Xin được 1 phút quảng cáo cho tất cả nỗ lực của mình để hoàn thành lời giải dưới đây:

Mình nhận đào tạo lập trình giải thuật thi đấu (Competitive Programming):

Học sinh cấp 2, 3:

- Thi vào lớp 10 chuyên tin

- Thi học sinh giỏi tin học cấp tỉnh thành

- Thi vào đội tuyển quốc gia

Sinh viên:

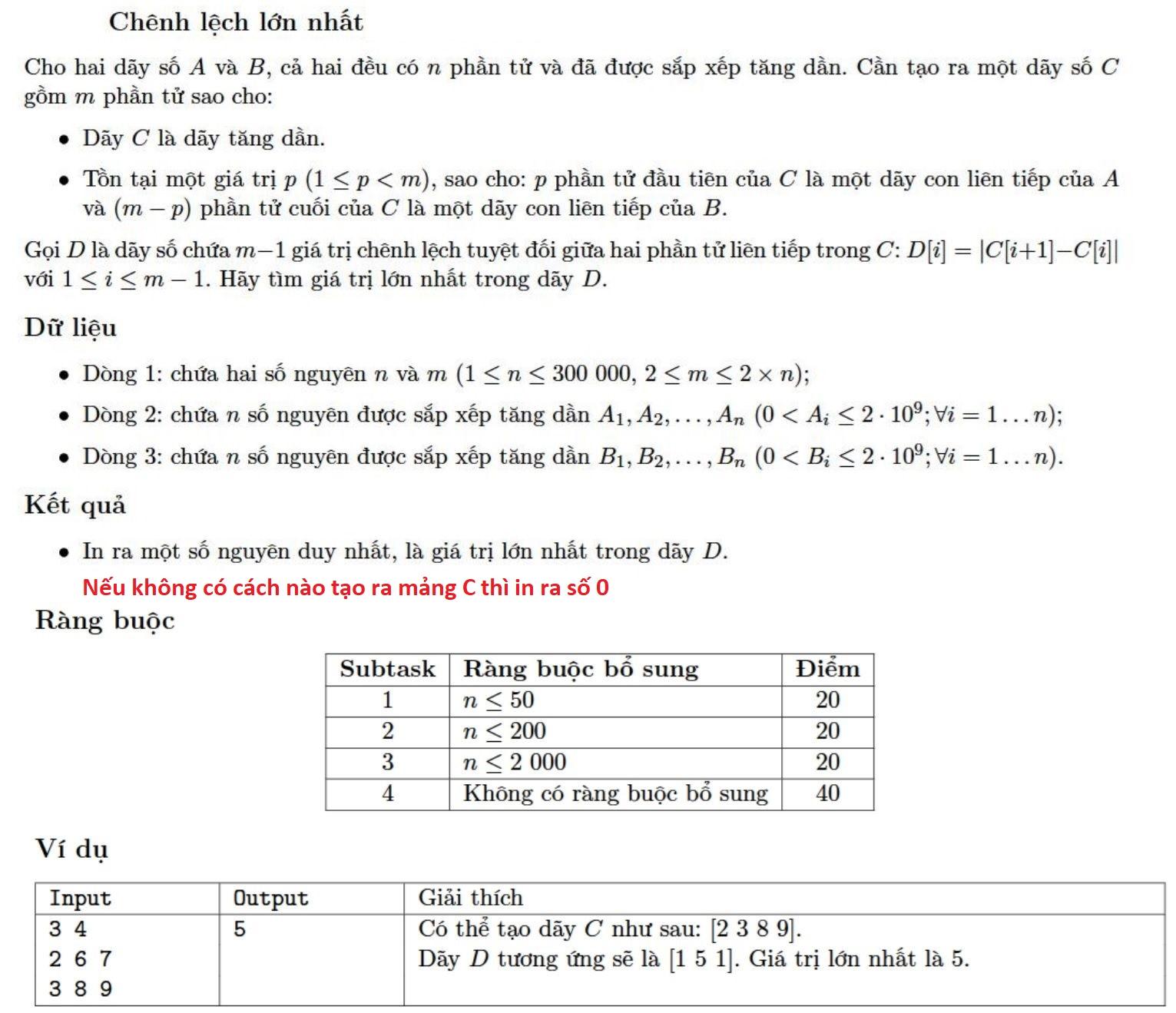
- Thi olympic tin học sinh viên/icpc

Người đi làm/Sinh viên:

- Phỏng vấn về thuật toán

Đánh giá nhận xét của các phụ huynh/học viên và Thành quả mình đã giúp được cho các bạn học viên - xem dưới bình luận của bài này: <https://www.facebook.com/share/1XawfdQ6Tc/>

**Chênh Lệch Lớn Nhất**



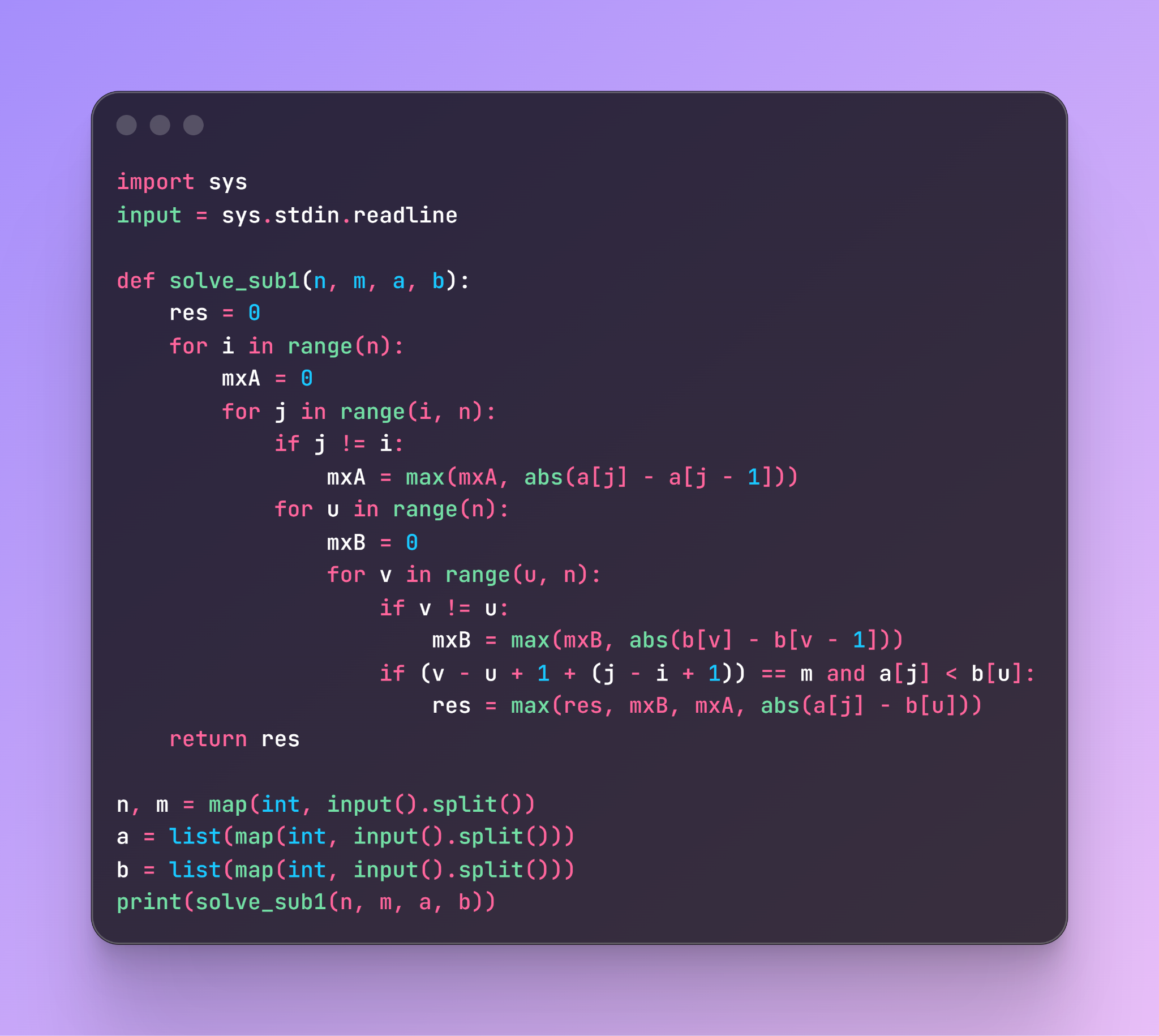
Bài này mình được một bạn học viên hỏi nên sẵn tiện mình viết bài giải chi tiết để nhiều bạn được tham khảo luôn. Hy vọng mang đến được giá trị cho các bạn 😇. Bạn nào muốn thử nộp bài để hệ thống chấm xem có đúng hay không thì link bài đây nhé, các bạn nào chưa có tài khoản thì tạo tài khoản rồi đăng nhập nộp bài thử hen: <https://codeforces.com/group/py59YCge5m/contest/637051/problem/C>

Subtask 1:

Duyệt trâu các khả năng ghép đoạn có thể xảy ra. Ta sẽ for i là chỉ số của vị trí bắt đầu bên dãy con liên tiếp của A, sau đó for j là chỉ số của vị trí kết thúc bên dãy con liên tiếp của A, tiếp theo trong đó ta sẽ for u là chỉ số bắt đầu của dãy con liên tiếp B và for tiếp v là chỉ số kết thúc của dãy con liên tiếp B. Lúc for phân đoạn của dãy A, ta có thể vừa tăng j vừa tính mxA (chênh lệch lớn nhất giữa hai phần tử sao cho hai phần tử nằm trong đoạn con liên tiếp A) . Tương tự với B, ta cũng sẽ vừa tăng v vừa tính mxB. Khi đấy, ta chỉ cần tính thêm chênh lệch giữa phần tử cuối của phân đoạn A và phần tử đầu của phân đoạn B (abs(B[u] - A[j])), kết hợp max với mxB, mxA là được chênh lệch của các phần tử trong mảng C. Lưu ý, phải xét a[j] < b[u] thì mới tính vào kết quả.

Độ phức tạp: O(n ^ 4)

Source Code Python, copy tại link này: <https://ray.so/A6TYsaA>



---------------------------------------

Subtask 2:

Ta nhận xét rằng : nếu duyệt 3 vòng lồng nhau với 3 biến chạy j, u, len trong đó:

j là vị trí kết thúc của phân đoạn A

u là vị trí bắt đầu của phân đoạn B

len là độ dài của phân đoạn A được lấy nằm trong C

Ta có thể dễ dàng suy ra được phân đoạn B sẽ có độ dài là m - len và có vị trí kết thúc là u + (m - len) - 1.

Khi đấy, việc ta cần xử lý là làm sao để tính nhanh được chênh lệch giữa các phần tử trong phân đoạn A được chọn và chênh lệch giữa các phần tử trong phân đoạn B được chọn.

Ta gọi mxA[i][j] là độ chênh lệch lớn nhất của phân đoạn A bắt đầu từ i và kết thúc tại j. Ta sẽ có công thức truy hồi sau: mxA[i][j] = max(mxA[i][j - 1], abs(a[j] - a[j - 1]))

Tương tự ta cũng sẽ tạo được mxB[i][j].

Lưu ý: vẫn phải kết hợp xét điều kiện a[j] < b[u]

Độ phức tạp: O(n ^ 3)

Source Code Python, copy tại link này: <https://ray.so/LA5hOl3>



---------------------------------------

Subtask 3:

Ta có thể chia dãy D vào trong các trường hợp sau:

Trường hợp 1: phần tử D lớn nhất là trị tuyệt đối của hai phần tử trong mảng A

Trường hợp 2: phần tử D lớn nhất là trị tuyệt đối của hai phần tử trong mảng B

Trường hợp 3: phần tử D lớn nhất là trị tuyệt đối của phần tử cuối của dãy con liên tiếp từ mảng A và phần tử đầu của dãy con liên tiếp từ mảng B

Ta thấy rằng ta chỉ cần tối ưu, nên ta sẽ tính riêng max tối ưu cho từng trường hợp và max chúng lại với nhau:

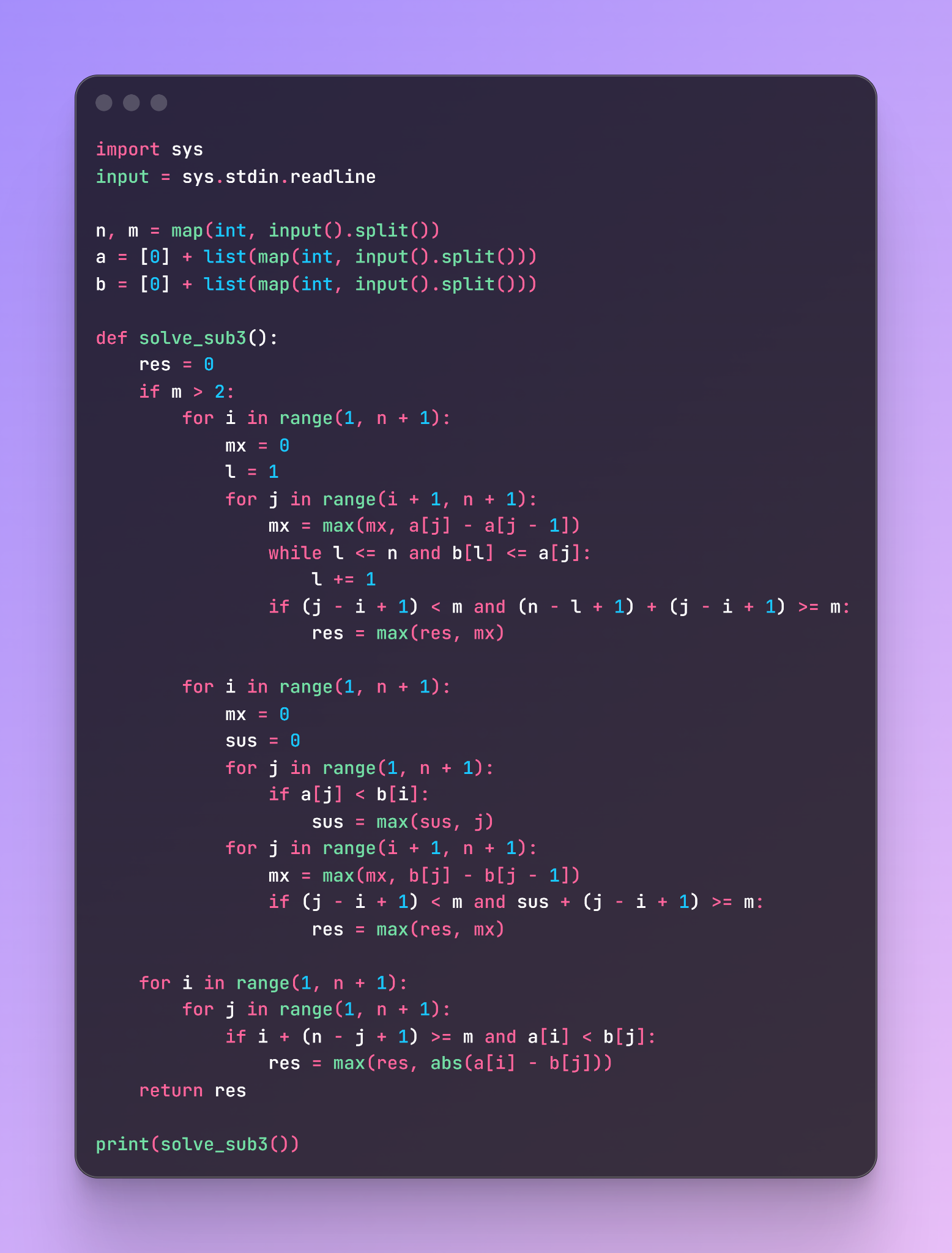
Trường hợp 1: Duyệt hết các phân đoạn con có thể của mảng A. Khi đấy, ta sẽ lấy max trị tuyệt đối giữa hai phần tử liền kề. Sau đó ta sẽ xem xét thử rằng có khả năng tạo được một mảng C m phần tử có sự xuất hiện của phân đoạn con này hay không. Nhận xét rằng nếu muốn tạo được, thì ta sẽ gọi u là vị trí nhỏ nhất sao cho b[u] > a[j] (j là vị trí cuối của phân đoạn con của A), tạo được chỉ khi min(n - u + 1, m - 1) (độ dài tối đa có thể tạo được của phân đoạn lấy từ B) + (j - i + 1) lớn hơn hoặc bằng m. Việc tính toán vị trí u có thể duy trì bằng hai con trỏ.

Trường hợp 2: Tương tự với trường hợp 1.

Trường hợp 3: Duyệt hết tất cả dãy A sau đó xem xét chúng với từng phần tử dãy B rồi lấy max trị tuyệt đối. O(N ^ 2). Ở đây, ta phải xét xem thử là có khả năng tạo một dãy gồm m phần tử nếu vị trí kết thúc là i bên mảng A và vị trí bắt đầu là u bên mảng B hay không. Việc này tương đương với việc i + (n - u + 1) có lớn hơn hoặc bằng m hay không vì i + (n - u + 1) là độ dài mảng C tối đa mà có thể tạo được tại vị trí kêt thúc i bên mảng A và vị trí bắt đầu u bên mảng B (kết hợp xét a[i] < b[u]).

Độ phức tạp: O(N ^ 2)

Source Code Python, copy tại link này: <https://ray.so/AiRU6sq>



---------------------------------------

Subtask 4:

Tương tự với subtask 3, nhưng ta sẽ xem xét tối ưu việc xử lý các trường hợp trong O(N).

Đối với trường hợp 1:

Nhận xét rằng, hai vị trí liền kề i , i + 1 có khả năng xuất hiện trong C khi có một vị trí j sao cho j > i và (j - i + 1) <= m - 1 (không thể chọn quá m - 1 phần tử vào C). Vị trí j này đồng thời phải thỏa mãn rằng vị trí u nhỏ nhất sao cho b[u] > a[j] phải min(m - 1, (n - u + 1)) + min(m - 1, j) (chỉ có thể chọn không quá m - 1 và không quá j phần tử có vị trí kết thúc là j trong đoạn A) >= m.

Ta nhận thấy min(m - 1, (n - u + 1)) + min(m - 1, j) có thể tính trước với mỗi j. Sau đó đối với mỗi i, ta chỉ cần lấy max của giá trị đó trong khoảng từ i + 1 đến (j - i + 1) <= m - 1 ⇔ j <= i + m - 2. Có thể dễ dàng xử lý bằng segment tree.

Đối với trường hợp 2: có thể xử lý tương tự vậy.

Đối với trường hợp 3:

Giả sử ta đang xét vị trí thứ i là vị trí kết thúc bên mảng A, ta có những nhận xét sau:

Nếu ta chọn b[u] < a[i] sao cho abs(a[i] - b[u]) là tối đa nhất có thể. Ta sẽ chọn b[u] tối thiểu nhất có thể để “mở rộng khoảng cách” giá trị giữa hai phần tử.

Nếu ta chọn b[u] > a[i] sao cho abs(a[i] - b[u]) là tối đa nhất có thể. Ta sẽ chọn b[u] lớn nhất có thể để “mở rộng khoảng cách” giá trị giữa hai phần tử.

Nếu ta chọn b[u] = a[i] thì điều này đang là giá trị tối thiểu có thể đạt được của giá trị lớn nhất mảng D nên không ảnh hưởng đến bài toán.

Ở vị trí i, ta phải chọn u sao cho i + (n - u + 1) >= m

i + n - u + 1 >= m

i + n + 1 - m >= u

Suy ra u <= i + n + 1 - m. Nếu ta có được i ta có thể dễ dàng tính vị trí tối đa mà u có thể đi tới. Tuy nhiên, chúng ta phải xét vị thêm rằng vị trí u phải có điều kiện là b[u] > a[i]. Khi đấy, ta vẫn duy trì hai con trỏ để tính toán cập nhật vị trí u. Đồng thời kết hợp với kiểu dữ liệu segment tree để thực hiện truy vấn lấy max, lấy min trên đoạn liên tiếp của B.

Độ phức tạp: O(n log n)

Source Code Python, copy tại link này: <https://ray.so/anH0uk0>

Lưu ý: Code C++ thì Segment Tree cài đặt theo kiểu đệ quy nên khi chuyển qua Python sẽ bị TLE. Ở code Python này cài đặt Segment Tree theo dạng mảng để chạy nhanh hơn, còn lại mọi logic xử lý y như code C++

|  |
| --- |
| import sys input = sys.stdin.readline  n, m = map(int, input().split()) a = [0] + list(map(int, input().split())) b = [0] + list(map(int, input().split()))  INF = 10\*\*18  size = 1 while size < n:  size <<= 1  mx = [-INF] \* (2 \* size) mn = [INF] \* (2 \* size)  def build\_max(arr):  for i in range(n):  mx[size + i] = arr[i + 1]  for i in range(size - 1, 0, -1):  mx[i] = max(mx[i << 1], mx[i << 1 | 1])  def build\_min(arr):  for i in range(n):  mn[size + i] = arr[i + 1]  for i in range(size - 1, 0, -1):  mn[i] = min(mn[i << 1], mn[i << 1 | 1])  def update\_max(pos, val):  i = size + pos - 1  mx[i] = val  i >>= 1  while i:  mx[i] = max(mx[i << 1], mx[i << 1 | 1])  i >>= 1  def update\_min(pos, val):  i = size + pos - 1  mn[i] = val  i >>= 1  while i:  mn[i] = min(mn[i << 1], mn[i << 1 | 1])  i >>= 1  def query\_max(l, r):  res = -INF  l += size - 1  r += size - 1  while l <= r:  if l & 1:  res = max(res, mx[l])  l += 1  if not (r & 1):  res = max(res, mx[r])  r -= 1  l >>= 1  r >>= 1  return res  def query\_min(l, r):  res = INF  l += size - 1  r += size - 1  while l <= r:  if l & 1:  res = min(res, mn[l])  l += 1  if not (r & 1):  res = min(res, mn[r])  r -= 1  l >>= 1  r >>= 1  return res  def solve\_sub4():  res = 0  if m > 2:  # case 1: D max in A  l = 1  for i in range(1, n + 1):  while l <= n and b[l] <= a[i]:  l += 1  update\_max(i, min(m - 1, (n - l + 1)) + min(m - 1, i))  for i in range(2, n + 1):  if query\_max(i, min(n, i + m - 3)) >= m:  res = max(res, abs(a[i] - a[i - 1]))   # reset  mx[:] = [-INF] \* (2 \* size)   l = 1  for i in range(1, n + 1):  while l <= n and a[l] < b[i]:  l += 1  update\_max(i, min(m - 1, n - i + 1) + min(m - 1, l - 1))  for i in range(2, n + 1):  if query\_max(max(1, i - m + 3), i - 1) >= m:  res = max(res, abs(b[i] - b[i - 1]))   # case 3: D max at ends  for i in range(1, n + 1):  update\_max(i, b[i])  update\_min(i, b[i])   l = 1  for i in range(1, n + 1):  length = min(i, m - 1)  while l <= n and b[l] <= a[i]:  l += 1  right = min(n, n - m + 1 + length)  if n - m + 1 + length >= 1 and l <= right:  res = max(res,  abs(query\_max(l, right) - a[i]),  abs(query\_min(l, right) - a[i]))  return res  print(solve\_sub4()) |

Rất vui khi anh em nào đã đọc đến những dòng cuối cùng này. Anh em đọc chắc cũng mỏi mắt luôn chứ 😅 vậy anh em nghĩ người viết ra hết đống ở trên còn thế nào nữa 🤪 . Vậy nên đừng tiếc gì cho mình một like nha để động viên mình nè . Chúc anh em học tốt và gặt hái nhiều thành công 😇