2026 KPSC Spring Algorithm Challenge
문제 해설

by

국민대학교 알고리즘 동아리 KPSC

Host

Staff

출제

- saywoo
- haru_101
- chan120714
- usb9245
- young_out
- tapris
- kangsm02

운영

- kevinlys00
- naixt1478
- skuru
- sodiumd

검수

- dinojaemin
- playsworld16
- shinhuichan
- sjhi00

Sponsor

Sponsor

Sponsor

Sponsor

- cubic
- ssskid
- usb9245
- haru_101
- lntanx
- not_strober
- moner_ari

문제		의도한 난이도	출제자
A	문제 셋 만들기	Easy	saywoo
B	맛있는 햄버거	Easy	haru_101
C	빼빼로데이	Easy	chan120714
D	여기는 몇 층일까	Medium	usb9245, saywoo
E	FLYING HIGH WITH U	Medium	young_out
F	정렬과 쿼리	Hard	young_out, tapris
G	철도계획	Hard	kangsm02
H	서로소 수열	Hard	saywoo
I	2N 격자판	Challenging	young_out, chan120714
J	주사위 놀이	Challenging	tapris
K	알고리즘 가르치기	Challenging	kangsm02

A. 문제 셋 만들기

#math

출제진 의도 - **Easy**

- 제출 236번, 정답 133명 (정답률 60.169%)
- 처음 푼 사람: leinad2 (0분 48초)
- 출제자: saywoo

A. 문제 셋 만들기

- $N \geq 2$일 때는 항상 좋은 셋을 만들 수 있습니다.
- $N = 1$일 때, 문제 한 개의 난이도가 홀수일 때만 좋은 셋을 만들 수 없습니다.
- 즉, $N = 1$이고 A_1이 홀수인 경우 이외에는 모두 YES를 출력하면 됩니다.

B. 맛있는 햄버거

#math, #greedy

출제진 의도 - **Easy**

- 제출 131번, 정답 61명 (정답률 48.855%)
- 처음 푼 사람: leinad2 (2분 43초)
- 출제자: haru_101

B. 맛있는 햄버거

- 단순하게 접근하는 방법으로는 인덱스 i에 대해서, $\max_{j<i} A_j$를 찾아서 V에 더하면 됩니다.
  - 하지만, 이 방법으로는 $O(N^2)$의 시간이 소요되므로 $N = 500,000$인 경우에 시간 내에 풀기 어렵습니다.
- 따라서 최댓값을 저장하는 m 변수를 도입해서 $i = 1, ..., N$을 차례대로 순회하면서 $A_i \leq m$인 경우 A_i를 m으로 바꾸고, 아니면 $m = A_i$로 바꾸면 됩니다. 그 다음에 A_i를 V에 더하면 됩니다.
  - 이렇게 하면 $O(N)$의 시간복잡도로 풀 수 있습니다.

C. 빼빼로데이

#implementation, #case_work, #bruteforcing

출제진 의도 - **Easy**

- 제출 82번, 정답 44명 (정답률 54.878%)
- 처음 푼 사람: nflight11 (4분 48초)
- 출제자: chan120714

C. 빼빼로데이

- 범위내에 속하는 날짜는 약 42만개로, 많지 않으므로 모든 날짜에 대해 전부 확인 해보면 됩니다.
- 범위상의 이슈로 400년 윤년 체크는 고려하지 않아도 됩니다.
- 번외로, 아래 코드는 검수진 중 한 분이 제출한 코드입니다.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int T, A, B, C, k, ans=0; cin>>T>>A>>B>>C;
    k = A*10000+B*100+C;
    for(int i=20230101 ; i<=k ; i++){
        int y=i/10000, m=i/100%100, d=i%100;
        if(d>(4460848-16*!(y&(y%25?3:15))>>m+m&3^31)*(m<13)*!!m){i=i/100*100+100; continue;}
        for(y=i,d=T;y;y/=10)d-=y%10==1;
        ans+=d<=0;
    }

    cout<<ans;
}
```

D. 여기는 몇 층일까

#prefix_sum

출제진 의도 - **Medium**

- 제출 54번, 정답 33명 (정답률 61.111%)
- 처음 푼 사람: `lindelof` (7분 6초)
- 출제자: usb9245, saywoo

D. 여기는 몇 층일까

- i번 건물과 $i+1$번 건물의 층수 차이는 연결 통로의 층수 차($B_i - A_i$)입니다.
- 인접하지 않은 건물끼리의 층수 차이는 누적 합을 통해 구할 수 있습니다.
  - 층수 차이의 누적합을 저장할 변수를 D라고 가정합니다.
  - D_1에는 0을 저장하고, D_{i+1}에는 $(B_i - A_i) + D_i$를 저장합니다.
  - 이후 2번 쿼리가 주어지면 $\left(D_j - D_i\right) + x$를 출력하면 됩니다.

E. FLYING HIGH WITH U

#greedy

출제진 의도 - **Medium**

- 제출 69번, 정답 29명 (정답률 56.522%)
- 처음 푼 사람: `lindelof` (10분 27초)
- 출제자: young_out

E. FLYING HIGH WITH U

- 머릿속 1번째 칸만 떼어놓고 보면 A_1개의 생각이 필요합니다.
- 1번째 칸에 2번째 칸을 이어붙인다고 생각해봅시다.
  - $A_1 \geq A_2$인 경우, 1번째 칸을 차지하는 생각 일부가 2번째 칸도 차지하면 A_1개의 생각으로 2번째 칸까지 표현할 수 있습니다.
  - $A_1 < A_2$인 경우, 1번째 칸을 차지하는 생각 전부가 2번째 칸도 차지하고 추가로 $A_2 - A_1$개의 생각이 있으면 $A_1 + (A_2 - A_1)$개의 생각으로 2번째 칸까지 표현할 수 있습니다.

E. FLYING HIGH WITH U

- 따라서 j번째 칸을 차지하는 생각은 $j+1$번째 칸도 최대한 차지하는 것이 최선입니다.
- j번째에서 $j+1$번째로 이어붙일 때 추가로 필요한 생각의 개수는 $\max(0, A_{j+1} - A_j)$개입니다.
- 1번째 칸부터 N번째 칸까지 이어붙인다면 $O(N)$에 해결 가능합니다.

F. 정렬과 쿼리

#ad_hoc

출제진 의도 - **Hard**

- 제출 60번, 정답 27명 (정답률 45.000%)
- 처음 푼 사람: ibasic (12분 14초)
- 출제자: young_out, tapris

F. 정렬과 쿼리

- 1번 쿼리에 대해 $A_i, A_{i+1}, ..., A_j$ 구간에 v를 직접 더하고, 2번 쿼리에 대해 $A_1, A_2, ..., A_N$이 단조증가수열인지 직접 검사한다면, $O(NQ)$로 제한 시간 내에 해결할 수 없습니다. 어떻게 시간복잡도를 줄일 수 있을까요?
- 전체 수열의 정렬 여부는 모든 인접한 원소들의 정렬 여부와 같습니다.
- 1번 쿼리에 대해, $A_i, A_{i+1}, ..., A_j$ 구간에 v를 더하더라도 구간 내부의 정렬 여부는 변하지 않습니다.
- 따라서 1번 쿼리가 수행될 때 구간의 경계인 $A_{i-1} \le A_i$, $A_j \le A_{j+1}$의 결과가 바뀌는지만 확인하면 됩니다.

F. 정렬과 쿼리

- B_k를 인접한 원소의 차, 즉 $A_k - A_{k-1}$라고 정의합시다.
- 초기에 주어진 $A_1, A_2, ..., A_N$에 대해 $B_2, B_3, ..., B_N$을 구한 뒤, $B_k \geq 0$을 만족하는 k의 개수를 구해 C라고 합시다.
- 1번 쿼리에 대해, B_i에 v만큼 더하고 B_{j+1}에 v만큼 뺀 뒤, $B_i \geq 0$, $B_{j+1} \geq 0$이 바뀌었는지에 대해 C를 갱신하면 $O(1)$에 가능합니다.
- 2번 쿼리에 대해, $C = N - 1$을 만족하면 `YES`, 아니라면 `NO`를 출력하면 $O(1)$에 가능합니다.
- 쿼리당 $O(1)$이므로 총 시간복잡도 $O(N + Q)$에 해결 가능합니다.

G. 철도계획

#disjoint_set, #greedy

출제진 의도 - **Hard**

- 제출 37번, 정답 21명 (정답률 56.757%)
- 처음 푼 사람: leinad2 (18분 3초)
- 출제자: kangsm02

G. 철도계획

- 조건을 만족하면서 예상 수익의 합을 최대화해야합니다.
- 모든 구간을 예상 수익을 기준으로 내림차순 정렬합니다.
- 가중치가 높은 간선부터 하나씩 확인합니다.
  - 만약 간선을 추가하였을 때에도 조건을 위반하지 않는 경우 간선을 추가합니다.
  - 아닌 경우 추가하지 않습니다.
- 이를 모든 간선에 대해 확인하면 $O(M \log M)$만에 해결할 수 있습니다.

G. 철도계획

- e를 선택되지 않은 간선 중 예상 수익이 가장 큰 간선이라고 정의합니다.
- 만약 e를 선택하였을 경우, 조건에 위배된다면 e를 선택하기 위해서 이미 선택한 간선 e'를 지워야 합니다.
- 수익 순으로 정렬하였기 때문에 e'은 e보다 높거나 같은 수익을 지닙니다.
- 따라서 e를 e' 대신 선택하는 경우 수익의 총합은 같거나 적어집니다.
- 또한 e' 대신 e를 선택하는 경우 뒤의 선택에는 영향을 주지 않음을 증명할 수 있습니다.

G. 철도계획

- 즉, 조건을 위배하지 않는 선에서 먼저 선택한 간선들을 유지하는 것이 최적의 결과를 도출합니다.
- 간선 추가가 불가능한 조건을 만족하는 그래프 상태를 A라고 정의할 때, 어떤 A에서 간선을 하나 빼고 하나 넣는 행동을 반복하여 모든 A'에 도달할 수 있음을 보일 수 있습니다.
- 따라서 매 순간 수익이 큰 간선을 선택하는 것의 정당성을 증명할 수 있습니다.

H. 서로소 수열

#constructive, #sieve

출제진 의도 - **Hard**

- 제출 46번, 정답 18명 (정답률 39.130%)
- 처음 푼 사람: leinad2 (25분 45초)
- 출제자: saywoo

H. 서로소 수열

- 길이가 n이고 $\gcd(A_i, A_j) = 1$인 개수가 k개인 수열을 $f(n, k)$라고 합시다.
- 이전에 사용하지 않은 임의의 소수 2개를 각각 P_1, P_2라고 했을 때, 다음과 같이 $f(n, k)$를 배치할 수 있습니다.
  - $k \leq n - 1$인 경우
    - P_1, P_2, $P_1 \times P_2$를 각각 1, k, $n - k - 1$개 배치하면 됩니다.
  - $k > n - 1$인 경우
    - P_1을 1개 배치하고 뒤에 $f(n - 1, k - n + 1)$를 배치하면 됩니다.

H. 서로소 수열

- 수열에 사용되는 소수들은 에라토스테네스의 체 등의 방법으로 전처리를 해주면 됩니다.
- 전처리 후, 재귀적으로 수열을 배치하면 $O(N)$으로 수열을 배치할 수 있습니다.

I. 2N 격자판

#greedy, #ad_hoc, #case_work

출제진 의도 - **Challenging**

- 제출 6번, 정답 3명 (정답률 50.000%)
- 처음 푼 사람: leinad2 (59분 48초)
- 출제자: young_out, chan120714

I. 2N 격자판

- 한 번 방문했던 칸은 다시 방문할 수 없으므로, 편의상 `O`로 변한다고 합시다.
- 어떤 경로의 총 비용은 `S`에서 시작해 모든 `X`를 한 번씩 방문할 때 중간에 지나는 `O`의 개수와 같습니다.
- 따라서 `O`를 이동 가능하되 1의 비용이 드는 칸이라 볼 수 있습니다.
- 우리의 목표는 모든 `X`를 지나되 `O`를 최대한 적게 지나는 경로를 찾는 것입니다.

I. 2N 격자판

- 한쪽 끝에 도달하고 다른쪽 끝에 도달해야 하므로, S를 기준으로
  - ① 왼쪽 순회 → 오른쪽 끝까지 이동
  - ② 오른쪽 순회 → 왼쪽 끝까지 이동
- ①, ② 중 하나가 최선입니다. S가 가운데가 아닌 한쪽 끝에 있더라도 같은 방식으로 접근할 수 있으며, ①, ②는 대칭이므로 ①의 경우만 따져봅시다.

왼쪽 순회

- 편의상 S는 항상 1행에 존재한다고 합시다. S의 위치를 $(1, c_S)$라 하고, 왼쪽 끝에 있는 X의 위치를 (r_l, c_l)이라고 합시다.
- 왼쪽 순회는 $c_S \to c_l \to c_S$열까지 돌아오는 과정입니다.
  - $[c_l, c_S]$열의 2행에 X가 존재한다면 : $(1, c_S) \to (1, c_l) \to (2, c_l) \to (2, c_S)$
  - 존재하지 않는다면 : $(1, c_S) \to (1, c_l) \to (1, c_S)$
- 방향으로 행의 이동을 최소화하며 이동하는 것이 최선입니다. 비용은 이동 경로상에 있는 O의 개수와 같으며, 이동 후에는 c_S열에 위치하게 됩니다.

오른쪽 끝까지 이동

- 가장 오른쪽 X가 있는 열을 c_r이라 하면, c_S열부터 c_r열에 대해 모든 X를 방문할 때까지 다음 규칙에 따라 이동합니다.
  - 같은 열 다른 행에 X가 있다면 : 같은 열 다른 행으로 이동합니다.
  - 없다면 : 같은 행 오른쪽 열로 이동합니다.
- 열의 이동을 최소화하는 것이 항상 유리하므로 위 규칙이 최선입니다.
- 비용은 이동 경로상에 있는 O의 개수와 같으며, 이 과정을 반복해 비용의 최솟값을 구할 수 있습니다.

- 좌우 대칭을 통해 ②를 구할 수 있으므로 ①, ②의 비용 중 최솟값이 정답입니다.
- 총 시간복잡도 $O(N)$에 해결 가능합니다.

J. 주사위 놀이

#math, #probability_theory, #linearity_of_expectation, #greedy, #dp, #priority_queue, #sorting, #tree_set

출제진 의도 - **Challenging**

- 제출 19번, 정답 3명 (정답률 15.789%)
- 처음 푼 사람: `lindelof` (1시간 4분 13초)
- 출제자: `tapris`

J. 주사위 놀이

- 참조할 칸들에 대해 (n, m)으로 도달하기 위한 기댓값이 주어져 있다 할 때, 특정 칸에서 (n, m)으로 도달하는 기댓값을 구하는 방법을 생각해봅시다.
- 각 눈금이 나왔을 때 이동한다면, 4방향 중 기댓값이 가장 작은 칸을 택하는 것이 최적입니다.
- 이 값들을 모아 하나의 배열로 생각해봅시다.
- 기댓값이 최소가 되게 하려면 d개의 값 중 작은 몇 개의 경우에만 이동하고, 나머지 경우는 이동을 포기하는 전략을 사용할 수 있습니다.
- 기댓값에 대한 식을 잘 정리하면, 이 값이 정렬되어있는 배열 S가 있을때 기댓값은 $\min_{1 \le k \le d} \left(\frac{d + \sum_{i=1}^{k} S_i}{k} \right)$로 나타낼 수 있습니다.

J. 주사위 놀이

- 위 식을 관찰해보면 항상 최솟값이 되도록 하는 k에 대해 $S_k < \frac{d+\sum_{i=1}^{k} S_i}{k}$ 임을 알 수 있습니다.
- 특정 칸의 기댓값을 구하기 위해 필요한 참조할 칸들의 기댓값이 항상 더 작으므로 다익스트라의 아이디어와 정당성을 그대로 적용할 수 있습니다.
- (n, m)에서 출발하여, 각 눈금에 대해 해당 거리만큼 떨어진 칸들의 기댓값을 갱신하고, 각 칸에서 모든 눈금의 기댓값을 정렬된 상태로 관리하면서 최소 기댓값을 우선순위 큐에 넣는 방식으로 해결할 수 있습니다.
- 우선순위 큐에서 발생하는 시간복잡도는 $O(nmd \times \log(nmd))$ 입니다.

J. 주사위 놀이

- 각 칸에서 기댓값을 구하는데 발생하는 시간복잡도는 정렬된 상태를 관리하기 위해 tree_set등을 이용하면 $O(nmd^2)$, 매번 정렬한다면 $O(nmd^2 \log d)$ 가 됩니다.
- 따라서 총 $O(nmd(\log(nmd) + d))$ 혹은 $O(nmd(\log(nmd) + d \log d))$로 해결할 수 있습니다.
- 추가적으로 본 문제에서는 d의 제한이 매우 작아 유의미하지 않지만 $\frac{d+\sum_{i=1}^{k} S_i}{k}$이 단봉형임을 이용해 S를 bbst로 관리하면서 k에 대해 삼분탐색을 수행하여 최솟값을 찾는다면 $O(nmd(\log(nmd) + (\log d)^2))$로 해결할 수 있습니다.

K. 알고리즘 가르치기

#dp_sum_over_subsets

출제진 의도 - **Challenging**

- 제출 12번, 정답 5명 (정답률 41.667%)
- 처음 푼 사람: jjwdi0 (1시간 11분 53초)
- 출제자: kangsm02

K. 알고리즘 가르치기

- 모든 학습 과정은 ‘배운 태그 집합’ 에 매일 새로운 태그 하나를 추가해 나가는 과정입니다.
- 각 상태 S에서 얻는 만족도는 요구 태그 집합이 S의 부분집합인 문제 수 $g(S)$와 일치합니다.
- $S \to S \cup \{i\}$ 간선은 정확히 ‘현재 배울 수 있는 태그 i를 하루 동안 학습하는 행위’와 일치합니다.
- 따라서 가능한 모든 학습 과정은 부분집합 DAG 위의 경로와 일대일 대응합니다.

K. 알고리즘 가르치기

- 배운 태그 집합이 정확히 S인 상태에 도달했을 때까지 얻을 수 있는 만족도 총합의 최댓값에 대한 점화식은 마지막 한 단계를 기준으로 한 최적 부분 구조로 증명할 수 있습니다.
- 최종적으로 더 이상 배울 수 없는 상태들만이 실제 학습 종료 상태이므로, 그들 중 최대 DP가 정답이 됩니다.
- 따라서 $g(S) = \sum_{T \subseteq S}$'정확히 태그 집합 T를 요구하는 문제 수' 는 SOS DP로 $O(K \times 2^K)$에 계산 가능합니다. 선행 조건에 따른 학습 가능 여부 또한 부분집합 전처리로 $O(K \times 2^K)$에 구할 수 있고, 따라서 전체 시간복잡도는 $O(N \times K + K \times 2^K)$ 입니다.