코드포스 11주차 문제 풀이

문제 난이도

	문제 번호	문제 이름	난이도
A	28107	회전초밥	S1
B	14235	크리스마스 선물	S3
C	31860	열심히 일하는 중	S1
D	2781	놀이 동산	G1
E	2461	대표 선수	G2
F	21773	가희와 프로세스 1	G5
G	26094	더 어려운 스케줄링	P5
H	10979	가넷이나 버는게 낫지 않아요?	P4

A. 회전초밥

자료 구조, 우선순위 큐, 큐

A. 회전초밥

- ✓ **입력 처리**
  - ✓ 각 손님의 주문 목록을 처리하여 각 초밥 종류에 대해 해당 초밥을 원하는 손님의 인덱스를 우선순위 큐에 저장합니다.
  - ✓ 요리사가 제공하는 초밥의 종류를 순서대로 처리하여 손님이 초밥을 먹을 수 있는지 확인합니다.

- ✓ **각 손님의 주문 목록 처리**:
  - ✓ 각 손님의 주문 목록에 포함된 초밥 종류를 우선순위 큐에 삽입합니다.
  - ✓ 손님의 인덱스는 나중에 큐에서 최상단의 손님이 가장 작은 인덱스를 가지도록 합니다.
  - ✓ 이 과정에서 각 초밥 종류별로 우선순위 큐에 손님 인덱스가 저장됩니다.

- ✓ **초밥 제공 및 처리**:
  - ✓ 요리사가 제공하는 초밥에 대해 우선순위 큐에서 손님을 확인합니다.
  - ✓ 손님이 해당 초밥을 먹을 수 있는지 확인하고, 먹을 수 있다면 초밥을 먹었다고 기록합니다.
  - ✓ 손님이 초밥을 먹은 후, 그 손님의 인덱스를 큐에서 제거합니다.

- ✓ **시간 복잡도:** $O((K+M)\log N)$

B. 크리스마스 선물

자료 구조, 우선순위 큐

B. 크리스마스 선물

- ✓ **입력 처리**:
  - ✓ 첫 번째 줄에서 방문 횟수 n을 읽습니다.
  - ✓ 이후 n개의 작업을 처리합니다. 각 작업은 두 가지 형태가 있습니다
    - ✓ **거점지**: 선물을 충전합니다.
    - ✓ **아이들**: 현재 가장 가치가 큰 선물을 줍니다.

- ✓ **거점지에서 선물 충전**:
  - ✓ 거점지에서 선물을 충전하는 경우, 입력받은 선물의 가치를 우선순위 큐에 추가합니다.
  - ✓ 우선순위 큐는 최대 힙을 사용하여 가장 큰 선물을 빠르게 추출할 수 있도록 합니다.

- ✓ **아이들에게 선물 주기**:
  - ✓ 아이들이 있는 경우, 현재 우선순위 큐에서 가장 큰 선물의 가치를 출력하고, 해당 선물을 큐에서 제거합니다.
  - ✓ 만약 우선순위 큐가 비어있다면 -1을 출력합니다.

- ✓ **시간 복잡도:** $O(n * aloga)$

C. 열심히 일하는 중

구현, 자료 구조, 시뮬레이션, 우선순위 큐

C. 열심히 일하는 중

- ✓ **우선순위 큐를 사용하여 중요도 관리**:
  - ✓ 중요도가 높은 일을 우선적으로 처리하기 위해 최대 힙(우선순위 큐)를 사용합니다.

- ✓ **일 처리 과정**:
  - ✓ 큐에서 중요도가 가장 높은 일을 꺼내어 처리합니다.
  - ✓ 하루의 만족감은 전날 만족감의 절반에 오늘 처리한 일의 중요도를 더한 값입니다. 초기 만족감은 0입니다.
  - ✓ 일의 중요도를 M만큼 감소시키고, 감소된 중요도가 K를 초과하면 큐에 다시 추가합니다.
  - ✓ 처리된 일을 기록하여 총 일수와 매일의 만족감을 추적합니다.

- ✓ **출력**:
  - ✓ 총 일수를 출력합니다.
  - ✓ 매일의 만족감을 순서대로 출력합니다.

- ✓ **시간 복잡도:** $O(NlogN)$

D. 놀이 동산

구현, 그래프 이론, 다익스트라, 최단 경로

D. 놀이 동산

- **분수 형태로 비용 관리**:
  - 각 격자를 지나면서 총 비용이 1 이하가 되도록 해야 합니다.
  - 이동 비용이 1 / Cij로 주어지기 때문에, 총 비용이 1 이하가 되도록 해야 합니다.
  - 이를 효과적으로 관리하기 위해 비용을 분수 형태로 표현합니다. 각 위치까지 도달하는데 필요한 비용을 (분자/분모) 형태로 표현합니다.

- **다익스트라 알고리즘의 변형**:
  - 다익스트라 알고리즘은 일반적으로 최단 경로 문제를 해결하는 알고리즘입니다. 이 문제에서는 단순히 거리(비용)뿐만 아니라, 시간의 단위로 나눠서 처리해야 합니다
  - 따라서, 다익스트라 알고리즘의 변형을 사용하여 각 위치까지의 최적 시간을 찾습니다. 각 이동에서의 총 비용이 1 이하가 되어야 하므로, 비용을 분수 형태로 처리합니다.

- **우선순위 큐를 이용한 최단 경로 탐색**:
  - **우선순위 큐**를 사용하여, 가장 낮은 비용을 가진 경로를 우선적으로 탐색합니다.
  - 우선순위 큐는 비용이 낮은 경로를 우선적으로 처리합니다. 이 문제에서는 분수 형태로 비용을 처리하므로, 분자와 분모를 이용하여 비교합니다.

D. 놀이 동산

- ✓ **분수 형태로 시간 계산**:
  - ✓ 시간 계산: 현재의 총 비용을 분수로 표현하고, 이동할 때마다 이 총 비용을 업데이트 합니다.
  - ✓ 예를 들어, 현재까지의 총 비용이 A / B이고, 이동할 격자의 비용이 Cij, 이동 후의 총 비용은 (A * Cij + B) / (B * Cij) 형태로 계산됩니다.
  - ✓ 이 값을 분수로 관리하며, 이를 통해 매일의 시간 단위로 처리할 수 있습니다.

- ✓ **매일의 만족감 계산**:
  - ✓ 각 시간 단위에서 총 비용이 1 이하가 되도록 해야 하며, 이를 만족하는 경로를 우선적으로 탐색합니다.
  - ✓ 매일의 만족감은 이전 만족감과 현재까지의 총 비용을 기준으로 계산됩니다.

- ✓ **시간 복잡도:** $O(NMlog(NM))$

E. 대표 선수

자료 구조, 정렬, 우선순위 큐, 두 포인터

E. 대표 선수

- ✓ **입력 데이터 읽기 및 준비**:
  - ✓ N (학급 수)과 M (각 학급의 학생 수)을 읽습니다.
  - ✓ 각 학급의 학생 능력치를 읽어 (능력치, 학급 번호) 형태로 저장합니다.
  - ✓ 이 데이터를 능력치에 따라 정렬합니다.

- ✓ **슬라이딩 윈도우 초기화**:
  - ✓ s와 e를 각각 시작 포인터와 끝 포인터로 초기화합니다.
  - ✓ cnt 배열을 사용하여 각 학급에서 현재 윈도우 내에 있는 학생 수를 추적합니다.
  - ✓ zeros 변수는 현재 윈도우 내에 최소한 한 명씩 있는 학급의 수를 추적합니다.
  - ✓ ans를 무한대로 초기화하여 최솟값을 기록합니다.

E. 대표 선수

- ✓ **슬라이딩 윈도우 적용**:
  - ✓ **윈도우 확장:** e 포인터를 오른쪽으로 이동시키며 윈도우를 확장합니다. 현재 e 위치의 학급의 학생이 추가될 때, 해당 학급의 카운트를 증가시키고, 학급 수(zeros)를 갱신합니다.
  - ✓ **윈도우 축소:** zeros가 모든 학급에 대해 1 이상일 때, 현재 윈도우의 능력치 차이를 계산하여 ans를 갱신합니다. 이후 s 포인터를 오른쪽으로 이동시키며 윈도우를 축소합니다.

- ✓ **결과 출력**:
  - ✓ 최종적으로 계산된 ans가 최소의 최댓값과 최솟값의 차이를 담고 있으며, 이를 출력합니다.

- ✓ **시간 복잡도:** $O(NMlog(NM))$

F. 가희와 프로세스 1

자료 구조, 우선순위 큐

F. 가희와 프로세스 1

- **입력 처리**:
  - T초 동안의 스케줄링을 시뮬레이션하며, 각 프로세스의 ID, 실행 시간, 초기 우선 순위를 읽어들입니다.
  - 프로세스는 우선 순위와 ID에 따라 정렬된 상태로 관리됩니다.

- **프로세스 저장 및 우선 순위 큐 초기화**:
  - 각 프로세스의 정보를 priority_queue를 사용하여 우선 순위에 따라 저장합니다.
  - priority_queue는 우선 순위가 높은 프로세스를 먼저 꺼낼 수 있도록 설정됩니다.

- **스케줄링 시뮬레이션**:
  - T초 동안 반복하여 매 초마다 우선 순위 큐에서 프로세스를 선택합니다.
  - 선택된 프로세스의 ID를 출력하고, 실행 시간이 남아 있다면 우선 순위를 업데이트하여 다시 큐에 넣습니다.
  - **프로세스의 우선 순위는 매 초마다 1씩 감소**하며, 실행 시간은 1초씩 감소합니다.

- **시간 복잡도:** $O(NlogN + TlogN)$

G. 더 어려운 스케줄링

자료 구조, 트리를 사용한 집합과 맵, 우선순위 큐, 덱

G. 더 어려운 스케줄링

- **데이터 구조**:
  - deque<int> dq: 실제 업무를 저장하는 덱(deque)입니다.
  - set<int> s1, set<int> s2: 두 개의 집합으로, 각각 정렬된 상태를 유지합니다. s1은 오름차순으로, s2는 스택처럼 사용됩니다. set대신 우선순위 큐를 사용할 수 있습니다.

- **명령 처리**:
  - 명령 0 p (업무 추가):
    - front 플래그에 따라 덱의 맨 앞 또는 맨 뒤에 업무 p를 추가합니다.
    - 추가된 업무는 s2 집합에 저장됩니다.
  - 명령 1 (정렬):
    - 현재 덱의 상태를 s2에서 s1으로 이동합니다.
    - 덱을 초기화한 후, s1에 저장된 업무를 덱의 뒤에 추가(-1를 추가하여 3 연산 때 -1가 뽑혔다면 set에서 비우게 합니다.)하여 정렬된 상태로 유지합니다. s2는 비워집니다.
  - 명령 2 (뒤집기):
    - front 플래그를 반전시켜 덱의 처리 방향을 뒤집습니다.
  - 명령 3 (업무 처리 및 출력):
    - front 플래그에 따라 덱의 가장 앞 또는 가장 뒤의 업무를 꺼냅니다.
    - -1이라면 s1에서, 아니라면 s2에서 비웁니다.
    - s1이 empty라면 -1를 제거합니다.

H. 가넷이나 버는게 낫지 않아요?

자료 구조, 그래프 이론, 다익스트라, 최단 경로

H. 가넷이나 버는게 낫지 않아요?

- ✓ **그래프 구조**:
  - ✓ 노드: 섬을 나타내며, 1부터 N까지의 정수로 표현됩니다.
  - ✓ 간선: 각 간선은 섬 간의 다리를 나타내며, 다리를 건너는 데 걸리는 시간 t와 얻을 수 있는 가넷의 수 g를 가집니다.

- ✓ **목표**:
  - ✓ 1번 섬에서 N번 섬으로 가는 최단 경로를 찾되, 두 번째로 짧은 시간과 그에 대한 최대 가넷을 찾는 것입니다.

- ✓ **사용하는 자료구조**:
  - ✓ 우선순위 큐: 다익스트라 알고리즘을 사용하기 위해 priority_queue를 활용합니다.
  - ✓ 거리 배열: Dist 배열은 각 노드에서 0번째와 1번째로 짧은 시간과 그에 따른 최대 가넷을 저장합니다. Dist[u][0]은 최단 시간과 가넷을 저장하고, Dist[u][1]은 두 번째로 짧은 시간과 그에 따른 가넷을 저장합니다.

H. 가넷이나 버는게 낫지 않아요?

- ✓ **다익스트라 알고리즘**:
  - ✓ 우선순위 큐에서 현재 노드, 경과 시간, 얻은 가넷을 가져와 처리합니다.
  - ✓ 각 노드의 인접 노드에 대해 경로를 업데이트하고, 두 가지 경우를 처리합니다:
    - ✓ 최단 경로 업데이트: 현재까지 발견된 최단 경로보다 짧은 시간과 최대 가넷을 갱신합니다.
    - ✓ 두 번째 최단 경로 업데이트: 현재까지 발견된 두 번째로 짧은 경로와 비교하여 가넷을 갱신합니다.
- ✓ **최종 결과**:
  - ✓ **두 번째 최단 경로**의 시간과 가넷을 출력합니다.
- ✓ **시간 복잡도:** $O((N+M)logN)$