Bacharelado em Ciência e Tecnologia

Algoritmos e Estrutura de Dados I

UFABC

Universidade Federal do ABC

Projeto Prático

Prof. André G. R. Balan

andre.balan@ufabc.edu.br

Objetivo

- Realizar um estudo empírico da complexidade de tempo dos algoritmos de ordenação interna estudados

Objetivo

- Algoritmos considerados:
  - Eficientes
    - HeapSort
    - QuickSort (com pivô mediano)
    - MergeSort
    - Sort C++
    - Quicksort C

Metodologia

- Inicialmente:
  - Criar uma classe em C++ para ser uma biblioteca de métodos de ordenação, utilizando *templates* e métodos estáticos. Exemplo:

ordenacao.hpp

```
#ifndef _ORDENACAO_H
#define _ORDENACAO_H

namespace ED {

    template <typename T>
    class Ordenacao {
        public:
            static void heapsort(T *vet, int n);
            . . .
        private:
            static void heapify(T *vet, int pai, int heapsize);
            . . .
    };

    // Implementações
}
#endif
```

Metodologia

- Exemplo

ordenacao.hpp

```
// Implementações

template <typename T>
void Ordenacao<T>::heapify(T *vet, int pai, int heapsize) {
     int fl, fr, imaior;
     fl = (pai << 1) + 1;
     fr = fl + 1;
     while (true) {
         if ((fl < heapsize) && (vet[fl] > vet[pai])) imaior = fl;
         else imaior = pai;
         if ((fr < heapsize) && (vet[fr] > vet[imaior]))  imaior = fr;
         if (imaior != pai){
             T aux = vet[pai];
             vet[pai] = vet[imaior];
             vet[imaior] = aux;
             pai =  imaior;
             fl  = (pai << 1) + 1;
             fr  =  fl + 1;
         }
         else  break;
     }
}
```

Metodologia

- Exemplo

ordenacao.hpp

```
// Implementações

template <typename T>
void Ordenacao<T>::heapsort(T *vet, int n) {
    int i;
    for (i=(n>>1)-1; i>=0; i--) heapify(vet, i, n);
    for (i=n-1; i > 0; i--) {
        T aux = vet[0];
        vet[0] = vet[i];
        vet[i] = aux;
        heapify(vet, 0, i);
    }
}
```

Metodologia

- De maneira simplificada, um método estático é um método que pode ser chamado diretamente pela classe, sem ser preciso instanciar um objeto daquela classe. Exemplo:

main.cpp

```
#include "ordenacao.h"


int main(int argc, char** argv) {
      int n = 1000;
      float *v = new float[n];

      for (int i=0; i<n; i++)   v[i] = n-i;

      ED::Ordenacao<float>::heapsort(v, n);

      for (int i=0; i<n; i++)   cout << v[i] << endl;
      .
      .
}
```

Metodologia

- Após criar a classe biblioteca de métodos, criar um programa para realizar o estudo empírico dos métodos.

- **Cada método** deve ser testado na ordenação de vetores de elementos do tipo **int**, dos seguintes tamanhos:
  - 10.000
  - 30.000
  - 90.000
  - 270.000
  - 810.000
  - 2.430.000
  - 7.290.000
  - 21.870.000
  - 65.610.000

Metodologia

- Os vetores são compostos por números inteiros, **com distribuição uniforme**
- **Para cada tamanho de vetor** definido, o programa deve rodar **cada método** com **6 sequências aleatórias diferentes**:
- 9 métodos x 9 tamanhos x 6 sequências = **486** ordenações
- Antes, uma visão geral sobre **números aleatórios no computador**

Metodologia

- Na verdade, os computadores geram números ***pseudoaleatórios***

- Toda função para criar números aleatórios, cria uma sequência de números do tipo

$$r_i = \begin{cases} seed, & para \quad i = 0 \\ f(r_{i-1}) & para \quad i > 0 \end{cases}$$

Metodologia

- Sendo assim, toda a sequência depende do valor inicial *seed*

- Assim, se quisermos criar duas sequências aleatórias idênticas em tempos diferentes, basta definirmos o mesmo valor de *seed* para as sequências

Metodologia

- Usar o gerador de números aleatórios do **C++11**

```
#include <iostream>
#include <random>

using namespace std;

int main() {

    uniform_int_distribution<int32_t> uidist;


    mt19937 rng;


    rng.seed(127);

    for(int i=0; i<10; i++)
        cout << uidist(rng) << endl;
}
```

Metodologia

- 6 Sementes:
  - seed[0] = 4
  - seed[1] = 81
  - seed[2] = 151
  - seed[3] = 1601
  - seed[4] = 2307
  - seed[5] = 4207

Resultados

- Para cada ordenação o programa deve medir:
  - O tempo total em milissegundos

- Cada par (método, tamanho de vetor) irá ter 6 medidas de tempo

Medindo tempo em milissegundos

```
#include <time.h>


int start;
int tmili;

start = clock();

  // Operações

tmili = (int)((clock()-start)*1000/CLOCKS_PER_SEC);

 cout << tmili << endl;
```

clock() retorna o número de pulsos de clock do processador desde que o computador iniciou.

Resultados

- Após desenvolver o programa, chegou a hora de colocar pra rodar e ir dormir.

- Deixe apenas o seu programa rodar. Não faça nada no computador enquanto o experimento estiver rodando (não mexa nem o mouse)

- O programa deve lhe retornar todos os dados necessários para você realizar o estudo empírico.

Resultados

- Com todos os dados obtidos, você deve elaborar um relatório contendo
  - **Gráficos:** (faça no excel, por exemplo) que permitam uma comparação fácil e rápida entre os métodos.

Resultados

- Gráficos de tempo:
  - Em um único gráfico, coloque várias curvas de tempo, uma para cada método. Cada ponto das curvas de tempo, é uma média aritmética de 6 medidas de tempo, uma para sequência aleatória.
  - Veja imagem a seguir

Resultados

- Modelo de gráfico a ser seguido: