REPORT

과목명 : 알고리즘
학 과 : 소프트웨어학부
학 번 : 2017203035
이 름 : 김수헌
교수명 : 김용혁 교수님

목차

Programing Language : C++

Compiler : MSVC, Microsoft Visual C++

1. 첫 구현에서의 실패

코드 설명

난 처음 과제를 접했을 땐 단순하게, '**부분집합을 1개를 구해서 그때의 값이 총합의 절반이면 Count 해주면 되겠다**'라는 결론을 냈다.

교수님이 DP를 써서 구현하라고 말씀하셨는데, 나는 단순히 점화식만 사용하면 DP가 되는 줄 알고 구현했다. 그렇게 해서 Simple Algoritm으로 짠 첫 코드가 완성되었다.

```
void dfs(int i, int sum, int arr[], int n, int total)
{
    if (sum > (total / 2)) //0부터 더 해가던 배열 값이 total/2보다 더 커진 경우는 더 이상 같은 값이 없는 경우
        return;

    if (i == n)
    {
        if (sum == (total - sum))
        {
            if (!flag) //한번만 실행되도록 함, 아무 부분 수열 조합 딱 한번만 출력하면 되기 때문
            {
                for (int j = 0; j < i; j++)
                {
                    if (pArr[j] != 0)
                        leftArr.push_back(pArr[j]); //부분 수열이 leftArr에 담김.
                }
                flag = true; // true를 사용하므로써 다시는 실행되지 않게
            }
            ans += 1; // 경우의 수 카운트
        }
    }
    else
    {
        pArr[i] = arr[i];
        dfs(i + 1, sum + arr[i], arr, n, total);
        pArr[i] = 0;
        dfs(i + 1, sum, arr, n, total);
    }
}
```

원리는 간단하였다. 함수의 인자로 인덱스 i, 총합 sum, 배열 arr, 개수 n, 토탈값 total을 인자로 넘긴다. 그러면 재귀함수는 계속해서, sum에 arr[i]를 더한만큼을 sum값으로 넘겨 재귀를 실행한다. 이렇게 부분집합의 모든 경우의 수를 탐색한다. 그리고 만약 I의값과 n값이 같아졌을 때, sum 값이 total의 절반이라면 그대의 배열 값을 vector에다가 저장하고 ans에다가 값 하나를 카운트 한다.

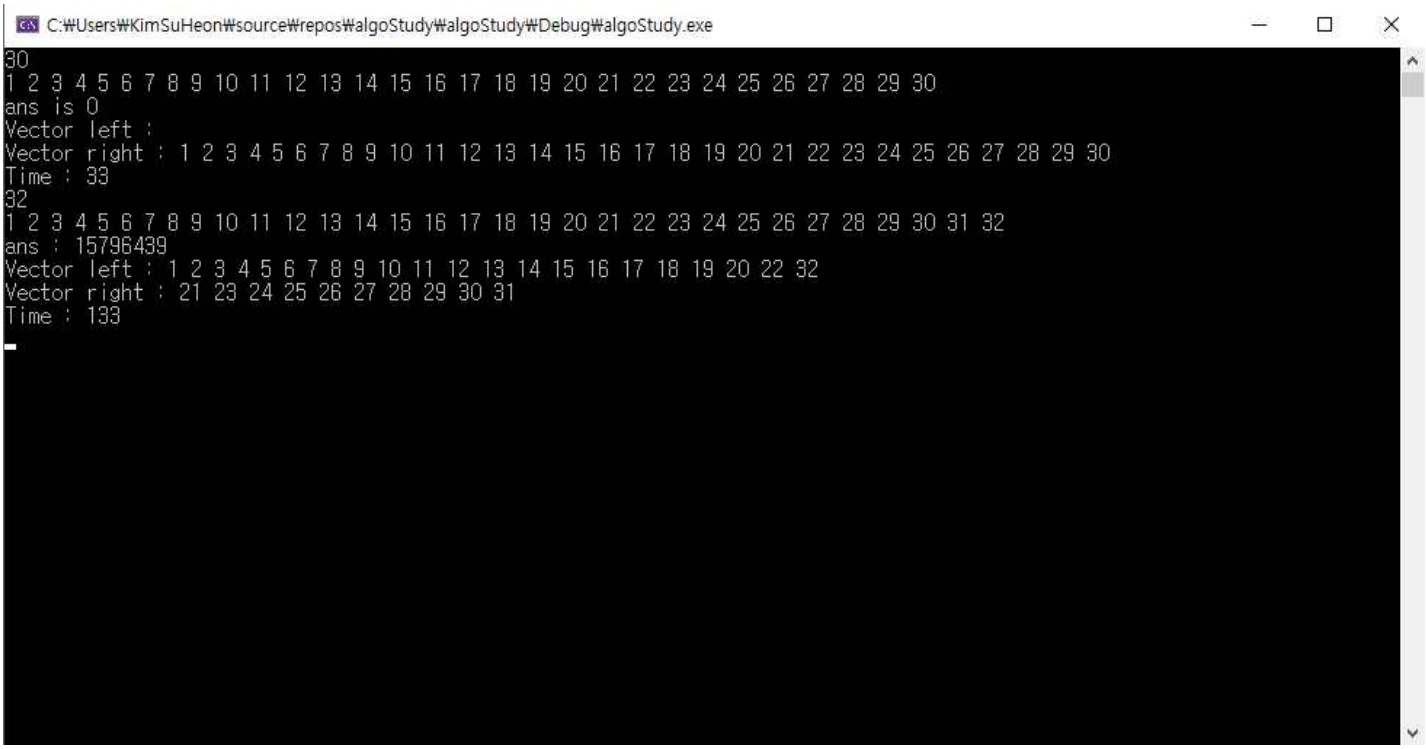
그리고 부분집합의 출력은 다음과 같다.
모든 요소를 담고 있는 allArr와 부분수열 집합 둘 중 하나를 담고 있는 leftArr를 차집합 시켜서 rightArr를 구하고 leftArr와 rightArr를 출력하는 형식이다.

```cpp
//배열 출력해주는 함수
void printSubset()
{
    //allArr, leftArr 정렬
    sort(allArr.begin(), allArr.end());
    sort(leftArr.begin(), leftArr.end());

    //차집합을 rightArray에 추가
    vector<int> rightArr(allArr.size() + leftArr.size());
    auto iter = set_difference(allArr.begin(), allArr.end(), leftArr.begin(), leftArr.end(), rightArr.begin());
    rightArr.erase(iter, rightArr.end());

    //rightArr 정렬
    sort(rightArr.begin(), rightArr.end());

    //출력
    cout << "{ ";
    for (int i = 0; i < leftArr.size() - 1; i++)
        cout << leftArr[i] << ", ";
    cout << leftArr[leftArr.size() - 1] << " }, ";

    cout << "{ ";
    for (int i = 0; i < rightArr.size() - 1; i++)
        cout << rightArr[i] << ", ";
    cout << rightArr[rightArr.size() - 1] << " }" << endl;
}
```

이 같은 동작으로 배열 요소를 출력한다.

테스트 케이스 결과

결과적으로 작동은 잘한다. 하지만 시간이 너무 많이 걸린다. 재귀 함수로 하나보니 n에 십의 자리수만 들어가도 시간이 상당히 많이 소요된다.



30은 33초가 걸리며, 32는 133초가 걸렸다. 나는 이 재귀적으로 하는 방법은 잘못됐음을 인지했다. 인풋 케이스가 1000이기 때문에 이렇게 시간이 많이 걸리면 인풋 케이스를 구할 수가 없었다.

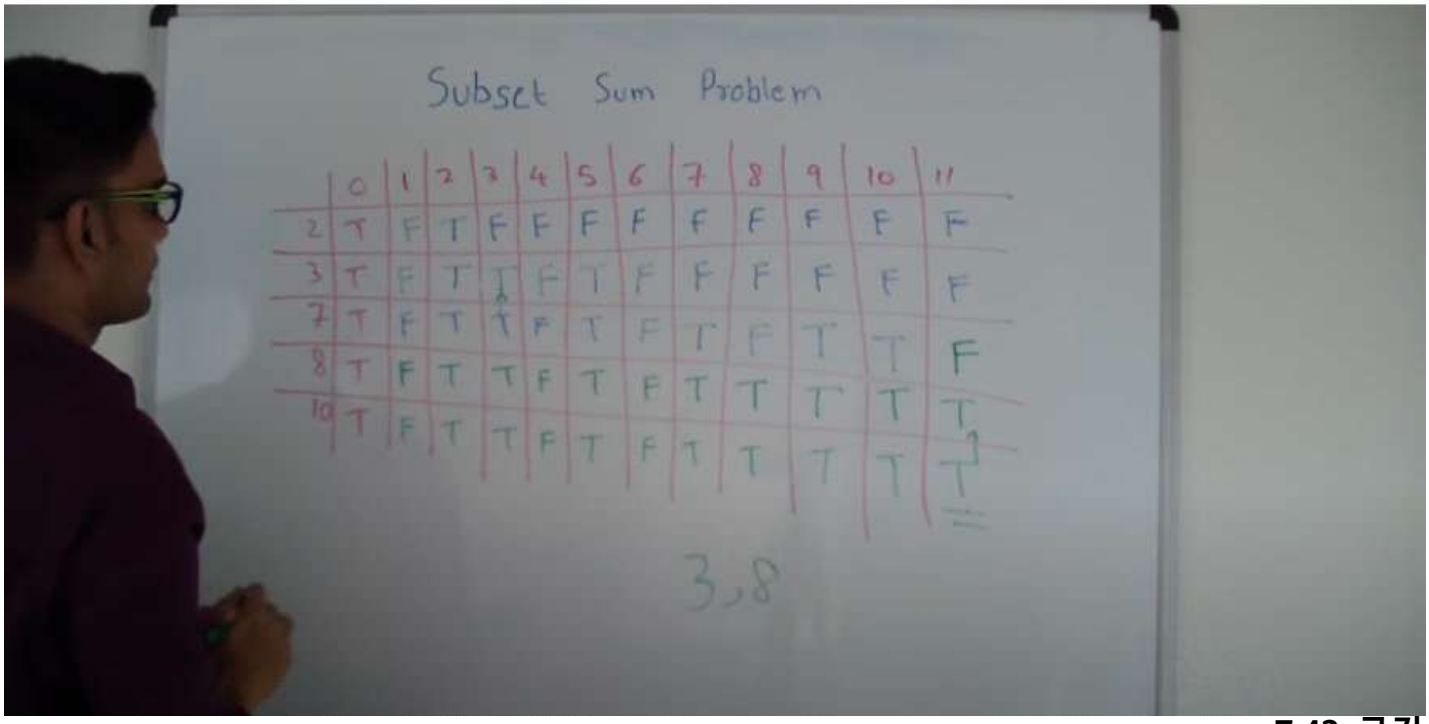
2. 두 번째 구현을 하기 위한 노력

도저히 내 머릿속으로는 DP에서 2차원 배열을 이용하여 값을 채우는 방법이 떠오르지 않았다. 그래서 관련 지식을 쌓기 위해 구글링을 시작했고, 부분 수열에 관한 구글 검색 결과는 모조리 찾아 공부했다.

백준 알고리즘, 그리고 아마존 인터뷰 DFS 등 다양한 것들을 찾아봤다. 그리고 유튜브에서 부분 수열에 관련된 영상을 찾아 이해하려고 노력했다. 그러다가 찾은 것이 이 영상이었다.

Subset Sum Problem Dynamic Programming - https://www.youtube.com/watch?v=s6FhG—P7z0

나는 영상을 보면서 DP의 원리를 이해했다. 부분 수열이 되는 것은 DP 2차원 배열에서 값이 TRUE 값으로 채워지는 원리였다. 그리고 영상 마지막에 보면 부분 수열의 배열 값도 구할 수 있다고 하는데, 이 부분을 보고 백트래킹의 원리에 대해 이해하였다.



7:42 구간

2차원 배열 마지막에서 출발하여 배열 위 값이 F라면 Columns을 요소값 만큼 뒤로 이동한다. 이 같은 로직이 가능한 이유는, 행은 요소의 개수이고, 열은 요소의 총합이기 때문이다. 위 로직으로 계속 백트래킹을 하면 참조했던 모든 값을 거치게 된다. 즉, **파티션 된 배열의 한 부분**을 구할 수 있다.

나는 이것을 보고 처음에 만들었던 subSetPrint 함수에 대입하면 똑같이 부분 수열의 파티션 값을 구할 수 있다고 생각했다.

또한 교수님이 참고하라고 알려주신 '**문제로 풀어보는 알고리즘**' 서적의 부분 수열 부분도 찾아보았다.

위 알고리즘은 원하는 합에 대하여 부분집합으로 그 합을 만드는 것이 가능한지 불가능한지 알려주는 예제였는데 이러한 충분한 공부 끝에 나는 HW2 과제 DP구현을 시도할 수 있었다.

3. 코드 설명 및 테스트 케이스

코드 설명

먼저 내 코드의 구조는 다음과 같이 구성된다.

```
//flag가 true라면 NUMEROUS
//limt 값 초과시 NUMEROUS\n
const long long limit = 4294967295; //2^32 - 1
bool flag = false;

//배열 요소 추적해주는 함수
void backTrackingDp(long long** T, int arr[], int n, int total);

//같은 셋이 없으면 0 출력
vector<int> leftArr;
vector<int> rightArr;
vector<int> allArr;
```

먼저 전역변수 부분이다. limit와 flag는 NUMEROUS 예외처리를 위한 전역변수이다. 처음엔 define으로 정의했었는데 long long 값이 비교가 안될 거 같은 불안한 느낌에, 그냥 동일한 long long 상수를 만들어줬다.

그리고 다음으로 살펴볼 부분은 main문이다.

```
int main()
{
    string num;
    int total = 0;
    long long ans = 0;

    //for (int i = 1; i <= 1000; i++)
    //  cout << i << " ";
    //cout << endl;

    while (1)
    {
        cin >> num;

        if (num == "EOI")
            break;

        int n = stoi(num);
        int* arr = new int[n + 1];

        for (int i = 0; i < n; i++)
        {
            cin >> arr[i];
            total += arr[i];
            allArr.push_back(arr[i]);
        }
```

```
    if (total % 2 != 0) //배열의 총 합이 2로 나누어 떨어지지 않는다면, 파티션 될 수 없음 0출력
    {
        cout << "0" << endl;
        allArr.clear();
        total = 0;
        delete arr;
        continue;
    }

    //Sorting & subsetCount() 실행
    sort(arr, arr + n);
    ans = subsetCount(arr, n, total / 2);

    //Result
    if (ans == 0) //2로 나눠지는 집합 중에 카운트 할 수 없는 집합, ex) {2, 4, 8, 12}
    {
        cout << "0" << endl;
    }
    else if(flag == true) //2^32 - 1 초과하는 값이라면
    {
        cout << "NUMEROUS" << endl;
        printSubset();
    }
    else  // 2^32 - 1 값 범위 안에 있다면
    else  // 2^32 - 1 값 범위 안에 있다면
    {
        cout << ans << endl;
        printSubset();
    }

    //전역/지역 변수 초기화
    leftArr.clear();
    rightArr.clear();
    allArr.clear();
    total = 0;
    flag = false;
    delete arr;
}

return 0;
```

main 문의 동작 원리는 다음과 같다. 시작부터 while 문 내에서 string 형식으로 입력을 받는데 만약 EOI면 while문을 탈출해 프로그램을 종료하고, 숫자 값이라면 정수로 형변환을 하여 부분 수열을 구할 준비를 한다.

입력 받는 수를 모두 arr와 allArr에 넣어주고 모든 수를 더한 값을 total값에 저장한다. 그리고 만약 total 값이 2의 배수가 아니면 0을 띄우고 다시 입력을 받는다. 왜냐하면 total 값이 2로 나누어 떨어지지 않는다는 것은 부분 수열로 파티션을 할 수 없다는 것을 의미한다.

2로 나누어 떨어지는 수라면 배열을 sort함수로 정렬하고 subsetCount 함수를 실행한다. 그리고 반환 값을 long long 형태인 ans 값에 저장한다. 이때 반환값은 부분집합의 개수이다.

만약 ans가 0이라면?
만족하는 부분집합이 없다는 뜻이다. 예를 들어 {2, 4, 6, 8}이 있다. 배열의 총합이 2로 나누어 떨어지는 수지만 만족하는 부분집합을 만들 수 없기에 이 같은 경우가 여기 if문에 걸려 0을 반환한다.

만약 flag가 true 값이라면?
이따가 설명하겠지만, flag가 true라는 것은 2^32 – 1 즉, 약 43억을 초과한다는 뜻이며 NUMEROUS에 대한 예외처리를 실시한다. 그리고 printSubset 함수로 부분 수열을 출력하고 다시 입력을 받는다.

나머지 모든 케이스
나머지 케이스는 약 43억 이하를 만족하는 경우이기 때문에 그때의 count 개수를 출력하고 똑같이 printSubset 함수로 부분 수열을 출력한다.

---

subsetCount 함수

```
long long subsetCount(int arr[], int length, int total)
{
    try
    {
        //메모리 해제하기 위해서 ans에 저장
        long long ans = 0;

        //2차원 배열 동적 할당 heap
        long long** T = new long long* [length + 1];
        for (int i = 0; i < length + 1; i++)
            T[i] = new long long[total + 1];

        ////배열 전체 초기화 - 굳이 할 필요 없음. 할당하면 전부 0으로 초기화 되기 때문
        //for (int i = 1; i <= length; i++) {
        //  for (int j = 1; j <= total; j++) {
        //      T[i][j] = 0;
        //  }
        //}

        //첫번째 열에 1 값 대입
        for (int i = 0; i <= length; i++)
            T[i][0] = 1;

        //첫번째 행에 0 값 대입
        for (int i = 1; i <= total; i++)
            T[0][i] = 0;
```

subsetCount 함수는 이번 알고리즘의 과제의 핵심이 되는 함수이다. 먼저 배열의 요소의 개수를 행, 배열 요소의 모든 합을 열로 하는 long long 형태의 2차원 배열을 만든다.

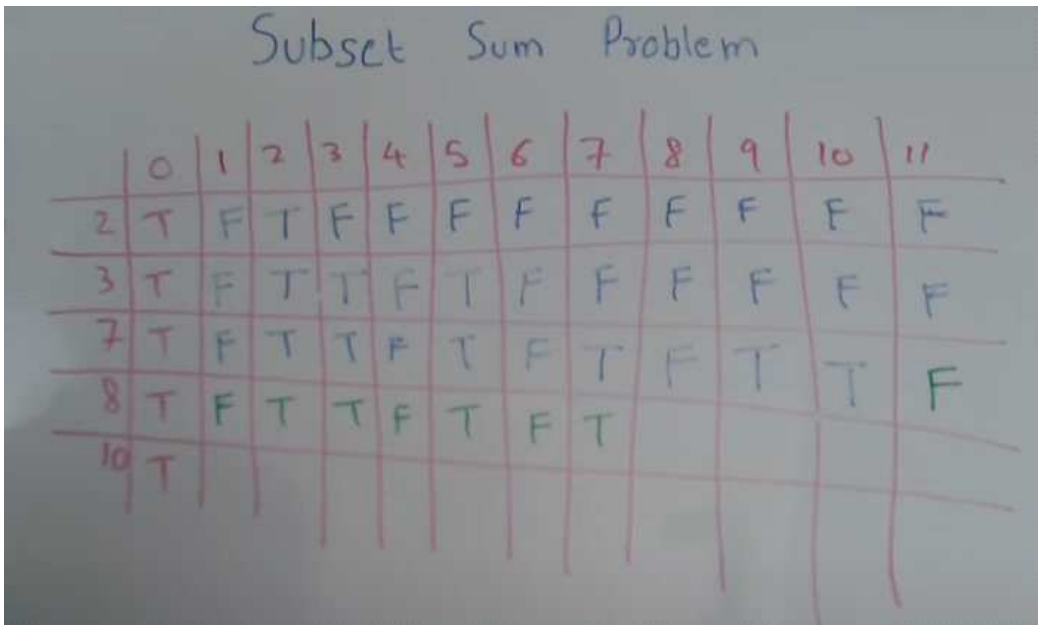
그리고 첫 번째 열 값에 1을 대입하는데 이유는 DP형식으로 값을 채워넣기 위함이다. 어떻게 보면 Boundary-Condition이라고 할 수도 있는데, 직관적으로 해석하자면 각각의 요소값으로 총합 0을 표현할 수 있냐는 물음을 **양수로 체킹**하는 것이다. 이것은 **I번째에 값의 가격이 0이면 모두 성공을 한다는 의미와 동일하다.** 또한 첫 번째 행에 0값을 대입하는 이유는 요소 0개로 값을 만들 수 없기 때문인데 **이것은 첫 번째부터 total의 모든 요소가 0번째이면 실패한다는 것을 의미한다. 그렇기 때문에 0을 대입한다.**

이 상태로 알고리즘이 시작된다.

```
//Dp fill-up
for (int i = 1; i <= length; i++)
{
    for (int j = 1; j <= total; j++)
    {

        if (j - arr[i - 1] >= 0)
        {
            T[i][j] = T[i - 1][j - arr[i - 1]] + T[i - 1][j]; //T[i][j] 값 업데이트 arr[i-1] 배열에 있는 수를 총합에서 빼준 카운
        }
        else
            T[i][j] = T[i - 1][j];

        if (T[i][j] / 2 > limit) //NUMEROUS 예외 처리
            flag = true;
    }
}

backTrackingDp(T, arr, length, total); //백트래킹으로 부분수열 벡터에 저장

ans = T[length][total] / 2;

//메모리 해제
for (int i = 0; i < length + 1; i++)
    delete[] T[i];
delete[] T;
```

처음 검사하는 조건은 **현재 검사하는 total이 현재 조사하는 값보다 같거나 크다면**
T[i][j]에는 **이전 행의 (현재 총합-arr[i-1])인 것**과 **이전 행의 현재 총합의 값**을 더해나간다. 이 과정은 현재 위치(i, 부분 수열의 총합 j)에서 가능한 부분집합의 수를 계속해서 DP 방식으로 계속 더해나가는 것이다. 근데 만약 그렇지 않다면 단순히 **이전 행의 (현재 총합 – arr[i-1])값**을 더하지 않고 **이전 행의 현재 총합의 값**만 더한다. 위 if문과 else문은 제일 **처음에 구현했던 재귀함수에서 sum/sum+arr[i] 기준으로 재귀를 두 개 돌린 것과 비슷한 로직이다**. 위 알고리즘을 반복되면 T[i][j]에는 파티션 할 수 있는 모든 경우의 수가 나온다. 물론 좌/우가 대칭이 되었을 때도 동일하게 카운팅 하기 때문에 /2를 해줘야한다. 그 값을 ans에 저장하고 반환해준다.

즉 현 위치에서 카운팅 된 값들의 합으로 다음 배열을 채우는 원리인데, 만약 j- arr[i – 1] >=0인 상태라면 그냥 이전 행에 있는 것을 그대로 가지고 온다.



이는 여기 보이는 8번 요소가 있는 3행에 초록색으로 칠한 것을 보면 2행과 값이 동일함을 알 수 있다.

이런 로직이며, F와 T가 나오는 대신, 위 설명과 다르게 그 위치에서 가능한 집합 카운팅 값을 누적해나가서 Fill-up 하는 것이 나의 알고리즘 코드의 구조이다.

또한 backTrackingDp 함수의 동작원리는 다음과 같다.

```cpp
void backTrackingDp(long long** T, int arr[], int n, int total)
{
    if (n == 0)
        return;

    if (T[n][total] != 0) //부분 집합이라면
    {
        if (T[n - 1][total] == 0) //이전 값 0이라면
        {
            leftArr.push_back(arr[n - 1]);
            backTrackingDp(T, arr, n - 1, total - arr[n - 1]);
        }
        else
            backTrackingDp(T, arr, n - 1, total);
    }
    else
        backTrackingDp(T, arr, n, total - 1);
}
```

backTrackingDp 함수는 subsetCount 내에서 동작하며 채워진 DP 2차원 배열을 인자로 받아, 부분 수열의 파티션 된 요소들의 집합을 구해주는 함수이다.

영상에서 봤듯이 마지막 요소 값부터 검사를 하는데 만약 0이 아닌 값이 들어있으면 부분집합이 있다고 간주하고 그 이전 행의 값이 0이라면 arr[n-1] 값을 leftArr에 넣고, backTrackingDp를 재귀적으로 실행하는데, total – arr[n-1]만큼 실행한다. 이 이유는 arr의 Column 값이 모든 배열 요소의 합이기 때문에 그만큼 뒤로 이동하는 것이다. 만약 이전 값이 0이 아니라면, n값을 하나 줄이고 함수를 실행시킨다. 또한 부분집합이 아니라면 아예 total 값을 –1 한 상태로 재귀함수를 호출한다.

위 함수가 시간이 재귀라서 굉장히 오래 걸릴 것 같지만 하나하나 해보면 그렇게 오래 걸리지 않는다.

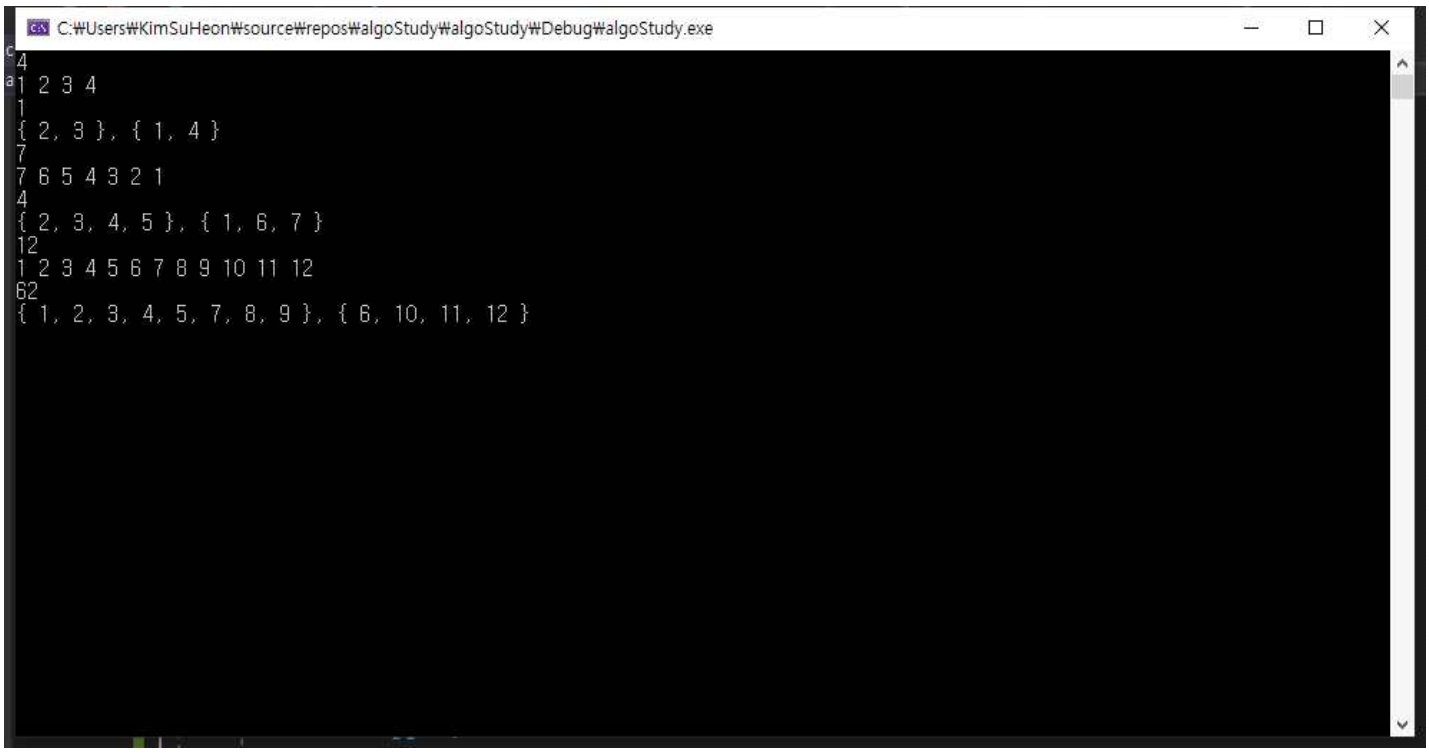
```
7
1 2 3 4 5 6 7
1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 2 1 1 1 0 0 0 0 0 0 0 0
1 1 2 2 2 2 2 1 1 1 0 0 0 0
1 1 2 2 3 3 3 3 3 3 2 2 1 1
1 1 2 2 3 4 4 4 5 5 5 5 4 4
1 1 2 2 3 4 5 5 6 7 7 8 8 8
4
{ 2, 3, 4, 5 }, { 1, 6, 7 }
12
1 2 3 4 5 6 7 8 9 10 11 12
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 2 2 2 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 2 3 3 3 3 3 3 2 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 2 3 4 4 4 5 5 5 5 4 4 4 3 2 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 2 2 3 4 5 5 6 7 7 8 8 8 8 8 7 7 6 5 5 4 3 2 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 2 2 3 4 5 6 7 8 9 10 11 12 13 13 13 14 13 13 13 12 11 10 9 8 7 6 5 4 3 2 2 1 1 1 0 0 0
1 1 2 2 3 4 5 6 8 9 10 12 13 15 17 18 19 21 21 22 23 23 23 23 22 21 21 19 18 17 15 13 12 10 9 8 6 5 4
1 1 2 2 3 4 5 6 8 10 11 13 15 17 20 22 24 27 29 31 33 35 36 38 39 39 40 40 39 39 38 36 35 33 31 29 27 24 22
1 1 2 2 3 4 5 6 8 10 12 14 16 19 22 25 28 32 35 39 43 46 49 53 56 59 62 64 66 68 69 69 70 69 69 68 66 64 62
1 1 2 2 3 4 5 6 8 10 12 15 17 20 24 27 31 36 40 45 51 56 61 67 72 78 84 89 94 100 104 108 113 115 118 121 122 123 124
62
{ 1, 2, 3, 4, 5, 7, 8, 9 }, { 6, 10, 11, 12 }
```

T배열을 출력했을 때의 모습이다. 7일 때, 위 배열을 따라 백트래킹을 진행하면
8 -> 4 -> 1 순서대로 올라가서 0을 만나면 5값이 담기게 되고 한칸 위로 이동 후 왼쪽으로 5만큼 뒤로 가게 된다. 그럼 바로 1을 만나는데 그 위에 값이 0이기 때문에 4가 담기고 위로 한칸 이동 후 왼쪽으로 4만큼 뒤로 이동하게 된다. 그러면 또 1을 만나는데 이때도 바로 한칸 위 값이 0이다. 때문에 3이 담기고 위로 한칸 이동 후 왼쪽으로 3만큼 이동한다. 2또한 같은 로직으로 하면 배열의 끝에 도달하게 되고 종료된다. 위 과정을 통해 {2, 3, 4, 5}라는 부분수열을 즉, 파티션 된 집합을 구할 수 있다.

테스트 케이스

테스트 케이스 설명에 앞서서 실험해본 결과, RAM에 한도에 따라 작동할 때가 있고 안 할 때가 있다는 것을 알게되었다. RAM 공간이 여유로울 때는 실행이 되는 반면, RAM 공간이 부족할 때는 프로그램에서 예외가 발생한다. 이 부분은 Try-Catch 문으로 처리를 해놨다.

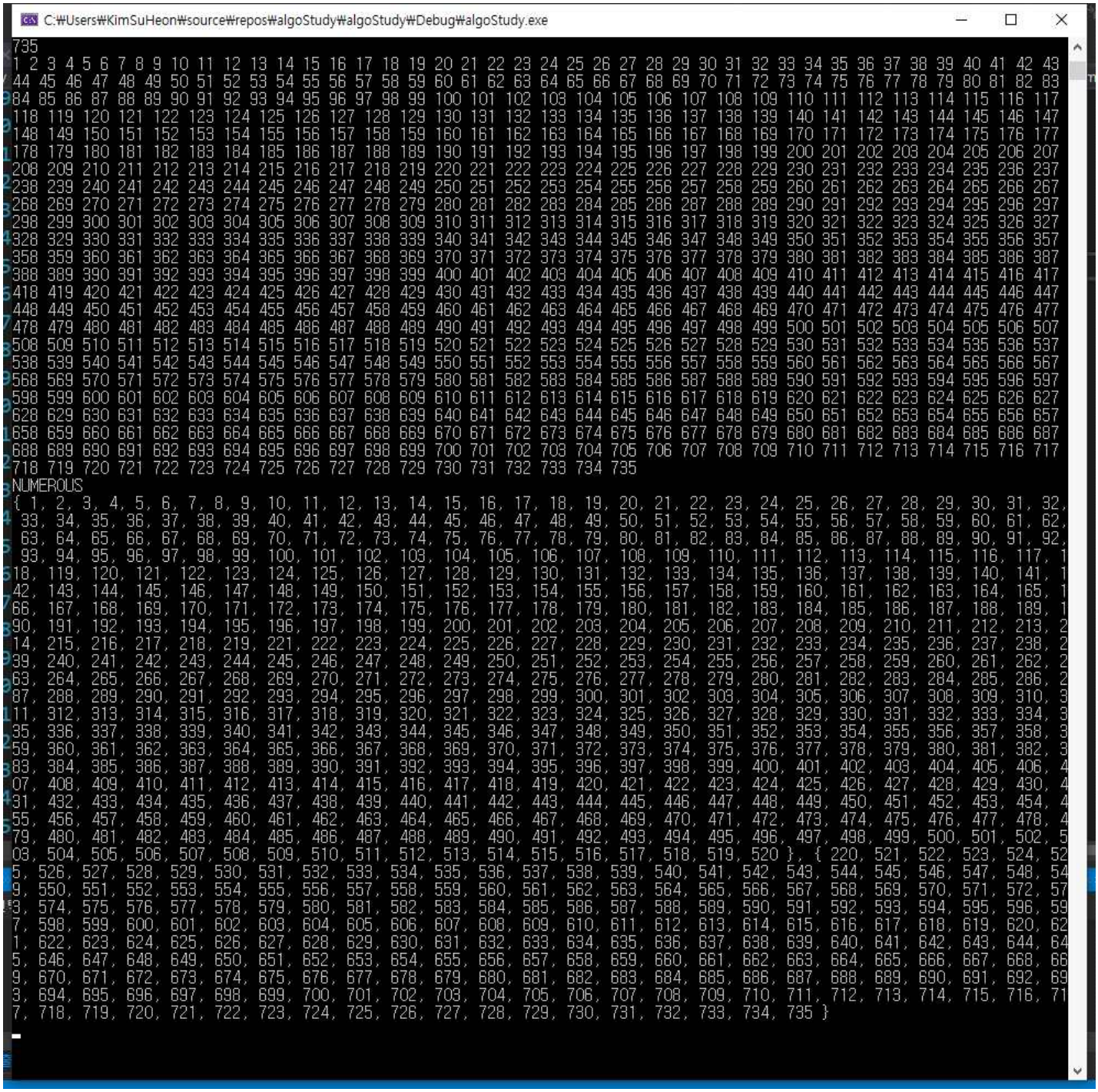
1, 7, 12 테스트 케이스



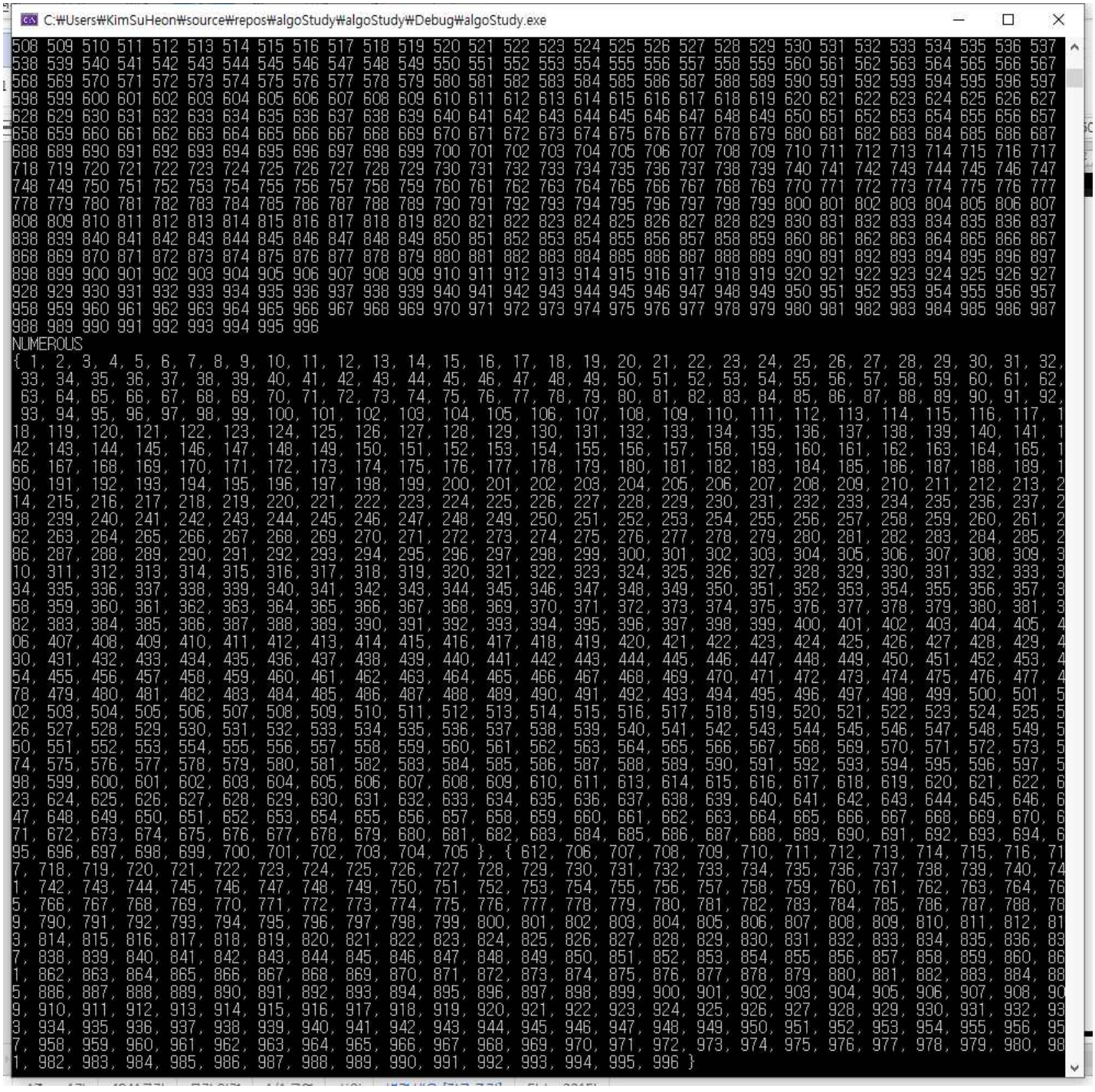
40, 44, 122, 219 테스트 케이스

```
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
2915017360
{ 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 }, { 25, 30, 31,
 32, 33, 34, 35, 36, 37, 38, 39, 40 }
44
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
NUMEROUS
{ 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 }, { 1,
 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44 }
122
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 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122
0
219
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 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219
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, 28, 29, 30, 31, 32,
 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155 }, { 45, 156, 157, 158, 159, 160, 161, 162, 163, 164, 16
5, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 18
9, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 21
3, 214, 215, 216, 217, 218, 219 }
```

735


C:₩Users₩KimSuHeon₩source₩repos₩algoStudy₩algoStudy₩Debug₩algoStudy.exe
735
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 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267
268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297
298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327
328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387
388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417
418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507
508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537
538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597
598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627
628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657
658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687
688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717
718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735
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, 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, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 1
18, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 1
42, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 1
66, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 1
90, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 2
14, 215, 216, 217, 218, 219, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 2
39, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 2
63, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 2
87, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 3
11, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 3
35, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 3
59, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 3
83, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 4
07, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 4
31, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 4
55, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 4
79, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 5
03, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520 }, { 220, 521, 522, 523, 524, 52
5, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 54
9, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 57
3, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 59
7, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 62
1, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 64
5, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 66
9, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 69
3, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 71
7, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735 }

996



각 테스트케이스가 정상적으로 작동함을 알 수 있습니다.

4. 간단한 소감

우선 DP와 백트래킹 개념에 대해 명확하게 이해하는 시간이 되었습니다. 평소 알고리즘 문제를 많이 풀어봤지만, 다른 문제는 전부 혼자 힘으로 해결할 수 있어도 DP 문제를 풀 때는 전혀 감을 잡을 수 없었습니다. 그런데 알고리즘 과목에서 DP를 정석적으로 배우고 과제도 DP에 대해서 풀어보다 보니, DP에 대한 자신감이 높아졌습니다. 하지만 제 자신이 아직 스스로 DP를 풀 정도의 실력은 아니라고 생각합니다. 코딩테스트를 나가게 되면 이런 문제들을 전부 혼자 힘으로 해결해야하는데 아직 그렇지 못한 제 자신을 보며 아직 스스로가 많이 부족하다고 생각합니다. 과제를 하면서 든 생각은 알고리즘을 틈틈이 더 준비해야겠다는 생각을 했습니다. 종합하자면 이번 HW2 과제를 통해서 나의 알고리즘 실력을 테스트해 볼 수 있었고, DP에 대한 자신감도 얻게 되었으며, 알고리즘에 대한 이해도 또한 쌓을 수 있는 시간을 보낸 것 같습니다. 실력 향상에 도움이 되는 좋은 과제를 내주신 교수님 및 조교님에게 감사드립니다.