다이나믹 프로그래밍(D.P.)

Gaussian

다이나믹 프로그래밍(D.P.)

- 큰 문제를 작은 문제로 나눠서 푸는 알고리즘
- 다음 두가지 속성을 만족해야 다이나믹 프로그래밍의 조건이 된다.

  ① Overlapping subproblem (문제들이 겹쳐야 한다.)

  : 피보나치 수에서 $F_n = F_{n-1} + F_{n-2}$의 점화식에는 작은 문제들이 겹쳐져 있다.

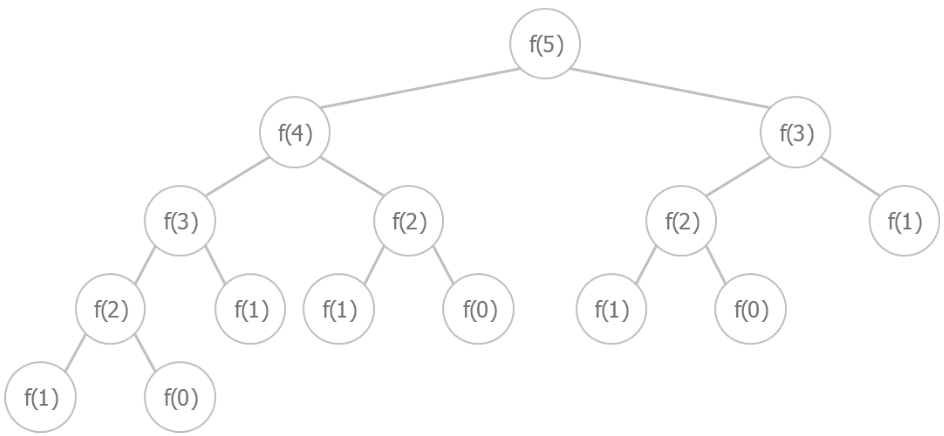
  ② Optimal substructure (각 문제들이 최적인 상태이어야 한다.)

  : 서울 → 대전 → 대구 → 부산 이 서울 → 부산으로 가는 최적의 길이라면
  대전 → 대구 → 부산이 대전 → 부산으로 가는 최적의 길이어야 한다.
  만약, 서울 → 부산으로 갈때에는 서울 → 대전 → 대구 → 부산 이 가장 빠르고
  대전 → 부산으로 갈때에는 대전 → 울산 → 부산이면 Optimal substructur가 아니다.
- 다이나믹 프로그래밍에서 각 문제는 한 번만 풀어야 한다. → 메모이제이션 필요
- Optimal substructure를 만족하므로 같은 문제는 같은 문제는 구할 때마다 값이 항상 같다.
- 피보나치 수 : $F_0 = 0, F_1 = 1, F_2 = 2$
- $F_n = F_{n-1} + F_{n-2}\ (n \geq 2)$
- 문제 : N번째 피보나치 수를 구하는 문제
- 작은 문제 : N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제…

Gaussian

다이나믹 프로그래밍(D.P.)

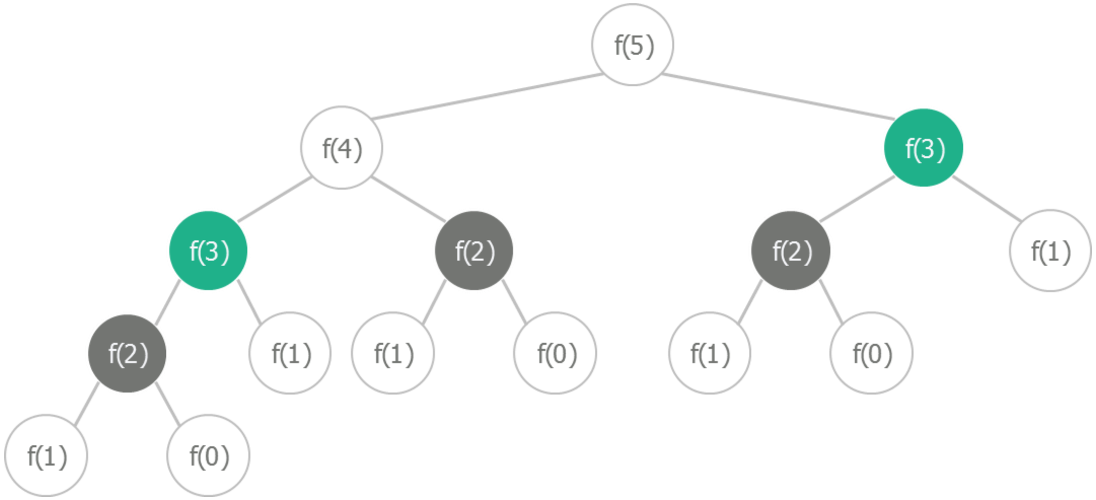
- 피보나치 수 : $F_0 = 0$, $F_1 = 1, F_2 = 2$
- $F_n = F_{n-1} + F_{n-2}\ (n \geq 2)$
- 문제 : N번째 피보나치 수를 구하는 문제
- 작은 문제 : N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제…




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

Gaussian

다이나믹 프로그래밍(D.P.)




```
int memo[100];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        if (memo[n] > 0) {
            return memo[n];
        }
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
        return memo[n];
    }
}
```

```
int d[100];
int fibonacci(int n) {
    d[0] = 1;
    d[1] = 1;
    for (int i=2; i<=n; i++) {
        d[i] = d[i-1] + d[i-2];
    }
    return d[n];
}
```

Gaussian

다이나믹 프로그래밍(D.P.)

① Top – Down 방식 : 재귀 호출을 이용하여 풀 수 있다.

- 문제를 작은 문제로 나눈다. : fibo(n) = fibo(n-1) + fibo(n-2)
- 작은 문제를 푼다. : fibo(n-1)과 fibo(n-2)를 호출해 푼다.
- 작은 문제를 푼 결과를 이용하여 전체 문제를 푼다.
  : fibo(n-1)과 fibo(n-2)결과로 fibo(n)을 푼다.

② Bottom – Up 방식 : 반복문을 이용하여 풀 수 있다.

- 문제를 크기가 작은 문제 부터 차례대로 푼다.
- 문제의 크기를 조금씩 크게 만들면서 문제를 점점 푼다.
- 작은 문제를 풀면서 점점 큰 문제를 접근하기 때문에 항상 큰 문제를 풀 수 있다.

Gaussian