DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Estrutura de Dados

Ordenação: Quicksort

Professores: Luiz Chaimowicz e Raquel Prates

Introdução

- Técnica de divisão e conquista
  - Subdividir um problema em subproblemas
  - Solução direta
- Passos
  - Divide
  - Conquista
  - Combina

Introdução

- Proposto por C. A. R. Hoare em 1960
  - Publicado em 1962 (refinamentos)
  - 26 anos
- Algoritmo de ordenação interna mais rápido que se conhece para diversas situações
  - Provavelmente é o mais utilizado

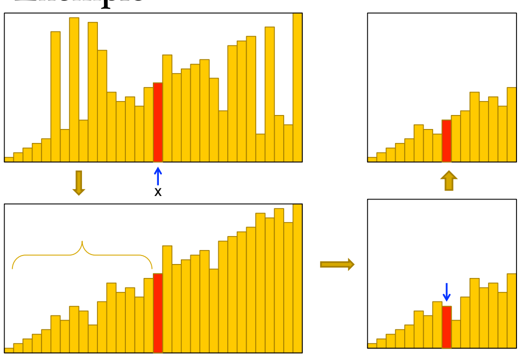
Introdução

- A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.

Introdução

- A parte mais delicada do método é o processo de partição.
- O vetor A [Esq ... Dir] é rearranjado por meio da escolha arbitrária de um **pivô** x.
- O vetor A é particionado em duas partes:
  - Parte esquerda: chaves ≤ x.
  - Parte direita: chaves ≥ x.

Exemplo



Quicksort - Partição

- Algoritmo para o particionamento:
  1. Escolha arbitrariamente um **pivô** x.
  2. Percorra o vetor com um índice **i** a partir da esquerda até que A[i] ≥ x.
  3. Percorra o vetor com um índice **j** a partir da direita até que A[j] ≤ x.
  4. Troque A[i] com A[j].
  5. Continue este processo (de 2 a 4) até os apontadores i e j se cruzarem.

Quicksort – Após a Partição

- Ao final, do algoritmo de partição:
  - o vetor A[Esq..Dir] está particionado de tal forma que:
    - Os itens em A[Esq], A[Esq + 1], ..., A[j] são menores ou iguais a *x*;
    - Os itens em A[i], A[i +1], ..., A[Dir] são maiores ou iguais a *x*.

Partição - Exemplo

- O pivô x é escolhido como sendo A[(i + j) / 2].
- Exemplo:

3	6	4	5	1	7	2

Partição - Exemplo

- O pivô x é escolhido como sendo A[(i + j) / 2].
- Exemplo:

Exemplo	3	6	4	5	1	7	2
Pivô	3	6	4	5	1	7	2
Primeira troca a ser feita	3	6	4	5	1	7	2
Segunda troca a ser feita	3	2	4	5	1	7	6
Resultado final	3	2	4	1	5	7	6

Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
  Item x, w;
  *i = Esq; *j = Dir;
  x = A[(*i + *j)/2]; /* obtem o pivo x */
  do
  {
    while (x.Chave > A[*i].Chave) (*i)++;
    while (x.Chave < A[*j].Chave) (*j)--;
    if (*i <= *j)
    {
      w = A[*i]; A[*i] = A[*j]; A[*j] = w;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}
```


Partição – "Casos Especiais"

- Pivô é o menor ou maior de todos

3	6	4	1	5	7	2

Partição – “Casos Especiais”

- Pivô é o menor ou maior de todos

	3	6	4	1	5	7	2
Primeira troca a ser feita	3	6	4	1	5	7	2
	j		i				
Resultado final	1	6	4	3	5	7	6

Depois da primeira troca, o índice i fica parado no elemento 6 (6 >= pivô) enquanto J é decrementado até parar no elemento 1, que é o próprio pivô. Como os índices se cruzam o procedimento termina

Partição – “Casos Especiais”

- Pivô não fica em uma das “bordas” após a partição

5	7	3	1	6	8	4	2	0

Partição – “Casos Especiais”

- Pivô não fica em uma das “bordas” após a partição

5	7	3	1	6	8	4	2	0
5	7	3	1	6	8	4	2	0
5	0	3	1	6	8	4	2	7
5	0	3	1	2	8	4	6	7

A partição continua ate os índices se cruzarem, mesmo após o pivô ter movido

5	0	3	1	2	4	8	6	7

Partição – “Casos Especiais”

- Pivô na posição correta

3	6	1	4	5	7	2

Partição – “Casos Especiais”

- Pivô na posição correta

	3	6	1	4	5	7	2
Primeira troca a ser feita	3	6	1	4	5	7	2

Após a primeira troca, os índices i e j continuam e param sobre o pivô. O pivô é trocado com ele mesmo e a partição termina com duas partições e mais um elemento (pivô) já na posição correta.

			j		i		
Resultado final	3	2	1	4	5	7	6

Quicksort

- O anel interno da função de Particao é extremamente simples.
- Razão pela qual o algoritmo Quicksort é tão rápido.

Quicksort - Função

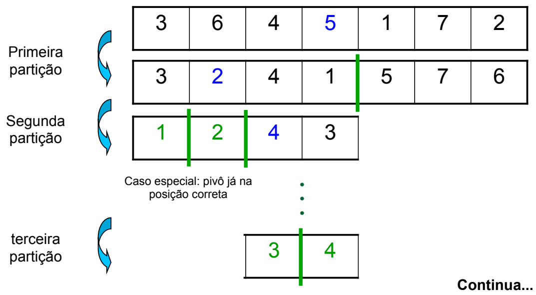
```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{ int i, j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```


Quicksort - Exemplo

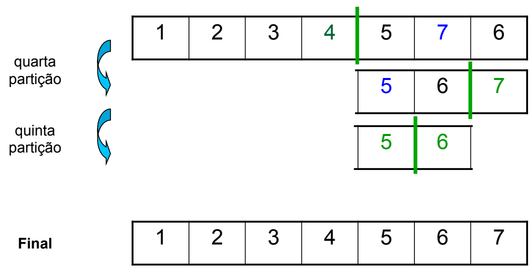
3	6	4	5	1	7	2

Quicksort - Exemplo



Continua...

Quicksort - Exemplo



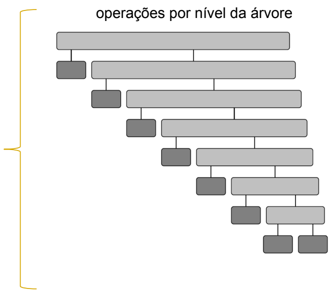
Quicksort - Exemplo

- **Características**
  - Qual o pior caso para o Quicksort? Qual sua ordem de complexidade?
  - Qual o melhor caso?
  - O algoritmo é estável?

Quicksort – Análise

- **Pior caso: (ex: entrada ordenada)**
  $C(n) = O(n^2)$
- O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.
- Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.
- $T(n) = n + T(n-1)$

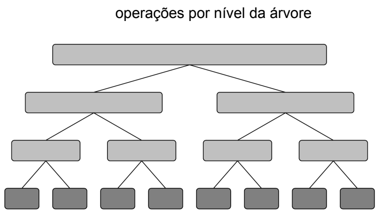
Quicksort – Análise



Quicksort – Análise

- **Melhor caso:**
  C(n) = O(n log n)
- Esta situação ocorre quando cada partição divide o arquivo em duas partes iguais.
- T(n) = 2T(n/2) + n

Quicksort – Análise



Quicksort – Análise

- **Caso médio** de acordo com Sedgewick e Flajolet (1996, p. 17):
  C(n) ≈ 1,386n log n – 0,846n,
- Isso significa que em média o tempo de execução do Quicksort é O(n log n).

Quicksort analysis

Let's call the time function for Quicksort, T, then:

$$T(1) = 1$$
$$T(high - low) = high - low + T(mid - low + 1) + T(high - mid)$$
$$T(n) = n + T(mid - low + 1) + T(high - mid)$$

Let's have a look at two cases:

1. Worst case: mid = low
2. Best case: mid = (low+high)/2

Quicksort analysis: worst case

When mid = low, the time function for Quicksort becomes:

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

Which we can prove by induction (left as an exercise).

So in the worst case Quicksort is $O(n^2)$. That's not good.

The worst case occurs exactly when Quicksort is run on an already sorted array.

We can avoid the worst case by choosing a better pivot. There are a few options, but the most common are: choose a random element as the pivot; choose the median of 3 elements (e.g. first, middle, last) as the pivot.

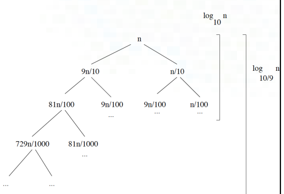
Quicksort analysis: uneven, but not worst case:

What if we're always somewhere between the worst and best case? Is Quicksort good or bad on average?

Consider a pivot that always gives us a 90%/10% split of the data (sounds pretty bad). In that case, the time function is:

Since each layer of the tree does at most n things and there are at most $\log_{10/9} n$ layers, total time is still $O(n \log n)$.

This is true for any split that **keeps the same proportion**, even a 99 to 1 split. (But note that a split of $n-1$ to 1 is not of constant proportion, which is why we got $O(n^2)$ in that case.)



Quicksort

- Vantagens:
  - É extremamente eficiente para ordenar arquivos de dados.
  - Necessita de apenas uma pequena pilha como memória auxiliar.
  - Requer cerca de n log n comparações em média para ordenar n itens.
- Desvantagens:
  - Tem um pior caso O(n²) comparações.
  - Sua implementação é muito delicada e difícil:
    - Um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
  - O método não é **estável**.

Melhorias no Quicksort

- Escolha do pivô:
  - Mediana de três: pior caso menos provavel
  - Mediana-de-medianas em O(n): evita pior caso
- Utilizar um algoritmo simples (seleção, inserção) para partições de tamanho pequeno
- Quicksort não recursivo
  - Evita o custo de várias chamadas recursivas
- Implementação estável
  - Custo adicional de memória O(n)

Quicksort não recursivo

```
void QuickSortNaoRec(Vetor A, int n)
{
//a pilha vai conter os limites esq e
//dir dos subvetores ordenados
     TipoPilha pilha;
     TipoItem item;  // campos esq e dir
     int esq, dir, i, j;

     FPVazia(&pilha);
     esq = 0;
     dir = n-1;
     item.dir = dir;
     item.esq = esq;
     Empilha(item, &pilha);
```

```
    do
       if (dir > esq)  {
          Particao(A,esq,dir,&i, &j);
//empilha os limites do maior subvetor
          if ((j-esq)>(dir-i))  {
             item.dir = j;
             item.esq = esq;
             Empilha(item, &pilha);
             esq = i;
          }
          else {
              item.esq = i;
              item.dir = dir;
              Empilha(item, &pilha);
              dir = j;
          }
       }
//proximo subvetor sera o que esta no topo
       else {
           Desempilha(&pilha,&item);
           dir = item.dir;
           esq = item.esq;
    } while (!Vazia(pilha));
}
```

Exercicio: Quicksort iterativo?

Create a stack which has the size of the array

1. Push Initial values of start and end in the stack ie, parent array(full array) start and end indexes

2. Till the stack is empty

3. Pop start and end indexes in the stack

4. call the partition function and store the return value it in pivot_index

5. Now, push the left subarray indexes that is less to the pivot into the stack ie, start, pivot_index -1

6. push the right subarray indexes that is greater than pivot into the stack ie, pivot+1, end


```
void iterativeQuicksort(int arr[], int start, int end){
     int stack[end - start + 1];//Initializing stack size
     int top = -1; //To keep track of top element in the stack
     //pushing intial start and end
     stack[++top] = start;
     stack[++top] = end;

     //pop till the stack is empty
     while(top >= 0)
     {
          //poping the top two elements
          //ie,poping parent subarray indexes to replace child subbary indexes
          end = stack[top--];
          start = stack[top--];
          int pivot_index = partition(arr, start, end);

      //Pushing the left subarray indexes that are less than pivot
      if (pivot_index - 1 > start ){
          stack[++top] = start;
          stack[++top] = pivot_index -1;
      }
      //pushing the right subarray indexes that are greater than pivot
      if (pivot_index + 1 < end){
          stack[++top] = pivot_index + 1;
          stack[++top] = end;
      }
     }
}
```

```
int partition(int arr[], int start, int end)
{
      int pivot = arr[end]; //rightmost element
is the pivot
      int pIndex = start;  //Is to push
elements less than pivot to left and greater
than to right of pivot
      for (int i = start; i < end; ++i)
      {
            if (arr[i] < pivot)
            {
                  swap(arr[i],arr[pIndex]);
                  pIndex++;
            }
      }
      swap(arr[pIndex], arr[end]);
      return pIndex;
}
```


Iterative Quick Sort

Input :

arr [ ] =

0	1	2	3	4
4	1	10	23	5

Initializing Stack,

Stack =

0	1	2	3	4

Push start, end indexes

Stack =

0	1	2	3	4
0	4			

Pop top two elements in stack and cull partition function and store return value in pivot_index, ie pivot_index = 2

arr =

0	1	2	3	4
4	1	5	23	10

Stack =

0	1	2	3	4

Now, Push left Subarray indexes and right Subarray indexes of the Pivot_indexes.

arr =

0	1	2	3	4
4	1	5	23	10

Stack =

0	1	2	3	4
0	1	3	4	

Now, Start = 3, end = 4, Pop top two elements call the partition function ;

arr =

0	1	2	3	4
4	1	5	10	23

Stack =

0	1	2	3	4
0	1			

Now, Start = 0, end = 1, Pop top two elements call the partition function ;

arr =

0	1	2	3	4
1	4	5	10	23

Stack =

0	1	2	3	4

∴ The sorted array is { 1, 4, 5, 10, 23 }

Exercício

- Execute o quicksort no vetor abaixo indicando qual é o pivô e os subvetores resultantes de cada partição.

65	77	51	25	03	84	48	21	05