Sieve of Eratosthenes

▼ 에라토스테네스의 체_Sieve of Eratosthenes

- 소수 prime number를 찾아내는 알고리즘
  - 소수는 2 이상의 정수에서 1과 그 수 자체로만 나눌 수 있는 수
  - 소수는 나열되어 있는 구간이 불규칙하므로 임의로 찾기가 힘들다.

> 👉 소수는 2 이상의 정수 중에서 1과 그 수 자신 외에는 나눌수 없는 숫자.
> 10이하에서는 2,3,5,7이 소수에 해당한다.
> 소수 prime에서의 '소'는 합성되지 않은 소박한 숫자라는 뜻을 가진다.
> 모든 수의 소(근본)을 의미하기도 한다.

소수인지 아닌지를 구분하는 것은 의외로 어렵다.

> 👉 소수의 내용만으로 보면 어렵지 않아보이지만 사실은 의외로 아주 어렵다. 무엇이 어려운지도 바로 찾아내기 어렵다.
> 예로 3의 배수는 3,6,9,12,15 처럼 3개의 간격으로 나열된다. 따라서 1에서 100까지의 사이에 있는 3의 배수를 찾는 일은 간단하다.
> 하지만 소수는 규칙성이 없기에(규칙이 있다고 생각하는게 리만가설) 간격이 불규칙하고 랜덤하다. 즉 소수를 한번에 열거하기가 어렵다.

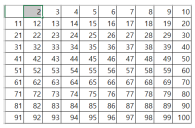
- 소수를 찾아내는 방법
  - 고대 그리스의 과학자인 에라토스테네스는 모든 숫자로 나눠서 소수를 찾는 방법을 개선하여 소수를 효율적으로 발견하는 방법을 알아냈다. 그의 이름을 따서 '에라토스테네스의 체'라고 부른다.

▼ Sieve of Eratosthenes

- 어떤 수 이하의 범위에 존재하는 모든 소수를 찾고 싶은 경우

> '그 수의 제곱근보다 작은 소수의 배수만 없애면 남은 소가 소수다'

라는 생각을 바탕으로 소수를 찾는 방법이다.



> 👉 예를 들어
> 1. 100 이하의 소수를 모두 찾아내려면 먼저 100의 제곱근 이라 소수를 선택한다.
> 2. 루트 100의 제곱근은 10이다. 즉 제곱하면 100이 되는 수는 10이다.
> 3. 10 이하에서 소수는 2,3,5,7, 네개이다.

> 👉 우선 2에서 100까지의 표에서 2로 나눌 수 있는 수를 2를 제외하고 나머지를 모두 삭제한다.

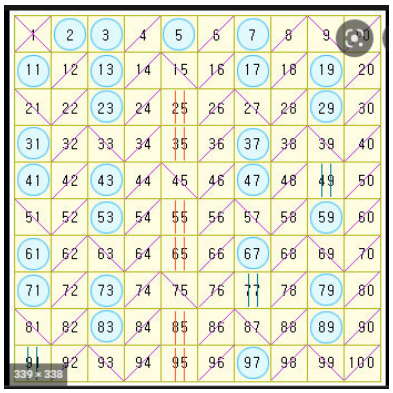
맨 처음에 소수인 2를 발견한 후 2의 배수를 모두 지운다.

	2	3		5		7		9	
11		13		15		17		19	
21		23		25		27		29	
31		33		35		37		39	
41		43		45		47		49	
51		53		55		57		59	
61		63		65		67		69	
71		73		75		77		79	
81		83		85		87		89	
91		93		95		97		99	

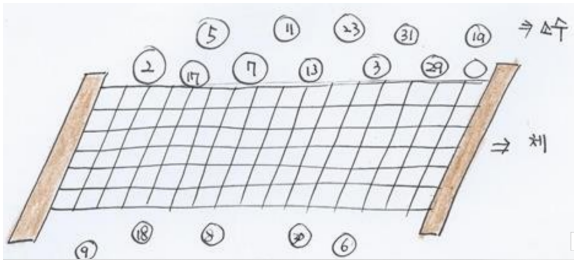
이런 방식으로 3보다 큰 수에 대해서도 n*n부터 지워주면 된다.

	2	3		5		7			
11		13				17		19	
		23		25				29	
31				35		37			
41		43				47		49	
		53		55				59	
61				65		67			
71		73				77		79	
		83		85				89	
91				95		97			

👉 같은 방식으로 5와 7을 모두 제거하면 다음과 같이 된다.



👉 이 방법을 사용하면 1에서 100까지의 모든 숫자를 해당 숫자보다 작은수로 나눌수 있는 지 하나하나 전부 순서대로 확인하는 것보다는 훨씬 더 빨리 소수를 찾을 수 있다.



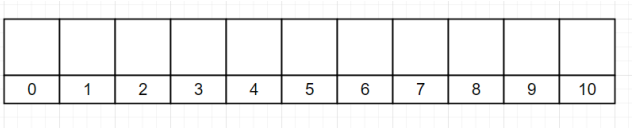
▼ Algorithm

- 에라토스테네스의 체는 크게 3개의 처리로 구성한다.
  1. 어떤 수 이하의 모든 정수 데이터 준비
  2. 어떤 수의 제곱근 보다 작은 소수의 배수들을 차례로 제거한다.
  3. 마지막까지 남은 수들을 출력한다.

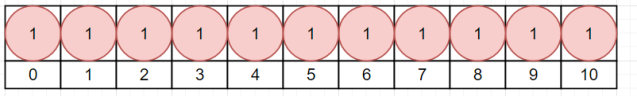
👉 우선 10이하의 소수를 구하는 경우로 생각해보자.
1. 10이하의 정수 데이터를 준비한다.
2. 10의 제곱근 약 3.16보다 작은 소수의 배수를 차례대로 제거한다.
3. 마지막까지 남은 수들을 출력한다.

▼ (1) 10 이하의 정수 데이터들을 준비한다.

- 먼저 체의 대상이 되는 10까지의 정수를 데이터로 준비하자. 여기에서는 11개의 요소를 가지는 정수형 배열을 준비한다. 배열의 이름은 arr로 정하고, 첨자는 0부터 시작하기 때문에 첨자를 10까지 동일하게 하기 위해 요소를 11개를 준비한다.

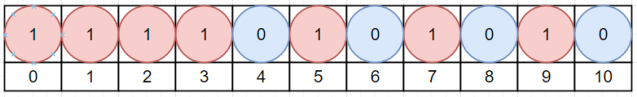


- 요소는 그 첨자가 소수인지의 여부를 판정하는 데이터를 넣는 것으로 사용하자. 즉, 소수가 아닌 것으로 판정된 첨자의 요소에 '소수가 아니다'라는 것을 나타내는 데이터를 넣자. 다시말해 소수의 가능성이 있는 경우에는 1을 대입하고, 소수가 아닌 경우에는 0을 대입하자.
  - 위의 개념 소개에서 사용한 소수가 아닌 수를 제거하는 방식을 0을 대입하는 방식으로 처리한다.
  - 따라서, 초기값을 1로 전부 대입해둔다. 그리고 소수가 아닌 것으로 판정된 수(첨자)의 요소에는 0을 대입한다.
  - 이러한 방식을 통해 소수의 배수를 모두 제거(0을 대입한 요소들)한 후 '1'이 남아 있는 요소들의 첨자는 모두 소수이고 0이 대입되어 있는 요소의 첨자는 '소수가 아니다'리고 구별할 수 있다.

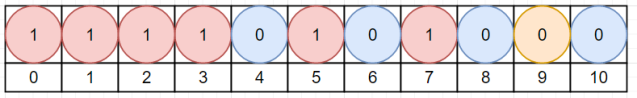


▼ (2) 10의 제곱근은 약 3.16이므로 제곱근의 이하의 소수 2와 3의 배수들을 모두 제거한다.

- 먼저 2의 배수들을 모두 지운다. 이때 지운다의 의미를 0을 대입한다는 표현으로 적용한다.

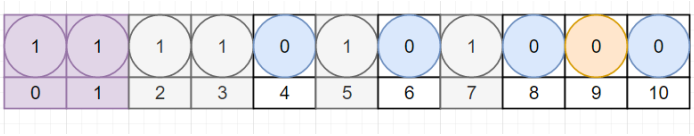


- 다음으로 3의 배수에 0을 대입한다.



▼ (3) 마지막까지 남은 수들을 출력한다.

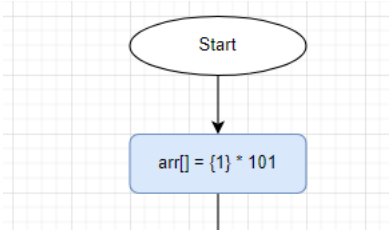
- 제곱근 이하의 소수들의 모든 배수를 제거(0을 대입)했다면 나머지는 모두 소수임으로 요소의 값이 '1'인 첨자는 소수이다.
- 첨자가 2이상이고 1이 들어 잇는 요소들의 첨자들만 뽑아낸다.

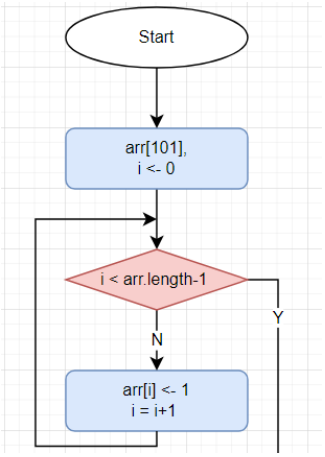


▼ Flow chart

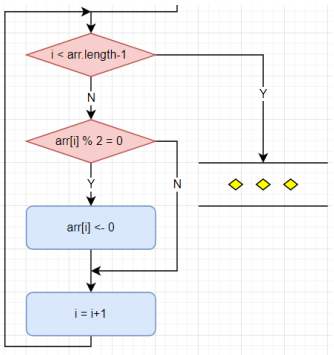
- 수를 늘려서 100 이하의 소수를 모두 구하는 경우로 생각해보자.

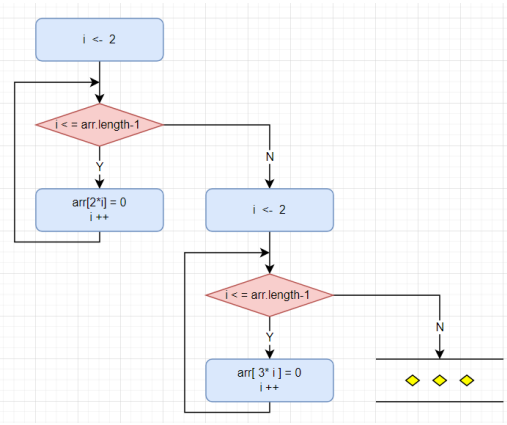
▼ (1) 100 이하의 정수 데이터들을 준비하고 1을 대입한다.





▼ (2) 100 제곱근인 10, 10 이하의 소수의 배수들을 모두 제거한다.

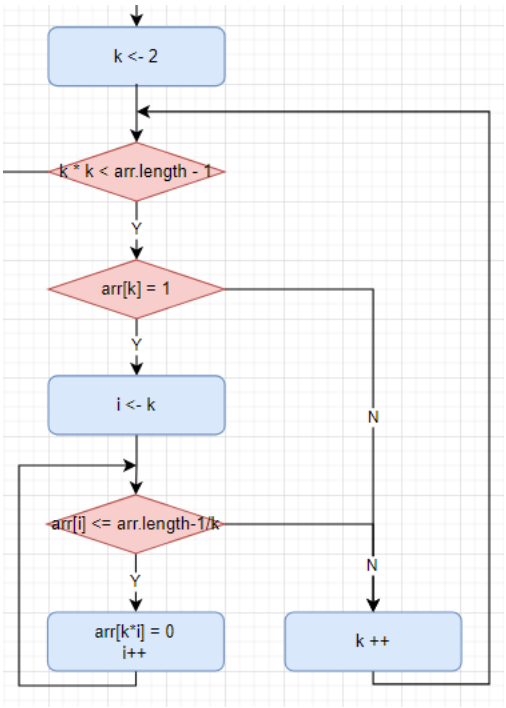




- K에 대입된 값이 소수인지의 여부에 따라 반복문으로 진입을 해야한다.
  - 따라서 arr[k] = 1이면 소수, arr[k]가 0이면 소수가 아니라는 것이다.

> 👉 변수 k는 하나씩 증가한다. 2인 경우에는 2의 배수들이 전부 0이되고, 3인 경우에는 3의 배수들을 전부 0이 되는 것을 반복한다. 그러나 4는 소수가 아니다.
> 3 다음의 소수는 0이므로 5의 배수를 제거해야한다. 따라서 k값이 소수인지를 판단하는 처리를 추가해야한다.
> 허나. 2와 3의 배수를 제거하는 처리를 한 뒤에는 이미 2와 3의 배수에는 모두 요소가 0이 되어 있다.
> 즉. 이 시점에는 첨자가 소수가 아닌 경우 0이 대입되어 있다.

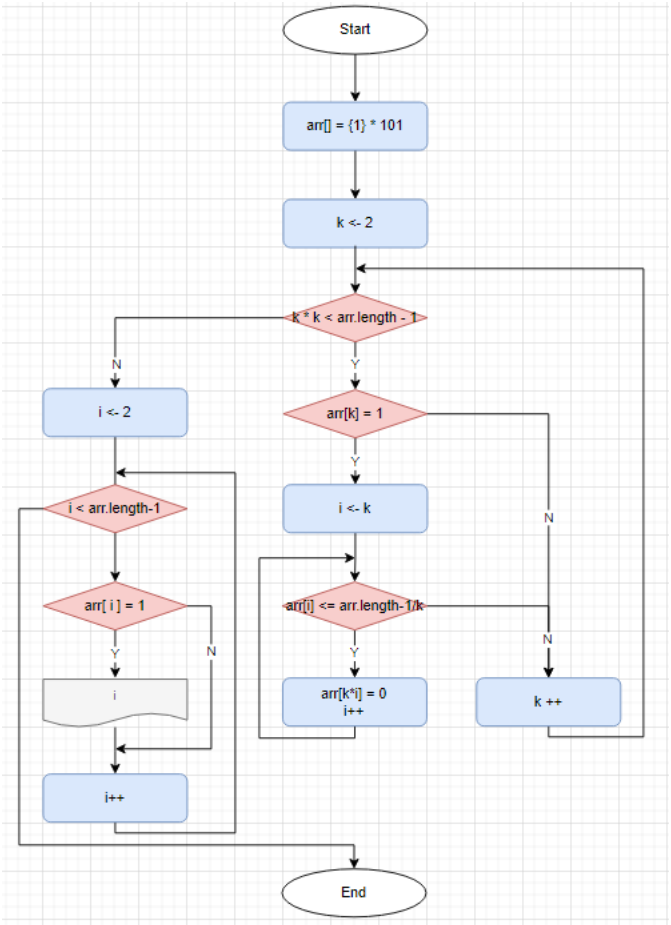
> 👉 arr[k] = 1 이 Yes인 경우 k는 소수이므로 k의 배수를 제거하는 반복처리가 실행된다.
> 이와 반대로 No의 경우는 k는 하나 늘려 k가 소수인지 판단하게 된다.
>
> 이를 과정을 통해 k=4인 경우 arr[4]는 이미 0이므로 배수를 제거하는 반복처리로 들어가지 않고 하나 증가 하게 된다.



허나... k는 무한이 증가하며 반복에 빠지게 된다.

- K의 무한루프 해결

👉 에라토스테네스의 체는 '어떤 숫자의 제곱근보다 작은 소수의 배수를 제거하면 남은 소가 소수이다.' 라는 이론이다. 따라서 이번 예제에서는 그 어떤 수가 100이므로 k는 100의 제곱근 즉 10 이하가 된다. 따라서 'k가 100의 제곱근 이하인가? ' 라는 판별식을 추가하여 반복을 마치고 소수를 출력하자.



▼ Java Code

```
import java.util.*;

public class SieveofEratosthenes {

  public static void main(String[] args) {
    // 101개의 배열을 생성하고 0으로 시작
    int[] arr = new int[101];
    System.out.println(Arrays.toString(arr));

//    for ( int i = 1; i < arr.length; i++) {
//      arr[i] = 1;
//    } //반복구조를 통해 arr를 1로 초기화

//    Arrays.fill(arr,1); //for문을 사용하지 않고 1로 초기화


    //소수를 제외하고 1을 대입
    //for문
    for (int k = 2; k*k <= arr.length-1; k++) {
      if(arr[k] == 0) {
        for (int i = k; i <= (arr.length-1)/k; i++) {
          if(arr[i] <= (arr.length-1)/k) {
          arr[k*i] = 1;
          }
        }
      }
    }

    //while문
//    int k = 2;
//    while(k*k < arr.length-1) {
//      if(arr[k] == 0) {
```

```
//        int i = k;
//        while(i <= (arr.length-1) / k) {
//          arr[k*i] = 1;
//          i++;
//        }
//      }
//      k++;
//    }
    System.out.println(Arrays.toString(arr));

    //소수를 출력(0을 출력)
    for (int i = 2; i < arr.length-1; i++) {
      if(arr[i] == 0) {
        System.out.print(i+", ");
      }
    }
  }
}
Outputs :
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
```