알고리즘 문제해결기법

제3주 Intermediate Level Data Structures

문제 07: 최소값 찾기 [백준 11003번]

N개의 수 A_1, A_2, ..., A_N과 L이 주어진다.

$D_i = A_{i-L+1} \sim A_i$ 중의 최소값이라고 할 때, D에 저장된 수를 출력하는 프로그램을 작성하시오. 이때, $i \le 0$ 인 A_i는 무시하고 D를 구해야 한다. $1 \le L \le N \le 5{,}000{,}000$이고 $-10^9 \le A_i \le 10^9$이다.

문제 07: 최소값 찾기 [백준 11003번]

L=4

data	1	10	6	5	21	1	2	16	9	3	23	19	3	6	31	5
D	1	1	1	1	5	1	1	1	1	2	3	3	3	3	3	3

매 순간 자신을 포함하여 가장 최근의 L개의 값 중에서 최소값을 새로 찾는다면 시간복잡도는 $O(LN)=O(N^2)$이다.

예를 만들고 충분히 관찰하라 !

문제 07: 최소값 찾기 [백준 11003번]

쓸모있는 수들이 가진 가장 중요한 특징은?

L=4

data	1	10	6	5	21	1	2	16	9	3	23	19	3	6	31	5
	1	10	6	5	21	1	2	16	9	3	23	19	3	6	31	5
	1	~~10~~	6	5	21	1	2	16	9	3	23	19	3	6	31	5
	1	~~10~~	~~6~~	5	21	1	2	16	9	3	23	19	3	6	31	5
	1	~~10~~	~~6~~	5	21	1	2	16	9	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	16	9	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	16	9	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	9	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	~~9~~	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	~~9~~	3	23	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	~~9~~	3	~~23~~	19	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	~~9~~	3	~~23~~	~~19~~	3	6	31	5
	~~1~~	~~10~~	~~6~~	~~5~~	~~21~~	1	2	~~16~~	~~9~~	~~3~~	~~23~~	~~19~~	3	6	31	5

60이 나온 상황에서
10은 무쓸모. 왜?

21이 나온 상황에서
1은 이제 무쓸모. 왜?

쓸모없는 수를 모두 지워보자.

각 스텝에서 출력될 수는 누구?

Double Ended Queue: Deque

- 양 끝에서 삽입과 삭제를 지원하는 큐
  - `addFront`
  - `removeFront`
  - `addRear`
  - `removeRear`
  - `isEmpty`
  - `peekFront`, `peekRear` 등
- std::deque in C++
- Interface Deque<E> in Java

문제 07: 최소값 찾기 [백준 11003번]

나보다 먼저 나온
나보다 크거나 같은 수는 모두 제거

```
for (int i=0; i<N; i++) {
    while(!isEmpty() && peekRear().value >= d[i])
        removeRear();
    addRear(new Item(i, d[i]));
    while (peekFront().index < i-L+1)
        removeFront();
    print(peekFront().value);
}
```

값과 인덱스를 함께 저장한다.

나로부터 L칸 이상 떨어진 수는
모두 제거

큐는 비어있지 않고 (why?)
오름차순으로 정렬되어 있으므로
항상 맨 앞의 원소를 출력한다.

문제 08: Substring

Given a string of uppercase alphabets of length N, find the length of the longest substring in which no alphabet appears more than K times. For an example, let the input string be

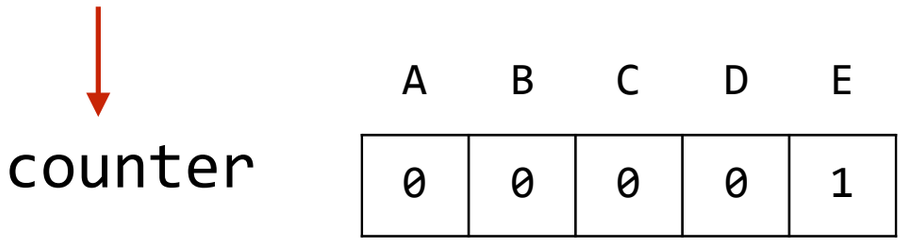
ABCACBDEFABC

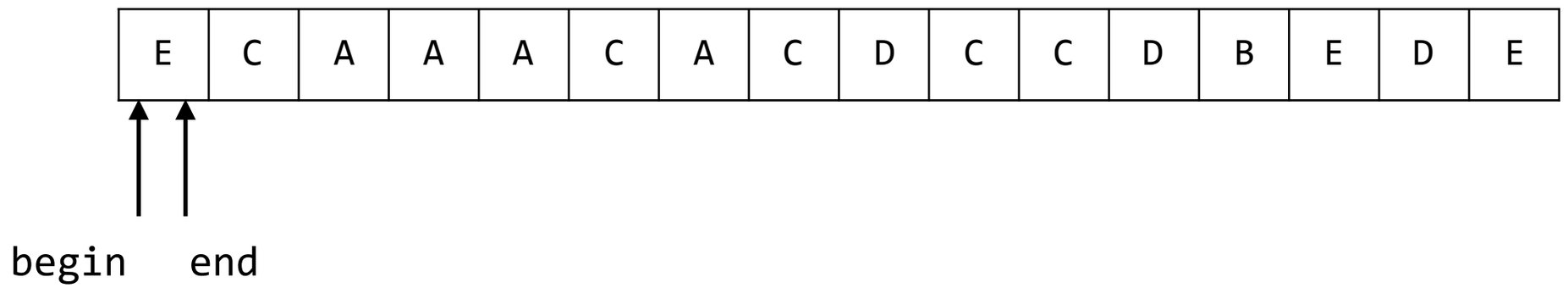
Then, if K=1, the answer is 6 since ACBDEF is the longest substring without duplication of any alphabet. If K=2, then the answer is 9 since no alphabet appears more than twice in ABCACBDEF.

Both N and K is at most 1,000,000.

문제 08: Substring

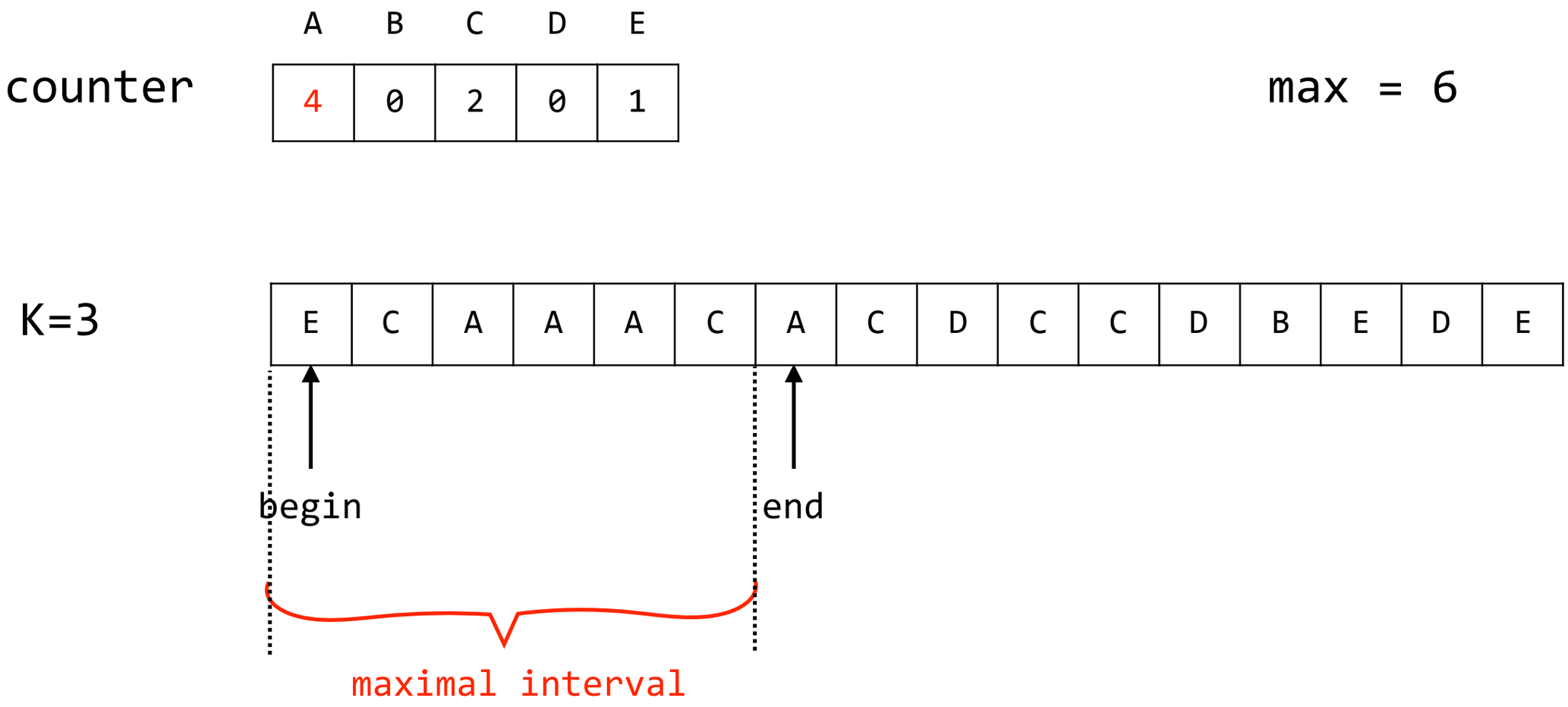
인덱스 begin에서 end사이에 등장하는 각 알파벳별 카운터





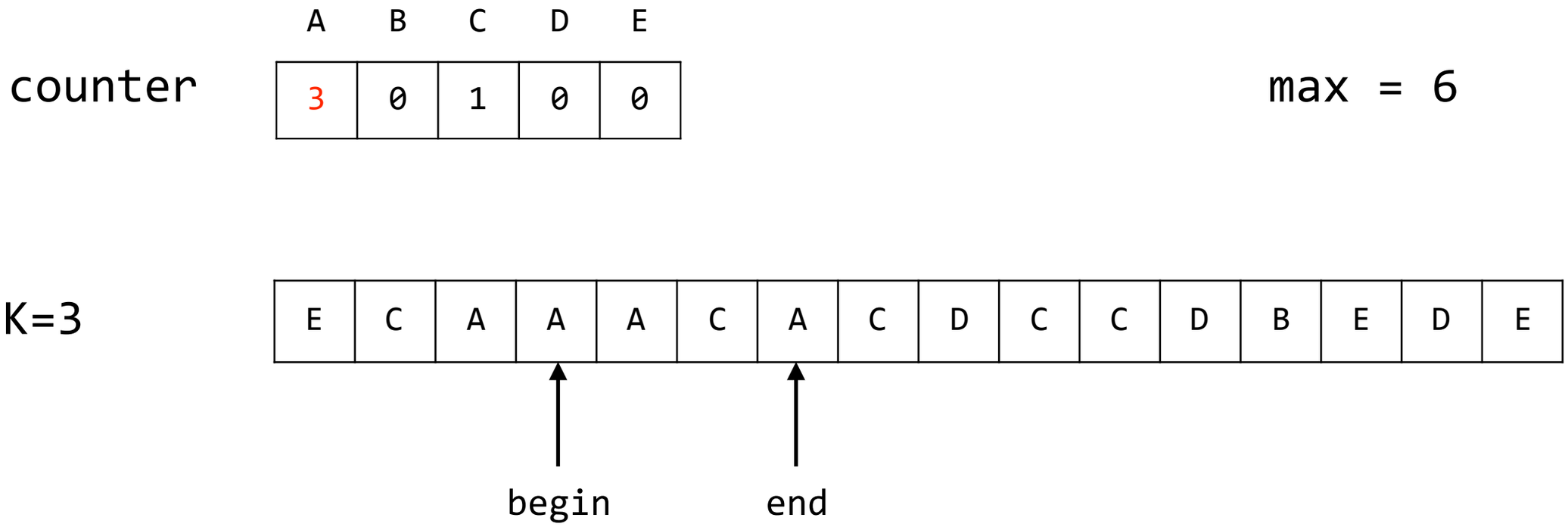
begin과 end는 feasible한 구간의 시작점과
끝점을 표현한다.

문제 08: Substring



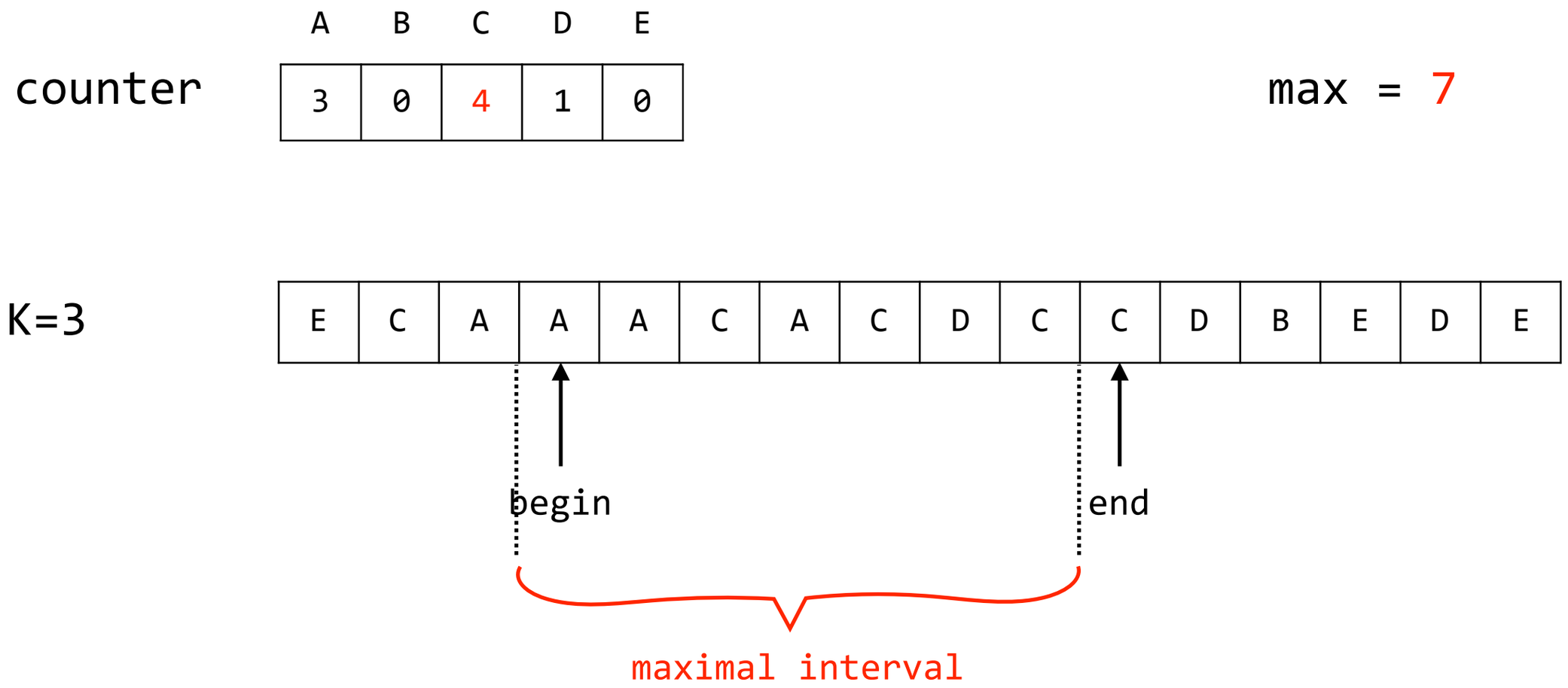
빈도가 K를 초과하는 알파벳이 나타날 때까지 end를 전진한다. 와중에 feasible한 구간의 최대 길이 max를 지속적으로 update한다.

문제 08: Substring



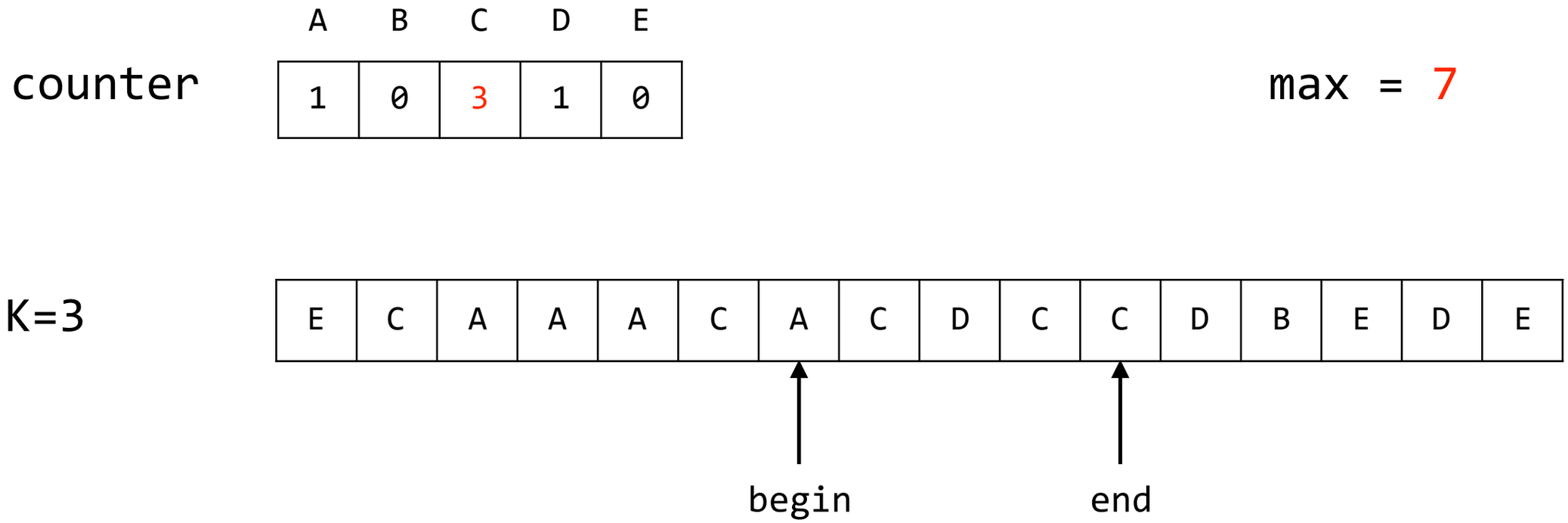
feasible해질 때까지, 즉 A의 count가 4미만이 될때까지 begin을 전진하면서
구간범위 밖으로 밀려나는 문자들의 카운트를
감소시킨다.

문제 08: Substring



다시 counter가 K를 초과하는 문자가 나타날때까지 end를 전진한다. 이 와중에 max값은 지속적으로 업데이트한다.

문제 08: Substring



feasible해질 때까지 begin을 전진하면서
구간범위 밖으로 밀려나는 문자들의 카운터를
감소시킨다.

문제 08: Substring

```
int [] count = new int [26];
int max = 0;
for (int begin=0, end=0; end<N; end++)  {
    count[data[end]-‘A’]++;
    while(count[data[end]-‘A'] > K)
        count[data[begin++]-‘A']--;
    if (end-begin+1 > max)
        max = end-begin+1;
}
print(max);
```

문제 09: Subsequence

Given a sequence of N integers, find the length of the longest contiguous subsequence whose elements are all distinct. For an example, suppose that the input sequence is as follows:

-2, 4, <u>11, 3, -2, 4</u>, -2, 10

The output is 4 since either 11, 3, -2, 4 is the longest subsequence without duplications. The number of integers N is at most 1,000,000 and each integer is between -10^9 and 10^9.

문제 09: Subsequence

- Problem 08과 본질적으로 동일한 문제 (K=1인 경우)
- 다만 -10억에서 10억 사이의 정수이므로 모든 정수에 대한 카운터를 만들 수 없음
- 어떤 정수가 이전에 이미 나왔는지 아닌지를 검사
- 필요한 자료구조 ?

문제 09: Subsequence

- **동적 집합 (dynamic set)**
  - 현재까지 나온 원소들의 집합을 유지
  - **삽입(insert)**, **삭제(remove)**, 그리고 어떤 원소가 있는지 아닌지(**membership**) 검사
- **집합의 표현**
  - **트리 기반의 자료구조**와 **해쉬 테이블**이 대표적인 2가지 방법
  - 이진검색트리, 레드블랙트리 등 - 평균 혹은 최악의 경우 O(logN)
  - **해쉬 - 평균 O(1) 시간에 멤버쉽 검사**
  - `HashSet` in Java
  - `set` or `unordered_set` in C++

문제 09: Subsequence

```
int max = 0;
HashSet<Integer> set = new HashSet<Integer>();
for (int begin=0, end=0; end<N; end++)   {
    while (set.contains(num[end]))
        set.remove(num[begin++]);
    set.add(num[end]);
    if (end-begin+1 > max)
        max = end-begin+1;
}
print(max);
```

contains와 remove가 O(1) 이라는 가정하에서 O(N)

Problem 9: Alternative Solution

위의 선분들 중 어떤 것도 **완전히 포함(contain)하지 않는** 가장 긴 구간은?

문제 10: Subsequence2

Given a sequence of N integers and another integer K, find the length of the longest contiguous subsequence in which no integer appears more than K times. For an example, suppose that the input sequence is as follows:

-2, 4, 11, 3, -2, 4, -2, 10

If K=1, the answer is 4 since either -2, 4, 11, 3 is the longest subsequence without duplications. If K=2, the answer is 7 since no integer appears more than twice in sequence 4, 11, 3, -2, 4, -2, 10 . Both N and K are at most 1,000,000 and each integer is between -1,000,000,000 and 1,000,000,000.

Map (or Dictionary)

- “key=value”형태의 데이터를 저장하는 자료구조
- 예: 어떤 사람에 대해서 이름, 나이, 성별 등의 정보를 저장

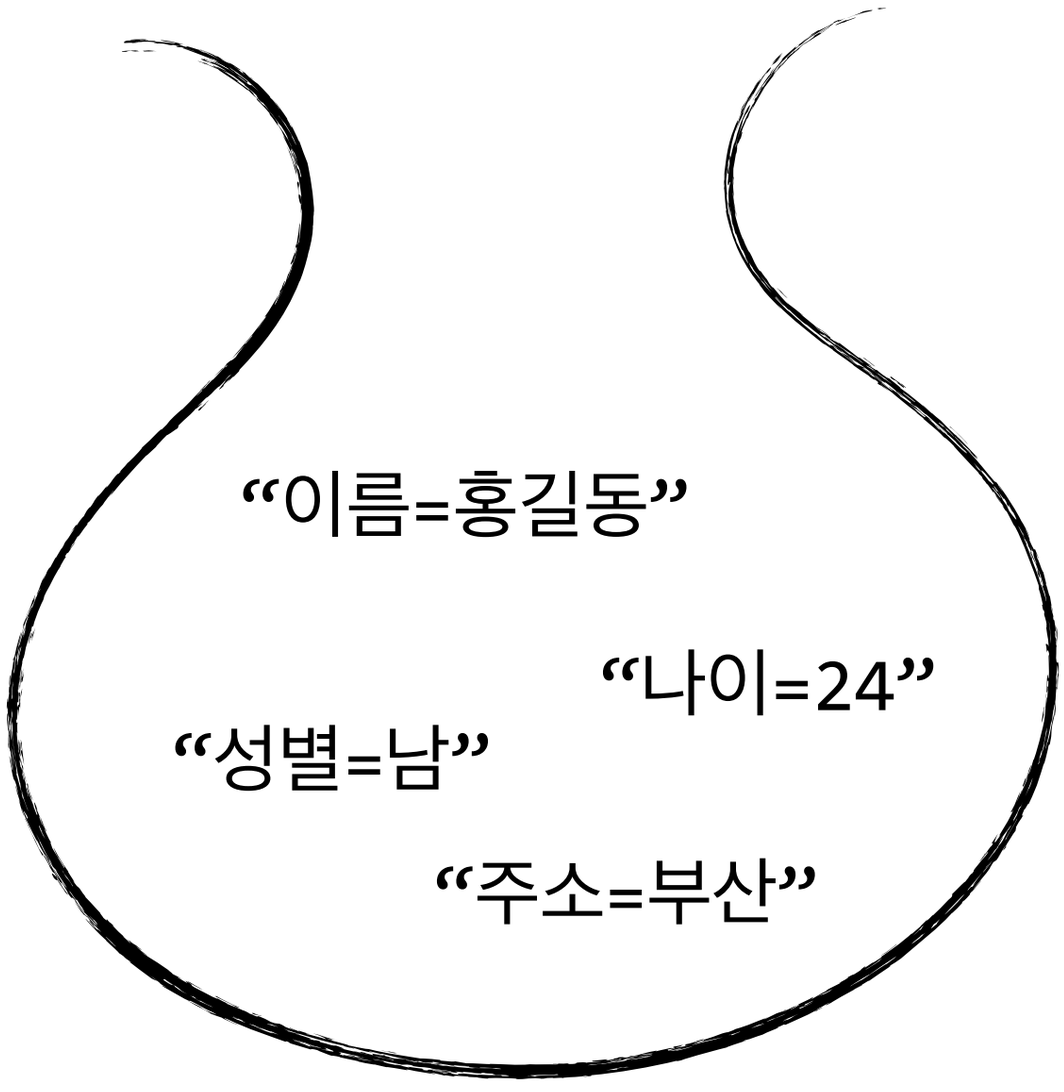
```
HashMap<String,String> map = new HashMap<String,String>();
map.put(“이름”, “홍길동”);
map.put(“나이”, “24”);
map.put(“성별”, “남”);
map.put(“주소”, “부산”);
```

이름, 나이, 성별, 주소 등을 key라고 부름

홍길동, 24, 남, 부산 등의 값은 각각의 키에 대한 value임

Map

```
HashMap<String,String> map = new HashMap<String,String>();
map.put(“이름”, “홍길동”);
map.put(“나이”, “24”);
map.put(“성별”, “남”);
map.put(“주소”, “부산”);
```

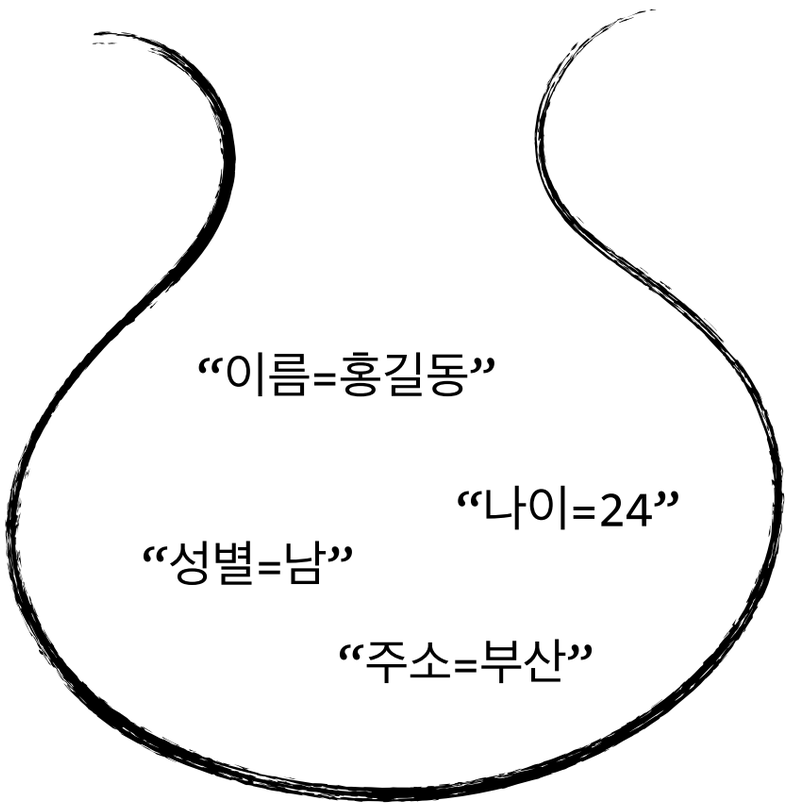


검색

- **key를 이용한 검색**

```
boolean result = map.containsKey(“이름”);
boolean result2 = map.containsKey(“직업”);
String name = map.get(“이름”);
```

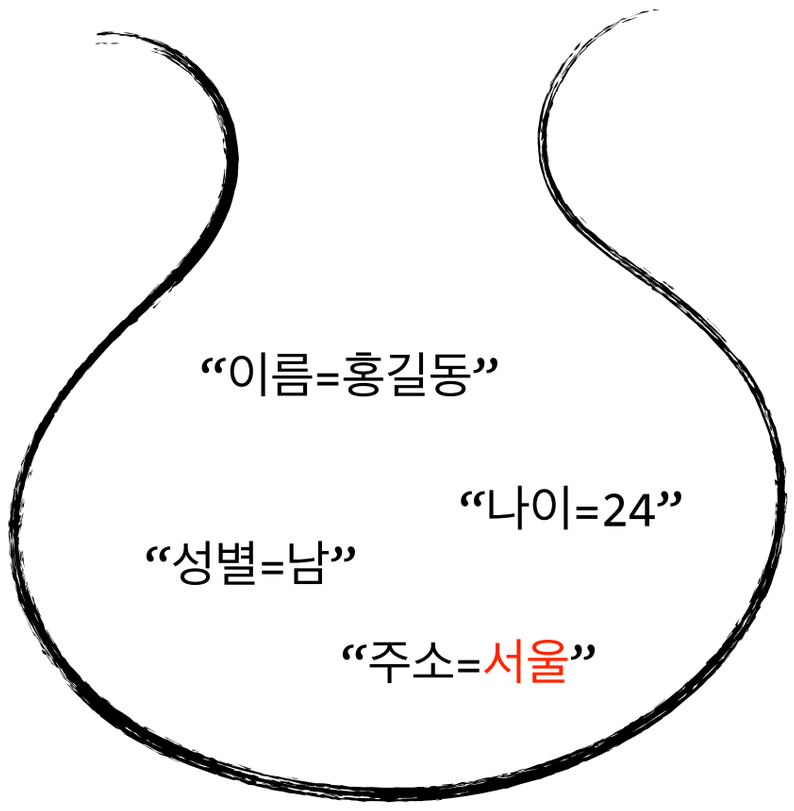
result는 true,
result2는 false,
name은 “홍길동”이 됨



특정 key의 value를 갱신

```
map.put(“주소”, “서울”);     // 주소를 “서울”로 갱신
String address = map.get(“주소”);
```

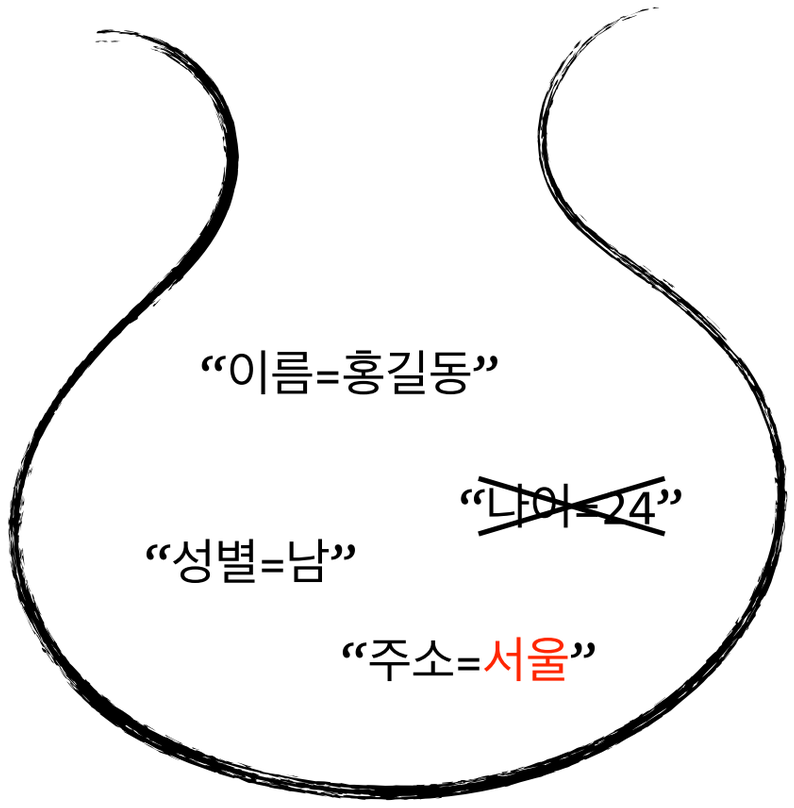
주소가 “서울”로 update됨.
따라서 address는 “서울”이 됨



특정 key의 삭제

```
map.remove("나이");
String age = map.get("나이");
```

"나이=24" 쌍이 삭제됨.
따라서 age는 null이 됨



Problem 10

- Problem 08과 본질적으로 동일한 문제
- 다만 -10억~10억 사이의 값이므로 배열을 이용해 각 정수의 등장횟수를 카운트하는 카운터를 만들 수 없음 (20억 크기의 배열이 필요하므로)
- HashMap을 이용해 실제로 입력에 나온 정수들에 대해서만 카운터를 만듦

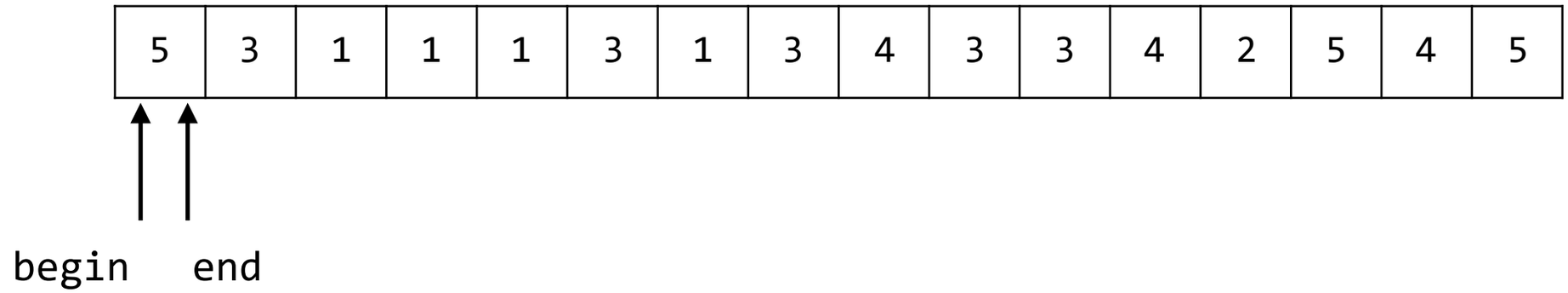
```
HashMap<Integer, Integer> map
        = new HashMap<Integer, Integer>();
```

입력에 등장한 각각의 정수가 하나의 Key가 됨

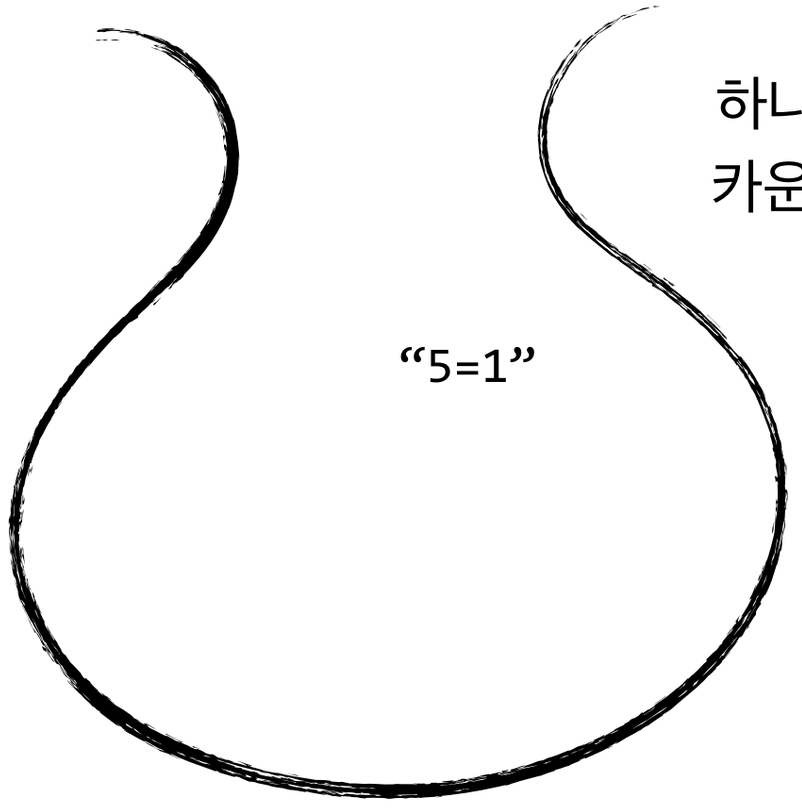
그 정수의 등장 횟수가 value가 됨

Problem 10

K=3인 경우



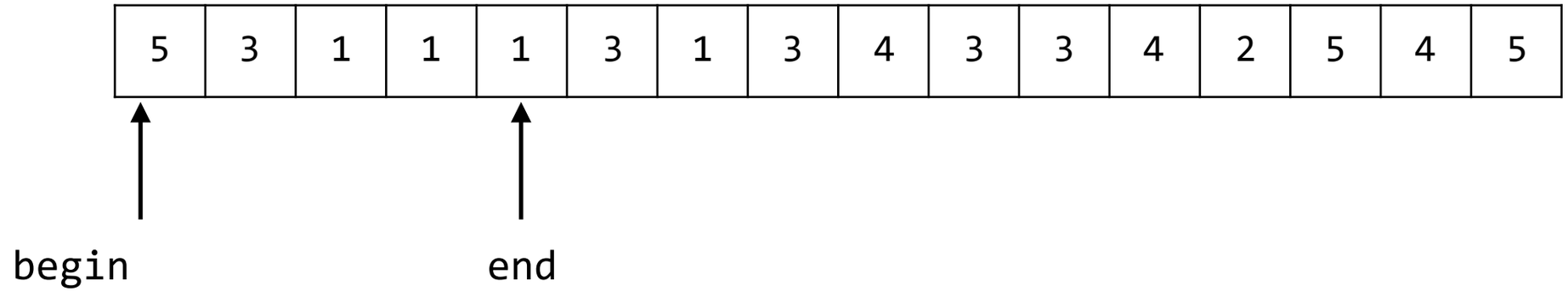
begin과 end는 feasible한
구간의 시작점과
끝점을 표현한다.



하나의 HashMap을
카운터로 사용한다.

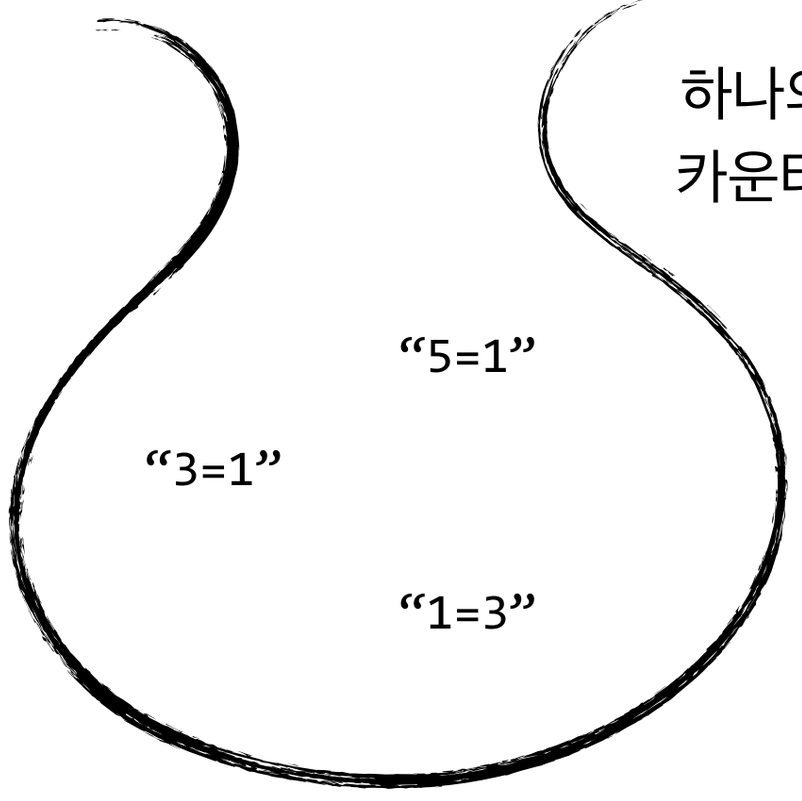
Problem 10

K=3인 경우




```
if (!map.containsKey(data[end]))
  map.put(data[end], 1);
else {
  int cnt = map.get(data[end]);
  map.put(data[end], cnt+1);
}
```

입력 정수를 하나씩 보면서 그 정수에 대한 카운터를 1 증가시킨다.

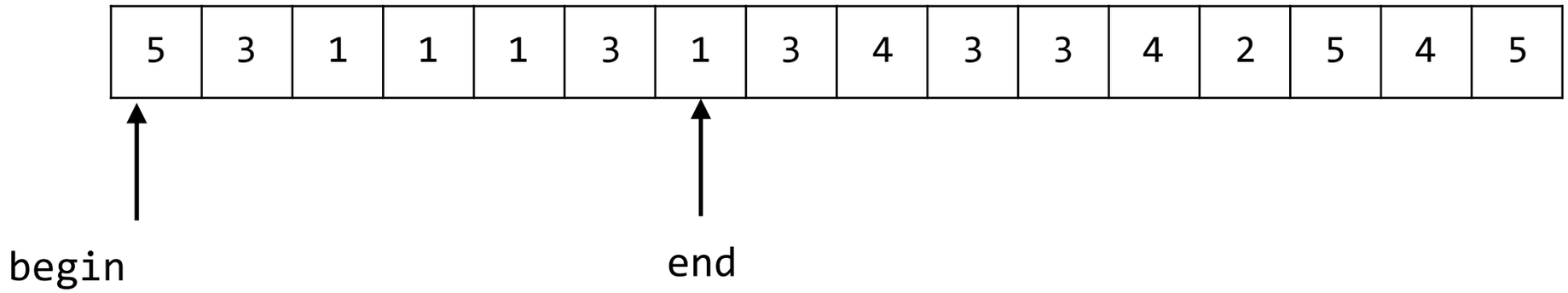


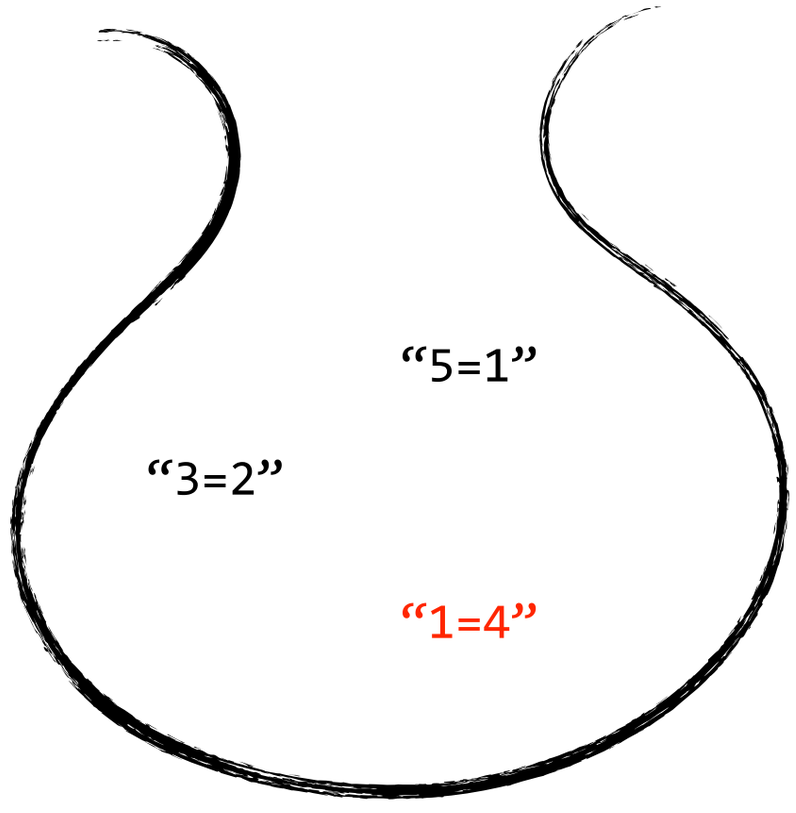
하나의 HashMap을 카운터로 사용한다.

“지금까지 5가 1번, 3이 1번, 그리고 1이 3번 나왔음을 표시”

Problem 10

K=3인 경우

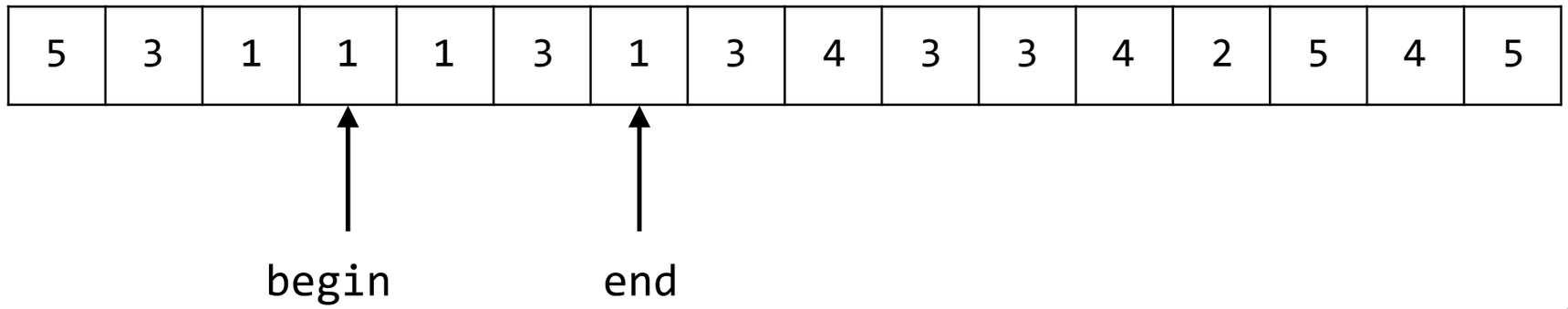




1이 4번 나와서 K번을 초과했음

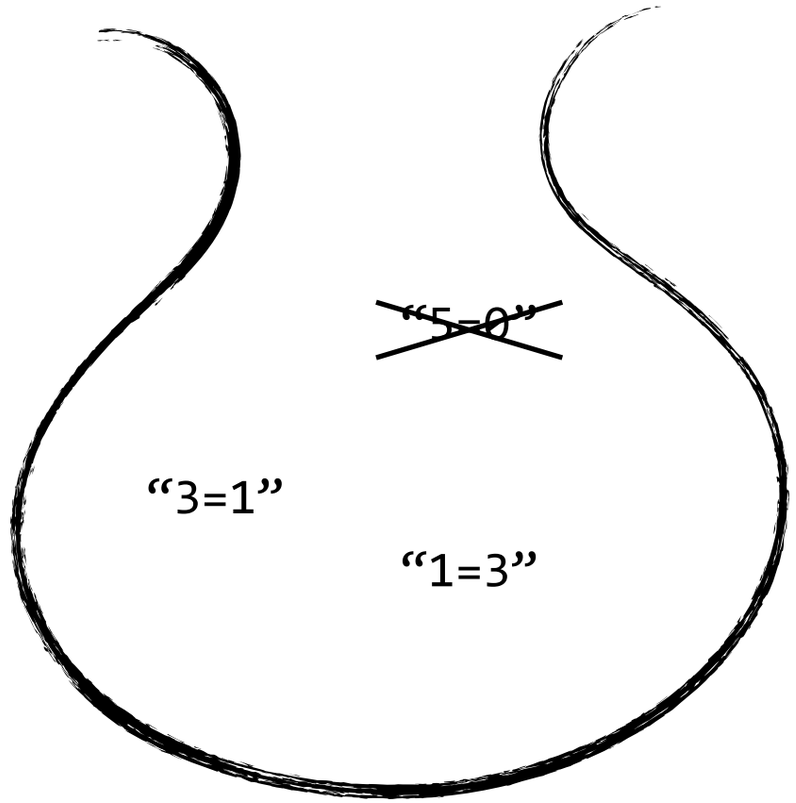
Problem 10

K=3인 경우




```
while (map.get(data[end])>K) {
    int cnt = map.get(data[begin]);
    map.put(data[begin], cnt-1);
    begin++;
}
```

입력 정수 data[end]의 카운트가 K이하가 될 때까지 begin을 전진하면서 [begin,end] 구간에서 빠지는 정수들의 카운터를 1 감소시킨다 (단,카운트가 0에 도달하는 정수들은 아예 map에서 remove한다)



map은 항상 [begin,end] 구간내의 정수들의 카운트를 유지한다.

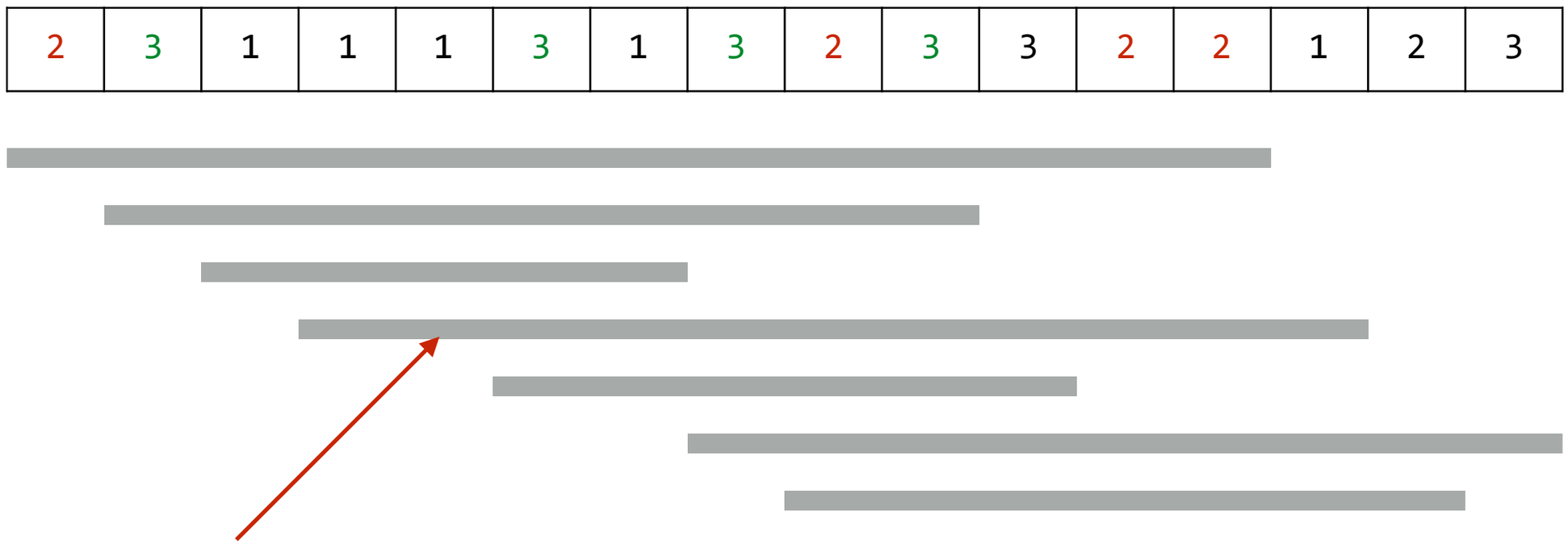
Problem 10

```
HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
int begin=0, max = 0;
for (int end=0; end<N; end++)  {
    if (!map.containsKey(data[end]))
        map.put(data[end], 1);
    else
        map.put(data[end], map.get(data[end])+1);

    while(map.get(data[end])>K) {
        int count = map.get(data[begin]);
        if (count > 1)
            map.put(data[begin], count-1);
        else
            map.remove(data[begin]);
        begin++;
    }

    if (end-begin+1>max)
        max = end-begin+1;
}
System.out.println(max);
```

HashMap 연산들이 O(1)이라는 가정하에 O(N)

Problem 10: Alternative Solution

K=3인 경우



각 선분에는 동일한 값이 K+1개 들어있다.

위의 선분들 중 어떤 것도 완전히 포함하지 않는 가장 긴 구간은?

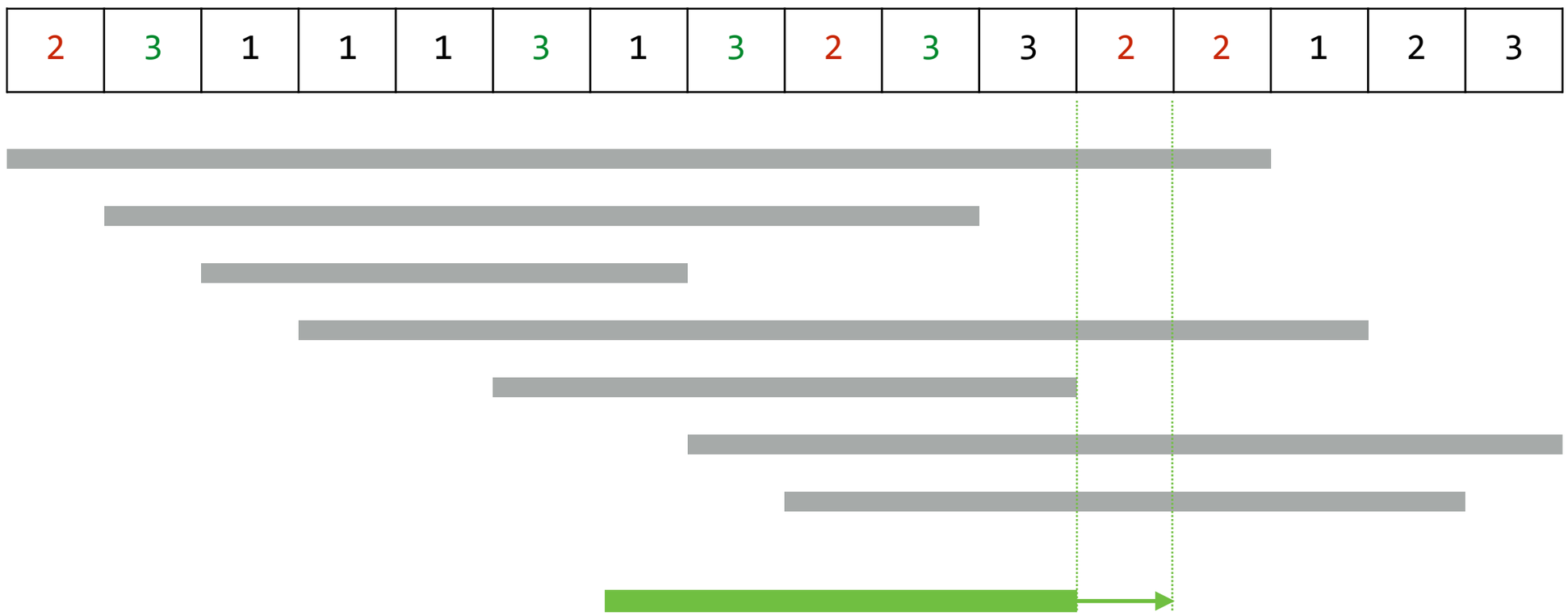
해결전략

모든 maximal한 구간을 찾아서

그 중에 가장 긴 것을 고른다.

Key Observation

K=3인 경우



임의의 maximal feasible한 구간의 끝점은
어떤 interval의 끝점 바로 앞 위치이거나
혹은 맨 마지막 위치이다.
(만약 아니라면 언제나 거기까지 연장할 수 있기 때문이다.)

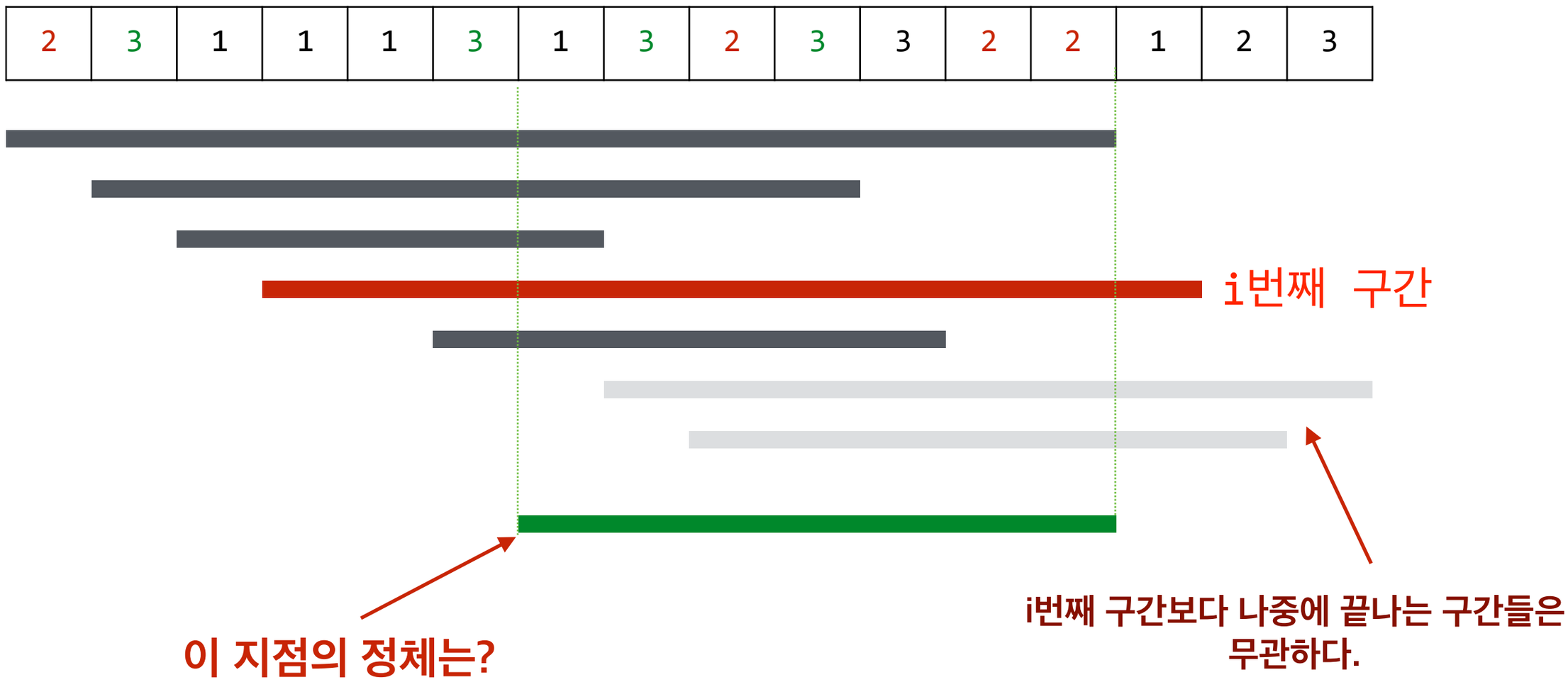
고려해 봐야할

maximal한 구간의 후보(candidate)는 M+1개이다.

여기서 M은 구간의 개수이다.

i번째 구간의 끝점 바로 앞에서 끝나는 구간

K=3인 경우



i번째 구간의 끝점 바로 앞에서 끝나는 구간의 시작점은?

Problem 10: Alternative Solution

각각의 구간 i에 대해서

T = 구간 i의 종료지점 - 1; (혹은 N-1)

S = **종료 지점이 구간 i보다 작거나 같은 구간들의 시작점의 최대값** + 1;

(혹은 만약 그런 구간이 없으면 0);

만약 T-S+1 > max이면, max = T-S+1;

Problem 10: Alternative Solution

구간들을 종료지점의 오름차순으로 정렬한다;

정렬된 구간들을 $I_i = [s_i, t_i]$, i=0,…,M-1,로 표시하자.

```
for i=0,…,M do
    If i==M,
        end = N-1;
    else
        End = ti-1;
if i==0,
    begin = 0;
else
    begin = max(begin, si-1 + 1);
if end-start+1 > max,
    max = end-begin+1;
```

i==M일 경우에는
배열의 끝까지 포함한다.

구간들이 종료지점에 대해서 정렬
되어 있으므로
이렇게 해주는 것으로 충분하다.

Problem 10: Alternative Solution

```
public static void solveInstance() {
    int N = sc.nextInt();
int K = sc.nextInt();
data = new Item [N];
for (int i=0; i<N; i++)
    data[i] = new Item(i, sc.nextInt());
Arrays.sort(data);
```

Problem 10: Alternative Solution

```
Interval [] intervals = new Interval[N];
int m = 0;     // number of intervals
for (int i=0; i<N-1; i++)  {
    if (i+K<N && data[i].val == data[i+K].val)
       intervals[m++] =
                  new Interval(data[i].index, data[i+K].index));
}
if (m==0) {
    System.out.println(N);
    return;
}
// sort intervals by increasing order of their ends
Arrays.sort(intervals, 0, m);
```

Problem 10: Alternative Solution

```
int begin, end, max = 0;
for (int i=0; i<=m; i++)  {
    end = (i==m ? N-1 : intervals[i].t-1);
        begin = (i==0 ? 0 : Math.max(intervals[i-1].s+1, begin));
        max = Math.max(max, end - begin + 1);
    }
    System.out.println(max);
}
```

Problem 10: Alternative Solution

```
private static class Item implements Comparable {
    int index;
    int val;

    public Item(int index, int val) {
        this.index = index;
        this.val = val;
    }

    @Override
    public int compareTo(Object o) {
        Item other = (Item)o;
        if (val > other.val)
            return 1;
        else if (val < other.val)
            return -1;
        else
            return index - other.index;
    }
}
```

Problem 10: Alternative Solution

```
private static class Interval implements Comparable {
    int s;
    int t;

    public Interval(int s, int t) {
        this.s = s;
        this.t = t;
    }

    @Override
    public int compareTo(Object other) {
       return t - (Interval)other.t;
    }
}
```

O(NlogN)