2023 Algorithm Study Week 3 - 과제

통계 수정 삭제

cheonwon · 방금 전 · 비공개

0

2023 Algorithm Study smarcle

힙과 힙정렬

이번 포스트를 볼때 뭔가 익숙한 알고리즘들이 보인다면 맞다. 이 코드는 알고리즘 실습시간에 짰던 코드를 일부 변형한 것이다.

N번째로 큰 수 <백준 2075>

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

int n = 0;
int H[2250001];
```

먼저 정수 n과 배열 H를 전역변수로 선언해주었다. 여기서 정수 n은 힙에 들어간 원소의 갯수를, 그리고 배열 H는 힙을 의미하게 될 것이다.

```
void printHeap() {
    for (int i = 1; i <= n; i++) { //첫번째부터 마지막 번째까지 힙의 원소를 출력
        printf("%d ", H[i]);
    }
    printf("\n");
}
```

힙을 전체 출력하는 코드이다. 배열을 이용해 구현한 순차힙이므로 0번원소를 비워뒀기에 1번부터 n번까지 출력이다. 다만, 이건 힙이 잘 구현됐는지 확인하는 용도일 뿐(메소드 구현 테스트용) main함수에서는 사용하지 않는다.

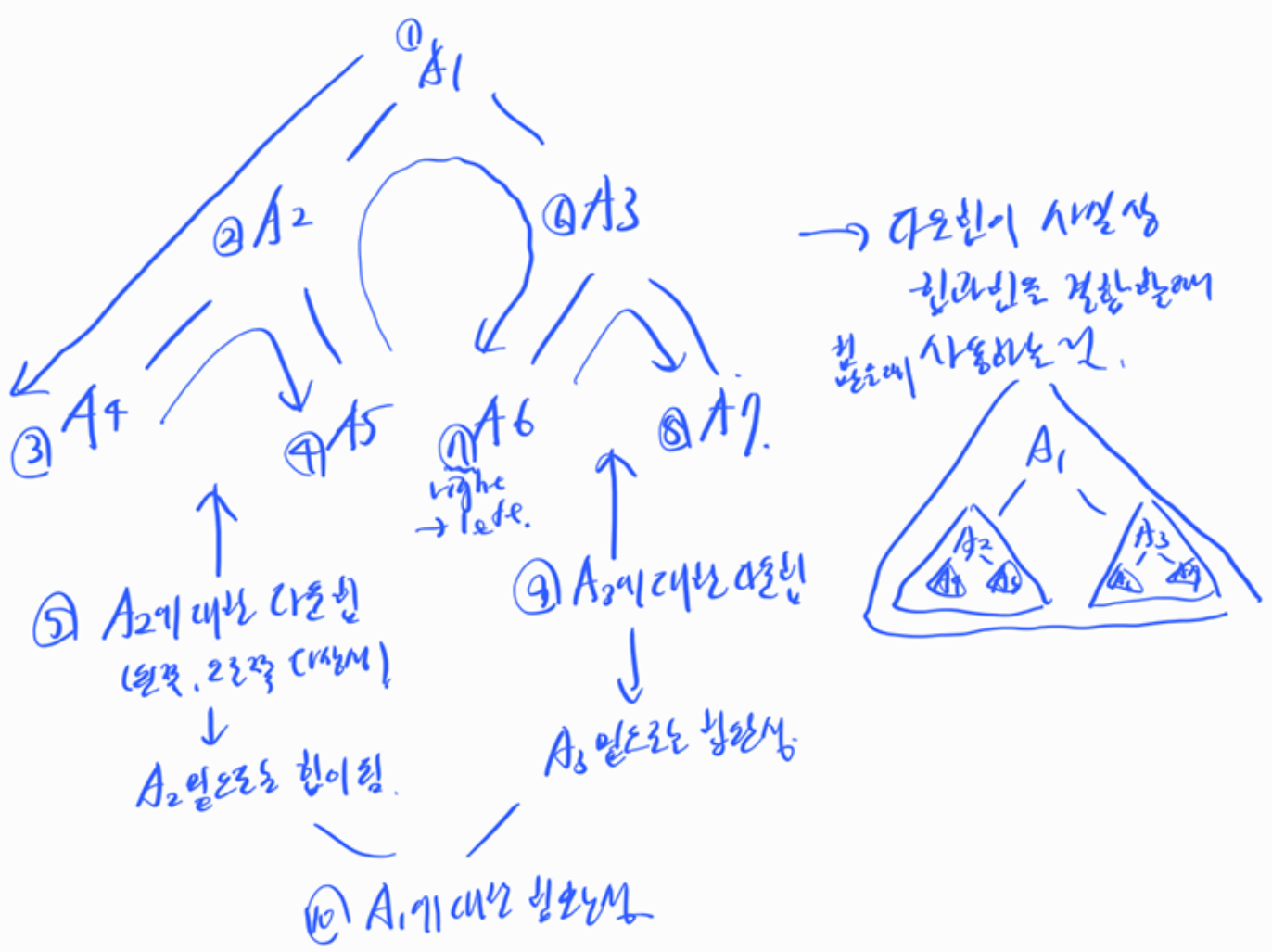
```
void downHeap(int i) {
    //일차적으로 왼쪽 자식과 오른쪽 자식중에서 더 큰값을 찾는 과정
    int left = 2 * i;
    int right = 2 * i + 1; //배열에서 왼쪽과 오른쪽 노드 역할을 하는 인덱스 찾기
    if (left > n) //2i에 해당하는 인덱스가 마지막 번째 인덱스를 넘어서면 종료
        return;
    int child = left; //일단 일차적으로 왼쪽값을 더 크다고 가정한 뒤에
    if (right <= n)
        if (H[child] < H[right]) { //더 큰 값을 찾아서 자식(child)에 저장
            child = right;
        }

    //자식중에서 더 큰 값을 찾았으니, 그 값과 현재노드(부모노드)값을 비교
    if (H[child] <= H[i]) //최대힙에서 부모가 자식보다 값이 크거나 같으면 정상적으로 정렬된것, 종료함.
        return;
    //부모가 자식보다 값이 작으면 둘의 순서를 바꿔서 힙속성 복구
    int tmp = H[i];
    H[i] = H[child];
    H[child] = tmp;
    //이 과정을 계속해서 반복하여 맨 위의 루트노드부터 리프노드까지 정렬(부모의 값이 밑의 두 자식보다 항상 크
    downHeap(child);
}
```

대망의 다운힙이다. 루트노드가 바뀌었을때(removeMax - 루트노드에 맨 밑의 노드값 들어감, rBuildHeap - 두 부분힙을 연결하는 루트노드를 삽입함 등), 맨 위에서부터 힙속성을 복구하면서 밑에까지 내려가는 과정이다. 자식들 중 더 큰쪽이 있는 방향으로 내려가며, 더 큰 자식이 부모보다도 클 경우는 스왑, 부모보다 작을 경우는 그대로 두는 형태이다. 이 과정을 반복한다.

```
int rBuildHeap(int i) {
    if (i > n)
        return n;
    rBuildHeap(2 * i); //왼쪽 힙을 만듬.
    rBuildHeap(2 * i + 1); //오른쪽 힙을 만듬.
    //그것들을 차례대로 정렬시킴(전체 힙을 완성하기 전, 각각의 부분힙부터 하나하나씩 정렬 시작 - 이후 그 힙을
    //가운데 원소, 즉 루트 원소들을 넣어주는 형태이기에 위에서부터 정렬해야 하므로 downheap사용
    downHeap(i);
    return n;
}
```

재귀문 방식을 이용해 구현한 rBuildHeap이며, 상향식 힙 구현 방식이다. 모든 값들이 먼저 다 들어오기에 구현할 수 있는 방법. 먼저 왼쪽힙을, 그리고 오른쪽 힙을 만들고 이를 정렬하는 방식으로 구현한다.



대충 이렇게 생각하면 될것 같다. 왼쪽 아래부터 쭉 정렬하는 방식이다.

```
int removeMax() {
    int tmp = H[1];
    H[1] = H[n];
    n = n - 1;
    downHeap(1);
    return tmp;
}
```

최대힙으로 구현했기에 가장 큰 값이 루트노드, 즉 H[1]에 온다. 따라서 H[1]을 반환하고 맨 밑의 값을 루트에 올린뒤 n을 하나 줄이고 재정렬하면 된다.

```
int main() {
    scanf("%d", &n); //n을 입력받음

    int tmp = n; //n값 미리 저장해둠(전역변수라서 어쩔수 없음)
    n = n * n;
    for (int i = 1; i < n + 1; i++) {
        scanf("%d", &H[i]); //상향식 힙을 만들기 전 기본 배열을 전부 입력받음
    }
```

```
    rBuildHeap(1); //상향식 힙생성(재귀문 이용방식)

    for (int i = 0; i < tmp - 1; i++) {
        removeMax();
    }
    printf("%d\n", removeMax());

    return 0;
}
```

출력예시에 맞게 출력되도록 main함수를 수정해주기만 하면 끝.

개선을 위한 노력

동적할당 하는게 메모리가 더 적게 들것 같아서 동적할당하려 했는데, 오히려 메모리가 근소한 차이지만 좀 더 많이 들고 시간도 좀 더 많이 걸렸다. 4kb 차이가 났는데, 순간 4라는 숫자에 혹시 int하나가 더 할당된거 아니야? 싶어서 코드를 다시보니 문자열 배열 할당하는 버릇이 남아서 main함수에서 n + 1 사이즈로 동적배열을 할당하고 있었다. 아마 이것때문에 오류가 난 것 같다.
->라고 생각하면 틀린것이다. int를 다시 찾아보니 4kb가 아니라 4byte였다.... 혹시 몰라서 n으로 동적할당을 다시 해서 코드도 돌려봤는데, 똑같은 메모리를 소모하는 것을 알 수 있었다.
거기다가 n + 1로 할당하는게 맞았는것 같은게, 우리가 배열로 구현하는 순차힙을 구현하는 특성상 인덱스 0의 위치를 사용하지는 않지만, malloc으로 할당할때는 0까지 포함해서 할당을 해주어야 하기 때문에 n + 1로 선언하는게 정확한 코딩인것 같다.
->근데 막상 n으로 바꿔도 이상없이 잘 채점되던데 왜 그런것인지는 잘 모르겠다. 4kb가 왜 차이가 나는지는 나중에 다시 찾아봐야 겠다.
->거기다가 똑같은 코드라도 백준에 다시 제출하면 시간이 계속 다르게 뜨던데, 왜 그런것인지는 잘 모르겠다. 들어가는 예시가 조금씩 다른가 싶다. 즉, 걸린 시간의 차이는 딱히 의미 없는것 같다.

배열 동적할당 한 버전

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

int n = 0;
int* H = NULL;

void printHeap() {
    for (int i = 1; i <= n; i++) { //첫번째부터 마지막 번째까지 힙의 원소를 출력
        printf("%d ", H[i]);
    }
```

```
        printf("\n");
    }

    void downHeap(int i) {
        //일차적으로 왼쪽 자식과 오른쪽 자식중에서 더 큰값을 찾는 과정
        int left = 2 * i;
        int right = 2 * i + 1; //배열에서 왼쪽과 오른쪽 노드 역할을 하는 인덱스 찾기
        if (left > n) //2i에 해당하는 인덱스가 마지막 번째 인덱스를 넘어서면 종료
            return;
        int child = left; //일단 일차적으로 왼쪽값을 더 크다고 가정한 뒤에
        if (right <= n)
            if (H[child] < H[right]) { //더 큰 값을 찾아서 자식(child)에 저장
                child = right;
            }

        //자식중에서 더 큰 값을 찾았으니, 그 값과 현재노드(부모노드)값을 비교
        if (H[child] <= H[i]) //최대힙에서 부모가 자식보다 값이 크거나 같으면 정상적으로 정렬된것, 종료함.
            return;
        //부모가 자식보다 값이 작으면 둘의 순서를 바꿔서 힙속성 복구
        int tmp = H[i];
        H[i] = H[child];
        H[child] = tmp;
        //이 과정을 계속해서 반복하여 맨 위의 루트노드부터 리프노드까지 정렬(부모의 값이 밑의 두 자식보다 항상 크
        downHeap(child);
    }

    int rBuildHeap(int i) {
        if (i > n)
            return n;
        rBuildHeap(2 * i); //왼쪽 힙을 만듬.
        rBuildHeap(2 * i + 1); //오른쪽 힙을 만듬.
        //그것들을 차례대로 정렬시킴(전체 힙을 완성하기 전, 각각의 부분힙부터 하나하나씩 정렬 시작 - 이후 그 힙을
        //가운데 원소, 즉 루트 원소들을 넣어주는 형태이기에 위에서부터 정렬해야 하므로 downheap사용
        downHeap(i);
        return n;
    }

    int removeMax() {
        int tmp = H[1];
        H[1] = H[n];
        n = n - 1;
        downHeap(1);
        return tmp;
    }

    int main() {
        scanf("%d", &n); //n을 입력받음

        int tmp = n; //n값 미리 저장해둠(전역변수라서 어쩔수 없음)
        n = n * n;

        H = (int*)malloc(sizeof(int) * (n + 1));
        for (int i = 1; i < n + 1; i++) {
            scanf("%d", &H[i]); //상향식 힙을 만들기 전 기본 배열을 전부 입력받음
        }

        rBuildHeap(1); //상향식 힙생성(재귀문 이용방식)
```

```
        for (int i = 0; i < tmp - 1; i++) {
            removeMax();
        }
        printf("%d\n", removeMax());
        free(H);
        return 0;
    }
```

<제출화면>

67135938	cheonwon	2075	맞았습니다!!	9904 KB	504 ms	C99 / 수정	2445 B	28분 전
67135917	cheonwon	2075	맞았습니다!!	9908 KB	492 ms	C99 / 수정	2496 B	29분 전
67135426	cheonwon	2075	맞았습니다!!	9908 KB	540 ms	C99 / 수정	2492 B	38분 전
67135405	cheonwon	2075	맞았습니다!!	9908 KB	488 ms	C99 / 수정	2492 B	38분 전
67135377	cheonwon	2075	맞았습니다!!	9908 KB	548 ms	C99 / 수정	2492 B	39분 전
67134969	cheonwon	2075	맞았습니다!!	9908 KB	504 ms	C99 / 수정	2496 B	47분 전
67134702	cheonwon	2075	맞았습니다!!	9904 KB	500 ms	C99 / 수정	2445 B	52분 전

같은 코드라도 여러번 넣을때마다 제출 시간이 달라지는 것을 확인할 수 있다. 9904가 첫번째 코드로 제출한 결과이고 9908이 두번째 코드로 제출한 결과라고 보면 된다. 중간에 n + 1을 n으로 바꾸기도 했지만 딱히 변화는 없다.

천승원

뭐든지 한걸음씩

이전 포스트

2023 Algorithm Study Week 1

0개의 댓글

댓글을 작성하세요

댓글 작성