Professional Coding Specialist

COS Pro JAVA 1 급

21 강-24 강. 모의고사 5 차

1. 모의고사 5 차(1-10 번)

과정 소개

COS Pro JAVA 1 급 모의고사 5 차를 풀어보며 문제 유형을 익히고, JAVA 를 이용하여 알고리즘을 구현하기 위해 필요한 관련 지식을 익혀보도록 한다.

학습 목차

1. 문제 1
2. 문제 2
3. 문제 3
4. 문제 4
5. 문제 5
6. 문제 6
7. 문제 7
8. 문제 8
9. 문제 9
10. 문제 10

학습 목표

1. YBM IT(www.ybmit.com) 에서 제공하는 COS Pro JAVA 1 급 모의고사(샘플 문제)를 풀어보며 JAVA 를 이용하여 주어진 문제를 해결하기 위한 알고리즘을 구성하는 능력을 배양한다.
2. 많이 등장하는 문제 유형을 익혀서 COS Pro 1 급 시험에 대비한다.

1. 문제 1

1) 문제 코드

```java
/*=======================================
  5차 1번   5차 1급 1_initial_code.java
 =======================================*/

class Solution {
    public int solution(int n) {
        int answer = 0;

        int[] steps = new int[n+1];
        steps[1] = 1;
        steps[2] = 2;
        steps[3] = 4;
        for(int i = 4; i <= n; i++)
            steps[i] = @@@;
        answer = steps[n];

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int n1 = 3;
        int ret1 = sol.solution(n1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int n2 = 4;
        int ret2 = sol.solution(n2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 코드를 완성하는 문제
- 한 번에 1 계단, 2 계단, 3 계단씩 오를 수 있을 때 매개변수로 전달된 n 개의 계단을 오르는 방법의 수를 구하는 코드를 동적 계획법의 타블레이션 방식을 이용하여 완성하는 문제

3) 동적 계획법(Dynamic Programming)

- 복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법
- 중복된 하위 문제들의 결과를 저장해 두었다가 상위 문제를 풀 때 그 결과를 재사용
- 타블레이션 방식은 재귀 호출을 하지 않고, 하위 문제들의 첫 번째부터 모든 결과를 저장해 두었다가 사용
- 메모이제이션 방식은 재귀 호출을 사용하지만, 하위 문제의 결과를 메모리에 저장한 후 재사용

4) 정답

- 주요 아이디어 정리

오르는 계단	오르는 방법	경우의 수
1 계단	1 계단	1
2 계단	1 계단+1 계단 2 계단	2
3 계단	1 계단+1 계단+1 계단 1 계단+2 계단 2 계단+1 계단 3 계단	4
4 계단	1 계단 오른 뒤 + 3 계단 오르는 법 사용 2 계단 오른 뒤 + 2 계단 오르는 법 사용 3 계단 오른 뒤 + 1 계단 오르는 법 사용	7
…	…	…
n 계단	1 계단 오른 뒤 + (n-1)계단 오르는 법 2 계단 오른 뒤 + (n-2)계단 오르는 법 3 계단 오른 뒤 + (n-3)계단 오르는 법	(n-1)경우의 수 +(n-2)경우의 수 +(n-3)경우의 수

➔ 4 개 이상 n 개의 계단을 오르는 방법의 수를 구하려면 1 계단 오른 뒤에 n-1 계단을 오르는 방법, 2 계단 오른 뒤에 n-2 계단을 오르는 방법, 3 계단 오른 뒤에 n-3 계단을 오르는 방법을 모두 더해야 함

➔ 1 개부터 n-1 개의 계단을 오르는 방법을 모두 구해야 n 개의 계단 오르는 방법을 구할 수 있음

- 정답 코드

```
public int solution(int n) {
    int answer = 0;

    int[] steps = new int[n+1];
    steps[1] = 1;
    steps[2] = 2;
    steps[3] = 4;
    for(int i = 4; i <= n; i++)
        steps[i] = steps[i-1] + steps[i-2] + steps[i-3]
    answer = steps[n];

    return answer;
}
```

①. 1 계단, 2 계단, 3 계단을 오르는 방법의 수를 steps 배열의 인덱스 번호 1, 2, 3 에 저장

②. for 문을 이용하여 4 계단부터 n 계단까지 오르는 방법의 수를 이전 세 개의 항목들을 이용하여 구함
steps 배열의 n 번째 항목을 answer 로 저장

5) 다른 코드 제안

① 동적 계획법 - 메모이제이션 방식으로 구현

```
public int solution(int[] steps, int n) {
    if (steps[n] > 0)
        return steps[n];

    if (n==1)
        steps[1] = 1;
    else if (n==2)
        steps[2] = 2;
    else if (n==3)
        steps[3] = 4;
    else
        steps[n]=solution(steps,n-1)+solution(steps,n-2)+solution(steps,n-3);

    return steps[n];
}
```

```
int n1 = 6;
int[] steps = new int[n1+1];
int ret1 = sol.solution(steps, n1);
```

② 재귀 함수 사용

```
int n1 = 6;
int[] steps = new int[n1+1];
int ret1 = sol.solution(steps, n1);
```

```
public int solution(int n) {
    if (n==1)
        return 1;
    else if (n==2)
        return 2;
    else if (n==3)
        return 4;
    else
        return solution(n-1)+solution(n-2)+solution(n-3);
}
```

```
int n1 = 6;
int ret1 = sol.solution(n1);
```

2. 문제 2

1) 문제 코드

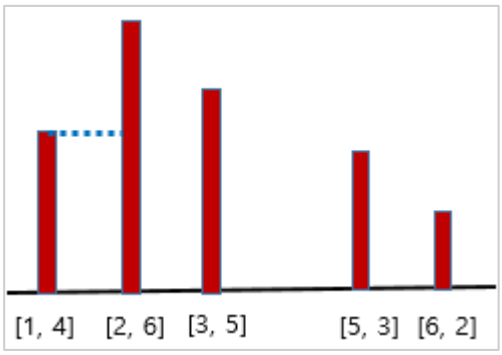
```java
/*======================================
   5차 2번   5차 1급 2_initial_code.java
 ======================================*/

class Solution {
    public int solution(int[][] walls) {
        int answer = 0;

        for(int i = 0; i < walls.length; i++) {
            for(int j = i+1; j < walls.length; j++) {
                int area = 0;
                if(walls[i][1] > walls[j][1])
                    area = walls[i][1] * (walls[j][0] - walls[i][0]);
                else
                    area = walls[j][1] * (walls[j][0] - walls[i][0]);

                if(answer < area)
                    answer = area;
            }
        }

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래에는 잘못된 부분이
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] walls = {{1, 4}, {2, 6}, {3, 5}, {5, 3}, {6, 2}};
        int ret = sol.solution(walls);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 벽의 위치와 높이를 2 차원 배열로 저장한 매개변수 walls 에서 각 항목값을 이용하여 최대로 담을 수 있는 물의 용량을 찾아내는 프로그램에서 잘못된 부분을 찾아 수정해야 함

3) 정답

- 주요 아이디어 정리 : walls = [[1,4], [2,6], [3,5], [5,3], [6,2]] 인 경우



- ✓ walls 의 항목 구성 = [벽의 위치, 벽의 높이]
  - walls[0][0] 은 0 번째 벽의 위치=1
  - walls[0][1] 은 0 번째 벽의 높이=4
  - n 번째 벽의 위치 값 = walls[n][0]
  - n 번째 벽의 높이 값 = walls[n][1]
- ✓ 두 벽 사이에 물을 담을 수 있는 용량 = 두 벽 중 낮은 벽의 높이 * 두 벽 사이의 거리
  - 0 번째 벽과 1 번째 벽 사이에 물을 담을 수 있는 용량 = 0 번째 벽의 높이 * (1 번째 벽 위치 - 0 번째 벽 위치)
    = walls[0][1] * (walls[1][0] - walls[0][0])
    ∵ 0 번째 벽의 높이가 낮기 때문

➔ 주어진 walls 를 이용해서 물을 담을 수 있는 최대 용량을 구하려면 중첩 for 문을 이용하여 두 개의 벽을 가져와 물을 담을 수 있는 용량을 계산하고 그 중 최댓값을 찾음

- 정답 코드

```
public int solution(int[][] walls) {
    int answer = 0;

①   for(int i = 0; i < walls.length; i++) {
②       for(int j = i+1; j < walls.length; j++) {
            int area = 0;
③           if(walls[i][1] < walls[j][1])
                area = walls[i][1] * (walls[j][0] - walls[i][0]);
④           else
                area = walls[j][1] * (walls[j][0] - walls[i][0]);

⑤           if(answer < area)
                answer = area;
        }
    }

    return answer;
}
```

① 바깥쪽 for 문을 이용하여 하나의 벽에 대한 인덱스를 i 로 받아오고, 안쪽 for 문을 이용하여 그 다음에 위치하는 벽에 대한 인덱스를 j 로 받음

② 물의 용량을 계산하는 변수 area 를 0 으로 초기화

③ i 번째 벽 높이가 그 다음에 위치한 j 번째 벽보다 높이가 낮으면 물의 용량은 i 번째 벽의 높이 * (j 번째 벽의 위치 - i 번째 벽의 위치)로 계산. 문제 코드는 조건식으로 사용한 벽 높이의 비교 연산식이 반대로 작성되었음

④ i 번째 벽 높이가 그 다음에 위치한 j 번째 벽보다 높이가 낮지 않으면 물의 용량은 j 번째 벽의 높이 * (j 번째 벽의 위치 - i 번째 벽의 위치)로 계산

⑤. 최대 담수 용량을 가지고 있는 answer 가 현재 계산한 용량 area 보다 적으면 answer 를 area 로 재할당

3. 문제 3

1) 문제 코드

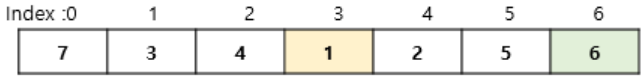
```java
/*=======================================
   5차 3번   5차 1급 3_initial_code.java
 =======================================*/

import java.util.*;

class Solution {
    public void swap(int[] numbers, int i, int j) {
        int tmp = numbers[i];
        numbers[i] = numbers[j];
        numbers[j] = tmp;
    }

    public int[] solution(int[] numbers) {
        int[] answer = {};

        Arrays.sort(numbers);
        int mid = (numbers.length - 1) / 2;
        swap(numbers, mid, numbers.length-1);

        int left = mid + 1;
        int right = numbers.length - 1;
        while(left <= right) {
            swap(numbers, left, right);
            left = left + 1;
            right = right - 1;
        }

        answer = numbers;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] numbers = {7, 3, 4, 1, 2, 5, 6};
        int[] ret = sol.solution(numbers);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.print("solution 메소드의 반환 값은 {");
        for(int i = 0; i < ret.length; i++) {
            if(i != 0)
                System.out.print(", ");
            System.out.print(ret[i]);
        }
        System.out.println("} 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 주어진 배열에서 앞의 절반까지는 오름차순으로 정렬하고 절반의 바로 다음 항목부터 마지막 항목까지는 내림차순으로 정렬하도록 작성된 프로그램에서 잘못된 부분을 찾아 수정해야 함 (문제 조건으로 배열의 항목 개수는 홀수로 지정됨)

3) 정답

- 주요 아이디어 정리: 배열의 가운데 인덱스와 마지막 인덱스 구하기 - 배열의 항목 개수가 = 7 인 경우



- 가운데 항목의 인덱스 = (배열의 길이(7)-1)/2=3
- 마지막 항목의 인덱스 = 배열의 길이(7) -1

- 정답 코드

```
class Solution {
    public void swap(int[] numbers, int i, int j) {
        int tmp = numbers[i];
        numbers[i] = numbers[j];
        numbers[j] = tmp;
    }

    public int[] solution(int[] numbers) {
        int[] answer = {};

①       Arrays.sort(numbers);
        int mid = (numbers.length - 1) / 2;
②       swap(numbers, mid, numbers.length-1);

③       int left = mid + 1;
        int right = numbers.length - 2;
④       while(left <= right) {
⑤           swap(numbers, left, right);
⑥           left = left + 1;
            right = right - 1;
        }

        answer = numbers;
        return answer;
    }
}
```

①. 배열 numbers 를 Arrays.sort()를 이용해 정렬 후 가운데 인덱스의 값을 구하여 mid 에 저장

- 배열 인덱스는 0 부터 시작하기 때문에 (배열 길이-1) 을 2 로 나눈 몫으로 가운데 인덱스 값을 구함

② 가운데 인덱스가 나타내는 항목값과 마지막 인덱스가 나타내는 항목값을 맞교환
③ 배열의 가운데 이후의 항목부터 배열의 마지막에서 두 번째 항목까지 내림차순으로 정렬하기 위해서 (가운데 인덱스 값 + 1)을 left 로, 마지막에서 두 번째 항목에 대한 인덱스를 right 로 지정. 배열의 마지막 항목은 이미 앞줄의 코드를 통해서 교환이 이루어졌기 때문에 문제 코드에서 제시된 대로 right 를 len(numbers) - 1 로 지정하면 원하는 결과를 얻을 수 없음
④ left 값이 right 값보다 작거나 같은 동안
⑤ left 값에 해당하는 인덱스의 항목과 right 값에 해당하는 인덱스의 항목을 맞교환
⑥ left 는 1 만큼 증가, right 는 1 만큼 감소하는 작업을 반복 실행

4. 문제 4

1) 문제 코드

```java
/*=====================================
   5차 4번   5차 1급 4_initial_code.java
 =====================================*/

class Solution {
    public String solution(int number) {
        String answer = "";

        int[] numberCount = new int[10];
        while(number > 0) {
            numberCount[number % 10]++;
            number /= 10;
        }

        for(int i = 0; i < 10; i++)
            if(numberCount[i] != 0)
                answer += (String.valueOf(i) + String.valueOf(numberCount[i]));

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. 아래에는 잘못된 부분이 없으
    public static void main(String[] args) {
        Solution sol = new Solution();
        int number1 = 2433;
        String ret1 = sol.solution(number1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int number2 = 662244;
        String ret2 = sol.solution(number2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 잘못된 부분을 찾아 수정하는 문제

- 매개변수 number 로 전달받은 수에서 사용된 숫자들과 각 숫자가 사용된 횟수를 return 하되 큰 숫자부터 먼저 문자열에 나타나도록 작성한 프로그램에서 잘못된 부분을 찾아 수정하는 문제
- 0 부터 9 까지의 인덱스를 갖는 배열에 number 에서 사용된 숫자별 빈도수를 집계

3) 정답

- 주요 아이디어 정리 : 어떤 수에서 일의 자리부터 각 자리 숫자를 가져오기
  ex) 2307 에서 각 자리 숫자를 추출하기 위한 절차

실행 순서	구분	필요한 계산식	결과
1	일의 자리 숫자	2307 % 10	7
2	십의 자리 숫자	2307 // 10 의 결과 230 → 230 % 10	0
3	백의 자리 숫자	230 // 10 의 결과 23 → 23 % 10	3
4	천의 자리 숫자	23 // 10 의 결과 2 → 2 % 10	2

- 정답 코드

```
public String solution(int number) {
    String answer = "";

①   int[] numberCount = new int[10];
②   while(number > 0) {
        numberCount[number % 10]++;
        number /= 10;
    }

③   for(int i = 9; i >= 0; i--)
④       if(numberCount[i] != 0)
            answer += (String.valueOf(i) + String.valueOf(numberCount[i]));

    return answer;
}
```

①. 1 부터 9 까지의 숫자가 나타나는 빈도수를 집계하기 위해 10 개의 항목을 갖는 배열 numberCount 를 생성하고 0 으로 초기화

②. number 가 0 보다 큰 동안 반복

- number 를 10 으로 나눈 나머지 값에 해당하는 인덱스의 항목 값을 1 만큼 증가
- number 를 10 으로 나눈 몫으로 number 값을 재설정

③. 큰 숫자부터 결과 문자열에 붙이기 위해 9 부터 1 까지의 수가 차례로 i 로 할당됨. 문제에서 제시된 대로 for 문의 반복 범위를 지정할 때 0 부터 9 까지 변화되면, 0 부터 9 까지의 수가 i 에 할당되고 결과 문자열을 저장하는 answer 에는 작은 수의 빈도 수부터 나타나게 됨

④. 숫자 i 의 빈도수를 저장한 numbeCount[i]의 값이 0 이 아니면 answer 에 저장된 문자열 끝에 i 를 문자열로 변환한 값과 numberCount[i]을 문자열로 변환한 값을 차례로 덧붙임

5. 문제 5

1) 문제 코드

```java
/*=====================================
   5차 5번   5차 1급 5_initial_code.java
 =====================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public int solution(int[] enemies, int[] armies) {
        // 여기에 코드를 작성해주세요.
        int answer = 0;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] enemies1 = {1, 4, 3};
        int[] armies1 = {1, 3};
        int ret1 = sol.solution(enemies1, armies1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int[] enemies2 = {1, 1, 1};
        int[] armies2 = {1, 2, 3, 4};
        int ret2 = sol.solution(enemies2, armies2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- armies 배열의 항목값을 이용하여 enemies 배열의 항목값을 이길 수 있는 최대 개수를 구하도록 프로그램을 작성
- armies 배열과 enemies 배열의 항목값을 정렬한 후 각각의 배열에서 하나씩 항목을 가져와 이길 수 있는 지 비교해서 이길 수 있는 항목 개수를 구해야 함
- enemies 의 항목을 이기려면 armies 의 항목값이 크거나 같아야 함

3) 정답

```
public int solution(int[] enemies, int[] armies) {
    int answer = 0;

①   Arrays.sort(enemies);
    Arrays.sort(armies);
②   int i = 0, j = 0;
③   while(i < enemies.length && j < armies.length) {
        if(enemies[i] <= armies[j]) {
            answer++;
④           i++;
            j++;
        } else {
⑤           j++;
        }
    }
    return answer;
}
```

① enemies 배열과 armies 배열을 오름차순으로 정렬

② enemies 의 인덱스로 사용될 변수 i 와 armies 의 인덱스로 사용될 변수 j 를 0 으로 초기화

③ i 값이 enemies 의 항목 개수보다 작고 j 값이 armies 의 항목 개수보다 작은 동안 반복. 즉, 두 배열에서 비교할 항목이 모두 존재할 동안 반복문 안쪽의 명령들을 실행

④ i 번째 enemies 의 값이 j 번째 armies 값보다 작거나 같으면 armies 가 승리하는 횟수를 저장하는 변수 answer 를 1 만큼 증가시키고 i 와 j 를 모두 1 만큼 증가

⑤ i 번째 enemies 의 값이 j 번째 armies 값보다 크면 armies 의 인덱스인 j 만 1 만큼 증가하여 armies 의 다음 항목을 가리키도록 조정

6. 문제 6

1) 문제 코드

```java
/*=======================================
   5차 6번   5차 1급 6_initial_code.java
 =======================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public String solution(String s1, String s2, int p, int q) {
        // 여기에 코드를 작성해주세요.
        String answer = "";
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        String s1 = new String("112001");
        String s2 = new String("12010");
        int p = 3;
        int q = 8;
        String ret = sol.solution(s1, s2, p, q);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- p 진법 수 두 개가 문자열로 형태로 전달된 것을 더하고 그 결과를 q 진법 수의 문자열로 return 하도록 코드를 작성해야 함

3) 정답

- 주요 아이디어 정리

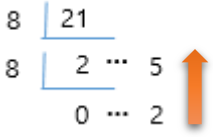
✓ P 진법 수 → 10 진법 수로 변환하기

ex) 3 진법 수 112001 $=1*3^0 + 0*3 + 0*3^2 + 2*3^3 + 1*3^4 + 1*3^5$ = 10 진법 수 379

: 오른쪽 끝의 마지막 자리 숫자부터 각 자리 숫자가 의미하는 수를 곱한 것을 더하여 십진수 값을 구함.

✓ 10 진법 수 → P 진법 수로 변환하기

ex) 10 진법 수 21 = = 8 진법 수 25



: 21 을 8 로 나눈 몫이 0 이 될 때까지 나누고,
나누는 과정에서 나온 나머지들을 역순으로
가져와 8 진수의 각 자리 숫자로 사용.

✓ 형 변환 함수

int(실수 or 문자열, [base]) :	실수의 정수부만 가져오거나 정수로 된 문자열을 정수 형태로 가져옴 매개변수 base(진법) 에 대해서 추가로 값을 지정하면 해당 진법으로 표현된 문자열을 십진수 정수로 변환함 ex) int(1.2) → 1, int('12') → 12, int('110', 2) → 5
str(실수 or 정수) :	실수나 정수로 전달된 값을 문자열 형태로 변환 ex) str(1.2) → '1.2', str(12) → '12'

- 정답 코드

```
class Solution {
    int [] numbers_int = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    char [] numbers_char = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

① int char_to_int(char ch) {
        for (int i = 0; i < 10; i++)
            if (ch == numbers_char[i])
                return numbers_int[i];
        return 0;
    }

② char int_to_char(int val) {
        for (int i = 0; i < 10; i++)
            if (val == numbers_int[i])
                return numbers_char[i];
        return 0;
    }

③ String convert_scale(int num, int q) {
        if (num == 0) return "";
        return convert_scale(num / q, q) + int_to_char(num % q);
    }

④ public int parse_decimal(String s, int p) {
        int num = 0;
        for (int i = s.length() - 1, mul = 1; i >= 0; i--, mul *= p) {
            num += char_to_int(s.charAt(i)) * mul;
        }
        return num;
    }

    public String solution(String s1, String s2, int p, int q) {
        int num1 = parse_decimal(s1, p);
        int num2 = parse_decimal(s2, p);
        String answer = convert_scale(num1 + num2, q);
        return answer;
    }
}
```

① char_to_int( ) 함수는 정수로 된 문자를 정수로 변환하는 것을 구현한 함수
- 매개변수 ch 와 같은 문자를 갖는 numbers_char 배열의 항목을 찾아서 해당 항목의 인덱스와 같은 인덱스를 갖는 numbers_int 배열의 항목을 return

② int_to_char( ) 함수는 정수를 문자로 변환하는 것을 구현한 함수
- 매개변수 val 와 같은 값을 갖는 numbers_int 배열의 항목을 찾아서 해당 항목의 인덱스와 같은 인덱스를 갖는 numbers_char 배열의 항목을 return

③ convert_scale( ) 함수는 십진수를 q 진법으로 변환하는 것을 구현한 함수
- 매개변수 num 을 q 로 나눈 몫이 0 이 아닌 동안 num 을 q 로 나눈 몫을 인수로 전달하여 재귀호출하고 동시에 num 을 q 로 나눈 나머지를 문자열로 변환하여 재귀호출한 결과 뒤에 붙여서 return

④ parse_decimal( ) 함수는 p 진법 수를 십진수로 변환하는 것을 구현한 함수
- for 문을 이용하여 매개변수로 받은 정수 문자열의 마지막 문자부터 가져와 그 문자와 10 의 거듭제곱을 곱한 것을 누적 합산하여 십진수로 변환

7. 문제 7

1) 문제 코드

```java
/*=======================================
   5차 7번   5차 1급 7_initial_code.java
 =======================================*/
class Solution {
    public int find(int[] parent, int u) {
        if(u == parent[u])
            return u;

        parent[u] = @@@;
        return parent[u];
    }

    public boolean merge(int[] parent, int u, int v) {
        u = find(parent, u);
        v = find(parent, v);

        if(u == v)
            return true;

        @@@;
        return false;
    }

    public int solution(int n, int[][] connections) {
        int answer = 0;

        int[] parent = new int[n+1];
        for(int i = 1; i <= n; i++)
            @@@;

        for(int i = 0; i < connections.length; i++)
            if(merge(parent, connections[i][0], connections[i][1])) {
                answer = i + 1;
                break;
            }
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int n = 3;
        int[][] connections = {{1, 2}, {1, 3}, {2, 3}};
        int ret = sol.solution(n, connections);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

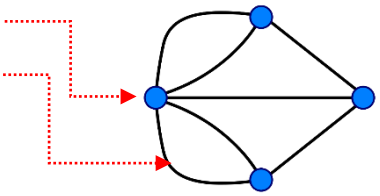
2) 문제 개요

- 제시된 과제를 해결하기 위해 작성된 프로그램에서 빈 곳을 채우는 문제
- 주어진 노드와 노드 연결 순서에 의해서 사이클이 생성되는 연결 순서가 몇 번인지 찾아서 return 하는 프로그램에서 빈 곳에 알맞은 코드를 채워 넣는 문제
- 사이클을 찾기 위해 Union-Find 알고리즘이 사용됨

3) 그래프 이론 정리

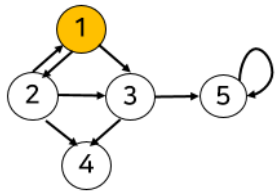
① 그래프(Graph) 소개

- 레온하르트 오일러가 쾨니히스베르크의 다리 문제를 풀기 위해 처음 도입한 이론
- 7 개의 다리를 한 번씩만 건너서 출발지로 다시 돌아올 수 있는 지 확인하는 문제
- 한 붓 그리기, 오일러의 경로는 모든 간선을 한 번씩 방문하는 유한 그래프의 대표적인 예
- 그래프의 구성
  - 정점(Vertex) 혹은 노드(Node) : 그래프의 각 지점
  - 간선(Edge) : 노드 사이를 연결하는 선을 간선

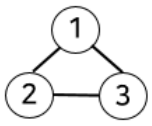


② 그래프 용어 정리

- 그래프는 (V, E)의 쌍으로 표현
- 각 정점(Vertex) 혹은 노드(Node)는 그래프의 특정 지점을 나타내며 숫자 혹은 알파벳으로 표시
- 간선(Edge) : 두 정점을 잇는 개체로 방향성이 있는 그래프에서는 화살표로 표시하고 방향이 없는 그래프에서는 일반 선으로 표시
- 경로(Path) : 거쳐가는 정점들의 순서를 열거한 것
  ex) 2 에서 5 까지의 경로(2→3→5 or 2→1→3→5)
- 경로의 길이(Path Length) : 간선의 수 or 가중치
- 차수(Degree) : 정점에 연결되어 있는 선의 개수
  - 입력차수(in-degree) : 정점으로 들어오는 선의 수
  - 출력차수(out-degree) : 정점에서 나가는 선의 수.
- 인접한다 : 정점과 정점 사이에 간선이 존재하는 것을 표현하는 용어
  - 방향성 그래프 : 3 은 1 에 인접한다. / 1 은 3 에 인접하지 않는다.
  - 무 방향성 그래프 : 1 과 3 은 서로 인접한다.



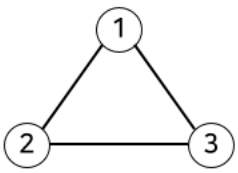
Directed Graph



Undirected Graph

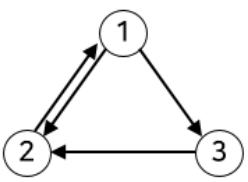
③ 그래프의 종류

- 무방향 그래프(Undirected Graph) : 간선에 방향이 없는 그래프



노드의 집합 V = { 1, 2, 3 }
간선의 집합 E = { (1, 2), (1, 3), (2, 3) }
= { (2, 1), (3, 1), (2, 3) }
.

- 방향 그래프(Directed Graph) : 간선에 방향이 있는 그래프

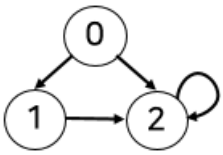


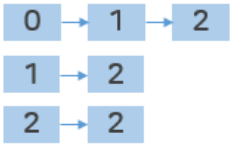
노드의 집합 V = { 1, 2, 3 }
간선의 집합 E = { (1, 2), (1, 3), (2, 1), (3, 2) }
(출발지, 도착지) 순으로 표현

④ 그래프의 표현

- 인접 배열(Adjacency List)
  - 출발 노드를 키로 하여 출발 노드에서 갈 수 있는 노드를 값으로 표현

Directed Graph

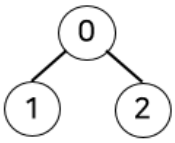


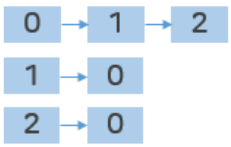


0 노드에서 갈 수 있는 곳은 1, 2
1 노드에서 갈 수 있는 곳은 2
2 노드에서 갈 수 있는 곳은 2

E={ (0, 1), (0, 2), (1, 2), (2, 2) }

Undirected Graph



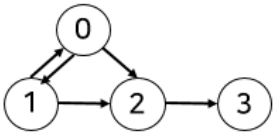


0 노드에서 갈 수 있는 곳은 1, 2
1 노드에서 갈 수 있는 곳은 0
2 노드에서 갈 수 있는 곳은 0

E={ (0, 1), (0, 2), (1, 0), (2, 0) }
프로그래밍할 때는 양 방향 경로를 모두 지정해야 함

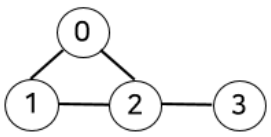
- 인접 행렬(Adjacency Matrix)
  - 행 번호를 출발 노드, 열 번호를 도착 노드로 사용
  - 출발 노드에서 도착 노드로 갈 수 있는 곳에 대해서 각 번호가 교차하는 지점의 값을 1 로 할당

Directed Graph



	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	0	0	0	1
3	0	0	0	0

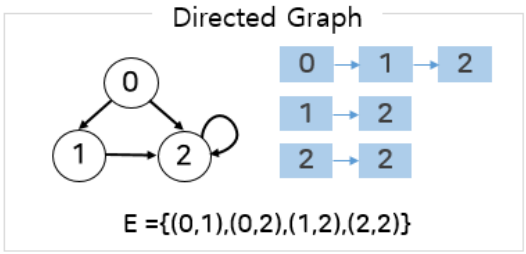
Undirected Graph



	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

⑤ 그래프를 JAVA 코드로 표현하는 방법

ex) 아래와 같은 Directed Graph 표현하기



- 인접 배열(Adjacency List) : 2 차원 배열을 활용 (여기서는 ArrayList 로 구현)

```
①ArrayList<Integer[]> gl = new ArrayList<Integer[]>();

②gl.add(new Integer[]{1,2});
 gl.add(new Integer[]{2});
 gl.add(new Integer[]{2});

 for(int i=0; i<gl.size(); i++) {
     for(int j=0; j<gl.get(i).length; j++)
         System.out.print(gl.get(i)[j]+" ");
     System.out.println();
 }
```

①. Integer 배열을 갖는 ArrayList 생성

②. 각 노드에서 갈 수 있는 노드 값 추가

- 0 번 노드에서 갈 수 있는 1, 2 를 0 번 배열의 항목 값으로 추가
- 1 번 노드에서 갈 수 있는 2 를 1 번 배열의 항목 값으로 추가
- 2 번 노드에서 갈 수 있는 2 를 2 번 배열의 항목 값으로 추가

〈 결과 〉

```
[[1, 2], [2], [2]]
 0     1    2
```

- 인접 행렬(Adjacency Matrix)

```
int[][] gm = new int[][] {
             {0,1,1},
             {0,0,1},
             {0,0,1}
             };
```

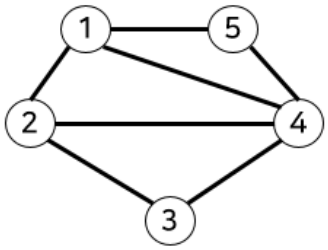
- 0 번 노드에서 갈 수 있는 1, 2 를 표현하기 위해 gm[0][1] 과 gm[0][2] 를 1 로 할당
- 1 번 노드에서 갈 수 있는 2 를 표현하기 위해 gm[1][2] 를 1 로 할당
- 2 번 노드에서 갈 수 있는 2 를 표현하기 위해 gm[2][2] 를 1 로 할당

〈 결과 〉

```
[[0, 1, 0], [0, 0, 1], [0, 0, 1]]
```

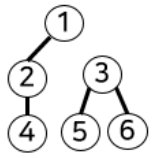
⑥ 그래프(Graph)와 순환(Cycle)

- 순환(Cycle) : 그래프에서 출발점과 도착점이 같은 경로를 말함(3→4→5→1→4→2→3)
- 단일 순환(Simple Cycle)
  - 순환 경로에서 거치는 모든 정점이 다른 경우
  - 반복되는 정점이 출발점과 도착점
  - 3→4→5→1→2→3

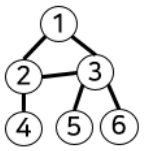


⑦ 그래프의 종류

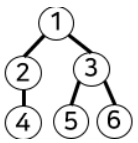
- 비순환 그래프(Acyclic Graph)
  - 순환이 없는 그래프
- 연결 그래프(Connected Graph)
  - 모든 정점 사이에 두 정점을 양 끝으로 하는 경로가 존재하는 그래프
- 트리(Tree)
  - 연결된 비순환 무방향성 그래프



비순환 그래프
무방향 그래프



연결 그래프
무방향 그래프

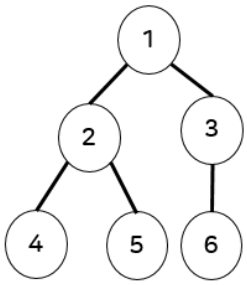


연결 그래프
비순환 그래프
무방향 그래프

⑧ 그래프 순회(Graph Traversals)

- 그래프 탐색(Graph Search) 라고도 함
- 그래프에 있는 모든 노드들을 방문하는 방법
- 깊이 우선 탐색 : Depth First Search(DFS)
  - 스택, 재귀로 구현
  - 백트래킹에 유용
  - 세로 방향에 있는 노드를 먼저 방문하는 방법
  - 오른쪽 트리의 노드 방문 순서
    1→2→4→5→3→6
- 너비 우선 탐색 : Breadth First Search(BFS)
  - 큐로 구현
  - 그래프의 최단 경로를 구하는 문제에 유용
  - 가로 방향에 있는 노드를 먼저 방문하는 방법
  - 오른쪽 트리의 노드 방문 순서
    1→2→3→4→5→6

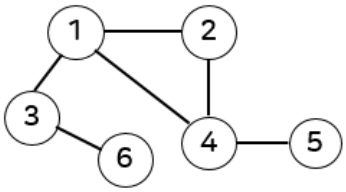
ex)



⑨ 그래프(Graph)에서 그래프 순회

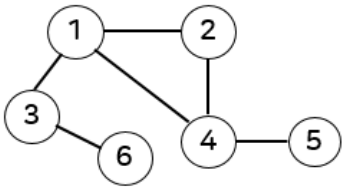
- 깊이 우선 탐색 : Depth First Search(DFS)
  - 스택 구조를 사용
  - 오른쪽 그래프의 노드 방문 순서
    1→2→4→5→3→6

ex)



- 너비 우선 탐색 : Breadth First Search(BFS)
  - 시작점에서 거리(간선의 수)를 하나씩 늘리면서 찾음
  - 오른쪽 그래프의 노드 방문 순서 1→2→3→4→6→5

⑩ 깊이 우선 탐색(DFS)의 구현
: 탐색할 그래프



- DFS Stack 사용

```
public static int[][] graph  = new int[][]{
        {}, {2,3,4}, {1,4}, {1,6}, {1,2,5}, {4}, {3}     };
public static boolean[] visited = new boolean[graph.length];

public static void DFS(int nodeIdx) {
    Stack<Integer> stack=new Stack<Integer>();
    int next;

    stack.push(nodeIdx);            //첫 번째 노드(1)를 스택에 추가
    visited[nodeIdx]=true;          //현재 노드를 방문 처리하고 출력
    System.out.print(nodeIdx+" ");

    while(!stack.isEmpty()){       //스택이 비어 있지 않은 동안
       int curNode = stack.peek();     //스택의 top 보기

       for(next=0; next< graph[curNode].length;next++) {  //현재 노드에 연결된 노드들을 탐색
              int nextNode=graph[curNode][next];          //현재 노드에 연결된 다음 노드
              if(!visited[nextNode]) {                    //방문한 적이 없으면
                  stack.push(nextNode);                   //스택에 추가
                  visited[nextNode]=true;                 //현재 노드를 방문 처리하고 출력
                  System.out.print(nextNode +" ");
                  break;                                  //dfs이므로 for문 빠져 나감
               }
        }
       if ( next == graph[curNode].length)   //해당 노드에서 연결된 모든 원소를 방문하면
           stack.pop();                      //스택에서 맨 위의 것을 꺼내기
   }
}
```

- 방문한 노드를 저장할 배열 visited 생성
- 다음에 방문할 노드로 시작 노드를 추가

- 현재 노드를 방문한 적이 없으면 visited 에 추가하고 다음 노드가 방문한 적이 없으면 스택에 추가

〈 결과 〉

```
['1', '2', '4', '5', '3', '6']
```

- DFS 재귀호출 사용

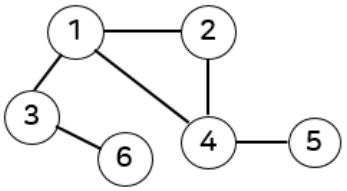
```
public static int[][] graph  = new int[][]{
        {}, {2,3,4}, {1,4}, {1,6}, {1,2,5}, {4}, {3}     };

public static boolean[] visited = new boolean[graph.length];

public static void DFS(int nodeIdx) {

    visited[nodeIdx]=true;            //1번 노드 방문으로 표시
    System.out.print(nodeIdx+" ");    //방문한 노드 번호 출력하기

    //다음 인접한 노드 찾기
    for (int nextNode: graph[nodeIdx]) {
        if (!visited[nextNode])       //방문한 적이 없으면 재귀호출(DFS)
                DFS(nextNode);
        }
}

public static void main(String[] args) {
    //1번 노드에서 시작
   DFS(1);
}
```

- v 노드를 방문한 것으로 체크
- 방문한 순서를 저장하는 result 에 v 를 추가
- v 노드와 연결되어 있는 노드를 방문한 적이 없으면 재귀 호출

〈 결과 〉

```
[1, 2, 4, 5, 3, 6]
```

① 너비 우선 탐색(BFS)의 구현
: 탐색할 그래프



- BFS Queue 사용

```java
public static int[][] graph  = new int[][]{
        {}, {2,3,4}, {1,4}, {1,6}, {1,2,5}, {4}, {3}     };

public static boolean[] visited = new boolean[graph.length];

public static void BFS(int nodeIdx) {
    Queue<Integer> q=new LinkedList<Integer>();

    q.offer(nodeIdx);                //첫 번째 노드(1)를 큐에 추가
    visited[nodeIdx]=true;           //현재 노드를 방문 처리하고 출력
    System.out.print(nodeIdx+" ");

    while(!q.isEmpty()) {

        int curNode=q.poll();                    //큐 맨 앞 노드 꺼내기

        //현재 노드와 인접한 모든 노드를 확인하면서
        for(int i=0; i<graph[curNode].length;i++) {
            int nextNode = graph[curNode][i];
            if(!visited[nextNode]) {             //방문한 적이 없으면
                q.offer(nextNode);               //큐에 추가하고
                visited[nextNode]=true;          //방문 처리하고 출력
                System.out.print(nextNode+" ");
            }
        }
    }
}
```

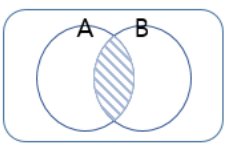
- 첫 번째 노드를 큐에 추가
- start 노드를 방문 처리하고 출력
- q 에 데이터가 있는 동안 q 의 가장 왼쪽(앞쪽) 데이터를 꺼내서 curNode 에 넣기
- 현재 노드와 인접한 모든 노드를 확인하면서 방문한 적이 없으면 큐에 추가하고 방문 처리

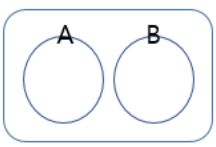
〈 결과 〉

```
[1, 2, 3, 4, 6, 5]
```

4) 합집합 - 찾기(Union find) 알고리즘

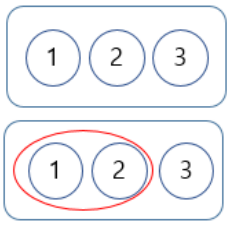
- 서로소 집합(disjoint set) 자료 구조 또는 병합 찾기 집합(merge find set) 자료 구조라고도 함
- 서로소 집합인지 판별하기 위해 사용하는 알고리즘
- 어떤 두 노드를 선택했을 때 두 노드가 같은 그래프에 속하는 지 찾아내는 알고리즘
- 무방향 그래프에서 사이클 판별 시에 사용됨





오른쪽 그림처럼 공통된 부분이 없는 두 집합의 관계를 '서로소'라고 함

- 어떤 노드 두 개를 선택했을 때 두 노드가 같은 그래프에 속하는 지 찾아내는 알고리즘으로 아래 두 가지 연산을 수행
  - 찾기(Find) : 대표(부모, 루트)를 찾는 함수
  - 합집합(Union) : 하나의 대표로 합해 주는 함수.

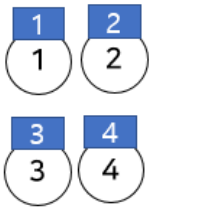


각 노드의 대표를 자기 자신으로 초기화
Find(1) → 대표=1, Find(2) → 대표=2, Find(3) → 대표=3

Union(1, 2)→ 대표 = 2
Find(1) → 대표=2, Find(2)→ 대표=2, Find(3) → 대표=3

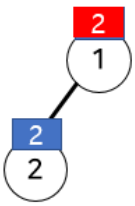
※ '대표' 라는 용어 대신 부모(parent), 루트(root) 라고도 함

- 합집합 - 찾기(Find-Union) 알고리즘의 구성
  ① 대표 노드 번호를 자신의 노드 번호로 초기화
  ② 파인드(Find) : 하나의 노드가 어떤 대표 노드에 속하는 지 확인
  (노드 번호와 대표 번호가 같지 않은 경우 계속해서 대표 번호를 찾음)
  ③ 유니온(Union) : 서로 다른 대표 노드에 속하는 두 노드를 하나의 대표 노드에 속하도록 병합



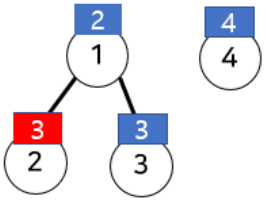
노드	1	2	3	4
대표	1	2	3	4

union(1, 2)



노드	1	2	3	4
대표	2	2	3	4

union(1, 3)



노드	1	2	3	4
대표	2	3	3	4

➔ union(1, 2) 실행 후 노드 1 과 2 는 대표가 같으므로 같은 집합에 속함
➔ union(1, 3) 실행 후 노드 2 의 대표가 3 으로 변경. 1 의 대표는 2, 2 와 3 의 대표는 3 이므로 결론적으로 노드 1, 2, 3 은 같은 집합에 속함

- 합집합-찾기(union-find) 를 JAVA 코드로 구현

```
private static int Find(ArrayList<Integer> parent, int u) {
    if (u == parent.get(u))
        return u;
    else
        return Find(parent,parent.get(u));
}
```

❖ Find() 함수

- 노드와 노드의 대표 값이 같으면 노드를 리턴
- 노드와 노드의 대표 값이 같지 않으면 노드의 대표 값에 대한 대표 값을 찾기 위해 Find() 함수를 재귀 호출

```
private static void Union(ArrayList<Integer> parent, int u, int v) {
    int u_parent = Find(parent, u);
    int v_parent = Find(parent, v);

    parent.set(u_parent, v_parent);
}
```

❖ Union() 함수

- u, v 노드의 대표 번호 찾기
- u 노드의 대표 번호를 v 대표 번호로 변경

```
private static void parentPrint(ArrayList<Integer> p) {
    for(int i=1; i <= 4; i++)
        System.out.print(p.get(i)+ " ");
    System.out.println();
}
```

❖ parentPrint() 함수

- 데이터 출력

```
parentNum.add(0);

//대표 노드 번호를 자신의 노드 번호로 초기화
for(int i = 1; i <= 4 ; i++)
    parentNum.add(i);

//union[1,2] 수행
Union(parentNum,1,2);
parentPrint(parentNum);
```

```
2 2 3 4
find(1): 2
```

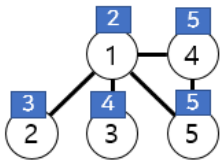
```
//Find(1) 확인해 보기
System.out.println("find(1): "+ Find(parentNum,1));

//union[1,3] 수행
Union(parentNum,1,3);
parentPrint(parentNum);
```

```
2 3 3 4
find(1): 3
```

```
//Find(1) 확인해 보기
System.out.println("find(1): "+ Find(parentNum,1));
```

❖ main 에서 Union 한 결과 확인해 보기

- 합집합-찾기(Union-Find)의 문제점



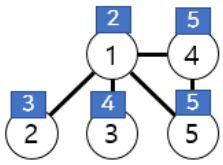
노드 수 = 5
연결 순서 = [[1, 2], [1, 3], [1, 4], [4, 5], [1, 5]]

노드	0	1	2	3	4	5
대표	0	2	3	4	5	5

➔ 대표를 찾는 과정에서 재귀 호출로 인한 시간 낭비 발생

```
private static int Find(ArrayList<Integer> parent, int u) {
    if (u == parent.get(u))
        return u;
    else
        return Find(parent,parent.get(u));
}
```

- 합집합-찾기(Union-Find) 경로 압축



노드 수 = 5
연결 순서 = [[1, 2], [1, 3], [1, 4], [4, 5], [1, 5]]

노드	0	1	2	3	4	5
대표	0	5	3	5	5	5

➔ 대표 값을 찾기 위한 순회 중 방문한 각 원소들이 직접 대표 원소를 가리키도록 갱신

```
private static int FindComp(ArrayList<Integer> parent, int u) {
    if (u == parent.get(u))
        return u;
    else
        parentNum.set(u, FindComp(parent,parent.get(u)));
        return parentNum.get(u);
}
```

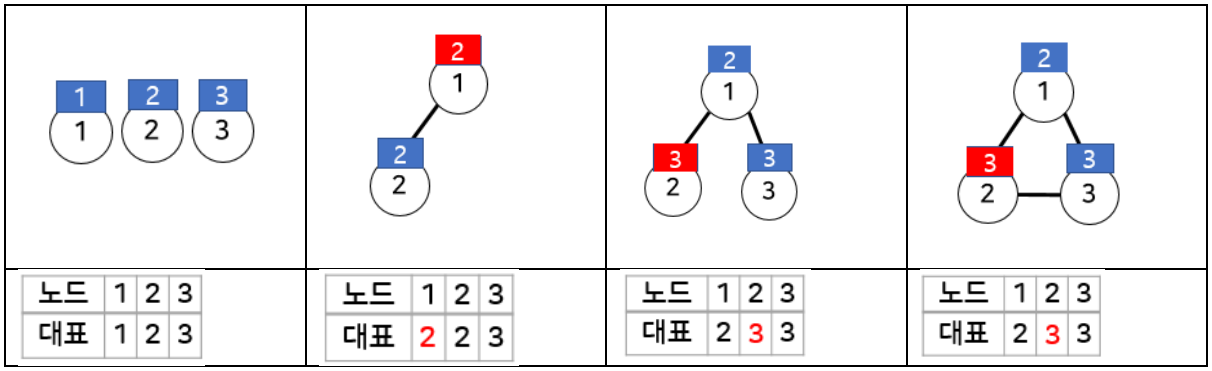
〈 결과 〉

```
각 원소의 대표(부모)값: [0, 5, 3, 5, 5, 5]
각 원소가 속한 집합 확인하기:5 5 5 5 5
```

5) 그래프 사이클 연결 확인

노드 수 = 3
연결 순서 = [[1, 2], [1, 3], [2, 3]]

	1 번째 연결 [1,2]	2 번째 연결 [1, 3]	3 번째 연결 [2, 3]



➔ 연결하려는 노드의 대표(루트, 부모) 노드가 같을 때 사이클 생성

6) 합집합-찾기를 활용하여 사이클 찾기

"연결하려는 노드의 대표(루트,부모)가 일치하면 사이클이 발생한 것"

① 대표 배열을 생성하고 초기화 : 1 번 노드부터 n 번 노드까지의 대표 노드를 자기 자신으로 갖도록 초기화
② 연결하려는 두 노드에 사이클이 발생한 번호를 리턴
③ 연결하려는 두 노드의 대표 찾기(Find) 경로 압축 사용
④ 4-1) 연결하려는 두 노드의 대표가 같으면 cycle 이 발생한 것임
4-2) 대표가 다르면, 더 큰 번호를 대표 번호로 설정(Union)

7) 정답

- 주요 아이디어 정리

✓ 합집합-찾기(Union-Find) 알고리즘의 find 와 union 연산 과정대로 작성된 프로그램의 빈 칸을 완성.

- 정답 코드

```
public int solution(int n, int[][] connections) {
    int answer = 0;

    int[] parent = new int[n+1];
    for(int i = 1; i <= n; i++)
        parent[i] = i;

    for(int i = 0; i < connections.length; i++)
        if(merge(parent, connections[i][0], connections[i][1])) {
            answer = i + 1;
            break;
        }

    return answer;
}
```

❖ solution( ) : 사이클이 생성되는 연결 횟수를 리턴

① 1 번 노드부터 n 번 노드까지의 대표 노드를 자기 자신으로 갖도록 초기화
② 인덱스와 연결할 노드 배열을 for 문을 이용하여 i, connections 에 받음
두 노드를 연결한 후 사이클이 발생하면 해당 인덱스+1 을 answer 에 할당

```
3 public int find(int[] parent, int u) {
      if(u == parent[u])
          return u;

      parent[u] = find(parent, parent[u]);
      return parent[u];
  }
  public boolean merge(int[] parent, int u, int v) {
3     u = find(parent, u);
      v = find(parent, v);

4-1   if(u == v)
          return true;

4-2   parent[u] = v;
      return false;
  }
```

❖ find( ) 함수 : 매개변수 u 로 전달된 노드의 대표 노드를 찾는 함수
③. u 와 u 의 대표 노드가 같으면 노드를 리턴
u 와 u 의 대표 노드가 다르면 재귀 호출을 이용하여 u 의 대표 노드의 대표 노드를 찾음

❖ merge( ) 함수 : u 와 v 가 나타내는 노드의 대표 노드를 찾고 두 노드의 대표 노드가 같으면 true 를 return 하고, 두 노드의 대표 노드가 다르면 u 의 대표 노드를 v 로 변경한 뒤에 false 를 return
④. 4-1) u 의 대표 노드와 v 의 대표 노드가 같으면 u 와 v 를 연결할 경우 사이클이 생성되므로 true 를 return
4-2) u 의 대표 노드와 v 의 대표 노드가 다르면 u 와 v 를 연결해도 사이클이 생성되지 않으므로 false 를 return 하고, u 와 v 가 연결되기 때문에 u 의 대표 노드를 v 로 변경

8. 문제 8

1) 문제 코드

```
/*=====================================
   5차 8번   5차 1급 8_initial_code.java
 =====================================*/
class Solution {
    public int func_a(int a, int b) {
        int mod = a % b;

        while(mod > 0) {
            a = b;
            b = mod;
            mod = a % b;
        }
        return b;
    }

    public int func_b(int n) {
        int answer = 0;

        for(int i = 1; i <= n; i++) {
            if(func_@@@(@@@))
                answer++;
        }
        return answer;
    }

    public boolean func_c(int p, int q) {
        if(p % q == 0)
            return true;
        else
            return false;
    }

    public int solution(int a, int b, int c) {
        int answer = 0;
        int gcd = func_@@@(func_@@@(@@@)@@@);
        answer = func_@@@(@@@);
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int a = 24;
        int b = 9;
        int c = 15;
        int ret = sol.solution(a, b, c);
        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

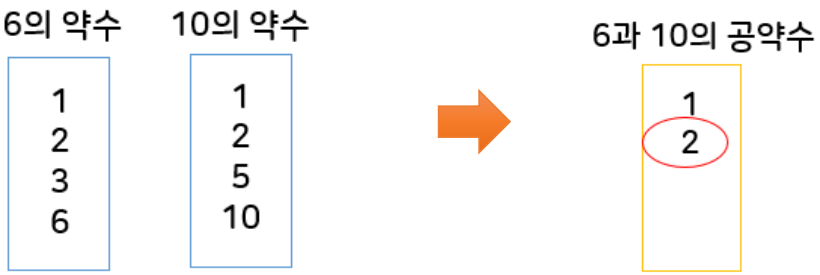
- 문제 코드 안에 작성된 함수를 파악한 후 제시된 과제를 해결하기 위한 알고리즘대로 알맞은 함수를 호출하도록 코드를 완성하는 문제

- 유클리드 호제법을 이용하여 세 수의 최대공약수를 구한 후 산출된 최대공약수의 약수의 개수를 집계하여 return 하기 위해 알맞은 함수와 인수를 적는 문제

3) 24, 9, 15 의 공약수 개수 구하기

24 의 약수	1, 2, 3, 4, 6, 8, 12, 24	→ 세 수의 공약수= 1, 3
9 의 약수	1, 3, 9	→ 공약수의 개수= 2 개
15 의 약수	1, 3, 5, 15	→ 최대공약수 = 3

4) 공약수, 최대공약수



- n 의 약수 : 자기 자신을 1 부터 자기 자신 수(n)까지의 수를 가져와 나누었을 때 나누어 떨어지는 수
- 두 수의 공약수는 두 수 중 작은 수를 넘을 수 없음(ex: 6 과 10 의 공약수는 6 을 넘을 수 없음)
- 최대공약수 구하기
  - 방법 1 : 1 ~ 주어진 수 중 작은 수까지의 모든 수를 가져와 구한 공약수 중 가장 큰 값을 리턴

```
public class gcdCode1 {
    int solution(int num1, int num2) {
        int answer=0;
        int start = Math.min(num1, num2);
        //int start=(num1<num2)?num1:num2;
        for(int i= start;i>=1;i--) {
            if(num1%i==0 && num2%i==0) {
                answer=i;
                break;
            }
        }
        return answer;
    }
```

  - 방법 2 : 유클리드 호제법 이용하기
    - 두 자연수의 최대공약수를 구하는 알고리즘.
    - 두 자연수 a, b 에 대해서 a 가 b 보다 큰 경우 a 를 b 로 나눈 나머지를 r 이라고 할 때 a 와 b 의 최대공약수는 b 와 r 의 최대공약수와 같다는 특징을 이용
    - a 를 b 로 나눈 나머지 r 을 구하고, b 를 r 로 나눈 나머지 r1 을 구하고, 다시 r 을 r1 로 나눈 나머지를 구하는 연산을 반복하여 나머지가 0 이 되게 하는 나누는 수가 최대공약수.
    - 큰 수들의 최대공약수를 구할 때 많이 사용.

- **예) 72 와 27 의 최대공약수 구하기**

a(큰 수)	b(작은 수)	r(a%b)
72	27	18
27	18	9
18	9	0

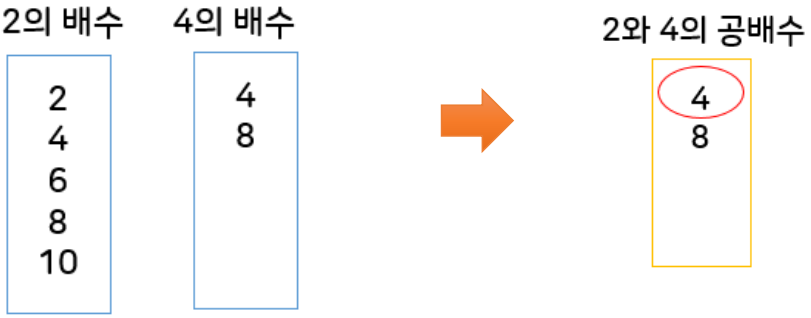
∴ 72 와 27 의 최대공약수 = 9

- JAVA 코드로 구현

```
public class gcdCode2 {
    int solution(int num1, int num2) {
        int a=(num1>num2)?num1:num2;
        int b=(num1<num2)?num1:num2;
        int r=a%b;
        while(r!=0) {
            a = b;
            b = r;
            r = a%b;
        }
        return b;
    }
```

- 유클리드 호제법 : a % b = r (a > b) 일 때 a와 b의 최대공약수 = b와 r의 최대공약수인 성질을 이용해서 b%r=r' 을 구했을 때 r'=0 이면 r 은 a 와 b 의 최대공약수라는 법칙

5) 공배수, 최소공배수



- n 의 배수 : n 으로 나눈 나머지가 0
- 두 수의 최소공배수는 두 수의 곱 이하의 수
- 최소공배수 구하기
  - 방법 1 : num1 에 1 부터 num2 까지의 수를 차례로 곱한 값을 num2 로 나누었을 때 나머지가 없는 수 중 최솟값을 리턴

```java
public class lcmCode1 {
    int solution(int num1, int num2) {
        int answer=0;
        for(int i=1; i<= num2; i++) {
            if((num1*i)%num2 == 0) {
                answer = num1 * i;
                break;
            }
        }
        return answer;
    }
```

- 방법 2 : num1 과 num2 중 큰 수부터 num1*num2 까지의 수 중 num1 과 num2 를 나누었을 때 나머지가 없는 수 중 최솟값을 리턴

```java
public class lcmCode2 {
    int solution(int num1, int num2) {
        int answer=0;
        int start = (num1 > num2)?num1:num2;
        for(int i=start; i<= num1*num2; i++) {
            if(i%num1 == 0 && i %num2==0) {
                answer = i;
                break;
            }
        }
        return answer;
    }
```

6) 정답

```java
public int func_a(int a, int b) {
    int mod = a % b;

    while(mod > 0) {
        a = b;
        b = mod;
        mod = a % b;
    }

    return b;
}
```

❖ func_a( ) 함수 : 유클리드 호제법을 이용하여 a, b 의 최대공약수를 구하는 함수

- a 를 b 로 나눈 나머지를 구하여 mod 로 저장
- b → a 로, mod → b 로 바꾼 뒤 a 를 b 로 나눈 나머지를 다시 구하여 mod 에 저장하는 명령을 mod 가 0 보다 큰 동안 반복 실행
- mod 값이 0 이 될 때의 b 값을 return

```
public int func_b(int n) {
    int answer = 0;

    for(int i = 1; i <= n; i++) {
        if(func_c(n, i))
            answer++;
    }

    return answer;
}
```

❖ func_b( ) 함수 : 약수의 개수를 구하는 함수

- for 문을 이용하여 1 부터 n 까지의 수를 i 로 가져온 후 함수 func_c( )를 이용하여 i 가 n 의 약수인지 판별하여 약수의 개수를 집계

```
public boolean func_c(int p, int q) {
    if(p % q == 0)
        return true;
    else
        return false;
}
```

❖ func_c( ) 함수 : q 가 p 의 약수인지 판별하여 그 결과를 return 하는 함수

- p 를 q 로 나누었을 때 나눈 나머지가 0 이면 q 는 p 의 약수이므로 true 를 return, 그렇지 않으면 false 를 return

```
public int solution(int a, int b, int c) {
    int answer = 0;

    int gcd = func_a(func_a(a, b), c);
    answer = func_b(gcd);

    return answer;
}
```

❖ solution( ) : 세 수의 공약수의 개수를 리턴

- func_a( ) 를 이용하여 세 수 a, b, c 의 최대공약수를 구해서 변수 gcd 에 저장
- 함수 func_b( ) 를 이용하여 최대공약수 gcd 의 약수의 개수를 구하여 answer 에 저장

9. 문제 9

1) 문제 코드

```java
/*=====================================
   5차 9번   5차 1급 9_initial_code.java
 =====================================*/

// 다음과 같이 import를 사용할 수 있습니다.
import java.util.*;

class Solution {
    public int solution(int number, int target) {
        // 여기에 코드를 작성해주세요.
        int answer = 0;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int number1 = 5;
        int target1 = 9;
        int ret1 = sol.solution(number1, target1);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret1 + " 입니다.");

        int number2 = 3;
        int target2 = 11;
        int ret2 = sol.solution(number2, target2);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret2 + " 입니다.");
    }
}
```

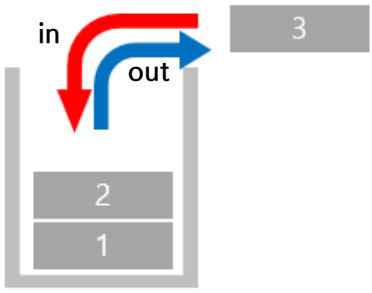
2) 문제 개요

- 제시된 과제를 해결하기 위해 프로그램을 작성하는 문제
- number 가 target 이 될 때까지 곱하기 2, 더하기 1, 빼기 1 연산 중 최소 연산 수행 횟수를 구하는 프로그램을 작성해야 함
- number 에 곱하기 2, 더하기 1, 빼기 1 연산을 한 결과를 차례로 큐에 저장
- 큐에 저장된 결과 값을 먼저 저장된 순서대로 하나씩 가져와 다시 곱하기 2, 더하기 1, 빼기 1 연산을 수행 → 결과를 다시 큐에 저장 → 큐에 결과값 꺼내는 과정을 큐에서 가져오는 결과값이 target 에 도달할 때까지 반복 수행

3) 스택(Stack) 자료구조

① 스택(Stack) 자료구조 소개

- LIFO(Last In First Out: **후입선출** (마지막으로 들어온 데이터가 가장 먼저 나감) 방식으로 자료를 관리하는 구조
- top : 스택에 들어 있는 데이터 중 가장 위(뒤)
- append( ) / put( ) : 마지막(top) 위(뒤)에 데이터 저장
- pop( ) / get( ) : 마지막(top) 데이터를 꺼냄



② Stack 구현하는 방법

- 제네릭 Stack 사용 : <Integer> 타입으로 Stack 을 생성

```
Stack<Integer> st = new Stack<Integer>();

st.push(1);
st.push(2);
st.push(3);

while(!st.empty())
    System.out.println(st.pop());
```

- Stack 객체를 생성
- stack 뒤에 1 을 추가
- stack 뒤에 2 을 추가
- stack 뒤에 3 을 추가
- stack 의 마지막 항목을 꺼내서 출력

< 결과 >

```
3
2
1
```

③ Stack 에서 주로 사용하는 메소드 명

메소드 명	기능
Boolean empty()	스택이 비어 있는지 알려줌
E peek()	삭제 없이 맨 앞(위)의 요소를 읽어 옴
E pop()	스택에서 맨 위(top)의 객체를 꺼내 반환(데이터 삭제)
E push(E item)	마지막(top) 위(뒤)에 데이터 저장
int search(Object o)	스택에서 주어진 객체 o 를 찾아서 그 위치를 반환 (없으면 -1, 시작 위치 1)

4) 큐(Queue) 자료 구조

① 큐(Queue) 자료 구조 소개

- FIFO(First In First Out: **선입선출** (먼저 들어온 데이터가 가장 먼저 나감) 방식으로 자료를 관리하는 구조

② 큐를 구현하는 방법

- 제네릭 Queue 사용 : <Integer> 타입으로 Queue 를 생성

```
Queue<Integer> q = new LinkedList<>();

q.offer(1);
q.offer(2);
q.offer(3);

while(!q.isEmpty())
    System.out.println(q.poll());
```

- Queue 는 보통 LinkedList 를 이용하여 생성
- Queue 뒤에 1 을 추가
- Queue 뒤에 2 을 추가
- Queue 뒤에 3 을 추가
- Queue 의 항목들을 앞에 있는 순서대로 꺼내서 출력

< 결과 >

```
1
2
3
```

③ Queue 에서 주로 사용하는 메소드 명

메소드 명	기능
Boolean add(E e)	지정된 객체를 큐에 추가 (저장 공간 부족 시 IllegalStateException 발생)
E element()	삭제 없이 요소를 읽어 옴
Boolean offer(E e)	큐에 객체를 저장
E peek()	삭제 없이 맨 앞(위)의 요소 읽어 옴
E poll()	큐에서 객체를 꺼내서 반환(큐에서 데이터 삭제)

5) 정답

- 주요 아이디어 정리

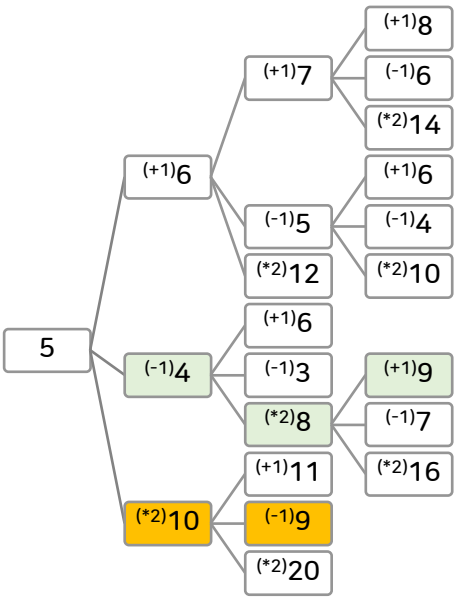
✓ 시작하는 수 number 에 문제에서 주어진 세 가지 연산에 대한 값을 산출

✓ 위에서 구한 각각의 값들에 대해서 다시 세 가지 연산을 수행

- 이미 구했던 값이 나오면 skip
- 각 값이 몇 번의 연산을 통해 산출되었는 지 저장하는 공간이 요구됨

✓ 목표값(target)이 나오면 종료하여 연산 횟수를 구함

ex) 5 에서 시작하여 세 가지 연산을 통해 9 를 구하기



	연산 횟수
5 * 2 - 1 = 9	2 (최소 연산 횟수)
(5 - 1) * 2 + 1 = 9	3

✓ 0 ~ 10000 인덱스를 갖는 배열을 생성하여 0 으로 값을 초기화하고, 산출된 적이 있는 수를 인덱스로 하는 항목에 대해 값을 연산횟수로 변경하여 다음 번 연산 시 재 산출되지 않도록 함

visited 변수 : 각 숫자가 나오기 위해 연산한 횟수 (횟수 +1) 를 저장

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	3	2	1	2	3	3	3	2	3	3	0	0	0	0	0	0	0	3

✓ 큐를 사용하여 다음 번 계산에 사용할 숫자를 순서대로 저장해서 연산을 수행.

q : queue에 다음 계산할 숫자를 저장 5	- 시작하는 수가 5 인 경우 5 를 큐에 넣고 연산 직전에 큐에서 5 를 가져온 후 세 가지 연산 수행
q : queue에 다음 계산할 숫자를 저장 6 4 10	- 5 를 이용한 연산 결과를 큐에 저장하고, 해당 결과들은 연산 실행 직전에 큐에서 순서대로 가져와서 연산 수행

● 정답 코드

```
public int solution(int number, int target) {
    int answer = 0;
①   int[] visited = new int[10001];
    Queue<Integer> q = new LinkedList<Integer>();
②   q.offer(number);
    visited[number] = 1;

③   while(!q.isEmpty()) {
        int x = q.poll();

④       if(x == target)
            break;

        if(x+1 <= 10000 && visited[x+1] == 0) {
            visited[x+1] = visited[x]+1;
            q.offer(x+1);
        }
        if(x-1 >= 0 && visited[x-1] == 0) {
⑤           visited[x-1] = visited[x]+1;
            q.offer(x-1);
        }
        if(2*x <= 10000 && visited[2*x] == 0) {
            visited[2*x] = visited[x]+1;
            q.offer(2*x);
        }
    }
⑥   answer = visited[target]-1;
    return answer;
}
```

① 10000 개의 0 을 항목으로 갖는 배열 visited 를 생성

② 선입선출 방식을 갖는 Queue 객체를 생성하고 인수로 전달된 시작 값을 Queue 객체에 입력 후에 visited 배열에서 시작 값과 동일한 인덱스의 항목 값을 1 로 설정. 예를 들어 시작 값이 5 인 경우 visited 배열의 number 인덱스 값을 방문 횟수를 의미하는 1 을 저장

③ 큐에 있는 값들 중 가장 먼저 입력된 값을 poll()을 이용해 꺼내서 x 에 할당

④ 큐에서 가져온 값이 목표 값과 같으면 반복문을 강제 종료

⑤ (큐에서 가져온 값 + 1), (큐에서 가져온 값 - 1), (큐에서 가져온 값 * 2) 연산을 수행한 값을 인덱스로 갖는 visited 배열의 항목 값이 0 이고, 연산을 수행한 결과값이 0 이상 10000 이하이면, (큐에서 가져온 값 + 1), (큐에서 가져온 값 - 1), (큐에서 가져온 값 * 2) 연산의 결과 값을 인덱스로 갖는 visited 배열의 항목 값을 visited[큐에서 가져온 값] + 1 로 저장하고, 세 개의 연산 결과 값을 큐의 뒤쪽에 차례로 추가

예) 초기값 5 에서 시작하여 목표 값 7 을 만들기 위해 필요한 최소 연산 횟수 구하기

- 큐에 저장된 값의 변화

- 연산 절차

x q.poll()	x+1	v[x+1] =v[x]+1	offer (x+1)	x-1	v[x-1] =v[x]+1	offer (x-1)	x*2	v[x*2] =v[x]+1	offer (x*2)
5	6	v[6]=v[5]+1	6	4	v[4]=v[5]+1	4	10	v[10]=v[5]+1	10
6	7	v[7]=v[6]+1	7	5			12	v[12]=v[6]+1	12
4	5			3	v[3]=v[4]+1	3	8	v[8]=v[4]+1	8
10	11	v[11]=v[10]+1	11	9	v[9]=v[10]+1	9			
7									

- visited 배열의 항목 값 변화

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	1	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	1	2	3	0	0	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	1	2	3	0	0	2	0	3	0	0	0	0	0	0	0	0
0	0	0	3	2	1	2	3	0	0	2	0	3	0	0	0	0	0	0	0	0
0	0	0	3	2	1	2	3	3	0	2	0	3	0	0	0	0	0	0	0	0
0	0	0	3	2	1	2	3	3	0	2	3	3	0	0	0	0	0	0	0	0
0	0	0	3	2	1	2	3	3	3	2	3	3	0	0	0	0	0	0	0	0
0	0	0	3	2	1	2	3	3	3	2	3	3	0	0	0	0	0	0	0	3

→ 마지막 offer( ) 에 의해 나온 7 이 목표 값과 동일하므로 반복문은 강제 종료되고, 초기 값 5 에서 목표 값 7 을 만들기 위해 필요한 최소 연산 횟수는 visited[7] - 1 를 계산한 결과인 2 로 산출됨

⑥. 시작값에서 시작하여 목표 값을 만들기 위해 필요한 연산 횟수 + 1 의 값이 visited 배열에 있으므로 리턴하는 값은 (visited 배열의 항목 값 - 1) 로 지정

10. 문제 10

1) 문제 코드

```
/*=======================================
   5차 10번   5차 1급 10_initial_code.java
 =======================================*/
class Solution {
    class Job {
        public int salary;

        public Job() {
            this.salary = 0;
        }
        public int getSalary() {
            return salary;
        }
    }

    class PartTimeJob @@@ {
        public int workHour, payPerHour;

        public PartTimeJob(int workHour, int payPerHour) {
            this.workHour = workHour;
            this.payPerHour = payPerHour;
        }
        @@@ {
            salary = workHour * payPerHour;
            if(workHour >= 40)
                salary += (payPerHour * 8);

            return salary;
        }
    }

    class SalesJob @@@ {
        public int salesResult, payPerSale;

        public SalesJob(int salesResult, int payPerSale) {
            this.salesResult = salesResult;
            this.payPerSale = payPerSale;
        }
        @@@ {
            if(salesResult > 20)
                salary = salesResult * payPerSale * 3;
            else if(salesResult > 10)
                salary = salesResult * payPerSale * 2;
            else
                salary = salesResult * payPerSale;

            return salary;
        }
    }
```

```
    public int solution(int[][] partTimeJobs, int[][] salesJobs) {
        int answer = 0;

        for(int i = 0; i < partTimeJobs.length; i++) {
            PartTimeJob partTimeJob = new PartTimeJob(partTimeJobs[i][0], partTimeJobs[i][1]);
            answer += partTimeJob.getSalary();
        }

        for(int i = 0; i < salesJobs.length; i++) {
            SalesJob salesJob = new SalesJob(salesJobs[i][0], salesJobs[i][1]);
            answer += salesJob.getSalary();
        }

        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] partTimeJobs = {{10, 5000}, {43, 6800}, {5, 12800}};
        int[][] salesJobs = {{3, 18000}, {12, 8500}};
        int ret = sol.solution(partTimeJobs, salesJobs);

        // [실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
        System.out.println("solution 메소드의 반환 값은 " + ret + " 입니다.");
    }
}
```

2) 문제 개요

- Job 클래스를 상속받도록 PartTimeJob 클래스와 SalesJob 클래스를 정의하는 문제
- PartTimeJob 클래스와 SalesJob 클래스의 부모 클래스를 지정하고, 각각의 자식 클래스에서 부모 클래스인 Job 의 getSalary( ) 메소드를 재정의하여 문제에서 제시된 대로 급여를 계산하도록 메소드 정의 부분의 빈 칸을 채워야 함

3) 정답

```
class Job {
    public int salary;

    public Job() {
        this.salary = 0;
    }

    public int getSalary() {
        return salary;
    }
}
```

❖ Job 클래스 정의

① 생성자 메소드를 이용하여 멤버 변수 salary 를 생성하고 0 으로 초기화

② getSalary( ) 메소드는 멤버 변수 salary 에 저장되어 있는 값을 리턴

```
class PartTimeJob extends Job {
    public int workHour, payPerHour;

    public PartTimeJob(int workHour, int payPerHour) {
        this.workHour = workHour;
        this.payPerHour = payPerHour;
    }

    public int getSalary() {
        salary = workHour * payPerHour;
        if(workHour >= 40)
            salary += (payPerHour * 8);

        return salary;
    }
}
```

❖ PartTimeJob 클래스 정의

③. PartTimeJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정

④. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화

⑤. 매개 변수로 받은 값을 이용하여 멤버 변수 workHour 와 멤버 변수 payPerHour 를 생성하고 초기화

⑥. 부모 클래스 Job 에 있는 getSalary( ) 메소드를 오버라이드

⑦. 멤버 변수 workHour * payPerHour 를 계산한 값을 멤버 변수 salary 로 저장

⑧. 멤버 변수 workHour >= 40 이면 멤버 변수 payPerHour * 8 을 계산한 것을 멤버 변수 salary 에 추가로 더함

```
class SalesJob extends Job {
    public int salesResult, payPerSale;

    public SalesJob(int salesResult, int payPerSale) {
        this.salesResult = salesResult;
        this.payPerSale = payPerSale;
    }

    public int getSalary() {
        if(salesResult > 20)
            salary = salesResult * payPerSale * 3;
        else if(salesResult > 10)
            salary = salesResult * payPerSale * 2;
        else
            salary = salesResult * payPerSale;

        return salary;
    }
}
```

❖ SalesJob 클래스 정의

⑨. SalesJob 클래스가 Job 클래스를 상속받도록 부모 클래스를 Job 으로 지정

⑩. 부모 클래스의 생성자 메소드를 실행하여 멤버 변수 salary 를 생성하고 0 으로 초기화
⑪. 매개 변수로 받은 값을 이용하여 멤버 변수 salesResult 와 멤버 변수 payPerSale 를 생성하고 초기화
⑫. 부모 클래스 Job 에 있는 getSalary( ) 메소드를 오버라이드