Chapter. 03

알고리즘

모의 코딩테스트 풀이
Mock Coding Test

FAST CAMPUS
ONLINE
알고리즘 공채 대비반I

강사. 류호석

Life Changing Education

Chapter. 03

모의 코딩테스트 풀이 - 1
Mock Coding Test Solution - 3

FAST CAMPUS
ONLINE
류호석 강사.

I 주의사항

실제 시험이라고 생각하고 5시간을 고정시켜 놓은 상태로

https://www.acmicpc.net/category/detail/2338

위 대회를 직접 경험해보신 다음에 이후 풀이 영상을 보세요.

어떤 부분이 어려운 것인지, 생각의 방향이 어떻게 다른 지를 직접 느껴보셔야 합니다.

11. 빌런 호석

난이도: 2

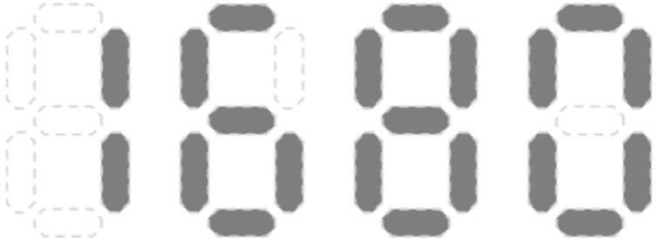
$1 \leq X \leq N < 10^K$
$1 \leq K \leq 6$
$1 \leq P \leq 42$

I 출제 의도

- **올바른 접근 방법을** 떠올리는가?
- **시간 & 공간 복잡도는** 제대로 계산하였는가?
- 문제를 **편하게** 구현하였는가?

I 접근 방법

X 층을 Y 층으로 바꿀 수 있는가?

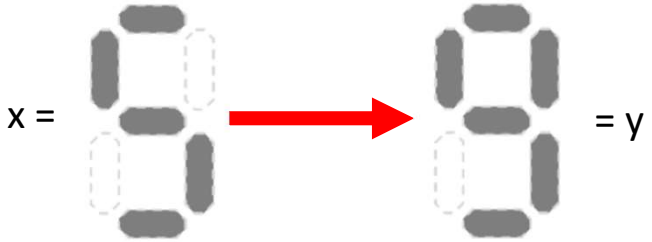
X = 

Y = 

필요한 변환 횟수 = (1 ➔ 0) + (6 ➔ 5) + (8 ➔ 0) + (0 ➔ 1)

이 횟수가 P **이하**인가? P **초과**인가?

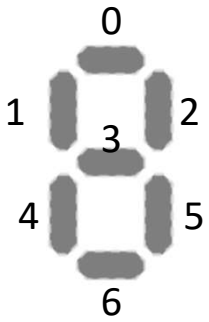
I 접근 방법 – 필요한 함수



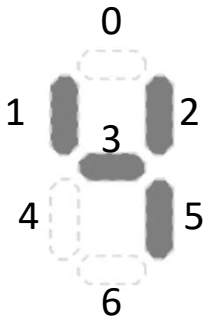
diff_one(x, y) := 숫자 x 를 y 로 바꾸는 횟수

Ex) diff_one(5, 9) = 2

I 접근 방법 – 구현 방법

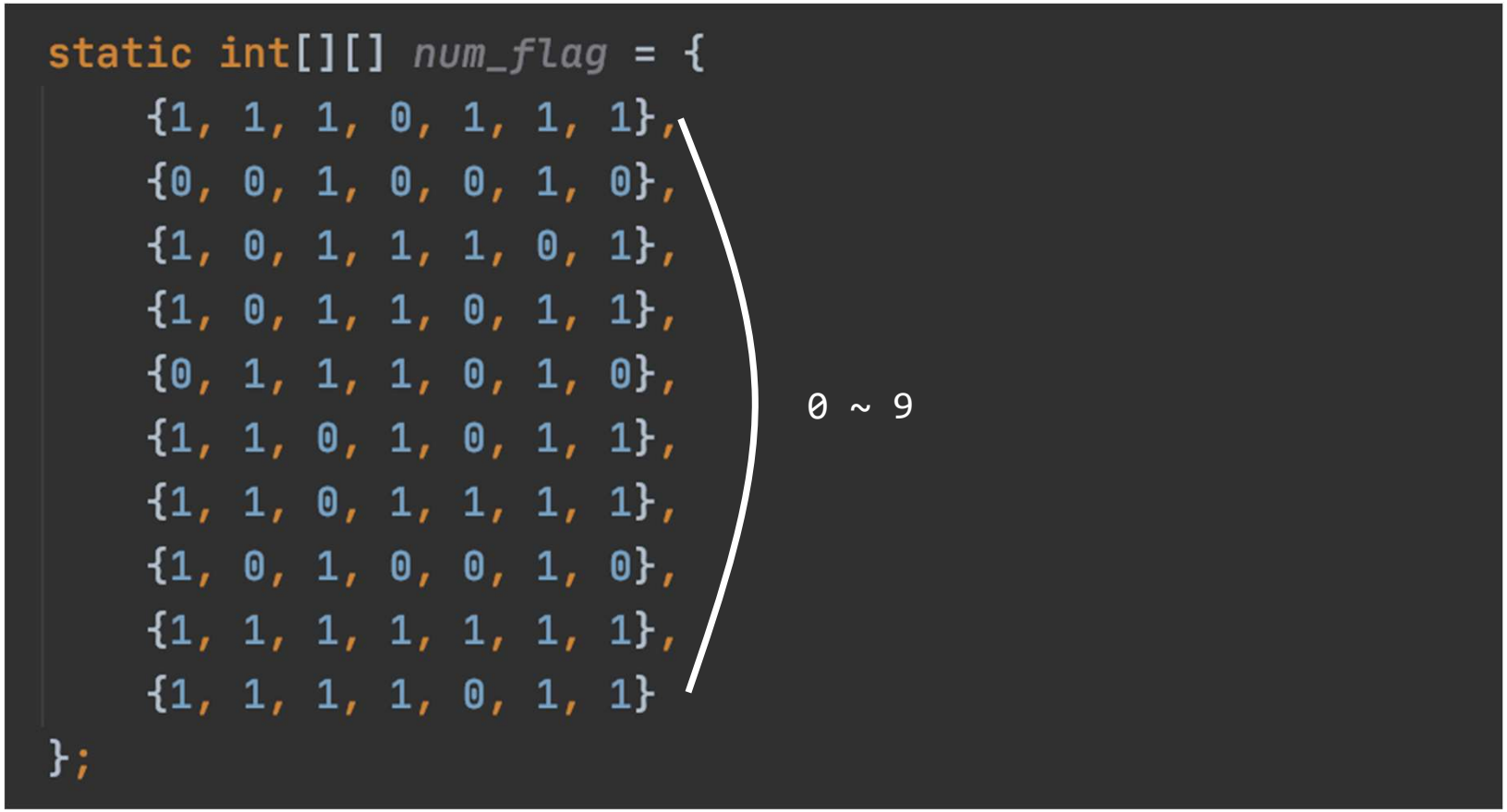


0	1	2	3	4	5	6
1	1	1	1	1	1	1



0	1	2	3	4	5	6
0	1	1	1	0	1	0

I 접근 방법 – 구현 방법



I 총 정리

- diff_one 함수 구현하기
- 이를 이용한 diff 함수 구현
- Y 를 1부터 N까지 바꿔보면서 변환 횟수가 P 이하인지 확인하기

- 총 시간 복잡도는 O(N * K) 이다.

I 구현

```java
// 숫자 x와 숫자 y의 LED 차이 개수
static int diff_one(int x, int y) {
    int res = 0;
    /* TODO */
    return res;
}

// 수 x와 수 y의 LED 차이 개수
static int diff(int x, int y) {
    int res = 0;
    /* TODO */
    return res;
}

static void pro() {
    int ans = 0;
    /* TODO */
    System.out.println(ans);
}
```

12. 정보 상인 호석

난이도: 3

$1 \leq$ 쿼리 개수, $Q \leq 100{,}000$
$\sum k \leq 1{,}000{,}000$

I 출제 의도

- 문제가 요구하는 **상황을 이해**했는가?

- **문자열**을 잘 다루는가?

- **필요한 연산**을 정의할 줄 아는가?

- 자료구조를 나열하고 시간 복잡도를 따져서 **최선의 자료구조**를 선택할 수 있는가?

I 접근 방법

먼저, 고릴라들의 이름을 어떻게 다룰 것인가?

1. 문자열 그 자체를 이용하는 경우

2. 숫자로 변경해서 이용하는 경우

사람마다 익숙한 것으로!

I 필요한 연산 정의

필요한 연산은?

1. 고릴라 G_x 가 $\{C_1, C_2, \ldots, C_k\}$의 정보를 획득한다.

2. 고릴라 G_x 가 가진 정보 중 가장 비싼 b 개를 구매한다.

I 연산으로부터 자료 구조 정의하기

고릴라 G_x 가 들고 있는 자료 구조에 필요한 연산은?

1. 자료 구조에 $\{C_1, C_2, \ldots, C_k\}$의 정보를 추가한다.

2. 자료 구조에 들어있는 자료 중에서 가장 비싼 b 개를 추출한다.

I 연산으로부터 자료 구조 정의하기

만약 일반 **배열(ArrayList)**을 사용한다면? (N: 자료 개수)

1. 자료 구조에 $\{C_1, C_2, \dots, C_k\}$의 정보를 추가한다. ➔ $\boldsymbol{O(k)}$

2. 자료 구조에 들어있는 자료 중에서 가장 비싼 b 개를 추출한다.

   ➔max 를 찾기 & max를 삭제하기 = O(N)

   ➔$\boldsymbol{O(N * b)}$

I 연산으로부터 자료 구조 정의하기

최악을 최선으로 해주는 자료구조는 ***Max Heap*** 이다. (N: 자료 개수)

1. 자료 구조에 $\{C_1, C_2, \dots, C_k\}$의 정보를 추가한다. ➔ $\boldsymbol{O(k * \log N)}$

2. 자료 구조에 들어있는 자료 중에서 가장 비싼 b 개를 추출한다.

   ➔max 를 찾기 & max를 삭제하기 = $O(\log N)$

   ➔$\boldsymbol{O(b * \log N)}$

I 총 정리

- 고릴라 마다 가지고 있는 정보의 가치들을 자료 구조(Max Heap)에 담아두고 있다.

- 총 시간은 정보를 삽입하고 삭제하는 과정에서 소모되는 시간이다. 정보가 많아야 100만개가 삽입 & 삭제 되기 때문에 $O(100만 \log 100만)$ 에 비례하는 시간이 걸리고, 시간 안에 충분히 나온다.

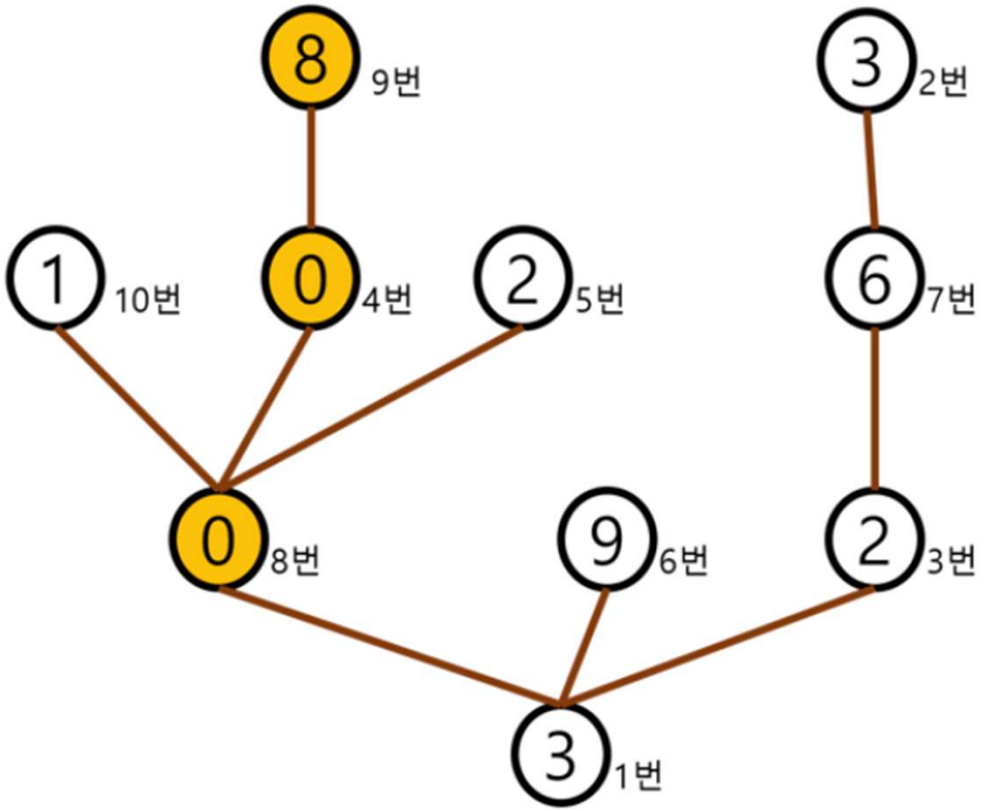
I 구현

```
// 이름을 고유 번호로 변경시켜주는 함수
static int get_pid(String name){
    /* TODO */
}

static void pro() {
    long ans = 0;

    /* TODO */

    System.out.println(ans);
}
```

13. 트리 디자이너 호석

난이도: 4

$1 \leq$ 정점 개수, $N \leq 100{,}000$



I 출제 의도

- **Rooted Tree** 구조에 익숙한가?
- 완전 탐색에서 동적 프로그래밍으로의 **연결**을 지을 수 있는가?
- **동적 프로그래밍** 풀이를 스스로 수행할 수 있는가?

I 생각의 흐름 – 1. 정답의 최대치

가능한 경우가 가장 많은 입력은?

정점 10만개가 일렬로 존재하고, 모두 0인 경우

➔정점을 어떻게 선택해도 조건을 만족하므로 2^{100000} 가지 가능하다.

I 생각의 흐름 – 2. 동적 프로그래밍

모든 경우를 하나씩 찾으면 당연히 “시간 초과”를 받을 것을 예상하기 쉽다.

이런 경우 동적 프로그래밍 접근을 시도해 볼 가치가 있다.

I 생각의 흐름 – 3. 문제 정의

Dy[i][k] := i번 정점을 root로 하는 subtree에서, 조건을 만족하게 선택하면서 마지막 숫자가 k 인 경우의 수

정답 := Dy[1][0] + … + Dy[1][9]

초기화 := Dy[i][a[i]] = 1 (i번 정점만 선택하는 경우)

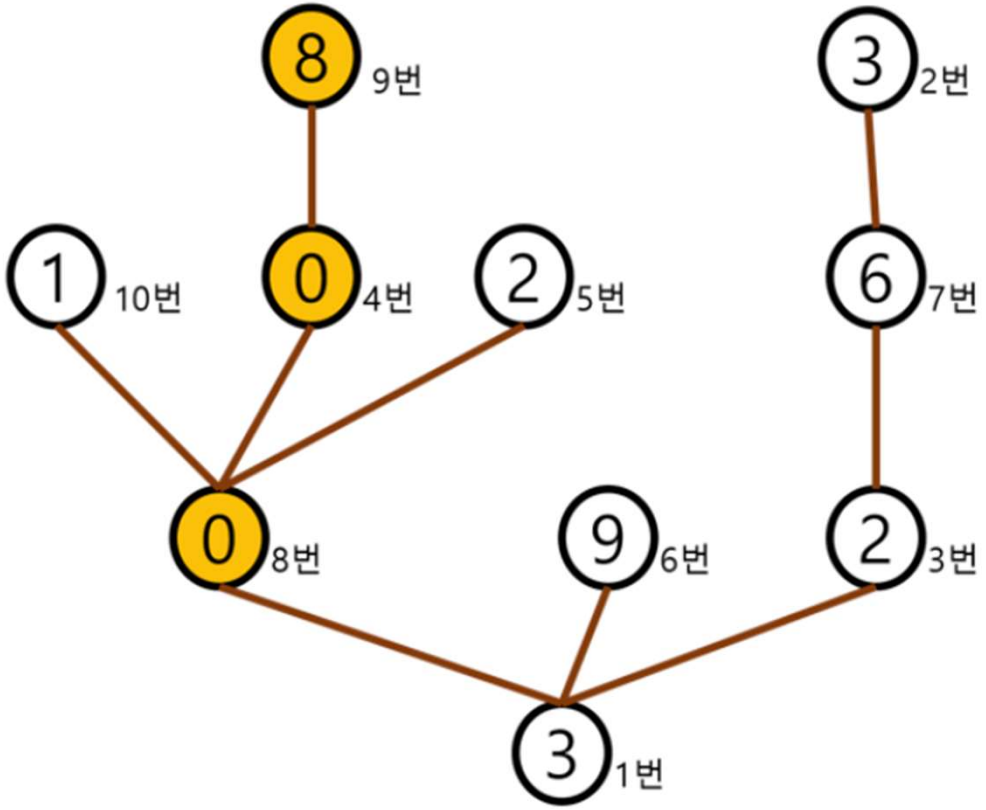
I 생각의 흐름 – 4. 점화식

Dy[i][…]을 계산하기 전에 i의 모든 자식 정점에 대해 문제를 이미 해결했다고 하자.

1. i번 정점을 선택하는 경우

2. i번 정점을 선택하지 않는 경우

두 가지를 모두 고려해서 Dy[i][…]에 반영하면 된다.



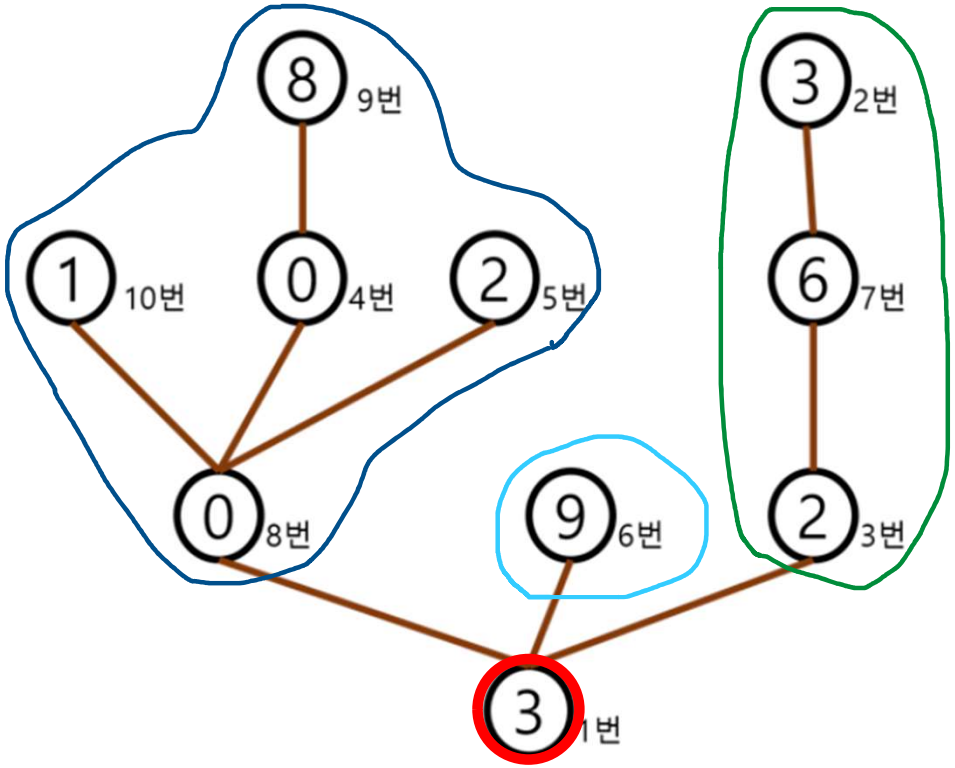
I 생각의 흐름 – 4. 점화식

1. i번 정점을 선택하는 경우

j 가 i의 자식 정점 중 하나라고 하면
Dy[i][a[i]] 는 자식 정점에서 a[i] 이상으로 끝났어야 한다.

예를 들어 i=1 이라면

```
Dy[1][3] +=
        Dy[8][3…9] +
        Dy[6][3…9] +
        Dy[3][3…9]
```



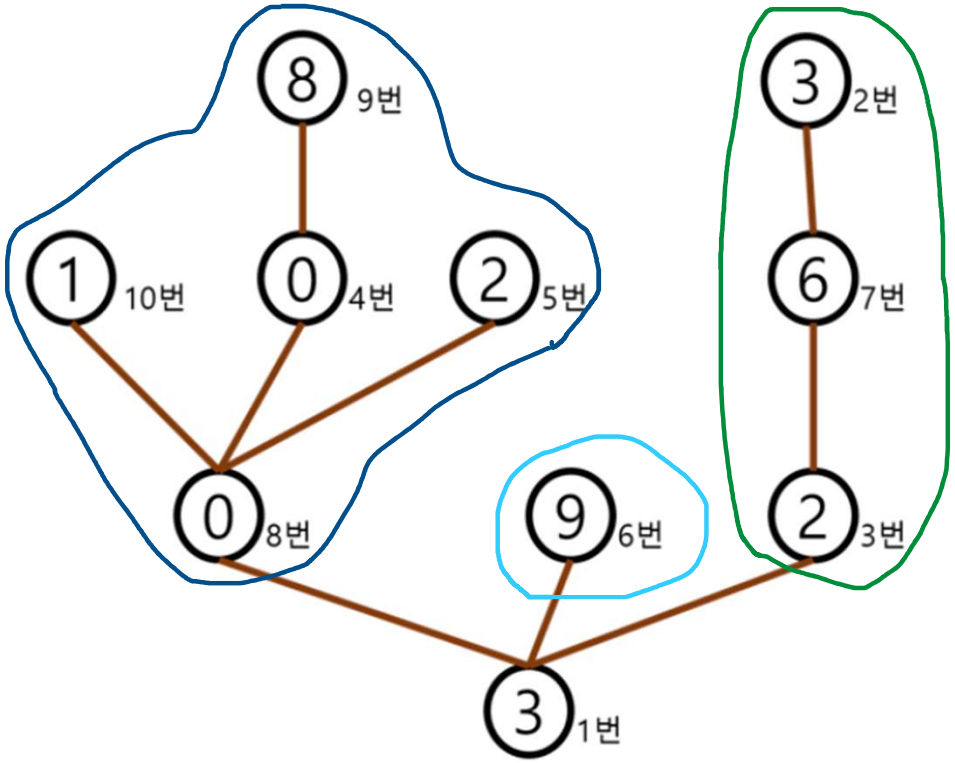
I 생각의 흐름 – 4. 점화식

2. i번 정점을 선택하지 않는 경우

j 가 i의 자식 정점 중 하나라고 하면 Dy[i][a[i]] 는 자식 정점에서 아무 숫자로 끝나도 상관없다.

예를 들어 i=1 이라면 k=0…9 에 대해

```
Dy[1][k] +=
        Dy[8][k] +
        Dy[6][k] +
        Dy[3][k]
```



I 총 정리

- Rooted Tree에서의 Dynamic Programming

- 시간 복잡도는 모든 정점을 한 번씩 보기 때문에 O(N)이다.

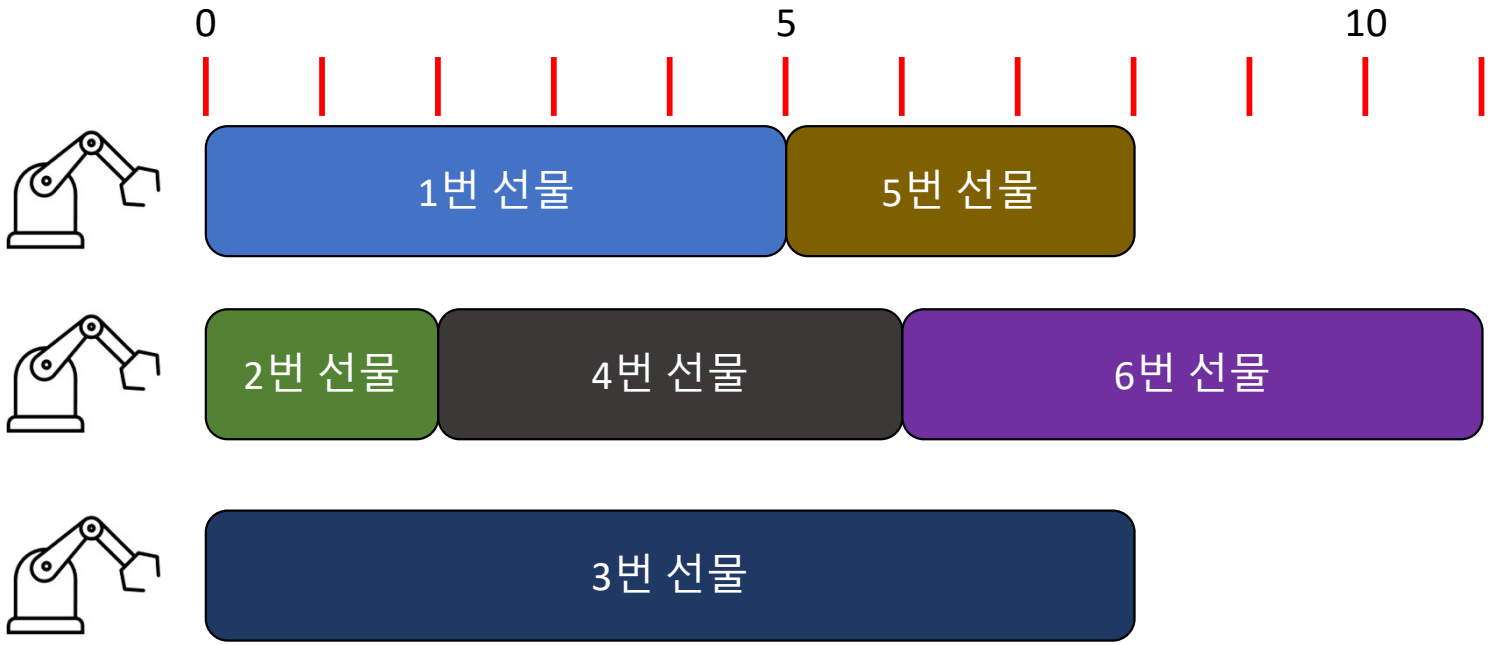
I 구현


```
// dfs(x, par) := 정점 x 의 부모가 par 였고, x-subtree의 dy 배열을 계산하는 함수
static void dfs(int x, int par) {
    /* TODO */
}

static void pro() {
    // 1 번 정점이 ROOT 이므로, 여기서 시작해서 Tree의 구조를 파악하자.
    /* TODO */
}
```

14. 공정 컨설턴트 호석

난이도: 4

$1 \leq$ 선물 개수, $N \leq 100{,}000$
$1 \leq$ 남은 시간, $X \leq 10^9$



I 출제 의도

- 공정 라인의 개수와 마감 시간의 관계를 관찰했는가?

- Parametric Search 기법을 떠올리고 올바르게 적용시켰는가?

I 생각의 흐름 – 1. 정답의 최대치

어떤 입력이 주어져도 공정 라인이 선물의 개수만큼 존재하면 성공적으로 제작할 수 있다.

따라서 정답의 최대치는 N의 최대치인 10만이다.

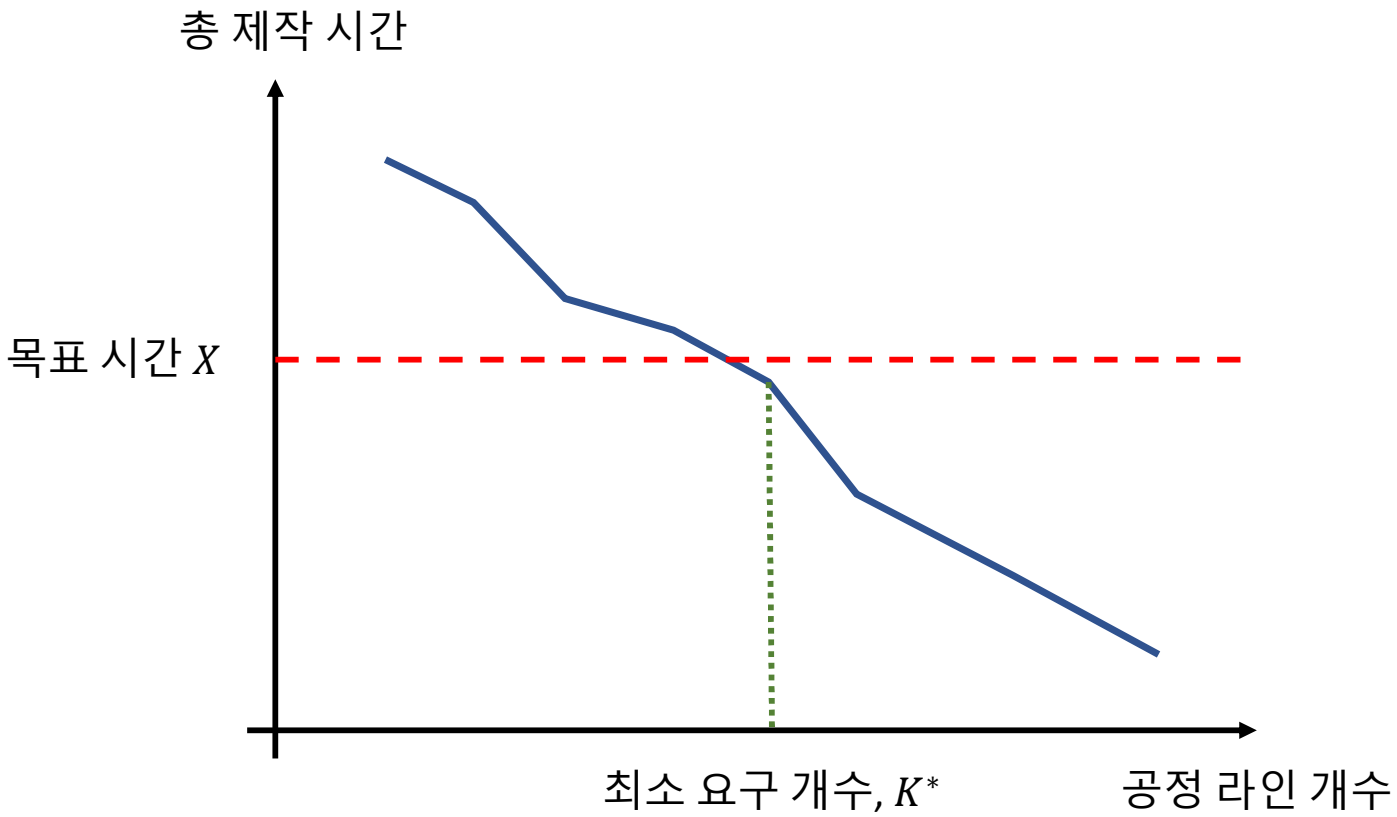
I 생각의 흐름 – 2. 관찰

공정 라인의 개수를 최소화 시켜야 한다.

단, 총 제작 시간에 제한이 걸려 있다.

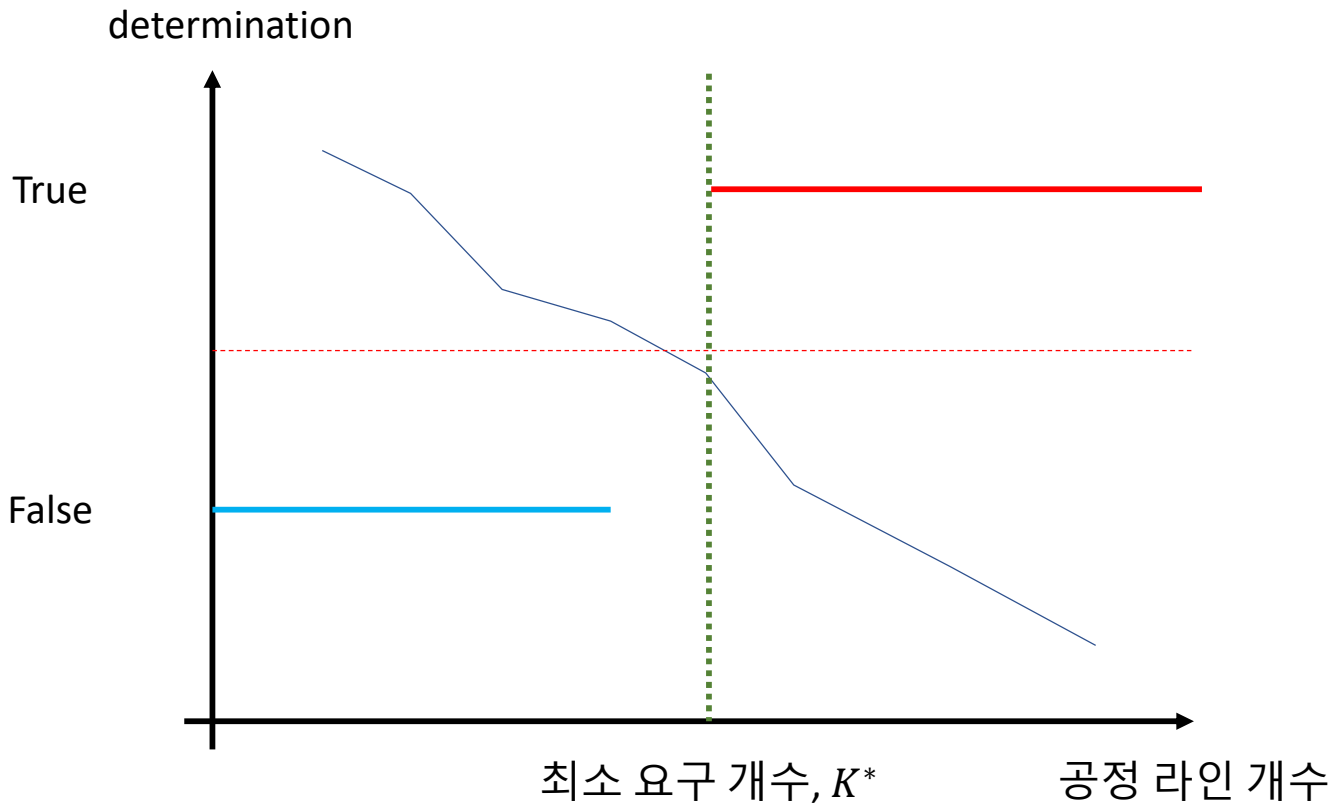
그렇다면 공정 라인의 개수와 총 제작 시간은 무슨 관계인가?

I 생각의 흐름 – 2. 관찰



I 생각의 흐름 – 3. 문제 정의

`determination(K): 공정 라인을 K개 사용하면 시간을 맞출 수 있는가?`



I 생각의 흐름 – 4. Parametric Search

`determination`(K): 공정 라인을 K개 사용하면 시간을 맞출 수 있는가?

공정 라인 개수가 정해져 있는 경우에는 모든 것이 *결정적으로* (*deterministic하게*) 진행되기 때문에 **"시뮬레이션"** 문제로 변경된다.

만약 한 번의 K에 대한 대답을 $O(T)$ 에 할 수 있다면?

이분 탐색을 통해 $O(\log N)$번의 수행이 이뤄지므로 $O(T \log N)$ 이면 K^*를 계산할 수 있다.

I 생각의 흐름 – 4. Parametric Search

determination(K): 공정 라인을 K개 사용하면 시간을 맞출 수 있는가?

K가 정해졌기 때문에, 1번부터 N번 선물을 차례대로 제작하면 된다.

필요한 연산

- 공정 라인 중에 사용 시간이 가장 적은 것을 찾는다.
- 특정 공정 라인의 사용 시간을 X 만큼 증가시킨다.

두 연산을 모두 빠르게($O(\log K)$) 해주는 Min Heap을 사용하자.

I 총 정리

- K^*를 Parametric Search를 통해 찾는다.

- K에 대해서
  - 공정 라인들의 사용 시간을 저장할 Min Heap 생성
  - 0을 K번 삽입
  - N개의 선물을 제작해보고 목표 시간 X와 비교
  - $O(N \log K)$

- 총 시간은 $O(N \log K \log N)$ 이다.

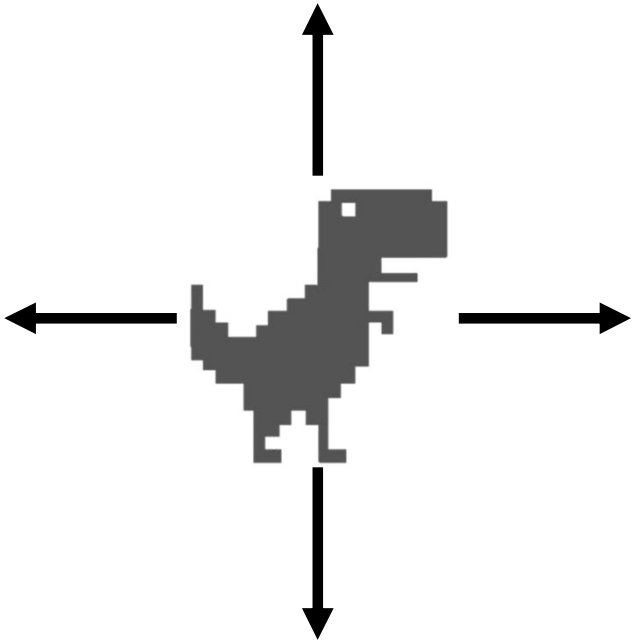
I 구현

```java
// 공정 라인을 num개 사용하면 X 시간 안에 전부 만들 수 있는가?
static boolean determination(int num) {
    /* TODO */
}

static void pro() {
    int L = 1, R = n, ans = n;
    // Parametric search를 통해 최적의 K* 찾기
    /* TODO */
    System.out.println(ans);
}
```

15. 호석 사우루스

난이도: 5

$1 \leq N, M \leq 100$



3K 번째 이동

3K+1 번째 이동

3K+2 번째 이동

I 출제 의도

- 최단 시간 키워드를 통해 **접근 방향**을 잡았는가?

- 문제의 요구 조건에 맞게 **거리 배열 정의**를 했는가?

- 구현을 **편하게** 했는가?

I 생각의 흐름 – 1. 키워드 발견

자신의 철칙을 지키되, 아픈 건 싫어하는 호석사우루스를 도와서 탈출구까지의 최소 충격량을 구해주자!

주어진 문제를 “**그래프 문제**”로 변환시켜서

“**최단 거리**” 계산으로 방향을 잡아보자.

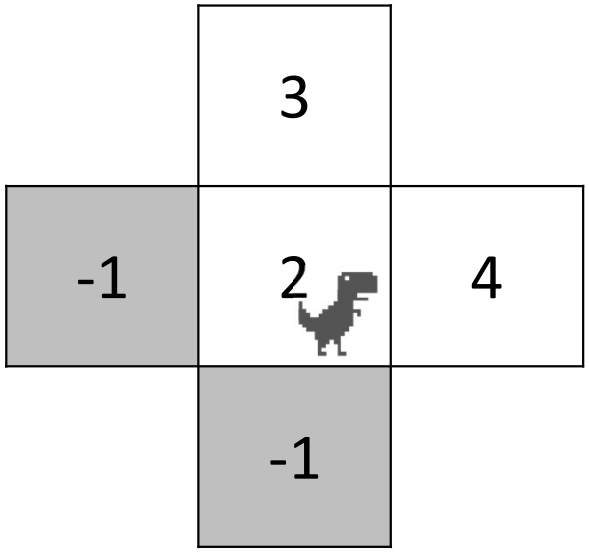
I 생각의 흐름 – 1. 키워드 발견

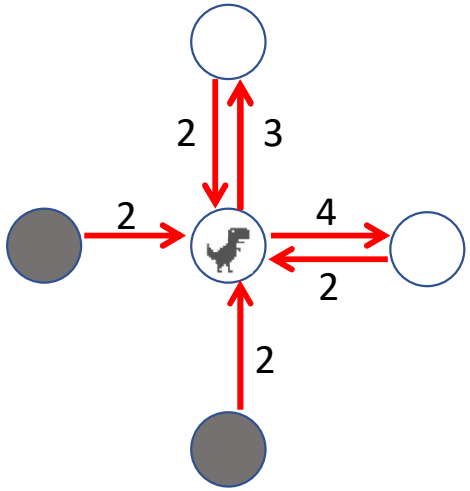
자신의 철칙을 지키되, 아픈 건 싫어하는 호석사우루스를 도와서 탈출구까지의 최소 충격량을 구해주자!

주어진 문제를 “**그래프 문제**”로 변환시켜서

“**최단 거리**” 계산으로 방향을 잡아보자.

I 생각의 흐름 – 2. 그래프 설정





I 생각의 흐름 – 3. 최단 거리 알고리즘

최단 거리 알고리즘 복기

<입력>

그래프 & 시작점 & 도착점

<출력>

최단 거리

배운 것 중에는 BFS & Dijkstra 존재

그 중 간선의 가중치가 다를 수 있다는 점을 통해 **Dijkstra** 선택

I 생각의 흐름 – 4. Dijkstra 알고리즘 거리 정의

Dijkstra 알고리즘은 `distance` 배열이 필요하다.

`dist[i][j] := i 행 j 열까지의 최소 충격량(최단 거리)`

으로 정의하면 되는가?

➔ 같은 위치에 대해서도 몇 번째 도착인지가 중요하기 때문에 정보에 손실이 생긴다!

I 생각의 흐름 – 4. Dijkstra 알고리즘 거리 정의

Dijkstra 알고리즘은 `distance` 배열이 필요하다.

`dist[i][j][move]` := `i` 행 `j` 열에 `(3K+move)` 번의 이동으로 도착하는 방법 중에서 최소 충격량(최단 거리)

I 시간, 공간 복잡도 계산하기

정점의 개수는 $N * M * 3$ 개라고 생각하면 된다.

즉 총 시간 복잡도는 $O(NMlogNM)$이다.

I 구현

```
static int dir[][][] = {
    {{1, 0}, {-1, 0}},  // 3K + 1 번째 이동
    {{0, 1}, {0,  -1}},  // 3K + 2 번째 이동
    {{1, 0}, {0,  1}, {-1, 0}, {0, -1}}  // 3K 번째 이동
};

static void pro() {
    /* TODO */
    out.println(ans);
    out.close();
}
```