Algoritmia e Desempenho em Redes de Computadores

2º Mini Projeto - *Inter-domain routing*

Bernardo Gomes, 75573

Tomás Falcato, 75876

12 de Novembro de 2015

1 Descrição do problema

Neste mini-projecto, pretende-se a implementação de diversas funções relacionadas com o tipo de rota comercial eleita de cada nó de uma dada rede para um destino eleito.

Assim, a função *Dijkstra* elege o tipo de rota e calcula o número de saltos necessários para chegar ao destino.

O cálculo das estatísticas para as diferentes rotas e para os números de saltos até ao destino, é feito depois da função anterior ser corrida para todos os destinos possíveis no grafo. Este é feito pelas funções *paths_statistics* e *number_hops_statistics*.

2 Abordagem ao problema

Decidimos utilizar um algoritmo baseado no algoritmo de *Dijkstra* para a computação das rotas a utilizar dos diversos nós para um destino eleito. Como tal, foi necessário que o programa interpretasse os dados de entrada provenientes de um ficheiro de texto por forma a ficar com a topologia da rede. A topologia é armazenada numa lista de adjacências.

Por questões de tempo de resolução do problema e tendo sido assegurado que o número de nós dos grafos não excederia o valor máximo de alocação de memória, a lista de adjacências criada tem por base um vector de ponteiros em que cada posição contém uma lista para as adjacências de cada nó.

Sendo as estatísticas calculadas depois de cada nó saber os caminhos a escolher tendo por base todos os destinos possíveis, após a leitura do ficheiro, o programa procede à realização da nossa função *Dijkstra* para todos os destinos, no fim de cada iteração atualiza-se o número de rotas utilizadas de cada tipo e o número de saltos utilizados para chegar ao destino. No fim de todos os cálculos são então feitas as estatísticas, sendo posteriormente apresentadas ao utilizador.

3 Função *Dijkstra*

Este algoritmo apesar de se basear no de *Dijkstra* é uma adaptação.

Neste programa temos um vector *heap_place* que que contém a posição instantânea de cada nó no *heap*. O vector *node_distance* contém o tipo de caminho para o nó corresponde à sua posição mais um, isto deve-se ao vector começar em zero e termos considerado que não existe identificador zero. O vector *node_hops* contém o número de saltos necessários até um destino para o nó corresponde à sua posição mais um.

Em cada iteração do algoritmo criado, para retirar o nó pretendido utilizaram-se *heaps*, de forma a melhorar a *performance* do programa para ficheiros de dimensões elevadas. Uma alternativa seria percorrer uma lista com os nós todos, em busca do que tivesse a rota mais favorável, no entanto a complexidade seria O(n), em vez de O(1).

A condição de substituição no vector *node_distance* em vez de ser o tradicional relaxamento de arestas de menor caminho passa a ser um relaxamento de arestas, em busca da melhor rota, não sendo estas somadas, mas considerando o pior caso, que irá definir a rota. Sempre que um valor do vector que controla as rotas é mudado, é necessário alterar a ordem dos nós dentro do acervo, usando a função *FixUp*. Tradicionalmente, utiliza-se a função *Heapify*, no entanto, devido ao elevado número de nós e ao facto de estarmos a utilizar o algoritmo para cada um deles, o *FixUp* reduz, substancialmente, o número de operações sobre o acervo. No entanto é necessário um compromisso de memória gasta *versus* tempo, sendo o tempo o facto preponderante no cálculo das rotas de *AS's* optámos pela utilização de um vector que guarda a posição de cada nó no acervo.

As funções *NewHeap*, *FixUp*, *Insert*, *RemoveMax* e *HeapEmpty* são adjacentes ao uso de *heaps* e, por isso, não achámos necessária a sua explicação.

Relativamente à complexidade desta função, sendo esta composta pelas funções *Initializa_distance*, *Heap_empty*, *Remove_max*, um ciclo *for* a percorrer as adjacências de um nó, e *Fix_up*, a complexidade será dada por: O(N)*O(log(N)) + O(M)*O(2log(N)) + O(N) = O(N)*O(log(N)) + O(M)*O(log(N)). Existindo mais arestas do que nós, a complexidade será O(M*log(N)) onde M é o número de ligações e N o número de nós do grafo.

O pseudo-código da função apresentada é o seguinte:

```
Result: Vector with flags of paths and number of hops
Initialize variables;
if there are nodes then
    Create heap;
    Initialize node distances, node hops and heap;
    while heap is not empty do
        Remove heap root;
        if root distance!=infinity then
            for each link do
                if link unseen then
                    if route through new link is more or equally favorable
                    than the stored route and route is usable then
                        if no consecutive peer links then
                            Stored route = New route;
                            if reaches destiny node in less hops then
                                Hops of new link = Hops of heap root+1;
                            end
                            FixUp heap in position of changed node;
                        end
                    end
                end
            end
        end
    end
    Invert flags of paths;
else

end
```

Algorithm 1: *Dijkstra*

4 Estatísticas

Para o cálculo das estatísticas pedidas, utilizámos as variáveis *ones*, *twos*, *threes* e o vector *stat_hops*, que estão constantemente a ser atualizados no fim de cada iteração, recalculando o número de rotas *customer*, *peer*, *provider* e o número de vezes que se chega ao destino em i saltos, sendo i o índice do vector *stat_hops*.

4.1 Função *paths_statistics*

O pseudo-código da função apresentada é o seguinte:

Result: estatísticas das rotas utilizadas (complexidade O(1))
stat_customer=ones/total_routes;
stat_peer=twos/total_routes;
stat_provider=threes/total_routes;
stat_unusable=unusable/total_routes;

Algorithm 2: *paths_statistics*

4.2 Função *number_hops_statistics*

O pseudo-código da função apresentada é o seguinte:

Result: estatísticas para cada número de saltos (complexidade O(1))
counts total number of hops;
for *each hop number* **do**
| prints ocurrencies/total;
end

Algorithm 3: *number_hops_statistics*

5 Considerações finais

No início do programa, este lê duas vezes o ficheiro que contém as ligações do grafo. Uma para saber qual o maior identificador dos nós e outra para armazenar o grafo sob a forma de lista de adjacências, já depois de a memória ter sido alocada (O(2L)=O(L), sendo L o número de linhas do ficheiro).

Posteriormente, são descobertas os tipos de rotas de todos os nós do grafo para todos destinos, ou seja, é corrido o algoritmo da nossa função *Dijkstra* N vezes. Para cada iteração, é contado o número de vezes que se utiliza cada tipo de rota e o número de saltos que foram necessários para chegar ao destino (O(2N) = O(N)).

Finalmente são feitas as estatísticas dos valores pretendidos (cálculo das probabilidades de cada tipo de rota e do número de saltos para chegar ao destino) cujas funções têm complexidade O(1).

O nosso programa tem assim complexidade O(L + NM*log(N) + 1), ou seja O(NM*log(N)).