Tutorial: Equilibrando Árvores

Pedro Henrique Lima Ferreira

Para zerar o valor da raíz, o vértice 1, é necessário que todos os seus descendentes tenham o mesmo valor que a raíz. Então, primeiramente, devemos ajustar o valor de todos os descendetes da raíz para terem o mesmo valor da raíz.

Como fazer isso? Seja o vértice X um filho direto da raíz, e suponha que todos os descendentes de X estejam ajustados para terem o mesmo valor que X. Para fazer o vértice X e seus descendentes terem o mesmo valor que o da raiz gastaremos $abs(V_x - V_1)$ movimentos!

Agora a nova pergunta é como ajustar o valor dos descendentes de X para terem o mesmo valor de X. Podemos simplesmente usar a mesma ideia utilizada para ajustar o valor de X para o valor da raiz. Isso sugere um algoritmo onde apenas vamos somando na resposta a diferença absoluta entre o valor de um vértice i e seu vértice pai p_i. Para descobrir o pai de cada vértice, podemos rodar uma busca em profundidade ou largura a partir da raiz.

Ps: Devemos também adicionar na resposta o valor da raiz, pois o objetivo da questão é zerar todos os vértices.

Tutorial: Jogo de Arco

José Marcos da Silva Leite

Tutorial: Soma Audaciosa

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido através de um algoritmo de programação dinâmica similar à solução do problema `Subset Sum`.

Primeiramente, é necessário computar todos os números primos dentro do intervalo $[2, 10000]$ e armazená-los em um vetor `primes`, isto pode ser feito facilmente através do crivo de Eratóstenes:

```
vector<bool> table(MAX_P+1,true);
table[0] = table[1] = false;
table[2] = true;
primes.push_back(2);
for(int i=4;i<=MAX_P;i+=2)
    table[i] = false;
for(int i=3;i*i<=MAX_P;i+=2){
    if(table[i]){
        for(int j=i*i;j<=MAX_P;j+=i){
            table[j]=false;
        }
    }
}

for(int i=0;i< (int) table.size();i++){
    if(table[i]){
        primes.push_back(i);
    }
}
```

Uma vez que os primos estejam computados, utiliza-se o seguinte algoritmo de programação dinâmica.

```
dp.assign(primes.size()+1,vector<bool>(MAX_P+1,false));
dp[0][0] = true;
for(int i=1;i<=primes.size();i++){
    for(int j=0;j<=MAX_P;j++){
        dp[i][j] = dp[i-1][j];
        if(primes[i-1]<=j){
            dp[i][j] = dp[i][j] || dp[i][j-primes[i-1]];
        }
    }
}
```

Dada a tabela de programação dinâmica, qualquer desafio q pode ser respondido partindo da última linha da tabela e reconstruindo a solução:

```
void trace(int  q){
    stack<int> s;
```

```
    int i,j;
    i = primes.size();
    j = q;
    bool ans = dp[i][j];
    while(i>0 && j>0 && ans){
        if(j>=primes[i-1] && dp[i][j-primes[i-1]]){
            s.push(primes[i-1]);
            j-=primes[i-1];
        }
        else if(dp[i-1][j]){
            i--;
        }
    }
    if(s.empty()){
        cout << "Impossivel!\n";
    }
    while(!s.empty()){
        cout << s.top();
        s.pop();
        cout << (s.empty()? "\n": " ");
    }
}
```

Outra possível estratégia é, dado um natural X_i qualquer:

- Se X_i é 1, então a resposta tem que ser `Impossivel!`.
- Se X_i é par, então o programa imprime uma quantidade $X_i/2$ de números `2`.
- Se X_i é ímpar, então o programa imprime número é ímpar, então o programa imprime uma quantidade $(X_i - 3)$ de números `2` e apenas um `3`.

Isto funciona pois qualquer valor pode ser expresso como soma sobre uma sequência de `2`s e `3`s.

Tutorial: Compartilhamento de Bateria

Edson Alves da Costa Júnior

Como a entrada tem valores pequenos, é possível simular os ciclos de carga, o que resulta em uma solução $O(a)$ no pior caso.

Contudo, este problema tem uma solução $O(1)$. O objetivo é encontrar o maior valor inteiro $t = T$ tal que

$$a - tm \geq b + tn$$

Isolando o valor de t obtemos

$$t \leq \frac{a-b}{m+n}$$

Logo

$$T = \left\lfloor \frac{a-b}{m+n} \right\rfloor$$

Tutorial: A Revolução Javaxiana

Rodrigo Guimarães Araújo

Tutorial: Finanças

José Marcos da Silva Leite

Tutorial: Dialeto Dothraki

Rodrigo Guimarães Araújo

Tutorial: Controlcê Controlvê

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido através de simples simulação do enunciado e de certa forma lembra o método de compressão LZ77.

Seja S o texto da monografia obtido até o momento e T o texto que ainda precisa ser digitado. Devemos procurar em S a maior subpalavra que é prefixo de T e adicioná-la ao final de S. Caso não exista tal subpalavra, escrevemos o primeiro caractere de T.

No pior caso, este algoritmo levará tempo $\Theta(n^2)$ em que n é o tamanho da entrada.

Tutorial: Talheres

Vinicius Ruela Pereira Borges

O problema é resolvido calculando-se o menor valor entre a quantidade de garfos e facas $p = min(G, F)$, pois formam assim um par de garfo e faca. Em seguida, deve-se somar tal valor com a quantidade de colheres e verificar a quantidade máxima de pessoas que conseguirão jantar como $min(p + C, N)$.

Tutorial: Correção de Provas

Daniel Saad Nogueira Nunes

Este problema pode ser resolvido utilizando o paradigma de busca completa.

Seja x um número inteiro, x pode ser interpretado como uma configuração de questões da prova de Robervaldo, de forma que se o i-ésimo bit de x estiver ligado, então a i-questão foi resolvida nesta configuração.

Tendo isso em mente, é só variar x sobre o espaço de busca que envolve todas as configurações possíveis, isto é, no intervalo $[0, 2^N - 1]$, e selecionar apenas aquelas configurações que possuem H bits 1.

Cada uma destas configurações é inserida em um vetor solução que é impresso ao final.

Abaixo segue o esboço da solução:

```
void preprocess(int h){
   for(int i=0;i<(1<<n);i++){
      int b = count_set_bits(i);
      if(b ==h){
         solution.push_back(i);
      }
   }
}
```

A complexidade final desta solução é $\Theta(2^n)$.

Tutorial: Drone

Vinicius Ruela Pereira Borges

Uma solução para o problema envolve uma técnica de busca em largura baseada em grafo implícito. A ideia consiste em construir um grafo, em que cada vértice representa uma localização da rua e as arestas estão associadas às operações de levar o drone de uma localidade para outra. A criação dos vértices está condicionada aos limites da rua e deve-se tratar o caso específico do botão "/2", que não pode resultar em uma localidade

Primeiramente, considere Q a fila utilizada no processo de busca em largura, e que ela armazene um par (x,nro) que representa a localidade x do drone e nro indica a quantidade de movimentos realizados pelo drone que o levaram até a localidade x. O pseudo-código que resolve o problema é fornecido abaixo:

- Insira o par $(N, 0)$ na fila Q;
- **enquanto** Q não for vazia faça:
  1. $(x, nro) \leftarrow$ obtém o primeiro par da fila Q;
  2. desenfileira Q;
  3. **se** x == M **então** imprime nro; encerra o programa;
  4. **se** $x + 1 \leq 10^5$ e $x + 1$ não foi visitado, enfileira o par $(x + 1, nro + 1)$ em Q;
  5. **se** $x - 1 \leq 10^5$ e $x - 1$ não foi visitado, enfileira o par $(x - 1, nro + 1)$ em Q;
  6. **se** $x * 2 \leq 10^5$ e $x * 2$ não foi visitado, enfileira o par $(x * 2, nro + 1)$ em Q;
  7. **se** $x/2$ é par e $x/2 > 0$ e $x/2$ não foi visitado, enfileira o par $(x/2, nro + 1)$ em Q;

Tutorial: Estrutura do Rasta

Rodrigo Guimarães Araújo

Tutorial: Formigas

Vinicius Ruela Pereira Borges

Uma solução simples para o problema é empregar uma variável que indica se existe uma rã ou não na entrada do formigueiro. Considere também uma variável `total` para armazenar a quantidade total de formigas ingeridas pelas rãs sobreviventes. Se existir uma rã na entrada do formigueiro, deve-se contabilizar as formigas que esta rã está ingerindo. Se aparecer outra ou não aparecer nenhum outro animal, tal contabilização é acrescentada à variável `total`.

Tutorial: Tabela Trigonométrica

Edson Alves da Costa Júnior

Primeiramente, é preciso converter n de ângulos para radianos. Em seguida, basta multiplicar o resultado do cosseno por 10^{m+1}, fazer a parte inteira igual a a e fazer $b = 10^{m+1}$. Após dividir ambos pelo maior divisor comum d, a fração resultante aproximará o valor do cosseno com erro inferior a 10^{-m}.

A solução tem complexidade $O(1)$.