

에디토리얼



- 출제자: ryute, rdd6584, leejseo
- 검수자: lawali, mixnuts

감사의 말

- 대회 검수에 참여하여 원활한 대회 운영을 도와주신 lawali님과 mixnuts님께 진심으로 감사드립니다.
- BOJ라는 좋은 플랫폼 위에서 대회를 열 수 있게 해주신 startlink에도 감사드립니다.
- 마지막으로, 대회에 참가해주신 여러분들께도 감사의 말을 전합니다.

A. 필름

- 출제자: ryute

서로 다른 단색광끼리는 영향을 주지 않으므로, 모든 실험 기록에 대해서 빨간색, 파란색, 초록색의 세 단색광으로 나누어서 생각할 수 있다. 따라서 각 단색광에 대해서만 생각하면, 실험 결과에 모순이 생기는 경우는 다음 두 가지가 있다.

1. 빛을 쏘지 않았는데 실험 결과에서는 빛이 나오는 경우
2. 가능한 조합이 존재하지 않는 경우

1번과 같은 경우는 각 실험 기록에 대해 간단하게 판별해 줄 수 있다.

2번에 대해서는 필름 X 가 그 단색광을 포함하면 논리 변수 X 가 참이라고 하자. 그렇다면 다음 네 가지 경우에 대해서 논리식을 세울 수 있다. 필름 A 와 B 가 있을 때,

- 파장이 H이고, 반사된 빛에 특정 단색광이 포함되어 있는 경우:
두 필름에 모두 특정 단색광이 포함되어 있으므로, $(A \vee A) \wedge (B \vee B)$ 여야 한다.
- 파장이 H이고, 반사된 빛에 특정 단색광이 포함되어 있지 않은 경우:
두 필름에 특정 단색광이 동시에 포함되어 있으면 안 되므로, $(\neg A \vee \neg B)$ 여야 한다.
- 파장이 L이고, 반사된 빛에 특정 단색광이 포함되어 있는 경우:
두 필름 중 최소한 하나에는 특정 단색광이 포함되어 있어야 하므로, $(A \vee B)$ 여야 한다.
- 파장이 L이고, 반사된 빛에 특정 단색광이 포함되어 있지 않은 경우:
두 필름에 모두 특정 단색광이 포함되어 있지 않으므로, $(\neg A \vee \neg A) \wedge (\neg B \vee \neg B)$ 여야 한다.

모든 실험 기록에 대해서 다음과 같이 논리식을 정의해준 뒤 이를 \wedge 로 이어 주면, 이 논리 조합을 만족하는 변수들이 존재하는지 판별하는 문제로 환원시킬 수 있다. 한 클로저에 들어가는 변수의 수가 최대 2개이므로 2-SAT 문제로 이를 해결할 수 있다. 이를 위해서 SCC를 찾아줄 수 있고, 따라서 시간복잡도는 $\mathcal{O}(n + m)$ 이다.

B. 편안한 문자열

- 출제자: leejseo

S 의 길이를 N 이라 할 때, $N \leq 5,000$ 이므로 $\mathcal{O}(N^2)$ 시간 안에 수행되면 충분하다. 여기에서는 $\mathcal{O}(N^2)$ 풀이를 소개하겠다.

이 문제를 해결하는 것은 크게 두 가지 단계로 나눌 수 있는데,

1. 각 부분 문자열이 대칭 문자열인지 판별하는 것
2. 각 부분 문자열이 올바른 문자열인지 판별하는 것

이다.

1. 대칭 문자열 여부 판별

$1 \leq i \leq j \leq N$ 에 대하여 $D[i][j]$ 를 $i \sim j$ 번째 문자로 이루어진 부분 문자열이 대칭 문자열이면 True, 아니면 False라 하자. 배열 D 를 채우는 것은

<https://www.geeksforgeeks.org/count-palindrome-sub-strings-string/>와 비슷하게 동적 계획법을 이용해 $\mathcal{O}(N^2)$ 시간에 가능하다. 점화식은 다음과 같다.

$$D[i][j] = \begin{cases} \text{False} & \text{if } j = i \text{ or } j = i + 1 \text{ and } S[i \sim j] \neq "()" \\ \text{True} & \text{if } j = i + 1 \text{ and } S[i \sim j] = "()" \\ S[i] \neq S[j] \text{ and } D[i+1][j-1] & \text{otherwise} \end{cases}$$

2. 올바른 문자열 여부 판별

어떤 문자열이 올바른 문자열임은 여는 괄호와 닫는 괄호의 수가 같고, (와) 각각을 +1과 -1에 대응시켰을 때, 부분합이 음수가 되지 않는 것과 동치임이 널리 알려져 있다.

이 과정을 $\mathcal{O}(N^2)$ 개의 모든 부분 문자열에 대해 수행하면 $\mathcal{O}(N^3)$ 시간 복잡도로 수행시간 조건을 위반한다. 하지만, 자세히 생각해보면, $S[i \sim i]$, $S[i \sim i+1]$, $S[i \sim i+2]$, …, $S[i \sim N]$ 이 올바른지 판단하는 과정 모두에서 i 로 부터 시작하는 부분합이 사용된다. 따라서, 시작하는 위치가 같은 부분 문자열들을 한 번에 처리해주면 $\mathcal{O}(N^2)$ 시간 복잡도로 각 부분 문자열이 올바른 문자열인지의 여부를 판단할 수 있다.

C. 일해라, 류트!

- 출제자: leejseo

화학 약품 i 를 1번 파이프로 넣는 시각을 S_i 라 하자. 문제의 조건에 의해 $T_i = (L_1 + L_2 + \dots + L_M) \cdot r_i$ 임을 알 수 있다. 그렇기에, S_i 들을 구할 수 있다면, 문제를 해결할 수 있다.

$S_{i+1} - S_i$ 를 구할 수 있다면 좋을 것 같다. 문제의 조건을 간단히 요약하자면, 이미 S_i 를 구했을 때, S_{i+1} 을 찾는 문제는

- 각각의 $1 \leq j \leq M$ 에 대하여 화학 약품 i 가 P_j 를 빠져 나오는 시각으로부터 C_j 이상의 시간이 흐른 이후에 화학 약품 $i+1$ 이 P_j 로 들어가야 한다.

를 만족하는 최소의 시각을 찾는 문제임을 알 수 있다. $A_i = L_1 + L_2 + \dots + L_i$ 라 하고, $A_0 = 0$ 이라 하자. 그러면, 위 조건을 수식으로 바꾸자면,

- 각각의 $1 \leq j \leq M$ 에 대하여:

$$S_i + r_i \cdot A_j + C_j \leq S_{i+1} + r_{i+1} \cdot A_{j-1}$$

이 성립해야한다.

이는,

$$S_{i+1} - S_i \geq C_j + r_i \cdot A_j - r_{i+1} \cdot A_{j-1}$$

가 각각의 j 에 대해 성립함과 동치이다. 이 조건을 이용해서 S_i 들을 전부 구하려고 하면 $\mathcal{O}(NM)$ 시간이 소요되어 수행 시간 조건을 위반한다.

그런데, $S_{i+1} - S_i$ 의 식을 관찰해보면 $(r_i, r_{i+1}) = (r_j, r_{j+1})$ 이면, $S_{i+1} - S_i = S_{j+1} - S_j$ 임을 쉽게 확인할 수 있다. 다시 말해, 모든 가능한 (r_i, r_{i+1}) 쌍에 대해 $S_{i+1} - S_i$ 를 미리 구해 놓으면, 불필요한 계산을 줄일 수 있다. 그런데, r_i 의 범위가 작아서 가능한 (r_i, r_{i+1}) 의 쌍이 10,000개 밖에 없으므로, 전처리를 통해 $\mathcal{O}(N + 10^4M)$ 시간에 문제를 해결할 수 있다.

- 참고: 32비트 정수 변수 범위를 초과할 수 있으므로 64비트 정수 변수를 사용해야 한다.

D. 감성 테트리스

- 출제자: rdd6584

서브태스크 1

위아래로 면을 공유하는 블럭들끼리 간선으로 연결되어 있다면, 쿼리의 정답은 이 블럭에서 아래방향으로 갈 수 있는 블럭의 개수가 정답이 된다. 각 블럭들간의 간선을 연결해보자. 위치 X 에 있는 블럭 중 가장 위에 있는 블럭은 무엇인지. 또 그 높이는 얼마인지를 저장하자.

'|'자 블럭의 경우 위치 a 에 있는 블럭과 간선을 이어주는 것으로 해결할 수 있고, '—'자 블럭의 경우 위치 $a \sim a + 3$ 에 있는 블럭들 중 높이가 가장 높은 블럭들과 간선을 이어주면 된다.

서브태스크 2

1. 블럭은 한 종류이며 길이가 모두 같다. 따라서 어떤 블럭을 기준으로 인접해 있는 블럭은 위쪽방향, 아래쪽방향에서 각각 최대 2개씩 존재할 수 있다.
2. 블럭들은 수직으로 떨어지므로 스택처럼 점점 위로 쌓인다. 따라서 이미 형성된 블럭의 자식노드는 추가되거나 변하지 않는다.
3. 2번의 사실을 이용해서 어떤 블럭이 누르고 있는 원소의 개수를 메모이제이션 해줄 수 있다. 어떤 블럭 아래에 있는 블럭을 child라고 하자. 1번을 이용해서, $DP[i] = 1 + DP[\text{left_child}] + DP[\text{right_child}]$ 로 정의하면 좋겠지만... 아쉽게도 겹치는 부분이 있다. 하지만 생각해보면 겹치는 부분은 어떤 원소를 root로 하는 subtree로 정의된다. 그러한 subtree의 root를 Least_Common_Child (LCC)라고 해보자. $DP[i] = 1 + DP[\text{left_child}] + DP[\text{right_child}] - DP[\text{LCC}]$ 이 된다.
4. 이 문제는 평면그래프로 표현되며 아래쪽부터 연속적으로 쌓이기 때문에 이러한 LCC는 유일하게 존재한다.
5. 4번의 사실을 이용하면, 오른쪽 자식과 그의 자식들은 왼쪽 방향으로, 왼쪽 자식과 그의 자식들은 오른쪽 방향으로만 탐색한다면 common_child중, 가장 상위에 있는 common_child를 찾을 수 있고 이 블럭이 LCC이다.
6. 어떤 원소에 대해 sparse table을 왼쪽/오른쪽 자식 방향으로만 진행했을 때의 도착점으로 정의하자. 만약 LCC를 넘어가는 깊이를 선택하게 될 경우,
 left_child 의 오른쪽 자식의 위치 $\geq \text{right_child}$ 의 왼쪽 자식의 위치가 되고, 이는 LCC거나 이를 이미 지나쳐버렸다는 얘기가 된다. 즉, 결정문제로 치환할 수 있고. 이분탐색을 통해 LCC를 구해줄 수 있다. 이제 3번의 식을 이용하여 각 쿼리를 해결할 수 있다.

서브태스크 3

'—'자 모양은 4개의 위치를 차지하게 되고, '|'자 모양은 1개의 위치를 차지하게 된다. 따라서 하나의 블럭당 최대 4개의 부모, 자식 블럭이 생길 수 있다. 서브태스크2에서의 DP식을 일반화해보자. 이는 결국 자식 블럭들의 합집합을 구하는 것과 같다. 가장 왼쪽에 있는 자식부터 끝에 있는 자식까지 1, 2, 3, 4라는 이름을 붙여보자. A 와 B 의 LCC를 $A \cap B$ 라고 해보자.

결국 쿼리의 정답은 $DP[1] + DP[2] + DP[3] + DP[4] - DP[1 \cap 2] - DP[1 \cap 3] - DP[1 \cap 4] - DP[2 \cap 3] - DP[2 \cap 4] - DP[3 \cap 4] + DP[1 \cap 2 \cap 3] + DP[1 \cap 2 \cap 4] + DP[1 \cap 3 \cap 4] + DP[2 \cap 3 \cap 4] - DP[1 \cap 2 \cap 3 \cap 4]$ 이 된다. 3개 항에서의 LCC는 어떻게 구할까? 서브태스크2의 5번처럼 탐색을 진행할 경우, $1 \cap 2$, 그리고 $2 \cap 3$ 은 항상 $1 \cap 3$ 을 찾으려 가는 경로상에 존재한다는 사실을 알 수 있다. 따라서 $(A < B < C)$ 에서 $A \cap B \cap C$ 는

$A \cap C$ 와 동치이며, 항이 4개일 때도 위 내용은 적용된다.

따라서 저 식을 정리하면, $DP[1] + DP[2] + DP[3] + DP[4] - DP[1 \cap 2] - DP[2 \cap 3] - DP[3 \cap 4]$ 와 같아진다.

이제 이러한 식을 서브태스크4에 적용시켜보자. 우리가 정의한 sparse table을 이용한 탐색은 모든 블럭의 높이가 같다고 가정한 경우에 성립한다. 하지만 ‘|’자 블럭과 ‘—’자 블럭의 높이는 각각 다르기 때문에, 이를 보완해 줄 필요가 있다. ‘|’자 블럭을 ‘.’자 블럭 4개를 위로 쌓아올린 형태로 생각하고 적절한 처리를 해주면, 어렵지 않게 sparse table을 정의하고 LCC를 탐색해 줄 수 있다.

E. 잉크를 엎질렸다

- 출제자: rdd6584

서브태스크 1

어떤 문자열 S 에 대해 대응되는 Z배열은 오직 하나이다. 즉, '#'자리에 들어갈 문자를 미리 정해주고 모든 경우에 대해 Z알고리즘을 돌려서 일치하는 Z배열이 나오는지 확인하면 된다. $\mathcal{O}(N \cdot 26^{\#})$

서브태스크 2

1. Z배열을 통해서 아래의 논리관계를 얻을 수 있다.
 - 임의의 $Z[i]$ 에 대해 $S[0 \sim Z[i] - 1]$ 과 $S[i \sim i + Z[i] - 1]$ 이 같음. (단, $Z[i] > 0$)
 - 임의의 $Z[i]$ 에 대해 $S[Z[i]]$ 와 $S[i + Z[i]]$ 가 다름. (단, $i + Z[i] < N$)
2. 논리관계를 통해, 같아야 하는 인덱스, 문자를 하나의 그룹으로 묶을 수 있고, 서로 달라야하는 것끼리 같은 그룹에 있다면 이는 모순된 결과다.
3. 남은 일은 #에 대해 적절히 논리관계를 배치시키고, 남은 문자를 배정하는 일이다. 우리가 사용할 수 있는 문자는 A-Z로 총 26개이므로, #에 대해 남은 문자를 적절히 배치하는 것은 최소 클리크 커버 문제와 동치임을 알 수 있다. 이를 완전탐색으로 해결하면 $\mathcal{O}(N^2 + 26^{\#})$ 이 된다.

논리관계에 모순이 없고, #개수가 26개를 넘지 않는다면 완전탐색 과정에서 뒤로 되돌아가는 간선이 존재하지 않는다는 것을 증명할 수 있다. 따라서 이 풀이는 사실 $\mathcal{O}(N^2 + 26^{\#})$ 이 아니라 $\mathcal{O}(N^2)$ 이 된다.

서브태스크 3

$\sum_{i=0}^{N-1} Z[i]$ 번의 논리를 사용하지 않고도 우리가 만든 배열이 올바른지 확인하는 방법을 우리는 알고 있다. 서브태스크1과 같이 이 Z배열을 만드는 알고리즘인 Z알고리즘을 이용하는 것이다. 현재 존재하는 모든 #에 대한 논리만을 기록해서 #을 채워보자. 이 논리는 많아야 $\# \cdot N$ 개임을 알 수 있다.

#에 대한 모든 문자 배정을 그리디하게 완료했다면 Z알고리즘을 돌려서 이와 일치하는지 확인해보자. 우리는 서브태스크2의 이유로 두번 이상의 Z알고리즘을 돌릴 필요가 없음을 알고 있다. Z배열이 일치한다면 문자열을 그대로 출력하면 되고, 그렇지 않다면 THINKINGFACE를 출력하면 된다. 이 경우 시간복잡도는 $O(N * \#)$ 이 된다.

F. 고양이 소개팅

- 출제자: ryute

서브태스크 1

보금자리들을 정점으로 생각하면, 정점들은 선형으로 연결되어 있다. 모든 수컷 고양이가 사는 정점에 대하여, 고양이가 아래로 어느 정점까지 떨어질 수 있는지를 저장해 둘 수 있다. 아직 소개팅을 하지 못한 암컷 고양이가 사는 정점 중 가장 아래쪽에 있는 정점에 대해서 생각해 보자. 그 정점으로 떨어질 수 있는 수컷 고양이 정점들의 집합이 있을 때 그 중에서 가장 먼저 이어줘야 하는 정점은, 정점들 중 가장 높이가 낮은 정점이다. 이는 매우 간단하게 증명할 수 있다.

따라서 모든 수컷 고양이 정점에 대해 내려갈 수 있는 높이를 저장 후 정렬해 두고, 가장 낮은 높이의 암컷 고양이 정점부터 살피면서 그 정점에 도달할 수 있는 모든 수컷 고양이 정점을 관리해줄 수 있다. 이는 amortized $\mathcal{O}(n)$ 에 가능하다. 그 중 가장 낮은 정점을 찾는 것은 priority queue 등을 이용해서 할 수 있으며, 총 시간복잡도는 $\mathcal{O}(n \lg n)$ 이 된다.

서브태스크 2

이 작으므로, 그리디 전략을 Naive하게 구현하는 것을 시도해 볼 수 있다. 수컷 고양이들을 루트로부터의 거리 순서대로 정렬하자. 이 고양이들은 트리 구조를 이루고 있으므로 루트로부터의 거리가 먼 순서대로 떨어트리는 것이 항상 최적임을 알 수 있다. 각 수컷 고양이를 떨어트릴 때 최대의 결과값을 얻으려면 떨어질 수 있는 가장 아래쪽 암컷 고양이 정점부터 채워야 함이 자명하다. 따라서 각 수컷 고양이 노드마다 서브트리에 존재하는 모든 노드들을 순회해 줄 수 있고, 시간복잡도는 $\mathcal{O}(n^2)$ 이 된다.

서브태스크 3

수컷의 뛰어내릴 수 있는 최대 거리가 같다는 것은 암컷이 뛰어올라갈 수 있는 최대 거리가 같다는 것과 동치이다. 이번에는 가장 아래에 있는 암컷부터 순서대로 탐색을 진행해보자. 이 암컷 고양이와 매칭될 수 있는 수컷들은 이 암컷의 조상들이므로 선형적으로 나타나게 된다. 따라서 우리는 임의의 암컷과 가장 가까운 수컷부터 매칭해야 함을 알 수 있다. 이 과정에서 아직 매칭되지 않은 수컷을 찾는 과정은 한 암컷당 $\mathcal{O}(N)$ 이지만 이를 개선해서 더 짧은 시간 안에 수행할 수 있다. 이미 모든 고양이가 매칭된 인접한 수컷 고양이 정점들끼리는 이 중 가장 상위에 있는 정점을 disjoint-set에서의 대표 노드로 설정해 이 노드에 대해서만 문제를 해결할 수 있다. 암컷 고양이 정점 또한 0마리의 수컷이 있는 정점과 같으므로 disjoint-set으로 같이 관리해 줄 수 있다.

서브태스크 4

서브태스크 2의 풀이에서 시간복잡도를 늘리는 주 원인은 각 수컷 고양이 노드마다 서브트리에 존재하는 모든 암컷 고양이 노드들을 순회해 주어야 한다는 점이다. 그 이유는 어떤 수컷 고양이 정점에서 떨어질 수 있는 최대 거리 제한 조건에 의해 갈 수 없는 정점들이 있기 때문이다. 그렇다면 이와 같은 정점들을 빠르게 처리해 줌으로서 시간을 아낄 수 있을 것이라고 생각해볼 수 있다. 트리를 DFS ordering 한 후 세그먼트 트리를 구축하고, 세그먼트 트리의 각 노드에 `std::set`을 비롯한 이진 탐색 트리를 넣어 줄 수 있다. `std::set`을 그 구간을 덮는 정점들의 거리를 기준으로 정렬하도록 해 두면 `std::set::upper_bound`로 어떤 정점에서 갈 수 있고 고양이가 남아 있는 암컷 고양이 정점들을 빠르게 찾을 수 있다. 2D 세그먼트 트리를 구축해도 완전히 동일한 작업을 할 수 있다. 이 풀이의 시간복잡도는 $\mathcal{O}(n \lg^2 n)$ 이다.

여담

의도한 풀이는 아니었지만, Small-to-Large 테크닉을 사용해서 set에 원소들을 넣고 관리해 줌으로서 더 쉽게 $\mathcal{O}(n \lg^2 n)$ 으로 문제를 해결할 수 있다고 한다.

잡담

- 난이도 조절에 약간 실패한 것 같아서 초보자 분들께 죄송할 따름입니다...
- 대회 치신 분들 정말 수고 많으셨습니다. 새해 복 많이 받으세요!

아래의 대화는 톡방에서 실제로 오간 대화이다.

leejseo: 혹시 에디토리얼에 잡담 넣고 싶은 것 있으신가요?

ryute: “본격 풀린 문제보다 뚫린 문제가 더 많은 대회!” 하나 넣어주세요

rdd6584: 그러면 저는 “흑 류트가 절 저격했어요”라고 넣어주세요

ryute: 그 아래에다가 “일 안해서 죄송합니다” 하나만 더 넣어주세요

rdd6584: ㅋㅋㅋㅋㅋㅋㅋㅋㅋ류트 일 많이 했죠 F번 풀이도 모를 뿐