SAET 2023 - Maratona de Programação

26 de Outubro de 2022

A: Alergia

Uma ração não pode ser dada ao Thomy caso ela contenha um ingrediente ao qual ele tenha alergia. Portanto, basta iterar sobre os ingredientes: se ocorrer que um ingrediente i esteja na ração e Thomy tiver alergia a esse ingrediente i, a saída deverá ser `"N"`. Caso contrário, a saída deverá ser `"S"`, pois a ração não causará reação no cachorrinho.

Iterar sobre os N ingredientes tem complexidade assintótica $O(N)$. Essa solução passa com folga, dado que o pior caso é $N = 100$.

Complexidade total: $O(N)$.

B: Jogo

Deve-se criar 2 vetores, um contendo o talento dos defensores e outro contendo o talento dos atacantes, e ordena-los. Criar 2 variaveis auxiliares $t1$ e $t2$, as quais vão controlar diferença de atacantes e defensores no time. Além de criar 2 variaveis $soma1$ e $soma2$ para guardar a soma dos talendos de cada time.

Então devem ser feitas $n/2$ iterações, sendo n o numero de jogadores a serem escolhidos, onde a cada iteração i serão escolhidos 2 jogadores, um para cada time. Para fazer a intercalação da ordem das escolhas, é utilizado $i mod 2$, sendo que, quando i é par o time 1 escolhe primeiro e quando i é impar o time 2 escolhe primeiro.

Para cada escolha é feita uma sequência de condições:

- Se o vetor de defensores está vazio, incrementar o ultimo valor do vetor de atacantes a $soma1/soma2$ e remove-lo do vetor;
- Se o vetor de atacantes está vazio, incrementar o ultimo valor do vetor de defensores a $soma1/soma2$ e remove-lo do vetor;
- Se ambos vetores não estiverem vazios e $t1/t2 < 0$, incrementar o ultimo valor do vetor de atacantes a $soma1/soma2$, remove-lo do vetor e incrementar 1 a $t1/t2$;
- Se ambos vetores não estiverem vazios e $t1/t2 > 0$, incrementar o ultimo valor do vetor de defensores a $soma1/soma2$, remove-lo do vetor e decrementar 1 a $t1/t2$;
- Se ambos vetores não estiverem vazios e $t1/t2 = 0$, comparar o ultimo valor de ambos os vetores. Caso o maior valor seja um atacante, incrementar o ultimo valor do vetor de atacantes a $soma1/soma2$, remove-lo do vetor e incrementar 1 a $t1/t2$. Caso o maior valor seja um defensor, incrementar o ultimo valor do vetor de defensores a $soma1/soma2$, remove-lo do vetor e decrementar 1 a $t1/t2$;

Ao fim das iterações, compara $soma1$ e $soma2$ e retorna a maior soma dividido por $n/2$, numero de jogadores do time.

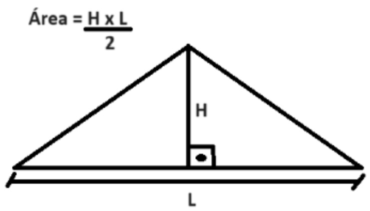
Complexidade total: $O(N \lg N)$.

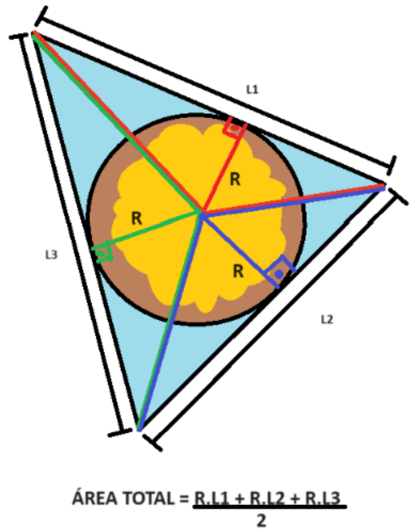
C: Vasilha Errada

Como o exercício fornece os lados do triângulo, é possível calcular sua área total pela fórmula de Heron:

Área $= \sqrt{p(p - L1)(p - L2)(p - L3)}$, sendo p o semiperímetro: $p = \dfrac{L1 + L2 + L3}{2}$.

Dessa forma:





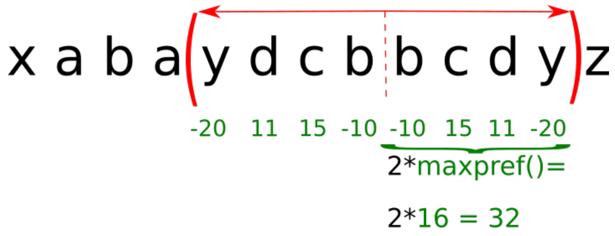
Isolando R, tem-se que $R = \dfrac{\text{Área total}}{P}$.

Complexidade: $O(1)$.

D: Drawkcabackward

O algoritmo de Manacher[1] encontra, para cada posição i da string, o maior valor de j tal que $s[i-j..i+j]$ é palíndrome, e o maior valor de j' tal que $s[i-j'+1..i+j']$ é palíndrome (isto é, o algoritmo consegue determinar, para cada posição da string, o tamanho da substring palíndrome maximal cujo centro ocorre naquela posição, tanto a de tamanho ímpar (j) quanto a de tamanho par (j')). O algoritmo tem complexidade $O(N)$.

A figura abaixo exemplifica a solução para o exemplo dado no enunciado. O tamanho obtido pelo algoritmo de Manacher é representado em vermelho:



O próximo passo é determinar o maior valor total possível para cada substring palindrome maximal. Note que, para substrings de tamanho par, isto é dado por duas vezes a soma do prefixo de maior soma na segunda metade da substring, e que, para substrings de tamanho ímpar, é dado por essa soma mais o valor da letra em seu centro. Na figura exemplificada acima, a resposta é dada por duas vezes a soma do prefixo de maior soma em $[-10, 15, 11, -20]$ (que é 16, dado pela soma de $[-10, 15, 11]$).

Esta soma pode ser obtida em $O(\lg N)$ através da construção de uma Árvore de Segmentos adaptada para responder essas consultas, conforme descrito em [2].

Complexidade total: $O(N \lg N)$.

[1] https://cp-algorithms.com/string/manacher.html

[2] https://www.geeksforgeeks.org/maximum-prefix-sum-given-range/

E: Empilha Copos

Dada uma base B, o número de copos necessários para formar um triângulo é dado por $\frac{B^2+B}{2}$. Assim, é possível:

- iterar B a partir de 1 enquanto $\frac{B^2+B}{2} \leq N$. Complexidade: $O(\sqrt{N})$;
- usar o algoritmo da busca binária para, dado um valor de B, calcular $\frac{B^2+B}{2}$ e comparar com N para seguir a busca com valores menores ou maiores. Complexidade: $O(\lg N)$;
- calcular a função inversa de $\frac{B^2+B}{2} \leq N$. Complexidade: $O(1)$.

F: Falco

Substitua cada letra s_i na string pela letra k posições após no alfabeto. Não é necessário iterar pelas k posições; utilizando dos códigos da tabela ASCII, basta calcular $(s_i - '0' + k)\%26 + '0'$.

Para evitar *overflow*, e utilizando uma propriedade da artimética modular, é possível reduzir o valor de k com $k = k\%26$ antes de realizar o cálculo.

Complexidade: $O(t)$.

G: Gafe

A solução pode ser encontrada utilizando programação dinâmica[1], tanto com tabulação ou memoização/*caching*, pois a melhor resposta para um estado $[E, D]$ pode ser determinada a partir de seus sub-estados.

No caso do problema, a resposta de um estado depende de três sub-estados:

1. Apertar o botão `E-` (somar 1 ao resultado do estado $[E - 1, D]$);
2. Apertar o botão `D-` (somar 1 ao resultado do estado $[E, D - 1]$);
3. Apertar o botão `S` (somar 1 ao resultado do estado $[E-D, D]$ ou $[E, D-E]$, dependendo de qual lado for maior. Se os dois lados tiverem mesmo nível, pode-se escolher qualquer um).

Deixa-se uma descrição da solução para cada uma das técnicas de memoização e de tabulação:

Memoização Uma função recursiva deve ser criada para solucionar o problema. Então, faz-se uma modificação: todo retorno de função é guardado na matriz de memoização, e toda vez que a função for chamada, sempre verificar na matriz se aquele estado já não foi computado anteriormente. Se já foi, retorna o valor memoizado antes de criar outras chamadas recursivas, assim podando subárvores de recursão e diminuindo a complexidade assintótica da solução.

Tabulação Cria-se uma tabela com uma matriz e preenche-se os casos base ($[0, 0], [0, 1], [0, 2], ..., [0, D]$ e $[1, 0], [2, 0], [3, 0], ...[E, 0]$). Então deve-se iterar sobre ela, calculando o estado $[i, j]$ a partir dos sub-estados já computados em iterações anteriores.

A complexidade será de $O(E \cdot D)$ com qualquer uma das técnicas de programação dinâmica, pois cada estado pode ser minimamente representado pelos volumes dos lados esquerdo e direito.

Complexidade final: $O(E \cdot D)$.

[1] Esse conceito é tão amplamente aplicável em diferentes problemas, fáceis e difíceis, que foi abordado em um TCC pensando em maratonistas: `pantheon.ufrj.br/bitstream/11422/14161/1/THNCoelho.pdf`

H: Raid

Dado os pontos esclarecidos pelas regras da raid e pelas regras da loja podemos concluir que precisamos encontrar o intervalo que a soma das magias seja a menor possível mas que seja maior ou igual do que a soma do HP de todos os monstros.

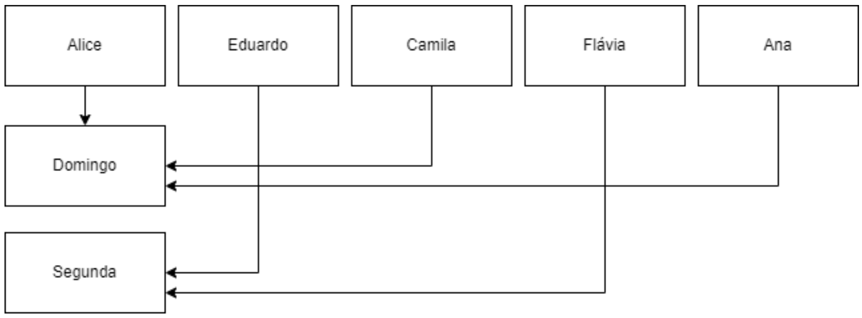
Primeiramente criamos um vetor que a posição a_i $(0 <= i <= n)$ é a soma das magias de a_0 até a_i. Neste vetor podemos acessar o valor da soma de todos os intervalos $[i, j]$ possíveis do vetor utilizando $a_i - a_j$.

Por fim, para cada i faremos busca binária calculamos o intervalo $[i, j]$ e comparamos com a soma dos HP, com $(0 <= i, j <= N)$.

Complexidade: $O(N \lg N)$

I: Lista

Como o exercício é uma ordenação, podemos apenas utilizar um método simples de vetores, logo cada vetor vai ser um dia da semana e cada dia da semana vai ter seus parâmetros para serem inseridas, por exemplo: as pessoas que tem o nome que começam com a letra A,B,C e D, vão ficar no vetor de domingo, podendo ser apenas ser implementado como um if ou case dentro de um for para fazer a verificação de todo o vetor de nomes.



Complexidade total: $O(n)$.

J: Meuzamigo

A resolução do problema trata-se de uma simples DFS (Busca em profundidade), acrescentando 1 para cada avanço em direção a folha e guardando o maior valor encontrado dentre todas as folhas.

Complexidade: O(V+A)

K: Tiras

Primeiro, é importante notar que a operação Máximo Divisor Comum é associativa[2]. Matematicamente, significa que, para caixas i, $i+1$ e $i+2$ quaisquer, temos que:

$$\text{MDC}(\text{MDC}(c_i, c_{i+1}), c_{i+2}) = \text{MDC}(c_i, \text{MDC}(c_{i+1}, c_{i+2})) = \text{MDC}(c_i, c_{i+1}, c_{i+2})$$

Dessa forma, uma sequência de operações ótima pode ser realizada em qualquer ordem, dada a propriedade associativa das operações.

Tendo tudo isso em mente, é possível encontrar uma solução gulosa[3]. Primeiro, deve-se computar o MDC M entre todas as caixas ($M = \text{MDC}(c_1, c_2, c_3, ..., c_n)$), pois, ao final, todas as caixas restantes terão tiras de tamanho M.

Em seguida, a escolha ótima se baseia no fato de que, se a caixa mais a esquerda tiver tiras de tamanho diferente de M, em algum momento será preciso realizar uma operação entre ela e a caixa seguinte, até que suas tiras resultem neste tamanho. Como já mencionado, a operação MDC é associativa, e por isso pode-se escolher realizar essa operação primeiro, sem prejuízo ou aumento na quantidade de operações totais realizadas.

Como a caixa anterior já tem tiras de tamanho M, ela não precisa de mais operações. Então pode-se pular para a caixa seguinte e realizar o mesmo processo, realizando a operação entre ela e a próxima caixa até ter tiras de tamanho M. Caso a última caixa não tenha tiras de tamanho M mesmo após todas as operações, será necessário realizar mais uma operação entre ela e a caixa anterior.

O algoritmo itera sobre as caixas duas vezes para realizar operações de MDC, e portanto tem complexidade assintótica $O(N \log c)$ se a operação MDC for $O(\log c)$ utilizando o Algoritmo de Euclides[4] com resto. Note que é possível chegar ao mesmo resultado na direção inversa (da última caixa até a primeira), e que outras soluções gulosas podem existir utilizando os mesmos princípios apresentados.

Complexidade final: $O(N \log c)$.

[2] pt.wikipedia.org/wiki/Associatividade
[3] pt.wikipedia.org/wiki/Algoritmo_guloso
[4] pt.wikipedia.org/wiki/Algoritmo_de_Euclides