# Uma Noite na Queima: Competição de Programação

# Editorial

**Nota:** Uma grande parte dos problemas desta competição são problemas do *codeforces* que foram reescritos com as nossas histórias. Como tal, deixamos nesses problemas links para o respetivo *problema original* e para o respetivo *editorial* do *contest*, no qual podem procurar a solução do problema correspondente.

## Problema A: *Amped-up Students*

Problema original
Editorial
Cada estudante do tipo A vai fazer com que todos os estudantes tipo P à direita dele se tornem As, até ao A mais à direita dele na *string* original. Por isso, basta encontrar a maior *string* constituída apenas por Ps que ocorre depois de um A e imprimir o tamanho desta *string* como output para cada caso de teste.

## Problema B: *Looking for the best price*

Problema original
Editorial
Cada elemento do *array* C poderá apenas ser -2, 0 ou 2. Além disso, C só será 2 se A for 2 e B for 1, e C só será -2 se A for 1 e B for 2; em qualquer outro caso C é 0. Assim, queremos fazer o maior número de pares A, B (2, 1) e o mínimo de pares A, B (1, 2). Por isso basta primeiro fazer o maior número de pares (2, 1), depois o maior número de pares (1, 0) e depois o maior número de pares (0, 2). Depois disto, emparelhamos os restantes números de qualquer forma.

## Problema C: *Pennywise Shots*

Este problema é da autoria do professor André Restivo e portanto não tem *links* para o codeforces.
Este problema pode ser reduzido ao seguinte: dado um *array* de N números, temos de escolher algum conjunto de números tal que este conjunto inclui o primeiro e o último elemento e que a distância no array entre dois elementos escolhidos não seja maior do que K + 1. Assim, podemos inicializar um array *dp* de tamanho N, em que cada elemento *dp[i]* corresponde à melhor solução que podemos atingir 'bebendo' um shot na barraca *i,* respeitando a condição do K. O primeiro elemento de *dp* é igual ao primeiro elemento do

array (o que corresponde a ‘beber o shot’ nesta posição). Depois, para cada posição *i,* podemos atualizar o array com o elemento do array original na posição *i* mais o mínimo de array *dp* nas K posições anteriores. Para esta última parte, como K < 100, podemos apenas usar um *for loop*, com complexidade temporal total O(NK). Também podemos usar um *multiset* e ter uma melhor complexidade temporal de O(N log K).
Um exemplo de solução.

## Problema D: *Scheduling Conflict*

Problema original
Editorial
Em cada dia vamos, por exemplo, escolher o primeiro amigo da lista. Assim, depois de escolher um amigo para cada dia, teremos no máximo um amigo que está inserido mais vezes do que o limite. Para resolver este problema, voltamos a iterar por todos os dias e em cada dia que o amigo que está alocado vezes a mais foi escolhido podemos escolher um amigo diferente, e fazemos isto enquanto o amigo que está originalmente a mais continuar ‘fora do limite’. Ou isto será possível e imprimimos a resposta ou o problema é impossível e respondemos NO.

## Problema E: *Roaming Around*

Problema original
Editorial
Neste problema temos de simular o caminho de um personagem num grafo. A *key observation* deste problema é que, se estamos no vértice *i,* então todas as arestas para números menores do que *i* são convertidas para $p_i$ , o que revela a fórmula:
*dp[i+1] = 2 * dp[i] + 2 - dp[p[i]].*

## Problema F: *Filipe’s last Student Festival*

Problema original
Editorial
A solução apresentada no editorial é bastante completa.

## Problema H: *The Organizer*

Problema original
Editorial
A solução deste editorial também é bastante completa.