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

17/11/2017

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 fucnionamento de uma internet e seguindo, de uma forma aproximada, as preferências e regras do protoco 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 propriedas.
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 conetividade.
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 repetivo 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 na 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*, 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:      N_Costumers ++;
6:      Adiciona Aux a ListaAdj_C
7:  else
8:      if type == 2 then
9:          Ligação R
10:         N_Peers ++
11:         Adiciona Aux a ListaAdj_R
12:     else
13:         Ligação P
14:         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 profundiade (DFS - *Depth First Search*), partindo de cada um dos *Tier1s* e usando apenas ligações *Costumer*. Quando uma AS for visitada, é usada uma variáve (*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 aprte 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* for 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 entar numa AS uma vez e esta pode ter vários 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 BLAdhjskd

```
1: FALTA ESTE
```

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* (T1) 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 um ligação entre eles do tipo R, visto que não podemutilizar 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 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

texto bla
bla
bla

Algorithm 4 gjfdksl

```
1: FALTA ESTE
```

2.6 Estatísticas

Estatisitcas BlaBlaBla

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 número de saltos.

3 Conclusão