Algoritmia e Desempenho em Redes de Computadores

3º Mini Projeto - *Connectivity in graphs*

Bernardo Gomes, 75573

Tomás Falcato, 75876

3 de Dezembro de 2015

1 Descrição do problema

Neste mini-projeto pretende-se avaliar a conectividade de uma rede representada por um grafo, ou seja, determinar qual o número mínimo de nós que ao retirar não permitem que o grafo esteja ligado.

Numa primeira fase, dado um par de nós do grafo, calcula-se o número mínimo de nós que é necessário retirar para que não haja nenhum caminho a ligar o par especificado.

Posteriormente, fez-se uma análise mais detalhada da rede, repetindo o processo anterior para todos os pares de nós de forma a armazenar a informação do número de nós que foi necessário retirar para cada par. Com a informação anterior, torna-se possível calcular a probabilidade cumulativa do número de nós a separar um par de nós.

Por fim, é verificado qual o número mínimo de nós que previne o grafo de ser conexo e é mostrado ao utilizador quais são os identificadores dos mesmos, por forma a que a informação do programa possa ser facilmente comprovada.

2 Abordagem ao problema

Sendo do nosso conhecimento que o número mínimo de nós que separa a fonte do destino é igual ao número máximo de caminhos independentes entre a origem e o destino, é de notar que o problema em causa é bastante semelhante à determinação de quais as arestas de um grafo que previnem que este seja conexo. No caso anterior, o problema seria reduzido a um problema de fluxos. Ora recorrendo à técnica de *vertex splitting* será possível resolver o problema de forma semelhante.

Ao dividir cada nó em dois, define-se "nó -" como o nó que irá receber todas as ligações, que o nó respetivo do grafo inicial receberia, entre os restantes e ele mesmo e que apenas tem ligação ao nó com o mesmo identificador que ele ("nó +"). Define-se "nó +" como o nó com o mesmo identificador que o nó que lhe deu origem, mantendo este as ligações entre este nó e todos os outros, que o nó respetivo do grafo inicial teria, recebendo apenas a ligação do respetivo "nó -".

Colocando as arestas que ligam as extremidades "-" e "+" de cada nó com capacidade "1" e todas as outras arestas, que já pertenciam ao grafo inicial, com capacidade infinita, aplicando o algoritmo de *Ford-Fulkerson*, é-nos possível resolver o problema enunciado.

3 Função *ford_fulkerson*

Apesar da referência ao algoritmo com o mesmo nome da função, o nosso código é uma adaptação do algoritmo em questão.

Tendo todas as arestas do grafo dado no ficheiro capacidade infinita e todas as arestas "internas" de cada nó capacidade "1", não foi necessário preocuparmo-nos com as atualizações dos fluxos. Apenas é necessário atualizar a rede residual no que respeita à inversão do sentido da aresta que liga os nós $-$ e $+$ de cada nó pertencente a dado caminho descoberto, garantindo assim que todos os caminhos descobertos em cada iteração são independentes.

O pseudo-código da função será o seguinte:

Result: number of nodes that separate the graph and its identifiers
see if there's a path from source to destination;
while *there's a path from source to destination* **do**
  invert edge link between - and + node in each original node from path;
  see if there's a path in new residual network;
end
see in discovered vector which nodes were only half discovered and count them;
if *counted nodes are less than previous counts* **then**
  pass to a string which nodes prevented the graph from being connex;
end

Algorithm 1: Função *ford_fulkerson*

Para a realização desta função, realizámos sucessivas BFSs por forma a verificar se existia um caminho independente da origem para o destino. Esta função, irá retornar um vetor *discovered* que indica quais os nós que foram descobertos e um vector *parent* que indica através de que nó o nó em causa foi descoberto.

Através desta informação, a função *path* irá reconstruir o caminho da BFS começando pelo destino. Se conseguir chegar à origem significa que o algoritmo conseguiu identificar um caminho independente dos descobertos anteriormente pelo que é necessário alterar novamente a rede residual.

Quando a função indica que não existe caminho entre os nós especificados, significa que se chegou a uma situação semelhante à ilustrada na figura 1, em que o nó mais à esquerda é a origem e o mais à direita o destino. Nesse caso, nos nós da fronteira, onde se verifica qual o corte mínimo, a posição do vetor *discovered* irá conter informação que apenas os nós − foram descobertos ("*half discovered*").

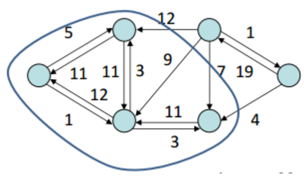

Figura 1

Este resultado deve-se à sucessiva inversão de arestas "interiores", até que a dado ponto seja impossível chegar desde a origem ao destino. Os nós "*half discovered*" serão os nós necessários para retirar para que o não seja descoberto nenhum caminho da origem para o destino.

Está também nesta função a terceira parte do projeto, em que nos é pedido para calcular a conetividade do grafo e fornecer os nós que separam o grafo. Esta parte de código encontra-se contemplada no *if* final da função, em que, caso o número de nós que é necessário retirar para separar um nó de outro seja inferior ao número de nós da iteração anterior, atualiza-se a *string* que contém os indentificadores desses nós, passando a conter apenas os que separaram o grafo nesta iteração. A conetividade é, simplesmente, o número de nós a retirar mínimo após corrido o programa para todos os pares de nós do grafo.

No que respeita à complexidade do algoritmo, tal como o algoritmo original, terá uma complexidade O(mf). No algoritmo original, m representa as arestas do grafo e f o fluxo máximo encontrado (para as inversões) que neste caso é unitário, ficando a complexidade simplificada para O(m).

4 Função *cumulative_statistics*

Esta função destina-se à segunda parte do projeto, em que é necessário calcular a distribuição cumulativa do número mínimo de nós a serem retirados. Não se irá apresentar pseudo-código dada a simplicidade da função.

Ao correr cada iteração da função *ford_fulkerson* é actualizado o vetor *node_statistics*, que é responsável pelo armazenamento do número de vezes que determinado número de nós é necessário retirar para separar dada origem e destino.

Quando o vetor fica com os valores finais relativos à rede dada no ficheiro, é calculado na função *cumulative_statistics* as ocorrências totais de cada número de nós a serem retirados. É, depois, impresso no terminal a divisão das ocorrências de cada índice do vetor pelo valor total das ocorrências, sendo omitida a posição 0 desse vetor. Esta posição representa o número de ocorrências em que se tentou obter o número de nós a retirar para separar dois nós diretamente ligados.

5 Considerações Finais

Com a resolução deste mini-projeto conseguimos compreender melhor o algoritmo de *Ford Fulkerson* que, apesar de adaptado, esteve bastante presente na nossa implementação, bem como algoritmos previamente estudados como a *BFS (Breadth First Search)* e, finalmente, o teorema do *max flow min cut*, dado recentemente nas aulas teóricas.

Existem no entanto, algumas limitações do programa, especificadas no enunciado, como a existência de um máximo de 100 nós com índices entre 0 e 99. Assumimos também que não pode existir nó 1 se não existir nó 0, pois apesar de o programa correr corretamente e o número de nós a retirar ser correto, as estatísticas não irão ser as corretas, visto que o programa assume que existe uma ligação do nó 0, que está em falta no ficheiro de entrada, para todos os outros nós do grafo.

De forma a tornar o programa mais *user friendly* criou-se um menu inicial em que o utilizador escolhe um de dois modos de funcionamento do programa:

1. No primeiro modo o utilizador escolhe uma origem e um destino, existentes no grafo, o programa irá retornar o número de nós necessários para os separar e qual esse conjunto de nós.
2. No segundo modo o utilizador não introduz quaisquer *inputs*, o programa irá correr, automaticamente, para todos os pares de nós, calculando a distribuição cumulativa, o número mínimo de nós a retirar para que o grafo seja conexo e, finalmente, o conjunto dos nós a retirar para que tal aconteça.

A complexidade geral do programa será assim, considerando o pior caso, O(n)[*complexidade da função "init_ vector"*]+O(n^2)[*repetição de "ford_folkerson" para todos os pares de nós do grafo*]*O(m)[*Complexidade da função "ford_folkerson"*] +O(n) [*Complexidade da função "cumulative_ statistics"*]. A complexidade final será entao O(mn^2), sendo m o número de arestas e n o número de nós.

Para testar *for correctness* o nosso código, utilizámos uma série de grafos, sendo os mais relevantes os apresentados nas figuras 1, 2, 3, 4 e 5.