아니메컵 2쿨 풀이

Official Editorial

by

아니메컵 제작위원회

2023년 2월 11일

문제		의도한 난이도	출제자
A	Gorani Command	**Easy**	ruykun
B	랩실에서 잘 자요	**Easy**	ruykun
C	주문은 토기입니까?	**Medium**	dhyang24
D	체인소 맨	**Medium**	ruykun
E	한별이 드롭킥!	**Medium**	dhyang24
F	플래그 대사 그만 좀 말해요	**Medium**	dhyang24
G	마키마씨가 정해주는 오늘 점심의 맛	**Medium**	aaat

문제		의도한 난이도	출제자
H	(재밌고 웃기고 센스있고 깔끔한 제목)	Hard	kirvy810
I	귀엽기만 한 게 아닌 한별 양	Hard	sprout_429
J	엔드롤이 끝나고	Hard	ruykun
K	카드캡터 한별	Hard	sivcde0405
L	치노와 코코아	Challenging	ibm2006
M	THE iDEM@STER	Challenging	dhyang24

2A. Gorani Command

`implementation`
출제진 의도 – **Easy**

- 제출 638번, 정답 389명 (정답률 63.009%)
- 처음 푼 사람: **ychangseok**, 2분
- 출제자: `ruykun`

- 도망친 고라니는 세로 N, 가로 M의 직사각형 영역에 존재하므로, 숨어있는 칸의 좌표를 (r, c)라 할 때 가능한 (r, c)의 쌍은 NM가지입니다.
- 모든 가능한 경우에 대해 'ㄴ' 모양으로 있는 각 칸과의 거리를 구해, 입력과 일치하는지 비교할 경우 $O(NM(N+M))$의 시간복잡도가 나옵니다.
- N, M이 최대 50이므로 이 방법으로도 충분히 빠르게 문제를 해결할 수 있습니다.

- 도망친 고라니가 숨어있는 칸에서 'ㄴ' 모양을 이루는 세로줄과 가로줄까지 도착하기 위한 가장 짧은 경로는 각각 왼쪽과 아래를 향해 일직선으로 이동하는 것입니다.
- 세로줄의 최솟값을 가진 칸과 가로줄에서 최솟값을 가진 칸만 알아도 도망친 고라니가 숨어있는 좌표를 알 수 있습니다.

– 각 줄의 최솟값을 가진 칸으로부터 도망친 고라니가 숨어있는 좌표를 구하기 위한 대표적인 방법으로 다음의 세 가지 방법이 있고, 어떤 방법을 사용하더라도 정답을 구할 수 있습니다.

- 세로줄에서 a 번째 수가 가장 작고, 가로줄에서 b 번째 수가 가장 작다면 도망친 고라니는 (a, b) 에 숨어있습니다.
- 세로줄에서 가장 작은 수가 v 고, 가로줄에서 가장 작은 수가 w 라면 도망친 고라니는 $(N - w, v + 1)$ 에 숨어있습니다.
- 세로줄에서 a 번째 수가 가장 작고, 그 수가 v 라면 도망친 고라니는 $(a, v + 1)$ 에 숨어있습니다.

- 위쪽 두 방법은 세로줄과 가로줄을 전부 확인하므로 $O(N+M)$, 마지막 방법은 세로줄만을 확인하므로 $O(N)$의 시간복잡도로 문제를 해결할 수 있습니다.

2B. 랩실에서 잘 자요

`greedy`
출제진 의도 – **Easy**

- 제출 376번, 정답 127명 (정답률 35.372%)
- 처음 푼 사람: **asdf1705**, 6분
- 출제자: `ruykun`

2B. 랩실에서 잘 자요

- 1 부터 N 까지의 각 페이지들은 존재하거나, 빠져있는 둘 중 하나의 상태를 가집니다.
- 입력에서 한 번 이상 주어진 페이지는 이미 인쇄된 페이지이고, 그 외의 페이지는 빠진 페이지가 됩니다.
- 빠진 페이지를 전부 인쇄하면서 소모하는 잉크의 합계를 최소화합니다.

2B. 랩실에서 잘 자요

- 한 번에 인쇄할 연속된 페이지를 하나의 구간으로 정합시다.
- 빠진 페이지는 전부 인쇄되어야 하므로 모든 빠진 페이지는 어떤 구간에 속해야만 합니다.
- 어떤 구간의 시작과 끝은 전부 빠진 페이지가 됩니다. 그렇지 않을 경우 이미 인쇄된 페이지를 구간 끝에서 제외해 잉크 소모를 줄일 수 있습니다.

- 어떤 페이지가 둘 이상의 구간에 속할 경우, 그 페이지가 하나의 구간에만 속하도록 하는 것이 반드시 잉크를 적게 소모합니다.
- 두 구간 사이에 다른 구간이 없고, 사이에 이미 인쇄된 페이지가 a $(0 \leq a \leq 2)$장 있을 경우, 두 구간을 하나로 합쳐 한 번에 인쇄하는 것이 잉크를 $5 - 2a$ 적게 소모합니다.
- $a \geq 3$일 경우 소모되는 잉크가 오히려 늘어납니다.

2B. 랩실에서 잘 자요

- 첫 페이지부터 마지막 페이지까지 각 빠진 페이지 사이에 오는 이미 인쇄된 페이지의 개수를 셉니다.
- 첫 빠진 페이지 이전의 페이지와 마지막 빠진 페이지 이후의 이미 인쇄된 페이지는 인쇄하지 않도록 합니다.
- 두 빠진 페이지 사이에 3장 이상의 이미 인쇄된 페이지가 있을 경우 그 페이지도 인쇄하지 않도록 합니다.

- 어떤 페이지가 빠졌는지 확인하는 것이 $\mathcal{O}(N)$, 어떤 페이지를 인쇄해야 하는지 확인하는 것이 $\mathcal{O}(N)$으로 알고리즘 전체의 시간복잡도는 $\mathcal{O}(N)$이 됩니다.

2C. 주문은 토기입니까?

`greedy`
출제진 의도 – **Medium**

- 제출 274번, 정답 61명 (정답률 22.993%)
- 처음 푼 사람: **evenharder**, 19분
- 출제자: dhyang24

2C. 주문은 토기입니까?

- 정수 시간에만 행동을 시작하는 것으로 최적해를 얻을 수 있습니다.
- 손님이 오는 시간에는 무조건 커피를 서빙해야 합니다.
- 만약 토기에 커피를 담아도 그 커피를 서빙하기 전에 커피가 흙탕물이 되지 않는다면, 항상 커피를 토기에 담는 것이 좋습니다.
- 위 두 상황이 아니라면, 토기를 만드는 것이 아무것도 하지 않는 것보다 항상 이득입니다.

2C. 주문은 토기입니까?

- 어떤 손님을 위한 커피를 만들 수 있으면(즉, 손님이 오기 전에 그 손님을 위한 토기를 만들고 그 토기에 커피를 담을 수 있으면) 그 손님에게 서빙할 수 있으므로, 현재 서빙해야 하는 손님을 저장해 놓고 커피를 만들 때마다 다음 손님을 불러오는 식으로 구현할 수 있습니다.
- 만약 현재 서빙해야 하는 손님이 오는 시간이 현재 시간+커피가 흙탕물이 되기까지 걸리는 시간 이전이며, 만들어진 토기가 1개 이상 있는 경우, 만들어진 토기 1개에 커피를 담는것이 이득입니다.

2C. 주문은 토기입니까?

- 시뮬레이션 중 어떤 상황에서라도 현재 서빙해야 하는 손님이 오는 시간이 현재 시간과 같거나 더 이전이면 한별이가 가게를 지켜낼 수 없으므로 프로그램을 종료합니다.
- 시간 순서대로 이동하며 동작을 $O(1)$에 결정하므로, 총 $O(N)$ 안에 풀 수 있습니다.
- 시간 역순으로 푸는 풀이도 존재합니다.

2D. 체인소 맨

`implementation`
출제진 의도 – **Medium**

- 제출 49번, 정답 27명 (정답률 55.102%)
- 처음 푼 사람: **imkavin43**, 40분
- 출제자: `ruykun`

- 나무를 자를 때는 격자를 따라 잘라야 하므로, 작업 영역의 경계를 제외한 격자를 이루는 $N-1$ 개의 가로줄과 $M-1$ 개의 세로줄이 있다면, 나무를 자르는 선은 전부 이 가로줄과 세로줄에 포함됩니다.
- 서로 다른 가로줄 또는 세로줄을 자르는 것을 독립이며, 어떤 줄에 포함되게 자르는 것은 다른 줄의 어떠한 요소에도 영향을 주지 않습니다.

- 각 가로줄은 길이가 1인 선분 M개로 나눌 수 있고, 각 세로줄은 길이가 1인 선분 N개로 나눌 수 있습니다.
- 각 선분은 다음 3가지 중 하나로 분류할 수 있습니다.
  - 두 개의 '#' 사이에 있는 선분은 절대로 잘라서는 안됩니다.
  - '#'와 '.' 사이에 있는 선분은 반드시 잘라야만 합니다.
  - 두 개의 '.' 사이에 있는 선분은 잘라도 자르지 않아도 상관 없습니다.
- 앞으로 위의 3가지 선분을 순서대로 A, B, C로 칭합니다.

- 첫 번째 방법으로 잘라 만든 절단면은 언제나 두 번째 방법으로 자르는 것으로도 만들 수 있습니다.
- 첫 번째 방법으로 자르는 것이 두 번째 방법으로 같은 절단면을 만드는 것보다 힘을 덜 소모하는 경우가 있으므로 어떤 상황에서 첫 번째 방법으로 자를 수 있는지, 또 어떤 상황에서 그렇게 자르는 것이 이득인지 알아봅시다.

- 어떤 B가 같은 줄에 있는 두 A를 잇는 선분에 포함된다면, 다시 말해 두 A 사이에 존재한다면 그 B를 포함해서 첫 번째 방법으로 자르는 것은 불가능합니다.
- 위 조건에 해당되지 않으며 한 번도 자르지 않은 모든 B에 대해, 해당 B를 포함하며 어떤 A도 포함하지 않게 첫 번째 방법으로 최대 몇 개의 B를 자를 수 있는 지 확인합니다. 자르는 반직선이 반드시 해당 B에서 시작할 필요는 없으므로 해당 B에 대해 한쪽 방향만 확인하지 않도록 주의합니다.
- 그 방법으로 F를 초과한 개수의 B를 자를 수 있다면 첫 번째 방법으로 자르는 것이 이득이므로 첫 번째 방법으로 자르도록 합니다.

- 첫 번째 방법으로 자르는 것이 불가능하거나 이득이 되지 않는 모든 B를 두 번째 방법으로 자릅니다.
- 두 번째 방법으로 C를 자를 경우 불필요한 힘이 소모되므로 잘라서는 안됩니다.
- 첫 번째와 두 번째 방법으로 자르는 모두에 대해 하나의 B를 두 번 이상 세지 않도록 주의합니다.

- $N-1$ 개의 가로줄에 대해 각각 M 개의 선분을 확인하고, $M-1$ 개의 세로줄에 대해 각각 N 개의 선분을 확인해야 합니다.
- 각 선분은 한 번 확인한 후에는 다시 확인하지 않으므로 시간복잡도는 $O(NM)$ 이 됩니다.

2E. 한별이 드롭킥!

dp
출제진 의도 – **Medium**

- 제출 78번, 정답 13명 (정답률 16.667%)
- 처음 푼 사람: **pichulia**, 51분
- 출제자: dhyang24

2E. 한별이 드롭킥!

- 우선 각각의 상승기류들을 한별이의 집을 기준으로 왼쪽의 집들과 오른쪽의 집들로 나누고, 왼쪽의 집들은 내림차순, 오른쪽의 집들은 오름차순으로 위치를 기준으로 정렬해 줍니다.
- 3차원 DP 테이블 DP[s][i][j]를 s는 한별이가 왼쪽/오른쪽에 있는지의 여부, i는 방문한 가장 왼쪽 기류의 인덱스, j는 방문한 가장 오른쪽 기류의 인덱스로 정의한 다음 DP[s][i][j] = 한별이가 왼쪽 또는 오른쪽에 있고 i j번 상승기류들을 전부 사용한 후 가능한 높이중 최대높이로 정의할 수 있습니다.
- 한 상태 내에서, 현재 높이가 가장 높을 수록 이득입니다. 기류를 모두 사용한 후의 총 높이는 정해져 있기 때문에, 다른 기류를 사용할 수 있는지에 직접적인 영향을 주는 현재 높이를 최대화하여야 합니다.

2E. 한별이 드롭킥!

- 편의상, 집을 세기가 0인 상승기류로 두고, [left][0][0]과 [right][0][0]의 값을 한별이의 최초 높이로 세팅하여 dp테이블을 채워나갑니다.
- 예를 들어, 한별이가 [left][i][j]에 있는 경우, 한별이는 [left][i+1][j] 또는 [right][i][j+1] 로 이동할 수 있습니다. 한별이가 i+1 번째 상승기류로 이동하기 위해서는 i번쨰 상승기류와 1+1 번째 상승기류 사이의 거리가 현재 높이보다 작아야 합니다.
- 반대로, 오른쪽으로 이동할 때에는 i번째 상승기류와 j+1 번째 상승기류 사이의 거리가 현재 높이보다 작아야 합니다. 상승기류에 갈 수 있다는 것이 판별된 이후에는 그 상승기류의 세기와 현재 높이를 합해, 원래 있던 값보다 더 크다면 dp를 업데이트합니다.

- 같은 방식으로, 한별이가 오른쪽에 있는 경우도 처리할 수 있습니다. 이 방법을 사용했을 경우, 총 dp테이블의 크기는 (한별이 집 왼쪽의 상승기류)*(한별이 집 오른쪽의 상승기류)와 같으므로 작은 상수의 $O(N^2)$ 시간 안에 작동한다는 것을 알 수 있습니다.
- challenge: 여러가지 그림을 그리면서 시도해 보면 최적해의 방향 전환 횟수의 상한선이 그렇게 크지 않다는 것을 알 수 있습니다. 이를 이용해서 $O(N^2)$ 보다 빠른 솔루션을 만들 수 있을까요?

- 여담으로, 한별이가 아무리 강한 드롭킥에 성공한다고 하더라도 지상에는 Farmer John이 Chain-cow를 데리고 대기하고 있기 때문에, 한별이의 미래는 불투명합니다....

2F. 플래그 대사 그만 좀 말해요

`greedy, implementation`
출제진 의도 – **Medium**

- 제출 92번, 정답 23명 (정답률 25.000%)
- 처음 푼 사람: **yooyou7**, 40분
- 출제자: dhyang24

- 한별이의 왼쪽에 있는 부하들은 한별이의 대사를 듣지 못하므로, 한별이가 대사를 외칠때, 목표 강함 수치를 달성하지 못한 부하들 중 가장 왼쪽에 있는 부하의 자리에서 대사를 하는 것이 최적임을 알 수 있습니다.
- 그렇지만, 한별이가 대사를 할 때마다 바뀐 부하들의 강함 수치를 모두 계산한다면 시간 안에 문제를 풀 수 없습니다.
- 때문에, 대사를 여러번 외친 후 부하의 강함 수치를 빠르게 계산하는 방법을 찾아야 합니다.

2F. 플래그 대사 그만 좀 말해요

- 어떤 칸에서 대사 없이 늘어나는 강함 수치의 양은 (바로 왼쪽 칸에서 대사 없이 늘어나는 강함 수치의 양)-(왼쪽 칸에서 적용되고 있던, 세기가 1 이상인 대사의 개수) 입니다.
- 이 방법을 사용한다면, 현재 적용되는 대사의 양만 아는 상태로 반드시 해야 하는 대사의 횟수를 쉽게 구할 수 있습니다.
- 어떤 칸에서 대사를 한다면, 그 다음 칸에 대사 없이 늘어나는 강함 수치의 양을 (K-1)*(대사 횟수) 만큼 증가시킵니다.
- 이를 위해 각 위치별로 대사 사용 횟수를 저장한다면, sliding window 기법과 비슷하게 deque를 이용하여 효과를 더 이상 발휘하지 않는 대사들을 deque에서 제거하는 방식으로 이전 대사들로 인한 현재 위치에서 늘어나는 강함 수치를 상수 시간 내에 구할 수 있습니다.
- 이를 잘 이용하면, 총 $O(N)$ 시간 안에 최소 대사 사용 횟수를 구할 수 있습니다.

2G. 마키마씨가 정해주는 오늘 점심의 맛

`graph traversal`
출제진 의도 – **Medium**

- 제출 30번, 정답 11명 (정답률 36.667%)
- 처음 푼 사람: **kcm1700**, 47분
- 출제자: `aaat`

- 각 정점이 (i, j, k) 로 되어 있고 (i_1, j_1, k_1) 에서 (i_2, j_2, k_2) 으로 연결되있다는 것이 원래 주어진 그래프에서 i_1 번 식당 에서 i_2 번 식당으로, j_1 번 식당에서 j_2 번 식당으로, k_1 번 식당에서 k_2 번 식당으로 이동할 수 있다는 것과 동치인 그래프를 생각해봅시다.
- 덴지, 아키와 파워가 있는 식당번호를 각각 i, j, k 라고 할때 $1 \leq i, j, k \leq N$ 이 성립합니다. 즉 위에 설명된 방식으로 만들어지는 그래프의 정점의 개수는 N^3 며 만약 정답이 존재한다면 거리가 같은 식당까지 가는 가장 짧은 경로는 항상 N^3 보다 같거나 작습니다.

- 모든 도로의 길이는 같기 때문에 만들어진 그래프에서 처음 셋이 있는 (A, B, C) 정점에서 (i, j, k) 정점에 도달할 수 있다면 원래 주어진 그래프에서 A 번 식당에서 부터 i 번 식당까지, B 번 식당에서 부터 j 번 식당까지, C 번 식당에서 부터 k 번 식당까지 가는 경로가 존재하며, 이 경로들의 길이는 같음을 알 수 있습니다.
- (A, B, C) 정점에서 bfs를 시작해서 처음으로 (i, i, i) 형태의 정점에 방문하면, i 번 식당이 이들이 최단거리의 이동으로 만날 수 있는 식당의 번호입니다.

- bfs를 하면서 새 정점에 방문할 때마다 이전에 방문한 정점을 저장하는 것으로 (A, B, C) 정점에서 (i, i, i) 정점까지의 경로를 구할 수 있습니다.
- 이러한 방식으로 문제를 풀었을때 만들어진 그래프에서 방문할 수 있는 최대 정점의 갯수는 N^3 개이며 각 정점이 가질 수 있는 간선의 갯수도 최대 N^3 개 이므로 총 시간 복잡도는 $O(N^6)$ 입니다

2H. (재밌고 웃기고 센스있고 깔끔한 제목)

`math`, `recursion`
출제진 의도 – **Hard**

- 제출 64번, 정답 17명 (정답률 26.562%)
- 처음 푼 사람: **surface_03**, 52분
- 출제자: `kirvy810`

2H. (재밌고 웃기고 센스있고 깔끔한 제목)

- S_n의 길이를 L_n라고 합시다. L_n은 다음과 같이 계산할 수 있습니다.
  - $L_n = \begin{cases} 2 & n = 1, 2 \\ 2 + L_{n-2} + L_{n-1} & n \geq 3 \end{cases}$
- $L_n \leq 10^{18}$을 만족하는 n의 최댓값을 M이라 하고, $1 \leq i \leq M$에 대해 L_i을 전처리합니다.
  - 실제로 M을 계산해보면 $M = 85$라는 것을 알 수 있습니다.
- $n \leq M$이고 $L_n < k$인 경우, '`0`'을 출력하면 됩니다.

2H. (재밌고 웃기고 센스있고 깔끔한 제목)

- S_n 은 S_{n-1} 와 S_{n-2} 을 포함하고 있으므로, 재귀적으로 n 와 k 의 값을 줄여나갈 수 있습니다.
- $S_n (n \geq 3)$ 을 4개의 부분으로 나누어 생각해, 이 과정을 반복합니다.
  - $k = 1$ 인 경우, '('을 출력하면 됩니다.
  - $2 \leq k \leq 1 + L_{n-2}$ 인 경우, S_{n-2} 의 $k - 1$ 번째 문자를 구하면 됩니다.
  - $2 + L_{n-2} \leq k \leq L_n - 1$ 인 경우, S_{n-1} 의 $k - L_{n-1} - 1$ 번째 문자를 구하면 됩니다.
  - $k = L_n$ 인 경우, ')'을 출력하면 됩니다.
- 이를 반복하여 n 이 2 이하가 될 때, 답을 쉽게 구할 수 있습니다.
- 이때 시간 복잡도는 $\mathcal{O}(n)$ 입니다.

2H. (재밌고 웃기고 센스있고 깔끔한 제목)

- n은 최대 10^{18}이므로 단순히 이 과정을 실행하면 시간 초과가 발생합니다.
- 주어진 n이 $M = 85$보다 클 때, n과 k의 값을 다음과 같이 줄일 수 있습니다.
  - $n \leftarrow n - \lfloor \frac{n-M}{2} \rfloor \times 2$
  - $k \leftarrow k - \lfloor \frac{n-M}{2} \rfloor$
- n이 홀수라면 M, 짝수라면 $M+1$이 되어 답을 빠르게 구할 수 있게 됩니다.
- 이때 $k < 0$이라면 ' (' 을 출력하면 됩니다.

21. 귀엽기만 한 게 아닌 한별 양

`graphs`, `graph traversal`, `bfs`
출제진 의도 – **Hard**

- 제출 41번, 정답 13명 (정답률 31.707%)
- 처음 푼 사람: **yooyou7**, 117분
- 출제자: `sprout_429`

- 기본적인 아이디어는 격자 상에서 최단경로를 구하는 문제와 동일합니다.
- 하지만, 일부 연속된 3개의 칸을 지나갈 수 없다는 제약 사항이 존재합니다.
- 이는 현재 위치와 이전 위치를 하나의 상태로 저장하는 방법으로 해결이 가능합니다.
  - 당신과 한별이가 어떤 경로로 이동했는지에 관계없이, 문제에 걸린 제약 조건은 가장 최근 지나온 3개 이하 칸에 걸려있기 때문에 현재 위치와 이전 위치만 체크해주면 됩니다.
  - 현재 칸에서 다음 칸으로 이동 불가능한 경우는 다음 칸이 'X'이거나, 이전 칸과 다음에 가고자 하는 칸이 동일하거나, 이전, 현재, 다음 칸의 불상사의 개수의 합이 K개를 초과하는 경우입니다.

- 각 위치에 대하여, 직전 위치로는 4 가지가 존재하고, 시작점의 경우 직전 위치가 존재하지 않으므로 가능한 상태의 개수는 $5 \times N \times M$ 개 이하가 됩니다.
- 불상사의 개수가 K 개를 넘어가는 상태를 제외한 뒤, BFS를 이용하면 최단경로를 시간 내에 구할 수 있습니다.
  - 시작점만 잘 처리해 주게 된다면 $4 \times N \times M$ 개의 상태로도 문제를 해결할 수 있습니다.
- 각 위치마다 인접한 위치는 최대 4 개이므로 전체 시간복잡도는 $O(16MN)$ 입니다.

TMI

- 이 문제는 출제자가 ICPC에서 BOJ 26106를 해결 한 이후 아이디어가 재미있어 해당 문제를 변형시키다 만들어졌습니다.
- 해당 문제를 “학교생활!”이라는 애니메이션을 기반으로 지문을 작성할까 고민하였지만 해당 작품에 대한 스포일러가 될 수 있을 것 같아 ”귀엽기만 한 게 아닌 시키모리 양”으로 변경하게 되었습니다.

2J. 엔드롤이 끝나고

`math`
출제진 의도 – **Hard**

- 제출 71번, 정답 12명 (정답률 16.901%)
- 처음 푼 사람: **ncy09**, 34분
- 출제자: `ruykun`

- $w_1, w_2, \cdots, w_K$ 의 실력을 가진 파티원이 있을 때, 어떤 파티원을 실력 차이를 정하는 기준으로 골라야 $\sum_{i=1}^{K} |w_i - x|$ 을 최소로 할 수 있는지 생각해 봅시다. x 는 기준이 되는 파티원의 실력입니다.

- $|w_i - x| = \begin{cases} w_i - x, & \text{if } x < w_i \\ x - w_i, & \text{if } x \geq w_i \end{cases}$ 는 볼록함수입니다.
- 볼록함수끼리의 합은 볼록함수이므로 $\sum_{i=1}^{K} |w_i - x|$ 또한 볼록함수이고, 기울기가 0 인 구간, 혹은 기울기가 음수에서 양수가 되는 지점에서 값이 최소가 됩니다.

- 단조 증가하도록 파티원이 정렬되어 있다고 합시다.
- K 가 홀수일 경우 실력이 $w_{\lceil K/2 \rceil}$ 인 파티원이, K 가 짝수일 경우 실력이 $w_{K/2}$ 혹은, $w_{K/2+1}$ 인 파티원이 기준이 된다면 반드시 파티의 실력 차이가 최소가 됩니다.
- 홀수일 때와 짝수일 때의 차이를 없애기 위해 반드시 실력이 $w_{\lceil K/2 \rceil}$ 인 파티원이 기준이 되도록 하고, 이후로 $x = w_{\lceil K/2 \rceil}$ 로 고정합니다.

- 어떤 파티원이 기준이 되어야 하는지를 알았으니 파티의 구성원을 어떻게 해야 할지 생각해 봅시다.
- 가능한 경우는 ${}_N\mathrm{C}_{K-1}$ 가지로, 이를 전부 확인할 경우 적어도 $O({}_N\mathrm{C}_{K-1})$ 의 시간복잡도를 가져 시간초과가 나게 됩니다.

- x 가 고정되어 있으므로, 각 w_i 가 x 와의 차이가 작아질 때 $\sum_{i=1}^{K}|w_i - x|$ 가 작아집니다.
- 기준이 되는 파티원과의 실력 차이가 적은 파티원 후보를 파티원으로 골라야 파티의 실력 차이를 작게 할 수 있습니다.
- 파티원 후보를 실력 순으로 정렬했을 때 연속된 후보를 파티원으로 하는 것이 그렇지 않은 것보다 이득입니다.

- 실력 순으로 정렬된 파티원 후보에서 연속된 $K-1$ 명을 고르는 방법은 $N-K+2$ 가지가 있습니다.
- 이 모든 경우에 대한 파티의 실력 차이의 최솟값을 구하면 문제의 정답을 구할 수 있습니다.
- 그러나 모든 경우에 대해 단순히 $|w_i - x|$ 를 K 번씩 구할 경우 $O(NK)$ 의 시간복잡도를 가지게 되고, 시간 초과가 나게 됩니다.

2J. 엔드롤이 끝나고

- 주어진 파티원 후보의 실력이 단조 증가하도록 정렬합시다.
- 정렬된 파티원 후보에서 어떤 연속된 구간을 정해도 그 구간 내에서 파티원은 실력 순으로 정렬되어 있습니다.
- $s \in [1, N-K+2]$ 를 정하고, $s, s+1, \cdots, s+K-2$ 번째 후보를 파티원으로 고릅시다.

- K 명의 파티원 중 용사의 실력이 몇 번째에 위치하는지에 따라 계산이 달라집니다.
  - $Y < x$ 인 경우 $v_{s+\lceil K/2 \rceil - 3} \leq x = v_{s+\lceil K/2 \rceil - 2} \leq v_{s+\lceil K/2 \rceil - 1}$ 입니다.
  - $Y = x$ 인 경우 $v_{s+\lceil K/2 \rceil - 2} \leq x \leq v_{s+\lceil K/2 \rceil - 1}$ 입니다.
  - $Y > x$ 인 경우 $v_{s+\lceil K/2 \rceil - 2} \leq x = v_{s+\lceil K/2 \rceil - 1} \leq v_{s+\lceil K/2 \rceil}$ 입니다.

- 어떤 구간에서 $v_i \leq x$ 이고, 어떤 구간에서 $v_i \geq x$ 인지를 알고 있으므로 파티의 실력 차이를 나타내는 식을 절대값 기호가 없는 형태로 나타낼 수 있게 됩니다.
- $Y < x$ 인 경우를 예로 들면, 파티의 실력 차이는 다음과 같이 나타낼 수 있습니다.

$$(\lceil K/2 \rceil - 1)x - \sum_{i=s}^{s+\lceil K/2 \rceil - 3} v_i - Y + \sum_{i=s+\lceil K/2 \rceil - 1}^{s+K-2} v_i - (K - \lceil K/2 \rceil)x$$

- $\sum_{i=a}^{b} v_i = \sum_{i=1}^{b} v_i - \sum_{i=1}^{a} v_i$ 이므로 $\forall a \in [1, N]$ 에 대한 $\sum_{i=1}^{a} v_i$ 를 미리 구해 놓는다면 이 계산은 $O(1)$ 에 실행할 수 있습니다.
- 이를 이용해 s 가 정해지면 파티의 실력 차이를 $O(1)$ 에 계산할 수 있습니다.
- $s \in [1, N - K + 2]$ 인 모든 s 에 대해 계산하므로 파티의 실력 차이의 최솟값은 $O(N)$ 에 구할 수 있습니다.

- 파티원 후보를 실력 순으로 정렬하는 데 $O(N \log N)$, 파티의 실력 차이의 최솟값을 구하는 데 $O(N)$ 으로, 전체 문제는 $O(N \log N)$ 으로 해결할 수 있습니다.

2K. 카드캡터 한별

`dijkstra`, `dynamic programming`, `bitmasking`
출제진 의도 – **Hard**

- 제출 18번, 정답 5명 (정답률 27.778%)
- 처음 푼 사람: **tlwpdus**, 174분
- 출제자: `sivcde0405`

2K. 카드캡터 한별

- 한별이가 카드를 모두 수집하려면 정점 $k_i (0 \leq i \leq N)$ 를 모두 방문해야 합니다.
  - 단, 입력에서 주어지지 않은 k_0 은 1로 정의합니다.
- 이는 한별이가 1 번 정점으로 돌아올 필요가 없다는 것과 한별이가 가진 카드의 개수에 따라 각 정점간의 최단경로가 달라진다는 것을 제외하면 외판원 순회 문제와 유사합니다.
- 따라서 최단거리 알고리즘으로 미리 전처리를 해주면 DP를 이용해 $N^2 2^N$ 회의 연산으로 문제를 해결할 수 있습니다.

- $0 \leq L \leq N-1, 1 \leq i, j \leq V$ 을 만족하는 모든 L, i, j 에 대하여 한별이가 L 개의 카드를 가지고 있을 때 정점 k_i 에서 정점 k_j 로 이동하는 최단거리 $dist[L][i][j]$ 를 모두 구합니다.
- 최단거리를 구하는 방법은 뒤에 서술했습니다.

- 정수형 2차원 배열의 원소 $A[bit][n]$를 다음과 같이 정의합니다.
  $(0 \leq bit \leq 2^N - 1, 0 \leq n \leq N)$
  - $A[bit][n]$의 초기값은 최단거리를 탐색할 때 설정한 거리의 최댓값인 INF 입니다.
  - 단, $A[0][0]$은 0으로 설정합니다.
  - 현재 정점 k_n에 위치하고, bit에 비트마스킹된 정점들을 방문했을 때 최단경로의 길이
  - bit를 이진수로 나타낼 때, $\alpha - 1$번째 자리가 1이면 정점 k_α를 방문함,
  - bit를 이진수로 나타낼 때, $\alpha - 1$번째 자리가 0이면 정점 k_α를 방문하지 않음 $(1 \leq \alpha \leq N)$

- $A[i+2^{j-1}][j]$를 $\min_{1\le k\le N, k\neq j} A[i][k]+dist[cnt][k][j]$와 원래의 $A[i][j]$ 중 더 작은 값으로 갱신합니다.
  - i를 이진수로 나타낼 때 $j-1$번째 자리가 1이라면 j번 정점을 이미 방문했으므로 갱신하지 않습니다.
  - cnt는 i를 이진수로 표현했을 때 등장하는 1의 개수로, 한별이의 현재 레벨 L과 같습니다.

- A를 DP를 이용하여 계산한 후 ans를 $A[2^n-1][1,\cdots,n]$중의 최솟값으로 정의합니다. 처음 최단거리를 탐색할 때 설정한 거리의 최댓값을 INF라 할 때 다음의 두 가지 경우가 존재합니다.
  - $ans < INF$일 때는 이동이 가능하므로 답이 ans입니다.
  - $ans >= INF$일 때는 이동이 불가능하므로 답이 -1입니다.
- 구현 방식에 따라 $\mathcal{O}\left(N^2E\log E+N^2 2^N\right), \mathcal{O}\left(N^2V^2+N^2 2^N\right), \mathcal{O}\left(N^2VE+N^2 2^N\right)$ 등의 시간복잡도로 문제를 해결할 수 있습니다.

- 최단거리를 구하는 알고리즘 중 일부의 연산 횟수는 다음과 같습니다.
  - 힙을 사용한 다익스트라 알고리즘: $N^2 E \log E$ 회
  - $\mathcal{O}\left(V^2\right)$ 다익스트라 알고리즘: $N^2 V^2$ 회
  - SPFA 알고리즘: $N^2 V E$ 회
  - 벨만-포드 알고리즘: $N^2 V E$ 회
  - 플로이드-워셜 알고리즘: $N V^3$ 회
- N, V, E 를 고려했을 때 벨만-포드 알고리즘, 플로이드-워셜 알고리즘 등은 시간 초과를 받을 수 있으므로 사용할 수 없습니다.
- Python 3, Kotlin 등의 언어로 문제를 해결하는 경우에는 시간 초과를 방지하기 위해 힙을 사용한 다익스트라 알고리즘의 사용을 권장합니다.

2K. 카드캡터 한별

여담

- 처음에 문제를 생각했을 때, 이 문제의 제목은 “마법소녀 한별이”였습니다. 출제자(염화은, 18세)가 아는 애니메이션이나 만화 중에는 마법소녀가 무언가를 획득하면서 성장하는 작품이 없었기 때문입니다.
- 하지만 sait2000님께서 고전 작품 중 하나인 “카드캡터 체리”라는 애니메이션을 패러디하는 것을 제안하셨고, 그것이 채택되어 지금의 제목을 가지게 되었습니다.[1]
- 악의 세력이 가진 이름인 “보기”는 출제자가 초록창 영어사전을 뒤지다가 발견한 단어로, 미식별 항공기라는 의미를 가지고 있습니다. n개의 이름 후보 중 뭔가 악당에 잘 어울리는 이름이라서 결국 이 이름이 채택되었습니다.
- Index님께서 이 문제의 솔루션과 제너레이터를 n개 제작해주셨습니다. 문제 세팅에 엄청난 도움을 주신 Index님 진짜 감사합니다..!

[1] 아니 저 제안한 적 없는데요 그땐 지문에 카드 대신 열쇠였으니까 그 애니 생각난단 말밖에 안 했어요 진짜로

2L. 치노와 코코아

`ad-hoc`, `constructive`
출제진 의도 – **Challenging**

- 제출 13번, 정답 3명 (정답률 23.077%)
- 처음 푼 사람: **tlwpdus**, 175분
- 출제자: `ibm2006`

– 한 정점의 높이를 그 정점과 루트를 잇는 경로에 포함된 간선의 개수로 정의하고, 치노가 받은 트리에서 1번 정점을 루트로 할 때 A_i를 높이가 i인 정점들의 집합으로 정의합니다. 다음의 성질들이 성립합니다.

- 모든 $1 \leq i \leq 10$에 대해, A_i의 어떤 원소들도 서로 인접하지 않습니다.
- 모든 $1 \leq i \leq 10$과 $v \in A_i$에 대해서, A_{i-1}에는 v와 인접한 정점이 정확히 하나 존재하고, 모든 $1 \leq j \leq i-2$에 대해서 A_j에는 v와 인접한 정점이 존재하지 않습니다.

- 첫 번째 성질을 이용하여 추가하는 간선의 개수가 $\frac{N^2}{20} + O(N)$ 이도록 치노가 간선을 적절히 추가할 수 있습니다.
  - 우선, 치노는 모든 $0 \leq i \leq 10$에 대해 A_i의 모든 정점들 사이에 간선을 추가해줍니다.
  - 코코아가 기존 트리 기준 높이 $i (1 \leq i \leq 9)$ 이하이며, 0 이상인 모든 정점들의 부모를 알아내었다고 가정합니다. 높이 $i+1$인 임의의 정점 v는 정확히 하나의 높이 i인 정점과 인접해 있고, 높이 $i+2$ 이상인 정점은 그 어떤 높이 i인 정점과도 연결되어 있지 않습니다. 높이 $i+1$의 정점들은 반드시 높이 i인 부모를 가지므로, 모든 높이 $i+1$인 정점들의 부모를 알아낼 수 있습니다.
  - 해당 과정을 반복하면 코코아는 트리 전체의 구조를 알아낼 수 있습니다.

A_i 의 어떤 두 정점도 기존 트리에서는 연결되어 있지 않았기 때문에, 각 $0 \le i \le 10$ 마다 추가하는 간선의 개수는 $\binom{|A_i|}{2}$ 가 됩니다. 따라서 추가하는 전체 간선의 개수는 $\sum_{i=1}^{10} \binom{|A_i|}{2} = \frac{1}{2}\left(\sum_{i=1}^{10} |A_i|^2 + \sum_{i=1}^{10} |A_i|\right) = \frac{N-1}{2} + \frac{1}{2}\left(\sum_{i=1}^{10} |A_i|^2\right)$ 가 됩니다. 코시 슈바르츠 부등식에 의해, $\sum_{i=1}^{10} |A_i|^2 \ge \frac{(N-1)^2}{10}$ 이 성립합니다.

따라서 추가하는 총 간선의 개수는 $\frac{N^2}{20} + O(N)$ 이며, 최악의 경우는 A_i 의 크기들이 균등할 때입니다.

- 두번째 성질을 활용하면 치노가 더욱 많은 간선을 추가하는 전략을 세울 수 있으며, 추가하는 간선의 개수가 $\frac{N^2}{4} + O(N)$ 임을 증명할 수 있습니다.
  - 치노는 모든 $0 \le i \le 10$ 에 대해 A_i 의 모든 정점들 사이에 간선을 추가합니다. 추가로, 모든 $0 \le i \le 10$ 에 대해 만약 $|A_i| \ge 2$ 이라면 $v \in A_i$ 인 모든 v 에 대해, 높이가 $i+2$ 이상인 모든 정점과 v 를 연결합니다.
  - 코코아가 높이 $i (0 \le i \le 9)$ 까지의 모든 정점들에 대해 기존 트리에서의 연결성을 알고 있었다고 가정합니다. 높이가 $i+2$ 이상인 모든 정점 v 는 $|A_i| \ge 2$ 라서 2개 이상의 A_i 의 정점과 연결되어 있거나, 그렇지 않아서 어떤 A_i 의 정점과도 연결되어 있지 않습니다. 따라서 높이가 i 초과인 모든 정점들에 대해, 정확히 하나의 A_i 의 원소와 인접한 정점들이 높이 $i+1$ 인 정점이며, 그러한 모든 정점들의 부모 또한 알 수 있습니다.
  - 해당 과정을 반복하면 코코아는 트리 전체의 구조를 알아낼 수 있습니다.

- 이러한 전략을 적용할 경우 추가하는 전체 간선의 개수가 $\left\lfloor \frac{N^2}{5} \right\rfloor$ 이상임을 증명할 수 있습니다.
- 우선, 추가하는 간선의 개수는 각 높이를 가지는 정점의 개수에 의존합니다. 높이가 i인 정점의 개수를 a_i라고 정의합니다. $i \geq 1$인 i에 대해 $|A_i| = 1$인 경우가 있다고 가정하고, A_i의 유일한 원소를 x라고 합시다. 이제 x를 제거하고, x의 자식들을 x의 부모와 연결하는 방식으로 트리를 변형하면, 동일한 전략을 적용할 때 추가하는 간선의 개수가 늘어나지 않습니다. 이렇게 변형한 트리에 추가되는 간선의 개수가 문제 조건을 만족함을 보이면 충분합니다.

- 새로운 트리의 높이를 k, 총 정점의 개수를 $n+1$ 이라고 정의합니다. 새로운 트리에서는 $i \geq 1$ 인 모든 A_i 가 $|A_i| \geq 2$ 을 만족하므로, 모든 정점은 자신과 높이가 1 차이나는 정점들을 제외한 모든 정점과 연결됩니다. 따라서, 총 간선의 개수는 $\frac{n(n-1)}{2} - \sum_{i=1}^{k-1} |A_i||A_{i+1}|$ 입니다.
- $k \geq 3$ 이었다고 가정합니다. 그렇다면 A_3 에 A_1 의 모든 원소를 추가하고, A_1 을 제거한 뒤 모든 정점들의 높이를 1 씩 내립니다. 그렇다면 기존의 $|A_1||A_2| + |A_2||A_3| + |A_3||A_4|$ 만큼 빠지던 값이 $|A_2|(|A_1| + |A_3|) + (|A_1| + |A_3|)|A_4|$ 가 되어, 전체 추가하는 간선의 개수가 감소하게 됩니다. 이러한 원리를 적용하면 $k = 1, 2$ 인 경우만 고려하면 충분합니다.

- $k = 1$인 경우, 추가하는 간선의 개수는 $\frac{n(n-1)}{2}$ 이고, 이 경우 $n \geq N - 10$이므로 $N \geq 100$ 일때 문제 조건을 만족합니다.
- $k = 2$인 경우, 추가하는 간선의 개수는 $\frac{n(n-1)}{2} - |A_1||A_2|$가 됩니다. n은 고정되어 있고 $|A_1| + |A_2| = n$이므로, $\frac{n(n-1)}{2} - |A_1||A_2| \geq \frac{n(n-1)}{2} - \frac{n^2}{4} = \frac{n^2 - 2n}{4}$가 됩니다.
- $k = 2$인 경우, $n \geq N - 9$이기 때문에, 기존 트리에 치노가 추가한 간선의 개수는 $\frac{(N-9)(N-11)}{4} \geq \frac{N^2}{5} \geq \left\lfloor \frac{N^2}{5} \right\rfloor$ 이상입니다. 첫 번째 부등식이 $N \geq 100$일 때 항상 성립한다는 사실은 쉽게 증명할 수 있습니다.
- 위의 결과를 토대로, 해당 전략의 최악의 경우는 $\{a_i\}_{i=0}^{10} = \left\{1, 1, 1, 1, 1, 1, 1, 1, 1, \left\lfloor \frac{N-9}{2} \right\rfloor, \left\lceil \frac{N-9}{2} \right\rceil \right\}$ 임 또한 알 수 있습니다.

- 다른 풀이로, 두 정점 사이의 연결 유무를 비트에 대응하여 치노가 추가하는 간선의 개수가 $\frac{N^2}{2} + O(N \log N)$ 인 전략을 세울 수 있습니다.
- 치노가 받은 트리의 각 정점간 연결 상태를 나타내는 $\frac{N(N-1)}{2}$ 개의 비트를 생각하고, 각 비트들을 $\lceil \log_2 n \rceil$ 개씩 묶어줍니다. 만약 어떤 묶음에 1 이 있었다면, 해당 묶음의 모든 비트들을 1 로 변경하고, 그에 대응되는 간선들을 추가해줍니다. 만약 어떤 묶음이 오직 0 으로만 이루어져 있다면, 여기에 정점 하나 분량의 정보를 저장할 수 있습니다. 각 정점의 번호를 이진법으로 표현한 다음, 그 수에 대응하여 비트를 변경해줍니다.

- 묶음의 개수는 $\frac{N(N-1)}{2\log N}$ 개이고, 켜져있는 비트의 개수는 많아야 $2N$ 개이므로 $N \geq 100$ 이라는 조건 하에 0 으로만 이루어진 묶음의 개수는 $2N$ 개 이상이고, 간선의 모든 정보를 담기에 충분합니다. 쓰지 않고 남은 묶음들은 전부 1 로 채워주면, 가능한 모든 간선 중 채워지지 않은 간선의 개수는 많아야 $2N \log N$ 개입니다.
- 코코아는 주어진 그래프에 대해 치노와 동일한 방식으로 비트 배열을 만들고, 묶음을 만들어줍니다. 각 묶음들을 통해 간선의 정보를 알아낼 수 있고, 따라서 치노에게 주어진 트리를 알아낼 수 있습니다.
- 문제 조건에서 $\lceil \log_2 n \rceil \leq 10$ 이고, $\frac{N(N-1)}{2} - 20N - (N-1) \geq \left\lfloor \frac{N^2}{5} \right\rfloor$ 이 $N \geq 100$ 일때 성립함은 어렵지 않게 보일 수 있습니다.

2M. THE iDEM@STER

`string, implementation`

출제진 의도 – **Challenging**

- 제출 9번, 정답 1명 (정답률 11.111%)
- 처음 푼 사람: **tlwpdus**, 206분
- 출제자: dhyang24

- 문제에서 주어진 문법을 만족하는 프로그램 S를 "올바른 프로그램"이라고 하고, 이 프로그램의 결과값을 $V(S)$라고 합시다.
- 그리고 $V(S) = N$인 S들 중에서 가장 짧은 것을 "최적의 프로그램"이라고 합시다. 어떤 N에 대해 최적의 프로그램은 여러 개일 수도 있습니다.
- 편의상 S의 부분 문자열이면서 주어진 문법을 만족하는 부분을 "부분 프로그램"이라고 하고, PPP $\cdots$ @@@ 에서 PPP와 @@@가 괄호 쌍일 때, 이 부분 프로그램을 "루프"라고 하겠습니다.

- 어떤 올바른 프로그램 S에 대해, S의 글자 순서를 뒤집고 @를 P로, P를 @로 치환한 프로그램을 S의 "반전", 또는 $inv(S)$라고 합시다. 그러면 다음이 성립합니다.
- Lemma 1. 프로그램 S의 반전 $inv(S)$에 대해 $V(inv(S)) = -V(S)$가 성립합니다.
- 증명은 다음 페이지에 있습니다.
- Lemma 1에 의해, 양의 정수 N에 대한 최적의 프로그램의 집합 $C = \{S_1, S_2, \cdots\}$를 알고 있다면, $-N$에 대한 최적의 프로그램의 집합은 C의 프로그램들의 반전의 집합 $inv(C) = \{inv(S_1), inv(S_2), \cdots\}$와 같습니다. 따라서 $|N|$에 대해서 C를 구한 다음, N이 양수이면 C에서, N이 음수라면 $inv(C)$에서 사전 순으로 빠른 프로그램을 고르면 됩니다.

– Proof. 모든 올바른 프로그램은 빈 문자열, @, P에서 시작해서, 두 올바른 프로그램을 이어 붙이거나 PPP$\cdots$@@@로 감싸는 방법으로 만들 수 있습니다. 각각의 경우에 대해 주어진 명제를 증명하면 구조적 귀납법에 의해 모든 올바른 프로그램에 대해 주어진 명제가 성립합니다.

- $S = \epsilon$: $V(S) = 0, inv(S) = \epsilon, V(inv(S)) = 0 = -V(S)$
- $S =$ P: $V(S) = 1, inv(S) =$ @, $V(inv(S)) = -1 = -V(S)$
- $S =$ @: $V(S) = -1, inv(S) =$ P, $V(inv(S)) = 1 = -V(S)$
- $S = S_1S_2$: $V(S) = V(S_1) + V(S_2), inv(S) = inv(S_2)inv(S_1)$, $V(inv(S)) = V(inv(S_2)) + V(inv(S_1)) = -V(S)$
- $S =$ PPPS_1@@@: $V(S) = 3V(S_1), inv(S) =$ PPP$inv(S_1)$@@@, $V(inv(S)) = 3V(inv(S_1)) = -V(S)$ □

- Lemma 2. 부분 프로그램 a, b, c, d, e에 대해, $S = a$PPPb@@@cPPPd@@@e는 최적의 프로그램이 아닙니다. (조건에 의해 b와 d는 각각 루프 안에 들어있는 형태입니다.)
- Proof. c가 빈 문자열이 아니라면, c는 반드시 P로 시작해서 @로 끝나고, e는 빈 문자열이거나 P로 시작하므로, $S_1 = a$PPPb@@@PPPd@@@ce는 올바른 프로그램이면서 $V(S) = V(S_1)$입니다. c가 빈 문자열이면 $S = S_1$입니다.
- 그리고 S_1에서 b는 P로 끝나고 d는 @로 시작하므로 두 루프를 하나로 합친 $S_2 = a$PPPbd@@@ce 또한 올바른 프로그램이며, $V(S_2) = V(S)$이고, S_2는 S보다 길이가 6 짧습니다. 따라서 S는 최적의 프로그램이 아닙니다. □

- Lemma 2에 의해, 주어진 N에 대한 최적의 프로그램은 최상위 루프가 없거나 하나만 있을 수 있고, 루프가 있다면 $S = a\text{PPP}b@@@c$일 때 b도 재귀적으로 이 조건을 만족해야 함을 알 수 있습니다.
- 규칙에 의해 루프 내부의 부분 프로그램 b는 @로 시작해서 P로 끝나야 합니다. 이 제약사항이 있을 때의 최적의 프로그램은 제약사항이 없을 때(문제의 최종 정답)와 다를 수 있으므로, 두 가지를 따로 고려해야 합니다.
- @로 시작해서 P로 끝나면서 결과값이 N인 프로그램의 최소 길이를 $in(N)$, 그 길이를 갖는 프로그램을 $ins(N)$, 시작과 끝 글자에 제한이 없는 경우는 각각 $out(N)$과 $outs(N)$이라고 합시다. 하나의 N 값에 대해 $ins(N)$이나 $outs(N)$은 여러 개일 수 있습니다.

- Lemma 1에 의해, $in(-N) = in(N)$, $ins(-N) = inv(ins(N))$, $out(-N) = out(N)$, $outs(-N) = inv(outs(N))$ 이 성립합니다.
- 따라서, N 이 음수이면 $-N$ 을 대신 고려하면 N 이 양수인 경우만을 고려할 수 있습니다.

- 이제 $in(N)$ 부터 해결해 봅시다. (@로 시작해서 P로 끝나야 하는 경우)
- 편의상 부분 프로그램 a를 k번 반복한 프로그램을 $(a)_k$라고 쓰겠습니다.
- 루프가 없는 경우의 최적의 프로그램은 $(\texttt{@PP})_N$이고, 이때 길이는 $3N$입니다.
- 루프를 포함하는 경우의 최적의 프로그램은 $N \bmod 3$에 따라 달라집니다. 수식의 간결함을 위해, 여기서부터 $\frac{N}{3}$은 N을 3으로 나눈 몫, 즉 $\lfloor \frac{N}{3} \rfloor$을 의미합니다. 이때 프로그램의 길이는 루프 안의 결과값에 따라 결정됩니다.

– $N \bmod 3 = 0$인 경우

$ins(N)$ 후보	$in(N)$ 후보
@PPP$(ins(\frac{N}{3}))$@@@P	$in(\frac{N}{3}) + 8$
@@P@@P@@PPP$(ins(\frac{N}{3}+1))$@@@P	$in(\frac{N}{3}+1) + 15$
$(@@P@@P@@P)_k$@@P@@P@@PPP$(ins(\frac{N}{3}+1+k))$@@@P	$in(\frac{N}{3}+1+k) + 15 + 9k$
@PPP$(ins(\frac{N}{3}-1))$@@@PP@PP@PP	$in(\frac{N}{3}-1) + 15$
@PPP$(ins(\frac{N}{3}-1-k))$@@@PP@PP@PP$(@PP@PP@PP)_k$	$in(\frac{N}{3}-1-k) + 15 + 9k$

- $N \bmod 3 = 1$ 인 경우

$ins(N)$ 후보	$in(N)$ 후보
@PPP$(ins(\frac{N}{3}))$@@@PP	$in(\frac{N}{3}) + 9$
@PPP$(ins(\frac{N}{3} - k))$@@@PP$(@PP@PP@PP)_k$	$in(\frac{N}{3} - k) + 9 + 9k$
@@P@@PPP$(ins(\frac{N}{3} + 1))$@@@P	$in(\frac{N}{3} + 1) + 12$
$(@@P@@P@@P)_k$@@P@@PPP$(ins(\frac{N}{3} + 1 + k))$@@@P	$in(\frac{N}{3} + 1 + k) + 12 + 9k$

– $N \bmod 3 = 2$인 경우

$ins(N)$ 후보	$in(N)$ 후보
@PPP$(ins(\frac{N}{3}))$@@@PP@PP	$in(\frac{N}{3}) + 12$
@PPP$(ins(\frac{N}{3} - k))$@@@PP@PP$(\text{@PP@PP@PP})_k$	$in(\frac{N}{3} - k) + 12 + 9k$
@@PPP$(ins(\frac{N}{3} + 1))$@@@P	$in(\frac{N}{3} + 1) + 9$
$(\text{@@P@@P@@P})_k$@@PPP$(ins(\frac{N}{3} + 1 + k))$@@@P	$in(\frac{N}{3} + 1 + k) + 9 + 9k$

- Theorem 3. 모든 $N \geq 1$에 대해, $in(N+1) - in(N) \in \{3, 1, -1\}$ 입니다.
- 어떤 미지의 값이 특정 집합의 원소 중 하나만 될 수 있을 때, 수식에서 편의상 이를 그 집합으로 표현하겠습니다. 예를 들어, 위 Theorem의 수식은 $in(N+1) - in(N) = \{3, 1, -1\}$, $in(N+1) = in(N) + \{3, 1, -1\}$로 쓸 수 있습니다.
- Theorem 3의 증명은 다음 페이지에서 시작합니다.

– Proof. (강한 귀납법) 먼저 $1 \leq N \leq 6$인 경우, $ins(N)$을 직접 구해 보면 다음과 같습니다.

N	$ins(N)$	$in(N)$
1	@PP	3
2	@PP@PP	6
3	@PP@PP@PP	9
4	@PP@PP@PP@PP, @PPP@PP@@@PP	12
5	@PP@PP@PP@PP@PP, @PPP@PP@@@PP@PP, @@PPP@PP@PP@@@P	15
6	@PPP@PP@PP@@@P	14

– 따라서 $1 \leq N \leq 4$일 때 $in(N+1) - in(N) = 3$이고, $N = 5$일 때는 -1입니다.

- 이제 $N \geq 6$인 경우를 봅시다.
- 앞서 $in(6) = 14 < 18$이고 루프가 없는 프로그램의 앞 18글자를 $ins(6)$으로 치환하여 항상 더 짧은 프로그램을 얻을 수 있으므로, 루프가 없는 프로그램은 더 이상 고려할 필요가 없습니다.
- 강한 귀납법의 가정을 이용해 $in(N)$의 후보를 더 좁힐 수 있습니다.

$$
\begin{aligned}
N \bmod 3 = 0 \Rightarrow in(N) &= in(\tfrac{N}{3}) + 8 \\
N \bmod 3 = 1 \Rightarrow in(N) &= \min(in(\tfrac{N}{3}) + 9, in(\tfrac{N}{3} + 1) + 12) \\
&= \min(in(\tfrac{N}{3}) + 9, in(\tfrac{N}{3}) + \{3, 1, -1\} + 12) = in(\tfrac{N}{3}) + 9 \\
N \bmod 3 = 2 \Rightarrow in(N) &= \min(in(\tfrac{N}{3}) + 12, in(\tfrac{N}{3} + 1) + 9) \\
&= \min(in(\tfrac{N}{3}) + 12, in(\tfrac{N}{3}) + \{3, 1, -1\} + 9) = in(\tfrac{N}{3} + 1) + 9
\end{aligned}
$$

– 이제 앞의 결과를 이용해 각 경우의 $in(N+1) - in(N)$을 구할 수 있습니다.

$$N \bmod 3 = 0 \Rightarrow in(N+1) - in(N) = 1$$
$$N \bmod 3 = 1 \Rightarrow in(N+1) - in(N) = in(\tfrac{N}{3}+1) - in(\tfrac{N}{3}) = \{3, 1, -1\}$$
$$N \bmod 3 = 2 \Rightarrow in(N+1) - in(N) = -1$$

– 따라서 모든 $N \geq 1$에 대해 $in(N+1) - in(N) = \{3, 1, -1\}$입니다. □

– Corollary 4. $N \geq 1$에 대해 $in(N)$은 다음과 같이 재귀적으로 $\mathcal{O}(\log N)$에 구할 수 있습니다.

$$in(N) = \begin{cases} 3N, & \text{if } N \leq 5 \\ in(\frac{N}{3}) + 8, & \text{if } N > 5 \text{ and } N \bmod 3 = 0 \\ in(\frac{N}{3}) + 9, & \text{if } N > 5 \text{ and } N \bmod 3 = 1 \\ in(\frac{N}{3} + 1) + 9, & \text{if } N > 5 \text{ and } N \bmod 3 = 2 \end{cases}$$

- 이제 $out(N)$을 구해봅시다.
- 루프가 없는 경우의 최적의 프로그램은 P, PP, 또는 PP(@PP)$_{N-2}$ 입니다.
- 루프를 포함하는 경우의 최적의 프로그램은 역시 $N \bmod 3$에 따라 달라집니다.

Cases	$outs(N)$ 후보	$out(N)$ 후보
$N \bmod 3 = 0$	PPP$(ins(\frac{N}{3}))$@@@	$in(\frac{N}{3}) + 6$
$N \bmod 3 = 1$	PPP$(ins(\frac{N}{3}))$@@@P	$in(\frac{N}{3}) + 7$
	@@PPP$(ins(\frac{N}{3}+1))$@@@	$in(\frac{N}{3}+1) + 8$
$N \bmod 3 = 2$	PPP$(ins(\frac{N}{3}))$@@@PP	$in(\frac{N}{3}) + 8$
	@PPP$(ins(\frac{N}{3}+1))$@@@	$in(\frac{N}{3}+1) + 7$

- 이제 Theorem 3의 증명 과정과 비슷한 방법으로 최적해의 길이를 특정할 수 있습니다.
- $1 \leq N \leq 6$인 경우, $outs(N)$을 직접 구해 보면 다음과 같습니다.

N	$outs(N)$	$out(N)$
1	P	1
2	PP	2
3	PP@PP	5
4	PP@PP@PP	8
5	PP@PP@PP@PP, PPP@PP@@@PP	11
6	PPP@PP@PP@@@	12

- $outs(6)$으로부터 $N \geq 6$일 때 루프가 없는 프로그램은 고려할 필요가 없음을 알 수 있습니다.

– 이제 $N \geq 6$인 경우 각각에 대해 $out(N)$을 계산해보면 다음과 같습니다.

$$
\begin{aligned}
N \bmod 3 = 0 \Rightarrow out(N) &= in(\tfrac{N}{3}) + 6 \\
N \bmod 3 = 1 \Rightarrow out(N) &= \min(in(\tfrac{N}{3}) + 7, in(\tfrac{N}{3} + 1) + 8) \\
&= \min(in(\tfrac{N}{3}) + 7, in(\tfrac{N}{3}) + \{3, 1, -1\} + 8) = in(\tfrac{N}{3}) + 7 \\
N \bmod 3 = 2 \Rightarrow out(N) &= \min(in(\tfrac{N}{3}) + 8, in(\tfrac{N}{3} + 1) + 7) \\
&= \min(in(\tfrac{N}{3}) + 8, in(\tfrac{N}{3}) + \{3, 1, -1\} + 7) \\
&= \min(in(\tfrac{N}{3}) + 8, in(\tfrac{N}{3}) + \{10, 8, 6\})
\end{aligned}
$$

- 여기까지 정리해 놓고, 원래 문제의 목표였던 $outs(N)$ 중 사전 순으로 가장 빠른 것을 찾는 문제를 생각해 봅시다.
- $outs(N)$와 $ins(N)$은 대다수의 경우 $ins(\frac{N}{3})$ 또는 $ins(\frac{N}{3}+1)$ 중 한쪽만 가지고 구할 수 있습니다. 길이가 다르면 한쪽은 볼 필요가 없기 때문입니다.
- 그렇다면 양쪽의 길이가 서로 같은 경우는 어떨까요?
- 다행히 모든 경우에 대해 사전 순으로 빠른 쪽이 정해져 있어서 그쪽으로만 재귀하면 됩니다. 자세한 내용은 다음 페이지에 정리되어 있습니다.

- $ins(N)$을 구할 때 $N \geq 6, N \bmod 3 = 2, in(\frac{N}{3}+1) - in(\frac{N}{3}) = 3$인 경우
  - 후보 1 @PPP$(ins(\frac{N}{3}))$@@@PP@PP vs. 후보 2 @@PPP$(ins(\frac{N}{3}+1))$@@@P
- $outs(N)$을 구할 때 $N \geq 6, N \bmod 3 = 1, in(\frac{N}{3}+1) - in(\frac{N}{3}) = -1$인 경우
  - 후보 1 PPP$(ins(\frac{N}{3}))$@@@P vs. 후보 2 @@PPP$(ins(\frac{N}{3}+1))$@@@
- $outs(N)$을 구할 때 $N \geq 6, N \bmod 3 = 2, in(\frac{N}{3}+1) - in(\frac{N}{3}) = 1$인 경우
  - 후보 1 PPP$(ins(\frac{N}{3}))$@@@PP vs. 후보 2 @PPP$(ins(\frac{N}{3}+1))$@@@
- 각각의 경우, 원래 입력 N이 양수이면 후보 2 (@의 prefix가 긴 쪽), 음수이면 후보 1 (P의 suffix가 긴 쪽)을 선택하면 됩니다.

- $outs(N)$과 $ins(N)$을 구할 때 각 재귀 단계에서 어느 쪽으로 재귀를 해야 할지 판별하기 위해 $in(\frac{N}{3}+1) - in(\frac{N}{3})$의 값이 필요할 수 있습니다. 이는 Theorem 3의 각 케이스별 결과를 사용하며, 최악의 경우 $\mathcal{O}(\log N)$ 시간이 소요됩니다.
- $ins(N)$ 재귀의 깊이 또한 $\mathcal{O}(\log N)$이고 매 깊이마다 위의 값을 구해야 할 수 있으므로, 전체적인 최악 시간 복잡도는 테스트 케이스 당 $\mathcal{O}\left(\log^2 N\right)$ 입니다.