UCPC 2023 본선 해설

Official Solutions

전국 대학생 프로그래밍 대회 동아리 연합

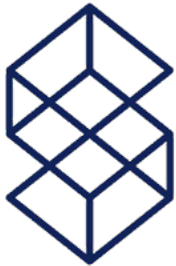
2023년 7월 22일

UCPC 2023

운영진 / 출제진 / 검수진

- 이종서 leejseo
- 김도훈 99asdfg
- 김태훈 amel
- 최은규 asdarwin03
- 박신욱 bnb2011
- 곽우석 bubbler
- 김도훈 dohoon
- 양성준 egod1537
- 이상헌 evenharder
- 배근우 functionx
- 유상혁 golazcc83
- 이혜아 ho94949
- 임병찬 hyperbolic
- 최재민 jh05013
- 구재원 jjaewon9
- 권혁준 juny2
- 이경찬 kclee2172
- 이성호 puppy
- 한동규 queued_q
- 박상훈 qwerasdfzxcl
- 장우성 saywoo
- 신용명 tlsdydaud1
- 박수찬 tncks0121
- 이한길 wapas
- 정명준 wider93
- 정우경 man_of_learning
- 채이환 benedict0724
- 이지후 silverwolf
- 윤창기 TAMREF
- 시제연 tlwpdus
- 이동훈 aru0504

Sponsors



SAMSUNG
SOFTWARE
MEMBERSHIP

MOLOCO

Sponsors

Sponsors

Sponsors

HYUNDAI AutoEver

Jane Street®

문제 목록

문제		의도한 난이도	출제자
A	계통수 추론	Challenging	evenharder
B	I forgor 💀	Medium	jh05013
C	황혼	Medium	qwerasdfzxcl
D	거대 로봇 전투	Challenging	functionx
E	격자 속의 직선 경로	Hard	bubbler
F	두 트리	Hard	puppy
G	K번째 스페이드 찾기	Medium	queued_q
H	피보나치 반반수열	Medium	bubbler
I	안테나 설치	Hard	cdjs1432
J	회전초밥	Hard	hyperbolic
K	그건 가지가 아니라 대파예요	Challenging	functionx
L	나무늘보	Easy	egod1537
M	산 색칠	Medium	amel

A. 계통수 추론

`ad_hoc`, `constructive`, `graph_theory`

출제진 의도 – **Challenging**

- 제출 4번, 정답 1팀 (정답률 25.000%)
- 처음 푼 팀: **멕시코시티노점상에서타코사먹는윤창기** (주머니에서폰훔쳐가기, 뒷통수치고튀기, 타코뺏어먹기), 111분
- 출제자: `evenharder`

A. 계통수 추론

- 모든 가설을 충족하는 계통수 T가 존재한다고 합시다.
- 존재한다면 T의 모든 정점 x에 대해 x는 리프 정점이거나 자식 정점이 2개 이상 있다고 가정해도 됩니다.
  - 자식 정점이 1개밖에 없다면 x와 x의 자식 정점을 병합해도 여전히 모든 가설을 충족함을 보일 수 있습니다.
- 이 성질을 이용하면 계통수가 존재한다면 정점이 최대 $2N-1$개인 계통수가 존재한다는 점도 밝힐 수 있습니다.

편의상 두 정점 a와 b에 대해 (a, b)는 a와 b의 최소공통조상으로, $a \neq b$일 때 $a < b$는 a가 b의 후손, b가 a의 조상이라는 뜻으로 표기하겠습니다.

T를 재귀적으로, 정점 1부터 정점 N까지 리프 정점으로 해서 만들어봅시다.

- T의 루트 정점의 자식 서브트리 $T_1, T_2, \cdots, T_r$를 생각해봅시다.
- 이 때 각 가설 $c_i = (a, b) < (c, d)$에 대해 다음을 충족해야 합니다.
  - a와 b는 같은 서브트리에 속해야 한다. (그렇지 않으면 (a, b)가 루트가 되기 때문)
  - c와 d가 같은 서브트리에 속하면 a, b, c, d 모두 같은 서브트리에 속해야 한다.

이 성질을 이용해서, 다음과 같이 정점의 집합 S을 $S_1, S_2, \cdots, S_r$로 분할해볼 수 있습니다.

1. $(a, b) < (c, d)$이면 a와 b는 같은 집합에 속한다.
2. $(a, b) < (c, d)$이고 c와 d가 같은 집합에 속하면, a, b, c, d 모두 같은 집합에 속한다.
3. 위의 두 규칙을 제외하고는 서로 다른 리프 정점은 집합에 속한다.

   같은 집합에 속한다를 같은 서브트리에 속한다에 대응시키면 됩니다.

S를 $S_1, S_2, \cdots, S_r$로 분할했습니다.

- 이 때 $r = 1$이면 리프 정점을 더 이상 분리할 수 없게 되어 계통수는 존재하지 않습니다.
- $r > 1$이면, $C_i = \{(a, b) < (c, d) \mid a, b, c, d \in S_i\}$로 정의되는 $C_1, C_2, \cdots, C_r$을 만듭니다. S_i에 대응되는 가설 집합이라고 볼 수 있습니다.

  이제 S_i, C_i는 각 서브트리에 대응이 되므로, 재귀적으로 계통수를 만들어 볼 수 있겠습니다.

요약하면 계통수를 만드는 알고리즘은 다음과 같습니다.

1. S에 원소가 1개 있으면 해당 원소를 반환합니다.
2. 현재 정점과 가설의 집합을 이용해 정점을 $S_1, S_2, \cdots, S_r$로 분할합니다.
   - $r = 1$이면 불가능하므로 -1을 반환합니다.
3. 루트 정점의 번호 x를 설정합니다.
4. $C_i = \{(a, b) < (c, d) \mid a, b, c, d \in S_i\}$로 정의되는 $C_1, C_2, \cdots, C_r$을 만듭니다.
5. 재귀적으로 $build(S_i, C_i)$를 수행하고, 반환되는 각 정점을 x의 자식으로 설정합니다.
   - 이 때 재귀 호출 중 -1이 하나라도 있으면 불가능하므로 -1을 반환합니다.
6. S와 C를 가지고 만든 계통수의 루트 정점인 x를 반환합니다.

$build(S, C)$는 다음 성질을 충족합니다.

1. $build(S, C)$가 -1을 반환하지 않으면, 만들어진 계통수는 C를 충족합니다.
2. C를 충족하는 계통수가 존재하면 $build(S, C)$는 -1을 반환하지 않습니다.

즉, $build(S, C)$가 -1을 반환하면 불가능하며, 아니면 만들어진 계통수를 출력하면 됩니다.

첫 번째 성질은 정점의 집합 S의 크기에 따른 수학적 귀납법으로 보일 수 있습니다.
S의 크기가 1일 때는 자명합니다. 위 성질이 S보다 크기가 작은 모든 정점에 대해 성립하며, 현재 만들어진 계통수를 T라고 합시다.

- 재귀 호출시 정점의 집합은 그 크기가 S보다 작으므로 위 성질이 성립합니다.
- 임의의 가설 $(a, b) < (c, d)$에 대해,
  - c랑 d가 같은 집합에 속하면 가설의 정점이 분할 시 같은 집합 S_i에 속합니다. 또한 해당 가설도 C_i에 속하므로 가정에 의해 T도 현재 가설을 충족합니다.
  - c랑 d가 다른 집합에 속하면 (c, d)가 T의 루트 정점입니다. 반면 a와 b는 같은 집합에 속하기 때문에 T도 현재 가설을 충족합니다.

그러므로 수학적 귀납법에 의해 성립함을 알 수 있습니다.

두 번째 성질은 가설 집합 C를 충족하는 계통수 T가 제일 정점의 개수가 적은 반례라고 설정하고 $build(S, C)$가 -1을 반화할 때 모순을 이끌어내는 귀류법으로 증명할 수 있습니다.
C가 공집합일 때 정점 1개짜리 트리가 반례가 되지 않음은 명백하므로 C가 공집합이 아니라고 가정해도 됩니다. 이 때,

- S를 분할한 결과가 S이거나
- 재귀 호출 $build(S_i, C_i)$에서 -1이 반환되어야 합니다.

두 번째 경우는 $build(S_i, C_i)$로 만들어지는 계통수가 T보다 작으면서 반례가 되므로 모순입니다.

첫 번째 경우에서는 다음 보조정리를 사용해야 합니다. 이는 분할 과정에서 집합을 병합하는 상황을 수학적 귀납법으로 따져가며 증명할 수 있습니다.

보조정리

가설 집합 C를 충족하는 임의의 계통수 T에 대해, T의 자식 서브트리를 $T_1, T_2, \cdots, T_r$ 이라고 하자. 정점 집합 S의 분할로 생성되는 임의의 집합 S_i에 대해, S_i를 모두 포함하는 T_j가 존재한다.

이를 이용하면 T는 하나의 자식만을 가짐을 알 수 있고, T의 루트 정점을 제거한 계통수는 C를 충족하면서 더 크기가 작으므로 모순입니다.

증명이 끝났으니 구현을 살펴보아야 합니다.

- 재귀적으로 정점의 집합과 조건의 집합을 기반으로, disjoint set과 queue 등을 이용해 $\mathcal{O}\left(M^2\right)$ 분할을 구현할 수 있고, $build$의 총 시간복잡도는 $\mathcal{O}\left(NM^2\right)$가 됩니다.
  - naive 알고리즘의 수행 시간은 재귀 깊이나 병합 횟수의 영향을 받기 때문에 시간복잡도에 비해 짧을 수 있으나, worst case에 해당하는 가설 집합을 만들 수 있습니다. (어떤 식으로 구성되어 있을까요?)
- small-to-large을 이용하면 분할의 복잡도가 $\mathcal{O}\left(M\lg M\right)$이 되어, 총 시간복잡도 $\mathcal{O}\left(NM\lg M\right)$ 정도에 계산할 수 있습니다.

보다 상세한 설명은 다음 논문을 참고하시면 됩니다.

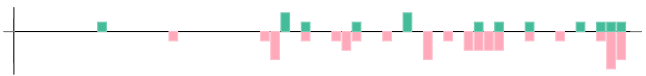
- Aho, A. V., Sagiv, Y., Szymanski, T. G., & Ullman, J. D. (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. **SIAM Journal on Computing**, **10**(3), 405-421.

B. I forgor 💀

`tree_set`

출제진 의도 – **Medium**

- 제출 47번, 정답 14팀 (정답률 29.787%)
- 처음 푼 팀: **경기과학고등학교36기화이팅** (18060, 18080, 18089), 43분
- 출제자: `jh05013`



B. I forgor 💀

- 뒤집은 두 카드에 같은 수가 적혀 있으면 "성공", 아니면 "실패"라고 합시다.
- 성공은 항상 정확히 N번 합니다.
- 실패의 개수만 세 주고 N을 더하면 됩니다.

B. I forgor 💀

- 각 수에 대해, 그 수가 적힌 카드 중 왼쪽에 있는 것을 A, 오른쪽에 있는 것을 B라고 합시다.
  - 예제의 처음 상태는 1A 2A 3A 1B 4A 4B 2B 3B입니다.
- 안 뒤집어 본 맨 왼쪽 카드의 위치를 "현재 위치"라고 합시다. 이는 1에서 시작해서 점차 증가하고 $2N$에서 끝납니다.
- 규칙:
  - 현재 위치의 카드가 B이면, **성공**하고 현재 위치가 1 증가합니다.
  - 현재 위치의 카드가 A이면서, 바로 다음 카드가 같은 수의 B 카드이면, **성공**하고 현재 위치가 2 증가합니다.
  - 아니라면, **실패**하고 현재 위치가 2 증가합니다.

B. I forgor 💀

- 사실 두 번째 규칙에서 현재 위치가 2가 아니라 1 증가해도 됩니다.
  - 바로 다음 카드가 B라서 게임 결과는 "성공 횟수"를 제외하고 바뀌지 않습니다.
  - 성공 횟수는 어차피 안 셀 거기 때문에 바뀌어도 됩니다.
- 규칙:
  - 현재 위치의 카드가 B이면, **성공**하고 현재 위치가 1 증가합니다.
  - 현재 위치의 카드가 A이면서, 바로 다음 카드가 같은 수의 B 카드이면, **성공**하고 현재 위치가 1 증가합니다.
  - 아니라면, **실패**하고 현재 위치가 2 증가합니다.

B. I forgor 💀

- 이제 어떤 카드가 B 카드이거나, A 카드이면서 바로 다음 카드가 같은 수의 B 카드이면, 그 카드를 "성공 카드"라고 합시다.
- 그렇지 않은 카드를 "실패 카드"라고 합시다.
  - 예제의 처음 상태 (1A 2A 3A 1B 4A 4B 2B 3B)는 [실-실-실-성-성-성-성-성]입니다.
- 규칙:
  - 현재 위치의 카드가 성공 카드이면, **성공**하고 현재 위치가 1 증가합니다.
  - 현재 위치의 카드가 실패 카드이면, **실패**하고 현재 위치가 2 증가합니다.

B. I forgor 💀

- 카드 쌍을 서로 바꿀 때마다 모든 카드의 종류(성공/실패)를 $\mathcal{O}(1)$의 시간 복잡도로 갱신할 수 있습니다.
  - 종류가 바뀌는 카드는 $\mathcal{O}(1)$개입니다.
  - 각 카드의 위치를 관리하면 각 카드의 종류를 $\mathcal{O}(1)$의 시간 복잡도로 알아낼 수 있습니다.
- 따라서 카드 쌍을 서로 바꾸는 쿼리는 카드 하나의 종류를 바꾸는 쿼리 여러 개로 대체할 수 있습니다.
- 이제 다음과 같은 쿼리를 처리하는 문제가 되었습니다.
  - 어떤 카드를 성공 카드로 바꾼다.
  - 어떤 카드를 실패 카드로 바꾼다.
  - 이 상태로 게임을 했을 때 실패 횟수를 계산한다.

B. I forgor 💀

- 연속된 실패 카드를 하나의 그룹으로 묶읍시다.
- 그룹의 크기를 s라고 하면, 그 그룹 안에서는 정확히 $\lfloor \frac{s+1}{2} \rfloor$번 실패합니다.
  - 성공할 때는 위치가 1씩 증가하기 때문에, 반드시 각 그룹의 맨 첫 카드에서 실패합니다.
  - 그 카드에서부터 $\lfloor \frac{s+1}{2} \rfloor$번 실패하고 나서 그룹을 빠져나옵니다.
- 따라서 모든 그룹에 대해 위 값의 합을 구하면 됩니다.
- 성공 카드의 위치를 순서대로 늘어놓으면 둘 사이 공간 하나하나가 그룹이라는 점을 사용하면, C++ std::set 등을 통해 그룹을 효율적으로 관리할 수 있습니다.
- 시간 복잡도는 $\mathcal{O}\left((N+Q)\log N\right)$입니다.

C. 황혼

`graph`, `dijkstra`

출제진 의도 – **Medium**

- 제출 87번, 정답 41팀 (정답률 45.055%)
- 처음 푼 팀: **Se Solve** (Do Solve, Hana Solve, Se Solve), 19분
- 출제자: `qwerasdfzxcl`

- $2N$개의 정점과 $4M$개의 간선으로 구성된 그래프 G를 만든 후, 다익스트라 알고리즘으로 답을 구할 수 있습니다.
- 각 도시 v에 대해 정점 v_0와 v_1을 G에 추가합니다.
- v_1은 도시 v에 있고, 어느 시점부터 v가 속한 단순 경로를 따라 이동해온 상태를 뜻합니다.
- v_0는 도시 v에 있으면서 v_1이 아닌 상태를 뜻합니다.
- 단순 경로의 첫 번째 도시 s와 마지막 도시 e에 대해 s_0와 e_1은 방문할 수 없는 정점임에 유의합시다.
- 입력으로 들어온 간선을 정점의 정의에 맞게 G에 추가해주면 G를 완성할 수 있습니다.
- 시간복잡도는 $\mathcal{O}\left(N + M\log M\right)$ 입니다.

D. 거대 로봇 전투

`dp`, `segtree`, `hld(sparse_table)`

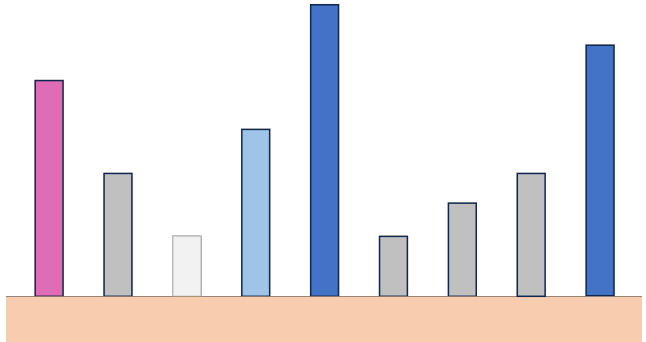
출제진 의도 – **Challenging**

- 제출 5번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: –
- 출제자: `functionx`

- 우선 쿼리가 하나일 때를 처리해봅시다.
- 거대 로봇보다 키가 크거나 같은 미니언 X_0 이 있다면,
  - X_0 뒤에 있는 모든 미니언은 공격 불가합니다.
  - X_0 및 그 앞에 있는 모든 미니언은 공격 가능합니다.
- 거대 로봇을 볼 수 없는 미니언을 일단 모두 없앱니다.

- 앞쪽 미니언 중 일부를 남겨놓고 X_0 을 없앤다면 앞쪽 미니언이 뒤쪽 미니언을 가릴 수도 있습니다.
- 따라서 다음 두 선택지를 고려할 수 있습니다.
  1. X_0 및 그 앞에 있는 모든 미니언을 다 없애기
  2. X_0 앞에 있는 가장 키가 큰 미니언을 제외하고 다 없애기
- 이걸 이용해서 DP를 설계할 수 있습니다.

D. 거대 로봇 전투



– 그림과 같이 미니언 키의 수열을 키가 B 보다 큰 미니언으로 끝나는 여러 증가수열로 쪼갭시다.

- $DP[i]$를 다음과 같이 정의합니다.
  - $DP[i]$: i번째 증가수열까지 성공적으로 공격한 다음 남길 수 있는 미니언 키의 최댓값
  - 만약 공격에 성공할 수 없다면 $DP[i] = -1$이고, 미니언을 남길 수 없다면 $DP[i] = 0$입니다.
- $DP[i-1]$에서 남긴 미니언을 그대로 쓸지 지울지 결정해서 $DP[i]$를 계산하면 됩니다.
- 로봇의 내구도 K를 `parametric_search`로 조절하여 DP를 계산하면 쿼리 당 $\mathcal{O}(N \log N)$ 시간에 답을 구할 수 있습니다.

- 이제 쿼리가 여러 개인 경우를 처리합니다.
  - 거대 로봇의 키가 작을 때 사용하는 전략은 키가 커지더라도 그대로 사용할 수 있으므로, 키가 작을 때 필요한 내구도 $K(B_j)$가 더 크거나 같습니다.
  - 따라서 쿼리를 키 내림차순으로 정렬한 다음 쿼리를 처리합니다.
- 맨 오른쪽에 키가 ∞인 가상 미니언을 세운 다음, 각 미니언과 자신의 오른쪽에 있는 키가 더 큰 첫 번째 미니언을 간선으로 이어줍니다.
- 그러면 미니언끼리 연결은 가상 미니언이 루트 노드인 트리가 됩니다.

- 앞서 언급했던 증가수열을 관리합니다.
  - 증가수열은 노드 S_0 부터 루트 노드까지 가는 단순 경로로 볼 수 있습니다.
  - 루트와 거리가 K 인 노드를 S_{mi}, 키가 B 보다 작은 루트와 가장 가까운 노드를 S_{mx} 라고 합시다.
  - S_{mi}, S_{mx} 는 `hld(sparse_table)` 로 $\mathcal{O}(\log N)$ 시간에 구할 수 있습니다.
- 앞쪽 미니언을 모두 제거하고 이 증가수열을 공격하려면 경로에 포함하는 노드가 $K+1$ 개 이하여야 합니다.
- 앞쪽 미니언 하나만 남기고 증가수열을 공격하려면 미니언의 키가 $H[S_{mi}]$ 이상이어야 합니다.

- 이 증가수열을 통해 도출되는 값 H_{mi}, H_{mx}, $force$를 segtree로 관리합니다.
- $force = 0$이면,
  - $H_{mi} \le H_{mx}$를 만족해야 합니다.
  - 증가수열을 공격하기 위해 필요한 미니언의 키 B가 H_{mi} 이상이어야 하며, 키가 $\max(B, H_{mx})$인 미니언을 남길 수 있습니다.
- $force = 1$이면,
  - 증가수열을 공격하기 위해 필요한 미니언의 키 B가 H_{mi} 이상이어야 하며, 키가 H_{mx}인 미니언을 남길 수 있습니다.

- 증가수열을 segtree 노드 하나로 만들 때 다양한 케이스가 존재합니다.
- 증가수열에서 $S_{mi} < S_{mx}$ 를 만족시키는 경우,
  - H_{mi} 는 S_{mi} 의 키와 동일합니다.
  - H_{mx} 는 S_{mx} 의 키와 동일합니다.
  - $force = 0$ 입니다.

- 증가수열에서 $S_{mi} = S_{mx}$ 를 만족시키는 경우,
- 갖고 온 미니언의 키 S_0 이 미니언 S_{mi} 보다 작으면 미니언을 버린 다음 증가수열을 공격할 수 있습니다.
- 결론적으로 남길 수 있는 미니언의 최대 키는 $\max(S_0, S_{mx})$ 입니다.
  - $H_{mi} = 0$ 입니다.
  - H_{mx} 는 S_{mx} 의 키와 동일합니다.
  - $force = 0$ 입니다.

- 증가수열에서 $S_{mi} > S_{mx}$ 인 경우는 두 가지 있습니다.
- 증가수열에 키가 B 보다 작은 미니언이 있거나 B 보다 큰 미니언이 K 개보다 많다면, 어떤 경우에도 동시에 K 개보다 많은 미니언이 거대 로봇을 공격합니다.
  - $H_{mi} = \infty$ 입니다.
  - H_{mx} 는 아무 값이나 넣어도 무방합니다.
  - $force = 1$ 입니다.

- 증가수열에 키가 B 보다 작은 미니언이 없다면 무조건 미니언 없이 증가수열을 공격해야 합니다.
  - $H_{mi} = 0$ 입니다.
  - $H_{mx} = 0$ 입니다.
  - $force = 1$ 입니다.

- `segtree`의 개별 노드가 바뀌는 경우는 두 가지가 있습니다.
- 거대 로봇의 키 B가 줄어들 때 키가 B 이상인 로봇 x가 새로 생기는데,
  - 기존 x를 포함하는 증가수열에서 H_{mx}가 수정됩니다.
  - $x+1$부터 시작하는 새로운 증가수열이 만들어집니다.
- 내구도가 K일 때 불가능하다고 판단되면 K가 증가하는데,
  - 모든 증가수열에 대해서 H_{mi}가 수정됩니다.
  - 루트와 거리가 K인 노드 S_{mi}는 중복이 발생하지 않기 때문에 총 수정 횟수는 amortized $\mathcal{O}(N)$입니다.

- 내구도가 K일 때 전략이 있는지 계산하려면,
  1. segtree에 있는 모든 노드를 합쳐서 새로운 노드를 만듭니다.
  2. 이 노드의 H_{mi} 값이 0이면 H_{mx}, $force$의 값에 상관없이 전략이 존재합니다.
- 총 시간복잡도는 쿼리 당 $\mathcal{O}(\log N)$ 입니다.

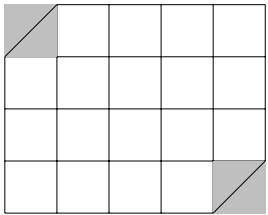
E. 격자 속의 직선 경로

`geometry`, `convex_hull`
출제진 의도 – **Hard**

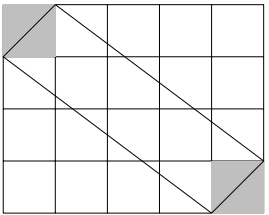
- 제출 34번, 정답 6팀 (정답률 17.647%)
- 처음 푼 팀: **멕시코시티노점상에서타코사먹는윤창기** (주머니에서폰훔쳐가기, 뒷통수치고튀기, 타코뺏어먹기), 185분
- 출제자: `bubbler`

E. 격자 속의 직선 경로

- $(1, 1)$과 (R, C)에 아래 그림 1처럼 사선을 그리면 "두 칸의 내부를 잇는 선분이 존재한다"와 "두 사선을 잇는 선분이 존재한다"는 동치입니다.
- 그러한 선분이 존재한다면, 그 선분은 두 사선을 이어서 만들어지는 평행사변형을 벗어날 수 없습니다.

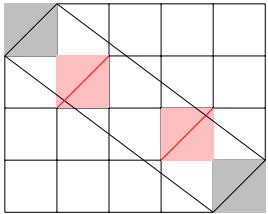


(a) 그림 1

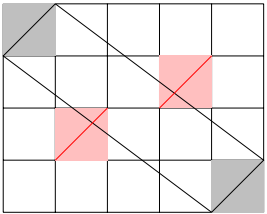


(b) 그림 2

– 또한, 장애물이 되는 검은 칸도 하나의 사선으로 치환할 수 있습니다. 이 사선은 앞의 평행사변형의 위쪽 변이나 아래쪽 변 중 하나와 교차하거나, 둘 다에 닿을 수 있습니다. 둘 다에 닿은 사선이 하나라도 있는 경우 답은 0입니다.



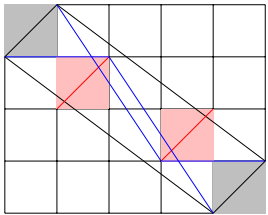
(a) 예제 1에 해당하는 그림



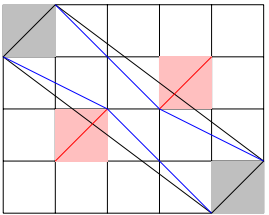
(b) 예제 2에 해당하는 그림

E. 격자 속의 직선 경로

- 두 사선을 잇는 선분을 그릴 때, 평행사변형의 위쪽 변과 교차한 사선들을 피하려면 위쪽 변의 양 끝점과 사선들의 아래쪽 점의 lower convex hull을 피해야 합니다.
- 마찬가지로, 아래쪽 변과 교차한 사선들을 피하려면 아래쪽 변의 양 끝점과 사선들의 위쪽 점의 upper convex hull을 피해야 합니다.



(a) 예제 1에 해당하는 그림



(b) 예제 2에 해당하는 그림

- 두 convex hull이 겹치거나 접하면 답은 0이고, 아니라면 답은 1입니다.
- 이는 x좌표에 대한 sweeping 등의 방법으로 구할 수 있습니다.
- 두 convex hull이 겹치거나 접하지 않는다면, 다음과 같은 방법으로 선분을 그을 수 있습니다.
  - 둘 사이의 간격이 가장 좁은 부분을 찾습니다. 그 부분이 한쪽 convex hull의 한 변과 반대쪽 convex hull의 꼭지점 사이라고 할 때, 그 위치의 한 점 X에서 시작해서 가까운 변과 평행하게 직선을 긋습니다.
  - 양쪽 convex hull은 정의에 의해 볼록하므로 X에서 멀어질수록 그린 직선과 양쪽 convex hull 사이의 거리는 단조증가하고, 따라서 이 직선은 convex hull과 교차하지 않고 평행사변형의 양쪽 끝 변에 닿게 됩니다.

F. 두트리

`euler_tour_technique`, `segment_tree`, `divide_and_conquer`, `sorting`

출제진 의도 – **Hard**

- 제출 72번, 정답 20팀 (정답률 27.778%)
- 처음 푼 팀: **멕시코시티노점상에서타코사먹는윤창기** (주머니에서폰훔쳐가기, 뒷통수치고튀기, 타코뺏어먹기), 21분
- 출제자: puppy

- T_1에서 dfs를 하여 리턴되는 정점부터 정렬합니다.
- T_1에서 u가 v의 서브트리 안에 있으면, u는 v보다 앞쪽에 정렬됩니다.

F. 두 트리

- 각 트리에서 euler tour을 진행하고, in과 out을 각각 $i.s1$, $i.e1$, $i.s2$, $i.e2$라고 합시다.
- 각 정점 p에 대해 $p.s1 \le i.s1 \le p.e1$, $p.s2 \le i.s2 \le p.e2$를 만족하는 i 중 $a[i]$의 최댓값을 구하면 됩니다.
- 정렬되었다는 사실을 이용하면, 위 조건에서 $p.s1 \le i.s1 \le p.e1$이 p보다 i가 앞에 있고, $p.s1 \le i.s1$인 것으로 환원됩니다.
- 이를 이용하여 분할 정복을 합니다.

- i가 p보다 앞에 있다는 조건을 무시한다면, $p.s1 \le i.s1$이고 $p.s2 \le i.s2 \le p.e2$인 정점 중 $a[i]$의 최댓값은 세그먼트 트리와 스위핑으로 $\mathcal{O}\left(N \log N\right)$에 계산할 수 있습니다.
- dnc(l, r)을 정렬 순서상 l번째부터 r번째까지의 답을 계산하는 함수로 정의하면 분할 정복을 이용해서 답을 계산할 수 있습니다.
- 시간복잡도는 $\mathcal{O}\left(N \log^2 N\right)$입니다.

G. K번째 스페이드 찾기

greedy

출제진 의도 – **Medium**

- 제출 74번, 정답 43팀 (정답률 58.108%)
- 처음 푼 팀: **세계 최고의 프로그래밍 언어** (킹이썬, 똥이썬, 엄준식), 38분
- 출제자: queued_q

- 유진이가 i번째 위치에서 "스탑!" 을 외쳤을 때, 현재까지 $S[i]$장의 스페이드를 내려놓았다고 합시다.
- 이제 남은 카드 중에서 $K - S[i]$번째 스페이드를 찾아야 합니다.
- "스탑!" 을 외칠 때마다 남은 카드 중에서 몇 번째 스페이드를 찾아야 하는지가 달라집니다.

전략

남은 카드 중에서 $x = K - S[i]$ 번째 스페이드를 찾아야 하는 상황을 가정합시다.

- 유진이는 딜러가 x 장의 카드를 추가로 내려놓은 뒤 "스탑!" 을 외쳐야 합니다.
  - 더 늦게 "스탑!" 을 외치면 x 번째 스페이드의 위치를 정확히 알지 못할 수도 있습니다.
  - 한편, 남은 카드 수가 적을수록 이득이므로 "스탑!" 을 최대한 늦게 외쳐야 합니다.
- 이때 딜러가 추가로 내려놓은 스페이드가 정확히 x 장이라면, 마지막으로 내려놓은 카드가 유진이가 찾는 카드입니다.

- 따라서 유진이가 마지막으로 "스탑!" 을 외친 위치를 i 라고 했을 때,
- i 를 $i + (K - S[i])$ 로 업데이트하고 "스탑!" 을 외치는 과정을 반복하다가
- $S[i] = K$ 가 되는 순간 종료하면 됩니다.
- 모든 K 에 대해 이를 직접 시뮬레이션하면 답을 구할 수 있습니다.

시간복잡도

- 1 부터 N 까지의 모든 K 에 대해,
- 최악의 경우 i 가 1 씩 증가하여 N 번의 "스탑!" 을 외쳐야 하므로,
- 총 $\mathcal{O}\left(N^2\right)$ 의 시간이 걸립니다.

하지만...

사실 앞선 풀이는 $\mathcal{O}\left(N \log N\right)$ 시간에 동작합니다!

- 극단적인 상황, 즉 모든 카드가 하트인 경우를 생각해 봅시다.
- 유진이는 매 K 장의 카드마다 "스탑!" 을 외치므로, 약 N/K 번 "스탑!" 을 외칩니다.
- 조화수열의 합을 이용하면 다음과 같이 전체 시간복잡도를 계산할 수 있습니다.

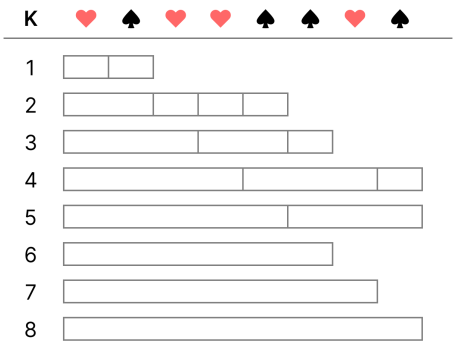
$$\mathcal{O}\left(\frac{N}{1}+\frac{N}{2}+\cdots+\frac{N}{N}\right)=\mathcal{O}\left(N \log N\right)$$

G. K번째 스페이드 찾기

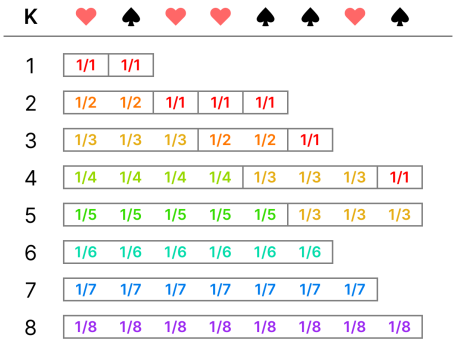
- 스페이드 카드가 많아질수록 카드를 찾는 평균 속도가 빨라질 것이라고 예상할 수 있습니다.
- 하지만 스페이드 카드가 추가된다고 해서, 각각의 K에 대해 "스탑!" 을 외치는 횟수가 줄어드는 것은 아닙니다.
- 믿음을 가지고 제출하면 해결할 수 있지만, 어떻게 증명할까요?

시간복잡도 증명

- 모든 K에 대해 "스탑!" 사이 구간들을 그려 봅시다. 이러한 구간들을 "점프" 라고 부릅시다.
- 시간복잡도는 점프의 개수에 비례합니다.

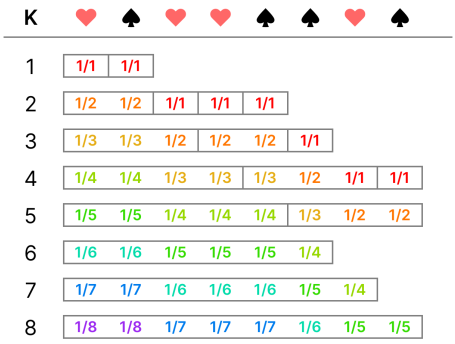


- 점프의 개수를 세는 대신, 길이 L의 점프를 단위 구간들로 쪼개서 각각에 $1/L$의 가중치를 부여합시다.
- 이때 가중치의 합을 P라고 합시다. P는 점프의 개수와 같습니다.



G. K번째 스페이드 찾기

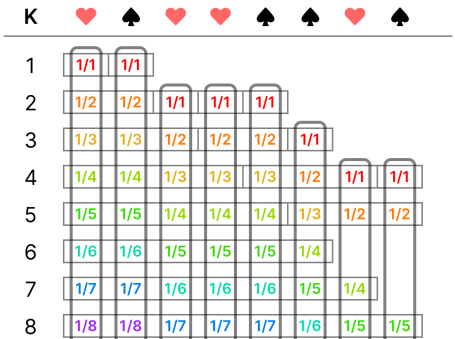
- 이번에는 각 K에 대한 i번째 단위 구간에 $1/(K - S[i-1])$의 가중치를 매깁시다.
- 이때 가중치의 합을 Q라고 합시다.



- i번째 단위 구간에 부여된 가중치를 모든 K에 대해서 더하면 다음과 같습니다.

$$1+\frac{1}{2}+\cdots+\frac{1}{N-S[i-1]}=\mathcal{O}\left(\log N\right)$$

- 따라서 $Q=\mathcal{O}\left(N\log N\right)$입니다.



UCPC 2023

- 한편, 각 점프에 속하는 단위 구간마다 $1/L$ 이상의 가중치가 부여됩니다.
- 그러므로 $P \le Q = \mathcal{O}\left(N \log N\right)$ 입니다. 따라서 전체 시간복잡도는 $\mathcal{O}\left(N \log N\right)$ 입니다.

H. 피보나치 반반수열

`math`, `ad_hoc`

출제진 의도 – **Medium**

- 제출 76번, 정답 32팀 (정답률 41.558%)
- 처음 푼 팀: **DDT** (두, 둥, 탁), 48분
- 출제자: `bubbler`

- a_n의 정의에 의해, $a_{f_n} = a_{a_{a_n}} = f_{a_n}$ 입니다. 따라서, 피보나치 수가 아닌 어떤 수 n에 대해서 a_n의 값이 정해지면 $a_{f_n}, a_{f_{f_n}}, \cdots$ 가 모두 정해지게 됩니다.
- 이를 모두 종합하면, 피보나치 수가 아닌 4 이상의 **모든** 수 n에 대해 a_n이 정해지면 피보나치 수를 포함한 모든 $n \geq 4$에 대한 a_n이 정해집니다.
- 3 이하의 경우 $a_1 = 1$, $a_2 = 2$, $a_3 = 3$임이 자명합니다.
- 앞으로 "피보나치 수가 아닌 수"를 nonfib이라고 부르겠습니다.

- 두 개의 nonfib n, m에 대해 $a_n = m$이면, 다음의 값들이 연쇄적으로 정해집니다.

$$a_n = m$$
$$a_m = a_{a_n} = f_n$$
$$a_{f_n} = a_{a_m} = f_m$$
$$a_{f_m} = a_{a_{f_n}} = f_{f_n}$$
$$\dots$$

- 이를 n과 m을 짝짓는다고 하겠습니다.

- "사전 순으로 최소"라는 조건에 의해, nonfib n에 대한 a_n은 n이 증가하는 순서대로 정해야 합니다.
- 먼저 a_4의 가능한 값을 찾아봅시다.
- 4 이하일 수 없음은 쉽게 알 수 있습니다.
- $a_4 = 5$이면 $a_{a_4} = f_4 = 5$여야 하는데, 그러면 $a_5 = 5$, $a_{a_5} \neq f_5 = 8$이므로 모순입니다.
- $a_4 = 6$이면 모순이 발생하지 않습니다. 4와 6이 짝지어져 $a_6 = 5$가 같이 정해집니다.
- a_7과 이후의 nonfib번째 항에 대해서도 두 개씩 짝지어 주면 모든 항을 모순 없이 정할 수 있으므로 $a_4 = 6$입니다.

- $n \geq 7$에 대해서 같은 방법이 최적임을 다음과 같이 보일 수 있습니다.
- 아직 a_n의 값이 정해지지 않은 가장 작은 nonfib 인덱스 n을 생각합니다.
- $a_{a_n} = f_n$여야 하는데, n번째 항 이전에는 $a_m = f_n$인 m이 없으므로 $a_n < n$일 수 없습니다. $a_n = n$일 수 없음은 자명하므로 $a_n > n$입니다.
- 만약 $n+1$이 nonfib이라면 n을 $n+1$과 짝지을 수 있습니다. 아니라면, $n+1 = f_m$을 만족하는 nonfib인 $m < n$이 존재하는데, $a_{a_m} = f_m$이므로 $a_{a_n} = a_{n+1} = a_{f_m} = f_{a_m} \neq f_n$ ($a_m = n$이면 가정에 모순)이 되어 $a_n \neq n+1$입니다. $n+1$이 피보나치 수라면 $n+2$가 nonfib이 되므로 n과 $n+2$를 짝지을 수 있습니다.

- 결론적으로, nonfib들을 크기 순으로 2개씩 짝지으면 문제에서 원하는 수열을 얻을 수 있습니다.
- 이제 n이 주어졌을 때 a_n은 다음과 같이 구할수 있습니다.
- $n = f_{n'}$이면, $a_n = a_{f_{n'}} = f_{a_{n'}}$을 구하면 됩니다. 이를 n이 nonfib이 될 때까지 반복합니다.
- 10^{18} 이하의 피보나치 수의 개수는 100개 미만이므로 모두 계산해 놓고 nonfib n이 몇 번째 nonfib인지 구할 수 있습니다. 홀수번째이면 a_n은 다음 nonfib이고, 짝수번째이면 이전 nonfib 번째 피보나치 수입니다.

I. 안테나 설치

`dp`, `dp_deque`, `data_structures`

출제진 의도 – **Hard**

- 제출 62번, 정답 18팀 (정답률 28.125%)
- 처음 푼 팀: **뭉끼얏호우** (뭉, 뭉하하하하하하, 뭉탱이), 56분
- 출제자: `cdjs1432`

- j번째 집부터 i번째 집까지 모든 집이 요구하는 네트워크 연결 속도를 하나의 안테나로 제공하기 위한 안테나의 최소 세기를 $f(j,i)$라 합시다.
- 그러면, $f(j,i)$는 다음과 같이 계산할 수 있습니다.
- $f(j,i) = i - j + \max_{k=j \dots i} a_k$
- 이때, 편의상 $\max_{k=j \dots i} a_k$를 $\max a(j,i)$라 합시다.
- 또한, $f(j,i) \le P$를 만족하는 최소 j를 x라 합시다.
- $f(j,i) \le f(j,i+1)$을 항상 만족하므로, 모든 i 값에 대한 x 값은 amortized $\mathcal{O}(N)$으로 계산해줄 수 있습니다.

- 이제, 1 번째 집부터 i 번째 집까지 모든 집이 요구하는 네트워크 연결 속도를 제공하는 안테나 세기의 합의 최솟값을 $dp[i]$ 라 하면, 다음과 같이 계산해줄 수 있습니다.
  - $dp[0] = 0$
  - $dp[i] = \min_{j=x \ldots i} dp[j-1] + f(j, i)$
- 그러나 별도 최적화 없이 위 식을 계산하면 $\mathcal{O}\left(N^2\right)$ 으로 시간초과가 나므로, 위 식을 최적화해봅시다.

- 임의의 $x \le j < i$를 만족하는 j에 대해, $dp[j-1] + f(j,i)$와 $dp[j] + f(j+1,i)$의 대소를 비교해봅시다.
- a_j 및 $\max a(j,i)$에 값에 따라, 다음의 두 경우를 고려합시다.
  1. $a_j \le \max a(j,i) = \max a(j+1,i)$인 경우
     - a_j가 구간 $[j,i]$에서 유일한 최댓값이 아닌 경우를 의미합니다.
  2. $a_j = \max a(j,i) > \max a(j+1,i)$인 경우
     - a_j가 구간 $[j,i]$에서 유일한 최댓값인 경우를 의미합니다.

1. $a_j \le \max a(j,i) = \max a(j+1,i)$ 인 경우
   - 이 경우, $f(j,i) = f(j+1,i) - 1$ 를 만족합니다.
   - 또한, dp 식의 정의에 의해 $dp[j-1] \le dp[j] - 1$ 을 항상 만족합니다.
   - 따라서, $dp[j-1] + f(j,i) \le dp[j] + f(j+1,i)$ 를 항상 만족합니다.
2. $a_j = \max a(j,i) > \max a(j+1,i)$ 인 경우
   - 이 경우, $f(j,i) = a_j + i - j$, $f(j+1,i) = \max a(j+1,i) + i - j - 1$ 이 됩니다.
   - 따라서, $dp[j-1] + a_j$ 와 $dp[j] + \max a(j+1,i) - 1$ 의 대소비교를 해야 합니다.

- 따라서, $dp[i]$를 계산하는 과정에서, a_j가 구간 $[j, i]$에서 유일한 최댓값이 아니라면 항상 $dp[j] + f(j+1, i)$를 고르는 것보다 $dp[j-1] + f(j, i)$를 고르는 것이 이득입니다.
- a_j가 구간 $[j, i]$에서 유일한 최댓값인 j에 대해 $dp[j-1] + f(j, i)$의 값을 multiset 등의 자료구조에 미리 저장해둡시다.
- 또한, a_i가 구간 $[j, i]$에서 유일한 최댓값이 아니라면, 미리 저장해둔 $dp[j-1] + f(j, i)$ 값은 모두 i값에 따라 1씩만 증가하므로, 저장할 때 i에 대한 항을 제거한 뒤 최종 dp값을 반영할 때만 값을 i 만큼 증가시켜서 게산해 주면 됩니다.
- 따라서, $x \le j \le i$인 모든 j에 대해 a_j 값을 항상 monotone하게 관리하는 deque를 활용하면 $\mathcal{O}(N \log N)$으로 문제를 해결할 수 있습니다.
- deque과 multiset을 활용하는 풀이 외에도, lazy segtree 등의 자료구조를 활용해도 동일한 시간복잡도로 문제를 해결할 수 있으나, 대신 구현량 및 상수 시간이 더 커집니다.

J. 회전초밥

`string,kmp,number_theory`

출제진 의도 – **Hard**

- 제출 2번, 정답 0팀 (정답률 0.000%)
- 처음 푼 팀: –
- 출제자: `hyperbolic`

먼저, 문제를 요약하면 다음과 같습니다.

1. $1 \le i \le N$인 i에 대하여, $a_i = (a_i + b_i) \bmod K$ 갱신
2. a_i, b_i를 회전
3. $1. \sim 2.$을 반복하며 모든 a_i가 0이 되는 가장 빠른 시점 찾기

초기 몇번의 행동을 반복하며 a_i에 더해지는 b_j의 규칙을 찾아보면 다음과 같습니다.

- 처음 $N-i+1$번은 b_i가 a_i에 더해집니다. 그 후의 N번은 b_{i+1}이 a_i에 더해집니다. 그 후의 N번은 b_{i+2}가 a_i에 더해집니다...
- N번의 b_{N+1}이 a_i에 더해진 후에는 N번의 b_1이 a_i에 더해집니다. 그 후의 N번은 b_2가 a_i에 더해집니다...

b_1 이 N 개, b_2 가 N 개, ..., b_{N+1} 이 N 개가 차례대로 등장하는 순환하는 수열을 생각해봅시다.
그리고, 이 수열의 누적합을 S_i 라고 정의하겠습니다.
r 번의 행동을 하였다고 가정할때, a_i 에는 총 $S_{r+(N+1)(i-1)} - S_{(N+1)(i-1)}$ 이 더해집니다.
그러므로, 저희가 풀어야하는 문제를 다음과 같이 변형할 수 있습니다.

- 모든 i 에 대하여 $a_i + S_{r+(N+1)(i-1)} - S_{(N+1)(i-1)} == 0 \bmod K$ 가 되는 최소의 r 을 출력한다.
  만약, 그러한 r 이 없다면 대신 -1 을 출력한다.

$c_i = S_{(N+1)(i-1)} - a_i \bmod K$라고 정의하겠습니다. 그러면, 문제를 다음과 같이 변형할 수 있습니다.

- 모든 i에 대하여 $c_i == S_{r+(N+1)(i-1)} \bmod K$가 되는 최소의 r을 출력한다.

이 문제는 r을 고정 시킨 후, $\{S_r, S_{r+(N+1)}, S_{r+2(N+1)}, \cdots\}$라는 문자열에서 $\{c_1, c_2, \cdots, c_N\}$이라는 문자열이 제일 처음 등장하는 시점을 찾는 문제로 생각할 수 있습니다. 이러한 문제는 KMP 알고리즘을 사용하면 해결할 수 있습니다.

$0 \le r \le N$인 r에 대해서만 KMP을 적용하면 충분하며, $N(N+1)K$번의 연산 후 a_i, b_i 상태가 완전히 원래대로 돌아오므로, 문자열 S의 길이도 $N(N+1)K$이하라고 가정하여도 충분합니다.

결론적으로, $\mathcal{O}\left(N^2K\right)$의 시간으로 이 문제를 도전해 볼 수 있습니다.

하지만 $N \le 2\,000, K \le 1\,000\,000$이기 때문에 $\mathcal{O}\left(N^2 K\right)$으로는 시간초과를 받게 됩니다.

기존 문제를 다음과 같이 수정해봅시다.

- 모든 i와 적당한 상수 D에 대하여, $c_i + D == S_{r+(N+1)(i-1)} \bmod K$가 되는 최소의 r을 출력한다.

이 문제는 r을 고정 시킨 후, $\{S_{r+(N+1)} - S_r, S_{r+2(N+1)} - S_{r+(N+1)}, \cdots\}$라는 문자열에서 $\{c_2 - c_1, c_3 - c_2 \cdots, c_N - c_{N-1}\}$이라는 문자열이 제일 처음 등장하는 시점을 찾는 문제로 생각할 수 있습니다.

또한, 이 문제의 경우 고려해야 하는 문자열 S의 길이가 $N(N+1)K$에서 $N(N+1)$로 줄어들게 됩니다. 그러므로, $\mathcal{O}\left(N^2\right)$의 시간으로 이 문제의 해답이 되는 모든 $0 \le r < N(N+1)$을 찾을 수 있습니다.

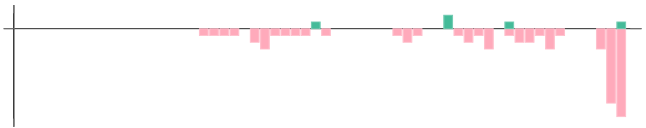
- 위 문제의 해답이 되는 r에 대하여, r번의 연산을 했을 경우 적당한 상수 D가 존재하여 모든 a_i가 D가 됩니다.
- 또한, $N(N+1)$번의 연산을 했을 경우 적당한 상수 M이 존재하여 모든 a_i에 M이 더해지게 됩니다.
- 그러므로, $r+xN(N+1)$번의 연산을 했을 경우 모든 a_i는 $D+Mx$가 됩니다.
  $D+Mx==0 \bmod K$이 되는 최소의 x를 찾았다면, $r+xN(N+1)$이 저희가 최종적으로 원하는 답의 후보가 됩니다. M의 역원을 $\mathcal{O}(K)$에 미리 찾아놓았다면, x는 $\mathcal{O}(1)$에 찾을 수 있습니다.
- 가능한 모든 r에 대하여, $r+xN(N+1)$의 최솟값을 출력하면 회전초밥 문제의 정답이 됩니다.
- 시간 복잡도는 $\mathcal{O}\left(N^2+K\right)$입니다. $N=1$인 경우나 $M=0$이 되는 경우 등등 예외처리에 주의해주세요.

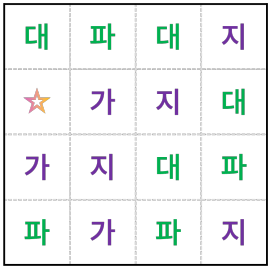
K. 그건 가지가 아니라 대파예요

`game_theory`, `ad_hoc`

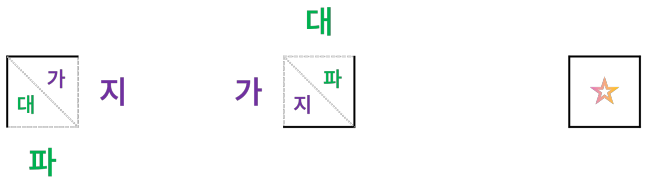
출제진 의도 – **Challenging**

- 제출 67번, 정답 5팀 (정답률 7.463%)
- 처음 푼 팀: **멕시코시티노점상에서타코사먹는윤창기** (주머니에서폰훔쳐가기, 뒷통수치고튀기, 타코뺏어먹기), 148분
- 출제자: `functionx`

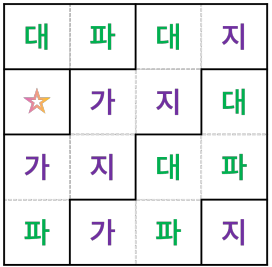




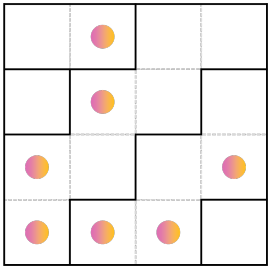
- 게임판입니다.



- 인접한 두 격자 사이의 경계선 중,
- 사용되지 않는 경계선을 모두 그립니다.

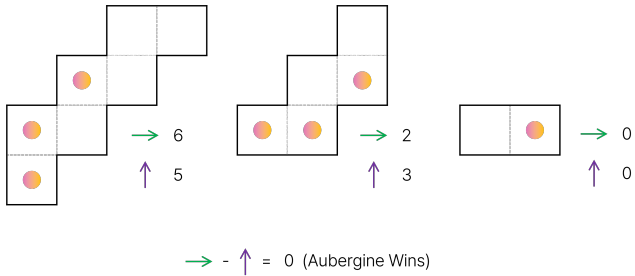


- 게임판은 지그재그 모양으로 분리됩니다.
- **가**, **파**에 돌을 놓고 **지**, **대**를 비웁니다.



- 게임은 돌을 옮기는 게임으로 바뀝니다.
- 스칼리온은 돌을 오른쪽으로, 어버진은 돌을 위쪽으로 옮겨야 합니다.

K. 그건 가지가 아니라 대파예요



- 나눠진 개별 게임판에서 순서 상관없이 돌을 최대한 많이 옮깁니다.
- 가로로 옮긴 횟수를 A, 세로로 옮긴 횟수를 B라고 합시다.
- 놀랍게도, 스칼리온이 이기는 조건은 $A - B$**의 합이 양의 정수** 입니다.

K. 그건 가지가 아니라 대파예요

- 개별 게임판에서 왼쪽 아래 칸에 돌이 없으면 무시할 수 있습니다.
- 개별 게임판에서 왼쪽 아래 칸에 돌이 있고 그 다음 칸에 돌이 없으면,
  - 그 돌이 어디까지 가더라도 돌을 먼저 옮긴 쪽이 0개 이상 1개 이하의 턴을 더 씁니다.
  - 따라서 이 돌은 옮기는 게 무조건 이득입니다.
- 개별 게임판에서 왼쪽 아래 칸에 돌이 있고 그 다음 칸에 돌이 있으면,
  - 두 번째 돌을 옮긴다면 상대방에서 첫 번째 돌을 옮기는 기회를 주게 됩니다.
  - 따라서 두 번째 돌을 옮기지 않는 것이 이득이고 첫 번째 돌은 옮기는 게 불가능합니다.
  - 즉, 첫 번째 돌과 두 번째 돌은 무시할 수 있습니다.
- 3가지 조건에 대해서 재귀적으로 계산하면 게임의 승리조건이 $A - B$의 합임을 증명할 수 있습니다.

- $A - B$의 합을 구하는 방법은 다양한데, 여백이 부족하므로 재귀를 이용한 방법만 설명합니다.
  - 첫 번째 칸(맨 왼쪽/아래)에 돌이 없으면 해당 칸을 지웁니다.
  - 첫째 칸에 돌이 있고 둘째 칸에 돌이 없으면 돌을 옮기고, 옮긴 방향에 따라 정답을 1 증가/감소시킵니다.
  - 첫째 칸과 둘째 칸에 모두 돌이 있으면 두 칸을 모두 지웁니다.
- 시간복잡도는 $\mathcal{O}(NM)$ 입니다.

L. 나무늘보

`trees`, `divide_and_conquer`

출제진 의도 – **Easy**

- 제출 92번, 정답 44팀 (정답률 47.312%)
- 처음 푼 팀: **DDT** (두, 둥, 탁), 17분
- 출제자: egod1537

- 순회 결과가 동일한 이진트리를 관찰하면 자식이 1 개만 있는 정점들의 자식을 왼쪽 혹은 오른쪽으로 위치를 변경해도 순회 결과가 변하지 않는 점을 관찰할 수 있습니다.
- 그러므로 순회 결과가 동일한 이진트리에서 자식이 1 개만 있는 정점들의 개수를 k라고 하면 문제의 정답은 2^k 이 됩니다.

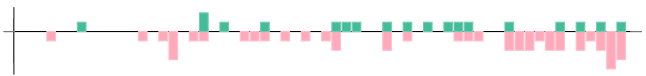
- 전위 순회 결과를 처음부터 순차적으로 탐색하면서 보고 있는 정점을 v라고 합시다.
- 그렇다면 v를 루트로 하는 서브 트리에 속하는 정점들은 후위 순회 결과에서 연속된 구간으로 대응됩니다.
- 분할 정복을 통해 각 정점 v에 대응되는 구간을 구할 수 있으며, 각 정점이 가지고 있는 자식 정점의 개수도 알 수 있습니다.
- 단순하게 분할 정복을 하면 $\mathcal{O}\left(N^2\right)$에 해결할 수 있지만 전위 순회 결과의 정점마다 분할되는 위치를 전처리해 두면 $\mathcal{O}\left(N\right)$에 해결할 수 있습니다.
- 전위, 후위 순회 결과와 동일한 이진트리가 존재하지 않는 경우는 각 정점에 대응되는 구간이 모순되는지 확인하면 됩니다.

M. 산 색칠

`stack`, `dp`, `two_pointer`

출제진 의도 – **Medium**

- 제출 64번, 정답 19팀 (정답률 29.688%)
- 처음 푼 팀: **25세 김동현 마지막 UCPC -많은 응원 부탁드려요-** (김동현, 안정현, 이하린), 34분
- 출제자: `amel`



- 주어진 "그림"을 반드시 완성할 수 있다고 가정하고 풀어봅시다.
- 그래도 될까요?
- 그렇게 정답을 구한 뒤, 실제로 색칠해 본 다음 주어진 그림과 일치하는지 확인합니다.
  - 일치하면 그대로 출력하고, 그렇지 않으면 -1을 출력합니다.
- 만일 주어진 그림이 원래 불가능한 그림이었다면, 옳은 풀이는 반드시 -1을 출력할 것입니다.
- 따라서 그래도 됩니다.

M. 산 색칠

- 산의 정의에 따라 "그림"에서 위로 볼록한 지점은 반드시 색칠을 통해 메꿔주어야 합니다.
- 따라서 위로 볼록한 모든 지점은 적어도 한 번 산의 정상이 되어야 합니다.
- 위로 볼록하지 않은 지점들은 어떻게 해야 할까요?
- 결론부터 말하자면, 어차피 다른 색칠에 의해 반드시 메꾸어 집니다.
  - 위로 볼록하지 않은 지점의 왼쪽, 오른쪽 중 한 방향은 높이가 더 높아집니다.
  - 그 방향대로 나아가다 보면 산의 정상이 반드시 존재합니다.
  - 현재 지점으로부터 산의 정상까지 높이가 단조증가하므로 현재 지점도 메꿔지게 됩니다.

- 풀이를 정리하자면,
  - "그림"의 히스토그램에서 위로 볼록한 지점들을 찾고,
  - 각 지점마다 "도화지"와 높이가 같은 부분을 찾아 산의 정상으로 저장합니다.
  - 그 후 직접 색칠해보아 주어진 그림과 모양이 일치하는 지 확인합니다.
- 그림의 지점과 도화지의 지점을 매칭하는 것은 투포인터 혹은 이분탐색으로 가능합니다.
- 직접 색칠하는 것 역시 선형 시간에 어렵지 않게 구현할 수 있습니다.
- 시간복잡도는 $\mathcal{O}(N+M)$(투포인터) 혹은 $\mathcal{O}\left((N+M)\log(N+M)\right)$(이분탐색) 입니다.