가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA Computer SCIENCE

알고리즘 설계 - 201921725 안성현

영화 스케줄

<2021/03/31>

문제 소개 및 접근법

1-1] 영화 스케줄

① 문제

당신은 인기있는 영화배우이고, n개의 서로 다른 영화에 출연 요청을 받았다고 상상해보자. 각 출연 요청에는 촬영한 첫 날과 마지막 날이 명시되어 있고, 출연하면 얻을 수 있는 수입이 제시되어 있다. 요청을 수락하기 위해서는 반드시 촬영기간 내내 출연할 수 있어야 한다. 따라서 촬영 기간이 겹치는 두 촬영 작업을 동시에 수락할 수는 없다. 어떤 두 작업도 서로 기간이 겹치지 않도록 하면서 얻을 수 있는 최대 수입을 구하는 효율적인 프로그램을 작성하시오.

(첫째 줄에 n(n<=50,000)이 입력된다. 둘째 줄부터 마지막 줄까지 영화에 대한 정보를 나타내는 음이 아닌 세 정수 (a_i, b_i, p_i)가 순서대로 주어진다. a_i와 b_i는 1억 이하이고 a_i <= b_i이다. p_i는 40,000 이하이다.)

② 접근법 (소스 간략 설명)

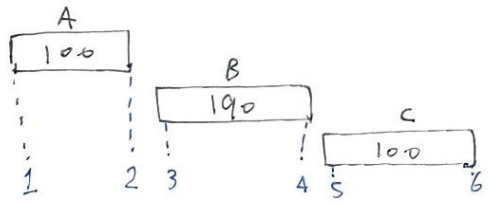
▶ 동적계획법으로 문제를 접근했다. 한 영화를 출연했을 때 최대 소득을 구하고 해당 값을 '다음 영화 출연할 시 최대 소득'을 구하는 데 이용하는 것이다. (경우에 따라 이용하지 않을 수도 있음) 이 작업을 마지막 영화까지 이어 나갔을 때, 각 영화마다 구한 최대 소득 중 최댓값을 구하면 된다. 이러한 작업에 필요한 일반화 식은

MAX(시작 일 이전에 얻을 수 있는 최대 소득 + 이 영화를 출연하면 얻는 소득, 이전 최대 소득)

이 된다. 이 MAX 값 계산을 모든 영화에 대해서 구하면 되는데 여기서 주의할 점이 있다. 바로 마지막 날(b_i)을 기준으로 정렬을 해야 된다. i번째 영화를 시작하기 전에 끝나는 영화들은 모두 0~i-1번째에 위치되어야만 한다. 그래야 '시작 일 이전에 얻을 수 있는 최대 소득'을 구하는 데 지장이 없다.

지금부터 예시 3개를 이용해서 설명을 이어 나가겠다. 참고로 a_i는 영화의 첫 날이므로 start, b_i는 영화의 마지막 날이므로 end, p_i는 수입이므로 income, 시작 일 이전에 얻을 수 있는 최대 소득은 c_max로 작성할 예정이다.

(예시 1) 아래와 같이 영화 세 개 {(1,2,100),(3,4,190),(5,6,100)}가 있다고 가정하겠다.



A 영화를 출연한다고 생각하자. 그러면 MAX(0+100, 0)=100이 최대 소득이다.

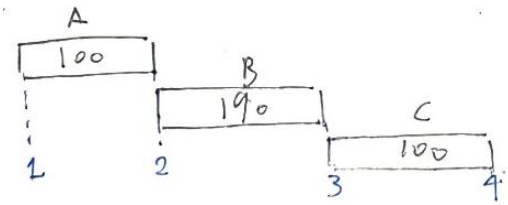
B 영화를 출연하다고 생각하자. 그러면 MAX(100+190, 100)=290이 최대 소득이다.

C 영화를 출연한다고 생각하자. 그러면 MAX(290+100, 100)=390이 최대 소득이다.

모든 영화에 대한 MAX 값을 구하였으니 최대 소득은 390이 된다.

이러한 방식을 **코드**로 구현하기 위해 영화 개수를 크기로 가지는 v배열이 있다고 생각하자. 이제 영화의 c_max를 구하고 income만큼 더한 값을 v배열에 저장할 것이다. A는 첫 영화이니 v[0]은 0+100이 된다. B를 기준으로 c_max은 100이라고 생각할 수 있는데 이 값은 v[0]에 이미 저장하였으니 v[0]+190=290이 v[1]에 저장된다. 마찬가지로 C는 v[1]+100=390이 v[2]에 저장된다.

(예시 2) 아래와 같이 영화 세 개 {(1,2,100),(2,3,190),(3,4,100)}가 있다고 가정하겠다.



A 영화를 출연한다고 생각하자. 그러면 MAX(0+100, 0)=100이 최대 소득이다.

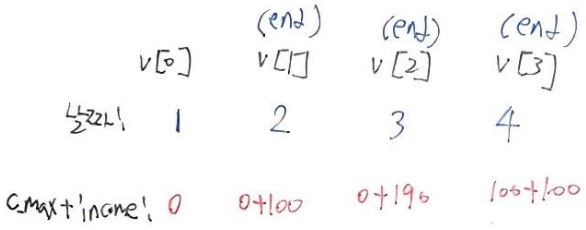
B 영화를 출연하다고 생각하자. 그러면 MAX(0+190, 100)=190이 최대 소득이다.

C 영화를 출연한다고 생각하자. 그러면 MAX(100+100, 190)=200이 최대 소득이다.

모든 영화에 대한 MAX 값을 구하였으니 최대 소득은 200이 된다.

(예시1)과는 달리 영화가 서로 겹쳐 있는 모습이다. 같은 시간은 동시 출연이 불가능하니 **앞처럼 단순히 영화의 개수를 배열 개수로 생각하면 안 된다.** (예시1에 제시된 방법으로 B입장에서 v[0]=100을 이용하면 틀린다.) 따라서 **이번에는 start, end에 대한 집합 크기를 배열의 개수로 생각하였다.** 각 배열은 (날짜, c_max+income)이 저장된다. 그리고 값을 저장할 때는 날짜가 end에 해당하는 배열 원소를 이용하였다. A영화의 start인 1이전에는 아무 영화도 없으므로 v[날짜=2]는 0+100이 된다. B영화의 start인 2이전에 영화가 있는지 없는지 확인은 v[날짜<2]을 이용한다. 이 때 값이 없으므로 v[날짜=3]은 0+190이 된다. 같은 이치로 C영화의 확인은 v[날짜<3]를 이용한다. v[날짜=2]에 값이 있으므로 v[날짜=4]는 100+100이 된다. **값이 있다는 의미는 해**

당 영화 시작 이전에 끝난 영화(들)이 있다는 것이다. 이러한 '끝난 영화(들)에서 구한 최대 소득' 중 **최댓값**이 바로 c_max(시작 일 이전에 얻을 수 있는 최대 소득)이다. 결과를 그림으로 나타내면 아래와 같다.

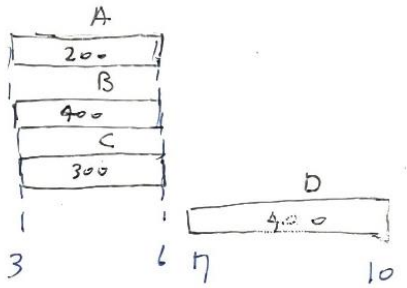


위 예시에서 '끝난 영화'는 A 영화 뿐이다. 그런데 만약 B영화의 end가 C영화의 start와 겹치지 않으면 어떻게 될까? 이 때는 (100,190) 중 큰 값을 채택하는 것이다. 더 어려운 예시가 주어진다면 약 n개의 영화의 최대 소득 중 큰 값을 채택해야 될 수도 있다. 또한 이러한 **최대값 탐색을 여러 번** 해야 될 수도 있다. 이럴 때 유용한 자료구조가 **'세그먼트 트리'**이다.

정리해서 말하면 c_max(시작 일 이전에 얻을 수 있는 최대 소득)를 구하기 위해 '끝난 영화들의 최대 소득'에 대한 최댓값 탐색을 하는 데, 영화마다 c_max를 구해야 되니 최댓값 탐색을 여러 번 한다는 것이다. 따라서 <u>최댓값 탐색을 O(logn)에 할 수 있는 세그먼트 트리를 이용</u>한다.

수식: MAX('c_max+ income' of v[날짜<start])

(예시 3) 아래와 같이 영화 네 개 {(3,6,200), (3,6,400), (3,6,300), (7,10,400)}가 있다고 가정하겠다.



A 영화를 출연한다고 생각하자. 그러면 MAX(0+ 200, 0)=200이 최대 소득이다.

B 영화를 출연하다고 생각하자. 그러면 MAX(0+ 400, 200)=400이 최대 소득이다.

C 영화를 출연한다고 생각하자. 그러면 MAX(0+ 300, 400)=400이 최대 소득이다.

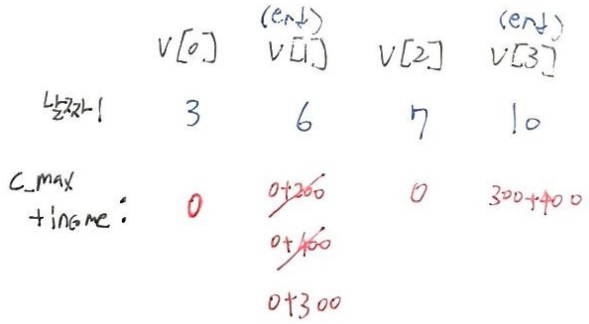
D 영화를 출연한다고 생각하자. 그러면 MAX(400+ 400, 400)=800이 최대 소득이다.

모든 영화에 대한 MAX 값을 구하였으니 최대 소득은 800이 된다.

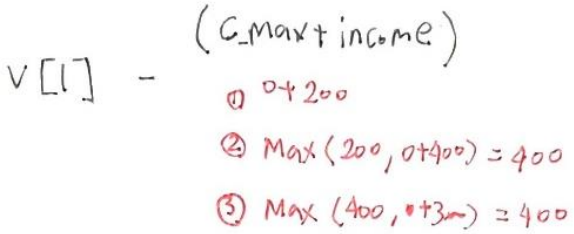
(예시2) 방법을 그대로 이용해보겠다.

A영화의 확인은 v[날짜<3]를 이용한다. 값이 없으므로 v[날짜=6]은 0+ 200이 된다.

B영화의 확인은 v[날짜<3]를 이용한다. 값이 없으므로 v[날짜=6]은 0+400이 된다.
C영화의 확인은 v[날짜<3]를 이용한다. 값이 없으므로 v[날짜=6]은 0+300이 된다.
D영화의 확인은 v[날짜<7]를 이용한다. v[날짜=6]에 값이 있으므로 v[날짜=10]은 300+400이 된다. 결과를 그림으로 나타내면 아래와 같다.



그런데 정답은 800이니 (예시2)의 방법을 그대로 따르면 문제가 있다. 따라서 배열을 업데이트 할 때 기존의 값이 더 크다면 더 큰 값을 유지하는 방법을 택한다.



수식: v[날짜=end]=MAX(c_max+ income, v[날짜=end])

지금까지의 설명을 정리하면 다음과 같다.

1) max_income = MAX(c_max+ income, max_income)
2) c_max = MAX('c_max+ income' of v[날짜<start])
3) v[날짜=end]=MAX(c_max+ income, v[날짜=end]), v={(날짜,c_max+ income), …}

실제 코드랑 비교해서 차이점이 있다면 v배열의 원소에 (날짜, c_max+ income)이 있지 않고 **v배열의 원소는 날짜, 세그먼트 트리의 leaf node가 c_max+ income이라는 사실이다.**

소스 코드와 실행 결과

2-1] 소스 코드와 실행 결과

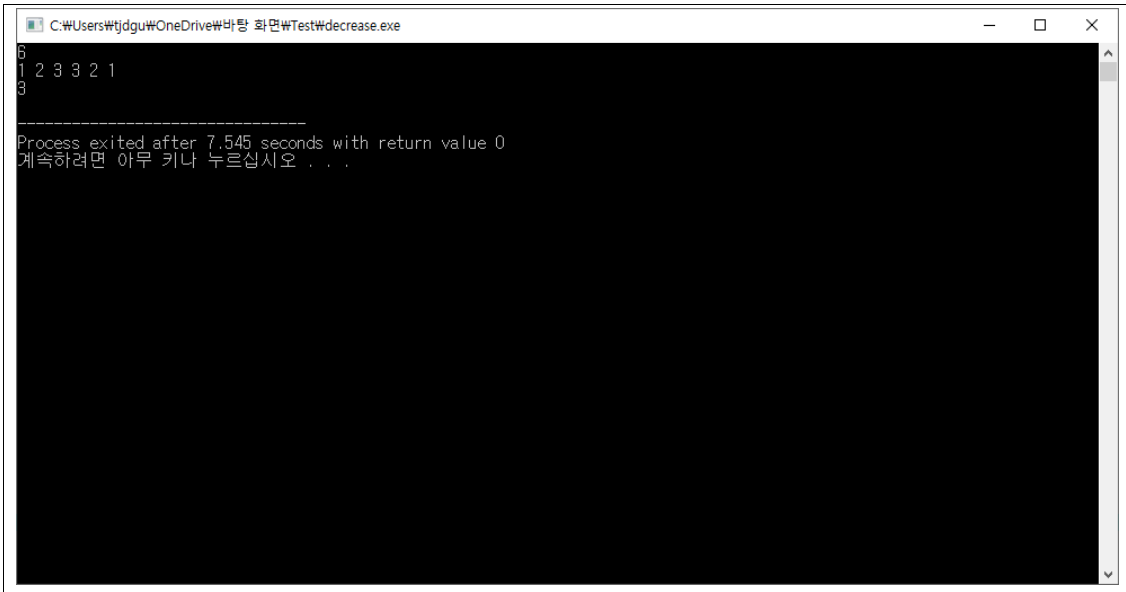
① 코드

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;
int* s_tree; // 세그먼트 트리

typedef struct Movie {
        int start, end, income;
}Movie;


// 끝 시간을 기준으로 오름차순
bool compare(Movie& m1, Movie& m2) {
        return m1.end < m2.end;
}

// 구간에서 최댓값 찾기
int ret(int node, int s, int e, int l, int r) {
        if (l > e || r < s)
                return 0;

        if (l <= s && e <= r)
                return s_tree[node];

        int m = (s + e) / 2;
        return max(ret(node * 2, s, m, l, r), ret(node * 2 + 1, m + 1, e, l, r));
}

// 세그먼트 트리 업데이트
int update(int node, int s, int e, int idx, int val) {
        if (s <= idx && idx <= e) {
                if (s == e) {
                        /*
                                기존 세그먼트 트리는 val값을 저장하지만 이 코드는 max값을
비교해서 저장함
                                이 세그먼트 트리에서 leaf node는 배열의 원소가 아니라
배열의 원소와 관련된 어떤 값(c_max+income)이 나와야 됨
                                start, end 값들을 압축해서 사용하므로 여러 개의 start에
대해 한 개의 end가 나올 수 있음
                                ex) 영화[start,end,income] = 영화[9,13,300],
영화[10,13,100], 영화[14,16,200] 이 있다고 가정하겠음
```

```
                                        영화[14,16,200]의 '현재 시작 시간 기준 최댓값(c_max)'은
300이 돼야 함
                                        그런데 영화[10,13,100]을 확인하면서 v[end_idx]=13에 대한
세그먼트 트리 값이 100으로 업데이트 됨
                                        따라서 중복된 end에 대한 세그먼트 트리 값은 최댓값(300)이
되어야만 함!!
                        */
                        s_tree[node] = max(s_tree[node], val);
                }
                else {
                        int m = (s + e) / 2;
                        s_tree[node] = max(update(node * 2, s, m, idx, val), update(node
* 2 + 1, m + 1, e, idx, val));
                }
        }

        return s_tree[node];
}

// 세그먼트 트리 출력
void sprint(int size) {
        for (int i = 1; i < size; i++) {
                cout << "[" << i << "] " << s_tree[i] << " ";
        }
        cout << endl;
}

// 이진 탐색 (x -> find v_vector_index)
int find(vector<int> &v, int x) {
        int low = 0;
        int high = v.size() - 1;
        int mid;
        while (low <= high) {
                mid = (low + high) / 2;
                if (v[mid] == x)
                        return mid;
                else if (x > v[mid])
                        low = mid + 1;
                else
                        high = mid - 1;
        }
        if (low > high)
                return -1;
}

int main(void)
{
        // 입/출력 최적화
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);

        int n; // 영화의 개수
        vector<Movie> mv; // 영화 자료구조 (start,end,income)
```

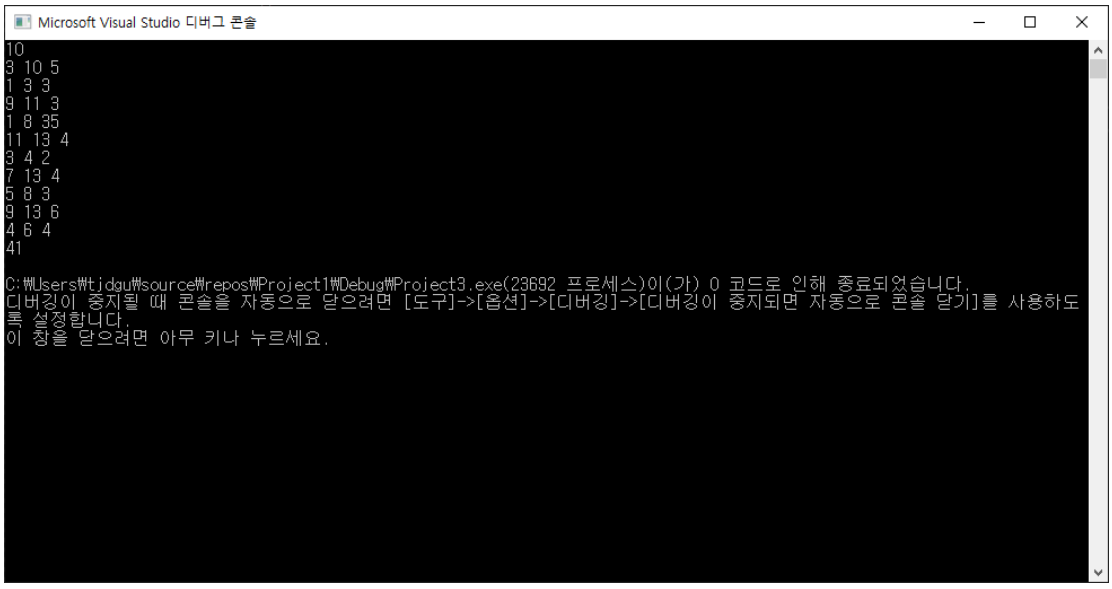
```
        vector<int>v; // 영화의 start, end가 담긴 배열

        // 변수,벡터 초기화
        cin >> n;
        for (int i = 0; i < n; i++) {
                int s, e, ic;
                cin >> s >> e >> ic;
                Movie M = { s, e, ic };
                mv.push_back(M);
                v.push_back(s);
                v.push_back(e);
        }

        // 벡터 정렬
        sort(mv.begin(), mv.end(), compare); // end 기준 오름차순
        sort(v.begin(), v.end());

        // 중복되는 원소 삭제
        v.erase(unique(v.begin(), v.end()), v.end());

        // 세그먼트 트리 생성
        int h = (int)ceil(log2(v.size()));
        int tree_size = (int)pow(2, h + 1);
        s_tree = new int[tree_size] {};

        // 세그먼트 트리로 최대 수입 얻기
        int max_idx = v.size() - 1;
        int max_income, sum, c_max, s_idx, e_idx;
        max_income = sum = c_max = s_idx = e_idx = 0;

        for (int i = 0; i < n; i++) {
                // 시작, 끝 시간을 보고 벡터 인덱스 찾기
                s_idx = find(v, mv[i].start);
                e_idx = find(v, mv[i].end);
                // 현재 시작 시간 기준 최댓값 찾기
                c_max = ret(1, 0, max_idx, 0, s_idx - 1);
                // MAX(현재 시작 시간 기준 최댓값 + 현재 영화 소득, 이전 최대 소득)
                sum = c_max + mv[i].income;
                max_income = max(max_income, sum);
                // 끝 시간 위치에 sum값 저장
                update(1, 0, max_idx, e_idx, sum);
        }

        // 최대 수입 출력
        cout << max_income << endl;

        // 동적 메모리 해제
        delete[] s_tree;

        return 0;
}
```

② DEV-C++ 컴파일러 이용 실행 결과



입력: n=10, 원소={1,2,3,3,2,1}

출력: 3



입력: n=10, 10개의 영화 정보 (시작, 끝, 수입)

출력: 41