○ 제출기한: 2023년 3월 29일 수요일 3pm

○ 제출방법: 각 문제에 대한 답을 손글씨로 작성하여 교탁 위에 제출

---

1. 오른편의 프로그램 코드를 보고 물음에 답하시오
   가. 이 코드의 fault는?
   나. 해당 fault를 실행하지 않는 test case는?
   다. 해당 fault를 실행하지만 error를 발생시키지 않는 test case는?
   라. error를 발생시키지만 failure를 일으키지는 않는 test case가 있는가? 있다면 찾으시오.

```
public static int oddOrPos(int[] x){
// effects: if x==null throw NullPointerException
// else return the number of elements in x that
// are either odd or positive (or both)
   int count = 0;
   for (int i=0; i < x.length; i++){
      if (x[i]%2 == 1 || x[i] > 0)  count++;
   }
   return count;
}
```

2. Binary Search 프로그램을 작성하기 전에 blackbox test case를 먼저 작성하고자 한다. 다음의 기법을 사용하여 test case 들을 디자인하라. (입력되는 정수형 배열의 각 원소값은 0~1000 이고, 컴퓨터 메모리가 감당할 수 있는 입력양의 최대 크기 n 은 임의로 설정할 수 있다고 가정한다)
   가. Strong robust equivalence class
   나. Classification Tree Method

3. 오른편의 프로그램 코드는 LCM(Least Common Multiple)을 찾는 프로그램이다. 다음 물음에 답하시오
   가. C/DC coverage 를 100% 만족하는 test case를 디자인 하시오
   나. MC/DC coverage를 100% 만족하는 test case를 디자인 하시오

LCM using while and if

```c
#include <stdio.h>

int main() {

    int n1, n2, max;

    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);

    // maximum number between n1 and n2 is stored in max
    max = (n1 > n2) ? n1 : n2;

    while (1) {
        if ((max % n1 == 0) && (max % n2 == 0)) {
            printf("The LCM of %d and %d is %d.", n1, n2, max);
            break;
        }
        ++max;
    }
    return 0;
}
```

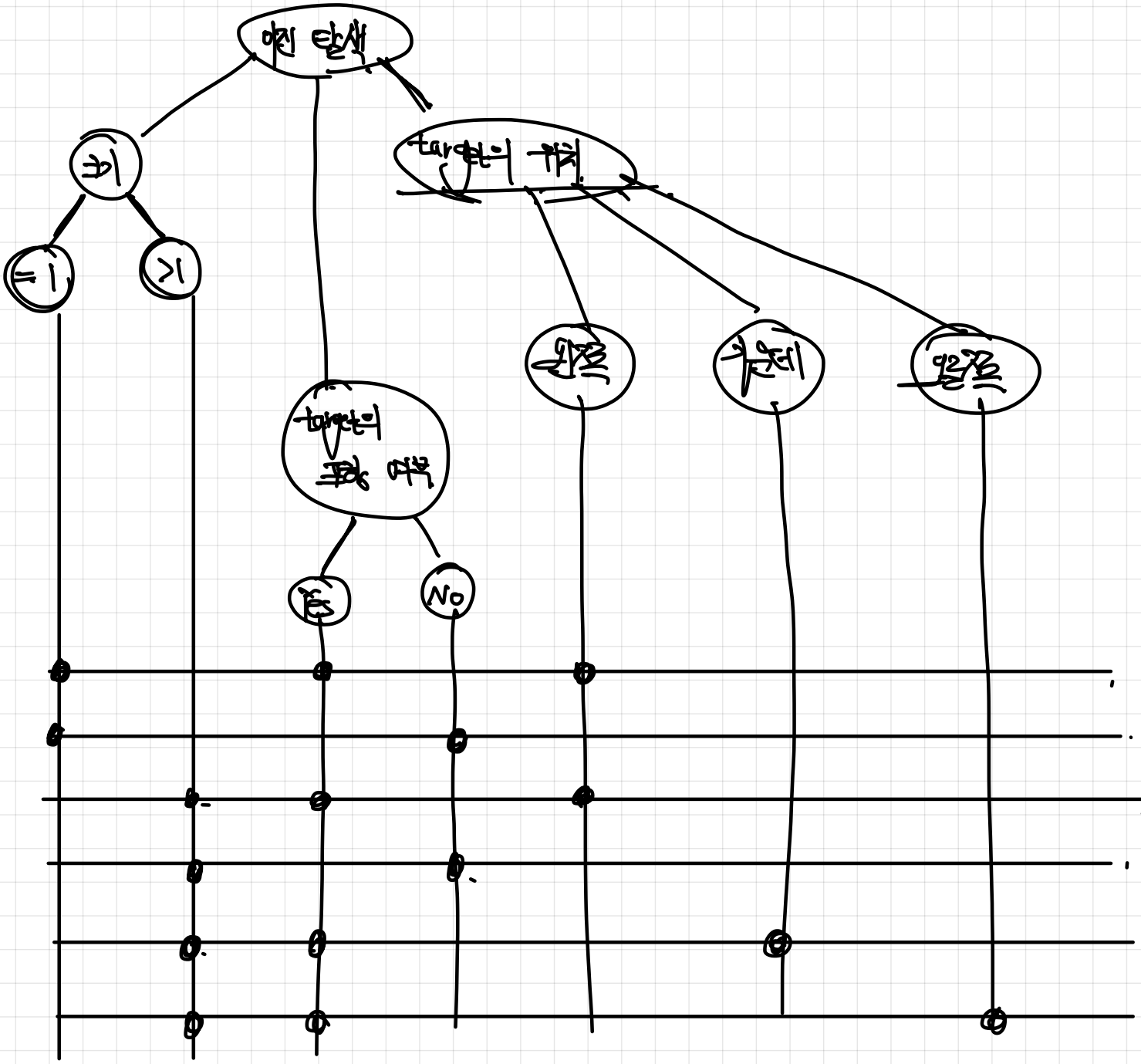
4. 다음 페이지의 프로그램 코드를 보고 물음에 답하시오
   가. pat 함수의 control flow graph를 작성하시오
   나. EPC를 100% 만족하는 (최소한의 수의)test path 들을 찾으시오
   다. Defs and Uses table 을 작성하시오
   라. isPat 변수에 대한 unique 한 DU path들을 모두 찾으시오
   마. 라번에서 찾아낸 DU path 들을 모두 cover할 수 있는 최소한의 수의 test path 와 해당 path를 수행할 수 있는 입력값들을 찾으시오

Binary search → Black box testing 한번 해보기

n. 배열크기는 생각 X.

CTM

Req 받는 부분 → 형식화의 특징

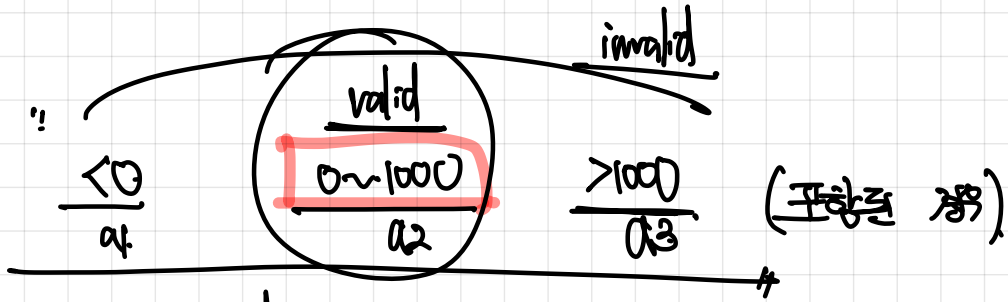


	list	target	output
TC1	[0]	0	0
TC2	[0]	1	-1
TC3	[0 1 2]	0	0
TC4	[0 1 2]	3	-1
TC5	[0 1 2]	1	1
TC6	[0 1 2]	2	2

list : 정렬된 리스트
target : 찾으려는 값.
⇒ list 내에 target이 있다면 해당 index. 없으면 -1을 반환.

---

strong robust equivalence class → [valid / invalid]



#정렬 input: (포함관계 정렬되어 있음).

#정렬

list: valid → 배열의 모든 값이 0~1000 의 정수 a_1

invalid → 배열의 값중 하나라도 < 0 a_2

target: valid → 배열의 값중 하나라도 >1000 a_3

invalid → 배열의 값중 하나라도 (중복값) a_4

배열의 길이 b_1

배열의 가운데 b_2

배열의 끝값 b_3

배열의 첫값 바로 뒤 b_4

배열의 끝값 바로 앞 b_5

target → list 에 있는 값

list : 정렬 된 배열

target : 찾는 값

output : 탐색 범위 내에 찾는 값이 있으면 해당 index

없으면 -1

즉 조건 list → invalid

한 경우 해당하면 -1

	list	target	output
a_1	b_1	[0 1 2] 0	0
a_1	b_2	[0 1 2] 1	1
a_1	b_3	[0 1 2] 2	2
a_1	b_4	[1 2 3] 0	-1
a_1	b_5	[1 2 3] 4	-1

a_2	b_1	[-1 0 2] -1	→ -1
a_2	b_2	[-1 0 2] 0	→ -1
a_2	b_3	[-1 0 2] 2	→ -1
a_2	b_4	[-1 0 2] -2	→ -1
a_2	b_5	[-1 0 2] 3	→ -1

2-(2) CTM.

(2) CTM → <입력>

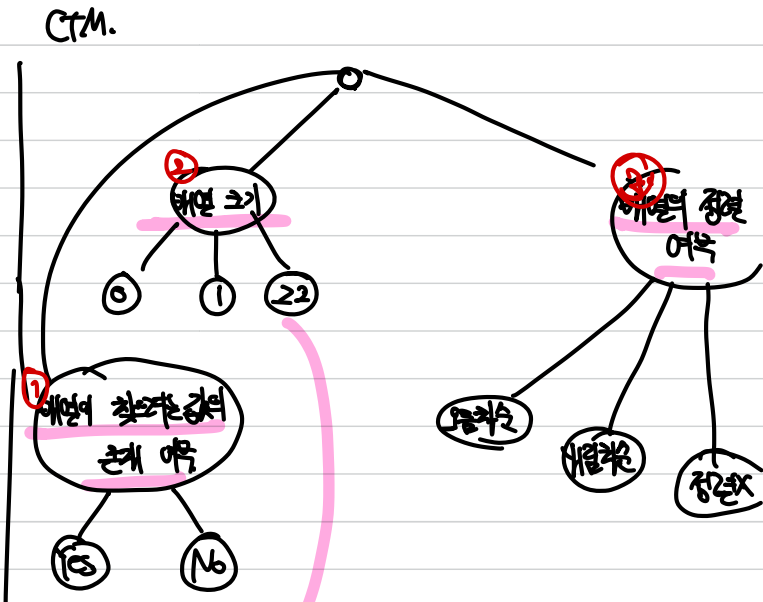
명세 정의
- arr : 정렬된 정수 배열
- n : 배열의 크기
- x : 찾으려는 값

<출력>

idx : 찾으려는 x의 인덱스. 배열에 없으면 -1을 반환.

그리고 배열의 크기가 0이면 -1을 반환.

그리고 배열이 정렬되어 있지 않으면 -1을 반환.



① arr[] n=0 x=1

No, 0, - / 1

② arr.[1] n=1 x=1

Yes, 1, - / 0

③ arr [1] n=1 x=2

No 1, - / -1

④ arr [1, 2, 3, 4, 2] n=5 x=1

Yes, ≥2, 정렬 / 0

arr [5 4 3 2 1] n=5 x=1

Yes, ≥2, 비정렬 / 1

arr [4 5 2 3 1] n=5 x=1

Yes, ≥2, 정렬x / -1

arr [4 5 2 3 1] n=5 x=6

No, ≥2, 정렬x -1

그리고 ≥2 인 경우. [홀수, 짝수]로 고려해

찾으려는 값이 [짝수, 홀수]에 있다고 해서 죽은 듯.

†

중복 O or X [0, X]도 추가할 수 있고.

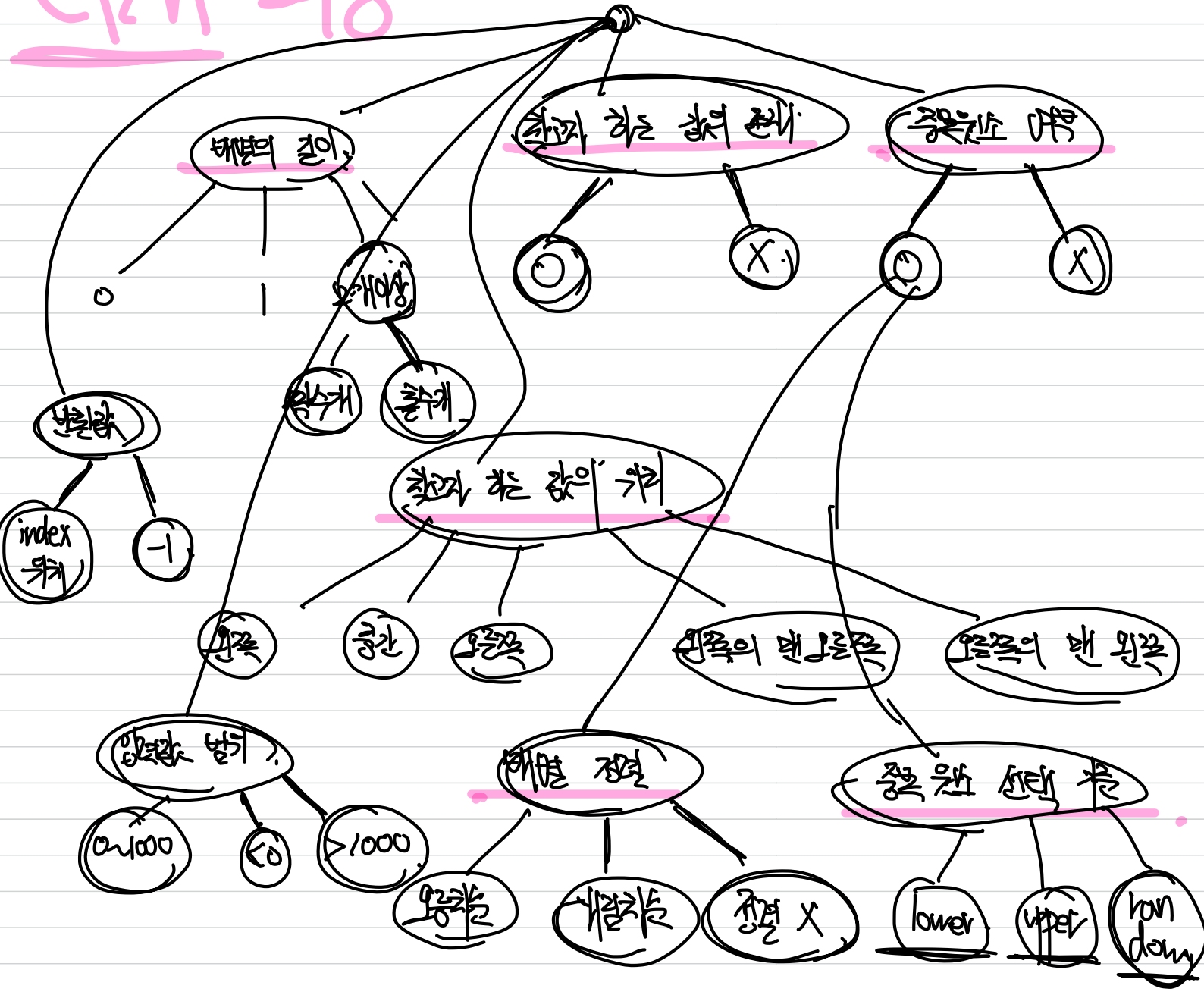
존재한다면 lower_bound, upper_bound를

반환할 것인지도 고려해야 할 것임.

범위에서 원소값의 범위가 0~1000 인지 아닌지로 고려

해줘야 할 것이다.

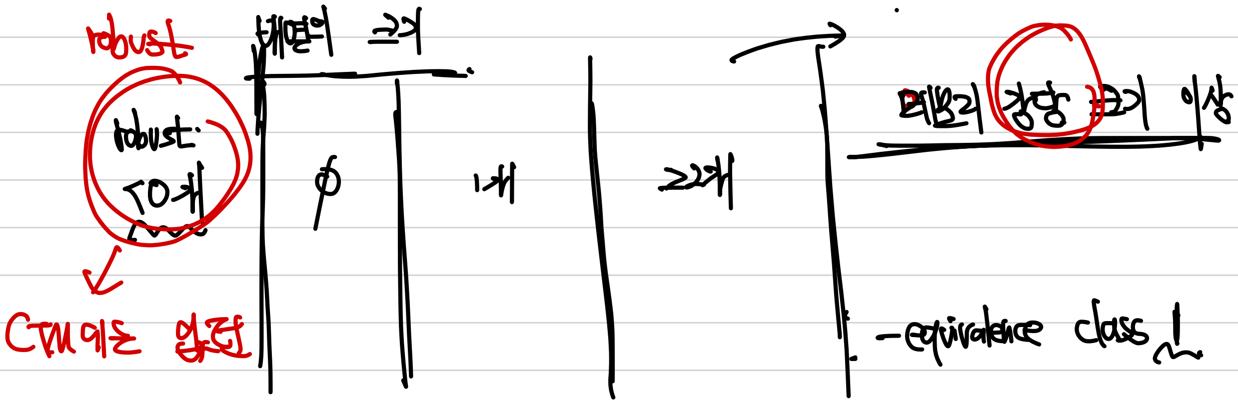
CTM 최종



2-(1) SREC

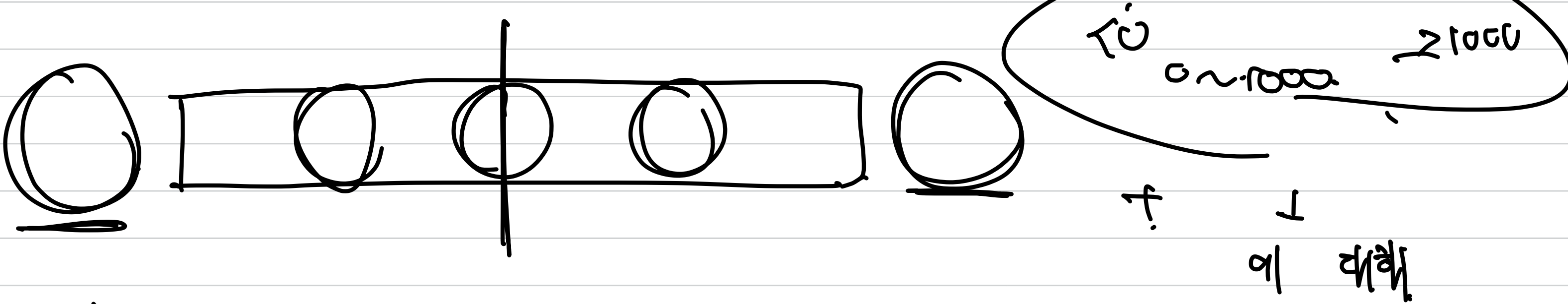
(1) strong robust equivalence class → 배열의 입력값 (0~1000)

모든 동치 관계의 쌍 (다 고려), → CTM에서 비정상적인 값도 포함해보자.

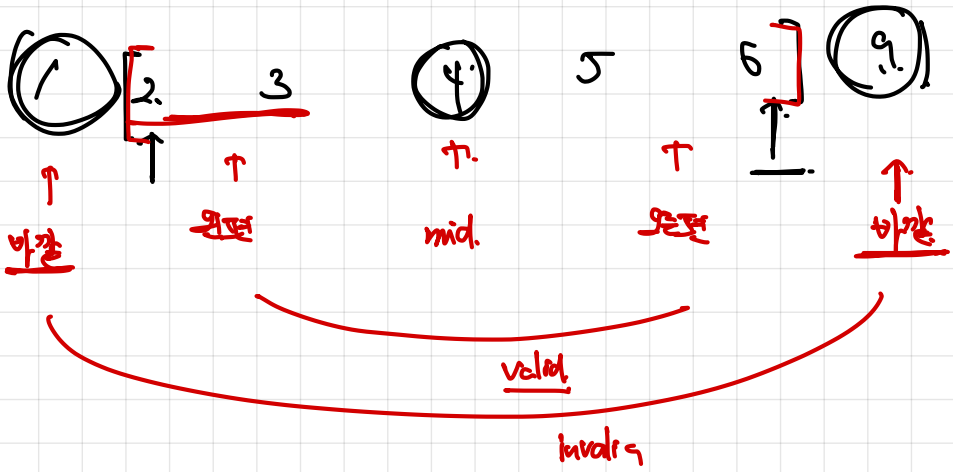


CTM 에서 invalid한 값은 포함시킬 수 있는 건 다 넣었기 때문
(분류기준이)

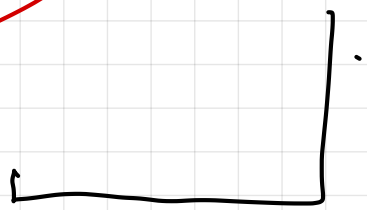
위 결을 추가 후 CTM의 분류 기준과 합쳐 모든 경우의 수를 고려해보면 된다.



robust는 입력값의 특성을 생각하지 않고,
valid invalid만 생각



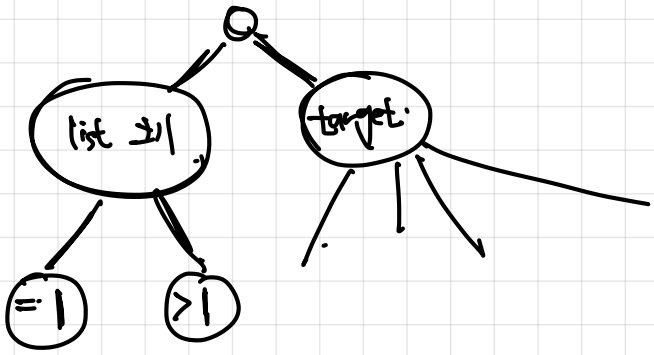
(<0 -> 1006 invalid.
0~1000 valid.



strong - robust equivalence class

-1

GM



#3. C/DC coverage.

$n_1 < n_2$

minimum은 X

ⓐ "max % n1 == 0" (최소 2)

ⓑ "max % n2 == 0" (최소 5)

C/DC coverage 100%

(가) $(n_1 < n_2)(\max \% n1 == 0 \text{ && } \max \% n2 == 0)$

F (T T) ∴ T → ($n_1 = 5$, $n_2 = 2$, max = 10) (LCM 최소 값)

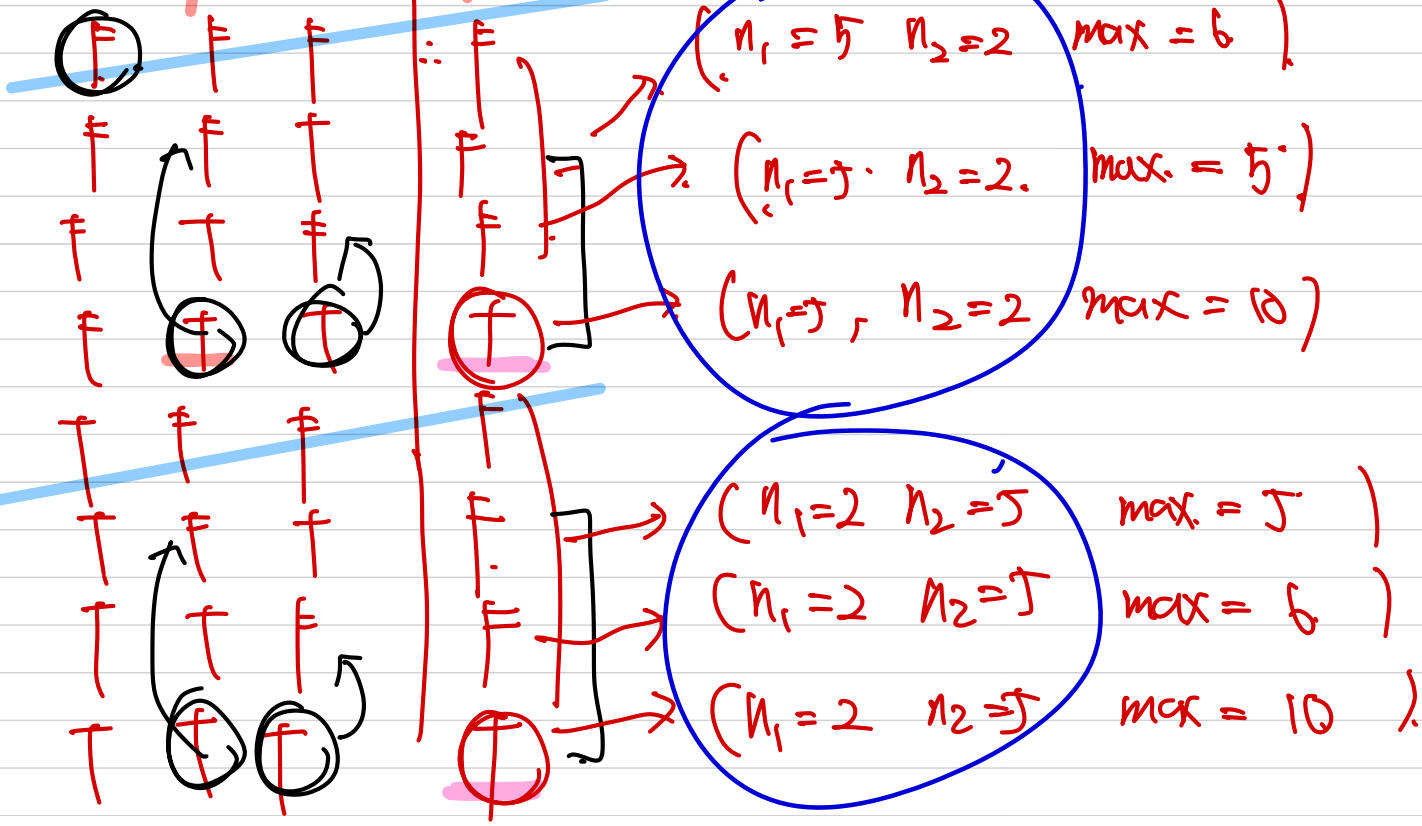
F (T F) ∴ F → ($n_1 = 5$, $n_2 = 2$, max = 5)

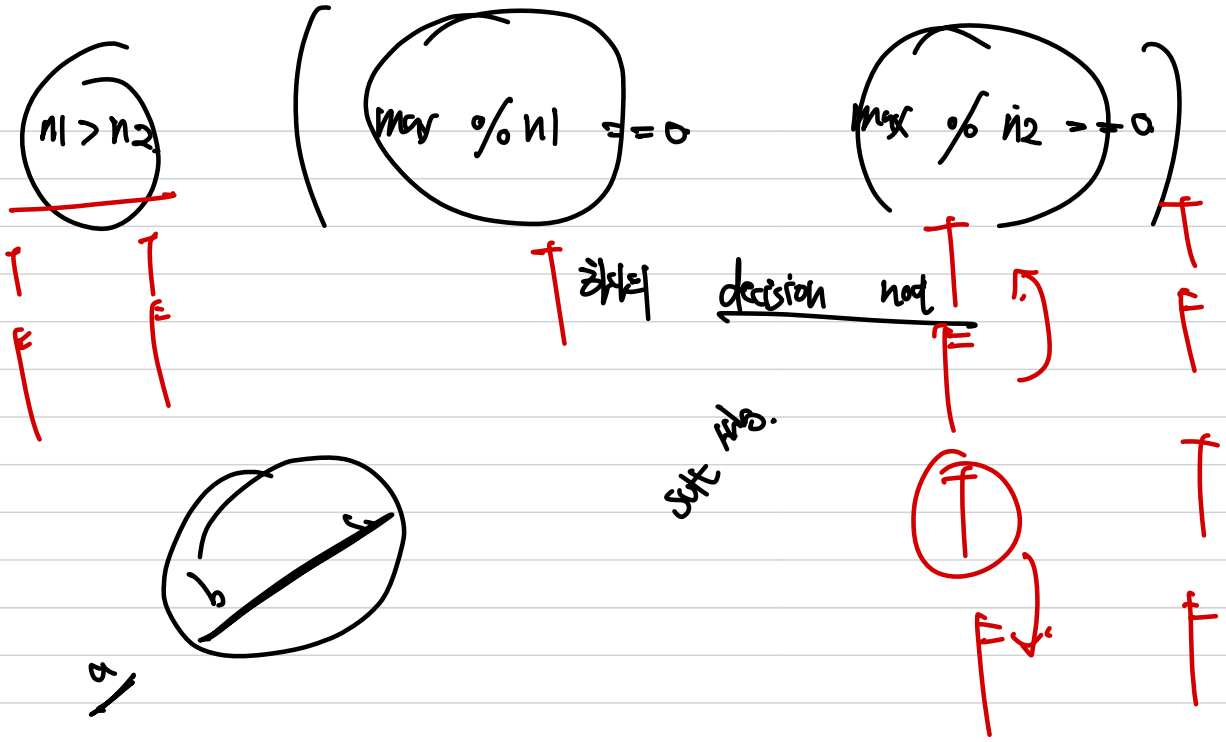
F (F T) ∴ F → ($n_1 = 5$, $n_2 = 2$, max = 6)

T (F F) ∴ F → ($n_1 = 2$, $n_2 = 5$, max = 10)

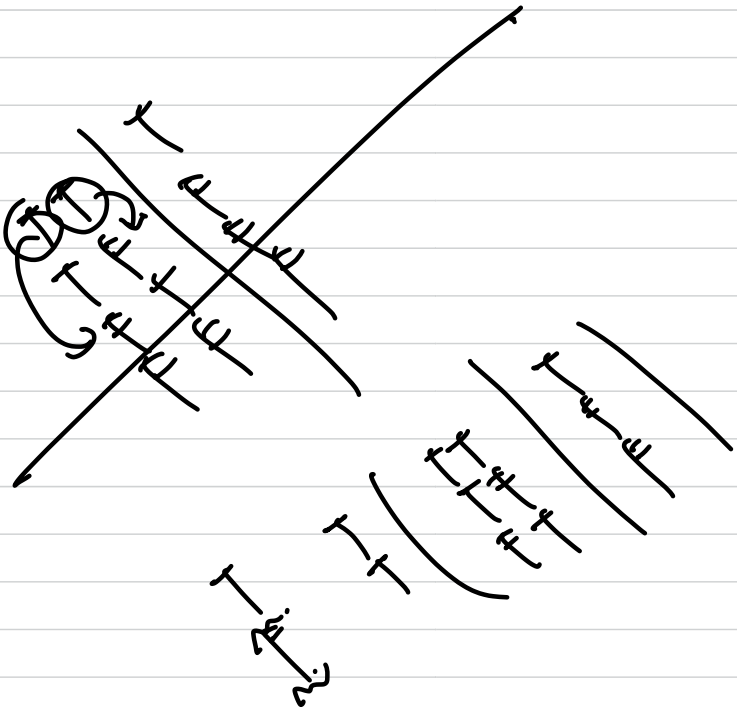
최소 $n_1 = 5$, $n_2 = 2$

(나) MC/DC coverage 100%




n1 > n2
mod % n1 == 0
mod % n2 == 0
leaf
decision node
T
F

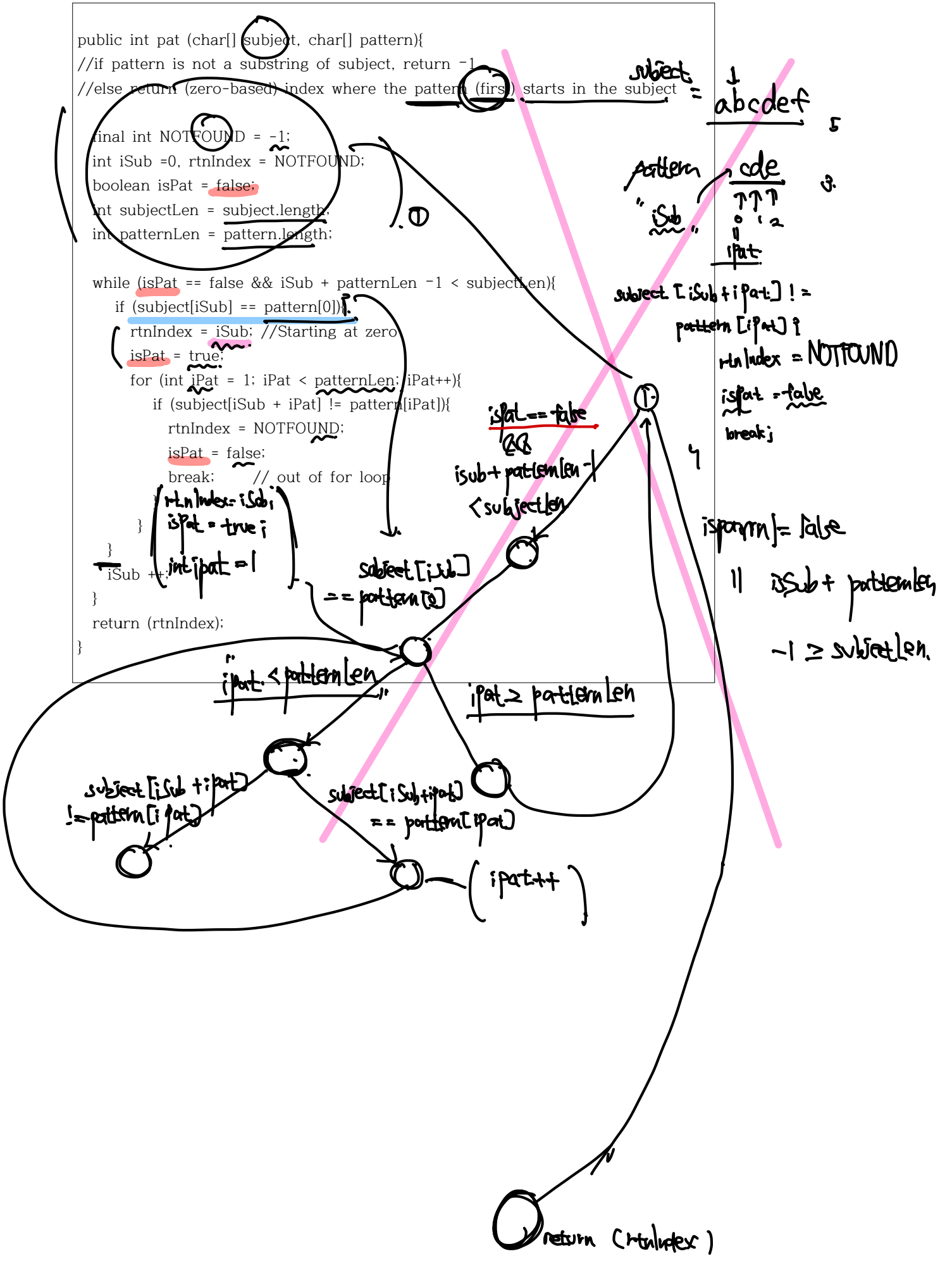
b
a



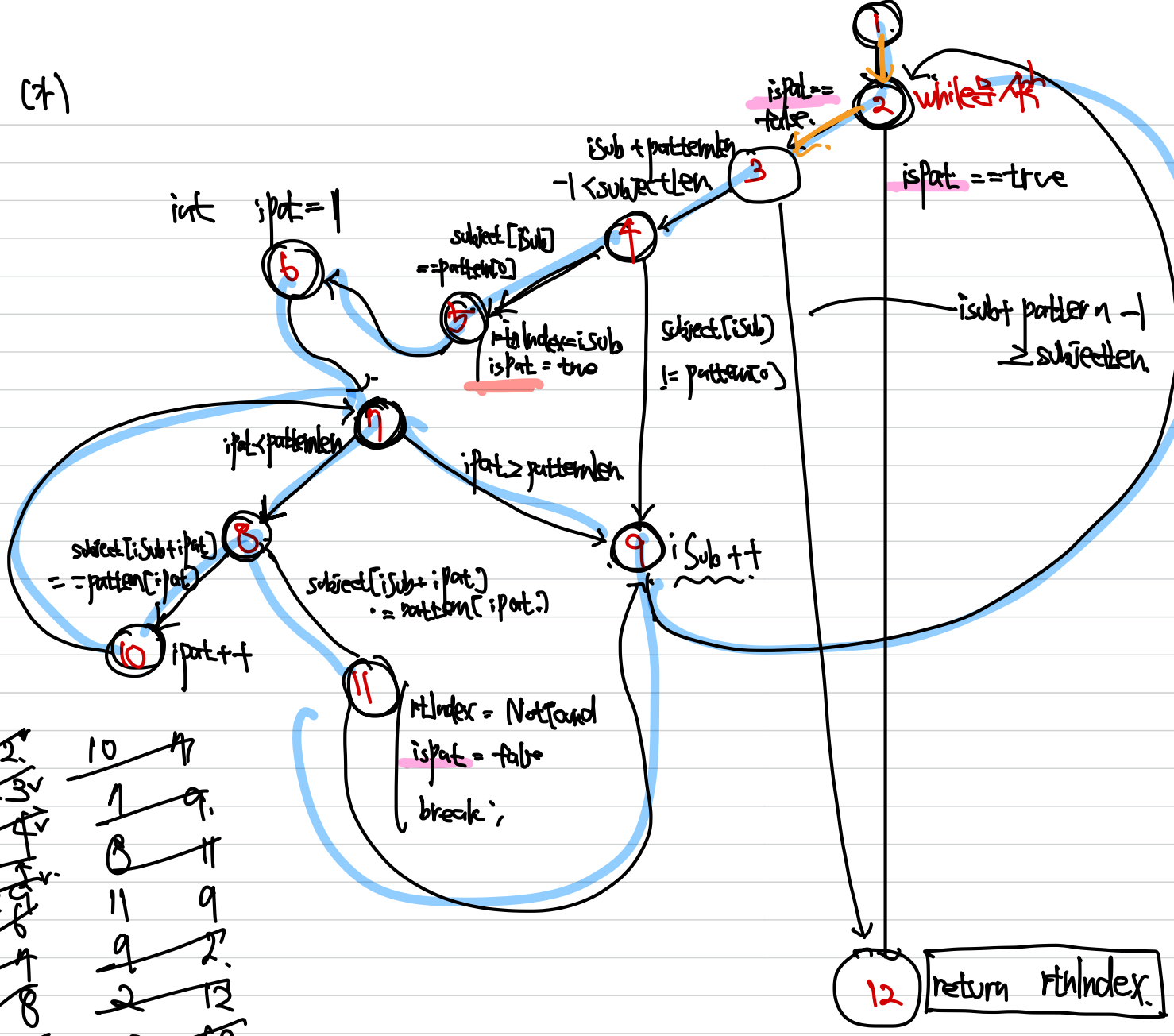
```
public int pat (char[] subject, char[] pattern){
//if pattern is not a substring of subject, return -1
//else return (zero-based) index where the pattern (first) starts in the subject

  final int NOTFOUND = -1;
  int iSub =0, rtnIndex = NOTFOUND;
  boolean isPat = false;
  int subjectLen = subject.length;
  int patternLen = pattern.length;

  while (isPat == false && iSub + patternLen -1 < subjectLen){
     if (subject[iSub] == pattern[0]){
       rtnIndex = iSub; //Starting at zero
       isPat = true;
       for (int iPat = 1; iPat < patternLen; iPat++){
          if (subject[iSub + iPat] != pattern[iPat]){
            rtnIndex = NOTFOUND;
            isPat = false;
            break;      // out of for loop
          }
       }
     }
     iSub ++;
  }
  return (rtnIndex);
}
```



(가)



1	10
2	7
3	8
4	11
5	9
6	2
7	12
8	3
9	12
10	7
11	9

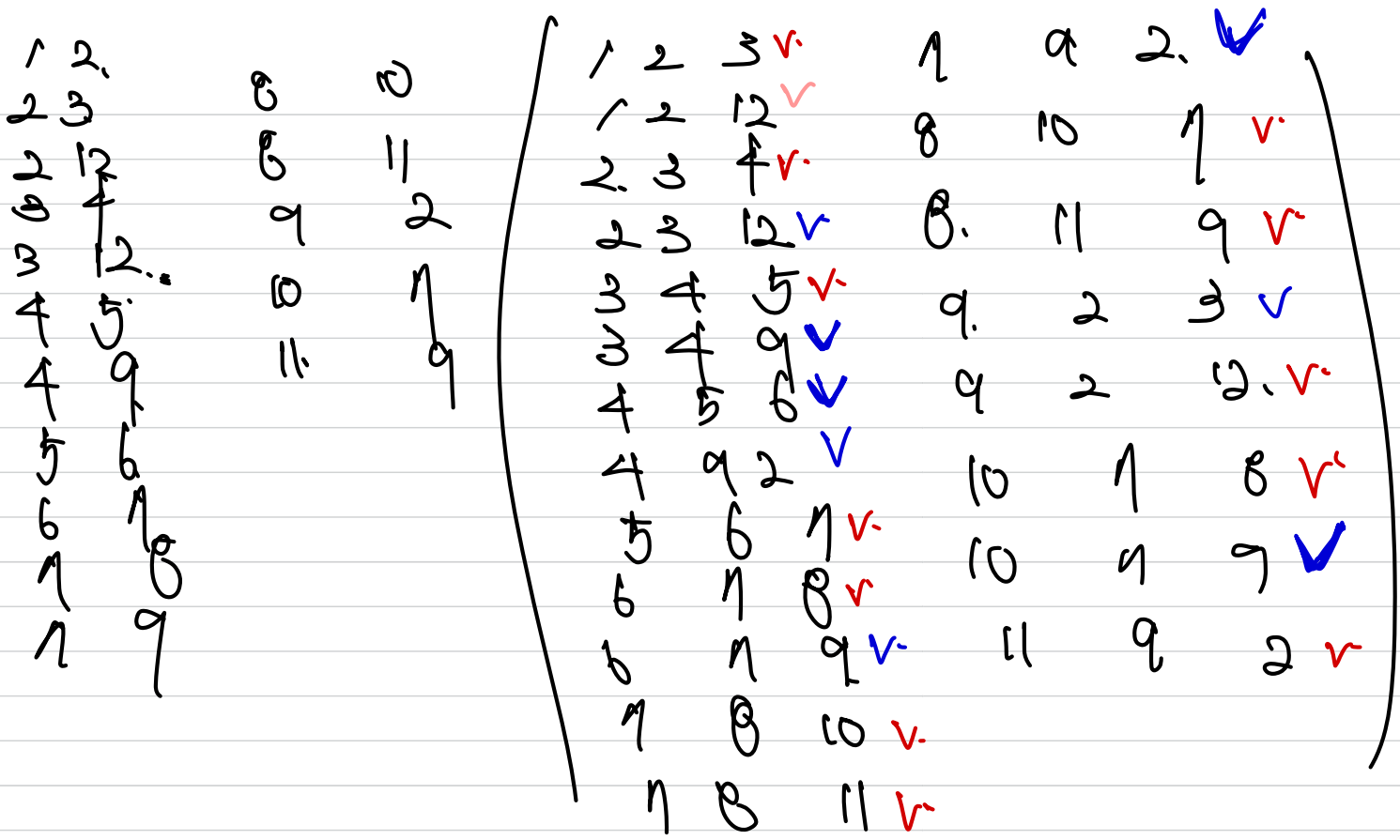
(나) Edge 100% 커버하는 test path를 찾아라

[1 2 3 4 5 6 7 8 10 7 8 10 7 8 10 11 9 2 12]

[1 2 3 4 5 6 7 8 10 7 9 2 3 4 9 2 3 12]

[1, 2, 12]

1	2
2	12
3	3
3	4
2	3
12	
3	4
5	
4	5
6	
5	6
7	
6	7
8	
7	8
10	
7	8
11	
8	10
7	
10	7
7	9
10	7
9	
7	9
2	
8	11
9	
11	9
9	2
3	4
9	
6	4
9	



또 EPC.

i)

[1 2 3 4 5 6 7 8 10 7 8 10 7 8 10 7 8 10 7 8 11 9 2 12]

ii)

[1 2 3 4 5 6 7 8 10 7 9 2 3 4 9 2 3 12]

iii)

[1 2 12]

↑

5 6 7.

끝

pre - def {subject, pattern}.

(1) Defs and Uses table.

	def	uses
1.	{.Notfound, isSub rtnIndex ispat subjectLen patternLen}. (1,2)	{.Notfound, subject, pattern}.

ispat = false

Edge가 def로 올 수 X.

2.
(2,3) → {isPat}.
(2,12) → {isPat}.

3.
(3,4) → {.isSub, patternLen, subjectLen}. → {.isSub, patternLen, subjectLen}.
(3,12) →

4.
(4,5) → {.subject, iSub, pattern}.
(4,9) → {subject, iSub, pattern}.

5. {.rtIndex, ispat}. {.iSub.}.

ispat = true

(5,6)

6. {ipat}.

(6,7)

7.
(7,8) → {ipat, patternLen}
(7,9) → {ipat, patternLen}

⑥

(6, 10) → {subject, iSub, ipat: pattern}

(6, 11) → {subject, iSub, ipat: pattern, iSub}

⑨ {iSub}

(9, 2)

⑩ {ipat2} {ipat}

(10, 11)

⑪ {retIndex, isPat: isPat = false} {-1, NotFound}

(11, 9)

(2) ipat 의 def 에 대한 unique DU path 찾기

① DU-pair def

[11, 9, 2, 3]
(11, (2, 3))
(11, (2, 12))

[11, 9, 2, 12] infeasible

② [5, 6, 11, 9, 2, 3]
(5, (2, 3))
(5, (2, 12))

[5, 6, 11, 6, 10, 11, 9, 2, 12] infeasible

③ [1, 2, 3]
(1, (2, 3))
(1, (2, 12))

[1, 2, 12] infeasible

① ~ ⑥ DU-path (but unique..?)

feasible

'a', 'a' 비교 대상. 'a' 'a' ≠ 'a' 'c'

[1 2 3 4 5 6 7 8 11 9 2 3
4 5 6 7 8 10 7 9 2 12]

매칭 완료 'a' 'c'

subTest: ['a', 'a', 'c']

pattern: ['a', 'c']