이것이 취업을 위한 코딩 테스트다 with 파이썬

기타 알고리즘 & 개발형 코딩 테스트

나동빈(dongbinna@postech.ac.kr)

Pohang University of Science and Technology

기타 알고리즘

소수 (Prime Number)

- **소수**란 1보다 큰 자연수 중에서 1과 자기 자신을 제외한 자연수로는 나누어떨어지지 않는 자연수입니다.
  - 6은 1, 2, 3, 6으로 나누어떨어지므로 소수가 아닙니다.
  - 7은 1과 7을 제외하고는 나누어떨어지지 않으므로 소수입니다.
- 코딩 테스트에서는 어떠한 자연수가 소수인지 아닌지 판별해야 하는 문제가 자주 출제됩니다.

소수의 판별: 기본적인 알고리즘 (Python)

```python
# 소수 판별 함수(2이상의 자연수에 대하여)
def is_prime_number(x):
    # 2부터 (x - 1)까지의 모든 수를 확인하며
    for i in range(2, x):
        # x가 해당 수로 나누어떨어진다면
        if x % i == 0:
            return False # 소수가 아님
    return True # 소수임

print(is_prime_number(4))
print(is_prime_number(7))
```

실행 결과

```
False
True
```

소수의 판별: 기본적인 알고리즘 (C++)

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

using namespace std;

// 소수 판별 함수(2이상의 자연수에 대하여)
bool isPrimeNumber(int x) {
    // 2부터 (x - 1)까지의 모든 수를 확인하며
    for (int i = 2; i < x; i++) {
        // x가 해당 수로 나누어떨어진다면
        if (x % i == 0) {
            return false; // 소수가 아님
        }
    }
    return true; // 소수임
}

int main() {
    cout << isPrimeNumber(4) << '\n';
    cout << isPrimeNumber(7) << '\n';
}
```

실행 결과

```
0
1
```

소수의 판별: 기본적인 알고리즘 (Java)

```java
class Main {
    // 소수 판별 함수(2이상의 자연수에 대하여)
    public static boolean isPrimeNumber(int x) {
        // 2부터 (x - 1)까지의 모든 수를 확인하며
        for (int i = 2; i < x; i++) {
            // x가 해당 수로 나누어떨어진다면
            if (x % i == 0) {
                return false; // 소수가 아님
            }
        }
        return true; // 소수임
    }

    public static void main(String[] args) {
        System.out.println(isPrimeNumber(4));
        System.out.println(isPrimeNumber(7));
    }
}
```

실행 결과

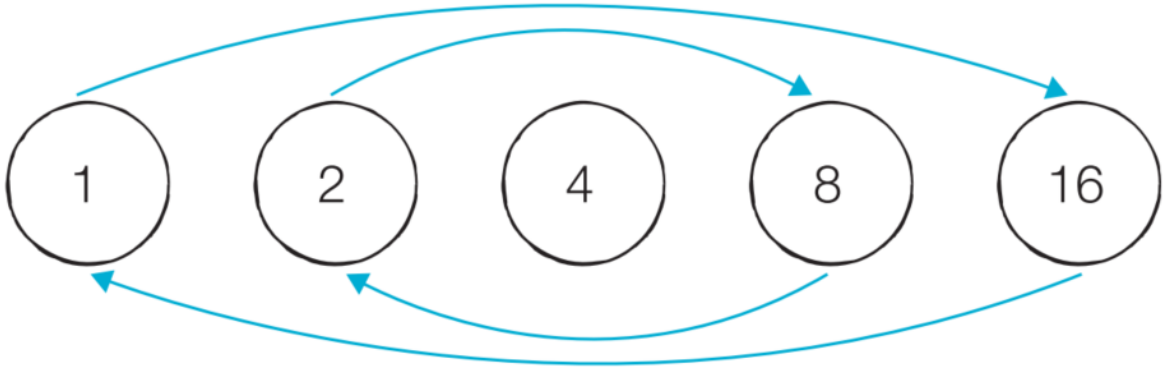
```
False
True
```

소수의 판별: 기본적인 알고리즘 성능 분석

- 2부터 $X-1$까지의 모든 자연수에 대하여 연산을 수행해야 합니다.
  - 모든 수를 하나씩 확인한다는 점에서 시간 복잡도는 $\boldsymbol{O(X)}$입니다.

약수의 성질

- **모든 약수가 가운데 약수를 기준으로 곱셈 연산에 대해 대칭**을 이루는 것을 알 수 있습니다.
  - 예를 들어 16의 약수는 1, 2, 4, 8, 16입니다.
  - 이때 2 X 8 = 16은 8 X 2 = 16과 대칭입니다.
- 따라서 우리는 특정한 자연수의 모든 약수를 찾을 때 가운데 약수(제곱근)까지만 확인하면 됩니다.
  - 예를 들어 16이 2로 나누어떨어진다는 것은 8로도 나누어떨어진다는 것을 의미합니다.



소수의 판별: 개선된 알고리즘 (Python)

```python
import math

# 소수 판별 함수 (2이상의 자연수에 대하여)
def is_prime_number(x):
    # 2부터 x의 제곱근까지의 모든 수를 확인하며
    for i in range(2, int(math.sqrt(x)) + 1):
        # x가 해당 수로 나누어떨어진다면
        if x % i == 0:
            return False # 소수가 아님
    return True # 소수임

print(is_prime_number(4))
print(is_prime_number(7))
```

실행 결과

```
False
True
```

소수의 판별: 개선된 알고리즘 (C++)

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

using namespace std;

// 소수 판별 함수(2이상의 자연수에 대하여)
bool isPrimeNumber(int x) {
    // 2부터 x의 제곱근까지의 모든 수를 확인하며
    for (int i = 2; i <= (int) sqrt(x); i++) {
        // x가 해당 수로 나누어떨어진다면
        if (x % i == 0) {
            return false; // 소수가 아님
        }
    }
    return true; // 소수임
}

int main() {
    cout << isPrimeNumber(4) << '\n';
    cout << isPrimeNumber(7) << '\n';
}
```

실행 결과

```
0
1
```

소수의 판별: 개선된 알고리즘 (Java)

```java
import java.util.*;

class Main {
    // 소수 판별 함수(2이상의 자연수에 대하여)
    public static boolean isPrimeNumber(int x) {
        // 2부터 x의 제곱근까지의 모든 수를 확인하며
        for (int i = 2; i <= Math.sqrt(x); i++) {
            // x가 해당 수로 나누어떨어진다면
            if (x % i == 0) {
                return false; // 소수가 아님
            }
        }
        return true; // 소수임
    }

    public static void main(String[] args) {
        System.out.println(isPrimeNumber(4));
        System.out.println(isPrimeNumber(7));
    }
}
```

실행 결과

```
False
True
```

소수의 판별: 개선된 알고리즘 성능 분석

- 2부터 X의 제곱근(소수점 이하 무시)까지의 모든 자연수에 대하여 연산을 수행해야 합니다.
  - 시간 복잡도는 $\boldsymbol{O(N^{\frac{1}{2}})}$입니다.

다수의 소수 판별

- 하나의 수에 대해서 소수인지 아닌지 판별하는 방법을 알아보았습니다.
- 하지만 특정한 수의 범위 안에 존재하는 모든 소수를 찾아야 할 때는 어떻게 할까요?
  - **에라토스테네스의 체 알고리즘**을 사용할 수 있습니다.

에라토스테네스의 체 알고리즘

- 다수의 자연수에 대하여 소수 여부를 판별할 때 사용하는 대표적인 알고리즘입니다.
- 에라토스테네스의 체는 N보다 작거나 같은 모든 소수를 찾을 때 사용할 수 있습니다.
- 에라토스테네스의 체 알고리즘의 **구체적인 동작 과정**은 다음과 같습니다.
  1. 2부터 N까지의 모든 자연수를 나열한다.
  2. 남은 수 중에서 아직 처리하지 않은 가장 작은 수 i를 찾는다.
  3. 남은 수 중에서 i의 배수를 모두 제거한다 (i는 제거하지 않는다).
  4. 더 이상 반복할 수 없을 때까지 2번과 3번의 과정을 반복한다.

에라토스테네스의 체 알고리즘 동작 예시

- **[초기 단계]** 2부터 26까지의 모든 자연수를 나열합니다. $(N = 26)$

2	3	4	5	6
7	8	9	10	11
12	13	14	15	16
17	18	19	20	21
22	23	24	25	26

에라토스테네스의 체 알고리즘 동작 예시

- **[Step 1]** 아직 처리하지 않은 가장 작은 수 2를 제외한 2의 배수는 모두 제거합니다.

2	3	4	5	6
7	8	9	10	11
12	13	14	15	16
17	18	19	20	21
22	23	24	25	26

에라토스테네스의 체 알고리즘 동작 예시

- **[Step 2]** 아직 처리하지 않은 가장 작은 수 3을 제외한 3의 배수는 모두 제거합니다.

2	3	4	5	6
7	8	9	10	11
12	13	14	15	16
17	18	19	20	21
22	23	24	25	26

에라토스테네스의 체 알고리즘 동작 예시

- **[Step 3]** 아직 처리하지 않은 가장 작은 수 5를 제외한 5의 배수는 모두 제거합니다.

2	3	4	5	6
7	8	9	10	11
12	13	14	15	16
17	18	19	20	21
22	23	24	25	26

에라토스테네스의 체 알고리즘 동작 예시

- **[Step 4]** 마찬가지의 과정을 반복했을 때 최종적인 결과는 다음과 같습니다.

2	3	4	5	6
7	8	9	10	11
12	13	14	15	16
17	18	19	20	21
22	23	24	25	26

에라토스테네스의 체 알고리즘 (Python)

```python
import math

n = 1000 # 2부터 1,000까지의 모든 수에 대하여 소수 판별
# 처음엔 모든 수가 소수(True)인 것으로 초기화(0과 1은 제외)
array = [True for i in range(n + 1)]

# 에라토스테네스의 체 알고리즘 수행
# 2부터 n의 제곱근까지의 모든 수를 확인하며
for i in range(2, int(math.sqrt(n)) + 1):
    if array[i] == True: # i가 소수인 경우(남은 수인 경우)
        # i를 제외한 i의 모든 배수를 지우기
        j = 2
        while i * j <= n:
            array[i * j] = False
            j += 1

# 모든 소수 출력
for i in range(2, n + 1):
    if array[i]:
        print(i, end=' ')
```

에라토스테네스의 체 알고리즘 (C++)

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

using namespace std;

int n = 1000; // 2부터 1,000까지의 모든 수에 대하여 소수 판별
vector<int> arr(n + 1, true); // 처음엔 모든 수가 소수(True)인 것으로 초기화(0과 1은 제외)

int main() {
    // 에라토스테네스의 체 알고리즘 수행
    for (int i = 2; i <= (int) sqrt(n); i++) { // 2부터 n의 제곱근까지의 모든 수를 확인하며
        if (arr[i] == true) { // i가 소수인 경우(남은 수인 경우)
            // i를 제외한 i의 모든 배수를 지우기
            int j = 2;
            while (i * j <= n) {
                arr[i * j] = false;
                j += 1;
            }
        }
    }
    for (int i = 2; i <= n; i++) { // 모든 소수 출력
        if (arr[i]) cout << i << ' ';
    }
}
```

에라토스테네스의 체 알고리즘 (Java)

```java
import java.util.*;

class Main {
    public static int n = 1000; // 2부터 1,000까지의 모든 수에 대하여 소수 판별
    public static boolean[] arr = new boolean[n + 1];

    public static void main(String[] args) {
        Arrays.fill(arr, true); // 처음엔 모든 수가 소수(True)인 것으로 초기화(0과 1은 제외)
        // 에라토스테네스의 체 알고리즘 수행
        for (int i = 2; i <= Math.sqrt(n); i++) {// 2부터 n의 제곱근까지의 모든 수를 확인하며
            if (arr[i] == true) { // i가 소수인 경우(남은 수인 경우)
                // i를 제외한 i의 모든 배수를 지우기
                int j = 2;
                while (i * j <= n) {
                    arr[i * j] = false;
                    j += 1;
                }
            }
        }
        for (int i = 2; i <= n; i++) { // 모든 소수 출력
            if (arr[i]) System.out.print(i + " ");
        }
    }
}
```

에라토스테네스의 체 알고리즘 성능 분석

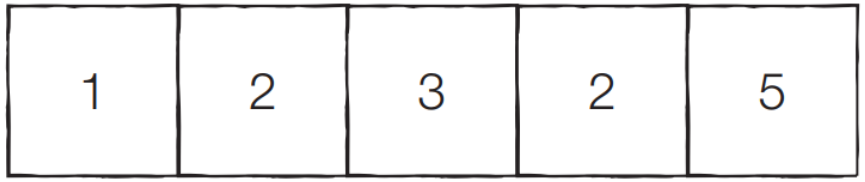
- 에라토스테네스의 체 알고리즘의 시간 복잡도는 사실상 선형 시간에 가까울 정도로 매우 빠릅니다.
  - 시간 복잡도는 $\boldsymbol{O(NloglogN)}$입니다.
- 에라토스테네스의 체 알고리즘은 다수의 소수를 찾아야 하는 문제에서 효과적으로 사용될 수 있습니다.
  - 하지만 각 자연수에 대한 소수 여부를 저장해야 하므로 **메모리가 많이 필요**합니다.
  - **10억**이 소수인지 아닌지 판별해야 할 때 에라토스테네스의 체를 사용할 수 있을까요?

투 포인터 (Two Pointers)

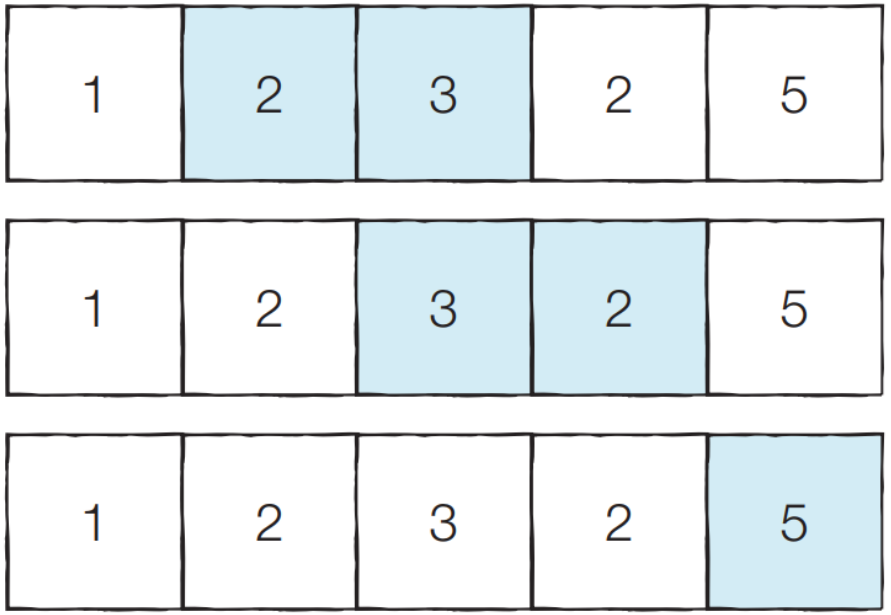
- **투 포인터 알고리즘**은 리스트에 순차적으로 접근해야 할 때 두 개의 점의 위치를 기록하면서 처리하는 알고리즘을 의미합니다.
- 흔히 2, 3, 4, 5, 6, 7번 학생을 지목해야 할 때 간단히 '2번부터 7번까지의 학생'이라고 부르곤 합니다.
- 리스트에 담긴 데이터에 순차적으로 접근해야 할 때는 **시작점**과 **끝점** 2개의 점으로 접근할 데이터의 범위를 표현할 수 있습니다.

특정한 합을 가지는 부분 연속 수열 찾기: 문제 설명

- N개의 자연수로 구성된 수열이 있습니다.
- **합이 M인 부분 연속 수열의 개수**를 구해보세요.
- 수행 시간 제한은 O(N)입니다.

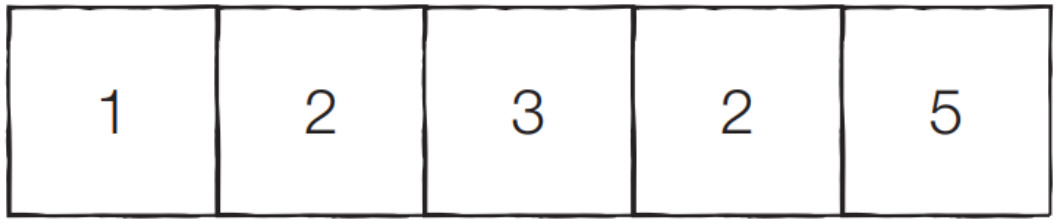


M = 5일 때



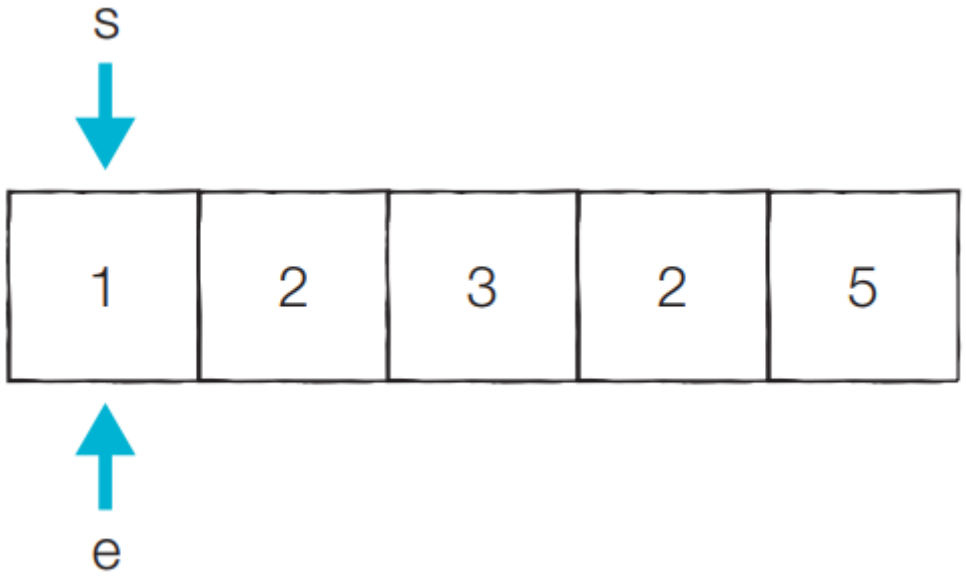
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- 투 포인터를 활용하여 다음과 같은 **알고리즘**으로 문제를 해결할 수 있습니다.
  1. 시작점(start)과 끝점(end)이 첫 번째 원소의 인덱스(0)를 가리키도록 한다.
  2. 현재 부분 합이 M과 같다면, 카운트한다.
  3. 현재 부분 합이 M보다 작다면, end를 1 증가시킨다.
  4. 현재 부분 합이 M보다 크거나 같다면, start를 1 증가시킨다.
  5. 모든 경우를 확인할 때까지 2번부터 4번까지의 과정을 반복한다.



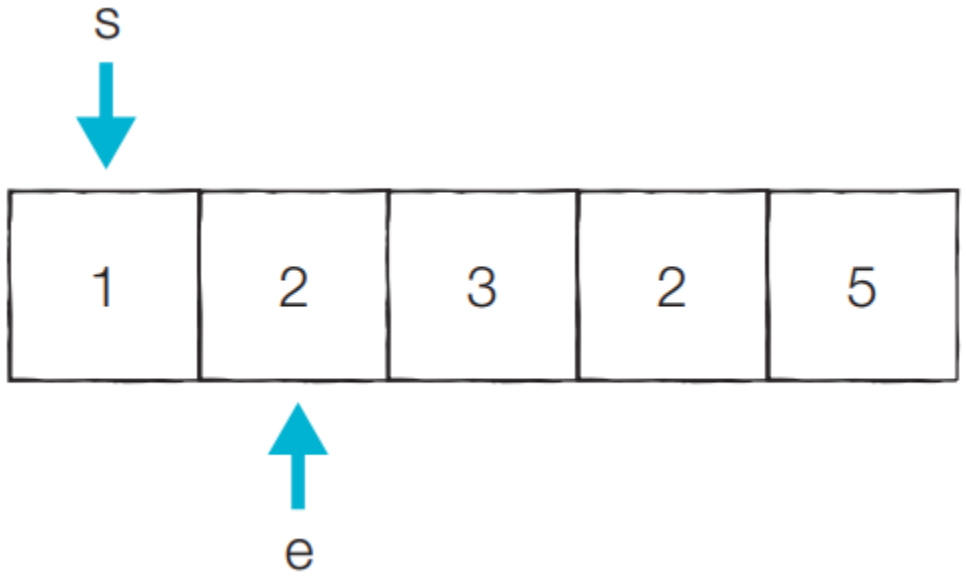
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- **[초기 단계]** 시작점과 끝점이 첫 번째 원소의 인덱스를 가리키도록 합니다.
  - 현재의 부분합은 1이므로 무시합니다.
  - **현재 카운트**: 0



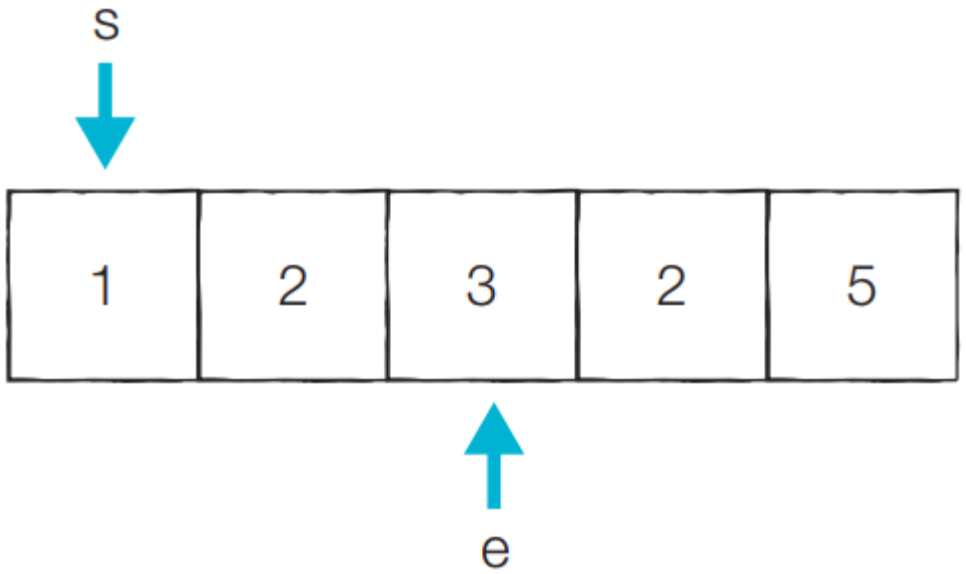
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 1] 이전 단계에서의 부분합이 1이었기 때문에 end를 1 증가시킵니다.
  - 현재의 부분합은 3이므로 무시합니다.
  - **현재 카운트**: 0



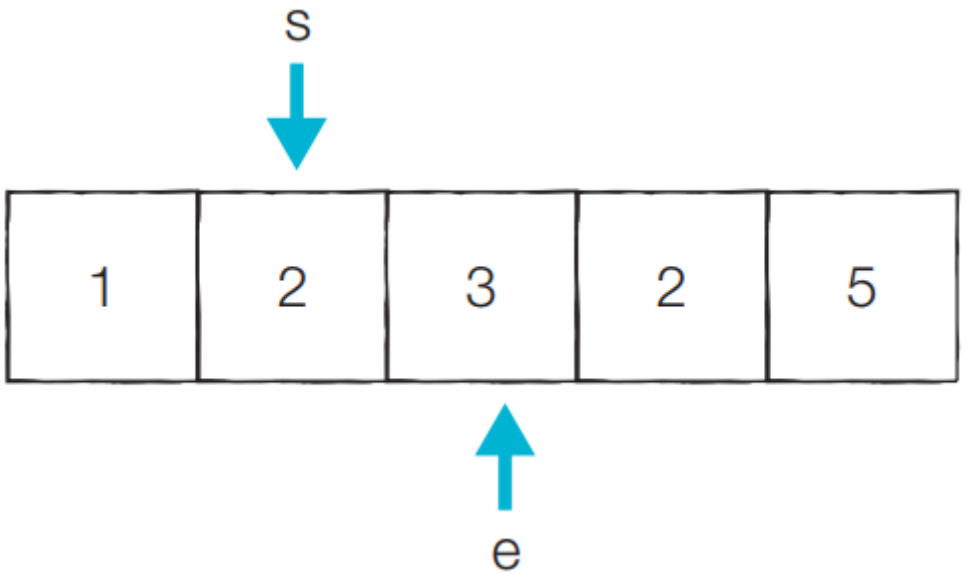
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 2] 이전 단계에서의 부분합이 3이었기 때문에 end를 1 증가시킵니다.
  - 현재의 부분합은 6이므로 무시합니다.
  - **현재 카운트**: 0



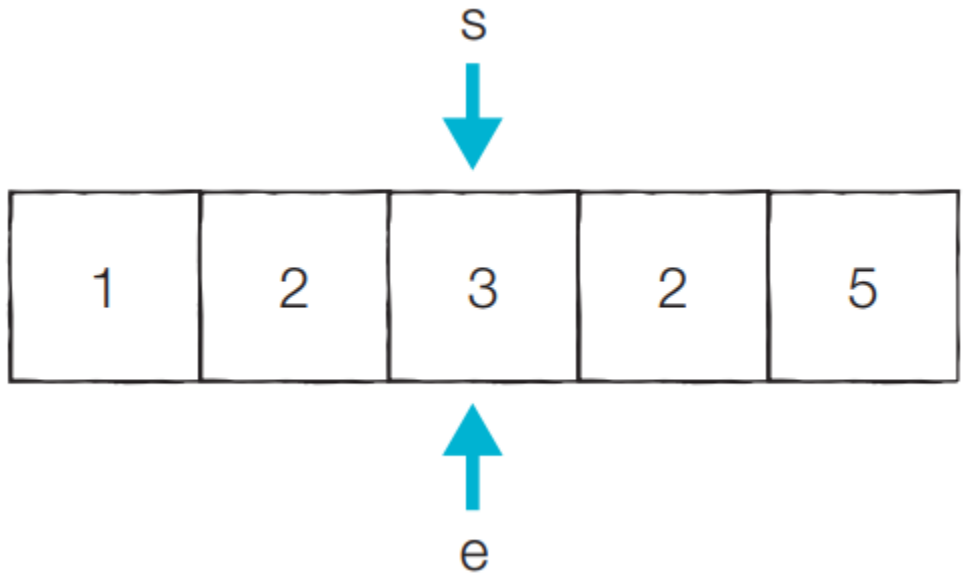
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 3] 이전 단계에서의 부분합이 6이었기 때문에 start를 1 증가시킵니다.
  - 현재의 부분합은 5이므로 카운트를 증가시킵니다.
  - **현재 카운트**: 1



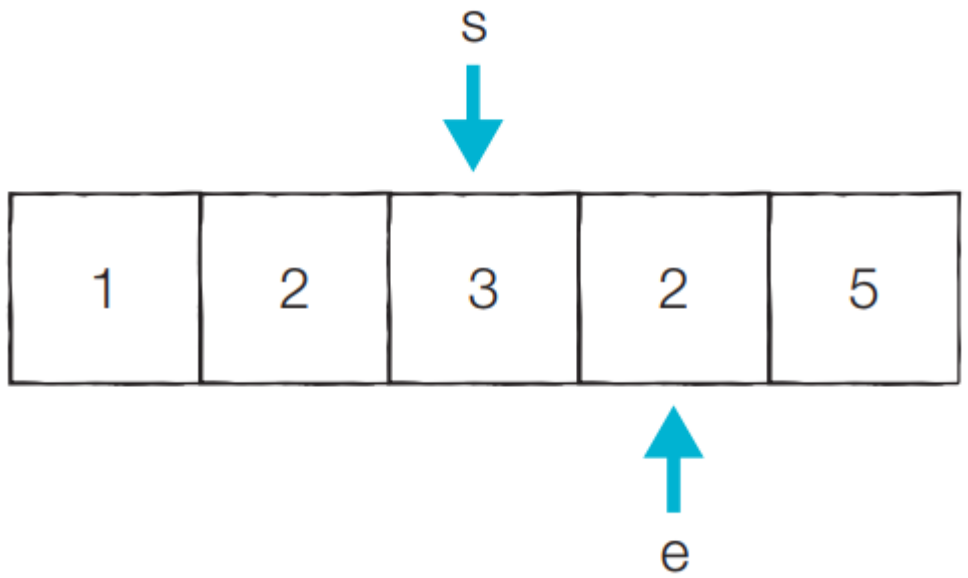
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- **[Step 4]** 이전 단계에서의 부분합이 5이었기 때문에 start를 1 증가시킵니다.
  - 현재의 부분합은 3이므로 무시합니다.
  - **현재 카운트**: 1



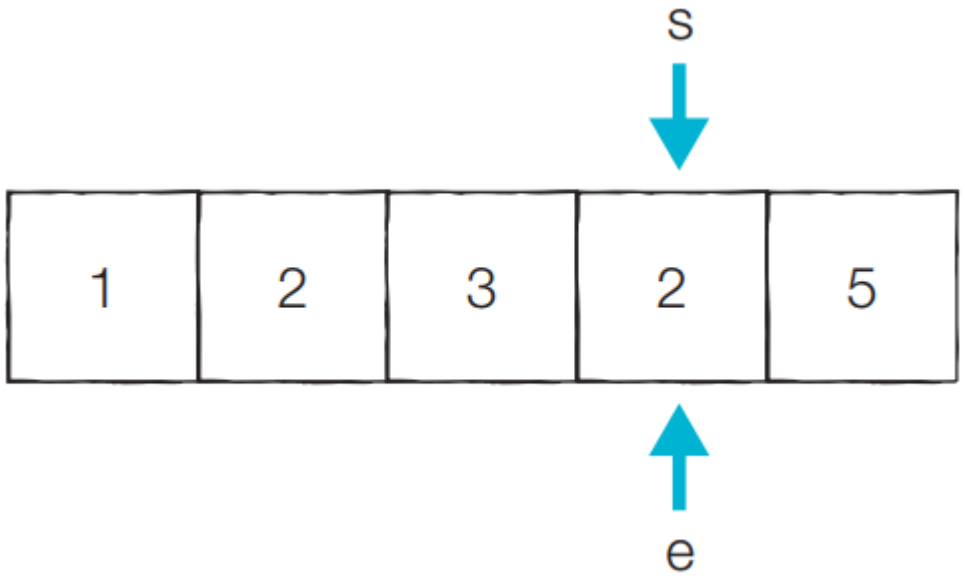
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 5] 이전 단계에서의 부분합이 3이었기 때문에 end를 1 증가시킵니다.
  - 현재의 부분합은 5이므로 카운트를 증가시킵니다.
  - **현재 카운트**: 2



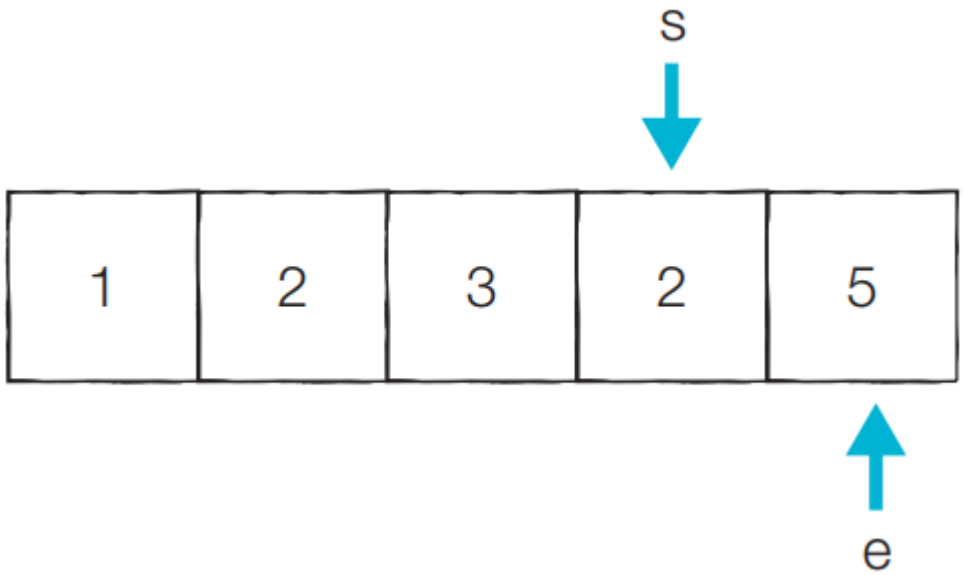
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 6] 이전 단계에서의 부분합이 5이었기 때문에 start를 1 증가시킵니다.
  - 현재의 부분합은 2이므로 무시합니다.
  - **현재 카운트**: 2



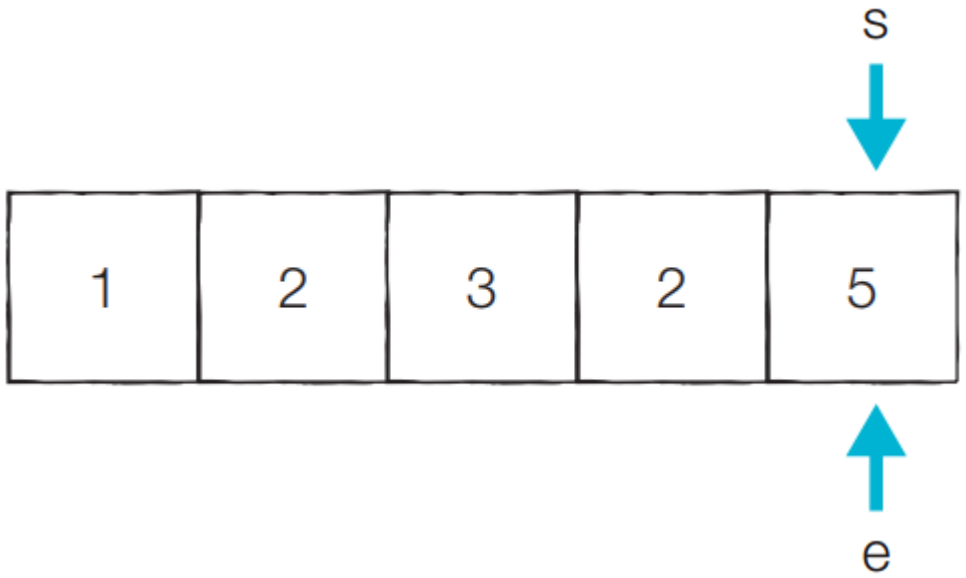
특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 7] 이전 단계에서의 부분합이 2였기 때문에 end를 1 증가시킵니다.
  - 현재의 부분합은 7이므로 무시합니다.
  - **현재 카운트**: 2



특정한 합을 가지는 부분 연속 수열 찾기: 문제 해결 아이디어

- $M = 5$
- [Step 8] 이전 단계에서의 부분합이 7이었기 때문에 start를 1 증가시킵니다.
  - 현재의 부분합은 5이므로 카운트를 증가시킵니다.
  - **현재 카운트**: 3



특정한 합을 가지는 부분 연속 수열 찾기: 코드 예시 (Python)

```python
n = 5 # 데이터의 개수 N
m = 5 # 찾고자 하는 부분합 M
data = [1, 2, 3, 2, 5] # 전체 수열

count = 0
interval_sum = 0
end = 0

# start를 차례대로 증가시키며 반복
for start in range(n):
    # end를 가능한 만큼 이동시키기
    while interval_sum < m and end < n:
        interval_sum += data[end]
        end += 1
    # 부분합이 m일 때 카운트 증가
    if interval_sum == m:
        count += 1
    interval_sum -= data[start]

print(count)
```

실행 결과

```
3
```

특정한 합을 가지는 부분 연속 수열 찾기: 코드 예시 (C++)

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

using namespace std;

int n = 5; // 데이터의 개수 N
int m = 5; // 찾고자 하는 부분합 M
int arr[] = {1, 2, 3, 2, 5}; // 전체 수열

int main() {
    int cnt = 0, intervalSum = 0, end = 0;
    for (int start = 0; start < n; start++) { // start를 차례대로 증가시키며 반복
        while (intervalSum < m && end < n) { // end를 가능한 만큼 이동시키기
            intervalSum += arr[end];
            end += 1;
        }
        if (intervalSum == m) { // 부분합이 m일 때 카운트 증가
            cnt += 1;
        }
        intervalSum -= arr[start];
    }
    cout << cnt << '\n';
}
```

실행 결과

```
3
```

특정한 합을 가지는 부분 연속 수열 찾기: 코드 예시 (Java)

```java
import java.util.*;

class Main {
    public static int n = 5; // 데이터의 개수 N
    public static int m = 5; // 찾고자 하는 부분합 M
    public static int[] arr = {1, 2, 3, 2, 5}; // 전체 수열

    public static void main(String[] args) {
        int cnt = 0, intervalSum = 0, end = 0;
        for (int start = 0; start < n; start++) { // start를 차례대로 증가시키며 반복
            while (intervalSum < m && end < n) { // end를 가능한 만큼 이동시키기
                intervalSum += arr[end];
                end += 1;
            }
            if (intervalSum == m) { // 부분합이 m일 때 카운트 증가
                cnt += 1;
            }
            intervalSum -= arr[start];
        }
        System.out.println(cnt);
    }
}
```

실행 결과

```
3
```

구간 합 (Interval Sum)

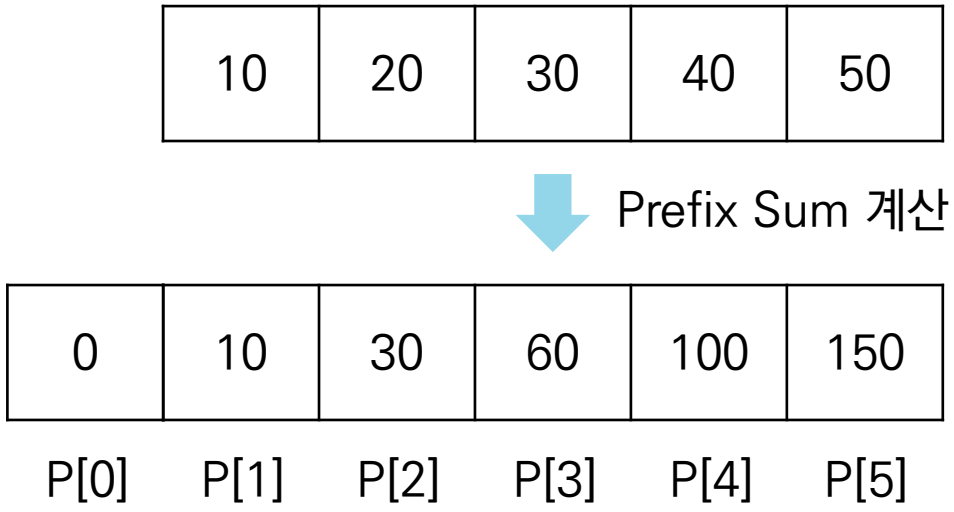
- **구간 합 문제**: 연속적으로 나열된 N개의 수가 있을 때 특정 구간의 모든 수를 합한 값을 계산하는 문제
- 예를 들어 5개의 데이터로 구성된 수열 {10, 20, 30, 40, 50}이 있다고 가정합시다.
  - 두 번째 수부터 네 번째 수까지의 합은 20 + 30 + 40 = 90입니다.

구간 합 빠르게 계산하기: 문제 설명

- N개의 정수로 구성된 수열이 있습니다.
- M개의 쿼리(Query) 정보가 주어집니다.
  - 각 쿼리는 $Left$와 $Right$으로 구성됩니다.
  - 각 쿼리에 대하여 $[Left, Right]$ 구간에 포함된 데이터들의 합을 출력해야 합니다.
- 수행 시간 제한은 $\boldsymbol{O(N+M)}$입니다.

구간 합 빠르게 계산하기: 문제 해결 아이디어

- **접두사 합(Prefix Sum)**: 배열의 맨 앞부터 특정 위치까지의 합을 미리 구해 놓은 것
- 접두사 합을 활용한 **알고리즘**은 다음과 같습니다.
  - N개의 수 위치 각각에 대하여 접두사 합을 계산하여 P에 저장합니다.
  - 매 M개의 쿼리 정보를 확인할 때 구간 합은 $P[Right] - P[Left - 1]$입니다.



1) $Left$ = 1, $Right$ = 3 ➡ P[3] − P[0] = **60**

2) $Left$ = 2, $Right$ = 5 ➡ P[5] − P[1] = **140**

...

M) $Left$ = 3, $Right$ = 4 ➡ P[4] − P[2] = **70**

구간 합 빠르게 계산하기: 코드 예시 (Python)

```python
# 데이터의 개수 N과 데이터 입력받기
n = 5
data = [10, 20, 30, 40, 50]

# 접두사 합(Prefix Sum) 배열 계산
sum_value = 0
prefix_sum = [0]
for i in data:
    sum_value += i
    prefix_sum.append(sum_value)

# 구간 합 계산(세 번째 수부터 네 번째 수까지)
left = 3
right = 4
print(prefix_sum[right] - prefix_sum[left - 1])
```

실행 결과

```
70
```

구간 합 빠르게 계산하기: 코드 예시 (C++)

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

using namespace std;

int n = 5; // 데이터의 개수 N과 데이터 입력받기
int arr[] = {10, 20, 30, 40, 50};
int prefixSum[6];

int main() {
    // 접두사 합(Prefix Sum) 배열 계산
    int sumValue = 0;

    for (int i = 0; i < n; i++) {
        sumValue += arr[i];
        prefixSum[i + 1] = sumValue;
    }

    // 구간 합 계산(세 번째 수부터 네 번째 수까지)
    int left = 3;
    int right = 4;
    cout << prefixSum[right] - prefixSum[left - 1] << '\n';
}
```

실행 결과

```
70
```

구간 합 빠르게 계산하기: 코드 예시 (Java)

```java
import java.util.*;

class Main {
    public static int n = 5; // 데이터의 개수 N과 데이터 입력받기
    public static int arr[] = {10, 20, 30, 40, 50};
    public static int[] prefixSum = new int[6];

    public static void main(String[] args) {
        // 접두사 합(Prefix Sum) 배열 계산
        int sumValue = 0;

        for (int i = 0; i < n; i++) {
            sumValue += arr[i];
            prefixSum[i + 1] = sumValue;
        }

        // 구간 합 계산(세 번째 수부터 네 번째 수까지)
        int left = 3;
        int right = 4;
        System.out.println(prefixSum[right] - prefixSum[left - 1]);
    }
}
```

실행 결과

```
70
```

개발형 코딩 테스트

개발형 코딩 테스트

- **정해진 목적에 따라서 동작하는 완성된 프로그램을 개발**하는 것을 요구하는 코딩 테스트 유형입니다.

알고리즘 코딩 테스트		개발형 코딩 테스트

• 요구 사항에 맞는 하나의 모듈 개발
  • 시간 복잡도 분석
  • 공간 복잡도 분석
	⟷	• 완성도 높은 하나의 프로그램을 개발
		• 모듈을 적절히 조합하는 능력 요구

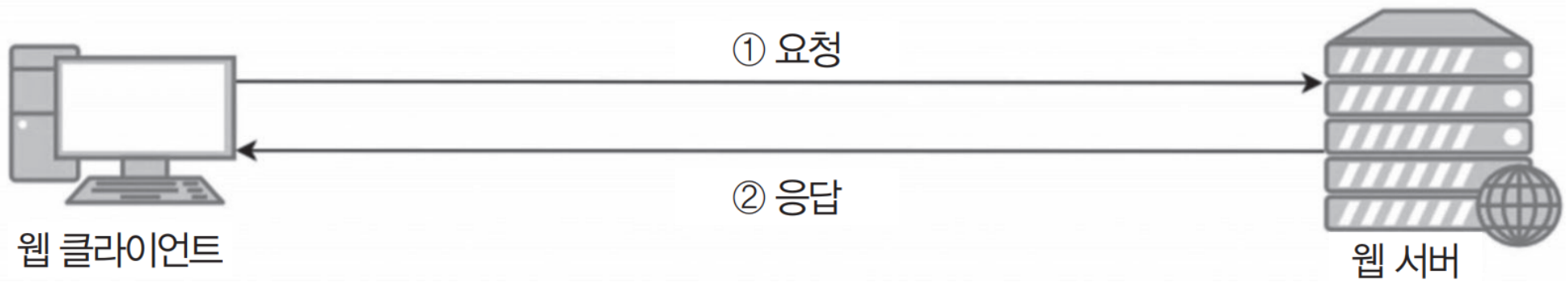
- 일부 기업은 해커톤을 통해 채용을 진행합니다.
  - **해커톤(Hackathon)**이란 단기간에 아이디어를 제품화하는 프로젝트 이벤트입니다.
  - 대개 1~2일 정도 진행되며 다수의 해커톤이 대회 형식을 빌려 해커톤이 끝나면 만든 프로그램을 시연하고 발표한 다음 채점을 진행합니다.

개발형 코딩 테스트

- 개발형 코딩 테스트는 분야에 따라 상세 요구사항이 다를 수 있습니다.
  - **예시 1)** 모바일 클라이언트 개발: 안드로이드, iOS 앱 개발
  - **예시 2)** 웹 서버 개발: 스프링(Spring), 장고(Django) 등의 서버 개발 프레임워크 활용
- 하지만 분야에 상관없이 꼭 알아야 하는 개념과 도구에 대해서 학습할 필요가 있습니다.
  - 서버, 클라이언트, JSON, REST API, …

서버와 클라이언트

- 클라이언트가 요청(Request)을 보내면 서버가 응답(Response)합니다.

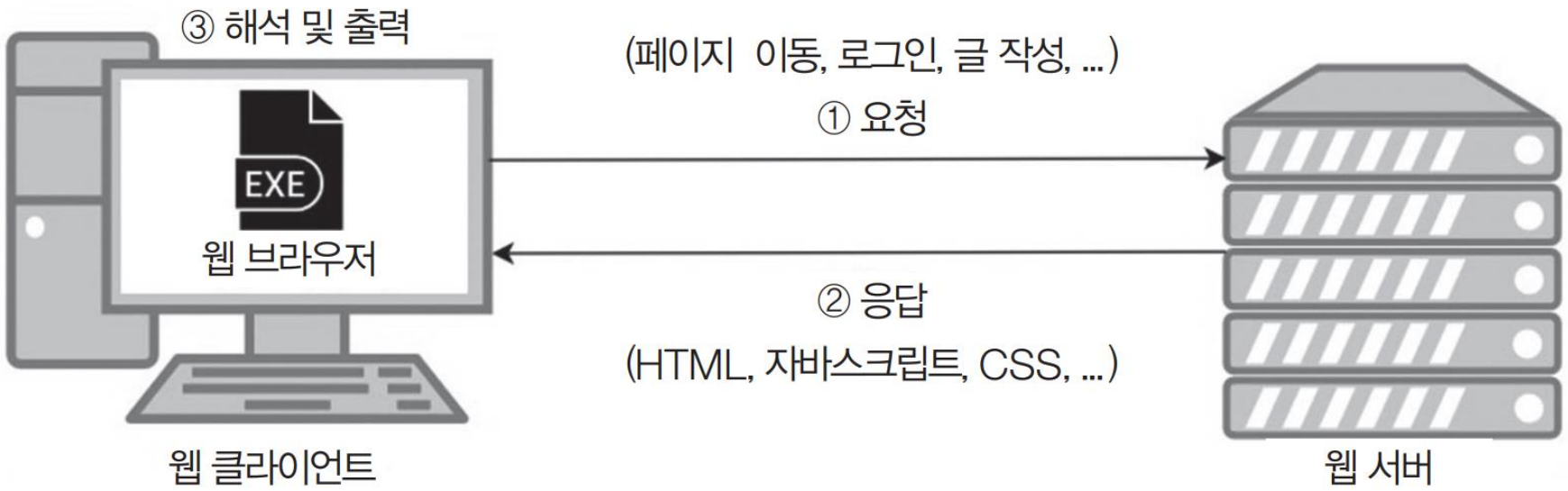


- PC
- 노트북
- 스마트 폰

- 워크스테이션

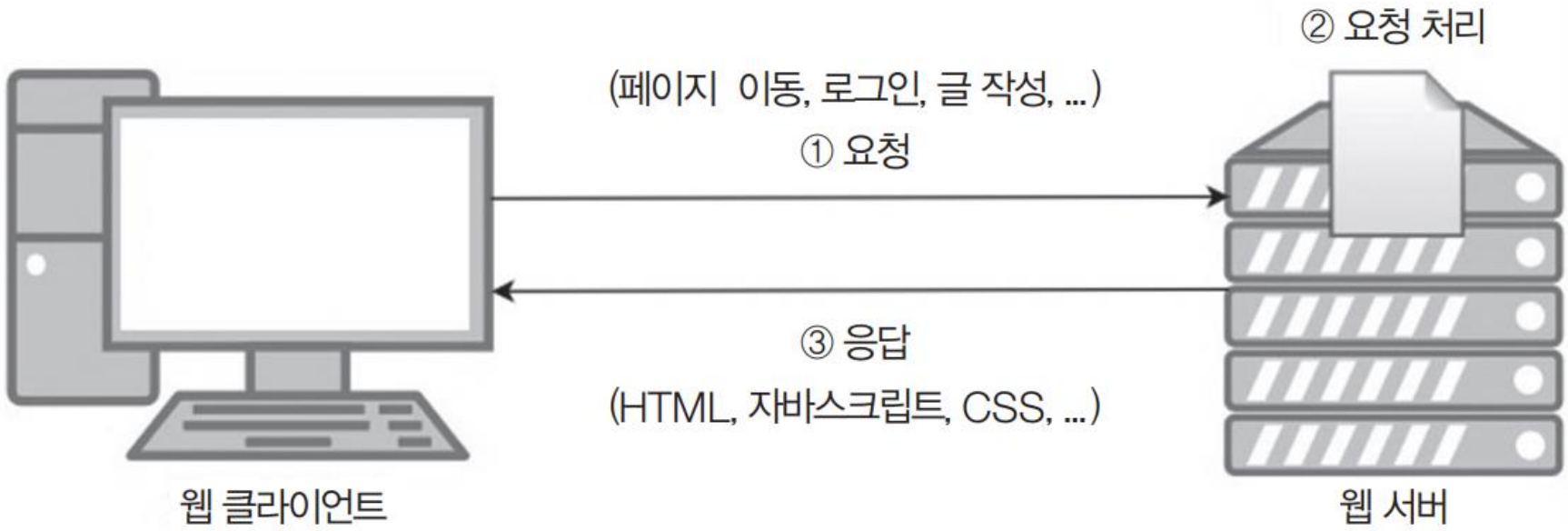
클라이언트(Client) = 고객

- 서버로 요청(Request)을 보내고 응답(Response)이 도착할 때까지 기다립니다.
- 서버로부터 응답을 받은 뒤에는 서버의 응답을 화면에 출력합니다.
  - **예시 1)** 웹 브라우저: 서버로부터 받은 HTML, CSS 코드를 화면에 적절한 형태로 출력합니다.
  - **예시 2)** 게임 앱: 서버로부터 받은 경험치, 친구 귓속말 정보 등을 화면에 적절한 형태로 출력합니다.



서버(Server) = 서비스 제공자

- 클라이언트로부터 받은 요청(Request)을 처리해 응답(Response)을 전송합니다.
  - **예시)** 웹 서버: 로그인 요청을 받아 아이디와 비밀번호가 정확한지 검사하고 그 결과를 응답합니다.



HTTP 개요

- **HTTP(HyperText Transfer Protocol)**는 웹상에서 데이터를 주고받기 위한 프로토콜을 의미합니다.
  - 보통은 웹 문서(HTML 파일)를 주고받는 데 사용됩니다.
  - 모바일 앱 및 게임 개발 등에서 특정 형식의 데이터를 주고받는 용도로도 사용됩니다.
- 클라이언트는 요청의 목적에 따라서 적절한 HTTP 메서드를 이용해 통신을 진행합니다.
  - 대표적인 **HTTP 메서드**는 다음과 같습니다.

HTTP 메서드	설명	사용 예시
GET	특정한 데이터의 조회를 요청한다.	특정 페이지 접속, 정보 검색
POST	특정한 데이터의 생성을 요청한다.	회원가입, 글쓰기
PUT	특정한 데이터의 수정을 요청한다.	회원 정보 수정
DELETE	특정한 데이터의 삭제를 요청한다.	회원 정보 삭제

파이썬 웹 요청 예제: GET 방식

```
import requests

target = "http://google.com"
response = requests.get(url=target)
print(response.text)
```

실행 결과

```
<!doctype html><html itemscope=" "...
```

개발형 코딩 테스트 준비하기

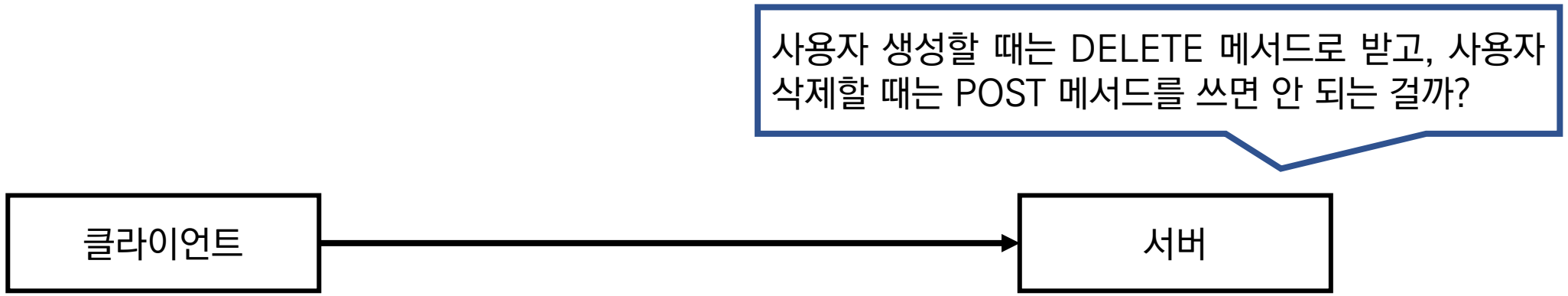
- 다음은 2020 카카오 2차 코딩 테스트 안내문에 쓰여 있던 문장입니다.

> “오프라인 코딩 테스트에서는 JSON format의 데이터를 응답하는 REST API를 활용해야 하니, REST API 호출과 JSON format 데이터를 파싱해 활용할 수 있는 parser 코드를 미리 준비해 오시기 바랍니다.”

- 개발형 코딩 테스트의 핵심 키워드: **REST API, JSON**

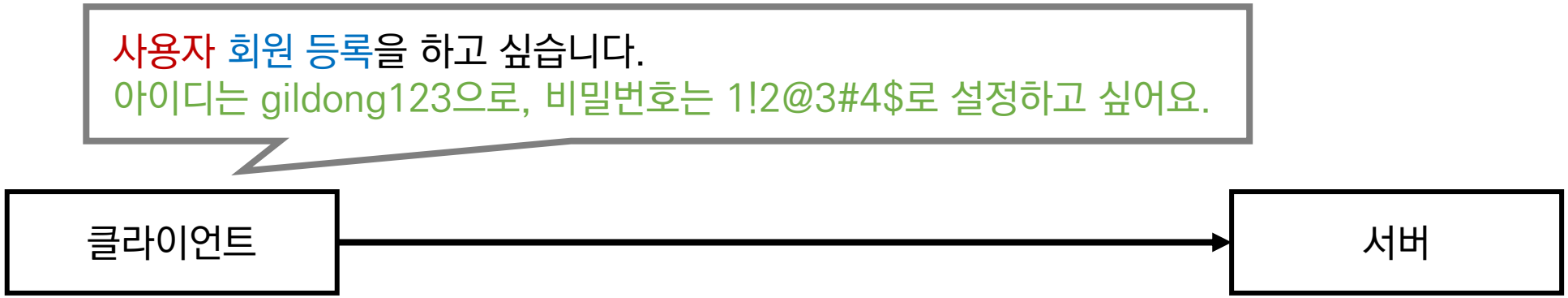
REST의 등장 배경

- **HTTP**는 GET, POST, PUT, DELETE 등의 다양한 HTTP 메서드를 지원합니다.
  - 실제로는 서버가 각 메서드의 기본 설명을 따르지 않아도 프로그램을 개발할 수 있습니다.
  - 하지만 저마다 다른 방식으로 개발하면 문제가 될 수 있어 기준이 되는 아키텍처가 필요합니다.



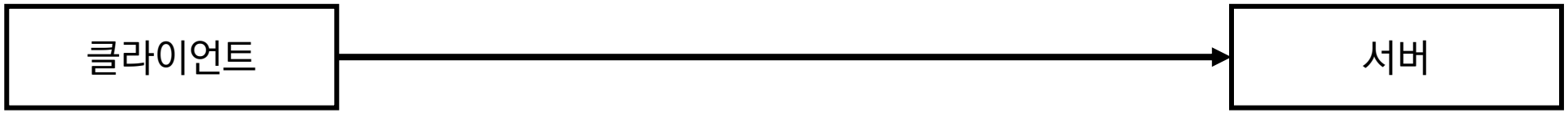
REST 개요

- **REST (Representational State Transfer)**는 각 자원(Resource)에 대하여 자원의 상태에 대한 정보를 주고받는 개발 방식을 의미합니다.
- REST의 **구성 요소**
  - 자원(Resource): URI를 이용
  - 행위(Verb): HTTP 메서드를 이용
  - 표현(Representations): 페이로드를 이용



REST 개요

- REST의 **구성 요소**
  - 자원(Resource): URI를 이용
  - 행위(Verb): HTTP 메서드를 이용
  - 표현(Representations): 페이로드를 이용




```
URI: https://www.example.com/users
HTTP Method: POST
Payload: {"id": "gildong123", "password": "1!2@3#4$"}
```

HTTP 패킷 정보

REST API란?

- **API (Application Programming Interface)**: 프로그램이 상호작용하기 위한 인터페이스를 의미합니다.
- **REST API**: REST 아키텍처를 따르는 API를 의미합니다.
- **REST API 호출**: REST 방식을 따르고 있는 서버에 특정한 요청을 전송하는 것을 의미합니다.

JSON

- **JSON (JavaScript Object Notation)**: 데이터를 주고받는 데 사용하는 경량의 데이터 형식
- JSON 형식을 따르는 데이터 예시는 다음과 같습니다.

```
{
    "id": "gildong123",
    "password": "1!2@3#4$",
    "age": 30,
    "hobby": ["football", "programming"]
}
```

- JSON 데이터는 **키와 값의 쌍**으로 이루어진 데이터 객체를 저장합니다.

JSON 객체 사용 예제

```python
import json

# 사전 자료형(dict) 데이터 선언
user = {
    "id": "gildong",
    "password": "1!2@3#4$",
    "age": 30,
    "hobby": ["football", "programming"]
}

# 파이썬 변수를 JSON 객체로 변환
json_data = json.dumps(user, indent=4)
print(json_data)
```

실행 결과

```
{
    "id": "gildong",
    "password": "1!2@3#4$",
    "age": 30,
    "hobby": [
        "football",
        "programming"
    ]
}
```

JSON 객체 파일 저장 예제

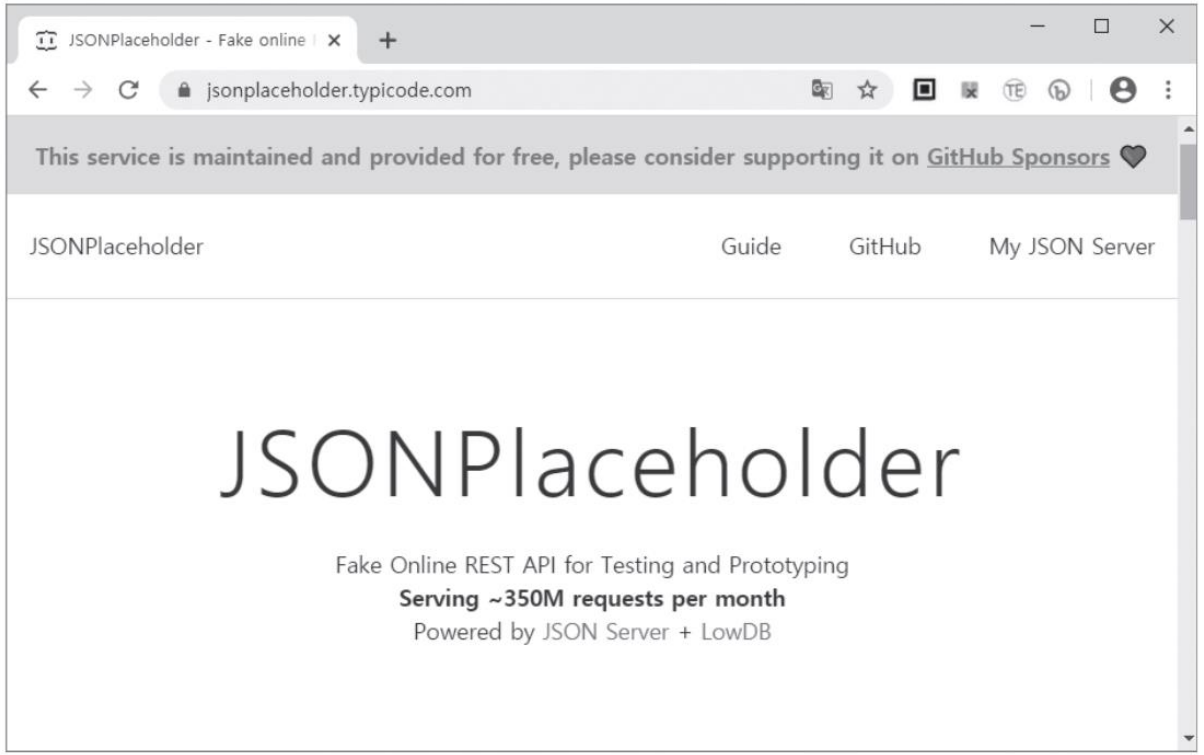
```
import json

# 사전 자료형(dict) 데이터 선언
user = {
    "id": "gildong",
    "password": "1!2@3#4$",
    "age": 30,
    "hobby": ["football", "programming"]
}

# JSON 데이터로 변환하여 파일로 저장
with open("user.json", "w", encoding="utf-8") as file:
    json_data = json.dump(user, file, indent=4)
```

REST API 연습용 서비스

- **목킹(Mocking)**이란 어떠한 기능이 있는 것처럼 흉내내어 구현한 것을 의미합니다.
- 가상의 REST API 제공 서비스: https://jsonplaceholder.typicode.com/



REST API 호출 실습해보기

- **API 호출 경로**: https://jsonplaceholder.typicode.com/users/1
- **HTTP 메서드**: GET

```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  (생략)
}
```

REST API 호출 실습해보기

- **API 호출 경로**: https://jsonplaceholder.typicode.com/users
- **HTTP 메서드**: GET

```
[
    {
        "id": 1,
        "name": "Leanne Graham",
        "username": "Bret",
        "email": "Sincere@april.biz",
        (생략)
    },
    {
        "id": 2,
        "name": "Ervin Howell",
        "username": "Antonette",
        "email": "Shanna@melissa.tv",
        (생략)
    },
    (생략)
]
```

REST API를 호출하여 회원 정보를 처리하는 예제

```
import requests

# REST API 경로에 접속하여 응답(Response) 데이터 받아오기
target = "https://jsonplaceholder.typicode.com/users"
response = requests.get(url=target)

# 응답(Response) 데이터가 JSON 형식이므로 바로 파이썬 객체로 변환
data = response.json()

# 모든 사용자(user) 정보를 확인하며 이름 정보만 삽입
name_list = []
for user in data:
    name_list.append(user['name'])

print(name_list)
```