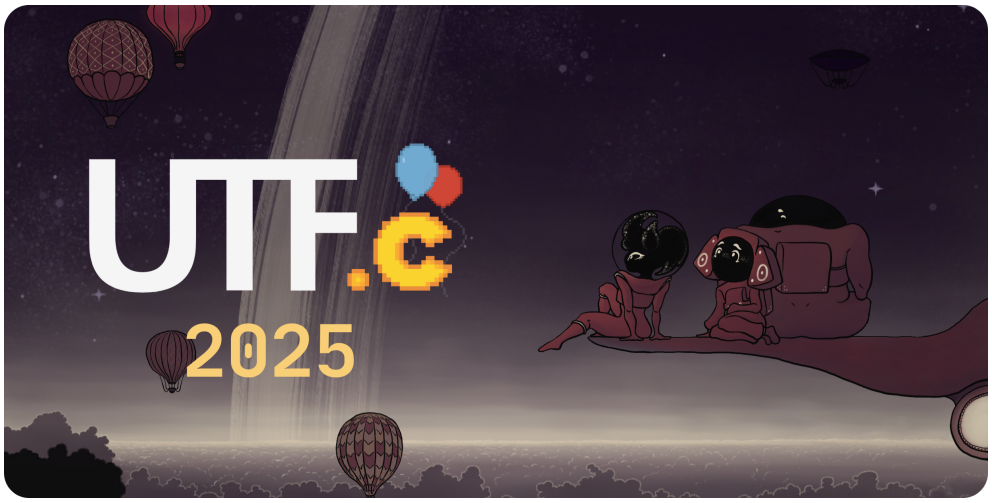

Caderno de Soluções

Problema A

Ao Infinito e Além

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Henrique Mendes

Solução

O problema pode ser resolvido utilizando estruturas condicionais (if/else). Primeiro verificamos se as condições para **HABITAVEL** são satisfeitas ($20 \leq T \leq 30$ e $O \geq 50$). Se não, verificamos se as condições para **ADAPTAVEL** são satisfeitas ($0 \leq T \leq 50$ e $O \geq 30$). Se nenhuma das anteriores for verdadeira, a resposta é **HOSTIL**.

Problema B

Burnout

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Leandro Zatesko

Solução

Vamos considerar os planetas um a um, de P_1 até P_N. Se $N = 1$, podemos viajar direto para P_1, sem precisar esperar, pois é garantido que $D_i \leq L$ para todo i. Suponhamos, então, que $N \geq 2$. Para cada planeta P_i considerado, vamos supor que, para o conjunto de planetas $P_1, \ldots, P_{i-1}$, já temos a solução ótima, i.e. já visitamos todos esses planetas no menor tempo total possível atendendo às restrições do problema. Agora, estando no planeta P_{i-1}, precisamos decidir se vamos esperar um pouco na órbita de P_{i-1}, ou se vamos tocar direto para P_i.

Ora, considerando a janela das últimas 24 horas, se a quantidade de horas AC em que a nave esteve em movimento dentro dessa janela não for maior que $L - D_i$, podemos tocar direto a viagem para P_i, e esta com certeza será a solução ótima para o conjunto de planetas $P_1, \ldots, P_i$. Neste caso, atualizamos o valor da variável AC somando D_i.

Por outro lado, se a quantidade de horas AC em que a nave esteve em movimento dentro dessa janela for maior que $L - D_i$, sabemos que vamos ter que esperar pelo menos $24 - L$ horas na órbita do planeta P_{i-1} para daí iniciarmos a viagem para P_i, pois, quando chegarmos em P_i, o número de horas em viagem na janela das últimas 24 horas terá sido exatamente L, o que significa que não seria possível esperar menos ainda no planeta P_{i-1}.

O único meio pelo qual esta não seria a solução ótima seria se a solução ótima para $P_1, \ldots, P_i$ precisasse usar uma solução não ótima para $P_1, \ldots, P_{i-1}$, i.e. se precisássemos esperar algumas horas em órbita entre P_j e P_{j+1} para algum $j < i - 1$ para compensar de algum modo as $24 - L$ horas que teríamos que esperar entre P_{i-1} e P_i.

Porém, veja que a quantia de horas que esperamos em P_{i-1} não depende do valor de D_i, e que a janela de 24 horas que termina após chegarmos em P_i já está saturada. Então, deslocar horas de espera de entre P_{i-1} e P_i para entre algum P_j e P_{j+1} não faria diferença na duração total da viagem. Isto é, cada hora de espera adicional introduzida entre algum P_j e P_{j+1} não seria capaz de compensar mais de uma hora entre P_{i-1} e P_i. Tudo o que estaríamos fazendo seria deslizar as viagens entre os planetas dentro dessa janela fixa que continuaria terminando no mesmo instante de tempo em P_i.

Note que, após chegar no planeta P_i, como a janela das últimas 24 horas está saturada, cada hora em movimento (da viagem para P_{i+1}) que entrar na janela implica em uma hora em movimento que sai da janela, mantendo a janela saturada. Assim, a partir do planeta P_i, como tivemos que esperar $24 - L$ horas no planeta P_{i-1} e estamos na solução ótima, podemos considerar os planetas a partir de P_i como uma instância nova, reinicializando AC para D_i.

Problema C

Contato Imediato de 2° Grau

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Fernando Monteiro Kiotheka

Solução

Para encontrar os coeficientes da equação quadrática $f(x) = ax^2 + bx + c$ precisamos de três pontos quaisquer da curva. O intervalo entre os pontos não importa desde que ele seja constante, e também não importa a translaçao da curva, você pode posicionar os pontos em qualquer lugar do plano cartesiano.

Peguemos então três pontos v_0, v_1 e v_2 contíguos na entrada. Faremos com que $f(0) = v_0$, $f(1) = v_1$ e $f(2) = v_2$ e descobriremos os coeficientes a, b e c da equação quadrática,

$$f(0) = 0^2a + 0b + c = c = v_0$$
$$f(1) = 1^2a + 1b + c = a + b + c = v_1$$
$$f(2) = 2^2a + 2b + c = 4a + 2b + c = v_2$$

Podemos então subtrair c (que já sabemos que é igual a v_0) de ambas as equações $f(1)$ e $f(2)$,

$$f(1) = a + b = v_1 - c$$
$$f(2) = 4a + 2b = v_2 - c$$

Em seguida podemos subtrair $2f(1)$ de $f(2)$ para resolver o sistema:

$$f(2) = 2a = v_2 - 2v_1 + c$$
$$a = (v_2 - 2v_1 + c)/2$$

Por fim temos b que acabamos de achar,

$$f(1) = a + b = v_1 - c$$
$$b = v_1 - c - a$$

Como o enunciado garante que a parábola é côncava para baixo, ou seja, $a < 0$, o ponto que maximiza é dado por $h = -b/(2a)$, e a resposta do problema é $f(h)$.

Problema D

DJALMA

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Henrique Mendes

Solução

O problema pede para encontrarmos a soma máxima de um subvetor contínuo (Maximum Subarray Sum), onde o "valor" de cada dia é dado por $R_i - C_i$ (receita - custo).

Podemos resolver isso criando um vetor virtual de lucros L, onde $L_i = R_i - C_i$, e aplicando o algoritmo de Kadane neste vetor.

Como os valores de receita e custo podem chegar a 10^9 e N até 10^5, a soma acumulada pode facilmente exceder o limite de um inteiro de 32 bits. Portanto, é essencial utilizar inteiros de 64 bits ("`long long`" em C++) para evitar overflow.

O algoritmo de Kadane funciona mantendo a maior soma que termina na posição atual e a maior soma global encontrada. Se a soma atual cair abaixo de 0, reiniciamos a contagem para 0. Note que o enunciado permite a resposta 0 para "não escolher nenhum dia".

Complexidade: $\mathcal{O}(N)$

Problema E

Este é um Problema Interativo

Tempo limite: 3000 ms	Memória limite: 256 MiB

Autor: Leandro Zatesko

Solução

O problema consiste em determinar se 2 é uma *raiz primitiva* módulo P, isto é, se o conjunto das potências de 2 módulo P é igual ao conjunto $\{1,\ldots,P-1\}$.

Ora, considerando a sequência $2^1, 2^2, 2^3, \ldots$ módulo P, sabemos, do Pequeno Teorema de Fermat, que $2^{P-1} \equiv 1 = 2^0 \pmod{P}$. Logo, essa sequência é cíclica e inclui o elemento neutro da multiplicação. Ainda, para todo elemento 2^i nessa sequência, $2^{i(P-2)}$ é um número tal que, quando multiplicado por 2^i, resulta em $(2^i)^{P-1} \equiv 1 \pmod{P}$. Logo, todo elemento na sequência possui um inverso multiplicativo que também está nessa sequência, o que implica que o conjunto dos elementos nessa sequência é um subgrupo H do grupo multiplicativo dos resíduos não-nulos módulo P. O que queremos, portanto, é saber se a ordem de H é igual a $P-1$ ou se é menor. Porém, se a ordem de H não é igual a $P-1$, temos, do Teorema de Lagrange, que a ordem de H é no máximo $(P-1)/2$, o que significa que para pelo menos metade dos números $m \in \{1,\ldots,P-1\}$, não existe x tal que $2^x \equiv m \pmod{P}$.

Assim, nosso algoritmo probabilístico interativo consiste simplesmente em repetir k vezes o seguinte teste: sorteie $m \in \{1,\ldots,P-1\}$; use o oráculo para saber se existe x tal que $2^x \equiv m \pmod{P}$. Se, para algum m, o Oráculo não encontrar x, podemos devolver com certeza que 2 não é uma raiz primitiva módulo P. No entanto, se após k consultas, todas as respostas do Oráculo tiverem sido positivas, podemos devolver que 2 é uma raiz primitiva módulo P, e nossa resposta estará errada, no caso de 2 não ser uma raiz primitiva módulo P, com probabilidade no máximo $1/2^k$, o que, para $k = 8$, é no máximo $1/256$.

Note que os próprios exemplos já sugerem esta solução, e mesmo um competidor que não possui o conhecimento teórico sobre teoria de grupos poderia intuir essa estratégia.

Há uma pequena dificuldade técnica. A solução probabilística garante o erro limitado por $1/2^k$ se o sorteio é de fato feito com distribuição suficientemente próxima da uniforme sobre todos os números entre 1 e $P-1$. No entanto, veja que P é muito grande. Então, aqueles que tiverem usado apenas `int rand()` estarão amostrando suas queries sem nenhuma garantia da limitação da probabilidade de erro, podendo receber **WRONG ANSWER**. O melhor a se fazer é amostrar de forma independente várias words e concatená-las numa sequência de bits que seja grande o suficiente para o tamanho de P.

Note que este é um problema interativo que não precisa da interação se o competidor conhecer um pouco mais de algoritmos de teoria de números. O problema de determinar se existe x tal que $2^x \equiv m \pmod{P}$, que é resolvido pelo Oráculo, pode ser resolvido pelo próprio competidor em tempo $O(\sqrt{P})$ com, por exemplo, o algoritmo *Baby-step-giant-step*, que é de fácil implementação.

Se o competidor quiser um algoritmo exato para determinar se 2 é raiz primitiva módulo P, pode usar o *Teorema de Lucas*: sendo P um primo ímpar, r é raiz primitiva módulo P se e somente se $r^{(P-1)/q} \not\equiv 1 \pmod{P}$ para todo divisor primo q de $P-1$. Implementar esse teste pode ser feito através da fatoração de $P-1$ em tempo $O(\sqrt{P}\log\log P)$.

Problema F

Fator Gama

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Fernando Monteiro Kiotheka

Solução

O problema se resume a calcular o fator gama dos valores dados. Note que como a velocidade v é dada por Xc, na equação o c^2 se cancela,

$$\gamma = \frac{1}{\sqrt{1-(Xc)^2/c^2}} = \frac{1}{\sqrt{1-X^2c^2/c^2}} = \frac{1}{\sqrt{1-X^2}}.$$

A distância D é em anos-luz, então o tempo de coordenada em anos é dado por $t_c = D/X$ pois na análise dimensional, o c se cancela em $D/v = D/(Xc)$.

Por fim, basta ajustar o tempo de coordenada t_c com o fator γ resultando em tempo próprio do relógio que ficou em movimento $t_p = \gamma t_c$. A resposta é dada por por $t_c - t_p$.

Problema G

Guia do Lixeiro das Galáxias

Tempo limite: 3000 ms	Memória limite: 1024 MiB

Autor: Tália de Aguiar

Solução

Uma forma de simularmos as reações em cadeia é implementarmos funções `visita_linha` e `visita_coluna`. `visita_linha` marca uma linha como visitada e então a percorre, contando os satélites e chamando `visita_coluna` para as colunas não visitadas que os contêm. `visita_coluna` age analogamente. Não revisitamos fileiras pois as cadeias podem ser cíclicas.

Cada simulação dessa é $\mathcal{O}(NM)$, de modo que simular para todas as possibilidades iniciais seria $\mathcal{O}((N+M)NM)$. Para reduzir a complexidade, notemos que as fileiras alcançadas por uma cadeia dada são as mesmas independente de qual escolhemos como inicial. Podemos, assim, evitar redundância simulando somente para fileiras não visitadas por simulações anteriores. Embora cada simulação ainda seja $\mathcal{O}(NM)$, agora o total fica restrito a também $\mathcal{O}(NM)$, pois visitamos cada fileira uma única vez.

Podemos expressar a mesma solução na linguagem de grafos. Imaginamos as fileiras como $N+M$ vértices e ligamos os vértices que representam a i-ésima linha e a j-ésima coluna se e somente se $A_{i,j}$ tem um satélite. Procuramos então a componente conexa com mais arestas via DFS.

Problema H

Hora das Estrelas

Tempo limite: 1000 ms	Memória limite: 1024 MiB

Autor: Alan Tara

Solução

Para solucionar esse problema basta ordenar, por tempo, os eventos de supernovas e simular as restrições de tempo dos ângulos de cada evento em um vetor de 360 posições, salvando para cada tempo diferente uma posição válida.

Mais especificamente, possuindo o vetor de supernovas ordenadas por tempo, podemos iterar para cada tempo diferente, calcular o intervalo de ângulos que as supernovas nesse tempo restringe utilizando as informações de posição e raio e geometria que será explicada abaixo, e para cada ângulo dentro desse intervalo salvar o tempo máximo até que o ângulo esteja restrito. Por fim, basta salvar algum ângulo na iteração que esteja disponível para o tempo e caso não exita, a resposta é -1.

Para o cálculo do intervalo de ângulos, podemos calcular o ângulo base entre a nave n e a supernova s por arcotangente de $(y_s - y_n)/(x_s - x_n)$ e podemos calcular o ângulo até a tangente do círculo utilizando arcoseno de $r/(dist(n,s))$ em que $dist()$ é a função para distância euclidiana. Basta então fazer $base_\theta - tang_\theta$ e $base_\theta + tang_\theta$, e cuidar de resultados maiores que 360, menores que 0 ou inversão no intervalo devido à essa aritmética modular.

A complexidade desse algoritmo é $NlogN$ pela ordenação + $360N$ pela simulação, ou seja $O(NlogN)$.

Problema I

Imprudência Artificial

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Thaís Say

Solução

Problema J

Jamanta e o Coffee Break

Tempo limite: 1000 ms	Memória limite: 256 MiB

Autor: Henrique Mendes

Solução

A ideia intuitiva para resolver esse problema consiste em adotar uma estratégia gananciosa: consideramos inicialmente a refinaria de menor custo por litro (independende de seu limite) e usamos o máximo possível dela para tentar atingir os T litros desejados. Caso esse valor ainda não seja atingido, consideramos a refinaria de segundo menor custo por litro e assim por diante, até atingirmos o valor exato de T litros.

Mais formalmente, podemos criar um vetor de pares R, em que $R_i = (C_i, L_i)$, e utilizar o seguinte critério de ordenação:

$$R_i < R_j \iff C_i < C_j$$

Desse modo, podemos ordenar R, seguindo esse critério de ordenação, em $\mathcal{O}(n \log(n))$. Agora, com esse vetor já ordenado, o percorremos da esquerda para direita. Nessa situação, considere que estamos na i-ésima posição, e que a quantidade de litros que queremos atingir, nesse momento, seja T'. Há duas possibilidades:

- Se $L_i \leq T'$, então precisamos utilizar todo o limite desta refinaria, a um custo de $L_i \cdot C_i$, atualizamos $T' \leftarrow T' - L_i$ e seguimos.
- Se $L_i > T'$, então utilizamos apenas T' litros desta refinaria, a um custo de $T' \cdot C_i$, e finalizamos o algoritmo.

Inicializando $T' \leftarrow T$, esse algoritmo permite encontrar a resposta ótima deste problema.

Demonstração:

Embora não essencial para resolver problema durante o contest, precisamos provar que de fato essa estratégia gananciosa encontra uma solução ótima.

(TODO)