AlgoWin 1.0 solutions

Editorial for the AlgoWin 1.0 conducted on 5th November 2022 has been provided below in C++. Kindly go through the approach and submit your own code. Contest link has been attached here
https://www.hackerrank.com/contests/algowin-1-0/challenges

1.Odd Factor!

Approach: The numbers without odd divisors are all the powers of 2.Using the AND bitwise operator we could solve the problem.
In binary, all powers of two have only a single bit as 1 in its binary representation,all others will be 0. Also the number before a power of two will have all its bits as 1.

```
#include <iostream>
using namespace std;
int main() {
    long int t;
    cin>>t;
    while(t--)
    {
        long int n;
        cin>>n;
        if((n&(n-1))==0)
            cout<<"NO"<<"\n";
        else
            cout<<"YES\n";
    }
    return 0;
}
```

2.Tesla Coils

Approach: Sort electric coils in non-decreasing order. Then we need to find the maximal distance between two neighbour

coils, let it be *d*. Also we need to consider length bounds and count distances from outside coil to length bounds, it will be (*a*[0] - 0) and (*l* - *a*[*n* - 1]). The answer will be *max*(*d* , *max*(*a*[0] - 0, *l* - *a*[*n* - 1]).

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    int n,l;
    cin>>n>>l;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    sort(arr,arr+n);
    int d = 2 * max(arr[0],l-arr[n-1]);
    for(int i=1;i<n;i++){
        d = max(d,(arr[i]-arr[i-1]));
    }
    cout<<fixed<<(double) d/2;//fixed will write the  floating
                              //values in fixed point notation
}
```

3.Grade Point

Approach: The problem is to find the nearest smallest element present at the left part of the present element.We start pushing element's index to stack and print the element position if and only if stack is empty else we go on popping the element from if stack if the next element entering the array is greater than its previous.

```
#include<bits/stdc++.h>
```

```
using namespace std;

int main() {
    stack<int>st; //Stores the index of the input elements
    int n;
    cin>>n;
    vector<int>A(n);
    for(int i=0;i<n;i++){
        cin>>A[i];
        while(st.empty() == false && A[st.top()] >= A[i])
            st.pop();
        cout<<((st.empty() == false)?st.top()+1:0)<<' ';
        st.push(i);
    }
    return 0;
}
```

4. 1 or 2 or 3

Approach: Recursive approach will lead to combinatorial explosion hence we directly go to DP approach.We are creating an array of size n+1 and store the indices 0,1,2 with 1,1,2 as base case. Since the person standing at ith position can take a jump from i-1 or i-2 or i-3 we loop from index 3 to n and store a[i] as (a[i-1]+a[i-2]+a[i-3])%MOD. The value present at the last position will give us the number of ways to reach the classroom.

```
#include <bits/stdc++.h>
using namespace std;
int MOD = (int)1e9 + 7;
int find(int n)
{
    int a[n+1];
    a[0]=a[1]=1;
    a[2]=2;
```

```
    for(int i=3;i<n+1;i++)
        a[i]=(a[i-1]+a[i-2]+a[i-3])%MOD;
    return a[n];
}
int main() {
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        long int res;
        cin>>n;
        res=find(n);
        cout<<res%MOD<<"\n";
    }

    return 0;
}
```

5.ICPC Championship

Approach: The task is to find the maximum luck points which is nothing but the single source longest path from starting city to the last city.We convert the edge weights into its negative value and then find shortest path using Bellmann algorithm at the end the result will be negative of the shortest path.

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

const int inf = 1e9;
const int ninf = -1e9;
struct custom {
```

```
    int from;
    int to;
    int weight;
};

int main() {
    ios_base::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    vector<custom> edges;
    for(int i = 0; i < m; i++) {
        custom input;
        cin >> input.from >> input.to >> input.weight;
        input.weight *= -1;
        edges.push_back(input);
    }
    vector<int> dist(n+1, inf);
    dist[1] = 0;
    // bellman ford algo
    for(int i = 0; i < n; i++) {
        for(auto e : edges) {
            int u = e.from, v = e.to, w = e.weight;
            if(dist[u] == inf)
                continue;
            dist[v] = min(dist[v], dist[u]+w);
            dist[v] = max(dist[v], ninf);
        }
    }

    //Identify the negative cycle and make nodes affected by
    //negative cycle as -infinity
    for(int i = 0; i < n; i++) {
        for(auto e : edges) {
            int u = e.from, v = e.to, w = e.weight;
            if(dist[u] == inf)
                continue;
```

```
            dist[v] = max(dist[v], ninf);
            if(dist[v] > dist[u]+w)
                dist[v] = ninf;
        }
    }
    if(dist[n] == ninf or dist[n] == inf) {
        cout << -1;
    } else {
        cout << -1*dist[n];
    }
    return 0;
}
```