Universidade de Brasília

Seletiva para a Maratona de Programação 2016

Editorial

18 de junho de 2016

A) Sobre a entrada

1. A entrada de seu programa deve ser lida da *entrada padrão*.
2. Quando uma linha da entrada contém vários valores, estes são separados por um único espaço em branco; a entrada não contém nenhum outro espaço em branco.
3. Cada linha, incluindo a última, contém o caractere final-de-linha.
4. O final da entrada coincide com o final do arquivo.

B) Sobre a saída

1. A saída de seu programa deve ser escrita na *saída padrão*.
2. Quando uma linha da saída contém vários valores, estes devem ser separados por um único espaço em branco; a saída não deve conter nenhum outro espaço em branco.
3. Cada linha, incluindo a última, deve conter o caractere final-de-linha.

C) Sobre os problemas

As situações retratadas nos problemas são inteiramente fictícias e não correspondem à realidade. Nada escrito nos enunciados tem a intenção de desrespeitar o leitor. Tudo foi escrito de maneira a se adequar às histórias do desenho animado escolhido como tema.

A AWESOM-O

Autor: Matheus Pimenta

Fácil.

B Bradley Biggle

Autor: Matheus Pimenta

Solução via programação dinâmica:

Primeiro, rotulamos os N vértices pelos números de 1 a N, em sentido horário, e denotamos por $c(i, j)$ o comprimento do segmento de reta que liga os vértices i e j, $1 \leq i, j \leq N$.

Defina então $f(i, j)$ como a soma de uma decomposição mínima para o subpolígono convexo formado pelos segmentos de reta i—$i+1$, $i+1$—$i+2$, ..., $j-1$—j e pelo segmento i—j; e defina $w(i, j) = c(i, j)+f(i, j)$. Então

$$f(i,j) = \begin{cases} 0 & \text{se } j \leq i+2 \\ \min(\{w(i+1,j), w(i,j-1)\} \cup \{w(i,k) + w(k,j) \mid i+2 \leq k \leq j-2\}) & \text{caso contrário} \end{cases}$$

e a resposta do problema é $f(1, N)$.

Há $O(N^2)$ estados e cada estado custa $O(N)$ para ser calculado. Logo, a complexidade de tempo da solução é $O(N^3)$, com custo de espaço $O(N^2)$.

Explicação:

No estado (i, j), a programação dinâmica que definimos deve dar a resposta do problema para o subpolígono formado pelos segmentos i—$i+1$, $i+1$—$i+2$, ..., $j-1$—j e i—j. Em qualquer triangulação ótima, cada um destes segmentos que formam o polígono deve fazer parte de um triângulo. Então, para obter a resposta que queremos, observe que basta fixar um destes segmentos e determinar qual é o triângulo do qual ele deve fazer parte. Fixamos então o segmento i—j. Com os vértices i e j fixados, resta apenas determinar o terceiro vértice, que fecha o triângulo. As únicas possibilidades são os vértices k tais que $i < k < j$. Assim, dividimos o subpolígono do estado (i, j) em dois subpolígonos menores, separados pelo triângulo do segmento i—j. O subpolígono da esquerda é formado pelo segmento i—k e pelos vértices entre i e k. O subpolígono da direita é formado pelo segmento k—j e pelos vértices entre k e j. Os três vértices do triângulo que separa os dois subpolígonos são i, k e j. Nesta decomposição, somamos os cortes ótimos internos aos dois subpolígonos menores ($f(i, k)$ e $f(k, j)$) e os comprimentos $c(i, k)$ e $c(k, j)$. O caso base, em que $j \leq i + 2$, é responsável por figuras geométricas que são no máximo triângulos, ou seja, nenhuma decomposição precisa ser feita. Os casos representados por $w(i + 1, j)$ e $w(i, j - 1)$ consideram as duas decomposições extremas em que exatamente um dos dois subpolígonos não aparece. No caso $w(i + 1, j)$, o subpolígono da esquerda não aparece, enquanto que no outro caso, $w(i, j - 1)$, é o subpolígono da direita que não aparece.

C Criptografando no Pentágono

Autor: Matheus Pimenta

Resumidamente, o problema pede a diferença entre M e a dimensão do espaço vetorial gerado pelos vetores dados na entrada. Basta, por exemplo, considerar os vetores dados como vetores-linha, montar a matriz associada e escalonar esta matriz (com o algoritmo da eliminação gaussiana, que é $O(\max(M, N)^3)$) contando o número de pivôs para encontrar a dimensão.

D Desordenando a Blockbuster

Autor: Matheus Pimenta

Solução via programação dinâmica:

Defina $f(i, j)$ como o número de sequências distintas de comprimento j feitas somente de cópias dos filmes de 1 a i. Então

$$f(i,j) = \begin{cases} 1 & \text{se } j = 0 \\ 0 & \text{se } j > 0 \text{ e } i = 0 \\ \sum_{k=0}^{\min\{c_i, j\}} \binom{j}{k} f(i-1, j-k) & \text{caso contrário} \end{cases}$$

e a resposta do problema é $f(N, K)$.

Seja $C = \max\{c_i \mid 1 \leq i \leq N\}$. Então há $O(NK)$ estados e cada estado custa $O(C)$ para ser calculado. Logo, a complexidade de tempo da solução é $O(NKC)$, com custo de espaço $O(NK)$.

Explicação:

Só existe uma única sequência de comprimento zero: a sequência vazia. Ou seja, $f(i, 0) = 1, \forall i$. No entanto, se $i = 0$ e consideramos somente elementos dos tipos $1, ..., i$, então, na verdade, não consideramos nenhum tipo de elemento. Ou seja, $f(0, j) = 0, \forall j > 0$. Por fim, decompomos a solução da seguinte maneira. Imagine uma prateleira vazia com j posições para filmes. Podemos colocar no máximo $\min\{c_i, j\}$ cópias do filme i nesta prateleira. Além disso, há exatamente $\binom{j}{k}$ maneiras distintas de colocar k cópias do filme i nesta prateleira, onde $0 \leq k \leq \min\{c_i, j\}$, ou seja, escolhemos k das j posições para preencher com as k cópias do filme i. E, para cada uma destas $\binom{j}{k}$ maneiras distintas de fixar k cópias do filme i na prateleira, há exatamente $f(i-1, j-k)$ sequências distintas de comprimento $j-k$ para preencher as $j-k$ posições restantes, entre as que consideram somente cópias dos filmes de 1 a $i-1$. Logo, para obter $f(i, j)$ no caso geral, basta somar $\binom{j}{k} f(i-1, j-k)$ para todos os valores possíveis de k.

E Eleições em South Park Elementary

Autor: Matheus Pimenta

Fácil.

F Fatbeard

Autor: Matheus Pimenta

Pelo enunciado, Fatbeard só irá saquear arquipélagos de correntes que não fazem parte de uma componente fortemente conexa do grafo associado ao problema, ou seja, Fatbeard só considera saquear as arestas que fazem parte do DAG das componentes fortemente conexas. O enunciado, no entanto, não diz que ele não pode navegar por outras arestas. Neste caso, basta encontrar $scc(u)$ para cada $u \in V$ (a componente fortemente conexa de cada vértice do grafo) com o algoritmo de Kosaraju, por exemplo, e aplicar a programação dinâmica descrita a seguir, que também possui custo de tempo linear.

Defina $f(u)$ como o lucro máximo saindo do vértice u. Então

$$f(u) = \max(\{0\} \cup \{w(u, v) + f(v) \mid (u, v) \in E\})$$

e $f(F)$ é a resposta do problema, onde $w(u, v)$ é zero se $scc(u) = scc(v)$, ou o lucro obtido ao navegar de u para v em caso contrário.

Alternativamente à esta programação dinâmica, é possível também adaptar o algoritmo de Dijkstra para este problema. É justamente pelo fato do grafo ser acíclico e dirigido que a possibilidade de laço infinito no algoritmo de Dijkstra é eliminada. A complexidade é um pouco pior, mas é satisfatória.

G Guitar Queer-O

Autor: Matheus Pimenta

Fácil. Basta notar que a pontuação máxima é o dobro do número de teclas que aparecem na primeira sequência e que a pontuação mínima é $-10N$.

H Heaven vs. Hell

Autor: Matheus Pimenta

Problema de simulação com escolha gulosa. Basta saber integrar retas sobre intervalos definidos (duh!) e simular a batalha ao longo do tempo.

A escolha do próximo alvo é feita encontrando o demônio que irá causar mais dano se não for escolhido naquele instante. Ou seja, fazemos uma varredura linear nos demônios que já estão presentes no Céu naquele instante, atualizando a escolha sempre que aparecer uma melhor. Para comparar dois demônios distintos i e j, calculamos os danos futuros

$$X_i = \int_{t_n+C}^{t_n+2C} d_i(t)dt \qquad \text{e} \qquad X_j = \int_{t_n+C}^{t_n+2C} d_j(t)dt$$

onde t_n é o instante atual da simulação. Se $X_j < X_i$, então i é uma escolha melhor que j. Note que, de fato, o intervalo de interesse é $[t_n + C, t_n + 2C]$. O que acontece no intervalo $[t_n, t_n + C]$ não importa! Os dois demônios vão causar dano neste intervalo de qualquer jeito!

Há um detalhe interessante sobre este problema. Um demônio que chegou mais tarde começa causando zero de dano por minuto. Mas ele pode rapidamente se tornar a pior das ameaças, se for rápido e forte o suficiente! Ou seja, uma escolha pode deixar de ser pior que outra ao longo do tempo! Isto quer dizer que, sempre que o tempo da simulação avançar e uma nova escolha tiver que ser feita, é necessário considerar todas as escolhas possíveis, recalculando os danos a partir daquele instante. Logo, não é possível usar *heaps*, ou filas de prioridade! O problema da escolha gulosa, neste caso, é inerentemente linear! Logo, a complexidade da solução é $O(N^2)$.

Por fim, ao escolher o demônio i no instante t_n, acumula-se na resposta do problema todo o dano causado por este demônio:

$$\int_0^{t_n+C} d_i(t)dt$$

I Intergaláticos Cometas e a Princesa do Canadá

Autor: Matheus Pimenta

É possível simular a fase do jogo para uma dada quantidade de energia cinética inicial e verificar se ela é suficiente para vencer, processando um evento de cada vez. Com isso, basta fazer uma busca binária no espaço das energias iniciais, que são os números inteiros do intervalo $[1, 10^{10}]$ (10^{10} é pior caso, porque o custo de energia é no máximo 10^5 por quilômetro e há no máximo 10^5 quilômetros para percorrer). Isto resulta em $O(N \lg 10^{10}) = O(10N \lg 10) = O(N)$.

Uma outra solução satisfatória é o algoritmo de Kadane, que também executa em $O(N)$.

J Jornada em Azeroth

Autor: Matheus Pimenta

Problema clássico de empacotamento de grafos. Considere o grafo com conjunto de vértices $V_P = \{(u, u_g, s) \mid u, u_g \in V \text{ e } s \text{ é um } \textit{bitmask} \text{ de } K \textit{ bits}\}$. Um vértice (u, u_g, s) deste grafo empacotado significa que o personagem de Stan está na cidade u, a gema sagrada está na cidade u_g e o *bitmask* s indica quais das K gemas profanas já foram destruídas. A partir desta representação, basta derivar as arestas implícitas do empacotamento e executar o algoritmo de Dijkstra, com custo $O(MN2^K \lg N^2 2^K)$.

K Kyle Schwartz

Autor: Matheus Pimenta

Problema clássico de logaritmo discreto. Basta utilizar o algorito *baby-step giant-step* ($O(\sqrt{p} \lg \sqrt{p})$), ou o algoritmo Pollard's rho ($O(\sqrt{p})$).