UM PROBLEMA INTERESSANTE SOBRE MEDIANAS

Ivan Henrique Costa Petrin, UNICAMP

Resumo

Comento sobre o problema Crystal Peak de um contest longo (10 dias) do grupo GoForGold no codeforces. Esse problema tem um resultado muito interessante sobre medianas e, segundo os autores, era o segundo problema mais difícil da prova. A solução esperada é baseada em um uso incomum de busca binária.

1. Contexto do contest

Na Índia, existe uma iniciativa chamada GoForGold, que pretende apoiar times indianos a conquistarem uma medalha de ouro no ICPC. A primeira vez em que ouvi falar dessa iniciativa foi no portal do codechef, há vários anos. Apesar das empresas que encabeçam a iniciativa terem mudado, o nome continuou o mesmo e expandiram a ideia para o time indiano na IOI.

Essa iniciativa organiza semanas de treinamento, similares às escolas de verão e inverno que temos no Brasil, e também oferece uma premiação em dinheiro caso algum time ou indivíduo consiga trazer a medalha de ouro para o solo indiano.

Em janeiro de 2025, convidaram os usuários do codeforces a participarem de contests longos mensais [2], cada contest com duração de 10 dias. Li os problemas de janeiro e, apesar de não serem necessariamente originais, todos os enunciados foram reformulados para remeterem a alguma parte do universo do jogo Hollow Knight [3]. Como esse é um dos meus jogos preferidos, resolvi tentar resolver os problemas e não me decepcionei porque encontrei um muito bonito e memorável com o título Crystal Peak.

2. Enunciado

Existe uma colina de cristal com N níveis. O nível i contém $2i - 1$ números. Também é dado um array a com $2n - 1$ números, esse é o nível mais baixo da colina. Como exemplo, essa é uma estrutura com $N = 4$ e $a = [5, 1, 4, 7, 6, 2, 3]$.

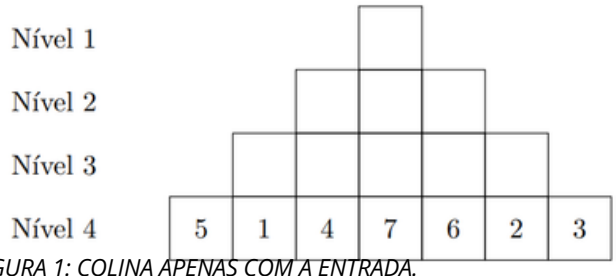

FIGURA 1: COLINA APENAS COM A ENTRADA.

O valor de cada célula nos níveis 1 a $N - 1$ são definidos como a mediana das três células abaixo. No exemplo anterior, essa é a colina preenchida:

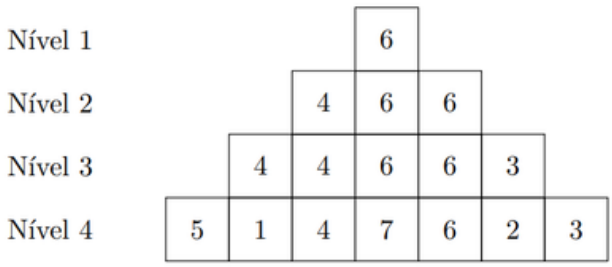

FIGURA 2: COLINA COM TODAS CÉLULAS PREENCHIDAS E PICO = 6.

O elemento no nível 1 é o topo da colina. Seu objetivo é calcular qual o topo da colina. Para o exemplo acima, seu código deve imprimir 6.

Restrições:
• $1 \le N \le 10^5$.
• $-10^9 \le a_i \le 10^9$, os elementos do array a, apenas números inteiros.

3. Solução ingênua

A mediana de um array com uma quantidade ímpar de elementos, é o elemento do meio do array quando ele está ordenado:

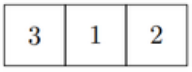

FIGURA 3: O ARRAY ORDENADO É [1, 2, 3]. ASSIM, SUA MEDIANA É 2.

Uma primeira ideia é preencher todas as células, desde o sopé até o topo para encontrar a resposta. Ordernar 3 números para encontrar a mediana leva tempo $O(1)$ mas devemos calcular muitos valores para chegar até o topo.

A colina com N níveis tem

$1 + 3 + 5 + ... + 2n - 1 = \sum_{k=1}^{N} (2k - 1) = N^2$ células.

Assim, mesmo se desconsiderarmos o último nível com $2n - 1$ elementos (que já vem preenchido na entrada), ainda precisaríamos calcular $O(N^2)$ células. Como $N \le 10^5$, essa solução não passa no tempo.

4. Um problema mais fácil

A verdade, caro leitor, é que não fui muito honesto em relação ao enunciado. As restrições acima são para a versão difícil do problema, porém, existia uma versão mais fácil com restrições menores:

Restrições, versão fácil:
• $1 \le N \le 10^5$.
• $0 \le a_i \le 1$, os elementos do array a, apenas números inteiros.

Essas restrições não ajudam a melhorar a nossa solução anterior porque ela só dependia de N e não de a_i. No entanto, com a_i assumindo poucos valores possíves, podemos testar alguns casos manualmente e procurar por alguma propriedade que nos ajude.

Uma observação interessante é que, se existem 2 números iguais lado a lado, esses números flutuam, em suas colunas, do nível mais baixo até o mais alto, independentemente dos elementos adjacentes a eles:

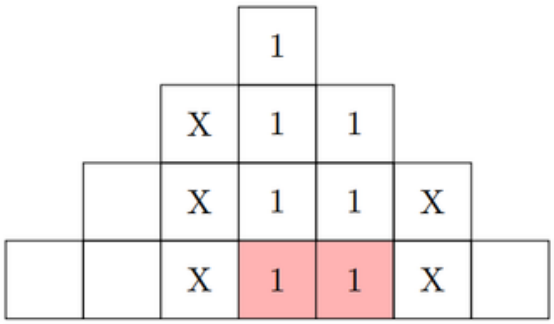

FIGURA 4: X PODE ASSUMIR QUALQUER VALOR, E O NÚMERO 1 É EMPURRADO PARA CIMA NAS 2 COLUNAS CENTRAIS. PODEMOS IGNORAR OS OUTROS NÚMEROS PARA CALCULAR QUALQUER CÉLULA NESSAS DUAS COLUNAS.

Mas o que acontece quando estamos na borda da colina e o vetor do nível acima diminui de tamanho? Primeiro, vamos assumir que não existem dois valores iguais mais próximos da coluna central (a coluna do pico da colina, i.e., da coluna com nossa resposta), porque dois números iguais, lado a lado, bloqueiam aquela coluna com aqueles valores:

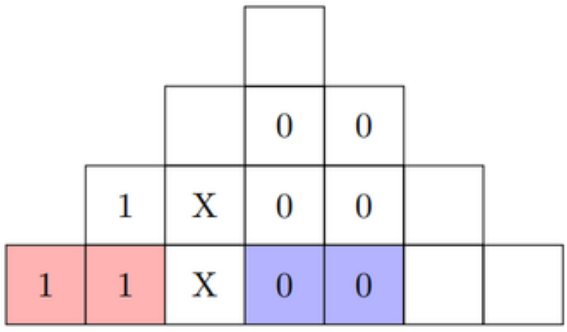

FIGURA 5: OS NÚMEROS 1 NÃO CONSEGUEM INFLUENCIAR A COLUNA CENTRAL PORQUE OS DOIS 0 BLOQUEIAM AQUELA COLUNA, MESMO QUANDO $X = 1$.

Nesse caso, os outros números formam uma sequência alternada entre 0s e 1s e os valores iguais, mesmo na borda da colina, deslizam para perto da coluna central. Veja como os dois números 1 deslizam para a coluna central quando a = [1, 1, 0, 1, 0, 1, 0]:

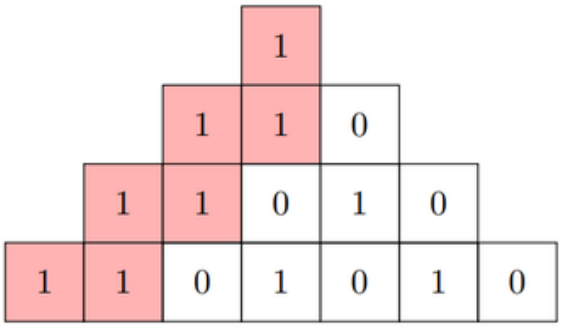

FIGURA 6: OS DOIS NÚMEROS 1 À ESQUERDA SÃO EMPURRADOS PARA CIMA MAS AGORA DESLIZAM RUMO AO CENTRO.

Por essas observações, nossa resposta será o número mais próximo da coluna central desde que ele seja adjacente a algum valor igual a ele.

Para termos um algoritmo, precisamos apenas resolver o caso em que a sequência de entrada é alternada. Nesse caso, precisamos saber apenas a quantidade de níveis na nossa colina. Se a quantidade de níveis é par, temos o complemento do elemento no centro do array a, se a quantidade de níveis é ímpar, é o próprio elemento:

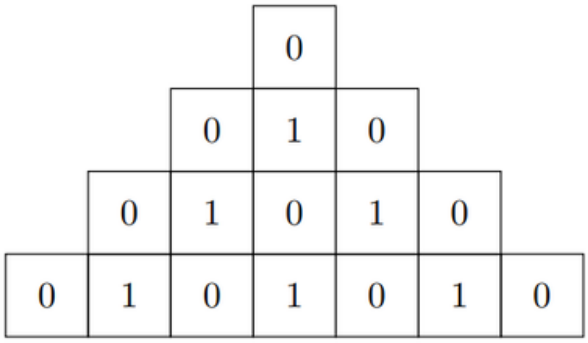

FIGURA 7: A SEQUÊNCIA DE ENTRADA CONTINUA ALTERNADA QUANDO SOBE OS NÍVEIS DA COLINA.

Após toda essa análise, conseguimos resolver a versão fácil do problema, com $O(N)$ tempo. Por praticidade, usamos a 1-indexado, i.e., a_{index} ∈ [1, $2n$ − 1].

```cpp
int solve_easy(vector<int>& a, int n) {
    int leftside_double_idx = -INF;

    for (int i = n; i >= 1; i--) {
        if (a[i] == a[i - 1]) {
            leftside_double_idx = i;
            break;
        }
    }

    int rightside_double_idx = INF;

    for (int i = n; i < 2 * n; i++) {
        if (a[i] == a[i + 1]) {
            rightside_double_idx = i;
            break;
        }
    }

    int dist_left_double = n - leftside_double_idx;
    int dist_right_double = rightside_double_idx - n;

    int min_dist = INF;
    int result_idx = -1;

    if (dist_left_double <= dist_right_double) {
        min_dist = dist_left_double;
        result_idx = leftside_double_idx;
    } else {
        min_dist = dist_right_double;
        result_idx = rightside_double_idx;
    }

    if (min_dist <= n) {
        return a[result_idx];
    } else {
        return get_alternating_answer(a, n);
    }
}

int get_alternating_answer(vector<int>& a, int n) {
    if ((n - 1) / 2 % 2 == 0)
        return a[n];
    else
        return 1 - a[n];
}
```

CÓDIGO 1: SOLUÇÃO DA VERSÃO FÁCIL.

5. Resolvendo a versão difícil

Uma propriedade da mediana é que não importa quais os elementos do array, apenas a ordem relativa entre eles. Isso é muito útil para nós e é o truque que nos permite resolver o problema.

Se propomos que o resultado é ≤ *guess*, podemos transformar o array a de entrada de modo a conter apenas 0 ou 1. Os valores $a_i \leq guess$ se transformam em 0 e os valores $a_i > guess$ se transformam em 1. Calculamos a resposta usando nossa solução da versão fácil, adaptando nosso chute de acordo com o valor no pico da colina. Qual o melhor valor para um chute? O meio do array é uma boa alternativa porque podemos sempre descartar metade do array ao fazer um chute e uma verificação:

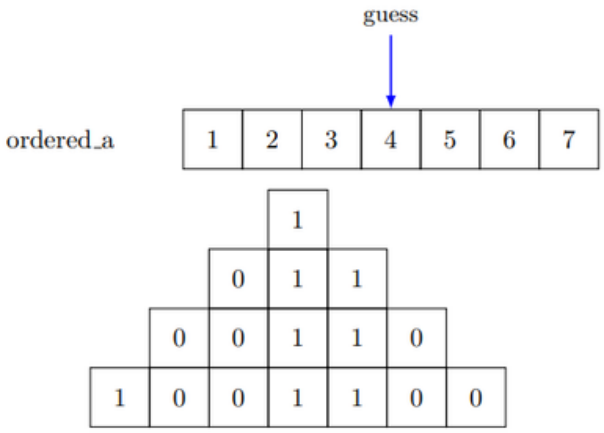

FIGURA 8: TRANSFORMAÇÃO DA COLINA DA FIGURA 1 QUANDO guess = 4 E SEU PREENCHIMENTO. COMO O RESULTADO É 1, A RESPOSTA É MAIOR QUE 4.

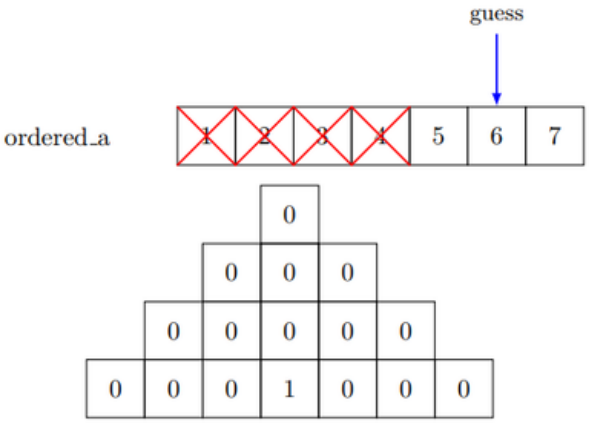

FIGURA 9: TRANSFORMAÇÃO DA COLINA DA FIGURA 1 QUANDO *guess* = 6 E SEU PREENCHIMENTO. COMO O RESULTADO É 0, A RESPOSTA É 6 (*guess*) OU MENOS.

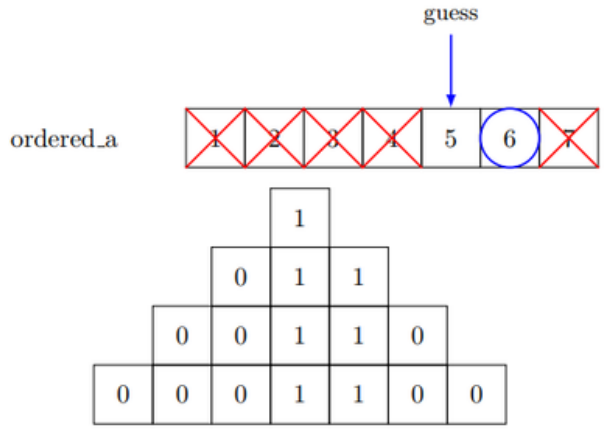

FIGURA 10: TRANSFORMAÇÃO DA COLINA DA FIGURA 1 QUANDO *guess* = 5 E SEU PREENCHIMENTO. COMO O RESULTADO É 1, A RESPOSTA É MAIOR QUE 5.

```cpp
int solve_hard(vector<int>& a, int n) {
    vector<int> ordered_a = get_ordered(a);

    int low = 1;
    int high = 2 * n - 1;
    int best = 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        vector<int> simpler_problem_a =
            build_simpler_array(a, n, ordered_a[mid]);
        int cur_result = solve_easy(simpler_problem_a, n);

        if (cur_result == 0) {
            best = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return ordered_a[best];
}

vector<int> build_simpler_array(vector<int>& a, int n, int val
    vector<int> simpler_problem_array(2 * n);

    for (int i = 1; i < 2 * n; i++) {
        if (a[i] <= val) {
            simpler_problem_array[i] = 0;
        } else {
            simpler_problem_array[i] = 1;
        }
    }

    return simpler_problem_array;
}
```

CÓDIGO 2: SOLUÇÃO DA VERSÃO DIFÍCIL.

A complexidade total fica a complexidade da ordenação mais a busca binária com $solve\_easy$ dentro dela. Como $solve\_easy$ é $O(N)$, a complexidade total de tempo é $O(N \log N)$.

6. Implementação

A implementação completa encontra-se em https://pastebin.com/idaSMmjS.

7. Considerações finais

O que torna esse problema interessante para mim é a simplicidade do enunciado e o processo de transformação da versão difícil para a versão fácil.

Ainda, com exceção das ideias e exploração inicial, para resolver esse problema precisávamos apenas de conhecimento sobre busca binária, que costuma ser um dos primeiros algoritmos ensinados em cursos de computação. Isso, para mim, aumenta ainda mais a beleza do problema.

Por fim, em 2026, foi publicado um blog no codeforces com uma breve descrição dessa ideia e problemas relacionados [1].

Referências

[1] Arvind "arvindr9" Ramaswami. A technique for some median-related problems. Disponível em: https://codeforces.com/blog/entry/154631. Acesso em: 21 jun. 2026.
[2] Naveen "evenvalue" Kulkarni. Invitation to goforgold long challenge jan 2025. Disponível em: https://codeforces.com/blog/entry/138266. Acesso em: 21 jun. 2026.
[3] Hollow knight. Disponível em: https://en.wikipedia.org/wiki/Hollow_Knight. Acesso em: 21 jun. 2026.