Busca sequencial e binária

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Resumo da aula anterior

- Corretude de algoritmos iterativos – **invariante de laço**:
  - Declaração do invariante (proposição).
  - Demonstração: inicialização, manutenção e término.
- Técnica de projeto de algoritmo: **incremental**.
  - Constrói a solução a medida em que processa a entrada elemento a elemento (viés dinâmico).
  - Demonstração de correção por **invariante de laço**.
  - Tempo de execução depende do tempo gasto em cada iteração.
  - Exemplos: MAXIMO, INSERTION-SORT, entre outros.
- Análise de complexidade:
  - Função especificando a ordem de crescimento em relação ao tamanho da entrada do algoritmo.
  - Contagem do número de execuções de cada linha.
  - Melhor caso, pior caso e caso médio.

Características da entrada

Em alguns algoritmos, o tempo de execução pode depender também dos dados de entrada (como no caso do INSERTION-SORT para uma entrada de tamanho n).

- **Melhor caso:** menor tempo de execução sobre todas as entradas de tamanho n.
- **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho n.
- **Caso médio:** média dos tempos de execução de todas as entradas de tamanho n.

Solução do exercício da aula anterior

SELECTION-SORT(A)

```
for i ← 1 to A.length − 1 do
    min ← i
    for j ← i + 1 to A.length do
        if A[j] < A[min] then
            min ← j
    SWAP(A[i], A[min]);
```

Solução do exercício da aula anterior

SELECTION-SORT(A)

for $i \leftarrow 1$ **to** $A.length - 1$ **do**
    $min \leftarrow i$
    for $j \leftarrow i + 1$ **to** $A.length$ **do**
        if $A[j] < A[min]$ **then**
            $min \leftarrow j$
    SWAP($A[i]$, $A[min]$);

Correção do Selection sort

- Invariante do laço interno: `A[min]` contém o menor elemento de `A[i..j-1]`. (Exercício.)
- Invariante do laço externo: O subvetor `A[1..i-1]` consiste dos `i-1` menores elementos do vetor `A[1..n]` e, este subvetor está ordenado.

Correção do Selection-Sort

Invariante de laço (externo)

O subvetor `A[1..i-1]` consiste dos `i-1` menores elementos do vetor `A[1..n]` em ordem.

Demonstração (A.length = n)

- **Inicialização:** Quando $i = 1$, o subvetor A está vazio e, portanto, o invariante é trivialmente verdadeiro.
- **Manutenção:** Considere uma iteração i qualquer e assuma que o invariante é verdadeiro no início desta iteração. Executando as linhas 2 a 5 do corpo do laço, `min` conterá a posição do menor valor do vetor `A[i..n]`. A linha 6 faz a troca entre os elementos na posição `i` e `min`. Assim, se `A[1..i-1]` continha os menores valores do vetor `A[1..n]` em ordem e agora A[i] contém o menor valor de `A[i..n]`, então é possível afirmar que `A[1..i]` contém os menores valores do vetor `A[1..n]` em ordem (e o invariante é verdadeiro para a próxima iteração).
- **Término:** Quando `i = n`, o subvetor `A[1..n-1]` consiste dos `n-1` menores elementos do vetor `A[1..n]` em ordem. Como `A[n]` é o maior dos elementos e está na posição correta, então o vetor inteiro está ordenado.

Solução do exercício da aula anterior

Análise de complexidade (no pior caso)[1]

selection-sort(A)	custo	# de execuções
`1 for i = 1 to A.length - 1`	c_1	n
`2   min = i`	c_2	$n-1$
`3   for j = i + 1 to A.length`	c_3	$\sum_{j=2}^{n} j$
`4     if A[j] < A[min]`	c_4	$\sum_{j=2}^{n}(j-1)$
`5       min = j`	c_5	$\sum_{j=2}^{n}(j-1)$
`6   Swap (A[i], A[min])`	c_6	$n-1$

Custo total

Multiplicando pelas constantes e desenvolvendo temos:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_3 \sum_{j=2}^{n} j + c_4 \sum_{j=2}^{n}(j-1) + \\
&\quad c_5 \sum_{j=2}^{n}(j-1) + c_6(n-1) \\
&= \left(\frac{c_3}{2} - \frac{c_4}{2} - \frac{c_5}{2}\right)n^2 + \left(c_1 + c_2 + \frac{c_3}{2} - \frac{c_4}{2} - \frac{c_5}{2} + c_6\right)n - \\
&\quad (c_2 + c_3 + c_6).
\end{aligned}
$$

Função quadrática.

[1] No melhor caso a linha 5 não executa nenhuma vez.

Objetivos desta aula

- Reforçar correção de algoritmos.
- Analisar a complexidade do problema de busca sequencial.
- Apresentar a ideia de busca binária.
- Análise do algoritmo de busca binária.

Busca sequencial

Entrada

Uma sequência de números $A = \langle a_1, a_2, \ldots, a_n \rangle$ e um valor v.

Saída

Um índice i tal que $v = A[i]$ ou o valor especial *NIL* se o valor v não aparece em A.

Um algoritmo

Sequential-Search(A, v)

```
for i ← 1 to A.length do
    if A[i] == v then
        return i
return NIL
```

Busca sequencial

Invariante de laço

O subvetor $A[1 \dots i-1]$ consiste de elementos diferentes de v.

- **Inicialização:** inicialmente o subvetor está vazio ($A[1 \dots 0] = \emptyset$), portanto, o invariante é trivialmente válido.
- **Manutenção:** Para uma iteração i qualquer, $A[1 \dots i-1]$ não contém v. Ao comparar $A[i]$ com v existem duas possibilidades:
  - Se $A[i] == v$: devolvemos o valor de i (resultado correto).
  - Se $A[i] \neq v$: sabemos que $A[1 \dots i-1]$ não contém v e que $A[i] \neq v$, portanto, o invariante é preservado para a próxima iteração.
- **Término:** o laço termina quando $i = A.length + 1$, ou seja, $A[1 \dots A.length{+}1{-}1] = A[1 \dots A.length]$ não contém v e consequentemente o algoritmo devolve NIL.

Análise de complexidade do SEQUENTIAL-SEARCH

Seja $n = A.length$:

- **Melhor caso:** v está na primeira posição do vetor ($A[1]$) e o algoritmo termina em seguida (tempo constante).
- **Pior caso:** v não está no vetor e o algoritmo efetua n comparações (tempo linear).
- **Caso médio:** como calcular?
  - Supor uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n e obter o custo médio com base nesta distribuição.
  - Problema: nem sempre é sabemos qual é esta distribuição.

Caso médio do Sequential-Search

Seja $n = A.length$:

- Assumiremos que a probabilidade do elemento v estar em cada posição é $1/n$.
- Se v está na posição 1, um elemento precisa ser verificado. Se v está na posição 2, dois elementos precisam ser verificados. Se o v está na i-ésima posição, i elementos precisam ser verificados.
- Temos que fazer a média ponderada da quantidade de verificações:

$$\frac{1}{n}1 + \frac{1}{n}2 + \frac{1}{n}3 + \cdots + \frac{1}{n}n = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{n(n+1)}{2} = \frac{(n+1)}{2}.$$

- Caso médio é linear.

Pergunta

É possível resolver este problema de maneira mais eficiente?

Pergunta

É possível resolver este problema de maneira mais eficiente?
Resposta: Sim, se o vetor estiver ordenado. Não, caso contrário.

Exemplo

Considere a sequência de números:
$A = \{1, 3, 4, 7, 8, 11, 13, 15\}$.
Como descobrir se um dado número v está em A sem ter que fazer n comparações?

Um algoritmo iterativo de Busca Binária

```
BINARY-SEARCH(A, key)
left ← 1
right ← A.length
while right ≥ left do
    mid ← (left + right)/2
    if key == A[mid] then
        return mid
    else if key < A[mid] then
        right ← mid − 1
    else
        left ← mid + 1
return NIL
```

Correção do algoritmo $\textsc{Binary-Search}(A, key)$

Invariante de laço

Se key está em $A[1 \dots n]$, então key está em $A[left \dots right]$.

Demonstração

- **Inicialização:** inicialmente considera-se o vetor inteiro, portanto, o invariante é trivialmente válido.
- **Manutenção:** Supondo que o invariante é válido em uma iteração qualquer, ou seja, se key está em $A[1 \dots n]$, então key está em $A[left \dots right]$; ao executar o corpo do laço existem três possibilidades:

Demonstração (continuação)

- ...três possibilidades:
  - Se *key* $== A[mid]$: devolvemos a posição do elemento *key* (resultado correto).
  - Se $key < A[mid]$: como o vetor está ordenado, então sabemos que *key* é menor que todos os elementos de $A[mid \dots right]$, portanto, podemos descartar esta parte do vetor e, se *key* está em $A[1 \dots n]$, então *key* está em $A[left \dots mid-1]$;
  - Se $key \geq A[mid]$: como o vetor está ordenado, então sabemos que *key* é maior que todos os elemento de $A[left \dots mid]$, portanto, podemos descartar esta parte do vetor e, se *key* está em $A[1 \dots n]$, então *key* está em $A[mid+1 \dots right]$;
- **Término:** o laço termina quando não encontrou o elemento, ou seja, $right < left$ (vetor vazio). Neste caso o algoritmo devolve corretamente o valor *NIL*.

Complexidade do algoritmo BINARY-SEARCH(A, key)

Seja $n = A.length$:

- **Melhor caso:** *key* está no meio do vetor ($A[mid]$) e o algoritmo termina em seguida (tempo constante).
- **Pior caso:** *key* não está no vetor e o algoritmo efetua $log(n)$ comparações.
- **Caso médio:** como calcular?
  - Supor uma distribuição de probabilidades uniforme da posição do elemento que queremos encontrar no vetor de tamanho n.
  - Para o caso de 1 comparação, a posição do elemento deve estar no meio, portanto a probabilidade é de $1/n$ para este caso.
  - Para o caso de 2 comparações (comparação com o elemento do meio mais uma comparação em cada um dos subvetores), a probabilidade é $2/n$.
  - Para o caso de i comparações, a probabilidade é de $2^{i-1}/n$.
  - Portanto, o caso médio será:

$$\sum_{i=1}^{\lg n} \frac{i2^{i-1}}{n} = \frac{1}{n} \sum_{i=1}^{\lg n} i2^{i-1} \approx \lg n$$

Exercícios

1. Leia o Apêndice A do Cormen (somatórios, fórmulas, propriedades, aproximações).
2. Faça um algoritmo recursivo para o problema de busca binária.