Projeto e Análise de Algoritmos I
QuickSort

Antonio Luiz Basile

Faculdade de Computação e Informática
Universidade Presbiteriana Mackenzie

August 21, 2018

QuickSort - Características Gerais

- O quicksort provavelmente é o algoritmo de ordenação mais utilizado.
- Foi criado em 1960 por C. A. R. Hoare.
- Não é difícil de se implementar.
- Funciona bem para diversos tipos de entrada de dados.
- Consome menos recursos que qualquer outro método de ordenação em muitas situações.
- Trabalha localmente.
- Na média requer tempo proporcional a $O(n \lg n)$.
- Não é estável.
- Leva tempo proporcional a $O(n^2)$ no pior caso.

Quicksort: Divisão e Conquista

Assim como no Mergesort, o paradigma da divisão e conquista se aplica ao Quicksort. Suponha um subvetor típico $A[p..r]$.

- **Dividir:** Particione o vetor $A[p..r]$ em dois (possivelmente vazios) subvetores $A[p..q-1]$ e $A[q+1..r]$ tal que cada elemento de $A[p..q-1]$ é menor ou igual a $A[q]$, que é, por sua vez, menor ou igual a cada elemento de $A[q+1..r]$. Compute o índice q como parte deste particionamento.
- **Conquistar:** Ordene os dois subvetores $A[p..q-1]$ e $A[q+1..r]$ por meio de chamadas recursivas ao Quicksort.
- **Combinar:** Como os subvetores já estão ordenados, nenhum trabalho é necessário para combiná-los: o vetor $A[p..r]$ já está ordenado.

Partition - Ideia Inicial

Seja um vetor de inteiros $v[0..n]$ que é a origem dos dados e sejam dois outros vetores (auxiliares) de mesmo tamanho $vmen[0..n]$ e $vmai[0..n]$ que serão usados como destino dos dados. Dado um valor inteiro p que é o primeiro elemento do vetor ($v[0]$), particione o vetor origem v do seguinte modo:

1. Os valores menores ou iguais à p vão para o vetor *vmen* e os maiores que p vão para *vmai*.
2. Em seguida os valores são copiados de volta para v do seguinte modo: a partir do índice 0 de v e progressivamente,
   - primeiro são copiados todos os valores de *vmen*,
   - em seguida é copiado o valor p e
   - finalmente são copiados todos os valores de *vmai*.
3. Simulação do partition-2way

Tarefa: Escreva em C o programa descrito acima.

Partition - 2 way

```
void part2way (int v[], int n)
{
  int vmen[n], vmai[n], i, j = 0, k = 0, p = v[0];

  for (i = 1; i < n; i++)
    if (v[i] <= p) vmen[j++] = v[i];
    else vmai[k++] = v[i];

  int tmen = j, tmai = k;
  k = j = i = 0;

  while (j < tmen) v[i++] = vmen[j++];
  v[i++] = p;
  while (k < tmai) v[i++] = vmai[k++];
}
```

Partition - Pseudo-Código

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Figure: Particionamento local (CLR)

Partition - local

O partition local particiona o vetor sem usar vetores auxiliares. Ele faz o serviço localmente. Em geral usa o primeiro ou o último elemento do vetor como pivô. Vamos simular o partition local para o vetor

$$v[] = \{8, 5, 9, 2, 10, 6, 11, 4, 1, 7\}$$

usando o último elemento do vetor como pivô.

Tarefa: escreva em C o partition local (descrito acima).

Partition local - Simulação

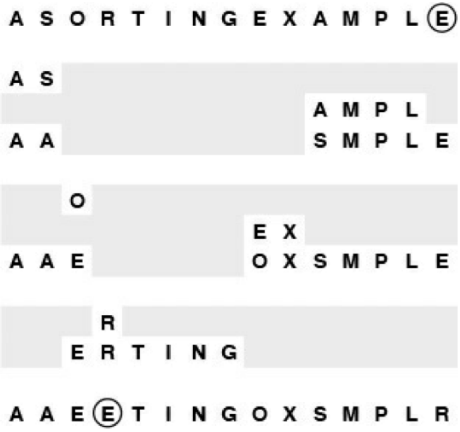


Figure: Particionamento local - Simulação (Sedgewick)

Partition local: Código em C

```c
#define exch(A, B) { int t = A; A = B; B = t; }

int partition(int v[], int l, int r) /* Sedgewick */
{
  int i = l-1, j = r, p = v[r];
  for (;;)
    {
      while (v[++i] < p);
      while (p < v[--j]) if (j == l) break;
      if (i >= j) break;
      exch (v[i], v[j]);
    }
  exch(v[i], v[r]);
  return i;
}
```

Quicksort - Pseudo-Código

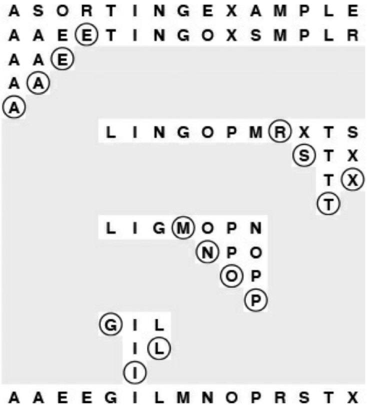
Observe o pseudo-código do Quicksort abaixo.

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

Figure: Quicksort (CLR)

Quicksort - Simulação (Sedgewick)

Agora observe a simulação da execução do Quicksort abaixo.



Tarefa: com base na simulação acima e no pseudo-código do slide anterior, escreva em C o Quicksort.

Quicksort em C

```
void qsort (int v[], int l, int r)
{
  int i;
  if (r <= l) return;
  i = partition (v, l, r);
  qsort (v, l, i-1);
  qsort (v, i+1, r);
}
```

Análise do Quicksort no Pior Caso

O pior caso do Quicksort ocorre quando o partition produz um subproblema com $n-1$ elementos e um com 0elementos. O particionamento custa tempo proporcional a $\Theta(n)$. Como a chamada recursiva de um vetor de tamanho 0 apenas retorna, $T(0) = \Theta(1)$, e a recorrência para o tempo de execução é

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

ou seja,

$$T(n) = T(n-1) + \Theta(n)$$

Tarefa: calcule a fórmula fechada para esta recorrência.

Análise do Quicksort no Melhor Caso

O melhor caso do Quicksort ocorre quando o partition produz dois subproblemas, cada um com tamanho não maior que $n/2$. Neste caso temos a recorrência para o tempo de execução de

$$T(n) = 2T(n/2) + \Theta(n)$$

.