· U C ·

UNIVERSIDADE DE COIMBRA
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

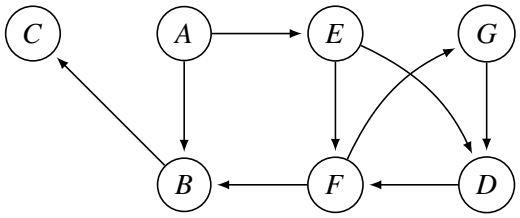
Estratégias Algorítmicas

Exame de Recurso – 28 de junho de 2022

Nome: Pedro Afonso Quintal Castro Nº de estudante: 2021198420

12 pontos no total, 2 horas, sem consulta.

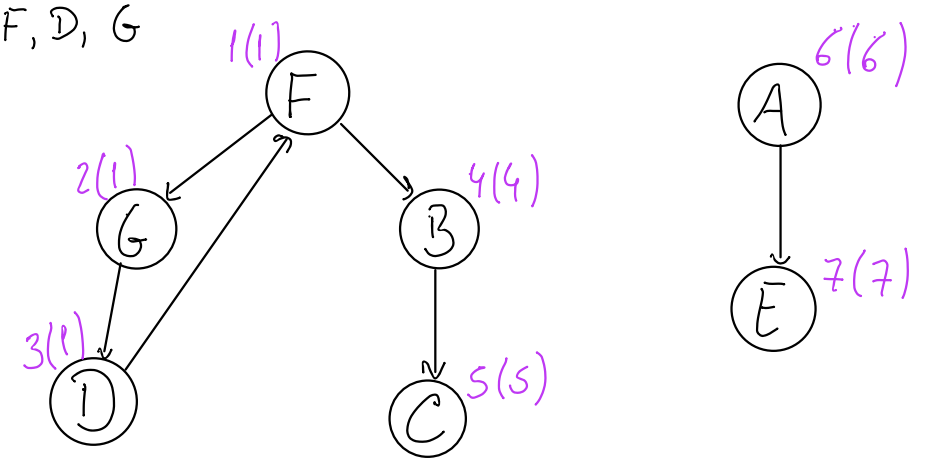
1. Considere o seguinte grafo dirigido.



(a) Encontre as componentes fortemente conexas do grafo seguinte recorrendo ao algoritmo de Tarjan. Reporte a árvore de procura em profundidade a partir do vértice F, escolhendo os vértices para a travessia de acordo com a ordem alfabética das etiquetas, e indique explicitamente os valores finais de `dfs` e `low` em cada vértice. Indique igualmente as componentes fortemente conexas, ordenadas pelo momento em que foram encontradas pelo algoritmo de Tarjan (na caixa). (2 pontos)

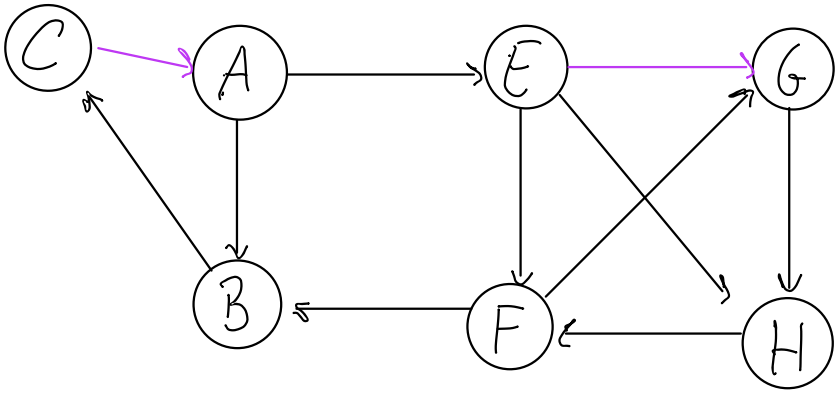
Componentes fortemente conexas:

F, D, G



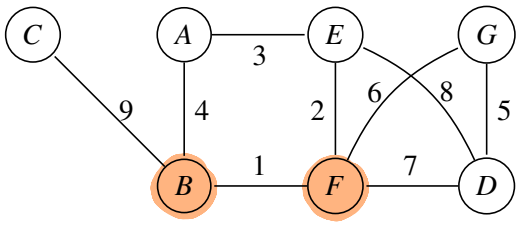
(b) Quais os arcos que deve adicionar ao grafo para que exista uma só componente fortemente conexa? Justique a sua resposta recorrendo ao algoritmo de Tarjan como fez no exercício anterior. Nota: Respostas com o menor número de arcos serão valorizadas. (2 pontos)

Arcos:

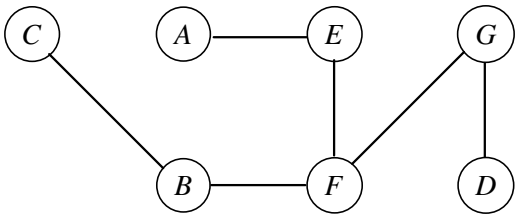


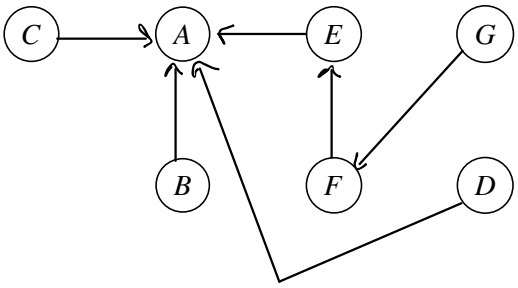
W	Source	Dest
1	B	F
2	F	E
3	A	E
4	B	A
5	D	G
6	F	G
7	F	D
8	E	D
9	C	B

2. Considere o seguinte grafo.



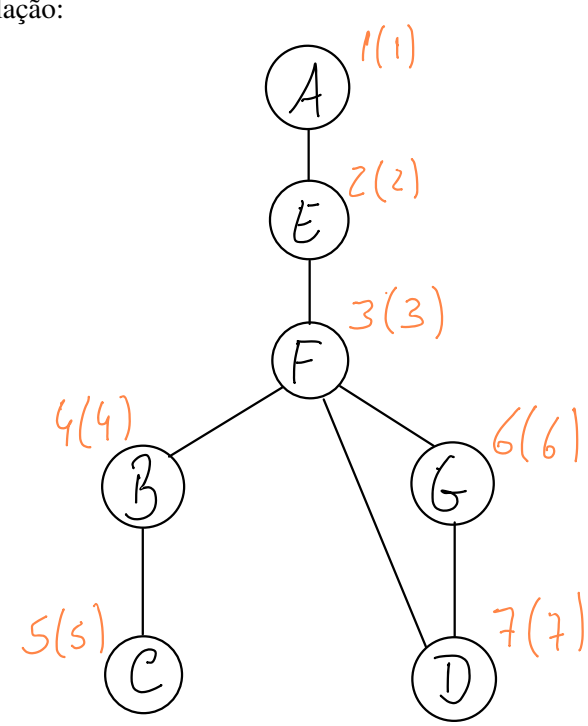
(a) Encontre a árvore geradora mínima na rede acima recorrendo ao algoritmo de Kruskal. Desenhe a árvore geradora mínima à sua esquerda e o grafo da estrutura de dados *union-find* (sem o passo de compressão de caminho) à sua direita. Quando necessário, ligue a raiz da árvore com menor altura à raiz da árvore com maior altura e, em caso de empate, escolha, como raiz, o nó que apresentar a etiqueta alfabeticamente menor. (2 pontos)





(b) Encontre os pontos de articulação no grafo anterior. Para justificação da sua resposta, reporte a árvore de procura em profundidade a partir do vértice A, escolhendo os vértices para a travessia de acordo com a ordem alfabética das etiquetas, e indique explicitamente os valores finais de `dfs` d `low` em cada vértice. Reporte igualmente os pontos de articulação ordenados pelo tempo em que foram encontrados durante a travessia em profundidade, e o critério utilizado para identificação de cada ponto. (2 pontos)

Pontos de articulação:



3. O seguinte pseudo-código permite calcular o número de Fibonacci para um determinado valor de n. Por que razão não é a escolha mais apropriada do ponto de vista de tempo computacional? (1 ponto)

Function $fib(n)$
  if $n = 0$ **or** $n = 1$ **then**
    return n
  else
    return $fib(n-1) + fib(n-2)$

A implementação recursiva ingénua do cálculo do número de Fibonacci é ineficiente devido aos cálculos repetidos, resultando em uma complexidade exponencial. É preferível usar programação dinâmica ou memoização para obter uma solução mais eficiente.

4. Considere o seguinte problema: Dado um conjunto A de n inteiros positivos cuja soma é S, descubra se é possível dividir A em três subconjuntos tal que a soma de cada subconjunto é exatamente $S/3$. Assuma que S é divisível por 3. Escreva o pseudo-código de uma possível abordagem para este problema. Discuta o tempo computacional e demonstre que está correta. (3 pontos)

```
function isPossibleSubset(A)
    n = tamanho do conjunto A
    soma = soma de todos os elementos em A
    soma_subconjuntos = [0, 0, 0]
    if soma % 3 != 0
        return false

    function buscaExaustiva(posição)
        if posição == n
            if soma_subconjunto[0] == soma_subconjuntos[1] ==
soma_subconjuntos[2] == soma/3
                return True
            else
                return False
        for i from 0 to 2
            if soma_subconjuntos[i] + A[posição] <= 3
                soma_subconjuntos[i] += A[posição]
                if buscaExaustiva(posição+1)
                    return True
                soma_subconjuntos[i] -= A[posição]
        return False
    return buscaExaustiva(0)
```

Para demonstrar que a abordagem está correta, precisamos analisar duas condições principais:

1. A soma total é divisível por 3: essa condição é verificada no início da função 'isPossibleSubset'. Se a soma total não for divisível por 3, retornamos 'False' imediatamente, pois não é possível dividir em três subconjuntos com soma igual.

2. A soma de cada subconjunto é igual a S/3: essa condição é verificada dentro da funçã 'buscaExaustiva', onde somamos os elementos em cada subconjunto e comparamos com a soma desejada. Se, em algum momemto, encontrarmos uma combinação que satisfaz essa condição, retornamos 'True'. Caso contrário, continuamos a explorar as outras combinações. Estas condições garantes que a abordagem está correta e retronará o resultado esperado para o problema.

O tempo computacional desta abordagem é exponencial, pois estamos explorando todas as possíveis combinações de elementos do conjunto A. A cada chamada recursiva da função 'buscaExaustiva', temos três opções para adicionar o elemento atual, o que resulta em uma árvore de recursão com três ramos em cada nível. O número total de chamadas recursivas será 3^n, onde n é o tamanho do conjunto A. Portanto, o tempo de execução será O(3^n).