www.kerinet.re.kr

정보올림피아드 알고리즘 자료집

중·고등부용

경기도교육정보연구원
Gyeonggi Province Educational Information Research Institute

차 례

1 짐 나르기

▸▸ 조합론에 기본을 둔 문제로 다양한 방법의 경우의 수를 구해내는 문제이다. 각각 중복이 가능하면서 순서가 없는 조합을 구하는 것이 핵심이 된다. 프로그램과 문제해결 및 검증의 기본이 되는 조합론에 대한 학습이다.

문제 (난이도 ★☆☆☆☆)

한 항구에 적재되어야 할 많은 컨테이너들이 있다. 이 컨테이너들은 색상을 가지며 이 색상에 의해 구분되어진다. 각 색상은 컨테이너의 종류를 나타내기 위한 것이다. 컨테이너 이 색상은 알파벳 값으로 주어진다. 드디어 컨테이너를 적재하는 시간이 되었다. 준비된 컨테이너들 중 몇 개를 타이타닉호에 적재해야 한다. 세준이 컨테이너를 적재할 차례가 되었는데 컨테이너들이 너무 많아서 무엇을 가지고 가야할지 고민하고 있다. 현재 항구에 남아있는 컨테이너의 총 개수와 그 종류, 그리고 세준이 적재할 컨테이너의 수가 주어졌을 때 컨테이너를 고르는 방법들을 모두 구하는 것이다. 단, 반드시 주어지는 수 만큼의 컨테이너를 골라야 한다.

프로그램의 이름은 “container”로 한다.

입력 형식

입력파일의 이름은 “Input.txt”로 하며 두 줄로 이루어진다.
첫 번째 줄에 컨테이너의 수 n과 고를 컨테이너의 수 k가 차례로 빈칸으로 구분되어 주어진다.($1 \le k \le n \le 100$).
다음 줄에는 준비된 파일의 이름(영문 소문자 하나) n개가 빈칸 하나로 구분되어 주어진다.
같은 색의 컨테이너가 여럿 존재할 수도 있다. 또한 같은 색의 컨테이너는 순차적으로 정렬이 되어 입력된다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
첫 번째 줄에 가능한 방법의 수 m을 출력한다.
다음 m개의 줄에는 컨테이너를 고르는 방법을 하나씩 출력한다. 각 줄에는 고른 컨테이너의 이름들을 빈칸으로 구분하여 출력한다. 한 줄에 출력되는 컨테이너들은 알파벳순으로 출력하며, 각각의 방법들도 사전 순서대로 출력되어야 한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
10 2
a a a b b c c c d e
```

출력의 예(Output.txt)

```
13
a a
a b
a c
a d
a e
b b
b c
b d
b e
c c
c d
c e
d e
```

2 문제의 배경 및 관련 이론

순열, 조합, 중복 조합 등 이산 수학의 한 내용으로 프로그래밍에 있어 가장 기본이 되는 수학적 지식이다. 이 문제는 조합과 순열에 관한 내용이 기본이면서 변형된 형태로 구성된 문제이다. 우선 간단하게 앞서 제시된 내용을 살펴보면 다음과 같다.

첫째, 순열은 n개의 원소를 가진 집합에서 모든 원소를 순차적으로 나열하는 방법의 수를 나타내는 것으로 순서에 관계를 가지는 것으로 경우의 수는 $_nP_r = \frac{n!}{(n-r)!}$으로 표시하며, n개 중 n개를 뽑는 순열의 수는 $n \times n \times n \times \cdots \times n = n^n$이 된다.

둘째, 조합은 순서를 고려하지 않고 뽑는 순열로 $_nC_r$로 표시 하며 $\frac{n!}{r!(n-r)!}$로 계산된다.

셋째, 중복조합은 반복을 허용하고 순서를 고려하지 않는 경우로 $a_1, a_1, \cdots, a_2$중에서 r개를 뽑는 방법의 수는 일반적으로 $S(k, r)$로 표현한다. 이것은 $k-1$개의 0과 r개의 1을 k개의 그룹으로 나누는 방법의 수이다. 그러므로 $C(k-1+r, k-1) = C(k-1+r, r)$이다.

이와 더불어 문제의 해가 될 수 있는 다양한 경우의 수를 구하게 순환(재귀) 방법이 있다. 이것은 여러 개의 함수로 구성된 프로그램에서 함수는 다른 함수를 호출 할 수 있는데 때로는 함수가 자기 자신을 호출하기도 하는데 이것을 순환 호출이라 부른다. 순환 호출을 이용하여 주어진 문제보다 “크기가 작은 문제”를 풀어서 원래 문제를 해결하는 방식의 알고리즘을 순환 알고리즘이라 한다. 또한 이 순환 호출을 위해서는 종료조건의 제시와 크기가 더 큰 문제를 순환 호출하여선 안 된다는 점을 주의해야 한다. 이 순환(재귀) 호출은 프로그램의 소스 코드의 양을 줄이는데 효과적이나 반드시 프로그램의 속도를 줄이는 효과를 가지는 것은 아님을 항상 기억해야 한다. 순환(재귀)로 풀 수 있는 문제는 비 순환(비 재귀)를 통하여서 풀 수 있다는 것도 기억하며 어떤 부분에서 또는 어떤 문제에서 순환(재귀)를 비 순환(비 재귀)를 사용할 것인가에 대한 판단도 중요하다.

이런 기본적인 지식을 토대로 이 문제의 해가 될 수 있는 범위를 가늠해 볼 수 있으며 최적화나 그 외의 유형의 문제에서는 탐색 공간에 대한 범위를 생각해 볼 수 있는 기본 학습이 될 것이다.

3 문제 해설

이 문제는 위에서의 설명과 같이 ${}_nC_r$의 정도의 탐색 공간을 가지는 문제이다. 즉, n의 값이 3이고 k의 값이 2인 경우 3가지의 경우의 수가 된다. 그러나 이 문제는 주어지는 n의 값이 같은 종류의 컨테이너가 제시될 수 있다는 것이다. n의 크기가 커지면 그것에 대한 경우의 수도 무수히 많아진다. 하지만 이 문제는 경우의 수를 모두 살펴보는 방법뿐이다. 이런 문제는 퇴각 검색이라는 알고리즘을 사용한다.

재귀 호출 방법에 대한 기본적인 방법은 다음과 같다. 예를 들어,
문자의 종류가 a, b, c로 주어지고 이 중에서 2개의 문자를 선택하여 나열 하는 방법을 구하는 문제라고 하면 단, 순서에 관계가 없다고 가정한다.

```
For  i 값을 하나씩 증가해 가면서 (i = 0부터 시작)
{
        i 번째 문자 "a" 임시 배열로 가져오고 cnt 값을 하나 증가한다. (a,  )
        if cnt < k 이면
        {
                (i + 1) 번째 문자 "a"를 가져온다. (a, b)
                cnt 값을 하나 증가한다.
        }
        else
        {
                (i + 1) 번째 문자를 원래의 배열로 가져간다. (a,  )
                cnt 값을 하나 감소한다.
                원래 배열의 (i + 2) 번째 문자를 임시 배열로 가져오고 cnt 값을
                하나 증가한다. (a, c)
                원래 배열의 마지막까지 위 과정을 반복한다.
        }
}
```

위와 같이 해결할 수 있으며 이것을 재귀 호출의 방법으로 코딩하면 간단하게 구현된다. 단, 이번 문제는 앞에서 언급한 바와 같이 주어지는 배열의 문자 값이 중복되어질 수 있다는 점을 고려하여야 한다. 그래서 위와 같은 방법으로 해결하면 같은 조합이 나오게 되므로 이미 한번 나온 조합에 대해서는 배제되도록 하여야 한다.

방법은 주어지는 문자의 배열에 개수를 세어 하나의 배열에 저장하고 저장된 값을 하나씩 줄여가면서 재귀 호출하면 된다.

아래의 소스는 주된 내용은 재귀 호출에 관한 부분이다. 각 배열과 변수가 가지는 의미는 다음과 같다.

1) container[26] : 같은 종류의 컨테이너 수를 저장하는 배열

container[26] =

2	1	0	…	0

2) temp[26] : selected[10000][26] 배열에 저장하기 전에 선택의 방법을 임시로 저장하 배열

3) selected[10000][26] : 해가 될 경우의 수를 10000으로 한정하고 실제 구한 선택 방법 저장하는 배열로 배열의 값은 0, 1, 2, …, 25로 저장된다.

4) cnt : temp[26] 배열의 첨자

```
for(int i=pre;i<=25;i++)
{
	if(containers[i] > 0)
	{
		containers[i]--;
		temp[cnt] = i; // cnt 는 선택될 컨테이너의 수를 세는 카운터 변수
		back(i,cnt+1);
		temp[cnt] = 0;
		containers[i]++;
	}
}
```

또한 소스 코드에서는 컨테이너의 종류가 알파벳으로 주어지는 입력형식을 고려하여 알파벳의 Ascii 코드값 97값을 출력과 입력의 함수에서 사용하고 있다.

소스 코드

```
#include <fstream.h>

// 파일의 입력과 출력에 대한 선언
ifstream in("input.txt");
ofstream out("output.txt");

int containers[26];
int n, k;
int selected[10000][26], temp[26];    // 경우의 수를 10000개 이하로 설정함
int o;

void input()
{
        in >> n >> k;
        char s;
        for(int i=0;i<=n-1;i++)
        {
                in >> s;
                containers[(int)s-97]++; // 컨테이너 종류의 수를 센다.
        }
}

void back(int pre, int cnt)
{
        if(cnt == k)
        {
                for(int i=0;i<=cnt-1;i++)
                {
                        selected[o][i] = temp[i];
                }
                o++;
                return;
        }
```

```
        for(int i=pre;i<=25;i++)
        {
                if(containers[i] > 0)
                {
                        containers[i]--; // 컨테이너 종류에 따른 개수를 하나씩 감소
                        temp[cnt] = i;
                        back(i,cnt+1);
                        temp[cnt] = 0;
                        containers[i]++; //
                }
        }
}

void output()
{
        out << o << endl;
        for(int i=0;i<=o-1;i++)
        {
                for(int j=0;j<=k-1;j++)
                        out << char(selected[i][j]+97) << " ";
                out << endl;
        }
}
int main()
{
        input();
        back(0,0); // 초기값을 0으로 시작
        output();
        return 0;
}
```

2 공 가져가기 게임

▸▸ 게임에서 항상 이길 수 있는 방법이 있는지 없는지를 찾아내는 문제로 수학적인 수 체계를 이용하여 규칙을 찾아내어 문제를 해결하는 것으로 수학적인 기반의 필요성을 이해하고 활용하는 학습하고자 한다.

문제 (난이도 ★★☆☆☆)

두 명이 번갈아서 한 개 이상의 공을 가져가는 게임이 있다. 먼저 공을 가져가는 사람을 A라고 하고 A의 상대편을 B라고 할 때 이 게임의 규칙은 다음과 같다.

(규칙-1) 맨 처음 게임을 시작할 때 A는 모든 공을 한 번에 가져가는 것을 제외하고는 임의의 수만큼 공을 가져갈 수 있다.
(규칙-2) 처음에 A가 공을 가져간 다음부터는 한개 이상의 공을 가져가며 상대편이 바로 전에 가져간 공의 2배 이하로만 공을 가져갈 수 있다.
(규칙-3) 마지막 공을 가져가는 사람이 이긴다.

예를 들어, 4개의 공을 먼저 가져는 게임이라고 하자. A가 처음에 가져갈 수 있는 공의 개수는 1개 이상 3개 이하이다. 각각의 경우에 두 사람이 최선을 다한다면 다음과 같은 결과가 된다.

(1) A가 공을 1개 가져가면, B는 2개 이하의 공을 가져갈 수 있다. B가 공 1개를 가져가는 경우와 공 2개를 가져가는 경우 모두에 대해 A가 나머지를 수를 모두 가져 갈 수 있으므로 A가 이긴다.
(2) A가 공 두 개를 가져가면 B가 남은 두 개의 공을 가져가 B가 이긴다.
(3) A가 공 3개를 가져가면 B가 남은 공을 가져가 B가 이긴다.

위의 경우를 보면 공 4개를 먼저 가져가기의 경우에는 A가 공 1개를 가져가면 다음에 B가 공을 몇 개 가져가더라도 A가 이길 수 있는 방법이 있다는 것을 알 수 있다. 한편, 공 2

개를 먼저 가져가는 경우에는 A는 공을 1개 가져갈 수밖에 없으므로 B가 항상 이긴다. 문제는 먼저 공의 수가 주어졌을 때 A가 항상 이길 수 있는 방법이 있느냐하는 것이다.

프로그램의 이름은 “ballgame”로 한다.

입력 형식

입력 파일의 이름은 “Input.txt”로 한다.
첫 줄에 게임의 회수 N이 주어진다.
두 번째 줄부터는 먼저 부르기의 수 M이 주어진다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
수 N에 대한 처음에 부를 수의 개수를 출력한다. 항상 이길 수 없는 경우에는 -1을 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
3
4
10
13
```

출력의 예(Output.txt)

```
1
2
-1
```

2 문제의 배경 및 관련 이론

올림피아드 문제의 기본적인 유형으로 간단하면서 수학적인 지식의 기반을 질문하는 유형으로 많이 제시되고 있다. 무작위로 프로그램을 작성하게 되면 많은 시행착오를 가지게 되고 몇몇 데이터에 대해서는 결과 값의 오류를 발생하는 경우가 많다. 하지만 수학적인 개념과 활용을 통하여 문제를 이해하고 문제를 해결하게 되면 앞에서 말한 오류를 줄일 수 있으며 프로그램이 간단해진다. 이 처럼 이 문제를 통하여 기본적으로 제시되어지는 피보나치수열, 파스칼의 삼각형, 카탈란 수, 스털링 수 등의 관심을 가지고 학습하기를 바라며 이 문제를 제시하게 되었다. 이 문제의 해결방법은 첫째, 피보나치수열의 특성을 이용하는 방법이고 둘째는 동적 계획(dynamic programming) 방법을 이용하는 것이다. 두 가지 방법 모두가 수학적으로 공통적인 승패를 결정하는 규칙을 찾아내는 것은 마찬가지이다. 단지 프로그램의 소스 코딩에 있어 구현 방법이 조금 다르게 제시되어질 뿐이다.

첫째, 피보나치수열의 특성은 이미 잘 알려진 것과 같이 다음과 같은 특성을 가진다.

$$
\begin{aligned}
F(1) &= 1 \\
F(2) &= 1 \\
F(3) = F(1) + F(2) &= 2 \\
\\
F(n-1) &= F(n-2) + F(n-3) \\
F(n) &= F(n-1) + F(n-2)
\end{aligned}
$$

둘째, 동적 계획법의 경우는 점화식을 기본으로 구성되는 것으로 작은 범위에서 얻어지는 값을 이용하여 큰 범위에 값을 얻어가는 것으로 문제를 해결한다.

이 같은 유형의 문제는 다양하게 변형되어 제시되어질 수 있으면 이번 한국올림피아드 중등부 문제인 구슬게임의 경우도 이 문제의 변형으로 볼 수 있으므로 이 문제의 학습 후에 구슬게임의 문제를 해결해 보는 것도 학습에 도움이 될 것으로 보인다.

3 문제 해설

공을 가져가는 규칙에서 처음 공을 가져간 수가 N일 때 다음 사람이 가져갈 수 있는 공의 수는 2N개 이하이다. 단, 처음 사람은 N개를 처음부터 가져갈 수 없으며 주어지는 공은 2이상이다. 그러므로 먼저 공을 가져가는 사람은 남겨지는 공의 개수에 관심을 가져야 한다. 예를 들어 공의 수가 10개 주어질 때 처음 공을 가져가는 사람이 m개를 가져갈 때 남겨지는 공의 개수가 2m개보다 작게 남겨지면 처음 먼저 공을 가져가는 사람이 절대 이길 수 없다.
그러므로 처음 공을 가져가는 사람은 2m이하의 공이 남겨지지 않도록 공을 가져가면 된다는 것이 이 문제의 주된 포인트가 된다.

그럼 이 문제를 푸는 과정은 다음과 같이 설명될 수 있다. 여기서, 처음 공을 가져가는 사람을 A, 다음에 공을 가져가는 사람을 B라고 하고 처음 주어지는 공의 개수가 N이라고 가정한다.

1. 피보나치수열을 이용

N = 2 일 때, A가 공을 가져가는 방법은 단 한 가지로 1개를 가져가는 방법이외는 없으므로 A가 이길 수 있는 방법이 없다.

N = 3 일 때, A가 공을 가져가는 방법은 1개나 2개를 가져가는 방법이나 A가 공을 가져가고 남겨지는 공의 수가 2 또는 1이 되므로 A가 이길 수 있는 방법이 없다.

N = 4 일 때, 위에서 N = 3일 때 A가 이길 방법이 없다는 것은 B가 이길 수 있는 방법이 없다는 것이다. 그러므로 N이 4인 경우는 B가 공을 가져갈 때 공의 개수 N을 3으로 만들면 항상 A가 이길 수 있는 방법이 있다.

N = 5 일 때, 위의 N이 3일 때와 마찬가지로 1, 2, 3, 4를 가져가는 방법이 있다고 해도 항상 A가 이길 수 있는 방법이 없다.
∵ A가 1개의 공을 가져가면 4개의 공이 남게 되고 그럼 B가 1개의 공을 가져가는 경우 위의 N=3일 때와 같이 A가 이길 수 있는 방법이 없다.

N = 6 일 때, A가 공을 가져가는 방법은 1, 2, 3, 4, 5로 다섯 가지 방법이 있으나 위의 N이 5일 때 먼저 공을 가져가는 사람이 항상 이길 수 있는 방법이 없다는 것을 알 수 있으므로 A는 1개의 공을 가져가면 항상 이길 수 있는 방법이 있다.

N = 7 일 때, 위의 N이 6일 때와 마찬가지로 A가 공을 가져가고 남겨질 공의 수를 5개로 남기면 A강 항상 이길 수 있는 방법이 있으므로 A는 공 2개를 가져가면 항상

이길 수 있는 방법이 있다.

위의 N의 값 2, 3, 4, 5의 경우에서 2, 3, 5, 8 등의 값에서는 절대 A가 항상 이길 수 있는 방법이 없다는 공통점을 찾을 수 있다. 또한, A가 처음 가져가야 하는 공의 수의 결정은 N이 4인 경우 1, N이 6인 경우 1인 경우 항상 이길 수 있다는 것을 알 수 있다.

이것은 다시 설명하면 N이 4인 경우 피보나치수열을 구성하는 수들 중 4에 가장 가까운 수에 다른 한 개의 합이 4가 되는 순서쌍 (1, 3)값 중에서 최소값 1이 되는 것을 알 수 있다. 마찬가지로 N이 6인 경우는 합이 6이 되는 순서쌍은 (1, 5)인 경우로 최소값 1이 된다. 이제 큰 수를 가지고 이 규칙을 적용해 보면 다음과 같다. 예를 들어, 19가 주어지면 이것은 13+6이 된다. 여기서 6은 피보나치수열의 값이 아니므로 다시 6을 피보나치수열의 값의 합으로 나타내면 13+5+1의 형태를 취하게 된다. 그러므로 답은 1과 6이 된다. 단, 피보나치 수열의 값이 제 공통적인 규칙을 이해할 수 있다고 본다. 그러므로 코딩의 방법은 다양하게 제시할 수 있다.

아래 함수는 주어지는 공의 수(N)이 피보나치수열의 수인지를 판단하여 피보나치수열의 수인 경우는 0을 리턴하고 아닌 경우에는 피보나치수열의 수중에서 주어지는 공의 수(N)가 보다 작은 수를 리턴하는 함수이다.

```
int isFibo(void)
{
        int first = true;
        for(;idx>=0; idx--) {
                it(num>fib[idx]) {
                        num-=fib[idx];
                        first=false;
                }
                else if(num==fib[idx]) {
                        if(!first)  answer = fib[idx];
                        break;
                }
        }
}
```

2. 동적 계획법 이용

위의 피보나치수열 이용에 있어서와 마찬가지로 주어지는 공의 수 N에 대하여 처음 가져가는 사람이 항상 이길 수 있는 경우는 처음 공을 m개 가져가고 남겨지는 공의 수가 2m 개 이상일 때 이길 방법이 존재하므로 아래의 코딩은 처음 가져가야 할 공의 수를 m이 2인 경우부터 임의의 설정 최대 값까지에 대하여 처음 공을 가져가는 사람이 가져갈 공의 수를 결정하는 배열을 만드는 것이다.

```
for(i=2; i<=1000; i++)¨
{
        for(j=i-1; j>=i/2+1; j--)
        {
                if(dy[j]>(i-j)*2) {
                        dy[i]=i-j;
                        break;
                }
        }
         if(dy[i]==0) dy[i]=i;
}
```

소스 코드

[피보나치수열 이용]

```
#include <fstream.h>
int fib[50], idx, num[100], answer= -1;

void setP(void)
{
	int idx, i=0;
	fib[0]=fib[1]=1;
	for(idx=2;fib[idx-1] < 1000000; idx++)
	{
		fib[idx]=fib[idx-2]+fib[idx-1];
	}
	idx-=2;

	ifstream infile;
	infile.open("Input.txt");

	while(!infile.eof())
	{
		infile >> num[i];
		i++;
	}
	infile.close();
}

void solveP(int n)
{
	int first = true;
	answer= -1;

	// 100000을 넘는 값은 idx의 값이 30이하에 값을 가진다.
	for(idx=30;idx>=0; idx--) {
		if(n>fib[idx]) {
```

```
                        n-=fib[idx];
                        first=false;
                }
                else if(n==fib[idx])
                {
                        if(!first)
                                answer = fib[idx];
                        break;
                }
        }
 }

int main()
{
        int i;
        setP();

        ofstream outfile;
        outfile.open("Output.txt");

        for(i=1;i<=num[0];i++)
        {
                solveP(num[i]);
                outfile << answer << endl;
        }
        outfile.close();
        return 0;
}
```

[동적 계획법]

```
#include <stdio.h>

FILE *inf=fopen("INPUT.TXT", "r");
FILE *outf=fopen("OUTPUT.TXT", "w");
int dy[1001], n;

void make()
{
	int i, j;

	// 주어지는 공의 수가 1000이하라 가정
	for(i=2; i<=1000; i++)
	{
		for(j=i-1; j>=i/2+1; j--)
		{
			// i-j만큼 가져갔을 때, 상대는 j부터, i-j의 2배만큼 가져간다.
			if(dy[j]>(i-j)*2)
			{
				dy[i]=i-j;
				  // j에서 이기기 위해서는 최소 dy[j]만큼 가져가야 한다.
				break;
				  // 이때, dy[j]가 (i-j)*2 보다 크면 상대는 이길 수 없다.
			}
		}
		if(dy[i]==0) dy[i]=i;
	}
}

int main()
{
	int i, su;

	make(); // 임의의 수에 대해 공을 몇 개 가져가야 이길 수 있는 가를 dy에 저장
	fscanf(inf, "%d", &n);
```

```
	for(i=1; i<=n; i++)
	{
		fscanf(inf, "%d", &su);
		if(dy[su]==su)
		{
			fprintf(outf, "No\n");
		}
		else
		{
			fprintf(outf, "%d\n", dy[su]);
		}
	}
	return 0;
}
```

3 스파이 보내기

▸▸ 수의 규칙성과 점화식으로 구성되는 수의 값을 계산하는 문제로 동적 프로그램을 통하여 해결이 가능하나 수의 크기가 커지는 것을 예상하여 큰 수에 대한 연산의 방법을 이해하고 활용하는 방법적인 모델이다.

문제 (난이도 ★★★☆☆)

강을 경계로 비토와 이를 스턴 마을이 있었다. 두 마을 사이에는 아무런 접촉도 없이 서로 고립되어 생활을 하고 있었다. 그러던 중 스턴 마을에서는 스파이를 비토 마을에 보내기로 결정하였다. 그러나 강을 건너 비토 마을로 들어갈 수 있는 배가 한척 밖에 없다. 또한 이 배는 특이한 점을 가지고 있었다. 배는 항상 S명이 탑승해야 하며 비토 마을에 도착하면 반드시 한 명만이 비토 마을에 남고 나머지는 다시 스턴 마을로 돌아와 S명을 태우고 다시 비토 마을로 갈 수 있다. 다시 말하면 이 배는 다음과 같은 방법으로 운행된다.

처음 스턴 마을을 떠날 때 스파이 1, 2, …, S명을 싣고 간다. 그 중 한명이 비토 마을에 내리고 나머지 S-1명이 스턴 마을로 되돌아온다.
두 번째 스턴 마을을 떠날 때 스파이 S+1, S+2, …, 2S와 되돌아 온 S-1명의 스파이를 싣고 비토 마을로 향한다. 그들 중 한 명이 비토 마을에 내리고 나머지 2S-2명의 스파이 턴 마을로 되돌아온다.
세 번째는 스파이 2S+1, 2S+2, …, 3S와 되돌아온 2S-2명의 스파이가 비토 마을로 향한다 그들 중 한 명이 비토 마을에 내린다.

이와 같이 계속한다. N번 비토 마을에 다녀온 후, 파견된 스파이 N명을 구성하는 방법의 수를 모두 구하는 프로그램을 작성하시오.

프로그램의 이름은 “spy”로 한다.

입력 형식

입력파일의 이름은 “Input.txt”로 한다.
한 줄에 한 번에 배에 탈 수 있는 사람의 수 S와 배가 이동하는 횟수 N이 주어진다.

출력 형식

출력파일의 이름은 “Output.txt”로 한다.
한 줄에 스파이 구성원을 구성하는 방법의 수를 출력한다.

제약 조건

$1 \leq S \leq 10$
$1 \leq N \leq 40$
배에 탑승할 수 있는 사람의 수는 무한하다.
비토 마을에 도착하는 스파이들의 구성원 순서는 무관하다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
2 3
```

출력의 예(Output.txt)

```
14
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
5 20
```

출력의 예(Output.txt)

```
75134301594081389660
```

2 문제의 배경 및 관련 이론

동적 계획법을 활용한 문제의 해결 유형으로 동적 계획법을 학습하기 위한 문제이다. 동적 계획법에 대한 점화식의 추출과 테이블을 구성하여 값을 얻어가는 과정을 단계적으로 이해하여 다른 문제에도 적절하게 활용할 수 있도록 학습하면 좋을 것이다. 동적 계획법에 대한 설명을 간단히 하면, 반복해서 여러 번 호출되는 부분문제가 있을 경우 해당되는 부분문제가 첫 번째 수행될 때 그 결과를 기록해 두었다가 이후에 다시 수행이 요구될 때 또 수행하는 것이 아니라 이전에 기록된 결과를 이용하는 것이 동적 계획법의 핵심이 되는 부분이다. 주어진 문보다 작은 모든 경우를 다 해결한 뒤에 주어진 문제를 해결하는 방식이다. 이때, 주어진 문제를 분할된 모든 경우에 대한 해(Solution)가 테이블 형태로 저장된다. 그리고 가장 작은 문제부터 시작해 해를 점점 키워 최종적으로 주어진 문제에 도달하게 되는 것이다. 그러나 동적 계획법을 이용할 때는 첫째는 작은 문제의 해를 병합하여 더 큰 문제의 해를 구하는 것이 항상 가능하지는 않다. 둘째, 주어진 문제를 해결하기 위해 작은 문제로 나눌 때, 해결해야 하는 작은 문제가 지나치게 많아질 수가 있다는 것에 주의해야 한다.

또한 이 문제는 연산되어지는 수의 크기가 컴파일러가 사용하는 자료형의 크기 이상으로 커지는 연산이 이루어지므로 큰 수를 다루는 방법에 대한 기본적인 학습을 함께 제시하였다. 그러므로 예외적인 값에 대한 감각을 길러줄 수 있는 문제라고 생각한다.

큰 수를 다루기 위해서는 배열이나 연결 리스트를 이용하여 구현할 수 있다. 여기서는 배열을 이용하여 연산하는 방법이다. 채워진 두 개의 배열의 합은 각 자리 수 별로 더하여 만들어지는 올림수를 다음 자리에서 더하여 계산하는 방식이다. 아래 그림은 배열 A, B의 값을 더한 결과이다.

A :	1	2	3	4	5	6	7	8	9	0	0	0	1	1	2
B :	0	0	0	0	0	0	0	0	0	0	0	0	9	1	2
A + B :	0	0	0	0	0	0	0	0	0	0	0	1	0	2	4

[배열을 이용한 큰 수 연산]

이런 유형의 문제들은 간단한 문제이나 수의 규칙성을 통한 점화식의 추출이 간단하지 않는 경우가 많아 문제 해결을 하지 못한다. 그러므로 이 문제를 통하여 점화식의 추출의 위하여 필요한 경우의 수를 계산하는 수학적 기본 지식을 간과하지 말아야 할 것이다.

3 문제 해설

이번 문제는 위에서 설명한 동적 계획법에 아주 적절하게 맞는 유형으로 계산되어지는 부분 문제의 간단한 점화식 추출하여 반복과 더불어 값을 얻어 가면 된다. 단, 이번 문제에서 중요한 점은 계산된 결과 값이 정수형이나 실수형으로 계산되어지는 그런 숫자가 아니기 때문에 큰 수에 대한 연산의 방법도 생각해야 하는 것이다.

이 문제의 주된 해결 방법은 사람의 수 m과 반복되어질 회수 n의 값에 대한 테이블로 생각할 수 있다. 예를 들어, m=2, N=3인 경우에 대하여 생각하여 보자.

N=1	N=2	N=3
1	2(1, 2)	3(1, 2, 3)
		4(1, 2, 4)
		5(1, 2, 5)
		6(1, 2, 6)
	3(1, 3)	~~2(1, 3, 2)~~
		4(1, 3, 4)
		5(1, 3, 5)
		6(1, 3, 6)
	4(1, 4)	~~2(1, 4, 2)~~
		~~3(1, 4, 3)~~
		5(1, 4, 5)
		6(1, 4, 6)
2	~~1(2, 1)~~	
	3(2, 3)	~~1(2, 3, 1)~~
		4(2, 3, 4)
		5(2, 3, 5)
		6(2, 3, 6)
	4(2, 4)	~~1(2, 4, 1)~~
		~~3(2, 4, 3)~~
		5(2, 4, 5)
		6(2, 4, 6)
2	5	14

위에 제시된 표에서 몇 가지의 규칙을 알 수 있다.

첫째는 남겨지는 순서는 의미가 없으므로 1, 2나 2, 1로 남겨지는 경우는 같은 경우로 보고 대각선이 그어져 있다.

둘째는 아래와 같이 각 경우만을 제외하고 N의 값보다 작은 경우의 수에 대하여 모두 대각선이 그어져 있다.

1) N=1인 경우 1 또는 2
2) N=2인 경우 1일 때 2, 3, 4
   2일 때 3, 4
3) N=3인 경우
   1이고 2일 때 3, 4, 5, 6
   1이고 3일 때 4, 5, 6
   1이고 4일 때 5, 6
   2이고 3일 때 4, 5, 6
   2이고 4일 때 5, 6

위와 같이 몇 가지의 경우를 가지고 표를 채워 본 결과 다음과 같은 점화식을 추출할 수 있다.

$$D_{0,0} = 1$$
$$D_{1,0} = 1$$
$$D_{i,j} = 0 (i < j)$$
$$D_{i,j} = D_{i-1,j-1} + D_{i-1,j} (i \le j)$$

이것을 테이블로 표현하여 보면, 아래 표와 같다. 아래의 표와 같이 채울 수 있도록 점화식을 코딩하면 된다.

	0	1	2	3
0	1	0	0	0
1	1	1	0	0
2	0	2	1	0
3	0	2	3	1
4	0	2	5	4
5	0	2	5	9
6	0	2	5	14

소스 코드에서는 위의 표를 2차원 배열이 아닌 3차원 배열을 이용하여 큰 수 연산까지 가능하도록 코딩하였다. 즉, d[i][j][k]의 배열을 선언하여 첨자 i 는 n의 값, j 는 위 표의 행 값, k는 $D_{i,j}$가 가지는 값을 100자리로 표시하기 위해서 사용하였다.

첨자 k에 따른 배열의 값은 각 자리를 $D_{i,j}$값을 10으로 나누어 몫은 배열의 앞에 나머지는 뒤로 반복해서 연산하여 넣는다.

```
//  D_{i,j} = D_{i-1,j-1} + D_{i,j}의 값 결정
for(i=1;i<=100;i++)
{
        d[x][y][i]+=d[x1][y1][i]+d[x2][y2][i];
}

for(i=1;i<100;i++)
{
        d[x][y][i+1]+=d[x][y][i]/10;
        d[x][y][i]%=10;
}
```

소스 코드

```
#include <fstream.h>
int n, m;
int d[41][401][101];

void INPUT()
{
	ifstream in("INPUT.TXT");
	in>>m>>n;
	in.close();
}

void f_plus(int x1, int y1, int x2, int y2, int x, int y)
{
	int i;
	// 경우의 수 값을 채우는 식으로 D_{i,j} = D_{i-1,j-1} + D원1,식
	for(i=1;i<=100;i++) d[x][y][i]+=d[x1][y1][i]+d[x2][y2][i];
	for(i=1;i<100;i++)
	{	// 큰 수의 연산을 위하여 자리수별로 배열에 저장
		d[x][y][i+1]+=d[x][y][i]/10;
		d[x][y][i]%=10;
	}
}

void OUTPUT()
{
	int i;
	ofstream out("OUTPUT.TXT");
	for(i=100;i>=1;i--) {
		if(d[n][m*n][i]!=0) break;
	}
	for(;i>=1;i--) {
		out<<d[n][m*n][i];
	}
	out.close();
}
```

```
void f1()
{
	int i, j, k;
	for(i=0;i<m;i++) d[0][i][1]=1;
	for(i=1;i<=n;i++)
	{
		for(j=i;j<=m*i;j++)
		{
			f_plus(i-1, j-1, i, j-1, i, j);
			for(k=100;k>=1;k--)
			{
				if(d[i][j][k]!=0) break;
			}
		}
		for(j=m*i+1;j<=m*n;j++)
		{
			for(k=1;k<=100;k++) d[i][j][k]=d[i][m*i][k];
		}
	}
}

int main()
{
	INPUT();
	f1();
	OUTPUT();
	return 0;
}
```

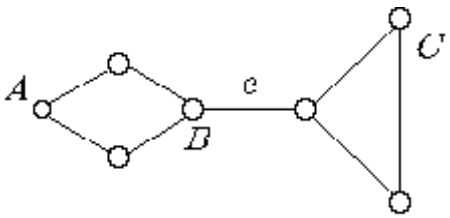
4 절점(Cut Vertex) 찾기

▸▸ 네트워크에 많이 나타나는 전송 경로에 관한 문제로 그래프 이론에 연결하여 그래프상의 문제로 바꾸는 방법과 그래프내의 절점에 관한 개념과 특성을 이해하여 해결하는 방법을 학습하게 된다.

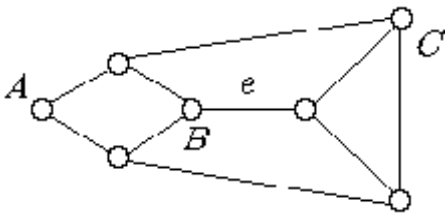
문제 (난이도 ★★★★☆)

컴퓨터망을 그래프 모델로 나타낼 수 있다. 즉, 컴퓨터는 그래프의 정점으로 컴퓨터간의 통신선은 에지로 나타낸다. 컴퓨터끼리 통신을 하기 위해서는 메시지가 통신선과 컴퓨터 등을 통하여 원하는 컴퓨터로 가야 한다. 그러나 통신선과 컴퓨터는 고장이 날 수 있으므로 신뢰성이 높은 컴퓨터 망을 구축한다. 다음 그림 (a)에서 컴퓨터 B가 고장이 나게 되면 컴퓨터 A와 C 는 서로 통신이 불가능하게 되지만 (b)에서는 B가 고장 나더라도 컴퓨터 A와 C는 다른 경로를 통해 통신이 가능하므로 (b) 컴퓨터 망이 더 신뢰도가 높다고 할 수 있다. 따라서 우리는 주어진 컴퓨터 통신망의 신뢰도를 떨어뜨리는 컴퓨터나 통신선을 효율적으로 찾는 것이 중요하다.

주어진 그래프에서 그림 (a)의 B와 같은 컴퓨터가 고장이 나므로 서로 통신이 불가능한 컴퓨터가 생기도록 하는 컴퓨터(절점 : Cut Vertex)를 모두 구하는 효율적인 알고리즘을 작성하라. 이 문제는 통신선의 신뢰도를 100%로 가정한다.



(a)



(b)

프로그램의 이름은 “cutvertex”로 한다.

입력 형식

입력파일의 이름은 “Input.txt”로 한다.
첫 번째 줄에는 컴퓨터 네트워크에 속한 컴퓨터의 통신 링크의 수 m이 주어진다. 여기서 m은 2,000,000이하인 양의 정수이다.
둘 째 줄부터 m개의 줄에 한 줄에 하나씩 통신 링크가 주어진다. 통신 링크는 그것이 잇는 컴퓨터의 쌍이다. 입력되는 컴퓨터 네트워크는 분리되어 있지 않다. 즉 그것을 표현하는 그래프가 연결되어 있다. 한 쌍의 컴퓨터를 잇는 통신 링크는 세 개 이상 입력으로 들어오지 않는다.

출력 형식

출력파일의 이름은 “Output.txt”로 한다.
첫째 줄에 절점 노드의 번호를 출력한다.
절점이 존재하지 않는 경우는 -1을 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

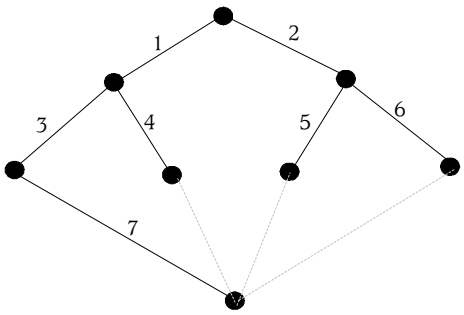
```
10
1 2
2 8
8 9
9 1
2 3
3 4
4 7
3 7
4 5
7 6
```

출력의 예(Output.txt)

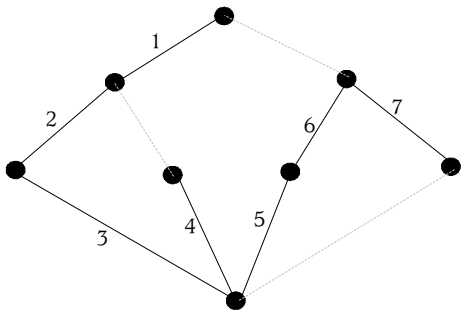
```
2 3 4
```

2 문제의 배경 및 관련 이론

컴퓨터과학 분야에서 가장 많이 사용되는 수학적 모델 중의 하나가 그래프이다. 18세기 오일러의 "쾨니스버그 다리 문제"에 처음 사용하여 문제를 해결하였으며 이에 의해 정립된 그래프의 이론은 운영제체의 프로세서 제어, 오토마타 이론, 각종 탐색 문제 등에 응용되고 있다. 그런 이유로 그래프에 관련된 이론을 익히고 학습하는 것은 매우 중요한 문제가 된다.
그래프 이론의 기본은 $G=(V,E)$그래프를 Vertex와 edge의 집합으로 표현하는 방법에서부터 관련된 이분 그래프, 부 그래프, 무향, 유향 그래프의 종류 및 차수, 사이클, 경로, 연결등에 대한 용어의 이해이다. 이번 문제는 그래프의 탐색법을 이용하여 해결하는 것으로 너비우선탐색(BFS)나 깊이우선탐색(DFS)의 방법에 대한 이해가 필요하다. 너비우선탐색(BFS)는 시작 정점으로부터 거리가 증가하는 순서로 방문하며, 두 노드간의 경로 중 가장 짧은 경로의 길이를 먼저 방문하는 것으로 큐를 이용한다. 반면에 깊이우선탐색(DFS)은 시작 정점에서 방문되지 않은 인접 노드가 있으면 인접 탐색한다.



너비우선탐색(BFS)



깊이우선탐색(DFS)

또한 이 문제에서 요구하는 절점(Cut Vertex)이란 연결 그래프 $G=(V,E)$의 정점으로서 그 정점과 연결된 에지들을 제거하였을 때, 나머지 부 그래프가 두 개 이상의 연결 성분으로 구성될 경우를 말한다. 다시 말하면 절점 및 이중 연결 성분의 특징은 다음과 같다.

- 두 이중연결성분들의 교집합은 최대 하나의 정점(=절점)
- 두 이중연결성분들의 에지(edge) 교집합은 공집합

위의 깊이우선탐색(DFS)와 절점 및 연결성분의 특징을 이용하여 문제를 해결하는 과정에서 그래프에 대한 기본적인 용어와 함께 각 그래프 종류별 특징을 학습하는 기회를 가지기 바란다. 이런 유형의 문제는 정보올림피아드 문제의 자주 등장하며 앞으로 문제해결을 위한 사고에 주된 기반이 될 것이다.

3 문제 해설

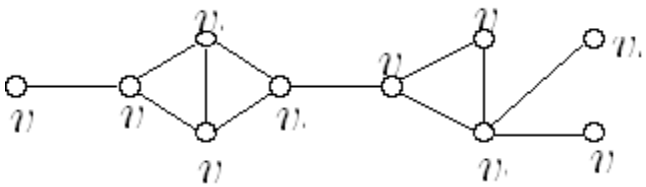
이 문제의 해결 방법은 깊이 우선 탐색 방법으로 노드를 방문하면서 노드간의 에지를 백 에지(Back edge)와 트리 에지(Tree edge)로 나누어 두 종류의 번호를 매기고 두 종류의 번호를 가지고 각 노드가 절점이 되는지를 결정하면 된다. 문제를 해결하기 위한 순서는 아래와 같다.

1) 깊이우선 탐색 트리를 형성
2) 정점 v를 방문하고 정점 w를 방문 - (v, w)는 트리 에지(tree edge)
3) v에 인접한 u가 이미 방문된 노드 - (v, u) 는 백 에지(back edge) - u가 v의 선조
4) 탐색 시 노드마다 두 종류의 번호를 매김
- $dfn[v]$: 깊이우선 탐색에 의해 방문된 순서(0 부터)
- $back[v]$: v에서 시작하여, 트리 에지와 하나의 백 에지를 통해 도달 가능한 루트 에 가장 가까운 정점의 dfn

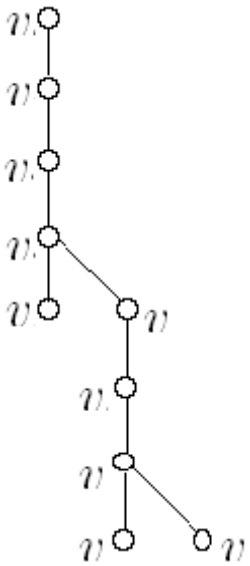
① 처음으로 정점 v를 만나는 경우 : $back[v] = dfn[v]$
② 백 에지 (v, w)를 만나는 경우 : $back[v] = \min(back[v], dfn[w])$
③ w로부터 v로 거슬러 올라가는 경우 : $back[v] = \min(back[v], back[w])$

참고로 v가 절점이기 위한 필요충분조건은 첫째, 깊이우선탐색 트리에서 두 개 이상의 자식을 갖는 루트이다.(∵ 루트가 없으면 한쪽 부 트리에서 다른 부 트리로 갈 수 없으므로) 둘째는 $dfn[v] \le back[w]$인 자식노드 w를 갖는 경우이다.(∵ v로부터 w의 선조로 갈 수 없으므로)

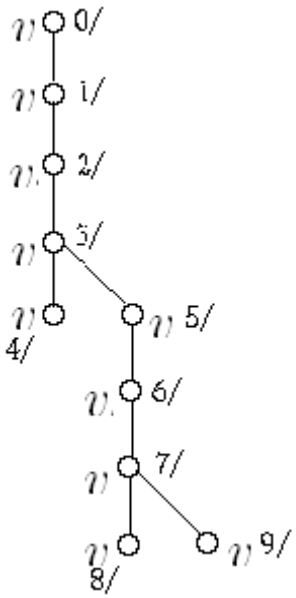
예를 들어 다음과 같은 그래프가 주어지고 임의의 정점 v_0를 시작으로 깊이우선탐색을 시작한다. 실선은 트리 에지가 되고 점선은 백 에지가 된다.



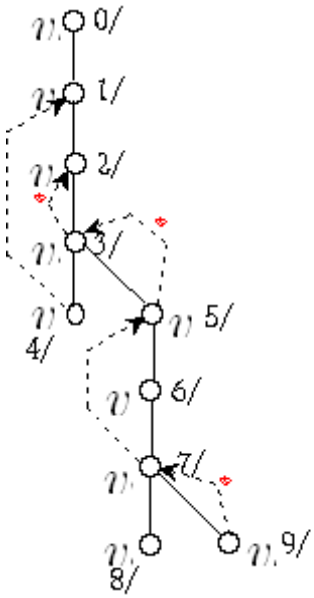
위의 그래프에서 실선의 깊이우선탐색의 순서대로 트리를 구성하면 아래 그림과 같이 만들어진다.



그럼 우선, 그려진 트리에 차례대로 번호를 매긴다. 번호는 0번부터 매겨간다. 이것이 결과적으로 앞서 말한 트리 에지 의 값이며 깊이우선탐색(dfs)에 의한 탐색의 순서이다.



이제 백 에지에 대한 값을 결정해야 하는데 이 값을 결정하기 위해서 먼저 현재 노드에서 바로 연결되어질 수 있는 노드로의 백 에지를 연결한다. 연결한 결과는 다음과 같다.

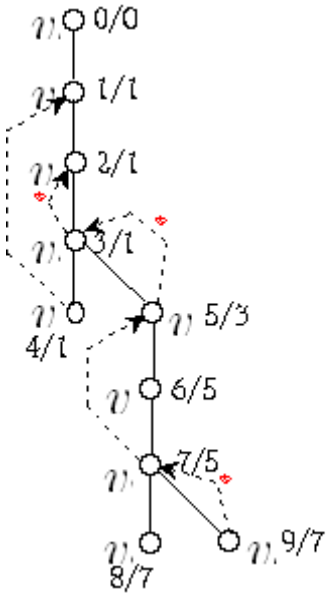


백 에지에 대한 값은 다음과 같은 규칙을 값을 설정한다. 깊이가 가장 깊은 노드부터 시작하여 백 에지의 값을 설정하게 되는데

우선 8, 9번 노드는 한 번에 도달 가능한 노드가 7번 노드이므로 백 에지 값은 모두 7이 된다. 7번 노드의 경우도 깊이가 낮은 곳으로는 5, 6번 노드에 도달 가능하므로 백 에지 값이 5와 6중에서 작은 값인 5와 8, 9번 노드에 모두 한 번에 도달 가능한 백 에지 값인 7중에서 작은 값으로 설정된다.

이와 같은 방법으로 차례대로 백 에지 값을 설정하면 되지만 root의 경우는 처음 방문되어지는 노드이므로 자신의 트리 에지 값을 그대로 백 에지 값으로 설정된다. 이 방법으로 설정된 트리에지와 백 에지의 값은 다음 그림과 같다.

트리 에지와 백 에지의 값이 결정되면 이제 root의 경우와 root 노드가 아닌 경우로 나누어 절점의 조건을 만족하는 노드를 결정하면 된다.



root 노드의 경우는 자식 노드가 2개인 경우에 절점의 조건이 된다. 그러므로 0번 노드는 절점이 아니다.

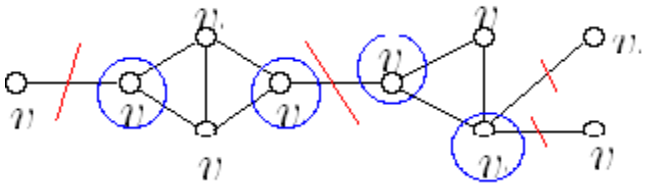
1번 노드의 경우는 트리 에지와 백 에지의 값이 같으므로 절점이 된다.

2번 노드의 경우는 3번 노드가 자식 노드이면 2번 노드의 트리 에지 값이 3번 노드의 백 에지 값보다 크므로 절점 조건을 만족하지 못한다.

3번 노드의 경우는 4번 노드가 자식 노드이면서 3번 노드의 트리 에지 값이 4번 노드의 백 에지 값과 작거나 같으므로 절점의 조건을 만족한다.

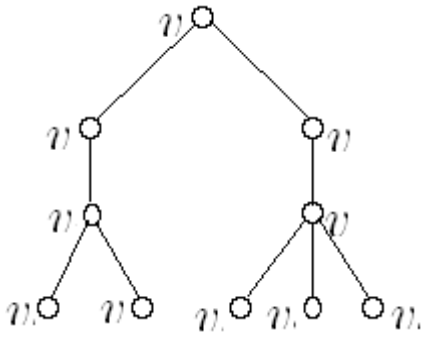
4번과 6번은 2번 노드가 절점의 조건을 만족하지 못하는 경우와 같다.
5번과 7번 모두 3번 노드의 절점 조건을 만족하므로 절점이 된다.

결과적으로 위의 예에서 절점은 v_1, v_7, v_5, v_3가 된다. 아래 그림과 같이 보면 절점과 같이 그래프를 절단하는 에지도 찾아 볼 수 있다.



위의 예에서는 간단히 시작점을 0번 노드에서 시작한 결과 깊이우선탐색트리의 깊이(depth)가 8이다. 깊이(depth)가 깊어 탐색 길다. 그러므로 여러분은 트리의 깊이(depth)가 적게 구성하는 것이 중요하다.
어떤 노드에서 시작하느냐에 따라 깊이(depth)가 달라지게 만들어질 수 있으므로 시작점의 선택에 있어서도 하나의 전략을 설정하는 방법을 가지는 것이 좋다. 위 예와 같은 그래프를 가지고 깊이(depth)를 4로 줄여 트리를 구성할 수도 있다. 깊이(depth)를 4로 구성한 트리의 예이다.

3번 노드를 시작점으로 깊이우선탐색을 한 경우이다.



기본적인 해결 알고리즘은 다음과 같다.

```
void biComponent(int v, int u) {
        nodePointer      ptr;
        int w, x, y;
        dfn[n] = back[v] = num++;
        for(ptr=graph[v];ptr;ptr=ptr->link) {
                w = ptr->vertex;
                if(u != w && dfn[w]<dfn[v]) push(&top, w, v);
                if(dfn[w]<0) {
                        bicomponent(w, v);
                        if(back[w] >= dfn[v]) {
                                printf("New Bicomponent :\n");
                                do {
                                        pop(&top, &x, &y);
                                        printf("< %d, %d >", x, y);
                                } while(!((x==v) && (y==w)));
                                printf("\n");
                        }
                }
                else if(w != u) back[v]=MIN(back[v],back[w]);
        }
}
```

소스 코드

```
#include <stdio.h>
#include <memory.h>
#define inpfilename "input.txt"
#define outfilename "output.txt"
#define N (100*2+2)

struct NODE
{
	int v;
	NODE *next;
};

int n;
NODE *matrix[N];
int check[N];
int isap[N];
FILE *fp=fopen(outfilename,"wt");
int rs=0;

void input()
{
	int i;
	int u,v;
	FILE *fp=fopen(inpfilename,"rt");
	fscanf(fp,"%d",&n);
	// 노드 생성하기
	for(i=0;i<n;i++)
	{
		matrix[i]=new NODE;
		matrix[i]->v=-1;
		matrix[i]->next=NULL;
	}
```

```
        for(i=0;i<n-1;i++)
        {
                fscanf(fp,"%d %d",&u,&v);
                u--;
                v--;
                // 그래프를 연결리스트로 구현
                NODE *temp=new NODE;
                temp->next=matrix[u]->next;
                temp->v=v;
                matrix[u]->next=temp;

                temp=new NODE;
                temp->next=matrix[v]->next;
                temp->v=u;
                matrix[v]->next=temp;
        }
}

int count=0;
int ap(int v)
{ // 깊이우선탐색하면서 트리에지와 백 에지의 값을 설정한다.
        NODE *now;
        int min,m;

        check[v]=min=++count;
        for(now=matrix[v]->next;now;now=now->next)
        {
                if(v==0 && !check[now->v]){
                        rs++;
                }

                if(!check[now->v]){
                        m=ap(now->v);
                        if(m<min){
                                min=m;
                        }
```

```
                        if(m>=check[v] && v!=0){
                                isap[v]=1;
                        }
                }
                else{
                        if(check[now->v] < min){
                                min=check[now->v];
                        }
                }
        }
        return min;
}

void process()
{
        int i;
        ap(0);

        if(rs>1){
                isap[0]=1;
        }

        FILE *fp=fopen(outfilename,"wt");
        for(i=0;i<n;i++){
                if(isap[i]){
                        fprintf(fp,"%d ",i+1);
                }
        }
        fprintf(fp,"\n");
}

void main()
{
        input();
        process();
}
```

5 숫자 퍼즐 채우기

▸▸ 마방진과 같이 퍼즐에 수를 넣는 방법으로 다양한 결과의 해 중에서 하나를 선택하는 문제로 Ford-Fulkerson의 Network Flov 방법을 이해하고 학습한다.

문제 (난이도 ★★★★★)

가로의 크기와 세로의 크기가 각각 N인 숫자판이 있다. 숫자판의 각 칸에는 음 아닌 정수들만 들어갈 수 있고 각 행과 각 열의 합이 미리 주어진다고 한다. N=2 인 경우의 예가 다음에 있다.

		12
		4
6	10	

위 그림에서 숫자판 옆의 수는 해당하는 행에 들어가는 숫자의 합을 나타내며, 숫자판 아래의 수는 해당하는 열에 들어가는 숫자의 합을 나타낸다. 이제, 숫자판에 주어진 합과 일치하도록 수를 넣으려고 한다. 합이 일치되도록 숫자를 넣는 방법은 여러 가지 있을 수 있으며, 위의 예에 대해 서로 다른 형태를 3가지만 보이면 다음과 같다.

5	7
1	3

6	6
0	4

4	8
2	2

이 문제에서는 가능한 여러 가지 형태 중 숫자판에 들어가는 최대 숫자의 값을 최소로 하는 형태를 찾고자 한다. 그러므로 위의 예에서는 최대 숫자가 6인 형태가 원하는 답이다. 이 문제를 해결하는 프로그램을 작성하시오. 수행시간은 PC에서 1초 이내이어야 한다.

프로그램 이름은 “digitpuzzle”로 한다.

입력 형식

입력 파일의 이름은 “Input.txt”이다.
첫째 줄은 행(열)의 크기 N이 주어진다.($1 \leq N \leq 50$).
둘째 줄에는 N개의 정수가 주어진다. 주어지는 정수는 1행부터 N행까지의 합을 차례대로 나타낸다.
셋째 줄에는 N개의 정수가 주어진다. 주어지는 정수는 1열부터 N열까지의 합을 차례대로 나타낸다. 합을 나타내는 각 정수는 0이상 10000이하이다. 숫자판을 구성할 수 없는 입력은 주어지지 않는다고 가정한다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
첫줄에는 배정된 수들 중 최대값을 출력한다.
둘째 줄부터 (N+1)째 줄까지는 각 행에 배정된 수들을 한 줄에 한 행씩 출력한다. 배정되는 각각의 정수는 0이상이어야 한다.

입력과 출력의 예 1

입력의 예(Input.txt)

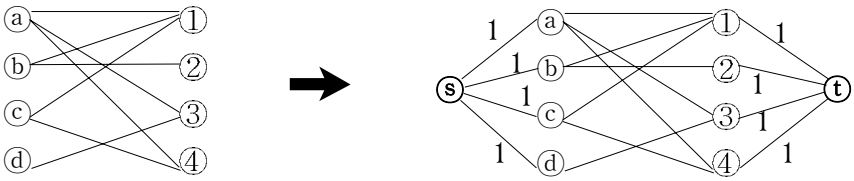
```
2
12 4
6 10
```

출력의 예(Output.txt)

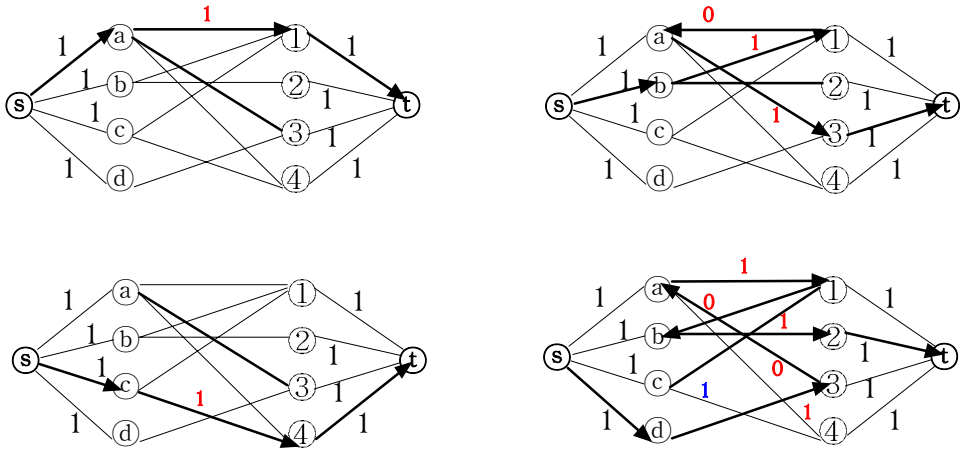
```
6
6 6
0 4
```

2 문제의 배경 및 관련 이론

송유관이나 수로에서의 유체의 흐름 통신망이나 도로망 혹은 공장의 조립라인 등의 모델링 문제의 해결을 위해 도입된 개념으로 Ford-Fulkerson의 Network Flow 알고리즘이라고 부른다. 여기서 네트워크는 어떤 물질이 입구에서부터 출구까지 제한된 용량을 가진 선을 통과(network)하여 도착하는 시스템을 표현하기 위해 고안된 그래프 형태이다. 네트워크는 시작점과 도착점을 가지고 있고 네트워크상에는 흐름이 정의되어 있는데 시작점(source), 도착점(target), 흐름(flow), 흐름은 네트워크상을 통과하는 물질의 양을 시작점은 흐름이 시작하는 입구를 도착점은 흐름의 도착지를 의미 한다. 그럼 이제 간단하게 이분 그래프의 최대 매칭 문제를 예를 들어 Network flow의 개념을 설명하면 다음과 같다.
첫째, 이분그래프를 아래 그림과 같이 Network flow 형태로 시작점과 도착점을 추가하여 에지별 가중치가 1인 그래프로 변경한다.



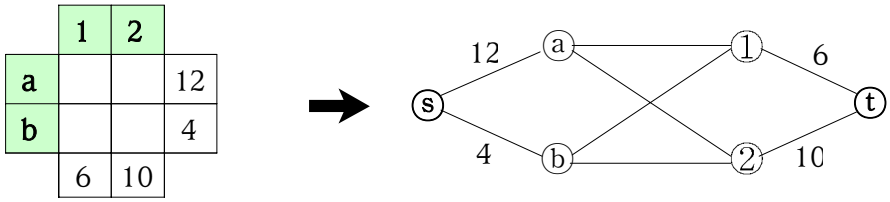
둘째, 시작점 ⓢ에서 도착점 ⓣ로의 경로를 하나씩 선택해 가면서 각 에지의 가중치 값을 모두 1로 정한다. 그리고 반대 방향으로 움직이면 에지의 가중치 값을 -1 해가며 가능한 모든 경로에 대하여 에지의 값을 결정한다. 더 이상의 경로가 없으면 마지막에 만들어진 에지의 값이 1인 경로만을 남기고 나머지 경로는 삭제한다.



그러므로 이 매칭의 결과는 ⓐ와 ①, ⓑ와 ②, ⓒ와 ④, ⓓ와 ③의 결과를 얻는다.
이 Network flow를 통하여 다양한 문제를 해결할 수 있으며, 이 문제 해결을 통하여 최대화 혹은 최소화하는 문제에 대한 해결의 아이디를 다른 방향에서 학습할 수 있는 기회가 되었으면 한다.

3 문제 해설

문제 해결의 주된 아이디어는 위에서 설명한 바와 같이 network flow 방법으로 해결된다. 우선 채워질 격자의 가로와 세로에 대한 값의 정점을 부여하고 그래프로 표현 한다. 예를 들면, 2 by 2의 격자라면, 아래와 같이 행을 a, b로 열을 1, 2의 형태로 정점을 설정하고 그래프로 표현하면 다음과 같다.



표현된 그래프를 프로그램 구현을 위하여 인접행렬로 구성하면 다음과 같은 테이블이 얻어진다. 각 인접 행렬의 값은 이 그래프의 에지의 가중치를 넣는다. 가중치가 주어지지 않는 에지는 모두 무한한 값으로 설정하게 된다. 에지가 없는 구간은 가중치 값을 모두 0으로 한다.

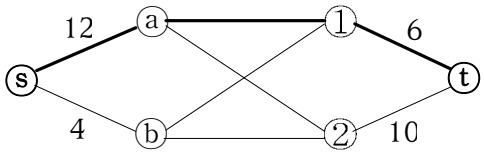
C	0	1	2	3	4	5
0		12	4			
1				9999	9999	
2				9999	9999	
3						6
4						10

이 capacity를 표현하는 테이블과 같은 크기의 잔여량(R)의 계산과 에지에 흐르는 값(F)을 계산하는 두 개의 테이블을 구성하면 다음과 같다.

R	0	1	2	3	4	5
0		12	4			
1				9999	9999	
2				9999	9999	
3						6
4						10

F	0	1	2	3	4	5
0						
1						
2						
3						
4						

capacity를 나타내는 테이블 C를 이용하여 시작점(s)에서 도착점(t)에 이르는 경로를 하나 찾는다. 여기서는 너비우선탐색(bfs) 보다 깊이우선탐색(dfs)이 경로를 찾는데 유리하다. 이유는 깊이가 깊어야 3을 넘지 않는다.



C	0	1	2	3	4	5
0	0	12	4			
1		0		9999	9999	
2				9999	9999	
3						6
4						10

찾아진 경로(s-a-1-t)의 에지의 값들 중에서 최소값을 찾는다.
(∴ 최소값은 6이 된다.)

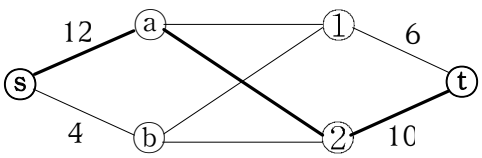
F	0	1	2	3	4	5
0		6				
1		-6		6		
2						
3						6
4						

위 단계에서 찾아진 값 6이 에지의 흐름의 양을 나타내므로 테이블 F에 값을 계산한다.
F[i][j] = F[i][j] + min
F[j][i] = -f[i][j]

R	0	1	2	3	4	5
0		6	4			
1		6		9993	9999	
2				9999	9999	
3						0
4						10

테이블 F의 값이 채워지고 나면 이제 잔여량을 표시하는 테이블 R에 값을 계산한다.
R[i][j] = C[i][j] - F[i][j]
R[j][i] = C[j][i] - F[j][i]

같은 방법으로 다음 경로를 찾는다. 예를 들어 다음 경로가 s-a-2-t 라면 테이블은 다음과 같이 만들어진다.



C	0	1	2	3	4	5
0	0	12	4			
1		0		9999	9999	
2				9999	9999	
3						6
4						10

F	0	1	2	3	4	5
0		12				
1		-12		6		
2				6		
3						6
4						6

R	0	1	2	3	4	5
0		0	4			
1		0		9993	9999	
2				9993	9999	
3						0
4						4

모든 경로에 위의 과정을 적용하면 결과는 다음과 같이 만들어지게 된다.

F	0	1	2	3	4	5
0		12	4			
1		-12	-4	6	0	
2				6	4	
3						6
4						10

이 알고리즘의 주된 것은 경로를 찾고 찾아진 경로의 가중치 즉 흐름의 양의 값을 변경하여 테이블을 작성하는 것이다.

소스 코드

```
#include<fstream.h>
int n, t;
int b[201];
int a[202][202]; //[0]은 시작 지점, [n*2+1]은 끝지점
int chk[202];
int v[202];
int foundway;

int f_min(int x, int y)
{
	if(x<y) return x;
	return y;
}

void INPUT()
{
	int i;
	ifstream in("input.txt");
	in>>n;
	for(i=1;i<=n;i++) {
		in>>b[i+n];
	}
	for(i=1;i<=n;i++) {
		in>>b[i];
	}
	in.close();
}

void f2(int x, int y, int st)
{
	int i;
	chk[x]=1;
	v[st]=x;
```

```
	if(x==n)
	{
		for(i=st;i>1;i--)
		{
			a[v[i]][v[i-1]]+=y;
			a[v[i-1]][v[i]]-=y;

		}
		foundway=1;
		return;
	}

	for(i=0;i<=n;i++)
	{
		if(chk[i]==1 || a[x][i]==0) {
			continue;
		}

		f2(i, f_min(y, a[x][i]), st+1);

		if(foundway==1) {
			return;
		}
	}
}

void f_reset()
{
	int i, j;
	for(i=0;i<=n*2+1;i++)
	{
		for(j=0;j<=n*2+1;j++) a[i][j]=0;
	}

	for(i=1;i<=n;i++){
		a[0][i]=b[i];
```

```
                a[i+n][n*2+1]=b[i+n];
        }

        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++) a[i][j+n]=t;
        }
}

void f1()
{
        int i;
        int flag;
        for(t=0;;t++) //최대값을 최소로 하기 위해서 최대값을 정해놓고 돌린다
        {
                f_reset();

                n=n*2+1; //계산의 편의를 위해 잠깐동안 n을 바꾼다
                while(1)
                {
                        for(i=0;i<=n;i++) chk[i]=0;
                        foundway=0;
                        f2(0, 99999999, 1);
                        if(foundway==0) break;
                }
                flag=0; //n을 다시 원래대로
                for(i=0;i<=n;i++)
                {
                        if(a[0][i]!=0) {
                                flag=1;
                                break;
                        }
                }
                n=(n-1)/2;
                if(flag==0) return;
        }
}
```

```
void OUTPUT()
{
	int i, j, max=0;
	ofstream out("output.txt");

	for(i=n+1;i<=n*2;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i][j] > max) {
				max = a[i][j];
			}
		}
	}
	out<< max << endl;

	for(i=n+1;i<=n*2;i++)
	{
		for(j=1;j<=n;j++) out<<a[i][j]<<" ";
		out<<endl;
	}
	out.close();
}

int main()
{
	INPUT();
	f1();
	OUTPUT();
	return 0;
}
```

6 괄호의 제거

▸▸ 컴퓨터에서의 사칙연산과 괄호 연산식의 해석에 대한 이해와 연산식을 트리로 만드는 방법 및 만들어진 트리를 중위(inorder), 전위(preorder), 후위(postorder) 운행하여 의미를 해석할 수 있게 구현하는 문제이다.

문제 (난이도 ★☆☆☆)

당신은 수학시간에 괄호의 사용법에 대해서 배운 적이 있다. 만약 어떤 식이 (1+2)*2라고 적혀있다면 이 결과는 6일 것이고, 1+(2*2)라고 적혀있다면 이 결과는 5가 될 것이다. 괄호의 역할은 우선순위를 정하는 것이다. 문자로 적혀있어도 괄호의 역할은 같다. 즉, 모든 실수 a, b, c에 대해서, (a+b)*c는 a+(b*c)와 다르다. 또한, 당신은 수학시간에 괄호를 생략해도 되는 경우에 대해서 배웠다. a+(b*c)는 괄호를 생략하여 다음과 같이 써도 모든 실수 a, b, c에 대해서, 같은 결과를 가지게 된다. a+b*c 이는 곱하기의 우선순위가 더하기보다 높기 때문이다.
입력으로 들어오는 연산자는 4가지이다. (+, -, *, /)
각각 사칙연산의 덧셈 뺄셈 곱셈 나눗셈을 뜻하며, +와 -는 우선순위가 같고, *와 /는 서로 우선순위가 같다. 또한 +, -는 *, /보다 우선순위가 낮다. +, -끼리는, 우선순위가 같으므로 왼쪽부터 차례대로 계산한다. 이제 당신이 해야 할 일은 주어진 수식에서 괄호를 제거할 수 있는 경우, 모든 괄호를 제거하는 것이다. 물론, 괄호를 제거해서 다른 결과가 나오면 안 된다. 괄호를 제거할 수 없는 경우는 입력받은 결과를 그대로 출력한다.
제한시간은 2초이다.

프로그램의 이름은 “postorder”로 한다.

입력 형식

입력 파일의 이름은 “Input.txt”로 한다.
첫째 줄에 수식이 하나 주어진다. 문자는 알파벳 소문자(a~z)만이 쓰이며, 한 글자이다. 괄호는 (,)만이 쓰이고, 연산자는 +, -, *, /만이 쓰인다. 문자와 괄호, 연산자 사이에는 하나 이상

의 공백이 있다. 문장의 총 길이는 200자 이하이다. 입력되는 알파벳은 연속적이 아닐 수 있다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
입력 형식과 같은 형식이되, 필요 없는 괄호만 “모두” 제거하여 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
7
a + ( b * c )
```

출력의 예(Output.txt)

```
a + b * c
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
23
( a + ( b * c ) * d +e * f ) + g * ( h + z )
```

출력의 예(Output.txt)

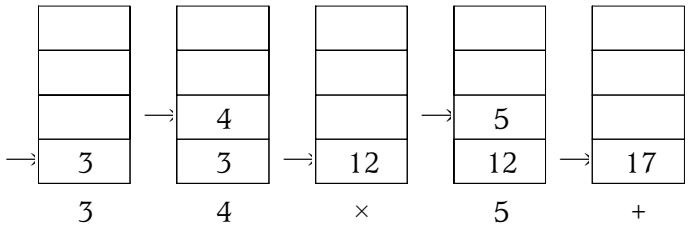
```
a + b * c * d +e * f + g * ( h + z )
```

2 문제의 배경 및 관련 이론

트리 만들기와 트리의 운행과 연산자에 대한 개념을 학습할 기회가 될 수 있는 문제로 연산식을 이진트리의 형태로 만들고 만들어진 이진트리를 역 폴리시(Polish) 방법인 전위(Preorder), 후위(Postorder), 중위(Inorder) 운행의 개념 이해하는 것이 중요하다. 컴퓨터의 기본적인 연산식의 연산방법을 이해하고 활용할 수 있도록 하기 위한 목적이 큰 문제로 이 문제를 통하여 트리의 구성 방법과 트리의 운행법을 정확히 학습할 기회가 되었으면 한다.
우선 이 문제의 주된 점은 역 Polish 표기법으로 어떤 수식의 값을 구하는 데 있어 매우 효율적인 스택의 구조에 맞게 도입된 개념의 이해이다. 이것은 Postfix 또는 Postorder라 불린다. 이것은 피연산자가 앞에 표시되고 후위에 연산자가 표시되는 형태를 취한다. 예를 들면 다음과 같다.

3 × 4 + 5 　　((3 × 4) + 5) → ((3 4) × 5) +
Postorder 로 표시하기 위한 중간 과정 　　3 4 × 5 +

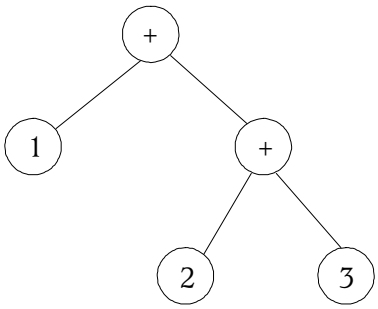
왼쪽의 3 × 4 + 5는 중위식(inorder) 표기라 한다. 이것을 후위식(Postorder)로 만드는 방법은 위에서 소개된 것과 같이 연산자를 피연산자의 후위에 표시하면 된다. 또한 Postorder로 표현된 연산식은 스택에서의 동작은 다음과 같다. 연산자를 만날 때 까지 스택에 값을 Push하고 연산자를 만나면 스택의 위의 두 개를 pop하여 연산하고 연산 결과를 다시 스택에 Push하는 방법으로 이루어진다.



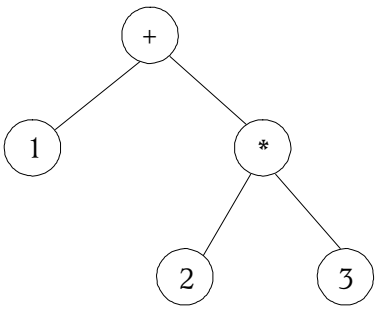
이외 나머지 중위식, 전위식 표기법도 다양하게 응용되어지는 형태를 찾아보고 각 운행법에 대한 개념과 구현 방법에 대한 학습이 필요하다. 구현할 때 배열 또는 리스트 중 무엇을 이용하는 것이 용이한지에 대한 관심도 필요하겠다.

3 문제 해설

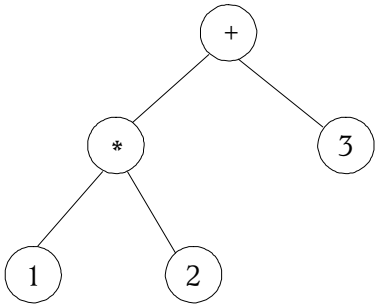
쉬운 문제라고 단순하게 처리하는 방식을 따라가려면 예외가 많아지고, 그에 따른 예외처리 조항을 많이 넣어야 한다. 그러므로 이 문제를 해결하기 위해서는 식 트리라고 하는 것이 있다. 우리가 흔히 쓰는 수식을 이진트리의 형태로 나타낸 것인데, 이 경우엔 이항 연산만이 가능하며, 계산하는 방법은 전위(inorder)운행으로 탐색하면서 계산하면 된다.
트리에서 노드는 연산자의 종류(+, -, *, /)를 나타내지만, leaf 노드는 모두 숫자(operand)를 가지고 있다.
예를 들면 다음과 같다. 1+2+3을 나타낸 것이다.



1+2*3은 다음과 같다. 물론 1+(2*3)도 같게 그려진다.

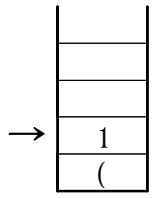


다음은 1*2+3이다.

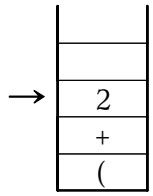


식 트리를 만들려면 첫째, 역폴란드 기법을 이용해서 수식을 후위식으로 만들어 놓고 이진트리로 구성한다.

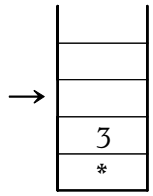
위의 (1 + 2) * 3을 후위식으로 표현하면 “1 2 + 3 *”가 된다. 후위식으로 표현하는 방법은 다음과 같다. 우선, 읽어 들인 수식을 차례대로 스택에 push, pop한다.



다음에 스택에 들어오는 값이 숫자가 아니고 연산자가 되므로 연산자를 push 하기 전에 숫자 1을 pop하여 배열에 저장한다. 연산자 "+" push 한다.
d[0]=1



다음에 스택에 들어오는 값이 “)” 이므로 이전이 스택에 있는 모든 값을 pop 하여 배열에 저장한다. 단, “(”와 “)” 괄호는 모두 삭제한다.
d[1]=2
d[2]="+"



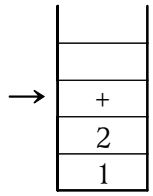
읽어 들인 모든 값을 스택에 push하였으므로 다시 pop하여 배열에 저장한다.
d[3]=3
d[4]="*"

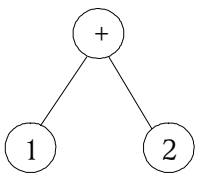
배열에 저장된 문자열을 차례대로 보면 후위식 표기가 만들어진 것을 볼 수 있다.

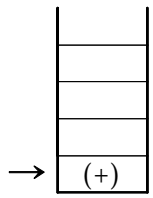
1	2	+	3	*	

둘째는 만들어진 후위식 표기를 가지고 식 트리를 다음과 같이 만든다.

배열에 저장된 값을 가지고 차례대로 다시 스택에 push한다. push할 값이 연산자가 되면 연산자를 root로 스택에 저장된 값을 pop하여 왼쪽, 오른쪽 자식노드를 연결한다.

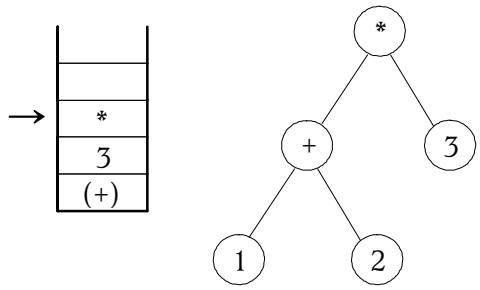






pop하여 두 수 1, 2를 "+"의 자식 노드로 만들고 스택에는 1, 2의 root 노드인 "+"의 주소를 push하여 놓는다.

다시 스택에 push하고 연산자를 만나면 다시 pop하여 연산자 “*”을 root 노드로 하고 pop된 두 값을 왼쪽, 오른쪽 자식노드로 연결한다.



마지막으로 구성된 식 트리를 중위 운행하여 가면서 출력하면 된다. 식 트리가 주어졌을 때 괄호를 쳐야 하는지 여부는, 지금 어떤 노드에 대해서, 그 노드의 왼쪽 자식이 자신보다 우선순위가 낮을 경우이다. 다시 말하면, 위의 예와 같이 최상위 레벨의 root와 두 번째 레벨의 root 간의 우선순위를 비교하여 우선순위가 낮은 연산자 “+”의 자식노드를 출력할 때 “(”와 “)”을 넣어 출력한다. 우선순위가 같은 경우에는 연산자가 “-”이거나 “/”일 때 또한 “(”와 “)”을 넣어 출력한다. 출력을 위한 소스코드는 다음과 같다.

```
void travel_tree(NODE *now)
{
        if(!now->type) {
                fprintf(fp," %c ",now->v+'a'); return;
        }
        int l,r,m;
        l=now->l->type; r=now->r->type; m=now->type;
        if(p[l]<p[m])   fprintf(fp," ( ");
        travel_tree(now->l);
        if(p[l]<p[m])   fprintf(fp," ) ");
        fprintf(fp," %c ",now->type);
        if(p[r]<p[m] || (p[r]==p[m] && (m=='-' || m=='/'))) {
                fprintf(fp," ( ");
        }
        travel_tree(now->r);
        if(p[r]<p[m] || (p[r]==p[m] && (m=='-' || m=='/'))) {
                fprintf(fp," ) ");
        }
}
```

소스 코드

```
#include <stdio.h>
#define inpfilename "input.txt"
#define outfilename "output.txt"
#define N 250
struct NODE {
        NODE *l,*r;
        char type;
        int v;
}*root;
struct DATA {
        char type;
        int v;
        NODE *node;
};
int n,m;
DATA data[N];
DATA d[N];
int p[255];
FILE *fp=fopen(outfilename,"wt");
void input()
{
        int i;
        char t;
        char temp[10];
        FILE *fp=fopen(inpfilename,"rt");
        fscanf(fp,"%d\n",&n);
        for(i=0;i<n;i++) {
                fscanf(fp,"%s",temp);
                t=temp[0];
                switch(t){ //입력된 문자열을 구조체에 저장한다.
                        case '+': data[i].type='+'; break;
                        case '-': data[i].type='-'; break;
                        case '*': data[i].type='*'; break;
                        case '/': data[i].type='/'; break;
```

```
                        case '(': data[i].type='(';          break;
                        case ')': data[i].type=')';          break;
                        default: data[i].type=0;   data[i].v=t-'a';
                }
        }
}

void rpn() // 역 폴리시 표기를 위한 루틴
{
        p[0]=10;
        p['+']=p['-']=8;
        p['*']=p['/']=9;
        p['(']=-999;
        int i, top=0;
        DATA stack[N];
        for(i=0;i<n;i++)
        {
                if(data[i].type==')') {
                        while(stack[top-1].type!='(') {
                                d[m++]=stack[--top];
                        }
                        top--;
                        continue;
                }
                if(data[i].type=='(') {
                        stack[top++]=data[i];
                        continue;
                }
                while(top && p[stack[top-1].type]>=p[data[i].type]) {
                        d[m++]=stack[--top];
                }
                stack[top++]=data[i];
        }
        while(top){
                d[m++]=stack[--top];
        }
}
```

```
void make_tree() // 식 트리 만드는 루틴
{
        int I, top=0;
        DATA *stack[N];
        NODE *a,*b;
        for(i=0;i<n;i++)
        {
                d[i].node=new NODE;
                d[i].node->l=NULL;
                d[i].node->r=NULL;
                d[i].node->type=d[i].type;
                d[i].node->v=d[i].v;
        }
        for(i=0;i<n;i++)
        {
                if(d[i].type==0)  stack[top++]=&d[i];
                else{
                        a=stack[top-2]->node;
                        b=stack[top-1]->node;
                        d[i].node->l=a;
                        d[i].node->r=b;
                        top--;
                        top--;
                        stack[top++]=&d[i];
                }
        }
}

void travel_tree(NODE *now) // 중위 운행
{
        if(!now->type) {
                fprintf(fp," %c ",now->v+'a'); return;
        }
        int l,r,m;
        l=now->l->type; r=now->r->type; m=now->type;
        // 연산자의 우선순위에 따른 괄호의 출력여부 결정
        if(p[l]<p[m])  fprintf(fp," ( ");
```

```
        travel_tree(now->l);
        if(p[l]<p[m])   fprintf(fp," ) ");
        fprintf(fp," %c ",now->type);
        if(p[r]<p[m] || (p[r]==p[m] && (m=='-' || m=='/'))) {
                fprintf(fp," ( ");
        }
        travel_tree(now->r);
        if(p[r]<p[m] || (p[r]==p[m] && (m=='-' || m=='/'))) {
                fprintf(fp," ) ");
        }
}

void process()
{
        rpn();
        n=m;
        root=new NODE;
        root->l=NULL; root->r=NULL;
        make_tree();
        root=d[n-1].node;
}

void output()
{
        travel_tree(root);
        fprintf(fp,"\n");
}

void main()
{
        input();
        process();
        output();
}
```

7 하노이 타워

▸▸ 재귀 호출의 기본 문제 하노이 타워의 변형 문제로 재귀 호출에 따른 수행속도의 저하를 간단한 규칙성을 이용하여 수행 속도 면에서 빠르게 처리하는 방법적 아이디어의 한 형태를 보여주고 있다.

문제 (난이도 ★★☆☆☆)

하노이 탑 문제를 들어 보았을 것이다. 3개의 막대기 중 하나에 n개의 원판이 꽂혀 있고, 이 원판들을 다른 막대기로 옮기는 문제이다. 이 문제를 풀 때의 이동 회수가 2^n -1임은 잘 알려져 있다.
하노이 탑에서 n개의 원판이 있을 경우에 이 원판의 이동 경로를 모두 출력하는 프로그램을 만드는 것은 쉬운 일이다. 하지만 n이 커지면 출력 개수가 2^n -1이기 때문에 출력하지 못한다. (일반적인 하노이 문제와 같이 막대는 항상 3개로 한다.)
당신이 할 일은, 원판이 n개 있을 때, 최소한의 횟수로 이동시키는 방법 중에서, k번째에 어떤 작업을 수행해야 하는지를 출력하는 것이다. k번째만을 출력하면 된다. 물론 k는 1에서 2^n-1까지의 정수이다. 제한시간은 2초이다.

프로그램 이름은 “hanopath”로 한다.

참고사항

Visual c++를 이용한다면, k를 입력 받을 때 큰 자릿수 연산을 하지 않고 다음과 같이 입력 받을 수 있다. 또한, 스트림 방식으로는 __int64형의 입력은 불가능하다.
원하지 않는다면, 또는 Visual c++를 이용하지 않는다면 이 방식을 이용하지 않아도 문제를 푸는데 지장은 없다.

```
__int64 k;
fscanf(fp,"%I64d",&k);
```

입력 형식

입력 파일의 이름은 “Input.txt”로 한다.
첫째 줄에 디스크의 수를 정수 n($1 \leq n \leq 60$)과 몇 번째 이동값k($1 \leq k \leq 2^n-1$)가 주어진다. 각 디스크들의 번호는 1, 2, …, n이며, 잘못된 입력은 주어지지 않는다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
k번째 순서에서 몇 번째 원판을 몇 번째 막대기에서 몇 번째 막대기로 이동시키는지 출력한다. 예를 들면, 3번 원판을 1번 막대에서 2번 막대로 움직일 경우엔 3 : 1 -> 2 로 출력한다.
숫자와, :, ->사이는 공백 하나 이상이 들어가야 하며, ->내부에는 공백이 없어야한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
3 4
```

출력의 예(Output.txt)

```
3 : 1 -> 3
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
4 10
```

출력의 예(Output.txt)

```
1 : 3 -> 1
```

2 문제의 배경 및 관련 이론

여러 개의 함수로 구성된 프로그램에서 함수는 다른 함수를 호출 할 수 있다. 때로는 함수가 자기 자신을 호출하기도 하는데 이것을 재귀 호출이라 부른다. 재귀 호출을 이용하여 주어진 문제보다 “크기가 작은 문제”를 풀어서 원래 문제를 해결하는 방식의 알고리즘을 재귀 알고리즘이라 한다. 이런 재귀 호출에 있어 주의점은 첫째, 종료조건 제시이고 둘째는 크기가 더 큰 문제를 재귀 호출하여선 안 된다는 것이다. 이런 경우에는 무한 루프에 빠진다. 재귀 호출의 문제는 어떤 유형의 문제에서도 항상 등장하는 기본적 구조로 매우 중요하다. 단, 재귀 호출은 수행시간 면에서 비재귀형보다는 좀 느리다는 점을 항상 인식하고 프로그램에 사용해야한다. 이 재귀 호출에 가장 좋은 예가 하노이 타워 문제이다. 하노이 타워는 처음에 세 개의 막대 A, B, C 가 있는데, 첫 번째 막대인 A 막대에는 N 개의 원반이 있고 B, C 막대는 비어 있다. 또한, 현재 A 막대에 있는 N 개의 원반은 크기가 큰 것일수록 아래쪽에 놓여 있다. 이때, 원반을 하나씩 움직여서 모든 원반을 C 막대로 옮기는 것이다. 단, 이 과정에서도 역시 큰 접시를 작은 접시 위에 놓을 수 없다는 규칙이 적용되는 문제이다. 프로그래밍을 하는 사람이라면 한번쯤 이 문제를 해결해 보았을 것이라고 본다. 재귀 호출을 이용하는 것으로 함수 자체는 아주 간단하게 구현된다. 아래 코드는 하노이 타워의 재귀 함수를 나타낸 것이다.

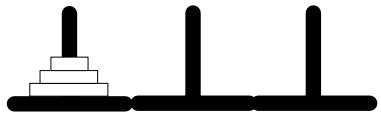
```
void Hanoi(char a, char b, char c, int n) {
        if( n > 0 ) {
                Hanoi(a, c, b, n-1);
                printf("%d는 %c 에서 %c로 이동“, n, a, c);
                Hanoi(b, a, c, n-1);
        }
}
```

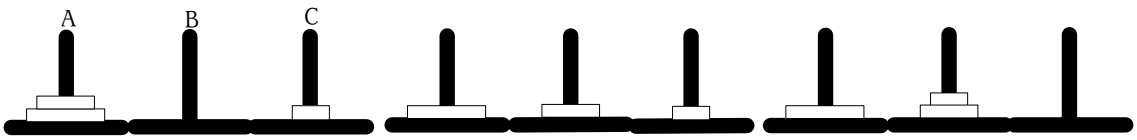
최근에는 기본 하노이 타워를 변형한 문제들이 많이 출제되고 있으나 기본적인 하노이 타워에 대한 이해가 없이는 어려운 문제로 인식하고 있다. 이번 하노이 타워 문제를 통하여 정확한 하노이 타워의 이해와 응용에 관한 관심을 가지고 다양하게 변형된 문제를 학습하는 기회가 되었으면 한다.

3 문제 해설

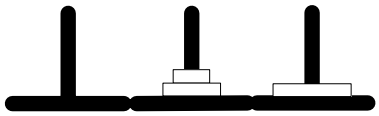
일반적인 하노이 타워 문제와 같이 단순 재귀호출을 이용하여 문제를 해결할 수 있다. 하지만 원반의 개수 n이 40이상 정도부터는 시간의 문제로 결과를 기대할 수 없다. 그렇기 때문에 일반적 재귀 호출로는 문제를 해결하기가 어렵다. 이 문제 해결을 위한 주 관심은 하노이 타워가 가지는 규칙성과 대칭성을 추출할 수 있으면 간단히 해결할 수 있다는 것이다.

우선 가장 기본적인 원판 3개를 가지고 살펴보면 다음 그림과 같이 이동 경로를 볼 수 있다.

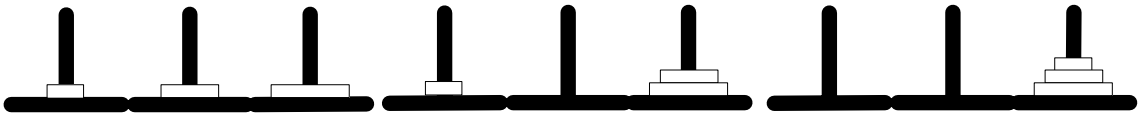




1) 1번 원판을 A에서 C로 이동 2) 2번 원판을 A에서 B로 이동 3) 1번 원판을 C에서 B로 이동



4) 3번 원판을 A에서 C로 이동



5) 1번 원판을 B에서 A로 이동 6) 2번 원판을 B에서 C로 이동 7) 1번 원판을 A에서 C로 이동

위와 같은 이동을 막대 A, B, C를 숫자 1, 2, 3에 대응하여 적어보면 다음과 같다.

1 : A → C	1 : 1 → 3
2 : A → B	2 : 1 → 2
1 : C → B	1 : 3 → 2
3 : A → C	3 : 1 → 3
1 : B → A	1 : 2 → 1
2 : B → C	2 : 2 → 3
1 : A → C	1 : 1 → 3

4번째 단계를 중심으로 3번째와 5번째, 2번째와 6번째, 1번째와 7번째가 서로 규칙성을 가지는 것을 볼 수 있다. 시작하는 막대를 From, 도착 막대를 To라고 할 때 대칭적인 위치에 있는 이동 막대의 번호는 6 - From - To의 식에 따른다.

예를 들어, 3번째 단계와 5번째 단계를 보면 3번째 단계에서 1번 원판이 3번(From) 막대에서 2번(To) 막대로 이동하고 있고, 5번째 단계에서는 1번 원판이 2번 막대에서 1(6-3-2=1)번 막대로 이동한다. 2번째와 6번째의 경우도 같은 규칙성을 가지므로 6단계는 2단계의 1번과 2번 막대로의 이동에 대해 2번 막대에서 3(6-1-2=3)번 막대로의 이동을 볼 수 있다.

이런 규칙을 이용하면 결과적으로 하노이 타워의 이동 경로는 $(2^n-1)/2+1$의 경우의 수 만큼만 계산되어지면 나머지 경우는 규칙성에 의해 계산되어 질 수 있는 것을 알 수 있다.

소스 코드에서는 S[N+1] 배열에 원판의 수에 따른 이동 회수를 넣는다.
주된 내용은 입력되어지는 원판의 수 n을 하나씩 감소시키면서 m-q(몇 번째 이동값 - 처음에는 초기값 0을 그 이후는 q의 변화값)의 값이 s[i-1](처음에는 i=n, i>=1일 때까지 i--한 값) 배열의 이동회수와 비교하여 비교값에 따라 아래와 같이 값을 변경한다.

```
m-q < s[i-1]   : to = 6 - from - to 로 변경
m-q = s[i-1]   : i, from, to 순으로 출력
m-q > s[i-1]   : q=q+s[i-1]+1,   from = 6 - to - from 으로 변경
```

이렇게 처리하면 하노이 타워의 주된 재귀 호출 없이 간단하게 몇 번째 이동값의 원판의 번호와 막대의 번호를 출력할 수 있다. 위에서 사용하는 from, to는 기본적인 막대의 시작과 도착을 나타내는 변수이다.

소스 코드

```
#include <stdio.h>
#define inpfilename "input.txt"
#define outfilename "output.txt"
#define N 60
FILE *fp=fopen(outfilename,"wt");
int n;
__int64 m;
__int64 q;
__int64 s[N+1];

void input()
{
	FILE *fp=fopen(inpfilename,"rt");
	fscanf(fp,"%d %I64d",&n,&m);

	int i;
	s[0]=1;
	for(i=1;i<=n;i++)
	{
		s[i]=s[i-1]*2;
	}
	for(i=0;i<=n;i++){
		s[i]--;
	}
}

void process()
{
	q=0;
	m--;
	int i;
	int from,to;
	from=1; to=3;
	int type=0;
```

```
	for(i=n;i>=1;i--)
	{
		if(m-q<s[i-1])
		{
			to=6-to-from;
		}
		else if(m-q==s[i-1])
		{
			fprintf(fp,"%d : %d -> %d\n",i,from,to);
			break;
		}
		else
		{
			q+=s[i-1]+1;
			from=6-to-from;
		}
	}
}

void main()
{
	input();
	process();

}
```

8 울타리 만들기

▸▸ 계산기하학의 기본적인 연산을 익히고 제시된 문제를 해결해 봄으로써 계산문제에 대한 기하학적인 관점을 학습하고 변형된 문제에 응용할 수 있는 기본기를 기르는 학습이다.

문제 (난이도 ★★☆☆☆)

로또에 당첨된 민우는 꿈에 그리던 섬을 하나 샀다. 섬은 직사각형으로 네모난 섬이고 그 섬에는 많은 큰 나무들이 여기 저기 자라고 있었다. 멋진 섬을 방문한 민우는 이 나무들의 모습을 보고 아주 좋은 생각을 해냈다. 그것은 바로 나무들을 이용하여 섬의 울타리를 만드는 것이다. 울타리는 섬에 자라는 많은 나무들을 모두 포함할 수 있는 위치에 자라는 나무들을 연결하여 울타리를 치는 것이다. 섬에 울타리를 치기 위해서 많은 나무들 중에 어떤 나무를 선택해야 하는지를 찾는 프로그램을 작성하시오. 단, 울타리는 볼록한 다각형의 형태를 가진다. 결코 오목하게 들어가는 부분이 없는 울타리를 만들어야 한다. 또한 나무들은 위치는 x축과 y축의 좌표 형식으로 표현될 수 있다.

프로그램의 이름은 “convexhull”로 한다.

입력 형식

입력파일의 이름은 “Input.txt”로 한다.
첫째 줄에 주어지는 정점의 수로 정수 $n(1 \le n \le 100)$이 주어진다.
다음 줄에는 n줄 만큼의 정점의 좌표가 주어진다.

출력 형식

출력파일의 이름은 “Output.txt”로 한다.
첫째 줄에는 convex hull을 구성하는 정점의 수를 표시한다.

다음 줄부터 n줄 만큼 convex hull을 구성하는 정점을 시계 반대방향으로 차례대로 표시한다.

입력과 출력의 예 1

입력의 예(Input.txt)

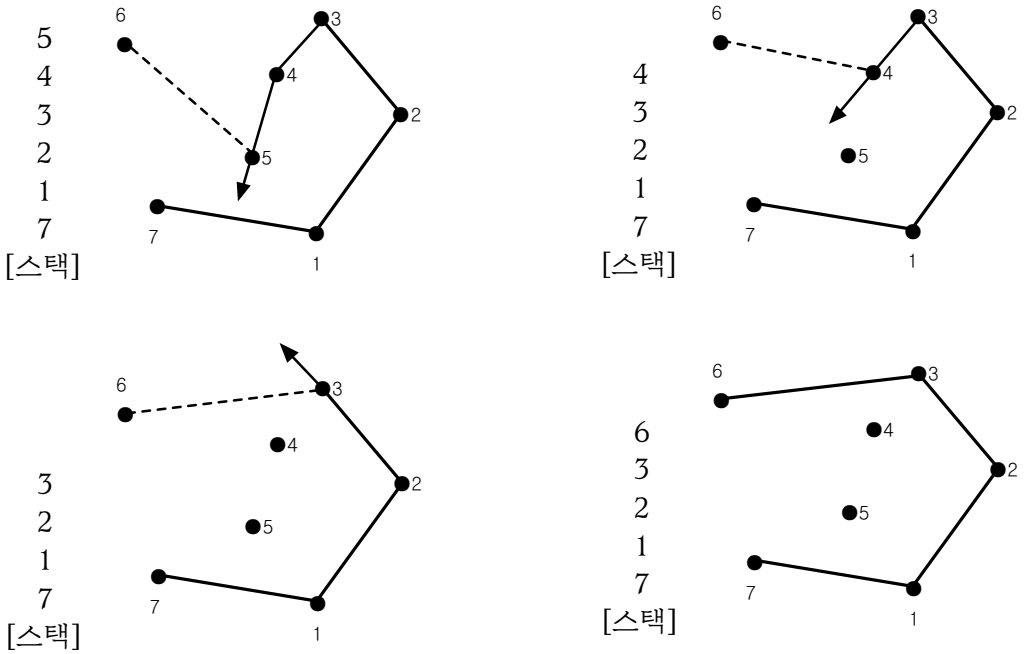
```
10
2 6
3 9
4 4
4 8
5 2
6 7
7 6
7 8
9 3
10 6
```

출력의 예(Output.txt)

```
6
2 6
4 2
8 3
9 6
7 8
3 9
```

2 문제의 배경 및 관련 이론

최초의 계산 기하학문제는 유클리드 이전 그리스 시대 수레에 물건을 싣고 다니며 물건을 파는 장사꾼이 두 집 a와 b에 물건을 배달하게 되었는데 물건의 무게가 무거워 a 집에 배달한 수 다시 수레로 되돌아와서 b 집의 물건을 배달하게 되었다. 수레를 길의 어느 위치에 두면 두 집을 왕복하는 거리가 가장 짧게 될까? 라는 유명한 문제부터라고 한다. 계산을 위한 수식의 사칙연산이 아닌 기하학적인 내용을 토대로 계산문제를 해결하는 방법이다. 컴퓨터를 통한 계산의 결과는 항상 오차를 가지게 되고 이런 오차가 크게는 하나의 커다란 오류를 발생시킬 수 있다. 이런 점에서 계산 기하를 이용하면 오차 없이 계산을 할 수 있다는 것이다. 어려운 연산의 문제도 기하적인 형태의 문제로 재설정하여 쉽게 이해할 수 있게 할 수 있다는 것이다. 이런 점에서 기하학의 발전이 이루어지고 있으며 이 추세와 문제의 유형이 다양하다는 점 때문에 경시대회 문제에 한 문제씩 제시되고 있는 상황이다. 그렇기 때문에 기하에 관련된 기본적인 표현과 평면상의 점과 선들의 관계에 대한 다양한 개념을 익히는 계기가 되기 바란다. 이번 문제에서는 계산기하의 기본적 내용인 polygon의 한 종류인 convex hull에 대한 내용이다. convex hull이란 평면상의 주어지는 많은 점들을 포함하는 가장 작은 볼록 다면체를 말하는 것으로 기하학의 기본 연산자로 세 점이 주어질 때 한 점이 다른 두 점의 직선상의 왼쪽 혹은 오른쪽에 위치하는가를 검사하는 연산과 직선위에 존재하는지의 검사하는 연산과 두 선분이 서로 교차하는지를 검사는 연산, 교차점을 구하는 연산 등을 이해하여야 한다. 간단히 그림을 통해 convex hull을 구하는 그래함 알고리즘을 제시하면 다음과 같다.

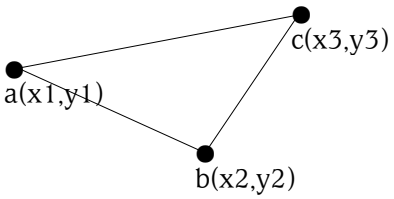


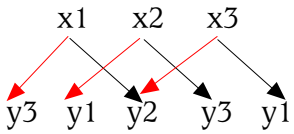
3 문제 해설

제시된 문제는 어떻게 생각하느냐에 따라 쉽게 또는 어렵게 판단되는 문제이다. 단순히 생각할 수 있는 알고리즘이라면 “convex hull”의 테두리 두 점을 잇는 선분은 이 선분을 중심으로 다른 모든 점들은 왼쪽 혹은 모두 오른쪽에 존재하면 된다. 그러므로 모든 선분의 쌍에 대하여 모든 점들이 한 직선에 대하여 한 쪽에 있는지를 조사하면 된다. 하지만 이렇게 모든 쌍에 대하여 조사한다면 시간 복잡도는 $O(n^3)$정도가 된다. 하지만 시간 복잡도를 n정도로 줄이면서 해결할 수 있는 방법이 있다. 이것은 앞서 소개한 Graham의 알고리즘이다. 이것은 주어지는 모든 정점을 각도에 따라 정렬하고 각 정점을 잇는 선분의 각도를 조사하여 convex hull의 점을 찾는다.

이 알고리즘을 사용하기 위해 필요한 계산기학학적인 연산은 다음과 같다.

첫째, 세 점을 꼭지점으로 하는 삼각형의 면적을 내는 연산이다.






```
int Area(point a, point b, point c)
{
        return a.x*b.y - a.y*b.x + a.y*c.x - a.x*c.y + b.x*c.y - c.x*b.y
}
```

둘째, 세 점 a, b, c가 주어졌을 때, 점 c가 점 a로부터 점 b를 통과하는 방향성을 가지는 직선의 왼쪽에 위치하는가를 검사하는 연산이다.

```
boolean left(point a, point b, point c)
{
        if(Area(a, b, c) > 0 ) return TRUE
        else return FALSE
}
```

위 Left 함수에서 Area(a, b, c)=0과 같으면 세 점은 한 직선상에 있다는 것을 알 수 있으며, Area(a, b, c) < 0이면 한 점은 한 직선의 오른쪽에 있다고 판단할 수 있다.

위 두 가지의 기하 연산을 기본으로 하여 주어지는 정점에 대해 각도에 따른 정렬을 한다. 각 점들을 점 P[1]과의 각도에 따라 정렬할 때 두 점 p1, p2의 상대적인 순서를 정하는 알고리즘은 다음과 같다.

```
int compare(point p1, point p2)
{
        area=Area(P[1], p1, p2);
        if(area > 0 ) return -1;
        else if(area < 0 ) return 1;
        else {
                length1 = 점 p1과 점 P[0]와의 거리;
                length2 = 점 p2과 점 P[0]와의 거리;
                if(length1 < length2 ) return -1;
                else if(length1 > length2 ) return 1;
                else return 0;
        }
}
```

위의 기본 연산자를 이용하여 두 선분의 교차유무와 한 선분위에 세 점의 위치, 교차점을 구하는 등 다양한 연산으로 확장할 수 있다.

소스 코드에서는 간단하게 선분의 각도를 구하고 이전의 점과의 각도를 비교하여 convex hull을 구성하는 정점으로 구분한다. 각도를 구하는 함수는 다음과 같다.

```
double GetAngle(int x1, int y1, int x2, int y2){
        double dx=x2-x1, dy=y2-y1, angle;
        if(dx==0){
                if(dy>=0) return 90.0;
                else return 270.0;
        }
        angle=atan(dy/dx) / 3.14 * 180.0;
        if(dx<0) angle+=180.0;
        if(angle<0) angle+=360.0;
        return angle;
}
```

소스 코드

```
#include <stdio.h>
#include <math.h>

double GetAngle(int x1,int y1,int x2,int y2);
struct point
{
        int x;
        int y;
} p[200];

FILE *in, *out;

void main()
{
        int minp, i, h, hull[200], n;
        double prev, angle, minangle;

        in=fopen("input.txt", "r");
        out=fopen("output.txt", "w");

        fscanf(in,"%d",&n);
        for(i=1; i<=n; i++)
        {
                fscanf(in,"%d %d", &p[i].x, &p[i].y);
        }

        p[n+1].x=p[1].x;   p[n+1].y=p[1].y;
        minp=1;
        for(i=2; i<=n; i++)
        {
                if(p[i].y < p[minp].y) minp=i;
        }
        h=1;
        hull[1]=minp;
```

```
        prev=0.0;
        while(h<= n)
        {
                minangle=361.0;
                for(i=1; i<=n; i++)
                {
                        if(i!=hull[h])
                        {
                                angle=GetAngle(p[hull[h]].x,p[hull[h]].y,p[i].x,p[i].y);
                                if(angle<0)
                                {
                                        angle+=360.0;
                                }
                                if(minangle>angle && angle >prev)
                                {
                                        minangle=angle;
                                        minp=i;
                                }
                        }
                }
                if(minp==hull[1])
                {
                        break;
                }
                hull[++h]=minp;
                prev=GetAngle(p[hull[h-1]].x, p[hull[h-1]].y, p[hull[h]].x, p[hull[h]].y);
        }

        fprintf(out,"%d\n", h);
        for(i=1; i<=h; i++)
        {
                fprintf(out,"%3d %d\n", p[hull[i]].x, p[hull[i]].y);
        }
}
```

```
double GetAngle(int x1, int y1, int x2, int y2)
{
	double dx=x2-x1, dy=y2-y1, angle;
	if(dx==0)
	{
		if(dy>=0)
		{
			return 90.0;
		}
		else
		{
			return 270.0;
		}
	}
	angle=atan(dy/dx) / 3.14 * 180.0;
	if(dx<0) angle+=180.0;
	if(angle<0) angle+=360.0;
	return angle;
}
```

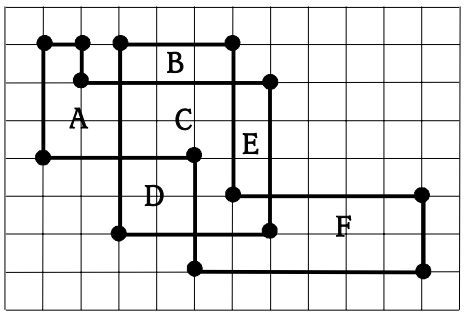
9 직각다각형의 면적 구하기

▸▸ 2차원 평면의 문제를 1차원의 문제로 변경하여 해결하는 기법의 학습과 sweep-line이라는 개념의 데이터 구조 형성 및 기하학적 문제로 한국 정보올림피아드 기출 문제이다.

문제 (20회 KOI 기출 문제)

2차원 평면상에서 직각다각형은 그 변을 구성하는 선분이 수직이거나 수평인 다각형이다. 일반 직각다각형이란 그 변을 이루는 선분이 서로 교차할 수 있는 직각다각형이고, 단순 직각다각형이란 그 변을 이루는 선분이 서로 교차하지 않는 다각형이다. 이 문제에서 다루고자 하는 다각형은 일반 직각다각형이면서 다음 조건을 만족하는 직각다각형이다.

① 다각형의 꼭지점은 서로 같은 지점에 위치할 수 없다.
② 다각형의 각 꼭지점에서는 다각형의 변을 이루는 하나의 수직선분과 하나의 수평선분이 그 선분의 끝점에서 만난다.
③ 다각형의 변이 교차하는 경우는 반드시 수평선분과 수직선분 사이에 교차한다. 즉, 수평선분끼리 교차하거나, 수직선분끼리 교차하는 경우는 없다.



이렇게 구성된 일반 직각다각형은 평면을 여러 개의 단순 직각다각형으로 분할하게 된다. 예를 들면, 아래 그림은 14개의 선분으로 구성된 일반 직각다각형이다. 이 직각다각형은 다각

형의 외부는 제외하고 평면을 6개의 단순 직각다각형 A, B, C, D, E, F 로 분할하고 있으며, 이들 단순 직각다각형 중에서 가장 면적이 넓은 다각형은 F 이다.
일반 직각다각형이 주어졌을 때, 이 직각다각형에 의하여 분할되는 단순 직각다각형 중에서 그 면적이 가장 큰 단순 직각다각형의 면적을 계산하는 프로그램을 작성하시오.

프로그램 이름은 “box”로 한다.

입력 형식

입력 파일의 이름은 “Input.txt”로 한다.
첫째 줄에는 일반 직각다각형의 꼭지점의 개수를 나타내는 정수 N $(4 \le N \le 1,000)$ 이 나온다.
다음 N 개의 줄에는 각각 하나의 꼭지점에 대한 좌표를 나타내는 두 개의 정수 x와 y $(0 \le x, y \le 10,000)$가 입력된다. 첫 번째 정수 x는 그 꼭지점의 X-좌표를 나타내며, 두 번째 정수 y는 그 꼭지점의 Y-좌표를 나타낸다.
연속되는 두 개의 꼭지점을 선분으로 연결하고, 마지막 꼭지점과 첫 번째 꼭지점을 연결하면 입력되는 일반 직각다각형이 구성된다.

출력 형식

출력 파일의 이름은 “Output.txt”로 한다.
첫 번째 줄에 입력되는 일반 직각다각형에 의하여 분할되는 가장 면적이 큰 단순 직각다각형의 면적을 나타내는 정수를 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

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

출력의 예(Output.txt)

```
5
```

입력과 출력의 예 2

입력의 예(Input.txt)

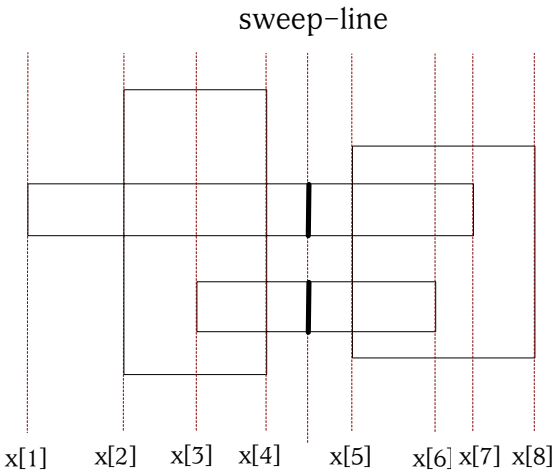
```
14
1 7
2 7
2 6
7 6
7 2
3 2
3 7
6 7
6 3
11 3
11 1
5 1
5 4
1 4
```

출력의 예(Output.txt)

```
10
```

2 문제의 배경 및 관련 이론

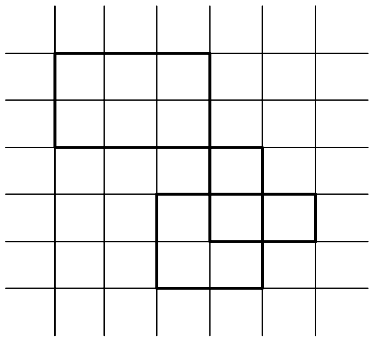
계산기하학 문제를 해결하는 방법으로 Plane-Sweep 기법에 대한 이해를 돕고자 문제를 제시하였으며 이 기법은 평면을 수직선(sweep-line이라 부름)으로 왼쪽에서 오른쪽으로 쓸어가면서 주어진 문제를 해결하는 기법이다. 기본적인 아이디어는 해결하기 복잡한 2차원의 문제를 단순한 1차원의 문제로 나누어 해결하고자 하는 것이다. sweep-line이 1차원을 표현한다. 이를 위하여 다음 두 가지의 데이터 구조를 이용한다.
첫째, sweep-line의 상태를 나타내는 데이터 구조로 해결하고자하는 데이터들과 현 위치의 sweep-line이 교차하고 있는 상태를 표현한다.
둘째, event point 순서를 결정하는 데이터 구조로 sweep-line이 움직임에 따라 그 상태를 변경하여야 하는데 상태가 변화되는 sweep-line의 위치(event-point라 부름)의 순서를 가지는 데이터 구조이다.
위의 두 개의 데이터 구조는 해결하고자 하는 문제에 따라 각각 달리 설정하여 이용한다.
이 기법을 이용하여 다양한 문제를 해결하게 되는데 예를 들면, 평면 x, y축에 평행한 사각형들이 주어졌을 경우, 그 사각형들의 union이 차지하는 면적을 계산하는 문제가 있다. 이 문제를 푸는 간단한 Plane-Sweep 기법을 소개하면 다음과 같다.



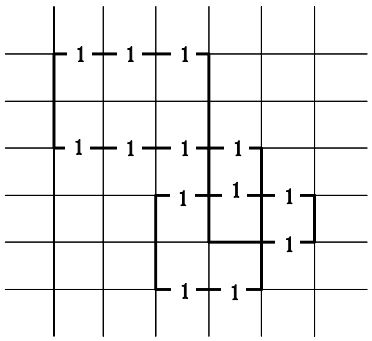
event-point : 주어진 각 사각형의 왼쪽, 오른쪽 테두리 선의 x좌표로 이들을 sorting 한 후, 차례로 x[1], x[2], …, x[2n]이라 한다.
event-points x[4]와 x[5]사이에서의 면적은 sweep-line에서의 굵은 선의 길이 * (x[5] - x[4])가 된다. 또한 sweep-line의 상태는 현재 위치에서 sweep-line은 각 사각형과 교차하거나 하지 않는다. sweep-line과 사각형의 교차점은 선분으로 나타내어진다. 현재 위치에서 sweep-line 상태는 sweep-line과 각 사각형과의 교차점들인 선분들의 집합으로 구성된다. 이 sweep-line 상태로부터, 우리는 1차원 선분들의 union의 길이를 구할 수 있다.
이외에 Line-Segment Intersection 문제로 평면상에서 n개의 선분들이 주어졌을 때, 모든 교차점을 구하는 문제의 경우도 이 Plane-Sweep 기법을 이용한다. 단순한 방법으로는 $O(n^2)$인 것을 이 기법을 이용하여 시간 복잡도를 낮출 수 있다.

3 문제 해설

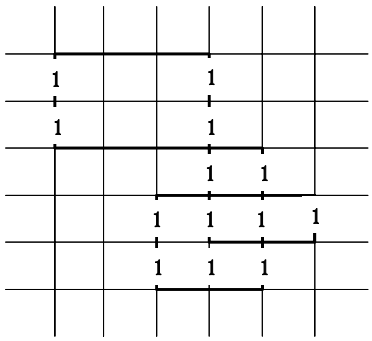
배열의 방 하나를 하나의 1*1 사각형으로 생각하고 10000 * 10000 배열을 잡아 하나씩 그리는 것인데 이것은 좌표의 범위가 크다면 메모리가 부족할 수도 가능 하더라도 시간이 너무 오래 걸리게 된다. 기하 문제의 가장 어려운 점이 또한 이러한 것이기 때문에 다른 방법을 생각하게 되는데 그 방법이 앞서 말한 Plane Sweep 방법과 Flood Fill 방법이다. 여기서는 Flood Fill 방법으로 설명한다.



일단 어떤 2개의 사각형 사이에 선이 있는지 없는지 판단할 수 있어야 하는데 모든 사각형에 대해 위, 아래, 오른쪽, 왼쪽에 선이 있는지 검사하는 것은 비효율적이다. $O(n^2 \log n)$ 혹은 $O(n^3)$이 걸린다. 그러므로 반대로 어떤 선이 어디에 걸쳐 있는지 알아보고 그 영역에서 어느 방향에 선이 있는지 알아보는 것이다. 일단 가로선을 예로 들면, 2개의 x좌표와 1개의 y좌표를 알면 선을 표시 할 수 있는데, 이 x, y좌표가 몇 번째 영역의 x, y좌표 인지 이분 검색으로 쉽게 찾을 수 있다. 이렇게 찾은 영역에 위, 혹은 아래선이 있다고 표시를 해두는 것이다. 다시 말하면 가로선에 대한 선이 연장선상에 있고 없음을 표시하는 것이다. 표시는 가로선과 세로선으로 나누어 표시한다.



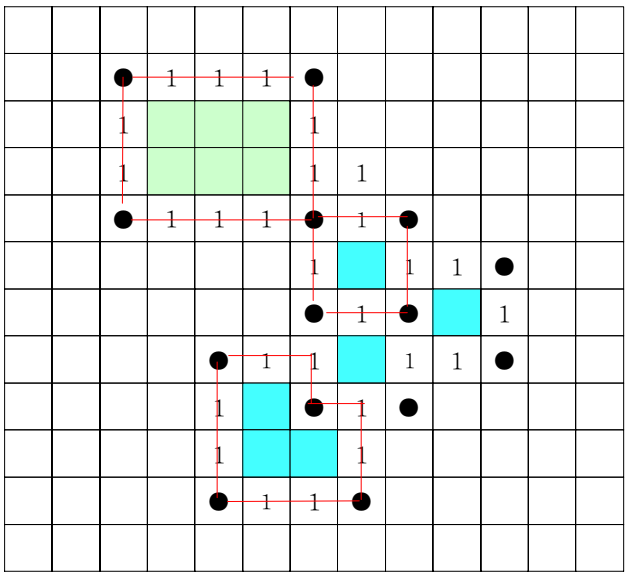
가로선



세로선

이렇게 하면 $O(n \log n)$에 할 수 있다. 이분 검색을 쓰지 않아도 $O(n^2)$ 에 빠르게 할 수 있다. 다시 말하면 주어지는 좌표를 이용하여 좌표간의 연결선이 있고 없음을 인접행렬로

표현한 것이다. 인접행렬을 너비우선탐색을 이용하여 단순사각형을 찾고 그것의 내부 면적을 계산한다.



단, 한번 찾아진 단순 사각형은 다시 찾아지지 않도록 체크할 수 있도록 하면 된다. 또한 면적을 계산하면서 면적의 최소값을 찾아서 리턴하면 된다.

소스 코드

```
#include <stdio.h>
#include <stdlib.h>
#define N 1200
#define M 200000000

int n, n2;
int q[N][2];
int line[2][N][3]; // ----- 0 x에 변화       | 1y에 변화
int x[N],xc;
int y[N],yc;
int map[N][N][4];  // 0 이 가능 1이 불가능
int dir[4][2]={{0,1},{1,0},{0,-1},{-1,0}}; // y, x
int check[N][N];
int ans;
int qu[N*N][2];
int qh, qt;

FILE *in, *out;
int sf(const void *a, const void *b)
{
        int aa,bb;
        aa=*(int *)a;
        bb=*(int *)b;
        if (aa<bb)
        {
                return -1;
        }
        if (aa>bb)
        {
                return 1;
        }
        return 0;
}
```

```
void delete_rpt(int t[],int &tc)
{       // 주어지는 좌표에서 중복되는 좌표값은 배제
        int i,j;
        int m_tc;
        m_tc=0;
        qsort(t,n,sizeof(t[0]),sf);
        for(i=0;i<tc;i++)
        {
                for(j=i+1;j<tc;j++)
                {
                        if (t[i]!=t[j]) break;
                        t[j]=M;
                        m_tc++;
                }
                i=j-1;
        }
        qsort(t, n, sizeof(t[0]), sf);
        tc-=m_tc;
}

int b_search(int t[],int a, int b, int key)
{       // 한 선분이 걸쳐있는 영역의 값을 찾는다.
        int m;
        m=(a+b)/2;
        if (t[m]>key)
        {
                return b_search(t, a, m, key);
        }
        else if (t[m]<key)
        {
                return b_search(t, m+1, b, key);
        }
        else { // if (t[m]==key)
        {
                return m;
        }
}
```

```
void swap(int &a, int &b)
{
	int t;
	t=a;
	a=b;
	b=t;
}

void make_map()
{	// 인접행렬 테이블 만들기
	int i, j;
	int a, b;
	int c;
	for(i=0;i<n2;i++)
	{	// 가로선
		a=b_search(x, 0, xc, line[0][i][0]);
		b=b_search(x, 0, xc, line[0][i][1]);
		c=b_search(y, 0, yc, line[0][i][2]);
		if (a>b)
		{
			swap(a, b);
		}
		for(j=a; j<b; j++)
		{
			map[c][j][3]=1;
			if (c>0) map[c-1][j][1]=1;
		}
	}

	for(i=0;i<n2;i++)
	{	// 세로선
		a=b_search(y, 0, yc, line[1][i][0]);
		b=b_search(y, 0, yc, line[1][i][1]);
		c=b_search(x, 0, xc, line[1][i][2]);
		if (a>b) {
			swap(a, b);
		}
```

```
                for(j=a;j<b;j++)
                {
                        map[j][c][2]=1;
                        if (c>0)
                        {
                                map[j][c-1][0]=1;
                        }
                }
        }
}

int sqr(int a,int b)
{       // 면적 구하기
        int s;
        s=(y[a+1]-y[a]) * (x[b+1]-x[b]);
        return s;
}

int area(int py,int px)
{       // bfs 로 탐색하여 단순사각형의 면적 알아내기
        int i;
        int qx,qy;
        int dx,dy;
        int sum;
        qh=0;
        qu[0][0]=py;
        qu[0][1]=px;
        qt=1;
        check[py][px]=1;
        sum=sqr(py,px);
        for(;;)
        {
                qy=qu[qh][0];
                qx=qu[qh][1];
                for(i=0;i<4;i++) {
                        if (map[qy][qx][i]==1) continue;
                        dy=qy+dir[i][0];
```

```
                        dx=qx+dir[i][1];

                        // 제한 구역 밖
                        if (dy<0 || dy>=yc-1 || dx<0 || dx>=xc-1) return -M;
                        if (check[dy][dx]==1) continue;
                        check[dy][dx]=1;
                        qu[qt][0]=dy;
                        qu[qt][1]=dx;
                        qt++;
                        sum+=sqr(dy,d x);
                }
                qh++;
                if (qh==qt) break;
        }
        return sum;
}

void pro()
{       // 선분에 대한 두 개의 x 좌표와 한 개의 y 좌표값
        int i,j;
        int sw;
        int v;
        if (q[0][0]!=q[1][0]) sw=0;
        else sw=1;
        for(i=0;i<n;i++)
        {
                line[sw][i/2][0]=q[i][sw];
                line[sw][i/2][1]=q[i+1][sw];
                line[sw][i/2][2]=q[i][!sw];
                sw=!sw;
        }

        delete_rpt(x, xc);
        delete_rpt(y, yc);
        make_map();

        ans=-M;
```

```
        for(i=0;i<yc-1;i++)
        {
                for(j=0;j<xc-1;j++)
                {
                        if (check[i][j]==0)
                        {
                                v=area(i,j);
                                if (v>ans) { // 단순 사각형의 면적중 최소값 찾기
                                        ans=v;
                                }
                        }
                }
        }
}

void main()
{
        int i;
        in=fopen("input.txt", "r");
        out=fopen("output.txt", "r");

        fscanf(in,"%d", &n);
        for(i=0;i<n;i++)
        {
                fscanf(in, "%d %d", &q[i][0], &q[i][1]);
                x[i]=q[i][0];
                y[i]=q[i][1];
        }
        q[n][0]=q[0][0];
        q[n][1]=q[0][1];
        xc=n;
        yc=n;
        n2=n/2;
        pro();
        fprintf(out, "%d\n", ans);
}
```

10 학교 저축

▸▸ 정렬되지 않은 데이터 중에서 시간복잡도가 되도록 작은 방법을 사용해서 주어진 데이터 중 원하는 순서를 갖는 데이터를 찾는 문제이다.

문제 (난이도 ★★☆☆☆)

정보올림피아드 학교는 N명의 학생이 학교를 다니고 있고, 학생들은 1번부터 N번까지 번호가 부여되어 있다. 학교에서는 학생들에게 저축하는 것을 권장하고 있어 매달 정기적으로 저축하는 날을 정해 모든 학생들이 저축을 하고 있다.
학생들은 매달 1,000에서 100,000원까지 1,000원 단위로 저축을 할 수 있고 저축기간은 학년 3월부터 부터 3학년 2월까지 36개월이다.
학교에서는 자주 저축을 하고 있는 학생 들 중 K번째로 저축을 많이 한 학생을 알고 싶어 한다. 저축하는 학생들 중 K번째로 저축을 많이 하는 학생과 저축금액을 찾는 프로그램을 작성하시오. 단, 시간복잡도가 O(N)에 가깝도록 프로그래밍 하시오.

예를 들어서, N=10이고, 저축액이 (136,000, 7,000, 54,000, 129,000, 12,000, 21,000, 437,000, 18,000, 215,000)이고 K가 4일 때, K번째로 저축을 많이 한 학생의 번호는 3번고 저축 금액은 54,000원이다. 만일 저축금액이 같은 학생이 있다면 같은 학생의 번호를 모두 출력한다.

프로그램의 이름은 "store"로 한다.

입력 형식

입력 파일 이름은 “Input.txt"로 한다.
첫 번째 줄에는 학생 수 N($1 \leq N \leq 1000$)이 주어지며 1번부터 N까지 학생들에게 번호가 부여

되어있다.
두 번째 줄에는 K가 주어진다.
세 번째 줄에는 학생의 저축 금액 N개의 금액이 주어진다.

출력 형식

출력 파일이름은 “Output.txt"로 한다.
첫 번째 줄에 K번째 저축을 많이 한 학생의 번호를 출력한다.
두 번째 줄에 K번째 저축을 많이 한 학생의 저축금액을 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
10
4
136000 7000 54000 129000 12000 21000 44000 37000 18000 215000
```

출력의 예(Output.txt)

```
3
54000
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
35
15
34000 4000 123000 325000 189000 75000 217000 119000 12000 450
36000 75000 17000 254000 329000 312000 1000 45000 317000 7500
275000 136000 170000 54000 75000 39000 132000 14000 5000 750
18000 75000
```

출력의 예(Output.txt)

```
11
89000
```

2 문제의 배경 및 관련 이론

시간복잡도란 입력되는 자료집합의 크기가 증가함에 따라 알고리즘의 효율이 어떻게 변화하는지를 대략적이나마 추정하는 함수로 표현하는 것을 말하며 보통 빅 O로 표기한다.

알고리즘의 복잡도를 표현하는 데 흔히 사용하는 함수들을 복잡도가 낮은 것에서 높은 것으로 나열하면, 상수, $\log n$, n, $n \log n$, n^2, n^3, 2^n, $n2^n$, $n!$이 있다.

n이 작을 때에는 1000n이 n^2보다 크지만 n이 커짐에 따라 n^2이 빠른 속도로 증가하므로 큰 n에 대해서는 n^2이 1000n보다 더 큰 시간 복잡도를 갖는 함수가 된다. 이때 분기점이 되는 것은 n^2이 1000n일 때이다.

시간 복잡도에 따른 수행 시간을 비교하면 다음과 같다.

복잡도	항목 16개	항목 32개	항목 64개	항목 128개
$O(\log n)$	4초	5초	6초	7초
$O(n)$	16초	32초	64초	128초
$O(n \log n)$	64초	160초	384초	896초
$O(n^2)$	256초	17분	68분	273분
$O(n^3)$	68분	546분	73시간	24일
$O(2^n)$	18시간	136년	5억년	계산불능

정렬 방법 중 평균적으로 가장 빠르다고 알려져 있고 가장 많이 이용하는 것이 퀵 정렬(quick sort)이다. 퀵 정렬은 주어진 데이터 목록에서 임의의 수를 선택하여 선택된 원소보다 작은 값을 가지는 원소들과 크거나 같은 값을 가지는 원소를 분리하여 순환적으로 정렬하는 방법으로 자료구조에서는 정렬에 속하며, 알고리즘에서는 Divide and Conque에 속한다.
이 밖의 다른 정렬 방법과 평균 시간 복잡도는 아래와 같다.

정렬방법	시간 복잡도	설명
선택 정렬	$O(n^2)$	데이터를 연속적으로 선택해 적절한 위치에 배치
버블 정렬	$O(n^2)$	서로 인접한 레코드의 킷값을 비교해 교환
삽입 정렬	$O(n^2)$	정렬된 데이터에 새로운 레코드를 적절한 위치에 삽입
쉘 정렬	$O(n \log n)$	서브파일을 형성해 서브 파일을 삽입 정렬하는 방식
힙 정렬	$O(n \log n)$	데이터를 완전 이진트리로 형성하여 정렬
병합 정렬	$O(n \log n)$	서브파일을 한 개의 서브파일로 변환하는 방식

3 문제 해설

자료처리의 효율성은 자료가 어떤 기준에 따라 정렬되어 있는가에 따라 좌우된다. 예를 들어 특정한 사람에 대한 전화번호를 전화번호부에서 찾는다고 할 때 전화번호부가 알파벳의 순서로 정렬되어 있지 않다면 특정 사람의 전화번호를 찾는다는 것은 불가능할 것이다.

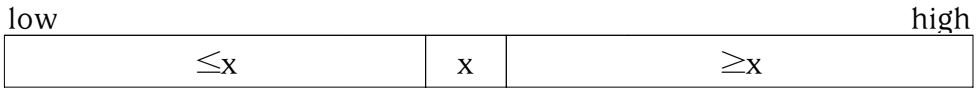
자료를 정렬하는 방법에는 기본적인 정렬 방법인 삽입 정렬, 선택 정렬, 버블 정렬, 효율적인 정렬 알고리즘인 쉘 정렬, 힙 정렬, 퀵 정렬, 병합 정렬, 기수 정렬 등이 있다.

주어진 문제는 정렬 문제이지만 시간 복잡도가 O(n)에 가깝도록 하라는 제한 사항이 있기 때문에 정렬방법에 있어 적절한 방법을 선택해서 사용해야 한다.

1. 퀵 정렬(Quick sort)의 이용

정렬알고리즘을 구현할 때 평균적으로 가장 빠른 시간복잡도를 갖는 정렬 알고리즘이 퀵 정렬이다. 퀵 정렬은 최악의 경우 O(n^2)이 되지만 최악의 경우가 되는 일은 거의 없고 대부분 평균 시간 복잡도는 정렬 알고리즘이 도달할 수 있는 가장 빠른 수준인 O($n \log n$)을 갖는다.

퀵 정렬을 하기 위해 먼저 주어진 배열을 두 개의 부분 배열로 나누는데 이 기준 값이 되는 것을 피봇(pivot)이라 한다. 첫 번째 부분배열은 피봇보다 작거나 같은 요소들을 원소로 하고 두 번째 부분배열은 피봇보다 더 크거나 같은 요소들을 원소로 갖는다.



두 부분 배열은 독립적으로 정렬될 수 있으며 이것이 행해지기 전에 분할 과정이 두 부분 배열에 대해 반복 수행된다. 또한 각각의 배열이 분할되기 위해 새로운 피봇이 각 부분배열에서 선택된다. 피봇을 어떠한 값을 선택하는가에 따라 수행시간에 영향을 주지만 좋은 피봇을 선택하는 것은 그리 쉬운 작업이 아니다.
분할되는 두 부분배열의 요소 수를 되도록 같게 분할 할 수 있는 피봇이 좋은 피봇이다. 피봇을 선택하는 방법은 3가지로 주어진 값의 첫 번째 값, 중간 값, 마지막 값을 선택할 수 있다. 그러나 데이터가 어떠한 상태로 되어 있는지 모르는 상황에서 정렬이 진행 될 수 록 정렬되어질 많은 요소가 적절한 위치에 놓여 질 확률이 높으므로 배열의 중간에 위치한 요소를 선택하는 것이 좋다.

퀵 정렬을 하는 절차는 다음과 같다.

1. 피봇을 배열의 중앙값으로 설정하고 가장 좌측 값과 교환한다.
2. i를 피봇을 제외한 수열의 왼쪽 끝, j를 수열의 오른쪽 끝에 설정한다.
3. 수열을 i위치에서 오른쪽으로 검색하면서 피봇 값보다 큰 값 lower을 j 위치에서 왼쪽으로 검색하면서 피봇 값보다 작은 값 upper을 찾는다.
4. lower와 upper을 교환한다.
5. 3-4항목을 조건에 맞는 i와 j가 없을 때까지 반복한다.
6. 피봇 값과 경계 값을 서로 교환하여 두 그룹으로 나눈다.
7. 자료가 두 부분으로 나뉘었으며 각각의 부분배열에 대해 1-6을 적용한다.

아래 예시를 통해 136,000, 7,000, 54,000, 129,000, 44,000, 12,000, 21,000, 37,000, 215,000의 데이터 값이 주어질 때 퀵 정렬 알고리즘을 통해 데이터 값이 정렬되어지는 과정을 살펴보자.

먼저 피봇을 중간에 있는 값 44,000으로 정하고 첫 번째 요소와 바꿔 준다.
[136,000 7,000 54,000 129,000 **44,000** 12,000 21,000 37,000 18,000 215,000]

[**44,000** 7,000 54,000 129,000 136,000 12,000 21,000 37,000 18,000 215,000]

피봇을 제외한 가장 좌측에서 피봇 값 44,000보다 큰 값을 우측에서 부터 피봇 값 44,00보다 작은 값을 찾아서 서로 교환한다.
[**44,000** 7,000 54,000 129,000 136,000 12,000 21,000 37,000 18,000 215,000]
↑ ↑
[**44,000** 7,000 18,000 129,000 136,000 12,000 21,000 37,000 54,000 215,000]

[**44,000** 7,000 18,000 129,000 136,000 12,000 21,000 37,000 54,000 215,000]
↑ ↑
[**44,000** 7,000 18,000 37,000 136,000 12,000 21,000 129,000 54,000 215,000]

[**44,000** 7,000 18,000 37,000 136,000 12,000 21,000 129,000 54,000 215,000]
↑ ↑
[**44,000** 7,000 18,000 37,000 21,000 12,000 136,000 129,000 54,000 215,000]

더 이상 좌우측과 교환할 데이터가 없으므로 피봇 값과 피봇 값의 경계에 있는 12,000의 자리를 서로 바꾸어 두 그룹으로 만든다.

[12,000 7,000 18,000 37,000 21,000] **44,000** [136,000 129,000 54,000 215,000]

[12,000 7,000 18,000 37,000 21,000] **44,000** [136,000 129,000 54,000 215,000]

각각의 그룹에 대해 이전 정렬 방법을 반복 수행한다.

[12,000 7,000 **18,000** 37,000 21,000] **44,000** [136,000 **129,000** 54,000 215,000]

[**18,000** 7,000 12,000 37,000 21,000] **44,000** [**129,000** 136,000 54,000 215,000]

[12,000 7,000 **18,000** 37,000 21,000] **44,000** [**129,000** 136,000 54,000 215,000]

↑ ↑

[12,000 7,000 **18,000** 37,000 21,000] **44,000** [**129,000** 54,000 136,000 215,000]

[12,000 7,000 **18,000** 37,000 21,000] **44,000** [54,000 **129,000** 136,000 215,000]

다시 두개의 그룹을 나눈다.

[12,000 7,000] **18,000** [37,000 21,000] **44,000** [54,000] **129,000** [136,000 215,000]

[**12,000** 7,000] **18,000** [**37,000** 21,000] **44,000** [54,000] **129,000** [**136,000** 215,000]

[7,000 **12,000**] **18,000** [21,000 **37,000**] **44,000** [54,000] **129,000** [**136,000** 215,000]

각 그룹에 대한 정렬이 완료되었다.

7,000 12,000 18,000 21,000 37,000 44,000 54,000 129,000 136,000 215,000

이상과 같은 퀵 정렬의 평균 수행속도는 O($n \log n$)이다. 그런데 문제에서 보면 시간복잡도를 O(n)에 가깝게 하도록 요구하고 있다. 시간 복잡도를 O(n)으로 하기 위해서는 일반 적인 정렬 알고리즘을 사용하면 안 된다.

2. 계수정렬 방법

계수정렬은 주어진 데이터의 범위가 정해져 있는 경우에 사용한다. 예를 들어 1부터 k사이의 정수 범위에 모든 데이터가 존재한다는 것을 알고 있을 때에만 적용할 수 있는 방법으로 입력 데이터의 출현 횟수를 세어서 정렬하는 방식으로 시간복잡도는 평균/최악의

경우 동일하게 O(n)을 갖는다.

문제에서 보면 학생들이 저축할 수 있는 금액은 매달 1,000원에서 100,000원까지 1,000 단위로 저축할 수 있다고 했다. 그리고 학생들이 저축할 수 있는 기간은 최대 36개월(고1-고3)이다.

그렇다면 한 학생이 한번 이상 저축을 할 때의 금액이 1,000이고 최대로 저축할 때의 금액이 3,600,000이다. 즉, 저축액 W은 1,000 ≤ W ≤ 3,600,000의 사이에 있고 W가 갖을 수 있는 금액의 종류는 3,600가지이다. 3600가지이면 배열에서 모든 경우의 수를 설정할 수 있다.

예를 들어 학생이 최대 10명이고 각각의 학생들이 저축한 총 금액이 1,000원에서 10,00원 범위라면 저축한 금액의 경우의 수는 모두 10가지이다. 입력되는 저축 금액의 데이터가 6,000, 8,000, 1,000, 10,000, 3,000, 3,000, 6,000, 4,000, 3,000, 3,000이고 이중 4번 축을 많이 한 학생과 저축 금액을 찾는다면 먼저 모든 경우의 수 만큼 배열을 확보해야 한다.

배열은 중복되지 않는 저축 금액의 종류를 알아보기 위한 배열과 학생 번호별로 저축금액을 알 수 있는 배열이 필요하다. 입력되어지는 데이터 값을 해당 배열에 저장한 다음 입력이 완료되면 저장된 값에서 원하는 순서를 갖는 금액을 O(n)의 시간복잡도로 찾을 수 있다.

학생 번호에 따른 저축금액은 다음 배열과 같다.

1	2	3	4	5	6	7	8	9	10
6,000	8,000	1,000	10,000	3,000	3,000	6,000	4,000	3,000	3,000

저축 금액이 나오는 빈도의 수를 배열에 나타내면 다음과 같다.

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

따라서 저축 금액의 종류에 따른 배열에서 4번째 나타나는 금액은 6,000이다. 이 금액을 가지고 학생 번호에 따른 저축액 배열에서 저축액이 6,000인 학생들의 번호를 찾으면 1번 학생과 7번 학생이 된다.

소스 코드

[1. 퀵 정렬]

```
#include <stdio.h>

#define INFILE "input.txt"
#define OUTFILE "output.txt"

#define MAX 10000

int n, k;
int store[MAX], num[MAX];

void input(void);
void output(void);

void quicksort(int left, int right);
int      partition(int pivot, int right);
void swap(int a, int b);

void input()
{
        FILE *fp=fopen(INFILE, "rt");

        int i;

        fscanf(fp, "%d %d", &n, &k);

        for (i=0; i<n; i++) {
                fscanf(fp, "%d", &store[i]);
                num[i]=store[i];
        }

        fclose(fp);
}
```

```
void output()
{
        int i;
        FILE *fp=fopen(OUTFILE, "wt");

        for (i=0; i<n; i++) {
                if(num[i]==store[k-1])
                        fprintf(fp, "%d ", i+1);
        }

        fprintf(fp, "\n%d", store[k-1]);

        fclose(fp);
}

void quicksort(int left, int right)
{
        if (left < right) {

                int middle = partition(left, right);

                                quicksort(left, middle);
                quicksort(middle + 1, right);

        }

        return;
}

int partition(int pivot, int right)
{
        int i, j, pivot_value;

                pivot_value = store[pivot];

                i = pivot - 1,  j = right + 1;
```

```
        while (1) {
                do {
                  i = i + 1;
                } while (store[i] < pivot_value);

                do {
                  j = j - 1;
                } while (store[j] > pivot_value);

            if (i < j)
                  swap(i, j);
            else
                  return j;
        }
}

void swap(int a, int b)
{
        int tmp = store[a];
        store[a] = store[b];
        store[b] = tmp;

        return;
}

int main(void)
{
       input();
       quicksort(0, n-1);
       output();

       return 0;
}
```

[2. 계수정렬]

```
#include <stdio.h>

#define INFILE "input.txt"
#define OUTFILE "output.txt"

#define MAX 3600

int n, k;
int store[MAX], num[MAX];

void input(void);
void countsort(void);
void output(void);

void input()
{
	FILE *fp=fopen(INFILE, "rt");

	int i;

	fscanf(fp, "%d %d", &n, &k);

	for (i=0; i<n; i++) {
		fscanf(fp, "%d", &num[i]);
	}

	fclose(fp);
}

void output()
{
	int i, pos, cnt=0;

	FILE *fp=fopen(OUTFILE, "wt");
```

```
	for (i=MAX-1; i>=0; i--)
	{
		if(store[i]>0 && cnt<k)
		{
			cnt=cnt+store[i];
			pos=i;
		}
	}

	for (i=MAX-1; i>=0; i--)
	{
		if((pos+1)*1000==num[i])
			fprintf(fp, "%d ", i+1);
	}

	fprintf(fp, "\n%d", (pos+1)*1000);

	fclose(fp);
}

void countsort()
{
	int i;

	for(i=0; i<n; i++) {
		store[(num[i]/1000)-1]++;
	}
}

int main(void)
{
	input();
	countsort();
	output();
	return 0;
}
```

11 119 구급대

▸▸ 그래프에서 어느 한지점이 시작점이 되고 다른 한 지점을 목표점으로 주어질 때 시작점에서 목표점까지 주어진 조건을 만족하면서 도착하는 문제이다.

문제 (난이도 ★★★☆☆)

철수네 동네의 소방서에서는 주민들의 편의를 위해 119 구급 대를 운영하고 있다. 가끔 일어나는 일이지만 위급한 일은 분초를 다투는 일이 많기 때문에 119 구급대가 위급한 일이 발생한 목표지점까지 최대한 빨리 도착해야 한다.
119 구급차는 성능이 좋아 일반 자동차보다 아주 빠른 속도로 주행할 수 있지만 코너를 돌 때에는 일반 자동차처럼 속도를 완전히 줄여야 하기 때문에 목표지점에 도달할 때 까지 되도록 코너를 돌지 않고 직선 도로를 통해 도착해야 시간을 줄일 수 있다.

(0,0)

S									
							T		

(9,7)

위의 테이블은 119 구급 대에서 위급한 일이 발생하는 목표지점까지의 지도를 표시한 것이다. 흰색으로 표시되어 있는 부분이 차가 지나갈 수 있는 길을 나타내고 회색으로 칠해져 있는 곳이 장애물을 의미한다. 이제 소방서에서 위급한 일이 발생한 목표지점까지 가장 빠른 시간에 도착할 수 있는 경로를 찾아보자. (출발점은 항상 왼쪽 맨 위이고 도착점은 좌표로 주어진다. 119 구급차는 가로와 세로 방향으로만 움직일 수 있고 대각선으

로는 움직일 수 없다.)

프로그램의 이름은 “ambulance”로 한다.

입력 형식

입력 파일 이름은 “Input.txt"로 한다.
첫째 줄에 지도의 크기 M(0≤M≤100)×N(0≤N≤100)이 주어진다.
두 번째 줄에는 목표점 m×n이 주어진다.(단 M=m+1, N=n+1)
다음 N줄에는 M개의 값이 주어지며 각 값에서 1은 도로이고 0은 진입할 수 없는 물건이 놓여 있다.

출력 형식

출력 파일 이름은 “Output.txt"로 한다.
첫째 줄에 최소 코너를 도는 횟수를 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
10 10
9 7
1 0 0 1 0 0 1 0 0 0
1 1 1 0 0 1 1 1 0 0
1 0 1 0 1 1 0 1 1 1
1 1 1 1 1 0 0 0 0 1
0 1 0 0 1 1 1 1 0 1
1 1 1 1 1 0 0 1 1 1
1 0 0 1 0 0 1 1 0 1
1 0 1 1 1 1 1 0 0 0
1 1 0 1 1 0 1 1 0 1
0 1 1 1 0 1 1 1 1 1
```

출력의 예(Output.txt)

```
6
```

2 문제의 배경 및 관련 이론

그래프는 정점(vertex)으로 부르는 노드들과 정점들 사이를 상호 연결하는 간선(edge)들로 구성되어 있다. 그래프는 모든 자료구조의 가장 일반적인 형태로 볼 수 있고 트리(tree)와 리스트(list)를 그래프의 특정한 형태라 할 수 있다.

그래프는 현실적인 문제와 추상적인 문제 모두를 모델링하기 위해 보편적으로 많이 사용한다. 그래프의 응용분야는 다음과 같다.

1. 컴퓨터의 연결과 통신 네트워크를 모델링한다.
2. 지점들과 지점들 사이를 거리로 나타내거나 어느 한 지점에서 다른 지점까지의 최단 거리를 구할 때 사용한다.
3. 시작 지점에서 목적 지점까지의 경로를 구한다.
4. 하나의 상태에서 다른 상태로 변하는 것을 보여준다.
5. 복잡한 작업을 완성시키기 위해 작업내의 하위 작업들이 완성되어가는 적절한 순서를 찾는다.
6. 인간관계나 일의 관계에서 각각의 관계에 대한 모델링을 한다.

일반적으로 그래프를 표현하기 위한 방법에는 인접 행렬과 인접 리스트의 2가지 방법이 있고 특수한 그래프로는 다음과 같은 것이 있다.

1. 완전 그래프
   모든 정점사이에 에지가 있는 그래프
2. 트리
   연결그래프로서 사이클이 없는 그래프
3. 이분 그래프

4. 오일러 그래프
   어떤 정점 u에서 시작하여 모든 에지를 한번만 지나면서 다시 u로 돌아오는 사이클이 존재하는 그래프
5. 평면 그래프
   그래프의 모든 에지를 교차하지 않도록 하면서 그래프를 평면상에 그릴 수 있는 그래프
6. 4채색 문제

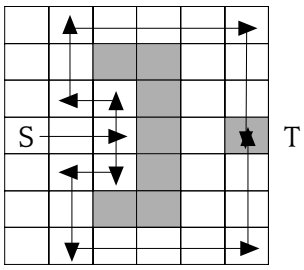
그리고 그래프의 모양에 따라 그래프의 정점을 방문하는 것이 자주 이용되는데 깊이 우선탐색(DFS, depth-first search)과 너비우선 탐색(BFS, breadth first search)이 있다.

3 문제 해설

이 문제는 주어진 좌표를 그래프로 표현하고 그래프의 정점들을 모두 한번 씩 방문하는 너비우선 탐색(Breadth First Search, BFS)이나 깊이 우선 탐색(Depth First Search, DFS) 알고리즘으로 구현할 수 있다.

시작지점에서 출발하여 목표지점까지의 중간에 별다른 장애물이 없다면 쉽게 시작점에서 목표지점까지 도달할 수 있다.

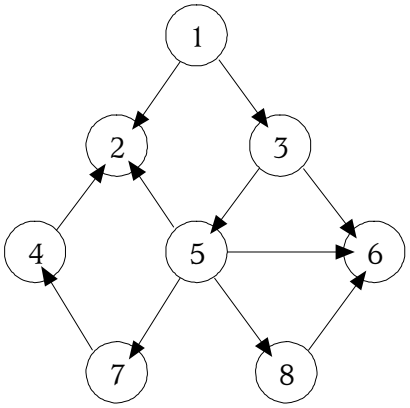
만약 시작지점에서 목표지점까지 중간에 장애물이 존재한다면 장애물을 피해 목표지점에 도달해야 하는데 장애물을 피하는 가장 간단한 방법으로 장애물의 외곽선을 따라 장애물을 피해 움직이다가 목표 지점으로 갈 수 있는 지점에서 다시 목표 지점으로 이동하는 벽 짚고 따라가기 방법이 있다.



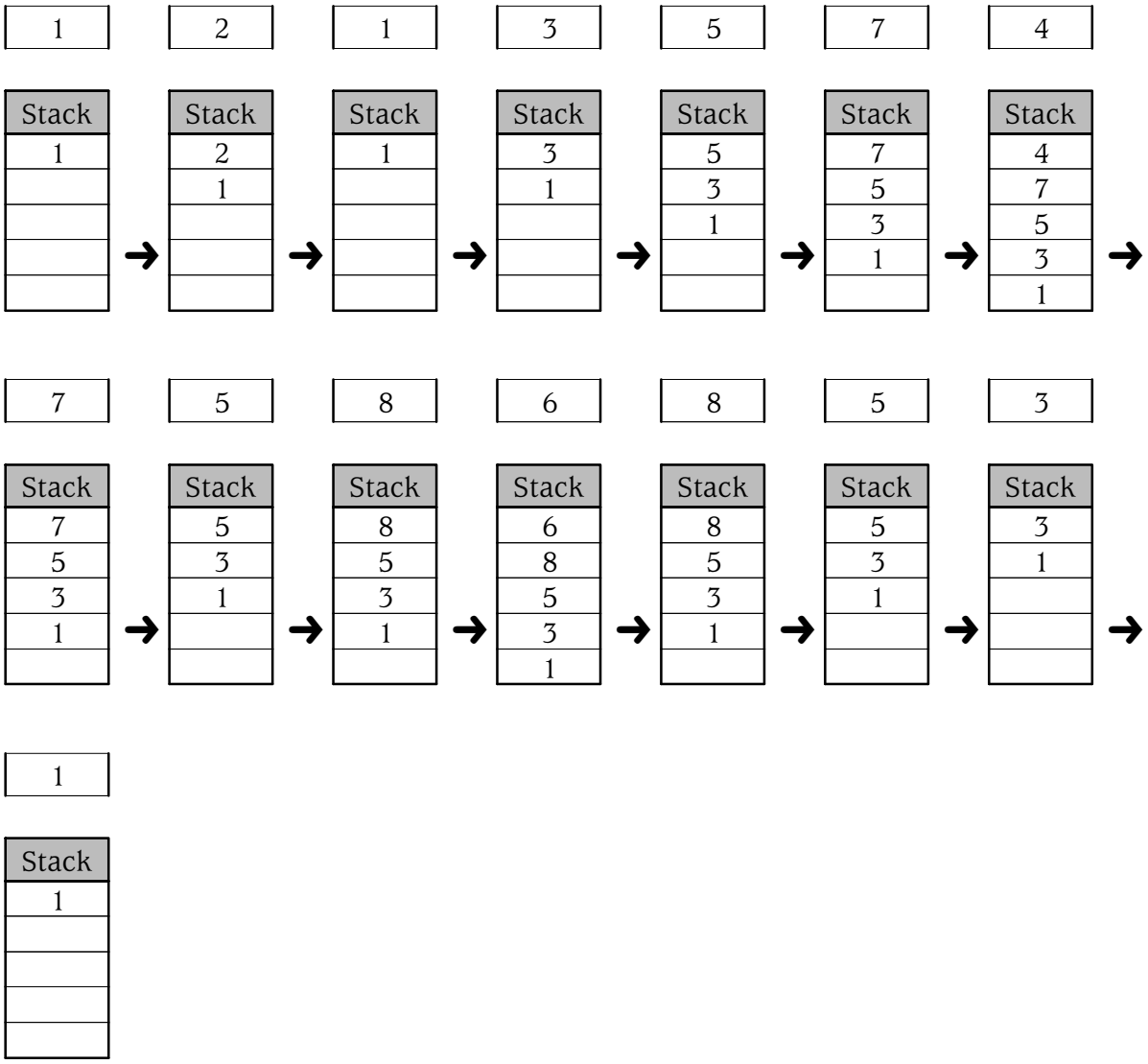
먼저 너비우선 탐색(BFS)를 이용하여 꺽이는 횟수에 관계없이 목표점까지 길을 찾는 방법에 대해 생각해보자.

① 시작 칸을 스택에 넣는다.
② 시작 칸에 시작 표시
③ 스택에서 첫 번째 칸을 뽑아 현재 칸으로 설정한다.
④ 현재 칸과 목표 칸을 비교해 같으면 종료한다.
⑤ 현재 칸의 자식 노드들 중 표시가 되어 있지 않은 칸들이 있으면 그 칸들의 이전 포인터를 현재 칸으로 설정하고 그 칸들을 스택에 추가한다.
⑥ 알고리즘이 종료되거나 스택이 empty될 때까지 단계 3에서 5를 반복한다.

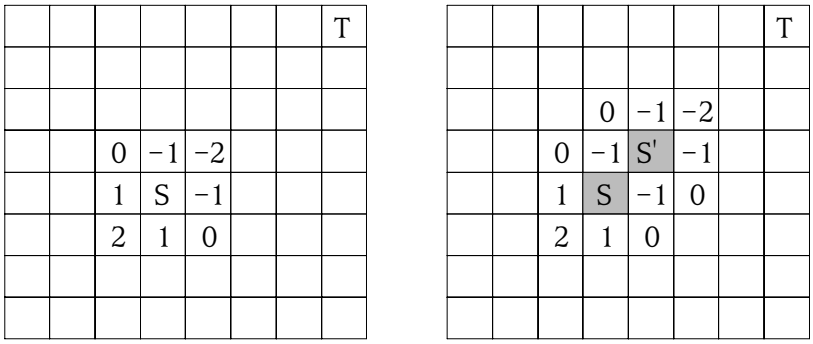
아래 그래프에서 BFS의 탐색 경로를 살펴보자.



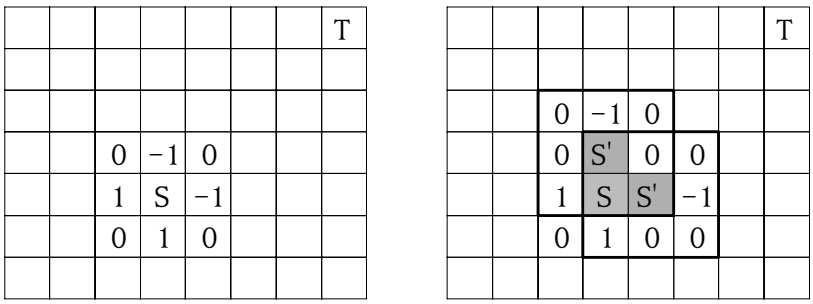
현재 방문하는 지점을 스택에 집어넣고 이어서 방문할 다른 곳을 찾는다. 이때 더 이상 방문할 곳이 없다면 스택에서 값을 하나 꺼내 그전에 방문한 곳으로 가서 새로운 방문 지점을 탐색한다.



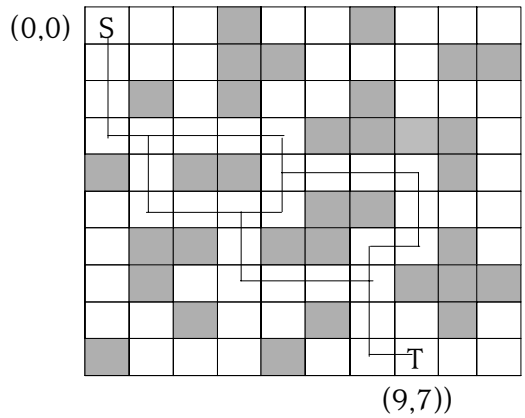
그러나 위와 같은 기본적인 BFS 방법은 목표지점과 관계없는 불필요한 방향까지도 모두 동일하게 탐색하기 때문에 더 많은 시간이 걸린다. 따라서 현재 위치에서 다음 위치를 결정할 때 목표지점과 근접한 칸에 더 많은 가중치를 주는 방법으로 비교 횟수를 줄일 수 있을 것이다.



그러나 119 구급대에서는 구급차가 대각선으로 움직일 수 없으므로 대각선에는 가중치를 주지 않고 목표지점에 가까운 상하좌우에 더 많은 가중치를 주어 목표 방향에 우선권을 주어 움직일 수 있도록 한다. 물론 이때 코너를 되도록 적게 돌아야 하기 때문에 상하좌우 하나의 방향이 결정되면 장애물을 만날 때까지 계속 한 방향으로 진행을 하도록 해야 한다.



주어진 그래프를 BFS로 검색하면 아래와 같은 검색 경로를 얻을 수 있다.



소스 코드

```
#include <stdio.h>

#define INFILE "input.txt"
#define OUTFILE "output.txt"
#define MAXM 100
#define MAXN 100
#define MAXQ 40000
#define INF 100000000

int n, m, tn, tm;
int table[MAXN+2][MAXM+2]={0,};
int matrix[MAXN+2][MAXM+2][4], trace[MAXN+2][MAXM+2];
int queue[MAXQ][3], qfront, qrear;
int dx[]={-1,1,0,0}, dy[]={0,0,-1,1}, neg[]={1,0,3,2};

int input(void)
{
	FILE* fp;
	int i, j;

	if((fp = fopen(INFILE, "r")) == NULL)
		return 1;

	fscanf(fp, "%d %d %d %d", &m, &n, &tm, &tn);

	for(i=0;i<n;i++)
		for(j=0;j<m;j++)
			fscanf(fp, "%d", &table[i+1][j+1]);

	tm++; tn++;

	fclose(fp);
	return 0;
}
```

```
inline int qpush(int n, int m, int d)
{
        queue[qfront][0] = n;
        queue[qfront][1] = m;
        queue[qfront][2] = d;
        qfront = (qfront+1)%MAXQ;

        return 0;
}

inline int qpop(int*n, int*m, int*d)
{
        *n = queue[qrear][0];
        *m = queue[qrear][1];
        *d = queue[qrear][2];
        qrear = (qrear+1)%MAXQ;

        return 0;
}

int process(void)
{
        int i, j, k;
        int cn, cm, cd;
        int tmp;

        for(i=0;i<n+2;i++)
                for(j=0;j<m+2;j++)
                        for(k=0;k<4;k++)
                                matrix[i][j][k] = INF;

        qfront = qrear = 0;
        matrix[tn][tm][0] = matrix[tn][tm][1] = matrix[tn][tm][2] = matrix[tn][tm][3]
0;
        qpush(tn, tm, 0);
        qpush(tn, tm, 1);
```

```
	qpush(tn, tm, 2);
	qpush(tn, tm, 3);

	while(qfront != qrear)
	{
		qpop(&cn, &cm, &cd);
		for(k=0;k<4;k++)
		{
			if(table[cn+dy[k]][cm+dx[k]] && k!=neg[cd])
			{
				tmp = matrix[cn][cm][cd] + (cd==k ? 0 : 1);
				if(matrix[cn+dy[k]][cm+dx[k]][k] > tmp)
				{
					matrix[cn+dy[k]][cm+dx[k]][k] = tmp;
					qpush(cn+dy[k], cm+dx[k], k);
				}
			}
		}
	}

	for(i=0;i<n+2;i++)
	{
		for(j=0;j<m+2;j++)
		{
			trace[i][j] = 0;
			for(k=1;k<4;k++)
				if(matrix[i][j][trace[i][j]] > matrix[i][j][k])
					trace[i][j] = k;
		}
	}
	return 0;
}

int output(void)
{
	FILE* fp;
```

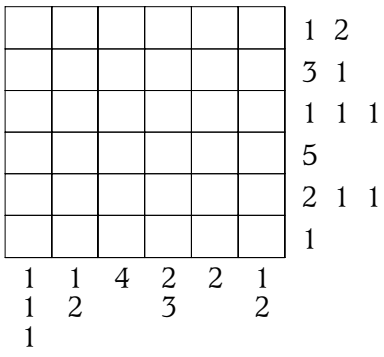
```
        if((fp = fopen(OUTFILE, "w")) == NULL)
                return 1;

        fprintf(fp, "%d\n", matrix[1][1][trace[1][1]]);

        fclose(fp);
        return 0;
}

int main(void)
{
        if(input())
                return 1;
        process();
        if(output())
                return 1;
        return 0;
}
```

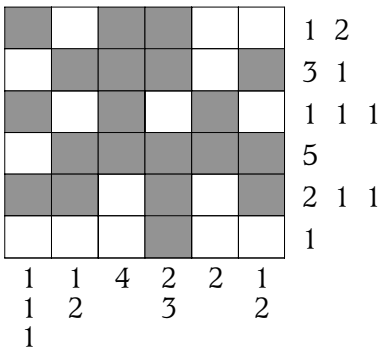
12 네모네모 로직

▸▸ 일반적으로 게임으로 널리 알려져 있지만 특별한 풀이 방법이 없는 네모네모 로직(노노그램) 게임을 컴퓨터를 이용해서 풀어보는 문제이다. 특별히 풀 수 있는 방법이 없기 때문에 모든 조건을 검사해 나가야 한다.

문제 (난이도 ☆☆☆☆★)

네모네모 로직(노노그램)이란 N×N의 빈칸이 주어지고 각각의 칸에 대한 가로 세로의 힌트가 주어지면 주어진 힌트들을 이용해서 검은색 칸을 채워 넣어 본래의 그림을 그려내는 문제이다.





각 가로줄의 오른쪽에 있는 숫자들은 그 줄에 나타나는 연결된 검은색 칸들의 그룹의 크기들이 순서대로 표시된 것이다. 이 숫자들이 의미하는 것은 그 숫자들이 있는 줄에 연속된 칸이 숫자만큼 순서대로 있다는 것이다. 예를 들면 첫 가로줄의 '1 2'는 이 줄에 1개의 검은색 칸으로 된 그룹과 2개의 연결된 검은색 칸으로 된 그룹이 차례로 나타남을 의미한다. 이때 임의의 흰색 칸이 각 그룹의 좌우에 위치할 수 있다. 가로와 세로 힌트를 잘 살펴서 그림을 완성하는 것이 게임의 목적이다.

프로그램의 이름은 "nemo"로 한다.

입력 형식

노노그램 크기 N(2≤N≤15)과 각 가로줄에 대한 정보, 각 세로줄에 대한 힌트가 차례로 주어진다. 입력 파일 이름은 “Input.txt"로 한다.
첫째 줄에는 테이블 N×N의 N이 주어진다.
다음 줄 부터 N개의 줄에 노노그램의 첫 번째 줄에서 마지막 줄까지의 행에 대한 힌트가 주어진다. 각 행의 첫 값은 행에 대한 힌트 개수이다.
다음 줄 부터 N개의 줄에 노노그램의 첫 번째 줄에서 마지막 줄까지의 열에 대한 힌트가 주어진다. 각 행의 첫 값은 열에 대한 힌트 개수이다.

출력 형식

출력 파일 이름은 “Output.txt"로 한다. N×N 테이블에서 검은색 칸은 1로 흰색 칸은 0으로 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

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

출력의 예(Output.txt)

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

2 문제의 배경 및 관련 이론

어떤 문제의 답을 구하기 위해 일반적으로 효율적인 알고리즘으로 알고 있는 시간복잡도가 O(n), O($n \log n$) 또는 O(n^2)의 경우로는 해결할 수 없어 주어진 경우의 수를 모두 검색해야 하는 경우를 종종 만날 수 있다.

가능한 경우의 수를 체계적으로 발생시켜서 하나씩 문제를 해결해 나가는 방법을 백트랙킹(backtracking)이라 한다. 이러한 백트랙킹은 일반적으로 시간복잡도가 높다.

우리는 시간복잡도가 O(n), O($n \log n$)이 되어야 좋은 알고리즘이고 O(n^2), O(n^3) 이상은 안좋은 알고리즘으로 생각하고 있지만 실질적으로 주어진 수행시간안에 문제를 해결할 수 있다면 이해하기 어렵게 작성한 시간복잡도 O($n \log n$)보다 누구나 봐도 명쾌하게 이해가 되게 만든 시간복잡도 O(n^{50})의 알고리즘이 더 뛰어난 것이다.

이렇게 완전검색을 해야만 해를 구할 수 있는 유형의 문제로 잘 알려진 "외판원(traveling salesmen)" 문제가 있다. “N개의 도시가 있을 때, 이 모두를 방문하여 연결하는 최단 경로를 구하는데 한번 방문한 도시는 다시 방문할 수 없다.”라는 문제로 아주 오랫동안 연구되어져 왔지만 N=1000이상인 경우에 대해서는 해결 방안을 아직 생각조차 하지 못하고 있다.

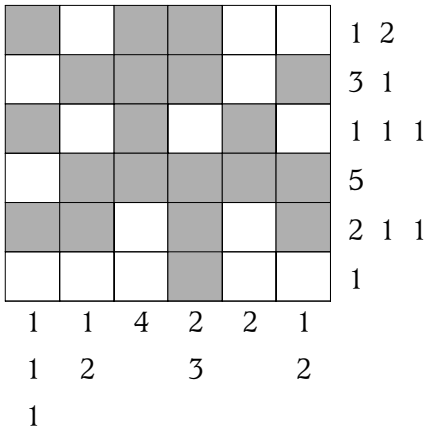
그래서 백트랙킹을 사용할 때에는 경우의 수를 현저히 줄일 수 있는 방법을 연구해보아야 한다. 이 방법으로 검색도중 불필요한 부분을 검색하지 않는 pruning을 하는 것이다. 즉 검색의 대상을 트리로 표현했을 때 특정 트리의 가지와 그에 연결된 모든 노드를 없앰으로써 더 이상 그 가지는 검색하지 않는 것이다.

이렇게 특정 문제를 해결하기 위해 검색을 제약할 수 있는 여러 복잡한 기법들이 개발되어 왔고 앞으로도 개발되겠지만 그 기준들이 얼마나 복잡하든지 백트랙킹 알고리즘은 수행시간이 지수시간(exponential time)이 걸린다는 것은 일반적으로 알려진 사실이다.

즉 검색트리내의 각 노드가 평균적으로 a개의 자식노드를 가지고 가능한 경우의 경로가 N이라면 그 트리에서의 노드 수는 a^N에 비례할 것이다.

3 문제 해설

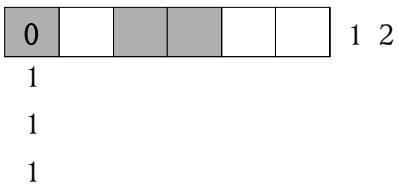
네모네모(노노그램) 문제를 풀기 위한 특별한 방법은 아직 알려져 있지 않다. 때문에 이 문제를 풀기 위해 모든 테이블을 다 검색해 보는 백트랙킹 방법을 생각해 보자.



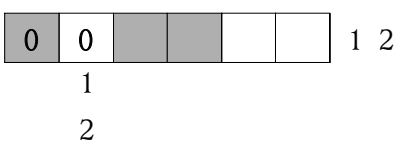
먼저 테이블 안의 셀들을 (0,0)->(0,1)->…->(1,0)->(1,1)->…->(5,5)까지 하나씩 선택하고 선택한 테이블의 값이 우측의 힌트와 밑의 힌트에 맞는 지 검사해 나간다.

검은색 칸은 1로 흰색 칸은 0으로 정한 후 첫 번째 행의 첫 셀부터 검사한다.

배열 (0,0)을 0(흰색)으로 정한 후 오른쪽과 밑의 힌트(1 2, 1 1 1)를 보면 배열 (0, 0)이 0이더라고 힌트의 정보대로 배열될 수 있음을 알 수 있다.



다음 배열 (0,1)의 값을 다시 0으로 정한 후 오른쪽과 밑의 힌트를 검사한다. 배열 (0,1)의 값이 0이더라도 힌트의 정보대로 배열 될 수 있음을 알 수 있다.



다음 배열 (0,2)로 넘어가서 값을 다시 0으로 정한 후 오른쪽과 밑의 힌트를 검사한다.

이때 배열 (0,2)의 값이 0이면 힌트의 정보대로 배열 될 수 없음을 알 수 있다.

힌트가 “1 2”은 하나의 셀이 검은색 칸이면 그 뒤에 하나 이상의 흰색 칸과 두 개의 검은색 칸이 와야 하기 때문에 아래와 같은 5가지의 경우 중에 하나이다.

[가로0-1] 1 0 1 1 0 0

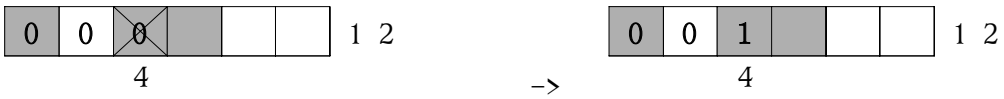
[가로0-2] 1 0 0 1 1 0

[가로0-3] 1 0 0 0 1 1

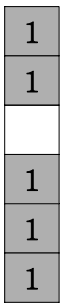
[가로0-4] 0 1 0 1 1 0

[가로0-5] 0 0 1 0 1 1

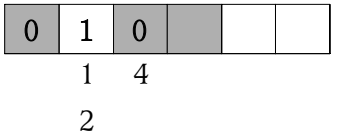
따라서 배열 (0,2)의 값은 1이어야 한다.



배열 (0,3)은 앞의 배열 (0,2)의 값이 1이고 힌트가 "1 2" 이므로 0이 올 수 밖에 없다. 그리고 오른쪽과 밑의 힌트(1 2, 2 3)보면 세로 힌트의 2 3이 나오는 경우는 아래 한가지 밖에 없다. 따라서 배열 (0, 3)의 값이 0이나 1이 와도 힌트와 맞지 않으므로 앞의 값이 잘못되었음을 알 수 있다.



다시 배열 (0,1)로 되돌아가 이번에는 값을 1을 넣어본다. 배열 (0,1)의 값이 1이고 힌트가 "1 2"이므로 배열 (0,2)의 값도 0이어야 하며 이때 가로 세로 힌트와 맞다.



배열 (0,3)의 값은 힌트 "2 3"에 의해 1이 와야 하므로 1로 정하면 배열 (0,4)는 힌트에 의해 1로 배열 (0,5)는 0의 값이 들어가야 한다.

0	1	0	1	1	0	1 2
	1	4	2	2	1	
	2		3		2	

이렇게 배열의 첫 번째 행을 검사하면서 수행 결과는 다음과 같다. 이것은 가로행의 힌트가 "1 2"때 나올 수 있는 3가지 경우의 한 가지 이므로 아직 이 결과가 정확히 맞다고 말할 수 없다.

0	1	0	1	1	0

다시 위의 결과를 가지고 다음 행을 다시 검사해 보자.

배열 (1,0)을 0으로 정한 후 오른쪽과 밑의 힌트(3 1, 1 1 1)를 보면 배열 (0, 0)이 0이면 가로 힌트 정보에는 맞지만 세로 정보 힌트에는 맞지 않는다.

첫 번째 열의 힌트 "1 1 1"에 맞는 경우를 보면 아래의 두 가지 경우가 올 수 있는데 배열 (0,0)의 값이 0이므로 [세로0-2]로 값이 되어야 한다.

[세로0-1] [세로0-2]

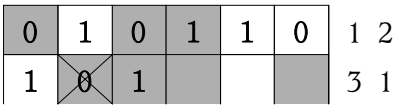
1
0
1
0
1
0

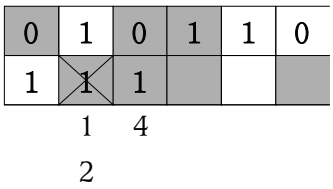
0
1
0
1
0
1

따라서 배열 (1,0)의 값은 1이 되어야 한다.

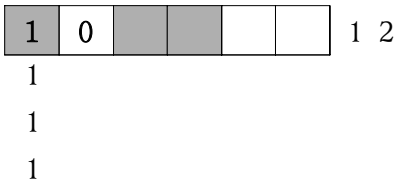
0	1	0	1	1	0	1 2
1						3 1
1						
1						
1						

배열 (1,1)의 값을 0으로 정한 후 힌트를 검사하면 세로 힌트는 맞지만 가로 힌트 "3 1"에는 맞지 않다 또한 배열 (1,1)의 값을 1로 정한 후 힌트를 검사하면 조건에 맞음을 알 수 있다. 따라서 가로 힌트에 따라 배열 (1,0) (1,1) (1,2)의 값이 모두 1이어야 하는데 이는 세로 힌트와 맞지 않는다.





따라서 다시 이전으로 되돌아가보면 배열 (0,0)이 잘못되었음을 알 수 있다. 배열 (0,0)의 값을 1로 하고 배열 (0,1)의 값을 0으로 정하고 다시 위의 방법대로 검사해 가면 최종 결과를 구할 수 있다.



위의 배열 하나하나를 검사하는 과정을 그래프로 표시하면 아래와 같다.

(0,0)

(0,1)

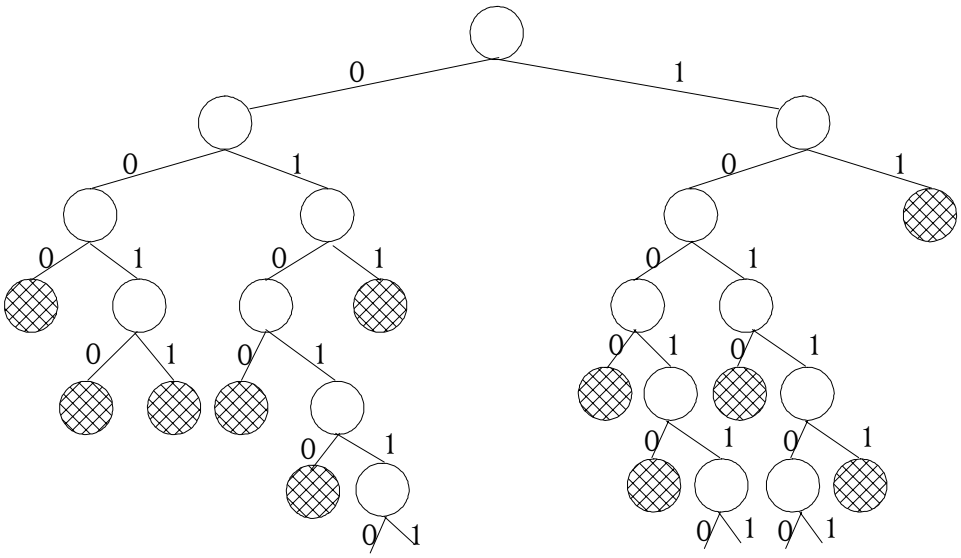
(0,2)

(0,3)

(0,4)

(0,5)

(1,0)



...

(5,5)

소스 코드

```
#include <stdio.h>

#define INFILE "input.txt"
#define OUTFILE "output.txt"

#define MAXN 15

int n, table[MAXN][MAXN];
int hseq[MAXN][MAXN]={0,}, vseq[MAXN][MAXN]={0,};
int hseqn[MAXN], vseqn[MAXN];

int input(void)
{
	FILE* fp;
	int i, j;

	if((fp = fopen(INFILE, "r")) == NULL)
		return 1;

	fscanf(fp, "%d", &n);

	for(i=0;i<n;i++)
	{
		fscanf(fp, "%d", &hseqn[i]);
		for(j=0;j<hseqn[i];j++)
			fscanf(fp, "%d", &hseq[i][j]);
	}

	for(i=0;i<n;i++)
	{
		fscanf(fp, "%d", &vseqn[i]);
		for(j=0;j<vseqn[i];j++)
			fscanf(fp, "%d", &vseq[i][j]);
	}
```

```
	fclose(fp);
	return 0;
}

int hcheck(int x, int y)
{
	int i;
	int p, l;

	p = l = 0;

	for(i=0;i<=y;i++)
	{
		if(table[x][i])
			l++;
		else if(l) {
			if(l != hseq[x][p])
				return 0;
			else
			{
				p++;
				l = 0;
			}
		}
	}

	if(i == n)
	{
		return ((p == hseqn[x] && l == 0) || (p == hseqn[x]-1 && l =
hseq[x][p]));
	}
	else
	{
		return (p <= hseqn[x] && l <= hseq[x][p]);
	}
}
```

```
int vcheck(int x, int y)
{
	int i;
	int p, l;

	p = l = 0;

	for(i=0;i<=x;i++)
	{
		if(table[i][y])
			l++;
		else if(l)
		{
			if(l != vseq[y][p])
				return 0;
			else
			{
				p++;
				l = 0;
			}
		}
	}

	if(i == n) {
		return ((p == vseqn[y] && l == 0) || (p == vseqn[y]-1 && l = vseq[y][p]));
	}
	else {
		return (p <= vseqn[y] && l <= vseq[y][p]);
	}
}

int output(void)
{
	FILE* fp;
	int i, j;
```

```
	if((fp = fopen(OUTFILE, "w")) == NULL)
		return 1;

	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			fprintf(fp, "%d ", table[i][j]);
		}
		fprintf(fp, "\n");
	}
	fclose(fp);

	return 0;
}

int shovel(int x, int y)
{
	if(x == n) {
		return 1;
	}
	else
	{
		table[x][y] = 0;

		if(hcheck(x,y) && vcheck(x,y))
			if(y == n-1 ? shovel(x+1,0) : shovel(x,y+1))
				return 1;
		table[x][y] = 1;
		if(hcheck(x,y) && vcheck(x,y))
			if(y == n-1 ? shovel(x+1,0) : shovel(x,y+1))
				return 1;
		return 0;
	}
}
```

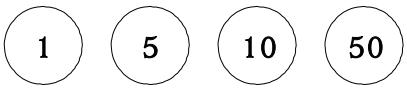
```
int process(void)
{
	if(shovel(0, 0))
	{
		output();
		return 0;
	}
	else
	{
		FILE* fp;
		if((fp = fopen(OUTFILE, "w")) == NULL)
			return 1;
		fprintf(fp, "error\n");
		fclose(fp);

		return 1;
	}
}

int main(void)
{
	if(input())
		return 1;

	if(process())
		return 1;

	return 0;
}
```

13 자동판매기 동전교환

▸▸ 주어진 금액을 동전으로 거슬러 줄 때 최소한의 동전 개수를 이용해서 거슬러 주는 문제이다. 동전은 일상적으로 사용하는 동전과 다르기 때문에 주의해서 알고리즘을 생각해야 한다.

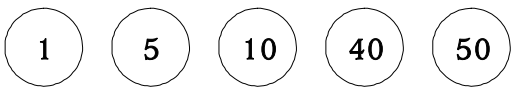
문제 (난이도 ★★☆☆☆)

작은 섬나라에는 동전이 50원, 10원, 5원, 1원짜리가 있다. 이 나라에서 사용하는 자동판매기는 지폐를 넣고 물건을 선택하면 잔돈을 동전으로만 거슬러주는데 손님의 편의를 위해 거슬러 주는 동전의 개수가 최소가 되도록 잔돈을 거슬러 준다.



즉 손님이 1000원 지폐를 자동판매기에 넣고 813원짜리 물건을 샀다면 거슬러 주어야 할 잔돈은 187원이 된다. 이때 거슬러 줘야할 동전을 50원짜리 3개, 10원짜리 3개, 5원짜리 개, 1원짜리 2개가 된다.

그런데 어느 날 국가 정책으로 40원짜리 동전을 새로 만들어 사용하기로 결정했다. 그래서 자동판매기가 기존의 거슬러 주는 방법으로 동전을 거슬러 주면 항상 최소 개수의 동전을 거슬러 줄 수 없는 문제가 발생하게 되었다.



이제 이러한 문제를 해결하기 위해 항상 적은 개수의 동전을 거슬러 줄 수 있는 프로그램을 작성하시오.

프로그램의 이름은 “coin"으로 한다.

입력 형식

입력 파일 이름은 “Input.txt"로 한다.
한 줄에 자동판매기에 넣은 지폐의 금액 W(0≤W≤100000)과 물건의 가격이 주어진다.

출력 형식

출력 파일 이름은 "Output.txt"로 한다.
각 줄에 각 동전별(50 40 10 5 1) 순서대로 사용한 동전의 개수를 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
1000 863
```

출력의 예(Output.txt)

```
1
2
0
1
2
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
100000 753
```

출력의 예(Output.txt)

```
1984
1
0
1
2
```

2 문제의 배경 및 관련 이론

그리드(greedy, 욕심쟁이) 알고리즘의 가장 기본적인 것은 현재 상황에서 가장 좋아 보이는 것을 택하는 것이다. 그 이유는 현재 상황에서 가장 좋은 것을 선택했을 때 전체적인 상황이 가장 좋게 되도록 하기 위함이다.

상당부분 그리드 알고리즘으로 최적 해를 구할 수 있다. 그러나 모든 경우에 그리드 방법이 통하는 것은 아니다. 또한 대부분의 휴리스틱의 문제의 경우 그리드 알고리즘을 사용하면 최적 해에 근접한 결과를 얻을 수 있다.

그리드 알고리즘을 사용하는 문제의 유형으로 동전교환문제, 냅색문제 등이 있다.

다이나믹(dynamic) 알고리즘의 원리는 모든 해를 다 찾아보되 한번 찾아본 해는 다시 찾지 않고 찾아진 결과를 가져다 보는 것이다.

다이나믹 알고리즘은 소스가 간결한 편이며 그리드 알고리즘과 함께 가장 많이 쓰이는 풀이법 중 하나이다. 그리고 다이나믹 방법은 모든 경우를 다 찾아보는 것이기 때문에 굳이 증명할 필요가 없다.

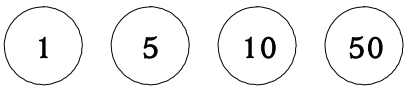
백트랙킹(backtracking, 퇴각검색)은 주어진 조건을 만족하는 최적해 또는 해들의 집합을 찾는 문제에 주로 사용하는 방법으로 주어진 상태공간을 트리로 구성해서 깊이우선 탐색(DFS) 등을 이용하여 상태공간을 체계적으로 탐색하는 방법이다.

이는 그리드 알고리즘 등으로는 효율적으로 해결할 수 없는 문제에 적용하여 최적해에 접근할 수 있으나 지수 시간의 복잡도를 갖는 단점이 있다. 이를 극복하기 위해 트리의 각 노드에서 더 이상 탐색이 필요 없는 부분을 제외시킴으로서 탐색할 범위를 줄이는 방법이 있다.

3 문제 해설

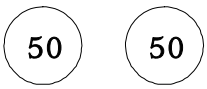
일반적으로 우리 일상에서 사용하고 있는 동전을 이용한 동전교환 문제는 그리드(Greedy) 알고리즘을 이용하여 최소 동전의 개수를 바꿀 수 있다.

만약 동전이 아래와 같이 1원, 5원, 10원, 50원의 동전을 사용하고 있을 때

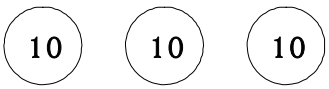


137원의 동전을 거슬러 줘야 한다면 사용하고 있는 1원, 5원, 10원, 50원의 동전 중 금액이 제일 큰 50원을 거슬러 줄 수 없을 때까지 거슬러 준 다음 10원짜리 동전을 같은 방법으로 거슬러 주고 다음은 5원짜리 1원짜리 동전 순으로 거슬러 주면 된다.

137원에서 50원짜리는 최대 2개를 거슬러 줄 수 있고 이때 37원이 남는다.

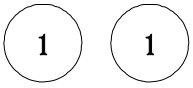


37원에서 10원짜리는 최대 3개를 거슬러 줄 수 있고 이때 7원이 남는다.



7원에서 5원짜리는 최대 1개를 거슬러 줄 수 있고 이때 2원이 남는다.

2원에서 1원짜리로 2개를 거슬러 주면 잔액은 0원이 된다.



따라서137원을 최소동전의 개수로 거슬러 주기위한 동전의 개수는 50원 2개, 10원 3개, 5원 1개, 1원 2개로 총 8개가 된다.

그리드 알고리즘을 사용하기 위해서는 큰 액수의 동전은 작은 액수의 동전의 “배수” 관계일 때에만 가능하다. 5원짜리가 2개 있다면 합이 10원으로 5원짜리 2개보다 10원짜리 한 개가 더 최소한의 동전 교환 개수가 되기 때문에 서로 배수 범위보다 큰 동전의 개수

를 갖지는 못한다. 즉 어느 동전을 가지고 있을 때 그 합이 한 단계 위의 동전만큼 된다면 그것은 최소한의 동전 개수라고 할 수 없다.

그러나 주어진 문제처럼 40원짜리 동전을 새로 만들어 냈다면 새로 만든 40원은 1원, 5원, 10원과의 배수관계는 성립하지만 50원과의 배수관계는 성립하지 않기 때문에 그리드 방법으로 해를 구하면 항상 최적해가 나오는 것은 아니다.

만약 137원을 동전으로 바꿀 때 그리드 알고리즘을 이용하면 50원짜리 2개, 40원짜리 0개, 10원짜리 3개, 5원짜리 1개, 1원짜리 2개로 총 8개의 동전이 필요하다. 그러나 달리 생각을 하면 50원짜리 1개, 40원짜리 2개, 10원짜리 0개, 5원짜리 1개, 1원짜리 2개로 총 6개가 필요함을 볼 수 있다.

이제 동전을 최소 개수로 거슬러줄 때 항상 최적 해를 구하기 위해 다이나믹(Dynamic) 방법을 생각해 보자.

다이나믹 방법은 동전을 거슬러줄 모든 금액을 다 살펴보아야 하지만 한번 살펴본 같은 금액을 찾기 위해 다시 반복해서 계산할 필요가 없는 특징이 있다.

예를 들어 1원짜리 동전과 5원짜리 동전이 있다고 할 때 11원을 만들려고 한다면 11원은 “6원”에 5원짜리 동전 1개를 추가해서 만들거나 아니면 “6원”에 1원짜리 동전 5개를 추가해서 만들 수 있다. 그렇다면 6원을 만들기 위해서는 최소한의 동전이 몇 개 필요한지 알아보자. 6원은 1원짜리 동전 6개나 1원짜리 동전 1개와 5원짜리 동전 1개로 만들 수 있는데 “1원+5원”으로 6원을 만드는 것이 최적이라는 것을 알 수 있다.

이 최적해인 6원(1원+5원)을 이용해 11원의 최소 동전 개수를 구할 때 다시 사용할 수 있다는 것이다.

각각 a, b, c, d원 짜리 동전이 여러 개 있다고 가정하자. 위 문제와 같은 방식으로 한다면 p원을 만들기 위해 a+(p-a)원이나, b+(p-b)원 등을 만들 수 있다.

즉 a+(p-a), b+(p-b), c+(p-c), d+(p-d) 등으로 나누어 생각할 수 있다. 반복되는 계산을 막기 위해 1원부터 p원까지 위의 방법으로 차례대로 만들어 나간다.

즉 m원을 만들기 위해, m-a, m-b, m-c, m-d원을 만들기 위해 필요한 최소 개수의 동전을 찾은 뒤 그 값에 +1(그 값에 a, b, c, d원을 추가시켰으므로)을 해주기만 하면 m원을 만드는데 필요한 최소 동전의 개수를 알 수 있는 것이다.

따라서,

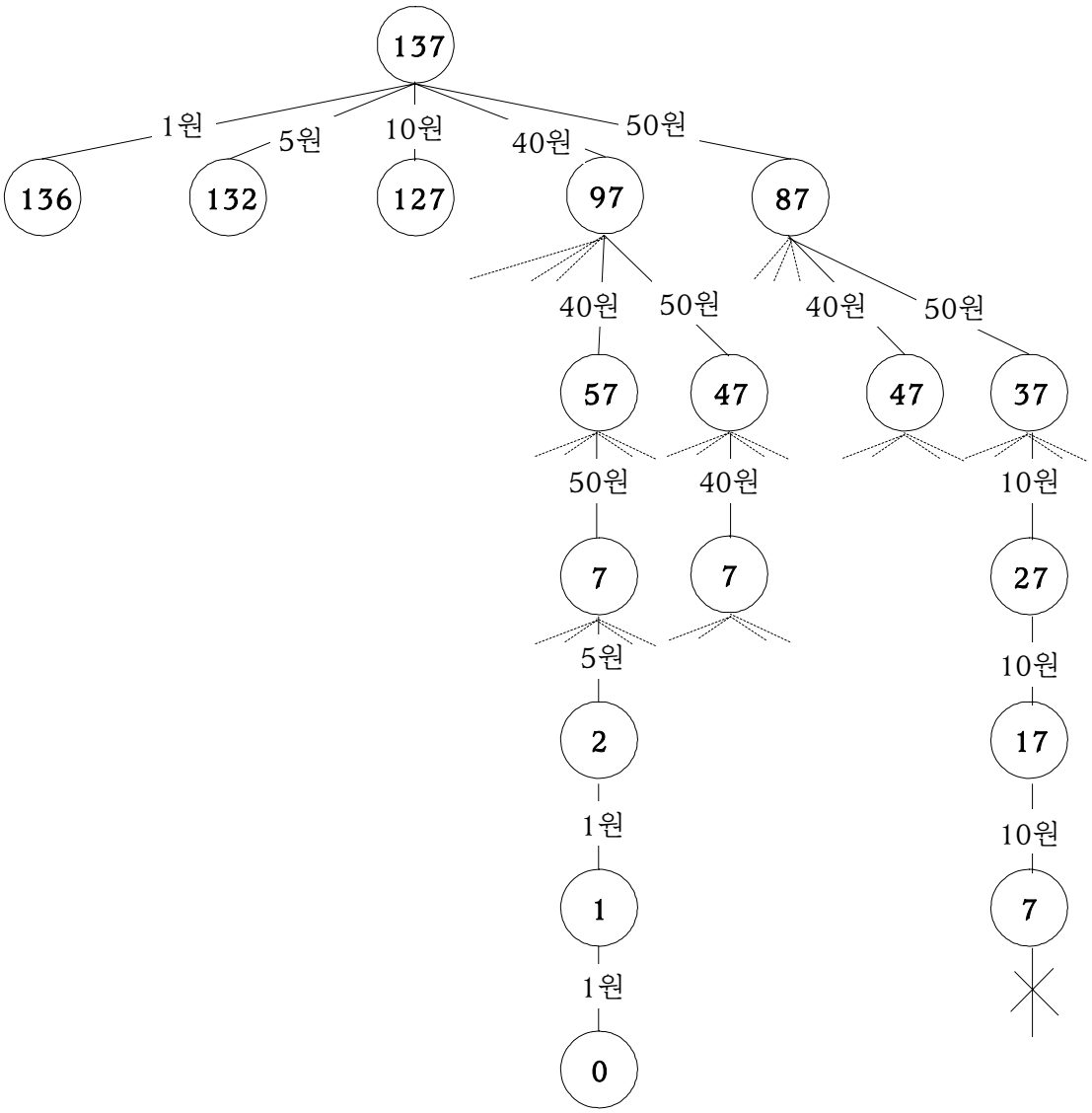
1원 1개 1원×1
…
4원 4개 1원×4
5원 1개 5원×1 ; 4(5-1)원, 0(5-5)원을 만들기 위한 최소 동전의 개수가 각각 4원이 4개, 0원이 0개 이므로 여기에 +1을 해주면 5개와 1개가 나온다. 1개가 최소 동전의 개수이므로 5원을 만들 수 있는 최소 동전의 개수는 1개이다.
6원 2개 1원×1, 5원×1
7원 3개 1원×2, 5원×1
…
10원 1개 10원×1 ; 9(10-1)원, 5(10-5)원, 0(10-10)원을 만들기 위한 최소 동전의 개수가 각각 9원이 5개, 5원이 1개, 0원이 0개 이므로 여기에 +1을 해주면 6개, 2개, 1개가 나온다. 1개가 최소 동전의 개수이므로 10원을 만들 수 있는 최소 동전의 개수는 1개이다.
…
80원 2개 40원×2 ; 79(80-1)원, 75(80-5)원, 70(80-10)원, 40(8-40)원, 30(80-50)원을 만들기 위한 최소 동전의 개수가 각각 8개, 4개, 3개, 1개, 3개이므로 여기에 +1을 해서 가장 적은 동전의 개수는 2개이다.
…

이제 다른 방법으로 퇴각검색을 생각해보자.

퇴각검색은 모든 가능한 경우를 검색하는 방법으로 퇴각검색의 효율을 높이기 위해서는 먼저 그리드 알고리즘으로 동전의 개수의 상한을 설정하고 그 이후에 발견된 해 중 가장 좋은 해를 결정해야한다.

만약 동전의 종류가 1원, 5원, 10원, 40원, 50원이 있을 때 137원을 최소 동전의 개수로 바꾼다면 먼저 137원에서 동전 1원, 5원, 10원, 40원, 50원을 하나씩 바꿀 때의 5가지 경우로 나누어 생각해 본다. 그런 다음 다시 136원, 132원, 127원, 97원, 87원 중에서 다시 1원, 5원, 10원, 40원, 50원을 하나씩 바꿀 때의 각각 5가지 경우를 금액이 0원이 될 때까지 고려하는 방법이다.

아래 그래프는 이러한 절차를 보여준다.



퇴각검색은 모든 가능한 경우를 다 살펴보는 경우이기 때문에 시간이 다른 알고리즘에 비해 많이 소요된다. 때문에 시간을 단축하려면 간단한 그리드 알고리즘을 이용해 해를 구한 다음 이 해를 기준점으로 해서 현재 검색하고 있는 방향의 퇴각검색의 해가 그리드 알고리즘의 결과 해보다 클 때에는 더 이상 그 방향을 검색하지 않는 pruning 기법을 이용하여 다른 방향을 검색해야 한다.

소스 코드

[1. 다이나믹 방법]

```
#include <stdio.h>
#include <memory.h>

#define MAX   100000

#define INPUTFILE   "input.txt"
#define OUTPUTFILE "output.txt"

int change, n, m;
int res[MAX+1][5];

void input(void)
{
        FILE *fp=fopen(INPUTFILE, "rt");

        fscanf(fp, "%d %d", &n, &m);

        change=n-m;

        fclose(fp);
}

void dynamic(void)
{
        int dyn[MAX+1], mon[5]={50, 40, 10, 5, 1};
        int i, j;

        memset(dyn, 127, sizeof(dyn));

        dyn[0]=0;

        memset(res[0], 0, sizeof(int)*5);
```

```
	for (i=1; i<=change; i++)
		for (j=0; j<5; j++)
			if (mon[j]<=i && dyn[i-mon[j]]+1<dyn[i])
			{
				dyn[i]=dyn[i-mon[j]]+1;
				memcpy(res[i], res[i-mon[j]], sizeof(int)*5);
				res[i][j]++;
			}
}

void output(void)
{
	FILE *fp=fopen(OUTPUTFILE, "wt");

	fprintf(fp, "%d\n %d\n %d\n %d\n %d\n", res[change][0], res[change][1],
res[change][2], res[change][3], res[change][4]);
	fclose(fp);
}

int main(void)
{
	input();
	dynamic();
	output();
	return 0;
}
```

[2. 백트랙킹 방법]

```
#include <stdio.h>

#define INPUTFILE  "input.txt"
#define OUTPUTFILE "output.txt"

int mincoin;

int input(void)
{
        FILE *fp;
        int n, m;

        if((fp = fopen(INPUTFILE, "r")) == NULL)
                return -1;
        fscanf(fp, "%d %d", &n, &m);
        fclose(fp);

        return n-m;
}

void backtrack(int price, int coin)
{
        printf("* %d\n", coin);
        if(coin + price/50 >= mincoin || price < 0)
                return;
        else
        {
                backtrack(price - 1, coin + 1);
                backtrack(price - 5, coin + 1);
                backtrack(price - 10, coin + 1);
                backtrack(price - 40, coin + 1);
                backtrack(price - 50, coin + 1);
        }
}
```

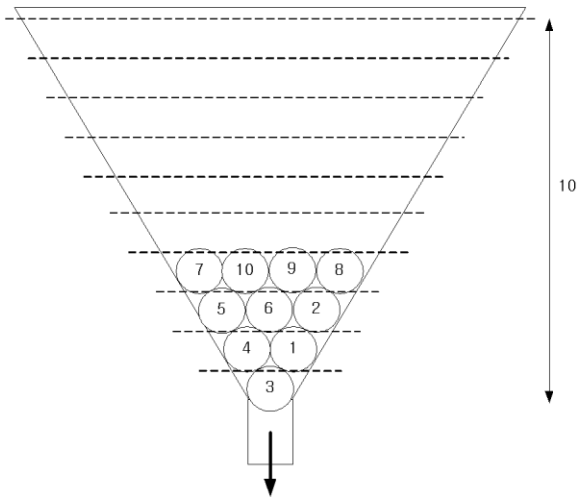
```
int process(int change)
{
	//그리드 알고리즘을 이용해 일반해를 구한다.
	mincoin = change / 50 + (change % 50) / 40 + ((change % 50) % 40) / 10 + (((change % 50) % 40) % 10) / 5 + ((((change % 50) % 40) % 10) % 5) / 1;

	printf("%d %d\n\n", change, mincoin);

	backtrack(change, 0);

	return mincoin;
}

int output(int result)
{
	FILE *fp;

	if((fp = fopen(OUTPUTFILE, "w")) == NULL)
		return 0;
	fprintf(fp, "%d\n", result);
	fclose(fp);

	return 1;
}

int main(void)
{
	int n;

	if((n = input()) < 0)
		return 1;
	if(!output(process(n)))
		return 1;
	return 0;
}
```

14 통에서 공 꺼내기

▸▸ 깔대기 안의 공이 무게에 따라 빠져나오는 순서를 정해주는 문제로 주어진 문제에 따라 어떠한 데이터 구조를 사용할지를 결정하는 문제이다.

문제 (난이도 ★☆☆☆☆)

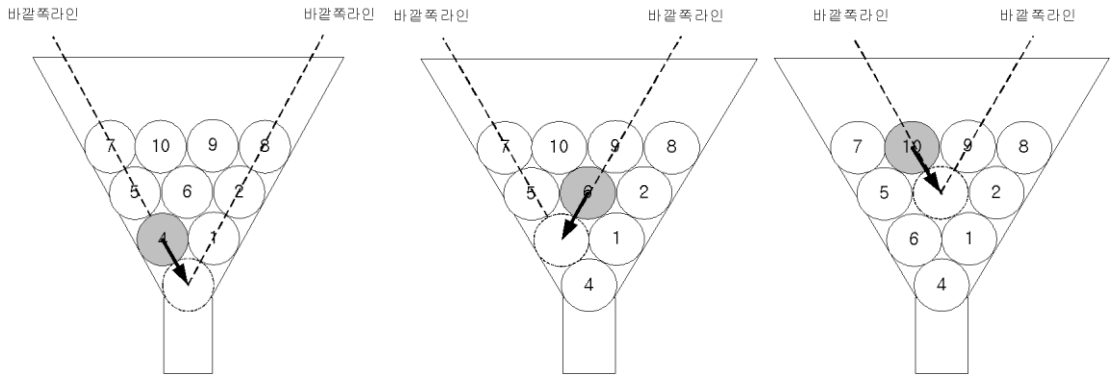
깊이가 10인 깔대기 모양의 큰 통이 있다. 우리는 이 통에 크기는 같지만 무게가 다른 공들을 각각 차례로 넣었을 때 그 공들이 어떤 순서로 나오는지 보려고 한다.



공에는 번호가 적혀있고, 이 번호는 각 공의 무게를 나타낸다. 공을 아래에서 하나씩 꺼내면 위에 있는 공들이 그 무게로 인하여 내려와서 그 자리를 메우게 된다.

공 하나가 꺼내어지면 비어 있는 자리에는 그 자리를 중심으로 두 개의 바깥쪽 라인의 공들의 무게 합이 큰 쪽의 공이 비어있는 자리를 메우게 된다. 만약 두 바깥쪽 라인 공들

의 무게가 같다고 하면 오른쪽 라인의 공이 비어있는 자리를 메우게 된다.



한편, 공을 넣을 때는 아래에서부터 차례로 쌓이며, 왼쪽에서 오른쪽으로 쌓여 나간다.

프로그램의 이름은 “ball”로 한다.

입력 형식

입력파일의 이름은 “Input.txt"로 한다.
첫 번째 줄에는 이 통에 넣는 공의 개수 N(1≤N≤55)의 값을 입력한다.
두 번째 줄에는 넣는 차례대로 각 공의 무게(1≤공의 무게≤99, 공의 무게는 정수 값)를 입력한다.

출력 형식

출력파일의 이름은 "Output.txt"로 한다.
두 번째 줄에는 공이 꺼내진 순서대로 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

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

출력의 예(Output.txt)

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

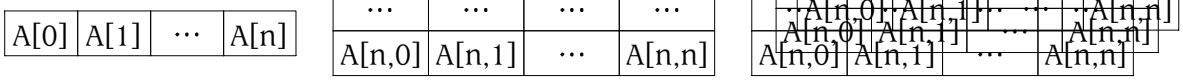
2 문제의 배경 및 관련 이론

이 문제는 주어진 데이터를 처리하는데 있어 어떤 자료구조를 사용할지 결정을 잘 하면 그다지 어렵지 않은 문제이다.

알고리즘에서 자주 사용되는 기본 구조인 배열(array)은 선형 리스트의 한 종류로써 대부분의 프로그래밍 언어에서 제공하는 기본적인 구조이다. 배열은 연속된 메모리 공간을 차지하는 동일한 성격의 데이터 형들의 유한 집합으로 인덱스(index)와 값(value)의 쌍으로 구성된 정적인(원소의 수가 정해진) 자료 구조이다.

배열은 모든 컴퓨터에서의 기억장치와 직접적인 대응이 된다는 점에서 기본적인 자료구조이다. 기계어에서 기억장소의 단어 내용을 검색하기 위해서 번지를 제공한다. 이와 같이 배열 인덱스에 대응되는 기억 장소 번지와 함께 배열로서 전체 컴퓨터 기억 장소를 생각하게 된다.

즉 배열은 리스트 상의 위치를 나타내는 인덱스 각각에 대응되는 하나의 값을 갖는다. 사용하는 인덱스에 따라 1차원, 2차원, 3차원, 다차원 배열이 있으며 배열로 표현할 수 있는 데이터 구조에는 스택, 큐, 그래프 등이 있다.

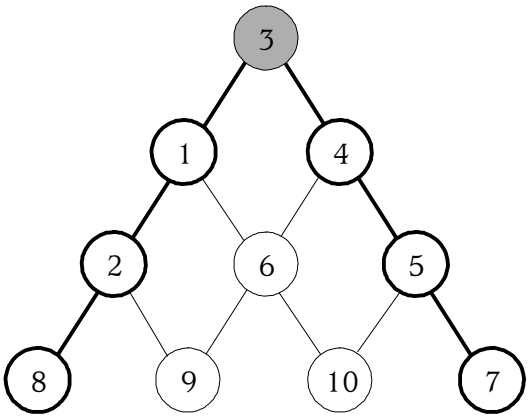


3 문제 해설

이 문제는 입력된 데이터를 어떠한 형태로 표현할지를 선택하는 것이 중요하다. 배열이나 트리, 리스트, 그래프 등에서 표현할 자료 형태를 올바로 결정을 하면 쉽게 알고리즘을 구성할 수 있다.

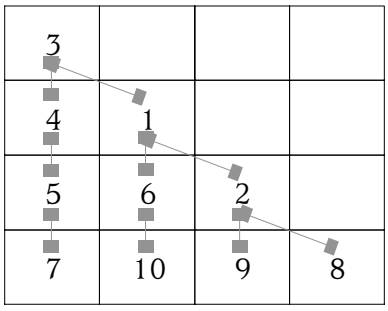
문제에서 각 공들은 깔대기 모양의 큰 통에 층별로 1개, 2개, …, 10개까지 총 55개의 공이 들어갈 수 있다.

이를 그래프로 구성하면 다음과 같다. 여기에서 어느 한 노드와 관계되는 것은 선택된 노드와 연결된 좌우측 노드들이다. 만약 3이 선택된 노드라면 노드 1, 노드 2, 노드 8과 노드 4, 노드 5, 노드 7이 관련된 노드라 할 수 있다.

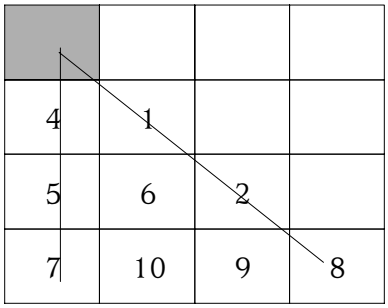


이렇게 데이터를 그래프로 표현할 수 있지만 그래프로 표현하면 데이터의 교환이나 관계된 노드들끼리의 비교할 때 배열을 사용하는 것보다 어려움이 많다.

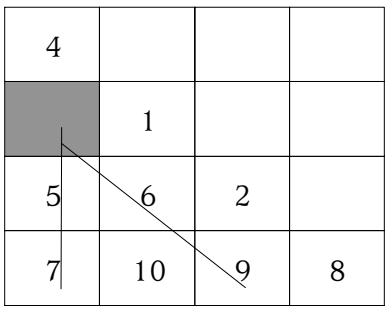
주어진 자료 10개의 공이 들어가 있는 깔대기를 상하로 뒤집어 배열로 표현하면 아래와 같이 표현할 수 있다.



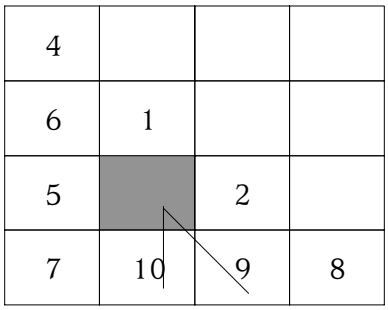
가장 최상위에 있는 공 3 즉 배열 (0, 0)에 있는 값을 빼내면 밑의 공 4와 공 1 즉 배열 (1, 0)과 (1, 1) 중 하나가 배열 (0, 0)의 값에 옮겨지게 된다. 이때 두 값 중 선택되는 값은 배열 (0, 0)에서 밑으로 수직선에 해당하는 값들의 합과 배열 (0, 0)에서 대각선에 해당하는 값들의 합 중 큰 값(합이 같을 경우 오른쪽에 있는 값)이 선택된다.



배열 (0, 0)에서 밑으로 수직선상에 있는 값들을 합하면 16이 되고 배열 (0, 0)에서 대각선상에 있는 값들을 합하면 11이 된다. 배열 (0, 0)에서 수직선상의 합이 더 크므로 배열 (1, 0)에 있는 값 4를 배열 (0, 0)으로 옮긴다.
이때 옮겨진 자리 배열 (1, 0)의 값이 비게 되므로 다시 배열 (1, 0)과 바로 연결된 배열 (2, 0)과 배열 (2, 1)의 값 중 하나를 선택해 배열 (1, 0)으로 옮겨야 한다.



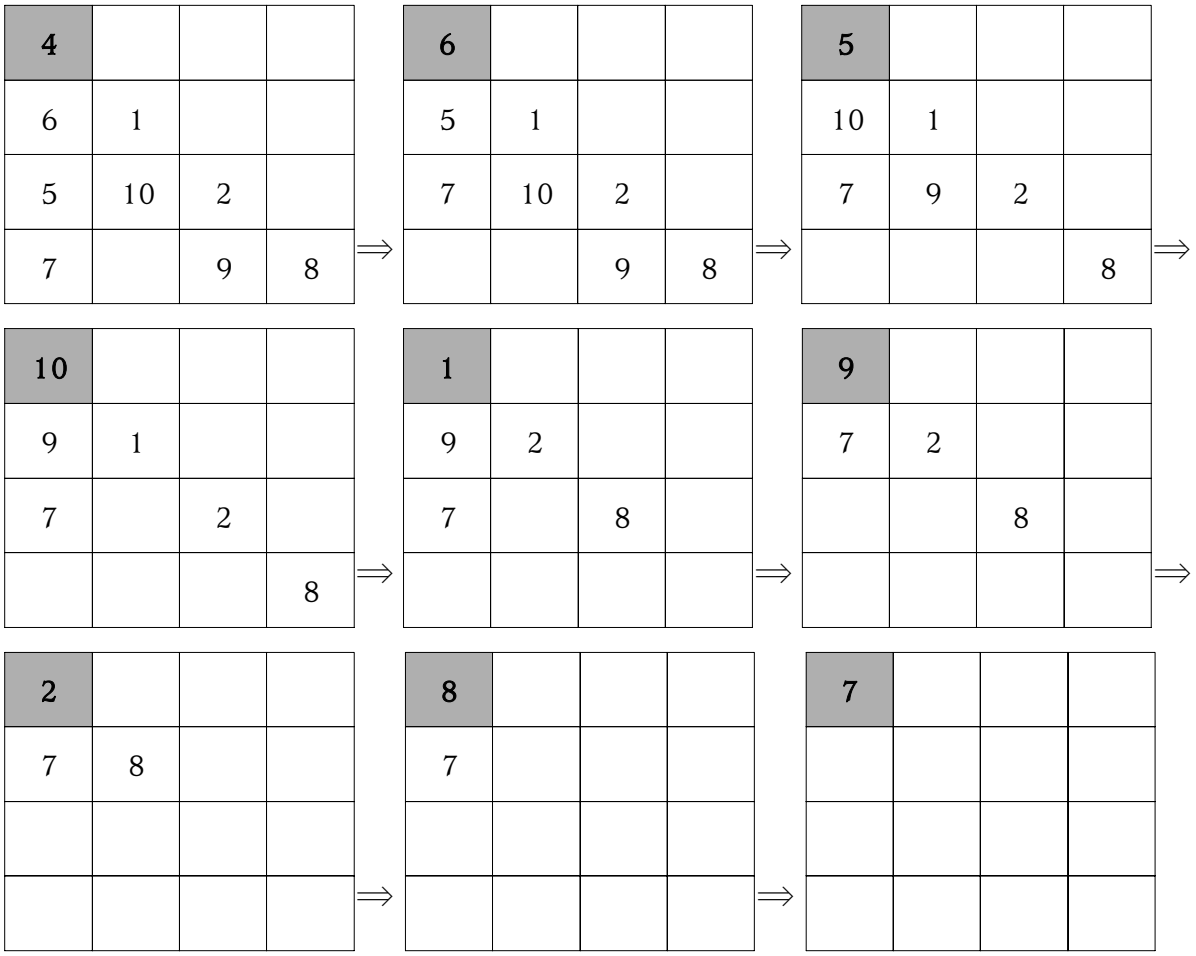
배열 (1, 0)에서 밑으로 수직선상에 있는 값들을 합하면 12가 되고 배열 (1, 0)에서 대각선상에 있는 값들을 합하면 15가 된다. 배열 (1, 0)에서 대각선상의 합이 더 크므로 배열 (2, 1)에 있는 값 6을 배열 (1, 0)으로 옮긴다.



역시 같은 방법으로 배열 (2, 1)의 자리를 메우기 위해 수직선과 대각선의 합을 구해 수직선에 있는 배열 (3, 1)의 값 10을 배열 (2, 1)의 자리로 옮긴다.

4			
6	1		
5	10	2	
7		9	8

이제 위와 같은 과정을 계속 반복하면 된다.
배열 (0, 0)에 있는 값을 빼내고 그 자리를 채우면 과정을 보면 아래와 같다.



따라서 깔대기 밖으로 공이 나오는 순서는 3-4-6-5-10-1-9-2-8-7이 된다.

소스 코드

```
#include <stdio.h>

#define INPUTFILE “input.txt"
#define OUTPUTFILE "output.txt"

#define MAXBALL 55
#define SPACESIZE 10

int matrix[SPACESIZE][SPACESIZE];
int weight1[SPACESIZE][SPACESIZE];
int weight2[SPACESIZE][SPACESIZE];
int n, path[MAXBALL];

int input(void)
{
        FILE *fp;
        int i, j, row, col;

        for(i=0;i<SPACESIZE;i++)
                for(j=0;j<=i;j++)
                        matrix[i][j] = weight1[i][j] = weight2[i][j] = 0;

        if((fp = fopen(INPUTFILE, "r")) == NULL)
                return 1;

        fscanf(fp, "%d", &n);

        for(i=row=col=0;i<n;i++)
        {
                fscanf(fp, "%d", &matrix[row][col]);
                if(row == col)          {
                        row++;
                        col = 0;
                }
```

```
                else
                {
                        col++;
                }
        }
        fclose(fp);
        return 0;
}

void process(void)
{
        int i, j, r, c;
        int row, col;

        for(i=0;i<SPACESIZE;i++)
                weight1[SPACESIZE-1][i] = weight2[SPACESIZE-1][i] =
matrix[SPACESIZE-1][i];

        for(i=SPACESIZE-2;i>=0;i--)
                for(j=i;j>=0;j--)
                {
                        weight1[i][j] = weight1[i+1][j] + matrix[i][j];
                        weight2[i][j] = weight2[i+1][j+1] + matrix[i][j];
                }

        for(i=0;i<n;i++)
        {
                path[i] = matrix[0][0];
                row = col = 0;
                for(j=0;j<SPACESIZE-1;j++)
                {
                        if(weight1[row][col] == 0 && weight2[row][col] == 0)
                                break;
                        if(weight1[row][col] > weight2[row][col])
                        {
                                matrix[row][col] = matrix[row+1][col];
```

```
                        matrix[++row][col] = 0;

                        for(r=row,c=col;r>=c;r--)
                                weight1[r][c] = weight1[r+1][c] + matrix[r][c];

                        for(r=row,c=col;c>=0;r--,c--)
                                weight2[r][c]   =   weight2[r+1][c+1]   +
matrix[r][c];

                        for(r=row-1,c=col;c>=0;r--,c--)
                                weight2[r][c]   =   weight2[r+1][c+1]   +
matrix[r][c];
                }
                else
                {
                        matrix[row][col] = matrix[row+1][col+1];
                        matrix[++row][++col] = 0;

                        for(r=row,c=col;r>=c;r--)
                                weight1[r][c] = weight1[r+1][c] + matrix[r][c];

                        for(r=row-1,c=col-1;r>=c;r--)
                                weight1[r][c] = weight1[r+1][c] + matrix[r][c];

                        for(r=row,c=col;c>=0;r--,c--)
                                weight2[r][c]   =   weight2[r+1][c+1]   +
matrix[r][c];
                }
        }
        matrix[row][col] = 0;
    }
}

int output(void)
{
    FILE *fp;
    int i;
```

```
	if((fp = fopen(OUTPUTFILE, "w")) == NULL)
		return 1;

	for(i=0;i<n;i++)
		fprintf(fp, "%d ", path[i]);
	fclose(fp);

	return 0;
}

int main(void)
{
	if(input())
		return 1;

	process();

	if(output())
		return 1;

	return 0;
}
```

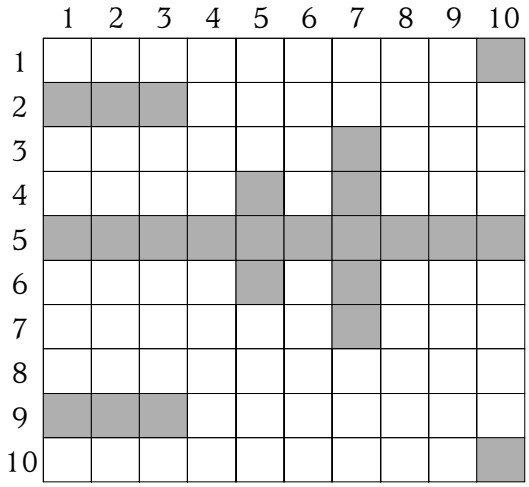
15 같은 모양 찾기

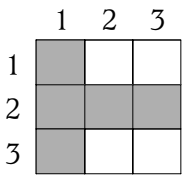
▸▸ 주어진 데이터와 찾고자 하는 패턴이 주어질 때 주어진 데이터 내에서 일치하는 패턴이 나오는 횟수를 묻는 문제이다.

문제 (난이도 ★★★★☆)

아래와 같이 모눈종이에 각각의 칸들이 칠해져 있는 그림이 있을 때 모눈종이에서 찾고 싶은 패턴의 모양이 몇 개가 있는지를 검사하려고 한다.

이 때, 찾고자 하는 모눈종이의 크기 M(0≤M≤100)과 패턴의 크기 P(0≤M≤100)은 주어진다.





찾고자 하는 패턴의 모양은 회색과 흰색으로 칠해지고, 패턴은 90°, 180°, 270°로 회전할 수 있으나 이때 회전한 패턴의 모양은 서로 다른 것으로 인식한다. 또한 회색으로 칠해진 칸은 패턴 매치 검사 시 반복되어 사용될 수 있다.

프로그램의 이름은 “pattern”로 한다.

입력 형식

입력 파일의 이름은 “Input.txt"로 한다.
첫 번째 줄에는 모눈종이이의 크기 M(0≤M≤100)이 주어진다.
두 번째 줄 부터 M 줄까지는 모눈종이에 그린 그림을 칠한 칸은 1로, 칠하지 않은 칸은 0으로 모눈종이의 줄별로 입력한다.
다음 줄에는 패턴의 크기 P(0≤P≤100)가 주어진다.
다음 줄부터 P개의 줄에 걸쳐 찾고 싶은 패턴의 모양이 주어진다. 모양이 있는 부분만 1로 입력하고 나머지는 0으로 처리한다.

출력 형식

출력 파일 이름은 “Output.txt"로 한다.
출력은 찾고 싶은 패턴의 모양이 모눈종이에 그린 그림에 몇 개가 있는지 그 개수를 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
10
0000000001
1110011100
0100101000
0100101000
1111111111
0000101000
0000001000
0010010000
1111110000
0010010001
3
100
111
100
```

출력의 예(Output.txt)

```
2
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
10
0110000001
0110000110
1001000110
0110111000
0110111011
0010101110
1111001100
1111101110
1111100000
0000000001
3
110
110
001
```

출력의 예(Output.txt)

```
4
```

2 문제의 배경 및 관련 이론

매칭은 기본적으로 스트링(string) 매칭과 패턴(pattern matching) 매칭과 더 나아가서 문자인식, 음성인식, 화상인식, DNA 염기 서열 분석 등의 인공지능 분야와 사원의 선발 문제나 결혼 문제 등 서로 간의 연결 관계에서 최적의 적임자를 찾는 문제로 확대할 수 있다.

스트링 매칭이나 패턴 매칭 문제는 주어진 데이터 안에서 주어진 패턴이 어느 곳에 어느 정도 빈번하게 나타나는지를 알아내는 문제이다. 즉 스트링 매칭인 경우 텍스트 T[1..n]에서 패턴 P[1..m]과 일치하는 모든 부분을 찾아내는 문제이다.

스트링 매칭 문제해결을 위한 방법은 아래와 같다.

① 직선적(brute-force) 스트링 매칭 방법 : 텍스트의 각 문자별 위치에서 패턴 일치가 발생하는 지를 전부 조사하는 방법이다.
② 라빈-카프(Rabin-Karp) 방법 : 크기가 m인 부분 스트링들을 해싱기법을 사용해 해시표에 키로 저장하는 방법으로 모든 스트링들을 해시 표에 저장할 필요는 없고, 해시값을 계산하여 찾고자 하는 패턴의 해시 값과 일치하는지 조사만 하면 된다.
③ 크누스-모리스-프랫(KMP) 방법 : 문자를 비교하는 과정에서 일치하지 않는 경우가 발생하면 다음 비교위치를 결정할 때 앞에서 비교한 정보를 이용하는 방법이다.
④ 보이어-무어(BM) 방법 : 패턴이 텍스트에 정렬된 각 위치에서 문자의 비교를 좌측에서 우측이 아니라 우측에서 좌측으로 행해나가는 방법으로 good suffix method와 bad character method를 도입하여 성능을 개선시켰다.

각각의 스트링 매칭의 성능을 비교하면 다음과 같다.

스트링 매칭	수행시간	
	평 균	최 악
직선적 알고리즘	O(nm)	
라빈-카프 알고리즘	O(n+m)	O(nm)
크누스-모리스-프랫 알고리즘	O(n+m)	
보이어-무어 알고리즘	$O((n-m+1)m+\|\Sigma\|)$	

3 문제 해설

만일 모눈종이의 크기가 10×10이고 패턴이 3×3이라면 이것을 배열로 표현한 것이 아래와 같다.

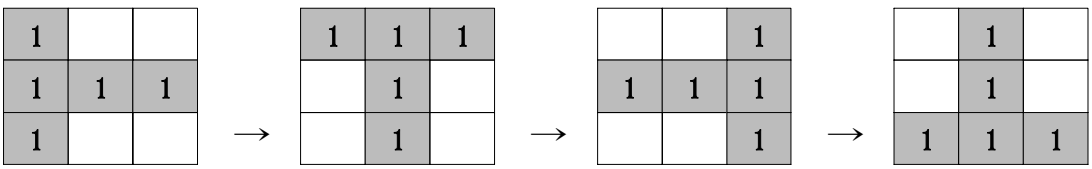
(0,0)	0	0	0	0	0	0	0	0	0	1	(0,9)
	1	1	1	0	0	1	1	1	0	0	
	0	1	0	0	1	0	1	0	0	0	
	0	1	0	0	1	0	1	0	0	0	
(4,0)	1	1	1	1	1	1	1	1	1	1	(4,9)
(5,0)	0	0	0	0	1	0	1	0	0	0	(5,9)
	0	0	0	0	0	0	1	0	0	0	
	0	0	1	0	0	1	0	0	0	0	
	1	1	1	1	1	1	0	0	0	0	
(9,0)	0	0	1	0	0	1	0	0	0	1	(9,9)

1	0	0
1	1	1
1	0	0

3×3 패턴을 10×10의 모눈종이 배열 (0, 0)부터 차례로 전부 검사해 나간다.

0	0	0	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0

또한 패턴을 비교할 때 패턴이 회전이 가능하므로 패턴을 시계방향으로 90°, 180°, 270° 회전하면 다음과 같다. 하나의 모눈종이 배열 위치에서 원래의 패턴과 회전한 패턴 총 4번의 패턴 검사를 해야 한다.

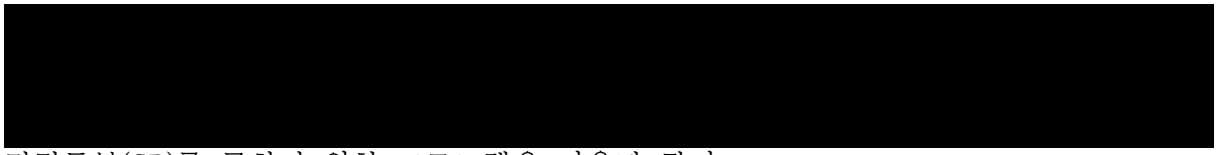


그러나 이러한 비교 방법은 모눈종이의 값 전부와 패턴을 일일이 비교해 보기 때문에 수행시간이 오래 걸린다. 수행시간을 단축하기 위해 스트링 패턴 알고리즘

중 KMP(크누스-모리스-프랫) 알고리즘을 이용해 보자.

KMP 알고리즘은 문자를 비교하는 과정에서 일치하지 않는 경우가 발생하면 다시 비교할 위치를 결정할 때 이미 비교한 패턴에 있는 정보를 활용하여 문서의 문자 위치를 나타내는 변수 i를 결정하는 것으로 비교회수를 줄일 수 있다.

KMP알고리즘은 반복되는 문자들이 많은 스트링에서 반복되는 문자들이 많은 패턴을 찾을 경우 유리하며 O(m+n)의 시간 복잡도를 갖는다.
하지만 여기에서의 문제는 하나의 행으로 이루어진 스트링이 아니라 2차원배열로 이루어져 있기 때문에 시간 복잡도는 O(m+n) 보다는 복잡해진다.
KMP 알고리즘을 이용하려면 먼저 패턴의 진접두부(SP)를 구해야 한다. 만약 패턴이 아래와 같다면



진접두부(SP)를 구하기 위한 프로그램은 다음과 같다.

```
//패턴의 최대 접두부 구하기
void preKmp(char *p, int m, int kmpNext[])
{
    int i, j;

    i = 0;
    j = kmpNext[0] = -1;
    while (i < m) {
        while (j > -1 && p[i] != p[j])
            j = kmpNext[j];
        i++;
        j++;
        if (p[i] == p[j])
            kmpNext[i] = kmpNext[j];
        else
            kmpNext[i] = j;
    }
}
```

결과 진접두부(SP) 값은 아래와 같다.

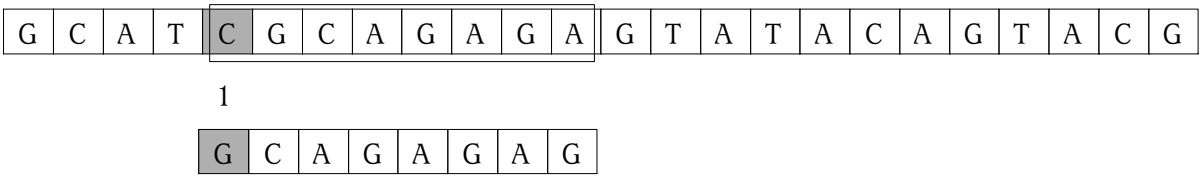
0	1	2	3	4	5	6	7	8
G	C	A	G	A	G	A	G	
-1	0	0	-1	1	-1	1	-1	1

아래와 같은 문자열이 주어지고 문자열 가운데 패턴을 찾는 과정을 보자.

첫 단계

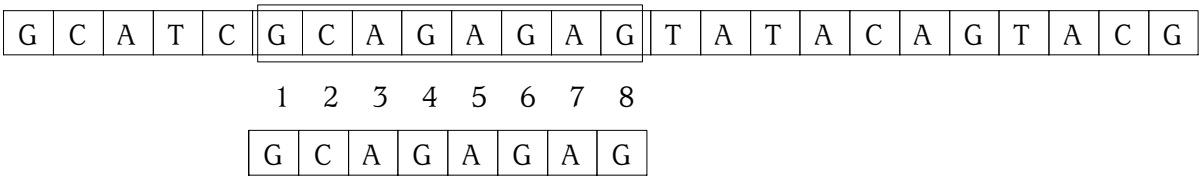
패턴의 4번째 위치(배열의 첨자 i=3)에서 불일치가 발생했으므로 다음 비교할 위치를 결정하기 위해 i-SP[i]를 수행하여 3-(-1)=4의 결과를 얻어 4칸 오른쪽으로 이동해 비교를 시작한다.

두 번째 단계



패턴의 1번째 위치(배열의 첨자 i=0)에서 불일치가 발생했으므로 다음 비교할 위치를 결정하기 위해 i-SP[i]를 수행하여 0-(-1)=1의 결과를 얻어 1칸 오른쪽으로 이동해 비교를 시작한다.

세 번째 단계



패턴과 문자열이 일치하므로 다음 비교할 위치(배열 첨자 i=8)를 결정하기 위해 i-SP[i]를 수행하여 8-(1)=7의 결과를 얻어 7칸 오른쪽으로 이동해 비교를 시작한다.

네 번째 단계

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G

1 2

G	C	A	G	A	G	A	G

패턴의 2번째 위치(배열의 첨자 i=1)에서 불일치가 발생했으므로 다음 비교할 위치를 결정하기 위해 i-SP[i]를 수행하여 1-(0)=1의 결과를 얻어 1칸 오른쪽으로 이동해 비교를 시작한다.

이러한 방법으로 패턴이 마지막 텍스트와 비교할 때까지 반복하면 문자열에서 패턴이 나타나는 빈도를 구할 수 있다.

제시된 문제에서의 텍스트는 0과 1로 주어지며 하나의 행만을 비교대상으로 하는 것이 아니라 패턴이 2차원 배열이고 이를 비교대상으로 하는 것이 약간 다르기 때문에 KMP 방법을 응용해 일치하는 패턴을 찾아간다.

먼저 주어진 텍스트의 첫 행부터 패턴의 첫 행 | 1 | 0 | 0 | 이 일치하는 곳을 차례로 찾아 나간다.

텍스트의 두 번째 행에서 패턴의 첫 행이 일치하는 두 곳을 발견할 수 있다.

0	0	0	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	0	0

패턴과 일치하는 곳이 발견되면 그 곳을 중심으로 패턴의 행 길이만큼의 밑의 행과 패턴과 일치여부를 확인하면 된다.

0	0	0	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	0	0
		0	0	1			0	0	0
		0	0	1			0	0	0

소스 코드

```
#include <stdio.h>
#include <string.h>

#define INFILE "input.txt"
#define OUTFILE "output.txt"

#define PKIND 4
#define MMAX 100
#define PMAX 100

char text[MMAX][MMAX], pattern[PKIND][PMAX][PMAX];
int cond[PKIND][MMAX][MMAX];
int tsize, psize, success=0;

FILE *ftp=fopen("hagio.txt", "wt");

void turn()
{
        int i, j, k;

        for (i=1; i<PKIND; i++)
                for (j=0; j<psize; j++)
                        for (k=0; k<psize; k++)
                                pattern[i][j][k]=pattern[i-1][psize-k-1][j];

        for (i=0; i<PKIND; i++)
        {
                fprintf(ftp,"\n\n패턴 종류 %d",i);
                for (j=0; j<psize; j++)
                {
                        fprintf(ftp,"\n");
                        for (k=0; k<psize; k++)
                                fprintf(ftp,"%c",pattern[i][j][k]);
                }
        }
```

```
}

void preKmp(char *p, int m, int kmpNext[])
{
    int i, j;

    i = 0;
    j = kmpNext[0] = -1;
    while (i < m) {
        while (j > -1 && p[i] != p[j])
            j = kmpNext[j];
        i++;
        j++;
        if (p[i] == p[j])
            kmpNext[i] = kmpNext[j];
        else
            kmpNext[i] = j;
    }
}


void KMP(char *p, int pkind, char *t, int tposx, int tposy)
{
        int i, j, k, l=-1, count=0;
        int kmpNext[1000]={0, };
        int m=strlen(p);
        int n=strlen(t);

        // KMP 선행처리
        preKmp(p, m, kmpNext);

        // 매칭 검사
        i = j = 0;
        while (j < n) {
                while (i > -1 && p[i] != t[j])
                        i = kmpNext[i];
                i++;
                j++;
```

```
                if (i >= m) {
                        for(k=1, count=0; k<psize; k++)
                        {
                                if((k-1)*psize!=count)
                                {
                                        fprintf(ftp,"break(l=%d)\n", l);
                                        break;
                                }

                                for(l=0; l<psize; l++)
                                        if(pattern[pkind][k][l]!=text[tposx+k][j-i+l])
                                                break;
                                        else
                                                count++;
                        }

                        if((k-1)*psize==count)
                                success++;
                        i = kmpNext[i];
                }
        }
}

void input(void)
{
        FILE *fp=fopen(INFILE, "rt");
        int i, j;

        fscanf(fp, "%d", &tsize);

        for (i=0; i<tsize; i++)
                fscanf(fp, "%s", text[i]);

        fscanf(fp, "%d", &psize);

        for (i=0; i<psize; i++)
                fscanf(fp, "%s", pattern[0][i]);
```

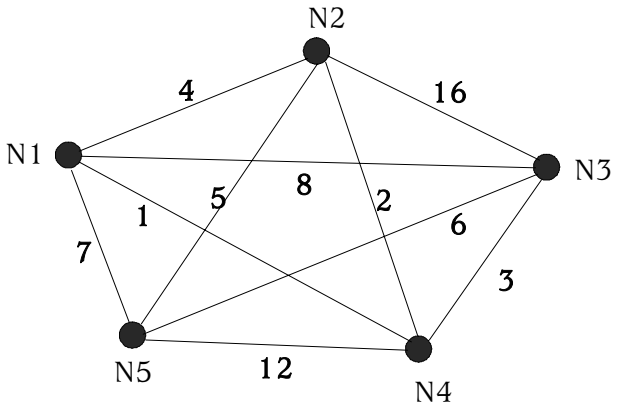
```
        fclose(fp);
}

void matching(void)
{
        int i, j, k;
        int match=0;

        for (i=0; i<PKIND; i++)
                for (j=0; j<(tsize-psize+1); j++)
                        KMP(pattern[i][0], i, text[j], j, 0);
}

void output(void)
{
        FILE *fp=fopen(OUTFILE, "wt");

        fprintf(fp, "%d", success);

        fclose(fp);
}

int main(void)
{
        input();
        turn();
        matching();
        output();
        fclose(ftp);
        return 0;
}
```

16 그룹 나누기

▸▸ 주어진 데이터를 그래프로 표현하는 방법과 그래프를 이용하여 더 빨리 주어진 문제를 해결할 수 있는 방법을 찾는가를 묻는 문제이다.

문제 (난이도 ★★★★☆)

철수가 살고 있는 마을은 네트워크가 완벽히 구축되어 있는 마을이다. 마을 어느 한 집에서 나머지 다른 모든 집으로 네트워크 라인이 연결되어 있다. 그런데 마을 사람들은 네트워크 라인의 속도가 너무 느린 것에 대해 많은 불만을 가지고 있었다.



그러던 중 정부의 IT 인프라 구축 사업을 통해 기존에 사용하던 라인보다 수백 배 빠른 라인으로 교체할 수 있는 기회를 얻게 되었다. 그러나 새로운 라인으로 교체하기 위해서는 라인의 거리의 제한 때문에 이 문제를 해결해야 한다.

새로운 라인의 속도는 기존의 라인보다 수백 배 빠른 대신 하나의 집과 다른 집을 연결할 때 일정한 길이 t(정수)를 넘어서면 잡음이 많아 제대로 정보를 주고받을 수 없다는 데 있다. 일단 두 집이 연결되면 다른 집을 연결하기 위한 길이는 다시 t 이내이어야 한다.

이 문제의 해결을 위해 정부에서는 마을에 두 개의 라인을 제공하기로 결정했다. 이제 남은 문제는 마을의 모든 집들을 A, B 두 그룹으로 나눌 때 그룹 A, B에 속한 모든 집들의 연결된 각각의 거리가 t보다 작아지게 집들을 나누는 것이다.

이제 마을의 모든 집들을 조건에 맞게 A, B 두 그룹으로 나눌 수 있는지 여부와 나눌 수 있다면 각각의 그룹을 출력하는 프로그램을 작성하시오.

프로그램의 이름은 "network"로 한다.

입력 형식

입력 파일의 이름은 “Input.txt"로 한다.
첫 번째 줄에는 점의 개수 N(2≤N100)이 주어진다. 각 점 N은 차례대로 1~N까지의 이름을 갖는다.
두 번째 줄에는 정수 t가 주어진다.
세 번째 줄부터 각 줄에 N 개씩 N 줄 형태의 테이블이 입력된다. 이 테이블은 각 점들 간의 거리를 테이블로 나타낸 것이다. 각 줄의 값 들은 공백으로 분리되어 있다.

출력 형식

출력 파일의 이름은 “Output.txt"로 한다.
그룹을 A, B 두 그룹으로 나눌 수 있다면
첫 번째 줄에 같은 색깔을 가진 A그룹 점의 이름을 출력한다.(이때 그룹 A, B 는 임의로 정한다.)
두 번째 줄에 같은 색깔을 가진 B 그룹 점의 이름을 출력한다.
만일 그룹을 A, B 두 그룹으로 나눌 수 없다면 첫 번째 줄에 error를 출력한다.

입력과 출력의 예 1

입력의 예(Input.txt)

```
5 4
0 4 8 1 7
4 0 16 2 5
8 16 0 3 6
1 2 3 0 12
7 5 6 12 0
```

출력의 예(Output.txt)

```
1 2 4
3 5
```

입력과 출력의 예 2

입력의 예(Input.txt)

```
5 5
0 4 8 1 7
4 0 16 2 5
8 16 0 13 9
1 2 13 0 12
7 5 9 12 0
```

출력의 예(Output.txt)

```
error
```

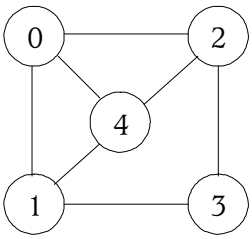
2 문제의 배경 및 관련 이론

자료구조에서 트리(tree)가 매우 융통성 있는 자료 구조이고 많은 응용 분야에서 적용되고 있지만 트리의 특성 때문에 발생되는 가장 큰 제약은 부모(parent)와 자식(child)간의 직접적인 관계와 형제들(sibling)들 간의 간접적인 관계만을 표현할 수 있다는 것이다.

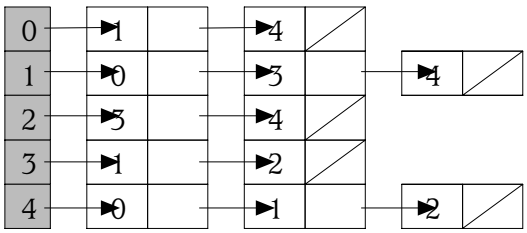
이런 트리의 단점을 보완하여 부모, 형제간의 관계 뿐 만이 아니라 다른 일반 관계도 표현할 수 있도록 트리를 일반화시킨 자료구조가 바로 그래프(graph)이다.

그래프는 정점(vertex) 혹은 노드(node)들의 집합과 이들 정점들을 연결하는 간선(edge)으로 구성되며 정점과 간선의 수에 대한 제약은 없다.

그래프를 표현하는 방법으로는 인접리스트와 배열로 표현할 수 있다.



	0	1	2	3	4
0		1			1
1	1			1	1
2				1	1
3		1	1		
4	1	1	1		



그래프의 모양에 따라 특정한 순서로 그래프의 정점을 방문하는 것이 자주 이용되며 이를 그래프의 순회라 하며, 트리의 순회와 개념이 유사하다. 그래프의 순회에는 정점을 방문할 때 멀리 갈 수 있는 데까지 우선 가보는 방법의 깊이우선탐색(DFS: Depth First Search)과 우선 가깝게 인접한 정점을 모두 방문한 후 그 다음으로 가깝게 인접한 정점을 방문하는 너비우선탐색(BFS: Breadth First Search)이 있다.

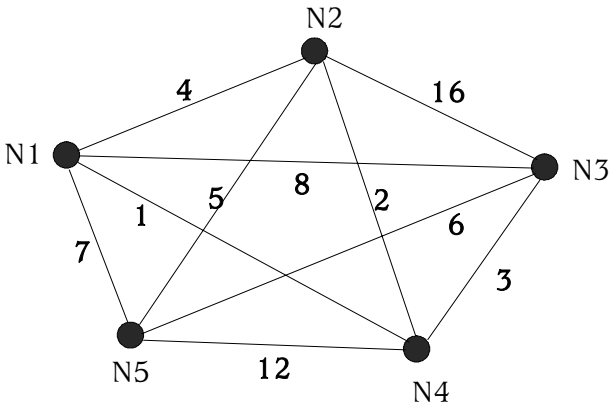
또한 그래프의 최단거리를 구하기 위한 디익스트라(Dijkstra) 알고리즘과 플로이드(Floyd) 알고리즘, 최소-비용 신장 트리(minimum-cost spanning tree, MST)를 구하기 위한 프림(Prim) 알고리즘과 크루스칼(Kruskal) 알고리즘, 이러한 그래프의 응용으로 m-채색 문제 등이 있다.

3 문제 해설

주어진 점들 간의 관계를 그래프로 나타내면 주어진 점들을 그래프의 각 노드(node)로 설정하고 점들 간의 거리를 각 노드들을 연결하는 간선(edge)으로 정할 수 있다.

다음과 같은 점 5개가 주어지고 각 점들 간의 거리가 테이블로 주어지면 이를 그래프로 표현할 수 있다.

N	1	2	3	4	5
1	0	4	8	1	7
2	4	0	16	2	5
3	8	16	0	3	6
4	1	2	3	0	12
5	7	5	6	12	0



위의 그래프는 N개의 노드를 가지며 각 노드들은 N-1개의 간선들이 다른 노드들과 연결되어 있는 형태의 완전 그래프가 된다. 이 완전 그래프를 t보다 작은 간선들로 이루어진 두 개의 집합으로 나누어야 한다.

그래프의 간선들 중 t값보다 작은 간선을 찾아보자.

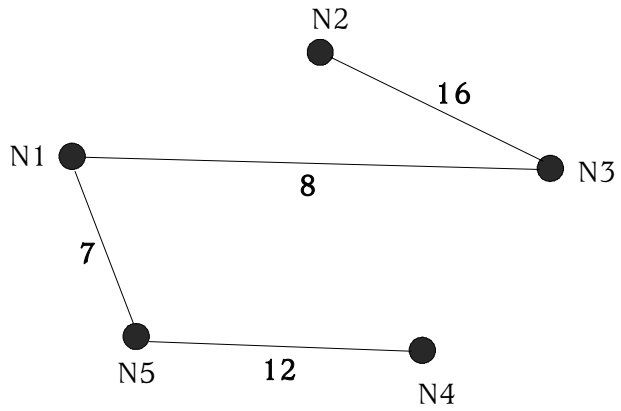
점 N1과 점 N2를 잇는 간선의 크기가 t보다 작다면 두 점 N1, N2는 같은 집합에 들어갈 가능성이 있다. 이는 가능성만 있는 것이지 반드시 같은 집합에 들어간다고 볼 수 없다. 왜냐하면 N2와 N3를 잇는 간선이 t보다 작다고 할 때 N2와 N3도 역시 같은 집합에 포함될 수 있다. 그러나 그렇다고 해서 점 N1과 N3가 t보다 작다고 말할 수 없다.

따라서 집합에 포함되는 관계는 같은 두 점 간의 관계 뿐 만이 아니라 다른 점과의 관계까지도 따져보아야 한다.

그리고 점 N1과 점 N2를 잇는 간선의 길이가 t보다 크다면 두 점 N1과 N2는 절대로 같은 집합에 속할 수 없다. 또한 두 점 N2와 N3를 잇는 간선의 길이가 t보다 크다면 점 N2와 N3도 같은 집합에 속할 수 없을뿐더러 점 N1, N2, N3 중에서 어느 두 개도 같은 집합에 속할 수 없게 된다.

즉 두 개의 집합으로 나눌 수 없는 경우가 된다.

주어진 거리 t를 6이라고 가정하고 t보다 큰 간선만을 선택해서 다시 그려보면 아래와 같다.

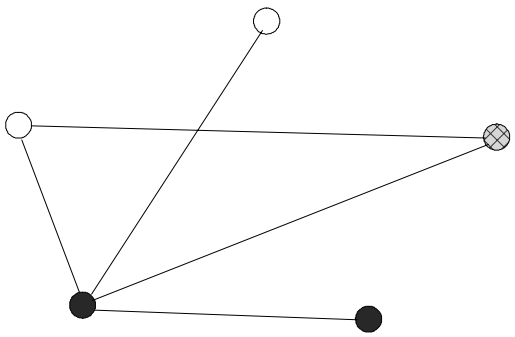


위의 그래프에서 나타나는 각각의 간선들로 연결되는 두 점들은 서로 다른 집합에 속해야 한다. 예를 들어 점N1과 점 N3은 서로 다른 집합에 속해야 하며 점N1과 N5도 서로 다른 집합에 속해야 한다.

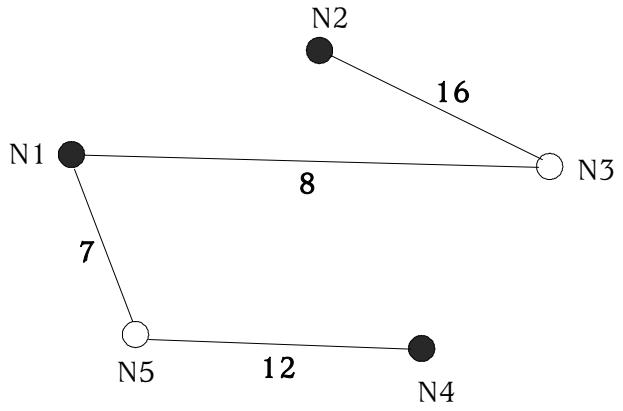
위의 그래프에서 모든 점들에 대해 같은 그룹에 속할 수 있는 점들에 대해 색을 칠해보자. 점 N1을 ●으로 칠하면 점 N1과 연결되어 있는 다른 점들은 같은 색으로 칠할 수 없다. 이렇게 각각의 점들이 간선으로 이어져 있는 점들에 대해서 항상 서로 다른 색이 칠해지도록 각 점에 색을 칠해보자.

만일 ●, 두 가지 색으로 모든 점을 칠할 수 있다면 집합은 2개로 나뉘어 진다는 뜻이다. 즉 모든 점들을 칠하기 위해 사용되어지는 색의 수가 집합이 된다는 말이다.

아래와 같이 사이클이 형성될 때에는 2가지 색으로 칠할 수 없다.



그룹 A에 속할 수 있는 점들은 ●으로 그룹 B에 속하는 점들은 으로 칠하면 아래와 같이 A 그룹에 속한 점들은 N1, N2, N4가 되고 B그룹에 속한 점들은 N3, N5가 된다.



그래프의 색칠을 위해 너비우선탐색(Breath First Search; BFS)을 이용하자.

임의의 한 노드 A를 선택한 다음 ●을 칠한다. 이때 A 노드에 연결된 다른 노드들의 집합 B는 ●으로 칠할 수 없으므로 다른 색으로 칠해야 한다. 그러므로 A에서 너비 우선탐색되는 모든 노드들의 집합 B는 다른 색인 으로 칠한다. B에서 탐색된 새로운 그룹 C는 색 로 칠할 수 없다.

이 경우 가능한 색은 ●을 포함한 다른 색이다. 만일 ●으로 칠할 수 없는 상황이 된다면 두 가지 그룹으로 나눌 수 없다는 말이다.

만일 그룹 C가 ●으로 칠할 수 있다면 C에서 너비 우선 탐색되는 모든 노드들은 을 포함한 다른 색으로 칠할 수 있으며 색으로 칠할 수 없다면 두 가지 그룹으로 나눌 수 없다는 말이다.

소스 코드

```
#include <stdio.h>

#define MAXN 100
#define MAXQ 100

int n, t, matrix[MAXN][MAXN], visited[MAXN], group[MAXN];
int qfront, qrear, queue[MAXQ];
FILE* fp;

int input(void)
{
	int i, j, temp;

	if((fp = fopen("input.txt", "r")) == NULL)
		return 1;

	fscanf(fp, "%d %d", &n, &t);

	for(i=0;i<n;visited[i++]=0)
		for(j=0;j<n;j++)
		{
			fscanf(fp, "%d", &temp);
			matrix[i][j] = (temp > t);
		}
	fclose(fp);
	return 0;
}

void qpush(int n)
{
	queue[qfront] = n;
	qfront = (qfront + 1) % MAXQ;
}
```

```
int qpop()
{
        int blah = queue[qrear];
        qrear = (qrear + 1) % MAXQ;
        return blah;
}

int process(void)
{
        int i, j, p;
        for(i=0;i<n;i++)
                if(!visited[i])
                {
                        qfront = qrear = 0;
                        qpush(i);
                        visited[i] = 1;
                        group[i] = i%2;

                        while(qfront != qrear)
                        {
                                p = qpop();
                                for(j=0;j<n;j++)
                                        if(matrix[p][j])
                                        {
                                                if(visited[j])
                                                {
                                                        if(group[p] == group[j])
                                                                return 0;
                                                } else {
                                                        visited[j] = 1;
                                                        qpush(j);
                                                        group[j] = !group[p];
                                                }
                                        }
                        }
                }
```

```
	return 1;
}

int output(int grouped)
{
	int i;

	if((fp = fopen("output.txt", "w")) == NULL)
		return 0;

	if(grouped)
	{
		for(i=0;i<n;i++)
			if(!group[i]) fprintf(fp, "%d ", i+1);

		fprintf(fp, "\n");

		for(i=0;i<n;i++)
			if(group[i])
				fprintf(fp, "%d ", i+1);
		fprintf(fp, "\n");
	}
	else {
		fprintf(fp, "error\n");
	}
	fclose(fp);
	return 0;
}

int main(void)
{
	if(input())
		return 1;
	if(output(process()))
		return 1;
	return 0;
}
```

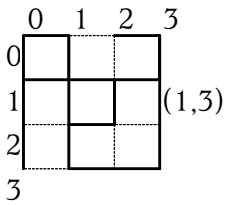
17 정사각형 수

▸▸ 판에 대한 정보가 배열로 주어지면 주어진 데이터를 가지고 정사각형의 수를 계산하는 문제로 판의 크기가 작을 때에는 모든 데이터를 다 검사하는 방법을 사용할 수 있지만 판의 크기가 커지면 더 최적화 할 수 있는 알고리즘을 생각해야 하는 문제이다.

문제 (난이도 ★★★★★)

바둑판처럼 점들의 2차 배열로 구성되어 있는 판에 일부의 점들은 선으로 연결되어 있고 나머지 점들은 연결되어 있지 않다. 이때 특정 크기의 정사각형 개수가 몇 개인지 알아보려고 한다.

예를 들어 아래 그림은 크기 1인 정사각형 2개와 크기 2인 정사각형 1개로 구성되어 있다.(여기에서 크기는 정사각형 한 변을 구성하는 선의 개수이다.)



이제 주어진 판에서 모든 가능한 정사각형의 개수를 세는 과정을 수행하는 프로그램을 작성하시오. 각 판은 $n^2(2 \le n \le 9)$개의 점들의 2차 배열과 수평선, 수직선으로 구성되어 있다. H(i, j)는 판 위의 점 (i, j) 와 (i, j+1)을 연결하는 수평선이 존재하는 가를 나타내며, V(i, j)는 점(i, j) 와 (i+1, j)를 연결하는 수직선이 존재하는 가를 나타낸다. 연결선이 존재하면 1, 존재하지 않으면 0의 값을 갖는다.

위의 판은 이차원 배열 H(n, n), V(n, n)을 이용하여 판 위의 연결된 수평 수직선의 존재를 표현하면 다음과 같다.

1 0 1	1 1 0 1
1 1 1	0 1 1 1
0 1 0	0 1 0 1
0 1 1	
수평선 H	수직선 V

파일의 이름은 “square"로 한다.

입력 형식

입력 파일의 이름은 “Input.txt"로한다.
첫 번째 줄에는 판의 크기인 n이 주어진다.
두 번째 줄부터 수평선에 대한 값 (n-1×n) 행렬이 주어진다.
다음 줄 부터는 수직선에 대한 값 (n×n-1) 행렬이 주어진다.

출력 형식

출력 파일의 이름은 “Output.txt"이다.
각 자료에 대해 사각형이 존재하는 크기 순으로 한 줄에 하나의 사각형씩 각각의 개수를 출력하시오.

입력과 출력의 예 1

입력의 예(Input.txt)

```
4
1 0 1
1 1 1
0 1 0
0 1 1
1 1 0 1
0 1 1 1
0 1 0 1
```

출력의 예(Output.txt)

```
2 1
1 2
```

2 문제의 배경 및 관련 이론

다이나믹(dynamic, 동적계획법) 알고리즘을 이용해서 문제를 해결하기 위해 주어진 문제의 순환적 성질을 먼저 구해야 한다. 문제의 순환적 성질이 구해지면 큰 문제를 여러 개의 작은 문제로 쪼갤 수 있게 된다.

큰 문제의 순환적 성질을 구하면 문제를 더 작은 문제들로 분할할 수 있고 더 작은 문제들은 다시 더 작은 문제들로 분할할 수 있다. 이들을 충분히 작은 문제로 분할했으면 문제에 대한 해답을 구할 수 있다. 충분히 작은 문제에 대한 답을 구했다면 다음에 상향식(bottom-up)으로 한 단계 위의 작은 문제부터 해결하기 시작한다.

이때 작은 문제가 해결되면 그 결과를 표에 저장하고 다음 똑같은 문제를 만났을 때 다시 문제에 대한 해답을 찾지 않고 전에 저장된 값을 사용한다.

이러한 다이나믹 알고리즘은 가능한 여러 선택 중에서 가장 좋은 것을 고르는 최적화 문제(optimization problem)에 자주 사용한다. 예를 들어 어떠한 일을 수행하는데 가장 비용이 적게 드는 방법을 구하라. 또는 가장 가까운 길을 찾는 최단거리경로문제나 최적 이진 탐색트리문제에 자주 사용한다.

최적화 원리(principle of optimality)는 주어진 문제의 부분의 해가 전체 문제의 해를 구성하는데 사용된다는 성질을 이용하는 것으로 문제가 해결되어 가는 과정에서 볼 때, 현재의 조그만 문제의 최적해가 나중에 계산 될 큰 문제의 해의 일부로 작용하다는 것이다.

따라서 작은 문제의 해를 구한 다음에 그들을 합하면 큰 문제의 해가 된다.

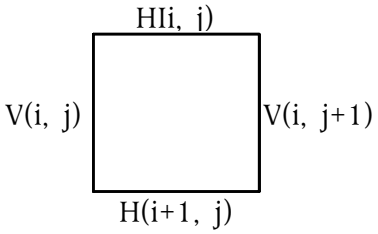
동적계획법으로 문제를 해결할 때 동적계획법은 부분 해를 계속 구성해가면서 전체 문제에 도달하는 방법을 사용하기 때문에 문제를 해결하는 과정 모두가 최적화 원리를 만족해야 하며 동적계획법은 모든 가능성을 고려하여 결정 부분 해를 결정하기 때문에 결과는 항상 최적의 결과를 얻을 수 있다.

그러나 동적계획법 수행 과정에서 많은 해들이 생성되고 그 해들을 모두 저장하고 있어야 하기 때문에 동적계획법을 구현하기 위해서 메모리 공간을 많이 필요하다.

3 문제 해설

먼저 크기가 1인 정사각형을 구하는 방법을 생각해 보자.

왼쪽 위 꼭지점의 위치가 (i, j)라면 사각형의 네 변을 나타내는 두 배열 H와 V의 원소는 H(i, j), H(i+1, j)와 V(i, j), V(i, j+1)이다.

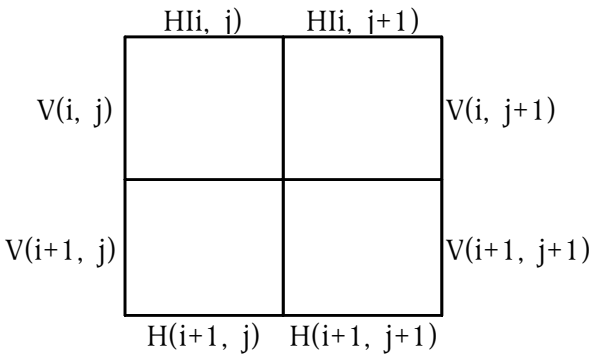


이들 H(i, j), H(i+1, j)와 V(i, j), V(i, j+1)의 모든 값들이 1이어야 점 (i, j)를 왼쪽 위 꼭지점으로 갖고 크기가 1인 정사각형이 존재하는 것이다.

주어진 판의 각각의 꼭지점에 대해 이러한 방법으로 배열 H와 V의 값을 읽어서 크기가 1인 모든 정사각형들을 구할 수 있다.

이제 크기가 2인 정사각형을 구하는 방법을 생각해 보자.

왼쪽 위 꼭지점의 위치가 (i, j)라면 크기가 2인 정사각형의 변을 이루는 배열의 원소는 H(i, j), H(i, j+1), H(i+2, j), H(i+2, j+1)와 V(i, j), V(i+1, j), V(i, j+2), V(i+1, j+2)가 된다.



이들 H(i, j), H(i, j+1), H(i+2, j), H(i+2, j+1)와 V(i, j), V(i+1, j), V(i, j+2), V(i+1, j+2)의 모든 값들이 1이어야 점 (i, j)를 왼쪽 위 꼭지점으로 갖고 크기가 2인 정사각형이 존재하는 것이다.

판에서 점(2, 2)를 왼쪽 위 꼭지점으로 갖는 정사각형이 있는지를 검사하기 위해서 H(2, 2), H(2, 3), H(4, 2), H(4, 3), V(2, 2), V(3, 2), V(2, 4), V(3, 4)의 값들을 조사해 보면 된다.

같은 방법으로 왼쪽 위 꼭지점의 위치가 (i, j)일 때 한 변의 크기가 3, 4, …, N인 정사각형을 찾을 수 있다.

이제 다른 방법을 생각해보자.
위의 방법은 모든 판들에 대해 한 변의 크기가 1인 정사각형을 찾고 다시 한 변의 크기가 2인 정사각형을 찾고, 이러한 방법을 N이 될 때까지 반복한다. 다시 말하면 한 점을 N번 반복해서 검사하기 때문에 시간복잡도가 증가한다는 말이다.

시간복잡도를 줄이기 위해 아래와 같은 동적계획법을 사용할 수 있다.

두 배열 H와 V를 수정하기 위해 편의상 크기가 d인 정사각형을 구하기 위한 두 배열의 원소를 H(i, j : d)와 V(i, j : d)로 표시한다.

먼저 크기가 1인 정사각형을 구하는 방법은 위에서 구한 방법과 같다.

왼쪽 위 꼭지점의 위치가 (i, j)이고 크기가 2인 정사각형을 구해보자.

이를 위해 두 배열의 원소 H(i, j : 1), H(i, j+1 : 1), H(i+2, j : 1), H(i+2, j+1 : 1), V(i, j : 1), V(i+1, j:1), V(i, j+2 : 1), V(i+1, j+2 : 1)를 검사해야 한다.

[H(i, j : 1) H(i, j+1 : 1)], [H(i+2, j : 1) H(i+2, j+1 : 1)], [V(i, j : 1), V(i+1, j : 1)], [V(i, j+2 : 1) V(i+1, j+2 : 1)]는 크기가 2인 정사각형의 한 변을 나타내는데 이들 각각의 두 값들을 묶어서 하나의 값에 저장하면 4개의 값만을 검사하여 크기가 2인 정사각형인지를 알 수 있다.

이를 위하여 H(i, j : 2)가 세 점 (i, j), (i, j+1), (i, j+2)를 연결하는 두 수평선 H(i, j : 1)와 H(i, j+1 : 1)이 모두 존재하는 경우에는 1로 그렇지 않은 경우에는 0으로 수정한다.

1 0 1 1 1 1 0 1 0 0 1 1 배열 H(i, j : 2)	->	0 0 1 1 0 0 0 1 배열 H(i, j : 2)

마찬가지로 V(i, j : 2)는 세 점 (i, j), (i, j+1), (i, j+2)를 연결하는 두 수직선 V(i, j : 1)와 V(i+1, j : 1)이 모두 존재하는 경우에는 1로 그렇지 않은 경우에는 0으로 수정한다.

1 1 0 1 0 1 1 1 0 1 0 1 배열 V(i, j : 2)	->	0 1 0 1 0 1 0 1 배열 V(i, j : 2)

이와 같이 하면 크기가 2인 정사가형이 존재하는 지를 검사하는 것은 단지 네 개의 값인 **H(i, j : 2), H(i+2, j : 2), V(i, j : 2), V(i, j+2 : 2)**의 값만을 비교하여 알 수 있다.

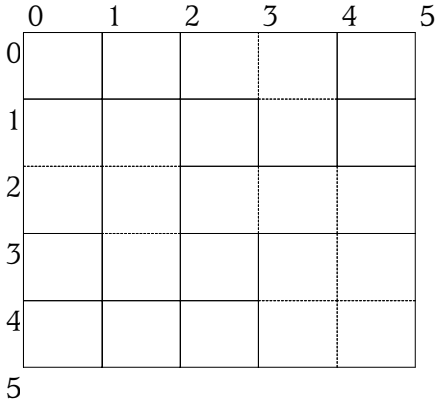
새로운 H(i, j : 2)의 값은 이전의 H(i, j : 1)와 H(i, j+1 : 1)의 값이 모두 1이면 1이고 그렇지 않으면 0이다. 같은 방법으로 배열 V(i, j : 2)의 값은 V(i, j : 1)와 V(i+1, j : 1)의 값이 모두 1인 경우에만 1이 된다. 따라서 H(i, j : 2)는 H(i, j : 1) and H(i, j+1 : 1)이 되고 V(i, j : 2)는 V(i, j : 1) and V(i+1, j : 1)이 된다.

0 0 1 1 0 0 0 1 배열 H(i, j : 2)	0 1 0 1 0 1 0 1 배열 V(i, j : 2)

두 배열 H와 V를 이용하여 크기가 2인 정사각형을 찾는 과정은 다음과 같다.
크기가 2인 정사각형의 왼쪽 꼭지점이 될 수 있는 것은 네 점 (1, 1), (1, 2), (2, 1), (2, 2 뿐이다. 각각의 경우에 H(i, j : 2), H(i+2, j : 2), V(i, j : 2), V(i, j+2 : 2)의 값을 검사한다. 점 (2, 2)에 대해 검사할 경우에는 H(2, 2 : 2), H(4, 2 : 2), V(2, 2 : 2), V(2, 4 : 2)의 값을 조사해야 한다.

이들의 값이 모두 1이므로 왼쪽 위 꼭지점이 (2, 2)이고 크기가 2인 정사각형이 존재한다.

만약 아래와 같이 판에 대해 위의 방법을 이용해 사각형을 구해보면,



두 배열 H1과 V1을 구하면 다음과 같다. 이를 이용하여 크기가 1인 정사각형의 개수를 구할 수 있다.

H1	V1
1 1 1 1 1	1 1 1 0 1 1
1 1 1 0 1	1 1 1 1 1 1
0 0 1 1 1	1 1 1 0 0 1
1 0 1 1 1	1 1 1 1 0 1
1 1 1 0 0	1 1 1 1 0 1
1 1 1 1 1	

d=1

다시 크기가 2인 정사각형의 개수를 구하기 위해 아래와 같이 H1과 V1을 수정하여 H2와 V2를 만든다.

H2	V2
1 1 1 1	1 1 1 0 1 1
1 1 0 0	1 1 1 0 0 1
0 0 1 1	1 1 1 0 0 1
0 0 1 1	1 1 1 1 0 1
1 1 0 0	
1 1 1 1	

d=2

정점 (0, 0)을 왼쪽 위 꼭지점으로 갖고 한 변의 크기가 2인 정사각형의 개수를 구하기 위해 H2(0, 0), H2(2, 0), H2(2, 0)과 V2(0, 0), V2(0, 2)가 1이어야 한다. 그러나 H(2, 0)이 0이므로 왼쪽 위 꼭지점이 H2(0, 0)이고 한 변의 크기가 2인 정사각형은 만들어 질 수가 없다. 계속해서 각 점을 왼쪽 위의 꼭지점으로 갖는 정사각형이 있는가를 검사하면 된다.

한 변의 크기가 3인 정사각형을 구해보자. 이를 위해 H2과 V2를 수정하여 H3와 V3를 만든다.

왼쪽 위 꼭지점이 (i, j)이고 크기가 3인 정사각형의 한 수평 변은 세 선들 H(i, j : 1), H(i, j+1 : 1), H(i, j+2 : 1)과 한 수직 변 V(i, j : 1), V(i+1, j : 1), V(i+2, j : 1)로 이루어진다.

H3	V3
1 1 **1**	1 1 **1** 0 0 **1**
1 0 0	1 1 1 0 0 1
0 0 **1**	1 1 **1** 0 0 **1**
0 0 **1**	
1 0 0	
1 1 **1**	

d=3

여기에서 한 변의 길이가 3인 정사각형은 2개임을 알 수 있다.
같은 방법으로 한 변의 길이가 4인 것과 5인 정사각형의 개수도 구할 수 있다.

H4	V4
1 1	1 1 1 0 0 1
0 0	1 1 1 0 0 1
0 0	
0 1	
1 1	
1 1	

d=4

H	V
1	1 1 1 0 0 1
0	
0	
0	
1	
1	

d=5

소스 코드

```
#include <stdio.h>

#define MAXN 100

int n, horiz[MAXN][MAXN], vert[MAXN][MAXN], nrect[MAXN];
FILE* fp;

int input(void)
{
	int i, j;

	if((fp = fopen("input.txt", "r")) == NULL)
		return 1;

	fscanf(fp, "%d", &n);

	for(i=0;i<n;i++)
		for(j=0;j<n-1;j++)
			fscanf(fp, "%d", &horiz[i][j]);
	for(i=0;i<n-1;i++)
		for(j=0;j<n;j++)
			fscanf(fp, "%d", &vert[i][j]);

	fclose(fp);
	return 0;
}

int process(void)
{
	int z, i, j;
	for(z=1;z<n;z++)
	{
		nrect[z-1] = 0;
		for(i=0;i<n-1;i++)
```

```
                        for(j=0;j<n-1;j++)
                                if(horiz[i][j] && horiz[i+z][j] && vert[i][j] &&
vert[i][j+z])
                                        nrect[z-1]++;

                for(i=0;i<n;i++)
                        for(j=1;j<n-z;j++)
                                horiz[i][j-1] = horiz[i][j-1] && horiz[i][j];

                for(i=1;i<n-z;i++)
                        for(j=0;j<n;j++)
                                vert[i-1][j] = vert[i-1][j] && vert[i][j];
        }
        return 1;
}

int output(void)
{
        int i;
        if((fp = fopen("output.txt", "w")) == NULL)
                return 0;
        for(i=n-1;i>=0;i--)
                if(nrect[i])
                        fprintf(fp, "%d %d\n", i+1, nrect[i]);
        fclose(fp);
        return 0;
}

int main(void) {
        if(input())
                return 1;
        process();
        if(output())
                return 1;
        return 0;
}
```

18 고장에 대비한 기관차 운행

▸▸ 어떤 문제를 푸는데 있어서 정형화 된 알고리즘을 적용할 수 없는 경우가 있다. 이런 경우에는 Dynamic Programming(동적 프로그래밍) 기법이 이용된다. 간단한 작은 기관차 문제를 통해서 Dynamic Programming 기법을 익혀본다.

문제 (난이도 ★★★☆☆)

잘 알다시피 기차는 맨 앞에 있는 기관차 1대가 손님들이 탄 객차 여러 칸을 끌고 다닌다. 요즘 운행되는 기관차들은 오랜 시간 운행을 해왔고, 또 낡아서 철도청에서는 갑작스런 기관차 고장에 대한 대책 회의를 했다. 회의의 결론은 새로운 기관차를 새로 사는 것은 돈이 많이 들기 때문에 값이 싼 소형 기관차를 각 역마다 3개씩 예비로 두는 것이다. 이 소형기관차는 값이 싼 반면에 원래의 기관차보다 적은 수의 객차를 끌 수밖에 없다. 하지만 소형 기관차 3대 만으로는 원래 기관차가 끌 던 수만큼의 객차를 모두 끌 수 없다는 문제점이 제기 되었다. 이미 정책을 시행해 버렸고 위와 같은 문제점이 발생했기 때문에 소형 기관차들이 어떤 객차를 끌고 가는 것이 좋을까 하는 문제를 놓고 다음과 같이 결론이 내려졌다.

첫째, 소형 기관차가 최대로 끌 수 있는 객차의 수를 미리 정해 놓고, 그 보다 많은 수의 객차를 절대로 끌게 하지 않는다. 소형 기관차가 최대로 끌 수 있는 객차의 수는 서로 같다.

둘째, 소형 기관차를 운행하게 될 때는 최대한 많은 손님을 목적지까지 운송하도록 한다. 각 객차에 타고 있는 손님의 수는 미리 알고 있으며, 다른 객차로 손님들은 이동하지 않는다고 가정한다.

셋째, 각 소형 기관차는 연속적으로 이어진 객차를 끌게 한다. 객차의 번호는 편의상 기관차 바로 뒤의 객차를 시작으로 1번부터 번호를 부여한다.

예를 들면 기관차가 끌고 가던 객차가 7칸이고, 소형 기관차 1대가 최대로 끌 수 있는 객차의 수는 2칸이라고 하자. 그리고 1번부터 7번까지 각 객차에 타고 있는 손님의 수가 아래 표와 같다고 하자. 괄호 속에 있는 숫자는 객차 번호를 나타낸다.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
35	40	50	10	30	45	60

위의 규칙을 따르면 소형 기관차 3대는 각각 1-2번, 3-4번, 그리고 6-7번 객차를 끌고 가면 손님 240명을 운송할 수 있고, 이보다 많은 수의 손님은 운송할 수 없다.
기관차가 끌고 가던 객차의 수와 각 객차에 타고 있던 손님의 수, 그리고 소형 기관차가 최대로 끌 수 있는 객차의 수가 주어질 때, 소형 기관차 3대를 이용하여 최대로 운송할 수 있는 손님의 수를 구하는 프로그램을 작성하시오.

프로그램의 이름은 "train"으로 하고, 프로그램의 실행 시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 "Input.txt"로 한다.
입력 파일의 첫째 줄에 기관차가 끌고 가던 객차의 수가 입력된다. 그 수는 50,000 이하이다.
둘째 줄에는 기관차가 끌고 가던 객차에 타고 있는 손님의 수가 1번 객차부터 차례로 입력된다. 한 객차에 타고 있는 손님의 수는 100명 이하이고, 입력되는 숫자들 사이에 빈칸이 하나씩 있다.
셋 째 줄에는 소형 기관차가 최대로 끌 수 있는 객차의수가 입력된다. 그 수는 기관차가 끌고 가던 객차 수의 1/3보다 적다.

출력 형식

출력 파일의 이름은 "Output.txt"이다. 한 줄에 소형 기관차 3대를 이용하여 최대로 운송할 수 있는 손님 수를 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

```
7
35 40 50 10 30 45 60
```

출력의 예("Output.txt")

```
240
```

2 문제의 배경 및 관련 이론

1. 동적 계획법과 분할 정복 알고리즘

앞서 설명한 동적 계획법과 분할 정복 알고리즘을 비교해 보자. 동적 계획법(dynamic programming)은 큰 문제를 하위 문제들로 나누고 나중에 재결합한다는 면에서 분할 정복 방법과 유사하다. 그러나 두 방법에서 하위 문제들 간의 관계가 서로 다르다. 분할 정복 알고리즘에서는 하위 문제들이 서로 독립적이다. 따라서 각 하위 문제들을 재귀를 사용해서 풀고 그 결과를 다른 하위 문제의 결과와 합친다.
동적 프로그래밍은 많은 알고리즘들에 사용되는 기법이다. 대표적으로 욕심쟁이 알고리즘이 있다. 욕심쟁이(greedy) 알고리즘은 매 순간 가장 좋아 보이는 것을 선택한다. 즉 지역적으로 최적인 결정을 내리고 그 결정이 전역적으로도 최적인 해가 되길 바란다. 하지만 불행히도 한 순간에 가장 좋아 보이는 결정이 최후에도 가장 좋지 않을 수도 있다. 따라서 욕심쟁이 알고리즘이 항상 최적의 결과를 내는 것은 아니다.

욕심쟁이 알고리즘의 예로 자료 압축에 사용되는 알고리즘인 허프만 코딩(Huffman coding)이 있다. 허프만 코딩에서는 허프만 트리를 만드는 부분이 가장 중요하다. 허프만 트리는 잎 노드(leaf node)들로부터 위쪽으로 만들어간다.

① 우선 압축할 각 기호들과 발생 빈도를 각자의 이진 트리의 루트 노드(root node)에 넣는다. 이때에는 각 트리가 하나의 노드로 이루어져 있음을 명심하여 삽입토록 해야 한다.

② 다음에 발생 빈도가 가장 낮은 두 트리를 합병하여 하나의 트리로 만들고 두 트리의 발생 빈도 합을 만들어진 트리의 루트 노드에 넣는다.

③ 이 과정을 하나의 트리가 될 때까지 반복하면 허프만 트리가 완성된다. 최종 트리의 루트노드에는 모든 기호들의 발생 빈도의 합이 들어가고 잎 노드에는 각 기호들과 그 기호의 발생 빈도가 들어가게 된다.

허프만 코딩은 항상 결합하기에 가장 좋은 두 트리만을 찾으므로 욕심쟁이(greedy) 알고리즘이다.

3 문제 해설

주어진 문제는 동적 계획법을 이용하는 가장 기본적인 문제이다.
소형 기관차의 숫자는 3, 소형 기관차는 최대 2개의 이어진 객차만을 끌 수 있다는 조건을 기준으로 프로그래밍 하면 된다.

먼저 전체적인 흐름을 살펴보자.

① 입력 파일로부터 각 객차별 승객의 수를 입력받는다
② 입력받는 동안 수의 승객의 합을 위한 배열에 각각 이전 객차의 승객 수를 더한 값을 저장한다.
③ 소형 기관차의 숫자와 원래 기관차가 끌 수 있는 수만큼의 이차원 배열에 값을 갱신시킨다.
④ 이때 중요한 것은 현재 배열의 위치에서 소형 기관차의 수 만큼을 뺀 배열의 인덱스 상에 존재하는 값과 현재까지의 sum의 값의 차에 따라 값을 갱신 시켜야 한다는 것이다.

위의 과정을 주어진 예와 그림을 통해서 설명해 보면 다음과 같다.
sum 배열에 들어가는 값

0	35	75	125	135	165	210	270

이렇게 저장된 값들을 2차원 배열에 갱신시킨다. 일단 loop을 돌면서 이전 인덱스의 값을 현재에 복사한다. 그리고 이차원 배열의 행에 대한 인덱스 값이 소형 기관차가 끌 수 있는 최대의 수보다 작을 때는 이전 요소의 값을 복사한다. 그렇지 않을 때는 최대로 끌 수 있는 기관차수의 값을 인덱스 값의 차로 하여, 그 때의 합의 차에 의해 다음 요소에 갱신될 값을 결정하면 된다. loop를 따라 2차원 배열에 들어가는 값들을 출력해 보면 다음과 같다.

0	0	0	0
0	0	0	0
0	75	75	75
0	90	90	90
0	90	135	135
0	90	135	135
0	90	165	210
0	105	195	240

소스 코드

```
#include<stdio.h>
#include<stdlib.h>
#define size 100000
#define k 3                                    // 소형 기관차의 숫자
#define in_file "Input.txt"
#define out_file "Output.txt"

int s[size + 1][4];
int a[size + 1];
int sum[size + 1];

FILE *fi, *fo;

int l, n, j;

void main(void)
{
        fi = fopen(in_file, "rt");
        fscanf(fi, "%d", &n);
        for(int i = 1; i <= n; i++)
        {
                fscanf(fi, "%d", &a[i]);
                sum[i] = sum[i - 1] + a[i];
        }
        fscanf(fi,"%d", &l);
        for(i = 1; i <= n; i++)
        {
                for(j = 1; j <= k; j++)
                {
                        s[i][j] = s[i - 1][j];
                        if(i - l >= 0)
                        {
                                if(s[i][j] < s[i - l][j - 1] + (sum[i] - sum[i - l]))
                                {
```

```
                                        s[i][j]=s[i - 1][j - 1] + (sum[i] - sum[i - 1]);
                                }
                        }
                }
        }

        fo = fopen(out_file, "wt");

        fprintf(fo, "%d\n", s[n][k]);

        fcloseall();
}
```

19 순찰 구역 관리

▸▸ 그래프를 이용하는 문제 중 MST(Minimal Spanning Tree, 최소 신장 트리)에 관한 문제에 도전해 본다. 정점의 수가 N이고 에지의 수가 W 일 때 O(NW)의 시간 복잡도로 최소 신장 트리를 만들어 보자.

문제 (난이도 ★★★☆☆)

한 도시의 경찰서에서 관리하는 순찰 구역들은 덤불로 분리되어 잇고, 1부터 N(1≤N≤200)까지 연속적으로 번호가 붙어 있다. 이 도시의 경찰관들은 N개의 순찰 지역을 자유롭게 이동하기를 원한다. 경찰관들은 임의의 순찰 구역으로부터 다른 모든 순찰 구역으로 이동할 수 있도록 순찰 구역 쌍들 사이의 샛길들을 관리하기를 원한다. 경찰관들은 그들이 관리하고 있는 샛길을 따라 양방향으로 이동할 수 있다.

경찰관들은 실제로 샛길을 처음부터 새로 만들지 않고, 대신에 그들이 발견한 범죄자들이 만들어 놓은 샛길을 관리함으로써 이동하는데 사용한다. 매주 그들은 이미 발견한 범죄자들이 만든 샛길들의 일부나 전부를 선택하여 관리할 수 있다.

신기하게도 경찰관들은 매주 초에 범죄자들이 만들어 놓은 새로운 샛길을 하나씩 발견한다. 그리고 나서 그들은 임의의 순찰 구역에서 다른 순찰 구역으로 이동할 수 있도록 그 주 동안 관리할 샛길들의 집합을 결정해야 한다. 경찰관들은 현재 그들이 관리하고 있는 샛길들만을 이용할 수 있다.

이와 함께 경찰관들은 항상 그들이 관리해야 하는 샛길의 총 길이를 최소화하기 원한다. 경찰관들이 샛길을 관리하는데 드는 비용은 샛길의 길이에 비례한다. 전번 주에 어떤 샛길을 관리했든지 상관없이, 경찰관은 그들이 이미 발견한 범죄자들의 샛길 집합에 임의의 부분 집합을 선택해서 관리할 수 있다.

범죄자들의 샛길은 결코 직선이 아니다. 같은 두 순찰 구역을 잇는 두 샛길의 길이가 서로 다를 수도 있다. 더구나 두 샛길이 교차하더라고, 경찰들은 지침에 따라, 순찰 구역에서가 아니면 샛길을 바꾸어 이동하지 않는다.

매주 초에 경찰관들은 그들이 새로 발견한 범죄자의 샛길 하나를 알려줄 것이다. 그러면 당신의 프로그램은 경찰관들이 임의의 순찰 구역에서 다른 모든 순찰 구역으로 이동할 수 있도록 그들이 그 주에 관리해야 할 샛길들의 최소 총 길이를 출력해야 한다. 만약 그런 샛길들이 존재하지 않는다면 존재하지 않는다는 표기를 해야 한다.

프로그램의 이름은 “police”로 하며 제한 시간은 1초를 넘지 않는다.

입력 형식

입력 파일의 이름은 "Input.txt"로 한다.
첫째 줄에는 두 정수 N(기준 순찰 구역)과 W가 빈칸을 사이에 두고 주어진다. W(1≤W<6000)는 프로그램이 처리해야 할 주(week)의 수이다.
W만큼 매주 발견된 범죄자들의 샛길에 대한 정보가 한 줄로 주어진다. 이 줄에는 세 정수가 빈칸을 사이에 두고 주어 진다
양 끝점(두 순찰 구역의 번호)과 샛길의 길이(1 이상 10000이하인 정수), 어떤 범죄자의 샛길도 양끝 점으로 같은 순찰 구역을 갖지 않는다.

출력 형식

출력 파일의 이름은 "Output.txt"로 한다.
새로 발견된 범죄자의 오솔길 하나가 입력되는 즉시 당신의 프로그램은 경찰관들이 임의의 순찰 구역에서 다른 순찰 구역으로 이동할 수 있도록 그들이 관리해야 하는 샛길들의 최소 총 길이를 한 줄에 출력해야 한다. 만일 현재까지 주어진 샛길들을 가지고 경찰관들이 임의의 순찰 구역에서 다른 순찰 구역으로 이동할 수 없다면 -1을 출력한다.

문제의 이해를 돕기 위해 위의 입력과 출력을 설명하면 4라는 순찰 구역을 기준으로 6주 동안의 새로운 샛길 정보가 들어오면 각 주마다의 순찰 구역의 최소 길이를 출력한다. 새로 들어온 순찰 구역으로 연결되는 길이 없을 때는 -1을 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

```
4 6
1 2 10
1 3 8
3 2 3
1 4 3
1 3 6
2 1 2
```

출력의 예("Output.txt")

```
-1
-1
-1
14
12
8
```

2 문제의 배경 및 관련 이론

1. MST(Minimal Spanning Tree : 최소 신장 트리)

최소 신장 트리란 어떤 그래프의 신장 트리중 간선들의 가중치의 합이 가장 작은 신장 트리를 말한다. 즉 그래프의 모든 정점들을 방문하는 경로 중에서 최소의 가중치를 가지는 경로를 찾는 것을 말하는 것이다. 그리고 최소 신장 트리를 그래프내에서 유일하지 않다. 같은 가중치를 가지는 다른 경로가 존재할 수 있음을 의미한다.

최소 신장 트리를 구하기 위한 알고리즘으로는 우선 순위 탐색(Priority First Search), Kruskal의 알고리즘, Sollin의 알고리즘, Prim의 알고리즘 등이 있다. 각 알고리즘에 대해 깊이 알고 싶다면 관련 서적을 찾아보기 바란다. 여기서는 우리가 이용하게 될 우선 순위 탐색법에 대해서만 설명하겠다.

2. 우선 순위 큐(Priority Queue)

우선 순위 큐는 스택이나 큐와 마찬가지로 자료의 임시 저장소 역할을 한다. 스택과 큐는 각각 처음 입력의 최종 출력, 처음 입력의 처음 출력등의 성질을 가지지만 우선순위 큐는 그보다 더 넓은 개념으로 우선순위(Priority)가 가장 높은 자료를 가장 먼저 꺼낸다. 물론 우리는 이 문제를 푸는데 있어 목초지간의 거리를 우선수위의 기준으로 둔다.

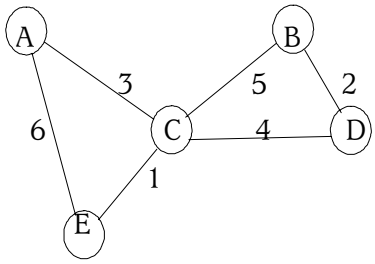
3. 우선 순위 탐색(Priority First Search)

우선 순위 탐색은 우선순위 큐를 이용하여 MST를 찾는다.

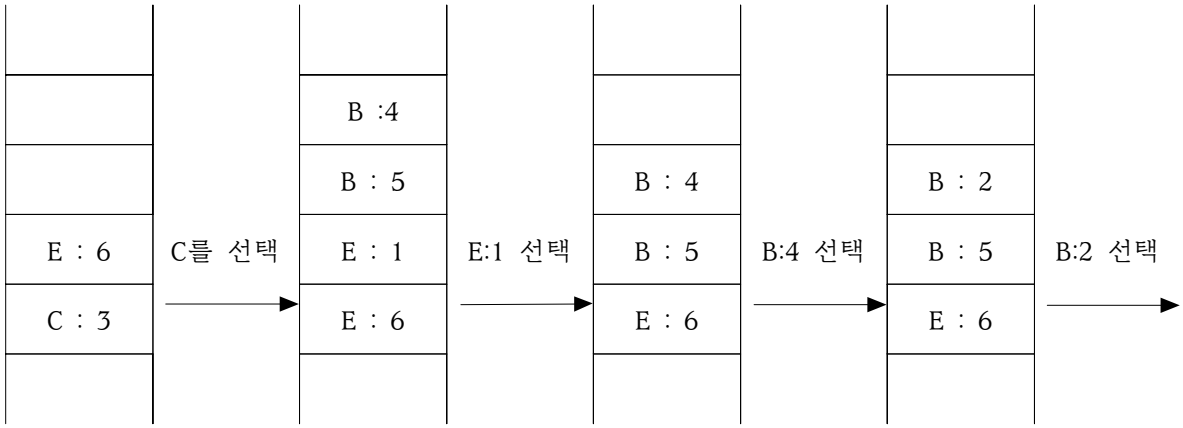
일단 우선 순위 탐색을 간단하게 설명하면 다음과 같다

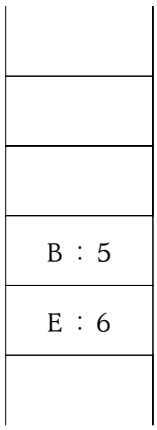
```
1. 탐색의 시작정점을 선택한다.
2. while(모든 정점들이 연결될 때까지)
{
      연결되어 있지 않은 정점중 가중치가 가장 작은 정점을 트리에 추가한다.
      새로 트리에 추가된 정점의 주변에 있던 정점들을 우선순위큐에 넣는다.
}
```

다음 그래프에서 MST를 위의 알고리즘에 따라 찾아보자. 그리고 그때의 스택의 모습을 살펴보기로 하자



A를 시작 정점이라 했을 때 Priority Queue의 변화는 다음과 같다.

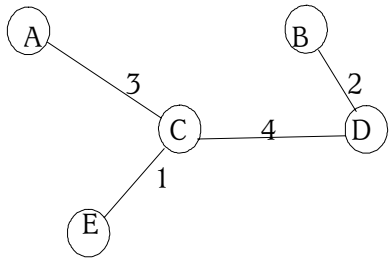




모든 정점들을 연결하는 트리가 완성되었으므로 이 여기서 실행을 멈추게 된다.

진행과정을 보면 알겠지만 현재까지 만들어진 트리의 주변경로를 우선순위 큐에 넣은 후 그중 우선순위가 가장 높은 요소를 큐로부터 꺼낸 후 그것을 트리에 추가한 후 같은 동작을 반복하면 된다.

위의 작업을 통해 만들어진 MST의 모양은 다음과 같다.



3 문제 해설

위에 설명한 알고리즘과 자료구조를 이용한다면 쉽게 해결할 수 있는 문제이다. 목초지를 그래프의 정점이라 생각하고 목초지를 연결하는 오솔길을 정점간의 간선으로 대치하여 생각해보면 주어진 문제는 MST를 찾는 문제로 완전히 매칭 시킬 수가 있다.
문제를 이렇게 매칭 시켰다면 해결 과정은 위에 제시한 알고리즘과 크게 다를 것 없이 풀 수 있다. 이러한 과정으로 문제에서 요구하는 내용을 파악하고, 그 내용에 부합하는 알고리즘을 선택하여 적용하는 훈련이 적절하다고 할 수 있다.

문제를 해결해 가는 동안 주의해야 할 점은 새로 만들어지는 트리 부분에서 늘어나는 정점들에 대한 처리이다. 이러한 정점들에 대한 처리는 신중을 기해야 한다. 만일 정점 V가 주변 다음에 연결되어야 할 정점이고 현재 저장된 간선의 가중치보다 작은 가중치를 가진다면 전체 가중치부분을 갱신하고 큐 정보를 최신화 해야 한다.

좀 더 빠른 실행을 위하여 우선순위 큐를 배열이 아닌 Heap과 같은 자료구조를 이용하여 O(logNW)의 시간 복잡도로 문제를 해결할 수도 있다.

소스 코드

```
#include<fstream.h>
#include<stdio.h>
#define Maxn 200                                    // 순찰 구역의 수
#define inputfilename "Input.txt"
#define outputfilename "Output.txt"

int process(int, int, int);
void chase(int);

static int adjec[Maxn][Maxn];              // 순찰 구역의 연결 가중치
                                           // 정보를 2차원 배열로 나타냄
static int check[Maxn], qcheck[Maxn], back[Maxn]; // 연결정보를 위한 임시 배열
static int queue[Maxn];                    // priority queue

int n, m, tot = 0, max, save1, save2;

void main(void)
{
        FILE *in = fopen(inputfilename, "r");
        FILE *out = fopen(outputfilename, "w");

        fscanf(in, "%d %d", &n, &m);
        int st, en, ga, i;                 // 새로 생긴 길에 대한 시작, 끝, 가중치값

        for(i = 1; i <= n; i++)
        {
                check[i] = i;
        }
        for(i = 0; i < m; i++)
        {
                fscanf(in, "%d %d %d", &st, &en, &ga);
                fprintf(out, "%d\n", process(st, en, ga));
        }
        fclose(in);
```

```
	fclose(out);
}

int process(int st, int en, int ga)
{
	int i, buf;
	// 처음 들어오는 연결정보 일 경우 않으면 가중치를 더해주고 연결정보 갱신
	if(check[st] != check[en])
	{
		tot += ga;
		buf = check[en];
		for(i = 1; i <= n; i++)
		{
			if(check[i] == buf)
				check[i] = check[st];
		}

		// 양방향으로 이동이 가능하므로 2차원
		// 배열상에서 대칭으로 같은 정보 저장
		adjec[st][en] = ga;
		adjec[en][st] = ga;
	}

	// 존재하던 연결에 대한 새로운 가중치 정보가 들어올 때
	else
	{
		int head = 0, tail = 1;
		queue[0] = st;
		for(i = 1; i <= n; i++)
		{
			qcheck[i] = 0;
		}
		while(1)
		{
			if(tail <= head)
				break;
```

```
                for(i = 1; i <= n; i++)
                {
                        if(adjec[queue[head]][i] != 0 && qcheck[i] == 0)
                        {
                                back[tail] = head;
                                queue[tail] = i;
                                qcheck[i] = 1;
                                if(i == en)
                                {
                                        max = -1;
                                        chase(tail);
                                        break;
                                }
                                tail++;
                        }
                }
                if(i <= n)
                        break;
                head++;
        }
        if(ga < max)
        {
                tot -= adjec[save1][save2];
                tot += ga;
                adjec[save1][save2] = 0;
                adjec[save2][save1] = 0;
                adjec[st][en] = ga;
                adjec[en][st] = ga;
        }
}
for(i = 1; i <= n; i++)
{
        if(check[i] != check[1])        {
                return -1;
        }
}
```

```
        return tot;
}

// 트리를 따라가면서 연결 인덱스를 최신화
void chase(int i)
{
        if(i == 0)
                return;
        if(max < adjec[queue[i]][queue[back[i]]])
        {
                save1 = queue[i];
                save2 = queue[back[i]];
                max = adjec[queue[i]][queue[back[i]]];
        }
        chase(back[i]);
}
```

20 신개념 잠금 장치

▸▸ 정보 올림피아드에서 절대 빼 놓을 수 없는 동적 프로그래밍에 관한 문제이다. 정형화된 알고리즘을 적용할 수 있는 능력도 중요하지만, 정형화되어 있지 않은 문제를 상황에 맞게 풀어내는 능력 또한 그에 못지않게 중요하다.

문제 (난이도 ★★★☆☆)

한 중소기업에서 새로운 형태의 가방 잠금 장치를 개발했다. 그림과 같이 가방의 잠금 장치는 위와 아래의 두 행과 N개의 열로 나누어져 있다. 따라서 잠금부분은 전체 2N개의 칸으로 이루어져 있고, 각 칸에는 0이 아닌 -10이상 10이하의 정수가 적힌 숫자판이 들어갈 수 있다.

아래 그림은 N = 7인 경우 어떤 잠금 장치의 상태를 보여주고 있다. 빈칸은 숫자판이 들어있지 않은 칸을 나타내며, 위와 아래의 행에 들어있는 숫자판의 개수는 같지 않을 수도 있다.

-3	-1	-2		5	-1	
	-3	2	4		5	2

숫자 박스의 “값”은 각 열의 위와 아래에 있는 두 숫자들의 곱을 더한 값으로 정의된다. 빈칸은 0으로 계산한다. 예를 들면 위 그림의 숫자 박스의 값은 (-3)*0 + (-1)*(-3) + (-2)*2 + 0*4 + 5*0 + (-1)*5 + 0*(-2) = -6이다.

각 행에 주어진 숫자판들에 대해 그 순서를 유지하면서 좌우로 움직이면 다른 잠금 장치의 값을 얻을 수 있다. 위의 예에서 위쪽 행에 있는 5와 -1을 오른쪽으로 각각 한 칸씩 옮기고, 아래 행의 -3을 왼쪽으로 한 칸, 2와 4를 오른쪽으로 각각 한 칸씩 옮기면

그 결과 잠금 장치는 아래와 같다.

-3	-1	-2			5	-1
-3			2	4	5	-2

이 잠금 장치의 값은 9 + 25 + 2 = 36이 된다. 주어진 잠금 장치의 위와 아래의 행에 놓인 숫자판들을 그 순서는 유지하면서 위의 조건을 만족하도록 움직여서 얻을 수 있는 숫자 박스의 최대값이 이 가방을 여는 비밀번호 값이 된다. 이 비밀번호를 구하는 프로그램을 작성하시오.

숫자판들은 좌우 빈칸으로만 움직일 수 있으며, 건너뛰는 형태로 다른 숫자판과 그 위치가 교환될 수는 없다. 다시 말하면 빈칸을 제외하면 각 행의 숫자판들의 순서는 항상 그대로 유지되어야 한다.

프로그램의 이름은 “lock”으로 하고, 실행시간은 1초를 초과할 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 "Input.txt"이다.
첫 줄에는 잠금 장치의 열의 수를 나타내는 정수 N(1≤N≤400)이 주어진다.
그 다음 두 줄에는 각각 잠금 장치의 위와 아래의 행에 놓인 초기 숫자판들의 숫자가 하나 이상 공백을 두고 나타나는데, 숫자판이 없는 빈칸은 0으로 표시된다. 단. 숫자판의 숫자는 모두 -10이상 10이하의 0이 아닌 정수이다.

출력 형식

출력 파일의 이름은 "Output.txt"이다. 입력으로 주어진 숫자 박스의 각 행의 숫자판들을 가로로 이동시켜 얻을 수 있는 잠금 장치의 최대값을 첫 번째 줄에 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

```
7
-3 -1 -2 0 5 -1 0
0 -3 2 4 0 5 -2
```

출력의 예("Output.txt")

```
36
```

2 문제의 배경 및 관련 이론

1. 동적 계획법

요즘 정보 올림피아드 문제가 정형화된 알고리즘을 적용하는 문제가 많이 출제되는 경향이기는 하지만 동적 계획법을 이용한 프로그래밍 문제에 대한 감각을 익히는 것도 매우 중요하다. 정형화된 알고리즘을 활용하는 문제들은 문제를 이해하고 맞는 알고리즘을 찾아내면 쉽게 해결할 수 있지만, 동적 계획법을 활용하는 프로그래밍 문제는 각 문제에 맞게 문제를 푸는 사람이 적절한 해법을 제시해야 하기 때문이다. 이번 문제 역시 동적 계획법을 활용한다.

동적 계획법에 대한 설명은 다른 문제들을 통해 자세히 설명했으므로 이번 문제에서는 생략하기로 한다.

2. 지극히 명확한 얘기이지만 곱이 최대가 되기 위해서는 두 정수의 부호가 같아야 한다.

(음수) * (음수) = 양수 (양수) * (양수) = 양수

누구나 다 아는 사실이지만 이번 문제에서 유념해야 할 부분이라 하겠다.
따라서 부호가 같으며 절대값이 큰 값들이 같은 행에 오도록 유도해야 하며 부호가 다른 숫자들 간의 조합이 발생할 때는 가능한 빈칸(0)과의 곱이 되도록 유도해야 한다.

3 문제 해설

입력에 주어지는 빈칸의 수는 가능한 조합의 수를 결정하게 되는 기준이 되므로 빈칸의 숫자를 먼저 알아야 한다. 박스의 열의 개수는 입력에 따라 변하지 않는 값이지만 빈칸은 입력된 형태에 따라 값이 항상 변하기 때문에 빈칸에 대한 처리부분이 반드시 들어가야만 한다.
예를 들어

-3	1	2	4
	-2		5

이러한 입력이 있을 경우 행의 수는 4이고 빈칸의 수는 2이다. 이때 아랫줄에서 가능한 조합의 수는 3 + 2 + 1이다. 위쪽 칸에도 빈칸이 2개 있다면 윗쪽 칸에서도 6가지 방법으로 숫자를 나열할 수 있을 것이다. 그러면 총 경우의 수는 36가지로 늘어나게 된다. 이처럼 입력에 주어지는 빈칸의 수에 따라 확인해야 할 경우가 늘어나므로 그것을 기준으로 최적의 해를 얻어야 한다.

소스코드 상에서 다음부분이 빈칸에 대한 처리이다.

```
l = limit * 2 - num[0] - num[1];
if(l > num[0] + num[1])
        l = num[0] + num[1];
```

빈칸의 처리가 끝났다면 이제 실제적인 계산에 들어간다.

```
for(i = 0; i < num[0]; i++)
{
        for(j = 0; j < num[1]; j++)
        {
                m = l - abs(num[0] - i - num[1] + j);
                for(k = abs(i - j); k <= m; k++)
                        table[i][j][k] = max(gett(i, j - 1, k - 1), gett(i - 1, j, k - 1),\
                        gett(i - 1, j - 1, k) + data[0][i] * data[1][j]);
        }
}
```

table 배열에는 얻어진 결과 값들이 저장되며 여기 저장된 값들 중에서 최대의 값을 선택해 내면 된다. 물론 내부 loop을 수행하는 기준 값은 위에서 얻어진 빈칸의 개수가 된다. 열의 수만큼 loop을 순환하는 것은 필요 없는 계산을 수행하는 것과 마찬가지다. 빈칸의 수만큼만 loop을 수행해도 결과를 얻어낸다.

소스 코드

```
#include<stdio.h>
#define MAXL 400
#define infile "Input.txt"
#define outfile "Output.txt"

int data[2][MAXL];
int num[2], limit;
int table[MAXL][MAXL][MAXL];
int answer;

void ready()
{
	FILE *fp;
	int i;

	fp = fopen(infile, "r");
	fscanf(fp, "%d", &limit);

	num[0] = 0;
	for(i = 0; i < limit; i++)
	{
		fscanf(fp, "%d", &data[0][num[0]]);
		if(data[0][num[0]])
			num[0]++;
	}
	num[1] = 0;
	for(i = 0; i < limit; i++)
	{
		fscanf(fp, "%d", &data[1][num[1]]);
		if(data[1][num[1]])
			num[1]++;
	}
	fclose(fp);
}
```

```
#define UNDEF -0x7fffffff
#define gett(i, j, k) ((k >= 0 && abs((i) - (j)) <= k) ? ((i >= 0 && j >= 0) ? \
        table[i][j][k] : 0) : UNDEF)

int max(int a, int b, int c)
{
    if(a > b)
    {
        if(a > c)
            return a;
        else
            return c;
    }
    else {
        if(b > c)
            return b;
        else
            return c;
    }
}

int abs(int a)
{
    if(a >= 0) return a;
    return -a;
}

void solve()
{
    int i, j, k, l, m;

    l = limit * 2 - num[0] - num[1];
    if(l > num[0] + num[1])
    {
        l = num[0] + num[1];
    }
```

```
	for(i = 0; i < num[0]; i++)
	{
		for(j = 0; j < num[1]; j++)
		{
			m = l - abs(num[0] - i - num[1] + j);
			for(k = abs(i - j); k <= m; k++)
				table[i][j][k] = max(gett(i, j - 1, k - 1), gett(i - 1, j, k - 1),\
				gett(i - 1, j - 1, k) + data[0][i] * data[1][j]);
		}
	}

	answer = UNDEF;
	for(k = abs(i - j); k <= l; k++)
	{
		if(answer < table[num[0] - 1][num[1] - 1][k])
			answer = table[num[0] - 1][num[1] -1 ][k];
	}
}

void output()
{
	FILE *fp = fopen(outfile, "w");
	fprintf(fp, "%d\n", answer);
	fclose(fp);
}

int main()
{
	ready();
	solve();
	output();

	return 0;
}
```

21 최고의 투자가가 되자

▸▸ 전형적이면서도 유명하기도 한 동적 계획법 문제이다. 동적 계획법의 기본기를 다질 수 있는 좋은 문제이므로 공부해 두는 것이 유리하다. 동적 계획법 문제를 풀기 위해서는 우선 점화식을 세워야 한다. 일단 점화식을 만들고 나면 문제를 해결하는 것은 비교적 간단하다.

문제 (난이도 ★★☆☆☆)

용석이는 우리나라 최고의 투자가가 되기를 꿈꾸고 있다. 용석이는 여러 회사에 투자를 해서 최대의 투자 이익을 얻고자 한다. 단, 투자는 만원 단위로 할 수 있으며 각 회사는 많이 투자할수록 많은 이익을 투자가에게 돌려준다. 돈을 투자하지 않은 경우에는 당연히 얻게 되는 이익금도 없다.
예를 들어, 용석이가 4만원을 가지고 두 개의 회사에 각각 만원 단위로 투자했을 경우에 얻을 수 있는 이익은 다음과 같다.

투자액수 (만원)	회사 A	회사 B
1	5	1
2	6	5
3	7	9
4	8	15

위의 경우를 가정하면, 회사 A에 1만원, 회사 B에 3만원을 투자하는 경우 투자가가 얻는 이익은 14만원 (5+9)이다. 4만원을 투자해서 가장 많은 이익을 얻을 경우 회사 B에만 4만원을 투자하는 경우로서 이때 용석이가 얻을 수 있는 투자이익금은 총 15만원이 된다 여기서 용석이는 한 회사에 돈을 나누어 투자할 수는 없다.

투자액이 정해져 있고 회사의 개수와 각 회사에 투자했을 경우에 얻게 되는 이익이 주어졌을 때 가장 많은 이익을 얻을 수 있는 투자방식과 이때의 이익금을 구하는 프로그램을 작성해야 한다.

프로그램의 이름은 “economy”로 한다.

입력 형식

입력 파일의 이름은 Input.txt이다.
이 파일의 첫째 줄에는 투자금액과 투자 가능한 회사의 개수가 나타난다.
둘째 줄부터는 각 줄은 투자액수 및 각 회사가 투자자에게 제공하는 이익금을 나타낸다.
단, 총 투자금액은 300만원을 넘을 수 없으며 투자 가능한 회사의 개수는 12개이다.

출력 형식

출력 파일의 이름은 "Output.txt"이다. 출력 파일의 첫 줄에는 각 회사에 투자한 액수가 출력된다.
둘째 줄에는 얻을 수 있는 최대 이익을 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

```
4 2
1 5 1
2 6 5
3 7 9
4 10 15
```

출력의 예("Output.txt")

```
0 4
15
```

2 문제의 배경 및 관련 이론

이러한 동적 계획법 문제를 해결하기 위해서는 우선 점화식을 만들어야 한다. 현재 돈이 m원 있고, 회사가 하나 있다고 가정해 보자. 그렇다면 최대의 이익을 얻기 위해서는 그 회사에 가진 돈을 전부 투자해야 할 것이다. 이것은 가장 간단한 방법이고, 이렇게 간단한 경우는 드물다. 이제 사고를 확장시켜서, 회사가 둘 있다고 가정해 보자. 이 경우도 그리 어렵진 않다. 회사 A에 3만원, 회사 B에 4만원을 투자하는 것을 (3, 4)라고 표현하기로 가정하자. 현재 돈이 5만원이 있다면,

(0, 5) (1, 4) (2, 3) (3, 2) (4, 1) (5, 0)

이 다섯 가지의 경우 중 가장 이익금이 많은 방법을 선택하면 될 것이다. 그럼 이제 모든 문제 해결은 끝났다. 회사가 세 개 이상이라면, 우리는 그 회사들을 2개인 것으로 축소해서 생각하면 된다. 회사가 4개가 있고, 5만원을 가지고 있다고 가정하자. 회사 a, b, c를 묶어서 A, 기업 d를 B라고 하자. 그럼 순서쌍은,

(0, 5) (1, 4) (2, 3) (3, 2) (4, 1) (5, 0)

로, 위에서 본 것과 동일한 형태이다. (3, 2)의 경우는 회사 d에 2만원을 투자하고 a, b, c에 3만원을 투자한다는 말인데, 그럼 a, b, c에는 돈을 어떻게 투자해야 최적이 될까? 여기서는 상관하지 않아도 된다. 이미 a, b, c에 3만원을 투자했을 때 얻을 수 있는 최대 이익이 계산되어 있다고 가정한 것이기 때문이다. 이제 회사 a, b, c, d에 5만원을 투자하는 문제가 a, b, c에 3만원을 투자하는 문제로 축소되었다는 것을 알 수 있다.
위에서 이야기한 것을 정리해 보자. C[i, j]가 1~i의 회사들에 j원을 투자했을 때의 최대 이익이라고 하자. 이 때 다음과 같은 경우들이 있다. (a[i, j]는 입력 데이터로 주어진 i번째 기업에 j원을 투자했을 때의 이익)

- i기업에 0원을 투자하는 경우 : C[i, j] = C[i-1, j] + a[i, 0]
- i기업에 1원을 투자하는 경우 : C[i, j] = C[i-1, j-1] + a[i, 1]
- i기업에 2원을 투자하는 경우 : C[i, j] = C[i-1, j-2] + a[i, 2]

.

- i기업에 j원을 투자하는 경우 : C[i, j] = C[i-1, 0] + a[i, j]

이 j+1가지 경우 중에 가장 최대의 이익을 계산하면 된다. 위의 경우들을 일반화 시켜보면, C[i, j] = max {C[i-1, j-k] + a[i, k]} (단, k는 $0 \le k \le j$인 정수) 가 된다. i가 1인 경우는 기업이 하나인 경우이므로 C[1, j] = a[1, j]가 된다.

문제 해설

동적계획법을 이용해 문제를 풀기 위해 가장 중요한 것은 점화식을 만드는 것이다. 물론 동적계획법을 이용해 최적해를 구하는 문제라면 최적해의 원리(principal of optimality)를 만족해야 한다.

이 문제에서는, 보통 점화식을 다음과 같이 잡을 수 있다.

F[i, j] = 1~i개의 회사에 j원을 투자했을 때의 최대 이익

이 부분 문제는 전체 문제를 C[N, M] (N은 회사 수, M은 가진 돈.)으로 표현할 수 있다. 전체 문제를 부분 문제로 분할하여 나타내는 것이 가능하다는 것이다 애초에 전체 문제도 표현하지 못하는 점화식이라면 답을 얻지 못하게 될 것이다. 문제를 어떻게 쪼갤 것인지를 생각해야 한다.

동적계획법은 기본적으로 문제를 쪼개면서 풀어 나가는 알고리즘 설계법이다. 이 점에 대해서는 분할 정복(divide and conquer)과 비슷하다. 그런데 분할 정복은 보통 큰 문제를 비슷한 크기들의 작은 문제로 쪼개서 해결하는데, 동적계획법은 조금 다르다. 일반적으로 동적계획법 문제는, 일정 회수의 선택을 반복하는 구조로 이루어져 있다.

최고의 투자자 문제에서 선택이란 어떤 것이 있을 수 있을 지 생각해보자. 어떤 회사에 대해서 얼마의 돈을 투자하는가가 바로 선택이 된다. 이 선택을 N번 반복하게 되는 것이다.

F[i, j]는, i번의 선택을 통해 답이 구해진다. 일단 F[i, j] 상태에서의 선택이란 바로 i번째 그룹에 얼마의 돈을 투자할 것인가이다. 그럼 i번째 그룹에 선택을 하기 전의 상황이란 바로 1~i-1개의 회사들에 투자를 하는 것이 될 것이다. 한 번의 선택에서는 생길 수 있는 모든 가능성을 고려해야 한다.

i번째 그룹에 투자를 하는 선택에서는, 0만원을 투자하는 경우, 1만원을 투자하는 경우.... j원을 투자하는 경우가 있다. 모든 가능성을 고려하지 않는다면 최적해를 구할 수 없다. 만약 i번째 회사에 2만원을 투자했다면(현재 상태에서의 선택), 1~i-1개의 기업에 j-2만원을 투자하는 문제(그 선택 전의 상태)로 쪼개진다. 최적해를 찾는 문제라면, 그 가능성 중에서 가장 좋은 선택을 찾아야 한다.

i번째 회사에 0원을 투자하는 경우, 1원을 투자하는 경우, ... j원을 투자하는 경우들 중에서 가장 좋은 방법을 찾아야 한다는 것이다. 지금까지 이야기했던 것을 정리하면, 다음과 같다.

F[i, j] = Max {F[i-1, j-k) + A[i, k]} (단, 0 <= k <= j)

위의 점화식은 i번째 회사에 k원을 투자하여 최대의 이익을 찾겠다는 뜻이다. 이런 방법으로 많이 생각해보면, 부분문제만 적절히 정했다면 점화식을 구하는 것은 그리 어렵지 않다는 것을 알 수 있다.

그러나 위의 점화식을 구한 것만으로도 모든 것이 끝나지 않는 경우도 있을 수 있다. 아무리 문제를 쪼개간다 하더라도, 언젠가는 쪼갤 수 없는 경우가 생기게 마련이기 때문이다. 그래서 보통 초기값이라고 부르는 것을 정의해 주어야 한다. 위의 점화식에 따르면, F[1, j]를 계산하기 위해 F[0, ...]을 참조해야 한다. 그런데 0이라는 말은 결국 아무 기업에도 투자를 안 한다는 뜻이 된다. 그러므로 F[0, j]는 j의 값에 상관없이 무조건 0이 될 것이다. 따라서 완성된 점화식은 다음과 같다,

F[i, j] = (if i=0) 0
(else) Max {F[i-1, j-k) + A[i, k]} (단, 0 <= k <= j)

소스 코드

```
#include <fstream.h>

int m, n;

int a[21][301];
int t[21][301] = {0, };
int b[21][301];

void input ()
{
	ifstream in ("Input.txt");
	in >> m >> n;
	int z;
	for (int i = 1; i <= m; i++)
	{
		in >> z;

		for (int j = 1; j <= n; j++)
			in >> a[j][z];
	}
	in.close ();
}

void process ()
{
	int i, j, k;

	for (i = 1; i <= n; i++)
	{
		for (j = 0; j <= m; j++)
		{
			for (k = 0; k <= j; k++)
			{
```

```
                                if (t[i - 1][j - k] + a[i][k] > t[i][j])
                                {
                                        t[i][j] = t[i - 1][j - k] + a[i][k];
                                        b[i][j] = k;
                                }
                        }
                }
        }
}

void output ()
{
        int g[21];
        int i, j, k;

        j = m;
        for (i = n; i >= 1; i--)
        {
                g[i] = b[i][j];
                j -= g[i];
        }

        ofstream out ("Output.txt");
        for (i = 1; i <= n; i++)
                out << g[i] << ' ';
        out << endl;
        out << t[n][m] << endl;
        out.close ();
}

void main ()
{
        input ();
        process ();
        output ();
}
```

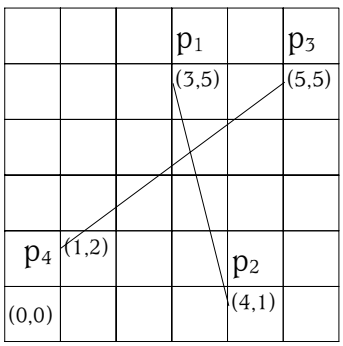
```
void printsolution(int area)
{
	FILE *out = fopen(outputfilename, "w");
	fprintf(out, "%d", area);
	fclose(out);
}

void process()
{
	memset(visit, false, sizeof(visit));

	dfssearch(0, 0);
	int i, j, max = -1;
	for(j = 0; j < MAX; j++)
	{
		for(i = 0; i < MAX; i++)
		{
			if(!visit[i][j])
			{
				int now = dfssearch(i, j);
				if(now > max)
					max = now;
			}
		}
	}
	printsolution(max);
}

int main()
{
	readData();
	process();
	return 0;
}
```

22 평면내 직선의 교점

▸▸ 좌표 평면 내에서의 선분은 일치하거나 평행하지 않는 한 교차하기 마련이다. 그러나 선분의 길이가 제한되어 있다면 만날 수도 있고 만나지 않을 수도 있다. 여러 개의 주어진 선분이 만나는 점이 있는지 없는지를 알아보는 문제이다.

문제 (난이도 ★★★☆☆)

좌표평면에서 두 개의 점이 주어지면 하나의 선분을 표현할 수 있다. 또한 선분이 두 개 이상 있으면 만날 가능성이 있다. 선분이 만난다는 것은 교점이 생긴다는 말이다. 그림으로 표현하면 다음과 같다.



교점이 생기는지 안 생기는지 판단하는 방법은 직접 그려보는 방법과 식으로 교점을 구하는 방법이 있다. 직접 그려보는 방법은 사람만이 할 수 있는 방법이고 선분의 식을 구해서 교점을 실제 구해보는 방법을 택해야 한다.

교점을 구하는 방법은 다음과 같다.
위 그림에서 각각 $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $p_3(x_3, y_3)$, $p_4(x_4, y_4)$를 기준으로 보았을 때

다음과 같은 식을 구하면,

$$t = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$$

$$x = x_1 + t(x_2 - x_1) y = y_1 + t(y_2 - y_1)$$

여기서 구한 x, y가 교점이 되는 것을 알 수 있다. 단 t의 값이 0과 1 사이에 있을 때만 교점이 생기고 0과 1 사이를 벗어나면 교점이 생기지 않는다.
선분이 더 많으면 더 많은 교점이 생길 수 있다. N개의 선분을 입력받고 몇 개의 교점이 생기는지 알아보는 프로그램을 작성하라.

프로그램의 이름은 “crossline”로 한다.

입력 형식

입력 파일의 이름은 "Input.txt"로 한다.
첫 번째 줄에 숫자 N을 입력 받는다. N은 10 이하인 수로 제한한다.
그 아랫줄부터 두 개의 점을 한 줄씩 입력받는다. 네 개의 숫자를 차례로 입력받아 각각 (x_1, y_1,), (x_2, y_2) 에 대입한다. N번 반복해서 입력받는다.

출력 형식

출력 파일의 이름은 "Output.txt"로 한다. 교점의 개수만 간결하게 출력한다.

입력과 출력의 예1

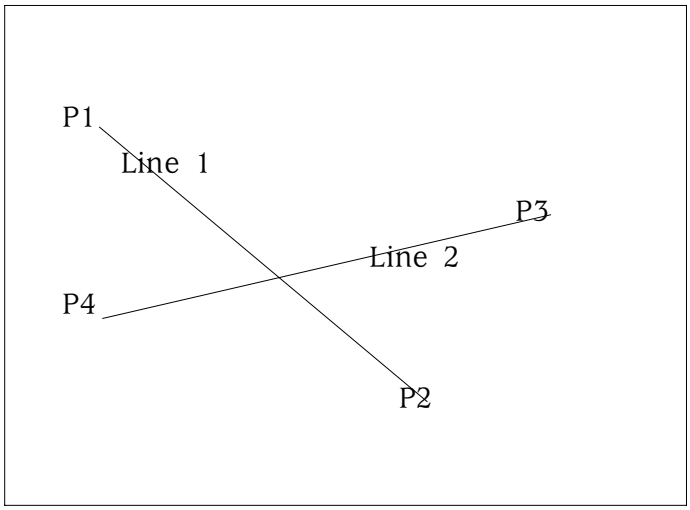
입력의 예("Input.txt")

```
3
1 1 3 3
3 5 12 5
1 5 4 0
```

출력의 예(“Output.txt")

```
1
```

2 문제의 배경 및 관련 이론



교점을 구할 때 두 선분의 식을 구하는 과정은 다음과 같다.
Line1은 P1과 P2로 이루어져 있으며, Line2는 P3와 P4로 이루어져 있다. 두개의 라인을 식으로 표현해보면 다음과 같다.

$$P(t) = (1-t)P_1 + tP_2$$

$$P(s) = (1-s)P_3 + sP_4$$

두 선의 교점은 두 선의 공통된 값이므로 P(t)와 P(s)를 같다고 하고 정리하면,

$$(1-t)P_1 + tP_2 = (1-s)P_3 - sP_4$$

과 같고, 위의 식을 x, y로 분리해 보면 아래와 같은 두 식으로 분리된다.

$$x_1 + t(x_2-x_1) = x_3 + s(x_4 - x_3)$$

$$y_1 + t(y_2-y_1) = y_3 + s(y_4 - y_3)$$

위의 식을 t와 s에 대해 정리해 보면 다음과 같다.

$$t = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$$

$$s = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$$

위의 t와 s는 두선이 서로 만날 때의 값이므로, 최종적으로 두선의 교점은 다음과 같이 나타낼 수 있다.

$$x = x_1 + t(x_2 - x_1)$$

$$y = y_1 + t(y_2 - y_1)$$

마지막으로 t와 s에 대해 정리해보자.
s와 t의 값이 0과 1 사이를 벗어나는 경우, 두 선은 교차하지 않는다고 판정해야 한다.
그리고 s와 t를 구하는 공식에서 분모가 0인 경우 두 선은 평행하다는 의미이므로
교점은 존재하지 않다. 분모와 분자 모두 0인 경우 두 선은 동일한 선이다.

위처럼 교과서에 나오는 수학적인 접근을 알고리즘화 한 것으로 Line Sweep이란 방법이 있다

※ 선분의 교차 테스트(line sweep)
두 선분이 있을 때 서로 교차하는지 교차하지 않는지를 검사하는 것은 육안으로는 매우 쉽게 알 수 있지만 컴퓨터가 계산하기에는 그리 쉽지 않다.

선분의 교차 여부를 테스트하기 위해서는 먼저 점들의 방향성에 대한 구분이 먼저 이루어 져야 한다. 평면상에 세 점이 있을 때 그 점들이 어떤 방향을 이루고 있는지를 검사하는 것은 기하 알고리즘의 기초가 되기 때문에 매우 중요한 내용이다. 먼저 a, b, c 세 점이 시계방향으로 존재하는지, 반 시계 방향으로 존재하는지, 그것도 아니면 일직선상에 존재하는 지를 검사하는 방법에 대해 알아보자.

선분의 방향을 확인하기 위해서는 벡터의 외적을 이용해야 한다. 수학적 이론에 대한 설명은 피하기로 하고 실제 이용되는 식만을 소개한다.

먼저 a.x * b.y + b.x * c.y + c.x * a.y - a.x * c.y - b.x * a.y - c.x * b.y의 값을 구하면 세 점 a, b, c가 이루는 삼각형의 넓이*2가 구해진다. 이 삼각형의 넓이가 양수라면 a, b, c는 반시계 방향이며(즉 선분 a-b를 중심으로 점 c가 선분의 왼쪽에 있다는 것을 의미) 넓이가 음수라면 a, b, c는 시계방향이 된다. 만약 넓이가 0이라면 세 점이 삼각형을 이루지 않고 한 직선 상에 존재한다는 뜻이다

이제 이 방법을 이용하여 두 선분 a-b와 c-d가 교차하는지를 쉽게 판별할 수 있다. 다음과 같은 조건식을 보자

※ left(a, b, c) 함수는 위에서 소개된 식에 따라 선분 a-b에 대하여 c가 직선의 왼쪽에 있을 경우 참을 반환하는 함수이다

intersect = (left (a, b, c) xor left (a, b, d)) and (left (c, d, a) xor left (c, d, b));

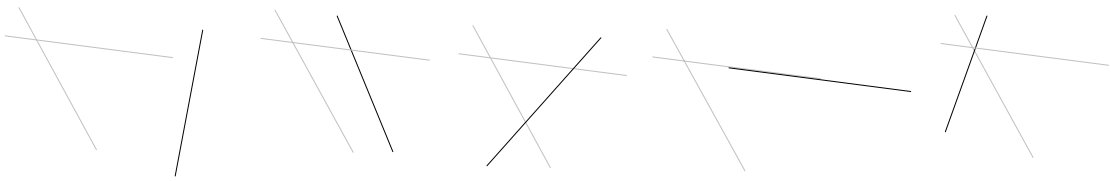
조건이 복잡해 보이는 듯 하지만 그렇지 않다
직선 a-b에 대하여 c는 왼쪽에 d는 오른쪽에 있고 직선 c-d에 대하여 a가 왼쪽에 있고 b가 오른쪽에 있다면 intersect 값이 참이 되고 이것은 두 선분이 교차하고 있음을 의미한다.

3 문제 해설

문제에 교점을 구하는 방법이 소개되어 있으므로 그 방법을 이용하도록 한다.
주의할 점은 주어지는 선이 여러 개라는 점이다. 그러므로 교점을 구하는 식이 여러 번 반복되어야 한다. 실제 교점을 구하는 문제는 아니지만 교점을 구해서 저장해 놓을 필요는 있다. 문제 해결의 과정을 보며 그 이유를 살펴보자.

하나의 선이 주어지고 또 하나의 선이 주어지면 그 선은 만나지 않거나, 한 점에서 만나거나 겹치게 된다. 모든 경우를 생각해 보아야 한다. 겹치는 경우는 교점이 있다고 생각해서는 안 된다. 또 하나의 선이 주어진다면 두 선 모두와 만나지 않는 경우와 한 선과 만나는 경우, 두 선 모두 만나는 경우, 둘 중 어느 선과 겹치는 경우, 또한 세 선이 모두 한 점에서 만나는 경우가 있을 수 있다. 그림으로 표현하면 다음과 같다.

1) 2) 3) 4) 5)



선이 하나씩 추가될 때마다 모든 선과의 관계를 생각하면서 계산을 해야 한다. 따라서 기존에 입력된 선과 만들어진 교점의 좌표를 계속 지니고 있어야 각각의 판단이 가능하다. 이것은 이차원 배열을 이용해 각 선의 두 좌표를 지니고 있고, 또 다른 이차원 배열에 교점의 좌표를 저장해 나가는 식으로 진행을 해야 하는 것이다.

각각의 선에 대해 나머지 선들과의 t, s 값을 구하고 교점의 여부를 따져보아야 하고, 교점을 구한 후에도 지금까지 교점들과 겹치는 것이 있는지 살펴 본 후에 개수를 센다.

```
for ( 모든 선에 대해 ){
      for ( 나머지 선에 대해 ){
            if ( 교점이 없으면 )
                  continue;
            else
                  if (겹치는 교점이 없으면)
                        교점 개수 증가
      }
}
```

소스 코드

```
#include <stdio.h>

int Line[100][8];
int interNum;
double interpoint[100][2];

void CountNode(int LineNum)
{
	FILE *outf=fopen("Output.txt", "w");
	int i, j, k;
	double x1, y1, x2, y2, x3, y3, x4, y4;
	int piled;
	double inter_x, inter_y, t, s, under;

	for (i = 1 ; i < LineNum ; i++)
	{
		x1 = Line[i][0];
		y1 = Line[i][1];
		x2 = Line[i][2];
		y2 = Line[i][3];
		for (j = 0 ; j < i ; j++)
		{
			x3 = Line[j][0];
			y3 = Line[j][1];
			x4 = Line[j][2];
			y4 = Line[j][3];
			under = ((y4-y3)*(x2-x1)-(x4-x3)*(y2-y1));
			if ( under == 0 )
				continue;
			t = (double)((x4-x3)*(y1-y3)-(y4-y3)*(x1-x3))/under;
			s = (double)((x2-x1)*(y1-y3)-(y2-y1)*(x1-x3))/under;
			if ( t < 0.0 || t > 1.0 || s < 0.0 || s > 1.0)
				continue;
			if ( t == 0 && s == 0 )
```

```
                        continue;

                inter_x = x1 + t * (x2-x1);
                inter_y = y1 + t* (y2-y1);
                piled = 0;
                for (k = 0 ; k < interNum ; k++)
                        if (interpoint[k][0] == inter_x && interpoint[k][1] == inter_y
                        {
                                piled = 1;
                                break;
                        }
                if ( piled )
                        continue;

                interpoint[interNum][0] = inter_x;
                interpoint[interNum][1] = inter_y;
                interNum++;
            }
        }
        fprintf(outf, "%d", interNum);
}

int main()
{
        FILE *inf=fopen("Input.txt", "r");
        int LineNum, i;
                fscanf(inf, "%d", &LineNum);

        for ( i = 0 ; i < LineNum ; i++ )
                fscanf(inf, "%d %d %d %d", &Line[i][0], &Line[i][1], &Line[i][2],
&Line[i][3]);

        interNum = 0;
        CountNode(LineNum);
        fclose(inf);
}
```

23 주사위 쌓기 놀이

▸▸ 효율적이고 좋은 프로그램은 주어진 문제를 바로 인식함에서부터 시작된다. 문제 상황에 대한 올바른 인식이야말로 남보다 빠르고 정확한 프로그래밍을 위한 선행조건임은 두말할 나위가 없다. 나는 얼마나 문제 해결능력이 뛰어난지 확인해 보자. 기본적인 프로그래밍 지식만을 가지고 해결할 수 있는 문제이다.

문제 (난이도 ★☆☆☆☆)

준영이는 여러 종류의 주사위를 가지고 쌓기 놀이를 하고 있다. 주사위 쌓기 놀이는 아래에서부터 1번 주사위, 2번 주사위, 3번 주사위, …의 순서로 쌓는 것이다. 주사위의 모양은 모두 크기가 같은 정육면체이며 각 면에는 1부터 6까지의 숫자가 하나씩 적혀 있다. 그러나 보통 주사위처럼 마주보는 면에 적혀진 숫자의 합이 반드시 7이 되는 것은 아니다.

주사위 쌓기 놀이의 규칙은 다음과 같다.

1. 서로 붙어있는 두 개의 주사위에서 아래에 있는 주사위의 윗면에 적혀있는 숫자는 위에 있는 주사위의 밑면에 적혀있는 숫자와 같아야 한다. 다시 말해서, 1번 주사위 윗면의 숫자는 2번 주사위 밑면의 숫자와 같고, 2번 주사위 윗면의 숫자는 3번 주사위 밑면의 숫자와 같아야 한다.
2. 1번 주사위는 마음대로 놓을 수 있다.

이렇게 쌓아 놓으면 긴 사각기둥이 된다. 이 사각기둥에는 4개의 긴 옆면이 있다. 이 4개의 옆면 중에서 어느 한 면의 숫자의 합이 최대가 되도록 주사위를 쌓고자 한다. 이렇게 하기 위하여 각 주사위를 위, 아래를 고정한 채 옆으로 90°, 180°, 또는 270° 돌릴 수 있다. 한 옆면의 숫자의 합의 최대 값을 구하는 프로그램을 작성하시오

프로그램의 이름은 “dice”로 하고 프로그램의 실행시간 1초 이내로 제한한다.

입력 형식

입력 파일의 이름은 "Input.txt"이다.
첫 줄에는 한 번에 집은 주사위의 개수가 입력된다.
그 다음 줄부터는 한 줄에 하나씩 주사위의 종류가 1번 주사위부터 주사위 번호 순서대로 입력된다. 주사위의 종류는 각 면에 적혀진 숫자가 그림 1에 있는 주사위의 전개도의 A, B, C, D, E, F의 순서로 입력된다. 입력되는 숫자 사이에는 빈칸이 하나씩 있으며, 종류가 같은 주사위도 있을 수 있다.

출력 형식

출력 파일의 이름은 "Output.txt"이며, 한 옆면의 숫자의 합이 가장 큰 값을 출력한다.

입력과 출력의 예

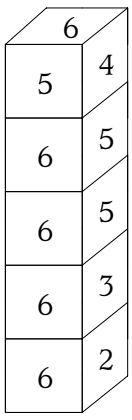
입력의 예("Input.txt")

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

출력의 예("Output.txt")

```
29
```

※ 입력 예의 주사위들을 쌓아서 출력 예와 같은 합이 나오는 모습을 그림으로 나타내면 오른쪽과 같다.



2 문제의 배경 및 관련 이론

● 동적 계획법의 개념(dynamic programming)
동적 계획법은 의사결정상황을 시간적·공간적으로 여러 단계로 나누어 취급한다. 따라서 결정변수의 값도 한꺼번에 결정하는 것이 아니라 각 단계마다 결정된다. 이러한 '단계적 결정'이라는 특성 때문에 다단계계획법(multistage programming)이라고도 불린다.

● 최적성의 원리(principle of optimality)
최적성의 원리란 일련의 의사결정과정에서 어느 단계 이후의 최적의사결정은 그 단계 이전까지의 의사결정과정에는 관계없이 그 단계의 상태만을 바탕으로 하여 이루어져야 한다는 원리이다. 위에서 설명한 동적 계획법은 선형계획법에 비해 현실을 더 잘 반영할 수 있는 반면에 뚜렷한 해법이 없다. 따라서 문제에 따라 해법이 서로 다른데, 모든 경우에 적용되는 개념이 최적성의 원리이다.

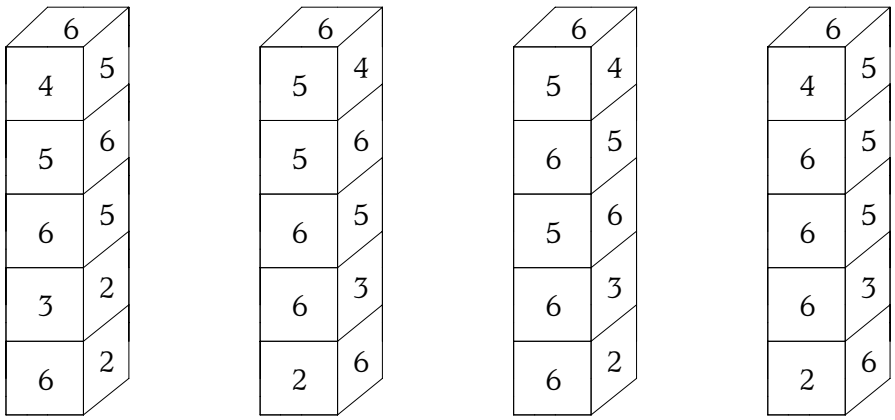
● 순환식(recursive equation)
최적성의 원리를 적용하여 각 단계간의 연관관계를 표현한 수식으로 n단계까지의 이익 = n단계의 이익 + 그 이전(후) 단계까지의 이익이라는 기본개념이 적용된다.

위의 이론적인 지식을 바탕으로 어떤 문제를 동적 계획법으로 처리하는 방법을 단계를 지어보면 다음과 같다.

① 문제가 최적화의 원리가 성립하는지(동적 계획법이 적용가능 한지) 검사한다.
② 부분문제를 정의한다.
③ 우리가 구해야 하는 큰 문제는 어떻게 정의되는지 알아본다.
④ 큰 문제와 작은 문제간의 관계를 찾는다. 즉, 순환식을 정의한다.
⑤ 순환식을 통해 작은 문제들을 어떤 순서로 구해 나갈 것인지를 결정한다.

3 문제 해설

먼저 주어진 문제를 주사위 하나에 대해 문제로 좁혀보자. 주사위가 하나일 때 문제의 핵심만을 간추려 보면 “밑면과 윗면이 정해진 주사위의 옆면에 있는 눈의 수들 중 가장 큰 수 찾기”라고 재정의 할 수 있겠다. 이렇게 좁혀진 문제를 다시 여러 개의 주사위를 쌓는 것으로 확장하면서 “다음 번에 쌓이는 주사위의 밑면의 눈 수는 아래에 있는 주사위의 윗면의 눈 수와 같아야 한다”라는 조건을 추가하면 원래 문제와 같은 문제가 될 것이다. 이제 문제는 해결된 것이다. N 개의 주사위에 대해 밑면과 윗면을 제외한 면에 적힌 숫자들 중 최대인 것들의 합을 구하면 되는 것이다.
특히 “주사위를 옆으로 회전시킬 수 있다”라는 조건은 전혀 필요 없는 조건이다. 이것에 현혹되어 프로그램을 복잡하게 작성하지 않도록 한다.
아래 그림을 보자.



위의 그림들은 예로 주어진 주사위 모양의 변형들이다. 주사위가 어떻게 나열되어 있던 간에 각 주사위의 옆면 중에서 최대의 수를 가지는 옆면은 언제나 동일하다. 따라서 옆면 중에 최대값을 찾아냈다면 굳이 회전시켜가며 값을 더할 필요가 없는 것이다.

소스 코드

```
#include <iostream>
#include <fstream>

using namespace std;

#define MAX 10000
#define ASSERT(x) if(!(x)) \
{
	cout << "Assertion Fail (" << __LINE__ << "):" << #x << endl;
}

const int NOT_REACHED = 0;
int n;
int dice[MAX][6];
int result;

void input()
{
	ifstream inf("Input.txt");
	inf >> n;
	int i, j;
	for(i = 0; i < n; i++)
		for(j = 0; j < 6; j++)
		{
			inf >> dice[i][j];
		}

	inf.close();
}
```

```
int across(int face)
{
	int aface[6] = {5, 3, 4, 1, 2, 0};
	ASSERT(0 <= face && face < 6)
	return aface[face];
}
int sidemax(int n, int top)
{
	int bottom = across(top);
	int max = 0;
	for(int i = 0; i < 6; i++)
	{
		if((i == top) || (i = bottom))
			continue;
		if(max < dice[n][i])
			max = dice[n][i];
	}
	return max;
}

int find(int n, int value)
{
	ASSERT(1 <= value && value <= 6)
	for(int i = 0; i < 6; i++)
	{
		if(dice[n][i] == value)
			return i;
	}
	ASSERT(NOT_REACHED)
	return -1;
}

void process()
{
	result = 0;
```

```
        for(int first = 0; first < 6; first++)
        {
                int prevbottom = dice[0][across(first)];
                int sum = sidemax(0, first);
                for(int i = 1; i < n; i++)
                {
                        int currenttop = find(i, prevbottom);
                        sum += sidemax(i, currenttop);
                        prevbottom = dice[i][across(currenttop)];
                }
                if(sum > result)
                        result = sum;
        }
}
void output()
{
        ofstream ouf("Output.txt");
        ouf << result << endl;
        ouf.close();
}

int main(int argc, char* argv[])
{
        input();
        process();
        output();
        return 0;
}
```

24 이진 트리의 너비 계산

▸▸ 컴퓨터 영역에서 배열과 같은 간단한 자료구조 외에 가장 많이 쓰이는 자료구조라 할 수 있는 이진 트리의 특징과 그 특징을 이용한 이진 트리의 너비 구하기 문제이다.

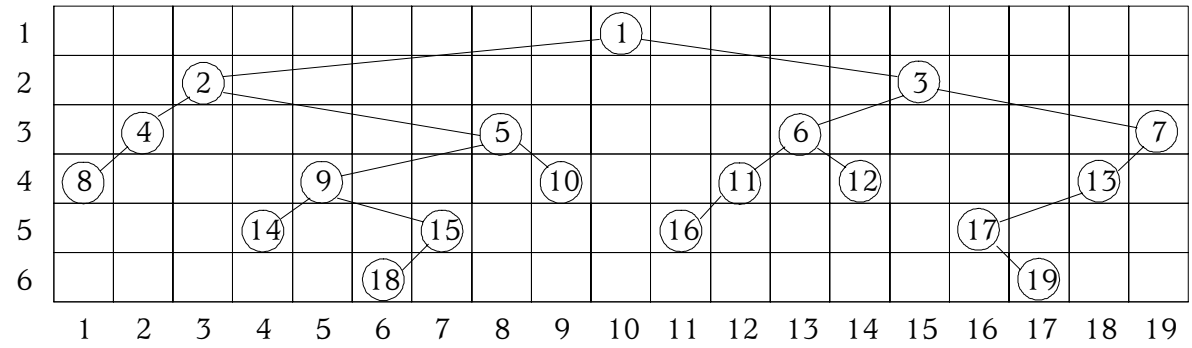
문제 (난이도 ★★★☆☆)

이진 트리를 다음의 규칙에 따라 행과 열에 번호가 붙여진 격자 모양의 틀 속에 그리려고 한다.

① 이진 트리에서 같은 레벨(level)에 있는 노드는 같은 행에 위치한다.
② 한 열에는 한 노드만 존재한다.
③ 임의의 노드의 왼쪽 부트리(left subtree)에 있는 노드들은 해당 노드보다 왼쪽의 열에 위치하고, 오른쪽 부트리(right subtree)에 있는 노드들은 해당 노드보다 오른쪽의 열에 위치한다.
④ 노드가 배치된 가장 왼쪽 열과 가장 오른쪽 열 사이에는 아무 노드도 없이 비어있는 열은 없다.

이와 같은 규칙에 따라 이진 트리를 그릴 때 각 레벨의 너비는 그 레벨에 할당된 노드 중 가장 오른쪽에 위치한 노드의 열 번호에서 가장 왼쪽에 위치한 노드의 열 번호를 뺀 값 더하기 1로 정의한다. 트리의 레벨은 가장 위쪽에 있는 루트 노드가 1이고 아래로 1씩 증가한다.

아래 그림은 어떤 이진트리를 위의 규칙에 따라 그려본 것이다. 첫 번째 레벨의 너비는 1, 두 번째 레벨의 너비는 13, 3번째, 4번째 레벨의 너비는 각각 18이고, 5번째 레벨의 너비는 13이며, 그리고 6번째 레벨의 너비는 12이다.



우리는 주어진 이진 트리를 위의 규칙에 따라 그릴 때에 너비가 가장 넓은 레벨과 그 레벨의 너비를 계산하려고 한다. 아래 그림의 예에서 너비가 가장 넓은 레벨은 3번째와 4번째로 그 너비가 18이다. 너비가 가장 넓은 레벨이 두 개 이상 있을 때는 번호가 작은 레벨을 답으로 한다. 그러므로 이 예에 대한 답은 레벨은 3이고, 너비는 18이다.

임의의 이진트리가 입력으로 주어질 때 너비가 가장 넓은 레벨과 그 레벨의 너비를 출력하는 프로그램을 작성하시오.

프로그램의 이름은 "bwidth"로 하고 실행 시간은 1초를 초과할 수 없다.
부분점수는 없다.

입력 형식

입력 파일의 이름은 "Input.txt"이다.
첫째 줄에는 노드의 개수를 나타내는 정수 N($1 \leq N \leq 1{,}000$)이 주어진다.
다음 N개의 줄에는 각 줄마다 노드 번호와 해당 노드의 왼쪽 자식 노드와 오른쪽 자식 노드의 번호가 순서대로 주어진다. 노드들의 번호는 1부터 N까지로 주어진다. 자식이 없는 경우에는 자식 노드의 번호가 -1로 주어진다. 루트 노드의 번호는 1이다.

출력 형식

출력 파일의 이름은 "Output.txt"이다.
첫째 줄에 너비가 가장 넓은 레벨과 그 레벨의 너비를 순서대로 출력한다. 단, 너비가 가장 넓은 레벨이 두 개 이상 있을 때는 번호가 작은 레벨을 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

```
19
1 2 3
2 4 5
3 6 7
4 8 -1
5 9 10
6 11 12
7 13 -1
8 -1 -1
9 14 15
10 -1 -1
11 16 -1
12 -1 -1
13 17 -1
14 -1 -1
15 18 -1
16 -1 -1
17 -1 19
18 -1 -1
19 -1 -1
```

출력의 예("Output.txt")

```
3 18
```

2 문제의 배경 및 관련 이론

1. 트리의 성질

① 트리 구조에서 한 노드에서 다른 노드로 가는 경로(path)는 유일하다.
트리구조의 임의의 두 노드는 least common ancestor를 갖는다. least common ancestor란 두 노드가 가질 수 있는 공통적인 상위 노드중에서 가장 레벨이 높은 상위 노드이다 트리를 순회하는 경로가 중복됨이 없고, 되돌아감이 없다면 두 노드간의 경로는 반드시 한 노드에서 least common ancestor까지 올라갔다가 다시 다른 노드로 내려오는 유일한 경로밖에 가지지 못한다. 이는 그래프와 트리를 구분하는 아주 중요한 성질이다.

② N 개의 노드를 가지는 트리는 N-1개의 링크를 가진다.
당연한 성질이지만 root를 제외한 모든 노드가 자신의 상위 노드를 향해 하나의 노드를 가지므로 N 개의 노드를 가진 나무는 N-1개의 링크를 가지게 된다.

2. 이진 트리의 순회(Tree Traversal)

트리 구조는 배열과 연결 리스트와 같이 선형적인 구조가 아니기 때문에 구조의 모든 노드를 순회하는 방법이 쉽지 않다. 일반적으로 네 가지의 Traversal 방법이 있는데 이들은 각각 트리의 모든 노드들을 한 번씩 중복없이 순회하는 방법을 제공한다. 트리를 순회하는 방법은 root를 먼저 방문하는 방법(preorder traversal), root를 나중에 방문하는 방법(postorder traversal), root를 중간에 방문하는 방법(inorder traversal), 마지막으로 level 별로 방문하는 방법(level order traversal)이 있다.

모든 traversal 방법들은 재귀적으로 정의하여 사용한다. 주어진 문제를 풀기위해 이용되는 inorder traversal을 예로 들면 다음과 같다.

```
void inorder_traverse(node *t)
{
	if(t != tail)
	{
		inorder_traverse(t->left);
		visit(t);
		inorder_traverse(t->right);
	}
}
```

간단하게 visit(t)가 해당 노드의 이름을 출력하는 것이라 가정하고 설명하면 한 노드에 이르면 그 노드의 왼쪽 노드에 다시 inorder_traversal을 적용하고 해당 노드의 이름을 출력한 후 다시 오른쪽 노드에 대해 inorder_traversal을 적용한다. 각 traversal 방식에 따라 접근된 노드에 대한 처리의 위치가 바뀌는 것만을 제외하면 기본적으로 같은 방식으로 적용된다. 물론 각 노드의 구조는 사용되는 용도에 따라 사용자의 임으로 정의할 수도 있다.

3 문제 해설

이진 트리의 노드들을 그리드 형태로 출력할 때의 배치와 관계된 문제이다. 문제의 조건에서 한 열(세로축)에는 공백이 없으며 중복되어 자리를 차지하는 노드들이 없게 되어있다. 따라서 그리드 상에서 각 노드가 나타날 위치는 트리를 inorder로 순회하면서 각 레벨에서의 하위노드의 수를 계산함으로써 알 수 있다.

1. 문제의 해결

노드의 개수가 1000개 이하이므로 빠른 처리를 위해 배열을 이용하도록 한다.
최대 거리와 레벨을 결정하는 과정을 요약하면 다음과 같다

① Tree를 위한 배열과 거리 값 계산을 위한 배열, level을 위한 배열을 할당한다.

② 파일로부터 Tree의 각 노드정보를 읽어서 Tree 배열에 저장한다.

③ Tree를 inorder traversal 하면서 각 레벨에서 최 좌측의 노드와 최 우측 노드의 위치를 갱신한다.

④ 두 노드의 차가 최대인 값이 나왔을 때 그 차와 레벨을 파일에 쓴다.

이 문제에서 핵심이 되는 노드간의 거리 계산을 위한 조건으로 각 레벨에서 최 좌측, 최 우측 노드가 되는 경우는 다음과 같다

<좌측이 되는 경우>
1. 단말인 좌측 노드의 부모가 그 상위 노드의 좌측자식 일 때
2. 단말인 우측 노드의 부모가 그 상위 노드의 좌측자식이고 좌측 자식 노드가 없을 때

<우측이 되는 경우>
1. 단말인 우측 노드의 부모가 그 상위 노드의 우측자식 일 때
2. 단말인 좌측 노드의 부모가 그 상위 노드의 우측자식 이고 우측 자식 노드가 없을 때

이러한 경우를 고려해서 최 좌측 노드와 최 우측 노드 정보를 갱신한다.

소스 코드

```
#include<stdio.h>

int tree[1000][2], index[1000], minmax[1000][2];
int num;
int answer, ansind;

#define infile "Input.txt"
#define outfile "Output.txt"

void ready()
{
		FILE *fp = fopen(infile, "r");
		int a, i;

		fscanf(fp, "%d", &num);

		for(i = 0; i < num; i++)
		{
			minmax[i][0] = 0x7fffffff;
			minmax[i][1] = 0;
		}

		for(i = 0; i < num; i++)
		{
			fscanf(fp, "%d", &a);
			a--;
			fscanf(fp, "%d %d", &tree[a][0], &tree[a][1]);
			if(tree[a][0] > 0) tree[a][0]--;
			if(tree[a][1] > 0) tree[a][1]--;
		}

		fclose(fp);
}
int seq = 0;
```

```
void traversal(int node, int depth)
{
	if(tree[node][0] != -1) traversal(tree[node][0], depth + 1);
	if(minmax[depth][0] > seq) minmax[depth][0] = seq;
	if(minmax[depth][1] < seq) minmax[depth][1] = seq;
	index[node] = seq++;
	if(tree[node][1] != -1) traversal(tree[node][1], depth + 1);
}

void solveproblem()
{
	int i;
	traversal(0, 0);
	answer = -1;
	for(i = 0; i < num; i++)
		if(answer < minmax[i][1] - minmax[i][0])
		{
			answer = minmax[i][1] - minmax[i][0];
			ansind = i;
		}
}

void outputs()
{
	FILE *fp = fopen(outfile, "w");
	fprintf(fp, "%d %d\n", ansind + 1, answer + 1);
	fclose(fp);
}

int main()
{
	ready();
	solveproblem();
	outputs();

	return 0;
}
```

25 행복한 고민

▸▸ 수학적에서 사용하는 증명 방법의 하나인 수학적 귀납법을 이용하는 문제에 도전해 보자. 초기 조건과 임의의 수 k 일 때 성립한다는 조건을 이용하여 그 다음 단계에서도 성립함을 증명함으로써 언제나 참이 됨을 이용하여 프로그램 작성에 이용한다.

문제 (난이도 ★★★☆☆)

(주)AI 는 요즘 한창 번창해 나가고 있는 중소기업이다. (주)AI에는 N명의 직원을 가지고 있으며 그들은 각각 N번까지의 번호표를 달고 있다. 그리고 (주)AI는 다음과 같은 사원 체계를 가지고 있다.

① 모든 직원은, 직원 번호 1인 사장을 제외하고, 모두 한명의 직속 상사가 있다.
② 모든 직원은 0, 1, 혹은 2명 직원의 직속 상사이다.

요즘 (주)AI의 사장님은 행복한 고민에 빠져있다. 회사가 너무 잘되서 회사를 어떻게 3개로 나눌지를 생각하고 있기 때문이다. 간부 직원들을 모아 놓고 아이디어 회의를 거쳐 최종 안건이 선택되었다. 선택된 안건의 내용은 다음과 같다.
① 모든 직원은 반드시 셋 중 한 회사에 분배되어야 한다.
② 직원과 그의 직속 상사는 같은 회사에 분배되어서는 안된다.
③ 만약 두 직원의 직속 상상가 같다면, 그들도 다른 회사에 분배되어야 한다.
④ 세 개 회사가 모두 효율적으로 운영되게 하기 위해서 N1, N2, N3를 각각 나누어진 회사의 직원수라 할때 (N = N1 + N2 + N3), max(N1, N2, N3) - min(N1, N2, N3)은 가능한 최소이여야 한다. 여기서 max는 3값중 최대값을 min은 3값중 최소값을 나타낸다

(주)AI의 사장이 당신에게 사원들을 효율적으로 나눠서 회사를 분할하기 위한 프로그램의 작성을 요청했다. 주어진 조건에 부합하도록 프로그램을 작성하여라. 실행 파일의 이름은 company로 하며, 실행시간은 1초를 넘어서는 안된다.

제약 조건
① 1 ≤ N ≤ 16,000이다
② 주어진 차가 최소이면 된다. 해가 유일하지 않을수도 있다.
③ 입력은 항상 문제의 조건에 맞는다.

입력 형식

입력파일은 "Input.txt"로 한다.
첫째 줄에 직원수인 N이 입력된다. 다음 N-1개 줄 각각은 두 사원을 나타내는 정수 a, b를 포함하는데, 직원 b가 직원 a의 직속 상사라는 의미이다. a, b는 각각 1이상 N이하인 정수이다.

출력 형식

출력 파일은 "Output.txt"로 한다.
출력은 N명의 사원에 대한 새로운 회사 정보를 출력한다. 첫 번째 정수는 1번 직원이 어느 회사에 분배되는지 나타내고 2번 정수는 2번 직원이 어느 회사에 분배되는지 나타낸다. N명의 사원에 대해 같은 방식으로 출력한다.

입력과 출력의 예

입력의 예("Input.txt")

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

출력의 예("Output.txt")

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

2 문제의 배경 및 관련 이론

1. 수학적 귀납법

수학적 귀납법은 대체로 자연수에 대한 어떤 공식이나 사실을 증명할 때 자주 쓰인다. 수학적 귀납법은 일반적인 귀납법과는 약간 다르다. 추론에는 연역법과 귀납법이 있다. 연역법은 가정에서 부터 논리적인 단계를 거쳐 결론을 유도하는 것이고 귀납법은 여러 구체적인 사례를 조사하여 어떤 결론은 얻는 방법이다.
그런데 수학에서는 이런 귀납법은 별로 쓰이지 않는다. 1+2+3+...+n={n(n+1)}/2 라는 공식을 예를 들어 보자. 이 식에 n=1 을 넣어보면 1={1*2}/2 가 되니깐 공식은 맞고, n=2 를 넣어보면 1+2={2*3}/2 가 되어 공식이 맞다. 계속해서 n=3,4,5,...10 까지 넣어보아도 역시 공식은 성립한다. 처음 10개의 자연수에 대해서 위의 공식이 성립한다고 해서 임의의 자연수에 대해서도 위의 공식이 성립한다고 말할 수는 없다. 11번째부터는 성립하지 않을수도 있기때문이다. 즉 10개의 사례로 부터 1+2+3+...+n={n(n+1)}/2 이 임의의 자연수 n 에 대해서 성립한다고 말할 수는 없다. 누가 n=500 일 때에도 성립하냐고 물어보면 또 n=500 일때 맞는지 확인해야 하는 어려움이 있다. 요약하자면 구체적인 사례를 조사하는 것은 문제를 확인하는 방법이 될 수는 있지만 문제를 증명하는 것은 될 수 없다.

그렇다면 1+2+3+...+n={n(n+1)}/2 이 모든 자연수 n 에 대해서 성립한다는 것을 증명해보자. 우선 n=1 일때 공식이 잘 성립한다는 것을 알아야 한다. 그리고 "n=k 일 때 이 공식이 성립하면(이것을 가정하고) n=k+1 일때도 이 공식이 성립한다(이것을 결론으로 얻어야한다)" 라는 사실을 증명해야 한다.

(증명) n=1 일 때 좌변은 1 이고 우변은 (1*2)/2=1 이므로 위 공식은 n=1 일 때 성립한다. 이제 다음 사실만 증명하면 된다.

"n=k 일 때 이 공식이 성립하면 n=k+1 일때도 이 공식이 성립한다"

이 사실을 증명하려면 가정에 의해 n=k 일 때
이 공식이 성립하므로1+2+3+...+k={k(k+1)}/2 라는 사실을 이용할 수 있다. 이제 결론을 얻기 위해 1+2+3+...+k+(k+1)={(k+1)(k+2)}/2 가 성립하는지만 살펴보면 된다.

좌변은 가정을 이용하면 1+2+3+...+k+(k+1) = {1+2+3+...+k}+(k+1) = {k(k+1)}/2 +(k+1) = {k(k+1)}/2 +2*(k+1)/2 = {k(k+1)+2(k+1)}/2 = {(k+1)(k+2)}/2 가 되어 n=k+1 일때도 위 공식이 성립함을 증명하였다.(증명끝)

3 문제 해설

이 문제는 max(N1, N2, N3)-min(N1, N2, N3)이 0이나 1로 나눌 수 있다.
수학적 귀납법을 증명하고 증명을 따라 recursive하게 구현하면 된다. 노드 수가 3의 배수이면 각 분할된 회사에 포함되는 직원 수가 같게, 그렇지 않으면 많은 쪽이 가장 작은 쪽보다 하나 많게 나눌 수 있음을 보인다.

induction을 이용하여 루트가 속한 그룹은 가장 많은 그룹의 번호를 가지게 할 수 있음을 보인다. 회사 번호가 같은 그룹끼리는 서로 회사 번호를 서로 바꿀 수 있음에 유의한다. 노드 수가 1, 2, 3으로 작은 경우는 가능하다. 노드 수가 n인 경우에 가능함을 보이기 위해서 그 보다 작은 경우에 성립한다고 하자.
자식의 수가 2인 경우는 각각을 귀납적으로 나눈다. 왼쪽 부 트리와 오른쪽 부 트리에 속한 노드 수를 3으로 나눈 나머지가 0, 1, 2인 경우 가능함을 확인한다. 자식이 하나인 경우에도 마찬가지임을 확인한다.

[N이 3의 배수일경우]

N = 3k라 하면
1. k = 1(N = 3) 일 때 3개의 회사에 나누는 것이므로 성립
2. k = k (N = 3k)일 때 성립한다 가정
3. k = k + 1 일 때
   N = 3(k + 1) = 3k + 3
   이므로 k일때 성립한다는 가정과 초기 조건으로 나눌 수 있으므로 성립(증명 끝)

[N이 3k + 2(나머지가 2인경우)]

위에서 3K일 때 성립하는 것을 보였으므로 2명을 세 개의 회사에 나눌 경우
(1, 1, 0), (1, 0, 1), (0, 1, 1) 이렇게 나눌수가 있다. 그 때 max(N1, N2, N3) - min(N1, N2, N3) = 1이 되므로 가장 작은 차라 할 수 있다.

[N이 3k + 1(나머지가 1인경우)]

나머지가 2인 경우와 마찬가지로 (1, 0, 0), (0, 1, 0), (0, 0, 1) 세 경우 중 한 가지이므로 역시 max(N1, N2, N3) - min(N1, N2, N3) = 1이 되므로 역시 가장 작은 차의 값이다.

위 경우를 통합해서 살펴보면 세 개의 회사로 나누었을 때 최대와 최소의 차이는 0 또는 1뿐임을 알 수 있으며, 차가 최소가 되는 경우는 1이라 할 수 있다.(0인 경우는 어차피 똑같이 나누어지므로)

여기서 추가적으로 생각해 줘야 하는 조건이 있다. (주)AI의 사원 체계상 회사의 모든 직원들은 최상위의 상관으로 사장을 가지게 되어 있다. 따라서 N = 3k + 1, N = 3k + 2 인 경우 모두 사장이 있는 쪽이 최대값을 가져야 한다. 문제의 제약 사항에서 입력은 항상 문제의 조건에 맞는다고 했으므로 4명인 경우를 생각해 보면 자신의 직속 상관과는 같은 회사에 속할수 없다는 조건에 의해 가장 마지막 번호를 가지는 사원은 사장과 같은 그룹내지는 사장이 속한 그룹과 같은 사원수를 가지는 그룹에 속해야 함을 알 수 있다.

이 조건들에 맞게 프로그램을 작성한다면 문제를 해결할 수 있다.

소스 코드

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define filein "input.txt"
#define fileout "output.txt"
#define maxn 16001
#define mare 1000000

long nf[maxn], fiu[maxn][2], final[maxn], col[maxn], howmany[maxn][
perm[maxn][3];
long psol[3], pact[3], i, j, k, n, p, hm, c, nmax, nmin;

void df(long node)
{
	long fs, fd;

	for(k = 0; k < 3; k++)
		perm[node][k] = k;
	if(nf[node] == 0)
	{
		col[node] = 0;
		howmany[node][0] = 1;
	}
	else if(nf[node] == 1)
	{
		fs = fiu[node][0];
		df(fs);

		for(k = 0; k < 3; k++)
			howmany[node][k] = howmany[fs][k];

		hm = mare;
		c = 0;
		for(k = 0; k < 3; k++)
		{
			if(howmany[node][k] < hm && col[fs] != k)
			{
				hm = howmany[node][k];
				c = k;
			}
		}
```

```
                howmany[node][c]++;
                col[node] = c;
        }
        else // nf[node] == 2
        {
                fs = fiu[node][0];
                fd = fiu[node][1];
                df(fs);
                df(fd);

                for(i = 0; i < 3; i++)
                        psol[i] = i;
                for(i = 0; i < 3; i++)
                        pact[0] = i;

                for(j = 0; j < 3; j++)
                        if(j != i)
                        {
                                pact[1] = j;

                                for(k = 0; k < 3; k++)
                                {
                                        if(k != i && k != j)
                                        {
                                                for(p = 0; p < 3; p++)
                                                        howmany[node][p]                =
howmany[fs][p];
                                                for(p = 0; p < 3; p++)
                                                        howmany[node][pact[p]]          +
howmany[fd][p];

                                                hm = mare;
                                                for(p = 0; p < 3; p++)
                                                        if(howmany[node][p] < hm)
                                                                hm = howmany[node][p]

                                                c = 0;
                                                for(p = 0; p < 3; p++)
                                                        if(howmany[node][p] > c)
                                                                c = howmany[node][p];

                                                nmax = 0;
                                                nmin = 0;

                                                for(p = 0; p < 3; p++)
                                                        if(howmany[node][p] == hm)
                                                                nmin++;
```

```
                                        else
                                                nmax++;

                                    if(c - hm <= 1 && (nmin == 3 |
(nmin == 2 &&
                                        (howmany[node][col[fs]] != hm|
howmany[node][pact[col[fd]]]
                                    != hm)) || (nmin == 1 &&
(howmany[node][col[fs]] != hm
&& howmany[node][pact[col[fd]]] != hm))) && (pact[col[fd]] !
                        col[fs]))
                                    {
                                        for(p = 0; p < 3; p++)
                                                psol[p] = pact[p];
                                    }
                                }
                        }
                }

                for(k = 0; k < 3; k++)
                        perm[fd][k] = psol[k];

                for(k = 0; k < 3; k++)
                        howmany[node][k] = howmany[fs][k];

                for(k = 0; k < 3; k++)
                        howmany[node][perm[fd][k]] += howmany[fd][k];

                hm = mare;
                c = 0;
                for(k = 0; k < 3; k++)
                        if(howmany[node][k] < hm && col[fs] != k &&
perm[fd][col[fd]] != k)
                        {
                                hm = howmany[node][k];
                                c = k;
                        }
                howmany[node][c]++;
                col[node] = c;
        }
}

void df2(long node)
{
        long p[3];

        //save current permutation
```

```
	for(k = 0; k < 3; k++)
		p[k] = pact[k];

	//modify current permutation
	for(k = 0; k < 3; k++)
		psol[k] = pact[perm[node][k]];
	for(k = 0; k < 3; k++)
		pact[k] = psol[k];

	final[node] = pact[col[node]];

	if(nf[node] > 1)
		df2(fiu[node][1]);
	if(nf[node] > 0)
		df2(fiu[node][0]);

	//restore the original permutation
	for(k = 0; k < 3; k++)
		pact[k] = p[k];
}

int main()
{
	FILE *in, *out;
	in = fopen(filein, "r");
	out = fopen(fileout, "w");

	fscanf(in, "%ld", &n);
	for(k = 1; k < n; k++)
	{
		fscanf(in, "%ld %ld", &i, &j);
		nf[j]++;
		fiu[j][nf[j] - 1] = i;
	}
	df(1);

	for(k = 0; k < 3; k++)
		pact[k] = k;
	df2(1);

	for(i = 1; i <= n; i++)
		fprintf(out, "%ld ", final[i] + 1);

	return 0;
}
```

〈부록〉

Test Data

중등_01(container)

[Input_01]
5 3
a b b b c
[Output_01]
4
a b b
a b c
b b b
b b c

[Input_02]
10 2
a a a b b b c c d e
[Output_02]
13
a a
a b
a c
a d
a e
b b
b c
b d
b e
c c
c d
c e
d e

[Input_03]
10 3
a b c d e f g g g h
[Output_03]
64
a b c
a b d
a b e
a b f
a b g
a b h
a c d
a c e
a c f
a c g
a c h
a d e
a d f
a d g
a d h
a e f
a e g
a e h
a f g
a f h
a g g
a g h
b c d
b c e
b c f
b c g
b c h
b d e
b d f
b d g
b d h
b e f
b e g
b e h
b f g
b f h
b g g
b g h

c d e
c d f
c d g
c d h
c e f
c e g
c e h
c f g
c f h
c g g
c g h
d e f
d e g
d e h
d f g
d f h
d g g
d g h
e f g
e f h
e g g
e g h
f g g
f g h
g g g
g g h

[Input_04]
40 7
a a a a a a a a a a a a a a a a a a a a b
b b b b b b b b b c c c c c c c c c c c

a a a a a b c
a a a a a c c
a a a a b b b
a a a a b b c
a a a a b c c
a a a a c c c
a a a b b b b
a a a b b b c
a a a b b c c
a a a b c c c
a a a c c c c
a a b b b b b
a a b b b b c
a a b b b c c
a a b b c c c
a a b c c c c
a a c c c c c
a b b b b b b
a b b b b b c
a b b b b c c
a b b b c c c
a b b c c c c
a b c c c c c
a c c c c c c
b b b b b b b
b b b b b b c
b b b b b c c
b b b b c c c
b b b c c c c
b b c c c c c
b c c c c c c
c c c c c c c

[Output_04]
36
a a a a a a a
a a a a a a b
a a a a a a c
a a a a a b b

중등_02(ballgame)

[Input_01]
4
1
8
13
24
[Output_01]
-1
-1
-1
3

[Input_02]
15
25
40
55
99
102
157
909
1024
1300
2000
2345
5600
8999
10000
100000
[Output_02]
1
1
-1
2
13
13
3
3
1
5
2
55
1
2
1

[Input_03]
5
89
144
244
2584
28657
[Output_03]
-1
3
1
-1
-1

중등_03(spy)

[Input_01]
4 10
[Output_01]
225568798

[Input_02]
3 40
[Output_02]
9202600068524372703278082352971

[Input_03]
5 30
[Output_03]
309357158069565341012824268937 6

[Input_04]
9 37
[Output_04]
145516173548114477029645099998606851963351218

[Input_05]
7 40
[Output_05]
356714494836474846364271944010429540721066 00

[Input_06]
8 10
[Output_06]
410333440536

[Input_07]
8 20
[Output_07]
1934134063853856881106420

[Input_08]
6 15
[Output_08]
8175951659117794

[Input_09]
9 40
[Output_09]
1599461518145464215858214546017769580320423082614

[Input_10]
10 30
[Output_10]
1570758468134776640589671769311535924

[Input_11]
10 35
[Output_11]
1439344278655987394597734695032153350743471 4

[Input_12]
10 40
[Output_12]
1357740187499993272317100164273173020889986718729 00

중등_04(cutvertex)

[Input_01]
6
1 3
2 4
3 2
1 2
3 5
4 5
[Output_01]
2 3

[Input_02]
15
1 3
2 5
2 4
3 4
5 6
6 7
6 9
7 8
9 10
10 12
10 11
12 13
13 11
14 15
14 9
[Output_02]
2 3 4 5 6 7 9 10

[Input_03]
20
1 3
1 4
2 3
3 5
4 2
5 7
5 9
6 7
7 8
7 2
8 10
9 11
10 1
10 4
11 13
11 6
12 11
13 14
13 8
14 15
[Output_03]
11 13

중등_05(digitpuzzle)

[Input_01]
4
6 4 8 6
6 10 3 5
[Output_01]
3
0 3 0 3
0 1 1 2
3 3 2 0
3 3 0 0

[Input_02]
5
80 54 50 23 15
10 175 20 10 7
[Output_02]
46
0 45 18 10 7
6 46 2 0 0
4 46 0 0 0
0 23 0 0 0
0 15 0 0 0

[Input_03]
10
1 2 5 6 6 8 10 5 2 7
5 10 2 4 6 10 4 8 2 1
[Output_03]
2
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 2 0
2 0 0 0 0 1 0 2 0 0
2 1 0 0 0 1 0 2 0 0
1 0 0 0 0 1 2 2 0 0
0 1 0 0 2 2 1 2 0 0
0 2 2 1 2 2 1 0 0 0
0 2 0 1 0 2 0 0 0 0
0 2 0 0 0 0 0 0 0 0
0 2 0 2 2 1 0 0 0 0

[Input_04]
18
1 13 3 7 11 9 13 2 6 7 3 8 10 11 5 9
17 4 7 9 2 10 4 3 8 9 11 2 9 12 13 1
[Output_04]
1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0
1 0 1 1 0 0 0 0 1 1 1 0 1 1 1 1 1 0
1 1 0 1 0 0 0 0 1 0 1 0 1 1 1 1 0 0
1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0
1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0
1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0
1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0
1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0
1 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 0 0
1 0 0 1 0 1 0 0 1 1 1 0 1 1 0 0 0 0
1 0 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 0

중등_06(postorder)

[Input_01]
13
a - ((b - c) - d) - e
[Output_01]
a - (b - c - d) - e

[Input_02]
15
a - ((b - c) - (d - e))
[Output_02]
a - (b - c - (d - e))

[Input_03]
7
a - (b + e)
[Output_03]
a - b + e

[Input_04]
7
a - (b - e)
[Output_04]
a - (b - e)

[Input_05]
19
a - b - c * e - (d + e - f) * z - q
[Output_05]
a - b - c * e - (d + e - f) * z - q

[Input_06]
35
(((((p + (q * e - w) * q - ((((z * e)))))))))
[Output_06]
p + (q * e - w) * q - z * e

[Input_07]
51
((q + w) * e - r) * (t + y * u - i * o * (p - a) * s) / (d + f) + (g + h) * (j * k * l)
[Output_07]
((q + w) * e - r) * (t + y * u - i * o * (p - a) * s) / (d + f) + (g + h) * j * k * l

[Input_08]
71
((z - x) * (c * v)) * (b / n / (m / a / (s * d)) * f * (((g))) * h / j / k) / l + ((q)) / (w * e * r) / t * (y - u)
[Output_08]
(z - x) * c * v * b / n / (m / a / (s * d)) * f * g * h / j / k / l + q / (w * e * r) / t * (y - u)

중등_07(hanoipath)

[Input_01]
5
30
[Output_01]
2:2->3

[Input_02]
6
53
[Output_02]
1:3->1

[Input_03]
40
30203
[Output_03]
1:3->1

[Input_04]
60
10203428
[Output_04]
3:3->1

[Input_05]
45
1000000000
[Output_05]
10:3->1

[Input_06]
60
4310446000
[Output_06]
5:1->2

[Input_07]
60
25646560863
[Output_07]
1:2->3

[Input_08]
60
256888960512
[Output_08]
10:3->2

[Input_09]
60
9007199256761296
[Output_09]
5:1->2

[Input_10]
60
576460752303423478
[Output_10]
2:2->1

중등_08(convexhull)

[Input_01]
7
1 3
4 6
3 4
10 12
5 6
11 6
4 2

[Output_01]
4
4 2
11 6
10 12
1 3

[Input_02]
100
38 53
71 146
10 72
70 12
98 42
52 126
48 73
86 164
65 99
93 109
98 18
48 74
46 1
35 161
41 85
89 13
95 158
1 6
61 114
3 69
32 156
81 125
16 117
83 101
82 33
56 91
88 85
46 172
71 49
54 129
1 129
12 174
9 189
21 95
16 112
11 163
81 99
76 51
32 139
84 154
31 181
65 74
75 173
76 109
74 7
40 36
15 87
14 65
93 168
35 150
94 42
17 82
91 141
57 65
68 54

75 178
44 6
96 161
46 75
5 165
23 120
87 157
89 101
34 119
53 3
82 132
64 108
91 62
36 8
76 46
24 11
33 96
42 135
81 128
99 102
49 38
28 90
62 101
67 58
2 90
88 40
49 37
23 63
45 23
79 92
3 69
34 159
56 145
75 92
65 60
16 115
30 45
64 177
10 160
40 179
10 117
74 197
25 108
18 176
88 86

[Output_02]

12
46 1
74 7
89 13
98 18
99 102
96 161
93 168
74 197
9 189
5 165
1 129
1 6

중등_09(box)

[Input_01]

8
1 1
5 1
5 6
3 6
3 2
8 2
8 4
1 4

[Output_01]

3

[Input_02]

42
20 10
20 100
50 100
50 110
40 110
40 130
90 130
90 110
70 110
70 100
80 100
80 80
90 80
90 100
110 100
110 50
90 50
90 70
80 70
80 50
60 50
60 40
70 40
70 20
30 20
30 40
40 40
40 50
50 50
50 70
40 70
40 60
10 60
10 90
40 90
40 80
60 80
60 140
30 140
30 120
100 120
100 10

[Output_02]

4

[Input_03]

300
1 1
120 1
120 279
239 279
239 173
237 173
237 270
298 270
298 73
61 73
61 202
19 202
19 148
271 148
271 13
184 13
184 38
85 38
85 162
26 162
26 263
189 263
189 18
221 18
221 67

40 67
40 91
187 91
187 48
228 48
228 84
295 84
295 291
16 291
16 172
217 172
217 31
46 31
46 282
7 282
7 103
212 103
212 97
202 97
202 143
174 143
174 40
53 40
53 244
6 244
6 155
215 155
215 24
115 24
115 112
78 112
78 113
161 113
161 52
59 52
59 294
138 294
138 17
129 17
129 57
30 57
30 56
152 56
152 247
52 247
52 196
149 196
149 4
253 4
253 76
263 76
263 63
211 63
211 5
148 5
148 42
99 42
99 255
146 255
146 151
248 151
248 83
49 83
49 126
281 126
281 206
60 206
60 265
133 265
133 70
21 70
21 287
159 287
159 149
44 149
44 216
276 216
276 110
188 110
188 286
45 286
45 90
182 90
182 295
33 295
33 234
260 234
260 272
87 272
87 284
235 284
235 266
283 266
283 221

214 221
214 278
206 278
206 10
247 10
247 182
38 182
38 236
167 236
167 107
216 107
216 249
124 249
124 3
108 3
108 65
282 65
282 191
201 191
201 124
257 124
257 261
293 261
293 87
156 87
156 94
240 94
240 127
69 127
69 159
125 159
125 212
242 212
242 59
23 59
23 139
199 139
199 85
195 85
195 283
259 283
259 71
251 71
251 276
39 276
39 203
145 203
145 298
8 298
8 301
47 301
47 156
3 156
3 226
123 226
123 174
62 174
62 224
246 224
246 99
56 99
56 292
288 292
288 175
171 175
171 163
11 163
11 274
224 274
224 239
270 239
270 6
170 6
170 92
31 92
31 241
20 241
20 213
264 213
264 168
29 168
29 137
262 137
262 235
106 235
106 86
94 86
94 108
96 108
96 140
297 140
297 146
4 146
4 243

280 243
280 68
141 68
141 49
186 49
186 78
210 78
210 89
89 89
89 145
277 145
277 208
58 208
58 131
81 131
81 115
118 115
118 281
229 281
229 12
279 12
279 273
136 273
136 262
225 262
225 34
227 34
227 79
197 79
197 19
163 19
163 136
185 136
185 27
151 27
151 28
135 28
135 61
32 61
32 147
243 147
243 74
134 74
134 69
101 69
101 256
116 256
116 135
267 135
267 53
294 53
294 166
70 166
70 98
196 98
196 181
93 181
93 88
24 88
24 246
181 246
181 271
153 271
153 186
91 186
91 227
100 227
100 222
109 222
109 100
43 100
43 171
180 171
180 128
122 128
122 176
236 176
236 293
88 293
88 142
193 142
193 154
233 154
233 277
51 277
51 192
1 192

[Output_03]

2340

중등_10(store)

[Input_01]
30
10
34000 24000 223000 325000 189000 7
217000 119000 12000 45000 189000 9
75000 17000 254000 329000 312000 1
45000 317000 75000 218000 39000 13
14000 5000 75000 37000 218000 75000
[Output_01]
10 19
45000

[Input_02]
100
60
134000 124000 223000 325000 18
75000 21000 189000 12000 45000 28
196000 5000 317000 24000 29000 1
11000 45000 317000 75000 218000 3
132000 14000 5000 75000 37000 21
75000 34000 24000 223000 325000 18
75000 217000 119000 12000 45000 1
96000 75000 17000 254000 229000 31
11000 4000 317000 317000 21000 3
132000 14000 5000 75000 37000 21
75000 4000 24000 23000 325000 18
7000 217000 19000 12000 45000 18
96000 75000 7000 24000 329000 1
1000 45000 317000 75000 218000 3
32000 14000 5000 75000 37000 21
75000 234000 12000 323000 9000 2
12000 23000 98000 34000 124000
[Output_02]
6 21 27 30 36 43 57 60 73 81 87 90
75000

[Input_03]
125
30
34000 14000 223000 325000 189000
2000 19000 12000 5000 289000 1
5000 17000 24000 9000 12000 1000
37000 75000 21000 39000 12000
5000 75000 3000 218000 75000 3
24000 23000 325000 189000 75000 21
19000 12000 45000 19000 96000 7
1000 254000 229000 312000 11000
317000 317000 21000 39000 13000 1
5000 75000 37000 218000 7000 20
24000 23000 325000 218000 275
217000 19000 119000 45000 89000 9
75000 7000 24000 32000 12000 1
45000 117000 75000 28000 39000 11
14000 5000 75000 37000 28000 11
234000 12000 323000 9000 360000 1
23000 98000 34000 112000 34000 2
223000 325000 189000 7000 217
119000 12000 45000 19000 96000 7
17000 254000 39000 312000 11000 4
317000 27000 218000 39000 132000 14
[Output_03]
9 17 24 39 77 92 96 109
12000

중등_11(ambulance)

```
[ Input_01 ]
 5 5
 4 4
   1 0 1 1 1
   1 1 1 0 1
   0 0 0 1 1
   1 1 0 1 0
   1 1 1 1 1
[ Output_01 ]
 7
```

```
[ Input_02 ]
 25 25
 24 24
 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0
 1 0 0 1 0
 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0
 0 0 1 1 1
 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0 1 1
 0 1 1 0 1
 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
 1 1 0 0 0
 0 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0
 0 1 1 1 1
 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1
 1 1 0 0 1
 1 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0
 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0
 1 1 1 1 0
 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0
 1 1 0 1 1
 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1
 1 0 1 1 1
 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0
 1 0 0 1 0
 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1
 0 0 0 0 1 1 1
 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0 1 1
 0 1 1 0 1
 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
 1 1 0 0 0
 0 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0
 0 1 1 1 1
 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1
 1 1 0 0 1
 1 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0
 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0
 1 1 1 1 0
 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0
 1 1 0 1 1
 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1
 1 0 1 1 1
 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1
 1 1 0 0 1
 1 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0
 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0
 1 1 1 1 0
 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0
 1 1 0 1 1
 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1
 1 0 1 1 1
[ Output_02 ]
 17
```

중등_12(nemo)

[Input_01]

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

[Output_01]

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

[Input_02]

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

[Output_02]

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

[Input_03]

```
10
5 1 1 1 1 1
4 2 2 1 1
3 3 2 2
3 1 3 1
3 3 1 4
4 1 1 1 1
3 1 1 1
3 1 1 1
3 1 1 2
2 1 2
4 1 3 1 1
3 2 2 1
2 5 1
3 1 1 1
3 4 1 1
2 3 1
2 3 1
3 3 1 1
3 1 1 3
2 2 2
```

[Output_03]

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

[Input_03]

```
15
3 2 1 2
```

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

[Output_03]

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

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

중등_13(coin)

[Input_01]
1000 953
[Output_01]
0
1
0
1
2

[Input_02]
100000 1753
[Output_02]
1964
1
0
1
2

[Input_03]
10000 6753
[Output_03]
64
1
0
1
2

[Input_04]
20000 15953
[Output_04]
80
1
0
1
2

[Input_05]
50000 35923
[Output_05]
279
3
0
1
2

[Input_06]
8000 923
[Output_06]
139
3
0
1
2

[Input_06]
700 623
[Output_06]
1
0
2
1
2

중등_14(ball)

[Input_01]
5
5 4 3 2 1
[Output_01]
5 4 3 2 1

[Input_02]
10
7 5 3 1 4 9 2 6 10 8
[Output_02]
7 3 9 10 4 8 5 6 1 2

[Input_03]
20
12 3 11 4 1 16 5 6 15 2 7 10 14 9 8 17 18 19 20
[Output_03]
12 11 3 15 14 19 6 9 10 18 4 1 20 17 5 7 13 2 8

[Input_04]
30
23 22 21 12 3 24 30 11 4 26 1 16 15 2 7 10 29 14 9 25 8 28 13 17 18 27 20
[Output_04]
23 22 21 24 26 12 30 6 29 1 2 25 27 28 14 4 15 17 18 3 11 20 9 13 5 19 8 7

[Input_05]
40
31 32 23 40 22 21 38 12 3 24 30 11 4 26 1 16 5 39 33 6 15 37 2 7 10 29 14 35 9 25 8 28 13 34 17 18 36 19 27 20
[Output_05]
31 32 40 38 30 16 37 23 21 24 25 26 22 28 39 29 34 4 9 36 33 27 3 1 15 10 20 14 13 19 17 18 11 7 8 5 6 2

[Input_06]
55
50 45 31 32 23 40 44 22 21 54 38 1 24 30 55 11 51 4 26 1 16 5 43 39 3 15 37 2 7 10 46 29 14 35 53 9 25 8 13 34 47 17 18 52 49 36 19 27 20 41 48
[Output_06]
50 45 32 44 38 55 16 37 53 52 18 0 0 40 0 54 0 21 3 51 24 43 39 46 30 33 0 26 1 15 47 0 35 0 23 49 0 7 25 42 34 0 36 0 14 17 48 0 27 0 6 22

중등_15(pattern)

[Input_01]
5
10101
01110
10111
01011
11111
2
10
01

[Output_01]
10

[Input_02]
10
0000000001
1000001100
0100010000
0010100000
0001000111
0000101000
0000001000
0010010000
1111110000
0010010001
4
1000
0100
0010
0001

[Output_02]
4

[Input_03]
25
01101000010110100000101100
01101111100110111111000011
10011000101010100010100 11
10011111000110111110001100
01101110110110111101101101
00101011100010101011000101
11110110001111110110011111
11101001101111101110111 11
11111001101111110000011111
00000110110000000111100000
01101000010110100000101101
01101111100110111111001101
10101000101010100010101 01
01101111000110111110001101
01101110110110111101101101
00101011100010101011000101
11111011001111101100111 11
11111011101111101110111 11
11111000001111100000111 11
10010011110000000111100000
01101000010110100000101101
01101111100110111111000110
10011000101010100010101 10
01101111000110111110001001
01101110110110111101101001
4
0011
1100
1100
0011

[Output_03]
3

[Input_03]
50
00100110001110101010101010101010

101010101010
000111100000
0010011000110101010101001010100110
0110100001011010000101100100010000
101010101011
010000101101
0010011111111100100101000100010111
1
0110111110011011111000110011011111
010101010101
011111000110
1111111000111110000011111000001111
1001100010101010001010110100110001
001111100000
010001010110
1111111000111011101101101011011101
0110111100011011110001001011011110
011101101101
011110001001
0010001110001010111000101001010111
0110111011011011101101001011011101
010111000101
011101101001
0010011000111110110011111111101110
1010101010101010101010101010101010
110110011111
101010101010
0101100110111111011101111111101001
0
1010101010101010101010010101001101
110111011111
101010101011
0101100110111111000001111111111001
0
1101010100101001001010001000101111
110000011111
010101010101
0101011011000000011110010000000111
0
1001111100011011110001100100111110
000111100000
011110001100
0101100001011010000100100011010010
0110111011011011101101101011011101
010000101101
011101101101
1010111110011011111000100011011101
1100001110001010111000101001010111
011000001101
010000011111
1010100010101010001011111101010001
1100011000111110110011111111101100
011000010101
110000011111
0110111100011011110011111111011110
1
1111100110111110111011111111010011
111111001101
110111111111
0110111011011011101101001000011101
1100000110111110000011111111110011
011000101101
110000011111
0010101110001010111000001001010111
1100011011001000111100000000001101
011000000101
000000011111
1111101100111111011001111111111110
1
0110100001001000000101101011010000
110110011111
010000101101
1111101110111111011101111111111111
0
0110111110011011111001101011011111
110111011111
011111001101
1111100000111111000001111111111000
0
1010100010101010001010101101010001
110000011111
010001010101
1001001111000000011110000010010011
1
0110111100011011110001101011011110

011110001101
01101110110110111011010000010011101
011101101101
00111000000010101110011000010011111
010111000101
11111000001111101100111110001001110
110110011111
11111111101111101110111111111110111
110111011111
11111000001111100000111111111110000
110000011111
10011000010000001111000000100100111
000111100000
00001001100010100001011010110100000
010001100011
01001001100010111111000110011011111
011111100011
10111111111111010001010110100110001
010001111111
01111111100010111110001001011011110
011111100011
01101111100010111011010010110111101
011101100011
5
00100
00100
00100
11111
11111

[Output_03]

14

중등_16(network)

[Input_01]
3 20
0 10 20
10 0 30
20 30 0
[Output_01]
1 3
2

[Input_02]
5 7
0 4 8 1 7
4 0 16 2 5
8 16 0 3 6
1 2 3 0 12
7 5 6 12 0
[Output_02]
1 2 5
3 4

[Input_03]
10 90
0 23 45 32 56 89 56 85 34 55
23 0 85 23 12 98 56 35 78 32
45 85 0 43 21 95 67 43 98 43
32 23 43 0 54 23 24 56 19 18
56 12 21 54 0 32 75 63 48 28
89 98 95 23 32 0 34 41 52 60
56 56 67 24 75 34 0 34 21 12
85 35 43 56 63 41 34 0 10 8
34 78 98 19 48 52 21 10 0 15
55 32 43 18 28 60 12 8 15 0
[Output_03]
1 5 6 7 9
2 3 4 8 10

[Input_04]
15 80
0 23 45 32 56 39 56 45 34 55 31 21
11 31
23 0 55 23 12 28 56 35 78 32 25 14
42 51
45 55 0 43 21 25 67 43 18 43 3 33
31 63
32 23 43 0 54 23 24 56 19 18 4 34
51 75
56 12 21 54 0 32 75 63 48 28 5 28
15 83
39 28 25 23 32 0 34 41 52 60 16 66
16 94
56 56 67 24 75 34 0 34 21 12 7 37
73 67
45 35 43 56 63 41 34 0 10 08 81 81
67 34
34 78 18 19 48 52 21 10 0 15 9 19
31 43
55 32 43 18 28 60 12 08 15 0 10 59
42 29
31 25 3 4 5 16 7 81 9 10 0 71
23 30
21 14 33 34 28 66 37 81 19 59 71 0
77 50
7 4 14 68 9 3 4 15 21 73 34 29
23 7
11 42 31 51 15 16 73 67 31 42 23 77
0 3
31 51 63 75 83 94 67 34 43 29 30 50
3 0
[Output_04]
1 3 5 6 7 9 11 12 13
2 4 8 10 14 15

중등_17(square)

[Input_01]
2
1
1
1 1
[Output_01]
1 1

[Input_02]
3
1 1
0 1
1 1
1 1 1
1 1 1
[Output_02]
2 1
1 2

[Input_03]
6
1 0 1 1 1
1 1 1 1 1
0 1 0 0 0
0 1 1 1 1
0 1 1 0 0
0 1 1 0 0
1 1 0 1 1 1
1 1 0 1 1 1
1 1 0 1 0 0
0 1 1 1 0 0
0 1 0 1 0 0
[Output_03]
2 2
1 5

[Input_04]
10
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 1 1 1 0
1 1 1 1 1 1 1 1 1 1
0 1 1 1 0 1 1 0 0
0 1 1 0 1 1 1 0 0
1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 0 0
1 1 1 1 0 0 1 1 1 1
0 1 0 1 1 1 1 1 1 0
[Output_04]
7 1
5 4
4 8
3 10
2 21
1 45

[Input_05]
20
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

```
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1    12 3
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    10 15
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    8 3
1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0    7 8
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    6 7
0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0    5 24
0 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0    4 42
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    3 61
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1    2 109
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    1 200
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0
0 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0
1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1
0 1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1
1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0
1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1
0 1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
```

[Output_05]

```
14 1
```

중등_18(train)

[Input_01]
7
35 40 50 10 30 45 60
2
[Output_01]
240

[Input_02]
7
35 40 50 10 30 45 60
1
[Output_02]
155

[Input_03]
16
35 40 50 10 30 45 60 70 20 10 30 20 35 45 20
5
[Output_03]
555

[Input_04]
50
45 65 85 90 75 65 40 30 20 10 60 20 50 80 90 85 75 70 25 30 70 60 20 10 35 35 25 70 80 65 55 80 30 20 20 25 50 10 75 65 55 35 20 40 90 70 20
18
[Output_04]
1895

[Input_05]
100
30 70 60 20 10 15 35 35 25 70 80 65 80 30 20 20 25 40 50 45 65 85 90 75 40 30 20 10 60 20 30 50 80 90 85 75 25 45 65 85 90 75 65 40 30 20 10 60 30 50 80 90 85 75 70 25 30 70 60 20 15 35 35 25 70 80 65 55 80 30 20 20 40 50 10 75 65 55 35 20 40 90 70 20 75 65 55 35 20 40 90 70 20
33
[Output_05]
4830

[Input_06]
200
45 65 85 90 75 65 40 30 20 10 60 20 50 80 90 85 75 70 25 30 70 60 20 10 35 35 25 70 80 65 55 80 30 20 20 25 50 10 75 65 55 35 20 40 90 70 20 10 65 55 35 20 40 90 70 20 30 70 60 20 15 35 35 25 70 80 65 55 80 30 20 20 40 50 45 65 85 90 75 65 40 30 20 10 20 30 50 80 90 85 75 70 25 30 70 60 10 15 35 35 25 70 80 65 55 80 30 20 25 40 50 10 75 65 55 35 20 40 90 70 10 75 65 55 35 20 40 90 70 20 30 70 20 10 15 35 35 25 70 80 65 55 80 30 20 25 40 50 45 65 85 90 75 65 40 30 10 60 20 30 50 80 90 85 75 70 25 10 65 55 35 20 40 90 70 20 10 75 65 55 20 40 90 70 20
50
[Output_06]
7470
[Input_07]
400
45 65 85 90 75 65 40 30 20 10 60 20

65 40 30 85 75 70 25 10 75 65 55 35
20 40 90 70 20 45 65 85 90 75 65 40
30 20 10 30 70 60 20 10 15 35 35 25
70 80 65 10 15 35 35 20 25 40 50 45
65 85 90 75 65 40 30 20 10 60 20 75
65 40 30 85 75 70 25 30 70 60 20 10
15 35 35 25 70 80 65 10 15 35 35 20
25 40 50 45 65 85 90 75 65 40 30 20
10 60 20 75 65 40 30 85 75 70 25 30
70 60 20 10 15 35 35 25 70 80 65 10
15 35 35 20 25 40 50 30 70 60 20 10
15 35 35 25 70 80 65 10 15 35 35 20
25 40 50 45 65 85 90 75 65 40 30 20
10 60 20 75 65 40 30 85 75 70 25 10
75 65 55 35 20 40 90 70 20 10 75 35
20 40 90 40 90 70 20 45 65 85 90 75
65 40 30 20 10 60 20 75 65 40 30 85
75 70 25 30 70 60 20 10 15 35 35 25
70 80 65 10 15 35 35 20 25 40 50 60
20 10 15 35 35 25 70 80 65 55 80 35
35 25 70 70 20 20 25 30 70 60 20 10
15 35 35 25 70 80 65 10 15 35 35 20
25 40 50 45 65 85 90 75 65 40 30 20
10 60 20 75 65 40 30 85 75 70 25 45
65 85 90 75 65 40 30 20 10 30 70 60
20 10 15 35 35 25 70 30 70 60 20 10
15 35 35 25 70 80 65 10 15 35 35 20
25 40 50 30 70 60 20 10 15 35 35 25
70 80 65 10 15 35 35 20 25 40 50 45
65 85 90 75 65 40 30 20 10 60 20 75
65 40 30 85 75 70 25 10 75 65 55 35
20 40 90 70 20 10 75 35 20 40 90 40
90 70 20 100

[Output_07]

13960

[Input_08]

800

30 70 60 20 10 15 35 35 25 70 80 65
15 35 35 20 25 40 50 60 20 10 15 35
25 70 80 65 55 80 35 35 25 70 70 20
25 30 70 60 20 10 15 35 35 25 70 80
10 15 35 35 20 25 40 50 45 65 85 90
75 65 40 30 20 10 60 20 75 65 40 30
85 75 70 25 45 65 85 90 75 65 40 30
20 10 30 70 60 20 10 15 35 35 25 70
30 70 60 20 10 15 35 35 25 70 80 65
10 15 35 35 20 25 40 50 30 70 60 20
10 15 35 35 25 70 80 65 10 15 35 35
20 25 40 50 45 65 85 90 75 65 40 30
20 10 60 20 75 65 40 30 85 75 70 25
10 75 65 55 35 20 40 90 70 20 10 75
35 20 40 90 40 90 70 20 45 65 85 90
75 65 40 30 20 10 60 20 75 65 40 30
85 75 70 25 10 75 65 55 35 20 40 90
70 20 10 75 35 20 40 90 40 90 70 20
45 65 85 90 75 65 40 30 20 10 60 20
75 65 40 30 85 75 70 25 30 70 60 20
10 15 35 35 25 70 80 65 10 15 35 35
20 25 40 50 60 20 10 15 35 35 25 70
80 65 55 80 35 35 25 70 70 20 20 25
30 70 60 20 10 15 35 35 25 70 80 65
10 15 35 35 20 25 40 50 45 65 85 90
75 65 40 30 20 10 60 20 75 65 40 30
85 75 70 25 45 65 85 90 75 65 40 30
20 10 30 70 60 20 10 15 35 35 25 70
30 70 60 20 10 15 35 35 25 70 80 65
10 15 35 35 20 25 40 50 30 70 60 20
10 15 35 35 25 70 80 65 10 15 35 35
20 25 40 50 45 65 85 90 75 65 40 30
20 10 60 20 75 65 40 30 85 75 70 25
10 75 65 55 35 20 40 90 70 20 10 75
35 20 40 90 40 90 70 20 45 65 85 90
75 65 40 30 20 10 60 20 75 65 40 30
85 75 70 25 10 75 65 55 35 20 40 90
70 20 45 65 85 90 75 65 40 30 20 10
30 70 60 20 10 15 35 35 25 70 80 65

10 15 35 35 20 25 40 50 45 65 85 90
75 65 40 30 20 10 60 20 75 65 40 30
85 75 70 25 30 70 60 20 10 15 35 35
25 70 80 65 10 15 35 35 20 25 40 50
45 65 85 90 75 65 40 30 20 10 60 20
75 65 40 30 85 75 70 25 30 70 60 20
10 15 35 35 25 70 80 65 10 15 35 35
20 25 40 50 30 70 60 20 10 15 35 35
25 70 80 65 10 15 35 35 20 25 40 50
45 65 85 90 75 65 40 30 20 10 60 20
75 65 40 30 85 75 70 25 10 75 65 55
35 20 40 90 70 20 10 75 35 20 40 90
40 90 70 20 45 65 85 90 75 65 40 30
20 10 60 20 75 65 40 30 85 75 70 25
30 70 60 20 10 15 35 35 25 70 80 65
10 15 35 35 20 25 40 50 60 20 10 15
35 35 25 70 80 65 55 80 35 35 25 70
70 20 20 25 30 70 60 20 10 15 35 35
25 70 80 65 10 15 35 35 20 25 40 50
45 65 85 90 75 65 40 30 20 10 60 20
75 65 40 30 85 75 70 25 45 65 85 90
75 65 40 30 20 10 30 70 60 20 10 15
35 35 25 70 30 70 60 20 10 15 35 35
25 70 80 65 10 15 35 35 20 25 40 50
30 70 60 20 10 15 35 35 25 70 80 65
10 15 35 35 20 25 40 50
45 65 85 90 75 65 40 30 20 10 60 20
65 40 30 85 75 70 25
250
[Output_08]
34290

중등_19(police)

[Input_01]
2 10
3 4 10
4 2 5
5 1 7
3 5 8
4 5 2
1 3 2
6 5 3
6 4 10
6 2 8
3 4 2
[Output_01]
-1
-1
12
12
12
12
12
12
12
12

[Input_02]
4 15
1 2 10
1 3 8
3 2 3
1 4 3
1 3 6
2 1 2
5 1 3
5 2 4
6 2 10
6 3 5
6 1 8
2 4 5
3 4 2
1 6 2
[Output_02]
-1
-1
-1
14
12
8
11
11
11
11
11
11
10
12
14

[Input_03]
7 25
1 2 10
1 3 8
3 2 3
1 4 3
1 3 6
2 1 2
5 1 3
5 2 4
6 2 10
6 3 5
6 1 8
2 4 5
3 4 2

1 6 2
4 7 2
3 7 5
6 7 4
8 5 2
8 3 1
8 1 4
2 7 3
5 7 2
4 8 12
5 2 1
6 4 3
[Output_03]
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
14
14
14
16
14
14
14
13
13
14
14
[Input_04]
7 30
4 5 2
1 3 2
6 5 3
6 4 10
6 2 8
3 4 10
3 4 2
4 2 5
5 1 7
3 5 8
8 9 1
7 10 8
10 8 7
11 7 7
12 4 3
3 12 5
11 5 4
6 8 4
12 6 10
8 12 12
12 5 5
11 7 1
9 7 15
9 2 18
5 3 15
5 6 8
9 3 7
8 7 6
5 2 10
2 3 6
4 7 8
12 2 20
5 8 10
1 2 4
10 1 15

[Output_04]

```
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
31
31
31
31
31
31
31
25
25
25
25
25
25
25
25
25
```

중등_20(lock)

[Input_01]
10
-5 0 0 -7 5 8 -4 3 0 -1
0 -5 3 -1 -2 0 0 0 -8 4
[Output_01]
113

[Input_02]
10
5 0 0 0 5 8 -4 3 0 -1
0 -5 3 7 -2 0 0 0 -8 4
[Output_02]
115

[Input_03]
20
-2 8 4 0 0 4 0 5 0 -8 5 0 0 0 5 8 -4 0 -1
7 5 4 -10 4 0 0 0 -2 -3 0 -5 3 7 -2 0 0 -8 4
[Output_03]
311

[Input_04]
20
0 4 0 5 0 -8 5 0 -5 -2 8 4 0 0 5 8 -4 0 -1
0 0 0 -2 -3 0 -5 3 7 -2 0 0 0 -8 4 7 4 -10 4
[Output_04]
296

[Input_05]
30
1 2 3 -4 0 0 -8 -9 4 3 0 0 4 0 5 0 -8 0 -5 -2 8 4 0 0 5 8 -4 3 0 -1
0 0 0 0 -10 -9 -8 7 6 5 0 0 0 -2 -3 -5 3 7 -2 0 0 0 -8 4 7 5 4 -10 4
[Output_05]
533

[Input_06]
30
0 -8 -9 4 3 0 0 4 0 5 0 -8 5 0 -5 -2 4 0 0 5 1 2 3 -4 0 8 -4 3 0 -1
-3 0 -5 3 7 -2 0 0 0 -8 4 7 5 4 -10 4 0 0 0 -10 -9 -8 7 6 5 0 0 0 -2
[Output_06]
483

[Input_07]
40
5 4 0 0 -1 -2 0 0 2 -4 0 -8 -9 4 3 0 4 0 5 0 -8 5 0 -5 -2 8 4 0 0 5 1 2 3 0 8 -4 3 0 -1
-5 -4 0 0 1 -2 0 0 -2 4 -3 0 -5 3 7 0 0 0 -8 4 7 5 4 -10 4 0 0 0 0 -10 -8 7 6 5 0 0 0 -2
[Output_07]
516

[Input_08]
40
-9 4 3 0 0 4 0 5 0 -8 5 0 -5 -2 8 4 0 5 1 2 3 -4 0 8 -4 3 0 -1 5 4 0 0 -1 0 0 2 -4 0 -8
7 5 4 -10 4 0 0 0 0 -10 -9 -8 7 6 5 0

0 -2 -5 -4 0 0 1 -2 0 0 -2 4 -3 0 -5
3 7 -2 0 0 0 -8 4
[Output_08]
565

[Input_09]
50
8 0 0 0 0 7 0 0 0 0-9 4 3 0 0 4 0 5
-8 5 0 -5 -2 8 4 0 0 5 1 2 3 -4 0 8
3 0 -1 5 4 0 0 -1 -2 0 0 2 -4 0 -8
0 -4 -3 -2 -3 0 -2 -3 -4 -5 7 5 4 -
4 0 0 0 0 -10 -9 -8 7 6 5 0 0 0 -2
-4 0 0 1 -2 0 0 -2 4 -3 0 -5 3 7 -2 0
0 -8 4
[Output_09]
599

[Input_10]
50
0 5 1 2 3 -4 0 8 -4 3 0 -1 0 0 0-9 4
0 0 4 0 5 0 -8 5 0 -5 -2 8 4 0 5 4 0
-1 -2 0 0 2 -4 0 -8 8 0 0 0 0 7 0
-8 7 6 5 0 0 0 -2 -5 -5 7 -4 0 0 1 -2
0 -2 4 -3 0 -4 -3 -2 -3 0 -2 -3 -4 5
-10 4 0 0 0 0 -10 -9 0 -5 3 7 -2 0 0
-8 4
[Output_10]
546

[Input_11]
100
0 5 1 2 3 -4 0 8 -4 3 0 -1 0 0 0-9 4
0 0 0 0 0 0 7 0 0 5 1 2 3 -4 0 8 -4 3
-1 0 0 0-9 4 3 0 0 4 0 5 0 -8 5 0 -5
8 4 0 5 4 0 0 -1 -2 0 0 2 -4 0 -8 8 0
0 0 7 0 0 4 0 5 0 -8 5 0 -5 -2 8 4 0
5 4 0 0 -1 -2 0 0 2 -4 0 -8 8
0 -2 4 -3 0 -4 -3 -2 -3 0 -2 -3 -4 5
-10 4 0 0 0 0 0 -10 -9 0 -5 3 7 -2 0 0
-8 4 -8 7 6 1 -2 0 0 -2 4 -3 0 -4
-2 -3 0 -2 -3 -4 5 4 -10 4 0 0 0 0 0 -
-9 0 -5 3 7 -2 0 0 0 -8 4 -8 7 6 5 0
0 -2 -5 -5 7 -4 0 0 1 -2 0 5 0 0 0
-5 -5 7 -4 0 0
[Output_11]
1213

중등_21(economy)

[Input_01]
50 3
1 8 4
2 10 3
3 4 5
4 6 8
5 9 7
6 6 5
7 7 4
8 5 5
9 3 4
10 1 2
[Output_01]
5 5 9
26

[Input_02]
40 12
1 8 4
2 10 3
3 4 5
4 6 7
5 9 6
6 6 4
7 7 3
8 5 5
9 3 4
10 1 2
11 3 2
12 4 5
[Output_02]
1 1 9 1 9 9 4 1 1 1 1 1
81

[Input_03]
200 10
1 10 3
2 5 5
3 4 5
4 6 7
5 4 5
6 9 6
7 6 4
8 7 3
9 3 4
10 1 2
11 3 2
12 8 4
[Output_03]
1 7 7 1 7 1 1 7 1 1
65

[Input_04]
200 5
1 15 3
2 20 5
3 4 8
5 14 5
10 10 4
[Output_04]
1 3 3 1 1
53

[Input_05]
300 8
1 40 1
2 60 3
3 80 5
4 100 7
5 120 9

6 140 11
7 160 13
8 180 15
[Output_05]
7 7 7 7 7 4 4 4
533

[Input_06]
400 12
1 40 2
2 60 4
3 80 6
4 100 8
5 120 10
6 140 12
7 160 14
8 180 16
9 200 18
10 220 20
11 240 22
12 260 24
[Output_06]
1 18 18 1 18 18 1 18 120 1 41 120
1840

[Input_07]
400 12
1 40 1
2 60 2
3 80 3
4 100 4
5 120 5
6 140 6
7 160 7
8 180 8
9 200 9
10 220 10
11 240 11
12 260 12

[Output_07]
1 9 9 1 9 9 1 9 9 1 180 120
1920

[Input_08]
20 12
1 5 2
2 7 3
3 10 7
4 8 3
5 9 5
6 6 1
7 1 1
8 2 3
9 10 2
10 11 3
11 13 4
12 4 3
[Output 08]
2 2 2 2 2 2 2 1 1 1 1 1
91

중등_22(crossline)

[Input_01]
6
12 4 5 15
43 23 12 87
23 17 53 24
5 9 0 2
77 2 3 12
2 55 2 33
[Output_01]
1

[Input_02]
10
12 4 5 15
43 23 12 87
23 17 53 24
5 9 0 2
77 2 3 12
2 55 2 33
45 20 5 10
60 10 3 7
4 32 50 11
8 8 50 50
[Output_02]
13

[Input_03]
8
43 23 12 87
23 17 53 24
77 2 3 12
2 55 2 33
45 20 5 10
60 10 3 7
4 32 50 11
8 8 50 50
[Output_03]
8

[Input_04]
8
11 11 33 33
1 1 44 44
90 90 2 2
5 5 8 8
7 7 50 50
20 20 10 10
80 80 15 15
25 25 35 35
[Output_04]
0

[Input_05]
10
11 11 33 33
1 5 44 44
1 1 3 3
5 5 8 8
7 7 50 50
1 5 4 0
80 80 15 15
25 25 35 35
0 0 45 45
7 3 52 58
[Output_05]
4

[Input_06]
10

51 11 39 93
1 5 47 84
1 1 3 3
5 9 88 8
70 7 50 50
1 5 4 0
60 8 55 15
25 25 35 35
0 0 45 45
7 3 25 89
[Output_06]
8

[Input_07]
10
15 9 88 8
70 7 50 50
1 5 4 0
60 8 55 15
125 25 35 35
0 0 45 45
17 3 25 89
151 11 39 93
1 5 47 84
1 1 3 3
[Output_07]
8

중등_23(dice)

[Input_01]
15
1 3 6 2 4 5
4 1 6 5 2 3
3 1 2 4 6 5
5 6 4 1 3 2
1 3 6 2 4 5
4 1 6 5 2 3
3 1 2 4 6 5
5 6 4 1 3 2
1 3 6 2 4 5
4 1 6 5 2 3
2 3 1 6 5 4
3 1 2 4 6 5
5 6 4 1 3 2
1 3 6 2 4 5
4 1 6 5 2 3
[Output_01]
87

[Input_02]
30
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 2 3 4 5 6
1 2 4 5 3 6
6 5 4 1 2 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
[Output_02]
174

[Input_03]
45
2 3 5 4 3 6
6 3 5 2 4 1
4 2 1 3 5 6
5 4 1 2 6 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
6 3 5 2 4 1
4 2 1 3 5 6
5 4 1 2 6 3
1 2 3 4 5 6

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

[Output_03]

256

[Input_04]

45
4 2 1 3 5 6
5 4 1 2 6 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
4 2 1 3 5 6
5 4 1 2 6 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
1 2 3 4 5 6
1 2 4 5 3 6
6 5 4 1 2 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 2 3 4 5 6
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 6 5 4 2 3
1 2 4 5 3 6
6 5 4 1 2 3
1 2 3 4 5 6

6 1 2 3 4 5
1 6 5 4 2 3
6 5 4 1 2 3
3 4 2 1 5 6
6 1 2 3 4 5
[Output_04]
258

중등_24(bwidth)

[Input_01]
19
1 2 3
2 4 5
3 6 7
4 8 -1
5 9 10
6 11 12
7 13 -1
8 -1 -1
9 14 15
10 -1 -1
11 16 -1
12 -1 -1
13 17 -1
14 -1 -1
15 18 -1
16 -1 -1
17 -1 19
18 -1 -1
19 -1 -1
[Output_01]
3 18

[Input_02]
30
1 2 3
2 4 5
3 6 7
4 8 9
5 10 11
6 12 13
7 13 14
8 15 16
9 17 18
10 19 20
11 21 22
12 23 24
13 25 26
14 27 28
15 29 30
16 -1 -1
17 -1 -1
18 -1 -1
19 -1 -1
20 -1 -1
21 -1 -1
22 -1 -1
23 -1 -1
24 -1 -1
25 -1 -1
26 -1 -1
27 -1 -1
28 -1 -1
29 -1 -1
30 -1 -1
[Output_02]
5 32

[Input_03]
35
1 2 3
2 4 -1
3 -1 5
4 -1 6
5 -1 7
6 8 9
7 -1 10
8 -1 11
9 12 -1
10 13 14
11 15 16

12 -1 17
13 18 -1
14 19 -1
15 20 -1
16 21 22
17 23 -1
18 24 -1
19 25 -1
20 -1 26
21 -1 27
22 -1 28
23 29 30
24 31 32
25 -1 -1
26 33 -1
27 -1 -1
28 -1 34
29 -1 -1
30 35 -1
31 -1 -1
32 -1 -1
33 -1 -1
34 -1 -1
35 -1 -1
[Output_03]
5 31

[Input_04]
40
1 2 3
2 -1 4
3 5 -1
4 -1 6
5 7 -1
6 -1 8
7 9 -1
8 -1 10
9 11 -1
10 -1 12
11 13 -1
12 -1 14
13 15 -1
14 -1 16
15 17 -1
16 -1 18
17 19 -1
18 -1 20
19 21 -1
20 -1 22
21 -1 -1
22 23 24
23 25 26
24 27 28
25 -1 -1
26 -1 -1
27 -1 -1
28 29 30
29 31 32
30 33 34
31 35 -1
32 36 -1
33 37 -1
34 38 -1
35 39 -1
36 40 -1
37 -1 -1
38 -1 -1
39 -1 -1
40 -1 -1
[Output_04]
2 40

[Input_05]
50

1 2 3
2 -1 4
3 5 -1
4 -1 6
5 7 -1
6 -1 8
7 9 -1
8 -1 10
9 11 -1
10 -1 12
11 13 -1
12 -1 14
13 15 -1
14 -1 16
15 17 -1
16 -1 18
17 19 -1
18 -1 20
19 21 -1
20 -1 22
21 -1 -1
22 -1 23
23 24 -1
24 -1 25
25 26 -1
26 -1 27
27 28 -1
28 -1 29
29 30 -1
30 -1 31
31 32 -1
32 -1 33
33 34 -1
34 -1 35
35 36 -1
36 -1 37
37 38 -1
38 -1 39
39 40 -1
40 -1 41
41 42 -1
42 -1 43
43 44 -1
44 -1 45
45 46 -1
46 -1 47
47 48 -1
48 -1 49
49 50 -1
50 -1 -1
[Output_05]
2 50

중등_25(company)

[Input_01]
6
2 1
3 2
4 2
5 3
6 3
[Output_01]
2 1 2 3 1 3

[Input_02]
15
2 1
3 1
4 2
5 3
6 3
7 4
8 5
9 6
10 6
11 7
12 8
13 9
14 9
15 10
[Output_02]
3 1 2 3 1 3 2 2 2 2 1 3 1 3 1

[Input_03]
20
2 1
3 2
4 3
5 4
6 4
7 5
8 5
9 6
10 7
11 8
12 8
13 9
14 10
15 12
16 13
17 14
18 14
19 15
20 17
[Output_03]
1 2 1 2 1 3 3 2 1 2 3 1 2 1 2 3 2 3 3

[Input_04]
40
2 1
3 2
4 2
5 3
6 3
7 4
8 4
9 5
10 5
11 7
12 7
13 9
14 9
15 11
16 11
17 13
18 13
19 15
20 15
21 16
22 18

23 18
24 20
25 20
26 21
27 22
28 23
29 24
30 25
31 25
32 27
33 27
34 28
35 29
36 30
37 33
38 34
39 35
40 39
[Output_04]
2 3 2 1 3 1 2 3 2 1 3 1 3 1 2 1 1 2 1
2 1 1 2 1 3 3 3 3 2 3 1 2 2 1 1 3 1 2

[Input_05]
50
2 1
3 2
4 3
5 3
6 4
7 4
8 5
9 5
10 7
11 8
12 9
13 10
14 10
15 11
16 11
17 12
18 12
19 13
20 13
21 14
22 14
23 15
24 15
25 16
26 16
27 17
28 17
29 18
30 18
31 20
32 20
33 22
34 22
35 24
36 24
37 26
38 26
39 28
40 28
41 30
42 30
43 31
44 31
45 32
46 32
47 33
48 33
49 40
50 41
[Output_05]
2 3 2 3 3 1 2 1 2 1 2 1 2 2 3 3 2 2 3
3 3 1 2 1 2 1 3 1 3 2 2 2 1 3 1 3 1 1
2 1 1 3 1 3 3 1 3 3

편찬 책임 : 김동민 (경기도교육정보연구원 교육연구사)
기 획 : 황철중 (경기도교육정보연구원 교육연구원)
지도 위원 : 박용순 (경기도교육정보연구원 교육연구원)
박종현 (경기도교육정보연구원 교육연구원)
집필 위원 : 강지성 (경기과학고등학교 교사)
정혜진 (경기과학고등학교 교사)
이병희 (아주대학교 강사)

정보올림피아드 알고리즘 자료집

(중·고등부용)

발 행 일	2004. 12. 30.
발 행 인	김 주 일
발 행 처	경기도교육정보연구원 경기도 수원시 장안구 조원동 536-19 전화 : 031-249-0734