



# **Manosaba Beginner Contest Official Editorial**

|    |             |                   |    |
|----|-------------|-------------------|----|
| A  | Easy        | 마법소녀              | 3  |
| B  | Easy        | 마녀재판              | 5  |
| C  | Medium      | Sar-gedy          | 8  |
| D1 | Medium      | 탈옥 작전 (Miria)     | 12 |
| D2 | Hard        | 탈옥 작전 (Nanoka)    | 14 |
| D3 | Medium      | 탈옥 작전 (Alisa)     | 18 |
| D4 | Hard        | 탈옥 작전 (Sherry)    | 21 |
| D5 | Medium      | 탈옥 작전 (Hanna)     | 24 |
| D6 | Medium      | 탈옥 작전 (Coco)      | 26 |
| E  | Hard        | 2시에 혼자 있었다니, 이상해! | 28 |
| F  | Hard        | 나를 봐줘!            | 33 |
| G  | Hard        | fiXmArge          | 38 |
| H  | Challenging | 붉은 나비             | 46 |

---

## A. 마법소녀

#constructive

출제진 의도 - Easy

✓ 출제자: flakepym

✓ 제출: 178번, 정답: 125명 (정답률 70.22%), 처음 푼 사람: yyyy7089, 1분

## A. 마법소녀

- ✓ 선택한  $y_i$ 의 값이 큰 순서대로 최소  $k$ 개의 인덱스를 선택하는 문제입니다.
- ✓ 단순히 내림차순으로 정렬한 후 첫  $k$ 개를 고르면, 정렬했을 때  $k$ 번째 값과  $k + 1$ 번째 값이 같을 때 조건을 만족하지 않게 됩니다.
- ✓ 따라서 같은 값들끼리는 묶어서 하나라도 선택되었다면 나머지 모든 인덱스들도 선택되도록 처리해야 합니다.
- ✓ 정렬된 값으로부터 원래 인덱스를 복원할 때는, 해시 테이블을 사용하거나, 처음부터 (값, 인덱스) pair들을 만든 후 정렬하면 쉽습니다.
- ✓ 하지만 마녀가 될 가능성이 있는 인간을 최소한으로 선택할 필요는 없습니다.
- ✓ 모든 인간이 마녀가 될 가능성이 있도록 하면 항상 문제의 조건을 만족합니다.

---

## B. 마녀재판

#ds, #greedy, #implementation

출제진 의도 - Easy

✓ 출제자: ThatOnePythonUser

✓ 제출: 196번, 정답: 123명 (정답률 62.76%), 처음 푼 사람: sigtuna, 3분

## B. 마녀재판

- ✓ 사람들의 투표가 주어졌을 때, 각자 몇 표를 받았는지 어떻게 알 수 있을까요?
- ✓ 표의 개수를 세는 배열을 활용하면 됩니다.
- ✓ 크기가  $N$ 인 `cnt`라는 배열을 만듭니다.
  - ✓ `cnt[i]`에  $i$ 번째 소녀가 받은 표의 수를 저장합니다.
  - ✓ 모두의 표를 보면서,  $x$ 번째 소녀가 표를 받았다면 `cnt[x]`를 1 증가시킵니다.
  - ✓ `cnt` 배열의 최댓값이 있는 위치가 가장 많이 표를 받은 사람이 됩니다.

## B. 마녀재판

- ✓ cnt 배열의 최댓값, 즉  $A$ 의 최빈값을  $m$ 이라고 합시다.
- ✓  $m$ 표를 받은 사람이 「나」로 유일하면 처형을 피할 수 없으므로  $-1$ 을 출력합니다.
  - ✓  $m - 1$ 표를 받은 사람이 있더라도, 투표하면 동점이 되어 처형당합니다.
- ✓ 그렇지 않다면, 「나」를 제외한  $m$ 표를 받은 사람이 존재합니다. 그 사람에게 투표하면 혼자서  $m + 1$ 표가 되어 처형되고, 「나」는 살아남을 수 있습니다.
- ✓ 구현 팁으로, 순서쌍  $(A_i, i)$ 의 최댓값이  $i = 1$ 인지 확인하면 됩니다. 「나」는 1번이므로,  $(A_1, 1)$ 보다 큰 인덱스가 없다면 최빈값은 「나」로 유일하다는 의미입니다.
- ✓ 이 풀이의 시간 복잡도는  $\mathcal{O}(N)$ 입니다.
- ✓ 별개로, 모든  $i$ 에 대해서  $\text{cnt}[i]$ 를 1 증가시켜보고, 최댓값이 유일하며 「나」가 아닌지 확인하는 브루트포스도 가능합니다. 이 풀이의 경우  $\mathcal{O}(N^2)$ 에 해결할 수 있습니다.

---

## C. Sar-gedy

#string, #bruteforcing, #hashset

출제진 의도 - Medium

✓ 출제자: flakepym

✓ 제출: 141번, 정답: 109명 (정답률 77.30%), 처음 푼 사람: 2, 3분

## C. Sar-gedy

- ✓ 모든 이름  $s_i$ 들을 저장한 해시 테이블을 관리합니다.
  - ✓ C++의 `unordered_map`, Python의 `dictionary` 등을 사용할 수 있습니다.
- ✓ 해시 테이블은 이름  $\rightarrow$  죄수번호 형태로 관리합니다. 만약 해시 테이블에 이름을 넣기 직전에 이미 테이블에 이름이 있다면, 기존에 저장된 죄수번호와 새로 저장하려는 죄수번호 중 최댓값으로 갱신하면 됩니다.
- ✓ 3가지 풀이 방법을 생각해볼 수 있습니다.
- ✓ 첫 번째 방법은, 각 기억 속 이름  $q$ 가 들어올 때마다, 해시 테이블의 모든 이름  $s_i$ 에 대해서  $q$ 와  $s_i$ 가 비슷한지 확인하는 것입니다.
  - ✓ 두 이름의 길이가 같고, 알파벳이 다른 위치가 최대 1개 있는지 확인하면 됩니다.
- ✓  $q$ 와 비슷한 이름 중 죄수번호의 최댓값을 출력하면 됩니다.
- ✓  $\mathcal{O}(N^2 |s_i|)$ 의 복잡도에 해결할 수 있습니다.

## C. Sar-gedy

- ✓ 두 번째 방법으로, 각 기억 속 이름  $q_i$ 의 각 인덱스  $j$  ( $1 \leq j \leq |q_i|$ )에 대해,  $q_i$ 의  $j$ 번째 글자가 알파벳 소문자라면,  $j$ 번째 글자를 a로 대체한 이름, b로 대체한 이름, ..., z로 대체한 이름을 모두 해시 테이블에서 검색합니다.
- ✓ 예를 들어 sakuraba\_ema와 비슷한 이름을 검색하고 싶을 때,  $j = 1$ 에서는 aakuraba\_ema, bakuraba\_ema, ..., zakuraba\_ema를 모두 확인하는 것입니다.
- ✓ 만약 테이블에서 비슷한 이름을 하나도 찾지 못했다면 -1을 출력하면 됩니다.
- ✓ Python의 경우, defaultdict(lambda: -1)를 사용하면 이 과정을 생략할 수 있습니다.
- ✓  $\mathcal{O}(26N |s_i|^2)$ 에 해결할 수 있습니다.

## C. Sar-gedy

- ✓ 마지막으로, 해시 테이블에 이름을 저장하고 조회할 때,  $a$ 부터  $z$ 까지 전부 하나씩 대체해보는 대신  $*$  등의 와일드카드 문자를 사용하면 26배 더 적은 반복으로 찾을 수 있습니다.
- ✓ 예를 들어 이름 `sakuraba_ema`와 비슷한 이름을 테이블에 넣고 싶을 때, `*akuraba_ema`, `s*kuraba_ema`, ..., `sakuraba_em*`를 넣는 것입니다.
- ✓ 조회도 마찬가지로 처리하면, 시간 복잡도와 공간 복잡도는 모두  $\mathcal{O}(N |s_i|^2)$ 입니다.

---

## D1. 탈옥 작전 (Miria)

#grid\_graph, #bfs, #shortest\_path

출제진 의도 - Medium

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: TKTYI, 21분

## D1. 탈옥 작전 (Miria)

- ✓  $s, E, ., !$ 를 갈 수 있는 정점으로 간주하고, 격자 그래프에서  $s$ 를 시작 정점으로 놓고 BFS로 최단 거리  $d(u)$ 를 구합니다.
- ✓ 단, 정점  $u$ 와 인접한 어떤 ! 정점  $v$ 를 큐에 넣기 전에  $d(v) = d(u) + 1 \geq T$ 이면 갈 수 없는 정점으로 판정하고, 큐에 넣지 않아야 합니다.
- ✓ BFS의 특성상 처음으로 방문한 정점은 최단 시간으로 방문한 것이 되는데, 이때 제한 시간 안에 진입할 수 없다는 것은 더 늦게 방문하더라도 진입할 수 없다는 것을 의미하기 때문입니다.
- ✓ 이 방법으로  $E$ 에 도달할 수 있다면, 최단 거리  $d(E)$ 를 출력합니다.
- ✓ 그렇지 않다면, !도 항상 진입할 수 있는 칸으로 간주하고 BFS를 수행한 후 최단 거리를 출력합니다. 이번에도  $E$ 에 도달할 수 없으면  $-1$ 을 출력합니다.
- ✓ 구현 팁으로,  $T$ 를 매우 큰 수로 놓고 BFS를 돌리면 이전에 사용한 BFS 함수를 재활용할 수 있습니다.

---

## D2. 탈옥 작전 (Nanoka)

#grid\_graph, #mcmf

출제진 의도 - Hard

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: dadasO8, 63분

## D2. 탈옥 작전 (Nanoka)

- ✓ S, E, ., !로 구성된 격자 그래프를 생각해 봅시다.
- ✓  $S \rightarrow E \rightarrow S \rightarrow E$  경로에서,中间的 E  $\rightarrow$  S를 뒤집어서 생각하면 결국 겹치지 않는 3개의  $S \rightarrow E$  경로의 최단 거리 합을 구하는 문제와 같습니다.
- ✓ 이는 모든 간선의 용량을 1로 생각했을 때, 최소 비용으로 유량 3을 흘리는 Minimum Cost Maximum Flow (MCMF) 문제로 생각할 수 있습니다.
- ✓ 단, 일반적인 MCMF에서는 정점에는 용량이 없어서 겹치는 것이 허용되고, 격자 그래프에서는 한 정점에 최대 4개의 간선이 연결되어 있으므로 4의 유량이 흐를 수도 있습니다. 이를 해결하기 위해 다음과 같이 그래프를 모델링할 수 있습니다.

## D2. 탈옥 작전 (Nanoka)

- ✓ 모든 정점을 들어오는 간선 전용  $u_{\text{in}}$  과 나가는 간선 전용  $u_{\text{out}}$  으로 분할합니다.
- ✓ 모든 인접한  $u, v$  에 대해서 용량과 비용이 모두 1인 간선  $u_{\text{out}} \rightarrow v_{\text{in}}$  과  $v_{\text{out}} \rightarrow u_{\text{in}}$  을 연결합니다.
- ✓ 경로가 정점에서 겹치는 것을 막기 위해 모든  $u$  에 대해 용량이 1이고 비용이 0인 간선  $u_{\text{in}} \rightarrow u_{\text{out}}$  을 연결합니다.
- ✓ 최대 유량을 3으로 제한하기 위해 용량이 3이고 비용이 0인 간선을  $\text{source} \rightarrow S_{\text{out}}$  과  $E_{\text{in}} \rightarrow \text{sink}$  에 연결합니다.
- ✓ 이렇게 하면  $S_{\text{in}} \rightarrow S_{\text{out}}$  과  $E_{\text{in}} \rightarrow E_{\text{out}}$  은 사용하지 않기 때문에, S와 E에서 경로가 겹치는 것을 허용할 수 있습니다.

## D2. 탈옥 작전 (Nanoka)

- ✓ 이 그래프에서 MCMF로 최대 유량과 최소 비용을 구합니다. 최대 유량이 3이면 최소 비용을 출력하고, 그렇지 않으면  $-1$ 을 출력합니다.
- ✓ 최대 유량이  $f = 3$ 이므로, 표준적인 SPFA 기반 MCMF를 사용할 경우 시간 복잡도는  $\mathcal{O}(VEf) = \mathcal{O}(N^2M^2)$ 입니다. 실제로는 격자 그래프의 특성상 증가 경로를 최대한 탐색하는 데에 한계가 있기 때문에, 상수가 작아서 시간 안에 통과할 수 있습니다.
- ✓ 이 그래프의 경우 초기에 음수 비용을 가진 간선이 없으므로,  $\mathcal{O}(Ef \log V) = \mathcal{O}(NM \log NM)$ 에 작동하는 Dijkstra 기반의 MCMF 알고리즘을 사용할 수도 있습니다.

---

## D3. 탈옥 작전 (Alisa)

#grid\_graph, #bfs, #shortest\_path

출제진 의도 - Medium

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: dadasO8, 44분

### D3. 탈옥 작전 (Alisa)

- ✓  $S, E, .$  정점으로 구성된 격자 그래프를 생각합니다.
- ✓ 먼저, 호숫가에 5 단위시간 안에 도달할 수 있는 칸을 판정해봅시다.
- ✓ 호숫가는 호수(#)로부터 거리 1인 정점으로 생각할 수 있습니다. 따라서 위의 조건은 호수까지 6 단위시간 안에 도달할 수 있다는 것과 같습니다.
- ✓ 호수로부터의 거리를 저장하는 배열  $L$ 을 만듭니다. 먼저 모든 호수 정점  $u$ 를 큐에 넣고, 이 정점들에 대하여  $L(u) = 0$ 으로 초기화합니다.
- ✓ 이제 큐가 빌 때까지 BFS를 돌려서 최단 거리를 구하면,  $L(u)$ 는 가장 가까운 호수까지의 최단 거리가 됩니다. 이러한 방식을 multi-source BFS라고 합니다.
- ✓ 가상의 정점  $s$ 를 만들어서, 모든 호수와  $s$ 를 연결한 후  $s$ 로부터의 최단 거리를 구하고 1을 빼는 방식도 가능합니다. 사실 multi-source BFS의 정당성은 이 방법과 동치임을 이용하여 증명할 수 있습니다.

### D3. 탈옥 작전 (Alisa)

- ✓  $L$ 을 구했으면 다시  $s$ 를 시작 정점으로 두고,  $1 \leq L(u) \leq 6$ 인 정점만 사용해서 BFS로  $E$ 까지의 최단 거리를 구하면 됩니다.
- ✓ 이때, 호수에서 도달 불가능한 정점을  $L(u) = -1$  등으로 표시했을 경우, 이러한 정점을 사용하지 않도록 주의합니다. 특히  $s$ 가 호수에서 도달 불가능한 정점일 경우는 BFS를 시작하지도 않아야 합니다.

---

## D4. 탈옥 작전 (Sherry)

#grid\_graph, #bfs, #shortest\_path

출제진 의도 - Hard

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: paracompact, 17분

## D4. 탈옥 작전 (Sherry)

- ✓ S, E, ., x, !로 구성된 격자 그래프를 생각해 봅시다.
- ✓ 세리가 벽을 부순 개수에 따라서 정점  $s$ 에서 정점  $u$ 까지의 최단 거리가 달라질 수도 있습니다.
- ✓ 이러한 문제를 해결하기 위해 다음과 같이 그래프를 구성해볼 수 있습니다.
  - ✓ 세리가 벽을  $k$ 개 부순 세계선을 생각합니다.  $k = 0$ 부터  $k = 5$ 까지 6가지의 세계가 있고, 각 세계에는 정점이  $NM$ 개씩 존재하므로, 그래프의 총 정점 개수는  $6NM$ 입니다.
  - ✓ 즉, 일반적인 격자 BFS에서는 위치  $(i, j)$ 를 정점으로 생각하지만, 여기서는 벽을 부순 개수  $k$ 까지 포함해서  $(k, i, j)$ 를 하나의 정점으로 취급하는 그래프를 구성하는 것입니다.

## D4. 탈옥 작전 (Sherry)

- ✓ 처음에는 벽을 부수지 않은 세계의  $s$  정점  $(0, s)$ 에서 출발합니다. 여기서  $(k, u)$ 까지의 거리를  $d(k, u)$ 로 정의합니다.
- ✓ 위치  $u$ 에서 인접한 위치  $v$ 로 이동할 때,
  - ✓  $S_v \neq x$ 이면  $(k, u)$ 에서  $(k, v)$ 로 이동할 수 있습니다.
  - ✓  $S_v = x$ 이면 벽을 부순 것이므로  $(k, u)$ 에서  $(k+1, v)$ 로 이동할 수 있습니다.
- ✓ 이와 같이 구성된 그래프에서 BFS로 최단 거리  $d(k, u)$ 를 구합니다.
- ✓ 단, 정점  $(k, u)$ 에서 인접한 정점  $(k, v)$ 로 이동할 때,  $v$ 가 ! 정점이고  $d(k, v) = d(k, u) + 1 \geq T$ 이면 갈 수 없는 정점으로 판정하고, 큐에 넣지 않아야 합니다.
- ✓ 이 방법으로 도달할 수 있는 정점  $(k, E)$  중 거리가 가장 작은 것을 출력합니다.
- ✓ 벽을 최대  $K = 5$ 개 부술 수 있을 때, 시간 복잡도는  $\mathcal{O}(KNM)$ 입니다.

---

## D5. 탈옥 작전 (Hanna)

#grid\_graph, #dijkstra

출제진 의도 - Medium

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: dadas1557, 17분

## D5. 탈옥 작전 (Hanna)

- ✓ S, E, ., #, !를 갈 수 있는 정점으로 간주한 격자 그래프를 생각합니다.
- ✓ 인접한 두 정점 중 호수(#)가 존재하면 가중치 5의 간선으로, 그렇지 않으면 가중치 1인 간선으로 취급합니다.
- ✓ S를 시작 정점으로 놓고 다익스트라 알고리즘을 통해 최단 거리  $d(u)$ 를 구합니다.
- ✓ 이때, 정점  $u$ 와 인접한 어떤 ! 정점  $v$ 를 우선순위 큐에 넣기 전에  $d(u) + w(u, v) \geq T$ 이면 우선순위 큐에 넣지 않아야 합니다.
  - ✓ 단, 이 조건을 만족한다고 해서 정점을 영구적으로 가지 못하게 표시하면 안 됩니다. 아직 방문하지 않은 다른 정점  $u'$ 에 대해  $d(u') + w(u', v) < T$ 를 만족할 수도 있기 때문입니다.
- ✓ 이 방법으로 E에 도달할 수 있다면 최단 거리  $d(E)$ 를 출력하고, 그렇지 않다면 -1을 출력합니다.

---

## D6. 탈옥 작전 (Coco)

#grid\_graph, #bfs, #shortest\_path

출제진 의도 - Medium

✓ 출제자: ThatOnePythonUser

✓ 제출: 205번, 정답: 73명 (정답률 35.61%), 처음 푼 사람: dadasO8, 36분

## D6. 탈옥 작전 (Coco)

- ✓  $s, E, ., !$ 를 갈 수 있는 정점으로 간주하고, 격자 그래프에서  $s$ 를 시작 정점으로 놓고 BFS로 최단 거리  $d(u)$ 를 구합니다. 이때,  $d(s) = 0$ 으로 둡니다.
- ✓ 단, 정점  $u$ 와 인접한 어떤 ! 정점  $v$ 를 큐에 넣기 전에  $d(v) = d(u) + 1 \geq T$ 이면 갈 수 없는 정점으로 판정하고, 큐에 넣지 않아야 합니다.
- ✓ BFS의 특성상 처음으로 방문한 정점은 최단 시간으로 방문한 것이 되는데, 이때 제한 시간 안에 진입할 수 없다는 것은 더 늦게 방문하더라도 진입할 수 없다는 것을 의미하기 때문입니다.
- ✓ 이 방법으로  $E$ 에 도달할 수 있다면, 이때의 최단 거리를  $t$ 라고 합니다. 귀환 가능성을 판정하려면  $E$ 를 시작점으로 놓고, 똑같은 BFS를 돌립니다. 단, 이번에는  $d(E) = t$ 로 시작합니다.
- ✓  $E$ 에서 출발하여  $s$ 에 도달할 수 있다면  $t$ 를 출력하고, 그렇지 않다면  $-1$ 을 출력합니다.

---

## E. 2시에 혼자 있었다니, 이상해!

#treeset, #priority\_queue

출제진 의도 - Hard

✓ 출제자: ThatOnePythonUser, nanoka

✓ 제출: 81번, 정답: 48명 (정답률 59.26%), 처음 푼 사람: 79brue, 16분

## E. 2시에 혼자 있었다니, 이상해!

- ✓ 모든 소녀들은 현장에 2시 이전에 나타났다가 2시 이후에 사라집니다.
- ✓ 즉, 현장에 있던 소녀들의 인원수는 시간에 따라 2시까지 증가했다가 2시 이후로 감소합니다.
- ✓ 따라서 어떤 소녀가 특정 시각에 혼자 있었다면, 그 소녀는 가장 일찍 나타나거나 가장 늦게 사라진 소녀여야 합니다.
- ✓ 2시 이전에 가장 일찍 나타난 소녀를 생각해 봅시다.  $i$ 번 소녀가 나타난 시각을  $L_i$ 라고 하면, 각 쿼리는 다음과 같이 처리할 수 있습니다.
  - ✓ 1  $i\ l\ r$ :  $L_i = l$ 로 설정하고, 집합  $S$ 에  $(L_i, i)$ 를 넣는다.
  - ✓ 2  $i$ : 집합  $S$ 에서  $(L_i, i)$ 를 제거한다.
  - ✓ 3: 집합  $S$ 의 가장 작은 두 값  $(L_i, i)$ 와  $(L_j, j)$  ( $L_i \leq L_j$ )를 뽑아서 확인한다.  $L_i < L_j$ 이면  $i$ 번 소녀는 시각  $L_i$ 에 혼자 있었다.

## E. 2시에 혼자 있었다니, 이상해!

- ✓ 이러한 문제에 빠르게 답하려면 집합에 원소를 삽입하고, 원소를 제거하고, 가장 작은 값을 찾는 연산을 빠르게 할 수 있는 자료 구조가 필요합니다.
- ✓ C++의 `std::set`을 사용하면 삽입과 삭제는  $\mathcal{O}(\log N)$ , 최솟값 쿼리는  $\mathcal{O}(1)$ 에 처리할 수 있습니다. Iterator를 사용하면 두 번째 최솟값도 편리하게 찾을 수 있습니다.
- ✓ 2시 이후에 가장 늦게 사라진 소녀에 대한 쿼리는, 가장 작은 값 대신 가장 큰 값을 이용하면 대칭적으로 풀 수 있습니다.
- ✓ 현장의 소녀가 1명 이하인 경우에 대한 예외 처리와, 2시 이전과 이후의 답이 같은 경우 중복 제거를 유의해 주면 정답을 받을 수 있습니다.
- ✓ 한 쿼리에서 혼자 있는 소녀는 최대 2명이므로, 출력하는 소녀의 수는  $2Q$  이하입니다. 따라서  $k$  합에 대한 제한은 사실 의미가 없습니다.
- ✓ 시각의 값은 중요하지 않고 순서만 중요하기 때문에, 입력을 정수로 받은 후 220000 이상일 때만 240000을 빼줘도 됩니다.

## E. 2시에 혼자 있었다니, 이상해!

- ✓ Python으로도 해결할 수 있을까요?
- ✓ 우선순위 큐 (heapq)를 사용하면 삽입(push)과 최솟값 제거(pop) 쿼리는  $\mathcal{O}(\log N)$ 에 쉽게 구현할 수 있습니다.
- ✓ 삭제 연산의 경우, 다음과 같이 구현하는テクニック이 있습니다. 흔히 Lazy deletion이라고 불리는 기법입니다. Python으로 살아남기 위해서는 필요한 일이 자주 있을테니 알아두시다.
- ✓ 두 우선순위 큐  $S$ 와  $T$ 를 준비합니다.  $T$ 는 삭제할 원소들을 임시로 담고 있는 목적으로 사용할 것입니다.
- ✓ flush 연산을 다음과 같이 정의합니다.
  - ✓ flush:  $S$ 의 최솟값과  $T$ 의 최솟값이 달라질 때까지  $S$ 와  $T$  모두에서 최솟값을 pop합니다.
  - ✓ 즉, “삭제로 표시된 원소”를 지울 수 있다면 모두 지우는 작업입니다.

## E. 2시에 혼자 있었다니, 이상해!

- ✓ 이 연산을 이용하면 필요한 3가지 쿼리를 모두 처리할 수 있습니다.
  - ✓  $x$  삽입: flush 후,  $S$ 에  $x$ 를 push합니다.
  - ✓  $x$  제거:  $T$ 에  $x$ 를 삽입합니다.
  - ✓ 최솟값 제거: flush 후,  $S$ 에서 pop합니다.
- ✓ flush를 제외하면 각 쿼리의 시간복잡도는  $\mathcal{O}(\log N)$ 입니다.
- ✓ flush에서 pop이 수행되는 횟수는 제거 쿼리의 수보다 많을 수 없으므로, flush의 총 시간복잡도는  $\mathcal{O}(Q \log N)$ 입니다. 따라서 모든 연산은 amortized  $\mathcal{O}(\log N)$ 에 동작합니다.
- ✓ 그 외에도, 최솟값 세그먼트 트리 또는 비트셋 등의 다양한 자료구조를 활용하여 해결하는 방법이 존재합니다.

---

## F. 나를 봐줘!

#dp

출제진 의도 - Hard

✓ 출제자: ThatOnePythonUser

✓ 제출: 54번, 정답: 40명 (정답률 74.07%), 처음 푼 사람: middle\_man, 11분

## F. 나를 봐줘!

- ✓ 자신이 과거에 출연했던 가장 최근의 방송을  $i$ 번째, 현재 방송을  $j$ 번째라고 합시다. 단, 현재 방송이 첫 출연일 경우  $i = 0$ 으로 취급합니다.
- ✓ 레이아가 이 방송에서 받는 스트레스를  $L(i, j)$ , 미리아가 받는 스트레스를  $M(i, j)$ 라고 하면, 아래와 같이 표현할 수 있습니다.

$$✓ L(i, j) = \begin{cases} 0 & (i = 0) \\ \max(0, C_i - C_j) & (i > 0) \end{cases}$$

$$✓ M(i, j) = \begin{cases} 0 & (i = 0) \\ \max(0, C_j - C_i) & (i > 0) \end{cases}$$

## F. 나를 봐줘!

- ✓ 최소 스트레스 합을 구하기 위해 다이나믹 프로그래밍(DP)을 사용할 수 있습니다.
- ✓ 다양한 상태 정의 방법으로 풀 수 있지만, 가장 간단한 한 가지를 소개합니다.
- ✓ 다음과 같이 DP 상태를 정의합니다.
  - ✓  $dp[i][j]$  = 레이아의 가장 최근 방송이  $i$ 번째, 미리아의 가장 최근 방송이  $j$ 번째일 때 지금까지 받은 스트레스 합의 최솟값.
  - ✓  $i > j$ 이면 현재 레이아가,  $i < j$ 이면 현재 미리아가 방송을 하고 있는 것입니다. 아직 방송을 하지 않았다면 0번째로 간주합니다.
- ✓ 첫 방송은 누가 출연하든 스트레스가 0이므로,  $dp[1][0] = dp[0][1] = 0$ 으로 초기화 해줍니다.

## F. 나를 봐줘!

- ✓ 상태의 전이는 다음과 같이 바텀업 방식으로 계산할 수 있습니다.
- ✓ 현재 레이아와 미리아의 최근 방송이  $(i, j)$ 번째라면, 다음 방송은  $k = \max(i, j) + 1$ 번째 방송입니다.
- ✓ 레이아가 다음 방송을 할 경우,  $dp[k][j] \leftarrow \min(dp[k][j], dp[i][j] + L(i, k))$ 로 업데이트 해줍니다.
- ✓ 미리아가 다음 방송을 할 경우,  $dp[i][k] \leftarrow \min(dp[i][k], dp[i][j] + M(j, k))$ 로 업데이트 해줍니다.
- ✓ 결과적으로, 마지막 방송이  $N$ 번째가 되는  $\min_{0 \leq k < N} \min(dp[k][N], dp[N][k])$ 이 답이 됩니다.
- ✓ DP 전이를 그대로 구현하면  $\mathcal{O}(N^2)$ 에 해결할 수 있습니다.

## F. 나를 봐줘!

- ✓ DP 상태를 조금 다른 방법으로 정의하는 풀이도 가능합니다.
- ✓  $dp[i][j]$  = 레이아의 가장 최근 방송이  $i$ 번째, 미리아의 가장 최근 방송이  $j$ 번째일 때 **앞으로 받을** 스트레스 합의 최솟값으로 정의합니다.
- ✓ 초기 조건은  $i = N$  또는  $j = N$ 이면  $dp[i][j] = 0$ 입니다.
- ✓  $k = \max(i, j) + 1$ 이라고 하면,  $k$ 번째 방송을 누가 할 것이냐에 따라 전이는  $dp[i][j] = \min(L(i, k) + dp[k][j], M(j, k) + dp[i][k])$ 가 됩니다.
- ✓ 답은 첫 방송을 모두 고려하면  $\min(dp[1][0], dp[0][1])$ 입니다.
- ✓ 이 문제에서 사용한 것과 거의 똑같은 테크닉으로 **바이토닉 투어**를 해결할 수 있습니다.

---

## G. fiXmArge

#string, #trie, #hashing

출제진 의도 - Hard

✓ 출제자: ThatOnePythonUser

✓ 제출: 52번, 정답: 26명 (정답률 50.00%), 처음 푼 사람: is\_it\_interactive, 28분

## G. fiXmArge

- ✓ 픽스마지어의 대소문자는 구분하지 않으므로 먼저 모두 소문자로 바꾼 후 시작합니다.
- ✓ 문자열  $S$ 에 대해,  $S[a : b]$ 를  $a - 1$ 번째 글자부터  $b$ 번째 글자까지로 구성된 부분문자열이라고 정의합니다.
- ✓ 길이  $n$ 인 픽스마지어  $F$ 가 해독 가능하려면, 어떤  $0 \leq k \leq n$ 에 대해서 다음 중 하나가 성립해야 합니다.
  - ✓  $F[0 : k]$ 가 어떤 영어 단어의 접두사,  $F[k : n]$ 이 어떤 일본어 단어의 접미사이다.
  - ✓  $F[0 : k]$ 가 어떤 일본어 단어의 접두사,  $F[k : n]$ 이 어떤 영어 단어의 접미사이다.
- ✓ 두 번째 경우는 영어와 일본어를 바꿔서 풀면 되므로, 첫 번째 경우만 해결해도 충분합니다.
- ✓ 대표적인 문자열 알고리즘을 활용한 2가지 방법을 생각할 수 있습니다.

## G. fiXmArge

### 1. 트라이(Trie)를 사용한 풀이

- ✓ 픽스마지어 단어  $F$ 에 대해, 아래 두 값을 생각해 봅시다.
  - ✓  $F$ 와 어떤 영어 단어의 공통 접두사의 최대 길이  $a$
  - ✓  $F$ 와 어떤 일본어 단어의 공통 접미사의 최대 길이  $b$
- ✓  $F[0 : a]$ 가 영어 단어  $E_i$ 의 접두사이면, 모든  $0 \leq k \leq a$ 에 대해  $F[0 : k]$ 도  $E_i$ 의 접두사입니다. 접미사도 마찬가지로  $0 \leq n - k \leq b$ 이 성립해야 합니다.
- ✓ 두 부등식을 더하면  $a + b \geq n$ 일 때만 단어를 해독할 수 있습니다.
- ✓ 공통 접두사의 최대 길이  $a$ 를 구하기 위해 트라이를 사용할 수 있습니다. 먼저 모든 영어 단어를 트라이에 삽입합니다.
- ✓ 트라이의 루트에서부터  $F$ 와 일치하는 알파벳을 따라갑니다. 더 이상 갈 수 없을 때 노드의 깊이가  $a$ 와 같습니다.

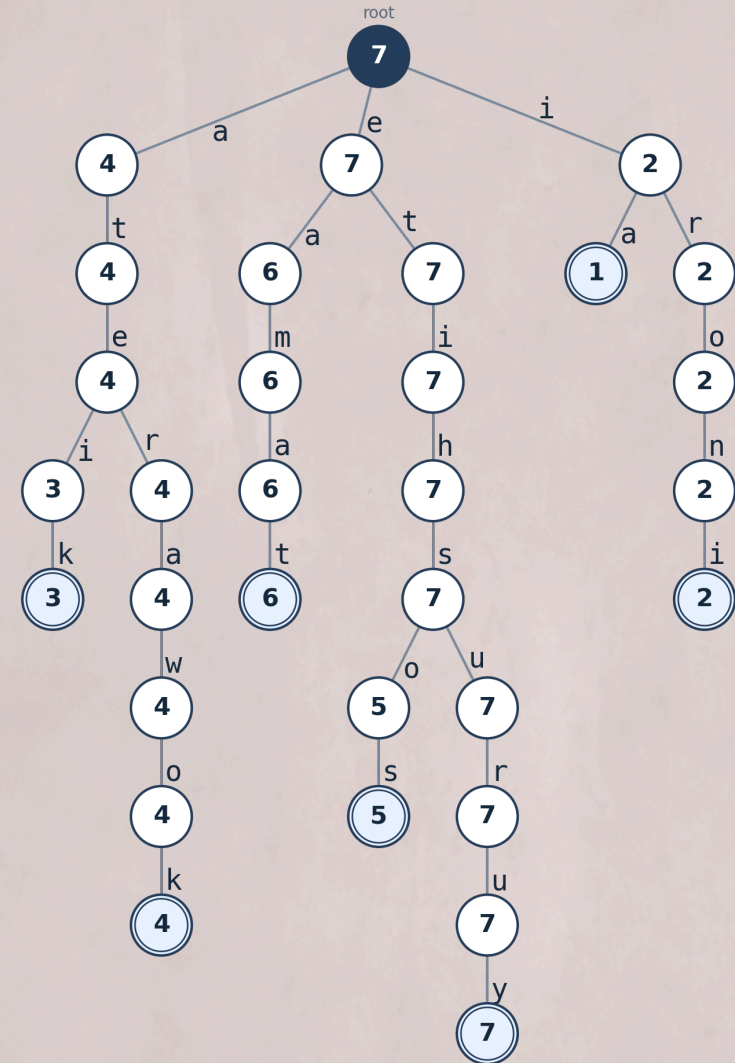
## G. fiXmArge

- ✓ 문자열  $S$ 를 뒤집은 문자열을  $\text{rev}(S)$ 라고 합시다.  $F$ 와 일본어 단어  $J_i$ 의 공통 접미사의 최대 길이는  $\text{rev}(F)$ 와  $\text{rev}(J_i)$ 의 공통 접두사의 최대 길이와 같습니다.
- ✓ 같은 방법으로, 일본어 단어를 모두 뒤집어서 트라이에 삽입한 후  $\text{rev}(F)$ 를 검색하면 공통 접미사의 최대 길이  $b$ 를 구할 수 있습니다.
- ✓ 트라이에 단어를 삽입할 때, 해당 단어의 인덱스  $i$ 를 트라이의 노드에 기록합니다. 이미 기록되어 있는 경우는 덮어쓰워도 괜찮습니다.
- ✓ 가장 긴 공통 접두사의 인덱스가  $i$ 라면, 그것보다 짧은 접두사도  $i$ 번째 단어의 접두사이므로, 인덱스도  $i$ 를 출력하면 됩니다.
- ✓ 최대 노드 개수가 약  $T = 10^5$ 인 트라이를 4개 ( $E, \text{rev}(E), J, \text{rev}(J)$ ) 만들고 문자열  $F$ 와  $\text{rev}(F)$ 를 검색하려면  $\mathcal{O}(4T + 2T)$  시간과  $\mathcal{O}(4T)$  공간을 사용합니다. 단, 트라이의 특성상 시간과 메모리에 알파벳 개수에 해당하는 26배 정도의 큰 상수가 붙을 수 있습니다.

## G. fiXmArge

- ✓ 오른쪽은 예제 2번의 일본어 단어를 뒤집어서 구성한 접미사 트라이의 예시입니다.
- ✓ 삽입된 인덱스와 단어는 아래와 같습니다.

1. ai
2. inori
3. kieta
4. kowareta
5. soshite
6. tamae
7. yurushite



## G. fiXmArge

### 2. 해싱을 사용한 풀이

- ✓ 다음과 같은 풀이를 생각해 봅시다.
  - ✓ 먼저 모든 영어 단어와 일본어 단어의 접두사와 접미사를 key로 하고, 인덱스를 value로 하는 map을 만듭니다.
  - ✓ 길이  $n$ 의 픽스마지어 단어  $F$ 가 주어지면, 모든  $0 \leq k \leq n$ 에 대해  $F[0 : k]$ 와  $F[k : n]$ 이 map에 있는지 찾아서 출력합니다.
- ✓ 위 풀이는 map에 최대 길이  $n = 10^5$ 인 문자열의 접두사와 접미사가 들어갈 경우  $\mathcal{O}(n^2)$ 의 시간과 메모리를 소요하기 때문에 제한 안에서 풀 수 없습니다.
- ✓ 이때, 모든 접두사와 접미사를 해싱하여 정수 하나로 취급할 수 있다면 유효한 풀이가 될 것입니다.

## G. fiXmArge

- ✓ 어떤 문자열  $S = c_1c_2...c_n$ 의 해시  $H$ 란, 간단히 말하면  $c_1c_2...c_n$ 을  $\text{mod } p$ 에서  $b$ 진 법으로 읽은 수라고 할 수 있습니다.  $b$ 는 적당히 큰 아무 수나 사용해도 됩니다.
- ✓ 즉,  $H(S) = \left(\sum_i c_i b^{n-i}\right) \text{mod } p$ 로 정의합니다.  $c_i$ 는 알파벳의 아스키 코드를 사용합니다.
- ✓  $p$ 가 충분히 큰 소수라면, 서로 다른 두 문자열  $S$ 와  $T$ 의 해시  $H(S)$ 와  $H(T)$ 가 겹칠 확률은 매우 낮습니다. 따라서  $H(S) = H(T)$ 이면  $S = T$ 인 것처럼 간주할 수 있습니다.
- ✓ 일반적으로,  $p \lesssim N^2$ 이면  $N$ 개의 해시가 서로 겹칠 확률이 매우 높아진다는 것이 알려져 있습니다. 특히  $p = 998244353$ 을 사용하면 거의 확실히 오답을 받게 됩니다.
- ✓ 따라서 이 문제에서는 약  $10^{15}$  이상의 큰 소수를 사용하거나, 여러 가지  $b$ 값으로 다중 해싱을 하는 것이 안전합니다. 잘 알려진 소수로는  $1 < 57 | 1557$ 이 있습니다.

## G. fiXmArge

- ✓ 접두사  $S[0 : k]$ 의 해시를  $A_k$ , 접미사  $S[k : n]$ 의 해시를  $B_k$ 라고 합시다. 문자열의  $b$ 진법 표현을 생각하면, 아래 식이 성립하는 것을 알 수 있습니다.
  - ✓  $A_0 = B_n = 0$
  - ✓  $A_{k+1} = (A_k b + c_k) \bmod p$
  - ✓  $B_k = (c_k b^{n-1-k} + B_{k+1}) \bmod p$
- ✓  $b$ 의 거듭제곱을 전처리하면, 각 단어  $S$ 의 접두사와 접미사 해시를  $\mathcal{O}(|S|)$ 에 모두 구할 수 있습니다.
- ✓ 이 방법으로 문자열의 해시를 통해 map의 key를 저장 및 조회한다면, 모든 단어에 대해  $\mathcal{O}(\sum |S|)$ 에 해결할 수 있습니다.

---

## H. 붉은 나비

#graphs, #combinatorics

출제진 의도 - Challenging

✓ 출제자: ThatOnePythonUser

✓ 제출: 33번, 정답: 18명 (정답률 54.55%), 처음 푼 사람: saturnalorbit, 15분

## H. 붉은 나비

- ✓ 그래프에서 세 정점이 붉게 물든 삼각형을 **붉은 삼각형**이라고 합시다.
- ✓ 그러면 붉은 나비는 아래 조건을 만족하는 부분그래프입니다.
  - ✓ 한 정점을 공유하는 서로 다른 두 개의 붉은 삼각형으로 이루어져 있다.
  - ✓ 단, 한 간선을 공유하지는 않는다.
- ✓ 이때 붉은 삼각형이 공유하는 정점을 붉은 나비의 **중심 정점**이라고 합시다. 붉은 나비를 세기 위해 다음 두 값을 관리합니다.
  - ✓  $\triangle_u$ : 정점  $u$ 를 포함하는 붉은 삼각형의 개수
  - ✓  ${}_u\triangle_v$ : 간선  $uv$ 를 포함하는 붉은 삼각형의 개수

## H. 붉은 나비

- ✓ 중심 정점이  $u$ 인 붉은 나비의 개수를  $\bowtie_u$ 라고 합시다. 위 두 조건을 고려하여 식으로 정리하면 다음과 같습니다.

$$\begin{aligned}\bowtie_u &= \left( \bowtie_u + \sum_{v \in \text{adj}(u)} u \square^v \right) - \left( \sum_{v \in \text{adj}(u)} u \square^v \right) \\ &= \binom{\Delta_u}{2} - \sum_{v \in \text{adj}(u)} \binom{u \Delta_v}{2}\end{aligned}$$

- ✓ 따라서, 총 나비의 개수  $\bowtie$ 는 다음과 같습니다.

$$\bowtie = \sum_{u \in V} \binom{\Delta_u}{2} - \sum_{(u,v) \in E} \binom{u \Delta_v}{2}$$

- ✓ 단, 간선의 집합은  $(u, v)$ 와  $(v, u)$ 를 모두 포함하는 것으로 간주합니다.

## H. 붉은 나비

- ✓ 정점  $u$ 가 붉게 칠해지면,  $u$ 를 포함하는 붉은 삼각형  $uvw$ 가 만들어졌는지  $u$ 에 인접한 모든  $(v, w)$  쌍을  $\mathcal{O}((\deg u)^2)$ 에 확인할 수 있습니다.
- ✓ 인접 행렬  $\text{adj}[v][w]$ 에 간선이 있으며,  $v$ 와  $w$ 가 붉게 물들었는지 확인하면 됩니다.
- ✓ 붉은 삼각형  $uvw$ 가 만들어졌다면, 다음 값이 모두 1 증가합니다.
  - ✓  $\triangle_u, \triangle_v, \triangle_w, u\triangle_v, v\triangle_u, v\triangle_w, w\triangle_v, w\triangle_u, u\triangle_w$
- ✓  $\binom{\triangle+1}{2} - \binom{\triangle}{2} = \triangle$ 임에 주목하면, 나비 개수의 계산에 필요한  $\binom{\triangle}{2}$  꼴의 값들은 모두  $\triangle$ 만큼 증가합니다.
- ✓ 따라서 정점  $u$ 가 칠해질 때마다  $\bowtie$ 의 변화량을 더해주면 붉은 나비의 개수를  $\mathcal{O}((\deg u)^2)$ 에 계산할 수 있고, 총 시간복잡도는  $\mathcal{O}(\sum_u (\deg u)^2)$ 입니다.
- ✓ 모든 정점에 대해  $\deg u \leq N$ 이므로,  $\sum_u (\deg u)^2 \leq N \sum_u \deg u = 2NM$ 입니다. 따라서  $\mathcal{O}(NM)$  시간과  $\mathcal{O}(N^2)$  메모리로 해결할 수 있습니다.

## H. 붉은 나비

- ✓  $\mathcal{O}(NM)$ 보다 빠르게 해결하는 것도 가능합니다.
- ✓ 모든 간선  $uv$ 에  $\deg(u) \leq \deg(v)$ 가 되도록 방향을 부여하고, 모든 간선  $u \rightarrow v$ 를 순회합니다. 차수가 같으면 순서를 임의로 정합니다.
- ✓  $v$ 에서 나가는 정점  $w$ 를 순회하면서,  $w$ 와  $u$ 가 인접하면  $uvw$ 는 삼각형입니다.
  - ✓ 항상 정점의 순서가  $\deg(u) \leq \deg(v) \leq \deg(w)$ 로 정해진 삼각형만 세어지므로, 삼각형은 중복되지 않습니다.
- ✓ 반복문의 순회 횟수는  $\sum_{u \rightarrow v} \deg^+(v)$ 입니다. 이때  $\deg^+(v)$ 는  $v$ 의 진출차수(outdegree)입니다. 이때,  $\deg^+(v) \leq \sqrt{2M}$ 임을 증명할 수 있습니다.
  - ✓  $\deg^+(v) > \sqrt{2M}$ 을 가정하면,  $\deg(w) \geq \deg(v) \geq \deg^+(v) > \sqrt{2M}$ 입니다. 그런데  $v$ 에 인접한  $w$ 가  $\sqrt{2M}$ 개보다 많으면  $\sum \deg(w) \leq 2M$ 에 모순입니다.

## H. 붉은 나비

- ✓ 따라서, 삼각형의 수는  $\sum_{u \rightarrow v} \mathcal{O}(\sqrt{M}) = \mathcal{O}(M\sqrt{M})$ 입니다.
- ✓ 이 방법으로  $u$ 에 인접한 삼각형의 목록을  $\mathcal{O}(M\sqrt{M})$ 에 전처리할 수 있습니다.
- ✓ 이후 정해와 같은 방법으로 해결하면 시간 복잡도는  $\mathcal{O}(M\sqrt{M})$ 이며, 구현 방법에 따라  $\mathcal{O}(N^2)$ 이 추가될 수 있습니다.



모두 수고하셨습니다!