

문제해결기법 - 201921725 안성현

부분배열 삭감

<2021/10/01>

문제 소개 및 접근법

1-1] 부분배열 삭감

① 문제

길이가 n 인 배열 A 가 있다. 이 배열의 i 번째부터 j 번째까지 원소들로 이루어진 배열 $A[i], \dots, A[j]$ 를 부분 배열이라고 하고 $A[i,j]$ 로 나타낸다. 이 때 $i \leq j$ 와 $0 \leq i, j \leq n-1$ 이 성립한다. 한 부분 배열 $A[i,j]$ 에 속한 모든 원소를 일률적으로 같은 양 x 만큼 삭감하는 연산을 '부분배열 삭감'이라고 한다.

음이 아닌 정수로 이루어진 배열 $A[0, n-1]$ 이 주어질 때, 모든 원소가 0인 배열이 되도록 하는 삭감 연산의 최소 횟수를 구하라.

(배열은 $n \leq 100,000$ 을 만족하고 각 원소들은 10^9 이하이다.)

② 접근법 (소스 간략 설명)

▶ 최소 횟수를 구하려면 같은 크기의 이웃을 최대한 많이 만들어서 한 번에 삭감해야 한다. 예를 들어 $\{1, 2, 3, 3, 2, 1\}$ 은 $A[2,3]$ 을 1만큼 삭감해서 $\{1, 2, 2, 2, 2, 1\}$ 로 만들면서 2를 최대한 많이 만든다. 그리고 $A[1,4]$ 를 1만큼 삭감해서 $\{1, \dots, 1\}$ 로 만들면서 1을 최대한 많이 만든다. 마지막으로 $A[0,5]$ 를 1만큼 삭감해서 모든 원소를 0으로 만드는 것이다. 즉 크기가 큰 원소들로부터 차례차례 작은 원소들을 늘리는 방식이다. ($3 \rightarrow 2 \rightarrow 1 \rightarrow 0$) 그러기 위해 삭감을 할 때는 같은 크기의 원소들을 한 그룹으로 보고, 원소의 크기가 가장 큰 그룹 기준으로 양 옆 그룹들 중에 그룹 내의 원소 크기가 큰 쪽으로 삭감을 진행한다. 예를 들어 $\{3, 3, 5, 5, 2\}$ 이면 그룹 $\{5, 5\}$ 에서 그룹 $\{3\}$ 과 그룹 $\{2\}$ 를 보고 원소 크기가 큰 그룹 $\{3\}$ 쪽으로 삭감을 해서 $\{3, 3, 3, 3, 2\}$ 를 만든다. 만일 양 그룹의 원소 크기가 같으면 왼쪽 그룹의 값으로 삭감을 진행한다. (왼쪽으로 해야 코딩이 더 쉬움) 이렇게 첫 번째 삭감을 진행하였으면 업데이트된 A배열에서 제일 크기가 큰 원소를 찾고 같은 방식으로 삭감을 진행한다. 하지만 앞의 예제에 따르면 그룹 $\{3, 3, 3, 3\}$ 의 왼쪽 그룹은 존재하지 않는다. 따라서 A배열의 양쪽 끝에 (-1) 이라는 크기 비교용 원소가 있다고 생각하고 문제를 푼다. 그러면 그룹 $\{3, 3, 3, 3\}$ 이 그룹 $\{-1\}$ 과 그룹 $\{2\}$ 중에서 원소 크기가 큰 그룹 $\{2\}$ 쪽으로 삭감을 해서 $\{2, 2, 2, 2, 2\}$ 를 만든다. 이렇게 어떤 특정한 수가 n 번 존재하면 전체에서 해당 수를 삭감하면 된다. 단 0일 때

는 제외한다. 이 로직대로 문제를 풀기 위해서는 빈도수만큼 삭감을 하므로 frequency 리스트가 필요하고 그 빈도수가 무슨 원소인지 나타내는 infor 리스트가 필요하다. 예를 들어 {3,3,5,5,2}의 infor리스트는 {3,5,2}이며 frequency 리스트는 {2,2,1}이다. A배열이 업데이트 돼서 {3,3,3,3,2}가 되면 infor리스트는 {3,2}, frequency 리스트는 {4,1}이 된다. 하지만 매번 A배열을 업데이트하고 frequency, infor리스트를 구상하는 것은 매우 비효율적이다. 사실 이 두 리스트 값을 조정하는 것 만으로도 A배열을 업데이트한 것과 같은 효과를 준다. infor리스트에서 한 원소는 A배열의 한 그룹과도 같다. 따라서 infor리스트에서 최대값을 찾고 그 최대값을 가지는 원소 기준으로 오른쪽과 왼쪽 원소의 크기를 비교해서 큰 쪽으로 frequency를 더해준다. 이후 해당 원소는 두 리스트에서 사용하지 않는다. 예를 들어 초기 infor리스트에서 최대값인 infor[1]를 기준으로 infor[0]이 infor[2]보다 크니까 frequency[0]+frequency[1]을 진행한다. 이후 infor[1], frequency[1]은 사용하지 않으니 frequency={4,2,1}은 {4,1}로 표현할 수 있다. 리스트에서 값을 아예 삭제할 수도 있지만 사용하지 않는다고 처리하는 것이 더 효율적인 코드이다. (삭제하면서 리스트 원소를 앞으로 뺄지 않아도 됨) 단, 사용하지 않다고 처리하기 때문에 right와 left필드가 필요하다. infor[0]의 right는 2가 돼서 앞으로 오른쪽 이웃은 infor[2]를 보게 하고 infor[2]의 left는 0이 돼서 앞으로 왼쪽 이웃은 infor[0]을 보게끔 처리하는 것이다. 삭감을 진행하고 다시 최댓값을 찾을 때는 Delete_max_heap함수를 이용한다. max에 해당하는 인덱스를 반환했는데 해당 원소가 사용하지 않는다고 처리된 것이면 다시 함수를 호출한다. 이렇게 최댓값을 찾으면 선형탐색으로 찾는 것보다 훨씬 효율적이라고 단언한다. 지금까지 코드 이해에 필요한 모든 아이디어를 제시하였다.

소스 코드와 실행 결과

2-1] 소스 코드와 실행 결과

① 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#pragma warning(disable:4996)
#define max 100002

typedef struct {
    int key;
    int list_idx;
}helement;

typedef struct {
    helement heap[max];
    int heap_size;
}HeapType;
typedef HeapType* HeapType_ptr;

// 힙 생성
HeapType_ptr hcreate() {
    return (HeapType*)malloc(sizeof(HeapType));
}

// 힙 초기화
void hinit(HeapType_ptr h) {
    h->heap_size = 0;
}

// 힙 삽입
void insert_max_heap(HeapType_ptr h, helement item) {
    int i;
    i = ++(h->heap_size);
    while ((i != 1) && (item.key > h->heap[i / 2].key)) {
        h->heap[i] = h->heap[i / 2];
        i /= 2;
    }
    h->heap[i] = item;
}

// 힙 최댓값 삭제
helement delete_max_heap(HeapType_ptr h) {
    int parent, child;
    helement item, temp;
    item = h->heap[1];
```

```

temp = h->heap[(h->heap_size)--];
parent = 1;
child = 2;
while (child <= h->heap_size) {
    if ((h->heap[parent * 2].key >= 0) && (h->heap[(parent * 2) + 1].key >=
0)) {
        if ((h->heap[parent * 2].key) > (h->heap[(parent * 2) + 1].key))
            child = parent * 2;
        else
            child = (parent * 2) + 1;
    }
    if (temp.key >= h->heap[child].key)
        break;
    h->heap[parent] = h->heap[child];
    parent = child;
    child *= 2;
}
h->heap[parent] = temp;
return item;
}

typedef struct {
    int infor;
    int freq;
    int left;
    int right;
    bool check;
}element;

typedef struct {
    element array[max];
    int size;
}ArrayListType;
typedef ArrayListType * ArrayList_ptr;

// 리스트 초기화
void init(ArrayList_ptr list) {
    list->size = 0;
}

// 리스트 생성
ArrayList_ptr create() {
    return (ArrayList_ptr)malloc(sizeof(ArrayListType));
}

// 빈 리스트 확인
int is_empty(ArrayList_ptr list) {
    if (list->size == 0)
        return 1;
    else
        return 0;
}

```

```

// 풀 리스트 확인
int is_full(ArrayList_ptr list) {
    if (list->size == max)
        return 1;
    else
        return 0;
}

// 아이템 삽입
void insert(ArrayList_ptr list, element item) {
    if (!is_full(list)) {
        int pos = list->size;
        list->array[pos] = item;
        list->size++;
    }
    else
        printf("리스트가 꽉 차있습니다\n");
}

// 인덱스로 아이템 삭제 (사용x)
void delete_idx(ArrayList_ptr list, int pos) {
    if (!is_empty(list)) {
        element tmp = list->array[pos];
        for (int i = pos + 1; i < list->size; i++)
            list->array[i - 1] = list->array[i];
        list->size--;
    }
    else
        printf("리스트가 비어 있습니다\n");
}

// 리스트 출력 (사용x)
void print_list(ArrayList_ptr list) {
    printf("\ninfor:");
    for (int i = 0; i < list->size; i++) {
        if (list->array[i].check) {
            printf("[%d(%d)] ", list->array[i].infor, i);
        }
    }
    printf("\nfreq:");
    for (int i = 0; i < list->size; i++) {
        if (list->array[i].check) {
            printf("[%d(%d)] ", list->array[i].freq, i);
        }
    }
}

// 부분배열 삭감
int decrease(int* array, int n) {
    ArrayList_ptr list = create(); // 리스트 생성
    init(list); // 초기화

    HeapType_ptr heap = hcreate(); // 힙 생성

```

```

hinit(heap); // 초기화

int num = 0;
int i;

// 값,빈도수,좌우 인덱스 구하기
for (i = -1; i <= n; i++) {
    if (i == -1 || i == n) { // 단순히 크기 비교를 위해서 사용
        element e = { -1,-1, -1, -1,false };
        insert(list, e);
    }
    else if (i == 0 || array[i] != array[i - 1]) {
        element e = { array[i],1,list->size - 1,list->size + 1,true };
        helement h = { array[i], list->size };
        insert(list, e);
        insert_max_heap(heap, h);
    }
    else {
        list->array[list->size - 1].freq++;
    }
}

while (1) {
    // 제일 큰 값을 기준으로 시작
    int idx = 0;
    while (!list->array[idx].check) {
        idx = delete_max_heap(heap).list_idx;
    }
    int left_idx = list->array[idx].left;
    int right_idx = list->array[idx].right;

    // 마지막 과정 (한 숫자가 n개)
    if (list->array[idx].freq == n) {
        if (list->array[idx].infor == 0) {
            return num;
        }
        else {
            return ++num;
        }
    }

    // 오른쪽이 더 큼 -> 오른쪽 수로 frequency번 채움
    else if (list->array[left_idx].infor < list->array[right_idx].infor) {
        list->array[right_idx].freq += list->array[idx].freq;
        list->array[left_idx].right = right_idx; // 왼쪽 요소의 right는
오른쪽 요소

        list->array[right_idx].left = left_idx; // 오른쪽 요소의 left는
왼쪽 요소

        list->array[idx].check = false;
        num++;
    }

    // 왼쪽이 더 큼 -> 왼쪽 수로 frequency번 채움

```

```

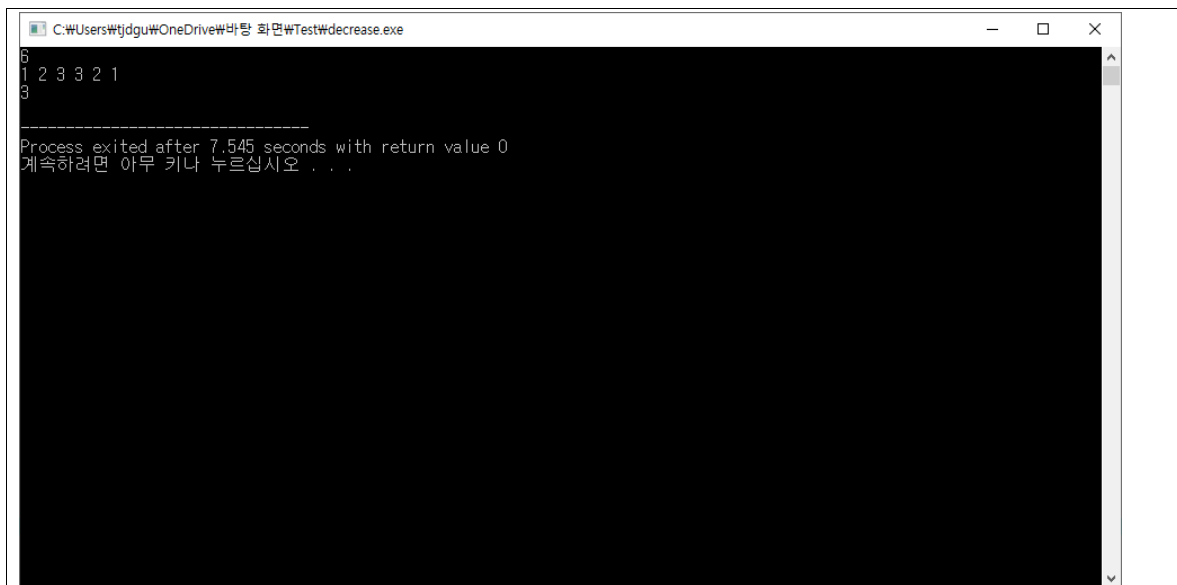
        else if (list->array[left_idx].infor > list->array[right_idx].infor) {
            list->array[left_idx].freq += list->array[idx].freq;
            list->array[left_idx].right = right_idx; // 왼쪽 요소의 right는
오른쪽 요소
            list->array[right_idx].left = left_idx; // 오른쪽 요소의 left는
왼쪽 요소
            list->array[idx].check = false;
            num++;
        }

        // 오른쪽과 왼쪽이 같음
        else {
            // 왼쪽으로 frequency를 몰아줌
            list->array[left_idx].freq += list->array[right_idx].freq;
            list->array[left_idx].freq += list->array[idx].freq;
            list->array[left_idx].right = list->array[right_idx].right; //
왼쪽 요소의 right는 오른쪽의 오른쪽 요소
            list->array[list->array[right_idx].right].left = left_idx; //
오른쪽의 오른쪽 요소의 left는 왼쪽 요소
            list->array[idx].check = list->array[right_idx].check = false;
            num++;
        }
    }
}

int main() {
    int n = 0;
    scanf("%d", &n);
    int* array = (int*)malloc(sizeof(int) * n);
    int i;
    for (i = 0; i < n; i++) {
        scanf("%d", &array[i]);
    }
    printf("%d\n", decrease(array, n));
    return 0;
}

```

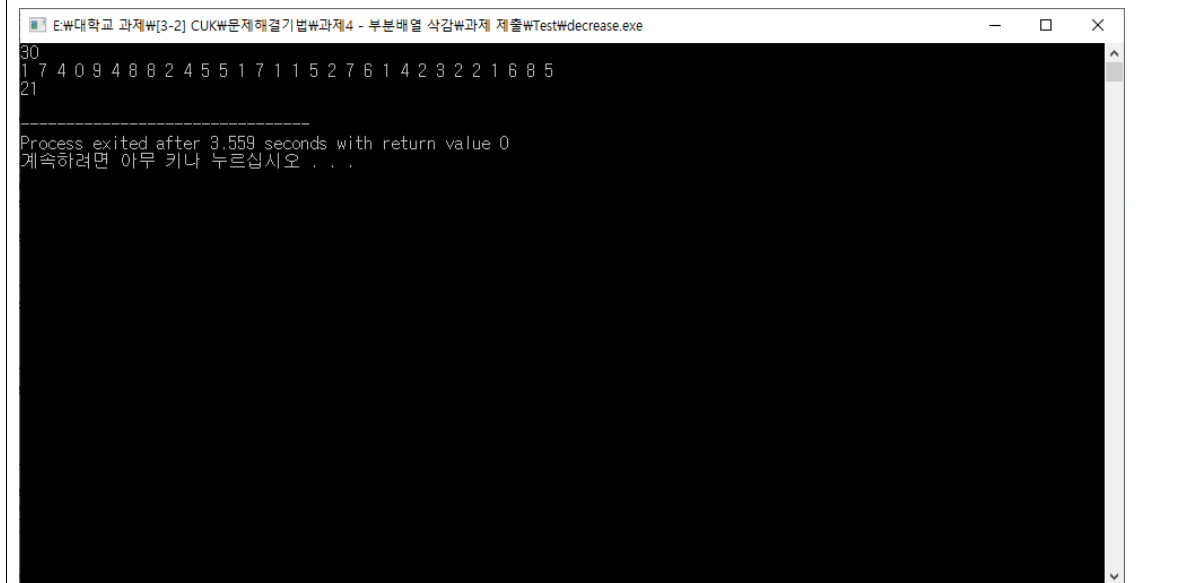

② DEV-C++ 컴파일러 이용 실행 결과



```
C:\Users\jtdgu\OneDrive\바탕 화면\Test\decrease.exe
6
1 2 3 3 2 1
3
-----
Process exited after 7.545 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

입력: $n=6$, 원소={1,2,3,3,2,1}

출력: 3



```
E:\대학과 과제\3-2] CUK\문제해결기법\과제4 - 부분배열 삭감\과제 제출\Test\decrease.exe
30
1 7 4 0 9 4 8 8 2 4 5 5 1 7 1 1 5 2 7 6 1 4 2 3 2 2 1 6 8 5
21
-----
Process exited after 3.559 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

입력: $n=30$, 원소=9미만의 음이 아닌 정수들

출력: 21