Good Bye, BOJ 2020!

Official Solutions

프리즈 시점까지

403명 참가

제출 **1,424**회

AC **700**회, WA **423**회, TLE **157**회, RE **86**회

총괄

✓	김준겸 ryute		고려대학교 컴퓨터학과

출제

✓	구재현 koosaga	MOLOCO	KAIST 전산학부
✓	권욱제 wookje	Riiid!	숭실대학교 컴퓨터학부
✓	권일우 rdd6584		인천대학교 컴퓨터공학부
✓	나정휘 jhnah917		선린인터넷고등학교 소프트웨어과
✓	이종서 leejseo	MOLOCO	KAIST 전산학부/수리과학과
✓	조영욱 urd05		대구과학고등학교
✓	한동규 queued_q		UNIST 전기전자컴퓨터공학부

검수

✓ 김동현	kdh9949	서울대학교 컴퓨터공학부
✓ 김준서	junseo	선린인터넷고등학교 소프트웨어과
✓ 김준원	junie	서울대학교 자유전공학부
✓ 노영훈	Diuven	MIT EECS
✓ 모현	ahgus89	서울과학고등학교
✓ 박선재	cs71107	서울대학교 컴퓨터공학부
✓ 서태수	cheetose	고려대학교 전기전자공학부

검수

✓	윤창기	TAMREF	MOLOCO	서울대학교 화학부
✓	이동관	cyberflower		고려대학교
✓	이상헌	evenharder		고려대학교
✓	이재웅	easrui		KAIST 새내기과정학부
✓	정재헌	gravekper		
✓	윤시우	cgiosy		도룡뇽고등학교

스트리밍

- ✓ 정재헌 `gravekper`

로고

- ✓ 권욱제 `wookje` Riiid! 숭실대학교 컴퓨터학부

조판

- ✓ 박수현 `shiftpsh` NEXON 서강대학교 컴퓨터공학과


STARTLINK


SOLVED.
AC

문제		의도한 난이도	출제자
A	끝말잇기	Easy	wookje
B	가장 가까운 세 사람의 심리적 거리	Easy	leejseo
C	양분	Medium	jhnah917
D	인간관계	Medium	leejseo
E	정점 간 통신 네트워크	Medium	urd05
F	상금 분배	Hard	rdd6584
G	최소 공통 조상과 쿼리	Hard	jhnah917
H	PCB 설계	Challenging	queued_q
I	남부순환로	Challenging	koosaga

A. 끝말잇기

ad-hoc
출제진 의도 – **Easy**

- 프리즈 전까지 제출 611번, 정답 403명 (정답률 68.085%)
- 처음 푼 사람 : **79brue**, 1분
- 출제자 : wookje

A. 끝말잇기

- 팰린드롬 문자열은 첫 문자와 끝 문자가 같으므로, 주어진 모든 문자열의 첫 문자와 마지막 문자가 같아야만 끝말잇기를 할 수 있습니다.
- 이를 이용해서, 여러가지 방식으로 문제를 해결할 수 있습니다.

- ✓ $S[i]$의 마지막 문자와 $S[i+1]$의 첫 문자가 하나라도 다른지 판정한다.
- ✓ `'a'-'z'`가 등장한 횟수를 세는 배열을 만들고, 주어진 모든 문자열들의 첫 문자와 마지막 문자를 카운팅 한다. 카운트가 0이 아닌 문자가 1개보다 많은지 판정한다.
- ✓ 각 문자열들의 첫 문자와 마지막 문자를 set에 insert하고, set의 size가 1인지 판정한다.
- ✓ 기타 등등 편한 방법으로 구현하면 됩니다.

B. 가장 가까운 세 사람의 심리적 거리

ad-hoc

출제진 의도 – **Easy**

- 프리즈 전까지 제출 347번, 정답 134명 (정답률 39.481%)
- 처음 푼 사람: **Green55**, 4분
- 출제자: `leejseo`
- 도움을 준 사람: `queued_q`

- 이 문제에 대해 가장 naive한 접근으로는 $\binom{N}{3}$ 가지의 모든 경우를 시도해보는 것이 있습니다.
- 하지만, 각 테스트 케이스당 $\mathcal{O}\left(N^3\right)$ 의 수행 시간을 가져 시간 초과를 받게 됩니다.

- ✓ 한 가지 관찰을 통해 수행시간을 크게 줄일 수 있습니다:
  - Observation: $33 (= 16 \times 2 + 1)$ 명 이상의 사람이 있다면, 그 중 어느 세 명은 MBTI 유형이 같다. (비둘기집의 원리)
- ✓ 위 관찰을 활용하면, 33 명 이상의 사람이 입력으로 주어지면 0 을 출력하고, 아닌 경우 $\binom{N}{3}$ 가지의 모든 경우를 시도해보는 방법을 생각할 수 있습니다.
- ✓ 이 경우, 하나의 테스트 케이스당 $\mathcal{O}\left(\max\{N, 32^3\}\right)$ 만큼의 수행 시간을 가지며, 이는 문제를 해결하기에 충분합니다.

C. 양분

`tree`, `graph`
출제진 의도 – **Medium**

- ✓ 프리즈 전까지 제출 226번, 정답 57명 (정답률 25.221%)
- ✓ 처음 푼 사람 : **gunwookim**, 9분
- ✓ 출제자 : jhnah917

- ✓ $M = N - 1$ 이면 트리입니다.
  - 트리에서 임의의 두 정점을 연결하는 경로는 1가지입니다.
- ✓ $M = N$ 이면 트리에 간선 하나를 추가한 그래프입니다.
  - 사이클 하나와 주변에 트리 형태의 가지 여러 개로 구성되어 있습니다.
  - 만약 동일한 트리 안에서 이동한다면 여전히 경로는 1가지입니다.
  - 그렇지 않은 경우, 사이클을 돌아가는 방법이 2가지이므로 경로가 2개 존재합니다.
- ✓ 사이클과 트리 부분을 잘 분리해야 합니다.

- ✓ 사이클과 트리 부분을 잘 분리해야 합니다.
  1. 그래프에서 Degree가 1인 정점을 계속 지워나가면, 마지막에 사이클 하나만 남게 됩니다.
  2. 위 과정에서 지워지는 정점과 **가장 가까운 사이클 위의 점**을 구하면 트리 부분을 구분할 수 있습니다.
- ✓ 쿼리로 주어진 두 정점이 같은 트리에 속한다면 1
- ✓ 그렇지 않다면 2를 출력하면 됩니다.

D. 친구관계

`graph`, `disjoint_set`, `dp`, `combinatorics`
출제진 의도 – **Medium**

- ✓ 프리즈 전까지 제출 106번, 정답 48명 (정답률 45.283%)
- ✓ 처음 푼 사람: **dlalswp25**, 8분
- ✓ 출제자: `leejseo`

- 친구관계를 그래프로 생각하면, 컴포넌트들의 집합을 분할하는 경우의 수를 구하는 문제가 됩니다.
- 컴포넌트의 개수는 Union-Find Tree 자료구조를 이용하여 효율적으로 관리하는 방법이 널리 알려져 있습니다.
- 고로, 컴포넌트가 n개 있다고 할 때, 크기 n인 집합을 분할하는 경우의 수를 구할 수 있다면, 문제를 해결할 수 있습니다.

- 점화식을 세우는 것을 시도해볼 수 있습니다. $dp[n]$ 을 크기 n 인 집합 (편의상 $\{1, 2, \cdots, n\}$ 이라 합시다.) 을 분할하는 경우의 수로 정의합시다.
- $\{1, 2, \cdots, n, n+1\}$ 을 $n+1$ 이 속하는 집합의 크기가 $k+1$ 이게 분할하는 경우의 수를 생각해보면 $\binom{n}{k} dp[n-k]$ 가 됨을 알 수 있습니다.
  - $\binom{n}{k}$: $n+1$ 과 같은 집합에 속할 원소들을 고르는 경우의 수
  - $dp[n-k]$: 나머지 $n-k$ 개의 원소를 분할하는 경우의 수

D. 친구관계

- 점화식을 세우는 것을 시도해볼 수 있습니다. $dp[n]$ 을 크기 n 인 집합 (편의상 $\{1, 2, \cdots, n\}$ 이라 합시다.) 을 분할하는 경우의 수로 정의합시다.
- $\{1, 2, \cdots, n, n+1\}$ 을 $n+1$ 이 속하는 집합의 크기가 $k+1$ 이게 분할하는 경우의 수를 생각해보면 $\binom{n}{k} dp[n-k]$ 가 됨을 알 수 있습니다.
- 따라서, $dp[n+1] = \sum_{k=0}^{n} \binom{n}{k} dp[n-k]$ 의 점화식을 얻게 됩니다.
- 팩토리얼과 팩토리얼 들의 곱셈 역원을 전처리 해놓든, $\binom{n}{r}$ 들을 DP로 전처리 해놓든 편한대로 하면, $dp[\cdot]$ 값들을 총 $\mathcal{O}\left(N^2\right)$ 시간에 계산할 수 있습니다.
- 이를 Union-Find Tree와 함께 활용하면 총 $\mathcal{O}\left(N^2 + M\alpha(N)\right)$ 정도 시간에 문제를 해결할 수 있습니다.

- ✓ 검수 과정에서 $dp[\cdot]$ 값의 수열이 Bell's number 라는 이름으로 알려져 있음이 확인되었으나, 문제가 적당히 Educational 하다는 의견 등이 있어 그대로 출제하였습니다.
- ✓ 참고로, $dp[\cdot]$ 수열의 생성함수는 $\exp(e^x - 1)$ 이 되며, 이를 이용하면 문제를 $\mathcal{O}\left((N+M)\log N\right)$ 정도에도 해결할 수 있습니다.

E. 정점 간 통신 네트워크

`tree`, `number_theory`, `dfs`
출제진 의도 – **Medium**

- ✓ 프리즈 전까지 제출 70번, 정답 20명 (정답률 28.571%)
- ✓ 처음 푼 사람 : **kcm1700**, 27분
- ✓ 출제자 : `urd0S`

- 어떤 정점에 대해서 그 정점의 조상들 중 어떤 주파수를 가지는 게 몇 개 있는지 저장해 두었다고 가정해봅시다.
- 그때 이 정점과 통신이 가능한 조상의 개수를 구하는 방법은 이 수의 약수와 배수를 모두 순회하면서 그 수를 주파수로 가지는 조상의 개수를 모두 더해주면 됩니다.
- 약수에 해당하는 수를 주파수로 가지는 조상의 개수를 모두 더하는 것은 $\mathcal{O}\left(\sqrt{MAX}\right)$ 시간에 가능하지만, 숫자가 꽤 작다면 배수에 해당하는 수를 주파수로 가지는 조상의 개수를 모두 더하는 것은 오랜 시간이 걸릴 수 있습니다.

- 그러므로 우리는 어떤 수의 배수를 주파수로 가지는 조상의 개수를 따로 저장하겠습니다.
- 이렇게 되면 어떤 수의 배수를 주파수로 가지는 조상의 개수는 $\mathcal{O}(1)$ 만에 구할 수 있습니다.
- 이제 어떤 정점의 조상들 중 어떤 주파수를 가지는 게 몇 개인지, 그리고 어떤 정점의 조상들 중 어떤 수의 배수인 주파수를 가지는 게 몇 개인지 저장만 되어있다면 $\mathcal{O}\left(\sqrt{MAX}\right)$ 라는 작은 시간복잡도에 이 정점의 조상 중에서 통신이 가능한 조상의 개수를 찾을 수 있습니다.

- 그럼 이제 이 배열은 어떻게 찾고 관리할 수 있을까요?
- 루트인 1번 정점부터 dfs를 합시다.
- 어떤 주파수를 가지는 게 몇 개인지 저장하는 배열, 어떤 수의 배수인 주파수를 가지는 게 몇 개인지 저장하는 배열은 전역에 놔둡니다.
- 먼저 앞서 말한 방식으로 현재 dfs 중인 정점에 대해서 그 정점과 통신 가능한 조상의 개수를 $\mathcal{O}\left(\sqrt{MAX}\right)$ 에 찾아줍시다.

- 그리고 이 정점의 주파수를 배열에 더해줍니다. 어떤 주파수를 가지는 게 몇 개인지 저장하는 배열에는 그 위치에 그냥 1을 더해주면 되고, 어떤 수의 배수인 주파수를 가지는 게 몇 개인지 저장하는 배열을 관리하기 위해서는 만약 이 정점의 주파수가 x라고 했을 때 x의 약수에 해당하는 모든 수의 위치에 1을 더해주면 됩니다.
- 그럼 이 더하는 과정은 $\mathcal{O}\left(\sqrt{MAX}\right)$ 에 수행됩니다.

- 이 배열이 현재 dfs 중인 정점의 조상들만을 관리하고 있게 하기 위해서, dfs로 자식들을 모두 순회하고 온 이후에는 아까 배열에 준 변화들을 롤백해줍니다. 단순하게 1을 더하던 것을 1을 빼는 것으로 바꿔주면 됩니다.
- 우리는 모든 정점에 대해서 자신과 통신 가능한 조상의 개수를 구했고, 이를 모두 더한 것이 답입니다.
- 이러면 $\mathcal{O}\left(N\sqrt{MAX}\right)$ 정도의 시간복잡도에 문제를 해결할 수 있습니다 .

F. 상금 분배

`sweeping`, `segment_tree`, `two_pointer`
출제진 의도 – **Hard**

- ✓ 프리즈 전까지 제출 22번, 정답 6명 (정답률 27.272%)
- ✓ 처음 푼 사람: **gs18115**, 28분
- ✓ 출제자: `rdd6584`

F. 상금 분배

- 상품권 배열 A를 내림차순으로 정렬합시다. 그리고 P_2와 P_3를 임의로 정해봅시다.
- 이때, P_4, P_5, P_6, P_7은 P_3와 연속한 위치의 상품권을 고르는 것을 시도하는 것이 최적입니다. 이 전략은 $P_2 + P_3 < P_4 + P_5 + P_6 + P_7$ 조건을 지키는 데에도, 총 상금을 최대화하는 데에도 항상 유리합니다.
- 즉, P_2의 인덱스를 i, P_3의 인덱스를 j라고 한다면 P_4의 인덱스는 $j+1$, ..., P_7은 $j+4$가 됩니다.
- 만약, A_i를 P_2로 정했다면 위 관찰을 통해서 $A_i < -A_j + A_{j+1} + A_{j+2} + A_{j+3} + A_{j+4}$를 만족하는 가장 큰 A_j를 P_3로 정하는 것이 최대임을 알 수 있습니다.
- 마찬가지로, 이 전략은 $P_1 < P_2 + P_3$ 조건을 지키는 데에도, 총 상금을 최대화하는 데에도 항상 유리합니다!! 와!!

- B_i를 $-A_i + A_{i+1} + A_{i+2} + A_{i+3} + A_{i+4}$로 하는 배열 B를 만들어봅시다.
- A 배열의 뒤부터 순서대로 훑으면서 P_2를 고정시키고 탐색한다면, 배열 B를 스택, 세그먼트 트리 등의 자료구조로 잘 관리하면서 P_3를 빠르게 찾을 수 있고, P_2와 P_3를 모두 결정했으므로 P_1도 빠르게 찾을 수 있습니다.

  ☆ ¤ **(별해)**

- 증명은 안됐지만, 문제의 특성 때문에 가장 큰 $N'(N' \geq 28 \times 7)$ 개 정도의 원소만 조사해도 되는 것으로 추측되며, 이에 따른 $\mathcal{O}\left(N'^2\right)$ 혹은 $\mathcal{O}\left(N'^3\right)$ 수준의 알고리즘을 작성해도 문제를 해결할 수 있습니다.

G. 최소 공통 조상과 쿼리

`tree`, `smaller_to_larger`, `offline_query`

출제진 의도 – **Hard**

- ✓ 프리즈 전까지 제출 28번, 정답 14명 (정답률 50.000%)
- ✓ 처음 푼 사람: **mhy908**, 34분
- ✓ 출제자: jhnah917

- ✓ 편의상 쿼리로 주어진 K 의 합을 X 로 정의합시다.
- ✓ 세 가지 풀이를 소개합니다.
  1. 오프라인 쿼리
  2. 트리 압축
  3. $\mathcal{O}\left((N+X)\sqrt{X}\right)$ 풀이

- ✓ 모든 쿼리를 입력받은 다음, 각 정점마다 쿼리 번호를 저장합시다.
- ✓ 아래에 있는 정점부터 위로 올라가면서 정점을 합치는데, 두 정점이 합쳐지는 지점이 정확히 **LCA**가 됩니다.
- ✓ i번 쿼리가 u번 정점에 a개, v번 정점에 b개 있다면 i번 쿼리의 정답은 $a \times b \times (LCA$의 레벨$)$ 만큼 증가합니다.
- ✓ 각 정점에 있는 쿼리 정보를 std::map으로 관리하고, 두 map을 합칠 때 크기가 작은 map에 있는 데이터를 큰 map으로 옮겨주면 $\mathcal{O}\left(N + X \log^2 Q\right)$에 문제를 해결할 수 있습니다.

- ✓ **쿼리로 주어진 정점**과 **그 정점들의 LCA**만 보면 됩니다. 필요한 정점으로만 구성된 트리를 만들어봅시다.
- ✓ 쿼리로 주어진 정점 중 dfs ordering 상에서 인접한 정점들의 LCA만 보면, 필요한 모든 정점을 볼 수 있습니다.
- ✓ 쿼리로 주어진 정점 K 개, dfs ordering 상에서 인접한 정점들의 LCA 최대 $K-1$ 개, 총 $\mathcal{O}(K)$ 개의 정점으로 구성된 트리가 만들어집니다.

- ✓ **이 정점이 LCA가 되는 경우의 수**를 알면 정답을 구할 수 있습니다.
- ✓ 간단한 트리 DP를 이용해 정점 개수에 비례한 시간에 원하는 값을 구할 수 있습니다.
- ✓ 트리 압축을 하는데 $\mathcal{O}(K \log N)$, 트리 DP를 하는데 $\mathcal{O}(K)$ 만큼 걸리므로 $\mathcal{O}(X \log N)$에 문제를 해결할 수 있습니다.

G. 최소 공통 조상과 쿼리 - $\mathcal{O}\left((N+X)\sqrt{X}\right)$ 풀이

- ✓ 시간 복잡도를 신경쓰지 않고, 이 문제의 답을 구할 수 있는 쉬운 풀이 2가지를 생각해봅시다.
  1. LCA를 $\mathcal{O}\left(K^2\right)$ 구하기
  2. 쿼리마다 $\mathcal{O}\left(N\right)$ DP 돌리기
- ✓ 첫 번째 풀이는 $\mathcal{O}\left(X^2 logN\right)$, 두 번째 풀이는 $\mathcal{O}\left(QN\right)$ 입니다.
- ✓ 첫 번째 풀이는 K, 두 번째 풀이는 Q가 커질수록 비효율적입니다.
- ✓ 두 풀이가 느려지는 조건이 서로 다르므로, 두 풀이를 잘 조합하면 뭔가 될 것 같습니다!

G. 최소 공통 조상과 쿼리 - $\mathcal{O}\left((N+X)\sqrt{X}\right)$ 풀이

- ✓ 적당한 상수 B를 잡아서, 각 쿼리마다
  1. $K \leq B$라면 LCA를 $\mathcal{O}\left(K^2\right)$번 구하고
  2. $K > B$라면 $\mathcal{O}\left(N\right)$ DP를 합시다.
- ✓ 시간 복잡도는 어떻게 될까요?
  - $K \leq B$라면 각 정점에 대해서 LCA를 최대 B번 구하게 됩니다. → $\mathcal{O}\left(XB\log N\right)$
  - $K > B$인 쿼리는 최대 $\frac{X}{B}$번만 주어집니다. → $\mathcal{O}\left(\frac{X}{B}N\right)$
  - 전체 시간 복잡도는 $\mathcal{O}\left(XB\log N + \frac{X}{B}N\right)$

G. 최소 공통 조상과 쿼리 - $\mathcal{O}\left((N+X)\sqrt{X}\right)$ 풀이

- ✓ $B=\sqrt{X}$ 라고 하면, 전체 시간 복잡도는 $\mathcal{O}\left(X\sqrt{X}\log N+N\sqrt{X}\right)$
- ✓ **시간 초과를 받게 됩니다!**
- ✓ Euler Tour와 Sparse Table를 이용해서 $\mathcal{O}\left(N\log N\right)$ 전처리를 통해 LCA를 $\mathcal{O}\left(1\right)$ 에 구하는 방법이 있습니다.
- ✓ $\mathcal{O}\left(1\right)$ LCA를 사용하면 시간 복잡도에서 $\log N$ 이 없어져서 $\mathcal{O}\left(X\sqrt{X}+N\sqrt{X}\right)$ = $\mathcal{O}\left((N+X)\sqrt{X}\right)$ 에 문제를 풀 수 있습니다!

H. PCB 설계

ad-hoc, graph

출제진 의도 – **Challenging**

- ✓ 프리즈 전까지 제출 5번, 정답 0명 (정답률 —%)
- ✓ 처음 푼 사람: **???**, ??? 분
- ✓ 출제자: queued_q

정의

- ✓ a 번 패드 둘을 잇는 도선을 a 번 도선이라고 부릅시다.
- ✓ 같은 번호가 붙은 패드 중 왼쪽에 있는 것을 왼쪽 패드, 오른쪽에 있는 것을 오른쪽 패드라고 부릅시다.

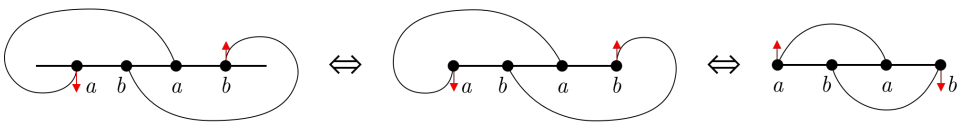
관찰

- **관찰 1:** 도선은 패드 배열 사이를 통과하여 지날 수 없다.
- **관찰 2:** 모든 도선이 서로 만나지 않음은, 각각의 도선 쌍이 서로 만나지 않음과 동치이다.
- **관찰 3:** 각 패드에서 도선은 위나 아래 중 한 방향으로만 뻗을 수 있다. (양 끝 패드는 좌우로도 뻗을 수 있지만, 풀이에는 영향이 없다.)
- 위 관찰들을 이용해서 패드마다 도선이 위로 뻗을지, 아래로 뻗을지를 결정하는 것이 풀이의 첫 번째 단계입니다.

각 패드의 도선 방향 결정

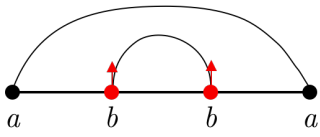
- ✓ a 번과 b 번 도선이 만나지 않도록 a 번 패드 둘과 b 번 패드 둘의 도선 방향을 결정하는 문제를 생각합시다.
- ✓ 네 페드를 제외한 나머지 패드는 무시하고, 네 패드의 배치에 따라 경우를 나눌 수 있습니다. 일반성을 잃지 않고 가장 왼쪽의 패드가 a 번 패드라고 하면 세 가지 경우가 존재합니다.
  - a, b, b, a
  - a, b, a, b
  - a, a, b, b

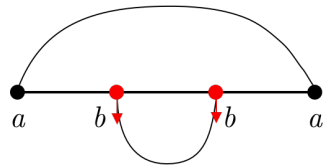
- 왼쪽 끝과 오른쪽 끝 패드의 도선 방향은 **도선의 배치 가능 여부에 영향이 없으므로** 사이에 위치한 두 패드의 도선 방향만 고려해 주면 됩니다.



경우 1: 패드의 배치가 a, b, b, a 인 경우

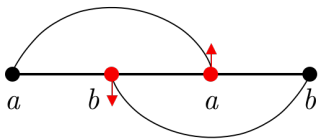
✓ 가운데 두 패드의 도선 방향이 같아야 합니다.

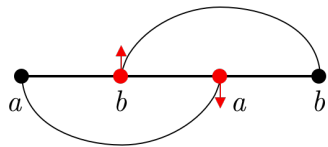




경우 2: 패드의 배치가 a, b, a, b인 경우

✓ 가운데 두 패드의 도선 방향이 달라야 합니다.





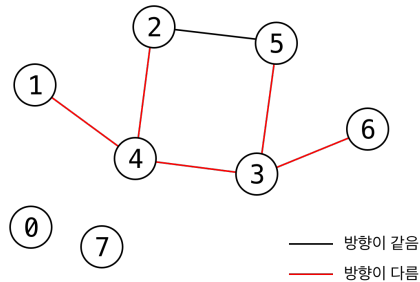
경우 3: 패드의 배치가 a, a, b, b인 경우

✓ 가운데 두 패드의 도선이 어떤 방향으로 뻗어 있어도 항상 도선이 만나지 않게 이을 수 있습니다.

H. PCB 설계

- ✓ 패드의 도선 방향 사이의 관계식을 찾았으니, 패드에 도선 방향을 배정할 차례입니다.
- ✓ 각각의 패드를 노드로, 관계식을 엣지로 갖는 그래프를 구성할 수 있습니다.

i	0	1	2	3	4	5	6	7
$A[i]$	1	2	3	4	1	3	2	4

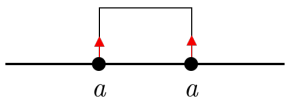


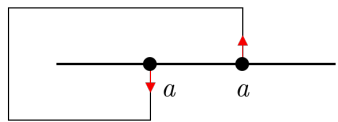
H. PCB 설계

- ✓ DFS로 그래프를 탐색하면서 패드에 도선의 방향을 배정하고, 모순이 있는지 확인합니다.
- ✓ 모순이 있다면 회로 설계가 불가능한 것입니다.
- ✓ 모순이 없다면 회로 설계가 가능함을 구성적으로 보일 수 있습니다.

도선 구성

- ✓ 각 도선을 다음 규칙에 따라 구성합니다.
  - 두 패드의 도선 방향이 같은 경우, 해당 방향으로 직접 잇는 도선을 그린다.
  - 두 패드의 도선 방향이 다른 경우, 왼쪽에서 만나는 도선을 그린다.
- ✓ 오른쪽에서 만나는 도선을 사용하지 않는 이유는, 설계된 회로에 그러한 도선이 있는 경우 항상 왼쪽으로 넘겨줄 수 있기 때문입니다.

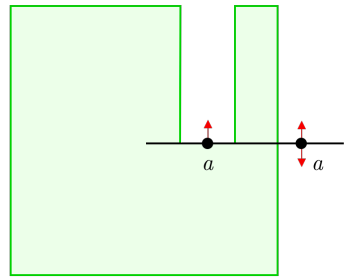




H. PCB 설계

- 패드를 왼쪽부터 훑으면서 오른쪽 패드를 만날 때마다, 대응되는 왼쪽 패드를 찾아 위 규칙대로 도선을 이어 줍니다.
- 이전에 그린 도선들과 겹치지 않도록 점점 바깥쪽으로 도선을 이어줘야 합니다.
- 이는 오른쪽 패드의 x 좌표가 증가한다는 사실을 이용해서, 오른쪽 패드의 x 좌표 크기만큼 도선을 바깥쪽으로 잇는 방식으로 구현이 가능합니다.
- 이것이 항상 이전에 그린 도선과 겹치지 않는다는 사실을 다음과 같이 보일 수 있습니다.
- 현재 보고 있는 패드가 오른쪽 a 번 패드이고, 왼쪽 a 번 패드의 도선 방향이 (일반성을 잃지 않고) 위쪽이라고 합시다.

✓ 현재까지 도선으로 채워질 수 있는 영역은 초록색 영역과 같습니다.



H. PCB 설계

- ✓ 초록색 영역의 전체적인 모양은 원점을 중심으로 갖는 정사각형이지만, 왼쪽 a 번 패드의 위쪽은 비어 있습니다. 왼쪽 a 번 패드의 위쪽으로는 도선이 지날 수 없기 때문입니다.
- ✓ 증명은 앞서 보여드린, 네 패드의 배치에 따른 도선의 방향을 생각해 보면 됩니다.
- ✓ 따라서 오른쪽 a 번 패드의 도선 방향이 어느 쪽이든 상관 없이, 초록색 영역의 바깥을 따라 도선을 연결하는 것이 가능합니다. 이를 구현하면 문제를 해결할 수 있습니다.

I. 남부순환로

`dp`, `persistent_data_structures`, `shortest_paths`

출제진 의도 – **Challenging**

- ✓ 프리즈 전까지 제출 6번, 정답 0명 (정답률 —%)
- ✓ 처음 푼 사람: **???**, ??? 분
- ✓ 출제자: koosaga

- 문제에서 원하는 올바른 설치의 조건은 다음과 같습니다.
  - 도로의 시점과 첫 가로등 사이에 최대 1개의 빈 블록이 존재할 수 있다.
  - 인접한 가로등 사이에 최대 2개의 빈 블록이 존재할 수 있다.
  - 마지막 가로등과 도로의 종점 사이에 최대 1개의 빈 블록이 존재할 수 있다.
- $K = 1$ 인 경우에는 최솟값만 찾으면 됩니다. 이 경우에는 DP로 문제를 해결할 수 있습니다.
- $DP[i][j] =$ (i 번 블록까지 설치 여부를 정했으며, 이 셀 뒤에 j 개의 빈 블록이 들어오는 것이 허용되었을 때의 최적해) 라고 상태를 정의합시다. $i + 1$ 번 셀에 블록을 설치하는 지 여부에 따라서 최대 2개의 새로운 상태로 전이됩니다.
- 초기 상태는 $DP[0][1]$ 이며 (빈 블록이 하나 가능) 최종 상태는 $DP[N][1], DP[N][2]$ 입니다 (빈 블록 설치 기회를 두번 사용하는 것은 금지).

- DP의 상태 전이는 DAG (Directed Acyclic Graph) 를 이룹니다. $K = 1$ 일 때 우리가 원하는 것은 이 그래프의 최단 경로입니다.
- 더 나아가서, 우리가 위에서 설계한 DP에서는 각 경로가 가로등을 올바르게 설치하는 경우의 수에 일대일 대응됨을 알 수 있습니다.
- 편의상, $(N, 2) \to (N, 1)$ 로 가는 가중치 0의 간선을 만들어서 시작점과 끝점을 고정시킵시다.
- 상태 $(0, 1)$ 에서 상태 $(N, 1)$ 로 가는 경로 중 K 번째 최단 경로를 찾으면 됩니다.
- 이는 Dijkstra's algorithm을 약간 변형하면 $\mathcal{O}(NK \log N)$ 에 할 수 있으나, 느립니다.

- S 를 시작점, E 를 끝점이라고 합시다.
- E 를 루트로 한 최단 경로 트리 T 를 잡습니다. 모든 간선은 E 를 방향으로 움직이며, 각 정점의 거리는 해당 정점에서 T 번 정점까지의 최단 경로의 길이입니다. 즉, 일반적으로 하는 S 에서 뻗어 나가는 최단 경로 트리의 반대라고 볼 수 있습니다.
- $S - E$ 경로를 단순하게 관리할 경우 길이가 매우 길어질 수 있습니다. 하지만 우리가 관심 있어하는 대부분의 경로는 T 의 간선들을 많이 벗어나지 않습니다.
- 이 점을 활용하여, 경로의 *Canonical Form* 을 찾을 것입니다.

- 트리에 없는 간선 $e \in G - T$ 에 대해서 이 간선을 대체 간선 으로 썼다는 것은, 최단 경로가 $start(e)$ 정점을 만나게 되면 e 를 따라 가고, $end(e)$ 에 도달함과 동시에 다시 최단 경로 트리를 통해서 T 로 간다는 것을 뜻합니다.
- 최단 경로는 대체 간선이 아예 없을 것입니다. 두 번째 최단 경로는 대체 간선을 정확히 하나만 사용합니다. (두 번째 최소 스패닝 트리랑 비슷합니다.)
- 이걸 Canonical Form으로 잡읍시다. $S - E$ 경로 P 상에서 T 에 존재하지 않는 간선을 등장 순서대로 $sidetracks(P) = [e_1, e_2, \ldots, e_k]$ 라고 정의합니다.

✓ 다음과 같은 성질이 성립합니다.

- $sidetracks(P)$ 가 정해지면 이에 따라서 P 를 유일하게 복구할 수 있다: S 에서 시작해서 e_1 의 시작점을 만날 때까지 움직이고, 움직이면 e_1 을 타고.. 를 반복한다. $sidetracks(P) = S$ 일 때, $path(S) = P$ 라고 하자.
- 모든 경로 P 에 대해서 $sidetracks(P)$ 가 정의된다. 하지만, $G - T$ 에 속하는 원소로 구성된 임의의 간선 수열 S 에 대해서 $path(S)$ 가 정의되지 않을 수 있다. 1번 항목의 알고리즘이 잘 작동해야만 정의된다.
- $delay(e_i)$ = $dist_T(end(e_i)) - dist_T(start(e_i)) + weight(e_i)$ 라고 정의하자. 이는 e_i 를 탔을 때 최단 경로 길이가 얼마나 증가하는 지를 뜻한다. 이 때 $delay(e_i) \geq 0$ 이며, P 의 길이는 (최단 경로 길이) + $\sum_{e \in sidetracks(P)} delay(e)$ 이다.

- 마지막 조건이 아주 이상적입니다. 각 에지가 최단 경로에 방해하는 기여도가 독립적이기 때문입니다. 이제 다음과 같은 탐색 트리를 정의합니다. 탐색 트리의 각 노드는 서로 다른 sidetrack에 대응됩니다.
  - 루트 노드는 빈 수열이다.
  - 어떠한 노드 $[e_1, e_2, \ldots, e_k]$ 에 대해서 해당 노드의 부모는 $[e_1, e_2, \ldots, e_{k-1}]$ 이다. 즉, 어떠한 노드의 자식은 sidetrack의 뒤에 올바른 대체 간선을 추가한 경우들이 된다.
  - 이 트리는 모든 가능한 sidetrack 수열을 열거하고 있다 (무한할 수 있음).
  - 이 트리는 Heap이다. (부모 노드의 값이 자식 노드의 값 이하이다.)
- 이 트리를 탐색할 수 있다면, 루트에서 타고 내려가면서 현재 보이는 자식 중 가장 가중치가 낮은 것을 k 번 반복해서 찾으면 됩니다.

- 이제 이 트리를 탐색해봅시다. 루트 노드는 빈 수열이니, Priority queue에 빈 수열을 넣습니다. 현재 보고 있는 노드의 sidetrack이 $[e_1, e_2, \ldots, e_k]$ 라고 하면, e_{k+1} 의 조건은, $start(e_{k+1})$ 이 $end(e_k)$ 의 조상이어야 한다는 조건입니다.
- 고로, $\mathcal{O}(KM \log M)$ 풀이는, $end(e_k)$ 에서 트리를 일일이 위로 타고 올라가면서, 각 노드에서 시작하는 대체 경로들을 모두 우선순위 큐에 넣는 것입니다.
- 이를 최적화하기 위해서는, 어떠한 노드 v 에 대해서, v 와 그 조상에서 시작하는 모든 대체 노드들을 $delay(e)$ 의 크기 순으로 잘 나열한 자료 구조가 필요합니다.
- 조상의 값을 빌려서 자식의 값을 형성하는 것은 어디서 많이 본 상황입니다. 일반적으로 Persistent한 자료구조를 통해서 해결할 수 있습니다.
- 여기서는 Persistent한 Heap이 필요합니다.

- ✓ Persistent하게 구현할 Heap은 Meldable heap을 사용합니다.
- ✓ 운영진은 Leftist Heap 혹은 Randomized Meldable Heap을 사용하였습니다.
- ✓ 사실 Meldable Heap을 쓸 필요는 없습니다. Meld 연산을 안 하기 때문입니다. 운영진 코드 중 그냥 일반 Heap을 Persistent하게 구현한 것도 있습니다. 다만 Meldable Heap의 구현이 더 간단하고, Persistent하게 구현하기에 알맞습니다.
- ✓ Persistent Heap을 구성하였다면, Heap의 루트 포인터를 넣고 우선순위 큐로 단순하게 탐색해주면 됩니다. 시간 복잡도는 $\mathcal{O}\left(N \log N + K \log K\right)$ 입니다.

- ✓ 이상의 내용은 *David Eppstein. Finding the k Shortest Paths (1998)* 에서 따왔습니다.
- ✓ DAG 형식으로 이야기 했지만, 사이클이 있어도 됩니다. 이 경우 단순 경로가 아닐 수 있습니다. 음수 간선이 있어도, 최단 경로 트리 계산을 Bellman Ford로 한 후 동일하게 진행하면 됩니다.
- ✓ DP 그래프는 KAIST 2018 가을 대회 *Fascination Street* 에서 사용한 것을 그대로 가져왔습니다. 문제 이름이 남부순환로인 이유입니다. 그래프의 특수한 구조를 활용한 Ad-hoc 한 풀이가 있을 거라는 생각도 했으나, 전 찾지 못했습니다. 찾는 것이 별 의미가 없을 지도...
- ✓ 대회가 끝나고 나서 공부해 보기에 좋은 내용이라고 생각하지만, 대회 중에 떠올리고 구현까지 하기에는 매우 복잡합니다. 위 논문이 유명하기도 하고, 몰라도 이런 알고리즘이 있지 않을까 상상해 볼 수는 있습니다. 좋은 코드 라이브러리를 가지고 있거나, 구글링으로 괜찮은 자료들을 많이 참고한다면, 아마 의도한 난이도에 비해 훨씬 쉽게 풀 수 있을 것이라고 생각합니다.