Aula 18: Laboratório
- Métodos simples de ordenação (parte 4)

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

3Q-2017

Atividade 01: Benchmark

Vetores ordenados

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[i] = i;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[n-i-1] = i;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

Vetor aleatório

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    srand(time(NULL));
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[i] = rand();
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

Vetor Parcialmente Ordenados

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    srand(time(NULL));

    long int i, j, aux;
    long int n = atoi(argv[1]);
    long int vetor[n];

    // vetor ordenado
    for (i=0; i<n; i++) {
        vetor[i] = i;
    }

    // vetor quase ordenado
    int k = 50;

    for (i=0; i<=n-k; i++) {
        j = i+rand()%k;
        aux = vetor[i];
        vetor[i] = vetor[j];
        vetor[j] = aux;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    srand(time(NULL));

    long int i, j, aux;
    long int n = atoi(argv[1]);
    long int vetor[n];

    // vetor ordenado
    for (i=0; i<n; i++) {
        vetor[n-i-1] = i;
    }

    // vetor quase ordenado
    int k = 50;

    for (i=0; i<=n-k; i++) {
        j = i+rand()%k;
        aux = vetor[i];
        vetor[i] = vetor[j];
        vetor[j] = aux;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

Benchmark

```
1  # vetores aleatorios
2  gcc gerarVetorAleatorio.c -o gerarVetorAleatorio
3  ./gerarVetorAleatorio 50000 > vetorAleatorio50000.dat
4
5  # vetores crescente
6  gcc gerarVetorCrescente.c -o gerarVetorCrescente
7  ./gerarVetorCrescente 50000 > vetorCrescente50000.dat
8
9  # vetores decrescente
10 gcc gerarVetorDecrescente.c -o gerarVetorDecrescente
11 ./gerarVetorDecrescente 50000 > vetorDecrescente50000.dat
12
13 # vetores parcialmente ordenados de forma crescente
14 gcc gerarVetorParcialmenteOrdenadoCrescente.c -o gerarVetorParcialmenteOrdenadoCrescente
15 ./gerarVetorParcialmenteOrdenadoCrescente 50000 > vetorPCrescente50000.dat
16
17 # vetores parcialmente ordenados de forma decrescente
18 gcc gerarVetorParcialmenteOrdenadoDecrescente.c -o gerarVetorParcialmenteOrdenadoDecrescente
19 ./gerarVetorParcialmenteOrdenadoDecrescente 50000 > vetorPDecrescente50000.dat
```

```
$ sh -v benchmark.sh
```

Algoritmos de ordenação

Selection Sort

```
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i=i+1) {
        iMin = i;

        for (j=i+1; j<n; j=j+1) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux     = v[iMin];
            v[iMin] = v[i];
            v[i]    = aux;
        }
    }
}
```

Insertion Sort

```
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i=i+1) {
        aux = v[i];

        for (j=i-1; j>=0 && v[j]>aux; j=j-1) {
            v[j+1] = v[j];
        }

        v[j+1] = aux;
    }
}
```

Bubble Sort (2 versões)

```c
void BubbleSort1 (int v[], int n) {
    int k, i, aux;

    for (k=n-1; k>=1; k=k-1) {
        for (i=0; i<k; i=i+1) {
            if (v[i]>v[i+1]) {
                aux = v[i];
                v[i] = v[i+1];
                v[i+1] = aux;
            }
        }
    }
}
```

```c
void BubbleSort2 (int v[], int n) {
    int i, aux, hasChanged;

    do {
        hasChanged = 0;
        for (i=0; i<n-1; i=i+1) {
            if (v[i]>v[i+1]) {
                aux = v[i];
                v[i] = v[i+1];
                v[i+1] = aux;
                hasChanged = 1;
            }
        }
    }while(hasChanged==1);
}
```

Cocktail sort

```
void cocktailSort(int v[], int n) {
    int i, t;
    int trocou = 1; // 1:True, 0:False
    int inicio = 0;
    int fim    = n-1;

    while (trocou==1) {
        trocou = 0;
        for (i=inicio; i<fim; i++) {
            if (v[i] > v[i+1]) {
                t      = v[i];
                v[i]   = v[i+1];
                v[i+1] = t;
                trocou = 1;
            }
        }
        fim--;

        if (trocou==0)
            break;

        trocou = 0;
        for (i=fim-1; i>=inicio; i--) {
            if (v[i] > v[i+1]) {
                t      = v[i];
                v[i]   = v[i+1];
                v[i+1] = t;
                trocou = 1;
            }
        }
        inicio++;
    }
}
```

Atividade

- Tempo de processamento (segundos)

Teste empírico

```c
int main(int argc, char *argv[])
{
    clock_t t1, t2;

    int i;
    int n = atoi(argv[1]);
    int vetor[n];

    // leitura do vetor a ser ordenado
    for (i=0; i<n; i++)
        scanf("%d", &vetor[i]);

    // calculo do tempo para ordenar o vetor
    t1 = clock();
        SelectionSort (vetor, n);
    t2 = clock();

    printf("%.6f\n", (double)(t2-t1)/CLOCKS_PER_SEC);
}
```

```
NAME
       clock - determine processor time

SYNOPSIS
       #include <time.h>

       clock_t clock(void);

DESCRIPTION
       The  clock()  function  returns an approximation of processor time used by the
       program.

RETURN VALUE
       The value returned is the CPU time used so far as a clock_t; to get the number
       of  seconds used, divide by CLOCKS_PER_SEC.  If the processor time used is not
       available or its value cannot be represented, the function returns  the  value
       (clock_t) -1.
```

Atividade 02: Teste empírico

```
# Testando os algoritmos de ordenacao
gcc insertionSort.c -o insertionSort.exe
gcc selectionSort.c -o selectionSort.exe
gcc bubbleSort.c -o  bubbleSort.exe

./selectionSort.exe 50000 < vetorAleatorio50000.dat
./insertionSort.exe 50000 < vetorAleatorio50000.dat
./bubbleSort.exe    50000 < vetorAleatorio50000.dat

./selectionSort.exe 50000 < vetorCrescente50000.dat
./insertionSort.exe 50000 < vetorCrescente50000.dat
./bubbleSort.exe    50000 < vetorCrescente50000.dat

./selectionSort.exe 50000 < vetorDecrescente50000.dat
./insertionSort.exe 50000 < vetorDecrescente50000.dat
./bubbleSort.exe    50000 < vetorDecrescente50000.dat

./selectionSort.exe 50000 < vetorPCrescente50000.dat
./insertionSort.exe 50000 < vetorPCrescente50000.dat
./bubbleSort.exe    50000 < vetorPCrescente50000.dat

./selectionSort.exe 50000 < vetorPDecrescente50000.dat
./insertionSort.exe 50000 < vetorPDecrescente50000.dat
./bubbleSort.exe    50000 < vetorPDecrescente50000.dat
```

Atualizar a seguinte tabela com os testes

https://docs.google.com/spreadsheets/d/1QYzRwqCJdevqiYXc1NVDVdHObR9SVrXkYXQHD554RQo/edit?usp=sharing

- http://sortvis.org/

- http://sorting.at/