

HỒ SĨ ĐÀM (Chủ biên)

ĐỖ ĐỨC ĐÔNG - LÊ MINH HOÀNG - NGUYỄN THANH HÙNG

TÀI LIỆU CHUYÊN TIN HỌC

BÀI TẬP

QUYỂN 2



NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

HÒ SĨ ĐÀM (Chủ biên)
ĐỖ ĐỨC ĐÔNG - LÊ MINH HOÀNG - NGUYỄN THANH HÙNG

**TÀI LIỆU CHUYÊN TIN HỌC
BÀI TẬP QUYỀN 2**

(Tái bản lần thứ nhất)

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

LỜI NÓI ĐẦU

Bộ sách *Tài liệu chuyên Tin học - Bài tập Quyển 1, 2, 3* được viết kèm với bộ *Tài liệu chuyên Tin học - Quyển 1, 2, 3* tương ứng đã được xuất bản. Các tác giả tham gia biên soạn bộ sách là những thầy giáo đã và đang dạy ở các trường chuyên, lớp chọn hoặc tham gia các khóa bồi dưỡng thi tin học quốc tế, bồi dưỡng giáo viên tin cho các trường chuyên theo chương trình của Bộ Giáo dục và Đào tạo, mong muốn xây dựng được các tài liệu có tính hệ thống phục vụ tốt các đối tượng thuộc lĩnh vực chuyên tin học.

Các cuốn *Tài liệu chuyên Tin học - Bài tập* đều có cấu trúc như nhau, gồm hai phần:

Phân I - Bài tập bao gồm tất cả các bài tập trong những chuyên đề của sách *Tài liệu chuyên Tin học* tương ứng và các bài tập bổ sung, được sắp xếp từ dễ đến khó, từ đơn giản đến phức tạp.

Phân II - Hướng dẫn giải bài tập có thể là những hướng dẫn chi tiết để giúp bạn đọc tìm được lời giải hoặc chỉ là đoạn chương trình chính giúp bạn đọc hiểu và tìm được lời giải hoặc chương trình hoàn chỉnh để tham khảo. Đối với một số bài tập thì có thể chỉ là đáp án hay hướng dẫn ngắn gọn.

Hai bộ sách *Tài liệu chuyên Tin học* và *Tài liệu chuyên Tin học - Bài tập* tạo thành hệ thống tài liệu khá hoàn chỉnh theo định hướng Chương trình các chuyên đề chuyên tin học đã được Bộ Giáo dục và Đào tạo ban hành. Do vậy cùng với bộ sách *Tài liệu chuyên Tin học*, bộ sách *Tài liệu chuyên Tin học - Bài tập* sẽ là tài liệu thiết thực phục vụ cho giáo viên, học sinh các trường chuyên, lớp chọn cả Trung học phổ thông và Trung học cơ sở. Ngoài ra, bộ sách còn là tài liệu tham khảo bổ ích cho việc tập huấn sinh viên các trường Đại học, Cao đẳng tham gia các kì thi Olympic Tin học Sinh viên Toàn quốc và Kì thi lập trình viên Quốc tế.

Lưu ý khi sử dụng bộ sách: Các bài tập trong bộ sách này được đánh số như trong sách lì thuyết; các bài tập bổ sung được để ở mục riêng và đánh số tiếp theo.

Mặc dù các tác giả và Ban biên tập đã cố gắng hoàn thiện nhưng chắc chắn bộ sách còn nhiều thiếu sót, các tác giả mong nhận được nhiều ý kiến đóng góp để sách sẽ hoàn thiện hơn, phục vụ bạn đọc được hiệu quả hơn. Các góp ý xin gửi về:

*Ban Toán-Tin, Công ty cổ phần Dịch vụ xuất bản Giáo dục Hà Nội -
Nhà xuất bản Giáo dục Việt Nam, tầng 4, tòa nhà Diamond Flower,
Số 1 Hoàng Đạo Thuý, Hà Nội.*

Các tác giả

L

C

6.1.

6.2.

6.3.

6.4.

CHUYÊN ĐỀ 6. KIỂU DỮ LIỆU TRÙU TƯỢNG VÀ CẤU TRÚC DỮ LIỆU

6.1. Viết chương trình thực hiện các thủ tục chèn, xoá, và tìm kiếm một phần tử trong danh sách các số nguyên đã sắp xếp theo thứ tự tăng dần biểu diễn bởi:

- a) Mảng;
- b) Danh sách nối đơn;
- c) Danh sách nối kép.

6.2. Hãy viết chương trình nối hai danh sách số nguyên đã sắp xếp.

Tổng quát hơn, hãy viết chương trình nối k danh sách số nguyên đã sắp xếp để được một danh sách gồm tất cả các phần tử được sắp xếp.

6.3. Giả sử chúng ta biểu diễn một đa thức

$$p(x) = a_1x^{b_1} + a_2x^{b_2} + \cdots + a_nx^{b_n},$$

trong đó $b_1 > b_2 > \cdots > b_n$ dưới dạng một danh sách nối đơn mà nút thứ i của danh sách chứa hệ số a_i , số mũ b_i và con trỏ trỏ tới nút kế tiếp (nút $i+1$). Hãy tìm thuật toán cộng và nhân hai đa thức theo biểu diễn này.

6.4. Một số nhị phân $a_n a_{n-1} \dots a_0$, trong đó $a_i \in \{0,1\}$ có giá trị bằng $\sum_{i=0}^n a_i 2^i$.

Người ta biểu diễn số nhị phân này bằng một danh sách nối đơn gồm n nút, có nút đầu danh sách chứa giá trị a_n , mỗi nút trong danh sách chứa một chữ số nhị phân a_i và con trỏ trỏ tới nút kế tiếp là nút chứa chữ số nhị phân a_{i-1} .

Hãy lập chương trình thực hiện phép toán “cộng 1” trên số nhị phân đã cho và đưa ra biểu diễn nhị phân của kết quả.

Gợi ý: Sử dụng đệ quy.

- 6.5. Hàng đợi hai đầu (*doubled-ended queue*) là một danh sách được trang bị bốn thao tác:

PushF(v): Đẩy phần tử v vào đầu danh sách.

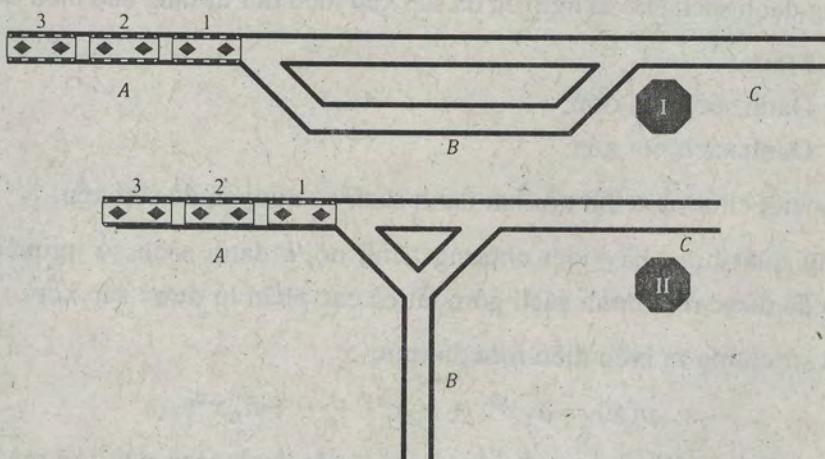
PushR(v): Đẩy phần tử v vào cuối danh sách.

PopF: Loại bỏ phần tử đầu danh sách.

PopR: Loại bỏ phần tử cuối danh sách.

Hãy tìm cấu trúc dữ liệu thích hợp để cài đặt kiểu dữ liệu trùm tượng hàng đợi hai đầu.

- 6.6. Có hai sơ đồ đường ray xe lửa bố trí như hình sau:



Ban đầu có n toa tàu xếp theo thứ tự từ 1 tới n từ phải qua trái trên đường ray A . Người ta muốn xếp lại các toa tàu theo thứ tự mới từ phải qua trái (p_1, p_2, \dots, p_n) lên đường ray C theo nguyên tắc: Các toa tàu không được “vượt nhau” trên ray, mỗi lần chỉ được chuyển một toa tàu từ $A \rightarrow B$, $B \rightarrow C$ hoặc $A \rightarrow C$.

Hãy cho biết điều đó có thể thực hiện được trên sơ đồ đường ray nào trong hai sơ đồ trên?

- 6.7. Xét hai nút x, y trên một cây nhị phân, ta nói nút x nằm bên trái nút y (nút y nằm bên phải nút x) nếu:

- Hoặc nút x nằm trong nhánh con trái của nút y ;
- Hoặc nút y nằm trong nhánh con phải của nút x ;

- Hoặc tồn tại một nút z sao cho x nằm trong nhánh con trái và y nằm trong nhánh con phải của nút z .

Hãy chỉ ra rằng với hai nút x, y bất kì trên một cây nhị phân ($x \neq y$) chỉ có đúng một trong bốn mệnh đề sau là đúng:

- x nằm bên trái y ;
- x nằm bên phải y ;
- x là tiền bối thực sự của y ;
- y là tiền bối thực sự của x .

- 6.8. Với mỗi nút x trên cây nhị phân T , giả sử rằng ta biết được các giá trị $Preorder[x]$, $Inorder[x]$ và $Postorder[x]$ lần lượt là thứ tự duyệt trước, giữa, sau của x .

Tìm cách chỉ dựa vào các giá trị này để kiểm tra hai nút có quan hệ tiền bối-hậu duệ hay không.

- 6.9. *Bậc (degree)* của một nút là số nút con của nó. Chứng minh rằng trên cây nhị phân, số lá nhiều hơn số nút bậc 2 đúng một nút.

- 6.10. Hãy chỉ ra rằng cấu trúc của một cây nhị phân có thể khôi phục một cách đơn định nếu ta biết được thứ tự duyệt trước và giữa của các nút.

Tương tự như vậy, cấu trúc cây có thể khôi phục nếu ta biết được thứ tự duyệt sau và giữa của các nút.

- 6.11. Hãy cho ví dụ về hai cây nhị phân khác nhau nhưng có thứ tự trước của các nút giống nhau và thứ tự sau của các nút cũng giống nhau.

- 6.12. Xét biểu thức có thể có dạng phức tạp, chẳng hạn biểu thức bao gồm cả phép lấy số đối ($-x$), phép tính luỹ thừa (x^y), hàm số với một hay nhiều biến số.

Ta có thể biểu diễn những biểu thức dạng này bằng một cây tổng quát và từ đó có thể chuyển biểu thức về dạng hậu tố hay kí pháp nghịch đảo Ba Lan (RPN) để thực hiện tính toán.

Hãy xây dựng thuật toán để chuyển biểu thức số học (dạng phức tạp) về dạng RPN và thuật toán tính giá trị biểu thức đó.

- 6.13. Viết chương trình chuyển biểu thức lôgic dạng trung tố sang dạng RPN.
Ví dụ chuyển: a and b or c and d thành: $a\ b\ \text{and}\ c\ d\ \text{and}\ \text{or}$.

6.14. Chuyển các biểu thức sau đây ra dạng RPN:

- a) $A \times (B + C);$
- b) $A + \frac{B}{C} + D;$
- c) $A \times (B + C);$
- d) $A - (B + C)^{\frac{D}{E}};$
- e) $(A \text{ or } B) \text{ and } (C \text{ or } (D \text{ or not } E));$
- f) $(A = B) \text{ or } (C = D).$

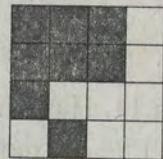
6.15. Với một ảnh đen/trắng kích thước $2^n \times 2^n$, người ta dùng phương pháp sau để mã hoá ảnh:

- Nếu ảnh chỉ gồm toàn điểm đen thì ảnh đó có thể được mã hoá bằng xâu chỉ gồm một kí tự ‘B’;
- Nếu ảnh chỉ gồm toàn điểm trắng thì ảnh đó có thể được mã hoá bằng xâu chỉ gồm một kí tự ‘W’;
- Nếu P, Q, R, S lần lượt là xâu mã hoá của bốn ảnh kích thước bằng nhau thì $\&PQRS$ là xâu mã hoá của ảnh tạo thành bằng cách đặt bốn ảnh ban đầu theo sơ đồ:

$P \quad Q$
 $S \quad R$

Ví dụ “ $\&B\&BWWB\&W\&BWBW$ ” và

“ $\&\&BBBB\&BWWB\&W\&BWBW$ ” là hai xâu mã hoá của



Bài toán đặt ra là cho số nguyên dương n và hai xâu mã hoá của hai ảnh kích thước $2^n \times 2^n$. Hãy cho biết hai ảnh đó có khác nhau không và nếu chúng khác nhau thì chỉ ra một vị trí có màu khác nhau trên hai ảnh.

6.16. Quá trình tìm kiếm trên cây nhị phân tìm kiếm (BST) có thể coi như một đường đi xuất phát từ nút gốc. Giáo sư X phát hiện ra một tính chất thú vị:

Nếu đường đi trong quá trình tìm kiếm kết thúc ở một nút lá, kí hiệu L là tập các giá trị chứa trong các nút nằm bên trái đường đi và R là tập các giá trị chứa trong các nút nằm bên phải đường đi. Khi đó $\forall x \in L, y \in R$, ta có

$x \leq y$. Hãy chứng minh phát hiện của giáo sư X là đúng hoặc chỉ ra một phản ví dụ.

- 6.17. Cho BST gồm n nút, bắt đầu từ nút Minimum^\wedge , người ta gọi hàm *Successor* để đi sang nút liền sau cho tới khi duyệt qua nút Maximum^\wedge .

Chứng minh rằng thuật toán này có thời gian thực hiện $O(n)$.

Gợi ý: Thực hiện n lời gọi hàm *Successor* liên tiếp để duyệt qua các liên kết cha/con trên BST, mỗi liên kết tối đa hai lần.

Điều tương tự có thể chứng minh được nếu ta bắt đầu từ nút Maximum^\wedge và gọi liên tiếp hàm *Predecessor* để đi sang nút liền trước.

- 6.18. Cho BST có chiều cao h . Bắt đầu từ một nút p_1 , người ta tìm nút p_2 là nút liền sau p_1 : $p_2 := \text{Successor}(p_1)$, tiếp theo lại tìm nút p_3 là nút liền sau p_2, \dots . Chứng minh rằng thời gian thực hiện k lần lời gọi hàm *Successor* như vậy là $O(k + h)$.

- 6.19. Người ta có thể thực hiện việc sắp xếp một dãy khoá bằng cây nhị phân tìm kiếm như sau: Chèn lần lượt các giá trị khoá vào một cây nhị phân tìm kiếm sau đó duyệt cây theo thứ tự giữa (thuật toán Tree Sort).

Hãy đánh giá thời gian thực hiện thuật toán trong trường hợp tốt nhất, xấu nhất và trung bình. Cài đặt thuật toán Tree Sort.

- 6.20. Hãy viết thuật toán *SearchLE*(k) để tìm nút chứa khoá lớn nhất không vượt quá k trong BST.

- 6.21. Hãy viết thuật toán *SearchGE*(k) để tìm nút chứa khoá nhỏ nhất không nhỏ hơn k trong BST.

- 6.22. Hãy viết thủ tục *MovetoRoot*(p) nhận vào nút p và dùng các phép quay để chuyển nút p thành gốc của cây BST.

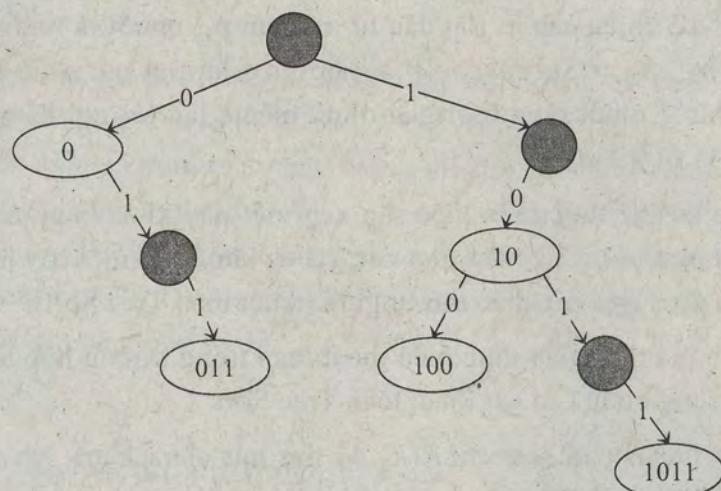
- 6.23. Hãy viết thủ tục *MovetoLeaf*(p) nhận vào nút p và dùng các phép quay để chuyển nút p thành một nút lá của cây BST.

- 6.24. Cây tìm kiếm cơ sở (*Radix Tree*) là một cây nhị phân, trong đó mỗi nút có thể chứa hoặc không chứa giá trị khoá (người ta thường dùng một giá trị đặc biệt tương ứng với nút không chứa giá trị khoá hoặc sử dụng thêm một bit đánh dấu những nút không chứa giá trị khoá).

Các giá trị khoá lưu trữ trên Radix Tree là các dãy nhị phân, hay tổng quát hơn là một kiểu dữ liệu nào đó có thể mã hoá bằng các dãy nhị phân.

Thủ tục chèn một khoá vào Radix Tree được thực hiện như sau: Bắt đầu từ nút gốc ta duyệt biểu diễn nhị phân của khoá, gấp bit 0 đi sang nhánh con trái và gấp bit 1 đi sang nhánh con phải. Mỗi khi không đi được nữa (đi vào liên kết *nilT*), ta tạo ra một nút và nối nó vào cây ở chỗ liên kết *nilT* vừa rẽ sang rồi đi tiếp. Cuối cùng, ta đặt khoá vào nút cuối cùng trên đường đi.

Hình dưới đây là Radix Tree sau khi chèn các giá trị 1011, 10, 100, 0, 011. Các nút tô đậm là nút không chứa khoá.



Gọi S là tập chứa các khoá gồm các dãy nhị phân, tổng độ dài các dãy nhị phân trong S là n .

Chứng minh rằng chỉ cần thời gian $O(n)$ để xây dựng Radix Tree chứa các phần tử của S , cần thời gian $O(n)$ để duyệt Radix Tree theo thứ tự giữa và liệt kê các phần tử của S theo thứ tự từ điển.

- 6.25. Chò BST tạo thành từ n khoá được chèn vào theo một trật tự ngẫu nhiên. Gọi X là biến ngẫu nhiên cho chiều cao của BST.

Chứng minh rằng kì vọng $E[X] = O(\lg n)$.

- 6.26. Gọi $b(n)$ là số lượng các cây nhị phân tìm kiếm chứa n khoá hoàn toàn phân biệt.

a) Chứng minh rằng $b(0) = 1$ và $b(n) = \sum_{k=0}^{n-1} b_k b_{n-1-k}$;

b) Chứng minh rằng $b(n) = \frac{1}{n+1} \binom{2n}{n}$ (số Catalan thứ n).

Từ đó suy ra xác suất để BST là cây nhị phân gần hoàn chỉnh (hoặc cây nhị phân suy biến) nếu n khoá được chèn vào theo thứ tự ngẫu nhiên;

c) Hãy chứng minh công thức xấp xỉ $b(n) = \frac{4^n}{\sqrt{\pi n^{\frac{3}{2}}}} \left(1 + O\left(\frac{1}{n}\right)\right)$.

6.27. Thứ tự thống kê

Cho một Treap, hãy xây dựng thuật toán tìm khoá đứng thứ p khi sắp thứ tự. Ngược lại cho một nút, hãy tìm số thứ tự của nút đó khi duyệt Treap theo thứ tự giũa.

6.28. Treap biểu diễn tập hợp

Khi dùng Treap T biểu diễn tập hợp các giá trị khoá (tức là các khoá trong Treap hoàn toàn phân biệt), phép thứ $k \in T$ có thể được thực hiện thông qua hàm *Search*. Việc thêm một phần tử vào tập hợp có thể được thực hiện thông qua một sửa đổi của hàm *Insert* (chỉ chèn nếu khoá chưa có trong Treap). Việc xoá một phần tử khỏi tập hợp cũng được thực hiện thông qua việc sửa đổi thủ tục *Delete* (tìm phần tử trong Treap, nếu tìm thấy thì thực hiện thủ tục xoá). Ngoài ra, còn có nhiều thao tác khác được thực hiện rất hiệu quả với cấu trúc Treap, hãy cài đặt các thao tác sau đây trên Treap:

Phép tách (Split): Với một giá trị k_0 , tách các khoá có giá trị nhỏ hơn k_0 và các khoá có giá trị lớn hơn k_0 ra hai Treap để biểu diễn hai tập hợp riêng rẽ.

Gợi ý: Tìm nút chứa phần tử k_0 trong Treap, nếu không thấy thì chèn k_0 vào một nút mới. Đặt độ ưu tiên của nút này bằng $+\infty$. Theo nguyên lí của cấu trúc Treap, nút này sẽ được đẩy lên thành gốc của cây. Ngoài ra, theo nguyên lí của cấu trúc BST, nhánh con trái của gốc sẽ chứa tất cả các khoá có giá trị nhỏ hơn k_0 và nhánh con phải của gốc sẽ chứa tất cả các khoá có giá trị lớn hơn k_0 .

6. *Phép hợp (Union):* Cho hai Treap chứa hai tập khoá, xây dựng Treap mới chứa tất cả các khoá của hai Treap ban đầu.

Phép giao (Intersection): Cho hai Treap chứa hai tập khoá, xây dựng Treap mới chứa tất cả các khoá có mặt trong cả hai Treap ban đầu.

Phép lấy hiệu (Difference): Cho hai Treap A, B chứa hai tập khoá, xây dựng Treap mới chứa các khoá thuộc A nhưng không thuộc B .

- 6.29. Cho một BST chứa n khoá, và hai khoá a, b . Liệt kê tất cả các khoá k của BST thoả mãn $a \leq k \leq b$. Hãy xây dựng thuật toán có độ phức tạp $O(m + \lg n)$ (m là số khoá được liệt kê).

- 6.30. Cho n là một số nguyên dương và $x = (x_1, x_2, \dots, x_n)$ là một hoán vị của dãy số $(1, 2, \dots, n)$.

Với $i: 1 \leq i \leq n$, gọi t_i là số phần tử đứng trước giá trị i mà lớn hơn i trong dãy x . Khi đó dãy $t = (t_1, t_2, \dots, t_n)$ được gọi là dãy nghịch thế của dãy $x = (x_1, x_2, \dots, x_n)$.

Ví dụ, với $n = 6$, cho dãy $x = (3, 2, 1, 6, 4, 5)$ thì dãy nghịch thế của nó là $t = (2, 1, 0, 1, 1, 0)$.

Hãy xây dựng thuật toán có độ phức tạp $O(n \lg n)$ tìm dãy nghịch thế từ dãy hoán vị cho trước và thuật toán có độ phức tạp $O(n \lg n)$ để tìm dãy hoán vị từ dãy nghịch thế cho trước.

- 6.31. Trên mặt phẳng với hệ toạ độ Descartes *Oxy* cho n hình chữ nhật có cạnh song song với các trục toạ độ. Hãy tìm thuật toán có độ phức tạp $O(n \lg n)$ để tính diện tích phần mặt phẳng bị n hình chữ nhật đó chiếm chỗ.

- 6.32. Cho n dây cung của một hình tròn, trong đó không có hai dây cung nào chung đầu mút. Tìm thuật toán có độ phức tạp $O(n \lg n)$ xác định số cặp dây cung cắt nhau bên trong hình tròn (ví dụ, nếu n dây cung đều là đường kính thì số cặp là $\binom{n}{2}$).

- 6.33. Trên trục số cho n đoạn, đoạn thứ i là $[a_i, b_i]$ ($a_i, b_i \in \mathbb{N}$). Hãy chọn trên trục số một số ít nhất các điểm nguyên phân biệt sao cho có ít nhất c_i điểm được chọn thuộc vào đoạn thứ i ($1 \leq n \leq 10^5; 0 \leq a_i, b_i, c_i \leq 10^5$).

6.34. Bản đồ một khu đất hình chữ nhật kích thước $m \times n$ được chia thành lưới ô vuông đơn vị, trên đó có đánh dấu k ô trống cây ($m, n, k \leq 10^5$). Người ta muốn giải phóng một mặt bằng nằm trong khu đất này. Bản đồ mặt bằng có các ràng buộc sau:

- Cạnh mặt bằng có độ dài là số nguyên;
- Mặt bằng chiếm trọn một số ô trên bản đồ;
- Cạnh mặt bằng song song với cạnh bản đồ.

Hãy trả lời hai câu hỏi sau:

- Nếu muốn xây dựng mặt bằng với cạnh là D thì phải giải phóng ít nhất bao nhiêu ô trống cây?
- Nếu không muốn giải phóng ô trống cây nào thì có thể xây dựng được mặt bằng với cạnh lớn nhất là bao nhiêu?

6.35. Một bộ $n \leq 10^5$ lá bài được xếp thành tập và mỗi lá bài được ghi số thứ tự ban đầu của lá bài đó trong tập bài (vị trí các lá bài được đánh số từ 1 tới n , từ trên xuống dưới).

Xét thao tác tráo kí hiệu bởi $S(i, j)$: rút ra lá bài thứ i và chèn lên trên lá bài thứ j trong số $n - 1$ lá bài còn lại ($1 \leq i, j \leq n$).

Quy ước rằng nếu $j = n$ thì lá bài thứ i sẽ được đặt vào vị trí dưới cùng của tập bài.

Ví dụ, với $n = 6$:

$$\begin{aligned} (1, \boxed{2}, 3, 4, 5, 6) &\xrightarrow{S(2,3)} (1, 3, \boxed{2}, 4, 5, 6) \\ (\boxed{1}, 3, 2, 4, 5, 6) &\xrightarrow{S(1,2)} (3, \boxed{1}, 2, 4, 5, 6) \\ (3, 1, 2, \boxed{4}, 5, 6) &\xrightarrow{S(4,5)} (3, 1, 2, 5, \boxed{4}, 6) \\ (\boxed{3}, 1, 2, 5, 4, 6) &\xrightarrow{S(1,6)} (1, 2, 5, 4, 6, \boxed{3}). \end{aligned}$$

Người ta tráo bộ bài bằng x phép tráo ($x \leq 10^5$). Người chơi được biết x thao tác tráo đó. Hãy sử dụng thêm một số ít nhất các thao tác tráo nữa để đưa các lá bài về vị trí ban đầu.

Với ví dụ trên, ta cần sử dụng thêm hai thủ tục tráo $S(6,3)$ và $S(5,4)$.

BÀI TẬP BỔ SUNG

6.36. Xoá số

Cho dãy số nguyên $A = (a_1, a_2, \dots, a_n)$. Người ta tìm chỉ số i ($1 < i < n$) nhỏ nhất thoả mãn điều kiện $a_i < a_{i-1}$ và $a_i < a_{i+1}$ rồi xoá đi số a_i khỏi dãy. Sau khi xoá, số phần tử trong dãy A giảm đi 1 và các phần tử còn lại của dãy được đánh chỉ số lại từ 1 bắt đầu từ a_1 . Công việc này lặp lại cho tới khi không tìm được chỉ số i thoả mãn điều kiện trên.

Yêu cầu: Cho biết số phần tử còn lại trong dãy.

Input: Tệp văn bản DELNUM.INP bao gồm:

- Dòng 1 chứa số nguyên dương $n \leq 10^5$;
- Dòng 2 chứa n số nguyên a_1, a_2, \dots, a_n ($\forall i: |a_i| \leq 10^9$), cách nhau ít nhất một dấu cách.

Output: Tệp văn bản DELNUM.OUT ghi một số nguyên duy nhất là số lượng phần tử còn lại trong dãy.

Ví dụ:

DELNUM.INP	DELNUM.OUT
6	3
3 1 2 4 0 1	

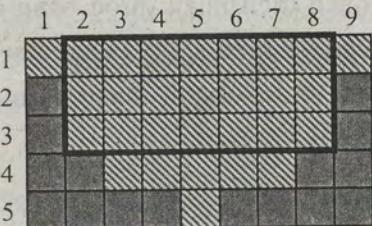
6.37. Hình chữ nhật lớn nhất

Cho một bảng hình chữ nhật kích thước $m \times n$ được chia thành lưới ô vuông đơn vị m hàng, n cột. Các hàng được đánh số từ 1 tới m theo thứ tự từ trên xuống dưới và các cột được đánh số từ 1 tới n theo thứ tự từ trái qua phải. Người ta tiến hành tô màu các ô của bảng theo từng cột: Các ô trên mỗi cột j sẽ được tô từ trên xuống dưới: h_j ô màu vàng tiếp đến là $m - h_j$ ô màu xanh. Như vậy tình trạng màu trên bảng hoàn toàn xác định nếu ta biết được số hàng m , số cột n và các số nguyên h_1, h_2, \dots, h_n .

Hãy xác định một hình chữ nhật gồm các ô trong bảng đã cho thoả mãn các yêu cầu sau:

- Có cạnh song song với cạnh bảng;
- Đơn sắc (chỉ gồm các ô vàng hoặc chỉ gồm các ô xanh);

- Diện tích lớn nhất có thể.



$$\begin{aligned}m &= 5 \\n &= 9 \\H &= (1, 3, 4, 4, 5, 4, 4, 3, 1)\end{aligned}$$

Input: Tệp văn bản RECT.INP bao gồm:

- Dòng 1: Chứa hai số nguyên dương m, n ($m, n \leq 10^6$);
- Dòng 2: Chứa n số nguyên h_1, h_2, \dots, h_n ($\forall j: 0 \leq h_j \leq m$).

Output: Tệp văn bản RECT.OUT bao gồm:

- Dòng 1: Ghi diện tích hình chữ nhật tìm được;
- Dòng 2: Ghi chỉ số hàng và chỉ số cột của ô ở góc trên trái của hình chữ nhật tìm được;
- Dòng 3: Ghi chỉ số hàng và chỉ số cột của ô ở góc dưới phải của hình chữ nhật tìm được.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

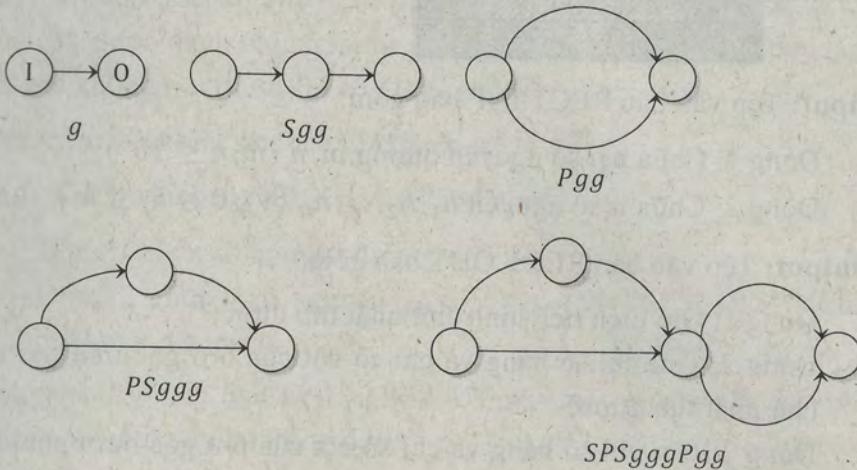
RECT.INP	RECT.OUT
5 9 1 3 4 4 5 4 4 3 1	21 1 2 > 3 8

6.38. Tập độc lập

Người ta mô hình hóa một mạch điện một chiều theo cách đệ quy như sau:

- Một mạch điện có một đầu vào I và một đầu ra O với một dây dẫn nối từ I tới O được kí hiệu bằng một kí tự g .
- Nếu G_1 là mạch điện có đầu vào I_1 và đầu ra O_1 , G_2 là mạch điện có đầu vào I_2 và đầu ra O_2 thì mạch điện nhận được bằng cách chập đầu ra O_1 và đầu vào I_2 thành một điểm sẽ trở thành mạch điện nối tiếp có đầu vào I_1 và đầu ra O_2 , kí hiệu bằng xâu kí tự SG_1G_2 .

- Nếu G_1 là mạch điện có đầu vào I_1 và đầu ra O_1 , G_2 là mạch điện có đầu vào I_2 và đầu ra O_2 thì mạch điện nhận được bằng cách chập hai đầu vào I_1, I_2 thành một đầu vào (kí hiệu I_{12}) và chập hai đầu ra O_1, O_2 thành một đầu ra (kí hiệu O_{12}) sẽ trở thành mạch điện song song có đầu vào I_{12} và đầu ra O_{12} , kí hiệu PG_1G_2 .



Một tập các điểm được gọi là tập độc lập nếu nó không chứa hai điểm nào có dây dẫn trực tiếp. Hãy xác định số lượng điểm trong tập độc lập lớn nhất của một mạng điện cho bởi xâu kí tự gồm các chữ cái P, S, g theo quy tắc trên.

Input: Tệp văn bản INDEP.INP gồm một dòng chứa không quá 10^6 kí tự.

Output: Tệp văn bản INDEP.OUT ghi số lượng điểm trong tập độc lập lớn nhất.

Ví dụ:

INDEP.INP	INDEP.OUT
SPSgggPgg	2

6.39. Bởi số nhỏ nhất

Cho số nguyên dương n và một tập $S \subset \{0, \dots, 9\}$. Hãy tìm một số nguyên dương m tạo thành từ các chữ số thuộc S thoả mãn các điều kiện sau đây:

- m chia hết cho n .
- m nhỏ nhất có thể.

Input: Tệp văn bản LM.INP bao gồm:

- Dòng 1 chứa số nguyên dương $n \leq 10^6$;
- Dòng 2 chứa không quá 10 kí tự liền nhau, mỗi kí tự là một chữ số trong tập S .

Output: Tệp văn bản LM.OUT gồm một dòng duy nhất chứa số m tìm được. Nếu không tồn tại số m thoả mãn các yêu cầu đặt ra thì ghi trên dòng này một số 0.

Ví dụ:

LM.INP	LM.OUT
7	42
24	
18	144
14	
10	0
1234	

6.40. Trộm đào

Tôn Ngộ Không lén vào vườn đào của Vương Mẫu và nhìn thấy một cây đào trĩu quả. Ngộ Không bèn niệm chú gọi Thổ Địa lên hỏi thì được biết: Cây đào này có n quả đánh số từ 1 tới n , quả thứ i phải tới thời điểm t_i mới chín và có giá trị k_i (giá trị k_i cho biết khi ăn quả đào đó thì tuổi thọ sẽ tăng lên k_i năm, còn gọi là giá trị của quả đào).

Tại mỗi thời điểm, Ngộ Không có thể chọn một quả đào chín để hái. Việc hái mỗi quả đào mất đúng một đơn vị thời gian, Ngộ Không phải hái từng quả một.

Ngoài ra, để tránh bị phát hiện, Ngộ Không dự kiến sẽ chỉ hái đào từ thời điểm a tới hết thời điểm b (quả đào cuối cùng phải được hái xong không muộn hơn thời điểm b).

Hãy chỉ cho Ngộ Không một cách chọn các quả đào để hái sao cho tổng giá trị những quả đào được chọn là lớn nhất có thể.

Input: Tệp văn bản PEACH.INP bao gồm:

- Dòng 1 chứa ba số nguyên dương n, a, b ($n \leq 10^5; a < b \leq 10^9$).

- Dòng thứ i trong n dòng tiếp theo chứa hai số nguyên dương t_i, k_i ($t_i, k_i \leq 10^9, \forall i: 1 \leq i \leq n$).

Output: Tệp văn bản PEACH.OUT gồm một số nguyên duy nhất là tổng giá trị những quả đào trong phương án tìm được.

Ví dụ:

PEACH.INP	PEACH.OUT
5 1 4	
1 10	
1 5	
1 6	
3 20	
4 100	36

Giải thích: Phương án tối ưu là:

Thời điểm 1: Háiquảđào 1 (giá trị 10);

Thời điểm 2: Háiquảđào 3 (giá trị 6);

Thời điểm 3: Háiquảđào 4 (giá trị 20);

Thời điểm 4: Rời khỏi vườn đào.

Tổng giá trị: $10 + 6 + 20 = 36$.

Lưu ý: Không hái được quả đào thứ 5 vì Ngô Không hái đào trong khoảng thời gian từ thời điểm 1 đến trước thời điểm 4 mà đúng thời điểm 4 quả đào thứ 5 mới chín (mặc dù giá trị của quả đào thứ 5 có thể rất lớn).

6.41. Cầu cảng

Một cảng biển có m cầu cảng để tiếp nhận các tàu cập bến. Tại một thời điểm, mỗi cầu cảng chỉ có thể tiếp nhận không quá một tàu. Ban đầu các cầu cảng đều trống và có n tàu xin đăng ký cập bến, tàu thứ i muốn đậu ở cảng từ ngay sau thời điểm s_i tới hết thời điểm f_i . Có thể coi thời gian tàu thứ i muốn đậu ở cảng là một khoảng $(s_i, f_i]$ trên trục thời gian. Tàu đã vào cầu cảng nào thì sẽ đậu ở đó trong suốt thời gian nằm cảng.

Yêu cầu: Hãy cho biết với m cầu cảng đã cho, có thể tiếp nhận tối đa bao nhiêu tàu và chỉ ra lịch trình tiếp nhận tại mỗi cầu cảng.

Input: Tệp văn bản SEAPORTS.INP bao gồm:

- Dòng 1: Chứa hai số nguyên dương $m, n \leq 10^5$;
- Dòng thứ i trong n dòng tiếp theo chứa hai số nguyên s_i, f_i ($0 \leq s_i < f_i \leq 10^5$).

Output: Tệp văn bản SEAPORTS.OUT bao gồm:

- Dòng 1: Ghi số lượng tàu được tiếp nhận phục vụ;
- Dòng 2: Ghi n số nguyên, số thứ i là số hiệu cảng sẽ tiếp nhận tàu thứ i trong trường hợp tàu thứ i được tiếp nhận, còn nếu tàu thứ i không được tiếp nhận thì số thứ i là 0.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

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

6.42. Thắng Bờm và Phú Ông

Bờm thắng Phú Ông trong một cuộc đánh cược và buộc Phú Ông phải đãi rượu. Phú Ông bèn bày ra một dãy n chai chứa đầy rượu và nói với Bờm rằng có thể uống bao nhiêu tùy ý, nhưng đã chọn chai nào thì phải uống hết và không được uống ở k chai liền nhau bởi đó là điều xui xẻo.

Bạn hãy chỉ cho Bờm cách uống được nhiều rượu nhất.

Input: Tệp văn bản BOTTLES.INP bao gồm:

- Dòng 1 chứa hai số nguyên n, k ($1 \leq n \leq 4.10^5; 2 \leq k \leq 4.10^5$);
- Dòng 2 chứa n số nguyên dương (nhỏ hơn hoặc bằng 10^9) là dung tích của các chai rượu Phú Ông bày ra, theo thứ tự liệt kê từ chai thứ nhất tới chai thứ n .

Output: Tệp văn bản BOTTLES.OUT bao gồm:

- Dòng 1 ghi số chai được chọn và lượng rượu tối đa có thể uống;
- Dòng 2 ghi chỉ số của các chai được chọn theo thứ tự tăng dần.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

BOTTLES.INP	BOTTLES.OUT
6 3	4 40
6 10 10 13 10 10	2 3 5 6

6.43. Tráo bài

Phú Ông có bộ bài gồm n lá bài. Phú Ông xếp chúng thành tập và ghi vào mỗi lá bài số thứ tự ban đầu của lá bài đó trong tập bài (vị trí các lá bài được đánh số từ 1 tới n , từ trên xuống dưới).

Tiếp theo Phú Ông tiến hành tráo tập bài, mỗi phép tráo kí hiệu bởi $S(i,j)$: rút ra lá bài thứ i và chèn lên trên lá bài thứ j trong số $n - 1$ lá bài còn lại ($1 \leq i, j \leq n$), quy ước rằng nếu $j = n$ thì lá bài thứ i sẽ được đặt vào vị trí dưới cùng của tập bài.

Ví dụ với $n = 6$:

$$\begin{aligned} (1, \boxed{2}, 3, 4, 5, 6) &\xrightarrow{S(2,3)} (1, 3, \boxed{2}, 4, 5, 6) \\ (\boxed{1}, 3, 2, 4, 5, 6) &\xrightarrow{S(1,2)} (3, \boxed{1}, 2, 4, 5, 6) \\ (3, 1, 2, \boxed{4}, 5, 6) &\xrightarrow{S(4,5)} (3, 1, 2, 5, \boxed{4}, 6) \\ (\boxed{3}, 1, 2, 5, 4, 6) &\xrightarrow{S(1,6)} (1, 2, 5, 4, 6, \boxed{3}). \end{aligned}$$

Sau x phép tráo, Phú Ông đưa cho Bờm tập bài và thách Bờm dùng ít phép tráo nhất để xếp lại các lá bài về vị trí ban đầu. Hãy giúp Bờm thực hiện điều đó.

Input: Tệp văn bản SHUFFLE.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương n, x ($n, x \leq 10^5$);
- Dòng thứ p trong x dòng tiếp theo chứa hai số nguyên i_p, j_p cho biết phép tráo thứ p của Phú Ông là $S(i_p, j_p)$.

Output: Tệp văn bản SHUFFLE.OUT bao gồm:

- Dòng 1 ghi số y là số phép tráo cần thực hiện để đưa các lá bài về vị trí ban đầu;

- Dòng thứ q trong y dòng tiếp theo ghi hai số nguyên i_q, j_q cho biết phép tráo thứ q của Bờm là $S(i_q, j_q)$.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

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

6.44. Những hòn đảo

Bản đồ một trang trại là một hình chữ nhật kích thước $m \times n$ được chia thành lưới các ô vuông đơn vị. Các hàng của lưới được đánh số từ 1 tới m từ trên xuống dưới và các cột của lưới được đánh số từ 1 tới n từ trái qua phải. Ô giao của hàng x , cột y được gọi là ô (x, y) và ô đó có độ cao h_{xy} .

Trong những ngày mưa tầm tã, mực nước dâng lên và trang trại bị ngập dần trong nước. Nếu mực nước là k thì những ô có độ cao nhỏ hơn hoặc bằng k được coi là ngập nước còn những ô có độ cao lớn hơn k được coi là chưa ngập nước. Những ô chưa ngập nước tạo thành những “đảo”, định nghĩa: Hai ô chưa ngập nước được gọi là cùng đảo nếu ta có thể đi từ ô này tới ô kia bằng cách di chuyển qua các ô kè cạnh chưa ngập nước, ngược lại hai ô đó được coi là nằm trên hai đảo khác nhau.

Ví dụ với bản đồ dưới đây, ta có bốn đảo khi mực nước bằng 2, có hai đảo khi mực nước bằng 7.

9	1	8	1	5	4
7	1	8	1	5	5
7	7	8	1	1	1
1	1	1	1	6	6
3	3	1	6	6	1
3	3	1	6	1	1

$$k = 2$$

9	1	8	1	5	4
7	1	8	1	5	5
7	7	8	1	1	1
1	1	1	1	6	6
3	3	1	6	6	1
3	3	1	6	1	1

$$k = 7$$

Yêu cầu: Giả sử trong những ngày mưa, mực nước dâng dần lên cho tới khi toàn bộ các ô đều ngập nước. Xác định số đảo tại một thời điểm trong những ngày mưa mà tại thời điểm đó có nhiều đảo nhất.

Input: Tệp văn bản ISLANDS.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương m, n ($m, n \leq 1000$);
- Dòng thứ i trong m dòng tiếp theo chứa n số nguyên dương, số thứ j là h_{ij} ($h_{ij} \leq 10^6$).

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

Output: Tệp văn bản ISLANDS.OUT gồm một số nguyên duy nhất là số đảo tại thời điểm có nhiều đảo nhất.

Ví dụ:

ISLANDS.INP	ISLANDS.OUT
6 6	4
9 1 8 1 5 4	
7 1 8 1 5 5	
7 7 8 1 1 1	
1 1 1 1 6 6	
3 3 1 6 6 1	
3 3 1 6 1 1	

6.45. Quản lý lương

Một công ty có n người đánh số từ 1 tới n , người thứ i có lương là w_i . Tổng Giám đốc công ty được đánh số 1, mỗi người từ 2 tới n có đúng một thủ trưởng trực tiếp của mình. Ta nói rằng người i quản lí người j nếu tồn tại dãy $i = x_1, x_2, \dots, x_k = j$ sao cho người x_i là thủ trưởng trực tiếp của người x_{i+1} .

Mỗi người được quyền tăng hoặc trừ lương của tất cả mọi người trong quyền quản lí của mình. Bạn cần viết một chương trình quản lí lương xử lí hai loại tác vụ:

- $p A x$: Người A tăng lương của tất cả những người trong quyền quản lí của mình thêm x đồng ($-10000 \leq x \leq 10000$);
- $u A$: Cho biết lương của người A .

tới
ong
ứ j
số
i.
pt
n
a
g
í

Chú ý: Lương của mỗi người trong suốt quá trình quản lí năm trong phạm vi các số nguyên có dấu 32 bit.

Input: Tệp văn bản SALARY.INP bao gồm:

- Dòng 1: Chứa hai số nguyên dương n, m ($n, m \leq 5.10^5$), trong đó m là số các tác vụ;
- Dòng thứ i trong n dòng tiếp theo chứa lương khởi điểm và số hiệu thủ trưởng của người i . Riêng với người 1, dòng tương ứng sẽ chỉ có lương khởi điểm.
- m dòng tiếp, mỗi dòng chứa một tác vụ.

Output: Tệp văn bản SALARY.OUT: Với mỗi tác vụ loại u , in ra kết quả trên một dòng.

Ví dụ:

SALARY.INP	SALARY.OUT
6 7	7
5	9
4 1	7
3 2	5
7 3	
2 3	
3 5	
p 3 2	
p 2 4	
u 3	
u 6	
p 5 -2	
u 6	
u 1	

CHUYÊN ĐỀ 7. ĐỒ THỊ

7.1. Cho đồ thị có hướng $G = (V, E)$ không có chu trình âm. Hãy tìm thuật toán có độ phức tạp $O(|V||E|)$ để tính tất cả các $\delta^*(u) = \min_{v \in V} \{\delta(u, v)\}$.

- 7.2. Cho đồ thị có hướng $G = (V, E)$ gồm n đỉnh và m cung.

Hãy tìm thuật toán có độ phức tạp $O(nm)$ để xác định đồ thị có chu trình âm hay không và chỉ ra một chu trình âm nếu có.

7.3. Hệ ràng buộc

Cho x_1, x_2, \dots, x_n là các biến số, cho m ràng buộc, mỗi ràng buộc có dạng:

$$x_j - x_i \leq w_{ij} (w_{ij} \in \mathbb{R}).$$

Hãy tìm cách gán giá trị cho các biến x_1, x_2, \dots, x_n thoả mãn tất cả các ràng buộc đã cho.

7.4. Cải tiến của Yen cho thuật toán Bellman-Ford

Với đồ thị có hướng $G = (V, E)$ gồm n đỉnh, ta đánh số các đỉnh từ 1 tới n . Chia tập cung E làm hai tập con: E_1 gồm các cung nối từ đỉnh có chỉ số nhỏ tới đỉnh có chỉ số lớn và E_2 gồm các cung nối từ đỉnh có chỉ số lớn tới đỉnh có chỉ số nhỏ. Đặt $G_1 = (V, E_1)$ và $G_2 = (V, E_2)$, hai đồ thị này là hai đồ thị có hướng không có chu trình được biểu diễn bằng danh sách kề.

Thuật toán Bellman-Ford sau đó được thực hiện như sau:

```

Init;
repeat
    Stop := True;
    for u := 1 to n - 1 do
        for ∀v: (u, v) ∈ E1 do
            if Relax(u, v) then Stop := False;
    for u := n downto 2 do
        for ∀v: (u, v) ∈ E2 do
            if Relax(u, v) then Stop := False;
until Stop;

```

Bên trong vòng lặp *repeat...until* là hai pha tối ưu nhãn: pha thứ nhất xét các đỉnh theo thứ tự tăng dần còn pha thứ hai xét các đỉnh theo thứ tự giảm dần của chỉ số. Mỗi khi một đỉnh được xét và thực hiện phép co theo tất cả các cung đi ra khỏi u , mỗi pha thực hiện tương tự như thuật toán tìm đường đi ngắn nhất trên đồ thị không có chu trình.

Hãy chỉ ra rằng vòng lặp *repeat...until* trong cải tiến của Yen lặp không quá $\left\lceil \frac{n}{2} \right\rceil$ lần. Cài đặt thuật toán và so sánh với cách cài đặt chuẩn của thuật toán Bellman-Ford.

- 7.5. Arbitrage là một cách sử dụng sự bất hợp lí trong hối đoái tiền tệ để kiếm lời. Ví dụ nếu 1\$ mua được 0.7£, 1£ mua được 190¥, 1¥ mua được 0.009\$ thì từ 1\$, ta có thể đổi sang 0.7£, sau đó sang $0.7 \times 190 = 133¥$, rồi đổi lại sang $133 \times 0.009 = 1.197\$$. Kiếm được 0.197\$ lãi.

Giả sử rằng có n loại tiền tệ đánh số từ 1 tới n . Bảng $R = \{r_{ij}\}_{n \times n}$ cho biết tỉ giá hối đoái: một đơn vị tiền i đổi được r_{ij} đơn vị tiền j .

Hãy tìm thuật toán để xác định xem có thể kiếm lời từ bảng tỉ giá hối đoái này bằng phương pháp Arbitrage hay không? Nếu có thể sử dụng Arbitrage, hãy chỉ ra một cách kiếm lời.

7.6. Thuật toán Karp tìm chu trình có trung bình trọng số nhỏ nhất

Cho đồ thị có hướng $G = (V, E)$ gồm n đỉnh, hàm trọng số $w: E \rightarrow \mathbb{R}$.

Ta định nghĩa trung bình trọng số của một chu trình C gồm các cung $\langle e_1, e_2, \dots, e_k \rangle$ như sau:

$$\mu(C) = \frac{1}{k} \sum_{i=1}^k w(e_i).$$

Đặt $\mu^* = \min_C \{\mu(C)\}$.

Khi đó chu trình C có $\mu(C) = \mu^*$ gọi là chu trình có trung bình trọng số nhỏ nhất (*minimum mean-weight cycle*). Chu trình này có nhiều ý nghĩa trong các thuật toán tìm luồng với chi phí cực tiểu.

Không làm mất tính tổng quát, giả sử mọi đỉnh $v \in V$ đều đến được từ một đỉnh $s \in V$ (ta có thể thêm một đỉnh giả s và cung trọng số 0 nối từ s tới mọi đỉnh khác, s không nằm trên chu trình đơn nào nên không ảnh hưởng tới tính đúng đắn của thuật toán). Đặt $\delta(v)$ là độ dài đường đi ngắn nhất từ s tới v . Đặt $\delta_k(v)$ là độ dài đường đi ngắn nhất trong số các đường đi từ s tới v qua đúng k cung (ta có thể thêm vào các cung trọng số đủ lớn để với

mọi cặp đỉnh u, v luôn tồn tại cung (u, v) và (v, u) . Việc tìm chu trình trung bình trọng số nhỏ nhất không bị ảnh hưởng bởi những cung thêm vào và $\delta_k(v)$ luôn là giá trị hữu hạn).

a) Chứng minh rằng nếu $\mu^* = 0$, thì G không có chu trình âm và:

$$\delta(v) = \min_{0 \leq k \leq n-1} \{\delta_k(v)\}, \forall v \in V.$$

b) Chứng minh rằng nếu $\mu^* = 0$ thì

$$\max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n-k} \geq 0, \forall v \in V.$$

Gợi ý: Sử dụng kết quả câu a).

c) Gọi C là một chu trình trọng số 0; u, v là hai đỉnh nằm trên C . Giả sử $\mu^* = 0$ và x là độ dài đường đi từ u tới v dọc theo chu trình C .

Chứng minh rằng $\delta(v) = \delta(u) + x$.

d) Chứng minh rằng nếu $\mu^* = 0$ thì trên mỗi chu trình trọng số 0 sẽ tồn tại một đỉnh v sao cho:

$$\max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n-k} = 0.$$

e) Chứng minh rằng nếu $\mu^* = 0$ thì

$$\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n-k} = 0.$$

f) Hãy chỉ ra rằng nếu ta cộng thêm một hằng số Δ vào tất cả các trọng số cung thì μ^* tăng lên Δ . Sử dụng tính chất này để chứng minh rằng

$$\mu^* = \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n-k}.$$

g) Hãy xây dựng thuật toán có độ phức tạp $O(|V||E|)$ và lập chương trình để tính μ^* rồi chỉ ra một chu trình có trung bình trọng số nhỏ nhất.

7.7. Trên mặt phẳng cho n đường tròn, đường tròn thứ i được cho bởi bộ ba số thực (x_i, y_i, r_i) , với (x_i, y_i) là tọa độ tâm và r_i là bán kính. Chi phí di chuyển trên mỗi đường tròn bằng 0. Chi phí di chuyển giữa hai đường tròn bằng khoảng cách giữa chúng. Hãy tìm phương án di chuyển giữa hai đường tròn s, t cho trước với chi phí ít nhất.

- nh
êm
- 7.8. Thuật toán Dijkstra có thể sai nếu đồ thị có cạnh trọng số âm. Hãy cho một ví dụ để chứng minh điều đó.
- 7.9. Cho đồ thị vô hướng $G = (V, E)$ có n đỉnh và m cạnh, các cạnh có trọng số là số nguyên trong phạm vi từ 0 tới k .
Hãy thay đổi thuật toán Dijkstra để được thuật toán có độ phức tạp $O(kn + m)$ tìm đường đi ngắn nhất xuất phát từ một đỉnh $s \in V$.
- 7.10. Yêu cầu như với Bài tập 7.9 nhưng thuật toán tìm được có độ phức tạp là $O((m + n) \log k)$.

7.11. Thuật toán Gabow

Xét đồ thị $G = (V, E)$ có các trọng số cạnh là số tự nhiên $w: E \rightarrow \mathbb{N}$. Giả sử rằng từ đỉnh xuất phát s có đường đi tới mọi đỉnh khác. Gọi k là trọng số lớn nhất của các cạnh trong E . Gọi $z = \lceil \lg(k + 1) \rceil$, khi đó trọng số mỗi cạnh có thể được biểu diễn bằng một dãy z bit. Với mỗi cạnh $e \in E$ mang trọng số $w(e)$, ta kí hiệu $w_i(e)$ là số tạo thành bằng i bit đầu tiên của $w(e)$, tức là:

$$w_i(e) = w(e) \text{ div } 2^{z-i}, (\forall i = 1, 2, \dots, z).$$

Ví dụ $z = 5$ và $w(e) = 11 = 01011_{(2)}$. Ta có:

$$\begin{aligned} w_1(e) &= 0_{(2)} = 0 \\ w_2(e) &= 01_{(2)} = 1 \\ w_3(e) &= 010_{(2)} = 2 \\ w_4(e) &= 0101_{(2)} = 5 \\ w_5(e) &= 01011_{(2)} = 11. \end{aligned}$$

Định nghĩa $\delta_i(s, v)$ là độ dài đường đi ngắn nhất từ s tới v trên đồ thị G với hàm trọng số w_i . Rõ ràng $w_z(e) = w(e)$ nên $\delta_z(s, v) = \delta(s, v)$ với $\forall v \in V$.

a) Giả sử rằng $\delta(s, v) \leq |E|$ với mọi đỉnh v có đường đi từ s . Hãy xây dựng thuật toán có độ phức tạp $O(|E|)$ để xác định tất cả các $\delta(s, v)$.

Gợi ý: Sử dụng hàng đợi ưu tiên.

b) Chứng minh rằng các $\delta_1(s, v)$ có thể tính được trong thời gian $O(|E|)$.

Gợi ý: Chú ý rằng các trọng số $w_1(e) \in \{0, 1\}$.

c) Hãy chỉ ra rằng với mọi $i = 2, 3, \dots, z$:

$$w_i(e) = 2 \cdot w_{i-1}(e) \text{ hoặc } w_i(e) = 2 \cdot w_{i-1}(e) + 1.$$

Từ đó chứng minh rằng:

$$2 \cdot \delta_{i-1}(s, v) \leq \delta_i(s, v) < 2 \cdot \delta_{i-1}(s, v) + |V|.$$

Gợi ý: Độ dài đường đi ngắn nhất từ s tới v sẽ nhân đôi nếu ta nhân đôi các trọng số cạnh.

d) Với mọi cạnh $e = (u, v) \in E$, định nghĩa:

$$\widehat{w}_i(e) = \widehat{w}_i(u, v) = w_i(u, v) + 2 \cdot \delta_{i-1}(s, u) - 2 \cdot \delta_{i-1}(s, v).$$

Chứng minh rằng với mọi đường đi $p: u \rightsquigarrow v$, ta có:

$$\widehat{w}_i(p) = w_i(p) + 2 \cdot \delta_{i-1}(s, u) - 2 \cdot \delta_{i-1}(s, v).$$

e) Định nghĩa $\widehat{\delta}_i(s, v)$ là độ dài đường đi ngắn nhất từ s tới v trên đồ thị G với hàm trọng số \widehat{w}_i . Chứng minh rằng với $i = 2, 3, \dots, z$ và $\forall v \in V$:

$$\widehat{\delta}_i(s, v) = \delta_i(s, v) - 2 \cdot \delta_{i-1}(s, v) \leq |E|.$$

f) Hãy xây dựng thuật toán tính các $\delta_i(s, v)$ từ các $\delta_{i-1}(s, v)$ trong thời gian $O(|E|)$. Từ đó chứng minh rằng có thể tìm đường đi ngắn nhất trên đồ thị G trong thời gian $O(|E| \cdot z) = O(|E| \lg k)$.

7.12. Cho một bảng kích thước $m \times n$ các số tự nhiên. Từ một ô có thể di chuyển sang một ô kè cạnh với nó.

Hãy tìm một cách đi từ ô (x, y) đến một ô biên sao cho tổng các số ghi trên các ô đi qua là nhỏ nhất.

7.13. Cho một dãy số nguyên $A = (a_1, a_2, \dots, a_n)$.

Hãy tìm một dãy con gồm nhiều nhất các phần tử của dãy đã cho mà tổng của hai phần tử liên tiếp là số nguyên tố.

7.14. Một công trình lớn được chia làm n công đoạn. Công đoạn i phải thực hiện mất thời gian t_i . Quan hệ giữa các công đoạn được cho bởi bảng $A = \{a_{ij}\}_{n \times n}$ trong đó $a_{ij} = 1$ nếu công đoạn j chỉ được bắt đầu khi công đoạn i đã hoàn thành và $a_{ij} = 0$ trong trường hợp ngược lại. Mỗi công đoạn khi bắt đầu cần thực hiện liên tục cho tới khi hoàn thành, hai công đoạn độc lập nhau có thể tiến hành song song.

Hãy bố trí lịch thực hiện các công đoạn sao cho thời gian hoàn thành cả công trình là ít nhất, cho biết thời gian ít nhất đó.

T.15. Cho đồ thị $G = (V, E)$, các cạnh được gán trọng số không âm. Xây dựng thuật toán và viết chương trình tìm một chu trình có độ dài nhỏ nhất trên G .

T.16. Cho T là cây khung nhỏ nhất của đồ thị G và (u, v) là một cạnh trong T .

Chứng minh rằng nếu ta trừ trọng số cạnh (u, v) đi một số dương thì T vẫn là cây khung nhỏ nhất của đồ thị G .

T.17. Cho G là một đồ thị vô hướng liên thông, C là một chu trình trên G và e là cạnh có trọng số lớn nhất của C . Chứng minh rằng nếu ta loại bỏ cạnh e khỏi đồ thị thì không ảnh hưởng tới trọng số của cây khung nhỏ nhất.

T.18. Chứng minh rằng đồ thị có duy nhất một cây khung nhỏ nhất nếu với mọi lát cắt của đồ thị, có duy nhất một cạnh nhẹ nối hai tập của lát cắt.

Cho một ví dụ để chỉ ra rằng điều ngược lại không đúng.

T.19. Gọi T là một cây khung nhỏ nhất của đồ thị vô hướng liên thông G . Ta giảm trọng số của một cạnh không nằm trong cây T . Hãy tìm một thuật toán đơn giản để tìm cây khung nhỏ nhất của đồ thị mới.

T.20. Giả sử rằng đồ thị vô hướng liên thông G có cây khung nhỏ nhất T , người ta thêm vào đồ thị một đỉnh mới và một số cạnh liên thuộc với đỉnh đó.

Hãy tìm thuật toán xác định cây khung nhỏ nhất của đồ thị mới.

T.21. Giáo sư X đề xuất một thuật toán tìm cây khung nhỏ nhất dựa trên ý tưởng chia để trị: Với đồ thị vô hướng liên thông $G = (V, E)$, phân hoạch tập đỉnh V làm hai tập rời nhau V_1, V_2 mà lực lượng của hai tập này hơn kém nhau không quá 1. Gọi E_1 là tập các cạnh chỉ liên thuộc với các đỉnh thuộc V_1 và E_2 là tập các cạnh chỉ liên thuộc với các đỉnh thuộc V_2 . Tìm cây khung nhỏ nhất trên đồ thị $G_1 = (V_1, E_1)$ và $G_2 = (V_2, E_2)$ bằng thuật toán đệ quy, sau đó chọn cạnh có trọng số nhỏ nhất nối V_1 với V_2 để nối hai cây khung tìm được thành một cây. Hãy chứng minh tính đúng đắn của thuật toán hoặc chỉ ra một phản ví dụ cho thấy thuật toán sai.

T.22. *Cây khung nhỏ thứ nhì*

Cho $G = (V, E, w)$ là đồ thị vô hướng liên thông có trọng số, giả sử rằng $|E| \geq |V|$ và các trọng số cạnh là hoàn toàn phân biệt (w là đơn ánh). Gọi

T là tập tất cả các cây khung của G và A là cây khung nhỏ nhất của G . Khi đó cây khung nhỏ thứ nhì được định nghĩa là cây khung $B \in T$ thoả mãn:

$$w(B) = \min_{T \in T - \{A\}} \{w(T)\}.$$

- a) Hãy chỉ ra rằng đồ thị G có duy nhất một cây khung nhỏ nhất là A , nhưng có thể có nhiều cây khung nhỏ thứ nhì.
- b) Chứng minh rằng luôn tồn tại một cạnh $(u, v) \in A$ và $(x, y) \notin A$ để nếu ta loại bỏ cạnh (u, v) khỏi A rồi thêm cạnh (x, y) vào A thì sẽ được cây khung nhỏ thứ nhì.
- c) Với $\forall u, v \in V$, gọi $f[u, v]$ là cạnh mang trọng số lớn nhất trên đường đi duy nhất từ u tới v trên cây A . Hãy xây dựng thuật toán có độ phức tạp $O(|V|^2)$ để tính tất cả các $f[u, v]$, $\forall u, v \in V$.
- d) Hãy đề xuất thuật toán hiệu quả để tìm cây khung nhỏ thứ nhì của đồ thị.

- 7.23. Cho s và t là hai đỉnh của một đồ thị vô hướng có trọng số $G = (V, E, w)$. Hãy tìm một đường đi từ s tới t thoả mãn: Trọng số cạnh lớn nhất đi qua trên đường đi là nhỏ nhất có thể.

Gợi ý: Có rất nhiều cách làm: Kết hợp một thuật toán tìm kiếm trên đồ thị với thuật toán tìm kiếm nhị phân, hoặc sửa đổi thuật toán Dijkstra, hoặc sử dụng thuật toán tìm cây khung nhỏ nhất.

7.24. Euclidean Minimum Spanning Tree

Trong trường hợp các đỉnh của đồ thị đầy đủ được đặt trên mặt phẳng trực chuẩn và trọng số cạnh nối giữa hai đỉnh chính là khoảng cách hình học giữa chúng. Người ta có một phép tiền xử lí để giảm bớt số cạnh của đồ thị bằng thuật toán tam giác phân Delaunay (thời gian thực hiện $O(n \lg n)$). Đồ thị sau phép tam giác phân Delaunay sẽ còn không quá $3n$ cạnh, do đó sẽ làm các thuật toán tìm cây khung nhỏ nhất hoạt động hiệu quả hơn.

Hãy tự tìm hiểu về phép tam giác phân Delaunay và cài đặt chương trình để tìm cây khung nhỏ nhất.

- 7.25. Trên một nền phẳng với hệ toạ độ Descartes đặt n máy tính, máy tính thứ i được đặt ở toạ độ (x_i, y_i) . Đã có sẵn một số dây cáp mạng nối giữa một số cặp máy tính. Cho phép nối thêm các dây cáp mạng nối giữa từng cặp máy

tính. Chi phí nối một dây cáp mạng tỉ lệ thuận với khoảng cách giữa hai máy cần nối. Hãy tìm cách nối thêm các dây cáp mạng để cho các máy tính trong toàn mạng là liên thông và chi phí nối mạng là nhỏ nhất.

- 7.26. Hệ thống điện trong thành phố được cho bởi n trạm biến thế và các đường dây điện nối giữa các cặp trạm biến thế. Mỗi đường dây điện e có độ an toàn là $p(e) \in (0,1]$. Độ an toàn của cả lưới điện là tích độ an toàn trên các đường dây. Hãy tìm cách bỏ đi một số dây điện để cho các trạm biến thế vẫn liên thông và độ an toàn của mạng là lớn nhất có thể.

Gợi ý: Bằng kĩ thuật lấy lôgarit, độ an toàn trên lưới điện trở thành tổng độ an toàn trên các đường dây.

- 7.27. Cho f_1 và f_2 là hai luồng trên mạng $G = (V, E, c, s, t)$ và α là một số thực nằm trong đoạn $[0,1]$. Xét ánh xạ:

$$\begin{aligned}f_\alpha: E &\rightarrow \mathbb{R} \\e &\mapsto f_\alpha(e) = \alpha f_1(e) + (1 - \alpha) f_2(e).\end{aligned}$$

Chứng minh rằng f_α cũng là một luồng trên mạng G với giá trị luồng:

$$|f_\alpha| = \alpha |f_1| + (1 - \alpha) |f_2|.$$

- 7.28. Cho f là một luồng trên mạng $G = (V, E, c, s, t)$. Chứng minh rằng với mọi $e \in E$, ta có:

$$c_f(e) + c_f(-e) = c(e) + c(-e).$$

- 7.29. Cho f là luồng cực đại trên mạng $G = (V, E, c, s, t)$. Gọi Y là tập các đỉnh đến được t bằng một đường thặng dư trên G_f và $X = V - Y$.

Chứng minh rằng (X, Y) là lát cắt $s - t$ hẹp nhất của mạng G .

- 7.30. Hãy viết chương trình nhận vào một đồ thị có hướng $G = (V, E)$ với hai đỉnh phân biệt s và t và tìm một tập gồm nhiều đường đi nhất từ s tới t sao cho các đường đi trong tập này đôi một không có cạnh chung.

- 7.31. Tương tự như Bài tập 7.29 nhưng yêu cầu thực hiện trên đồ thị vô hướng.

7.32. Hệ đại diện phân biệt

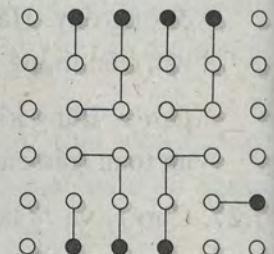
Một lớp học có n bạn nam và n bạn nữ. Nhân ngày 8/3, lớp có mua n món quà để các bạn nam tặng các bạn nữ. Mỗi món quà có thể thuộc sở thích của một số bạn trong lớp.

Hãy lập chương trình tìm cách phân công tặng quà thỏa mãn:

- Mỗi bạn nam phải tặng quà cho đúng một bạn nữ và mỗi bạn nữ phải nhận quà của đúng một bạn nam. Món quà được tặng phải thuộc sở thích của cả hai người.
- Món quà nào đã được một bạn nam chọn để tặng thì bạn nam khác không được chọn nữa.

7.33. Cho mạng điện gồm $m \times n$ điểm nằm trên một lưới m hàng, n cột. Một số điểm nằm trên biên của lưới là nguồn điện, một số điểm trên lưới là các thiết bị sử dụng điện. Người ta chỉ cho phép nối dây điện giữa hai điểm nằm cùng hàng hoặc cùng cột.

Hãy tìm cách đặt các dây điện nối các thiết bị sử dụng điện với nguồn điện sao cho hai đường dây bất kì nối hai thiết bị sử dụng điện với nguồn điện tương ứng của chúng không được có điểm chung.



7.34. Kỹ thuật dẫn sức chứa

Cho mạng $G = (V, E, c, w, s, t)$ với sức chứa nguyên: $c: E \rightarrow \mathbb{N}$.

Gọi $C := \max_{e \in E} c(e)$.

- Chứng minh rằng lát cắt $s - t$ hẹp nhất của G có lưu lượng không vượt quá $C|E|$.
- Với một số nguyên k , tìm thuật toán xác định đường tăng luồng có giá trị thặng dư không nhỏ hơn k trong thời gian $O(|E|)$.
- Chứng minh rằng thuật toán sau đây tìm được luồng cực đại trên mạng G :

```
procedure MaxFlowByScaling;
begin
  f := «Luồng 0»;
  k := C; //k là sức chứa lớn nhất của một cung trong E
  while k ≥ 1 do
    begin
      while «Tìm được đường tăng luồng P có giá trị
            thặng dư không nhỏ hơn k» do
        «Tăng luồng dọc theo đường P»;
```

```

    k := k div 2;
end;
end;

```

d) Chứng minh rằng khi bước vào mỗi lượt lặp của vòng lặp:

```
while k ≥ 1 do...
```

lưu lượng của lát cắt hẹp nhất trên mạng thặng dư G_f không vượt quá $2k|E|$.

e) Chứng minh rằng trong mỗi lượt lặp của vòng lặp:

```
while k ≥ 1 do...
```

vòng lặp while bên trong thực hiện $O(|E|)$ lần với mỗi giá trị của k .

f) Chứng minh rằng thuật toán trên (*maximum flow by scaling*) có thể cài đặt để tìm luồng cực đại trên G trong thời gian $O(|E|^2 \log C)$.

7.35. Phân công

Có p thợ và q việc. Biết rằng mỗi thợ có thể làm được những việc nào và mỗi việc khi giao cho một thợ thực hiện sẽ được hoàn thành xong trong đúng một đơn vị thời gian. Tại một thời điểm, mỗi thợ chỉ thực hiện không quá một việc. Hãy phân công các thợ làm các công việc sao cho:

- Mỗi việc chỉ giao cho đúng một thợ thực hiện.
- Thời gian hoàn thành tất cả các công việc là nhỏ nhất.

Chú ý là các thợ có thể thực hiện song song các công việc được giao, việc của ai người này làm, không ảnh hưởng tới người khác.

7.36. Một bộ ghép M trên đồ thị hai phía gọi là tối đai nếu việc bổ sung thêm bất cứ cạnh nào vào M sẽ làm cho M không còn là bộ ghép nữa.

- a) Hãy chỉ ra một ví dụ về bộ ghép tối đai nhưng không là bộ ghép cực đai trên đồ thị hai phía.
- b) Hãy xây dựng thuật toán $O(|E|)$ để xác định một bộ ghép tối đai trên đồ thị hai phía.
- c) Chứng minh rằng nếu A và B là hai bộ ghép tối đai trên cùng một đồ thị hai phía thì $|A| \leq 2|B|$ và $|B| \leq 2|A|$. Từ đó chỉ ra rằng nếu thuật toán đường mòn được khởi tạo bằng một bộ ghép tối đai thì số lượt tìm

đường mở giảm đi ít nhất một nửa so với việc khởi tạo bằng bộ ghép rỗng.

7.37. Phủ đỉnh – Vertex Cover

Cho đồ thị hai phía $G = (X \cup Y, E)$. Hãy chọn một tập C gồm ít nhất các đỉnh sao cho mọi cạnh thuộc E đều liên thuộc với ít nhất một đỉnh thuộc C .

Bài toán tìm phủ đỉnh nhỏ nhất trên đồ thị tổng quát là NP-đầy đủ, hiện tại chưa có thuật toán đa thức để giải quyết. Tuy vậy trên đồ thị hai phía, phủ đỉnh nhỏ nhất có thể tìm được dựa trên bộ ghép cực đại.

Dựa vào mô hình luồng của bài toán bộ ghép cực đại, giả sử các cung (X, Y) có sức chứa $+\infty$, các cung (s, X) và (Y, t) có sức chứa 1.

Gọi (S, T) là lát cắt hẹp nhất của mạng. Đặt $C = \{x \in T\} \cup \{y \in S\}$.

- Chứng minh rằng C là một phủ đỉnh.
- Chứng minh rằng C là phủ đỉnh nhỏ nhất.
- Giả sử ta tìm được M là bộ ghép cực đại trên đồ thị hai phía, khi đó chắc chắn không còn tồn tại đường mở tương ứng với bộ ghép M . Đặt:

$$Y^* = \{y \in Y : \exists x \in X \text{ chưa ghép, } x \text{ đến được } y \text{ qua một đường pha}\};$$

$$X^* = \{x \in X : x \text{ đã ghép và đỉnh ghép với } x \text{ không thuộc } Y^*\}.$$

Chứng minh rằng (X^*, Y^*) là lát cắt hẹp nhất.

- Hãy xây dựng thuật toán tìm phủ đỉnh nhỏ nhất trên đồ thị hai phía dựa trên thuật toán tìm bộ ghép cực đại.

7.38. Cho M là một bộ ghép trên đồ thị hai phía $G = (X \cup Y, E)$. Gọi k là số X _đỉnh chưa ghép. Chứng minh rằng ba mệnh đề sau đây là tương đương:

- M là bộ ghép cực đại.
- G không có đường mở tương ứng với bộ ghép M .
- Tồn tại một tập con A của X sao cho $|N(A)| = |A| - k$. Ở đây $N(A)$ là tập các Y _đỉnh kề với một đỉnh nào đó trong A .

7.39. Định lí Hall

Cho $G = (X \cup Y, E)$ là đồ thị hai phía có $|X| = |Y|$.

ghép

Chứng minh rằng G có bộ ghép đầy đủ (bộ ghép mà mọi đỉnh đều được ghép) nếu và chỉ nếu $|A| \leq |N(A)|$ với mọi tập $A \subseteq X$.

7.40. Phủ đường tối thiểu

Cho $G = (V, E)$ là đồ thị có hướng không có chu trình.

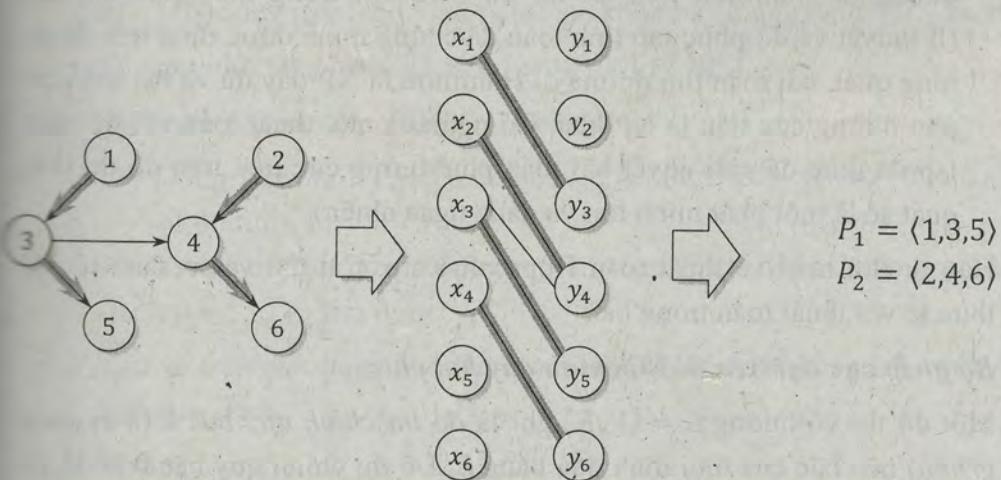
Một *phủ đường* (*path cover*) là một tập P các đường đi trên G thỏa mãn: Với mọi đỉnh $v \in V$, tồn tại duy nhất một đường đi trong P chứa v . Đường đi có thể bắt đầu và kết thúc ở bất cứ đâu, tính cả đường đi độ dài 0 (chỉ gồm một đỉnh). Bài toán đặt ra là tìm *phủ đường tối thiểu* (*minimum path cover*): Phủ đường gồm ít đường đi nhất.

Gọi n là số đỉnh của đồ thị, ta đánh số các đỉnh thuộc V từ 1 tới n .

Xây dựng đồ thị hai phía $G' = (X \cup Y, E')$ trong đó:

$$X = \{x_1, x_2, \dots, x_n\}; Y = \{y_1, y_2, \dots, y_n\}.$$

Tập cạnh E' được xây dựng như sau: Với mỗi cung $(i, j) \in E$, ta thêm vào một cạnh $(x_i, y_j) \in E'$ (hình vẽ).



Bài toán tìm phủ đường tối thiểu trên đồ thị có hướng, không có chu trình có thể quy về bài toán bộ ghép cực đại trên đồ thị hai phía

Gọi M là một bộ ghép trên G' . Khởi tạo P là tập n đường đi, mỗi đường đi chỉ gồm một đỉnh trong G , khi đó P là một phủ đường. Xét lần lượt các cạnh của bộ ghép, mỗi khi xét tới cạnh (x_i, y_j) ta đặt cạnh (i, j) nối hai đường đi trong P thành một đường,...

Khi thuật toán kết thúc, P vẫn là một phủ đường.

- a) Hãy chứng minh tính bất biến vòng lặp: Tại mỗi bước khi xét tới cạnh $(x_i, y_j) \in M$, cạnh $(i, j) \in E$ chắc chắn sẽ nối hai đường đi trong P : một đường đi kết thúc ở i và một đường đi khác bắt đầu ở j . Từ đó chỉ ra tính đúng đắn của thuật toán.

Gợi ý: Mỗi khi xét tới cạnh $(x_i, y_j) \in M$ và đặt cạnh (i, j) nối hai đường đi của P thành một đường thì $|P|$ giảm 1. Vậy khi thuật toán trên kết thúc, $|P| = n - |M|$, tức là muốn $|P| \rightarrow \min$ thì $|M| \rightarrow \max$.

- b) Hãy viết chương trình tìm phủ đường cực tiểu trên đồ thị có hướng không có chu trình.
- c) Hãy cho ví dụ để thấy rằng thuật toán trên không đúng trong trường hợp G có chu trình.
- d) Chứng minh rằng nếu tìm được thuật toán giải bài toán tìm phủ đường cực tiểu trên đồ thị tổng quát trong thời gian đa thức thì có thể tìm được đường đi Hamilton trên đồ thị đó (nếu có) trong thời gian đa thức (lí thuyết về độ phức tạp tính toán đã chứng minh được rằng trên đồ thị tổng quát, bài toán tìm đường đi Hamilton là NP-đầy đủ và bài toán tìm phủ đường cực tiểu là NP-khó. Có nghĩa là một thuật toán với độ phức tạp đa thức để giải quyết bài toán phủ đường cực tiểu trên đồ thị tổng quát sẽ là một phát minh lớn và đáng ngạc nhiên).

7.41. Hãy tự tìm hiểu về thuật toán Hopcroft-Karp. Cài đặt và so sánh tốc độ thực tế với thuật toán trong bài.

7.42. *Bộ ghép cực đại trên đồ thị chính quy hai phía*

Một đồ thị vô hướng $G = (V, E)$ gọi là *đồ thị chính quy bậc k* (k -regular graph) nếu bậc của mọi đỉnh đều bằng k . Đồ thị chính quy bậc 0 là đồ thị không có cạnh nào, đồ thị chính quy bậc 1 là đồ thị mà các cạnh tạo thành bộ ghép đầy đủ, đồ thị chính quy bậc 2 có các thành phần liên thông là các chu trình đơn.

- a) Chứng minh rằng đồ thị hai phía $G = (X \cup Y, E)$ là đồ thị chính quy thì $|X| = |Y|$.

- cạnh
- một
- chỉ ra
- ường
- n kết
- ướng
- hợp
- ròng
- lược
- thức
- đô thị
- tìm
- phúc
- tổng
- độ
- ular
- thị
- ành
- các
- thì
- b) Chứng minh rằng luôn tồn tại bộ ghép đầy đủ trên đồ thị hai phía chính quy bậc k ($k \geq 1$).
- c) Hãy xây dựng thuật toán có độ phức tạp $O(|E| \log |E|)$ để tìm một bộ ghép đầy đủ trên đồ thị chính quy bậc $k \geq 1$.

BÀI TẬP BỔ SUNG

7.43. Hệ ràng buộc

Cho n biến số nguyên v_1, v_2, \dots, v_n và một tập m ràng buộc. Mỗi ràng buộc được biểu diễn bởi ba số nguyên i, j, c_{ij} có dạng: $v_j - v_i \leq c_{ij}$ ($c_{ij} \in \mathbb{Z}$).

Hãy tìm cách gán giá trị nguyên nằm trong phạm vi $[a, b]$ cho các biến v_1, v_2, \dots, v_n để thỏa mãn tất cả m ràng buộc đã cho.

Input: Tệp văn bản SDC.INP bao gồm:

- Dòng 1 chứa bốn số nguyên dương n, m, a, b với $n \leq 1000, m \leq 10000, a < b \leq 10^6$;
- Mỗi dòng trong m dòng tiếp theo chứa ba số nguyên i, j, c_{ij} tương ứng với một ràng buộc ($1 \leq i, j \leq n; |c_{ij}| \leq 10^6$).

Output: Tệp văn bản SDC.OUT bao gồm:

- Dòng 1 ghi từ YES nếu có phương án thực hiện, ghi từ NO nếu không có phương án.
- Trong trường hợp có phương án thực hiện, dòng 2 ghi n giá trị của v_1, v_2, \dots, v_n tìm được.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

SDC.INP	SDC.OUT
3 3 1 4	YES
1 2 5	1 4 2
2 3 -2	
3 1 -1	

7.44. Miền hai màu

Cho một bảng kích thước $m \times n$ được chia thành lưới ô vuông đơn vị. Các hàng của bảng được đánh số từ 1 tới m , từ trên xuống dưới và các cột của bảng được đánh số từ 1 tới n , từ trái qua phải. Ô giao của hàng i và cột j được gọi là ô (i, j) và ô đó cần phải tô bằng một màu có mã màu là một số nguyên a_{ij} , hai màu khác nhau có mã khác nhau.

Một tập các ô của bảng được gọi là một miền nếu giữa hai ô bất kì của miền đó, ta có thể đi từ ô này sang ô kia bằng các phép di chuyển qua các ô kè cạnh.

Yêu cầu: Tìm một miền gồm nhiều ô nhất sao cho có thể tô màu các ô của miền bằng không quá hai màu.

Input: Tệp văn bản COLORING.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương $m, n \leq 1000$;
- Dòng thứ i trong m dòng tiếp theo chứa n số nguyên dương, số thứ j là a_{ij} ($a_{ij} \leq 10^6$).

Output: Tệp văn bản COLORING.OUT gồm một số nguyên duy nhất là số ô của miền tìm được.

Ví dụ:

COLORING.INP	COLORING.OUT
5 6	
1 1 1 1 3 1	
2 3 4 5 4 1	
1 2 1 2 5 1	
6 7 8 1 6 2	
1 1 1 1 7 2	
	14

7.45. Du lịch nhiều tour nhất

Một khu thăng cảnh gồm n điểm tham quan được đánh số từ 1 tới n và m đường đi hai chiều. Mỗi đường đi nối hai địa điểm khác nhau trong số n địa điểm đã cho và giữa hai địa điểm bất kì có nhiều nhất một đường đi nối chúng. Một tour du lịch là một hành trình xuất phát từ một địa điểm, đi thăm ít nhất hai địa điểm khác và quay trở về điểm xuất phát. Ngoại trừ địa điểm xuất phát, không địa điểm nào được thăm tới hai lần.

Yêu cầu: Hãy tìm một số tour du lịch sao cho mỗi tour du lịch tìm được đều có một đoạn đường riêng hoàn toàn không có mặt trong các tour du lịch còn lại.

Input: Tệp văn bản TOURS.INP bao gồm:

- Dòng 1: Ghi hai số n, m ($1 \leq n, m \leq 20000$);
- Mỗi dòng trong m dòng tiếp theo ghi hai số nguyên dương x, y tương ứng với một đường đi trực tiếp nối địa điểm x với địa điểm y .

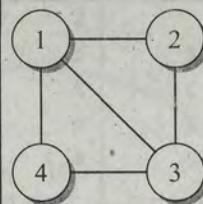
Output: Tệp văn bản TOURS.OUT bao gồm:

- Dòng 1: Ghi số k là số tour du lịch tìm được;
- Dòng thứ i trong k dòng tiếp theo mô tả tour du lịch thứ i : bắt đầu là địa điểm xuất phát, tiếp theo là danh sách các địa điểm sẽ đi tiếp theo thứ tự trong hành trình, kết thúc là địa điểm xuất phát.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

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



7.46. Chuyến đi ngắn nhất

Làng của Buba là nơi nổi tiếng với nhiều danh lam thắng cảnh, thu hút hàng nghìn khách du lịch từ nhiều nơi trên thế giới. Làng có n địa điểm du lịch (đánh số từ 1 đến n) và $n - 1$ con đường độ dài 1, nối các cặp địa điểm. Hai địa điểm bất kì đều có thể đi đến nhau qua các con đường này.

Có n khách du lịch đứng tại địa điểm 1, những du khách được đánh số từ 1 tới n . Du khách thứ nhất muốn thăm tất cả các địa điểm, du khách thứ hai muốn thăm tất cả các địa điểm có số thứ tự chia hết cho 2, du khách thứ ba muốn thăm tất cả các địa điểm có số thứ tự chia hết cho 3, ... Cụ thể là du khách thứ i muốn thăm tất cả các địa điểm có số thứ tự chia hết cho i .

Yêu cầu: Công ty du lịch muốn lập hành trình cho từng du khách bắt đầu từ địa điểm 1 đi thăm các địa điểm khác rồi quay trở về địa điểm 1 sao cho mỗi du khách được đi qua tất cả các địa điểm anh ta muốn thăm (hành trình có thể qua những địa điểm khác nữa). Bạn cần cho biết độ dài hành trình ngắn nhất của mỗi du khách thỏa mãn yêu cầu trên.

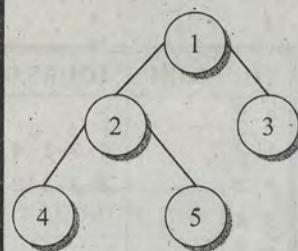
Input: Tệp văn bản TRIP.INP bao gồm:

- Dòng 1: Chứa số nguyên dương n ($n \leq 10^5$).
- Mỗi dòng trong $n - 1$ dòng tiếp theo gồm hai số nguyên u, v cách nhau ít nhất một dấu cách mô tả một con đường nối địa điểm u tới địa điểm v .

Output: Tệp văn bản TRIP.OUT gồm n dòng, dòng thứ i ghi một số nguyên duy nhất là độ dài quãng đường ngắn nhất mà du khách thứ i phải đi.

Ví dụ:

TRIP.INP	TRIP.OUT
5	8
1 2	4
1 3	2
4 2	4
5 2	4



Giải thích:

Du khách 1 đi theo hành trình $1 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 2 \rightarrow 1$;

Du khách 2 và 4 đi theo hành trình $1 \rightarrow 2 \rightarrow 4 \rightarrow 2 \rightarrow 1$.

7.47. Du lịch lữ hành

Bản đồ một khu du lịch gồm n địa điểm và m đường đi hai chiều. Giữa hai địa điểm có thể có nhiều đường đi nối trực tiếp giữa chúng. Một khách du lịch lữ hành muốn đi bộ từ địa điểm 1 tới địa điểm n rồi quay trở lại địa điểm 1 bằng một hành trình thỏa mãn các yêu cầu sau:

- Chỉ đi trên các đường đi trong m đường đã cho;
- Không có đường đi nào qua nhiều hơn một lần trong hành trình;

- Độ dài hành trình (tính bằng tổng độ dài các đường đi trên hành trình) là nhỏ nhất có thể.

Hãy chỉ ra một hành trình thoả mãn các yêu cầu trên hoặc cho biết rằng yêu cầu của du khách là không thể thực hiện được.

Input: Tệp văn bản WALK.INP bao gồm:

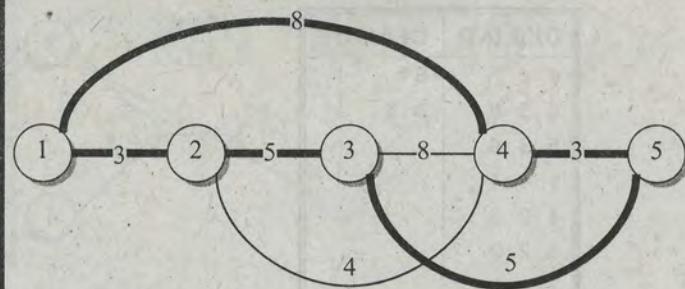
- Dòng 1: Chứa hai số nguyên dương n, m ($n \geq 2; n, m \leq 10^5$);
- Dòng thứ i trong m dòng tiếp theo chứa ba số nguyên dương u_i, v_i, w_i tương ứng với một đường đi nối địa điểm u_i với địa điểm v_i và độ dài đường đi đó là w_i . Độ dài đường đi được đo bằng micro mét (μm), không có đường đi nào dài hơn 2 km.

Output: Tệp văn bản WALK.OUT bao gồm:

- Dòng 1: Ghi độ dài hành trình, trong trường hợp không tồn tại hành trình thoả mãn các yêu cầu thì dòng này ghi số -1 ;
- Nếu tồn tại hành trình, dòng 2 ghi các địa điểm theo đúng thứ tự trên hành trình tìm được, bắt đầu và kết thúc ở địa điểm 1.

Ví dụ:

WALKINP	WALK.OUT
5 7	24
1 2 3	1 2 3 5 4 1
2 4 8	
2 3 5	
2 4 4	
3 5 5	
4 3 8	
4 5 3	



7.48. Chống buôn lậu ma tuý

Một mạng lưới giao thông gồm n thành phố đánh số từ 1 tới n và m con đường hai chiều đánh số từ 1 tới m . Con đường thứ i nối từ thành phố u_i tới thành phố v_i và cho phép đi lại theo cả hai chiều.

Tổ chức DEA muốn ngăn chặn những vụ vận chuyển ma tuý từ thành phố 1 tới thành phố n . Họ cần phân bố các nhân viên của mình trên những con đường để vây bắt bọn buôn lậu ma tuý có vũ trang. Theo những tính toán

thì để tuần tra và ngăn chặn bọn buôn ma tuý đi lại trên con đường thứ i , DEA cần bố trí c_i nhân viên trên con đường này.

Yêu cầu: Hãy giúp tổ chức DEA lên kế hoạch phân bổ các nhân viên của mình tuần tra trên một số tuyến đường, sao cho muốn đi từ 1 tới n chắc chắn phải đi qua ít nhất một tuyến đường được tuần tra và số nhân viên DEA cần huy động là ít nhất có thể.

Input: Tệp văn bản DEA.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương n, m ($n \leq 10^3; m \leq 10^4$);
- Dòng thứ i trong m dòng tiếp theo chứa ba số nguyên dương u_i, v_i, c_i ($c_i \leq 10^9$).

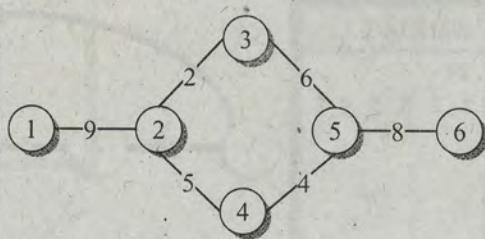
Output: Tệp văn bản DEA.OUT bao gồm:

- Dòng 1 ghi số nhân viên cần huy động;
- Dòng 2 ghi chỉ số các con đường được chọn để tuần tra.

Các số trên một dòng của tệp Input và Output được ghi cách nhau ít nhất một dấu cách.

Ví dụ:

DEA.INP	DEA.OUT
6 6	6
6 5 8	3 5
5 3 6	
5 4 4	
4 2 5	
3 2 2	
2 1 9	



7.49. Đạo phố

Giáo sư X cảm thấy mệt mỏi với công việc giảng dạy và nghiên cứu nên ông quyết định xác xe đi dạo quanh các con đường trong thành phố để thay đổi không khí. Có n địa điểm đánh số từ 1 tới n và m con đường *một chiều* đánh số từ 1 tới m . Con đường thứ i cho phép đi từ địa điểm u_i tới địa điểm v_i và có độ dài w_i . Hệ thống đường cho phép đi lại giữa hai địa điểm bất kì.

thứ i ,
en của
chắc
viên
 v_i, c_i

Giáo sư X xuất phát từ trường nằm tại địa điểm 1. Ông muốn đi qua tất cả m con đường rồi sau đó quay trở về trường. Ông có thể đi qua một con đường nhiều lần nhưng buộc phải đi theo chiều đã định của những con đường, bởi nếu đi ngược chiều thì ông sẽ được hưởng vài giờ nghỉ bất đắc dĩ tại trụ sở cảnh sát giao thông.

Yêu cầu: Tìm hành trình ngắn nhất cho giáo sư X thoả mãn yêu cầu trên.

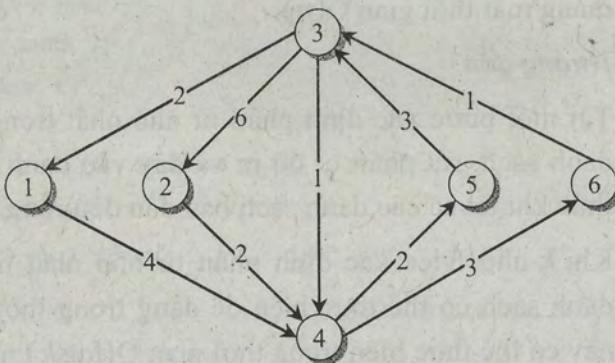
Input: Tệp văn bản DCPP.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương n, m ($n \leq 10^3; m \leq 10^4$);
- Dòng thứ i trong m dòng tiếp theo chứa ba số nguyên dương u_i, v_i, w_i ($w_i \leq 10^6$).

Output: Tệp văn bản DCPP.OUT gồm một số nguyên duy nhất là độ dài hành trình tìm được.

Ví dụ:

DCPP.INP	DCPP.OUT
6 9	28
1 4 4	
2 4 2	
3 1 2	
3 2 6	
3 4 1	
4 5 2	
4 6 3	
5 3 3	
6 3 1	



Giải thích: Hành trình cần tìm là

$$1 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 1.$$

Phần II HƯỚNG DẪN - GIẢI BÀI TẬP

CHUYÊN ĐỀ 6

6.1. Hướng dẫn

Trên danh sách gồm n phần tử có thứ tự tăng dần;

Tìm kiếm: Trên mảng có thể dùng thuật toán tìm kiếm nhị phân, trên danh sách mốc nối buộc phải sử dụng thuật toán tìm kiếm tuần tự;

Chèn/xoá: Áp dụng thuật toán tìm kiếm để xác định vị trí chèn/xoá, việc chèn trên danh sách mốc nối được thực hiện trong thời gian $O(1)$, trên mảng mất thời gian $O(n)$.

6.2. Hướng dẫn

Tại mỗi bước xác định phần tử nhỏ nhất trong các phần tử đứng đầu các danh sách, rút phần tử đó ra và đẩy vào danh sách kết quả, thuật toán kết thúc khi tất cả các danh sách ban đầu đều rỗng.

Khi k nhỏ, việc xác định phần tử nhỏ nhất trong các phần tử đứng đầu danh sách có thể thực hiện dễ dàng trong thời gian $O(k)$. Khi k lớn, việc này có thể thực hiện trong thời gian $O(\log k)$ nhờ cấu trúc dữ liệu Heap.

6.3. Hướng dẫn

Cộng hai đa thức coi như trộn hai danh sách giảm dần để được một danh sách giảm dần. Giả sử ta có hai đa thức:

Tại mỗi bước ta xét hai nút đầu hai danh sách:

- Nếu hai nút chứa hai số mũ khác nhau, nút mang số mũ lớn hơn sẽ được lấy ra đưa vào danh sách kết quả.
- Nếu hai nút chứa hai số mũ bằng nhau (chẳng hạn bằng b) và hệ số lần lượt là a và a' , cả hai nút đều được lấy ra, tạo nút mới chứa hệ số là $a + a'$ và số mũ là b đưa vào danh sách kết quả.

6.4. Thuật toán

Giả sử ta biểu diễn các chữ số trong một danh sách mốc nối đơn, các nút có kiểu *TNode*; *PNode* là kiểu con trỏ trỏ tới nút. Trong mỗi nút có chứa trường *value* là chữ số tương ứng và trường *link* trỏ sang nút kế tiếp. Đầu danh sách (*head*) là nút chứa chữ số a_n . Khi đó ta có thể mô tả phép “cộng 1” như sau:

```

function Add1(P: PNode): Integer;
var
    temp: Integer;
begin
    if P^.link <> nil then
        Result := Add1(P^.link)
    else
        Result := 1;
        temp := P^.value + Result;
        P^.value := temp mod 2;
        Result := temp div 2;
    end;
begin
    if Add1(head) = 1 then
        «Bổ sung nút chứa chữ số 1 vào đầu danh sách»;
end.
```

6.5. Hướng dẫn

Để cài đặt hàng đợi hai đầu, cách tổng quát nhất là sử dụng danh sách nối kép.

Trong những trường hợp cụ thể (biết trước số phần tử tối đa, số thao tác tối đa), có thể sử dụng mảng.

6.7. Chứng minh

Nếu x và y có quan hệ tiền bối-hậu duệ, chỉ mệnh đề iii) hoặc iv) là đúng.

Nếu x và y không có quan hệ tiền bối-hậu duệ, gọi a là tiền bối chung thấp nhất của x và y (a là nút sâu nhất mà nhánh gốc a chứa cả x và y), rõ ràng $a \neq x$ và $a \neq y$ vì x và y không có quan hệ tiền bối-hậu duệ.

Đi từ x lên a theo liên kết nút cha, giả sử đường đi qua x' trước khi tới a .

Tương tự như vậy đi từ y lên a theo liên kết nút cha, giả sử đường đi qua y' trước khi tới a .

Khi đó x' và y' là nút con của a , mặt khác $x' \neq y'$ vì nếu $x' = y' = b$ thì nút b cũng là tiền bối chung của x và y mà lại có độ sâu lớn hơn nút a .

Như vậy nếu x' có thứ tự nút con nhỏ hơn y' , ta có x nằm bên trái y , ngược lại nếu x' có thứ tự nút con lớn hơn y' , ta có x nằm bên phải y .

6.8. Cách giải

Nếu x là tiền bối của y , $\text{Preorder}[x] < \text{Preorder}[y]$

và $\text{Postorder}[x] > \text{Postorder}[y]$,

tương tự trong trường hợp y là tiền bối của x .

Nếu x và y không có quan hệ tiền bối-hậu duệ, ta dựa vào ngay kết quả của bài 6.7: Hoặc x nằm bên trái y hoặc x nằm bên phải y :

- Nếu x nằm bên trái y , x sẽ đứng trước y cả trong ba loại thứ tự Preorder, Inorder và Postorder.
- Nếu x nằm bên phải y , x sẽ đứng sau y cả trong ba loại thứ tự Preorder, Inorder và Postorder.

Như vậy, ta có thuật toán:

```
If (Preorder[x] < Preorder[y]) = (Inorder[x] < Inorder[y])
    = (Postorder[x] < Postorder[y])
Then <<x và y không có quan hệ tiền bối-hậu duệ>>
Else
    If Preorder[x] < Preorder[y]
    Then
        <<x là tiền bối của y>>
    Else
        <<y là tiền bối của x>>
```

6.9. Chứng minh

Kí hiệu tổng bậc của các nút trên cây T là $\text{sumdeg}(T)$ và số nút trên cây T là $|T|$, ta chứng minh một kết quả tổng quát:

Xét cây tổng quát gồm n nút, khi đó tổng bậc các nút trên cây bằng $n - 1$.

$$\text{sumdeg}(T) = |T| - 1.$$

Phép chứng minh được thực hiện bằng quy nạp.

Rõ ràng điều này đúng với cây chỉ gồm 1 nút (bậc 0).

Nếu cây có nhiều hơn 1 nút, gọi r là gốc cây và các cây con của gốc r là T_1, T_2, \dots, T_k . Theo giả thiết quy nạp, ta có

$$\text{sumdeg}(T_i) = |T_i| - 1, \forall i = 1, 2, \dots, k.$$

Suy ra:

$$\begin{aligned}\text{sumdeg}(T) &= k + \sum_{i=1}^k \text{sumdeg}(T_i) \\ &= k + \sum_{i=1}^k (|T_i| - 1) \\ &= \sum_{i=1}^k |T_i| = n - 1.\end{aligned}$$

Trong trường hợp T là cây nhị phân, bậc của mỗi đỉnh chỉ có thể là 0, 1 hoặc 2. Các đỉnh bậc 0 là nút lá.

Nếu T là p lá và q nút bậc 2, số nút bậc 1 còn lại là $n - p - q$. Khi đó

$$\text{sumdeg}(T) = p \times 0 + (n - p - q) \times 1 + q \times 2 = n - p + q.$$

Mặc khác, $\text{sumdeg}(T) = n - 1$ tức là $n - p + q = n - 1$ hay $p - q = 1$.

Nghĩa là số lá nhiều hơn số nút bậc 2 đúng một đơn vị.

6.10. Hướng dẫn

Giả sử ta có các nút với thứ tự duyệt trước và giữa.

Nút x có $\text{Preorder}[x]$ nhỏ nhất chắc chắn phải là gốc cây và không thể có lựa chọn nào khác.

Những nút có $\text{Inorder}[\cdot] < \text{Inorder}[x]$ chắc chắn nằm trong nhánh con trái và những nút có $\text{Inorder}[\cdot] > \text{Inorder}[x]$ chắc chắn nằm trong nhánh con phải. Việc khôi phục nhánh con trái và nhánh con phải được thực hiện

tương tự như trên (lấy nút có *Preorder*[.] nhỏ nhất làm gốc và tách danh sách làm đôi dựa trên thứ tự *Inorder*[.]).

6.11. Hướng dẫn

Ví dụ đơn giản nhất là với cây có hai nút x, y . Ta cho y làm con trái của x sẽ thu được một cây mà *Preorder*[.] cũng như *Postorder*[.] không thay đổi nếu cho y làm con phải của x .

6.13. Xây dựng thuật toán và cài đặt chương trình

Ta coi

- Phép *not* tương đương với phép lấy số đối;
- Phép *and* tương đương với phép nhân;
- Phép *or/xor* tương đương với phép cộng.

Và thực thiện thuật toán chuyển đổi RPN như biểu thức số học. Lưu ý là phép *not* chỉ có một toán hạng.

Chương trình

```
{$MODE OBJFPC}
program ConvertInfixToRPN;
uses SysUtils;
type
  TStackNode = record
    value: AnsiString;
    link: Pointer;
  end;
  PStackNode = ^TStackNode;
var
  Infix: AnsiString;
  top: PStackNode;
procedure Push(const c: AnsiString);
var
  p: PStackNode;
begin
  New(p);
  p^.value := c;
  p^.link := top;
  top := p;
```

```

ch danh
ii của x
ng thay
ru ý là

end;
function Pop: AnsiString;
var
  p: FStackNode;
begin
  Result := top^.value;
  p := top^.link;
  Dispose(top);
  top := p;
end;
function Get: AnsiString;
begin
  Result := top^.value;
end;
function Priority(const c: AnsiString): Integer;
begin
  if SameText(c, 'not') then Exit(3);
  if SameText(c, 'and') then Exit(2);
  if SameText(c, 'or') or SameText(c, 'xor') then Exit(1);
  Result := 0;
end;
procedure ProcessToken(const token: AnsiString);
var
  x: AnsiString;
begin
  if token = '(' then
    begin
      Push(token);
      Exit;
    end;
  if token = ')' then
    begin
      repeat
        x := Pop;
        if x <> '(' then Write(x, ' ')
        else Break;
      until False;
      Exit;
    end;

```

```

if SameText(token, 'not') or SameText(token, 'and')
    or SameText(token, 'or') or SameText(token, 'xor')
then
    begin
        while (top <> nil) and
            (Priority(token) <= Priority(Get)) do
                Write(Pop, ' ');
                Push(token);
                Exit;
            end;
        Write(token, ' ');
    end;
procedure Parsing;
const
    Operators = ['(', ')', '+', '-', '*', '/'];
var
    i, j: Integer;
begin
    j := 0;
    for i := 1 to Length(Inf) do
        if Inf[i] in ['(', ')', ' '] then
            begin
                if i > j + 1 then
                    ProcessToken(Copy(Inf, j + 1, i - j - 1));
                if Inf[i] in ['(', ')'] then
                    ProcessToken(Inf[i]);
                j := i;
            end;
        while top <> nil do
            Write(Pop, ' ');
            WriteLn;
    end;
    begin
        ReadLn(Inf);
        Inf := Inf + ' ';
        top := nil;
        Parsing;
    end.

```

6.15. Thuật toán

Khởi tạo stack rỗng, đọc ngược xâu từ phải qua trái, đọc được kí tự nào tạo ra một nút x chứa kí tự đó, sau đó:

- Nếu kí tự là B hoặc W: Đẩy nút chứa vào ngăn xếp;
- Nếu gặp kí tự &: Lấy trong ngăn xếp ra bốn nút P, Q, R, S cho làm con của x theo đúng thứ tự lấy ra, xong đẩy x vào ngăn xếp.

Kết thúc thuật toán, trong ngăn xếp chỉ còn đúng một nút duy nhất là gốc của một cây từ phân. Cây từ phân này gồm n nút với n là số kí tự trong xâu.

Thuật toán dựng cây có độ phức tạp $O(n)$.

Mỗi nút trên cây quản lí một phần ảnh vuông. Trong nút ta lưu thêm thông tin: Toạ độ góc trên trái của vùng quản lí, điều này có thể thực hiện bằng một phép duyệt cây $O(n)$.

Trước tiên ta lập hàm $\text{Find}(x, c)$: Tìm một điểm có màu khác c trong vùng quản lí của nút x :

- Nếu x chứa kí tự c : Anh không có điểm mang màu khác c ;
- Nếu x chứa kí tự \bar{c} (nếu c là B thì \bar{c} là W và ngược lại): Trả về toạ độ góc trên trái của vùng quản lí;
- Nếu x chứa kí tự &: Gọi đệ quy tìm trong bốn nhánh con.

Việc đối sánh hai ảnh quy về đối sánh hai cây. Giả sử hai cây đó có gốc r và gốc s .

- Nếu gốc r và gốc s có một gốc chứa kí tự B và một gốc chứa kí tự W: Trả về toạ độ góc trên trái là điểm có màu khác nhau;
- Nếu gốc r và gốc s đều chứa kí tự B hoặc đều chứa kí tự W: Hai ảnh hoàn toàn giống nhau;
- Nếu gốc r chứa kí tự B/W, gốc s chứa kí tự &: Quy về tìm điểm có màu khác với gốc r trong cây s bằng hàm Find ;
- Nếu cả hai gốc chứa kí tự &: Việc đối sánh được thực hiện đệ quy trên bốn cây con.

Thuật toán đối sánh có độ phức tạp $O(m + n)$ với m, n là độ dài hai xâu đầu vào.

6.16. Chứng minh

Nhận xét của giáo sư X là đúng, vì mỗi khi rẽ phải, các nút bên nhánh trái nhỏ hơn hoặc bằng khoá tìm kiếm, mỗi khi rẽ trái, các nút bên nhánh phải lớn hơn hoặc bằng khoá tìm kiếm.

6.17. Chứng minh

Trong dãy các lời gọi hàm Successor, mỗi cạnh của cây bị duyệt qua hai lần theo hai hướng ngược chiều. Cây có n nút thì có $n - 1$ cạnh nên thời gian thực hiện giải thuật là $O(n)$.

6.18. Chứng minh

Tương tự như bài 6.17, lời gọi Successor đầu tiên mất thời gian $O(h)$, $k - 1$ lời gọi tiếp theo mất thời gian $O(k)$.

6.19. Hướng dẫn

Việc cài đặt được thực hiện dựa trên thủ tục chèn và duyệt cây theo thứ tự giữa đã nêu trong phần lí thuyết. Chi phí duyệt cây theo thứ tự giữa là $O(n)$ với n là độ dài dãy khoá. Chi phí chèn n khoá vào cây là:

- $O(n \log n)$ trong trường hợp tốt nhất (cây nhị phân tạo thành là dạng cây nhị phân gần hoàn chỉnh);
- $O(n^2)$ trong trường hợp xấu nhất (cây nhị phân suy biến);
- $O(n \log n)$ trong trường hợp trung bình.

Việc chứng minh các kết quả trên tương tự như việc đánh giá độ phức tạp của giải thuật QuickSort.

6.20. Thuật toán

```
Result := nilT;  
x := root;  
while x <> nilT do  
    if x^.key <= k then  
        begin  
            Result := x;  
            x := x^.R;  
        end  
    else  
        x := x^.L;
```

6.21. Thuật toán

```
Result := nilT;  
x := root;  
while x <> nilT do  
    if x^.key >= k then  
        begin  
            Result := x;  
            x := x^.L;  
        end  
    else  
        x := x^.R;
```

6.22. Thuật toán

```
while x^.P <> nilT do UpTree(x);  
root := x;
```

6.23. Thuật toán

```
repeat  
    if x^.L <> nilT then y := x^.L  
    else y := x^.R;  
    if y = nilT then Break;  
    UpTree(y);  
until False;  
while root^.P <> nilT do root := root^.P
```

6.25. Chứng minh

Không làm mất tính tổng quát, giả sử BST chứa tập khoá $\{1, 2, \dots, n\}$.

Ta định nghĩa ba biến ngẫu nhiên để xác định chiều cao của BST gồm n nút.

- X_n : Chiều cao của BST.
- $Y_n = 2^{X_n}$.
- R_n là khoá chứa trong nút gốc.

Nếu $R_n = i$, nhánh con trái của BST chứa tập khoá $\{1, \dots, i - 1\}$ và nhánh con phải của BST chứa tập khoá $\{i + 1, \dots, n\}$. Tức là:

$$X_n = \max\{X_{i-1}, X_{n-i}\};$$

$$Y_n = 2 \cdot \max\{Y_{i-1}, Y_{n-i}\}.$$

Tại trường hợp $n = 1$, ta có $Y_1 = 1$, đê tiện, ta coi $Y_0 = 0$.

Xác suất đê $R_n = i$ là $\frac{1}{n}$, vì vậy

$$\begin{aligned} E[Y_n] &= \frac{1}{n} \sum_{i=1}^n 2 \cdot E[\max\{Y_{i-1}, Y_{n-i}\}] \\ &\leq \frac{1}{n} \sum_{i=1}^n 2E[Y_{i-1} + Y_{n-i}] \\ &\leq \frac{2}{n} \sum_{i=1}^n E[Y_{i-1}] + E[Y_{n-i}]. \end{aligned}$$

Bởi các hạng tử $E[Y_0], E[Y_1], \dots, E[Y_{n-1}]$ xuất hiện hai lần trong tổng nên:

$$E[Y_n] = \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i].$$

Ta sẽ chứng minh rằng với mọi số tự nhiên n ,

$$E[Y_n] \leq \frac{1}{4} \binom{n+3}{3}.$$

Thật vậy, với $n = 0$, ta có

$$0 = Y_0 = E[Y_0] \leq \frac{1}{4} \binom{3}{3} = \frac{1}{4}.$$

Với $n = 1$, ta có

$$1 = Y_1 = E[Y_1] \leq \frac{1}{4} \binom{4}{3} = 1.$$

Với $n \geq 2$

$$\begin{aligned} E[Y_n] &\leq \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i] \leq \frac{4}{n} \sum_{i=0}^{n-1} \frac{1}{4} \binom{i+3}{3} \\ &\leq \frac{1}{n} \sum_{i=0}^{n-1} \binom{i+3}{3} \leq \frac{1}{n} \cdot \binom{n+3}{4} \\ &= \frac{1}{n} \cdot \frac{(n+3)!}{4!(n-1)!} = \frac{1}{4} \cdot \frac{(n+3)!}{3!n!} = \frac{1}{4} \binom{n+3}{3}. \end{aligned}$$

Mặt khác hàm 2^x là một hàm lồi nên

$$\begin{aligned} 2^{E[X_n]} &\leq E[2^{X_n}] \\ &= E[Y_n] = \frac{1}{4} \binom{n+3}{3} = \frac{n^3 + 6n^2 + 11n + 6}{24}. \end{aligned}$$

Lấy lôgarit hai vế, ta có $E[X_n] = O(\lg n)$.

6.26. Chứng minh

- a) Không làm mất tính tổng quát giả sử tập khoá chứa trong cây là $\{1, 2, \dots, n\}$, $b[0] = 1$ ứng với cây không có nút nào.

Khi $n > 1$, nếu nút gốc cây chứa khoá k thì nhánh con trái là BST của $k - 1$ khoá $\{1, 2, \dots, k - 1\}$ và nhánh con phải là BST của $n - k$ khoá $\{k + 1, \dots, n\}$. Suy ra

$$b(n) = \sum_{k=1}^n b(k)b(n-k)$$

$$\text{hay } b(n) = \sum_{k=0}^{n-1} b_k b_{n-1-k}.$$

- b) Các kết quả còn lại suy ra từ lí thuyết tổ hợp về số Catalan, tóm tắt tại:

http://en.wikipedia.org/wiki/Catalan_number

6.27. Hướng dẫn

Với mỗi nút x của Treap, lưu trữ $c[x]$ là số nút trong nhánh gốc x . Để tìm thứ tự giữa của nút x . Lần theo đường đi từ x về gốc và dựa vào các giá trị $c[\cdot]$ để tìm thứ tự.

6.28. Hướng dẫn

Phép tách: Để tách Treap dựa vào khoá k , ta tìm nút x đầu tiên chứa khoá lớn hơn k và dùng phép UpTree đẩy x lên gốc, nhánh con trái của gốc bây giờ là Treap chứa các khoá nhỏ hơn hoặc bằng k . Tách rời nhánh con trái và kéo x xuống cho phù hợp với ràng buộc của Treap.

Phép hợp: Để nối hai Treap rời nhau T_1, T_2 , tìm nút cực phải x của T_1 dùng phép UpTree đẩy lên thành gốc, cho T_2 làm nhánh con phải và kéo x xuống cho phù hợp với ràng buộc của Treap.

6.29. Hướng dẫn

Tìm nút đầu tiên chứa khoá $k \geq a$ sau đó dùng hàm Successor để duyệt các nút nhỏ hơn hoặc bằng b . Bài tập 6.18 đã chỉ ra rằng thuật toán này có độ phức tạp $O(m + h)$ với h là chiều cao của cây.

Với các dạng cây cân bằng như cây AVL hay cây đỏ đen, $h = O(\lg n)$ và như vậy có thể thực hiện thuật toán này trong thời gian $O(m + \lg n)$.

Cách dễ hơn là sử dụng cây Splay.

6.30. Xây dựng thuật toán và cài đặt chương trình

a) Xây dựng dãy hoán vị x từ dãy nghịch thế t

Khởi tạo mảng $x[1..n]$ chưa điền phần tử nào, các vị trí chưa điền coi như “khoảng trống”;

Duyệt lần lượt t_1, t_2, \dots, t_n . Khi duyệt qua t_i , điền giá trị i vào “khoảng trống” thứ $i + 1$ tính từ đầu dãy x . Việc xác định khoảng trống thứ $i + 1$ có thể thực hiện trong thời gian $O(\lg n)$ bằng một cấu trúc dữ liệu truy vấn phạm vi.

b) Xây dựng dãy nghịch thế t từ dãy hoán vị x

Xét lần lượt các giá trị $1, 2, \dots, n$. Với mỗi giá trị k , xác định vị trí p của nó trong dãy x và đánh dấu lại vị trí p này. Khi đó t_k sẽ bằng số vị trí đứng trước vị trí p mà chưa bị đánh dấu. Sử dụng một cấu trúc dữ liệu truy vấn phạm vi, ta có thể đếm số vị trí chưa đánh dấu đứng trước vị trí p trong thời gian $O(\lg n)$.

Chương trình

```
{$MODE OBJFPC}
program InversionVector;
const
  InputFile  = 'IVECTOR.INP';
  OutputFile = 'IVECTOR.OUT';
  max = 100000;
type
  TArray = array[1..max] of Integer;
var
  bit: array[1..max] of Integer;
```

```

x, t, pos: TArray;
n: Integer;
fi, fo: TextFile;
procedure OpenFiles;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  AssignFile(fo, OutputFile);
  Rewrite(fo);
  ReadLn(fi, n);
end;
procedure CloseFiles;
begin
  Close(fi);
  Close(fo);
end;
procedure BuildTree;
var
  i: Integer;
begin
  for i := 1 to n do
    bit[i] := i and -i;
end;
function CountAndMark(i: Integer): Integer;
var
  x: Integer;
begin
  x := Pred(i);
  Result := 0;
  while x > 0 do
    begin
      Inc(Result, bit[x]);
      x := x and Pred(x);
    end;
  x := i;
  while x <= n do
    begin
      Dec(bit[x]);
      Inc(x, x and -x);
    end;
end;

```

```

    end;
end;

procedure FindInversionVector;
var
  i, j: Integer;
begin
  for i := 1 to n do Read(fi, x[i]);
  ReadLn(fi);
  BuildTree;
  for i := 1 to n do pos[x[i]] := i;
  for i := 1 to n do
    begin
      j := pos[i];
      t[i] := CountAndMark(j);
    end;
  for i := 1 to n do Write(fo, t[i], ' ');
  WriteLn(fo);
end;

function LocateAndMark(i: Integer): Integer;
var
  x, next, mask: Integer;
begin
  mask := 1;
  while mask shl 1 <= n do mask := mask shl 1;
  x := 0;
  while mask <> 0 do
    begin
      next := x + mask;
      if next <= n then
        begin
          if i = bit[next] then
            Result := next;
          if i > bit[next] then
            begin
              x := next;
              Dec(i, bit[next]);
            end;
        end;
      mask := mask shr 1;
    end;
end;

```

```

    end;
    x := Result;
  while x <= n do
    begin
      Dec(bit[x]);
      Inc(x, x and -x);
    end;
  end;
procedure FindPermutationVector;
var
  i, j: Integer;
begin
  for i := 1 to n do Read(fi, t[i]);
  FillChar(x, sizeof(x), 0);
  BuildTree;
  for i := 1 to n do
    begin
      j := LocateAndMark(t[i] + 1);
      x[j] := i;
    end;
  for i := 1 to n do Write(fo, x[i], ' ');
end;
begin
  OpenFiles;
  try
    FindInversionVector;
    FindPermutationVector;
  finally
    CloseFiles;
  end;
end.

```

6.31. Hướng dẫn

Sử dụng một đường thẳng nằm ngang quét từ trên xuống. Khi quét vào cạnh trên của một hình chữ nhật, ta bổ sung khoảng $[x_{min}, x_{max}]$ là hình chiếu của hình chữ nhật đó trên trục hoành vào tập S . Nếu quét qua cạnh dưới của hình chữ nhật, ta loại bỏ hình chiếu tương ứng của nó trên trục hoành khỏi tập S .

Trong quá trình đường quét di chuyển, qua mỗi đơn vị độ dài ta tính phần trực số bị các khoảng trong tập S phủ lên và cộng dồn vào kết quả.

Để thực hiện nhanh chóng thao tác bổ sung/loại bỏ khoảng cũng như tính độ dài phần trực số bị các khoảng thuộc S phủ lên, cần tổ chức tập S dưới dạng một cây quản lí phạm vi như cây chèo khoảng.

Các thao tác này mất thời gian $O(\lg n)$.

6.32. Hướng dẫn

Dánh số các đầu mút của các dây cung từ 1 tới $2n$ theo chiều kim đồng hồ.

Giả sử dây cung thứ i nối từ điểm i tới điểm $f(i)$.

Với hai dây cung i, j ($i < j$), chúng cắt nhau nếu $f(i) < f(j)$.

Với mỗi vị trí j , việc đếm số lượng các chỉ số i đứng trước j mà $f(i) < f(j)$ có thể thực hiện trong thời gian $O(\lg n)$ bằng các dạng cây quản lí phạm vi.

Độ phức tạp $O(n \lg n)$.

6.33. Hướng dẫn

Xếp các đoạn theo thứ tự tăng dần của b_i và duyệt các đoạn theo thứ tự đã sắp.

Với mỗi đoạn $[a_i, b_i]$ xét qua, dùng một cấu trúc dữ liệu truy vấn phạm vi để đếm xem có bao nhiêu điểm đã chọn nằm trong đoạn $[a_i, b_i]$, nếu chưa đủ c_i điểm thì bổ sung thêm những điểm có toạ độ lớn nhất cho đủ c_i điểm.

6.35. Thuật toán

Dùng cây biểu diễn danh sách thực hiện phép tráo và tìm ra trạng thái bộ bài sau x phép tráo trong thời gian $O(n + x \lg n)$.

Tìm dãy con tăng dài nhất của các số ghi trên bộ bài theo thứ tự sau x phép tráo.

Các lá bài không thuộc dãy con tăng dài nhất sẽ được rút ra chèn vào vị trí đúng của nó để khôi phục thứ tự $(1, 2, \dots, n)$.

Độ phức tạp $O((n + x) \lg n)$.

6.36. Xây dựng thuật toán và cài đặt chương trình

Khởi động một ngăn xếp rỗng, xét lần lượt các phần tử a_1, a_2, \dots, a_n . Khi xét một phần tử a_i , đọc phần tử ở đỉnh ngăn xếp (b) và phần tử tiếp theo trong đỉnh ngăn xếp (c). Chừng nào b nhỏ hơn cả a_i và c thì b sẽ được rút khỏi ngăn xếp bằng lệnh POP. Đến khi hai phần tử ở đỉnh ngăn xếp không thỏa mãn điều kiện này thì a_i được đẩy vào ngăn xếp. Kết thúc thuật toán, ngăn xếp chứa những phần tử còn lại của dãy.

Chương trình

```
{$MODE OBJFPC}
program DelNum;
const
  InputFile = 'DELNUM.INP';
  OutputFile = 'DELNUM.OUT';
  maxN = 100000;
var
  i, n, res, a, b, c: Integer;
  stack: array[1..maxN + 2] of Integer;
  top: Integer;
  f: TextFile;
begin
  AssignFile(f, InputFile); Reset(f);
  ReadLn(f, n); top := 2;
  stack[1] := Low(Integer);
  stack[2] := Low(Integer);
  for i := 1 to n do
    begin
      Read(f, a);
      repeat
        b := stack[top];
        c := stack[top - 1];
        if (b < c) and (b < a) then Dec(top)
        else Break;
      until False;
      Inc(top);
      stack[top] := a;
    end;
end;
```

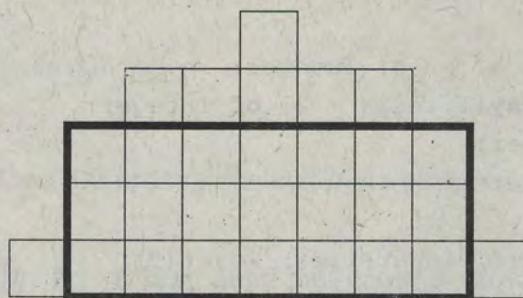
```

CloseFile(f);
res := top - 2;
AssignFile(f, OutputFile);
Rewrite(f);
Write(f, res);
CloseFile(f);
end.

```

6.37. Xây dựng thuật toán và cài đặt chương trình

Trước hết, ta tìm hình chữ nhật lớn nhất màu vàng (hình chữ nhật lớn nhất màu xanh sẽ được tìm bằng phương pháp tương tự). Để tìm hình chữ nhật lớn nhất màu vàng, ta phát biểu lại bài toán như sau: Cho n cột chữ nhật đặt liên tiếp có đáy nằm trên một đường thẳng nằm ngang, mỗi cột có chiều ngang 1 đơn vị, cột thứ i có độ cao h_i . Ta cần tìm hình chữ nhật lớn nhất nằm trong phần bị phủ bởi các cột đã cho



Với mỗi cột i ta xác định $left[i]$ là cột đứng trước cột i , có chiều cao nhỏ hơn h_i và gần cột i nhất. Tức là:

$$left[i] = \max\{j : j < i \text{ và } h_j < h_i\}.$$

Trong trường hợp không có cột nào đứng trước cột i mà có chiều cao nhỏ hơn h_i , ta coi $left[i] = 0$. Việc xác định các giá trị $left[1..n]$ có thể thực hiện bằng một thuật toán đơn giản: Khởi tạo một ngăn xếp rỗng dùng để chứa các cột và xét lần lượt các cột từ 1 tới n . Khi mỗi cột i được xét, phần tử tại đỉnh ngăn xếp sẽ lần lượt được lấy ra nếu chúng có chiều cao lớn hơn hoặc bằng h_i . Khi phần tử tại đỉnh ngăn xếp (cột j nào đó) có chiều cao nhỏ hơn h_i , ta đặt $left[i] := j$ rồi đẩy i vào ngăn xếp.

```

stack := Ø; //Ngăn xếp để chứa các cột
for i := 1 to n do

```

```

begin
    while (stack ≠ Ø) and (h[Get] ≥ h[i]) do
        //Cột ở đỉnh ngăn xếp không thấp hơn h[i] sẽ bị bỏ
        Pop;
        if stack = Ø then left[i] := 0 else left[i] := Get;
        Push(i)
end;

```

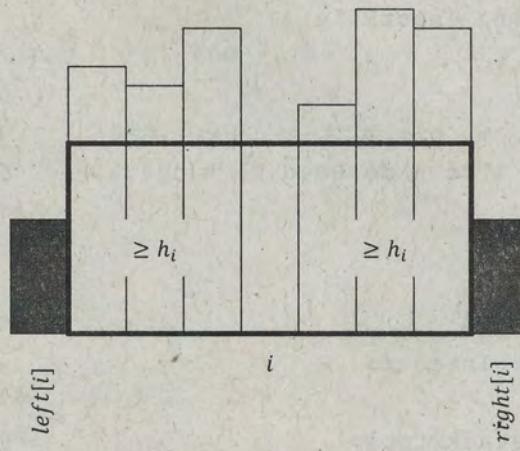
Dễ thấy tổng cộng có n thao tác Push được thực hiện nên thuật toán không thể thực hiện quá n thao tác Pop. Thời gian thực hiện giải thuật là $\Theta(n)$.

Tương tự như mảng $left[1..n]$, ta tính mảng $right[1..n]$ theo cách tương tự, ở đây $right[i]$ là cột đứng sau cột i , có chiều cao nhỏ hơn h_i và gần cột i nhất. Tức là:

$$right[i] = \min\{j: j > i \text{ và } h_j < h_i\}.$$

Trong trường hợp không có cột nào đứng sau cột i mà có chiều cao nhỏ hơn h_i , ta coi $right[i] = n + 1$.

Khi đã có mảng $left[1..n]$ và $right[1..n]$, xét từng cột i . Do $left[i]$ và $right[i]$ lần lượt là cột bên trái và cột bên phải gần nhất có chiều cao nhỏ hơn cột i , suy ra các cột từ chỉ số $left[i] + 1$ tới chỉ số $right[i] - 1$ đều có độ cao lớn hơn hoặc bằng h_i .



Vậy thì hình chữ nhật lớn nhất chứa trọn cột i sẽ có độ cao bằng h_i và phủ từ cột $left[i] + 1$ tới cột $right[i] - 1$. Diện tích hình chữ nhật này là $h_i \times (right[i] - left[i] + 1)$. Vấn đề còn lại là chọn cột i có diện tích

hình chữ nhật phủ cột đó là lớn nhất, hình chữ nhật này là hình chữ nhật cần tìm.

Chương trình

```
{$MODE OBJFPC}
program MaxRect;
const
  InputFile = 'RECT.INP';
  OutputFile = 'RECT.OUT';
  maxMN = 1000000;
var
  stack: array[1..maxMN] of Integer;
  top: Integer;
  h: array[1..maxMN] of Integer;
  left, right: array[1..maxMN] of Integer;
  m, n: Integer;
  x1, y1, x2, y2: Integer;
  resS: Int64;
procedure Enter;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, m, n);
    for i := 1 to n do Read(f, h[i]);
  finally
    CloseFile(f);
  end;
end;
function Get: Integer;
begin
  Result := stack[top];
end;
procedure Pop;
begin
  Dec(top);
end;
```

```

end;

procedure Push(i: Integer);
begin
  Inc(top);
  stack[top] := i;
end;

procedure DoAnalyse;
var
  i: Integer;
begin
  //Tính left[1..n]
  top := 0;
  for i := 1 to n do
    begin
      while (top <> 0) and (h[Get] >= h[i]) do
        Pop;
      if top = 0 then left[i] := 0
      else left[i] := Get;
      Push(i);
    end;
  //Tính right[1..n]
  top := 0;
  for i := n downto 1 do
    begin
      while (top <> 0) and (h[Get] >= h[i]) do
        Pop;
      if top = 0 then right[i] := n + 1
      else right[i] := Get;
      Push(i);
    end;
end;

procedure Solve;
var
  i: Integer;
  temp: Int64;
begin
  //Tìm hình chữ nhật vàng lớn nhất
  DoAnalyse;
  resS := -1;

```

```

for i := 1 to n do
begin
  temp := Int64(right[i] - left[i] - 1) * h[i];
  if resS < temp then
    begin
      resS := temp;
      x1 := 1;
      y1 := left[i] + 1;
      x2 := h[i];
      y2 := right[i] - 1;
    end;
end;

//Việc tìm hình chữ nhật xanh lớn nhất, làm tương tự
for i := 1 to n do h[i] := m - h[i];
DoAnalyse;
for i := 1 to n do
begin
  temp := Int64(right[i] - left[i] - 1) * h[i];
  if resS < temp then
    begin
      resS := temp;
      x1 := m - h[i] + 1;
      y1 := left[i] + 1;
      x2 := m;
      y2 := right[i] - 1;
    end;
end;
end;
procedure PrintResult;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    WriteLn(f, resS);
    WriteLn(f, x1, ' ', y1);
    WriteLn(f, x2, ' ', y2);
  end;
end;

```

```

    finally
        CloseFile(f);
    end;
end;
begin
    Enter;
    Solve;
    PrintResult;
end.

```

6.38. Xây dựng thuật toán và cài đặt chương trình

Mạng điện trong bài là hình ảnh của một đồ thị có hướng, đồ thị này được xây dựng theo một cách đặc biệt mà ta có thể tìm được tập độc lập cực đại (ĐLCD) bằng thuật toán tốt. Ở đây ta không thể sử dụng phương pháp xây dựng đồ thị một cách tường minh, bởi bài toán tìm tập ĐLCD là bài toán NP-đầy đủ, hiện tại chưa có thuật toán hiệu quả để giải quyết trong trường hợp tổng quát.

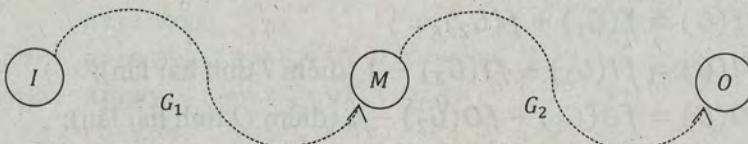
Biểu diễn mỗi đồ thị G (có đầu vào I và đầu ra O) bởi bốn số nguyên:

- $f(G)$: Lực lượng của tập ĐLCD không chứa cả I và O ;
- $fI(G)$: Lực lượng của tập ĐLCD có chứa I và không chứa O ;
- $fO(G)$: Lực lượng của tập ĐLCD không chứa I và có chứa O ;
- $fIO(G)$: Lực lượng của tập ĐLCD chứa cả I và O .

Với đồ thị G ta có

$$\begin{aligned} f(G) &= 0; \\ fI(G) &= 1; fO(G) = 1; \\ fIO(G) &= -\infty. \end{aligned}$$

Nếu $G = SG_1G_2$ là một mạng mắc nối tiếp của hai mạng G_1 và G_2 . Giả sử G có đầu vào I (là đầu vào G_1) và đầu ra O (là đầu ra G_2). Đầu ra G_1 và đầu vào G_2 chập vào nhau tại điểm M . Giả sử đã biết cách biểu diễn G_1 và G_2 , ta sẽ tìm cách biểu diễn G .



Theo định nghĩa $f(G)$ là lực lượng tập ĐLCD trên G không chứa cả I và O (kí hiệu là tập A).

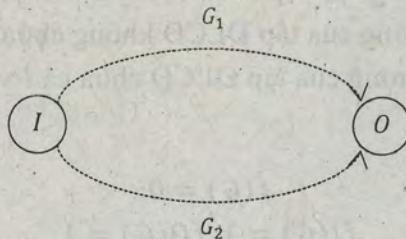
- Nếu $M \notin A$ thì A hạn chế trên G_1 sẽ có lực lượng bằng $f(G_1)$ và A hạn chế trên G_2 sẽ có lực lượng bằng $f(G_2)$. Suy ra $|A| = f(G_1) + f(G_2)$.
 - Nếu $M \in A$ thì A hạn chế trên G_1 sẽ có lực lượng bằng $fO(G_1)$ và A hạn chế trên G_2 sẽ có lực lượng bằng $fI(G_2)$.
- Suy ra $|A| = fO(G_1) + fI(G_2) - 1$ (phải trừ 1 do điểm M được tính hai lần).

Vậy thì $f(G) = \max\{f(G_1) + f(G_2), fO(G_1) + fI(G_2) - 1\}$.

Lập luận tương tự, ta tính ra được bốn số nguyên biểu diễn G bằng công thức truy hồi:

$$\begin{aligned} f(G) &= \max\{f(G_1) + f(G_2), fO(G_1) + fI(G_2) - 1\}; \\ fI(G) &= \max\{fI(G_1) + f(G_2), fIO(G_1) + fI(G_2) - 1\}; \\ fO(G) &= \max\{f(G_1) + fO(G_2), fO(G_1) + fIO(G_2) - 1\}; \\ fIO(G) &= \max\{fI(G_1) + fO(G_2), fIO(G_1) + fIO(G_2) - 1\}. \end{aligned}$$

Trường hợp mắc song song đơn giản hơn. Nếu $G = PG_1G_2$ là một mạng mắc song song của hai mạng G_1 và G_2 . Giả sử G có đầu vào I và đầu ra O .



Theo định nghĩa $f(G)$ là lực lượng tập ĐLCD trên G không chứa cả I và O (kí hiệu là tập B). Vì B hạn chế trên G_1 có lực lượng bằng $f(G_1)$ và B hạn chế trên G_2 có lực lượng bằng $f(G_2)$ nên ta có $|B| = f(G_1) + f(G_2)$. Lập luận tương tự, ta sẽ tính được bốn số nguyên biểu diễn G bằng công thức truy hồi:

$$\begin{aligned} f(G) &= f(G_1) + f(G_2); \\ fI(G) &= fI(G_1) + fI(G_2) - 1 \text{ (điểm } I \text{ tính hai lần)}; \\ fO(G) &= fO(G_1) + fO(G_2) - 1 \text{ (điểm } O \text{ tính hai lần)}; \\ fIO(G) &= fIO(G_1) + fIO(G_2) - 2 \text{ (điểm } I \text{ và } O \text{ tính hai lần)}. \end{aligned}$$

Ta đã xây dựng xong công thức truy hồi tính các tham số biểu diễn đồ thị lớn từ các tham số biểu diễn hai đồ thị thành phần.

Thuật toán còn lại khá đơn giản. Kí pháp biểu diễn đồ thị là kí pháp tiền tố. Để tính ra các tham số biểu diễn đồ thị đã cho, ta có thể áp dụng thuật toán giải mã kí pháp tiền tố:

Khởi động ngăn xếp rỗng để chứa các đồ thị (chú ý mỗi đồ thị chỉ là một bản ghi gồm bốn số nguyên). Đọc xâu biểu diễn từ cuối lên đầu (từ phải qua trái):

- Nếu đọc được kí tự g : Đẩy g vào ngăn xếp;
- Nếu đọc được kí tự S : Lấy từ định ngăn xếp ra hai đồ thị X và Y , tính $G = SXY$ và đẩy G vào ngăn xếp;
- Nếu đọc được kí tự P : Lấy từ định ngăn xếp ra hai đồ thị X và Y , tính $G = PXY$ và đẩy G vào ngăn xếp.

Kết thúc thuật toán ngăn xếp chỉ chứa đúng một bản ghi biểu diễn đồ thị đầu vào, việc cuối cùng là chọn tham số lớn nhất trong bốn tham số của bản ghi để in kết quả.

Chương trình

```
{$MODE OBJFPC}
{$INLINE ON}
program IndependentSet;
const
  InputFile = 'INDEP.INP';
  OutputFile = 'INDEP.OUT';
  maxN = Round(1E6);
  neginf = -maxN - 1;
type
  TGraph = record
    f, fi, fo, fio: Integer;
  end;
var
  s: AnsiString;
  stack: array[1..maxN] of Integer;
  graph: array[1..maxN] of TGraph;
  unitg: TGraph;
  top: Integer;
```

```

    res: Integer;
procedure Enter;
var
  f: TextFile;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, s);
  finally
    CloseFile(f);
  end;
end;
procedure Init;
begin
  top := 0;
  with unity do
    begin
      f := 0;
      fi := 1;
      fo := 1;
      fio := neginf;
    end;
end;
function Max(x, y: Integer): Integer; inline;
begin
  if x > y then Result := x else Result := y;
end;
function OS(const x, y: TGraph): TGraph;
begin
  with Result do
    begin
      f := Max(x.f + y.f, x.fo + y.fi - 1);
      fi := Max(x.fi + y.f, x.fio + y.fi - 1);
      fo := Max(x.f + y.fo, x.fo + y.fio - 1);
      fio := Max(x.fi + y.fo, x.fio + y.fio - 1);
    end;
end;
function OP(const x, y: TGraph): TGraph;

```

```

begin
  with Result do
    begin
      f := x.f + y.f;
      fi := x.fi + y.fi - 1;
      fo := x.fo + y.fo - 1;
      fio := x.fio + y.fio - 2;
      if fio < neginf then fio := neginf;
    end;
  end;
procedure Push(i: Integer); inline;
begin
  Inc(top);
  stack[top] := i;
end;
function Pop: Integer; inline;
begin
  Result := stack[top];
  Dec(top);
end;
procedure Solve;
var
  i: Integer;
begin
  for i := Length(s) downto 1 do
    begin
      case s[i] of
        'g': graph[i] := unitg;
        'S': graph[i] := OS(graph[pop], graph[pop]);
        'P': graph[i] := OP(graph[pop], graph[pop]);
      end;
      Push(i);
    end;
  with graph[1] do
    begin
      res := f;
      if fi > res then res := fi;
      if fo > res then res := fo;
      if fio > res then res := fio;
    end;
end;

```

```

    end;
end;

procedure PrintResult;
var
  f: TextFile;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    Write(f, res);
  finally
    CloseFile(f);
  end;
end;
begin
  Enter;
  Init;
  Solve;
  PrintResult;
end.

```

6.39. Xây dựng thuật toán và cài đặt chương trình

Gọi $f[i]$ là số nguyên dương nhỏ nhất tạo thành từ các chữ số thuộc S sao cho $f[i] \bmod n = i$. Ban đầu các giá trị $f[0..n-1]$ được đánh dấu là “chưa biết” và ta sẽ tính các giá trị này.

Với từng chữ số $d \neq 0$ của S xét theo thứ tự tăng dần, nếu $f[d \bmod n]$ chưa biết thì ta đặt $f[d \bmod n] = d$ và đẩy d vào một hàng đợi Q .

Thuật toán loang được thực hiện chừng nào hàng đợi $Q \neq \emptyset$. Tại mỗi bước, một giá trị $f[r]$ được rút khỏi hàng đợi, xét lần lượt từng chữ số $d \in S$ theo thứ tự tăng dần, nối d vào sau $f[r]$ thành chữ số hàng đơn vị để được một số (gọi là A), xác định số dư r' trong phép chia A cho n (lưu ý rằng $r' = (r * 10 + d) \bmod n$), nếu $f[r']$ chưa biết, ta cập nhật $f[r'] := A$ và đẩy $f[r']$ vào hàng đợi. Thuật toán kết thúc khi hàng đợi rỗng hoặc khi ta đã biết giá trị $f[0]$.

Tính đúng đắn của thuật toán có thể suy ra được từ tiến trình loang: Mỗi khi một giá trị $f[r]$ được đẩy vào hàng đợi, đó chắc chắn là số nhỏ nhất chia n dư r' .

Vấn đề còn lại cần giải quyết là việc lưu trữ các giá trị $f[.]$: Do số chữ số của $f[.]$ có thể rất lớn, việc lưu trữ các $f[.]$ trực tiếp cần phải dùng các kiểu dữ liệu mảng hoặc xâu, và như vậy có thể không đủ bộ nhớ để cài đặt hàng đợi.

Thay vì lưu trữ trực tiếp giá trị $f[r]$ ta sẽ lưu trữ ba thông tin:

- Số dư r khi chia $f[r]$ cho n ;
- Chữ số hàng đơn vị của $f[r]$ ($LastDigit[r]$);
- Số $prev[r]$ sao cho $f[r]$ được tạo thành từ $f[prev[r]]$ nối thêm $LastDigit[r]$ vào hàng đơn vị.

Ngoài ra có một mảng đánh dấu $unknown[0..n-1]$ ban đầu được khởi tạo bằng True, $unknown[r] = \text{True}$ có nghĩa là ta chưa xác định được giá trị $f[r]$.

Thuật toán loang khi đó sẽ được vận hành như sau:

Tại mỗi bước; một số r được rút khỏi hàng đợi, lần lượt các chữ số thuộc S được xét theo thứ tự tăng dần, khi xét tới chữ số d , ta tính

$$r' := (r \times 10 + d) \bmod n.$$

Nếu $unknown[r'] = \text{True}$, ta cập nhật lại $unknown[r'] := \text{False}$, đặt

$$LastDigit[r'] := d \text{ và } prev[r'] := r.$$

- Ngay khi ta có $unknown[0] = \text{False}$, thuật toán kết thúc ngay lập tức và chỉ cần truy vết theo các giá trị $prev[.]$ và $LastDigit[.]$ để tìm nghiệm.
- Nếu hàng đợi rỗng trong khi $unknown[0]$ vẫn là True, không tồn tại bộ số cần tìm theo yêu cầu đề bài.

Chương trình

```
{$MODE OBJFPC}
program LeastDenominator;
const
  InputFile = 'LM.INP';
```

```

OutputFile = 'LM.OUT';
maxN = 1000000;
type
TQueue = record
  items: array[1..maxN] of Integer;
  front, rear: Integer;
end;
TDigit = 0..10;
var
n: Integer;
S: set of TDigit;
Queue: TQueue;
unknown: array[0..maxN - 1] of Boolean;
LastDigit: array[0..maxN - 1] of TDigit;
prev: array[0..maxN - 1] of Integer;
ResDigits: array[1..maxN] of TDigit;
procedure Enter; //Nhập dữ liệu
var
f: TextFile;
c: Char;
begin
AssignFile(f, InputFile);
Reset(f);
try
  ReadLn(f, n);
  S := [];
  while not SeekEoln(f) do
    begin
      Read(f, c);
      Include(S, Ord(c) - Ord('0'));
    end;
  finally
    CloseFile(f);
  end;
end;
procedure Init; //Khởi tạo
var
d: TDigit;
r: Integer;

```

```

begin
  FillChar(unknown, n, True);
  with Queue do
    begin
      front := 1;
      rear := 0;
      for d in S do
        if d <> 0 then //Xét các chữ số khác 0 của S
          begin
            r := d mod n;
            if unknown[r] then
              begin
                Inc(rear);
                items[rear] := r;
                LastDigit[r] := d;
                prev[r] := 0;
                unknown[r] := False;
                //Số nhỏ nhất chia n dư r là số có một chữ số d
              end;
            end;
          end;
        end;
      procedure BFS; //Thuật toán loang
      var
        r, newr: Integer;
        d: TDigit;
      begin
        if not unknown[0] then Exit;
        with Queue do
          while front <= rear do
            begin
              r := items[front];
              Inc(front); //Pop: Lấy một số dư r khỏi hàng đợi
              for d in S do //Xét các chữ số trong S từ nhỏ tới lớn
                begin
                  newr := (r * 10 + d) mod n;
                  //Nối d vào sau số chia n dư r sẽ được số chia n dư newr
                  if unknown[newr] then

```

```

//Cập nhật nếu chưa biết số nào chia n dư newr
begin
    unknown[newr] := False;
    LastDigit[newr] := d;
    //Chữ số hàng đơn vị của số nhỏ nhất chia n dư newr
    prev[newr] := r; //Vết
    if newr = 0 then Exit;
    //Ngừng ngay khi unknown[0] = False
    Inc(rear);
    items[rear] := newr;
    //Push: Đẩy số dư newr vào hàng đợi chờ xử lý tiếp
end;
end;
end;
procedure PrintResult;
var
    f: TextFile;
    i, nd: Integer;
    r: Integer;
    d: TDigit;
begin
    AssignFile(f, OutputFile);
    Rewrite(f);
    try
        if unknown[0] then Write(f, 0) //Vô nghiệm
        else
            begin //Truy vết
                nd := 0;
                r := 0;
                repeat
                    d := LastDigit[r]; //Lưu chữ số hàng đơn vị
                    Inc(nd);
                    ResDigits[nd] := d;
                    r := prev[r]; //Nhảy tới số "sinh ra" r
                until r = 0;
                for i := nd downto 1 do
                    Write(f, ResDigits[i]);

```

```

    end;
finally
    CloseFile(f);
end;
end;
begin
    Enter;
    Init;
    BFS;
    PrintResult;
end.

```

6.40. Xây dựng thuật toán và cài đặt chương trình

Bài này có thuật toán tham lam mang ý tưởng rất đơn giản: Tại mỗi thời điểm, ta chọn quả đào có giá trị nhất để ăn, ăn xong lại chọn tiếp...

Tuy nhiên để có một thuật toán hiệu quả, cần có sự tổ chức dữ liệu hợp lý:

Cách 1. Xếp các quả đào theo thứ tự tăng dần của $t[.]$ vào danh sách L . Duyệt các thời điểm từ a tới $b - 1$, khi duyệt tới thời điểm $time$, những quả đào có $t[.] < time$ sẽ được lấy ra khỏi L và đẩy vào một “kho”. Sau đó lấy quả đào giá trị nhất trong kho ra ăn.

Kho có thể tổ chức bằng cấu trúc dữ liệu Heap, cho phép cài đặt phép toán đẩy một quả đào vào và lấy ra quả đào có giá trị nhất trong thời gian $O(\log n)$.

Chú ý rằng mặc dù $b - a$ có thể rất lớn, ta hoàn toàn có thể đặt $a := b - n$ nếu a đang nhỏ hơn $b - n$. Vì vậy việc duyệt các thời điểm nguyên từ a tới $b - 1$ cũng chỉ mất thời gian $O(n)$.

Cách 2. Sử dụng cấu trúc dữ liệu rừng các tập rời nhau.

Khởi tạo khoảng $[a, b]$ gồm $b - a$ “khe”, mỗi “khe” là một khoảng đơn vị $[i, i + 1]$.

Duyệt các quả đào theo thứ tự giảm dần của $k[.]$. Khi xét đến quả đào i , khe trống đầu tiên đứng sau thời điểm $t[i]$ sẽ được dùng để ăn quả đào i . Nếu không còn khe trống đứng sau thời điểm $t[i]$, quả đào i sẽ bị bỏ qua. Việc tìm khe trống và đánh dấu khe không còn trống được thực hiện dễ dàng bằng cấu trúc dữ liệu rừng các tập rời nhau (độ phức tạp $O(n\alpha(n))$).

Tuy nhiên, để duyệt các quả đào theo thứ tự giảm dần của $k[.]$ cần một thuật toán sắp xếp nữa.

Chương trình

```
{$MODE OBJFPC}
program PeachSelection;
const
  InputFile = 'PEACH.INP';
  OutputFile = 'PEACH.OUT';
  maxN = 100000;
type
  THeap = record
    items: array[1..maxN] of Integer;
    nItems: Integer;
  end;
var
  n, m, a, b: Integer;
  t, k: array[1..maxN] of Integer;
  id: array[1..maxN] of Integer;
  heap: THeap;
  res: Int64;
procedure Enter;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, n, a, b);
    if b - a > n then a := b - n;
    for i := 1 to n do
      ReadLn(f, t[i], k[i]);
  finally
    CloseFile(f);
  end;
end;
procedure SortByT;
var
```

ân một

```
i: Integer;
procedure QSort(L, H: Integer);
var
  i, j: Integer;
  pivot: Integer;
begin
  if L >= H then Exit;
  i := L + Random(H - L + 1);
  pivot := id[i];
  id[i] := id[L];
  i := L;
  j := H;
repeat
  while (t[id[j]] > t[pivot]) and (i < j) do Dec(j);
  if i < j then
    begin
      id[i] := id[j];
      Inc(i)
    end
  else Break;
  while (t[id[i]] < t[pivot]) and (i < j) do Inc(i);
  if i < j then
    begin
      id[j] := id[i];
      Dec(j);
    end
  else Break;
until i = j;
id[i] := pivot;
QSort(L, i - 1);
QSort(i + 1, H);
end;
begin
  for i := 1 to n do id[i] := i;
  QSort(1, n);
end;
procedure Insert(peach: Integer);
var
  p, c: Integer;
```

```

begin
  with heap do
    begin
      Inc(nItems);
      c := nItems;
      repeat
        p := c div 2;
        if (p = 0) or (k[items[p]] >= k[peach]) then
          Break;
        items[c] := items[p];
        c := p;
      until False;
      items[c] := peach;
    end;
  end;
function Extract: Integer;
var
  p, c: Integer;
  peach: Integer;
begin
  with heap do
    begin
      Result := items[1];
      peach := items[nItems];
      Dec(nItems);
      p := 1;
      repeat
        c := p * 2;
        if (c < nItems) and
          (k[items[c]] < k[items[c + 1]]) then Inc(c);
        if (c > nItems) or (k[peach] >= k[items[c]]) then
          Break;
        items[p] := items[c];
        p := c;
      until False;
      items[p] := peach;
    end;
  end;
end;

```

```

procedure Solve;
var
  Time: Integer;
  i, q: Integer;
begin
  heap.nItems := 0;
  Res := 0; q := 1;
  for Time := a to b - 1 do
    begin
      while (q <= n) and (t[id[q]] <= Time) do
        begin
          Insert(id[q]);
          Inc(q);
        end;
      if heap.nItems <> 0 then
        begin
          i := Extract;
          Inc(Res, k[i]);
        end;
    end;
end;
procedure PrintResult;
var
  f: TextFile;
begin
  AssignFile(f, OutputFile); Rewrite(f);
  try
    Write(f, Res);
  finally
    CloseFile(f);
  end;
end;
begin
  Enter;
  SortByT;
  Solve;
  PrintResult;
end.

```

6.41. Xây dựng thuật toán và cài đặt chương trình

Trước hết ta xét một bài toán nhỏ: *Xác định số cầu cảng ít nhất để phục vụ tất cả các tàu.*

- Mỗi tàu ứng với một khoảng $(s_i, f_i]$ trên trục số. Đầu mút s_i gọi là đầu mút mở, còn đầu mút $f_i]$ gọi là đầu mút đóng. Tổng cộng có $2n$ đầu mút.
- Xếp các đầu mút theo thứ tự tăng dần trên trục số. Nếu có nhiều đầu mút ở cùng vị trí thì các đầu mút đóng được xếp trước các đầu mút mở.
- Khởi tạo biến đếm $c := 0$. Duyệt các đầu mút theo thứ tự sắp xếp, gặp đầu mút mở thì tăng c lên 1, gặp đầu mút đóng thì giảm c đi 1.
- Biến đếm c khi xét qua một đầu mút chính là số tàu đang đồng thời neo đậu ở cảng ngay sau thời điểm đó. Vì vậy trong quá trình biến đổi, giá trị lớn nhất mà biến c đạt được chính là số cầu cảng tối thiểu cần sử dụng.

Tiếp theo ta xét bài toán thứ hai: *Khi đã xác định được số cầu cảng ít nhất, cần phân phối các cầu cảng cho các tàu để phục vụ toàn bộ các tàu.*

Một thuật toán tham lam hoạt động rất tự nhiên có thể áp dụng để phân phối các cầu cảng: Khởi tạo “kho” chứa tất cả cầu cảng. Do vai trò các cầu cảng là như nhau, ta có thể thực hiện phép phân phối như sau: Duyệt các đầu mút theo thứ tự đã sắp xếp ở trên:

- Gặp đầu mút mở: Tương ứng với sự kiện một tàu đến cảng, lấy một cầu cảng bất kì trong “kho” cấp cho tàu đó;
- Gặp đầu mút đóng: Tương ứng với sự kiện một tàu rời cảng, trả cầu cảng đã cấp cho tàu đó về “kho”.

“Kho” có thể được cài đặt bằng bất kì cấu trúc dữ liệu gì hỗ trợ hai phép toán: Lấy một cầu cảng ra khỏi “kho” và trả một cầu cảng về “kho”. Cách đơn giản nhất để cài đặt “kho” là sử dụng cấu trúc dữ liệu ngăn xếp.

Ban đầu lần lượt các cầu cảng từ số hiệu lớn tới nhỏ được đẩy vào ngăn xếp (cầu cảng thứ nhất nằm ở đỉnh), sau đó thực hiện thuật toán trên. Khi kết thúc thuật toán, những cầu cảng nằm ở đáy ngăn xếp có thể không bao

giờ được sử dụng, ta có thể loại bỏ những cầu cảng này và từ đó có thể xác định được luôn số cầu cảng tối thiểu cần sử dụng.

Bây giờ ta quay lại bài toán chính với m cầu cảng: Áp dụng thuật toán tương tự trên, nếu mọi việc diễn ra bình thường thì ta có luôn phương án phục vụ các tàu. Chỉ còn phải xử lí duy nhất một vấn đề: Khi gặp đầu mút mở (một tàu đến cảng) nhưng “kho” đã rỗng và không còn cầu cảng để phục vụ.

Trong trường hợp này, bắt buộc ta phải từ chối phục vụ một tàu. Tàu bị từ chối có thể là tàu vừa đến cảng hoặc là một trong những tàu đang (tạm) neo đậu ở cảng sẽ phải rời đi nhường chỗ cho tàu mới đến. Rõ ràng để có lợi nhất, trong số $m + 1$ tàu (m tàu ở cảng và một tàu mới đến), tàu nào có thời điểm rời cảng muộn nhất sẽ phải rời đi.

Như vậy ta sẽ phải quản lí tập Q những tàu đang neo đậu ở cảng, nếu tập này đã đủ m phần tử và có thêm một tàu mới đến (tổng cộng $m + 1$ phần tử), thì tàu có thời điểm rời cảng muộn nhất sẽ bị “đuổi” khỏi tập Q . Cấu trúc dữ liệu để cài đặt tập Q cần hỗ trợ hai phép toán: Đưa thêm một tàu vào tập và loại bỏ tàu có thời điểm rời cảng muộn nhất, không có cấu trúc dữ liệu nào hợp lí hơn hàng đợi ưu tiên mà ta có thể cài đặt bằng Heap.

Chương trình

```
{$MODE OBJFPC}
program SeaPorts;
const
    InputFile = 'SEAPORTS.INP';
    OutputFile = 'SEAPORTS.OUT';
    max = 100000;
type
    TInterval = record
        s, f: Integer;
        lab: Integer;
    end;
    TEndPointType = (eptClose, eptOpen); //Close < Open được xét trước
    TEndPoint = record
        x: Integer;
        ept: TEndPointType;
        segID: Integer;
```

```

end;
THeap = record
  nItems: Integer;
  Items: array[1..max] of Integer;
end;
TStack = record
  Top: Integer;
  Items: array[1..max] of Integer;
end;
var
  Intervals: array[1..max] of TInterval;
  EndPoints: array[1..2 * max] of TEndPoint;
  Heap: THeap;
  Stack: TStack;
  res, m, n: Integer;
procedure Enter;
var
  fi: TextFile;
  i: Integer;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  try
    ReadLn(fi, m, n);
    for i := 1 to n do
      with Intervals[i] do
        begin
          ReadLn(fi, s, f);
          with EndPoints[i * 2 - 1] do
            begin
              x := s;
              segID := i;
              ept := eptOpen;
            end;
          with EndPoints[i * 2] do
            begin
              x := f;
              segID := i;
              ept := eptClose;
            end;
        end;
  except
  end;
end;

```

```

        end;
    end;
finally
    CloseFile(fi);
end;
end;
operator < (const p, q: TEndPoint): Boolean;
begin
    Result := (p.x < q.x) or (p.x = q.x)
                and (p.ept < q.ept);
end;
procedure SortPT(L, H: Integer);
var
    Pivot: TEndPoint;
    i, j: Integer;
begin
    if L >= H then
        Exit;
    i := L + Random(H - L + 1);
    Pivot := EndPoints[i];
    EndPoints[i] := EndPoints[L];
    i := L;
    j := H;
repeat
    while (Pivot < EndPoints[j]) and (i < j) do
        Dec(j);
    if i < j then
        begin
            EndPoints[i] := EndPoints[j];
            Inc(i);
        end
    else Break;
    while (EndPoints[i] < Pivot) and (i < j) do
        Inc(i);
    if i < j then
        begin
            EndPoints[j] := EndPoints[i];
            Dec(j);
        end
end

```

```

    else Break;
until i = j;
EndPoints[i] := Pivot;
SortPT(L, i - 1);
SortPT(i + 1, H);
end;
procedure HeapPush(Value: Integer);
var
  r, c: Integer;
begin
  with Heap do
  begin
    Inc(nItems);
    c := nItems;
    repeat
      r := c div 2;
      if (r = 0) or
        (Intervals[Items[r]].f >= Intervals[Value].f) then
        Break;
      Items[c] := Items[r];
      c := r;
    until False;
    Items[c] := Value;
  end;
end;
function HeapExtractMax: Integer;
var
  r, c: Integer;
  temp: Integer;
begin
  with Heap do
  begin
    Result := Items[1];
    temp := Items[nItems];
    Dec(nItems);
    r := 1;
    repeat
      c := r * 2;
      if (c < nItems) and

```

```

(Intervals[Items[c + 1]].f > Intervals[Items[c]].f)
  then Inc(c);
  if (c > nItems) or
    (Intervals[temp].f >= Intervals[Items[c]].f)
  then Break;
  Items[r] := Items[c];
  r := c;
until False;
Items[r] := temp;
end;
end;
procedure RemoveIntervals;
var
  i: Integer;
  Count: Integer;
begin
  for i := 1 to n do
    Intervals[i].lab := -1;
  Count := 0;
  Heap.nItems := 0;
  res := n;
  for i := 1 to 2 * n do
    with EndPoints[i] do
      case ept of
        eptClose:
          if Intervals[segID].lab <> 0 then Dec(Count);
        eptOpen:
          begin
            Inc(Count);
            HeapPush(segID);
            if Count > m then
              begin
                Intervals[HeapExtractMax].lab := 0;
                Dec(Count);
                Dec(res);
              end;
            end;
          end;
      end;
end;

```

```

procedure StackPush(Value: Integer);
begin
  with Stack do
    begin
      Inc(Top);
      Items[Top] := Value;
    end;
end;
function StackPop: Integer;
begin
  with Stack do
    begin
      Result := Items[Top];
      Dec(Top);
    end;
end;
procedure MakingServiceSchedule;
var
  i: Integer;
begin
  with Stack do
    begin
      Top := m;
      for i := 1 to m do
        Items[i] := i;
    end;
  for i := 1 to 2 * n do
    with EndPoints[i] do
      if Intervals[segID].lab <> 0 then
        case ept of
          eptOpen: Intervals[segID].lab := StackPop;
          //Lấy port ở đỉnh stack phục vụ tàu i
          eptClose: StackPush(Intervals[segID].lab);
          //Port trở thành rỗng khi kết thúc thời gian
        end;
    end;
  procedure PrintResult;
  var
    f: TextFile;

```

```

i: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    WriteLn(f, res);
    for i := 1 to n do
      Write(f, Intervals[i].lab, ' ');
  finally
    CloseFile(f);
  end;
end;
begin
  Enter;
  SortPT(1, 2 * n);
  RemoveIntervals;
  MakingServiceSchedule;
  PrintResult;
end.

```

6.42. Xây dựng thuật toán và cài đặt chương trình

Gọi a_i là dung tích của chai rượu thứ i . Để bài yêu cầu tìm cách uống nhiều rượu nhất, ta tiếp cận theo tiêu chuẩn bỏ lại ít rượu nhất.

Gọi $f[i]$ là số rượu ít nhất phải bỏ trong trường hợp được chọn trong các chai từ 1 tới i đồng thời không được chọn k chai liền nhau và chắc chắn có bỏ chai i .

Vì chai i chắc chắn phải bỏ và không được uống k chai liền nhau, chai liền trước chai i phải bỏ sẽ phải là một chai j nào đó trong phạm vi từ $i - k$ tới $i - 1$. Vì vậy ta có công thức truy hồi tính $f[i]$ như sau:

$$f[i] = \min\{f[j]: j \in [i - k, i - 1]\} + a_i.$$

Cuối cùng ta quan tâm tới $f[n + 1]$: Số rượu ít nhất phải bỏ khi được chọn từ chai 1 tới chai $n + 1$ mà chắc chắn chai $n + 1$ phải bỏ. Chai $n + 1$ được dùng để đánh dấu và đơn giản hóa phép tính có dung lượng $a_{n+1} = 0$. Khi đã tính hết mảng f , việc truy vết tìm phương án bỏ đi các chai không có gì khó khăn.

Một thuật toán tầm thường với độ phức tạp $\Theta(n \times k)$ đến đây cũng đã có thể áp dụng để giải công thức truy hồi. Tuy vậy để có một thuật toán hiệu quả hơn, cần có sự hỗ trợ về phía cấu trúc dữ liệu để giảm thời gian thực hiện giải thuật.

Ta sẽ tính lần lượt các giá trị $f[1], f[2], \dots, f[n + 1]$. Có ba cách giải có thể áp dụng:

Cách 1. Phép tính $\min\{f[j] : j \in [i - k, i - 1]\}$ có thể tăng tốc bằng cấu trúc dữ liệu Heap:

Heap dùng để chứa các chỉ số, trong đó chỉ số i được gọi là ưu tiên hơn chỉ số j nếu $f[i] < f[j]$. Mỗi khi tính xong một giá trị $f[i]$, chỉ số i được đẩy vào Heap. Khi muốn tính giá trị $f[i]$, phần tử j ở đỉnh Heap sẽ được đọc.

- Nếu $j \geq i - k$, ta đặt $f[i] := f[j] + a_i$ bởi j là vị trí có $f[j]$ nhỏ nhất và thuộc phạm vi truy vấn $[i - k, i - 1]$;
- Nếu $j < i - k$, phần tử j đó chắc chắn không tham gia vào quá trình tính các $f[\cdot]$ tính từ vị trí i trở về sau theo công thức truy hồi, phần tử j này sẽ được lấy ra và lặp lại quá trình đó tới khi phần tử ở đỉnh Heap nằm trong phạm vi từ $i - k$ tới $i - 1$...

Dễ thấy rằng mỗi chỉ số chỉ được đẩy vào Heap đúng một lần và vì vậy số lần lấy phần tử ra khỏi Heap không thể vượt quá n . Thời gian thực hiện giải thuật là $O(n \log n)$.

Cách 2. Từ công thức truy hồi $f[i] = \min\{f[j] : j \in [i - k, i - 1]\} + a_i$, ta thấy rằng phép tính \min được thực hiện trên một khoảng liên tiếp các $f[j]$. Một cấu trúc dữ liệu truy vấn phạm vi có thể áp dụng để trả lời nhanh truy vấn \min bằng các thuật toán RMQ (range-minimum query). Chẳng hạn với cấu trúc dữ liệu cây phân đoạn, ta có thể giải công thức truy hồi trong thời gian $O(n \log n)$.

Cách 3. Cách 1 và cách 2 tuy đã khá hiệu quả nhưng còn một phương pháp khác cài đặt đơn giản hơn và đạt tốc độ cao hơn. Dùng một hàng đợi hai đầu Q chứa các chỉ số. Hàng đợi hai đầu này cần hỗ trợ các phép toán:

- PushRear(i): Đẩy chỉ số i vào cuối hàng đợi;
- GetRear: Đọc phần tử ở cuối hàng đợi;
- PopRear: Lấy ra một phần tử ở cuối hàng đợi;

- GetFront: Đọc phần tử ở đầu hàng đợi;
- PopFront: Lấy ra một phần tử ở đầu hàng đợi.

Thuật toán tính các $f[.]$ khi đó có thể viết như sau:

```

a[n + 1] := 0;
f[0] := 0;
Q := {0}
for i := 1 to n + 1 do
begin
  while (Q <> Ø) and (GetFront < i - k) do
    PopFront;
  f[i] := f[GetFront] + a[i];
  while (Q <> Ø) and (f[i] < f[GetRear]) do
    PopRear;
  PushRear(i);
end;

```

Tính đúng đắn và hiệu suất của thuật toán trên được chỉ ra bởi nhận xét sau:

- (i) Hàng đợi Q chứa các chỉ số i theo thứ tự tăng dần của $f[i]$. Bởi trước khi i được đẩy vào Q , những chỉ số đứng cuối Q mà có $f[.] \geq f[i]$ sẽ bị lấy ra khỏi Q vì chắc chắn những chỉ số này không cần thiết để tính công thức truy hồi nữa.
- (ii) Từ (i), khi bắt đầu tính $f[i]$, chừng nào chỉ số đứng đầu Q còn nhỏ hơn $i - k$, ta loại bỏ chỉ số này bằng lệnh PopFront.

Khi chỉ số đứng đầu Q đã nằm trong phạm vi $[i - k, i - 1]$, ta tính

$$f[i] := f[GetFront] + a[i]$$

bởi $f[GetFront]$ chính bằng $\min\{f[j]: j \in [i - k, i - 1]\}$.

- (iii) Thuật toán có thời gian thực hiện $\Theta(n)$ nếu cài đặt Q bằng mảng. Thật vậy, tổng cộng có đúng $n + 1$ thao tác PushRear, từ đó tổng số thao tác PopFront cũng như PopRear không thể vượt quá $n + 1$.

Chương trình

Thuật toán dùng Double-Ended Queue (hàng đợi hai chiều)

```
{$MODE OBJFPC}
program Bottles;
const
  InputFile = 'BOTTLES.INP';
  OutputFile = 'BOTTLES.OUT';
  maxN := Round(4E5);
var
  a: array[1..maxN + 1] of Integer;
  f: array[0..maxN + 1] of Int64;
  queue: array[1..maxN + 1] of Integer;
  front, rear: Integer;
  n, k: Integer;
  Sum, Res, m: Int64;
procedure Enter;
var
  fi: TextFile;
  i: Integer;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  try
    ReadLn(fi, n, k);
    Sum := 0;
    a[n + 1] := 0;
    for i := 1 to n do
      begin
        Read(fi, a[i]);
        Sum := Sum + a[i];
      end;
    finally
      CloseFile(fi);
    end;
  end;
procedure PushRear(i: Integer); inline;
begin
  Inc(rear);
```

```

queue[rear] := i;
end;
function GetRear: Integer; inline;
begin
    Result := queue[rear];
end;
procedure PopRear; inline;
begin
    Dec(rear);
end;
function GetFront: Integer; inline;
begin
    Result := queue[front];
end;
function PopFront: Integer; inline;
begin
    Inc(front);
end;
function NotEmpty: Boolean; inline;
begin
    Result := front <= rear;
end;
procedure Optimize;
var
    i, j: Integer;
    temp: Int64;
begin
    a[n + 1] := 0;
    f[0] := 0;
    front := 1;
    rear := 1;
    queue[1] := 0;
    for i := 1 to n + 1 do
        begin
            while NotEmpty and (GetFront < i - k) do
                PopFront;
            f[i] := f[GetFront] + a[i];
            while NotEmpty and (f[i] < f[GetRear]) do
                PopRear;

```

```

    PushRear(i);
  end;
//Truy vết
Res := Sum - f[n + 1];
temp := f[n + 1];
m := n;
for i := n downto 1 do
  if temp = f[i] then
    begin
      temp := temp - a[i];
      a[i] := -1;
      Dec(m);
    end;
  end;
procedure PrintResult;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    WriteLn(f, m, ' ', res);
    for i := 1 to n do
      if a[i] > 0 then Write(f, i, ' ');
  finally
    CloseFile(f);
  end;
end;
begin
  Enter;
  Optimize;
  PrintResult;
end.

```

Thuật toán dùng Heap

```

{$MODE OBJFPC}
program Bottles;
const

```

```

InputFile = 'BOTTLES.INP';
OutputFile = 'BOTTLES.OUT';
maxN = Round(4E5);

type
  THeap = record
    nItems: Integer;
    items: array[1..maxN + 2] of Integer;
  end;
var
  a: array[1..maxN + 1] of Integer;
  f: array[0..maxN + 1] of Int64;
  trace: array[0..maxN + 1] of Integer;
  n, k: Integer;
  Sum, Res, m: Int64;
  Heap: THeap;
procedure Enter;
var
  fi: TextFile;
  i: Integer;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  try
    ReadLn(fi, n, k);
    Sum := 0;
    a[n + 1] := 0;
    for i := 1 to n do
      begin
        Read(fi, a[i]);
        Inc(Sum, a[i]);
      end;
  finally
    CloseFile(fi);
  end;
end;
function Get: Integer;
begin
  Result := heap.items[1];
end;

```

```

procedure Extract;
var
  p, c: Integer;
  temp: Integer;
begin
  with heap do
    begin
      p := 1;
      temp := items[nItems];
      Dec(nItems);
      repeat
        c := p * 2;
        if (c < nItems) and
          (f[items[c + 1]] < f[items[c]])
        then Inc(c);
        if (c > nItems) or (f[temp] <= f[items[c]]) then
          Break;
        items[p] := items[c];
        p := c;
      until False;
      items[p] := temp;
    end;
end;

procedure Insert(temp: Integer);
var
  p, c: Integer;
begin
  with heap do
    begin
      Inc(nItems);
      c := nItems;
      repeat
        p := c div 2;
        if (p = 0) or (f[items[p]] <= f[temp]) then Break;
        items[c] := items[p];
        c := p;
      until False;
      items[c] := temp;
    end;

```

```

end;
procedure Optimize;
var
  i, j, t: Integer;
begin
  f[n + 1] := 0;
  with Heap do
    begin
      nItems := 1;
      Items[1] := 0;
    end;
  for i := 1 to n + 1 do
    begin
      repeat
        j := Get;
        if i - j > k then Extract
        else Break;
      until False;
      Trace[i] := j;
      f[i] := f[j] + a[i];
      Insert(i);
    end;
  //Truy vết
  Res := Sum - f[n + 1];
  m := n;
  i := n + 1;
  repeat
    j := Trace[i];
    if j = 0 then Break;
    a[j] := -1;
    i := j;
    Dec(m);
  until False;
end;
procedure PrintResult;
var
  f: TextFile;
  i: Integer;
begin

```

```

AssignFile(f, OutputFile);
Rewrite(f);
try
  WriteLn(f, m, ' ', res);
  for i := 1 to n do
    if a[i] > 0 then Write(f, i, ' ');
finally
  CloseFile(f);
end;
end;
begin
  Enter;
  Optimize;
  PrintResult;
end.

```

Thuật toán dùng cây phân đoạn

```

{$MODE OBJFPC}
program Bottles;
const
  InputFile = 'BOTTLES.INP';
  OutputFile = 'BOTTLES.OUT';
  maxN = Round(4E5);
  maxV = Round(1E9);
  infty = maxN * maxV + 1;
var
  a: array[1..maxN + 1] of Integer;
  f: array[1..maxN + 1] of Int64;
  tree: array[1..4 * maxN] of Int64;
  l, h: array[1..4 * maxN] of Integer;
  leaf: array[1..maxN] of Integer;
  n, k: Integer;
  Sum, Res: Int64;
procedure Enter;
var
  fi: TextFile;
  i: Integer;
begin
  AssignFile(fi, InputFile);

```

```

Reset(fi);
try
  ReadLn(fi, n, k);
  Sum := 0;
  a[n + 1] := 0;
  for i := 1 to n do
    begin
      Read(fi, a[i]);
      Inc(Sum, a[i]);
    end;
  finally
    CloseFile(fi);
  end;
end;
procedure Build(x: Integer; low, high: Integer);
var
  *middle: Integer;
begin
  l[x] := low;
  h[x] := high;
  tree[x] := 0;
  if low = high then
    leaf[low] := x
  else
    begin
      middle := (low + high) shr 1;
      Build(x * 2, low, middle);
      Build(x * 2 + 1, middle + 1, high);
    end;
end;
function Min(p, q: Int64): Int64;
begin
  if p < q then Result := p else Result := q;
end;
procedure Update(i: Integer; f: Int64);
var
  x: Integer;
begin
  x := leaf[i];

```

```

tree[x] := f;
while x > 1 do
begin
  x := x shr 1;
  tree[x] := Min(tree[2 * x], tree[2 * x + 1]);
end;
end;

function Query(i, j: Integer): Int64;
function Request(x: Integer): Int64;
begin
  if (h[x] < i) or (l[x] > j) then Exit(infty);
  if (l[x] >= i) and (h[x] <= j) then Exit(tree[x]);
  Result := Min(Request(2 * x), Request(2 * x + 1));
end;
begin
  Result := Request(1);
end;

procedure Solve;
var
  i: Integer;
begin
  for i := 1 to n + 1 do
  begin
    if i <= k then f[i] := a[i]
    else f[i] := Query(i - k, i - 1) + a[i];
    if i <= n then
      Update(i, f[i]);
  end;
  res := Sum - f[n + 1];
end;

procedure PrintResult;
var
  fo: TextFile;
  i, nSel: Integer;
  temp: Int64;
begin
  temp := f[n + 1];
  nSel := 0;
  for i := n downto 1 do

```

```

if f[i] = temp then
begin
    temp := temp - a[i];
    a[i] := -1;
end
else
    Inc(nSel);
AssignFile(fo, OutputFile);
Rewrite(fo);
try
    WriteLn(fo, nSel, ' ', res);
    for i := 1 to n do
        if a[i] <> -1 then Write(fo, i, ' ');
finally
    CloseFile(fo);
end;
end;
begin
    Enter;
    Build(1, 1, n);
    Solve;
    PrintResult;
end.

```

6.43. Xây dựng thuật toán và cài đặt chương trình

Cài đặt một dạng cây nhị phân tìm kiếm T (chẳng hạn cây Splay) để biểu diễn và duy trì danh sách sau các phép tráo (độ phức tạp $O(x \log n)$).

Đồng bộ cây với một mảng A , tìm dãy con đơn điệu tăng dài nhất trên mảng (độ phức tạp $O(n \log n)$).

Những phần tử thuộc dãy con đơn điệu tăng dài nhất được cố định, những phần tử khác được rút ra chèn vào vị trí đúng của nó, việc xoá và chèn được thực hiện trên cây T . Mỗi phép tráo trong phương án tối ưu sẽ được in ra tệp kết quả.

Chương trình

```

{$MODE OBJFPC}
{$M 10000000}
program ShuffleCards;

```

```

const
  InputFile = 'SHUFFLE.INP';
  OutputFile = 'SHUFFLE.OUT';
  max = 100000;
type
  PNode = ^TNode;
  TNode = packed record
    value: Integer;
    size: Integer;
    parent: PNode;
    childL, childR: PNode;
  end;
var
  fi, fo: TextFile;
  root, nilT: PNode;
  sentinel: TNode;
  tmp: array[1..max + 1] of TNode;
  ptr: array[1..max + 1] of PNode;
  a: array[0..max + 1] of Integer;
  mark: array[1..max] of Boolean;
  trace: array[0..max + 1] of Integer;
  s: array[1..max + 2] of Integer;
  maxlen, res: Integer;
  n, m: Integer;
procedure OpenFiles;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  AssignFile(fo, OutputFile);
  Rewrite(fo);
end;
procedure CloseFiles;
begin
  CloseFile(fi);
  CloseFile(fo);
end;
procedure SetLink(ParentNode, ChildNode: PNode; InLeft:Boolean);
begin
  ChildNode^.parent := ParentNode;

```

```

if InLeft then ParentNode^.ChildL := ChildNode
else ParentNode^.ChildR := ChildNode;
end;
procedure DelLink(ParentNode, ChildNode: PNode; InLeft:Boolean);
begin
    ChildNode^.parent := nilT;
    if InLeft then
        ParentNode^.ChildL := nilT
    else
        ParentNode^.ChildR := nilT;
end;
procedure UpdateSize(x: PNode);
begin
    x^.size := x^.ChildL^.size + x^.ChildR^.size + 1;
end;
procedure InitTree;
var
    i: Integer;
    function Build(L, H: Integer): PNode;
    var
        left, right: PNode;
        M: Integer;
    begin
        if L > H then Exit(nilT);
        M := (L + H) div 2;
        ptr[m] := @tmp[m];
        Result := ptr[m];
        tmp[m].value := m;
        left := Build(L, M - 1);
        right := Build(M + 1, H);
        SetLink(Result, left, True);
        SetLink(Result, right, False);
        UpdateSize(Result);
    end;
    begin
        sentinel.size := 0;
        nilT := @sentinel;
        ReadLn(fi, n, m);
        root := Build(1, n + 1);
    end;

```

```

root^.parent := nilT;
end;
procedure Splay(x: PNode);
var
  y, z: PNode;
  procedure UpTree(x: PNode);
  var
    y, z, branch: PNode;
  begin
    y := x^.parent;
    z := y^.parent;
    if y^.ChildL = x then
      begin
        branch := x^.ChildR;
        SetLink(y, branch, True);
        SetLink(x, y, False);
      end
    else
      begin
        branch := x^.ChildL;
        SetLink(y, branch, False);
        SetLink(x, y, True);
      end;
    SetLink(z, x, z^.ChildL = y);
    UpdateSize(y);
    UpdateSize(x);
  end;
  begin
    repeat
      y := x^.parent;
      if y = nilT then Break;
      z := y^.parent;
      if z <> nilT then
        if (z^.ChildL = y) = (y^.ChildL = x) then
          UpTree(y)
        else
          UpTree(x);
      UpTree(x);
    until False;
  
```

```

    root := x;
end;
function FindNode(r: PNode; pos: Integer): PNode;
begin
    if pos <= r^.ChildL^.size then
        Result := FindNode(r^.ChildL, pos)
    else
        if pos = r^.ChildL^.size + 1 then
            Result := r
        else
            Result := FindNode(r^.ChildR, pos - r^.ChildL^.size - 1);
end;
procedure Delete(x: PNode);
var
    a, b: PNode;
begin
    Splay(x);
    a := x^.ChildL; b := x^.childR;
    b^.parent := nilT;
    while b^.ChildL <> nilT do b := b^.ChildL;
    Splay(b);
    SetLink(b, a, True);
    UpdateSize(b);
    root := b;
end;
procedure Insert(id: Integer; x: PNode);
var
    left, right: PNode;
begin
    right := FindNode(root, id);
    Splay(right);
    left := right^.ChildL;
    DelLink(right, left, True);
    UpdateSize(right);
    SetLink(x, left, True);
    SetLink(x, right, False);
    UpdateSize(x);
    root := x;
end;

```

```
procedure DoShuffle(i, j: Integer);
var
  x: PNode;
begin
  x := FindNode(root, i);
  Delete(x);
  Insert(j, x);
end;
procedure CopyToArray;
var
  count: Integer;
  procedure Visit(r: PNode);
  begin
    if r = nil then Exit;
    Visit(r^.ChildL);
    Inc(count); a[count] := r^.value;
    Visit(r^.ChildR);
  end;
begin
  Count := 0;
  a[0] := 0;
  Visit(root);
end;
procedure GetLastConfig;
var
  k: Integer;
  i, j: Integer;
begin
  for k := 1 to m do
    begin
      ReadLn(fi, i, j);
      DoShuffle(i, j);
    end;
  CopyToArray;
end;
procedure GetLIS;
var
  i: Integer;
  lambda: Integer;
```

```

function Find(v: Integer): Integer;
var
    L, M, H: Integer;
begin
// $a[s[1]] < a[s[2]] < \dots < a[s[maxlen]]$ 
    L := 1; H := maxlen;
    while L <= H do // $a[s[L - 1]] < v \leq a[s[H + 1]]$ 
        begin
            M := (L + H) div 2;
            if a[s[M]] < v then L := M + 1
            else H := M - 1
        end;
        Result := H;
    end;
begin
    a[0] := 0;
    a[n + 1] := n + 1;
    MaxLen := 1;
    s[1] := 0;
    for i := 1 to n + 1 do
        begin
            lambda := Find(a[i]) + 1;
            trace[i] := s[lambda - 1];
            if lambda > MaxLen then MaxLen := lambda;
            s[lambda] := i;
        end;
    FillChar(mark[1], n, True);
    i := trace[n + 1];
    repeat
        mark[a[i]] := False;
        i := trace[i];
    until i = 0;
    res := n - (maxlen - 2)
end;
procedure PrintResult;
var
    i, j, k: Integer;
begin
    WriteLn(fo, res);

```

```

for k := n downto 1 do
  if mark[k] then
    begin
      Splay(ptr[k]);
      i := root^.ChildL^.size + 1;
      Delete(ptr[k]);
      Splay(ptr[k + 1]);
      j := root^.ChildL^.size + 1;
      Insert(j, ptr[k]);
      WriteLn(fo, i, ' ', j);
    end;
  end;
begin
  OpenFiles;
  InitTree;
  GetLastConfig;
  GetLIS;
  PrintResult;
  CloseFiles;
end.

```

6.44. Xây dựng thuật toán và cài đặt chương trình

Xét các mức nước từ 10^6 giảm về 1, ban đầu khi mức nước là 10^6 , các ô đều bị chìm.

Khi giảm dần mức nước, các ô sẽ dần nổi lên. Khi một ô nổi lên, tạo một tập chứa ô đó và hợp tập này với những tập chứa các ô lân cận (nếu ô lân cận đã nổi lên).

Thuật toán có thể cài đặt dễ dàng bằng cấu trúc dữ liệu rỗng các tập rời nhau với độ phức tạp $O(n\alpha(n))$.

Chương trình

```

{$MODE OBJFPC}
{$INLINE ON}
const
  InputFile = 'ISLANDS.INP';
  OutputFile = 'ISLANDS.OUT';
  maxMN = 1000;
  maxH = Round(1E6);

```

```

dx: array[1..4] of Integer = (0, 0, 1, -1);
dy: array[1..4] of Integer = (1, -1, 0, 0);

type
  TPoint = record
    x, y: Integer;
  end;

var
  h: array[1..maxH] of Integer;
  p: array[1..maxMN * maxMN] of TPoint;
  link: array[1..maxMN * maxMN] of Integer;
  upper: array[0..maxMN + 1, 0..maxMN + 1] of Boolean;
  lab: array[1..maxMN * maxMN] of Integer;
  m, n: Integer;
  res, nc: Integer;

procedure Enter;
var
  f: TextFile;
  i, j, k, height: Integer;
begin
  AssignFile(f, InputFile); Reset(f);
  try
    FillChar(h, SizeOf(h), 0);
    ReadLn(f, m, n);
    k := 0;
    for i := 1 to m do
      begin
        for j := 1 to n do
          begin
            Inc(k);
            p[k].x := i;
            p[k].y := j;
            Read(f, height);
            link[k] := h[height];
            h[height] := k;
          end;
        ReadLn(f);
      end;
  finally
    CloseFile(f);
  end;
end;

```

```

    end;
end;
procedure Init;
begin
  FillChar(upper, SizeOf(upper), False);
  res := -1;
  nc := 0;
end;
function Encode(const p: TPoint): Integer; inline;
begin
  with p do
    Result := Pred(x) * n + y;
end;
procedure MakeSet(u: Integer); inline;
begin
  lab[u] := 0;
  Inc(nc);
end;
function FindSet(u: Integer): Integer;
begin
  if lab[u] <= 0 then Result := u
  else
    begin
      Result := FindSet(lab[u]);
      lab[u] := Result;
    end;
end;
procedure Union(u, v: Integer); inline;
begin
  u := FindSet(u);
  v := FindSet(v);
  if u = v then Exit;
  if lab[u] < lab[v] then
    lab[v] := u
  else
    begin
      if lab[u] = lab[v] then Dec(lab[v]);
      lab[u] := v;
    end;
end;

```

```

    Dec(nc);
end;
procedure Solve;
var
    height: Integer;
    i, d: Integer;
    pt, qt: TPoint;
begin
    for height := maxH downto 1 do
        begin
            i := h[height];
            while i <> 0 do
                begin
                    pt := p[i];
                    upper[pt.x, pt.y] := True;
                    MakeSet(Encode(pt));
                    for d := 1 to 4 do
                        begin
                            qt.x := pt.x + dx[d];
                            qt.y := pt.y + dy[d];
                            if upper[qt.x, qt.y] then
                                Union(Encode(pt), Encode(qt));
                            end;
                            i := link[i];
                        end;
                        if nc > res then res := nc;
                    end;
                end;
            if nc > res then res := nc;
        end;
    end;
procedure PrintResult;
var
    f: TextFile;
begin
    AssignFile(f, OutputFile);
    Rewrite(f);
    try
        Write(f, res);
    finally
        CloseFile(f);
    end;

```

```

end;
begin
    Enter;
    Init;
    Solve;
    PrintResult;
end.

```

6.45. Xây dựng thuật toán và cài đặt chương trình

Dựng cây biểu diễn tập n người, mỗi nút chứa một người, đồng nhất số hiệu nút với số hiệu người chứa trong nút. Nút i là cha của nút j nếu người i là thủ trưởng trực tiếp của người j .

Duyệt cây bằng thuật toán tìm kiếm theo chiều sâu (DFS) bắt đầu từ nút gốc 1, duyệt xong nút nào đưa nút đó vào danh sách L . Gọi $last[i]$ là vị trí của nút i trong danh sách L . Ngoài ra, khi bắt đầu duyệt tới một nút i , ghi lại số phần tử đã có trong danh sách vào biến $first[i]$.

```

procedure Visit(i: Integer);
begin
    first[i] := count;
    for j ∈ {Nút con của i} do
        Visit(j);
    Inc(count);
    last[i] := count;
end;
begin
    count := 0;
    Visit(1);
end;

```

Khi đó tập những người nằm trong phạm vi quản lí của người i sẽ nằm trong danh sách L từ vị trí $first[i] + 1$ tới vị trí $last[i] - 1$.

- Tác vụ $p A X$ đơn giản là tăng tất cả các phần tử trong danh sách L từ vị trí $first[A] + 1$ tới vị trí $last[A] - 1$ lên X đơn vị;
- Tác vụ $u A$ là truy vấn giá trị của phần tử tại vị trí $last[A]$ trong danh sách.

Hai tác vụ trên có thể cài đặt dễ dàng bằng một cấu trúc dữ liệu quản lý phạm vi. Trong bài này ta có thể sử dụng cây chỉ số nhị phân (binary indexed trees).

Chú ý: Mỗi nút x của cây chỉ số nhị phân chứa $sum[x]$ là tổng lương của những người trong nhánh đó. Do lương của mỗi người trong suốt quá trình quản lí nằm trong phạm vi các số nguyên có dấu 32 bit. Ta có thể cho $sum[x]$ thuộc kiểu số nguyên 32 bit. Các tác vụ p có thể làm cho $sum[x]$ bị tràn, nhưng do cơ chế quay vòng giá trị khi bị tràn số và kết quả cuối cùng là *longint*, việc tràn này không gây ảnh hưởng tới kết quả. Chỉ lưu ý dịch chương trình với dẫn hướng biên dịch {\$Q-}.

Chương trình

```
{$MODE OBJFPC}
{$R-, Q-, S-, I-}
{$OPTIMIZATION LEVEL2}
{$INLINE ON}
{$M 30000000}
program SALARY;
const
    InputFile = 'SALARY.INP';
    OutputFile = 'SALARY.OUT';
    maxN = Round(5E5);
    maxM = Round(5E5);
var
    fi, fo: TextFile;
    w: array[1..maxN] of Integer;
    link: array[1..maxN] of Integer;
    head: array[1..maxN] of Integer;
    bit: array[1..maxN] of integer;
    n, m: Integer;
    first, last: array[1..maxN] of Integer;
    nlist: Integer;
procedure OpenFiles;
begin
    AssignFile(fi, InputFile);
    Reset(fi);
    AssignFile(fo, OutputFile);
```

```

Rewrite(fo);
end;
procedure CloseFiles;
begin
  CloseFile(fi);
  CloseFile(fo);
end;
procedure Enter;
var
  i, j: Integer;
begin
  ReadLn(fi, n, m);
  ReadLn(fi, w[1]);
  FillDWORD(head[1], n, 0);
  for i := 2 to n do
    begin
      ReadLn(fi, w[i], j);
      link[i] := head[j];
      head[j] := i;
    end;
  end;
procedure Visit(u: Integer);
var
  i: Integer;
begin
  first[u] := nList;
  i := head[u];
  while i <> 0 do
    begin
      Visit(i);
      i := link[i];
    end;
  Inc(nlist);
  last[u] := nlist;
end;
procedure Increase(i, x: Integer); inline;
begin
  while i > 0 do
    begin

```

```

    Inc(bit[i], x);
    i := i and Pred(i);
  end;
end;
procedure Update(a, x: Integer); inline;
begin
  Increase(last[a] - 1, x);
  Increase(first[a], -x);
end;
function Query(i: Integer): integer; inline;
begin
  Result := 0;
  while i <= nlist do
    begin
      Inc(Result, bit[i]);
      Inc(i, i and -i);
    end;
  end;
procedure Solve;
var
  i: Integer;
  cmd: Char;
  a, x: Integer;
begin
  Enter;
  nList := 0;
  Visit(1);
  FillChar(bit, SizeOf(bit), 0);
  for i := 1 to m do
    begin
      Read(fi, cmd);
      case cmd of
        'p':
          begin
            ReadLn(fi, a, x);
            Update(a, x);
          end;
        'u':
          begin

```

```

        ReadLn(fi, a);
        WriteLn(fo, Query(last[a]) + w[a]);
    end;
end;
begin
    OpenFiles;
    try
        Solve;
    finally
        CloseFiles;
    end;
end.

```

CHUYÊN ĐỀ 7

7.1. Thuật toán

Đảo ngược chiều cung của đồ thị, thêm một đỉnh 0 nối tới tất cả các đỉnh khác bằng cung trọng số 0.

Áp dụng thuật toán Bellman-Ford với đỉnh 0 là đỉnh xuất phát.

Khi thuật toán kết thúc, $\delta^*(u) = \delta(0, u)$.

7.2. Hướng dẫn

Thực hiện thuật toán Bellman-Ford (tối đa $n - 1$ lần quét danh sách cung và thực hiện phép co), sau đó ta quét lại danh sách cung xem có thể thực hiện được phép co nào nữa hay không.

Nếu còn có thể co theo cung (u, v) nào đó, ta kết luận đồ thị có chu trình âm và thực hiện phép co này, sau đó lần ngược vết đường đi từ đỉnh v .

Nếu quá trình lần vết đi lặp lại một đỉnh x nào đó, ta có một chu trình bắt đầu và kết thúc ở đỉnh x .

7.3. Hướng dẫn

Có nhiều ví dụ về hệ ràng buộc trên thực tế, chẳng hạn một công trình xây dựng có n công đoạn. Vì lí do kĩ thuật, một công đoạn x_j không được bắt đầu muộn hơn w_{ij} thời gian so với công đoạn x_i . Ràng buộc này có thể viết dưới dạng $x_j - x_i \leq w_{ij}$.

Một dạng ràng buộc khác là công đoạn x_j phải bắt đầu sau khi công đoạn x_i bắt đầu được ít nhất w_{ij} thời gian, ràng buộc này cũng có thể viết dưới dạng $x_j - x_i \geq w_{ij}$ hay $x_i - x_j \leq -w_{ij}$.

Coi mỗi biến là một đỉnh của đồ thị, mỗi ràng buộc $x_j - x_i \leq w_{ij}$ cho tương ứng với một cạnh (x_i, x_j) có trọng số w_{ij} .

Khi đó nếu đồ thị có chu trình âm thì không tồn tại giải pháp. Thật vậy, giả sử tồn tại chu trình âm, không làm mất tính tổng quát, giả sử chu trình âm đó là $C = \langle x_1, x_2, \dots, x_k, x_1 \rangle$, ta có

$$x_2 - x_1 \leq w_{12}$$

$$x_3 - x_2 \leq w_{23}$$

...

$$x_1 - x_k \leq w_{k1}$$

Tổng về trái của các bất đẳng thức trên bằng 0 và tổng về phải chính là trọng số của chu trình (âm), bất đẳng thức $0 \leq w(c) < 0$ không bao giờ xảy ra.

Nếu đồ thị không có chu trình âm, ta thêm vào một đỉnh s và các cạnh nối trọng số 0 từ s tới mọi đỉnh khác. Dùng thuật toán Bellman-Ford để tìm đường đi ngắn nhất từ s , khi đó các nhãn $d[x_i] = \delta(s, x_i)$ chính là một cách gán giá trị thỏa mãn tất cả các ràng buộc. Hơn nữa, cách này còn làm khoảng giá trị gán cho các biến là hẹp nhất:

$$\max_{1 \leq i \leq n} \{x_i\} - \min_{1 \leq j \leq n} \{x_j\} \rightarrow \min.$$

7.4. Hướng dẫn

Để chứng minh vòng lặp chính của thuật toán thực hiện không quá $\left\lceil \frac{n}{2} \right\rceil$

lần, ta nhận xét rằng với một đường đi đơn bắt kì thì số những vị trí mà hai

cung liên tiếp nhau trên đường đi có một cung thuộc E_1 và một cung thuộc E_2 không vượt quá $\left\lceil \frac{n}{2} \right\rceil$.

7.5. Hướng dẫn

Coi mỗi loại tiền là một đỉnh của đồ thị có hướng, trọng số $w(i, j) = \lg r_{ij}$.

Tìm chu trình âm.

7.6. Chứng minh và các bước xây dựng thuật toán

a) Nếu $\mu^* = 0$, đồ thị không có chu trình âm. Tồn tại đường đi ngắn nhất từ s tới v là đường đi đơn (qua không quá $n - 1$ cạnh). Tức là:

$$\delta(v) = \min_{0 \leq k \leq n-1} \{\delta_k(v)\}.$$

b) Ta thấy rằng $n - k > 0$ vì $k \leq n - 1$. Ngoài ra, $\delta_n(v) > \delta(v)$ và chắc chắn tồn tại một giá trị k để $\delta_k(v) = \delta(v)$. Từ đó suy ra

$$\frac{\delta_n(v) - \delta_k(v)}{n - k} > 0.$$

c) Trước hết, ta có $\delta(v) \leq \delta(u) + x$ vì độ dài đường đi từ u tới v dọc chu trình C có độ dài x .

Nếu $\delta(v) < \delta(u) + x$, lại từ $\delta(u) \leq \delta(v) + (-x)$ vì độ dài đường đi từ v tới u dọc chu trình C có độ dài $-x$.

Suy ra $\delta(v) < \delta(v) + (-x) + x = \delta(v)$, sự mâu thuẫn này cho thấy $\delta(v) = \delta(u) + x$.

d) Gọi C là chu trình độ dài 0, tìm một đường đi đơn ngắn nhất Q từ s tới một đỉnh $u \in C$ sau đó đi tiếp dọc chu trình C để được một đường đi qua n cung.

Giả sử cuối cùng ta đến đỉnh v , từ kết quả câu c) ta có $\delta(v) = \delta(u) + x$ với x là độ dài quãng đường từ u tới v dọc chu trình C .

Đường đi P chắc chắn đi lặp đỉnh và chứa trọn chu trình C (vì P qua n cung), ngoài ra đi lặp chu trình C bao nhiêu vòng thì cũng không làm tăng độ dài quãng đường. Từ P là đường đi từ s tới v qua đúng n cung, ta suy ra P cũng là đường đi ngắn nhất từ s tới v . Tức là $\delta_n(v) = \delta(v)$.

7.7.

Mặt khác phải tồn tại một giá trị k để $\delta_k(v) = \delta(v)$ và với mọi $k' \neq k$ thì $\delta_{k'}(v) > \delta_k(v)$. Suy ra:

$$\max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n - k} = 0.$$

e) Vì đồ thị không có chu trình âm, $\forall v \in V$ ta có

$$\max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n - k} \geq 0.$$

Ngoài ra, lại tồn tại đỉnh v để bất đẳng thức trên xảy ra dấu “=” . Suy ra

$$\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n - k} = 0.$$

f) Nếu cộng tất cả các trọng số cạnh thêm Δ , tất cả các $\delta_n(v)$ tăng lên $n\Delta$ và tất cả các $\delta_k(v)$ tăng lên $k\Delta$, ngoài ra μ^* cũng tăng lên Δ .

Trừ tất cả các trọng số cạnh đi μ^* , đồ thị vẫn không có chu trình âm nên từ kết quả câu e), ta có:

$$\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - n\mu^* - \delta_k(v) + k\mu^*}{n - k} = 0;$$

$$\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n - k} - \mu^* = 0.$$

g) Tính μ^* bằng thuật toán quy hoạch động trong thời gian $O(|V||E|)$. Xác định

$$\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\delta_n(v) - \delta_k(v)}{n - k}$$

trong thời gian $O(|V|^2)$.

7.7. Hướng dẫn

Coi mỗi đường tròn là một đỉnh đồ thị, trọng số cạnh là khoảng cách giữa hai đường tròn.

Hai đường tròn giao nhau (khoảng cách 0) nếu

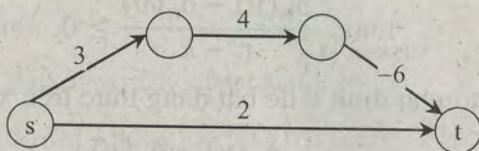
$$|R_1 - R_2| \leq |O_1 O_2| \leq |R_1 + R_2|,$$

ở đây R_1 và R_2 là bán kính của hai đường tròn và $|O_1 O_2|$ là khoảng cách giữa hai tâm.

Nếu $|O_1O_2| > |R_1 + R_2|$, khoảng cách giữa chúng là $|O_1O_2| - |R_1 + R_2|$;
 Nếu $|O_1O_2| < |R_1 - R_2|$, khoảng cách giữa chúng là $|R_1 - R_2| - |O_1O_2|$.
 Dùng thuật toán Dijkstra tìm đường đi ngắn nhất.

7.8. Hướng dẫn

Phản ví dụ



Như ví dụ trên, đỉnh t sẽ được cố định nhãn $d[t] = 2$, nhưng trên thực tế $\delta(s, t) = 1$.

7.9. Hướng dẫn

Để ý rằng nếu một nhãn $d[v] < +\infty$ thì nhãn này phải là số nguyên nằm trong khoảng $[0 \dots (n - 1) \times k]$, đồng thời nếu xét nhãn khoảng cách của các đỉnh lấy ra khỏi hàng đợi ưu tiên thì các nhãn khoảng cách này được sắp xếp theo thứ tự không giảm. Ta tổ chức hàng đợi ưu tiên dưới dạng bảng băm: $q[0 \dots (n - 1) \times k]$, trong đó $q[x]$ là chốt của một danh sách mốc nối chứa các đỉnh v mà $d[v] = x$. Khi đó các thủ tục chèn, cập nhật trên hàng đợi ưu tiên chỉ mất thời gian thực hiện là $O(1)$. Thao tác lấy ra một phần tử trong hàng đợi ưu tiên mất thời gian thực hiện là $O(kn)$.

7.10. Hướng dẫn

Để ý rằng, tại mỗi bước của thuật toán Dijkstra có tối đa $k + 2$ giá trị khác nhau của các nhãn $d[v]$ trong hàng đợi ưu tiên. Mỗi giá trị x sẽ cho tương ứng với một danh sách mốc nối các nút v mà $d[v] = x$. Các chốt của danh sách mốc nối được lưu trữ trong một Binary Heap. Khi đó các thủ tục đầy vào, lấy ra, co nhãn khoảng cách được thực hiện trong thời gian $O(\log k)$.

7.11. Chứng minh và các bước xây dựng thuật toán

- a) Nếu $\delta(s, v) \leq |E|$, ta xây dựng hàng đợi ưu tiên gồm $|E| + 1$ danh sách mốc nối kép đánh số từ 0 tới $|E|$. Ban đầu các danh sách mốc nối kép khởi tạo bằng \emptyset , riêng danh sách 0 có đỉnh s và danh sách 0 là danh sách hoạt tính.

Để lấy đỉnh có nhãn nhỏ nhất khỏi hàng đợi ưu tiên: Ta xét danh sách hoạt tính hiện tại (danh sách i), nếu danh sách rỗng thì chuyển sang danh sách kế tiếp (danh sách $i + 1$) cho tới khi danh sách hoạt tính khác rỗng. Lấy ra một phần tử bất kì trong danh sách hoạt tính để trả về và loại bỏ phần tử đó trong danh sách hoạt tính. Giả sử thuật toán Dijkstra thực hiện tới khi hàng đợi ưu tiên rỗng, khi đó mỗi thao tác xoá đỉnh trong danh sách mốc nối kép được thực hiện trong thời gian $O(1)$, phép dịch con trỏ danh sách hoạt tính xét trong tổng thể thuật toán thực hiện không quá $|E|$ lần.

Để cập nhật nhãn của một đỉnh $d[v] := k$. Nếu v đang nằm trong hàng đợi ưu tiên, xoá nút v khỏi danh sách mốc nối kép chứa nó. Cuối cùng chèn v vào danh sách k . Mỗi thao tác cập nhật thực hiện trong thời gian $O(1)$.

Vậy ta có thuật toán với độ phức tạp $O(|E|)$.

- b) Các giá trị $\delta_1(s, v)$ có thể tính được bằng thuật toán Dijkstra trong thời gian $O(|E|)$. Thật vậy, kết quả này suy ra trực tiếp từ phần a) bởi $w(e) \in \{0, 1\}, \forall e \in E$ nên $\delta_1(s, v) \in [0, |E|]$.
- c) Từ tính chất: $w_{i-1}(u, v)$ bao gồm $i - 1$ bit đầu tiên trong số i bit của $w(u, v)$, ta có:

$$2w_{i-1}(u, v) \leq w_i(u, v) \leq 2w_{i-1}(u, v) + 1.$$

Ngoài ra:

- Nếu mọi cạnh đồ thị được nhân trọng số lên hai lần thì độ dài đường đi ngắn nhất tăng lên hai lần;
- Nếu mọi cạnh đồ thị được cộng trọng số lên không quá 1 thì độ dài đường đi ngắn nhất tăng lên không quá $|V| - 1$.

Ta suy ra:

$$\begin{aligned} 2\delta_{i-1}(s, v) &\leq \delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1 \\ &< 2\delta_{i-1}(s, v) + |V|. \end{aligned}$$

- d) Ta có:

$$\begin{aligned} \hat{w}_i(u, v) &= w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v) \\ &\geq 2w_{i-1}(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v) \geq 0. \end{aligned}$$

Theo bất đẳng thức tam giác: $\delta_{i-1}(s, u) + w_{i-1}(u, v) \geq \delta_{i-1}(s, v)$.

e) Với $P = \langle x_1, x_2, \dots, x_k \rangle$ là một đường đi từ x_1 tới x_k . Xét tổng trọng số các cạnh trên P theo hàm trọng số \widehat{w}_i :

$$\widehat{w}_i(x_1, x_2) = w_i(x_1, x_2) + 2\delta_{i-1}(s, x_1) - 2\delta_{i-1}(s, x_2)$$

$$\widehat{w}_i(x_2, x_3) = w_i(x_2, x_3) + 2\delta_{i-1}(s, x_2) - 2\delta_{i-1}(s, x_3)$$

...

$$\widehat{w}_i(x_{k-1}, x_k) = w_i(x_{k-1}, x_k) + 2\delta_{i-1}(s, x_{k-1}) - 2\delta_{i-1}(s, x_k)$$

$$\widehat{w}_i(P) = w_i(P) + 2\delta_{i-1}(s, x_1) - 2\delta_{i-1}(s, x_k).$$

Điều này chỉ ra rằng nếu P là đường đi ngắn nhất từ x_1 tới x_k theo hàm trọng số w_i thì nó cũng là đường đi ngắn nhất từ x_1 tới x_k theo hàm trọng số \widehat{w}_i . Nếu P là đường đi ngắn nhất từ s tới v theo hàm trọng số w_i (hay hàm trọng số \widehat{w}_i), thay $x_1 = s$ và $x_k = v$, ta có:

$$\hat{\delta}_i(s, v) = \delta_i(s, v) - 2\delta_{i-1}(s, v) < |V| \leq |E|.$$

f) Thuật toán Gabow:

- Để tính δ_i từ các δ_{i-1} trong thời gian $O(|E|)$;
- Tính các trọng số \widehat{w}_i trong thời gian $O(|E|)$;
- Theo phần e), $\hat{\delta}_i(s, v) \leq |E|$, ta có thể dùng kết quả phần a) để tính các $\hat{\delta}_i(s, v)$ trong thời gian $O(|E|)$;
- Tính các $\delta_i(u, v)$ từ các $\delta_{i-1}(u, v)$ và các $\hat{\delta}_i(u, v)$ trong thời gian $O(|V|)$.

Quá trình này được thực hiện với $i = 2, 3, \dots, z$.

Thuật toán Gabow có độ phức tạp $O(|E| \cdot z) = O(|E| \lg k)$.

7.12. Hướng dẫn

Coi mỗi ô là một đỉnh của đồ thị, mỗi ô có cạnh nối tới các ô lân cận, trọng số cạnh là số ghi trên ô lân cận.

Dùng thuật toán Dijkstra tìm đường đi ngắn nhất, thuật toán dừng ngay khi một ô biên được cố định nhãn.

7.13. Hướng dẫn

Coi mỗi chi số là một đỉnh của đồ thị, đỉnh i có cung tới đỉnh j nếu $i < j$ và $a_i + a_j \in \mathbb{P}$ (được định nghĩa là tập các số nguyên tố).

Trọng số tất cả các cung đặt bằng -1 .

Thêm đỉnh 0 nối tới mọi đỉnh khác bằng cung trọng số 0 , thêm đỉnh $n + 1$ nối từ mọi đỉnh khác bằng cung trọng số 0 .

Tìm đường đi ngắn nhất từ đỉnh 0 tới đỉnh n , các đỉnh đi qua trên đường đi ứng với chỉ số các phần tử được chọn.

Đây là đồ thị không có chu trình, thuật toán tìm đường đi ngắn nhất có thể áp dụng là thuật toán đường găng (critical path).

7.14. Hướng dẫn

Xây dựng đồ thị có hướng $G = (V, E)$, mỗi đỉnh tương ứng với một công đoạn, đỉnh u có cung nối tới đỉnh v nếu công đoạn u phải hoàn thành trước khi công đoạn v bắt đầu. Thêm vào G một đỉnh s và cung nối từ s tới tất cả các đỉnh còn lại. Gán trọng số mỗi cung (u, v) của đồ thị bằng t_v .

Nếu đồ thị có chu trình, không thể có cách xếp lịch, nếu đồ thị không có chu trình (DAG) thì tìm đường đi dài nhất xuất phát từ s tới tất cả các đỉnh của đồ thị, khi đó nhãn khoảng cách $d[v]$ chính là thời điểm hoàn thành công đoạn v , ta chỉ cần xếp lịch để công đoạn v được bắt đầu vào thời điểm $d[v] - t_v$.

7.15. Hướng dẫn

Dùng thuật toán Floyd hoặc Johnson xác định tất cả các $\delta(i, j)$ là độ dài đường đi ngắn nhất từ i tới j .

Nếu đồ thị là có hướng, chu trình cần tìm có độ dài bằng

$$\min_{i,j}\{\delta(i, j) + \delta(j, i)\}.$$

Nếu đồ thị là vô hướng, với mỗi cạnh (u, v) mà $\delta(u, v) \neq w(u, v)$, chu trình nhỏ nhất chứa cạnh (u, v) có độ dài bằng $\delta(u, v) + w(v, u)$. Nếu $\delta(u, v) = w(u, v)$, chu trình nhỏ nhất chứa cạnh (u, v) được tính theo thuật toán:

- Loại bỏ cạnh (u, v) hoặc đặt trọng số của nó bằng $+\infty$.
- Dùng thuật toán Dijkstra tìm đường đi ngắn nhất P từ v tới u , trọng số chu trình nhỏ nhất chứa cạnh (u, v) là $w(P) + w(u, v)$.

Việc còn lại là tìm chu trình nhỏ nhất (duyệt mọi cạnh (u, v) và thực hiện thuật toán trên).

7.16. Chứng minh

Nếu trừ trọng số cạnh (u, v) đi Δ ($\Delta > 0$). Trong số các cây khung của đồ thị, nếu cây khung không chứa cạnh (u, v) thì trọng số của nó không đổi. Nếu cây khung có chứa cạnh (u, v) thì trọng số của cây giảm đi Δ . Điều này chỉ ra rằng nếu ban đầu T là cây khung nhỏ nhất của G thì sau khi giảm trọng số cạnh (u, v) , T vẫn là cây khung nhỏ nhất của G .

7.17. Chứng minh

Nếu tồn tại cây khung nhỏ nhất của G không chứa e suy ra việc loại bỏ e không ảnh hưởng tới trọng số của cây khung nhỏ nhất.

Nếu mọi cây khung nhỏ nhất của G đều chứa e thì với một cây T là cây khung nhỏ nhất bất kì, loại bỏ e sẽ tách T làm hai thành phần liên thông. Trên chu trình C , chắc chắn có một cạnh $e' \neq e$ nối liền hai thành phần liên thông đó và $w(e') \leq w(e)$, thêm e' nối liền hai thành phần liên thông ta thu được cây T' có trọng số $w(T') \leq w(T)$ và T' không chứa e , mâu thuẫn với giả thiết mọi cây khung nhỏ nhất đều phải chứa e .

7.18. Chứng minh

Nếu với mọi lát cắt của đồ thị, có duy nhất một cạnh nhẹ nối hai tập của lát cắt. Khi đó xét hai cây khung nhỏ nhất T và T' , ta sẽ chứng minh rằng $T \equiv T'$. Thực vậy:

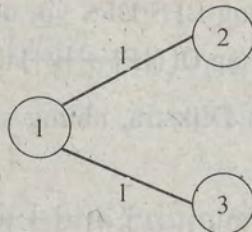
Xét một cạnh $(u, v) \in T$. Loại bỏ cạnh này cây T sẽ tách thành hai thành phần liên thông X, Y . Xét lát cắt (X, Y) .

Trước hết, ta chỉ ra rằng (u, v) là cạnh nhẹ ứng với lát cắt (X, Y) . Thực vậy nếu (u, v) không phải cạnh nhẹ của (X, Y) , ta thêm cạnh nhẹ nối hai thành phần liên thông X, Y được cây khung có trọng số nhỏ hơn T . Mâu thuẫn.

Từ (u, v) là cạnh nhẹ duy nhất với lát cắt (X, Y) . Ta chứng minh rằng $(u, v) \in T'$. Thực vậy, nếu $(u, v) \notin T'$, xét một đường đi trên T' từ u sang v . Đường đi này xuất phát từ X đi sang Y nên chắc chắn phải đi qua một cạnh nối X với Y kí hiệu (x, y) . Bởi $w(u, v) < w(x, y)$. Thêm (u, v) vào T'

và xoá (x, y) khỏi T' , ta được cây khung mới có trọng số nhỏ hơn T' . Điều này mâu thuẫn.

Phản ví dụ cho giả thuyết ngược lại chỉ ra trong hình dưới đây. Cây khung nhỏ nhất là duy nhất nhưng lát cắt $\{1\} - \{2,3\}$ có hai cạnh nhẹ.



7.19. Hướng dẫn

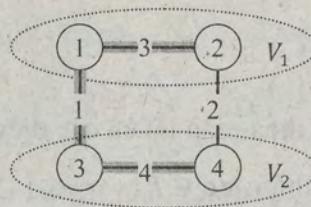
Gọi cạnh bị giảm trọng số là (u, v) , thêm (u, v) vào T ta sẽ được đúng một chu trình đơn, loại bỏ cạnh trọng số lớn nhất trên chu trình đơn này sẽ được cây khung nhỏ nhất của đồ thị mới.

7.20. Hướng dẫn

Chỉ xét những cạnh thuộc T ($n - 1$ cạnh) và những cạnh thêm vào (không quá n cạnh). Áp dụng thuật toán Kruskal hoặc Prim.

7.21. Cách giải

Phản ví dụ



Nếu tách tập đỉnh làm hai tập $V_1 = \{1, 2\}$ và $V_2 = \{3, 4\}$, và thực hiện cách làm trên, ta thu được cây khung nhỏ nhất có trọng số 8, trong khi đó cây khung nhỏ nhất của đồ thị ban đầu có trọng số 6 (cạnh $(1,2)$, $(1,3)$ và $(2,4)$).

7.22. Hướng dẫn

Dựa trên cây khung nhỏ nhất T , thử tất cả các cách thay cạnh trong cây T bởi cạnh ngoài cây T . Mỗi phép thử có thể thực hiện trong thời gian $O(1)$ bởi thuật toán LCA (Least Common Ancestor).

7.23. Hướng dẫn

Cách 1. Dùng thuật toán tìm kiếm nhị phân xác định số Δ nhỏ nhất thỏa mãn: Tồn tại đường đi từ s tới t mà mọi cạnh trên đường đi đều có trọng số nhỏ hơn hoặc bằng Δ . Việc kiểm chứng có tồn tại đường đi như vậy hay không được thực hiện bằng BFS/DFS với độ phức tạp $O(|E| + |V|)$. Toàn bộ thuật toán có độ phức tạp $O((|E| + |V|) \lg W)$.

Cách 2. Dùng thuật toán Dijkstra, nhưng sửa lại công thức phép co trên cung (u, v) từ

$$d[v] = \min\{d[v], d[u] + w(u, v)\}$$

thành

$$d[v] := \min\{d[v], \max\{d[u], w(u, v)\}\}.$$

Độ phức tạp $O(|V| \lg |V| + |E|)$.

Cách 3. Dùng thuật toán Prim với cây T khởi đầu chỉ gồm mỗi đỉnh $\{s\}$. Lần lượt mở rộng cây T theo thuật toán cho tới khi $t \in T$, đường đi từ s tới t trên cây T là đường đi cần tìm. Độ phức tạp $O(|V| \lg |V| + |E|)$.

7.26. Hướng dẫn

Bằng kĩ thuật lấy lôgarit, độ an toàn trên lưới điện trở thành tổng độ an toàn trên các đường dây. Vấn đề còn lại đơn thuần là tìm cây khung lớn nhất

7.27. Chứng minh

Ta chứng minh rằng f_α thoả mãn tất cả các điều kiện của luồng. Thật vậy:

- Ràng buộc về sức chứa: $\forall e \in E$:

$$\begin{aligned} f_\alpha(e) &= \alpha f_1(e) + (1 - \alpha) f_2(e) \\ &\leq \alpha c(e) + (1 - \alpha) c(e) \\ &= c(e). \end{aligned}$$

- Ràng buộc về tính đối xứng lệch: $\forall e \in E$:

$$\begin{aligned} f_\alpha(e) &= \alpha f_1(e) + (1 - \alpha) f_2(e) \\ &= -[\alpha f_1(-e) + (1 - \alpha) f_2(-e)] \\ &= -f_\alpha(-e). \end{aligned}$$

- Ràng buộc về tính bảo tồn: $\forall u \in V - \{s, t\}$:

$$\begin{aligned} \sum_{e=(u,v)} f_\alpha(e) &= \sum_{e=(u,v)} \alpha f_1(e) + (1-\alpha) f_2(e) \\ &= \underbrace{\alpha \sum_{e=(u,v)} f_1(e)}_{=0} + (1-\alpha) \underbrace{\sum_{e=(u,v)} f_2(e)}_{=0} \\ &= 0. \end{aligned}$$

7.28. Chứng minh

$$\begin{aligned} c_f(e) + c_f(-e) &= c(e) - f(e) + c(-e) - f(-e) \\ &= c(e) + c(-e) - \left(\underbrace{f(e) - f(-e)}_{=0} \right) \\ &= c(e) + c(-e). \end{aligned}$$

7.29. Chứng minh

Rõ ràng $s \in X$ và $t \in Y$.

Trước hết, ta chứng minh với mọi cung $e = (u, v)$ nối từ X sang Y đều là cung bão hòa ($c(e) = f(e)$).

Thật vậy, nếu e là cung thăng dư thì từ $u \in X$ ta có thể đi sang t bằng một đường thăng dư, trái với cách xây dựng lát cắt.

Sức chứa của lát cắt (X, Y) bằng

$$c(X, Y) = \sum_{e \in X \rightarrow Y} c(e) = \sum_{e \in X \rightarrow Y} f(e) = f(X, Y) = |f|.$$

Giá trị luồng thông qua lát cắt nhỏ hơn hoặc bằng tổng sức chứa các cung thông qua lát cắt.

Dấu “=” xảy ra khi f là luồng cực đại và (X, Y) là lát cắt hẹp nhất.

7.30. Hướng dẫn

Coi s là đỉnh phát và t là đỉnh thu, các cung đều có sức chứa 1. Tìm luồng cực đại trên mạng bằng thuật toán Ford-Fulkerson, theo Định lí 7.15 (định lí về tính nguyên), luồng trên các cung chỉ có thể là 0 hoặc 1. Loại bỏ các cung có luồng 0 và chỉ giữ lại các cung có luồng 1. Tiếp theo ta tìm một đường đi từ s tới t , chọn đường đi này vào tập hợp, loại bỏ tất cả các cung

dọc trên đường đi này khỏi đồ thị và lặp lại, ..., thuật toán sẽ kết thúc khi đồ thị không còn cạnh nào (không còn đường đi từ s tới t).

Về kĩ thuật cài đặt, ta có thể tìm một đường đi từ s tới t trên đồ thị G , đảo chiều tất cả các cung trên đường đi này và lặp lại cho tới khi không còn đường đi từ s tới t nữa. Có thể thấy rằng đồ thị G tại mỗi bước chính là đồ thị các cung thặng dư và đường đi tìm được ở mỗi bước chính là đường tăng luồng.

Đồ thị G giờ đây không còn đường đi từ s tới t , ta tìm một đường đi từ t về s , kết nạp đường đi theo chiều ngược lại (từ s tới t) vào tập hợp, xoá bỏ tất cả các cung trên đường đi và cứ tiếp tục như vậy cho tới khi không còn đường đi từ t về s nữa.

7.31. Hướng dẫn

Ta cũng quy về bài toán luồng, tuy nhiên trên đồ thị vô hướng, khi thay thế mỗi cạnh bởi hai cung có hướng ngược chiều, ta có thể coi chúng là cung đối của nhau (cả hai đều có sút chúa 1).

7.32. Hướng dẫn

Xây dựng một mạng trong đó tập đỉnh V gồm ba lớp đỉnh S , X và T :

- Lớp đỉnh phát $S = \{s_1, s_2, \dots, s_n\}$, mỗi đỉnh tương ứng với một bạn nam.
- Lớp đỉnh $X = \{x_1, x_2, \dots, x_n\}$, mỗi đỉnh tương ứng với một món quà.
- Lớp đỉnh thu $T = \{t_1, t_2, \dots, t_n\}$, mỗi đỉnh tương ứng với một bạn nữ.

Nếu bạn nam i thích món quà k , ta cho cung nối từ s_i tới x_k , nếu bạn nữ j thích món quà k , ta cho cung nối từ x_k tới t_j . Sút chúa của các cung đặt bằng 1 và sút chúa của các đỉnh cũng đặt bằng 1. Tìm luồng nguyên cực đại trên mạng G có n đỉnh phát, n đỉnh thu, đồng thời có cả ràng buộc sút chúa trên các đỉnh, những cung có luồng 1 sẽ nối giữa một món quà và người tặng/nhận tương ứng.

7.33. Hướng dẫn

Xây dựng mạng với các đỉnh là các nút lưới, các đỉnh phát là nguồn điện và các đỉnh thu là các thiết bị sử dụng điện. Mỗi nút có cạnh nối tới các

nút lân cận và sức chứa của cạnh đó đặt bằng ∞ . Sức chứa của các đỉnh đặt bằng 1..

Tìm luồng cực đại trên mạng, mỗi đơn vị luồng chuyển từ nguồn điện tới một thiết bị sử dụng điện cho ta một dây nối. Rõ ràng không có hai dây nối nào cắt nhau vì mỗi nút lối chỉ có tối đa luồng 1 đi qua.

7.35. Xây dựng thuật toán và cài đặt chương trình

Dựng đồ thị hai phía $G = (X \cup Y, E)$ với X là tập thợ và Y là tập các công việc. Dùng thuật toán tìm kiếm nhị phân xác định số k nhỏ nhất sao cho tồn tại cách phân công mà không thợ nào được phân công quá k việc.

Để xác định có hay không tồn tại cách phân công mà không thợ nào được phân công quá k việc, ta sửa đổi một chút thuật toán tìm bộ ghép cực đại trên đồ thị hai phía. Cụ thể là tại mỗi bước lặp ta cần tìm đường mở xuất phát từ một thợ đang làm ít hơn k việc tới một việc chưa được phân công cho ai.

Input: Tập văn bản ASSIGN.INP bao gồm:

- Dòng 1 chứa hai số nguyên dương m, n ($m, n \leq 10^5$);
- Tiếp theo là không quá 10^5 dòng, mỗi dòng có dạng hai số nguyên i, j cho biết người thợ i có thể làm được công việc j ($1 \leq i \leq m; 1 \leq j \leq n$).

Output: Tập văn bản ASSIGN.OUT bao gồm:

- Dòng 1 ghi thời gian nhanh nhất để hoàn thành tất cả các công việc;
- Dòng 2 ghi n số nguyên, số thứ j là số hiệu người thợ được giao thực hiện công việc j .

Ví dụ:

ASSIGN.INP	ASSIGN.OUT
4 4	2
1 1	1 1 2 2
1 2	
2 3	
2 4	
3 4	
4 4	

Chương trình

```
{$MODE OBJFPC}
{$M 20000000}
program Assignment;
const
  InputFile = 'ASSIGN.INP';
  OutputFile = 'ASSIGN.OUT';
  maxMN = 100000;
  maxK = 100000;
var
  adj: array[1..maxK] of Integer;
  link: array[1..maxK] of Integer;
  head: array[1..maxMN] of Integer;
  m, n, k: Integer;
  match: array[1..maxMN] of Integer;
  deg: array[1..maxMN] of Integer;
  res: Integer;
  avail: array[1..maxMN] of Boolean;
  z: array[1..maxMN] of Integer;
  nz: Integer;
procedure Enter;
var
  f: TextFile;
  u, v: Integer;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, m, n);
    k := 0;
    FillDWord(head[1], m, 0);
    while not SeekEof(f) do
      begin
        ReadLn(f, u, v);
        Inc(k); adj[k] := v;
        link[k] := head[u];
        head[u] := k;
      end;
  end;
```

```

    finally
      CloseFile(f);
    end;
end;
procedure Init;
begin
  FillDWord(match[1], n, 0);
  FillDWord(deg[1], m, 0);
end;
function RefineMatch(threshold: Integer): Boolean;
var
  x, i, y: Integer;
  count: Integer;
begin
  nz := 0;
  count := 0;
  for x := 1 to m do
    begin
      i := head[x];
      while i <> 0 do
        begin
          y := adj[i];
          if (deg[x] < threshold) and (match[y] = 0) then
            begin
              match[y] := x;
              Inc(deg[x]);
            end;
          if (deg[x] > threshold) and (match[y] = x) then
            begin
              match[y] := 0;
              Dec(deg[x]);
            end;
          i := link[i];
        end;
      if deg[x] < threshold then
        begin
          Inc(nz);
          z[nz] := x;
        end;
    end;
  end;

```

```

        Inc(count, deg[x]);
    end;
    Result := count = n;
end;
function TryMatch(threshold: Integer): Boolean;
var
    i: Integer;
    s, y: Integer;
    Stop, Found: Boolean;
procedure Visit(x: Integer);
var
    i, y: Integer;
begin
    i := head[x];
    while i <> 0 do
        begin
            y := adj[i];
            if avail[y] then
                begin
                    avail[y] := False;
                    if match[y] = 0 then Found := True
                    else Visit(match[y]);
                    if Found then
                        begin
                            match[y] := x;
                            Exit;
                        end;
                end;
            i := link[i];
        end;
    end;
begin
    if RefineMatch(threshold) then
        Exit(True);
    repeat
        FillChar(avail[1], n, True);
        Stop := True;
        for i := nz downto 1 do
            begin

```

```

s := z[i]; //deg[s] < threshold
repeat
    Found := False;
    Visit(s);
    if Found then
        begin
            Inc(deg[s]);
            Stop := False;
            if deg[s] = threshold then
                begin
                    z[i] := z[nz];
                    Dec(nz);
                end;
        end
    else
        Break;
    until deg[s] = threshold;
end;
until Stop;
for y := 1 to n do
    if match[y] = 0 then Exit(False);
Result := True;
end;
procedure Solve;
var
    Low, Middle, High: Integer;
begin
    Low := (n - 1) div m + 1;
    High := n - 1;
    while Low <= High do //Low - 1: Không được, High + 1: Được
        begin
            Middle := (Low + High) div 2;
            if TryMatch(Middle) then High := Middle - 1
            else Low := Middle + 1;
        end;
    //Low = High + 1;
    TryMatch(Low);
    Res := Low;
end;

```

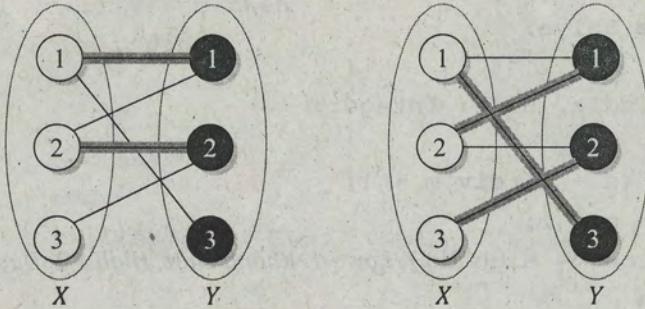
```

procedure PrintResult;
var
  f: TextFile;
  y: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    WriteLn(f, Res);
    for y := 1 to n do Write(f, match[y], ' ');
  finally
    CloseFile(f);
  end;
end;
begin
  Enter;
  Init;
  Solve;
  PrintResult;
end.

```

7.36. Cách giải

- a) Ví dụ về bộ ghép tối đại và bộ ghép cực đại



- b) Thuật toán tìm một bộ ghép tối đại

Xét mọi cạnh (x_i, y_j) , nếu cả x_i và y_j chưa được ghép thì ta ghép x_i với y_j .

Dễ dàng chứng minh được tính đúng đắn và độ phức tạp của thuật toán là $O(|E|)$.

c) *Chứng minh*

Xét hai bộ ghép tối đại A, B . Với mọi cạnh $(x_i, y_j) \in A$, ít nhất một trong hai đỉnh x_i, y_j phải được ghép trong B , bởi nếu cả hai đỉnh này đều chưa ghép trong B , ta bổ sung cạnh (x_i, y_j) vào B được bộ ghép mới lớn hơn B và chứa B .

Từ đó suy ra trong B có ít nhất $|A|$ đỉnh được ghép, hay nói cách khác lực lượng của B tối thiểu phải bằng $\frac{|A|}{2}$.

Tức là $|A| \leq 2|B|$.

Chứng minh $|B| \leq 2|A|$ hoàn toàn tương tự.

Ngoài ra, nếu bộ ghép cực đại A chắc chắn là bộ ghép tối đại, thì việc khởi tạo bộ ghép ban đầu là một bộ ghép tối đại B , số lần tìm đường mở sẽ là

$$|A| - |B| \leq |A| - \frac{|A|}{2} = \frac{|A|}{2}.$$

7.37. *Chứng minh*

- a) Nếu tập C không phải là phủ định, tồn tại một cạnh (x, y) ($x \in X$ và $y \in Y$) mà cả x và y đều không thuộc C .

Theo định nghĩa, $C = (T \cap X) \cup (S \cap Y)$ đồng thời ta có $x \notin T$ và $y \notin S$ tức là $x \in S$ và $y \in T$. Điều này mâu thuẫn với giả thiết (S, T) là lát cắt hẹp nhất (có sức chứa hữu hạn), trong khi (x, y) lại là một cạnh thông qua lát cắt mang sức chứa $+\infty$.

- b) Với phủ định C' bất kì, gọi $T' = (C' \cap X) \cup \{t\}$ và $S' = (C' \cap Y) \cup \{s\}$. Sức chứa của lát cắt (S', T') đúng bằng $|C'|$.

Ngoài ra, sức chứa của lát cắt (S, T) đúng bằng $|C|$ nên C là phủ định nhỏ nhất do (S, T) là lát cắt hẹp nhất.

- c) Suy ra trực tiếp từ cách xây dựng hai phía của lát cắt.

7.38. *Hướng dẫn*

Chọn A là tập các X _đỉnh đến được từ một X _đỉnh chưa ghép bằng một đường pha.

7.42. Hướng dẫn

Gọi d là số nhỏ nhất để $2^d \geq k$. Bổ sung vào đồ thị một số cạnh để mọi đỉnh đều có bậc 2^d . Các cạnh thêm vào được gọi là cạnh giả. Mỗi thành phần liên thông của đồ thị mới là một đồ thị Euler, trên mỗi thành phần đó ta tìm chu trình Euler và dọc trên chu trình Euler ta tô màu xanh/đỏ xen kẽ cho các cạnh. Thực hiện điều này mất thời gian là $O(m)$.

Các cạnh xanh được tách ra và lập thành đồ thị G_1 , các cạnh đỏ được tách ra lập thành đồ thị G_2 . Không giảm tính tổng quát, giả sử số cạnh giả trong G_1 không nhiều hơn số cạnh giả trong G_2 .

Xét đồ thị G_1 , bậc của mỗi đỉnh là 2^{d-1} . Lại thực hiện tương tự trên, ta thu được đồ thị G_{11} mà bậc của mỗi đỉnh là $2^{d-2} \dots$

Sau d lần như vậy, ta thu được một đồ thị không có cạnh giả nào mà bậc của mọi đỉnh đều bằng 1. Theo định nghĩa đây chính là bộ ghép đầy đủ.

Thời gian thực hiện thuật toán là $O(md)$ hay $O(m \lg m)$.

7.43. Xây dựng thuật toán và cài đặt chương trình

Dựng đồ thị $G = (V, E)$ với n đỉnh ứng với n biến. Mỗi ràng buộc dạng $v_j - v_i \leq c_{ij}$ ứng với một cung nối từ đỉnh i tới đỉnh j với trọng số c_{ij} . Bổ sung thêm một đỉnh số 0 nối tới mọi đỉnh khác bằng cung trọng số 0.

Nhận xét: Nếu đồ thị có chu trình âm thì không tồn tại cách gán giá trị cho các biến thoả mãn tất cả các ràng buộc.

Chứng minh: Giả sử có chu trình âm $C = (1, 2, \dots, k, 1)$. Vì có các cung $(1, 2); (2, 3); \dots; (k-1, k); (k, 1)$ nên theo cách gán trọng số, ta có:

$$v_2 - v_1 \leq c_{12}$$

$$v_3 - v_2 \leq c_{23}$$

...

$$v_k - v_{k-1} \leq c_{k-1,k}$$

$$v_1 - v_k \leq c_{k1}$$

Cộng vế với vế các bất đẳng thức trên, vế trái (bằng 0) không thể nhỏ hơn hoặc bằng vế phải là trọng số chu trình C (nhỏ hơn 0).

Nếu đồ thị không có chu trình âm, thuật toán Bellman-Ford có thể áp dụng để xác định $d[v]$ là độ dài đường đi ngắn nhất từ 1 tới v ($\forall v = 1, 2, \dots, n$). Từ bất đẳng thức tam giác: $\forall (i, j) \in E: d[j] \leq d[i] + c_{ij}$, ta có

$$d[j] - d[i] \leq c_{ij}.$$

Tức là các giá trị $d[.]$ thoả mãn tất cả các ràng buộc. Việc cuối cùng là cộng thêm các $d[.]$ cùng một hằng số để các giá trị này rơi vào trong khoảng $[a, b]$ (nếu $d[.]_{max} - d[.]_{min} > b - a$ thì cũng không tồn tại cách gán giá trị biến theo yêu cầu đề bài).

Chương trình

```

{$MODE OBJFPC}
program SystemOfDifferenceConstrains;
const
  InputFile  = 'SDC.INP';
  OutputFile = 'SDC.OUT';
  maxN = 1000;
  maxM = 10000;
  maxD = 1000000;
type
  TEdge = record
    u, v, c: Integer;
  end;
var
  d: array[1..maxN] of Integer;
  n, m: Integer;
  e: array[1..maxM] of TEdge;
  a, b: Integer;
  resS: string[3];
procedure Enter;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, n, m, a, b);
  except
  end;
end;
procedure Output;
var
  f: TextFile;
  i: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  for i := 1 to n do
    WriteLn(f, d[i]);
  Close(f);
end;
begin
  Enter;
  Output;
end.

```

```

for i := 1 to m do
  with e[i] do
    ReadLn(f, u, v, c);
  finally
    CloseFile(f);
  end;
end;

function Relaxation: Integer; //1: Continue; 0: Stop; 2: No Res
var
  i: Integer;
begin
  Result := 0;
  for i := 1 to m do
    with e[i] do
      if d[v] > d[u] + c then
        begin
          d[v] := d[u] + c;
          if d[v] < -maxD then
            begin
              Result := 2;
              Exit;
            end;
          Result := 1;
        end;
  end;
procedure FordBellman;
var
  i, t: Integer;
  inf, sup: Int64;
begin
  FillChar(d, SizeOf(d), 0);
  resS := '';
  for i := 1 to n - 1 do
    begin
      t := Relaxation;
      if t <> 1 then Break;
    end;
  if (t = 1) and (Relaxation <> 0) then
    begin

```

```

    resS := 'NO';
    Exit;
end;
resS := 'YES';
inf := d[1];
sup := d[1];
for i := 2 to n do
begin
    if d[i] < inf then inf := d[i];
    if d[i] > sup then sup := d[i];
end;
if sup - inf > b - a then
begin
    resS := 'NO';
    Exit;
end;
for i := 1 to n do
    d[i] := d[i] - inf + a;
end;
procedure PrintResult;
var
    f: TextFile;
    i: Integer;
begin
    AssignFile(f, OutputFile);
    Rewrite(f);
    try
        WriteLn(f, resS);
        if resS = 'YES' then
            for i := 1 to n do Write(f, d[i], ' ');
    finally
        CloseFile(f);
    end;
end;
begin
    Enter;
    FordBellman;
    PrintResult;
end.

```

7.44. Xây dựng thuật toán và cài đặt chương trình

Liệt kê các miền liên thông gồm các ô cần tô cùng màu (gọi tắt là miền đơn sắc) (lưu ý đặt dẫn hướng \$M rồi cài DFS), đánh số các miền đơn sắc này từ 1 trở đi.

- Gọi $component[x, y]$ là số hiệu miền đơn sắc chứa ô (x, y) ;
- Gọi $color[i]$ là màu của miền đơn sắc thứ i , tức là màu của tất cả các ô (x, y) mà $component[x, y] = i$;
- Gọi $area[i]$ là số ô (diện tích) của miền đơn sắc thứ i , tức là số ô (x, y) mà $component[x, y] = i$.

Xét tất cả các “khe” giữa hai ô kề cạnh, tức là với mỗi ô xét ô bên phải nó và bên dưới nó nếu có (tổng cộng có $m(n - 1) + n(m - 1)$ khe như vậy).

Với mỗi khe giữa hai ô khác màu $((x, y)$ và (x', y')), tạo một bản ghi chứa bốn thông tin sau:

- Miền đơn sắc chứa ô (x, y) : $cmp1 = component[x, y]$;
- Màu của ô (x, y) : $c_1 = color[x, y]$;
- Miền đơn sắc chứa ô (x', y') : $cmp2 = component[x', y']$;
- Màu của ô (x', y') : $c_2 = color[x', y']$.

(Lưu ý, không cần lưu c_1 và c_2 , chỉ cần $cmp1$ và $cmp2$ là đủ vì

$$c_1 = color[cmp1] \text{ và } c_2 = color[cmp2].$$

Lưu tất cả các bản ghi này lại, mỗi bản ghi ứng với một khe nối giữa hai miền đơn sắc $cmp1$ (màu c_1) và $cmp2$ (màu c_2). Không làm mất tính tổng quát, giả sử $c_1 < c_2$ (nếu ngược lại thì ta đảo $cmp1$ và $cmp2$ cũng như đảo c_1 và c_2).

Kĩ thuật giảm độ phức tạp ở đây dựa vào nhận xét rằng số khe nhỏ hơn hoặc bằng hai triệu. Và để tìm miền liên thông lớn nhất gồm hai màu XANH và ĐỎ thì ta chỉ cần xét các khe có $c_1 = \text{XANH}$ và $c_2 = \text{ĐỎ}$.

Sắp xếp lại danh sách các khe để các khe có cùng giá trị c_1 và c_2 dồn lại một đoạn liên tiếp trong dãy (đó là lí do ta quy ước $c_1 < c_2$). Xây dựng cấu trúc dữ liệu (CTDL) Disjoint-set forest (DSF) biểu diễn các tập miền đơn sắc, mỗi tập s trong CTDL này đi kèm với giá trị $weight[s]$ là tổng diện tích các miền đơn sắc thuộc tập.

- $\text{MakeSet}(u)$: Tạo tập chỉ chứa một miền đơn sắc u :
 $\text{lab}[u] := 0$;
 $\text{weight}[u] := \text{area}[u]$;
- $\text{FindSet}(u)$: Tìm tập chứa u , như bình thường (đi từ u lên gốc theo $\text{lab}[u]$, nén đường);
- $\text{Union}(r, s)$: Hợp hai tập r, s . Chú ý khi hợp xong thì giá trị weight của tập mới bằng tổng giá trị weight hai tập cũ. Giá trị $\text{weight}[x]$ chỉ có nghĩa khi x là một gốc cây biểu diễn tập hợp trong DSF.
- Khởi tạo: Duyệt từng khe trong đoạn, với mỗi khe nối hai miền đơn sắc $cmp1 - cmp2$, gọi hai thủ tục
 $\text{MakeSet}(cmp1)$ và $\text{MakeSet}(cmp2)$,
cập nhật MaxArea bằng giá trị weight lớn nhất vừa đặt.
- Xử lí: Duyệt từng khe trong đoạn, với mỗi khe $cmp1 - cmp2$ mà
 $\text{FindSet}(cmp1) = r \neq s = \text{FindSet}(cmp2)$ thì gọi thủ tục
 $\text{Union}(r, s)$.

Cập nhật MaxArea theo giá trị weight của tập hợp thành nếu nó lớn hơn MaxArea cũ.

- Kết thúc việc xử lí đoạn liên tiếp MaxArea cho ta diện tích miền lớn nhất chỉ gồm hai màu c_1 và c_2 .

Cuối cùng, xử lí mỗi đoạn liên tiếp cho ta diện tích miền lớn nhất gồm hai màu ứng với hai giá trị c_1, c_2 của các khe trong đoạn đó.

Độ phức tạp: Ngoài các thao tác tìm miền đơn sắc và sắp xếp khe. Việc xử lí một đoạn liên tiếp độ dài l mất thời gian $O(l \times \alpha(l))$. Tổng độ dài các đoạn liên tiếp nhỏ hơn hoặc bằng số khe và bằng $m(n - 1) + n(m - 1)$ (nhỏ hơn hoặc bằng hai triệu).

Chương trình

```
{$MODE OBJFPC}
program Coloring;
const
  InputFile  = 'COLORING.INP';
  OutputFile = 'COLORING.OUT';
  maxMN = 1000;
  maxColor = Round(1E6);
```

```

maxComponents = maxMN * maxMN;
maxLinks = 2 * maxMN * maxMN;
dx: array[1..4] of Integer = (0, 0, 1, -1);
dy: array[1..4] of Integer = (1, -1, 0, 0);
type
  TComponentLink = record
    cmp1, cmp2: Integer;
    code: Int64;
  end;
var
  a: array[0..maxMN + 1, 0..maxMN + 1] of Integer;
  component: array[1..maxMN, 1..maxMN] of Integer;
  nComponents: Integer;
  area, color, weight, lab: array[1..maxComponents] of Integer;
  link: array[1..maxLinks] of TComponentLink;
  nlinks: Integer;
  qx, qy: array[1..maxMN * maxMN] of Integer;
  front, rear: Integer;
  m, n: Integer;
  res: Integer;
  fi, fo: TextFile;
procedure Enter;
var
  i, j: Integer;
begin
  FillChar(a, SizeOf(a), 0);
  ReadLn(fi, m, n);
  for i := 1 to m do
    begin
      for j := 1 to n do Read(fi, a[i, j]);
      ReadLn(fi);
    end;
  end;
procedure GetConnectedComponents;
var
  i, j: Integer;
  s: Integer;
  procedure Visit(x, y: Integer);
  var

```

```

d, newx, newy: Integer;
begin
  Inc(nComponents);
  color[nComponents] := a[x, y];
  qx[1] := x;
  qy[1] := y;
  front := 1;
  rear := 1;
  component[x, y] := nComponents;
  s := 0;
repeat
  x := qx[front];
  y := qy[front];
  Inc(front);
  Inc(s);
  for d := 1 to 4 do
    begin
      newx := x + dx[d];
      newy := y + dy[d];
      if (a[x, y] = a[newx, newy]) and
        (component[newx, newy] = 0) then
        begin
          component[newx, newy] := nComponents;
          Inc(rear);
          qx[rear] := newx;
          qy[rear] := newy;
        end;
    end;
  until front > rear;
  area[nComponents] := s;
end;
begin
  FillChar(component, SizeOf(component), 0);
  for i := 1 to m do
    for j := 1 to n do
      if component[i, j] = 0 then
        Visit(i, j);
end;
procedure GetLinks;

```

```

var
  x, y: Integer;
  procedure AddLink(c1, c2: Integer);
  begin
    Inc(nlinks);
    with link[nlinks] do
      begin
        if color[c1] < color[c2] then
          begin
            cmp1 := c1;
            cmp2 := c2;
            end
        else
          begin
            cmp1 := c2;
            cmp2 := c1;
            end;
        code := Int64(color[cmp1]) * (maxColor+1)+color[cmp2];
        end;
      end;
  begin
    nlinks := 0;
    for x := 1 to m - 1 do
      for y := 1 to n do
        if a[x, y] <> a[x + 1, y] then
          AddLink(component[x, y], component[x + 1, y]);
    for x := 1 to m do
      for y := 1 to n - 1 do
        if a[x, y] <> a[x, y + 1] then
          AddLink(component[x, y], component[x, y + 1]);
  end;
  procedure SortLink(L, H: Integer);
  var
    i, j: Integer;
    pivot: TComponentLink;
  begin
    if L >= H then Exit;
    i := L + Random(H - L + 1);

```

```

pivot := link[i];
link[i] := link[L];
i := L;
j := H;
repeat
  while (pivot.code < link[j].code) and (i < j) do
Dec(j);
  if i < j then
    begin
      link[i] := link[j];
      Inc(i);
    end
  else Break;
  while (link[i].code < pivot.code) and (i < j) do
Inc(i);
  if i < j then
    begin
      link[j] := link[i];
      Dec(j);
    end
  else Break;
until i = j;
link[i] := pivot;
SortLink(L, Pred(i));
SortLink(Succ(i), H);
end;
procedure Maxmimize(var target:Integer;value:Integer);inline;
begin
  if target < value then target := value;
end;
procedure Process(L, H: Integer);
var
  i: Integer;
  maxweight, r, s: Integer;
  function FindSet(u: Integer): Integer;
begin
  if lab[u] > 0 then
    begin
      Result := FindSet(lab[u]);

```

```

        lab[u] := Result;
    end
else
    Result := u;
end;
procedure Union(r, s: Integer);
begin
    if lab[r] < lab[s] then
        begin
            lab[s] := r;
            Inc(weight[r], weight[s]);
            Maxmimize(maxweight, weight[r]);
        end
    else
        begin
            if lab[r] = lab[s] then Dec(lab[s]);
            lab[r] := s;
            Inc(weight[s], weight[r]);
            Maxmimize(maxweight, weight[s]);
        end;
    end;
begin
    maxweight := 0;
    for i := L to H do
        with link[i] do
            begin
                lab[cmp1] := 0;
                lab[cmp2] := 0;
                weight[cmp1] := area[cmp1];
                weight[cmp2] := area[cmp2];
                Maxmimize(maxweight, weight[cmp1]);
                Maxmimize(maxweight, weight[cmp2]);
            end;
    for i := L to H do
        with link[i] do
            begin
                r := FindSet(cmp1);
                s := FindSet(cmp2);
                if r <> s then Union(r, s);
            end;

```

```

    end;
maxmimize(res, maxweight);
end;
procedure Solve;
var
  i, j: Integer;
begin
  res := 0;
  for i := 1 to nComponents do
    Maxmimize(res, area[i]);
  j := 0;
  for i := 1 to nlinks do
    if (i = nlinks) or (link[i].code <> link[i+1].code)
    then
      begin
        Process(j + 1, i);
        j := i;
      end;
  end;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  AssignFile(fo, OutputFile);
  Rewrite(fo);
  try
    Enter;
    GetConnectedComponents;
    GetLinks;
    SortLink(1, nLinks);
    Solve;
    Write(fo, res);
  finally
    CloseFile(fi);
    CloseFile(fo);
  end;
end.

```

7.45. Xây dựng thuật toán và cài đặt chương trình

Dựng cây khung của mỗi thành phần liên thông. Với mỗi cạnh không thuộc cây khung nào, nó sẽ nối hai đỉnh thuộc cùng một cây và tạo ra một chu trình đơn, đây chính là một tour cần tìm. Số tour lớn nhất chính là số cạnh không thuộc cây khung nào (bằng $m - n + k$, với m là số cạnh, n là số đỉnh và k là số thành phần liên thông của đồ thị).

Chứng minh (cho số tour lớn nhất):

Xét một thành phần liên thông C_1 gồm n_1 đỉnh và m_1 cạnh. Phương án cho số tour nhiều nhất chắc chắn phải cho số tour hạn chế trên C_1 là nhiều nhất. Gọi S_1 là tập các tour hạn chế trên trên C_1 . Các cạnh nằm trên các tour thuộc S_1 chắc chắn làm cho các đỉnh của C_1 liên thông.

Vì mỗi tour thuộc S_1 là một chu trình, nếu bỏ đi một cạnh trên chu trình thì không làm mất khả năng đi lại giữa hai đỉnh bất kì. Mặt khác, mỗi tour lại chứa một cạnh không thuộc tour nào khác (gọi là cạnh riêng) nên nếu ta bỏ tất cả các cạnh riêng của các tour (tổng cộng $|S_1|$ cạnh) thì các đỉnh của C_1 vẫn liên thông, tức là C_1 còn lại ít nhất $n_1 - 1$ cạnh. Từ đó ta có:

$$m_1 - |S_1| \geq n_1 - 1$$

hay:

$$|S_1| \leq m_1 - n_1 + 1.$$

Tương tự, với các thành phần liên thông C_2, C_3, \dots, C_k , ta có

$$|S_1| + |S_2| + \dots + |S_k| \leq \sum_{i=1}^k m_i - \sum_{i=1}^k n_i + k = m - n + k.$$

Chương trình

Khi cài đặt ta có thể sử dụng DFS để kết hợp tìm cây khung với liệt kê chu trình cơ sở.

```
{$MODE OBJFPC}
program MaximumTours;
const
  InputFile := 'TOURS.INP';
  OutputFile = 'TOURS.OUT';
  max = 20000;
```

```

type
  TIntArr = array of Integer;
var
  adj: array of TIntArr;
  Trace: TIntArr;
  Number: TIntArr;
  n, m: Integer;
  Count: Integer;
  s, t: TIntArr;
  nPath: Integer;
procedure Enter;
var
  f: TextFile;
  i: Integer;
  u, v: TIntArr;
  Count: TIntArr;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, n, m);
    SetLength(Count, n + 1);
    FillChar(Count[1], n * SizeOf(Integer), 0);
    SetLength(u, m);
    SetLength(v, m);
    for i := 0 to m - 1 do
      begin
        ReadLn(f, u[i], v[i]);
        Inc(Count[u[i]]);
        Inc(Count[v[i]]);
      end;
    SetLength(adj, n + 1);
    for i := 1 to n do
      SetLength(adj[i], Count[i]);
    for i := 0 to m - 1 do
      begin
        Dec(Count[u[i]]);
        adj[u[i], Count[u[i]]] := v[i];
        Dec(Count[v[i]]);
      end;
  end;

```

```

        adj[v[i], Count[v[i]]] := u[i];
    end;
finally
    CloseFile(f);
end;
end;

procedure Visit(u: Integer);
var
    iv, v: Integer;
begin
    Inc(Count);
    Number[u] := Count;
    for iv := 0 to High(adj[u]) do
        begin
            v := adj[u, iv];
            if Number[v] = 0 then
                begin
                    Trace[v] := u;
                    Visit(v);
                end
            else
                if (Number[v] < Number[u]) and
                   (Trace[u] <> v) then
                    begin
                        s[nPath] := u;
                        t[nPath] := v;
                        Inc(nPath);
                    end;
        end;
end;
end;

procedure DepthFirstSearch;
var
    i: Integer;
begin
    nPath := 0;
    SetLength(s, m);
    SetLength(t, m);
    SetLength(Trace, n + 1);
    SetLength(Number, n + 1);

```

```
FillChar(Number[1], n * SizeOf(Integer), 0);

Count := 0;
for i := 1 to n do
  if Number[i] = 0 then
    Visit(i);
end;
procedure PrintResult;
var
  f: TextFile;
  i: Integer;
  u, v: Integer;
begin
  AssignFile(f, OutputFile);
  Rewrite(f);
  try
    WriteLn(f, nPath);
    for i := 0 to nPath - 1 do
      begin
        v := t[i];
        u := s[i];
        repeat
          Write(f, u, ' ');
          if u = v then
            Break;
          u := Trace[u];
        until False;
        WriteLn(f, s[i]);
      end;
    finally
      CloseFile(f);
    end;
  end;
begin
  Enter;
  DepthFirstSearch;
  PrintResult;
end.
```

7.46. Xây dựng thuật toán và cài đặt chương trình

Dựng mô hình đồ thị dạng cây có gốc ở đỉnh 1. Gán cho mỗi đỉnh một mã số là số thứ tự của đỉnh đó theo quá trình thăm DFS.

Gọi mã số của đỉnh u là $number[u]$ và độ sâu của u là $depth[u]$.

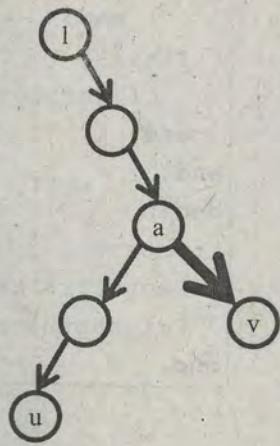
Gọi v_1, v_2, \dots, v_k là một dãy các đỉnh xếp theo thứ tự tăng dần của mã số. Giả sử một người muốn đi thăm các địa điểm thuộc tập $S = \{v_1, v_2, \dots, v_k\}$ thì những nhánh cây không chứa đỉnh nào thuộc S có thể tia bỏ để được một cây con T . Để thấy rằng từ đỉnh 1, để thăm mọi điểm thuộc S thì hành trình buộc phải qua mọi đỉnh của cây con T này. Vì hành trình là một chu trình nên mỗi cạnh (u, v) trên T sẽ phải đi qua ít nhất hai lần, một lần $u \rightarrow v$ và một lần $v \rightarrow u$. Suy ra mọi chu trình qua tất cả các đỉnh phải có độ dài lớn hơn hoặc bằng hai lần số cạnh của T .

Thay mỗi cạnh trên cây con bằng hai cung có hướng ngược chiều, ta được một đồ thị Euler (chu trình Euler trong trường hợp này có thể tìm bằng DFS). Chu trình Euler có độ dài đúng bằng hai lần số cạnh của T nên đây chính là chu trình ngắn nhất cần tìm.

Vấn đề còn lại là xác định số cạnh của cây T . Coi như các cạnh ban đầu đều chưa tô màu. Duyệt các đỉnh thuộc S theo thứ tự tăng dần của $number[.]$. Khi xét một đỉnh u , ta đi từ đỉnh 1 đến đỉnh u và đi qua cạnh nào ta tô màu luôn cạnh đó nếu nó chưa được tô. Khi duyệt qua mọi đỉnh của S thì các cạnh đã tô màu cho ta cây T và số lần tô màu chính là số cạnh của T .

Nhắc lại rằng ta duyệt các đỉnh của S theo thứ tự tăng dần của $number[.]$. Giả sử ta vừa đi từ đỉnh 1 đến đỉnh u và tô màu xong các cạnh trên đường đi, sau đó đi từ đỉnh 1 đến đỉnh v để tô màu tiếp. Gọi a là tiền bối chung thấp nhất của cả u và v ($LCA(u, v)$).

Khi đi từ đỉnh 1 đến đỉnh v , đoạn đường từ đỉnh 1 đến đỉnh a không có cạnh nào phải tô vì những cạnh đó đã tô khi ta đi từ đỉnh 1 đến đỉnh u , đoạn đường từ đỉnh a đến đỉnh v qua cạnh nào ta cũng phải tô



cạnh đó, suy ra số cạnh phải tô khi đi từ đỉnh 1 đến đỉnh v đúng bằng $depth[v] - depth[a]$.

Từ đó suy ra cây T nhỏ nhất chứa đỉnh 1 và các đỉnh v_1, v_2, \dots, v_k có số cạnh bằng:

$$\sum_{i=1}^k depth[v_i] - \sum_{i=2}^k depth[LCA(v_i, v_{i-1})].$$

Giá trị này có thể tính trong thời gian $O(k)$ hay $O(k \log n)$ tùy theo thuật toán tìm LCA được sử dụng.

Chương trình

```
{$MODE OBJFPC}
{$INLINE ON}
{$M 20000000}
program Trip;
const
  InputFile = 'TRIP.INP';
  OutputFile = 'TRIP.OUT';
  maxN = Round(1E5);
  maxLg2N = Trunc(Ln(2 * MaxN) / Ln(2)) + 1;
type
  TEdge = record
    x, y: Integer;
    link: Integer;
  end;
var
  e: array[1 .. maxN..maxN - 1] of TEdge;
  head, depth, map, a: array[1..maxN] of Integer;
  list: array[1..maxN * 2] of Integer;
  nlist: Integer;
  adepth: array[1..maxN * 2, 0..maxLg2N] of Integer;
  avail: array[1..maxN] of Boolean;
  n, na: Integer;
  fi, fo: TextFile;
procedure Enter;
var
  i: Integer;
```

v

```

begin
  ReadLn(fi, n);
  for i := 1 to n - 1 do
    begin
      ReadLn(fi, e[i].x, e[i].y);
      e[-i].x := e[i].y;
      e[-i].y := e[i].x;
    end;
  FillDWord(head[1], n, 0);
  for i := 1 - n to n - 1 do
    if i <> 0 then
      with e[i] do
        begin
          link := head[x];
          head[x] := i;
        end;
    end;
procedure DFS;
procedure Visit(u: Integer; d: Integer);
var
  i: Integer;
begin
  depth[u] := d;
  Inc(nlist);
  list[nlist] := u;
  map[u] := nlist;
  avail[u] := False;
  i := head[u];
  while i <> 0 do
    with e[i] do
      begin
        if avail[y] then
          begin
            Visit(y, Succ(d));
            Inc(nlist);
            list[nlist] := u;
          end;
        i := link;
      end;
end;

```

```

    end;
begin
  FillChar(avail[1], n, True);
  nlist := 0;
  Visit(1, 0);
end;
function lg(n: Integer): Integer; inline;
begin
  Result := Trunc(ln(n) / ln(2));
  if 1 shl (Result + 1) <= n then
    Inc(Result);
end;
procedure Init;
var
  i, j, k, maxk, d1, d2: Integer;
begin
  for i := 1 to nlist do
    adepth[i, 0] := depth[list[i]];
  maxk := lg(nlist);
  for k := 1 to maxk do
    for i := 1 to nlist do
      begin
        d1 := adepth[i, k - 1];
        j := i + 1 shl (k - 1);
        if j <= nlist then d2 := adepth[j, k - 1]
        else d2 := maxN + 1;
        if d1 < d2 then
          adepth[i, k] := d1
        else
          adepth[i, k] := d2;
      end;
end;
procedure QSort(L, H: Integer);
var
  pivot: Integer;
  i, j: Integer;
begin
  if L >= H then Exit;
  i := L + Random(H - L + 1);

```

```

pivot := a[i];
a[i] := a[L];
i := L;
j := H;
repeat
  while (map[a[j]] > map[pivot]) and (i < j) do Dec(j);
  if i < j then
    begin
      a[i] := a[j];
      Inc(i);
    end
  else Break;
  while (map[a[i]] < map[pivot]) and (i < j) do Inc(i);
  if i < j then
    begin
      a[j] := a[i];
      Dec(j);
    end
  else Break;
until i = j;
a[i] := pivot;
QSort(L, Pred(i));
QSort(Succ(i), H);
end;
function LCADepth(u, v: Integer): Integer; inline;
var
  k: Integer;
begin
  u := map[u];
  v := map[v];
  k := lg(v - u + 1);
  v := v - 1 shl k + 1;
  if adepth[u, k] < adepth[v, k] then
    Result := adepth[u, k]
  else
    Result := adepth[v, k];
end;
procedure Solve;
var

```

```

i, j, res: Integer;
begin
  for i := 1 to n do
    begin
      na := 0;
      for j := 1 to n div i do
        begin
          Inc(na);
          a[na] := i * j;
        end;
      QSort(1, na);
      res := depth[a[1]];
      for j := 2 to na do
        Inc(res, depth[a[j]] - LCADepth(a[j - 1], a[j]));
      WriteLn(fo, res shl 1);
    end;
  end;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  AssignFile(fo, OutputFile);
  Rewrite(fo);
  try
    Enter;
    DFS;
    Init;
    Solve;
  finally
    CloseFile(fi);
    CloseFile(fo);
  end;
end.

```

7.47. Xây dựng thuật toán và cài đặt chương trình

Dựng đồ thị, gọi $w(u, v)$ là trọng số cạnh (u, v) .

Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 tới mọi đỉnh khác. Gọi $\pi[v]$ là độ dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh v .

Xét hàm trọng số: $w_\pi(u, v) = w(u, v) + \pi[u] - \pi[v]$. Có ba tính chất của hàm trọng số w_π :

- w_π là một hàm trọng số không âm;
- Những cạnh (u, v) thuộc một đường đi ngắn nhất từ đỉnh 1 đến đỉnh n có $w_\pi(u, v) = 0$;
- Với mọi đường đi $P: 1 \rightsquigarrow n$, ta có $w_\pi(P) = w(P) + \pi(1) - \pi(n)$. Tức là đường đi ngắn nhất theo trọng số w cũng là đường đi ngắn nhất theo trọng số w_π .

Gọi $P: 1 \rightsquigarrow n$ là đường đi ngắn nhất từ đỉnh 1 đến đỉnh n . Dọc trên đường đi, định hướng cạnh (u, v) thành cung (v, u) . Sau đó áp dụng thuật toán Dijkstra tìm Q là đường đi ngắn nhất từ đỉnh 1 đến đỉnh n (theo hàm trọng số w_π). Những cạnh thuộc chỉ một trong hai đường đi P hoặc Q cho ta chương trình cần tìm.

Chương trình

```
{$MODE OBJFPC}
program DoubleTours;
const
  InputFile  = 'WALK.INP';
  OutputFile = 'WALK.OUT';
  maxN = 100000;
  maxM = 100000;
  maxC = 2000000000;
  maxD = maxN * maxC;
type
  TEdge = record
    x, y: Integer;
    w: Integer;
    selected: Boolean;
  end;
  THeap = record
    nItems: Integer;
    items: array[1..maxN] of Integer;
    pos: array[1..maxN] of Integer;
  end;
  TTrace = array[1..maxN] of Integer;
```

```

var
  e: array[-maxM..maxM] of TEdge;
  link: array[-maxM..maxM] of Integer;
  head: array[1..maxN] of Integer;
  d: array[1..maxN] of Int64;
  trace: array[1..maxN] of Integer;
  price: array[1..maxN] of Int64;
  heap: THeap;
  n, m: Integer;
  d1, d2, res: Int64;
  p, q: array[1..maxN] of Integer;
  np, nq: Integer;
procedure Enter;
var
  f: TextFile;
  i, j, u, v: LongInt;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, n, m);
    for i := 1 to m do
      begin
        with e[i] do ReadLn(f, x, y, w);
        j := -i;
        e[j].x := e[i].y;
        e[j].y := e[i].x;
        e[j].w := e[i].w;
      end;
    finally
      CloseFile(f);
    end;
  end;
procedure BuildIncidentLists;
var
  i: Integer;
begin
  FillDWord(head[1], n, 0);
  for i := -m to m do

```

```

if i <> 0 then
  with e[i] do
    begin
      link[i] := head[x];
      head[x] := i;
    end;
end;
procedure Init;
var
  i: Integer;
begin
  FillChar(price[1], n * SizeOf(price[1]), 0);
  for i := -m to m do
    e[i].selected := False;
end;
function wp(const e: TEdge): Int64;
begin
  with e do
    Result := w + price[x] - price[y];
end;
procedure DijkstraInit;
var
  i: LongInt;
begin
  for i := 1 to n do d[i] := maxD + 1;
  d[1] := 0;
  FillDWord(heap.pos[1], n, 0);
  heap.nItems := 1;
  heap.items[1] := 1;
end;
function Extract: LongInt;
var
  p, c: Integer;
  v: Integer;
begin
  with Heap do
    begin
      Result := items[1];
      v := items[nItems];

```

```

Dec(nItems);
p := 1;
repeat
  c := p * 2;
  if (c < nItems) and (d[items[c + 1]] < d[items[c]])
  then Inc(c);
  if (c > nItems) or (d[items[c]] >= d[v]) then
    Break;
  items[p] := items[c];
  pos[items[p]] := p;
  p := c;
until False;
items[p] := v;
pos[v] := p;
end;
end;
function Update(v: Integer; dv: Int64): Boolean;
var
  p, c: Integer;
begin
  Result := d[v] > dv;
  if not Result then Exit;
  d[v] := dv;
  with Heap do
    begin
      c := pos[v];
      if c = 0 then
        begin
          Inc(nItems);
          c := nItems;
        end;
      repeat
        p := c div 2;
        if (p = 0) or (d[items[p]] <= d[v]) then Break;
        items[c] := items[p];
        pos[items[c]] := c;
        c := p;
      until False;
    end;
  end;

```

```

        items[c] := v;
        pos[v] := c;
    end;
end;

function Dijkstra: Boolean;
var
    u, v, i: LongInt;
begin
    DijkstraInit;
    repeat
        u := Extract;
        if u = n then Exit(True);
        i := head[u];
        while i <> 0 do
            begin
                v := e[i].y;
                if not e[i].selected and
                    Update(v, d[u] + wp(e[i]))
                then trace[v] := i;
                i := link[i];
            end;
        until heap.nItems = 0;
        Result := False;
    end;
procedure AdjustPotentials;
var
    u, v, i: LongInt;
begin
    for v := 1 to n do
        if d[v] < d[n] then
            Dec(price[v], d[n] - d[v]);
    v := n;
    repeat
        i := trace[v];
        e[i].selected := True;
        e[-i].w := -e[i].w;
        v := e[i].x;
    until v = 1;
end;

```

```

procedure RemoveEdges;
var
  i: Integer;
begin
  for i := 1 to m do
    if e[i].selected and e[-i].selected then
      begin
        e[i].selected := False;
        e[-i].selected := False;
      end;
  end;
function Solve: Int64;
var
  u: Integer;
  i: Integer;
begin
  if not Dijkstra then Exit(-1);
  AdjustPotentials;
  d1 := price[n] - price[1];
  if not Dijkstra then Exit(-1);
  AdjustPotentials;
  d2 := price[n] - price[1];
  Result := d1 + d2;
  RemoveEdges;
  np := 0;
  u := 1;
  repeat
    Inc(np);
    p[np] := u;
    i := head[u];
    while not e[i].selected do i := link[i];
    u := e[i].y;
    e[i].selected := False;
  until u = n;
  nq := 0;
  u := 1;
  repeat
    Inc(nq);
    q[nq] := u;

```

```

    i := head[u];
    while not e[i].selected do i := link[i];
    u := e[i].y;
    e[i].selected := False;
    until u = n;
end;
procedure PrintResult;
var
    f: TextFile;
    k: LongInt;
begin
    AssignFile(f, OutputFile);
    Rewrite(f);
    try
        WriteLn(f, Res);
        if Res <> -1 then
            begin
                for k := 1 to np do Write(f, p[k], ' ');
                Write(f, n);
                for k := nq downto 1 do Write(f, ' ', q[k]);
            end;
    finally
        CloseFile(f);
    end;
end;
begin
    Enter;
    BuildIncidentLists;
    Init;
    res := Solve;
    PrintResult;
end.

```

7.48. Xây dựng thuật toán và cài đặt chương trình

Áp dụng thuật toán tìm luồng cực đại và lát cắt hẹp nhất.

Số nhân viên phải huy động là giá trị luồng cực đại, trong đó các cạnh nội giữa hai phía của lát cắt là những tuyến đường cần bố trí lực lượng tuần tra.

Chương trình

```
{$MODE OBJFPC}
program MaximumFlow;
const
  InputFile = 'DEA.INP';
  OutputFile = 'DEA.OUT';
  maxN = Round(1E3);
  maxM = Round(1E4);
  maxC = Round(1E9);
type
  TEdge = record
    x, y: Integer;
    c, f: Integer;
  end;
  TQueue = record
    items: array[1..maxN] of Integer;
    front, rear: Integer;
  end;
var
  e: array[-maxM..maxM] of TEdge;
  link: array[-maxM..maxM] of Integer;
  head: array[1..maxN] of Integer;
  trace: array[1..maxN] of Integer;
  n, m: Integer;
  FlowValue: Int64;
  Queue: TQueue;
procedure Enter;
var
  i: Integer;
  u, v, capacity: Integer;
  f: TextFile;
begin
  AssignFile(f, InputFile);
  Reset(f);
  try
    ReadLn(f, n, m);
    FillChar(head[1], n * SizeOf(head[1]), 0);
    for i := 1 to m do
```

```

begin
  ReadLn(f, u, v, capacity);
  with e[i] do
    begin
      x := u;
      y := v;
      c := capacity;
      link[i] := head[u];
      head[u] := i;
    end;
  with e[-i] do
    begin
      x := v;
      y := u;
      c := capacity;
      link[-i] := head[v];
      head[v] := -i;
    end;
  end;
finally
  CloseFile(f);
end;
procedure InitZeroFlow;
var
  i: Integer;
begin
  for i := -m to m do e[i].f := 0;
  FlowValue := 0;
end;
function FindPath: Boolean;
var
  u, v: Integer;
  i: Integer;
begin
  FillChar(trace[1], n * SizeOf(trace[1]), 0);
  trace[1] := 1;
  with Queue do
    begin

```

```

items[1] := 1;
front := 1;
rear := 1;
repeat
  u := items[front];
  Inc(front);
  i := head[u];
  while i <> 0 do
    begin
      v := e[i].y;
      if (trace[v] = 0) and (e[i].f < e[i].c) then
        begin
          trace[v] := i;
          if v = n then Exit(True);
          Inc(rear);
          items[rear] := v;
        end;
      i := link[i];
    end;
  until front > rear;
  Result := False;
end;
end;
procedure AugmentFlow;
var
  Delta: Integer;
  v, i: Integer;
begin
  v := n;
  Delta := maxC;
  repeat
    i := trace[v];
    if e[i].c - e[i].f < Delta then
      Delta := e[i].c - e[i].f;
    v := e[i].x;
  until v = 1;
  v := n;
  repeat
    i := trace[v];

```

```

    Inc(e[i].f, Delta);
    Dec(e[-i].f, Delta);
    v := e[i].x;
  until v = 1;
  FlowValue := FlowValue + Delta;
end;
procedure PrintResult;
var
  i: Integer;
  fo: TextFile;
begin
  AssignFile(fo, OutputFile);
  Rewrite(fo);
  try
    WriteLn(fo, FlowValue);
    for i := 1 to m do
      with e[i] do
        if (trace[x] = 0) xor (trace[y] = 0) then
          Write(fo, i, ' ');
  finally
    CloseFile(fo);
  end;
end;
begin
  Enter;
  InitZeroFlow;
  while FindPath do
    AugmentFlow;
  PrintResult;
end.

```

7.49. Xây dựng thuật toán và cài đặt chương trình

Đây là bài toán người đưa thư Trung Hoa trong trường hợp đồ thị có hướng (Directed Chinese Postman Problem).

Gọi X là tập đỉnh “thừa”: Những đỉnh có số cung đi vào nhiều hơn số cung đi ra, mức thừa của một đỉnh v là $\deg^-(v) - \deg^+(v)$.

Gọi Y là tập đỉnh “thiếu”: Những đỉnh có số cung đi ra nhiều hơn số cung đi vào, mức thiếu của một đỉnh v là $\deg^+(v) - \deg^-(v)$.

Tìm một bộ đường đi từ X sang Y thỏa mãn:

- Mỗi đường đi xuất phát từ một đỉnh thừa và kết thúc tại một đỉnh thiếu;
- Mỗi đỉnh thừa có số đường đi xuất phát từ nó đúng bằng mức thừa;
- Mỗi đỉnh thiếu có số đường đi kết thúc tại nó đúng bằng mức thiếu;
- Tổng độ dài các đường đi là ngắn nhất có thể.

Để tìm bộ đường đi này có thể sử dụng thuật toán ghép cặp có trọng số hoặc luồng chi phí cực tiểu.

Chương trình

```
{$MODE OBJFPC}
program DirectedChinesePostman;
const
  InputFile = 'DCPP.INP';
  OutputFile = 'DCPP.OUT';
  maxN = Round(1E3);
  maxM = Round(1E4);
  maxW = Round(1E6);
  maxD = maxN * maxW;
type
  TEdge = record
    u, v: Integer;
    c, f, w: Integer;
    link: Integer;
  end;
  THeap = record
    items, pos: array[1..maxN] of Integer;
    nItems: Integer;
  end;
var
  fi, fo: TextFile;
  n, m: Integer;
  e: array[-maxM..maxM] of TEdge;
  head: array[1..maxN] of Integer;
  p: array[1..maxN] of Integer;
  d: array[1..maxN] of Integer;
  excess: array[1..maxN] of Integer;
```

```

trace: array[1..maxN] of Integer;
z: array[1..maxN] of Integer;
nz: Integer;
heap: THeap;
res: Int64;
procedure Enter;
var
  i: Integer;
begin
  ReadLn(fi, n, m);
  for i := 1 to m do
    ReadLn(fi, e[i].u, e[i].v, e[i].w);
  for i := 1 to m do
    begin
      e[-i].u := e[i].v;
      e[-i].v := e[i].u;
      e[i].c := m;
      e[-i].c := 0;
      e[i].f := 0;
      e[-i].f := 0;
      e[-i].w := -e[i].w;
    end;
  FillDWord(head[1], n, 0);
  for i := -m to m do
    if i <> 0 then
      with e[i] do
        begin
          link := head[u];
          head[u] := i;
        end;
    end;
  function cf(const e: TEdge): Integer; inline;
begin
  with e do Result := c - f;
end;
  function wp(const e: TEdge): Integer; inline;
begin
  with e do
    Result := w + p[u] - p[v];

```

```

end;
procedure Init;
var
  i: Integer;
begin
  FillDWord(excess[1], n, 0);
  for i := 1 to m do
    with e[i] do
      begin
        Dec(excess[u]);
        Inc(excess[v]);
      end;
  FillDWord(p[1], n, 0);
end;
procedure DijkstraInit(start: Integer);
begin
  FillDWord(d[1], n, maxD);
  d[start] := 0;
  with heap do
    begin
      FillDWord(pos[1], n, 0);
      nItems := 1;
      items[1] := start;
      pos[start] := 1;
    end;
end;
function Extract: Integer;
var
  r, c: Integer;
  temp: Integer;
begin
  with heap do
    begin
      Result := items[1];
      temp := items[nItems];
      Dec(nItems);
      r := 1;
      repeat
        c := r * 2;

```

```

    if (c < nItems) and
      (d[items[c]] > d[items[c + 1]])
    then Inc(c);
    if (c > nItems) or (d[items[c]] >= d[temp]) then
      Break;
    items[r] := items[c];
    pos[items[r]] := r;
    r := c;
    until False;
    items[r] := temp;
    pos[temp] := r;
  end;
end;
function Update(v, dv: Integer): Boolean;
var
  r, c: Integer;
begin
  Result := d[v] > dv;
  if not Result then Exit;
  d[v] := dv;
  with heap do
    begin
      c := pos[v];
      if c = 0 then
        begin
          Inc(nItems);
          c := nItems;
        end;
      repeat
        r := c div 2;
        if (r = 0) or (d[items[r]] <= dv) then Break;
        items[c] := items[r];
        pos[items[c]] := c;
        c := r;
      until False;
      items[c] := v;
      pos[v] := c;
    end;
  end;
end;

```

```

function Dijkstra(start: Integer): Integer;
var
    uPop: Integer;
    i: Integer;
begin
    nz := 0;
    DijkstraInit(start);
repeat
    uPop := Extract;
    if excess[uPop] < 0 then
        begin
            Result := uPop;
            Break;
        end;
    Inc(nz);
    z[nz] := uPop;
    i := head[uPop];
    while i <> 0 do
        with e[i] do
            begin
                if (cf(e[i]) > 0) and
                    Update(v, d[uPop] + wp(e[i]));
                then trace[v] := i;
                i := link;
            end;
    until False;
//Update price function
for i := 1 to nz do
    Inc(p[z[i]], d[z[i]] - d[Result]);
end;
procedure AugmentFlow(s, t: Integer);
var
    i, v, Delta: Integer;
begin
    if excess[s] < -excess[t] then Delta := excess[s]
    else Delta := -excess[t];
    v := t;
    repeat
        i := trace[v];

```

```

Inc(e[i].f, Delta);
Dec(e[-i].f, Delta);
v := e[i].u;
until v = s;
Dec(excess[s], Delta);
Inc(excess[t], Delta);
end;
procedure SuccessiveShortestPaths;
var
  s, t, i; Integer;
begin
  for s := 1 to n do
    while excess[s] > 0 do
      begin
        t := Dijkstra(s);
        AugmentFlow(s, t);
      end;
  res := 0;
  for i := 1 to m do res := res + e[i].w;
  for i := -m to m do
    if (i <> 0) and (e[i].f > 0) then
      res := res + Int64(e[i].f) * e[i].w;
end;
begin
  AssignFile(fi, InputFile);
  Reset(fi);
  AssignFile(fo, OutputFile);
  Rewrite(fo);
  try
    Enter;
    Init;
    SuccessiveShortestPaths;
    Write(fo, res);
  finally
    CloseFile(fi);
    CloseFile(fo);
  end;
end.

```

MỤC LỤC

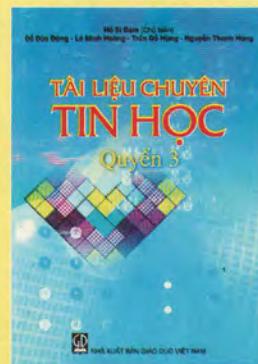
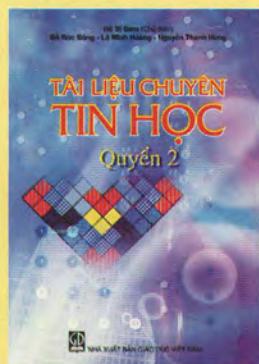
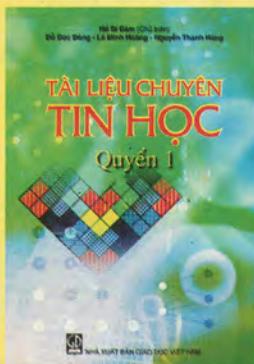
Lời nói đầu	3
BÀI TẬP	
Chuyên đề 6. KIẾU DỮ LIỆU TRÙU TƯỢNG VÀ CẤU TRÚC DỮ LIỆU	
Bài tập chuyên đề 6	5
Bài tập bổ sung	14
Chuyên đề 7. ĐỒ THỊ	
Bài tập chuyên đề 7	23
Bài tập bổ sung	37
HƯỚNG DẪN – GIẢI BÀI TẬP	
Chuyên đề 6.	44
Chuyên đề 7.	116



VƯƠNG MIỆN KIM CƯƠNG
CHẤT LƯỢNG QUỐC TẾ

GIỚI THIỆU BỘ SÁCH

TÀI LIỆU CHUYÊN TIN HỌC DÀNH CHO HỌC SINH PHỔ THÔNG



Bạn đọc có thể mua sách tại các Công ty Sách - Thiết bị trường học ở các địa phương hoặc các cửa hàng sách của Nhà xuất bản Giáo dục Việt Nam :

- Tại TP. Hà Nội : 45 Phố Vọng ; 187, 187C Giảng Võ ; 232 Tây Sơn ; 25 Hàn Thuyên;
51 Lò Ðúc ; 45 Hàng Chuối ; Ngõ 385 Hoàng Quốc Việt;
17 T2 - 17T3 Trung Hòa - Nhân Chính ; Tòa nhà HESCO Văn Quán - Hà Đông
- Tại TP. Đà Nẵng : 78 Pasteur ; 145 Lê Lợi ; 223 Lê Đình Lý.
- Tại TP. Hồ Chí Minh : 261C Lê Quang Định, Quận Bình Thạnh ; 231 Nguyễn Văn Cừ, Quận 5 ;
23 Đinh Tiên Hoàng, Phường Đa Kao, Quận 1, TP. Hồ Chí Minh
- Tại TP. Cần Thơ : 162D Đường 3 tháng 2, Phường Xuân Khánh, Quận Ninh Kiều.
- Tại Website bán hàng trực tuyến : www.sach24.vn

Website : www.nxbgd.vn

ISBN 978-604-0-08216-9

9 786040 082169



Giá : 31.000 đ