Programação 2

Classificação Trocas (Bubble, Skake e Comb Sort)

Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

Definição

- Os métodos de classificação por trocas caracterizam-se por efetuarem a classificação por comparação entre pares de elementos, trocando-as de posição caso estejam fora de ordem no par.

Bubble Sort

- Nesse método, o principio geral é aplicado a todos os pares consecutivos de elementos. Este processo é executado enquanto houverem pares consecutivos de chaves não ordenados. Quando não restarem mais pares não ordenados, o vetor estará classificado.

Bubble Sort

- A função **swapbubble** troca dois elementos subsequentes.

- Já a função **bubbleSort** funciona da seguinte forma: ela vai enviando o maior elemento para o final do vetor através de trocas, quando ela consegue fazer isso (colocar o maior elemento no final do vetor) ela diminui o tamanho do vetor (através da variável qtd) e aplica a mesma técnica no vetor restante. Ao final do processo o vetor está ordenado.

```
void swapbubble( int *v, int i) {
    int aux=0;
    aux=v[i];
    v[i] = v[i+1];
    v[i+1] = aux;
}

 void bubbleSort(int *v, int qtd) {
    int i;
    int trocou;
    do
    {
          qtd--;
          trocou = 0;
          for(i = 0; i < qtd; i++)
          {
              if(v[i] > v[i + 1])
              {
                  swapbubble(v, i);
                  trocou = 1;

              }
          }
    }while(trocou);
}
```

Exemplo

```
void escreveVetor(int *vetor, int n){
    for(int i=0; i<n; i++)
        printf("vet[%d] vale %d \n", i, vetor[i]);
    printf("\n\n");
}


int _tmain(int argc, _TCHAR* argv[])
{
    int vet[5] = {28, 26, 30, 24, 25};
    bubbleSort(vet, 5);
    escreveVetor(vet, 5);
    return 0;
}
```

Exemplo

Primeira varredura:

28	26	30	24	25	compara par (28, 26): troca.
26	28	30	24	25	compara par (28, 30): não troca.
26	28	30	24	25	compara par (30, 24): troca.
26	28	24	30	25	compara par (30, 25): troca.
26	28	24	25	30	fim da primeira varredura.

Exemplo

Segunda varredura:

26	28	24	25	30	compara par (26, 28): não troca.
26	28	24	25	30	compara par (28, 24): troca.
26	24	28	25	30	compara par (28, 25): troca.
26	24	25	28	30	fim da segunda varredura.

Terceira varredura:

26	24	25	28	30	compara par (26, 24): troca.
24	26	25	28	30	compara par (26, 25): troca.
24	25	26	28	30	fim da terceira varredura.

Análise de Desempenho

- Vemos que o laço interno realizará n operações:

```
for(i = 0; i < qtd; i++)
```

- Agora veremos o melhor caso: Os elementos já estão na ordem desejada, assim, ao final da primeira varredura o algoritmo detectará que não foi feita nenhuma troca e portanto o vetor já estará ordenado. Nesse caso, apenas o laço interno é executado, logo o algoritmo será O(n).
- No pior caso os elementos estão na ordem inversa a desejada. Nesse caso o laço externo será executado n vezes. Pela regra do produto nosso algoritmo será então $O(n^2)$ no pior caso.
- O caso médio pode ser considerado Cmedio = (Cpior + Cmelhor) / 2. Isso nos daria $c1 * n + c2 * n^2$, assim, pela regra dos polinômios nosso algoritmo é $O(n^2)$.

Shake Sort

- Ao efetuarmos a análise do desempenho do método ***Bubble Sort***, verificamos que a cada varredura efetuada, a maior das chaves consideradas é levada até sua posição definitiva, enquanto as demais se aproximam da sua posição correta. Isso sugere um aperfeiçoamento do método, no qual, ao final de uma varredura da esquerda para a direita, seja efetuada outra, da direita para esquerda, de tal modo que, ao final desta, a menor chave também se desloque para sua posição definitiva. Essa varreduras em direções opostas devem se alternar sucessivamente até a ordenação completa do vetor. A essa alteração chamamos de ***Shake Sort***.

Programação 2

Shake Sort

```
void shakeSort(int *vetor, int n) {
    int bottom,top, swapped,i,aux;
    bottom = 0;
    top = n - 1;
    swapped = 0;
    while(swapped == 0 && bottom < top)//Se não houver troca de posições ou o ponteiro que
    {                                  //sobe ultrapassar o que desce, o vetor esta ordenado
        swapped = 1;
        //Este for é a "ida" para a direita
        for(i = bottom; i < top; i = i + 1)
        {
            if(vetor[i] > vetor[i + 1])   //indo pra direita:testa se o próximo é maior
            {   //indo pra direita:se o proximo é maior que o atual,
                //troca as posições
                aux=vetor[i];
                vetor[i]=vetor[i+1];
                vetor[i+1]=aux;
                swapped = 0;
            }
        }
        // diminui o  `top` porque o elemento com o maior valor
        // já está na direita (atual posição top)
        top = top - 1;
        //Este for é a "ida" para a esquerda
        for(i = top; i > bottom; i = i - 1)
        {  if(vetor[i] < vetor[i - 1])
            {
                aux=vetor[i];
                vetor[i]=vetor[i-1];
                vetor[i-1]=aux;
                swapped = 0;
            }
        }
        //aumenta o `bottom` porque o menor valor já está
        //na posição inicial (bottom)
        bottom = bottom + 1;
    }
 }
```

Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

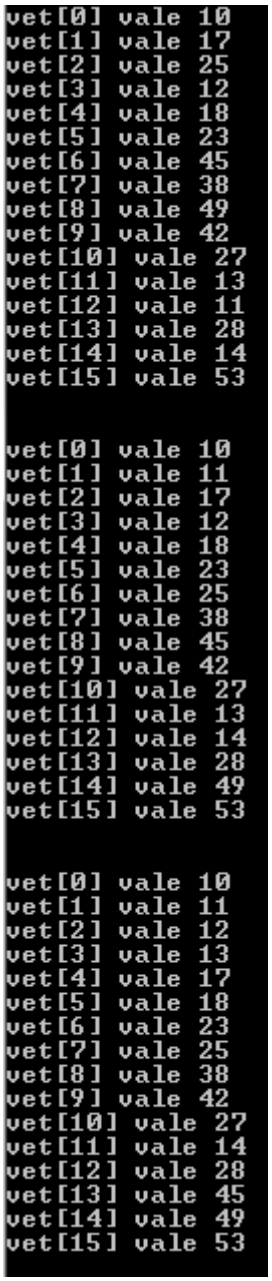
Exemplo

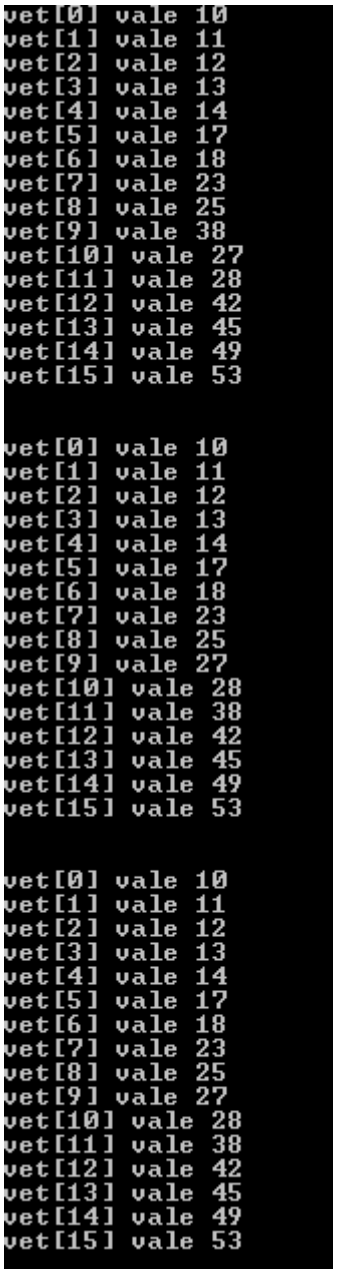
```
void escreveVetor(int *vetor, int n){
    for(int i=0; i<n; i++)
        printf("vet[%d] vale %d \n", i, vetor[i]);
    printf("\n\n");
}

int _tmain(int argc, _TCHAR* argv[])
{
    int vet[16] = {17, 25, 49, 12, 18, 23, 45, 38, 53, 42, 27, 13, 11, 28, 10, 14};
    shakeSort(vet, 16);
    escreveVetor(vet, 16);
    return 0;
}
```

Exemplo

- Se executarmos o exemplo proposto teremos o resultado das imagens ao lado:
- Para obter esse resultados imprimimos o vetor após cada varredura (uma para direita e uma para a esquerda). Notar que o 10 e o 53 estão em suas posições desde a primeira execução do código.





Análise de Desempenho

- A melhoria obtida reside apenas na quantidade de comparações efetuadas, uma vez que, o método evita muitas das comparações supérfulas que o *Bubble Sort* efetua.
- Já o número de trocas necessárias não se altera em relação ao *Bubble Sort*.
- Como o número de trocas é mais oneroso que as comparações efetuadas, o ganho de tempo não será significativo. Assim, as complexidades desse método se mantem em relação ao *Bubble Sort*.

Comb Sort

- O algoritmo *Comb Sort* é outra variação do *Bubble Sort*, nele a comparação não é feita entre pares consecutivos de elementos, mas pares formados por chaves que distam umas das outras uma certa distância h.
- h é determinado em função de n e experimentalmente tem seu melhor resultado obedecendo a seguinte expressão: ***h = n div 1,3***
  - *Onde div é a divisão desprezando-se as cadas decimais após a vírgula)*

Comb Sort

- Experimentalmente também foi detectado que, quando h = 9 ou h = 10 deve-se forçar seu valor para h = 11.
- Notem que quando h = 1 o método se confunde com o *Bubble Sort*.

Comb Sort

```
#include "stdafx.h"

#define F 0;
#define V 1;

int max(int a, int b){
    if (a > b) return a;
    else return b;
}
```

Comb Sort

```
void combSort(int *vet, int n) {
      int h = n;
      int troca;
      int aux;
      do{
           h = max( (int)(h/3), 1);
           if ((h == 9) || (h == 10)) h = 11;
           troca = F;
           for(int i=0; i<n-h; i++){
                    if (vet[i] > vet[i+h]){
                        aux = vet[i];
                        vet[i] = vet[i + h];
                        vet[i + h] = aux;
                        troca = V;
                    }
           }

      } while (troca || (h!=1));
}
```

Exemplo

```
void escreveVetor(int *vet, int n){
     for(int i=0; i<n; i++)
               printf("vet[%d] vale %d \n", i, vet[i]);
     printf("\n\n");
}


int _tmain(int argc, _TCHAR* argv[])
{
     int vet[5] = {28, 26, 30, 24, 25};
     combSort(vet, 5);
     escreveVetor(vet, 5);
     return 0;
}
```

Exemplo

varre-dura	itera-ção	vetor de chaves	salto	par comparado	ação
1	1	**28** 26 30 **24** 25	3	28, 24	troca
	2	24 **26** 30 28 **25**	3	26, 25	troca
2	3	**24** 25 **30** 28 26	2	24, 30	não troca
	4	24 **25** 30 **28** 26	2	25, 28	não troca
	5	24 25 **30** 28 **26**	2	30, 26	troca
3	6	**24** **25** 26 28 30	1	24, 25	não troca
	7	24 **25** **26** 28 30	1	25, 26	não troca
	8	24 25 **26** **28** 30	1	26, 28	não troca
	9	24 25 26 **28** **30**	1	28, 30	não troca

Análise de Desempenho

- A redução do tempo de classificação desse método em relação ao *Bubble Sort* tradicional (sem qualquer tipo de otimização) foi da ordem de 27 vezes.
- As sucessivas reduções dos saltos são análogas a pentear cabelos longos e embaraçados, inicialmente apenas com os dedos e usando pentes com espaços entre os dentes cada vez menores. Daí a denominação do método pelos autores.
- Com relação a ordem de complexidade esse algoritmo compartilha da mesma dificuldade encontrada no *Shell Sort.* Assim, optaremos por utilizar os limites do *Bubble Sort*, ou seja, $O(n^2)$ no pior caso.

Bibliografia

- AZEREDO, Paulo A. **Métodos de Classificação de Dados**. Rio de Janeiro: Campus,1996.
- SANTOS, Henrique José. **Curso de Linguagem C da UFMG**, apostila.
- FORBELLONE, André Luiz. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** São Paulo: MAKRON, 1993.