Vetores: Algoritmos de Pesquisa

Algoritmos e Estruturas de Dados

2020/2021

FEUP

Introdução

- Algoritmo: conjunto claramente especificando instruções a seguir para resolver um problema
  - noção de algoritmo muito próxima da noção de programa (imperativo)
  - o mesmo algoritmo pode ser "implementado" por programas de computador diferentes
  - o mesmo problema pode ser resolvido por algoritmos diferentes

- Descrição de algoritmos:
  - em linguagem natural, pseudo-código, numa linguagem de programação, etc.

Pesquisa Sequencial

- Problema *(pesquisa de valor em vetor)*: verificar se um valor existe no vetor e, no caso de existir, indicar a sua posição
  - variantes para o caso de vetores com valores repetidos:
    - (a) indicar a posição da primeira ocorrência
    - (b) indicar a posição da última ocorrência
    - (c) indicar a posição de uma ocorrência qualquer

- Algoritmo *(pesquisa sequencial)*: percorrer sequencialmente todas as posições do vetor, da primeira para a última (a) ou da última para a primeira (b), até encontrar ovalor pretendido ou chegar ao fim do vetor
  - (a) caso se pretenda saber a posição da primeira ocorrência
  - (b) caso se pretenda saber a posição da última ocorrência

- Adequado para vetores não ordenados ou pequenos

Implementação Pesquisa Sequencial em C++

- Caso de vetores de inteiros, na variante (a):

```
/* Procura um valor x num vetor v de inteiros. Retorna o índice da
primeira ocorrência de x em v, se encontrar; senão, retorna -1. */

int SequentialSearch(const vector<int> &v, int x)
{
   for (unsigned int i = 0; i < v.size(); i++)
      if (v[i] == x)
         return i;   // encontrou

   return -1;         // não encontrou
}
```

  - Como generalizar para vetores de elementos de qualquer tipo?
    - Com *templates* de funções!

Implementação Genérica da Pesquisa Sequencial em C++

- *Template* de função em C++, na variante (a):

```
/* Procura um valor x num vetor v de elementos comparáveis com os
operadores de comparação. Retorna o índice da primeira ocorrência
de x em v, se encontrar; senão, retorna -1. */

template <class T>
int SequentialSearch(const vector<T> &v, T x)
{
   for (unsigned int i = 0; i < v.size(); i++)
      if (v[i] == x)
         return i; // encontrou

   return -1; // não encontrou
}
```

Eficiência da Pesquisa Sequencial

- Eficiência temporal de SequentialSearch
  - A operação realizada mais vezes é o teste da condição de continuação do ciclo **for**, no máximo **n+1** vezes (no caso de não encontrar **x**).
  - Se **x** existir no vetor, o teste é realizado aproximadamente **n/2** vezes em média (1 vez no melhor caso)
  - **T(n) = *O(n)*** (linear) no pior caso e no caso médio

- Eficiência espacial de SequentialSearch
  - Gasta o espaço das variáveis locais (incluindo argumentos)
  - Como os vetores são passados "por referência" (de facto o que é passado é o endereço do vetor), o espaço gasto pelas variáveis locais é constante e independente do tamanho do vetor
  - **S(n) = *O(1)*** (constante) em qualquer caso

Exemplo de aplicação: totoloto

```
// Programa para gerar aleatoriamente uma aposta do totoloto

#include <iostream.h>
#include <stdlib.h>   // para usar rand() e srand()
#include <time.h>     // para usar time()
#include <vector.h>

// Constantes
const unsigned int TAMANHO_APOSTA = 6;
const unsigned int MAIOR_NUMERO = 49;

// Gera inteiro aleatório entre a e b
int rand_between(int a, int b)
{
   return (rand() % (b - a + 1)) + a;
   // Nota: rand() gera um inteiro entre 0 e RAND_MAX
}
```

Exemplo de aplicação: totoloto (cont.)

```
// Verifica se existe o valor x no vetor v
// (inline: explicado adiante)
inline bool existe(int x, const vector<int> &v)
{
   return SequentialSearch(v, x) != -1;
}

// Preenche a aposta ap com valores aleatórios sem repetições
void geraAposta(vector<int> &ap)
{
    int num;
    for (unsigned int i = 0; i < TAMANHO_APOSTA; i++) {
       do
          num = rand_between(1, MAIOR_NUMERO);
       while ( existe(num, ap) );
       ap.push_back(num);
    }
}
```

Exemplo de aplicação: totoloto (concl.)

```
// Imprime a aposta ap
void imprimeAposta(const vector<int> &ap)
{
    cout << "A aposta gerada é: ";
    for (unsigned int i = 0; i < ap.size(); i++)
       cout << ap[i] << ' ';
    cout << endl;
}

int main()
{
    vector<int> aposta;
    srand(time(0)); /* inicializa gerador de nºs pseudo-
                       aleatórios com valor diferente de
                       cada vez que o programa corre */
    geraAposta(aposta);
    imprimeAposta(aposta);
    return 0;
}
```

Nota sobre funções `inline` em C++

- Qualificador **`inline`** antes do nome do tipo retornado por uma função: "aconselha" o compilador a gerar uma cópia do código da função no lugar em que é chamada
- Evita o peso do mecanismo de chamada de funções

- Exemplo: estando definida uma função

```
inline int max(int a, int b)
{ return a > b? a : b; }
```

  o compilador pode converter uma chamada do género

```
x = max(z, y);
```

  em algo do género:

```
x = z > y? z : y;
```

- Usar só com funções pequenas, porque o código da função é replicado em todos os sítios em que a função é chamada!

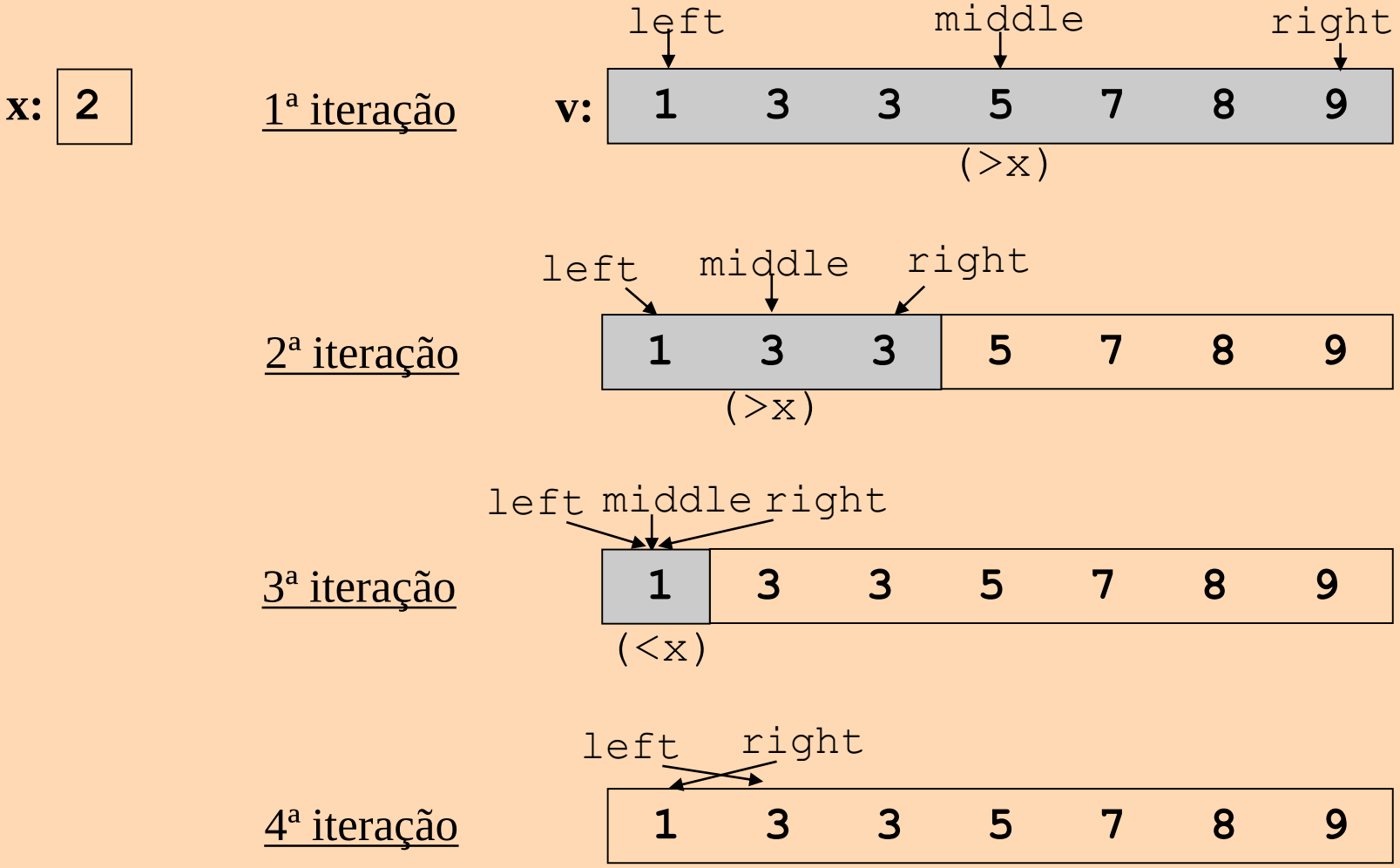
Pesquisa Binária

- Problema (*pesquisa de valor em array ordenado*): verificar se um valor (*x*) existe num vetor (*v*) previamente ordenado e, no caso de existir, indicar a sua posição
  - no caso de vetores com valores repetidos, consideramos a variante em que basta indicar a posição de uma ocorrência qualquer (as outras ocorrências do mesmo valor estão em posições contíguas)

- Algoritmo (*pesquisa binária*):
  - Comparar o valor que se encontra a meio do vetor com o valor procurado, podendo acontecer uma de três coisas:
    - é igual ao valor procurado⇒ está encontrado
    - é maior do que o valor procurado ⇒ continuar a procurar (do mesmo modo) no sub-vetor à esquerda da posição inspeccionada
    - é menor do que o valor procurado ⇒ continuar a procurar (do mesmo modo) no sub-vetor à direita da posição inspeccionada.
  - Se o vetor a inspecionar se reduzir a um vetor vazio, conclui-se que o valor procurado não existe no vetor inicial.

Exemplo de Pesquisa Binária

x: 2



vetor a inspeccionar vazio ⇒ o valor 2 não existe no vetor inicial!

Implementação da Pesquisa Binária em C++

```
/* Procura um valor x num vetor v previamente ordenado. Retorna o índice
de uma ocorrência de x em v, se encontrar; senão, retorna -1. Supõe que os
elementos do vetor são comparáveis com os operadores de comparação. */

template <class T>
int BinarySearch(const vector<T> &v, T x)
{
    int left = 0, right = v.size() - 1;
    while (left <= right)
    {
        int middle = (left + right) / 2;
        if (v[middle] < x)
            left = middle + 1;
        else if (x < v[middle)
            right= middle - 1;
        else
            return middle; // encontrou
    }
    return -1; // não encontrou
}
```

Eficiência Temporal da Pesquisa Binária

- Em cada iteração, o tamanho do sub-vetor a analisar é dividido por um factor de aproximadamente 2
- Ao fim de **k** iterações, o tamanho do sub-vetor a analisar é aproximadamente $\mathbf{n / 2^k}$
- Se não existir no vetor o valor procurado, o ciclo só termina quando $\mathbf{n / 2^k \approx 1} \Leftrightarrow \log_2 n - k \approx 0 \Leftrightarrow \mathbf{k \approx \log_2 n}$

- Assim, no pior caso, o nº de iterações é aproximadamente $\log_2 n$

  $\Rightarrow$ **T(n) = *O(log n)*** (logarítmico)

- É muito mais eficiente que a pesquisa sequencial, mas só é aplicável a vetores ordenados!

Algoritmos da STL

- Pesquisa sequencial em vetores:

  iterator find(iterator start, iterator end, const T& val);

  procura 1ª ocorrência em *[start, end[* de um elemento idêntico a *val* (comparação efectuada pelo operador ==)

  - sucesso, retorna iterador para o elemento encontrado
  - não sucesso, retorna iterador para fim do vetor (*v.end()*)

  iterator find_if(iterator start, iterator end, Predicate pred);

  procura 1ª ocorrência em *[start, end[* para a qual o predicado unário *pred* é verdadeiro

- Pesquisa binária em vetores:

  bool binary_search(iterator start, iterator end, const T& val);

  - usa operador <

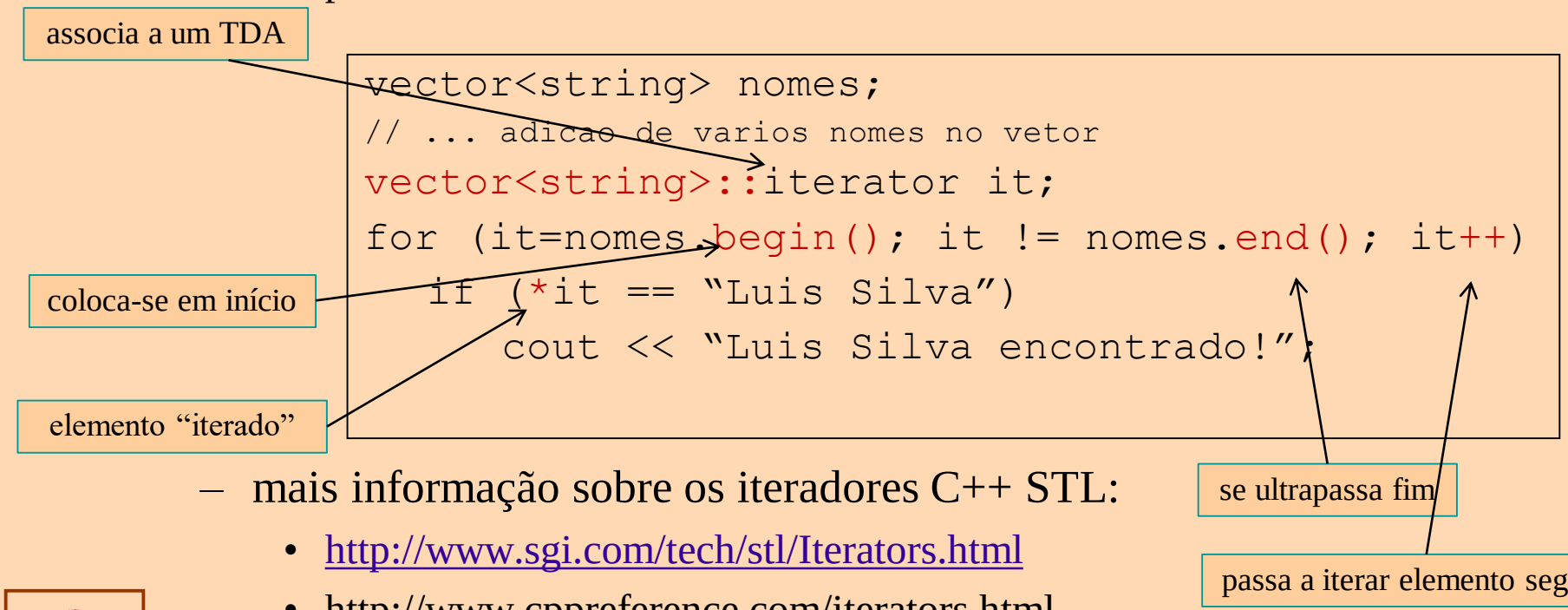
  bool binary_search(iterator start, iterator end, const T& val, Compare comp);

  - usa predicado *comp* (*comp* deve comparar dois valores)

Nota: Iteradores

Iterador

- associa-se a um Tipo de Dados Abstractos ou a uma sua implementação
- *Exemplo* de uso de iteradores em vetores:
  - considere o vetor **nomes** (vetor de strings: nomes de pessoas, p.ex)
  - procurar o nome "Luis Silva" no vetor **nomes**




```
vector<string> nomes;
// ... adicao de varios nomes no vetor
vector<string>::iterator it;
for (it=nomes.begin(); it != nomes.end(); it++)
    if (*it == "Luis Silva")
        cout << "Luis Silva encontrado!";
```

- mais informação sobre os iteradores C++ STL:
  - http://www.sgi.com/tech/stl/Iterators.html
  - http://www.cppreference.com/iterators.html

Algoritmo *find* da STL (exemplo)

```
class Pessoa {
    string BI;
    string nome;
    int idade;
public:
    Pessoa (string BI, string nm="", int id=0);
    string getBI() const;
    string getNome() const;
    int getIdade() const;
    bool operator < (const Pessoa & p2) const;
    bool operator == (const Pessoa & p2) const;
};

Pessoa::Pessoa(string b, string nm, int id):
        BI(b),nome(nm), idade(id) {}

string Pessoa::getBI() const { return BI; }
string Pessoa::getNome() const { return nome; }
int Pessoa::getIdade() const { return idade; }
```

Algoritmo *find* da STL (exemplo)

```
bool Pessoa::operator < (const Pessoa & p2) const
{ return nome < p2.nome; }

bool Pessoa::operator == (const Pessoa & p2) const
{ return BI == p2.BI; }


ostream & operator << (ostream &os, const Pessoa & p)
{
   os << "(BI: " << p.getBI() << ", nome: " << p.getNome()
   << ", idade: " << p.getIdade() << ")";
   return os;
}
```

Algoritmo *find* da STL (exemplo)

```
template <class T>
void write_vector(vector<T> &v) {
   for (unsigned int i=0 ; i < v.size(); i++)
        cout << "v[" << i << "] = " << v[i] << endl;
   cout << endl;
}


bool eAdolescente(const Pessoa &p1) {
   return p1.getIdade() <= 20; }


bool menorIdade(const Pessoa &p1, const Pessoa &p2) {
   if (p1.getIdade()<p2.getIdade()) return true;
   else return false;
}
```

Algoritmo *find* da STL (exemplo)

```
int main()
{
  vector<Pessoa> vp;
  vp.push_back(Pessoa("6666666","Rui Silva",34));
  vp.push_back(Pessoa("7777777","Antonio Matos",24));
  vp.push_back(Pessoa("1234567","Maria Barros",20));
  vp.push_back(Pessoa("7654321","Carlos Sousa",18));
  vp.push_back(Pessoa("3333333","Fernando Cardoso",33));

  cout << "vetor inicial:" << endl;
  write_vector(vp);
```

vetor inicial:
v[0] = (BI: 6666666, nome: Rui Silva, idade: 34)
v[1] = (BI: 7777777, nome: Antonio Matos, idade: 24)
v[2] = (BI: 1234567, nome: Maria Barros, idade: 20)
v[3] = (BI: 7654321, nome: Carlos Sousa, idade: 18)
v[4] = (BI: 3333333, nome: Fernando Cardoso, idade: 33)

Algoritmo *find* da STL (exemplo)

```
Pessoa px("7654321");
vector<Pessoa>::iterator it = find(vp.begin(),vp.end(),px);
if (it==vp.end())
   cout << "Pessoa " << px << " não existe no vetor " << endl;
else
   cout<< "Pessoa "<< px << " existe no vetor como:" << *it <<endl;
```

Pessoa (BI: 7654321, nome: , idade: 0) existe no vetor como: (BI: 7654321, nome: Carlos Sousa, idade: 18)

```
it = find_if(vp.begin(),vp.end(),eAdolescente);
if (it==vp.end())
  cout << "Pessoa adolescente nao existe no vetor " << endl;
else
  cout << "pessoa adolescente encontrada: " << *it << endl;
```

pessoa adolescente encontrada: (BI: 1234567, nome: Maria Barros, idade: 20)

Algoritmo *find* da STL (exemplo)

```
  // note que vetor vp2 está ordenado por idade
vector<Pessoa> vp2;
vp2.push_back(Pessoa("7654321","Carlos Sousa",18));
vp2.push_back(Pessoa("1234567","Maria Barros",20));
vp2.push_back(Pessoa("7777777","Antonio Matos",24));
vp2.push_back(Pessoa("3333333","Fernando Cardoso",33));
vp2.push_back(Pessoa("6666666","Rui Silva",34));

Pessoa py("xx","xx",24);
bool existe = binary_search(vp2.begin(),vp2.end(),py,menorIdade);
if (existe==true)
   cout << "Existe pessoa de idade " << py.getIdade() << endl;
else
   cout << "Não existe pessoa de idade" << py.getIdade()<<  endl;
```

Existe pessoa de idade 24