Projeto e Análise de Algoritmos

Prova de corretude

Indução

Invariantes de laço

Análise de algoritmos

- Eficiência do algoritmo
  - Análise da ordem de crescimento da função de complexidade: tempo ou espaço
  - Notação assintótica
- Corretude do algoritmo
  - Analisar se o algoritmo um resultado correto

Corretude de algoritmos

- Demonstração de incorretude
  - Produzir uma instância em que o algoritmo produz um resultado incorreto
    - Contra-exemplo
  - Propriedades de bons contra-exemplos
    - Verificabilidade
      - É necessário calcular a resposta produzida pelo algoritmo para a instância e mostrar uma resposta melhor que o algoritmo não foi capaz de produzir
    - Simplicidade
      - Mostra de forma simples e clara que um algoritmo é falho.

Corretude de algoritmos

- Demonstração de corretude
  - Invariante de laço
    - Propriedade que relaciona as variáveis do algoritmo a cada execução de um laço
    - Utilizado para provar corretude de algoritmos iterativos
    - Dificuldade: enunciar o invariante apropriado
  - Indução
    - Método matemático que pode ser usado para demonstrar corretude de algoritmos iterativos ou recursivos
    - Dificuldade: erros sutis e provas convincentes

Indução Matemática

- Prova de uma dada proposição
- Uma proposição para cada n: para cada valor de $n \geq 0$, seja $S(n)$ uma proposição, que pode ser verdadeira para alguns valores de n e falso para outros.
- Objetivo: Provar que para $\forall n \geq 0, S(n)$ é verdadeira.
- Prova por indução
  - Hipótese de indução: Definir a proposição, para cada $n \geq 0, S(n)$....
  - Caso base: Provar que $S(0)$ é verdadeiro.
  - Passo indutivo: para $n \geq 1$, provar $S(n-1) \Rightarrow S(n)$, assumindo que $S(n-1)$ é verdadeiro e portanto $S(n)$ também deve ser.
  - Conclusão: Concluir que $\forall n \geq 0, S(n)$ é verdadeira.

Indução

- Exemplo

$$S(n) = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

É valido para todos os números naturais n.

- Caso base: verdadeiro para $n = 1$, pois $1 = \frac{1(1+1)}{2}$.
- Provar que $S(n)$ é verdadeiro para $n = k$.
- Por hipótese de indução, a equação $S(n)$ vale para $n = k - 1$, ou seja
  - $S(k-1) = 1 + 2 + \cdots + (k-1) = \frac{(k-1)\left((k-1)+1\right)}{2}$

- Passo de indução:

$$\begin{aligned} S(k) = 1 + 2 + \cdots + (k-1) + k &= \frac{(k-1)\left((k-1)+1\right)}{2} + k \\ &= \frac{k(k-1) + 2k}{2} \\ &= \frac{k((k-1)+2)}{2} \\ &= \frac{k(k+1)}{2} \end{aligned}$$

Logo, assumindo que $S(k-1)$é verdadeiro, podemos concluir que $S(k)$ é verdadeiro. Portanto, podemos concluir que $S(n)$ é válido para todos os números naturais n.

Invariantes de laço

- Usados para nos ajudar a entender por que um algoritmo é correto.
- Mostrar três situações:
  - **Inicialização**: verdadeiro antes da primeira iteração do laço.
  - **Manutenção**: verdadeiro antes da iteração do laço e permanece verdade antes da próxima iteração.
  - **Terminação:** quando o laço termina, a invariante dá uma propriedade útil que nos ajuda a mostrar que o algoritmo é correto.

Invariante de laço

- Se o invariante de laço for verdadeiro nas 2 primeiras situações, o invariante de laço é verdadeiro antes de cada iteração
- Similaridade à indução matemática
  - Inicialização: caso base
  - Manutenção: passo indutivo
  - Terminação: paramos quando o laço termina, diferentemente da indução.

```
int calc(int a){
    int x=1;
    int i=1;
    while(i<=a){
        x = x*i;
        i = i+1;
    }
    return x;
}
```

- O que este algoritmo calcula?

```
int calc(int a){
    int x=1;
    int i=1;
    while(i<=a){
        x = x*i;
        i = i+1;
    }
    return x;
}
```

- O que este algoritmo calcula?

- Qual invariante de laço este algoritmo mantém?

- Invariante de laço

“Antes da execução do laço, $x = (i - 1)!$”

- Demostre a corretude do algoritmo anterior por este invariante de laço

– Inicialização
  - Antes da execução da primeira iteração, $i = 1$ e $x = 1$. Como $x = (i - 1)! = 1$, então o invariante é satisfeito.

– Manutenção:

- A cada iteração do laço, são executados as operações $x' = x * i$ e $i' = i + 1$, sendo x' e i' as variáveis com seus valores atualizados pelas operações.
- Antes da execução do laço para uma dada iteração i, pelo invariante de laço temos $x = (i-1)!$, portanto a primeira atribuição realiza a operação $x' = x * i = (i-1)! * i = i!$.
- Na próxima linha, a variável i é incrementada, $i' = i + 1$, fazendo com que o invariante seja mantido para a próxima iteração, pois $i = i' - 1$, logo $x' = i! = (i'-1)!$.

- Terminação
  - O laço termina quando $i = a + 1$. Pelo invariante de laço, $x = (i - 1)!$, mas como $i = a + 1$, então, $x = (a + 1 - 1)!$. Logo, a função retornará o valor $x = a!$.

Insertion sort

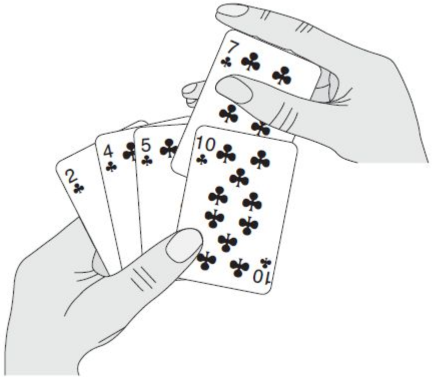
- Problema de ordenação
  - Entrada
    - Sequência de n números $\langle a_1, a_2, \dots, a_n \rangle$.
  - Saída
    - Permutação $\langle a'_1, a'_2, \dots, a'_n \rangle$ da sequência da entrada tal que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

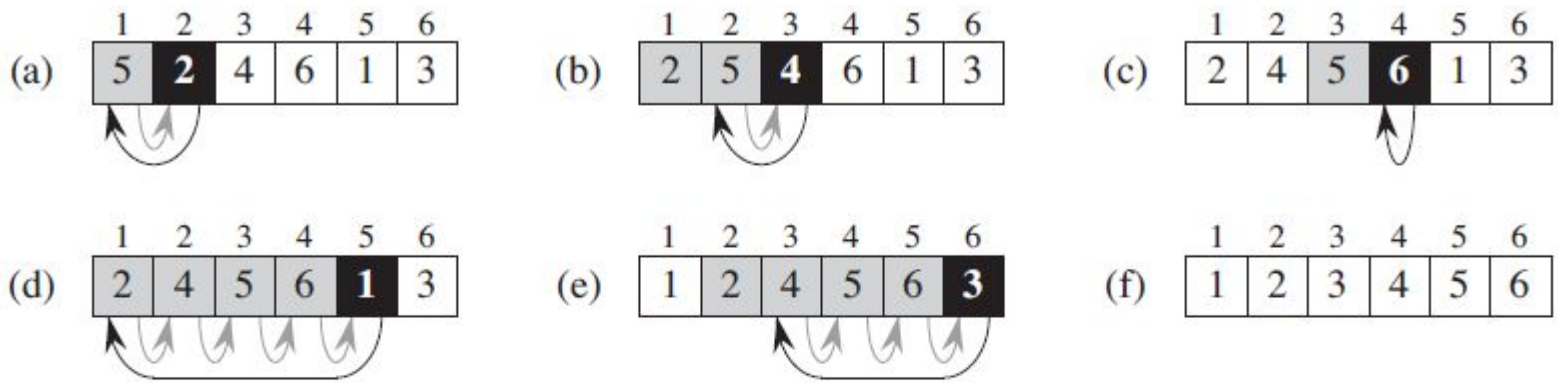
Insertion Sort

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- Entender o funcionamento do algoritmo é importante para determinar o invariante de laço
  - Qual propriedade deve ser mantida antes de cada iteração e após a última iteração?

Insertion sort





- Índice j indica a “carta corrente” a ser inserida entre as cartas ordenadas
- No início de cada iteração do laço for, o subarranjo $A[1 \ldots j-1]$ representa a mão ordenada
- O subarranjo $A[j+1 \ldots n]$ corresponde a pilha de carta ainda na mesa
- No início de cada iteração do for, o subarranjo $A[1 \ldots j-1]$ consiste dos elementos orinalmente em $A[1 \ldots j-1]$, mas já de forma ordenada.

Insertion sort

Invariante de laço

- Inicialização
  - Para $j = 2$: $A[1 \ldots j-1]$ consiste apenas de $A[1]$, que é o elemento original em $A[1]$. O subarranjo está ordenado, portanto a propriedade é valida antes da primeira iteração

Insertion sort

Invariante de laço

- Inicialização
  - Para $j = 2$: $A[1 \dots j-1]$ consiste apenas de $A[1]$, que é o elemento original em $A[1]$. O subarranjo está ordenado, portanto a propriedade é valida antes da primeira iteração
- Manutenção
  - O corpo do for move para direita os elementos $A[j-1]$, $A[j-2]$, e assim por diante, enquanto esses elementos forem maior que $A[j]$, inserindo ao final, o elemento $A[j]$ na posição apropriada. Portanto, $A[1 \dots j]$ consiste dos elementos originalmente em $A[1 \dots j]$, mas ordenados. O incremento de j para a próxima iteração preserva o invariante de laço.

Insertion sort

Invariante de laço

- Inicialização
  - Para $j = 2$: $A[1 \dots j-1]$ consiste apenas de $A[1]$, que é o elemento original em $A[1]$. O subarranjo está ordenado, portanto a propriedade é valida antes da primeira iteração
- Manutenção
  - O corpo do for move para direita os elementos $A[j-1]$, $A[j-2]$, e assim por diante, enquanto esses elementos forem maior que $A[j]$, inserindo ao final, o elemento $A[j]$ na posição apropriada. Portanto, $A[1 \dots j]$ consiste dos elementos originalmente em $A[1 \dots j]$, mas ordenados. O incremento de j para a próxima iteração preserva o invariante de laço.
- Terminação
  - O laço for termina quando $j > A.length = n$, logo $j = n + 1$ neste instante, pois o for incrementa j em 1 a cada iteração. Substituindo $n + 1$ no invariante de laço, temos que o subarranjo $A[1 \dots n]$ consiste de elementos originalmente em $A[1 \dots n]$, mas já ordenados. Como $A[1 \dots n]$ é o arranjo inteiro, concluímos que o arranjo inteiro está ordenado.

Insertion sort

- Corretude por indução
  - Caso base: $n = 1$, um vetor de um elemento já está ordenado.
  - Hipótese: podemos assumir que os $n - 1$ primeiros elementos do vetor A já estão ordenados após a iteração $j = n - 1$.
  - Passo de indução: inserir um novo elemento x de forma ordenada com os $n - 1$ primeiros elementos do vetor A. Isso é realizado através do deslocamento de todos elementos maiores que x uma posição a direita, criando espaço para inserir x na posição ordenada, ou seja, x é inserido na posição a esquerda do último elemento deslocado, ou mantido em sua posição inicial ($j = n$), caso x seja o maior elemento de A.

Exemplo

mdc(44, 26) = ?

Primeira versão

Iteração			
1	44	26	18
2	26	18	8
3	18	8	10
4	8	10	2
5	**2**	0	

- Se $x > y$ e ambos positivos
- Invariante de laço
  - $\mathrm{mdc}(x, y) = \mathrm{mdc}(x - y, y)$
- Passos principais
  - Tornar x e y menores, mantendo o mesmo mdc.
  - $\mathrm{mdc}(x, y) = \mathrm{mdc}(x - y, y)$
    - Ex: $\mathrm{mdc}(52{,}10) = \mathrm{mdc}(42{,}10) = 2$
    - Qualquer valor que divide x e y divide $x - y$ também
    - Logo, ao fazermos $x' = x - y$, fazemos progresso e mantemos o invariante de laço
    - $\mathrm{mdc}(1000000000{,}1)$ vai levar muito tempo
    - Seja $n = \log a + \log b$, o número de bits pra representar $\langle a, b \rangle$ esse algoritmo consumiria tempo em $\Theta(a) = \Theta(2^n)$.

Algoritmo de Euclides

- Método para calcular o Máximo Divisor Comum (MDC ou GCD – Greatest Common Divisor) entre dois números naturais
  - Desenvolvido por Euclides, na Grécia Antiga

```
int mdc(a,b){
    x = a;
    y = b;
    while (y ≠ 0){
        r = x mod y;
        x = y;
        y = r;
    }
    return x;
}
```

Exemplo

mdc(26, 44) = ?

Iteração	x	y	r
1	26	44	26
2	44	26	18
3	26	18	8
4	18	8	2
5	8	2	0
6	**2**	0	

Corretude do algoritmo de Euclides

- Invariante de laço
  - Invariante de laço incomum
  - Variáveis x e y cujos valores mudam a cada iteração do laço sob o invariante que o MDC entre eles, $\mathrm{mdc}(x, y)$, não se altera, permanecendo igual a $\mathrm{mdc}(a, b)$
  - Algoritmo funciona como uma recorrência

- Invariante de laço
  - Definir x como sendo a e y como sendo b
  - Medida de progresso
    - Tornar x ou y menores
  - Finalização
    - O algoritmo termina quando x ou y são suficientemente pequenos para que se possa calcular o mdc facilmente. Pelo invariante de laço, isso será a resposta desejada.

- Se $x > y$ e ambos positivos
- Passos principais
  - Subtrair um múltiplo de y
    - O menor múltiplo de y que não torna x negativo
    - $x' = x - \left\lfloor \frac{x}{y} \right\rfloor y = x \bmod y$
  - Ex: $52 \bmod 10 = 2$
- Invariante de laço
  - Inicialização: Antes da primeira iteração, $x = a$ e $y = b$, portanto $\mathrm{mdc}(x, y) = \mathrm{mdc}(a, b)$
    - Casos particulares
      - Fazer $x = a$ e $y = b$ não estabelece o invariante de laço, que diz que $x >= y$, se $a < b$. Porém, o algoritmo faz a troca automática de a por b na primeira iteração, se $a < b$.
      - Se a e b forem negativos, a primeira iteração torna y positivo, e a iteração seguinte tornará tanto x como y positivos

- Manutenção:
  - Mostrar que o passo $x' = x \bmod y$ mantém o invariante de laço, pois $\mathrm{mdc}(x, y) = \mathrm{mdc}(x \bmod y, y)$
  - Ex: $\mathrm{mdc}(52,10) = \mathrm{mdc}(2,10) = 2$
  - Progresso: O passo $x' = x \bmod y$ só faz progresso (torna x menor) se $x \geq y$
    - Solução: trocar x por y (novo passo principal)
    - $x' = y$ e $\mathrm{y}' = x \bmod y$
    - Mantém o invariante de laço, pois $\mathrm{mdc}(x, y) = \mathrm{mdc}(y, x \; mod \; y \;)$ e também o novo invariante $0 \leq y \leq x$
    - Ex: $\mathrm{mdc}(52,10) = \mathrm{mdc}(10,2) = 2$
    - Como $\mathrm{y}' = x \bmod y \in [0 \ldots y - 1]$, portanto reduzimos o y (progresso)

- Terminação
  - O algoritmo vai acabar parando da seguinte maneira
    - $\mathrm{y}' = x \bmod y \in [0 \ldots y-1]$ garante que y se torna estritamente menor a cada passo e não fica negativo. Logo, vai acabar se tornando 0 em algum momento.

- Solução recursiva

```
mdc(a, b)
if b == 0
  return a
else return mdc(b, a mod b)
```

Exercícios

1) Considere o código abaixo que recebe dois números naturais como parâmetros.

a) O que o algoritmo calcula?

b) Qual invariante de laço este algoritmo mantém?

c) Mostre, por invariante de laço, que este algoritmo é correto.

```
F(a, n){
        x=1;
        for(i=0; i<n; i++)
                x=a*x;
        return x;
}
```

Exercícios

2) Considere o algoritmo de ordenação Selection sort.

a) Qual invariante de laço este algoritmo mantém para o laço externo?

b) Mostre, por invariante de laço, que este algoritmo é correto.

```
Selecao(A){
    for i=1 to n-1 do
        min = i
        for j=i+1 to n do
            if(A[j] < A[min])
                min = j
        troca(A[min], A[i]);
}
```

Exercícios

3) Considere o código abaixo que recebe dois números naturais como parâmetros.

a) O que o algoritmo calcula?

b) Qual invariante de laço este algoritmo mantém?

c) Mostre, por invariante de laço, que este algoritmo é correto.

```
A(a,b){
    x=0;
    while(b > 0){
        if(b % 2 == 1) x = x + a;
        a= 2 * a;
        b = ⌊b / 2⌋;
    }
    return x;
}
```

Referências

- CLRS, Introduction to Algorithms, 3rd ed.
  - 2.1
- Manber, Introduction to Algorithms – A creative approach, 1st ed.
  - 2.1, 2.2, 2.12
- Skiena, The Algorithm Design Manual, 2nd ed.
  - 1.1
- Edmonds, How to think about algorithms, 1st ed.
  - Cap 1, cap 6.