1. **Largest N numbers**

문제 : N x N 크기의 정수 행렬 M이 있을 때, M 행렬의 각 원소들이 서로 달라 동일한

두 원소가 없다고 가정한다. 이때. M 행렬의 가장 큰 N개의 원소를 효율적으로 찾을 수

있는 알고리즘을 제안해야 한다. 또한 그 알고리즘의 정확성에 대한 증거를 제시해야 한다.

문제 해석 : 문제의 마지막 문장을 보게 되면 행렬 M의 초기 상태를 보존할 필요가 없다고

나와 있다. 이를 다시 생각해보면 행렬 M의 초기 상태가 이미 존재하기 때문에

보존할 필요가 없다는 뜻으로 해석했고, 따라서 수학적인 행렬의 형태를

생각해봤을 때, 2차원 배열이 행렬 M의 초기상태일 것이라는 결론을 내렸다.

제안 : 행렬의 모든 원소들 중에서 가장 큰 N개의 원소를 찾아야 하기 때문에

행렬의 원소들을 정렬한 뒤 가장 큰 숫자가 있는 인덱스부터 N까지의 원소들을 뽑게

하려고 한다. 본인이 선택한 정렬은 최대 힙 정렬이다.

최대 힙 : 이진 트리 구조로 구성되어 있다.

삽입 - 데이터를 삽입하면 최하단부 노드에 입력된다. 그 다음 삽입된 노드의 부모노드를

보고 데이터가 더 크다면 부모노드와 위치를 맞바꾼다. 이 과정을 부모노드의

데이터가 더 크거나 루트 노드까지 이동할 때까지 진행된다.

도표, 원, 클립아트, 화이트이(가) 표시된 사진

자동 생성된 설명 원, 도표, 클립아트, 화이트이(가) 표시된 사진

자동 생성된 설명 원, 도표, 클립아트, 스케치이(가) 표시된 사진

자동 생성된 설명

삭제 – 최댓값을 찾는 자료구조이기 때문에 루트노드를 삭제한다. 루트노드가 삭제되면

최하단부 노드를 루트로 옮긴다. 그 다음 자식노드와 비교하여 데이터가 더

작으면 자식노드와 위치를 맞바꾼다. 만약 두 자식노드의 데이터 모두 크다면

그 중 더 큰 자식노드와 위치를 맞바꾼다. 이 과정 역시 자식노드의 데이터가

더 작거나 최하단부 노드까지 올 동안 진행된다.

스케치, 원, 클립아트, 화이트이(가) 표시된 사진

자동 생성된 설명 원, 스케치, 클립아트, 그림이(가) 표시된 사진

자동 생성된 설명 원, 도표, 스케치, 클립아트이(가) 표시된 사진

자동 생성된 설명

따라서 최대 힙을 이용해서 행렬의 모든 원소를 삽입한 뒤 N개의 원소를 순서대로

힙에서 추출하면 가장 큰 N개의 원소를 구할 수 있다

시간 복잡도 : 최대 힙에 원소 하나를 삽입할 때, 시간복잡도 logN이 걸리며,

원소 하나를 삭제할 때도 시간복잡도 logN이 걸린다.

따라서 행렬의 모든 원소를 힙에 삽입할 때의 시간복잡도는 N2⋅logN이다.

그 다음 가장 큰 N개의 원소를 뽑기 위해 힙에서 N개의 원소를

삭제할 때의 시간 복잡도는 N⋅logN이다.

따라서 최종적인 시간복잡도 T(N) = N2⋅logN + N⋅logN이다.

공간 복잡도 : 최대 힙의 원소들을 저장할 공간이 필요하기 때문에

공간복잡도 T(N) = N2이다.

1. **Position of k**

문제 : non-decreasing order로 배열된 N x N 크기의 정수 행렬이 있다고 가정한다.

이 행렬과 정수 k가 주어지면 행렬내에서 k의 위치를 효율적으로 구하는 알고리즘을

제안해야 한다. 만약 k가 행렬 내에 여러 개가 있다면 알고리즘은 그 중 하나를 식별한다.

제안 : 본인은 2차원 배열로 저장된 행렬을 빠르게 탐색하기 위해 non-decreasing order을

이용하려고 한다. Non-decreasing order는 다시 말해 오름차순으로 행렬이 배열되어

있다는 것을 의미하며 이를 통해 이진 탐색을 적용시킬 수 있다.

이진 탐색을 적용하기 위해 2차원 배열을 1차원 배열로 축소시켜서 보게 되면

0 ~ N\*N까지의 인덱스가 있고, 나머지 알고리즘은 중간 값을 구하고 그 중간 값이

목표 값보다 높으면 마지막 인덱스를 중간 값으로 바꾸거나 목표 값보다 낮으면

첫번째 인덱스를 중간 값으로 바꾸는 것은 기존 이진 탐색 알고리즘과 같다.

여기서 관건은 1차원 정보를 바탕으로 2차원 배열의 위치를 알아내어

목표 값과 비교할 수 있어야 하는 것이다.

행에 대한 정보는 N으로 나눈 몫을 이용하고, 열에 대한 정보는 N으로 나눈 나머지를

이용하면 구할 수 있다.

예시 : N = 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 열(%N)  행(/N) | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 🡪  (0,0) | 1 🡪  (0,1) | 2 🡪  (0,2) | 3 🡪  (0,3) | 4 🡪  (0,4) | 5 🡪  (0,5) |
| 1 | 6 🡪  (1,0) | 7 🡪  (1,1) | 8 🡪  (1,2) | 9 🡪  (1,3) | 10 🡪  (1,4) | 11 🡪  (1,5) |
| 2 | 12 🡪  (2,0) | 13 🡪  (2,1) | 14 🡪  (2,2) | 15 🡪  (2,3) | 16 🡪  (2,4) | 17 🡪  (2,5) |
| 3 | 18 🡪  (3,0) | 19 🡪  (3,1) | 20 🡪  (3,2) | 21 🡪  (3,3) | 22 🡪  (3,4) | 23 🡪  (3,5) |
| 4 | 24 🡪  (4,0) | 25 🡪  (4,1) | 26 🡪  (4,2) | 27 🡪  (4,3) | 28 🡪  (4,4) | 29 🡪  (4,5) |
| 5 | 30 🡪  (5,0) | 31 🡪  (5,1) | 32 🡪  (5,2) | 33 🡪  (5,3) | 34 🡪  (5,4) | 35 🡪  (5,5) |

시간 복잡도 : 기존 이진탐색을 생각해보면 목표 값과 비교할 때 인덱스의 범위가

절반으로 줄어들기 때문에 만약 처음 행렬의 원소가 N⋅N개라면,

한번 탐색하면 남은 원소는 N2/2개, 두 번 탐색하면 (N2/2) ⋅ (1/2)개

이렇게 절반씩 줄어든다.

이를 일반화하면 k번 탐색을 했을 때 남은 원소 수는 (1/2)k X N2 개이다.

최악의 경우로 생각하면 (1/2)k X N2 = 1이고, k만 한 변에 남겨두면

최종식은 k = 2⋅log2N이 된다.

따라서 본인이 구현한 알고리즘의 시간복잡도 T(N) = 2⋅log2N이다.

공간 복잡도 : 이진 탐색을 할 때, 첫 인덱스, 마지막 인덱스, 중간 인덱스를 저장할

변수 3개가 필요하며 2차원 배열의 인덱스로 변환하여 저장할 변수 2개도

필요하다. 그리고 최종적으로 찾아야 할 목표 값과 그 위치를 저장할

변수 3개까지 필요하기 때문에 본인이 구현한 알고리즘의 공간복잡도

T(N) = 8이다.