UCPC 2021 예선 풀이

Official Solutions

by

전국 대학생 프로그래밍 대회 동아리 연합

문제		의도한 난이도	출제자
A	수학은 체육과목 입니다 3	**Easy**	xiaowuc1
B	항체 인식	**Easy**	xiaowuc1
C	헤이카카오	**Easy**	evenharder
D	돌 가져가기	**Medium**	ainta
E	말뚝	**Medium**	golazcc83
F	종이, 펜, 삼각형	**Hard**	evenharder
G	경품 추첨	**Medium**	doju
H	스키장	**Medium**	xiaowuc1
I	흔한 타일 색칠 문제	**Medium**	evenharder
J	UCPC 만들기	**Hard**	jjwdi0

A. 수학은 체육과목 입니다 3

`string, implementation, brute force`
출제진 의도 – **Easy**

- 제출 501번, 정답 176팀 (정답률 35.13%)
- 처음 푼 팀: **[URGENT]** (ENCAPTURED BY THE SHADOW, Unleashed World, JUST FUN), 1분
- 출제자: `xiaowuc1`

- A 를 1부터 999까지의 정수 중 하나로 찍어 봅니다.
- 빈 문자열을 만들고, 입력으로 주어진 문자열 S 의 길이에 도달할 때까지, $A, A+1, A+2, \ldots$ 를 계속 붙입니다.
- 길이에 도달했다면, 두 문자열이 같은 지 확인합니다. 같다면 마지막으로 추가한 정수가 B 입니다.

- 숫자를 문자열로 변환하는 기능은 대부분의 언어에 내장되어 있습니다. (C 언어의 경우 `sprintf`, C++ 언어의 경우 `stringstream` 등)
- 만약 이러한 함수를 사용할 줄 모른다면, 직접 구현할 수 있습니다.
- 숫자를 십진법으로 출력해야 하니, 10으로 나눈 나머지를 구해서 일의 자리, 또 10으로 나눈 나머지를 구해서 십의 자리... 를 반복적으로 계산하면 됩니다.

- S 의 최대 길이가 2889이니, 최대 1000×2889 번 정도의 연산을 사용합니다. 일반적으로 1 초에 1억 번 이상의 연산이 가능하니, 시간 제한을 문제 없이 통과할 수 있습니다.
- 더 효율적인 풀이도 존재하며, A 가 아주 큰 정수여도 문제를 풀 수 있습니다.

B. 항체인식

`graph, dp`
출제진 의도 – **Easy**

- 제출 549번, 정답 181팀 (정답률 32.97%)
- 처음 푼 팀: **Capital Cities Starting with O** (Oslo, Ottawa, Ouagadougou), 3분
- 출제자: `xiaowuc1`

- 항체는 어떠한 셀에 투입된 후 반복적으로 값이 같은 인접한 셀을 탐색합니다.
- 이는 격자 그래프에서 같이 같은 셀에 대해서 Flood-fill을 하는 것과 동일합니다.
- Flood-fill은 DFS나 BFS로 구현할 수 있습니다.

- 항체가 어떠한 셀에 투입될지 모르니, 모든 NM 개의 셀을 시도해 봅니다.
- 항체가 처음 투입된 셀 $A_{i,j}$ 를 고정하면, 해당 셀에 새롭게 채워야 할 값은 $B_{i,j}$ 입니다.
- Flood-Fill을 사용하여 실제로 채워보고, 같은지 확인해 봅니다.
- 각 셀에 대해서 Flood-Fill을 해 보고 같은지 확인해 보는 것이 모두 $\mathcal{O}(NM)$ 시간에 해결됩니다.
- 고로 전체 문제는 $\mathcal{O}(N^2M^2)$ 에 해결됩니다.

- 사실 $A_{i,j} \neq B_{i,j}$ 인 아무 셀 하나만 골라서 Flood fill을 해 봐도 됩니다.
- 이 셀에 항체가 투입되어서 $A = B$가 되지 않았다면, 다른 곳에 투입해도 $A = B$ 가 되지 않습니다.
- 이 경우 전체 문제는 $\mathcal{O}(NM)$ 에 해결됩니다.

C. 헤이카카오

`math`

출제진 의도 – **Easy**

- 제출 406번, 정답 99팀 (정답률 24.63%)
- 처음 푼 팀: **[URGENT]** (ENCAPTURED BY THE SHADOW, Unleashed World, JUST FUN), 4분
- 출제자: `evenharder`

- 편의상 끝말잇기 한 판에 1분이 걸린다고 합시다.
- 초기 승리 확률이 d, 패배시 승률 증가 비율이 k라고 할 때 $r = k + 1$이라 하면 $P_{d,r}$를 승리할 때까지 걸리는 시간의 기댓값이라 할 수 있습니다. 그러면 초기값으로 $P_{1,r} = 1$이며, 점화식을 세워보면

$$\begin{aligned} P_{d,r} &= d \times 1 + (1 - d) \times (1 + P_{\min(dr,1),r}) \\ &= 1 + (1 - d) \times P_{\min(dr,1),r} \end{aligned}$$

이 성립합니다.

조금 더 풀어쓰면 다음과 같습니다. 편의상 $dr^3 \le 1$을 가정하겠습니다.

$$\begin{aligned}
P_{d,r} &= 1 + (1-d) \times P_{dr,r} \\
&= 1 + (1-d) + (1-d)(1-dr) \times (1 + P_{dr^2,r}) \\
&= 1 + (1-d) + (1-d)(1-dr) + (1-d)(1-dr)(1-dr^2) \times (1 + P_{dr^3,r}) \\
&= \cdots \\
&\vdots
\end{aligned}$$

$\log 100 / \log 1.01$이 약 462.8이기 때문에 약 500번 정도만 점화식을 풀어서 계산하면 됩니다.

D. 돌 가져가기

ad-hoc
출제진 의도 – **Medium**

- 제출 406번, 정답 99팀 (정답률 24.63%)
- 처음 푼 팀: Almost Retired, 10분
- 출제자: `ainta`

- 무게가 정해져 있는 N 개 돌이 흰색 또는 검은색으로 칠해져 일렬로 나열되어 있습니다.
- 돌을 하나씩 가져갈 때, 가져간 돌이 양 끝에 있지 않으며 인접한 두 돌과 모두 다른 색일 때 무게에 해당하는 점수를 얻습니다.
- 점수를 최대화하는 것이 문제입니다.

- 어떤 연속된 돌들이 같은 색이라면, 그 돌들 중 두개 이상에서 점수를 얻는 것은 불가능합니다.
- 따라서, 그 돌들 중 무게가 가장 무거운 하나만 남겨도 답은 동일합니다.

- 그러면 문제는 이제 흰색과 검은색 돌이 번갈아서 있을 때, 얻을 수 있는 최대 점수입니다.
- 지금부터는 처음 상태가 흰색과 검은색 돌이 번갈아 있는 상태라고 가정합시다.

- 양 끝의 돌에서 점수를 얻을 수 없고, 연속한 두 개의 돌에서 모두 점수를 얻는 것이 불가능합니다.
- 따라서 돌이 N개 있을 때, 어떤 순서로 돌을 가져가더라도 최대 $\lceil \frac{N-2}{2} \rceil$개의 돌에서만 점수를 얻을 수 있습니다.

- 그렇다면, 최대 점수를 얻는 방법은 어떻게 될까요?

- 돌이 N 개 있을 때, 어떤 순서로 돌을 가져가더라도 최대 $\lceil \frac{N-2}{2} \rceil$ 개의 돌에서만 점수를 얻을 수 있습니다.
- 한편, 양 끝의 두 돌을 제외한 $\lceil \frac{N-2}{2} \rceil$ 개의 돌을 어떻게 고르더라도 해당 돌들로부터 모두 점수를 얻는 방법이 존재합니다.
- 따라서, 양 끝의 돌을 제외한 돌들 중 가장 무거운 $\lceil \frac{N-2}{2} \rceil$ 개의 돌로부터 점수를 얻는것이 최적 전략입니다.

- 이전 슬라이드에서 말한것이 항상 가능함을 증명해봅시다.
- 먼저, $N = 2, N = 3$일 때 성립함은 자명합니다.
- $N \geq 4$이고 점수를 얻어야하는 돌 $\lceil \frac{N-2}{2} \rceil$개가 주어졌을 때, 만족하는 방법이 존재함을 보이면 충분합니다.

- $N \geq 4$이므로 양 끝의 돌이 아니면서 점수를 얻어야하는 돌이 아닌 돌이 존재합니다.
- 따라서 점수를 얻어야 하는 돌 x가 존재하여 x 양 옆의 돌 중 하나는 양 끝의 돌도 아니고, 점수를 얻어야 하는 돌도 아닙니다.
- 이를 y 라고 하면 x를 가져가고 y를 가져가고 나면 이제 x에서 점수를 얻었고, 돌 개수가 2개 줄었고, 점수를 얻어야 하는 돌 개수가 1개 줄어든 상태가 됩니다.
- 이제 귀납 가정에 의해 조건을 만족하도록 돌을 가져갈 수 있습니다.
- 따라서, 수학적 귀납법에 의해 돌 개수에 관계없이 성립함을 알 수 있습니다.

- 따라서, 가장 무거운 $\lceil \frac{N-2}{2} \rceil$ 개 돌의 합이 답이 됩니다.
- 시간복잡도는 정렬을 해야하므로 $\mathcal{O}\left(N \log N\right)$ 입니다.

E. 말뚝

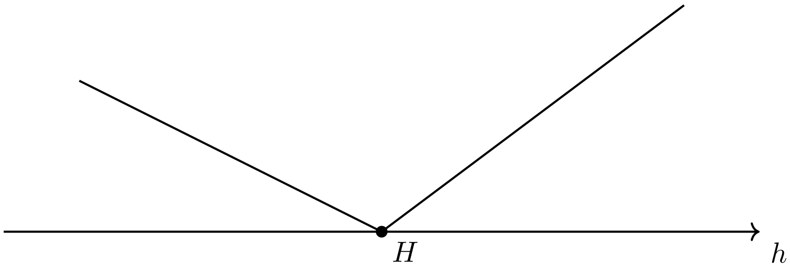
binary search, data structures
출제진 의도 – **Medium**

- 제출 406번, 정답 99팀 (정답률 24.63%)
- 처음 푼 팀: **여기가월파2020인가요** (월파, 치러, 왔어요), 10분
- 출제자: golazcc83

- 문제를 푸는 간단한 방법을 생각해 봅시다.
  - 크기가 $100\,000$인 배열을 만들고, 여기에 말뚝의 높이를 h로 만드는 데 필요한 힘을 저장합니다.
  - 모든 h에 대해, 1번부터 K번 말뚝의 높이를 h로 만드는 데 필요한 힘을 배열에 저장하고 최솟값을 가져옵니다.
  - $K+1$번째 말뚝의 높이를 h로 만드는 데 필요한 힘을 배열에 더하고, 1번째 말뚝에 대한 힘은 빼고, 최솟값을 가져옵니다.
  - 스위핑 기법을 이용하여 이 과정을 $N-K+1$번 반복하고, 최솟값이 가장 작은 것을 출력합니다.

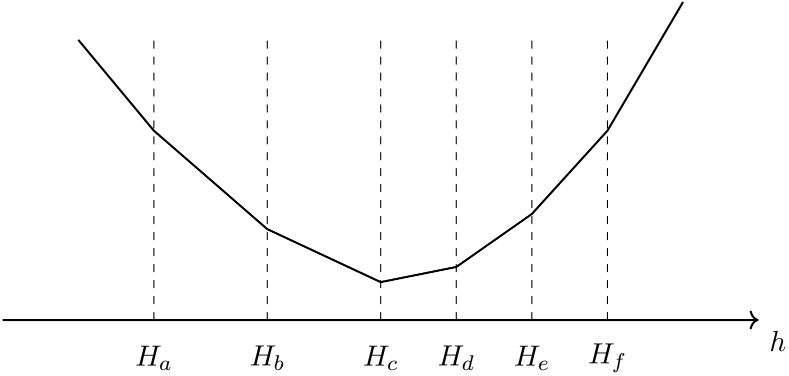
- 앞의 풀이는 모든 높이 H 에 대해 말뚝 N 개의 비용을 더하거나 빼는 과정이 필요하므로 $\mathcal{O}(NH)$ 의 시간이 걸립니다.
- 따라서 최솟값을 빠르게 찾을 수 있는 방법과 말뚝의 높이를 특정 높이로 만드는 데 필요한 힘을 관리하는 과정을 효율적으로 처리할 수 있는 자료구조가 필요합니다.

- 말뚝 하나의 처음 높이가 H cm일 때, 말뚝의 높이를 h cm으로 만들기 위한 힘은 아래와 같이 2개의 일차함수로 이루어져 있습니다.



- 이제 말뚝 여러 개의 높이를 h cm로 만드는 비용은 어떤 그래프 개형을 가질 지 생각해 봅시다.
  - h가 무한히 작아지면 기울기는 음수가 됩니다.
  - h가 무한히 커지면 기울기는 양수가 됩니다.
  - 각 말뚝의 최초 높이를 기준으로 왼쪽 그래프의 기울기보다 오른쪽 그래프의 기울기가 더 큽니다.

- 앞에서 관찰한 3가지 사실로 아래의 그래프를 그릴 수 있습니다.



- 이 때 최솟값이 되는 h를 찾는 방법에는
  - 앞의 그래프가 unimodal하다는 점을 이용해 삼분 탐색하거나
  - 그래프의 기울기가 0 이상이 되는 지점을 찾아 이분 탐색할 수 있습니다.
- 위 2가지 방법을 이용하면 $1 \sim K$ 번째 말뚝으로 UCPC 농장을 아름답게 만들기 위해 필요한 힘의 최솟값을 $\mathcal{O}(\log H)$의 시간 안에 구할 수 있습니다.

- i 번째 말뚝의 높이를 h 로 만드는 그래프 $f(h)$ 는 다음과 같이 두 개의 1차 함수로 나타낼 수 있습니다.

$$f(h) = \begin{cases} A_i(h - H_i),\ h \ge H_i \\ B_i(H_i - h),\ h < H_i \end{cases} = \begin{cases} A_ih - A_iH_i,\ h \ge H_i \\ -B_ih + B_iH_i,\ h < H_i \end{cases}$$

- 각 h 마다 위 함수의 1차 항의 계수와 상수항을 관리하는 자료구조가 있다고 가정합시다.

- 앞의 자료구조에 i번째 말뚝이 차지하는 비용을 추가하거나 제외하려면
- $[1, H_i]$ 에 왼쪽 그래프의 1차 항의 계수와 상수항을,
  $(H_i, 100\,000]$ 에 오른쪽 그래프의 1차 항의 계수와 상수항을 더하거나 빼주면 됩니다.
- 이는 Segment Tree나 Fenwick Tree를 활용한 구간합 문제로 치환하여 각 연산 당 $\mathcal{O}(\log H)$ 에 수행할 수 있습니다.
- 말뚝들의 높이를 h 로 만드는 비용을 구할 때에도 1차항의 계수와 상수항을 $\mathcal{O}(\log H)$ 시간에 구하여 1차항$\times h +$ 상수항 을 계산하면 됩니다.

- Segment Tree로 $[i, i+K-1]$번째 말뚝의 비용 함수를 관리하고 있을 때,
- 말뚝들의 높이를 h로 만드는 비용을 구하는 데 $\mathcal{O}(\log H)$
- 최솟값을 찾아 이분 탐색 또는 삼분 탐색을 하는 데 $\mathcal{O}(\log H)$
- 총 $\mathcal{O}(\log^2 H)$의 시간 복잡도에 UCPC 농장을 아름답게 만들기 위해 드는 힘의 최솟값을 구할 수 있습니다.
- 그래프의 기울기가 0 이상이 되는 지점을 찾는 쿼리를 구현한다면 $\mathcal{O}(\log H)$ 안에 구할 수도 있습니다.

- Segment Tree로 $[i, i+K-1]$ 번째 말뚝의 비용 함수를 $[i+1, i+K]$ 번째 말뚝의 비용 함수로 만들기 위해서는 스위핑 기법을 이용하여 $i+K$ 번째 말뚝 비용 함수를 더하고, i 번째 말뚝 비용 함수를 빼면 됩니다.
- 이 모든 과정을 최대 N 번 반복하므로 총 시간 복잡도는 $\mathcal{O}\left(N \log^2 H\right)$, $\mathcal{O}\left(N \log H\right)$ 입니다.

F. 종이, 펜, 삼각형

prefix sum, fft
출제진 의도 – **Hard**

- 제출 ??번, 정답 ??팀 (정답률 ??.??%)
- 처음 푼 팀: **???** (??, ??, ??), ??분
- 출제자: evenharder

- 기울기가 다르며 공통교점이 없는 세 직선을 선택하면 삼각형이 유일하게 결정됩니다.
- 그러므로 기울기별로 직선의 절편 a_i, b_i, c_i 를 어떻게 잘 고를지가 관건입니다.
- d가 120으로 들어오는 직선의 절편 비스무리한 값 x를 $m - x$로 바꾸어 생각해봅시다. 그럼 큰 정삼각형을 이루는 각 변은 기울기별로 값을 0으로 표현할 수 있습니다.
  - 같은 기울기인 변과의 상대거리로도 볼 수 있습니다.

기울기에 따라 직선별로 값을 a_i, b_i, c_i 처럼 부여하면 삼각형이 만들어질 조건도 고려할 수 있습니다.

$$a_i + b_i \leq m, b_i + c_i \leq m, c_i + a_i \leq m, a_i + b_i + c_i \neq m$$

- 처음 세 조건은 만들어진 정삼각형의 꼭짓점이 큰 정삼각형을 벗어나지 않게 해줍니다.
  - 정삼각형 R의 세 꼭짓점이 큰 정삼각형 안에 있다와 정삼각형 R이 큰 정삼각형 안에 있음은 동치입니다.
- 마지막 조건은 세 직선이 한 점에서 만나는 경우를 제외합니다.

$a_i + b_i + c_i = m$ 이면 처음 세 조건이 성립하기 때문에, 정답은 처음 세 조건만 고려했을 때의 순서쌍 개수에서 $a_i + b_i + c_i = m$ 을 만족하는 순서쌍 개수를 뺀 값입니다.

- 처음 세 조건을 만족하는 직선의 쌍은 각 기울기별로 i 이하의 직선의 개수를 저장한 누적 합을 이용해 $\mathcal{O}(m)$에 셀 수 있습니다.
  - $m/2$보다 큰 값은 최대 하나만 있을 수 있기 때문에, 하나를 $m/2$보다 크게 잡으면 나머지 수의 최댓값이 정해집니다. 이 두 수의 합은 m을 넘지 않기 때문에 나머지 두 직선의 개수를 곱하면 됩니다.
  - 전부 다 $m/2$ 이하이면 조건을 항상 만족하므로 직선의 개수를 곱하면 됩니다.
- 제외해야 하는 조건인 $a_i + b_i + c_i = m$는 유명한 3SUM 문제입니다. 값의 범위가 0 이상 m 이하이기 때문에 고속 푸리에 변환(FFT)을 이용해 $\mathcal{O}(m \log m)$에 쌍의 개수를 구할 수 있습니다.

G. 경품 추첨

constructive, math

출제진 의도 – **Medium**

- 제출 –번, 정답 –팀 (정답률 –.–%)
- 처음 푼 팀: –
- 출제자: doju

G. 경품 추첨

- 한 상자에 같은 수가 두 개 이상 있으면 당연히 불가능합니다.
- 두 상자에서 두 수의 차이가 같은 쌍을 하나씩 뽑을 수 있다면 불가능합니다.
  - $a_i + b_i = a_j + b_j \iff a_i - a_j = b_j - b_i$

$$1,\quad \overbrace{2,\quad 9}^{7},\quad 10$$

$$\underbrace{3,\quad 6,\quad 8,\quad 10}_{7}$$

$\rightarrow$ 2 + 10 = 3 + 9

- 따라서 각 상자에서 나올 수 있는 두 수의 차를 모두 나열했을 때, 어느 두 상자도 서로 겹치는 값이 있어서는 안 됩니다.
- 한 상자 안에서는 서로 겹쳐도 됩니다.

$$
\begin{array}{ccc}
[\ 20,\ 5,\ 17,\ 1\ ] & & \{\ 3,\ 4,\ 12,\ 15,\ 16,\ 19\ \} \\
[\ 18,\ 11,\ 16,\ 5\ ] & \rightarrow & \{\ 2,\ 5,\ 6,\ 7,\ 11,\ 13\ \} \\
[\ 13,\ 3,\ 12,\ 21\ ] & & \{\ 1,\ 8,\ 9,\ 9,\ 10,\ 18\ \}
\end{array}
$$

- 따라서 등장하는 차이 값들이 다양하지 않고 규칙적일수록 유리합니다.

- 따라서 등장하는 차이 값들이 다양하지 않고 규칙적일수록 유리합니다.
- 각 상자에 **등차수열**을 채우는 전략을 생각해 봅시다.

- 상자 A에 공차가 a인 등차수열, 상자 B에 공차가 b인 등차수열을 넣습니다.
- A에서 등장하는 두 수의 차는 $\{a, 2a, 3a, ..., (N-1)a\}$ 입니다.
- B에서 등장하는 두 수의 차는 $\{b, 2b, 3b, ..., (N-1)b\}$ 입니다.

G. 경품 추첨

- 상자 A에 공차가 a인 등차수열, 상자 B에 공차가 b인 등차수열을 넣습니다.
- A에서 등장하는 두 수의 차는 $\{a, 2a, 3a, \ldots, (N-1)a\}$ 입니다.
- B에서 등장하는 두 수의 차는 $\{b, 2b, 3b, \ldots, (N-1)b\}$ 입니다.

- 두 수열이 공유할 수 있는 가장 작은 수는 a와 b의 최소공배수입니다.
- 따라서 최소공배수가 적어도 두 수열 중 하나에서 등장하지 않는다면, 두 수열에는 서로 겹치는 수가 없습니다.
  - 그리고 두 상자를 통해 만들어지는 당첨 번호는 서로 겹치지 않습니다.

- a와 b의 최대공약수를 g라고 할 때, $a \geq gN$ 또는 $b \geq gN$를 만족해야 합니다.
- 공에 적을 수 있는 수가 $5\,000\,000$ 이하이므로, N이 2000일 경우 공차는 약 2500 이하여야 합니다.
- 따라서 a와 b는 서로소여야 합니다.

- 종합하면 모두 서로 서로소이고, 최대 하나를 제외하고 전부 N 이상인 공차 K개를 찾으면 문제를 풀 수 있습니다.

- 가장 간단한 방법은 모든 공차를 **소수**로 정하는 것입니다.
- N 이상인 소수 K 개를 찾고, 각각을 공차로 하는 길이가 N 인 등차수열 K 개를 출력하면 답이 됩니다.
  - 2000 이상 2500 이하인 소수는 총 64 개입니다.
  - 굳이 원한다면 공차 하나는 N 미만이어도 됩니다.

H. 스키장

graph, dp
출제진 의도 – **Medium**

- 제출 254번, 정답 68팀 (정답률 26.77%)
- 처음 푼 팀: **갓갓갓** (갓, 삿갓, 갓난아이), 9분
- 출제자: xiaowuc1

- 주어진 그래프는 Directed Acyclic Graph (DAG) 입니다.
- 이 그래프에서 동적 계획법 (DP, Dynamic Programming) 을 사용하여 해결할 수 있습니다.

- $K = 0$ 이라고 생각하고 문제를 해결해 봅시다.
- $DP[v]$ 를 v 에서 시작해서 T 로 가면서 스키를 타는 최대 시간이라고 정의합시다.
- $v = T$ 일 경우 $DP[v] = 0$ 입니다.
- 그렇지 않을 경우, $v = a_i$ 에서 나가는 모든 코스 (b_i, t_i) 에 대해 $DP[v] = max(DP[v], DP[b_i] + t_i)$ 입니다. 나가는 간선이 없다면 $DP[v] = -\infty$ 라고 둡시다.
- v 를 $N, N-1, N-2, \ldots, 1$ 순서대로 계산하면 참조하는 값이 항상 이미 계산되어 있습니다. 고로 알고리즘은 올바릅니다.

- 모든 정점 v 에 대해서, $v = a_i$ 에서 나가는 코스들을 인접 리스트에 저장합시다. (C++ 에서는 일반적으로 `std::vector` 를 사용합니다.)
- 이렇게 저장할 경우 각 정점에 대해서 $\mathcal{O}(1)$ + $\mathcal{O}$ (해당 정점에 연결된 코스 수) 만큼의 시간에 DP 값을 계산할 수 있습니다.
- 모든 정점에 대해 이를 반복하면 $\mathcal{O}(N+M)$ 시간에 문제를 해결할 수 있습니다.
- 그래프의 최단 경로를 구하는 문제가 아니기 때문에 Dijkstra's Algorithm은 사용할 수 없음에 유의하세요.

- $K > 0$ 일 때는 이 DP를 조금 변형해야 합니다.
- $DP[k][v]$ 를 v 에서 시작해서 최대 k 번 리프트를 타면서 스키를 타는 최대 시간이라고 정의합시다.
- $k = 0, v = T$ 일 경우 $DP[k][v] = 0$ 입니다.
- 그렇지 않을 경우, $v = a_i$ 에서 나가는 모든 코스 (b_i, t_i) 에 대해 $DP[k][v] = max(DP[k][v], DP[k][b_i] + t_i)$ 입니다.
- 추가로, $v = b_i$ 에서 나가는 모든 리프트 (a_i, t_i) 에 대해 $DP[k][v] = max(DP[k][v], DP[k-1][a_i])$ 입니다. ($k > 0$ 일 때만 사용 가능합니다.)

- k 를 $0, 1, \ldots, K$ 순서대로, 그 안에서 v 를 $N, N-1, N-2, \ldots, 1$ 순서대로 계산하면 참조하는 값이 항상 이미 계산되어 있습니다. 고로 알고리즘은 올바릅니다.
- 맨 처음 알고리즘의 연산을 K 번 반복하니 $\mathcal{O}\left(K(N+M)\right)$ 시간에 문제를 해결할 수 있습니다.

I. 흔한 타일 색칠 문제

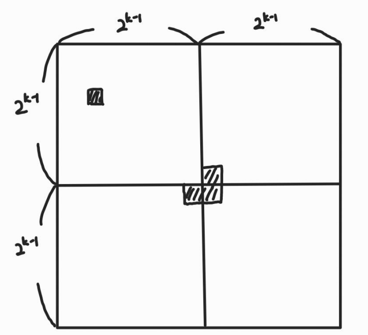
divide and conquer, constructive

출제진 의도 – **Medium**

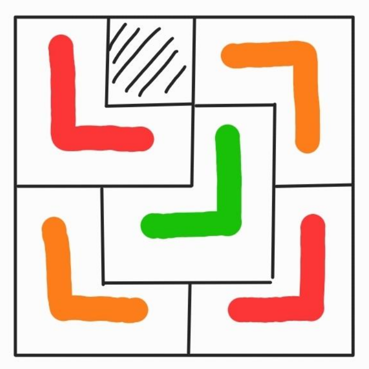
- 제출 ??번, 정답 ??팀 (정답률 ??.??%)
- 처음 푼 팀: **???** (??, ??, ??), ??분
- 출제자: evenharder

우선 색깔을 무시하면, 수학적 귀납법에 기반한 분할 정복을 이용해 배치를 구할 수 있습니다. BOJ에도 올라와 있습니다.

- $k = 1$일 때는 L-트로미노가 1개입니다.
- $k \geq 2$일 때는 판을 변의 길이가 절반이 되게 4개의 정사각형으로 나눕니다.
  - 구멍이 있는 쪽은 재귀적으로 해결할 수 있습니다.
  - 나머지는 네 정사각형이 모이는 모서리 타일을 연결해 L-트로미노를 만듭니다. 정사각형에 구멍이 1개씩 생겼으므로 재귀적으로 해결할 수 있습니다.



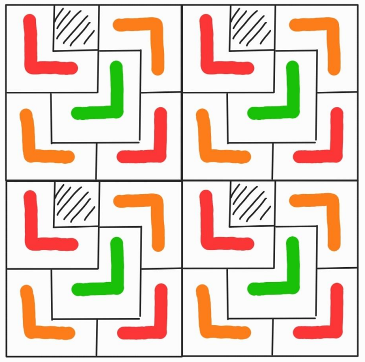
흥미롭게도, 이 분할 정복 아이디어를 색칠에도 그대로 적용할 수 있습니다.



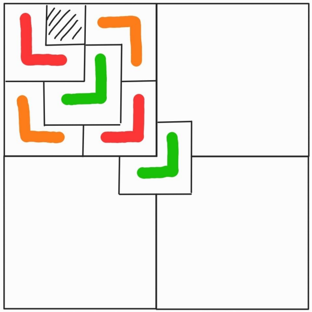
- $k = 1$일 때는 L-트로미노가 1개입니다.
- $k = 2$일 때는 배치 및 칠할 수 있는 방법이 유일합니다. 편의상 좌상단 2×2를 a로, 이웃한 귀퉁이를 b로 채웠습니다. 가운데에는 c가 위치합니다.

$k \geq 3$일 때도 좌상단 2×2를 a로, 우상단 2×2를 b로 채우고 가운데에는 c가 위치하는 배치를 만들 수 있습니다. 거기에 배치의 테두리에 c가 없게 칠할 수 있습니다.

- 동일하게 판을 변의 크기가 2^{k-1}인 정사각형 4개로 분할합니다.
- 구멍이 있는 정사각형은 앞에서 말한 분할 정복 방식으로 L-트로미노를 배치하고 좌상단 모서리 타일이 a가 되도록 칠합니다.
- 이 때 정사각형의 테두리를 보면, a와 b가 두 칸 간격으로 체커보드처럼 교대하며 나타납니다.

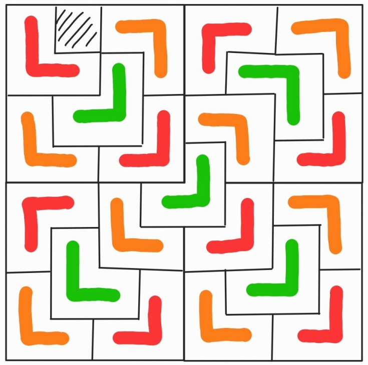


이 상태에서, 정사각형이 모이는 귀퉁이 타일만 c로 칠해볼 수 있습니다. 앞서 배치의 테두리에 c가 없다고 했기에 구별되게 칠할 수 있습니다.



나머지 세 개의 정사각형은 귀퉁이 타일을 구멍이라 생각하고 똑같은 규칙으로 모든 L-트로미노가 구별되게 색칠할 수 있습니다.

- 색깔 a와 b의 L-트로미노는 체커보드 구조에 의해 구별됩니다.
- 가운데에 생기는 색깔 c의 L-트로미노는 테두리 타일의 색이 a 또는 b이기 때문에 구별됩니다.
- 각 정사각형의 테두리 안에 있는 L-트로미노는 귀납법에 의해 모두 구별됩니다.



- 결론적으로 분할 정복 기반으로 3개의 색으로 모든 L-트로미노가 구별되게 배치할 수 있습니다. 시간 복잡도 및 공간 복잡도는 $\mathcal{O}\left(2^{2K}\right)$ 입니다.
- 원래는 4색으로 칠하는 문제였으나, 검수 과정에서 타일을 하나씩 차례로 보며 그리디하게 색을 칠해도 된다는 성질이 발견되어 3색으로 수정되었습니다.

J. UCPC 만들기

centroid decomposition, smaller to larger
출제진 의도 – **Hard**

- 제출 ??번, 정답 ??팀 (정답률 123.45%)
- 처음 푼 팀: **???** (?, ?, ?), ??분
- 출제자: jjwdi0

- $(\text{UCPC})^k$ 형태의 문자열을 만드는 조건은 무엇일까요?

- $(\text{UCPC})^k$ 형태의 문자열을 만드는 조건은 무엇일까요?
- 경로에 들어 있는 U, C, P의 개수가 1:2:1을 만족해야 합니다.

- $(\text{UCPC})^k$ 형태의 문자열을 만드는 조건은 무엇일까요?
- 경로에 들어 있는 U, C, P의 개수가 1:2:1을 만족해야 합니다.
- 가능한 모든 $\mathcal{O}\left(N^2\right)$ 가지 경로를 고려하는 것은 아무리 잘 세어 주어도 시간초과를 받습니다.

- 만약 트리가 일직선이라면 어떻게 풀까요?

- 만약 트리가 일직선이라면 어떻게 풀까요?
- 트리가 번호 순서대로($1 - 2 - 3 - \ldots - N$) 연결되어 있는 형태라 가정하고, $S[i]$를 i번 정점의 문자라고 가정합시다.
- 경로상의 (U의 개수, C의 개수, P의 개수)를 (u, c, p)라고 나타내면, $(u, c, p) = (k, 2k, k)$를 만족하는 어떤 정수 k가 존재하는 구간을 찾아야 합니다.

- $(u, c, p) = (k, 2k, k)$ 임을 이용하면 조건을 좀 더 간단하게 바꿀 수 있습니다.
  - $u = p, u + p = c$ 와 같은 형태의 조건을 뜻합니다.

- $(u, c, p) = (k, 2k, k)$ 임을 이용하면 조건을 좀 더 간단하게 바꿀 수 있습니다.
  - $u = p, u + p = c$ 와 같은 형태의 조건을 뜻합니다.
- 그렇다면 (UCPC)^k 형태의 문자열을 만드는 경로는, $(u - p, u + p - c) = (0, 0)$ 을 만족함을 알 수 있습니다.

- 앞에서 관찰한 사실을 일직선 모양의 트리에 적용해보겠습니다.

- 앞에서 관찰한 사실을 일직선 모양의 트리에 적용해보겠습니다.
- 편의상 정점 u에서 정점 v까지의 경로를 $path(u, v)$라 하겠습니다.
- 트리가 일직선 모양이면, $path(i, j)$는 $path(1, j)$에서 $path(1, i-1)$을 뺀 것과 같습니다.
- 배열에서 부분합을 구하는 방법과 유사합니다.

- j 를 증가시키면서 $path(i,j)$ 의 $(u-p,u+p-c)$ 가 $(0,0)$ 인 i 의 개수를 빠르게 세어주면 됩니다.
- 이러한 작업은 이진 탐색 트리(C++의 `std::map` 등)를 이용하면 빠르게 수행할 수 있습니다.
- 이와 비슷한 문제로는 boj.kr/17736이 있습니다.

- j를 증가시키면서 $path(i,j)$의 $(u-p, u+p-c)$가 $(0,0)$인 i의 개수를 빠르게 세어주면 됩니다.
- 이러한 작업은 이진 탐색 트리(C++의 `std::map` 등)를 이용하면 빠르게 수행할 수 있습니다.
- 이와 비슷한 문제로는 boj.kr/17736이 있습니다.

- 이제 일직선에서 일반적인 트리 형태로 확장해봅시다.

- 트리에서 경로를 다뤄야 하는 경우에는 분할 정복을 종종 이용합니다.

- 트리에서 경로를 다뤄야 하는 경우에는 분할 정복을 종종 이용합니다.

- 어떤 정점 u에 대해, u를 지나는 모든 경로를 처리해주고 u를 트리에서 제거합니다.
- 그 다음 여러 컴포넌트로 나뉘어진 트리에서 재귀적으로 문제를 해결합니다.

- 트리에서 경로를 다뤄야 하는 경우에는 분할 정복을 종종 이용합니다.

- 어떤 정점 u에 대해, u를 지나는 모든 경로를 처리해주고 u를 트리에서 제거합니다.
- 그 다음 여러 컴포넌트로 나뉘어진 트리에서 재귀적으로 문제를 해결합니다.
- 여기서 정점 u를 트리의 센트로이드로 정하면, $\mathcal{O}(N \log N)$의 시간복잡도로 모든 경로를 처리할 수 있습니다. 이러한 기법을 `centroid decomposition`이라 합니다.

- 분량 관계상 이에 대한 자세한 설명은 생략합니다.

- 트리의 센트로이드에 해당하는 정점 u에 대해, u까지 이르는 모든 경로를 이진 탐색 트리에 저장합시다.

- 트리의 센트로이드에 해당하는 정점 u에 대해, u까지 이르는 모든 경로를 이진 탐색 트리에 저장합시다.
- 그 후, 이진 탐색 트리를 이용해 문제의 조건에 맞는 경로를 세어줍니다. 센트로이드까지 이르는 경로는 $\mathcal{O}(N)$개고, 각 경로마다 $\mathcal{O}(\log N)$의 삽입/탐색을 거치니 $\mathcal{O}(N \log N)$의 시간복잡도로 세어줄 수 있습니다.

- 트리의 센트로이드에 해당하는 정점 u에 대해, u까지 이르는 모든 경로를 이진 탐색 트리에 저장합시다.
- 그 후, 이진 탐색 트리를 이용해 문제의 조건에 맞는 경로를 세어줍니다. 센트로이드까지 이르는 경로는 $\mathcal{O}(N)$개고, 각 경로마다 $\mathcal{O}(\log N)$의 삽입/탐색을 거치니 $\mathcal{O}(N \log N)$의 시간복잡도로 세어줄 수 있습니다.
- 그 후 트리를 분할하여 재귀적으로 위 작업을 반복합니다.

- 트리의 센트로이드에 해당하는 정점 u에 대해, u까지 이르는 모든 경로를 이진 탐색 트리에 저장합시다.
- 그 후, 이진 탐색 트리를 이용해 문제의 조건에 맞는 경로를 세어줍니다. 센트로이드까지 이르는 경로는 $\mathcal{O}(N)$개고, 각 경로마다 $\mathcal{O}(\log N)$의 삽입/탐색을 거치니 $\mathcal{O}(N \log N)$의 시간복잡도로 세어줄 수 있습니다.
- 그 후 트리를 분할하여 재귀적으로 위 작업을 반복합니다.
- 마스터 정리를 이용하면 전체 시간복잡도는 $\mathcal{O}\left(N \log^2 N\right)$임을 알 수 있습니다.

- 분할 정복을 사용하지 않는 방법도 있습니다.

- 분할 정복을 사용하지 않는 방법도 있습니다.

- 각 서브트리마다 루트까지 이르는 경로의 정보를 이진 탐색 트리로 관리합니다. 그렇다면 부모 정점의 이진 탐색 트리를 자식 정점의 이진 탐색 트리를 이용해 구성할 수 있습니다.

- 분할 정복을 사용하지 않는 방법도 있습니다.

- 각 서브트리마다 루트까지 이르는 경로의 정보를 이진 탐색 트리로 관리합니다. 그렇다면 부모 정점의 이진 탐색 트리를 자식 정점의 이진 탐색 트리를 이용해 구성할 수 있습니다.
- 이 과정에서 이진 탐색 트리의 정보를 합쳐줘야 하는데, 크기가 작은 집합에서 큰 집합으로 합쳐주게 된다면 삽입 횟수가 $\mathcal{O}(N \log N)$ 이 된다는 사실이 알려져 있습니다.

- 분할 정복을 사용하지 않는 방법도 있습니다.

- 각 서브트리마다 루트까지 이르는 경로의 정보를 이진 탐색 트리로 관리합니다. 그렇다면 부모 정점의 이진 탐색 트리를 자식 정점의 이진 탐색 트리를 이용해 구성할 수 있습니다.
- 이 과정에서 이진 탐색 트리의 정보를 합쳐줘야 하는데, 크기가 작은 집합에서 큰 집합으로 합쳐주게 된다면 삽입 횟수가 $\mathcal{O}(N \log N)$ 이 된다는 사실이 알려져 있습니다.

- 위 방법 역시 $\mathcal{O}\left(N \log^2 N\right)$ 의 시간복잡도로 문제를 해결할 수 있습니다.