Tutorial: Alfabeto da Gamma

Guilherme Ramos

Há diversas formas de resolver o problema. Uma delas é ler as dimensões e apresentar cada letra individualmente. Para isso, basta usar um vetor das letras do alfabeto e avançar incrementando um índice i, que deve ser zerado a cada 26 iterações (sempre que passar pelo 'Z'). Claro, a cada M é preciso inserir uma quebra de linha. O programa encerra após apresentar os $N \cdot M$ caracteres que compõem o tapete.

Tutorial: Bomberman

Daniel Saad Nogueira Nunes

Essa questão poderia ser resolvida com dois laços de repetição aninhados variando de i e j de $-x$ a x. Sempre que $i = 0$ ou $j = 0$ a posição é marcada como um asterisco (explosão), caso contrário, um espaço em branco é impresso. O algoritmo abaixo ilustra o processo.

Algorithm 1: Algoritmo para imprimir a explosão do Bomberman

```
1 for( i ← −x to x )
2     for( j ← −x to x )
3         if( i = 0 or j = 0 )
4             print("*")
5         else
6             printSpace()
7     printLine()
```

Tutorial: Dia de Trote

João Carlos Gonçalves de Oliveira

O problema pede para realizar atualizações em um vetor e responder consultas em range. Uma forma de resolver por força bruta seria simular cada ação, alterando um vetor e depois iterando com um loop for para responder as perguntas. Essa abordagem possui complexidade $\mathcal{O}(N \times Q)$ no pior caso, o que não é rápido o suficiente para o limite de tempo.

Para otimizar, podemos utilizar uma estrutura de dados de intervalos. Deixo aqui uma solução utilizando uma Árvore de Segmentos (*Segment Tree*), mas existem outras soluções diferentes com estruturas diferentes.

Cada nó da nossa Segment Tree representará um intervalo fechado $[l, r]$ e armazenará três informações essenciais:

- `first`: O valor do primeiro elemento do intervalo (ou seja, $A[l]$).
- `last`: O valor do último elemento do intervalo (ou seja, $A[r]$).
- `ans`: A quantidade de posições válidas acumuladas estritamente dentro desse intervalo.

O merge pode ser feito da seguinte forma: a resposta base do pai será a soma das respostas internas de seus filhos: `left.ans` + `right.ans`. No entanto, precisamos verificar a fronteira entre os dois blocos. O último elemento do bloco da esquerda (`left.last`) está imediatamente ao lado do primeiro elemento do bloco da direita (`right.first`). Se `right.first` > `left.last`, significa que descobrimos uma nova posição válida. Portanto, somamos +1 na resposta do pai.

```
Node merge(Node & left, Node & right) {
    if (left.ans == -1ll) return right;
    if (right.ans == -1ll) return left;
    return {
        left.first,
        right.last,
        left.ans + right.ans + (right.first > left.last)
    };
}
```

Tutorial: E.T.

Jeremias Moreira Gomes

Mantemos cinco contadores, um para cada luz do OVNI.
Ao ler um registro de luzes, incrementamos os contadores correspondentes às posições marcadas com `*`.

Tutorial: Gooool

Jeremias Moreira Gomes

Seja

$$f(n) = T \cdot \sqrt{n} + G \cdot \log_2(n+1) + D$$

a função que calcula o atraso total da transmissão para n espectadores.
Observe que $f(n)$ é monotonicamente crescente, pois:

- o termo $T \cdot \sqrt{n}$ cresce quando n aumenta;
- o termo $G \cdot \log_2(n+1)$ cresce quando n aumenta;
- D é constante.

Portanto, existe um maior valor de n tal que:

$$f(n) \leq X.$$

Como a função é monotônica, podemos utilizar busca binária sobre a resposta.
Para um candidato m, basta verificar se:

$$T \cdot \sqrt{m} + G \cdot \log_2(m+1) + D \leq X.$$

Caso a desigualdade seja satisfeita, podemos tentar valores maiores. Caso contrário, devemos buscar valores menores. Como cada verificação é feita em tempo constante e a busca binária realiza $O(\log(N))$ iterações, essa é a complexidade da solução.

Tutorial: Herói das Cartas

João Carlos Gonçalves de Oliveira

É necessário analisar o real impacto da regra do "Herói". Sem essa regra, encontrar o maior produto de três elementos em um array exige testar os três maiores valores do array e também os dois menores valores (mais negativos) multiplicados pelo maior valor positivo. No entanto, a habilidade permite que o jogador multiplique o poder de uma das cartas por -1, de forma opcional. Seja um trio de cartas escolhido com valores x, y e z, o produto original dessas cartas é um valor P. Se escolhermos aplicar a regra do herói em qualquer uma dessas três cartas, o produto final se tornará exatamente -P. Como a escolha de usar o herói é opcional, o jogador sempre pode escolher entre o produto original P e o produto invertido -P. Assim, o maior valor possível que pode ser extraído de um trio específico é, portanto, o máximo entre P e -P. Matematicamente, isso é a exata definição de valor absoluto:

$$\max(P, -P) = |P| = |x \cdot y \cdot z| = |x| \cdot |y| \cdot |z|$$

Como provamos que o score máximo obtido de qualquer trio de cartas será sempre o produto dos valores absolutos dessas cartas, o problema se reduz a uma simples escolha gulosa: para maximizar o produto absoluto, basta escolher as três cartas que possuem os maiores valores absolutos individualmente. Não importa se os números são originalmente positivos ou negativos, a regra do herói garante que qualquer combinação de sinais negativos que resulte em um produto final negativo poderá ser invertida para um valor positivo.

Tutorial: Império dos Bits

Felipe Louza

O algoritmo que resolve esse problema é o seguinte:

Algorithm 1: Algoritmo para calcular o nível imperial de um cidadão

Input: Um inteiro positivo x
Output: O nível imperial de x

```
1 ans ← 0
2 while x ≠ 1 do
3 |   x ← popcount(x)
4 |   ans ← ans + 1
5 return ans
```

A operação popcount conta quantos 1s existem na representação binária de x. É possível respondê-la desde o jeito mais ingênuo, fazendo sucessivas divisões por dois, como métodos mais eficientes de manipulação de bits, como o `std::popcount` do C++20.

Tutorial: Jogadores Icônicos

Daniel Saad Nogueira Nunes

Uma forma simples de resolver esse problema é usando mapeamento por meio de um dicionário. Criaremos um dicionário que mapeia cada par (n, t) em uma string j, em que n é um inteiro com o número da camisa, t uma string com o nome do time e j a string com o nome do jogador.

Uma vez que o dicionário seja construído, basta ler cada linha da entrada, extrair o número da camisa e o nome do time, e usar o dicionário para obter o nome do jogador correspondente. O algoritmo abaixo ilustra esse processo.

Algorithm 1: Algoritmo para mapear o número da camisa e o nome do time ao nome do jogador

```
1 foreach linha da entrada do
2 |   Extrair n, t e j
3 |   D[(n, t)] ← j
4 foreach linha da consulta do
5 |   Extrair n e t
6 |   PRINT(D[(n,t)])
```

Em `C++` basta usar o `std::map` ou `std::unordered_map` para implementar o dicionário.

Tutorial: Lógica dos Sinais

Daniel Saad Nogueira Nunes

Esse problema é uma variação do *Subset Sum* e também pode ser resolvido usando programação dinâmica conforme a seguinte modelagem recursiva:

$$dp(i,j) = \begin{cases} \textbf{true} & \text{se } i = n \text{ e } j = k \\ \textbf{false} & \text{se } i = n \text{ e } j \neq k \\ dp(i+1, j+v[i]) \textbf{ or } (i+1, j+v[i]) & \text{se } i < n \end{cases}$$

Com essa modelagem, basta calcular $dp(0,0)$ para obter a resposta do problema. Para evitar computação de estados repetidos, usamos uma tabela de programação dinâmica para armazenar os resultados já calculados. O algoritmo abaixo ilustra esse processo.

Algorithm 1: Algoritmo $dp(i,j)$ para resolver o problema usando programação dinâmica

```
1 if( i = n )
2 |   D[(i,j)] ← j = k
3 |   return D[(i,j)]
4 if( D[(i,j)] já foi calculado )
5 |   return D[(i,j)]
6 D[(i,j)] ← dp(i+1, j+v[i]) or dp(i+1, j−v[i])
7 return D[(i,j)]
```

Como são $n \cdot 2S$ estados distintos, em que S é a soma de todos os elementos do vetor v, a complexidade do algoritmo é $\Theta(n \cdot S)$.

Uma vez que D esteja computado, podemos recuperar a resposta do problema usando o seguinte processo.

Algorithm 2: Algoritmo para recuperar a resposta do problema usando a tabela de programação dinâmica D

```
1  if( D[(n, k)] = false )
2      PRINT("impossivel")
3      return
4  sum ← 0
5  i ← 0
6  while i < n do
7      state ← (i + 1, sum + v[i])
8      if( D[state] = true )
9          PRINT("A")
10         sum ← sum + v[i]
11     else
12         PRINT("L")
13         sum ← sum − v[i]
14     i ← i + 1
```