

Aula 20: Ordenação

Professor(a): Liliane dos Reis Gade

INTRODUÇÃO A PROGRAMAÇÃO - SETOR DE INFORMÁTICA



Ordenação

- Definição do problema: Dada uma coleção de n elementos, representada em um vetor de 0 a $n-1$, deseja-se obter uma outra coleção, cujos elementos estejam ordenados segundo algum critério de comparação entre os elementos.
- Tipos de algoritmos que serão discutidos: Iterativos e Recursivos
- Alguns exemplos: InsertionSort, **BubbleSort**, RadixSort, HeapSort, **MergeSort**, **QuickSort**.

BubbleSort

- Os elementos do vetor devem ser trocados entre si para que fiquem na ordem desejada.

```
1 void bubbleSort(int *p, int t){
2     int i, j, aux;
3     for ( i=0; i < t; i++){
4         for ( j=t-1; j > i; j-- ){
5             if ( p[j] < p[j-1]){
6                 aux = p[j];
7                 p[j] = p[j-1];
8                 p[j-1] = aux;
9             }
10        }
11    }
12 }
```

<http://www.youtube.com/watch?v=lyZQPjUT5B4>

- A idéia básica do MergeSort é criar uma sequência ordenada a partir de duas outras também ordenadas. Para isso, o algoritmo MergeSort divide a sequência original em pares de dados, agrupa estes pares na ordem desejada; depois as agrupa as sequências de pares já ordenados, formando uma nova sequência ordenada de quatro elementos, e assim por diante, até ter toda a sequência ordenada.

- Os três passos úteis dos algoritmos dividir-para-conquistar, que se aplicam ao MergeSort são:
 - **Dividir:** Dividir os dados em subsequências pequenas; Este passo é realizado recursivamente, iniciando com a divisão do vetor de n elementos em duas metades, cada uma das metades é novamente dividida em duas novas metades e assim por diante, até que não seja mais possível a divisão (ou seja, sobrem n vetores com um elemento cada).
 - **Conquistar:** Classificar as duas metades recursivamente aplicando o mergesort;
 - **Combinar:** Juntar as duas metades em um único conjunto já classificado. Para completar a ordenação do vetor original de n elementos, faz-se o merge ou a fusão dos sub-vetores já ordenados.

http://www.youtube.com/watch?v=XaqR3G_NVoo

MergeSort

```
1 void merge(int a[], int low, int
  high, int mid){
2   int i, j, k, c[50];
3   i=low;
4   j=mid+1;
5   k=low;
6   while((i<=mid)&&(j<=high)){
7     if(a[i]<a[j]){
8       c[k]=a[i];
9       k++;
10      i++;
11    }
12    else{
13      c[k]=a[j];
14      k++;
15      j++;
16    }
17  }
18  while(i<=mid){
19    c[k]=a[i];
20    k++;
21    i++;
22  }
23  while(j<=high){
24    c[k]=a[j];
25    k++;
26    j++;
27  }
28  for(i=low; i<k; i++)
29    a[i]=c[i];
30 }
```

```
1 void mergeSort(int a[], int low, int
  high){
2   int mid;
3   if(low>high){
4     mid=(low+high)/2;
5     mergeSort(a, low, mid);
6     mergeSort(a, mid+1, high);
7     merge(a, low, high, mid);
8   }
9 }
```

- Determina-se um elemento pivô. O pivô é posicionado dentro do vetor de tal forma que, todos à esquerda do pivô são menores que ele e, todos à direita do pivô são maiores. O pivô "divide" o vetor em dois subvetores. Recursivamente o quicksort é realizado na primeira metade do vetor e na segunda metade.

- Algoritmo: Seja x o vetor a ser ordenado e n o número de elementos de x . Seja a um elemento de x escolhido ao acaso (por exemplo, $a=x[0]$). Suponha que os elementos de x estejam divididos de tal forma que a é colocado na posição j e as seguintes condições são verdadeiras:
 - Todos os elementos nas posições de 0 a $j-1$ são menores que a .
 - Todos os elementos nas posições de $j+1$ a $n-1$ são maiores ou iguais a a .
- Então a está na posição correta no vetor. Se este processo for repetido para os sub-vetores $x[0]$ a $x[j-1]$ e $x[j+1]$ a $x[n-1]$, o resultado é o vetor ordenado.

<http://www.youtube.com/watch?v=kDgvnbU1qT4>

QuickSort

```
1  int partition( int a[], int l, int r) {
2      int pivot, i, j, aux;
3      pivot = a[l];
4      i = l;
5      j = r+1;
6      while(1){
7          do ++i; while( a[i] <= pivot && i <= r );
8          do --j; while( a[j] > pivot );
9          if( i >= j ) break;
10         aux = a[i];
11         a[i] = a[j];
12         a[j] = aux;
13     }
14     aux = a[l];
15     a[l] = a[j];
16     a[j] = aux;
17     return j;
18 }
19
20 void quickSort( int a[], int l, int r){
21     int j;
22     if( l > r ){
23         //dividir e conquistar
24         j = partition( a, l, r );
25         quickSort( a, l, j-1);
26         quickSort( a, j+1, r );
27     }
28 }
```

No main o QuickSort será chamado como quickSort(vetor, 0, tam);