· U · C ·

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

# Estratégias Algorítmicas

Exame Época Normal – 9 de junho de 2025

Nome: ______________________ Nº de estudante: __________

15 pontos no total, 2 horas, sem consulta.

1. A estrutura *union-find* utiliza o método $find(a)$ para identificar a raiz da árvore à qual o elemento $a$ pertence.

   a) Dado o seguinte pseudo-código que implementa uma variante desse método, e considerando o vetor $P = [2,0,4,2,4,2]$, indique qual será o conteúdo do vetor após a execução de $find(1)$. Assuma que $P$ está declarado como uma variável global e que seu primeiro elemento tem índice 0 (1.5 pontos).

   **Function** $find(a)$
   **if** $P[a] \neq a$ **then**
   $P[a] = find(P[a])$
   **return** $P[a]$

   P = [4, 4, 4, 2, 4, 2]

   b) O pseudo-código do método *find* na alínea *a*) é recursivo. Escreva o pseudo-código do mesmo método em versão iterativa. Indique a sua complexidade computacional no pior caso se o $P$ tiver $n$ elementos (2 pontos).

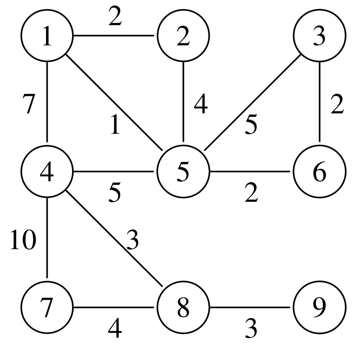
```
Function find(a)
  root = a
  while P[root] != root do
    root = P[root]

  current = a
  while P[current] != root do
    next = P[current]
    P[current] = root
    current = next

  return root
```

Também dá para fazer com stack

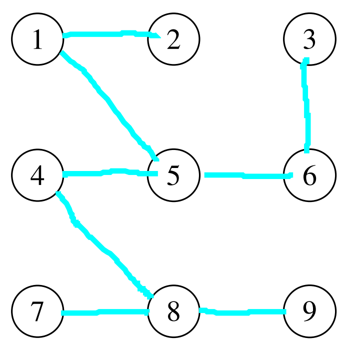
Worst Case: O(α(n))

α é a Ackermann function,
acaba por ser praticamente O(1)

2. Considere o seguinte grafo.



a) Desenhe o caminho mais curto do nó 9 a cada nó do grafo usando o algoritmo de Dijkstra, indicando a distância mais curta em cada nó. Assuma que pode ir em ambos os sentidos em cada aresta. Indique os nós visitados pela ordem que o algoritmo os seleciona (2 pontos).



Nós visitados:

b) Considere o problema de encontrar o caminho entre dois nós num grafo em que a distância máxima associada a um arco ou aresta nesse caminho é a menor possível. Que alteração no algoritmo de Dijkstra deve efetuar para encontrar esse caminho (1.5 pontos)?

Passo de relaxation:
- para uma edge (u, v) com peso w, a distância d[v] ao node v é atualizada com:
  d[v] = min(d[v], max(d[u], w))

Priority queue:
- passa a ser sorted por esta nova métrica

3. Considere a seguinte recorrência para $i = 0, \ldots, n$ e $c_i \geq 0$:

$$S(i) = \begin{cases} c_i & \text{se } i = 0 \text{ or } i = 1 \\ c_i + \min\{S(i-1), S(i-2)\} & \text{caso contrário} \end{cases}$$

a) Tendo em conta que o valor que pretende obter é retornado por $\min\{S(n-2), S(n-1)\}$, apresente o pseudo-código de um algoritmo de programação dinâmica descendente (*top-down*) que explore a recorrência acima para obter esse valor. (1.5 pontos)

```
Function S_top_down(i)
  if i == 0 or i == 1:
    return c[i]

  if dp[i] is cached:
    return dp[i]

  dp[i] = c[i] + min(S_top_down(i - 1), S_top_down(i - 2))
  return dp[i]

Function Solve(n, c)
  dp = array of size (n + 1)

  return min(S_top_down(n - 1), S_top_down(n - 2))
```

b) Apresente de seguida o pseudo-código de um algoritmo de programação dinâmica ascendente (*bottom-up*) que explore a recorrência acima para obter esse valor e discuta a sua complexidade computacional e espacial. (1.5 pontos)

```
Função Solve(c, n)
    if n < 2:
        return c[n]

    dp = array de tamanho n + 1
    dp[0] = c[0]
    dp[1] = c[1]

    for i from 2 to n:
        dp[i] = c[i] + min(dp[i-1], dp[i-2])

    return dp[n]
```

O(n) espaço e tempo

c) Discuta possíveis melhorias na sua abordagem apresentada na alínea b), com foco na otimização do uso de memória (1 ponto).

Substituir o array dp por duas variáveis
Atualizar as variáveis a cada iteração

(ou nem sequer usar dp)

Independentemente, ambas as soluções resultam numa complexidade espacial de O(1)

4. Considere uma matriz $M$ de tamanho $3 \times 3$ e um inteiro $k$, com $0 \leq k \leq 4$.

a) Escreva o pseudo-código de um algoritmo recursivo que imprima todas as configurações distintas possíveis de colocar $k$ símbolos x em posições adjacentes (cima, baixo, esquerda, direita) à posição central de $M$. Cada configuração deve conter exatamente $k$ símbolos colocados em posições distintas e adjacentes à posição central. Assuma que $k$ e a matriz $M$ estão declaradas como variáveis globais, e que existe um método $print(M)$ que imprime o conteúdo da matriz $M$. Adicione comentários ao seu código (3 pontos).

há muitas maneiras de fazer isto

```
adj_pos = [(0,1), (2,1), (1,0), (1,2)] // Cima, Baixo, Esquerda, Direita

Function colocar_simbolos(index, count):
  if count == k:
      print(M)
      return

  if index == 4:
      return

  (r, c) = adj_pos[index]
  M[r][c] = 'x'
  colocar_simbolos(index + 1, count + 1)
  M[r][c] = ' ' // Assume-se que a posição vazia é representada por um espaço.
  colocar_simbolos(index + 1, count)


Function solve():
  colocar_simbolos(0, 0)
```

b) Qual é a complexidade temporal da sua abordagem, em termos do número de chamadas recursivas, em função de $k$? Justifique a sua resposta (1 ponto).

O(1), há um número fixo de posições

Número total de chamadas na árvore binária de decisão:

Número Total de Chamadas: O número total de chamadas recursivas é a soma dos nós nesta árvore.

Nível 0: 1 chamada (colocar_simbolos(0,0))
Nível 1: 2 chamadas
Nível 2: 4 chamadas
Nível 3: 8 chamadas
Nível 4: 16 chamadas
O total de chamadas é 1+2+4+8+16=31.

O valor de k é usado apenas na condição de paragem (if count == k:)

6