TRẠI HÈ HÙNG VƯƠNG
LẦN THỨ XVI – ĐIỆN BIÊN 2022

KỲ THI CHỌN HỌC SINH GIỎI
MÔN: TIN HỌC - KHỐI: 11
Ngày thi: 12 tháng 8 năm 2022
Thời gian làm bài: 180 phút (không kể thời gian giao đề)

HƯỚNG DẪN GIẢI

Bài 1. Đối xứng đôi

Subtask 1 (30%): $1 \le n \le 500$

Do xâu đối xứng đôi được ghép từ hai xâu đối xứng có độ dài bằng nhau nên xâu đối xứng đôi có độ dài chẵn. Vì vậy, ta xét mọi xâu con của s có độ dài chẵn và kiểm tra xem hai nửa của xâu con đó có đối xứng hay không.

Độ phức tạp thuật toán $O(n^3)$.

Subtask 2 (30%): $1 \le n \le 5000$

Ta có thể kiểm tra xâu con có đối xứng hay không với độ phức tạp thời gian $O(1)$ bằng cách sử dụng hàm băm hoặc thuật toán Manacher.

Độ phức tạp thuật toán $O(n^2)$.

Subtask 3 (40%): Không có thêm ràng buộc nào

Do xâu đối xứng đôi được ghép từ hai xâu đối xứng có độ dài bằng nhau nên hai nửa của nó là hai xâu đối xứng độ dài cùng chẵn hoặc cùng lẻ.

Trường hợp 1: Xâu đối xứng đôi được tạo từ hai xâu đối xứng có độ dài lẻ bằng nhau

Gọi $d_1[i]$ là số xâu con đối xứng độ dài lẻ với vị trí trung tâm là i. Chú ý rằng vị trí trung tâm của xâu con $s[l..r]$ là $\left\lfloor \frac{l+r}{2} \right\rfloor$. Khi đó mọi xâu con tâm i, bán kính R $(0 \le R < d_1[i])$ đều đối xứng, hay nói cách khác các xâu con $s[i-R..i+R]$ là đối xứng. Còn các xâu con tâm i, bán kính $R \ge d_1[i]$ là không đối xứng.

Chúng ta có thể tính $d_1[i]$ bằng hàm băm trong thời gian $O(n.log_2 n)$ hoặc thuật toán Manacher trong thời gian $O(n)$.

Bây giờ ta xem khi nào hai xâu đối xứng độ dài lẻ tâm i và j sẽ tạo ra một xâu đối xứng đôi:

0	**1**	2	3	**4**	5
a	**b**	a	c	**a**	c

Ta nhận thấy rằng điều kiện cần và đủ là khoảng cách giữa hai tâm $|i-j|$ phải là lẻ và

$$\frac{|i-j|+1}{2} \le min(d_1[i], d_1[j])$$

Dựa vào điều kiện trên, ta đếm số cặp hai xâu đối xứng độ dài lẻ tạo thành xâu đối xứng đôi như sau.

Đầu tiên tạo mảng seg_odd chứa các xâu đối xứng độ dài lẻ $(i, d_1[i])$ theo thứ tự giảm dần của $d_1[i]$. Gọi thành phần thứ nhất, thứ hai của mỗi phần tử của mảng là $center$ và $radius$.

Sau đó duyệt lần lượt từng phần tử trong mảng seg_odd, với mỗi phần tử thứ j ta sẽ đếm số phần tử thứ i trước đó $(i < j)$ mà chúng tạo thành xâu đối xứng đôi. Phần tử thứ i cần thỏa mãn các điều kiện sau:

- $|seg\_odd[i].center - seg\_odd[j].center|$ là số lẻ, hay nói cách khác $seg\_odd[i].center$ và $seg\_odd[j].center$ khác tính chẵn lẻ;
- $\frac{|seg\_odd[i].center - seg\_odd[j].center|+1}{2} \le min(seg\_odd[i].radius, seg\_odd[j].radius)$

  $\Leftrightarrow \frac{|seg\_odd[i].center - seg\_odd[j].center| + 1}{2} \le seg\_odd[j].radius$

  $\Leftrightarrow |seg\_odd[i].center - seg\_odd[j].center| \le 2 \times seg\_odd[j].radius - 1$

  $\Leftrightarrow -2 \times seg\_odd[j].radius + 1 \le seg\_odd[i].center - seg\_odd[j].center \le 2 \times seg\_odd[j].radius - 1$

  $\Leftrightarrow seg\_odd[j].center - 2 \times seg\_odd[j].radius + 1 \le seg\_odd[i].center \le seg\_odd[j].center + 2 \times seg\_odd[j].radius - 1$

Tóm lại là ta cần đếm số phần tử thứ i $(i < j)$ sao cho $seg\_odd[i].center$ và $seg\_odd[j].center$ khác tính chẵn lẻ và $seg\_odd[i].center$ nằm trong đoạn từ $seg\_odd[j].center - 2 \times seg\_odd[j].radius + 1$ đến $seg\_odd[j].center + 2 \times seg\_odd[j].radius - 1$.

Để xử lý yêu cầu trên hiệu quả, ta dùng hai cấu trúc BIT (Binary Indexed Tree): $odd[0], odd[1]$ để lưu trữ các phần tử với tâm ở vị trí chẵn và lẻ (có thể dùng cấu trúc Segment Tree).

Trường hợp 2: Xâu đối xứng đôi được tạo từ hai xâu đối xứng có độ dài chẵn bằng nhau

Trường hợp này làm tương tự như trường hợp trước, cụ thể như sau.

Gọi $d_2[i]$ là số xâu con đối xứng độ dài chẵn với vị trí trung tâm là i. Chú ý rằng vị trí trung tâm của xâu con $s[l..r]$ là $\left\lceil \frac{l+r}{2} \right\rceil$. Khi đó mọi xâu con tâm i, bán kính R $(0 \le R < d_2[i])$ đều đối xứng, hay nói cách khác các xâu con $s[i-1-r..i+r]$ là đối xứng. Còn các xâu con tâm i, bán kính $R \ge d_2[i]$ là không đối xứng.

Chúng ta có thể tính $d_2[i]$ bằng hàm băm trong thời gian $O(n.log_2 n)$ hoặc thuật toán Manacher trong thời gian $O(n)$.

Bây giờ ta xem khi nào hai xâu đối xứng độ dài chẵn tâm i và j sẽ tạo ra một xâu đối xứng đôi:

0	1	**2**	3	4	5	**6**	7
a	b	**b**	a	x	y	**y**	x

Ta nhận thấy rằng điều kiện cần và đủ là khoảng cách giữa hai tâm $|i-j|$ phải là chẵn và

$$\frac{|i-j|}{2} \le min(d_2[i], d_2[j])$$

Dựa vào điều kiện trên, ta đếm số cặp hai xâu đối xứng độ dài chẵn tạo thành xâu đối xứng đôi như sau.

Đầu tiên tạo mảng seg_even chứa các xâu đối xứng độ dài chẵn $(i, d_2[i])$ theo thứ tự giảm dần của $d_2[i]$. Gọi thành phần thứ nhất, thứ hai của mỗi phần tử của mảng là $center$ và $radius$.

Sau đó duyệt lần lượt từng phần tử trong mảng seg_even, với mỗi phần tử thứ j ta sẽ đếm số phần tử thứ i trước đó $(i < j)$ mà chúng tạo thành xâu đối xứng đôi. Phần tử thứ i cần thỏa mãn các điều kiện sau:

- $|seg\_even[i].center - seg\_even[j].center|$ là số chẵn, hay nói cách khác $seg\_even[i].center$ và $seg\_even[j].center$ cùng tính chẵn lẻ;
- $\frac{|seg\_even[i].center - seg\_even[j].center|}{2} \leq min(seg\_even[i].radius, seg\_even[j].radius)$

  $\Leftrightarrow \frac{|seg\_even[i].center - seg\_even[j].center|}{2} \leq seg\_even[j].radius$

  $\Leftrightarrow -2 \times seg\_even[j].radius \leq seg\_even[i].center - seg\_even[j].center \leq 2 \times seg\_even[j].radius$

  $\Leftrightarrow seg\_even[j].center - 2 \times seg\_even[j].radius \leq seg\_even[i].center \leq seg\_even[j].center + 2 \times seg\_odd[j].radius$

Tóm lại là ta cần đếm số phần tử thứ i $(i < j)$ sao cho $seg\_even[i].center$ và $seg\_even[j].center$ cùng tính chẵn lẻ và $seg\_odd[i].center$ nằm trong đoạn từ $seg\_odd[j].center - 2 \times seg\_odd[j].radius$ đến $seg\_odd[j].center + 2 \times seg\_odd[j].radius$.

Để xử lý yêu cầu trên hiệu quả, ta dùng hai cấu trúc BIT (Binary Indexed Tree): $even[0], even[1]$ để lưu trữ các phần tử với tâm ở vị trí chẵn và lẻ (có thể dùng cấu trúc Segment Tree).

Độ phức tạp thuật toán là $O(n.log_2 n)$.

Bài 2. Tham quan Điện Biên Phủ

Ta xây dựng mô hình đồ thị cho bài toán. Địa điểm i tương ứng với đỉnh i. Con đường hai chiều nối hai địa điểm u_i, v_i với thời gian di chuyển giữa chúng là w_i sẽ tương ứng với cạnh vô hướng nối hai đỉnh u_i, v_i với trọng số w_i. Bài toán trở thành tìm đường đi bắt đầu từ đỉnh 1 sao cho đi qua tất cả k đỉnh $p_1, p_2, \dots, p_k$ (theo thứ tự tùy ý) với độ dài của đường đi là nhỏ nhất. Chú ý rằng khi đến một đỉnh p_i, ta có thể di chuyển tức thời đến một đỉnh p_j nào đó đã thăm trước đó.

Subtask 1 (10%): $k = 1$

Trong subtask 1 ta có $k = 1$, tức là chỉ có duy nhất một địa điểm hấp dẫn. Vì vậy việc di chuyển tức thời trong subtask này không có ý nghĩa.

Nhiệm vụ của subtask 1 đơn giản là tìm độ dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh p_1. Ta có thể sử dụng thuật toán Dijkstra để tìm câu trả lời.

Độ phức tạp thuật toán là $O(n.log_2 n)$.

Subtask 2 (10%): $k = n$

Trong subtask 2 ta có $k = n$, tức là mọi địa điểm đều hấp dẫn. Dễ dàng nhận thấy rằng câu trả lời sẽ là trọng số của cây khung nhỏ nhất của đồ thị. Chúng ta có thể tính trọng số của cây khung nhỏ nhất bằng thuật toán Kruskal.

Độ phức tạp thuật toán là $O(m.log_2 n)$.

Subtask 3 (20%): $1 \leq n \leq 10; 0 \leq m \leq 10; 1 \leq w_i \leq 10^2; 1 < k < n$

Trong subtask 3 kích thước dữ liệu khá nhỏ, nên ta duyệt mọi hoán vị đi thăm lần lượt các địa điểm hấp dẫn. Khi đi từ địa điểm hấp dẫn này đến địa điểm hấp dẫn tiếp theo, ta chọn đường đi ngắn nhất. Sau khi thăm một địa điểm hấp dẫn, ta bổ sung hoặc gán lại trọng số của các cạnh từ địa điểm hấp dẫn này đến các địa điểm hấp dẫn đã thăm bằng 0. Trong quá trình duyệt, ta lưu lại đường đi có độ dài nhỏ nhất.

Độ phức tạp thuật toán là $O(k!.n.log_2 n)$.

Subtask 4 (20%): $1 \le n \le 10^2; 0 \le m \le 10^2; 1 \le w_i \le 10^5; 1 < k < n$

Ta xây dựng một đồ thị mới gồm k đỉnh là các đỉnh hấp dẫn của đồ thị ban đầu. Giữa hai đỉnh có cạnh (vô hướng) nối với trọng số là độ dài đường đi ngắn nhất giữa chúng. Dùng thuật toán Floyd-Warshall để tính toán độ dài đường đi ngắn nhất giữa mọi cặp đỉnh hấp dẫn.

Áp dụng thuật toán Kruskal để tìm cây khung nhỏ nhất trong đồ mới. Khi đó câu trả lời của bài toán là trọng số của cây khung này cộng với độ dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh hấp dẫn gần nó nhất.

Độ phức tạp thuật toán là $O(n^3)$.

Subtask 5 (20%): $1 \le n \le 10^3; 0 \le m \le 10^3; 1 < k < n$

Ta cải tiến thuật toán trong subtask 4 trong việc tính độ dài đường đi ngắn nhất giữa các đỉnh hấp dẫn, bằng cách áp dụng thuật toán Dijkstra lần lượt cho các đỉnh xuất phát là đỉnh hấp dẫn.

Độ phức tạp thuật toán là $O(k.n.log_2 n)$.

Subtask 6 (20%): Không có thêm ràng buộc nào

Đầu tiên với mỗi đỉnh i, ta tính $d[i]$ gồm hai thông tin: khoảng cách từ đỉnh i đến đỉnh hấp dẫn (ứng với địa điểm hấp dẫn) gần nó nhất và đỉnh hấp dẫn gần nó nhất. Để thực hiện công việc này, ta sẽ tạo thêm một đỉnh giả 0 và nối đỉnh này với các đỉnh hấp dẫn bởi các cạnh có trọng số bằng 0. Sau đó áp dụng thuật toán Dijkstra từ đỉnh 0. Thực tế khi cài đặt, ta không cần sinh ra đỉnh giả 0, chỉ cần sửa đổi trong bước khởi tạo của thuật toán Dijkstra là gán nhãn tất cả các đỉnh hấp dẫn $d\big[p[i]\big] = \{0, p[i]\}$ với $i = 1, 2, \dots, k$ và đẩy chúng vào hàng đợi ưu tiên.

Tiếp theo ta sẽ xây dựng đồ thị mới chỉ bao gồm các đỉnh hấp dẫn. Với mỗi cạnh (u_i, v_i, w_i) trong đồ thị cũ, gọi $x = d[u_i].second$ và $y = d[v_i].second$ tương ứng là đỉnh hấp dẫn gần đỉnh u_i và v_i. Nếu $x \ne y$ thì ta thêm một cạnh vô hướng vào đồ thị mới nối hai đỉnh x và y với trọng số là $d[u_i].first + w_i + d[v_i].first$.

Sau đó ta tính trọng số của cây khung nhỏ nhất trong đồ thị mới này bằng thuật toán Kruskal. Cuối cùng câu trả lời bài toán sẽ là $d[1].first$ cộng với trọng số của cây khung nhỏ nhất tìm được ở trên.

Độ phức tạp thuật toán là $O\big((n+m).log_2 n\big)$.

Bài 3. Điện toán đám mây

Subtask 1 (4%): $k = 1$

Câu trả lời đơn giản là $n \times max(a_1, a_2, \dots, a_n)$.

Độ phức tạp $O(n)$.

Subtask 2 (6%): Số các giá trị khác nhau của $a_1, a_2, \dots, a_n$ nhỏ hơn hoặc bằng k

Chi phí của mỗi tác vụ bằng số chương trình nhân với giá trị lớn nhất của bộ nhớ các chương trình trong tác vụ. Do đó nếu một tác vụ gồm các chương trình có cùng dung lượng bộ nhớ thì việc chúng ở trong cùng một tác vụ hay chia chúng ra thành nhiều tác vụ đều cho cùng một chi phí. Mặt khác, sẽ tốt hơn nếu mỗi tác vụ chỉ gồm một chương trình.

Do giới hạn của subtask 2 là số các giá trị khác nhau của $a_1, a_2, \dots, a_n$ nhỏ hơn hoặc bằng k, nên từ nhận xét trên ta sẽ có thuật toán tham lam như sau. Ta chia n chương trình thành k tác vụ, sao cho các chương trình trong cùng một tác vụ sẽ có dung lượng bộ nhớ bằng nhau. Khi đó câu trả lời bài toán là $a_1 + a_2 + \dots + a_n$.

Độ phức tạp $O(n)$.

Subtask 3 (10%): $n \leq 10$

Để giảm chi phí cho mỗi tác vụ, ta cần nhóm các chương trình theo thứ tự dung lượng bộ nhớ giảm dần vào mỗi tác vụ. Do vậy đầu tiên ta sắp xếp các a_i giảm dần và mỗi tác vụ sẽ gồm các đoạn chương trình liên tiếp. Từ subtaks này trở đi, ta sẽ giả thiết các a_i đã được sắp xếp giảm dần.

Bây giờ ta sẽ chia n chương trình theo thứ tự giảm dần của bộ nhớ, vào k tác vụ. Việc này giống như ta đặt $k-1$ vách ngăn vào $n-1$ vị trí giữa hai phần tử liên tiếp của mảng $a_1, a_2, \dots, a_n$, giữa hai vị trí không được đặt quá một vách ngăn. Số cách sẽ là C_{n-1}^{k-1}. Công việc này chính là bài toán chia kẹo của Euler.

Ta sẽ duyệt tất cả các cách chia n chương trình vào k tác vụ. Mỗi cách tính số chi phí phải trả và lưu lại chi phí nhỏ nhất trong số các cách.

Độ phức tạp $O\left(n.log_2 n + n.C_{n-1}^{k-1}\right)$.

Subtask 4 (20%): $n \leq 10^2$

Sắp xếp các a_i theo thứ tự giảm dần.

Gọi $dp[i][j]$ là chi phí tối thiểu để chia i chương trình thành không quá j tác vụ. Chú ý rằng số tác vụ càng ít thì chi phí càng cao, nên $dp[i][j]$ cần phải sử dụng nhiều tác vụ nhất có thể. Vì vậy câu trả lời của bài toán là $dp[n][k]$ với $k \leq n$, sẽ chắc chắn phải sử dụng k tác vụ.

Ta có công thức truy hồi tính $dp[i][j]$ như sau:

- $dp[i][1] = i \times a_1$ với mọi $i = 1, 2, \dots, n$;
- $dp[i][j] = \min\limits_{l=1,2,\dots,i}(dp[l][j-1] + (i-l) \times a_{l+1})$ với $i = 1, 2, \dots, n$ và $j = 2, 3, \dots, k$.

Độ phức tạp $O(n^2.k)$.

Subtask 5 (20%): $n \leq 10^3$

Nhận thấy rằng với i cố định, $dp[i][j]$ sẽ giảm dần khi j tăng. Vì vậy ta có thể loại bỏ tham số j và sử dụng tìm kiếm nhị phân để tính $dp[i]$ là chi phí tối thiểu của i chương trình với tối đa k tác vụ. Kỹ thuật xử lý này được gọi là "Aliens trick" (vì nó được biết đến qua bài toán "Aliens" của IOI 2016). Kỹ thuật này cũng còn có các tên gọi khác như: "tối ưu lambda (lambda optimization)" hay "nhân tử Lagrange (Lagrange multipliers)".

Trong kỹ thuật "Alien trick", với mỗi tác vụ ta sẽ cộng thêm vào một chi phí λ. Việc tăng/giảm λ sẽ làm giảm/tăng hoặc giữ nguyên số tác vụ trong một giải pháp tối ưu. Điều đó cho thấy rằng có thể tìm kiếm nhị phân giá trị nhỏ nhất của λ để tạo ra một giải pháp tối ưu có không nhiều hơn k tác vụ. Gọi $dp_\lambda[i]$ là chi phí nhỏ nhất của i chương trình với tối đa k tác vụ trên nó, trong đó mỗi tác vụ sẽ được cộng thêm một chi phí λ. Ta có:

$$dp_\lambda[i] = \min_{j=0,1,2,\dots,i-1}\left(dp_\lambda[j] + (i-j) \times a_{j+1} + \lambda\right)$$

Ngoài ra chúng ta sẽ lưu trữ một mảng bổ trợ khác $cnt_\lambda[i]$ là số tác vụ tối thiểu mà $dp_\lambda[i]$ sử dụng.

Cuối cùng chú ý rằng chi phí cho n chương trình cần được điều chỉnh, bởi vì các tác vụ nhiều hơn hoặc ít hơn có thể sẽ cho cùng một giá trị λ.

Độ phức tạp $O\left(n^2.log_2(n.max\{a_i\})\right)$.

Subtask 6 (20%): $n \leq 10^4$ và $k \leq 10^3$

Ta tối ưu thuật toán subtask 4 theo một hướng khác. Ta biến đổi:

$$dp[i][j] = \min_{l=1,2,\dots,i}(dp[l][j-1] + (i-l) \times a_{l+1}) = \min_{l=1,2,\dots,i}(a_{l+1} \times i + dp[l][j-1] - a_{l+1} \times l)$$

Ta có thể tính nhanh $\min_{l=1,2,\dots,i}(a_{l+1} \times i + dp[l][j-1] - a_{l+1} \times l)$ bằng kỹ thuật bao lồi (convex hull trick) trên tập các đường thẳng $y = A.x + B$ với hệ số góc $A = a_{l+1}$, hệ số tự do $B = dp[l][j-1] - a_{l+1} \times l$ và các truy vấn tại $x = i$.

Theo giả thiết các hệ số góc a_{l+1} có thứ tự giảm dần nên các đường thẳng đã được sắp xếp theo hệ số góc giảm dần. Vì trong kĩ thuật bao lồi, các đường thẳng đưa vào bao lồi cần theo thứ tự hệ số góc giảm dần nên ta không cần phải sắp xếp lại các đường thẳng này nữa. Hơn nữa cần chú ý rằng có thể có hai đường thẳng cùng hệ số góc, nên ta cần phải xử lý trường hợp này.

Mặt khác do các truy vấn tại $x = i$ theo thứ tự i tăng dần ($i = 1, 2, \dots, n$), nên thay vì tìm kiếm nhị phân cho từng truy vấn, ta chỉ cần tìm kiếm tuần tự một lần để trả lời cho tất cả các truy vấn.

Độ phức tạp là $O(k.n)$.

Subtask 7 (20%): Không có thêm ràng buộc nào

Sử dụng cả hai kỹ thuật tối ưu đã áp dụng trong subtask 5 và subtask 6.

Ta tiếp tục tối ưu thuật toán trong subtaks 5 như sau. Biến đổi:

$$dp_\lambda[i] = \min_{j=0,1,2,\dots,i-1}\left(dp_\lambda[j] + (i-j) \times a_{j+1} + \lambda\right) = \min_{j=0,1,2,\dots,i-1}\left(a_{j+1} \times i + dp_\lambda[j] - j \times a_{j+1} + \lambda\right)$$

Để tính nhanh $\min_{j=0,1,2,\dots,i-1}\left(a_{j+1} \times i + dp_\lambda[j] - j \times a_{j+1} + \lambda\right)$, ta sử dụng kỹ thuật bao lồi (convex hull trick) trên tập các đường thẳng $y = A.x + B$ với hệ số góc $A = a_{j+1}$, hệ số tự do $B = dp_\lambda[j] - j \times a_{j+1} + \lambda$ và các truy vấn tại $x = i$.

Độ phức tạp $O\left(n.log_2(n.max\{a_i\})\right)$.

---------- HẾT ----------