ĐỒ THỊ 17/10

BGRAPTH

Xây dựng các thành phần liên thông, mỗi thành phần liên thông (TPLT) là một đồ thị hai phía trong đó một phía chỉ bao gồm đỉnh chẵn, phía còn lại chỉ bao gồm đỉnh lẻ

Gọi a_i, b_i lần lượt là số đỉnh của phía đỉnh chẵn và phía đỉnh lẻ của TPLT thứ i

Kết quả cho TPLT thứ i là $2^{a_i} + 2^{b_i}$

Kết quả bài toán là $\prod 2^{a_i} + 2^{b_i}$

CSP

Subtask 1 cần tính tổng quãng đường di chuyển ứng với một phương án phân phối xe. Việc giải quyết subtask 1 với kích thước n lớn còn gợi ý dùng phương pháp 2 con trỏ, kỹ thuật sẽ được nâng cấp thành Knuth optimization.

Subtask 2 gợi ý rằng bài toán này có thể giải bằng quy hoạch động:

Để tiện cho việc xử lý vòng tròn, ta tăng số bến lên gấp đôi, bến n + 1 trùng với bến 1, bến n+ 2 trùng với bến 2, …, bến 2n trùng với bến n . Khi đó mảng A được nhân đôi kích thước: $a_1, a_2, ,.., a_n, a_{n+1}, a_{n+2}, ,.., a_{2n}$

Bây giờ một đoạn liên tiếp gồm đúng n bến tương đương với bến n vòng quanh hồ theo chiều đánh số, vì vậy ta chỉ cho phép các học sinh di chuyển từ bến chỉ số nhỏ hơn tới bến chỉ số lớn hơn.

Chi phí để đưa các học sinh ở bến t về bến j ($t \le j$) bằng:

$$a_t \times (j-t)$$

Xét các bến từ i đến j ($I \le j$), chi phí để dồn các học sinh trong phạm vi đó về bến j được tính bằng:

$$\text{Cost}(i, j) = \sum_{t=i}^{j} (a_t \times (j-t))$$

Gọi f(p,i ,j) là chi phí nhỏ nhất (tính bằng tổng độ dài quãng đường di chuyển của các học sinh) nếu đón bởi p xe. Lưu ý là nếu số xe dư thừa thì một vài xe có thể đón ở cùng bến, trong trường hợp một xe số hiệu nhỏ nhất tại bến đón học sinh, các xe khác cùng bến coi như vô dụng.

Giá trị f(k, i, i + n − 1) chính là chi phí ít nhất để đón tất cả các học sinh bởi k xe trong đó xe cuối cùng đón ở bến i + n − 1. Đáp số là:

$$\min\{ f(k, i, i+n-1)\}, \forall i : 1 \le i \le n$$

Cách tính f(p ,i ,j) như sau:

- Nếu p = 1, thì **f(1, i, j) = Cost(i, j)** .
- Nếu p > 1 thì xe thứ p − 1 sẽ phải đón ở bến t nào đó thuộc [i …j], tất cả các học sinh từ bến t + 1 tới j sẽ phải di chuyển tới bến j để lên xe thứ p , vậy:

$$f(p, i, j) = \min\{ f(p-1, i, t) + \text{Cost}(t+1, j)\}, \forall t : i \le t \le j$$

Subtask 3: Xây dựng ma trận chi phí M, ta sử dụng thuật toán tính toán trên ma trận.

Khởi tạo ma trận M trong đó M[i][j] = f(1, i, j) = Cost(i, j).

Sau đó tính ma trận:

$$M = M^k \quad (\text{k ma trận M})$$

Phần tử M[i][j] lúc này chính là f (k, i, j).

REBUILD

Cho hai cây đồ thị T_S và T_T, bài toán yêu cầu thực hiện chuyển từ cây này sang một cây kia bằng cách di chuyển ít cạnh nhất mà vẫn duy trì sự liên thông của cây (vẫn là cây) tại mỗi bước thực hiện.

Đầu tiên ta nhận xét rằng số cạnh ít nhất cần di chuyển là số cạnh khác nhau giữa hai cây. Gọi S là tập các cạnh có trong cây T_S mà không có trong cây T_T, T là tập các cạnh có trong cây T_T mà không có trong cây T_S và C là tập cạnh chung của hai cây khi đó ta có $|S| = |T = (N-1) - |C|$. Giả thiết là ta cần di chuyển số cạnh ít nhất trong S.

Gọi T_c là cây thu được trong quá trình xây dựng, lúc đầu T_c bằng cây thứ nhất, với cạnh $e \in S$, sau khi bỏ cạnh e thì cây T_c sẽ chia thành hai cây con T_1, T_2. Trong T sẽ tồn tại một vài cạnh e_2 nối hay cây T_1, T_2 thành một cây. Thay e bởi e_2 ta giảm đi được một cạnh trong S.

Khi này bài toán đưa về việc tìm e_2 sao cho hiệu quả?

Thuật toán 1: Duyệt mọi cách chọn $e_2 \in T$, thay dần các cạnh $e \in S$ của cây thứ nhất. Bằng cách chọn lần lượt $e \in S$, với mỗi e ta duyệt lần lượt $e_2 \in T$. Thay e bởi e_2 và kiểm tra tính liên thông của cây T_c. Nếu T_c liên thông thì ta bỏ e khỏi S và e_2 khỏi T và dừng (S giảm đi 1), ngược lại ta duyệt qua e_2 mới.

Thuật toán thực hiện việc bỏ, dựng cây và kiểm tra tính liên thông nhiều lần nên dùng cấu trúc union-find hiệu quả hơn khi cài đặt.

Thuật toán duyệt qua S và T, mỗi lần kiểm tra tính liên thông mất $O(N)$. Nên độ phức tạp của thuật toán $O(N^3)$.

Thuật toán 2: Bỏ lần lượt các cạnh $e \in S$ trên cây T_c, mỗi lần bỏ sẽ chia cây thu được thành hai cây, $DFS()$ từ một đỉnh bất kỳ T_c ta thu được tập đỉnh V_1 của cây T_1 và V_2 của cây T_2 và tìm $e_2 \in T$ sao cho $e_2 = (u, v)$ có $u \in V_1$ và $v \in V_2$. Thêm e_2 vào cây T_c. Cây T_c đã thay thêm một cạnh.

Duyệt $e \in S$ mất $O(N)$, $DFS()$ mất $O(N)$, việc tìm e_2 chỉ là $O(1)$ nên thuật toán có độ phức tạp là $O(N^2)$.

Thuật toán 3: Thay đổi cách chọn e và e_2, nếu ta chọn e nối một nút lá của cây T_S, và chọn e_2 tương ứng với nút lá đó của cây T_T thì ta không cần phải kiểm tra tính liên thông của cây T_c. Để thực hiện ta dùng một danh sách lưu các cạnh nối đến mỗi $u \in T_S$ và mỗi $v \in T_T$ và dùng một hàng đợi để lưu các nút sẽ trở thành nút lá trong quá trình thay.

Để ý các nút được nối với nhau trong C thì cũng sẽ nối với nhau sau khi chuyển các cạnh nên ta xem các cạnh này sẽ tạo ra các cây (nối thêm các cạnh trong S thì ta được cây T_S). Trong trường hợp xấu nhất thì cây này cũng có ít nhất một nút lá và ta có thể áp dụng cách làm trên.

Để thực hiện ta dùng cấu trúc union – find để ghép hết các cạnh trong C, sau đó ghép dần các cạnh (đi ra) từ danh sách đã lưu.

Có tối đa N nút lá, mỗi thao tác trong union – find mất $O(logN)$ nên độ phức tạp của thuật toán là $O(NlogN)$.

CEZAR

Tìm cách tính tất cả các tổng khoảng cách nếu chọn mỗi đỉnh làm gốc. Trước hết chọn đỉnh 1 làm gốc, từ đó tính được các đỉnh con của đỉnh 1, và cứ như thế...

Dùng cấu trúc dặc biệt BIT để lưu trữ trọng số các cạnh rồi sử dụng chặt nhị phân để tính tổng của (N-1-k) cạnh có trọng số nhỏ nhất. ĐPT: O(N. logN.logN)

YENOM

Thuật toán 1: Với ràng buộc $T \leq 2 \times 10^4$.

Gọi $F[x]$ = true nếu có cách thanh toán số tiền x.

$F[0] = true$

$F[x] = OR_{i=1}^{n} F[x - a_i], x > 0.$

Thuật toán 2: Với ràng buộc $T \leq 2 \times 10^5$.

Lưu mảng F bằng một bitset.

$F[0] = 1$

Xét $i = 1..n, F = F|(F \ll a_i)$.

Thuật toán 3: Với ràng buộc $T \leq 10^{18}$.

Sử dụng thuật toán Dijkstra.

Giả sử $x\%a_1 = r$, khi đó ta sẽ đếm phân hoạch theo r (tức là xét từng r và đếm số nghiệm x của bài toán mà $x\%a_1 = r$).

Dựng đồ thị với a_1 đỉnh $0,1,\ldots,a_1 - 1$. Từ đỉnh k sẽ có n cung, cung thứ i có trọng số a_i, nối k với $(k + a_i)\%a_1$.

Tìm đường đi ngắn nhất từ 0 đến tất cả các đỉnh khác.

- Nếu $d_r > T$ thì không có nghiệm nào ứng với trường hợp $x\%a_1 = r$.

- Nếu $d_r \leq T$ thì tất cả các nghiệm x ứng với trường hợp $x\%a_1 = r$ là: $d_r, d_r + a_1, d_r + 2a_1$,...cho đến khi vượt quá T. Có thể hiểu d_r là số nhỏ nhất tạo đượng từ tập a mà đồng dư với r theo mod a_1.

Độ phức tạp thuật toán là: $O(a_1^2)$