Thuật toán ứng dụng buổi 3

Bài 1. Disjoint Segment

Tìm tập các khoảng thời gian không giao nhau lớn nhất

- Cho đầu vào là tập X gồm n khoảng thời gian X = {(a1, b1), . . . , (an, bn)} với ai < bi
- Hãy tìm và đưa ra tập con lớn nhất của X chứa các khoảng thời gian này sao cho không có 2 khoảng nào giao nhau

Input

- Dòng 1 là giá trị của n (*1 <= n <= 100000*)
- Các dòng tiếp theo là các khoảng thời gian (a_i, b_i) ($1 <= a_i <= b_i <= 1000000$)

Output

- Số lượng phần tử của tập con lớn nhất

stdin	stdout
6 0 10 3 7 6 14 9 11 12 15 17 19	4

Ý tưởng

- Hai khoảng (a1,b1) và (a2,b2) không giao nhau nếu b1<=a2
- Sắp xếp các đoạn theo thứ tự tăng dần thời gian, hoặc độ dài khoảng
- Tham lam 1. Chọn khoảng bắt đầu sớm nhất
- Tham lam 2. Chọn khoảng ngắn nhất
- Tham lam 3. Chọn khoảng kết thúc sớm nhất
- Vét cạn: Duyệt hết tập con ➔ không khả thi

Câu hỏi

Câu 1. Trong các thuật toán tham lam 1,2 trên hãy chỉ ra ít nhất 1 phản ví dụ của nó?

Câu 2. Liệu bạn có thể chứng minh tính chính xác của thuật toán tham lam 3?

Câu 3. Độ phức tạp trong trường hợp tối nhất của thuật toán 3?

Bài 2. MAX-DISTANCE SUB-SEQUENCE

Cho 1 tập gồm N điểm trong không gian 1 chiều (đường thẳng). **Độ kết nối** của cụm (tập con) gồm C phần tử được tạo ra từ tập ban đầu được tính bằng khoảng cách của 2 điểm gần nhau nhất trong cụm

VD. Cụm gồm C=5 điểm là 12,30,15,10,37 thì độ kết nối của cụm sẽ là 2, chính là khoảng cách nhỏ nhất giữa 12 và 10

stdin	stdout
1 5 3 1 2 8 4 9	3

- Dòng đầu tiên sẽ là số lượng test trong bộ test T (1 <= T <= 20)

- Các dòng tiếp theo sẽ là chi tiết các test. Mỗi test tương ứng gồm 2 giá trị là N và C, trong đó N là số lượng phần tử và C là kích thước tập con. Trong 1 bộ dữ liệu thì giá trị N sẽ là giống nhau trong các test.
- Các dòng tiếp sau sẽ lần lượt là giá trị tọa độ của các điểm trong tập N điểm
- Giá trị N($2 \le N \le 100,000$) và tọa độ các điểm x1,…, xn ($0 \le x \le 1,000,000,000$)
- Đầu ra sẽ là độ kết nối lớn nhất của tập con với kích thước C tương ứng (với các test). Nếu có T test thì sẽ có T giá trị đầu ra, với mỗi kết quả đầu ra được in trên 1 dòng.

Gợi ý.

- Nếu dùng vét cạn để tìm khoảng cách nhỏ nhất giữa 2 điểm trong không gian 1D thì sẽ mất thời gian $O(n^2)$. Ta có thể sắp xếp dãy trước (chi phí sắp xếp là O(nlogn) hoặc $O(n^2)$), rồi tìm khoảng cách nhỏ nhất với thời gian chỉ O(n).
- Nếu vét cạn hết các tập con có thể thì thời gian quá lớn (2^n), ta cần cách tiếp cận thông minh hơn.
- Nếu xuất phát từ tập con 2 điểm, thì tập con có độ kết nối lớn nhất sẽ tạo bởi việc chọn 2 điểm xa nhất có thể. Khi thêm điểm tiếp theo thì chọn điểm sao cho nó xa 2 điểm kia nhất (điểm gần giữa 2 điểm đã chọn trước).
- Cách tiếp cận tham lam?
  - Sắp xếp trước danh sách N điểm
  - Chọn tập đầu là 2 điểm mút (2 điểm xa nhất)
  - Lần lượt chọn các điểm tiếp theo(cho tới khi đủ C) là điểm giữa dãy (đoạn con)

VD. Các điểm sau khi sắp xếp là 1,10,12,15,21,34

Và C = 3

Tập điểm 2 ban đầu là {1,34}, vậy điểm tiếp sẽ là 15 với độ kết nối là 14

Nếu C = 4

Tập điểm 2 ban đầu là {1,34}, điểm tiếp sẽ là 15, và điểm tiếp sẽ là 21 với độ kết nối sẽ là 6

Câu hỏi 1. Lần chia tiếp theo ta sẽ chia tại nửa nào?

Câu hỏi 2. Độ phức tạp của thuật toán sẽ là?

Nếu các phần tử trong dãy lệch nhau nhiều thì cách chia có khác?

VD. Dãy 1,5,10,21,55,100,300,1000

Gợi ý tham lam 2.

- Tính khoảng cách của 2 điểm gần nhau trước
- Chia tổng khoảng cách ra sao cho đều nhất có thể

Với dãy ví dụ thì các khoảng cách sẽ lần lượt là

4,5,11,34,45,200,700

Vậy điểm thứ 3 được chọn (sau khi chọn 2 mút là 1 và 1000) sẽ là 300, điểm thứ 4 được chọn sẽ là 100,…

Câu hỏi 3. Thuật toán tham lam này sẽ có độ phức tạp là bao nhiêu?

Cách chia như vậy có đảm bảo là đúng?

VD với dãy ban đầu 1,5,10,40,50,70,100

- Với C = 2 thì ta chọn 1 và 100
- Với C = 3 ta chọn 1,50,100
- Nhưng C=4 thì liệu 50 còn được chọn?
  Ta sẽ chọn tối ưu là 1,40,70 và 100

Vậy ngoài việc cố định được mút là điểm nhỏ nhất và lớn nhất thì KHÔNG có gì đảm bảo các điểm giữa sẽ được chọn trong các giá trị C tiếp theo.

Thuật toán đúng sẽ làm thế nào?

- Sắp xếp dãy trước với thời gian cỡ O(nlogn)
- Min và Max chắc chắn sẽ là 2 điểm thuộc C
- Min và Max sẽ quyết định độ lớn tối đa của kết nối.
  VD. Với dãy 1,5,10,40,50,70,100 và C = 5 thì độ kết nối lớn nhất sẽ phải nhỏ hơn (100-1)/(5-2) là cỡ 33
- Ta sẽ dò tìm giá trị chính xác của độ kết nối lớn nhất dùng kết hợp thuật toán tìm kiếm nhị phân và backtracking

B1. Sắp xếp dãy

B2. Tìm min, max

B3. Tìm giá trị ban đầu (giới hạn trên) của độ kết nối sẽ là (max-min)/(C-2) với C là số lượng phần tử của tập con

B4. A_1= min, A_C = max, k=(max-min)/(C-2), kmax=k lần lượt tìm A_2,A_3,..A_{max-1}

Tìm A_i bằng cách tìm nhị phân phần tử gần A_{i-1} + k trong dãy, và cập nhật lại k_{max} = min(k_{max}, A_i-A_{i-1})

Nếu dãy tiếp theo KHÔNG còn đủ phần tử để chọn thì quay lại bước trước chọn phần tử ngay trước A_i đã chọn trong dãy.

Nếu chọn đủ phần tử thì ghi nhận lại giá trị k_{max} đã tìm được, sau đó cũng quay lui để thử tìm xem còn cách chọn nào cho k_{max} lớn hơn không. Ta cũng tiến hành quay lui như trường hợp không còn phần tử nào để chọn.

Ví dụ với dãy 1, 5,10,30,45,55,70,100 và C = 5

A1= 1 và A5 = 100, k=33

Tìm A2. Bằng cách tìm nhị phân phần tử gần A1 + k = 34 nhất trong dãy 5,10,30,45,55,70. Ta chọn được A2=30. Với kmax sẽ là min(33,29) = 29

Tìm A3. Bằng cách tìm nhị phân phần tử gần A2 + k = 63 nhất trong dãy 45,55,70. Ta chọn được 70. Với kmax là min(29,40) = 29

Tìm A4. Lúc này dãy tiếp theo là rỗng nên phải backtracking lại chọn A3

Ta sẽ chọn A3 mới là phần tử gần hơn trong dãy còn lại của **45,55**,70. Ta chọn 55, với kmax = min(29,25) = 25.

Tìm A4. Trong dãy 70. Ta chọn 70 và kmax = min(25,15)=15

Giờ muốn xét nốt phương án tối ưu hơn?

Quay lại tìm A4 → Quay lại tìm A3. Chọn A3 mới là 45 thì bằng kmax lớn nhât ta tìm được ở lần trước → quay lui lại A2 → chọn A2 là giá trị trước 30, là 10 →kmax nhỏ hơn giá trị lớn nhất ta tìm được → dừng

Vậy trong dãy trên với C=5 ta tìm được kmax = 15

Câu hỏi 4. Ta sẽ không quay lui hết mà dừng quay lui tới khi nào?

Câu hỏi 5. Độ phức tạp tính toán của thuật toán trên?

Bài 3. **Inversion – đảo ngược**

- Cho một dãy số nguyên $a_1, a_2, \ldots, a_n$. Một cặp (a_i, a_j) được gọi là bị đảo ngược nếu $i<j$ nhưng giá trị $a_i > a_j$
- Hãy đếm số lượng các cặp bị đảo ngược trong dãy

Input

- Dòng 1: là giá trị của n ($1 <= n <= 10^6$)
- Dòng 2: là các giá trị của $a_1, a_2, \ldots, a_n$. **($0 <= a_i <= 10^6$)**

Output

Số lượng cặp bị đảo Q **module $10^9 + 7$**

stdin	stdout
6 3 2 4 5 6 1	6

Gợi ý:

- Số lượng cặp trong dãy n phần tử sẽ là n(n-1)/2
- Duyệt và đếm số lượng cặp theo phương pháp tham lam thông thường: $O(n^2)$
- Thuật toán nhanh hơn?

Ý tưởng chia để trị?

- Chia dãy thành 2 nửa bằng nhau:
  - Số lượng đổi chỗ gồm: là tổng số đổi chỗ trong nửa trái L, và nửa phải R
  - Cộng với số lượng đổi chỗ của 1 phần tử thuộc nửa trái sang 1 phần tử nửa phải - M
- Cách tìm số lượng đổi chỗ giữa 2 nửa M sao cho nhanh?
  - Nếu duyệt vét cạn vẫn là $n/2 \times n/2 = n^2/4 \rightarrow$ thuật toán vẫn là $O(n^2)$ như vét cạn
  - Không vét cạn?

Giả sử dãy trái và phải đều có thứ tự tăng thì số lượng đổi chỗ giữa 1 phần tử nửa trái và 1 phần tử nửa phải sẽ tính là bao nhiêu

VD. Nửa trái L = 1,5,10, 12 và nửa phải R= 3,11,17,21

Hãy để ý tới giá trị chốt của mỗi nửa

- Nửa trái: Chốt là phần tử cuối cùng

- Nửa phải: Chốt là phần tử đầu tiên

Với dãy trên thì rõ ràng 1 sẽ ko cần đổi chỗ, nhưng 5,10,12 sẽ là 3 lần đổi chỗ có thể với 3 bên nửa phải. 12 sẽ có 1 lần đổi chỗ với 11 bên nửa phải

Cách tính?

- So sánh với phần tử cuối cùng của nửa trái hoặc đầu tiên nửa phải
- Thời gian tìm: O(n)

Vậy trong ý tưởng chia để trị ta kết hợp việc đếm với việc sắp xếp dãy như trong sắp xếp trộn – Merge Sort

Câu hỏi

Câu 1. Công thức đệ quy tổng quát cho ý tưởng chia để trị sẽ là?

Câu 2. So với thuật toán vét cạn thì thuật toán chia để trị này có nhược điểm gì? Liệu thuật toán vét cạn có tồi hơn chia để trị không?
