
# Caderno de Soluções

## Organização


# Problema A. Amizade

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: João Guska*

## Solução

Para resolver este problema, pode-se criar um vetor de *pairs* de tamanho 10, associando todos os moradores ao número de pontos de amizade acumulados (inicialmente 0). Após todas as interações, divida por 250 os pontos de cada morador e imprima todos cujo resultado seja maior ou igual a 1. Para aqueles cujo valor ultrapasse 8, deve-se limitar a saída a 8.

Complexidade: $O(N)$.

# Problema B. Bruto

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Felipe Alberto*

## Solução

A solução consiste em ler o valor $N$ e, em seguida, processar os resultados dos ataques de cada personagem. Para cada valor $X$ lido, aplicamos diretamente as regras: ataques com $X < 15$ não causam dano; ataques com $15 \leq X < 20$ causam 1 ponto de dano; e ataques com $X = 20$ causam 2 pontos de dano. Somamos todo o dano causado pelos personagens.

Ao final, comparamos o dano total com os pontos de vida do Terrível Bruto, que é igual a $N$. Se o dano acumulado for maior ou igual a $N$, o Bruto é derrotado; caso contrário, ocorre um TPK. Complexidade total: $O(N)$.

# Problema C. CS2 — A Faca de Fallen

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Joao Tanaka*

## Solução

Ideia: $s_{l..r}$ tem período $d$ sse $s_{l..r-d} = s_{l+d..r}$. Mantenha **hashes polinomiais** (de preferência duplos) em uma **árvore de segmentos** com *lazy propagation* para: 1) atribuir um único dígito a um intervalo ($O(\log n)$); 2) consultar o hash de qualquer substring ($O(\log n)$).

Para um teste 2 l r d, compare os hashes de $s_{l..r-d}$ e $s_{l+d..r}$. Se forem iguais nos dois módulos, responda Sim; caso contrário, Nao. (O hash de bloco constante usa soma geométrica pré-computada.)

Uma referência para a árvore de segmentos com lazy propagation: https://cp-algorithms.com/data_structures/segment_tree.html#range-updates-lazy-propagation

# Problema D. Deck do Erick

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Erick (Clash Royale Edition)*

## Solução

Ideia principal da solução

Queremos encontrar, entre todos os trechos contínuos de tamanho exatamente $L$, aquele que possui a menor fadiga total. O problema é que $N$ pode ser muito grande, então não dá para calcular a fadiga do zero para cada trecho.

A chave para resolver isso de forma eficiente é manter uma janela deslizante de tamanho fixo $L$ e atualizar a fadiga quando uma carta entra e outra sai. Como o custo de cada carta depende apenas de quantas vezes ela aparece na janela, basta guardar as contagens e, ao incrementá-las ou decrementá-las, atualizar a contribuição daquela carta para a fadiga. Assim, percorremos toda a sequência em $O(N)$, sempre sabendo a fadiga atual e o mínimo encontrado. Note que isso funciona porque:

- o tamanho do trecho é sempre o mesmo ($L$);
- quando passamos de um trecho para o próximo, apenas uma carta sai e uma carta entra.

# Problema E. Esmeraldas do Caos

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Ricardo Oliveira*

## Solução

Considere inicialmente a versão mais simples do jogo que contém apenas uma esmeralda. Primeiramente, vamos resolver o problema de, dada a posição inicial da esmeralda, determinar se Tails pode vencer ou não.

Considere cada estação como um estado possível do jogo, e a estação com a esmeralda como sendo o estado atual do jogo no momento. Note que o parque é um DAG (grafo direcionado acíclico), e que perde o jogo o jogador que estiver em um estado sem saída (com grau de saída igual a zero). Sob essas condições, o vencedor pode ser determinado através da teoria dos *Grundy Numbers*.

A cada estação, associamos um *grundy number* da seguinte forma: o *grundy* de uma estação sem saída (perdedora) é 0 (zero); o *grundy* das outras é dado pelo *mex*[1] dos *grundies* das estações para as quais é possível se mover diretamente. O jogador que inicia o jogo pode vencer se e somente se o *grundy* da estação inicial é diferente de 0 (zero). Você pode conferir a prova em https://cp-algorithms.com/game_theory/sprague-grundy-nim.html.

O primeiro passo da solução é calcular o *grundy number* de cada estação, usando Memorização/Programação Dinâmica.

Vamos considerar agora a versão do jogo que tem duas esmeraldas. As duas esmeraldas podem ser tratadas como dois jogos independentes: a cada turno, um jogador escolhe um jogo para mover, e perde se não há mais movimentos possíveis em ambos os jogos. Nessas condições, o jogador que inicia o jogo pode vencer se e somente se o XOR (ou-exclusivo bit-a-bit) dos *grundies* das duas estações iniciais é diferente de 0 (zero) (prova no link acima).

Note que o XOR de dois inteiros serem diferentes de 0 (zero) é equivalente aos dois inteiros serem diferentes. Assim, o problema se reduz a contar o número de pares de estações cujos *grundies* são diferentes.

Uma possível maneira de calcular a resposta é considerar $\frac{N \times (N-1)}{2}$ (todos os pares possíveis) e subtrair disso o número de pares cujos *grundies* são iguais. Para cada inteiro $g$, Seja $cnt[g]$ o número de estações cujo *grundy* é $g$. O número de tais pares pode ser dado então por $\sum_{g=0}^{N+1} \frac{cnt[g] \times (cnt[g]-1)}{2}$.

Por fim, lembre-se de tirar o módulo $10^9+7$ após cada soma e multiplicação. Use o inverso modular de 2 (módulo $10^9+7$) para calcular as divisões.

Complexidade total: $O((N+M)\lg N)$ (considerando que o cálculo do *mex* é feito com um `set`).

[1] *menor excluído*: menor natural não presente no conjunto. Ex: $mex\{0,1,2,4,6\} = 3$

# Problema F. Factorio

*Tempo limite: 4000 ms*
*Memória limite: 512 MiB*
*Autor: Eduardo Marca*

## Solução

O problema pode ser pensado como um problema de fluxo de grafos com várias fontes e sumidouros. Cada máquina geradora atua como uma fonte e cada máquina coletora atua como um sumidouro do grafo e as velocidades são as capacidades de cada aresta no grafo. Os algoritmos tracionais de fluxo máximo atuam sobre grafos com uma única fonte e um único sumidouro, mas é possível transformar o grafo do problema em um grafo desse tipo. Para isso, cria-se uma nova fonte (chamada de super fonte) no grafo e arestas com capacidade infinita que vão da super fonte para todas as outras fontes do grafo. De forma similar, cria-se um novo sumidouro (chamado de super sumidouro) no grado e arestas com capacidade infinita que vão dos sumidouros para o super sumidouro do grafo. Com esse novo grafo, é posível aplicar um algoritmo de fluxo máximo (como o Dinic) que usa como fonte a super fonte criada e usa como sumidouro o super sumidouro criado.

Complexidade: $O(N^2 \times (M + N))$

Uma referência para o algoritmo de fluxo máximo de Dinic: https://cp-algorithms.com/graph/dinic.html

# Problema G. Grand Theft Auto

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Gabriel Gimenez Vieira*

## Solução

Precisamos comprar exatamente $K$ produtos de um total de $N$ produtos disponíveis, gastando o mínimo possível, sem ultrapassar o orçamento $A$. A estratégia ótima é simples: escolher os $K$ produtos mais baratos. O algoritmo consiste em ler a entrada com $N$, $K$, $A$ e os $N$ preços, ordenar os preços em ordem crescente, somar os $K$ primeiros preços (os mais baratos) e verificar se a soma é menor ou igual ao orçamento $A$. Se for possível, imprimimos a soma; caso contrário, imprimimos "Nao e possivel".

Complexidade: $O(NlogN)$

# Problema H. Hollow Knight

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Erick Rosim*

## Solução

A solução é uma variação do clássico *problema da mochila 0/1*, com a diferença de que o Cavaleiro pode ultrapassar o limite de entalhes uma única vez utilizando a técnica de *Sobrecarga*.

Utiliza-se programação dinâmica para resolver o problema. Definimos o estado:

$$dp[pos][valor][sobrecarga]$$

onde:

- `pos` representa o índice do amuleto atual;
- `valor` é o total de entalhes já utilizados;
- `sobrecarga` indica se a habilidade de sobrecarga já foi usada (0 = não, 1 = sim).

As transições são:

1. Ignorar o amuleto atual;
2. Equipá-lo normalmente, caso ainda caiba dentro de $M$;
3. Equipá-lo utilizando a sobrecarga, caso ainda não tenha sido usada.

A complexidade da solução é $O(N \times M)$, já que existem aproximadamente $2 \times N \times M$ estados, cada um processado em tempo constante.

Assim, o algoritmo encontra o **máximo poder de combate** possível, considerando o uso opcional e único da sobrecarga.

# Problema I. Injustice

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Erick Rosim*

## Solução

A história descreve dois parâmetros essenciais: o valor energético base $P$ e o fator de intensificação $M$. A solução consiste simplesmente em multiplicar $P$ e $M$ conforme indicado pelo enunciado e imprimir o resultado.

A complexidade de tempo é $O(1)$ e a complexidade de memória também é $O(1)$.

# Problema J. Jornada do Ethan

*Tempo limite: 3000 ms*
*Memória limite: 1024 MiB*
*Autor: Lucas Severino*

## Solução

Este problema pode ser resolvido utilizando Busca em Largura (BFS) adicionando estados.

**Modelagem do Estado:** Cada estado é representado por $(x, y, c)$ onde:

- $(x, y)$ é a posição atual de Ethan no mapa.
- $c$ é o nível da chave que ele possui (0, 1, 2 ou 3)

**Algoritmo:**

1. Iniciar BFS na posição $E$ $(0, 0)$ com nível de chave 0.
2. Para cada estado $(x, y, c)$, explorar os 4 vizinhos adjacentes
3. Se encontrar:
    - Parede (`#`): ignorar
    - Portão: verificar se $c \geq$ nível do portão
    - Chave $n$: só coletar se $k = n - 1$ (ordem obrigatória)
    - Rose (`R`): retornar a distância
4. Marcar estados visitados como $(posição, nível\_chave)$ para evitar reprocessamento de estados ja visitados com chaves melhores
5. Se a fila esvaziar sem encontrar Rose, retornar -1

**Observação:** O controle de visitados deve considerar o par $(posição, nível\_chave)$. Uma mesma posição pode ser visitada múltiplas vezes com diferentes níveis de chave, mas só processamos se o novo nível for estritamente maior que o anterior naquela posição.

Complexidade: $O(L \times C)$

# Problema K. Kurt, o Jogador de Golfe

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: João Vitor Lima Martins*

## Solução

O problema consiste em simular tacadas de minigolfe em um campo infinito contendo $N$ obstáculos em posições fixas. A bola sempre se move em linha reta a partir de uma direção inicial, perdendo uma unidade de velocidade por célula percorrida. Encontrar um portal muda a direção imediatamente, entrar em areia faz a tacada terminar e alcançar o buraco finaliza o jogo. O restante do campo não produz efeitos.

Como o movimento é sempre horizontal ou vertical, não é necessário simular passo a passo. Em vez disso, para cada tacada basta identificar qual é o primeiro obstáculo alinhado com a direção atual e cuja distância seja menor ou igual à velocidade disponível. Se tal obstáculo existe, a bola avança exatamente até ele; caso contrário, avança até a velocidade zerar. A natureza do obstáculo determina se a tacada continua, interrompe ou encerra o jogo.

Complexidade: $O(N \times M)$

# Problema L. Lara Croft

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Caio Welter Bogo*

## Solução

Neste problema, precisamos encontrar o ponto de ancoragem ideal para Lara Croft em uma matriz $N \times N$, onde cada posição representa um tipo de terreno da montanha. O caractere # indica uma ancoragem segura, local onde o piolet pode ser fixado, enquanto 0 representa uma superfície escorregadia, imprópria para escalada. O objetivo é determinar as coordenadas (l, c) do ponto ideal, seguindo as prioridades definidas: o ponto deve estar o mais alto possível (menor índice de linha) e, em caso de empate na mesma linha, o mais à direita (maior índice de coluna).

Como o tamanho da matriz é pequeno (N $\leq$ 20), é possível resolver o problema de forma simples e direta. A estratégia consiste em ler toda a matriz e percorrê-la linha por linha, de cima para baixo. Em cada linha, percorremos as colunas da direita para a esquerda. Assim, o primeiro caractere # encontrado representa exatamente o ponto mais alto e, dentro dessa linha, o mais à direita. Após encontrá-lo, basta imprimir suas coordenadas e encerrar o programa, já que não é necessário continuar a busca.

Essa abordagem tem complexidade $O(N^2)$ no pior caso.

# Problema M. Monstros do RicardAsh

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: RicardAsh*

## Solução

**Complexidade esperada:** $\mathcal{O}\left(n\sqrt{\max(s_i)}\right)$.

**Ideia principal:** fatoração e contagem de fatores primos.

O problema se reduz a encontrar um grupo de Pokémons tal que o **mdc de todas as forças do grupo seja maior que 1**. Isso equivale a dizer que **existe um primo $p$ que divide todas as forças do grupo**.

Portanto, basta fatorar cada $s_i$, considerar apenas os **fatores primos distintos** de cada número e contar quantas vezes cada primo aparece. A resposta será o **maior número de ocorrências de um mesmo primo**.

Os fatores primos de cada número podem ser obtidos usando crivo pré-computado ou fatoração por tentativa até a raiz quadrada.

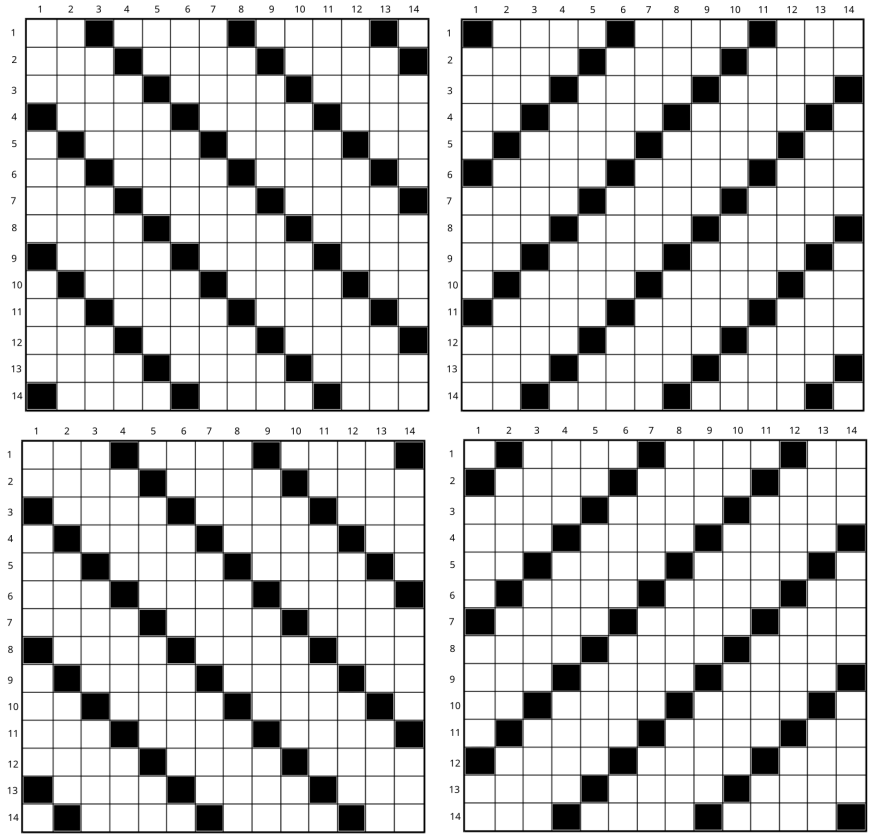
**Edge Case:** como um Pokémon não pode lutar consigo mesmo, o tamanho mínimo da resposta é sempre 1. Assim, mesmo que todos os números sejam coprimos entre si, a resposta permanece 1, e como $N$ nunca é 0 então a resposta mínima é 1.

# Problema N. Naval

*Tempo limite: 1000 ms*
*Memória limite: 256 MiB*
*Autor: Henrique Farias*

## Solução

Para achar o navio porta-aviões de maneira ótima deve-se fazer seus chutes em uma das 4 configurações a seguir:



Pois assim com 39 chutes você está cobrindo a grade toda pois nessas configurações o espaçamento entre as coordenadas chutadas não é maior do que 4, e como o navio é tem comprimento igual a 5 é garantido que você vai fazer um acerto com esses 39 chutes. Porém é importante deixar o chute na quina por último e um chute em uma das laterais em penúltimo, pois na quina como só tem duas direções adjacentes livres, permite no máximo um erro após o primeiro acerto, logo no pior caso tem 1 erro e 4 acertos até a fundar, assim $39 + 1 + 4 = 44$ está dentro do limite, o mesmo ocorre com a lateral, com 3 direções disponíveis: $38 + 2 + 4 = 44$.

Uma referência para soluções de problemas interativos: https://codeforces.com/blog/entry/45307

Complexidade: $O(1)$.