과제 #4 구간 결합 - 201921438 조현태

1) 소스코드

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// 함수 선언
int seq_vec(int MAX, int x, int y, int i);
void Data_add(int index, int sum, int x, int y);
int add(int n);
int binarySearch(int a, int x, int d);

int n;
int len; // 2차원 벡터의 크기
vector <int> vec; // 수열
vector <vector <int>> Map;// 2차원 벡터로 표현
vector <vector <int>> Data; // (sum,x,y)로 표현
vector <int> Sum; // 구간합을 저장할 벡터

int main()
{
	cin >> n;
	for (int i = 0; i < n; i++)
	{
		int a;
		cin >> a;
		vec.push_back(a);
	}

	// n x n의 벡터
	Map.assign(n, vector<int>(n, 0));
	// len x 3의 벡터 -> (sum, x, y)
	add(n);
	Data.assign(len, vector<int>(3, 0));

	int sum = 0;
	int index = 0;
	for (int i = 0; i < n; i++)
	{
		sum += vec[i];
```

```
		Map[0][i] = sum;
		// sum, x, y 값을 저장
		Data_add(index, sum, 0, i);
		index++;
		Sum.push_back(sum);
}

for (int i = 1; i < n; i++)
		for (int j = 0; j < n; j++)
		{
			sum = Sum[j] - vec[i-1];
			Map[i][j] = sum;
			if (sum > 0)
			{
				// sum, x, y 값을 저장
				Data_add(index, sum, i, j);
				index++;
			}
			Sum[j] = sum;
		}

// 부분합을 기준으로 내림차순 정리
sort(Data.begin(), Data.end(), greater<>());

int answer = 0;
// 최대 부분합부터 시작
for (int i = 0; i < len; i++)
{
		// 부분합에 대한 x,y좌표 설정
		int MAX = Data[i][0];
		int x = Data[i][2];
		int y = Data[i][1];

		// 내림차순 이진탐색 함수실행
		if (seq_vec(MAX, x, y, i) != -1)
		{
			answer = MAX;
			break;
		}
}

// 함수를 실행했음에도 답이 없을 경우
if (answer == 0)
```

```
                // 기본 수열에서 가장 큰 값 출력
                cout << *max_element(vec.begin(), vec.end()) << endl;
        else
                // 위에서 찾은 answer 출력
                cout << answer << endl;

        return 0;
}

// 1~n까지의 합 -> 함수의 크기를 설정하기 위해
int add(int n)
{
        if (n == 0)
                return len;
        else
                len = n + add(n-1);
}

// 2차 벡터에 sum,x,y값 넣기
void Data_add(int index, int sum, int x, int y)
{
        Data[index][0] = sum;
        Data[index][1] = x;
        Data[index][2] = y;
}

// 내림차순 이진탐색 함수
int binarySearch(int a, int x, int val)
{
        int s = a+1; //시작
        int e = n-1; //끝
        int m; // 중간
        while (s <= e) {
                m = (s + e) / 2;
                if (Map[m][x] == val)
                        return m;
                else if (Map[m][x] < val)
                        e = m - 1;
                else
                        s = m + 1;
        }
        return -1;
}
```

```
// 구간합을 기준으로 해당 x좌표와 y좌표의 같은 값이 있는지 확인하는 함수
int seq_vec(int MAX, int x, int y, int i)
{
	for (int a = y; a < x; a++)
	{
		int val = Map[y][a];
		// 이진탐색 -> y좌표는 a+1 부터 n까지 val과 같은 값이 있다면 MAX출력.
		if (binarySearch(a, x, val) != -1)
		{
			return MAX;
			break;
		}
	}
	return -1;
}
```

2) 문제 설명

이 문제는 n개의 수열 중 크기가 최대가 되는 구간을 구하는 문제입니다.
여기서 크기란, 구간에 포함된 원소들의 합을 의미합니다.
문제의 조건에서 크기가 같고 서로 연속할 경우, 구간을 연결할 수 있다고 했습니다.

저는 여기서 구간에 포함된 원소들의 합에서 구간합을 생각했고,
구간합이 같을 경우, 그 구간끼리 연결하면 되겠다는 생각을 했습니다.
각 구간별로 구간합을 구하니 2차배열이 되었고 그 사이에서 나름의 규칙을 발견했습니다.

수열 = {1,2,3,4}로 간단한 예시를 들자면, 이 수열의 정답은 [1,3] = 6입니다.
즉 **[1,2] = 1+2 = 3** 과 **[3] = 3**을 연결하여 [1,3] = 1+2+3 = 6이 된 것입니다.

```
    1 2 3 4
  | 1 2 3 4  (x)
1 | 1 3 6 10
2 | x 2 5 9
3 | x x 3 7
4 | x x x 4
(y)
```

위의 2차배열을 x,y좌표로 생각했을 때, (2,1) => [1,2]의 구간합이 3이고
(3,3) => [3,3]의 구간합이 3이기에 두 구간을 합쳐 [1,3] = 6이 되었다는 것을 알 수 있습니다.
역으로 생각해보면 (3,1) = 6에서 x=3과 y=1에서 같은 구간합을 가진다는 것입니다.
문제는 구간의 크기가 최대가 되는 값을 구하는 것이고 이미 모든 경우의 구간합을 구했기 때문에
구간합을 기준으로 최댓값부터 진행하여 만족하는 경우, 그 구간합(=크기)를 출력하도록 했습니다.

제 코드에 대해서 간략히 설명하자면,
1. 모든 구간의 구간합을 Map이라는 2차배열로 표현한다.
2. 동시에 구간합과 그 구간합의 2차배열 Map의 x,y좌표를 저장하는 2차배열 Data를 만든다.
(이때 x,y좌표는 인덱스 기준. ex) 수열 : [1,2] + [3] = [1,3] -> 인덱스 : [0,1] + [2] = [0,2]
3. Data를 구간합을 기준으로 내림차순정렬한다. (최댓값부터 실행하기위해)
4. 구간합을 (a,b) = MAX일 때, x=a와 y=b에서 같은 값을 찾는다면 구간합이 MAX이므로
코드을 종료하고 바로 출력한다. (단, y >= k+1 (k는 y=b위의 임의의 값))
5. y=b위의 임의의 값 k에 대하여 x=a위의 같은 값을 찾을 때는 이진탐색을 사용한다.
(stl의 이진탐색은 2차배열에서 적용하기가 힘들어서 이진탐색함수를 따로 구현했습니다.)

Data
{10, 3, 0}
{9, 3, 1}
{7, 3, 2}
{6, 2, 0}
. . .

Map

	1	2	3	4
	0	1	2	3
0	1	3	6	10
1	x	2	5	9
2	x	x	3	7
3	x	x	x	4

Data[0] = {10, 3, 0}이므로 sum = 10일 때, x = 3 , y = 0입니다.
Map[a++][0] = 1이므로 이진탐색을 이용해 Map[3][(a+1)++] == 1을 만족하는지 계산합니다.
Map[0][0] = 1 -> Map[3][1] = 9, Map[3][2] = 7, Map[3][3] = 4
Map[1][0] = 3 -> Map[3][2] = 7, Map[3][3] = 4
Map[2][0] = 6 -> Map[3][3] = 4
만족하는 값이 없으므로 Data[1] = {9, 3, 1}으로 실행합니다.

Map

	1	2	3	4
	0	1	2	3
0	1	3	6	10
1	x	2	5	9
2	x	x	3	7
3	x	x	x	4

이때 Data[3] = {6, 2, 0}일 때, Map[1][0] == Map[2][2]로
[0,1] + [2,2] = [0,2] = 6 -> [1,2] + [3,3] = [1,3] = 6임을 알 수 있습니다.
최댓값부터 진행했으므로 정답은 6입니다.