Intra-University Programming Contest 2025
Seniors
Editorial
Contest Link
Github Link

Rahul Kumar Ghosh and Sourav Mondal
Northern University of Business and Technology Khulna
Department of Computer Science and Engineering

Contents

1 A. Arko and His Innovative Running Track

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Rahul Kumar Ghosh, Sourav Mondal
Tag : Mathematics, Number Theory, Geometry

1.1 Editorial

To design the track, Arko selected three specific points on a flat field: A, B, and C. These points represent three key checkpoints that every runner must pass. Arko intends to build a circular track such that all three of these points lie on the circumference of the circle.

To begin his plan, he measured the straight-line distances between each pair of points: AB, BC, and CA. Let these lengths be a, b, and c respectively. The problem now is to determine the minimum area of the circular track (the area of the circumcircle of $\triangle ABC$).

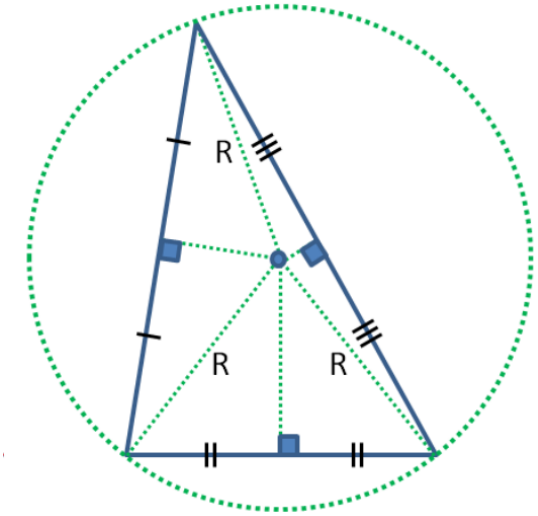

Figure 1: Triangle ABC inscribed in a circle (circumcircle).

1.2 Approach

We know that the area of a triangle can be calculated using Heron's formula. The perimeter is:

$$p = a + b + c$$

The semi-perimeter is:

$$s = \frac{p}{2} = \frac{a + b + c}{2}$$

Thus, the area of the triangle is:

$$A = \sqrt{s \times (s-a) \times (s-b) \times (s-c)}$$

$$A = \sqrt{\frac{a+b+c}{2} \times \left(\frac{a+b+c}{2} - a\right) \times \left(\frac{a+b+c}{2} - b\right) \times \left(\frac{a+b+c}{2} - c\right)}$$

$$A = \sqrt{\frac{a+b+c}{2} \times \frac{(b+c)-a}{2} \times \frac{(a+c)-b}{2} \times \frac{(a+b)-c}{2}}$$

$$A = \sqrt{\frac{(a+b+c)}{16} \times (b+c-a) \times (a+c-b) \times (a+b-c)} \tag{1}$$

For a circumcircle, the radius r is given by:

$$r = \frac{a \times b \times c}{4 \times A} \qquad \text{(circumscribed circle (circumcircle))}$$

$$r = \frac{a \times b \times c}{4 \times \sqrt{\frac{(a+b+c)}{16} \times (b+c-a) \times (a+c-b) \times (a+b-c)}}$$

Squaring both sides:

$$r^2 = \left(\frac{a \times b \times c}{4 \times \sqrt{\frac{(a+b+c)}{16} \times (b+c-a) \times (a+c-b) \times (a+b-c)}}\right)^2$$

$$r^2 = \left(\frac{a^2 \times b^2 \times c^2}{16 \times \frac{(a+b+c)}{16} \times (b+c-a) \times (a+c-b) \times (a+b-c)}\right)$$

$$r^2 = \left(\frac{a^2 \times b^2 \times c^2}{(a+b+c) \times (b+c-a) \times (a+c-b) \times (a+b-c)}\right)$$

The area of the circle is:

$$\pi \times r^2 = \frac{22}{7} \times \frac{a^2 \times b^2 \times c^2}{(a+b+c) \times (b+c-a) \times (a+c-b) \times (a+b-c)} \tag{2}$$

Let:

$$p = 22 \times a^2 \times b^2 \times c^2$$

$$q = 7 \times (a+b+c)(b+c-a)(a+c-b)(a+b-c)$$

We reduce the fraction by dividing numerator and denominator with $\gcd(p, q)$:

$$x = \frac{p}{\gcd(p,q)}, \quad y = \frac{q}{\gcd(p,q)}$$

So the final answer is printed as:

$$\frac{x}{y}$$

1.2.1 Complexity

1. **Time Complexity:** $O(1)$
2. **Space Complexity:** $O(1)$

1.2.2 Implementation in C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int test;
    cin >> test;
    while(test--){
       int a, b, c;
       cin >> a >> b >> c;
       int x = 22 * a * a * b * b * c * c;
       int y = 7 * (a + b + c) * (a + b - c) * (a + c - b) * (b +
           c - a);
       int d = gcd(x, y);
       cout << (x / d) << "/" << (y / d) << endl;
    }
    return 0;
}
```

1.2.3 Implementation in Python

```python
import math

def QKnot():
    a, b, c = map(int, input().split())
    x = 22 * a * a * b * b * c * c
    y = 7 * (a + b + c) * (a + b - c) * (a + c - b) * (b + c - a)
    d = math.gcd(x, y)
    print(f"{x // d}/{y // d}")

def main():
    try:
        test = int(input())
        for _ in range(test):
            QKnot()
    except:
        QKnot()

if __name__ == "__main__":
    main()
```

2 B. Prime Number Obsession Turns Into Madness

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Rahul Kumar Ghosh, Sourav Mondal
Tag: Mathematics, Number Theory

2.1 Editorial

We need to check whether a number n can be prime and whether it can be represented in the form

$$n = 2^{2^m} + 1.$$

Since n can be as large as $2^{63} - 1$, we must determine valid values of m.

2.2 Approach

The value of m can range only up to 5, because for $m \geq 6$ the value of n exceeds the limit $2^{63} - 1$. Let us compute the values step by step:

$$\begin{aligned} m &= 0 \quad 2^{2^0} + 1 = 3, \\ m &= 1 \quad 2^{2^1} + 1 = 5, \\ m &= 2 \quad 2^{2^2} + 1 = 17, \\ m &= 3 \quad 2^{2^3} + 1 = 257, \\ m &= 4 \quad 2^{2^4} + 1 = 65537, \\ m &= 5 \quad 2^{2^5} + 1 = 4294967297. \end{aligned} \tag{1}$$

Thus, only these six numbers are possible candidates. We then check if the number is prime. If it is prime, we print "YES"; otherwise, "NO".

$$m = 5,\ 2^{2^5} + 1 = 4294967297$$

This number is not a prime, so in this case we have to print "NO".

2.3 Implementation in C

```c
#include<stdio.h>
int main()
{
    int test;
    scanf("%d", &test);
    while(test--){
        int n;
        scanf("%d", &n);
        if (n == 3 || n == 5 || n == 17 || n == 257 || n ==
           65537){
            printf("Yes\n");
        }else{
            printf("No\n");
        }
    }
}
```

2.4 Implementation in C++

2.4.1 Implementation 1

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
auto QKnot = []()->void
{
    int n;
    cin >> n;
    if (n == 3 || n == 5 || n == 17 || n == 257 || n == 65537) {
        cout << "YES" << endl;
    } else{
        cout << "NO" << endl;
    }
};
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    bool bl = true;
    bl?[]()->void
    {
        int test;
        cin >> test;
        while(test--) QKnot();
    }():QKnot();
    return 0;
}
```

2.4.2 Implementation 2

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    vector<int> arr;
    for(int i = 0; i < 6; i++){
        int x = pow(2, i);
        arr.push_back(pow(2, x) + 1);
    }
    vector<int> prime;
    auto isprime = [](int n)->bool
    {
        if(n == 2) return true;
        else if (n == 1 || n % 2 == 0) return false;
        else{
            for(int i = 3; i * i <= n; i += 2){
                if(n % i == 0) return false;
            }
        }
        return true;
    };
    for(int i = 0; i < arr.size(); i++){
        if(isprime(arr[i])){
            prime.push_back(arr[i]);
        }
    }
    int test;
    cin >> test;
    while(test--){
       int n;
       cin >> n;
       bool tag = false;
       for(int i = 0; i < prime.size(); i++){
            if(n == prime[i]){
                tag = true;
            }
       }
       cout << (tag ? "yes" : "no") << endl;
    }
    return 0;
}
```

2.4.3 Implementation 3

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
const int mod = 1e9 + 7;
auto QKnot = []()->void
{
    int n;
    cin >> n;
    double x = log2(log2(n - 1));
    int p = round(x);
    if (p <= 4 && fabs(x - p) < 1e-9){
        cout << "YES" << endl;
    }else {
        cout << "NO" << endl;
    }
};
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    bool bl = true;
    bl?[]()->void
    {
        int test;
        cin >> test;
        while(test--) QKnot();
    }():QKnot();
    return 0;
}
```

2.5 Implementation in Python

```python
def main():
    test = int(input())
    for _ in range(test):
        n = int(input())
        if n == 3 or n == 5 or n == 17 or n == 257 or n == 65537:
            print("YES")
        else:
            print("NO")
if __name__ == "__main__":
    main()
```

3 MrBeast's Golden Feastables Giveaway

Author: Sourav Mondal
Legend/Story: Sourav Mondal
Tester: Sourav Mondal, Rahul Kumar Ghosh
Tag : Mathematics, Binary Search

3.1 Editorial

Let's look at one more problem — the rope problem.

We have n ropes, the length of the i-th rope is a_i. We want to cut k pieces of the same length out of them. Of all the cutting methods, you need to choose the one in which the length of the cut pieces is maximum.

To solve this problem, let's define the following function:

$$f(x) = \begin{cases} 1 & \text{if it is possible to cut at least } k \text{ pieces of length } x \\ 0 & \text{otherwise} \end{cases}$$

If we can cut pieces of length x, then we can also cut pieces of smaller size. This makes binary search applicable.

3.1.1 Binary Search on Real Numbers

Unlike integers, real numbers have no adjacent neighbors. Hence, we run binary search with a precision ε. The process stops when the difference between the search boundaries $[l, r]$ is less than ε.

To avoid infinite loops (due to double precision limits), we usually iterate a fixed number of times, e.g., 100 iterations, which is sufficient for most problems.

3.1.2 Implementation of $f(x)$

The function $f(x)$ iterates over all ropes, dividing their lengths by x (integer division). This gives the number of pieces of length x from each rope. Summing these values gives the total number of pieces. If the total is at least k, then $f(x) = 1$; otherwise, $f(x) = 0$.

3.2 Approach

We apply binary search on the answer space (from 0 to $\max(a_i)$) with precision ε.

3.2.1 Complexity

1. **Time Complexity:** $O(n \cdot \log \frac{\max(a_i)}{\varepsilon})$
2. **Space Complexity:** $O(1)$

3.2.2 Implementation in C++

```cpp
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif

template <typename T, typename U>
T realTrue(T lo, T hi, U f) {
    for (int i = 0; i < 100; ++i) {
        T mid = 0.5 * (hi + lo);
        f(mid) ? lo = mid : hi = mid;
    }
    return lo;
}

int32_t main() {
    ios::sync_with_stdio(0);
    cin.tie(nullptr);

    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    for (auto& z : a) cin >> z;

    auto ans = realTrue(0.0, 2E9, [&](auto len) {
        int c = 0;
        for (auto& z: a) c += z/len;
        return c >= k;
    });
    cout << fixed << setprecision(15) << ans << "\n";

    return 0;
}
```

4 D. It was a stormy day

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Rahul Kumar Ghosh, Sourav Mondal

4.1 Editorial:

When a lightning strike occurs, both light and sound are generated. Light travels almost instantaneously to the human eye ($v_{light} \approx 3 \times 10^8$ m/s), while sound travels much slower. At $0°C$, the speed of sound is 331.5 m/s, and for every $1°C$ increase in temperature, it increases by 0.6 m/s:

$$v_{sound} = 331.5 + 0.6 \times C$$

Given the time difference T (seconds) between seeing the lightning and hearing the thunder, and the air temperature C (Celsius), we are asked to compute the distance S (in kilometers) between the observer and the origin of the thunderbolt. The answer must be rounded to 9 decimal places.

From the newton's law of motion:

$$s = v_{light} \times t_{light} \tag{1}$$

$$s = v_{sound} \times t_{sound} \tag{2}$$

$$\Delta t = t_{sound} - t_{light} (t_{sound} > t_{light})$$

From the equation (1) and (2)

$$\Delta t = \frac{s}{v_{sound}} - \frac{s}{v_{light}}$$

$$\Delta t = s \times (\frac{1}{v_{sound}} - \frac{1}{v_{light}})$$

$$\Delta t = s \times \frac{(v_{light} - v_{sound})}{(v_{light} \times v_{sound})}$$

$$s = \frac{(v_{light} \times \Delta t \times v_{sound})}{(v_{light} - v_{sound})}$$

$$s = \frac{((3 \times 10^8) \times \Delta t \times (331.5 + 0.6 \times C))}{((3 \times 10^8) - (331.5 + 0.6 \times C))}$$

4.2 Implementation in C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int test;
    cin >> test;
    while(test--){
       double del_t, c;
       cin >> del_t >> c;
       double v_sound = (331.5 + (double)(0.6 * c));
       double v_light = 3 * 1e8;
       double s = (v_light * del_t * v_sound) / (v_light -
          v_sound);
       cout << setprecision(10) << s / 1000 << endl;
    }
    return 0;
}
```

5 E. The most beautiful equation in mathematics

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Rahul Kumar Ghosh, Sourav Mondal
Tag : Mathematics, ad hoc, Number theory

5.1 Editorial

We want to evaluate:

$$e^{ni\pi} + 1 \tag{1}$$

From Euler's formula:

$$e^{ni\pi} = \cos(n\pi) + i\sin(n\pi)$$

Since $\sin(n\pi) = 0$, we have:

$$e^{ni\pi} = (-1)^n$$

Therefore,

$$e^{ni\pi} + 1 = (-1)^n + 1$$

5.2 Approach

Let us consider cases for n:

- If n is even, then $(-1)^n = 1$:

$$e^{ni\pi} + 1 = 1 + 1 = 2$$

- If n is odd, then $(-1)^n = -1$:

$$e^{ni\pi} + 1 = -1 + 1 = 0$$

5.2.1 Complexity

1. **Time Complexity:** $O(1)$
2. **Space Complexity:** $O(1)$

5.2.2 Implementation in C

```c
#include<stdio.h>
int main()
{
    int test;
    scanf("%d", &test);
    while(test--){
        int n;
        scanf("%d", &n);
        if(n % 2 == 0){
            printf("%d\n", 2);
        }else{
            printf("%d\n", 0);
        }
    }
    return 0;
}
```

5.2.3 Implementation in C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
const int mod = 1e9 + 7;
auto solve = []()->void
{
    int n;
    cin >> n;
    cout << ((n & 1) ? 0 : 2) << endl;
};
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int test;
    cin >> test;
    while(test--){
        solve();
    }
    return 0;
}
```

5.2.4 Implementation in Python

```python
def main():
    test = int(input())
    for i in range(test):
        n = int(input())
        print(2 if n % 2 == 0 else 0)
if __name__ == "__main__":
    main()
```

6 F. A Man, A Myth, A Legend of NUBTK

Author: Sourav Mondal
Legend/Story: Sourav Mondal
Tester: Rahul Kumar Ghosh and Sourav Mondal
Tag: Tree, DFS simillar

6.1 Editorial

The problem can be solved using a basic dynamic programming (DP) approach on trees in linear time. We define:

$$\text{subordinates}[u] = \sum_{v \in \text{children}[u]} (1 + \text{subordinates}[v])$$

where v is a child of u. Thus, for each employee u, the number of subordinates is the sum of all subordinates of its direct children plus the children themselves.

6.2 Approach

We can use a DFS (Depth First Search) traversal starting from the root (employee 1). During the DFS, we compute the size of each subtree and store the result for each employee.

6.2.1 Complexity

1. **Time Complexity:** $O(n)$, since each edge is visited once in DFS.
2. **Space Complexity:** $O(n)$, for storing the adjacency list and the recursion stack in the worst case.

6.2.2 Implementation in C++

```cpp
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif

int32_t main() {
        ios::sync_with_stdio(0);
        cin.tie(nullptr);

        int n;  cin >> n;

    vector<vector<int>> adj(n);
        for (int u = 1; u < n; ++u) {
                int v;
                cin >> v;
                v--;
                adj[u].push_back(v);
                adj[v].push_back(u);
        }

        vector<int> ans(n);
        auto dfs = [&](this auto&& self, int u, int p)-> void {
                for (auto& v: adj[u]) {
                        if (v != p) {
                                self(v, u);
                                ans[u] += ans[v]+1;
                        }
                }
        };
    dfs(0, -1);

        for (int i = 0; i < n; ++i) {
                cout << ans[i] << " \n"[i == n-1];
        }

        return 0;
}
```

7 G. Johann Carl Friedrich Gauss

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Rahul Kumar Ghosh, Sourav Mondal
Tag : Number theory, Mathematics

7.1 Editorial

In this problem, if we sum the numbers one by one, it will result in a time limit exceeded. Instead, we can use a formula instead of the brute force solution.

$$1, 2, 3, 4, \ldots, n-2, n-1, n$$

We can pair the numbers as follows:

$$1+n,\ 2+(n-1),\ 3+(n-2), \ldots$$

Each pair sums to $(n+1)$, and there are $\frac{n}{2}$ such pairs. So, the total sum will be:

$$\frac{n \times (n+1)}{2}$$

Since the value of n can be as large as 10^9, the result may exceed the range of a 32-bit integer. Therefore, we should use a 64-bit integer (`long long`) to avoid overflow.

7.2 Approach

7.2.1 Complexity

1. **Time Complexity:** $O(1)$
2. **Space Complexity:** $O(1)$

7.2.2 Implementation in C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int test;
    cin >> test;
    while(test--){
        int n;
        cin >> n;
        cout << n * (n + 1) / 2 << endl;
    }
    return 0;
}
```

7.2.3 Implementation in Python

```python
def main():
    test = int(input())
    for i in range(test):
        n = int(input())
        print((n * (n + 1)) // 2)

if __name__ == "__main__":
    main()
```

8 H. Sourav vs Arko: Battle at Khulna Railway Station

Author: Rahul Kumar Ghosh
Legend/Story: Rahul Kumar Ghosh
Tester: Sourav Mondal, Rahul Kumar Ghosh
Tag : Game Theory

8.1 Editorial

8.1.1 Problem restatement

Two players (Sourav goes first, then Arko) play a game with a pile of n stones. They alternately remove between 1 and k stones (inclusive) on their turn. The player who cannot make a move (i.e. when the pile is empty on their turn) loses. Given n and k, determine the winner when both play optimally.

8.1.2 Key observation

This is a standard impartial take-away game. Let us call a position (pile size) *losing* if the player to move from that position will lose with perfect play, and *winning* otherwise.

Consider the values modulo $k + 1$:

$$n \bmod (k + 1).$$

The crucial observation is:

- If $n \equiv 0 \pmod{k+1}$, then the position with n stones is **losing** for the player to move.
- If $n \not\equiv 0 \pmod{k+1}$, then the position is **winning** for the player to move.

8.1.3 Proof (simple and standard)

1. Base / induction view. All positions $0, k + 1, 2(k + 1), \ldots$ are losing:

- If $n = 0$ the player to move loses (no legal moves).
- Suppose $n = m(k + 1)$ for some $m \geq 1$. Any legal move removes between 1 and k stones, leaving $n' = m(k + 1) - r$ with $1 \leq r \leq k$. Then

  $$n' \equiv -r \not\equiv 0 \pmod{k+1},$$

  so every move from n moves to a position that is not a multiple of $k+1$ (a winning position for the next player). Hence n is losing.

2. Winning positions. If $n \not\equiv 0 \pmod{k+1}$, write $n = q(k + 1) + r$ with $1 \leq r \leq k$. The player to move can remove exactly r stones (allowed since $1 \leq r \leq k$), leaving $n' = q(k + 1)$, a losing position for the opponent. Thus n is winning.

This completes the proof: all and only multiples of $k + 1$ are losing positions.

8.1.4 Optimal strategy

- If $n \bmod (k+1) = 0$, the first player (Sourav) loses with perfect play from the opponent.
- Otherwise, Sourav should remove $r = n \bmod (k+1)$ stones on his first move. Thereafter he can *mirror* Arko's moves: whenever Arko removes x stones, Sourav removes $k+1-x$ stones. This keeps the pile size after Sourav's turns always a multiple of $k+1$, ensuring eventual victory.

8.1.5 Algorithm and complexity

For each test case (n, k):

1. Compute $r \leftarrow n \bmod (k+1)$.
2. If $r = 0$, print `Arko` (Sourav loses). Otherwise print `Sourav`.

Time complexity per test case: $O(1)$. With t test cases the total complexity is $O(t)$. The operations fit easily within standard 64-bit integer limits given constraints $1 \le n, k \le 10^9$.

8.1.6 Examples

- $n = 7,\ k = 3$: here $k+1 = 4$, $7 \bmod 4 = 3 \neq 0 \Rightarrow$ Sourav wins.
- $n = 8,\ k = 3$: $8 \bmod 4 = 0 \Rightarrow$ Arko wins.

8.1.7 Implementation note (one-liner logic)

```
if (n % (k+1) == 0) print("Arko"); else print("Sourav");
```

8.1.8 Implementation in C++

```
#include<stdio.h>
int main()
{
    int test;
    scanf("%d", &test);
    while(test--){
        int n, k;
        scanf("%d %d", &n, &k);
        printf((n % (k + 1))?"Sourav\n":"Arko\n");
    }
    return 0;
}
```