

알고리즘 HW02

실행환경 : **Eclipse IDE**
사용 언어 : **Java**



2018203065 김보현

1. Main 함수 코드 설명

(1) 입력 : n, n가지의 배열 원소

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    Scanner scanner = new Scanner(System.in);  
    while (true) {  
        //(1) 입력 : n, array원소  
        String N = scanner.nextLine();  
        if (N.equals("EOI")) {  
            break;  
        }  
        int n = Integer.parseInt(N);  
        int[] arr = new int[n];  
        for (int i = 0; i < n; i++) {  
            arr[i] = scanner.nextInt();  
        }  
    }  
}
```

-n 입력

입력을 먼저 String으로 받고, 입력이 EOI일 때 while문을 종료시킨다.

아니라면,

Int형이 들어오는 것이기 때문에 int n 에 String을 int형으로 변환하는 ParseInt함수를 이용해 대입한다.

-배열원소 입력

입력 받은 n 개수 만큼 for문을 통해 배열의 원소를 입력받는다

(2) Partition할 합 구하기

```
//(2) partition할 same sum 구하기  
int sum = 0;  
for (int i = 0; i < arr.length; i++) {  
    sum += arr[i];  
}
```

같은 합의 부분집합을 구하려면, 전체 원소의 합/2 의 값으로 같은 합의 부분집합으로 나뉜다.

먼저 전체 원소의 합을 구하기 위해 for문으로 sum에 arr원소를 arr배열의 길이만큼 더한다.

(3) Partition순서쌍 구하기

```
if(sum%2 == 0) {  
    //(3) Partition 순서쌍 구하기  
    countSubsetSum(arr, sum / 2);  
  
    //(4) Partition 존재시, 부분집합 순서쌍 만들기 출력하기  
    if(PartitionExist == 1) {  
        MakingTwoOfSubsets(arr, sum / 2, PartitionExist);  
        PrintTwoOfSubsets(arr, sum / 2, PartitionExist);  
    }  
}  
else {  
    System.out.println("0");  
}  
  
scanner.nextLine();
```

*Partition이 안되는 경우

(1) sum/2 값이 홀 수 일때

(2) (1)을 만족하지 않아도, sum/2의 합으로 Partition이 안되는 경우

(ex: {3, 4, 9} sum/2 = 8 인 경우, 8의 합으로 부분집합이 만들어지지 않는다)

- if문으로 Partition이 짝수인 경우
 - countSubsetSum. 순서쌍의 개수를 구하고, 출력한다
 - 만약 countSubsetSum의 값이 0이면 PartitionExist의 값이 0이다.
 - while문으로 올라간다.
 - 만약 countSubsetSum의 값이 0이 아니면
 - 두 부분집합을 만들고, 출력한다 (MakingTwoOfSubsets, PrintTwoOfSubsets)
- else문 Partition이 홀수인 경우
 - 0을 출력한다

2. private static void countSubsetSum(int[] array, int sum) 코드 설명

```
private static void countSubsetSum(int[] array, int sum) {
    int n = array.length;
    if (sum <= 0 || array.length == 0 ) {
        return;
    }
}
```

예외처리 구문 : sum이 음수일 때, 배열크기 0일때

```
//Making Array : [배열의 크기] [원하는 합+1] 으로 이차원 배열 생성
count = new long[n][sum + 1];
//Boundary Condition(1) : 모든 원소들은 합 0을 만들 수 있기 때문에 1로 초기화
for (int i = 0; i < n; i++) {
    count[i][0] = 1;
}
```

- Making Array : count라는 long형 이차원배열 생성, 배열의크기, 원하는 합+1으로 만든다
- Boundary Condition : 모든 원소들은 합 0을 만들 수 있기 때문에 1로 초기화한다.

```
//Recursion1 : 첫번째 원소가 sum의 값이라면 1 넣기
for (int j = 0; j <= sum; j++) {
    if (array[0] == j) {
        count[0][j] = 1;
    }
}
//Recursion2 : 첫번째 원소가 sum의 값이 아니라면
for (int i = 1; i < n; i++) {
    for (int j = 1; j <= sum; j++) {

        //(1) : 배열원소보다 sum이 더 크거나 같은 경우, 배열원소의 전의 원소가 sum-array[i]를 만족하고 있는지 확인하기
        // 만족한다면, 값을 더해준다.
        if (j - array[i] >= 0) {
            count[i][j] += count[i - 1][j - array[i]];
        }
        //모든 경우들은 배열원소의 전원소가 sum을 만날 때의 값을 더해준다
        count[i][j] += count[i - 1][j];
    }
}
```

Bottom up 방식으로 배열을 채운다.

-Recursion1

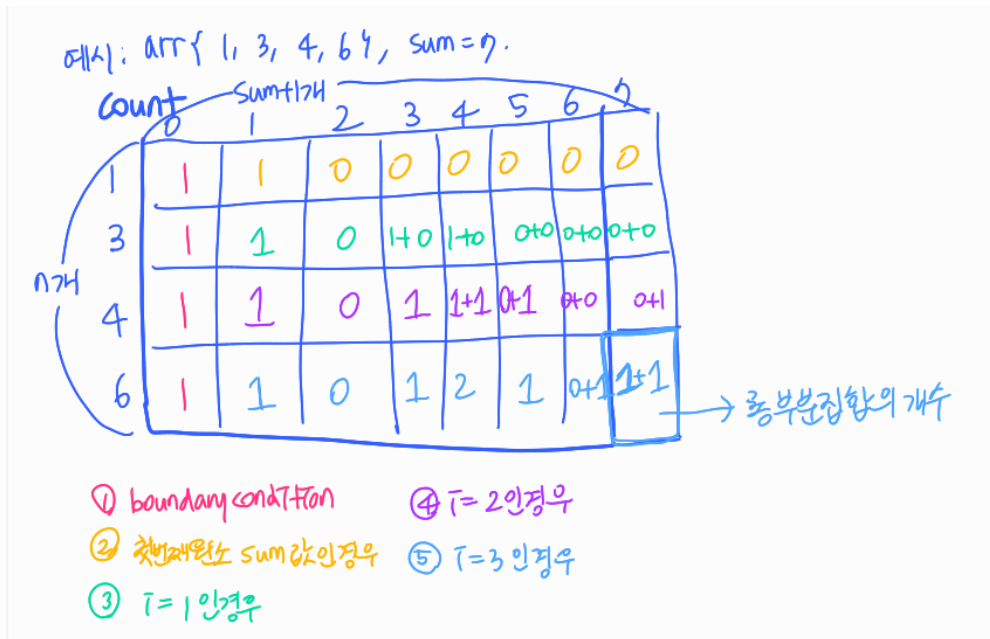
첫번째 원소가 sum의 값이라면 1을 넣어준다.

-Recursion2

두번째원소부터 sum의 값을 늘려가면서 값을 넣어준다.

- 배열의 원소값과 sum이 같거나 sum이 더 큰 경우
 - (1)같은 경우 : 배열의 원소값만큼 뒤로 가서 sum이 0일때의 값=1을 넣어준다
 - (2) 더 큰 경우 : 배열의 원소 값 만큼 뒤로 가서 sum-배열원소값일 때의 값을 넣어준다

- 배열의 원소값이 더 큰 경우
현재의 배열원소값으로 **sum**을 넣어줄 수 없기에 전의 배열원소값인 경우와 같은 개수가 들어간다.



```
//총 부분집합의 개수 출력하기
long coun = count[n - 1][sum] / 2;
PartitionExist=0;
if(coun != 0) {
    PartitionExist = 1;
}
if(coun > Integer.MAX_VALUE || coun < 0 ){
    System.out.println("NUMEROUS");
} else {
    System.out.println(coun);
}
return;
}
```

- n과 sum에 대해 부분집합의 개수는 `count[n-1][sum]`에 들어가게 된다.
그 값을 /2해준 값이 부분집합의 순서쌍 개수이다.
- 만약, 부분집합의 개수가 0이 아니라면,
나중에 부분집합 생성 및 출력을 하기 위해 `PartitionExist = 1`을 해준다.
- 부분집합의 개수가 `ULONG_MAX`보다 크다면
`NUMEROUS`를 출력해준다.
 - `ULONG_MAX` 는 unsigned long int의 최대값 이다. 4294967295
- 아니면
개수의 값을 출력해준다.

3.private static void MakingTwoOfSubsets(int [] array, int sum, int exist)

```
private static void MakingTwoOfSubsets(int [] array, int sum, int exist) {
    if(exist == 1 ) {
        //(1) : 두개의 부분집합 ArrayList 생성
        Subset = new ArrayList<>();
        OtherSubset = new ArrayList<>();

        //(2) : 변수 값 초기화
        int n = array.length;
        int arraynum = n - 1;
        int sumnum = sum;

        //(3) : 첫번째 배열 값 대입
        while (arraynum != 0) {
            if (count[arraynum][sumnum] != count[arraynum - 1][sumnum]) {
                Subset.add(array[arraynum]);
                sumnum -= array[arraynum];
            }
            arraynum--;
        }
        if (sumnum > 0) {
            Subset.add(array[arraynum]);
        }
        //(4) : 두번째 배열 값 대입
        for (int i =0;i<n;i++) {
            if (!Subset.contains(array[i])) {
                OtherSubset.add(array[i]);
            }
        }
    }
}
```

<배열 만든 구조>

-Recursion1

첫번째 원소가 sum의 값이라면 1을 넣어준다.

-Recursion2

두번째원소부터 sum의 값을 늘려가면서 값을 넣어준다.

- 배열의 원소값과 sum이 같거나 sum이 더 큰 경우
(1)같은 경우 : 배열의 원소값만큼 뒤로 가서 sum이 0일때의 값=1을 넣어준다
(2) 더 큰 경우 : 배열의 원소 값 만큼 뒤로 가서 sum-배열원소값일 때의 값을 넣어준다
- 배열의 원소값이 더 큰 경우
현재의 배열원소값으로 sum을 넣어줄 수 없기에 전의 배열원소값인 경우와 같은 개수가 들어간다.

만든 이차원배열 순서의 반대방향으로 부분집합의 원소를 구한다.

(1)첫번째 부분집합

-while문

if문(배열의 원소값이 sum보다 더 큰 경우가 아닐때)

즉, 배열의 원소값과 sum이 같거나 sum이 더 큰 경우

자신의 값이므로, 부분집합에 넣어준다.

sum의 값을 배열의원소값만큼 빼고, arraynum도 하나 빼준다.

-if문

while문을 다 돌아도 sumnum이 0이 아닌 경우가 있을 수 있다 (위의 예시 그림 참고)

이유는 while문에서 배열num이 1까지인 경우만 확인하기 때문이다.

while문에서 num=0확인시, 뒤의 배열 확인불가(0값에 음수가 들어가기 때문)

(2)두번째 부분집합

첫번째에 들어가지 않은 원소들을 넣어준다.

4.private static void PrintTwoOfSubsets(int[] array,int sum,int exist)

```
private static void PrintTwoOfSubsets(int[] array,int sum,int exist) {
    if(exist == 1 ) {
        //(1) : 첫 번째 Subset 출력
        System.out.print("{");
        for (int i = 0; i < Subset.size(); i++) {
            System.out.print(Subset.get(i));
            if (i != Subset.size() - 1) {
                System.out.print(",");
            }
        }
        System.out.print("} , ");

        //(2) : 두 번째 Subset 출력
        System.out.print("{");
        for (int i = 0; i < OtherSubset.size(); i++) {
            System.out.print(OtherSubset.get(i));
            if (i != OtherSubset.size() - 1) {
                System.out.print(",");
            }
        }
        System.out.println("}");
    }
}
```

for문과 get()을 이용하여 만들어진 두 부분집합을 출력한다.
size() 만큼 출력하고, size()-1만큼 “,”을 넣어 출력형식을 맞추었다.

5. 고찰

이번 알고리즘 과제를 통해, 다이나믹 프로그래밍에 대한 시간복잡도의 효율성을 많이 깨달았다.
처음에, recursion식으로 코드를 짰 상황에서 n이 40 이상만 되어도 출력이 되지 않는 부분을 확인하였다.
(아래 코드 사진 : recursion코드로 부분집합 개수 구한 사진)

```
int countSubsetSum(int array [], int n, int index, int sum, int count) {
    if (index == n) {
        if (sum == 0) {
            oneOfAnswer = selectedSubset;
            count++;
        }
        return count;
    }
    selectedSubset[index] = array[index];
    count = countSubsetSum(array, n, index + 1, sum - array[index], count);
    selectedSubset[index] = 0;
    count = countSubsetSum(array, n, index + 1, sum, count);
    return count;
}
```

그 뒤에 4-5절 knap과 3장의 dp 개념을 이용해보면서 개수를 구하니, 빠른 속도로 구해짐을 알았다.
n의 제한값이 광범위하다보니, c++으로 구현시 동적할당에 대한 어려움이 많았다.
머릿 속의 알고리즘을 쓰기 전에 동적할당에 대한 어려움때문에 코드테스트도 잘 하지 못하게 되어

개발환경을 자바로 바꾸었다. 자바의 쉬운 객체할당으로 인해 알고리즘구현 속도도 빨라져 앞으로 자바를 애용할 것 같다.

dp[][] 작성시, dp[n+1][sum+1] 할당으로 착각하여 계속 값 오류가 났었다.

이를 해결하고자, 수기로 dp[][]를 계속 Test해보며, dp[n][sum+1]으로 변경하였다.

dp[][]를 만들고, 부분집합의 개수를 더할 때, 첫번째 원소가 sum인 경우를 넣지 않아서, bottom-up 이 제대로 이뤄지지 않았었고, 최대 부분집합의 개수가 2인 경우만 되는 경우들을 접하였다.

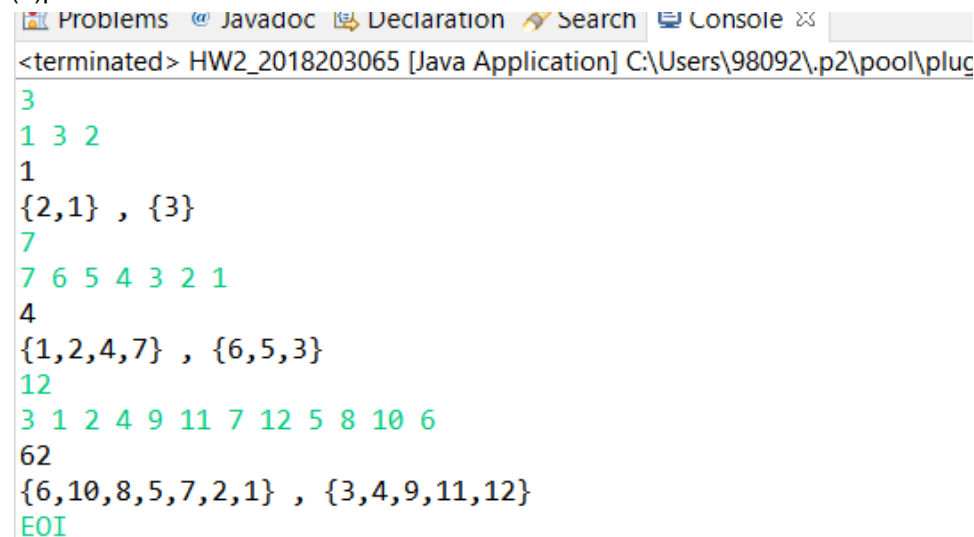
count를 해주어야하는데, 그 때의 상황에서만 개수를 더한 것이 착각이었다.

sum의 값보다 작고 같거나한 경우에는 count를 두번 해주어야 했었고,

모든 경우에 전 상황에서 sum구한 count값을 더해줌을 깨달았다.

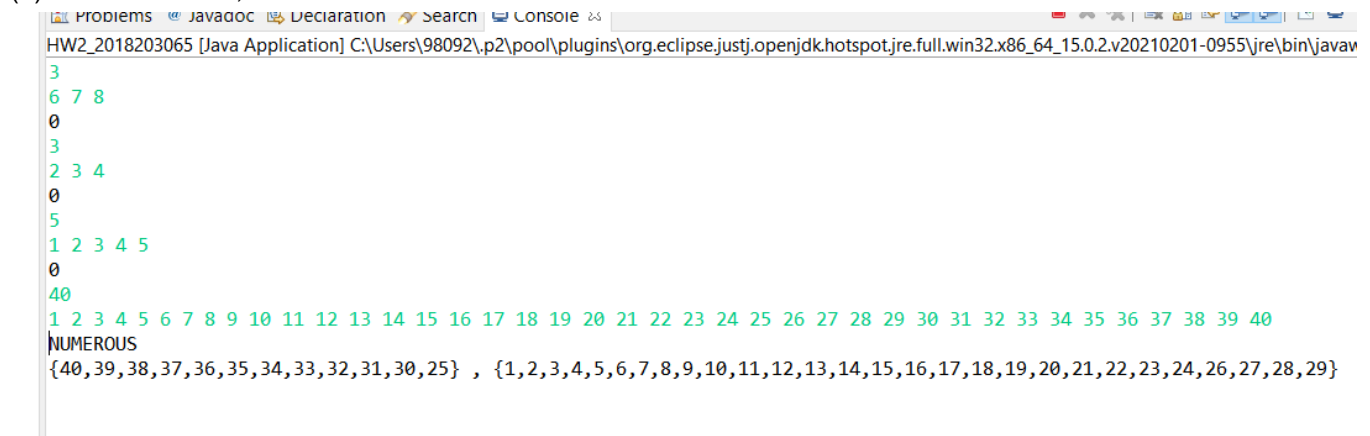
6.실행 예시

(1)pdf 실행 예시



```
<terminated> HW2_2018203065 [Java Application] C:\Users\98092\p2\pool\plug
3
1 3 2
1
{2,1} , {3}
7
7 6 5 4 3 2 1
4
{1,2,4,7} , {6,5,3}
12
3 1 2 4 9 11 7 12 5 8 10 6
62
{6,10,8,5,7,2,1} , {3,4,9,11,12}
EOI
```

(2) 0이 나오는 경우, NUMEROUS가 나오는 경우 예시



```
HW2_2018203065 [Java Application] C:\Users\98092\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javav
3
6 7 8
0
3
2 3 4
0
5
1 2 3 4 5
0
40
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
NUMEROUS
{40,39,38,37,36,35,34,33,32,31,30,25} , {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27,28,29}
```

(1) 0이 나오는 경우 : sum합이 짝수지만, 나누어지지 않는 경우 두가지, sum이 홀수인 경우 한가지

(2) NUMEROUS의 경우

(3) 최대값 n=999 출력

999의 원소를 입력하기 위해 입력부분을 i+1으로 수정하였다.

```
121         break;
122     }
123     int n = Integer.parseInt(N);
124     int[] arr = new int[n];
125     for (int i = 0; i < n; i++) {
126         arr[i] = i+1;
127     }
128
```

Problems Javadoc Declaration Search Console

HW2_2018203065 [Java Application] C:\Users\98092\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (May 7, 2021, 7:03:00 PM)

999

NUMEROUS

{999,998,997,996,995,994,993,992,991,990,989,988,987,986,985,984,983,982,981,980,979,978,977,976,975,974,973,972,971,970,969,968,967,966,965,964,963,962,9