3

Brute Force và
Tìm kiếm đầy đủ

Khoa học khác xa với vũ lực như thanh kiếm này khỏi xà beng.

—Edward Lytton (1803–1873), Leila, Quyển II, Chương I

Làm tốt một việc thường rất lãng phí thời gian.

—Robert Byrne, một người chơi bida và bida bậc thầy, đồng thời là một nhà văn

Người giới thiệu khung và phương pháp phân tích thuật toán trong chương trước, chúng ta đã sẵn sàng bắt tay vào thảo luận về các chiến lược thiết kế thuật toán. Mỗi trong số tám chương tiếp theo được dành cho một chiến lược thiết kế cụ thể. Chủ đề của chương này là bạo lực và trường hợp đặc biệt quan trọng của nó, tìm kiếm toàn diện. Tính vũ phu có thể được mô tả như sau:

> Brute force là một cách tiếp cận đơn giản để giải quyết một vấn đề, thường trực tiếp dựa trên tuyên bố vấn đề và định nghĩa của các khái niệm liên quan.

"Lực lượng" được ngụ ý trong định nghĩa của chiến lược là lực lượng của một máy tính chứ không phải trí tuệ của một người. "Cứ làm đi!" sẽ là một cách khác để mô tả quy định của cách tiếp cận vũ phu. Và thông thường, chiến lược bạo lực thực sự là chiến lược dễ áp dụng nhất.

Ví dụ, hãy xem xét bài toán lũy thừa: tính một cho một số khác a và một số nguyên không âm n. Mặc dù vấn đề này có vẻ tầm thường, nhưng nó cung cấp một phương tiện hữu ích để minh họa một số chiến lược thiết kế thuật toán, bao gồm cả lực lượng vũ phu. (Cũng lưu ý rằng tính toán một modm cho một số số nguyên lớn là một thành phần chính của thuật toán mã hóa hàng đầu.) Theo định nghĩa của lũy thừa,

$$a^n = \underbrace{a \cdot \ldots \cdot a}_{n \text{ lần}}.$$

Điều này gợi ý rằng chỉ cần tính toán một bằng cách nhân 1 với một lần.

Chúng tôi đã gặp ít nhất hai thuật toán brute-force trong cuốn sách: thuật toán kiểm tra số nguyên liên tiếp để tính toán gcd (m, n) trong Phần 1.1 và thuật toán dựa trên định nghĩa cho phép nhân ma trận trong Phần 2.3. Nhiều các ví dụ khác đư ợc đư a ra sau trong chư ơng này. (Bạn có thể xác định một vài thuật toán không bạn đã biết là dựa trên cách tiếp cận vũ phu?)

Mặc dù hiếm khi là nguồn cung cấp các thuật toán thông minh hoặc hiệu quả, như ng không nên bỏ qua brute-force ap proach như một chiến lư ợc thiết kế thuật toán quan trọng. Đầu tiên, không giống như một số chiến lư ợc khác, vũ phu có thể áp dụng cho một loạt các vấn đề. Trên thực tế, nó dư ờng như là cách tiếp cận chung duy nhất mà nó khó hơn để chỉ ra những vấn đề mà nó không thể giải quyết. Thứ hai, đối với một số vấn đề quan trọng — ví dụ: sắp xếp, tìm kiếm, phép nhân ma trận, so khớp chuỗi— phư ơng pháp tiếp cận brute-force mang lại các thuật toán hợp lý của ít nhất một số giá trị thực tế mà không có giới hạn về kích thư ớc phiên bản. Thứ ba, chi phí thiết kế thuật toán hiệu quả hơn có thể không chính đáng nếu chỉ cần giải quyết một số trư ờng hợp lỗi có thể xảy ra và thuật toán brute-force có thể giải quyết các trư ờng hợp đó bằng tốc độ chấp nhận đư ợc. Thứ tư , ngay cả khi nói chung là quá kém hiệu quả, thì một thuật toán brute-force vẫn có thể hữu ích để giải quyết các trư ờng hợp kích thư ớc nhỏ của một vấn đề. Cuối cùng, một thuật toán brute-force có thể phục vụ một quan trọng về lý thuyết hoặc giáo dục như một thư ớc đo để đánh giá các lựa chọn thay thế hiệu quả hơn để giải quyết một vấn đề.

3.1 Sắp xếp lựa chọn và Sắp xếp bong bóng

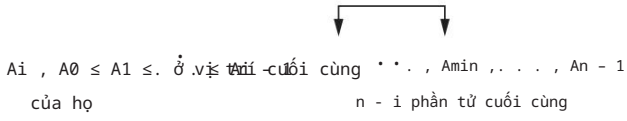
Trong phần này, chúng tôi xem xét việc áp dụng cách tiếp cận bạo lực đối với vấn đề sắp xếp: đư a ra một danh sách gồm n mục có thể sắp xếp thứ tự (ví dụ: số, ký tự từ một số bảng chữ cái, chuỗi ký tự), sắp xếp lại chúng theo thứ tự không giảm. Như chúng tôi đã đề cập trong Phần 1.3, hàng chục thuật toán đã đư ợc phát triển cho giải quyết vấn đề rất quan trọng này. Bạn có thể đã học một vài trong số chúng trong quá khứ. Nếu có, hãy cố gắng quên chúng đi và nhìn nhận vấn đề làm mới.

Bây giờ, sau khi tâm trí của bạn không còn gánh nặng về kiến thức trư ớc đây về phân loại nhịp điệu thuật số, hãy tự hỏi bản thân một câu hỏi: “Đâu sẽ là phư ơng pháp đơn giản nhất để giải quyết vấn đề sắp xếp? " Những ngư ời hợp lý có thể không đồng ý về câu trả lời cho câu hỏi này. Hai thuật toán đư ợc thảo luận ở đây — sắp xếp lựa chọn và bong bóng sắp xếp — dư ờng như là hai ứng cử viên chính.

Sắp xếp lựa chọn

Chúng tôi bắt đầu sắp xếp lựa chọn bằng cách quét toàn bộ danh sách đã cho để tìm phần tử nhỏ nhất của nó và trao đổi nó với phần tử đầu tiên, đặt phần tử nhỏ nhất vào phần tử cuối cùng của nó vị trí trong danh sách đã sắp xếp. Sau đó, chúng tôi quét danh sách, bắt đầu với phần tử thứ hai, để tìm phần tử nhỏ nhất trong số n - 1 phần tử cuối cùng và đổi nó với phần tử thứ hai , đặt phần tử nhỏ thứ hai vào vị trí cuối cùng của nó. Nói chung, trên

thứ i chuyển qua danh sách mà chúng tôi đánh số từ 0 đến n - 2, thuật toán tìm kiếm mục nhỏ nhất trong số n - i phần tử cuối cùng và hoán đổi nó với Ai :



Sau n - 1, danh sách được sắp xếp.

Đây là mã giả của thuật toán này, để đơn giản, giả sử rằng danh sách được triển khai dưới dạng một mảng:

THUẬT TOÁN Lựa chọn Sắp xếp (A [0..n - 1])

// Sắp xếp một mảng đã cho theo cách sắp xếp

lựa chọn // Đầu vào: Một mảng A [0..n - 1] gồm các phần tử

có thể sắp xếp được // Đầu ra: Mảng A [0..n - 1] được sắp xếp theo

thứ tự không giảm cho i 0 to n - 2 do min i for j i + 1 to n

- 1 do if A [j] <A [min] min j swap A [i] và A [min]

Ví dụ: hoạt động của thuật toán trong danh sách 89, 45, 68, 90, 29, 34, 17 được minh họa trong Hình 3.1.

Việc phân tích loại lựa chọn rất đơn giản. Kích thước đầu vào được cho bởi số phần tử n; phép toán cơ bản là so sánh khóa A [j] <A [min].

Số lần nó được thực thi chỉ phụ thuộc vào kích thước mảng và được cho bởi tổng sau:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i).$$

89 |	45	68	90	29	34	17
17 |	45	68	90	29	34	89
17	29 |	68	90	45	34	89
17	29	34 |	90	45	68	89
17	29	34	45 |	90	68	89
17	29	34	45	68 |	90	89
17	29	34	45	68	89 |	90

HÌNH 3.1 Ví dụ về sắp xếp với sắp xếp lựa chọn. Mỗi dòng tương ứng với một lần lặp lại của thuật toán, tức là, đi qua phần đuôi của danh sách ở bên phải của thanh dọc; một phần tử in đậm cho biết phần tử nhỏ nhất được tìm thấy. Các phần tử ở bên trái của thanh dọc đang ở vị trí cuối cùng của chúng và không được xem xét trong lần lặp này và các lần lặp tiếp theo.

Vì chúng ta đã gặp tổng cuối cùng trong việc phân tích thuật toán của Ví dụ 2 trong Phần 2.3, nên bây giờ bạn có thể tự tính toán nó.
Cho dù bạn tính tổng này bằng cách phân phối ký hiệu tổng hay bằng cách lấy ngay tổng của các số nguyên giảm dần, tất nhiên, câu trả lời phải giống nhau:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n - 1 - i) = \frac{(n-1)}{2}.$$

Do đó, sắp xếp lựa chọn là một thuật toán (n2) trên tất cả các đầu vào. Tuy nhiên, lưu ý rằng số lần hoán đổi khóa chỉ là (n), hay chính xác hơn là n - 1 (một lần cho mỗi lần lặp lại của vòng lặp thứ i). Thuộc tính này phân biệt lựa chọn sắp xếp tích cực với nhiều thuật toán sắp xếp khác.

Sắp xếp bong bóng

Một ứng dụng brute-force khác cho vấn đề sắp xếp là so sánh các phần tử liền kề của danh sách và trao đổi chúng nếu chúng không theo thứ tự. Bằng cách thực hiện lặp đi lặp lại, cuối cùng chúng tôi sẽ "đẩy" phần tử lớn nhất lên vị trí cuối cùng trong danh sách. Lần vượt qua tiếp theo làm bong bóng phần tử lớn thứ hai, và cứ tiếp tục như vậy, cho đến sau lần thứ n - 1, danh sách được sắp xếp. Vượt qua i (0 ≤ i ≤ n - 2) của sắp xếp bong bóng có thể được biểu diễn bằng sơ đồ sau:

$$A_0, \ldots, A_j \overset{?}{\gtrless} A_{j+1}, \ldots, A_{n-i-1} \mid A_{n-i} \leq \ldots \leq A_{n-1}$$

ở các vị trí cuối cùng của chúng

Đây là mã giả của thuật toán này.

THUẬT TOÁN BubbleSort (A [0..n - 1])
// Sắp xếp một mảng đã cho theo sắp xếp bong
bóng // Đầu vào: Một mảng A [0..n - 1] gồm các phần tử có thể
sắp xếp được // Đầu ra: Mảng A [0..n - 1] được sắp xếp theo thứ tự
không giảm cho i 0 to n - 2 làm cho j 0 thành n - 2 - tôi làm nếu
A [j + 1] <A [j] hoán đổi A [j] và A [j + 1]

Hoạt động của thuật toán trong danh sách 89, 45, 68, 90, 29, 34, 17 được minh họa như một ví dụ trong Hình 3.2.

Số lượng so sánh chính cho phiên bản sắp xếp bong bóng được đưa ra ở trên là giống nhau đối với tất cả các mảng có kích thước n; nó nhận được bằng một tổng gần giống với tổng cho sắp xếp lựa chọn:

HÌNH 3.2 Hai lần chuyển đầu tiên của sắp xếp bong bóng trong danh sách 89, 45, 68, 90, 29, 34, 17. Một dòng mới được hiển thị sau khi hoán đổi hai phần tử được thực hiện. Các phần tử ở bên phải của thanh dọc nằm ở vị trí cuối cùng của chúng và không được xem xét trong các lần lặp tiếp theo của thuật toán.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = 2\frac{(n-1)n}{} \quad (n^2).$$

Tuy nhiên, số lần hoán đổi phím phụ thuộc vào đầu vào. Trong trường hợp xấu nhất của việc giảm mảng, nó cũng giống như số lượng so sánh chính:

$$S_{worst}(n) = C(n) = \frac{(n-1)n}{2} \quad (n^2).$$

Như thường xảy ra với việc áp dụng chiến lược brute-force, phiên bản đầu tiên của thuật toán thu được thường có thể được cải thiện với một lượng nỗ lực khiêm tốn. Cụ thể, chúng ta có thể cải thiện phiên bản thô của sắp xếp bong bóng được đưa ra ở trên bằng cách khai thác quan sát sau: nếu một lần chuyển qua danh sách không tạo ra trao đổi, danh sách đã được sắp xếp và chúng ta có thể dừng thuật toán (Bài toán 12a trong bài tập của phần này). Mặc dù phiên bản mới chạy nhanh hơn trên một số đầu vào, nhưng nó vẫn ở (n^2) trong trường hợp trung bình và tồi tệ nhất. Trên thực tế, ngay cả trong số các phương pháp sắp xếp cơ bản, sắp xếp bong bóng là một lựa chọn kém hơn, và nếu không phải vì cái tên hấp dẫn của nó, có lẽ bạn sẽ chưa bao giờ nghe nói về nó. Tuy nhiên, bài học chung mà bạn vừa học được rất quan trọng và đáng được nhắc lại:

> Ứng dụng đầu tiên của phương pháp brute-force thường dẫn đến một thuật toán có thể được cải thiện với một lượng nỗ lực khiêm tốn.

Bài tập 3.1

1. a. Đưa ra một ví dụ về một thuật toán không nên được coi là một ứng dụng của phương pháp brute-force. b. Đưa ra một ví dụ về một vấn đề không thể giải quyết bằng vũ lực thuật toán.

2. a. Hiệu suất thời gian của thuật toán brute-force để tính toán một hàm của n là bao nhiêu? Là một hàm của số bit trong biểu diễn nhị phân của n? b. Nếu bạn tính toán một mod trong đó a> 1 và n là một số nguyên dương lớn, bạn sẽ giải quyết vấn đề về độ lớn rất lớn của an như thế nào?

3. Đối với mỗi thuật toán trong các Bài toán 4, 5 và 6 của Bài tập 2.3, hãy cho biết thuật toán có dựa trên cách tiếp cận brute-force hay không.

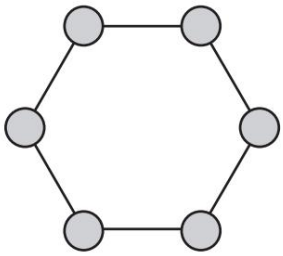
4. a. Thiết kế một thuật toán brute-force để tính toán giá trị của một đa thức

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

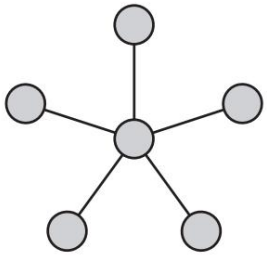
tại một điểm x0 cho trước và xác định lớp hiệu quả trong trường hợp xấu nhất của nó. b. Nếu thuật toán bạn thiết kế nằm trong (n2), hãy thiết kế một thuật toán tuyến tính cho điều này vấn đề.

C. Có thể thiết kế một thuật toán với hiệu quả tốt hơn tuyến tính cho vấn đề này?

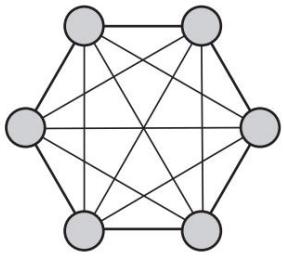
5. Cấu trúc liên kết mạng chỉ định cách máy tính, máy in và các thiết bị khác được kết nối qua mạng. Hình dưới đây minh họa ba cấu trúc liên kết phổ biến của mạng: vòng, hình sao và lưới được kết nối đầy đủ.



nhẫn



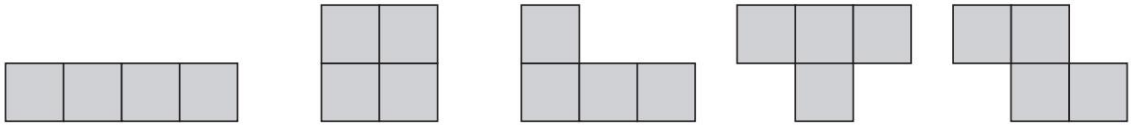
ngôi sao



lưới kết nối đầy đủ

Bạn được cung cấp ma trận boolean A [0..n - 1, 0..n - 1], trong đó n> 3, được cho là ma trận kề của biểu đồ mô hình hóa mạng có một trong các cấu trúc liên kết này. Nhiệm vụ của bạn là xác định cấu trúc liên kết nào trong số ba cấu trúc liên kết này, nếu có, đại diện cho ma trận. Thiết kế một thuật toán brute-force cho nhiệm vụ này và chỉ ra lớp hiệu quả thời gian của nó.

6. Ốp lát Tetromino Tetromino là những viên gạch lát được tạo thành từ bốn hình vuông 1 × 1. Ở đó là năm loại tetrominoes được hiển thị bên dưới:



tetromino thẳng tetromino vuông L-tetromino T-tetromino Z-tetromino

Có thể xếp - tức là che chính xác mà không có chồng chéo - một bàn cờ 8 × 8 với a. các tetromino thẳng? b. tứ diện vuông?

C. L-tetrominoes? d. T-tetrominoes?

e. Z-tetrominoes?

7. Một xấp đồng xu giả Có n xấp đồng xu có hình dạng giống hệt nhau. Tất cả các đồng xu trong một trong các ngăn xếp này là hàng giả, trong khi tất cả các đồng xu trong các ngăn xếp khác là hàng thật. Mỗi đồng tiền chính hãng nặng 10 gam; mỗi giả nặng 11 gam. Bạn có một cân phân tích có thể xác định trọng lư ợng chính xác của bất kỳ số lư ợng tiền xu nào. Một. Phát triển một thuật toán brute-force để xác định ngăn xếp với các đồng tiền giả và

xác định lớp hiệu quả trong trư ờng hợp xấu nhất của nó.

b. Số lần cân tối thiểu cần thiết để xác định đồng tiền giả là bao nhiêu?

8. Sắp xếp danh sách E, X, A, M, P, L, E theo thứ tự bảng chữ cái bằng cách sắp xếp lựa chọn.

9. Sắp xếp lựa chọn có ổn định không? (Định nghĩa của một thuật toán sắp xếp ổn định đã đư ợc đư a ra trong Phần 1.3.)

10. Có thể thực hiện sắp xếp lựa chọn cho các danh sách đư ợc liên kết với cùng hiệu suất (n2) như phiên bản mảng không?

11. Sắp xếp danh sách E, X, A, M, P, L, E theo thứ tự bảng chữ cái bằng cách sắp xếp bong bóng.

12. a. Chứng minh rằng nếu sắp xếp bong bóng không thực hiện trao đổi nào khi nó chuyển qua một danh sách, danh sách đư ợc sắp xếp và thuật toán có thể dừng lại.

b. Viết mã giả của phư ơng pháp kết hợp cải tiến này. C. Chứng minh rằng hiệu suất trong trư ờng hợp xấu nhất của phiên bản cải tiến là bậc hai.

13. Sắp xếp bong bóng có ổn định không?

14. Đĩa xen kẽ Bạn có một dãy 2n đĩa có hai màu, n đậm và n sáng.

Chúng xen kẽ: tối, sáng, tối, sáng, v.v. Bạn muốn chuyển tất cả các đĩa tối ở đầu bên phải và tất cả các đĩa sáng ở đầu bên trái. Các động thái duy nhất bạn đư ợc phép thực hiện là chuyển vị trí của hai đĩa lân cận.

Thiết kế một thuật toán để giải câu đố này và xác định số lần di chuyển mà nó cần. [Gar99]

3.2 Tìm kiếm tuần tự và đối sánh chuỗi Brute-Force

Chúng ta đã thấy trong phần trước hai ứng dụng của phương pháp bạo lực đối với vấn đề phân loại. Ở đây chúng ta thảo luận về hai ứng dụng của chiến lược này đối với vấn đề tìm kiếm. Phần đầu tiên giải quyết vấn đề chính tắc khi tìm kiếm một mục có giá trị nhất định trong một danh sách nhất định. Thứ hai khác ở chỗ nó giải quyết vấn đề so khớp chuỗi.

Tìm kiếm tuần tự

Chúng ta đã gặp một thuật toán brute-force cho vấn đề tìm kiếm chung: nó được gọi là tìm kiếm tuần tự (xem Phần 2.1). Để lặp lại, thuật toán chỉ cần so sánh các phần tử liên tiếp của một danh sách nhất định với một khóa tìm kiếm nhất định cho đến khi gặp một kết quả phù hợp (tìm kiếm thành công) hoặc danh sách hết mà không tìm thấy kết quả phù hợp (tìm kiếm không thành công). Một thủ thuật bổ sung đơn giản thường được sử dụng trong việc triển khai tìm kiếm tuần tự: nếu chúng ta nối khóa tìm kiếm vào cuối danh sách, thì việc tìm kiếm khóa sẽ phải thành công và do đó chúng ta có thể loại bỏ hoàn toàn việc kiểm tra cuối danh sách. Đây là mã giả của phiên bản nâng cao này.

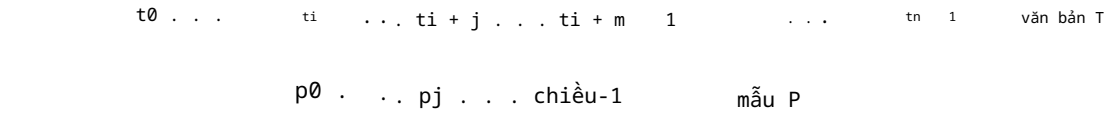
```
THUẬT TOÁN Tuần tự Tìm kiếm2 (A [0..n], K)
    // Thực hiện tìm kiếm tuần tự với khóa tìm kiếm là trạm gác // Đầu
    vào: Một mảng A gồm n phần tử và khóa tìm kiếm K // Đầu ra: Chỉ số
    của phần tử đầu tiên trong A [0..n - 1] có giá trị là // bằng K hoặc   1
    nếu không tìm thấy phần tử như vậy A [n] ← K i ← 0 trong khi A [i] = K
    do i ← i + 1 nếu tôi < n return i else return −1
```

Một cải tiến đơn giản khác có thể được kết hợp trong tìm kiếm tuần tự nếu một danh sách nhất định được biết là đã được sắp xếp: tìm kiếm trong danh sách như vậy có thể dừng lại ngay khi gặp phải phần tử lớn hơn hoặc bằng khóa tìm kiếm.

Tìm kiếm tuần tự cung cấp một minh họa tuyệt vời về phương pháp brute-force ap proach, với sức mạnh đặc trưng của nó (tính đơn giản) và điểm yếu (hiệu quả kém hơn). Kết quả hiệu quả thu được trong Phần 2.1 đối với phiên bản tiêu chuẩn của tìm kiếm tuần tự chỉ thay đổi rất ít đối với phiên bản nâng cao, do đó thuật toán vẫn tuyến tính trong cả trường hợp xấu nhất và trung bình. Chúng ta sẽ thảo luận ở phần sau của cuốn sách về một số thuật toán tìm kiếm với hiệu quả thời gian tốt hơn.

Khớp chuỗi Brute-Force

Nhắc lại bài toán so khớp chuỗi đã giới thiệu ở mục 1.3: cho một chuỗi n ký tự đư ợc gọi là văn bản và một chuỗi gồm m ký tự (m ≤ n) đư ợc gọi là mẫu, hãy tìm một chuỗi con của văn bản phù hợp với mẫu. Nói chính xác hơn, chúng tôi muốn tìm i — chỉ số của ký tự ngoài cùng bên trái của chuỗi con phù hợp đầu tiên trong văn bản — sao cho ti = p0 ,, ti + m 1 = pm 1 : · · · , ti + j = pj , · · ·



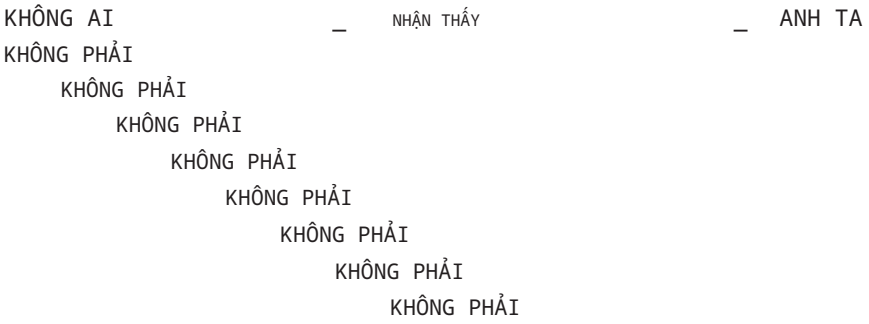
Nếu các kết quả phù hợp khác với kết quả đầu tiên cần đư ợc tìm thấy, thuật toán so khớp chuỗi có thể tiếp tục hoạt động cho đến khi hết toàn bộ văn bản.

Một thuật toán brute-force cho vấn đề so khớp chuỗi khá rõ ràng: căn chỉnh mẫu so với m ký tự đầu tiên của văn bản và bắt đầu so khớp các cặp ký tự tư ơng ứng từ trái sang phải cho đến khi tất cả m cặp ký tự khớp nhau (sau đó thuật toán có thể dừng) hoặc gặp phải một cặp không khớp. Trong trư ờng hợp thứ hai, hãy chuyển vị trí của mẫu sang phải và tiếp tục so sánh ký tự, bắt đầu lại với ký tự đầu tiên của mẫu và ký tự đối của nó trong văn bản. Lư u ý rằng vị trí cuối cùng trong văn bản vẫn có thể là phần đầu của chuỗi con phù hợp n - m (với điều kiện các vị trí văn bản đư ợc lập chỉ mục từ 0 đến n - 1). Ngoài vị trí đó, không có đủ ký tự để khớp với toàn bộ mẫu; do đó, thuật toán không cần thực hiện bất kỳ so sánh nào ở đó.

THUẬT TOÁN BruteForceStringMatch (T [0..n - 1], P [0..m - 1])

// Thực hiện so khớp chuỗi brute-force //
Đầu vào: Một mảng T [0..n - 1] gồm n ký tự đại diện cho một văn bản và
một mảng P [0..m - 1] gồm m ký tự đại diện cho một mẫu //
// Đầu ra: Chỉ số của ký tự đầu tiên trong văn bản bắt đầu // khớp
với chuỗi con hoặc -1 nếu tìm kiếm không thành công đối với i 0
đến n - m do j 0 trong khi j < m và P [j] = T [i + j] do
j j + 1 nếu j = m trả về tôi trả về 1

Hoạt động của thuật toán đư ợc minh họa trong Hình 3.3. Lư u ý rằng đối với ví dụ này, thuật toán thay đổi mẫu hầu như luôn luôn sau khi so sánh một ký tự. Trư ờng hợp xấu nhất còn tệ hơn nhiều: thuật toán có thể phải thực hiện tất cả m so sánh trư ớc khi chuyển mẫu và điều này có thể xảy ra với mỗi lần thử n - m + 1. (Vấn đề 6 trong các bài tập của phần này yêu cầu bạn đư a ra một ví dụ cụ thể về một tình huống như vậy.) Vì vậy, trong trư ờng hợp xấu nhất, thuật toán làm cho



HÌNH 3.3 Ví dụ về đối sánh chuỗi brute-force. Các ký tự của mẫu đó là so với các đối tác văn bản của chúng được in đậm.

m (n - m + 1) so sánh ký tự, đặt nó vào lớp O (nm). Tuy nhiên, đối với một tìm kiếm từ điển hình trong văn bản ngôn ngữ tự nhiên, chúng ta nên mong đợi rằng hầu hết các thay đổi sẽ xảy ra sau rất ít so sánh (hãy kiểm tra lại ví dụ). Do đó, hiệu quả trường hợp trung bình phải tốt hơn đáng kể so với hiệu quả trường hợp xấu nhất. Thật vậy, nó là: để tìm kiếm trong các văn bản ngẫu nhiên, nó đã được chứng minh là tuyến tính, tức là, (n). Có một số thuật toán phức tạp hơn và hiệu quả hơn để tìm kiếm chuỗi. Cách được biết đến rộng rãi nhất trong số đó - bởi R. Boyer và J. Moore - được nêu trong Phần 7.2 cùng với sự đơn giản hóa nó do R. Horspool đề xuất.

Bài tập 3.2

1. Tìm số lần so sánh được thực hiện bởi phiên bản tuần tự của phiên bản sentinel Tìm kiếm

   Một. trong trường hợp xấu nhất.

   b. trong trường hợp trung bình nếu xác suất tìm kiếm thành công là p (0 ≤ p ≤ 1).

2. Như được trình bày trong Phần 2.1, số lượng trung bình của các phép so sánh chính được thực hiện bằng tìm kiếm tuần tự (không có trạm gác, theo các giả định tiêu chuẩn về đầu vào của nó) được đưa ra bởi công thức

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p),$$

   trong đó p là xác suất tìm kiếm thành công. Xác định, với một n cố định, các giá trị của p (0 ≤ p ≤ 1) mà công thức này mang lại giá trị lớn nhất của Cavg (n) và giá trị nhỏ nhất của Cavg (n).

3. Kiểm tra đồ dùng Một công ty muốn xác định tầng cao nhất của trụ sở chính tầng n của mình mà từ đó một thiết bị có thể rơi xuống mà không bị vỡ. Công ty có hai thiết bị giống hệt nhau để thử nghiệm. Nếu một trong số chúng bị hỏng, nó không thể được sửa chữa và thử nghiệm sẽ phải được hoàn thành với thiết bị còn lại. Thiết kế một thuật toán trong lớp hiệu quả tốt nhất mà bạn có thể để giải quyết vấn đề này.

4. Xác định số lượng các phép so sánh ký tự được thực hiện bởi thuật toán brute-force để tìm kiếm mẫu GANDHI trong văn bản

   THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

   Giả sử rằng độ dài của văn bản – dài 47 ký tự – được biết trước khi bắt đầu tìm kiếm.

5. Có bao nhiêu phép so sánh (cả thành công và không thành công) được thực hiện bởi thuật toán brute-force khi tìm kiếm mỗi mẫu sau đây trong văn bản nhị phân của một nghìn số không? Một. 00001 b. 10000 c. 01010

6. Cho một ví dụ về văn bản có độ dài n và một mẫu có độ dài m tạo thành đầu vào trong trường hợp xấu nhất cho thuật toán so khớp chuỗi brute-force. Chính xác có bao nhiêu phép so sánh ký tự sẽ được thực hiện cho đầu vào như vậy?

7. Khi giải bài toán so khớp chuỗi, sẽ có lợi thế nào khi so sánh các ký tự mẫu và văn bản từ phải sang trái thay vì từ trái sang phải?

8. Hãy xem xét vấn đề đếm, trong một văn bản nhất định, số chuỗi con bắt đầu bằng chữ A và kết thúc bằng chữ B. Ví dụ, có bốn chuỗi con như vậy trong CABAAXBYA. Một. Thiết kế một thuật toán brute-force cho vấn đề này và xác định hiệu quả của nó

   lớp ciency.

   b. Thiết kế một thuật toán hiệu quả hơn cho vấn đề này. [Gin04]

9. Viết chương trình trực quan hóa cho thuật toán so khớp chuỗi brute-force.

10. Tìm từ Một trò chơi phổ biến ở Hoa Kỳ, câu đố “tìm từ” (hoặc “tìm kiếm từ”) yêu cầu người chơi tìm từng từ trong một tập hợp từ nhất định trong một bảng vuông chứa đầy các chữ cái. Một từ có thể đọc theo chiều ngang (trái hoặc phải), theo chiều dọc (lên hoặc xuống), hoặc dọc theo đường chéo 45 độ (theo bất kỳ hướng nào trong bốn hướng) được tạo thành bởi các ô liền kề liên tiếp của bảng; nó có thể quấn quanh ranh giới của bảng, nhưng nó phải đọc theo cùng một hướng mà không ngoằn ngoèo. Cùng một ô của bảng có thể được sử dụng trong các từ khác nhau, nhưng trong một từ nhất định, cùng một ô có thể được sử dụng không quá một lần. Viết một chương trình máy tính để giải câu đố này.

11. Game Battleship Viết chương trình dựa trên phiên bản đối sánh mẫu brute-force để chơi game Battleship trên máy tính. Các quy tắc của trò chơi như sau. Có hai đối thủ trong trò chơi (trong trường hợp này là người chơi và máy tính). Trò chơi được chơi trên hai bảng giống hệt nhau (10 × 10 bảng ô vuông), trên đó mỗi đối thủ đặt các tàu của mình để đối phương không nhìn thấy. Mỗi người chơi có năm tàu, mỗi tàu chiếm một số ô vuông nhất định trên bàn cờ: tàu khu trục (hai ô vuông), tàu ngầm (ba ô vuông), tàu tuần dương (ba ô vuông), tàu chiến (bốn ô vuông) và máy bay tàu sân bay (năm ô vuông). Mỗi tàu được đặt theo chiều ngang hoặc chiều dọc, không để hai tàu chạm vào nhau. Trò chơi được chơi bằng cách các đối thủ lần lượt “bắn” vào tàu của nhau. Các

kết quả của mọi cú đánh được hiển thị dưới dạng một cú đánh hoặc một cú đánh trượt. Trong trường hợp bị bắn trúng, người chơi được quay lại và tiếp tục chơi cho đến khi mất tích. Mục đích là đánh chìm tất cả các tàu của đối thủ trước khi đối thủ thực hiện thành công trước. Để đánh chìm một con tàu, tất cả các ô vuông mà con tàu chiếm giữ phải bị đánh.

3.3 Các vấn đề về cặp gần nhất và lồi lõm bởi Brute Force

Trong phần này, chúng ta xem xét một cách tiếp cận đơn giản đối với hai lỗi xác suất nổi tiếng giải quyết một tập hợp hữu hạn các điểm trong mặt phẳng. Những vấn đề này, ngoài mối quan tâm lý thuyết của chúng, còn phát sinh trong hai lĩnh vực ứng dụng quan trọng: đo đạc tính toán địa lý và nghiên cứu hoạt động.

Vấn đề về cặp gần nhất

Bài toán về cặp gần nhất yêu cầu tìm hai điểm gần nhất trong tập hợp n điểm. Đây là bài toán đơn giản nhất trong nhiều bài toán trong hình học tính toán đề cập đến sự gần nhau của các điểm trong mặt phẳng hoặc các không gian có chiều cao hơn. Các điểm được đề cập có thể đại diện cho các đối tượng vật lý như máy bay hoặc bưu điện cũng như hồ sơ cơ sở dữ liệu, mẫu thống kê, trình tự DNA, v.v. Một kiểm soát viên không lưu có thể quan tâm đến hai chiếc máy bay gần nhất như là những ứng cử viên có khả năng xảy ra va chạm cao nhất. Một nhà quản lý dịch vụ bưu điện khu vực có thể cần một giải pháp cho vấn đề cặp gần nhất để tìm các địa điểm bưu điện ứng cử viên sẽ đóng cửa.

Một trong những ứng dụng quan trọng của bài toán cặp gần nhất là phân tích cụm trong thống kê. Dựa trên n điểm dữ liệu, phân tích cụm phân cấp tìm cách phân tích tổ chức chúng trong một hệ thống phân cấp của các cụm dựa trên một số chỉ số tương tự. Đối với dữ liệu số, số liệu này thường là khoảng cách Euclide; đối với dữ liệu văn bản và dữ liệu phi số khác, các chỉ số như khoảng cách Hamming (xem Vấn đề 5 trong các lỗi chính của phần này) được sử dụng. Thuật toán từ dưới lên bắt đầu với mỗi phần tử là một cụm riêng biệt và hợp nhất chúng thành các cụm lớn hơn liên tiếp bằng cách kết hợp cặp cụm gần nhất.

Để đơn giản, chúng ta xem xét trường hợp hai chiều của cặp prob lem gần nhất. Chúng tôi giả định rằng các điểm được đề cập đến được xác định theo kiểu chuẩn bởi tọa độ Descartes (x, y) của chúng và khoảng cách giữa hai điểm pi (xi, yi) và pj (xj , yj) là khoảng cách Euclide tiêu chuẩn

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Cách tiếp cận brute-force để giải quyết vấn đề này dẫn đến một thuật toán sai lầm sau đây: tính toán khoảng cách giữa mỗi cặp điểm phân biệt và tìm một cặp có khoảng cách nhỏ nhất. Tất nhiên, chúng ta không muốn tính khoảng cách giữa cùng một cặp điểm hai lần. Để tránh làm như vậy, chúng ta chỉ xem xét các cặp điểm (pi, pj) mà $i < j$.

Mã giả bên dư ới tính toán khoảng cách giữa hai điểm gần nhất;
nhận đư ợc các điểm gần nhất chỉ cần một sửa đổi nhỏ.

```
THUẬT TOÁN BruteForceClosestPair (P)
    // Tìm khoảng cách giữa hai điểm gần nhất trong mặt phẳng bằng brute force // Đầu
    vào: Danh sách P gồm n (n ≥ 2) điểm p1 (x1 , y1 ) ,. . . , pn (xn , yn )
    // Đầu ra: Khoảng cách giữa cặp điểm gần nhất d    ∞ đối với i
    1 đến n - 1 do


        for j    i + 1 to n do
            d    min (d, sqrt ((xi - xj ) 2 + (yi - yj ) 2 )) // sqrt là căn bậc hai
    trở lại d
```

Hoạt động cơ bản của thuật toán là tính căn bậc hai. Trong thời đại của máy tính điện tử với nút căn bậc hai, ngư ời ta có thể tin rằng tính căn bậc hai là một phép toán đơn giản, chẳng hạn như phép cộng hoặc phép nhân. Tất nhiên, nó không phải là. Đối với ngư ời mới bắt đầu, ngay cả đối với hầu hết các số nguyên, căn bậc hai là số vô tỉ do đó chỉ có thể tìm đư ợc một cách gần đúng. Hơn nữa, việc tính toán các phép tính gần đúng như vậy không phải là một vấn đề tầm thư ờng. Như ng trên thực tế, việc tính toán căn bậc hai trong vòng lặp có thể tránh đư ợc! (Bạn có thể nghĩ cách nào không?) Bí quyết là nhận ra rằng chúng ta có thể đơn giản bỏ qua hàm căn bậc hai và so sánh các giá trị (xi - xj) 2 + (yi - yj) 2 với chính chúng. Chúng ta có thể làm điều này bởi vì một số nhỏ hơn mà chúng ta lấy căn bậc hai, căn bậc hai của nó càng nhỏ, hoặc, như các nhà toán học nói, hàm căn bậc hai đang tăng lên một cách nghiêm ngặt.

Khi đó, hoạt động cơ bản của thuật toán sẽ là bình phư ơng một số. Các
Số lần nó sẽ đư ợc thực thi có thể đư ợc tính như sau:

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$$

$$= 2[(n-1) + (n-2) + \ldots + 1] = (n-1)n \quad (n^2).$$

Tất nhiên, việc tăng tốc vòng lặp trong cùng của thuật toán chỉ có thể làm giảm thời gian chạy của thuật toán bằng một hệ số không đổi (xem Vấn đề 1 trong các bài tập của phần này), như ng nó không thể cải thiện lớp hiệu quả tiệm cận của nó. Trong Chư ơng 5, chúng ta thảo luận về một thuật toán tuyến tính cho vấn đề này, thuật toán này dựa trên một kỹ thuật thiết kế phức tạp hơn.

Vấn đề lồi lõm

Đối với vấn đề khác - đó là tính toán vỏ lồi. Tìm phần lồi của một tập hợp các điểm trong mặt phẳng hoặc một không gian có chiều cao hơn là một trong những vấn đề quan trọng nhất — một số ngư ời tin rằng vấn đề quan trọng nhất — trong hình học đặt com. Sự nổi bật này là do một loạt các ứng dụng trong đó

vấn đề này cần được giải quyết, tự nó hoặc là một phần của nhiệm vụ lớn hơn. Sev eral các ứng dụng như vậy dựa trên thực tế là các vỏ lồi cung cấp các xấp xỉ thuận tiện về hình dạng đối tượng và các tập dữ liệu đã cho. Ví dụ, trong máy tính, việc thay thế các đối tượng bằng vỏ lồi của chúng sẽ tăng tốc độ phát hiện va chạm; ý tưởng tương tự cũng được sử dụng trong việc lập kế hoạch đường đi cho các chuyến thám hiểm sao Hỏa. Vỏ lồi được sử dụng trong các bản đồ trợ năng tính toán được tạo ra từ ảnh vệ tinh của Hệ thống Thông tin Địa lý. Chúng cũng được sử dụng để phát hiện các ngoại lệ bằng một số kỹ thuật thống kê. Một thuật toán hiệu quả để tính toán đường kính của một tập hợp điểm, là khoảng cách lớn nhất giữa hai trong số các điểm, cần vỏ lồi của tập hợp để tìm khoảng cách lớn nhất giữa hai trong số các điểm cực trị của nó (xem bên dưới). Cuối cùng, vỏ lồi rất quan trọng để giải quyết nhiều vấn đề tối ưu hóa, bởi vì các điểm cực trị của chúng cung cấp một tập hợp giới hạn các ứng cử viên giải pháp.

Chúng ta bắt đầu với định nghĩa của một tập hợp lồi.

ĐỊNH NGHĨA Một tập hợp các điểm (hữu hạn hoặc vô hạn) trong mặt phẳng được gọi là lồi nếu với hai điểm p và q bất kỳ trong tập hợp thì toàn bộ đoạn thẳng có điểm cuối tại p và q thuộc tập hợp đó.

Tất cả các tập hợp được mô tả trong Hình 3.4a là lồi, và đường thẳng, tam giác, hình chữ nhật và nói chung là bất kỳ đa giác lồi nào, 1 hình tròn và toàn bộ mặt phẳng. Mặt khác, các tập được mô tả trong Hình 3.4b, bất kỳ tập hợp hữu hạn nào gồm hai hoặc nhiều điểm phân biệt, ranh giới của bất kỳ đa giác lồi và một chu vi nào đều là ví dụ về các tập hợp không lồi.

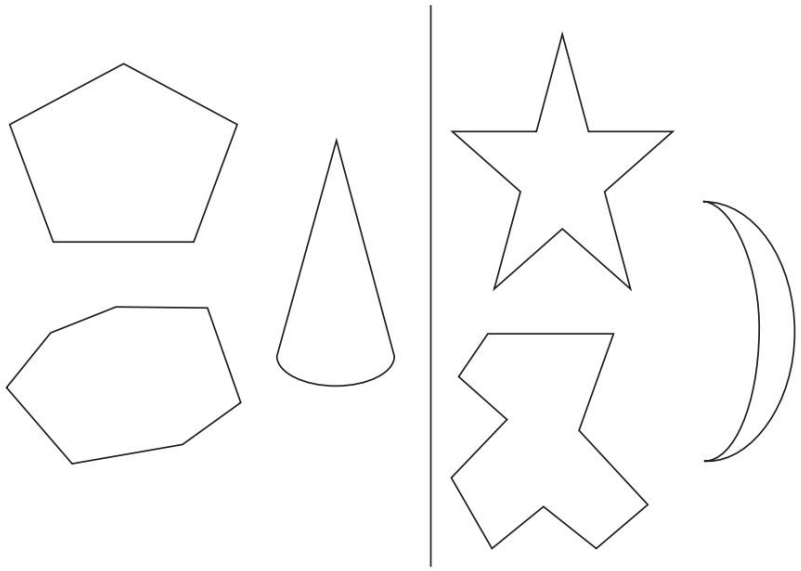
Bây giờ chúng ta đã sẵn sàng cho khái niệm về thân tàu lồi. Nói một cách trực quan, lồi của một tập hợp n điểm trong mặt phẳng là đa giác lồi nhỏ nhất chứa tất cả chúng nằm trong hoặc trên biên của nó. Nếu công thức này không kích thích sự nhiệt tình của bạn, hãy coi vấn đề như một trong những chướng ngại vật đang ngủ bằng hàng rào có chiều dài ngắn nhất. Sự giải thích này là do D. Harel [Har92]; Tuy nhiên, nó có phần sống động, bởi vì các hàng rào phải được dựng lên ngay tại những điểm mà một số con hổ ngủ! Có một cách giải thích khác, thuần thục hơn nhiều về khái niệm này. Hãy tưởng tượng rằng các điểm được đề cập được biểu diễn bằng những chiếc đinh đóng vào một tấm ván ép lớn đại diện cho mặt phẳng. Lấy một sợi dây chun và kéo căng nó để bao gồm tất cả các móng, sau đó để nó cố định vào vị trí. Phần lồi của thân tàu là khu vực được giới hạn bởi dây chun buộc (Hình 3.5).

Một định nghĩa chính thức về vỏ lồi có thể áp dụng cho các tập hợp tùy ý, bao gồm các tập hợp các điểm tình cờ nằm trên cùng một đường thẳng, theo sau.

ĐỊNH NGHĨA Phần lồi của tập hợp các điểm S là tập lồi nhỏ nhất chứa S. (Yêu cầu “nhỏ nhất” có nghĩa là phần lồi của S phải là tập con của bất kỳ tập lồi nào chứa S.)

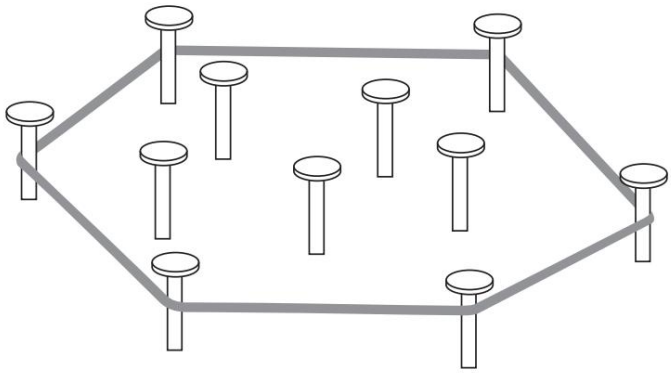
Nếu S là lồi, thì vỏ lồi của nó hiển nhiên là chính S. Nếu S là tập hợp hai điểm thì phần lồi của nó là đoạn thẳng nối các điểm này. Nếu S là bộ ba

1. Bởi "một hình tam giác, một hình chữ nhật và nói chung là bất kỳ đa giác lồi nào", chúng tôi muốn nói ở đây là một vùng, tức là, tập hợp các điểm cả bên trong và bên ngoài ranh giới của hình dạng được đề cập.



(Một) (b)

HÌNH 3.4 (a) Các tập hợp lồi. (b) Tập hợp không lồi.

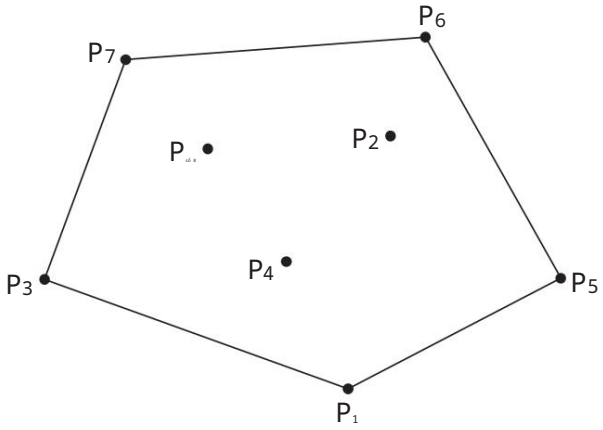


HÌNH 3.5 Giải thích dây cao su của thân tàu lồi.

các điểm không nằm trên cùng một đường thẳng thì lồi của nó là tam giác có các đỉnh tại ba điểm đã cho; nếu ba điểm nằm trên cùng một đường thẳng thì phần lồi là đoạn thẳng có các điểm cuối của nó tại hai điểm cách xa nhau nhất. Để biết ví dụ về vỏ lồi cho một tập hợp lớn hơn, hãy xem Hình 3.6.

Một nghiên cứu về các ví dụ làm cho định lý sau đây trở thành một kết quả mong đợi.

LÝ THUYẾT Phần lồi của tập hợp S bất kỳ gồm n> 2 điểm không nằm trên cùng một đường thẳng là một đa giác lồi với các đỉnh tại một số điểm của S. (Nếu tất cả các điểm nằm trên cùng một đường thẳng thì đa giác suy biến thành một đoạn thẳng nhưng vẫn có các điểm cuối tại hai điểm của S.)



HÌNH 3.6 Vỏ lồi cho tập hợp tám điểm này là đa giác lồi với các đỉnh tại p1 , p5 , p6 , p7 và p3 .

Bài toán thân lồi là bài toán dựng vỏ lồi cho một tập hợp n điểm S cho trư ớc. Để giải quyết nó, chúng ta cần tìm các điểm sẽ là đỉnh của đa giác đư ợc đề cập. Các nhà toán học gọi các đỉnh của một đa giác như vậy là “điểm cực trị”. Theo định nghĩa, một điểm cực trị của tập lồi là một điểm của tập hợp này không phải là điểm giữa của bất kỳ đoạn thẳng nào có điểm cuối trong tập hợp. Ví dụ, các điểm cực trị của một tam giác là ba đỉnh của nó, các điểm cực trị của một đư ờng tròn là tất cả các điểm thuộc chu vi của nó và các điểm cực trị của phần lồi của tập hợp tám điểm trong Hình 3.6 là p1 , p5 , p6 , p7 và p3 .

Các điểm cực trị có một số tính chất đặc biệt mà các điểm khác của tập lồi không có. Một trong số chúng đư ợc khai thác bằng phư ơng pháp simplex, một thuật toán rất quan trọng đư ợc thảo luận trong Phần 10.1. Thuật toán này giải các bài toán lập trình tuyến tính, là các bài toán tìm điểm cực tiểu hoặc cực đại của một hàm tuyến tính gồm n biến chịu các ràng buộc tuyến tính (xem ví dụ Bài toán 12 trong các bài tập của phần này và các Phần 6.6 và 10.1 để thảo luận chung). Tuy nhiên, ở đây, chúng tôi quan tâm đến các điểm cực trị vì việc xác định chúng giải quyết đư ợc vấn đề thân tàu lồi. Trên thực tế, để giải quyết triệt để vấn đề này, chúng ta cần biết nhiều hơn một chút về điểm nào trong số n điểm của một tập hợp đã cho là điểm cực trị của phần lồi của tập hợp đó: chúng ta cần biết những cặp điểm nào cần đư ợc nối với nhau để tạo thành biên. của vỏ tàu lồi. Lư u ý rằng vấn đề này cũng có thể đư ợc giải quyết bằng cách liệt kê các điểm cực trị theo thứ tự chiều kim đồng hồ hoặc ngư ợc chiều kim đồng hồ.

Vì vậy, làm thế nào chúng ta có thể giải quyết vấn đề lồi-thân tàu một cách vũ phu? Nếu bạn không thấy một kế hoạch ngay lập tức cho một cuộc tấn công trực diện, đừng lo lắng: vấn đề thân tàu lồi là một vấn đề không có giải pháp thuật toán rõ ràng. Tuy nhiên, có một thuật toán đơn giản như ng không hiệu quả dựa trên quan sát sau đây về các đoạn thẳng tạo nên ranh giới của một thân lồi: một đoạn thẳng nối hai điểm pi và pj của tập hợp n điểm là một phần của vỏ lồi. ranh giới nếu và

chỉ khi tất cả các điểm khác của tập hợp nằm trên cùng một phía của đường thẳng đi qua hai điểm này. tạo nên ranh giới lồi của vỏ tàu.

Một số dữ kiện cơ bản từ hình học phân tích là cần thiết để thực hiện thuật toán này. Đầu tiên, đường thẳng qua hai điểm (x1 , y1), (x2 , y2) trong mặt phẳng tọa độ có thể được xác định bằng phương trình

$$ax + by = c,$$

trong đó a = y2 - y1 , b = x1 - x2 , c = x1 y2 - y1 x2 .

Thứ hai, đường thẳng như vậy chia mặt phẳng thành hai nửa mặt phẳng: với tất cả các điểm trong một trong số chúng, ax + by> c, trong khi đối với tất cả các điểm thuộc nửa mặt phẳng khác , ax + bởi <c. (Tất nhiên, đối với các điểm trên đường thẳng, ax + by = c.) Vì vậy, để kiểm tra xem các điểm nhất định có nằm trên cùng một phía của đoạn thẳng hay không, chúng ta chỉ cần kiểm tra xem biểu thức ax + by - c có trùng không. ký tên cho mỗi điểm này. Chúng tôi để lại các chi tiết thực hiện như một bài tập.

Hiệu quả thời gian của thuật toán này là bao nhiêu? Đó là trong O (n3): với mỗi n (n - 1) / 2 cặp điểm phân biệt, chúng ta có thể cần tìm dấu của ax + by - c đối với n - 2 điểm khác nhau. Có nhiều thuật toán hiệu quả hơn cho vấn đề quan trọng này, và chúng ta sẽ thảo luận về một trong số chúng ở phần sau của cuốn sách.

Bài tập 3.3

1. Giả sử rằng sqrt mất khoảng thời gian dài hơn khoảng 10 lần so với mỗi hoạt động khác trong vòng lặp trong cùng của BruteForceClosestPoints, được cho là mất cùng một khoảng thời gian, hãy ước tính thuật toán sẽ chạy nhanh hơn bao nhiêu sau khi cải tiến được thảo luận trong Phần 3.3 .

2. Bạn có thể thiết kế một thuật toán hiệu quả hơn thuật toán dựa trên chiến lược brute force để giải bài toán cặp gần nhất cho n điểm x1 , x2 , đường thẳng thực không? · · · , xn vào

3. Cho x1 < x2 <. ˙ . < xn là các số thực thể hiện tọa độ của n làng nằm trên một con đường thẳng. Một bưu điện cần phải được xây dựng ở một trong những ngôi làng này. Một. Thiết kế một thuật toán hiệu quả để tìm vị trí bưu điện giảm thiểu khoảng cách trung bình giữa các làng và bưu điện. b. Thiết kế một thuật toán hiệu quả để tìm vị trí bưu điện giảm thiểu khoảng cách tối đa từ một làng đến bưu điện.

2. Để đơn giản, ở đây chúng ta giả sử rằng không có ba điểm nào của một tập hợp đã cho nằm trên cùng một đường thẳng. Một sửa đổi cần thiết cho trường hợp chung được để lại cho các bài tập.