20250919 ICTS 미니 대회

문제 1. 클리크 조절

- 정렬

- 문제 링크 : https://www.acmicpc.net/problem/33884
- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 1. 클리크 조절

- a, b, c, d, … 를 정렬하는 것과 a + x, b + x, c + x, d+ x,… 보면, 모든 원소에 같은 값을 더한 것이기에, 원소들 간의 크기 관계가 변하지 않으니까 정렬했을 때 순서는 똑같다.

- 첫 번째 사격 훈련에서 발사한 총알이 표적지에 꽂힌 위치와 두 번째 사격 훈련에서 발사한 총알이 표적지에 꽂힌 위치를 배열로 각각 입력 받아, 정렬한 뒤, 가장 앞 (또는 뒤)의 값을 비교하여 답을 출력

문제 2. 세과영엔 슬픈 전설이 있어

- 그리디
- 정렬
- 수학

- 링크: https://www.acmicpc.net/problem/30504
- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 2. 세과영엔 슬픈 전설이 있어

- 세종이가 받아야 하는 돈을 A, 영재가 줄 수 있는 돈은 B라고 하면, $A \le B$를 항상 만족시킬 수 있도록 B를 A와 매칭시켜야 한다.
- 가장 간단한 방법은 이 조건을 만족하려면, 작은 A에는 작은 B, 큰 A에는 큰 B를 배정하는 방법이다.
- 위 방법을 구현하려면 모든 A와 모든 B를 각각 오름차순 정렬 후, 일일이 비교해가며 대응되는 위치에 $A \le B$인지 확인하면 된다.
- 이때, 만약 한 번이라도 $B < A$라면 불가능하므로 -1을 출력한다.
- $B < A$ 인 경우가 나타난다면, 현재 A로 값을 수 있는 날이 없다는 의미이다. 즉 분노를 피해 빚을 갚는 방법이 없다.

문제 3. 수열 걷기

- 그리디
- 구현

- 링크 : https://www.acmicpc.net/problem/4929
- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 3. 수열 걷기

- 각 교차점 기준으로 수열을 나눌 수 있다. 이때 각 수열은 같은 개수의 교차점을 가지고 있으므로, 교차점이 총 n개가 있으면 각 수열을 n + 1개의 부분 수열로 나눌 수 있다 (이때, 각 부분 수열의 길이는 상이할 수 있다!)
- 이때, 각 교차점에서 어느 길을 갈 지 고를 수 있으므로, 다음 교차점까지 가장 많은 점수를 받을 수 있는 길을 교차점에서 선택하면 된다.

- 가장 간단한 방법은, 각 부분 수열의 합을 구한 뒤, 각 구간에서의 총합이 가장 큰 수를 계속 골라 최종 답변에 더하면 된다.

문제 4. 이브, 프시케 그리고 푸른 MEX의 아내

- 수학
- 에드 혹
- 조합론

- 링크 : https://www.acmicpc.net/problem/28250
- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 4. 이브, 프시케 그리고 푸른 MEX의 아내

- 정의에 의해 다음을 알 수 있다.
  - mex{0,0} == 1
  - mex{0,1} == 2
  - mex{0,x} == 1 (x >1),
  - mex{1,x} == 0 (x > 0),
- 즉 수열의 0, 1, n (n은 1, 0 아닌 정수)의 개수를 각각 확인하고, mex 값이 1인 경우와 2인 경우를 조합식을 활용해 구하면 된다.

문제 5. 택배

- 데이크스트라
- 역추적

- 링크: https://www.acmicpc.net/problem/1719
- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 5. 택배

- 각 점에서부터 데이크스트라 알고리즘을 돌린다 (데이크스트라 설명 링크).
- 이때, 데이크스트라 알고리즘을 구현하면서 역추적을 해둘 수 있어야 한다.
- 간단한 방법으로는 우선순위 큐에 넣는 정보에 이전 노드의 정보를 넣고, 데이크스트라 알고리즘을 실행시키면서 새로운 노드에 도달 했을 때, 같이 있었던 이전 노드에 대한 정보를 역추적 배열에 저장 시키는 방법이 있다.

- 별해: 플로이드–워셜과 역추적을 활용하여 풀이도 가능.

문제 5. 택배

데이크스트라 (역추적)

```
backtrack = [-1] * n
visited = [False]*n
queue= [(0,starting, -1)]
while queue:
    currTime,node, prev = heapq.heappop(queue)

    if visited[node]:
        continue

    backtrack[node] = prev
    visited[node] = True

    for nextNode, nextTime in connected[node]:
        if visited[nextNode]:
            continue
        heapq.heappush(queue,
                (currTime + nextTime,nextNode,node))
```

역추적 코드

```
current = ending
before = backtrack[current]
while before != starting:
    current = before
    before = backtrack[before]

if result[starting][ending] == -1:
    result[starting][ending] = current
```

문제 6. 엉성한 도토리 분류기

- 이분탐색

링크: https://www.acmicpc.net/problem/31848

- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 6. 엉성한 도토리 분류기

- 각 구멍이 받을 수 있는 도토리의 크기는 ai + i (i 는 구멍의 인덱스)
- 이때, 한 칸 앞의 구멍 A가 받을 수 있는 도토리의 크기가 더 크면 해당 구멍 B가 받을 수 있는 모든 도토리는 모두 앞에 있는 구멍이 받게 된다. 즉 B가 마주하게 되는 도토리는 필연적으로 못 받게 된다.
- 그러므로 도토리는 자신을 받을 수 있는 구멍을 만날 때 까지, 즉 A구멍보다 큰 도토리를 받을 수 있는 구멍을 만나기 전까지 굴러가게 된다.
- 예시)
  - 예제에서 각 구멍들의 크기는: 5 6 1 4 9 2 8 10 3 7
  - 실질적으로 받을 수 있는 도토리 크기: 5 7 3 7 13 7 14 17 11 16
  - 하지만 결국 5 7 X X 13 X 14 17 X X 처럼 되어버리기에, 각 X에 이전 값들 중 최댓값을 넣은 후 이분 탐색 돌리면 된다.
  - 결론: 5 7 7 7 13 13 14 17 17 17 에서 lower bound로 도토리가 들어갈 구멍의 인덱스를 찾자

문제 6. 엉성한 도토리 분류기

- 별해: 그리디, 오프라인 쿼리 (설명 링크)
- 오프라인 쿼리로 각 크기에 대해 어떤 구멍으로 들어갈지 O(N)에 미리 정해놓고 쿼리를 O(1)로 해결할 수 있다.
  - 도토리의 크기를 인덱스로 하는 배열 선언.
  - 각 구멍들의 크기: 5 6 1 4 9 2 8 10 3 7
  - 이때, 인덱스 i인 구멍 a_i가 받을 수 있는 도토리의 크기는 a_i + i 이다.
  - (배열의 비어있는 마지막 인덱스 x 부터 a_i + i 까지 i로 채운다.
    - 만일 x > a_i + i 인 경우 과정 스킵)
  - 각각의 도토리가 어떤 구멍으로 들어갈지: 1 1 1 1 1 2 2 5 5 5 5 5 5 7 8 8 ....
- 만약 N 제한이 10^6이었다면 별해가 아니라 정해였을 것 ~~(난이도는 동일한 듯?)~~

문제 7. 임스의 땅따먹기

- 브루트포스
- 누적 합

링크: https://www.acmicpc.net/problem/33561

- 처음 푼 사람:
- 가장 빠른 풀이를 한 사람:

문제 7. 임스의 땅따먹기

- 2차원 누적 합 문제.
- 0,0 부터 N, M 까지의 땅 가치를 저장한 2차원 누적 합 배열을 만든다.
- 0,0 부터 N, M 까지의 0의 개수를 누적하여 저장하는 2차원 배열을 만든다.

- 시작점을 0,0 부터, (n-1, n-1) 까지 잡으면서, 크기도 1 부터 x^2 (1 ≤ x ≤ n)인 정사각형을 모두 확인하여 조건에 맞는 최대값을 찾으면 된다.
- 우리는 최댓값을 원하기에, 0을 채울 필요가 있다면 가장 가치가 높은 것 부터 쓰면 된다.

문제 7. 임스의 땅따먹기

2차원 누적 합 구하기

```
for (int i = 0; i < n; i ++){
        for (int j = 0 ; j < n ; j ++) {
            cityVal[i][j] = city[i][j];
            if (i > 0)
                cityVal[i][j] += cityVal[i - 1][j];
            if (j > 0)
                cityVal[i][j] += cityVal[i][j - 1];
            if (i > 0 && j > 0)
                cityVal[i][j] -= cityVal[i- 1][j- 1];
        }
    }.
```

(i,j) 부터 (k,l)까지의 땅 가치 구하기

```
int partialSum(int i, int j, int k, int l, int arr[][500]) {
    int result = arr[k][l];
    if (j > 0)
        result -= arr[k][j - 1];
    if (i > 0)
        result -= arr[i - 1][l];
    if (i > 0 && j > 0)
        result += arr[i - 1][j - 1];

    return result;
}
```