Algoritmia e Desempenho em Redes de Computadores

1º Mini Projeto - *Forwarding traffic in the Internet*

Bernardo Gomes, 75573 Tomás Falcato, 75876

9 de Outubro de 2015

1 Descrição do problema

Neste mini-projecto, pretende-se a implementação de diversas funções relacionadas com a transmissão de dados em pacotes consoante o seu destino.

Nesta fase, apenas se desenvolveu funções de conversão de *forwarding table* para *binary tree*, desta última para *binary2-tree*, adição de novos prefixos, remoção de prefixos e impressão de uma árvore binária no formato de *forwarding table*.

2 Função *ReadTable*

Nesta função o programa deverá ler de um ficheiro os prefixos de uma tabela e os correspondentes *next hops* e criar a árvore binária correspondente. Como tal, a função irá ler do ficheiro um prefixo de cada vez, adicionando-o à árvore com o *next hop* correspondente.

O pseudo-código da função será o seguinte:

```
root node creation();
while there are lines in the document do
    get_table_line();
    AddPrefix();
end
```

Algorithm 1: ReadTable

Desta forma, sendo o objectivo a inserção de todos os prefixos, será necessário fazer n vezes a chamada à função *AddPrefix*. A complexidade do algoritmo será assim O(n)O(get_table_line).

3 Função *AddPrefix*

Tal como descrito no enunciado, a função tem por objectivo a adição de um novo prefixo à árvore binária. Assumimos que esta função apenas é chamada quando a árvore binária do programa já foi previamente inicializada, ou seja, apenas é chamada dentro da função *ReadTable*, onde o *root node* é inicializado internamente, ou depois desta ser realizada.

Nesta função, testa-se se o prefixo a colocar na árvore é válido. No caso de ser, verifica se o prefixo é do *root* ou se é um endereço binário. Se for do *root*, copia as informações directamente para a estrutura apontada pelo ponteiro. Se não for, irá percorrer os nós da árvore correspondentes (criando os não existentes no caminho), copiando os valores pretendidos na estrutura correspondente ao prefixo indicado.

No pior caso, o algoritmo implementado terá de percorrer a altura da árvore.

A complexidade irá ser O(Comprimento do prefixo).

```
if adress is invalid then
    return;
end
if adress is "*"(root) then
    root->next_hop:=next_hop;
    root->prefix:=prefix;
else
    auxiliar_node:=root_node;
    while all prefix nodes havent been visited yet do
        if bit is 0 then
            if there is no node then
                node creation();
            end
            auxiliar:=auxiliar->zero;
        else
            /*bit is one*/
            if there is no node then
                node creation();
            end
            auxiliar:=auxiliar->one;
        end
        auxiliar->next_hop:=next_hop;
    end
end
```

Algorithm 2: AddPrefix

4 Função *DeletePrefix*

Nesta função, assumimos os seguintes factos: o comando de *delete* do *root* deve ser considerado como inválido e no caso de se apagar um prefixo que seja "filho único"o "pai"também deve ser removido.

Assim, no início da função, esta faz chamadas recursivas a ela própria até atingir o prefixo pretendido, retornando com uma mensagem de erro no caso de este não existir. Após apagar o nó correspondente, a função, ao retornar 1, "avisa"o nó de cima de que o ponteiro para este será agora NULL. Este factor será bastante importante na medida em que a condição de *free* baseia-se no facto de os dois ponteiros para estruturas "filho"serem NULL.

5 Considerações finais