



2º PROJETO:
STATIC ANALYSIS OF INTER-AS ROUTING

ALGORITMIA E DESEMPENHO EM REDES DE COMPUTADORES
DOCENTE: JOÃO SOBRINHO

MIGUEL RODRIGUES Nº 76176
PEDRO ESTEVES Nº 77060
GRUPO 6

1 Introdução

Neste segundo mini-projeto são explorados os conceitos de *routing*, fazendo uma análise estática de uma rede de *Autonomous Systems* (AS), simulando o funcionamento de uma internet e seguindo, de uma forma aproximada, as preferências e regras do protocolo BGP (*Border Gateway Protocol*). Através da leitura de um ficheiro de texto que indica as diferentes ligações entre os diferentes nós (ASes), temos como objetivo criar o grafo que a representa e realizar uma análise das suas propriedades.

Em primeiro lugar, é feita a verificação da existência de ciclos de clientes, isto é, se é possível chegar de um nó a si mesmo novamente, mas seguindo apenas ligações do tipo cliente (*Costumer*). Tal não pode acontecer numa rede como esta.

Depois, é feita uma análise quanto à sua conectividade: ver se é *Comercialmente Conexa*, isto é, se partindo de um nó é possível chegar a qualquer outro, respeitando as regras referentes aos caminhos possíveis. Numa internet tem de se verificar esta conectividade.

Por fim, é necessário verificar quais os caminhos eleitos por todos os nós para chegar a todos os outros nós e produzir estatísticas relativas a esta rede.

2 Funções e Estruturas

2.1 Estruturas

Para representar o grafo, ou seja, a rede de ASes e repetivas ligações, é usado um vetor de ponteiros para nós (*Nodes*), inicializado com 70000 posições, pois temos a informação de que o número não será superior a este (será de aproximadamente $2^{16} = 65536$), não sendo necessário fazer alocação dinâmica neste caso. Cada uma destas posições apresenta para um nó em específico, cujo identificador é um número (neste caso entre 0 e 70000) e que contem várias informações sobre a AS em questão, como o número de *Providers* (P), de *Peers* (R) e de *Costumers* (C). Em conjunto com as várias informações relevantes da AS, existem três apontadores para listas de adjacência, uma para cada tipo de ligação.

Estas listas de adjacência vão conter estruturas do tipo *NodeAdj* que guardam apenas o o ponteiro para a próxima estrutura (ou seja, o próximo vizinho do tipo repetitivo à lista de adjacência: C, P ou R) e o número do identificador da AS.

2.2 Leitura da Rede

De forma a criar a rede, é lido o ficheiro de texto correspondente linha a linha e é feita a alocação dos nós, alocando a AS que é a *tail*, inicializando os vários parâmetros desse nó (como o número de *peers*, de *providers*, etc) e a respetiva ligação. Como se sabe que as ligações são dadas de forma correcta e que para cada ligação P é dada uma C e que para cada ligação R é dada outra R, não há problema em alocar apenas uma das AS que aparecem na linha lida. Tendo em conta o tipo de ligação entre as ASes, é criado um nó no início da lista de adjacência correspondente da *tail* dessa ligação.

Para criar a ligação é usada a função *AddEdge* que, recebendo o nó de onde a ligação provém, cria essa aresta para a *head* da ligação e insere-a na respetiva lista de adjacência, tendo em conta o tipo de ligação e a informação do nó de destino. No final da alocação são percorridos todos os nós de forma a obter a lista de *Tier1s* (T1s), que será útil nas restantes funções.

Algorithm 1 AddEdge

```
1: type := Tipo de Ligação
2: Aloca nó auxiliar Aux
3: if type == 1 then
4:   Ligação C
5:   No->N_Costumers ++;
6:   Adiciona Aux a ListaAdj_C
7: else
8:   if type == 2 then
9:     Ligação R
10:    No->N_Peers ++
11:    Adiciona Aux a ListaAdj_R
12:  else
13:    Ligação P
14:    No->N_Providers ++
15:    Adiciona Aux a ListaAdj_P
16:  end if
17: end if
```

Para que não existam problemas de memória são criadas as funções *FreeGraph* e *FreeAdjList*, que vão percorrer todos os nós do grafo e todas as listas de adjacência desses nós, libertando-os. Assim, essa libertação tem uma complexidade de $\mathcal{O}(n+m)$, onde n são os nós e m as arestas. Também a inserção terá esta complexidade.

2.3 Verificação de Ciclo de Clientes

Para fazer a verificação da existência de ciclos de clientes, são usadas pesquisas em profundidade (DFS)

- *Depth First Search*), partindo de cada um dos *Tier1s* e usando apenas ligações *Costumer*. Quando uma AS for visitada, é usada uma variável (*visit*) que não permite que seja visitada outra vez, pois se já foi visitada e não voltou a si mesma então não faz parte de um ciclo, logo não é necessário voltar a entrar nesta AS. Por outro lado, sempre que uma AS faz parte do caminho que está a ser percorrido pela DFS, tem ativa uma variável (*ciclo*) que a identifica como parte do ciclo, se este existir. Assim, se se voltar a uma AS cuja variável *ciclo* seja 1, então é encontrado um ciclo e o programa pode encerrar. Sendo uma DFS, a sua complexidade é dada por $\mathcal{O}(n+m)$, onde n são os nós e m as arestas, mas nem todas as arestas serão percorridas, pois basta entrar numa AS uma vez e esta pode ter várias ligações até ela.

As funções que fazem parte deste algoritmo são *VerifyCycle*, que percorre a lista de T1s para chamar a função que percorre os nós (*DFS*) e verificar se existe ciclo ou não.

Algorithm 2 DFS

```

1: ciclo := 0
2: search := No->AdjList_C
3: while search != NULL do
4:   if ciclo == 1 then
5:     Há um ciclo. Sair
6:   else
7:     Marcar nó como visitado.
8:     No->ciclo = 1;
9:     DFS(Nó seguinte)
10:    No->ciclo = 0;
11:   end if
12:   search := search->next
13: end while

```

2.4 Verificação de Conectividade

Tendo em conta as regras descritas no enunciado, para que uma rede seja Comercialmente Conexa, é necessário que todos os nós *Tier1* sejam *peers* uns dos outros, se não existirão ligações que não serão possíveis (por exemplo, um T1 A só pode chegar a um T1 C se existir uma ligação entre eles do tipo R, visto que não podem utilizar várias ligações R nem usar ligações P após ligações C). No entanto, esta condição é, também, suficiente, pois todos os nós que não sejam *Tier1s* vão ter pelo menos uma ligação do tipo P (e respetiva ligação C no sentido contrário), até que eventualmente se chegue a um T1 que, sendo *peer* de todos os outros T1s, vai permitir que exista uma ligação até qualquer outro nó tipo. Se o nó em

questão não tiver uma ligação P, isso faz dele um T1, o que leva a que tenha ligações R com todos os outros T1s, pela condição inicial. Sabendo que não existem ciclos de clientes pelo algoritmo anterior, também não existirão ciclos de fornecedor que possam fazer com que o que é aqui descrito não se aplique.

Assim, basta verificar a existência de ligações *peer* entre todos os *Tier1s*. Tendo em conta que nos é dada a garantia de que as ligações no ficheiro de texto são apresentadas de forma correcta, basta verificar o número de ligações R que cada T1 tem com outros T1s (pois não vão existir ligações repetidas ou em falta, devido à garantia de que as ligações são dadas de forma correcta).

Como descrito no algoritmo seguinte, basta guardar a informação de todos os T1s (ou seja, número de ligações P é zero), contá-los e verificar que cada T1 tem N-1 ligações R com outros T1s, sendo N o número de T1s.

Algorithm 3 VerifyCommerc

```

1: T := Tier1 array
2: N := Number of Tier1s
3: for All T1 present in T do
4:   count := 0;
5:   for All Peers do
6:     if Peer is a T1 then
7:       count ++;
8:     end if
9:   end for
10:  if count != N-1 then
11:    Not all T1s are Peers
12:    End
13:  end if
14: end for

```

A complexidade deste algoritmo é de $\mathcal{O}(n^2)$, no entanto, n é o número de nós T1, que é um número bastante reduzido de nós, portanto não é problemático.

2.5 Procura de Caminhos

De forma a saber todos os caminhos para todos os nós existentes na rede, é usado um algoritmo que recebe um nó de destino e calcula qual o tipo de caminho que cada nó vai ter para esse nó de destino, tendo em conta as prioridades de caminhos. Tendo em conta que é um elevado número de nós, o cálculo de todos os caminhos é um processo exigente.

Sabendo que a rede é comercialmente conexa e que não existem ciclos de clientes, a primeira coisa a

fazer é identificar todos os nós que terão caminhos C para o destino, ou seja, os nós que são *Providers* do nó destino e os respetivos *Providers* e assim sucessivamente. De seguida, os nós que vão ter ligações do tipo R são os nós que são *peers* do nó de destino ou *peers* dos nós que já apresentam ligações do tipo C. Todos os restantes nós serão caminhos P, daí a inicialização ser feita para este tipo de caminho, poupando tempo com a decisão do caminho para estes nós.

Algorithm 4 Path

```

1: Path := 3 para todos os nós
2: for Todos os Providers do nó do
3:   Path := 1;
4:   for Todos os Peers do Provider do
5:     if Path != 1 then
6:       Path = 2
7:     end if
8:   end for
9:   Repetir para todos os Providers dos Providers
   até ao T1
10: end for

```

Este algoritmo é repetido N vezes, onde N é o número de ASes existentes, de forma a obter estatísticas a apresentar sobre a rede toda.

Usando este algoritmo, é possível saber quais são os tipos de caminhos a ser usados de forma eficiente, obtendo-os em menos de cinco minutos, sendo que o algoritmo apresenta uma complexidade de $\mathcal{O}(n + m)$, onde n é o número de nós com caminhos P e m as respetivas arestas entre eles e entre os seus pares. Para toda a rede, será $\mathcal{O}(n^2)$. No entanto, embora este algoritmo cumpra aquilo que é pedido no enunciado do projeto, para que fosse uma representação mais fiel do algoritmo BGP, ter-se-ia em conta, também, o número de hops de cada caminho, escolhendo sempre o caminho com menor número de hops (dentro do mesmo tipo de caminho). Esse algoritmo apresentaria uma complexidade e tempo de execução bastante superior.

2.6 Estatísticas

Através da análise feita tendo em conta apenas as preferências de caminho (C > R > P), pela análise da rede dada em *LargeNetwork.txt*, contam-se 1,933,668,702 caminhos ($\approx 1.9 \times 10^9$), sendo que estão divididos da forma apresentada na Tabela 1.

Tabela 1: Distribuição dos caminhos na rede fornecida

Path Type	# de Paths	% de Paths
C	11,828,240	0.611699
R	217,369,490	11.241299
P	1,704,470,972	88.147001

É esperado que assim seja, pois há um número muito maior de *stubs* e de nós intermédios que de T1s, portanto a maioria dos caminhos serão caminhos em que é necessário recorrer a ASes que sejam fornecedoras do nó em questão (88,15% das vezes).

Outra estatística interessante a apresentar e a analisar seria o número de ASes pelas quais um pacote tem de viajar para ir de um nó até ao outro e quais os tipos de ligações que tem de percorrer até chegar ao seu destino. Tal implementação poderia ter sido feita no algoritmo que informa qual o melhor caminho a percorrer pela preferência de tipo de caminho. Podia, ainda, ser acrescentada a escolha do caminho mais curto, de forma a ser uma representação mais fiel do BGP, acrescentando a condição de que, para além de escolher o caminho preferencial, escolher o que apresentar o menor número de saltos.

3 Conclusão

Neste segundo mini-projecto foi possível ganhar capacidade na análise de um problema real caracterizado por alguma complexidade, no qual foi importante saber correlacionar as especificações do problema com alguns algoritmos apresentados nas aulas teóricas referentes a grafos.

Tivemos a oportunidade de compreender de forma extensiva o algoritmo DFS e a sua implementação prática. Houve ainda a necessidade de alterar o comportamento típico da DFS de forma a aproveitar as suas capacidades de forma eficiente. Tendo impacto na performance do programa, reduzindo o tempo de execução e omitindo tarefas redundantes.