Relatório 2º projeto ASA 2019/2020

Grupo: tp003

Aluno(s): Catarina Sousa (93695) e Nelson Trindade (93743)

Descrição do Problema e da Solução

Problema: Uma vez que os cidadãos não se podem cruzar quando saem de casa, 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 desenvolvermos 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 primeiro caminho mais curto encontrado. De forma a otimizar o código e não ser necessário correr a BFS para encontrar todos os caminhos, quando se encontra o primeiro caminho que atinge a sink, verifica--se se mais algum vértice, presente na queue da BFS, é predecessor da sink. Se for adiciona-se a uma lista de vértices. Esta lista de vértices permite encontrar todos os caminhos possíveis de igual tamanho para se atingir a sink através da source. Assim, só é preciso correr a BFS novamente quando não há mais caminhos do mesmo tamanho.

Na aplicação do algoritmo de Ford-Fulkerson, se a BFS retornar um caminho não vazio, então adicionamos fluxo a todos os arcos do mesmo. De seguida verificamos os vértices pertencentes à lista de possíveis predecessores da *sink*. Verificamos se todos os arcos do novo caminho estão disponíveis. Se sim, é um caminho de aumento e passamos ao vértice seguinte. Se não, eliminamos o vértice da lista e passamos ao seguinte.

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(V);

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

2. 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);

- *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 como visitado, define o predecessor e o arco que o liga ao predecessor → O(V + E);

- 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 → O(V);

- Posteriormente, construímos o primeiro caminho através dos arcos partindo da *source* e atingindo a *sink* e retornamos no final → O(V + E);

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

- <u>Aplicação do algoritmo Ford-Fulkerson no caminho descoberto pela BFS:</u>

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

- A lista retornada pela BFS é um caminho de aumento e assim, percorremos todos os vértices e arcos do caminho e definimos as capacidades dos arcos residuais correspondentes a 1 → O(VE).

Grupo: tp003

Aluno(s): Catarina Sousa (93695) e Nelson Trindade (93743)

- Posteriormente, verificamos os vértices que se encontram na lista de vértices de predecessores da *sink* (para descobrir novos caminhos com o mesmo tamanho). Definimos o predecessor da *sink* como o primeiro vértice da lista e eliminamos o vértice da lista. Verificarmos se é um caminho de aumento possível e se sim percorremos os arcos todos do caminho e definimos as capacidades dos arcos residuais a 1. Repetimos o mesmo processo até a lista de vértices estar vazia → O(VE).

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

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(VE^2).

3. Apresentação dos dados: Retorna o valor máximo de fluxo calculado→O(1);

Avaliação Experimental dos Resultados

Seja:

A → Nº de avenidas;

R → Nº de ruas;

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

E → 4 * 2 (ligações dos vértices dos cantos)

+ (2 * (R – 2)) * 3 (os vértices da primeira e da última rua têm 3 ligações (sem contar com o primeiro e último vértice))

+ (2 * (A – 2)) * 3 (os vértices da primeira e da última avenida têm 3 ligações (sem contar com o primeiro e último vértice))

+ [(A – 2) * (R – 2)] * 4 (restantes vértices têm 4 ligações)

+ S + C + V (S ligações dos supermercados à *sink*, C ligações da *source* até aos cidadãos e V ligações porque cada vértice tem uma ligação interna do Vin para o Vout)

E → 8 + 6 [(R – 2) + (A – 2)] + 4[(A – 2)(R – 2)] + C + S + V

Como cada arco (exceto os arcos da *source* para os cidadãos e os arcos dos supermercados para a *sink*) tem um arco residual na rede de fluxo:

E → 2 * [8 + 6 [(R – 2) + (A – 2)] + 4[(A – 2)(R – 2)] + C + S + V] – S – C

Relatório 2º projeto ASA 2019/2020

Grupo: tp003

Aluno(s): Catarina Sousa (93695) e Nelson Trindade (93743)

Efetuámos vários testes com mapas n * n gerados pelo gerador fornecido pelo professor. Gerámos mapas com n = 250, 500, 750, 1000, 1250, 1500, 1750 e 2000.

Assim, **V** $\rightarrow 2(n^2) + 2 = O(n^2)$

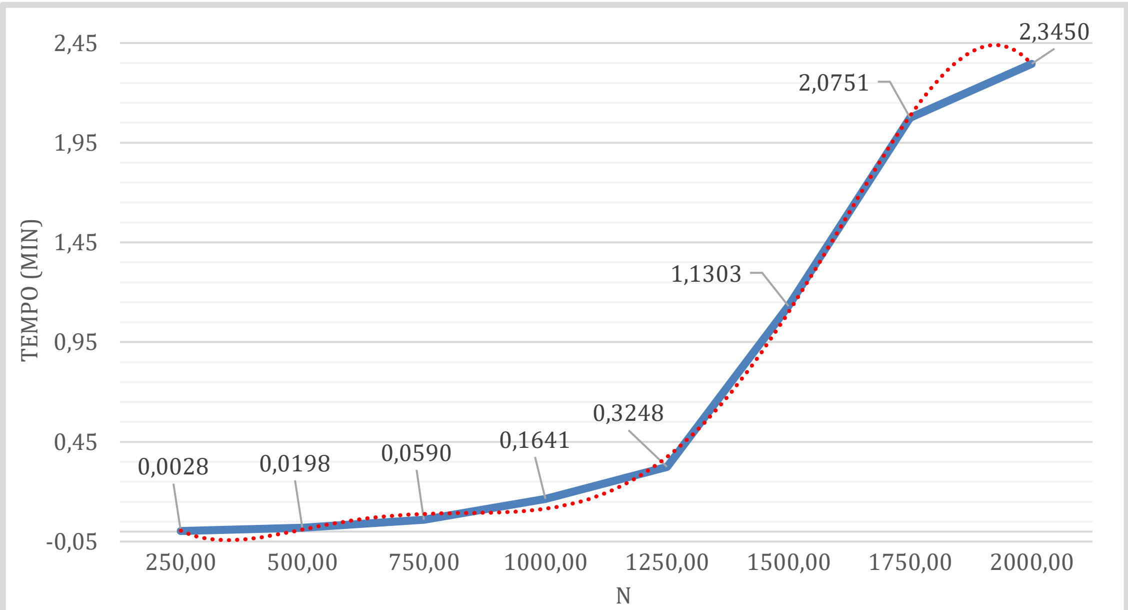
E $\rightarrow 10n^2 - 7.9n = O(n^2)$

VE² $\rightarrow O(n^2) * O((n^2)^2) = O(n^2) * O(n^4) = O(n^6)$

Os cidadãos e os supermercados corresponderam a 5% de n. Como pode ser visto no gráfico a seguir representados, calculámos a média do tempo para cada instância e obtivemos o gráfico correspondente aos valores obtidos experiencialmente.

N	Arcos	N	Arcos
250	623025,0	**1250**	15615125,0
500	2496050,0	**1500**	22488150,0
750	5619075,0	**1750**	30611175,0
1000	9992100,0	**2000**	39984200,0

Tabela 1 – Número de Arcos em função de N



A vermelho: Função polinomial de grau 6 aproximada aos pontos do eixo dos N's

Analisando o gráfico, concluímos que a nossa solução aproxima-se à análise teórica previamente feita, uma vez que a função obtida (segmento de pontos a azul) é, aproximadamente, uma função polinomial de grau 6 (segmento de pontos a vermelho).