# 20.12.16 동아리 풀이

선린인터넷고등학교 소프트웨어과
30610 나정휘
https://JusticeHui.github.io

# 문제 목록

- A. 동작 그만. 밑장 빼기냐? - 2020 가톨릭대 H번
- B. 뒤집힌 계산기 - 2020 연세대 F번
- C. 클레어와 물약 - 2020 경북대 H번
- D. 카드 셔플 - 2020 경북대 J번
- E. 중2병 호반우 - 2020 경북대 K번
- F. 부스터 - 2020 카카오 코드 페스티벌 예선 D번
- G. 모래시계 2 - 2020 연세대 J번
- H. Salty Fish - PTZ Camp Summer 2019 Day1 A번

# 동작 그만. 밑장 빼기냐? (1)

- 밑장 빼기 이전: 홀수 번째 카드 가져 감
- 밑장 빼기 이후: 짝수 번째 카드 가져 감

- 밑장 빼기 위치에 따른 (N+1)가지 경우를 각각 확인하자.
- 누적합(Prefix Sum)을 이용하면 각각 O(1)에 확인할 수 있다.

- http://boj.kr/89714b75bc704a84824b567b1ad7420c

# 뒤집힌 계산기 (1)

- 연산자 하나씩 처리하면 된다.
- O(N)짜리 루틴을 4번 반복하므로 O(N)

- 단순 구현 문제
- 스택 쓰면 편하다.

- http://boj.kr/5ea2370d71de4a25b5ea04ceba085622

# 클레어와 물약 (1)

- 뭔가 위상 정렬 느낌이 난다.

- BFS를 위상 정렬 느낌으로 돌리면 된다.
  - 정점의 Degree가 아닌, 어떤 레시피를 조합하기 위해 완성시켜야 하는 다른 레시피의 개수를 관리하면 된다.

- http://boj.kr/97fc44a85cdf4ecba4866b96d5ca6098

# 카드 셔플 (1)

- 0-based로 생각하자.

- X-셔플을 하면 i번째에 있는 카드가 2i mod N으로 간다.
- Y-셔플을 하면 2i+1 mod N으로 간다.

- a에 있는 카드가 셔플을 K번해서 갈 수 있는 곳의 범위는
  - [a *  2^K, (a+1) * 2^K)이다.
  - K를 1부터 30까지만 확인해봐도 된다.

# 카드 셔플 (2)

- 필요한 셔플의 최소 횟수를 x라고 하자.
  - mod 연산을 없애면
  - 적당한 t에 대해 a에서 Nt+b로 갈 수 있다는 것을 의미한다.
  - 이때 적당한 t는 이분 탐색으로 찾을 수 있다.
- a*2^K와 Nt+b의 거리 (Nt+b)-(a*2^K)를 d라고 하자.
  - d를 이진법으로 나타냈을 때
  - 0과 1을 각각 X와 Y로 치환한 것이 셔플 순서가 된다.
  - 왜 그런지는 잘 생각해보자.
- http://boj.kr/b97c6dd443fd4eb598bd084101b1772a

# 중2병 호반우 (1)

- 제한을 보니 BitDP같다.
  - mn(i_bit, j_bit): 열/행 방향으로 레이저를 쏜 상태를 비트로 낸 것이 각각 i_bit, j_bit일 때 얻을 수 있는 최소 점수
  - mx(i_bit, j_bit): 최대 점수

- http://boj.kr/6d331551687c430ba37f48f6e8c50b44

# 부스터 (1)

- (Xi, Yi)에서 (Xj, Yj)로 이동한다고 하자.
  - |Xi - Xj|와 |Yi - Yj| 중 더 큰 쪽은 부스터로 이동하고
  - 더 작은 쪽은 걸어가는 것이 이득이다.

  - 걷게 되는 최소 거리는 min(|Xi - Xj|, |Yi - Yj|) 이다.

- Ai에서 Bi로 가는 경로 상에 있는 간선 중
  - 최댓값을 최소화 시켜서
  - 그 값이 Xi보다 작거나 같은지 확인하는 문제다.

# 부스터 (2)

- 간선이 너무 많다.
- 사실 MST에 속한 간선만 봐도 된다.
  - 왜 되는지는 생각해보자.
  - MST를 막 구하면 그것도 O(N^2 log N)이라 안 된다.

  - MST를 O(N^2)보다 빠르게 구하고
  - 트리에서 임의의 두 정점을 잇는 경로에서 최댓값을 구하면 된다.

# 부스터 (3)

- 가중치가 min(|Xi - Xj|, |Yi - Yj|)인 간선 하나 대신
  - 가중치가 |Xi - Xj|인 간선, |Yi - Yj|인 간선 두 개로 분리하자.
  - MST는 가중치가 더 작은 간선을 고르기 때문에 상관 없다.

- 가중치가 |Xi - Xj|인 간선은
  - 체크포인트를 X좌표 기준으로 정렬한 뒤
  - 인접한 것들만 이어주면 된다. -> 간선 N-1개
- 가중치가 |Yi - Yj|인 것도 똑같이 N-1개만 만들면 된다.

# 부스터 (4)

- 간선이 O(N)개이므로 O(N log N)만에 MST를 만들 수 있다.

- 트리의 경로 상에 있는 간선의 가중치 중 최대값을 뽑는 것은
  - LCA + Sparse Table이나
  - Segment Tree + Heavy Light Decomposition으로 할 수 있다.
  - 아래 코드는 후자로 구현했다.

- http://boj.kr/0f78b35d3aa64f828c062e2a8a128249

# 모래시계2 (1)

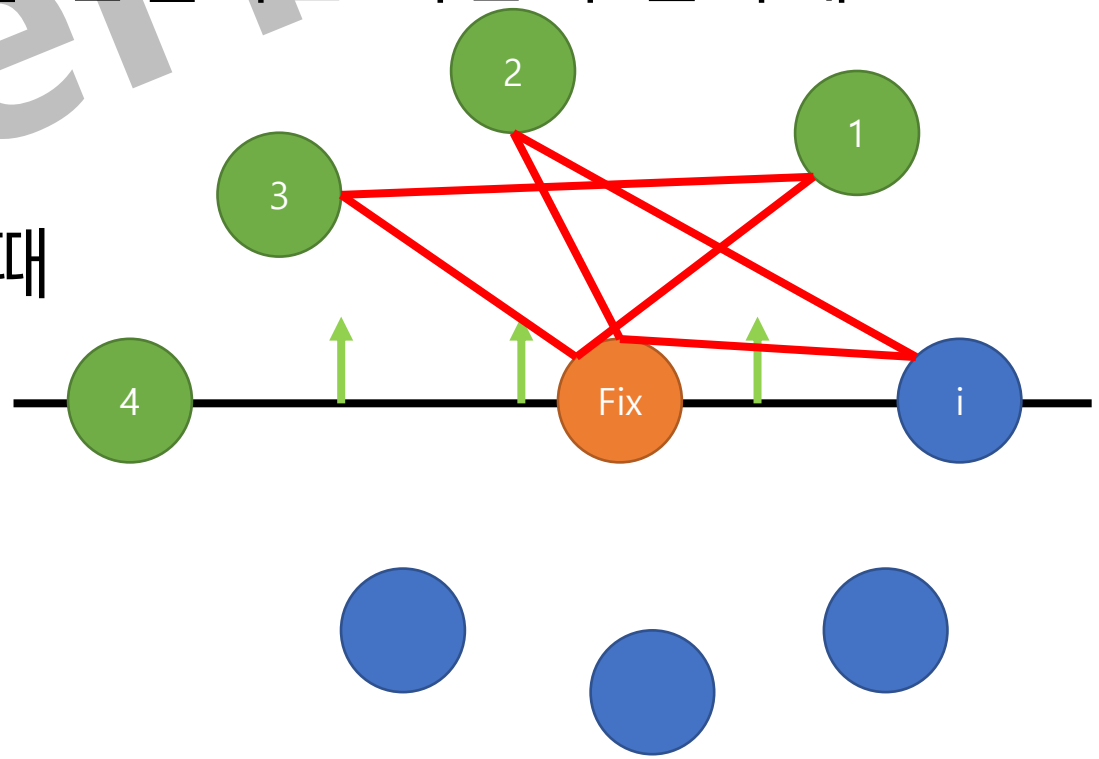
- 경우의 수를 직접 계산하는 것보다는
- 여사건을 구하는 것이 편하다.

- 가운데 들어갈 정점을 고정하자.
  - 나머지 정점 4개를 선택하는 경우의 수는 (n-1)C4이다.
  - 정점 4개를 선택해서 만들어지는 두 삼각형이 겹치는 경우의 수를 구해서 빼자.

# 모래시계2 (2)

- 가운데 고정한 정점을 기준으로 나머지 정점을 각도 순으로 정렬하자.
- 가운데 고정한 정점과 어떤 정점 i를 연결하는 직선의 왼쪽에 있는 정점의 개수를 K_i라고 하자.

- 가운데 정점, i, 나머지 3개로 만들 때
- 두 삼각형이 겹치는 경우의 수는
- sum( (K_i)C 3 )이다.



# 모래시계2 (3)

- 각도 정렬은 https://koosaga.com/97 에 잘 나와있다.
- K_i를 구하는 것은 투포인터를 이용하면 O(N)만에 구할 수 있다.
  - 인덱스 mod N하는 것이 귀찮으면 배열 2개 이어 붙이면 된다.

- 정점 하나를 고정했을 때 정렬 O(N log N) + 투포인터 O(N)
  - 전체 시간 복잡도는 O(N^2 log N)이다.

- http://boj.kr/83e1e2e60a504c259e204005a73faaca

# Salty Fish (1)

- 무려 **koosaga**님이 추천해준 문제!

- 아래 두 가지 행동을 해서 이득을 최대화하는 것이 목적이다.
  - (1) 카메라를 매수하고 정점을 먹는 것과
  - (2) 정점을 포기하는 것

- 일단 모든 정점을 먹는 것을 기본으로 하고
- 각 행동을 선택함으로써 손해를 보게 되는 비용을 최소화하자.

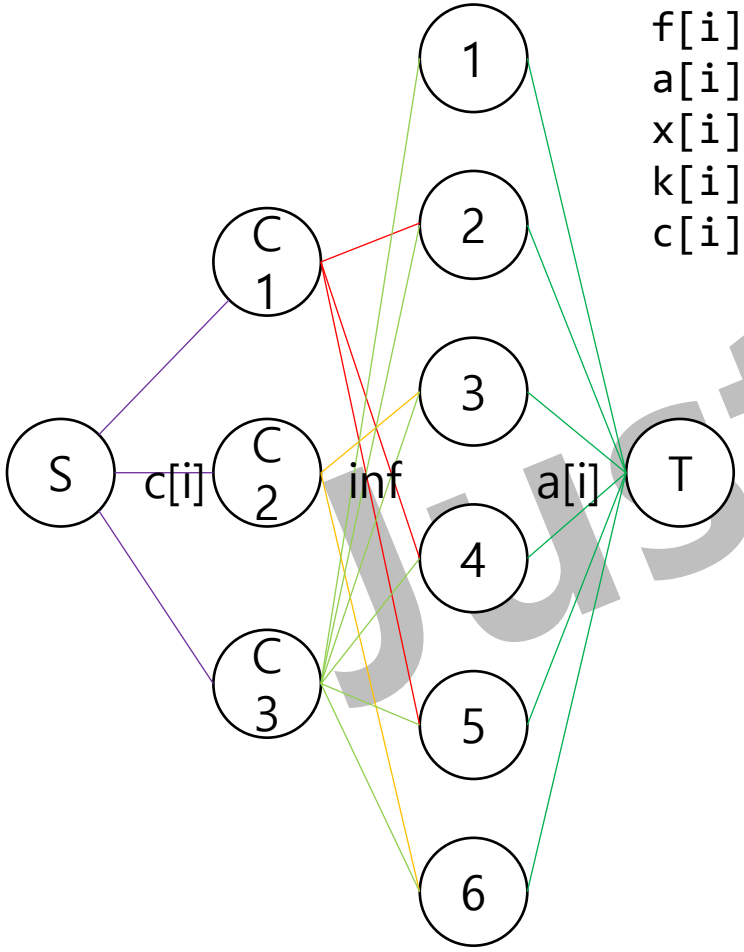
# Salty Fish (2)

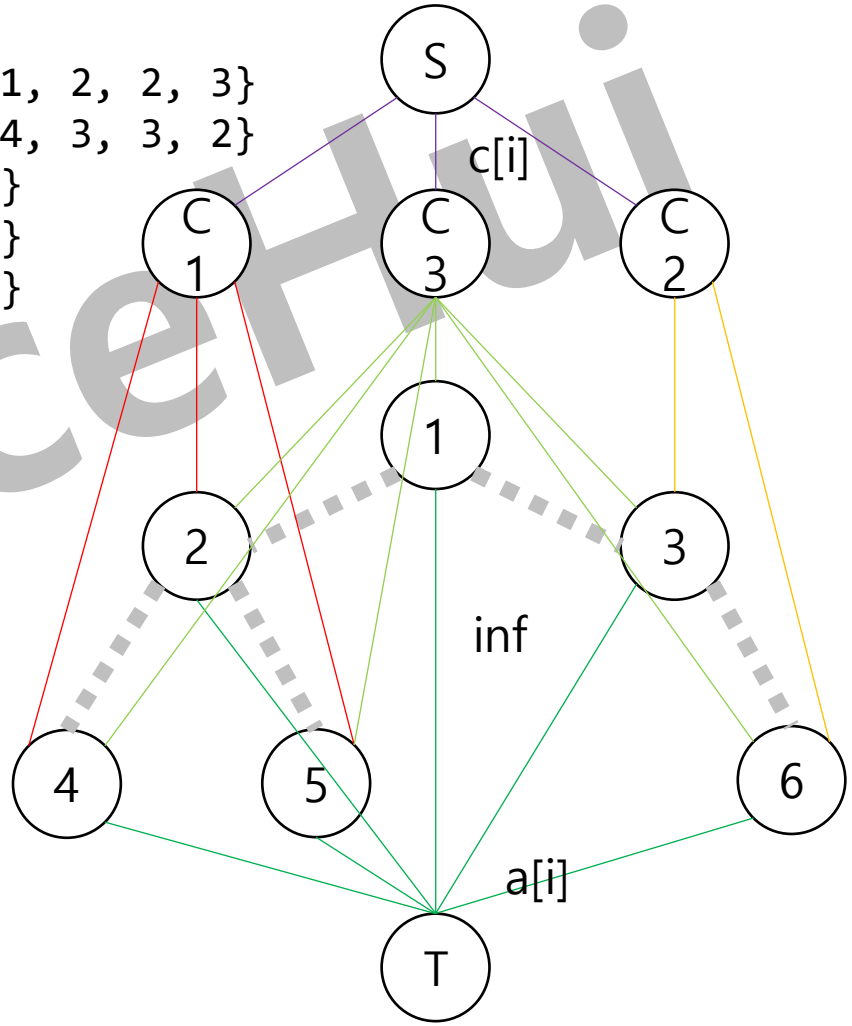
- 각 행동으로 인해 손해를 보는 비용
  - (1) 카메라를 매수하는 경우, 카메라의 가격인 c_i 손해
  - (2) 정점을 포기하는 경우, 해당 정점에 있는 사과의 개수인 a_i 손해

- 문제를 Min Cut으로 모델링할 수 있을 것 같다!
  - Source와 카메라를 가중치가 c_i인 간선
  - 카메라와 그 카메라가 담당하는 정점을 가중치가 inf인 간선
  - 정점과 Sink를 가중치가 a_i인 간선으로 이어주면 된다.
  - 이때 정답은 sum(a_i) - mincut이 된다.
  - maxflow = mincut이므로 정답은 sum(a_i) - maxflow이다.

# Salty Fish (3)

(예제)
N = 6, M = 3
f[i] = {-1, 1, 1, 2, 2, 3}
a[i] = { 2, 5, 4, 3, 3, 2}
x[i] = {2, 3, 1}
k[i] = {3, 1, 7}
c[i] = {1, 2, 4}



# Salty Fish (4)

- 그래프의 크기가 매우 크기 때문에($V \leq 60$만) 일반적인 플로우 알고리즘을 사용하면 안 된다.

- 기본적인 흐름은 Ford-Fulkerson과 유사하다.
  - 유량을 흘릴 수 있는 곳을 경로를 찾아서 흘린다.

# Salty Fish (5)

- D(v, d): v를 루트로 하는 서브 트리의 정점 중, **1번 정점과 d만큼 떨어져 있는 정점**에서 **Sink로 가는 간선**들의 잔여 유량의 합

- Tree DP처럼 DFS를 하면서 아래부터 처리하자.
  - 현재 정점 v에 있는 카메라는 깊이가 dep[v] + k_i인 정점까지 관리
  - 흘릴 수 있는 가장 깊은 곳부터 유량을 보내는 것이 이득이다.
    - D(v, *)를 std::map으로 관리하면서 prev(upper_bound)를 구해 유량을 보내면 된다.
  - 현재 정점과 부모 정점의 dp값을 합치는 것은 Small to Large로

# Salty Fish (6)

- O((N+M) log^2 (N+M)) 정도에 문제를 풀 수 있다.

- http://boj.kr/f5e6072984bb46288ecec9a8491be7d6