

백준 | 2075번 N번째 큰 수

통계 수정 삭제

khi8930 · 방금 전

❤️ 0



백준

스터디

2023 Algorithm

▼ 목록 보기

6/6



- 소요시간 : 1시간
- 자료 참고 여부 : X
- 체감 난이도 : 中 ~ 下

2075번 문제

N번째 큰 수 성공 ☆

🏆 실버 II

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|-------|-------------------------------|-------|-------|-------|---------|
| 1 초 | 12 MB (하단 참고) | 26389 | 10586 | 7667 | 39.022% |

문제

N×N의 표에 수 N²개 채워져 있다. 채워진 수에는 한 가지 특징이 있는데, 모든 수는 자신의 한 칸 위에 있는 수보다 크다는 것이다. N=5일 때의 예를 보자.

| | | | | |
|----|----|----|----|----|
| 12 | 7 | 9 | 15 | 5 |
| 13 | 8 | 11 | 19 | 6 |
| 21 | 10 | 26 | 31 | 16 |
| 48 | 14 | 28 | 35 | 25 |
| 52 | 20 | 32 | 41 | 49 |

이러한 표가 주어졌을 때, N번째 큰 수를 찾는 프로그램을 작성하시오. 표에 채워진 수는 모두 다르다.

- <https://www.acmicpc.net/problem/2075>

- 오랜만에 초심자의 행운이 따른 문제
- 자료구조 수업에서 사용했던 코드 + 힙 구조이기 때문에 쉽게 풀 수 있었음

코드 전문

```
#pragma warning(disable:4996)
#pragma warning(disable:4013)
#include<stdio.h>
#include<string.h>
#include <stdlib.h>
# include <time.h>
#define MAX_NUM 1500
```

```

typedef struct{
    int data;
}element;

typedef struct{
    element heap[MAX_NUM * MAX_NUM];
    int heap_size;
}HeapType;

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void upHeap(HeapType* n, int i) {

    if (i == 1) return;

    if (n->heap[i].data > n->heap[i / 2].data) {
        swap(&n->heap[i].data, &n->heap[i / 2].data);
        i = i / 2;
        upHeap(n, i);
    }

}

void DownHeap(HeapType* n, int i) {

    int left = i * 2;
    int right = i * 2 + 1;

    if (left > n->heap_size) return;

    int larger = left;

    if (n->heap[larger].data < n->heap[right].data) {
        larger = right;
    }

    if (n->heap[larger].data > n->heap[i].data) {
        swap(&n->heap[larger].data, &n->heap[i].data);
    }

    DownHeap(n, larger);

}

void insertHeap(HeapType* n, int a) {
    n->heap_size++;
    n->heap[n->heap_size].data = a;
    upHeap(n, n->heap_size);

    return;
}

int removeHeap(HeapType* n) {
    int key = n->heap[1].data;
    n->heap[1].data = n->heap[n->heap_size].data;
    n->heap_size--;
    DownHeap(n, 1);

    return key;
}

int main() {

    int N, a;

    scanf("%d", &N);

    HeapType* m = (HeapType*)malloc(sizeof(HeapType));

```

```

m->heap_size = 0;

for (int i = 1; i < N*N + 1; i++) {
    scanf("%d", &a);
    insertHeap(m, a);
}

for (int i = 1; i <= N; i++) {
    int b = removeHeap(m);

    if (i == N) {
        printf("%d ", b);
    }
}

return 0;
}

```

코드 해석

```

typedef struct{
    int data;
}element;

typedef struct{
    element heap[MAX_NUM * MAX_NUM];
    int heap_size;
}HeapType;

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

```

- 힙의 구조 사용
 - 최대로 주어질 수 있는 수 (1500)을 제공한 수 만큼의 배열 크기
 - 배열 속에 데이터 변수 할당
 - 힙의 크기를 알 수 있는 변수를 할당하여 upHeap, downHeap 등을 수행할 때 편리하게 변수를 사용할 수 있도록 함

```

void upHeap(HeapType* n, int i) {

    if (i == 1) return;

    if (n->heap[i].data > n->heap[i / 2].data) {
        swap(&n->heap[i].data, &n->heap[i / 2].data);
        i = i / 2;
        upHeap(n, i);
    }
}

```

- upHeap () : 노드를 가장 아래 리프 노드에 붙이고 부모 노드와 비교하여 교환하는 함수
 - 루트 노드이면 반환
 - 부모 노드보다 크다면 두 수를 변환하고 재귀 함수 실행

```

void DownHeap(HeapType* n, int i) {

    int left = i * 2;
    int right = i * 2 + 1;

    if (left > n->heap_size) return;

    int larger = left;

    if (n->heap[larger].data < n->heap[right].data) {
        larger = right;
    }

    if (n->heap[larger].data > n->heap[i].data) {
        swap(&n->heap[larger].data, &n->heap[i].data);
    }

    DownHeap(n, larger);
}

```

- DownHeap() : 특정 노드에서부터 자식노드와 비교하면서 교환하는 함수
 - 만약 left 자식 노드가 힙의 크기보다 크면 반환
 - left 노드와 right 노드의 변수 크기 비교를 하고 더 큰 노드를 larger 변수에 저장
 - 만약 지금 노드보다 자식노드가 크다면 두 수를 변환
 - 리프 노드에 도달해서 자동으로 반환되기 전까지 DownHeap 함수 실행

```

void insertHeap(HeapType* n, int a) {
    n->heap_size++;
    n->heap[n->heap_size].data = a;
    upHeap(n, n->heap_size);

    return;
}

int removeHeap(HeapType* n) {
    int key = n->heap[1].data;
    n->heap[1].data = n->heap[n->heap_size].data;
    n->heap_size--;
    DownHeap(n, 1);

    return key;
}

```

- insertHeap () : 힙에 노드를 삽입하는 함수
 - 함수가 실행되었다면 힙의 크기를 1씩 증가
 - 힙의 가장 마지막 노드에 a 저장
 - upHeap 함수 실행해서 힙 양식에 맞게 정렬
- removeHeap(): 힙에서 루트 노드를 삭제하고 재정렬하는 함수
 - 루트 노드를 key 변수에 저장
 - 가장 마지막에 위치한 노드를 루트 노드로 옮김
 - 힙 크기 조정하고 루트 노드부터 DownHeap 함수 실행
 - 루트 노드 반환

```

int main() {

```

```

int N, a;

scanf("%d", &N);

HeapType* m = (HeapType*)malloc(sizeof(HeapType));

m->heap_size = 0;

for (int i = 1; i < N*N + 1; i++) {
    scanf("%d", &a);
    insertHeap(m, a);
}

for (int i = 1; i <= N; i++) {
    int b = removeHeap(m);

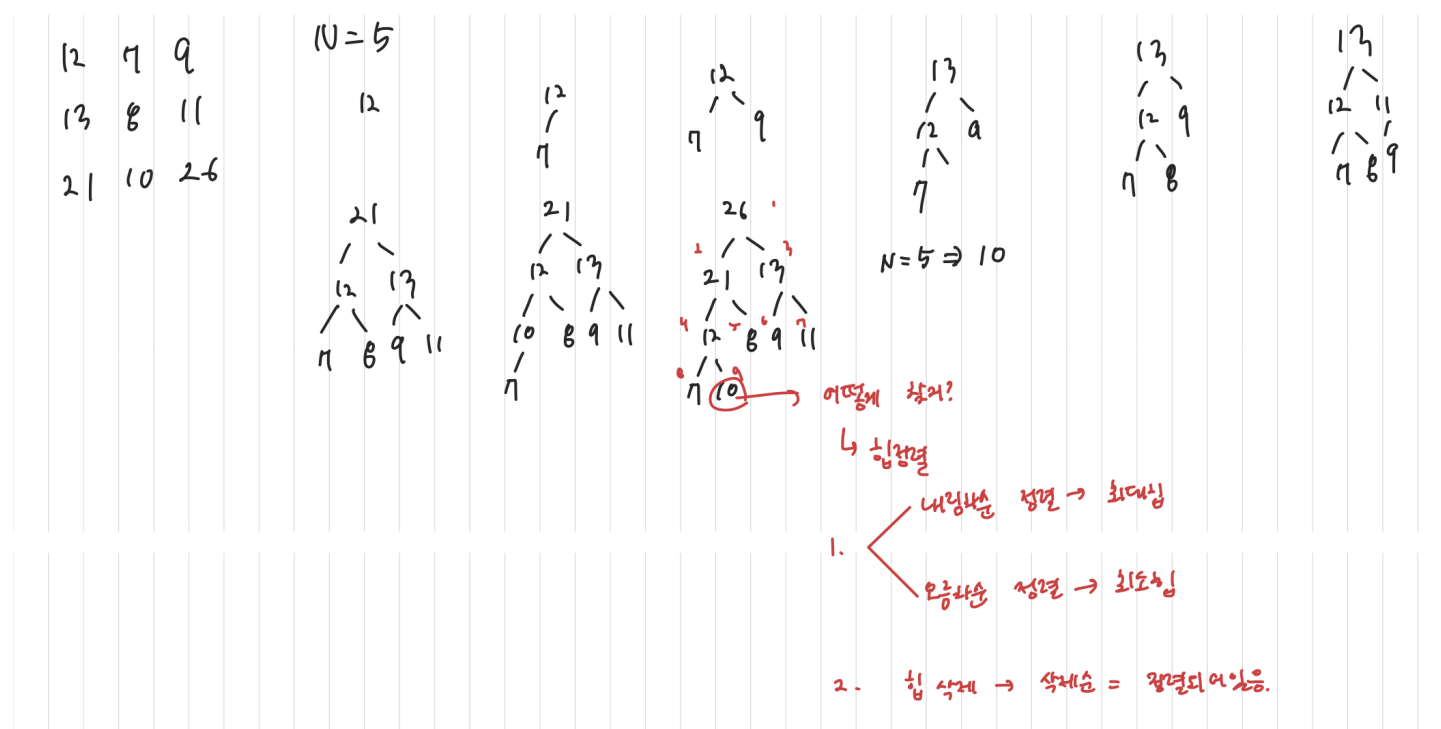
    if (i == N) {
        printf("%d ", b);
    }
}

return 0;
}

```

- 구조체 동적 할당, 구조체의 heap_size 변수 초기화
- 구조체 입력 받으며 insertHeap 함수 실행
- removeHeap 함수를 지속해서 실행하고 처음 입력한 순번이 되면 출력
 - 최대 힙의 루트 노드는 무조건 최대 값 → removeHeap을 지속적으로 실행하면 그 루트노드의 최대 값이 출력

📌 총평



정말 딱 1시간 조금 안걸려서 깜짝 놀랐다. 심지어 저번 스터디 시간에 힙의 오름차순, 내림차순 정렬에 대해 이야기한 적이 있었는데 여기서 이렇게 쓸 줄 몰랐다. 실습 시간에도 pseudo code만 보고 insertheap과 downheap을 적었는데, 이렇게 다시 문제에 응용해보니 다시 복습할 수 있어 좋았다.



향임

코딩 공부 중인 대학생 🧑💻



이전 포스트

백준 | 2696번 중앙값 구하기

0개의 댓글

댓글을 작성하세요

댓글 작성



Powered by
Stellate