**Descrição do Problema e da Solução**

Problema: Uma vez que os cidadãos não se podem cruzar, visto que há possibilidade de ficarem infetados com tal aproximação, cada rua, cada avenida e cada supermercado apenas pode ter um cidadão a deslocar-se de cada vez. Assim, neste problema foi nos pedido para desenvolver um algoritmo que devolvia o número máximo de cidadãos que podiam deslocar-se até supermercados diferentes sem se cruzarem uns com os outros.

Solução: Uma vez que se trata de um problema de fluxo máximo, modelámos o mapa da cidade como uma rede de fluxo construída da seguinte forma:

* Para cada cruzamento, criámos um node na rede de fluxo. Uma vez que cada cruzamento tem capacidade igual a 1, cada node é constituído por dois vértices, um de entrada e outro de saída, que são ligados por um arco de capacidade igual a 1;
* Adicionámos também dois vértices auxiliares *s* e *t* que representam a *source* e a *sink* da rede, respetivamente;
* O vértice *s* tem um arco para cada cidadão com capacidade igual a 1, visto que é o número máximo de pessoas que podem sair de casa (cidadãos representam múltiplas fontes);
* Cada supermercado tem um arco para o vértice *t* com capacidade igual a 1, visto que é o número máximo de cidadãos em cada supermercado (supermercados representam múltiplos destinos);
* Cada node tem o seu vértice de saída a apontar para o vértice de entrada dos outros nodes, que correspondem aos cruzamentos que lhe são adjacentes;

Assim, com o problema modelado, aplicámos um algoritmo de cálculo de fluxo máximo para obtermos o valor pretendido. Escolhemos o algoritmo Edmonds-Karp, porque a capacidade e o fluxo entre cada arco apenas variam entre 0 e 1. Esta escolha também teve em conta o facto do Edmonds-Karp utilizar uma BFS, o que permite calcular os caminhos mais curtos em primeiro lugar. (Ford-Fulkerson + BFS)

Na aplicação da BFS retornamos o conjunto de vértices que são predecessores da sink e estão à mesma distância da *source*, de forma a otimizar o código quando há vários caminhos de tamanho igual. Assim, não é necessário correr a BFS novamente para encontrar um novo caminho, mas sim, verificar se os vértices dos caminhos seguintes têm vértices que já foram utilizados nos caminhos anteriores com o mesmo tamanho. Só quando não há caminhos de tamanho igual, é que aplicamos a BFS novamente.

Na aplicação do algoritmo de Ford-Fulkerson, se a BFS retornar uma lista vazia, significa que a sink não é atingível a partir da source, logo, retorna o valor máximo de fluxo calculado até ao momento. Se a BFS retornar uma lista não vazia, verificamos se se pode adicionar fluxo a todos os arcos do caminho e se for possível, adicionamos, se não for, passamos para o caminho seguinte. Para esta verificação, guardamos numa lista os vértices utilizados nos caminhos anteriores do mesmo tamanho. Quando a BFS é novamente chamada, esta lista volta a estar vazia.

**Análise Teórica**

1. Leitura dos dados de entrada: na função que lê o input temos 2 *for’s:*

- Um para ler os supermercados e outro para ler os cidadãos → O(C) + + O(S) = O(V);

- Função que cria as ligações entre nodes (inicializa o grafo residual com arcos de capacidade igual a 1 ou 0) → O(E);

Assim, a complexidade da leitura dos dados de entrada é O(V + E).

1. Aplicação do algoritmo de Edmonds-Karp:

* Aplicação de BFS:

- *For* para efetuar *reset* a todos os nodes, que implica inicializar todos os vértices (vértice de entrada e vértice de saída) como não visitados, declarar o vértice predecessor e o arco que liga o predecessor ao vértice em questão a NULL→ O(V);

- Adiciona o vértice inicial (source) à lista de vértices (queue) → O(1);

*- While* a queue da BFS não está vazia, percorre os arcos de cada vértice da queue e adiciona o vértice de destino de cada arco à queue, marca o vértice como visitado, define o predecessor e o arco que liga o predecessor ao vértice → O(V + E);

- Se a sink não foi descoberta (predecessor igual a NULL), então retornamos uma lista de vértices vazia;

- Se a sink foi descoberta na aplicação da BFS, percorremos o resto da queue e verificamos se algum vértice tem uma ligação direta para a sink. Se sim, adicionamos a uma lista de vértices e retornamos no final → O(V);

Assim, a complexidade da aplicação do algoritmo da BFS é O(V + E) = O(E).

* Aplicação do algoritmo Ford-Fulkerson no caminho descoberto pela BFS:

- Chamada à função que aplica a BFS → O(V + E) = O(E);

- *While* lista que a BFS retorna não é vazia:

Definimos o predecessor da sink como o primeiro vértice da lista e removemos o vértice → O(1);

*For* para descobrir o arco que liga o predecessor à lista e definimos o novo arco da sink → O(E);

Construímos o caminho da source até à sink → O(V + E);

Se a lista de vértices utilizados nesta chamada da BFS estiver vazia, então podemos adicionar fluxo;

Se a lista não estiver vazia, verificamos se algum vértice do caminho de aumento é igual a algum vértice da lista de vértices utilizados → O(V). Se for, não podemos utilizar este caminho de aumento. Se não for, podemos utilizar;

Para adicionar fluxo ao caminho de aumento, percorremos todos os arcos do caminho (*for*) → O(VE), adicionamos os vértices utilizados (exceto a source e a sink) à lista de vértices utilizados → O(1) e percorremos os arcos dos vértices para definir a capacidade dos arcos residuais a 1 →O(E);

Assim, o número de aumentos de fluxos é O(VE).

-No fim, retorna o valor o valor máximo de fluxo calculado → O(1);

Assim, a complexidade da aplicação do Ford Fulkerson é O(VE).

Concluindo, a complexidade da aplicação do algoritmo Edmonds-Karp é O(VE)\*O(E) = O

1. Apresentação dos dados:

- Retorna para a linha de comandos o valor máximo de fluxo calculado pelo algoritmo Edmonds-Karp →O(1);

**Avaliação Experimental dos Resultados**

Seja:

M → Nº de avenidas;

N → Nº de ruas;

V → Nº de vértices = 2 \* (M \* N) + 2 (Vin e Vout para cada cruzamento + source + sink);

E →

Descrição do tipo experiências feitas e gráfico demonstrativo da avaliação de tempos associados.

Gerar vários grafos de tamanho incremental e cálculo dos tempos para cada instância. Gerar o gráfico do tempo em função do tamanho do grafo de entrada como exemplificado abaixo.



Concluir se o gráfico gerado está concordante com a análise teórica prevista.