Problem A. A Sequência de Simbanacci

A solução pode ser construída numa abordagem bottom-up, considerando a quantidade de Starrs para cada possível quantidade de horas vividas. São apenas quatro valores, de 0 a 3, já que eles morrem depois disso.

Para cada hora decorrida, basta deslocar as quantidades em uma hora, e atualizar a quantidade de recém nascidos multiplicando a taxa de crescimento pela quantidade de *Starrs* férteis (que já viveram pelo menos 2 horas).

Essa solução é $O(n)$.

Problem B. Beto e DP

A solução do problema parte da implementação eficiente dos quatro comandos citados, a qual depende da representação escolhida para os polinômios.

Como o grau dos polinômios podem chegar a 10^9, a representação por meio de vetores de coeficientes estoura o limite de memória estabelecido. Partindo da observação que, segundo as restrições do problema, um polinômio pode ter, no máximo, 4000 coeficientes não-nulos, é possível representar os polinômios como dicionários, onde a chave é o grau do monômio e o valor é o coeficiente associado. Esta representação permite a implementação das operações 1 e 2 em $O(\log Q)$ (ou $O(1)$, para dicionários não ordenados).

A multiplicação de polinômios tem complexidade $O(Q^2)$. Se a multiplicação for feita a cada ocorrências das operações 3 e 4, o limite de tempo será excedido. Contudo, cada atualização de $f(x)$ ou $g(x)$ modifica, no máximo, $O(Q)$ coeficientes de $h(x)$. Assim, se $h(x)$ for atualizado em cada operação do tipo 1 ou 2, elas passam a ter complexidade $O(Q)$.

Esta abordagem permite implementar a operação 3 em $O(\log N)$ através de uma única consulta ao dicionário de $h(x)$. Já a operação 4 pode ser respondida em $O(1)$, pois a resposta depende unicamente da paridade de x: caso seja par, a resposta é a paridade do coeficiente c_0 de $h(x)$; caso seja ímpar, será a paridade da soma de todos os coeficientes. Esta soma pode ser mantida em uma variável e atualizada na execução dos comandos do tipo 1 e 2.

Esta solução tem complexidade $O(Q^2)$.

Problem C. Cotas e Rateio Linear

Simular a distribuição cota a cota por meio de uma fila tem complexidade $O(Q)$, o que leva a um veredito TLE, uma vez que $Q \leq 10^{18}$. Para reduzir a complexidade, é preciso observar que, a cada iteração, será possível distribuir grupos de m das C cotas restantes aos n investidores que ainda permanecem na fila, onde

$$m = \min\{C/n, \min\{q_1, q_2, \ldots, q_n\}\}$$

e q_i é a quantidade de cotas que o investidor i ainda deseja comprar.

Desta forma, a cada iteração um ou mais investidores deixarão a fila, de modo que no pior caso a complexidade é $O(N \log N)$, considerando-se um custo $O(\log N)$ para a identificação do menor q_i a cada iteração. Como $N \leq 2 \times 10^5$, esta solução terá veredito AC.

Problem D. Dobra ou Paga

Uma solução gulosa, que escolhe sempre a carta com o maior prêmio ainda não virada, leva a uma solução WA: por exemplo, para uma fila de três cartas com valores `2 3 2`, escolher a carta com o valor `3` levaria a um prêmio de 2 reais, enquanto que escolher ambas cartas com o valor `2` resultaria em um prêmio de 5 reais.

A solução, portanto, deve avaliar, para cada carta i, se é melhor pegá-la ou não. Assim são 2^N escolhas possíveis, o que resulta em um veredito TLE. Contudo, se as cartas forem processadas da esquerda para a direita, escolher a carta i leva a um problema idêntico ao original para a fila que inicia em $i + 2$; não

escolher a carta i leva ao subproblema da fila que inicia em $i+1$. Assim, é possível resolver este problema usando programação dinâmica.

Seja $p(i)$ a soma máximas das cartas que podem ser escolhidas na fila de cartas $R_i, R_{i+1}, \ldots, R_N$. Observe que

$$p(i) = \max\{\ p(i+1), p(i+2) + R_i\ \},$$

ou seja, o melhor cenário entre não escolher e escolher a carta i. Também temos $p(j) = 0$ para $j > N$.

A solução do problema original é, portanto,

$$S = 3p(1) - \sum_{i-1}^{N} R_i,$$

que significa pagar por todas as cartas, e somar o triplo das escolhidas (o que anula o pagamento anterior e soma o dobro).

Como são N estados distintos e as transições são feitas em $O(1)$, a solução tem complexidade $O(N)$.

Problem E. Eds e a Prova Perfeita

A solução é relativamente simples, ler as entradas e verificar se as regras são atendidas analisando as respostas de cada time e as acumuladas por questão. Esta abordagem é $O(PT)$.

Problem F. F1

O problema consiste em ordenar os pilotos de acordo com os tempos obtidos, do menor para o maior tempo. Para preservar a posição da entrada em caso de empate, há duas formas: ou utilizar um algoritmo de ordenação estável (como o `stable_sort()` do C++) ou guardar o índice do piloto de acordo com a ordem da entrada.

Em qualquer caso, o número de pilotos é reduzido ($N \leq 20$), de forma que qualquer algoritmo de ordenação, seja quadrático, seja linearítmico, resolve o problema.

Problem H. Holiday na Sucupira

Podemos ver que a Floresta da Nlogonia pode ser representada por um grafo, em que os vértices são os pontos de interesse (portaria, mirantes e a cachoeira) e as arestas são as trilhas que conectam tais pontos. Estamos interessados em determinar se existe um segundo menor caminho entre os vértices 1 e N para que a Dona Marileusa possa pedalar. Vamos enumerar os seguintes passos:

1. Identifica-se todos os caminhos entre os vértices 1 e N que possuem a menor distância. Para esse propósito, podemos modificar o algoritmo de Dijkstra para armazenar todos os menores caminhos entre o vértice 1 e qualquer vértice u do grafo determinar todos os menores caminhos entre 1. A estrutura `predecessor` pode ser adaptada para armazenar uma lista de predecessores para cada vértice u. Além disso, no momento de relaxar as arestas, temos que tratar o caso em que a distância $d[v] == d[u] + w$ e incluir esse vértice u predecessor na lista de predecessores de v. Não se deve esquecer de limpar os predecessores de um vértice v e inserir como predecessor de v um vértice u toda vez que $d[v] > d[u] + w$.

2. Em seguida, devemos remover (ou marcar) todas as arestas do grafo que pertencem a algum dos caminhos mínimos encontrados. Para isso, pode-se utilizar Breadth First Search (BFS) ou Depth First Search (DFS).

3. Executa-se o algoritmo de Dijkstra novamente no grafo desconsiderando as arestas do grafo que foram marcadas ou removidas na etapa anterior. Assim, vamos obter o segundo caminho mínimo, e consequentemente, a segunda menor distância desde o vértice 1 até o vértice N.

Considere que o grafo em questão possui V vértices e E arestas. O algoritmo de Dijkstra, com as adaptações propostas, possui complexidade computacional $O(E + V \log V)$. Já o algoritmo de busca em profundidade realizado na estrutura `predecessor` tem complexidade $O(V + E)$, pois no pior caso, tal estrutura pode conter o grafo inteiro e não apenas um único caminho. Com a adaptação para demarcar as arestas do menor caminho, para cada vértice, devemos procurar pelos seus predecessores. Portanto a complexidade total dessa etapa é $O(2V + E)$. Portanto, a complexidade computacional dessa solução é $2O(E+V \log V)+O(2V+E)$, uma vez que o algoritmo de Dijkstra é executado duas vezes nessa solução.

Problem I. Ímpar/Par Sorting

Há duas soluções possíveis para este problema. A primeira delas é codificar as etapas descritas no texto. A cada iteração a sequência é dividida em duas partes iguais, de modo que acontecerão $\log_2 N$ iterações, e cada iteração custa $O(N)$, de modo que o algoritmo descrito tem complexidade $O(N \log N)$.

A segunda solução é observar que, se os índices forem deslocados uma unidade para a esquerda, a ordenação descrita no problema é precisamente a ordenação utilizada na transformada rápida de Fourier (FFT). A sequência dos índices ao final da ordenação corresponde à ordenação dos valores destes índices segundo o reverso do padrão de seus k *bits*, onde $N = 2^k$. De fato, o elemento de índice i troca de posição com o elemento de índice j se, e somente se, os k *bits* da representação binária de i e j são mutuamente inversos.

Assim, é possível resolver o problema em $O(N)$.

Problem K. Ki-Sorte!

A solução do problema é simples: basta gerar os números da sorte para cada um dos bilhetes possíveis e registrar o menor valor obtido. Esta solução é linear no número N de bilhetes distintos (segundo o texto do problema, $N = 99$).

Problem L. Luís, o lazer e a louça

Uma técnica muito comum para problemas de otimização é a busca binária. Se conseguirmos criar uma função `bool check(double mid)` que consegue verificar que a resposta é >= mid, conseguimos aplicar busca binária. Focaremos nessa função.

Queremos então verificar se existe um conjunto S com $L \leq \sum_{i \in S} c_i \leq R$ tal que $\frac{\sum_{i \in S} d_i}{\sum_{i \in S} t_i} \geq mid$. Observe que:

$\frac{\sum_{i \in S} d_i}{\sum_{i \in S} t_i} \geq mid \implies \sum_{i \in S} d_i \geq mid * \sum_{i \in S} t_i \implies \sum_{i \in S} d_i - mid * t_i \geq 0$

Agora queremos encontrar um conjunto de índices S com $L \leq \sum_{i \in S} c_i \leq R$ tal que $\sum_{i \in S} d_i - mid * t_i \geq 0$ que é comumente conhecido como Problema da Mochila onde cada item tem peso c_i e valor $d_i - mid * t_i$ e é resolvido em $O(N * R)$.

Com a busca binaria a solução fica $O(N * R * \log)$.

Problem M. Music Player

Basta ler todas as N músicas da entrada, salvá-las em um registro contendo o código e o nome da música, ordená-las pelo código e responder a cada uma das consultas usando busca binária. Um dicionário pode ser utilizado para o mesmo fim.