Solução para o problema "Interplanetário" da XXIV Maratona de Programação

Claro Henrique Silva Sales[1], **Matheus Xavier Sampaio**[1], **Wladimir Araújo Tavares**[1]

[1]Universidade Federal do Ceará (UFC) – Campus Quixadá

{henriquesales, xavier_sampaio}@alu.ufc.br, wladimir@lia.ufc.br

Abstract. *This paper analyzes and describes a solution to the "Interplanetário" problem of the regional phase of the SBC 2019 Programming Marathon. The solution must be efficient in meeting the time and memory constraints imposed on the Marathon. To do so, we will adapt the Floyd-Warshall algorithm and divide the problem into several subproblems and solve them in an orderly manner. Finally, the performance of the solution will be presented in an Online Judge.*

Resumo. *Este artigo analisa e descreve uma solução para o problema "Interplanetário" da fase regional da Maratona de Programação da SBC 2019. A solução deve ser eficiente para atender às restrições de tempo e memória impostas na Maratona. Para isso, adaptaremos o algoritmo de Floyd-Warshall, além de dividir o problema em vários subproblemas e resolvê-los de forma ordenada. Por fim, será apresentada a performance da solução em um juiz on-line.*

1. Introdução

A Maratona de Programação [SBC 2019] é um evento existente desde 1996 realizado pela Sociedade Brasileira de Computação (SBC). Ela é destinada a estudantes de cursos de graduação e início de pós-graduação na área de Computação e afins. Nesta competição, equipes formadas por três estudantes tentam resolver o máximo de problemas possíveis dentro de cinco horas.

Para uma solução ser aceita, além fornecer o resultado esperado pelos juízes, ela deve atender os limites de tempo e memória especificados para cada problema. Dessa forma, a competição incentiva aos estudantes a criatividade, capacidade de trabalho em equipe, a busca de novas soluções mais eficientes de software e a habilidade de resolver problemas sob pressão. Os problemas exigem que os competidores realizem adaptações no algoritmo e explorem suas características. O problema abordado por esse artigo é um exemplo desta situação. No problema "Interplanetário", apresentado na primeira fase da XXIV Maratona de Programação, os competidores precisam realizar adaptações interessantes no algoritmo Floyd-Warshall [Cormen. 2009], algoritmo conhecido por encontrar o caminho mínimo entre todos os pares de vértices.

Esse artigo está organizado como descrito a seguir. Na Seção 2, apresentaremos a descrição do problema apresentada na Maratona de Programação. Na Seção 3, apresentamos o algoritmo de Floyd-Warshall genérico. Na Seção 4, mostramos as modificações realizados no algoritmo para resolver o problema proposto. Na Seção 5, faremos um breve relato dos resultados obtidos.

2. Descrição do Problema

2.1. Problema "Interplanetário"

Estamos no ano de 2306 e, com o avanço da nanotecnologia, viagens interplanetárias estão cada vez mais acessíveis. Bibika trabalha na maior agência de viagem interplanetária do universo e recebe clientes interessados diariamente.

Os clientes de Bibika são exigentes e fazem várias demandas antes de fechar o roteiro de suas viagens, como minimizar a distância total percorrida. Mas as maiores restrições são com relação às temperaturas dos planetas visitados no percurso (excluindo os planetas de origem e de destino). A temperatura de um planeta, medida em graus Anidos, pode variar de 10^9 graus Anidos negativos até 10^9 graus Anidos positivos. Os clientes de Bibika são oriundos de planetas de climas variados e, consequentemente, possuem preferências diferentes em relação a temperatura: alguns se incomodam com planetas muito frios e outros com planetas muito quentes. Bibika precisa planejar a rota das viagens de forma a poupar seus clientes de qualquer desconforto, mesmo que para isso o comprimento total da rota não seja o menor possível (ou até mesmo que não exista uma rota: nesse caso Bibika simplesmente informa os clientes de que a viagem é impossível).

Bibika lhe forneceu a temperatura média histórica de cada um dos N planetas e as R rotas que ligam pares de planetas diretamente (é garantido que entre dois planetas existe no máximo uma rota direta), juntamente com suas respectivas distâncias. Ela lhe fornecerá também os pedidos de viagem de Q clientes. Cada pedido consiste de um planeta de origem A, um planeta de destino B, e a restrição do cliente em relação às temperaturas dos planetas intermediários: cada cliente pode exigir passar apenas por planetas com temperaturas entre as K menores ou K maiores dentre todos os N planetas.

Sua tarefa é, para cada pedido de viagem, encontrar a menor distância percorrida possível dadas as restrições descritas, ou dizer que a viagem é impossível.

2.2. Entrada

A primeira linha contém dois inteiros N e R ($2 \leq N \leq 400$ e $0 \leq R \leq N \cdot (N-1)/2$), representando a quantidade de planetas conhecidos e a quantidade de rotas diretas entre eles. O primeiro planeta é representado pelo número 1, o segundo pelo número 2, ..., até o N-ésimo pelo número N. A segunda linha contém N inteiros T_i ($-10^9 \leq T_i \leq 10^9$), representando a temperatura média de cada um dos planetas. A seguir haverá R linhas, cada uma contendo três inteiros X, Y e D ($1 \leq X, Y \leq N$ com $X \neq Y$ e $1 \leq D \leq 103$), representando uma rota direta de comprimento D entre os planetas X e Y. Em seguida haverá um inteiro Q ($1 \leq Q \leq 105$), representando a quantidade de pedidos de viagens dos clientes. Por fim, cada uma das próximas Q linhas conterá quatro inteiros A, B, K e T ($1 \leq A, B, K \leq N$ com $A \neq B$ e $T \in \{0, 1\}$), representando um cliente que deseja ir do planeta A para o planeta B passando apenas por planetas que tenham alguma das K menores temperaturas, se $T = 0$ ou K maiores temperaturas, se $T = 1$.

2.3. Saída

Seu programa deve produzir uma linha para cada cliente contendo um inteiro que representa a menor distância total de viagem entre os dois planetas dadas as restrições do cliente, ou 1 caso a viagem não seja possível.

3. Algoritmo de Floyd–Warshall

O algoritmo de Floyd–Warshall [Cormen. 2009] é um algoritmo de programação dinâmica usado para encontrar a menor distância entre todos os nós de um grafo ponderado. O resultado da execução do algoritmo é uma matriz contendo a menor distância entre todos os pares de vértices pertencentes ao grafo.

3.1. Descrição

Seja G um grafo não-direcionado cujo os vértices V são numerados de 1 até N, onde $N = |V|$. Seja D uma matriz quadrada de tamanho $N \cdot N$, onde $D[i][j]$ guarda a distância do menor caminho entre os vértices i e j encontrada até o momento.

O algoritmo consiste em k iterações $(k = 1, 2, .., N)$. Após a k-ésima iteração, a matriz $D[i][j]$ guarda a distância do menor caminho entre os vértices i e j utilizando somente os vértices de $[1, 2, .., k]$ como vértices intermediários. Em outras palavras, antes da k-ésima iteração, $D[i][j]$ guarda a distância do menor caminho entre os vértices tal que nesse caminho entre i e j, só serão permitidos os vértices com números entre $[1, 2, .., k - 1]$. Somente os vértices intermediários são restritos a essas regras, ou seja, i e j não serão afetados.

Supondo que todas os menores caminhos foram calculados considerando os vértices $[1, 2, .., k - 1]$ como intermediários, podemos então calcular os menores caminhos adicionando o vértice k como intermediário. Nesse caso, ao calcular $D[i][j]$ entre dois vértices i e j, existem duas possibilidades:

- O vértice k não está presente no menor caminho possível entre i e j. Nesse caso, $D[i][j]$ não é alterado.
- O vértice k faz parte do menor caminho entre i e j. Isso indica que podemos dividir esse caminho de $i \rightarrow j$ em dois caminhos de $i \rightarrow k$ e $k \rightarrow j$. Portanto, $D[i][k] + D[k][j]$ representa o comprimento do menor caminho entre os vértices i e j utilizando o vértice k como intermediário.

Considerando esses dois casos, é possível recalcular todos os pares (i,j) na k-ésima fase com a seguinte fórmula:

$$D[i][j] = min(D[i][j], D[i][k] + D[k][j])$$

Esse cálculo deve ser repetido até a N-ésima iteração, onde os vértices de 1 até N poderão ser usados como vértices intermediários, garantindo então a resposta ótima para todos os pares de vértices do grafo.

4. Solucionando o Problema "Interplanetário"

4.1. Modelagem

Um planeta será representado por uma tupla *(temperatura, indice)*, representando, respectivamente, a temperatura média do planeta e o número que o identifica. Cada pedido será representado como uma 5-tupla *(origem,destino,i,k,tipo)* representando, respectivamente, o planeta inicial, o planeta destino, número que identifica o índice do pedido, escolha entre os k planetas mais frios se tipo = 0 ou entre os k planetas mais quentes se tipo = 1.

Para armazenar as informações dos planetas, será utilizado um vetor P de tamanho N. De forma semelhante, o vetor Q de tamanho M armazenará os dados dos pedidos.

Para armazenar as rotas entre os planetas, será utilizada uma matriz de adjacência G onde:

$$G(i,j) = \begin{cases} w, & Se\ existe\ aresta\ (i,j)\ de\ distância\ w \\ \infty, & Caso\ contrário \end{cases}$$

Será utilizado um vetor $Kfrios$, onde $Kfrios[i]$ é a temperatura do i-ésimo planeta mais frio. Por último, um vetor $Result$ será usado para armazenar a resposta de cada pedido, onde $Result[i]$ é a menor distância calculada para satisfazer o pedido de índice i.

4.2. Solução

Discutiremos como resolver os pedidos do tipo 0. Em seguida iremos resolver os pedidos do tipo 1 de forma análoga.

Como foi discutido na Seção 3, o algoritmo de Floyd–Warshall é um algoritmo iterativo onde em cada fase um vértice é inserido como caminho intermediário. Considere que o algoritmo armazena seus resultados em uma matriz D. Dessa forma, uma solução trivial seria, para cada pedido $Q[i]$, inserir todos os planetas cujo a temperatura esteja entre os $Q[i].k$ planetas mais frios e depois consultar $D[P[i].origem][P[i].destino]$.

No entanto, a complexidade de inserir um elemento como intermediário no Floyd–Warshall é O(N^2). No pior caso, um pedido pode exigir que todos os planetas sejam inseridos ($P[i].k = N$), alcançando uma complexidade $O(N^3)$ para cada pedido, totalizando $O(Q.N^3)$ para resolver todo os problemas.

Para uma melhor eficiência, ao invés de executar o Floyd–Warshall por inteiro para todas as consulta, vamos ordenar os pedidos, e dessa forma aproveitar o resultado do calculo do i-ésimo pedido para calcular o (i+1)-ésimo pedido.

Considere que o vetor de planetas P está ordenado de forma não-decrescente em relação ao seu atributo k. Ou seja, para todo $i \in \{1..n\text{-}1\}$, $P[i].k \leq P[i+1].k$. Por consequência, o conjunto de planetas que $Q[i]$ aceita como intermediário está contido no conjunto de planetas que $Q[i+1]$ aceita, uma vez que um k maior apenas acrescenta novos planetas aceitáveis. Temos dois casos:

- $Q[i].k = Q[i+1].k$, Os dois pedidos possuem a mesma restrição de planetas intermediários, nesse caso, podemos aproveitar a matriz D calculada na interação anterior e responder o $(i+1)$-ésimo pedido em $O(1)$ apenas consultando $D[Q[i].origem][Q[i].destino]$.
- $Q[i].k$¡$Q[i+1].k$, Nesse caso, para resolver o pedido $i+1$ devemos calcular D adicionando novos planetas que respeitem a restrição de estar entre os $Q[i+1].k$ mais frios. A complexidade de adicionar um novo planeta é $O(N^2)$.

Ao reaproveitar a matriz de distâncias, garantimos que o número de iterações do Floyd–Warshall é no máximo N. Com a matriz já calculada, responder um pedido consiste apenas em consultar a matriz D, ou seja, complexidade é $O(1)$ para cada pedido. Além disso, é necessário ordenar Q pedidos e N planetas, gerando uma complexidade final de $O(Q.\log(Q) + N^3)$.

4.3. Pseudocódigo

A construção do algoritmo segue a estrutura discutida:

Algoritmo 1: *AdicionaPlaneta*

```
Entrada: Um inteiro x, matriz de distância D
Saída: Matriz D calculada após inserir o planeta x como intermediário
1 início
2     para i ← 1 até N faça
3         para j ← 1 até N faça
4             se D[i][k] + D[k][j] < D[i][j] então
5                 D[i][j] ← D[i][k] + D[k][j];
6             fim
7         fim
8     fim
9 fim
```

Algoritmo 2: SOLUÇÃO PARA OS PEDIDOS DO TIPO 0

```
Entrada: Matriz de rotas M, vetor P com informações dos planetas, vetor Q com informações dos pedidos
Saída: Vetor result com as respostas dos pedidos do tipo 0s
1  início
2      D ← M
3      Q_0 ← pedidos de Q do tipo 0
4      ordene P de forma não-decrescente em k
5      ordene Q_0 de forma não-decrescente em k
6      result ← vetor de inteiros com tamanho |Q|
7
8      para i ← 1 até |Q_0| faça
9          enquanto P[j].temperatura ≤ kTemp[Q_0[i].k] faça
10             AdicionaPlaneta(P[j].id, D)
11             j ← j + 1
12         fim
13         result[Q_0[i].id] ← D[Q_0[i].origem][Q_0[i].destino]
14     fim
15     retorna result
16 fim
```

O algoritmo para resolver os pedidos de tipo 1 pode ser construído de forma análoga, é preciso apenas ordenar P e Q_1 de forma não-crescente.

5. Conclusão e Resultados

Neste artigo foi apresentada uma forma eficiente de solucionar o problema "Interplanetário" da fase regional da Maratona de Programação da SBC 2019. O algoritmo de Floyd–Warshall foi estudado e adaptado para inserir vértices em uma ordem interessante para o problema, além de resolver os pedidos de forma ordenada, ou seja, off-line.

Esta solução foi escrita em C++ e foi submetida na plataforma Codeforces, onde recebeu o veredito de aceito. O tempo de execução alcançou 265 milissegundos, respeitando o limite imposto de 1000 milissegundos. Em relação ao custo de memória, a solução submetida utilizou 4,9 megabytes dos 256 megabytes permitidos.

Referências

Cormen., T. H. (2009). Floyd–warshall algorithm. In *Introduction to Algorithms*. third edition.

SBC (Data de acesso: 30-10-2019). *http://maratona.ime.usp.br/*. XXIV Maratona de Programação SBC.