Professional Coding Specialist

COS Pro JAVA 1 급

11 강-15 강. 모의고사 2 차

1. 모의고사 2 차(1-10 번)

과정 소개

COS Pro JAVA 1 급의 모의고사 2 차를 풀어보며 문제 유형을 익히고, JAVA 를 이용하여 알고리즘을 구현하기 위해 필요한 관련 지식을 익혀보도록 한다.

학습 목차

학습 목표

1. YBM IT(www.ybmit.com) 에서 제공하는 COS Pro 1 급 JAVA 샘플 문제를 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

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

//Book 인터페이스와 ComicBook, Novel 클래스는 Inner Class로
class Solution {
  interface Book{
      public int getRentalPrice(int day);
  }

  class ComicBook @@@ {
      @@@{
          int cost = 500;
          day -= 2;
          if(day > 0)
              cost += @@@;
          return cost;
      }
  }

  class Novel @@@ {
      @@@{
          int cost = 1000;
          day -= 3;
          if(day > 0)
              cost += @@@;
          return cost;
      }
  }

  public int solution(String[] bookTypes, int day) {
      Book[] books = new Book[50];
      int length = bookTypes.length;
      for(int i = 0; i < length; ++i){
          if(bookTypes[i].equals("comic"))
              books[i] = new ComicBook();
          else if(bookTypes[i].equals("novel"))
              books[i] = new Novel();
      }
      int totalPrice = 0;
      for(int i = 0; i < length; ++i)
          totalPrice += books[i].getRentalPrice(day);
      return totalPrice;
  }
```

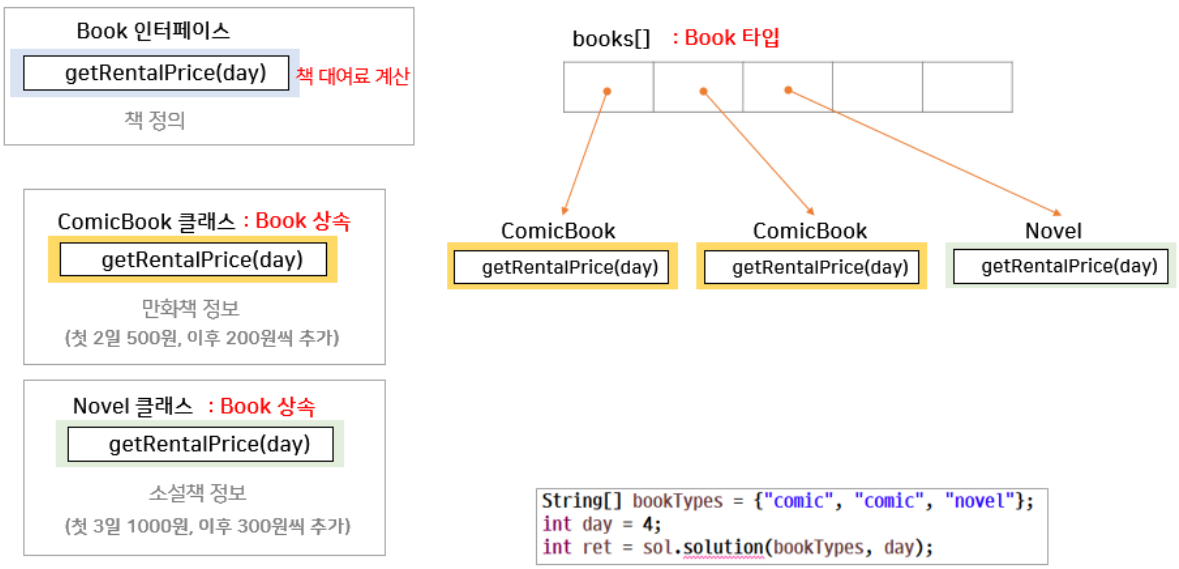
※ 문제 출처 : https://www.ybmit.com/cos_pro/cos_pro_r_test.jsp

```
// 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
public static void main(String[] args) {
    Solution sol = new Solution();
    String[] bookTypes = {"comic", "comic", "novel"};
    int day = 4;
    int ret = sol.solution(bookTypes, day);

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

2) 문제 개요

- 인터페이스 Book 을 상속받는 ComicBook 클래스와 Novel 클래스를 정의하는 문제
- 인터페이스 Book 에 있는 추상 메소드인 getRentalPrice( )를 자식 클래스인 ComicBook 클래스와 Novel 클래스에서 각 클래스의 특징에 맞게 정의해야 함
- ComicBook 클래스의 getRentalPrice( ) 메소드에서는 기본 요금 500 원에 대여 기간이 2 일을 초과한 일자 당 200 원의 추가 요금을 계산하도록 정의해야 하고, Novel 클래스의 getRentalPrice( ) 메소드에서는 기본 요금 1000 원에 대여 기간이 3 일을 초과된 일자 당 300 원의 추가 요금을 계산하도록 정의해야 함

3) 전체 구조도



4) 정답

```
1  interface Book{
       public int getRentalPrice(int day);
   }

2  class ComicBook @@@ {                    implements Book
       @@@{
           int cost = 500;                  public int getRentalPrice(int day)
           day -= 2;
           if(day > 0)
               cost += @@@;                 day*200
           return cost;
       }
   }

3  class Novel @@@ {                        implements Book
       @@@{
           int cost = 1000;                 public int getRentalPrice(int day)
           day -= 3;
           if(day > 0)
               cost += @@@;                 day*300
           return cost;
       }
   }
```

① Book 을 인터페이스로 정의 : getRentalPrice() 추상 메소드 정의

② ComicBook 클래스에서 Book 인터페이스를 상속 받도록 implements
Book 인터페이스의 추상 메소드인 getRentalPrice( ) 메소드를 ComicBook 클래스 안에서 재정의

- 매개 변수로 전달받은 값 day 에서 '기본 대여일'로 지정된 2 일을 빼도 day 값이 0 보다 크다면 '기본 대여일'을 초과한 날짜만큼 대여 금액을 계산해야 함
- '기본 대여일'을 초과한 일수에 200 원을 곱한 값을 추가로 더하여 대여 요금을 계산하도록 계산식을 입력

③ Novel 클래스에서 Book 인터페이스를 상속 받도록 implements
Book 클래스의 추상 메소드인 getRentalPrice( ) 메소드를 Novel 클래스 안에서 재정의

- 매개 변수로 전달받은 값 day 에서 '기본 대여일'로 지정된 3 일을 빼도 day 값이 0 보다 크다면 '기본 대여일'을 초과한 날짜만큼 대여 금액을 계산해야 함
- '기본 대여일'을 초과한 일수에 300 원을 곱한 값을 추가로 더하여 대여 요금을 계산하도록 계산식을 입력

2. 문제 2

1) 문제 코드

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

class Solution {
    public int func_a(String times){
        int hour = Integer.parseInt(times.substring(0, 2));
        int minute = Integer.parseInt(times.substring(3));
        return hour*60 + minute;
    }
    public int solution(String[] subwayTimes, String currentTime) {
        int currentMinute = func_a(@@@);
        int INF = 1000000000;
        int answer = INF;
        for(int i = 0; i < subwayTimes.length; ++i){
            int subwayMinute = func_a(@@@);
            if(@@@){
                answer = subwayMinute - currentMinute;
                break;
            }
        }
        if(answer == INF)
            return -1;
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        String[] subwayTimes1 = {"05:31", "11:59", "13:30", "23:32"};
        String currentTime1 = "12:00";
        int ret1 = sol.solution(subwayTimes1, currentTime1);

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

        String[] subwayTimes2 = {"14:31", "15:31"};
        String currentTime2 = "15:31";
        int ret2 = sol.solution(subwayTimes2, currentTime2);

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

2) 문제 개요

- 프로그램의 알고리즘에 맞도록 func_a( ) 함수에 적절한 인수를 전달하고, 비어 있는 조건식을 완성하는 문제
- 문자열로 구성된 지하철 도착시각 배열을 이용하여 지하철을 타기 위해 기다려야 하는 최소 대기 시간을 구하는 프로그램으로, 현재 시각과 지하철 도착 시각을 00:00 기준으로 하는 분 단위 수로 변환해서 비교

3) 정답

```
public int func_a(String times){
    int hour = Integer.parseInt(times.substring(0, 2));
    int minute = Integer.parseInt(times.substring(3));
    return hour*60 + minute;
}

public int solution(String[] subwayTimes, String currentTime) {
1   int currentMinute = func_a( currentTime
    int INF = 1000000000;
    int answer = INF;
2   for(int i = 0; i < subwayTimes.length; ++i){
        int subwayMinute = func_a( subwayTimes[i]
        if(@@@){   subwayMinute >= currentMinute
            answer = subwayMinute - currentMinute;
            break;
        }
    }
3   if(answer == INF)
        return -1;          (오늘 탈 수 있는 지하철이 없으면 -1 )
    return answer;
}
```

① func_a( ) 함수는 문자열 형태('hh:mm')로 전달 받은 시각 값을 분 단위로 수로 변환시킨 정수 값을 return

〈 문자열로 들어온 시각 값 〉 **〈 00:00 부터 시작해서 분 단위 수로 변환한 값 〉**

0	1	2	3	4
'1'	'3'	':'	'3'	'0'

시간 = times[: 2]를 정수로 변환 * 60
분 = times[3:]를 정수로 변환
→ 두 값을 더해서 return

Times 의 데이터가 문자열이므로 Integer.parseInt() 사용하여 int 로 형 변환

② solution( ) 메소드의 매개변수인 currentTime 에는 현재 시각, subwayTimes 에는 순서대로 나열된 지하철 도착 시각이 배열로 전달

- func_a( ) 함수를 이용하여 currentTime 을 분 단위 수로 변환
- for 문을 이용하여 subwayTimes 에 저장되어 있는 열차 도착 시각을 하나씩 가져와 func_a( ) 함수로 전달하여 분 단위 수로 변환
- 열차 도착 시각이 현재 시각보다 크거나 같으면 (열차 도착 시각 - 현재 시각)을 계산하여 answer 에 저장하고 break 문을 사용하여 for 반복문을 종료

③ 오늘 탈 수 있는 지하철이 없으면 -1 을 return

3. 문제 3

1) 문제 코드

```
/*=======================================
    2차 3번   2차 1급 3_initial_code.java
 =======================================*/
class Solution {
    public int func_a(int n){
        int ret = 1;
        while(n > 0){
            ret *= 10;
            n--;
        }
        return ret;
    }

    int func_b(int n){
        int ret = 0;
        while(n > 0){
            ret++;
            n /= 10;
        }
        return ret;
    }

    int func_c(int n){
        int ret = 0;
        while(n > 0){
            ret += n%10;
            n /= 10;
        }
        return ret;
    }
```

```
    public int solution(int num) {
        int nextNum = num;
        while(true){
            nextNum++;
            int length = func_@@@(@@@);
            if(length % 2 != 0)
                continue;

            int divisor = func_@@@(@@@);
            int front = nextNum / divisor;
            int back = nextNum % divisor;

            int frontSum = func_@@@(@@@);
            int backSum = func_@@@(@@@);
            if(frontSum == backSum)
                break;
        }
        return nextNum - num;
    }
```

※ 문제 출처 : https://www.ybmit.com/cos_pro/cos_pro_r_test.jsp

```
    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int num1 = 1;
        int ret1 = sol.solution(num1);

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

        int num2 = 235386;
        int ret2 = sol.solution(num2);

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

2) 문제 개요

- 문제 코드 안에 작성된 함수의 알고리즘을 파악하여 알맞은 함수와 매개변수를 작성하는 문제

3) 정답

```
public int func_a(int n){
    int ret = 1;
    while(n > 0){
        ret *= 10;
        n--;
    }
    return ret;
}
```
divisor 구하기

```
int func_b(int n){
    int ret = 0;
    while(n > 0){
        ret++;
        n /= 10;
    }
    return ret;
}
```
숫자의 자릿수

```
int func_c(int n){
    int ret = 0;
    while(n > 0){
        ret += n%10;
        n /= 10;
    }
    return ret;
}
```
각 자리 합

```
public int solution(int num) {
    int nextNum = num;
1   while(true){
        nextNum++;
        int length = func_ b(nextNum)
        if(length % 2 != 0)
            continue;
2       int divisor = func_ a(length/2)
        int front = nextNum / divisor;
        int back = nextNum % divisor;

        int frontSum = func_ c(front) ;
        int backSum = func_ c(back) ;
        if(frontSum == backSum)
            break;
    }
3   return nextNum - num;
}
```

- func_a( ) 함수는 매개변수 n 만큼 10 을 제곱하여 return. 예를 들어 매개변수로 전달받은 값이 3 이면 10 의 세제곱수 인 1000 을 리턴
  - 결과값 변수 ret 을 1 로 초기화
  - while 문을 사용하여 n>0 인 동안 n 을 하나씩 빼면서 ret * 10 씩 곱함

- func_b( ) 함수는 전달 받은 수 n 의 자릿수를 return
  - while 문을 사용하여 n>0 인 동안 자릿수를 세는 변수 ret 를 1 만큼 증가
  - n 을 10 으로 나눈 몫으로 n 의 값을 변경

- func_c( ) 함수는 전달 받은 수의 각 자릿수에 있는 숫자를 합산한 결과를 return
  - while 문을 사용하여 n >0 인 동안 n 을 10 으로 나눈 나머지를 ret 에 더함
  - n 을 10 으로 나눈 몫으로 n 의 값을 변경

① while 문을 반복하면서 func_b( ) 함수를 이용하여 '다음 게시글 번호' 의 자릿수를 구함
'다음 게시글 번호'의 자릿수가 짝수가 아니면 다음 번 반복문을 실행하기 위해 continue 문을 사용

② '다음 게시글 번호'의 자릿수를 2 로 나눈 값을 인수로 전달하며 func_a( ) 함수를 호출하여 '다음 게시글 번호' 를 이등분하는 기준값을 divisor 에 저장
'다음 게시글 번호' 를 divisor 로 나눈 몫으로 구하여 front 에 저장 → '다음 게시글 번호'를 2 등분한 것의 앞부분이 front 에 저장됨
'다음 게시글 번호' 를 divisor 로 나눈 나머지를 구하여 back 에 저장 → '다음 게시글 번호'를 2 등분한 것의 뒷부분이 back 에 저장됨
func_c( ) 함수를 사용하여 front 와 back 의 각 자릿수의 합을 구해서 frontSum. backSum 변수에 넣고, 그 합이 서로 같으면 while 반복문을 빠져나감

③ (현재의 '다음 게시글 번호' - 현재 작성되어 있는 마지막 게시글 번호)를 계산한 값을 리턴

4. 문제 4

1) 문제 코드

```
/*=======================================
    2차 4번   2차 1급 4_initial_code.java
 =======================================*/

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

class Solution {
    public int solution(int[] arr, int K) {
        // 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[] arr = {1, 2, 3, 4, 5};
        int K = 3;
        int ret = sol.solution(arr, K);


        // [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
- arr 배열에서 서로 다른 세 수를 선택하여 합한 값이 K 의 배수인 경우의 수를 세는 solution( ) 메소드를 작성하는 문제

3) 정답

```
class Solution {
    public int solution(int[] arr, int K) {
        int n = arr.length;
        int count = 0;
        for(int i = 0; i < n; ++i)
            for(int j = i + 1; j < n; ++j)
                for(int k = j + 1; k < n; ++k)
                    if((arr[i]+arr[j]+arr[k])%K == 0)
                        count += 1;
        return count;
    }
}
```

- 세 개의 중첩 for 문을 사용하여 arr 배열의 항목 중 서로 다른 세 개를 가져와 합산한 결과가 K 로 나누어 떨어지면 결과값을 1 만큼 증가
  - 모든 경우를 탐색하며 값을 찾는 방식을 브루트 포스(Brute-Force) 방식이라고 함

- 세 개의 중첩 for 문을 사용하기 때문에 시간 복잡도는 $O(n^3)$

4) 보충 학습 : 비슷한 유형의 문제 - 합이 K 가 되는 세 숫자 고르기

① 브루트 포스(Brute-Force) 방식 : $O(n^3)$

```
public int solution(int[] arr, int K) {
    // 여기에 코드를 작성해주세요.
    int answer = 0;
    int n=arr.length;

    for (int p=0; p<n-2;p++) {
        for (int q=p+1; q<n-1 ; q++) {
            for (int r=q+1;r<n ; r++ ) {
                if((arr[p]+arr[q]+arr[r]) ==K) {
                    answer+=1;
                }
            }
        }
    }
    return answer;
}
```

- 제시된 문제와 동일하게 세 개의 중첩 for 문을 사용하여 배열에 저장되어 있는 모든 수의 조합이 조건에 만족하는지 확인하며 답을 구함

② 투 포인터(Two Pointer) 방식 : $O(n^2)$

```
Arrays.sort(arr);

for (p=0; p<n-2;p++) {
    left=p+1;
    right=n-1;
    while (left<right) {
        sum = arr[p]+arr[left]+arr[right];
        if (sum < K)
            left+=1;
        else if (sum > K)
            right-=1;
        else {
            answer+=1;
            left+=1;
            right-=1;
        }
    }
}
```

- 배열 arr 의 항목들을 순서대로 정렬하는 작업이 선행되어야 함
- for 문을 이용하여 변수 p 에 배열의 인덱스 0 부터 차례대로 값을 받아오고, left 는 p 가 나타내는 항목의 다음 인덱스 값을, right 에는 배열의 마지막 항목 인덱스를 할당
- left 값이 right 보다 작은 동안

- sum 에 arr[p] + arr[left] + arr[right] 을 할당
- 만일 sum 값이 K 보다 작으면 left 를 1 만큼 증가시켜 현재 가리키는 항목의 바로 다음 항목을 가리키도록 조정
- 만일 sum 값이 K 보다 크면 right 를 1 만큼 감소시켜 현재 가리키는 항목의 바로 앞 항목을 가리키도록 조정
- 그렇지 않다면 K 와 같은 경우이므로 answer 를 1 만큼 증가시키고 arr[p] 를 선택한 상태에서 세 수의 합이 K 와 같아지는 다른 경우가 없는지 탐색하기 위해 left 를 1 만큼 증가시키고 right 를 1 만큼 감소시킴

5. 문제 5

1) 문제 코드

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

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

class Solution {
    public int solution(int[] arr) {
        // 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[] arr = {3, 1, 2, 4, 5, 1, 2, 2, 3, 4};
        int ret = sol.solution(arr);

        // [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
- 매개변수로 전달된 배열에서 연속해서 값이 증가하는 구간 중 가장 긴 구간을 찾아 그 길이를 출력하는 프로그램을 작성

3) 정답

A. 다이나믹 프로그래밍 방식 : O(n)

```
public int solution(int[] arr) {
    int dp[] = new int[arr.length];
    for(int i = 0; i < dp.length; ++i)
        dp[i] = 1;
    for(int i = 1; i < arr.length; ++i)
        if(arr[i] > arr[i-1])
            dp[i] = dp[i-1] + 1;
    int answer = 0;
    for(int i = 0; i < arr.length; ++i)
        answer = Math.max(answer, dp[i]);
    return answer;
}
```

①. arr 와 같은 길이를 갖는 새로운 배열 dp 를 생성하여 각 항목을 1 로 초기화

- 배열 dp 는 arr 의 각 항목별 연속된 증가 구간 값을 저장하기 위해 사용

②. arr 의 현재 항목값이 이전 항목값보다 크면, 이전 항목값과 동일한 인덱스의 dp 항목 값에 +1 한 것을 현재 항목값과 동일한 인덱스의 dp 항목에 저장

③. dp 배열에 있는 값 중 가장 큰 값을 answer 에 저장한 후 return

B. 브루트 포스(Brute-Force) 방식 : O(n^2)

```
int answer=0;
int cnt=1,i,j;

for (i=0; i<arr.length -1;i++) {
    for (j=i+1; j < arr.length ; j++ ) {
        if (arr[j-1] < arr[j])
                cnt+=1;
        else
            break;
    }

    answer=Math.max(answer, cnt);
    cnt=1;

}
```

①. arr 배열에서 한 항목을 가져온 후 가져온 항목 이후에 위치한 항목을 중첩 for 문을 이용하여 가져옴

②. 안쪽 for 문을 이용해 가져온 항목이 이전 항목값보다 크면 증가 구간을 세는 변수 cnt 를 +1 하고, 그렇지 않으면 break 명령문을 이용하여 안쪽 for 문을 빠져나감

③. answer 와 cnt 중 큰 값을 answer 에 할당한 후 cnt 를 1 로 초기화

4) 보충 학습 : 동적 계획법(Dynamic Programming)

① 정의

- 복잡한 문제를 작은 문제로 나누어 해결하는 방법으로 하위 문제의 답을 이용하여 상위 문제의 답을 구하는 방식

- 문제를 분할한다는 점에서 분할 정복법(divide & conquer)와 유사하지만 분할 정복법은 하위 문제의 답이 상위 문제의 답에 영향을 미치지 않음
- 최적화 문제를 해결하는데 효과적인 기법

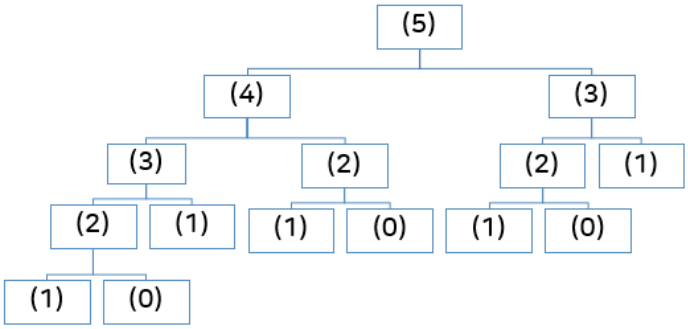
② 적용 사례 : 피보나치 수열을 구하는 문제

- 피보나치 수열이란 첫 번째 항=1, 두 번째 항=1, 세 번째 항부터 그 이전에 있는 두 항을 더한 값으로 구성된 수열

- 재귀 호출 방식으로 구한 피보나치 수열

```
public static int fibo_re(int n) {
    if (n<=2)
        return 1;
    else
        return fibo_re(n-1) + fibo_re(n-2);
}
```

- 상위 문제에서 하위 문제를 재귀적으로 호출
- 같은 계산을 반복 수행하는 경우가 발생함으로 비효율적



- fibo_re(5) : 1 회 호출
- fibo_re(4) : 1 회 호출
- fibo_re(3) : 2 회 호출
- fibo_re(2) : 3 회 호출
- fibo_re(1) : 5 회 호출
- fibo_re(0) : 3 회 호출

- 동적 계획법-메모이제이션(memoization)으로 구한 피보나치 수열

```
public static int dp[]=new int[500];

public static int fibo_dp(int n) {
    if (n<=2)
        return 1;

    if (dp[n]!=0)
        return dp[n];
    else {
        dp[n]=fibo_dp(n-1) + fibo_dp(n-2);
        return dp[n];
    }
}
```

- 결과 저장 공간을 0 으로 초기화
- 계산한 값이 있으면 그 값을 다시 계산하지 않고 반환
- 계산한 값이 없으면 재귀 호출을 통해 계산하고 그 결과를 저장공간에 저장한 후 반환

- 메모이제이션(memoization) : 동일한 계산을 반복하지 않도록 이전 계산 결과를 저장해 놓고 필요한 경우에만 재귀 호출을 사용함으로써 프로그램 실행 속도를 높이는 기술

- 동적 계획법 - 타블레이션으로 구한 피보나치 수열

```
public static int fibo_dp(int n) {
    int dp[]=new int[n+1];
    dp[1]=1;
    dp[2]=1;
    for(int i=3; i<=n;i++)
        dp[i]=dp[i-1]+dp[i-2];
    return dp[n];
}
```

- 결과 저장 공간을 dp 로 정의하고 첫 번째, 두 번째 항목에 대한 값을 지정.
- 피보나치 수열의 항목 값을 얻기 위해 세 번째 항목부터 원하는 순번의 항목까지 결과를 산출해서 저장.

5) 이론 정리 : 브루트 포스, 그리디, 분할 정복법, 동적 계획법

브루트 포스 (Brute-Force)	그리디 (Greedy)	분할 정복법 (Divide&Conquer)	동적 계획법 (Dynamic Programming)	
			메모이제이션 (Memoization)	타블레이션 (Tabulation)
모든 경우를 탐색	큰 문제를 작은 문제로 나누어 해결			
완전 탐색이 필요한 문제	작은 문제의 최적을 선택하면 전체가 최적이 되는 문제	하위 문제가 상위와 종속되지 않는 문제	하위의 결과값이 상위의 결과에 영향을 주는 문제(종속) 하위 문제가 최적 부분 구조를 구성하고 동일한 하위 문제를 반복적으로 해결하여 결과를 저장하는 문제	
시간이 많이 소요	눈 앞의 이익만 고려 최적의 결과를 보장할 수 없음	재귀호출 사용 너무 작게 나누면 복잡해질 수 있음	재귀호출 방식이나 하위 문제 결과를 메모리에 저장한 후 재사용	재귀호출을 하지 않음 첫 번째부터 모든 결과를 저장
예외 없는 정답	모든 방법을 고려하지 않지만 최적인 경우가 존재	큰 문제를 작게 나눠 효과적이나 재귀호출로 인한 시간 증가	최적해를 구함 시간 감소, 공간 증가(브루트 포스에 비해)	
	큰 화폐 단위부터 거스름돈 주기	이진 검색 퀵 정렬	최적화 문제에 적합 : 최적 경로 문제, 최소 비용 문제 (최적화 문제 : 하나의 문제에 답이 여러 개인 경우의 문제)	

6) 다른 코딩 제안

```
int answer = 0;
int cnt=1;
int i=1;

1 while (i<arr.length) {
    if (arr[i] > arr[i-1])
        cnt+=1;
2   else
        cnt=1;
    i+=1;

3   answer=Math.max(cnt, answer);
}
```

① while 문을 이용해서 arr 배열의 항목을 1 번 인덱스부터 가져와 이전 항목과 크기 비교

② 현재 항목이 이전 항목보다 크면 증가 구간의 크기를 나타내는 변수 cnt 에 1 을 더하고, 그렇지 않으면 cnt=1 로 저장

다음 항목 값을 가져오기 위해 i 를 1 만큼 증가시키고, cnt 와 answer 중 큰 값을 answer 에 할당

③. cnt 와 answer 중 큰 값을 answer 에 저장

6. 문제 6

1) 문제 코드

```java
/*=======================================
    2차 6번   2차 1급 6_initial_code.java
 =======================================*/


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

class Solution {
    public int[] solution(String commands) {
        // 여기에 코드를 작성해주세요.
        int[] answer = {};
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        String commands = "URDDL";
        int[] ret = sol.solution(commands);

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

2) 문제 개요

- 제시된 과제를 해결하기 위해 solution( )에 프로그램 코드를 작성하는 문제
- 매개변수 commands 로 전달된 문자열 안의 문자를 하나씩 가져와 문자별 지정된 방향으로 좌표를 이동시킨 후 마지막 좌표의 위치를 출력하는 문제

3) 정답

```java
public int[] solution(String commands) {
    int[] answer = {0, 0};
    for(int i = 0; i < commands.length(); ++i){
        if (commands.charAt(i) == 'L')
            answer[0] -= 1;
        else if(commands.charAt(i) == 'R')
            answer[0] += 1;
        else if(commands.charAt(i) == 'U')
            answer[1] += 1;
        else if(commands.charAt(i) == 'D')
            answer[1] -= 1;
    }
    return answer;
}
```

①. for 문을 이용하여 commands 에 있는 문자를 하나씩 가져옴

②. if 문을 사용하여 가져온 문자가 'L', 'R' 이면 x 좌표에 해당하는 answer[0] 값을 각각 -1, +1 만큼 증가

③. if 문을 사용하여 가져온 문자가 'D', 'U' 이면 y 좌표에 해당하는 answer[1] 값을 각각 -1, +1 만큼 증가

7. 문제 7

1) 문제 코드

```java
/*=======================================
    2차 7번   2차 1급 7_initial_code.java
 =======================================*/

class Solution {
    public int solution(int money) {
        int coin[] = {10, 50, 100, 500, 1000, 5000, 10000, 50000};
        int counter = 0;
        int idx = coin.length - 1;
        while (money > 0){
            counter += @@@;
            money %= @@@;
            idx -= @@@;
        }
        return counter;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다.
    public static void main(String[] args) {
        Solution sol = new Solution();
        int money = 2760;
        int ret = sol.solution(money);

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

2) 문제 개요

- 프로그램의 알고리즘에 맞도록 빈 칸의 구문을 완성하는 문제
- 탐욕 알고리즘(Greedy Algorithm)의 대표적인 사례로 현재 금액에서 사용할 수 있는 최대 단위 화폐부터 시작하여 거슬러줘야 하는 금액에 대한 화폐의 개수를 집계하는 프로그램을 완성해야 함.

3) 정답

```java
public int solution(int money) {
    int coin[] = {10, 50, 100, 500, 1000, 5000, 10000, 50000};
    int counter = 0;
    int idx = coin.length - 1;
    while (money > 0){
        counter += @@@;   // 1
        money %= @@@;     // 2
        idx -= @@@;       // 3
    }
    return counter;
}
```

- coin 배열에 저장되어 있는 마지막 항목(가장 큰 단위 화폐)부터 가져와 매개변수 money 가 0 이 되지 않는 동안 실행
  ①. money 를 coin 배열의 현재 가져온 항목 값으로 나눈 몫을 현재 선택한 단위의 화폐 개수로 집계하여 counter 에 저장
  ②. money 를 coin 배열에서 현재 가져온 항목 값으로 나눈 나머지로 지정 → 현재 사용 중인 단위 화폐를 이용하여 거슬러줄 수 있는 금액을 계산하고, 나머지 금액을 money 로 다시 저장
  ③. 그 다음으로 큰 단위 화폐를 가져올 수 있도록 인덱스를 나타내는 변수 idx 에서 1 만큼 감소

8. 문제 8

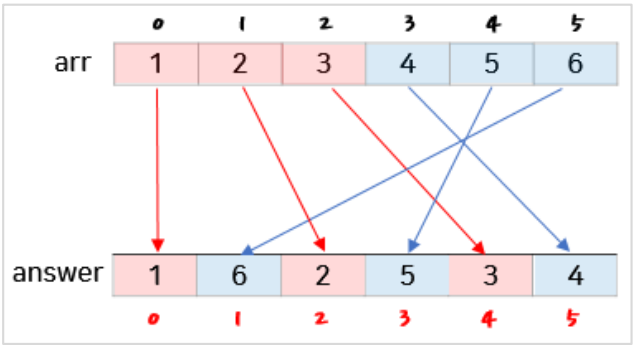
1) 문제 코드

```java
/*======================================
    2차 8번   2차 1급 8_initial_code.java
 ======================================*/

import java.util.*;

class Solution {
    public int[] solution(int[] arr) {
        int left = 0, right = arr.length - 1;
        int idx = 0;
        int[] answer = new int[arr.length];
        while(left <= right){
            if(left % 2 == 0){
                answer[idx] = arr[left];
                left += 1;
            }
            else{
                answer[idx] = arr[right];
                right -= 1;
            }
            idx += 1;
        }
        return answer;
    }

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

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

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제
- 원본 배열의 **첫 번째 항목**은 answer 의 **0 번 인덱스 항목으로**, 원본 배열의 **마지막 항목**은 answer 의 **1 번 인덱스 항목으로**, 다시 원본 배열의 **두 번째 항목**은 answer 의 **2 번 인덱스**

항목으로, 원본 배열의 **뒤에서 두 번째 항목**은 answer 의 **3 번 인덱스 항목으로**, 즉 원본 배열 arr 의 앞쪽 항목과 뒤쪽 항목을 answer 로 모두 복사될 때까지 번갈아 저장



➔ 원본 배열의 앞쪽 항목들은 answer 배열의 짝수 인덱스 항목으로, 원본 배열의 뒤쪽 항목들은 answer 배열의 홀수 인덱스 항목으로 저장

3) 정답

```
def solution(arr):
①   left, right = 0, len(arr) - 1
②   idx = 0
③   answer = [0 for _ in range(len(arr))]
    while left <= right:
        if idx % 2 == 0:
④           answer[idx] = arr[left]
            left += 1
        else:
⑤           answer[idx] = arr[right]
            right -= 1
⑥       idx += 1
    return answer

#아래는 테스트케이스 출력을 해보기 위한 코드입니다. 아래 코드는 잘못된
arr = [1, 2, 3, 4, 5, 6]
ret = solution(arr)

#[실행] 버튼을 누르면 출력 값을 볼 수 있습니다.
print("solution 함수의 반환 값은 ", ret, " 입니다.")
```

```java
public int[] solution(int[] arr) {
    int left = 0, right = arr.length - 1;
    int idx = 0;
    int[] answer = new int[arr.length];
    while(left <= right){
        if( idx % 2 == 0){
            answer[idx] = arr[left];
            left += 1;
        }
        else{
            answer[idx] = arr[right];
            right -= 1;
        }
        idx += 1;
    }
    return answer;
}
```

① left 는 원본 배열의 앞 부분 항목을 가리키는 인덱스, right 는 원본 배열의 뒷 부분 항목을 가리키는 인덱스로 사용
idx 는 answer 배열의 항목을 가리키는 인덱스로 사용
항목을 재배치하여 저장할 배열 answer 를 arr 배열과 같은 길이로 생성

② left 가 right 보다 작거나 같은 동안 새로 작성하는 answer 배열의 항목을 가리키는 idx 가 짝수이면 원본 배열의 앞부분 항목을 answer 배열에 할당하고, 원본 배열의 앞 부분을 가리키는 left 를 1 만큼 증가
idx 가 홀수이면 원본 배열의 뒷부분 항목을 answer 배열에 할당하고, 원본 배열의 뒷부분을 가리키는 right 를 1 만큼 감소

③ 원본 배열 항목을 가리키는 idx 를 1 만큼 증가

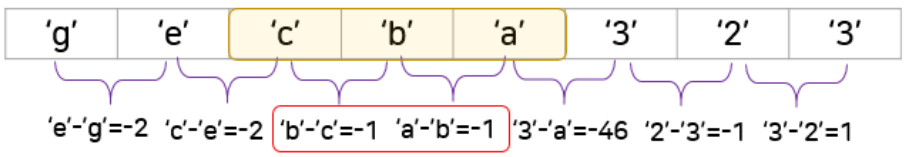
9. 문제 9

1) 문제 코드

```
/*=======================================
    2차 9번   2차 1급 9_initial_code.java
 =======================================*/

class Solution {
    public boolean solution(String password) {
        int length = password.length();
        for(int i = 0; i < length - 2; ++i){
            int firstCheck = password.charAt(i + 1) - password.charAt(i);
            int secondCheck = password.charAt(i) - password.charAt(i+1);
            if(firstCheck == secondCheck && (firstCheck == 1 || firstCheck == -1))
                return false;
        }
        return true;
    }


    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. main 메소드는 잘못된 부분
    public static void main(String[] args) {
        Solution sol = new Solution();
        String password1 = "cospro890";
        boolean ret1 = sol.solution(password1);

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


        String password2 = "cba323";
        boolean ret2 = sol.solution(password2);

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

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 한 줄만 수정하는 문제
- 암호 문자열에 연속된 3 자리 문자가 사용되었는지 확인하는 구문을 구현해야 함



➔ (뒷 문자 아스키 값 - 앞 문자 아스키 값)의 연산 결과로 -1 혹은 1 이 연속으로 두 번 나오면 안전한 비밀번호가 될 수 없음

3) 정답

```java
public boolean solution(String password) {
    int length = password.length();
    for(int i = 0; i < length - 2; ++i){
        int firstCheck = password.charAt(i + 1) - password.charAt(i);
        int secondCheck = password.charAt(i+2) - password.charAt(i+1);
        if(firstCheck == secondCheck && (firstCheck == 1 || firstCheck == -1))
            return false;
    }
    return true;
```

- solution( ) 메소드의 매개변수 password 로 전달된 문자열 중 첫 번째 문자부터 마지막 세 번째 문자까지 for 문을 이용하여 확인
  - 문제에서 제시된 코드는 현재 선택한 문자와 그 다음 문자만 비교하고 있음
  - 연달아 붙어 있는 세 개의 문자가 연속된 문자인지 확인하려면 (다음 문자의 아스키 값 - 현재 선택한 문자의 아스키 값) 과 (다음 다음 문자의 아스키 값 - 다음 문자의 아스키 값) 을 구하여 각각 firstCheck, secondCheck 에 저장해야 함
  - 두 문자에 대한 정수 값의 차이가 1 혹은 -1 이라면 두 문자는 연속된 문자열 관계임
  - firstCheck 와 secondCheck 가 모두 1 혹은 -1 인지 if 문을 사용하여 확인하여 그 결과가 참이면 false 를 return

10. 문제 10

1) 문제 코드

```java
/*=======================================
    2차 10번   2차 1급 10_initial_code.java
 =======================================*/

class Solution {
    public String solution(String s) {
        s += '#';
        String answer = "";
        for(int i = 0; i < s.length(); ++i){
            if (s.charAt(i) == '0' && s.charAt(i+1) != '0')
                answer += "0";
            else
                answer += "1";
        }
        return answer;
    }

    // 아래는 테스트케이스 출력을 해보기 위한 main 메소드입니다. main 메소드는 잘못
    public static void main(String[] args) {
        Solution sol = new Solution();
        String s = "101100011100";
        String ret = sol.solution(s);

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

2) 문제 개요

- 제시된 과제가 바르게 수행되도록 문제 코드를 수정하는 문제
- 문자열 안에서 연속되는 '0' 을 하나의 '0' 으로 줄여서 표시하도록 코드를 수정

3) 정답

```java
public String solution(String s) {
    s += '#';
    String answer = "";
    for(int i = 0; i < s.length(); ++i){
        if (s.charAt(i) == '0' && s.charAt(i+1) != '0')
            answer += "0";
        else if(s.charAt(i) == '1')
            answer += "1";
    }
    return answer;
}
```

① 매개변수로 전달받은 문자열 마지막에 임의로 '#' 를 붙임.

② for 문을 이용하여 문자열의 인덱스를 0 부터 차례로 가져와 현재 인덱스의 문자가 '0' 이고 다음 인덱스의 문자가 '0' 이 아니면 '0' 을 answer 에 추가

- for 문을 이용하여 i 가 s 의 마지막 인덱스 값을 가질 때 s[i+1]를 참조하면, 배열의 인덱스 범위를 벗어나므로 오류가 발생함. 따라서 for 문의 반복 범위를 정할 때 s.length()-1 을 써야 하는데 문제의 경우 변수 s 의 마지막 문자가 항상 '#' 이므로 s[i] == 0 이 되지 않기 때문에 'and' 연산 뒤에 오는 s[i+1] 자체를 실행하지 않아서 오류가 발생하지 않음
- 현재 인덱스의 문자가 '1' 이면 그대로 answer 에 추가하도록 조건식을 수정
- 만일 문제로 제시된 코드를 그대로 실행하면 현재 인덱스의 문자가 '0' 이고 다음 인덱스의 문자도 '0' 인 경우에 '1' 이 answer 에 추가됨

4) 다른 코딩 제안

```
public String solution(String s) {
    String answer = "" + s.charAt(0);
    for(int i = 1; i < s.length(); ++i){
        if ( (s.charAt(i-1)=='1') || (s.charAt(i)=='1') )
            answer+=s.charAt(i);

    }
    return answer;
}
```

- 맨 앞의 숫자는 무조건 추가
- 반복문 인덱스를 1 부터 시작하여 그 앞과 비교
- 현재 위치의 값이 1 이거나 그 이전 값이 1 이면 추가 (즉, 0 이 여러 개일 경우 처음 만난 0 을 추가)
- 0 뒤의 0 은 처리 안 함