UCPC 2025

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2025

Finals

Official Solutions

본선 해설

전국 대학생 프로그래밍 대회 동아리 연합 · UCPC 2025 출제진

LG전자 · HYUNDAI MOBIS · Jump Trading · FURIOSA · KREW CAPITAL · NEXON · HRT hudson river trading

채널톡 · Jane Street · MOLOCO · codetree · HYUNDAI AutoEver · Samsung Software Membership · STARTLINK · 나정휘 (jhnah917) · 정재헌(gravekper)

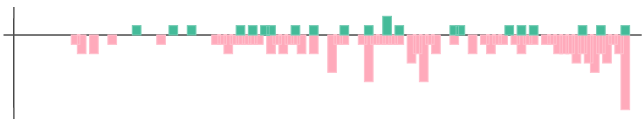
문제		의도한 난이도	출제자
A	트리 이사	**Hard**	mingyu331
B	중력 발전소	**Medium**	ncy09
C	It's a Mod, Mod, Mod, Mod World 2	**Medium**	young_out
D	Wildcard and Query	**Hard**	molamola
E	중간 뒤집기	**Easy**	kep1er07
F	ChannelTalk	**Challenging**	79brue
G	자율 주행 프로그램 개발	**Hard**	leo020630
H	시간선 통합	**Hard**	evenharder
I	골드리치의 비밀 금고	**Easy**	wapas
J	격자 조각 자르기	**Hard**	79brue
K	자동 광고 배치 시스템	**Hard**	bubbler
L	망각의 최장 경로	**Challenging**	queued_q
M	Only Shallow	**Hard**	lighton

A. 트리 이사

`ad_hoc`, `tree`

출제진 의도 – **Hard**

- 제출 128번, 정답 22팀 (정답률 17.188%)
- 처음 푼 팀: **대회중에LLM을구현해서문제를풀리는행복한상상** (Gemini, GPT, Claude), 57분
- 출제자: `mingyu331`
- 가장 짧은 정해: 1533B^{C++}



- 모든 리프 노드에서 부모 노드로 가는 격자 경로는 각 축의 방향으로 $+1, -1$ 이동하는 두 종류로 총 $2d$ 가지의 경우가 있습니다.
- 두 리프 u, v 가 부모에 대해 같은 방향으로 위치한 경우, u 와 v 사이의 거리가 트리를 통해 이동한 거리보다 작기 때문에 조건을 만족할 수 없습니다.
- 즉, 리프 노드의 개수를 l 이라 하면, $2d \geq l$ 이여야 하므로 필요한 최소 차원의 수는 $\left\lceil \frac{l}{2} \right\rceil$ 입니다.

- $\left\lceil \frac{l}{2} \right\rceil$ 차원으로 이사 방법을 구성해봅시다.
- 더미 리프 노드를 하나 추가하는 방식으로 l이 짝수라 가정할 수 있습니다.
- 각 간선 $\{u, v\}$에 대하여 u와 v의 상대적 위치를 정한 후, 이를 통해 절대 위치를 계산하여 구성할 수 있습니다.
- 각 간선의 상대적 위치는 다음과 같이 정할 수 있습니다.
  - 이전 시행에서 선택하지 않은 두 리프 노드 u와 v를 선택합니다.
  - 이번 시행을 i번째 시행이라 했을 때, u와 v 사이의 경로 중 상대 위치가 배정되지 않은 간선을 i번 차원에서 $+1$ 이동하는 것으로 배정합니다.

- 이를 통해 모든 간선의 상대적 위치가 결정된다면, 올바르게 구성이 되었음을 보일 수 있습니다.
- 모든 리프 노드를 dfs 오더링 순서대로 나열하여 $u_1, u_2, \cdots, u_l$ 이라 하였을때, i 번째 시행에서 u_i 와 $u_{i+l/2}$ 를 선택하면 모든 간선의 상대적 위치가 결정됨을 보일 수 있습니다.

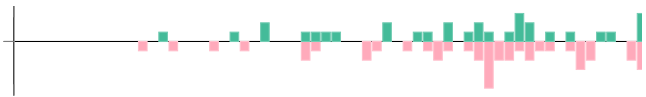
B. 중력 발전소

UCPC 2025

`segtree`, `parametric_search`, `binary_search`, `prefix_sum`

출제진 의도 – **Medium**

- 제출 127번, 정답 43팀 (정답률 33.858%)
- 처음 푼 팀: **인공지능 사용 적발 시 실격 처리 됩니다** (Yann LeCun, Yoshua Bengio, Geoffrey Hinton), 42분
- 출제자: `ncy09`
- 가장 짧은 정해: 1336B[C++]



- 생산되는 에너지의 양이 식으로 어떻게 표현되는지 생각해봅시다.
- 만약 반중력 장치를 사용하지 않는다면, 총 에너지는 $sum = A_1 + \cdots + A_n$ 입니다.
- 만약 j 번 공간에서 i 번 공간으로 올리는 반중력 장치를 사용할 경우, $K(j-i)$ 만큼의 에너지를 추가적으로 사용하며 $A_{i+1} + \cdots + A_j$ 만큼의 에너지를 추가적으로 생산하게 됩니다.
- 이때의 알짜 생산량은 $(A_{i+1} + \cdots + A_j) - K(j-i) = (A_{i+1} - K) + \cdots + (A_j - K)$ 입니다.
- 이는 $(A_i - K)$ 로 이루어진 배열에서의 구간합입니다.

- 즉, 반중력 장치를 사용할 때마다 $(A_i - K)$ 배열의 특정 구간합만큼의 에너지를 얻게 됩니다.
- 많은 에너지를 얻기 위해서는 구간합의 크기가 큰 구간을 선택해야 합니다.
- 이때, 구간들의 조합을 어떻게 선택하더라도 이에 대응되는 반중력 장치들을 모두 사용하는 것이 항상 가능합니다. (매 순간 사용할 수 있는 반중력 장치를 사용하면 됩니다.)
- 그러므로 구간합이 최대인 구간부터 선택하는 것이 무조건 유리합니다.

- 따라서 이 문제는, "구간합이 가장 큰 M개의 구간 구하기"와 동치입니다.
- 그러나 N^2개의 구간을 전부 탐색하기엔 무리입니다.
- 이 문제를 결정 문제로 바꾸어봅시다.
- 즉, "양의 정수 X에 대해 구간합이 X 이상인 구간이 M개 이상인가?"
- 해당 결정 문제의 정답이 변하는 경계값을 생각해보면, M번째로 큰 구간합의 크기를 알 수 있습니다.

- 각각의 결정 문제를 빠르게 해결할 수 있다면, 경계값은 이분 탐색으로 찾을 수 있습니다.
- 결국 구해야 하는 것은 구간합이 X 이상인 구간의 개수, 그리고 그 구간합들의 합입니다.
- 이는 다양한 방법이 있으며, 누적합과 세그먼트 트리의 조합 혹은 분할 정복으로 해결 가능합니다.
- 시간 복잡도는 $\mathcal{O}\left(30N\log N\right)$ 입니다.

구현 시 주의해야 할 부분입니다.

- 0 이상의 구간합만 고려해야 하므로, 이를 고려해 이분 탐색의 하한을 설정해야 합니다.
- 정답의 오버플로우에 주의합니다.
- 경계값에서 같은 구간합이 여러 개 존재할 수 있습니다. 최종 답을 계산할 때 이를 고려해야 합니다.

C. It's a Mod, Mod, Mod, Mod World 2

UCPC 2025

`number_theory`, `pigeonhole_principle`, `randomization`

출제진 의도 – **Medium**

- 제출 499번, 정답 45팀 (정답률 9.018%)
- 처음 푼 팀: **BIGSHOT** (ITS, PS, TIME), 19분
- 출제자: `young_out`
- 가장 짧은 정해: $938B^{C++}$, $731B^{Python}$

- 임의의 K에 대해서, 해시맵을 이용하면 $O(N)$에 최대 부분집합 크기를 구할 수 있습니다.
- K의 후보로는 소수만 관찰해도 충분합니다.
- 만약 최대 부분집합 크기를 만들 수 있는 합성수 K가 존재한다면, 합성수 K의 약수 또한 최대 부분집합 크기를 만들 수 있기 때문입니다.
- 10^9 이하인 소수의 개수는 약 5×10^7 개로, 시간 내에 모두 관찰하는 것은 불가능합니다. 따라서 관찰할 후보를 줄여야 합니다.

- 비둘기집 원리에 의해, $K = 2$일 때 답은 $\lceil N/2 \rceil$ 이상임이 보장됩니다. 따라서 A의 원소들 중 $\lceil N/2 \rceil$ 개 이상은 정답을 가리키는 부분집합에 포함됩니다.
- 따라서 임의의 두 원소 A_i, A_j를 뽑았을 때, $|A_i - A_j|$의 약수에 정답인 K가 포함될 확률은 $1/4$ 이상입니다.
- K의 후보들은 소인수분해를 통해 $O(\sqrt{|A_i - A_j|})$에 구할 수 있습니다.
- 뽑는 시행을 30번 반복한다면 $1 - (3/4)^{30} \approx 0.9998$ 이상의 확률로 K 값을 구할 수 있습니다.
- 따라서, 30번 반복했을 때 시간복잡도 $O(30(\sqrt{10^9} + N \log 10^9))$에 해결 가능합니다.

D. Wildcard and Query

`suffix_array`, `segment_tree`

출제진 의도 – **Hard**

- 제출 28번, 정답 0팀 (정답률 -%)
- 처음 푼 팀: –
- 출제자: `molamola`
- 가장 짧은 정해: 4789B^{C++}

- T에 wildcard가 포함되어 있을 때, S와 T의 매칭 여부를 $\mathcal{O}(|T|)$에 구하는 문제입니다.
- T를 wildcard를 기준으로 분리했을 때 $*R_1 * R_2 * \ldots * R_N *$ 이라고 해 봅시다.
- S 위에서 R_1이 최대한 빠르게 등장하는 위치를 찾고,
- 그 위치부터 R_2이 최대한 빠르게 등장하는 위치를 찾고,
- ...
- 이것을 반복하면 매칭 여부를 판단할 수 있습니다.

- 매칭이 유일한지 여부는 반대로 R_N 부터 최대한 뒤쪽으로 매칭하면 됩니다.
- 이는 S를 뒤집고, T_i 들을 전부 뒤집은 뒤 같은 매칭이 구해지는지 여부로 확인하는 것이 편합니다.
- 가장 어려운 부분은 (t, R_j) 에 대해 S[t...]에서 R_j 의 최초 등장 위치를 찾는 것입니다.
- 이는 Suffix Array를 활용하여 해결할 수 있습니다.

- 먼저 S, T_1, T_2, ..., T_Q 를 모두 이은 문자열 P 의 Suffix Array를 만듭니다.
- 이 때 R_j 들은 P 의 substring입니다.
- R_j 로 시작하는 Suffix의 시작 인덱스들은 Suffix Array 위에서 구간으로 나타납니다.
- 이 구간은 LCP Array를 사용해 찾을 수 있습니다.
- 이들 중 t 이상이면서 최소 인덱스를 찾으면 됩니다.
- 이는 t 가 작은 것부터 차례로 처리하는 offline + segment tree를 사용하거나,
- PST를 사용하면 해결할 수 있습니다.

- 쿼리 양 끝에 wildcard가 있는 경우, wildcard가 2개 연속한 경우 처리에 유의합시다.
- 모든 문자열 길이의 합을 L이라 할 때 시간 복잡도는 $\mathcal{O}\left(L \log L\right)$ 입니다.

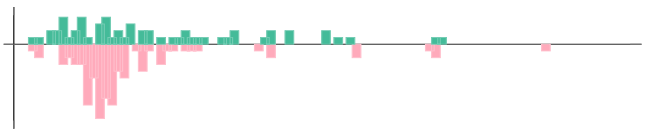
E. 중간 뒤집기

UCPC 2025

ad_hoc

출제진 의도 – **Easy**

- 제출 147번, 정답 59팀 (정답률 40.136%)
- 처음 푼 팀: **공백공백으로 시작하거나 끝날 수 없습니다마침표공백** (jjang36524, sorohue, heeew), 7분
- 출제자: kep1er07
- 가장 짧은 정해: 340B^{C++}, 228B^{Python}



- A의 연속된 부분 수열을 한 번 뒤집어서 얻을 수 있는 서로 다른 수열의 개수를 구하는 문제입니다.
- $f(i, j)$에서 $i = j$인 경우 수열 A의 변화는 없습니다. 따라서 본래의 수열 A는 항상 만들 수 있습니다. 앞으로 $i < j$임을 가정합시다.
- 만약 $A_i = A_j$인 경우, $f(i, j) = f(i + 1, j - 1)$이므로 다른 i와 j를 선택한 경우와 결과가 같아집니다.
- $A_i \neq A_j$에 대한 핵심적인 관찰을 할 수 있습니다.

- **Observation.** $A_i \neq A_j$인 모든 $i < j$의 쌍에 대하여, $f(i, j)$는 모두 다르다.

- **Proof.** $A_i \neq A_j$, $A_x \neq A_y$, $(i, j) \neq (x, y)$를 만족하는 인덱스 $i < j$, $x < y$를 생각해 봅시다.
- $A_i \neq A_j$이므로 $f(i, j)$와 A가 달라지는 첫 인덱스는 i입니다. 마찬가지의 이유로 $f(x, y)$와 A가 달라지는 첫 인덱스는 x입니다. 따라서 $i \neq x$이면 $f(i, j) \neq f(x, y)$입니다.
- 원소가 일치하지 않는 마지막 인덱스에 대해서도 같은 논리를 적용할 수 있습니다. $f(i, j)$와 A가 달라지는 마지막 인덱스는 j이고, $f(x, y)$와 A가 달라지는 마지막 인덱스는 y이므로 $j \neq y$이면 $f(i, j) \neq f(x, y)$입니다.
- $(i, j) \neq (x, y)$이므로 $i \neq x$와 $j \neq y$ 중 하나 이상이 성립합니다. 따라서 $f(i, j) \neq f(x, y)$입니다.

- 이 관찰에 따라 이 문제는 $A_i \neq A_j$를 만족하는 순서쌍 (i, j)의 개수를 효율적으로 구하는 문제로 귀결됩니다.
- $i < j$인 순서쌍의 전체 개수 $\frac{N(N-1)}{2}$에서 $A_i = A_j$인 순서쌍의 개수를 빼면 됩니다.
- 즉, 모든 정수 k 각각에 대해 $A_i = k$를 만족하는 i의 개수를 c_k라 할 때, $\frac{c_k(c_k-1)}{2}$를 빼면 됩니다.
- 마지막으로, $f(i, j)$가 본래의 수열 A인 경우에 해당하는 1을 더하면 답을 구할 수 있습니다.

F. ChannelTalk

`segtree`, `lazyprop`, `offline_query`

출제진 의도 – **Challenging**

- 제출 21번, 정답 0팀 (정답률 -%)
- 처음 푼 팀: –
- 출제자: `79brue`
- 가장 짧은 정해: 6148B[C++]

- 문제를 어떤 방식으로 접근해야 할까요?
- 가장 먼저 보이는 길은 적당한 자료구조를 이용해 시간에 따른 각 채널의 인원 수 변화를 관리하는 것입니다.
- 그러나, 사람이 한 번 들어갈 때 일어나는 연쇄 작용을 세그먼트 트리나 셋 등의 자료구조로 처리하는 것은 굉장히 어렵습니다. (이러한 방향으로 문제를 해결하는 것도 가능하긴 합니다.)

- 이 문제에서 가장 핵심적인 아이디어는 시간 축에 해당하는 세그먼트 트리를 놓고, 위치를 옮기며 스위핑하는 것입니다.
- 이렇게 하면 스위핑할 때 생기는 이벤트들을 구간 쿼리의 형태로 바꾸는 것이 훨씬 쉬워집니다.
- 이 접근 방식에 대해 더 자세히 알아봅시다.
- 그 전에, 문제 상황에 대한 몇 가지 관찰로 시작해 봅시다.

- **Observation 1.** 최대 수용 인원이 c명인 어떤 채널에 지금까지 한 번이라도 들어온 적 있는 사람 중 찬성 인원이 a명, 반대 인원이 b명일 때, 마지막에 남은 찬성/반대 인원 수는 $a+b$명의 사람이 들어온 순서와 상관없다.

- **Proof.** 다음과 같은 세 가지 경우가 있습니다.
  - 만약 $a+b \le c$, 즉 아직 정원을 초과하지 않았다면, 모든 사람들이 남아 있습니다. 즉, 최종 찬성 인원은 a명, 반대 인원은 b명입니다.
  - 만약 $a+b > c$이고, $a \ge c/2$, $b < c/2$인 경우, 채널 정원을 초과했을 때 반대 인원은 절반을 넘지 못하므로, 찬성 인원이 무조건 반대 인원보다 많습니다. 따라서 찬성 인원만 나가게 되고, 반대 인원은 나가는 일이 없습니다. 즉, 찬성 인원은 $c-b$명, 반대 인원은 b명이 최종적으로 남습니다. 반대의 경우도 마찬가지입니다.
  - 만약 $a+b > c$이고, $a \ge c/2$, $b \ge c/2$라면, 비슷한 원리로 인해 최종 찬성 인원과 반대 인원 모두 $c/2$명씩 남게 됩니다. (이 시점에서는 누가 들어오더라도 해당 의견을 가진 사람이 바로 나가게 됩니다.)

- **Observation 2.** 시간 t까지 채널 x에 한 번이라도 들어온 사람은 아래 두 가지 유형 중 하나이다.
  - 시간 t 또는 그 이전에 채널 x에 직접 들어왔다.
  - 시간 t 또는 그 이전에 특정 채널 y ($y < x$)에 처음 입장했으나, 정원 초과로 인해 채널 x까지 이동했다.
- 이 관찰은 자명하므로 증명은 생략합니다.

- 다음과 같은 방식을 생각해 봅시다.
  - c: 현재 보고 있는 채널 번호
  - A_t, B_t, S_t: t번 쿼리까지 진행했을 때 c번 채널에 한 번이라도 들어온 적 있는 찬성 인원의 수, 반대 인원의 수, 전체 인원의 수
  - 특정한 c 값에 도착하면 해당 채널에 해당하는 인원 추가 업데이트를 처리하고, 해당 채널을 대상으로 하는 쿼리에 답합니다. A_t, B_t는 한 번이라도 들어온 적 있는 인원의 수와 같지만, **Observation 1**에 의해 A_t와 B_t의 값만으로 쿼리 t의 시점에 남아 있는 정확한 인원 수를 구할 수 있습니다.
  - 이후 $c+1$번 채널로 이동하고, 배열 A_t, B_t, S_t에서 인원 초과로 $c+1$번 채널로 이동하는 인원만 남도록 업데이트합니다.

- 이제 A_t, B_t, S_t를 다룰 자료구조는 어떤 업데이트와 쿼리를 지원해야 하는지 자세히 생각해 봅시다.
- 먼저 인원이 추가되는 시점을 생각해 봅시다.
- 시간 t에 찬성 인원 a명 또는 반대 인원 b명이 추가된다면, 단순히 $t' \geq t$인 모든 $A_{t'}$ 또는 $B_{t'}$에 각각 a나 b를 더해 주면 됩니다.
- $S_{t'}$ 역시 비슷한 방식으로 더해 줘야 합니다.
- 따라서 구간 덧셈 업데이트를 지원해야 함을 알 수 있습니다.
- 또한, 인원 수 쿼리에 답하려면 원소 하나의 값을 묻는 쿼리에도 답할 수 있어야 합니다.

- 다음으로, 채널 번호를 1 증가시키며 정원 초과되는 인원만 남기는 부분을 처리하기 위해 어떤 업데이트를 지원해야 하는지 알아봅시다.
- 배열 A_t, B_t, S_t에 모두 접근할 수 있다고 가정한다면, 앞과 같이 세 가지 시간 구간으로 나누는 것이 처리에 편리합니다.
  - 아직 정원을 초과하지 않은 시점
  - 정원을 초과했으나, 채널 정원 C_i에 대해 $C_i/2$명 이상으로 들어온 쪽이 찬성과 반대 중 한쪽뿐이라, 해당 채널에서 다음 채널로 이동하게 되는 사람이 모두 찬성 의견이거나 모두 반대 의견인 시점
  - 정원을 초과했고, 찬성과 반대 모두 $C_i/2$명 이상씩 들어와서, 해당 채널에 최종적으로 찬성과 반대 반반씩 남게 되는 시점

- 각각의 경우에 세 값 A_t, B_t, S_t가 어떻게 바뀌는지 살펴봅시다.
- 첫 번째 케이스의 경우, 모든 사람들이 c번 채널에 남으므로, $c+1$번 채널로 이동되는 사람이 없습니다. 따라서 $A_t = B_t = S_t = 0$으로 갱신해 줘야 합니다.
- 따라서 구간을 특정 값(정확히는 0)으로 바꾸는 업데이트를 지원해야 합니다.
- 세 번째 케이스의 경우, c번 채널에 남는 사람의 수를 정확하게 특정할 수 있습니다. 찬성과 반대 모두 $C_i/2$명씩 남으므로, A_t와 B_t에서 $C_i/2$씩, S_t에서 C_i만큼 빼 줘야 합니다.
- 이 부분을 처리하기 위해서는 앞의 구간 덧셈 업데이트가 필요합니다.

- 문제가 되는 것은 두 번째 케이스입니다.
- 일반성을 잃지 않고, $A_t \geq C_i / 2$인 경우를 생각해 봅시다.
- 기존의 값이 (A_t, B_t, S_t)였다고 해 봅시다.
- 먼저 반대 의견을 가진 사람은 모두 현재 채널에 남을 것이므로, $B_t = 0$으로 갱신해 줘야 합니다.
- 총 C_i 명이 현재 채널에 남을 것이므로, S_t에서 C_i를 빼 줘야 합니다.
- 여기까지는 구간을 상수로 정하는 업데이트와 구간 덧셈 업데이트로 충분히 가능합니다.

- 그러나 A_t 를 갱신하는 부분에서 문제가 생깁니다.
- A_t 는 $S_t - C_i$ 로 갱신해 줘야 합니다.
- 그러나 구간에 대해 이러한 형태로 갱신하는 것은 꽤나 어렵습니다.
- Persistent한 구조를 이용하면 Range Copy 업데이트를 구현하는 것이 가능하지만, Lazy Proapgation과 함께 구현하는 것은 가능할 것으로 추정되나 굉장히 어렵습니다.
- 대신 일반적인 세그먼트 트리로 이 문제를 해결해 봅시다.

- 앞서 봤듯이 일반적인 세그먼트 트리로는 두 번째 유형의 업데이트를 처리하지 못합니다.
- 그러나 한 가지 알 수 있는 사실은 A_t 나 B_t 둘 중 하나는 업데이트가 가능하다는 사실입니다.
- 그렇다면, A_t 와 B_t 둘 중 하나만을 관리하고 있는다면 어떨까요?
- S_i 역시 실시간으로 관리하므로, A_t 나 B_t 중 관리하지 않는 쪽의 값을 아는 것도 가능할 것입니다.

- 따라서, 각 t에 대해 관리하는 정보를 다음과 같이 바꿉니다.
- 기존: A_t, B_t, S_t
- 새로운 방식: A_t 또는 B_t (둘 중 하나의 값), 둘 중 어떤 쪽을 저장하고 있는지, S_t
- 위와 같이 저장하는 값을 바꾸면 놀랍게도 Lazy Propagation을 통해 값을 지속적으로 관리하는 것이 가능합니다.
- 지금부터 어떤 방식으로 Lazy Propagation을 관리해야 하는지 살펴봅시다.

- Lazy Propagation에 대해 생각하기 앞서 필요한 업데이트와 쿼리의 종류를 다시 한 번 정리해 봅시다.
  - 인원 추가를 위한 구간 덧셈 업데이트 (A_t, B_t, S_t 각각에 상수만큼 더함)
  - 인원 쿼리에 답하기 위한 점 쿼리 (A_t, B_t, S_t 값 하나를 구해야 함)
  - 채널 위치를 변경하기 위해 시간 구간을 셋으로 나누기 위한 Walking on Segment Tree 지원 (A_t, B_t, S_t 값을 전체적으로 이용해 세그먼트 트리 위의 이분 탐색)
  - 첫 번째 종류의 시간대 업데이트를 위한 구간 상수 변경 업데이트 (A_t, B_t, S_t를 모두 0으로 변경)
  - 두 번째 종류의 시간대 업데이트를 위해, 구간에서 A_t나 B_t 중 한쪽을 0으로 설정하고, S_t에서 상수를 빼는 업데이트
  - 세 번째 종류의 시간대 업데이트는 위에서 서술된 업데이트 유형으로 처리할 수 있습니다.

- Lazy Propagation 상태를 다음 네 가지 중 하나로 정의합니다. (앞으로 편의상 A_t, B_t, S_t의 값을 a, b, s로 표기합니다.)
  - 상태 0: $a' := a + lx$, $s' := s + ly$
  - 상태 1: $a' = lx$, $s' += ly$
  - 상태 2: $b' = lx$, $s' += ly$
  - 상태 3: $a' = lx$, $s' = ly$
- Lazy Propagation을 노드에 push하는 부분은 자명합니다.

– 다음 코드와 같이 Lazy Propagation 상태를 서로 합칠 수 있습니다.

```
inline void mergeLazy(int &lst2, ll &lx2, ll &ly2, int &lst1, ll &lx1, ll &ly1){
    if(lst1 == 0){
        if(lst2 == 0)      lst1 = 0, lx1 += lx2, ly1 += ly2;
        else if(lst2 == 1) lst1 = 1, lx1 = lx2, ly1 += ly2;
        else if(lst2 == 2) lst1 = 2, lx1 = lx2, ly1 += ly2;
        else               lst1 = 3, lx1 = lx2, ly1 = ly2;
    }
    else if(lst1 == 1){
        if(lst2 == 0)      lst1 = 1, lx1 += lx2, ly1 += ly2;
        else if(lst2 == 1) lst1 = 1, lx1 = lx2, ly1 += ly2;
        else if(lst2 == 2) lst1 = 2, lx1 = lx2, ly1 += ly2;
        else               lst1 = 3, lx1 = lx2, ly1 = ly2;
    }
    else if(lst1 == 2){
        if(lst2 == 0)      lst1 = 2, lx1 += ly2 - lx2, ly1 += ly2;
        else if(lst2 == 1) lst1 = 1, lx1 = lx2, ly1 += ly2;
        else if(lst2 == 2) lst1 = 2, lx1 = lx2, ly1 += ly2;
        else               lst1 = 3, lx1 = lx2, ly1 = ly2;
    }
    else if(lst1 == 3){
        if(lst2 == 0)      lst1 = 3, lx1 += lx2, ly1 += ly2;
        else if(lst2 == 1) lst1 = 3, lx1 = lx2, ly1 += ly2;
        else if(lst2 == 2) lst1 = 3, lx1 = (ly1 + ly2) - lx2, ly1 += ly2;
        else               lst1 = 3, lx1 = lx2, ly1 = ly2;
    }
}
```

- 해당 방식으로 세그먼트 트리를 구현하면 문제에서 필요한 모든 업데이트와 쿼리를 $\mathcal{O}\left(\log Q\right)$에 처리할 수 있습니다.
- 따라서 문제를 $\mathcal{O}\left((N+Q)\log Q\right)$에 해결할 수 있습니다.

G. 자율 주행 프로그램 개발

`trees`

출제진 의도 – **Hard**

- 제출 49번, 정답 8팀 (정답률 16.327%)
- 처음 푼 팀: **BIGSHOT** (ITS, PS, TIME), 123분
- 출제자: `leo020630`
- 가장 짧은 정해: 2144B[C++]

- 문제 풀이에 필요한 성질을 하나씩 찾아 봅시다. 편의상 거점을 정점으로, 도로를 간선으로 칭하겠습니다.
- **Claim 1.** 정점 A와 정점 B 사이의 거리가 홀수라면 불가능하다.
- 프로그램을 두 번 실행해 이동한 거리는 항상 짝수이기 때문입니다.
- **Claim 2.** 가장 짧은 프로그램은 후진하다 전진하는 형태이다. 즉, 항상 B가 모두 등장한 후 L과 R이 등장한다.
- 프로그램에 LB나 RB가 부분 문자열로 등장한다면 이를 없애도 실제 경로에 영향을 주지 않기 때문에 지워주는 것이 항상 이득입니다.

- 이제 조건을 만족하는 프로그램 S의 형태를 더 명확히 정의해 봅시다.
  - S는 B가 x번 등장한 후, L 혹은 R이 y번 등장하는 문자열입니다.
  - 이때 $\mathrm{dep}(i)$를 정점 1에서 정점 i까지의 거리로 정의하면, $x - y = \dfrac{\mathrm{dep}(A) - \mathrm{dep}(B)}{2}$ 여야 합니다.
- 프로그램 S를 두 번 실행하면서 거치는 정점을 다음과 같이 정의합시다.
- 정점 $A \xrightarrow[\text{B } x\text{번}]{}$ 정점 $P \xrightarrow[\text{L/R } y\text{번}]{}$ 정점 $C \xrightarrow[\text{B } x\text{번}]{}$ 정점 $Q \xrightarrow[\text{L/R } y\text{번}]{}$ 정점 B

- 정점 C를 고정해 봅시다.
- 정점 P는 정점 A와 정점 C의 최소 공통 조상입니다.
- $x = \mathrm{dep}(A) - \mathrm{dep}(P)$, $y = x - \dfrac{\mathrm{dep}(A) - \mathrm{dep}(B)}{2}$ 입니다.
- 정점 Q는 정점 C의 x번째 조상 정점입니다.
- 따라서 C를 고정하면 x, y, P, Q를 모두 알 수 있습니다.

- 이때 프로그램 S가 **에러** 없이 올바르게 작동할 조건은 다음과 같습니다.
  1. $\mathrm{dep}(C) \geq x$여야 합니다. 즉, 정점 Q가 존재해야 합니다.
  2. $P \to C$ 경로와 $Q \to B$ 경로에서 사용하는 명령어가 동일해야 합니다.
- 이때 2번 조건에 의해 임의의 x에 대해 **에러** 없이 동작하는 프로그램은 유일함을 알 수 있습니다. 따라서 조건을 만족하는 프로그램이 존재한다면 가장 짧은 프로그램은 유일합니다. (이 문제는 스페셜 저지를 사용하지 않습니다!)
- 이제 각 C에 대해 두 조건을 확인하면 문제를 해결할 수 있습니다.
- 1번 조건은 쉽게 확인할 수 있습니다. 2번 조건은 다소 어려우며, 크게 두 가지 방법으로 확인할 수 있습니다.

1. Intended Solution

- 모든 정점을 C의 후보로 고려합니다. 후보로 결정된 각 C를 B에서 루트로 가는 방향에 맞춰 **끌어올려** 봅시다. 이 과정은 y를 1씩 증가시킵니다.
- 끌어올리는 과정에서 C가 A의 조상 정점이 된다면 그 정점이 P가 됩니다.
- 이때 C의 x번째 조상이 Q라면 조건을 만족하는 프로그램이 만들어집니다.
- 한 정점이 다른 정점의 조상임을 판별하는 것은 Euler Tour Technique으로 $\mathcal{O}(1)$ 또는 Sparse Table으로 $\mathcal{O}(\log N)$에 처리할 수 있습니다.
- 이 알고리즘은 $\mathcal{O}(N^2)$ 또는 $\mathcal{O}(N^2 \log N)$에 동작하므로 시간 초과를 받게 됩니다.

- 정점 C가 만족해야 하는 조건을 생각해 봅시다.
- $\mathrm{dep}(C) = \dfrac{\mathrm{dep}(A) + \mathrm{dep}(B)}{2}$ 이어야 함을 어렵지 않게 알 수 있습니다.
- 이를 만족하는 모든 정점을 C의 후보로 고려한 채로 같은 알고리즘을 시행해 봅시다.
- 이 알고리즘은 $\mathcal{O}(N)$ 또는 $\mathcal{O}(N \log N)$에 동작합니다.
- 이유를 분석해 봅시다.

- 정점 C를 끌어 올리는 과정에서, 각 간선이 사용되는 횟수를 알아봅시다.
- **Claim 3.** 해당 알고리즘에서 각 간선은 최대 한 번 사용된다.
- 간선의 부모 정점을 p라 합시다. 간선이 두 번 사용되는 경우는 어떤 상황일까요?
- p의 자손 정점 중 다음과 같은 조건을 만족하는 서로 다른 두 정점 c_1, c_2가 존재해야 합니다.
  - $\mathrm{dep}(c_1) = \mathrm{dep}(c_2) = \dfrac{\mathrm{dep}(A) + \mathrm{dep}(B)}{2}$
  - p에서 c_1로 가는 경로에 사용하는 명령어와 c_2로 가는 경로에 사용하는 명령어가 같아야 한다.
- 이는 불가능하므로, 각 간선은 최대 한 번 사용됩니다.
- 따라서 상기한 알고리즘은 $\mathcal{O}(N)$ 또는 $\mathcal{O}(N \log N)$에 동작하므로 문제를 해결할 수 있습니다.

2. Hashing + Sparse Table

- `table[i][x]`를 정점 x의 2^i 번째 조상에서 정점 x로 향하는 경로 상의 명령어를 해싱한 값으로 정의합시다.
- 이를 통해 각 C에 대해 $P \to C$ 경로와 $Q \to B$ 경로에서 사용하는 명령어가 일치하는지를 $\mathcal{O}(\log N)$에 확인할 수 있습니다.
- 시간복잡도는 $\mathcal{O}(N \log N)$ 입니다.

H. 시간선 통합

UCPC 2025

`constructive`

출제진 의도 – **Hard**

- 제출 91번, 정답 12팀 (정답률 13.187%)
- 처음 푼 팀: **공백공백으로 시작하거나 끝날 수 없습니다마침표공백백** (jjang36524, sorohue, heeew), 97분
- 출제자: `evenharder`
- 가장 짧은 정해: 2707B[C++]

- 순열 형태의 시간선의 목록이 주어지면, 인접한 값 중 하나를 남기면서 최종적으로 현재 시간선의 시각만 남기는 문제입니다.
- 이 때, 최댓값 / 최솟값을 취할 수 있는 횟수에 제한이 있습니다.
- 현재 시각을 제외하면 나머지 시각은 대소 관계만 중요합니다.
- 때문에 원래 시간선의 시각을 0으로 바꾸고, 나머지 시간선의 값은 현재 시간선의 시각보다 크면 $+1$, 작으면 -1로 고려해도 동일합니다.
- 또한, 병합할 때 0이 한 번 사라지면 복구가 불가능하기 때문에, 0 기준 왼쪽과 오른쪽을 독립적으로 고려할 수 있습니다.

인접한 두 시간선의 시각이 일치하면 어떤 통합을 사용해도 결과가 동일하기 때문에, 하나로 합쳐서 고려한 후 맨 마지막에 남은 통합을 수행해도 됩니다. 그러므로 이후 설명은 인접한 시간선의 시각은 서로 다르다고 가정합니다.

- 합쳐야 할 시간선이 존재하지 않으면 연산이 필요하지 않습니다.
- 합쳐야 할 시간선이 $+1$ 1개면 있으면 최솟값 통합이 1회 필요합니다.
- 합쳐야 할 시간선이 -1 1개면 있으면 최댓값 통합이 1회 필요합니다.

- 합쳐야 할 시간선에 $+1$와 -1이 둘 다 있으면, 필수적으로 필요한 연산이 어떻게 될까요?
- 최댓값 통합 1회와 최솟값 통합 1회만 필수적으로 필요하고, 나머지는 임의의 연산으로 구성할 수 있습니다.
- 기저 조건으로, 길이 2인 `0 -1` `+1`은 최댓값 통합 1회와 최솟값 통합 1회가 필요합니다.

`*m : minimum merge`
`*M : maximum merge`

`0 [+1 -1] -> [0 +1] -> 0 (Mm)`

- 길이 3인 `0 -1 +1` `-1 0 +1` `-1 +1`도 최댓값 통합 1회, 최솟값 통합 1회, 임의의 통합 1개로 해결할 수 있습니다.

  * `m` : minimum merge
  * `M` : maximum merge
  * `_` : any merge

  `0 [+1 -1] +1 -> 0 [+1 +1] -> [0 +1] -> 0 (M_m)`
  `0 [-1 +1] -1 -> 0 [-1 -1] -> [0 -1] -> 0 (m_M)`

- $M > 2$개 시간선이 $+1$과 -1로 교대하면 (최댓값 통합 1회 / 임의의 통합 1회)나 (최솟값 통합 1회 / 임의의 통합 1회)로 $M - 2$개의 교대하는 시간선으로 줄일 수 있습니다.

```
* m : minimum merge
* M : maximum merge
* _ : any operator

0 ... +1 [-1 +1] ... -> 0 ... [+1 +1] -> 0 ... +1 ... (M_)
0 ... -1 [+1 -1] ... -> 0 ... [-1 -1] -> 0 ... -1 ... (m_)
```

이에 따라, 양쪽에 각각 시간선이 2개 이상 있을 때는 각 통합을 최소 2회 수행할 수 있어야 가능합니다.

전체 알고리즘은 다음과 같습니다.

- 각 시간선을 목표 시각과의 대소 관계 값으로 변경해서 -1, 0, 1로 변환
- 연속된 동일한 시간선 제거
- 필수적으로 필요한 통합 계산 및 확인
- 가능할 경우 왼쪽 시간선들과 오른쪽 시간선들을 통합
- 제거된 동일한 시간선을 복원하며 남은 통합 수행

시간 복잡도는 큰 문제가 되지 않으나, 통합 과정을 구현 시 조금씩 비슷하면서도 다른 동작이 다수 필요해 고려할 점이 많습니다. 알고리즘이 n이 크나 작으나 대동소이하기 때문에, 조그마한 n에서 정상 동작하는지 확인하면 구현 실수를 줄일 수 있습니다.

I. 골드리치의 비밀 금고

`ad_hoc`, `sorting`, `constructive`

출제진 의도 – **Easy**

- 제출 142번, 정답 62팀 (정답률 43.662%)
- 처음 푼 팀: **Fox is cute** (Fox, is, cute), 6분
- 출제자: `wapas`
- 가장 짧은 정해: 453B^{C++}, 248B^{Python}

- 문제를 재정의하면 다음과 같습니다.
  - 음이 아닌 정수로 구성된 길이가 N인 수열 A가 주어진다.
  - 집합 $P = \{\text{mex}(A_i, \cdots, A_j) \mid 1 \le i \le j \le N\}$라 하자. 다음 값을 구하여라
    - 수열 A를 원하는 대로 재배열할 때 가능한 $\text{mex}(P)$ 값 중 최댓값

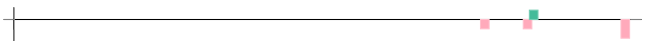
- 재배열했을 때 가능한 최댓값은 다음과 같이 구할 수 있습니다.
  - $\mathrm{mex}(A)$를 M이라 합니다. mex 정의상 $M \le N$를 만족합니다.
  - 부분 수열의 mex는 M을 넘지 않으므로 $\mathrm{mex}(P) \le M+1$입니다.
  - 따라서 수열 A는 $[0, 1, \cdots, M-2, M-1, \cdots]$ 형태로 정렬을 통해 항상 구성할 수 있으며, 이 경우 $\mathrm{mex}(P) = M+1$가 됩니다.
  - 단, 수열 A의 모든 원소가 0이라면 $\mathrm{mex}(P) = 0$이므로 이 경우는 예외 처리를 해주어야 합니다.

J. 격자 조각 자르기

`flow`, `grid_graph`

출제진 의도 – **Hard**

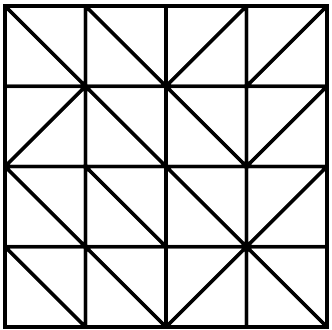
- 제출 5번, 정답 1팀 (정답률 20.000%)
- 처음 푼 팀: **BIGSHOT** (ITS, PS, TIME), 252분
- 출제자: `79brue`
- 가장 짧은 정해: 2276B[C++]



- 문제의 제한이 $N \leq 50$, $M \leq 50$으로 작습니다.
- 따라서 플로우를 이용한 풀이를 생각해볼 수 있습니다.
- 문제의 상황을 어떤 방식으로 플로우로 모델링할 수 있을까요?

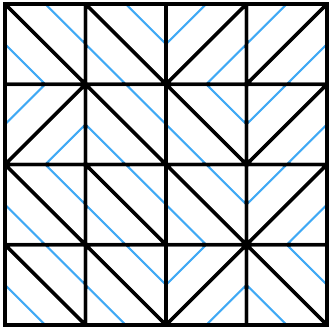
- 먼저 어떤 경우에 조각이 아름다운 조각이 되는지를 살펴봅시다.
- 조각의 경계로 가능한 것은 격자의 경계를 이루는 가로/세로 방향 선분, 그리고 각 격자를 자른 대각선입니다.
- 모든 조각은 필연적으로 대각선 방향 경계를 포함하게 됩니다.
- 따라서, 격자의 경계를 이루는 가로/세로 방향 선분을 경계로 포함하지 않고, 오직 대각선 방향 경계만 포함하는 조각이 아름다운 조각이 됩니다.
- 즉, 경계에 격자의 경계(가로선/세로선)를 포함하는 조각과 그렇지 않은 조각을 구분할 방법이 필요합니다.

- 모든 칸을 자르는 방법이 결정되었다고 가정해 봅시다.



- 다음 규칙대로 선을 그려 봅시다.
  - 각 격자칸의 상하좌우 변 중점에 총 4개의 점을 찍습니다.
  - 대각선을 기준으로 칸이 두 개의 영역으로 나뉘어집니다. 같은 영역에 있는 것끼리 연결해 봅시다.

- 결과는 다음과 같습니다.

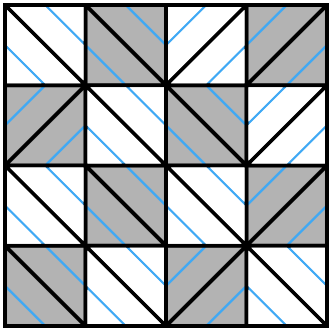


- 이 그림에서 몇 가지 사실을 관찰해 봅시다.
- 각 가로변과 세로변의 중점을 정점으로 보고, 각 격자칸에 그은 두 선분을 정점들을 잇는 간선이라고 생각해 봅시다.
- 두 행/열 사이의 가로변/세로변에 놓인 정점들의 차수는 항상 2이고, 격자 경계에 놓인 정점들의 차수는 1입니다.
- 따라서, 모든 연결 성분은 격자의 한 경계에서 시작해 다른 경계에서 끝나는 path, 또는 cycle을 이룹니다.

- 아름다운 조각은 대각선으로만 이루어져 있으므로, cycle을 이루어야 합니다.
- 입력으로 들어오는 가로변과 세로변은 위 그래프에서 정점에 해당합니다.
- 따라서 주어지는 정점들을 모두 cycle에 포함하도록 대각선 방향을 조절할 수 있는지를 찾는 문제가 됩니다.

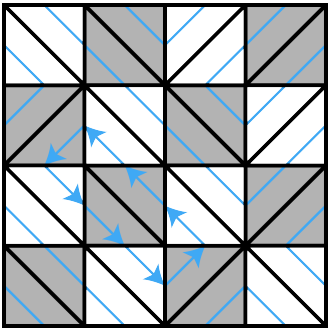
J. 격자 조각 자르기

- 플로우로 모델링을 하려면 방향을 부여해야 합니다.
- 사이클에 방향을 부여하는 방법을 찾아 봅시다.
- 격자판의 각 칸을 체스판처럼 색칠해 봅시다.



- 이제 흰 칸은 가로변에서 세로변으로, 검은 칸은 세로변에서 가로변으로 간선 방향을 설정합니다.
- 모든 경로와 사이클은 검은 칸과 흰 칸을 번갈아 지나게 되고, 이때 가로변과 세로변도 번갈아 통과합니다.
- 따라서 경로와 사이클 내의 방향은 항상 일관되게 정해집니다.

– 예시 그림의 사이클에 앞의 규칙대로 방향을 정하면 다음과 같습니다.

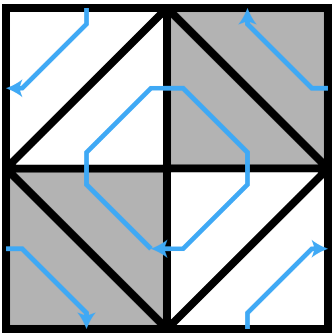


- 이전 그림과 같은 방식으로 모든 간선에 방향을 매깁니다.
- 흰 칸과 검은 칸은 각각 대각선의 방향에 따라 아래와 같은 두 가지 형태 중 하나가 됩니다.

- 이제 풀이의 개요를 설명합니다.
- 일단은 모든 칸을 자르는 방향이 사전에 정해져 있다고 생각합시다.
- 지금까지는 격자의 경계에 위치하지 않은 모든 가로변과 세로변을 모두 하나의 정점으로 생각했습니다.
- 그러나, 앞으로는 각각을 정점 분할하여, 두 개의 정점으로 생각합니다.

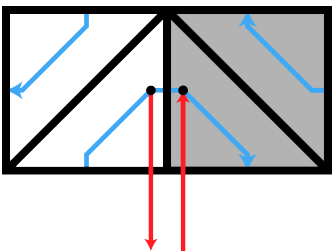
J. 격자 조각 자르기

- 다음 그림과 같이 정점 분할 상태에서 그래프의 정점들을 이어 줍니다.
- 편의상 연결된 성분은 화살표를 한 곳에만 표기하였습니다.



J. 격자 조각 자르기

- 모든 간선의 유량을 1로 설정합니다.
- 만약 어떤 선분이 아름다운 조각 안에 포함되어야 한다는 조건이 있다면, 해당 선분의 한쪽 정점에 1 만큼의 유량을 더해주고, 반대쪽 정점에 1 만큼의 유량을 내보내줍니다.
- 이는 서큘레이션으로 해석할 수 있고, 실제 구현에서는 가상의 소스 정점과 싱크 정점에 간선을 연결해 주는 것으로 생각할 수 있습니다.

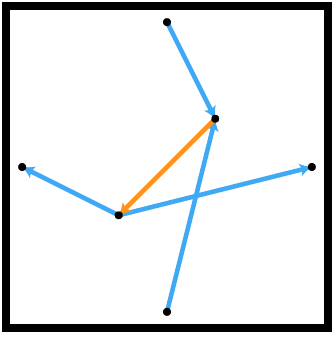


- 입력에 주어진 모든 변에 대해 이전 그림과 같이 간선을 연결했다면, 이제 최대 유량이 변의 개수와 같은지 확인합니다.
- 입력으로 주어진 격자에서 K개의 변이 모두 아름다운 조각에 속하는 것과 최대 유량이 K인 것이 동치임을 증명할 수 있습니다.
- 증명의 핵심은, 어떤 변 e에 대해, 해당 변으로 들어간 유량이 다시 싱크 노드로 흘러나올 때, 그 꼭짓점이 해당되는 변을 $f(e)$라고 정의하면, 그래프의 특징에 의해 f는 bijection이므로, f의 Functional Graph는 사이클로만 구성된다는 것입니다.

- 이제 아직 자를 방향을 모르는 격자칸이 있을 때를 생각해 봅시다.
- 일반성을 잃지 않고, 흰색 격자칸만 생각합니다.
- 흰색 격자칸의 경우 항상 가로변에서 세로변 방향으로 간선이 연결되어야 합니다.
- 이 칸에 플로우가 흐를 때, 저절로 격자를 자르는 방법을 찾을 수 있도록 그래프를 구성해 봅시다.

J. 격자 조각 자르기

- 다음 그림과 같이 간선을 연결합니다.
- 파란색 간선은 용량 1로, 주황색 간선은 용량 2로 설정합니다.



- 이렇게 하면 가로변에 해당하는 정점에서 나오는 유량만큼 세로변으로 보내주게 됩니다.
- 모든 간선의 용량이 정수인 경우, 최대 유량의 원리에 의해 모든 간선에 흐르는 유량이 정수인 최적해가 존재합니다.
- 따라서 가로변과 세로변을 하나씩 대응시켜 주는 것이 항상 가능합니다.
- 위와 같은 방식으로 전체 그래프를 구성하고, 최대 유량의 값을 K와 비교하는 방식으로 해가 존재하는지를 판별할 수 있습니다.

- 마지막으로 해를 찾는 방법에 대해 살펴봅시다.
- 입력에서 자를 방향이 주어지지 않은 격자판에 대해서만 생각합니다.
- 칸의 주황색 간선을 살펴봅니다.
- 흐르는 유량이 0이라면 이 간선은 문제 조건과 상관없으므로, 아무 방향으로나 잘라도 됩니다.
- 흐르는 유량이 1이라면, 단 하나의 가로변과 세로변만 사이클에 포함되므로, 해당 두 변이 연결되도록 격자칸을 자릅니다.
- 흐르는 유량이 2라면, 어떻게 잘라도 가로변과 세로변이 적절히 매칭되므로, 아무렇게나 잘라도 됩니다.

- 설명한 대로 그래프를 구성하고, 최대 유량 알고리즘을 이용해 문제를 해결할 수 있습니다.
- 그래프의 정점과 간선의 개수가 총 $\mathcal{O}(NM)$ 개이므로, 약 $\mathcal{O}\left((NM)^{2.5}\right)$ 정도의 시간에 해결하는 것으로 충분합니다.
- 디닉 알고리즘 등을 이용하여 구현하면 시간 제한보다 훨씬 빠른 시간 내에 문제를 해결할 수 있습니다.

K. 자동 광고 배치 시스템

`dp`, `segment_tree`

출제진 의도 – **Hard**

- 제출 87번, 정답 18팀 (정답률 20.690%)
- 처음 푼 팀: **팀명못정함** (imeimi2000, numbering, blackking26), 80분
- 출제자: `bubbler`
- 가장 짧은 정해: 2349 B^{C++}

- 먼저, 수열의 길이 n이 짝수이면 수열의 끝에 0을 하나 추가하여 길이를 홀수로 만들어 줍니다.
- n이 짝수일 경우 마지막에 남는 2개의 수를 반드시 선택해야 하는데, 3개의 수가 남아있을 경우 최소가 아닌 두 개의 수를 먼저 처리하는 것이 항상 유리하므로, 원래의 수열 끝에 0을 추가해도 답은 유지됩니다.
- 선택한 두 수의 max를 합한 결과를 최소화하는 것은 min을 합한 결과(마지막에 남는 수 제외)를 최대화하는 것과 동치입니다.

- 주어진 수열에서 k회의 동작을 하고 나면 첫 $2k+1$개의 수 중에서 하나만 남게 됨을 이용하여, $\mathrm{DP}[k][i]$를 **첫 k회의 동작 이후에 남아있는 첫 번째 원소가 i번째일 때 선택한 쌍들의 min의 합의 최댓값**으로 정의하면 $\mathcal{O}(n^2)$ DP를 얻을 수 있습니다.

$$\mathrm{DP}[0][1] = 0 \tag{1}$$

$$\mathrm{DP}[k][2k+1] = \max_{1 \le i \le 2k-1} \mathrm{DP}[k-1][i] + \min(v_i, v_{2k}) \tag{2}$$

$$\mathrm{DP}[k][2k] = \max_{1 \le i \le 2k-1} \mathrm{DP}[k-1][i] + \min(v_i, v_{2k+1}) \tag{3}$$

$$\mathrm{DP}[k][i] = \mathrm{DP}[k-1][i] + \min(v_{2k}, v_{2k+1}) \quad \text{where } i < 2k \tag{4}$$

- 여기서, 어떤 원소 v_i 가 마지막까지 사용되지 않고 남아있으려면 v_i 가 첫 원소가 된 시점부터 매번 그 원소를 제외한 둘을 선택해야 함을 알 수 있습니다.
- $i = 2k$ 또는 $i = 2k+1$ 일 때 $k+1$ 번째부터의 그러한 선택 방법은 유일하므로, 그 선택의 결과를 미리 반영해볼 수 있습니다. 이렇게 하면 행동 횟수 파라미터 k 를 떼고 다음과 같은 DP식을 얻을 수 있습니다.

$$\mathrm{DP}[1] = \min(v_2, v_3) + \min(v_4, v_5) + \cdots \tag{5}$$

$$\mathrm{DP}[2k] = \max_{1 \le i \le 2k-1} \mathrm{DP}[i] + \min(v_i, v_{2k+1}) - \min(v_{2k}, v_{2k+1}) \tag{6}$$

$$\mathrm{DP}[2k+1] = \max_{1 \le i \le 2k-1} \mathrm{DP}[i] + \min(v_i, v_{2k}) - \min(v_{2k}, v_{2k+1}) \tag{7}$$

- 이제 이를 세그먼트 트리를 이용하여 최적화할 수 있습니다.
- $\mathrm{DP}[2k]$ 와 $\mathrm{DP}[2k+1]$ 의 식이 거의 똑같으므로 전자의 경우만 생각해 봅시다.
- $\min(v_i, v_{2k+1})$ 의 값은 v_i 가 작으면 v_i, 아니면 v_{2k+1} 입니다.
- 따라서 $\max_{1 \le i \le 2k-1} \mathrm{DP}[i] + \min(v_i, v_{2k+1})$ 부분을 다음과 같이 표현할 수 있습니다.

$$\max\left(\begin{array}{c}\max\limits_{1 \le i \le 2k-1; v_i < v_{2k+1}} (\mathrm{DP}[i] + v_i) \\ \max\limits_{1 \le i \le 2k-1; v_i \ge v_{2k+1}} (\mathrm{DP}[i] + v_{2k+1})\end{array}\right) = \max\left(\begin{array}{c}\max\limits_{1 \le i \le 2k-1; v_i < v_{2k+1}} (\mathrm{DP}[i] + v_i) \\ v_{2k+1} + \max\limits_{1 \le i \le 2k-1; v_i \ge v_{2k+1}} (\mathrm{DP}[i])\end{array}\right)$$

- 이제 DP$[i] + v_i$를 저장하는 최댓값 세그먼트 트리와 DP$[i]$를 저장하는 최댓값 세그먼트 트리를 각각 만듭니다. 각 원소의 값은 0으로 초기화합니다.
- 그리고 v_i가 원본 수열을 오름차순으로 정렬했을 때 몇 번째로 오는지를 o_i라고 하면, DP$[i]$를 계산할 때마다 두 세그먼트 트리의 o_i번째에 각각 DP$[i] + v_i$와 DP$[i]$를 넣습니다.
- 이제 다음 DP$[2k]$의 값을 구하려면 첫 번째 트리에서는 o_{2k+1}의 왼쪽을, 두 번째 트리에서는 o_{2k+1}의 오른쪽을 쿼리하여 구할 수 있습니다.
- 최종적으로는 모든 DP값의 max를 출력하면 됩니다.

L. 망각의 최장 경로

`dp`, `disjoint_set`, `trees`

출제진 의도 – **Challenging**

- 제출 3번, 정답 0팀 (정답률 -%)
- 처음 푼 팀: –
- 출제자: queued_q
- 가장 짧은 정해: 2041B^{C++}, 1806B^{Python}

우선 문제를 단순화해서, 기억 정보가 주어지지 않을 때 최장 경로를 찾아 봅시다.

- 어떤 $s \to e$ 최장 경로가 지나는 정점 중 가장 번호가 큰 것을 m이라고 합시다.
- m을 기점으로 모든 기억이 사라지므로 $s \to m$과 $m \to e$ 경로는 독립입니다.
  - 즉, 각 경로가 도중에 어떤 나무를 지나든 서로에게 영향을 미치지 않습니다.
- 따라서 $s \to e$ 최장 경로는 $s \to m$ 최장 경로와 $m \to e$ 최장 경로를 이어 붙인 형태가 됩니다.

관찰: 최장 경로는 항상 연결 컴포넌트 내에서 가장 번호가 큰 정점을 지난다.

- 만약 $m < N$ 이라면,
  - $s \to m \to e$ 경로보다 긴 $s \to m \to N \to m \to e$ 경로를 만들 수 있습니다.
  - 나눠진 각 부분 경로는 서로 독립이므로 $m \to N \to m$ 길이만큼 더 길어집니다.
- 이는 $s \to e$ 최장 경로가 지나는 가장 번호가 큰 정점이 m 이라는 가정에 모순입니다. 따라서 $m = N$ 입니다.

이제 $s \to N$ 최장 경로를 구해야 합니다. ($N \to e$는 경로를 뒤집으면 동일)

- 정점 N을 지우면 그래프가 몇 개의 연결 컴포넌트로 나뉩니다.
- 정점 s가 속한 연결 컴포넌트를 C라고 합시다.
  - $s \to N$ 경로는 (N을 제외하면) C 안의 정점만을 지나며,
  - 앞선 관찰과 같은 논리로 C 안에서 가장 번호가 큰 정점 m을 지나야 합니다.
- 따라서 $s \to m \to N$으로 경로를 나눠서 생각할 수 있습니다.

- $m \to N$ 경로에서 N 직전에 방문한 정점은 무엇일까요?
  - N과 이웃한 C 안의 정점들 중, m과 이루는 최장 경로가 가장 긴 것이 됩니다.
- 만일 C 안의 모든 정점 x에 대해, C 안에서의 $x \to m$ 최장 경로를 안다면,
  - $s \to m$ 최장 경로는 이미 알고 있으며
  - N과 이웃한 C 안의 정점들을 확인해서 $m \to N$ 최장 경로를 구할 수 있으므로
  - $s \to m \to N$ 최장 경로를 구할 수 있습니다!

이제 남은 것은 C 안의 모든 $x \to m$ 최장 경로를 구하는 일입니다. 이는 같은 과정을 재귀적으로 적용해서 구할 수 있습니다.

정리하면 다음과 같습니다.

- $x < m$에 대해, m 이하의 정점만을 지나는 $x \to m$ 최장 경로의 길이를 $D(x, m)$ 이라고 합시다.
- 이 경로는 m 이하의 정점으로 이루어진, m을 포함하는 연결 컴포넌트 C 안에서만 이동합니다.
- $C - \{m\}$ 에서 x가 속한 연결 컴포넌트를 C', 그 안의 가장 번호가 큰 정점을 m' 이라고 합시다.
- 다음 DP 식을 만족합니다.
  - $x < m'$ 일 때, $D(x, m) = D(x, m') + D(m', m)$
  - $x = m'$ 일 때, $D(m', m) = \max_{y \in C' \cap \mathrm{adj}(m)} D(y, m') + 1$

이를 단순히 구현하면

- 부분 연결 컴포넌트가 최대 N 개 만들어지고,
- 각각이 최대 N 개의 정점과 이루는 최장 경로를 구해야 하므로
- 적어도 $O(N^2)$ 의 시간이 듭니다.
- 정점을 지웠을 때 연결 컴포넌트를 다시 만드는 과정에도 각각 $O(N)$ 의 시간이 듭니다.

최적화를 해 봅시다.

정점을 제거해 나가면서 연결 컴포넌트를 만드는 대신, 역순으로 정점을 붙여 나가면서 연결 컴포넌트를 만들어 봅시다.

- 1 번부터 N 번까지 순서대로 Union-find 자료구조를 통해 합쳐나갑니다.
- 연결 컴포넌트 집합의 루트는 가장 큰 번호의 정점으로 선택합니다.
- i 번 정점을 추가할 때, 이웃한 집합들의 루트와 i 번 정점을 잇는 그래프를 만든다면 트리 구조가 만들어집니다.
- 이를 Union-find tree라고 합시다.

- $D(x, m) = D(x, m') + D(m', m)$ 이므로,
  - m'과 m을 잇는 Union-find tree의 간선에 $D(m', m)$ 가중치를 부여하면
  - $D(x, m)$은 x에서 m까지 부모를 따라 올라가는 경로의 가중치 합과 같아집니다.
- 가중치 합은 Sparse table이나 HLD 등을 이용해서 빠르게 계산할 수 있지만, 가장 간단한 방법은 find 연산에서 경로를 압축할 때 압축된 경로의 가중치 합도 함께 관리하는 것입니다.
  - 현재 m까지 union을 수행했다면, x에 대한 압축된 경로 길이가 $D(x, m)$이 됩니다.

정리하면 다음과 같은 과정으로 DP 값을 구할 수 있습니다.

- 먼저 정점 m의 이웃한 정점들을 확인하며 $D(m', m)$을 계산합니다.
  - $D(m', m) = \max_{y \in C' \cap \mathrm{adj}(m)} D(y, m') + 1$ 인데,
  - 아직 union을 수행하기 전이므로 y에서 압축된 경로의 길이를 확인하여 $D(y, m')$를 구할 수 있습니다.
- 그 다음 m과 각각의 m'을 $D(m', m)$의 가중치로 이어 union 연산을 수행합니다.
- 이후 같은 컴포넌트 안의 임의의 x에서 압축된 경로 길이를 구한다면 $D(x, m)$과 같아집니다.

이제 기억 속 방문했던 나무들의 정보를 토대로 최장 경로를 구해 봅시다.

- a번 정점과 b번 정점이 기억 속 정보에서 연속해 있다고 가정합시다. $(a > b)$
- $a \to b$ 경로에는 b보다 큰 번호의 정점이 포함되면 안 됩니다.
- 따라서 b번 정점까지 union을 마친 상태에서 $D(x, b)$ 값들을 토대로 $a \to b$ 최장 경로를 구할 수 있습니다.
  - b가 속한 연결 컴포넌트를 $C(b)$라고 하면
  - $a \to b$ 최장 경로 길이는 $\max_{x \in C(b) \cap \mathrm{adj}(a)} D(x, b) + 1$와 같습니다.

- 기억 속 나무들을 역순으로 보면서 이 과정을 반복해 주면, 그 사이사이 최장 경로들을 전부 알 수 있습니다.
- 맨 처음에 출발한 정점 S가 기억 속에 없는 경우, $S \to a$ 꼴의 경로를 추가적으로 처리해 줍시다. 그러면 전체 최장 경로를 알 수 있습니다.
- 전체 시간복잡도는 $O(N \log N)$ 입니다.

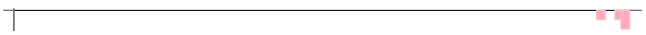
- TMI: 원래 Floyd-Warshall 알고리즘에서 min을 max로 바꾸고 약간의 예외 처리를 하면 풀리는 독특한 유형의 문제를 의도했습니다.
- 하지만 좀 더 고민해 보니 $O(N^3)$ 보다 빠른 $O(N \log N)$ 풀이를 발견하여 풀이의 방향을 바꿨습니다.
- 관심 있는 독자는 해당 풀이도 고민해 보시기 바랍니다.

M. Only Shallow

2-sat

출제진 의도 – **Hard**

- 제출 4번, 정답 0팀 (정답률 -%)
- 처음 푼 팀: –
- 출제자: lighton
- 가장 짧은 정해: 3086B^{C++}

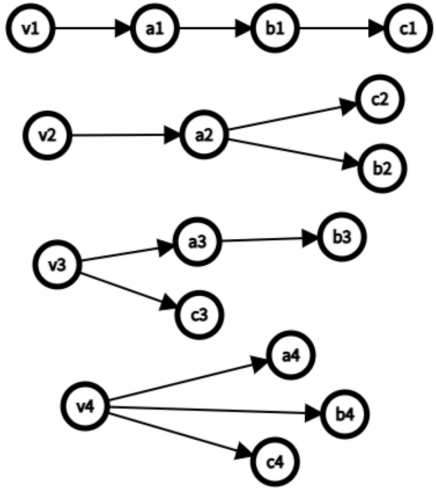


- Let $f(v)$ =(v에서 본인을 제외하고 도달 가능한 정점 수)
- 문제의 조건을 만족시키는 그래프, 즉 모든 v에 대해 $f(v) \leq 2$ 인 그래프의 특징에 대해 관찰해봅시다.

- 먼저, 모든 정점에 대해 out degree가 3 이상일 수 없습니다. 따라서 v의 차수가 3 이상이면 in degree를 적어도 하나 가지고 있습니다.
- 반면, 만약 v의 out degree가 2이면, in degree를 하나 가지고 있으므로, 해당 in degree에 해당하는 정점 u가 존재합니다. $f(v) \geq 2$이고 (out degree가 2), $u \to v$ 간선이 존재하므로, $f(u) \geq 3$이 되므로 조건을 만족시킬 수 없습니다.
- **조건 A)** 따라서 v의 차수가 3 이상이면 out degree는 0 또는 1이다.

- 정점 a, b, c에 대해 $a \to b \to c$ 경로가 존재한다고 해봅시다.
- a는 b, c 이외의 다른 정점과 연결될 수 없습니다. 만약 b, c가 아닌 다른 정점 v에 대해, $a \to v$가 존재하면, $f(a) \geq 3$(a는 v, b, c에 도달 가능), $v \to a$가 존재하면 $f(v) \geq 3$(v는 a, b, c에 도달 가능)이 되기 때문입니다.
- **조건 B)** 따라서, (무향 그래프로 생각할 때) 모든 경로 $a - b - c$, 중 a가 b, c가 아닌 다른 정점과 연결되있는 경로에 대해, $a \to b \to c$일 수는 없다.

- 놀랍게도, **조건 A)**와 **조건 B)**만 만족하면, 해당 그래프는 조건을 만족합니다.
- 이를 대우명제로 증명해봅시다. 어떤 v가 존재해, $f(v) \geq 3$ 이라고 해봅시다.
- 그러면, 방향 그래프는 반드시 다음과 같은 구조를 부분그래프로 가져야 합니다.



Enter Caption

- 2,4 번 예시는 **조건 B)**에 위배되고, 1,3 번 예시는 **조건 A)**에 위배됩니다.
- 따라서, 조건 A) 와 조건 B) 를 모두 만족하는 방향 그래프는 모든 v에 대해 $f(v) \leq 2$입니다.

- 각 간선의 방향을 x, $\neg x$ 로 생각해, 2-SAT 문제를 풀 수 있습니다.
- **조건 A)** 의 경우, 차수가 3 이상인 모든 정점 v에 대해, v와 연결된 모든 간선 중 모든 두 쌍에 대해 둘 다 out degree 방향이 아니라는 조건, 즉 둘 중 하나는 v를 향하는 방향이라는 조건을 설정할 수 있습니다.
- **조건 B)** 의 경우, 무향 그래프로 생각했을때 a가 b, c가 아닌 정점 최소 하나와 연결된 모든 경로 $a-b-c$에 대해 적어도 한 간선은 $b \to a$, 또는 $c \to b$ 방향이라는 조건을 설정할 수 있습니다.
- 시간복잡도는 $O(N^2)$ 입니다.
- 문제의 조건을 만족하는 그래프 구조의 분석과 적절한 컬러링과 Case Work 를 통하여 $O(N)$ 에 푸는 풀이도 존재합니다.