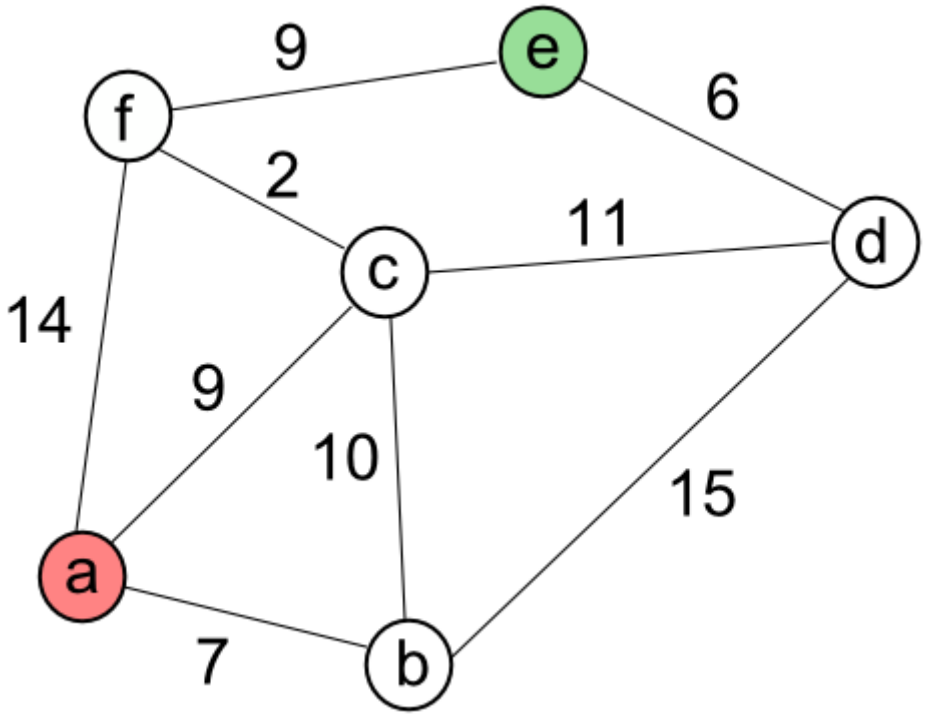
AEDs III - Trabalho 2:
“Diâmetro Máximo de um Grafo”



Autor: Vinícius Eduardo de Souza Honório - 2021.1.08.024

Sobre o Processo de Implementação

A ideia de calcular a distância entre cada par de vértices em um grafo não-completo parecia, a princípio, trabalhosa e complexa, mas se mostrou algo relativamente trivial ao encontrar-se um algoritmo capaz disso: o Algoritmo de Floyd-Warshall.

O restante da implementação foi relativamente trivial: algumas rotinas de lista encadeada foram separadas em função para melhor visibilidade, as arestas e as distâncias mínimas foram salvas em matrizes dinamicamente alocadas e a leitura de arquivo realizada duas vezes para registrar-se o número de arestas e poupar memória.

O código, por alguma razão, não funciona em meu computador pessoal; mas compila e retorna saída perfeitamente na máquina do Laboratório Acadêmico, na qual este foi desenvolvido e testado.

Sobre o Código

A ideia aqui utilizada na implementação do algoritmo solucionador do problema é relativamente simples: consistindo ela num laço triplo de repetições "*for*", utilizado para avaliar se a distância entre dois vértices **X** e **Y** é mais curto se percorrido através de um vértice intermediário **Z** ou não; este caminho alternativo é também tomado caso não haja caminho direto entre **X** e **Y**. Complementarmente: o vértice **Z** é também adicionado a uma lista de intermediários do caminho **XY**, isso através de uma Lista Encadeada de Índices.

Terminado, então, este processo e determinada a distância mínima entre cada um dos pares de vértices do grafo (isto se os dois vértices forem conexos): basta selecionar-se a menor dentre estas e, finalmente, exibir o resultado obtido.

Pseudocódigo

```
abre_arquivo("nome.txt")
n ← leitura_arquivo()
enquanto leitura_arquivo() ≠ VAZIO, faça
        a ← leitura_arquivo()
        b ← leitura_arquivo()
        distância(a,b) ← leitura_arquivo()
fimenquanto
fecha_arquivo("nome.txt")

para todo k, de 1 a n, faça
        para todo i, de 1 a n, faça
                para todo j, de 1 a n, faça
                        se distância(i,j) > distância(i,k) + distância(k,j), então
                                distância(i,j) ← distância(i,k) + distância(k,j)
                                se k ∉ lista[i][j], então
                                        lista[i][j] ← lista[i][j] ∪ {k}
                                fimse
                        fimse
                fimpara
        fimpara
fimpara
```

Saída Retornada de Cada Instância

Entrada: *n25.txt*
Tempo de Execução: 0.408s

```
n25
21 0 5 24
55
```

Entrada: *n50.txt*
Tempo de Execução: 0.472s

```
n50
22 0 8 9 11 27 29 30
97
```

Entrada: *n100.txt*
Tempo de Execução: 1.31s

```
n100
39 71
214
```

Entrada: *n250.txt*
Tempo de Execução: 46.676s

```
n250
131 0 15 37 39 88 126 201 245 152
515
```

Entrada: *n500.txt*
Tempo de Execução: 822.537.s

```
n500
46 0 11 17 424 191
1032
```

Entrada: *n750.txt*
Tempo de Execução: 5537.846s

```
n750
491 0 7 10 11 44 92 134 297 725
1628
```

Entrada: *n1000.txt*
Tempo de Execução: Desconhecido

```
n1000
30 … 355
2115
```

Observação: O último caso (arquivo *n1000.txt*) não pôde ser desempenhado até o fim por falta de tempo disponível, mas acredita-se que este ainda retornaria seu resultado completo e coerente. Foi então, em substituição, executada uma versão simplificada do código, que calculava a maior distância do grafo sem registrar caminho. O tempo de execução foi de 17.73s e o resultado retornado aquele, acima.