Atividade ANP assíncrona - Selection Sort

A aula está dividida em algumas partes: compreensão do algoritmo, análise deste e implementação. A atividade deverá ser realizada durante o horário da aula, sendo que o prazo para o envio das respostas das questões está especificado na atividade aberta no Moodle. As implementações deverão ser enviadas no formato compactado para o Moodle. Esta esta aula ANP, se corretamente realizada, garante a presença no dia da aula.

Na aula de hoje, será estudado um algoritmo de ordenação "Bonus", ou seja, não visto em aula: **Ordenação por Seleção** ou ***Selection Sort***. Este algoritmo é um dos mais simples e consiste em, a cada passo encontrar o menor valor do vetor e passar para a posição da vez. Como exemplo, considere o seguinte cenário de números para ordenação:

3	40	23	61	12	2	71	11

No início, o *Selection sort* precisa achar o menor (considera-se ordenação crescente, mas poderia ser também decrescente) elemento para colocar na primeira posição, para tanto, ele efetua uma varredura no vetor:

Posição atual					Menor elem.		
3	40	23	61	12	2	71	11

Quando o menor elemento do vetor é encontrado, troca-se ele pelo primeiro:

2	40	23	61	12	3	71	11

A primeira posição agora está correta, possui o menor elemento. Então o algoritmo prossegue para a segunda posição, efetuando a busca pelo segundo menor elemento

	Posição atual				Menor elem.		
2	40	23	61	12	3	71	11

Quando o menor elemento do restante do vetor é encontrado, troca-se ele pelo segundo:

2	3	23	61	12	40	71	11

O mesmo é feito para a terceira posição:

		Posição atual					Menor elem.
2	3	23	61	12	40	71	11

Efetuando a troca:

2	3	11	61	12	40	71	23

Se o processo continuar até o última posição, consegue-se:

2	3	11	12	23	40	61	71

Formalizando o algoritmo

Considerando o exemplo fornecido anteriormente, pode-se apresentar a formalização do algoritmo em sua forma mais simples:

```
1:  procedure ORDENACAO_SELECAO(A[], tamanho)
2:      for i de 0 e i < tamanho − 1 do
3:          menor ← i
4:          for j de i + 1 e j < tamanho do
5:              if A[j] < A[menor] then
6:                  menor ← j
7:              end if
8:          end for
9:          if menor ≠ i then
10:             temp ← A[i]
11:             A[i] ← A[menor]
12:             A[menor] ← temp
13:         end if
14:     end for
15: end procedure
```

No algoritmo anterior, o **for** que se inicia na linha 2 serve para definir qual o índice do elemento que será atualizado na rodada atual (começa no primeiro e vai até o penúltimo). Na linha 3, assume-se que o elemento da posição i é o menor elemento. Já o **for** iniciado na linha 4 varre o vetor partindo do elemento seguinte a i até o final para encontrar o menor elemento do subvetor restante. Se um menor elemento é encontrado, uma atualização é feita na linha 6. Terminado este **for**, caso seja encontrado um elemento menor que o que está apontado por i (**if** da linha 9), estes são trocados de posição.

Analisando a complexidade do algoritmo

No algoritmo anterior, pode-se entender como operação dominante (a qual executa mais vezes) o **If** inteiro que abrange as linhas 5-7, que é a operação de comparação. Considerando um n (quantidade de dados) qualquer, a primeira rodada executará (n-1) vezes, pois o **for** irá de *j = 0+1 até tamanho*[1]*-1*. Na segunda rodada será (n-1), pois agora o **for** executará de *j = 1+1 até tamanho-1.* Isto se repete até o penúltimo elemento, onde o for executará 1 vez acessando apenas o último elemento, pois *j = tamanho-2+1 até tamanho-1.* Generalizando, obtém-se o seguinte número de execuções da operação dominante:

$$(n-1)+(n-2)+\ldots+1=\sum_{i=1}^{n-1} i$$

A progressão anterior é uma progressão aritmética. Para derivar a expressão do somatório para um valor genérico de n, pode-se recorrer a **soma da progressão**, ou **série.** Essa soma pode ser encontrada rapidamente pegando o número de termos sendo adicionados (neste caso n-1), multiplicando pela soma do primeiro e último número na progressão (aqui (n-1) + 1 n) e dividindo por 2. Para a progressão anterior:

$$\sum_{i=1}^{n-1} i=\frac{((n-1)+1)\times(n-1)}{2}=\frac{n\times(n-1)}{2}=\frac{1}{2}\times(n^2-n)$$

Como considera-se a notação assintótica, elimina-se a constante multiplicativa (1/2) e o termo de menor expressividade (n), sobrando apenas n^2, então a complexidade de pior caso será $O(n^2)$. A maneira intuitiva de derivar a complexidade é observar que existe uma estrutura for dentro de outra, sendo ambos delimitados por n, logo, têm-se n^2 execuções, equivalendo então a$O(n^2)$.

Propriedades do algoritmo

As propriedades do algoritmo são as seguintes:

- Complexidades de pior, melhor e caso médio iguais: independente da entrada, o algoritmo executará as mesmas operações, não sendo parcialmente dependente da estrutura da entrada, como o inserção e o bolha otimizado.
- Não é estável: elementos iguais podem ser trocados de ordem.
- Ordenação *in loco*, pois não precisa de memória adicional.
- Um dos algoritmos $O(n^2)$ mais rápidos para vetores pequenos.

1 Tamanho e n são equivalentes.

Atividades

1. Apresente um exemplo que evidencie o fato do algoritmo não ser estável. Pode ser uma explicação em um arquivo de texto.
2. Implemente o algoritmo e C.

Envie sua resposta e implementação pelo Moodle.

Bom trabalho!