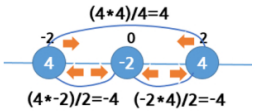
환영합니다. 임베디드 전문가 그룹 월텍에서 운영하는 비공개 온라인 채점서버입니다.

문제 A3: [SOL]전하량

실행시간 제한: 1 Sec 메모리사용 제한: 128 MB
제출: 2 통과: 100%
[제출]

문제 설명

총 N 개의 구슬이 있으며, 각 구슬은 '+전하' 와 '-전하' 량을 가지고 있습니다.
그리고 각 구슬이 들어 갈 수 있는 M 개의 홀이 있습니다.
홀이 여러 개일 경우 구슬은 한 개 또는 모든 홀에 배치 될 수 있으며, 구슬의 양이 홀의 양보다 많거나 적을 수도 있습니다.
전하를 가지고 있는 각 구슬은 서로 힘을 주고 받으며, 두 구슬 사이에 작용하는 힘은 " (전자1의 크기 x 전자2의 크기) / 거리 " 로 계산합니다.
서로 다른 부호의 전하(-와 +, +와 -)를 가진 두 구슬은 서로 끌어당기고, 같은 부호의 전하(-와-, +와+)를 가진 두 구슬은 서로 밀어냅니다.
이때 당기는 힘(인 력)과 밀어내는 힘(척력)의 크기가 같으면 구슬은 움직이지 않습니다.
모든 구슬들이 제자리에 멈춰있도록 하면서, 최대 몇 개의 구슬을 배치할 수 있는지를 찾아야 합니다.



구슬 1이 (4)의 크기를 갖고, -2 위치의 홀에 있고, 구슬 2가 (-2)의 크기를 갖고, 0 위치의 홀에 있고, 구슬 3이 (4)의 크기를 갖고, 2 위치의 홀에 있는 그림입니다.
위의 예제는 세 개의 구슬이 서로에게 작용하는 인력과 척력이 같으므로 모든 구슬이 제자리에 멈춰 있게 됩니다.
구슬1 입장에서 구슬 1, 2 사이의 힘은 (4*(-2))/2 = -4가 되고, 구슬 1, 3 사이의 힘은 (4*4)/4 = 4가 되어 인력과 척력이 같은 힘이므로 제자리에 있게 되고,
구슬2 입장에서 구슬 2, 1 사이의 힘은 ((-2)*4)/2 = -4가 되고, 구슬 2, 3 사이의 힘은 ((-2)*4)/2 = -4가 되는데 둘 다 인력이지만 방향이 다르기 때문에 같은 힘이 되어 제자리에 있게 되고,
구슬 3 입장에서는 구슬 3, 1 사이의 힘은 (4*4)/4 = 4가 되고, 구슬 3, 2 사이의 힘은 (4*(-2))/2 = -4가 되어 인력과 척력이 같은 힘이므로 제자리에 있게 되어 3개 구슬이 모두 제자리에 멈춰 있게 됩니다.

---

* C 솔루션은 다음 코드를 참조하시오.

```
// 구슬 배치하기
#if 01
#include <stdio.h>
#define MAX (10)
int N, M; // N 구슬의수, M 홀의 수
int g[MAX];
int h[MAX];
int v[MAX];  // 구슬을 중복배치하지 않기 위해 사용
int sol;

int cnt;
int list[MAX];              // 홀에 배치된 구슬 상태
void printList(int L)
{
    int i;
    printf("%2d : ", ++cnt);
    for (i=1; i < L; i++)
    {
        printf("%2d ", list[i]);
    }
    printf("\n");
}

void input(void)
{
    int i;

    scanf("%d %d", &N, &M);
    for (i=1; i <= N; i++)
    {
        scanf("%d", &g[i]);
    }
    for (i=1; i <= M; i++)
    {
        scanf("%d", &h[i]);
    }
}

// 홀을 정렬
// 홀의 위치를 정렬해서 고정해 놓아야 (=> depth로 사용해라!)
// "거리" 계산이나 "방향" 확인에 유용하게 사용할 수 있음
void sort(void)
{
    int iy, jx, temp;
    for (iy=1; iy <= M - 1; iy++)
```

```
    {
        for (jx=iy + 1; jx <= M; jx++)
        {
            if (h[iy] > h[jx])  //오름차순 정렬
            {
                temp = h[iy];
                h[iy] = h[jx];
                h[jx] = temp;
            }
        }
    }
}

int getPower(void)
{
    int i, j;
    int sum[MAX] ={ 0 };
    int power;

    for (i=1; i <= M - 1; i++)
    {
        for (j=i + 1; j <= M; j++)
        {
          //if (구슬(i)전하량 == 0 || 구슬(j)전하량 == 0) continue;
            if (list[i] == 0 || list[j] == 0) continue;
            //(홀(i)에 배치된 구슬전하량 * 홀(j)에 배치된 구슬전하량) / (위치(j)-위치(i))
            power = (list[i] * list[j]) / (h[j] - h[i]);
            sum[i] += power;
            sum[j] += -power;
        }
    }
    // 제자리 확인
    for (i=1; i <= M; i++) {
        if (sum[i] != 0) return 0;
    }
    return 1;
}
// L: 홀의 번호, C: 배치된 구슬의 개수
void DFS(int L, int C )
{
    int i;
    // 지금까지 배치된 구슬 수 + 앞으로 최대 배치될 수 있는 구슬 수 <= sol
    //if (C + M - L + 1 <= sol) return;
    if (L > M)
    {
        //printList(L);
        if (C > sol && getPower() == 1)  // 모두 제자리에 있으면 1 리턴
        {
            sol = C;
        }
        return;
    }
    for (i=1; i <= N; i++)  // i : 구슬의 번호
    {
        if (v[i] == 0)
        {
            v[i] = 1; list[L] = g[i];
            DFS(L + 1, C + 1);
            v[i] = 0; list[L] = 0;
        }
    }
    DFS(L + 1, C);  // 구슬을 배치하지 않음
}
int main(void)
{
    int tc, T;
    //T = 1;
    scanf("%d", &T);
    for (tc=1; tc <= T; tc++)
    {
        input();
        sort();
        sol = 1; // max를 찾는 것이므로 min으로 초기화
        DFS(1, 0);
        printf("%d\n", sol);
    }
    return 0;
}
#endif
```

입력 설명

첫 줄에 테스트 케이스의 수 T가 입력되고, 두 번째 줄에 구슬의 수 N과 홀의 수 M이 입력됩니다.
세 번째 줄에는 구슬의 전하량의 값이, 네 번째 줄에는 홀의 위치가 공백으로 구분되어 하나씩 입력됩니다.
구슬의 수와 홀의 수 N 과 M 은 모두 3 이상 7 이하이며, 홀은 일직선에 위치해야 하고, 위치는 -9,999,999부터 9,999,999의 값을 갖습니다.
구슬의 전하 량의 크기는 -32,768 ~ +32,768 범위이며 0일 수 없습니다.

출력 설명

모든 구슬들이 제자리에 멈춰있도록 하면서, 최대 몇 개의 구슬을 배치할 수 있는지 출력합니다.

입력 예시

```
5
5 4
-3 5 -4 7 15
-5 0 3 6
3 4
7 8 9
1 8 -9 0
4 5
9 8 3 2
-2 2 0 6 -6
6 5
3 -1 -2 10 6 5
7 0 -3 -6 -9
7 7
84 -186 62 -147 -196 32768 -93
9999992 -9999990 -9999999 9999996 0 -9999996 9999999
```

출력 예시

```
3
1
2
4
7
```

[제출]