Relatório do Problema de Programação 3 – *Bike Lanes*

Equipa:

Nº de Estudante: 2018285941 Nome: Cláudia Filipa Peres Saraiva de Campos
Nº de Estudante: 2018275530 Nome: Dário Filipe Torres Félix

1. Descrição do Algoritmo

1.1. Identificação dos Circuitos

No contexto do problema, um circuito corresponde ao conjunto máximo de Pontos de Interesse (POI) cujas conexões entre eles permitem qualquer POI viajar até outro POI qualquer. Esta definição de Ponto de Interesse corresponde à definição de Componentes Fortemente Conexos (SCC). Por isso, para a identificação dos circuitos, utilizámos o algoritmo de *Tarjan*, adaptado ao nosso problema: apenas adicionamos os circuitos cujo comprimento é superior a 1.

$\textbf{Function } tarjanStep(v, t, G, dfs, low, stack, inStack, circuits)$
  $dfs[v] \leftarrow t$
  $low[v] \leftarrow t$
  $t \leftarrow t + 1$
  $stack.push(v)$
  $inStack[v] \leftarrow \textbf{true}$
  $\textbf{for each } arc\ \{v, w\} \textbf{ in } G \textbf{ do}$
    $\textbf{if } dfs[w] = -1 \textbf{ then}$
      $t \leftarrow tarjanStep(w, t, dfs, low, stack, inStack, circuits)$
      $low[w] \leftarrow min(low[v], low[w])$
    $\textbf{else if } inStack[v] = \textbf{true then}$
      $low[v] \leftarrow min(low[v], dfs[w])$
  $\textbf{if } low[v] = dfs[v]\ then$
    $circuit \leftarrow \emptyset$
    $\textbf{do}$
      $w \leftarrow stack.pop(\quad)$
      $circuit.add(w)$
    $\textbf{while } w \neq v$
    $\textbf{if } circuit.length > 1 \textbf{ then}$
      $circuits.add(circuit)$
  $\textbf{return } t$

$\textbf{Function } tarjan(G)$
  $t \leftarrow 1$
  $\textbf{for each } v \in G \textbf{ do}$
    $dfs[v] \leftarrow -1$
    $low[v] \leftarrow -1$
  $inStack[v] \leftarrow \textbf{false}$
  $stack \leftarrow \emptyset$
  $circuits \leftarrow \emptyset$
  $\textbf{for each } v \in G \textbf{ do}$
    $\textbf{if } dfs[v] = -1 \textbf{ then}$
      $t \leftarrow tarjanStep(v, t, G, dfs, low, stack, inStack, circuits)$
  $\textbf{return } circuits$

1.2. Seleção das ruas para a construção das ciclovias

Na segunda fase do problema, depois de identificar todos os circuitos, é referido que, para cada circuito, deve-se escolher as estradas que permitem conectar todos os Pontos de Interesse com o menor comprimento possível. Ora esta definição é semelhante à definição de *Minimum Spanning Tree*.

Para calcular a *Minimum Spanning Tree* de um circuito utilizámos o algoritmo de *Kruskal*.

O grafo que corresponde à cidade é um grafo direcionado, porém o algoritmo de Kruskal só pode ser aplicado a grafos não-direcionados. No entanto, como as ciclovias não são de sentido único, ou seja, as bicicletas podem circular em ambos os sentidos, este algoritmo pode ser aplicado sem produzir resultados incorretos.

```
Function getCircuitsEdges(G, circuit)
  edges ← new List of Edge
  for each v in circuit do
    for each arc {v, u} in G do
      if u ∈ circuit then
        edge.a ← v
        edge.b ← u
        edge.d ← distance(v, u)
        edges.add(edge)
  return edges

Function kruskalStep(edges, n)
  set ← new Array[1..n] of Integer
  rank ← new Array[1..n] of Integer
  make_set(set, rank, n)
  length ← 0
  sort(edges) by distance
  for each edge in Edges do
    if find_set(set, edge.a) ≠ find_set(set, edge.b) then
      length ← length + edge.d
      union_set(set, rank, edge.a, edge.b)
  return length

Function kruskal(G, circuits, n)
  lengths ← new List of Integer
  for each circuit in circuits do
    edges ← getCircuitEdges(G, circuit)
    length ← kruskalStep(edges, n)
    lengths.add(length)
  return lengths
```

1.3. Melhoria na eficiência do algoritmo

1) O algoritmo de *Kruskal*, só é executado para todos os circuitos se o número de questões a responder for superior a 2 porque só a terceira e quarta perguntas é que dependem deste algoritmo.

Como as duas primeiras perguntas são dependentes do algoritmo de *Tarjan* e como o nosso algoritmo terá sempre de responder à primeira pergunta, independentemente do número de questões, o algoritmo de *Tarjan* tem sempre de ser executado.

2) Em vez de construir uma lista com todas as arestas selecionadas pelo algoritmo de *Kruskal* basta calcular o somatório do comprimento dessas mesmas arestas visto que esse é o único valor que vai ser necessário para responder às 2 últimas questões.

2. Estruturas de Dados

Utilizámos uma lista de adjacências para representar o grafo, na prática, traduz-se por um vetor de vetores que armazena uma estrutura que contém 2 inteiros. Optámos por uma lista em vez de uma matriz de adjacência porque, no algoritmo de *Tarjan*, temos de percorrer/aceder aos vizinhos de um determinado nó. Para podermos aplicar o algoritmo de *Kruskal* a cada circuito necessitamos de criar uma lista de arestas para cada circuito.

3. Verificação do Algoritmo

Os algoritmos de *Tarjan* e *Kruskal* são algoritmos bastante conhecidos e foram amplamente estudados e verificados. Por isso, deduzimos que a nossa abordagem, que é resultado da aplicação direta destes dois algoritmos, seja correta/válida.

4. Análise do Algoritmo

4.1. ***Tarjan***

4.1.1. Complexidade Temporal: $O(|V| + |E|)$

4.1.2. Complexidade Espacial: $O(|V|)$

4.2. ***Kruskal***

4.2.1. Complexidade Temporal:

Da aplicação do algoritmo de *Tarjan* resultam X circuitos. Cada circuito pode ter um número variável de vértices, aos quais denominamos $Y_i, i \in \{1, 2, ..., X\}$. A esses vértices estão associadas $Z_i, i \in \{1,2, ..., X\}$ arestas do grafo original. Tendo isto em conta concluímos que a complexidade da nossa adaptação deste algoritmo é:

$$O\left(\sum_{i=1}^{X} |Z_i| \cdot log|Y_i|\right) \approx O(X \cdot (|Z| \cdot log|Y|))$$

4.2.2. Complexidade Espacial: $O(|V| + |Z|)$

Dependendo do número de questões a responder a complexidade total do algoritmo irá variar.

5. Referências

Slides disponibilizados na disciplina de Estratégias Algorítmicas. 😉