UCPC 2024

전국 대학생 프로그래밍 대회 동아리 연합
여름 대회 2024

Preliminaries

Official Solutions

예선 해설

전국 대학생 프로그래밍 대회 동아리 연합 · UCPC 2024 출제진

박승원 (veydpz) 김동현 (kdh9949) 정재헌 (Gravekper) pjshwa 임지환 (raararaara) kclee2172 이종서 (leejseo) 이상헌 (evenharder) 김준겸 (ryute)

문제		의도한 난이도	출제자
A	체육은 수학과목 입니다	**Easy**	kipa00
B	Two Trees, Twelve Forests	**Challenging**	ainta
C	미어캣	**Hard**	golazcc83
D	이진 검색 트리 복원하기	**Medium**	chansol
E	지금 자면 꿈을 꾸지만	**Easy**	cozyyg
F	두 배	**Hard**	mingyu331
G	석고 모형 만들기	**Medium**	queued_q
H	만보기 대행 서비스	**Medium**	functionx
I	민들레	**Hard**	sorohue
J	동전 쌍 뒤집기	**Hard**	bubbler
K	나무 심기	**Hard**	molamola

A. 체육은 수학과목입니다

UCPC 2024

`arithmetic`

출제진 의도 – **Easy**

- 제출 245번, 정답 204팀 (정답률 83.265%)
- 처음 푼 팀: **개척단 훈련소**, 0분
- 출제자: `kipa00`
- 가장 짧은 정해: 120B^{C++}, 43B^{Python}

A. 체육은 수학과목 입니다

- 직사각형의 변의 길이가 주어졌을 때, 내접하는 원의 반지름의 길이를 구하는 문제입니다.
- 두 변의 길이 중 작은 쪽의 절반이 길이가 됩니다.
- 입력 단위는 미터(m), 출력 단위는 센티미터(cm)이므로 100을 곱해야 합니다.
- 최종적으로 두 수 중 작은 쪽에 50을 곱해 출력하면 됩니다.

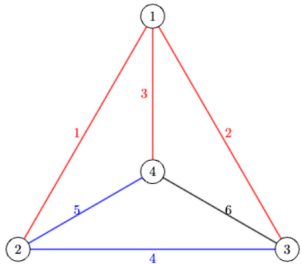
B. Two trees, twelve forests

`constructive`, `ad-hoc`

출제진 의도 – **Challenging**

- 제출 19번, 정답 6팀 (정답률 31.579%)
- 처음 푼 팀: **Maruiimtoretyu**, 89분
- 출제자: `ainta`
- 가장 짧은 정해: 866B^{C++}, 1,301B^{Python}

- $k = 3, N = 4$인 경우의 해가 다음과 같이 존재합니다.



- Idea: (k, N)의 해에서 $(k+1, 2N-1)$의 해를 만들 수 있습니다.

- Idea: (k, N)의 해에서 $(k+1, 2N-1)$의 해를 만들 수 있습니다.
- (k, N)의 해에서 정점 $N+1, N+2, \cdots, 2N-1$ 및 다음과 같은 간선들이 추가되었다고 생각해봅시다.
  - $N+i$번 정점과 i번 정점을 연결하는 간선 $(1 \le i \le N-1)$
  - $N+i$번 정점과 $i+1$번 정점을 연결하는 간선 $(1 \le i \le N-1)$
- 새로 추가된 간선들은 기존 간선들보다 가중치가 작습니다.

- (k, N)의 해를 두 트리 T_1과 T_2의 합집합으로 표현할 수 있었다면, T_1에는 $N+i$번 정점과 i번 정점을 잇는 간선들을, T_2에는 $N+i$번 정점과 $i+1$번 정점을 연결하는 간선들을 추가하면 새로운 해의 두 트리가 됩니다.
- 이제 F_i들이 어떻게 계산되는지 살펴봅시다.
  - F_1은 새로 추가된 간선 $2N-2$개로 이루어진 트리가 됩니다.
  - $F_2, \cdots, F_{k+1}$은 (k, N)의 해에서 계산된 $F_1, \cdots, F_k$가 됩니다.
- 따라서, (k, N)의 해에서 $(k+1, 2N-1)$의 해를 만들 수 있습니다.

- 앞서 설명한 아이디어를 이용하면 $k = 12$일 때 $N = 1537$이 되어 정점의 개수가 2024개를 넘지 않습니다.
- Challenge: 간선을 공유하지 않는 트리 k개의 합집합 G에 대해, 숲 점수의 최대값은 $O(k \log N)$임을 증명해봅시다.

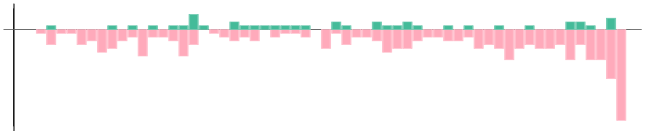
C. 미어캣

UCPC 2024

gɪeedy

출제진 의도 – **Hard**

- 제출 272번, 정답 42팀 (정답률 15.441%)
- 처음 푼 팀: **Rainy Day**, 10분
- 출제자: golazcc83
- 가장 짧은 정해: 1,131B^{C++}, 1,234B^{Python}



- 같은 방향을 바라보는 미어캣 둘을 고르고 서로 자리를 바꾸는 행동을 여러 번 할 수 있다면, 같은 방향을 바라보는 미어캣끼리는 자유롭게 자리를 배치할 수 있습니다.
- 편의상 왼쪽을 바라보는 미어캣은 L미어캣, 오른쪽을 바라보는 미어캣은 R미어캣이라 하겠습니다.
- 키가 가장 큰 미어캣을 기준으로 왼쪽은 L미어캣만, 오른쪽은 R미어캣만 망을 볼 수 있습니다.

C. 미어캣

- 키가 가장 큰 미어캣을 기준으로 왼쪽은 L미어캣만, 오른쪽은 R미어캣만 망을 볼 수 있습니다. 그렇다면 왼쪽과 오른쪽에는 각각 어떤 미어캣을 배치해야 좋을까요?
- 왼쪽에는 L미어캣만 망을 볼 수 있으므로 키가 큰 L미어캣이, R미어캣은 L미어캣의 시야를 방해하므로 키가 작은 R미어캣이 배치되어야 합니다.
- 오른쪽에는 R미어캣만 망을 볼 수 있으므로 키가 큰 R미어캣이, L미어캣은 R미어캣의 시야를 방해하므로 키가 작은 L미어캣이 배치되어야 합니다.
- 왼쪽과 오른쪽에 어떤 미어캣을 배치할 지는 L미어캣과 R미어캣끼리 묶고, 키 순서대로 정렬하여 구할 수 있습니다.

- 키가 가장 큰 미어캣을 기준으로 왼쪽과 오른쪽에 어떤 미어캣을 배치할 지 정할 수 있습니다. 이 때 각 구역마다 망을 볼 수 있는 미어캣의 수를 최대화하는 방법은 아래와 같습니다.
  - 망을 볼 수 있는 방향과 반대 방향을 바라보고 있는 미어캣은 키가 작은 미어캣부터 앞쪽으로 배치합니다.
  - 망을 볼 수 있는 방향을 바라보고 있는 미어캣은 앞쪽부터 시작하여 그 자리에서 망을 볼 수 있는 키가 가장 작은 미어캣을 배치합니다.
- 위의 과정은 $\mathcal{O}(N)$에 수행 가능하며, 키가 가장 큰 미어캣의 위치는 최대 N개이므로 총 시간복잡도는 $\mathcal{O}(N^2)$입니다. 해당 시간복잡도로 이 문제를 풀 수 있습니다.
- 별해 : 모든 구역마다 망을 볼 수 있는 미어캣의 수를 이분 탐색을 활용하여 $\mathcal{O}(N \log N)$에 전처리하고, 위의 과정을 $\mathcal{O}(1)$에 수행하여 총 시간복잡도 $\mathcal{O}(N \log N)$에 해결하는 풀이가 있습니다.

D. 이진 검색 트리 복원하기

`sorting`, `two_pointer`, `binary_search`

출제진 의도 – **Medium**

- 제출 631번, 정답 79팀 (정답률 12.520%)
- 처음 푼 팀: **moruii**, 12분
- 출제자: `chansol`
- 가장 짧은 정해: 1,006B^{C++}, 1,975B^{Python}

D. 이진 검색 트리 복원하기

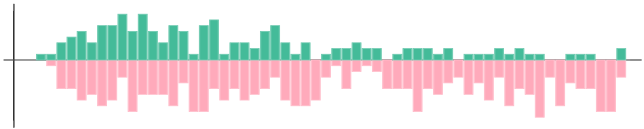
- 루트부터 깊이가 낮은 순서대로 이진 검색 트리를 복원해나가면 됩니다.
- 깊이가 i인 노드들을 복원할 때,
  - 깊이가 $i-1$인 각 노드의 왼쪽/오른쪽 자식이 가질 수 있는 값의 구간을 구한 다음, 구간에 맞는 깊이 i짜리 노드를 할당합니다. 이분 탐색 또는 투 포인터로 빠르게 처리할 수 있습니다.
  - 같은 구간에 여러 노드가 있거나 할당하지 못한 깊이 i짜리 노드가 있다면, 입력에 맞는 이진 검색 트리를 만들 수 없으므로 -1을 출력합니다.
  - $[l, r]$ 구간에 값이 x인 노드를 할당했다면, 해당 노드의 왼쪽/오른쪽 자식이 가질 수 있는 값의 구간은 각각 $[l, x-1]$, $[x+1, r]$입니다.

E. 지금 자면 꿈을 꾸지만

`bruteforcing`, `two_pointer`, `greedy`

출제진 의도 – **Easy**

- 제출 465번, 정답 147팀 (정답률 31.613%)
- 처음 푼 팀: **사상 첫 포핏**, 7분
- 출제자: cozyyg
- 가장 짧은 정해: 602B^{C++}, 417B^{Python}



- 잠을 최대 한 번 잘 수 있기 때문에, **잠을 자기 전에 완료할 과제의 수** M을 결정해야 합니다.
- 또한 **잠을 잘 시간** BX에서의 X를 결정해야 합니다.
- 잠을 자는 시간과 과제를 하는 시간 이외의 시간을 낭비할 필요는 없습니다.
- M과 X를 결정하면, 과제를 완료하는 시각들은 모두 결정됩니다.
- $0 \le M \le N, 0 \le X \le A-1$이므로 총 $\mathcal{O}(AN)$가지 경우만 해보면 됩니다.

- 과제를 완료하는 시각마다, 기한이 지나지 않은 과제들 중 기한이 가장 이른 것을 하는 것이 최적입니다.
- 기한이 가장 이른 과제를 알아내기 위해 모든 과제를 확인한다면 최악의 경우 $\mathcal{O}(N^2)$ 의 시간이 걸립니다.
- 이때 전체 시간복잡도는 $\mathcal{O}(AN^3)$ 이며, 이것을 구현하면 경우에 따라 정답을 받을 수 있습니다.

E. 지금 자면 꿈을 꾸지만

- 과제를 완료한 k번째 시각에 기한이 지난 과제는 이후에도 완료할 수 없습니다.
- 따라서 과제를 기한이 이른 순으로 미리 정렬한 다음, **투 포인터**를 사용하여 $\mathcal{O}(N)$에 모든 확인을 완료할 수 있습니다.
- 이때 전체 시간복잡도는 $\mathcal{O}(AN^2)$입니다.

- 다른 접근: 만약 과제 P개를 완료할 수 있다면, 기한이 가장 늦은 P개를 완료할 수 있습니다.
- 또한 과제 P개를 완료할 수 있다면 $P-1$개도 완료할 수 있습니다.
- 이를 이용해 완료할 과제의 개수에 대한 **매개 변수 탐색**을 하는 방법을 고려해볼 수 있습니다.
- P개의 과제 각각을 완료해야 한다는 조건이 M, X에 대한 선형 부등식으로 나타나며, 조건을 모두 만족하는 M, X가 존재하는지 판정하면 됩니다.
- 이때 시간복잡도는 $\mathcal{O}(N^2 \log N)$이 되어, **과제를 완료하는 데 걸리는 초기 시간에 의존하지 않습니다.**

F. 두배

`dijkstra`, `Z`

출제진 의도 – **Hard**

- 제출 117번, 정답 15팀 (정답률 12.821%)
- 처음 푼 팀: **사상 첫 포핏**, 42분
- 출제자: `mingyu331`
- 가장 짧은 정해: 1,306B^{C++}, 2,655B^{Python}

- 마지막 글자가 아닌 한번의 입력을 하였을 때, 기존 문자열 M은 입력 후 문자열 M'의 접두사입니다.
- 현재 M의 i번째 글자가 T와 다르다고 한다면 M의 i번째 글짜까지 삭제를 한 후에 추가적인 연산을 해야 합니다.
- 즉, 연산 후에 M이 T의 접두사가 되도록 해야 합니다.

T의 접두사들인 M에 대해서 연산을 한 후 도달하는 T의 접두사 M'을 구해봅시다.
입력을 통해 "두 배"가 일어나는 경우

- 연산 후 M'의 길이는 T와 $M+M+c$의 최장 공통 접두사의 길이 $\mathrm{lcp}(T, M+M+c)$ 입니다.
- M이 T의 접두사이므로 $\mathrm{lcp}(T, M+M+c) = |M| + \mathrm{lcp}(T[|M|..], M+c)$ 입니다.
- $\mathrm{lcp}(T[|M|..], M+c)$는 $\mathrm{lcp}(T[|M|..], M)$과 c, $T[2|M|]$을 통하여 구할 수 있습니다.
- M이 T의 접두사이니 $\mathrm{lcp}(T[|M|..], M)$은 $\min(\mathrm{lcp}(T[|M|..], T), |M|)$으로, $0 \le i < |T|$인 i에 대해 $\mathrm{lcp}(T[i..], T)$를 구해야 합니다.

- $\mathrm{lcp}(T[i..], T)$를 모든 $0 \le i < |T|$에 대해서 구하는 것은 Z 알고리즘을 통하여 $\mathcal{O}(|T|)$에 구할 수 있습니다.
- 필요한 연산의 수는 M의 길이가 l에서 l'으로 변화했다면 $(2l+1) - l' + 1$번의 입력이 필요합니다.

"두 배"가 일어나지 않는 경우

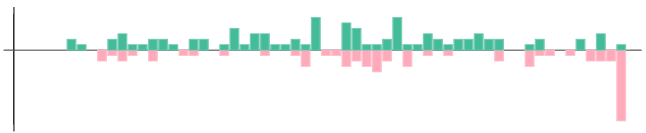
- 글자 c를 입력하는 경우 $c = T[|M|]$이면 $\mathrm{lcp}(T, M + c) = |M| + 1$이고, 아니면 $|M|$입니다.
- 이 경우 1 또는 2번의 연산이 필요합니다.
- M의 마지막 글자를 지우는 경우 M'의 길이는 $|M| - 1$이고, 1회의 연산이 필요합니다.

- 입력 전 길이 l에 해당되는 상태에서 l'에 해당되는 상태로 필요한 입력의 개수만큼의 가중치를 가지는 간선을 추가합시다. 입력 중 삭제도 가능함에 유의합시다.
- 정점은 총 $|T|+1$개, 간선은 약 $|\Sigma||T|$개의 그래프가 만들어집니다. 이 그래프에서 길이 0의 정점에서 길이 $|T|$의 정점까지의 최소 길이를 다익스트라 알고리즘으로 찾으면 이것이 T를 만들기 위한 최소의 입력 수입니다.
- 총 시간복잡도는 $\mathcal{O}\left(|\Sigma||T|\log|T|\right)$이고, 추가적인 최적화를 하면 $\mathcal{O}\left(|T|\right)$에 문제를 해결할 수도 있습니다.

G. 석고모형만들기

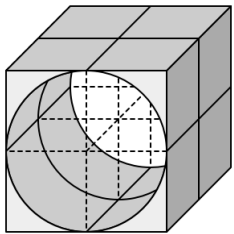
graph_traversal
출제진 의도 – **Medium**

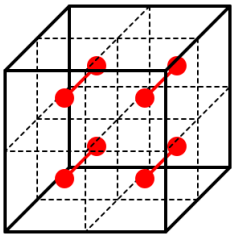
- 제출 153번, 정답 89팀 (정답률 58.170%)
- 처음 푼 팀: **개척단 훈련소**, 15분
- 출제자: queued_q
- 가장 짧은 정해: 694B^{C++}, 1,274B^{Python}



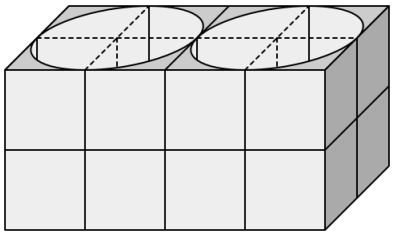
- 우선 단위 정육면체를 변의 길이가 $1/2$인 작은 정육면체 8개로 쪼갭시다.
- 두 이웃한 작은 정육면체 안의 석고 조각이 서로 연결되어 있다면, 둘 사이에 간선을 이어서 그래프를 만들 것입니다.

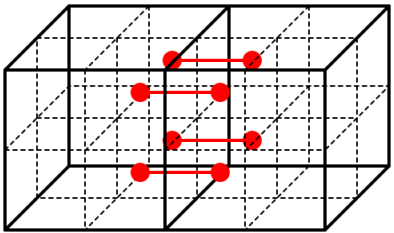
- 단위 정육면체 내에서 회전축과 평행하게 4개의 간선을 잇습니다.





– 단위 정육면체끼리 맞닿은 면 사이에 4개의 간선을 잇습니다.



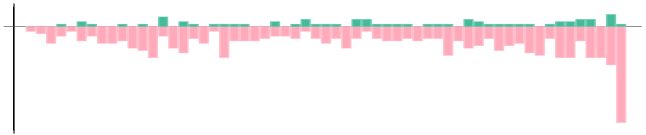


- 이와 같이 그래프를 구성하고 DFS 또는 BFS 등의 그래프 탐색을 통해 연결 컴포넌트의 개수를 세면 됩니다.
- 시간 복잡도는 $O(RC)$ 입니다.

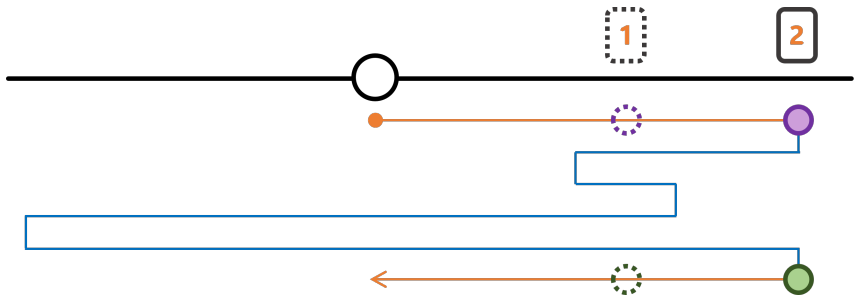
H. 만보기 대행 서비스

greedy, case-work
출제진 의도 – **Medium**

- 제출 513번, 정답 66팀 (정답률 12.865%)
- 처음 푼 팀: **while false**, 14분
- 출제자: functionx
- 가장 짧은 정해: 552B^{C++}, 280B^{Python}

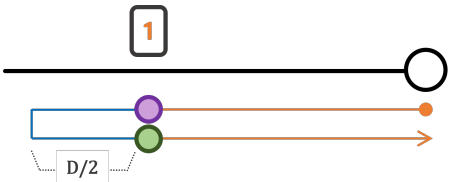


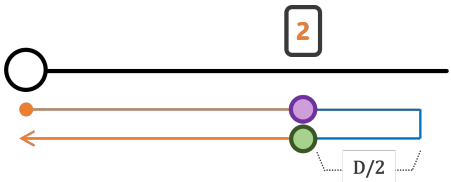
동쪽 및 서쪽 끝에 있는 휴대폰만 집고 복귀하는 전략을 짰다면, 나머지 휴대폰은 더 일찍 집고 더 늦게 반납할 수 있습니다.



양쪽 끝 휴대폰만 남기고 문제를 풀어 봅시다.

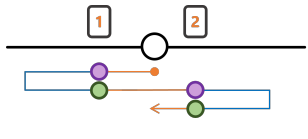
동쪽 끝에만 휴대폰이 있거나 서쪽 끝에만 휴대폰이 있으면, 아래 전략만 고려하면 됩니다.



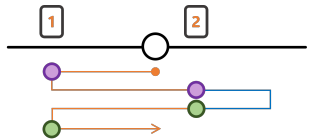


- 양 끝에 모두 휴대폰이 있으면 휴대폰을 집고 반납하는 순서에 따라 6가지 경우로 나눌 수 있습니다.
  1. 서쪽 휴대폰 입수 → 서쪽 휴대폰 반납 → 동쪽 휴대폰 입수 → 동쪽 휴대폰 반납
  2. 서쪽 휴대폰 입수 → 동쪽 휴대폰 입수 → 서쪽 휴대폰 반납 → 동쪽 휴대폰 반납
  3. 서쪽 휴대폰 입수 → 동쪽 휴대폰 입수 → 동쪽 휴대폰 반납 → 서쪽 휴대폰 반납
  4. 동쪽 휴대폰 입수 → 동쪽 휴대폰 반납 → 서쪽 휴대폰 입수 → 서쪽 휴대폰 반납
  5. 동쪽 휴대폰 입수 → 서쪽 휴대폰 입수 → 동쪽 휴대폰 반납 → 서쪽 휴대폰 반납
  6. 동쪽 휴대폰 입수 → 서쪽 휴대폰 입수 → 서쪽 휴대폰 반납 → 동쪽 휴대폰 반납

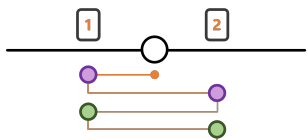
앞 3개 경우를 그림으로 나타내면 아래와 같습니다. 뒤 3개 경우는 앞 3개 경우와 좌우대칭입니다.



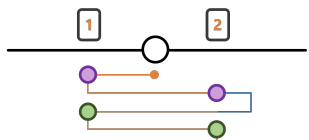
① +W ⇒ -W ⇒ +E ⇒ -E



② +W ⇒ +E ⇒ -E ⇒ -W



③ +W ⇒ +E ⇒ -W ⇒ -E



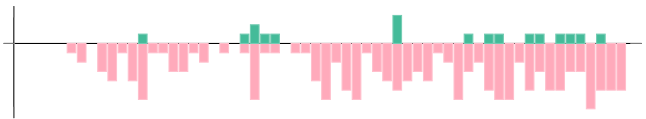
③ +W ⇒ +E ⇒ -W (추가 이동) ⇒ -E

입력 및 동/서쪽 끝 휴대폰을 구하는 데 $\mathcal{O}(N)$, 답을 구하는 데 $\mathcal{O}(1)$ 이 필요합니다.

I. 민들레

`data_structures`
출제진 의도 – **Hard**

- 제출 188번, 정답 18팀 (정답률 9.574%)
- 처음 푼 팀: **개척단 훈련소**, 36분
- 출제자: `sorohue`
- 가장 짧은 정해: 1,142B^{C++}, 5,381B^{Python}



- 인접한 화분에 심긴 민들레들을 하나의 민들레 무리로 부릅시다.
- 어떤 두 민들레 무리 사이에 다른 민들레가 없고, 사이에 민들레가 심기지 않은 화분이 K 개 있다면 두 민들레 무리가 인접해 있으며 그 거리가 K 라고 부릅시다.
- 어느 방향으로 바람이 불든 모든 민들레는 한쪽으로 길이 1 만큼 팽창합니다.
- 따라서 바람이 불 때마다 민들레가 심긴 화분의 개수는 민들레 무리의 수만큼 늘어납니다.

- L, R은 기존의 민들레 무리를 팽창시키기만 하기 때문에, 민들레 무리는 오로지 C x를 통해서만 새로 생길 수 있습니다.
- 만약 어떤 민들레 무리가 다른 민들레 무리보다 서쪽에 있다면, 그 민들레 무리 안에서 처음으로 민들레가 심긴 화분은 다른 민들레 무리에서 처음으로 민들레가 심긴 화분보다 서쪽에 있을 것입니다.
- 따라서 C x로 주어지는 모든 화분의 위치를 저장해두면, 이를 이용해 각각의 민들레 무리의 위치를 인덱싱할 수 있습니다.
- 이때 각 민들레 무리를 대표할 수 있는 화분의 번호를 중심 화분이라고 부릅시다.

- 바람이 불 때마다 모든 민들레 무리의 서쪽 끝 화분과 동쪽 끝 화분의 위치를 업데이트하는 것은 비효율적입니다.
- 각 민들레 무리를 (중심 화분의 번호, 중심 화분에서 민들레 무리의 서쪽 끝 화분까지의 거리, 중심 화분에서 민들레 무리의 동쪽 끝 화분까지의 거리)로 표현할 수 있습니다.
- 이때 각 방향으로의 끝 화분까지의 거리 대신 그 값을 나타낼 수 있는 **오프셋**을 저장합시다.
- 오프셋과 현재까지 각 방향으로 바람이 분 총 횟수를 더했을 때 실제 민들레 무리의 양 끝 화분 번호가 나오도록 오프셋을 잡으면, 바람이 부는 이벤트를 각 방향으로 바람이 분 횟수만을 업데이트하는 것으로 표현할 수 있습니다.

- 이제 현재 존재하는 민들레 무리들 자체의 값을 변경해야 하는 상황은 새로운 민들레가 생기거나 두 민들레 무리가 합쳐지는 경우밖에 없습니다.
- 이러한 경우는 각각 많아봐야 중심 화분으로 가능한 화분의 수 이하이므로, 각각 $\mathcal{O}(Q)$ 번 발생할 수 있습니다.

- 두 인접한 무리 사이의 간격을 무리의 양 끝을 관리하는 것처럼 오프셋을 이용해 관리해 줍시다. 두 방향으로 바람이 분 횟수의 총합이 오프셋 이상이라면 두 무리를 합쳐줄 수 있습니다.
- 우선순위 큐와 같이 정렬성이 보장되는 자료 구조를 이용해 인접한 두 무리의 오프셋을 관리해 주면 언제 어느 무리를 합쳐야 할지를 전체 쿼리를 처리하는 동안 $\mathcal{O}\left(N \log N\right)$에 알아낼 수 있습니다.
- 인접한 두 무리를 합치는 것은 중심 화분을 기준으로 분리 집합 등을 활용하면 빠르게 처리할 수 있습니다.
- 무리를 합칠 때 각 방향으로의 오프셋을 잘 업데이트해 주어야 합니다.

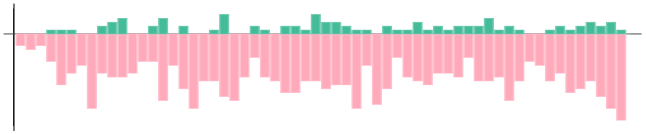
- 이제 C x 쿼리에 의해 민들레가 새로 심기는 경우를 처리해야 합니다.
- `std::set` 등을 이용해 현재 존재하는 민들레 무리들을 중심 화분의 위치를 기준으로 정렬해 저장하면, 새 민들레를 심을 중심 화분과 인접한 두 민들레 무리를 $\mathcal{O}(\log N)$에 찾을 수 있습니다.
- 이미 해당 화분에 민들레가 심겨 있는 경우는 무시하고, 비어있는 화분이라면 해당 화분을 중심으로 하는 민들레 무리를 새로 만들어 저장해 준 뒤에 인접한 무리끼리 합치는 작업을 처리해 줍시다.
- 이렇게 하면 항상 존재하는 서로 다른 민들레 무리가 1 이상의 거리를 두고 존재하도록 관리할 수 있습니다.
- 이상의 내용을 잘 구현하면 총 시간 복잡도 $\mathcal{O}(N \log N)$에 문제를 해결할 수 있습니다.

J. 동전 쌍 뒤집기

`ad-hoc`, `prefix-sum`, `stack`

출제진 의도 – **Hard**

- 제출 798번, 정답 89팀 (정답률 11.153%)
- 처음 푼 팀: **개척단 훈련소**, 11분
- 출제자: `bubbler`
- 가장 짧은 정해: 396B^{C++}, 407B^{Python}



- 먼저, 모든 조작은 홀수 번째와 짝수 번째 자리의 T의 개수를 둘 다 1 증가시키거나, 둘 다 1 감소시킴을 알 수 있습니다.
- 따라서 둘이 서로 다르면 답은 불가능입니다.

- 주어진 문자열의 길이 i 인 prefix에서 홀수 번째 자리의 T의 개수와 짝수 번째 자리의 T의 개수의 absolute difference를 a_i 라고 합시다.
- 수열 $a_1, a_2, \cdots, a_n$ 에 대해 다음의 사실을 관찰할 수 있습니다.
  - 왼쪽에서 i 번째와 $i+1$ 번째 동전을 뒤집으면 a_i 의 값이 1 증가 또는 1 감소하고 나머지는 변하지 않습니다.
  - a_i 와 a_{i+1} 의 차이는 1 이하입니다. 또한, a_1 과 a_{n-1} 은 1 이하이며, a_n 은 0 입니다.

- 이제 a_i 들 중에서 가장 큰 값 중 가장 왼쪽에 있는 원소를 a_k 라고 합시다.
- T 동전이 존재한다고 가정하면 a_k 의 값은 양수입니다.
- a_{k-1} 이 존재한다면, 이 값은 $a_k - 1$ 과 같습니다. 즉, 현재 k 번째 동전은 T입니다.
- 또한, k 번째에 있는 T 동전으로 인해 값이 증가하였으므로, 길이 k 의 prefix에는 k 와 홀짝성이 같은 T 동전이 더 많은 상황입니다.

- 이때 a_{k+1} 은 두 가지 가능성이 있습니다.
  - $a_{k+1} = a_k - 1$ 이라면, $k+1$ 번째 동전 또한 T입니다. 따라서 k 번째와 $k+1$ 번째 동전을 뒤집어서 a_k 를 1 감소시킬 수 있습니다.
  - $a_{k+1} = a_k$ 라면, $k+1$ 번째 동전은 H입니다. $a_{k+1} \neq 0$ 이므로 $k+1 < n$ 이고, 따라서 $k+2$ 번째 동전이 존재합니다. $k+2$ 번째 동전이 T이면 $a_{k+2} > a_k$ 가 되어 모순이므로, $k+2$ 번째 동전 또한 H입니다. 이제 $k+1$ 번째와 $k+2$ 번째 동전을 뒤집으면 k 와 $k+1$ 의 홀짝성이 다르므로 a_{k+1} 이 1 감소합니다.

- 따라서 임의의 주어진 상황에서 반드시 1회의 조작으로 $\sum_{i=1}^{n} a_i$를 1 감소시킬 수 있습니다.
- 같은 방법을 반복하여 모든 a_i를 0으로 만들 수 있고, 이것보다 빠르게 $\sum_{i=1}^{n} a_i$를 감소시킬 수 없으므로, 입력 문자열의 $\sum_{i=1}^{n} a_i$의 값이 정답이 됩니다.

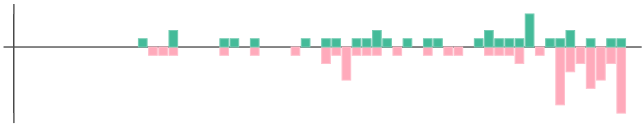
- 앞의 풀이 이외에도 다양한 방법으로 풀 수 있습니다.
- 별해 1: 홀수 자리의 T와 짝수 자리의 T 사이에 다른 T가 없으면 두 인덱스의 차이만큼의 조작으로 둘을 제거할 수 있음을 이용하여, 괄호 문자열 매칭과 비슷한 알고리즘(홀-짝, 짝-홀 모두 허용)을 사용해서 각 매칭의 거리의 합을 구하면 같은 답을 얻습니다.
- 별해 2: 주어진 입력에서 모든 짝수 번째 동전을 뒤집어서 생각해 보면, 뒷면이 보이는 동전이 $\left\lfloor \frac{n}{2} \right\rfloor$ 개가 주어지고 HT 또는 TH의 쌍을 뒤집을 수 있을 때, 뒷면이 보이는 동전을 모두 짝수 번째에 위치하도록 하는 문제로 바꿀 수 있습니다. HT나 TH를 뒤집는 것은 뒷면 동전을 한 칸 움직이는 것과 같으므로, 뒷면 동전의 순서를 바꾸지 않고 각 동전이 도달해야 하는 목적지까지의 거리를 모두 합해주면 답이 됩니다.

K. 나무 심기

`constructive`, `ad-hoc`, `case-work`

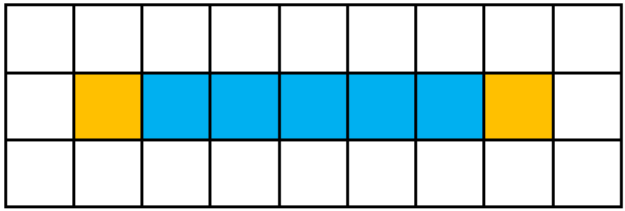
출제진 의도 – **Hard**

- 제출 91번, 정답 34팀 (정답률 37.363%)
- 처음 푼 팀: **jjmik12321**, 37분
- 출제자: `molamola`
- 가장 짧은 정해: 1,004B^{C++}, 1,463B^{Python}

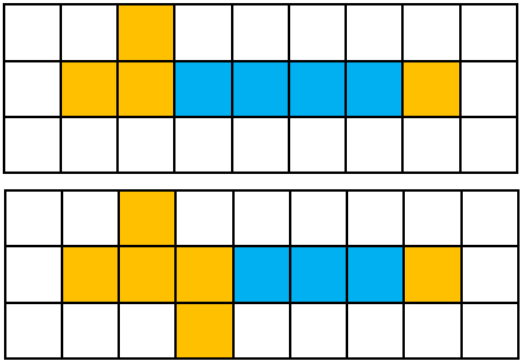


- 임의의 그래프에서, (모든 정점의 차수의 합) $= 2 *$ (총 간선 수) $=$ 짝수 입니다.
- B 가 홀수라면 모든 정점의 차수의 합이 홀수가 되어 배치가 불가능합니다.

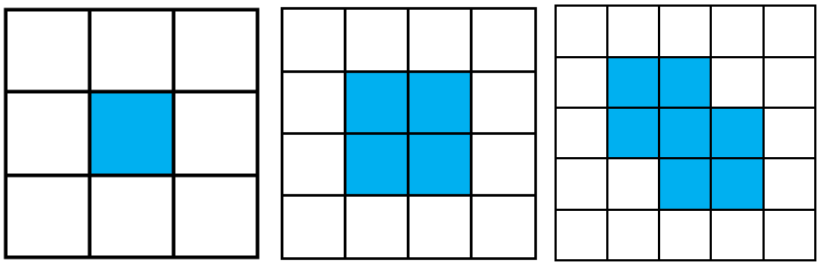
$B = 2$ 일 때 아래와 같이 배치하면 됩니다. 노란색이 복숭아나무, 파란색이 사과나무 입니다.



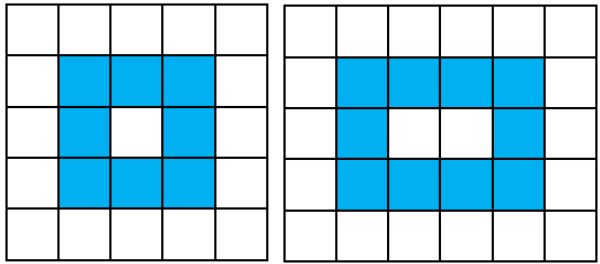
$B = 4, 6, \ldots$ 일 때 아래와 같이 배치하면 됩니다.



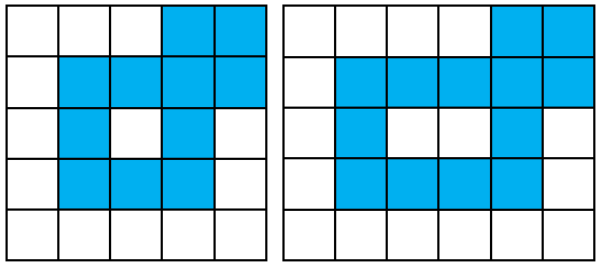
$B = 0$은 다소 까다롭습니다. $A = 1, 4, 7$을 아래와 같이 처리합니다.



$A = 8 + 2k (k \geq 0)$를 아래와 같이 배치합니다. k에 따라 가운데를 적당히 늘리면 됩니다.



$A = 11 + 2k (k \geq 0)$ 를 아래와 같이 배치합니다. k에 따라 가운데를 적당히 늘리면 됩니다.



- 그 외의 경우인 $A = 2, 3, 5, 6, 9$는 불가능함을 보일 수 있습니다.