SciOI 2025 에디토리얼

Official Solutions

by

서울과학고 컴퓨터과학연구부 SciCom

2025년 12월 27일

Problems

문제		의도한 난이도	출제자
A	SSHS 문자열	Easy	kdy40929
B1	낚시 (Easy)	Easy	luca200799k
B2	낚시 (Hard)	Medium	phoenixrafael
C	Longest Common Path	Medium	phoenixrafael
D	맥스웰의 수수께끼 악마	Hard	luca200799k
E	Protect The SSHS	Hard	kdy40929
F	꽃을 밟지 마세요	Hard	pk661
G	SSHS Demon Hunters	Hard	phoenixrafael
H	급식실 식탁 배정	Challenging	mingyu331
I	사랑은 괄호를 타고	Challenging	phoenixrafael
J	열도	Challenging	pk661
K	Bloom in Two	Challenging	mingyu331

SciOI 2025





SciOI 2025

- ✓ **운영진**
  - kdy40929
  - phoenixrafael
  - mingyu331
  - pk661
  - luca200799k
- ✓ **검수진**
  - kuhyaku
  - yijw0930
  - utilforever
  - realpsdoingdamyoo
  - octane
  - mj1000j
  - mhy908
  - ncy09

A. SSHS 문자열

`string`, `constructive`

출제진 의도 – **Easy**

- ✓ 제출 304번, 정답 160명 (정답률 61.513%)
- ✓ 처음 푼 사람: **pyb1031**, 0분
- ✓ 출제자: kdy40929

A. SSHS 문자열

- SSHS와 SHSS의 등장 횟수의 합을 최대화해야 합니다.
- 길이 N의 문자열에 포함되는 길이 4의 부분 문자열은 $N-3$개입니다.
- 한 칸 간격으로 길이 4의 부분 문자열을 3개 고르면 이들 모두가 SSHS 또는 SHSS가 될 수는 없습니다.
- 따라서 부분 문자열 중 SSHS와 SHSS의 개수 합은 최대 $N-3-\lfloor \frac{N-3}{3} \rfloor$ 입니다.
- 이를 만족하는 실례는 SSHSSHSSH... 입니다.

B1. 낚시 (Easy)

`ad_hoc`, `greedy`

출제진 의도 – **Easy**

- ✓ 제출 160번, 정답 45명 (정답률 28.750%)
- ✓ 처음 푼 사람: **79brue**, 5분
- ✓ 출제자: `luca200799k`

B1. 낚시 (Easy)

- 현재 당신에게 F 개의 물고기가 있다고 가정합시다.
- 낚시와 떡밥 만들기를 진행하며, (보유한 물고기의 개수)+(보유한 떡밥의 개수)는 항상 단조 감소합니다.
- 초기에는 보유한 물고기의 개수가 F, 보유한 떡밥의 개수는 0 개입니다.
- 따라서, 모든 낚시를 진행한 후 보유한 물고기의 개수의 최댓값은 F 개입니다.
- 만약 낚시터와 당신의 손에 있는 모든 물고기들 중 가장 가치가 높은 F 개의 물고기를 얻을 수 있다면, 해당 전략이 최선의 전략입니다.

- 다음과 같은 탐욕적인 전략을 생각합시다 :

항상 손에 있는 가장 가치가 낮은 물고기를 골라 떡밥으로 만들어 낚시를 진행합니다.
손에 가장 가치가 높은 F 개의 물고기가 남을 때까지 반복합니다.

- 이때, 다음과 같은 사실을 어렵지 않게 증명할 수 있습니다,
  - 손에는 항상 F 개의 물고기가 있게 됩니다. 즉, 낚시를 진행할 때마다 새로운 물고기가 잡힙니다.
  - 가치가 가장 높은 F 개의 물고기는 위 전략으로 떡밥으로 만들어 버려지지 않습니다.
  - 낚시터에 있는 물고기의 개수는 낚시를 진행할 때마다 감소합니다.

- 낚시터에 있는 물고기가 모두 없어졌다면, 가장 가치가 높은 F 개의 물고기들은 위 전략으로 버려지지 않기 때문에, 손에 있는 F 개의 물고기는 가장 가치가 높은 F 개의 물고기입니다.
- 낚시터에 있는 물고기가 모두 없어지기 전에 전략이 종료되었다면, 전략의 종료 조건에 의해 손에 있는 F 개의 물고기는 가장 가치가 높은 F 개의 물고기입니다.
- 따라서, 위 전략은 항상 낚시터와 당신의 손에 있는 모든 물고기들 중 가장 가치가 높은 F 개의 물고기를 얻을 수 있습니다. 즉, 최선의 전략입니다.

- 하지만, 문제 상황은 조금 다릅니다. 시작할 때 당신의 손에는 F 개의 물고기가 아닌 N 개의 떡밥이 있습니다.
- 이러한 경우에는 앞서 찾은 탐욕적인 전략을 약간 변형하여 물고기의 가치의 합을 최대로 얻을 수 있습니다.

먼저, N 개의 떡밥을 하나씩 사용하여 물고기를 낚습니다.
이때, 떡밥을 사용할 순서는 임의로 정해도 무방하며,
떡밥을 던져서 물고기를 낚지 못하더라도 괜찮습니다.
이후에는 항상 손에 있는 가장 가치가 낮은 물고기를 골라 떡밥으로 만들어 낚시를 진행합니다.
손에 가장 가치가 높은 물고기들만이 남을 때까지 반복합니다.

- 이러한 greedy 가 왜 성립할까요?

B1. 낚시 (Easy)

- 모든 행동이 끝났을 때 손에 떡밥이 아무 것도 없는 최적의 전략이 존재합니다. (떡밥이 남아 있다면, 해당 떡밥을 모두 버려도 물고기의 가치의 합이 줄어들지 않기 때문입니다.)
- 따라서, 모든 행동이 끝난 뒤, 떡밥을 모두 사용하는 전략만을 고려해도 무방합니다.
- 언급했듯 $S =$(보유한 물고기의 개수)+(보유한 떡밥의 개수)는 항상 단조감소합니다. 하지만, 물고기가 아닌 떡밥을 가지고 있는 경우에는 S의 값을 완전히 감소시키지 않는 것은 때에 따라서 불가능합니다.
- 하지만, S의 값을 최대한 크게 만드는 것은 유용합니다. 따라서, "모든 떡밥을 사용한 이후에 S의 값이 가장 적게 감소하는 전략"을 생각합시다.

- 이때, 다음과 같은 사실을 증명할 수 있습니다.
  - 떡밥을 한 번에 여러 개 사용하여 낚시를 진행하는 것은 떡밥을 하나씩 낚시로 사용하는 것보다 S의 값을 적게 감소시키지 않습니다.
  - N개의 떡밥을 모두 사용하기 전에 다른 물고기를 떡밥으로 만들어 낚시를 진행하는 것은 S의 값을 적게 감소시키지 않습니다.
  - N개의 떡밥을 어떤 순서로 사용하든 간에, S의 값은 항상 일정하게 감소합니다.
- 즉, 떡밥 만들기를 진행하지 않고, N개의 떡밥을 하나씩 모두 사용하여 낚시를 진행하는 것이 S의 값을 가장 적게 감소시키는 방법입니다. 이때, 떡밥을 사용하는 순서는 중요하지 않습니다.

- ✓ 이때, N 개의 떡밥을 모두 던진 이후에는 줄어든 최종 $S = S_f$ 값에 따라 S_f 개의 물고기를 가지게 됩니다. 이때, 이후 탐욕적인 방법을 통해 S_f 개의 가장 큰 물고기들을 얻을 수 있음을 앞서 보였습니다.
- ✓ 어떤 전략이 존재해서 S' 개의 물고기를 최종적으로 얻게 되었다고 해봅시다. 이때, $S' \leq S_f$ 이기 때문에, 그 어떤 전략도 가장 큰 S_f 개의 물고기의 가치의 합보다 더 큰 물고기의 가치 합을 얻을 수 없습니다.
- ✓ 따라서, 그 어떤 전략도 처음에 N 개의 떡밥을 모두 던진 뒤, 얻은 S_f 개의 물고기를 탐욕적으로 이용하여 S_f 개의 가장 큰 물고기를 얻는 전략보다 물고기의 가치의 합을 크게 만들 수 없습니다. 즉, 이것이 최선의 전략입니다.

B1. 낚시 (Easy)

- ✓ S_f 의 값은 정렬을 통해 $O((N+M)\log(N+M))$ 에 계산할 수 있습니다.
  - 떡밥과 물고기를 전부 합쳐 그 가치를 토대로 오름차순으로 정렬합니다.
  - 떡밥을 어떤 순서로 사용하여도 무방하기 때문에, 떡밥을 제일 가치가 낮은것부터 오름차순으로 던진다고 해봅시다.
  - x 를 지금까지 던진 후 물고기가 잡히지 않은 떡밥의 수, y 를 얻은 물고기의 수라고 합시다.
  - 가장 가치가 낮은 것부터 탐색하며, 떡밥이 나올 때마다 x 의 값을 1 증가시킵니다. 물고기 나올 때마다 x 의 값을 1 감소시키고 y 의 값을 1 증가시킵니다. 물고기가 나왔지만 $x=0$ 인 경우, 해당 물고기는 잡을 수 없습니다. y 의 값을 증가시키지 않습니다.
  - 모든 물고기와 떡밥을 고려하고 난 뒤, y 의 값이 구하고자 하는 S_f 의 값입니다.
- ✓ 이후 정렬을 통해 $O(M\log M)$ 에 물고기들의 가치만을 정렬한 뒤, 가장 큰 S_f 의 합을 구하면 구하고자 하는 답입니다.

B2. 낚시 (Hard)

`ad_hoc`, `greedy`, `segtree`, `tree_set`

출제진 의도 – **Medium**

- ✓ 제출 172번, 정답 30명 (정답률 18.023%)
- ✓ 처음 푼 사람: **79brue**, 11분
- ✓ 출제자: `luca200799k`, `phoenixrafael`

B2. 낚시 (Hard)

- **B1** 문제에서 찾은 최적의 전략을 실제로 구현하면 되는 문제입니다.
- 하지만, naïve하게 구현하고자 한다면, 낚시터에 남아 있는 물고기, 손에 있는 물고기, 그리고 가지고 있는 떡밥을 빠르고 효율적으로 관리하는 것이 쉽지 않습니다.
- 실제로 낚시터에 남아 있는 물고기, 손에 남아 있는 물고기, 가지고 있는 떡밥을 적당한 자료구조로 관리한다면, 원소의 제거/추가, 원소들 중 q보다 큰 가장 작은 원소 찾기 등의 연산을 상수 시간 혹은 로그 시간에 수행할 수 있는 자료구조를 구현해야 합니다.
  - 이를 구현할 수 있는 적당한 자료구조는 segment tree, fenwick tree, tree set, priority queue 등이 있습니다.
  - 위 자료구조 중 어떤 것을 사용하더라도 AC를 받을 수 있음을 확인하였습니다.

- 하지만, **B1** 문제에서 설명한 알고리즘을 발전시키면, 복잡한 자료구조 없이도 충분히 빠르게 최적의 전략을 찾을 수 있습니다.
- 떡밥과 물고기를 전부 합쳐 그 가치를 토대로 오름차순으로 정렬합니다.
- x를 지금까지 던진 후 물고기가 잡히지 않은 떡밥의 수라고 합시다.
- 가장 가치가 낮은 것부터 탐색하며, 떡밥이 나올 때마다 x의 값을 1 증가시킵니다. 물고기 나올 때마다 x의 값을 1 감소시키고, 해당 물고기는 '초기에 가지고 시작하는 물고기'로 표시합니다. 물고기가 나왔지만 $x = 0$인 경우, 해당 물고기는 잡을 수 없으므로 건너뜁니다,
- '초기에 가지고 시작하는 물고기'의 개수를 S_f라고 합시다.

B2. 낚시 (Hard)

- 먼저 모든 떡밥에 대해 낚시(`fish`)를 진행합니다.
- 물고기 카운터 $cnt = 0$으로 설정하고, 물고기 리스트를 오름차순으로 $f_1 \leq f_2 \leq \cdots \leq f_M$과 같이 순회합니다.
- 물고기 f_i가 '초기에 가지고 시작하는 물고기'라면, 물고기 카운터 cnt를 1 증가시킵니다.
- 물고기 f_i가 '초기에 가지고 시작하는 물고기'가 아니라면, f_i를 현재 가지고 있는지 판단합니다. f_1부터 f_i까지 수 중에 f_i가 현재 $multi$번째로 등장했다고 할 때, $multi \leq cnt$라면 f_i는 현재 가지고 있는 물고기입니다.
- 현재 가지고 있는 물고기라면, 일단 무조건 떡밥으로 만들어서 낚시를 진행합니다.

- ✓ 위와 같이 구현한다면, $O((N+M)\log(N+M)+M\log M)$에 정렬을 한 뒤, $O(N+M)$에 전략을 구성할 수 있습니다.
- ✓ 하지만, '가장 큰 S_f 개의 물고기'만을 가지고 있다면 종료한다는 조건을 반영하지 않았습니다.
- ✓ 위 알고리즘에서 떡밥으로 만들어 낚시에 사용한 물고기들 중, 마지막 S_f 개는 '가장 큰 S_f 개의 물고기'였습니다.
- ✓ 따라서, 위에서 진행한 모든 행동 중, 마지막 $2S_f$ 개를 제거한 뒤 출력하면 AC를 받을 수 있습니다.

SciOI 2025

C. Longest Common Path

`trees`, `graph_traversal`, `bfs`

출제진 의도 – **Medium**

- ✓ 제출 132번, 정답 36명 (정답률 27.273%)
- ✓ 처음 푼 사람: **hyperbolic**, 20분
- ✓ 출제자: `phoenixrafael`

C. Longest Common Path

- 두 경로가 모두 트리에 포함되려면, 경로를 이루는 간선이 모두 두 트리의 간선의 교집합에 포함되어야 합니다.
- 두 트리의 정점을 정점으로 가지며, 두 트리의 간선을 교집합한 것을 간선으로 가지는 그래프는 포레스트(Forest)입니다.
- 포레스트의 각각의 연결성분은 트리이므로, 각각의 트리에서 가장 긴 단순 경로를 찾으면 됩니다.
- 따라서, hashset 등을 이용하여 $O(N)$에 두 트리의 간선을 교집합한 뒤, 생긴 연결 성분 T_i 각각에 대해 트리의 지름을 $O(sizeof(T_i))$에 찾습니다.
- 트리의 지름 중 길이가 최대인 것을 출력합니다. 이때, 간선의 교집합이 존재하지 않는다면 -1을 출력합니다.

D. 맥스웰의 수수께끼 악마

`greedy`, `stack`, `constructive`

출제진 의도 – **Hard**

- ✓ 제출 74번, 정답 15명 (정답률 20.270%)
- ✓ 처음 푼 사람: **79brue**, 36분
- ✓ 출제자: `luca200799k`

- ✓ 상온인 블록이 없는 경우만 먼저 고려하겠습니다.
- ✓ 연속한 저온 블록들과 연속한 고온 블록들을 각각 덩어리라고 합시다.
- ✓ 덩어리 기준으로는 저온과 고온이 반복되어 배치됩니다.
- ✓ 이때 4개 이상의 덩어리를 한꺼번에 처리할 필요가 없습니다.
- ✓ 왜냐하면 4개 이상의 덩어리에서 저온 블록의 크기 합이 고온 블록의 크기 합보다 크다면 이 안에 저온 블록의 크기 합이 더 큰 덩어리가 내부에 존재합니다.
- ✓ 따라서 최대 3개의 덩어리만 한 번에 처리해도 충분합니다.

- 저온 블록은 항상 덩어리 단위로 처리해도 유효합니다.
- 앞쪽부터 순서대로 스택에 넣으면서 가장 바깥쪽의 2개 혹은 3개에 대해 저온 블록의 크기 합이 고온 블록의 크기 합이 더 큰지 비교합니다.
- 만약 저온 블록의 크기 합이 더 작다면 이를 전부 합치고 하나의 저온 덩어리로 만듭니다.
- 이를 반복하여 마지막에 가장 큰 저온 블록 하나만 남는다면 그 방법을 출력하고, 아니라면 -1을 출력합니다.
- 따라서 문제를 $O(N)$에 해결할 수 있습니다.

SciOI 2025

E. Protect The SSHS

`dijkstra`, `segtree`, `binary_search`

출제진 의도 – **Hard**

- 제출 58번, 정답 15명 (정답률 25.862%)
- 처음 푼 사람: **79brue**, 54분
- 출제자: kdy40929

- 다익스트라 알고리즘을 이용해 $O(MlogN)$ 에 정문과 후문에서 각 정점까지의 최단 거리를 구할 수 있습니다.
- 정문에서 정점 i 까지의 거리를 x_i, 후문에서 정점 i 까지의 거리를 y_i 라 합시다.
- 그러면 문제는 $min(ax_i, by_i)$ 가 최대가 되는 정점 i 를 찾는 문제가 됩니다.
- $x_i < x_j$ 이고 $y_i < y_j$ 이면 정점 i 는 항상 정점 j 보다 먼저 도달하게 되므로 요새가 될 수 없습니다.

- ✓ x 값 혹은 y 값을 기준으로 정렬한 후 스택을 이용하면 요새가 될 수 있는 후보 정점들만 $O(NlogN)$ 에 남길 수 있습니다.
- ✓ 이 정점들을 x 좌표 오름차순으로 정렬하면 y 좌표 내림차순이 됩니다.
- ✓ 즉 뒤로 갈수록 ax_i 항은 증가하고 by_i 항은 감소하므로 이분 탐색을 이용하여 매 쿼리마다 $O(logN)$ 의 시간에 $min(ax_i, by_i)$ 가 최대가 되는 정점을 찾을 수 있습니다.
- ✓ 전체 시간복잡도는 $O((M+N+Q)logN)$ 입니다.

F. 꽃을 밟지 마세요

`dp` `tree`, `rerooting`, `combinatorics`

출제진 의도 – **Hard**

- ✓ 제출 32번, 정답 13명 (정답률 40.625%)
- ✓ 처음 푼 사람: **sorohue**, 14분
- ✓ 출제자: pk661

- ✓ 만약 사이클이 존재한다면, 항상 불가능함을 알 수 있습니다.
- ✓ 그러므로 $N \leq M$ 이라면 답은 0 이 됩니다.

- ✓ 이제 트리일 때만 생각하면 됩니다.
- ✓ a_N 은 리프여야 함을 알 수 있습니다.
- ✓ 또한, a_{N-1} 은 a_N 을 제거한 트리에서 리프가 되어야 합니다.
- ✓ 귀납적으로 a_{i-1} 은 a_i 을 제거한 트리에서 리프가 되어야 합니다.

- ✓ 순열 a에서 i가 등장하는 위치를 A_i라 합시다.
- ✓ 어떤 순열이 꽃을 밟지 않으려면, $A_i = 1$인 i를 루트로 하는 트리에서 부모의 A값이 자식의 A값보다 작아야 합니다.

- ✓ 이를 이용해 tree DP를 할 수 있습니다.
- ✓ 특정 루트를 고정했을 때 값을 $O(N)$에 계산해봅시다.
- ✓ $DP[i] = i$의 서브트리의 상대적인 A_i 관계를 정하는 경우의 수.
- ✓ 모든 루트에 대해서 계산하면 $O(N^2)$에 풀 수 있습니다.
- ✓ rerooting과 같은 방법을 이용하면 $O(N)$에 모든 루트에 대해 답을 구할 수 있습니다.

G. SSHS Demon Hunters

`segtree`, `lazyprop`, `sweeping`, `coordinate_compression`
출제진 의도 – **Hard**

- ✓ 제출 20번, 정답 10명 (정답률 50.000%)
- ✓ 처음 푼 사람: **79brue**, 83분
- ✓ 출제자: `phoenixrafael`

- (X_i, Y_i)에 헌터를 배치하여 없앨 수 있는 균열은 $(1, Y_i)$에 헌터를 배치하더라도 제거할 수 있습니다. 따라서, 모든 헌터를 $x = 1$ 상에 배치한다고 생각해도 문제가 없습니다.
- 각각의 균열에 대해 (x, y)에 있는 균열을 $(1, Y_i)$에 있는 헌터를 이용하여 처리하기 위해서는 $y - x + 1 \leq Y_i \leq y + x - 1$이어야 합니다.
- 즉, (x, y)에 있는 균열은 구간 $[y - x + 1, y + x - 1]$에 적어도 한 개의 헌터가 있어야 처리할 수 있습니다. 이때, $y - x + 1 < 1$인 경우와 $y + x - 1 > N$인 경우는 구간의 왼쪽 끝과 오른쪽 끝을 각각 $1, N$으로 생각합니다.
- 모든 균열을 해당하는 구간으로 생각한다면, N개의 구간에 대해 3개의 정수를 골라 최대한 많은 구간이 정수를 한 개라도 포함도록 하는 문제로 귀결됩니다.

G. SSHS Demon Hunters

- 헌터 $(1, Y_1)$, $(1, Y_2)$, $(1, Y_3)$에 헌터를 배치한다고 합시다. $1 \le Y_1 \le Y_2 \le Y_3 \le N$입니다.
- 정렬을 통해 N개의 구간의 y좌표를 $O(N \log N)$에 압축한 뒤, 느리게 갱신되는 세그먼트 트리를 이용하여 각각의 y좌표를 포함하는 구간의 수를 기록합니다.
- 헌터 Y_2를 결정하고, Y_2로 인해 처리되는 구간을 세그먼트 트리에서 제외합니다. Y_2의 값을 1부터 N까지 스위핑하며, 구간 입장/퇴장 시 해당하는 구간에 포함되는 y좌표 전체에 1을 더하고 빼는 쿼리를 던지는 것으로 이것을 $O(\log N)$에 구현할 수 있습니다.
- 세그먼트 트리에서 $[1, Y_2 - 1]$와 $[Y_2 + 1, N]$ 구간에 최댓값 쿼리를 던져, 해당 Y_2 위치에 대해 처리되는 균열의 수가 최대가 되도록 Y_1과 Y_3의 위치를 $O(\log N)$에 계산할 수 있습니다.
- 모든 $O(N)$개의 Y_2 값에 대해 이것을 반복하여 처리되는 균열 수의 최댓값을 $O(N \log N)$에 계산합니다.

H. 급식실 식탁 배정

`sQrt_decomposition`, `segtree`

출제진 의도 – **Challenging**

- 제출 37번, 정답 7명 (정답률 18.918%)
- 처음 푼 사람: **79brue**, 153분
- 출제자: `mingyu331`

- ✓ 다음과 같이 배정하면 최소 개수의 식탁으로 모든 인원을 배정할 수 있음을 증명할 수 있습니다:
- ✓ 가장 낮은 선호도를 X 라고 합시다. 선호도가 가장 낮은 X 명의 사람들을 한 식탁에 배정합니다. 이를 모든 사람을 배정할 때 까지 반복합니다.
- ✓ 서브태스크 1. 이 전략을 자유롭게 $\mathcal{O}(Q^2)$ 에 구현하면 해결할 수 있습니다.
- ✓ 서브태스크 2. X 의 최대값이 작기 때문에 이를 활용할 수 있습니다. 어떤 X 에 대하여 선호도 X 를 가진 모든 사람을 배정하는 행동을 상수 시간에 진행할 수 있습니다. 이를 모든 X 에 대해 증가 순서대로 반복하면 $O(50 \cdot Q)$ 시간에 문제를 해결할 수 있습니다.

- ✓ 서브태스크 3. X 의 최소값이 크기 때문에 각 식탁에는 $20\,000$ 명 이상의 사람이 착석하게 되고, 필요한 식탁의 개수는 10 개 이하입니다.
- ✓ 세그먼트 트리와 같은 적당한 자료구조에 X 들을 정렬된 상태로 저장하여 관리하면 이분탐색을 통하여 각 테이블에 들어가는 사람들의 배정을 $\mathcal{O}(\log Q)$ 에 진행할 수 있고, 고로 총 $\mathcal{O}(10Q \log Q)$ 에 문제를 해결할 수 있습니다.

- 서브태스크 4. X 가 B 이상, B 미만인 두 그룹으로 사람들을 분리합니다.
- B 미만인 그룹에 대해서는 서브태스크 2의 풀이를 적용하고, B 이상인 그룹에 대해서는 서브태스크 3의 풀이를 적용합니다. 이를 통해 $O(Q \log Q \cdot Q/B + Q \cdot B)$ 시간에 문제를 해결할 수 있습니다.
- B 를 대략 $\sqrt{Q \log Q}$ 로 잡으면 $\mathcal{O}(Q\sqrt{Q \log Q})$ 시간에 문제를 해결할 수 있습니다.
- 추가적인 테크닉을 통하여 서브태스크 3 부분의 시간 복잡도를 $\mathcal{O}(Q \cdot Q/B)$ 로 줄일 수 있고, 이로 인해 $\mathcal{O}(Q\sqrt{Q})$ 시간에도 문제를 해결할 수 있습니다.

I. 사랑은 괄호를 타고

`constructive`, `diffential_cryptanalysis`
출제진 의도 – **Challenging**

- ✓ 제출 18번, 정답 5명 (정답률 27.778%)
- ✓ 처음 푼 사람: **mickeyjung**, 213분
- ✓ 출제자: `phoenixrafael`

- ✓ 정답 S는 괄호 문자열이며, 괄호 문자열 Q_i를 질문하였을 때 Q_i의 N개의 문자 중 틀린 문자의 개수인 x_i를 알 수 있습니다.
- ✓ 괄호 문자열 조건이 없다면 해당 문제를 어떻게 풀었을까요?
- ✓ ((((...((라는 문자열을 질문한 뒤, 2N개의 문자 중 한 개씩을 {,), }로 변경해가며 질문합니다.
- ✓ 문자를 바꿈으로써 x_i의 값이 감소하였다면, 해당 위치에 해당 문자가 들어가야 함을 의미합니다.
- ✓ 위와 같은 쿼리를 통해 최대 $6N + O(1)$번의 질문에 문제를 해결할 수 있습니다.

- ✓ 쿼리가 괄호 문자열 조건을 만족하도록 쿼리를 바꿔봅시다.
- ✓ 앞서 알아본 방법과 같이 단 한 개의 문자만을 바꾸는 방법은 이용할 수 없습니다. 괄호 문자열에서 문자를 바꿔 다른 괄호 문자열을 만드려면 최소한 두 개의 문자를 바꾸어야 합니다.
- ✓ 어떤 괄호 문자열에서 ( ) 를 { } 로 바꾸는 것에 대해 생각해 봅시다.
- ✓ 모든 인접한 두 개의 자리 $i, i+1$ 에 대해 ( ) 에서 { } 로 바꾸었을 때 x 값의 변화량 Δx_i 를 알고 있다고 해봅시다.

- ✓ 정답이 괄호 문자열 조건을 만족하기 때문에, 다음과 같은 두 사실을 만족합니다. 이를 이용하면, Δx_i 값만을 이용하여 원본 문자열을 복원할 수 있습니다.
  - 인접한 두 개의 자리에 대해) } 와 { (가 등장하지 않습니다.
  - 처음에는 (혹은 { 중 하나인 여는 괄호가 등장합니다.
- ✓ Δx_i 의 값은 -2부터 2까지의 정수가 가능하며, 각각에 대해 아래와 같은 관찰이 가능합니다.
  - Δx_i 의 값이 2 또는 -2 라면, i 번째 자리와 $i+1$ 번째 자리는 각각 ( ) 또는 { } 입니다.
  - Δx_i 의 값이 1 또는 -1 이라면, i 번째 자리가 각각 (또는 { 이고 $i+1$ 번째에도 여는 괄호가 오거나, $i+1$ 번째 자리가 각각) 또는 } 이고 i 번째에도 닫는 괄호가 왔습니다.
  - Δx_i 의 값이 0 이라면, i 번째 자리에는 닫는 괄호, $i+1$ 번째 자리에는 여는 괄호가 왔습니다.

- ✓ 위와 같은 사실을 이용하여, 정답 문자열을 복원해봅시다.
- ✓ 각 위치에 여는 괄호가 와야 하는지, 닫는 괄호가 와야 하는지를 다음과 같이 판단할 수 있습니다.
  - 맨 처음에는 여는 괄호가 옵니다.
  - Δx_i 의 값이 1 또는 -1 이라면, i 번째 자리에 오는 괄호의 열림/닫힘 여부와 $i+1$ 번째 자리의 열림/닫힘 여부가 같습니다.
  - Δx_i 의 값이 0 이라면, i 번째 자리는 닫히는 괄호이며, $i+1$ 번째 자리는 열리는 괄호입니다.
  - Δx_i 의 값이 2 또는 -2 이라면, i 번째 자리는 열리는 괄호이며, $i+1$ 번째 자리는 닫히는 괄호입니다.

- ✓ 각 위치에 여는 괄호가 와야 하는지, 닫는 괄호가 와야 하는지를 결정했다면, 다음과 같이 어떤 괄호가 오는지 판단할 수 있습니다.
  - 만약 i 번째 위치에 여는 괄호가 온다면, Δx_i 의 값을 이용해 (와 { 중 하나로 결정할 수 있습니다.
  - 만약 i 번째 위치에 닫는 괄호가 온다면, Δx_{i-1} 의 값을 이용해) 와 } 중 하나로 결정할 수 있습니다.
- ✓ 따라서, 모든 자리에 어떤 문자가 오는 지 판단할 수 있습니다.

- ✓ Δx_i의 값은 다음과 같은 $2N+1$개의 쿼리를 통해 알 수 있습니다.
  - $Q_1 =$`()()()()...()`
  - Q_{i+1}=`()()()()...()`에서 $2i-1, 2i$번째 문자를 `{}`로 바꾼 문자열 $(1 \le i \le N)$
  - $Q_{N+2} =$`(()()()()...())`
  - Q_{i+N+2}=`(()()()()...())`에서 $2i, 2i+1$번째 문자를 `{}`로 바꾼 문자열 $(1 \le i \le N-1)$

J. 열도

`divide and conquer`, `bfs`, `centroid`

출제진 의도 – **Challenging**

- ✓ 제출 12번, 정답 1명 (정답률 8.333%)
- ✓ 처음 푼 사람: **jhwest2**, 250분
- ✓ 출제자: pk661

- ✓ 분할 정복을 사용합니다.
- ✓ 적당한 가로 분할선을 골라, 이 분할선을 지나는 쿼리를 처리합니다.

- 분할선에서 분할선에 수직인 방향으로 출발했을 때 각 정점에 도달하는 최소 거리를 $d(i, j)$ 라 합시다.
- 반대로 각 정점에서 $d(i, j)$ 코스트를 사용해 도달할 수 있는 분할선 위 점들은 구간으로 나타납니다.
- 이 구간을 $[L_{i,j}, R_{i,j}]$ 라 합시다.

- 어떤 쿼리 $(i_1, j_1) \to (i_2, j_2)$ 를 처리한다 생각합니다.
- 이때, 답은 $d(i_1, j_1) + d(i_2, j_2)$ 과 최대 1 차이납니다.
- $d(i_1, j_1) + d(i_2, j_2) - 1$ 일 조건은, $[L_{i_1,j_1}, R_{i_1,j_1}]$ 과 $[L_{i_2,j_2}, R_{i_2,j_2}]$ 가 교집합이 존재하는 것입니다.

- ✓ 나머지 경우도 비슷하게 처리할 수 있습니다.
- ✓ 각 정점마다 $d(i,j)+1$번 이하로 이동해서 분할선과 수직이게 들어갈 때 구간을 구하면 됩니다.
- ✓ 구간을 구하는 팁은, $L_{i,j}$와 $R_{i,j}$를 분리해서 각각 0-1 BFS로 구하면 됩니다.

- 적절한 분할선을 잡는게 중요합니다.
- (x, y) 와 $(x, y+1)$ 이 모두 육지라면, 두 육지를 묶어줍니다.
- 즉, 가로 막대들로 묶어줍니다.
- 인접한 막대들 사이에 간선을 추가하면, 이는 트리의 형태가 됩니다.
- 이 트리에서 센트로이드를 찾은 후, 그 정점에 대응되는 가로 막대를 분할선으로 사용하면 됩니다.
- 높이가 $O(\log(NM))$ 가 되므로, 전체 시간복잡도는 $O(NM \log(NM))$ 이 됩니다.

K. Bloom in Two

`ad_hoc`, `2-sat(?)`

출제진 의도 – **Challenging**

- 제출 27번, 정답 0명 (정답률 0.000%)
- 처음 푼 사람: -, -분
- 출제자: `mingyu331`

- ✓ 서브태스크 1. 모든 꽃을 E 방향으로 배정하여 해결할 수 있습니다.
- ✓ 서브태스크 2. 다음과 같이 배정할 수 있습니다.
- ✓ 왼쪽부터 오른쪽으로 순서대로 방향을 배정합니다. 현재 꽃이 배정하지 않은 꽃들 중 가장 낮은 y 좌표를 가지고 있다면, E로 배정합니다. 아닌 경우 C로 배정합니다.

- 서브태스크 3. 2-SAT 알고리즘을 활용합니다.
- 모든 꽃의 방향을 상하 중 어느 방향을 보는지에 대한 변수 하나, 좌우 중 어느 방향을 보는지에 대한 변수 하나로 표현합니다.
- 모든 화분이 놓이지 않은 칸에 대하여 이 칸을 볼 수 있는 두 방향에 해당하는 변수의 OR로 아름다울 조건을 표현할 수 있습니다.
- 이를 통해 변수 $2N$ 개, 조건 $N^2 - N$ 개의 2-SAT 문제로 문제를 환원할 수 있고, 이를 해결하면 $\mathcal{O}(N^2)$ 시간에 문제를 해결할 수 있습니다.

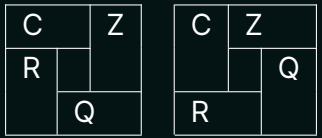
- ✓ 서브태스크 4. 2-SAT 풀이에서 크루스칼 알고리즘을 사용하는 경우, 그래프의 특성을 활용하여 탐색 과정을 $\mathcal{O}(N \log N)$ 으로 계산 과정을 최적화 할 수 있습니다.
- ✓ DFS 과정에서 현재 변수와 연결된 변수 중 방문하지 않은 변수가 있는지 판정을 하고, 있다면 해당 변수를 찾는 과정이 필요합니다.

- 세그먼트 트리를 활용해서 각 방향에 대하여 방문하지 않은 변수 중 x 좌표 구간에 대한 y 좌표 구간의 최대/최소를 계산하면 이를 해결할 수 있습니다.
- 예를 들어, 현재 변수와 연결된 방문하지 않은 밑 방향 변수들을 모두 찾으려면 현재 변수의 왼쪽에 위치하고, y 좌표가 최대인 방문하지 않은 밑 방향 변수와 현재 변수의 y 좌표를 비교하여 방문할 수 있는 변수가 있는지 체크하고, 존재하면 방문할 수 있습니다.
- 이를 통하여 모든 변수를 한번만 방문하게 되고, 총 $\mathcal{O}(N \log N)$ 시간에 문제를 해결할 수 있습니다.

- ✓ 서브태스크 5. 적당한 배치가 존재한다고 가정해 봅시다.
- ✓ 같은 방향으로 배치된 모든 화분은 E, Z의 경우 x 좌표가 증가할 때 y 좌표도 증가하는 순열, Q, C의 경우 x 좌표가 증가할 때 y 좌표가 감소하는 순열을 이루어야 합니다.

- ✓ 또한, 두 방향을 이루는 화분에 대하여 비슷한 논리를 적용하면 각 방향의 화분에 대하여 다음과 같은 관계가 나옵니다:
  - E, Z 방향의 화분은 증가 순열을 이루어야 합니다.
  - Q, C 방향의 화분은 감소 순열을 이루어야 합니다.
  - 모든 E 방향의 화분은 모든 Q 방향의 화분보다 더 **왼쪽**에 있어야 합니다.
  - 모든 C 방향의 화분은 모든 Z 방향의 화분보다 더 **왼쪽**에 있어야 합니다.
  - 모든 E 방향의 화분은 모든 C 방향의 화분보다 더 **밑**에 있어야 합니다.
  - 모든 Q 방향의 화분은 모든 Z 방향의 화분보다 더 **밑**에 있어야 합니다.
  - C 방향의 화분보다 왼쪽 밑에 존재하는 Q 방향의 화분은 존재하면 안됩니다.
  - R 방향의 화분보다 왼쪽 밑에 존재하는 Z 방향의 화분은 존재하면 안됩니다.

- 이를 정리하면 크게 다음 두 종류의 화분 및 방향 배치가 가능합니다.



- 또한, 이와 같이 화분과 방향이 배치된 경우 조건을 만족함을 보일 수 있어, 이와 같은 모양으로 배치되었음이 필요충분조건입니다.

- ✓ 어떤 수직, 수평 분할선이 존재하여 네 체인들이 2개 씩 양쪽으로 쪼개질 수 있다면, 이 경우에는 서브태스크 2와 비슷한 방식으로 네 방향에 대한 배정을 진행할 수 있습니다.
- ✓ 고로, 이 경우에는 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.

- ✓ 분할선이 없고 왼쪽 경우인 상황을 살펴봅시다. 오른쪽 경우에 대해서는 적당한 대칭을 통하여 동일하게 해결 가능합니다.
- ✓ 각 방향에 대한 우두머리 원소를 다음과 같이 정의합시다.
  - Q방향의 가장 왼쪽 위 원소
  - E방향의 가장 오른쪽 위 원소
  - Z방향의 가장 왼쪽 밑 원소
  - C방향의 가장 오른쪽 밑 원소
- ✓ 각 방향의 우두머리 원소를 구하면 이로 인해 네 방향의 배치 위치가 결정되게 되고, 결국 모든 화분의 배치를 확정할 수 있습니다.

- 우두머리 원소들을 확정하기 위해 다음과 같이 탐색을 할 수 있습니다:
- Q 체인의 우두머리 원소에 대하여 오른쪽 위에 있는 화분 중 가장 밑에 있는 화분을 선택하면 이는 Z 체인의 우두머리 원소입니다.
- 이를 반복하여, 한 체인의 우두머리 원소를 가지고 모든 체인의 우두머리 원소를 알아낼 수 있습니다.

- 모든 방향에 대하여 스택을 활용하여 $\mathcal{O}(N)$에 각 방향으로 가장 우선인 원소를 계산할 수 있고, 또한 모든 화분에 대하여 네 방향으로 우선인 원소를 따라가게 되면 생기는 체인의 길이를 계산할 수 있습니다.
- 이를 계산한 후, 모든 화분에 대하여 현재 화분을 우두머리라고 가정한 뒤, 나머지 우두머리들을 계산한 할 수 있습니다.
- 이때, 각 우두머리에 대한 체인의 길이 합을 통하여 이 배치가 모든 화분을 배정할 수 있는 배치인지 $\mathcal{O}(1)$에 확인할 수 있고, 고로 총 $\mathcal{O}(N)$에 분할선이 없는 경우에 대한 배정을 진행할 수 있습니다.
- 고로, 총 $\mathcal{O}(N)$에 문제를 해결할 수 있습니다.