Busca

Aula 4

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

1. Introdução
2. Busca sequencial
3. Busca em vetor ordenado
4. Exercícios

Introdução

- Busca é uma operação básica em Computação
- Depende da maneira como o conjunto está modelado
- Convenção: conjunto de números inteiros armazenados em um vetor
  - Busca sequencial
  - Busca binária

Introdução

- Busca é uma operação básica em Computação
- Depende da maneira como o conjunto está modelado
- Convenção: conjunto de números inteiros armazenados em um vetor
  - Busca sequencial
  - Busca binária

Introdução

- Busca é uma operação básica em Computação
- Depende da maneira como o conjunto está modelado
- Convenção: conjunto de números inteiros armazenados em um vetor
  - Busca sequencial
  - Busca binária

Introdução

- Busca é uma operação básica em Computação
- Depende da maneira como o conjunto está modelado
- Convenção: conjunto de números inteiros armazenados em um vetor
  - Busca sequencial
  - Busca binária

Introdução

- Busca é uma operação básica em Computação
- Depende da maneira como o conjunto está modelado
- Convenção: conjunto de números inteiros armazenados em um vetor
  - Busca sequencial
  - Busca binária

Busca sequencial

Problema

Dado um número inteiro $n \geqslant 0$, um vetor de números inteiros $v[0..n-1]$ e um número inteiro x, encontrar um índice k tal que $v[k] = x$

```
/* Recebe um número inteiro n >= 0, um vetor v[0..n-1] com n nú-
   meros inteiros e um número inteiro x e devolve k no intervalo
   [0, n-1] tal que v[k] == x. Se tal k não existe, devolve -1. */
int busca_sequencial(int n, int v[MAX], int x)
{
   int k;

   for (k = n - 1; k >= 0 && v[k] != x; k--)
      ;

   return k;
}
```

Busca sequencial

Problema

Dado um número inteiro $n \geqslant 0$, um vetor de números inteiros $v[0..n-1]$ e um número inteiro x, encontrar um índice k tal que $v[k] = x$

```
/* Recebe um número inteiro n >= 0, um vetor v[0..n-1] com n nú-
   meros inteiros e um número inteiro x e devolve k no intervalo
   [0, n-1] tal que v[k] == x. Se tal k não existe, devolve -1. */
int busca_sequencial(int n, int v[MAX], int x)
{
   int k;

   for (k = n - 1; k >= 0 && v[k] != x; k--)
      ;

   return k;
}
```

Busca sequencial recursiva

```
/* Recebe um número inteiro n >= 0, um vetor de números in-
   teiros v[0..n-1] e um número x e devolve k tal que 0 <= k
   < n e v[k] == x. Se tal k não existe, devolve -1. */
int busca_sequencial_R(int n, int v[MAX], int x)
{
   if (n == 0)
      return -1;
   else
      if (x == v[n - 1])
         return n - 1;
      else
         return busca_sequencial_R(n - 1, v, x);
}
```

Busca sequencial

- Tempo de execução (de pior caso) proporcional a n ou $O(n)$ ou linear no tamnaho da entrada n
- Correção semelhante à mostrada no exercício 3.1
- “Maus exemplos” no capítulo 3, páginas 12 e 13, do livro de P. Feofiloff

Busca sequencial

- Tempo de execução (de pior caso) proporcional a n ou $O(n)$ ou linear no tamnaho da entrada n
- Correção semelhante à mostrada no exercício 3.1
- “Maus exemplos” no capítulo 3, páginas 12 e 13, do livro de P. Feofiloff

Busca sequencial

- Tempo de execução (de pior caso) proporcional a n ou $O(n)$ ou linear no tamnaho da entrada n
- Correção semelhante à mostrada no exercício 3.1
- “Maus exemplos” no capítulo 3, páginas 12 e 13, do livro de P. Feofiloff

Busca sequencial

- Tempo de execução (de pior caso) proporcional a n ou $O(n)$ ou linear no tamnaho da entrada n
- Correção semelhante à mostrada no exercício 3.1
- "Maus exemplos" no capítulo 3, páginas 12 e 13, do livro de P. Feofiloff

Busca sequencial

- Tempo de execução (de pior caso) proporcional a n ou $O(n)$ ou linear no tamnaho da entrada n
- Correção semelhante à mostrada no exercício 3.1
- “Maus exemplos” no capítulo 3, páginas 12 e 13, do livro de P. Feofiloff

Busca em vetor ordenado

Definição

Um vetor de números inteiros $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$ e **decrescente** se $v[0] \geqslant v[1] \geqslant \cdots \geqslant v[n-1]$

Definição

Um vetor é **ordenado** se é crescente ou decrescente

Problema

Dado um número inteiro $n \geqslant 0$, um vetor de números inteiros ordenado $v[0..n-1]$ e um número inteiro x, encontrar um índice k tal que $v[k-1] < x \leqslant v[k]$

Busca em vetor ordenado

Definição

Um vetor de números inteiros $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$ e **decrescente** se $v[0] \geqslant v[1] \geqslant \cdots \geqslant v[n-1]$

Definição

Um vetor é **ordenado** se é crescente ou decrescente

Problema

Dado um número inteiro $n \geqslant 0$, um vetor de números inteiros ordenado $v[0..n-1]$ e um número inteiro x, encontrar um índice k tal que $v[k-1] < x \leqslant v[k]$

Busca em vetor ordenado

Definição

Um vetor de números inteiros $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$ e **decrescente** se $v[0] \geqslant v[1] \geqslant \cdots \geqslant v[n-1]$

Definição

Um vetor é **ordenado** se é crescente ou decrescente

Problema

Dado um número inteiro $n \geqslant 0$, um vetor de números inteiros ordenado $v[0..n-1]$ e um número inteiro x, encontrar um índice k tal que $v[k-1] < x \leqslant v[k]$

Busca em vetor ordenado

- $v[k-1] < x \leqslant v[k]$ vale para todo k, com $0 \leqslant k \leqslant n$
  - se $k = 0$ então a condição é $x \leqslant v[0]$
  - se $k = n$ então a condição é $v[n-1] < x$
- Supor $n \geqslant 1$

Busca em vetor ordenado

- $v[k-1] < x \leqslant v[k]$ vale para todo k, com $0 \leqslant k \leqslant n$
  - se $k = 0$ então a condição é $x \leqslant v[0]$
  - se $k = n$ então a condição é $v[n-1] < x$
- Supor $n \geqslant 1$

Busca em vetor ordenado

- $v[k-1] < x \leqslant v[k]$ vale para todo k, com $0 \leqslant k \leqslant n$
  - se $k = 0$ então a condição é $x \leqslant v[0]$
  - se $k = n$ então a condição é $v[n-1] < x$
- Supor $n \geqslant 1$

Busca em vetor ordenado

- $v[k-1] < x \leqslant v[k]$ vale para todo k, com $0 \leqslant k \leqslant n$
  - se $k = 0$ então a condição é $x \leqslant v[0]$
  - se $k = n$ então a condição é $v[n-1] < x$
- Supor $n \geqslant 1$

Busca em vetor ordenado

```
/* Recebe um número inteiro n > 0, um vetor de números in-
   teiros crescente v[0..n-1] e um número inteiro x e devol-
   ve um índice k em [0, n] tal que v[k-1] < x <= v[k] */
int busca_ordenada(int n, int v[MAX], int x)
{
   int k;

   for (k = 0; k < n && v[k] < x; k++)
      ;

   return k;
}
```

Tempo de execução $O(n)$

Busca em vetor ordenado

```
/* Recebe um número inteiro n > 0, um vetor de números in-
   teiros crescente v[0..n-1] e um número inteiro x e devol-
   ve um índice k em [0, n] tal que v[k-1] < x <= v[k] */
int busca_ordenada(int n, int v[MAX], int x)
{
   int k;

   for (k = 0; k < n && v[k] < x; k++)
      ;

   return k;
}
```

Tempo de execução $O(n)$

Busca em vetor ordenado

- **Busca binária**: processo automático que usamos para busca uma palavra em um dicionário

```
/* Recebe um número inteiro n > 0, um vetor de números in-
   teiros crescente v[0..n-1] e um número inteiro x e devol-
   ve um índice k em [0, n] tal que v[k-1] < x <= v[k] */
int busca_binaria(int n, int v[MAX], int x)
{
   int esq, dir, meio;

   esq = -1;
   dir = n;
   while (esq < dir - 1) {
      meio = (esq + dir) / 2;
      if (v[meio] < x)
         esq = meio;
      else
         dir = meio;
   }
   return dir;
}
```

Busca em vetor ordenado

- Busca binária: processo automático que usamos para busca uma palavra em um dicionário

```
/* Recebe um número inteiro n > 0, um vetor de números in-
   teiros crescente v[0..n-1] e um número inteiro x e devol-
   ve um índice k em [0, n] tal que v[k-1] < x <= v[k] */
int busca_binaria(int n, int v[MAX], int x)
{
   int esq, dir, meio;

   esq = -1;
   dir = n;
   while (esq < dir - 1) {
      meio = (esq + dir) / 2;
      if (v[meio] < x)
         esq = meio;
      else
         dir = meio;
   }
   return dir;
}
```

Busca em vetor ordenado

- Busca binária: processo automático que usamos para busca uma palavra em um dicionário

```
/* Recebe um número inteiro n > 0, um vetor de números in-
   teiros crescente v[0..n-1] e um número inteiro x e devol-
   ve um índice k em [0, n] tal que v[k-1] < x <= v[k] */
int busca_binaria(int n, int v[MAX], int x)
{
   int esq, dir, meio;

   esq = -1;
   dir = n;
   while (esq < dir - 1) {
      meio = (esq + dir) / 2;
      if (v[meio] < x)
         esq = meio;
      else
         dir = meio;
   }
   return dir;
}
```

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*
- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`.

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor *v* é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é *n*. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Invariante (para provar correção):
  no início de cada repetição `while`*, imediatamente antes da comparação de* `esq` *com* `dir - 1`*, vale a relação* `v[esq] < x <= v[dir]`*.*

- Tempo de execução: em cada iteração, o tamanho do vetor v é dado por `dir - esq - 1`. No início da primeira iteração, o tamanho do vetor é n. No início da segunda iteração, o tamanho do vetor é aproximadamente $n/2$. No início da terceira, aproximadamente $n/4$. No início da $(k+1)$-ésima, aproximadamente $n/2^k$. Quando $k > \log_2 n$, temos $n/2^k < 1$ e a execução da função termina. Assim, o número de iterações é aproximadamente $\log_2 n$. O consumo de tempo da função é proporcional ao número de iterações e portanto proporcional a $\log_2 n$.

Busca em vetor ordenado

- Busca binária recursiva: procura o elemento *x* no vetor crescente **`v[esq..dir]`**

```
/* Recebe dois números inteiros esq e dir, um vetor de números
   inteiros crescente v[esq..dir] e um número inteiro x tais
   que v[esq] < x <= v[dir] e devolve um índice k em
   [esq+1, dir] tal que v[k-1] < x <= v[k] */
int busca_binaria_R(int esq, int dir, int v[MAX], int x)
{
   int meio;

   if (esq == dir - 1)
      return dir;
   else {
      meio = (esq + dir) / 2;
      if (v[meio] < x)
         return busca_binaria_R(meio, dir, v, x);
      else
         return busca_binaria_R(esq, meio, v, x);
   }
}
```

Busca em vetor ordenado

- Busca binária recursiva: procura o elemento *x* no vetor crescente `v[esq..dir]`

```
/* Recebe dois números inteiros esq e dir, um vetor de números
   inteiros crescente v[esq..dir] e um número inteiro x tais
   que v[esq] < x <= v[dir] e devolve um índice k em
   [esq+1, dir] tal que v[k-1] < x <= v[k] */
int busca_binaria_R(int esq, int dir, int v[MAX], int x)
{
   int meio;

   if (esq == dir - 1)
      return dir;
   else {
      meio = (esq + dir) / 2;
      if (v[meio] < x)
         return busca_binaria_R(meio, dir, v, x);
      else
         return busca_binaria_R(esq, meio, v, x);
   }
}
```

Busca em vetor ordenado

- É fácil mostrar que a versão recusiva da busca binária está correta (por indução)
- Tempo de execução: quando a função `busca_binaria_R` é chamada com argumentos $(-1, n, v, x)$, ela chama a si mesma cerca de $\lfloor \log_2 n \rfloor$ vezes. Este número de chamadas é a profundidade da recursão e determina o tempo de execução da função

Busca em vetor ordenado

- É fácil mostrar que a versão recusiva da busca binária está correta (por indução)
- Tempo de execução: quando a função `busca_binaria_R` é chamada com argumentos $(-1, n, v, x)$, ela chama a si mesma cerca de $\lfloor \log_2 n \rfloor$ vezes. Este número de chamadas é a profundidade da recursão e determina o tempo de execução da função

- É fácil mostrar que a versão recusiva da busca binária está correta (por indução)
- Tempo de execução: quando a função `busca_binaria_R` é chamada com argumentos $(-1, n, v, x)$, ela chama a si mesma cerca de $\lfloor \log_2 n \rfloor$ vezes. Este número de chamadas é a profundidade da recursão e determina o tempo de execução da função

Exercícios

1. Tome uma decisão de projeto diferente daquela da seção 2: se x não estiver em $v[0..n-1]$, a função deve devolver n. Escreva a versão correspondente da função `busca`. Para evitar o grande número de comparações de k com n, coloque uma "sentinela" em $v[n]$.

Exercícios

```
/* Recebe um número inteiro n >= 0, um vetor de números intei-
   ros v[0..n-1] e um número inteiro x e devolve k no intervalo
   [0, n-1] tal que v[k] == x. Se tal k não existe, devolve n */
int busca_sequencial_sentinela(int n, int v[MAX+1], int x)
{
   int k;

   v[n] = x;
   for (k = 0; v[k] != x; k++)
      ;

   return k;
}
```

Exercícios

2. Considere o problema de determinar o valor de um elemento máximo de um vetor $v[0..n-1]$ e a função a seguir.

```
int maximo(int n, int v[MAX])
{
   int i, x;

   x = v[0];
   for (i = 1; i < n; i++)
      if (x < v[i])
         x = v[i];

   return x;
}
```

   (a) A função `maximo` acima resolve o problema?
   (b) Faz sentido trocar `x = v[0]` por `x = 0`?
   (c) Faz sentido trocar `x = v[0]` por `x = INT_MIN`?
   (d) Faz sentido trocar `x < v[i]` por `x <= v[i]`?

3. O autor da função abaixo afirma que ela decide se x está no vetor $v[0..n-1]$. Critique seu código.

```
int buscaR2(int n, int v[MAX], int x)
{
   if (v[n-1] == x)
      return 1;
   else
      return buscaR2(n-1, v, x);
}
```

Exercícios

4. A operação de remoção consiste de retirar do vetor $v[0..n-1]$ o elemento que tem índice k e fazer com que o vetor resultante tenha índices $0, 1, \ldots, n-2$. Essa operação só faz sentido se $0 \leqslant k < n$.
   (a) Escreva uma função não-recursiva com a seguinte interface:
   ```
   int remove(int n, int v[MAX], int k)
   ```
   que remove o elemento de índice k do vetor $v[0..n-1]$ e devolve o novo valor de n, supondo que $0 \leqslant k < n$.
   (b) Escreva uma função recursiva para a remoção com a seguinte interface:
   ```
   int remove_R(int n, int v[MAX], int k)
   ```

Exercícios

5. A operação de inserção consiste em introduzir um novo elemento y entre a posição de índice $k-1$ e a posição de índice k no vetor $v[0..n-1]$, com $0 \leqslant k \leqslant n$.
   (a) Escreva uma função não-recursiva com a seguinte interface:

   ```
   int insere(int n, int v[MAX], int k, int y)
   ```

   que insere o elemento y entre as posições $k-1$ e k do vetor $v[0..n-1]$ e devolve o novo valor de n, supondo que $0 \leqslant k \leqslant n$.
   (b) Escreva uma função recursiva para a inserção com a seguinte interface:

   ```
   int insere_R(int n, int v[MAX], int k, int x)
   ```

Exercícios

6. Na busca binária, suponha que $v[i] = i$ para todo i.
   (a) Execute a função `busca_binaria` com $n = 9$ e $x = 3$;
   (b) Execute a função `busca_binaria` com $n = 14$ e $x = 7$;
   (c) Execute a função `busca_binaria` com $n = 15$ e $x = 7$.
7. Execute a função `busca_binaria` 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?
8. Confira a validade da seguinte afirmação: quando $n + 1$ é uma potência de 2, o valor da expressão `(esq + dir)` é divisível por 2 em todas as iterações da função `busca_binaria`, quaisquer que sejam v e x.

Exercícios

9. Responda as seguintes perguntas sobre a função `busca_binaria`.
   (a) Que acontece se a sentença `while (esq < dir - 1)` for substituída pela sentença `while (esq < dir)`?
   (b) Que acontece se a sentença `if (v[meio] < x)` for substituída pela sentença `if (v[meio] <= x)`?
   (c) Que acontece se `esq = meio` for substituído por `esq = meio + 1` ou então por `esq = meio - 1`?
   (d) Que acontece se `dir = meio` for substituído por `dir = meio + 1` ou então por `dir = meio - 1`?
10. Se t segundos são necessários para fazer uma busca binária em um vetor com n elementos, quantos segundos serão necessários para fazer uma busca em n^2 elementos?

Exercícios

11. Escreva uma versão da busca binária para resolver o seguinte problema: dado um inteiro x e um vetor decrescente $v[0..n-1]$, encontrar k tal que $v[k-1] > x \geqslant v[k]$.
12. Suponha que cada elemento do vetor $v[0..n-1]$ é uma cadeia de caracteres (ou seja, temos uma matriz de caracteres). Suponha também que o vetor está em ordem lexicográfica. Escreva uma função eficiente, baseada na busca binária, que receba uma cadeia de caracteres x e devolva um índice k tal que x é igual a $v[k]$. Se tal k não existe, a função deve devolver -1.

Exercícios

13. Suponha que cada elemento do vetor $v[0..n-1]$ é um registro com dois campos: o nome do(a) estudante e o número do(a) estudante. Suponha que o vetor está em ordem crescente de números. Escreva uma função de busca binária que receba o número de um(a) estudante e devolva seu nome. Se o número não estiver no vetor, a função deve devolver a cadeia de caracteres vazia.
14. Escreva uma função que receba um vetor crescente $v[0..n-1]$ de números inteiros e devolva um índice i entre 0 e $n-1$ tal que $v[i] = i$. Se tal i não existe, a função deve devolver -1. A sua função não deve fazer mais que $\lfloor \log_2 n \rfloor$ comparações envolvendo os elementos de v.