Chapter 08.
파트별 쪽지시험#2

Clip 01 | [3078] 좋은 친구
큐

Clip 02 | [9935] 문자열 폭발
스택

Clip 03 | [24268] 2022는 무엇이 특별할까?
순열

Clip 04 | [1316] 그룹 단어 체커
구현

Clip 05 | [1167] 트리의 지름
트리

Ch08. 파트별 쪽지시험 #2

1. [3078] 좋은 친구

쪽지시험 #2

BOJ3078: 좋은 친구

문제

상근이는 환갑을 바라보던 나이에 수능 시험을 다시보고 교대에 입학했고, 초등학교 선생님으로 취직했다.

- 상근: 요즘 애들은 친구를 사귀지 않나봐. 내가 앞에서 보고 있으면, 친구가 있는 학생이 별로 없는 것 같아.
- ??: 오빠! 오빠는 말콤의 친구와 성적이라는 책 안 읽어 봤어? 이 책에는 성적과 친구가 무슨 관계가 있는지 나와. 요즘 애들은 친구를 사귀기 전에 먼저 그 친구의 반 등수를 살펴봐. 말콤은 이 연구를 하기 위해서 6년동안 초등학교에서 선생님으로 위장 했었지. 하지만, 6년이라는 시간을 초등학교에서 보냈지만, 그 사람은 결국 결론을 얻지 못했어.
- 상근: 근데?
- ??: 말콤이 어느 날 자신이 초등학생이 되어 학교를 활보하는 꿈을 꾸었어. 근데 잠을 깨고 나니 내가 꿈을 꾸고 초등학생이 된건지, 아니면 초등학생이 꿈을 꾸고 지금의 내가 되어있는지를 모르겠는거야. 그래서 말콤은 상식적인 사고 방식에 큰 의문을 가졌지. 그 때 말콤은 깨달았던거야. 초등학교 친구는 부질없구나. 그제서야 알게된거야. 모든 학생은 자신과 반 등수의 차이가 K를 넘으면 친구가 아니라는거.
- 상근: 아? 근데 K는 어떻게 구해?
- ??: K는 문제에서 주어지지. 근데, 더 중요한 사실이 있어. 친구와 좋은 친구의 차이야. 말콤이 친구와 성적을 쓰고 2년 뒤에 낸 책인 좋은 친구라는 책에는 좋은 친구는 이름의 길이가 같아야 된다는 말이 나와.
- 상근: 아! 그럼 난 오늘 집에 가서 우리 반에 좋은 친구가 몇 쌍이나 있는지 구해봐야 겠어!

상근이네 반의 N명 학생들의 이름이 성적순으로 주어졌을 때, 좋은 친구가 몇 쌍이나 있는지 구하는 프로그램을 작성하시오. 좋은 친구는 등수의 차이가 K보다 작거나 같으면서 이름의 길이가 같은 친구이다.

입력

첫째 줄에 N과 K가 주어진다. ($3 \le N \le 300,000$, $1 \le K \le N$) 다음 N개 줄에는 상근이네 반 학생의 이름이 성적순으로 주어진다. 이름은 알파벳 대문자로 이루어져 있고, 2글자 ~ 20글자이다.

BOJ3078: 좋은 친구

문제

상근이네 반의 N명 학생들의 이름이 성적순으로 주어졌을 때, 좋은 친구가 몇 쌍이나 있는지 구하는 프로그램을 작성하시오. 좋은 친구는 등수의 차이가 K보다 작거나 같으면서 이름의 길이가 같은 친구이다.

입력

첫째 줄에 N과 K가 주어진다. (3 ≤ N ≤ 300,000, 1 ≤ K ≤ N) 다음 N개 줄에는 상근이네 반 학생의 이름이 성적순으로 주어진다. 이름은 알파벳 대문자로 이루어져 있고, 2글자 ~ 20글자이다.

출력

첫째 줄에 좋은 친구가 몇 쌍이 있는지 출력한다.

맨 마지막줄만 읽어도 충분히 문제를 풀 수 있다

BOJ3078: 좋은 친구

문제 요약

- N개의 문자열이 입력으로 주어진다
- 좋은 친구 조건
  - index 값의 차이가 K보다 작거나 같다
  - 문자열의 길이가 같다
- 3 <= N <= 300,000, 1<= K <= N

BOJ3078: 좋은 친구

문제 분석

- 좋은 친구 조건
  - index 값의 차이가 K보다 작거나 같다

index 값의 차이가 K보다 큰 친구는 관심을 가지지 않아도 된다

BOJ3078: 좋은 친구

A	B	C	D	E	F	G

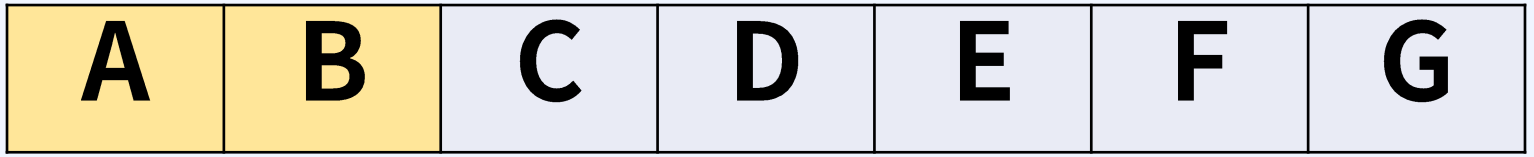
K = 3

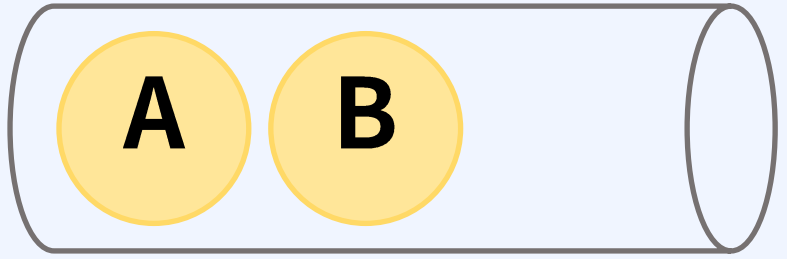
BOJ3078: 좋은 친구

A	B	C	D	E	F	G

K = 3

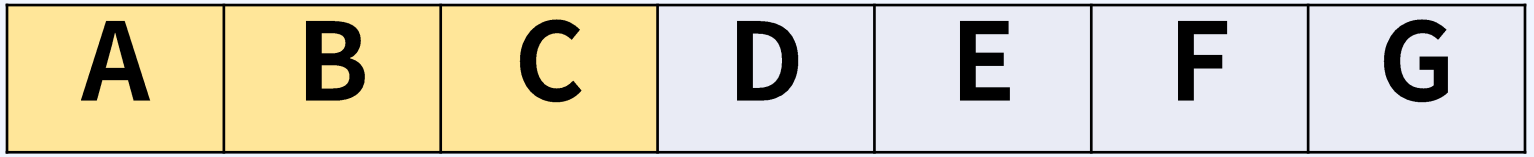
BOJ3078: 좋은 친구

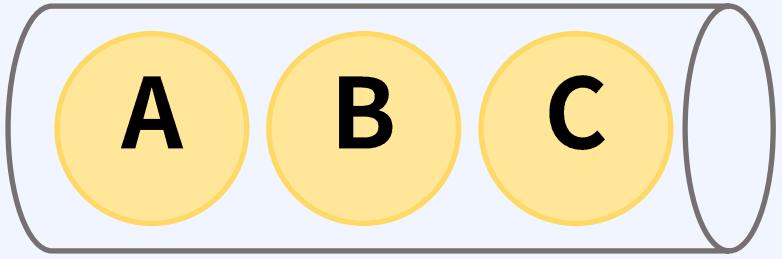




K = 3

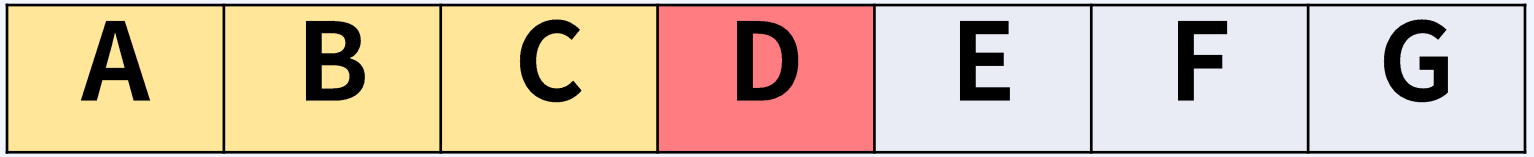
BOJ3078: 좋은 친구



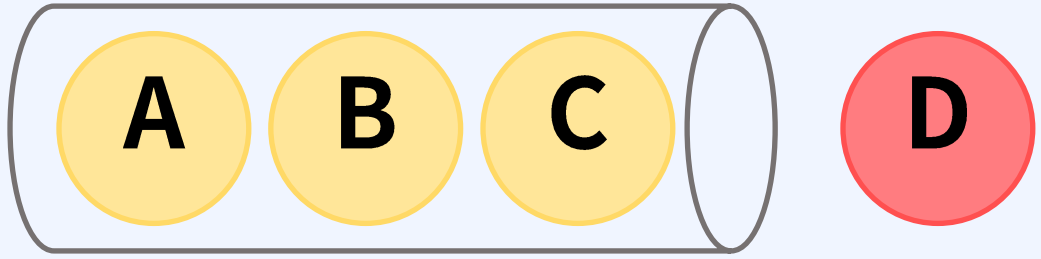


K = 3

BOJ3078: 좋은 친구



큐가 가득 차 있을 때 새로운 데이터가 들어오면?

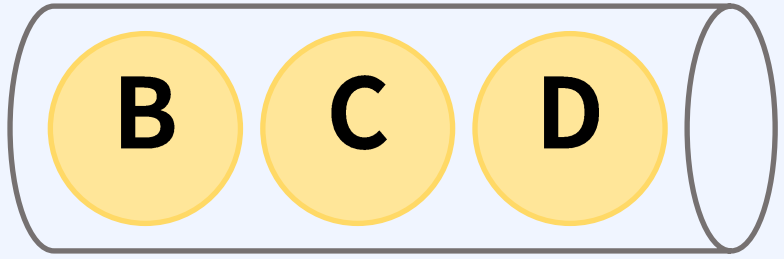


K = 3

BOJ3078: 좋은 친구

A	B	C	D	E	F	G

먼저 들어온 Front 데이터를 삭제하고, 새 데이터를 넣는다

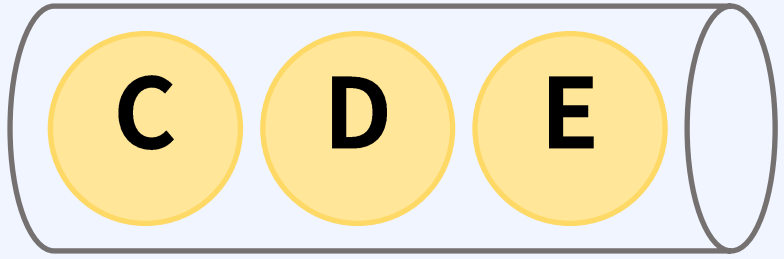


K = 3

BOJ3078: 좋은 친구

A	B	C	D	E	F	G

먼저 들어온 Front 데이터를 삭제하고, 새 데이터를 넣는다

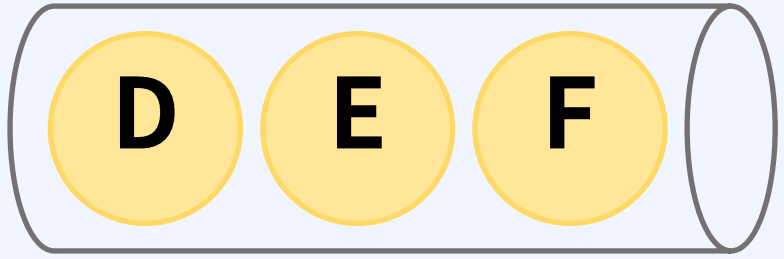


K = 3

BOJ3078: 좋은 친구

A	B	C	D	E	F	G

먼저 들어온 Front 데이터를 삭제하고, 새 데이터를 넣는다

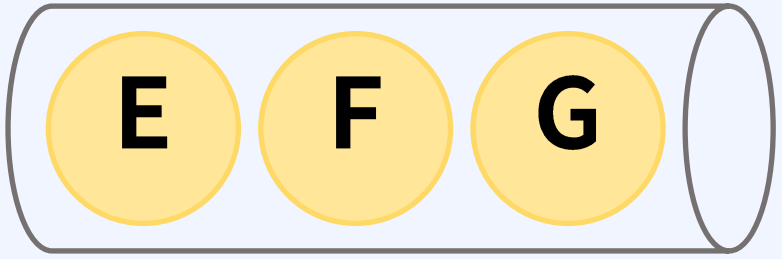


K = 3

BOJ3078: 좋은 친구

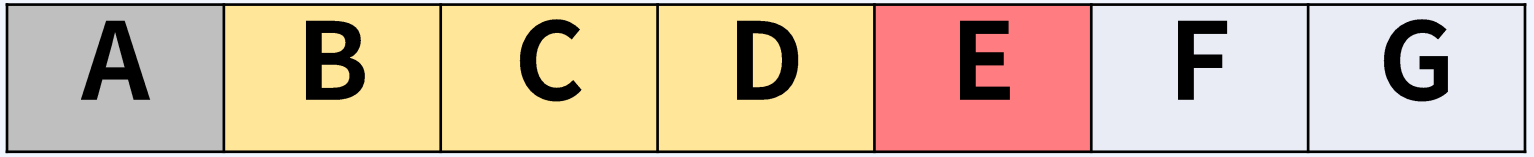
A	B	C	D	E	F	G

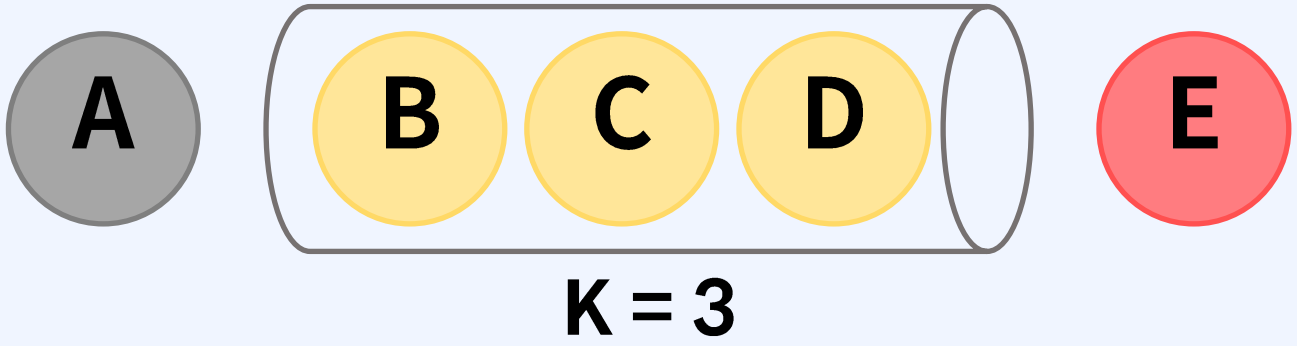
먼저 들어온 Front 데이터를 삭제하고, 새 데이터를 넣는다



K = 3

BOJ3078: 좋은 친구





K보다 등수가 먼 친구는 좋은 친구가 아니다

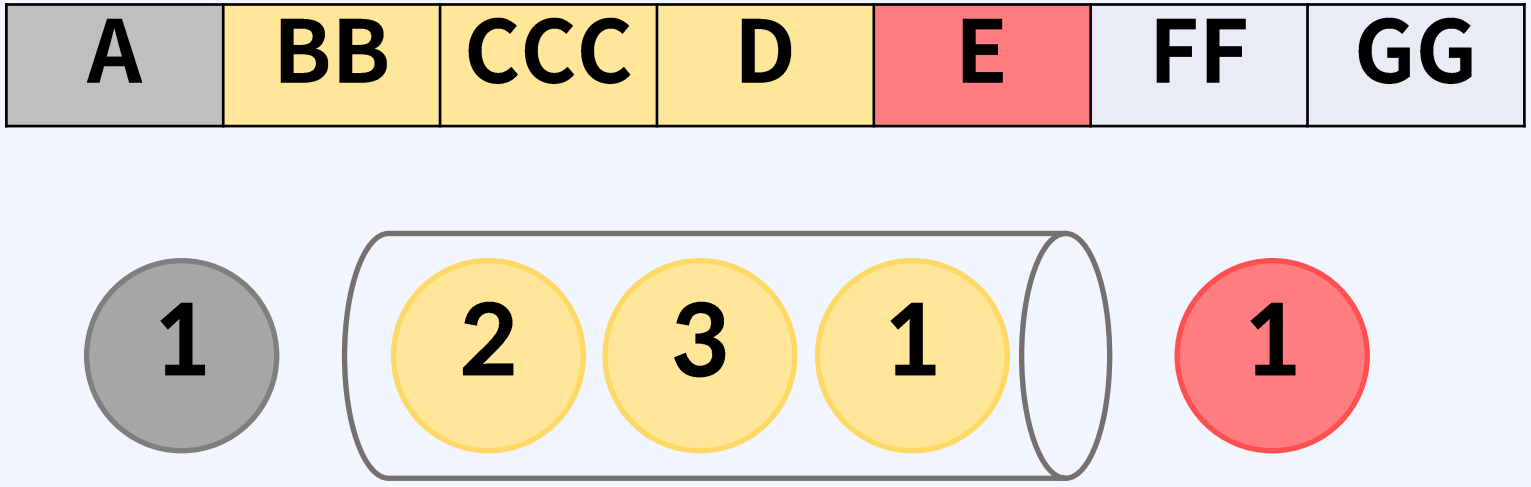
→

큐에서 이미 나가거나, 들어오지 않은 데이터는 관심 밖의 영역이다

BOJ3078: 좋은 친구

좋은 친구 조건

- 문자열의 길이가 같다

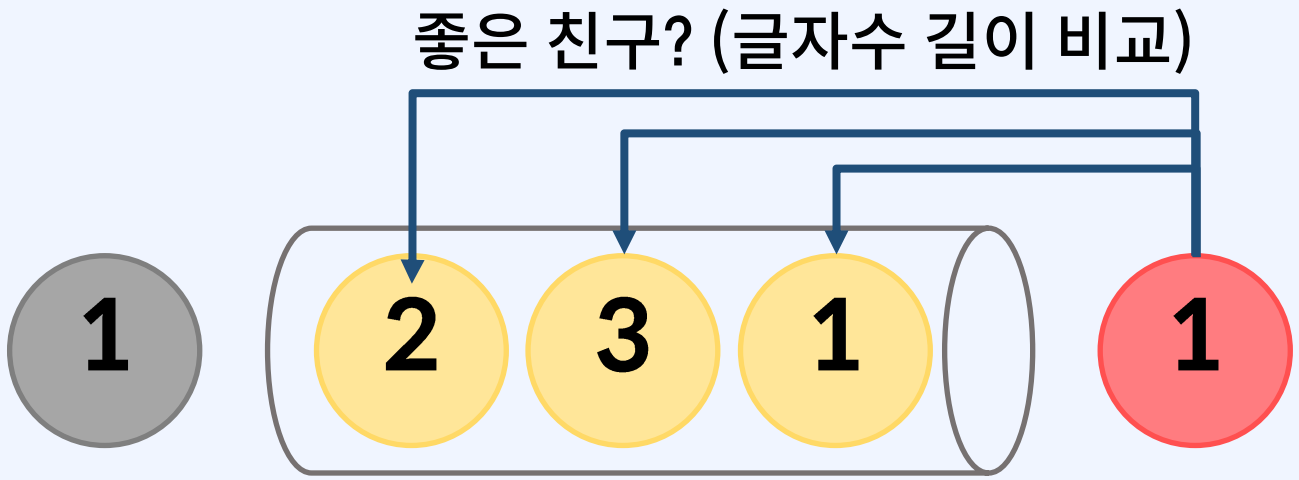


문제는 인원 수에만 관심이 있다 (대상이 누구인지 관심 X)
큐에 저장할 데이터? → 문자열의 길이

BOJ3078: 좋은 친구

좋은 친구 쌍을 구하는 방법?
- 새로 들어오는 원소에 대해, 큐에 있는 모든 데이터를 비교?

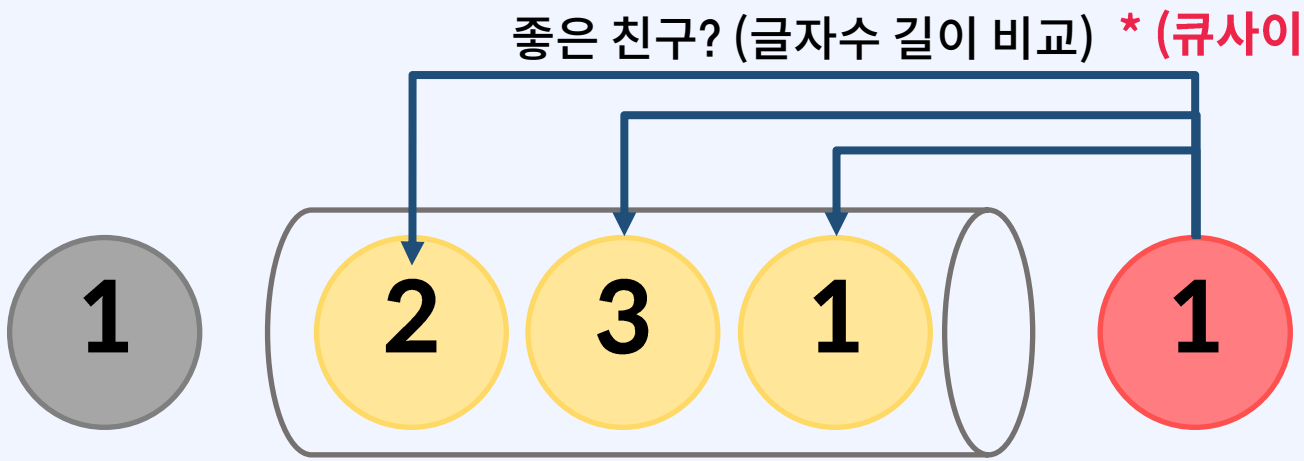
A	BB	CCC	D	E	FF	GG



BOJ3078: 좋은 친구

좋은 친구 쌍을 구하는 방법?
- 새로 들어오는 원소에 대해, 큐에 있는 모든 데이터를 비교?

A	BB	CCC	D	E	FF	GG



(1 <= K <= N <= 300,000)

BOJ3078: 좋은 친구

좋은 친구 쌍을 구하는 방법?
- 새로 들어오는 원소에 대해, 길이에 대한 빈도 수 배열 생성

A	BB	CCC	D	E	FF	GG

0	1	2	3
0	1	1	1

```
queue.add(next);
friends[next]++;
```

```
if(queue.size() > k) {
    int out = queue.poll();
    friends[out]--;
}
```

BOJ3078: 좋은 친구

좋은 친구 쌍을 구하는 방법?
- 새로 들어오는 원소에 대해, 길이에 대한 빈도 수 배열 생성

A	BB	CCC	D	E	FF	GG

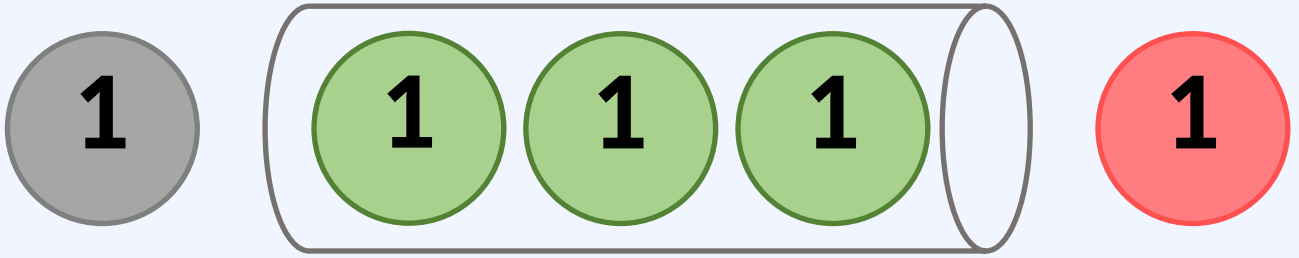
0	1	2	3
0	1	1	1

큐에 (size == 1) 데이터가 새로 들어올 예정이다
현재 큐에는 (size == 1) 데이터가 1개 있다
→ 새 데이터와 좋은 친구가 1쌍 만들어진다

BOJ3078: 좋은 친구

좋은 친구 쌍을 구하는 방법?
- 새로 들어오는 원소에 대해, 길이에 대한 빈도 수 배열 생성

A	B	C	D	E	F	G



0	1	2	3
0	3	0	0

큐에 (size == 1) 데이터가 새로 들어올 예정이다
현재 큐에는 (size == 1) 데이터가 3개 있다
→ 새 데이터와 좋은 친구가 3쌍 만들어진다

BOJ3078: 좋은 친구

문제 정리

- 좋은 친구는 K등수 이내이다
  - 큐의 사이즈가 K를 초과하면 POP

- 좋은 친구는 같은 길이의 이름을 가지고 있다
  - 큐에 넣을 데이터는 '문자열의 길이'로 지정한다

- 좋은 친구쌍의 수?
  - 큐에 있는 데이터의 빈도수를 기록
  - 큐에 새로 들어올 데이터와 길이가 같은 빈도수를 덧셈

BOJ3078: 좋은 친구

```
int[] friends = new int[21];
Queue<Integer> queue = new LinkedList<>();
for(int i=0; i<n; i++) {
    int next = students[i];
    sum += friends[next];
    friends[next]++;

    queue.add(next);
    if(queue.size() > k) {
        int out = queue.poll();
        friends[out]--;
    }
}
```

next = 큐에 새로 들어올 데이터

새로 만들어질 조합의 수 누적

큐에 있는 데이터 길이 빈도 기록

queue에 학생 추가
k를 초과하면 pop

BOJ9935: 문자열 폭발

문제 요약

- 문자열과 폭발문자열이 주어진다
- 문자열에 폭발문자열이 포함되어 있으면 제거하고 이어 붙여 새로운 문자열을 만든다
- 이어 붙인 문자열에 다시 폭발문자열이 포함되어 있으면 위의 과정을 반복한다
- 1 <= {문자열} <= 1,000,000 | 1 <= {폭발문자열} <= 36
- 남아있는 문자가 없으면 “FRULA”를 출력한다

BOJ9935: 문자열 폭발

입력 데이터
mirkovC4nizCC44 C4

출력 데이터
mirkovniz

BOJ9935: 문자열 폭발

원본 문자열
mirkovC4nizCC44

제거 문자열
C4

(1) mirkovC4nizCC44

(2) mirkovnizCC44

(3) mirkovnizC4

(4) mirkovniz

출력 데이터
mirkovniz

BOJ9935: 문자열 폭발

문제 분석

- 인덱스 순차대로 문자열 폭발을 진행하면?
  - 새로운 문자열에서 폭발할 대상이 만들어질 수도 있다
  - 앞의 문자열을 다시 봐야한다

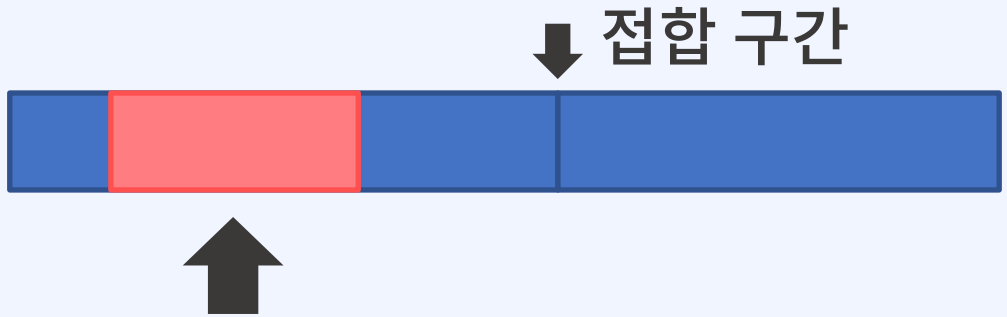
- 문자열 N에서 폭발문자열 M길이의 Sub-String의 조합 수: N – M + 1
- 폭발이 발생할 때마다 다시 처음부터 비교를 하면?: N * (N – M + 1)
  - 1 <= N <= 1,000,000 | 1 <= M <= 36 (시간 초과)

- 폭발이 발생할 때, 적당한 수준만 앞으로 돌아가서 비교해야 한다

BOJ9935: 문자열 폭발

적당한 수준?

문자열에서 {i 번째} 폭발대상을 제거한 뒤에 얼마나 뒤로 돌아가야 할까?



폭발 뒤 문자열을 이어 붙였는데, 그 이전 구간에서 폭발 대상이 등장할 수 있을까?
→ {i -1} 번째 폭발 때 이미 제거 되었어야 한다 (등장할 수 없다)

BOJ9935: 문자열 폭발

문제 분석

문자열의 모음을 스택으로 취급할 수 있다

문자열을 하나씩 추가하면서, 현재 스택의 top이 {i} 라고 할 때
폭발 대상인지 검사를 하려면 {i - m} 부터 동일한 문자인지 비교

폭발 대상으로 확인되면 m회 pop을 진행하고, 위의 과정을 반복

시간복잡도는?
{i} 번째 인덱스마다 m의 길이만큼 앞으로 돌아가서 검사함
→ 한 글자마다 m번의 비교가 포함됨
→ n * m = 1,000,000 * 36

BOJ9935: 문자열 폭발

원본 문자열	제거 문자열
mirkovC4nizCC44	C4

m i r k o v

m i r k o v C

m i r k o v C 4 BOMB

BOJ9935: 문자열 폭발

원본 문자열
mirkovC4nizCC44

제거 문자열
C4

BOJ9935: 문자열 폭발

원본 문자열	제거 문자열
mirkovC4nizCC44	C4

mirkovni

mirkovniz

mirkovnizC

mirkovnizCC

mirkovnizCC4 BOMB

mirkovnizC4 BOMB

BOJ9935: 문자열 폭발

```
Stack<Character> s = new Stack<>();
for(int i=0; i<n; i++) {
    s.add(str.charAt(i));
    if(s.size() < m) continue;

    boolean isSame = true;
    for(int j=0; j<m; j++) {
        if(s.get(s.size() - m + j) != bomb.charAt(j)) {
            isSame = false;
            break;
        }
    }
    if(isSame) {
        for(int j=0; j<m; j++) s.pop();
    }
}
```

1. 스택에 문자 추가
2. 폭발 대상보다 짧으면 무시
3. 스택의 끝에서 m만큼의 길이를 가진 문자열이 같은지 검사
4. 만약 같다면 m길이의 문자를 스택에서 pop

BOJ24268: 2022는 무엇이 특별할까?

문제 요약

- N(10진법) 보다 큰 수를 d진법으로 표기
- 표기한 수 중 [0, d-1] 범위의 모든 숫자가 등장하는 가장 작은 수
- 1 <= N <= 1,000,000,000 | 2 <= d <= 9

BOJ24268: 2022는 무엇이 특별할까?

문제 분석

- 0부터 d-1까지 숫자를 모두 사용?
  - 길이가 d인 수열을 만들어야 함
- 길이가 d <= 9 이므로, 가능한 모든 조합은 최악의 경우에 9!
  - 362,880 개 경우의 수
- 모든 조합을 다 만들어본 뒤, N보다 큰지 비교해도 충분하다

BOJ24268: 2022는 무엇이 특별할까?

순열을 생성하는 방법

1. 인덱스로 사용하는 배열을 선언한다 (number[])
   - 사이즈는 사용할 수의 개수 만큼

2. 재귀의 깊이가 깊어질수록 depth를 1씩 증가한다

3. 반복문을 [0, d) 구간을 순회하며 number[depth]의 수를 {i} 번째 수로 사용한다
   - ex) 순열 13042를 표현한 배열

[0]	[1]	[2]	[3]	[4]
3	1	5	2	4

BOJ24268: 2022는 무엇이 특별할까?

순열을 생성하는 방법

```
public static void nextPermutation(int depth) {
    if(depth == d) {
        for (int i = 0; i < d; i++) {
            System.out.print(number[i] + " ");
        }
        System.out.println();
        return;
    }
    for (int i = 0; i < d; i++) {
        number[depth] = i;
        nextPermutation(depth + 1);
    }
}
```

ex) d = 3

```
0 0 0
0 0 1
0 0 2
0 1 0
0 1 1
0 1 2
0 2 0
0 2 1
0 2 2
1 0 0
1 0 1
1 0 2
1 1 0
1 1 1
...
```

BOJ24268: 2022는 무엇이 특별할까?

이번 문제를 위해 순열생성 함수를 수정해보자

1. 중복되는 수가 사용되지 않도록 해야 한다
   - 사용한 숫자를 기록해두는 Boolean 배열 생성

```
for (int i = 0; i < d; i++) {
    if(!usedNumber[i]) {
        usedNumber[i] = true;
        number[depth] = i;
        findPermutation(depth + 1);
        usedNumber[i] = false;
    }
}
```

BOJ24268: 2022는 무엇이 특별할까?

이번 문제를 위해 순열생성 함수를 수정해보자

2. 앞자리가 0으로 시작하면 안된다
   - depth 의 상태에 따라 반복문의 시작 숫자를 변경한다

```
int start = 0;
if(depth == 0) start = 1;
for (int i = start; i < d; i++) {
   // 순열 생성 코드
}
```

BOJ24268: 2022는 무엇이 특별할까?

이번 문제를 위해 순열생성 함수를 수정해보자

3. 생성한 순열은 10진법 N보다 커야 한다
   - 진법변환 결과를 함수의 결과로 반환한다
   - 반환 결과가 N보다 커지면 재귀를 종료한다

```
if (depth == d) {
    int result = 0;
    for (int i = 0; i < depth; i++) {
        result *= d;
        result += number[i];
    }
    return result;
}
```

```
int num = findPermutation(depth + 1);
if (num > n) return num;
```

BOJ24268: 2022는 무엇이 특별할까?

이번 문제를 위해 순열생성 함수를 수정해보자

3. 생성한 순열은 10진법 N보다 커야 한다
   - 진법변환 결과를 함수의 결과로 반환한다
   - 반환 결과가 N보다 커지면 재귀를 종료한다

```
if (depth == d) {
    int result = 0;
    for (int i = 0; i < depth; i++) {
        result *= d;
        result += number[i];
    }
    return result;
}
```

```
int num = findPermutation(depth + 1);
if (num > n) return num;
```

BOJ1316: 그룹 단어 체커

문제 요약

- N개의 소문자만 포함된 문자열
- 같은 문자가 문자열 내에서 연속되어 사용되는지 확인
  - 중간에 다른 문자가 포함되어 있으면 그룹단어가 아님
- 그룹 단어의 총 개수를 출력
- 1 <= N <= 100, 알파벳 소문자(26자)

BOJ1316: 그룹 단어 체커

문제 분석

- 사용한 문자열을 배열에 체크하며 입력 처리
  - 단, 다음 글자가 동일한 문자라면 체크를 유예

- 이미 체크 되어있는 문자가 들어오면
  그룹단어가 아니도록 판정

쪽지시험 #2

BOJ1316: 그룹 단어 체커

초기화

```
boolean[] check = new boolean[26];
boolean isGroupWord = true;
char[] str = sc.next().toCharArray();
```

```
for (int j = 0; j < str.length - 1; j++) {
    char now = str[j];
    char next = str[j + 1];
    if(now == next) continue;

    if (check[now - 'a']) {
        isGroupWord = false;
        break;
    }
    check[now - 'a'] = true;
}
```

next (str[j+1]) 가 범위를 벗어나지 않도록 str.length -1 까지 비교

다음 글자가 동일한 문자라면 체크를 유예

이미 등장한 문자면 그룹단어가 아닌 것으로 판정

문자를 체크하여 이후에 등장하는지 판별

BOJ1316: 그룹 단어 체커

```
for (int j = 0; j < str.length - 1; j++)
```

next (str[j+1]) 가 범위를 벗어나지 않도록 str.length -1 까지 비교

```
if (check[str[str.length - 1] - 'a']) isGroupWord = false;
```

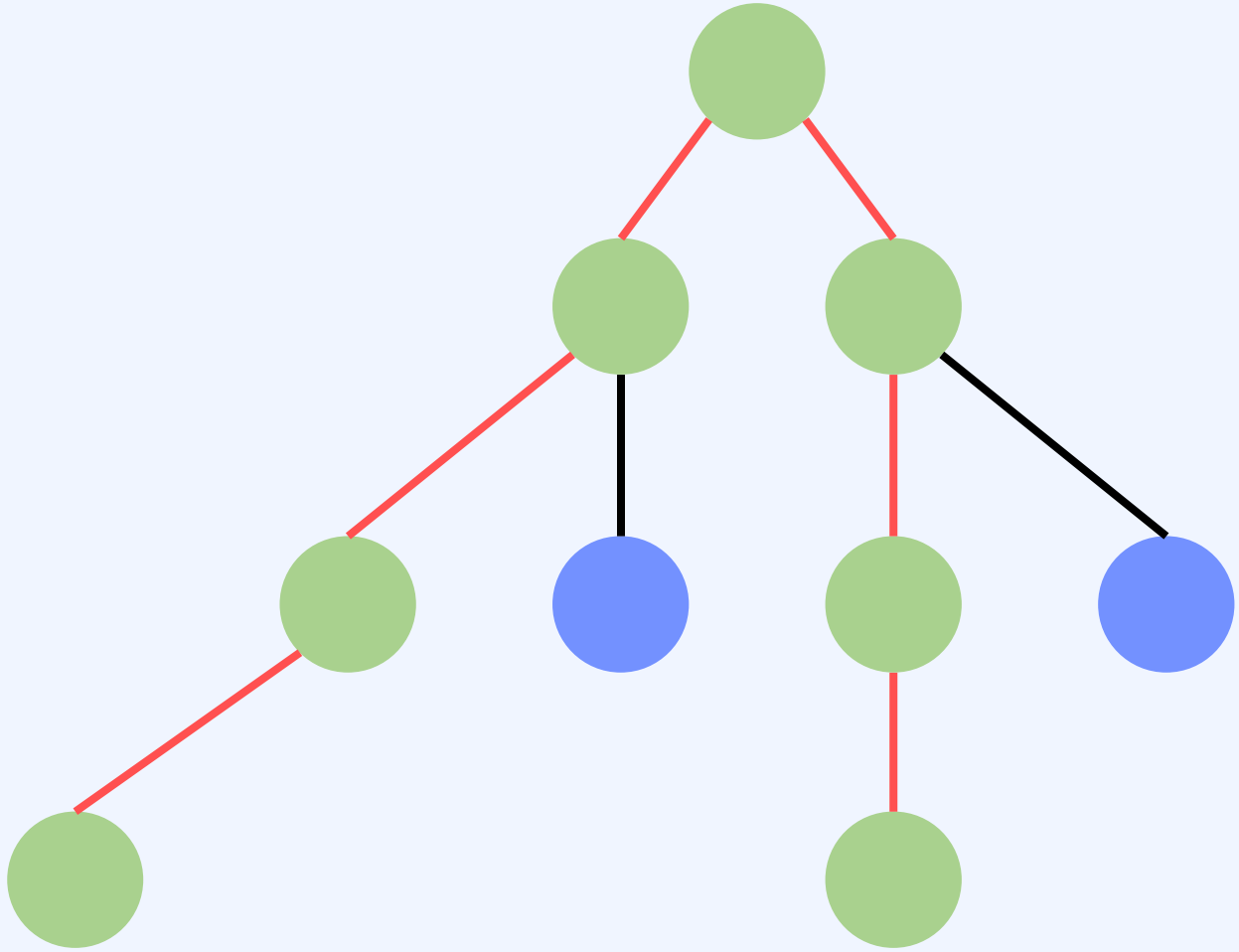
마지막 글자는 반복문 밖에서 판별

BOJ1167: 트리의 지름

문제 요약

- 트리의 지름?
  - 임의의 두 정점 간의 거리 중에서, 가장 먼 거리
- 간선마다 가중치를 부여, 거리는 간선의 모든 가중치 합
- 2 <= V (정점) <= 100,000

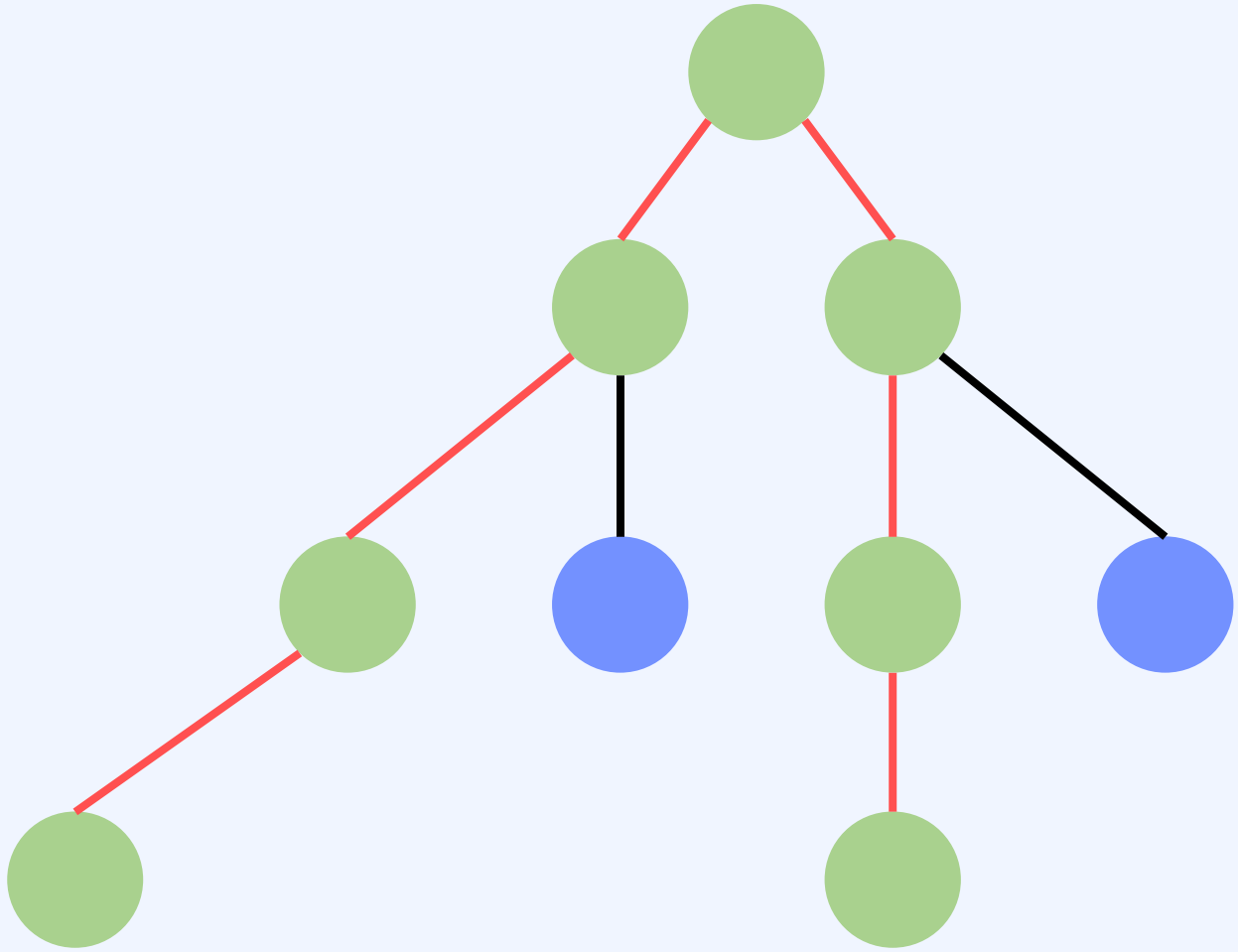
BOJ1167: 트리의 지름



간선의 모든 가중치가 1이라면, 빨간색 간선 구간이 트리의 지름이 된다

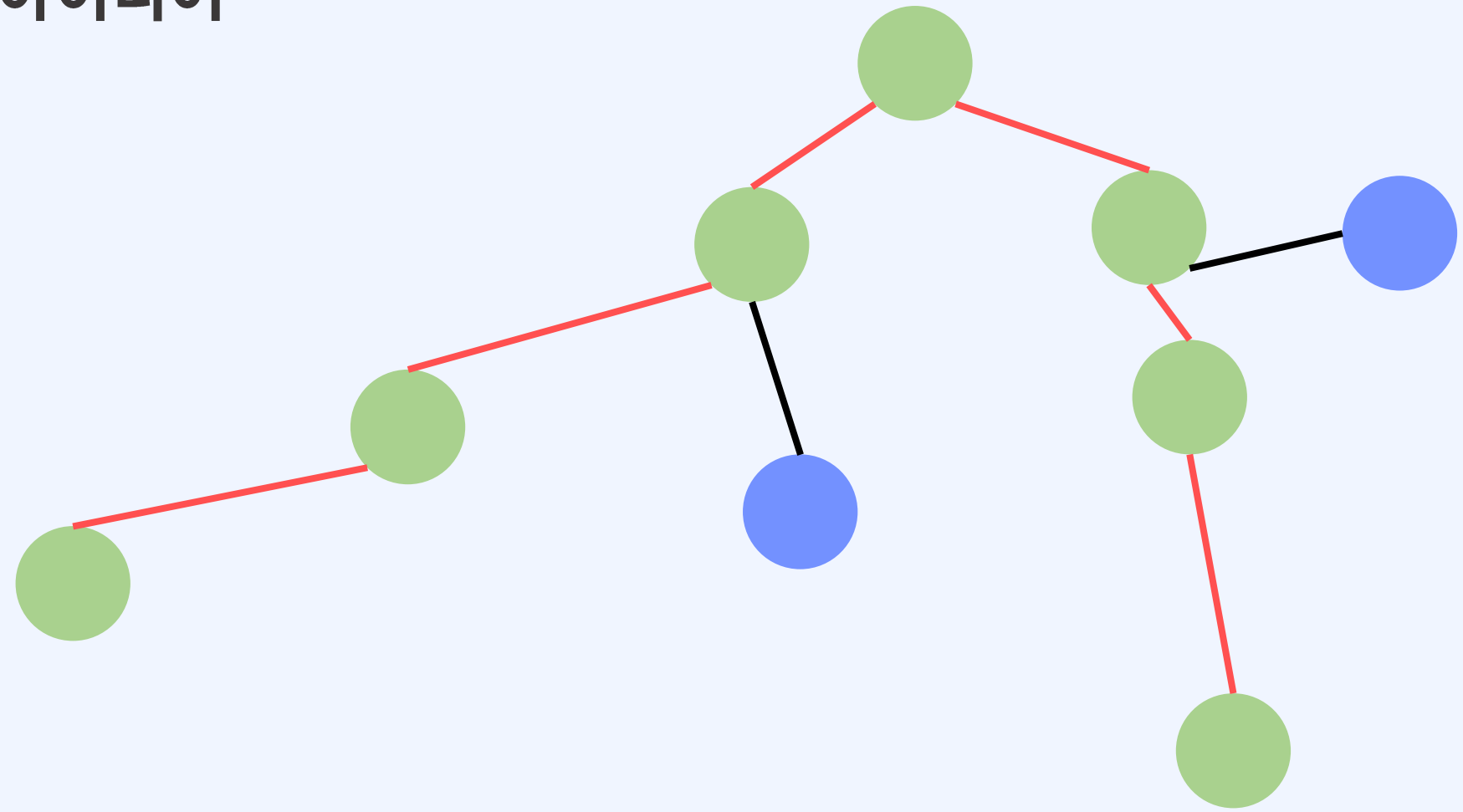
BOJ1167: 트리의 지름

아이디어



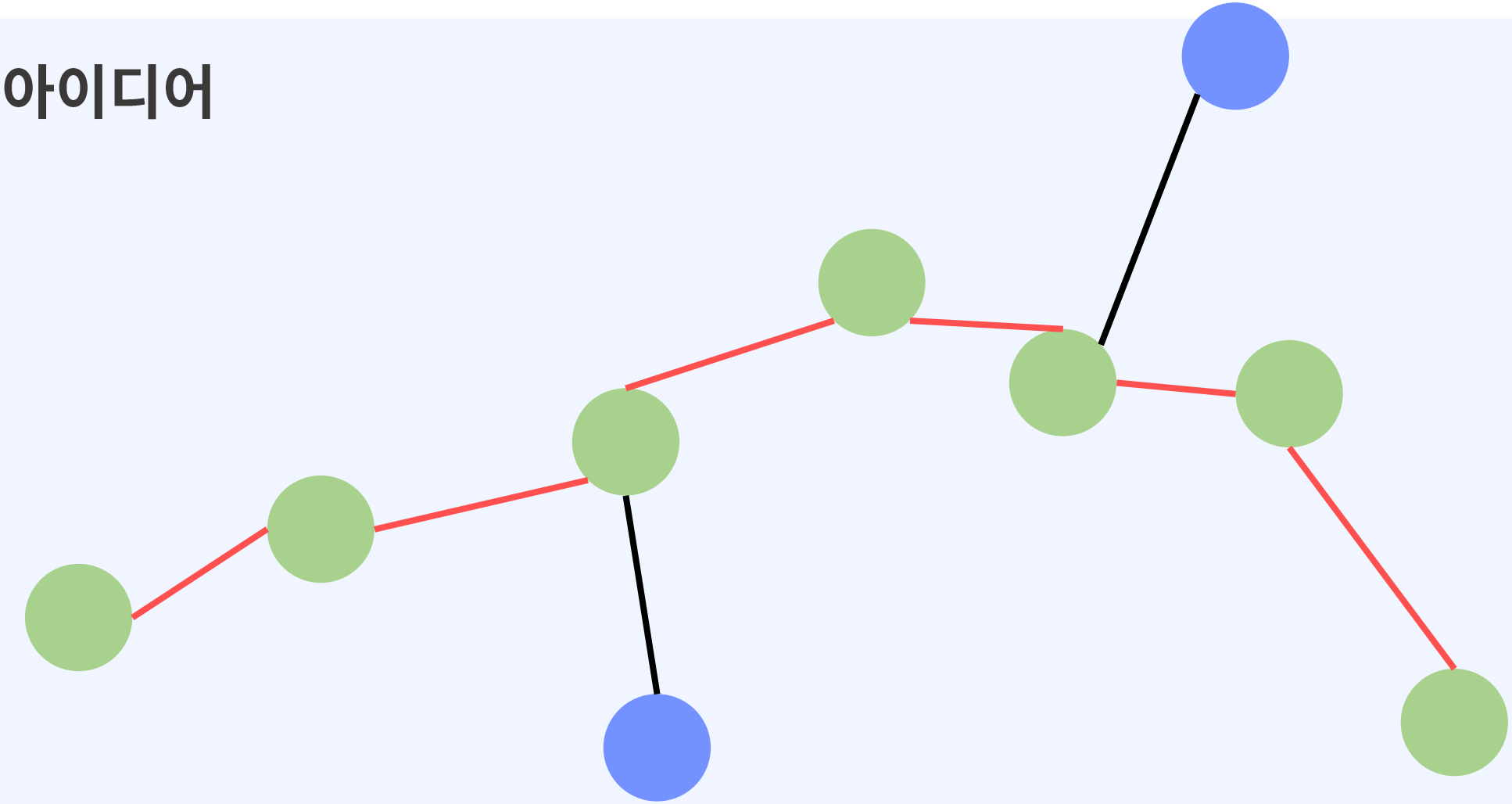
BOJ1167: 트리의 지름

아이디어



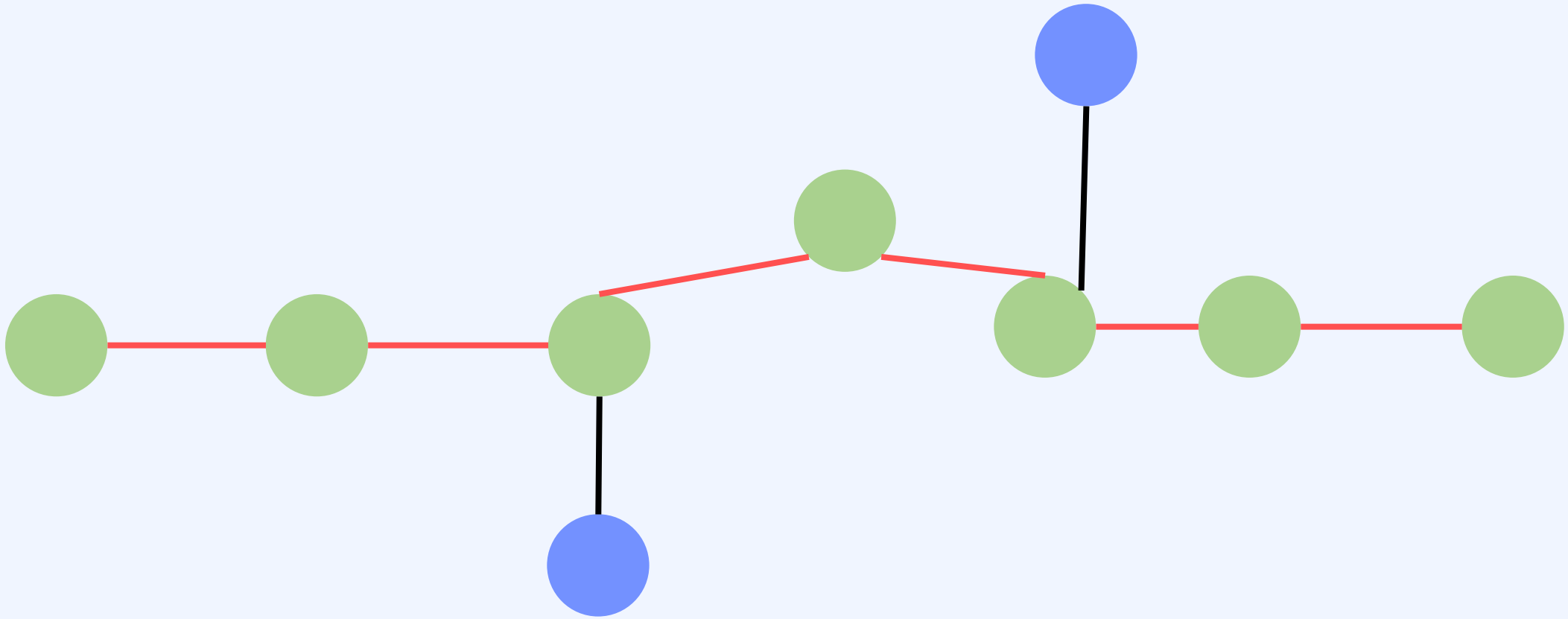
BOJ1167: 트리의 지름

아이디어



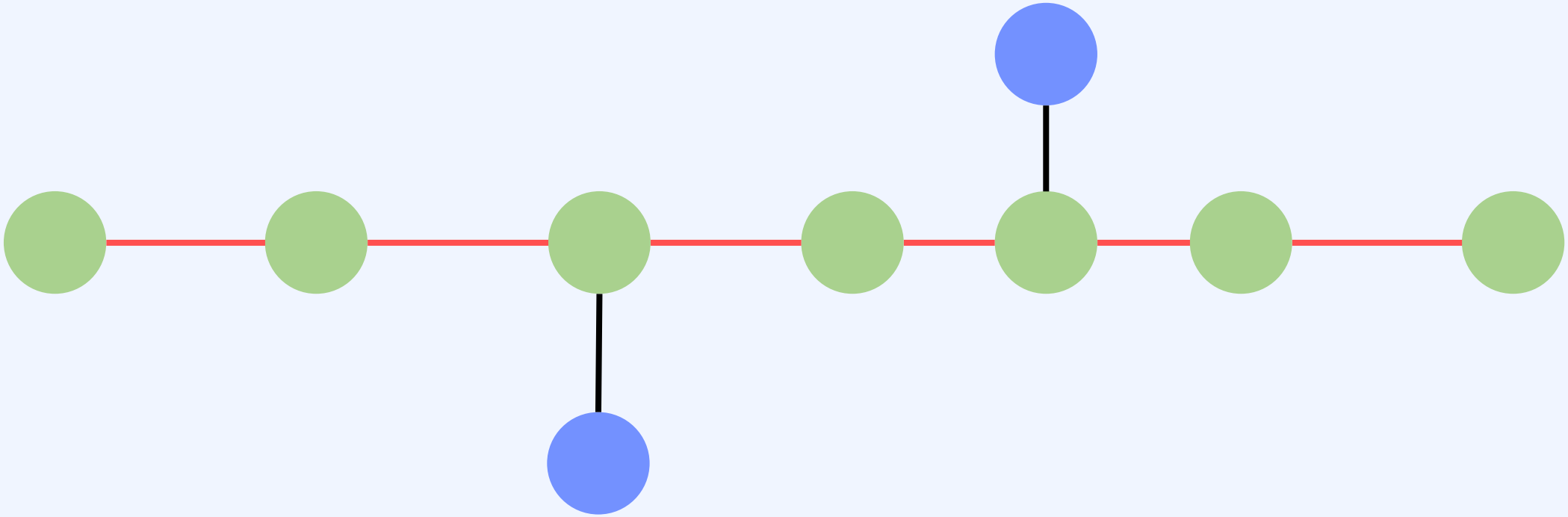
BOJ1167: 트리의 지름

아이디어



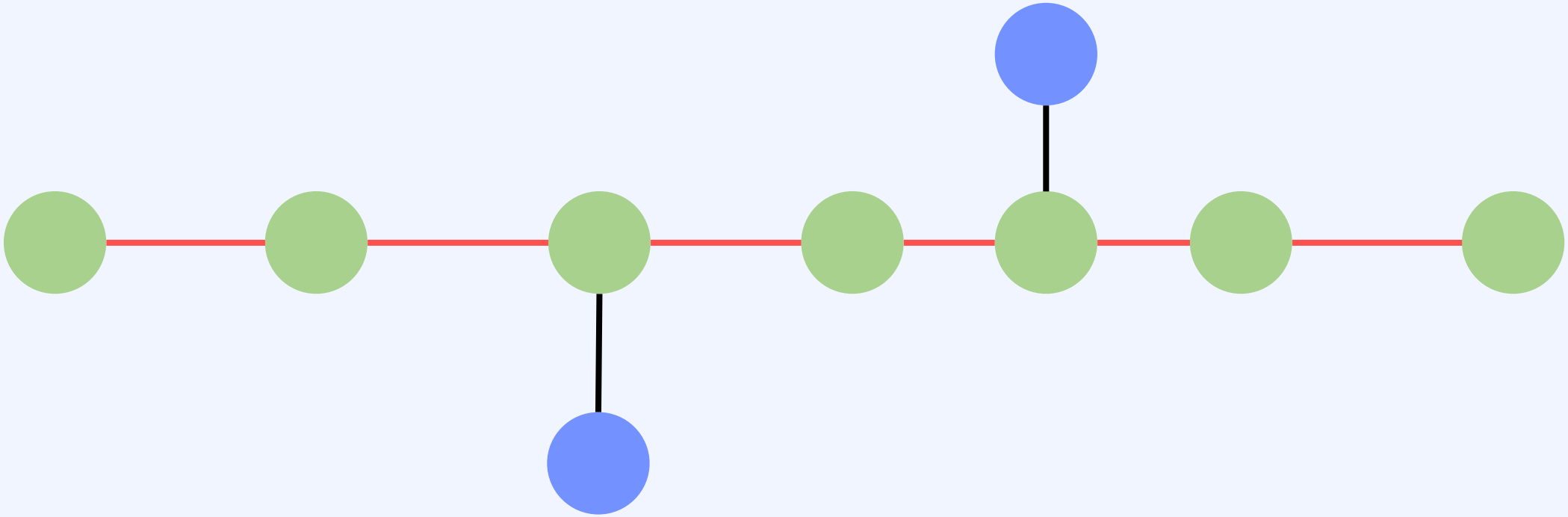
BOJ1167: 트리의 지름

아이디어



BOJ1167: 트리의 지름

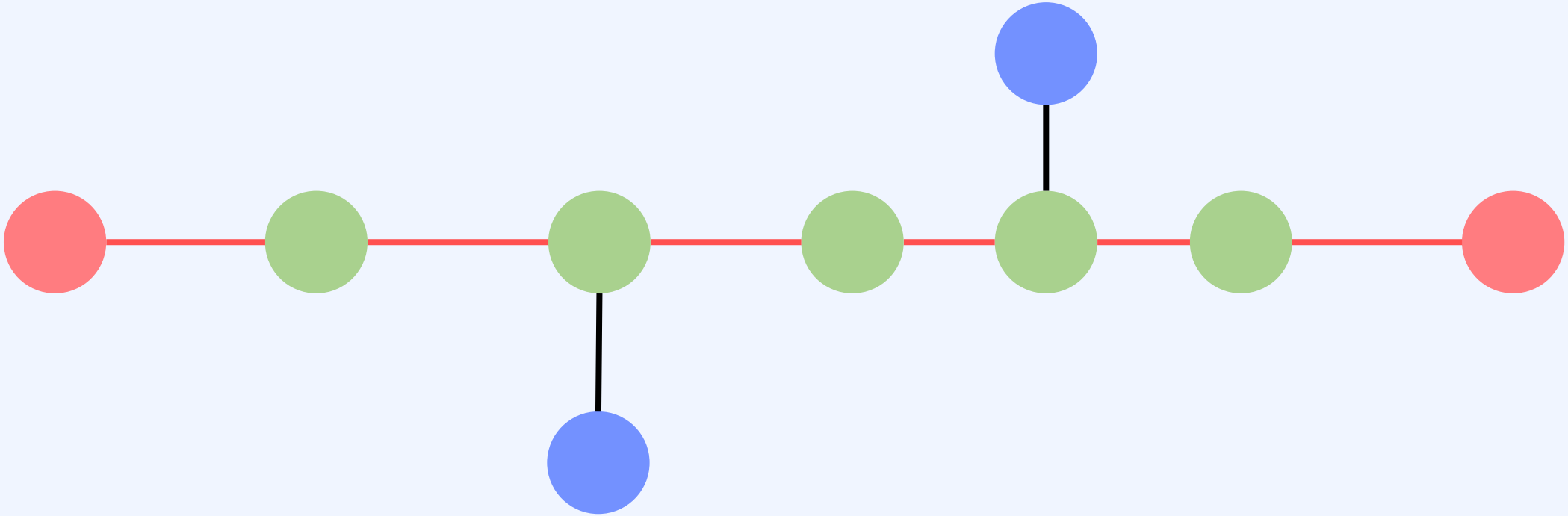
아이디어



트리의 지름에 해당하는 간선은 선분처럼 놓여있다고 생각할 수 있다

BOJ1167: 트리의 지름

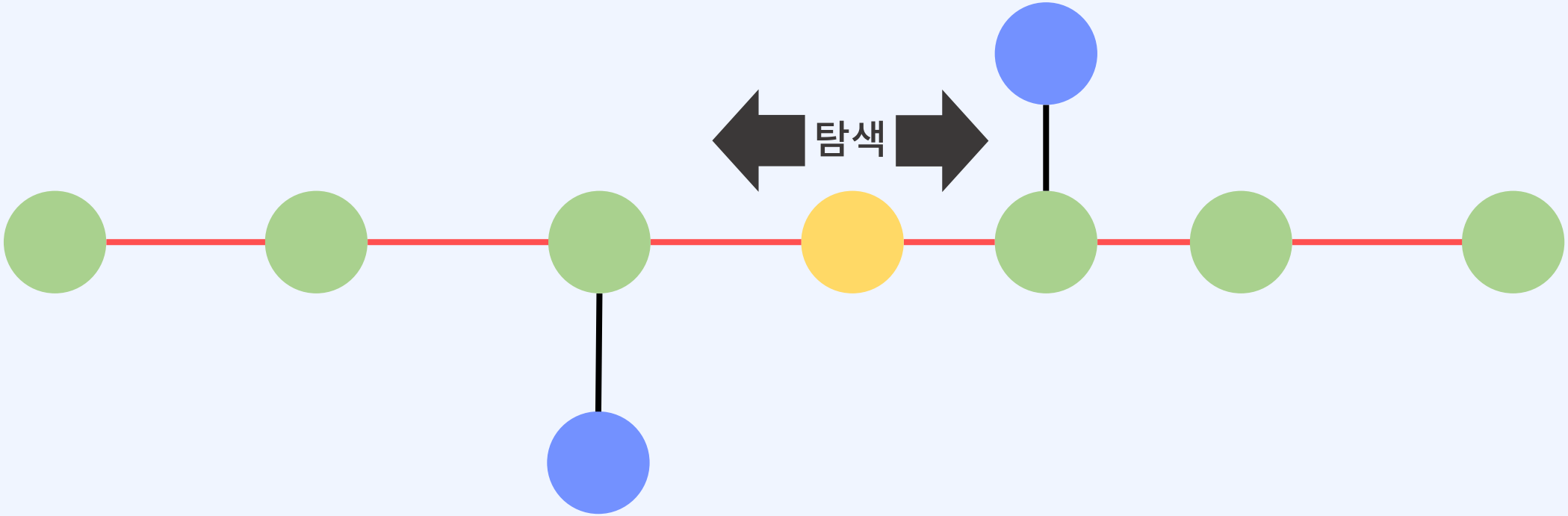
접근방법



선분의 끝에 해당하는 정점을 찾아야 한다 (리프노드)

BOJ1167: 트리의 지름

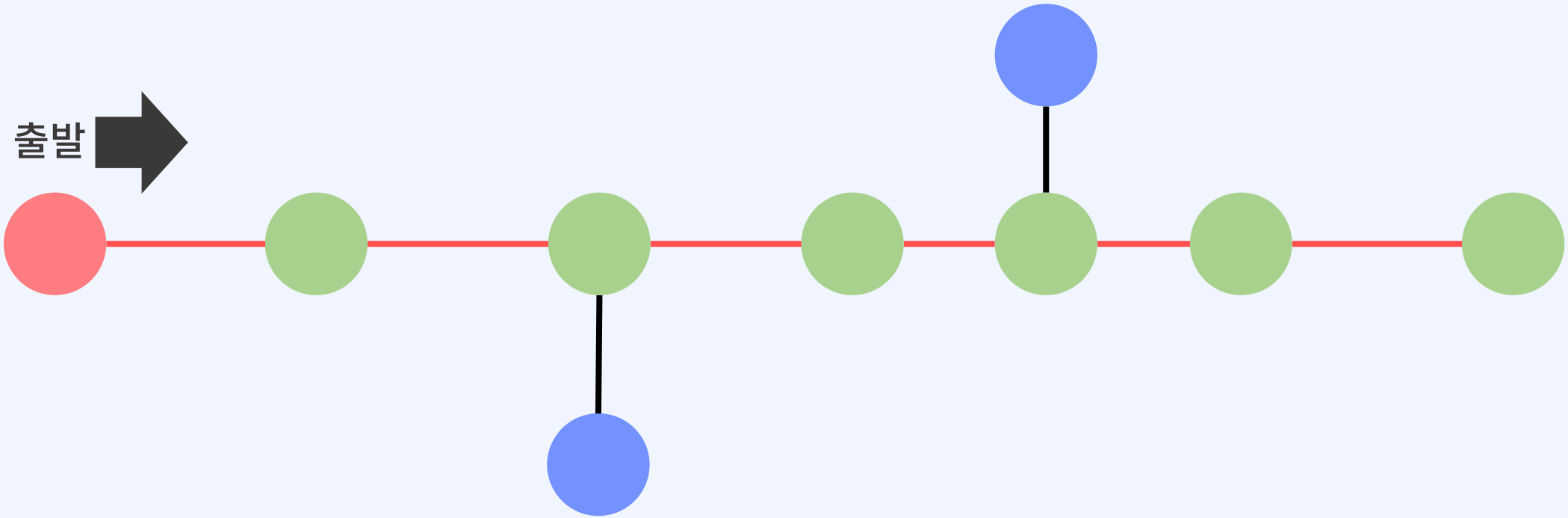
접근방법



임의의 정점 1개를 잡고, 가장 멀리 떨어진 리프 노드를 향해 탐색한다

BOJ1167: 트리의 지름

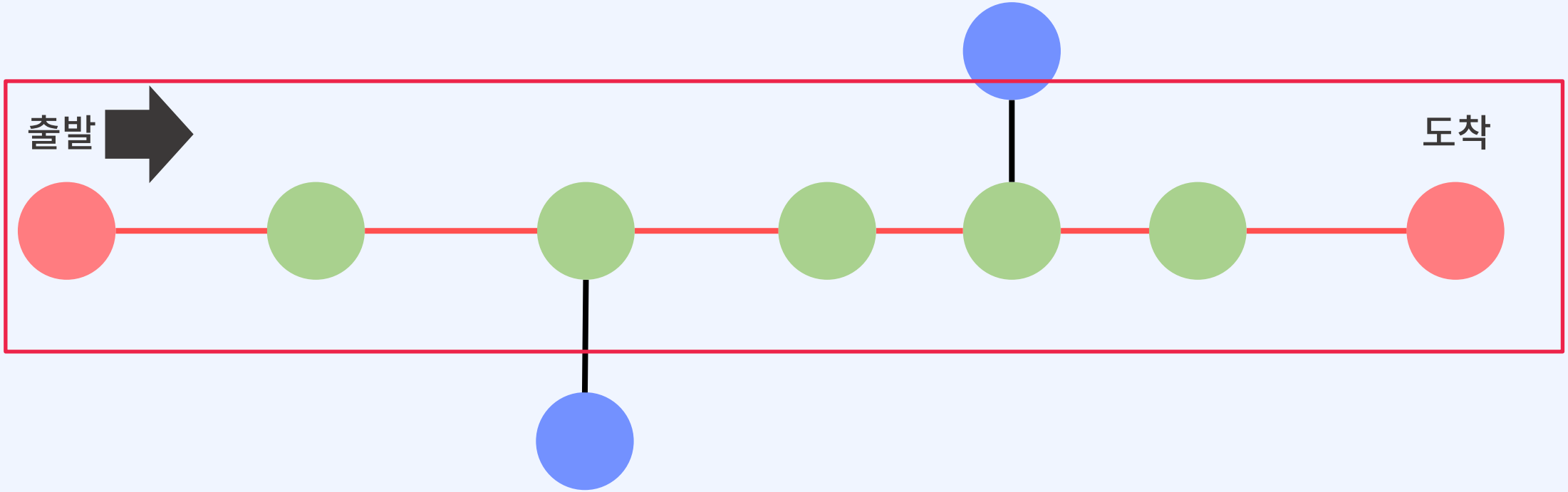
접근방법



가장 멀리 떨어진 리프 노드를 출발점으로 지정하고
해당 노드에서 가장 멀리 떨어진 노드를 찾는다

BOJ1167: 트리의 지름

접근방법



도착한 노드까지의 간선의 가중치 합이 트리의 지름이 된다

BOJ1167: 트리의 지름

입력

노드에는 두가지 정보가 필요하다
- 노드 번호
- 가중치

```
class Node {
    int node;
    int dist;

    public Node(int node, int dist) {
        this.node = node;
        this.dist = dist;
    }
}
```

이를 묶어서 표현할 수 있는 자료형(클래스) 를 만들면
코드를 깔끔하게 작성할 수 있다

BOJ1167: 트리의 지름

그래프 표현 방법

V <= 100,000 으로 인접행렬로 표현하면 메모리를 초과한다

간선 기반 그래프인 인접 리스트 형태로 데이터를 저장해야 한다

```
5
1 3 2 -1
2 4 4 -1
3 1 2 4 3 -1
4 2 4 3 3 5 6 -1
5 4 6 -1
```

[1]	정점 3 / 가중치 2		
[2]	정점 4 / 가중치 4		
[3]	정점 1 / 가중치 2	정점 4 / 가중치 3	
[4]	정점 2 / 가중치 4	정점 3 / 가중치 3	정점 5 / 가중치 6
[5]	정점 4 / 가중치 6		

BOJ1167: 트리의 지름

```
5
1 3 2 -1
2 4 4 -1
3 1 2 4 3 -1
4 2 4 3 3 5 6 -1
5 4 6 -1
```

[1]	정점 3 / 가중치 2		
[2]	정점 4 / 가중치 4		
[3]	정점 1 / 가중치 2	정점 4 / 가중치 3	
[4]	정점 2 / 가중치 4	정점 3 / 가중치 3	정점 5 / 가중치 6
[5]	정점 4 / 가중치 6		

```
List<Node>[] tree = new ArrayList[v+1];
for(int i = 1; i <= v; i++) {
    tree[i] = new ArrayList<>();
}
```

```
for (int i = 0; i < v; i++) {
    int node = sc.nextInt();
    while (true) {
        int next = sc.nextInt();
        if (next == -1) break;
        int dist = sc.nextInt();
        tree[node].add(new Node(next, dist));
    }
}
```

BOJ1167: 트리의 지름

```
for (int i = 0; i < v; i++) {
    int node = sc.nextInt();
    while (true) {
        int next = sc.nextInt();
        if (next == -1) break;
        int dist = sc.nextInt();
        tree[node].add(new Node(next, dist));
        tree[next].add(new Node(node, dist));
    }
}
```

트리는 방향성이 없는 양방향 그래프로
역방향 간선 정보도 추가해야 한다

쪽지시험 #2

BOJ1167: 트리의 지름

```
static int maxDepth = 0;
static int maxDepthLeaf = 1;
```

```
public static void findMaxDepthLeaf(int node, int depth) {
    if (depth > maxDepth) {
        maxDepth = depth;
        maxDepthLeaf = node;
    }
    for (Node next : tree[node]) {
        if(visited[next.node] == 1) continue;
        visited[next.node] = 1;
        findMaxDepthLeaf(next.node, depth + next.dist);
    }
}
```

임의의 정점 node 로부터, 방문하지 않은 정점을 탐색한다
탐색 중에 가장 먼 거리와 해당 정점을 갱신 한다

BOJ1167: 트리의 지름

```
// 1차 탐색: 임의의 노드에서 가장 먼 노드를 찾는다.
maxDepth = 0;
visited[1] = 1;
findMaxDepthLeaf(1, 0);

// 2차 탐색: 가장 먼 노드에서 다시 가장 먼 노드를 찾는다.
for(int i = 1; i <= v; i++) visited[i] = 0;
maxDepth = 0;
visited[maxDepthLeaf] = 1;
findMaxDepthLeaf(maxDepthLeaf, 0);
```

1차 탐색은 임의의 정점(1번 노드)에서 가장 멀리 떨어진 리프 노드를 찾는다
2차 탐색은 1차에서 찾은 리프 노드에서, 가장 멀리 떨어진 다른 리프 노드를 찾는다
두 리프 노드간 간선 가중치의 합이 트리의 지름이 된다