2019 Winter Data Structure
3차 Test

A. 올해는 2020년이다!	(구현)
B. 오엑스를 맞춰보자 1	(이론)
C. 오엑스를 맞춰보자 2	(이론)
D. 에디터	(문제 해결)
E. 수식 트리	(구현)
F. 오엑스를 맞춰보자 3	(이론)
G. 유기체	(문제 해결)
H. 회전하는 큐	(문제 해결)
I. 객관식을 맞춰보자 1	(이론)
J. 요세푸스 문제	(문제 해결)
K. 트리인가?	(문제 해결)
L. 트리의 순회	(구현)
M. 경로 찾기	(문제 해결)
N. DFS와 BFS	(구현)
O. 스택 수열	(문제 해결)
P. 연결 리스트	(구현)
Q. 하노이의 탑	(문제 해결)
R. 가장 가까운 공통 조상	(문제 해결)

2020.01.07. 13:10 ~ 16:40 (3시간 30분)

주의 사항 “0. #include <algorithm>의 사용을 지양하시길 권장합니다.”

1. 여러분은 C++ 언어로 문제를 풀 수 있습니다. C언어로 문제를 푸시더라도 cpp 파일로 제출하셔야 합니다. 또한 C++ Reference Site를 이용하여 문제를 푸실 수 있습니다. 25장 이내의 단면 A4용지에 인쇄되거나 자필로 작성하신 치팅 페이퍼 역시 허용됩니다. 단, 책을 참고하거나 인터넷 검색을 하는 행위는 불가합니다.

* C++ reference site : https://en.cppreference.com/w/

2. 총 문제는 18문제이며, 한 문제당 100점입니다. 모든 문제에는 문제 유형이 표기되어 있으며, 부분 점수가 존재합니다. 이번 Test까지에 한해서, 빨리 풀거나 늦게 푸는 것은 점수에 전혀 영향을 미치지 않습니다. (최종 Test에서는 영향을 미칩니다.) 틀린 횟수 역시 점수에 영향을 미치지 않으며, 여러 번 채점을 시도할 경우 제일 높은 점수로 인정됩니다.

*** 문제 배치와 난이도는 아무런 관계가 없습니다!
즉 앞에 있는 문제가 어려울 수도, 뒤에 있는 문제가 쉬울 수도 있으니
모든 문제를 한번 읽어보시고 잘 결정하시기 바랍니다 ***

3. 문제 유형의 경우 세 가지가 존재합니다. 이론, 구현, 문제 해결이 이에 해당합니다. 일부 구현 문제의 경우 코드를 제시하여 줍니다. 이는 문제지와 답안 입력 창에 예시 코드가 적혀 있으니 복사하시어 사용하시면 됩니다. 그 예시 코드를 이용해서, 특정 함수 부분에 코드를 추가하여 완성하여야 합니다. 이 때 다른 헤더 파일을 include하거나 다른 부분을 고치는 행위는 금지합니다. 문제 해결 유형의 경우 프로그램을 작성하는 데에는 제한이 없습니다.

** 구현 문제에서 제시된 코드들은 문제 해결 문제에서 모두 사용 가능 **

4. 이번 시험에서 800점 미만의 점수를 획득할 경우, 스터디에서 탈락됨을 알려드립니다. (총점 1800점)

5. exit(0)을 제외한 System Call 함수의 사용을 금지합니다. 또한, 코드 제출 전 무한루프 유무, 출력 포맷이 잘못되지 않았는지, 과도한 출력을 하지 않는지를 점검하세요. 서버에 손상이 가서 과금 유도가 될 경우 일부 책임을 물을 수 있음을 알려드립니다. 제출에 신중하세요!

6. 1회 제출 당 10초의 Delay를 부여합니다. 제출 시 더블 클릭을 삼가 주세요.

Problem A

올해는 2020년이다!

제한 시간 : 1초, 메모리 제한 : 128MB, 문제 유형 : 구현 문제

문제

드디어 2019년이 가고 2020년이 왔다! 윤원이는 2020년을 맞아서 신이 난 나머지, 유세인트에서 지금 학기까지 받은 점수들을 실수로 전체 공개해 버렸다! 10분 뒤 실수한 것을 안 윤원이는 그 점수들을 비공개로 돌려버렸지만, 로그 파일은 남아 있는 법!

여러분은 지금부터, 이진 탐색 알고리즘(Binary Search)을 사용해서 윤원이가 해당 점수를 받은 적이 있는지를 알아보고자 한다. 친절하게도 인터넷에는 아래와 같은 예시 코드가 있었으나, 우리의 입맛에 맞게 수정해야 한다. 아래 예시 코드를 보고, 코드에서 비어 있는 부분을 추가하여 완성해 보자.

조건

1. 추가적으로 새로운 헤더 파일을 include할 수 없음.
2. STL은 string 관련 함수, vector 객체와 관련 함수에 한하여 사용 가능
3. 아래 예시 코드에서, 빨간색으로 칠해진 부분에 한해서 수정 가능함. 단 함수의 원형은 건드릴 수 없음.
4. 이진 탐색(Binary Search)이 아닌 순차 탐색(Linear Search)을 사용하는 경우, 시간 초과가 나는 것을 책임지지 않음. 빈칸에 코드를 제대로 삽입만 한다면 문제는 없음.

Input

첫 번째 줄에 윤원이가 수강한 과목의 개수 N이 제시되며, N은 1부터 100 사이의 자연수이다.
두 번째 줄에 윤원이가 받은 성적 N개가 제시되며, 각각의 성적은 공백으로 구분된다. 각각의 성적은 0 ~ 100 사이의 정수이며 오름차순으로 정렬되어 제시된다. 중복되는 값은 들어오지 않는다.
세 번째 줄에 윤원이가 이 성적을 받았는지를 물어보는 성적 T가 제시된다. T 역시 0 ~ 100 사이의 정수이다.

Output

만약 윤원이가 T 성적을 받은 적이 있다면 True를, T 성적을 받은 적이 없다면 False를 출력한다.

예제 1

Sample Input	Sample Output
6 35 38 40 41 42 43 35	True

예제 2

Sample Input	Sample Output
4 28 32 41 42 35	False

예시 코드 - 빨간색 부분의 코드를 완성시킬 것

< Sort_And_Recursive_BinarySearch.cpp >

```
#include <iostream>
#include <string>
#include <cstring>
#include <vector>
using namespace std;

int arr[100000];   // 윤원이가 받은 성적을 저장하기 위한 배열

// 이진 탐색을 수행하여 그 결과를 알려주는 함수
// target이 ar[]에 존재하면 그 타겟의 위치를 반환하고, 존재하지 않으면 -1을 반환한다.
// 다음 함수에서 빈칸에 해당하는 부분을 구현하기
int BSearchRecur(int ar[], int first, int last, int target) {
	int mid;
	if (                                )		// first와 last는 index로서 서로를 비교해야 한다.
		return -1;				// -1의 반환은 탐색의 실패를 의미
	mid =                                   ;	// 탐색대상의 중앙을 찾는다.

	if (ar[mid] == target)
		return mid;                                     // 검색된 타겟의 인덱스 값 반환
	else if (target < ar[mid])
		return BSearchRecur(                    );	// BsearchRecur 함수를 재귀호출한다.
	else
		return BSearchRecur(                    );	// BsearchRecur 함수를 재귀호출한다.
}

int main(void)
{
	ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);    // cin과 cout의 속도를 개선하기 위함
	int N; // 윤원이가 수강한 과목의 개수
	cin >> N;

	// 윤원이가 받은 성적을 저장함
	for (int i = 0; i < N; i++) {
		cin >> arr[i];
	}

	int T;    // 윤원이가 이 성적을 받았는지를 알아보기 위해, 타겟 점수를 저장
	cin >> T;

	// 타겟 점수에 대해, 윤원이가 받은 성적을 저장한 배열에서 이진 탐색을 수행
	int idx;
	idx = BSearchRecur(arr, 0, N - 1, T);
	if (idx == -1)
		printf("False");
	else
		printf("True");

	return 0;
}
```

Problem B

오엑스를 맞춰보자 1

제한 시간 : 1초, 메모리 제한 : 128MB, 문제유형 : 이론

여러분은 두 달간 자료구조 이론에 대해 공부했었다. 이제 다음 오엑스 문제들에 대해 답할 시간이 되었다!

우리가 만들 프로그램은, 문제 번호가 입력으로 제시될 때,
그 문제에 해당하는 답을 출력하여 주는 프로그램을 완성해 보는 것이다.
이 때, 답이 O라면 1을, 답이 X라면 2를 출력하기로 하자.

현재 미완성된 프로그램은 다음과 같다. (6번 줄에서, 배열의 빈칸에 정답을 채워 넣은 소스 코드를 제출하면 된다.)

```
1  #include <iostream>
2  #define PROBLEM_NUM_MAX 10 // 최대 문제 수
3  using namespace std;
4
5  // 1번 문제의 정답은 answer[0]에, 2번 문제의 정답은 answer[1]에, ... 기입한다.
6  int answer[PROBLEM_NUM_MAX] = { ?, ?, ?, ?, ?, ?, ?, ?, ? };
7
8  int main(void) {
9      int problem_number;
10     cin >> problem_number;
11     for (int i = 1; i <= PROBLEM_NUM_MAX; i++) {
12         if (problem_number == i) {
13             cout << answer[i - 1] << endl;
14             break;
15         }
16     }
17     return 0;
18 }
19
```

아래 10문제를 보고, O, X 여부를 판단하여 위 소스 코드를 완성하여라. 예를 들어, 1번 문제의 답이 O라면, 입력값이 1일 때 출력값은 1이다. 만약 3번 문제의 답이 X라면, 입력값이 3일 때 출력값은 2이다.

1번 문제. 큐는 먼저 들어간 것이 나중에 나오는 구조이다. ( )
2번 문제. 연결 리스트, 스택, 큐는 선형적인 자료구조이고, 트리와 그래프는 비선형적인 자료구조이다. ( )
3번 문제. 트리는 계층적인 자료구조라고도 하며 데이터베이스(DB)에서 널리 쓰인다. ( )
4번 문제. 이진 트리가 되려면, 루트 노드를 중심으로 둘로 나뉘는 두 개의 서브 트리도 이진 트리여야 하고,그 서브 트리의 모든 서브 트리도 이진 트리이어야 한다. ( )
5번 문제. 모든 레벨이 꽉 찬 이진 트리를 포화 이진 트리(Full binary tree)라고 한다. ( )
6번 문제. 다음 수식 7+4*2-1은 중위 표기법을 이용한 것이다. ( )
7번 문제. 다음 수식 12+7*은 후위 표기법을 이용한 것이다. ( )
8번 문제. 12+7- 과 1+2-7은 실질적으로는 동일한 수식이다. ( )
9번 문제. 수식 트리의 구성과정에서 피연산자는 무조건 스택으로 옮긴다. ( )
10번 문제. C++ STL에서, set의 경우 중복된 원소는 한 번만 넣을 수 있으며 여러 번 insert(삽입 연산)를 한다고 해도 한 번만 들어가게 된다. ( )

Input

문제 번호가 입력으로 제시된다. 1번 문제의 경우 1, 2번 문제의 경우 2가 제시된다. 문제 번호는 1부터 5 사이의 정수이다.

Output

각 문제에 대한 답을 하나의 정수로 출력한다. 예를 들어, 1번 문제의 답이 3이라면 3을 출력한다.
아래 예시는 1번 문제의 답이 3일 때와, 2번 문제의 답이 5일 때를 나타낸 것이다.
예제는 채점하지 않으며 자신이 맞다고 생각하는 답을 출력하면 된다.

예제 1

Sample Input	Sample Output
1	3

예제 2

Sample Input	Sample Output
2	5

Problem C

오엑스를 맞춰보자 2

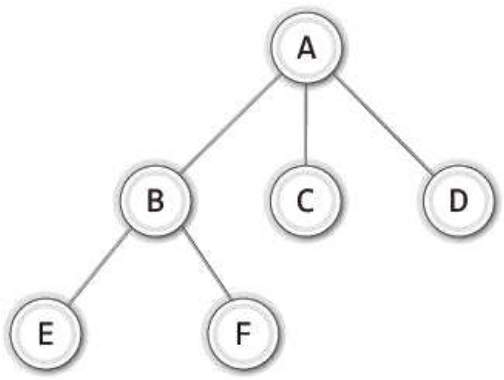
제한 시간 : 1초, 메모리 제한 : 128MB, 문제유형 : 이론

여러분은 두 달간 자료구조 이론에 대해 공부했었다. 이제 다음 오엑스 문제들에 대해 답할 시간이 되었다!
우리가 만들 프로그램은, 문제 번호가 입력으로 제시될 때,
그 문제에 해당하는 답을 출력하여 주는 프로그램을 완성해 보는 것이다.
이 때, 답이 O라면 1을, 답이 X라면 2를 출력하기로 하자.
현재 미완성된 프로그램은 다음과 같다. (6번 줄에서, 배열의 빈칸에 정답을 채워 넣은 소스 코드를 제출하면 된다.)

```
1  #include <iostream>
2  #define PROBLEM_NUM_MAX 10 // 최대 문제 수
3  using namespace std;
4
5  // 1번 문제의 정답은 answer[0]에, 2번 문제의 정답은 answer[1]에, ... 기입한다.
6  int answer[PROBLEM_NUM_MAX] = { ?  ?  ?  ?  ?  ?  ?  ?  ? };
7
8  int main(void) {
9      int problem_number;
10     cin >> problem_number;
11     for (int i = 1; i <= PROBLEM_NUM_MAX; i++) {
12         if (problem_number == i) {
13             cout << answer[i - 1] << endl;
14             break;
15         }
16     }
17     return 0;
18 }
19
```

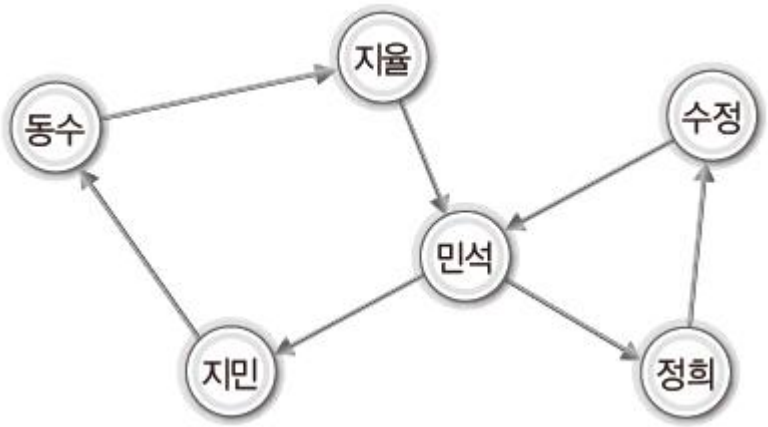
아래 10문제를 보고, 위 소스 코드를 완성하여라.

[1~4번 문제 : 다음 그림을 보고 물음에 답하라]



1번 문제. Root Node는 A에 해당한다. ( )
2번 문제. Edge는 총 6개이다. ( )
3번 문제. Leaf Node는 B,C,D,E,F에 해당한다. ( )
4번 문제. 노드 B, C, D는 부모 노드가 같으므로, 서로가 서로에게 sibling node이다. ( )

[5~7번 문제 : 다음 그림을 보고 물음에 답하라]



5번 문제. 위 그래프는 트리에 해당한다. ( )
6번 문제. 위 그래프는 방향 완전 그래프이다. ( )
7번 문제. 다음 그래프에서 V(G) = {동수, 지율, 민석, 지민, 수정, 정희} 이고,
E(G) = {<동수, 지율>, <지율,민석>, <민석, 지민>, <지민, 동수>, <민석, 정희>, <수정, 민석>, <정희, 수정>} 으로 나타낼 수 있다. ( )

8번 문제. 그래프는 인접 행렬과 인접 리스트로 나타낼 수 있다. ( )
9번 문제. 그래프의 탐색 방법 중 BFS는 깊이 우선 탐색으로 Stack을 사용한다. ( )
10번 문제. 최소 비용 신장 트리(MST)를 구성하기 위한 대표적인 알고리즘으로 Prim의 알고리즘과 Kruskal의 알고리즘이 있다. 이는 MST 내에 사이클이 존재하면 안되는 성질에 기반한다. ( )

Input
문제 번호가 입력으로 제시된다. 1번 문제의 경우 1, 2번 문제의 경우 2가 제시된다. 문제 번호는 1부터 5 사이의 정수이다.

Output
각 문제에 대한 답을 하나의 정수로 출력한다. 예를 들어, 1번 문제의 답이 3이라면 3을 출력한다.
아래 예시는 1번 문제의 답이 3일 때와, 2번 문제의 답이 5일 때를 나타낸 것이다.
예제는 채점하지 않으며 자신이 맞다고 생각하는 답을 출력하면 된다.

예제 1

Sample Input	Sample Output
1	3

예제 2

Sample Input	Sample Output
2	5

Problem D

에디터

제한 시간 : 0.3초, 메모리 제한 : 512MB, 문제유형 : 문제 해결

문제

한 줄로 된 간단한 에디터를 구현하려고 한다. 이 편집기는 영어 소문자만을 기록할 수 있는 편집기로, 최대 600,000글자까지 입력할 수 있다.

이 편집기에는 '커서'라는 것이 있는데, 커서는 문장의 맨 앞(첫 번째 문자의 왼쪽), 문장의 맨 뒤(마지막 문자의 오른쪽), 또는 문장 중간 임의의 곳(모든 연속된 두 문자 사이)에 위치할 수 있다. 즉 길이가 L인 문자열이 현재 편집기에 입력되어 있으면, 커서가 위치할 수 있는 곳은 L+1가지 경우가 있다.

이 편집기가 지원하는 명령어는 다음과 같다.

L	커서를 왼쪽으로 한 칸 옮김 (커서가 문장의 맨 앞이면 무시됨)
D	커서를 오른쪽으로 한 칸 옮김 (커서가 문장의 맨 뒤이면 무시됨)
B	커서 왼쪽에 있는 문자를 삭제함 (커서가 문장의 맨 앞이면 무시됨) 삭제로 인해 커서는 한 칸 왼쪽으로 이동한 것처럼 나타나지만, 실제로 커서의 오른쪽에 있던 문자는 그대로임
P $	$라는 문자를 커서 왼쪽에 추가함

초기에 편집기에 입력되어 있는 문자열이 주어지고, 그 이후 입력한 명령어가 차례로 주어졌을 때, 모든 명령어를 수행하고 난 후 편집기에 입력되어 있는 문자열을 구하는 프로그램을 작성하시오. 단, 명령어가 수행되기 전에 커서는 문장의 맨 뒤에 위치하고 있다고 한다.

입력

첫째 줄에는 초기에 편집기에 입력되어 있는 문자열이 주어진다. 이 문자열은 길이가 N이고, 영어 소문자로만 이루어져 있으며, 길이는 100,000을 넘지 않는다. 둘째 줄에는 입력할 명령어의 개수를 나타내는 정수 M(1 ≤ M ≤ 500,000)이 주어진다. 셋째 줄부터 M개의 줄에 걸쳐 입력할 명령어가 순서대로 주어진다. 명령어는 위의 네 가지 중 하나의 형태로만 주어진다.

출력

첫째 줄에 모든 명령어를 수행하고 난 후 편집기에 입력되어 있는 문자열을 출력한다.

예제 입력 1 복사

```
abcd
3
P x
L
P y
```

예제 출력 1 복사

```
abcdyx
```

예제 입력 2 복사

```
abc
9
L
L
L
L
L
P x
L
B
P y
```

예제 출력 2 복사

```
yxabc
```

예제 입력 3 복사

```
dmih
11
B
B
P x
L
B
B
B
P y
D
D
P z
```

예제 출력 3 복사

```
yxz
```

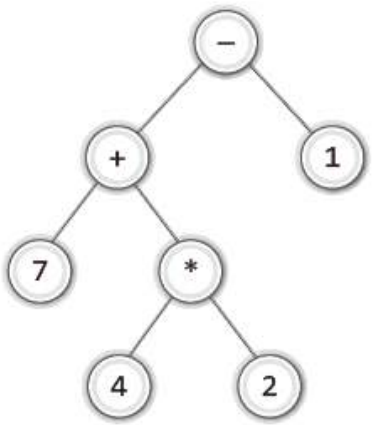
Problem E

수식 트리

제한 시간 : 1초, 메모리 제한 : 128MB, 문제유형 : 구현

문제

다음은 수식 트리(Expression Tree)를 구현하는 코드의 일부이다. 우리는 다음 코드를 완성시켜서 정상적으로 연산이 수행되게끔 해야 한다. 수식 트리(Expression Tree)는 이진 트리를 이용해서 수식을 표현해 놓은 것이며, 예를 들어 7 + 4 * 2 - 1의 식을 수식 트리로 표현할 경우 다음 그림과 같이 된다.



위의 수식 트리는, 다음과 같이 곱셈 연산을 먼저 수행하고, 그 뒤 덧셈 연산을, 마지막으로 뺄셈 연산을 수행한다.

▶ [그림 08-27: 수식 트리의 연산과정 1/3]

▶ [그림 08-28: 수식 트리의 연산과정 2/3]

▶ [그림 08-29: 수식 트리의 연산과정 3/3]

따라서, 다음 제시되는 코드는 중위 표기법의 수식을 후위 표기법의 수식으로 바꾼 뒤 수식 트리를 만드는 프로그램이며, 우리가 채워야 할 EvaluateExpTree 함수는 수식 트리에 담겨있는 수식을 계산하는 함수이다. 다음 코드를 보고, EvaluateExpTree 함수를 완성하여라.

현재 미완성된 프로그램은 다음과 같으며, EvaluateExpTree 함수만 수정 가능하다.
다른 헤더 파일의 삽입이나 삭제는 불가능하다.

다음 장에 코드가 있으며 EvaluateExpTree 함수 부분을 완성하여라. 출력 결과 예시는 다음과 같다.

```
Microsoft Visual Studio 디버그 콘솔
127*+5-
전위 표기법의 수식: - + 1 * 2 7 5 ..
중위 표기법의 수식: 1 + 2 * 7 - 5 ..
후위 표기법의 수식: 1 2 7 * + 5 - ..
연산의 결과: 10 ..

C:₩Users₩weonw₩Desktop₩수식 트리₩Debug₩수식 트리.exe(20620 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

Input

후위 표기법 수식이 제시된다. 각 숫자는 1 ~ 9 사이의 숫자(일의 자리의 숫자)이다. 예를 들어 12+3- 의 경우는 1 + 2 - 3을 의미하며 숫자 12를 의미하지 않는다.

Output

첫 번째 줄에 전위 표기법의 수식, 두 번째 줄에 중위 표기법의 수식, 세 번째 줄에 후위 표기법의 수식을 출력한다. 네 번째 줄에는 연산의 결과를 출력한다.

예제 1

Sample Input	Sample Output
127*+5-	전위 표기법의 수식 : - + 1 * 2 7 5 중위 표기법의 수식 : 1 + 2 * 7 - 5 후위 표기법의 수식 : 1 2 7 * + 5 - 연산의 결과 : 10

예제 2

Sample Input	Sample Output
78+1-	전위 표기법의 수식: - + 7 8 1 중위 표기법의 수식: 7 + 8 - 1 후위 표기법의 수식: 7 8 + 1 - 연산의 결과: 14

〈 ExpressionTree.c 〉

```
/*************** 헤더 파일 선언부 ***************/
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cctype>
#include <iostream>
using namespace std;

/******* 이진 트리와 관련된 부분을 선언,정의 *********/
typedef int BTData;

typedef struct _bTreeNode
{
	BTData data;
	struct _bTreeNode* left;
	struct _bTreeNode* right;
} BTreeNode;

BTreeNode* MakeBTreeNode(void);
BTData GetData(BTreeNode* bt);
void SetData(BTreeNode* bt, BTData data);

BTreeNode* GetLeftSubTree(BTreeNode* bt);
BTreeNode* GetRightSubTree(BTreeNode* bt);

void MakeLeftSubTree(BTreeNode* main, BTreeNode* sub);
void MakeRightSubTree(BTreeNode* main, BTreeNode* sub);

typedef void VisitFuncPtr(BTData data);

void PreorderTraverse(BTreeNode* bt, VisitFuncPtr action);
void InorderTraverse(BTreeNode* bt, VisitFuncPtr action);
void PostorderTraverse(BTreeNode* bt, VisitFuncPtr action);

BTreeNode* MakeBTreeNode(void)
{
	BTreeNode* nd = (BTreeNode*)malloc(sizeof(BTreeNode));

	nd->left = NULL;
	nd->right = NULL;
	return nd;
}

BTData GetData(BTreeNode* bt)
{
	return bt->data;
}

void SetData(BTreeNode* bt, BTData data)
{
	bt->data = data;
}

BTreeNode* GetLeftSubTree(BTreeNode* bt)
{
	return bt->left;
}
```

```
BTreeNode* GetRightSubTree(BTreeNode* bt)
{
	return bt->right;
}

void MakeLeftSubTree(BTreeNode* main, BTreeNode* sub)
{
	if (main->left != NULL)
		free(main->left);

	main->left = sub;
}

void MakeRightSubTree(BTreeNode* main, BTreeNode* sub)
{
	if (main->right != NULL)
		free(main->right);

	main->right = sub;
}

void PreorderTraverse(BTreeNode* bt, VisitFuncPtr action)
{
	if (bt == NULL)
		return;

	action(bt->data);
	PreorderTraverse(bt->left, action);
	PreorderTraverse(bt->right, action);
}

void InorderTraverse(BTreeNode* bt, VisitFuncPtr action)
{
	if (bt == NULL)
		return;

	InorderTraverse(bt->left, action);
	action(bt->data);
	InorderTraverse(bt->right, action);
}

void PostorderTraverse(BTreeNode* bt, VisitFuncPtr action)
{
	if (bt == NULL)
		return;

	PostorderTraverse(bt->left, action);
	PostorderTraverse(bt->right, action);
	action(bt->data);
}

/********************** 스택과 큐와 관련된 부분을 선언, 정의 ******************************/
#define TRUE	1
#define FALSE	0

typedef BTreeNode* Data;

typedef struct _node
```

```
{
	Data data;
	struct _node* next;
} Node;

typedef struct _listStack
{
	Node* head;
} ListStack;

typedef ListStack Stack;

void StackInit(Stack* pstack);
int SIsEmpty(Stack* pstack);

void SPush(Stack* pstack, Data data);
Data SPop(Stack* pstack);
Data SPeek(Stack* pstack);

void StackInit(Stack* pstack)
{
	pstack->head = NULL;
}

int SIsEmpty(Stack* pstack)
{
	if (pstack->head == NULL)
		return TRUE;
	else
		return FALSE;
}

void SPush(Stack* pstack, Data data)
{
	Node* newNode = (Node*)malloc(sizeof(Node));

	newNode->data = data;
	newNode->next = pstack->head;

	pstack->head = newNode;
}

Data SPop(Stack* pstack)
{
	Data rdata;
	Node* rnode;

	if (SIsEmpty(pstack)) {
		printf("Stack Memory Error!");
		exit(-1);
	}

	rdata = pstack->head->data;
	rnode = pstack->head;

	pstack->head = pstack->head->next;
	free(rnode);

	return rdata;
```

```
}

Data SPeek(Stack* pstack)
{
	if (SIsEmpty(pstack)) {
		printf("Stack Memory Error!");
		exit(-1);
	}

	return pstack->head->data;
}

/**********************수식 트리와 관련된 부분을 선언, 정의 *********************************/

BTreeNode* MakeExpTree(char exp[]);
int EvaluateExpTree(BTreeNode* bt);

void ShowPrefixTypeExp(BTreeNode* bt);
void ShowInfixTypeExp(BTreeNode* bt);
void ShowPostfixTypeExp(BTreeNode* bt);

BTreeNode* MakeExpTree(char exp[])
{
	Stack stack;
	BTreeNode* pnode;

	int expLen = strlen(exp);
	int i;

	StackInit(&stack);

	for (i = 0; i < expLen; i++)
	{
		pnode = MakeBTreeNode();

		if (isdigit(exp[i]))		// 피연산자일 경우
		{
			SetData(pnode, exp[i] - '0');
		}
		else						// 연산자일 경우
		{
			MakeRightSubTree(pnode, SPop(&stack));
			MakeLeftSubTree(pnode, SPop(&stack));
			SetData(pnode, exp[i]);
		}

		SPush(&stack, pnode);
	}

	return SPop(&stack);
}
```

```
int EvaluateExpTree(BTreeNode* bt)
{
	int op1, op2;

	// 이 부분을 구현해야 함!




}

void ShowNodeData(int data)
{
	if (0 <= data && data <= 9)
		printf("%d ", data);
	else
		printf("%c ", data);
}

void ShowPrefixTypeExp(BTreeNode* bt)
{
	PreorderTraverse(bt, ShowNodeData);
}

void ShowInfixTypeExp(BTreeNode* bt)
{
	InorderTraverse(bt, ShowNodeData);
}

void ShowPostfixTypeExp(BTreeNode* bt)
{
	PostorderTraverse(bt, ShowNodeData);
}

/********************************** 메인 함수 *************************************************/
int main(void)
{
	char exp[100];
	cin >> exp;

	BTreeNode* eTree = MakeExpTree(exp);

	printf("전위 표기법의 수식: ");
	ShowPrefixTypeExp(eTree); printf("..\n");

	printf("중위 표기법의 수식: ");
	ShowInfixTypeExp(eTree); printf("..\n");

	printf("후위 표기법의 수식: ");
	ShowPostfixTypeExp(eTree); printf("..\n");

	printf("연산의 결과: %d ..\n", EvaluateExpTree(eTree));

	return 0;
}
```

Problem F

오엑스를 맞춰보자 3

제한 시간 : 1초, 메모리 제한 : 128MB, 문제유형 : 이론

여러분은 두 달간 자료구조 이론에 대해 공부했었다. 이제 다음 오엑스 문제들에 대해 답할 시간이 되었다!

우리가 만들 프로그램은, 문제 번호가 입력으로 제시될 때,
그 문제에 해당하는 답을 출력하여 주는 프로그램을 완성해 보는 것이다.
이 때, 답이 O라면 1을, 답이 X라면 2를 출력하기로 하자.

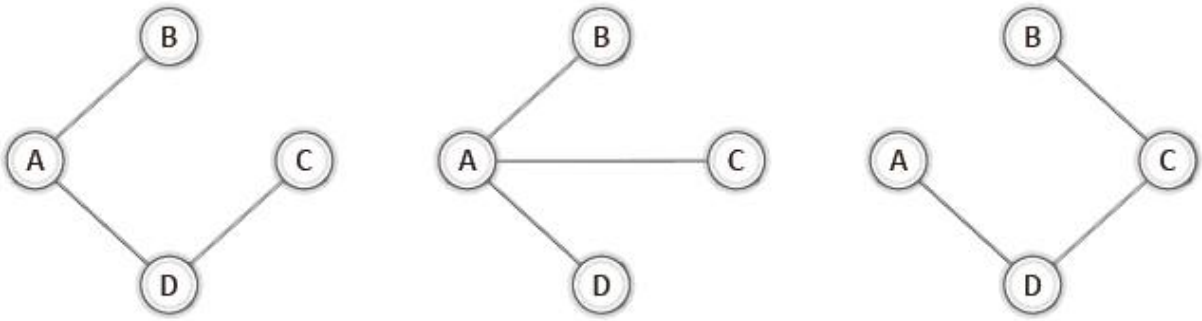
현재 미완성된 프로그램은 다음과 같다. (6번 줄에서, 배열의 빈칸에 정답을 채워 넣은 소스 코드를 제출하면 된다.)

```
1  #include <iostream>
2  #define PROBLEM_NUM_MAX 10 // 최대 문제 수
3  using namespace std;
4
5  // 1번 문제의 정답은 answer[0]에, 2번 문제의 정답은 answer[1]에, ... 기입한다.
6  int answer[PROBLEM_NUM_MAX] = { ?  ?  ?  ?  ?  ?  ?  ?  ? };
7
8  int main(void) {
9      int problem_number;
10     cin >> problem_number;
11     for (int i = 1; i <= PROBLEM_NUM_MAX; i++) {
12         if (problem_number == i) {
13             cout << answer[i - 1] << endl;
14             break;
15         }
16     }
17     return 0;
18 }
19
```

아래 10문제를 보고, O, X 여부를 판단하여 위 소스 코드를 완성하여라. 예를 들어, 1번 문제의 답이 O라면, 입력값이 1일 때 출력값은 1이다. 만약 3번 문제의 답이 X라면, 입력값이 3일 때 출력값은 2이다.

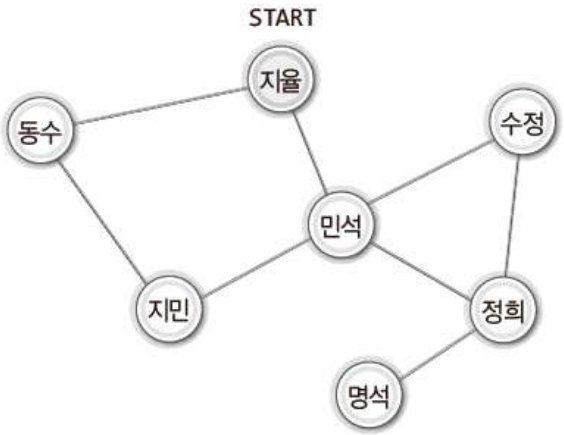
1번 문제. 스택은 먼저 들어간 것이 나중에 나오는 구조이다. ( )
2번 문제. 연결 리스트에서는 정렬 기능, 삽입 기능을 구현할 수 없다. ( )
3번 문제. 연결 리스트에서, 새 노드를 추가할 때 리스트의 머리에 저장할 경우 포인터 변수 tail이 불필요하다. ( )
4번 문제. 리스트는 연결 리스트만을 의미한다. ( )
5번 문제. O(1)은 상수형 빅-오라고 하며 데이터 수에 상관없이 연산횟수가 고정임을 의미한다. ( )
6번 문제. T(n) = $n^2 + \log n + n + 3$일 때 이를 Big O로 나타내면 O($\log n$)이다. ( )

[7 ~ 9번 문제] 다음 그림을 보고 물음에 답하라.



7번 문제. 위 그림들은 모두 방향 그래프이다. ( )
8번 문제. 위 그림들은 모두 트리이다. ( )
9번 문제. 위 그림들은 모두 신장 트리(Spanning Tree)에 해당한다. ( )

[10번 문제] 다음 그림을 보고 물음에 답하라.



10번 문제. 지율을 기준으로 너비 우선 탐색을 시행하면 조건에 따라 여러 가지 순서가 존재할 수 있으며, 지율, 동수, 민석, 지민, 수정, 정희, 명석의 순으로 탐색할 수 있다. ( )

Input
문제 번호가 입력으로 제시된다. 1번 문제의 경우 1, 2번 문제의 경우 2가 제시된다. 문제 번호는 1부터 5 사이의 정수이다.

Output
각 문제에 대한 답을 하나의 정수로 출력한다. 예를 들어, 1번 문제의 답이 3이라면 3을 출력한다.
아래 예시는 1번 문제의 답이 3일 때와, 2번 문제의 답이 5일 때를 나타낸 것이다.
예제는 채점하지 않으며 자신이 맞다고 생각하는 답을 출력하면 된다.

예제 1

Sample Input	Sample Output
1	3

예제 2

Sample Input	Sample Output
2	5

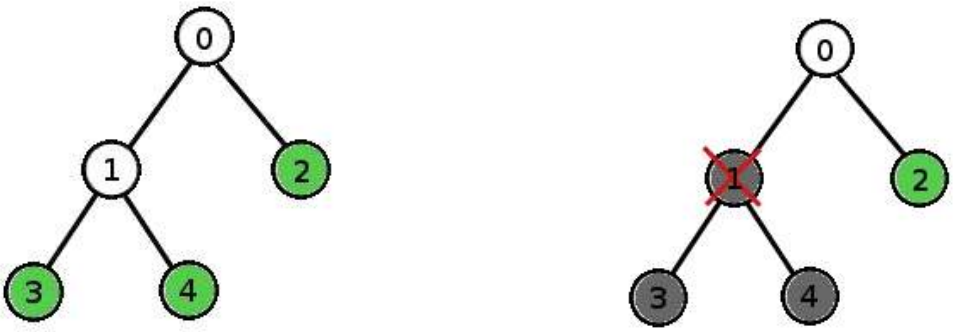
Problem G

유기체

제한 시간 : 2초, 메모리 제한 : 256MB, 문제유형 : 문제 해결

문제

생물학에서 유기체는, 첫 번째 세포에서 시작하여 유기체의 발달 과정 동안 각 세포는 2개의 다른 세포로 분할되거나 또는 전혀 분할되지 않습니다. 유기체는 모든 세포가 더 이상 분열되지 않을 때 성숙하다고 합니다. 유기체의 개발 과정에서, 각 세포에 고유 번호를 할당하였습니다. 예를 들어, 아래 왼쪽 그림에서 각 유기체가 세포 0으로 시작하여 세포 1과 2로 나뉘는 종을 고려해 봅시다.



세포 1은 세포 3과 4로 나뉘고, 세포 2, 3, 4는 나뉘지 않습니다. 따라서 이 종의 모든 성숙한 유기체는 정확히 3개이며, 2, 3, 4번 세포가 이에 해당합니다. 개발 과정에서 세포를 죽이면 자신과 그 밑의 성숙한 형태의 유기체 역시 죽게 됩니다. 만일 그 세포가 분열하는 세포인 경우, 세포는 분열되기 전에 사멸되기 때문에, 그 세포의 후손을 모두 잃게 됩니다. 위에서 오른쪽 그림을 봅시다. 예를 들어, 위에서 설명한 유기체가 있다고 할 때, 개발 과정에서 1번 세포를 죽이면 성숙한 유기체는 2번 세포만 남게 됩니다. 따라서 성숙한 유기체는 1개가 남게 됩니다.

Input

첫 번째 줄에는 세포의 개수 N이 주어집니다. N은 50보다 작거나 같은 자연수입니다. 두 번째 줄에는 유기체의 발달 과정을 설명하는 배열이 제공됩니다. 배열의 i번째 요소는 셀 i의 상위 셀입니다 (여기서 i는 0부터 시작합니다.) 최초 부모는 -1입니다. 세 번째 줄에는 개발 과정에서 죽일 세포 번호 하나가 제시됩니다.

입력되는 값의 조건은 다음과 같습니다.

1. 첫 번째 줄에서 제시되는 배열은 정확히 N 개의 요소를 포함하며, 여기서 N은 1에서 50 사이의 홀수 정수입니다.
2. 첫 번째 줄에서 제시되는 배열에는 정확히 하나의 "-1"요소가 있습니다.
3. 첫 번째 줄에서 제시되는 배열의 모든 요소는 -1과 N-1 사이에 있습니다.
4. 첫 번째 줄에서 제시되는 배열은 결론적으로 이진 트리를 형성합니다.
5. 두 번째 줄에서 제시되는 죽일 세포의 값은 0에서 N-1 사이입니다.

Output

세포 하나를 죽인 이후 남은 성숙한 유기체의 개수를 출력합니다.

예제

Sample Input	Sample Output
5 -1 0 0 1 1 2	2

Problem H

회전하는 큐

제한 시간 : 2초, 메모리 제한 : 128MB, 문제유형 : 문제 해결

문제

지민이는 N개의 원소를 포함하고 있는 양방향 순환 큐를 가지고 있다. 지민이는 이 큐에서 몇 개의 원소를 뽑아내려고 한다.

지민이는 이 큐에서 다음과 같은 3가지 연산을 수행할 수 있다.

1. 첫 번째 원소를 뽑아낸다. 이 연산을 수행하면, 원래 큐의 원소가 $a_1, ..., a_k$이었던 것이 $a_2, ..., a_k$와 같이 된다.
2. 왼쪽으로 한 칸 이동시킨다. 이 연산을 수행하면, $a_1, ..., a_k$가 $a_2, ..., a_k, a_1$이 된다.
3. 오른쪽으로 한 칸 이동시킨다. 이 연산을 수행하면, $a_1, ..., a_k$가 $a_k, a_1, ..., a_{k-1}$이 된다.

큐에 처음에 포함되어 있던 수 N이 주어진다. 그리고 지민이가 뽑아내려고 하는 원소의 위치가 주어진다. (이 위치는 가장 처음 큐에서의 위치이다.) 이때, 그 원소를 주어진 순서대로 뽑아내는데 드는 2번, 3번 연산의 최솟값을 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 큐의 크기 N과 뽑아내려고 하는 수의 개수 M이 주어진다. N은 50보다 작거나 같은 자연수이고, M은 N보다 작거나 같은 자연수이다. 둘째 줄에는 지민이가 뽑아내려고 하는 수의 위치가 순서대로 주어진다. 위치는 1보다 크거나 같고, N보다 작거나 같은 자연수이다.

출력

첫째 줄에 문제의 정답을 출력한다.

예제 입력 1 복사

```
10 3
1 2 3
```

예제 출력 1 복사

```
0
```

예제 입력 2 복사

```
10 3
2 9 5
```

예제 출력 2 복사

```
8
```

예제 입력 3 복사

```
32 6
27 16 30 11 6 23
```

예제 출력 3 복사

```
59
```

예제 입력 4 복사

```
10 10
1 6 3 2 7 9 8 4 10 5
```

예제 출력 4 복사

```
14
```

Problem I

객관식을 맞춰보자 1

제한 시간 : 1초, 메모리 제한 : 128MB, 문제유형 : 이론

문제

여러분은 두 달간 자료구조 이론에 대해 공부했었다. 이제 다음 오엑스 문제들에 대해 답할 시간이 되었다!

우리가 만들 프로그램은, 문제 번호가 입력으로 제시될 때,
그 문제에 해당하는 답을 출력하여 주는 프로그램을 완성해 보는 것이다.
이 때, 답이 O라면 1을, 답이 X라면 2를 출력하기로 하자.

현재 미완성된 프로그램은 다음과 같다. (6번 줄에서, 배열의 빈칸에 정답을 채워 넣은 소스 코드를 제출하면 된다.)

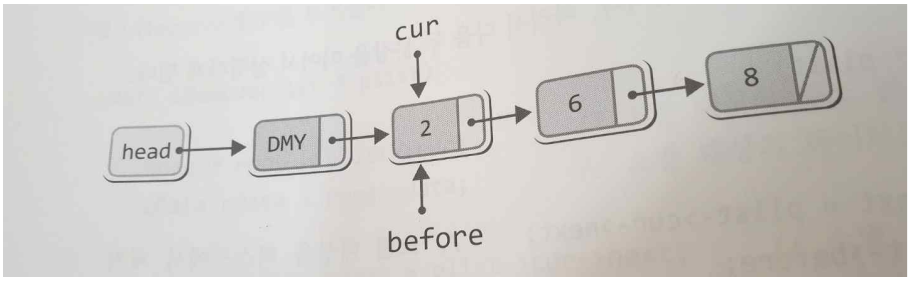
```
1  #include <iostream>
2  #define PROBLEM_NUM_MAX 10 // 최대 문제 수
3  using namespace std;
4
5  // 1번 문제의 정답은 answer[0]에, 2번 문제의 정답은 answer[1]에, ... 기입한다.
6  int answer[PROBLEM_NUM_MAX] = { ㅇ  ㅇ  ㅇ  ㅇ  ㅇ  ㅇ  ㅇ  ㅇ  ㅇ };
7
8  int main(void) {
9      int problem_number;
10     cin >> problem_number;
11     for (int i = 1; i <= PROBLEM_NUM_MAX; i++) {
12         if (problem_number == i) {
13             cout << answer[i - 1] << endl;
14             break;
15         }
16     }
17     return 0;
18 }
19
```

1번 문제. 다음 중 틀린 것은? ( )
(1) 이진 탐색 알고리즘(binary search algorithm)보다 순차 탐색 알고리즘(sequential search algorithm)이 더 느리다.
(2) ADT(Abstract Data Type)는 추상 자료형이라고 하며, 구체적인 기능의 완성과정을 언급한 것이다.
(3) 리스트는 배열을 통해서도 구현할 수 있다.
(4) 연결 리스트에서 더미 노드(Dummy Node)를 사용하는 이유는 노드의 추가, 삭제 및 조회 과정을 일관되게 구성할 수 있기 때문이다.
(5) 큐는 선입선출(FIFO)의 성질을 지닌다. 즉, 먼저 들어온 값이 먼저 나가게 된다.

2번 문제. 다음 보기 중 가장 적절하지 않은 것은? ( )
(1) 재귀함수의 문제점 중 하나는 중복된 계산이 시행될 수 있다는 점이다.
(2) T(n) = 3n + 2일 때, O(n) = n이다.
(3) 일반적인 이진 탐색 알고리즘의 시간 복잡도는 O(log n)이다.

(4) 리스트를 자료형과 포인터 변수를 이용하여 구현하는 것의 장점은 데이터의 참조가 쉽다는 점이다.
(5) 스택을 이용해서 계산기 프로그램을 구현할 수 있다.

3번 문제. 다음 그림에 가장 잘 맞는 자료구조는 무엇인가? ( )



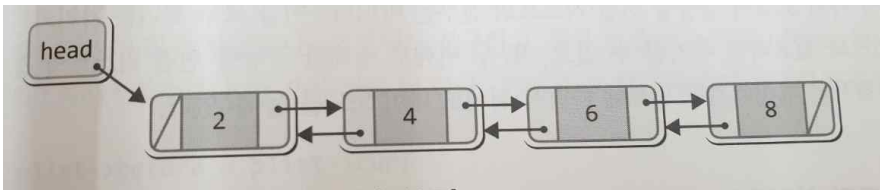
(1) 단순 연결 리스트 **(2) 양방향 연결 리스트**
(3) 원형 연결 리스트 **(4) 순차 리스트** **(5) 배열**

4번 문제. <보기>에 있는 수행시간 함수들의 시간복잡도를 Big-O로 나타내고자 한다. Big-O가 작은 것부터 큰 것까지 정렬된 것을 찾아라. ( )

< 보기 >
$n^2+3n+100000,\ n!,\ 2^n, 3^n, 1, \log n$

(1) $\log n < 1 < n^2+3n+100000 < n! < 2^n < 3^n$
(2) $\log n < 1 < n^2+3n+100000 < 2^n < 3^n < n!$
(3) $1 < \log n < n^2+3n+100000 < n! < 2^n < 3^n$
(4) $1 < \log n < n^2+3n+100000 < 2^n < 3^n < n!$
(5) $n^2+3n+100000 < \log n < 1 < 2^n < 3^n < n!$

5번 문제. 다음 그림이 가장 잘 나타내는 자료구조는 무엇인가? ( )



(1) 단순 연결 리스트 **(2) 양방향 연결 리스트**
(3) 원형 연결 리스트 **(4) 배열** **(5) 순차 리스트**

Input
문제 번호가 입력으로 제시된다. 1번 문제의 경우 1, 2번 문제의 경우 2가 제시된다. 문제 번호는 1부터 5 사이의 정수이다.

Output
각 문제에 대한 답을 하나의 정수로 출력한다. 예를 들어, 1번 문제의 답이 3이라면 3을 출력한다.
아래 예시는 1번 문제의 답이 3일 때와, 2번 문제의 답이 5일 때를 나타낸 것이다.
예제는 채점하지 않으며 자신이 맞다고 생각하는 답을 출력하면 된다.

예제 1

Sample Input	Sample Output
1	3

예제 2

Sample Input	Sample Output
2	5

Problem J

요세푸스 문제

제한 시간 : 2초, 메모리 제한 : 256MB, 문제유형 : 문제 해결

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 빈 칸을 사이에 두고 순서대로 주어진다. (1 ≤ K ≤ N ≤ 5,000)

출력

예제와 같이 요세푸스 순열을 출력한다.

예제 입력 1 복사

```
7 3
```

예제 출력 1 복사

```
<3, 6, 2, 7, 5, 1, 4>
```

Problem K

트리인가?

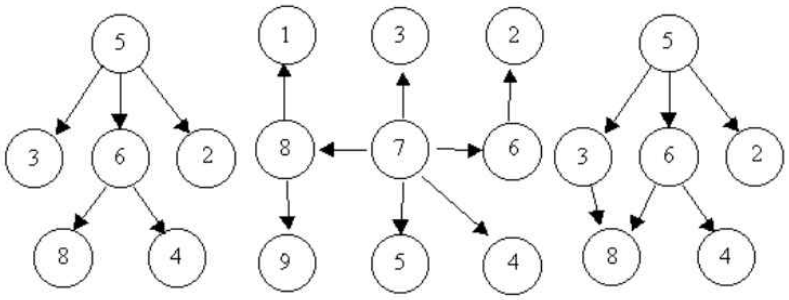
제한 시간 : 1초, 메모리 제한 : 128MB, 문제 유형 : 문제 해결

문제

트리는 굉장히 잘 알려진 자료 구조이다. 트리를 만족하는 자료 구조는 비어 있거나(노드의 개수가 0개), 노드의 개수가 1개 이상이고 방향 간선이 존재하며 다음과 같은 조건을 만족해야 한다. 이때, 노드 u에서 노드 v로 가는 간선이 존재하면 간선을 u에 대해서는 '나가는 간선', v에 대해서는 '들어오는 간선'이라고 하자.

1. 들어오는 간선이 하나도 없는 단 하나의 노드가 존재한다. 이를 루트(root) 노드라고 부른다.
2. 루트 노드를 제외한 모든 노드는 반드시 단 하나의 들어오는 간선이 존재한다.
3. 루트에서 다른 노드로 가는 경로는 반드시 가능하며, 유일하다. 이는 루트를 제외한 모든 노드에 성립해야 한다.

아래의 그림을 보자. 원은 노드, 화살표는 간선을 의미하며, 화살표의 방향이 노드 u에서 노드 v로 향하는 경우 이는 이 간선이 u에서 나가는 간선이며 v로 들어오는 간선이다. 3개의 그림 중 앞의 2개는 트리지만 뒤의 1개는 트리가 아니다.



당신은 간선의 정보들을 받아서 해당 케이스가 트리인지를 판별해야 한다.

입력

입력은 여러 개의 테스트 케이스로 이루어져 있으며, 입력의 끝에는 두 개의 음의 정수가 주어진다.

각 테스트 케이스는 여러 개의 정수쌍으로 이루어져 있으며, 테스트 케이스의 끝에는 두 개의 0이 주어진다.

각 정수쌍 u, v에 대해서 이는 노드 u에서 노드 v로 가는 간선이 존재함을 의미한다. u와 v는 0보다 크다.

출력

각 테스트 케이스에 대해서, 테스트 케이스의 번호가 k일 때(k는 1부터 시작하며, 1씩 증가한다) 트리일 경우 "Case k is a tree."를, 트리가 아닐 경우 "Case k is not a tree."를 출력한다.

예제 입력 1 복사

```
6 8  5 3  5 2  6 4
5 6  0 0

8 1  7 3  6 2  8 9  7 5
7 4  7 8  7 6  0 0

3 8  6 8  6 4
5 3  5 6  5 2  0 0
-1 -1
```

예제 출력 1 복사

```
Case 1 is a tree.
Case 2 is a tree.
Case 3 is not a tree.
```

Problem L

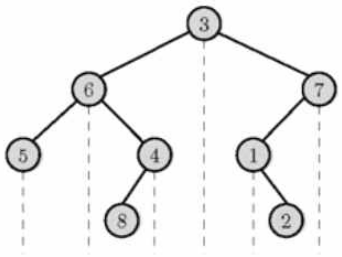
트리의 순회

제한 시간 : 1초, 메모리 제한 : 256MB, 문제 유형 : 구현

문제

이진 트리는 매우 중요한 기본 자료 구조이다. 아래 그림은 루트 노드가 유일한 이진 트리이다. 모든 노드는 최대 2개의 자식 노드를 가질 수 있으며, 왼쪽 자식이 순서가 먼저이다. 노드 n개로 이루어진 이진 트리를 BT라고 하자. BT의 노드는 1부터 n까지 유일한 번호가 매겨져 있다.

아래 그림에 나와있는 BT의 루트는 3번 노드이다. 1번 노드는 오른쪽 자식만 가지고 있고, 4와 7은 왼쪽 자식만 가지고 있다. 3과 6은 왼쪽과 오른쪽 자식을 모두 가지고 있다. 나머지 노드는 모두 자식이 없으며, 이러한 노드는 리프 노드라고 부른다.



BT의 모든 노드를 순회하는 방법은 전위 순회(preorder), 중위 순회(inorder), 후위 순회(postorder)로 총 세 가지가 있다. 이 세 방법은 아래에 C 스타일의 의사 코드로 나와 있다. BT의 노드 v에 대해서, v.left는 왼쪽 자식, v.right는 오른쪽 자식을 나타낸다. v가 왼쪽 자식이 없으면 v.left는 ∅와 같고, 오른쪽 자식이 없으면 v.right는 ∅와 같다.

```
preorder(node v)
{
   printf(id number of v);
   if (v.left ≠ ∅)
     preorder(v.left);
   if (v.right ≠ ∅)
     preorder(v.right);
}
```

```
inorder(node v)
{
   if (v.left ≠ ∅)
     inorder(v.left);
   printf(id number of v);
   if (v.right ≠ ∅)
     inorder(v.right);
}
```

```
postorder(node v)
{
   if (v.left ≠ ∅)
     postorder(v.left);
   if (v.right ≠ ∅)
     postorder(v.right);
   printf(id number of v);
}
```

BT를 전위 순회, 중위 순회한 결과가 주어진다. 즉, 위의 함수 중 preorder(root node of BT)와 inorder(root node of BT)를 호출해서 만든 리스트가 주어진다. 두 순회한 결과를 가지고 다시 BT를 만들 수 있다. BT의 전위, 중위 순회한 결과가 주어졌을 때, 후위 순회했을 때의 결과를 구하는 프로그램을 작성하시오.

예를 들어, 위의 그림을 전위 순회하면 3,6,5,4,8,7,1,2, 중위 순회하면 5,6,8,4,3,1,2,7이 된다. 이를 이용해 후위 순회하면 5,8,4,6,2,1,7,3이 된다.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스의 첫째 줄에는 노드의 개수 n이 주어진다. ($1 \le n \le 1{,}000$) BT의 모든 노드에는 1부터 n까지 서로 다른 번호가 매겨져 있다. 다음 줄에는 BT를 전위 순회한 결과, 그 다음 줄에는 중위 순회한 결과가 주어진다. 항상 두 순회 결과로 유일한 이진 트리가 만들어지는 경우만 입력으로 주어진다.

출력

각 테스트 케이스마다 후위 순회한 결과를 출력 한다.

예제 입력 1 복사

```
2
4
3 2 1 4
2 3 4 1
8
3 6 5 4 8 7 1 2
5 6 8 4 3 1 2 7
```

예제 출력 1 복사

```
2 4 1 3
5 8 4 6 2 1 7 3
```

Problem M

경로 찾기

제한 시간 : 1초, 메모리 제한 : 256MB, 문제 유형 : 문제 해결

문제

가중치 없는 방향 그래프 G가 주어졌을 때, 모든 정점 (i, j)에 대해서, i에서 j로 가는 경로가 있는지 없는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정점의 개수 N (1 ≤ N ≤ 100)이 주어진다. 둘째 줄부터 N개 줄에는 그래프의 인접 행렬이 주어진다. i번째 줄의 j번째 숫자가 1인 경우에는 i에서 j로 가는 간선이 존재한다는 뜻이고, 0인 경우는 없다는 뜻이다. i번째 줄의 i번째 숫자는 항상 0이다.

출력

총 N개의 줄에 걸쳐서 문제의 정답을 인접행렬 형식으로 출력한다. 정점 i에서 j로 가는 경로가 있으면 i번째 줄의 j번째 숫자를 1로, 없으면 0으로 출력해야 한다.

예제 입력 1 복사

```
3
0 1 0
0 0 1
1 0 0
```

예제 출력 1 복사

```
1 1 1
1 1 1
1 1 1
```

예제 입력 2 복사

```
7
0 0 0 1 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
0 0 0 0 1 1 0
1 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 1 0 0 0 0
```

예제 출력 2 복사

```
1 0 1 1 1 1 1
0 0 1 0 0 0 1
0 0 0 0 0 0 0
1 0 1 1 1 1 1
1 0 1 1 1 1 1
0 0 1 0 0 0 1
0 0 1 0 0 0 0
```

Problem N

BFS와 DFS

제한 시간 : 2초, 메모리 제한 : 128MB, 문제 유형 : 구현

문제

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

입력

첫째 줄에 정점의 개수 N(1 ≤ N ≤ 1,000), 간선의 개수 M(1 ≤ M ≤ 10,000), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

출력

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

예제 입력 1 복사

```
4 5 1
1 2
1 3
1 4
2 4
3 4
```

예제 출력 1 복사

```
1 2 4 3
1 2 3 4
```

예제 입력 2 복사

```
5 5 3
5 4
5 2
1 2
3 4
3 1
```

예제 출력 2 복사

```
3 1 2 5 4
3 1 4 2 5
```

예제 입력 3 복사

```
1000 1 1000
999 1000
```

예제 출력 3 복사

```
1000 999
1000 999
```

Problem O

스택 수열

제한 시간 : 2초, 메모리 제한 : 128MB, 문제 유형 : 문제 해결

문제

스택 (stack)은 기본적인 자료구조 중 하나로, 컴퓨터 프로그램을 작성할 때 자주 이용되는 개념이다. 스택은 자료를 넣는 (push) 입구와 자료를 뽑는 (pop) 입구가 같아 제일 나중에 들어간 자료가 제일 먼저 나오는 (LIFO, Last in First out) 특성을 가지고 있다.

1부터 n까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 순서는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라.

입력

첫 줄에 n (1 ≤ n ≤ 100,000)이 주어진다. 둘째 줄부터 n개의 줄에는 수열을 이루는 1이상 n이하의 정수가 하나씩 순서대로 주어진다. 물론 같은 정수가 두 번 나오는 일은 없다.

출력

입력된 수열을 만들기 위해 필요한 연산을 한 줄에 한 개씩 출력한다. push연산은 +로, pop 연산은 -로 표현하도록 한다. 불가능한 경우 NO를 출력한다.

예제 입력 1 복사

```
8
4
3
6
8
7
5
2
1
```

예제 출력 1 복사

```
+
+
+
+
-
-
+
+
-
+
+
-
-
-
-
-
```

예제 입력 2 복사

```
5
1
2
5
3
4
```

예제 출력 2 복사

```
NO
```

힌트

1부터 n까지에 수에 대해 차례로 [push, push, push, push, pop, pop, push, push, pop, push, push, pop, pop, pop, pop, pop] 연산을 수행하면 수열 [4, 3, 6, 8, 7, 5, 2, 1]을 얻을 수 있다.

Problem P

이중 연결 리스트

제한 시간 : 5초, 메모리 제한 : 512MB, 문제 유형 : 구현

문제

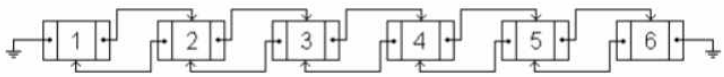
창영이는 1학년 때 숙제로 했던 이중 연결 리스트 소스를 상근이에게 생일 선물로 보내주었다. 상근이는 드디어 자신이 원하던 기능이 있는 소스를 받게 되어서 매우 기뻤다. 상근이는 하루종일 이 리스트를 가지고 놀려고 한다.

리스트에는 총 N개의 노드가 포함되어 있고, 가장 왼쪽 노드가 1번이며 나머지는 오른쪽으로 갈 수록 1씩 번호가 증가한다. 리스트가 수행할 수 있는 연산은 아래와 같이 2가지이다.

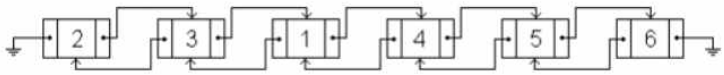
A) 노드 X를 노드 Y의 앞으로 이동

B) 노드 X를 노드 Y의 뒤으로 이동

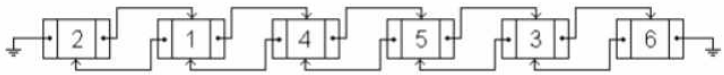
아래 그림은 노드가 6개인 이중 연결 리스트의 모습이다.



여기에 "A 1 4" 연산을 수행하면 아래와 같이 된다. (노드 1을 노드 4의 앞으로 이동)



그 다음, "B 3 5" 연산을 수행하면 아래 그림과 같은 모습이 된다. (노드 3을 노드 5의 뒤로 이동)



모든 연산을 수행하고 난 뒤의 이중 연결 리스트의 값을 head에서 tail까지 출력하는 프로그램을 작성하여 보자! 편의상 위 그림에서 제일 왼쪽에 있는 노드를 Head라 하고, 제일 오른쪽에 있는 노드를 Tail이라고 하자.

Input

첫째 줄에 노드의 수 N과 연산의 수 M이 주어진다. ($2 \le N \le 100$, $0 \le M \le 10$)
두 번째 줄에 초기에 설정된 이중 연결 리스트의 값들이 Head부터 Tail까지 제시된다. 편의상 위 그림에서 제일 왼쪽에 있는 노드를 Head라 하고, 제일 오른쪽에 있는 노드를 Tail이라고 하자.
다음 M개 줄에는 상근이가 입력한 연산이 문제 설명에 나온 형식으로 주어진다.

Output

첫째 줄에 처음 상태로 만들기 위해서 필요한 연산의 최솟값을 출력한다. 이 값을 K라고 한다.
다음 K개 줄에는 리스트가 수행해야 하는 연산을 순서대로 출력한다.

예제

Sample Input	Sample Output
6 2 1 2 3 4 5 6 A 1 4 B 3 5	2 1 4 5 3 6

Problem Q

하노이의 탑

제한 시간 : 5초, 메모리 제한 : 512MB, 문제 유형 : 문제 해결

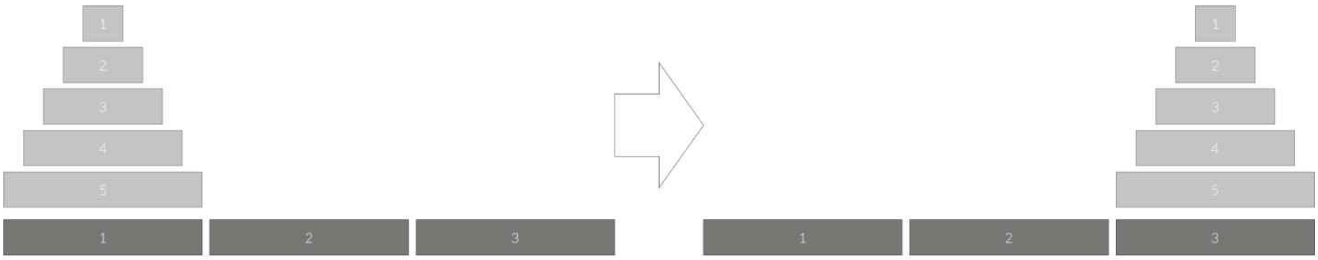
문제

세 개의 장대가 있고 첫 번째 장대에는 반경이 서로 다른 n개의 원판이 쌓여 있다. 각 원판은 반경이 큰 순서대로 쌓여있다. 이제 수도승들이 다음 규칙에 따라 첫 번째 장대에서 세 번째 장대로 옮기려 한다.

1. 한 번에 한 개의 원판만을 다른 탑으로 옮길 수 있다.
2. 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다.

이 작업을 수행하는데 필요한 이동 순서를 출력하는 프로그램을 작성하라. 단, 이동 횟수는 최소가 되어야 한다.

아래 그림은 원판이 5개인 경우의 예시이다.



Input

첫째 줄에 첫 번째 장대에 쌓인 원판의 개수 N (1 ≤ N ≤ 6) 이 주어진다.

Output

첫째 줄에 옮긴 횟수 K를 출력한다.

두 번째 줄부터 수행 과정을 출력한다. 두 번째 줄부터 K개의 줄에 걸쳐 두 정수 A B를 빈칸을 사이에 두고 출력하는데, 이는 A번째 탑의 가장 위에 있는 원판을 B번째 탑의 가장 위로 옮긴다는 뜻이다.

예제 입력 1 복사

```
3
```

예제 출력 1 복사

```
7
1 3
1 2
3 2
1 3
2 1
2 3
1 3
```

Problem R

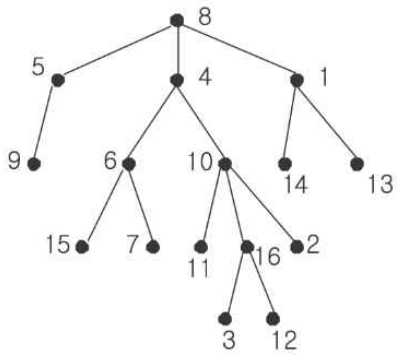
가장 가까운 공통 조상

제한 시간 : 1초, 메모리 제한 : 128MB, 문제 유형 : 문제 해결

문제

루트가 있는 트리(rooted tree)가 주어지고, 그 트리 상의 두 정점이 주어질 때 그들의 가장 가까운 공통 조상(Nearest Common Anscestor)은 다음과 같이 정의됩니다.

- 두 노드의 가장 가까운 공통 조상은, 두 노드를 모두 자손으로 가지면서 깊이가 가장 깊은(즉 두 노드에 가장 가까운) 노드를 말합니다.



예를 들어 15와 11를 모두 자손으로 갖는 노드는 4와 8이 있지만, 그 중 깊이가 가장 깊은(15와 11에 가장 가까운) 노드는 4 이므로 가장 가까운 공통 조상은 4가 됩니다.

루트가 있는 트리가 주어지고, 두 노드가 주어질 때 그 두 노드의 가장 가까운 공통 조상을 찾는 프로그램을 작성하세요

입력

첫 줄에 테스트 케이스의 개수 T가 주어집니다.

각 테스트 케이스마다, 첫째 줄에 트리를 구성하는 노드의 수 N이 주어집니다. ($2 \leq N \leq 10,000$)

그리고 그 다음 N-1개의 줄에 트리를 구성하는 간선 정보가 주어집니다. 한 간선 당 한 줄에 두 개의 숫자 A B 가 순서대로 주어지는데, 이는 A가 B의 부모라는 뜻입니다. (당연히 정점이 N개인 트리는 항상 N-1개의 간선으로 이루어집니다!) A와 B는 1 이상 N 이하의 정수로 이름 붙여집니다.

테스트 케이스의 마지막 줄에 가장 가까운 공통 조상을 구할 두 노드가 주어집니다.

출력

각 테스트 케이스 별로, 첫 줄에 입력에서 주어진 두 노드의 가장 가까운 공통 조상을 출력합니다.

예제 입력 1 복사

```
2
16
1 14
8 5
10 16
5 9
4 6
8 4
4 10
1 13
6 15
10 11
6 7
10 2
16 3
8 1
16 12
16 7
5
2 3
3 4
3 1
1 5
3 5
```

예제 출력 1 복사

```
4
3
```