# ABC 166 解説

gazelle, kort0n, kyopro_friends, sheyasutaka, ynymxiaolongbao

2020 年 5 月 3 日

*For International Readers: English editorial starts on page 8.*

## A: A?C

与えられた文字列が ABC なら ARC を、ARC なら ABC を出力します。

```
#include <iostream>
using namespace std;
int main(){
    string s;
    cin >> s;
    if(s == "ABC"){
        cout << "ARC" << endl;
    }else{
        cout << "ABC" << endl;
    }
}
```

## B: Trick or Treat

各すぬけ君について, お菓子を持っているか否かを調べます.
以下は C++ での実装例です.

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

int N, K;
vector<int> d;
vector<vector<int>> A;

void input() {
    cin >> N >> K;
    d.resize(K);
    A.resize(K, vector<int>());
    for(int i = 0; i < K; i++) {
        cin >> d[i];
        A[i].resize(d[i]);
        for(int j = 0; j < d[i]; j++) {
            cin >> A[i][j];
        }
    }
}

void solve() {
    int ans = 0;
    for(int i = 1; i <= N; i++) {
        bool have = false;
        for(auto v : A) {
            for(auto p : v) {
                if(p == i) have = true;
            }
        }
        if(!have) ans++;
    }
    cout << ans << endl;
}

int main() {
    input();
    solve();
```

```
38      return 0;
39  }
```

## C. Peaks

グラフの問題であるので隣接リストでグラフを表現するといった実装もありますが、今回はそれを用いない方法を紹介します。「展望台 $i$ から一本の道を使って辿り着けるどんな展望台よりも展望台 $i$ の方が標高が高い」ことは、「展望台 $i$ から一本の道を使って辿り着ける展望台の標高の最大値よりも展望台 $i$ の標高が高い」ことと等しいです。このことを利用すると、展望台 $i$ から一本の道を使って辿り着ける展望台の標高の最大値を求めることで答えを求めることができます。

以下が、c++ のサンプルコードです。

```
#include<iostream>
#include<algorithm>
using namespace std;
#define N 100010
int main(){
    int n,m,h[N],ma[N];
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>h[i];
        ma[i]=0;
    }
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        ma[a]=max(ma[a],h[b]);
        ma[b]=max(ma[b],h[a]);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        ans+=h[i]>ma[i];
    }
    cout<<ans<<endl;
}
```

## D. I hate Factorization

添字が負でもよい数列 $a_n = n^5 - (n-1)^5$ すなわち $...781, 211, 31, 1, 1, 31, 211, 781...$ を考えます。この数列は、$a_0$ から左に、$a_1$ から右に向けて値が大きくなっていくことがわかります。数列の値がはじめて $10^9$ を超えるのは、$a_{120} = 120^5 - 119^5 = 1019663401$ のときと、$a_{-119} = (-119)^5 - (-120)^5 = 1019663401$ のときになります。よって、$A - B = 1$ を仮定すると、$-118 \leq A \leq 119$ のケースだけ考えれば良いことがわかります。

そして、$n^5$ が $n$ に対して単調増加であることに注意すると、$A - B > 1$ の場合でも $-118 \leq A \leq 119$ のケースだけ見ていれば十分だとわかります。同じ $A$ の値に対して、$A - B$ の値が大きくなって $B$ の値が小さくなれば、$B^5$ の値も小さくなって、$A^5 - B^5$ の値は大きくなるためです。$A - B = 1$ のときと同様に数列 $a_n = n^5 - (n-2)^5$ や $a_n = n^5 - (n-3)^5$ などについて考えることで更に $A$ の範囲を狭くすることができますが、今回は計算時間に余裕があるためその必要はありません。

$A$ と $B$ が整数かつ $1 \leq A^5 - B^5$ より $A - B \geq 1$ なので、これですべての場合が網羅されています。$B$ に対しても同じ考察をすると $-118 \leq A \leq 119$ かつ $-119 \leq B \leq 118$ なる整数の組 $(A, B)$ に候補を絞ることができ、これらの候補が実際に式を満たすかどうか判定していくことで解くことができます。

## E: This Message Will Self-Destruct in Five Seconds

参加者 $i, j$ $(i < j)$ について，条件は以下のように書けます:

$$j - i = A_i + A_j$$

これを式変形すると，以下のようになります:

$$i + A_i = j - A_j$$

そこで，$L_i = i + A_i, R_i = i - A_i$ と定義すると，条件は $L_i = R_j$ と言い換えられます．したがって，とり得る値 $X$ を全て試し，

$$\begin{cases} L_i = X \\ R_j = X \end{cases}$$

を満たす $(i, j)$ の組を求めればいいですが，上と下の条件は独立なので，たとえば連想配列を使えば簡単に求まります．

計算量は $\mathcal{O}(N \log N)$ 時間 ($X$ を見る範囲を少し工夫すれば $\Theta(N)$ 時間) です．

## F:Three Variables Game

- $A+B+C=0$ のとき
  $N$ は 1 以上なので、答えは No です。
- $A+B+C=1$ のとき
  いつどの二つの変数を見ても少なくとも一方は零なので、それぞれのターンでの行動が一意に定まります。
- $A+B+C>=2$ のとき
  一回目のターンで足し引きする変数が両方零であれば答えは No、そうでなければ答えは Yes です。後者の場合、以下のような戦略をそれぞれのターンで取ることで足し引きする変数のいずれかが正である状態を保つことができます。
  - 足し引きする変数の片方が零でもう一方が正のとき、零である方に 1 を足し、もう一方から 1 を引く。
  - $A+B+C=2$ であり、足し引きする変数が両方 1 であり、最後のターンでなく、次のターンの文字列と現在のターンの文字列が異なるとき、次のターンで足し引きする予定がある方の変数に 1 を足し、もう一方から 1 を引く。
  - それ以外のとき、適当に選択する

## A: A?C

If the given string is ABC, then print ARC; if it is ARC, then print ABC.

```
#include <iostream>
using namespace std;
int main(){
    string s;
    cin >> s;
    if(s == "ABC"){
        cout << "ARC" << endl;
    }else{
        cout << "ABC" << endl;
    }
}
```

## B: Trick or Treat

For each Snuke, judge if he has snacks or not.

The following is a sample code in C++.

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

int N, K;
vector<int> d;
vector<vector<int>> A;

void input() {
    cin >> N >> K;
    d.resize(K);
    A.resize(K, vector<int>());
    for(int i = 0; i < K; i++) {
        cin >> d[i];
        A[i].resize(d[i]);
        for(int j = 0; j < d[i]; j++) {
            cin >> A[i][j];
        }
    }
}

void solve() {
    int ans = 0;
    for(int i = 1; i <= N; i++) {
        bool have = false;
        for(auto v : A) {
            for(auto p : v) {
                if(p == i) have = true;
            }
        }
        if(!have) ans++;
    }
    cout << ans << endl;
}

int main() {
    input();
    solve();
```

```
38      return 0;
39  }
```

## C. Peaks

Since this is a problem of a graph, an implementation of representing the graph with adjacency list, but this time we will introduce a way without them. That "the elevation of Obs. $i$ is higher than those of all observatories that can be reached from Obs. $i$ using just one road" is equivalent to that "the elevation of Obs. $i$ is higher than the maximum elevation of observatories that can be reached from Obs. $i$ using just one road." Using this fact, you can find the answer by calculating the maximum elevations of observatories that can be reached from Obs. $i$ using just one road.

The following is a sample code in C++.

```
#include<iostream>
#include<algorithm>
using namespace std;
#define N 100010
int main(){
    int n,m,h[N],ma[N];
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>h[i];
        ma[i]=0;
    }
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        ma[a]=max(ma[a],h[b]);
        ma[b]=max(ma[b],h[a]);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        ans+=h[i]>ma[i];
    }
    cout<<ans<<endl;
}
```

## D. I hate Factorization

Consider a sequence that allows negative index $a_n = n^5 - (n-1)^5$, i.e. $...781, 211, 31, 1, 1, 31, 211, 781....$ This sequence is increasing toward left from $a_0$, and toward right from $a_1$. The elements exceed $10^9$ for the first time when $a_{120} = 120^5 - 119^5 = 1019663401$ and when $a_{-119} = (-119)^5 - (-120)^5 = 1019663401$. Therefore, assuming that $A - B = 1$, it appears that it is sufficient to consider only the range of $-118 \leq A \leq 119$.

Moreover, considering the increasing monotonicity of $n^5$ to $n$, it appears that it is sufficient to consider only the range of $-118 \leq A \leq 119$ even when $A - B > 1$. When $A$ is fixed, if $A - B$ increases and $B$ decreases, then $B^5$ decreases too and $A^5 - B^5$ increases. As with when $A - B = 1$, you can restrict the range of $A$ even narrower by considering sequences $a_n = n^5 - (n-2)^5$ or $a_n = n^5 - (n-3)^5$, but this time there is a plenty of execution time, so there is no need to do so.

Since $A$ and $B$ are integers and $1 \leq A^5 - B^5$, it holds that $A - B \geq 1$, so all the cases are covered by them. By applying the same observations for $B$, you can narrow down the candidates of ranges to the pairs of integers $(A, B)$ such that $-118 \leq A \leq 119$ and $-119 \leq B \leq 118$, And check if those candidates actually satisfies the equation.

## E: This Message Will Self-Destruct in Five Seconds

For participants $i, j$ $(i < j)$, the condition can be written as follows:

$$j - i = A_i + A_j$$

By transforming this equation, the following can be obtained:

$$i + A_i = j - A_j$$

So, let $L_i = i + A_i, R_i = i - A_i$, then the condition can be written as $L_i = R_j$. Therefore, it is sufficient to search exhaustively through all the possible $X$ and count the number of $(i, j)$ such that

$$\begin{cases} L_i = X \\ R_j = X \end{cases}$$

. Since the former and the latter are independent, it can be obtained easily by using associative arrays for example.

The total time complexity is $\mathcal{O}(N \log N)$ (or $\Theta(N)$ if you properly narrowed the range of $X$).

## F:Three Variables Game

- When $A + B + C = 0$
  Since $N$ is at least one, the answer is No.
- When $A + B + C = 1$
  Whenever you see whatever two variables, at least one of them is zero, so the action for each step is uniquely determined.
- When $A + B + C >= 2$
  If the two variables which you modify in the first turn are both zero, the answer is No; otherwise the answer is Yes. In the latter case, you can always keep one of the two variables modified for each turn to be positive by the following strategy:
  - If one of the variables that will be modified is zero and the other is positive, then add one to the variable whose value is zero, and subtract one from the other.
  - If $A + B + C = 2$, the variables that will modified is both one, it is not the last turn, and the string for this turn and the next turn differs, then add one to the variables that will be modified for the next time, and subtract one from the other.
  - Otherwise, choose arbitrarily