The Albatross Tower.

The Weekly K512

Sogang ICPC Team Programming Contest

0¢
ZERO CENTS

NO. 2024 MAPO-GU, SEOUL, SATURDAY, MAY 18, 2024 1 Page

2024 서강대학교 K512컵 에디토리얼

주최 Sogang ICPC Team

후원 서강대학교 SOGANG UNIVERSITY HYUNDAI MOBIS STARTLINK

문제 목록

문제		의도한 난이도	출제자
A	K512컵 개최!	**Beginner**	송근수 hulahula3247
B	K512에 바이러스 퍼뜨리기	**Beginner**	이승형 tmdgud0617
C	근수	**Easy**	이승형 tmdgud0617
D	FA수의 진	**Easy**	유호영 tkfkdd159323
E	Stacking Sticks	**Easy**	채성우 lem0nad3
F	Yunny's Trip	**Medium**	송근수 hulahula3247
G	Gridev's Protocol	**Hard**	송근수 hulahula3247
H	앳코더 스터디	**Hard**	유호영 tkfkdd159323
I	Hula's Cardgame	**Challenging**	송근수 hulahula3247
J	Lem0nad3's Bar	**Challenging**	송근수 hulahula3247



A. K512컵 개최!

`math`

출제진 의도 – **Beginner**

- 제출 66번, 정답 44명 (정답률 66.666%)
- 처음 푼 사람: **이지훈**, 1분
- 출제자: 송근수 `hulahula3247`

A. K512컵 개최!



- 우선 $\times 0$ 주문서는 값을 0으로 바꿔버리므로 시작하자마자 다 사용하고 시작합시다.
- $\times 1$ 주문서는 값을 변화시키지 않으므로 아무 때나 사용해도 괜찮습니다.

A. K512컵 개최!



- 이제 $+$ 주문서랑 2 이상의 $\times$ 주문서만 남았습니다.
- 모든 $+$ 주문서를 먼저 사용하고 모든 $\times$ 주문서를 사용하면 됩니다. 밑에는 간단한 증명입니다.
- 주문서 배치 중 어떠한 부분이 $\times a, +b$ 순서로 되어있다고 합시다.
- 이 둘의 위치를 바꾸면 행운 수치가 최소 $b \times (a-1)$ 만큼 더 올라갑니다. 따라서 모든 $+$ 주문서를 사용한 후 $\times$ 주문서를 사용하는 것이 최적입니다.

B. K512에 바이러스 퍼뜨리기

`simulation`

출제진 의도 – **Beginner**

- 제출 146번, 정답 41명 (정답률 28.082%)
- 처음 푼 사람: **강효규**, 3분
- 출제자: 이승형 `tmdgud0617`

B. K512에 바이러스 퍼뜨리기



- 시뮬레이션을 통해 문제를 해결해 봅시다.
- 1번 쿼리가 들어왔을 때, 해당 컴퓨터를 감염된 컴퓨터 목록에 추가합니다.
- 2번 쿼리가 들어왔을 때, 해당 컴퓨터를 감염된 컴퓨터 목록에서 삭제합니다.
- 3번 쿼리가 들어왔을 때, (전체 컴퓨터의 수) − (감염된 컴퓨터의 수)를 계산하여 출력합니다.

B. K512에 바이러스 퍼뜨리기



- 이 때, 3번 쿼리의 상황에서 감염된 컴퓨터의 수를 반복문을 통해 세어 준다면 $\mathcal{O}(NQ)$의 시간 복잡도를 갖습니다. 이는 너무 느립니다.
- $1, 2$번 쿼리로 인해 답이 변화할 때마다 이를 반영하는 방법을 사용합니다.
- 초기에 답은 전체 컴퓨터의 수 N에서 시작하여 컴퓨터가 감염되지 않았던 컴퓨터를 새로 감염시키면 답이 1 감소하고, 감염되었던 컴퓨터를 치료하면 답이 1 증가합니다.

B. K512에 바이러스 퍼뜨리기



- 감염된 컴퓨터 목록을 관리할 때, array와 같은 자료구조를 활용하면 $\mathcal{O}(Q)$의 시간 복잡도로 문제를 해결할 수 있습니다.
- 감염된 컴퓨터 목록을 관리할 때, set/map의 자료구조를 활용하고 size()의 멤버 함수를 사용하는 방법도 존재합니다. 이는 $\mathcal{O}(Q \log N)$의 시간 복잡도를 갖습니다.

C. 근수

`math, combinatorics`
출제진 의도 – **Easy**

- 제출 171번, 정답 36명 (정답률 21.637%)
- 처음 푼 사람: **하건영**, 5분
- 출제자: 이승형 `tmdgud0617`

C. 근수



- 2가 연속으로 k개 존재하는 문자열을 k-근수라고 합시다.
- k-근수의 점수는 어떻게 계산될까요?
- k-근수에는 서로 다른 근수가 k개, 근근수가 $k-1$개, 근근근수가 $k-2$개 ... k-근수가 1개 포함되어 있습니다.
- 점수의 합은 $1 \times k + 2 \times (k-1) + ... + k \times 1$이 될 것입니다.

C. 근수



- 이를 수식으로 나타내면 다음과 같습니다.

$$\sum_{i=1}^{k} (i \times (k+1-i))$$

- 위 식을 정리하고 급수의 부분합을 구해줍니다.
- 반복문을 통해 문자열에 포함된 연속된 2로 이루어진 k-근수들을 모두 찾아 점수를 구합니다. 이는 $\mathcal{O}(N)$의 시간 복잡도를 갖습니다.



D. FA 수의 진

`greedy`
출제진 의도 – **Easy**

- 제출 56번, 정답 35명 (정답률 62.500%)
- 처음 푼 사람: **손기령**, 15분
- 출제자: 유호영 `tkfkdd159323`

D. FA수의 진

- 각 구역에 도착할 때마다 최대한 빨리 가는게 항상 이득입니다.
- 따라서 각 구역에 도착했을 때마다 신호등을 기다렸을 때와 육교를 건넜을 때중 더 빠른 방법을 매번 선택하도록 구현하면 됩니다.

E. Stacking Sticks

`tree_set`, `hash_set`
출제진 의도 – **Easy**

- 제출 62번, 정답 27명 (정답률 43.548%)
- 처음 푼 사람: **김동건**, 18분
- 출제자: 채성우 `lem0nad3`

E. Stacking Sticks



- 세로로 긴 4×1 블럭을 떨어뜨리는 경우부터 생각해 봅시다.
- 블럭은 $x = i$를 차지하고 있는 칸들 중, 최대 y 좌표의 바로 한 칸 위에 쌓입니다.
- 가로로 긴 1×4 블럭을 떨어뜨리는 경우도 비슷합니다.
- 블럭은 $x = i$, $x = i + 1$, $x = i + 2$, $x = i + 3$을 차지하고 있는 칸들 중, 최대 y 좌표의 바로 한 칸 위에 쌓입니다.

E. Stacking Sticks



- p를 다음과 같이 정의합시다.
- $p_i = (x = i$를 차지하고 있는 칸들 중, 가장 높은 y 좌표$)$
- 1번 쿼리의 경우, 최대 y 좌표는 $M = \max(p_i,\ p_{i+1},\ p_{i+2},\ p_{i+3})$ 입니다.
- $p_i,\ p_{i+1},\ p_{i+2},\ p_{i+3}$를 모두 $M + 1$으로 바꿔주면 됩니다.
- 2번 쿼리의 경우, 최대 y 좌표는 p_i 입니다.
- p_i를 $p_i + 4$로 바꿔주면 됩니다.

E. Stacking Sticks



- 3번 쿼리가 들어올 때마다, p_i를 출력해주면 됩니다.
- 좌표의 범위가 크기 때문에, map을 이용하여 p를 관리하면 됩니다.
- 접근하는 인덱스는 최대 $4Q$개 존재합니다.
- 전체 시간복잡도는 $\mathcal{O}(Q \log Q)$입니다.



F. Yunny's Trip

`tree_set`, `hash_set`
출제진 의도 – **Normal**

- 제출 93번, 정답 16명 (정답률 17.204%)
- 처음 푼 사람: **손기령**, 36분
- 출제자: 송근수 `hulahula3247`

F. Yunny's Trip



- 우선 K 값은 5 를 초과하지 않기 때문에 아이템은 많아야 2번 사용할 수 있습니다.
- 아이템을 각각 $0, 1, 2$ 번 사용하는 경우를 나누어서 생각해 봅시다.

F. Yunny's Trip



- 아이템을 0번 사용하는 경우입니다.
- 아이템을 사용하지 않기 때문에 직접 한 칸씩 이동해서 목적지에 도착해야 합니다.
- 이때 기력 요구 값은 $|E_x| + |E_y|$ 입니다.

F. Yunny's Trip



- 아이템을 1 번 사용하는 경우입니다.
- 어느 한 아이템을 사용하고 나서 목적지까지의 거리를 (D_x, D_y) 라고 합시다.
- 이때 한 칸씩 이동해서 목적지까지 도달하기 위한 기력은 $|D_x| + |D_y|$ 입니다.
- 아이템을 사용할 때 2의 기력을 소모하므로 필요한 총 기력 요구 값은 $|D_x| + |D_y| + 2$ 입니다.
- 위 과정을 N 개의 모든 아이템에 대해서 시도해 보면 됩니다. 시간복잡도는 $\mathcal{O}(N)$ 입니다.

F. Yunny's Trip



- 아이템을 2번 사용하는 경우입니다.
- 아이템을 1번 사용하는 경우와 비슷하게 구해낼 수 있습니다. 하지만 아이템을 2개 고정해야 하므로 시간복잡도가 $\mathcal{O}(N^2)$가 됩니다. 이는 너무 느립니다.
- 이를 해결하기 위해 다음과 같은 관찰을 해봅시다.
- 관찰1. 최대 기력 K는 5이하이고, 아이템을 2번 사용할 때 소모되는 기력은 4이니, 아이템을 2번 사용하고 나서는 최대 한 칸까지만 이동할 수 있습니다.

F. Yunny's Trip



- 우선 아이템 한 개를 사용하고 이때 목적지까지의 거리를 (D_x, D_y) 라고 합시다.
- (D_x, D_y) 아이템이 있으면 기력 요구 값은 4 입니다.
- (D_x, D_y) 와 좌표가 1 차이 나는 아이템이 있으면 기력 요구 값은 5 입니다.
- 그 외에는 기력 요구 값이 5 를 넘어가는 경우입니다. 탐색하지 않아도 됩니다.
- 특정 값을 만족시키는 아이템이 있는지는 std::set 이나 이분 탐색을 활용하면 됩니다.
- 위 과정을 N 개의 모든 아이템에 대해서 시도해 보면 됩니다. 특정 아이템이 있는지 확인할 때 $\mathcal{O}(\log N)$ 의 시간이 걸리므로 시간복잡도는 $\mathcal{O}(N \log N)$ 입니다.
- 총 시간복잡도 $\mathcal{O}(N \log N)$ 에 문제를 해결할 수 있습니다.



G. Gridev’s Protocol

`graph_traversal`, `simulation`
출제진 의도 – **Hard**

- 제출 52번, 정답 7명 (정답률 13.462%)
- 처음 푼 사람: **강효규**, 64분
- 출제자: 송근수 `hulahula3247`

- 같은 행동을 연속으로 2번 이상 해도 바뀌는 것은 없습니다.
- 따라서 행동1과 행동2를 번갈아 가며 하는 게 최적입니다.
- 행동1을 먼저 하는 경우, 행동2를 먼저 하는 경우, 2가지 경우만 확인하면 됩니다.



- 이제 적절하게 시뮬레이션을 돌려보는 일만 남았습니다.
- 짧은 구현을 위하여 행동1을 먼저 하는 경우만 구현합시다.
- 행동2를 먼저 하는 경우는 오류의 x, y 좌표만 서로 바꿔주면 위의 구현을 재활용해서 시뮬레이션을 돌려볼 수 있습니다.



- 어떻게 구현해야 할까요?
- 격자판의 크기가 N^2 이므로 배열에다가 오류의 위치를 저장하면 안 됩니다.
- N^2 사이즈의 bool 배열을 잡으려면 $N = 2 \times 10^5$ 기준으로 약 4GB의 메모리와 $\mathcal{O}(N^2)$ 의 시간이 필요합니다. 이는 너무 무겁습니다.
- 매 행동마다 모든 가로줄, 세로줄을 훑어도 안 됩니다.
- 오류를 전부 지우기 위해서 최악의 경우 $\mathcal{O}(M)$ 번의 행동을 요구합니다. 모든 줄을 훑으면 최소 $\mathcal{O}(NM)$ 의 시간복잡도가 나오게 됩니다. 이는 너무 느립니다.

- 위에 내용을 준수하면서 시뮬레이션을 해야 합니다.
- 그래프에서의 위상 정렬과 비슷하게 구현을 하면 빠른 시간안에 시뮬레이션을 끝낼 수 있습니다.



- 1. 가로줄, 세로줄을 각각 그래프에서의 정점, 오류를 그래프에서의 간선으로 봅니다.
- (a, b)에 오류가 있으면 a번 가로줄 정점과 b번 세로줄 정점을 간선으로 연결해 줍니다.
- 이는 오류를 효율적으로 저장할 수 있을뿐더러 뒤에 있을 구현을 효율적으로 할 수 있게 해줍니다.



- 2. 각 가로줄, 세로줄은 최초로 오류가 K 이하가 될 때, 한 번만 방문하고 넘어갑니다.
- 오류가 K 개 이하인 가로줄, 세로줄(정점)을 저장소에 따로 저장합니다.
- 이제 저장소에 담긴 정점을 확인하면서 그 정점과 간선으로 연결된 정점의 오류 개수를 1개 낮춰줍니다.
- 이때 연결된 정점을 방문하지 않았고, 그 정점의 오류 개수가 K 개 이하라면 그 정점을 저장소에 추가해줍니다. 이를 반복합니다.
- 위와 같은 방식으로 시뮬레이션을 돌리면 각 가로줄, 세로줄을 최대 한 번씩만 볼 수 있습니다.



- 저장소에 정점이 없으면 탐색을 종료하면 됩니다.
- 모든 정점을 방문하지 않았다면 모든 오류를 지우는 것이 불가능한 경우입니다.
- 가로줄과 세로줄을 분리해서 저장하고, 행동 턴에 맞춰서 지워나가야 하는 점에 주의합시다.
- 위를 구현하기 위해서 출제자/검수자들의 코드에는 vector, set, priority_queue 등 다양한 자료구조가 등장하였으며 구현 디테일도 전부 달랐습니다. 자신만의 구현을 하시면 됩니다:)
- 시간복잡도는 구현에 따라 $\mathcal{O}(N+M)$ 혹은 $\mathcal{O}((N+M)\log M)$ 입니다.



H. 앳코더 스터디

`greedy`, `dp`, `two_pointer`
출제진 의도 – **Hard**

- 제출 24번, 정답 1명 (정답률 2.381%)
- 처음 푼 사람: **손기령**, 163분
- 출제자: 유호영 `tkfkdd159323`

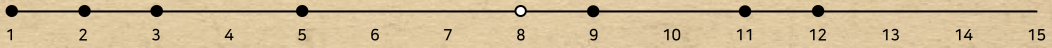


- 오른쪽으로 최대 $N-1$번, 왼쪽으로 최대 $N-1$번까지는 움직여야 할 수도 있지만 각 방향으로 그 이상 움직이는 최적해는 존재할 수 없습니다.
- 각 방향으로 움직이는 도중에 2번 보법을 사용하는 게 이득인지 아닌지를 판단해야 하는데, 관찰과 조건 분기를 사용해서 greedy하게 문제를 해결할 수도 있고, dp를 사용해도 동일한 시간복잡도로 해결할 수 있습니다.

H. 앳코더 스터디

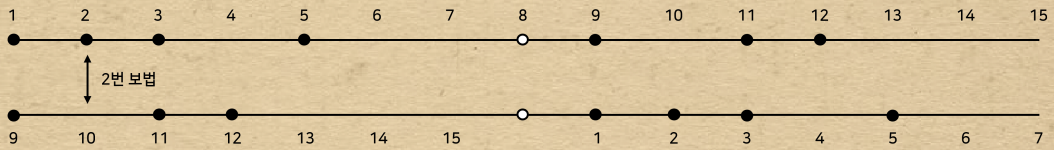


- 문제에서 주어진 예제 입력 2 번을 그림으로 표현하면 아래와 같습니다.



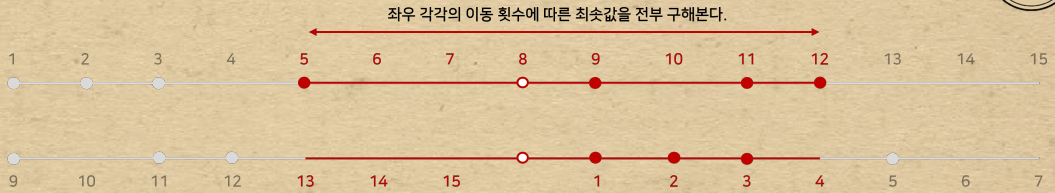
- 위 상태에서 직접 2 번 보법의 손익을 계산하는 건 어려우므로 모델링을 약간 바꿔줍니다.

H. 앳코더 스터디



- 위와 같은 형태로 바꿔주면, 시작지점으로부터 오른쪽 혹은 왼쪽으로 계속 이동하면서 현재 위쪽에 있는지 아래쪽에 있는지 정도만 추가로 관리하면서 최적값을 구할 수 있습니다.
- 여기서 최적해는 dp를 사용해서 구해도 괜찮고, greedy하게 구해도 괜찮습니다.

H. 앳코더 스터디



- 좌우로 각각 몇칸씩 이동했는지에 따라 최적해가 바뀌므로, 그 부분만 투포인터로 관리해줍니다.
- 이때 좌우의 길이가 총 N 칸이 되게만 관리하면 필요없는 곳까지 이동할 수 있으므로 필요한 만큼만 이동할 수 있게 포인터 관리를 잘 해줘야 합니다.



I. Hula's Cardgame

`game_theory`, `lca`

출제진 의도 – **Challenging**

- 제출 2번, 정답 0명 (정답률 0.000%)
- 출제자: 송근수 `hulahula3247`

- 우선 아래의 관찰과 지식이 있으면 문제를 해결할 수 있습니다.
- 관찰1. 근수가 테이블에 있는 카드 3개중 2개를 미리 골라두면, 승형이가 어떤 행동을 하든 근수는 미리 골라둔 2개의 카드 중 하나는 반드시 고를 수 있다.
- 증명: 승형이는 카드를 1장밖에 제거하지 못하기 때문입니다.
- 지식1. 트리에서 3개의 정점 중 2개를 골라 얻은 lca(최소 공통 조상)들은 서로 부모-자식 관계이다.
- 증명: 3개의 정점을 각각 a, b, c라고 하고 lca(a, b)와 lca(b, c)가 서로 부모-자식 관계가 아닌 트리를 그려보면 사이클이 만들어집니다. 이는 트리의 정의에 모순되므로 지식1은 참입니다.



- 위의 관찰과 지식을 가지고 게임을 모델링해 봅시다.
- 근수가 게임에서 패배한 경우는 자기 자신으로 돌아오는 경우입니다. 이를 $N+1$ 번째 노드로 가는 거로 둡니다.
- 양쪽이 완벽하게 플레이하는 경우 E 값과 관계없이 정점 s_i 에서 반드시 방문하는 정점 e_i 가 있다고 가정해봅시다.
- 모든 s_i 에 대해 정점 e_i 가 존재하는 경우 $N+1$ 을 루트로 하는 트리가 만들어지게 됩니다.
- 이러한 트리를 만들 수 있는지 알아봅시다.



- N 번째 테이블부터 역순으로 보면서 트리를 구성해봅시다.
- 현재 테이블의 카드가 a, b, c라고 하면 관찰1에 의해서 근수는 lca(a, b), lca(a, c), lca(b, c) 중 하나로 항상 이동할 수 있습니다.
- 지식1에 의해서 E 값과 관계없이 저 3개의 노드 중 가장 깊이가 깊은 노드로 이동하는 것이 항상 이득입니다.
- E 값과 관계없이 모든 정점에서 이동해야 할 정점이 정해지므로 트리를 항상 만들 수 있습니다.



- 1번 노드와 그 부모 노드들은 근수가 최선을 다한다면 항상 방문할 수 있습니다.
- 1번 노드와 그 부모 노드가 아닌 노드들은 승형이가 최선을 다한다면 항상 방문하지 못합니다.
- 트리는 두 노드의 가장 깊은 lca로만 이루어져 있기에 그보다 더 깊은 노드는 카드 한 장을 밴하면 반드시 도달하지 못하기 때문입니다.
- lca 알고리즘으로 정점마다 $\mathcal{O}(\log N)$의 시간마다 lca를 구하고 업데이트할 수 있습니다.
- 한번에 희소 배열을 채우는 일반적인 lca 알고리즘과 다르게 이 문제에서는 부모가 정해질 때마다 희소 배열을 채워줘야 합니다.
- 총 시간복잡도는 $\mathcal{O}(N \log N)$입니다.



- 위의 풀이보다 좀 더 간단한 출제자의 별해를 간략하게 소개합니다.
- a 번 테이블에 b 번 카드가 있는 경우, b 번 저장소에 a 를 저장합니다.
- b 번 저장소에 어떠한 원소 a 가 2 개 이상 있는 경우 a 에서 항상 b 로 갈 수 있습니다. b 저장소에 a 저장소에 있는 원소들을 전부 추가해줍니다.
- 이제 1 번 테이블부터 순서대로 원소들을 확인해 주면서 2개 이상 들어있는 원소들을 합쳐주면 됩니다. x 저장소에 1 이 2 개 이상 있으면 1 번 테이블에서는 반드시 x 로 갈 수 있다는 뜻입니다.
- 저장소들은 최대 $N-1$ 번 합쳐지고 원소는 $3N$ 개입니다. set/map, dsu, small to large 테크닉을 활용해서 저장소들을 잘 관리하면 $\mathcal{O}(N \log^2 N)$ 에 문제를 해결할 수 있습니다.
- 이 풀이의 정당성과 몇몇 디테일은 생략하였습니다.



J. Lem0nad3's Bar

`dp_bitfield`

출제진 의도 – **Challenging**

- 제출 2번, 정답 0명 (정답률 0.000%)
- 출제자: 송근수 `hulahula3247`



- 두 가지의 중요한 관찰을 해봅시다.
- 관찰1. 레모네이드를 마시는 시각은 오직 레모네이드가 나온 시각 t_i 들 뿐이다.
- t_i 가 아닌 시각에 어떠한 레모네이드를 마신다면 그 시각보다 바로 앞에 있는 t_i 에 그 레모네이드를 마시는 것이 항상 이득입니다.
- 관찰2. 레모네이드의 청량감이 빠지는 정도는 오직 시간에 비례한다.
- 청량감이 빠지는 정도를 계산할 때는, 어떤 레모네이드인지 기록할 필요가 없습니다.

J. Lem0nad3's Bar



- 이제 시간 순서대로 레모네이드를 마실지 말지 결정하면서 레모네이드를 마실 상태를 비트필드로 관리합시다.
- 예를 들어 $K = 5$ 이고 현재 비트필드가 11010 이라면 현재 $1, 2, 4$ 번째로 마실 레모네이드를 결정했다는 뜻입니다. 시간으로 줄어드는 행복함을 신경쓰지 않고 행복함을 올리고 시작합니다.
- 위에 상태에서 $1, 2$ 번째로 마시는 레모네이드는 별다른 비용이 발생하지 않습니다.
- 4 번째로 마실 레모네이드는 실제로는 지금 마시지 못합니다. 3 번째로 마실 레모네이드를 먼저 마시고 마셔야만 합니다.
- 3 번째 비트가 채워질 때까지 $(K + 1 - 4) \times \Delta T$ 만큼 행복함을 줄여줘야 합니다.
- 위의 원리를 응용하면 모든 비트에 대해서 줄어드는 행복함을 계산해 낼 수 있습니다.



- 지금까지의 정보를 바탕으로 dp 식을 정의해봅시다.
- $dp(idx, state) = V$, 현재 $idx - 1$ 번까지의 레모네이드를 봤고, 현재 레모네이드를 마신 상태가 $state$ 일 때, 앞으로 얻을 수 있는 행복함의 최댓값 V.
- idx 번의 레모네이드를 k 번째에 마시기로 결정했으면 $state$ 에서 k 번째 비트를 1 로 바꿔주면 됩니다.
- 다음 $idx + 1$ 로 넘어갈 때 패널티를 받는 i 번째 비트들을 구하고, 그 모든 i 에 대해서 $(K + 1 - i) \times \Delta T$ 만큼 값을 빼줘야 합니다.



- 레모네이드를 시간 순서대로 정렬하는 걸 까먹지 않도록 주의해야 합니다.
- 위의 dp 식을 통하여 $\mathcal{O}(NK^2 2^K)$ 의 시간복잡도로 정답을 구해낼 수 있습니다.
- 모든 비트에 대해서 패널티를 전처리하는 최적화를 통해 $\mathcal{O}(NK2^K)$ 로 시간복잡도를 개선할 수 있습니다.
- 비재귀 등의 최적화로 더 빠른 코드를 작성할 수 있습니다.
- c/c++ 은 어떠한 최적화를 거치지 않아도 시간 안에 통과가 됩니다. 파이썬은 소개한 2 가지 최적화 중 적어도 하나의 최적화를 거쳐야지만 시간 초과를 받지 않습니다.