Dynamic Programming

Dynamic Programming (DP)

- DP
  - 복잡한 문제를 간단한 문제들로 나누고
  - 간단한 문제들을 해결하고
  - 해결한 답을 조합해 복잡한 문제의 답을 구함

Dynamic Programming (DP)

- DP
  - 복잡한 문제를 간단한 문제들로 나누고
  - 간단한 문제들을 해결하고
  - 해결한 답을 조합해 복잡한 문제의 답을 구함
- 그렇다면
  - 복잡한 것과 간단한 것의 기준?
  - 항상 가능한가?
  - 효율적인가?

최적 부분 구조 (Optimal Substructure)

- 복잡한 문제의 최적해가 간단한 문제의 최적해를 **포함**한다
  - 복잡한 문제는 큰 문제, 간단한 문제는 작은 문제라고 보면 된다
- 부분 구조가 아니라면 간단한 문제의 답을 이용해서 복잡한 문제의 답을 구할 수 없다
  - DP가 항상 가능하지는 않다

피보나치 수 5 BOJ 10870

- 다음과 같이 정의된 피보나치 수열의 n번째 항 구하기 $(n \le 20)$

$$f_i = \begin{cases} 0 \ (i = 0) \\ 1 \ (i = 1) \\ f_{i-1} + f_{i-2} \ (i \ge 2) \end{cases}$$

피보나치 수 5 BOJ 10870

- 재귀적으로 그대로 함수를 작성하면?

```
int fibo(int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibo(n - 1) + fibo(n - 2);
}
```

- 이 코드의 시간복잡도는?

피보나치 수 5 BOJ 10870

- $fibo(5)$를 불러보자
  - $fibo(4) + fibo(3)$
  - $fibo(3) + fibo(2) + fibo(2) + fibo(1)$
  - $fibo(2) + fibo(1) + fibo(1) + fibo(0) + fibo(1) + fibo(0) + 1$
  - $fibo(1) + fibo(0) + 1 + 1 + 0 + 1 + 0 + 1$
  - $1 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 5$

피보나치 수 5 BOJ 10870

- $T(n) = T(n-1) + T(n-2) > 2T(n-2)$
- $= 2\big(T(n-3) + T(n-4)\big) > 2\big(2T(n-4)\big) = 4T(n-4)$
- $= \cdots = 2^n T(0) = O(2^n)$

- 다행히 2^{20}은 $1\ 000\ 000$ 정도의 크기라 1초 안에 해결 가능하다

피보나치 수 5 BOJ 10870

- 다음과 같이 정의된 피보나치 수열의 n번째 항 구하기
- 결국 $fibo(i)$는 $a \times fibo(0) + b \times fibo(1)$로 나타낼 수 있다
  - 최적 부분 구조 (Optimal Substructure)를 가진다

$$f_i = \begin{cases} 1 \ (i = 0, 1) \\ \boxed{f_{i-1} + f_{i-2} \ (i \geq 2)} \end{cases}$$

중복 부분 문제 (Overlapping Subproblem)

- 간단한 문제의 답을 **여러 번** 사용한다
- 한 번 답을 계산하여 저장하면, 다시 해당 문제로 나뉘었을 때 저장된 값을 사용하면 됨
  - 연산량을 줄일 수 있다

피보나치 수 2 BOJ 2748

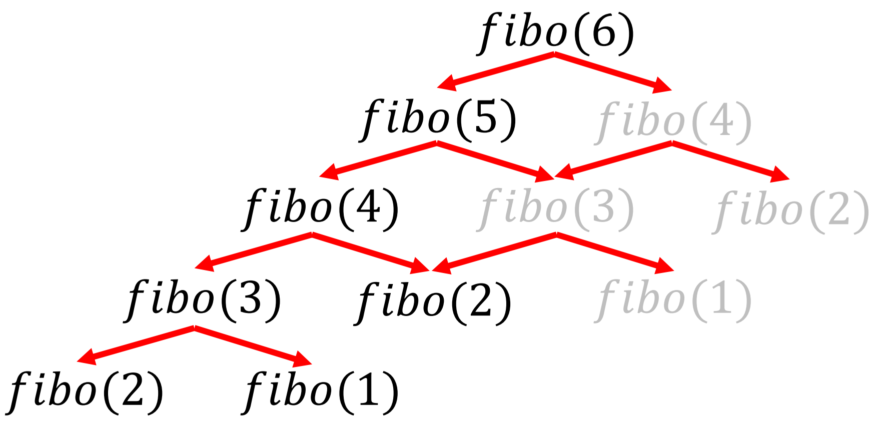
- 다음과 같이 정의된 피보나치 수열의 n번째 항 구하기 $(n \leq 90)$

$$f_i = \begin{cases} 0 \ (i = 0) \\ 1 \ (i = 1) \\ f_{i-1} + f_{i-2} \ (i \geq 2) \end{cases}$$

피보나치 수 2 BOJ 2748

- 트리의 형태에서 겹치는 부분은 한 번만 계산
- 한 번 계산한 값은 바뀌지 않음 (f_4는 항상 3임)

```
// f[0] = 0, f[1] = 1
int fibo(int n) {
    int& ret = f[n];
    if (ret != -1) return ret;
    ret = fibo(n - 1) + fibo(n - 2);
    return ret;
}
```



- 이 코드의 시간복잡도는?

피보나치 수 2 BOJ 2748

- 점화식을 이용해서 아래에서 계산을 하면서 채워도 된다

```
// f[0] = 0, f[1] = 1
int fibo(int n) {
    for (int i = 2; i <= n; i++)
        f[i] = f[i - 1] + f[i - 2];
    return f[n];
}
```

- 이 코드의 시간복잡도는?

Top-down과 Bottom-up

```c
int fibo(int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibo(n - 1) + fibo(n - 2);
}
```

```c
// f[0] = 0, f[1] = 1
int fibo(int n) {
    for (int i = 2; i <= n; i++)
        f[i] = f[i - 1] + f[i - 2];
    return f[n];
}
```

- 이렇게 중복되는 계산을 저장하여 동일한 계산에 대해 반복 계산을 방지하는 기법을 memoization 이라고 한다
- Top-down은 함수의 재귀호출로 인한 오버헤드가 있어서 일반적으로 Bottom-up이 더 빠르게 작동한다 (이로 인해 Top-down으로 풀리지 않는 문제도 존재한다)

Dynamic Programming

- 복잡한 문제를 한 개 이상의 간단한 문제로 분할할 수 있어야 함
- 복잡한 문제와 간단한 문제를 동일한 방법으로 해결할 수 있어야 함
- 복잡한 문제의 최적해가 간단한 문제들의 최적해들로 구성되어야 함

1로 만들기 BOJ 1463

- 정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지이다
  - X가 3으로 나누어 떨어지면, 3으로 나눈다
  - X가 2로 나누어 떨어지면, 2로 나눈다
  - 1을 뺀다
- 정수 N이 주어졌을 때, 위와 같은 연산 세 개를 적절히 사용해서 1을 만들려고 한다
- 연산을 사용하는 횟수의 최솟값을 출력해라 N $(1 \le N \le 10^6)$

1로 만들기 BOJ 1463

- 항상 나누면 1보다 더 큰 값이 줄어든다
  - 이게 최선일까?
- 10을 1로 만들려면 몇 번의 연산이 필요할까?
  - 10 → 5 → 4 → 2 → 1

1로 만들기 BOJ 1463

- 항상 나누면 1보다 더 큰 값이 줄어든다
  - 이게 최선일까?
- 10을 1로 만들려면 몇 번의 연산이 필요할까?
  - 1을 먼저 빼고 나눈다면?
  - 10 → 9 → 3 → 1

1로 만들기 BOJ 1463

- 수학적 귀납법으로 확인해보자
  - $f(n) = n$을 1로 만드는데 드는 최소 연산 횟수
  - Base: $n = 1$일 때, 연산이 필요하지 않다 → 0
  - Step: $n = i$일 때,
    - i가 3의 배수라면, $f(\frac{i}{3})$이 올바른 값을 구해준다면, 3으로 나누는 비용 1을 더하고 최솟값으로 유지한다
    - 2의 배수, 1을 뺄 때에도 마찬가지로 진행한다

1로 만들기 BOJ 1463

- 이미 계산을 했던 값이라면 바로 반환하자

```
int f(int x) {
    if (x == 1) return 0;
    int& ret = dp[x];
    if (ret != INF) return ret;
    if (x % 3 == 0) ret = min(ret, f(x / 3) + 1);
    if (x % 2 == 0) ret = min(ret, f(x / 2) + 1);
    ret = min(ret, f(x - 1) + 1);
    return ret;
}
```

1로 만들기 BOJ 1463

- Bottom-up 방식으로도 풀어보자

```
dp[0] = dp[1] = 0;
for (int i = 2; i <= X; i++) {
    dp[i] = dp[i - 1] + 1;
    if (i % 2 == 0) dp[i] = min(dp[i], dp[i / 2] + 1);
    if (i % 3 == 0) dp[i] = min(dp[i], dp[i / 3] + 1);
}
```

Dynamic Programming

- 지금이야 주제가 DP이므로 문제를 접근하는 것에 어려움이 없음
- 이 문제가 DP문제인지 판단하는 방법은?
  - Optimal Substructure 성질을 만족하는지를 본다
  - 지금까지 문제를 풀어본 **경험**을 토대로 추측한다
    - 최소/최대 비용, 트리, 경우의 수, 확률, …
  - DP문제들은 문제를 많이 풀어서 유형을 외우는 것이 편함

Dynamic Programming

- DP인 것을 알았다면?
  - 복잡한 문제와 간단한 문제 간의 상관 관계(점화식)는 어떻게 찾을까?

  - 점화식을 정의해보자
    - 현재 **상태**를 잘 표현할 방법을 생각한다
      - 배열 인덱스, 선택한 원소의 개수 / 합 / 곱 / ... 등
    - 다른 방식으로 표현할 수 없는지 생각한다
    - **상태**의 **차원**을 줄여도 온전하게 표현 가능한지 생각한다
      - 현재 상태를 (A + B, A, B)로 표현한 경우 → (A + B, A)만 사용해도 온전히 표현할 수 있음
  - 관계를 찾는다
    - 현재 **상태**로 가기 바로 **이전 상태**는 어떠할까

스위치 BOJ 30460

- i초에 $A_i(|A_i| \le 1\ 000)$의 점수를 얻는 게임을 $N(3 \le N \le 200\ 000)$초 동안 진행한다
  - T초에 스위치를 눌러 $T, T+1, T+2$초에 얻는 점수를 2배로 만들 수 있다
  - T초에 스위치를 누르면 $T+3$초부터 다시 스위치를 누를 수 있다
- 점수를 최대로 얻어보자

스위치 BOJ 30460

- i초에 $A_i(|A_i| \leq 1\ 000)$의 점수를 얻는 게임을 $N(3 \leq N \leq 200\ 000)$초 동안 진행한다
  - T초에 스위치를 눌러 $T, T+1, T+2$초에 얻는 점수를 2배로 만들 수 있다
  - T초에 스위치를 누르면 $T+3$초부터 다시 스위치를 누를 수 있다
- 점수를 최대로 얻어보자

-2	10	2	-7	9	1	-2	-3	4

스위치 BOJ 30460

- i초에 $A_i(|A_i| \le 1\ 000)$의 점수를 얻는 게임을 $N(3 \le N \le 200\ 000)$초 동안 진행한다
  - T초에 스위치를 눌러 $T, T+1, T+2$초에 얻는 점수를 2배로 만들 수 있다
  - T초에 스위치를 누르면 $T+3$초부터 다시 스위치를 누를 수 있다
- 점수를 최대로 얻어보자

-2	10	2	-7	9	1	-2	-3	4

스위치 BOJ 30460

- i초에 $A_i(|A_i| \leq 1\ 000)$의 점수를 얻는 게임을 $N(3 \leq N \leq 200\ 000)$초 동안 진행한다
  - T초에 스위치를 눌러 $T, T+1, T+2$초에 얻는 점수를 2배로 만들 수 있다
  - T초에 스위치를 누르면 $T+3$초부터 다시 스위치를 누를 수 있다
- 점수를 최대로 얻어보자

-4	20	4	-7	18	2	-4	-3	8

스위치 BOJ 30460

- 현재 상태를 다음과 같이 정의하자
  - i초에 스위치를 누르고 2배가 되는 시간이 T초가 남았을 때의 최대 점수
  - $dp(i, T)$로 표현 가능
  - $dp(i, 0)$는 스위치가 눌리지 않은 상태로 정의하자
- 초항은?
  - $dp(1, 0) = A_1$ → 1초에 스위치가 눌리지 않은 상태이다
  - $dp(1, 1) = -INF$ → 1초에서 스위치를 누르고 2배가 되는 시간이 1초가 남을 수 없다
  - $dp(1, 2) = -INF$ → 1초에서 스위치를 누르고 2배가 되는 시간이 2초가 남을 수 없다
  - $dp(1, 3) = A_1 \times 2$ → 1초에 스위치를 눌러서 점수가 2배가 되었다

스위치 BOJ 30460

- 이어서 해보면
  - $dp(i,0) = \max\big(dp(i-1,0), dp(i-1,1)\big) + A_i$
    - i초에 스위치가 눌리지 않은 상태는 $i-1$초에 스위치가 눌리지 않은 상태이거나 2배가 되는 시간이 1초 남은 상태이다
  - $dp(i,3) = \max\big(dp(i-1,0), dp(i-1,1)\big) + A_i \times 2$
    - i초에 스위치를 누를 수 있는 상태는 $i-1$초에 스위치가 눌리지 않은 상태이거나 2배가 되는 시간이 1초 남은 상태이다
  - $dp(i,2) = dp(i-1,3) + A_i \times 2$
    - i초에 2배가 되는 시간이 2초가 남은 상태가 되려면 $i-1$초에 스위치를 막 누른 상태이다
  - $dp(i,1) = dp(i-1,2) + A_i \times 2$
    - i초에 2배가 되는 시간이 1초가 남은 상태가 되려면 $i-1$초에 2배가 되는 시간이 2초가 남은 상태여야 한다
- 마지막 항에서 최댓값을 찾으면 된다
  - $\max(dp(n,0), dp(n,1), dp(n,2), dp(n,3))$

스위치 BOJ 30460

```cpp
const int MAXN = 200'200;
const int INF = 0x3f3f3f3f;

int N;
long long A[MAXN], dp[MAXN][4];
int main(void) {
    // 점수를 얻는 총 시간 입력
    cin >> N;
    // i초에 얻는 점수 입력
    for (int i = 1; i <= N; i++)
        cin >> A[i];

    // DP 초항 작성
    dp[1][0] = A[1], dp[1][1] = -INF, dp[1][2] = -INF, dp[1][3] = A[1] << 1;

    for (int i = 2; i <= N; i++) {
        dp[i][0] = max(dp[i - 1][0], dp[i - 1][1]) + A[i];
        dp[i][3] = max(dp[i - 1][0], dp[i - 1][1]) + (A[i] << 1);
        dp[i][2] = dp[i - 1][3] + (A[i] << 1);
        dp[i][1] = dp[i - 1][2] + (A[i] << 1);
    }

    // DP 마지막 항에서 정답 갱신
    long long answer = -INF;
    for (int i = 0; i <= 3; i++)
        answer = max(answer, dp[N][i]);

    // 정답 출력
    cout << answer << endl;
    return 0;
}
```

스위치 BOJ 30460

- 차원을 줄일 수 있을까?
  - $dp(i,1), dp(i,2)$의 상태는 가변적이지 않고 이전 상태에 고정되어 있음
- 스위치를 i초에 누르거나 누르지 않았을 때의 최대 점수로 dp를 정의해도 된다
  - $dp(i,0)$ → i초에 스위치를 누르지 않았을 때
  - $dp(i,1)$ → i초에 스위치를 눌렀을 때

LCS(Longest Common Sequence)

- 최장 공통 부분 수열
  - **ACA**Y**K**P
  - C**AP**CA**K**
- 공통 부분 수열은 이어지지 않아도 되며, 한 쪽으로 읽어 나갔을 때 겹치는 문자들이다
- 최장 공통 부분 수열은 이 중에서 가장 긴 것

LCS BOJ 9251

- 두 문자열의 LCS(Longest Common Subsequence)의 길이를 구하여라
  - 문자열은 최대 1000글자로 이루어져 있다

- 어떻게 나타내야 할까?

LCS BOJ 9251

- 두 문자열의 LCS(Longest Common Subsequence)의 길이를 구하여라
  - 문자열은 최대 1000글자로 이루어져 있다

- $f(i, j)$: $a[1..i], b[1..j]$의 LCS길이

LCS BOJ 9251

- 두 문자열의 LCS(Longest Common Subsequence)의 길이를 구하여라
  - 문자열은 최대 1000글자로 이루어져 있다

- $f(i,j)$: $a[1..i], b[1..j]$의 LCS길이
- Base: $i \leq 0$ 또는 $j \leq 0$, LCS는 0
- Step:
  - $a[i] = b[j]$인 경우, $f(i-1, j-1)$에 1을 더한 값
  - 그렇지 않은 경우, $f(i-1, j), f(i, j-1)$중 더 큰 값

LCS BOJ 9251

- $f(i, j)$: $a[1..i], b[1..j]$의 LCS길이

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0						
A	0						
P	0						
C	0						
A	0						
K	0						

LCS BOJ 9251

- $a[i] \neq b[j]$ 라면, $f(i-1, j)$와 $f(i, j-1)$에서 더 큰 값을 취한다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0					
A	0						
P	0						
C	0						
A	0						
K	0						

LCS BOJ 9251

- $a[i] == b[j]$ 라면, 각자 한 글자 전에서 LCS 길이를 1씩 더한 것과 같다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1				
A	0						
P	0						
C	0						
A	0						
K	0						

LCS BOJ 9251

- $a[i] \neq b[j]$ 라면, $f(i-1, j)$와 $f(i, j-1)$에서 더 큰 값을 취한다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1			
A	0						
P	0						
C	0						
A	0						
K	0						

LCS BOJ 9251

- 구하려고 했던 값은 $f(N, M)$이다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 두 문자열의 LCS(Longest Common Subsequence)를 구하여라
  - 문자열은 최대 1000글자로 이루어져 있다
- 최장 공통 부분 수열
  - **ACA**Y**K**P
  - C**APCAK**
- ACAK 부분까지 구해야 한다

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 만약 $f(i,j)$의 값과 같은 값이 인접해 있다면 해당 방향으로 이동한다
  - 이는 이전 상태에 이미 LCS가 존재한다는 의미이다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 만약 $f(i,j)$의 값과 같은 값이 인접해 있지 않다면 $f(i-1,j-1)$방향으로 이동한다
  - 현재 상태에서 문자가 동일하여 LCS가 만들어졌다는 의미이다

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

LCS 2 BOJ 9252

- 역추적을 해보자

	-	A	C	A	Y	K	P
-	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
P	0	1	1	2	2	2	3
C	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
K	0	1	2	3	3	4	4

Prefix Sum

- 길이 N인 배열이 주어진다
- $l, r(1 \le l, r \le N)$이 주어졌을 때 $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$의 값을 구하자

- Bruteforce
  - 위의 식을 그대로 계산

구간 합 구하기 4 BOJ 11659

- 길이 N인 배열이 주어진다
- $l, r (1 \leq l, r \leq N)$이 주어졌을 때 $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$의 값을 Q번 구하자
- $N, Q \leq 100\,000$
- Bruteforce: $O(NQ)$

구간 합 구하기 4 BOJ 11659

- 길이 N인 배열이 주어진다
- $l, r (1 \leq l, r \leq N)$이 주어졌을 때 $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$의 값을 Q번 구하자
- $N, Q \leq 100\,000$
- 모든 구간에 대한 전처리를 하더라도 $O(N^2 + Q)$

구간 합 구하기 4 BOJ 11659

- 길이 N인 배열이 주어진다
- $l, r(1 \leq l, r \leq N)$이 주어졌을 때 $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$의 값을 Q번 구하자
- $N, Q \leq 100\,000$
- 구간에 대한 전처리를 처음부터 누적하는 형태로 한다면?
  - $S_1 = A_1$
  - $S_2 = A_1 + A_2$
  - $S_3 = A_1 + A_2 + A_3$
  - $S_i = A_1 + A_2 + A_3 + \cdots + A_i$
- $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$은 $S_r - S_{l-1}$로 나타낼 수 있다

구간 합 구하기 4 BOJ 11659

- 길이 N인 배열이 주어진다
- $l, r (1 \leq l, r \leq N)$이 주어졌을 때 $A_l + A_{l+1} + \cdots + A_{r-1} + A_r$의 값을 Q번 구하자
- $N, Q \leq 100\,000$
- 구간에 대한 전처리를 처음부터 누적하는 형태로 한다면?
  - $S_1 = A_1$
  - $S_2 = A_1 + A_2$
  - $S_3 = A_1 + A_2 + A_3$
  - $S_i = A_1 + A_2 + A_3 + \cdots + A_i$
- 전처리는 $O(N)$, 쿼리 하나 당 $O(1)$로 문제를 해결 가능하다
  - $O(N + Q)$

구간 합 구하기 4 BOJ 11659

```
for (int i = 1; i <= N; i++)
    cin >> A[i], S[i] = S[i - 1] + A[i];
while (Q--) {
    int l, r;
    cin >> l >> r;
    cout << S[r] - S[l - 1] << '\n';
}
```

구간 합 구하기 5 BOJ 11660

- 2차원 배열에서 누적합을 진행해보자
  - (x_1, y_1) 부터 (x_2, y_2) 까지의 합을 출력

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

구간 합 구하기 5 BOJ 11660

- 2차원 배열에서 누적합을 진행해보자
  - (x_1, y_1) 부터 (x_2, y_2) 까지의 합을 출력

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

1	3	6	10
3	8	15	24
6	15	27	42
10	24	42	64

구간 합 구하기 5 BOJ 11660

- 2차원 배열에서 누적합을 진행해보자
  - (x_1, y_1) 부터 (x_2, y_2) 까지의 합을 출력

1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

1	3	6	10
3	8	15	24
6	15	27	42
10	24	42	64

Prefix Sum + a

- 누적 합은 다른 여러 가지 알고리즘과 같이 쓰인다
  - 투 포인터, 이분 탐색, ...

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 수열의 원소는 자연수로 주어진다

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 수열의 원소는 자연수로 주어진다
- Bruteforce: $O(N^2)$

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 수열의 원소는 자연수로 주어진다
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 5$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 6$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 9$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 14$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 24, ans = 5$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 19, ans = 4$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 18, ans = 3$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 15, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 17, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 7, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 11, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 20, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 13, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 15, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 19, ans = 2$

5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

부분합 BOJ 1806

- 연속된 수들의 부분합 중 합이 S 이상이 되는 것 중, 가장 짧은 것의 길이를 구해보자
- 투포인터를 이용하면?
  - 누적합에서 오른쪽 범위를 뒤로 옮기면 항상 합이 증가함
  - 누적합에서 왼쪽 범위를 뒤로 옮기면 항상 합이 감소함
- $S = 15, psum = 10, ans = 2$

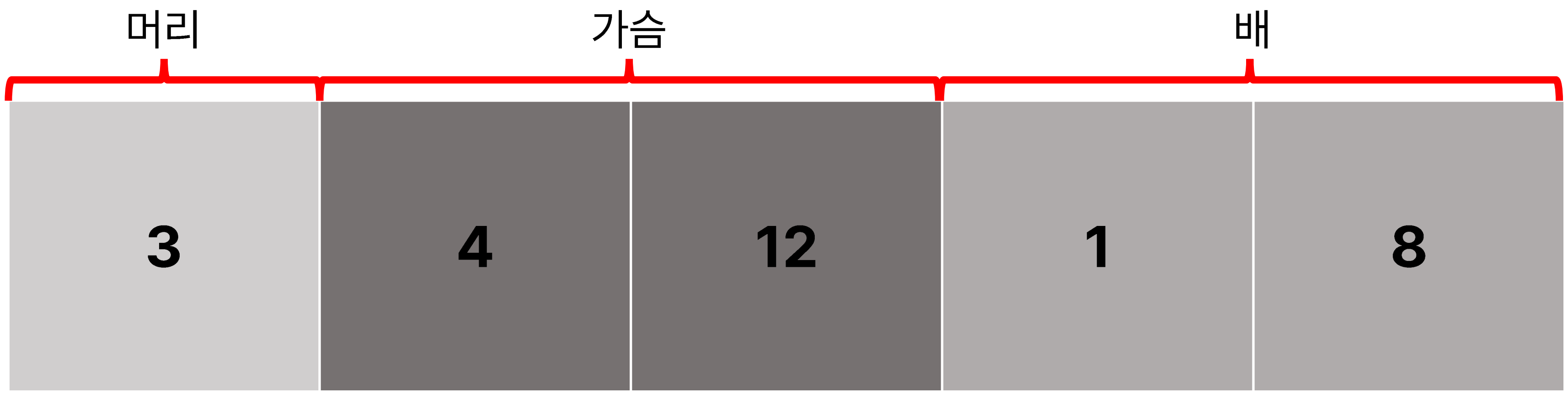
5	6	9	14	24	31	35	44	46	54
5	**1**	**3**	**5**	**10**	**7**	**4**	**9**	**2**	**8**

벌레컷 BOJ 27651

- 크기 N의 1차원 양의 정수 배열로 이루어진 자벌레가 있다
- 이를 머리, 가슴, 배로 구분 가능한 경우의 수를 출력하자
  - 배열상 머리는 왼쪽에, 가슴은 가운데에, 배는 오른쪽에 존재한다
  - 가슴이 배보다 크고 배가 머리보다 크며 크기는 구간의 합으로 정의한다
  - $\sum_{i=1}^{X} A_i < \sum_{i=Y+1}^{N} A_i < \sum_{X+1}^{Y} A_i \; (1 \le X < Y < N)$

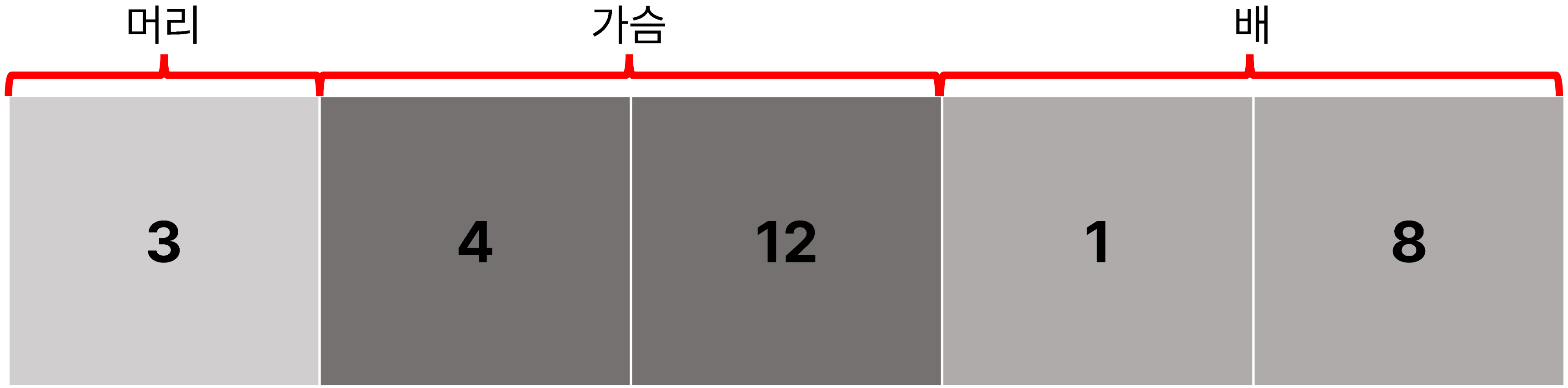
벌레컷 BOJ 27651

- 크기 N의 1차원 양의 정수 배열로 이루어진 자벌레가 있다
- 이를 머리, 가슴, 배로 구분 가능한 경우의 수를 출력하자
  - 배열상 머리는 왼쪽에, 가슴은 가운데에, 배는 오른쪽에 존재한다
  - 가슴이 배보다 크고 배가 머리보다 크며 크기는 구간의 합으로 정의한다
  - $\sum_{i=1}^{X} A_i < \sum_{i=Y+1}^{N} A_i < \sum_{X+1}^{Y} A_i \ (1 \le X < Y < N)$



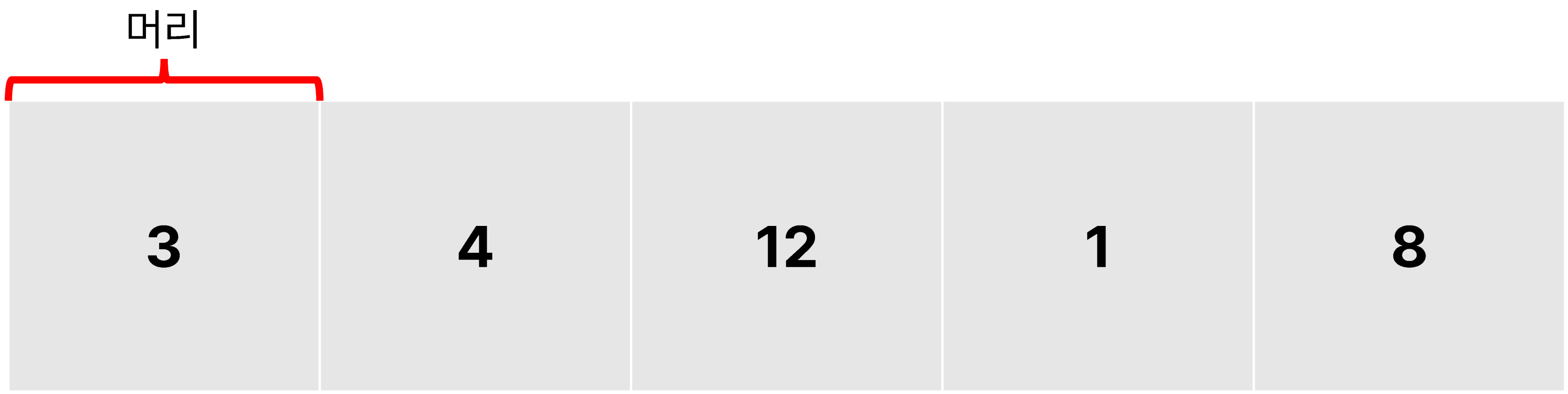
벌레컷 BOJ 27651

- 머리의 크기는 계속해서 증가한다
- 가슴과 배가 나뉘어지는 위치만 정하면 된다



벌레컷 BOJ 27651

- 머리의 크기는 계속해서 증가한다
- 가슴과 배가 나뉘어지는 위치만 정하면 된다
  - 나눌 수 있는 시작점과 끝점만 찾으면 됨



벌레컷 BOJ 27651

- 머리의 크기는 계속해서 증가한다
- 가슴과 배가 나뉘어지는 위치만 정하면 된다
  - 나눌 수 있는 시작점과 끝점만 찾으면 됨
  - 시작점은 가슴이 배보다 크기 시작한 위치
  - 끝점은 배가 머리보다 크기 시작한 위치

벌레컷 BOJ 27651

- 머리의 크기는 계속해서 증가한다
- 가슴과 배가 나뉘어지는 위치만 정하면 된다
  - 나눌 수 있는 시작점과 끝점만 찾으면 됨
  - 시작점은 가슴이 배보다 크기 시작한 위치
  - 끝점은 배가 머리보다 크기 시작한 위치

평범한 배낭 BOJ 12865

- 각각 무게가 W_i이고 가치가 V_i인 물건 N개가 있다
- 최대 K만큼의 무게를 넣을 수 있는 배낭을 들고 간다
- 배낭에 넣을 수 있는 물건들의 가치합의 최대값은?

평범한 배낭 BOJ 12865

- 간단하게 생각하면, 하나의 물건을 넣거나, 넣지 않거나 두 가지 상태로 볼 수 있음
  - 모두 고려하면 2^N으로 시간초과
- 적절한 점화식을 어떻게 세울까?

평범한 배낭 BOJ 12865

- $f(i, r)$: i번째 물건까지 고려했을 때, 현재 남은 배낭의 크기가 r일 때 가치합의 최댓값
- Base: $i > N$이거나 $r = 0$이면 더 이상 담을 수 없다
- Step: 물건을 고르거나 고르지 않을 수 있다
  - 고르지 않는 경우 배낭의 남은 크기는 변하지 않는다 $f(i + 1, r)$
  - 고르기 위해서는 $r \geq W_i$여야 한다
    - 가능한 경우 $f(i + 1, r - W_i) + V_i$가 답이 **될 수** 있다.

평범한 배낭 BOJ 12865

```
int f(int i, int r) {
    if (i > N || r < 0) return 0;
    int& ret = dp[i][r];
    if (ret != -1) return ret;
    ret = f(i + 1, r);
    if (r >= w[i])
        ret = max(ret, f(i + 1, r - w[i]) + v[i]);
    return ret;
}
```

평범한 배낭 BOJ 12865

```
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= K; j++) {
        if (j - w[i] >= 0)
            f[i][j] = max(f[i - 1][j], f[i - 1][j - w[i]] + v[i]);
        else f[i][j] = f[i - 1][j];
    }
}
```

연습 문제

10870	: 피보나치 수 5
2748	: 피보나치 수 2
1463	: 1로 만들기
30460	: 스위치
9251	: LCS
9252	: LCS2
11659	: 구간 합 구하기 4
11660	: 구간 합 구하기 5
1806	: 부분합
27651	: 벌레컷
12865	: 평범한 배낭

References

- https://github.com/justiceHui/Sunrin-SHARC
- https://github.com/KU-AlKon/study