ABC 132 解説

DEGwer, gazelle, potetisensei, tozangezan, yuma000

2019 年 6 月 29 日

For International Readers: English editorial starts on page 7.

A: Fifty-Fifty

以下のように、3 通りに場合分けして答えを求めればよいです。

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main()
{
	char s[10];
	scanf("%s", s);
	if(s[0] == s[1] && s[1] != s[2] && s[2] == s[3])printf("Yes\n");
	else if(s[0] == s[2] && s[2] != s[1] && s[1] == s[3])printf("Yes\n");
	else if(s[0] == s[3] && s[3] != s[1] && s[1] == s[2])printf("Yes\n");
	else printf("No\n");
}
```

また、以下のように文字列を並び替えてから判定を行っても良いです。

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main()
{
	char s[10];
	scanf("%s", s);
	sort(s, s + 4);
	printf((s[0] == s[1] && s[1] != s[2] && s[2] == s[3]) ? "Yes\n" : "No\n");
}
```

B: Ordinary Number

p_i が条件を満たすことと、$p_{i-1} < p_i < p_{i+1}$ または $p_{i-1} > p_i > p_{i+1}$ のいずれかの条件が成り立つことは同値です。よって、for ループを回してこの条件を満たす要素を数えていけばよいです。

以下は C++ での実装例です。

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> p(n);
    for(int i = 0; i < n; i++) cin >> p[i];
    int ans = 0;
    for(int i = 1; i < n - 1; i++) {
        if((p[i - 1] < p[i]) && (p[i] < p[i + 1])) ans++;
        else if((p[i - 1] > p[i]) && (p[i] > p[i + 1])) ans++;
    }
    cout << ans << endl;
    return 0;
}
```

C. Dividing Problems(writer : yuma000)

結論から言うと、

- $N/2$ 番目に難しい問題が「ARC 用の問題」、$N/2-1$ 番目に難しい問題が「ABC 用の問題」となること
- 「ARC 用の問題」の数と「ABC 用の問題」の数が同じになること

は、同値であると言えます。よって、解法は以下のようになります。

1. 問題を難易度順に昇順でソートする。
2. $N/2$ 番目の要素から、$N/2-1$ 番目の要素を引いたものを出力する。

多くの言語にはソート用のライブラリが用意されているので、それを利用するのが良いでしょう。(C++ なら std::sort)

以下が、C++ のサンプルコードです。

```cpp
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

int main(){
    int N;cin>>N;
    vector<int>v(N);
    for(int i=0;i<N;++i){
        cin>>v[i];
    }
    sort(v.begin(),v.end());

    int answer=v[v.size()/2]-v[v.size()/2-1];

    cout<<answer<<endl;

    return 0;
}
```

D. Blue and Red Balls

K 個の青いボールを回収するのに高橋君がちょうど i 回操作をする必要があるというのは、K 個の青ボールが赤いボールによって i 箇所に区切られているということを意味します。

そこで、次のように組み合わせを考えていくことにしましょう。

1. まず、赤いボールを $N-K$ 個一列に並べます。
2. この中で赤いボールと赤いボールの間、左端、右端の中から i 箇所を選んでそこに青いボールを K 個置くことを考えます。これらの選び方は、 ${}_{N-K+1}C_i$ 通りあります。
3. それぞれの選び方について、青いボールをそれぞれの隙間に何個割り当てていくかを考えます。それぞれに 1 個以上割り当てる必要があるので、この決め方は ${}_{K-1}C_{i-1}$ 通りあります。(*)

よって、それぞれの i について、答えは ${}_{N-K+1}C_i \times {}_{K-1}C_{i-1}$ となります。コンビネーションの計算は、${}_nC_k = \frac{n!}{k!(n-k)!}$ であることを利用し、階乗、逆元、階乗の逆元を前計算しても良いですが、今回は $N \leq 2000$ なので、パスカルの三角形を上から求めていく要領で $C[i][j] = {}_iC_j$ を DP で ($C[i][j] = C[i-1][j] + C[i-1][j-1]$) 求めてもよいです。

(*) K 個のボールを並べて、 $K-1$ 個のボールとボールの間から $i-1$ 箇所選び、そこで切り分けると i 箇所に分かれる。各部分は最小 1 個のボールを含んでおり、ボールの総数は K 個になる。

E: Hopscotch Addict

言い換えると、この問題で求めたい値は「有向グラフ $G = (V, E)$ 上の、S から T への（単純パスとは限らない）路であって、路長が 3 の倍数であるようなものの内、最短の長さ（を 3 で割ったもの)」です。

これを計算するためには、以下のようにして構成したグラフ G' 上での最短経路問題を解けば良いです:

1. G' は頂点集合として $V' = \{v_t \mid v \in V, t = 0, 1, 2\}$ を持つ。すなわち、元のグラフ G の状態数を “3 倍化” する。
2. グラフ G が u から v への辺を持つ時、G' 上に 3 辺 $u_0 \to v_1, u_1 \to v_2, u_2 \to v_0$ を張る。

こうして構成された G' 上での S_0 から T_0 への最短経路（すなわち単純パスで良い）の長さは上記の求めたい値に一致します。したがって、結局これは重みなし有向グラフ上の最短経路問題に帰着され、G' の頂点数や辺数は G の高々定数倍であるため、BFS や DFS によって $O(N + M)$ で解くことが可能です。

F: Small Products

まず、$\mathrm{DP}[i][x]$: 最後の整数が x であるような i 個の整数を並べて条件を満たすようにする場合の数 とした $O(NK)$ 状態の DP を考えます。このままでは計算量が大きいので、この DP を高速化することを考えます。

正整数 x, y が $\lfloor \frac{N}{x} \rfloor = \lfloor \frac{N}{y} \rfloor = t$ を満たすとします。このとき、$\mathrm{DP}[i+1][x]$ と $\mathrm{DP}[i+1][y]$ は同じ更新式 $\sum_{z=1}^{t} \mathrm{DP}[i][z]$ で求まるため、値は等しくなります。よって、このような x, y に対する DP の状態は同一視してよいことがわかります。

上記の規則で同一視できる状態をすべて同一視すると、DP の状態数は $O(\sqrt{N}K)$ になります。これは、$x \leq \sqrt{N}$ のときは x の値が、$x > \sqrt{N}$ のときは $\lfloor \frac{N}{x} \rfloor$ の値が $O(\sqrt{N})$ 個しかないことからわかります。適切に累積和を用いれば、この DP は $O(\sqrt{N}K)$ 時間で動くように実装できるため、この問題を解くことができました。

ABC 132

DEGwer, gazelle, potetisensei, tozangezan, yuma000

06/29/2019

A: Fifty-Fifty

You can solve this problem by splitting into those three cases.

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main()
{
        char s[10];
        scanf("%s", s);
        if(s[0] == s[1] && s[1] != s[2] && s[2] == s[3])printf("Yes\n");
        else if(s[0] == s[2] && s[2] != s[1] && s[1] == s[3])printf("Yes\n");
        else if(s[0] == s[3] && s[3] != s[1] && s[1] == s[2])printf("Yes\n");
        else printf("No\n");
}
```

You can also do by sorting the characters.

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main()
{
        char s[10];
        scanf("%s", s);
        sort(s, s + 4);
        printf((s[0] == s[1] && s[1] != s[2] && s[2] == s[3]) ? "Yes\n" : "No\n");
}
```

B: Ordinary Number

p_i satisfies the condition if and only if $p_{i-1} < p_i < p_{i+1}$ or $p_{i-1} > p_i > p_{i+1}$. So you can solve the problem by counting the element that satisfies the condition with for-loop.

The following is an implementation example in C++.

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> p(n);
    for(int i = 0; i < n; i++) cin >> p[i];
    int ans = 0;
    for(int i = 1; i < n - 1; i++) {
        if((p[i - 1] < p[i]) && (p[i] < p[i + 1])) ans++;
        else if((p[i - 1] > p[i]) && (p[i] > p[i + 1])) ans++;
    }
    cout << ans << endl;
        return 0;
}
```

C. Dividing Problems(writer : yuma000)

To come to the point, the following two conditions are equivalent:

- The $N/2$-th hardest problem is *for ARCs* and the $N/2-1$-th problem is *for ABCs*
- The number of problems for ARCs and the number of problems for ABCs are same

Therefore, you can solve it by following steps:

1. Sort the problems in a increasing order of difficulty.
2. Output the $N/2$-th element subtracted by the $N/2-1$-the element.

You may use sorting library, which is available in many languages. (e.g. std::sort in C++)

The following is an implementation example in C++.

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

int main(){
    int N;cin>>N;
    vector<int>v(N);
    for(int i=0;i<N;++i){
        cin>>v[i];
    }
    sort(v.begin(),v.end());

    int answer=v[v.size()/2]-v[v.size()/2-1];

    cout<<answer<<endl;

    return 0;
}
```

D. Blue and Red Balls

When Takahashi needs exactly i moves to collect all K blue balls, K blue balls are separated by red balls into i segments.

Then let's think about the combination in the following steps:

1. First, arrange the $N-K$ red balls into a line.
2. Choose i spots out of the places between the red balls or the left or right endpoint, and place the K blue balls to them; there are ${}_{N-K+1}C_i$ ways to do so.
3. For each ways of choosing, decide how many balls to distribute to each spots. You have to put at least 1 ball into each spot, so there are ${}_{K-1}C_{i-1}$ ways to do so. (*)

Therefore, for each i, the answer will be ${}_{N-K+1}C_i \times {}_{K-1}C_{i-1}$. To calculate binomial coefficients, you can make use of the formula ${}_nC_k = \frac{n!}{k!(n-k)!}$ and calculate each factorials, inverse element, and factorials of inverse element; or alternatively, since $N \leq 2000$, you can calculate each $C[i][j] =_i C_j$ with DP ($C[i][j] = C[i-1][j] + C[i-1][j-1]$), in manner of Pascal's triangle.

(*) When you arrange K balls into a line and put bars into $i-1$ spots out of $K-1$ spots between the balls, the balls will be separated into i parts by them. Each segment contains at least 1 ball and the total number of balls is K.

E: Hopscotch Addict

To put the problem statement into other words, you have to find "the shotrtest length of path (divided by 3) among all the (not necessarily simple) path from S to T whose length is a multiple of 3."

This can be calculated by solving a shortest path problem on a graph G' constructed by the following procedure:

1. The vertex set of G' is $V' = \{v_t \mid v \in V, t = 0, 1, 2\}$. Frankly, this graph "triples" the number of states in original graph G.

2. Iff there exist a edge from u to v in G, there exist three edges $u_0 \to v_1, u_1 \to v_2, u_2 \to v_0$.

The length of shortest path from S_0 to T_0 on G' (which is always a simple path) is equal to the length of the path mentioned above.

Therefore, the problem is reduced into a shortest path problem on the unweighted graph, and the number of vertices and edges of G' is at most several times as many as that of original graph, so you can solve it by BFS or DFS in $O(N + M)$.

F: Small Products

Define Let $\text{DP}[i][x]$ as the number of ways to satisfy the constraints of the problem where are array is of size i and the last element is x. There are $O(K * N)$ states. Since there are too many states, we need to reduce the number of states by removing states with the same dp values.

Consider two positive integers x and y that satisfy $\lfloor \frac{N}{x} \rfloor = \lfloor \frac{N}{y} \rfloor = t$. Consider the formula for the values of $\text{DP}[i][x]$ and $\text{DP}[i][y]$. Both the formulas are $\sum_{z=1}^{t} \text{DP}[i][z]$ so the two values are equal.

We can now change our DP state to $\text{DP}[i][t]$ and since there are $\sqrt{N}$ values of t, the number of states has reduced to $O(\sqrt{N}K)$. Note that while there are only $\sqrt{N}$ values of t, the values are not consecutive. Using prefix sums, we can calculate each state in $O(1)$. This makes the runtime $O(\sqrt{N}K)$.