ALGORITMOS II

5ª LISTA DE EXERCÍCIOS – ALGORITMOS DE BUSCA

"Binary search is to algorithms what a wheel is to mechanics:
It is simple, elegant, and immensely important."
—Udi Manber, *Introduction to Algorithms*

"A good algorithm is like a sharp knife:
it does what it is supposed to do with a minimum amount of applied effort.
Using the wrong algorithm to solve a problem is like trying to cut a steak with a screwdriver:
you may eventually get a digestible result,
but you will expend considerably more effort than necessary,
and the result is unlikely to be aesthetically pleasing."
—Th. Cormen, Ch. Leiserson, R. Rivest, *Introduction to Algorithms*

"Binary search is a notoriously tricky algorithm to program correctly.
It took seventeen years after its invention until
the first correct version of binary search was published!"
—Steven Skiena, *The Algorithm Design Manual*

Considere a seguinte função que faz uma busca seqüencial em um vetor previamente ordenado:

```
int  busca (int x, int n, int v[]) {
  int j = 0;
  while (j < n && v[j] < x) ++j;
  return j;
}
```

1 Critique a seguinte formulação do problema de busca: dado x e um vetor *crescente* v[0..n-1], encontrar um índice j tal que v[j-1] ≤ x ≤ v[j]. Critique a formulação construída em torno de "v[j-1] < x < v[j]".

2 Critique a seguinte versão da função:

```
int busca (int x, int n, int v[]) {
  int j = 0;
  while (v[j] < x && j < n) ++j;
  return j;
}
```

3 Discuta a seguinte versão recursiva da função busca:

```
// Esta função devolve j em 0..n tal que
// v[j-1] < x <= v[j]. Ela supõe que n >= 0.
int busca2 (int x, int n, int v[]) {
  if (n == 0) return 0;
  if (x > v[n-1]) return n;
  return busca2 (x, n-1, v);
}
```

Considere a seguinte função que faz uma busca binária em um vetor previamente ordenado:

```
// A função abaixo recebe um vetor crescente v[0..n-1]
// com n >= 1 e um número x.
// Ela devolve um índice j em 0..n tal que v[j-1] < x <= v[j].
int buscabinaria (int x, int n, int v[]) {
  int e, m, d;
  if (v[n-1] < x) return n;
  if (x <= v[0]) return 0;  // agora v[0] < x <= v[n-1]
  e = 0; d = n-1;
  while (e < d-1) {
    m = (e + d)/2;
    if (v[m] < x) e = m;
    else d = m;
  }
  return d;
}
```

4 Responda as seguintes perguntas sobre a função *buscabinaria* descrita acima.

a) Que acontece se "*while (e < d-1)*" for trocado por "*while (e < d)*"? ou por "while (e <= d-1)"?
b) Que acontece se "*if (v[m] < x)*" for trocado por "*if (v[m] <= x)*"?
c) Que acontece se "*e = m*" for trocado por "*e = m+1*" ou por "*e = m-1*"? E se "*d = m*" for trocado por "*d = m+1*" ou por "*d = m-1*"?

5 Execute *buscabinaria* com n = 9, com v[i] = i para cada i e com vários valores de x. Repita o exercício com n = 14 e x = 9.

6 Se preciso de *t* segundos para fazer uma busca binária em um vetor com *n* elementos, de quando tempo preciso para fazer uma busca em n^2 elementos?

7 Escreva uma função de busca binária simplificada que devolva *j* tal que *v[j] == x* ou devolva *-1* se tal *j* não existe.

Considere a função que faz uma busca binária (uma versão mais limpa) em um vetor previamente ordenado:

```
int buscabinaria2 (int x, int n, int v[]) {
  int e, m, d;
  e = -1; d = n;
  while (e < d-1) {
    m = (e + d)/2;
    if (v[m] < x) e = m;
    else d = m;
  }
  return d;
}
```

8 É verdade que m está em 0..n-1 sempre que a instrução *if (v[m] < x)* é executada?

9 Execute *buscabinaria2* com n = 7, com v[i] = i para cada i e com vários valores de x.

10 Execute a função *buscabinaria2* com n = 15, com v[i] = i para cada i e com vários valores de x.

11 Confira a validade da seguinte afirmação: quando n+1 é uma potência de 2, a expressão (e+d) é sempre divisível por 2, quaisquer que sejam v e x.

12 Execute a função *buscabinaria2* com n = 16. Quais os possíveis valores de m na primeira iteração? Quais os possíveis valores de m na segunda iteração? Na terceira? Na quarta?

13 Escreva uma versão da busca binária que receba um número x e um vetor v[0..n-1] e devolva *j* tal que em $v[j-1] \leq x < v[j]$. (Quais os possíveis valores de j?).

14 Escreva uma versão da busca binária que procure x em v[0..n] (atenção para os índices).

15 Escreva uma versão da busca binária que procure x em v[1..n] (atenção para os índices).

16 Mostre que a seguinte alternativa de *buscabinaria2* funciona corretamente. Ela é quase tão bonita quanto a versão discutida acima.

```
e = 0; d = n;
while (e < d)  // v[e-1] < x <= v[d]
{
   m = (e + d)/2;
   if (v[m] < x) e = m+1;
   else d = m;
}              // e == d
return d;
```

Que acontece se trocarmos "*while (e < d)*" por "*while (e <= d)*"? Que acontece se trocarmos "*(e+d)/2*" por "*(e-1+d)/2*"?

17 Mostre que a seguinte alternativa de *buscabinaria2* funciona corretamente. Ela é um pouco "mais feia" que as versões anteriores.

```
e = 0; d = n-1;
while (e <= d)         // v[e-1] < x <= v[d+1]
{
  m = (e + d)/2;
  if (v[m] < x) e = m+1;
  else d = m-1;
}          // e == d+1
return d+1;
```

18 A seguinte alternativa de *buscabinaria2* funciona corretamente?

```
e = -1; d = n-1;
while (e < d)
{
  m = (e + d)/2;
  if (v[m] < x) e = m;
  else d = m-1;
}
return d+1;
```

19 A seguinte alternativa de *buscabinaria2* funciona corretamente?

```
e = -1; d = n-1;
while (e < d) {
  m = (e + d + 1)/2;
  if (v[m] < x) e = m;
  else d = m-1;
}
return d+1;
```

20 Preencha os "??" corretamente.

```
int buscabinaria2 (int x, int n, int v[]) {
  int e = ??,  d = ??;
  while (e ?? d-1) {
    m = (e + d)/2;
    if (v[m] ?? x) e = m;
    else d = m;
  }
  return ??;
}
```

Considere a seguinte <u>função recursiva</u> que faz uma busca binária em um vetor previamente ordenado:

```
// Esta função recebe um vetor crescente v[e+1..d-1] e um
// número x tal que v[e] < x <= v[d] (portanto, e < d).
// Ela devolve um índice j em e+1..d tal que// v[j-1] < x <= v[j].
int bb (int x, int e, int d, int v[]) {
  if (e == d-1)
    return d;    // base da recursão
  else {
    int m = (e + d)/2;
    if (v[m] < x)
      return bb (x, m, d, v);
    else
      return bb (x, e, m, v);
  }
}

// A função abaixo recebe um vetor crescente v[0..n-1]
// com n >= 1 e um número x.
// Ela devolve um índice j em 0..n tal que v[j-1] < x <= v[j].
int buscabinaria3 (int x, int n, int v[]) {
  if (v[n-1] < x)
    return n;
  if (x <= v[0])
    return 0;
  return bb (x, 0, n-1, v);
}
```

21 Mostre que *buscabinaria3* pode ser simplificada:

```
int buscabinaria3 (int x, int n, int v[]) {
   return bb (x, -1, n, v);
}
```

(Basta imaginar que v[-1] vale infinito negativo e v[n] vale infinito positivo.)

22 Escreva uma função recursiva de busca binária que devolva j tal que v[j] == x ou devolva -1 se tal j não existe.

23 ***Overflow.*** Se o número de elementos do vetor (valor da variável n) estiver próximo de ***INT_MAX***, o código da busca binária pode descarrilar na linha *m = (e + d)/2;* em virtude de um overflow aritmético. Como evitar isso?

24 ***Vetor de strings.*** Suponha que v[0..n-1] é um vetor de strings em ordem lexicográfica. Escreva uma função que receba uma string x e devolva um índice j tal que a string x é igual à string v[j]. Se tal j não existe, a função deve devolver -1.

25 ***Vetor de structs.*** Suponha que cada elemento do vetor v[0..n-1] é uma *struct* com dois campos: o nome de um aluno e o número do aluno. Suponha que o vetor está em ordem crescente de números. Escreva uma função de busca binária que receba o número de um aluno e devolva o seu nome. Se o número não está no vetor, a função deve devolver a string vazia.

26 Familiarize-se com a função *bsearch* da biblioteca *stdlib*.

Esses exercícios foram retirados do endereço
http://www.ime.usp.br/~pf/algoritmos/aulas/bubi.html