Trabalho 1 INF330

Grupo:

Rodrigo Chichorro - 92535
Matheus Nunes - 92537
Caio Henrici - 92558

Entrada:

Visto que a entrada dos arquivos é no formato .json, fizemos uso da biblioteca jsoncpp, que consiste em um parser de json para facilitar a leitura. Para ler as arestas do grafo, procura-se os atributos “viaGlobalId”, e para cada novo ID, insere-se numa estrutura de dados “map” o código da aresta. É feita a mesma coisa para os vértices, onde cada novo ID é uma nova entrada no map. Caso a aresta já esteja no map, a resta é adicionada em um outro map, tal que a i-ésima aresta repetida é relacionado ao inteiro ‘i’.

Além disso, cada aresta lida é adicionada em uma lista de adjacência, a qual armazena na posição ‘i’ todos os vizinhos do vértice ‘i’ e a aresta que liga ‘i’ a este vizinho. Ex: 4 - 7,2; 7 - 4,2 (O vértice 4 está ligado ao vértice 7 pela aresta 2). O código da aresta é armazenado apenas para a impressão da saída. Esse método trata o caso de arestas repetidas sem alterar o grafo.

Em seguida, lemos os “starting points” do problema. Caso o starting point seja uma aresta, apagamos essa aresta da lista de adjacência e a transformamos em um vértice artificial, que estará ligado aos 2 (ou mais, no caso de aresta repetida) vértices adjacentes a essa aresta.

Resolução:

Para que todos os caminhos dos “starting points” para os “controllers” sejam gerados, usamos uma espécie de busca em profundidade modificada: quando um vértice v é visitado, ele é marcado como visitado. Contudo, quando seu processamento é finalizado, ele é desmarcado como finalizado. Isso garante que todos os possíveis caminhos serão verificados. Quando o vértice visitado é um “controller”, no retorno da chamada de função todo vértice e aresta que está na pilha (logo, no caminho) é marcado como parte da resposta. Uma implementação e explicação do algoritmo pode ser visto neste site:

https://www.geeksforgeeks.org/find-paths-given-source-destination/

Contudo, fizemos uma otimização no algoritmo: se a função visita a partir de um vértice ‘u’, um vértice ‘v’ que já está na resposta, ‘u’ também estará na resposta, ou seja, não precisamos continuar a busca a partir de ‘v’. Depois que o algoritmo percorre todo o grafo, todos os vértices e arestas que estavam em algum caminho são imprimidos na tela.

Essa função é chamada para cada par de “starting point” e “controller”.

Análise do método:

O algoritmo é bastante ineficiente para grafos grandes. Mesmo em casos com apenas 1 “starting point” e 1 “controller”, uma grafo com ~100.000 vértices demora um tempo indeterminado e inviável (mais do que 5 minutos). Havia outras otimizações que pensamos

em implementar (exemplo: o resultado de uma busca de um “starting point” ‘s1’ para um “controller“ ‘c’ pode ser aproveitado para outro par ‘s2’ -> ‘c’).
Contudo, como dito acima, mesmo com apenas 1 ”starting point” e 1 “controller”, um grafo grande o suficiente já não é resolvido em tempo aceitável, tentamos (sem sucesso), resolver este caso de um grafo grande. Outro problema é que grafos grandes estouram a pilha de chamada do computador (já que a função é recursiva), mas isso pode ser resolvido removendo o limite de memória da pilha. Uma implementação iterativa da função é possível, mas, novamente, nosso foco foi tentar (sem sucesso) resolver grafos grandes.
Porém, testes com casos menores e com exceções mostraram que o algoritmo resolve corretamente o problema, imprimindo todos os vértices e arestas que fazem parte da solução.

Casos de teste:

Numero de vertices: 500
Numero de arestas: 13000
Numero de controladores: 1
Numero de starting points: 1
real 0m19.722s
user 0m19.708s
sys 0m0.012s

Numero de vertices: 400
Numero de arestas: 8196
Numero de controladores: 1
Numero de starting points: 1
real 0m0.182s
user 0m0.177s
sys 0m0.005s

Numero de vertices: 300
Numero de arestas: 4793
Numero de controladores: 1
Numero de starting points: 1
real 0m0.104s
user 0m0.099s
sys 0m0.005s

Fontes de consulta:
Geeks for geeks: geeksforgeeks.org
Cplusplus reference: cplusplus.com/reference
https://en.wikibooks.org/wiki/JsonCpp

Modo de execução:
O programa inclui um makefile. Execute o comando make para compilar, “./programa (argumentos)” para executar e “make clear” para remover o executável.