CHƯƠNG 5

CÂY VÀ CÂY KHUNG CỦA ĐỒ THỊ

(tuần 7,8,9: Tổng cộng có 4 tiết lý thuyết,6 tiết hướng dẫn bài tập/thực hành,2 tiết kiểm tra lý thuyết lần 1)

Đồ thị vô hướng liên thông không có chu trình gọi là cây. Khái niệm cây lần đầu tiên được Cayley đưa ra vào năm 1857, khi ông sử dụng chúng để đếm một dạng cấu trúc phân tử của các hợp chất hoá học trong hoá học hữu cơ. Cây còn được sử dụng rộng rãi trong rất nhiều lĩnh vực khác nhau, đặc biệt trong tin học, cây được sử dụng để xây dựng các thuật toán tổ chức các thư mục, các thuật toán cất giữ, truyền dữ liệu và tìm kiếm…

5.1. CÂY VÀ CÁC TÍNH CHẤT CƠ BẢN CỦA CÂY

Định nghĩa 1.

Ta gọi cây là đồ thị vô hướng liên thông không có chu trình. Đồ thị không có chu trình được gọi là rừng.

Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Có thể nói cây là đồ thị vô hướng đơn giản nhất. Định lý sau đây cho ta một số tính chất của cây.

Định lý 1. Giả sử G=(V,E) là đồ thị vô hướng n đỉnh. Khi đó các mệnh đề sau đây là tương đương:

(1) T là cây;

(2) T không chứa chu trình và có n-1 cạnh;

(3) T liên thông và có n-1 cạnh;

(4) T liên thông và mỗi cạnh của nó đều là cầu;

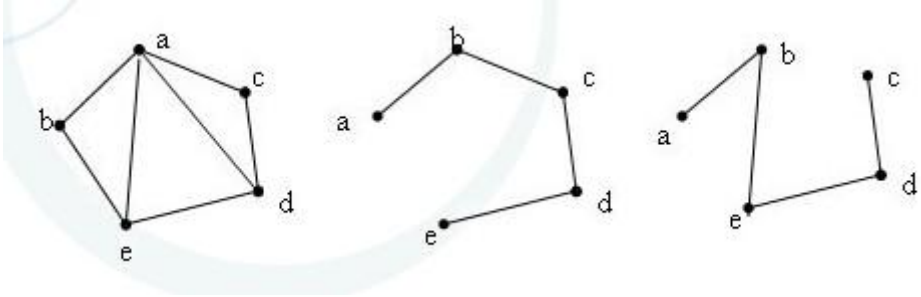
(5) Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;

(6) T không chứa chu trình nhưng hễ cứ thêm vào một cạnh ta thu được đúng một chu trình.

5.2. CÂY KHUNG CỦA ĐỒ THỊ

Định nghĩa 2.

Đồ thị G và cây khung của nó được cho trong hình 1



Hình 1. Đồ thị và các cây khung của nó

Định lý sau đây cho biết số lượng cây khung của đồ thị đầy đủ K_n:

Định lý 2 (Cayley). *Số lượng cây khung của đồ thị K_n là n^{n-2}.*

Áp dụng các thuật toán tìm kiếm theo chiều sâu và theo chiều rộng trên đồ thị để xây dựng cây khung của đồ thị vô hướng liên thông. Trong cả hai trường hợp mỗi khi ta đến được đỉnh mới u (tức Chuaxet[u]=true) từ đỉnh v thì cạnh (v, u) sẽ được kết nạp vào cây khung. Hai thuật toán tương ứng được trình bày trong hai thủ tục sau đây.

void stree_DFS(v)

(tim kiem theo chieu sau ap dung vao tim tap canh cua cay khung T cua do thi vo huong lien thong G cho boi danh sach ke. Cac bien Chuaxet, Ke, T la toan cuc*)*

```
{
        Chuaxet[v]=0;
        For u ∈ Ke(v)
        if Chuaxet[u]
        {
                T=T ∪(u,v);
                STREE_DFS(u);
        }
}
(* Main Program *)
{
        (* Initialition *)
        for u∈ V Chuaxet[u]=1;
        T = ∅; (* T la tap canh cua cay khung *)
        STREE_DFS(root); ( root la dinh nao do cua do thi *)
}
```

void Stree_BFS(v)

```
(* tim kiem theo chieu rong ap dung tim tap canh cua cau khung T cua do thi vo
huong lien thong G cho boi danh sach Ke *)
{
        Queue=∅;
        Queue ⇐r;
        Chuaxet[r]=0;
```

```
        While queue <> ∅
        {
                V ⇐ queue;
                For r ∈ Ke(v)
                If Chuaxet[u]
                {
                        Queue ⇐ u;
                        Chuaxet[u]=0;
                        T= T ∪ (u,v);
                }
        }
}
(* Main Program *);
{
        for u ∈ V do Chuaxet[u]=1;
        T = ∅; (* T la tap canh cua cay khung *)
        Stree_BFS(root); (* root la mot dinh tuy y cua do thi *)
}
```

Chú ý:

1. Lập luận tương tự như trong phần trước có thể chỉ ra được rằng các thuật toán mô tả ở trên có độ phức tạp tính toán O(m+n).

2. Cây khung tìm được theo thủ tục Stree_BFS() là cây đường đi ngắn nhất từ gốc r đến tất cả các đỉnh còn lại của đồ thị.

5.3. BÀI TOÁN CÂY KHUNG NHỎ NHẤT

Bài toán cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Trong mục này chúng ta trình bày những thuật toán cơ bản để giải bài toán nào. Trước hết chúng ta phát biểu nội dung bài toán.

Cho G=(V,E) là đồ thị vô hướng liên thông với tập đỉnh V={1, 2, . . . ,n} và tập cạnh E gồm m cạnh. Mỗi cạnh E của đồ thị G được gán với một số không âm c(e), gọi là độ dài của nó. Giả sử H=(V,T) là cây khung của đồ thị G. Ta gọi độ dài c(H) của cây khung H là tổng độ dài các cạnh của nó:

$$C(H) = \sum_{e \in T} c(e).$$

Bài toán đặt ra là trong tất cả cây khung của đồ thị G hãy tìm cây khung với độ dài nhỏ nhất. Cây khung như vậy như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán cây khung nhỏ nhất.

Để minh hoạ cho những ứng dụng bài toán cây khung nhỏ nhất, dưới đây, ta phát biểu hai mô hình thực tế tiêu biểu của nó.

Bài toán xây dựng hệ thống đường sắt. Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất kỳ một thành phố nào đến bất kỳ một trong các thành phố còn lại. Mặt khác trên quan điểm kinh tế đòi hỏi là chi phí xây dựng hệ thống đường phải nhỏ nhất. Rõ ràng đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng với phương án xây dựng tối ưu phải là cây. Vì vây, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố, với độ dài trên các các cạnh chính là chi phí xây dựng đường ray nối hai thành phố tương ứng (chú ý là trong bài toán này ta giả thiết là không xây dựng tuyến đường sắt có các nhà ga phân tuyến nằm ngoài các thành phố).

Bài toán nối mạng máy tính. Cần nối mạng một hệ thống gồm n máy tính đánh số từ 1 đến n. Biết chi phí nối máy i với máy j là c[i,j], i,j = 1, 2, . . . ,n (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí nối mạng là nhỏ nhất.

Để giải bài toán cây khung nhỏ nhất, tất nhiên có thể liệt kê tất cả các cây khung của đồ thị và chọn trong số cây khung ấy cây khung nhỏ nhất. Phương pháp như vậy, trong trường hợp đồ thị đầy đủ, sẽ đòi hỏi thời gian cỡ nn^{-2} , và rõ ràng không thể thực hiện được ngay cả với những đồ thị với số đỉnh cỡ hàng chục. Rất may là đối với bài toán cây khung nhỏ nhất chúng ta đã có những thuật toán rất hiệu quả để giải chúng. Chúng ta xét hai trong số những thuật toán như vậy: Thuật toán Kruskal và Thuật toán Prim.

5.3.1.Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh T của cây khung nhỏ nhất H=(V,T) theo từng bước. Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của độ dài. Bắt đầu từ tập T=∅ , ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập T gồm n-1 cạnh. Cụ thể, thuật toán có thể mô tả như sau:

```
void Kruskal()
{
        T=∅;
        While (|T| < n-1 && (E<>∅)
```

{

E=E\{e};

if (T∪{e} không chứa chu trình T= T∪{e} ;

}

if (|T| < n-1) Đồ thị không liên thông;

}

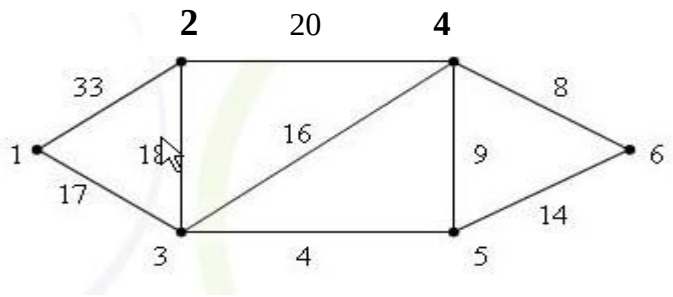
Thí dụ 3.Tìm cây khung nhỏ nhất của đồ thị cho trong hình 3 dưới.

Bước khởi tạo. Đặt T=∅ . Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài ta có dãy:

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

dãy độ dài tương ứng của chúng

4, 8, 9, 14, 16, 17, 18, 20, 23.



Hình 2. Đồ thị và cây khung nhỏ nhất

Ở ba lần gặp đầu tiên ta lần lượt bổ sung vào tập T các cạnh (3,5) , (4,6) , (4,5). Rõ ràng nếu thêm cạnh (5,6) vào T thì sẽ tạo thành 2 cạnh (4,5), (4,6) đã có trong T chu trình. Tình huống tương tự cũng xảy ra đối với cạnh (3,4) là cạnh tiếp theo của dãy. Tiếp theo ta bổ sung cạnh (1,3), (2,3) vào T và thu được tập T gồm 5 cạnh:

T = {(3,5) , (4,6) , (4,5) , (1,3) , (2,3) }

Chính là tập cạnh của cây khung nhỏ nhất cần tìm.

Về việc lập trình thực hiện thuật toán.

Khối lượng tính toán nhiều nhất của thuật toán chính là ở bước sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài để lựa chọn cạnh bổ sung. Đối với đồ thị m cạnh cần phải thực hiện m log m phép toán để sắp xếp các cạnh của đồ thị thành dãy không giảm theo độ dài. Tuy nhiên, để xây dựng cây khung nhỏ nhất với n-1 cạnh, nói chung ta không cần phải sắp thứ tự toàn bộ các cạnh mà chỉ cần xét phần trên của dãy đó chứa r < m cạnh. Để làm việc đó ta có thể sử dụng các thủ tục sắp xếp dạng Vun đống (Heap Sort). Trong thủ tục này, để tạo đống đầu tiên ta mất cỡ O(m) phép toán, mỗi phần tử tiếp theo trong

đống có thể tìm sau thời gian O(log m). Vì vậy, với cải tiến này thuật toán sẽ mất thời gian cỡ O(m+p) log m) cho việc sắp xếp các cạnh. Trong thực tế tính toán số p nhỏ hơn rất nhiều so với m.

Vấn đề thứ hai trong việc thể hiện thuật toán Kruskal là việc lựa chọn cạnh để bổ sung đòi hỏi phải có một thủ tục hiệu quả kiểm tra tập cạnh T ∪ { e} có chứa chu trình hay không. Để ý rằng, các cạnh trong T ở các bước lặp trung gian sẽ tạo thành một rừng. Cạnh e cần khảo sát sẽ tạo thành chu trình với các cạnh trong T khi và chỉ khi cả hai đỉnh đầu của nó thuộc vào cùng một cây con của rừng nói trên. Do đó, nếu cạnh e không tạo thành chu trình với các cạnh trong T, thì nó phải nối hai cây khác nhau trong T. vì thế, để kiểm tra xem có thể bổ sung cạnh e vào T ta chỉ cần kiểm tra xem nó có nối hai cây khác nhau trong T hay không. Một trong các phương pháp hiệu quả để thực hiện việc kiểm tra này là ta sẽ phân hoạch tập các đỉnh của đồ thị ra thành các tập con không giao nhau, mỗi tập xác định bởi một cây con trong T(được hình thành ở các bước do việc bổ sung cạnh vào T). chẳng hạn, đối với đồ thị trong ví dụ 3, đầu tiên ta có sáu tập con 1 phần tử: { 1} , { 2} , { 3} , { 4} , { 5} , { 6} . Sau khi bổ sung cạnh (3,5), ta có các tập con { 1} , { 2} , { 3,5} , { 4} , { 6} . Ở bước thứ 3, ta chọn cạnh (4,5), khi đó hai tập con được nối lại và danh sách các tập con là { 1} , { 2} , { 3,4,5,6} . Cạnh có độ dài tiếp theo là (4,6), do hai đầu của nó thuộc vào cùng một tập con { 3,4,5,6} , nên nó sẽ tạo thành chu trình trong tập này. Vì vậy cạnh này không được chọn. Và thuật toán sẽ tiếp tục chọn cạnh tiếp theo để khảo sát …

Như vậy, để giải quyết vấn đề thứ hai này ta phải xây dựng hai thủ tục: Kiểm tra xem hai đầu u, v của cạnh e=(u,v) có thuộc vào hai tập con khác nhau hay không, và trong trường hợp câu trả lời là khẳng định, nối hai tập con tương ứng thành một tập. Chú ý rằng mỗi tập con trong phân hoạch có thể lưu trữ như là một cây có gốc, và khi đó mỗi gốc sẽ được sử dụng làm nhãn nhận biết tập con tương ứng.

5.3.2. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả với những đồ thị dày (đồ thị với số cạnh m ≈ n(n-1)/2). Trong trường hợp đó thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất. Trong phương pháp này bắt đầu từ một đỉnh tuỳ ý của đồ thị, đầu tiên ta nối s với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh y. Nghĩa là trong số các cạnh kề của đỉnh s, cạnh (s,y) có độ dài nhỏ nhất. Tiếp theo trong số các cạnh kề với hai đỉnh s hoặc y ta tìm cạnh có độ dài nhỏ nhất, cạnh này dẫn đến đỉnh thứ ba z, và ta thu được cây bộ phận gồm 3 đỉnh và 2 cạnh. Quá trình này sẽ tiếp tục cho đến khi ta thu được cây gồm n đỉnh và n-1 cạnh sẽ chính là cây khung nhỏ nhất cần tìm.

Giả sử đồ thị cho bởi ma trận trọng số C = {c[i,j], i, j= 1, 2, . . . , n} . trong quá trình thực hiện thuật toán, ở mỗi bước để có thể nhanh chóng chọn đỉnh và cạnh cần bổ sung vào cây khung, các đỉnh của đồ thị sẽ được gán cho các nhãn. Nhãn của một đỉnh v sẽ gồm hai phần và có dạng [d[v], near[v]], trong đó d[v] dùng để ghi nhận độ dài của cạnh có độ dài nhỏ nhất trong số các cạnh nối với đỉnh v với các đỉnh của cây khung đang xây dựng (ta sẽ gọi là khoảng cách từ đỉnh v đến tập đỉnh của cây khung), nói một cách chính xác

d[v]:= min { c[v,w] : w $\in$ V_H } (= c[v,z]),

còn near[v] ghi nhận đỉnh của cây khung gần v nhất (near[v]:=z).

Thuật toán Prim được mô tả đầy đủ trong thủ tục sau:

```
void Prim()
{
        (* buoc khoi tao *)
        chon s la mot dinh nao do cua do thi;
        VH={ s } ; T=∅ ; d[s]=0; near[s]=s.
        For v∈ V\VH
        {
        D[v]=c[s,v];
        near[v]=s;
        }
        (* buoc lap *)
        stop=0;
        while (!stop)
        {
            tim u ∈ V\VH thoa man:
                d[u]=min { d[v]: u∈ V\VH } ;
            VH= VH ∪ { u } ; T = T ∪ { (u, near[u]) } ;
            If (| VH | ==n )
```

```
        {
            H=( VH,T) la cay khung nho nhat cua do thi;
            Stop=1;
        }
    Else
        For v∈ V\ VH
            If d[v]>c[u,v]
            {
                d[v]=c[u,v];
                near[v]=u;
            }
    };
}
```

Thí dụ 4. Tìm cây khung nhỏ nhất cho đồ thị xét trong ví dụ 3 theo thuật toán Prim. Ma trận trọng số của đồ thị có dạng

		1	2	3	4	5	6
	1	0	33	17	∞	∞	∞
	2	33	0	18	20	∞	∞
C =	3	17	18	0	16	4	∞
	4	∞	20	16	0	9	8
	5	∞	∞	4	9	0	14
	6	∞	∞	∞	8	14	0

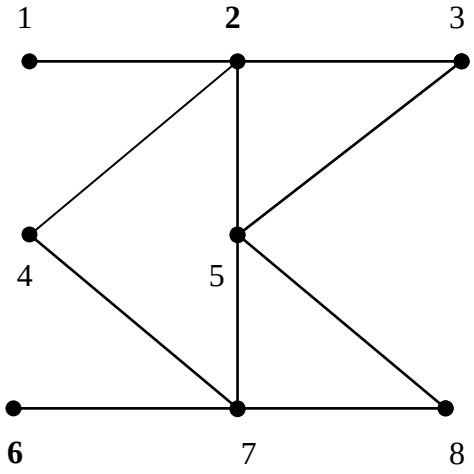
Bảng dưới đây ghi nhãn của các đỉnh trong các bước lặp của thuật toán, đỉnh đánh dấu * là đỉnh được chọn để bổ sung vào cây khung (khi đó nhãn của nó không còn bị biến đổi trong các bước lặp tiếp theo, vì vậy ta đánh dấu – để ghi nhận điều đó):

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	VH Tập đỉnh cần tìm	T Tập cạnh cần tìm
Khởi tạo	[0,1]	[33,1]	[17,1]*	[∞ ,1]	[∞ ,1]	[∞ ,1]	1	∅
1	-	[18,3]	-	[16,3]	[4,3]*	[∞ ,1]	1,3	(3,1)
2	-	[18,3]	-	[9,5]*	-	[14,5]	1,3,5	(3,1), (5,3)
3	-	[18,3]	-	-	-	[8,4]*	1,3,5,4	(3,1), (5,3), (4,5)
4	-	[18,3]*	-	-	-	-	1,3,5,4,6	(3,1), (5,3), (4,5), (6,4)
5	-	-	-	-	-	-	1,3,5,4,6,2	(3,1), (5,3), (4,5), (6,4), (2,3

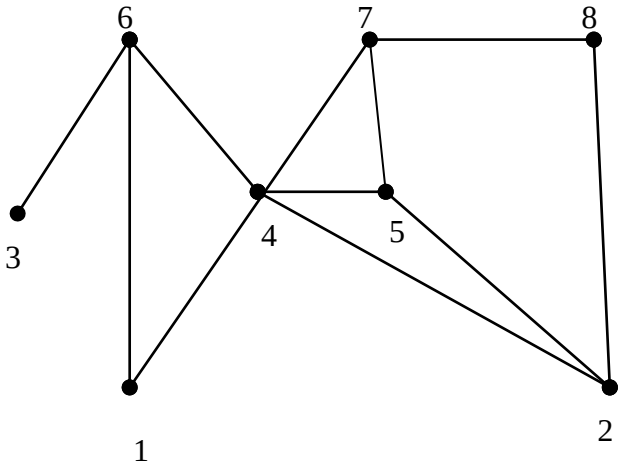
Bài tập lý thuyết

5-1. Giả sử đồ thị G liên thông, có 13 đỉnh và 20 cạnh. Hỏi cây khung của G có bao nhiêu đỉnh ? có bao nhiêu cạnh ?

5-2.Tìm cây khung của đồ thị (hình 1,2) sau theo phương pháp DFS, BFS (chọn đỉnh 1 làm gốc)

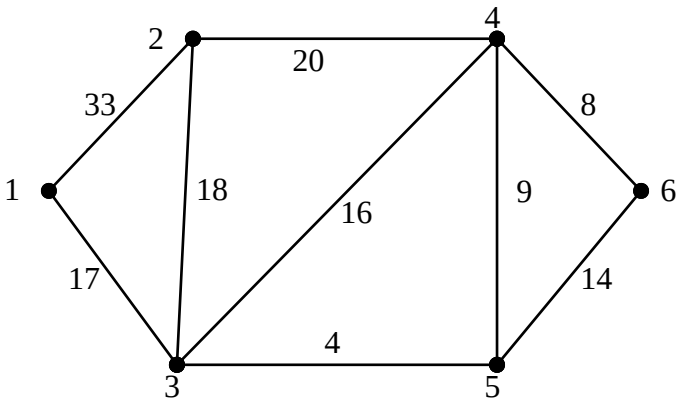


Hình 1

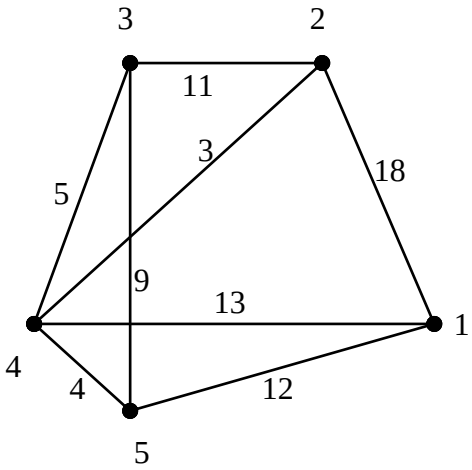


Hình 2

5-3.Tìm cây khung bé nhất của các đồ thị sau (hình 3,4) theo phương pháp KrusKal, Prim

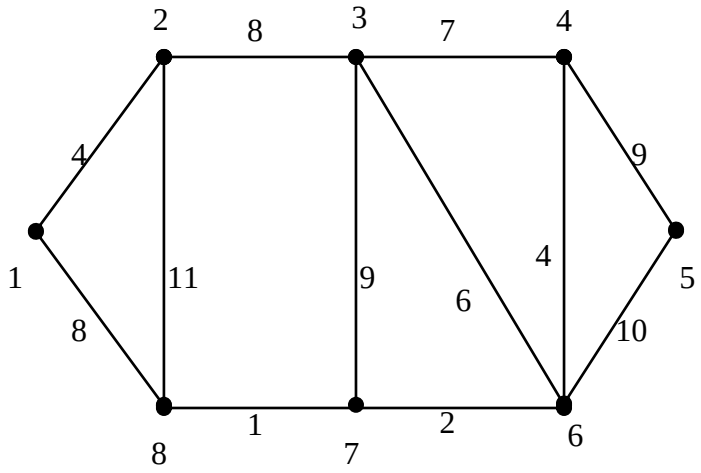


Hình 3

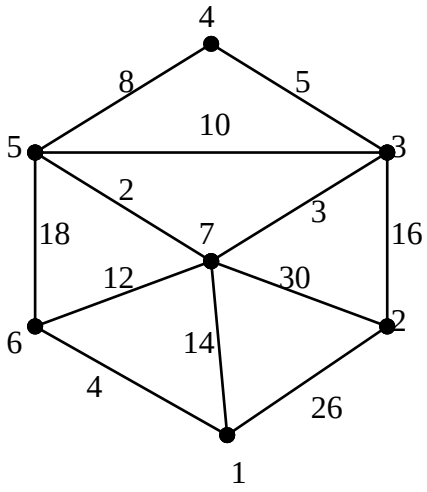


Hình 4

5-4.Tìm cây khung bé nhất của các đồ thị sau (hình 5,6) theo phương pháp KrusKal, Prim

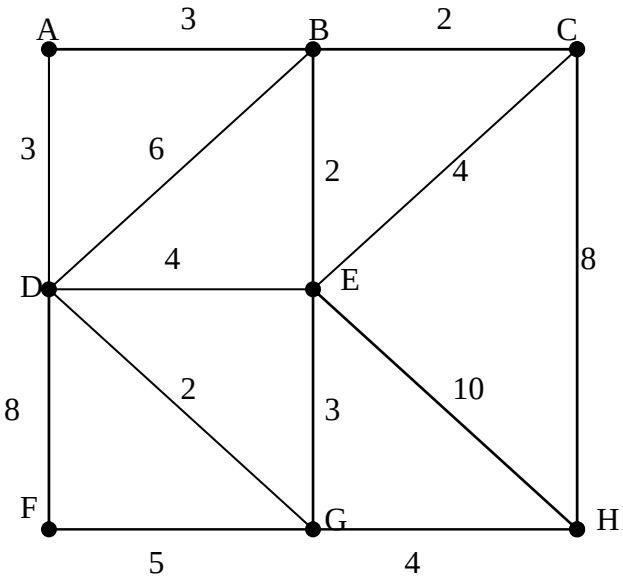


Hình 5



Hình 6

5-5.Tìm cây khung bé nhất của các đồ thị sau (hình 7) theo các phương pháp KrusKal,Prim



Hình 7

Bài tập thực hành

5-6. Mạng an toàn

Cho một mạng N (N <= 20) máy tính được đánh số từ 1 đến N. Sơ đồ mạng được cho bởi hệ gồm M kênh (đoạn) nối trực tiếp giữa một số cặp máy trong mạng, m kênh tương ứng với m cặp. Cho biết chi phí truyền 1 đơn vị thông tin theo mỗi kênh của mạng.

Người ta cần chuyển một bức thông điệp từ máy s đến máy t. Để đảm bảo an toàn, người ta chuyển bức thông điện này theo hai đường truyền tin khác nhau (tức không có kênh nào) của mạng được sử dụng trong cả hai đường truyền tin; cho phép hai đường truyền tin cùng đi qua một số máy tính). Chi phí của một đường truyền được hiểu là tổng chi phí trên các kênh của nó. Đơn giá đường truyền từ máy s sang máy t được tính như sau:

Với hai máy s và t, cùng bức thông điệp có độ dài là 1 đơn vị thông tin, đơn giá truyền cho cặp (s, t) được tính bằng tổng chi phí chuyển thông điệp an toàn (bằng tổng chi phí của hai đường truyền tin) là nhỏ nhất.

Người ta mong muốn mạng máy tính (mạng truyền tin nói trên thỏa mãn tính chất an toàn theo nghĩa là từ một máy bất kỳ luôn truyền được (một cách an toàn) thông điệp tới một máy bất kỳ khác. Khi một mạng an toàn, người ta tính được đơn giá của mạng là tổng đơn giá mọi đường truyền từ một máy bất kỳ tới một máy bất kỳ khác.

Ma trận đơn giá của mạng là mảng hai chiều A có N dòng và N cột, mà giá trị phần tử A[i, j] chính là đơn giá từ máy i sang máy j.

Câu 1: Cho trước một mạng, hãy kiểm ra tính an toàn của mạng đó.

Câu 2: Khi mạng không an toàn được phép bổ sung một số kênh truyền để nó trở thành an toàn. Đơn giá mỗi kênh truyền bổ sung theo được coi bằng hai lần giá trị cực đại đơn giá các kênh đã có. Mọi kênh bổ sung được coi có đơn giá như nhau. Hãy tìm cách bổ sung các kênh mới mà đơn giá mạng là nhỏ nhất.

Câu 3: Khi mạng an toàn hoặc sau khi bổ sung kênh để mạng an toàn, hãy in ra đơn giá mạng và ma trận đơn giá.

Dữ liệu vào: cho trong file INP.B2 với cấu trúc như sau:

Dòng đầu tiên ghi 2 số n, m cách nhau bởi dấu cách.

Mỗi dòng thứ i trong số m dòng tiếp theo ghi thông tin về kênh nối thứ i của mạng gồm 3 số d[i], c[i], g[i] trong đó d[i], c[i] là chỉ số của hai máy tương ứng với kênh này và g[i] (nguyên dương) là chi phí để truyền một đơn vị thông tin từ máy d[i] đến máy c[i] theo kênh này. Các giá trị g[i] cho trước không vượt quá 40 (và như vậy đơn giá các kênh bổ sung không vượt quá 80).

Kết quả: ghi ra file OUT.B2 theo qui cách sau:

Dòng đầu tiên ghi 1 số nguyên p thể hiện mạng có an toàn hay không và p có ý nghĩa là số lượng kênh cần bổ sung. p=0 có nghĩa mạng an toàn; p>0 có nghĩa mạng không an toàn và cần bổ sung p kênh nữa để mạng an toàn với chi phí bổ sung ít nhất.
p dòng tiếp theo ghi p kênh bổ sung. Cách ghi như trong file dữ liệu vào.
Dòng tiếp theo ghi đơn giá của mạng an toàn.
N dòng tiếp theo ghi ma trận đơn giá của mạng an toàn: mỗi hàng của ma trận ghi trên một dòng.

5-7.Xây dựng đường ống nước

Có 1 trạm cấp nước và N điểm dân cư. Hãy xây dựng chương trình thiết kế tuyến đường ống nước cung cấp đến mọi nhà sao cho tổng chiều dài đường ống phải dùng là ít nhất. Giả sử rằng các đường ống chỉ được nối giữa 2 điểm dân cư hoặc giữa trạm cấp nước với điểm dân cư.

Cài đặt một số thuật toán căn bản quan trọng

Tìm cây khung dựa vào DFS

```
1. #include <conio.h>
2. #include <iostream.h>
3. #include <stdio.h>
4. int daxet[100];
5. int a[100][100];
6. int dau[100],cuoi[100];
7. int socanh=0;
8. int n;
9. void dfs(int v)
10. {
11. daxet[v]=1;
12. for (int u=1;u<=n;u++)
13. if (a[v][u]!=0 && !daxet[u])
14. {
15. dau[++socanh]=v;
16. cuoi[socanh]=u;
17. dfs(u);
18. }
19. }
20. void readfile()
21. {
22. FILE *f;
23. clrscr();
24. f=fopen("d:\\dothi\\tree.inp","rt");// hinh2.inp
25. fscanf(f,"%d",&n);
26. for (int v=1;v<=n;v++)
27. for (int u=1;u<=n;u++)
28. fscanf(f,"%d",&a[u][v]);
29. fclose(f);
30. }
31. void find()
32. {
```

```
33. for (int v=1;v<=n;v++)
34. daxet[v]=0;
35. for (v=1;v<=n;v++)
36. if (!daxet[v])
37. dfs(v);
38. }
39. void treedfs()
40. {   cout<<"cay khung cua do thi:"<<endl;
41. for (int i=1; i<=socanh;i++)
42. cout<<"("<<dau[i]<<","<<cuoi[i]<<")"<<endl;
43. }
44. void main()
45. {
46. readfile();
47. find();
48. treedfs();
49. }
```

Tree.inp

```
8
0 0 0 1 0 1 0 0
0 0 0 1 1 0 0 1
0 0 0 0 0 1 0 0
1 1 0 0 1 1 1 0
0 1 0 1 0 0 1 0
1 0 1 1 0 0 0 0
0 0 0 1 1 0 0 1
0 1 0 0 0 0 1 0
```

Tìm cây khung dựa vào BFS

```
1. #include <conio.h>
2. #include <iostream.h>
3. #include <stdio.h>
4. int daxet[100];
5. int a[100][100];
6. int Queue[100];
7. int dau[100],cuoi[100];
8. int socanh=0;
9. int n;
10. void BFS(int u)
11. {
12. int w,v;
13. int dauQ,cuoiQ;
14. dauQ=1;cuoiQ=1;
15. Queue[dauQ]=u;
16. daxet[u]=1;
17. while (dauQ<=cuoiQ)
18. {
19. v=Queue[dauQ++];
20. for (w=1;w<=n;w++)
21. if (a[v][w]==1 && !daxet[w])
22. {
23. Queue[++cuoiQ]=w;
24. daxet[w]=1;
25. dau[++socanh]=v;
26. cuoi[socanh]=w;
27. }
28. }
29. }
30. void find()
31. {
32. for (int v=1;v<=n;v++)
33. daxet[v]=0;
```

```
34. for (v=1;v<=n;v++)
35. if (!daxet[v])
36. BFS(v);
37. }
38. void readfile()
39. {
40. FILE *f;
41. clrscr();
42. f=fopen("d:\\dothi\\tree.inp","rt");
43. fscanf(f,"%d",&n);
44. for (int v=1;v<=n;v++)
45. for (int u=1;u<=n;u++)
46. fscanf(f,"%d",&a[u][v]);
47. fclose(f);
48. }
49. void treebfs()
50. {
51. {   cout<<"cay khung cua do thi:"<<endl;
52. for (int i=1; i<=socanh;i++)
53. cout<<"("<<dau[i]<<","<<cuoi[i]<<")"<<endl;
54. }
55. }
56. void main()
57. {
58. readfile();
59. find();
60. treebfs();
61. }
```

Tree.inp

```
8
0 0 0 1 0 1 0 0
0 0 0 1 1 0 0 1
0 0 0 0 0 1 0 0
```

1 1 0 0 1 1 1 0

0 1 0 1 0 0 1 0

1 0 1 1 0 0 0 0

0 0 0 1 1 0 0 1

0 1 0 0 0 0 1 0

THUẬT TOÁN KRUSKAL

Dữ liệu vào:

Được lưu trên file kruskal.inp có cấu trúc như sau:

Dòng đầu ghi 2 số nguyên dương n,m lần lượt là số đỉnh và số cạnh của đồ thị.

Trong m dòng tiếp theo mỗi dòng ghi 3 số lần lượt là đỉnh đầu, đỉnh cuối và trọng số của cạnh tương ứng.

Dữ liệu ra:

Xuất lên màn hình các thông tin:

Các cạnh tạo thành cây khung bé nhất và tổng độ dài của cây khung bé nhất này.

Ví dụ

6 9

2 3 18

2 4 20

1 2 33

3 5 4

4 6 8

4 5 9

5 6 14

3 4 16

1 3 17

Phần khai báo các biến

Khai báo kiểu dữ liệu có tên là canh để lưu trữ thông tin các cạnh của đồ thị, mỗi cạnh của đồ thị được biểu diễn bởi đỉnh đầu (dau), đỉnh cuối (cuoi) và trọng số (w). n là số đỉnh của đồ thị và m là số cạnh của đồ thị. c là một mảng 1 chiều kiểu canh. connect là mảng 2 chiều để cho biết hai đỉnh nào đó trong cây khung tìm được có liên thông nhau hay không ? connect[i][j] =1 nếu i và j là liên thông và ngược lại

```
.const max=100; // hoac viet #define max 100
    struct canh
    {       int dau,cuoi;
            int w;
    };
    int     n,m,connect[max][max];
    canh    c[max];
```

Chúng ta cần thực hiện một số hàm sau:

1.Hàm đọc file

2.Hàm sắp xếp các cạnh theo chiều tăng dần

3.Hàm cập nhật lại các đỉnh liên thông của cây khung tìm được

Tại mỗi thời điểm ta xét cạnh (u,v) trong dãy các cạnh đã được sắp, nếu hai đỉnh u,v là không liên thông - nghĩa là connect[u][v]=0 thi cạnh (u,v) này sẽ được đưa vào cây khung kết quả và ta cần cập nhật lại sự liên thông của các đỉnh đã có trong cây khung đang tìm thấy đến thời điểm đó bằng cách cho connect[i][j]=1 và connect[j][i]=1 và chú ý là khi đưa một cạnh (u,v) mới vào cây khung thì ta phải cập nhật để tất cả các đỉnh đã có trong cây khung cùng với hai đinh u,v là liên thông.

4.Hàm KrusKal

Nếu tìm thấy một cạnh (trong dãy cạnh được sắp) có 2 đỉnh không liên thông thì ta làm một số bước như sau:

Tăng số cạnh của cây khung lên 1

Cập nhật lại sự liên thông của các đỉnh trong cây khung đang tìm được

Thêm trọng số cạnh đang xét vào độ dài của cây khung.

Toàn văn của chương trình này như sau:

```
1.  #include<conio.h>
2.  #include<stdio.h>
3.  #include<iostream.h>
4.  const max=100; // hoac viet #define max 100
5.  struct canh
6.  {   int dau,cuoi;
7.  int w;
8.  };
9.  int  n,m,connect[max][max];
10. canh        c[max];
11. void readfile()
12. {
13. FILE        *f;
14. f=fopen("d:\\dothi\\kruskal.inp","rt");
15. fscanf(f,"%d%d",&n,&m);
16. for (int i=1;i<=m;i++)
17. fscanf(f,"%d%d%d",&c[i].dau,&c[i].cuoi,&c[i].w);
```

```
18. fclose(f);
19. for (i=1;i<=n;i++)
20. for (int j=1;j<=n;j++)
21. connect[i][j]=0;
22. for (i=1;i<=n;i++)
23. connect[i][i]= 1;
24. }
25. void sort()
26. {canh temp;
27. for (int i=1;i<=m-1;i++)
28. for (int j=i+1;j<=m;j++)
29. if( c[i].w > c[j].w)
30. {     temp=c[i];
31. c[i]=c[j];
32. c[j]=temp;
33. }
34. for (i=1;i<=m;i++)
35. cout<<"Canh thu "<<i<<" noi dinh "<<c[i].dau<<" voi dinh "<<c[i].cuoi<<" co trong
    so = " <<c[i].w<<endl;
36. }
37. void updateconnect(int i, int j)
38. {
39. connect[i][j] = 1;
40. connect[j][i] = 1;
41. for (int k=1;k<=n;k++)
42. for (int l=1;l<=n;l++)
43. if(connect[k][i]==1 && connect[j][l] ==1 )
44. {
45. connect[k][l]=1;
46. connect[l][k]=1;
47. }
48. }
49. void  kruskal()
50. {
```

```
int tong=0,i=1,socanhchon=0,dinhdau,dinhcuoi;
while (socanhchon < n-1)
{dinhdau=c[i].dau;
dinhcuoi=c[i].cuoi;
if( connect[dinhdau][dinhcuoi]==0)
{
socanhchon++;
updateconnect(dinhdau,dinhcuoi);
cout<<dinhdau<<","<<dinhcuoi<<","<<c[i].w<<endl;
tong=tong+c[i].w;
}
i++;
}
cout<<tong;
}
void main()
{
readfile();
sort();
kruskal();
}
```

Toàn văn của chương trình ***cho thuật toán Prim*** này như sau:

```
1. #include<conio.h>
2. #include<stdio.h>
3. #include<iostream.h>
4. const max=100; // hoac viet #define max 100
5. const MAXINT=32767;
6. int   n,c[max][max];
7. void readfile()
8. {
9. FILE         *f;
10. f=fopen("d:\\dothi\\prim1.inp","rt");
11. fscanf(f,"%d",&n);
12. for (int i=1;i<=n;i++)
13. for (int j=1;j<=n;j++)
14. fscanf(f,"%d",&c[i][j]);
15. fclose(f);
16. }
17. void  prim()
18. {
19. int vh[max],daxet[max];
20. int tong=0,dinhdau, dinhcuoi,sodinh=1;
21. vh[sodinh]=1;
22. for (int i=1;i<=n;i++) daxet[i]=0;
23. daxet[sodinh]=1;
24. while (sodinh<n)
25. {
26. int min=MAXINT;
27. for (int i=1;i<=sodinh;i++)
28. for (int j=1;j<=n;j++)
29. if  (vh[i]!=j && c[vh[i]][j]!=0 && !daxet[j] && c[vh[i]][j]<min)
30. {
31. min=c[vh[i]][j];
32. dinhdau=vh[i];
33. dinhcuoi=j;
```

```
34. }
35. daxet[dinhcuoi]=1;
36. vh[++sodinh]=dinhcuoi;
37. tong=tong+c[dinhdau][dinhcuoi];
38. cout<<dinhdau<<"," <<dinhcuoi<<endl;
39. }
40. cout<<"Tong do dai cua cay khung nho nhat la : "<< tong;
41. }
42. void main()
43. {
44. clrscr();
45. readfile();
46. prim();
47. }
```

Prim1.inp

```
6
0 33 17 0 0 0
33 0 18 20 0 0
17 18 0 16 4 0
0 20 16 0 9 8
0 0 4 9 0 14
0 0 0 8 14 0
```

Prim2.inp

```
9
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
```

0 0 2 0 0 0 6 7 0