#14501, 15486: 퇴사 전 까지 돈을 최대로 벌 수 있도록 스케줄을 작성한다.

sianke1991

7일간 상담 일정표를 통해 7일간 최대로 돈을 벌 수 있는 스케줄을 작성한다.

날짜	1	2	3	4	5	6	7	8
소요 시간	3	5	1	1	2	4	2	
급여	10	20	10	20	15	40	200	

1일의 상담을 진행한 경우 3일의 시간이 걸리므로 2일 째, 3일 째의 상담은 진행할 수 없다.

int[] dpCache을 선언하여 해당 날짜, 상담을 진행하기 전 기준 보수의 최대치를 저장해 둔다.

날짜	1	2	3	4	5	6	7	8
소요 시간	3	5	1	1	2	4	2	
급여	10	20	10	20	15	40	200	
dpArray				10	10	10	10	10

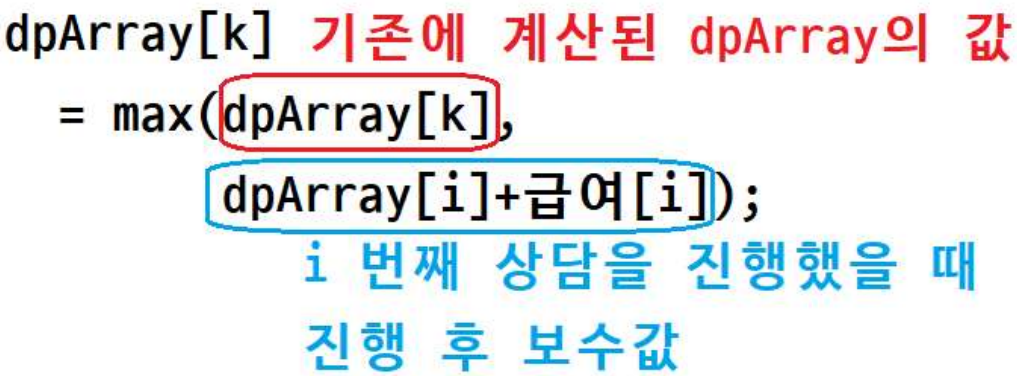
예를 들어 1일 째 상담을 진행한 경우 4일 째 부터 dpArray의 값을 10으로 설정한다. 비슷하게 2일 째, 3일 째 상담을 고려하여 dpArray를 계속 갱신해 나가면 다음과 같이 된다.

날짜	1	2	3	4	5	6	7	8
소요 시간	3	5	1	1	2	4	2	
급여	10	20	10	20	15	40	200	
dpArray,1				10	10	10	10	10
dpArray,2				10	10	10	20	20
dpArray,3				10	10	10	20	20
dpArray,4				10	30	30	30	30
dpArray,5				10	30	30	45	45
dpArray,6				10	30	30	45	45

마지막 상담까지 고려한 뒤 dpArray[8] = 45가 퇴사 후의 보수의 최대치가 된다.

i일 째 상담을 고려하는 경우 int j = i + 소요 시간[i]에 대해 dpArray[j] 부터 갱신이 될 수 있다. 정확히는 다음과 같이 갱신이 이루어진다.

dpArray[k] = max(dpArray[k], dpArray[i]+급여[i]), k>=j



따라서 전체 코드는 다음과 같이 작성된다.

```
for (int i=0; i<날짜수; i++) {
    int j = i+소요 시간[i];
    for (int k=j; k<=날짜수; k++) { //i 번째 일을 잡으면 그 보수는 i+timeReq[i] 번째 날에 받는 것으로 계산한다.
        //마지막 날에 종료되는 일도 보수를 받을 수 있으므로 dpCache[N]에 값을 저장한다.
        dpArray[k] = max(dpArray[k], dpArray[i]+급여[i]); //i 번째 일을 하면 받은 보수는 dpArray[i]+급여[i]이므로 이 값을 dpArray[k]와 비교하여 최대치를 dpArray에 저장한다.
    } //k loop
} //i loop
cout << dpArray[날짜수];
```

하지만 이 방식은 날짜수^{2}에 비례하는 시간이 소요되므로 날짜수가 큰 경우 시간을 많이 소요하게 된다. 하지만 더 빠른 코드를 적용하기 위해 곧바로 안쪽 반복문을 적용하지 않으면 잘못된 답을 내놓게 된다.

잘못된 코드의 예시-a

```
for (int i=0; i<날짜수; i++) {
    int j = i+소요 시간[i];
    if (j>=날짜수) continue; //ArrayIndexOutOfBounds 문제를 회피하기 위한 문장
    dpArray[j] = max(dpArray[j], dpArray[i]+급여[i]); //dpArray를 갱신할 때 "휩쓸어서" 갱신하지 않으므로 dpArray[i]가 곧바로 i 번째 날, 상담을 진행하기 전 기준 보수의 최대치라고 볼 수 없다.
} //i loop
```

날짜	1	2	3	4	5	6
소요 시간	1	2	3	4	5	
급여	10	20	30	40	50	
dpArray, 1	0	10 (b)	0	0	0	0
dpArray, 2	0	10	0	30	0	0
dpArray, 3	0	10	0	30	0	30 (a)
dpArray, 4	0	10	0	30	0	30

a: 3일 째 상담을 진행한 결과 30이라고 갱신이 되었지만 실은 1일 째 상담을 진행한 뒤 3일 째 상담을 진행하므로 40으로 갱신이 되어야 한다. 이 값이 30으로 갱신이 되는 이유는 1일 째 상담을 진행한 결과가 b만 갱신을 했기 때문이다.

날짜	1	2	3	4	5	6
소요 시간	1	2	3	4	5	
급여	10	20	30	40	50	
dpArray, 1	0	10 (b)	0	0	0	0

따라서 i 번째 상담을 고려하는 경우 dpArray[i] 까지의 부분 배열에서 최대치를 알고 있어야 한다. 이를 위해 부분 최대치를 저장하는 변수를 마련해 주자. 이 값은 상담을 하나 씩 고려할 때 마다 갱신되는 값이다. (i 번째 상담을 고려하는 중이라면 dpArray[i] 까지의 부분 배열의 최대치)

잘못된 코드의 예시-b

```
int 부분 최대치 = 0;
int 전체 최대치 = 0;
for (int i=0; i<날짜수; i++) {
    int j = i+소요 시간[i];
    if (j>날짜수) continue; //ArrayIndexOutOfBounds 문제를 회피하기 위한 문장
    부분 최대치 = max(부분 최대치, dpArray[i]); //부분 최대치 갱신
    dpArray[j] = max(dpArray[j], 부분 최대치+급여[i]); //dpArray[i] 까지의 최대치인 "부분 최대치"야 말로 해당 날짜, 상담을 진행하기 전 기준 보수의 최대치가 된다.
    전체 최대치 = max(전체 최대치, dpArray[j]); //새로 계산된 값과 비교하여 (dpArray의) 전체 최대치를 계속하여 갱신해 나간다.
} //i loop
```

그럼에도 불구하고 이 코드는 잘못된 값을 내놓을 수 있는데 이는 부분 최대치를 갱신하는 문장이 if (j>날짜수) continue; 아래에 위치하기 때문이다. dpArray[i]에 최대치가 저장되어 있어도 j (=i+소요 시간[i])가 범위를 벗어나게 되면 부분 최대치가 갱신되지 않으므로 잘못된 값을 출력하게 된다. 이를 올바르게 잡기 위해서는 부분 최대치를 갱신하는 문장을 if (j>날짜수) continue; 위로 올려서 j (=i+소요 시간[i])가 범위를 벗어나는 여부와 관계 없이 부분 최대치를 갱신할 수 있도록 하는 것이다.

전체 코드는 다음과 같이 된다.

```cpp
#include <iostream>
using namespace std;

//소요 시간[i]: i번째 일을 하는데 소요되는 시간 (day)
int 소요 시간[1500001];
//급여[i]: i번째 일을 할 때 받을 수 있는 보수
int 급여[1500001];
//max(dpCache[i]): i번째 날, i번째 일을 하기 전에 받은 보수의 최대치
int dpArray[1500001];
//현재 dp를 진행한 시점에서의 dpCache의 최대치
int 부분 최대치 = 0;
int 전체 최대치 = 0;

int max(int num0, int num1) {
    return num0>num1 ? num0 : num1;
}

int main() {
    int 날짜수;
    cin >> 날짜수;
    for (int i=0; i<날짜수; i++) {
        cin >> 소요 시간[i];
        cin >> 급여[i];
    } //i loop

    for (int i=0; i<날짜수; i++) {
        부분 최대치 = max(부분 최대치, dpArray[i]); //부분 최대치는 j가 범위를 넘기는 여부와 관계 없이 갱신한다.
        int j = i+소요 시간[i]; //i 번째 일을 잡으면 그 보수는 i+소요 시간[i] 번째 날에 받는 것으로 계산한다.
        if (j>날짜수) continue; //기한을 넘는 일은 고려할 필요가 없다.
        dpArray[j] = max(dpArray[j], 부분 최대치+급여[i]);
        전체 최대치 = max(전체 최대치, dpArray[j]);
    } //i loop
    cout << 전체 최대치;
    return 0;
}
```