MC102 - Algoritmos e Progração de Computador

Prof. Alexandre Xavier Falcão

19° Aula: Recursão (cont.)

1 Ordenação por indução fraca

Os algoritmos de ordenação de seqüências por seleção, inserção, e permutação podem ser implementados usando indução fraca. Na ordenação por inserção, ordenamos a seqüência até a penúltima posição e inserimos o último elemento na posição correta da seqüência ordenada. Na ordenação por seleção, selecionamos o maior elemento, colocamos ele na última posição e depois repetimos o processo para a subseqüência terminada no penúltimo. Na ordenação por permutação, o processo é similar. Os elementos são permutados até que o maior seja o último, e depois repetimos o processo para a subseqüência terminada no penúltimo.

Por exemplo, a função abaixo ordena de forma recursiva um vetor v de inteiros e de tamanho n por inserção.

```
void Insercao(int *v, int n)
{
  int i,j; /* variáveis locais */

  /* A condição de parada é não fazer nada. Caso contrário: */
  if (n > 1) {
    /* comandos iniciais: vazio */
    /* chamada recursiva */
    Insercao(v,n-1);
    /* comandos finais: insere o último elemento na posição correta. */
    i = n-2; j=n-1;
    while ( (i >= 0) && (v[i] > v[j])){
      troca(&v[i],&v[j]);
      i--; j--;
    }
  }
}
```

2 Ordenação por indução forte

A vantagem da indução forte é reduzir a complexidade da ordenação de $O(n^2)$ para $O(n \log n)$. O algoritmo mais simples nesta linha é o $merge - sort$. Este algoritmo subdivide a seqüência em duas, ordena de forma recursiva cada parte, e depois intercala as partes ordenadas.

```
/* Considerando que o vetor está ordenado do início até o meio e do
   meio + 1 até o final, intercala seus elementos para que fique
   ordenado do início ao fim. */

void Intercala(int *v, int inicio, int meio, int fim)
{
  int i,j,k,vaux[N]; /* Ordenação requer memória auxiliar do mesmo
                        tamanho da entrada. */

  i=inicio;
  j=meio+1;
  k=inicio;

  while((i<=meio)&&(j<=fim)){
    if (v[i] <= v[j]){
      vaux[k]=v[i];
      i++; k++;
    }else{
      vaux[k]=v[j];
      j++; k++;
    }
  }
  for(i=i; i <= meio; i++,k++)
    vaux[k]=v[i];
  for(j=j; j <= fim; j++,k++)
    vaux[k]=v[j];
  /* copia de volta para v */
  for (i=inicio; i <= fim; i++)
    v[i]=vaux[i];
}

void MergeSort(int *v, int inicio, int fim)
{
  int meio; /* variável local */

  /* A condição de parada é não fazer nada. Caso contrário: */
  if (inicio < fim) {
    /* comando inicial: calcula o meio */
    meio = (inicio+fim)/2;
    /* chamadas recursivas */
    MergeSort(v,inicio,meio);
    MergeSort(v,meio+1,fim);
    /* comando final: intercalação */
    Intercala(v,inicio,meio,fim);
  }
}
```

Uma desvantagem do algoritmo acima, porém, é a necessidade de memória auxiliar, na função de intercalação, do mesmo tamanho da entrada. Isto pode ser um problema para seqüências muito grandes.

Outro algoritmo que usa indução forte, tem complexidade $O(n \log n)$ no caso médio, e $O(n^2)$ no pior caso, mas não requer memória auxiliar é o $quick - sort$. Este algoritmo particiona a seqüência em duas partes de tal forma que todos os elementos da primeira parte são menores ou iguais aos da segunda. A seqüência é ordenada repetindo-se este processo de forma recursiva para cada parte.

```
void QuickSort(int *v, int inicio, int fim)
{
  int p;

  if (inicio < fim){
    p = Particiona(v,inicio,fim);
    QuickSort(v,inicio,p);
    QuickSort(v,p+1,fim);
  }
}
```

3 Exercícios

1. Escreva as funções de ordenação de forma recursiva, por seleção e por permutação, de um vetor v com n elementos.

2. Escreva uma função recursiva para ordenar v por partição. Isto é, complete o código do $quick - sort$.