**●알고리즘에서 복잡도란?**

알고리즘의 성능은 **시간복잡도**와 **공간복잡도**로 표현된다.  
 **시간 복잡도**는 입력 값의 개수와 알고리즘의 처리 시간과의 상관관계를 표현한 말이다. 입력 데이터 양이 많아짐에 따라 처리속도가 어떻게 변하는 지를 수학적으로 표현한 것이다.

**공간 복잡도**는 메모리 사용량의 변화를 수학적으로 표현한 것이다.

어떤 알고리즘의 입력 데이터 개수가 100개에서 1000개로 증가할 때 탐색 시간이 선형적으로 증가한다면 이 알고리즘은 ‘선형 시간 복잡도’를 가지고 있다고 한다.

알고리즘 성능은 빅오(big-O)표기법으로 표현한다. 이는 최악의 상황에서도 이 성능 이상을 보장한다는 의미이다.  
이때 n은 입력데이터 개수이다.

O(n)은 입력데이터 n개에 대해 n번 처리를 수행한다는 의미이다  
O(n2)는 입력데이터 n개에 대해 2\*n번 처리를 수행한다는 의미이다.  
O(1)은 입력데이터 n개에 상관없이 항상 고정된 시간이 걸린다는 의미이다.

-O(1) : 상수

아래 예제 처럼 입력에 관계없이 복잡도는 동일하게 유지된다.

def hello\_world():

print("hello, world!")

-O(N) : 선형

입력이 증가하면 처리 시간또는 메모리 사용이 선형적으로 증가한다.

def print\_each(li):

for item in li:

print(item)

-O(N^2) : Square

반복문이 두번 있는 케이스

def print\_each\_n\_times(li):

for n in li:

for m in li:

print(n,m)

-O(log n) O(n log n)

주로 입력 크기에 따라 처리 시간이 증가하는 정렬알고리즘에서 많이 사용된다.

다음은 이진검색의 예이다.

def binary\_search(li, item, first=0, last=None):

if not last:

last = len(li)

midpoint = (last - first) / 2 + first

if li[midpoint] == item:

return midpoint

elif li[midpoint] > item:

return binary\_search(li, item, first, midpoint)

else:

return binary\_search(li, item, midpoint, last)

**●DP 동적계획법(Dynamic Programming)**

큰 문제를 작은 문제로 나누어 푸는 알고리즘이다. 메모이제이션은 문제를 풀 때 각 계산과정을 저장해두는 것을 의마한다. 이렇게 한 번 했던 계산과정을 저장해두면 나중에 동일한 계산이 나오면 생략할 수 있게 되어 시간복잡도가 줄어든다. 또한 메모이제이션은 값을 저장하는 방법이므로 캐싱(Caching)이라고도 한다.

동적계획법은 모든 방법을 일일이 검토하여 최적의 해를 찾아내는 방식의 알고리즘이다.

그리디와 다른 점은 그리디는 현재 주어진 상황에서의 최적해를 찾기 때문에 그 해가 전체 중에서 최적해라고 할 수 없지만 동적계획법은 모든 방법을 계산한 뒤 전체에서의 최적해를 찾는 점이 다르다.

다이나믹 프로그래밍을 사용할 수 있는 조건은 다음과 같다.  
1. 큰 문제가 작은 문제로 나눠진다.  
2. 작은 문제에서 구한 답이 큰 문제에서도 포함된다.

-피보나치 수열 예시

먼저 재귀함수로 피보나치 수열을 구하는 메서드를 구현해본다.

int fibo( int n ) {  
 if( n==0 ) return 0;

if( n==1 ) return 1;

return fibo( n-1 ) + fibo( n-2 );

}

재귀함수로 구현시 중복되는 피보나치 메서드가 있다. 예를 들면 n이 5일 경우 계산하다보면   
fibo(5) = fibo(4) + fibo(3)  
fibo(4) = fibo(3) + fibo(2)

이때 fibo(3)은 계산 과정에서 중복됨을 알 수 있다.

이러한 중복과정을 없애기 위해 DP 알고리즘을 사용한다.  
아래 코드에서 memset함수는 dp라는 배열을 -1값으로 전부 초기화시켜준다.  
그리고 dp[n]에 피보나치 수열의 결과값이 저장되는데 이를 메모이제이션이라고 한다.

int dp[50];  
int fibo( int n ) {  
 if( n==0 ) return 0;  
 if( n==1 ) return 1;  
 if( dp[n] != -1 ) return dp[n];  
 return dp[n] = ( fibo( n-1 ) + fibo( n-2 ) );  
}  
public static main(String[] args) {  
 memset( dp, -1, sizeof( dp ) );  
 printf( “fibo(40)=%d\n”, fibo(40) );  
}

이처럼 재귀 함수를 이용하여 다이나믹 프로그래밍 소스코드를 작성하는 방법을, 큰 문제를 해결하기 위해 작은 문제를 호출한다고 하여 ‘탑다운방식’이라고 한다.  
반면에 단순히 반복문을 이용하여 소스코드를 작성하는 경우 작은 문제부터 차근차근 답을 도출한다고 하여 ‘바텀업방식’이라고 한다. 피보나치 수열 문제를 아래에서 위로 올라가는 바텀업방식으로 풀면 다음과 같다.

int dp[50];  
dp[1] = 1;  
dp[2] = 1;

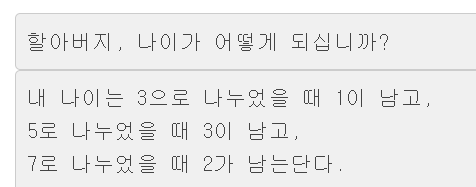
public static main(String[] args) {  
 memset( dp, -1, sizeof( dp ) );  
 for( int i=3; i<sizeof( dp ); i++ ) {  
 dp[i] = dp[i-1] + dp[i-2];  
 }  
 System.out.println( dp[50] );  
}

보통 다이나믹 프로그래밍의 전형적인 형태는 바텀업방식이다. 바텀업방식에서 사용되는 결과 저장용 리스트는 DP테이블 이라고하며 메모이제이션은 탑다운 방식에 국한되어 사용되는 표현이다. 또한 다이나믹 프로그래밍과 메모이제이션의 개념을 혼용하면 안된다. 메모이제이션은 단순히 이전에 계산된 결과를 일시적으로 기록해 놓는 넓은 개념이다.

**●브루트포스(Brute Force) 알고리즘**

brute는 무식한, force는 힘으로서 완전탐색 알고리즘이란 뜻이다. 즉, 가능한 모든 경우의 수를 모두 탐색하면서 요구조건에 충족되는 결과만을 가져온다.  
이 알고리즘의 강력한 점은 예외 없이 100%의 확률로 정답만을 출력한다.  
알고리즘 설계의 가장 기본적인 접근방법은 해가 존재할 것으로 예상되는 모든 영역을 전체 탐색하는 방법이다.  
선형 구조를 전체적으로 탐색하는 순차탐색, 비선형 구조를 전체적으로 탐색하는 깊이 우선 탐색과 너비우선탐색이 가장 기본적인 방법이다.

**●중국인의 나머지 정리(CRT)**

특정한 수 x를 여러 개의 수로 나눴을 때 나머지가 다 다른 경우 이를 만족하는 x를 구하는 것.  
즉, 아래와 같은 문제가 있다고 했을 때  
  
이를 식으로 나타내면  
x = 1 (mod 3)  
x = 3 (mod 5)  
x = 2 (mod 7)  
위 세 식을 만족하는 x를 구하는 것이다.  
  
이때 x는 아래와 같이 나타낼 수 있다. a는 첫 번째 식의 몫, b는 두 번째 식의 몫, c는 세 번째 식의 몫이다.  
x = 3a + 1  
x = 5b + 3  
x = 7c + 2  
이를 연립한다.