Chuyên đề xếp loại xuất sắc

QUI HOẠCH ĐỘNG TRÊN ĐỒ THỊ CÓ HƯỚNG, KHÔNG CHU TRÌNH

Lê Thanh Bình

THPT Chuyên Nguyễn Trãi

Giải các bài toán có nội dung đồ thị là một phần quan trọng trong chương trình tin học khuôn khổ chuyên đề này, tôi chỉ xin trao đổi với các bạn đồng nghiệp một nội dung nhỏ của lý thuyết đồ thị là "Các bài toán qui hoạch động trên đồ thị có hướng, không có chu trình".

Chuyên đề trình bày một số kinh nghiệm khi dạy về đồ thị có hướng không chu trình. Một trong những điều khá lý thú là ở đây hai nội dung chính của chương trình tin học là Qui hoạch động và lý thuyết đồ thị được kết hợp. Chính điều này cho phép xây dựng cho học sinh một cách nhìn tổng quan khi tiếp cận cả hai dạng toán này.

Phần lớn các ví dụ minh họa trong chuyên đề được lấy từ các kỳ thi học sinh giỏi khác nhau. Mục đích làm như vậy là tôi muốn trao đổi với các bạn đồng nghiệp về cách xây dựng một đề toán đồ thị sao cho vừa có thể ôn tập kiến thức học sinh, vừa có thể bám sát được chương trình thi trong các kỳ thi học sinh giỏi tin học.

I-MỘT SỐ KHÁI NIÊM VÀ BÀI TOÁN CƠ BẢN

Như chúng ta đã biết, đồ thị có thể được hình dung như là một cặp (V, E) trong đó V là tập hợp các đỉnh (trong các bài toán tin học thì V là tập hợp hữu hạn các đỉnh có thể đánh số 1, 2, ..., N) còn E là tập hợp các cung của đồ thị.

Một đồ thị có hướng không có chu trình là đồ thị không tồn tại đường đi khép kín. Cũng có thể hình dung đây là đồ thị mà số lượng đỉnh trong tất cả các thành phần liên thông mạnh đều bằng 1.

Một đồ thị có hướng không có chu trình luôn tồn tại một sắp xếp topo. Chính xác hơn, một sắp xếp topo là một cách sắp xếp các đỉnh của đồ thị thành một dãy

$x_1, x_2, \cdots, x_n$

Sao cho mọi cung $(x_i, x_j) \in E$ đều kéo theo i < j.

Việc chỉ ra một sắp xếp topo trên đồ thị có hướng không có chu trình là điều kiện tiên quyết để làm các bài toán qui hoạch động trên loại đồ thị này. Lý

do đơn giản là nếu như coi mỗi đỉnh của đồ thị là một trạng thái thì với việc sắp xếp topo chúng ta có một thứ tự trên các trạng thái này và đây chính là cách tiếp cận vấn đề theo quan điểm qui hoạch động.

Có hai cách chính đề xây dựng một sắp xếp topo trên đồ thị có hướng không có chu trình:

Cách thứ nhất: Dựa vào một tiêu chí tự nhiên mà nếu sắp xếp tăng/giảm theo tiêu chính này thì đương nhiên ta có một sắp xếp topo.

Ví dụ 1 (VOI 2008): Cho n hình tròn bán kính $r_1, r_2, ..., r_n$. Ta nói từ đường tròn bán kính a có thể nhảy tới hình tròn bán kính b nếu tồn tại một hình tròn bán kính c sao cho a+c=b (*) . Hãy tìm đường đi qua nhiều hình tròn nhất.

Dễ nhận thấy rằng điều kiện (*) chứng tỏ từ một hình tròn chỉ có thể nhảy đến một hình tròn có bán kính lớn hơn nên hiển nhiên rằng nếu ta sắp xếp lại các hình tròn sao cho bán kính của chúng tăng dần ta sẽ có một sắp xếp topo.

Thông thường các tiêu chí tự nhiên này thường dễ thấy và việc sắp xếp topo qui về việc sắp xếp tăng/giảm trên tiêu chí này. Do đó, hiển nhiên tiêu chí sắp xếp phải dựa trên dữ liệu có mối quan hệ sắp xếp hoàn toàn (thông thường là các số).

Cách thứ hai: Dựa vào thuật toán Tarjan tìm thành phần liên thông mạnh. Chú ý rằng khi đồ thị là không có chu trình thì các thành phần liên thông mạnh đều có số lượng đỉnh bằng 1. Do vậy trong trường hợp này ta chỉ cần liệt kê các đỉnh theo thứ tự sau của phép duyệt đồ thị ưu tiên chiều sâu. Mã giả của nó được viết như dưới đây

```
PROCEDURE visit(u)
  Đánh dấu u được thăm
  For v ∈ Ke(u) do if (v chưa được thăm) then
        visit(v)
  Đưa u vào danh sách sắp topo
```

(Có thể tham khảo mã Pascal trong sách giáo khoa chuyên tin. Tập 1)

Cách thứ hai được dùng khi không thể tìm được tiêu chí tự nhiên trong việc sắp xếp topo. Tuy rằng đây là cách tổng quát áp dụng cho mọi trường hợp nhưng theo kinh nghiệm của tôi thì thông thường khi sắp xếp topo ta hay sử dụng cách thứ nhất hơn.

Giả sử trên đồ thị có hướng không có chu trình G=(V,E) ta đã có một sắp xếp topo $x_1, x_2, ..., x_n$. Khi đó ta có hai bài toán cơ bản sau:

Bài toán 1: Cho mỗi cung của đồ thị một trọng số. Hãy tìm đường đi dài nhất từ đỉnh s đến đỉnh t

Đặt $f[x_i]$ lần là độ dài đường đi dài nhất từ s đến xi. Khi đó

$f[x_i] = max\{f[x_k] + d(x_k, x_i) : (x_k, x_i) \in E\}$

Một điều lý thú là thay vì tính toán trên các cung ngược (các cung tới x_i) của đồ thị theo như cách tư duy truyền thống của qui hoạch động, chúng ta sẽ sửa (update) theo các cung xuôi (đây là đặc điểm chính khi thực hiện qui hoạch động trên DAG vì nói chung xây dựng các cung ngược là một vấn đề khá phức tạp):

```
PROCEDURE DuongDiMax
  For i ∈ {1,...,n} f[i]=-∞
  For i ∈ {1,...,n}
    u=x[i]
    if (u=s) f[u]=0
    if (f[u]<>-∞)
      For v ∈ Ke(u) if f[v]<f[u]+d(u,v) then f[v]=f[u]+d(u,v)
```

Hoàn toàn tương tự ta có thể tìm đường đi ngắn nhất

Bài toán 2: Đếm số đường đi từ đỉnh s tới đỉnh t?

Gọi $f[x_i]$ là số đường đi từ s đến x_i ta có công thức

$f[x_i] = \sum\{f[x_k] : (x_k, x_i) \in E\}$

Và một lần nữa ta có chương trình qui hoạch động tương tự như trên:

```
PROCEDURE SoDuongDi
  For i ∈ {1,...,n} f[i]=0
  For i ∈ {1,...,n}
    u=x[i]
    if (u=s) f[u]=1
    if (f[u]<>-∞)
      For v ∈ Ke(u) f[v]=f[v]+f[u]
```

Hai bài toán trên là hai bài toán cơ bản trong các bài toán qui hoạch động trên đồ thị có hướng. Một lần nữa nhắc lại điều đặc biệt của qui hoạch động trên đồ thị có hướng là ta tính toán theo cung của đồ thị, do vậy ta thực hiện việc sửa (update) nhãn thay vì tính max, tính min hoặc đếm như trong qui hoạch động thông thường (lý do đơn giản là xây dựng đồ thị ngược nói chung là khá phức tạp và tốn kém)

II-MỘT SỐ BÀI TẬP MINH HỌA

Bài tập 1 (VOI 2008):

Nhảy lò cò là trò chơi dân gian của Việt Nam. Người trên hành tinh X cũng rất thích trò chơi này và họ đã cải biên trò chơi này như sau: Trên mặt phẳng vẽ n vòng tròn được đánh số từ 1 đến n. Tại vòng tròn i người ta điền số nguyên dương ai. Hai số trên hai vòng tròn tùy ý không nhất thiết phải khác nhau. Tiếp đến người ta vẽ các mũi tên, mỗi mũi tên hướng từ một vòng tròn đến một vòng tròn khác. Quy tắc vẽ mũi tên là: Nếu có ba số ai, aj, ak thỏa mãn ak = ai + aj thì vẽ mũi tên hướng từ vòng tròn i đến vòng tròn k và mũi tên hướng từ vòng tròn j đến vòng tròn k. Người chơi chỉ được di chuyển từ một vòng tròn đến một vòng tròn khác nếu có mũi tên xuất phát từ một trong số các vòng tròn, di chyển theo cách mũi tên đã vẽ để đi đến các vòng tròn khác. Người thắng cuộc sẽ là người tìm được cách di chuyển qua nhiều vòng tròn nhất.

Yêu cầu:Hãy xác định xem trong trò chơi mô tả ở trên, nhiều nhất có thể di chuyển được qua bao nhiêu vòng tròn.

Như phần trước đã nhận xét, nếu coi mỗi vòng tròn là một đỉnh của đồ thị. Hai vòng tròn kề nhau nếu có thể nhảy trực tiếp đến nhau thì ta có một DAG và bài toán qui về tìm đường đi dài nhất trên đồ thị này.

Một điểm cần lưu ý là để chương trình chạy đạt yêu cầu thì điều kiện $j \in Ke(i)$ cần thực hiện việc tìm kiếm nhị phân để kiểm tra

Bài tập 2:

Một ông chủ có 2 cái máy cày cho thuê, có N người nông dân đăng ký thuê máy cày. Người thứ i muốn thuê máy bắt đầu từ thời điểm s[i] đến hết thời điểm t[i]. Giá thuê một máy cày trong một đơn vị thời gian mất một đồng, như vậy nếu cho người thứ i thuê ông chủ có thể thu về được t[i]-s[i]+1 đồng. Tại một thời điểm một máy có nhiều nhất một người sử dụng.

Yêu cầu: Tính số tiền nhiều nhất có thể thu được.

Input:

Dòng đầu là số nguyên N (N<=100)

N dòng sau, mỗi dòng 2 số nguyên thể hiện số s[i], t[i] (0<=s[i]<=t[i]<=109)

Output:

Gồm 1 dòng duy nhất chứa 1 số là số tiền lớn nhất có thể thu được.

Ta xây dựng đồ thị như sau:

Tập đỉnh V={(i,j): với ý nghĩa là máy thứ nhất làm công việc cuối cùng là i và máy thứ hai làm công việc cuối cùng là j}

Tập cung E={(i,j)-(i,k): nếu sau khi làm j máy thứ 2 làm được công việc k và (i,j)-(k,j) nếu sau khi làm i máy thứ nhất làm được k}

Dễ thấy bài toán qui về tìm đường đi dài nhất từ đỉnh (0,0).

Đây là DAG và một sắp xếp topo tự nhiên là sắp xếp theo thời gian kết thúc thuê máy tăng dần. Do vậy ta hoàn toàn có thể sử dụng mô hình bài toán 1 để giải quyết:

Bài tập 3:

Cho đồ thị có hướng N đỉnh (N≤16) trong đó các cạnh có trọng số. Hãy tìm đường đi Haminton (đường đi qua tất cả các đỉnh) ngắn nhất?

Ta xây dựng một đồ thị mới trong đó mỗi đỉnh là một cặp gồm dãy bit ($b_1, b_2, ..., b_n$) với $b_i = 1$ nếu như đỉnh i đã đi qua còn $b_i = 0$ nếu như đỉnh i chưa đi qua và đỉnh u thể hiện đỉnh cuối cùng trên hành trình là u . Như vậy mỗi đỉnh là một cặp (x, u) với x là số nguyên thể hiện dãy bit trên. Đỉnh (x, u) đi đến được (y,v) nếu như bit v của x bằng 0 và bít v của y bằng 1 (các bit khác trùng nhau) và u đi đến được v.

Dễ thấy rằng đồ thị xây dựng như trên là DAG với sắp xếp topo tự nhiên là sắp xếp các đỉnh (x, u) theo số lượng bit 1 của x tăng dần. Khi đó trên sắp xếp topo này các đỉnh được chia thành từng lớp (x có 0 bit 1, x có 1 bit 1, ..., x có n bit 1) và ta có thể sử dụng mô hình bài toán 1 (tìm đường đi dài nhất từ tập đỉnh có 1 bit 1 đến tập đỉnh có n bít 1) với một chút cải tiến là kết hợp với một hàng đợi.

III-CÁC ĐỒ THỊ CÓ HƯỚNG KHÔNG CÓ CHU TRÌNH CẢM SINH

Khi dạy học sinh các thuật toán cơ bản như duyệt đồ thị ưu tiên chiều rộng, duyệt đồ thị ưu tiên chiều sâu, tìm đường đi ngắn nhất trên đồ thị không có chu trình âm, cần phải nhấn mạnh đến các sản phẩm của các thuật toán này. Một điều rất thú vị là có rất nhiều sản phẩm là đồ thị bộ phận có hướng không có chu trình mà tôi tạm gọi là các đồ thị có hướng không có chu trình cảm sinh. Có rất nhiều bài tập về đồ thị trong các kỳ thi gần đây sử dụng các đồ thị bộ phận này.

1. DAG đường đi ít cạnh nhất

Khi thực hiện duyệt đồ thị ưu tiên chiều rộng (BFS) từ đỉnh s ta gọi d[i] là độ dài đường đi ít cạnh nhất từ s đến i (d[i]=∞ nếu không có đường đi từ s đến i). Xây dựng đồ thị bộ phận G'=(V',E') như sau:

$V' \equiv V$

E' = {(u,v) ∈ E: d[v]=d[u]+1}

Đồ thị này còn được gọi là đồ thị đường đi ít cạnh nhất vì tất cả các đường đi ngắn nhất (theo nghĩa ít cạnh nhất) đều đi qua các cung của đồ thị này.

Dễ dàng nhận thấy G' là một DAG và đặc biệt, sắp xếp topo tự nhiên của nó là sắp xếp theo giá trị d[i] tăng dần. Nó cũng chính là thứ tự của các đỉnh khi đưa vào/lấy ra khỏi hàng đợi trong phép duyệt BFS (điều này khiến cho ta không phải thực hiện một phép sort đầy đủ nữa). Chính xác hơn, nếu gọi $x_1, x_2, ..., x_p$ là các đỉnh theo thứ tự đưa vào hàng đợi thì ta có một sắp xếp topo trên G'.

Với đồ thị G' ta có thể xây dựng một số bài toán mở rộng cho BFS như:

Bài toán 3: Hãy tìm đường đi ngắn nhất (ít cạnh nhất) từ đỉnh s đến đỉnh t. Nếu như có nhiều đường đi như vậy thì tìm đường đi có:

Giá thành nhỏ nhất/lớn nhất

Số đỉnh đi qua nhiều nhất/ít nhất

Để giải bài toán 3, trước tiên chúng ta thực hiện BFS để xây dựng đồ thị G' sau đó trên G' ta giải bài toán tìm đường đi ngắn nhất/dài nhất dựa trên sắp xếp topo của nó (bài toán 1)

Bài toán 4: Hãy đếm số đường đi ngắn nhất từ s đến t?

Đây chính là bài toán 2 (đếm số đường đi) trên đồ thị G' với một sắp xếp topo

Bài tập 4 (VOI 2009):

Trong mạng xã hội, mỗi trang web được tổ chức trên một máy tính thành viên và cung cấp dịch vụ truy nhập tới một số trang web khác. Để truy nhập tới một trang web nào đó không có trong danh mục kết nối trực tiếp của mình, người dùng phải truy nhập tới trang web khác có kết nối với mình, dựa vào danh mục dịch vụ của trang web này để chuyển tới trang web khác theo tùy chọn, cứ như thế cho đến khi tới được trang web mình cần. Thời gian để truy nhập tới một trang web phụ thuộc chủ yếu và số lần mở trang web trong quá trình truy nhập. Như vậy, người dùng cần chủ động chọn lộ trình truy nhập hợp lí.

Sau một thời gian làm việc trên mạng, Sáng - một thành viên nhiệt thành đã tích lũy kinh nghiệm, tạo một cơ sở dữ liệu, cho biết từ một trang web có thể đi tới những trang web nào trong mạng. Trong cơ sở dữ liệu, các trang web được đánh số từ 1 đến n và có m bản ghi, mỗi bản ghi có dạng cặp có thứ tự (u, v) cho biết trang web u có kết nối tới trang web v ($1 \le u, v \le n, u \ne v$). Cơ sở dữ liệu chưa được chuẩn hóa, vì vậy có thể chứa các cặp (u, v) giống nhau.

Trang web của Sáng có số hiệu là s. Dựa vào cơ sở dữ liệu, Sáng có thể xác định lộ trình truy nhập nhanh nhất (tức là số lần phải mở trang web là ít nhất) từ trang web s tới trang web u bất kì. Tuy vậy, ở mạng xã hội, mọi chuyện đều có thể xảy ra: một khu vực nào đó bị mất điện, máy của một thành viên bị hỏng, trang web đó đang bị đóng để nâng cấp, ... Kết quả là một vài trang web nào đó có thể tạm thời không hoạt động. Như vậy, nếu từ s có ít nhất hai lộ trình nhanh nhất khác nhau tới u thì khả năng thực hiện truy nhập được một cách nhanh nhất tới u là lớn hơn so với những trang web chỉ có duy nhất một lộ trình nhanh nhất. Hai lộ trình gọi là khác nhau nếu có ít nhất một trang web có ở lộ trình này mà không có ở lộ trình kia hoặc cả hai lộ trình cùng đi qua những trang web như nhau nhưng theo các trình tự khác nhau. Những trang web mà từ s tới đó có ít ra là hai lộ trình nhanh nhất khác nhau được gọi là ổn định đối với s. Trang web mà từ s không có lộ trình tới nó là không ổn định đối với s.

Yêu cầu: Cho các số nguyên dương n, m, s và m cặp số (u, v) xác định từ u có thể kết nối trực tiếp tới được v. Hãy xác định số lượng trang web ổn định đối với s.

Dữ liệu:

Dòng đầu tiên chứa 3 số nguyên n, m và s ($2 \le 10000, 1 \le m \le 50000, 1 \le s \le n$).

Mỗi dòng trong m dòng tiếp theo chứa 2 số nguyên u và v ($1 \le u, v \le n, u \ne v$).

Các số trên một dòng được ghi cách nhau ít nhất một dấu cách.

Kết quả: Một số nguyên - số trang web ổn định đối với s.

Bài toán trên là bài toán đếm các đỉnh có ít nhất hai đường đi ngắn nhất từ s. Phương pháp giải nó là bài toán 4 (với một lưu ý là ta không thực sự đếm mà chỉ cần phân biệt các đỉnh có 0, 1, hơn 1 đường đi ngắn nhất từ s)

2. DAG đường đi ngắn nhất

Các thuật toán Dijkstra, Ford_bellman tìm đường đi ngắn nhất từ một đỉnh s đều cho một mảng dist[i] là khoảng cách ngắn nhất từ đỉnh s đến đỉnh i (dist[i]=∞ nếu không có đường đi từ s đến i). Tương tự như trên, sau khi có mảng dist[i] ta có đồ thị bộ phận G'=(V',E') như sau:

$V' \equiv V$

$E'=\{(u,v) \in E: dist[v]=dist[u]+L(u,v)\}$

G' cũng là DAG, DAG này là DAG đường đi ngắn nhất. Nếu sử dụng thuật toán Dijkstra thì một sắp xếp topo trên DAG này là thứ tự lấy ra các đỉnh khỏi hàng đợi ưu tiên, còn nếu sử dụng Ford_Bellman thì ta phải thực hiện một phép sort trên mảng dist.

Cũng như DAG đường đi ít cạnh nhất, chúng ta cũng có một số bài toán dựa trên DAG này giống như bài toán 3, bài toán 4. Dưới đây là một số ví dụ điển hình:

Bài tập 5 (VOI 2007):

Trên một mạng lưới giao thông có n nút, các nút được đánh số từ 1 đến n và giữa hai nút bất kỳ có không quá một đường nối trực tiếp (đường nối trực tiếp là một đường hai chiều). Ta gọi đường đi từ nút s đến nút t là một dãy các nút và các đường nối trực tiếp có dạng:

s = u1, e1, u2,..., ui, ei, ui+1, ..., uk-1, ek-1, uk = t,

trong đó u1, u2, …, uk là các nút trong mạng lưới giao thông, ei là đường nối trực tiếp giữa nút ui và ui+1 (không có nút uj nào xuất hiện nhiều hơn một lần trong dãy trên, j = 1, 2, …, k).

Biết rằng mạng lưới giao thông được xét luôn có ít nhất một đường đi từ nút 1 đến nút n.

Một robot chứa đầy bình với w đơn vị năng lượng, cần đi từ trạm cứu hoả đặt tại nút 1 đến nơi xảy ra hoả hoạn ở nút n, trong thời gian ít nhất có thể. Thời gian và chi phí năng lượng để robot đi trên đường nối trực tiếp từ nút i đến nút j tương ứng là tij và cij ($1 \le i, j \le n$). Robot chỉ có thể đi được trên đường nối trực tiếp từ nút i đến nút j nếu năng lượng còn lại trong bình chứa không ít hơn cij ($1 \le i, j \le n$). Nếu robot đi đến một nút có trạm tiếp năng lượng (một nút có thể có hoặc không có trạm tiếp năng lượng) thì nó tự động được nạp đầy năng lượng vào bình chứa với thời gian nạp coi như không đáng kể.

Yêu cầu: Hãy xác định giá trị w nhỏ nhất để robot đi được trên một đường đi từ nút 1 đến nút n trong thời gian ít nhất.

Input

Dòng đầu tiên chứa một số nguyên dương n ($2 \le n \le 500$);

Dòng thứ hai chứa n số, trong đó số thứ j bằng 1 hoặc 0 tương ứng ở nút j có hoặc không có trạm tiếp năng lượng (j = 1, 2, …, n);

Dòng thứ ba chứa số nguyên dương m ($m \le 30000$) là số đường nối trực tiếp có trong mạng lưới giao thông;

Dòng thứ k trong số m dòng tiếp theo chứa 4 số nguyên dương i, j, tij, cij (tij, cij ≤ 10000) mô tả đường nối trực tiếp từ nút i đến nút j, thời gian và chi phí năng lượng tương ứng.

Hai số liên tiếp trên một dòng trong file dữ liệu cách nhau ít nhất một dấu cách.

Output: Ghi ra số nguyên dương w tìm được.

Trước tiên sử dụng thuật toán Dijkstra chúng ta tìm được DAG đường đi ngắn nhất. Một lần nữa chú ý rằng sắp xếp topo trên DAG này chính là thứ tự lấy ra các đỉnh khỏi hàng đợi ưu tiên. Trên DAG đường đi ngắn nhất này ta giải bài toán tìm năng lượng tối thiểu. Kỹ thuật dùng ở đây có thể là tìm kiếm nhị phân và ta đi đến bài toán cơ bản "Cho năng lượng x, hỏi rằng với năng lượng này có thể đi đến được n hay không?" bài toán này hoàn toàn giải bằng qui hoạch động.

Bài tập 6 (IOICamp maraton 2006):

Ngày 27/11 tới là ngày tổ chức thi học kỳ I ở trường ĐH BK. Là sinh viên năm thứ nhất, Hiếu không muốn vì đi muộn mà gặp trục trặc ở phòng thi nên đã chuẩn bị khá kỹ càng. Chỉ còn lại một công việc khá gay go là Hiếu không biết đi đường nào tới trường là nhanh nhất.

Thường ngày Hiếu không quan tâm tới vấn đề này lắm cho nên bây giờ Hiếu không biết phải làm sao cả . Bản đồ thành phố là gồm có N nút giao thông và M con đường nối các nút giao thông này. Có 2 loại con đường là đường 1 chiều và đường 2 chiều. Độ dài của mỗi con đường là một số nguyên dương.

Nhà Hiếu ở nút giao thông 1 còn trường ĐH BK ở nút giao thông N. Vì một lộ trình đường đi từ nhà Hiếu tới trường có thể gặp nhiều yếu tố khác như là gặp nhiều đèn đỏ , đi qua công trường xây dựng, ... phải giảm tốc độ cho nên Hiếu muốn biết là có tất cả bao nhiêu lộ trình ngắn nhất đi từ nhà tới trường. Bạn hãy lập trình giúp Hiếu giải quyết bài toán khó này.

Input

Dòng thứ nhất ghi hai số nguyên N và M.

M dòng tiếp theo, mỗi dòng ghi 4 số nguyên dương K, U, V, L. Trong đó:

K = 1 có nghĩa là có đường đi một chiều từ U đến V với độ dài L.

K = 2 có nghìa là có đường đi hai chiều giữa U và V với độ dài L.

Output: Ghi hai số là độ dài đường đi ngắn nhấn và số lượng đường đi ngắn nhất. Biết rằng số lượng đường đi ngắn nhất không vượt quá phạm vì int64 trong pascal hay long long trong C++.

Đầu tiên chúng ta xây dựng DAG đường đi ngắn nhất (bằng thuật toán Dijkstra). Bài toán qui về bài đếm số đường đi trên DAG này. Đây chính là bài toán 3

Bài tập 7 (IOICAMP4):

Theo thống kê cho biết mức độ tăng trưởng kinh tế của nước Peace trong năm 2006 rất đáng khả quan. Cả nước có tổng cộng N thành phố lớn nhỏ được đánh số tuần tự từ 1 đến N phát triển khá đồng đều. Giữa N thành phố này là một mạng lưới gồm M đường đi hai chiều, mỗi tuyến đường nối 2 trong N thành phố sao cho không có 2 thành phố nào được nối bởi quá 1 tuyến đường. Trong N thành phố này thì thành phố 1 và thành phố N là 2 trung tâm kinh tế lớn nhất nước và hệ thống đường đảm bảo luôn có ít nhất một cách đi từ thành phố 1 đến thành phố N.

Tuy nhiên,cả 2 trung tâm này đều có dấu hiệu quá tải về mật độ dân số. Vì vậy, đức vua Peaceful quyết định chọn ra thêm một thành phố nữa để đầu tư thành một trung tâm kinh tế thứ ba. Thành phố này sẽ tạm ngưng mọi hoạt động thường nhật, cũng như mọi luồng lưu thông ra vào để tiến hành nâng cấp cơ sở hạ tầng. Nhưng trong thời gian sửa chữa ấy, phải bảo đảm đường đi ngắn nhất từ thành phố 1 đến thành phố N không bị thay đổi, nếu không nền kinh tế quốc gia sẽ bị trì trệ.

Vị trí và đường nối giữa N thành phố được mô tả như một đồ thị N đỉnh M cạnh. Hãy giúp nhà vua đếm số lượng thành phố có thể chọn làm trung tâm kinh tế thứ ba sao cho thành phố được chọn thỏa mãn các điều kiện ở trên

Input

Dòng đầu tiên ghi 2 số nguyên dương N và M là số thành phố và số tuyến đường.

Dòng thứ i trong số M dòng tiếp theo ghi 3 số nguyên dương xi, yi và di với ý nghĩa tuyến đường thứ i có độ dài di và nối giữa 2 thành phố xi, yi.

Output:

Dòng đầu tiên ghi số tự nhiên S là số lượng các thành phố có thể chọn làm trung tâm kinh tế thứ ba.

S dòng tiếp theo, mỗi dòng ghi 1 số nguyên dương là số thứ tự của thành phố được chọn (In ra theo thứ tự tăng dần)

Một thành phố được chọn là thành phố mà khi rút nó ra khỏi đồ thị không ảnh hưởng đến số lượng đường đi ngắn nhất từ 1 đến n.

Đặt f[u] là số lượng đường đi ngắn nhất từ 1 đến u và g[u] là số lượng đường đi ngắn nhất từ u đến n (hai mảng này có thể tính trên các DAG đường đi ngắn nhất của đồ thị xuôi và đồ thị ngược). u là thành phố được chọn khi f[u]*g[u]<f[n]

3. DAG Liên thông mạnh

Khi tìm thành phần liên thông mạnh một sản phẩm hết sức quan trọng là đồ thị các thành phần liên thông mạnh trong đó mỗi đỉnh của đồ thị này là một thành phần liên thông mạnh của đồ thị ban đầu và thành phần liên thông A kề với thành phần liên thông B nếu như có cung của đồ thị ban đầu đi từ một đỉnh của A đến một đỉnh của B.

Dễ nhận thấy đồ thị các thành phần liên thông mạnh là một DAG (vì nếu không ta có thể mở rộng một thành phần liên thông mạnh nào đó) đây là DAG liên thông mạnh

DAG liên thông mạnh có một sắp xếp topo tự nhiên là thứ tự tìm thấy các thành phần liên thông mạnh trong thuật toán Tarjan (thành phần liên thông mạnh nào tìm thấy trước thì xếp trước, thành phần liên thông mạnh nào tìm thấy sau thì xếp sau).

DAG liên thông mạnh phải được xây dựng riêng (bằng một vòng lặp duyệt qua các cung của đồ thị cũ). Hơn nữa, ta cần lưu thêm các thông tin về mỗi đỉnh của đồ thị này.

Bài tập 8:

Tất cả các đường trong thành phố của Siruseri đều là một chiều. Theo luật của quốc gia này, tại mỗi giao lộ phải có một máy ATM. Điều đáng ngạc nhiên là các cửa hàng chơi điện tử cũng nằm ở các giao lộ, tuy nhiên, không phải tại giao lộ nào cũng có cửa hàng chơi điện tử.

Banditji là một tên trộm nổi tiếng. Hắn quyết định làm một vụ động trời: khoắng sạch tiền trong các máy ATM trên đường đi, sau đó ghé vào một cửa hàng chơi điện tử để thư giản. Nhờ có mạng lưới thông tin rộng rãi, Banditji biết được số tiền có ở mỗi máy ATM ngày hôm đó. Xuất phát từ trung tâm, tên trộm lái xe đi dọc theo các phố, vét sạch tiền ở các ATM gặp trên đường đi. Banditji có thể đi lại nhiều lần trên một số đoạn phố, nhưng sẽ không thu gì được thêm từ các ATM đã bị khoắng trước đó. Lộ trình của Banditji phải kết thúc ở giao lộ có cửa hàng chơi điện tử. Banditji biết cách vạch lộ trình để tổng số tiền trộm được là lớn nhất.

Yêu cầu: Cho biết n – số giao lộ, m – số đoạn đường nối 2 giao lộ, p – số giao lộ có cửa hàng chơi điện tử và các nơi có cửa hàng, ai – số tiền trong ATM đặt ở giao lộ i, s – giao lộ trung tâm. Hãy xác định tổng số lượng tiền bị trộm (n, $m \le 500\,000$, $0 \le ai \le 4\,000$).

Dữ liệu: Vào từ file văn bản ATM.INP:

Dòng đầu tiên chứa 2 số nguyên n và m,

Mỗi dòng trong m dòng tiếp theo chứa 2 số nguyên u và v xác định đường đi từ giao lộ u tới giao lộ v,

Dòng thứ i trong n dòng tiếp theo chứa số nguyên ai,

Dòng thứ n+m+2 chứa 2 số nguyên s và p,

Dòng cuối cùng chứa p số nguyên xác định các giao lộ có cửa hàng chơi điện tử.

Kết quả: Đưa ra file văn bản ATM.OUT một số nguyên – số tiền bị trộm.

Sử dụng Tarjan chúng ta tìm được DAG các thành phần liên thông mạnh. Với mỗi đỉnh (tức là mỗi thành phần liên thông mạnh) chúng ta lưu hai thông tin: tổng số tiền trong các trạm ATM và số cửa hàng điện tử.

Bài toán trở thành tìm đường đi có tổng tiền lớn nhất đến các đỉnh có số cửa hàng điện tử lớn hơn không. Do là DAG và có sắp xếp topo nên điều này có thể làm được bằng qui hoạch động tương tự như trên.

Có thể thấy DAG cho một lớp bài toán khá phong phú và đa dạng trên đồ thị. Các DAG cảm sinh dựa trên các thuật toán cơ bản như BFS, Dijkstra, Tarjan có lẽ là những DAG thú vị nhất. Điều làm cho việc giải quyết các bài toán trên các DAG này là dễ dàng chính là do các sắp xếp topo tự nhiên mà các thuật toán cơ bản mang lại.

Dưới quan điểm dạy học thì khai thác hết các kết quả của các thuật toán là một thói quen tốt cần xây dựng cho học sinh như là một kỹ năng rèn luyện. Nếu các em có kỹ năng này thì việc áp dụng các thuật toán một cách uyển chuyển là một hệ quả hiển nhiên.

Trên đây là một vài kinh nghiệm muốn trao đổi với các bạn đồng nghiệp. Rất mong được mọi người chỉ giáo. Để kết thúc, xin trích hai câu cuối trong "Truyện Kiều" của cụ Nguyễn Du:

"Lời quê chắp nhặt dông dài
Mua vui cũng được một vài trống canh!"

Chuyên đề xếp loại A

Chuyên đề xếp loại A

Nhóm GV Tin trường THPT chuyên Lào Cai

CHUYÊN ĐỀ
ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ

A. MỞ ĐẦU

1. Lý do chọn đề tài

Lý thuyết đồ thị là một lĩnh vực được phát triển từ rất lâu, được nhiều nhà khoa học quan tâm nghiên cứu nó có vai trò hết sức quan trọng trong nhiều lĩnh vực. Trong Tin học lý thuyết đồ thị được ứng dụng một cách rộng rãi có rất nhiều thuật toán được nghiên cứu và ứng dụng. Trong chương trình môn Tin học của THPT chuyên phần lý thuyết đồ thị nói chung và các thuật toán tìm đường đi ngắn nhất trên đồ thị là nội dung rất quan trọng, trong các kỳ thi học sinh giỏi xuất hiện rất nhiều các bài toán liên quan đến việc tìm đường đi ngắn nhất trên đồ thị. Tuy nhiên trong qua trình giảng dạy tôi thấy học sinh vẫn còn khó khăn trong việc phân tích bài toán để có thể áp dụng được thuật toán và cài đặt giải bài toán. Vì vậy tôi chọn chuyên đề này để giúp học sinh có cái nhìn tổng quan hơn về các thuật toán tìm đường đi ngắn nhất trên đồ thị.

2. Mục đích của đề tài

Về nội dung kiến thức của các thuật toán tìm kiếm trên đồ thị đã có rất nhiều tài liệu đề cập đến, trong chuyên đề này tôi chỉ tổng hợp lại các nội dung kiến thức đã có và đưa vào áp dụng để giải một số bài toán cụ thế, để làm tài liệu tham khảo cho học sinh và giáo viên trong quá trình học tập và giảng dạy.

A. NỘI DUNG

I, Giới thiệu bài toán đường đi ngắn nhất

- Trong thực tế có rất nhiều các bài toán chẳng hạn như trong mạng lưới giao thông nối giữa các Thành Phố với nhau, mạng lưới các đường bay nối các nước với nhau người ta không chỉ quan tâm tìm đường đi giữa các địa điểm với nhau mà phải lựa chọn một hành trình sao cho tiết kiệm chi phí nhất (chi phí có thể là thời gian, tiền bạc, khoảng cách…). Khi đó người ta gán cho mỗi cạnh của đồ thị một giá trị phản ánh chi phí đi qua cạnh đó và cố gắng tìm ra một hành trình đi qua các cạnh với tổng chi phí thấp nhất.
- Ta đi xét một đồ thị có hướng $G = (V, E)$ với các cung được gán trọng số (trọng số ở đây là chi phí). Nếu giữa hai đỉnh u, v không có cạnh nối thì ta

có thể thêm vào cạnh "giả" với trọng số $a_{ij} = +\infty$. Khi đó đồ thị G có thể giả thiết là đồ thị đầy đủ.

- Nếu dãy $v_0, v_1, ..., v_p$ là một đường đi trên G thì độ dài của nó được định nghĩa là tổng các trọng số trên các cung của nó: $\sum_{i=1}^{p} a(v_{i-1}, v_i)$
- Bài toán đặt ra là tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh đích $t \in V$. Đường đi như vậy gọi là đường đi ngắn nhất từ s đến t và độ dài của nó ta còn gọi là khoảng cách từ s đến t, kí hiệu $d(s, t)$.

- Nhận xét:

+ Khoảng cách giữa hai đỉnh của đồ thị có thể là số âm.

+ Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt $d(s, t) = +\infty$.

+ Nếu như trong đồ thị, mỗi chu trình đều có độ dài dương thì đường đi ngắn nhất sẽ không có đỉnh nào bị lặp lại. Đường đi không có đỉnh lặp lại gọi là *đường đi cơ bản*. Còn nếu trong đồ thị có chứa chu trình với độ dài âm (gọi là chu trình âm) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị là không xác định, bởi vì bằng cách đi vòng theo chu trình này một số lần đủ lớn, ta có thể chỉ ra đường đi giữa các đỉnh này có độ dài nhỏ hơn bất kì số thực nào cho trước. Trong những trường hợp như vậy ta có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất.

+ Trong thực tế, bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Nhiều bài toán có thể dẫn về bài toán trên. Ví dụ bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thuỷ hoặc đường không. Bài toán lập lịch thi công các công đoạn trong một công trình lớn, bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, ... Hiện nay có rất nhiều phương pháp để giải bài toán trên. Trong bài này ta xét các giải thuật được xây dựng trên cơ sở lý thuyết đồ thị tỏ ra là hiệu quả cao nhất.

II, Đường đi ngắn nhất xuất phát từ một đỉnh

1, Bài toán tìm đường đi ngắn nhất xuất phát từ một đỉnh được phát biểu như sau : Cho đồ thị có trọng số G=(V,E,w) hãy tìm đường đi ngắn nhất từ đỉnh xuất phát s đến các đỉnh còn lại của đồ thị. Độ dài đường đi từ đỉnh s đến đỉnh t kí hiệu là δ(s,t) gọi là khoảng cách từ s tới t nếu như không tồn tại khoảng cách từ s tới t thì ta đặt khoảng cách đó là + ∞.

2,Giải thuật Ford-Bellman

- Thuật toán Ford – Bellman có thể dùng để tìm đường đi ngắn nhất xuất phát từ một đỉnh s thuộc V trong trường hợp đồ thị G=(V,E,w) không có chu trình âm thuật toán như sau:

+ Gọi $d[v]$ là khoảng cách từ đỉnh s đến đỉnh $v \in V$, $d[v]=0$, $d[t]=+\infty$. Sau đó ta thực hiện phép co tức là mỗi khi phát hiện $d[u] + a[u, v] < d[v]$ thì cận trên $d[v]$ sẽ được tốt lên $d[v] := d[u] + a[u, v]$. Quá trình trên kết thúc khi nào ta không thể làm tốt thêm được bất cứ cận trên nào . Khi đó giá trị của mỗi $d[v]$ sẽ cho khoảng cách từ s đến v. Nói riêng $d[t]$ là độ dài ngắn nhất giữa hai đỉnh s và t.

Cài đặt thuật toán

```
Procedure    Ford_Bellman ;
Begin
    For i := 1 to n do
    begin
        d [i]:=maxint ;
        tr[i]:=maxint ;
    end ;
    d[s]:=0;
    Repeat
        Ok:=true;
        For i:=1 to n do
         if d[i]<>maxint then
        for j:=1 to n do
        if (a[i,j]<>0)and(d[i]+a[i,j]<d[j]) then
        begin
                ok:=false;
                d[j]:=d[i]+a[i,j];
                tr[j]:=i;
        end;
    until ok ;
```

Nhận xét:

- Việc chứng minh tính đúng đắn của giải thuật trên dựa trên cơ sở nguyên lý tối ưu của quy hoạch động.

- Độ phức tạp tính toán của giải thuật Ford-Bellman là $O(n^3)$.
- Thực chất của thuật toán này là thuật toán Quy Hoạch Động trong đó , d[i] là mảng độ dài ngắn nhất đi từ s đến i . vậy nếu t là đỉnh cần thiết thì d[t] là độ dài cần tìm . Còn nếu muốn lưu lại đường đi thì chúng ta dùng mảng Tr [i] để đi ngược lại .

3, Thuật toán Dijkstra

Trong trường hợp đồ thị G=(V,E,w) có trọng số trên các cung không âm thuật toán Dijkstra đề cập dưới đây hoạt động hiệu quả hơn nhiều so với thuật toán Ford – Bellman. Thuật toán Dijkstra như sau:

Bước 1: Khởi tạo

Với đỉnh v ∈ V , ta gọi d[v] là độ dài đường đi từ s tới v ban đầu khởi tạo d[v]=0, d[t]=+ ∞. ∀v≠ s . Nhẫn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhẫn tự do có nghĩa là có thể tối ưu được nữa, nhẫn cố định d[v] là đường đi ngắn nhất từ s tới v nên không thể tối ưu được nữa. Ta dùng thêm một mảng Free[] để đánh dấu nếu d[v] là tự do thì Free[v]=True, ngược lại Free[v]=Flase. Ban đầu các nhẫn đều tự do.

Bước 2: Lặp

Bước lặp gồm hai thao tác :

- Cố định nhẫn: chọn trong các đỉnh có nhẫn tự do lấy ra đỉnh u có d[u] nhỏ nhất và cố định d[u]
- Sửa nhẫn: dùng đỉnh u để xét tất cả các đỉnh v và sửa lại các nhẫn d[v] theo công thức sau:

d[v]= min (d[v], d[u]+c[u,v])

Bước lặp sẽ kết thúc khi mà đỉnh t(đỉnh đích) đã được cố định nhẫn.

Bước 3: Kết hợp với lưu vết đường đi trên từng bước sửa nhẫn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi d[t]=+ ∞.

Cài đặt thuật toán:

```
Const
  MAX_N = 100;
  FI = 'dijkstra.inp';
  FO = 'dijkstra.out';
Var
  n, nU, s, t : integer;
  a : array[1..MAX_N, 1..MAX_N] of integer;
  d, tr, U : array[1..MAX_N] of integer;
```

```
  f : text;
Procedure Doc;
Var i, j : integer;
Begin
  assign(f, FI); reset(f);
  read(f, n, s, t);
  for i := 1 to n do
    for j := 1 to n do read(f, a[i, j]);
  close(f);
End;
Procedure Khoi_tao;
Var i : integer;
Begin
  for i := 1 to n do
    begin
      tr[i] := s;
      d[i] := a[s, i];
    end;
  for i := 1 to n do U[i] := i;
  U[s] := U[n];
  nU := n - 1;
End;
Function Co_dinh_nhan : integer;
Var i, j, p : integer;
Begin
  { Tim p }
  i := 1;
  for j := 2 to nU do
    if d[U[i]] > d[U[j]] then i := j;
  p := U[i];
  { Loai p ra khoi U }
  U[i] := U[nU];
  nU := nU - 1;
  Co_dinh_nhan := p;
End;
```

```
Procedure Sua_nhan(p : integer);
Var i, x : integer;
Begin
  for i := 1 to nU do
    begin
      x := U[i];
      if d[x] > d[p] + a[p, x] then
        begin
          tr[x] := p;
          d[x] := d[p] + a[p, x];
        end;
    end;
End;
Procedure Print(i : integer);
Begin
  if i = s then
    begin
      writeln(f, d[t]);
      write(f, s);
      exit;
    end;
  Print(tr[i]);
  write(f, ' ', i);
End;
Procedure Ghi;
Begin
  assign(f, FO); rewrite(f);
  Print(t);
  close(f);
End;
Procedure Dijkstra;
Var p : integer;
Begin
  Khoi_tao;
  repeat
```

```
    p := Co_dinh_nhan;
    Sua_nhan(p);
  until p = t;
End;
Begin
  Doc;
  Dijkstra;
  Ghi;
End.
```

4, Thuật toán Dijkstra với cấu trúc Heap

Cấu trúc Heap và một số phép xử lí trên Heap

* *Mô tả Heap:* Heap được mô tả như một cây nhị phân có cấu trúc sao cho giá trị khoá ở mỗi nút không vượt quá giá trị khoá của hai nút con của nó (suy ra giá trị khoá tại gốc Heap là nhỏ nhất).

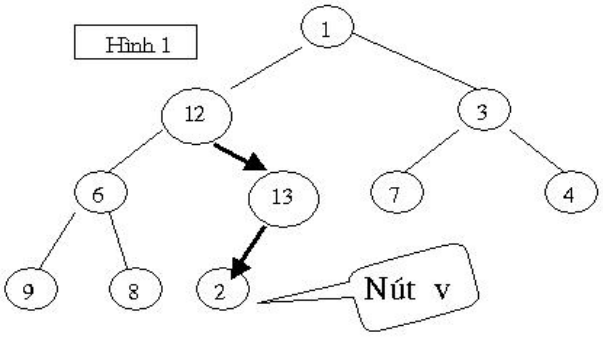
** Hai phép xử lí trên Heap*

- Phép cập nhật Heap

Vấn đề: Giả sử nút v có giá trị khoá nhỏ đi, cần chuyển nút v đến vị trí mới trên Heap để bảo toàn cấu trúc Heap

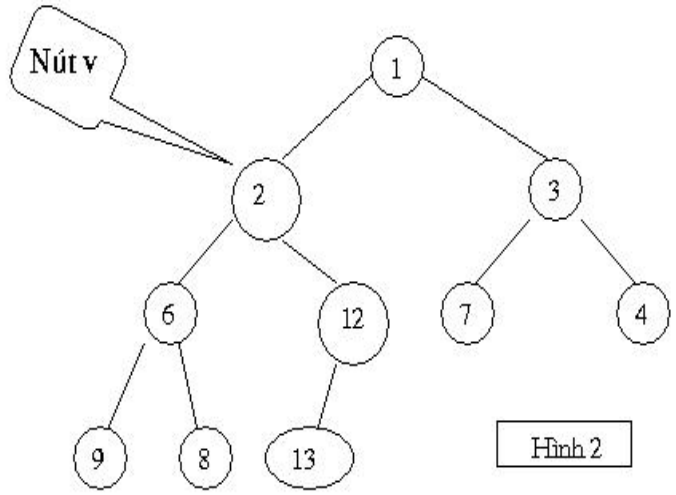
Giải quyết:

+ Nếu nút v chưa có trong Heap thì tạo thêm nút v thành nút cuối cùng của Heap (*hình1*)



+ Chuyển nút v từ vị trí hiện tại đến vị trí thích hợp bằng cách tìm đường đi ngược từ vị trí hiện tại của v về phía gốc qua các nút cha có giá trị khoá lớn hơn giá trị khoá của v. Trên đường đi ấy dồn nút cha xuống nút con, nút cha cuối cùng chính là vị trí mới của nút v (*hình 2*).

Chú ý: trên cây nhị phân, nếu đánh số các nút từ gốc đến lá và từ con trái sang con phải thì dễ thấy: khi biết số hiệu của nút cha là i có thể suy ra số hiệu hai nút con là 2*i và 2*i+1, ngược lại số hiệu nút con là j thì số hiệu nút cha là j div 2.



Hình 2

- Phép loại bỏ gốc của Heap

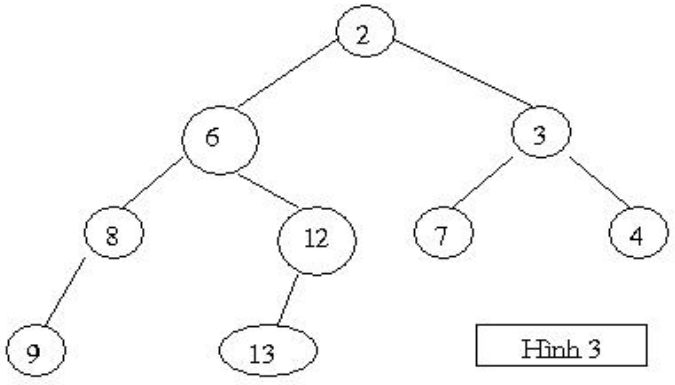
Vấn đề: Giả sử cần loại bỏ nút gốc khỏi Heap, hãy sắp xếp lại Heap (gọi là phép vun đống)

Giải quyết:

+ Tìm đường đi từ gốc về phía lá, đi qua các nút con có giá trị khoá nhỏ hơn trong hai nút con cho đến khi gặp lá.

+ Trên dọc đường đi ấy, kéo nút con lên vị trí nút cha của nó.

Ví dụ trong hình vẽ 2 nếu bỏ nút gốc có khoá bằng 1, ta sẽ kéo nút con lên vị trí nút cha trên đường đi qua các nút có giá trị khoá là 1, 2, 6, 8 và Heap mới như hình 3



Hình 3

Thuật toán Dijkstra tổ chức trên cấu trúc Heap (tạm kí hiệu là Dijkstra_Heap)

Tổ chức Heap: Heap gồm các nút là các đỉnh i tự do (chưa cố định nhãn đường đi ngắn nhất), với khoá là nhãn đường đi ngắn nhất từ s đến i là d[i].

Nút gốc chính là đỉnh tự do có nhãn d[i] nhỏ nhất. Mỗi lần lấy nút gốc ra để cố định nhãn của nó và sửa nhãn cho các đỉnh tự do khác thì phải thức hiện hai loại xử lí Heap đã nêu (phép cập nhật và phép loại bỏ gốc).

Vậy thuật toán Dijkstra tổ chức trên Heap như sau:
Cập nhật nút 1 của Heap (tương ứng với nút s có giá trị khoá bằng 0)
Vòng lặp cho đến khi Heap rỗng (không còn nút nào)
Begin
+ Lấy đỉnh u tại nút gốc của Heap (phép loại bỏ gốc Heap)
+ Nếu u= t thì thoát khỏi vòng lặp
+ Đánh dấu u là đỉnh đã được cố định nhãn
+ Duyệt danh sách cung kề tìm các cung có đỉnh đầu bằng u, đỉnh cuối là v
Nếu v là đỉnh tự do và d[v] > d[u] + khoảng cách (u,v) thì
Begin
Sửa nhãn cho v và ghi nhận đỉnh trước v là u
Trên Heap, cập nhật lại nút tương ứng với đỉnh v.
End;
End;

**** Đánh giá***

+ Thuật toán Dijkstra tổ chức như nêu ở mục 1. Có độ phức tạp thuật toán là $O(N^2)$, nên không thể thực hiện trên đồ thị có nhiều đỉnh.
+ Các phép xử lí Heap đã nêu (cập nhật Heap và loại bỏ gốc Heap) cần thực hiện không quá 2.lgM phép so sánh (nếu Heap có M nút). Số M tối đa là N (số đỉnh của đồ thị) và ngày càng nhỏ dần (tới 0). Ngoài ra, nếu đồ thị thưa (số cung ít) thì thao tác tìm đỉnh v kề với đỉnh u là không đáng kể khi ta tổ chức danh sách các cung kề này theo từng đoạn có đỉnh đầu giống nhau (dạng Forward Star). Do đó trên đồ thị thưa, độ phức tạp của Dijkstra_Heap có thể đạt tới O(N. k.lgN) trong đó k không đáng kể so với N
+ *Kết luận*: Trên đồ thị nhiều đỉnh ít cung thì Dijkstra_Heap là thực hiện được trong thời gian có thể chấp nhận.

III, Đường đi ngắn nhất giữa tất cả các cặp đỉnh - Thuật toán Floyd

Ta có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng *n* lần giải thuật đã Ford –Bellman hoặc Dijkstra , trong

đó ta sẽ chọn *s* lần lượt là các đỉnh của đồ thị. Khi đó ta sẽ thu được giải thuật với độ phức tạp là $O(n^4)$ (nếu sử dụng giải thuật Ford - Bellman) hoặc $O(n^3)$ đối với trường hợp đồ thị có trọng số không âm hoặc không có chu trình.

Trong trường hợp tổng quát việc sử dụng giải thuật Ford-Bellman *n* lần không phải là cách làm tốt nhất. Ở đây ta xét giải thuật Floyd giải bài toán trên với độ phức tạp tính toán $O(n^3)$.

Đầu vào là đồ thị cho bởi ma trận trọng số $a[i, j]$, $i, j = 1, 2, ..., n$.

Đầu ra : - Ma trận đường đi ngắn nhất giữa các cặp đỉnh: $d[i, j]$ ($i, j = 1, 2, ..., n$).

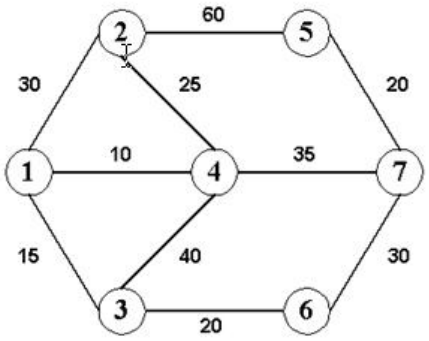
- Ma trận ghi nhận đường đi $tr[i, j]$ ($i, j = 1, 2, ..., n$) trong đó $tr[i, j]$ ghi nhận đỉnh đi trước đỉnh *j* trong đường đi ngắn nhất từ *i* đến *j*.

```
Procedure Floyd;
Var i, j, k : integer;
Begin
  { Khởi tạo }
  for i := 1 to n do
 for j := 1 to n do
  begin
    d[i, j] := a[i, j];
    tr[i, j] := i;
  end;
{ Bước lặp }
for k := 1 to n do
 for i := 1 to n do
   for j := 1 to n do
     if d[i, j] > d[i, k] + d[k, j] then
       begin
         d[i, j] := d[i, k] + d[k, j];
         tr[i, j] := tr[k, j];
       end;
End;
```

6, Một số bài toán tìm đường đi ngắn nhất

Bài toán 1: Hướng dẫn viên du lịch

Ông G. là một hướng dẫn viên du lịch. Công việc của ông ta là hướng dẫn một vài "tua" du lịch từ thành phố này đến thành phố khác. Trên các thành phố này, có một vài con đường hai chiều được nối giữa chúng. Mỗi cặp thành phố có đường kết nối đều có dịch vụ xe buýt chỉ chạy giữa hai thành phố này và chạy theo đường nối trực tiếp giữa chúng. Mỗi dịch vụ xe buýt đều có một giới hạn lớn nhất lượng khách mà xe buýt có thể trở được. Ông G có một tấm bản đồ chỉ các thành phố và những con đường nối giữa chúng. Ngoài ra, ông ta cũng có thông tin về mỗi dịch vụ xe buýt giữa các thành phố. Ông hiểu rằng ông không thể đưa tất cả các khách du lịch đến thành phố thăm quan trong cùng một chuyến đi. Lấy ví dụ: Về bản đồ gồm 7 thành phố, mỗi cạnh được nối giữa các thành phố biểu thị những con đường và các số viết trên mỗi cạnh cho biết cho biết giới hạn hành khách của dịch vụ xe buýt chạy trên tuyến đường đó.



Bây giờ, nếu ông G muốn đưa 99 khách du lịch từ thành phố 1 đến thành phố 7. Ông ta sẽ phải yêu cầu ít nhất là 5 chuyến đi, và lộ trình ông ta nên đi là 1 – 2 – 4 – 7.

Nhưng, Ông G. nhận thấy là thật khó để tìm ra tất cả lộ trình tốt nhất để sao cho ông ta có thể đưa tất cả khách du lịch đến thành phố thăm quan với số chuyến đi là nhỏ nhất. Do vậy mà ông ta cần sự trợ giúp của các bạn.

Dữ liệu: Vào từ file **Tourist.inp**

- Tệp **Tourist.inp** sẽ chứa một hay nhiều trường hợp test.
- Dòng đầu tiên trong mỗi trường hợp test chứa hai số nguyên N ($N \le 100$) và R mô tả lần lượt số thành phố và số đường đi giữa các thành phố.
- R dòng tiếp theo, mỗi dòng chứa 3 số nguyên: C1, C2, P. C1, C2 mô tả lộ trình đường đi từ thành phố C1 đến thành phố C2 và P ($P > 1$) là giới hạn lớn nhất có thể phục vụ của dịch vụ xe buýt giữa hai thành phố.

Các thành phố được đánh dấu bằng một số nguyên từ 1 đến N. Dòng thứ (R+1) chứa ba số nguyên S, D, T mô tả lần lượt thành phố khởi hành, thành phố cần đến và số khách du lịch được phục vụ.

Kết quả: Đưa ra file **Tourist.out**

Ghi ra số lộ trình nhỏ nhất cần phải đi qua các thành phố thỏa mãn yêu cầu đề bài.

Ví dụ: Tourist.inp

```
7 10
1 2 30
1 3 15
1 4 10
2 4 25
2 5 60
3 4 40
3 6 20
4 7 35
5 7 20
6 7 30
1 7 99
0 0
```

Tourist.out

```
5
```

Lời giải :

Đây là một bài toán hay, đòi hỏi các bạn phải nắm vững về thuật toán Dijkstra. Bài toán này là bài toán biến thể của bài toán kinh điển tìm đường đi ngắn nhất. Với con đường (u,v) gọi C[u, v] là số người tối đa có thể đi trên con đường đó trong một lần. C[u,v]-1 sẽ là số khách tối đa có thể đi trên con đường đó trong một lần. C[u,v] = 0 tương đương với giữa u và v không có con đường nào. Gọi D[i] là số khách nhiều nhất có thể đi 1 lần từ điểm xuất phát đến i. Với mỗi đỉnh j kề với i, ta cập nhật lại D[j] = min(D[i], C[i, j]). Số khách có thể đi cùng một lúc từ điểm xuất phát tới điểm kết thúc T là D[T]. Một chú ý nữa là khi tính số lần đi, các bạn chỉ cần dùng các phép div, mod để tính.

Chương trình thể hiện thuật toán trên (độ phức tạp: n^2)

```
{$R+,Q+}
  const
    INP = 'tourist.inp';
    OUT = 'tourist.out';
    maxn = 100;
  var
    fi, fo: text;
```

```
  c: array [1..maxn, 1..maxn] of longint;
  d: array [1..maxn] of longint;
  chua: array [1..maxn] of boolean;
  n, m, s, t, w: longint;
procedure open_file;
begin
  assign(fi, INP);  reset(fi);
  assign(fo, OUT);  rewrite(fo);
end;
procedure close_file;
begin
  close(fi);
  close(fo);
end;
procedure read_data;
var i, u, v, x: longint;
begin
  readln(fi, n, m);
  for i := 1 to m do
    begin
      readln(fi, u, v, x);
      c[u, v] := x - 1;
      c[v, u] := x - 1;
    end;
  readln(fi, s, t, w);
end;
function min2(x, y: longint): longint;
begin
  if x > y then min2 := y else min2 := x;
end;
procedure process;
var
  i, max, last: longint;
begin
  fillchar(chua, sizeof(chua), true);
```

```
  fillchar(d, sizeof(d), 0);
  chua[s] := false;
  last := s;
  d[s] := maxlongint;
Khởi tạo d[s] = vô cùng hay tất cả mọi người đều có thể đến S cùng lúc
  while chua[t] do
    begin
      for i := 1 to n do
{Tìm các đỉnh i kề với last để cập nhật lại}
        if chua[i] and (d[i] < min2(c[last, i], d[last])) then
          d[i] := min2(c[last, i], d[last]);
      max := -1;
      for i := 1 to n do
        if chua[i] and (d[i] > max) then
          begin
            max := d[i];
            last := i;
          end;
      chua[last] := false;
    end;
end;
procedure write_result;
begin
  if w mod d[t] = 0 then
    writeln(fo, w div d[t])
  else
    writeln(fo, w div d[t] + 1);
end;
begin
  open_file;
  read_data;
  process;
  write_result;
  close_file;
end.
```

Bài 2: Chuyển Hàng

Bản đồ một kho hàng hình chữ nhật kích thước mxn được chia thành các ô vuông đơn vị (m hàng, n cột: các hàng đánh số từ trên xuống dưới, các cột được đánh số từ trái qua phải). Trên các ô vuông của bản đồ có một số ký hiệu:

- Các ký hiệu # đánh dấu các ô đã có một kiện hàng xếp sẵn.
- Một ký hiệu *: Đánh dấu ô đang có một rôbốt.
- Một ký hiệu $: Đánh dấu ô chứa kiện hàng cần xếp.
- Một ký hiệu @: Đánh dấu vị trí mà cần phải xếp kiện hàng vào $ vào ô đó.
- Các ký hiệu dấu chấm ".": Cho biết ô đó trống.

Tại môt thời điểm, rô bốt có thể thực hiện một trong số 6 động tác ký hiệu là:

- L, R, U, D: Tương ứng với phép di chuyển của rô bốt trên bản đồ: sang trái, sang phải, lên trên, xuống dưới. Thực hiện một phép di chuyển mất 1 công
- +, − : Chỉ thực hiện khi rôbốt đứng ở ô bên cạnh kiện hàng $. Khi thực hiện thao tác +, rôbốt đứng yên và đẩy kiện hàng $ làm kiện hàng này trượt theo hướng đẩy, đến khi chạm một kiện hàng khác hoặc tường nhà kho thì dừng lại. Khi thực hiện thao tác − , rô bốt kéo kiện hàng $ về phía mình và lùi lại 1 ô theo hướng kéo. Thực hiện thao tác đẩy hoặc kéo mất C công. Rô bốt chỉ được di chuyển vào ô không chứa kiện hàng của kho.

Hãy tìm cách hướng dẫn rôbốt thực hiện các thao tác để đưa kiện hàng $ về vị trí @ sao cho số công phải dùng là ít nhất.

Dữ liệu: Vào từ file văn bản CARGO.INP

- Dòng 1: Ghi ba số nguyên dương m, n, C (m, n ≤ 100; C ≤ 100)
- m dòng tiếp theo, dòng thứ i ghi đủ n ký kiệu trên hàng i của bản đồ theo đúng thứ tự trái qua phải.

Các ký hiệu được ghi liền nhau.

Kết quả: Ghi ra file văn bản CARGO.OUT

- Dòng 1: Ghi số công cần thực hiện
- Dòng 2: Một dãy liên tiếp các ký tự thuộc {L, R, U, D, +, -} thể hiện các động tác cần thực hiện của rô bốt.

Rằng buộc: Luôn có phương án thực hiện yêu cầu đề bài.

Ví dụ:

CARGO.INP	CARGO.OUT
6 8 3 ###. .### *$ . . . .## #### .### #### . .## #@. . . .## ########	23 +RRRR-UR+DDDRD+

Phân tích:

Thuật toán: Ta sẽ dùng thuật toán Dijkstra để giải bài toán này.

* **Mô hình đồ thị:**

Mỗi đỉnh của đồ thị ở đây gồm 3 trường để phân biệt với các đỉnh khác:

- i: Tọa độ dòng của kiện hàng (i = 1..m)
- j: Tọa độ cột của kiện hàng (j = 1..n)
- h: Hướng của rô bốt đứng cạnh kiện hàng so với kiện hàng (h = 1..4: Bắc, Đông, Nam, Tây).

	1	
2	$	3
	4	

Bạn có thể quan niệm mỗi đỉnh là (i,j,u,v): trong đó i,j: tọa độ của kiện hàng; u,v: tọa độ của rôbốt đứng cạnh kiện hàng. Nhưng làm thế sẽ rất lãng phí bộ nhớ và không chạy hết được dữ liệu. Ta chỉ cần biết hướng h của rôbốt so với kiện hàng là có thể tính được tọa độ của rôbốt bằng cách dùng 2 hằng mảng lưu các số ra:

dx : array[1..4] of integer = (-1,0,1,0)

dy : array[1..4] of integer = (0,1,0,-1)

Khi đó, tọa độ(u,v) của rôbốt sẽ là : u := i + dx[h]; v := j + dy[h];

- Hai đỉnh (i1,j1,h1) và (i2,j2,h2) được gọi là kề nhau nếu qua 1 trong 2 thao tác + hoặc - kiện hàng được rôbốt đẩy hoặc kéo từ ô (i1, j1) đến ô (i2, j2) và rôbốt có thể di chuyển được từ ô (u1,v1) đến ô (u2,v2) (u1 = i1+dx[h1]; v1=j1+dy[h1]; u2=i2+dx[h2]; v2= j2+dy[h2]). Tất nhiên các ô (i2,j2) và (u2,v2) phải đều không chứa kiện hàng.
- Trọng số giữa 2 đỉnh là C (số công mà rô bốt đẩy kiện hàng từ ô (i1,j1) đến ô (i2,j2)) cộng với công để rô bốt di chuyển từ ô (u1,v1) đến ô (u2,v2).

Giả sử kiện hàng cần xếp đang ở ô (is,js) và hướng của rôbốt đứng cạnh kiện

hàng là hs và ô cần xếp kiện hàng vào là ô (ie, je). Khi đó, ta sẽ dùng thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh (is,js,hs) đến đỉnh (ie,je,he) với he thuộc {1..4}.

Mảng d sẽ là 1 mảng 3 chiều: d[i,j,h]: Độ dài đường đi ngắn nhất từ đỉnh xuất phát (is,js,hs) đến đỉnh (i,j,h). Kết quả của bài toán sẽ là d[ie,je,he] với he thuộc {1..4}.

Để ghi nhận phương án ta sẽ dùng 3 mảng 3 chiều tr1, tr2, tr3. Khi ta di từ đỉnh (i1,j1,h1) đến đỉnh (i2,j2,h2) thì ta sẽ gán: tr1[i2,j2,h2]:= i1; tr2[i2,j2,h2]:= j1; tr3[i2,j2,h2] := h1 để ghi nhận các thông tin: tọa độ dòng, cột, hướng của dỉnh trước đỉnh (i2,j2,h2). Từ 3 mảng này ta có thể dễ dàng lần lại đường đi.

Bài 3: Ông Ngâu bà Ngâu

Hẳn các bạn đã biết ngày "ông Ngâu bà Ngâu" hàng năm, đó là một ngày đầy mưa và nước mắt. Tuy nhiên, một ngày trước đó, nhà Trời cho phép 2 "ông bà" được đoàn tụ. Trong vũ trụ vùng thiên hà nơi ông Ngâu bà Ngâu ngự trị có N hành tinh đánh số từ 1 đến N, ông ở hành tinh Adam (có số hiệu là S) và bà ở hành tinh Eva (có số hiệu là T). Họ cần tìm đến gặp nhau.

N hành tinh được nối với nhau bởi một hệ thống cầu vồng. Hai hành tinh bất kỳ chỉ có thể không có hoặc duy nhất một cầu vồng (hai chiều) nối giữa chúng. Họ luôn đi tới mục tiêu theo con đường ngắn nhất. Họ đi với tốc độ không đổi và nhanh hơn tốc độ ánh sáng. Điểm gặp mặt của họ chỉ có thể là tại một hành tinh thứ 3 nào đó.

Yêu cầu: Hãy tìm một hành tinh sao cho ông Ngâu và bà Ngâu cùng đến đó một lúc và thời gian đến là sớm nhất. Biết rằng, hai ngươời có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau.

Dữ liệu Trong file văn bản ONBANGAU.INP gồm

Dòng đầu là 4 số N M S T ($N \le 100$, $1 \le S \ne T \le N$), M là số cầu vồng. M dòng tiếp, mỗi dòng gồm hai số I J L thể hiện có cầu vồng nối giữa hai hành tinh I , J và cầu vồng đó có độ dài là L ($1 \le I \ne J \le N$, $0 < L \le 200$).

Kết quả Ra file văn bản ONBANGAU.OUT, do tính chất cầu vồng, mỗi năm một khác, nên nếu nhưư không tồn tại hành tinh nào thoả mãn yêu cầu thì ghi ra một dòng chữ CRY. Nếu có nhiều hành tinh thoả mãn thì ghi ra hành tinh có chỉ số nhỏ nhất.

Ví dụ:

ONBANGAU.INP	ONBANGAU.OUT
4 4 1 4 1 2 1 2 4 1 1 3 2 3 4 2	2

Tư tưởng thuật toán:

Chúng ta có một số nhận xét sau:

+ Hai hành tinh bất kì chỉ được nối đến nhau bởi nhiều nhất một cầu vồng
+ Ông Ngâu và bà Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất
+ Họ đi với vận tốc không đổi và nhanh hơn vận tốc ánh sáng

Thực chất đây là một bài toán đồ thị: Từ hành tinh S(nơi ông Ngâu ở) ta xây dựng bảng SP. Trong đó SP[i] là đường đi ngắn nhất từ hành tinh S đến hành tinh i (do ông Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất). SP[i] = 0 tức là không có đường đi từ hành tinh S đến hành tinh i. Tương tự ta sẽ xây dựng bảng TP, trong đó TP[i] là đường đi ngắn nhất từ hành tinh T đến hành tinh i. Và TP[i] = 0 tức là không có đường đi từ hành tinh T đến hành tinh i. Do yêu cầu của bài toán là tìm hành tinh khác S và T mà 2 ông bà Ngâu cùng đến một lúc và trong thời gian nhanh nhất. Tức là ta sẽ tìm hành tinh h sao cho (h khác S và T) và(SP[h] = ST[h]) đạt giá trị nhỏ nhất khác 0. Nếu không có hành tinh h nào thoả mãn thì ta thông báo CRY

Để xây dựng mảng SP và ST ta có rất nhiều giải thuật khác nhau. ở đây ta chọn giải thuật Djkstra tìm đường đi ngắn nhất giữa 2 đỉnh đồ thị.

Chương trình như sau:

```
uses crt;
const MaxN = 101;
fi= 'ONBANGAU.inp';
fo= 'ONBANGAU.out';
var n,m,s,t,h:byte;
a:array[0..MaxN,0..MaxN] of byte;
SP,ST,B:array[0..MaxN] of integer;
f:text;
{*-------------*thnt*------------*}
procedure Init;
var i,u,v,ts:byte;
begin
```

```
fillchar(a,sizeof(a),0);
assign(f,fi);
reset(f);
readln(f,n,m,s,t);
for i:=1 to m do
begin
readln(f,u,v,ts);
a[u,v]:=ts;
a[v,u]:=ts;
end;
close(f);
end;
{*----------*thnt*----------*}
procedure Build(s:byte);
var tt:array[0..maxN] of byte;
min,i,vtr:integer;
begin
fillchar(tt,sizeof(tt),0);
fillchar(b,sizeof(b),0);
for i:=1 to n do
b[i] := a[s,i];
tt[s]:=1;
min:=0;
while min <> maxint do
begin
min:=maxint; vtr:=0;
for i:=1 to n do
if tt[i] = 0 then
if (b[i] <>0) and (b[i]
begin min:=b[i]; vtr:=i; end;
if vtr <> 0 then tt[vtr]:=1;

for i:=1 to n do
if (tt[i] = 0) then
if a[vtr,i] <> 0 then
```

```
if (b[vtr] + a[vtr,i]
b[i]:=b[vtr] + a[vtr,i];
end;
end;
{*-------------*thnt*------------*}
procedure FindWay;
var i:integer;
begin
build(s); {xay dung mang SP }
SP:=B;
build(t); {xay dung mang ST}
ST:=B;
h:= 0;{hanh tinh can tim}
sp[0]:= Maxint;
for i:=1 to n do
if (SP[i] = ST[i]) then
if (SP[i]<>0) then
if (SP[i] < SP[h]) then
h:=i;
end;
{*-------------*thnt*------------*}
procedure ShowWay;
begin
assign(f,fo);
rewrite(f);
if h <> 0 then writeln(f,h)
else writeln(f,'CRY');
close(f);
end;
{*-------------*thnt*------------*}
begin
Init;
FindWay;
ShowWay;
end.
```

Bài 4: Máy chủ

Một Công ty muốn phát triển một hệ thống mạng máy tính lớn bao gồm các máy chủ cung cấp nhiều loại hình dịch vụ khác nhau.

Mạng dược kết nối từ n máy chủ bở các kênh nối cho phép truyền tin hai chiều. Hai máy chủ có thể được nối trực tiếp với nhau bởi không quá một kênh nối. Mỗi máy chủ được nối trực tiếp với không quá 10 máy chủ khác và hai máy chủ bất kỳ luôn có thể kết nối được với nhau hoặc thông qua một kênh nối trực tiếp giữa chúng hoặc thông qua các máy chủ khác. Đối với kênh nối ta biết thời gian truyền thông được đo bằng mili giấy là một số dương. Khoảng cách (tính bằng mili giấy) d(u,v) giữa máy chủ u và máy chủ v được xác định như là độ dài của đường đi ngắn nhất (ứng với thời gian truyền thông trên cạnh) nối u và v trên mạng. Để tiện dùng, ta qui ước d(v,v)=0 với mọi v.

Có một số máy chủ cung cấp nhiều dịch vụ hơn các máy chủ khác. Vì thế mỗi máy chủ v được gán với một số tự nhiên r(v) gọi là hạng của nó. Máy chủ có hạng càng cao càng là máy chủ mạnh hơn. Tại mỗi máy chủ dữ liệu về các máy chủ gần nó thường được cất giữ. Tuy nhiên không phải máy chủ nào cũng là đáng quan tâm. Dữ liệu về các máy chủ lân cận với hạng thấp hơn không được cất giữ. Chính xác hơn, *máy chủ w là dáng quan tâm đối với máy chủ v* nếu với mọi máy chủ u sao cho d(v,u)≤d(v,w) ta có r(u)≤r(w). Chẳng hạn, tất các các máy chủ với hạng lớn nhất đều là đáng quan tâm đối với tất cả các máy chủ. Nếu máy chủ v có hạng lớn nhất thì rõ ràng chỉ có các máy chủ với hạng lớn nhất mới là đáng quan tâm đối với v.

Gọi B(v) là tập các máy chủ đáng quan tâm đối với máy chủ v. Ta gọi *kích thước dữ liệu về các máy chủ đáng quan tâm đối với máy chủ v là* $|B(v)|$.

Yêu cầu: Tính tổng kích thước dữ liệu về các máy chủ đáng quan tâm của tất cả các máy chủ trong toàn mạng. Biết rằng tổng này có giá trị không vượt quá 30n.

Dữ liệu: Vào từ file văn bản SERVER.INP:

+ Dòng đầu chứa hai số n, m trong đó n là số máy chủ (1≤n≤3000) và m là số kênh nối (1≤m≤5n)

+ Dòng thứ i trong số n dòng tiếp theo chứa r_i (1≤r_i≤10) là hạng của máy chủ i.

+ Tiếp đến là m dòng, mỗi dòng chứa thông tin về một kênh nối bao gồm a, b, t (1≤t≤1000, 1≤a,b≤n, a≠b), trong đó a, b là chỉ số của hai máy chủ

được nối bởi kênh đang xét còn t là thời gian truyền thông của kênh (đo bằng mili giây).

Kết quả: Ghi ra file văn bản SERVER.OUT một số nguyên duy nhất là tổng kích thước dữ liệu về các máy chủ đáng quan tâm của tất cả các máy chủ trong toàn mạng

Ví dụ:

SERVER.INP	SERVER.OUT	Giải thích
4 3 2 3 1 1 1 4 30 2 3 20 3 4 20	9	Ta có: B(1)={1,2} B(2)={2} B(3)={2,3} B(4)={1,2,3,4}

Bài này ta dùng thuật toán Dijkstra để giải chương trình như sau

```
const
  tfi          =    'SERVER.INP';
  tfo          =    'SERVER.OUT';
  maxN         =    3000;
type
  mang1        =    array[1..10] of integer;
  mang2        =    array[1..maxN] of integer;
var
  fi,fo        :    text;
  N,M          :    longint;
  Sol          :    array[1..maxN] of byte;
  r            :    array[1..maxN] of byte;
  a            :    array[1..maxN] of ^mang1;
  d            :    array[1..maxN] of ^mang1;
  kq           :    longint;
  Q            :    array[1..maxN] of longint;
  vt           :    ^mang2;
  qn           :    longint;
  kc           :    array[1..maxN] of longint;
```

```
  loai        :  array[1..maxN] of byte;
  Rank        :  array[1..maxN] of byte;
  Q1          :  ^mang2;
  q1n         :  integer;
  kcold       :  longint;
  t1          :  longint;
  t2          :  longint absolute 0:$46c;
  rmax        :  byte;

procedure CapPhat;
var i: longint;
begin
  for i:=1 to maxN do new(a[i]);
  for i:=1 to maxN do new(d[i]);
  new(q1);
  new(vt);
end;
procedure InitQ;
begin
  qn:=0;
end;
procedure Put(u: longint);
begin
  inc(qn);
  q[qn]:=u;
end;
function Get: longint;
var i,u: longint;
begin
  u:=1;
  for i:=2 to qn do
    if kc[q[i]]<kc[q[u]] then u:=i;
    Get:=q[u];
    q[u]:=q[qn];
    dec(qn);
```

```
end;
function Qempty: boolean;
begin
  Qempty:=(qn=0);
end;
procedure Docdl;
var i,u,v,w:longint;
begin
  assign(fi,tfi); reset(fi);
  readln(fi,n,m);
  for i:=1 to N do sol[i]:=0;
  for i:=1 to N do readln(fi,r[i]);
  for i:=1 to M do
    begin
      readln(fi,u,v,w);
      inc(sol[u]); a[u]^[sol[u]]:=v; d[u]^[sol[u]]:=w;
      inc(sol[v]); a[v]^[sol[v]]:=u; d[v]^[sol[v]]:=w;
    end;
  close(fi);
  rmax:=0;
  for i:=1 to N do
    if rmax<r[i] then rmax:=r[i];
end;

procedure Dijstra(xp: longint);
var i,u,v,ll: longint;
   MaxRank: byte;
begin
  InitQ;
  kcold:=-1;
  for i:=1 to N do loai[i]:=0;
  for i:=1 to N do rank[i]:=rmax;
  MaxRank:=0;
  Put(xp); loai[xp]:=1; kc[xp]:=0;
  repeat
```

```
    u:=Get; loai[u]:=2;
    if MaxRank<r[u] then MaxRank:=r[u];
    if kc[u]=kcold then
      begin
        inc(q1n);
        q1^[q1n]:=u;
      end
    else
      begin
        q1n:=1;
        q1^[1]:=u;
      end;
    kcold:=kc[u];
    for i:=1 to q1n do
      Rank[q1^[i]]:=MaxRank;
    if (maxRank<rmax) then
    for i:=1 to sol[u] do
      begin
        v:=a[u]^[i]; ll:=d[u]^[i];
        if (loai[v]=1) and (kc[v]>kc[u]+ll) then kc[v]:=kc[u]+ll;
        if loai[v]=0 then
          begin
            Loai[v]:=1;
            kc[v]:=kc[u]+ll;
            Put(v);
          end;
      end;
  until Qempty;
end;
function Dem: longint;
var k,i: longint;
begin
  k:=0;
  for i:=1 to N do
    if (Rank[i]<=r[i]) then k:=k+1;
```

```
    Dem:=k;
  end;
  procedure Solve;
  var i,k: longint;
  begin
    kq:=0;
    for i:=1 to N do
      begin
        Dijstra(i);
        kq:=kq+Dem;
      end;
  end;
  procedure Inkq;
  begin
    assign(fo,tfo); rewrite(fo);
    writeln(fo,kq);
    close(fo);
  end;
  BEGIN
    clrscr;
    t1:=t2;
    CapPhat;
    Docdl;
    Solve;
    Inkq;
    writeln('Total time =',(t2-t1)/18.3:0:4,' s');
    readln;
END.
```

Bài 5: Hành trình trên xe lửa

Lịch hoạt động của tuyến đường sắt trong một ngày bao gồm thông tin của từng chuyến tầu có trong ngày đó. Thông tin của mỗi chuyến tầu bao gồm:

- Số hiệu chuyến tầu (được đánh số từ 1 đến *M*),
- Danh sách các ga mà chuyến tầu đó dừng lại, mỗi ga bao gồm:
  + Số hiệu ga (các ga được đánh số từ 1 trở đi),
  + Giờ đến (số thực),

+ Giờ đi (số thực).

Các giá trị thời gian tính theo đơn vị giờ và viết dưới dạng thập phân (ví dụ 7.5 có nghĩa là 7 giờ 30 phút).

Một hành khách bất kỳ khi đi đến một ga nào đó (gọi là ga hiện tại) cho biết yêu cầu của mình gồm: thời điểm mà từ đó anh ta có thể đi được, số hiệu ga cần đến và thời gian tối thiểu cho mỗi lần chuyển tầu. Nhân viên nhà ga phải trả lời được là có đáp ứng được yêu cầu của khách không? Nếu đáp ứng được, nhân viên nhà ga phải đưa ra được hành trình cần đi cho khách.

Hãy giải bài toán trong 2 trường hợp:

a. Tìm hành trình đến ga cuối cùng sớm nhất.

b. Tìm hành trình ít phải chuyển tầu nhất. Nếu tồn tại nhiều phương án như vậy, hãy tìm phương án đến ga cuối cùng sớm nhất.

Dữ liệu: File vào gồm các dòng:

- Dòng 1: Ghi 4 số theo thứ tự: thời điểm đi, ga hiện tại, ga cần đến và thời gian tối đa cho mỗi lần chuyển tầu;
- Dòng 2: Ghi số nguyên dương M ($M \le 50$);
- Dòng i+2 (i = 1, 2, ..., M): Ghi thông tin của chuyến tầu số hiệu i bao gồm: số lượng ga mà chuyến tầu đó dừng lại (≤ 20), danh sách các ga theo trình tự đi đến của chuyến tầu, trong đó mỗi ga được mô tả bởi 3 số theo thứ tự: số hiệu ga, giờ đến, giờ đi.

Các số trên cùng một dòng ghi cách nhau bởi một dấu trắng.

Kết quả: Trong trường hợp không tìm thấy hành trình thì ghi giá trị 0. Trái lại, ghi hành trình tìm được dưới dạng sau:

- Dòng đầu ghi S là số hiệu chuyến tầu mà khách bắt đầu đi,
- Dòng tiếp ghi T_1 là thời điểm đi của chuyến tầu này,
- Dòng tiếp ghi K là số lần khách phải chuyển tầu,
- K dòng tiếp, mỗi dòng ghi thông tin của một lần chuyển tầu gồm số hiệu ga mà khách phải chuyển tầu và số hiệu chuyến tầu cần đi tiếp (ghi cách nhau một dấu trắng),
- Dòng cuối ghi T_2 là thời điểm đến ga cuối cùng của hành trình.

Kết quả của câu a và câu b ghi cách nhau bởi 1 dòng trắng.

Ví dụ:

XELUA.INP	XELUA.OUT
6 4 3 1.5 6 3 1 7 7 2 8 9.1 3 9.5 9.5 2 4 6 6 2 7 7.5 2 2 7.5 7.5 5 8 8 3 6 8 8 5 9 9.5 3 10 10 3 4 6.5 6.5 7 9 9.5 3 11 11 2 4 7 7 3 12 12	26.02 2 3 5 4 10.0 5 6.5 0 11.0

Chương trình như sau

```
Const
  FI = 'xelua.inp';
  FO = 'xelua.out';
  MAX_VALUE = 999999999;
Var
  n, nU, ga_di, ga_den, dem : integer;
  t0, t_di, t_cho : real;
  tau, ga, tr, U : array[0..1001] of integer;
  d, gio_den, gio_di : array[0..1001] of real;
  f : text;
Procedure Doc;
Var m, i, j, k : integer;
Begin
  assign(f, FI); reset(f);
  read(f, t_di, ga_di, ga_den, t_cho, m);
  tau[0] := 0;
  ga[0] := ga_di;
  gio_den[0] := t_di;
  gio_di[0] := t_di;
  n := 0;
  for i := 1 to m do
    begin
```

```
    read(f, k);
    for j := 1 to k do
     begin
       n := n + 1;
       tau[n] := i;
       read(f, ga[n], gio_den[n], gio_di[n]);
     end;
   end;
  close(f);
End;
Function Khoang_cach(i, j : integer) : real;
Var t : real;
Begin
  if tau[i] = tau[j] then
    begin
      t := gio_di[i] - gio_den[i];
      if (j = i+1) and (t <= t_cho) then Khoang_cach := gio_den[j] - gio_den[i]
                                    else Khoang_cach := MAX_VALUE;
    end
  else
    if ga[i] = ga[j] then
      begin
        t := gio_di[j] - gio_den[i];
        if (t >= 0) and (t <= t_cho) then Khoang_cach := t + t0
                                     else Khoang_cach := MAX_VALUE;
      end
    else Khoang_cach := MAX_VALUE;
End;
Procedure Khoi_tao;
Var i : integer;
Begin
  for i := 0 to n do
    begin
      d[i] := Khoang_cach(0, i);
      tr[i] := 0;
```

```
    end;
  nU := n;
  for i := 1 to nU do U[i] := i;
End;
Function Co_dinh_nhan : integer;
Var i, j : integer;
Begin
  i := 1;
  for j := 2 to nU do
    if d[U[j]] < d[U[i]] then i := j;
  Co_dinh_nhan := U[i];
  U[i] := U[nU];
  nU := nU - 1;
End;
Procedure Sua_nhan(p : integer);
Var
  x, i : integer;
  kc : real;
Begin
  for i := 1 to nU do
    begin
      x := U[i];
      kc := Khoang_cach(p, x);
      if d[x] > d[p] + kc then
        begin
          d[x] := d[p] + kc;
          tr[x] := p;
        end;
    end;
End;
Procedure Print(i : integer);
Begin
  if tr[i] = 0 then
   begin
     writeln(f, tau[i]);
```

```
    writeln(f, gio_di[i] : 0 : 1);
    writeln(f, dem);
    exit;
   end;
  if tau[tr[i]] <> tau[i] then dem := dem + 1;
  Print(tr[i]);
  if tau[tr[i]] <> tau[i] then writeln(f, ga[i], ' ', tau[i]);
End;
Procedure Ghi;
Var
  dich, i : integer;
  som_nhat : real;
Begin
  som_nhat := MAX_VALUE;
  for i := 1 to n do
    if (ga[i] = ga_den) and (d[i] < som_nhat) then
      begin
        som_nhat := d[i];
        dich := i;
      end;
  if som_nhat = MAX_VALUE then writeln(f, 0)
  else
    begin
      dem := 0;
      Print(dich);
      writeln(f, gio_den[dich] : 0 : 1);
    end;
  writeln(f);
End;
Procedure Dijktra;
Var p : integer;
Begin
  Khoi_tao;
  while nU > 0 do
    begin
```

```
        p := Co_dinh_nhan;
        Sua_nhan(p);
      end;
    Ghi;
  End;
  Procedure Xu_ly;
  Begin
    assign(f, fo); rewrite(f);
    { Cau a }
    t0 := 0;
    Dijktra;
    { Cau b }
    t0 := 9999;
    Dijktra;
    close(f);
  End;
  Begin
    Doc;
    Xu_ly;
End.
```

Bài 6: Hội thảo trực tuyến

Một trung tâm quản trị một mạng gồm N (≤ 100) cổng truy cập được đánh số từ 1 đến N. Giữa hai cổng có thể không có đường nối hoặc có đường nối trực tiếp và thông tin truyền hai chiều trên đường nối. Mạng có M đường nối trực tiếp giữa các cổng và nếu đường nối trực tiếp giữa hai cổng i, j được sử dụng thì chi phí truyền tin phải trả là c_{ij} (≤ 32767).

Trung tâm nhận được hợp đồng tổ chức một cuộc hội thảo trực tuyến từ 3 địa điểm khác nhau truy cập vào mạng từ 3 cổng. Bạn hãy giúp công ty tổ chức sử dụng các đường nối truyền tin sao cho tổng chi phí là ít nhất có thể được.

Dữ liệu: File vào gồm các dòng:

- Dòng đầu tiên ghi hai số N và M;
- M dòng tiếp theo, mỗi dòng chứa 3 số nguyên dương trong đó 2 số đầu là chỉ số của hai cổng, số thứ 3 là chi phí khi truyền tin trên hai cổng đó;

- Dòng cuối cùng chứa 3 số nguyên dương theo thứ tự là chỉ số của 3 cổng tại 3 địa điểm của hội thảo.

Kết quả: File ra gồm:

- Dòng đầu ghi xâu 'No' nếu không thể tổ chức hội thảo trực tuyến được, ngược lại ghi 'Yes'.
- Nếu tìm được cách tổ chức thì dòng thứ hai ghi *S* là chi phí nhỏ nhất tìm được và dòng thứ ba ghi *P* là số đường nối cần sử dụng. *P* dòng tiếp theo mỗi dòng ghi hai số *i*, *j* thể hiện một đường nối giữa hai cổng *i* và *j* được sử dụng. Các số trên một dòng ghi cách nhau bởi một dấu cách.

Ví dụ:

NET.INP	NET.OUT
8 12 1 2 20 2 3 8 2 4 3 2 5 3 2 6 6 3 5 2 3 6 9 4 7 5 5 6 1 5 7 7 6 8 4 7 8 6 1 4 6	Yes 27 4 1 2 2 4 2 5 5 6

Lời giải: Chúng ta thấy rằng chắc chắn đoạn nối đó phải là một cây . Tức là sẽ có một cây đồ thị bao lấy ba địa điểm đó . Mà cây đó là cây có độ dài nhỏ nhất . Vì vậy tồn tại một điểm là trung gian T (có thể trùng với 1 trong ba địa điểm đó) . Thì tổng đường truyền từ T đến 3 đỉnh đó phải nhỏ nhất . Tức là ta sẽ dùng thuật toán Floyd . Sau đó tìm đỉnh nào có tổng khoảng cách nhỏ nhất đến ba đỉnh làn nhỏ nhất thì các đường nối đó chính là các đường nối thoả mãn .

Chương trình

```
Program Hoi_thao_truc_tuyen;
Uses crt;
Const
  FI = 'net.inp';
  FO = 'net.out';
  MAX_N = 100;
  MAX_VALUE = 999999999;
Var
  n, x, y, z, sum, so_canh : integer;
  c : array[1..MAX_N, 1..MAX_N] of longint;
  tr : array[1..MAX_N, 1..MAX_N] of byte;
  f : text;
Procedure Doc;
Var m, chi_phi, i, j, k : integer;
Begin
  assign(f, FI); reset(f);

  readln(f, n, m);
  for i := 1 to n do
    for j := 1 to n do c[i, j] := MAX_VALUE;
  for k := 1 to m do
    begin
      readln(f, i, j, chi_phi);
      c[i, j] := chi_phi;
      c[j, i] := chi_phi;
    end;
  readln(f, x, y, z);
  close(f);
End;
Procedure Floyd;
Var i, j, k : integer;
Begin
  for i := 1 to n do
    for j := 1 to n do tr[i, j] := i;
```

```
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if c[i, j] > c[i, k] + c[k, j] then
          begin
            c[i, j] := c[i, k] + c[k, j];
            tr[i, j] := tr[k, j];
          end;
End;
Procedure Print(i, j : integer);
Begin
  if i = j then exit;
  if c[tr[i, j], j] <> -1 then
    begin
      so_canh := so_canh + 1;
      sum := sum + c[tr[i, j], j];
      c[tr[i, j], j] := -1;
      c[j, tr[i, j]] := -1;
    end;
  Print(i, tr[i, j]);
End;
Procedure Ghi;
Var min, t, i, j : longint;
Begin
  assign(f, FO); rewrite(f);
  min := MAX_VALUE;
  for i := 1 to n do
    if min > c[x, i] + c[y, i] + c[z, i] then
      begin
        t := i;
        min := c[x, i] + c[y, i] + c[z, i];
      end;
  if min = MAX_VALUE then write(f, 'No')
  else
    begin
```

```
    so_canh := 0;
    sum := 0;
    Print(x, t);
    Print(y, t);
    Print(z, t);
    writeln(f, 'Yes');
    writeln(f, sum);
    writeln(f, so_canh);
    for i := 1 to n do
      for j := i+1 to n do
        if c[i, j] = -1 then writeln(f, i, ' ', j);
   end;
  close(f);
End;
Begin
  Doc;
  Floyd;
  Ghi;
End.
```

Bài 7: Chợ trung tâm

Có *N* địa điểm dân cư đánh số từ 1 đến *N*. Giữa *M* cặp địa điểm trong số *N* địa điểm nói trên có tuyến đường nối chúng. Cần xây dựng một trung tâm dịch vụ tổng hợp tại một địa điểm trùng với một địa điểm dân cư, sao cho tổng khoảng cách từ trung tâm dịch vụ đến *N* địa điểm dân cư là nhỏ nhất. Ta gọi khoảng cách giữa hai địa điểm là độ dài đường đi ngắn nhất nối chúng. Giả sử *N* địa điểm trên là liên thông với nhau. Nếu có nhiều phương án thì đưa ra phương án đặt trung tâm dịch vụ tại địa điểm có số hiệu nhỏ nhất.

Dữ liệu: File vào gồm *M*+1 dòng:

- Dòng 1: Chứa hai số nguyên dương *N* và *M* (*N* ≤ 100);
- Dòng *i*+1 (1 ≤ *i* ≤ *M*): Chứa 3 số nguyên dương *x*, *y*, *z*, ở đó hai số đầu *x*, *y* là số hiệu của hai địa điểm dân cư được nối với nhau bởi tuyến đường này, còn số thứ ba *z* (≤ 32767) là độ dài của tuyến đường này.

Kết quả: File ra gồm 2 dòng:

- Dòng 1: Ghi vị trí trung tâm dịch vụ;
- Dòng 2: Ghi tổng khoảng cách từ trung tâm dịch vụ đến các địa điểm dân cư.

Ví dụ:

MARKET.INP	MARKET.OUT
5 7 1 2 9 2 3 4 1 4 2 4 5 5 5 3 1 5 1 5 3 1 4	3 15

```
Program Cho_trung_tam;
Uses crt;
Const
  FI = 'market.inp';
  FO = 'market.out';
  MAX_N = 100;
  MAX_VALUE = 999999999;
Var
  n, dia_diem, min : longint;
  d : array[1..MAX_N, 1..MAX_N] of longint;
  f : text;
Procedure Doc;
Var i, j, k, m : integer;
Begin
  assign(f, FI); reset(f);

  read(f, n, m);
  for i := 1 to n do
    begin
      d[i, i] := 0;
      for j := i+1 to n do
        begin
          d[i, j] := MAX_VALUE;
          d[j, i] := MAX_VALUE;
```

```
      end;
    end;
  for k := 1 to m do
    begin
      read(f, i, j);
      read(f, d[i, j]);
      d[j, i] := d[i, j];
    end;
  close(f);
End;
Procedure Floyd;
Var sum, i, j, k : longint;
Begin
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if d[i, j] > d[i, k] + d[k, j] then d[i, j] := d[i, k] + d[k, j];
  min := MAX_VALUE;
  for i := 1 to n do
    begin
      sum := 0;
      for j := 1 to n do sum := sum + d[i, j];
      if sum < min then
        begin
          dia_diem := i;
          min := sum;
        end;
    end;
End;
Procedure Ghi;
Begin
  assign(f, FO); rewrite(f);
  writeln(f, dia_diem);
  write(f, min);
```

```
  close(f);
End;
Begin
  Doc;
  Floyd;
  Ghi;
End.
```

Bài 8: Thành phố trên sao hoả

Đầu thế kỷ 21, người ta thành lập một dự án xây dựng một thành phố trên sao Hoả để thế kỷ 22 con người có thể sống và sinh hoạt ở đó. Giả sử rằng trong thế kỷ 22, phương tiện giao thông chủ yếu sẽ là các phương tiện giao thông công cộng nên để đi lại giữa hai điểm bất kỳ trong thành phố người ta có thể yên tâm chọn đường đi ngắn nhất mà không sợ bị trễ giờ do kẹt xe. Khi mô hình thành phố được chuyển lên Internet, có rất nhiều ý kiến phàn nàn về tính hợp lý của nó, đặc biệt, tất cả các ý kiến đều cho rằng hệ thống đường phố như vậy là quá nhiều, làm tăng chi phí xây dựng cũng như bảo trì.

Hãy bỏ đi một số đường trong dự án xây dựng thành phố thoả mãn:

+ *Nếu giữa hai địa điểm bất kỳ trong dự án ban đầu có ít nhất một đường đi thì sự sửa đổi này không làm ảnh hưởng tới độ dài đường đi ngắn nhất giữa hai địa điểm đó.*
+ *Tổng độ dài của những đường phố được giữ lại là ngắn nhất có thể*

Dữ liệu: Vào từ file văn bản CITY.INP, chứa bản đồ dự án

+ Dòng thứ nhất ghi số địa điểm N và số đường phố m (giữa hai địa điểm bất kỳ có nhiều nhất là một đường phố nối chúng, $n \le 200$; $0 \le m \le n*(n-1)/2$)
+ m dòng tiếp theo, mỗi dòng ghi ba số nguyên dương u, v, c cho biết có đường hai chiều nối giữa hai địa điểm u, v và độ dài của con đường đó là c ($c \le 10000$)

Kết quả: Ghi ra file văn bản CITY.OUT, chứa kết quả sau khi sửa đổi

+ Dòng thứ nhất ghi hai số k,d. Trong đó k là số đường phố còn lại còn d là tổng độ dài của các con đường phố còn lại.
+ k dòng tiếp theo, mỗi dòng ghi hai số nguyên dương p, q cho biết cần phải giữ lại con đường nối địa điểm p với địa điểm q

Các số trên một dòng của các file CITY.INP, CITY.OUT được ghi cách nhau ít nhất một dấu cách

Ví dụ:

CITY.INP	**CITY.OUT**
10 12	9 21
1 2 1	1 2
1 5 2	1 5
2 6 7	3 4
3 4 1	3 7
3 7 2	5 6
4 8 8	6 7
5 6 3	6 9
6 7 1	7 8
6 9 2	9 10
7 8 5	
7 10 8	
9 10 4	

Chương trình

```
const
  tfi          =    'CITY.INP';
  tfo          =    'CITY.OUT';
  maxN         =    200;
  Unseen       =    2000000;
type
  mangB        =    array[1..maxN] of byte;
  mangL        =    array[1..maxN] of LongInt;
var
  fi,fo        :    text;
  N,M          :    LongInt;
  a            :    array[1..maxN] of ^mangL;
  Gr           :    array[1..maxN] of ^mangB;
  Tr           :    array[1..maxN,1..maxN] of byte;
  S,D          :    LongInt;

procedure CapPhat;
var i: integer;
```

```
begin
  for i:=1 to maxN do new(a[i]);
  for i:=1 to maxN do new(Gr[i]);
end;
procedure GiaiPhong;
var i: integer;
begin
  for i:=1 to maxN do Dispose(a[i]);
  for i:=1 to maxN do Dispose(Gr[i]);
end;
procedure Docdl;
var i,j,u,v,l: LongInt;
begin
  assign(fi,tfi); reset(fi);
  readln(fi,N,M);
  for i:=1 to N do
    for j:=1 to N do a[i]^[j]:=Unseen;
  for i:=1 to N do
    for j:=1 to N do Gr[i]^[j]:=0;
  for i:=1 to M do
    begin
      readln(fi,u,v,l);
      a[u]^[v]:=l; a[v]^[u]:=l;
      Gr[u]^[v]:=1; Gr[v]^[u]:=1;
    end;
  close(fi);
end;
procedure Floyd;
var k,i,j: integer;
begin
  Fillchar(Tr,sizeof(Tr),0);
  for k:=1 to N do
    for i:=1 to N do
      for j:=1 to N do
        if a[i]^[j]>=a[i]^[k]+a[k]^[j] then
```

```
            begin
              a[i]^[j]:=a[i]^[k]+a[k]^[j];
              Tr[i,j]:=k;
            end;
end;
procedure Solve;
var i,j: LongInt;
begin
  for i:=1 to N do
    for j:=1 to N do
      if (Gr[i]^[j]=1) and (Tr[i,j]>0) then
        begin
          Gr[i]^[j]:=0;
          Gr[j]^[i]:=0;
        end;
  S:=0;
  D:=0;
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if Gr[i]^[j]=1 then
        begin
          S:=S+a[i]^[j];
          D:=D+1;
        end;
end;
procedure inkq;
var i,j: LongInt;
begin
  assign(fo,tfo); rewrite(fo);
  writeln(fo,d,' ',S);
  for i:=1 to N-1 do
    for j:=i+1 to N do
      if Gr[i]^[j]=1 then
        writeln(fo,i,' ',j);
  close(fo);
```

```
end;
BEGIN
  CapPhat;
  Docdl;
  Floyd;
  Solve;
  Inkq;
  GiaiPhong;
END.
```

B. KẾT LUẬN

Để tìm đường đi ngắn nhất trên đồ thị còn có nhiều thuật toán nữa và cũng còn nhiều cách để cài đặt các thuật toán trên hiệu quả hơn. Tuy nhiên trong chuyên đề này tôi chỉ đưa ra các cách cài đặt cơ bản nhất để từ đó học sinh tự nghiên cứu và phát triển thêm. Vì thời gian và trình độ có hạn nên chuyên đề này có thể còn nhiều hạn chế, thiếu sót mong các đồng nghiệp và các em học sinh góp ý.

C. TÀI LIỆU THAM KHẢO

1, Tài liệu chuyên tin quyển 2 – Hồ Sỹ Đàm
2, Giải thuật và lập trình – Lê Minh Hoàng
3, Một số tài liệu khác của các đồng nghiệp

Chuyên đề xếp loại B

ỨNG DỤNG BFS VÀ DFS TRONG GIẢI BÀI TẬP LÝ THUYẾT ĐỒ THỊ

Tổ Tin học trường THPT Chuyên Lương Văn Tụy – Ninh Bình

1. Phần mở đầu

1.1 Lý do chọn đề tài.

- Bước sang thế kỷ 21, nhìn lại thế kỷ 20 là thế kỷ mà con người đạt được nhiều thành tựu khoa học rực rỡ nhất, một trong những thành tựu đó là sự bùng nổ của ngành khoa học máy tính. Sự phát triển kỳ diệu của máy tính trong thế kỷ này gắn liền với sự phát triển toán học hiện đại, đó là toán rời rạc. Toán rời rạc nói chung và lý thuyết đồ thị nói riêng là công cụ thiết yếu cho nhiều ngành khoa học kỹ thuật
- Trong chương trình học tập học sinh chuyên Tin ở trường THPT được trang bị các kiến thức về lý thuyết đồ thị để nhằm phục vụ cho việc lập trình giải toán, làm bài tập lập trình. Bởi điều căn bản thông qua giải bài tập, học sinh phải thực hiện những hoạt động nhất định bao gồm cả nhận dạng và thể hiện định nghĩa, định lý, quy tắc hay phương pháp, những hoạt động toán học phức hợp. Học sinh sẽ nắm được lý thuyết một cách vững vàng hơn thông qua việc làm bài tập.
- Việc cung cấp thêm một phương pháp giải bài tập cho học sinh chuyên Tin là một nhu cầu cần thiết. Hiện nay việc nghiên cứu khai thác một số yếu tố của lý thuyết đồ thị cũng được một số tác giả quan tâm. Nếu ta có các phương pháp giúp học sinh chuyên Tin trung học phổ thông vận dụng kiến thức về lý thuyết đồ thị vào giải toán thì sẽ giúp học sinh giải quyết được một số lớp bài toán góp phần nâng cao chất lượng dạy học giải bài tập cho học sinh chuyên Tin**.**
- **BFS** và **DFS** là những thuật toán tìm kiếm cơ bản nhưng rất quan trọng trên đồ thị. Những thuật toán này sẽ là nền móng quan trọng để có thể xây dựng và thiết kế những thuật giải khác trong lý thuyết đồ thị. Xuất phát từ những lý do trên tôi lựa chọn đề tài: ***"Ứng dụng BFS và DFS trong giải bài tập lý thuyết đồ thị "***.

1.2. Mục tiêu, nhiệm vụ của đề tài.

- Mục tiêu của đề tài: Chỉ ra hướng vận dụng DFS và BFS trong lý thuyết đồ thị vào giải các bài toán và tìm ra các biện pháp để giúp học sinh chuyên Tin

trung học phổ thông hình thành và phát triển năng lực vận dụng lý thuyết đồ thị vào giải bài tập lập trình.

- Nhiệm vụ của đề tài:

+ Tìm hiểu những nội dung cơ bản của lý thuyết đồ thị được trang bị cho học sinh chuyên Tin. Trong đó đi sâu vào hai ***thuật toán tìm kiếm trên đồ thị là DFS và BFS***

+ Chỉ ra hệ thống bài tập trong chương trình toán có thể vận dụng DFS và BFS để giải các bài tập trong lý thuyết đồ thị

+ Kiểm tra hiệu quả của các biện pháp, phương án lý thuyết đồ thị vào giải toán trong thực tế.

1.3. Phương pháp nghiên cứu.

- Nghiên cứu lý luận

+ Tài liệu Giáo khoa chuyên tin, sách nâng cao, sách chuyên đề.

+ Các tài liệu về lý thuyết đồ thị và những ứng dụng của nó trong thực tiễn cuộc sống và trong dạy học.

+ Các công trình nghiên cứu các vấn đề liên quan trực tiếp đến phương pháp đồ thị.

- Thực nghiệm sư phạm

+ Chỉ ra cho học sinh các dấu hiệu "nhận dạng" và cách thức vận dụng lý thuyết đồ thị vào giải bài tập toán.

+ Biên soạn hệ thống bài tập luyện tập cho học sinh và một số đề bài kiểm tra để đánh giá khả năng vận dụng lý thuyết đồ thị vào giải toán.

+ Tiến hành thực nghiệm và đánh giá kết quả thực nghiệm.

2. Phần nội dung

2.1. Cơ sở lý luận

Theo triết học duy vật biện chứng, mâu thuẫn là động lực thúc đẩy quá trình phát triển. Một vấn đề được gợi ra cho học sinh học tập chính là một mâu thuẫn giữa yêu cầu nhiệm vụ nhận thức với tri thức và kinh nghiệm sẵn có.

Theo các nhà tâm lý học, con người chỉ bắt đầu tư duy tích cực khi nảy sinh nhu cầu tư duy, tức là khi đứng trước một khó khăn về nhận thức cần phải khắc phục, một tình huống gợi vấn đề.

Theo tâm lý học kiến tạo, học tập chủ yếu là một quá trình trong đó người học xây dựng tri thức cho mình bằng cách liên hệ những cảm nghiệm mới với những tri thức đã có.

2.2.Thực trạng

a. Thuận lợi

- Được sự quan tâm, giúp đỡ tận tình của Ban Gíam Hiệu và tổ chức đoàn thể trong nhà trường. Sự ủng hộ nhiệt tình của các đồng nghiệp đã giúp cho quá trình giảng dạy Tin học của tôi đạt hiệu quả cao hơn.
- Học sinh lớp trương chuyên nói chung, học sinh lớp chuyên tin nói riêng thông minh, ham học. Trong lớp đa số học sinh tích cực phát biểu xây dựng bài, đó là nguồn động viên lớn trong quá trình giảng dạy của tôi.
- Nhìn chung, học tập theo phương pháp mới thì học sinh có hứng thú học tập hơn so với so với phương pháp dạy học truyền thống. Vì thế, có điều kiện phát triển tư duy và khả năng diễn đạt của các em.

b. Khó khăn

- Đội ngũ giáo viên Tin học còn thiếu, đặc biệt là giáo viên dạy chuyên Tin. Công việc mỗi giáo viên dạy tin học trong nhà trường phải đảm nhận rất nhiều, thời gian đầu tư cho chuyên môn còn hạn chế.
- Dạy học hiện đang theo lối dạy nhồi nhét, dạy luyện thi, đối phó với thi, kiểm tra sao cho có điểm số cao mà chưa quan tâm đến sự phát triển trí tuệ, năng lực cá nhân học sinh. Giáo viên cũng như học sinh chưa khắc phục được nhận thức, thói quen dạy học truyền thống, nặng về lý thuyết coi nhẹ thực hành ứng dụng. Các em học sinh thường chỉ nắm lý thuyết, việc vận dụng lý thuyết để làm các bài tập còn hạn chế. Giáo viên phải song hành việc dạy lý thuyết cho học sinh cùng với đưa ra phương pháp làm bài tập vận dụng các kiến thức đã học. Việc làm bài tập thực hành sẽ giúp học sinh nắm vững kiến

thức, và từ đó phát triển tư duy một cách tổng quát, giúp các em giải được một lớp bài toán lớn. Qua việc giải được các bài tập học sẽ sinh yêu thích, hứng thú với môn học hơn. Đề tài nghiên cứu "***Ứng dụng BFS và DFS trong giải bài tập lý thuyết đồ thị***" sẽ là nguồn tài liệu bổ ích cho giáo viên và học sinh trong việc giảng dạy chuyên đề lý thuyết đồ thị.

2.3. Quá trình thực hiện.

a. Các Khái niệm cơ bản của lý thuyết đồ thị

- Định nghĩa đồ thị: *Đồ thị là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này. Chúng ta phân biệt các loại đồ thị khác nhau bởi* ***kiểu*** *và* ***số lượng*** *cạnh nối hai đỉnh nào đó của đồ thị.*

- Định nghĩa 1. *Đơn đồ thị vô hướng G = (V,E) bao gồm V là tập các đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.*

- Định nghĩa 2. *Đa đồ thị vô hướng G= (V, E) bao gồm V là tập các đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp nếu chúng cùng tương ứng với một cặp đỉnh.*

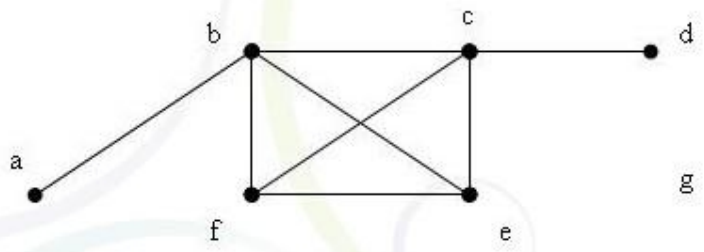
- Định nghĩa 3. *Giả đồ thị vô hướng G = (V, E) bao gồm V là tập các đỉnh và E là tập các cặp không có thứ tự gồm hai phần tử (không nhất thiết phải khác nhau) của V gọi là cạnh. Cạnh e được gọi là khuyên nếu nó có dạng e = (u, u).*

- Định nghĩa 4. *Đơn đồ thị có hướng G = (V, E) bao gồm V là tập các đỉnh và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung.*

- Định nghĩa 5. *Đa đồ thị có hướng G = (V, E) bao gồm V là tập các đỉnh và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp.*

- Cạnh liên thuộc: *Hai đỉnh u và v của đồ thị vô hướng G được gọi là kề nhau nếu (u,v) là cạnh của đồ thị G. Nếu e = (u, v) là cạnh của đồ thị ta nói cạnh này là liên thuộc với hai đỉnh u và v, hoặc cũng nói là nối đỉnh u và đỉnh v, đồng thời các đỉnh u và v sẽ được gọi là các đỉnh đầu của cạnh (u, v).*

- Bậc của đỉnh: *Bậc của đỉnh v trong đồ thị G=(V, E), ký hiệu deg(v) là số cạnh liên thuộc với nó. Nếu cạnh là khuyên thì được tính là 2.*



Hình 1. Đồ thị vô hướng

Thí dụ 1.

Xét đồ thị cho trong hình 1, ta có

deg(a) = 1, deg(b) = 4, deg(c) = 4, deg(f) = 3,

deg(d) = 1, deg(e) = 3, deg(g) = 0

Đỉnh bậc 0 gọi là *đỉnh cô lập*. Đỉnh bậc 1 được gọi là đỉnh treo. Trong ví dụ trên đỉnh g là đỉnh cô lập, a và d là các đỉnh treo.

Định lý 1.

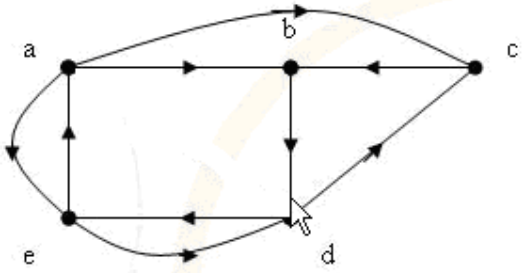
Giả sử G = (V, E) là đồ thị vô hướng với m cạnh. Khi đó tông bậc của tất cả các đỉnh bằng hai lần số cạnh.

Thí dụ 2.

Đồ thị với n đỉnh có bậc là 6 có bao nhiêu cạnh?

Giải: Theo định lý 1 ta có *2m = 6n*. Từ đó suy ra tổng các cạnh của đồ thị là 3n.

Ta gọi bán bậc ra (bán bậc vào) của đỉnh v trong đồ thị có hướng là số cung của đồ thị đi ra khỏi nó (đi vào nó) và ký hiệu là $deg^+(v)$ ($deg^-(v)$)



Hình 2. Đồ thị có hướng

Thí dụ 3.

Xét đồ thị cho trong hình 2. Ta có

$deg^-(a)=1, deg^-(b)=2, deg^-(c)=2, deg^-(d)=2, deg^-(e) = 2.$

$deg^+(a)=3, deg^+(b)=1, deg^+(c)=1, deg^+(d)=2, deg^+(e)=2.$

Định lý 2.

Giả sử G = (V, E) là đồ thị có hướng. Khi đó

Tổng tất cả các bán bậc ra bằng tổng tất cả các bán bậc vào bằng số cung.

Đồ thị vô hướng thu được bằng cách bỏ qua hướng trên các cung được gọi là *đồ thị vô hướng tương ứng* với đồ thị có hướng đã cho.

- Đường đi, chu trình trên đồ thị

Đường đi độ dài n *từ đỉnh u đến đỉnh v, trong đó n là số nguyên dương, trên đồ thị vô hướng G = (V, E) là dãy* $x_0, x_1,\ldots, x_{n-1}, x_n$

trong đó $u = x_0$, $v = x_n$, $(x_i , x_{i+1}) \in E$, $i = 0, 1, 2,\ldots, n-1$.

Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cạnh:

$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là u = v) được gọi là ***chu trình****. Đường đi hay chu trình được gọi là* ***đơn*** *nếu như không có cạnh nào bị lặp lại.*

- Tính liên thông của đồ thị - *Đồ thị vô hướng G = (V, E) được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.*

b. Biểu diễn đồ thị trên máy tính

- Có nhiều cách khác nhau để lưu trữ các đồ thị trong máy tính. Sử dụng cấu trúc dữ liệu nào thì tùy theo cấu trúc của đồ thị và thuật toán dùng để thao tác trên đồ thị đó. Trên lý thuyết, người ta có thể phân biệt giữa các cấu trúc danh sách và các cấu trúc ma trận. Tuy nhiên, trong các ứng dụng cụ thể, cấu trúc tốt nhất thường là kết hợp của cả hai. Người ta hay dùng các cấu trúc danh sách cho các đồ thị thưa (sparse graph), do chúng đòi hỏi ít bộ nhớ. Trong khi đó, các cấu trúc ma trận cho phép truy nhập dữ liệu nhanh hơn, nhưng lại cần lượng bộ nhớ lớn nếu đồ thị có kích thước lớn.

- Các cấu trúc danh sách

Danh sách liên thuộc (*Incidence list*) - Mỗi đỉnh có một danh sách các cạnh nối với đỉnh đó. Các cạnh của đồ thị được có thể được lưu trong một danh sách riêng (có thể cài đặt bằng mảng (*array*) hoặc danh sách liên kết động (*linked list*)), trong đó mỗi phần tử ghi thông tin về một cạnh, bao gồm: cặp đỉnh mà cạnh đó nối (cặp này sẽ có thứ tự nếu đồ thị có hướng), trọng số và các dữ liệu khác. Danh sách liên thuộc của mỗi đỉnh sẽ chiếu tới vị trí của các cạnh tương ứng tại danh sách cạnh này.

Danh sách kề (*Adjacency list*) - Mỗi đỉnh của đồ thị có một danh sách các đỉnh kề nó (nghĩa là có một cạnh nối từ đỉnh này đến mỗi đỉnh đó).

Trong đồ thị vô hướng, cấu trúc này có thể gây trùng lặp. Chẳng hạn nếu đỉnh 3 nằm trong danh sách của đỉnh 2 thì đỉnh 2 cũng phải có trong danh sách của đỉnh 3. Lập trình viên có thể chọn cách sử dụng phần không gian thừa, hoặc có thể liệt kê các quan hệ kề cạnh chỉ một lần. Biểu diễn dữ liệu này thuận lợi cho việc từ một đỉnh duy nhất tìm mọi đỉnh được nối với nó, do các đỉnh này đã được liệt kê tường minh.

- Các cấu trúc ma trận

Ma trận liên thuộc (*Incidence matrix*) - Đồ thị được biểu diễn bằng một ma trận $[b_{ij}]$kích thước *p* × *q*, trong đó *p* là số đỉnh và *q* là số cạnh, $b_{ij} = 1$chứa dữ liệu về quan hệ giữa đỉnh $v_i$và cạnh x_j. Đơn giản nhất: $b_{ij} = 1$nếu đỉnh $v_i$là một trong 2 đầu của cạnh x_j, bằng 0 trong các trường hợp khác.

Ma trận kề (*Adjaceny matrix*) - một ma trận *N* × *N*, trong đó *N* là số đỉnh của đồ thị. Nếu có một cạnh nào đó nối đỉnh v_ivới đỉnh $v_j$thì phần tử $M_{i,j}$bằng 1, nếu không, nó có giá trị 0. Cấu trúc này tạo thuận lợi cho việc tìm các đồ thị con và để đảo các đồ thị.

Ma trận dẫn nạp (*Admittance matrix*) hoặc **ma trận Kirchhoff** (*Kirchhoff matrix*) hay **ma trận Laplace** (*Laplacian matrix*) - được định nghĩa là kết quả thu được khi lấy ma trận bậc (*degree matrix*) trừ đi ma trận kề. Do đó, ma trận này chứa thông tin cả về quan hệ kề (có cạnh nối hay không) giữa các đỉnh lẫn bậc của các đỉnh đó.

c. Thuật toán tìm kiếm trên đồ thị

**** Thuật toán tìm kiếm theo chiều rộng.***

Trong lý thuyết đồ thị, **tìm kiếm theo chiều rộng** (**BFS**) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) thăm một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa thăm vào danh sách có thể thăm trong tương lai. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt thăm các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt thăm các đỉnh kề với nó mà chưa được thăm trước đó và lặp lại. Xem thêm thuật toán tìm kiếm theo chiều sâu, trong đó cũng sử dụng 2 thao

tác trên nhưng có trình tự thăm các đỉnh khác với thuật toán tìm kiếm theo chiều rộng.

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

1. Chèn đỉnh gốc vào hàng đợi
2. Lấy ra đỉnh đầu tiên trong hàng đợi và thăm nó
   - Nếu đỉnh này chính là đỉnh đích, dừng quá trình tìm kiếm và trả về kết quả.
   - Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được thăm trước đó vào hàng đợi.
3. Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được thăm – dừng việc tìm kiếm và trả về "không thấy".
4. Nếu hàng đợi không rỗng thì quay về bước 2.

```
1  Thủ tục BFS(G,v):
2      tạo hàng đợi Q
3      chèn v vào Q
4      đánh dấu đã thăm v
5      while Q còn khác rỗng:
6          lấy ra phần tử t đầu tiên trong Q
7          if t là đỉnh đích:
8              trả về t
9          for all cung e=(t, o) xuất phát từ t do
10             if chưa thăm o:
11                 đánh dấu đã thăm o
12                 chèn o vào Q
```

Thuật toán tìm kiếm theo chiều rộng được dùng để giải nhiều bài toán trong lý thuyết đồ thị, chẳng hạn như:

- Tìm tất cả các đỉnh trong một thành phần liên thông
- Thuật toán Cheney cho việc dọn rác
- Tìm đường đi ngắn nhất giữa hai đỉnh *u* và *v* (với chiều dài đường đi tính bằng số cung)

- Kiểm tra xem một đồ thị có là đồ thị hai phía
- Thuật toán Cuthill–McKee
- Thuật toán Ford–Fulkerson để tìm luồng cực đại trong mạng

**** Thuật toán tìm kiếm theo chiều sâu***

Tư tưởng chính của thuật toán là: Giả sử chúng ta đang xét trên đồ thị G(V,E). Từ một đỉnh u V hiện thời nào đó ta sẽ thăm tới đỉnh kề v của u và quá trình được lặp lại đối với đỉnh v. ở bước tổng quát, giả sử hiện tại đang xét đỉnh u_0, chúng ta sẽ có hai khả năng sẽ xảy ra:

-Nếu như tồn tại một đỉnh v_0 kề với u_0 mà chưa được thăm thì đỉnh v_0 đó sẽ trở thành đỉnh đã thăm và quá trình tìm kiếm lại bắt đầu từ đỉnh v_0 đó.

-Ngược lại, nếu mọi đỉnh kề với u_0 đều đã thăm thì ta sẽ quay trở lại đỉnh mà trước đó ta đến đỉnh u_0 để tiếp tục quá trình tìm kiếm.

Như vậy, trong quá trình thăm đỉnh bằng thuật toán tìm kiếm theo chiều sâu, đỉnh được thăm càng muộn càng sớm được duyệt xong (Cơ chế Last In First Out - *Vào sau ra trước).* Do đó, ta có thể tổ chức quá trình này bằng một thủ tục đệ quy như sau:

```
Procedure DFS(u);
 Begin
   Visit(u);
   Daxet[u]:=True;
   For v  Kề(u do
    if not Daxet[v] then DFS(v);
 End;
```

Và thủ tục duyệt hệ thống toàn bộ đỉnh của đồ thị sẽ là:

```
Procedure Find;
 Begin
   Fillchar(Daxet,SizeOf(Daxet),False);
   For u  V do
    If not Daxet[u] then DFS(u);
End;
```

Dễ nhận thấy rằng, mỗi lần gọi DFS(u) thì toàn bộ các đỉnh cùng thành phần liên thông với u sẽ được viếng thăm. Thủ tục Visit(u) là thao tác trên đỉnh u trong từng bài toán đặt ra cụ thể.

Độ phức tạp không gian của DFS thấp hơn của BFS (tìm kiếm ưu tiên chiều rộng). Độ phức tạp thời gian của hai thuật toán là tương đương nhau và bằng O(|V| + |E|).

Ý tưởng thuật toán

1. DFS trên đồ thị vô hướng cũng giống như khám phá mê cung với một cuộn chỉ và một thùng sơn đỏ để đánh dấu, tránh bị lạc. Trong đó mỗi đỉnh s trong đồ thị tượng trưng cho một cửa trong mê cung.
2. Ta bắt đầu từ đỉnh s, buộc đầu cuộn chỉ vào s và đánh đấu đỉnh này "đã thăm". Sau đó ta đánh dấu s là đỉnh hiện hành u.
3. Bây giờ, nếu ta đi theo cạnh (u,v) bất kỳ.
4. Nếu cạnh (u,v) dẫn chúng ta đến đỉnh "đã thăm" v, ta quay trở về u.
5. Nếu đỉnh v là đỉnh mới, ta di chuyển đến v và lăn cuộn chỉ theo. Đánh dấu v là "đã thăm". Đặt v thành đỉnh hiện hành và lặp lại các bước.
6. Cuối cùng, ta có thể đi đến một đỉnh mà tại đó tất cả các cạnh kề với nó đều dẫn chúng ta đến các đỉnh "đã thăm". Khi đó, ta sẽ quay lui bằng cách cuộn ngược cuộn chỉ và quay lại cho đến khi trở lại một đỉnh kề với một cạnh còn chưa được khám phá. Lại tiếp tục quy trình khám phá như trên.
7. Khi chúng ta trở về s và không còn cạnh nào kề với nó chưa bị khám phá là lúc DFS dừng.

d. Bài tập áp dụng DFS và BFS

Bài toán 1. Bài toán tìm thành phần liên thông của đồ thị

Cho một đồ thị G=(V.E). Hãy cho biết số thành phần liên thông của đồ thị và mỗi thành phần liên thông gồm những đỉnh nào.

Gợi ý làm bài:

Điều kiện liên thông của đồ thị thường là một yêu cầu tất yếu trong nhiều ứng dụng, chẳng hạn một mạng giao thông hay mạng thông tin nếu không liên thông thì xem như bị hỏng, cần sửa chữa. Vì thế, việc kiểm tra một đồ thị có liên thông hay không là một thao tác cần thiết trong nhiều ứng dụng khác nhau của đồ thị. Dưới đây ta xét một tình huống đơn giản (nhưng cũng là cơ bản) là xác định tính liên thông của một đồ thị vô hướng với nội dung cụ thể như sau: “cho trước một đồ thị vô hướng, hỏi rằng nó có liên thông hay không?”.

Để trả lời bài toán, xuất phát từ một đỉnh tùy ý, ta bắt đầu thao tác tìm kiếm từ đỉnh này (có thể chọn một trong hai thuật toán tìm kiếm đã nêu). Khi

kết thúc tìm kiếm, xảy ra hai tình huống: nếu tất cả các đỉnh của đồ thị đều được thăm thì đồ thị đã cho là liên thông, nếu có một đỉnh nào đó không được thăm thì đồ thị đã cho là không liên thông. Như vậy, câu trả lời của bài toán xem như một hệ quả trực tiếp của thao tác tìm kiếm. Để kiểm tra xem có phải tất cả các đỉnh của đồ thị có được thăm hay không, ta chỉ cần thêm một thao tác nhỏ trong quá trình tìm kiếm, đó là dùng một biến đếm để đếm số đỉnh được thăm. Khi kết thúc tìm kiếm, câu trả lời của bài toán sẽ phụ thuộc vào việc so sánh giá trị của biến đếm này với số đỉnh của đồ thị: nếu giá trị biến đếm bằng số đỉnh thì đồ thị là liên thông, nếu trái lại thì đồ thị là không liên thông. Trong trường hợp đồ thị là không liên thông, kết quả tìm kiếm sẽ xác định một thành phần liên thông chứa đỉnh xuất phát. Bằng cách lặp lại thao tác tìm kiếm với đỉnh xuất phát khác, không thuộc thành phần liên thông vừa tìm, ta nhận được thành phần liên thông thứ hai, ..., cứ như vậy ta giải quyết được bài toán tổng quát hơn là xác định các thành phần liên thông của một đồ thị vô hướng bất kỳ.

Như ta đã biết, các thủ tục DFS(u) và BFS(u) cho phép viếng thăm tất cả các đỉnh có cùng thành phần liên thông với u nên số thành phần liên thông của đồ thị chính là số lần gọi thủ tục trên. Ta sẽ dùng thêm biến đếm Connect để đếm số thành phần liên thông.

Và vòng lặp chính trong các thủ tục tìm kiếm theo chiều sâu hay chiều rộng chỉ cần sửa lại như sau:

```
Procedure Find;
 Begin
  Fillchar(Daxet,SizeOf(Daxet),False);
  Connect:=0;
  For u   V do
   If not Daxet[u] then
     Begin
       Inc(Connect); DFS(u); (*BFS(u)*)
     End;
 End;
```

Thủ tục Visit(u) sẽ làm công việc đánh số thành phần liên thông của đỉnh u:
LienThong[u]:=Connect;

Bài toán 2. Bài toán tìm đường đi giữa hai đỉnh của đồ thị

Cho đồ thị G=(V,E). Với hai đỉnh s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t.

Gợi ý làm bài:

Do thủ tục DFS(s) và BFS(s) sẽ thăm lần lượt các đỉnh liên thông với u nên sau khi thực hiện xong thủ tục thì có hai khả năng:

-Nếu Daxet[t]=True thì có nghĩa: tồn tại một đường đi từ đỉnh s tới đỉnh t.

-Ngược lại, thì không có đường đi nối giữa s và t.

Vấn đề còn lại của bài toán là: *Nếu tồn tại đường đi nối đỉnh s và đỉnh t thì làm cách nào để viết được hành trình (gồm thứ tự các đỉnh) từ s đến t.* Về kỹ thuật lấy đường đi là: Dùng một mảng Truoc với: Truoc[v] là đỉnh trước của v trong đường đi. Khi đó, câu lệnh If trong thủ tục DFS(u) được sửa lại như sau:

```
If not Daxet[v] then
  Begin
    DFS(v);
    Truoc[v]:=u;
  End;
```

Còn với thủ tục BFS ta cũng sửa lại trong lệnh If như sau:

```
If not Daxet[w] then
  Begin
    Kết nạp w vào Queue;
    Daxet[w]:=True;
    Truoc[w]:=v;
  End;
```

Việc viết đường đi lên màn hình (hoặc ra file) có thể có 3 cách:

-Viết trực tiếp dựa trên mảng Truoc: Hiển nhiên đường đi hiển thị sẽ ngược từ đỉnh t trờ về s như sau:

$$t \leftarrow p1 := Truoc[t] \leftarrow p2 := Truoc[p1] \leftarrow \ldots \leftarrow s$$

-Dùng thêm một mảng phụ P: cách này dùng để đảo đường đi từ mảng Truoc để có đường đi thuận từ đỉnh s đến đỉnh t.

-Cách thứ 3: là dùng chương trình đệ quy để viết đường đi.

```
Procedure Print_Way(i:Byte);
  If i<>s then
    Begin
      Print_Way(Truoc[i]);
```

```
    Write('đ',i);
  End;
```

Lời gọi thủ tục đệ quy như sau:

```
Write(s);
Print_Way(s);
```

Các bạn có thể tuỳ chọn cách mà mình thích nhưng thiết nghĩ đó chưa phải là vấn đề quan trọng nhất. Nếu tinh ý dựa vào thứ tự thăm đỉnh của thuật toán tìm kiếm theo chiều rộng BFS ta sẽ có một nhận xét rất quan trọng, đó là: Nếu có đường đi từ s đến t, thì đường đi tìm được do thuật toán tìm kiếm theo chiều rộng cho chúng ta một hành trình cực tiểu về số cạnh.

Bài toán 3: Truyền tin

Một lớp gồm N học viên, mỗi học viên cho biết những bạn mà học viên đó có thể liên lạc được (chú ý liên lạc này là liên lạc một chiều, ví dụ : Bạn An có thể gửi tin tới Bạn Vinh nhưng Bạn Vinh thì chưa chắc đã có thể gửi tin tới Bạn An). Thầy chủ nhiệm đang có một thông tin rất quan trọng cần thông báo tới tất cả các học viên của lớp (tin này phải được truyền trực tiếp). Để tiết kiệm thời gian, thầy chỉ nhắn tin tới 1 số học viên rồi sau đó nhờ các học viên này nhắn lại cho tất cả các bạn mà các học viên đó có thể liên lạc được, và cứ lần lượt như thế làm sao cho tất cả các học viên trong lớp đều nhận được tin .

Câu hỏi

Có phương án nào giúp thầy chủ nhiệm với một số ít nhất các học viên mà thầy chủ nhiệm cần nhắn?

Gợi ý làm bài:

- Có thể nhận thấy bài toán này chính là bài toán 1 đã phát biểu phía trên. Có thể coi mỗi học sinh là một đỉnh của đồ thị. Hai học sinh có thể liên lạc được với nhau là một cạnh. Từ đó suy ra bài toán này là . Bài toán tìm thành phần liên thông của đồ thị.

Bài toán 4: Đường đi đến số 0

Mỗi một số nguyên dương đều có thể biểu diễn dưới dạng tích của 2 số nguyên dương X,Y sao cho X<=Y. Nếu như trong phân tích này ta thay X bởi X-1 còn Y bởi Y+1 thì sau khi tính tích của chúng ta thu được hoặc là một số nguyên dương mới hoặc là số 0.

Ví dụ: Số 12 có 3 cách phân tích 1*12,3*4, 2*6 . Cách phân tích thứ nhất cho ta tích mới là 0 : (1-1)*(12+1) = 0, cách phân tích thứ hai cho ta tích mới 10 :

(3-1)*(4+1) = 10, còn cách phân tích thứ ba cho ta 7 : (2-1)*(6+1)=7. Nếu như kết quả là khác không ta lại lặp lại thủ tục này đối với số thu được. Rõ ràng áp dụng liên tiếp thủ tục trên, cuối cùng ta sẽ đến được số 0, không phụ thuộc vào việc ta chọn cách phân tích nào để tiếp tục

Yêu cầu: Cho trước số nguyên dương N (1<=N<=10000), hãy đưa ra tất cả các số nguyên dương khác nhau có thể gặp trong việc áp dụng thủ tục đã mô tả đối với N.

Dữ liệu: Vào từ file Zeropath.Inp chứa số nguyên dương N.

Kết quả: Ghi ra file văn bản Zeropath.Out :

Dòng đầu tiên ghi K là số lượng số tìm được

Dòng tiếp theo chứa K số tìm được theo thứ tự tăng dần bắt đầu từ số 0.

Lưu ý: Có thể có số xuất hiện trên nhiều đường biến đổi khác nhau, nhưng nó chỉ được tính một lần trong kết quả.

Ví dụ:

ZEROPATH.INP	ZEROPATH.OUT
12	6 0 3 4 6 7 10

Gợi ý làm bài:

Đơn giản là sau mỗi lần phân tích thì chắc chắn kết quả mới luôn nhỏ hơn số đó. Vì vậy ta chỉ cần Lưu trữ dưới mảng A: [0..10000] of boolean ; trong đó A[i] =true nếu nó xuất hiện trên đường đi đó, ngược lại thì A[i] =false. Bằng cách loang theo chiều sâu, chúng ta sẽ đánh dấu các số nếu nó được dùng đến, cho đến khi không thể nào loang được nữa thì dừng.

Bài toán 5. Con ngựa

Một bàn cờ hình chữ nhật kích thước MxN, M,N nguyên dương không lớn hơn 100. Bàn cờ chia thành các ô vuông đơn vị bằng các đường song song với các cạnh. Các dòng ô vuông đánh số từ 1 đến M từ trên xuống dưới, các cột đánh số từ 1 đến N từ trái sang phải. Cho trước một số nguyên dương K<=1000. Một con ngựa đứng ở ô [u,v] và nhảy không quá k bước.

Yêu cầu: Hãy cho biết con ngựa có thể nhảy đến bao nhiêu ô khác ô[u,v] trên bàn cờ và đó là những ô nào (khi đứng tại một ô, con ngựa có thể nhảy tới ô đối đỉnh của hình chữ nhật kích thước 2x3).

Dữ liệu: Vào từ file MA.INP trong đó :

Dòng đầu tiên ghi hai số M,N

Dòng thứ hai ghi số K

Dòng thứ ba ghi hai số U,V

Kết quả: Ghi ra file MA.OUT :

Dòng đầu tiên ghi S là số ô con ngựa có thể nhảy đến

Tiếp theo là S dòng, mỗi dòng ghi chỉ số dòng và chỉ số cột của một ô mà con ngựa có thể nhảy đến.

Ví dụ:

MA.INP	MA.OUT
5 5	6
1	1 1 1 5 3 1 3 5 4 2 4 4
2 3	

Gợi ý làm bài

Chúng ta sẽ loang theo chiều sâu, tìm kiếm xem những ô nào con mã có thể đặt chân đến trong vòng K bước nhảy.

Bài toán 6: Đường đi trên lưới ô vuông

Cho một lưới ô vuông kích thước N x N. Các dòng của lưới được đánh số từ 1 đến N từ trên xuống dưới, các cột của lưới được đánh số từ 1 đến N từ trái qua phải. Ô nằm trên giao của dòng i, cột j sẽ được gọi là ô (i, j) của lưới. Trên mỗi ô (i, j) của lưới người ta ghi một số nguyên dương aị, i, j = 1,2,..., N. Từ một ô bất kỳ của lưới được phép di chuyển sang ô có chung cạnh với nó. Thời gian để di chuyển từ một ô này sang một ô khác là 1 phút. Cho trước thời gian thực hiện di chuyển là K (phút), hãy xác định cách di chuyển bắt đầu từ ô (1, 1) sao cho tổng các số trên các ô di chuyển qua là lớn nhất (Mỗi ô của lưới có thể di chuyển qua bao nhiêu lần cũng được).

Dữ liệu: Vào từ file văn bản NETSUM.INP:

- Dòng đầu tiên chứa các số nguyên dương N, K (2 N 100), 1 K 10000).
- Dòng thứ i trong số N dòng tiếp theo chứa các số nguyên ai1, ai2..., aiN, 0 < aị 10000.

(Các số trên cùng một dòng được ghi cách nhau bởi ít nhất một dấu cách).

Kết quả: Ghi ra file văn bản NETSUM.OUT:

- Dòng đầu tiên ghi tổng số các số trên đường di chuyển tìm được.
- K dòng tiếp theo mỗi dòng ghi toạ độ của một ô trên đường di chuyển (bắt đầu t ô (1, 1)).

Ví dụ:

NETSUM.INP	NETSUM.OUT
5 7	2 1
1 1 1 1 1	1 1
1 1 3 1 9	1 2
1 1 6 1 1	1 3
1 1 3 1 1	2 3
1 1 1 1 1	2 4
2 3	
2 4	

Gợi ý làm bài:

Loang các ô có thể đến của các đường đi trên lưới. Tìm cách đi nào có đường đi mà tổng lớn nhất thì sẽ lấy.

Bài toán 7:Bàn cờ thế

2793. Bàn cờ thế

Mã bài: CHESSCBG

Một bàn cờ thế là một bảng gồm 4 dòng, 4 cột. Mỗi thế cờ là một cách sắp xếp 8 quân cờ, hai quân khác nhau ở hai ô khác nhau. Bài toán đặt ra là cho hai thế cờ 1 và 2, hãy tìm một số ít nhất bước di chuyển quân để chuyển từ thế 1 sang thế 2; một bước di chuyển quân là một lần chuyển quân cờ sang ô trống kề cạnh với ô quân cờ đang đứng.

Dữ liệu vào

Từ file văn bản gồm 8 dòng, mỗi dòng là một xâu nhị phân độ dài 4 mà số 1/0 tương ứng với vị trí có hoặc không có quân cờ. Bốn dòng đầu là thế cờ 1, bốn dòng sau là thế cờ 2.

Dữ liệu ra

Gồm 1 dòng duy nhất là số bước chuyển quân ít nhất

Ví dụ

Dữ liệu vào:

1111
0000
1110

0010
1010
0101
1010
0101

Dữ liệu ra :

4

Gợi ý làm bài :

Chúng ta sẽ giải quyết nhờ một phương pháp hết sức đơn giản : Tìm kiếm theo chiều rộng. Ta sẽ coi một trạng thái l của bảng là một đỉnh của đồ thị mới. Mỗi lần di chuyển một quân cờ trên bàn thì nó sẽ tạo ra một trạng thái mới của bảng, tức là sẽ đến một đỉnh mới. Trong bài toán này chúng ta sẽ chỉ xét với (m=n=4). Tức là ở file input, không có dòng đầu tiên.

Mỗi trạng thái của bảng là một loạt các ô có giá trị 0 và 1. Chúng ta sẽ trải nó ra thành một hàng thì sẽ tạo ra một bảng một chiều chỉ toàn các số 1 và 0. Vì có 16 ô, nên mỗi bảng như vậy sẽ tương ứng với hệ nhị phân của một số nào đó nằm trong word (16 bit). Tức là số đỉnh của đồ thị có thể có sẽ là 216.

a1	a2	a3	a4
a5	a6	a7	a8
a9	a10	a11	a12
a13	a14	a15	a16

Bảng 1

Bảng mới sau khi trải như sau :

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16

Bảng 2

Khi di chuyển các quân thì các quân (ở bảng 1) có thể đi tới 4 ô bên cạnh nếu có thể (trừ trờng hợp đi ra ngoài bảng). Tức là tương ứng ở bảng 2, giả sử tại vị trí Ai thì nó có thể có đi đến những ô : Ai-1, Ai+1, Ai-4, Ai+4 (Phải trừ những trờng hợp nó đi ra ngoài bảng). Quá trình di chuyển quân 1 như vậy tức là bít thứ i sẽ được tắt, còn các bít được đến sẽ được bật. Loang theo chiều rộng của quá trình chuyển bit cho đến khi chuyển đến được trạng thái

mong muốn. Đường di chuyển đó chính là cách di chuyển cờ với số bước ít nhất để đến trạng thái mong muốn.

Bài toán 8. Số rõ ràng

Các nhà toán học đưa voà lý thuyết nhiều cách phân loại số, ví dụ, với các số nguyên ta có số chẵn và số lẻ, số nguyên tố và hợp số, số chính phương và không chính phương… Bob cũng muốn đặt dấu ấn của mình trong lĩnh vực phân loại số. Bob chia các số nguyên dương thành 2 loại: *rõ ràng là luẫn quẫn*. Việc xác định một số thuộc loại nào được thực hiện theo giải thuật sau: Với số nguyên dương n, ta tạo số mới bằng cách lấy tổng bình phương các chữ số của nó, với số mới này ta lặp lại công việc trên. Nếu trong quá trình trên, ta nhận được số mới là 1, thì số n ban đầu được gọi là số rõ ràng. Ví dụ với n=19, ta có:

$\mathbf{19 \to 82 (=1^2+9^2) \to 68 \to 100 \to 1}$

Như vậy, 19 là số rõ ràng.

Không phải mọi số đều rõ ràng. Ví dụ, với n=12 ta có:

$12 \to 5 \to 25 \to 29 \to 85 \to 89 \to 145 \to 42 \to 20 \to 4 \to 16 \to 37 \to 58 \to 89 \to 145$

Rất thú vị với cách phân loại của mình, Bob muốn biết, trong thực tế, số rõ ràng nhiều hay ít?.

Yêu cầu: Cho hai số nguyên dương A và B($1 \le A \le B \le 10000000$). Hãy xác định K-số lương số rõ ràng nằm trong khoảng [A,B].

Dữ liệu: Vào từ file CLEAR.INP gồm 1 dòng chứa 2 số nguyên A và B

Kết quả: Đưa ra file văn bản CLEAR.OUT số nguyên K

Ví dụ:

CLEAR.INP	CLEAR.OUT
2 20	4

Gợi ý làm bài:

Bằng cách loang theo chiều sâu, chúng ta sẽ đánh dấu các số nếu nó được dùng đến là số rõ ràng.

Bài toán 9:

Từ tập các bài có trên SPOJ (oi)

2195. Điều kiện thời tiết

Mã bài: WEATHER

Hãng hàng không OlympAirways thực hiện các chuyến bay giữa n sân bay được đánh số từ 1 đến n. Hệ thống các chuyến bay được thiết lập sao cho giữa 2 sân bay bất kỳ được phục vụ bởi hãng luôn có một đường bay bao

gồm một hoặc nhiều chuyến bay trực tiếp giữa hai sân bay. Mỗi chuyến bay thực hiện việc di chuyển giữa hai thành phố theo cả hai chiều.

Trung tâm điều khiển của hãng đưa ra khái niệm độ dính kết giữa cặp hai sân bay A và B được xác định như là số lượng các chuyến bay mà việc không thực hiện một trong số chúng (các chuyến bay khác vẫn thực hiện bình thường) dẫn đến không thể bay từ sân bay A đến sân bay B.

Một nghiên cứu cho biết rằng, trong điều kiện thời tiết xấu, tổng độ dính kết giữa các cặp sân bay phải đạt đến một giá trị nhất định thì hệ thống đường bay mới được gọi là an toàn.

Yêu cầu: Hãy giúp trung tâm điều khiển tính tổng độ dính kết giữa mọi cặp sân bay.

Dữ liệu

Dòng đầu tiên chứa số nguyên n ($1 \le n \le 100$)

Dòng thứ hai chứa số nguyên m ($1 \le m \le 5000$) - số lượng các chuyến bay

Mỗi dòng trong số m dòng tiếp theo chứa thông tin về một chuyến bay, bao gồm hai số nguyên dương trong khoảng từ 1 đến n: chỉ số của hai sân bay được nối bởi chuyến bay.

Kết qủa

In ra 1 số nguyên duy nhất là tổng độ dính kết giữa mọi cặp sân bay (A, B) (với A < B).

Ví dụ

Dữ liệu:

5
5
1 2
4 2
4 5
3 2
3 1

Kết qủa

10

Gợi ý làm bài:

Bài này với mỗi cạnh chúng ta cần kiểm tra xem nó có là cầu không (dùng DFS hoặc BFS). Nếu là cầu thì tổng độ kết dính sẽ tăng lên một giá trị = tích của các đỉnh thuộc hai miền mà cạnh đó làm cầu.

Bài toán 10:

2719. Bãi cỏ ngon nhất

Mã bài: VBGRASS

Bessie dự định cả ngày sẽ nhai cỏ xuân và ngắm nhìn cảnh xuân trên cánh đồng của nông dân John, cánh đồng này được chia thành các ô vuông nhỏ với R (1 <= R <= 100) hàng và C (1 <= C <= 100) cột. Bessie ước gì có thể đếm được số khóm cỏ trên cánh đồng.

Mỗi khóm cỏ trên bản đồ được đánh dấu bằng một ký tự '#' hoặc là 2 ký tự '#' nằm kề nhau (trên đường chéo thì không phải). Cho bản đồ của cánh đồng, hãy nói cho Bessie biết có bao nhiêu khóm cỏ trên cánh đồng.

Ví dụ như cánh đồng dưới dây với R=5 và C=6:

.#....
..#...
..#..#
...##.
.#....

Cánh đồng này có 5 khóm cỏ: một khóm ở hàng đầu tiên, một khóm tạo bởi hàng thứ 2 và thứ 3 ở cột thứ 2, một khóm là 1 ký tự nằm riêng rẽ ở hàng 3, một khóm tạo bởi cột thứ 4 và thứ 5 ở hàng 4, và một khóm cuối cùng ở hàng 5.

Dữ liệu

Dòng 1: 2 số nguyên cách nhau bởi dấu cách: R và C

Dòng 2..R+1: Dòng i+1 mô tả hàng i của cánh đồng với C ký tự, các ký tự là '#' hoặc '.' .

Kết quả

Dòng 1: Một số nguyên cho biết số lượng khóm cỏ trên cánh đồng.

Ví dụ

Dữ liệu

```
5 6
.#....
..#...
..#..#
...##.
```

.#....

Kết quả

5

Gợi ý làm bài:

Coi mỗi ô trên cánh đồng là một đỉnh của đồ thị. Sử dụng thuật toán loang theo chiều rộng để đếm số bãi cỏ

Bài toán 11.

2969. Bin Laden

Mã bài: BINLADEN

Bin Laden

Trùm khủng bố Bin Laden trốn trong 1 căn hầm được đào sâu xuống mặt đất M tầng, mỗi tầng có N phòng. Các phòng được ngăn cách bằng các cửa rất khó phá. Các phòng có cửa xuống phòng ngay phía dưới và 2 phòng ở 2 bên. Từ trên mặt đất có N cửa xuống N phòng tầng -1. Bin Laden ở tầng dưới cùng (tầng -M) phòng thứ N (phòng ở bên phải nhất). Mỗi cửa được làm bằng một kim loại khác nhau với độ dày khác nhau nên việc phá cửa cần thời gian khác nhau.

Bạn hãy tìm cách đi từ mặt đất xuống phòng của Bin Laden nhanh nhất không hắn thoát mất.

Dữ liệu

Dòng 1 ghi M và N

Dòng 2 đến 2M + 1, dòng chẵn ghi N số, dòng lẻ ghi N - 1 số là chi phí để phá cửa.

Kết quả

Ghi ra 1 số là thời gian nhỏ nhất để đến được phòng của Bin Laden

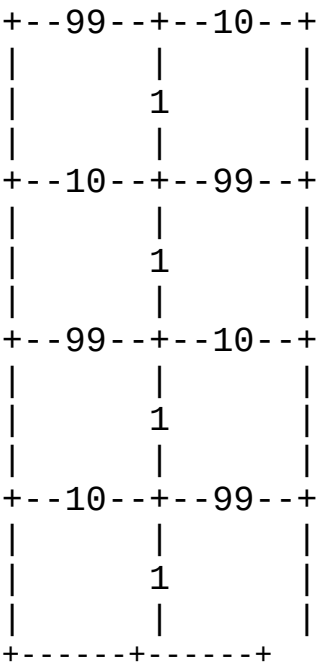
Ví dụ

Dữ liệu

4 2
99 10
1
10 99
1
99 10
1

10 99
1

Kết quả
44



Đi theo đường zigzac

Giới hạn

- 1 <= M <= 2222
- 1 <= N <= 10
- Chi phí của các cánh cửa thuộc [0, 1000].

Gợi ý làm bài:

Coi mỗi phòng là một đỉnh của đồ thị. Hai đỉnh có đường nối nếu các phòng kề cạnh, và có trọng số bằng thời gian phá tường ngăn cách. Bài toán trở thành tìm đường đi ngắn nhất từ 1 phòng nào đó của tầng trên xuống một phòng cuối cùng của tầng dưới.

Bài toán 12:

3892. Trồng cây

Mã bài: GARDEN25

Nhà sherry có 1 khu vườn rất rộng và trồng nhiều loại cây. Để đón tết năm 2010 sherry sẽ trồng thật nhiều mai và đào. Và chỉ có mai và đào mà thôi.

Khu vườn nhà sherry có dạng hình chữ nhật, kích thước M x N. Trên đó có 1 số ô được đánh dấu để trồng cây. Để tăng tính thẩm mỹ của khu vườn sherry muốn số cây mai và đào trong khu vườn chênh lệch nhau không quá 1. Đồng thời số cây mai, đào trên mỗi hàng, cột của khu vườn cũng chênh lệch nhau không quá 1.

Input

Dòng 1: ghi 2 số nguyên M, N ($1 \le M, N \le 250$)

M dòng tiếp theo: Mỗi dòng ghi N số, trong đó số thứ j của hàng thứ i bằng 1/0 tương ứng với ô (i, j) có/không trồng cây.

Output

Gồm M dòng: Mỗi dòng ghi N số nguyên, các ô không trồng cây ghi ra 0, các ô trồng cây có giá trị 1/2 tương ứng ở đó trồng mai/đào.

Example

Input:

4 4
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1

Output:

2 0 1 0
0 2 0 1
1 0 2 0
0 1 0 2

Gợi ý làm bài:

Ta coi mỗi hàng , mỗi cột là một đỉnh của đồ thị . Nếu ô (i,j) có giá trị <> 0 thì đỉnh hàng i nối với đỉnh cột j . Bài toán trở thành :

Tìm các tô các cạnh của một đồ thị bằng hai màu , sao cho :

- với mỗi đỉnh thì độ chênh lệch hai màu tô các cạnh nối nó chênh lệch không quá 1 .
- Với cả đồ thị chúng cũng chênh lệch nhau không quá 1 . Chúng ta có phương pháp giải quyết bài toán này như sau :
- Nhận xét 1 : Nếu xuất phát từ một đỉnh bậc lẻ và đi một cách bất kỳ theo các cung của đồ thị , mỗi cung đi qua chỉ một lần thì trạng thái tắc đường phải

xảy ra tại một đỉnh bậc lẻ khác (số đỉnh bậc lẻ nếu có trong đồ thị là một số chẵn)

- Nhận xét 2 : Nếu xuất phát từ một đỉnh bậc chẵn trong đồ thị không có đỉnh bậc lẻ và đi một cách bất kỳ các cung của đồ thị , mỗi cung đi qua chỉ một lần thì trạng thái tắc đường phải xảy ra tại chính đỉnh xuất phát . Trạng thái tắc đường là trạng thái mà tại đỉnh vừa tới không còn cung nào chưa đi qua . Dựa vào hai nhận xét chúng ta có :
- Trường hợp 1 : Khi đồ thị còn đỉnh bậc lẻ. Chọn một đỉnh lẻ bất kỳ để xuất phát . Bằng một cách đi bất kỳ qua các cung của đồ thị màu chưa được tô , mỗi cung đi qua ta tô xen kẽ bằng hai màu cho đến khi tắc đường . Trong trường hợp này ,tại đỉnh bậc lẻ kết thúc đường đi trên ,không còn cung nào chứa nó chưa được tô , đồng thời , tại đỉnh xuất phát , số cung còn lại chưa tô (nếu có) là một số chẵn , còn tại các đỉnh còn lại trên đường đi số cung được tô bằng các màu bằng nhau
- Trường hợp 2 : Khi đồ thị chỉ còn đỉnh bậc chẵn Trong trường hợp này , tất cả các cung kề với các đỉnh bậc lẻ (nếu có) của đồ thị ban đầu đều đã được tô . Chọn một đỉnh nào đó còn có cung chưa tô chứa nó làm đỉnh xuất phát và cũng đi một cách bất kỳ theo các cung chưa tô cho đến khi đạt được trạng thái kết thúc (tại đỉnh xuất phát) . Bằng cách tô màu các cung xen kẽ trên lộ trình đã đi qua . Khi đó só lượng các cung được tô hai màu được tô kề với mỗi đỉnh trên lộ trình là bằng nhau .

2.4. Kết quả thu được

- Học sinh sau khi học chuyên đề này sẽ hứng thú với việc học lý thuyết đồ thị. Khi gặp một bài toán về lý thuyết đồ thị sẽ tự tin làm bài. DFS và BFS còn là nền tảng để dạy các phần lý thuyết khác trong chuyên đề đồ thị

3. Phần kết luận

- BFS và **DFS** là những thuật toán tìm kiếm cơ bản nhưng rất quan trọng trên đồ thị. Những thuật toán này sẽ là nền móng quan trọng để có thể xây dựng và thiết kế những thuật giải khác trong lý thuyết đồ thị. Tuy nhiên có thể thấy rằng phương pháp này còn hạn chế khi số lượng các phần tử của tập D lớn. Nó thể hiện ở chỗ thời gian tính toán để cho ra kết quả thường không chấp nhận được. Do đó trong phương pháp Tìm kiếm theo DFS và BFS cần phải bổ sung các phương pháp cho phép bỏ qua hoặc gộp một số phần tử. Điều này cải thiện đáng kể thời gian thực hiện chương trình.

- Phương pháp Tìm kiếm theo DFS và BFS là một trong những phương pháp dễ hiểu nhất với học sinh và có thể áp dụng để giải rất nhiều bài toán tối ưu với dữ liệu nhỏ (thường đạt đến 50% đến 60% số tets của một bài thi).

4. Tài liệu tham khảo:

1. Cấu trúc dữ liệu và giải thuật – Lê Minh Hoàng (DHSP Hà Nội)

2. Tài liệu tập huấn phát triển chuyên môn giáo viên Tin học - Nhiều tác giả

3. Tài liệu hội thảo phát triển chuyên môn giáo viên Tin học - Nhiều tác giả

4. Thuật toán quay lui – Lê Sỹ Hùng (Hương Sơn – Hà Tĩnh)

5. Tài Liệu sách giáo khoa chuyên tin tập 1,2 - Nhiều tác giả

6. VNOI - Olympic tin học Việt Nam - Mục lục diễn đàn - Forum

Chuyên đề xếp loại B

Chuyên đề

MỘT SỐ ỨNG DỤNG CỦA DFS

Ngô Trung Tưởng-GV trường THPT chuyên Lê Hông Phong-Nam Định

Rất nhiều thuật toán trên đồ thị được xây dựng dựa trên cơ sở duyệt qua tất cả các đỉnh của đồ thị sao cho mỗi đỉnh của nó được thăm đúng một lần. Vì vậy, việc xây dựng những thuật toán cho phép duyệt một cách có hệ thống tất cả các đỉnh của đồ thị cùng các ứng dụng của nó là một vấn đề quan trọng thu hút sự quan tâm nghiên cứu của nhiều tác giả. Những thuật toán như vậy gọi là thuật toán tìm kiếm trên đồ thị. Trong chuyên đề này tôi sẽ giới thiệu một số ứng dụng của thuật toán tìm kiếm theo chiều sâu (DFS-Depth First Search) vào việc giải một số bài toán trên đồ thị.

I. Thứ tự duyệt đến và duyệt xong:

Thủ tục: DFS(u)

- Khi bắt đầu vào thủ tục DFS(u) ta nói đỉnh u được duyệt đến hay được thăm (discover), tức là tại thời điểm đó quá trình tìm kiếm theo chiều sâu bắt đầu, từ u sẽ xây dựng nhánh cây DFS gốc u.
- Khi chuẩn bị thoát khỏi thủ tục DFS(u) để lùi về, ta nói đỉnh u được duyệt xong (finish), tức là tại thời điểm đó quá trình tìm kiếm theo chiều sâu kết thúc.

Trong thủ tục DFS ta thêm vào biến đếm Time để xác định thời điểm duyệt đến d[u] và thời điểm duyệt xong f[u]

- Mô hình cài đặt thuật toán DFS có thêm vào thứ tự duyệt đến và duyệt xong

```
Procedure DFS(u ∈ V)
Begin
    Time:=Time+1;
    d[u]:=Time;
    output ←u; // thăm u
    for  ∀v∈ V: (u,v) ∈ E do //duyệt mọi đỉnh nối từ v tới u
    // nếu v chưa thăm gọi đệ qui tìm kiếm theo chiều sâu từ v
        If  d[v]=0 then       DSF(v)
    Time:=Time+1;
    f[u]:=Time;
```

end;

- Thứ tự duyệt đến và duyệt xong có ý nghĩa rất quan trọng trong nhiều thuật toán có sử dụng DFS, như tìm thành phần liên thông mạnh, tìm cầu, khớp của đồ thị,…

II. Một số ứng dụng

1. Tìm thành phần liên thông mạnh trên đồ thị có hướng (thuật toán Tarjan)

a.Ý tưởng: Trong thuật toán Tarjan để liệt kê các thành phần liên thông mạnh trên đồ thị có hướng dựa trên thuật toán tìm kiếm theo chiều sâu DFS.

- Cài đặt thuật toán dựa trên thứ tự duyệt đến.

+ Number[u] là thứ tự duyệt đến của đỉnh u

+ Color[u] là màu đỉnh u, nếu Color[u] là white (màu trắng) thì đỉnh u chưa được thăm, nếu là Gray(màu xám) thì đỉnh u đã được thăm nhưng chưa duyệt xong, nếu là Black (màu đen) thì đỉnh u đã bị xóa khỏi nhánh cây DFS.

+ Low[u] là giá trị Number[.] nhỏ nhất trong các đỉnh mà có thể đến được từ một đỉnh v nào đó của nhánh DFS gốc u bằng một cung. Tính Low[u] như sau:

Khởi tạo Low[u]:=+∞, xét đỉnh v nối từ u có hai khả năng

++ Nếu v có màu Gray (xám):

Low[u]:=min(Low[u],Number[v])

++ Nếu v có màu White (trắng):

Thăm V

Low[u]:=min(Low[u],Low[v])

+ Khi duyệt xong một đỉnh u: so sánh Low[u] và Number[u], nếu Low[u] >= Number[u], thì u là đỉnh đầu tiên trong một thành phần liên thông mạnh thuộc cây DFS gốc u, bởi vì không có cung nối từ đỉnh DFS gốc u tới một đỉnh thăm trước.

b. Mô hình cài đặt thuật toán Tarjan

```
Procedure Tarjan(u);
Begin
   Time:=Time+1;
   Number[u]:=Time;
   Low[u]:=+∞
   Color[u]:=Gray;
   Push(u);//đẩy u vào stack
   For ∀v∈V; (u,v)∈E do
```

```
    If  Color[v]=Gray then //đỉnh v đã thăm rồi
        Begin
            Low[u]:=Min(Low[u],Number[v]);
            End
    Else
            If  Color[v]=White  then  //đỉnh  v  chưa
        thăm
        Begin
            Tarjan(v);
            Low[u]:=Min(Low[u],Low[v]);
        End;
  If Low[u]>=Number[u] then //u là chốt
    Begin
        //thông báo thành phần liên thông
        Repeat
            v:=pop;
                Output→v
                Color[v]:=Black;
                //xóa  các  đỉnh  trong  một  tplt  vừa
            tìm được
        Until v=u;
    End;
End;
BEGIN
  Time:=0;
  For i:=1 to n do Number[i]:=0;
  For i:=1 to n do
    If Number[i]=0 then
        Tarjan(i);
END.
```

c. Một số ví dụ:

Bài Truyền tin (SPOJ)

Một lớp gồm N học sinh, mỗi học sinh cho biết những bạn mà học sinh đó có thể liên lạc được (chú ý liên lạc này là liên lạc một chiều : u có thể gửi tin tới v nhưng v thì chưa chắc đã có thể gửi tin tới u).

Thầy chủ nhiệm đang có một thông tin rất quan trọng cần thông báo tới tất cả các học sinh. Để tiết kiệm thời gian, thầy chỉ nhắn tin tới 1 số học sinh rồi sau đó nhờ các học sinh này nhắn lại cho tất cả các bạn mà các học sinh đó có thể liên lạc được, và cứ lần lượt như thế làm sao cho tất cả các học sinh trong lớp đều nhận được tin .

Hãy tìm một số ít nhất các học sinh mà thầy chủ nhiệm cần nhắn.

Input

- Dòng đầu là N, M (N <= 800, M là số lượng liên lạc 1 chiều)
- Một số dòng tiếp theo mỗi dòng gồm 2 số u, v cho biết học sinh u có thể gửi tin tới học sinh v

Output

- Gồm 1 dòng ghi số học sinh cần thầy nhắn tin.

Example

Input	Output
12 15 1 3 3 6 6 1 6 8 8 12 12 9 9 6 2 4 4 5 5 2 4 6 7 10 10 11 11 7 10 9	2

Hướng dẫn:

- Liệt kê các thành phần liên thông mạnh của đồ thị
- Xây dựng đồ thị mới:
  + Mỗi đỉnh là một thành phần liên thông mạnh

+ Mỗi cung là cung đồ thị ban đầu mà nối từ thành phần liên thông mạnh này sang thành phân liên thông mạnh kia.

- Trên đồ thị mới, ta tìm số đỉnh không có cung đi vào. Đó chính là kết quả của bài toán.

Chương trình

```
uses    math;
const
        fi='';
        fo='';
        maxN=800+5;
        oo=maxn+5;
type
        TColor=(White,Gray,Black);
        TEdge=record
                u,v:longint;
        end;
var
        top,ans,n,m,time,res:longint;
        a:array[0..maxN,0..maxN] of boolean;
        color:array[0..maxN] of TColor;
        dd,s,number,low:array[0..maxN] of longint ;
        e:array[0..maxN*maxN] of TEdge;
        count:array[0..maxN] of boolean;
procedure read_input;
var     i,u,v:longint;

begin
        fillchar(a,sizeof(a),false);
        assign(input,fi);
        reset(input);
        readln(n,m);
        for i:=1 to m do
                begin
                        readln(u,v);
                        a[u,v]:=true;
```

```
                        e[i].u:=u;
                        e[i].v:=v;
                end;
        close(input);
end;
procedure write_output;
begin
        assign(output,fo);
        rewrite(output);
        write(res);
        close(output);
end;
procedure Tarjan(u:longint);
var     v:longint;
begin
        time:=time+1;
        number[u]:=time;
        low[u]:=oo;
        color[u]:=gray;
        top:=top+1;
        s[top]:=u;//bo u vao ngan xep
        for v:=1 to n do
           if a[u,v] then
             begin
                if color[v]=Gray then
                        low[u]:=min(low[u],number[v])
                else
                        if color[v]=white then
                                begin
                                        Tarjan(v);

low[u]:=min(low[u],low[v]);
                                end;
             end;
        if low[u]>=number[u] then
          begin
       ans:=ans+1;//dem duoc 1 thanh phan lien thong manh
                repeat
```

```
                v:=s[top]; //lay dinh v ra khoi ngan xep
                     top:=top-1;
                       color[v]:=black;
                       dd[v]:=ans;
                until u=v;
          end;
end;
procedure solve;
var     u,i:longint;
begin
        time:=0;
        for u:=1 to n do
                begin
                        color[u]:=white;
                        count[u]:=true;
                end;
        ans:=0;
        for u:=1 to n do
                if color[u]=white then
                        begin
                                top:=0;
                                Tarjan(u);
                        end;
        //danh dau dinh trong do thi moi co cung di vao
        for i:=1 to m do
                if dd[e[i].u]<>dd[e[i].v] then
                        count[dd[e[i].v]]:=false;
        //dem so dinh trong do thi moi khong co cung di vao
        res:=0;
        for i:=1 to ans do
                if count[i] then
                        res:=res+1;
end;
BEGIN
        read_input;
        solve;
        write_output;
END.
```

Biến đổi số (Mã bài: NUMBER)

Cho M máy biến đổi số được đánh số từ 1 đến M và 1 số nguyên dương N. Hoạt động của máy i được xác định bởi cặp số nguyên dương (a_i,b_i) $(1<=a_i,b_i<=N)$. Máy nhận đầu vào là số nguyên dương a_i và trả lại ở đầu ra số nguyên dương b_i.

Ta nói một số nguyên dương X có thể biến đổi thành số nguyên dương Y nếu hoặc X=Y hoặc tồn tại dãy hữu hạn các số nguyên dương $X= P_1,P_2,...,P_k =Y$ sao cho đối với 2 phần tử liên tiếp P_i và P_{i+1} bất kỳ trong dãy, luôn tìm được 1 trong số các máy đã cho để biến đổi P_i thành P_{i+1}

Cho trước 1 số nguyên dương T (T<=N). Hãy bổ sung thêm 1 số ít nhất các máy biến đổi số để bất kì số nguyên dương nào từ 1 đến N đều có thể biến đổi thành T

Input

- Dòng 1: 3 số nguyên dương N, M, T (1<=N,M,T<=10^4)
- M dòng tiếp theo mỗi dòng chứa 1 cặp số tương ứng với một máy biến đổi số. Các số trên một dòng cách nhau bởi 1 dấu cách

Output

Ghi ra 1 dòng duy nhất chứa 1 số nguyên dương là số lượng máy biến đổi số cần thêm

Example

Input	Output
6 4 5 1 3 2 3 4 5 6 5	**1**

Hướng dẫn:

- Liệt kê các thành phần liên thông mạnh của đồ thị
- Xây dựng đồ thị mới:
  + Mỗi đỉnh là một thành phần liên thông mạnh
  + Mỗi cung là cung đồ thị ban đầu mà nối từ thành phần liên thông mạnh này sang thành phân liên thông mạnh kia.
- Trên đồ thị mới, ta tìm số đỉnh không có cung đi ra. Đó chính là kết quả của bài toán.

Cài đặt giống bài truyền tin, ta sửa đếm số cung đi vào bằng đếm số cung đi ra.

2. Liệt kê các cạnh cầu, đỉnh khớp của đồ thị vô hướng

Tương tự thuật toán Tarjan, ta định nghĩa thêm Low[u] và Number[u]. Hãy để ý cung DFS(u,v) (u là nút cha của v trên cây DFS).

a. Liệt kê các cạnh cầu:

- Nếu từ nhánh DFS gốc v không có cung nào ngược lên phía trên v, có nghĩa là từ một đỉnh thuộc nhánh DFS gốc v đi theo các cung định hướng chỉ đi được tới những đỉnh nội bộ trong nhánh DFS gốc v mà thôi chứ không thể tới được u => (u,v) là một cầu. Vậy (u,v) là một cầu nếu và chỉ nếu Low[v]>=Number[v].
- Thuật toán liệt kê các cầu của đồ thị: (ứng dụng cơ chế tô màu cho các đỉnh của đồ thị: mỗi đỉnh đặc trưng bởi 3 màu: chưa thăm (màu White); đang thăm (màu Gray); thăm xong (màu Black).
- Cài đặt:

```
procedure DFS(u:PointType);
{Global: G, Color, Time, D (Number), L (Low)}
Var
      v : PointType;
      pq:List;
Begin{DFS}
        inc(Time);
        D[u]:=Time;
        L[u]:=oo;//maxlongint
        Color[u]:=Gray;
        pq:=G[u];
        while pq<>nil Do
        begin
           v:=pq^.v;
           If Color[v]=White Then
             begin
                parent[v]:=u;
                DFS(v);
                if L[v]<L[u] then L[u]:=L[v];
             end
```

```
            else
            if
  Color[v]=Gray)and(parent[u]<>v)and(D[v]<L[u]) then
                  L[u]:=D[v];
            pq:=pq^.link;
         end{while};
         if (u<>1)and(L[u]>=D[u]) then
         begin
                  {parent[u]-u là một cạnh cầu}
                  inc(S);
                  E[S].v:=u;
                  E[S].u:=Parent[u];
         end;
         Color[u]:=Black;
End {DFS};
```

- Một số ví dụ:

Nâng cấp đường đi. (Đề thi HSG Nam Định)

Hiện nay nhiều thành phố có cơ sở hạ tầng kém phát triển cho nên cảnh tắc đường rất hay xảy ra. Nhà nước đã có kế hoạch nâng cấp nhiều con đường trong thành phố để giảm thiểu nạn tắc đường. Hàng ngày mọi người vẫn cần phải đi lại trên các con đường nên việc nâng cấp đường cần phải nhanh chóng hoàn thành. Hiện tại, hệ thống giao thông của thành phố ND đều đáp ứng được nhu cầu đi lại từ địa điểm A đến địa điểm B (A, B là hai địa điểm bất kì thuộc thành phố ND). Để đi từ A đến B có thể bằng con đường nối từ A đến B hoặc thông qua một hay nhiều địa điểm khác. Không được đi qua con đường nối từ A đến B nếu con đường đang trong thời gian nâng cấp. Hệ thống giao thông của thành phố bị ngưng trệ nếu tồn tại hai địa điểm A và B mà không thể đi được từ A đến B.

Yêu cầu: Cho biết mạng lưới giao thông của thành phố ND có n địa điểm và m con đường nối trực tiếp giữa hai địa điểm. Hãy xác định số lượng s các con đường mà khi nâng cấp thì hệ thống giao thông của thành phố bị ngưng trệ (để đơn giản ta coi như trong một đơn vị thời gian chỉ có không quá một con đường được tiến hành nâng cấp).

Dữ liệu vào: Từ tệp văn bản SD.INP, có cấu trúc:

- Dòng 1: chứa 2 số n và m đều nguyên dương (n≤100000; m≤200000).

- Trong m dòng tiếp theo, mỗi dòng chứa hai số u và v; thể hiện có con đường nối trực tiếp từ địa điểm u đến địa điểm v.

Dữ liệu ra: Đưa ra tệp văn bản SD.OUT, chứa duy nhất một số s tìm được theo yêu cầu.

Ví dụ về dữ liệu vào /ra:

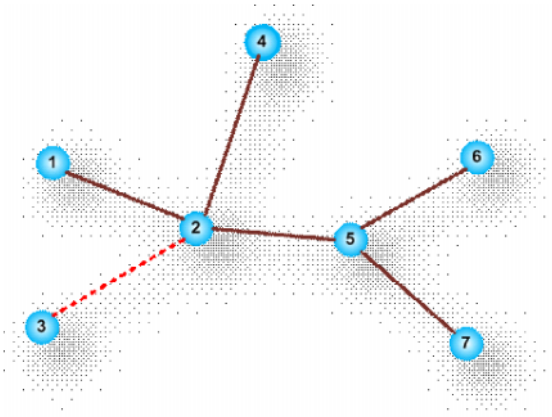
SD.INP	SD.OUT
5 5 1 2 1 3 1 4 2 3 4 5	2

Hướng dẫn: Đếm số cạnh cầu của đồ thị (cài đặt thuật toán như trên)

TÀU ĐIỆN

Rạng Đông là một thành phố không lớn nhưng có một mạng giao thông công cộng bằng tàu điện rất thuận tiện và hợp lý.Từ hai bến đỗ bất kỳ có thể đi tới nhau bằng tàu điện và chỉ có một cách đi duy nhất.Như vậy mạng tàu điện tạo thành một cây mà nút là các bến đỗ và cạnh là tuyến đường tàu.

Ban đầu, giữa hai bến đổ bất kỳ có ít nhất một tuyến tàu điện chạy. Nhưng với sự phát triển của thành phố và các loại phương tiện giao thông công cộng khác một số tuyến bị hủy bỏ vì gần như không còn hành khách. Điều này dẫn đến việc một số đoạn đường sắt không có tàu nào chạy qua.Chính quyền thành phố quyết định tháo dỡ những đoạn đường này.



Yêu cầu: Cho số nguyên n ($2 \le n \le 100\,000$) – số bến đỗ. Các bến được đánh số từ 1 đến n. Cho (n-1) cặp số b_i, e_i xác định các cặp bến đỗ có đường tàu nối trực tiếp. Cho m – số tuyến đang hoạt động ($0 \le m \le 100\,000$) và m cặp số (x, y), mỗi cặp số xác định một tuyến đi từ x tới y theo đường ngắn nhất. Hãy xác định số các đoạn đường cần tháo dỡ.

Dữ liệu: Vào từ file văn bản TRAM.INP:

- Dòng đầu tiên chứa số nguyên n,

- Dòng thứ i trong n-1 dòng sau chứa 2 số nguyên b_i và e_i,
- Dòng tiếp theo chứa số nguyên m,
- Mỗi dòng trong m dòng sau chứa 2 số nguyên x và y.

Kết quả: Đưa ra file văn bản TRAM.OUT một số nguyên – số các đoạn đường cần tháo dỡ.

Ví dụ:

TRAM.INP		TRAM.OUT
7 1 2 2 3 2 4 5 2 5 6 7 5 3 1 7 2 4 7 6		1

Hướng dẫn: (bài này có thể dùng thuật toán LCA-tìm cha chung gần nhất, ta sẽ không bàn đến)

- Đồ thị ban đầu có n đỉnh, n-1 cạnh (đồ thị liên thông, không có chu trình).
- m cặp (x,y), mỗi cặp (x,y) thêm cạnh (x,y) vào đồ thị ban đầu, ta sẽ được một chu trình.

=> Trên đồ thị này ta đếm số cạnh cầu chính là số tuyến đường phải bỏ.

Chú ý: đếm số cạnh cầu trên đa đồ thị

b. Liệt kê các đỉnh khớp:

- Nếu từ nhánh DFS gốc v không có cung nào ngược lên phía trên u, tức là nếu bỏ u đi thì từ v không có cách nào lên được các tiền bối của u. Điều này chỉ ra rằng nếu u không phải là nút gốc của một cây DFS thì u là khớp. Vậy nếu u không phải là nút gốc của một cây DFS thì u là khớp nếu và chỉ nếu Low[v]>=Number[u].
- Thuật toán liệt kê các đỉnh khớp của đồ thị: (ứng dụng cơ chế tô màu cho các đỉnh của đồ thị: mỗi đỉnh đặc trưng bởi 3 màu: chưa thăm (màu White); đang thăm (màu Gray); thăm xong (màu Black).

Chú ý: gốc của cây DFS thì là khớp nếu và chỉ nếu có từ hai nhánh con trở lên.

- Cài đặt (giống cài đặt tìm cạnh cầu, ta chỉ sửa điều kiện)

```
procedure DFS(u:longint);
var  pq:Graph;
     v:longint;
begin
     Time:=Time+1;
     d[u]:=Time;
     l[u]:=maxlongint;
     Color[u]:=gray;
     pq:=G[u];
     while pq<>nil do
         begin
             v:=pq^.v;
             if p[u]<>v then
                 begin
                    if color[v]=white then
                        begin
                           p[v]:=u;
                     con[u]:=con[u]+1;//đếm số con của u
                           DFS(v);
                           l[u]:=min(l[u],l[v]);
                 if(p[u]<>-1)and(l[v]>=d[u])and   not
  dd[u] then
                                    begin
                                        khop:=khop+1;
                                        dd[u]:=true;//u  là
  khớp
                                    end;
                        end
                       else
                          if color[v]=gray then
                               l[u]:=min(l[u],d[v]);
                 end;
                 pq:=pq^.link;
```

```
        end;
     if (p[u]=-1) and (con[u]>1) then
         khop:=khop+1;//nếu nút cha có hai con trở
    lên
     color[u]:=black;
end;
```

- Ví dụ: mã bài **Graph_** tìm khớp và cầu trên Spoj

Xét đơn đồ thị vô hướng G = (V, E) có n(1<=n<=10000) đỉnh và m(1<=m<=50000) cạnh. Người ta định nghĩa một đỉnh gọi là khớp nếu như xoá đỉnh đó sẽ làm tăng số thành phần liên thông của đồ thị. Tương tự như vậy, một cạnh được gọi là cầu nếu xoá cạnh đó sẽ làm tăng số thành phần liên thông của đồ thị.

Vấn đề đặt ra là cần phải đếm tất cả các khớp và cầu của đồ thị G.

Input

+Dòng đầu: chứa hai số tự nhiên n,m.

+M dòng sau mỗi dòng chứa một cặp số (u,v) (u<>v, 1<=u<=n, 1<=v<n) mô tả một cạnh của G.

Output

Gồm một dòng duy nhất ghi hai số, số thứ nhất là số khớp, số thứ hai là số cầu của G

Example

Input	Output
10 12 1 10 10 2 10 3 2 4 4 5 5 2 3 6 6 7 7 3 7 8 8 9 9 7	**4 3**

- Chương trình

```
uses    math;
const
        fi='Graph_.inp';
        fo='graph_.out';
type
        Graph=^Node;
        Node=record
                v:longint;
                link:Graph;
        end;
        Tcolor=(white,gray,black);
var
        n,m,i,cau,khop,u,v,time:longint;
        G:array[0..10000+5] of Graph;
        dd:array[0..10000+5] of boolean;
        con,p,d,l:array[0..10000+5] of longint;
        color:array[0..10000+5] of TColor;
procedure add(u,v:longint);
var     q:Graph;
begin
        new(q);
        q^.v:=v;
        q^.link:=g[u];
        g[u]:=q;
end;
procedure DFS(u:longint);
var     q:Graph;
        v:longint;
begin
        Time:=Time+1;
        d[u]:=Time;
        l[u]:=maxlongint;
        Color[u]:=gray;
        q:=G[u];
        while q<>nil do
            begin
                v:=q^.v;
                if p[u]<>v then
                     begin
```

```
                        if color[v]=white then
                            begin
                               p[v]:=u;
                               con[u]:=con[u]+1;
                               DFS(v);
                               l[u]:=min(l[u],l[v]);
                           if (p[u]<>-1)and(l[v]>=d[u]) and
not dd[u] then
                                  begin
                                      khop:=khop+1;
                                      dd[u]:=true;
                                  end;
                           end
                              else
                                  if color[v]=gray then
                                     l[u]:=min(l[u],d[v]);
                    end;
                    q:=q^.link;
           end;
        if (p[u]<>-1) and (l[u]>=d[u]) then cau:=cau+1;
        if (p[u]=-1) and (con[u]>1) then begin khop:=khop+1;
end;
        color[u]:=black;
end;
begin
        assign(input,fi);
        reset(input);
        assign(output,fo);
        rewrite(output);
        readln(n,m);
        for i:=1 to n do
                begin
                        G[i]:=nil;
                        color[i]:=white;
                        dd[i]:=false;
                        con[i]:=0;
                end;
        for i:=1 to m do
                begin
                        readln(u,v);
```

```
                        add(u,v);
                        add(v,u);
                end;
        cau:=0;
        khop:=0;
        time:=0;
        for i:=1 to n do
                if color[i]=white then
                        begin
                                p[i]:=-1;
                                DFS(i);
                        end;
        write(khop,' ',cau);
        close(input);
        close(output);
end.
```

III. Kết luận

Hiểu rõ được cơ chế hoạt động thuật toán tìm kiếm theo chiều sâu (DFS) bằng đệ quy cho ta cách cài đặt rất ngắn gọn, rõ ràng. Những cải tiến nhỏ trong thuật toán có thể đem lai nhiều điều thú vị, giải quyết được nhiều lớp bài toán khác nhau. Trong phạm vi chuyên đề này tôi không thể trình bày hết được những ứng dụng của DFS, nhưng phần nào cho thấy được tầm quan trọng của DFS.

Tài liệu tham khảo:

- Tài liệu chuyên tin quyển 1 – Hồ Sĩ Đàm (chủ biên)
- Toán rời rạc – Nguyễn Đức Nghĩa – Nguyễn Tô Thành
- Website http://vn.spoj.com

Chuyên đề xếp loại B

TÊN CHUYÊN ĐỀ: CÂY KHUNG CỦA ĐỒ THỊ
MÔN: TIN HỌC
ĐƠN VỊ: THPT CHUYÊN THÁI NGUYÊN

LỜI MỞ ĐẦU

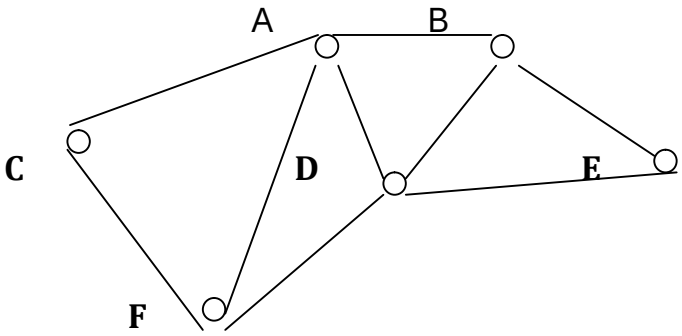
Việc bồi dưỡng học sinh giỏi tin học, tạo nguồn sinh viên giỏi và đáp ứng yêu cầu đào tạo nhân lực chất lượng cao của xã hội là một việc cực kỳ cấp bách trong giai đoạn hiện nay. Vì vậy cùng với các trường chuyên trong vùng chúng tôi luôn trăn trở làm thế nào để nâng cao chất lượng dạy tin học nhất là đối với chương trình chuyên. Điều đó thôi thúc đội ngũ giáo viên chuyên phải tìm tòi, nghiên cứu và sáng tạo.

Trong chương trình tin học chuyên thì đồ thị là vấn đề phong phú nhất, đa dạng nhất, khó nhất... và cũng là nguồn cảm hứng chưa bao giờ cạn không chỉ đối với chúng tôi. Vì thế năm nay chúng ta chọn vấn đề đồ thị làm đề tài nghiên cứu là sự lựa chọn hay. Việc chọn vấn đề cây khung của đồ thị để tìm tòi, nghiên cứu là một trong seri các vấn đề cần nghiên cứu về đồ thị.

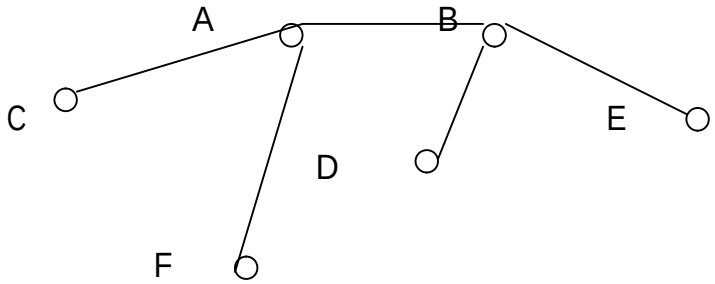
Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Mô hình đồ thị đã được sử dụng từ lâu nhưng ngày nay lại có những ứng dụng hiện đại. Những ý tưởng cơ bản của đồ thị được nhà toán học người Thuỵ Sĩ Leonhard Euler đưa ra từ thế kỷ 18 để giải quyết bài toán các cây cầu ở Konígberg nổi tiếng.

Đồ thị cũng được dùng để giải các bài toán trong nhiều lĩnh vực khác nhau. Chẳng hạn, trong lĩnh vực giao thông có bài toán thực tế sau:

Hệ thống đường giao thông ở một địa phương nào đó được biểu diễn bằng một đơn đồ thị. Để những con đường có thể đi lại được về mùa đông thì cách duy nhất là phải cào tuyết thường xuyên. Chính quyền địa phương muốn cào tuyết trên một số ít nhất các con đường sao cho sao cho luôn có đường thông suốt nối hai thành phố bất kỳ. Có thể làm điều đó bằng cách nào



Rõ ràng là phải cào tuyết trên ít nhất năm con đường đó là (A,C); (A,F); (A,B); (B,D); (B,E). Đây là sơ đồ biểu diễn tập các con đường đó:



Sơ đồ trên cho ta hình ảnh một cây, gồm tất cả các đỉnh của đồ thị biểu diễn hệ thống giao thông và số ít nhất các cạnh nối các đỉnh để hệ thống thông suốt. Đó chính là cây khung (câybao trùm) của đồ thị. Một đồ thị có thể có hơn một cây khung.

Từ bài toán thực tế trên mở ra hai vấn đề:

Thứ nhất, từ đồ thị cho trước, tìm cây khung của nó.

Thứ hai, nếu mỗi cạnh của đồ thị được gán cho một trọng số thì hãy tìm cây khung có tổng trọng số nhỏ nhất.

Trong khuôn khổ văn bản này, chúng tôi xin trình bày cách giải quyết của các vấn đề nêu trên.

1. CÂY KHUNG CỦA ĐỒ THỊ

1.1. Định nghĩa cây

Cây : là một đồ thị hữu hạn, vô hướng, liên thông và không có chu trình.

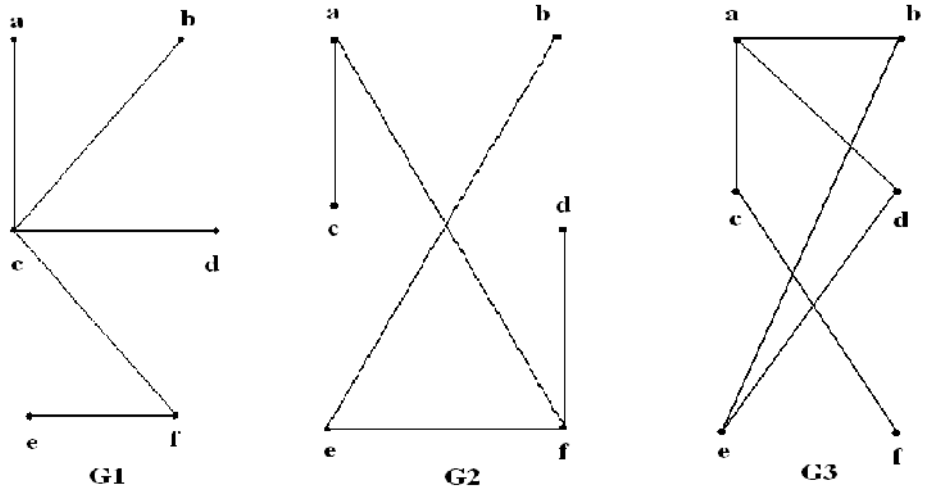
Rừng: là một đồ thị hữu hạn, vô hướng và không có chu trình.

Bụi: Đồ thị G=(X,U) hữu hạn, có hướng là một bụi có gốc $x_1 \in X$ nếu nó có ít nhất hai đỉnh và thoả mãn 3 điều kiện sau:

- Mỗi đỉnh khác x_1 đều là điểm cuối của một cung duy nhất.

- Đỉnh x_1 không là đỉnh cuối của bất kì cung nào.
- Đồ thị không có chu trình.

Ví dụ: Quan sát các đồ thị dưới đây:



Dựa vào định nghĩa của cây ta thấy: G1, G2 là cây; G3 không là cây do có chu trình.

1.2. Tính chất của cây

Định lý 1

Nếu T là đồ thị vô hướng, n đỉnh (n>1) và T có một trong sáu tính chất sau thì T là cây. Mỗi tính chất là một mệnh đề. Khi đó, các mệnh đề sau là tương đương:

(1) T là cây.

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

(3) T có (n-1) cạnh và liên thông.

(4) T liên thông và mỗi cạnh của T đều là cạnh cắt (cầu).

(5) Hai đỉnh bất kì của T được nối với nhau bằng đúng một đường đi đơn.

(6) T không chứa chu trình nhưng nếu thêm một cạnh bất kì vào T thì ta được thêm đúng một chu trình.

Chứng minh định lý:

(1) → (2): T là cây → T không chứa chu trình và có (n-1) cạnh.

- Hiển nhiên T không chứa chu trình (do T là cây).

- Ta chỉ cần chứng minh T có (n-1) cạnh.
- Xét T_n là cây có n đỉnh. Ta sẽ chứng minh quy nạp theo n:
  - n = 2. Cây có 2 đỉnh thì có 1 cạnh. Đúng.
  - Giả sử cây có k đỉnh thì có (k-1) cạnh.
  - Xét T_{k+1} là cây có (k+1) đỉnh. Dễ thấy trong cây này luôn tồn tại ít nhất một đỉnh treo.
  - Loại đỉnh treo này (cùng với cạnh nối) ra khỏi T_{k+1} ta được đồ thị T' có k đỉnh. Dễ thấy, T' vẫn liên thông và không có chu trình (do T_{k+1} không có chu trình).
  - Suy ra T' là cây. Theo giả thiết quy nạp, T' có k đỉnh thì sẽ có (k-1) cạnh. Vậy T_{k+1} có k cạnh (đpcm).

(2) → (3): T không có chu trình và có (n-1) cạnh → T liên thông và có (n-1) cạnh.

- Hiển nhiên T có (n-1) cạnh (theo giả thiết).
- Ta chỉ cần chứng minh T liên thông.
- Giả sử T có k thành phần liên thông với số đỉnh lần lượt là $n_1, n_2,..., n_k$.
- Khi đó mỗi thành phần liên thông của T sẽ là một cây và sẽ có số cạnh lần lượt là $n_1-1, n_2-1,..., n_k-1$.
- Suy ra, số cạnh của T sẽ là: $n_1-1 + n_2-1 +...+ n_k-1 = n-k$.
- Theo giả thiết, số cạnh của cây là (n-1). Từ đó suy ra k=1 hay T chỉ có một thành phần liên thông. Suy ra, T liên thông (đpcm).

(3) → (4): T có (n-1) cạnh và liên thông → T liên thông và mỗi cạnh của T đều là cạnh cắt (cầu).

- Hiển nhiên T liên thông (theo giả thiết).
- Ta chỉ cần chứng minh mỗi cạnh của T đều là cạnh cắt.
- Xét (u,v) là cạnh bất kì của T. Nếu bỏ (u,v) ra khỏi T thì ta sẽ được đồ thị T' có n đỉnh và (n-2) cạnh.

- Ta đã chứng minh được đồ thị có n đỉnh và (n-2) cạnh thì không thể liên thông.
- Vậy nếu bỏ cạnh (u,v) ra thì sẽ làm mất tình liên thông của đồ thị. Suy ra (u,v) là cạnh cắt (đpcm).

(4) → (5): T liên thông và mỗi cạnh của T đều là cạnh cắt (cầu) → Hai đỉnh bất kì của T được nối với nhau bằng đúng một đường đi đơn.

- Xét u,v là 2 đỉnh bất kì trong T.
- Do T liên thông nên luôn tồn tại đường đi giữa u,v. Ta sẽ chứng minh đường đi này là duy nhất.
- Giả sử có hai đường đi đơn khác nhau giữa u và v. Khi đó hai đường đi này sẽ tạo thành chu trình.
- Suy ra các cạnh trên chu trình này sẽ không thể là các cạnh cắt được.
- Vậy giữa u và v sẽ chỉ được tồn tại đúng một đường đi đơn (đpcm).

(5) → (6): Hai đỉnh bất kì của T được nối với nhau bằng đúng một đường đi đơn → T không chứa chu trình nhưng nếu thêm một cạnh bất kì vào T thì ta được thêm đúng một chu trình.

- T không thể có chu trình, vì nếu có chu trình thì 2 đỉnh trên chu trình này sẽ có hai đường đi đơn khác nhau → Mâu thuẫn với giả thiết.
- Giả sử ta thêm vào T cạnh (u,v) bất kì (trước đó không có cạnh này trong T).
- Khi đó cạnh này cùng với đường đi duy nhất giữa u và v trong T sẽ tạo thành một chu trình (vì nếu tạo hai chu trình thì chứng tỏ trước đó có hai đường đi khác nhau giữa u và v → Mâu thuẫn với giả thiết).

(6) → (1): T không chứa chu trình nhưng nếu thêm một cạnh bất kì vào T thì ta được thêm đúng một chu trình → T là cây.

- Hiển nhiên T không chứa chu trình.
- Giả sử T không liên thông. Khi đó T sẽ có nhiều hơn một thành phần liên thông.

- Suy ra, nếu thêm vào một cạnh bất kì giữa hai đỉnh thuộc hai thành phần liên thông khác nhau sẽ không tạo thêm một chu trình nào → Mâu thuẫn với giả thiết.
- Vậy T phải liên thông → T là cây (đpcm).

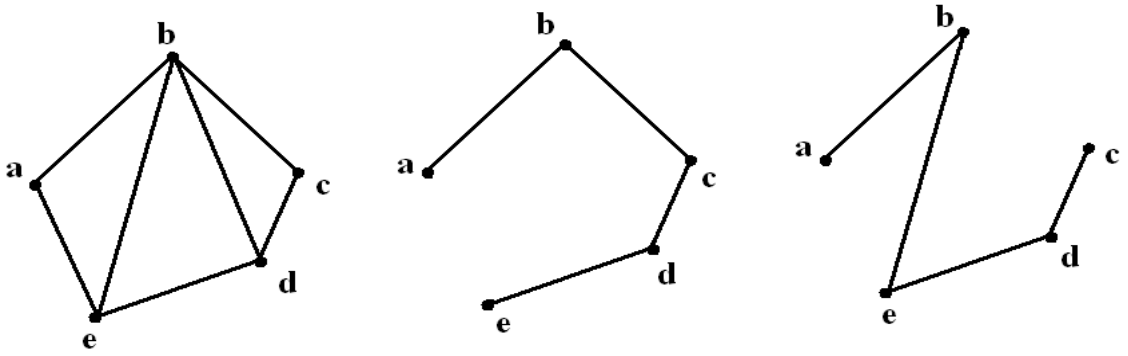
Định lí 2

Một bụi nếu thay các cung bằng cạnh thì thành cây.

1.3. Cây khung của đồ thị

Định nghĩa cây khung: Cho đồ thị vô hướng G=(X,E) liên thông, có n đỉnh (n>1). Mỗi đồ thị bộ phận của G nếu là cây thì được gọi là cây khung của đồ thị G (hoặc cây bao trùm).

Ví dụ:



Đồ thị và các cây khung của nó.

Định lí: Mọi đồ thị vô hướng có số đỉnh n>1 liên thông khi và chỉ khi nó có cây khung.

Chứng minh định lí:

- Nếu G có chứa cây khung thì do tính chất của cây khung là liên thông và cây khung chứa tất cả các đỉnh của G. Suy ra các đỉnh của G luôn được nối với nhau hay G liên thông.
- Xét G liên thông. Giả sử trong G còn tồn tại chu trình, xoá bớt một cạnh trong chu trình này, khi đó đồ thị vẫn còn liên thông. Nếu vẫn còn chu trình thì lặp lại bước trên. Cứ thế cho đến khi không còn chu trình nữa. Khi đó ta sẽ được cây khung.

1.4. Thuật toán tìm cây khung

1.4.1. Bài toán

Cho đồ thị G liên thông, vô hướng, hãy tìm một cây khung của nó.

- Dữ liệu: số đỉnh và danh sách các cạnh của đồ thị.
- Kết quả: các cạnh của cây khung

CK.inp	CK.out
6 1 6 2 3 4 5 2 5 1 5 1 2 6 5 5 3	1 6 2 3 4 5 2 5 1 5

1.4.2. Thuật toán

Thuật toán 1:

Ý tưởng: Duyệt và thăm các đỉnh, mỗi đỉnh một lần. Vì đồ thị liên thông nên thăm đủ n đỉnh (cùng lúc với đã qua n-1 cạnh , ta được cây khung). Có thể dùng thuật toán DFS hoặc BFS để thăm các đỉnh.

Cài đặt:

```
program tim_caykhung_bang_DFS;
const   fi='CayKhung.inp';
        fo='CKhung.DFS.out';
        MN=1000;
var     A:array[1..MN,1..MN] of longint;
        vs:array[1..MN] of boolean;
        n,m:integer;

procedure nhap;
var i,j:integer;
begin
     assign(input,fi);
     reset(input);
     readln(n,m);
```

```
     while not seekeof(input) do                //doc
cac canh cua do thi
       begin
          readln(i,j);
          a[i,j]:=1;
          A[j,i]:=1;
       end;
     close(input);
end;
//---------------------
procedure DFS_VS(i:integer);
var j:integer;
begin
  vs[i]:=true;
  for j:=1 to n do
      if not vs[j] and (a[i,j]=1) then
         begin
            writeln(i,' ',j);
            DFS_VS(j);
         end;
end;
//---------------------
Begin
  nhap;
  fillchar(vs, sizeof(vs),false);
  assign(output,fo); rewrite(output);
  DFS_VS(1);
  close(output);
end.
```

Thuật toán 2: *Hợp nhất dần các vùng liên thông*

Ý tưởng: Mỗi lần hợp nhất hai vùng liên thông khác nhau bằng một cạnh nối hai vùng này thì nạp cạnh đó vào cây khung đang hình thành. Quá trình chấm dứt khi nạp đủ (n-1) cạnh.

Thực hiện cụ thể:

Bước 1: Coi mỗi đỉnh thuộc một vùng có mã vùng là v[i]=i, số cạnh đã nạp vào cây khung là sl=0.

Bước 2: Duyệt tất cả các cạnh của đồ thị:

- Nếu sl=n-1 thì dừng vòng lặp duyệt.
- Nếu cạnh (i,j) có đỉnh i và j khác mã vùng (v[i]≠v[j]) thì:
  - Nếu v[i]<v[j]: tất cả các đỉnh cùng mã vùng với j được gán lại mã vùng là v[i], nạp vào cây khung cạnh (i,j), tăng biến sl một đơn vị.
  - Nếu v[i]>v[j]: tất cả các đỉnh cùng mã vùng với i được gán lại mã vùng là v[j], nạp vào cây khung cạnh (i,j), tăng biến sl một đơn vị.

Cài đặt:

```
program CayKhung;
const   fi='CayKhung.inp';
        fo='CK.out';
var     b,dau,cuoi:array[1..10000] of longint;
        i,j,k,n,t,sc:longint;
        f:text;
procedure nhap;
begin
     assign(f,fi);
     reset(f);
     readln(f,n);
     for i:=1 to n do
          b[i]:=i;
     sc :=0;
     while not eof(f) do              {doc cac canh
  cua do thi}
     begin
          readln(f,i,j);
              if b[i] <> b[j] then {khac ma vung
  lien thong}
                 begin
                   inc(sc);                {tang so
  canh}
```

```
                dau[sc] := i;      {nap them mot
  canh vao 2 mang dau va cuoi}
                cuoi[sc] := j;
                if b[i]<b[j] then
                    for t:=1 to n  do {hop nhat 2
  vung lien thong}
                        if     b[t]=b[j]     then
  b[t]:=b[i];
                if b[i]>b[j] then
                    for t:=1 to n do
                        if     b[t]=b[i]     then
  b[t]:=b[j];
              end;
          if sc = n-1 then break;      {nap du n-1
  canh thi dung lai}
     end;
     close(f);
end;
procedure xuat;
begin
     assign(f,fo);
     rewrite(f);
     for i := 1 to n -1 do
      writeln(f,dau[i],' ',cuoi[i]);
     close(f);
end;
begin
     nhap;
     xuat;
    end.
```

Thuật toán 3: Hợp nhất dần các cây

Ý tưởng: Mỗi lần hợp nhất hai cây có gốc khác nhau bằng một cạnh của đồ thị (nối hai đỉnh thuộc hai cây này)thì cạnh đó được xác nhận là một cạnh của cây khung đang hình thành. Quá trình kết thúc khi nạp đủ n-1 cạnh của cây khung.

Cài đặt:

```
program CayKhung; {su dung thuat toan hop nhat hai
cay}
const   fi='Caykhung.inp';
        fo='CKhung.out';
        MN=5000;

var     cha,dau,cuoi:array[1..MN] of integer;
        n,m,socanh:longint;
//------------------------
function root(x:integer):integer;     //tim goc cay
chua dinh x
var     i:integer;
begin
  i:=x;
  while cha[i]>0 do i:=cha[i];
  exit(i);
end;
//------------------------
procedure Union(x,y:integer);         // hop nhat hai
cay oc x, goc y
var  temp:integer;
begin
  temp:=cha[x]+cha[y];
  if cha[x]>cha[y] then               // cay chua
dinh x co it nut hon
     begin
       cha[x]:=y;                    //tam coi y la
cha cua x trong cay hop nhat
       cha[y]:=temp;                 //goc moi cua 2
cay la y cay chua dinh x co it nut hon
     end
  else
     begin                           //cay chua dinh
y co it nut hon
       cha[y]:=x;
```

```
        cha[x]:=temp;
     end;
end;
//------------------------

procedure nhap_taocay;
var     i, x, y,r1,r2:longint;
begin
     assign(input,fi);
     reset(input);
     readln(n);
     for i:=1 to n do cha[i]:=-1;      // moi dinh la
cay co goc la chinh no
     socanh :=0;
     while not seekeof(input) do       //doc cac canh
cua do thi
     begin
          if socanh=n-1 then exit;    //la cay khung,
ket thuc
          readln(x,y);
          r1:=root(x);
          r2:=root(y);
          if  r1 <> r2 then           //hai cay co
goc khac nha
                begin
                  inc(socanh);        //tang so canh
                  dau[socanh] := x;   //nap them mot
canh vao 2 mang dau va cuoi
                  cuoi[socanh] := y;
                  union(r1,r2);       //hop nhat hai
cay
                end;
     end;
     close(input);
end;
procedure xuat;
```

```
var     i:integer;
begin
     assign(output,fo);
     rewrite(output);
     writeln(socanh);
     for i := 1 to n -1 do
      writeln(dau[i],' ',cuoi[i]);
     close(output);
end;
begin
     nhap_taocay;
     xuat;
end.
```

1.5. Thuật toán tìm cây khung ngắn nhất

1.5.1. Bài toán

Định nghĩa: Cho đồ thị G vô hướng, liên thông và có trọng số không âm. Cây khung ngắn nhất của đồ thị G là cây khung có tổng trọng số trên các cạnh của nó nhỏ nhất (gọi là trọng số của cây khung ngắn nhất).

Bài toán: Cho đồ thị G vô hướng, liên thông và không có trọng số âm. Tìm cây khung ngắn nhất của đồ thị G.

Dữ liệu: số đỉnh, danh sách các cạnh kèm theo trọng số của cạnh (mỗi dòng mô tả cạnh gồm 3 số i, j, l có nghĩa cạnh nối đỉnh i và j thì có trọng số là l).

Kết quả: các cạnh của cây khung ngắn nhất và trọng số của cây khung ngắn nhất.

Ck.inp	Ck.out
5 1 2 2 1 3 2 1 4 1 1 5 4 2 3 5 2 4 3 2 5 3 3 4 2 3 5 4 4 5 8	Cay khung la: (1,4) (3,4) (1,2) (2,5) Tong trong so : 8

Để giải bài toán cây khung ngắn nhất có 2 thuật toán thông dụng: Kruskal và Prim

1.5.2. Thuật toán Kruskal

Ý tưởng: Nạp dần các cạnh ngắn nhất và cây khung nếu cạnh ấy không tạo thành chu trình với các cạnh đã nạp.

Thuật toán:

- Sắp xếp các cạnh tăng dần (thường dùng Quicksort)
- Lần lượt kết nạp các cạnh có trọng số nhỏ nhất trong các cạnh còn lại vào cây nếu sau khi kết nạp cạnh này không tạo thành chu trình trong cây. Để thực hiện yêu cầu này, ta có thể sử dụng thuật toán hợp nhất các vùng liên thông ở trên. Quá trình này dừng khi kết nạp được n-1 cạnh vào cây.

Cài đặt:

```
program Kruskal;
const fi='ck.inp';
      fo='ck.out';
type canh = record
        d,c,l : longint;
     end;
var b:array[1..10000] of longint;
    a,ck : array[1..10000] of canh;
    i,j,k,n,m,t,sc,sum:longint;
    f:text;
Procedure QuickSort(dau, cuoi : longint);
var x, L, R : longint;
    tmp : canh;
begin
    x := a[(dau+cuoi) div 2].l; L := dau; R :=
  cuoi;
    repeat
        while a[L].l < x do inc(L);
        while a[R].l > x do dec(R);
        if L <= R then
            begin
```

```
                tmp := a[L];a[L] := a[R]; a[R]
  := tmp;
                inc(L);dec(R);
             end;
    until L > R;
    if R > dau then QuickSort(dau,R);
    if L < cuoi then QuickSort(L,cuoi);
end;
procedure nhap;
begin
     assign(f,fi);  reset(f);
     readln(f,n);
     m := 0;
     while not eof(f) do {doc cac canh cua do
  thi}
     begin
          inc(m);
          readln(f,a[m].d,a[m].c,a[m].l);
     end;
     close(f);
end;
procedure xuli;
begin
  QuickSort(1,m);
  for i:=1 to n do
          b[i]:=i;
  sc :=0;sum := 0;
  for k := 1 to m do
    begin
      i := a[k].d;j:= a[k].c;
      if b[i] <> b[j] then {khac ma vung lien
  thong}
        begin
          inc(sc);          {tang so canh}
          ck[sc] := a[k];   {them canh vao cay
  khung}
```

```
            sum := sum + a[k].l;
            if b[i]<b[j] then
               for t:=1 to n  do {hop nhat 2 vung
  lien thong}
                  if b[t]=b[j] then b[t]:=b[i];
            if b[i]>b[j] then
               for t:=1 to n do
                  if b[t]=b[i] then b[t]:=b[j];
         end;
      if sc = n-1 then break; {nap du n-1 canh
  thi dung lai}
    end;
end;
procedure xuat;
begin
     assign(f,fo);
     rewrite(f);
     writeln(f,'Cay khung la: ');
     for i := 1 to n -1 do
      writeln(f,'
  (',ck[i].d,',',ck[i].c,')');
     writeln(f,'Tong trong so : ',sum);
     close(f);
end;
begin
     nhap;
     xuli;
     xuat;
end.
```

1.5.3. Thuật toán Prim

Ý tưởng: Nạp dần tập cách đỉnh vào cây khung. Mỗi lần chọn một đỉnh chưa nạp là đỉnh kề và gần các đỉnh đã nạp nhất.

Thuật toán:

Bước 1: Nạp một đỉnh đầu tiên vào cây khung (thường là đỉnh 1)

Bước 2: Lần lượt nạp n-1 đỉnh còn lại (tương ứng với n-1 cạnh) vào cây khung bằng cách: mỗi lần chọn một cạnh có trọng số nhỏ nhất mà một đầu của cạnh đã thuộc cây, đầu kia chưa thuộc cây (nghĩa là chọn một đỉnh gần các đỉnh đã nạp nhất)

Cài đặt

```
program Prim;
const max=100;
      f1='ck.inp';
      f2='ck.out';
var a: array[1..max,1..max] of integer;
    d1,d2,d:array[1..max] of integer;
    n: integer;
procedure nhap;
var g:text;
    i,j,x:integer;
begin
     assign(g,f1); reset(g);
     readln(g,n);
     while not seekeof(g) do
     begin
          readln(g,i,j,x);
          a[i,j]:=x; a[j,i]:=x;
     end;
     close(g);
end;
procedure timcanh( var i,j:integer);
{Tim canh i, j ngan nhat}
var x,y,min:integer;
begin
     min:=maxint;
     for x:=1 to n do
         if d[x]=1 then
            for y:=1 to n do
                if d[y]=0 then
```

```
                    if  (a[x,y]>0)  and  (a[x,y]<min)
  then
            begin
                 i:=x;
                 j:=y;
                 min:=a[x,y];
            end;
end;
procedure prim;
var i,j,k:integer;
    begin
         for i:=1 to n do d[i]:=0;
         d[1]:=1;
         for k:=1 to n-1 do
         begin
              timcanh(i,j);
              d[j]:=1;                        d1[k]:=i;
  d2[k]:=j;
         end;
    end;
procedure ghi;
var g:text; i,tong: integer;
begin
     assign(g,f2);     rewrite(g);
     tong:=0;
     writeln(g,'Cay khung la : ');
     for i:=1 to n-1 do
     begin
          writeln(g,d1[i],' ',d2[i]);
          tong:=tong+a[d1[i],d2[i]];
     end;
     writeln(g,'Tong trong so: ',tong);
     close(G);
end;
begin
     nhap; prim; ghi;
```

end.

2. MỘT SỐ BÀI TOÁN ỨNG DỤNG

Bài toán 1. Mạng rút gọn

Một hệ thống gồm N máy tính được nối thành một mạng có M kênh nối, mỗi kênh nối hai máy tính trong mạng, giữa hai máy tính có không quá một kênh nối. Các máy tính được đánh số từ 1 đến N, các kênh nối đánh số từ 1 đến M. Việc truyền tin trực tiếp có thể thực hiện được đối với hai máy có kênh nối. Các kênh nối trong mạng được chia thành ba loại 1, 2 và 3. Ta nói giữa hai máy a và b trong mạng có đường truyền tin loại k (k=1 hoặc k=2) nếu tìm được dãy các máy ($a=v_1, v_2,...,v_p=b$) thoả mãn điều kiện: giữa hai máy v_i và v_{i+1} hoặc có kênh nối loại k hoặc có kênh nối loại 3 (i=1, 2,..., p-1).

Yêu cầu: Cần tìm cách loại bỏ khỏi mạng một số nhiều nhất kênh nối nhưng vẫn đảm bảo luôn tìm được cả đường truyền loại 1 lẫn đường truyền tin loại 2 giữa hai máy bất kỳ trong mạng.***Dữ liệu*** vào từ tệp văn bản MRG.INP như sau:

- Dòng đầu tiên chứa hai số N, M (N≤500); M≤10000).
- Dòng thứ i trong M dòng tiếp theo chứa ba số nguyên dương u, v, s cho biết kênh thứ i nối hai máy u và v thuộc loại s.

Kết quả ghi ra tệp văn bản MRG.OUT gồm:

- Dòng đầu tiên ghi số r là số kênh cần loại bỏ.
- Nếu r=-1 thì có nghĩa là trong mạng đã cho tồn tại hai máy không có đường truyền loại 1 hoặc loại 2.
- nếu r=0 có nghĩa là mạng có đường truyền thoả mãn nhưng số kênh loại bỏ bằng 0.
- Nếu r>0 thì r dòng tiếp theo, mỗi dòng ghi chỉ số của một kênh cần loại bỏ.

Các số trên cùng một dòng của các tệp dữ liệu và tệp kết quả cách nhau ít nhất một dấu cách.

Ví dụ:

MRG.INP	MRG.OUT
5 7 1 2 3 2 3 3 3 4 2 5 3 2 5 4 1 5 2 2 1 5 1	2 6 7

Cách giải

+ Xây dựng đồ thị vô hướng với mỗi đỉnh là một máy tính, có N đỉnh. Mỗi cạnh là một kênh trong M kênh, có M cạnh; trọng số trên cạnh là loại kênh (1, 2, 3).
+ Dùng thuật toán hợp nhất dần các cây (Thuật toán 3) để tìm cây khung. Cụ thể như sau:
  - Vì đường đi loại 1 và 2 theo yêu cầu phải chứa kênh loại 3 nên ta tìm rừng cây chỉ gồm những cạnh loại 3, gọi là (R3) .
  - Nếu rừng cây đó là cây khung (tức là có n-1 cạnh) thì bài toán có nghiệm và loại bỏ tất cả các cạnh loại 1 và loại 2.
  - Nếu rừng cây đó chưa là cây khung thì phải xem xét bổ sung cạnh loại 1 hoặc loại 2 vào rừng cây đó để mạng có đường truyền loại 1 hoặc loại 2. Tiến hành các việc sau:
    1. Cùng với R3, xét thêm các cạnh loại 1, thực hiện lại thuật toán 3 để xem có tạo thành cây khung (chỉ gồm cạnh loại 1 và 3) không. Nếu không là cây khung thì vô nghiệm (r=-1); nếu có thì thực hiện tiếp bước 2:
    2. Cùng với R3, xét thêm các cạnh loại 2, thực hiện tiếp thuật toán 3 xem có tạo thành cây khung (chỉ gồm cạnh loại 3 và 2) không. Nếu không thì vô nghiệm (r=-1); Nếu có cây khung thì

bài toán có nghiệm, các cạnh cần loại bỏ là các cạnh loại 1 và 2 không thuộc cây khung tìm thấy ở trên. Để thực hiện việc này cần đánh dấu các cạnh đã được nạp vào cây khung.

Văn bản chương trình

```
program mang_rut_gon;
const   fi='MRG.IN2';
        fo='MRG.OUT';
        MN=500;
        MM=10000;
type    canh=record  u,v:integer; w:shortint end;
var     m,n:integer;
        socanh, lsc: integer;           //so canh,
luu so canh
        l, ll:array[1..MN] of integer;  //nhan cua
dinh, luu nhan ddinh
        ds:array[1..MM] of canh;        //danh sach
canh
        caykhung:boolean;               //co la cay
khung hay khong
//----------------------
procedure readf;
var     i:integer;
begin
  assign(input,fi); reset(input);
  readln(n,m);
  for i:=1 to m do
     with ds[i] do readln(u,v,w);
  close(input);
  for i:=1 to n do l[i]:=-1;             // moi cay
co goc la chinh no

end;
//-----------------------
function root(u:integer):integer;              //tra
ve goc cay chua u
```

```
begin
  while l[u]>=0 do u:=l[u];
  exit(u);
end;
//-----------------------
procedure union(r1,r2:integer);         //hop nhat
hai cay co goc la r1, r2
var     x:integer;
begin
  x:=l[r1]+l[r2];                        //nhan cua
goc cay hop nhat
  if l[r1]>l[r2] then
       begin
         l[r1]:=r2;
         l[r2]:=x
       end
  else
       begin
         l[r2]:=r1;
         l[r1]:=x;
       end;
end;
//-----------------------
function KRUSKAL(k:integer):boolean;    //co la cay
khung khi them canh loai k khong
var     i, r1,r2:integer;
begin
  for i:=1 to m do
      with ds[i] do if w=k then
         begin
           r1:=root(u);                 //goc cua cay
chua dinh u
           r2:=root(v);                 //goc cay
chua dinh v
           if r1<>r2 then
              begin
```

```
                w:=-w;                    // danh dau
da canh da nap vao cay
                inc(socanh);
                if socanh=n-1 then exit(true); //la
cay khung
                union(r1,r2);             // chua la
cay thi hop nhat hai cay
              end;
         end;
   exit(false);
end;
//-------------------------
procedure writef;
var i:integer;
begin
   assign(output,fo); rewrite(output);
   if not caykhung then writeln(-1)    //khong co
duong truyen thoa man
   else
       begin
         writeln(m-n+n-2-lsc);
         for i:=1 to m do
            if ds[i].w>0 then writeln(i);
       end;
  close(output);
end;
//------------------------
begin
  readf;
  socanh:=0;
  caykhung:=KRUSKAL(3);
  if not caykhung then
     begin
       lsc:=socanh;
       ll:=l;
       caykhung:=KRUSKAL(1);
```

```
        socanh:=lsc;
        l:=ll;
        caykhung:=caykhung and KRUSKAL(2);
      end;
  writef;
end.
```

Bài toán 2. Mạng giao thông

Theo thiết kế, một mạng giao thông gồm N nút có số hiệu từ 1 đến N (N≤1000).

Chi phí để xây dựng đường hai chiều trực tiếp từ nút i đến nút j bằng A[i,j]=A[j,i]. Hai tuyến đường khác nhau không cắt nhau tại các điểm không là đầu mút. Hiện đã xây dựng được K tuyến đường.

Bài toán đặt ra như sau: Hệ thống đường đã xây dựng có bảo đảm sự đi lại giữa hai nút bát kỳ chưa? Nếu chưa, hãy chọn một số tuyến đường cần xây dựng thêm sao cho:

1. Các tuyến đường sẽ xây dựng thêm cùng với K tuyến đường đã xây dựng bảo đảm sự đi lại giữa hai nút bất kỳ.
2. Tổng kinh phí xây dựng thêm các tuyến đường là ít nhất.

Dữ liệu vào từ tệp văn bản MGT.INP như sau:

- Dòng đầu tiên chứa hai số N, K (N≤500); M≤10000).
- Trong K dòng tiếp theo mỗi chứa hai số nguyên dương là số hiệu hai nút, đó là các tuyến đường đã xây dựng.
- Cuối cùng là N dòng, dòng thứ i ghi N số A[i,1], A[i,2], ..., A[i,N].

Kết quả ghi ra tệp văn bản MGT.OUT gồm:

- Dòng đầu tiên ghi số CP là chi phí xây dựng thêm.
- Nếu CP>0 thì trong N dòng tiếp theo, mỗi dòng ghi hai số là số hiệu hai nút, đó là hai đầu của tuyến đường cần xây dựng thêm.

Các số trên cùng một dòng của các tệp dữ liệu và tệp kết quả cách nhau ít nhất một dấu cách.

Ví dụ:

MGT.INP	MGT.OUT
5 4 1 2 2 3 3 1 4 5 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0	1 3 4

Cách giải:

+ Dựa vào mạng giao thông xây dựng đồ thị vô hướng, có trọng số:
  - Mỗi đỉnh của đồ thị là một nút giao thông (N đỉnh);
  - Mỗi cạnh là đoạn đường trực tiếp nối 2 nút;
  - Trọng số trên cạnh tương ứng với đoạn đường đã xây dựng bằng 0; trên cạnh chưa xây dựng bằng chi phí xây dựng quãng đường tương ứng.
+ Tìm cây khung ngắn nhất trên đồ thị. Nếu trọng số của cây bằng 0, có nghĩa là K đoạn đường đã xây dựng đã đảm bảo sự đi lại giữa hai nút bất kỳ (đồ thị đã liên thông). Ngược lại, nếu trọng số khác 0, thì trên cây có những đoạn đường chưa xây dựng (là những cạnh có trọng số khác 0). Đó chính là những đoạn đường cần xây dựng thêm.
+ Văn bản chương trình

```
Program         MangGiaoThong;
Const   Fi='MGT.INP';
        Fo='MGT.OUT';
        nm=100;
Var     f:text;
        n:integer;
        a:array[1..nm,1..nm] of longint;
        tr:array[1..nm] of integer;
        vs:array[1..nm] of boolean;
        res:longint;
```

```
Procedure       Nhap;
var     i,j:integer;
        k:longint;
begin
        assign(f,fi);
        reset(f);
        readln(f,n,k);
        fillchar(a,sizeof(a),255);
        while k>0 do
          begin
            readln(f,i,j);
            a[i,j]:=0; a[j,i]:=0;
            dec(k);
          end;
        for i:=1 to n do
          for j:=1 to n do
            begin
              read(f,k);
              if a[i,j]=-1 then a[i,j]:=k;
            end;
        close(f);
end;

Procedure       Prim;
var     i,j,sc:integer;
        min:longint;
begin
        for i:=1 to n do tr[i]:=1;
        fillchar(vs,sizeof(vs),false);
        vs[1]:=true; res:=0;
        for sc:=1 to n-1 do
          begin
            min:=High(longint);
            for i:=1 to n do
              if not vs[i] and (a[tr[i],i]<min) then
```

```
                begin
                  min:=a[tr[i],i];
                  j:=i;
                end;
            vs[j]:=true;
            res:=res+a[tr[j],j];
            for i:=1 to n do
              if not vs[i] and (a[j,i]<a[tr[i],i])
then tr[i]:=j;
          end;
end;

Procedure       Xuat;
var     i:integer;
begin
        assign(f,fo);
        rewrite(f);
        writeln(f,res);
        for i:=1 to n do
          if a[tr[i],i]<>0 then writeln(f,tr[i],'
',i);
        close(f);
end;

Begin
                Nhap;
                Prim;
                Xuat;
End.
```

Bài toán 3. Tìm cây khung dài nhất

Các thuật toán Kruslal và Prim không đòi hỏi về dấu của trọng số. Vì vậy ta có thể áp dụng với cây có các cạnh có trọng số dấu tuỳ ý. Do đó, để tìm cây khung dài nhất ta chỉ việc đổi dấu tất cả các trọng số và áp dụng một trong hai thuật toán trên để tìm cây khung nhỏ nhất trên đồ thị mới xây dựng. Cuối

cùng chỉ việc đổi dấu trong số của cây khung. Tính đúng đắn của thuật toán là hiển nhiên vì một số lớn nhất thì dẫn đến số đối của nó phải nhỏ nhất.

Bài toán 4. Tìm mạng điện với sự tin cậy lớn nhất

Bài toán: Cho lưới điện có N nút. đường dây nối nút i với nút j có độ tin cậy là một số thực 0<Pij<1. Độ tin cậy của toàn bộ lưới điện bằng tích độ tin cậy trên tất cả các đường dây. Hãy tìm cây khung với độ tin cậy lớn nhất.

Cách giải:

+ Xây dựng đồ thị vô hướng, có trọng số với số đỉnh là số nút của lưới điện, mỗi cạnh (i,j) của đồ thị là đoạn đường dây nối hai nút i và j. Trọng số trên cạnh (i,j) được gán bằng –ln(Pij).
+ Tìm cây khung nhỏ nhất của đồ thị vừa xây dựng bằng một trong hai thuật toán Kruskal hoặc Prim.
+ Tính đúng đắn của thuật toán được chứng minh nhờ tính chất của logarit như sau:

Ta có công thức toán học: $\ln(x_1.x_2...x_N)=\ln(x_1)+\ln(x_2)+...+\ln(x_N)$. Vì thế nên thực hiện tìm độ tin cậy lớn nhất của mạng điện thay vì tính tích của các trọng số trên cây khung T, ta đưa về tính tổng của của các trọng số mới. Khi đó ta có tổng trọng số trên cây khung ngắn nhất T là một số âm nhỏ nhất. Khi đổi dấu đó là số lớn nhất.

Bài toán 5. Tìm cây khung ngắn nhất trên đồ thị, sử dụng câu trúc HEAP.

+ Biểu diễn đồ thị ban đầu bởi danh sách kề với trọng số. Khi đó có thể cài đặt đồ thị với số đỉnh rất lớn (10000 đỉnh).
+ Áp dụng thuật toán Prim tìm cây khung ngắn nhất trên đồ thị vừa xây dựng, với thao tác tìm đỉnh gần nhất, tức là cạnh liên thuộc có trọng số nhỏ nhất bằng cách sử dụng cấu trúc HEAP. Vì vậy mặc dù với đồ thị có số đỉnh rất lớn chương trình vẫn đáp ứng được yeu cầu về thời gian.

+ Cài đặt cụ thể

```
Program         CayKhungNhoNhat_Heap;
Const   Fi='caykhung.INP';
        Fo='CK_HEAP.INP';
        nm=10000;
        mm=15000;
        vc=10001;
Var     f:text;
```

```
        n,m,nH:longint;
        ke,t:Array[1..mm*2] of longint;
        index:array[0..nm] of longint;
        minc:array[2..nm] of longint;
        c1,c2,c,info,pos:array[1..mm] of longint;
        res:longint;
Procedure       Nhap;
var     i:longint;
begin
        assign(f,fi);
        reset(f);
        readln(f,n,m);
        for i:=1 to m do readln(f,c1[i],c2[i],c[i]);
        close(f);
end;
Procedure       TaoKe(x,y,c:longint);
begin
        ke[index[x-1]]:=y;
        t[index[x-1]]:=c;
        dec(index[x-1]);
end;
Procedure        Chuyen;    {Chuyen tu Ds canh -> Ds
  ke}
var     i:longint;
begin
        fillchar(index,sizeof(index),0);
        for i:=1 to m do
          begin
            inc(index[c1[i]-1]);
            inc(index[c2[i]-1]);
          end;
        for   i:=1   to   n   do   index[i]:=index[i-
  1]+index[i];
        for i:=1 to m do
          begin
            TaoKe(c1[i],c2[i],c[i]);
```

```
            TaoKe(c2[i],c1[i],c[i]);
          end;
end;
Procedure       Swap(var a,b:longint);
var     tg:longint;
begin
        tg:=a; a:=b; b:=tg;
end;

Procedure       UpHeap(i:longint);
var     c,r:longint;
begin
        c:=pos[i];
        while c>1 do
          begin
            r:=c div 2;
            if minc[info[c]]<minc[info[r]] then
              begin
                Swap(info[c],info[r]);
                pos[info[c]]:=c;
                pos[info[r]]:=r;
                c:=r;
              end else break;
          end;
end;
Procedure       DownHeap(i:longint);
var     c,r:longint;
begin
        r:=pos[i];
        while r*2<=nH do
          begin
           c:=r*2;

  if(c<nH)and(minc[info[c]]>minc[info[c+1]])then
  inc(c);
            if minc[info[c]]<minc[info[r]] then
```

```
            begin
              Swap(info[c],info[r]);
              pos[info[c]]:=c;
              pos[info[r]]:=r;
              r:=c;
            end else break;
          end;
end;
Procedure       Insert(i:longint);
begin
        inc(nH);
        info[nH]:=i;
        pos[i]:=nH;
        UpHeap(i);
end;
Procedure       Del;
begin
        pos[info[1]]:=0;
        info[1]:=info[nH];
        pos[info[1]]:=1;
        dec(nH);
        DownHeap(info[1]);
end;
Procedure       Prim;
var     i,j,sc:longint;
begin
        for i:=2 to n do minc[i]:=vc;
        for    i:=index[0]+1    to    index[1]    do
  minc[ke[i]]:=t[i];
        nH:=0;
        for i:=2 to n do Insert(i);
        res:=0;
        for sc:=2 to n do
          begin
            i:=info[1];
            res:=res+minc[i];
```

```
            Del;
            for j:=index[i-1]+1 to index[i] do
              if          (pos[ke[j]]<>0)          and
  (t[j]<minc[ke[j]]) then
                begin
                  minc[ke[j]]:=t[j];
                  UpHeap(ke[j]);
                end;
          end;
end;
Procedure       Xuat;
begin
        assign(f,fo);
        rewrite(f);
        writeln(f,res);
        close(f);
end;
Begin
                Nhap;
                Chuyen;
                Prim;
                Xuat;
End.
```

LỜI KẾT

Trên đây là kết quả của việc tìm hiểu, nghiên cứu đưa vào giảng dạy của nhóm giáo viên tin học trường THPT Chuyên Thái Nguyên về cây khung và cây khung ngắn nhất trên đồ thị. Đó mới chỉ là những kiến thức mang tính cơ sở. Việc cài đặt mặc dù đã có sự tìm tòi, kết hợp việc sử dụng thuật toán với cấu trúc dữ liệu tiên tiến song chắc chắn vẫn chưa đáp ứng được yêu cầu cao trong việc bồi dưỡng họ sinh giỏi. Rất mong được đồng nghiệp các nơi đóng góp ý kiến để chúng tôi hoàn thiện kiến thức về vấn đề này.

Xin trân trọng cảm ơn!

Chuyên đề xếp loại B

MỘT SỐ ỨNG DỤNG THUẬT TOÁN DIJKSTRA

Bùi Thu Hiền – THPT chuyên Thái Bình

LỜI MỞ ĐẦU

Lý thuyết đồ thị là một phần quan trọng trong nội dung chương trình chuyên môn Tin học tại các trường chuyên. Hầu như trong các đề thi học sinh giỏi đều có các bài toán liên quan đến lý thuyết đồ thị, do đó để học sinh có được kết quả cao chúng ta cần trang bị cho các em một nền tảng tốt cũng như các kỹ thuật cài đặt các bài toán cơ bản của lý thuyết đồ thị

Trong tham luận của mình tôi xin đề cập đến ***Một số ứng dụng của thuật toán Dijkstra*** - tìm đường đi ngắn nhất giữa đỉnh s với tất cả các đỉnh của đồ thị có trọng số không âm.

THUẬT TOÁN DIJKSTRA

Bài toán: Cho G = (V, E) là đơn đồ thị có hướng gồm n đỉnh và m cung, trọng số trên các cung không âm. Yêu cầu tìm đường đi ngắn nhất từ đỉnh xuất phát $s \in V$ đến đỉnh đích $f \in V$

Thuật toán Dijkstra (E.Dijkstra - 1959) có thể mô tả như sau:

Bước 1: Khởi tạo

Với đỉnh $v \in V$, gọi nhãn d[v] là độ dài đường đi ngắn nhất từ s tới v. Ban đầu d[v] được khởi gán như trong thuật toán Ford-Bellman (d[s] = 0 và d[v] = $+\infty$ với $\forall v \neq s$). Nhãn của mỗi đỉnh có hai trạng thái tự do hay cố định, nhãn tự do có nghĩa là có thể còn tối ưu hơn được nữa và nhãn cố định tức là d[v] đã bằng độ dài đường đi ngắn nhất từ s tới v nên không thể tối ưu thêm. Để làm điều này ta có thể sử dụng kỹ thuật đánh dấu: Free[v] = TRUE hay FALSE tuỳ theo d[v] tự do hay cố định. Ban đầu các nhãn đều tự do.

Bước 2: Lặp

Bước lặp gồm có hai thao tác:

- Cố định nhãn: Chọn trong các đỉnh có nhãn tự do, lấy ra đỉnh u là đỉnh có d[u] nhỏ nhất, và cố định nhãn đỉnh u.
- Sửa nhãn: Dùng đỉnh u, xét tất cả những đỉnh v và sửa lại các d[v] theo công thức:

$$d[v] := \min(d[v], d[u] + c[u,v])$$

Bước lặp sẽ kết thúc khi mà đỉnh đích f được cố định nhãn (tìm được đường đi ngắn nhất từ s tới f); hoặc tại thao tác cố định nhãn, tất cả các đỉnh tự do đều có nhãn là +∞ (không tồn tại đường đi). Có thể đặt câu hỏi, ở thao tác 1, tại sao đỉnh u như vậy được cố định nhãn, giả sử d[u] còn có thể tối ưu thêm được nữa thì tất phải có một đỉnh t mang nhãn tự do sao cho d[u] > d[t] + c[t, u]. Do trọng số c[t, u] không âm nên d[u] > d[t], trái với cách chọn d[u] là nhỏ nhất. Tất nhiên trong lần lặp đầu tiên thì s là đỉnh được cố định nhãn do d[s] = 0.

Bước 3: Kết hợp với việc lưu vết đường đi trên từng bước sửa nhãn, thông báo đường đi ngắn nhất tìm được hoặc cho biết không tồn tại đường đi (d[f] = +∞).

```
for (∀v ∈ V) do d[v] := +∞;
d[s] := 0;
repeat
  u := arg min(d[v]|∀v ∈ V); {Lấy u là đỉnh có nhãn d[u]
  nhỏ nhất}
  if (u = f) or (d[u] = +∞) then Break; {Hoặc tìm ra
  đường đi ngắn nhất từ s tới f, hoặc kết luận không có
  đường}
  for (∀v ∈ V: (u, v) ∈ E) do {Dùng u tối ưu nhãn những
  đỉnh v kề với u}
    d[v] := min (d[v], d[u] + c[u, v]);
until False;
```

Chú ý: Nếu đồ thị thưa (có nhiều đỉnh, ít cạnh) ta có thể sử dụng danh sách kề kèm trọng số để biểu diễn đồ thị, tuy nhiên tốc độ của thuật toán Dijkstra vẫn khá chậm vì trong trường hợp xấu nhất, nó cần n lần cố định nhãn và mỗi lần tìm đỉnh để cố định nhãn sẽ mất một đoạn chương trình

với độ phức tạp O(n). Để thuật toán làm việc hiệu quả hơn, người ta thường sử dụng cấu trúc dữ liệu Heap để lưu các đỉnh chưa cố định nhãn.

Bài tập

Bài 1: Ông Ngâu bà Ngâu

Hẳn các bạn đã biết ngày "ông Ngâu bà Ngâu" hàng năm, đó là một ngày đầy mưa và nước mắt. Tuy nhiên, một ngày trước đó, nhà Trời cho phép 2 "ông bà" được đoàn tụ. Trong vũ trụ vùng thiên hà nơi ông Ngâu bà Ngâu ngự trị có N hành tinh đánh số từ 1 đến N, ông ở hành tinh Adam (có số hiệu là S) và bà ở hành tinh Eva (có số hiệu là T). Họ cần tìm đến gặp nhau.

N hành tinh được nối với nhau bởi một hệ thống cầu vồng. Hai hành tinh bất kỳ chỉ có thể không có hoặc duy nhất một cầu vồng (hai chiều) nối giữa chúng. Họ luôn đi tới mục tiêu theo con đường ngắn nhất. Họ đi với tốc độ không đổi và nhanh hơn tốc độ ánh sáng. Điểm gặp mặt của họ chỉ có thể là tại một hành tinh thứ 3 nào đó.

Yêu cầu: Hãy tìm một hành tinh sao cho ông Ngâu và bà Ngâu cùng đến đó một lúc và thời gian đến là sớm nhất. Biết rằng, hai người có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau

Dữ liệu: vào từ file văn bản ONGBANGAU.INP:

- Dòng đầu là 4 số N, M, S, T ($N \le 100$, $1 \le S \ne T \le N$), M là số cầu vồng.
- M dòng tiếp, mỗi dòng gồm ba số nguyên I, J, L thể hiện có cầu vồng nối giữa hai hành tinh i và J có độ dài là L ($1 \le I \ne J \le N$, $0 < L \le 200$).

Kết quả: ghi ra file văn bản ONGBANGAU.OUT: do tính chất cầu vồng, mỗi năm một khác, nên nếu như không tồn tại hành tinh nào thoả mãn yêu cầu thì ghi ra một dòng chữ CRY. Nếu có nhiều hành tinh thoả mãn thì ghi ra hành tinh có chỉ số nhỏ nhất.

***Ví dụ*:**

ONGBANGAU.INP	ONGBANGAU.OUT
4 4 1 4 1 2 1 2 4 1 1 3 2 3 4 2	2

Thuật toán:

Ta có nhận xét:

+ Hai hành tinh bất kì chỉ được nối đến nhau bởi nhiều nhất một cầu vồng
+ Ông Ngâu và bà Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất
+ Họ đi với vận tốc không đổi và nhanh hơn vận tốc ánh sáng

Thực chất đây là một bài toán đồ thị, ta có thuật toán như sau:

Từ hành tinh S (nơi ông Ngâu ở) ta xây dựng bảng SP, trong đó SP[i] là đường đi ngắn nhất từ hành tinh S đến hành tinh i (do ông Ngâu luôn đi tới mục tiêu theo con đường ngắn nhất). SP[i] = 0 tức là không có đường đi từ hành tinh S đến hành tinh i.

Tương tự ta sẽ xây dựng bảng TP, trong đó TP[i] là đường đi ngắn nhất từ hành tinh T đến hành tinh i. Và TP[i] = 0 tức là không có đường đi từ hành tinh T đến hành tinh i.

Do yêu cầu của bài toán là tìm hành tinh khác S và T mà 2 ông bà Ngâu cùng đến một lúc và trong thời gian nhanh nhất. Tức là ta sẽ tìm hành tinh h sao cho (h khác S và T) và(SP[h] = ST[h]) đạt giá trị nhỏ nhất khác 0. Nếu không có hành tinh h nào thoả mãn thì ta thông báo CRY

Để xây dựng mảng SP và ST ta chọn giải thuật Dijkstra tìm đường đi ngắn nhất giữa 2 đỉnh đồ thị.

Bài 2: Đôi bạn

Trước kia Tuấn và Mai là hai bạn cùng lớp còn bây giờ hai bạn học khác trường nhau. Cứ mỗi sáng, đúng 6 giờ cả hai đều đi từ nhà tới trường của mình theo con đường mất ít thời gian nhất (có thể có nhiều con đường đi mất thời gian bằng nhau và đều ít nhất). Nhưng hôm nay, hai bạn muốn gặp nhau để bàn việc họp lớp cũ nhân ngày 20-11.

Cho biết sơ đồ giao thông của thành phố gồm N nút giao thông được đánh số từ 1 đến N và M tuyến đường phố (mỗi đường phố nối 2 nút giao thông). Vị trí nhà của Mai và Tuấn cũng như trường của hai bạn đều nằm ở các nút giao thông. Cần xác định xem Mai và Tuấn có cách nào đi thoả mãn yêu cầu nêu ở trên, đồng thời họ lại có thể gặp nhau ở nút giao thông nào đó trên con đường tới trường hay không ? (Ta nói Tuấn và Mai có thể gặp nhau tại một nút giao thông nào đó nếu họ đến nút giao thông này tại cùng một thời

điểm). Nếu có nhiều phương án thì hãy chỉ ra phương án để Mai và Tuấn gặp nhau sớm nhất.

***Dữ liệu*:** vào từ file văn bản FRIEND.INP

- Dòng đầu tiên chứa 2 số nguyên dương N, M ($1 \le N \le 100$);
- Dòng tiếp theo chứa 4 số nguyên dương Ha, Sa, Hb, Sb lần lượt là số hiệu các nút giao thông tương ứng với: Nhà Tuấn, trường của Tuấn, nhà Mai, trường của Mai.
- Dòng thứ i trong số M dòng tiếp theo chứa 3 số nguyên dương A, B, T. Trong đó A và B là hai đầu của tuyến đường phố i. Còn T là thời gian (tính bằng giây ≤ 1000) cần thiết để Tuấn (hoặc Mai) đi từ A đến B cũng như từ B đến A.

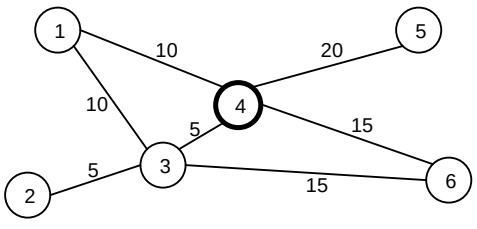
Giả thiết là sơ đồ giao thông trong thành phố đảm bảo để có thể đi từ một nút giao thông bất kỳ đến tất cả các nút còn lại.

***Kết quả* :** ghi ra file văn bản FRIEND.OUT

- Dòng 1: Ghi từ YES hay NO tuỳ theo có phương án giúp cho hai bạn gặp nhau hay không. Trong trường hợp có phương án:
- Dòng 2: Ghi thời gian ít nhất để Tuấn tới trường
- Dòng 3: Ghi các nút giao thông theo thứ tự Tuấn đi qua
- Dòng 4: Ghi thời gian ít nhất để Mai tới trường
- Dòng 5: Ghi các nút giao thông theo thứ tự Mai đi qua
- Dòng 6: Ghi số hiệu nút giao thông mà hai bạn gặp nhau
- Dòng 7: Thời gian sớm nhất tính bằng giây kể từ 6 giờ sáng mà hai bạn có thể gặp nhau.

Ví dụ : Với sơ đồ giao thông sau: (N=6,M=7, Ha=1, Sa=6, Hb=2, Sb=5)

Dòng	FRIEND.INP	FRIEND.OUT
1	**6 7**	**YES**
2	**1 6 2 5**	**25**
3	**1 3 10**	**1 4 6**
4	**1 4 10**	**30**
5	**2 3 5**	**2 3 4 5**
6	**3 4 5**	**4**
7	**3 6 15**	**10**
8	**4 5 20**	



9	4 6 15	

Thuật toán:

Sử dụng thuật toán Dijkstra, xây dựng thủ tục: Dijkstra(start:intger, var d: mảng_nhãn); để xây dựng mảng nhãn d cho đường đi ngắn nhất từ điểm xuất phát start đến mọi đỉnh (có thể tới từ xuất phát). Sau đó gọi thủ tục này 4 lần bằng các lời gọi:

Dijkstra(ha,d1); sẽ được d1 cho biết các đường đi ngắn nhất xuất phát từ nhà Tuấn

Dijkstra(sa,d2); sẽ được d2 cho biết các đường đi ngắn nhất xuất phát từ nhà Mai

Dijkstra(hb,d3); sẽ được d3 cho biết các đường đi ngắn nhất xuất phát từ trường Tuấn

Dijkstra(sb,d4); sẽ được d4cho biết các đường đi ngắn nhất xuất phát từ trường Mai

Điểm hẹn là nút u cần thỏa mãn các điều kiện sau:

d1[u] + d3[u]=d1[sa] {thời gian Tuấn đi từ nhà tới điểm hẹn + Tuấn}

d2[u] + d4[u] = d2[sb] {thời gian Mai đi từ nhà tới điểm hẹn + từ điểm hẹn tới trường Mai}

d1[u] = d2[u] {thời gian đi từ nhà tới điểm hẹn của Tuấn và Mai bằng nhau}

d1[u] nhỏ nhất {thời gian Tuấn đi từ nhà tới điểm hẹn sớm nhất}

Để ghi kết quả vào file FRIENDS.OUT, cần gọi thủ tục Dijkstra một lần nữa: Dijkstra(u,d); sẽ được mảng d(N) cho biết nhãn đường đi ngắn nhất

Bài 3: Đường đi giới hạn

Một mạng giao thông gồm N nút giao thông đánh số từ 1 đến N. Với mỗi cặp nút i, j có đường đi hai chiều và trên đoạn đường đó, người ta quy định một chiều cao nguyên không âm c[i,j] không lớn hơn 6000 là chiều cao tối đa cho mọi xe đi trên đoạn đường đó (c[i,j]=0 có nghĩa là không có đường đi từ i đến j). Cho hai nút s và t. Hãy tìm một hành trình từ s đến t qua các nút khác nhau sao cho chiều cao cho phép tối đa với xe chạy trên hành trình đó là lớn nhất có thể được.

Dữ liệu: vào từ file văn bản HIGHT.INP :

- Dòng thứ nhất ghi 3 số N, s, t (N<=100)

- Tiếp theo là một số dòng, mỗi dòng ghi 3 số i, j, m với ý nghĩa có đường đi hai chiều từ i đến j với chiều cao cho phép h.

Kết quả: ghi ra file văn bản HIGHT.OUT

- Dòng thứ nhất ghi số h là chiều cao cho phép, nếu h>0 trong một số dòng tiếp theo, mỗi dòng ghi một đỉnh trên hành trình lần lượt từ s đến t với chiều cao tối đa cho phép là h.

∞

Thuật toán:

Gọi H[i] là chiều cao lớn nhất có thể của xe để đi từ s đến i

Khởi tạo gán H[s]:=+∞ và H[i] =0 với i ≠ s

Thuật toán sửa nhãn tương tự thuật toán Dijkstra

```
Repeat
     u:=0; max:=0;
     for v:=1 to n do
        if free[v] and (h[v] > max) then
            begin
                 max:=h[v]; u:=v;
            end;
     if u=0 then break;
     free[u]:=false;
     for v:=1 to n do
        if a[u,v] then
            if h[v] < min(h[u],c[u,v]) then
                begin
                     h[v]:=min(h[u],c[u,v]);
                     trace[v]:=u;
                end;
until false;
```

Bài 4: Tổng số đường đi ngắn nhất

Cho đồ thị vô hướng G gồm N đỉnh, M cạnh, mỗi cạnh có 1 trọng số nguyên dương, giữa hai đỉnh bất kì có không quá một cạnh nối. Cho trước hai đỉnh s và t, hãy tính số đường đi ngắn nhất từ s đến t. Hai đường đi khác nhau nếu thứ tự các đỉnh trên 2 đường đi khác nhau.

Thuật toán:

Kết hợp Dijkstra với quy hoạch động

- Theo thuật toán Dijkstra gọi d[i] là độ dài đường đi ngắn nhất từ đỉnh s đến đỉnh i.

Khởi tạo d[i]=+∞ với mọi i ≠ s và d[s]=0

- Quy hoạch động gọi f[i] là số đường đi ngắn nhất từ đỉnh s đến đỉnh i.

Khởi tạo f[i]=0 với mọi i ≠ s và f[s]=1

Trong chương trình Dijkstra:

- Mỗi khi tìm được đường đi mới có độ dài ngắn hơn (d[v]>d[u]+c[u,v]) ta tiến hành thay đổi d[v]:=d[u]+c[u,v]) đồng thời f[v]:=f[u].
- Mỗi khi tìm được 2 đường đi có độ dài bằng nhau (d[v]=d[u]+c[u,v]) ta thay đổi f[v]:=f[v]+f[u].

Kết quả cần tìm là f[t]

Đoạn chương trình Dijkstra kết hợp quy hoạch động

```
For v:=1 to n do d[v]:=maxlongint; d[s]:=0;
For v:=1 to n do f[v]:=0; f[s]:=1;
Fillchar(Free,sizeof(Free),true);
Repeat
    U:=0; mi:=maxlongint;
    For v:=1 to n do
        If (Free[v]) and (d[v]<mi) then
            Begin
                mi:=d[v];
                u:=v;
            end;
    If u=0 then break;
    Free[u]:=false;
    For v:=1 to n do
        If Free[v] then
```

```
            If d[v]>d[u] + c[u,v] then
                Begin
                    d[v]:=d[u] + c[u,v];
                    f[v]:=f[u];
                end
            Else    if  d[v]  =  d[u]  +  c[u,v]  then
        f[v]:=f[v] + f[u];
Until false;
```

Sử dụng Dijkstra để đặt cận cho một số bài toán duyệt

Bài 5: ROADS

N thành phố được đánh số từ 1 đến N nối với nhau bằng các con đường một chiều. Mỗi con đường có hai giá trị: độ dài và chi phí phải trả để đi qua. Bob ở thành phố 1. Bạn hãy giúp Bob tìm đường đi ngắn nhất đến thành phố N, biết rằng Bob chỉ có số tiền có hạn là K mà thôi.

Dữ liệu: vào từ file văn bản ROADS.INP

Dòng đầu tiên ghi t là số test. Với mỗi test:

- Dòng đầu ghi số nguyên K ($0 \le K \le 10000$) là số tiền tối đa mà Bob còn có thể chi cho lệ phí đi đường.
- Dòng 2 ghi số nguyên N ($2 \le N \le 100$) là số thành phố.
- Dòng 3 ghi số nguyên R ($1 \le R \le 10000$) là số đường nối.
- Mỗi dòng trong N dòng sau ghi 4 số nguyên S, D, L, T mô tả một con đường nối giữa S và D với độ dài L ($1 \le L \le 100$) và chi phí T ($0 \le T \le 100$).

Kết quả: ghi ra file văn bản ROADS.OUT

Với mỗi test, in ra độ dài đường đi ngắn nhất từ 1 đến N mà tổng chi phí không quá K. Nếu không tồn tại, in ra -1.

Ví dụ:

ROADS.INP	ROADS.OUT
2 5 6 7 1 2 2 3 2 4 3 3 3 4 2 4 1 3 4 1 4 6 2 1 3 5 2 0 5 4 3 2 0 4 4 1 4 5 2 1 2 1 0 2 3 1 1 3 4 1 0	11 -1

Thuật toán:

Sử dụng thuật toán Dijkstra:

- Lần 1: tìm đường đi ngắn nhất (về khoảng cách) ngược từ đỉnh N về các đỉnh khác để tạo mảng mindist.
- Lần 2: tìm đường đi ngắn nhất (về chi phí tiền) ngược từ đỉnh N về các đỉnh khác để tạo mảng mincost.

Hai mảng mindist và mincost sẽ được dùng làm cận cho quá trình duyệt sau:

Thực hiện duyệt theo các đỉnh từ đỉnh 1. Giả sử đã duyệt tới đỉnh i, và đã đi được quãng đường là d và số tiền đã tiêu là t. Ngay đầu thủ tục Duyet(i,d,t) đặt cận:

Nếu (d+mindist[i]>= đường đi của phương án tốt nhất) thì không cần duyệt tiếp phương án hiện thời nữa.

Nếu (t+mincost[i]>số tiền có của Bob là k) thì không cần duyệt tiếp phương án hiện thời nữa.

Trong chương trình chính gọi thủ tục Duyet(1,0,0).

Chú ý: Để quá trình tìm đỉnh duyệt tiếp theo được nhanh chóng ta cần tổ chức danh sách kề.

Chương trình tham khảo:

```
const   fi              = 'ROADS.INP';
        fo              = 'ROADS.OUT';
        maxn            = 100;
        infinity        = 20000;
        maxtime         = 180;

type    pt              = ^tnode;
        tnode           = record
                                v         : byte;
                                l, t      : byte;
                                next      : pt;
                          end;
        m1              = array[1..maxn] of word;
        m2              = array[1..maxn, 1..maxn] of word;

var
  list                  : array[1..maxn] of pt;
  dd                    : array[1..maxn] of b∞lean;
  cost, dist            : m2;
  mincost, mindist      : m1;
  k                     : word;
  n                     : byte;
  best                  : word;
  f,g                     : text;
  t,test: longint;

procedure init;
var  i, r, u, v, l, t : word;
     tmp              : pt;
begin

  readln(f, k);   {so tien cua Bob}
  readln(f, n);   {so thanh pho}
  readln(f, r);   {so con duong}

  for u:=1 to n do   {khoi tri nhan gia tien , nhan khoang cach}
    for v:=1 to n do
      begin
        cost[u, v]:=infinity;
        dist[u, v]:=infinity;
      end;
```

```
  {to chuc cac danh sach lien ket 1 chieu cua cac con duong. Moi
danh sach
  list[i] cho biet cac thanh pho co duong truc tiep tu i sang}
  for i:=1 to n do {khoi tri cac nut goc cua cac danh sach lien ket
list[i]}
    list[i]:=nil;
  for i:=1 to r do
    begin
      readln(f, u, v, l, t);
      new(tmp);
      tmp^.v:=v;
      tmp^.l:=l;
      tmp^.t:=t;
      tmp^.next:=list[u];
      list[u]:=tmp;

      {so gian lai du lieu}
      if l < dist[u, v] then
        dist[u, v]:=l;
      if t < cost[u, v] then
        cost[u, v]:=t;
    end;
end;

procedure dijkstra(var a : m2; var dist : m1);
{Thuat toan dijkstra tim khoang cach ngan nhat tu thanh pho i toi
thanh pho N}
var   chua             : array[1..maxn] of b∞lean;
      min              : word;
      i, j, last       : byte;
begin
    fillchar(chua, sizeof(chua), true);    {mang danh dau thanh pho
da xet}
    for i:=1 to n do
        dist[i]:=infinity;   {khoi tri mang nhan}
    dist[n]:=0;              {nhan cua dinh N}
    chua[n]:=false;          {danh dau da xet N}

    last:=n;          {last: dinh chua xet co nhan nho nhat}
    for i:=2 to n do  {n-1 lan sua nhan thi xong}
    begin
        {sua nhan cho cac dinh j chua xet dua vao nhan cua last}
        for j:=1 to n do
        if chua[j] and (a[j, last] + dist[last]  < dist[j]) then
```

```
                                        dist[j]:=dist[last] + a[j,
last];
        {tim dinh chua xet o nhan nho nhat}
        min:=infinity+1;
        for j:=1 to n do
        if chua[j] and (dist[j] < min) then
        begin
            min:=dist[j];
            last:=j;
        end;
        {danh dau da xet xong dinh last}
        chua[last]:=false;
    end;
end;

procedure try(last : byte; l, t : word);
{Duyet tiep khi Bob da toi thanh pho last, da di doan duong l, da
tieu t xu}
var  tmp : pt;
begin
   if (l + mindist[last] >= best) {dk can ve duong di}
    or (t + mincost[last] > k) then {dk can ve tien}
                               exit;
    if last = n then  {ve toi dich: Diem dung de quy}
    begin
        best:=l;
        exit;
    end;

    tmp:=list[last]; {tmp: thanh pho last}
    while tmp <> nil do {duyet chon cac de cu cho thanh pho tiep
theo last}
    begin
        if not dd[tmp^.v] then {thanh pho v chua qua}
        begin
          dd[tmp^.v]:=true;  {danh dau da qua v}
          try(tmp^.v, l+tmp^.l, t+tmp^.t); {di tiep tu v}
          dd[tmp^.v]:=false; {quay lui}
        end;
        tmp:=tmp^.next; {de cu thanh pho khac}
    end;
end;

procedure process;
begin
```

```
    {xay dung cac mang cost va dist de lam can phuc vu duyet de quy}
    dijkstra(cost, mincost);
    dijkstra(dist, mindist);
    {khoi tri}
    best:=infinity;
    fillchar(dd, sizeof(dd), false);
   try(1, 0, 0); {duyet tu thanh pho 1 (duong da di =0, tien da
tieu=0}
end;

procedure done;
begin
    if best = infinity then writeln(g, -1)
    else writeln(g, best);
end;

BEGIN
    assign(f, fi); reset(f);
    assign(g, fo); rewrite(g);
    readln(f,test);
    for t:=1 to test do
      begin
         init;
         process;
         done;
      end;
    close(f); close(g);
END.
```

Bài 6: Du lịch khứ hồi

Cho N thành phố đánh số từ 1 đến N. Một người muốn đi du lịch từ thành phố A đến thành phố B và sau đó quay lại A. Người đó muốn rằng trên đường đi từ B về A sẽ không quay lại những thành phố đã qua trên đường đi từ A đến B. Hãy tìm cho người đó một hành trình với chi phí ít nhất.

Dữ liệu: vào từ file văn bản TOURIST.INP

- Dòng thứ nhất ghi 3 số nguyên dương N, A, B (N<=100, 1<=A, B<=N) trong đó N là số thành phố, A là thành phố xuất phát, B là thành phố cần đến.
- Các dòng tiếp theo mỗi dòng ghi 3 số nguyên dương i, j, k với ý nghĩa: giữa thành phố i và thành phố j có đường đi trực tiếp và chi phí đi quãng đường đó là k

Kết quả: ghi ra file văn bản TOURIST.OUT

- Dòng thứ nhất ghi chi phí tổng cộng trên hành trình tìm được

- Dòng thứ hai ghi các thành phố đi qua trên hành trình tìm được theo đúng thứ tự, cách nhau 1 dấu cách. Nếu không có cách đi nào như vậy thì ghi thông báo "No Solution"

Ví dụ:

TOURIST.INP	TOURIST.OUT
10 1 5 1 2 2 2 3 2 3 4 2 4 5 2 5 6 1 6 7 1 7 8 1 8 9 1 9 10 1 10 1 1 1 9 5 9 3 5 3 7 5 7 5 5	14 1 2 3 4 5 6 7 8 9 10 1

Thuật toán:

Dùng thuật toán duyệt có quay lui và đánh giá cận để tìm một đường đi từ thành phố xuất phát A đến thành phố đích B.

Tại mỗi bước, thử chọn một thành phố j vào hành trình đó, ta đánh dấu tất cả các thành phố đã đi qua giữa thành phố A và thành phố j, sau đó dùng thuật toán Dijkstra để tìm độ dài đường đi (là chi phí) ngắn nhất từ thành phố j quay về thành phố A (không được đi qua các thành phố đã đánh dấu). Nếu không tìm thấy đường đi thì gán chi phí là $+\infty$

Nếu chi phí từ A đến thành phố j (tại mỗi bước của quá trình duyệt) cộng với chi phí cho đường đi ngắn nhất từ j về A không tốt hơn giá trị phương án tối ưu tìm được trước đó thì loại phương án chọn thành phố j và thử sang phương án khác.

Tổ chức dữ liệu:

- Gọi X[0..N] là mảng lưu các thành phố đi qua trong quá trình duyệt, X[i] sẽ là thành phố đi qua tại bước thứ i trong tiến trình duyệt. Đặc

biệt X[0]:= A. Để có nghiệm tối ưu, dùng mảng LX[0..N] lưu lại hành trình tốt nhất khi duyệt.

- Gọi D là mảng đánh dấu, ta sẽ đánh dấu bằng các số 0, 1, 2. Khởi tạo ban đầu các đỉnh đều chưa đánh dấu ngoại trừ đỉnh xuất phát A : D[i]=0 với mọi i ≠A ; D[A]:=1;
- Mảng Tien[0..N] có ý nghĩa: Tien[i] cho ta biết chi phí khi đến thành phố thứ i trong duyệt (là thành phố X[i]). Khởi tạo Tien[i]:=0
- Mỗi bước thử chọn thành phố j vào hành trình tại bước thứ i (D[j]=0), ta đặt chi phí tới thành phố j là Tien[i] bằng chi phí cho đến thành phố trước đó là Tien[i-1] cộng với chi phí từ thành phố trước đó (là X[i-1]) tới thành phố j vừa chọn. Đồng thời đánh dấu thành phố j đã đi qua là D[j]:=1;
- Viết một hàm Dijkstra(j) cho ta chi phí ít nhất từ j về A
    - Trước hết xóa đánh dấu cho 2 đỉnh j và A: D[j]=D[A]=0
    - Sau đó áp dụng thuật toán Dijkstra trên tập các đỉnh i có D[i]=0. Mỗi lần cố định nhãn cho đỉnh i ta đặt D[i]=2
    - Trước khi kết thúc, đánh dấu lại 2 đỉnh j và A, đồng thời đặt lại tất cả các D[i]=2 trở về 0 (nghĩa là phục hồi lại mảng đánh dấu D như cũ để không làm hỏng tiến trình duyệt tiếp).
    - Để tăng tốc độ, hàm này không cần lưu vết đường đi mà chỉ cần trả lại độ dài đường đi ngắn nhất (hàm này trả về +∞ nếu không có đường quay về.
- Tại mỗi nút thứ i của duyệt, ta đánh giá cận: Tien[i] + Dijkstra(X[i]) là độ dài đường đi từ A đến X[i] cộng với độ dài đường đi ngắn nhất từ X[i] quay về A. Nếu con số này nhỏ hơn chi phí đường đi trước đó là MinT thì ta tiếp tục tìm kiếm, ngược lại thì không duyệt tiếp nữa. Khi đến được B thì ghi nhận đường đi
- Kết thúc duyệt, nếu không ghi nhận được đường nào (MinT=+∞) thì ghi "No Solution". Ngược lại, tìm được đường đi từ A đến B (và có đường quay về A không đi lặp lại bất cứ một thành phố nào) và chi phí trên cả chu trình là tối thiểu thì in ra đường đi từ A đến B (dựa vào mảng LX) và áp dụng thuật toán Dijkstra (lần này có lưu vết đường đi) để in ra đường quay về từ B đến A.
-

Bài 7: Cuộc đua tiếp sức

Vùng đất Alpha có N thành phố được đánh số từ 1 đến N. Giữa hai thành phố có thể có đường nối trực tiếp hoặc không. Các con đường này được đánh số từ 1 tới M. Ban lãnh đạo thể dục thể thao vùng Alpha tổ chức cuộc chạy đua tiếp sức "thông minh" theo quy luật sau:

- Thành phố xuất phát là thành phố 1, thành phố đích là thành phố N
- Mỗi đội thi đấu có K người dự thi. Lần lượt từng người chạy từ thành phố 1 về thành phố N
- Khi người thứ nhất đến được thành phố N thì người thứ hai mới bắt đầu rời khỏi thành phố 1, khi người thứ hai đến được thành phố N thì người thứ ba mới bắt đầu rời khỏi thành phố 1, ..., người thứ i đến được thành phố N thì người thứ i+1 mời bắt đầu rời khỏi thành phố 1, ..., (i<K). Người thứ K về tới đích tại thời điểm nào thì thời điểm đó được coi là thời điểm về đích của toàn đội.
- Đường chạy của các đội viên không được giống nhau hoàn toàn.
- Có thể chạy lại đoạn đường đã chạy.

Hãy viết chương trình tính thời gian nhỏ nhất để một đội hoàn thành cuộc chạy đua tiếp sức nêu trên nếu các vận động viên có tốc độ chạy như nhau.

Dữ liệu: vào từ file văn bản RELAY.INP

- Dòng đầu tiên ghi 3 số nguyên dương K, N, M (2<=K<=40; 4<=N<=800; 1<=M<=4000)
- M dòng tiếp theo, mỗi dòng chứa 3 số nguyên i, j, w thể hiện một đường đi trực tiếp giữa hai thành phố i và j mất thời gian chạy là w (đơn vị thời gian) (1<=i,j<=N; 1<=w<=9500).

Kết quả: ghi ra file văn bản RELAY.OUT:

- Dòng thứ nhất chứa một số nguyên duy nhất là thời gian chạy nhỏ nhất của một đội
- K dòng tiếp theo, mỗi dòng thể hiện hành trình chạy của một vận động viên trong đội là dãy số hiệu các thành phố liên tiếp trên hành trình đó.

Ví dụ:

RELAY.INP	RELAY.OUT
4 5 8 1 2 1 1 3 2 1 4 2 2 3 2 2 5 3 3 4 3 3 5 4 4 5 6	23 1 3 2 5 1 3 5 1 2 1 2 5 1 2 5

Thuật toán:

Cải tiến từ thuật toán Dijkstra cổ điển

Gọi L[i,j] là độ dài đường đi thứ j trong K đường đi ngắn nhất từ đỉnh 1 đến đỉnh i (i=1, 2, ..., N; j=1, 2, ..,k). Khởi tạo L[i,j] bằng vô cùng với mọi i, j và L[1,1]=0.

- Mỗi lần tìm một cặp (ii,jj) chưa đánh dấu có nhãn L[ii,jj] nhỏ nhất
- Từ (ii,jj) ta tiến hành sửa nhãn cho các cặp (i,j) thỏa mãn: i kề với ii , cặp (i,j) chưa được đánh dấu và L[i,j] >= L[ii,jj] + C[ii,i] (*)

  Khi điều kiện (*) xảy ra thì đường đi ngắn nhất thứ j tới đỉnh i sẽ thành đường đi ngắn nhất thứ j+1 tới i và đường ngắn nhất thứ j tới i sẽ thành đường đi qua ii trước, rồi tới i.

  Do đó với mỗi cặp (i,j) thỏa mãn (*) ta sẽ sửa nhãn cho cặp (i,j) và các cặp có liên quan như sau: L[i,j+s]:=L[i,j+s-1] với mọi s=1 đến k-j và L[i,j]=L[ii,jj] + C[ii,i]

  Tương tự cập nhật lại vết đường đi của các cặp (i,j)
- Đánh dấu cặp (ii,jj) đã cố định nhãn

Quá trình lặp lại cho đến khi không còn cặp (i,j) nào chưa cố định nhãn hoặc cặp (n,k) đã được cố định nhãn.

Sau cùng ta tính tổng độ dài tối ưu của toàn đội K vận động viên

Minpath = L[N,1] + L[N,2] + ... + L[N.K]

và tìm hành trình của từng vận động viên dựa vào mảng theo dõi vết đường đi.

Với số đỉnh và số cạnh của đồ thị tương đối lớn, cần tổ chức danh sách kề.

LUYỆN TẬP

Bài 1: Đến trường

Ngày 27/11 tới là ngày tổ chức thi học kỳ I ở trường ĐH BK. Là sinh viên năm thứ nhất, Hiếu không muốn vì đi muộn mà gặp trục trặc ở phòng thi nên đã chuẩn bị khá kỹ càng. Chỉ còn lại một công việc khá gay go là Hiếu không biết đi đường nào tới trường là nhanh nhất.

Thường ngày Hiếu không quan tâm tới vấn đề này lắm cho nên bây giờ Hiếu không biết phải làm sao cả . Bản đồ thành phố là gồm có N nút giao thông và M con đường nối các nút giao thông này. Có 2 loại con đường là đường 1 chiều và đường 2 chiều. Độ dài của mỗi con đường là một số nguyên dương.

Nhà Hiếu ở nút giao thông 1 còn trường ĐH BK ở nút giao thông N. Vì một lộ trình đường đi từ nhà Hiếu tới trường có thể gặp nhiều yếu tố khác như là gặp nhiều đèn đỏ , đi qua công trường xây dựng, ... phải giảm tốc độ cho nên Hiếu muốn biết là có tất cả bao nhiêu lộ trình ngắn nhất đi từ nhà tới trường. Bạn hãy lập trình giúp Hiếu giải quyết bài toán khó này.

Dữ liệu: vào từ file văn bản ROADS.INP

- Dòng thứ nhất ghi hai số nguyên N và M.
- M dòng tiếp theo, mỗi dòng ghi 4 số nguyên dương K, U, V, L. Trong đó:

  K = 1 có nghĩa là có đường đi một chiều từ U đến V với độ dài L.

  K = 2 có nghìa là có đường đi hai chiều giữa U và V với độ dài L.

Kết quả: ghi ra file văn bản ROADS.OUT hai số là độ dài đường đi ngắn nhấT và số lượng đường đi ngắn nhất. Biết rằng số lượng đường đi ngắn nhất không vượt quá phạm vì int64 trong pascal hay long long trong C++.

Ví dụ:

ROADS.INP	ROADS.OUT
3 2 1 1 2 3 2 2 3 1	4 1

Giới hạn:

$1 \le N \le 5000$

$1 \le M \le 20000$

Độ dài các con đường ≤ 32000

Bài 2: HIWAY

Một mạng giao thông gồm N nút giao thông, và có M đường hai chiều nối một số cặp nút, thông tin về một đường gồm ba số nguyên dương u, v là tên hai nút đầu mút của đường, và w là độ dài đoạn đường đó. Biết rằng hai nút giao thông bất kì có không quá 1 đường hai chiều nhận chúng làm hai đầu mút. Cho hai nút giao thông s và f, hãy tìm hai đường đi nối giữa s với f sao cho hai trên hai đường không có cạnh nào được đi qua hai lần và tổng độ dài 2 đường đi là nhỏ nhất.

Dữ liệu: vào từ file văn bản HIWAY.INP

- Dòng đầu ghi N, M ($N \le 100$)
- Dòng thứ 2 ghi hai số s, f.
- M dòng tiếp theo, mỗi dòng mô tả một đường gồm ba số nguyên dương u, v, w.

Kết quả: ghi ra file văn bản HIWAY.OUT

- Dòng đầu ghi T là tổng độ dài nhỏ nhất tìm được hoặc -1 nếu không tìm được.
- Nếu tìm được, hai dòng sau, mỗi dòng mô tả một đường đi gồm: số đầu là số nút trên đường đi này, tiếp theo là dãy các nút trên đường đi bắt đầu từ s, kết thúc tại f.

(Phạm vi tính toán trong vòng Longint)

Ví dụ:

HIWAY.INP	HIWAY.OUT
5 8 1 5 1 2 1 1 4 8 2 3 5 2 4 1 3 5 1 4 3 8 4 5 1 1 3 1	5 3 1 3 5 4 1 2 4 5

Bài 3: SHORTEST

Một hệ thống giao thông gồm N thành phố và M đoạn đường một chiều. Các thành phố có số hiệu từ 1 đến N. Mỗi đoạn đường ta biết thành phố xuất phát và thành phố đích và độ dài. Ta nói rằng đoạn đường F là *tiếp nối* của đoạn đường E nếu thành phố đích của đoạn đường E là thành phố xuất phát của đoạn đường F. Một *hành trình* từ thành phố A đến thành phố B là một dãy liên tiếp các đoạn đường sao cho thành phố xuất phát của đoạn đường đầu tiên là A, mỗi đoạn đường khác là tiếp nối của một đoạn đường trước đó và thành phố đích của đoạn đường cuối cùng là thành phố B. Độ dài của hành trình là tổng độ dài của các đoạn đường trong hành trình. Một hành trình từ A đến B là *hành trình ngắn nhất* nếu không có hành trình nào từ A đến B có độ dài ngắn hơn.

Yêu cầu: Với mỗi đoạn đường, cho biết có bao nhiêu hành trình ngắn nhất chứa đoạn đường đó.

Dữ liệu: Cho trong tệp SHORTEST.INP gồm có:

- Dòng đầu ghi hai số nguyên N và M ($1 \le N \le 1500$, $1 \le M \le 5000$), là số thành phố và số đoạn đường.
- Dòng thứ i trong M dòng tiếp chứa ba số nguyên U_i , V_i , L_i tương ứng là thành phố xuất phát, thành phố đích và độ dài của đoạn đường thứ i (các đoạn đường đều là một chiều; các số U_i, V_i là khác nhau và giá trị L_i tối đa là 10000).

Kết quả: Ghi ra tệp SHORTEST.OUT gồm có M dòng, trong đó dòng thứ i dòng ghi một số nguyên C_i là số hành trình ngắn nhất khác nhau chứa đoạn đường thứ i (vì số C_i có thể là rất lớn nên bạn hãy viết nó dưới dạng số dư của 1 000 000 007).

Ví dụ:

SHORTEST.INP	SHORTEST.OUT
4 3 1 2 5 2 3 5 3 4 5	3 4 3
4 4 1 2 5 2 3 5 3 4 5 1 4 8	2 3 2 1
5 8 1 2 20 1 3 2 2 3 2 4 2 3 4 2 3 3 4 5 4 3 5 5 4 20	0 4 6 6 6 7 2 6

LỜI KẾT

Bài tập ứng dụng thuật toán Dijkstra vô cùng phong phú và đa dạng, từ cơ bản đến nâng cao. Phần lớn các ví dụ được nêu ra trong tham luận được tổng hợp từ nhiều nguồn tài liệu tham khảo khác nhau.. Tôi rất mong nhận được ý kiến đóng góp của các quý thầy cô để tham luận hoàn thiện hơn.

Rất mong được đồng nghiệp các nơi đóng góp ý kiến để chúng tôi hoàn thiện kiến thức về vấn đề này.

Xin trân trọng cảm ơn!

Chuyên đề xếp loại B

TÊN CHUYÊN ĐỀ:
PHÉP DUYỆT MỘT ĐỒ THỊ

GIÁO VIÊN: ĐẶNG TUẤN THÀNH
TRƯỜNG THPT CHUYÊN NGUYỄN TẤT THÀNH – YÊN BÁI

Các bài toán về đồ thị ngày càng được quan tâm nghiên cứu, phát triển, ứng dụng trong khoa học và cuộc sống. Một trong những cách tiếp cận các bài toán này là Phép duyệt đồ thị. Trong phạm vi tham luận của mình tôi xin đề cập đến một số phép duyệt đồ thị cơ bản, hiệu quả.

Như bạn đã biết: Khi biết gốc của một cây ta có thể thực hiện phép duyệt cây đó để thăm các nút của cây theo thứ tự nào đấy. Với đồ thị vấn đề đặt ra cũng tương tự. Xét một đồ thị không định hướng G(V,E) và một đỉnh v trong V(G), ta cần thăm tất cả các đỉnh thuộc G mà có thể với tới được đỉnh v (nghĩa là thăm mọi nút liên thông với v).

Ta có hai cách giải quyết trên đây: Phép tìm kiếm theo chiều sâu (Depth First Search-DFS) và phép tìm nhiếu theo chiều rộng (Breadth First Search-BFS).

1. Tìm kiếm theo chiều sâu.

Đỉnh xuất phát v được thăm, tiếp theo đó một đinh w chưa được thăm, mà là lân cận của v, sẽ được chọn và một phép tìm kiếm theo chiều sâu xuất phát từ w lại được thực hiện.

Khi một đỉnh u đã được với tới mà mọi đỉnh lân cận của nó đều đã được thăm rồi, thì ta sẽ quay ngược lên đỉnh cuối cùng vừa được thăm (mà còn có đỉnh w lân cận với nó chưa được thăm). Và một phép tìm kiếm theo chiều sâu xuất phát từ w lại được thực hiện. Phép tìm kiếm sẽ kết thúc khi không còn một nút nào chưa được thăm mà vẫn có thể với tới được từ nút đã được thăm.

Giải thuật của phép duyệt này:

```
Procedure DFS(v)
  1) Visited(v) :=1;  //Visited dùng để đánh dấu các đỉnh đã được thăm
  2) For mỗi đỉnh w lân cận của v Do
        If Visited(w)=0 then Call DFS(w);
  3) Return
```

Ta thấy: Trong trường phợp G được biểu diễn bởi một danh sách lân cận thì đỉnh w lân cận của v sẽ được xác định bằng cách dựa vào danh sách móc nối ứng với v. Vì giải thuật DFS chỉ xem xét mỗi nút trong một danh sách lân cận nhiều nhất một lần mà thôi mà lại có 2e nút danh sách (ứng với e cung), nên thời gian để hoàn thành phép tìm kiếm chỉ là **O(e)**. Còn nếu G được biểu diễn bởi ma trận lân cận thì thời gian để xác định mọi đỉnh lân cận của v là O(n). Vì tối đa có n đỉnh được thăm, nên thời gian tìm kiếm tổng quát sẽ là $\mathbf{O(n^2)}$.

Giải thuật DFS(V_1) sẽ đảm bảo thăm mọi đỉnh liên thông với V_1. Tất cả các đỉnh được thăm cùng với các cung liên quan tới các đỉnh đó gọi là một bộ phận liên thông (vùng liên thông) của G. Với phép duyệt DFS ta có thể xác định được G có liên thông hay không, hoặc tìm được các bộ phận liên thông của G nếu G không liên thông.

Áp dụng giải thuật tìm kiếm theo chiều sâu DFS để giải các bài toán sau, sẽ giúp ta hiểu hơn về DFS.

Bài toán: **Bãi cỏ ngon nhất -** VBGRASS

Bessie dự định cả ngày sẽ nhai cỏ xuân và ngắm nhìn cảnh xuân trên cánh đồng của nông dân John, cánh đồng này được chia thành các ô vuông nhỏ với R (1 <= R <= 100) hàng và C (1 <= C <= 100) cột. Bessie ước gì có thể đếm được số khóm cỏ trên cánh đồng.

Mỗi khóm cỏ trên bản đồ được đánh dấu bằng một ký tự '#' hoặc là 2 ký tự '#' nằm kề nhau (trên đường chéo thì không phải). Cho bản đồ của cánh đồng, ***hãy nói cho Bessie biết có bao nhiêu khóm cỏ trên cánh đồng.***

Ví dụ như cánh đồng dưới dây với R=5 và C=6:

.#....
..#...
..#..#
...##.
.#....

Cánh đồng này có 5 khóm cỏ: Một khóm ở hàng đầu tiên, một khóm tạo bởi hàng thứ 2 và thứ 3 ở cột thứ 2, một khóm là 1 ký tự nằm riêng rẽ ở hàng 3, một khóm tạo bởi cột thứ 4 và thứ 5 ở hàng 4, và một khóm cuối cùng ở hàng 5.

Dữ liệu

- Dòng 1: 2 số nguyên cách nhau bởi dấu cách: R và C

- Dòng 2..R+1: Dòng i+1 mô tả hàng i của cánh đồng với C ký tự, các ký tự là '#' hoặc '.' .

Kết quả

- Dòng 1: Một số nguyên cho biết số lượng khóm cỏ trên cánh đồng.

Ví dụ

Dữ liệu

5 6
.#....
..#...
..#..#
...##.
.#....

Kết quả

5

Nhận xét: Số lượng các khóm cỏ có thể xem là số vùng liên thông trên đồ thị. Trong đó, khi a[i,j] là cỏ và 4 đỉnh lân cận của nó, nếu cũng là cỏ thì tồn tại đường đi từ a[i,j] đến đỉnh đó.

```
Uses math;
Const
fi          ='VBGRASS.INP';
fo          ='VBGRASS.OUT';

MAXN    = 200;

Var
f           : array [0..MAXN+1,0..MAXN+1] of boolean;
m,n         : longint;
Res         : longint;

Procedure Init();
begin
    Fillchar(f,sizeof(f),false);
    Res := 0;
end;

Procedure ReadData();
var i,j : longint;
c           : char;
begin
    Readln(m,n);
    for i:=1 to m do
    begin
        for j:=1 to n do
                begin
                        read(c);
                        f[i,j] := c = '#';
                end;
        readln;
    end;
```

```
Procedure Dfs(x,y: longint);
const
tx    : array [1..4] of longint = (1,-1,0,0);
ty    : array [1..4] of longint = (0,0,1,-1);
var
i,u,v: longint;
begin
    f[x,y]:=false;
    for i:=1 to 4 do
        begin
        u:=tx[i] + x;
        v:=ty[i] + y;
        if f[u,v] then
                dfs(u,v);
        end;
end;
Procedure Solve();
var i,j : longint;
begin
    for i:=1 to m do
        for j:=1 to n do
                if f[i,j] then
                begin
                        dfs(i,j);
                        inc(Res);
                end;
end;

BEGIN
assign(input,fi); reset(input);
assign(output,fo); rewrite(output);
    Init();
```

end;	ReadData(); Solve(); Writeln(Res); close(input); close(output); END.

Bài Toán: **V8ORG**

Ở một đất nước nọ, lực lượng an ninh vừa phát hiện một tổ chức đối lập. Tổ chức đối lập này được tổ chức chặt chẽ, bao gồm mạng lưới thành viên và chỉ huy ở các cấp bậc khác nhau. Các thành viên của tổ chức được đánh số từ 1 đến N. Tổ chức có một chỉ huy tối cao, luôn được đánh số 1. Mỗi thành viên chỉ biết viên chỉ huy trực tiếp của mình (có duy nhất một viên chỉ huy trực tiếp) chứ không biết các chỉ huy cấp cao hơn.

Khi tiến hành việc bắt giữ các thành viên, tổ chức sẽ bị phân rã thành các nhóm nhỏ không liên kết với nhau, ví dụ sau khi bắt giữ thành viên số 2 (hình 1), tổ chức bị phân rã thành 4 nhóm. Lực lượng an ninh khẳng định, một nhóm chứa ít hơn K thành viên sẽ không còn là mối đe dọa cho đất nước. Để không làm giảm hình ảnh của đất nước trước dư luận quốc tế, các nhà lãnh đạo an ninh muốn bắt giữ một số lượng ít nhất phần tử đối lập, sao cho các nhóm bị phân rã đều không còn gây nguy hại cho đất nước.

Cho biết cấu trúc của tổ chức đối lập, việc chương trình giúp các nhà lãnh đạo an ninh xác định số lượng phần tử đối lập ít nhất cần bắt giữ.

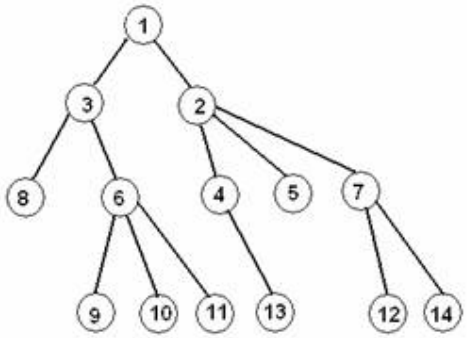
Dữ liệu

- Dòng đầu tiên chứa số nguyên K ($1 \le K \le 10000$).
- Dòng thứ hai chứa số nguyên N ($1 \le N \le 10000$).
- Dòng thứ ba chứa N-1 số nguyên cách nhau bởi khoảng trắng, chỉ số của chỉ huy trực tiếp của mỗi phần tử của tổ chức (trừ chỉ huy tối cao): Số đầu tiên cho biết chỉ huy của phần tử thứ hai, số thứ hai cho biết chỉ huy của phần tử thứ ba,...

Kết qủa

In ra một số nguyên duy nhất là số phần tử đối lập ít nhất cần bắt giữ.

Ví dụ

Dữ liệu	Kết quả	Mô tả
3 14 1 1 2 2 3 2 3 6 6 6 7 4 7  Hình 1	4	Có thể bắt giữ 4 phần tử 6, 2, 7 và 8.

Ý tưởng giải thuật: Gọi s[i] là số lượng phần tử dưới quyền chỉ huy của phần tử i (bao gồm cả chính i) , dễ thấy trong quá trình duyệt Dfs , nếu có một phần tử có s[i] >= k thì ta sẽ "bắt giữ" phần tử này, tức là cho s[i] = 0, việc tính các s[u] (với mọi u nhận i là chỉ huy sẽ được tính trước khi tính s[i]);

```
Const
fi          ='V8ORG.INP';
fo          ='V8ORG.OUT';

MAXN    = 20000;

type
link  =^node;
node = record
       v      : longint;
       next: link;
       end;

var
a     : array [0..MAXN] of link;
s     : array [0..MAXN] of longint;
n,k   : longint;
Res   : longint;

Procedure Push(u,v: longint);
var p: link;
begin
       new(p);
       p^.v:=v;
       p^.next:=a[u];
       a[u]:=p;
end;
```

```
Procedure Dfs(x: longint);
var p: link;
begin
       p:=a[x];
       s[x]:=1;
       while p<>nil do
       begin
              dfs(p^.v);
              inc(s[x],s[p^.v]);
              p:=p^.next;
       end;
       if s[x] >= k then
              begin
              inc(Res);
              s[x]:=0;
              end;
end;

BEGIN
assign(input,fi); reset(input);
assign(output,fo); rewrite(output);
       ReadData();
       Dfs(1);
       Write(Res);
close(input); close(output);
END.
```

```
Procedure ReadData();
var i: longint;
x          : longint;
begin
      Readln(k);
      Readln(n);
      for i:=1 to n-1 do
            begin
                  read(x);
                  push(x,i+1);
            end;
end;
```

Bài toán: Dạo chơi đồng cỏ

Có N con bò (1 <= N <= 1,000), để thuận tiện ta đánh số từ 1->N, đang ăn cỏ trên N đồng cỏ, để thuận tiện ta cũng đánh số các đồng cỏ từ 1->N. Biết rằng con bò i đang ăn cỏ trên đồng cỏ i.

Một vài cặp đồng cỏ được nối với nhau bởi 1 trong N-1 con đường 2 chiều mà các con bò có thể đi qua. Con đường i nối 2 đồng cỏ A_i và B_i (1 <= A_i <= N; 1 <= B_i <= N) và có độ dài là L_i (1 <= L_i <= 10,000).

Các con đường được thiết kế sao cho với 2 đồng cỏ bất kỳ đều có duy nhất 1 đường đi giữa chúng. Như vậy các con đường này đã hình thành 1 cấu trúc cây.

Các chú bò rất có tinh thần tập thể và muốn được thăm thường xuyên. Vì vậy lũ bò muốn bạn giúp chúng tính toán độ dài đường đi giữa Q (1 <= Q <= 1,000) cặp đồng cỏ (mỗi cặp được mô tả là 2 số nguyên p1,p2 (1 <= p1 <= N; 1 <= p2 <= N).

DỮ LIỆU

- Dòng 1: 2 số nguyên cách nhau bởi dấu cách: N và Q
- Dòng 2..N: Dòng i+1 chứa 3 số nguyên cách nhau bởi dấu cách: A_i, B_i, và L_i
- Dòng N+1..N+Q: Mỗi dòng chứa 2 số nguyên khác nhau cách nhau bởi dấu cách mô tả 1 yêu cầu tính toán độ dài 2 đồng cỏ mà lũ bò muốn đi thăm qua lại p1 và p2.

KẾT QUẢ

- Dòng 1..Q: Dòng i chứa độ dài đường đi giữa 2 đồng cỏ ở yêu cầu thứ i.

VÍ DỤ

Dữ liệu

4 2
2 1 2
4 3 2

1 4 3
1 2
3 2

Kết quả

2
7

GIẢI THÍCH

Yêu cầu 1: Con đường giữa đồng cỏ 1 và 2 có độ dài là 2. Yêu cầu 2: Đi qua con đường nối đồng cỏ 3 và 4, rồi tiếp tục đi qua con đường nối 4 và 1, và cuối cùng là con đướng nối 1 và 2, độ dài tổng cộng là 7.

Ý tưởng giải thuật : Với mỗi truy vấn (p1,p2) ta thực hiện Dfs bắt đầu từ p1, trong quá trình dfs ta lưu lại f[i] là độ dài trên đường đi từ i đến p1, kết quả là f[p2];

```
Const
fi              ='PWALK.INP';
fo              ='PWALK.OUT';
MAXN     = 2000;

type
link   =^node;
node =record
        v,w   : longint;
        next:link;
        end;



var
a               : array [0..MAXN] of link;
n,q             : longint;
p1,p2           : longint;

l               : array [0..MAXN] of longint;
free   : array [0..MAXN] of boolean;

Procedure push(u,v,w: longint);
var p: link;
begin
        new(p);
        p^.v:=v;
        p^.w:=w;
        p^.next:=a[u]; a[u]:=p;
end;
```

```
Procedure Dfs(x: longint);
var p : link;
v               : longint;
begin
        free[x] := false;
        p:=a[x];
        while p<>nil do
                begin
                        v:=p^.v;
                        if free[v] then
                        begin
                                l[v] := l[x] + p^.w;
                                dfs(v);
                        end;
                        p:=p^.next;
                end;
end;

Procedure Solve();
begin
        Fillchar(l,sizeof(l),0);
        Fillchar(free,sizeof(free),true);
        Dfs(p1);
end;

BEGIN
assign(input,fi); reset(input);
assign(output,fo); rewrite(output);
        ReadData();
```

```
Procedure ReadData();
var i   : longint;
u,v,w: longint;
begin
      Readln(n,q);
      for i:=1 to n-1 do
            begin
            readln(u,v,w);
            push(u,v,w);
            push(v,u,w);
            end;
end;
```

```
      While q>0 do
            begin
            Readln(p1,p2);
            Solve();
            Writeln(l[p2]);
            dec(q);
            end;
close(input); close(output);
END.
```

Bài toán: Bảo vệ nông trang

Nông trang có rất nhiều ngọn đồi núi, để bảo vệ nông trang nông dân John muốn đặt người canh gác trên các ngọn đồi này.

Anh ta băn khoăn không biết sẽ cần bao nhiêu người canh gác nếu như anh ta muốn đặt 1 người canh gác trên đỉnh của mỗi đồi. Anh ta có bản đồ của nông trang là một ma trận gồm N (1 < N <= 700) hàng và M (1 < M <= 700) cột. Mỗi phần tử của ma trận là độ cao H_ij so với mặt nước biển (0 <= H_ij <= 10,000) của ô (i, j). Hãy giúp anh ta xác định số lượng đỉnh đồi trên bản đồ.

Đỉnh đồi là 1 hoặc nhiều ô nằm kề nhau của ma trận có cùng độ cao được bao quanh bởi cạnh của bản đồ hoặc bởi các ô có độ cao nhỏ hơn. Hai ô gọi là kề nhau nếu độ chênh lệch giữa tọa độ X không quá 1 và chênh lệch tọa độ Y không quá 1.

Dữ liệu

* Dòng 1: Hai số nguyên cách nhau bởi dấu cách: N và M
* Dòng 2..N+1: Dòng i+1 mô tả hàng i của ma trận với M số nguyên cách nhau bởi dấu cách: H_ij

Kết quả

* Dòng 1: Một số nguyên duy nhất là số lượng đỉnh đồi.

Ví dụ

Dữ liệu:

8 7
4 3 2 2 1 0 1
3 3 3 2 1 0 1
2 2 2 2 1 0 0
2 1 1 1 1 0 0
1 1 0 0 0 1 0
0 0 0 1 1 1 0

0 1 2 2 1 1 0
0 1 1 1 2 1 0
Kết quả: 3
Ý tưởng giải thuật : Ta sẽ làm 2 bước:

Bước 1 : Với mỗi đỉnh [i,j] chưa thăm, ta dfs đánh dấu các đỉnh có chiều cao < a[i,j], ta sẽ đảm bảo rằng từ đỉnh có chiều cao a[u,v] nào đó, thủ tục dfs1 sẽ đánh dấu những đỉnh có chiều cao <= a[u,v] lận cận;

Như vậy chỉ có các đỉnh có chiều cao "đỉnh" còn lại;

Bước 2: Dfs để tìm các nhóm đỉnh, công việc này khá dễ dàng, cách làm tương tự với bài VBGRASS.

```
Const
fi                      ='NKGUARD.INP';
fo                      ='';

MAXN    = 1000;

tx                      : array [1..8] of longint = (1,1,1,-1,-1,-1,0,0);
ty                      : array [1..8] of longint = (-1,0,1,-1,0,1,1,-1);

Var
a                       : array [0..MAXN+1,0..MAXN+1] of longint;
m,n                     : longint;

Res                     : longint = 0;

free    : array [0..MAXN+1,0..MAXN+1] of boolean;

Procedure ReadData();
var i,j: longint;
begin
        Readln(m,n);
        for i:=1 to m do
                for j:=1 to n do
                read(a[i,j]);
end;

Procedure Init();
var i: longint;
begin
        Fillchar(free,sizeof(free),true);
        for i:=0 to m+1 do
        begin
                free[i,n+1]:=false;
                free[i,0]:=false;
        end;
        for i:=0 to n+1 do
```

```
Procedure Dfs1(x,y,s: longint);
var i: longint;
u,v     : longint;
begin
        for i:=1 to 8 do
                begin
                u:=x+tx[i];
                v:=y+ty[i];
        if (free[u,v]) and (a[u,v]<=a[x,y]) and (a[u,v]<s) then
                                begin
                                free[u,v]:=false;
                                Dfs1(u,v,s);
                                end;
                end;
end;

Procedure Dfs2(x,y: longint);
var i: longint;
u,v     : longint;
begin
        for i:=1 to 8 do
                begin
                                u:=x+tx[i];
                                v:=y+ty[i];
                                if free[u,v] then
                                begin
                                free[u,v]:=false;
                                Dfs2(u,v);
                                end;
                end;
end;

Procedure Solve();
var i,j     : longint;
begin
        for i:=1 to m do
                for j:=1 to n do
```

```
        begin
                free[m+1,i]:=false;
                free[0,i]:=false
        end;
end;
```

```
                if free[i,j] then Dfs1(i,j,a[i,j]);

        for i:=1 to m do
                for j:=1 to n do
                if free[i,j] then
                begin
                        Dfs2(i,j);
                        inc(Res);
                end;
end;

BEGIN
assign(input,fi); reset(input);
assign(output,fo); rewrite(output);
        ReadData();
        Init();
        Solve();
        Writeln(Res);
close(input); close(output);
END.
```

Bài toán: Leo núi

Cho một bản đồ kích thước NxN (2 <= N <= 100), mỗi ô mang giá trị là độ cao của ô đó (0 <= độ cao <= 110). Bác John và bò Bessie đang ở ô trên trái (dòng 1, cột 1) và muốn đi đến cabin (dòng N, cột N). Họ có thể đi sang phải, trái, lên trên và xuống dưới nhưng không thể đi theo đường chéo. Hãy giúp bác John và bò Bessie tìm đường đi sao cho chênh lệch giữa điểm cao nhất và thấp nhất trên đường đi là nhỏ nhất.

Dữ liệu

- Dòng 1: N
- Dòng 2..N+1: Mỗi dòng chứa N số nguyên, mỗi số cho biết cao độ của một ô.

Kết quả

Một số nguyên là chênh lệch cao độ nhỏ nhất.

Ví dụ

Dữ liệu

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

Kết quả

2

Ý tưởng : Do giới hạn chiều cao của đỉnh đồi là 200 nên ta sẽ thực hiện tìm kiếm nhị phân và Dfs;

Bắt đầu với 2 biến hmin là chiều cao nhỏ nhất sẽ xét, hmax là chiều cao lớn nhất sẽ xét, ta duyệt hmin từ 1 đến 200 và dùng hàm chặt nhị phân tìm hmax nhỏ nhất sao cho nếu đoạn đường từ (1,1) đến (n,n) chỉ có các đỉnh có độ cao nằm trong đoạn [hmin,hmax]

Với mỗi cặp hmin, hmax tìm được, ta so sánh hiệu với kết quả và cập nhật.

```
{Thuật toán : DFS + chặt nhị phân}
uses Math;

Const
fi          ='MTWALK.INP';
fo          ='MTWALK.OUT';

MAXN    =200;
INF         =99999;
var
a           : array [1..MAXN,1..MAXN] of longint;
n           : longint;

res         : longint = INF;

free   : array [0..MAXN,0..MAXN] of boolean;
hmin,hmax: longint;

Procedure ReadData();
var i,j : longint;
begin
      Readln(n);
      for i:=1 to n do
            for j:=1 to n do
                  read(a[i,j]);
end;

Procedure Dfs(x,y: longint);
const
tx : array [1..4] of longint = (1,-1,0,0);
ty : array [1..4] of longint = (0,0,1,-1);
var i,u,v: longint;
begin
      for i:=1 to 4 do
            begin
            u:=x+tx[i];
            v:=y+ty[i];
            if free[u,v] and (a[u,v] >= hmin) and (a[u,v] <=hmax)
      then
                  begin
                  free[u,v]:=false;
                  Dfs(u,v);
                  end;
            end;
end;
```

```
Function ok():boolean;
var i: longint;
begin
      Fillchar(free,sizeof(free),true);
      for i:=1 to n do
            begin
            free[i,0]:=false;
            free[0,i]:=false;
            free[i,n+1]:=false;
            free[n+1,i]:=false;
            end;
      if (a[1,1] >= hmin) and (a[1,1] <=hmax) then
      Dfs(1,1);
      exit(not(free[n,n]));
end;

Function f():longint;
var u,v,mid: longint;
begin
      u:=hmin; v:=200;
      while u<v-1 do
            begin
                  mid:= (u+v) div 2;
                  hmax:=mid;
                  if ok() then v:=mid else u:=mid;
            end;
      hmax:=u;
if ok() then exit(u-hmin);
      hmax:=v;
if ok() then exit(v-hmin);
exit(INF);
end;

BEGIN
assign(input,fi); reset(input);
assign(output,fo); rewrite(output);
      ReadData();
      For hmin:=0 to 200 do Res := min(Res,f());
      Writeln(Res);
close(input); close(output);
END.
```

Bài toán: Nước lạnh

Mùa hè oi ả ở Wisconsin đã khiến cho lũ bò phải đi tìm nước để làm dịu đi cơn khát. Các đường ống dẫn nước của nông dân John đã dẫn nước lạnh vào 1 tập N (3 <= N <= 99999; N lẻ) nhánh (đánh số từ 1..N) từ một cái bơm đặt ở chuồng bò.

Khi nước lạnh chảy qua các ống, sức nóng mùa hè sẽ làm nước ấm lên. Bessie muốn tìm chỗ có nước lạnh nhất để cô bò có thể tận hưởng mùa hè một cách thoải mái nhất.

Bessie đã vẽ sơ đồ toàn bộ các nhánh ống nước và nhận ra rằng nó là một đồ thị dạng cây với gốc là chuồng bò và ở các điểm nút ống thì có chính xác 2 nhánh con đi ra từ nút đó. Một điều ngạc nhiên là các nhánh ống này đều có độ dài là 1.

Cho bản đồ các ống nước, hãy cho biết khoảng cách từ chuồng bò tới tất cả các nút ống và ở các phần cuối đường ống.

"Phần cuối" của một đường ống, có thể là đi vào một nút ống hoặc là bị bịt, được gọi theo số thứ tự của đường ống. Bản đồ có C (1 <= C <= N) nút ống, được mô tả bằng 3 số nguyên: là "phần cuối" của ống E_i (1 <= E_i <= N) và 2 ống nhánh đi ra từ đó là B1_i và B2_i (2 <= B1_i <= N; 2 <= B2_i <= N). Đường ống số 1 nối với chuồng bò; khoảng cách từ phần cuối của đường ống này tới chuồng bò là 1.

Dữ liệu

- Dòng 1: 2 số nguyên cách nhau bởi dấu cách: N và C
- Dòng 2..C+1: Dòng i+1 mô tả nút ống i với ba Số nguyên cách nhau bởi dấu cách: E_i, B1_i, và B2_i

Kết quả

- Dòng 1..N: Dòng i chứa 1 số nguyên là khoảng cách từ chuồng tới "phần cuối" của ống thứ i.

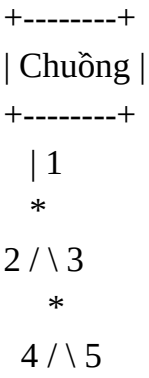
Ví dụ

Dữ liệu

5 2
3 5 4
1 2 3

Giải thích:

Dữ liệu ở trên mô tả bản đồ ống nước sau:



Kết quả

1
2
2
3
3

Giải thích:

Ống 1 luôn cách chuồng 1 đoạn là 1. Ống 2 và 3 nối với ống 1 nên khoảng cách sẽ là 2. Ống 4 và 5 nối với ống 3 nên khoảng cách sẽ là 3.

Ý tưởng thuật toán: Gọi h[i] là độ dài từ ống i đến chuồng, r[i] là ống phải của i và l[i] là ống trái, ta có h[r[i]] = h[l[i]] = h[i] + 1;

```
Const   fi='VCOLDWAT.INP';
        fo='';
        mxF=100000;
        mxT=1000;

Type    Nut=record
              t,p:longint;
        end;

Var     n:longint;
        h:array [1..mxF] of longint;
        a:array [1..mxF] of Nut;

Procedure Init;
Var     c,i,e:longint;
Begin
        assign(input,fi);
        reset(input);
```

```
procedure DFS(u:longint);
begin
        if u=1 then h[u]:=1;
        if a[u].t<>0 then
                begin
                        h[a[u].t]:=h[u]+1;
                        DFS(a[u].t);
                end;
        if a[u].p<>0 then
                begin
                        h[a[u].p]:=h[u]+1;
                        DFS(a[u].p);
                end;
end;

procedure GetOut;
var     i:longint;
```

```
    readln(n,c);
    for i:=1 to c do readln(e,a[e].t,a[e].p);
    close(input);
End;
```

```
begin
    assign(output,fo);
    rewrite(output);
    for i:=1 to n do DFS(i);
    for i:=1 to n do writeln(h[i]);
    close(output);
end;

BEGIN
    Init;
    GetOut;
END.
```

2. Tìm kiếm theo chiều rộng

Trong phép duyệt đồ thị BFS, đỉnh xuất phát v ở đây cũng được thăm đầu tiên, nhưng có khác với DFS ở chỗ là: Sau đó các đỉnh chưa được thăm mà là lân cận của v sẽ được thăm kế tiếp nhau, rồi mới đến các đỉnh chưa được thăm là lân cận lần lượt của các đỉnh này và cứ tương tự như vậy. Sau đây là giải thuật BFS:

```
Procedure BFS(v)
  1) Visited(v) :=1;  //Visited dùng để đánh dấu các đỉnh đã được thăm
  2) Khởi tạo queue với v đã được nạp vào
  3) While Q không rỗng Do
  Begin
     Call pop(v,Q); //Lấy đỉnh v ra khỏi Q
     For  mỗi đỉnh w lân cận với v Do
                  if Visited(w)=0 then
                  Begin
                        Callpush(w,Q);
                        Visited(w) :=1;
           End;
```

Mỗi đỉnh được thăm sẽ được nạp vào queue chỉ một lần vị vậy câu lệnh while lặp lại nhiều nhất n lần.Nếu G được biểu diễn bởi ma trận lân cận thì câu lệnh For sẽ chi phí O(n) thời gian đối với mỗi đỉnh, do đó thời gian chi phí toàn

bộ sẽ là $O(n^2)$. Còn trường hợp G được biểu diễn với danh sách lân cận thì chi phí tổng quát chung là O(e).

Để hiểu rõ hơn về BFS ta nghiên cứu các toán sau:

Bài toán: Gặm cỏ

Bessie rất yêu bãi cỏ của mình và thích thú chạy về chuồng bò vào giờ vắt sữa buổi tối. Bessie đã chia đồng cỏ của mình là 1 vùng hình chữ nhật thành các ô vuông nhỏ với R (1 <= R <= 100) hàng và C (1 <= C <= 100) cột, đồng thời đánh dấu chỗ nào là cỏ và chỗ nào là đá. Bessie đứng ở vị trí R_b,C_b và muốn ăn cỏ theo cách của mình, từng ô vuông một và trở về chuồng ở ô 1,1 ; bên cạnh đó đường đi này phải là ngắn nhất. Bessie có thể đi từ 1 ô vuông sang 4 ô vuông khác kề cạnh.

Dưới đây là một bản đồ ví dụ [với đá ('*'), cỏ ('.'), chuồng bò ('B'), và Bessie ('C') ở hàng 5, cột 6] và một bản đồ cho biết hành trình tối ưu của Bessie, đường đi được dánh dấu bằng chữ 'm'.

Bản đồ	Đường đi tối ưu
1 2 3 4 5 6 <-cột	1 2 3 4 5 6 <-cột
1 B . . . * .	1 B m m m * .
2 . . * . . .	2 . . * m m m
3 . * * . * .	3 . * * . * m
4 . . * * * .	4 . . * * * m
5 * . . * . C	5 * . . * . m

Bessie ăn được 9 ô cỏ.

Cho bản đồ, hãy tính xem có bao nhiêu ô cỏ mà Bessie sẽ ăn được trên con đường ngắn nhất trở về chuồng (tất nhiên trong chuồng không có cỏ đâu nên đừng có tính nhé)

Dữ liệu

- Dòng 1: 2 số nguyên cách nhau bởi dấu cách: R và C
- Dòng 2..R+1: Dòng i+1 mô tả dòng i với C ký tự (và không có dấu cách) như đã nói ở trên.

Kết quả

- Dòng 1: Một số nguyên là số ô cỏ mà Bessie ăn được trên hành trình ngắn nhất trở về chuồng.

Ví dụ

Dữ liệu

5 6

B...*.
..*...
.**.*.
..***.
...C

Kết quả

9

Ý tưởng : Bfs bắt đầu từ đỉnh B, với bảng f[i,j] là độ dài đường đi ngắn nhất từ đỉnh (i,j) đến đỉnh B, kết quả là f[cx,cy];

```
Const
fi            ='VMUNCH.inp';
fo            ='';
MAXN          =1500;
tx            : array [1..4] of longint = (1,0,-1,0);
ty            : array [1..4] of longint = (0,1,0,-1);

Type
pos=record
      x,y    : longint;
      end;

Var
a             : array [0..MAXN,0..MAXN] of boolean;
d             : array [1..MAXN,1..MAXN] of longint;
q             : array [1..MAXN*MAXN] of pos;
r,c,cx,cy     : longint;

Procedure nhap;
var i,j       : longint;
t             : char;
begin
   assign(input,fi);reset(input);
   fillchar(a,sizeof(a),false);
          readln(r,c);
          for i:=1 to r do
            begin
               for j:=1 to c do
               begin
               read(t);a[i,j]:=t='.';
               d[i,j]:=1;
               if t='B' then
                  begin
                     q[1].x:=i;
                     q[1].y:=j;
                  end;
               if t='C' then
                  begin
                     a[i,j]:=true;
```

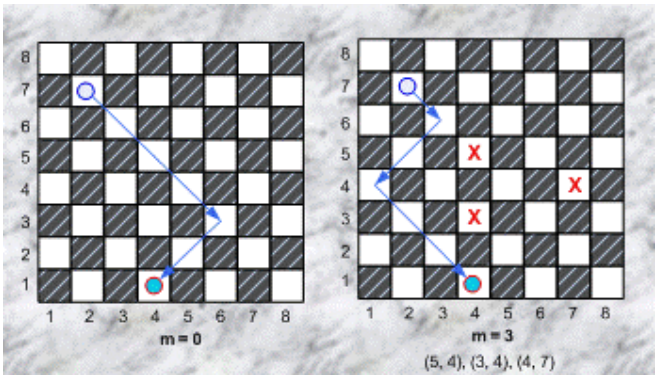
```
Procedure xuly;
var bot,top,x,y,i : longint;
u             : pos;
begin
   bot:=1;top:=1;
   repeat
   u:=q[top];inc(top);
   a[u.x,u.y]:=false;
      for i:=1 to 4 do
         begin
         x:=u.x+tx[i];y:=u.y+ty[i];
         if a[x,y] then
           begin
                                    a[x,y]:=false;
           inc(bot);
              q[bot].x:=x;
              q[bot].y:=y;
              d[x,y]:=d[u.x,u.y]+1;
           end;
         end;
   until top>bot;
end;

Procedure xuat;
begin
   assign(output,fo);rewrite(output);
            writeln(d[cx,cy]-1);
   close(output);
end;

BEGIN
   NHAP;
   XULY;
   XUAT;
END.
```

```
                    cx:=i;
                    cy:=j;
                  end;
                end;
                readln;
              end;
  close(input);
end;
```

Bài toán: VOI06 Quân tượng

Xét bàn cờ vuông kích thước n×n. Các dòng được đánh số từ 1 đến n, từ dưới lên trên. Các cột được đánh số từ 1 đến n từ trái qua phải.

Ô nằm trên giao của dòng i và cột j được gọi là ô (i,j). Trên bàn cờ có m ($0 \le m \le n$) quân cờ. Với $m > 0$, quân cờ thứ i ở ô (r_i, c_i), i = 1,2,..., m. Không có hai quân cờ nào ở trên cùng một ô. Trong số các ô còn lại của bàn cờ, tại ô (p, q) có một quân tượng. Mỗi một nước đi, từ vị trí đang đứng quân tượng chỉ có thể di chuyển đến được những ô trên cùng đường chéo với nó mà trên đường đi không phải qua các ô đã có quân



Cần phải đưa quân tượng từ ô xuất phát (p, q) về ô đích (s,t). Giả thiết là ở ô đích không có quân cờ. Nếu ngoài quân tượng không có quân nào khác trên bàn cờ thì chỉ có 2 trường hợp: hoặc là không thể tới được ô đích, hoặc là tới được sau không quá 2 nước đi (hình trái). Khi trên bàn cờ còn có các quân cờ khác, vấn đề sẽ không còn đơn giản như vậy.

Yêu cầu: Cho kích thước bàn cờ n, số quân cờ hiện có trên bàn cờ m và vị trí của chúng, ô xuất phát và ô đích của quân tượng. Hãy xác định số nước đi ít nhất cần thực hiện để đưa quân tượng về ô đích hoặc đưa ra số -1 nếu điều này không thể thực hiện được.

Input

Dòng đầu tiên chứa 6 số nguyên n, m, p, q, s, t.

Nếu m > 0 thì mỗi dòng thứ i trong m dòng tiếp theo chứa một cặp số nguyên ri , ci xác định vị trí quân thứ i.

Hai số liên tiếp trên cùng một dòng được ghi cách nhau ít nhất một dấu cách.

Output

Gồm 1 dòng duy nhất là số nước đi tìm được

Example

Input:

8 3 7 2 1 4

5 4

3 4

4 7

Output:

3

Hạn chế:

Trong tất cả các test: 1 ≤ n ≤ 200. Có 60% số lượng test với n ≤ 20.

Ý tưởng: giống như bài VMUNCH, chỉ khác nhau cách thêm đỉnh vào trong queue.

```
Const
fi            ='QBBISHOP';
fo            ='';
MAXN          =300;
tx            : array [1..4] of longint = (1,1,-1,-1);
ty            : array [1..4] of longint = (1,-1,1,-1);

Type
pos           = record
                  x,y     : longint;
                end;
Var
a             : array [0..MAXN,0..MAXN] of longint;
free,tick     : array [0..MAXN,0..MAXN] of boolean;
queue         : array [1..MAXN*MAXN] of pos;
n,s,t,p,q     : longint;

Procedure nhap;
var i,u,v,m     : longint;
begin
fillchar(free,sizeof(free),true);
tick:=free;
readln(n,m,p,q,s,t);
for i:=0 to n+1 do
  begin
    tick[0,i]:=false;
    tick[i,0]:=false;
    tick[n+1,i]:=false;
    tick[i,n+1]:=false;
  end;

for i:=1 to m do
  begin
        readln(u,v);
    tick[u,v]:=false;
  end;
```

```
Procedure BFS;
var d,c,i,k     : longint;
u,v             : pos;
begin
   d:=1;c:=1;
   queue[d].x:=p;
   queue[d].y:=q;
   free[p,q]:=false;
   repeat
   u:=queue[d];inc(d);
   for i:=1 to 4 do
     begin
        k:=1;
        while (tick[u.x+k*tx[i],u.y+k*ty[i]]) do
        begin
       if free[u.x+k*tx[i],u.y+k*ty[i]] then
       begin
         free[u.x+k*tx[i],u.y+k*ty[i]]:=false;
         a[u.x+k*tx[i],u.y+k*ty[i]]:=a[u.x,u.y]+1;
         inc(c);
         queue[c].x:=u.x+k*tx[i];
         queue[c].y:=u.y+k*ty[i];
       end;
      inc(k);
      end;
    end;
   until d>c;
end;
Procedure xuat;
begin

   if (s=p) and(t=q) then writeln(0)
   else if a[s,t]=0 then writeln(-1)
   else
   writeln(a[s,t]);
```

end;	end;
	
	BEGIN
	assign(input,fi);reset(input);
	assign(output,fo);rewrite(output);
	NHAP;
	BFS;
	XUAT;
	close(input);
	close(output);
	END.

Bài toán: Laser Phones

FJ mua một hệ thống liên lạc mới cho đàn bò để chúng có thể trò chuyện với nhau trong khi ăn cỏ. Đồng cỏ được mô tả bằng một lưới hình chữ nhậtkích thước WxH (1 <= W <= 100; 1 <= H <= 100).

Hiện tại FJ mới cung cấp điện thoại cho 2 con bò đầu đàn. Tuy nhiên vấn đề là liên lạc giữa hai con bò chỉ thực hiện được nếu đường truyền thông tin giữa chúng không bị chắn. Ở đây, thông tin chỉ được truyềntheo các đường thẳng và dừng lại nếu nó bị chắn bởi núi đá, cây to (kí hiệu bằng các kí tự '*').

Do đó, FJ phải mua thêm một số gương (kí hiệu bằng các kí tự '/' và '\') để đổi hướng đường đi của tia laser. Xét ví dụ minh họa dưới đây :

Kích thước của đồng có là 8x7, H = 8 và W = 7. Hai con bò đầu đàn được kí hiệu là 'C', đá và cây to kí hiệu là '*':

```
7 . . . . . .          7 . . . . . .
6 . . . . . C          6 . . . . . /-C
5 . . . . . *          5 . . . . . | *
4 * * * * * . *        4 * * * * * | *
3 . . . . * . .        3 . . . . * | .
2 . . . . * . .        2 . . . . * | .
1 . C . . * . .        1 . C . . * | .
0 . . . . . .          0 . \-------/ .
0 1 2 3 4 5 6          0 1 2 3 4 5 6
```

Cần xác định M - số lượng gương ít nhất FJ cần mua để có thể đảm bảo liên lạc giữa hai con bò nói trên. Dữ liệu luôn đảm bảo có ít nhất một cách thực hiện.

INPUT

* Dòng 1: Chứa 2 số nguyên W và H cách nhau ít nhất 1 kí tự.

* Dòng 2..H+1: Mô tả cánh đồng, mỗi dòng gồm W kí tự 'C' hoặc '*', và '.'. Thông tin không bị chặn khi đi qua các kí tự '.' và chỉ có 2 chữ 'C'.

Ví dụ :

7 8

.....

.....C

.....*

*****.*

....*..

....*..

.C..*..

.....

OUTPUT

* Dòng 1: Một số nguyên duy nhất ghi giá trị M - số gương ít nhất cần mua.

Ví dụ :

3

Ý tưởng: Giống như bài QBBISHOP, chỉ khác cách thêm đỉnh.

```
{$R+}
Const
fi          =';//MLASERP.INP';
fo          =';
MAXN        =200;

Var
a,free      : array [0..MAXN,0..MAXN] of boolean;
f           : array [0..MAXN,0..MAXN] of longint;
queuex      : array [0..MAXN*MAXN] of longint;
queuey      : array [0..MAXN*MAXN] of longint;
cx,cy       : array [1..2] of longint;
n,m         : longint;
kq          : longint;

Procedure Nhap;
var i,j,l   : longint;
c           : char;
begin
  Readln(n,m);
  l:=0;
```

```
Procedure Xuly;
Const
tx          : array [1..4] of longint = (1,0,-1,0);
ty          : array [1..4] of longint = (0,1,0,-1);
var d,c     : longint;
u,v,x,y,i   : longint;
begin
  d:=1; c:=1; a[cx[1],cy[1]]:=false;
  queuex[1]:=cx[1]; queuey[1]:=cy[1];
  Repeat
  x:=queuex[d]; y:=queuey[d]; Inc(d);
  for i:=1 to 4 do
  begin
    u:=x+tx[i]; v:=y+ty[i];
    While a[u,v] do
    begin
      if free[u,v] then
      begin
        f[u,v]:=f[x,y]+1;
        free[u,v]:=false;
```

```
   for i:=1 to m do
   begin
      for j:=1 to n do
         begin
         Read(c); a[i,j] := c<>'*';
         if upcase(c) = 'C' then
            begin
            Inc(l);
            cx[l]:=i; cy[l]:=j;
            end;
         end;
   Readln;
   end;
end;

Procedure KhoiTao;
var i          : longint;
begin
    for i:=0 to m+1 do a[i,0]:=false;
    for i:=0 to m+1 do a[i,n+1]:=false;
    for i:=0 to n+1 do a[0,i]:=false;
    for i:=0 to n+1 do a[m+1,i]:=false;
    Fillchar(f,sizeof(f),0);
    Fillchar(free,sizeof(free),true);
end;
```

```
            Inc(c);
            QueueX[c]:=u;
            QueueY[c]:=v;
          end
           else
          begin
            if f[u,v]>f[x,y]+1 then
             f[u,v]:=f[x,y]+1;
         end;
   u:=u+tx[i]; v:=v+ty[i];
     end;
   end;
   until d>c;
Kq:=f[cx[2],cy[2]]-1;
if (cx[2] = 0) and (cy[2] = 0) then kq:=0;
end;
BEGIN
   assign(input,fi); reset(input);
   assign(output,fo); rewrite(output);
   Nhap;
          KhoiTao;
   Xuly;
   Writeln(Kq);
   close(input); close(output);
END.
```

Việc nắm vững được phương pháp và cài đặt được thuật toán tìm kiếm theo chiều rộng (DFS) và tìm kiếm theo chiều sâu (BFS) là những nội dung, kĩ năng quan trọng đối với học sinh trong đội tuyển Tin học. Tôi hi vọng, tham luận này trở thành nguồn tài liệu nhỏ bé có ích trong vô vàn nguồn tài liệu đã có hướng dẫn học nội dung đồ thị. Tôi mong nhận được ý kiến đóng góp của các thầy, cô để tham luận hoàn thiện hơn.

Tác giả: Đặng Tuấn Thành

MỘT SỐ BÀI TOÁN VỀ CÂY KHUNG NHỎ NHẤT

Lê Thị Hải Hằng
Trường THPT Chuyên Biên Hòa - Hà Nam

Bài toán cây khung nhỏ nhất là một trong những bài toán tối ưu thuộc phần lý thuyết đồ thị. Như chúng ta biết, có 2 thuật toán để giải quyết bài toán này, đó là thuật toán Prim và thuật toán Kruskal, trong cuốn *Tài liệu Giáo khoa chuyên Tin (Quyển 2)* đã trình bày rất kỹ thuật toán, hướng dẫn cách cài đặt cụ thể và đánh giá độ phức tạp tính toán. Trong bài viết này, tôi xin đưa ra một số bài tập áp dụng thuật toán.

i toán 1: Vòng đua F1- Mã bài: NKRACING

Singapore sẽ tổ chức một cuộc đua xe Công Thức 1 vào năm 2008. Trước khi cuộc đua diễn ra, đã xuất hiện một số cuộc đua về đêm trái luật. Chính quyền muốn thiết kế một hệ thống kiểm soát giao thông để bắt giữ các tay đua phạm luật. Hệ thống bao gồm một số camera đặt trên các tuyến đường khác nhau. Để đảm bảo tính hiệu quả cho hệ thống, cần có ít nhất một camera dọc theo mỗi vòng đua.

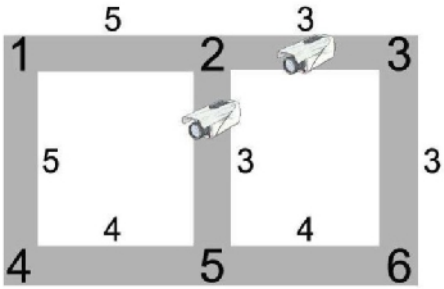
Hệ thống đường ở Singapore có thể được mô tả bởi một dãy các nút giao thông và các đường nối hai chiều (xem hình vẽ). Một vòng đua bao gồm một nút giao thông xuất phát, tiếp theo là đường đi bao gồm ít nhất 3 tuyến đường và cuối cùng quay trở lại điểm xuất phát. Trong một vòng đua, mỗi tuyến đường chỉ được đi qua đúng một lần, theo đúng một hướng.

Chi phí để đặt camera phụ thuộc vào tuyến đường được chọn. Các số nhỏ trong hình vẽ cho biết chi phí để đặt camera lên các tuyến đường. Các số lớn xác định các nút giao thông. Camera được đặt trên các tuyến đường chứ không phải tại các nút giao thông. Bạn cần chọn một số tuyến đường sao cho chi phí lắp đặt là thấp nhất đồng thời vẫn đảm bảo có ít nhất một camera dọc theo mỗi vòng đua.

Viết chương trình tìm cách đặt các camera theo dõi giao thông sao cho tổng chi phí lắp đặt là thấp nhất.

Dữ liệu

- Dòng đầu tiên chứa 2 số nguyên n, m ($1 \le n \le 10000$, $1 \le m \le 100000$) là



số nút giao thông và số đường nối. Các nút giao thông được đánh số từ 1 đến n.

- m dòng tiếp theo mô tả các đường nối, mỗi dòng bao gồm 3 số nguyên dương cho biết hai đầu mút của tuyến đường và chi phí lắp đặt camera. Chi phí lắp đặt thuộc phạm vi [1, 1000].

Kết quả

In ra 1 số nguyên duy nhất là tổng chi phí lắp đặt thất nhất tìm được.

Ví dụ

Dữ liệu:

6 7
1 2 5
2 3 3
1 4 5
4 5 4
5 6 4
6 3 3
5 2 3 *Kết quả*
6

Thuật toán:

Ban đầu ta giả sử đã đặt camera ở mọi tuyến đường, như vậy cần tìm cách bỏ đi một số các camera với tổng chi phí giảm được là lớn nhất.

Tập hợp các tuyến đường bỏ đi không được chứa chu trình vì nếu chứa sẽ tạo ra một vòng đua không được giám sát, suy ra chỉ có thể bỏ đi nhiều nhất là n-1 camera ở n-1 tuyến đường và n-1 tuyến đường đó là một cây khung của đồ thị.

Để giảm được nhiều chi phí nhất thì cần tìm cây khung lớn nhất của đồ thị để bỏ camera trên các cạnh của cây khung đó.

Chương trình:

```
{$mode objfpc}
const
fi='nkracing.inp';
fo='nkracing.out';
max=10000;
maxm=100000;
vc=100000000;
var f:text;
n,m,kq:longint;
x,y,c:array[0..maxm+1]of longint;
{a,ts:array[0..maxm*2+1]of longint;}
goc:array[0..max+1]of longint;
chon:array[0..maxm+1]of longint;
dd:array[0..max+1]of boolean;
procedure doc;
    var i,j:longint;
    begin
        assign(f,fi);
        reset(f);
        readln(f,n,m);
        kq:=0;
        for i:=1 to m do
            begin
            read(f,x[i],y[i],c[i]);
            kq:=kq+c[i];
            end;
        close(f);
    end;
procedure viet;
    var i,j:longint;
    begin
        assign(f,fo);
        rewrite(f);
        writeln(f,kq);
        close(f);
    end;
function laygoc(u:longint):longint;
    begin
        while goc[u]<>-1 do
            u:=goc[u];
        laygoc:=u;

    end;
procedure doi(var i,j:longint);
    var tg:longint;
    begin
```

```
        tg:=i;
        i:=j;
        j:=tg;
    end;
procedure sort(d1,c1:longint);
    var i,j,gt:longint;
    begin
        if d1>=c1 then exit;
        i:=d1;
        j:=c1;
        gt:=c[(c1+d1)div 2];
        repeat
            while c[i]>gt do inc(i);
            while c[j]<gt do dec(j);
            if i<=j then
                begin
                if i<j then
                    begin
                    doi(x[i],x[j]);
                    doi(y[i],y[j]);
                    doi(c[i],c[j]);
                    end;
                dec(j);
                inc(i);
                end;
        until i>j;
        sort(d1,j);
        sort(i,c1);
    end;

procedure lam;
    var i,j,dem,u,v,i1,j1,p:longint;
    begin
        for i:=0 to n do
            goc[i]:=-1;
        sort(1,m);
        dem:=0;
        for i:=1 to m do
            begin
            u:=laygoc(x[i]);
            v:=laygoc(y[i]);
            if u<>v then
                begin
                inc(dem);
                goc[u]:=x[i];
                kq:=kq-c[i];
                goc[x[i]]:=y[i];
```

```
                chon[dem]:=i;
                if dem=n-1 then break;
                end;
            end;

    end;
BEGIN
doc;
lam;
viet;
END.
```

i toán 2: Xây dựng thành phố - Mã bài: NKCITY

Nước Anpha đang lập kế hoạch xây dựng một thành phố mới và hiện đại. Theo kế hoạch, thành phố sẽ có N vị trí quan trọng, được gọi là N trọng điểm và các trọng điểm này được đánh số từ 1 tới N. Bộ giao thông đã lập ra một danh sách M tuyến đường hai chiều có thể xây dựng được giữa hai trọng điểm nào đó. Mỗi tuyến đường có một thời gian hoàn thành khác nhau.

Các tuyến đường phải được xây dựng sao cho N trọng điểm liên thông với nhau. Nói cách khác, giữa hai trọng điểm bất kỳ cần phải di chuyển được đến nhau qua một số tuyến đường. Bộ giao thông sẽ chọn ra một số tuyến đường từ trong danh sách ban đầu để đưa vào xây dựng sao cho điều kiện này được thỏa mãn.

Do nhận được đầu tư rất lớn từ chính phủ, bộ giao thông sẽ thuê hẳn một đội thi công riêng cho mỗi tuyến đường cần xây dựng. Do đó, thời gian để hoàn thành toàn bộ các tuyến đường cần xây dựng sẽ bằng thời gian lâu nhất hoàn thành một tuyến đường nào đó.

Yêu cầu: Giúp bộ giao thông tính thời gian hoàn thành các tuyến đường sớm nhất thỏa mãn yêu cầu đã nêu.

Dữ liệu

Dòng chứa số N và M ($1 \le N \le 1000$; $1 \le M \le 10000$).

M tiếp theo, mỗi dòng chứa ba số nguyên u, v và t cho biết có thể xây dựng tuyến đường nối giữa trọng điểm u và trọng điểm v trong thời gian t. Không có hai tuyến đường nào nối cùng một cặp trọng điểm.

Kết quả

Một số nguyên duy nhất là thời gian sớm nhất hoàn thành các tuyến đường thỏa mãn yêu cầu đã nêu.

Ví dụ

Dữ liệu

5 7
1 2 2
1 5 1
2 5 1
1 4 3
1 3 2
5 3 2
3 4 4 *Kết quả*

Thuật toán:

Đề bài là tìm ra cây khung có cạnh lớn nhất là nhỏ nhất và đưa ra cạnh lớn nhất đó, tuy nhiên tôi nghĩ rằng mọi cây khung nếu đã là nhỏ nhất thì cạnh lớn nhất của nó cũng là nhỏ nhất trong số các cạnh lớn nhất của các cây khung.

Vì vậy, tôi dùng thuật toán Kruskal tìm cây khung nhỏ nhất áp dụng cho bài toán này, cạnh cuối cùng được thêm vào là cạnh lớn nhất của cây khung.

Chương trình:

```
{$mode objfpc}
const
fi='nkcity.inp';
fo='nkcity.out';
max=1000;
maxm=10000;
vc=100000000;
var f:text;
n,m,kq1,kq2:longint;
x,y,c:array[0..maxm+1]of longint;
{a,ts:array[0..maxm*2+1]of longint;}
goc:array[0..max+1]of longint;
chon:array[0..maxm+1]of longint;
dd:array[0..max+1]of boolean;
procedure doc;
    var i,j:longint;
    begin
        assign(f,fi);
        reset(f);
```

```
        readln(f,n,m);
        for i:=1 to m do
            begin
            read(f,x[i],y[i],c[i]);
            end;
        close(f);
    end;
procedure viet;
    var i,j:longint;
    begin
        assign(f,fo);
        rewrite(f);
        writeln(f,kq1);
        close(f);
    end;
function laygoc(u:longint):longint;
    begin
        while goc[u]<>-1 do
            u:=goc[u];
        laygoc:=u;

    end;
procedure doi(var i,j:longint);
    var tg:longint;
    begin
        tg:=i;
        i:=j;
        j:=tg;
    end;
procedure sort(d1,c1:longint);
    var i,j,gt:longint;
    begin
        if d1>=c1 then exit;
        i:=d1;
        j:=c1;
        gt:=c[(c1+d1)div 2];
        repeat
            while c[i]<gt do inc(i);
            while c[j]>gt do dec(j);
            if i<=j then
                begin
                if i<j then
                    begin
                    doi(x[i],x[j]);
                    doi(y[i],y[j]);
                    doi(c[i],c[j]);
                    end;
```

```
                dec(j);
                inc(i);
                end;
        until i>j;
        sort(d1,j);
        sort(i,c1);
    end;

procedure lam;
    var i,j,dem,u,v,i1,j1,p:longint;
    begin
        for i:=0 to n do
            goc[i]:=-1;
        sort(1,m);
        kq1:=0;
        dem:=0;
        for i:=1 to m do
            begin
            u:=laygoc(x[i]);
            v:=laygoc(y[i]);
            if u<>v then
                begin
                inc(dem);
                kq1:=c[i];
                goc[u]:=x[i];
                goc[x[i]]:=y[i];
                chon[dem]:=i;
                if dem=n-1 then break;
                end;
            end;

    end;
BEGIN
doc;
lam;
viet;
END.
```

Bài toán 3: Mạng truyền thông - Mã bài: COMNET *(Đề thi HSG QG 2013)*

Tổng công ty Z gồm N công ty con, đánh số từ 1-N. Mỗi công ty con có một máy chủ. Để đảm bảo truyền tin giữa các công ty, Z thuê M đường truyền tin để kết nối N máy chủ thành một mạng máy tính của Tổng công ty. Không có 2 đường truyền nối cùng 1 cặp máy chủ. Đường truyền i nối máy chủ của 2 công ty u_i, v_i có chi phí là w_i. Mạng máy tính có tính *thông suốt*, nghĩa là từ

một máy chủ có thể truyền tin đến một máy chủ bất kì khác bằng đường truyền trực tiếp hoặc qua nhiều đường trung gian.

Một đường truyền gọi là *không tiềm năng* nếu như : một mặt, việc loại bỏ đường truyền này không làm mất tính thông suốt; mặt khác, nó phải có tính không tiềm năng, nghĩa là không thuộc bất cứ mạng con thông suốt gồm N máy chủ và N-1 đường truyền tin với tổng chi phí thuê bao nhỏ nhất nào của mạng máy tính.

Trong thời gian tới, chi phí thuê bao của một số đường truyền tin thay đổi. Tổng công ty muốn xác định với chi phí mới thì đường truyền thứ k có là đường không tiềm năng hay không để xem xét chấm dứt việc thuê đường truyền này.

Yêu cầu: Cho Q giả định, mỗi giả định cho biết danh sách các đường truyền tin với chi phí thuê mới và chỉ số k. Với mỗi giả định về chi phí mới thuê đường truyền tin, hãy xác định đường truyền tin thứ k có là đường truyền tin không tiềm năng trong mạng không.

Input

- Dòng đầu là T – số testcase. T nhóm dòng, mỗi nhóm cho thông tin về một testcase.
- Dòng thứ nhất gồm 3 số nguyên dương N, M, Q (Q <= 30).
- Dòng thứ i trong M dòng tiếp theo chứa 3 số nguyên dương u_i, v_i, w_i ($u_i \neq v_i$, $w_i < 10^9$).
- Dòng thứ j trong Q dòng tiếp theo mô tả giả định thứ j:
  - Số đầu tiên là chỉ số k_j của đường truyền tin cần xem xét
  - Tiếp theo là s_j (s_j <= 100) cho biết số lượng đường truyền có chi phí thuê mới
  - Cuối cùng là s_j cặp số nguyên dương t_p, c_p cho biết đường truyền thứ t_p có chi phí thuê mới là c_p ($c_p < 10^9$).

Output

- Gồm T nhóm dòng, mỗi nhóm gồm Q dòng. Mỗi dòng là câu trả lời cho giả định tương ứng trong input. Ghi YES nếu câu trả lời là khẳng định và NO trong trường hợp ngược lại.

Example

Input:	*Output:*

1 3 3 2 1 2 1 1 3 2 2 3 3 3 2 2 4 3 4 1 1 1 4	NO YES

Giới hạn

- 30% số test đầu có $1 \le N \le 100$;
- 30% số test tiếp theo có $1 \le N \le 10^4$ và $1 \le M \le 10^5$;
- 40% số test còn lại có $1 \le N \le 10^5$ và $1 \le M \le 10^6$.

Thuật toán:

Ta tóm tắt đề bài như sau: Cho đồ thị vô hướng N đỉnh M cạnh và Q truy vấn. Mỗi truy vấn yêu cầu thay đổi trọng số S cạnh của đồ thị và hỏi xem cạnh K có thuộc mọi cây khung nhỏ nhất của đồ thị hay không.

Nhận thấy, nếu sau khi bỏ cạnh K khỏi đồ thị ta không tìm được cây khung hoặc tìm được cây khung nhỏ nhất nhưng có trọng số lớn hơn ban đầu thì K sẽ là cạnh nằm trên mọi cây khung nhỏ nhất. Độ phức tạp O(Q x độ phức tạp tìm cây khung nhỏ nhất).

30% số test đầu: cài đặt thuật toán Prim hoặc Kruskal thông thường.

30% số test tiếp theo, ta cải tiến thuật toán Prim sử dụng cấu trúc dữ liệu Heap có độ phức tạp O(Q x NlogN), hoặc dùng thuật toán Kruskal với cấu trúc dữ liệu Disjoint-set forest- độ phức tạp O(Q x (O(MlogM)+O(N))), trong đó O(MlogM) là chi phí sắp xếp M cạnh và O(N) là chi phí quản lý Disjoint-set forest.

Để đạt 100% số test ta cũng dùng dùng thuật toán Kruskal với cấu trúc dữ liệu Disjoint-set forest, duyệt hết các cạnh có trọng số nhỏ hơn cạnh K, khi duyệt đến cạnh (u,v) thì ta hợp tập chứa cạnh u và tập chứa cạnh v lại, Cuối cùng cạnh K là cạnh tiềm năng nếu nó nối hai tập rời nhau.

Chương trình:

```
Program comnet;
const
  fi='comnet.inp';
  fo='comnet.out';
  mn=100000+100;
```

```
  mm=1000000+1000;
type
  tedge=record
         u,v,w:longint;
        end;
Var
  edge:array[0..mm] of tedge;
  tmp:array[0..mm] of tedge;
  p:array[0..mn] of longint;
  n,m,q:longint;
  ntest:longint;
Function getRoot(u:longint):Longint;
begin
    if p[u]=u then exit(u);
    p[u]:=getRoot(p[u]);
    exit(p[u]);
end;
procedure union(u,v:longint);
begin
  u:=getRoot(u);
  v:=getRoot(v);
  if u=v then exit;
  p[u]:=v;
end;
procedure solve;
var i,k,s,t,c:longint;
begin
  readln(n,m,q);
  for i:=1 to m do
    with edge[i] do
      readln(u,v,w);
  while q>0 do
    begin
      dec(q);
      // dung mang tmp de luu trong so cac canh ban dau
      for i:=1 to m do
        tmp[i]:=edge[i];
      read(k,s);
      // thay doi s canh teo truy van
      for i:=1 to s do
        begin
          read(t,c);
          tmp[t].w:=c;
        end;
      //khoi tao disjoin set
      for i:=1 to n do
        p[i]:=i;
      //duyet qua cac canh co trong so nho hon canh K
```

```
      for i:=1 to m do
        with tmp[i] do
          if w<tmp[k].w then union(u,v);
      // thu xem canh k co noi 2 dinh thuoc 2 tap roi nhau hay khong
      with tmp[k] do
        begin
          if getRoot(u)<>getRoot(v) then writeln('YES')
          else writeln('NO');
        end;
    end;
end;

begin{mai}
  assign(input,fi);
  reset(input);
  assign(output,fo);
  rewrite(output);
  readln(ntest);
  while ntest>0 do
    begin
      dec(ntest);
      solve;
    end;
end.
```

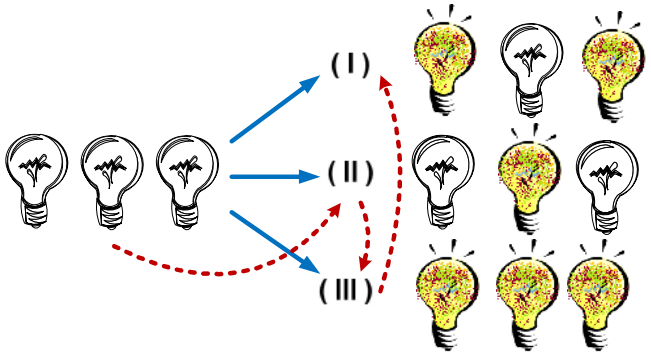
Rèn luyện kỹ năng duyệt trong đồ thị

Nguyễn Quang Minh, trường THPT chuyên Hạ Long

Để rèn luyện kỹ năng duyệt trong đồ thị , tôi xin giới thiệu cách giải một vài bài toán sau đây:

Bài 1 ĐÈN TRANG TRÍ

Rôn mua một bộ đèn trang trí gồm *n* đèn ($1 \le n \le 1\,000$). Mỗi đèn có một công tắc để bật hay tắt riêng đèn đó. Mỗi giây Rôn có thể bật hoặc tắt một bóng đèn tùy chọn. Ban đầu tất cả các bóng đều ở trạng thái tắt. Một cấu hình của bộ đèn là trạng thái khi một số đèn nào đó được bật sáng, những đèn còn lại – tắt. Rôn đặc biệt thích một số cấu hình vì chúng có vẻ phù hợp với khung cảnh căn phòng của Rôn.Mỗi trạng thái của bộ đèn được biểu diễn bằng một xâu *n* ký tự từ tập {0, 1}. Ký tự thứ *i* xác định trạng thái đèn thứ *i*, 0 tương ứng với trạng thái đèn tắt, 1 là trạng thái đèn được bật sáng. Ví dụ, với *n* = 3 và Rôn đặc biệt thích 3 cấu hình {1, 0, 1}, {0, 1, 0}, {1, 1, 1}. Để kiểm tra xem cấu hình nào là thích hợp nhất Rôn phải lần lượt bật tắt một số đèn. Trong trường hợp này Rôn cần 4 giây để xem xét hết mọi cấu hình.



Yêu cầu: Cho biết ***n*** và ***m***, trong đó ***m*** – số cấu hình khác nhau mà Rôn đặc biệt yêu thích ($1 \le m \le 15$). Hãy xác định thời gian tối thiểu cần thiết để kiểm tra hết tất cả các trạng thái mà Rôn quan tâm.

Dữ liệu: Vào từ file văn bản GARLAN.INP:

- Dòng đầu tiên chứa 2 số nguyên ***n*** và ***m***,
- Mỗi dòng trong ***m*** dòng tiếp theo chứa xâu ***n*** ký tự xác định một cấu hình Rôn yêu thích.

Kết quả: Đưa ra file văn bản GARLAN.OUT một số nguyên – thời gian tối thiểu kiểm tra các cấu hình.

Ví dụ:

GARLAN.INP
3 3 101 010 111

GARLAN.OUT
4

Lời giải : - Mỗi trạng thái coi như 1 đỉnh của đồ thị (trạng thái ban đầu là đỉnh số 0)

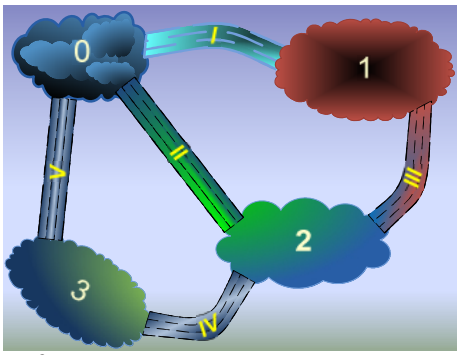
- **Trong số của mỗi cạnh là chi phí chuyển từ trạng thái nọ sang trạng thái kia**
- **Bìa toán trở thành tìm đường đi từ 0 qua lần lượt các đỉnh với tổng trọng số nhỏ nhất**

- **Gọi mỗi canh là (u,v,w), trong đó u,v là 2 đỉnh , w là trọng số .**

Trong ví dụ trên ta có các cạnh (0,1,2) , (0,2,1) , (0,3,3) , (1,2,3) , (1,3,1) , (2,3,2)

Ta được đường đi tốt nhất là 0-2-1-3 (hoặc 0-2-3-1) với chi phí = 4.

Bài 2 BẮC CẦU



Chính phủ quốc đảo Oceani quyết định xây dựng ***m*** chiếc cầu nối ***n*** đảo của mình, tạo một mạng lưới giao thông đường bộ cho phép đi từ dảo bất kỳ tới đảo khác bằng đường bộ (trực tiếp hoặc qua một số đảo trung gian). Mỗi cây cầu sẽ nối 2 đảo khác nhau và cho phép đi lại hai chiều. Các đảo được đánh số từ 0 đến ***n***-1. Bị hạn chế bởi kinh phí và nguồn nhân lực, người ta quyết định sẽ xây dựng lần lượt từng chiếc cầu một và lên kế hoạch xác định cầu và trình tự xây. Mỗi cây cầu được xác định bởi cặp đảo ***u***, ***v*** mà nó nối. Trong quá trình thực hiện kế hoạch có thể đến một lúc nào đó từ một đảo đã có thể đi đến bất kỳ đảo khác bằng đường bộ.

Ví dụ, ở Oceani có 4 đảo và người ta quyết định xây dựng 5 cầu theo trình tự lần lượt là 0 – 1, 0 – 2, 1 – 2, 2 – 3, 3 – 0. Tuy vậy, không cần chờ đợi đến khi hoàn thành kế hoạch xây cầu, sau khi cầu thứ 4 được xây xong tất cả các đảo đã được nối liền bằng đường bộ.

Yêu cầu: Cho ***n***, ***m*** và các cây cầu dự kiến xây. Thông tin về các cây cầu đưa ra theo đúng trình tự xây dựng. Hãy xác định số cầu tối thiểu cần xây theo kế hoạch để từ một đảo đã có thể đi đến bất kỳ đảo khác bằng đường bộ.

Dữ liệu: Vào từ file văn bản BRIDGES.INP:

- Dòng đầu tiên chứa 2 số nguyên ***n*** và ***m*** ($1 \le n \le 10^6$, $1 \le m \le 5\times10^6$),
- Dòng thứ ***i*** trong ***m*** dòng tiếp theo chứa 2 số nguyên ***u*** và ***v*** xác định cây cầu thứ ***i*** cần xây.

Kết quả: Đưa ra file văn bản BRIDGES.OUT kết quả tìm được dưới dạng một số nguyên.

Ví dụ:

BRIDGES.INP
4 5 0 1 0 2 1 2 2 3 3 0

BRIDGES.OUT
4

Lời giải : - Tìm đáp số bằng tìm kiếm nhị phân (Ds min = n-1 , ds max = m)

- V**ới mỗi ds dự đoán , ta chỉ việc kiểm tra tính liên thông với danh sách cạnh từ 1 đến ds .**

Bài 3 Đường đi của Robot

Một bảng hình chữ nhật có kích thước MxN (M,N nguyên dương và không lớn hơn 100) được chia thành các ô vuông đơn vị bằng các đường thẳng song song với các cạnh. Một số ô vuông nào đó có thể đặt các vật cản. Từ một ô vuông, Robot có thể đi đến một ô vuông kề cạnh với nó nếu ô vuông đó không có vật cản. Hỏi rằng nếu Robot bắt đầu xuất phát từ một ô vuông không có vật cản thuộc dòng K, cột L thì có thể đi đến được ô vuông không có vật cản thuộc dòng H, cột O hay không? Nếu có thì hãy chỉ ra đường đi qua ít ô vuông nhất.

Dữ liệu vào là tệp văn bản BAI3.INP có cấu trúc:

- Dòng đầu tiên ghi các chữ số M, N, K, L, H, O. Các số ghi cách nhau ít nhất một ký tự trống;

- M dòng tiếp theo, mỗi dòng ghi N số 1 hoặc 0 tuỳ thuộc vào ô vuông tương ứng trong bảng hình chữ nhật nêu trên có vật cản hay không (ghi số 1 nếu có vật cản); các số trên mỗi dòng ghi liên tiếp nhau.

Dữ liệu ra là tệp văn bản BAI3.OUT có cấu trúc:

Nếu Robot có thể đi được từ ô vuông thuộc dòng K, cột L đến ô vuông thuộc dòng H, cột O thì:

- Dòng đầu tiên ghi ‘Co duong di ‘;

- Các dòng tiếp theo, mỗi dòng ghi 2 số là chỉ số dòng và chỉ số cột của các ô vuông trong đường đi tìm được từ ô vuông thuộc dòng K, cột L đến ô vuông thuộc dòng H, cột O mà qua ít ô vuông nhất. Hai số trên mỗi dòng ghi cách nhau ít nhất một ký tự trống;

- Ngược lại, nếu Robot không thể đi được từ ô vuông thuộc dòng K, cột L đến ô vuông thuộc dòng H, cột O thì ghi ‘Khong co duong di’.

Ví dụ 1:

robot.inp:	robot.out:
4 7 3 4 2 6	Co duong di
1000000	3 4
0010100	3 5
0000000	3 6
1101000	2 6

Ví dụ 2:

robot.inp:	robot.out:
4 7 2 2 1 3	Khong co duong di
1010000	
0010100	
0100000	
1101000	

Phân tích:

Yêu cầu của bài toán thực chất là tìm đường đi từ ô [K,L] đến ô [H,O] sao cho qua ít ô vuông nhất. Ta dễ thấy thuật toán để xử lý một cách hợp lý nhất là thuật toán Loang. Ta bắt dầu "loang" từ ô [K,L], nếu "loang" đến được ô [H,O] thì có đường đi, ngược lại không có đường đi.

Hàng đợi phục vụ "loang" được thể hiện bởi mảng 2 chiều Q:Array[1..2,Mmax*Max] of Byte; hàng thứ 1 của Q để lưu thông tin chỉ số hàng, hàng thứ 2 lưu thông tin của chỉ số cột của các ô khi nạp vào Q.

Mảng A lưu thông tin tình trạng các ô - có vật cản hay không của bảng hình chữ nhật chứa các ô vuông.

Mảng P dùng để đánh dấu những ô đã "loang" đến; đồng thời để phục vụ cho việc truy xuất đường đi sau này nên khi ô [i,j] được "loang" đến thì P[i,j] được gán giá trị là r (r là giá trị tương ứng với hướng mà ô trước đó "loang" đến, hướng nào tương ứng với giá trị k bao nhiêu tuỳ theo quy định, ví dụ r = 1 - sang phải, 2 - đi xuống, 3 - sang trái, 4 - đi lên).

Sau khi thực hiện xong việc "loang", nếu P[H,O] = 0 thì điều có có nghĩa là ô [H,O] chưa được "loang" đến (không có đường đi), nếu P[H,O] = r (r=1..4 - loang theo 4 hướng) thì dựa vào hướng "loang" đến mà ta tìm được ô trước đó, rồi ta lại dựa vào giá trị k của ô tìm được ta tìm được ô trước đó nữa ... quá trình trên kết thúc khi tìm được ô [K,L].

Sau khi "loang" xong thì giá trị các phần tử trong mảng Q không còn giá trị sử dụng nữa nên ta có thể dùng mảng Q phục vụ cho việc truy xuất kết quả.

Bài 4 Gặp gỡ của hai Robot.

Trên một lưới ô vuông MxN (M,N<100), người ta đặt Robot A ở góc trái trên, Robot B ở góc phải dưới. Mỗi ô của lưới ô có thể đặt một vật cản hoặc không (ô trái trên và ô phải dưới không có vật cản). Hai Robot bắt đầu di chuyển đồng thời với tốc độ như nhau và không Robot nào được dừng lại trong khi Robot kia di chuyển (trừ khi nó không thể đi được nữa). Tại mỗi bước, Robot chỉ có thể di chuyển theo 4 hướng - đi lên, đi xuống, sang trái, sang phải - vào các ô kề cạnh. Hai Robot gặp nhau nếu chúng cùng đứng trong một ô vuông. Bài toán đặt ra là tìm cách di chuyển ít nhất mà 2 Robot phải thực hiện để có thể gặp nhau.

Dữ liệu vào cho bởi tệp robot.inp:

- Dòng đầu ghi 2 số M, N cách nhau ít nhất một ký tự trống;

- M dòng tiếp theo, mỗi dòng ghi N số 0 hoặc 1 liên tiếp nhau mô tả trạng thái của các ô vuông: 1 - có vật cản, 0 - không có vật cản.

Dữ liệu ra ghi vào tệp robot.out:

- Nếu 2 Robot không thể gặp nhau thì ghi ký tự '#'.

- Ngược lại, ghi hai dòng, mỗi dòng là một dãy các ký tự viết liền nhau mô tả các bước đi của Robot: U - đi lên, D - đi xuống, L - sang trái, R - sang phải. Dòng đầu là các bước đi của Robot A, dòng sau là các bước đi của Robot B.

Ví dụ:

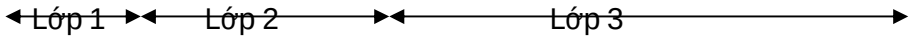
robot.inp	robot.out	robot.inp	robot.out
4 6 011000 000001 001001 010100	DRRR LULU	3 4 0000 0000 0000	#

Phân tích:

Với dạng bài toán như vậy thì ta nghĩ ngay đến thuật toán Loang để tìm đường đi cho 2 Robot. Như vậy là phải "loang" từ 2 phía (loang của Robot A và loang của Robot B). Nhưng vì 2 Robot di chuyển đồng thời trong khi không cho phép ta cài đặt việc "loang" song song từ 2 phía nên ta phải thiết kế "loang" thế nào cho hợp lý.

Xin đề xuất một ý tưởng "loang" như sau: Cứ Robot A loang 1 lớp thì dừng lại để Robot B loang 1 lớp, quá trình đó được lặp đi lặp lại cho đến khi 2 Robot gặp nhau tại một ô hoặc 1 trong 2 Robot dừng "loang". Một lớp "loang" ở đây là "loang" từ các phần tử hiện có trong hàng đợi (từ phần tử Queue[dau] đến phần tử Queue[cuoi]). Sau mỗi lớp "loang", biến dau và biến cuoi lại được điều chỉnh để trở thành vị trí đầu và vị trí

cuối của các phần tử mới trong Queue. Ta có thể mô tả cụ thể các lớp "loang" của 2 Robot với dữ liệu vào là tệp robot.inp thứ 2 ở trên:



Queue	1	2	1	3	1	2	1	.....
Robot A	1	1	2	1	1	2	3	

Queue	3	3	2	3	2	3	1	.....
Robot B	4	3	4	2	3	4	4	

Lớp 1 Lớp 2 Lớp 3

Q1,Q2 là 2 mảng dùng để biểu diễn cấu trúc hàng đợi để phục vụ việc "loang" của 2 Robot. Trong quá trình "loang" ta phải lưu giữ thông tin hàng, cột của ô khi "loang" đến, bởi vậy các phần tử của Q1, Q2 là các record có kiểu HC

```
HC = Record
  h,c:Byte; {h: lưu chỉ số hàng, c: lưu chỉ số cột}
 end;
```

Hai hàng đợi Q1, Q2 được khởi tạo như sau:

```
Procedure KT_Queue;
  Begin
    dau1:=1;
    cuoi1:=1;
    Q1[cuoi1]:=1;
    Q1[cuoi1]:=1; {Robot A xuất phát từ ô [1,1]}
    dau2:=1;
```

```
    cuoi2:=1;
    Q2[cuoi2]:=M;
    Q2[cuoi2]:=N; {Robot B xuất phát từ ô [M,N]}
  End;
```

Ngay sau khi khởi tạo thì trong Q1 chứa ô [1,1], Q2 chứa ô [M,N]. Đó là các ô xuất phát để "loang" của 2 Robot.

Mỗi Robot từ một ô có thể "loang" theo bốn hướng: đi xuống, sang trái, đi lên, sang phải; nên để thuận tiện cho việc cài đặt ta sử dụng kỷ thuật "rào": Mảng A[i,j] chứa thông tin các ô trong lưới ô vuông được khai báo A:Array[0..Mmax + 1,0..Nmax + 1] of Byte (chứ không phải như thông thường là [1..Mmax,1..Nmax]) và được khởi tạo FillChar(A,SizeOf(A),1) (như vậy là xung quanh lưới ô vuông được "rào" bởi số 1); đồng thời sử dụng 2 mảng hằng Hi=(1,0,-1,0), Hj=(0,-1,0,1).

Khi đó việc "loang" theo lớp của Robot A được thực hiện như sau:

```
Procedure LoangA;
  Var
    k:Byte;
  Begin
    j:=Cuoi1;
    For i:=dau1 to cuoi1 do
      For k:=1 to 4 do
        Begin
          h:= Q1[i].h + Hi[k]; {k=1 - đi xuống, 2 - sang trái, 3 - đi lên, 4 - sang phải}
          c:= Q1[i].c  + Hj[k];
          If A[h,c] = 0 then {ô [h,c] không có vật cản và chưa "loang" đến}
            Begin
```

```
                Inc(j);
                Q1[j].h:= h;
                Q1[j].c:= c; {Nạp ô [h,c] vào hàng đợi Q1}
                A[h,c]:=k;{Đánh dấu ô bằng cách gán giá trị tương ứng với hướng loang}
                B[h,c]:=True; {Dấu hiệu cho Robot B nhận biết đã gặp Robot A}
              End;
          End;
      dau1:=cuoi1 + 1;
      cuoi1:=j; {Điều chỉnh lại biến dau1, cuoi1 cho các phần tử mới trong Q1}
      If dau1 > cuoi1 then ST:=True; {ST=True là Q1 rỗng, kết thúc "loang"}
  End;
```

Việc "loang" theo lớp của Robot B cũng tương tự như Robot A nhưng chỉ khác ở chổ khi "loang" đến một ô [h,c] nào đó thì phải xét dấu hiệu B[h,c] xem thử đã gặp Robot A chưa:

```
.....
If B[h,c] then    {Nếu tại ô [h,c] Robot B gặp Robot A thì}
  Begin
    lk:=k; {Lưu lại giá trị tương ứng với hướng "loang" để lấy kết quả}
    hm:=h; {Lưu lại chỉ số hàng của ô mà 2 Robot gặp nhau để lấy kết quả}
    cm:=c; {Lưu lại chỉ số cột của ô mà 2 Robot gặp nhau để lấy kết quả}
    TT:=True; {Dấu hiệu dừng "loang" của 2 Robot vì đã gặp nhau}
    Exit;
  End;
```

.....

Sở dĩ ta phải lưu lại giá trị tương ứng với hướng "loang" (lk:=k) là vì tại ô gặp nhau [h,c] Robot A đã "loang" đến trước nên đã gán giá trị của A[h,c] bằng giá trị tương ứng với hướng "loang" đến nên khi Robot B "loang" đến ô [h,c] buộc ta phải lưu lại giá trị tương ứng với hướng "loang" vào biến lk để sau này truy xuất đường đi của Robot B.

Quá trình "loang" theo từng lớp của 2 Robot được thực hiện như sau:

```
Procedure Loang_lop;
 Begin
  TT:=False;
  ST:=False;
  While (ST=False) and (TT=False) do
   Begin
    FillChar(B,SizeOf(B),False); {Đánh dấu theo từng lớp loang}
    Loang1;
    Loang2;
   End;
 End;
```

Lệnh đánh dấu theo từng lớp "loang" tại vị trí như ở trên: FillChar(B,SizeOf(B),False) là rất quan trọng vì Robot B gặp Robot A tại ô [h,c] chỉ khi B[h,c] = True tại thời điểm lớp "loang" của Robot A cùng lớp "loang" với Robot B. Còn nếu B[h,c] = True của lớp "loang" trước nào đó của Robot A thì không thể kết luận 2 Robot gặp nhau vì khi đó 2 Robot sẽ di chuyển khập khểnh chứ không đồng thời.

Việc lấy kết quả dựa vào giá trị của biến TT: TT=True - Hai Robot gặp nhau, TT=False - Hai Robot không gặp nhau.

Trong trường hợp gặp nhau thì dựa vào việc đã lưu thông tin ô gặp nhau vào 2 biến hm ,cm (hm - chỉ số hàng, cm - chỉ số cột) ta sẽ truy xuất đường đi của 2 Robot.

CÁC PHƯƠNG PHÁP TÌM KIẾM TRÊN ĐỒ THỊ

Lê Thị Tuyết Vân-Tổ Tin trường THPH chuyên Quốc Học, Huế

Một bài toán quan trọng trong lí thuyết đồ thị là bài toán duyệt tất cả các đỉnh có thể đến được từ một đỉnh xuất phát nào đó. Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là không được bỏ sót hay lặp lại bất kì đỉnh nào. Chính vì vậy mà ta phải xây dựng những thuật toán cho phép duyệt một cách hệ thống các đỉnh, những thuật toán như vậy gọi là những thuật toán tìm kiếm trên đồ thị (graph traversal). Ta quan tâm đến hai thuật toán cơ bản nhất: thuật toán tìm kiếm theo chiều sâu và thuật toán tìm kiếm theo chiều rộng.

1. Thuật toán tìm kiếm theo chiều sâu :

a. Thuật toán tìm kiếm theo chiều sâu:

Ý tưởng:

Tư tưởng của thuật toán tìm kiếm theo chiều sâu (Depth-First Search - DFS) có thể trình bày như sau: Trước hết, dĩ nhiên đỉnh s đến được từ s, tiếp theo, với mọi cung (s, x) của đồ thị thì x cũng sẽ đến được từ s. Với mỗi đỉnh x đó thì tất nhiên những đỉnh y nối từ x cũng đến được từ s...

Điều đó gợi ý cho ta viết một thủ tục đệ quy DFSVisit(u) mô tả việc duyệt từ đỉnh u bằng cách thăm đỉnh u và tiếp tục quá trình duyệt DFSVisit(v) với v là một đỉnh chưa thăm nối từ u .

Kĩ thuật đánh dấu được sử dụng để tránh việc liệt kê lặp các đỉnh: Khởi tạo avail[v]:=true, $\forall v \in V$, mỗi lần thăm một đỉnh, ta đánh dấu đỉnh đó lại (avail[v]:=false) để các bước duyệt đệ quy kế tiếp không duyệt lại đỉnh đó nữa.

Thuật toán:

```
procedure DFSVisit(u ∈ V); //Thuật toán tìm kiếm theo chiều sâu từ đỉnh u
begin
avail[u] := False; //avail[u] = False ⇔ u đã thăm
Output ← u; //Liệt kê u
for ∀v ∈ V:(u, v)∈ E do //Duyệt mọi đỉnh v chưa thăm nối từ u
     if avail[v] then DFSVisit(v);
end;
begin //Chương trình chính
Input → Đồ thị G
```

```
for ∀v ∈ V do avail[v] := True; //Đánh dấu mọi đỉnh đều chưa thăm
DFSVisit(s);
end.
```

b. Thuật toán tìm đường đi theo DFS:

Bài toán tìm đường đi:

Cho đồ thị G=(V,E) và hai đỉnh s, t ∈ V.

Nhắc lại định nghĩa đường đi: Một dãy các đỉnh:

$$P=<s=p_0, p_1, ..., p_k=t> (\forall i: (p_{i-1}, p_i) \in E)$$

được gọi là một đường đi từ s tới t, đường đi này gồm k+1 đỉnh $p_0, p_1, ..., p_k$ và cạnh $(p_0, p_1), (p_1, p_2), ...,(p_{k-1}, p_k)$. Đỉnh s được gọi là đỉnh đầu và đỉnh t được gọi là đỉnh cuối của đường đi. Nếu tồn tại một đường đi từ s tới t, ta nói s đến được t và t đến được từ s: s t.

Thuật toán:

Để lưu lại đường đi từ đỉnh xuất phát s, trong thủ tục DFSVisit(u), trước khi gọi đệ quy DFSVisit(v) với v là một đỉnh chưa thăm nối từ u chưa đánh dấu), ta lưu lại vết đường đi từ u tới v bằng cách đặt trace[v]:=u, tức là trace[v] lưu lại đỉnh liền trước v trong đường đi từ s tới v . Khi thuật toán DFS kết thúc, đường đi từ s tới t sẽ là: $<p_1=t \leftarrow p_2=trace[p_1] \leftarrow p_3=trace[p_2] \leftarrow ... \leftarrow s>$

```
procedure DFSVisit(u∈V); //Thuật toán tìm kiếm theo chiều sâu từ đỉnh u
begin
avail[u] := False; //avail[u] = False ⇔ u đã thăm
for ∀ v∈ V:(u, v)∈ E do //Duyệt mọi đỉnh v chưa thăm nối từ u
   if avail[v] then
      begin
         trace[v] := u; //Lưu vết đường đi, đỉnh liền trước v trên đường đi từ s tới v là u
         DFSVisit(v); //Gọi đệ quy để tìm kiếm theo chiều sâu từ đỉnh v
      end;
end;
begin //Chương trình chính
Input → Đồ thị G, đỉnh xuất phát s, đỉnh đích t;
for ∀v ∈ V do avail[v] := True; //Đánh dấu mọi đỉnh đều chưa thăm
DFSVisit(s);
if avail[t] then //s đi đến được t
   «Truy theo vết từ t để tìm đường đi từ s tới t»;
```

```
end.
```

Có thể không cần mảng đánh dấu avail[1 … n] mà dùng luôn mảng trace[1 … n] để đánh dấu: Khởi tạo các phần tử mảng trace[1 … n] là:

$$\begin{cases} Trace[s] \neq 0 \\ Trace[v] = 0, \forall v \neq s \end{cases}$$

Khi đó điều kiện để một đỉnh v chưa thăm là trace[v] = 0, mỗi khi từ đỉnh u thăm đỉnh v, phép gán trace[v]= u sẽ kiêm luôn công việc đánh dấu v đã thăm (trace[v] ≠0).

Tính chất của BFS

Nếu ta sắp xếp danh sách kề của mỗi đỉnh theo thứ tự tăng dần thì thuật toán DFS luôn trả về đường đi có thứ tự từ điển nhỏ nhất trong số tất cả các đường đi từ s tới tới t.

c. Thuật toán duyệt đồ thi theo DFS

Cài đặt trên chỉ là một ứng dụng của thuật toán DFS để liệt kê các đỉnh đến được từ một đỉnh. Thuật toán DFS dùng để duyệt qua các đỉnh và các cạnh của đồ thị được viết như sau:

```
procedure DFSVisit(u∈V); //Thuật toán tìm kiếm theo chiều sâu từ đỉnh u
begin
Time := Time + 1;
d[u] := Time; //Thời điểm duyệt đến u
Output ← u; //Liệt kê u
for □v∈V:(u, v) ∈E do //Duyệt mọi đỉnh v nối từ u
      if d[v] = 0 then DFSVisit(v); //Nếu v chưa thăm, gọi đệ quy để tìm
                                    // kiếm theo chiều sâu từ đỉnh v
Time := Time + 1;
  f[u] := Time; //Thời điểm duyệt xong u
end;
begin //Chương trình chính
Input → Đồ thị G
for □v∈V do d[v] := 0; //Mọi đỉnh đều chưa được duyệt đến
Time := 0;
for □v∈V do
   if d[v] = 0 then DFSVisit(v);
end.
```

Thời gian thực hiện giải thuật của DFS có thể đánh giá bằng số lần gọi thủ tục DFSVisit (|V| lần) cộng với số lần thực hiện của vòng lặp for bên trong thủ tục DFSVisit. Chính vì vậy:

- Nếu đồ thị được biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, vòng lặp for bên trong thủ tục DFSVisit (xét tổng thể cả chương trình) sẽ duyệt

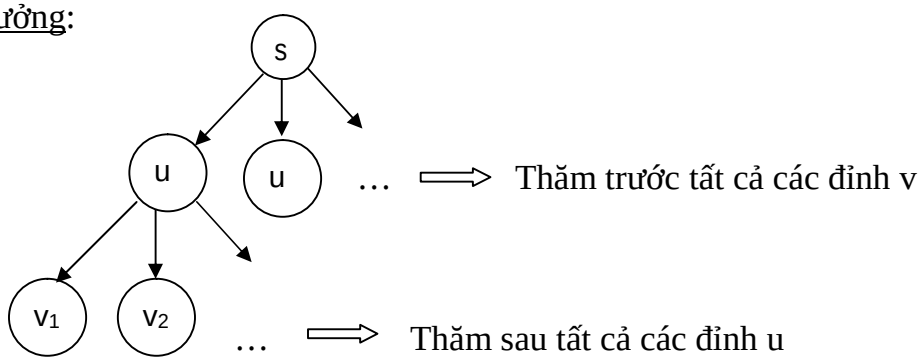
qua tất cả các cạnh của đồ thị (mỗi cạnh hai lần nếu là đồ thị vô hướng, mỗi cạnh một lần nếu là đồ thị có hướng). Trong trường hợp này, thời gian thực hiện giải thuật DFS là $\Theta(|V|+|E|)$

- Nếu đồ thị được biểu diễn bằng ma trận kề, vòng lặp for bên trong mỗi thủ tục DFSVisit sẽ phải duyệt qua tất cả các đỉnh 1 … n. Trong trường hợp này thời gian thực hiện giải thuật DFS là $\Theta(|V|+|V|^2) = \Theta(|V|^2)$.
- Nếu đồ thị được biểu diễn bằng danh sách cạnh, vòng lặp for bên trong thủ tục DFSVisit sẽ phải duyệt qua tất cả danh sách cạnh mỗi lần thực hiện thủ tục. Trong trường hợp này thời gian thực hiện giải thuật DFS là $\Theta(|V||E|)$.

2. Thuật toán tìm kiếm theo chiều rộng:

a. Thuật toán tìm kiếm theo chiều rộng

Ý tưởng:



Tư tưởng của thuật toán tìm kiếm theo chiều rộng (Breadth-First Search – BFS) là "lập lịch" duyệt các đỉnh. Việc thăm một đỉnh sẽ lên lịch duyệt các đỉnh nối từ nó sao cho thứ tự duyệt là ưu tiên chiều rộng (đỉnh nào gần đỉnh xuất phát s hơn sẽ được duyệt trước). Đầu tiên ta thăm đỉnh s. Việc thăm đỉnh s sẽ phát sinh thứ tự thăm những đỉnh $u_1, u_2, \ldots$ nối từ s (những đỉnh gần s nhất). Tiếp theo ta thăm đỉnh u_1, khi thăm đỉnh u_1 sẽ lại phát sinh yêu cầu thăm những đỉnh $r_1, r_2, \ldots$ nối từ u_1. Nhưng rõ ràng các đỉnh r này "xa" s hơn những đỉnh u nên chúng chỉ được thăm khi tất cả những đỉnh u đã thăm. Tức là thứ tự duyệt đỉnh sẽ là: $s, u_1, u_2, \ldots, r_1, r_2, \ldots$

Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách để chứa những đỉnh đang "chờ" thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách, loại nó ra khỏi danh sách và cho những đỉnh chưa "xếp hàng" kề với nó xếp hàng thêm vào cuối danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng.

Vì nguyên tắc vào trước ra trước, danh sách chứa những đỉnh đang chờ thăm được tổ chức dưới dạng hàng đợi (Queue): Nếu ta có Queue là một hàng đợi với thủ tục Push(r) để đẩy một đỉnh r vào hàng đợi và hàm Pop trả về một đỉnh lấy ra từ hàng đợi thì thuật toán BFS có thể viết như sau:

Thuật toán:

```
Queue := (s); //Khởi tạo hàng đợi chỉ gồm một đỉnh s
for ∀v∈V do avail[v] := True;
```

```
avail[s] := False; //Đánh dấu chỉ có đỉnh s được xếp hàng
repeat //Lặp tới khi hàng đợi rỗng
u := Pop; //Lấy từ hàng đợi ra một đỉnh u
Output ← u; //Liệt kê u
   for ∀v∈V:avail[v] and (u, v) ∈ E do //Xét những đỉnh v kề u chưa được
                                       //đẩy vào hàng đợi
   begin
      Push(v); //Đẩy v vào hàng đợi
      avail[v] := False; //Đánh dấu v đã xếp hàng
   end;
until Queue = Ø;
```

2. Thuật toán tìm đường đi theo BFS:

```
Queue := (s); //Khởi tạo hàng đợi chỉ gồm một đỉnh s
for ∀v∈V do avail[v] := True;
avail[s] := False; //Đánh dấu chỉ có đỉnh s được xếp hàng
repeat //Lặp tới khi hàng đợi rỗng
u := Pop; //Lấy từ hàng đợi ra một đỉnh u
   for ∀v∈V:avail[v] and (u, v) ∈ E do //Xét những đỉnh v kề u chưa được
                                       //đẩy vào hàng đợi
   begin
      trace[v] := u; //Lưu vết đường đi
      Push(v); //Đẩy v vào hàng đợi
      avail[v] := False; //Đánh dấu v đã xếp hàng
   end;
until Queue = Ø;

if avail[t] then //s đi tới được t
«Truy theo vết từ t để tìm đường đi từ s tới t»;
```

Tương tự như thuật toán tìm kiếm theo chiều sâu, ta có thể dùng mảng Trace[1…n] kiêm luôn chức năng đánh dấu.

Tính chất của BFS

Thuật toán BFS luôn trả về đường đi qua ít cạnh nhất trong số tất cả các đường đi từ s tới t. Nếu ta sắp xếp các danh sách kề của mỗi đỉnh theo thứ tự tăng dần và nếu có nhiều đường đi từ s tới t đều qua ít cạnh nhất thì thuật toán BFS sẽ trả về đường đi có thứ tự từ điển nhỏ nhất trong số những đường đi đó.

c. Thuật toán duyệt đồ thi theo BFS

Tương tự như thuật toán DFS, trên thực tế, thuật toán BFS cũng dùng để xác định một thứ tự trên các đỉnh của đồ thị và được viết theo mô hình sau:

```
procedure BFSVisit(s∈V);
begin
```

```
Queue := (s); //Khởi tạo hàng đợi chỉ gồm một đỉnh s
Time := Time + 1;
d[s] := Time; //Duyệt đến đỉnh s
repeat //Lặp tới khi hàng đợi rỗng
   u := Pop; //Lấy từ hàng đợi ra một đỉnh u
   Time := Time+1;
   F[u]:=Time; //Ghi nhận thời điểm duyệt xong đỉnh u
   Output ← u; //Liệt kê u
   for ∀v∈V:(u, v) ∈E do //Xét những đỉnh v kề u
      if d[v] = 0 then //Nếu v chưa duyệt đến
         begin
            Push(v); //Đẩy v vào hàng đợi
            Time := Time + 1;
            d[v] := Time; //Ghi nhận thời điểm duyệt đến đỉnh v
         end;
until Queue = Ø;
end;
begin //Chương trình chính
Input → Đồ thị G;
for ∀v∈V do d[v] := 0; //Mọi đỉnh đều chưa được duyệt đến
Time := 0;
for ∀v∈V do
   if d[v]=0 then BFSVisit(v);
end.
```

Thời gian thực hiện giải thuật của BFS tương tự như đối với DFS, bằng $\Theta(|V|+|E|)$ nếu đồ thị được biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, bằng $\Theta(|V|^2)$ nếu đồ thị được biểu diễn bằng ma trận kề, và bằng $\Theta(|V||E|)$ nếu đồ thị được biểu diễn bằng danh sách cạnh.

Bài tập:

Bài 1:

Mê cung hình chữ nhật kích thước m×n gồm các ô vuông đơn vị (m, n ≤ 1000). Trên mỗi ô ghi một trong ba kí tự:

- O: Nếu ô đó an toàn
- X: Nếu ô đó có cạm bẫy
- E: Nếu là ô có một nhà thám hiểm đang đứng.

Duy nhất chỉ có 1 ô ghi chữ E. Nhà thám hiểm có thể từ một ô đi sang một trong số các ô chung cạnh với ô đang đứng. Một cách đi thoát khỏi mê cung là

một hành trình đi qua các ô an toàn ra một ô biên. Hãy chỉ giúp cho nhà thám hiểm một hành trình thoát ra khỏi mê cung đi qua ít ô nhất.

Dữ liệu vào từ tệp văn bản MECUNG.INP

- Dòng 1: Ghi m, n (1<m, n≤1000).
- M dòng tiếp theo thể hiện bảng kích thước m×n, mô tả trạng thái của mê cung theo thứ tự từ trên xuống dưới, mỗi dòng n ký tự theo thứ tự từ trái qua phải.

Kết quả ghi ra file MECUNG.OUT

- Dòng 1: Ghi số bước đi tìm của hành trình tìm được.
- Dòng 2: Ghi một xâu ký tự S mô tả hành trình tìm được (xâu ký tự S chỉ gồm các chữ cái in hoa E, W, S, N mà mỗi ký tự trong xâu S thể hiện việc đi sang ô chung cạnh theo hướng được mô tả bởi ký tự đó. Ví dụ: E: đi sang ô chung cạnh theo hướng Đông, W: đi sang ô chung cạnh theo hướng Tây, S: đi sang ô chung cạnh theo hướng Nam, N: đi sang ô chung cạnh theo hướng Bắc)

Ví dụ:

MECUNG.INP
4 5 XXXOX XOOOX XEXOO XXXOO

MECUNG.OUT
4 NEEN

Chương trình

```
{$MODE OBJFPC}
Const NMax = 1000;
      Fi  = 'MECUNG.INP';
      Fo  = 'MECUNG.OUT';
      dd: Array[1..4] of integer = ( 0,-1, 0, 1);
      dc: Array[1..4] of integer = (-1, 0, 1, 0);
      h: array[1..4] of char=('W','N', 'E', 'S');
Var   a, tr: Array[1..NMax,1..NMax] of integer;
      queue : Array[1..NMax*NMax] of Record
                                       d,c : integer;
                                     End;
      N, M, dau, cuoi, x0, y0, x1, y1: integer;
      ok: boolean;

Procedure DocF;
Var i,j : integer;
    s: string;
Begin
```

```
  Assign(Input,Fi);
  Reset(Input);
  Readln(M,N);
  For i:=1 to M do
    begin
      readln(s);
      For j:=1 to N do
        case s[j] of
          'O': a[i,j]:=0;
          'X': a[i,j]:=1;
          'E': begin
                 a[i,j]:=1;
                 x0:=i;
                 y0:=j;
               end;
        end;
    end;
  Close(Input);
End;

Procedure BFS(i,j : integer);
Var k,dong,cot,u,v : integer;
Begin
  ok:=false;
  if (x0=1) or (x0=m) or (y0=1) or (y0=n) then
    begin
      x1:=x0;
      y1:=y0;
      ok:=true;
      exit;
    end;
  Dau:=1;
  Cuoi:=1;
  queue[cuoi].d := i;
  queue[cuoi].c := j;
  tr[i,j] := 1;
  While dau<=cuoi do
    Begin
      dong := queue[dau].d;
      cot  := queue[dau].c;
      inc(dau);
      For k:=1 to 4 do
        Begin
```

```
          u := dong + Dd[k];
          v := cot  + Dc[k];
          If (u>0) and (u<=M) and (v>0) and (v<=N) then
            If (a[u, v]=0) and (tr[u,v]=0) then
              Begin
                Inc(cuoi);
                queue[cuoi].d := u;
                queue[cuoi].c := v;
                tr[u,v]    := k;
                if (u=1) or (u=m) or (v=1) or (v=n) then
                  begin
                    x1:=u;
                    y1:=v;
                    ok:=true;
                    exit;
                  end;
              End;
        End;
    End;
End;

Procedure Inkq;
Var i, x, y: integer;
    s:string;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  if not ok then writeln(-1)
    else
      begin
        s:='';
        while (x1<>x0) or (y1<>y0) do
          begin
            s:=h[tr[x1,y1]]+s;
            x:=x1;
            y:=y1;
            x1:=x-dd[tr[x,y]];
            y1:=y-dc[tr[x,y]];
          end;
        writeln(length(s));
        writeln(s);
      end;
  Close(Output);
```

```
End;

BEGIN
  DocF;
  BFS(x0,y0);
  Inkq;
END.
```

Bài 2:

Trên bàn cờ m×n (1 ≤ m, n ≤ 1000) ô vuông có k quân mã đang đứng ở những ô nào đó (1 ≤ k ≤ 1000). Trong quá trình di chuyển, quân mã có thể nhảy đến ô đã có những quân mã khác đang đứng. Hãy tìm cách di chuyển k quân mã đến vị trí ô [x0, y0] cho trước sao cho tổng bước đi của các quân mã là nhỏ nhất.

Dữ liệu vào tệp văn bản HORSES.INP:

- Dòng 1 chứa 5 số nguyên dương m , n, x0, x0, k .
- k dòng tiếp theo mỗi dòng ghi 2 số nguyên là tọa độ của một quân mã.

Kết quả ghi vào tệp văn bản HORSES.OUT:

- Ghi một số duy nhất là tổng số bước đi của các quân mã. Trong trường hợp không di chuyển được một quân mã nào đó về vị trí [x0, y0] thì ghi -1.

Ví dụ:

HORSES.INP
8 8 8 8 3 1 1 2 2 3 3

HORSE.OUT
14

Phân tích:

Loang từ điểm (x0, y0) ra hết bảng.

Trong bảng len[1..n, 1..n], tại mỗi ô ghi số bước đi của quân mã di chuyển từ ô [x0, y0] đến ô đó.

Nếu tại ô có quân mã không có giá trị thì không có cách di chuyển quân mã đó đến ô [x0, y0] ghi -1, ngược lại ta tính tổng số của các số ghi trong các ô có quân mã đang đứng, tổng số đó là đáp số bài toán.

Chương trình

```
{$MODE OBJFPC}
Const NMax = 1000;
      Fi  = 'HORSES.INP';
      Fo  = 'HORSES.OUT';
      dd: Array[1..8] of integer = (-1,-2,-2,-1, 1, 2, 2, 1);
```

```
      dc: Array[1..8] of integer = (-2,-1, 1, 2, 2, 1,-1,-2);
Var   len: Array[1..NMax,1..NMax] of integer;
      queue : Array[1..NMax*NMax] of Record
                                      d,c : integer;
                                    End;
      N, M, x0, y0, q, dau, cuoi: integer;
      x, y: array[1..NMax] of integer;

Procedure ReadFile;
Var i, j : integer;
Begin
  Assign(Input,Fi);
  Reset(Input);
  Readln(M,N, x0, y0, q);
  For i:=1 to q do readln(x[i],y[i]);
  Close(Input);
End;

Procedure BFS(i,j : integer);
Var k,dong,cot,u,v,t : integer;
Begin
  Dau:=1;
  Cuoi:=1;
  queue[cuoi].d:=i;
  queue[cuoi].c:=j;
  fillchar(len, sizeof(len),0)
  len[i,j] := 1;
  While dau<=cuoi do
    Begin
      dong := queue[dau].d;
      cot  := queue[dau].c;
      inc(dau);
      For k:=1 to 8 do
        Begin
          u := dong + Dd[k];
          v := cot  + Dc[k];
          If (u>0) and (u<=M) and (v>0) and (v<=N) then
            If len[u,v]=0 then
              Begin
                Inc(cuoi);
                queue[cuoi].d := u;
                queue[cuoi].c := v;
                len[u,v] := len[dong,cot]+1;
```

```
                End;
            End;
        End;
End;

Procedure PrintResult;
Var i, s: integer;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  s:=0;
  for i:=1 to q do
    if len[x[i],y[i]]=0 then
      begin
        s:=-1;
        break;
      end
      else s:=s+len[x[i],y[i]]-1;
  writeln(s);
  close(Output);
End;

BEGIN
  ReadFile;
  BFS(x0,y0);
  PrintResult;
END.
```

Bài 3:

Cho một đồ thị vô hướng có N đỉnh được đánh số từ 1 đến N. Hãy tìm các vùng liên thông của đồ thị.

Dữ liệu vào từ file văn bản SVLT.INP

- Dòng 1: Ghi n, m lần lượt là số đỉnh và số cạnh của đồ thị (1< n≤100)
- M dòng tiếp theo: mỗi dòng ghi hai đỉnh đầu của một cạnh.

Kết quả ghi ra file SVLT.OUT

- Dòng 1: Ghi số K là số vùng liên thông.
- K dòng tiếp theo: mỗi dòng ghi các đỉnh thuộc cùng 1 vùng liên thông.

Ví dụ :

SVLT.INP
11 10 1 2 3 4

SVLT.OUT
4 1 2 3 4 5 6 7 8

```
3 6
4 5
4 6
5 7
6 7
6 8
7 8
10 11
```

```
9
10 11
```

```
Const Max = 100;
      Fi  = 'SVLT.INP';
      Fo  = 'SVLT.OUT';
Var   A: Array[1..Max,1..Max] of boolean;
      D: Array[1..Max] of integer;
      queue   : Array[1..Max*Max] of Integer;
      N, dau, cuoi, sv: integer;

Procedure ReadFile;
Var i, u, v, m : integer;
Begin
  Assign(Input,Fi);
  Reset(Input);
  Readln(N, m);
  fillchar(a, sizeof(a), false);
  For i:=1 to M do
    begin
      Read(u,v);
      a[u, v]:=true;
      a[v, u]:=true;
    end;
  Close(Input);
End;

Procedure BFS(u : integer);
Var v : integer;
Begin
  Dau:=1;
  Cuoi:=1;
  queue[cuoi] := u;
  D[u] := sv;
  While dau<=cuoi do
    Begin
      u := queue[dau];
```

```
      inc(dau);
      For v:=1 to n do
        If A[u,v] and (D[v]=0) then
              Begin
                Inc(cuoi);
                queue[cuoi] := v;
                D[v]    := sv;
              End;
    End;
End;

Procedure Timsvlt;
var i: integer;
Begin
  Sv := 0;
  fillchar(D, sizeof(d), 0);
  Fillchar(D,sizeof(D),0);
  for i:=1 to n do
    if D[i]=0 then
        begin
          inc(sv);
          BFS(i);
        end;
End;

Procedure Inkq;
Var i, j: integer;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  writeln(sv);
  For i:=1 to sv do
    Begin
      For j:=1 to N do
          If D[j]=i then Write(j,' ');
      Writeln;
    end;
  Close(Output);
End;

BEGIN
  ReadFile;
  Timsvlt;
```

```
  Inkq;
END.
```

Bài 4:

Cho bảng hình chữ nhật chia thành m×n ô vuông đơn vị, mỗi ô vuông có ghi số 0 hoặc 1. Một miền 0 của bảng là tập hợp các ô chung đỉnh chứa số 0. Hãy tính số miền 0 của bảng và diện tích của từng miền 0.

Dữ liệu vào từ file văn bản MIEN0.INP

- Dòng 1: Ghi m, n (1<m, n≤100).
- M dòng tiếp theo thể hiện bảng số theo thứ tự từ trên xuống dưới, mỗi dòng n số theo thứ tự từ trái qua phải.

Kết quả ghi ra file MIEN0.OUT

- Dòng 1: Ghi số lượng miền 0.
- Dòng 2: ghi diện tích của các miền 0

Ví du :

MIEN0.INP
8 10
0 1 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 0 1 0
0 0 0 1 1 0 0 0 1 0
1 1 1 0 1 1 0 0 1 0
0 0 1 1 0 0 0 0 1 0
0 0 0 1 1 1 1 1 1 0
1 1 0 1 0 0 0 1 0 1
0 0 0 1 0 0 1 0 1 0

MIEN0.OUT
4
1 25 14 9

```
Const Max = 100;
      Fi  = 'MIEN0.INP';
      Fo  = 'MIEN0.OUT';
      dc: Array[1..8] of integer = ( 0, 1, 1, 1, 0,-1,-1,-1);
      dd: Array[1..8] of integer = (-1,-1, 0, 1, 1, 1, 0,-1);
Var   A, D: Array[1..Max,1..Max] of integer;
      QUEUE   : Array[1..Max*Max] of Record
                                       d,c : integer;
                                    End;
      DT  : Array[1..Max*Max] of Integer;
      N, M , dau, cuoi, sv : integer;

Procedure DocF;
Var i,j : integer;
Begin
  Assign(Input,Fi);
```

```
  Reset(Input);
  Readln(M,N);
  For i:=1 to M do
      For j:=1 to N do  Read(A[i,j]);
  Close(Input);
End;

Procedure BFS(i,j : integer);
Var k,dong,cot,u,v : integer;
Begin
  Dau:=1;
  Cuoi:=1;
  QUEUE[cuoi].d := i;
  QUEUE[cuoi].c := j;
  D[i,j] := sv;
  DT[SV]:=1;
  While dau<=cuoi do
    Begin
      dong := QUEUE[dau].d;
      cot  := QUEUE[dau].c;
      inc(dau);
      For k:=1 to 8 do
        Begin
          u := dong + Dd[k];
          v := cot  + Dc[k];
          If (u>0) and (u<=M) and (v>0) and (v<=N) then
            If (A[u,v]=0) and (D[u,v]=0) then
              Begin
                Inc(cuoi);
                QUEUE[cuoi].d := u;
                QUEUE[cuoi].c := v;
                D[u,v]    := sv;
                Inc(DT[sv]);
              End;
        End;
    End;
End;

Procedure Timsvlt;
var i, j: integer;
Begin
  Sv := 0;
  fillchar(D, sizeof(d), 0);
```

```
  fillchar(DT, sizeof(DT), 0);
  Fillchar(D,sizeof(D),0);
  for i:=1 to m do
    for j:=1 to n do
      if (a[i,j]=0) and (D[i,j]=0) then
        begin
          inc(sv);
          BFS(i,j);
        end;
End;

Procedure Inkq;
Var i: integer;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  writeln(sv);
  For i:=1 to sv do Write(DT[i],' ');
  Close(Output);
End;

BEGIN
  DocF;
  Timsvlt;
  Inkq;
END.
```

Bài 5:

Một lâu đài được chia thành m×n modul vuông (1<m, n<=50). Mỗi modul vuông có từ 0 đến 4 bức tường. Hãy viết chương trình tính :

1 - Lâu đài có bao nhiêu phòng ?

2 - Diện tích phòng lớn nhất là bao nhiêu ?

3 - Bức tường nào cần loại bỏ để phòng càng rộng càng tốt ?

Dữ liệu vào từ tệp văn bản LAUDAI.INP

Dòng 1: ghi số lượng các môdul theo hướng Bắc-Nam và số lượng các modul theo hướng Đông Tây.

Trong các dòng tiếp theo, mỗi modul được mô tả bởi 1 số (0 ≤p≤15). Số đó là tổng của : 1 (= tường phía Tây), 2 (=tường phía Bắc) ,4 (=tường phía Đông) , 8 (= tường phía Nam) .

Các bức tường ở bên trong được xác định hai lần ; bức tường phía Nam trong modul (1,1) đồng thời là bức tường phía Bắc trong modul (2,1)

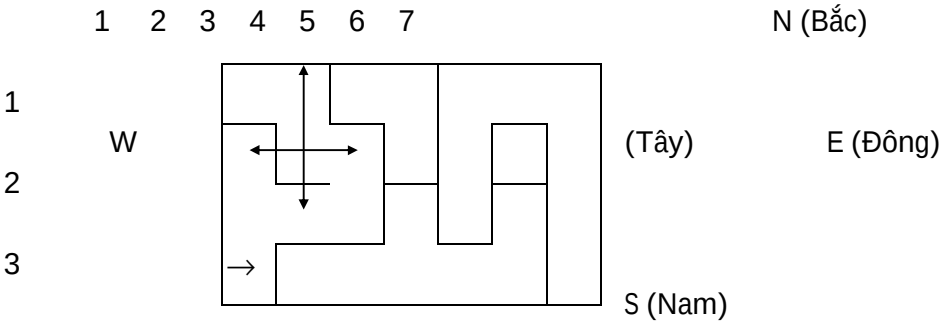
Kết quả ghi ra tệp văn bản LAUDAI.OUT

Dòng 1: ghi số lượng phòng.

Dòng 2: ghi diện tích của phòng lớn nhất (tính theo số modul)

Dòng 3: ghi bức tường cần loại bỏ (trước tiên là hàng sau đó là cột của modul có tường đó) và dòng cuối cùng là hướng của bức tường .

Ví dụ:



Mũi tên chỉ bức tường cần loại bỏ theo kết quả ở ví dụ

```
Const NMax = 50;
      Fi  = 'LAUDAI.INP';
      Fo  = 'LAUDAI.OUT';
      dd: Array[0..3] of integer = ( 0,-1, 0, 1);
      dc: Array[0..3] of integer = (-1, 0, 1, 0);
      h: array[0..3] of char=('W','N', 'E', 'S');
Var   A, D: Array[1..NMax,1..NMax] of integer;
      queue   : Array[1..NMax*NMax] of Record
                                    d,c : integer;
                                  End;
      DT  : Array[1..NMax*NMax] of Integer;
      N, M, dau, cuoi, sp, MaxDT, i0, j0, k0: integer;

Procedure DocF;
Var i,j : integer;
Begin
  Assign(Input,Fi);
  Reset(Input);
  Readln(M,N);
  For i:=1 to M do
      For j:=1 to N do  Read(A[i,j]);
  Close(Input);
End;

Procedure BFS(i,j : integer);
Var k,dong,cot,u,v : integer;
Begin
```

```
  Dau:=1;
  Cuoi:=1;
  queue[cuoi].d := i;
  queue[cuoi].c := j;
  D[i,j] := sp;
  DT[sp]:=1;
  While dau<=cuoi do
    Begin
      dong := queue[dau].d;
      cot  := queue[dau].c;
      inc(dau);
      For k:=0 to 3 do
        Begin
          u := dong + Dd[k];
          v := cot  + Dc[k];
          If (u>0) and (u<=M) and (v>0) and (v<=N) then
            If ((A[dong,cot] shr k) and 1 =0) and (D[u,v]=0)
              then
              Begin
                Inc(cuoi);
                queue[cuoi].d := u;
                queue[cuoi].c := v;
                D[u,v]    := sp;
                Inc(DT[sp]);
              End;
        End;
    End;
End;

procedure TimDtMax;
var i: integer;
begin
  MaxDT:=0;
  for i:=1 to sp do
    if MaxDT<DT[i] then MaxDT:=DT[i];
end;

procedure TimTuong;
var i, j, k, max, u, v: integer;
begin
  max:=0;
  for i:=1 to m-1 do
    for j:=1 to n-1 do
```

```
      for k:=2 to 3 do
        begin
          u:=i+dd[k];
          v:=j+dc[k];
          if ((a[i,j] shr k) and k =1) and (D[i,j]<>D[u,v])
            then
            if max < DT[D[i,j]]+DT[D[u,v]] then
              begin
                max:=DT[D[i,j]]+DT[D[u,v]];
                i0:=i;
                j0:=j;
                k0:=k;
              end;
        end;
end;

Procedure Timsvlt;
var i, j: integer;
Begin
  sp := 0;
  fillchar(DT, sizeof(DT), 0);
  Fillchar(D,sizeof(D),0);
  for i:=1 to m do
    for j:=1 to n do
      if D[i,j]=0 then
        begin
          inc(sp);
          BFS(i,j);
        end;
End;
Procedure Inkq;
Var i: integer;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  writeln(sp);
  Writeln(MaxDT);
  writeln(i0,' ', j0, ' ', h[k0]);
  Close(Output);
End;
BEGIN
  DocF;
  Timsvlt;
```

```
  TimDtMax;
  TimTuong;
  Inkq;
END.
```

Bài 6:

Cho một lưới hình chữ nhật kích thước m×n gồm các ô vuông đơn vị, mỗi ô được tô 1 trong 6 màu ký hiệu màu 1 , màu 2... màu 6. Giả thiết màu của 2 ô trái trên và phải dưới là khác nhau. Hai ô chung cạnh cùng thuộc một miền nếu cùng màu . Người A đứng ở miền có chứa ô góc trái trên, người B đứng ở miền có chứa ô phải dưới . Hai người chơi lần lượt, đến lượt mình người chơi có thể tô lại màu của miền mà mình đang đứng. Trò chơi kết thúc khi hai người đứng ở hai miền cạnh nhau (chung nhau ít nhất một cạnh của một ô vuông). Tính số lượt đi ít nhất để trò chơi đó kết thúc.

Giới hạn: 1 ≤ m, n ≤ 100. Số lượng miền ≤ 100.

Dữ liệu vào từ tệp văn bản DOIMAU.INP:

- Dòng đầu: ghi hai số m , n.
- M dòng tiếp theo, số thứ j của dòng j ghi số hiệu màu của ô [i, j].

Kết quả ghi ra tệp văn bản DOIMAU.OUT: ghi 1 số duy nhất là số lượt đi ít nhất để trò chơi kết thúc.

Ví du:

DOIMAU.INP
4 3 1 2 2 2 2 1 1 4 3 1 3 2

DOIMAU.OUT
3

Phân tích:

+ Loang từ ô [1, 1] để tìm số miền (sm) .

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

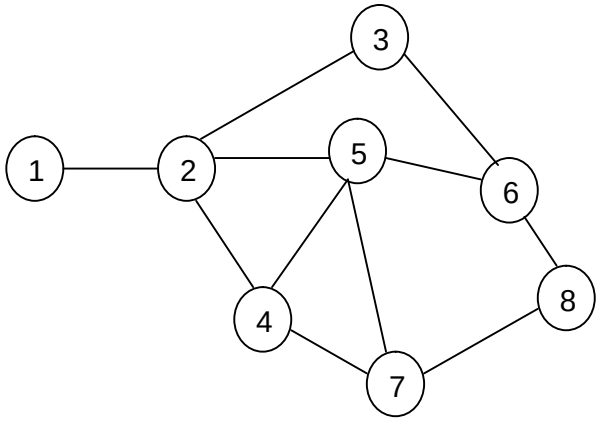
+ Xây dựng véc tơ V màu của từng miền

V=

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

+ Xây dựng đồ thị gồm sm đỉnh, xem một miền là một đỉnh của đồ thị. Giữa hai đỉnh có cạnh nối nếu hai miền đó có chung nhau ít nhất một cạnh của một ô vuông.

+ Tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh sm



Trong thủ tục BFS, tại mỗi bước, ta thăm một đỉnh đầu danh sách (giả sử đỉnh đó là đỉnh u), loại nó ra khỏi danh sách và cho những đỉnh v, chưa "xếp hàng" kề với u xếp hàng thêm vào cuối danh sách, tô màu đỉnh v giống màu đỉnh u, đồng thời cho các đỉnh kề với đỉnh v có màu giống với đỉnh u, chưa xếp "xếp hàng" thêm vào cuối danh sách.

```
{$MODE OBJFPC}
Const Max = 100;
      Fi  = 'DOIMAU.INP';
      Fo  = 'DOIMAU.OUT';
      dd: Array[1..4] of integer = ( 0,-1, 0, 1);
      dc: Array[1..4] of integer = (-1, 0, 1, 0);
Var   A, B, D: Array[1..Max,1..Max] of integer;
      Queue  : Array[1..Max*Max] of record
                                      d, c: integer;
                                    end;
      len: array[1..max] of integer;
      mau: array[1..max] of integer;
      N, M, sv : integer;

Procedure DocF;
Var i,j : integer;
Begin
  Assign(Input,Fi);
  Reset(Input);
  Readln(M,N);
  For i:=1 to M do
      For j:=1 to N do  Read(A[i,j]);
  fillchar(b, sizeof(b),0);
  fillchar(mau, sizeof(mau),0);
  Close(Input);
End;
```

```
Procedure BFS(i,j : integer);
Var k,dong,cot,u,v, dau, cuoi: integer;
    Queue  : Array[1..Max*Max] of record
                                          d, c: integer;
                                        end;
Begin
  Dau:=1;
  Cuoi:=1;
  Queue[cuoi].d := i;
  Queue[cuoi].c := j;
  D[i,j] := sv;
  mau[sv]:=a[i,j];
  While dau<=cuoi do
    Begin
      dong := Queue[dau].d;
      cot  := Queue[dau].c;
      inc(dau);
      For k:=1 to 4 do
        Begin
          u := dong + Dd[k];
          v := cot  + Dc[k];
          If (u>0) and (u<=M) and (v>0) and (v<=N) then
            begin
              If (a[u,v]=a[i,j])and(D[u,v]=0) then
                Begin
                  Inc(cuoi);
                  Queue[cuoi].d := u;
                  Queue[cuoi].c := v;
                  D[u,v]    := sv;
                End;
              if (a[u,v]<>a[i,j]) and (D[u,v]<>0) then
                begin
                  b[d[u,v],sv]:=1;
                  b[sv,d[u,v]]:=1;
                end;
            end;
        End;
    End;
End;


Procedure Timsvlt;
var i, j: integer;
Begin
  Sv := 0;
  fillchar(D, sizeof(d), 0);
  for i:=1 to m do
    for j:=1 to n do
      if D[i,j]=0 then
        begin
          inc(sv);
          BFS(i,j);
        end;
```

```
End;

procedure BFS1;
Var k, u, v, dau, cuoi : integer;
    queue: array[1..max] of integer;
Begin
  Dau:=1;
  Cuoi:=1;
  Queue[cuoi]:=1;
  len[1]:=1;
  While dau<=cuoi do
    Begin
      u:=queue[dau];
      inc(dau);
      For v:=1 to sv do
        if (b[u,v]=1) and (len[v]=0) then
           Begin
             Inc(cuoi);
             Queue[cuoi]:=v;
             len[v]:=len[u]+1;
             mau[v]:=mau[u];
             for k:=1 to sv do
                 if (b[v,k]=1)and(mau[k]=mau[u])and(len[k]=0) then
                   begin
                     inc(cuoi);
                     queue[cuoi]:=k;
                     len[k]:=len[v];
                   end;
          End;
    End;
End;

Procedure Inkq;
Var i, j: integer;
Begin
  Assign(OutPut,fo);
  Rewrite(OutPut);
  {writeln(sv);
  For i:=1 to m do
    begin
      for j:=1 to n do Write(D[i,j],' ');
      writeln;
    end;}
  write(len[sv]-1);
  Close(Output);
End;

BEGIN
  DocF;
  Timsvlt;
  BFS1;
  Inkq;
END.
```

Trên đây là một số bài tập tôi thu thập được để dạy cho học sinh trong phần các phương pháp tìm kiếm trên đồ thị. Vì thời gian chuẩn bị quá ngắn nên không tránh khỏi những sai sót, rất mong nhận được những đóng góp chân tình của các Thầy Cô, tôi xin chân thành cảm ơn.

TÀI LIỆU THAM KHẢO:

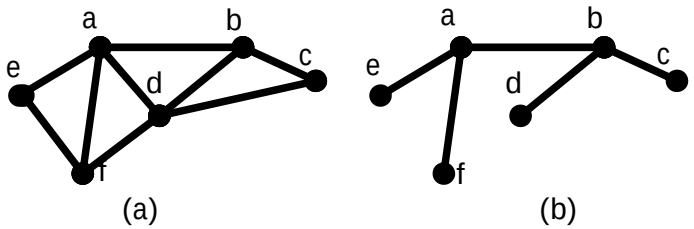
1. Tài liệu giáo khoa chuyên Tin quyển 1.
2. Chuyên đề bồi dưỡng học sinh giỏi Tin Học Trung học phổ thông Ứng dụng lý thuyết đồ thị (tác giả Hồ Sĩ Đàm – Trần Đỗ Hùng)

CÂY KHUNG VÀ CÂY KHUNG NHỎ NHẤT

Tống Thanh Kiều
THPT Chuyên Vĩnh Phúc

1. Đặt vấn đề

Hệ thống đường giao thông của một thành phố được biểu thị như một đơn đồ thị cho bởi hình 1a. Các duy nhất để những con đường có thể đi lại được vào mùa đông là phải cào tuyết thường xuyên. Chính quyền địa phương muốn cáo tuyết một số ít nhất các con đường sao cho luôn có đường thông suốt nối hai thành phố bất kỳ. Có thể làm điều này bằng cách nào?



Hình 1. a) Hệ thống đường và b) tập các con đường cần phải cào tuyết

Cần phải cào tuyết ít nhất năm con đường mới đảm bảo có đường đi giữa hai thành phố bất kỳ. Hình 1b biểu thị một tập hợp các con đường như vậy. Ta nhận thấy đồ thị con biểu diễn các con đường này là một cây vì nó liên thông và chứa sáu đỉnh, năm cạnh.

Bài toán trên được giải bằng một đồ thị con có một số tối thiểu các cạnh và chứa tất cả các đỉnh của đồ thị xuất phát. Đồ thị như thế phải là một cây.

2. Cây khung

2.1. Định nghĩa

Cho G là một đơn đồ thị. Một cây được gọi là cây khung của G nếu nó là một đồ thị con của G và chứa tất cả các đỉnh của G.

Một đơn đồ thị có cây khung sẽ là một đồ thị liên thông vì có đường đi trong cây khung giữa hai đỉnh bất kỳ. Điều ngược lại cũng đúng, tức là mọi đồ thị liên thông đều có cây khung.

2.2. Định lí

Một đơn đồ thị là liên thông nếu và chỉ nếu nó có cây khung

Chứng minh: Trước tiên, giả sử đồ thị G có cây khung T. T chứa tất cả các đỉnh của G. Hơn nữa, có đường đi trong T giữa hai đỉnh bất kỳ. Vì T là đồ thị con của G nên có đường đi trong G giữa hai đỉnh của nó. Do đó G là liên thông.

Bây giờ, giả sử G là liên thông. Nếu G không phải là một cây thì nó phải có chu trình đơn. Xóa đi một cạnh của một trong các chu trình đơn này. Đồ thị nhận được một số ít cạnh hơn nhưng vẫn còn chứa tất cả các đỉnh của G và vẫn liên thông. Nếu đồ thị con này không là cây thì nó còn chứa chu trình đơn. Cũng giống như trên, ta lại xóa đi một cạnh của chu trình đơn. Lặp lại quá trình này cho đến khi không còn chu trình đơn. Điều này là có thể vì chỉ có một số hữu hạn các cạnh trong đồ thị. Quá trình kết thúc khi không còn chu trình đơn trong đồ thị nhận được. Cây được tạo ra vì đồ thị vẫn còn liên thông khi xóa đi các cạnh. Cây này là cây khung vì nó chứa tất cả các đỉnh của G.

2.3. Tìm kiếm ưu tiên theo chiều sâu

Cách chứng minh định lí 1 đưa ra một thuật toán tìm cây khung bằng cách xóa đi các cạnh khỏi các chu trình đơn. Thuật toán này là không hiệu quả vì nó đòi hỏi phải nhận biết được các chu trình đơn. Thay cho việc xây dựng cây khung bằng cách loại bỏ các cạnh, cây khung có thể được xây dựng bằng cách lần lượt ghép thêm các cạnh.

Ta sẽ xây dựng cây khung của một đồ thì liên thông bằng phương pháp tìm kiếm ưu tiên theo chiều sâu. Nghĩa là sẽ tạo một cây có gốc và cây khung sẽ là đồ thị vô hướng nền của cây có gốc này. Chọn tùy ý một đỉnh của đồ thị làm gốc. Xây dựng đường đi từ đỉnh này bằng cách lần lượt ghép thêm các cạnh vào sao cho mỗi mạnh mới ghép sẽ nối đỉnh cuối cùng trên đường đi với một đỉnh còn chưa thuộc đường đi. Tiếp tục ghép thêm cạnh vào đường đi chừng nào không thể thêm được nữa thì thôi. Nếu đường đi qua tất cả các đỉnh của đồ thị thì cây do đường đi này tạo nên sẽ là cây khung. Nhưng nếu đường đi không đi qua tất cả các đỉnh thì cần thêm các cạnh khác vào đường đi. Lùi lại đỉnh trước đỉnh cuối cùng của đường đi và nếu có thể, xây dựng đường đi mới xuất phát từ đỉnh này qua các đỉnh còn chưa thuộc đường đi. Nếu điều đó không thể làm được thì lùi thêm một đỉnh nữa trên đường đi, tức là lùi lại hai đỉnh trên đường đi và thử xây dựng đường đi mới.

Lặp lại thủ tục này, bắt đầu từ đỉnh cuối cùng được ghé thăm lùi theo đường đi mỗi lần một đỉnh, xây dựng đường đi mới càng dài càng tốt cho tới khi nào không thể thêm được một cạnh nào nữa. Vì đồ thị có hữu hạn cạnh và là liên thông nên quá

trình đó sẽ kết thúc và tạo được cây khung. Mỗi đỉnh mà tại đó đường đi kết thúc ở mỗi giai đoạn của thuật toán sẽ là lá trong cây có gốc. Mỗi đỉnh tại đó đường đi bắt đầu từ đó sẽ là một đỉnh trong.

Tìm kiếm ưu tiên chiều sâu cũng được gọi là thủ tục quay lui vì nó quay lại đỉnh đã ghé thăm trước trên đường đi.

Các cạnh của đồ thị tìm được nhờ tím kiếm ưu tiên theo chiều sâu gọi là các cạnh của cây. Các cạnh khác của đồ thị có thể nối với đỉnh trước hoặc sau nó trong cây. Các cạnh này gọi là các cạnh quay lui.

2.4. Thuật toán tìm kiếm ưu tiên theo chiều sâu

Trong thuật toán này, chúng ta xây dựng cây khung của đồ thị G với các đỉnh $v_1, v_2, ..., v_n$ bằng cách lấy đỉnh v1 làm gốc của cây. Khởi tạo tập T là cây chỉ có một đỉnh này. Trong mỗi bước, thêm một đỉnh mới vào cây T cùng với cạnh đi ra từ đỉnh của T không chứa chu trình vì không có cạnh được thêm vào mà nó nối với đỉnh đã có trong cây. Tuy nhiên, T vẫn là liên thông như nó được xây dựng. Vì G là liên thông, mọi đỉnh trong G đều được thăm và được ghép vào cây. Từ đó suy ra T là cây khung của G

```
procedure DFS(G: đồ thị liên thông với các đỉnh v1, v2, ..., vn)
T:= cây chỉ chứa một đỉnh v1
visit(v1)

procedure visit(v: đỉnh của G)
for mỗi đỉnh w liền kề với v và chưa có trong T
Begin
   thêm đỉnh w và cạnh (v, w) vào T
   visit(w)
end
```

Phân tích độ phức tạp: Với mỗi đỉnh v thủ tục visit(v) được gọi khi đỉnh v lần đầu tiên được gặp trong tìm kiếm và không được gọi lại. Giả sử ta có danh sách kề của G, để tìm các đỉnh kề của v không cần phải tính toán gì cả. Theo từng bước của thuật toán chúng ta xem xét một cạnh nhiều nhất hai lần để quyết định xem có nên thêm cạnh này vào đỉnh cuối của nó hay không. Do vậy thủ tục DFS xây dựng cây khung dùng O(e) hay $O(n^2)$ các bước trong đó e và n tương ứng là số cạnh, số đỉnh của G.

2.5. Tìm kiếm ưu tiên chiều rộng

Có thể xây dựng cây khung của một đơn đồ thị bằng thuật toán tìm kiếm ưu tiên theo chiều rộng. Một lần nữa, cây có gốc sẽ được xây dựng và đồ thị vô hướng nền của cây có gốc sẽ tạo nên cây khung.

Chọn một đỉnh bất kỳ của đồ thị làm gốc. Sau đó ghép vào tất cả các cạnh liên thuộc với đỉnh này. Các đỉnh mới ghép vào trong giai đoạn này trở thành các đỉnh ở mức 1 của cây khung. Sắp xếp chúng theo một thứ tự tùy ý.

Tiếp theo với mỗi đỉnh ở mức 1 được ghé thăm theo thứ tự được sắp xếp ở trên ta ghép tất cả các cạnh liên thuộc với nó vào cây mà không tạo ra chu trình. Sắp xếp các đỉnh con ở mỗi đỉnh ở mức 1 theo một trật tự nào đó. Quá trình này tạo ra đỉnh ở mức 2 của cây. Tiếp tục làm lại thủ tục này cho tới khi tất cả các đỉnh của đồ thị được ghép vào cây. Thủ tục này kết thúc vì chỉ có một số hữu hạn các cạnh của đồ thị. Cây khung được tạo ra vì xây dựng được cây chứa tất cả các đỉnh của đồ thị.

2.6. Thuật toán tìm kiếm ưu tiên theo chiều rộng

Trong thuật toán này chúng ta giả sử các đỉnh v1, v2, ..., vn của đồ thị liên thông G là được sắp xếp theo thứ tự nào đó. Chúng ta cũng dùng từ xử lí để mô tả thủ tục thêm đỉnh mới và cạnh mới vào cây kề với đỉnh hiện thời đang được xử lí để không tạo ra vòng lặp

procedure *BFS*(*G*: đồ thị liên thông với các đỉnh $v_1, v_2, ..., v_n$)

T:= cây chỉ chứa một đỉnh v1

L:= danh sách rỗng

Đặt *v1* vào danh sách *L* gồm các đỉnh không xử lí

While L khác rỗng

Begin

Xóa đỉnh đầu tiên, v1 khỏi *L*

For mỗi đỉnh kề *w* của *v*

If w chưa nằm trong *L* và không thuộc *T* **then**

Begin

Thêm đỉnh *w* vào cuối danh sách *L*

Thêm đỉnh *w* và cạnh {*v*, *w*} vào *T*

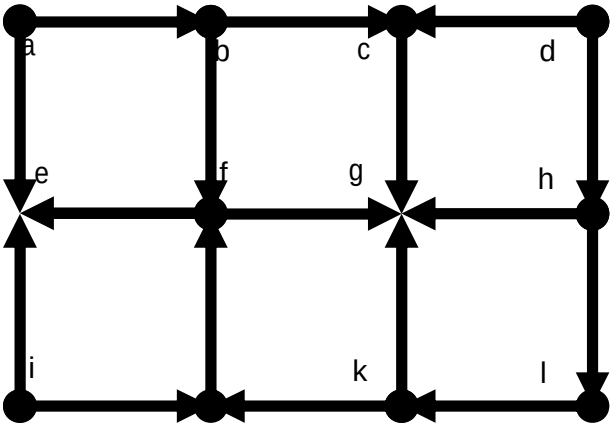
```
    end
end
```

Phân tích độ phức tạp: Với mỗi đỉnh v của đồ thị xem xét tất cả các đỉnh liền kề với v và thêm vào cây T mỗi đỉnh còn chưa được thăm. Giả sử có danh sách các đỉnh kề của đồ thị. Khi đó dễ dàng xác định xem đỉnh nào liền kề với đỉnh đã cho, xét mỗi cạnh nhiều nhất hai lần để xem có thêm cạnh này hay không và đỉnh cuối đã nằm trong cây hay chưa. Từ đó suy ra thuật toán tìm kiếm ưu tiên chiều rộng dùng O(e) hoặc $O(n^2)$ bước.

2.7. Tìm kiếm ưu tiên chiều rộng trong đồ thị có hướng

Chúng ta cũng có thể dễ dàng thay đổi cả tìm kiếm ưu tiên chiều sâu và tìm kiếm ưu tiên chiều rộng để chúng có thể chạy khi đầu vào là các đồ thị có hướng. Tuy nhiên, thông tin ra không nhất thiết là cây khung mà có lẽ là rừng khung. Trong cả hai thuật toán có thể thêm một cạnh chỉ khi mà nó có hướng đi ra từ đỉnh đang được thăm đi tới đỉnh chưa được thêm vào. Nếu ở giai đoạn của thuật toán mà không có cạnh bắt đầu từ đỉnh đã được thêm vào tới đỉnh chưa được thêm vào thì cạnh tiếp theo được đưa vào thuật toán sẽ trở thành gốc của một cây mới trong rừng khung.

Ví dụ: Cho đồ thị có hướng G. Hãy xác định rừng khung của nó bằng thuật toán tìm kiếm ưu tiên chiều sâu.



Giải: Chúng ta bắt đầu tìm kiếm ưu tiên chiều sâu tại đỉnh a và thêm các đỉnh b, c và g và các cạnh tương ứng, đến đây thì không đi tiếp được nữa. Khi đó cần quay lại c nhưng ở đó vẫn bị tắc và do vậy quay lui tới b. Tại đây chọn đỉnh f và e và các cạnh tương ứng. Việc quay lui tiếp đưa ta về a. Khi đó xây được cây mới tại d và thêm các đỉnh h, l, k, j và các cạnh tương ứng. Khi đó không đi tiếp được nữa, quay lui về k, sau đó l, rồi h, và về d. Cuối cùng chúng ta lại xây cây mới tại i và kết thúc tìm kiếm.

3. Cây khung nhỏ nhất

Một lớp rất rộng các bài toán có thể giải bằng cách tìm cây khung nhỏ nhất trong một đồ thị có trọng số sao cho tổng trọng số của các cạnh của cây là nhỏ nhất.

Định nhĩa: Cây khung nhỏ nhất trong một đồ thị liên thông có trọng số là một cây khung có tổng trọng số trên các cạnh của nó là nhỏ nhất.

2.9. Thuật toán tìm cây khung nhỏ nhất

Sau đây trình bày hai thuật toán tìm cây khung nhỏ nhất. Cả hai đều được tiến hành bằng cách ghép các cạnh có trọng số nhỏ nhất trong số các cạnh có một tính chất nào đó mà chưa được dùng. Những thuật toán này là những ví dụ về thuật toán tham lam. Thuật toán tham lam là một thủ tục thực hiện một lựa chọn tối ưu ở mỗi giai đoạn. Tối ưu hóa ở mỗi giai đoạn của thuật toán không đảm bảo tạo ra lời giải tối ưu toàn cục, nhưng hai thuật toán sau đây để xây dựng cây khung nhỏ nhất là các thuật toán tham lam tạo ra lời giải tối ưu.

Thuật toán đầu tiên do Robert Prim đưa ra năm 1957. Để thực hiện thuật toán ta bắt đầu bằng việc chọn một cạnh bất kỳ có trọng số nhỏ nhất, đặt nó vào cây khung. Lần lượt ghép vào cây các cạnh có trọng số tối thiểu liên thuộc với một đỉnh của cây và không tạo ra chu trình trong cây. Thuật toán dừng khi (n-1) cạnh đã được ghép vào cây.

procedure *PRIM(G: đồ thị liên thông có trọng số với n đỉnh)*

T:= cạnh có trọng số nhỏ nhất

For i:=1 to n-2

Begin

E:= cạnh có trọng số tối thiểu liên thuộc với một đỉnh trong T và không tạo ra chu trình trong T nếu ghép nó vào T

T:=T với e được ghép vào

End {T là cây khung nhỏ nhất}

Lưu ý: Việc chọn một cạnh ghép vào cây trong mỗi giai đoạn của thuật toán là không xác định khi có nhiều hơn một cạnh cùng trọng số và thỏa mãn những tiêu chuẩn nào đó. Cần sắp xếp các cạnh theo một thứ tự nào đó để việc chọn một cạnh được xác định. Cũng cần chú ý là có nhiều hơn một cây khung nhỏ nhất ứng với một đồ thị liên thông và có trọng số.

Thuật toán thứ hai do Joeseph Kruskal phát minh vào năm 1956. Để thực hiện thuật toán này chọn cạnh có trọng số nhỏ nhất của đồ thị. Lần lượt ghép thêm vào cạnh có

trọng số tối thiểu và không tạo thành chu trình với các cạnh đã được chọn. Thuật toán dừng sau khi (n-1) cạnh đã được chọn.

procedure *KRUSKAL(G: đồ thị n đỉnh, liên thông, có trọng số)*

T:= đồ thị rỗng

For i:=1 to n-1

Begin

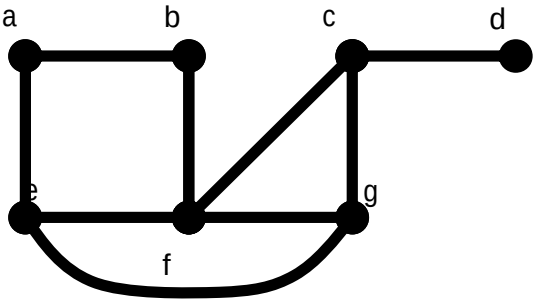
E:=một cạnh bất kỳ của G với trọng số nhỏ nhất và không tạo ra chu trình trong T, khi ghép nó vào T.

T:=T với cạnh e đã được ghép thêm vào.

End {T là cây khung nhỏ nhất}

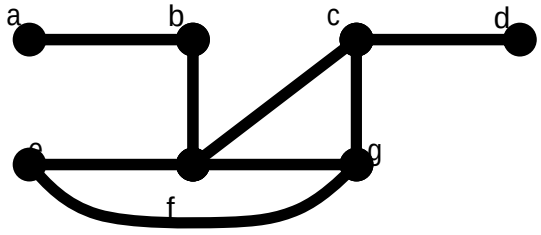
Sự khác nhau giữa hai thuật toán: Trong PRIM chọn các cạnh có trọng số tối thiểu liên thuộc với các đỉnh đã thuộc cây và không tạo ra chu trình. KRUKAL chọn các cạnh có trọng số tối thiểu mà không nhất thiết phải liên thuộc với các đỉnh của cây và không tạo ra chu trình.

2.10. Bài tập ứng dụng:

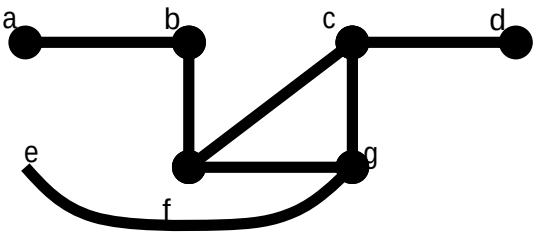


Bài 1. Cho đồ thị như hình dưới bên phải, tìm cây khung.

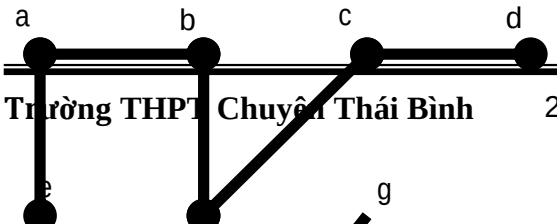
Giải: Đồ thị G liên thông, nhưng không phải là một cây vì nó chứa chu trình đơn. Xóa cạnh {a, e} sẽ loại được một chu trình, đồ thị con nhận được vẫn còn liên thông và chứa tất cả các đỉnh của G. Tiếp theo xóa cạnh {e, f} sẽ loại được một chu trình nữa, cuối cùng xóa cạnh {c, g} sẽ sinh ra một đồ thị không có chu trình. Đồ thị này là cây khung vì nó là cây và chứa tất cả các đỉnh của G. Đáp án được cho bởi hình dưới đây: **Các cây khung của G cho bởi các hình dưới đây:**

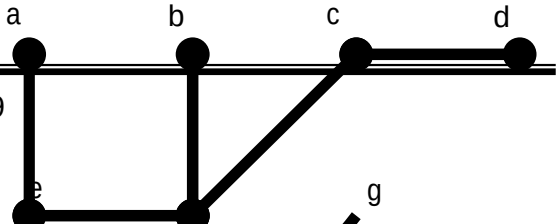


{a, e}

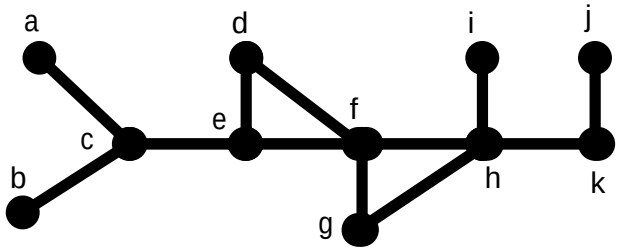


{e, f}

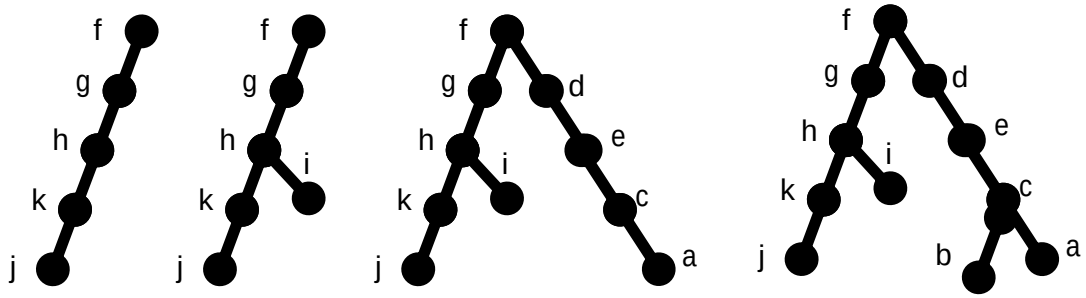




Bài 2. Dùng thuật toán tìm kiếm ưu tiên chiều sâu, tìm cây khung của đồ thị G cho bởi hình bên.

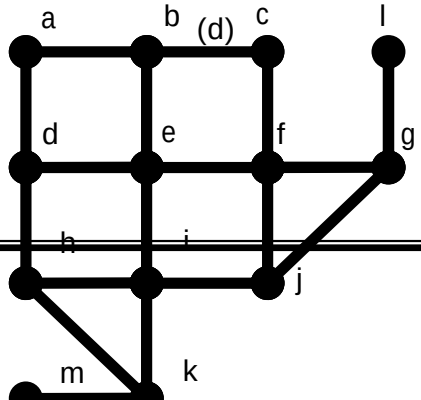


Giải: Xuất phát từ một đỉnh tùy ý, ví dụ đỉnh f. Đường đi được xây dựng bằng cách lần lượt ghép càng nhiều càng tốt, các cạnh liên thuộc với các đỉnh còn chưa thuộc đường đi, điều đó tạo ra được đường đi f, g, h, k j. Tiếp theo, lùi lại k. Không còn đường đi bắt đầu từ k chứa các đỉnh chưa được ghé thăm. Vì thế lùi lại tới h, từ h có đường đi h, i. Sau đó lùi về h và tiếp tục lùi về f. Từ f có đường đi f, d, e, c, a. Ta lại lùi về c và xây dựng đường đi c, b. Thủ tục này đã xây dựng xong cây khung.



(a) (b) (c)

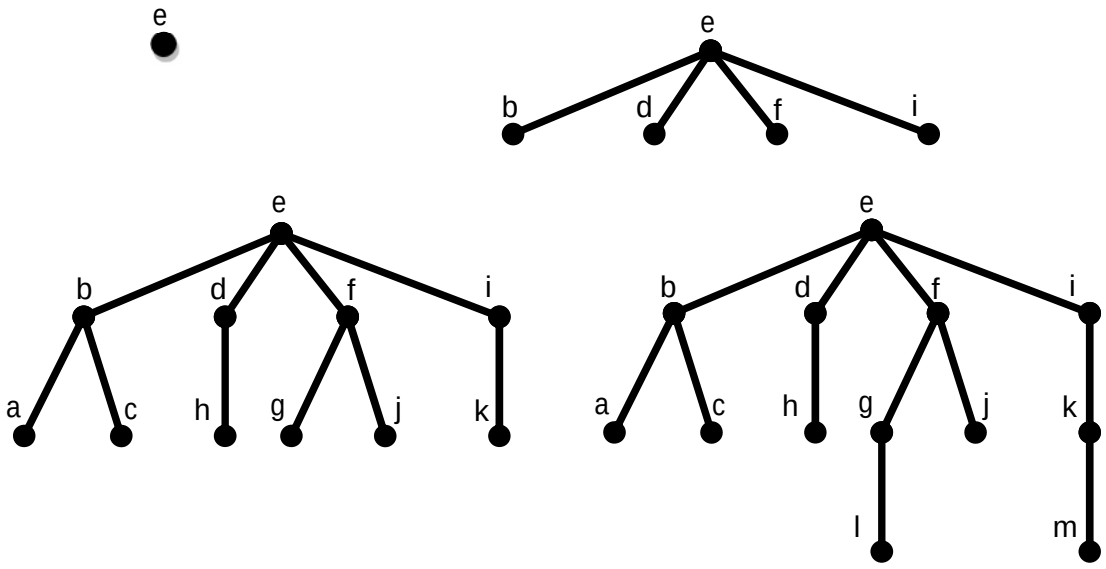
Bài 3. Dùng thuật toán tìm kiếm ưu tiên chiều rộng, tìm



(d)

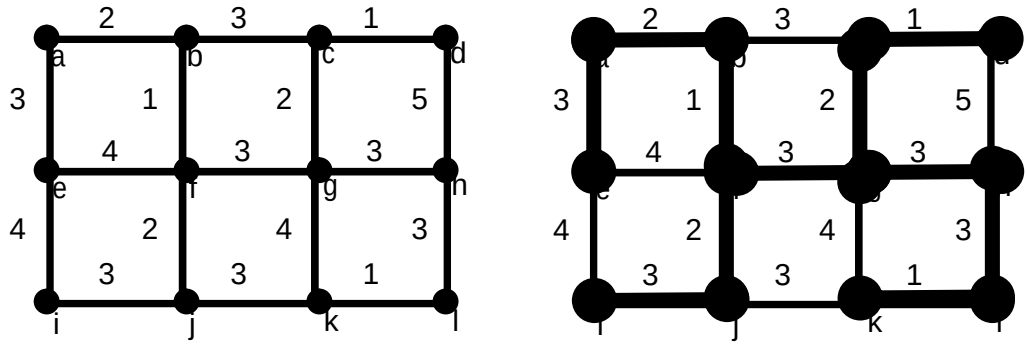
cây khung của đồ thị G cho bởi hình bên.

Giải: Chọn đỉnh e làm gốc của cây, sau đó thêm các cạnh liên thuộc vào tất cả các đỉnh lền kề với e, tức là các cạnh từ e tới b, d, f và i được ghép vào. Vậy mức 1 của cây có 4 đỉnh. Tiếp theo, ghép các cạnh từ các đỉnh ở mức 1 nối với các đỉnh còn chưa có trong cây. Vì thế các cạnh từ b tới a và c được ghép vào, cũng như thế, các cạnh từ d tới h, từ f tới j và g và từ i tới k. Các đỉnh mới a, c, h, g, j, k ở mức 2 của cây. Tiếp theo, ghép các cạnh từ các đỉnh này nối với các đỉnh còn chưa thuộc vào cây, tức là ghép thêm các cạnh từ g tới l và từ k tới m.



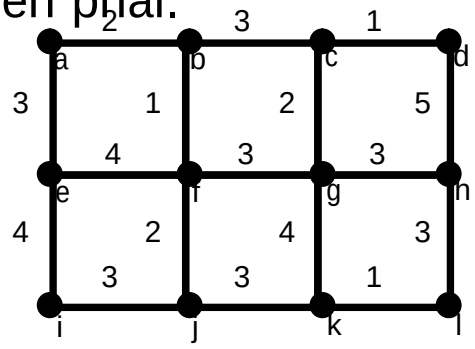
Bài 4. Dùng thuật toán PRIM, tìm cây khung nhỏ nhất của đồ thị đã cho như hình dưới đây.

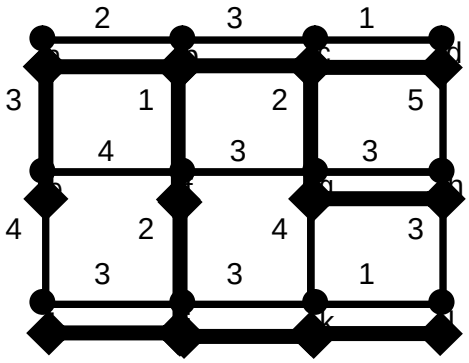
Giải: Cây khung nhỏ nhất được xây dựng bằng thuật toán PRIM thể hiện như hình dưới, bên phải.



Bài 5. Dùng thuật toán KRUSKAL, tìm cây khung nhỏ nhất của đồ thị đã cho như hình dưới đây.

Giải: Cây khung nhỏ nhất được xây dựng bằng thuật toán KRUSKAL thể hiện như hình dưới, bên phải.





PHỤ LỤC: MỘT SỐ CHƯƠNG TRÌNH MẪU THAM KHẢO.

1. KRUSCAL:

```
#include<iostream>
#include<algorithm>
#define SIZE 100
using namespace std;

struct Edge {
    int beginVertex, endVertex, weight;
};

Edge edges[SIZE*SIZE] =
   {{0,1,6},{0,2,5},{0,3,4},{1,3,8},{2,3,3},{2,4,2},{3,4,1}};
int vertexNumber = 5;
int edgeNumber = 7;

Edge mst[SIZE*SIZE];
int mstNumber;
int parent[SIZE];

bool compareEdge(const Edge& e1, const Edge& e2) {
    return (e1.weight < e2.weight);
}

int findSet(int u) {
    int v = u;
    while(parent[v] >= 0)
        v = parent[v];
    return v;
}

void unionSet(int u,int v) {
    int x = findSet(u);
    int y = findSet(v);
    if (parent[x] > parent[y]){
        parent[y] += parent[x];
        parent[x] = y;
    } else {
        parent[x] += parent[y];
        parent[y] = x;
    }
}

void kruskal() {
    int i, beginRoot, endRoot;

    for(i=0; i<vertexNumber; i++)
        parent[i] = -1;

    sort(edges, edges+edgeNumber, compareEdge);
```

```
    for(i = 0; i<edgeNumber; ++i) {
        beginRoot = findSet(edges[i].beginVertex);
        endRoot = findSet(edges[i].endVertex);
        if(beginRoot != endRoot) {

            mst[mstNumber++] = edges[i];
            unionSet(beginRoot, endRoot);
            if(mstNumber == vertexNumber - 1) break;
        }
    }
}

int main() {
    kruskal();

    int i;
    for(i=0; i<mstNumber; ++i)
        cout<<mst[i].beginVertex<<" "<<mst[i].endVertex<<"
   "<<mst[i].weight<<endl;
    return 0;
}
```

2. PRIM

```
#include <iostream.h>
#include "Graph.h"
#include "Queue.h"
#include "Tree.h"

#define MAX 200

#define NIL -1
#define VoCung 32765

int Pi[MAX],key[MAX];

void Prim(const GRAPH G,const int r);
void Print(const GRAPH &G,int A[]);
double TiSoTrongSoMST(const GRAPH &G,int A[]);

void main()
{
    int ch,r;
    GRAPH G;
    do{
        cout<<"\n1.Nhap Do Thi.";
        cout<<"\n2.Xuat Do Thi.";
        cout<<"\n3.Prim.";
        cout<<"\n4.Ti So Trong So MST voi DoThi.";
        cout<<"\n0.Thoat.";
        cout<<"\n  Ban Chon :";cin>>ch;
        switch(ch)
```

```
        {
            case 1:
                cin>>G;
                break;
            case 2:
                cout<<G;
                break;
            case 3:
                cout<<"Nhap Dinh Bat dau :";cin>>r;
                Prim(G,r);
                Print(G,Pi);
                break;
            case 4:
                Prim(G,1);//mac dinh lay dinh dau tien
                cout<<"\n"<<TiSoTrongSoMST(G,Pi)<<"\n";
                break;
        }
    }while(ch!=0);
}

void Print(const GRAPH &G,int A[])
{
    for(int k=1;k<=G.TongSoDinh;k++)
        if(A[k]!=NIL)
            cout<<"("<<k<<","<<A[k]<<"), ";
}
////////////////////////////////////
double TiSoTrongSoMST(const GRAPH &G,int A[])
{

    int sTrongSoMST = 0;
    for(int k=1;k<=G.TongSoDinh;k++)
        if(A[k]!=NIL)
            sTrongSoMST = sTrongSoMST + G.A[k][A[k]];

    int sTrongSoG = 0;
    for(int i=1;i<=G.TongSoDinh;i++)
        for(int j = i;j<=G.TongSoDinh;j++)
            sTrongSoG = sTrongSoG + G.A[i][j];

    return  (1.0*sTrongSoMST/sTrongSoG)*100;
}

void Prim(const GRAPH G,const int r)
{
    int u;
    Queue Q;
    Tree T;

    for(u=1;u<=G.TongSoDinh;u++)
    {
        key[u] =VoCung;
```

```
        Pi[u] = NIL;
        Q.Them(u);
    }

    key[r] = 0;
    Pi[r] = NIL;

    while(Q.isEmpty()!=1)
    {
        u = Q.Extract_min(G,T,r);// tieu chuan uu tien la canh noi dinh se
   duoc bo sung vao Tree la nho nhat
        for(int v = 1;v<=G.TongSoDinh;v++)
            if(G.A[u][v]!=0)
                if((Q.isContain(v)==1) && (G.A[u][v]<key[v]) )
                {
                    key[v] = G.A[u][v];
                    Pi[v] = u;
                }
        T.Them(u);
    }
}
```

3. Breadth First Search

```
#include <iostream>
#include <queue>

using std::cout;
using std::endl;
using std::endl;
using std::cin;

const int maxx = 20;

void Read_input_from_user(bool grid[][maxx], int vertices)
{
    int u, v;
    for(int x = 0; x < vertices; ++x)
    {
        cout << "Enter u : \t";
        cin >> u;
        u--;
        cout << "Enter v : \t";
        cin >> v;
        v--;
        grid[u][v] = true;
                grid[v][u] = true;
        cout << "----------------------\n";
    }
}

void Breadth_first_search(std::queue<int> &Q, std::vector<int> &trace,
                          bool grid[][maxx], int start, int nodes)
```

```
{
    int u;
    Q.push(start);
    trace[start] = -1;
    do{
        u = Q.front();
        Q.pop();
        for(int v = 0; v < nodes; ++v)
        {
            if((grid[u][v] == true) && trace[v] == 0)
            {
                Q.push(v);
                trace[v] = u;
            }
        }
    }while(!Q.empty());
}

void Trace_result(std::vector<int> &trace, int start, int end, int nodes)
{
    cout << "From _nodes" << start + 1 << " you can visit :\n";
    for(int v = 0; v < nodes; ++v)
    {
        if(trace[v] != 0)
        {
            cout << " _nodes : " << v + 1 << " , ";
        }
    }

    cout << "\n-----------------------------------------\n";
    cout << "The path from " << start + 1 << " to " << end + 1 << '\n';

    if(trace[end] == 0){
        cout << "Unavailable.! to go to from " << end + 1
             << " to -> " << start + 1 << '\n';
    }
    else{
        while(end != start)
        {
            cout << end + 1 << "<-";
            end = trace[end];
        }
        cout << start + 1 << endl;
    }

}


int main()
{
    //Initialization
    std::vector<int> trace(maxx, 0);
```

```
    std::queue<int> Q;
    bool grid[maxx][maxx] = {false};

    int nodes, vertices;
    cout << "Please input the number of Node : \n";
    cin >> nodes;
    cout << "Please input the number of Vertices : \n";
    cin >> vertices;

    //Set value for all vertices.
    Read_input_from_user(grid, vertices);

    //Read the necessary path
    int starting_position, finishing_position;
    cout << "Please Input the Starting Node : \n";
    cin >> starting_position;
    cout << "Please Input the Finishing Node : \n";
    cin >> finishing_position;
    //Decrease to fit with index of C++ start from 0->size-1
    starting_position--;
    finishing_position--;
    //Algorithm starts
    Breadth_first_search(Q, trace, grid, starting_position, nodes);
    Trace_result(trace, starting_position, finishing_position, nodes);

    return 0;
}
```

4. Depth First Search

```
/*Bài toán duyêt dinh cua dò thi :

  Input : file van ban path.inp trong dó :
    - Dòng 1 chu só dinh n ( n <= 100 ), só canh m cua dò thi và dinh xuát
   phát s, dinh ket thúc f cách nhau 1 dáu cách.
    - m dòng tiép theo, mõi dòng có dang 2 só nguyên duong u và v cách nhau
   1 dáu cách, the hien canh nói dinh u và v trong dò thi.
  Output :
    - Danh sách các dinh có the dén dc s.
    - Duong di tù s->f.

    Ví du:

    Path.inp:                                   Path.out

  #############                          ############################
  #  8 7 1 5 #                          #                          #
  #          #                          #  From 1 you can visit    #
  #  1 2     #                          #    1, 2, 3, 4, 5         #
  #  1 3     #                          #  The path from 1 -> 5:   #
  #  2 3     #                          #     5 <- 3 <- 2 <- 1     #
  #  2 4     #                          #                          #
  #  3 5     #                          ############################
```

```
  #  4 6     #
  #  7 8     #
  #############

*/
#include <iostream>
#include <vector>

using std::cout;
using std::cin;
using std::endl;

const int maxx = 20;

void Read_input_from_user(bool grid[][maxx], int vertices)
{
    int u, v;
    for(int x = 0; x < vertices; ++x)
    {
        cout << "Enter u : \t";
        cin >> u;
        u--;
        cout << "Enter v : \t";
        cin >> v;
        v--;
        grid[u][v] = true;
        grid[v][u] = true;
        cout << "----------------------\n";
    }
}

void Depth_first_search(bool grid[][maxx], std::vector<int> &trace,
                        int nodes, int u)
{
    for(int v = 0; v < nodes; ++v)
    {
        if((grid[u][v] == true) && (trace[v] == 0))
        {
            trace[v] = u;
            //recursive step
            Depth_first_search(grid, trace, nodes, v);
        }
    }
}

void Trace_result(std::vector<int> &trace, int start, int end, int nodes)
{
    cout << "From _nodes" << start + 1 << " you can visit :\n";
    for(int v = 0; v < nodes; ++v)
    {
        if(trace[v] != 0)
        {
```

```
            cout << " _nodes : " << v + 1 << " , ";
        }
    }

    cout << "\n-----------------------------------------\n";
    cout << "The path from " << start + 1 << " to " << end + 1 << '\n';

    if(trace[end] == 0){
        cout << "Unavailable.! to go to from " << end + 1
             << " to -> " << start + 1 << '\n';
    }
    else{
        while(end != start)
        {
            cout << end + 1 << "<-";
            end = trace[end];
        }
        cout << start + 1 << endl;
    }
}

int main()
{
    bool grid[maxx][maxx] = { false };
    std::vector<int> trace(maxx, 0);
    int nodes, vertices;
    cout << "Please input the number of Node : \n";
    cin >> nodes;
    cout << "Please input the number of Vertices : \n";
    cin >> vertices;

    //Set value for all vertices.
    Read_input_from_user(grid, vertices);

    //Read the necessary path
    int starting_position, finishing_position;
    cout << "Please Input the Starting Node : \n";
    cin >> starting_position;
    cout << "Please Input the Finishing Node : \n";
    cin >> finishing_position;
    //Decrease to fit with index of C++ start from 0->size-1
    starting_position--;
    finishing_position--;

    Depth_first_search(grid, trace, nodes, starting_position);
    Trace_result(trace, starting_position, finishing_position, nodes);
    return 0;
}
```

TÀI LIỆU THAM KHẢO

1. Kenneth H. Rosen *Toán học rời rạc ứng dụng trong Tin học* NXB Khoa học và kỹ thuật, Hà Nội 2007.
2. Nguyễn Đức Nghĩa, Nguyễn Tô Thành *Toán học rời rạc* NXB Đại học Quốc gia Hà Nội 2003.
3. Bùi Minh Trí *Giáo trình Toán ứng dụng trong Tin học* NXB Giáo dục 2004.
4. Đỗ Xuân Lôi *Giáo trình Cấu trúc dữ liệu và Giải thuật* NXB Giáo dục 2005.
5. Kenneth H. Rosen *Discrete Mathematics and Its Applications, Fourth Edition* McGraw-Hill Education 1999.
6. Hồ Sĩ Đàm, Đỗ Đức Đông, Lê Minh Hoàng, Nguyễn Thanh Hùng *Tài liệu giáo khoa chuyên tin* NXB Giáo dục Việt Nam 2009.