Análise dos problemas da III Maratona de Programação do Norte
Gabriel Duarte

Autores da Prova: *Dâmi Henrique - INATEL, Diego Rangel Piranga Costa - FACIT, Felipe Ochial - URI, Francisco Elio Parente Arcos Filho - UEA, Gabriel Duarte - UFF*

Problema A - Cobra Norato
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Difícil
Tema(s): Programação Dinâmica

Com a palavra o autor do problema:

Podemos interpretar o problema como um grafo com N camadas de vértices, uma para cada Ti. Em cada camada de Ti, temos um vértice para cada valor 0, Ti, 2Ti, 3Ti, 4Ti, … e uma aresta direcionada de cada valor para o próximo maior na mesma camada. Com exceção da primeira camada (a mais acima), também temos uma aresta direcionada de cada valor para o mesmo valor na camada anterior.

Modelado assim o problema se transforma em achar o número de formas de ir do vértice 0 na n-ésima camada - camadas são 1-indexadas - ao vértice C na primeira camada já que se mover horizontalmente na camada de Ti significa usar Ti uma vez na soma. Como só podemos nos mover horizontalmente e pra cima, ordenando os valores de Ti contaremos cada possível caminho no grafo, ou seja, cada jeito de fazer a soma, uma vez.

Agora vamos usar programação dinâmica para achar a quantidade de formas de chegar no valor i na j-ésima camada. Para fazer a transição do valor i para o valor i+1, precisamos considerar o movimento horizontal, o qual não muda a camada, e também o movimento vertical para cima, o qual implica que temos que somar número de jeitos de chegar na camada j ao número de formas de atingir cada camada acima, assumindo que a camada j tem o vértice i+1. Se escrevermos todas as respostas para algum i como um vetor, transições de i para i+1 corresponde a multiplicar esse vetor por uma matriz, e há N diferentes matrizes em jogo, dependendo somente do maior valor de Ti que divide i+1. Feito tudo isso é só acelerar as computações com exponenciação rápida de matrizes.

Complexidade O(N*N*N*log(N*C))

Problema B - Quanta Mandioca?
Autor: *Felipe Ochial*
Dificuldade: Fácil
Tema(s): Adhoc

Provavelmente o problema mais simples da prova, basta ler 5 valores e fazer a conta conforme explicado no enunciado.

Complexidade: O(1)

Problema C - Jaçanã
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Fácil
Tema(s): Adhoc, Simulação

Basta simular o processo descrito no texto e ir pulando até chegar ao final ou não ser mais possível.

Complexidade: O(N)

Problema D - Casais
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Médio
Tema(s): Programação Dinâmica, Combinatória

Neste problema podemos enxergar cada casal como uma peça de dominó tradicional (2x1) e temos um grid de tamanho 2x**N** e **N** peças, queremos saber de quantas formas distintas podemos preencher tal grid. A primeira observação que podemos ter é que dada uma configuração válida de distribuição das peças, sempre podemos inverter uma peça, assim gerando uma nova possibilidade, com isso, já temos 2^N formas, além disso, podemos também trocar peças de posição, ou seja, fazer um embaralhamento entre elas, como são **N** peças, temos $N!$ formas, já nos gera $N! * 2^N$, possibilidades para cada forma de preencher o grid, precisamos agora só saber quantas maneiras existem de preencher.

Para contar as maneiras de inverter podemos recorrer a programação dinâmica, é fácil ver que se o grid em questão tem largura 1, só existe uma possibilidade, colocar a peça em pé, e se tem tamanho 2, existem duas possibilidades, colocar as duas deitadas ou as duas em pé, tendo esses casos base conseguimos pensar em uma $dp[i]$ que representa quantas formas distintas temos de preencher um grid de tamanho 2xi. Sua recorrência fica da seguinte forma:

$dp[i] = 0, \ se \ i < 0$

$dp[i] = i, \ se \ i \leq 2$

$dp[i] = dp[i-1] + dp[i-2], \ se \ i > 2$

Com essas informações vemos que a resposta do problema é $dp[n] * N! * 2^N$, lembrando que devemos tirar módulo em todas as operações que fizermos.

Complexidade: O(N)

Problema E - Máquina do Tempo Quebrada
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Difícil
Tema(s): Matemática

Com a palavra o autor do problema:

O problema pode ser resumido em somar alternadamente os elementos do conjunto A com o negativo dos elementos do conjunto B para produzir a soma (T-S). Atenção para o fato de que a soma sempre começa com um elemento de A.

Seja X = (T-S), Ai um elemento de A e Bi um elemento de B; Indo direto ao ponto: a resposta é ‘S’ quando X é divisível por M ou quando X deixa o mesmo módulo por M que um elemento de A. Caso contrário é ‘N’. Sendo M o mdc dos elementos de C = {Ck = Ai - Bj, para todo i e j}.

Agora vamos provar isso.

As duas condições são respectivas de dividir o problema em dois casos:

1. Os números X que são formados pela soma de K elementos de A com K negativo dos elementos de B.
2. Os números X que são formados pela soma de (K+1) elementos de A com K negativo dos elementos de B.

Observe que o primeiro caso é o caso dos Xs formados por um número par de viagens, ou seja, somando só elementos de C. É óbvio que toda soma de elementos de C é divisível por M, já que M divide cada elemento dessa soma. Agora para provar o reverso, que todo número divisível por M pode ser soma de elementos de C, só precisamos garantir que seja possível formar M como soma de elementos de C. Porque provado isso, podemos fazer qualquer múltiplo de M só repetindo a soma de Cis que usamos para fazê-lo! (provando a primeira condição da resposta)

O que queremos provar então é que podemos escrever M = Q1*C1 + Q2*C2 + Q3*C3 + … com Qi inteiro. O problema é que para essa equação ser verdadeira pode ser que Qi precise ser negativo, o que implicaria em usar -Qi vezes um elemento de C. Como fazer isso? Seja Cj esse elemento, para produzir -Cj:

- Sendo Cj positivo: basta que exista ao menos um número negativo em C, -N por exemplo, e então podemos fazer -Cj = (N-1)*Cj + Cj *(-N), ou seja, usar (N-1) vezes Cj e Cj vezes -N
- Sendo Cj negativo: basta que exista a condição análoga: ao menos um positivo

Então M pode ser produzido como soma de elementos de C(logo todos os múltiplos de M) se houver ao menos um número positivo e ao menos um número negativo em C. O que é garantido pela seguinte parte do enunciado:

“*Apesar de possuir dois painéis, a máquina foi projetada inicialmente para possuir apenas um único painel. Onde todos os botões estariam ordenados de forma não-decrescente da esquerda para direita. Mas essa idéia foi logo descartada visto que os botões de ao menos um dos dois tipos (A e B) nunca ficavam todos juntos, formando assim um padrão estético não muito agradável.*”

Traduzindo matematicamente: min(A) < max(B) e min(B) < max(A).

Até aqui provamos um modo de checar se um ano é atingível com um número par de viagens no tempo (Se ele é divisível por M).

Para saber os anos atingíveis com número ímpar de viagens, podemos pensar neles como sendo um ano que é atingível por um número par de viagens mais um elemento de A.

É notável que esses anos deixam o mesmo módulo por M que os elementos de A. O que talvez não seja tão óbvio é que todos os elementos de A deixam o mesmo módulo por M!

Isso acontece porque M divide a diferença de quaisquer dois elementos de A (implicando que todos Ai tem mesmo módulo por M): Observe que que M divide Ai-Bk e Aj-Bk para i e j quaisquer e k fixo já que essas duas diferenças são elementos de C, logo

também divide a diferença deles (Ai-Bk)-(Aj-Bk) = Ai-Aj que representa todas as possíveis diferenças de elementos de A!

Pronto, provamos a segunda condição, a dos anos atingíveis com número de viagens ímpares: Quando X deixa mesmo módulo por M que qualquer elemento de A.

Essa jornada podia estar terminada se não fosse a impossibilidade de calcular M só seguindo a definição de que ele é o mdc de todas as diferenças possíveis de elementos de A com elementos de B. Porque fazer isso implicaria numa lógica quadrática o que os limites da questão não permitem. Mas podemos espremer nosso cérebro para mais um pouco de matemática e calcular o mdc de C sem efetivamente produzir todos os elementos de C!

Como já sabemos que M divide as diferenças de elementos de A e as diferenças de elementos de B (pelo mesmo princípio) podemos criar cada elemento de C como soma de três números divisíveis por M: Ai-Bj = (Ai-A0) + (A0-B0) + (B0-Bj) para todo i e j.

Seja D { Dk = (Ai-A0) para todo i, ou Dk = (B0-Bj) para todo j, ou Dk = (A0-B0)} podemos calcular o mdc de D complexidade de tempo linear. Depois basta observar que o mdc de D é o próprio M! Isso acontece porque o mdc de D divide M já que também divide cada um dos três números que soma cada elemento de C e M divide o mdc de D porque divide as diferenças dos elementos de C, ou seja, os números que compõe D!

Sumarizando tudo: Calculamos M calculando o mdc de D e depois testando as condições em cada diferença (T-S).

Complexidade: O(N+M+Q)

Problema F - Fibra Ótica
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Fácil/Médio
Tema(s): Grafos, Árvore Geradora Mínima

A primeira coisa a se notar é que o problema quer no final das contas saber a árvore geradora mínima, podendo adicionar arestas ou remover algumas. Uma observação importante é que é sempre ótimo manter o máximo de arestas que já estão no grafo, assim não “pagamos” nada, mas se tivermos que remover algumas delas, também é ótimo remover as que têm menor custo, minimizando nosso custo total. Além disso, se tivermos que adicionar arestas vamos sempre querer adicionar as que têm menor custo. Com isso, modelamos o grafo da seguinte forma:

Toda aresta que já está no grafo de custo X colocamos com custo -X e arestas não presentes colocamos com o seu valor normal. Agora basta ordenar todas as arestas e percorrer uma a uma. Se estamos em uma aresta *uv* com custo *w,* se *uv* não gera ciclo no grafo, adicionamos ela e se $w > 0$ somamos *w* na resposta. Caso *uv* gere ciclo, não iremos adicionar, porém se *uv* já estava no grafo, ela é uma aresta de remoção, então somando *-w* na resposta.

Complexidade: O(ElogE)

Problema G - Mistura de Bits
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Médio
Tema(s): Adhoc, Contagem de inversões, Bitwise

A primeira parte do problema é pensar sobre como funciona a operação descrita, o que podemos notar é que o xor é uma operação que se cancela se aplicado diversas vezes, $a \; xor \; b = \; c \;\; \Rightarrow c \; xor \; b \; = a$, sendo assim, temos uma quantidade limitada de possibilidades de números que podem ser geradas no *array*. Além disso, conseguimos ver que se aplicado as operações em posições consecutivas, estamos "empurrando" um valor para frente. Levando essas observações em consideração, conseguimos enxergar que o xor de elementos adjacentes no *array* A, tem que existir no *array* B. Criaremos dois array *xA e xB,* onde

$$xA[i] = \; arrayOriginal[i] \; xor \; arrayOriginal[i+1] \; e \; xB[i] = \; arrayFinal[i] \; xor \; arrayFinal[i+1]$$

Com isso, sabemos que o *xB* tem que ser uma permutação de *xA*, caso não seja é um caso impossível. Caso contrário, precisamos saber a quantidade de operações, para isso é fácil ver que a resposta é igual a quantas trocas entre pares adjacentes são necessárias fazer em *xB* para ele ficar igual a *xA*. Para implementar isso, é possível fazer um mapeamento de todos os valores de *xB* para qual posição ele deve ir, devemos tomar cuidado quando se tem valores repetidos, mas quando isso acontecer podemos usar uma estratégia gulosa para escolher tal posição. Ao final, esse mapeamento nos dará um novo *array* e com ele caímos em um problema clássico que é a contagem de inversões, que pode ser feito com Merge-sort ou com uma BIT, por exemplo.

A complexidade total é $O(nlogn)$.

Problema H - Smider Pan
Autor: *Dâmi Henrique*
Dificuldade: Fácil/Médio
Tema(s): Programação Dinâmica, Maior Subsequência Crescente (LIS)

Podemos ver que queremos saber qual o maior triângulo que é possível formar, onde ele irá crescer até um ponto e depois diminuir. Podemos dividir o problema em dois:

1 - A parte esquerda do triângulo, ou seja, o lado crescente, para isso podemos achar para todas as posições, qual a maior sequência crescente que podemos formar olhando para um *array* que termina nesta posição, ficando assim:

$$dp[i] \; = \; 1, \; se \; i \; = 0$$
$$dp[i] \; = \; 1 + max_{\,j<i}(dp[j]), \; se \; i \; \neq 0 \; e \; altura[i] \; > altura[j]$$

2 - A parte direita do triângulo, que é análoga a descrita acima, mas aqui queremos montar um vetor que diz para cada posição qual a maior subsequência decrescente, o que nada mais é que o mesmo algoritmo do tipo 1, só que começando agora pela direita:

$$dp2[i] \; = \; 1, \; se \; i \; = n-1$$
$$dp2[i] \; = \; 1 + max_{\,j>i}(dp2[j]), \; se \; i \; \neq n-1 \; e \; altura[i] \; > altura[j]$$

Com esses dois *arrays* em mãos, *dp* e *dp2,* já conseguimos achar a resposta, pois nada mais é que uma posição *i*, onde a soma de $dp[i] \; + \; dp2[i]$ seja máxima.

Complexidade: O(n^2) ou O(n*logn) se utilizar uma estrutura de dados para fazer as consultas de máximo em tempo logarítmico.

Problema I - Emergência em manaus
Autor: *Diego Rangel Piranga Costa*
Dificuldade: Médio
Tema(s): Grafos, componentes fortemente conexas, busca em profundidade

A primeira coisa que podemos fazer é eliminar do grafo todos os vértices que já são alcançáveis através de *s*, para isso basta uma *dfs* partindo de *s* e ir marcando todos os vértices como visitados. Após isso, podemos ver que o grafo resultante terá apenas algumas componentes fortemente conexas e agora devemos apenas escolher de forma ótima em quais iremos colocar arestas de *s* até elas. Vamos supor que exista uma componente A que liga em uma outra componente B, é fácil ver que o ótimo é colocar uma aresta de *s* para qualquer um dos vértices de A, pois toda componente A e B será alcançada. Como não importa em qual vértice de uma componente iremos colocar a aresta, podemos comprimir o grafo, transformando todas as componentes fortemente conexas em um único vértice. Tendo esse novo grafo em mãos conseguimos ver que o ótimo é colocar aresta em vértices que têm grau de entrada igual a 0. Portanto, a resposta do problema é quantos vértices possuem grau de entrada igual a zero nesse novo grafo.

Complexidade: O(V+E)

Problema J - Monitor
Autor: *Diego Rangel Piranga Costa*
Dificuldade: Médio
Tema(s): Estruturas de dados, Componentes conexas, BIT

Se não existisse a condição número 2 descrita no problema, esse seria apenas mais um problema clássico de consulta onde queremos saber quantos valores maiores que X existem e realizar alterações de alguns valores. Para resolver, bastaria utilizar uma BIT, por exemplo, deixando a complexidade em O(logn) por consulta.

Para tratarmos a condição 2, não é difícil, podemos montar um grafo com as relações de amizade, neste grafo terá algumas componentes conexas, conseguimos ver que cada consulta agora só pode ser aplicada em uma única componente. Assim, seja *w* o número de componentes, iremos criar *w* BIT's e quando formos realizar uma consulta, basta verificar em qual componente o elemento em questão está e fazer na BIT correspondente.

Complexidade: O(Qlogn)

Problema K - Divisibilidade em binário
Autor: Gabriel Duarte
Dificuldade: Fácil
Tema(s): Matemática, Conversão de base

Sabemos que qualquer número *n* pode ser descrito como sendo $n = m * d + r$, onde *m* é a parte inteira da divisão de *n* por *d*, e *r* é o resto desta divisão. Então conseguimos ver que se tivermos um número qualquer podemos ver se ele é divisível por *d* olhando dígito a dígito. Suponhamos que temos o número 245 na base 10 e queremos testar a divisibilidade dele por 11, uma primeira coisa a se notar é que 245 = 2*100 + 4*10 + 5*1, então para testar, a gente não precisa olhar o número por inteiro podemos ir em etapas, pois como já

sabemos $(a * b)\%c = (a\%c * b\%c)\%c$ e $(a + b)\%c = (a\%c + b\%c)\%c$ para 245 ser divisível por 11, $(200\%11 + 40\%11 + 5\%11)\%11$ tem que ser igual a zero (Não ter resto da divisão). Com isso bolamos um algoritmo genérico para qualquer valor de *d:*

1. $r = 0$
2. $n = 245$
3. $m = 11$
4. $Para\ 0 \leq i \leq tamanho(n)$:
5. $r = (r * 10 + n[i])\%m$ // $n[i]$ é *igual ao* $i-$ésimo *dígito de* n
6. $Se\ r == 0$ *então* n é *divisível por* m

Isso que fizemos é para um número na base 10. Porém, podemos generalizar para qualquer base, basta a gente ir fazendo uma "conversão" de base e salvando apenas os módulos, evitando *overflow*, assim para o problema em questão, podemos apenas alterar a linha 5 para:

$r = (r * 2 + n[i])\%m$ // $n[i] = valor\ do\ dígito,\ 0\ ou\ 1$

Portanto basta executarmos esse algoritmo para todos os valores da lista.

Complexidade: O(|N|*M)

Problema L - Gabarito
Autor: Francisco Elio Parente Arcos Filho
Dificuldade: Fácil
Tema(s): Guloso

Podemos ver que é sempre ótimo escolher para cada questão a alternativa que mais aparece e que seja diferente da que Desafortunado escolheu. Então para cada coluna a gente pega a letra que mais aparece e soma na resposta, como são 26 letras no máximo podemos criar um array de tamanho 26 para cada coluna.

Complexidade: $O(K * N * 26)$

Problema M - Plantação de Açaí
Autor: *Diego Rangel Piranga Costa*
Dificuldade: Médio/Difícil
Tema(s): Geometria, Círculos

Este é um problema clássico de geometria, o *Smallest-circle problem*, onde temos diversos pontos e queremos encontrar o menor círculo possível que engloba todos eles. Como o limite do problema está bem alto, temos que ter uma solução inteligente. Existem algoritmos bem conhecidos na literatura que resolvem esse problema em tempo linear e é isso que precisamos aqui.

Complexidade: O(N)