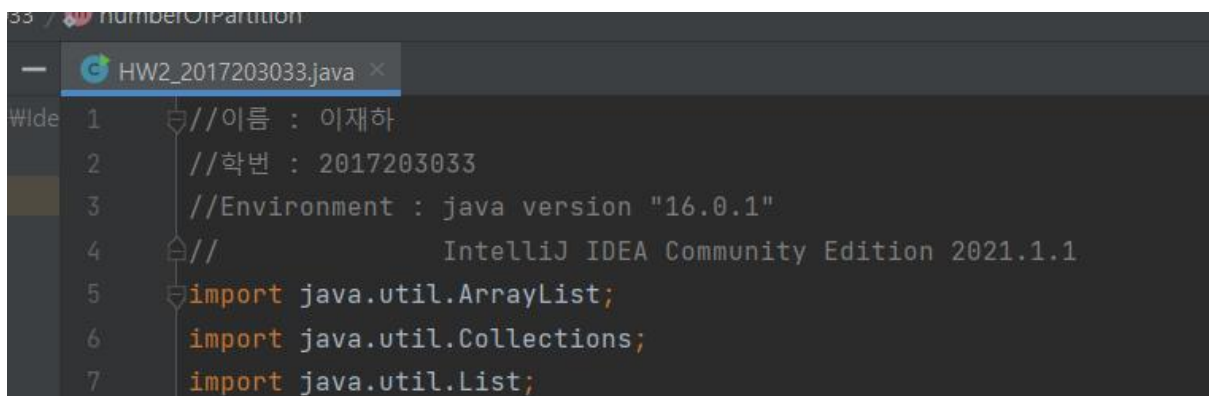




알고리즘 HW2 Report

2017203033 이재하

<구현환경>

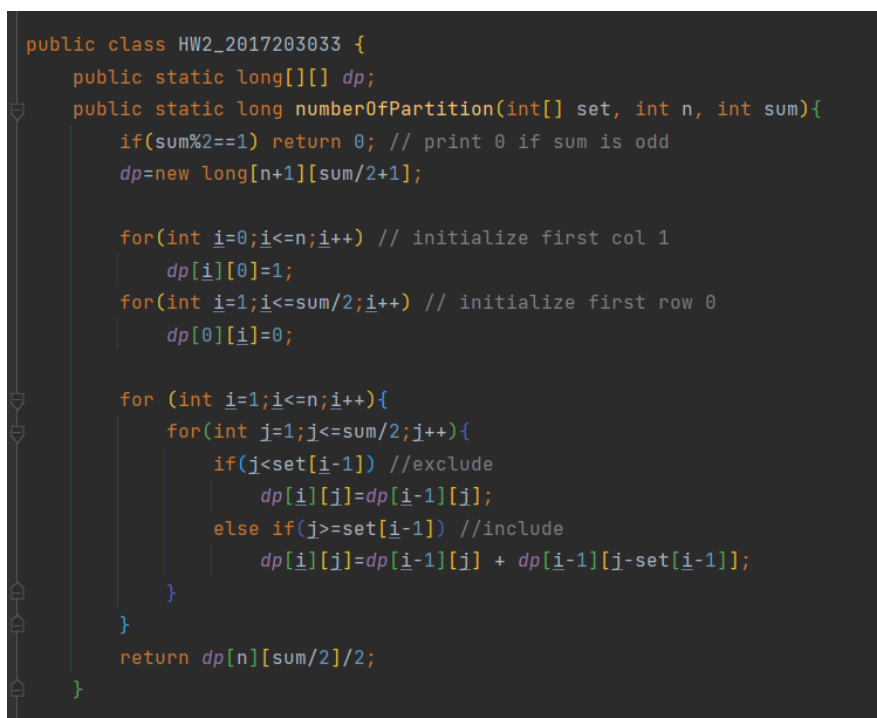


```
33 / numberOfPartition
HW2_2017203033.java
1 //이름 : 이재하
2 //학번 : 2017203033
3 //Environment : java version "16.0.1"
4 // IntelliJ IDEA Community Edition 2021.1.1
5 import java.util.ArrayList;
6 import java.util.Collections;
7 import java.util.List;
```

Programming language : Java (version "16.0.1")

Environment : IntelliJ IDEA Community Edition 2021.1.1

<소스코드 구성>



```
public class HW2_2017203033 {
    public static long[][] dp;
    public static long numberOfPartition(int[] set, int n, int sum){
        if(sum%2==1) return 0; // print 0 if sum is odd
        dp=new long[n+1][sum/2+1];

        for(int i=0;i<=n;i++) // initialize first col 1
            dp[i][0]=1;
        for(int i=1;i<=sum/2;i++) // initialize first row 0
            dp[0][i]=0;

        for (int i=1;i<=n;i++){
            for(int j=1;j<=sum/2;j++){
                if(j<set[i-1]) //exclude
                    dp[i][j]=dp[i-1][j];
                else if(j>=set[i-1]) //include
                    dp[i][j]=dp[i-1][j] + dp[i-1][j-set[i-1]];
            }
        }
        return dp[n][sum/2]/2;
    }
}
```

위의 소스코드는 부분집합으로 나눌 수 있는 개수를 파악하는 함수이다. 함수를 어떻게 구현하였는지 간단히 설명하자면 다음과 같다. 우선 함수의 처음 부분에서 sum이 홀수인 경우, 나눌 수 없으므로 0을 리턴 해준다. 그리고 전역으로 선언한 dp배열에 첫 column에는 1을 넣어 초기화를 해주고, 첫 row에는 0을 넣어 초기화를 해준다. 이렇게 하는 이유는 첫 column에 들어갈 값이 나타내는 것은 $\text{sum}/2$ 의 값이 0인 경우 공집합이 들어가는 경우는 무조건 포함하므로 1이 들어가고, 첫 row에 들어갈 값이 나타내는 것은 원소가 0개일 때 만들 수 있는 값을 나타내는 것인데, 이 경우는 sum이 0일 때 말고는 없으므로 0을 넣어주는 것이다.

그리고 나서 (1,1)부터 (n,sum/2)까지는 일정한 규칙에 따라 값을 넣어주었다. 만약, j의 값이 요소의 값보다 작은 경우는 이 원소를 넣는 것을 배제해야 하므로 그 전의 값을 넣어주었다. 그리고 만약 j의 값이 요소의 값보다 크거나 같다면 이 원소를 넣는 것도 허용되므로 이 원소를 넣는 경우 까지 같이 계산해주었다. 그리고 나서 dp테이블의 가장 오른쪽 아래 값, (n,sum/2)의 값이 우리가 원하는 값의 두배인데, 그 이유는 2개의 부분집합으로 나뉘지는데 2개 중 어느 부분집합이 먼저 나오냐에 따라 중복된다. 그래서 우리가 원하는 값은 겹치지 않게 해야 하므로, (n,sum/2)의 값을 2로 나눈 값을 리턴 해주었다.

```
public static void printSubset(int[] set, int n, int sum) {
    List<Integer> subset = new ArrayList<Integer>(); //save a subset
    List<Integer> subset2 = new ArrayList<Integer>(); //save another subset
    int i = n;
    int j = sum / 2;
    while (i > 0 && j >= 0) {
        if (dp[i - 1][j] != 0) {
            i--;
            subset2.add(set[i]);
        } else if (dp[i - 1][j - set[i - 1]] != 0) {
            i--;
            j -= set[i];
            subset.add(set[i]);
        }
    }
    Collections.sort(subset);
    Collections.sort(subset2);

    System.out.print("{");
    for (int a = 0; a < subset.size(); a++) {
        if(a==subset.size()-1) System.out.print(subset.get(a) + "}");
        else System.out.print(subset.get(a) + ",");
    }

    System.out.print(", {");
    for (int b = 0; b < subset2.size(); b++) {
        if(b==subset2.size()-1) System.out.println(subset2.get(b) + "}");
        else System.out.print(subset2.get(b) + ",");
    }
}
```

다음으로 구현한 함수는 printSubset 함수로 부분집합을 출력해주는 함수를 만들었다. 부분집합을 출력할 때, 두개의 부분집합으로 출력하므로, ArrayList를 활용하여 만든 subset, subset2에 원소를 저장하였다. While문 안에서 subset과 subset2 중 어느 것에 들어갈 것인지를 정했다. 만약, (i-1,j) 값이 0이 아니라면, 현재의 값에 기여하지 않은 것이므로, subset2에 넣어주고, i의 값을 1 빼주었다. 그리고 만약 (i-1,j-set[i-1])의 값이 0이 아니라면, 현재의 값에 기여한 것이므로, 이 때의 index i에 있는 값을 subset에 넣어준다. 그리고, i와 j를 각각 업데이트해주었다.

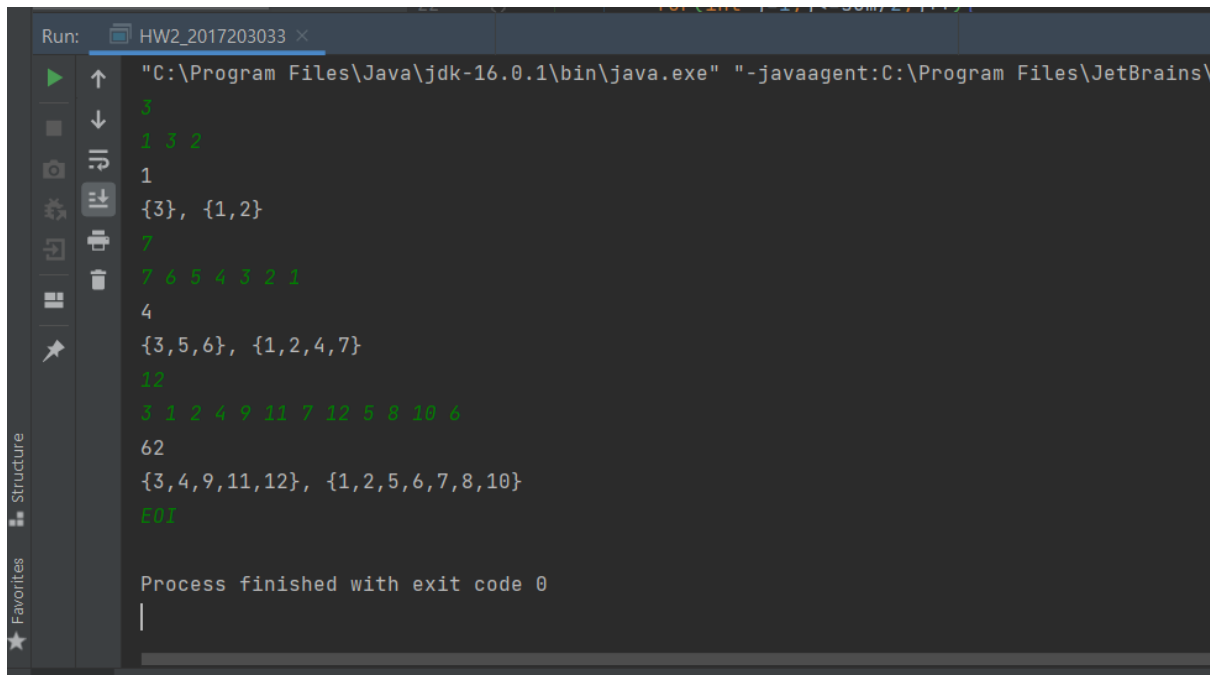
이렇게 되면, subset과 subset2에는 각각 부분집합들이 담기게 되는데, 이 요소들을 정렬해주어야 한다고 했으므로, 각각 sort해주었다. 그리고 나서 각각의 부분집합들을 출력 형태에 맞게 반복문

을 돌려서 출력해주었다.

```
}  
public static void main (String[]args){  
    Scanner sc = new Scanner(System.in);  
    String str = sc.next();  
    while (!str.equals("EOI")) {  
        int n = Integer.parseInt(str); //change input type string to int  
        int[] set = new int[n]; //element set  
        for (int i = 0; i < n; i++)  
            set[i] = sc.nextInt();  
        int sum = 0; //sum of element  
        for (int i = 0; i < n; i++)  
            sum += set[i];  
  
        //print result  
        long result = numberOfPartition(set, n, sum);  
        if (result > (long) (Math.pow(2, 32) - 1) || result < 0) {  
            System.out.println("NUMEROUS");  
            printSubset(set, n, sum);  
        }  
        else if(result==0)  
            System.out.println(result);  
        else {  
            System.out.println(result);  
            printSubset(set, n, sum);  
        }  
        str = sc.next();  
    }  
}
```

다음은 메인함수이다. 위에서 만든 함수들을 적절히 활용하였다. 먼저 입력이 EOI일수도 있으므로, string으로 받아서 입력이 EOI가 아닐 경우만 while문을 돌게 만들었다. 그래서 while문 안에서는 입력을 받아서 정수형태로 변환해준 뒤, sum을 구하였다. 그리고 이렇게 입력 받은 집합, n과 sum을 활용하여 함수에 넣어주었는데, numberOfPartition의 값이 만약 ULONG_MAX($2^{32}-1$)보다 크면, NUMEROUS를 출력하고, 부분집합을 출력해주었고, 0이라면 방법이 없는 것이므로 0을 출력하였고, 그렇지 않은 경우는 numberOfPartition을 출력하고, 부분집합을 출력해주었다.

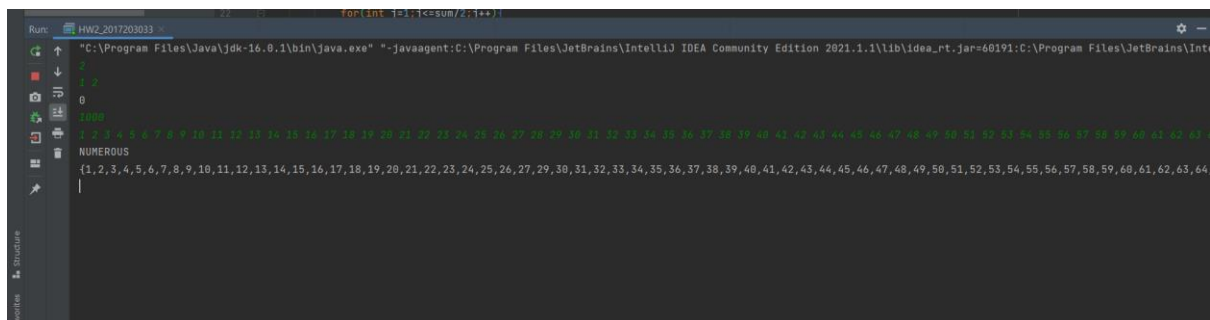
다음은 이러한 코드를 토대로 입력 받아 나온 결과들의 예시들이다.



```
Run: HW2_2017203033 x
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\
3
1 3 2
1
{3}, {1,2}
7
7 6 5 4 3 2 1
4
{3,5,6}, {1,2,4,7}
12
3 1 2 4 9 11 7 12 5 8 10 6
62
{3,4,9,11,12}, {1,2,5,6,7,8,10}
EOI

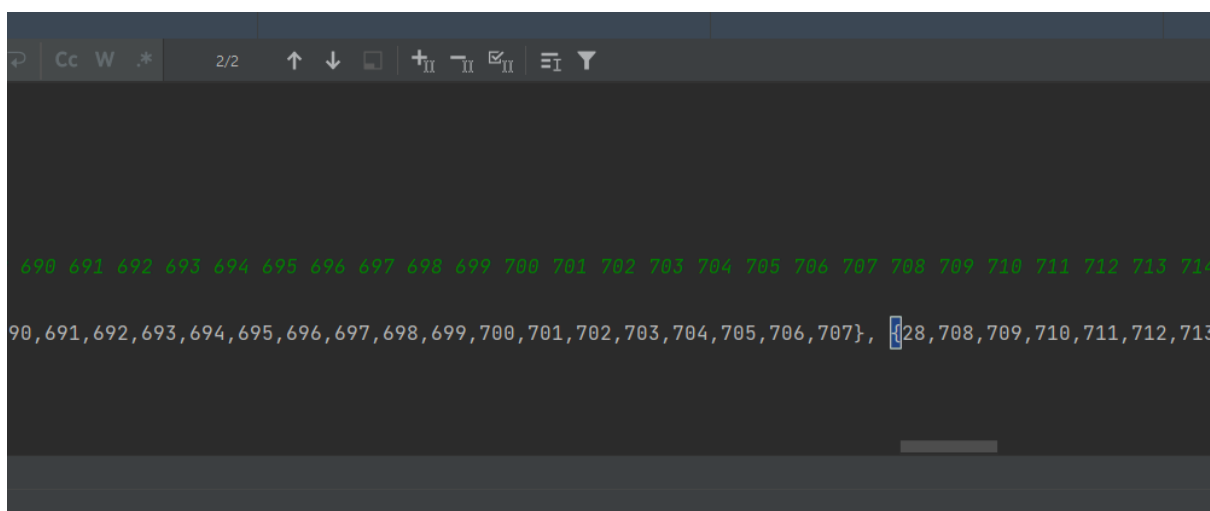
Process finished with exit code 0
```

1,2,3번째 예시들은 각각 과제pdf에 나온 예시들이다. 결과들을 보면 잘 나온 것을 알 수 있다. 또한 입력을 계속 받다가 EOI를 입력하면 종료되는 것을 알 수 있다.



```
Run: HW2_2017203033 x
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1.1\lib\idea_rt.jar=60191:C:\Program Files\JetBrains\Int
2
1 2
0
1000
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 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
NUMEROUS
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64}

Process finished with exit code 0
```



```
Run: HW2_2017203033 x
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1.1\lib\idea_rt.jar=60191:C:\Program Files\JetBrains\Int
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714
90,691,692,693,694,695,696,697,698,699,700,701,702,703,704,705,706,707}, {28,708,709,710,711,712,713}

Process finished with exit code 0
```

위의 두 캡처 사진은 두 예시를 보여준다. 첫 예시는 0이 출력되는 경우를 보여주고, 두번째 예시는 1000과 1~1000을 넣었을 때 2개의 부분집합으로 잘 나뉘지는 것을 알 수 있다.

이번 과제를 통해 dynamic programming을 활용한 것이 무엇인지 알 수 있게 되었고, 조금 더 익숙해진 것 같다. 또한 과제를 하기 전에는 강의에서 들은 개념은 잘 알아도, 문제에 응용하는 것은 익숙하지 않았는데, 이번 과제를 토대로 dynamic programming 문제를 어떤 과정을 통해 풀어나가야 하는지 잘 알 수 있게 되었다고 생각한다.