ZOAC 2025 풀이

Official Solutions

by

한양대학교 ERICA 알고리즘 학회 0&1

2025년 11월 15일



문제		의도한 난이도	출제자
A	추석은 언제나 좋아	**Easy**	김재민 jaemkim01
B	01MEX	**Normal**	김동건 firework72
C	ZOAC 8	**Normal**	김재민 jaemkim01
D	백준 빙고 스피드러너	**Normal**	김범수 bumsoo0515
E	닷지 테이블	**Hard**	이준호 rhdqor213
F	기열과 쿼리	**Hard**	장성혁 rabbit007
G	최강 테토 뚱뽑	**Challenging**	김동우 gildong2
H	제설 작업	**Challenging**	오익준 putdata



A. 추석은 언제나 좋아

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

- ✓ 제출 31번, 정답 14팀 (정답률 45.161%)
- ✓ 처음 푼 팀: **onnu396**, 3분
- ✓ 출제자: 김재민 `jaemkim01`

A. 추석은 언제나 좋아



- 입력으로 받은 금액 N에 대해 조건문으로 금액 구간을 나누고,
- 해당 구간에 맞는 비율(5%, 10%, 15%, 20%)을 적용해 공제액 `sol`을 계산한다.
- 계산된 금액 `sol`과 남은 금액 $N - \texttt{sol}$을 출력하는 단순한 누진 공제 계산 프로그램이다.

B. 01MEX

`math`
출제진 의도 – **Normal**

- 제출 15번, 정답 3팀 (정답률 20.000%)
- 처음 푼 팀: **초비상**, 65분
- 출제자: 김동건`firework72`

- 가장 간단하게 생각할 수 있는 방법은 모든 (i, j) 쌍에 대하여 2중 for문으로 $a_i, a_{i+1}, \ldots, a_j$ 의 mex값을 구한 후 모두 더하는 것입니다.
- 하지만 이 방법은 $\mathcal{O}\left(N^2\right)$ 의 시간복잡도가 소요되므로 제한 시간 내에 해결할 수 없습니다.

- 문제의 제한 조건 내에서 어떤 부분 수열의 mex값은 $0, 1, 2$ 밖에 존재하지 않는다는 점에 주목합시다.
- 구체적으로는 다음과 같습니다.
  - 부분 수열의 원소로 1 만 포함되어 있다면 mex값은 0 입니다.
  - 부분 수열의 원소로 0 만 포함되어 있다면 mex값은 1 입니다.
  - 부분 수열의 원소로 0 과 1 이 모두 포함되어 있다면 mex값은 2 입니다.

- $f(i, j)$를 $a_i, a_{i+1}, \ldots, a_j$의 mex값으로 정의합시다.
- 고정된 $r\ (1 \leq r \leq N)$에 대해, a_r의 값에 따라 $X_r = \sum_{k=1}^{r} f(k, r)$의 값을 빠르게 계산할 수 있습니다.

- ✓ $a_r = 0$인 경우
  - $i < r$과 $a_i = 1$을 만족하는 i 중 가장 큰 값을 l이라고 합시다.
  - 만약 그러한 l이 없다면, $1 \le k \le r$을 만족하는 모든 k에 대해 $f(k, r) = 1$이므로 $X_r = r$입니다.
  - 만약 그러한 l이 존재한다면, $1 \le k \le l$을 만족하는 모든 k에 대해 $f(k, r) = 2$이고, $l + 1 \le k \le r$을 만족하는 모든 k에 대해 $f(k, r) = 1$이므로 $X_r = l + r$입니다.



- ✓ $a_r = 1$인 경우
  - $i < r$과 $a_i = 0$을 만족하는 i 중 가장 큰 값을 l이라고 합시다.
  - 만약 그러한 l이 없다면, $1 \le k \le r$을 만족하는 모든 k에 대해 $f(k, r) = 0$이므로 $X_r = 0$입니다.
  - 만약 그러한 l이 존재한다면, $1 \le k \le l$을 만족하는 모든 k에 대해 $f(k, r) = 2$이고, $l + 1 \le k \le r$을 만족하는 모든 k에 대해 $f(k, r) = 0$이므로 $X_r = 2 \times l$입니다.

- 최종적으로 구하고자 하는 답은 $\sum_{i=1}^{N} X_i$ 입니다.
- r 을 $1, 2, \ldots, N$ 순으로 순회하면 가장 오른쪽에 있는 0 과 1 의 위칫값을 저장해둘 수 있으므로 X_r 을 $\mathcal{O}(1)$ 에 구할 수 있습니다.
- 따라서 최종 시간복잡도는 $\mathcal{O}(N)$ 입니다.

C. ZOAC 8

`number_theory`, `primality_test`, `sieve`, `prefix_sum`
출제진 의도 – **Normal**

- ✓ 제출 38번, 정답 8팀 (정답률 21.053%)
- ✓ 처음 푼 팀: **사과게임 144점**, 35분
- ✓ 출제자: 김재민[jaemkim01]



- 에라토스테네스의 체를 이용해 10만 미만의 소수를 미리 판별하고,
- 각 숫자까지의 소수 개수를 담은 누적 합 배열을 만든다.
- 참가자의 ID 두 구간 사이에 존재하는 소수의 개수를 점수로 계산하고,
- 가장 높은 점수(동점 시 이름 사전순)와 가장 낮은 점수를 가진 참가자를 각각 선발한다.
- 최종적으로 1등 참가자 이름과 꼴찌 참가자 이름을 출력하는 소수 기반 점수 산정 프로그램이다.

D. 백준 빙고 스피드러너

`implementation, simulation`
출제진 의도 – **Normal**

- ✓ 제출 0번, 정답 0팀 (정답률 -.-%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 김범수 `bumsoo0515`

- 지문에 설명된 스피드런 전략을 잘 구현해주면 되는 문제입니다.
- 행, 열, 대각선마다 남아 있는 시간의 합을 저장해둡니다.
- 배열을 만들어 행의 합을 인덱스 $1 \sim N$, 열을 $N+1 \sim 2N$, 그리고 두 대각선을 각각 $2N+1$, $2N+2$에 저장하면,
  우선순위를 복잡하게 구현하지 않아도 미완성인 줄 중 가장 앞에 있는 최솟값을 구함으로써 줄을 선택할 수 있습니다.
- 이렇게 선택한 줄의 시간을 현재까지의 시간에 누적시켜 줍니다.

- 선택한 줄에 포함된 N 개의 칸을 순회하며, 해당 칸이 포함된 모든 줄마다 남은 시간에서 그 칸의 시간을 빼줍니다.
  - 각 칸 A_{ij} 가 포함될 수 있는 줄은 i 행, j 열, 그리고 두 대각선으로 최대 4 개입니다.
- 다 빼준 후 칸을 0 으로 설정하거나 다른 표시를 해 나중에 중복해서 빼는 일이 없도록 합시다.
- 완성한 줄의 합도 나중에 최솟값을 구할 때 포함되지 않도록 처리하여 줍시다.
- 한번에 여러 줄이 완성되는 경우도 저장된 합이 0 이 된 줄을 세주며 검사할 수 있습니다.
- 이제 줄 선택을 반복하며 갱신되는 빙고 수에 따라 누적 시간을 적절히 출력하면 됩니다.
- 시간 복잡도는 $\mathcal{O}\left(N^2\right)$ 입니다.

E. 닷지 테이블

dp
출제진 의도 – **Hard**

- 제출 0번, 정답 0팀 (정답률 -.-%)
- 처음 푼 팀: -, -분
- 출제자: 이준호rhdqor213

E. 닷지 테이블



- 가로, 세로, 시간을 축으로 3차원 배열을 생각합니다.
- 각 공격 형식이 주어지면, 해당 위치에 공격의 지속시간 p를 채웁니다.
- 이후 시간 흐름에 따라 각 칸에 공격이 있는 시점을 0, 또는 1로 모두 채워 넣습니다.
- 이제 공격 유무에 따른 3차원 0-1 배열을 기반으로 dp를 통해 답을 구합니다.
- 시간복잡도는 $\mathcal{O}(K)$ 입니다.

F. 기열과 쿼리

`tree_set`
출제진 의도 – **Hard**

- 제출 0번, 정답 0팀 (정답률 -.-%)
- 처음 푼 팀: -, -분
- 출제자: 장성혁 `rabbit007`

- ✓ set::lower_bound 활용: set은 자동으로 정렬을 유지하므로, lower_bound를 사용해 직속 선임을 효율적으로 찾을 수 있습니다. 이는 기열 연산에서 핵심적인 역할을 합니다.
- ✓ C++의 set은 균형 이진 탐색 트리(보통 레드-블랙 트리)로 구현되어, 요소를 정렬된 상태로 유지하며 삽입, 삭제, 탐색 연산을 효율적으로 지원합니다.



✓ 쿼리:

- Type 1 (갱신):
  - set에서 삭제: $\mathcal{O}(\log N)$.
  - set에 삽입: $\mathcal{O}(\log N)$.
  - 총: $\mathcal{O}(\log N)$.
- Type 2 (기열):
  - 두 번의 lower_bound: 각각 $\mathcal{O}(\log N)$.
  - 최대 두 번의 삭제와 두 번의 삽입(군인과 선임): 각각 $\mathcal{O}(\log N)$.
  - 총: $\mathcal{O}(\log N)$.
- Q개의 쿼리 총합: $\mathcal{O}(Q \log N)$.
- 전체 복잡도: $\mathcal{O}((N+Q)\log N)$.

G. 최강 테토 뚱뽑

`dp_tree`, `prefix_sum`, `set`, `smaller_to_larger`
출제진 의도 – **Challenging**

- ✓ 제출 10번, 정답 0팀 (정답률 0.000%)
- ✓ 처음 푼 팀: -, -분
- ✓ 출제자: 김동우`gi1dong2`



- ✓ 올바른 괄호 문자열을 판별하는 방법에는 누적 합을 사용하는 방법이 있습니다.
- ✓ 여는 괄호를 $+1$, 닫는 괄호를 -1로 치환합니다.
- ✓ 올바른 괄호 문자열은 다음 두 조건을 만족해야 합니다.
  - 괄호 전체의 합이 0입니다.
  - 모든 구간의 누적 합이 0 이상입니다.
- ✓ 해당 방법을 트리에 적용해야 합니다.



- ✓ 모든 정점에서의 경우를 확인하는 것은 시간이 오래 걸리므로 우선 루트인 1 번부터 말단 정점까지 내려가며 누적 합을 계산해줍니다.
- ✓ 여기서 테토 정점이 될 수 있는 정점은 여는 괄호인 정점들 뿐입니다.
- ✓ 그리고 도달할 수 있는 말단 정점 중 닫는 괄호인 말단 정점이 있어야 합니다.
- ✓ 만약 말단 정점이 아닌 여는 괄호 정점의 누적 합 값이 x 일 때, 도달할 수 있는 닫는 괄호 말단 정점의 누적 합 값이 $x-1$ 이고, 경로 사이에 누적 합 값이 모두 $x-2$ 이상이면 이 정점은 테토 정점입니다.



- ✓ 누적 합 값 x의 범위만큼의 테토 정점 후보 집합들 sets와 정점의 범위만큼의 후보 제외 집합들 unvalid를 만듭니다.
- ✓ 루트부터 dfs를 시작하여 정점 v에 해당하는 경우를 처리합니다.
- ✓ v가 여는 괄호 정점일 때, v를 sets$[x]$에 저장합니다.
- ✓ 말단 정점이 아닌 닫는 괄호 정점일 때, sets$[x+2]$의 정점들을 unvalid$[v]$로 옮깁니다.
- ✓ 닫는 괄호인 말단 정점일 때, sets$[x+1]$에 있는 정점들이 테토 정점입니다.
- ✓ unvalid$[v]$에 있는 정점들은 v의 깊이 우선 탐색이 종료될 때 다시 sets$[x+2]$로 옮깁니다.
- ✓ 정점들을 다른 집합으로 옮길 때 시간이 오래 걸릴 수 있으므로 smaller to larger을 사용합니다.

H. 제설 작업

`pbs`, `segtree`, `disjoint_set`
출제진 의도 – **Challenging**

- 제출 0번, 정답 0팀 (정답률 -.-%)
- 처음 푼 팀: -, -분
- 출제자: 오익준`putdata`

H. 제설 작업

문제 요약 (1/2)



- 도로 구간은 $1 \sim N$ 까지 있고, 각 구간 i 에는 눈 S_i 가 쌓여 있다.
- 작업 t: 구간 $[L_t, R_t]$ 에 적재량 C_t 인 제설차 한 대를 투입한다.
- 제설차는 오른쪽으로만 이동하고, 한 구간의 눈을 모두 치운 뒤에야 다음 구간으로 이동한다.
- 눈은 자연스럽게 녹거나 새로 쌓이지 않고, 제설로 치운 양만 고려한다.

문제 요약 (2/2)

- 작업 t까지 끝났을 때, i번 구간에 남은 눈을 $S_i^{(t)}$라고 하자.
- 이때 구간 $[A, B]$에서 치워진 눈의 총량은

$$\sum_{i=A}^{B} \left(S_i - S_i^{(t)}\right)$$

로 표현된다.

- 쿼리 (A, B, T)는 이 값이 T 이상이 되는 가장 작은 작업 번호 t를 구하는 문제이다.
- 그런 t가 없다면 -1을 출력한다.

단순 접근

- 쿼리 i에 대해 함수

$$f_i(t) = \sum_{k=A_i}^{B_i} \left(S_k - S_k^{(t)}\right)$$

를 생각해 보자.

- 작업 번호가 늘어날수록 눈은 계속 치워지므로

$$t_1 \le t_2 \Rightarrow f_i(t_1) \le f_i(t_2)$$

가 성립한다.

- 따라서 각 쿼리마다 $f_i(t) \ge T_i$가 처음으로 성립하는 t를 이분 탐색으로 찾고 싶어진다.

단순 접근 한계



- 어떤 t를 정할 때마다, $1 \sim t$번 작업을 처음부터 끝까지 모두 시뮬레이션해야 한다.
- 순진한 시뮬레이션 한 번의 비용은 대략 $\mathcal{O}(N+M)$ 이다.
- 쿼리가 Q 개이고, 각 쿼리에 대해 작업 번호에 대해 $\log M$ 번 이분 탐색을 한다.
- 전체 시간은 $\mathcal{O}(Q \cdot (N+M) \log M)$ 정도가 되어, $N, M, Q \leq 2 \cdot 10^5$ 에서는 약 10^{12} 연산 수준이 되어 버린다.
- 시간 초과가 확실한 접근이다.

아이디어 1: 병렬 이분 탐색



- ✓ 문제점: 같은 종류의 시뮬레이션을 쿼리마다 따로 반복하고 있다는 점이다.
- ✓ 발상 전환: "작업 번호 t"를 기준으로 여러 쿼리를 묶어서 처리하자.
- ✓ 병렬 이분 탐색(parallel binary search):
  - 각 쿼리에 대해 이분 탐색을 진행하되,
  - 같은 중간값 t를 사용하는 쿼리들끼리 시뮬레이션을 공유하는 기법이다.

병렬 이분 탐색 구조



- ✓ 각 쿼리 i에 대해 정답 후보 구간 $[lo_i, hi_i]$를 둔다. (처음엔 $[1, M]$)
- ✓ 아직 $lo_i \le hi_i$인 쿼리에 대해

$$mid_i = \left\lfloor \frac{lo_i + hi_i}{2} \right\rfloor$$

  를 계산한다.
- ✓ 같은 mid를 갖는 쿼리들을 "작업 번호가 mid인 그룹"에 모아 둔다.
- ✓ 이후 한 라운드에서 $t = 1 \sim M$까지 작업을 시뮬레이션하며, 시점 t에서 이 그룹에 속한 쿼리들을 한꺼번에 판정한다.

병렬 이분 탐색에서의 판정

- 시뮬레이션이 진행되는 동안, “지금까지 각 구간에서 치운 눈의 양”을 계속 유지한다고 하자.
- 작업 t까지 끝났을 때, 어떤 쿼리 (A_i, B_i, T_i)에 대해 구간 $[A_i, B_i]$에서 치운 눈의 총합을 계산한다.
- 이 값이 T_i 이상이면, 작업 번호를 더 줄여 볼 수 있으므로 hi_i를 줄이고 현재의 mid_i를 답 후보로 저장한다.
- 아직 T_i에 도달하지 못했다면, 더 뒤쪽 작업을 봐야 하므로 lo_i를 늘린다.

아이디어 2: 이미 치운 구간 건너뛰기



- ✓ 한 라운드 안에서도, 작업 하나를 단순히 구현하면 $[L_t, R_t]$를 칸마다 훑어야 해서 최악의 경우 $\mathcal{O}(N)$ 시간이 든다.
- ✓ 관찰: 한 라운드에서 어떤 칸의 눈이 완전히 치워지면, 이후의 작업들에서는 그 칸을 다시 볼 필요가 없다.
- ✓ 즉, “아직 눈이 남아 있는 칸만” 빠르게 건너다닐 방법이 있다면 전체 비용을 크게 줄일 수 있다.

DSU로 다음 미청소 칸 관리하기



- 각 위치마다 “이 위치 이상에서 아직 눈이 남아 있는 가장 왼쪽 구간이 어디인지”를 가리키는 정보를 유지한다.
- 어떤 구간의 눈을 모두 치웠다면, 그 구간은 더 이상 유효하지 않으므로 “바로 오른쪽에 있는 다음 후보 구간”을 대신 가리키도록 연결을 바꾼다.
- 이렇게 하면, 특정 위치에서 시작할 때 실제로 눈이 남아 있는 구간까지 한 칸씩 이동하지 않고도 한 번에 건너뛸 수 있다.
- 이 구조를 Union-Find(Disjoint Set Union)으로 구현하면, 전체 라운드 동안의 이동 비용을 거의 선형에 가깝게 만들 수 있다.

DSU를 이용한 작업 시뮬레이션



- ✓ 라운드마다 “각 구간에 남아 있는 눈의 양”을 저장하는 배열을 하나 둔다.
- ✓ 작업 t를 처리할 때, 먼저 이 작업의 왼쪽 끝에서 시작해 “아직 눈이 남아 있는 첫 번째 구간”으로 점프한다.
- ✓ 그 위치에서 남은 용량만큼 눈을 치우고, 그 칸을 완전히 비우면 방금 설명한 연결 정보를 이용해 자연스럽게 그 다음 후보 칸으로 건너뛴다.
- ✓ 한 칸이 완전히 치워지는 순간은 한 번뿐이므로, 이 과정 전체는 전체 칸 수에 비례하는 비용으로 처리할 수 있다.

아이디어 3: 세그먼트 트리

- 쿼리는 항상 “현재까지 구간 $[A_i, B_i]$에서 치운 눈의 합”을 묻는다.
- 한 라운드 동안, 각 작업이 진행되면서 위치별로 눈을 얼만큼 치웠는지만 누적해서 기록하면 된다.
- 이를 위해, 한 지점에서 치운 양을 더해 두고 임의의 구간에 대한 합을 빠르게 구할 수 있는 자료구조인 세그먼트 트리를 사용한다.

세그먼트 트리로 합 구하기



- 작업 시뮬레이션 중 어떤 위치에서 눈을 일정 양만큼 치웠다면, 그 위치에 그만큼의 값을 누적해서 더해 둔다.
- 세그먼트 트리는 이렇게 쌓인 값들에 대해, 임의의 구간에 대한 합을 $\mathcal{O}(\log N)$ 시간에 계산해 줄 수 있다.
- 따라서 시점 t에서 쿼리 (A_i, B_i, T_i)를 판정할 때, "구간 $[A_i, B_i]$에서 치운 눈의 총합"을 빠르게 구해 사용할 수 있다.

전체 알고리즘 요약 (1/2)



- 입력: 눈 높이 S_i, 작업 (L_t, R_t, C_t), 쿼리 (A_i, B_i, T_i).
- 각 쿼리 i에 대해

$$lo_i = 1, \quad hi_i = M, \quad ans_i = -1$$

로 초기화한다.

- 이후 병렬 이분 탐색 라운드를 반복하면서, 모든 쿼리의 정답을 좁혀 나간다.

전체 알고리즘 요약 (2/2)



- ✓ 한 라운드에서는 다음을 수행한다.
  - 아직 $lo_i \le hi_i$ 인 쿼리들에 대해 중간 값 mid_i 를 정하고, 같은 값을 갖는 쿼리끼리 작업 번호별로 묶어 둔다.
  - 남은 눈 배열, DSU, 세그먼트 트리를 초기화한다.
  - 작업을 1번부터 순서대로 시뮬레이션하면서, 각 시점에서 해당 작업 번호를 mid로 사용하는 쿼리들을 판정한다.
- ✓ 더 이상 $lo_i \le hi_i$ 인 쿼리가 없으면 라운드를 종료하고, 각 쿼리의 ans_i 를 출력한다.

시간 복잡도 분석 (1/2)



- ✓ 병렬 이분 탐색에서 각 쿼리의 정답 구간 $[1, M]$을 반씩 줄이므로, 라운드 수는 $\mathcal{O}(\log M)$번이다.
- ✓ 한 라운드에서의 초기화 비용:
  - DSU 구조를 다시 세팅하는 데 $\mathcal{O}(N)$.
  - 세그먼트 트리의 초기화에 $\mathcal{O}(N)$.

시간 복잡도 분석 (2/2)



- ✓ 한 라운드에서 작업들을 처리하는 비용:
  - DSU를 이용해 이미 치운 구간을 건너뛰며 눈을 치운다.
  - 각 칸은 한 번만 완전히 치워지므로, 이 부분의 총 비용은 $\mathcal{O}\left((N+M)\alpha(N)\right)$ 이다.
  - 칸이 치워질 때마다 세그먼트 트리에 값을 더하는 데 전체 $\mathcal{O}\left(N \log N\right)$ 정도가 든다.
- ✓ 쿼리를 판정하는 비용:
  - 각 라운드에서 모든 쿼리에 대해 세그먼트 트리의 구간 합을 한 번씩 계산하므로 $\mathcal{O}\left(Q \log N\right)$.
- ✓ 따라서 전체 시간 복잡도는 $\mathcal{O}\left(\left((N+M)\alpha(N) + (N+Q)\log N\right)\log M\right)$