Programação 1

TP5: *aula 11*

Pesquisa e ordenação

Plano

Pesquisa de valores em sequências

- Pesquisa sequencial
- Pesquisa binária

Ordenação de sequências

- Ordenação sequencial
- Ordenação por flutuação

Pesquisa de valores em arrays

Existem vários algoritmos para a pesquisa de valores em arrays, mas vamos focar-nos apenas em dois:

- Pesquisa sequencial: se o array náo estiver ordenado
- Pesquisa binária: só é possível se o array estiver ordenado

Esta tarefa pode ser computacionalmente dispendiosa se estivermos a tratar grandes volumes de dados

A complexidade dos algoritmos pode variar significativamente

Pesquisa sequencial

Percorre os valores do array um a um, começando pelo primeiro, até encontrar o valor pretendido ou até atingir o último elemento.

Os elementos do array não precisam de estar ordenados.

Em todos os algoritmos de pesquisa é sempre necessário encontrar uma forma de ‘sinalizar’ que não encontramos o valor pretendido.

Pesquisa sequencial

```
static int PesquisaSequencial(int seq[],int nElem, int valor) {
	int n=0;
	int pos = -1; // inicia com um valor inválido
	do {
		if(seq[n++] == valor) {
			pos = n-1;
		}
	} while(pos == -1 && n <nElem);
	return pos;
}
```

Pesquisa binária

Se tivermos informação à priori sobre os elementos da sequência, podemos acelerar o processo de pesquisa. Por exemplo, **se a sequência estiver ordenada por ordem crescente ou decrescente**, podemos fazer uma pesquisa binária.

- O algoritmo começa por selecionar o elemento central da sequência e compara-o com o elemento procurado.
- Se o elemento foi maior, podemos excluir a primeira metade da sequência, caso contrário podemos excluir a segunda metade.
- O processo é repetido até que o elemento seja o procurado ou até deixarmos de ter elementos para analisar.

Pesquisa binária

```
public static int PesquisaBinaria(int[] lista, int valor){
	int inicio=0, fim=lista.length-1, meio;
	int haValor= -1;
	do {
		meio=(inicio+fim)/2;
		if (valor > lista[meio] ) {
			inicio=meio+1;
		} else if (valor < lista[meio]) {
			fim=meio-1;
		} else {
			haValor = meio;
		}
	} while(haValor == -1 && inicio <= fim );
	return haValor;
}
```

Ordenação

Existem vários algoritmos para a ordenação de arrays mas vamos apenas analisar dois: **ordenação por seleção** e **ordenação por flutuação**.

- Na ordenação sequencial vamos colocando em cada posição da sequência o valor correto, começando no primeiro.
- Na ordenação por flutuação vamos comparando pares de valores da sequência e trocamos se fora de ordem. Repetimos o processo enquanto houver trocas.

Ordenação por seleção

Lista de tamanho *n*

1a	2a	3a	4a	5a	6a	
55	7	7	7	7	7	7
23	23	14	14	14	14	14
32	32	32	23	23	23	23
7	55	55	55	27	27	27
14	14	23	32	32	32	32
27	27	27	27	55	55	45
45	45	45	45	45	45	55

Repetir $n - 1$ vezes

- Procurar o mínimo (ou máximo) da lista não ordenada;
- Trocar mínimo (ou máximo) com 1ª posição da lista não ordenada;
- lista ordenada (amarelo) corresponde às primeiras posições que vão ficando com os mínimos sucessivos;

Ordenação por seleção

```
public static void ordenacaoSequencial(int seq[], int n){
   int tmp, i, j;
   for(i = 0 ; i < n - 1 ; i++){      // fixamos uma posicao
      for(j = i + 1 ; j < n ; j++){   // percorremos as posicoes seguintes
         if(seq[i] > seq[j])          // se necessário de sitio, trocamos
         {
            tmp = seq[i];
            seq[i] = seq[j];
            seq[j] = tmp;
         }
      }
   }
}
```

Ordenação por flutuação (bolha ou bubble sort)

1a	2a	3a	4a	5a	6a	
55	23	23	23	23	23	23
23	55	32	32	32	32	32
32	32	55	7	7	7	7
7	7	7	55	14	14	14
14	14	14	14	55	27	27
27	27	27	27	27	55	45
45	45	45	45	45	45	55

- Percorre a lista não ordenada (verde) comparando elementos sucessivos e troca-os se o 1º for maior que o 2º
- No fim da 1ª passagem o maior fica no fim da lista (azul), ficando ordenado (para a ordenação decrescente);
- No fim da 2ª passagem o 2º maior fica na penúltima posição;

Ordenação por flutuação

1a	2a	3a	4a	5a	
23	23	23	23	23	23
32	32	7	7	7	7
7	7	32	14	14	14
14	14	14	32	27	27
27	27	27	27	32	32
45	45	45	45	45	45
55	55	55	55	55	55

1a	2a	3a	4a	
23	7	7	7	7
7	23	14	14	14
14	14	23	23	23
27	27	27	27	27
32	32	32	32	32
45	45	45	45	45
55	55	55	55	55

1a	2a	3a	
7	7	7	7
14	14	14	14
23	23	23	23
27	27	27	27
32	32	32	32
45	45	45	45
55	55	55	55

Ordenada –
Não há trocas

- Repete o processo até que faltem ordenar apenas 2 elementos (para lista de tamanho ***n***, são ***n-1*** vezes);

-Ou até quando não hajam trocas (melhora eficiência);

Ordenação por flutuação

```
public static void OrdenacaoFlutuacao(String [] seq, int n){
        String tmp;
        int i, j;
        boolean trocas;
        do {
                trocas = false;                 // partir do principio que já está ordenado...
                for(i = 0 ; i < n -1 ; i++){
                        if(seq[i].compareTo(seq[i+1]) < 0){
                                tmp = seq[i];
                                seq[i] = seq[i+1];
                                seq[i+1] = tmp;
                                trocas = true;          // houve trocas...
                        }
                }
        }while(trocas);                         // repetir enquanto houver trocas
}
```

Outros algoritmos de ordenação

Existem muitos outros algoritmos de ordenação:

- shell
- Fusão
- QuickSort

E muitos outros

Animações de algoritmos de ordenação

https://www.toptal.com/developers/sorting-algorithms
http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html.