통신망 2 – 풀이

작성자: 구재현

서브태스크 1

제한이 아주 작기 때문에 다양한 풀이가 있습니다. 모든 $0 \le t \le T$ 에 대해서, t초의 통신망의 연결 컴포넌트를 DFS (Flood-Fill) 로 구합시다. x와 y가 t초에 연결되어 있다는 것은 t초의 통신망 상태에서 x와 y가 같은 컴포넌트에 있다는 것입니다. $comp[t][v]$ 를, t초에 정점 v가 있는 연결 컴포넌트의 번호라고 정의하면, 이는 $comp[t][x] = comp[t][y]$ 라는 것과 동치입니다. 모든 쿼리에 대해 y 를 고정하고, 위 조건을 모든 시간 $l, l+1, \ldots, r$ 에 대해 확인할 경우 $O(TS + QNT)$ 시간에 문제를 해결할 수 있습니다.

서브태스크 2

서브태스크 1과 동일하게 진행하나, 쿼리 처리 부분을 최적화합니다. y 를 고정했을 경우, 우리가 알고 싶은 것은 모든 $l \le i \le r$ 에 대해 $comp[x][i] = comp[y][i]$ 인지의 여부입니다. 이는 두 문자열의 부분문자열이 일치하는지를 판별하는 문제와 동일합니다. $O(TN)$ 시간에 각 문자열의 해시를 전처리해두면, 각 y 에 대해 이를 $O(1)$ 시간에 판별할 수 있습니다. 고로 전체 문제를 $O(TS + TN + QN)$ 시간에 해결할 수 있습니다.

서브태스크 3

오프라인 동적 연결성 알고리즘을 사용하여 전체 문제를 $O(N + S \log S \log N + Q \log N)$ 시간에 해결할 수 있습니다. 이 알고리즘에 대해서는 여러 자료에서 잘 설명되어 있으나, 풀이의 완결성을 위해 이 글에서 다시 설명합니다.

입력으로 주어진 간선 갱신을, "시간 구간 $[l, r]$ 에 간선 (u, v) 추가" 의 형태로 변환합시다. 시간 축에 대해 세그먼트 트리를 만들면, 위에서 설명한 간선 갱신은 $O(\log n)$ 개의 노드에 간선 (u, v) 을 추가하는 것으로 해석할 수 있습니다. 각 시간의 연결 상태는, 해당 시간 위치에서 루트로 가는 경로에 있는 모든 노드의 간선 합집합을 그래프로 두었을 때, 연결 컴포넌트의 상태와 동일합니다.

일반적인 Union Find를 약간 수정해서 다음과 같은 기능을 지원하게끔 구현할 수 있습니다.

- Worst-case $O(\log n)$ 시간에 두 집합 union.
- $O(\log n)$ 시간에 집합 size 계산.
- $O(\log n)$ 시간에 마지막으로 진행한 연산 Undo.

구체적으로, path compression을 하지 않고 대신 union-by-size로 시간 복잡도를 보장해 주면 됩니다.

위에서 구성한 세그먼트 트리를 top-down으로 순회합니다. 구체적으로, DFS 함수가 특정 노드를 방문하면, 해당 노드에 있는 모든 간선을 Union-find에 추가하고, 재귀적으로 자식을 방문한 후, 노드를 떠날 때 해당 노드에서 합쳐준 간선들을 모두 undo해줍니다. 이렇게 할 경우 다음 두 invariant가 보장됩니다:

- 특정 노드를 방문하고 간선을 다 추가한 시점에서, Union-find에 있는 간선들은 루트에서 현재 노드로 가는 경로 상에 있는 간선들의 합집합과 동일.

- 특정 노드를 떠나기 전 간선을 제거하지 않은 시점에서, Union-find에 있는 간선들은 루트에서 현재 노드로 가는 경로 상에 있는 간선들의 합집합과 동일하며, 이 간선들이 추가된 순서는 루트에서 현재 노드로 내려가는 방향임 (즉 마지막으로 진행한 연산을 undo하면 현재 노드에서 추가한 간선들이 undo됨.)

이제 각 시간에 대한 그래프에 대한 Union-find가 있으니, 쿼리는 리프 노드에서 응답하면 됩니다.

서브태스크 4

서브태스크 3과 동일하게, 간선 갱신을 "시간 구간 $[l, r]$ 에 간선 $(u, u+1)$ 추가" 의 형태로 변환합시다. 가능한 간선의 후보가 $N-1$ 개이기 때문에, 위 표현을 뒤집어서 간선 추가 대신 "시간 구간 $[l, r]$에u$ 와 $u+1$ 사이에 간선이 없음" 이라는 형태로 그래프의 상태를 표현할 수도 있습니다. 이렇게 $[l, r]$ 시간 구간에 u 와 $u+1$ 이 연결되지 않았다는 정보를 튜플 (u, l, r) 의 형태로 표현합시다.

쿼리 (v, l', r') 이 주어졌을 때, 해당 쿼리에 대해 도달 가능한 정점은 구간을 이룹니다. 도달 가능한 정점 중 인덱스 최소를 x 라고 합시다. 각 튜플 (u, l, r) 를 2차원 평면 상에서 (u, l) 과 (u, r) 을 잇는 y 축에 평행한 선분으로 생각해 봅시다. $u < v$ 인 모든 선분 중 $[l', r']$ y 축 구간을 지나는 선분이 있다면, 이러한 선분 중 최대 $u+1$ 이 우리가 구하는 x 입니다. 그러한 선분이 없을 경우 $x = 0$ 입니다.

이를 계산하는 방법은 다음과 같습니다:

- $u < v, l \in [l', r']$ 인 (u, l, r) 중 최대 u 구하기는, l 에 대해서 최댓값 세그트리를 만들고 u 가 증가하는 순서대로 스위핑을 하면서 구할 수 있습니다.
- $u < v, r \in [l', r']$ 인 (u, l, r) 중 최대 u 구하기는, r 에 대해서 최댓값 세그트리를 만들고 u 가 증가하는 순서대로 스위핑을 하면서 구할 수 있습니다.
- 위 경우는 모두 $[l', r']$ y 축 구간을 지나는 올바른 선분들만 고려했지만, 만약에 선분이 $l < l' \le r' < r$ 을 만족한다면 해당 경우는 고려되지 않습니다. 이 경우는 선분을 y 축이 증가하는 순서대로 스위핑한 후, 각 쿼리에 대해서 (v, l') 에서 왼쪽으로 갔을 때 처음 닿는 선분을 구하는 식으로 계산할 수 있습니다. 정확히 저 조건만 만족하는 선분이 고려되지는 않지만, 저 조건을 만족하는 선분은 전부 고려되며 그 외에 고려된 선분들도 전부 가능한 후보라서 괜찮습니다.

최대 v 는 대칭적으로 해결하면 됩니다. 전체 문제가 $O((N+S)\log(N+S) + Q\log Q + Q\log N)$ 시간에 해결됩니다.

만점 풀이

$s(u)$ 를, 정점 u 가 속한 연결 컴포넌트의 번호를 시간 순으로 나타낸 길이 $T+1$ 의 문자열이라고 합시다. 또한, $s(u)[l \dots r]$ 을 이 문자열의 $l, l+1, \dots, r$ 번째 문자로 이루어진 부분문자열이라고 합시다 ($0 \le l \le r \le T$). 서브태스크 2에서는 $s(u)$ 를 직접 구한 후 적절한 전처리로 두 문자열 구간의 동일성을 비교했습니다. 만점 풀이에서도 유사한 방식의 풀이를 사용합니다.

서브태스크 3과 동일한 식으로 세그먼트 트리를 구성합시다. 세그먼트 트리의 노드 p 에 대해, 정점 v 가 **활성화** 되었다는 것은, p 와 p 의 서브트리에서 v 를 끝점으로 하는 간선이 존재한다는 것을 뜻합니다. 각 간선은 $O(\log T)$ 개의 새로운 활성화된 정점을 도입하니, 총 $O(S\log T)$ 개의 활성화된 정점이 세그먼트 트리에 존재합니다.

세그먼트 트리의 노드 p 에 대해서, 다음을 정의합시다:

- $ord(p, v)$: 노드 p 가 시간 구간 $[l, r]$ 을 대표한다고 하자. p 에 있는 모든 활성화된 정점들에 대해서, $s(v)[l \dots r]$ 를 사전순으로 비교했을 때 v 는 사전 순으로 몇 번째인가? (두 문자열이 같을 경우, 같은 번호를 배정)
- $rev(p, i)$: $ord(p, v) = i$ 인 정점은 무엇인가?
- $lcp(p, i)$: $rev(p, i)$ 와 $rev(p, i-1)$ 이 시간 $[l, x]$ 에 연결되어 있게끔 하는 최대 x 는 무엇인가?
- $ord^R(p, v)$: x^R 을 x 를 뒤집은 문자열로 정의하자. $(s(v)[l \dots r])^R$ 를 사전순으로 비교했을 때 v 는 사전 순으로 몇 번째인가? (두 문자열이 같을 경우, 아무 순서대로 배정)
- $rev^R(p, i)$: $ord^R(p, v) = i$ 인 정점은 무엇인가?
- $lcp^R(p, i)$: $rev^R(p, i)$ 와 $rev^R(p, i-1)$ 이 시간 $[l, x]$ 에 연결되어 있게끔 하는 최대 x 는 무엇인가?

이 정보를 저장하기 위해서는, p 에서 활성화된 정점에 비례하는 공간이 필요합니다. 고로 노드들의 공간 복잡도 합은 $O(S \log T)$ 입니다.

이제 위 정보들을 세그먼트 트리를 구성하면서 top-down으로 계산합시다. $m = (l + r)/2$ 라고 하고, v 가 Union-find의 루트라고 합시다 (아닌 경우는 $v = find(v)$ 로 맞춰 줍시다). $s(v)[l \dots r]$ 은, 다음 두 문자열을 합친 것과 동일합니다:

- 만약 v 가 왼쪽 자식에서 활성화된 노드라면, $s(v)[l \dots m]$. 아닐 경우 v 를 $m - l + 1$ 번 반복.
- 만약 v 가 오른쪽 자식에서 활성화된 노드라면, $s(v)[m + 1 \dots r]$. 아닐 경우 v 를 $r - m$ 번 반복.

$s(v)[l \dots r]$ 을 정렬하는 것은 이 두 쌍을 순서대로 정렬하는 것과 동일합니다. 왼쪽 자식에서 $s(v)$ 의 순서와 오른쪽 자식에서 $s(v)$ 의 순서가 이미 구해져 있으니, $ord(p, v)$ 는 $(ord(2p, v), ord(2p+1, v))$ 를 구해서 사전순으로 정렬한 후 좌표압축하는 식으로 구할 수 있습니다. 만약 v 가 활성화된 노드가 아니라면, $ord(*, v)$ 를 $v + n$ 으로 대체하는 식으로 처리할 수 있습니다. 즉, 위와 같이 정의된 두 쌍을 정렬하는 식으로 ord, rev 를 $O(S \log T \log N)$ 시간에 계산할 수 있습니다. 위 자료들을 통해서 임의의 서브트리에서 두 노드가 같은 문자열을 가지는지를 $O(\log N)$ 시간에 판단할 수 있으니, 세그먼트 트리를 이분탐색 하는 식으로 $lcp(p, i)$ 역시 동일한 복잡도에 계산할 수 있습니다. $(...)^R$ 에 대한 값은 뒤집어서 동일하게 계산하면 됩니다.

이제 쿼리를 처리하는 부분을 설명합니다. 쿼리는 세그먼트 트리를 타고 내려가면서 처리해 줍니다 (구성하는 것과 같은 분할 정복에서 해결하는 것이 효율적입니다). 쿼리 구간 $[l', r']$ 을 포함하는 가장 깊은 노드를 p 라고 합시다. p 에서 v 가 활성화되어 있지 않다면, p 로 내려가면서 활성화 되지 않게 된 가장 첫 지점을 통해서 답을 쉽게 계산할 수 있습니다.

p 에서 v 가 활성화되어 있음을 가정합시다. $s(*)[l \dots m]^R, s(*)[m+1, r]$ 을 기준으로 모든 활성화된 노드들이 정렬되어 있으니, $[l', m]$ 구간에서 문자열이 일치하는 v 는 $ord^R(2p, v)$ 값이 특정 구간에 있습니다. 이 구간은 $lcp^R(2p)$ 값을 사용하여 v 가 있는 지점 양 옆으로 세그트리에 이분탐색을 하여 계산할 수 있습니다. 비슷하게, $[m+1, r']$ 구간에서 문자열이 일치하는 v 는 $ord(2p+1, v)$ 값이 특정 구간에 있습니다. 고로 이 노드에서 우리가 구하는 것은, $ord^R(2p)$ 값이 특정 구간에 있고 $ord(2p+1)$ 값이 특정 구간에 있는 모든 정점 v 에 대해서 Union-find 컴포넌트의 크기 합을 구하는 것으로 환원됩니다. 다시 말해, 현재 노드에서 활성화된 모든 정점들을 2차원 평면에 올렸을 때, 각 쿼리는 2차원 직사각형 안에 들어간 정점의 가중치 합을 묻는 문제가 됩니다. 이는 스위핑을 사용하여 쉽게 해결할 수 있습니다.

최종 시간 복잡도는 $O(S \log T \log N + Q \log N)$ 입니다.