Seletiva para Maratona de Programação de 2016

Comentários sobre os problemas

Patrocínio:

Departamento de Ciência da Computação IME-USP

Problema A: Renzo e o artefato perdido

Autor do problema: Marcos Kawakami
Análise: Marcos Kawakami

Seja $F(X)$ a função que leva um ponto X do mapa maior em seu respectivo ponto do mapa menor. Seja X^* um dos pontos procurados (ou seja, $F(X^*) = X^*$) e X_0 um ponto arbitrário do mapa maior. Note que

$$q \cdot d(X_0, X^*) = d(F(X_0), F(X^*)) = d(F(X_0), X^*),$$

onde $d(A, B)$ denota a distância euclidiana entre os pontos A e B, e q é a razão entre as escalas do mapa menor e maior.

Considere então a sequência $\{X_n\}$ definida por $X_n = F(X_{n-1})$. Temos que

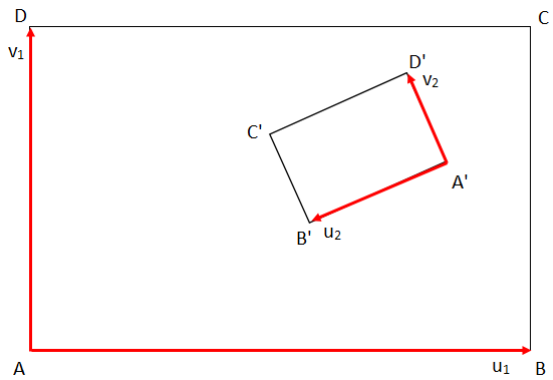
$$\begin{aligned} d(X_n, X^*) &= d(F(X_{n-1}), X^*) = q \cdot d(X_{n-1}, X^*) = q \cdot d(F(X_{n-2}, X^*) \\ &= q^2 \cdot d(X_{n-2}, X^*) = \ldots = q^n \cdot d(X_0, X^*). \end{aligned}$$

Como $q < 1$, temos que $\lim_{n\to\infty} d(X_n, X^*) = 0$ e, portanto, a sequência converge para o ponto fixo (note que isto mostra que o ponto fixo, se existir, é único).

Assim, uma possível solução é escolher um ponto X_0 qualquer do mapa maior e aplicar F iteradamente sobre este ponto até que fique suficientemente próximo do ponto procurado.

A função F pode ser encontrada de várias maneiras. Uma delas é tomar os vetores u_2 e v_2 como indicados na figura abaixo e definir

$$F(x, y) = A' + \frac{x}{W}\vec{u_2} + \frac{y}{H}\vec{v_2}.$$



Outra solução é resolver o sistema linear $(x, y) = F(x, y)$, que sabemos que é possível e determinado, uma vez que a solução existe e é única.

Problema B: Búfalos

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

Esse problema pode ser reduzido à "encontre o número de permutações de tamanho N com LIS (Subsequência Crescente Máxima) não 3 ou 4", pois as filas são permutações, e uma possível família é uma subsequência decrescente (que é análoga a uma crescente).

Um algoritmo bem conhecido para achar a LIS de uma sequência é determinar, para cada i, o menor número que pode ser o i-ésimo número em uma subsequência crescente. Perceba que i não excede o tamanho da LIS. Aqui a ideia é similar, mas teremos esses números como estado em uma programação dinâmica. Esta será usada para contar o número de permutações de LIS 3 ou 4, e a resposta é subtrair esse resultado de $N!$.

A pd construirá a permutação número por número. Seja B_i o menor número selecionado que pode ser o i-ésimo número de uma subsequência crescente, não guardaremos B_i explicitamente, mas algo similar. Considere `solve(a1, a2, a3, a4)`, onde `a1` é a quantidade de números *não selecionados* que são menores que B_1, `a2` a quantidade de números não selecionados entre B_1 e B_2, e `a3` e `a4` definidos analogamente. Perceba que `a5` não é necessário pois não calcularemos IS's de tamanho maior que 4.

Em cada passo, devemos escolher um número ainda não selecionado, e adicioná-lo à permutação. Se escolhermos um número menor que B_1 (existem `a1` possibilidades), o valor B_1 mudará, e os tamanhos `a1` e `a2` devem ser ajustados para o próximo passo da pd, de acordo com qual número foi escolhido. O mesmo vale para os `a2` números no intervalo $[B_1, B_2]$ e os números em $[B_2, B_3]$. Não podemos escolher qualquer número no intervalo $[B_3, B_4]$, apenas o maior, caso contrário o maior vai ser escolhido em alguma iteração e eventualmente teremos uma permutação de LIS 5. (Por isso `a5` não é necessário)

Pode ser necessário carrega um valor booleano na pd para evitar contar as permutações com LIS 1 e 2.

Precisamos testar `a1 + a2 + a3 + 1` números e temos que `a1 + a2 + a3 + a4` $\leq N$, então a complexidade é $\mathcal{O}(N^5)$, com uma constante bem pequena (aproximadamente $\frac{1}{12}$).

Problema C: As ruínas de Cahokia

Autor do problema: Renzo Gonzalo Gómez Diaz
Análise: Renzo Gonzalo Gómez Diaz

Sejam L e R vetores de inteiros que representam a largura das paredes oeste e leste, respectivamente. Note que, se fixamos uma linha i, o tempo que leva as paredes colidirem nessa linha é $\frac{1}{2}(W - L[i] - R[i])$. Logo, o que precisamos encontrar é

$$\frac{1}{2}\min\{W - L[i] - R[i] : 1 \leq i \leq H\}.$$

Claramente, podemos achar essa resposta em $\mathcal{O}(H)$.

Problema D: Joias douradas de Black Hills

Autor do problema: Marcos Kawakami
Análise: Marcos Kawakami

No pior caso, as $K-1$ pessoas à frente de Nay na fila comprarão as $K-1$ combinações mais baratas de joias, forçando Nay a gastar ao menos o preço da K-ésima menor combinação. Queremos saber, então, qual o valor do K-ésimo par de joias mais barato.

Seja $Q(c)$ a quantidade de combinações de joias que custam c ou menos. Estamos interessados em encontrar o menor c^* tal que $Q(c^*) \geq K$. Como Q é não-decrescente, c^* pode ser encontrado usando busca binária.

Resta então saber como calcular $Q(c)$ para todo c. Para isto, considere o vetor

$$v[] = \{j_0, j_1, \cdots, j_{N-1}\}$$

dos preços das categorias de joias, com $j_0 \leq j_1 \leq \cdots \leq j_{N-1}$. Contaremos a quantidade de pares (i, j) tais que $v[i] + v[j] \leq c$, com $i < j$.

Isto pode ser feito utilizando N buscas binárias, encontrando, para todo i, o maior $j > i$ tal que $v[i] + v[j] \leq c$. Assim, o cálculo de Q é feito em $\mathcal{O}(NlogN)$.

Alternativamente, podemos calcular o j máximo para cada i em tempo constante amortizado, observando que o j máximo de $i+1$ não excede o j máximo de i. Assim, podemos calcular os valores de j em ordem crescente de i, e Q passa a ser calculado em $\mathcal{O}(N)$.

Problema E: Ofensa Gratuita

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

Para resolver esse problema, deve-se usar o algoritmo Aho-Corasick (existem muitas boas referências na internet e livros), uma trie para um conjunto de strings que tem links de falha, um link que aponta, para um nó v, para o maior sufixo próprio de v que é prefixo de alguma string do conjunto, assumindo que o nome de um vértice é seu prefixo associado. Essa estrutura pode ser preprocessada para que, de qualquer vértice u, dada uma letra c, seja possível em tempo $\mathcal{O}(1)$ determinar o vértice v que é o maior sufixo da string uc e está representado na trie. (isso é, é sufixo de alguma das strings do conjunto).

Um caminho $u_1, \ldots, u_k$ nesse digrafo (não é mais uma árvore) representa uma string $c_1c_2 \ldots c_k$, e no passo g o nó u_g é o maior prefixo de alguma string do conjunto que é sufixo de $c_1c_2 \ldots c_g$.

Criaremos um digrafo dessa forma para o conjunto de todas as $N + M$ strings. O problema pede o caminho de menor tamanho tal que todas as N palavras horríveis ocorrem e nenhuma das M palavras gentis. Isso será feito usando uma BFS. Como N é muito pequeno, usaremos uma bitmask para identificar quais strings horríveis já foram criadas. Para cada nó u precalcularemos o valor b_i, a bitmask do conjunto de todas as palavras horríveis que são sufixos de u (isso pode ser feito usando os links de falha).

O estado da BFS é (u, b), onde u é o nó do digrafo e b a bitmask das strings já completadas. Quando chegamos em um nó v devemos atualizar o valor de b fazendo um OR bit a bit com b_v. Marcamos os nós u tais que u tem uma string gentil como sufixo, e nunca visitamos esses nós (garantindo assim que não teremos strings gentis como substring), isso acontece se algum nó de uma string gentil pode ser alcançado usando links de falha. A BFS determinará o caminho mínimo de $(\texttt{raiz}, 0)$ para $(\texttt{qualquer nó}, 2^N - 1)$, pois $2^N - 1$ é a bitmask completa, significando que criarmos todas strings horríveis.

A complexidade fica $\mathcal{O}(2^N \cdot \sum_{i=1}^{N+M} |S_i|)$, onde S_i é a i-ésima string da entrada. Essa soma é garantida não ser no máximo 300.

Problema F: Detector de metais

Autor do problema: Marcio T. I. Oshiro
Análise: Marcio T. I. Oshiro

O problema pede para simular uma fila com N elementos tal que ao remover um elemento, o elemento seguinte é movido para o final da fila. Simular essa fila gasta tempo e espaço lineares, o que seria bom se não fosse o fato de N poder assumir valores muito grandes, como 10^9. Logo, é preciso uma solução mais eficiente.

Considere que inicialmente a fila consiste na sequência de números de 1 a N. Assim, o i-ésimo elemento da fila tem valor i. Note que nessa fila os números ímpares sempre serão removidos antes dos pares e serão removidos na ordem. Então, se i é ímpar, ele será o $\lceil \frac{i}{2} \rceil$-ésimo a ser removido.

Agora, suponha que i é par. Antes da $\lceil \frac{N}{2} \rceil$-ésima remoção temos a seguinte configuração:

$$[N-1,\ N,\ 2,\ 4,\ 6,\ \ldots], \text{ se } N \text{ é par;}$$
$$[N,\ 2,\ 4,\ 6,\ \ldots], \text{ se } N \text{ é ímpar.}$$

Se N é par, após remover todos os números ímpares sobram os números pares em ordem crescente. Logo, o i-ésimo número na fila inicial é o $\frac{i}{2}$-ésimo na fila após a remoção de todos os números ímpares. Ou seja, a resposta para a instância (N, i) do problema é $\lceil \frac{N}{2} \rceil$ mais a resposta para a instância $(\lfloor \frac{N}{2} \rfloor, \frac{i}{2})$.

Se N é ímpar, a ideia é quase a mesma. A diferença é que a sequência de números pares só não está ordenada pelo fato do 2 aparecer no final. Logo, basta ajustar o valor de i adequadamente na chamada recursiva.

Seja $T(N, i)$ o tempo gasto pelo algoritmo recursivo acima para resolver a instância (N, i). Para i ímpar, temos $T(N, i) = \mathcal{O}(1)$. Para o caso de i ser par, temos que $T(N, i) = T(\lfloor \frac{N}{2} \rfloor, i') + \mathcal{O}(1)$, onde $1 \leq i' \leq \lfloor \frac{N}{2} \rfloor$. Observe que a recorrência sempre chega em $N = 1$, que é ímpar. Resolvendo a recorrência, concluímos que no pior caso $T(N, i) = \mathcal{O}(\log N)$.

Problema G: A Declaração da Independência

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

O que o problema pede é para implementar uma fila persistente, isso é, uma fila que pode ser mudada para uma versão anterior.

Existem duas formas de fazer isso, uma *online* e outra mais rápida e *offline*.

Se pudermos responder as queries de forma offline, como podíamos nesse problema, podemos criar uma árvore de versões, isso é, criamos um nó para cada versão, e se aplicarmos a operação i na versão v, criamos um novo nó i e conectamos este ao nó v, deixando na aresta com a informação dessa operação.

Depois disso, podemos processar todas as respostas com uma DFS, cada aresta "descendo" (para longe da raiz) significa que a operação daquela aresta deve ser feita, e quando subimos essa aresta, desfazemos essa operação. Dessa forma, quando examinamos o nó v, só as operações que estão no caminho da raiz até v estão realizadas, e isso é exatamente a versão v da fila.

É necessário usar uma `deque` para conseguir desfazer operações da fila. Fazendo essa DFS, sempre precisaremos desfazer apenas a última operação que fizemos, como em uma pilha. Essa solução é offline e tem complexidade $\mathcal{O}(Q)$.

A solução online é mais complicada. Também queremos criar uma árvore, mas não é a mesma árvore que na solução anterior, essa árvore conterá todos os valores adicionados na fila até agora. Para cada operação i guardaremos dois nós s_i e e_i, o começo e o fim da fila na versão i, e podemos mudar para a versão correta da fila em cada operação.

Se a operação i é para adicionar x na versão v da fila, apenas criamos um novo nó e conectamos esse à e_v, e atualizamos $s_i = s_v$ e $e_i =$ `novo nó`. Se a operação i é para remover o primeiro elemento da fila de versão v, precisamos mover s_v um passo mais perto de e_v. Mas como estamos em uma árvore e não um caminho isso não é tão simples. Uma solução é usar tabelas esparsas para cada nó, e assim pular de e_v para o próximo nó depois de s_v no caminho para e_v, similarmente a uma operação de Level Ancestor.

Essa solução é online e tem complexidade $\mathcal{O}(Q \cdot \log(Q))$.

Problema H: Divas do Pop

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

A primeira observação é que a raiz quadrada na média glauberiana não faz diferença, e podemos consideram apenas o produto dos números. Como os limites são pequenos, podemos calcular a média de todos os $2^N - 1$ álbuns possíveis para Taylor e todos $2^M - 1$ valores para Katy, e usando alguma estrutura (`set` por exemplo) encontrar rapidamente para cada álbum de Katy se Taylor tem um álbum com mesma média.

O problema é que o produto de todas as músicas pode ser muito grande (até 10^{144}), não é possível guardá-los em um inteiro normal. Existem três soluções principais:

- Usando `BigInteger` (Java) ou alguma outra classe de bignum. Isso funciona, mas se você não usa Java pode ser complicado.
- Usando hashing, isso é, pegar o resultado do produto módulo um primo grande (da ordem de 10^9). Só fazer isso não funciona pois estamos checando $\approx 2^{32}$ pares, então a chance de colisão usando apenas um primo é grande. A solução é fazer isso para 2 primos diferentes.
- Antes da multiplicação, fatore todos os números, e guarde sua decomposição prima. Quando for multiplicar dois números, some seus primos. O tamanho dessa representação é aproximadamente log, que é pequena o bastante. Recomendo usar `map<int, int>` para representar cada número, para cada primo guardando sua potência.

Isso pode ser implementado em tempo $\mathcal{O}(2^K \cdot K)$ (hash) ou $\mathcal{O}(2^K \cdot K \cdot P)$, onde $K = \max(N, M)$ e P é o número de primos na decomposição de todos os números da entrada, esse valor é bem pequeno.

Problema I: Protegendo o Central Park

Autor do problema: Renzo Gonzalo Gómez Diaz
Análise: Renzo Gonzalo Gómez Diaz

Seja $G = (V, E)$ o grafo não orientado tal que V representa os lugares que desejamos proteger, e E representa os caminhos que ligam esses lugares. O problema se reduz a achar uma partição das arestas do grafo usando caminhos de tamanho 2, ou seja, queremos uma coleção de caminhos

$$\mathcal{C} = \{P_1, P_2, \ldots, P_k\}$$

tal que $|E(P_i)| = 2$, para $i = 1, \ldots, k$, $E(P_i) \cap E(P_j) = \emptyset$ se $i \neq j$, e $\cup_{i=1}^{k} E(P_i) = E$. Observe que o enunciado garante a existência de tal solução.

Em primeiro lugar, observamos que retirar quaisquer duas arestas adjacentes em G, pode deixar o grafo desconexo, e teríamos que lidar com cada componente separadamente, além de ter cuidado de deixar cada componente com número par de arestas, para poder aplicar um algoritmo recursivo. Por tal motivo, o que desejamos é projetar um algoritmo que cria essa coleção de caminhos de maneira que deixa o grafo conexo após retirar duas arestas adjacentes no grafo.

Seja T uma árvore geradora de G. Observe que ao retirar uma folha de T o grafo resultante é uma árvore e, portanto, conexo. Logo, a estratégia será processar os vértices G segundo uma post-ordem de T. Seja v uma folha de T e u o único vizinho de v em T. Seja $\delta_G(v)$ o conjunto de arestas incidentes em v no grafo G. Notamos que, se $|\delta_G(v)|$ é par, podemos pairar arbitrariamente as arestas incidentes em v, formando caminhos de tamanho 2. Além disso, se consideramos o grafo $G' = G - v$, temos um grafo conexo com número par de arestas. Por outro lado, se $|\delta(v)|$ é ímpar, podemos pairar arbitrariamente as arestas em $\delta(v) - uv$. Note que, se consideramos o grafo $G' = G - \delta(v) + uv$, temos um grafo conexo com número par de arestas. Desta forma, podemos processar os vértices de G segundo uma post-ordem em T, para obter a resposta. Este algoritmo consome tempo $\mathcal{O}(|V| + |E|)$.

Problema J: King of Tokyo

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

A primeira coisa a se notar é que a ordem dos dados não importa, só a quantidade, e todas as faces não numeradas podem ser tratadas igualmente. Isso significa que em vez de considerar todas as 6^6 rolagens de dados, podemos considerar apenas as $\binom{9}{3} = 84$ rolagens essencialmente diferentes (aviso: essas rolagens não são mais equiprováveis).

Resolveremos o problema usando programação dinâmica, simulando todas as possíveis rerrolagens e resultados. A pd `solve(dados, k)` dirá o número esperado de pontos, jogando de forma ótima, quando a primeira rolagem foi `dados` (uma das 84 rolagens diferentes) e podemos rerrolar um subconjunto `k` vezes.

Para fazer isso, vamos testar todos os subconjuntos de `dados` possíveis (existem $\binom{10}{4} = 210$ jeitos diferentes de rerrolar entre todas as possíveis rolagens, então isso é um limite superior para rerrolar `dice`). Depois de fazer essa rerrolagem de, por exemplo, G dados, cada um dos 6^G resultados são equiprováveis, mas só existem $\binom{G+3}{3} \leq 84$ resultados essencialmente diferentes, então podemos precalcular a probabilidade de cada um destes acontecer, e usar isso para chamar a pd recursivamente e determinar o valor esperado fazendo essa rerrolagem. A resposta será o máximo entre todas possíveis rerrolagens.

É preciso usar a mesma pd para todos os T testes, sem recalculá-la. O número de operações desse código vai ser por volta de $K \cdot 84 \cdot 210 \cdot 84$.

Problema K: Monte Rushmore e Aniversários

Autor do problema: Yan Soares Couto
Análise: Yan Soares Couto

Considerando um ano com N dias, a probabilidade que i pessoas em uma sala tenham aniversários em dias diferentes é $P(i) = \frac{N}{N} \cdot \frac{N-1}{N} \cdots \frac{N-i+1}{N}$. É preciso calcular o menor i tal que $1-P(i) \geq 0.5$, isso é, $P(i) \leq 0.5$.

Problema L: O Problema da Mochila

Autor do problema: Arthur Nascimento
Análise: Yan Soares Couto

Uma forma (em geral não tão esperta) de resolver o Problema da Mochila é usar programação dinâmica da seguinte forma: $f(S)$ é o melhor jeito de resolver o problema com no máximo S de peso. A recorrência tem 3 casos possíveis:

1. Não escolher nenhum item, nesse caso $f(S) = 0$.
2. Escolher 1 item, nesse caso a resposta é o maior valor de um único objeto com custo no máximo S.
3. Escolher 2 ou mais itens, nesse caso, o conjunto de itens escolhidos pode ser particionado em dois conjuntos não vazios de tamanho que não excede X e $S - X$, para algum X, e então $f(S) = f(X) + f(S - X)$.

O problema com o Caso 3 é que não sabemos o tamanho X, então a abordagem direta é calcular

$$f(S) = \min_{1 \leq X < S} (f(X) + f(S - X)).$$

Note que isso vai funcionar para **qualquer** particionamento correto de S. Neste problema $w_i \leq 10^3$, podemos assumir que $|X - (S - X)| \leq 10^3$ e, portanto, $\frac{S}{2} - 500 \leq X \leq \frac{S}{2} + 500$. Queremos calcular apenas $f(S)$, e para isso devemos calcular o intervalo $[\frac{S}{2} - W, \frac{S}{2} + W]$, onde $W = 500$. Para obter os valores de f nesse intervalo, devemos calcular o intervalo $[\frac{S}{4} - \frac{3W}{2}, \frac{S}{4} + \frac{3W}{2}]$ e assim por diante.

Pode ser provado que a constante que multiplica W nunca excederá 2 (exercício, é fácil :P), e então o "nível" K não vai ser maior que $[\frac{S}{2^K} - 1000, \frac{S}{2^K} + 1000]$. É claro que só precisamos calcular por volta de $\log_2(S)$ níveis, e então só precisamos calcular $\approx 4 \cdot W \cdot \log_2(S)$ estados da pd, e cada um deles leva tempo $2 \cdot W$. Então, esse problema pode ser resolvido em tempo $\mathcal{O}(W^2 \cdot \log(S))$.