Chapter 02.
고급 자료구조

핵심 유형 문제풀이

핵심 유형 문제풀이 | 다양한 문제를 접하며 코딩 테스트에 익숙해지기

강사 나동빈

Chapter 02. 고급 자료구조

핵심 유형 문제풀이

절댓값 힙: https://www.acmicpc.net/problem/11286

혼자 힘으로 풀어보기

문제 제목: 절댓값 힙

문제 난이도: 중(Medium)

문제 유형: 자료구조, 우선순위 큐

추천 풀이 시간: 40분

절댓값 힙: https://www.acmicpc.net/problem/11286

문제 풀이 핵심 아이디어

- 본 문제는 우선순위 큐 자료구조를 이용해 해결할 수 있는 문제입니다.
- 파이썬의 $heapq$ 라이브러리를 이용해 **우선순위 큐**를 활용해 해결합니다.

절댓값 힙: https://www.acmicpc.net/problem/11286

문제 풀이 핵심 아이디어

[알고리즘 요약]

- 절댓값 힙은 단순히 하나의 힙(heap)을 이용해 구현할 수 있다.
- 연산의 개수가 10만 개 이므로, $O(NlogN)$의 시간 복잡도가 요구된다.

1. 힙에 각 원소를 넣을 때 $(|x|, x)$를 넣으면 된다.
2. 힙에서 원소를 꺼낼 때는 절댓값을 기준으로 꺼내게 된다.
    - 절댓값이 동일한 데이터가 여러 개라면 낮은 값부터 꺼내야 한다.
    - 따라서 두 번째 원소에 원본 데이터를 넣어주는 것이다.

절댓값 힙: https://www.acmicpc.net/problem/11286

소스 코드


```
import sys
input = sys.stdin.readline # 빠른 입력 함수 사용
import heapq


heap = []
n = int(input()) # 연산의 개수 N
for i in range(n):
    x = int(input()) # 연산 정보
    if x == 0: # 삭제 연산이라면
        if len(heap) == 0: # 힙이 비어있는 경우
            print(0)
        else: # 힙이 비어있지 않은 경우
            absolute, original = heapq.heappop(heap)
            print(original)
    else: # 삽입 연산이라면
        heapq.heappush(heap, (abs(x), x))
```

이중 우선순위 큐: https://www.acmicpc.net/problem/7662

혼자 힘으로 풀어보기

문제 제목: 이중 우선순위 큐

문제 난이도: 중상

문제 유형: 자료구조, 우선순위 큐

추천 풀이 시간: 60분

문제 풀이 핵심 아이디어

- 본 문제는 우선순위 큐 자료구조를 이용해 해결할 수 있는 문제입니다.
- 파이썬의 $heapq$ 라이브러리를 이용해 **2개의 우선순위 큐**로 해결합니다.

문제 풀이 핵심 아이디어

[알고리즘 요약]

- 이중 우선순위 큐는 두 개의 힙(heap)을 이용해 구현할 수 있다.
- 연산의 개수가 100만 개이므로, $O(NlogN)$의 시간 복잡도가 요구된다.

1. 먼저 최소힙과 최대힙을 하나씩 만들어 모든 데이터를 넣는다.
2. 데이터를 삭제할 때는 최소힙과 최대힙의 싱크를 맞추어야 한다.
    - 예를 들어 우선순위가 높은 것을 삭제할 때는 최대힙에서 빼고, 최소힙과 싱크를 맞춘다.
    - 따라서 각 원소에 대해서는 별도의 **번호(ID) 값으로 삭제 여부를 기록**한다.
    - 즉, 데이터를 넣을 때 (값, 번호) 형태로 넣어야 한다.
    - 나중에 데이터를 뺄 때 이미 삭제된 데이터라면 한 번 더 뽑는다.

이중 우선순위 큐: https://www.acmicpc.net/problem/7662

소스 코드


```python
import sys
input = sys.stdin.readline # 빠른 입력 함수 사용
import heapq


def pop(heap): # heap과 deleted는 global 변수
    while len(heap) > 0: # 삭제할 원소를 찾기
        data, id = heapq.heappop(heap) # 꺼내기
        if not deleted[id]: # 처음 삭제하는 원소일 때
            deleted[id] = True
            return data
    return None # 삭제할 원소가 없으면 None 반환
```

```python
for _ in range(int(input())): # 테스트 케이스만큼 반복
    k = int(input()) # 연산의 개수 K
    min_heap = [] # 최소 힙 (heapq는 최소 힙)
    max_heap = [] # 최대 힙 (삽입/삭제할 때 음수로 넣기)
    current = 0 # 삽입할 원소의 인덱스(ID 값)
    deleted = [False] * (k + 1) # 각 원소의 삭제 여부
    for i in range(k): # 차례대로 연산을 입력받기
        command = input().split()
        operator, data = command[0], int(command[1])
        if operator == 'D': # 삭제(delete) 연산
            if data == -1: pop(min_heap)
            elif data == 1: pop(max_heap)
        elif operator == 'I': # 삽입(insert) 연산
            heapq.heappush(min_heap, (data, current))
            heapq.heappush(max_heap, (-data, current))
            current += 1
    max_value = pop(max_heap)
    if max_value == None: print("EMPTY")
    else:
        # max_value는 min_heap에서도 꺼내지므로
        heapq.heappush(min_heap, (-max_value, current))
        print(-max_value, pop(min_heap))
```

소수의 곱: https://www.acmicpc.net/problem/2014

혼자 힘으로 풀어보기

문제 제목: 소수의 곱

문제 난이도: 중상

문제 유형: 자료구조, 우선순위 큐

추천 풀이 시간: 60분

소수의 곱: https://www.acmicpc.net/problem/2014

문제 풀이 핵심 아이디어

- 본 문제는 우선순위 큐 자료구조를 이용해 해결할 수 있는 문제입니다.
- 소수의 곱을 **오름차순**으로 나타내야 하므로, 우선순위 큐를 활용할 수 있습니다.
- **매번 가장 작은 원소를 꺼내서**, 처음에 부여받은 소수와 곱하여 다시 우선순위 큐에 넣습니다.

소수의 곱: https://www.acmicpc.net/problem/2014

문제 풀이 핵심 아이디어

[알고리즘 요약]

1. 최소힙(min heap)을 준비하여 "처음에 부여받은 모든 소수"를 삽입한다.
2. 이후에 다음의 과정을 N**번 반복**한다.
   - $top()$의 원소를 꺼낸다.
   - **처음에 부여받은 소수들과 "곱한 결과"**를 다시 삽입한다.

- 이러한 과정에서 두 가지를 고려한다.
  1. 힙의 크기가 N 이상이고, 힙의 최댓값보다 **[곱한 결과]**가 크다면 무시한다.
  2. 한 번 구한 결과는 다시 큐에 넣을 필요가 없다.

소수의 곱: https://www.acmicpc.net/problem/2014

문제 풀이 핵심 아이디어

- **초기 원소 구성**: [2, 5, 7]

① 처음에 모든 원소를 삽입한다.

우선순위 큐: [2, 5, 7] / **꺼낸 원소**: []

② 원소 2를 꺼내어 [2, 5, 7]과 각각 곱하여 삽입한다. (4, 10, 14 삽입)

우선순위 큐: [4, 5, 7, 10, 14] / **꺼낸 원소**: [2]

③ 원소 4를 꺼내어 [2, 5, 7]과 각각 곱하여 삽입한다. (8, 20, 28 삽입)

우선순위 큐: [5, 7, 8, 10, 14, 20, 28] / **꺼낸 원소**: [2, 4]

소수의 곱: https://www.acmicpc.net/problem/2014

Ch2. 고급 자료구조
핵심 유형 문제풀이

소스 코드

```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
import heapq


# 입력 소수의 개수 K와 구할 N번째 수
k, n = map(int, input().split())
data = list(map(int, input().split()))

heap = [] # 우선순위 큐(최소 힙)
visited = set() # 이미 처리된 수
max_value = max(data) # 최댓값

for x in data: # 초기 원소를 최소 힙에 삽입
    heapq.heappush(heap, x)
```

```python
# 힙에서 N - 1개의 원소 꺼내기
for i in range(n - 1):
    element = heapq.heappop(heap)
    for x in data:
        now = element * x # 곱한 결과 계산
        # 힙의 크기가 N 이상이고 힙의 최댓값보다 곱한 결과가 크다면
        if len(heap) >= n and max_value < now:
            continue
        if now not in visited: # 처음 방문하는 수라면
            heapq.heappush(heap, now)
            max_value = max(max_value, now)
            visited.add(now) # 방문 처리

# N번째 원소 꺼내기
print(heapq.heappop(heap))
```