Algoritmos de Ordenação

1. importância do estudo dos algoritmos de ordenação:

1. bom exemplo da resolução de problemas via computador
2. exemplificar o conceito de *classe* de algoritmos
3. apresentação de algoritmos *competitivos*
4. ilustração da manipulação de estruturas de dados

2. conceitos

1. a *chave* controla a ordenação
2. estabilidade: chaves iguais não mudam suas posições relativas
3. ordenação interna $\neq$ externa

3. aspectos importantes na escolha de um algoritmos

1. complexidade do número de comparações
2. complexidade do número de trocas

4. classificação

1. métodos simples ($O(n^2)$ comparações)
2. métodos eficientes($O(n \log n)$ comparações)

5. ordenação por seleção:

	chaves				
iteração	C	A	R	O	L
2	**A**	**C**	R	O	L
3	A	**C**	R	O	L
4	A	C	**R**	O	**L**
5	A	C	L	**O**	R
6	A	C	L	O	R

Copyright ©1999, F. R. B. Cruz.

6. ordenação por inserção:

	chaves				
iteração	C	A	R	O	L
2	**A**	C	R	O	L
3	A	**C**	R	O	L
4	A	C	**O**	R	L
5	A	C	L	**O**	R

7. *shellsort* (estende o método da inserção):

	chaves				
h	C	A	R	O	L
4	C	A	R	O	L
2	C	A	L	O	R
1	A	C	L	O	R

8. *quicksort*:

	chaves				
iteração	C	A	**R**	O	L
2	C	**A**	L	O	R
3	A	C	L	O	R

9. resumo:

método	estável	complexidade	
seleção	não	$C(n) = (n^2 - n)/2$	$M(n) = 3(n-1)$
inserção	sim	$C(n) = (n^2 + 3n)/4 - 1$	$M(n) = \frac{n^2+11n}{4} - 3$
shellsort	não	$O(n^{1,25})$ ou $O(n(\log n)^2)$	(?)
quicksort	não	$O(n \log n)$	

Copyright ©1999, F. R. B. Cruz.

Algoritmos de Pesquisa

1. recuperação de informações
2. encontrar uma ou mais ocorrências de registros com a *chave* desejada
3. algoritmos operam em *tabelas* ou *arquivos*
4. a escolha do algoritmo depende da:
   1. quantidade de dados nos quais se dá a pesquisa
   2. sujeição a insersões e retiradas frequentes
5. a pesquisa é um TAD, denominado *dicionário*, no qual estão disponíveis as operações:
   1. iniciliazação da estrutura
   2. pesquisa do registro
   3. inserção de registro
   4. retirada de um registro
   5. ordenação dos registros
   6. fusão de dois dicionários
   7. exemplo:

```
const int DIC_TAM_MAX = 1000;

typedef int TipoChave;

struct TipoElem {
  TipoChave chave; // chave
  // outros campos
};

class Dicionario {
```

Copyright ©1999, F. R. B. Cruz.


```
private:
  TipoElem *elem;
  int noDeElem;
public:
  Dicionario(); // inicializa
  ~Dicionario();
  int Insere(const TipoElem &elem);
  int Pesquisa(TipoChave chave);
};
```

6. pesquisa sequencial

```
#include <stdio.h>
#include "dicio.hpp"

Dicionario::Dicionario() {
 elem = new TipoElem[DIC_TAM_MAX];
 noDeElem = 0;
}

Dicionario::~Dicionario() {
 delete elem;
}

int Dicionario::Insere(const TipoElem &elem) {
   if ( noDeElem == DIC_TAM_MAX ) {
      fprintf(stdout,
      "Dicionario::Insere: erro dicionario cheio\n");
      return 1;
   }
   Dicionario::elem[noDeElem++].chave = elem.chave;
   // ...
```

Copyright ©1999, F. R. B. Cruz.

```
   return 0;
}

int Dicionario::Pesquisa(TipoChave chave) {
  int i = 0;
  while ( (i < noDeElem)&&(chave != elem[i].chave) )
     i++;
  if ( i != noDeElem )
     return i;
  else
     return -1; // erro: elemento nao encontrado
}
```

7. exemplo de utilização

```
// file diciomain.cpp

#include <stdio.h>
#include "dicio.hpp"

int main() {
 Dicionario Dic;
 TipoElem elem;
 TipoChave chave;
 int posicaoChave;
 int i;

// insere numeros
 for (i=0; i < 100; i++) {
   chave = i * 4;
   elem.chave = chave;
   Dic.Insere(elem);
```

Copyright ©1999, F. R. B. Cruz.


```
 }
// pesquisa
 for (i=0; i < 5 ; i++) {
   chave = i;
   posicaoChave = Dic.Pesquisa(chave);
   if ( posicaoChave == -1 )
      fprintf(stdout,
      "Chave %d nao encontrada\n", chave);
   else
      fprintf(stdout,
      "A chave %d esta na posicao %d\n",
      chave, posicaoChave);
 }
 return(0);
}

// end of file diciomain.cpp
```

8. saída

```
>dicio
A chave 0 esta na posicao 0
Chave 1 nao encontrada
Chave 2 nao encontrada
Chave 3 nao encontrada
A chave 4 esta na posicao 1
```

9. pesquisa binária

10. *hashing*

Copyright ©1999, F. R. B. Cruz.