. Có thể thấy độ phức tạp của các cách tiếp cận khác nhau trong trường hợp này từ Bảng 5.3, trong đó với $\log * n$, chúng ta đã biểu thị số ứng dụng của hàm logarit là n, cần thiết để lấy 0. Các dữ liệu đầu vào được sử dụng trong mã nguồn

Số phép tính cơ sở tìm kiếm và hợp nhất	Cânbằng trên cây	Rút ngắn đường đi
n^2	KHÔNG	KHÔNG
$n.\log n$	CÓ	KHÔNG
$n.\log * n$	CÓ	CÓ

Bảng 5.3. Số lượng phép toán cơ sở tìm kiếm và hợp nhất các cây với n phần tử trong các chiến lược khác nhau

dưới đây là cho đồ thị trong Hình 5.27.

Chương trình 5.21. Cây bao phủ nhỏ nhất trong đồ thị (520kruskal.c)

```
#include <stdio.h>
/*Số đỉnh tối đa có thể trong đồ thị */
```

```
#define MAXN 200
#define NO_PARENT (unsigned)(-1)
/*Số cạnh tối đa có thể trong đồ thị */
#define MAXM 2000
const unsigned n = 9; /*Số đỉnh trong đồ thị */
const unsigned m = 14; /*Số cạnh trong đồ thị*/
struct arc {
  unsigned i, j;
  int f;
};
/* Danh sách cạnh đồ thị và trong số của chúng*/
struct arc S[MAXM] = {
  { 1, 2, 1},
  { 1, 4, 2},
  { 2, 3, 3},
  { 2, 5, 13},
  { 3, 4, 4},
  { 3, 6, 3},
  { 4, 6, 16},
  { 4, 7, 14},
  { 5, 6, 12},
  { 5, 8, 1},
  { 5, 9, 13},
  { 6, 7, 1},
  { 6, 9, 1}
};

int prev[MAXN + 1];

int getRoot(int i)
{ int root, savei;
  /*Tìm gốc của cây */
  root = i;
  while (NO_PARENT != prev[root]) root = prev[root];
  /* Rút ngắn đường*/
  while (i != root) {
    savei = i;
    i = prev[i];
    prev[savei] = root;
  }
```

```
  return root;
}

void kruskal(void)
{ int MST = 0;
  unsigned i, j;
  /* Sắp xếp danh sách với cạnh trọng số tăng dần*/
  sort(S, 0, m);

printf("Các cạnh tham gia vào cây bao phủ nhỏ nhất:\n");
for (i = 0; i < m; i++) {
  int r1 = getRoot(S[i].i);
  int r2 = getRoot(S[i].j);
  if (r1 != r2) {
    printf("(%u,%u) ", S[i].i, S[i].j);
    MST += S[i].f;
    prev[r2] = r1;
  }
}
printf("\nGiá cây bao phủ nhỏ nhất là %d.\n", MST);
}

int main() {
  unsigned i;
  for (i = 0; i < n + 1; i++) prev[i] = NO_PARENT;
  kruskal();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Các cạnh tham gia vào cây bao phủ nhỏ nhất:
(1.2) (5.8) (6.7) (6.9) (1.4) (2.3) (3.6)
Giá cây bao phủ nhỏ nhất là 24.
```

Bài tập

▷ **5.62.** Hãy chứng minh rằng thuật toán của Kruskal hoạt động chính xác.

▷ **5.63.** Thuật toán của Kruskal có hoạt động đối với một đồ thị

không liên thông không? Nếu vậy, tiêu chí dừng việc bổ sung thêm cạnh mới là gì?

▷ **5.64.** Thuật toán của Kruskal có hoạt động đối với một đồ thị có định hướng không?

- Thuật toán của Prim

Một giải pháp thay thế cho thuật toán của Kruskal để tìm cây bao trùm nhỏ nhất là thuật toán Prim.

Thuật toán Prim

1) Chúng ta bắt đầu xây dựng cây bao trùm nhỏ nhất từ đỉnh s bất kỳ: ban đầu cây sẽ là $T(H, D)$, $H = s, D = \emptyset$.

2) Lặp lại $n - 1$ lần:

2.1) Chúng ta chọn cạnh $(i, j) \in E$, sao cho:

- $i \in H, j \in V \setminus H$;
- $f(i, j)$ là cực tiểu.

2.2) Thêm đỉnh j vào H và cạnh (i, j) vào D.

Đây là cách thuật toán được áp dụng cho ví dụ trong Hình 5.27: Chúng ta chọn một đỉnh ban đầu tùy ý, ví dụ 1. Cạnh nhỏ nhất nối nó với một đỉnh không tham gia vào H là $(1, 2)$. Thêm đỉnh 2 vào H và cạnh $(1, 2)$ vào D, thành D. Cạnh nhỏ nhất tiếp theo nối đỉnh H với đỉnh không thuộc H là $(1, 4)$. Tiếp theo, các sườn được chọn sẽ là $(2, 3), (3, 6), (6, 7), (6, 9), (6, 5)$ và $(5, 8)$, với gỗ phủ tối thiểu là được xây dựng.

Hãy triển khai thuật toán được mô tả. Bước cần được chỉ rõ thêm là 2.1). Nếu chúng ta áp dụng phương pháp tiếp cận trực tiếp, trong đó chúng ta luôn kiểm tra và duy trì khoảng cách nhỏ nhất giữa các cặp đỉnh thuộc và không thuộc H, chúng ta sẽ thu được thuật toán có độ phức tạp $\Theta(n^3)$. Để tránh kiểm tra nhiều hơn n lần, chúng ta sẽ giới thiệu thêm một mảng $T[]$ - trong đó chúng ta sẽ giữ khoảng cách ngắn nhất đến các đỉnh (không nằm trên H) chưa được xem xét, tức là khoảng cách từ cây đến mỗi đỉnh ngoài cây:

- Giả sử rằng đỉnh bắt đầu là s, chúng ta khởi tạo ở đầu:
  - $T[j] = A[s][j], j = 1, 2, ..., n.$, cho mỗi $(s, j) \in E$
  - $T[j] = MAX_V ALUE$, ngược lại.

- Ở mỗi bước:
  - Ta thấy rằng j mà $T[j]$ có giá trị nhỏ nhất.
  - Với mỗi $k \ni H$ ta thực hiện $T[k] = \min(T[k], A[j][k])$.

Sau đây là mã nguồn của chương trình thực hiện thuật toán được mô tả.

Chương trình 5.22. Thuật toán Prim tìm cây bao trùm (521prim.c)

```
#include <stdio.h>
/* Số đỉnh lớn nhất có thể */
#define MAXN 150
#define MAX_VALUE 10000
/* Số đỉnh của đồ thị */
const unsigned n = 9;
/* Ma trận trọng số của đồ thị */
const int A[MAXN][MAXN] = {
  { 0,  1,  0,  2,  0,  0,  0,  0,  0 },
  { 1,  0,  3,  0, 13,  0,  0,  0,  0 },
  { 0,  3,  0,  4,  0,  3,  0,  0,  0 },
  { 2,  0,  4,  0,  0, 16, 14,  0,  0 },
  { 0, 13,  0,  0,  0, 12,  0,  1, 13 },
  { 0,  0,  3, 16, 12,  0,  1,  0,  1 },
  { 0,  0,  0, 14,  0,  1,  0,  0,  0 },
  { 0,  0,  0,  0,  1,  0,  0,  0,  0 },
  { 0,  0,  0,  0, 13,  1,  0,  0,  0 }
};

char used[MAXN];
unsigned prev[MAXN];
int T[MAXN];

void prim(void)
{ int MST = 0; /*ở đây tích lũy giá của cây bao phủ tối thiểu*/
  unsigned i, k;
  /* Khởi tạo */
  for (i = 0; i < n; i++) { used[i] = 0; prev[i] = 0; }
  used[0] = 1; /* Chọn đỉnh bắt đầu bất kỳ */
  for (i = 0; i < n; i++)
    T[i] = (A[0][i]) ? A[0][i] : MAX_VALUE;
  for (k = 0; k < n - 1; k++) {
    /* Tìm cạnh nhỏ nhất tiếp theo */
```

```
    int minDist = MAX_VALUE, j = -1;
    for (i = 0; i < n; i++)
    if (!used[i])
      if (T[i] < minDist) {
        minDist = T[i];
        j = i;
      }
  used[j] = 1;
  printf("(%u,%u) ", prev[j] + 1, j + 1);
  MST += minDist;
  for (i = 0; i < n; i++)
    if (!used[i] && A[j][i]) {
      if (T[i] > A[j][i]) {
        T[i] = A[j][i];
        /* lưu cạnh phía trước để in được cạnh tối thiểu tiếp theo*/
      prev[i] = j;
      }
    }
  }
  printf("\nGiá của cây phủ nhỏ nhất là %d.\n", MST);
  printf("\n");
}

int main() {
  prim();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
(1.2) (1.4) (2.3) (3.6) (6.7) (6.9) (6.5) (6.5)
Giá của cây bao phủ tối thiểu là 24.
```

Dễ dàng nhận thấy rằng độ phức tạp của thuật toán Prim được thực hiện theo cách này là bậc hai về số đỉnh của đồ thị (Tại sao?).

Với việc lựa chọn cấu trúc dữ liệu cẩn thận hơn (ví dụ: nếu sử dụng cấu trúc hình chóp [Cormen, Leiserson, Rivest-1997]), độ phức tạp của thuật toán Prim có thể đạt đến $\Theta(m + n.log_2 n)$.

Đối với trường hợp đặc biệt của đồ thị thưa, tức là khi $m \in O(n)$, thuật toán Prim thậm chí còn hiệu quả hơn, với độ phức

tạp $\Theta(m.\beta(m,n))$ và thậm chí $\Theta(m.\log_2(\beta(m,n)))$, ở đâu ([Cormen, Leiserson, Rivest-1997] [Fredman, Tarjan-1987]):

$$\beta(m,n) = \min i \text{ sao cho } \underbrace{\log(\log(...\log(n)))}_{i \text{ lần}} < \frac{m}{n}$$

Bài tập

▷ **5.65.** Chứng minh rằng thuật toán Prim hoạt động chính xác.

▷ **5.66.** Thuật toán của Prim có hoạt động đối với một đồ thị không liên quan không?

▷ **5.67.** Thuật toán của Prim có hoạt động đối với một đồ thị có định hướng không?

▷ **5.68.** Những tối ưu hóa nào có thể được áp dụng cho các thuật toán của Prim và Kruskal cho trường hợp trọng số của các cạnh của đồ thị nằm trong khoảng $[1, n]$?

- Cây bao phủ tối thiểu một phần

Chúng ta sẽ xem xét sửa đổi Bài toán xây dựng Cây bao phủ nhỏ nhất.

Định nghĩa 5.28. Một đồ thị $G(V, E)$ đã cho. Với k cho trước, $k \leq n$ *cây bao phủ một phần cực tiểu* $T_k(V_k, D)$ được gọi là cây chứa đúng k đỉnh, sao cho:

- $V_k \subseteq V$.
- $D \subseteq E$.
- tổng trọng số của các cạnh liên quan đến D là nhỏ nhất.

Thuật toán tìm cây bao phủ nhỏ nhất một phần:

Đối với mỗi đỉnh của đồ thị, chúng ta sẽ cố gắng xây dựng một cây bao phủ tối thiểu tương ứng theo thuật toán Prim, nhưng chứa đúng k đỉnh. Chúng ta sẽ ngắt thuật toán tại thời điểm $k-1$ cạnh đã được thêm vào (tức là $k-1$ bước đã được thực hiện). Theo cách này, chúng ta sẽ lấy n cây (mỗi đỉnh một cây), từ đó chúng ta sẽ chọn ra cây có giá thấp nhất.

Chúng ta cung cấp cho người đọc việc triển khai sự thích ứng của thuật toán Prim.

Bài tập

▷ **5.69.** Hãy chứng minh thuật toán được đề xuất để xây dựng cây bao phủ tối thiểu một phần trên cơ sở thuật toán Prim là đúng.

▷ **5.70.** Hãy triển khai sửa đổi đề xuất của thuật toán Prim.

5.7.2.

5.7.2. Tập đỉnh độc lập

Chúng ta sẽ nhắc lại hai định nghĩa từ đầu chương: Một đồ thị $G(V, E)$ được gọi là đầy đủ nếu tồn tại một cạnh (i, j) với mọi $i, j \in V$. Số lần click là số đỉnh trong đồ thị con đầy đủ lớn nhất $G'(V', E')$ của G. Phần nghịch đảo của đồ thị đầy đủ là đồ thị trống - không chứa cạnh. Chúng ta sẽ xem xét hai bài toán và xác định một khái niệm, trái ngược với số lần click.

Bài toán 1: Một nhóm gồm n người được đã cho, trong đó cứ hai người là bạn hoặc không phải là bạn của nhau. Chọn số lượng tối đa để mọi người trong nhóm đã chọn là bạn bè.

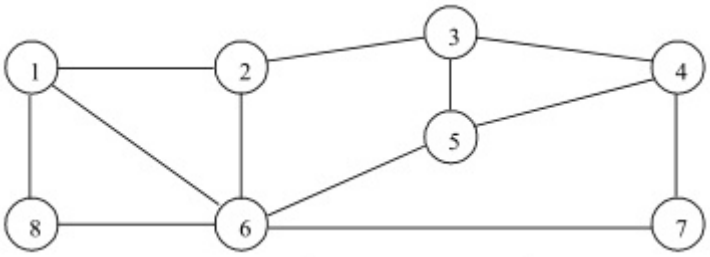
Bài toán 2: Một nhóm gồm n người được đã cho, trong đó cứ hai người là hoặc không phải là kẻ thù của nhau. Chọn số người tối đa để không có hai người trong nhóm được chọn là kẻ thù.

Chúng ta sẽ giải quyết bài toán thứ hai. Câu thứ nhất có thể được giải quyết với sự trợ giúp của câu thứ hai (chúng ta để nó như một bài tập để người đọc tìm ra cách, sau khi làm quen với tài liệu thêm).

Định nghĩa 5.29. Một đồ thị vô hướng $G(V, E)$ được cho. Tập hợp $H, H \subseteq V$ được gọi là *tập các đỉnh độc lập* nếu nó không chứa các đỉnh liền kề.

Định nghĩa 5.30. Số phần tử của tập đỉnh độc lập có số đỉnh lớn nhất được gọi là *số độc lập* của G.

Nói cách khác, số lượng độc lập của một đồ thị vô hướng là số đỉnh trong số các đỉnh lớn nhất của nó trong một đồ thị con trống.



Hình 5.28. Đồ thị vô hướng

Ví dụ, cho đồ thị trong Hình 5.28 các tập đỉnh độc lập là `{7, 8, 2}`, `{3, 1}`, `{7, 8, 2, 5}`. Số độc lập của đồ thị là 4.

- Tập hợp độc lập tối đa

Định nghĩa 5.31. Một tập độc lập H trong đồ thị vô hướng được gọi là *cực đại* nếu không có tập độc lập H' nào khác chứa H như một tập con thực sự.

Ví dụ, trong cột của Hình 5.28, tập độc lập lớn nhất là $H_1 = \{7, 8, 2, 5\}$, trong khi $H_2 = \{7, 8, 2\}$ thì không, vì nó được chứa trong H_1. Các tập hợp độc lập tối đa cũng là `{1, 3, 7}` và `{4, 6}`.

Thuật toán tìm tất cả các tập độc lập cực đại

Chúng ta hãy xem xét đồ thị $G(V, E)$. Đầu tiên chúng ta hãy xem xét một thuật toán để tìm một tập độc lập tối đa T:

1) Cho S và T là các tập rỗng lúc đầu.

2) Ta chọn một đỉnh tùy ý từ đồ thị $i \in V, i \notin S \cup T$.

3) Thêm i vào T. Trong S thêm tất cả các đỉnh kề với i.

4) Lặp lại các bước 2) và 3) cho đến khi chúng ta đạt đến vị trí mà mỗi đỉnh của đồ thị được bao gồm trong S hoặc T, tức là. $S \cup T = V$. Khi đó T là một tập cực đại độc lập.

Để tìm tất cả các tập độc lập, chúng ta sẽ sử dụng đệ quy. Trong bước 2) của thuật toán, chúng ta sẽ không chọn một đỉnh tùy ý, nhưng chúng ta sẽ khảo sát điều gì sẽ xảy ra khi thêm mỗi đỉnh tự do của đồ thị, tức là. không thuộc S và T. Điều này sẽ được thực hiện bởi hàm đệ quy `maxSubSet (last)`

```
maxSubSet(last) {
  if (<S hợp T = V>) {
    <In ra  tập hợp độc lập tìm được>;
    return;
  }
  for (<mỗi đỉnh i | i  thuộc V, i  thuộc S,  i  thuộc T, i  > last ) {
    <Thêm i vào T>;
    <Thêm vào S nối mọi đỉnh với i>;
    maxSubSet(i);
    <Loại i  khỏi T>;
    <Khôi phục S trước gọi hồi quy>;
  }
}
```

Trong đoạn trên, cuối cùng là đỉnh được thêm vào cuối cùng trong tập độc lập tối đa mà chúng ta đang xây dựng. Chúng ta chỉ chọn những đỉnh i mà $i > last$. Điều này là cần thiết để tránh tạo các tập hợp độc lập (chẳng hạn như `{1, 3, 7}` và `{1, 7, 3}`, v.v.).

Sau đây là mã nguồn của chương trình. Đồ thị được biểu diễn bằng ma trận lân cận và dữ liệu đầu vào được đặt dưới dạng hằng số.

Chương trình 5.23. Tập độc lập cực đại trong đồ thị (522maxindep.c)

```
#include <stdio.h>
/* Số đỉnh cực đại có thể */
#define MAXN 200
/* Số đỉnh trong đồ thị  đã cho */
const unsigned n = 8;
/* Ma trận kề của đồ thị  */
const char A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 0, 1, 0, 1 },
  { 1, 0, 1, 0, 0, 1, 0, 0 },
  { 0, 1, 0, 1, 1, 0, 0, 0 },
  { 0, 0, 1, 0, 1, 0, 1, 0 },
  { 0, 0, 1, 1, 0, 1, 0, 0 },
  { 1, 1, 0, 0, 1, 0, 1, 1 },
  { 0, 0, 0, 1, 0, 1, 0, 0 },
  { 1, 0, 0, 0, 0, 1, 0, 0 } };

unsigned S[MAXN], T[MAXN], sN, tN;
```

```
void print(void)
{ unsigned i;
  printf("{ ");
  for (i = 0; i < n; i++)
    if (T[i]) printf("%u ", i + 1);
  printf("} \n");
}

void maxSubSet(unsigned last)
{ unsigned i, j;
  if (sN + tN == n) {
    print();
    return;
  }
  for (i = last; i < n; i++) {
    if (!S[i] && !T[i]) {
      for (j = 0; j < n; j++)
        if (A[i][j] && !S[j]) {
          S[j] = last+1; sN++;
        }
      T[i] = 1; tN++;
      maxSubSet(i+1); /* Hồi quy */
      T[i] = 0; tN--;
      for (j = 0; j < n; j++)
        if (S[j] == last+1) { S[j] = 0; sN--; }
      }
    }
  }

int main() {
  unsigned i;
  printf("Tất cả tập độc lập tối đa trong đồ thị là:\n");
  sN = tN = 0;
  for (i = 0; i < n; i++) S[i] = T[i] = 0;
  maxSubSet(0);
  return 0;
}
```

Kết quả thực hiện chương trình:

Tất cả các tập độc lập tối đa trong đồ thị là:
{1 3 7}
{1 4}
{1 5 7}
{2 4 8}
{2 5 7 8}
{3 6}
{3 7 8}
{4 6}

Bài tập

▷ **5.71.** Chứng minh rằng thuật toán được đề xuất để tìm tất cả các tập độc lập tối đa hoạt động đúng đắn.

▷ **5.72.** Đề xuất cách giải quyết bài toán trên.

5.7.3. Tập đỉnh trội

Định nghĩa 5.32. Một đồ thị có định hướng $G(V, E)$ đã cho. *Tập hợp các đỉnh trội* được gọi là tập S sao cho:

- $S \subseteq V$
- Mọi đỉnh không thuộc S đều là con trực tiếp của đỉnh nào đó từ S.

Định nghĩa 5.33. Một tập hợp đỉnh trội được gọi là *tối thiểu* nếu nó không chứa tập hợp đỉnh trội khác làm tập hợp con của chính nó.

Hãy xem một ví dụ minh họa thuật ngữ được giới thiệu.

Bài toán: Có một số dịch giả nói các ngôn ngữ khác nhau, những người này sẽ dịch từ tiếng Việt sang các ngôn ngữ tương ứng. Trong Bảng 5.4 chúng ta đã đánh dấu người dịch bằng các chữ cái A, B, C, D, E và đối với các ngôn ngữ họ nói, chúng ta đã đặt 1 vào vị trí tương ứng.

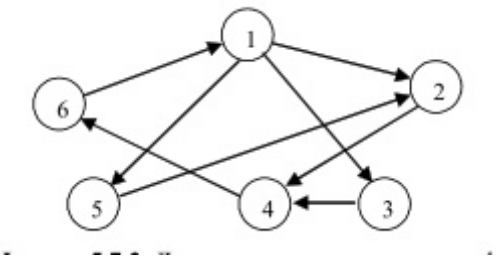
Chọn số lượng người dịch tối thiểu để chúng ta có một người dịch cho từng ngôn ngữ trong số bảy ngôn ngữ được liệt kê ở trên. Ví dụ, nếu chúng ta chọn B, C và D, điều kiện bắt buộc sẽ được đáp ứng.

	A	B	C	D	E
Tiếng Anh	1	0	1	1	0
Tiếng Pháp	1	1	0	0	0
Tiếng Đức	0	1	0	0	0
Tiếng Ý	1	0	0	1	0
Tiếng Nga	0	1	1	0	1
Tiếng Trung	0	0	0	1	1
Tiếng Phần Lan	0	0	1	0	0

Bảng 5.4. Người dịch.

Bài toán này và các bài toán khác, mà chúng ta sẽ xem xét sau này, được giải quyết bằng cách tìm các tập hợp chi phối tối thiểu trong một đồ thị (Trong ví dụ này, độ phức tạp tính toán đa thức [Manev-1996].).

Tập đỉnh trội có thể có nhiều hơn một và với một số phần tử khác nhau. Nếu chúng ta biểu thị bằng $p_1, p_2, ..., p_k$ số phần tử của mỗi tập đỉnh trội tối thiểu của một đồ thị đã cho, thì $p = min p_1, p_2, ..., p_k$ được gọi là *số trội* đối với đồ thị.



Hình 5.29. Tập đỉnh trội trong đồ thị

Cho đồ thị Hình 5.29 các tập đỉnh trội là `{1, 4}`, `{1, 4, 6}`, `{3, 5, 6}`, v.v., và các tập hợp `{1, 4}` và `{3, 5, 6}` là các tập hợp chi phối tối thiểu. `{1, 4, 6}` chiếm ưu thế nhưng không tối thiểu (bản thân nó chứa `{1, 4}`). Số trội của đồ thị là 2.

Định nghĩa 5.34. Một đồ thị có hướng $G(V, E)$ và một tập $T, T \subseteq V$

được cho trước. *Phủ* của tập T được gọi là tập $P(T)$, bao gồm tất cả các đỉnh của T và tất cả các đỉnh kề với một đỉnh của T.

Thuật toán để tìm tất cả các tập đỉnh trội tối thiểu

Cho T là tập rỗng. Ở mỗi bước, chúng ta sẽ thêm một đỉnh mới vào T, sao cho sau khi bổ sung, điều kiện sau vẫn được đáp ứng:

(*) không có đỉnh $i \in T$ sao cho $P(T \setminus \{i\}) \equiv P(T)$.

Nếu ở bước nào đó $P(T) \equiv V$, thì chúng ta đã tìm thấy một tập hợp trội tối thiểu.

Để tìm tất cả các tập hợp trội tối thiểu, chúng ta sẽ xây dựng T theo tất cả các cách có thể bằng cách vết cạn hoàn toàn.

Trong cách triển khai được hiển thị bên dưới, hàm `ok()` xác minh rằng điều kiện trên được đáp ứng cho bất kỳ tập nào T. Việc xây dựng T được thực hiện theo sơ đồ đệ quy sau:

```
void findMinDom(int last) {
  if (P(T) == V) {
    <In ra tập trội tìm được>;
      return;
  }
  for (<mỗi đỉnh i của đồ thị, i >= last>) {
    T = T hợp với {i};
    if (ok()) findMinDom(i+1);
    T = T\{i};
  }
}
```

Triển khai mã nguồn

Chương trình 5.24. Tập trội tối thiểu trong đồ thị (523mindom.c)

```
#include <stdio.h>
/* Số đỉnh cực đại có thể */
#define MAXN 200
/* Số đỉnh trong đồ thị */
const unsigned n = 6;
/* Ma trận cạnh kề */
const char A[MAXN][MAXN] = {
  { 0, 1, 1, 0, 1, 0 },
  { 0, 0, 0, 1, 0, 0 },
```

```
  { 0, 0, 0, 1, 0, 0 },
  { 0, 0, 0, 0, 0, 1 },
  { 0, 1, 0, 0, 0, 0 },
  { 1, 0, 0, 0, 0, 0 }
};
unsigned cover[MAXN];
char T[MAXN];

void printSet(void)
{ unsigned i;
  printf("{ ");
  for (i = 0; i < n; i++)
    if (T[i])
      printf("%u ", i + 1);
  printf("}\n");
}

char ok(void)
{ unsigned i, j;
  for (i = 0; i < n; i++) if (T[i]) {
    /*kiểm tra xem lớp phủ có được bảo toàn không khi bỏ đỉnh i*/
    if (0 == cover[i]) continue;
    for (j = 0; j < n; j++) if (cover[j])
      if (!(cover[j] - A[i][j]) && !T[j])
        break; /* còn đỉnh không phủ */
    if (j == n) return 0;
  }
  return 1;
}

void findMinDom(unsigned last)
{ unsigned i, j;
  /* kiểm tra phải chăng lời giải tìm được*/
  for (i = 0; i < n; i++)
    if (!cover[i] && !T[i]) break;
  if (i == n) { printSet(); return; }
  /* không - tiếp tục xây dựng tập trội */
  for (i = last; i < n; i++) {
    T[i] = 1;
    for (j = 0; j < n; j++) if (A[i][j]) cover[j]++;
```

```
    if (ok()) findMinDom(i + 1);
    for (j = 0; j < n; j++) if (A[i][j]) cover[j]--;
    T[i] = 0;
  }
}

int main() {
  unsigned i;
  printf("Các tập trội nhỏ nhất trong đồ thị là: \n");
  for (i = 0; i < n; i++) { cover[i] = 0; T[i] = 0; }
  findMinDom(0);
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Các tập trội tối thiểu trong đồ thị là:
{1 2 6}
{1 3 6}
{1 4}
{3 5 6}
```

Bài tập

Chứng minh rằng thuật toán được đề xuất để tìm tất cả các tập trội ối thiểu hoạt động chính xác.

5.7.4. Tập cơ sở

Bài toán: Một mạng máy tính N được đã cho và các kênh để liên lạc giữa chúng được biết đến. Xác định một số lượng tối thiểu của máy tính (cơ sở) sao cho khi một số thông tin cần được phổ biến, nó có thể tiếp cận tất cả các máy tính trong mạng từ cơ sở.

Trong Hình 5.30. Máy tính được biểu diễn dưới dạng các đỉnh của biểu đồ định hướng. Một cơ sở có thể trong biểu đồ là bộ `{4, 7, 9}`. Nó không phải là duy nhất: như vậy là `{5, 7, 9}`, `{7, 8, 9}` và những người khác.

Định nghĩa 5.35. Một đồ thị có định hướng $G(V, E)$ đã cho. *Một tập cơ sở* của các đỉnh (hoặc chỉ tập cơ sở) được gọi là tập hợp $T \subseteq V$ sao cho:

- mỗi đỉnh có thể tiếp cận được bằng một đỉnh T
- T cực tiểu theo nghĩa là không có tập con thích hợp của T tương ứng với điều kiện đầu tiên cho khả năng tiếp cận.

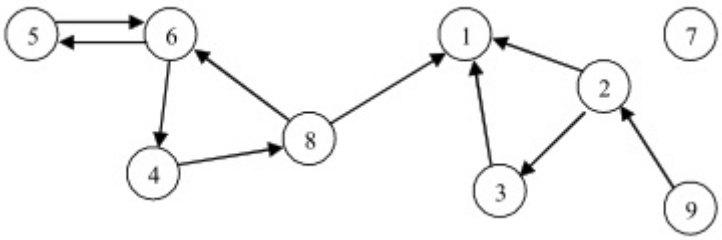
Mệnh đề 5.4. *Đây là cơ sở nếu và chỉ khi nó được thực hiện đồng thời:*

1) Mỗi đỉnh không tham gia T có thể đạt tới một đỉnh T.

2) Không có đỉnh $i \in T$ nào có thể đạt tới đỉnh khác từ T.

Mệnh đề 5.5. *Trong một đồ thị có chu trình có định hướng, có một cơ sở T duy nhất và nó bao gồm tất cả các đỉnh có bậc đầu vào bằng 0 (tức là chúng không có đỉnh trước nó).*

Theo định nghĩa (cũng như từ Mệnh đề 5.4 và Mệnh đề 5.5), suy ra các cơ sở trong đồ thị có thể là một số, nhưng chúng phải bằng số đỉnh.



Hình 5.30. Đồ thị có hướng

Thuật toán tìm cơ sở

Cơ sở chúng ta sẽ gọi những đỉnh tham gia vào cơ sở. Chúng ta sẽ nhập một `base[]`, chẳng hạn như `base[i] == 1`, khi i là cơ sở và `base[i] == 0`, ngược lại. Ban đầu, chúng ta đánh dấu tất cả các đỉnh là cơ sở. Chúng ta duyệt theo chiều sâu mọi đỉnh k, với `cbase[k] == 1`. Khi đi ngang, tất cả các đỉnh $i (i \neq k)$ mà chúng ta đi qua sẽ được đánh dấu là `base[i] = 0` . Do đó, các đỉnh vẫn được đánh dấu là cơ sở sẽ tạo thành một cơ sở trong đồ thị. Độ phức tạp của thuật toán được mô tả là $\Theta(n + m)$, bởi vì mỗi đỉnh và mỗi cạnh được vượt qua nhiều nhất một lần (Tại sao?). Triển khai đầy đủ như sau:

Chương trình 5.25. Các đỉnh tạo thành tập cơ sở trong đồ thị (524v-base.c)

```
#include <stdio.h>
/* Số đỉnh cực đại có thể trong đồ thị */
#define MAXN 200
/* Số đỉnh trong đồ thị */
const unsigned n = 9;
/* Ma trận liền kề của đồ thị */
const char A[MAXN][MAXN] = {
  { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 1, 0, 1, 0, 0, 0, 0, 0, 0 },
  { 1, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 1, 0 },
  { 0, 0, 0, 0, 0, 1, 0, 0, 0 },
  { 0, 0, 0, 1, 1, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 1, 0, 0, 0, 0, 1, 0, 0, 0 },
  { 0, 1, 0, 0, 0, 0, 0, 0, 0 }
};

char used[MAXN], base[MAXN];

void DFS(unsigned i)
{ unsigned k;
  used[i] = 1;
  for (k = 0; k < n; k++)
    if (A[i][k] && !used[k]) {
      base[k] = 0;
      DFS(k);
    }
}

void solve(void) {
  unsigned i, j;
  for (i = 0; i < n; i++) base[i] = 1;
  for (i = 0; i < n; i++)
    if (base[i]) {
      for (j = 0; j < n; j++) used[j] = 0;
      DFS(i);
    }
```

```
}

void print() {
  unsigned i, count = 0;
  printf("Các đỉnh tạo ra tập sơ sở trong đồ thị là: \n");
  for (i = 0; i < n; i++)
    if (base[i]) { printf("%u ", i + 1); count++; }
  printf("\nSố đỉnh trong tập cơ sở : %u\n", count);
}

int main() {
  solve();
  print();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
Các đỉnh tạo thành cơ sở trong đồ thị là:
4 7 9
Số lượng đỉnh trong cơ sở: 3
```

Bài tập

▷ **5.73.** Chứng minh Mệnh đề 5.4 và Mệnh đề 5.5).

▷ **5.74.** Hãy chứng minh rằng thuật toán được đề xuất để tìm tập cơ sở dữ liệu hoạt động đúng.

5.7.5. Tâm, bán kính và đường kính

Các bài toán để chọn một hoặc một số đỉnh cố định của đồ thị sao cho đáp ứng một tiêu chí tối ưu nhất định được sử dụng rộng rãi. Ví dụ, chúng ta hãy xem các đỉnh của một đồ thị là các vùng lân cận và giải thích các cạnh liên thông là các đường dẫn trực tiếp giữa chúng. Chúng ta đang tìm kiếm một đỉnh từ đồ thị để phục vụ như một dịch vụ khẩn cấp (hoặc cứu hỏa, cảnh sát, tổng đài điện thoại, v.v.) sao cho khoảng cách đến vùng lân cận xa nhất là nhỏ nhất. Bài toán này được gọi là *tối ưu hóa tình huống xấu nhất*.

Cũng có thể tìm kiếm tập hợp các đỉnh p (một số dịch vụ khẩn

cấp), trong trường hợp đó, mục đích sẽ là giảm thiểu khoảng cách xa nhất $S(i)$ từ đỉnh i đến dịch vụ khẩn cấp gần nhất.

Định nghĩa 5.36. Một đồ thị có hướng liên thông $G(V, E)$ đã cho. *Số lượng phân tách bên ngoài* $S_o(i)$ cho một đỉnh $i \in V$ cho trước được gọi là $d(i, j)$ lớn nhất, đối với mỗi $j \in V$, trong đó $d(i, j)$ là độ dài của đường đi nhỏ nhất từ i đếnj. Nghĩa là, số này là đường đi ngắn nhất đến đỉnh xa nhất của i. Tương tự, *số lần phân tách trong* $S_t(i)$ cho đỉnh i được gọi là giá trị lớn nhất của khoảng cách nhỏ nhất từ một số đỉnh $j \in V$ đến đỉnh i.

Định nghĩa 5.37. Từ tất cả các đỉnh, chúng ta sẽ chọn một đỉnh $i \in V$ mà $p_o = S_o(i)$ là cực tiểu. Đỉnh i được gọi là *tâm ngoài* của đồ thịG, và số p_o được gọi là bán kính ngoài của đồ thị. Tương tự, chúng ta sẽ chọn một đỉnh j sao cho $p_t = S_t(j)$ là cực tiểu. Đỉnh j này được gọi là *tâm bên trong* của G, và p_t được gọi là bán kính bên trong của đồ thị.

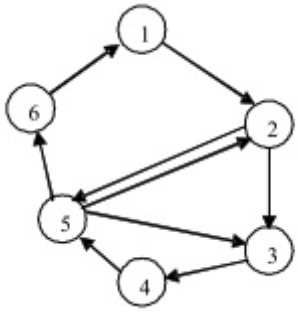
Định nghĩa 5.38. Gọi $S_{ot}(i)$ biểu thị giá trị nhỏ nhất của $d(i, j) + d(j, i)$, được tính cho mỗi $j \in V$. Gọi v là đỉnh mà $S_{ot}(v)$ là cực tiểu. Khi đó v được gọi là tâm ngoài-trong (hoặc chỉ tâm), và giá trị tương ứng $S_{ot}(v)$ được gọi là bán kính ngoài-trong (hoặc chỉ bán kính) của đồ thị.

Lưu ý rằng trong cả ba định nghĩa, đỉnh được đề cập có thể không phải là định nghĩa duy nhất.

Ví dụ về một trung tâm trong đồ thị: Chúng ta muốn đặt một đồn cảnh sát sao cho nếu S là khoảng cách nhỏ nhất từ nó đến bất kỳ vùng lân cận nào cộng với khoảng cách nhỏ nhất trên đường về thì S đến vùng lân cận xa nhất sẽ là nhỏ nhất.

Trước khi tổng quát bài toán (p-tâm và p-bán kính trong đồ thị), chúng ta sẽ chỉ ra tâm trong đồ thị như thế nào. Hãy xem xét một ví dụ cụ thể (xem Hình 5.31).

Chúng ta đã tìm thấy các đường đi ngắn nhất trong đồ thị (được hiển thị bên phải trong Hình 5.31) theo thuật toán Floyd. Sau bước cơ bản này, tâm và bán kính được xác định bằng cách thực hiện trực tiếp định nghĩa ở trên. Độ phức tạp của thuật toán là $\Theta(n^3)$. (Tại sao?)



Hình 5.31. Tâm và bán kính trong đồ thị (các cạnh có trọng số 1)

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

Bảng 5.5. Tâm của đồ thị: đỉnh 2

Trong cách triển khai bên dưới, đồ thị định hướng được biểu diễn bằng ma trận trọng số `A[][]`, vì số đỉnh của nó là n. Dữ liệu đầu vào được đặt ở đầu chương trình dưới dạng hằng số.

Chương trình 5.26. Tâm và bán kính của đồ thị (525a-center.c)

```
#include <stdio.h>
/* Số đỉnh lớn nhất có thê */
#define MAXN 150
#define MAX_VALUE 10000
/* Số đỉnh đồ thị đã cho */
const unsigned n = 6;
/* Ma trận trọng số*/
int A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 0, 0 },
  { 0, 0, 1, 0, 1, 0 },
  { 0, 0, 0, 1, 0, 0 },
```

```
    { 0, 0, 0, 0, 1, 0 },
    { 0, 1, 0, 0, 0, 1 },
    { 1, 0, 0, 0, 0, 0 }
};

    /* Tìm độ dài của đường ngắn nhất giữa mọi cặp điểm*/
void floyd(void)
{ unsigned i, j, k;
  /*Các giá trị 0 thay đổi của  MAX_VALUE */
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    if (A[i][j] == 0)
      A[i][j] = MAX_VALUE;

  /* Thuật toán floyd*/
  for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if (A[i][j] > (A[i][k] + A[k][j]))
          A[i][j] = A[i][k] + A[k][j];
  for (i = 0; i < n; i++)
    A[i][i] = 0;
}

void findCenter(void)
{ unsigned i, j, center;
  int max, min;
  min = MAX_VALUE;
  /* Sot(Xi) = max { Vj [d(Xi, Xj)+d[Xj,Xi])] }, tâm là đỉnh X* */
 /* với nó  Sot(X*) là nhỏ nhất */
  for (i = 0; i < n; i++) {
    max = A[i][0] + A[0][i];
    for (j = 0; j < n; j++)
      if ((i != j) && ((A[i][j] + A[j][i]) > max))
        max = (A[i][j] + A[j][i]);
  if (max < min) {
    min = max; center = i;
  }
 }
  printf("Tâm của đồ thị là đỉnh %u\n", center + 1);
```

```
    printf("Bán kính của đồ thị là %u\n", min);
}

int main() {
    floyd();
    findCenter();
    return 0;
}
```

Kết quả thực hiện chương trình:

```
Tâm của đồ thị là đỉnh 2
Bán kính của đồ thị là 4
```

Bài tập

▷ **5.75.** Cho một ví dụ về một đồ thị với một số tâm có thể có.

▷ **5.76.** Hãy sửa đổi chương trình trên để nó tìm thấy tất cả các tâm có thể có của đồ thị.

- p-tâm và p-bán kính

Tổng quát bài toán trên là chọn đồng thời p số đỉnh tâm của đồ thị. Chúng ta muốn giảm thiểu khoảng cách xa nhất từ bất kỳ đỉnh nào và đến đỉnh gần nhất trong số các đỉnh đã chọn. Gọi $d(i,j)$ là độ dài của đường đi nhỏ nhất giữa các đỉnh i và j.

Định nghĩa 5.39. Đồ thị định hướng $G(V,E)$ và tập $Q = \{v_{i1}, v_{i2}, ..., v_{ip} | v_{ik} \in V\}$. Chúng ta đưa vào ký hiệu:

$$S(v,Q) = \min\{d(v_k,v) + d(v,v_k) | v_k \in Q\}, v \in V$$
$$S(Q) = \max\{S(v,Q) | v \in V\}$$

$S(Q)$ nhỏ nhất, được tính trên tất cả các tập con p phần tử có thể có Q của V, được gọi là p-tâm của đồ thị.

Đây là định nghĩa trong trường hợp đơn giản hơn, khi biểu đồ không có hướng:

Định nghĩa 5.40. Đồ thị vô hướng $G(V,E)$ và tập $Q = \{v_{i1}, v_{i2}, ..., v_{ip} | v_{ik} \in V\}$. Chúng ta đưa vào ký hiệu:

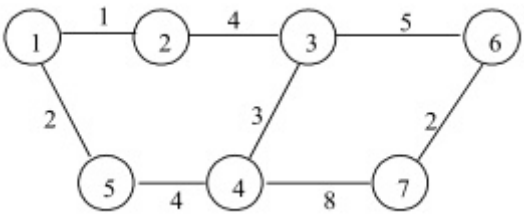
$S(v,Q) = \min\{d(v_k,v)|v_k \in Q\}, v \in V$
$S(Q) = \max S(v,Q)|v \in V$

$S(Q)$ nhỏ nhất, được tính trên tất cả các tập con p phần tử có thể có Q của V, được gọi là p-tâm của đồ thị.

Thuật toán tìm p-tâm

Chúng ta sẽ xem xét trường hợp đồ thị vô hướng. Nếu đồ thị được định hướng, những thay đổi là không đáng kể. Giải pháp bao gồm việc khai thác hoàn toàn các biến thể có thể có: với mỗi tập con p-phần tử Q, $Q \subseteq V$, chúng ta sẽ tính khoảng cách S đến đỉnh xa nhất. Trong tất cả các khả năng (tổng $\binom{p}{n}$ theo số), chúng ta sẽ giữ nguyên những khả năng cho giá trị nhỏ nhất của S.

Ban đầu, chúng ta tính toán độ dài của các đường đi nhỏ nhất giữa mỗi hai đỉnh bằng cách sử dụng thuật toán Floyd. Khi chúng ta tìm thấy ma trận có các đường đi nhỏ nhất $A[][]$, chúng ta sẽ tạo ra tất cả các tổ hợp có thể có (tất cả các tập con p-phần tử có thể có Q của V) và với mỗi tổ hợp, chúng ta sẽ tính khoảng cách từ nó đến đỉnh xa nhất. Điều này không khó lắm, vì chúng ta có ma trận Floyd theo ý của nó. Ví dụ cụ thể như Hình 5.32.



Hình 5.32. Tìm p-tâm

Chúng ta hãy tìm kiếm một 3-tâm, tức là $p = 3$ trong đồ thị Hình 5.32. Chúng ta đã tìm thấy ma trận Floyd và xem xét tập con của các đỉnh $\{2,3,6\}$:

$$S = \max\{\min\{A[i][2], A[i][3], A[i][6]\}|i = 1,2,...,7\}$$
$$= \max\{1,0,0,3,3,0,2\} = 3.$$

Hơn nữa, kiểm tra tất cả các khả năng khác, chúng ta sẽ thấy rằng $\{2,3,6\}$ là tập tối thiểu hóa S và theo đó nó là 3-tâm bắt buộc

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

Bảng 5.6. Ma trận trong số của đồ thị

trong đồ thị. Tuy nhiên, nó không phải là duy nhất! Ví dụ, đối với tập $\{1,3,6\}$ chúng ta lại nhận được 3-bán kính $S = 3$.

Điều tiếp theo là sự nhận ra của C, mà chỉ tìm thấy một giải pháp. Chúng ta đề nghị người đọc sửa đổi nó theo cách thích hợp để ta tìm ra tất cả các giải pháp.

Chương trình 5.27. Tìm p-tâm trong đồ thị trong đồ thị (524526p-center.c)

```
#include <stdio.h>
/* Số đỉnh lớn nhất có thể trong ồ thị */
#define MAXN 150
#define MAX_VALUE 10000
/* Số đỉnh đồ thị đã cho */
const unsigned n = 7;
const unsigned p = 3; /* p-tâm */
/* Ma trận trọng số của đồ thị */
int A[MAXN][MAXN] = {
  { 0, 1, 0, 0, 2, 0, 0 },
  { 1, 0, 4, 0, 0, 0, 0 },
  { 0, 4, 0, 3, 0, 5, 0 },
  { 0, 0, 3, 0, 4, 0, 8 },
  { 2, 0, 0, 4, 0, 0, 0 },
  { 0, 0, 5, 0, 0, 0, 2 },
  { 0, 0, 0, 8, 0, 2, 0 }
};
```

```
unsigned center[MAXN], pCenter[MAXN], pRadius;
/* Tìm độ dài của đường ngắn nhất giữa hai cặp đỉnh */
void floyd(void)
  { unsigned i, j, k;

  /* Các giá tr  0 thay đổi của  MAX_VALUE */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (A[i][j] == 0) A[i][j] = MAX_VALUE;

  /* Thuật toán floyd */
  for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if (A[i][j] > (A[i][k] + A[k][j]))
          A[i][j] = A[i][k] + A[k][j];
  for (i = 0; i < n; i++)
    A[i][i] = 0;
}

/* Tính  S cho tập con vừa sinh ra */
void checkSol(void)
{ unsigned i, j, cRadius = 0;
  for (j = 0; j < n; j++) {
    int bT = MAX_VALUE;
    for (i = 0; i < p; i++)
      if (A[center[i]][j] < bT) bT = A[center[i]][j];
    if (cRadius < bT) cRadius = bT;
  }
  if (cRadius < pRadius) {
    pRadius = cRadius;
    for (i = 0; i < p; i++) pCenter[i] = center[i];
  }
}

/* Tổ hợp C(n,p) - sinh ra  tất  cả  tập con p-phần tử của G */
void generate(unsigned k, unsigned last)
{ unsigned i;
  for (i = last; i < n - p + k; i++) {
    center[k] = i;
```

```
    if (k == p)
      checkSol();
    else
      generate(k + 1, i + 1);
    }
  }

  /* In ra p-tâm và p-bán kính*/
  void printPCenter(void)
  { unsigned i;
    printf("%u-Tâm của đồ thị là tập hợp các đỉnh sau: {", p);
    for (i = 0; i < p; i++) printf("%d ", pCenter[i] + 1);
    printf("}\n");
    printf("%u-bán kính của ồ thị là %u\n", p, pRadius);
  }

int main() {
  floyd();
  pRadius = MAX_VALUE;
  generate(0, 0);
  printPCenter();
  return 0;
}
```

Kết quả thực hiện chương trình:

```
3-Tâm của đồ thị là tập hợp các đỉnh sau: {1 3 6}
3-Bán kính của đồ thị là 3
```

Bài tập

▷ **5.77.** Kiểm tra xem tập $\{2,3,6\}$ có phải là p-tâm của đồ thị trong Hình 5.32 hay không.

▷ **5.78.** Cho một ví dụ về một đồ thị với một số p-tâm có thể có.

▷ **5.79.** Sửa đổi chương trình trên để nó tìm thấy tất cả các p-tâm có thể.

▷ **5.80.** Thuật toán trên có hoạt động trong trường hợp đồ thị có định hướng không? Nếu có thì có thay đổi nào được yêu cầu không (nếu có: cái gì?), Nếu không - tại sao (đưa ra một ví dụ cụ thể)?

5.7.6. Kết hợp cặp. Kết hợp cặp tối đa

Việc tìm kiếm kết hợp theo từng cặp tối đa trong một đồ thị (đối sánh) được liên kết với một số bài toán của các luồng trong biểu đồ mà chúng ta đã xem xét.

Định nghĩa 5.41. *Kết hợp cặp* trong đồ thị $G(V, E)$ được gọi là tập con $E' \subseteq E$ sao cho, trong đó mỗi đỉnh của đồ thị là điểm cuối của không quá một cạnh của E'. Các đỉnh trùng với một cạnh của E' được gọi là *phủ*, và những đỉnh không phải là đầu của các cạnh của kết hợp kép được gọi là *tự do*. Tương tự, chúng ta xác định các *cạnh bao phủ* và *tự do* tùy thuộc vào việc chúng có thuộc kết hợp kép hay không.

Định nghĩa 5.42. Một tổ hợp ghép đôi E' từ k cạnh được gọi là *cực đại* nếu với mọi kết hợp ghép đôi khác E'' của l cạnh $k \geq l$ được thỏa mãn.

Định nghĩa 5.43. Gọi $M \subseteq E$ là một kết hợp kép tùy ý trong G. Một đường đi trong G được gọi là *xen kẽ đối với* M nếu một trong hai cạnh liên tiếp của đường đi, đúng một đường tham gia vào M. Một đường đi xen kẽ với kết hợp M được gọi là *tăng* nếu đỉnh ban đầu và đỉnh cuối cùng của nó là tự do đối với M.

Thuật toán tìm kết hợp cặp tối đa

Chúng ta bắt đầu với một kết hợp kép ngẫu nhiên (có thể bao gồm một cạnh duy nhất). Ở mỗi bước, chúng ta tìm thấy một con đường tăng dần liên quan đến cặp kết hợp cuối cùng. Nếu một con đường như vậy không tồn tại, nó là tối đa. (Tại sao?) Nếu không, chúng ta xây dựng một kết hợp cặp mới có nhiều cạnh hơn tổ hợp đầu tiên như sau: Gọi M là kết hợp cặp và P là đường tăng. Sau đó, kết hợp theo từng cặp mới sẽ chứa tất cả các cạnh của M và P không thuộc cả M và P.

Chúng ta cung cấp cho người đọc một bài toán thực sự phức tạp như bài tập: vẽ một sơ đồ để tìm một đường đi tăng dần cần thiết cho việc thực hiện các thuật toán.

Trong trường hợp đồ thị hai phần, mọi thứ được đơn giản hóa rất nhiều:

Định nghĩa 5.44. Đồ thị vô hướng $G(V, E)$ được gọi là lưỡng phân nếu có một phân hoạch $V = V' \cup V'', V' \cap V'' = \emptyset$, sao cho mỗi cạnh $(i, j) \in E$ thỏa mãn $i \in V'$ và $j \in V''$.

Bài toán: Tìm kết hợp kép lớn nhất trong đồ thị hai phần.

Thuật toán 1

Chúng ta sẽ rút gọn bài toán thành bài toán tìm dòng cực đại trong đồ thị.

Cho đồ thị hai phần $G(V, E)$ bằng cách phá vỡ tập các đỉnh $V = V' \cup V''$, $V' \cap V'' = \emptyset$. Chúng ta sẽ giải quyết bài toán bằng cách tìm dòng chảy lớn nhất trong đồ thị mà:

- các đỉnh của tập V' là nguồn.
- các đỉnh của tập V'' là người tiêu dùng.
- tất cả các cạnh của đồ thị đều có độ thông qua $c(i, j) = 1$.

Thuật toán 2

Thuật toán tổng quát (được mô tả ở trên) để tìm kết hợp nhị phân tối đa trong một đồ thị tùy ý có thể dễ dàng thực hiện trong trường hợp đồ thị lưỡng phân. Trong trường hợp này, số lượng đường đi tăng lên bị giới hạn bởi $\Theta(n)$ và bạn có thể dễ dàng tìm thấy một đường đi đang tăng lên bằng cách thu thập thông tin theo chiều rộng (Làm thế nào?). Như vậy tổng độ phức tạp là $\Theta(n.(M + 1)) = \Theta(n.m)$. Ngoài ra còn có các thuật toán hiệu quả hơn để tìm kết hợp nhị phân tối đa trong đồ thị hai phần, với độ phức tạp $\Theta(\sqrt{n}.m)$ [Hopcroft-Karp-1973], sau này được tóm tắt cho một đồ thị tùy ý [Micali, Vazirani-1980].

Bài tập

▷ **5.81.** Hãy chứng minh rằng nếu không có đường tăng nào của một kết hợp đã cho trong đồ thị thì kết hợp đó là cực đại.

▷ **5.82.** Hãy vẽ sơ đồ tìm đường đi tăng dần, cần thiết cho việc thực hiện thuật toán tổng quát để tìm kết hợp tối đa.

▷ **5.83.** Hãy vẽ sơ đồ tìm đường đi tăng dần trong đường hai phần với duyệt theo chiều rộng.

▷ **5.84.** Hãy thực hiện triển khai thuật toán 1 và 2 ở trên.

5.8. Tô màu và đồ thị phẳng

5.8.1. Tô màu đồ thị và sắc số

Nhiều bài toán, chẳng hạn như lập kế hoạch quy trình sản xuất, lập lịch trình, lưu trữ và vận chuyển hàng hóa, có thể dễ dàng giải quyết nếu chúng được giảm xuống thành bài toán tô màu đồ thị tương ứng.

Định nghĩa 5.45. r-tô màu các đỉnh của đồ thị được gọi là so khớp chính xác một phần tử của một tập hợp $\{c_1, c_2, ..., c_r\}$ đã cho trên mỗi đỉnh của nó. Màu r của các cạnh của đồ thị được xác định tương tự.

Định nghĩa 5.46. Một đồ thị vô hướng $G(V, E)$ được gọi là r-sắc số nếu các đỉnh của nó có thể có màu r sao cho mọi hai đỉnh liền kề có màu khác nhau. Số r nhỏ nhất mà đồ thị có màu r được gọi là *số sắc đỉnh* của G (hoặc chỉ số màu của G) và được ký hiệu là $\gamma_V(G)$. Tương tự, số r nhỏ nhất sao cho các cạnh của đồ thị có thể tô màu r để không có các cạnh ngẫu nhiên được tô cùng màu được gọi là *số sắc cạnh* của G và được ký hiệu là $\gamma_E(G)$.

Sau đây là hai ví dụ:

Ví dụ 1: Chúng ta cần sắp xếp các báo cáo của đại hội sao cho không người tham gia nào bị buộc phải bỏ sót một báo cáo mà họ muốn tham dự. Thời lượng tối thiểu của chương trình là bao nhiêu nếu chúng ta có đủ số lượng giảng đường để có thể gửi bất kỳ báo cáo nào cùng một lúc? Trong ví dụ này, chúng ta có thể xây dựng một đồ thị G, các đỉnh của chúng là các báo cáo. Hai đỉnh sẽ được kết nối với sườn khi và chỉ khi có một người tham gia muốn đến thăm cả hai người họ. Giải pháp cho vấn đề này là tìm $\gamma_V(G)$, cung cấp thời lượng tối thiểu của chương trình.

Ví dụ 2: Mỗi người trong số n doanh nhân muốn có cuộc gặp trực tiếp với một số người khác. Chúng ta đang tìm thời gian tối thiểu mà các cuộc họp sẽ diễn ra, nếu mỗi cuộc họp kéo dài một ngày và có chính xác hai người tham gia. Vấn đề có thể được giải quyết bằng cách tìm số sắc độ của đồ thị $\gamma_E(G)$, cho chúng ta số ngày cần thiết.

- Giới hạn dưới cùng của số sắc

Trong phần tiếp theo, chúng ta sẽ xem xét các thuật toán để tìm số sắc độ của một đồ thị. Sau đó, chúng ta sẽ thấy rằng một ràng buộc có thể có ở dưới cùng của $\gamma_V(G)$ theo một số tiêu chí sẽ tiết kiệm rất nhiều thời gian tính toán. Các tiêu chí như vậy đã cho các tuyên bố sau (một số trong số đó tuân theo rõ ràng, trong khi tính hợp lệ của các tiêu chí khác được kiểm tra ít tầm thường hơn):

- $\gamma_V(G) \geq 2 \Leftrightarrow G$ chứa ít nhất một cạnh.
- $\gamma_V(G) \geq 3 \Leftrightarrow G$ chứa một vòng lặp có độ dài lẻ.
- $\gamma_E(G) \geq d_m(G)$, với $d_m(G)$ là tung độ cực đại của đỉnh G.
- $\gamma_V(G) \geq n/\gamma_V(G')$, với n là số đỉnh của G và G' là phần bù của G.
- $\gamma_V(G) \geq n^2/(n^2 - 2m)$, với n là số đỉnh và m là số cạnh của G.

- Tìm sắc số đỉnh

Thuật toán tìm $\gamma_V(G)$

1) Sử dụng các phát biểu đã thảo luận trong đoạn trước, chúng ta giới hạn số màu $\gamma_V(G)$ bên dưới bằng một số $\gamma_{V_{\min}}$.

2) `for (r = gamma_{Vmin}; r <n; r ++) {`
`if (G là r-sắc số) {Chuyển đến bước 3}`

3) r là sắc số của đồ thị.

Rõ ràng, bài toán là kiểm tra xem một đồ thị có phải là r-chromatic hay không. Vấn đề thứ hai là một bài toán NP-đầy đủ và chúng ta sẽ giải quyết nó trong **??**: Kiểm tra tuần tự tất cả các màu có thể có của mỗi đỉnh, không tiếp tục tô màu, nếu đã có hai đỉnh liền kề được tô cùng màu. Cách tiếp cận toàn diện này có hiệu suất tốt đối với đồ thị có số lượng đỉnh nhỏ, đối với đồ thị thưa thớt và đồ thị gần như hoàn chỉnh. Có các thuật toán toàn diện khác để tìm số sắc độ đỉnh. Ví dụ, có thể tìm thấy các tập độc lập lớn nhất của đồ thị: các đỉnh thuộc một tập độc lập có thể được tô cùng màu. Đối với đồ thị ngẫu nhiên, đây là một trong những thuật toán tốt nhất [Christofides-1975] [Korman-1979].

Nói chung, để đảm bảo chúng ta tìm đúng số màu chính xác,

chúng ta phải áp dụng một thuật toán dựa trên sự cạn kiệt hoàn toàn. Tuy nhiên, trong thực tế, chúng ta thường không quan tâm đến chính xác màu r tối thiểu (số màu thực), mà quan tâm đến một số giá trị gần đúng đủ tốt của nó. Đối với những trường hợp như vậy, có những thuật toán giải quyết vấn đề một cách nhanh chóng, nhưng không phải lúc nào cũng chính xác. Những điều này sẽ được thảo luận lại trong Chương 9.

Bài tập

▷ **5.85.** Cố gắng chứng minh các tiêu chí ràng buộc của $\gamma_V(G)$.

5.8.2. Đồ thị phẳng

Định nghĩa 5.47. Một đồ thị được gọi là *đồ thị phẳng*, vì vậy nó có thể được vẽ trên một mặt phẳng (vẽ bằng đồ thị với các dấu chấm và đường thẳng) sao cho không có hai cạnh nào cắt nhau.

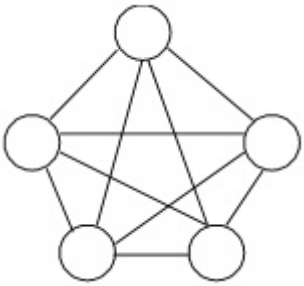
Ví dụ, một đồ thị mô tả các hành lang của một phòng trưng bày là phẳng. Bản đồ chính trị của thế giới cũng có thể được biểu diễn bằng một biểu đồ phẳng (các đỉnh tương ứng với các quốc gia và mỗi cạnh thể hiện một đường biên giới trên bộ chung giữa hai quốc gia).

Biết rằng bất kỳ đồ thị phẳng nào cũng có thể được vẽ trong mặt phẳng sao cho không những không có hai cạnh nào cắt nhau mà còn có các cạnh là *đoạn thẳng*.

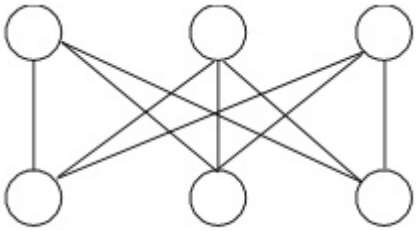
Năm 1933 Kuratovski lần đầu tiên xác định chặt chẽ khái niệm độ phẳng của một đồ thị và đưa ra điều kiện cần và đủ cho đồ thị phẳng.

Chúng ta sẽ nhắc lại Định nghĩa 5.8. và chúng ta sẽ cung cấp thêm một vài định nghĩa cần thiết cho định lý phẳng cơ bản của đồ thị:

Cho các đồ thị vô hướng $G(V, E)$ và $G'(V, E')$ và với mỗi $i, j \in V$ thì cạnh (i, j) thuộc đúng một trong hai tập E' và E. Khi đó G'được gọi là *phần bù* của G. Nếu $(i, j) \in E$ thỏa mãn với mỗi $i, j \in V$ thì đồ thị được gọi là *đầy đủ*. Chúng ta sẽ biểu thị một đồ thị đầy đủ với n đỉnh bằng K_n (K_5 được thể hiện trong Hình 5.33).



Hình 5.33. K_5



Hình 5.34. $K_{3,3}$

Định nghĩa 5.48. Nếu trong đồ thị hai phần $G(V, E)$ với phân hoạch $V = V' \cup V'', V' \cap V'' = \emptyset$, với mỗi $i \in V'$ và $j \in V''$ thì suy ra $(i, j) \in E$, thì chúng ta nói rằng $G(V = V' \cap V'', E)$ là một đồ thị hai phần đầy đủ.

Nếu số phần tử của V' là m và số phần tử của V'' là n thì đồ thị có kí hiệu là $K_{m,n}$. Trong Hình 5.34 được hiển thị $K_{3.3}$.

Định nghĩa 5.49. *Sự co* lại của một đồ thị $G(V, E)$ được gọi là đồ thị $G'(V', E')$, nhận được từ G bằng cách loại bỏ tất cả các đỉnh bậc 2 và thay thế hai cạnh qua mỗi đỉnh như vậy bằng một cạnh.

Định nghĩa 5.50. Hai đồ thị G và G được gọi là *đẳng cấu* (viết $G' \approx G''$ nếu tồn tại biểu diễn ánh xạ lên của các đỉnh của G' trong các đỉnh của G'' bảo toàn lân cận.

Định lý 5.9 (Kuratovski)**.** *Một đồ thị là phẳng bởi vì không có đồ thị con nào của nó, mà sự co lại của nó là đẳng cấu thành K_5 hoặc $K_{3,3}$.*

Định lý 5.10 (cho bốn màu)**.** *Sắc số của đồ thị phẳng nhỏ hơn hoặc bằng 4.*

Định lý cuối cùng có nghĩa là chúng ta luôn có thể tô màu một bản đồ chính trị với bốn màu để không có các quốc gia láng giềng một màu. Nó được trình bày như một giả thuyết vào năm 1850. Nhiều nỗ lực không thành công sau đó đã được thực hiện để chứng minh điều đó, mãi đến năm 1976 K. Appel và W. Hacken mới thành công. Bằng chứng bao gồm phân tích 2.000 trường hợp khác nhau

và được thực hiện với hơn 1.000 giờ sử dụng máy tính. Ngày nay, có rất nhiều bằng chứng về nó, hầu hết trong số đó lại sử dụng các thuật toán và tính toán máy tính.

Ngoài ra còn có các thuật toán xác minh đa thức [Chiba, Nishizeki, Abe, Ozawa-1985], và trong trường hợp đồ thị phẳng, chúng cũng tìm biểu diễn phẳng tương ứng. Tuy nhiên, những vấn đề này bao gồm tài liệu đủ rộng và sẽ không được xem xét ở đây.

Bài tập

▷ **5.86.** Định lý nghịch đảo cho bốn màu có đúng không, tức là nếu sắc số của một đồ thị nhỏ hơn 5, thì nó có phải là đồ thị phẳng không?

5.9. Câu hỏi và bài tập

Trong phần cuối cùng, một số bài toán lý thuyết thú vị được đính kèm, cũng như các bài toán được đã cho tại các cuộc thi lập trình dành cho học sinh và trường học ở Bungary và quốc tế.

▷ **5.87.** *Bài toán cho Người đưa thư Trung Quốc, Duke programming contest, 1992, problem f.*

Bài toán này (từ Bài toán *Người đưa thư Trung Quốc*) được đặt theo tên của nhà toán học Trung Quốc Mei Guo Guang).

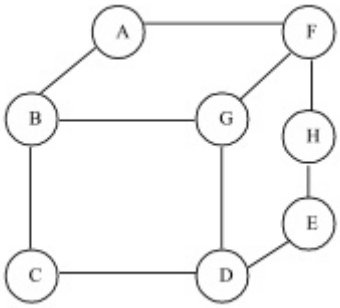
Tìm một vòng có độ dài nhỏ nhất trong một đồ thị có trọng số đi qua mỗi cạnh ít nhất một lần. Đồ thị không nhất thiết phải là Euler (Nếu là Euler thì chu trình Euler bất kỳ thỏa mãn điều kiện).

▷ **5.88.** *Duke programming contest, 1993, problem g.* Danh sách các quan hệ giữa các biến kiểu $i < j$ được đã cho. Viết chương trình in ra tất cả các cách sắp xếp có thể có của các biến tương thích với các quan hệ đã cho. Ví dụ, đối với các quan hệ $x < y$ và $x < z$, có hai cách sắp xếp: xyz và xzy.

▷ **5.89.** *Duke programming contest, 1993, problem h.* Các hình vuông được kết nối bởi các đường một chiều được đã cho. Viết chương trình tìm số đường nằm giữa hai hình vuông đã cho. Có thể có vô số con đường giữa hai hình vuông (ví dụ, nếu bạn đi qua một vòng). Trong trường hợp này, chương trình phải in -1.

▷ **5.90.** *ACM Regional contest, New Zealand, 1991, problem A.* Một đồ thị vô hướng $G(V, E)$ được đã cho. Nếu một phép thứ tự được cho trên các đỉnh của nó, thì B-bậc của đỉnh $v \in V$ được xác định như khoảng cách dài nhất giữa v và mọi đỉnh lân cận.

B-Bậc của đồ thị được gọi là cực đại của các B-bậc riêng lẻ cho mỗi đỉnh.



Hình 5.35. B-bậc của đồ thị

Ví dụ, đối với đồ thị của Hình 5.35 và đối với hai thứ tự mẫu (A, B, C, D, E, H, F, G) và (A, B, C, D, G, F, H, E) B-bậc của các đỉnh là: $6, 6, 1, 4, 1, 1, 6, 6$ cho thứ tự đầu tiên và $5, 3, 1, 4, 3, 5, 1, 4$ - cho thứ tự thứ hai, nó cung cấp B-bậc cho đồ thị là 6 ở thứ nhất , và 5 - trong trường hợp thứ hai.

Viết chương trình tìm thứ tự của các đỉnh mà B-bậc của đồ thị là cực tiểu.

▷ **5.91.** *chu trình đơn tối thiểu.*

Một đồ thị có trọng số không định hướng được đã cho. Tìm độ dài tối thiểu của một vòng lặp đơn giản trong một cột có ít nhất 3 đỉnh.

Lời giải: Thực tế là một đồ thị đã cho không cần thiết trong trường hợp này. Nếu có nhiều hơn một cạnh giữa hai đỉnh i và j, thì chúng ta chỉ để lại giá trị nhỏ nhất của chúng (trong một chu trình đơn, mỗi đỉnh có thể tham gia nhiều nhất một lần - vì vậy không thể có nhiều hơn một cạnh giữa hai đỉnh tham gia vào nó. chỉ cạnh nhỏ nhất, vì chúng ta muốn tìm chu trình nhỏ nhất). Sau đây, giải như sau:

Thuật toán 1. Với mỗi cạnh (i, j) của đồ thị ta thực hiện:

- chúng ta loại trừ (i, j) khỏi đồ thị;
- chúng ta tìm đường đi nhỏ nhất giữa i và j. Một chu trình có chiều dài L_{ij} thu được cùng với cạnh bị loại trừ.

L_{ij} tối thiểu là độ dài của chu trình tối thiểu cần thiết.

Độ phức tạp của thuật toán là $\Theta(m.n.log2n)$.

Thuật toán 2. Cho:

- $d(i, j)$ là độ dài của đường đi nhỏ nhất giữa các đỉnh i và j.
- $f(i, j)$ là trọng lượng của cạnh (i, j).

Khi đó độ dài của chu trình đơn nhỏ nhất là giá trị nhỏ nhất của tổng $d(i, k) + f(k, j) + d(j, i)$, với mỗi i, j và k mà đường đi nhỏ nhất giữa i và k không có điểm chung với đường đi nhỏ nhất giữa j và i.

Giải thích cách thu được độ phức tạp được chỉ ra của *thuật toán 1*. Tính độ phức tạp của *thuật toán 2*.

▷ **5.92.** *chu trình nhỏ nhất qua đỉnh k*

Đã cho một số tự nhiên k cố định. Đề xuất một thuật toán cho một đa đồ thị có trọng số không định hướng tùy ý $G(V, E)$ tìm thấy một chu trình đơn có độ dài nhỏ nhất trong G chứa ít nhất k đỉnh.

Rõ ràng, $k = n$ dẫn đến bài toán Người bán hàng, tuy nhiên bài toán này không là NP-đầy đủ (xem Chương 6), vì độ phức tạp của nó được xác định bởi n và k là cố định.

Độ phức tạp tốt nhất là đa thức n mà bạn có thể đạt được để giải quyết bài toán, với điều kiện k là hằng số tùy ý nhưng cố định?

▷ **5.93.** *Cơ sở số trong đồ thị*

Định nghĩa 5.51. Cho trước một đồ thị có hướng $G(V, E)$ và một hàm $v : V \to \mathbb{R}^+$, tập các trọng số của các đỉnh của đồ thị. *Css* của các đỉnh được gọi là tập $T \subseteq V$ sao cho:

- mỗi đỉnh của V có thể truy cập được bởi một số đỉnh của T
- tổng các trọng số của các đỉnh của T là nhỏ nhất.

Tìm cơ sở số của các đỉnh của đồ thị.

Có hai vấn đề thoạt nhìn không liên quan gì đến đồ thị, nhưng trong thực tế chúng ta dễ dàng giải quyết được nếu chúng ta trình bày chúng bằng đồ thị và tìm đường đi dài nhất trong đó (xem **??**).

▷ **5.94.** *Số lượng tối đa không giảm*

Tìm dãy số không giảm lớn nhất của một dãy số đã cho. Hay nói đúng hơn: Cho một dãy số nguyên $a_1, a_2, \ldots, a_n$. Tìm một dãy $a_{i1}, a_{i2}, \ldots,$ và $a_{ik}(i1 < i2 < \ldots < ik)$ sao cho $a_{ij} \leq a_{i(j+1)}$ với $j = 1, 2, \ldots, k-1$, và tổng $S = \sum_{i=1,2,\ldots k} a_{ij}$ là cực đại.

▷ **5.95.** *Chèn ba chiều*

Có n hộp được xác định bởi tọa độ các đỉnh của chúng (x_i, y_i, z_i). Tìm số hộp tối đa để chúng có thể xếp thành một hàng.

Gợi ý: Trong bài toán này, chúng ta có thể xây dựng một đồ thị trong đó các đỉnh sẽ đại diện cho các hộp và các cạnh được định hướng sẽ cho biết liệu một hộp có thể vừa với hộp khác hay không.

▷ **5.96.** *Đường dẫn nhỏ nhất trong đồ thi chu trình*

Có thể cải thiện độ phức tạp của thuật toán Dijkstra nếu chúng ta tìm kiếm một đường đi nhỏ nhất chỉ giữa hai đỉnh trong một đồ thị không?

▷ **5.97.** *Gốc ngược*

Một đồ thị có định hướng được đã cho. Hãy lập một thuật toán để kiểm tra xem có một đỉnh trong đồ thị với một bậc vào ở đầu vào $n-1$ và một bậc đầu ra là 0.

▷ **5.98.** *Kiểm tra xem biểu đồ có phải là cặp đôi không*

Một đồ thị có n đỉnh và m cạnh đã cho. Kiểm tra với độ phức tạp $\Theta(m+n)$ xem đồ thị đã cho có phải là cặp đôi hay không.

Gợi ý: Sử dụng một sửa đổi của chức năng duyệt theo chiều sâu.

▷ **5.99.** *Kiểm tra tính đẳng cấu của đồ thị*

Độ phức tạp tốt nhất mà bạn có thể đạt được để xác minh rằng hai đồ thị đã cho là đẳng cấu?

▷ **5.100.** *Bổ sung một đồ thị phẳng*

Cho trước một đồ thị phẳng $G(V, E)$ với $n(n \geq 11)$ đỉnh. Kiểm tra phần bù của G là một đồ thị phẳng.

Gợi ý: Để chứng tỏ rằng phần bù của G không thể là một đồ thị phẳng, bất kể G thuộc loại nào.

▷ **5.101.** *Tạp chí máy tính-5/1994*

Một đồ thị vô hướng được đã cho. Đối với một số đỉnh trong đồ thị, một số thực được cho trước - trọng số của đỉnh. Kiểm tra xem có thể so sánh các số thực trên các đỉnh khác hay không, sao cho trọng số của mỗi đỉnh bằng trung bình cộng của trọng số các đỉnh lân cận của nó.

Gợi ý: Biểu đồ được chia thành các thành phần kết nối. Tất cả các đỉnh trong một thành phần phải có cùng trọng số để đáp ứng điều kiện của nhiệm vụ.

▷ **5.102.** *d-nén*

Cho trước một ma trận vuông `A[][]` với kích thước $n \times n$, bao gồm các số không và đơn vị. Đã biết số lượng đơn vị trong mỗi hàng và mỗi cột của ma trận. Hãy khôi phục ma trận.

▷ **5.103.** *Olympic lập trình Balkan*

Hệ thống dầu là một bể chứa n ($2 \leq n \leq 200$) có thể tích từ 1 đến 100 và $n-1$ ống nối các bể trong một hệ thống kín (không có chu trình). Hệ thống được lấp đầy bởi một trong các bồn chứa S (nguồn), bồn này luôn trống và được sử dụng như một máy bơm. Trong một đơn vị thời gian, một đơn vị dầu đi qua mỗi ống nối với nguồn S. Quá trình tiếp tục cho đến khi tất cả các bể chứa đầy.

Đối với hệ thống bể chứa được kết nối, hãy tìm nguồn S tối ưu mà từ đó quá trình làm đầy yêu cầu tối thiểu thời gian.

Lời giải: Một đồ thị vô hướng xoay chiều liên thông $G(V, E)$ với các trọng số đỉnh được cho. Sau khi loại bỏ nguồn S, đồ thị được chia thành $k(2 \leq k \leq n-1)$ thành phần kết nối. Gọi tổng trọng số của các đỉnh trong thành phần thứ i của kết nối là T_i. Khi đó, nguồn S phải được chọn sao cho $T = \max\{T_i\}$, với $i = 1, 2, ..., k$ là cực tiểu.

Độ phức tạp tốt nhất có thể đạt được để giải quyết vấn đề sao cho bộ nhớ được sử dụng là $\Theta(n)$?

▷ **5.104.** *Tách các đỉnh (các cạnh)*

Một đồ thị vô hướng được cho trong đó hai đỉnh s và t được chọn. Tìm số phần tử của tập đỉnh (cạnh) H nhỏ nhất của đồ thị sao cho nếu các đỉnh (cạnh) tham gia H bị loại khỏi đồ thị thì s và t vẫn nằm trong các thành phần liên thông khác nhau.

▷ **5.105.** *Dòng chảy trong một đồ thị với tính thấm của các cạnh của số thực*

Bài toán tìm lưu lượng cực đại trong đồ thị sẽ thay đổi như thế nào nếu trọng số đã cho (độ cho qua được) của các cạnh là số thực?

Gợi ý: Độ phức tạp của thuật toán trong trường hợp này sẽ phụ thuộc vào trọng số của các cạnh và kích thước của luồng. Một thuật toán có độ phức tạp chỉ phụ thuộc vào số đỉnh n của đồ thị không được biết đến.

▷ **5.106.** *Bổ sung cho một đồ thị được liên thông chặt*

Một đồ thị được liên thông yếu có định hướng được đã cho. Thêm số lượng cạnh tối thiểu vào đồ thị để nó trở nên liên thông chặt.

▷ **5.107.** 2^n

Một số nguyên n được cho trước. Tìm một chuỗi có độ dài nhỏ nhất sao cho các biểu diễn nhị phân của tất cả các số từ 1 đến n tham gia như các tập con.

Gợi ý: Vấn đề được giải quyết bằng cách tìm chu trình Euler. Ngoài ra còn có một số thuật toán đơn giản hơn, tính đúng đắn của nó được chứng minh bằng cách xem xét các chu trình Euler.

▷ **5.108.** *Đường kính của đồ thị*

Viết chương trình tìm đường kính của đồ thị.

Bạn có thể soạn một thuật toán có độ phức tạp nhỏ hơn $\Theta(n^3)$ không?

▷ **5.109.** *Phương pháp sóng*

Ma trận vuông với các phần tử 0 và X và tọa độ của hai ô từ nó được cho: ban đầu (s_x, s_y) và cuối cùng (t_x, t_y). Để tìm cách giữa chúng, tức là. một chuỗi các ô sao cho:

- Đường dẫn chỉ nên bao gồm các ô có giá trị 0.
- Mỗi hai ô liên tiếp trong đường dẫn phải là "liền kề", tức là. khác nhau chính xác một trong hai tọa độ của chúng.
- Tổng chiều dài của con đường phải nhỏ nhất.

Giải pháp: Vấn đề được giải quyết một cách hiệu quả với cái gọi là. phương pháp sóng (diễn giải cụ thể của kỹ thuật thu thập thông tin theo chiều rộng). Phương pháp này bao gồm những điều sau:

1) Tại vị trí ban đầu ta viết số 1. Ta đặt $i = 1$.

2) Chúng ta tìm tất cả các ô trong đó số i được viết. Trong tất cả các hàng xóm tự do của chúng (tức là các ô có giá trị 0), chúng ta viết số $i + 1$.

3) Nếu một trong các lân cận từ bước 2 trùng với vị trí cuối cùng, thuật toán kết thúc (tìm được đường đi nhỏ nhất và độ dài của nó là $i + 1$). Nếu không, chúng ta tăng i và lặp lại bước 2).

X	X	X	X	X	X	X	X
X	**1**	**2**	**3**	**4**	**5**	**6**	X
X	**2**	X	X	X	**6**	**7**	X
X	**3**	**4**	**5**	X	**8**	X	X
X	**4**	X	**6**	**7**	**8**	X	X
X	**5**	**6**	**7**	X	X	X	X
X	**6**	**7**	**8**	**9**	X	X	X
X	X	X	X	X	X	X	X

Hình 5.36. Phương pháp sóng.

Phương pháp này được gọi là phương pháp sóng, bởi vì việc tìm kiếm đường đi với tiến trình nhất quán theo mọi hướng có thể tương tự như cách nước được phân phối trong một không gian kín, với vị trí bắt đầu của nguồn. Hiệu quả của việc thực hiện thuật toán trên được xác định bởi cách tìm kiếm các láng giềng trong mỗi bước.