# ASA - Relatório 2

Marisa Roque - 76653

23 de Abril de 2014

## Introdução

Foi apresentado um problema que consistia em modelar um sistema para o controlo de multidões dado um mapa de uma cidade com pontos onde as pessoas se podem concentrar e ligações entre esses pontos. Este modelo tem que garantir a não existência de comunicação entre um conjunto de pontos críticos. Para solucionar e para garantir a segurança da cidade as forças de segurança podem barrar algumas da ligações, de modo a evitar que seja iniciado um motim, minimizando os custos desta operação.

## Descrição da solução

Este problema pode ser modelado com um grafo, em que os pontos onde as pessoas se podem concentrar serão os vértices do grafo e as estradas entre estes pontos serão os arcos do grafo. O problema exposto desta maneira dá origem a um grafo não dirigido.

A passagem de pessoas entre os vários pontos da cidade constitui um fluxo de pessoas, o que em teoria de grafos equivale a um problema de fluxos. No entanto, este requer que os arcos entre os vértices possuam uma capacidade especificada e que o grafo seja dirigido. O segundo problema é resolvido criando um par de arcos anti-paralelos para cada ligação entre pontos.

Para o primeiro problema, como não é especificado a capacidade de cada estrada, assume-se que todas terão uma capacidade equivalente. Como a capacidade num problema de fluxo de grafos é meramente relativa, escolheu-se o valor unitário para todos os arcos.

O objectivo deste projecto é descobrir o valor mínimo de estradas a cortar pelas forças de segurança, ou seja descobrir o valor de corte mínimo de um grafo. Na teoria de grafos, o corte mínimo de um grafo é equivalente ao fluxo máximo do mesmo, pelo Teorema do Fluxo Máximo e Corte Mínimo [1].

O algoritmo clássico para determinar o fluxo máximo de um grafo, entre um vértice de origem $s$ e um vértice de destino $t$, é o método de Ford-Fulkerson. Este método considera duas ideias importantes em problemas de fluxo: redes residuais e caminhos de aumento. Com a condição inicial de um fluxo nulo, o método irá repetir enquanto existir um caminho de aumento na rede residual entre $s$ e $t$. Em cada iteracção, o fluxo ao longo desse caminho é aumentado com o mínimo das capacidades da rede residual ao longo do mesmo.

O método usado neste projecto é o Edmonds–Karp, que é uma variante do método Ford-Fulkerson, em que os caminhos de aumento são escolhidos usando um método de pesquisa em largura (BFS) na rede residual. A pesquisa dos caminhos de aumento desta forma garante que o caminho encontrado seja sempre o caminho mais curto, que por sua vez garante uma ordem de grandeza do tempo de execução do algoritmo de $O(VE^2)$, sendo $V$ os vértices e $E$ os arcos.

A solução para o problema apresentado foi implementada em ANSI C, aproveitando algum código de representação de grafos do projecto anterior.

Na solução apresentada, a representação do grafo foi efectuada com um vector de listas ligadas, denominada lista das adjacências. Esta representação assenta em três tipos definidos. Um tipo *Vertex* que é um simples inteiro, uma estrutura *Node* para ser usada nas listas ligadas, onde está contida a informação das capacidades e fluxos de cada arco e a terceira estrutura *Graph* que contêm o vector das listas das adjacências e o seu tamanho.

As principais funções do programa apresentado que merecem destaque são:

```c
Graph GRAPHresidual(Graph G);
Vertex *bfs(Graph G, Vertex s, Vertex t, int *size, int *gray_size);
int ford_fulkerson(Graph G, Vertex s, Vertex t);
int main(int argc, char *argv[]);
```

A função *GRAPHresidual* recebe uma estrutura que representa o grafo que se pretende resolver, $G$, e retorna um nova estrutura que representa a sua rede residual, $G_f$. Internamente à função, são determinadas as capacidades residuais da rede residual a partir do fluxo e capacidade do grafo $G$ e filtram-se apenas as que são positivas.

A função *bfs* recebe um grafo $G$, um vértice de origem $s$ e um vértice de destino $t$ e devolve um vector de vértices que representam o caminho de $s$ até $t$. Os argumentos *size* e *gray_size* são usados para retornar, via referências, o tamanho do vector do caminho e o número de arcos que a pesquisa não conseguiu visitar entre $s$ e $t$.

A lógica desta função consiste em percorrer todos vértices do grafo, marcando-os como não-visitados, em visita actual e visitados. Os vértices são percorridos recorrendo a uma fila, com o princípio FIFO (first in, first out), onde se vão colocando todos os vértices adjacentes do vértice em visita actual. A função termina quando atingir o vértice de destino $t$ ou quando a fila ficar vazia.

A função *ford_fulkerson* recebe a estrutura que representa o grafo $G$ e os vértices de origem e destino $s$ e $t$, respectivamente. O valor de retorno da função é o valor de corte mínimo do grafo, para o par origem-destino fornecido.

A função começa por inicializar a zero o fluxo de todo o grafo e inicia um ciclo em que vai calcular a rede residual, determinar um caminho de aumento entre $s$ e $t$ na rede residual. Se o caminho existir, vai determinar a capacidade residual mínima ao longo desse caminho e vai aumentar o fluxo do grafo com o valor da capacidade residual calculado. Se o caminho não existir, vai retornar o valor *gray_size* que é devolvido pela função *bfs*. Este valor corresponde ao número de arcos que a pesquisa não conseguiu transpor para atingir $t$ que equivale ao valor do corte mínimo e retorna-o.

A função *main* é o início do programa, como todos em C, pelo que os seus argumentos de entrada bem como o valor de saída não necessitam de explicação. A função começa por ler do *stdin* as linhas que correspondem ao grafo e constrói o objecto que o representa. Segue-se a leitura das $h$ linhas que contêm o conjunto dos pontos críticos a ser avaliados pelo programa e por cada uma determina-se o seu corte mínimo. O cálculo do corte mínimo de cada linha corresponde ao mínimo dos cortes da combinação $C_2^k$ dos $k$ pontos críticos, calculado com recurso à função *ford_fulkerson*, uma vez que a ordem dos pontos não afecta o cálculo e os pontos não se repetem.

## Análise teórica

Este programa executa em tempo linear ao tamanho do problema. A complexidade temporal de cada função do programa:

- main $O(hn^2VE^2)$
- leitura do input $\Theta(E)$
- conjuntos de pontos críticos $O(hn^2VE^2)$
- determinar corte mínimo para conjunto de pontos críticos $O(n^2VE^2)$
- Ford-Fulkerson $O(VE^2)$
- Rede Residual $O(E)$
- BFS $O(V+E)$

em que $V$ são os vértices, $E$ são os arcos, $n$ o número de pontos críticos no conjunto e $h$ o número de conjuntos de pontos críticos. Assim, a complexidade final do programa é $O(hn^2VE^2)$.

# Resultados da avaliação experimental

Esta solução passa com sucesso nos 16 testes presentes no sistema de testes automáticos Mooshak.

Foi verificado experimentalmente, que o tempo de execução varia com as variáveis definidas atrás, $V$, $E$, $n$ e $h$. Ou seja, quando se aumenta, por exemplo o número de pontos críticos de um conjunto, o tempo de execução aumentava, assim como para o número de conjuntos a determinar e o número de arcos e vértices.

# Referências

[1] Thomas H. Cormen, *Introduction to Algorithms.* Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Third Edition, September 2009.