UCPC 2022

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2022

Finals

Official Solutions

본선 해설

전국 대학생 프로그래밍 대회 동아리 연합 · UCPC 2022 출제진

정현환 (LIBe)

문제		의도한 난이도	출제자
A	니은숲 예술가	**Hard**	functionx
B	NPU 최적화	**Challenging**	ho94949
C	라즈베리 파이	**Medium**	heeda0528
D	수열과 쿼리의 부분합의 합	**Medium**	hyperbolic
E	반도체 제작	**Challenging**	lky7674
F	대충 카드로 몬스터 잡는 게임	**Hard**	99asdfg, functionx
G	Traveling Junkman Problem	**Hard**	queued_q
H	특별상	**Easy**	kclee2172
I	사건의 지평선	**Hard**	jh05013
J	교집합 만들기	**Easy**	jh05013
K	전국 대학생 프로그래밍 대회 동아리 연합 토너먼트	**Medium**	doju
L	커넥티드 카 실험	**Easy**	ho94949, man_of_learning
M	×+ +×	**Challenging**	ho94949

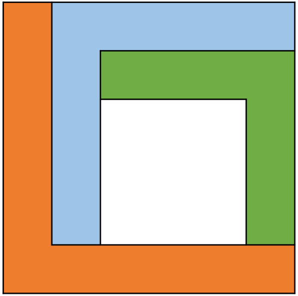
A. 니은숲 예술가

`dp, prefix_sum`

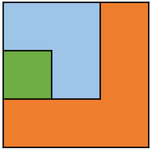
출제진 의도 – **Hard**

- 제출 36번, 정답 6팀 (정답률 16.67%)
- 처음 푼 팀: **UCPC의 최신 동향** (나폴리폴리, 고슬고슬비빈, 탕맛기픈), 155분
- 출제자: `functionx`

- 이 문제를 접근하는 방식은 두 가지 있습니다.

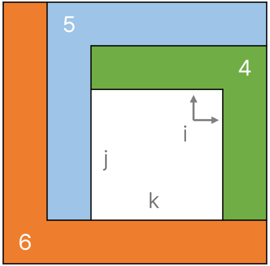


큰 조각부터 붙이기

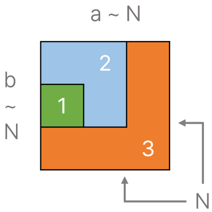


작은 조각부터 붙이기

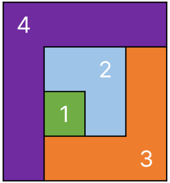
- 큰 조각부터 채우면 DP 식 자체는 매우 깔끔하게 나옵니다.
- 하지만 시간복잡도를 $\mathcal{O}\left(N^3\right)$ 미만으로 줄이기가 상당히 어렵습니다.



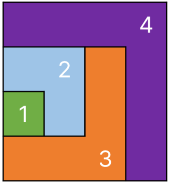
- 이번엔 작은 조각부터 채워봅시다.
- 모아진 조각을 유심히 살펴보면, 두 면에는 크기가 N인 조각만 있고, 나머지 두 면에는 크기가 $a \sim N$, $b \sim N$인 모든 조각들이 있습니다.
- 이런 성질을 이용해서 $D[N][a][b]$를 N개의 조각으로 해당 조건을 만족하도록 조형물을 만드는 경우의 수로 정의합시다.



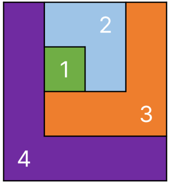
- 조각 $N(\geq 2)$ 개를 합친 상태에서 크기 $N+1$ 짜리 조각을 붙이면 아래와 같이 4가지 상태 전이가 발생합니다.



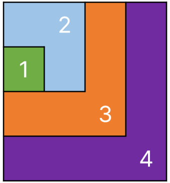
D[N][a][b] →
D[N+1][N][N]



D[N][a][b] →
D[N+1][N][b]



D[N][a][b] →
D[N+1][a][N]



D[N][a][b] →
D[N+1][a][b]

- 상태 전이를 역으로 계산해보면 아래와 같이 점화식을 정리할 수 있습니다.
- 단, $X[i]$는 i번 조각과 C_i가 같은 최대 $j(j < i$, 그런 j가 없으면 0)입니다.
  - $D[N+1][i][j] = D[N][i][j] (1 \le i, j < N)$
  - $D[N+1][i][N] = \sum_{X[N+1]<j<N} D[N][i][j] (1 \le i < N)$
  - $D[N+1][N][j] = \sum_{X[N+1]<i<N} D[N][i][j] (1 \le j < N)$
  - $D[N+1][N][N] = \sum_{X[N+1]<i,j<N} D[N][i][j]$

- $D[N][i][j]$를 한 번 구해놓으면 $D[M][i][j] (M > N)$은 $D[N][i][j]$와 같으므로 DP 결과를 저장할 때 N을 제외하고 i, j만 이용해서 $\mathcal{O}\left(N^2\right)$ 공간에 저장하면 됩니다.
- $D[N][i][j]$를 구하는 모든 점화식은 2차원 부분합 형태이므로 하나를 구하는 데 $\mathcal{O}(1)$ 시간만 있으면 됩니다.
- 따라서 $\mathcal{O}\left(N^2\right)$ 시간과 공간으로 문제를 해결할 수 있습니다.

B. NPU 최적화

`dp_tree, parsing, greedy`

출제진 의도 – **Challenging**

- 제출 15번, 정답 1팀 (정답률 6.67%)
- 처음 푼 팀: **McDic 없는 키위맛 파스타** (키위, 파, 스타), 297분
- 출제자: ho94949

- 입력을 파싱합시다.
- 하나의 수 다음에 나오는 글자가 '('라면 해당 수는 연산자, 아닌 경우 해당 수는 메모리 주소입니다.
- 이를 이용해 재귀적으로 파싱을 해서 parse tree를 만듭니다.
- 다음과 같은 용어를 정의합시다.
  - 연산자의 subtree: parse tree에서 연산자 노드의 subtree입니다.
  - 연산자의 계산 과정: 연산자의 subtree에 속한 데이터를 다루는 모든 명령어의 (순서 있는) 나열

B. NPU 최적화

- 연산자의 계산 과정은 >>로 시작해서, 최종적으로 해당하는 연산자 명령어 호출로 끝납니다.
  - 데이터를 불러 올 수 있는 유일한 방법은 >>이고, 해당 연산자 결과를 얻어낸 이후에는 추가 호출을 할 필요가 없습니다.
- <<는 연산자 실행 직후에 사용하는 것만 고려합니다. 또한 >>는 연산자 명령어 사용 직전에만 사용합니다.
  - 계산 과정의 연산자 순서를 위와 같이 바꿔서 올바른 계산 과정을 만들 수 있습니다.
- 이제 데이터의 생명을 얘기할 수 있습니다.
  - 데이터는 연산자의 계산 혹은 >> 명령어에 의해 태어나고, 다른 연산자의 입력으로 쓰일 때 죽습니다.

- 한 연산자의 두 입력에 대해 두 연산자의 계산 과정이 겹쳐있을 필요가 없습니다.
- 입력을 A, B 라고 하고, 연산자의 계산 과정에 A 의 subtree 계산이 가장 먼저 등장한다고 하면, A 의 계산과정을 모두 진행하고 B 의 계산과정을 모두 진행하는 것으로 바꿔도 됩니다.
  - A 가 최종 결과를 호스트에 저장하면, 새로운 B 의 계산 과정은 메모리 제약 없이 실행할 수 있습니다.
  - A 가 최종 결과를 호스트에 저장하지 않고 A 가 항상 메모리 하나 이상을 차지하는 경우에는, 새 계산 과정에서 B 는 메모리 하나를 사용할 수 없다는 제약만 있으므로 올바르게 실행됩니다.

- A의 계산 과정의 모든 데이터를 호스트에 저장하는 때가 존재하며, A의 최종과정을 호스트에 저장하지 않는 경우
  - A와 B의 계산 과정의 모든 데이터를 호스트에 저장하는 시점이 존재합니다.
  - 이 때까지의 A와 B의 프로그램을 각각 A_1, B_1이라고 하고, 나머지 부분에 대해서 항상 A가 메인 메모리를 한 공간 이상 차지하고 각각 A_2, B_2라고 합시다.
  - A_1, B_1의 실행은 메모리를 차지하지 않고 자유롭게 실행될 수 있으므로
  - B_1이 비어있는 경우 계산 과정을 A_1, A_2, B_2와 같이 바꿉시다.

- B_1 이 비어있지 않은 경우 B_1 의 마지막 저장 연산을 없애고 (B_1^-), A_2 의 실행 결과를 호스트에 저장하는 것으로 (A_2^+) 계산 과정을 결과를 바꿉시다. 그 후 A_1, A_2^+, B_1^-, B_2 순으로 재배치합시다.
  - 계산 과정의 명령어 수는 변하지 않습니다.
  - A_1, A_2^+, B_1^- 는 서로에 대해 메모리에 제약이 없는 상태에서 실행이 됩니다.
  - B_2 가 A 의 결과를 저장하느라 사용하지 못했던 공간을 B_1 의 결과를 저장하는 공간으로 대신 사용한 이후, B_2 의 실행 중 B_1 의 결과가 필요한 경우 불러오지 않고 사용할 수 있습니다.
- 각 연산자의 subtree의 계산을 독립적으로 수행할 수 있습니다.

- 데이터를 계산한 이후 호스트에 저장한 다음 불러오는 데이터의 <<, >> 명령은 스택처럼 일어납니다. 즉 A와 B의 데이터를 차례로 저장한 후 차례로 불러오는 경우가 없도록 할 수 있습니다.
  - 이는 각 연산자의 subtree의 계산을 독립적으로 수행할 수 있기 때문입니다.
- 연산자의 입력을 차례로 계산할 때, 결과를 저장하는 것을 먼저 계산합니다.
  - 그렇지 않은 경우 결과를 저장하지 않는 인자의 계산 결과가 메모리에 남아 있습니다.
  - 메모리 공간을 더 한정적으로 사용하는 것이 됩니다.

- 다음과 같이 연산자의 입력을 네 가지 종류로 나눕시다.
  - A 입력의 계산 과정에 반드시 호스트에 저장이 필요하고, 해당 연산 결과를 저장하는 연산
  - B 입력의 계산 과정에 반드시 호스트에 저장이 필요하고, 해당 연산 결과를 저장하지 않는 연산
  - C 입력의 계산 과정에 반드시 호스트에 저장이 필요하지 않고, 해당 연산 결과를 저장하는 연산
  - D 입력의 계산 과정에 반드시 호스트에 저장이 필요하지 않고, 해당 연산 결과를 저장하지 않는 연산
- B 종류의 연산은 최대 하나이고, B 와 C 연산은 한 종류만 존재합니다.
  - 둘 이상일 경우, 뒤에 계산하는 입력의 계산 과정에서 >>를 사용할 때 메인 메모리가 비어있지 않습니다.
- 연산자 내부 입력의 계산 순서는 $A \to B, C \to D$ 순으로 일어나게 됩니다.

- 결과를 저장하지 않고 연산자를 계산하기 위한 메모리의 최소크기 (B, D 타입)를 계산하는 법은 각 인자의 메모리 사용량이 $m_1 \geq m_2 \geq \cdots \geq m_l$ 일 때, $\max(m_1, m_2 + 1, \cdots, m_l + (l - 1), l + 1)$ 입니다.
  - i 번째 입력의 계산을 위해서는 미리 계산된 $i - 1$ 개의 입력이 이미 계산되어 있어, 메인 메모리에 들어있어야 합니다.
- 최대한 B 와 D 종류의 연산을 많이 만들어서 >>, << 연산자의 개수를 줄이려고 합니다.

- 우선 모든 입력 계산을 A와 C 종류의 연산으로 만들어서 메모리에 제약이 없도록 합니다.
- C 종류를 하나씩 D 종류로 옮깁니다. i번째 옮길 때는 메모리 크기가 $M-i$ 이하인지 확인합니다.
- 위 연산에 의해 C 종류를 모두 없앤 경우 A 인자를 최대 하나 B로 옮길 수 있습니다.
- 이때, 남은 D 종류 입력에 대해 메모리 크기가 1 줄어도 계산 과정이 올바른지 확인합니다.
- 동적계획법을 사용하면 특정 연산자의 결과를 저장하는 여부를 계산할 수 있습니다.

- 이제 데이터의 생명을 관리해주면서 메모리가 언제 태어나고 언제 죽는지에 따라서 실제로 계산과정을 만듭니다.
- 현재 사용할 수 있는 메모리 공간을 관리하면서, 메모리가 태어나면 사용이 불가능한 공간으로, 메모리가 죽으면 사용이 가능한 공간으로 만들어줍니다.
- `std::set` 등의 자료구조로 관리를 하면 $\mathcal{O}(\log M)$ 에 확인할 수 있습니다.
- 프로그램의 최대 길이는 $|V|$ 를 넘지 않기 때문에, 총 시간복잡도는 $\mathcal{O}((M + |V|) \log M$ 이 됩니다.
- 최종 계산 결과가 저장되는 메모리의 위치가 0 이 아닌데, 이 경우 메모리의 위치 번호를 적당히 재부여해줍니다.

C. 라즈베리파이

ad_hoc

출제진 의도 – **Medium**

- 제출 116번, 정답 14팀 (정답률 12.07%)
- 처음 푼 팀: 🐟🐚🦂 (실러캔스, 암모나이트, 삼엽충), 67분
- 출제자: heeda0528

문제를 다음과 같이 재정의할 수 있습니다.

- 아래의 연산을 최소한으로 반복해 모든 $i\,(1 \le i \le N)$에 대해 $a_i = b_i$가 성립하게 만들자.
  - $1 \le x \le M$인 정수 x를 임의로 정하고, $b_i = x$를 만족하는 모든 i에 대해 b_i를 $(b_i \bmod M) + 1$로 바꾼다.

- $b_i = b_j\ (i \neq j)$ 인 경우가 존재한다면 연산을 아무리 반복해도 $b_i = b_j$ 이므로 $a_i \neq a_j$ 임에 모순되어 불가능합니다.
- $b_i \neq b_j\ (1 \leq i < j \leq N)$ 를 가정합시다.
- 만약 연산을 반복하는 중에 $b_i = b_j\ (i \neq j)$ 인 경우가 생기면 마찬가지로 불가능합니다.
- 따라서 $b_i = b_j\ (i \neq j)$ 인 경우가 생기게 하는 연산은 할 수 없습니다.

- 시계 방향 순서대로 1에서 M까지의 번호가 붙은 원형 배열이 있고, 수 $i(1 \le i \le N)$가 b_i번 칸에 들어가 있는 상황을 생각해봅시다.
- 수가 들어 있는 칸에 연산을 한 번 행하면 그 수가 시계 방향으로 한 칸 이동하게 됩니다.
- 문제의 목표는 모든 i가 a_i번 칸에 들어가 있도록 하는 것입니다.

- 잘 생각해 보면, 연산 과정 중 원형 배열 내에서 i 들의 순서는 변하지 않습니다.
  - 연산을 진행해도 어떤 수의 다음 수가 변하지 않기 때문입니다.
- 따라서, b를 a와 같게 만들기 위해서는 배열의 처음 상황과 목표로 하는 상황에서 i 들의 순서가 동일해야 합니다.

- 불가능한 경우가 하나 더 남았습니다.
- 앞서 $b_i = b_j$ 인 경우가 생기면 불가능하다고 했으므로, 수를 이동시킬 때는 수가 이동하게 될 칸이 반드시 비어 있어야 합니다.
- 그런데 $M = N$ 이면 원형 배열의 모든 칸이 차 있어 어떤 연산도 할 수 없습니다.
- 따라서 $M = N$ 일 때 $a = b$ 이면 답은 0, 아니면 -1 입니다.

- 배열의 처음 상황과 목표로 하는 상황에서 i 들의 순서가 동일하고, 빈 칸도 하나 이상 있다면 b 를 주어진 연산을 통해 a 와 같게 만드는 것이 항상 가능합니다.
- 이제 이동 횟수의 최솟값을 구해봅시다.
- 먼저, (a_i, b_i) 쌍을 b_i 가 오름차순이 되도록 정렬합니다. 이러면 원형 배열에는 각 i 가 시계 방향을 따라 1 부터 순서대로 놓이게 됩니다.
- 이제 원형 배열을 일직선 배열이 반복되는 형태로 생각해봅시다.
  - 일직선 배열에서는 b_i 를 증가시킬 때 나머지 연산을 적용하지 않습니다.
- 일직선 배열에서 i 의 목적지는 원형 배열의 기준점$(M \to 1)$을 통과한 횟수에 따라 $a_i, a_i + M, a_i + 2M, \cdots$ 이 될 수 있습니다.

- i 의 목적지 위치인 a_i 가 만족해야하는 조건에 대해 살펴봅시다.
  - $a_1 < a_2 < \cdots < a_N$
    - i 의 상대적인 순서가 변하지 않아야 합니다.
  - $a_N < a_1 + M$
    - 원래 배열이 원형이기 때문에, a_N 이 (한 바퀴 회전한) a_1 이전에 있어야 합니다.
  - $b_i \le a_i$
    - b_i 는 항상 증가합니다.
- 위의 조건을 만족하면 적절한 순서로 i 를 a_i 칸까지 보낼 수 있습니다.
  - $a_i \neq b_i$ 이고, $b_i + 1$ 번 칸이 비어 있으면 b_i 를 1 증가시키면 됩니다.
  - b가 a와 일치하기 전까지는 $b_i + 1$ 번 칸이 비어 있는 어떤 b_i 가 항상 존재합니다.

- 만약 $a_i > a_{i+1}$ 인 $i(1 \le i < N)$ 가 있다면 a를 오름차순으로 만들기 위해 $a_{i+1}, \cdots, a_N$ 에 M을 더해줍니다.
- 만약 $b_i > a_i$ 인 $i(1 \le i \le N)$ 가 있다면 a_i 를 증가시켜 줘야 합니다.
  - 오름차순 조건과 $a_N < a_1 + M$ 이 유지되어야 하기 때문에, 모든 $a_1, \cdots, a_N$ 에 M을 더할 필요가 있습니다.
- 최솟값은 위 과정을 모두 진행한 이후에 $\sum_{i=1}^{N}(a_i - b_i)$ 입니다.
- 정렬 방법에 따라 전체 과정을 $\mathcal{O}(M)$ 또는 $\mathcal{O}\left(NlogN\right)$ 에 구현할 수 있습니다.

D. 수열과 쿼리의 부분합의 합

`math, segtree, lazyprop`

출제진 의도 – **Medium**

- 제출 64번, 정답 33팀 (정답률 51.56%)
- 처음 푼 팀: **BabyPenguin** (retro3014, gs18115, moonrabbit2), 35분
- 출제자: `hyperbolic`

이 문제는 크게 다음 2가지 방법으로 풀 수 있습니다.

- D를 줄여나가면서 스위핑
  (Lazy segment tree 필요, $\mathcal{O}(N + Q \log N)$)
- L을 늘려나가면서 스위핑
  (BST 필요, $\mathcal{O}(N + Q \log Q)$)

본 풀이에서는 첫번째 방법에 대해서 설명하겠습니다.

Q개의 쿼리 l_i, r_i, c_i가 주어질때, $f(U, D, L, R)$은 항상 적절한 c_i들의 합이 됩니다. 덧셈을 재정렬하면, 다음 식을 얻어낼 수 있습니다.

$$\sum_{U=1}^{Q} \sum_{D=U}^{Q} \sum_{L=1}^{N} \sum_{R=L}^{N} f(U, D, L, R)$$

$$= \sum_{U=1}^{Q} \sum_{D=U}^{Q} \sum_{L=1}^{N} \sum_{R=L}^{N} \text{적절한 } c_i \text{들의 합}$$

$$= \sum_{i=1}^{Q} c_i \times \text{적절한 } (U, D, L, R) \text{ 쌍의 개수}$$

c_i 에 대하여 적절한 (U, D, L, R) 쌍의 개수는 몇일까요?

- $l_i \le k \le r_i$ 인 k에 대하여, next$[i][k]$ = $i+1, i+2, \cdots, Q$번 쿼리중 쿼리의 구간이 k를 포함하면서 번호가 가장 작은 쿼리의 번호라고 합시다. (만약 그런 쿼리가 하나도 없다면 $Q+1$)
- 그렇다면, k번째에 위치한 c_i 값이 $f(U, D, L, R)$ 값에 영향을 주기 위해서는, U, D, L, R이 다음을 동시에 만족해야 합니다.
- $1 \le U \le i,\ i \le D <$ next$[i][k],\ 1 \le L \le k,\ k \le R \le N$

- 따라서, k가 고정되었을때 답에 영향을 주는 (U, D, L, R)의 개수는 $i(\text{next}[i][k] - i)k(N - k + 1)$ 입니다.
- $l_i \le k \le r_i$므로, c_i에 대하여 적절한 (U, D, L, R)쌍의 개수는 $\sum_{k=l_i}^{r_i} i(\text{next}[i][k] - i)k(N - k + 1)$ 입니다.
- 즉, 최종적으로 계산해야 하는 값은 $\sum_{i=1}^{Q} c_i \times \sum_{k=l_i}^{r_i} i(\text{next}[i][k] - i)k(N - k + 1)$ 입니다.

다시 식정리를 해봅시다. $s_k = k(N-k+1)$로 정의합니다.

$$\sum_{i=1}^{Q} c_i \times \sum_{k=l_i}^{r_i} i(\text{next}[i][k]-i)k(N-k+1)$$

$$= \sum_{i=1}^{Q} c_i \times \sum_{k=l_i}^{r_i} i\,\text{next}[i][k]\,s_k - i^2\,s_k$$

$$= \sum_{i=1}^{Q} i\,c_i \times \sum_{k=l_i}^{r_i} \text{next}[i][k]\,s_k - \sum_{i=1}^{Q} i^2\,c_i \sum_{k=l_i}^{r_i} s_k$$

- $\sum_{i=1}^{Q} i^2 c_i \sum_{k=l_i}^{r_i} s_k$는 s_k의 누적합을 미리 계산해 놓는 것으로 $\mathcal{O}(N+Q)$에 계산할 수 있습니다.
- 따라서, 모든 i에 대하여 $\sum_{k=l_i}^{r_i} \text{next}[i][k]\ s_k$를 빠르게 계산할 수 있다면 이 문제를 해결하는게 가능합니다.

next 배열은 다음 성질을 가지고 있습니다.

- 모든 k에 대하여 next$[Q][k] = Q+1$
- next$[i][1,2,\cdots,N]$ 에서 l_i 번째부터 r_i 번째까지의 값을 i 로 바꾸면 next$[i-1][1,2,\cdots,N]$ 이 됨

따라서 $\sum_{k=l_i}^{r_i}$ next$[i][k]$ s_k 는 range weight sum and update segment tree를 사용하면 $\mathcal{O}(\log N)$ 에 계산이 가능합니다. 그러므로 $\sum_{i=1}^{Q} i\ c_i \times \sum_{k=l_i}^{r_i}$ next$[i][k]$ s_k 는 i 를 Q 에서 1 까지 줄여나가면서 스위핑하면 $\mathcal{O}(Q\log N)$ 에 계산이 가능합니다.

E. 반도체제작

`linear_programming, duality, flow, circulation`

출제진 의도 – **Challenge**

- 제출 24번, 정답 1팀 (정답률 3.07%)
- 처음 푼 팀: **초비상!!** (코딩 말렸어요, 시간초과 떴어요, Ofast가 해줘야 해요), 290분
- 출제자: lky7674

- 주어진 문제를 다음과 같은 Linear Programming으로 표현할 수 있습니다.

$$
\begin{array}{lll}
\text{minimize} & \sum_{e \in E} (a_e p_e + b_e m_e) & \\
\text{subject to} & p_e + m_e \geq E_u - E_v & \text{for } e = (u, v) \in E \\
& E_1 = 1 & \\
& E_n = -1 & \\
& p_e \geq 0 & \text{for } e \in E \\
& m_e \leq 0 & \text{for } e \in E
\end{array}
$$

- 수식의 개수와 변수의 개수가 매우 많습니다. 때문에 LP Solver를 통해서 시간 내에 이 문제를 해결할 수 없습니다.
- 이 LP를 좀 더 알아보기 쉬운 형태의 LP로 변형해봅시다.

- 주어진 LP의 optimal solution value가 같은 Dual LP를 만들 수 있습니다.

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \le b \\ & x \ge 0 \end{aligned} \qquad\qquad \begin{aligned} \text{minimize} \quad & b^T y \\ \text{subject to} \quad & y \ge 0 \\ & A^T y \ge c \end{aligned}$$

- 위는 LP Dual의 기본형입니다. 다른 case들도 살펴봅시다.

– 다음과 같은 방법을 통해서 임의의 LP의 Dual LP를 구할 수 있습니다.

maximize LP	minimize LP
$Ax \le b$	$y \ge 0$
$Ax \ge b$	$y \le 0$
$Ax = b$	y : free
$x \ge 0$	$A^T y \ge 0$
$x \le 0$	$A^T y \le 0$
x : free	$A^T y = 0$

- Dual LP를 구하기 쉽게 LP를 보기 쉬운 형태로 변형하면 다음과 같습니다.

$$\begin{array}{lll} \text{minimize} & \sum_{e \in E} (a_e p_e + b_e m_e) & \\ \text{subject to} & p_e + m_e + E_v - E_u \geq 0 & \text{for } e = (u, v) \in E,\ u \neq 1,\ v \neq n \\ & p_e + m_e + E_v \geq 1 & \text{for } e = (1, v) \in E,\ v \neq n \\ & p_e + m_e - E_u \geq 1 & \text{for } e = (u, n) \in E,\ u \neq 1 \\ & p_e + m_e \geq 2 & \text{for } e = (1, n) \in E \end{array}$$

- Dual LP를 구하면 다음과 같습니다.

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(1,v)\in E} f_{1,v} + \sum_{(u,n)\in E} f_{u,n} & \\
\text{subject to} & b_e \le f_e \le a_e & \text{for } e \in E \\
& \displaystyle\sum_{(u,v)\in E} f_{u,v} = \sum_{(v,w)\in E} f_{v,w} & \text{for } v \neq 1,\ n \\
& f_e \ge 0 &
\end{array}
$$

- 해당 LP의 optimal value는 간선 e에 흐르는 flow의 lower bound가 b_e, upper bound가 a_e인 maximum flow value의 2배입니다. 이는 circulation을 통해서 해결할 수 있습니다.
- Push-Relabel 뿐만 아니라 Dinic, Edmond-Karp로도 시간 내에 이 문제를 해결할 수 있습니다.

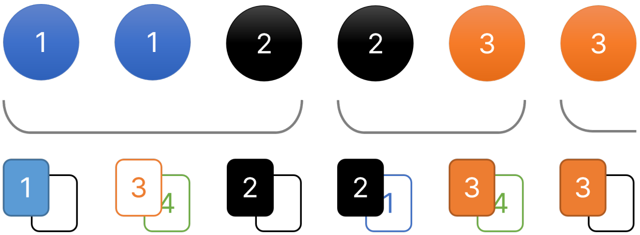
F. 대충 카드로 몬스터 잡는 게임

`dp`, `segtree`, `lazyprop`

출제진 의도 – **Hard**

- 제출 114번, 정답 22팀 (정답률 19.29%)
- 처음 푼 팀: **초비상!!** (코딩 말렸어요, 시간초과 떴어요, Ofast가 해줘야 해요), 63분
- 출제자: `99asdfg`, `functionx`

- 카드 덱 한 세트를 받아서 소모하는 턴을 한 그룹으로 묶어봅시다.
  - 하나의 그룹은 연속한 몇 개의 턴을 묶습니다.
  - 한 턴에는 카드를 최대 2장 쓸 수 있으므로 그룹의 크기는 $\lceil K/2 \rceil$ 이상이어야 합니다.



- 턴들을 그룹으로 묶었다면,
  - 그룹에서 특정 몬스터가 처음으로 등장했으면 그 몬스터를 무조건 잡을 수 있습니다.
  - 그룹에서 몬스터가 여러 번 등장했으면 하나 빼고는 잡을 수 없습니다.
  - 몬스터를 잡는 데 쓰는 카드를 제외한 나머지 카드는 아무 턴에 배정해도 됩니다.
- 따라서 한 그룹에서 잡을 수 있는 몬스터의 수 $\text{cost}(i, j)$는 i 이상 j 이하 턴에 등장하는 몬스터의 종류 수와 같습니다.

- 이제, 첫 i개 턴을 사용하고 **카드를 모두 쓸 때** 잡을 수 있는 최대 몬스터의 수를 $D[i]$로 정의합시다.
  - $D[0] = 0$이다.
  - $D[i] = \max_{j=1}^{i-\lceil K/2 \rceil} D[j] + \text{cost}(j+1, i)$이다.
- 별도 최적화 없이 식을 계산하면 $\mathcal{O}\left(N^2\right)$로 시간초과가 납니다.

- $V[j] = D[j] + \text{cost}(j+1, i)$ 를 자료구조로 관리해봅시다.
- i 가 늘어날 때,
  - $i+1$ 번째 턴에 몬스터가 없으면 V 값은 전부 1 증가합니다.
  - $i+1$ 번째 턴에 몬스터가 있고 그 몬스터가 이전에 p 번째 턴에 나왔다면 $p < j \le i$ 를 만족하는 모든 j 에 대하여 $V[j]$ 값이 1 증가합니다.

- 즉, 아래 세 쿼리를 처리하는 자료구조를 구현해서 V를 관리하면 됩니다.
  - i가 늘어날 때, p를 구한 다음 $p < j \leq i$ 범위의 $V[j]$ 값을 1 늘립니다.
  - $D[i]$를 구할 때, $D[i] = \max_{j=1}^{i-\lceil K/2 \rceil} V[j]$를 구합니다.
  - $D[i]$를 구했으면, $V[i] = D[i]$로 값을 할당합니다.
- 세그먼트 트리와 lazy propagation 등을 이용하여 자료구조를 구현하면 쿼리 당 $\mathcal{O}(logN)$ 시간으로 답을 구할 수 있습니다.

- 문제를 해결하는 데 필요한 시간복잡도는 $\mathcal{O}(NlogN)$ 입니다.
- 마지막 그룹은 크기가 $\lceil K/2 \rceil$ 을 넘을 필요가 없는데, 이 부분은 몬스터가 없는 $\lceil K/2 \rceil$ 개의 더미 턴을 추가하여 처리하면 됩니다.

별해 앞서 언급했던 $\mathrm{cost}(i, j)$ 가 사각부등식을 만족하므로 (monge), monotone queue optimization 등의 DP 최적화 알고리즘을 이용하는 방법도 있으나, 공간복잡도가 크므로 상수 최적화를 상당히 잘 해야 합니다.

G. Traveling Junkman Problem

`dp_bitfield`

출제진 의도 – **Hard**

- 제출 11번, 정답 5팀 (정답률 45.45%)
- 처음 푼 팀: **BabyPenguin** (retro3014, gs18115, moonrabbit2), 178분
- 출제자: queued_q

1. 느린 DP

- N개의 집 중에서 마지막으로 방문하는 몇 개의 집 집합을 S라고 합시다.
- S에서 처음으로 방문하는 집을 i라고 할 때, i번 집으로부터 어떤 물건들을 매입해야 하는지 생각해 보겠습니다.
- 나머지 집들 ($= S \setminus \{i\}$)을 방문하는 순서에 상관 없이, 어느 한 집이라도 관심 있는 물건들을 매입하면 최종적으로 $t_j - s_j$의 수익이 생기고, 어떤 집도 관심 없는 물건들은 수익을 얻을 수 없으므로 매입하지 않는 것이 좋습니다.
- 따라서 $S \setminus \{i\}$를 적절한 순서로 방문해서 최적의 수익을 얻는 방법을 안다면, S에서 i번 집을 처음 방문하는 경우에 얻을 수 있는 최적의 수익도 알 수 있습니다.

- 앞서 언급한 관계를 이용하면 다이나믹 프로그래밍으로 문제를 해결할 수 있습니다. 다음과 같이 DP 배열을 정의합시다.
  - $D(S, i)$: 방문할 집합 S에서 집 i를 가장 먼저 방문하는 경우에 얻을 수 있는 최대 수익
  - $E(S)$: 방문할 집합 S에서 얻을 수 있는 최대 수익
- 그러면 다음과 같은 관계식이 성립합니다. (여기서 $A(i)$는 집 i가 판매하는 물건 종류의 집합, $B(i)$는 집 i가 관심 있는 물건 종류의 집합을 나타내며, $B(S) = \bigcup_{i \in S} B(i)$ 입니다.)
  - $D(S \cup \{i\}, i) = E(S) + \sum_{j \in A(i) \cap B(S)} (t_j - s_j)$
  - $E(S) = \max_{i \in S}\{D(S, i)\}$
- 시간복잡도를 계산해 보면 $\mathcal{O}(N^2 M 2^N)$으로, 제한 시간 안에 통과하지 못합니다.

2. 전처리 (SOS DP) + DP

- 앞의 DP 식에서 $\sum_{j \in A(i) \cap B(S)} (t_j - s_j)$ 를 계산하는 과정이 $\mathcal{O}(NM)$ 씩 걸려서 병목이 되므로, 이를 미리 계산해 둡시다.
- 그러면 각각의 DP 식은 $\mathcal{O}(1)$ 에 계산할 수 있고, DP에 드는 시간복잡도가 $\mathcal{O}\left(N2^N\right)$ 으로 줄어듭니다.
- 이제 전처리를 얼마나 빠르게 하느냐가 관건입니다. 단순히 모든 집합 S 에 대해 합을 계산하는 방법은 여전히 $\mathcal{O}\left(NM2^N\right)$ 으로 느립니다.

- 먼저 i를 고정하고, i번 집이 판매하는 물건들만 모아 보겠습니다. 다른 종류의 물건들은 존재하지 않는다고 가정합시다.
- 계산해야 하는 것은 집합 S에서 i번 집을 처음으로 방문할 때, i번 집으로부터 매입할 물건들의 수익의 합입니다.
- 이는 (전체 물건들의 수익의 합)에서 (i번 집으로부터 매입하지 않을 물건들의 수익의 합)을 빼는 방법으로 구할 수 있습니다.
  - 전자는 간단하게 계산할 수 있으므로 후자에 집중합시다. 후자를 $F(S, i)$라고 하겠습니다.
  - $F(S, i)$는 집합 $S \setminus \{i\}$에서 어떠한 집도 관심 없는 물건들에 대한 $t_j - s_j$의 합과 같습니다.

- 각 물건에 대한 정보를 이진수로 나타내 봅시다. j 번째 비트는 j 번 집이 해당 물건에 관심을 가지는지 (1) 아닌지 (0) 를 나타냅니다.
  - 서로 다른 물건이 동일한 이진수 표현을 가질 수도 있는데, 이들은 수익의 크기를 합쳐서 한 종류의 물건으로 취급해도 됩니다.
- 이렇게 각 물건의 이진수 표현에 대해 수익의 크기를 미리 계산해 놓은 배열을 C 라고 합시다.
- $F(S, i)$ 는 $S \setminus \{i\}$ 의 각 집에 대응되는 비트들이 0인 이진수들에 대한 C 의 합과 같습니다.
- SOS DP (Zeta Transform) 을 이용하면, (고정된 i 를 포함하는) 모든 S 에 대해 $F(S, i)$ 를 $\mathcal{O}\left(N2^N\right)$ 시간에 계산할 수 있습니다.

- 모든 i 에 대해 이를 반복하면
  - 모든 배열 C 를 $\mathcal{O}(NM)$ 의 시간에 계산할 수 있습니다.
  - 모든 $F(S, i)$ 를 $\mathcal{O}\left(N^2 2^N\right)$ 의 시간에 계산할 수 있습니다.
- DP 계산에는 $\mathcal{O}\left(N 2^N\right)$ 의 시간이 듭니다.
- 따라서 전체 문제를 $\mathcal{O}\left(NM + N^2 2^N\right)$ 의 시간에 해결할 수 있습니다.

H. 특별상

`greedy, ad_hoc, sorting`

출제진 의도 – **Easy**

- 제출 79번, 정답 55팀 (정답률 70.89%)
- 처음 푼 팀: **민규솔로기투** (사민규, 오민규, 육민규), 7분
- 출제자: kclee2172

- 심판이 준 점수가 가장 큰 k명은 무조건 특별상이나 본상을 받게 됩니다.
- 즉, 해당 k명을 제외하고 주최자가 준 점수가 가장 큰 m명을 뽑는 것이 최적의 방식이 됩니다.

I. 사건의 지평선

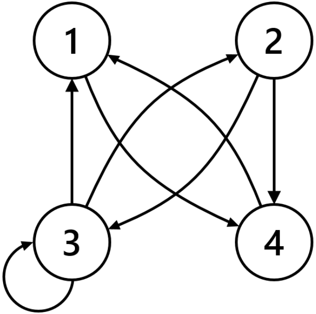
`graph_theory, sccc, dp, data_structures, segtree`

출제진 의도 – **Hard**

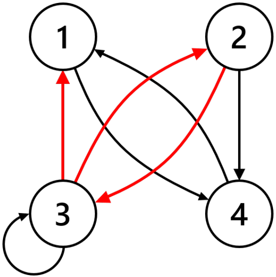
- 제출 149번, 정답 13팀 (정답률 8.73%)
- 처음 푼 팀: **우리가 우승할 수 있을 리 없잖아, 무리무리!** (U+203B, 무리가, 아니었다?!), 108분
- 출제자: `jh05013`

- 수열이 0과 1만으로 이루어져 있다고 해봅시다.
- i 번째 칸에 1이 있을 경우, 다음 순간에는 쿼리 구간이 i 를 포함하는 모든 칸에 1이 적힙니다. 이를 "전파"된다고 부릅시다.

- 각 칸에서 전파되는 칸들로 간선을 이어 봅시다.
- t초 후에 i번째 칸에 1이 적혀 있으려면, 초기에 1이 적힌 칸에서 출발하여 i번째 칸에 도달하는 길이 t의 경로가 있어야 합니다.

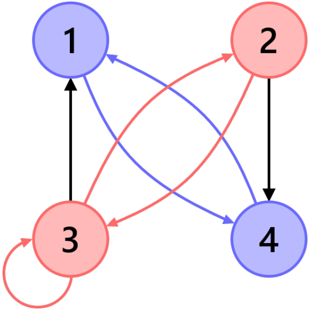


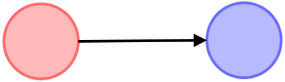
- i 번째 칸에 1이 무한히 많이 적히려면, 초기에 1이 적힌 칸에서 출발하여 사이클을 거치고 i 번째 칸에 도달하는 경로가 있어야 합니다.
- 사이클을 거치면 경로의 길이를 무한정 늘릴 수 있고, 안 거치면 늘릴 수 없기 때문입니다.



3에서 시작하여 사이클을 거치고 1에 도달하는 경로

- 사이클의 유무를 따질 때에는 SCC가 유용합니다. 그래프를 SCC로 묶어줍시다.
- 같은 SCC에 있는 정점들은 정답이 같으므로, 각 SCC마다 정답을 구하면 됩니다.



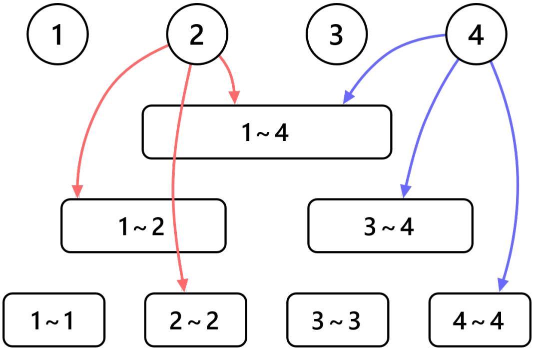


- 이제 SCC들을 위상 정렬한 순서로 DP를 돌려서 풀 수 있습니다.
- 한 SCC에 대해, 거기로 들어오는 간선이 있는 다른 SCC들을 봅시다.
- (1) 그 SCC들 중에 정답이 1인 것이 있거나 (2) 현재 SCC 자체에 사이클이 있으면서, 들어있는 정점 중 초기 값이 1인 것이 있으면, 현재 SCC의 정답은 1입니다. 그렇지 않으면 0입니다.

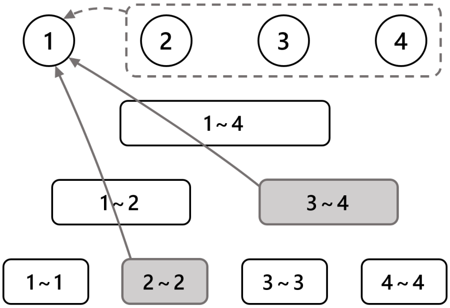
그런데 간선이 최대 N^2 개라서 너무 많습니다.

- 사실 지문에 있는 "수열과 쿼리"가 힌트입니다.
- 쿼리 구간들을 세그먼트 트리로 분할해 봅시다.

- 세그먼트 트리의 각 노드에 해당하는 정점을 하나씩 새로 만듭니다.
- 각 칸에서 그 칸을 포함하는 모든 세그먼트 트리 노드로 간선을 긋습니다.



- i 번째 쿼리 구간을 세그먼트 트리 노드로 분할하고, 그 노드들에서 i 로 간선을 긋습니다.
- 이제 간선이 $\mathcal{O}(N \log N)$ 개이기 때문에 시간 내에 돌아갑니다.



첫 번째 칸에 2~4번째 칸에 표시되어 있던 수 중 가장 큰 값이 표시될 때의 예시

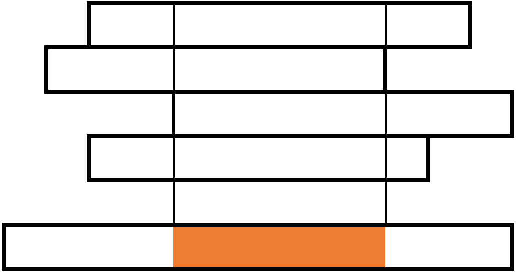
- 수열에 0과 1만이 아니라 다른 수가 있어도 풀이는 거의 같습니다.
- 한 SCC에 대해, 거기로 들어오는 간선이 있는 다른 SCC들을 봅시다.
- (1) 그 SCC들의 모든 정답을 모으고 (2) 현재 SCC 자체에 사이클이 있을 경우, 거기에 들어있는 모든 정점의 초기 값까지 모았을 때, 그중 최댓값이 현재 SCC의 정답입니다.

J. 교집합 만들기

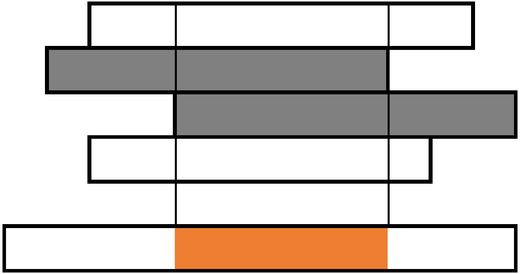
ad_hoc
출제진 의도 – **Easy**

- 제출 131번, 정답 55팀 (정답률 41.99%)
- 처음 푼 팀: 🐟🐚🦂 (실러캔스, 암모나이트, 삼엽충), 4분
- 출제자: `jh05013`

- $[l_i, r_i]$ 구간들의 교집합이 $[l, r]$일 필요충분조건은 다음과 같습니다.
  - 모든 구간 $[l_i, r_i]$에 대해 $l_i \le l$, $r_i \ge r$.
  - $l = l_i$인 구간이 존재함.
  - $r = r_i$인 구간이 존재함.



- 이중에서 $l = l_i$ 인 구간을 하나 잡아 A라고 하고, $r = r_i$ 인 구간을 하나 잡아 B라고 합시다.
- 그러면 A와 B의 교집합은 정확히 $[l, r]$ 입니다.
- 따라서 답이 존재할 경우 최대 2입니다.



- 답이 1인지 판별하려면, $[l, r]$ 이 정확히 N 개의 구간 중 하나인지 판별하면 됩니다.
  - C++, Python 등에서 기본적으로 제공하는 set을 사용하면 $\mathcal{O}(\log n)$ 또는 $\mathcal{O}(1)$ 시간에 알아낼 수 있습니다.
- 답이 2인지 판별하려면, (1) $l_i = l$, $r_i > r$ 인 구간과 (2) $l_i < l$, $r_i = r$ 인 구간이 존재하는지 판별하면 됩니다. 이를 효율적으로 하려면,
  - 모든 l 에 대해, $l_i = l$ 인 구간 중 r_i 가 가장 큰 것을 미리 기억해 둡니다. 물론 그러한 구간이 없으면 아무 것도 기억하지 않습니다.
  - 질의가 들어올 때, 주어진 l 에 대해 기억해 둔 구간을 보고 $r_i > r$ 인지 확인합니다.
  - 반대쪽도 마찬가지로, 모든 r 에 대해 $r_i = r$ 인 구간 중 l_i 가 가장 작은 것을 기억해 두면 됩니다.
- 위의 두 경우가 아니면 답은 -1입니다.

K. 전국 대학생 프로그래밍 대회 동아리 연합 토너먼트

`ad_hoc, case_work`

출제진 의도 – **Medium**

- 제출 265번, 정답 34팀 (정답률 13.21%)
- 처음 푼 팀: **우리가 우승할 수 있을 리 없잖아, 무리무리!** (U+203B, 무리가, 아니었다?!), 42분
- 출제자: doju

- 싱글 엘리미네이션 토너먼트의 경기 기록에서 누락된 경기 하나를 찾는 문제입니다.

싱글 엘리미네이션 토너먼트에서는 다양한 성질이 성립합니다.

- 우승자를 제외하면 모든 참가자가 정확히 한 번씩 패배합니다.
- 2^n 명이 참가한 대회에서 한 번도 승리하지 못한 참가자는 2^{n-1} 명, 단 한 번 승리한 참가자는 2^{n-2} 명, $\cdots$, $n-1$ 번 승리한 참가자는 한 명, 그리고 우승자가 한 명 있습니다.
- k 번 승리한 참가자는 한 번도 승리하지 못한 참가자, 단 한 번 승리한 참가자, $\cdots$, $k-1$ 번 승리한 참가자를 차례로 이깁니다.

이를 이용하면 한 경기가 누락되었을 때 누락된 경기에 대한 몇몇 단서를 쉽게 찾을 수 있습니다.

- 한 번도 패배하지 않은 참가자가 두 명 있습니다. 이 중 한 명은 우승자이고, 한 명은 누락된 경기에서 패배한 참가자입니다.
- 각 참가자가 몇 승씩 거뒀는지 세어 보면, 누락된 경기에서 승리한 참가자가 몇 승을 거뒀는지 알 수 있습니다.

하지만 그 뒤의 과정은 그렇게 간단하지는 않을 것입니다.

1. 패배하지 않은 두 명의 참가자의 승리 횟수를 세어 누가 우승자이고 누가 패배한 참가자인지 구분합니다.
2. 우승자의 경기 기록을 확인해 누락된 경기가 우승자와 패배한 참가자 간의 경기인지 판별합니다.
3. 그렇지 않다면, 누락된 경기에서 패배한 참가자를 이긴 참가자 A가 있고, 다시 이 참가자를 이긴 참가자 B가 있습니다.
4. 그 다음에는· · ·

이 과정을 더 설명하는 것은 그다지 재미있지 않을 것 같으니, 여기서는 다른 풀이를 소개합니다.

이 풀이는 대진표의 가장 아래 층에서 시작해 N을 하나씩 줄여 나가는 방향으로 접근합니다.

- 그리고 앞서 소개한 성질들을 거의 사용하지 않습니다.

먼저 대진표의 가장 아래 층에 누락된 경기가 없다고 가정합니다.

이때 다음과 같은 성질이 성립합니다.

- 0승 1패의 전적으로 대회를 마무리한 참가자(패자조)가 2^{N-1}명, 반대로 1승 이상을 거둔 참가자(승자조)가 똑같이 2^{N-1}명 있습니다.
- 두 그룹은 일대일 매칭됩니다.

따라서 패자조의 참가자들과 이 참가자들이 치른 2^{N-1}개 경기를 제외시키면 N이 1 줄어든 같은 문제가 됩니다.

이번에는 대진표의 가장 아래 층에 누락된 경기가 있다고 가정합니다.

이때는 다음과 같은 성질이 성립합니다.

- 누락된 경기에서 패배한 참가자는, 이 참가자가 참가한 유일한 경기가 누락되었으므로 기록에 등장하지 않습니다.
- 그럼 패자조는 한 명이 빠졌으니 $2^{N-1} - 1$ 명… 일까요?

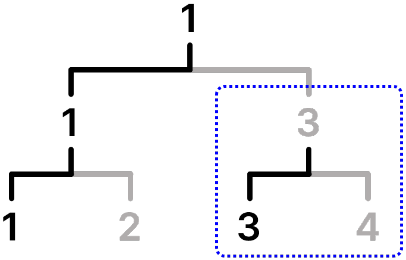
먼저 누락된 경기에서 승리한 참가자가 그 이후로도 추가 승리를 챙겼다고 가정합니다.

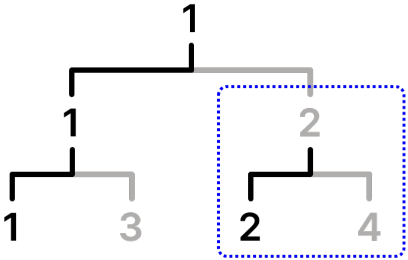
- 그러면 패자조는 $2^{N-1}-1$명, 승자조는 2^{N-1}명이 됩니다.
- 패자조의 참가자와 경기를 치르지 않은 승자조의 참가자가 유일하게 존재하며, 이 참가자가 누락된 경기의 승자가 됩니다.

재미있는 경우는 누락된 경기에서 승리한 참가자가 그 다음 경기에서 바로 패배한 경우입니다.

- 이 경우 이 참가자의 기록상 전적은 0승 1패가 되고, 패자조에 들어가게 됩니다.
- 그러면 패자조는 2^{N-1} 명, 승자조는 $2^{N-1} - 1$ 명이 됩니다.
- 패자조의 참가자들이 치른 경기를 모두 모아 보면, 어느 두 참가자는 같은 참가자에게 패배합니다.
  - 이 중 한 명은 실제로 한 번도 승리하지 못한 참가자이고, 한 명은 누락된 경기의 승자입니다.

이때 이 두 참가자 중 누구를 누락된 경기의 승자로 세워도 문제가 없습니다!





결과적으로, 문제의 답은 다음과 같은 세 가지 경우가 있습니다.

- 답이 한 가지인 경우
- 승리 횟수가 1 차이나는 두 참가자 간의 경기가 누락되어 답이 두 가지가 되는 경우
- 결승전이 누락되어 답이 두 가지가 되는 경우

시간복잡도 $\mathcal{O}\left(2^N\right)$ 또는 $\mathcal{O}\left(N \cdot 2^N\right)$ 으로 문제를 해결할 수 있습니다.

L. 커넥티드 카 실험

`two_pointer`

출제진 의도 – **Easy**

- 제출 128번, 정답 55팀 (정답률 42.97%)
- 처음 푼 팀: **숭실사이버의대를다니고나의성공시대시작됐다** (모, 새, 기), 12분
- 출제자: `ho94949`, `man_of_learning`

- S번 커넥티드 카에서 시작하여 BFS를 수행합니다. 이때 나이브하게 구현하면 $\mathcal{O}(N^2)$으로 시간초과가 납니다.
- 다음과 같은 관찰을 이용해 봅시다
  - 연결된 커넥티드 카들이 이동할 수 있는 구간은 언제나 연속 구간입니다.
- 새로운 커넥티드 카를 연결한 후 이동할 수 있는 구간이 변했다면, 늘어난 구간에 x_i가 포함되는 커넥티드 카만을 추가합니다.
  - 이는 구간의 양 끝을 가리키는 포인터 등을 이용하여 구현할 수 있습니다.
- 이로서 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.

M. ×+ +×

`fft, linearity_of_expectation, divide_and_conquer, combinatorics`

출제진 의도 – **Challenging**

- 제출 6번, 정답 1팀 (정답률 16.67%)
- 처음 푼 팀: **우리가 우승할 수 있을 리 없잖아, 무리무리!** (U+203B, 무리가, 아니었다?!), 286분
- 출제자: ho94949

- 칠판에 써있는 수를 $v_1, \cdots, v_N$ 이라고 합시다.
- k번 연산 이후 v_i 들의 곱이 더해질 확률을 계산할 수 있다고 합시다.
- 기댓값의 선형성에 의해 각 수에 대해서 더해질 확률을 모두 곱해 더해주면 전체 기댓값이 됩니다.
- 수는 모두 같은 확률로 선택되기 때문에, 확률은 곱해진 개수에만 영향을 받습니다.
- 같은 개수의 수를 곱한 것의 합 $S_x = \sum_{T \subset [N], |T| = x} \left(\prod_{t \in T} v_t \right)$ 을 유용하게 쓸 수 있습니다.
- 이는 $\prod_{i=1}^{n} (1 + v_i x) = \sum_{i=1}^{N} S_i x^i$ 을 이용해 계산합니다.
- 좌변은 FFT와 분할정복을 이용하여 $\mathcal{O}(N \log^2 N)$ 에 계산 가능합니다.

1. ×+

- $k = (x-1) + i$번 연산 이후에, $v_1 v_2 \cdots v_x$가 존재할 확률을 계산합시다. ($i \geq 0$)
- 수를 $v_1, v_2, \cdots, v_x$와 $v_{x+1}, v_{x+2}, \cdots, v_n$의 두 집합으로 나눕시다.
  - 쌍을 고를 수 있는 전체 경우의 수를 우선 구합시다.
    - ▶ $n(n-1) \times (n-1)(n-2) \times \cdots \times (n-k+1)(n-k) = \dfrac{n!(n-1)!}{(n-k)!(n-k-1)!}$
  - $x-1$개의 쌍이 첫째 집합에서, i개의 쌍이 둘째 집합에서 선택되어야 합니다
    - ▶ $\dfrac{k!}{(x-1)!i!} = \dfrac{k!}{(x-1)!(k-x+1)!}$

1. ×+

- $k = (x-1) + i$ 번 연산 이후에, $v_1 v_2 \cdots v_x$ 가 존재할 확률을 계산합시다. ($i \geq 0$)
- 수를 $v_1, v_2, \cdots, v_x$ 와 $v_{x+1}, v_{x+2}, \cdots, v_n$ 의 두 집합으로 나눕시다.
  - 첫째 집합에서 고르는 쌍은 크기가 $x, x-1, \cdots, 2$ 인 집합에서 $x-1$ 개의 쌍을 각각 골라야 합니다.
    - $x(x-1) \times (x-1)(x-2) \times \cdots \times 2 \cdot 1 = (x)!(x-1)!$
  - 둘째 집합에서 고르는 쌍은 크기가 $n-x, n-x-1, \cdots, n-x-(i-1)$ 인 집합에서 i 개의 쌍을 각각 골라야 합니다.
    - $(n-x)(n-x-1) \times (n-x-1)(n-x-2) \times \cdots \times (n-x-(i-1)) \cdot (n-x-1-(i-1)) = \dfrac{(n-x)!(n-x-1)!}{(n-x-i)!(n-x-1-i)!} = \dfrac{(n-x)!(n-x-1)!}{(n-k-1)!(n-k-2)!}$

1. ×+

- $k = (x-1)+i$ 번 연산 이후에, $v_1 v_2 \cdots v_x$ 가 존재할 확률을 계산합시다. $(i \geq 0)$
- 확률을 정리하면 $\dfrac{k!x!(n-x)!(n-x-1)!(n-k)(n-k-1)}{(k-x+1)!n!(n-1)!}$ 이 나옵니다.
- 왠지 모르게 식을 x에 관한 식, k에 관한 식과 $k-x$에 관한 식의 곱 형태로 표현하고 싶습니다.
- 정리하면 $\dfrac{x!(n-x)!(n-x-1)!}{n!(n-1)!} \times k!(n-k)(n-k-1) \times \dfrac{1}{(k-x+1)!}$ 이 됩니다.
- 우리가 결국 k번 연산 후에 구하려는 값은 $x = 1, \cdots, k+1$에 대해 수와 확률의 곱을 모두 합한 값입니다.
- $A_k = k!(n-k)(n-k-1)) \times \sum_{i=1}^{k+1} \left[\left(\dfrac{x!(n-x)!(n-x-1)!}{n!(n-1)!} S_x \right) \times \left(\dfrac{1}{(k-x+1)!} \right) \right]$
- 위 식은 FFT를 이용하면 모든 k에 대해 구할 수 있습니다.

2. +×

- k 번 연산 이후에, $v_1 v_2 \cdots v_{n-k}$ 이 존재할 확률을 계산합시다. ($i \geq 0$)
- 마지막의 별개의 수로 곱해져야하기 때문에, $v_1, v_2, \cdots, v_{n-k}$ 중 어느 두 쌍도 서로 선택되면 안 됩니다.
- i 번 연산을 진행 한 이후 연산에서 선택할 수 없는 쌍은 항상 $(n-k)(n-k-1)$ 개 있습니다.
- 총 경우의 수는 $\prod_{i=0}^{k-1} ((n-i)(n-i-1)-(n-k)(n-k-1))$ 입니다.
- $(n-i)(n-i-1)-(n-k)(n-k-1) = (k-i)(2n-(k+i+1))$ 을 사용합니다.
- $\prod_{i=0}^{k-1} ((n-i)(n-i-1)-(n-k)(n-k-1)) = \dfrac{k!(2n-k+1)!}{(2n-2k)!}$ 으로 정리 가능합니다.
- $B_k = \dfrac{n!(n-1)!k!(2n-k+1)!}{(2n-2k)!(n-k)!(n-k-1)!} S_{n-k}$ 입니다.