2022 AlgoLive 9th study

Dynamic Programming - I

동적 계획법(DP)

목차

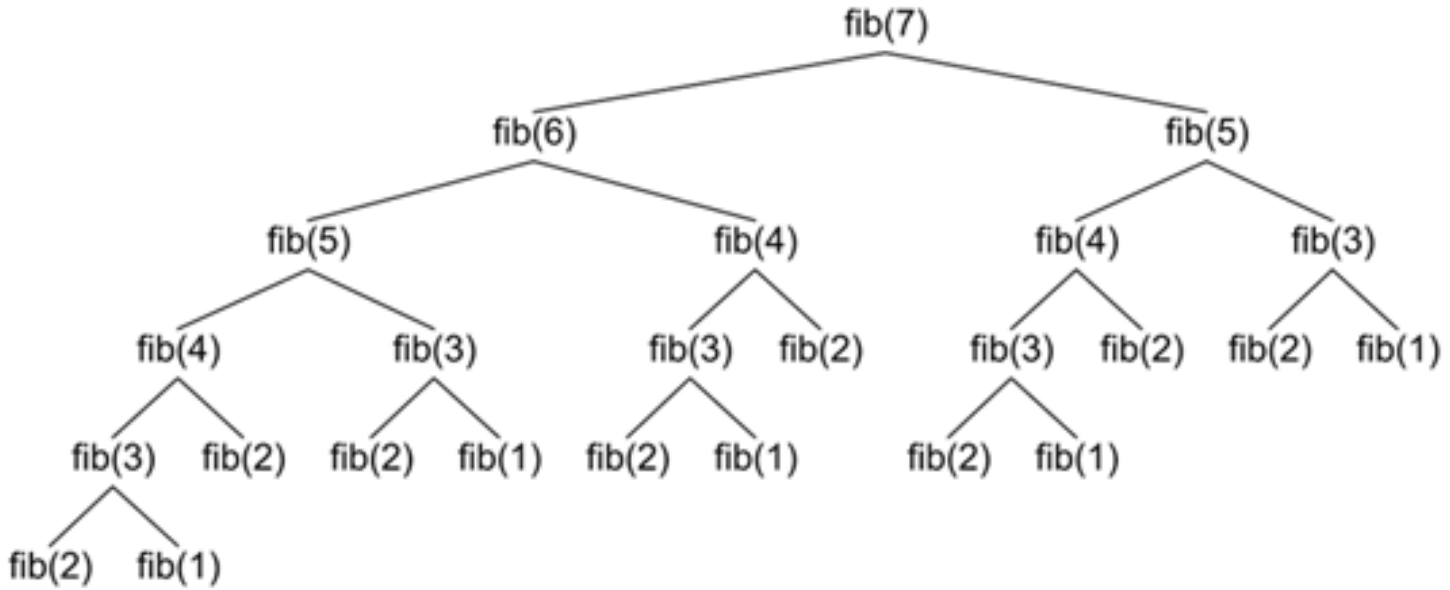
1. Dynamic Programming이란?

Dynamic Programming 이란?

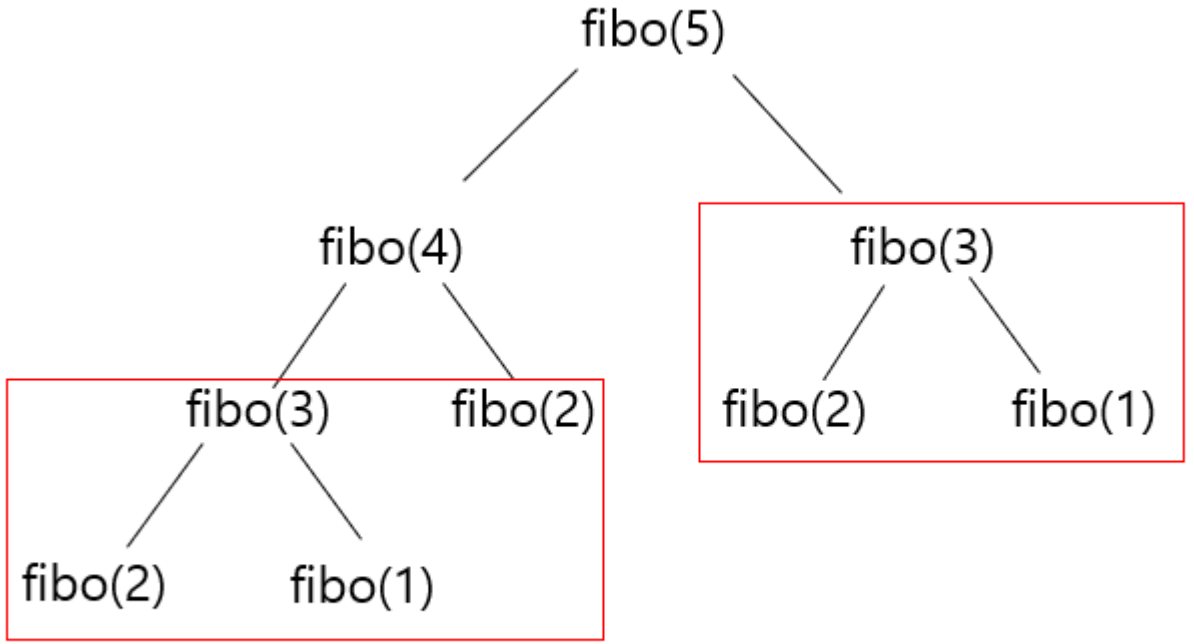
- **컴퓨터의 연산 속도와 메모리 공간은 한계 有**
  -> 효율적인 알고리즘을 위해 고안됨
- 큰 문제를 **작은 문제로 나누어 푸는** 방법
- 특정한 자료구조가 아닌 **문제에 대한 접근 방식과 풀이 방법**
- **두 가지 조건**을 만족할 때 DP를 사용할 수 있음
  - **부분 반복 문제**
  - **최적 부분 구조**

부분 반복 문제 (Overlapping Subproblem)

- 계속해서 **같은 부분 문제가 여러 번 재사용**되거나 **재귀 알고리즘**을 통해 해결되는 문제
- 즉, **작은 문제가 반복되어 일어날 때 DP를 사용**할 수 있음



부분 반복 문제 (Overlapping Subproblem)




```
fib(n) = fib(n-1) + fib(n-2);
```

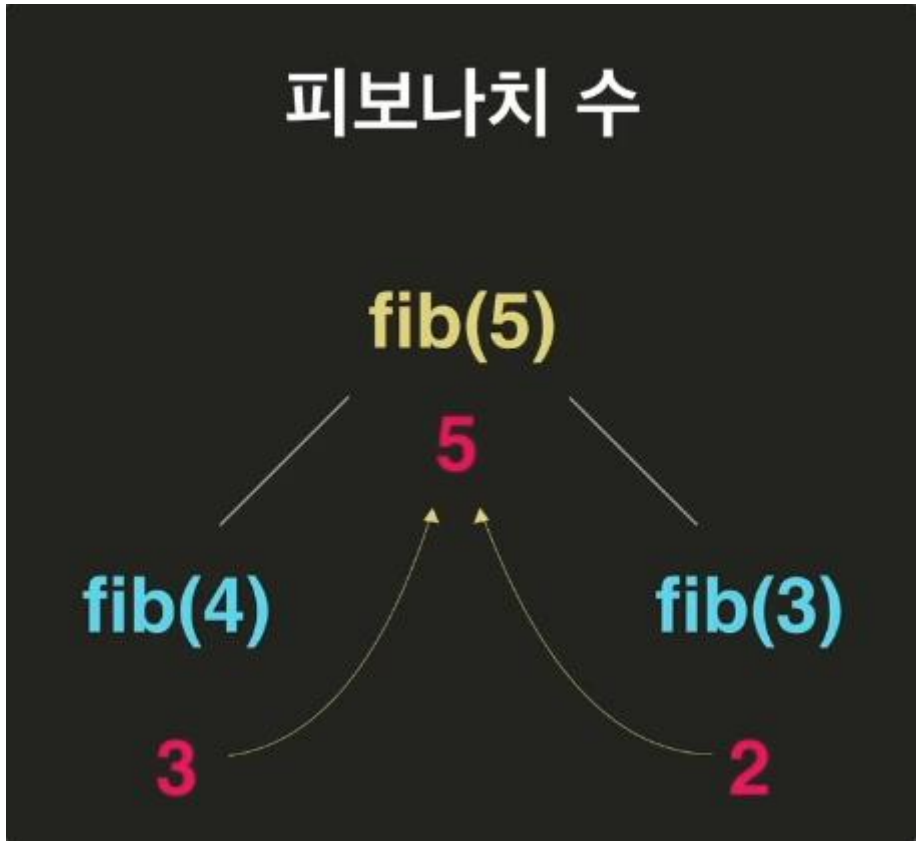
최적 부분 구조 (Optimal Substructure)

- **작은 부분 문제에서 구한 최적의 답**으로 **합쳐진 큰 문제의 최적의 답을 구할 수 있어야** 함

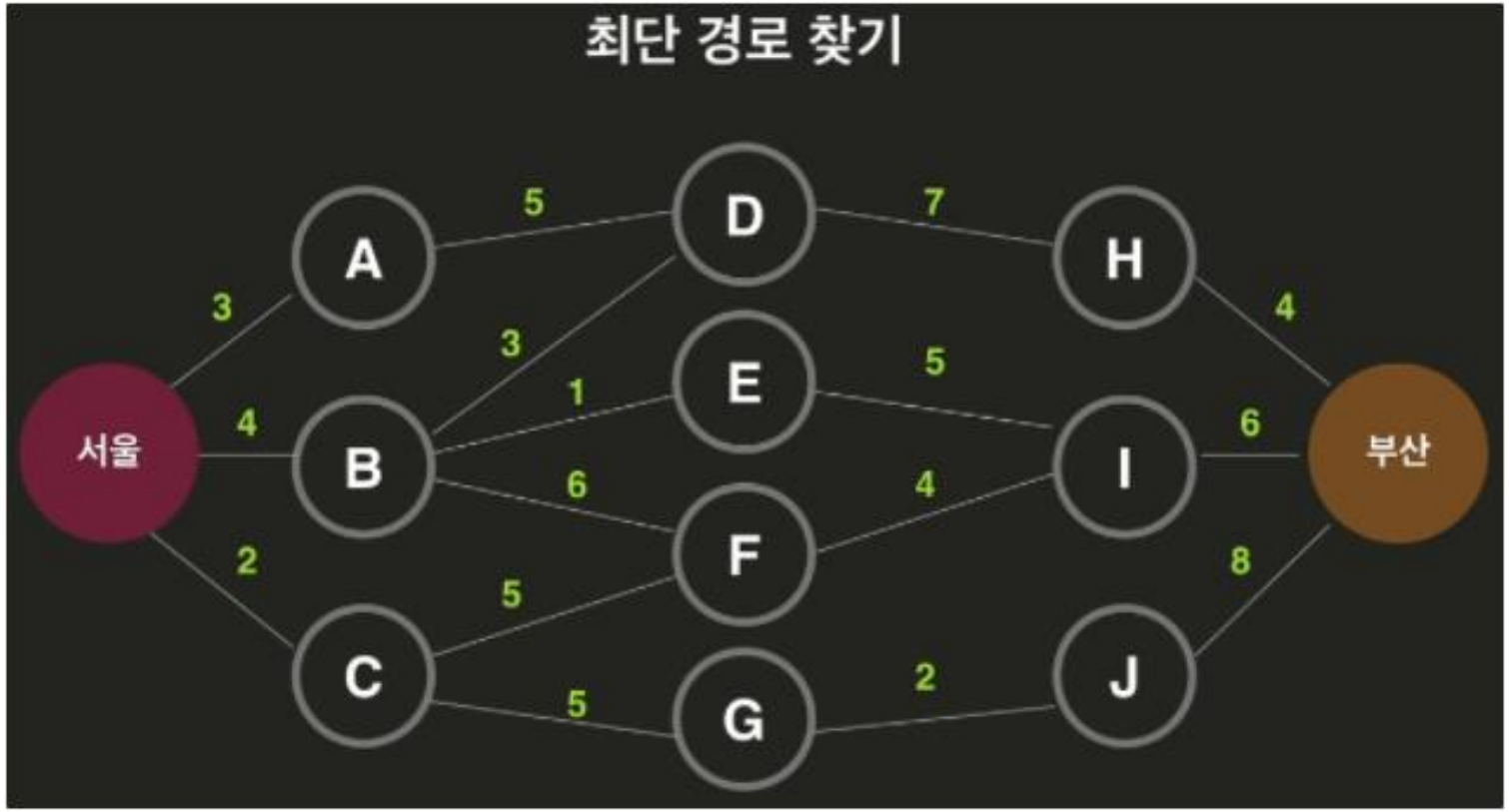
```
fib(n) = fib(n-1) + fib(n-2);
```

-> fib(n)을 구하려면, fib(n-1)과 fib(n-2)를 구하면 됨
-> fib(n-1)과 fib(n-2)가 정답이면 fib(n)도 정답

최적 부분 구조 (Optimal Substructure)

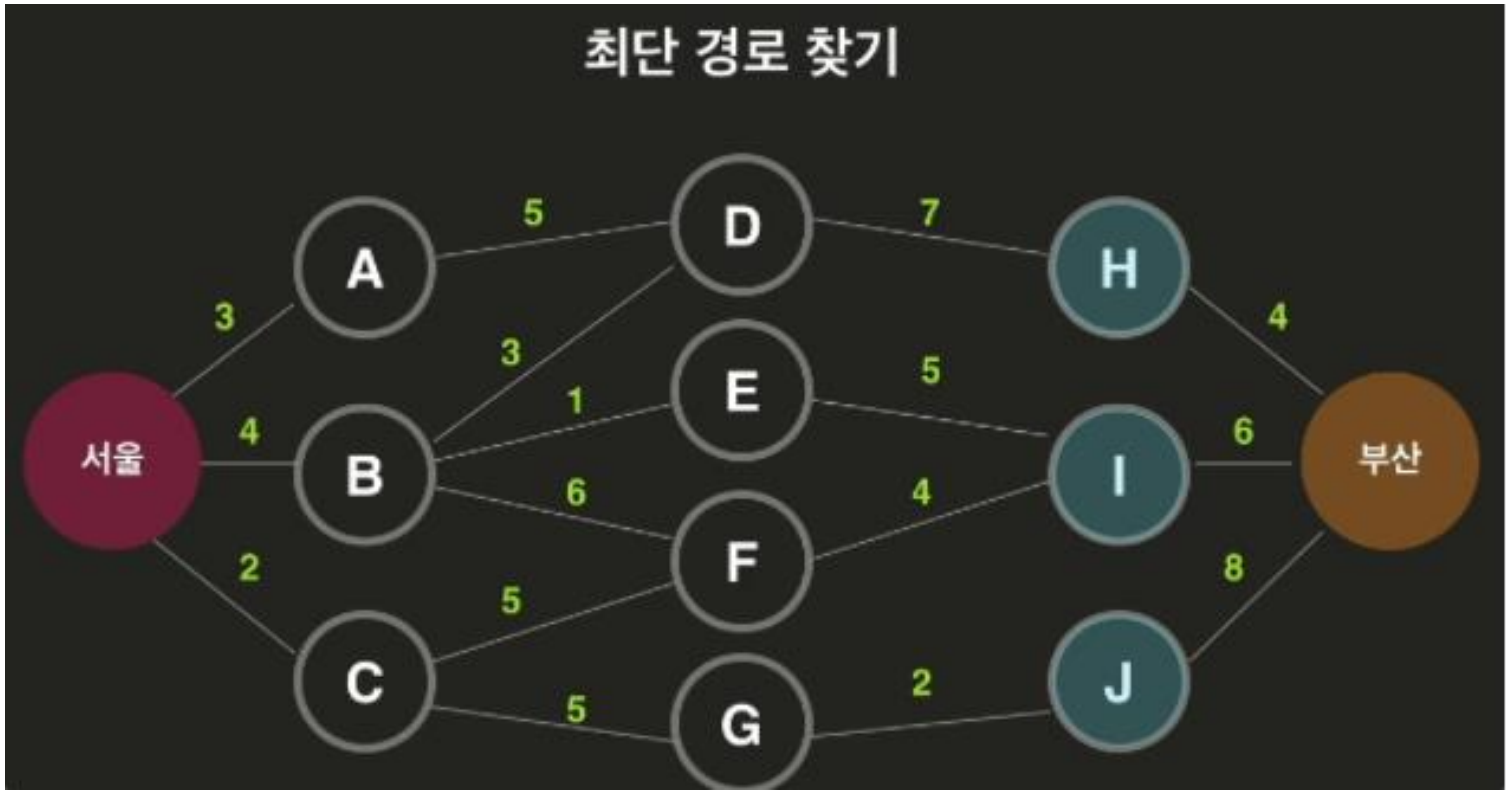


최적 부분 구조 (Optimal Substructure)



서울 -> 부산

최적 부분 구조 (Optimal Substructure)



서울 -> H -> 부산

서울 -> I -> 부산

서울 -> J -> 부산

⋮

2. Dynamic Programming 기법

DP의 두 가지 기법

Top-Down vs Bottom-Up

Top-Down

- 위에서 아래로 접근
- 큰 문제를 부분 문제로
  -> 재귀 호출

```cpp
#include <iostream>

using namespace std;

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

메모이제이션(Memoization)

컴퓨터 프로그램이 **동일한 계산을 반복**해야 할 때, 이전에 **계산한 값을 메모리에 저장**함으로써 동일한 계산의 **반복 수행을 제거**하여 프로그램 **실행 속도를 빠르게 하는 기술**

-> 즉, 리스트 등으로 **메모리에 계산한 값을 저장**

-> **다음 반복 수행 때는 연산 없이 저장된 값을 불러와 줌**

Top-Down

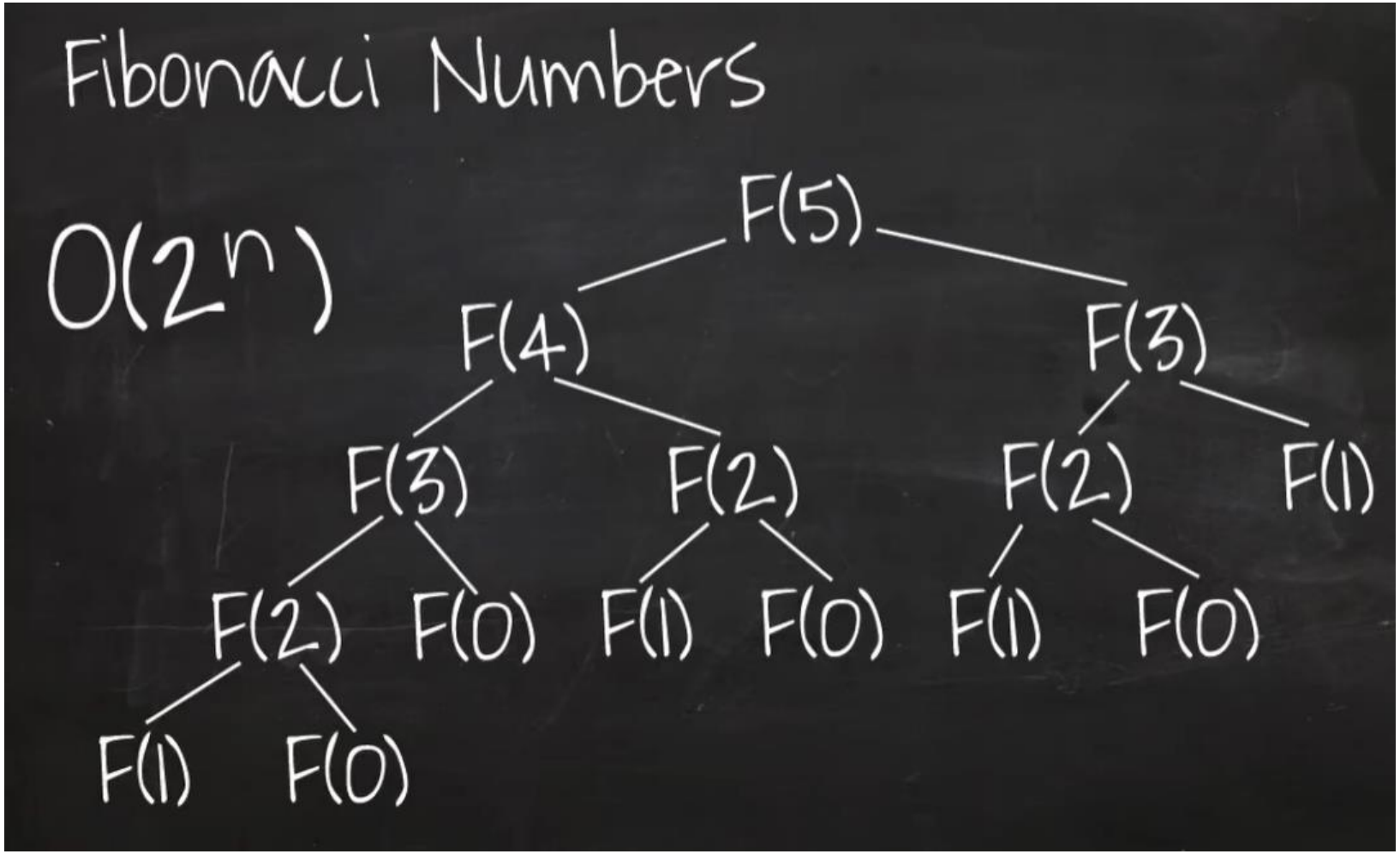
```cpp
#include <iostream>

using namespace std;

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

DFS 방식

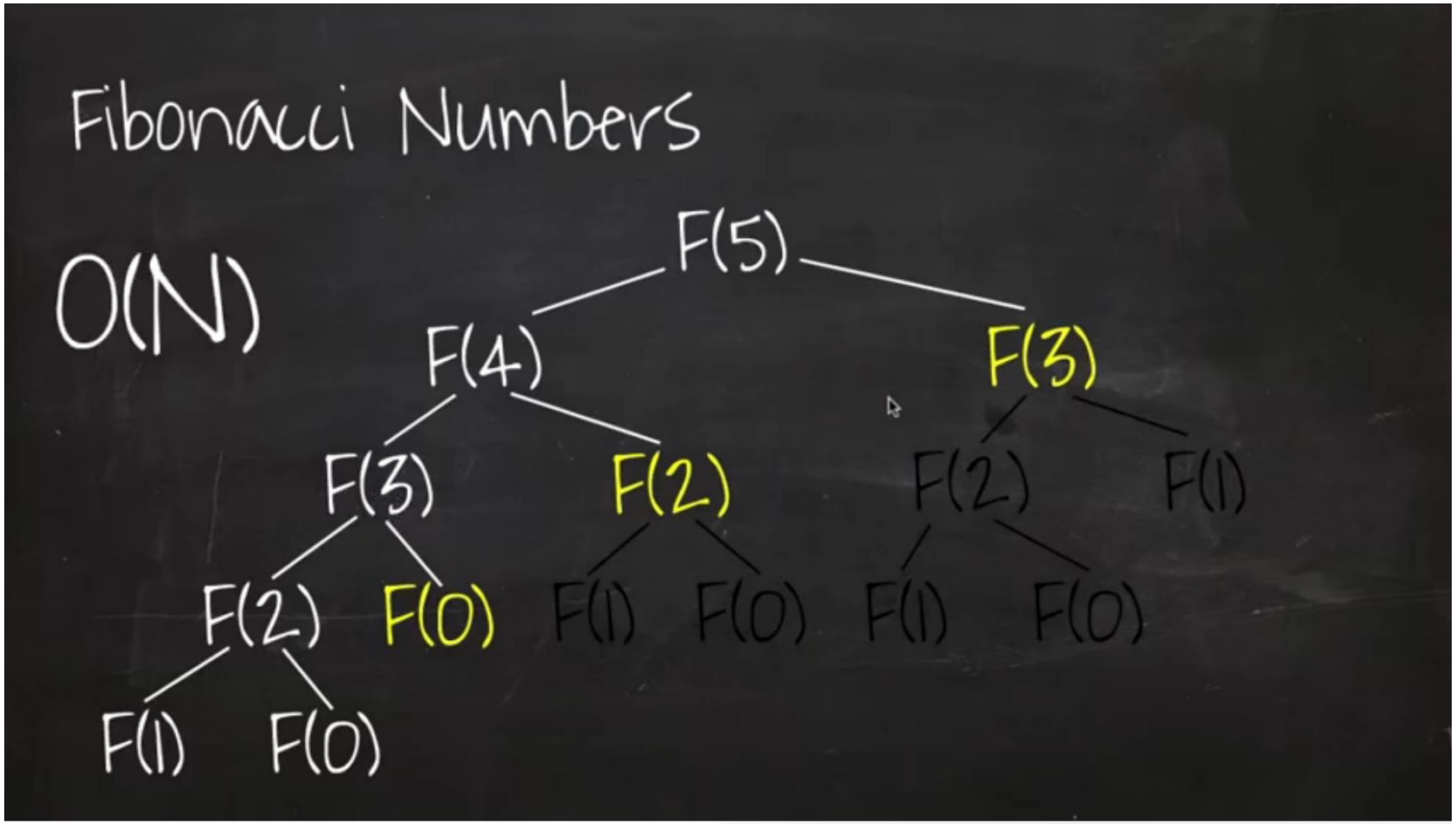
```cpp
#include <iostream>

using namespace std;

int dp[100] = {0, };

int fibonacci(int n) {
    if (n <= 1) return n;
    else {
        if (dp[n] > 0) return dp[n];
        dp[n] = fibonacci(n-1) + fibonacci(n-2);
        return dp[n];
    }
}
```

Memozation 방식

Top-Down

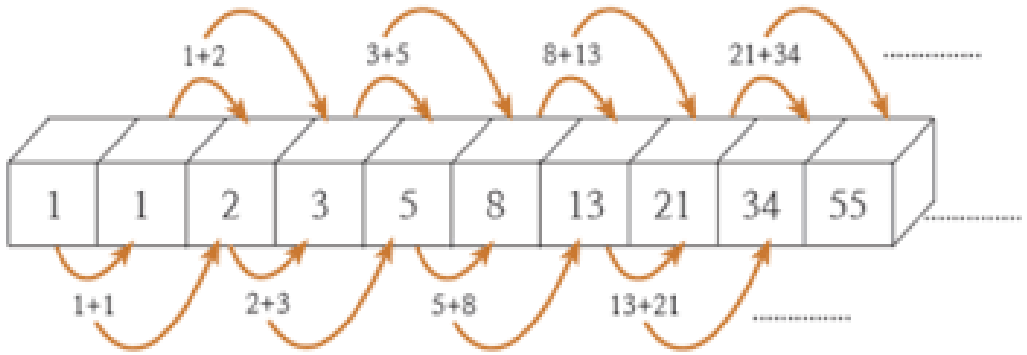


Top-Down



Bottom-Up

- 아래에서 위로 접근
- 부분 문제 -> 큰 문제
- for문 이용



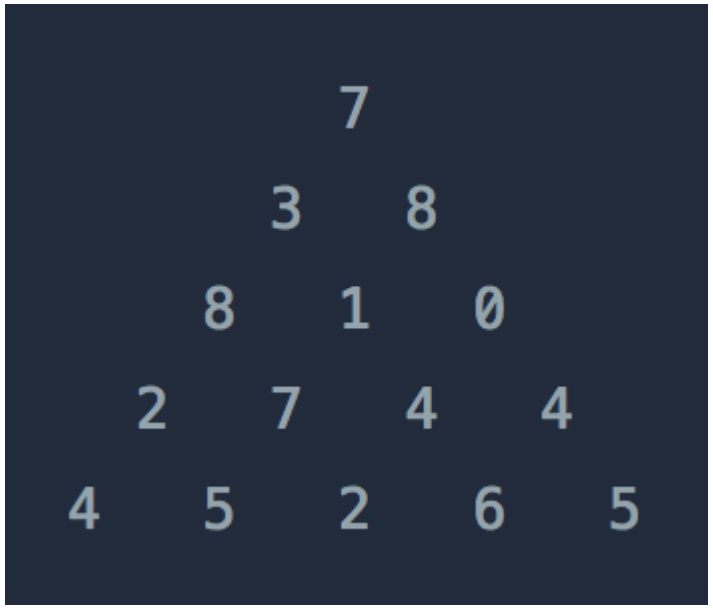

```
5   int memo[101];
6
7   memo[1] = 1;
8   memo[2] = 1;
9
10  int fibonacci(int n){
11      for (int i = 3; i <= n; ++i){
12          memo[i] = memo[i-1] + memo[i-2];
13      }
14      return memo[n];
15  }
```

3. 활용

활용 방법

1. DP가 가능한 문제인지 조건 확인
2. 문제를 부분 문제로 나눠 표현 (ex. 점화식)
3. Top-Down / Bottom-Up 방식을 지정해 답 도출

정수 삼각형



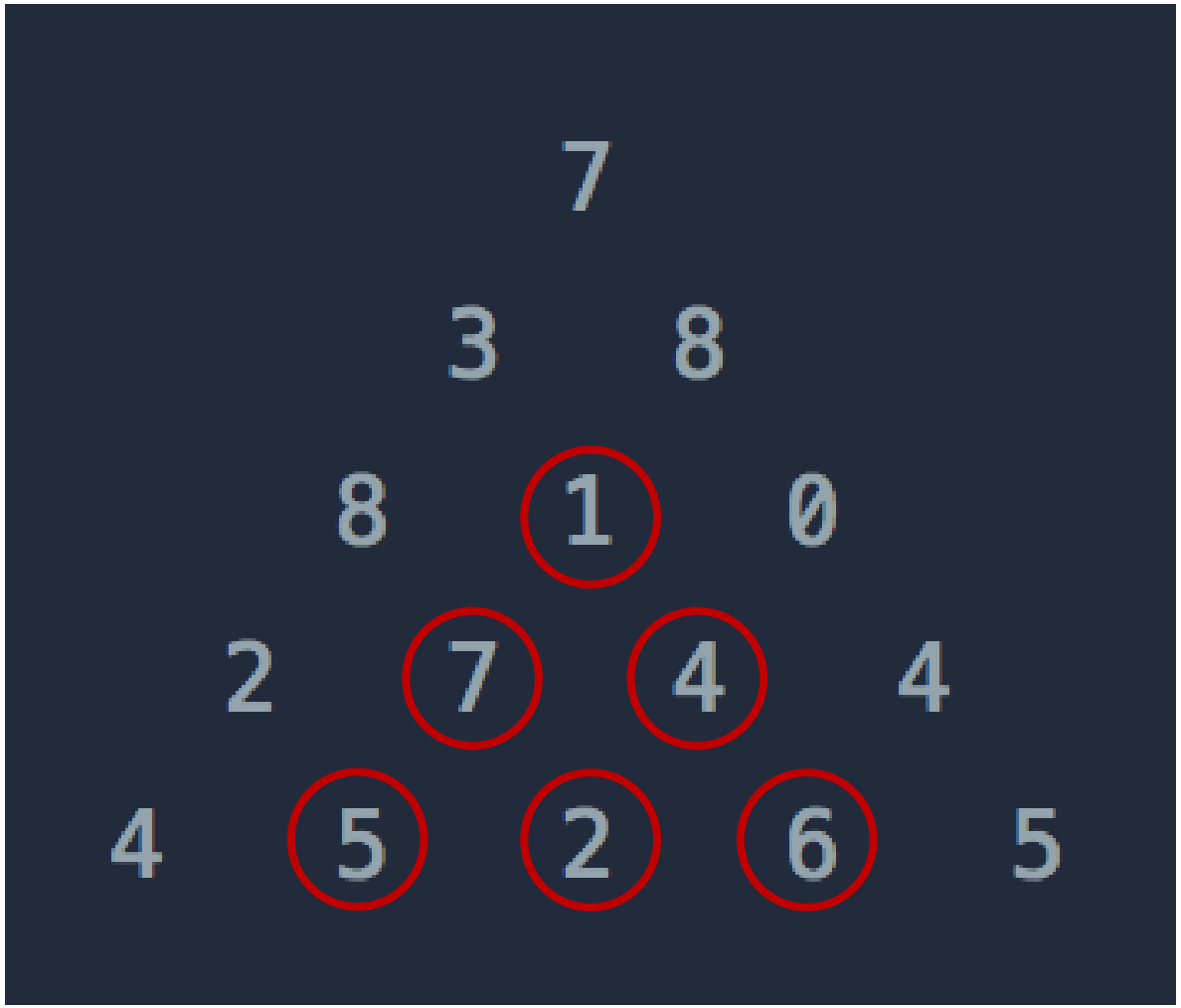
제한사항

- 삼각형의 높이는 1 이상 500 이하입니다.
- 삼각형을 이루고 있는 숫자는 0 이상 9,999 이하의 정수입니다.

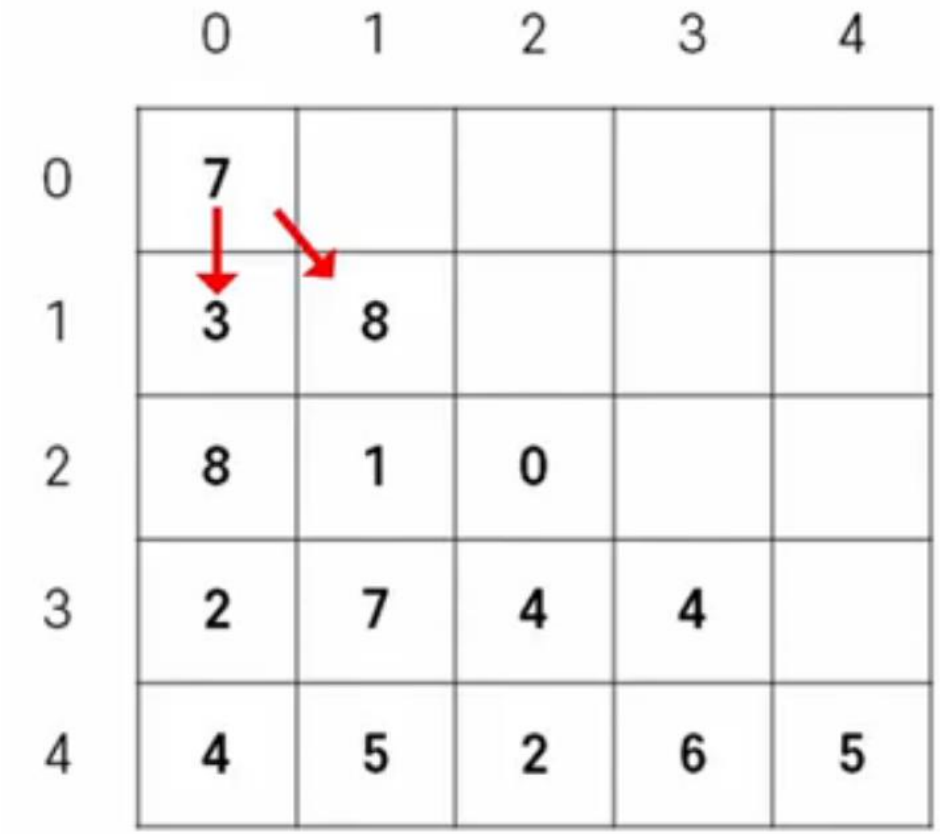
위와 같은 삼각형의 꼭대기에서 바닥까지 이어지는 경로 중, 거쳐간 숫자의 합이 가장 큰 경우를 찾아보려고 합니다. 아래 칸으로 이동할 때는 대각선 방향으로 한 칸 오른쪽 또는 왼쪽으로만 이동 가능합니다. 예를 들어 3에서는 그 아래칸의 8 또는 1로만 이동이 가능합니다.

삼각형의 정보가 담긴 배열 triangle이 매개변수로 주어질 때, 거쳐간 숫자의 최댓값을 return 하도록 solution 함수를 완성하세요.

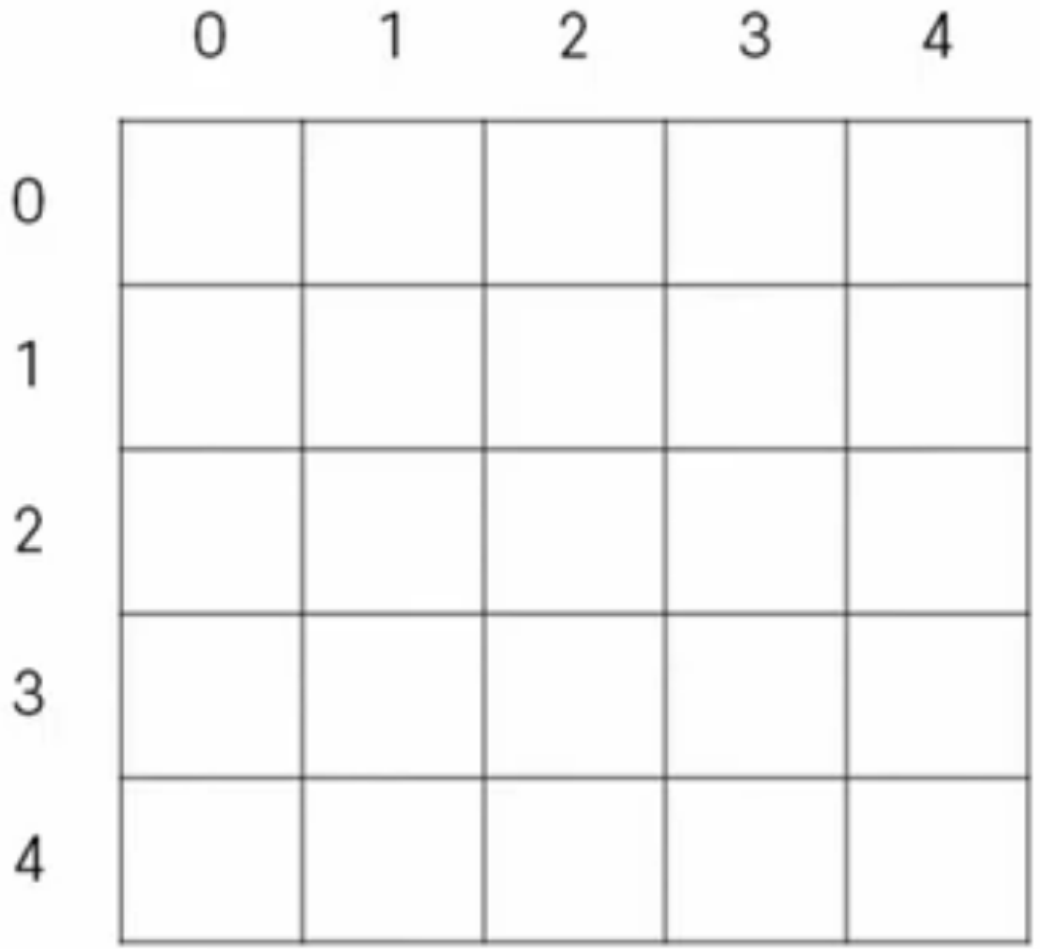
정수 삼각형



정수 삼각형



	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5



	0	1	2	3	4
0					
1					
2					
3					
4					

정수 삼각형

Memo(0, 0) = Value(0, 0)

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1					
2					
3					
4					

answer = 7

정수 삼각형

Memo(1, n) = Memo(0, 0) + Value(1, n)

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2					
3					
4					

answer = 15

정수 삼각형

Memo(2, 0) = Memo(1, 0) + Value(2, 0)

Memo(2, 1) = Max(Memo(1, 0), Memo(1, 1)) + Value(2, 1)

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3					
4					

answer = 18

정수 삼각형

Memo(3, 0) = Memo(2, 0) + Value(3, 0)

Memo(3, 1) = Max(Memo(2, 0), Memo(2, 1)) + Value(3, 1)

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3	20	25	20	19	
4					

answer = 25

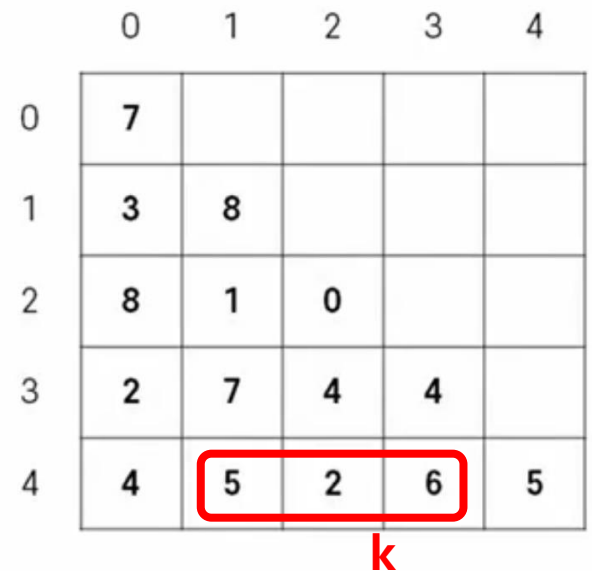
정수 삼각형

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3	20	25	20	19	
4	24	30	27	26	24

answer = 30

정수 삼각형



x가 n일 때 (즉, n번째 줄일 때)

Memo(n, 0) = Memo(n-1, 0) + Value(n, 0)

Memo(n, n) = Memo(n-1, n) + Value(n, n)

Memo(n, k) = Max(Memo(n-1, k-1), Memo(n-1, k)) + Value(n, k)

(0 < k < n)

Bottom-Up C++

```cpp
#include <string>
#include <vector>
#define max_int 501

using namespace std;
int answer, height, d[max_int][max_int];

int max(int a, int b){
    return a > b ? a : b;
}

int solution(vector<vector<int>> triangle) {
    answer = d[0][0] = triangle[0][0];
    height = (int)triangle.size();

    for(int i=1; i<height; i++){
        for(int j=0; j<=i; j++){
            if(j == 0){
                d[i][j] = d[i-1][j] + triangle[i][j];
            }else if(j == i){
                d[i][j] = d[i-1][j-1] + triangle[i][j];
            }else{
                d[i][j] = max(d[i-1][j-1], d[i-1][j]) + triangle[i][j];
            }

            answer = max(answer, d[i][j]);
        }
    }

    return answer;
}
```

정수 삼각형

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10				
2	18				
3	20				
4	24				

answer = x

정수 삼각형

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2	18	16			
3	20	25			
4	24	30			

answer = x

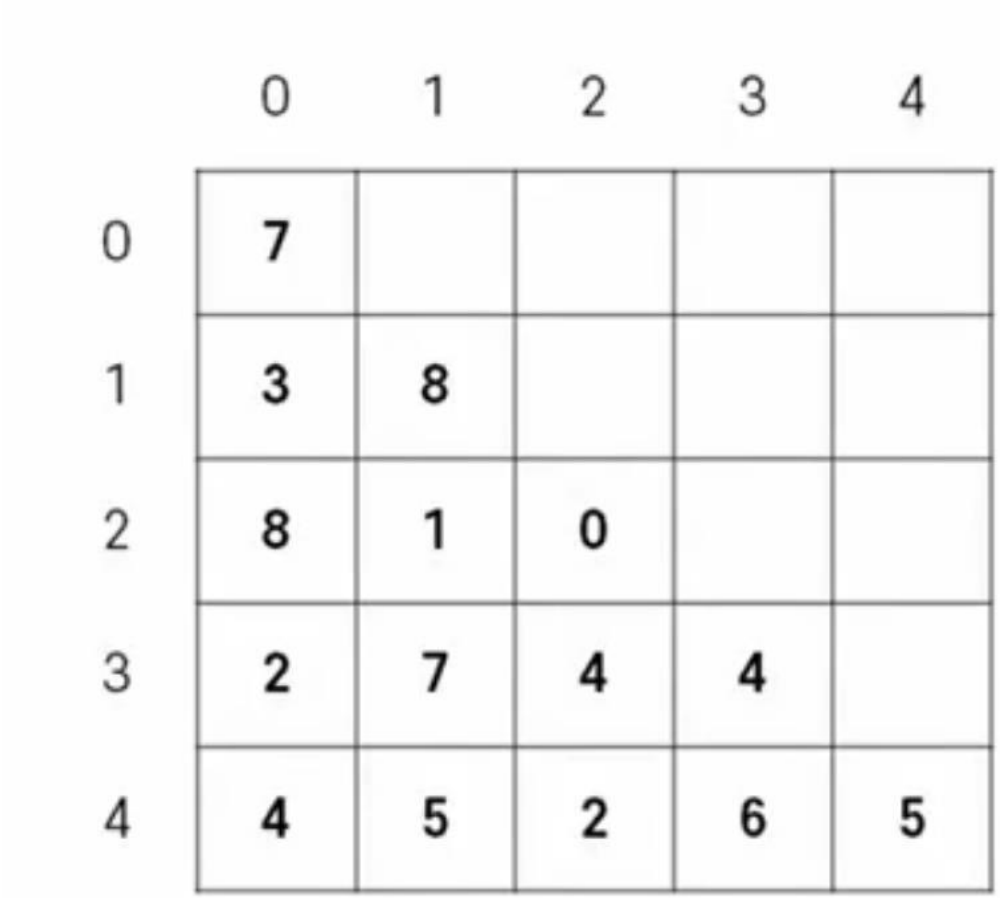
정수 삼각형

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

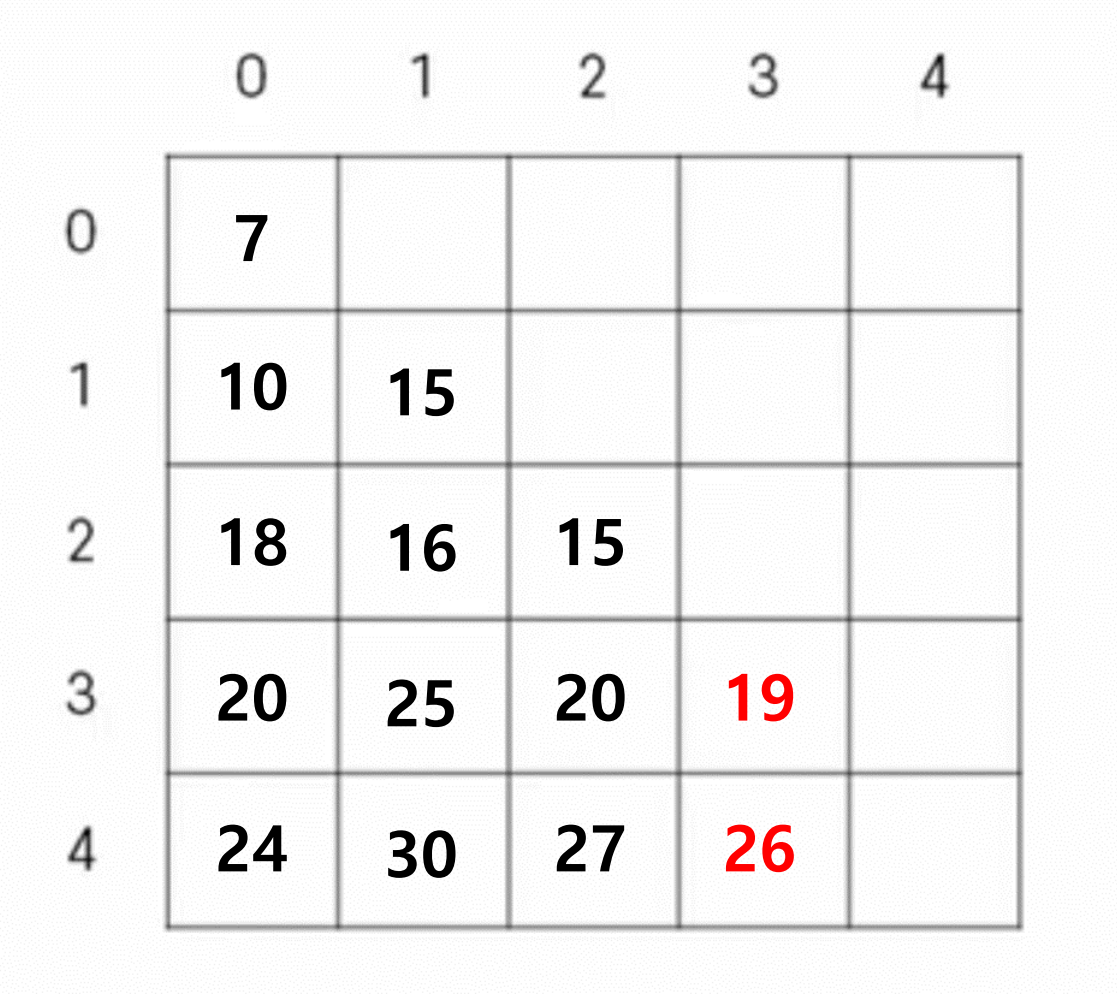
	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3	20	25	20		
4	24	30	27		

answer = x

정수 삼각형



	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5



	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3	20	25	20	19	
4	24	30	27	26	

answer = x

정수 삼각형

	0	1	2	3	4
0	7				
1	3	8			
2	8	1	0		
3	2	7	4	4	
4	4	5	2	6	5

	0	1	2	3	4
0	7				
1	10	15			
2	18	16	15		
3	20	25	20	19	
4	24	30	27	26	24

answer = 30

Top-Down C++

```cpp
#include <string>
#include <vector>
#define max_int 501

using namespace std;

int answer, height, t[max_int][max_int], d[max_int][max_int];
// t -> triangle array 표현
// d -> 메모이제이션

int max(int a, int b){
    return a > b ? a : b;
}

int go(int i, int j){
    if(i == 0 && j == 0) return d[i][j];

    if(d[i][j] > 0) return d[i][j];

    for(int j=0; j<=i; j++){
        if(j == 0){
            d[i][j] = go(i-1, j) + t[i][j];
        }else if(j == i){
            d[i][j] = go(i-1, j-1) + t[i][j];
        }else{
            d[i][j] = max(go(i-1, j-1), go(i-1, j)) + t[i][j];
        }
    }
    return d[i][j];
}
```

```cpp
int solution(vector<vector<int>> triangle) {
    d[0][0] = triangle[0][0];
    height = (int)triangle.size();

    /*
     최대 500 * 500인 벡터를 재귀호출때 마다 인자값으로 넣어주면 시간초과 걸린다.
     전역변수에 넣어주었다.
     */
    for(int i=0; i<height; i++){
        for(int j=0; j<=i; j++){
            t[i][j] = triangle[i][j];
        }
    }

    for(int j=0; j<height; j++){
        answer=max(answer, go(height - 1, j));
    }

    return answer;
}
```

Top-Down

Practice

- #11053
- #1010

#11053

가장 긴 증가하는 부분 수열 성공

☆

실버 II

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	109112	42913	28233	37.316%

문제

수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 프로그램을 작성하시오.

예를 들어, 수열 A = {10, 20, 10, 30, 20, 50} 인 경우에 가장 긴 증가하는 부분 수열은 A = {**10**, **20**, 10, **30**, 20, **50**} 이고, 길이는 4이다.

입력

첫째 줄에 수열 A의 크기 N (1 ≤ N ≤ 1,000)이 주어진다.

둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. (1 ≤ A_i ≤ 1,000)

출력

첫째 줄에 수열 A의 가장 긴 증가하는 부분 수열의 길이를 출력한다.

#1010

다리 놓기

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
0.5 초 (추가 시간 없음)	128 MB	56176	25922	21217	48.328%

문제

재원이는 한 도시의 시장이 되었다. 이 도시에는 도시를 동쪽과 서쪽으로 나누는 큰 일직선 모양의 강이 흐르고 있다. 하지만 재원이는 다리가 없어서 시민들이 강을 건너는데 큰 불편을 겪고 있음을 알고 다리를 짓기로 결심하였다. 강 주변에서 다리를 짓기에 적합한 곳을 사이트라고 한다. 재원이는 강 주변을 면밀히 조사해 본 결과 강의 서쪽에는 N개의 사이트가 있고 동쪽에는 M개의 사이트가 있다는 것을 알았다. ($N \le M$)

재원이는 서쪽의 사이트와 동쪽의 사이트를 다리로 연결하려고 한다. (이때 한 사이트에는 최대 한 개의 다리만 연결될 수 있다.) 재원이는 다리를 최대한 많이 지으려고 하기 때문에 서쪽의 사이트 개수만큼 (N개) 다리를 지으려고 한다. 다리끼리는 서로 겹쳐질 수 없다고 할 때 다리를 지을 수 있는 경우의 수를 구하는 프로그램을 작성하라.

#11053 – C++

```cpp
#include<bits/stdc++.h>

using namespace std;

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int N; cin >> N;
    vector<int> V(N), DP(N);
    for(auto &i: V) cin >> i;
    for(int i=0;i<N;i++) {
        DP[i] = 1;
        for(int j=0;j<i;j++) {
            if(V[i] > V[j]) {
                DP[i] = max(DP[i], DP[j] + 1);
            }
        }
    }
    cout << *max_element(DP.begin(), DP.end());

    return 0;
}
```

#11053 – Java

```java
import java.util.*;
import java.io.*;

public class Main {
    static int dp[];
    static int arr[];
    public static void main(String[] args) {
        FastReader rd = new FastReader();

        int N = rd.nextInt();
        // dp[i]는 1 ~ i까지 'i번째 숫자를 포함한' 최장길이
        dp = new int[N + 10];
        arr = new int[N + 10];

        // 배열을 입력 받음과 동시에 dp값을 전부 1로 초기화 합니다.
        // 항상 길이는 최소 1 이상이기 때문
        for(int i = 1;i <= N;i++) {
            arr[i] = rd.nextInt();
            dp[i] = 1;
        }

        // 1 ~ (i - 1) 까지 숫자를 살피는데 arr[i]보다 작고
        // dp[i] < dp[j] + 1 이라면 dp[i]값을 갱신합니다.
        for(int i = 2;i <= N;i++) {
            for(int j = 1;j < i;j++) {
                if(arr[i] > arr[j] && dp[i] < dp[j] + 1)
                    dp[i] = dp[j] + 1;
            }
        }

        // dp배열을 돌며 최대값을 찾아 출력합니다.
        int max = 0;
        for(int i = 1;i <= N;i++)
            max = Math.max(max, dp[i]);

        System.out.println(max);
    }
    static class FastReader {
        BufferedReader br;
        StringTokenizer st;

        public FastReader() {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        String next() {
            while(st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }

        int nextInt() { return Integer.parseInt(next()); }
        long nextLong() { return Long.parseLong(next()); }
        String nextLine() {
            String str = "";
            try {
                str = br.readLine();
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }
}
```

#1010 – C++, Python

```cpp
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;

ll DP[33][33];

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int T; cin >> T;
    DP[0][0] = 1;
    for(int i=1;i<=30;i++) {
        DP[i][0] = 1;
        for(int j=1;j<=30;j++) {
            DP[i][j] = DP[i - 1][j] + DP[i - 1][j - 1];
        }
    }
    while(T--) {
        int N, M; cin >> N >> M;
        cout << DP[M][N] << '\n';
    }

    return 0;
}
```

```python
import sys
def input():
    return sys.stdin.readline().rstrip()

T = int(input())

dp = [[0 for _ in range(31)] for _ in range(31)]
dp[0][0] = 1

for num in range(1,31):
    dp[num][0] = 1
    for pick in range(1,31):
        dp[num][pick] = dp[num-1][pick] + dp[num-1][pick-1]

for _ in range(T):
    N, M = map(int,input().split())
    print(dp[M][N])
```

#1010 – Java

```java
import java.util.*;
import java.io.*;

public class Main {
    static int[][] dp = new int[30][30];

    public static void main(String[] args) {
        FastReader rd = new FastReader();

        int T = rd.nextInt();

        // 조합공식을 이용하여 풀이가능 (M C N)
        dp[0][0] = 1;
        for(int j = 1;j < 30;j++) {
            dp[j][0] = 1;
            for(int k = 1;k <= j;k++)
                dp[j][k] = dp[j - 1][k - 1] + dp[j - 1][k];
        }

        for(int i = 0;i < T;i++) {
            int N = rd.nextInt();
            int M = rd.nextInt();

            System.out.println(dp[M][N]);
        }
    }

    static class FastReader {
        BufferedReader br;
        StringTokenizer st;

        public FastReader() {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        String next() {
            while(st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }

        int nextInt() { return Integer.parseInt(next()); }
        String nextLine() {
            String str = "";
            try {
                str = br.readLine();
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }
}
```