Análise da Complexidade de Algoritmos Recursivos VI

Joaquim Madeira

29/04/2021

Sumário

- Recap
- Seleção do k-ésimo elemento: o Algoritmo Quickselect
- Cálculo do valor de um polinómio: o Método de Horner
- Multiplicação de matrizes: o Algoritmo de Strassen
- Sugestões de leitura

Recapitulação

Quicksort

- Ordenar o array de modo recursivo, sem usar memória adicional

- Particionar o conjunto de elementos, trocando de posição, se necessário

- Com base no valor de um elemento pivot

Quicksort

- Escolher o valor do element pivot

- Particionar o array
- Elementos da 1ª partição são menores ou iguais do que o pivot
- Elementos da 2ª partição são maiores ou iguais do que o pivot

- Ordenar de modo recursivo a 1ª partição e a 2ª partição

1ª versão

```c
void quicksort(int* A, int left, int right) {
  // Casos de base
  if (left >= right) return;

  // Caso recursivo
  // FASE DE PARTIÇÃO
  int pivot = (left + right) / 2;
  int i = left;
  int j = right;

  do {
    while (A[i] < A[pivot]) i++;
    while (A[j] > A[pivot]) j--;

    if (i <= j) {
      trocar(&A[i], &A[j]);
      i++;
      j--;
    }
  } while (i <= j);

  // Chamadas recursivas
  quicksort(A, left, j);
  quicksort(A, i, right);
}
```

Eficiência

- Todas as comparações são feitas na fase de partição !!

- **O(n log n)** para o melhor caso e o caso médio
- MAS **$O(n^2)$** para o pior caso !!
  - Muito raro, se escolhermos “bem” o pivot !!
  - Ou se gerarmos uma permutação aleatória do array dado

- Mais comparações do que o Mergesort !!
- MAS, na prática, é mais rápido do que o Mergesort !!

K-Selection
– Selecionar o k-ésimo elemento

K-Selection

- Dado um array com n elementos : A[0,…,n – 1]
- O array não está ordenado !!

- Se o array estivesse ordenado
- Qual seria o valor do elemento na posição de índice k ?
  - Min (k = 0) / Max (k = n – 1) / Mediano (k = n **div** 2)
  - Aplicação: **top k** elementos

- Possíveis soluções ?
- Complexidade ?

K-Selection

- Possíveis estratégias ?

- Eficiência ?

- Como usar o procedimento de partição do algoritmo Quicksort ?

K-Selection – Estratégia direta

- Ordenar por ordem não decrescente os n elementos

- Consultar o elemento na posição k

- Quanto tempo ?
  - 1.000.000 elementos / 10.000.000 elementos / …

K-Selection – Estratégia direta

- Ordenar por ordem não decrescente os n elementos

  $O(n^2)$ ou $O(n \log n)$

- Consultar o elemento na posição k

  $O(1)$

- Quanto tempo ?
  - 1.000.000 elementos / 10.000.000 elementos / …

K-Selection – Estratégia melhorada

- Copiar os primeiros (k + 1) elementos para um array S
- Ordenar o array S por ordem não decrescente
- Para cada um dos restantes (n – k – 1) elementos de A
    - Ignorar se maior ou igual que S[k]
    - Caso contrário, inserir ordenadamente em S
        - O elemento S[k] é expulso do array e substituído

K-Selection – Estratégia melhorada

- O que demora mais tempo ?

- Ordenar os primeiros (k + 1) elementos

  $O(k^2)$ ou $O(k \log k)$

- Para cada um dos restantes (n – k – 1) elementos
  - Ignorar se maior ou igual que S[k] **$O(1)$**
  - Caso contrário, inserir ordenadamente em S **$O(k)$**

K-Selection – Estratégia melhorada

- Ordem de complexidade ?

$$O(k \log k) + (n - k - 1) \times O(k) = O(n \times k)$$

- Encontrar o elemento mediano $\qquad O(n^2)$

- Quanto tempo ?
  - 1.000.000 elementos / 10.000.000 elementos / ...

K-Selection

- Será ainda possível fazer melhor ?

- O que poderemos usar dos algoritmos e estruturas de dados que conhecemos ?

- Será necessário manter o conjunto dinâmico de (k + 1) elementos completamente ordenado ?

K-Selection – Usar uma MIN-HEAP

- Transformar o array numa MIN-Heap com n elementos

- Efetuar (k + 1) operações deleteMin()

- O último elemento removido é o procurado

- Esta estratégia lembra-nos alguma coisa ?

K-Selection – Usar uma MIN-HEAP

- Transformar o array numa MIN-Heap com n elementos

  $O(n)$

- Efetuar (k + 1) operações deleteMin()

  $(k + 1) \times O(\log n)$

- O último elemento removido é o procurado

$$O(n + k \times \log n)$$

K-Selection – Usar uma MIN-HEAP

$$O(n + k \times \log n)$$

- Se k = $O(n / \log n)$ então $O(n)$
- Encontrar o elemento mediano $O(n \log n)$

K-Selection – MAX-HEAP “mais pequena”

- Copiar os primeiros (k + 1) elementos para um array S
- Transformá-lo numa MAX-Heap com (k + 1) elementos
- Para cada um dos restantes (n – k – 1) elementos de A
  - Comparar com o elemento do topo da heap: S[0]
  - Se for menor, substituir S[0] e reorganizar a heap
- O elemento do topo da heap é o procurado: S[0]

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

6	**3**	**4**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

6	**3**	**4**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

6	**3**	**4**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

2	**3**	**4**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

2	**3**	**4**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

4	**3**	**2**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

4	**3**	**2**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

4	**3**	**2**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

1	**3**	**2**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

1	**3**	**2**

K-Selection – k = 2 – 3º elemento

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

3	**1**	**2**

K-Selection – MAX-HEAP “mais pequena”

- Criar a MAX-Heap com $(k + 1)$ elementos

$$O(k)$$

- Para cada um dos restantes $(n - k - 1)$ elementos
  - Comparar com o elemento do topo da heap: S[0] $O(1)$
  - Se for menor, substituir S[0] e reorganizar a heap $O(\log k)$

$$(k + (n - k) \times \log k) = O(n \times \log k)$$

Eficiência

- Versão 1 : Ordenar array de n elementos

  O(n x log n)

- Versão 2 : Manter ordenado array com (k + 1) elementos

  O(k x n)

- Versão 3 : MIN-Heap de n elementos com k apagamentos

  O(n + k x log n)

- Versão 4 : MAX-Heap de (k + 1) elementos com substituições

  O(n x log k)

Eficiência – Elemento mediano

- Versão 1 : Ordenar array de n elementos

  $O(n \times \log n)$

- Versão 2 : Manter ordenado array com (k + 1) elementos

  $O(n^2)$

- Versão 3 : MIN-Heap de n elementos com k apagamentos

  $O(n \times \log n)$

- Versão 4 : MAX-Heap de (k + 1) elementos com substituições

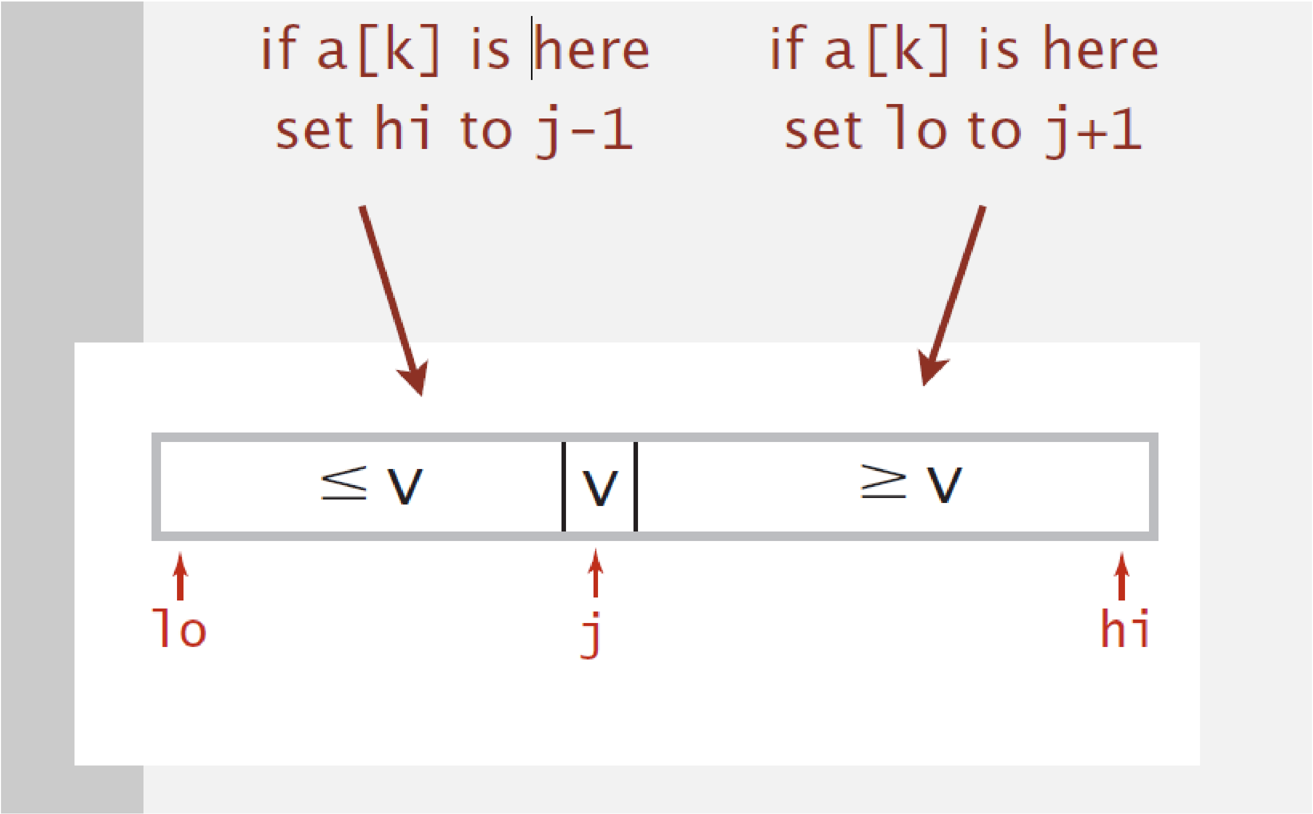
  $O(n \times \log n)$

K-Selection
– O algoritmo Quickselect

K-Selection – Algoritmo Recursivo de Partição

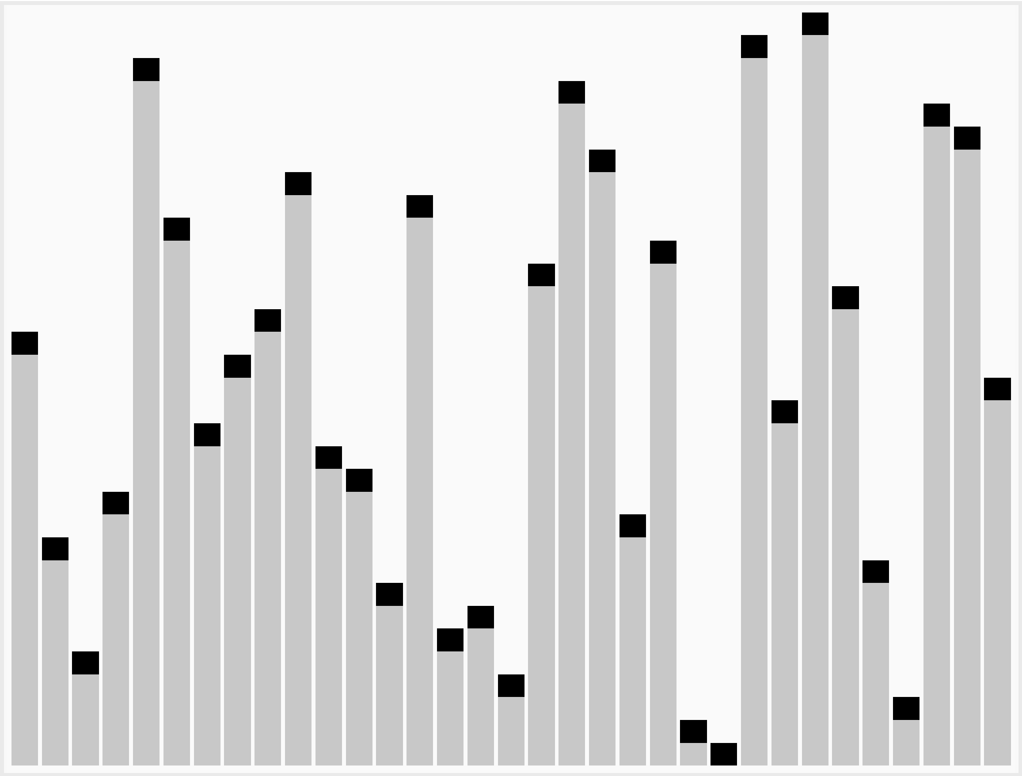
- Como aproveitar a ideia de partição do Quicksort ?
- Particionar o array de modo que:
  - Elemento A[j] esteja no seu lugar
  - Não há elementos maiores à sua esquerda
  - Não há elementos menores à sua direita
- Proceder de modo recursivo
  - MAS, processar uma só das partições !
  - Onde estará o valor procurado !

K-Selection – Algoritmo Recursivo de Partição



[Sedgewick & Wayne]

K-Selection – Algoritmo Recursivo de Partição



[Wikipedia]

K-Selection – Algoritmo Recursivo de Partição

```c
int qselect(int *v, int len, int k)
{
#       define SWAP(a, b) { tmp = v[a]; v[a] = v[b]; v[b] = tmp; }
        int i, st, tmp;

        for (st = i = 0; i < len - 1; i++) {
                if (v[i] > v[len-1]) continue;
                SWAP(i, st);
                st++;
        }

        SWAP(len-1, st);

        return k == st  ?v[st]
                        :st > k ? qselect(v, st, k)
                                : qselect(v + st, len - st, k - st);
}
```

[rosettacode.org]

Tarefa 1 – 3º elemento ?

0	1	2	3	4	5
6	**3**	**4**	**2**	**5**	**1**

Eficiência – Caso Médio

- Algoritmo LINEAR !!
- Cada partição é dividida em 2 “metades”
- Nº total de comparações aprox. igual a

  $n + n/2 + n/4 + \ldots + 1 \sim 2 \times n$ comparações
- Ex: 3.38 x n comparações para encontrar o mediano

Eficiência – Pior Caso

- Algoritmo QUADRÁTICO !!
  - Random shuffling no início, para evitar que aconteça

- A partição seguinte só tem menos 1 elemento
  - Tal como no pior caso do Quicksort

- Nº total de comparações aprox. igual a

$$n + (n - 1) + \dots + 1 \sim n^2/2 \text{ comparações}$$

Cálculo do valor de um polinómio – O Método de Horner

Calcular o valor de um polinómio P(x)

- $P(x) = ?$

- $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$

- Algoritmo direto e ingénuo – Quantas multiplicações ?
  - Sucessivas potências
  - Termos do polinónio

- Tarefa: fazer em casa !!

Método de Horner

- $P(x) = ?$

- $P(x) = a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + \ldots + x\,(a_{n-1} + x\,a_n) \ldots))$

- Método de Horner – Quantas multiplicações ?

- Tarefa: fazer em casa – V. iterativa + V. recursiva !!

Multiplicação de matrizes – O Algoritmo de Strassen

Multiplicação de matrizes – Alg. Direto

```
for( int i = 0; i < n; ++i )    // Initialization
    for( int j = 0; j < n; ++j )
        c[ i ][ j ] = 0;

for( int i = 0; i < n; ++i )
    for( int j = 0; j < n; ++j )
        for( int k = 0; k < n; ++k )
            c[ i ][ j ] += a[ i ][ k ] * b[ k ][ j ];
```

$O(n^3)$

[Weiss]

Multiplicação de matrizes – Alg. Direto

- Caso mais simples ?
  - Multiplicar matrizes (2 x 2) !!
- Algoritmo direto
  - 8 multiplicações
  - 8 adições
- Como fazer menos multiplicações ?
- MAS, não se fazem omeletas sem partir ovos…

Multiplicação de matrizes

- Como multiplicar matrizes de grande dimensão ?

- O algoritmo direto é $\Theta(m \times n \times p)$
- Ou $\Theta(n^3)$ no caso de matrizes quadradas $(n \times n)$

- Como fazer melhor ?

Multiplicação de matrizes quadradas

- Matrizes quadradas (n x n), com $n = 2^k$

- Divide-and-Conquer
  - Subdividir cada matriz em 4 sub-matrizes (n/2 x n/2)
  - Multiplicar recursivamente sub-matrizes
  - Combinar resultados intermédios para obter a matriz final

- Caso de base ?

1ª versão – Sub-Matrizes

$$\left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right]\left[\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}\right]=\left[\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}\right]$$

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$
$$C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$
$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$
$$C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

[Weiss]

1ª versão

$$\mathbf{AB} = \begin{bmatrix} 3 & 4 & 1 & 6 \\ 1 & 2 & 5 & 7 \\ 5 & 1 & 2 & 9 \\ 4 & 3 & 5 & 6 \end{bmatrix} \begin{bmatrix} 5 & 6 & 9 & 3 \\ 4 & 5 & 3 & 1 \\ 1 & 1 & 8 & 4 \\ 3 & 1 & 4 & 1 \end{bmatrix}$$

[Weiss]

we define the following eight $N/2$-by-$N/2$ matrices:

$$A_{1,1} = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \quad A_{1,2} = \begin{bmatrix} 1 & 6 \\ 5 & 7 \end{bmatrix} \quad B_{1,1} = \begin{bmatrix} 5 & 6 \\ 4 & 5 \end{bmatrix} \quad B_{1,2} = \begin{bmatrix} 9 & 3 \\ 3 & 1 \end{bmatrix}$$

$$A_{2,1} = \begin{bmatrix} 5 & 1 \\ 4 & 3 \end{bmatrix} \quad A_{2,2} = \begin{bmatrix} 2 & 9 \\ 5 & 6 \end{bmatrix} \quad B_{2,1} = \begin{bmatrix} 1 & 1 \\ 3 & 1 \end{bmatrix} \quad B_{2,2} = \begin{bmatrix} 8 & 4 \\ 4 & 1 \end{bmatrix}$$

1ª versão

- Quantas chamadas recursivas em cada passo ?
- Quantas multiplicações são efetuadas no total ?

$M(1) = 1$

$M(n) = 8\ M(n / 2)$

$M(n)$ in $\Theta(n^{\log_2 8}) = \Theta(n^3)$!!??!!

- Como melhorar a ordem de complexidade ?
- Como reduzir o nº de chamadas recursivas ?

Algoritmo de Strassen – Caso de base

- C = A x B
  - matrizes (2 x 2)

$m_1 = (a_{00} + a_{11}) \times (b_{00} + b_{11})$
$m_2 = (a_{10} + a_{11}) \times b_{00}$
$m_3 = a_{00} \times (b_{01} - b_{11})$
$m_4 = a_{11} \times (b_{10} - b_{00})$
$m_5 = (a_{00} + a_{01}) \times b_{11}$
$m_6 = (a_{10} - a_{00}) \times (b_{00} + b_{01})$
$m_7 = (a_{01} - a_{11}) \times (b_{10} + b_{11})$

$c_{00} = m_1 + m_4 - m_5 + m_7$
$c_{01} = m_3 + m_5$
$c_{10} = m_2 + m_4$
$c_{11} = m_1 + m_3 - m_2 + m_6$

- 7 multiplicações (!)
- 18 adições / subtrações

Algoritmo de Strassen – Caso geral

- C = A x B
  - matrizes (n x n)
  - $n = 2^k$
- Subdividir cada matriz em 4 sub-matrizes !
- Usar as mesmas "fórmulas" para processar cada sub-matriz
  - Adicionar, subtrair e multiplicar sub-matrizes
- Estas multiplicações são efetuadas de modo recursivo !!

Algoritmo de Strassen – Caso geral

p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

[GeeksforGeeks]

Algoritmo de Strassen – Eficiência

- Multiplicações ? Adições / subtrações ?

$M(1) = 1$

$M(n) = 7\ M(n / 2)$

$M(n) \text{ in } \Theta(n^{\log_2 7}) = \Theta(n^{2.807})$

$A(1) = 0$

$A(n) = 7\ A(n / 2) + 18 \times (n / 2)^2$

$A(n) \text{ in } \Theta(n^{2.807})$

Algoritmo de Strassen

- Há detalhes de implementação a ter em conta…

- Se necessário, acrescentar “zeros” para transformar em matrizes quadradas de tamanho apropriado

- Só é vantajoso para matrizes muito grandes !!

Algoritmo de Strassen

- Melhor do que o algoritmo direto !!
- Menos multiplicações, MAS mais adições / subtrações !!
- Hoje em dia, há algoritmos mais eficientes
  - E.g., o algoritmo de Coppersmith e Winograd é $\Theta(n^{2.376})$
  - Implementações complexas
  - Pouca aplicabilidade

Sugestões de leitura

Sugestões de leitura

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
  - Capítulo 4: secção 4.5
  - Capítulo 5: secção 5.4
  - Capítulo 6: secção 6.5

- M. A. Weiss, Data Structures and Algorithm Analysis in C++, 4th Edition, 2014
  - Capítulo 1: secção 1.1
  - Capítulo 6: secção 6.4
  - Capítulo 7: secção 7.7
  - Capítulo 10: secção 10.2