João Lopes 90732
Tomás Gomes 90782

# Relatório - 1º Projeto ASA

## 1- Introdução

O 1º Projeto de ASA é sobre uma rede, constituída por routers. O programa que foi desenvolvido recebe como input a representação de uma rede, num grafo, com vértices(routers) e arestas(ligações) e, deve ter como output o numero de sub-redes da rede global, bem como os seu identificadores(id) e os routers que quando removidos causa o aumento do numero de sub-redes, bem como o tamanho da maior sub-rede gerada por essa operação de remoção dos routers que causam o aumento das sub-redes.

O input é:

- Uma linha com o número de routers da rede N ($N \geq 2$).
- Uma linha com o número de ligações entre routers na rede M ($M \geq 1$).
- Uma lista de M linhas, em que cada linha contém dois inteiros u e v (separados por um espaço em branco) indicando que há ligação entre u e v.

O output é:

- Uma linha com um inteiro R que denota o número de sub-redes.
- Uma linha com R valores inteiros separados por espaços em branco com os identificadores das R sub-redes. Nesta linha, os R valores têm que aparecer ordenados de forma crescente.
- Uma linha com um inteiro C que denota o número de routers que quebram uma sub-rede. Ou seja, o número de routers que se removidos individualmente, resultaria no aumento de sub-redes.
- Uma linha com um inteiro m que denota o número de routers da maior sub-rede resultante da remoção de todos os C routers que quebram uma sub-rede.

## 2- Descrição da solução

Como linguagem de programação decidimos usar C. C tem nível baixo o suficiente para ser eficaz e eficiente na implementação deste projeto, onde tempo e uso de memoria importam.

Como abstração face ao problema descrito, consideramos usar um nó do grafo como representação de um router e para cada ligação um “nó falso”(struct igual mas só com o int id;). Assim, temos um vetor de nós e cada nó liga-se a uma lista ligada de “nós falsos” que representas as ligações do nó.

O objetivo é encontrar pontos de articulação e árvores descobertas na DFS. Assim, temos como solução sugerida do nosso problema:

1) Construção do grafo a partir de inputs lidos no stdin.

2) Executar uma DFS, começando no nó de maior id(para que quando uma árvore é descoberta o id do nó inicial, raiz, seja o maior). Com isso, as árvores encontradas equivalem às sub-redes e, os seus identificadores também são descobertos.

Enquanto a DFS executa, a cada visita a um nó filho os nós que são pontos de articulação são identificados assim que possível e são marcados. Esses nós são os routers que se removidos individualmente, aumenta o numero de sub-redes. Este passo dá-nos os três primeiros outputs.

João Lopes 90732
Tomás Gomes 90782

3) Executar DFS uma segunda vez. Desta vez os pontos de articulação estão identificados(feito na DFS anterior). Corre-se então a DFS ignorando os pontos de articulação, ou seja, como se o grafo fosse o grafo menos esses pontos. As sub-redes resultantes de tal são descobertas e a maior delas é guardada(o tamanho da mesma). Assim, temos o 4º output.

## 3- Análise Teórica

(Contextualização):

```
int main(){
---------------------- PASSO 1 ----------------------
        lerGrafo();
---------------------- PASSO 2 ----------------------
        DFS();
        printf(Numero de sub-redes);
        para cada (id das sub-redes){
                printf(id da sub-rede);
        }
        para cada (router){
                se(o router for ponto de articulação){
                        Numero de pontos de articulação++;
                }
        }
        printf(Numero de pontos de articulação);
---------------------- PASSO 3 ----------------------
        Tamanho da maior sub-rede = 0;
        DFS();
        printf(Tamanho da maior sub-rede);
}
```

Guarda representação do grafo em memória;

Outputs 1, 2 e 3;

Output 4;

### Passo 1: Construção do grafo

```
        void lerGrafo(){
                para i=0 até numero lido de routers{
                        adicionarRouterAoGrafo[i];
                }
                para i=0 até numero lido de ligações{
                        adicionarAdjacencia(u, v); //(Adiciona ligação de u → v e v->u)
                }
        }
```

Tempo: O(V)
Memória: O(V)

Tempo: O(E)
Memória: O(E)

Complexidade total do passo 1:
- Tempo: O(V+E)
- Memória: O(V+E)

João Lopes 90732
Tomás Gomes 90782

Passo 2 e 3: DFS, marcação de pontos de articulação.

```
void DFS(){
        Tempo = 1;
        para cada i (um router), começando pelo fim de Graph[i]{
                se(tempo de descoberta do router == 0){
                        visitar(i+1);
                }
        }
}

void visit(id){
        while(1){
                se(Graph[id]→next == NULL) break;
                filho = Graph[id]→next;
                se(tempo de descoberta do filho == 0){
                        visit(filho->id);
                }
        }
}
```

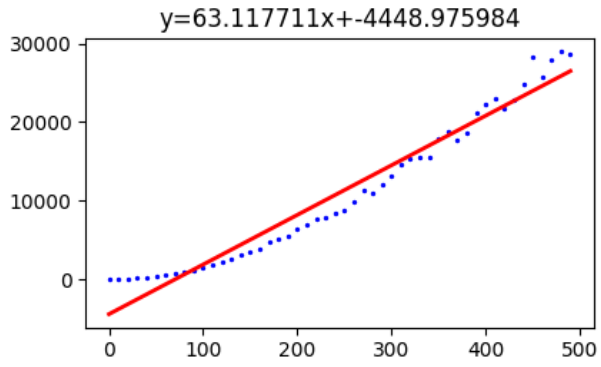
Tempo: O(V + E)

Assim, a complexidade do algoritmo é: O(V + E)

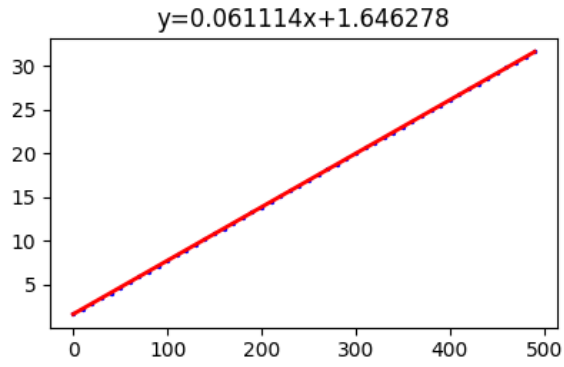
## 4- Avaliação experimental dos resultados

Foram gerados 50 inputs desde 1 router até 500000. Os resultados obtidos foram:

Tempo




Memória




Como esperado a complexidade é linear, O(V + E). É notável também que para grafos pequenos o problema é quase resolvido em tempo constante.