Algoritmos Iterativos

Projeto e corretude

Motivação

- Algoritmo iterativo
  - Resolve o problema em vários passos (usa repetição)
  - Cada passo deixa mais próximo da solução
- Provar corretude depois de projetar pode ser difícil
- Mais fácil projetar e provar corretude simultaneamente

- Estruturar a tarefa em pequenos passos
  - Se os passos forem cumpridos, teremos um projeto e uma prova de corretude

Sequência de ações x Sequência de assertivas

- Algoritmo podem ser vistos como:
  - Sequência de ações
  - Sequência de fotos do estado do comp.

- Visão dupla melhora a compreensão
  - Fácil se perder em if’s e while’s

- Expressamos estados com assertivas
  - O que deve ser verdadeiro em cada ponto
  - Pré-condições e pós-condições
  - Estados intermediários
  - Geral o bastante p/ facilitar entendimento

- Ação garante assertiva, base na anterior
  - $\langle assertiva_i \rangle$ & $code_i \Rightarrow \langle assertiva_{i+1} \rangle$
  - Corretude: provar a pós-condição

Sequência de ações x Sequência de assertivas

- As assertivas geralmente são comentários
    - Pode intercalar com português (cuidado com ambiguidades)
    - Podemos implementá-las para usar como ferramenta de depuração
        - Porém, algumas demandam muita computação (não são viáveis implementar)
    - Todas as suposições feitas devem estar explícitas nas assertivas

- Estruturação baseada nos dados

Passos para projetar um algoritmo iterativo

- Estrutura:

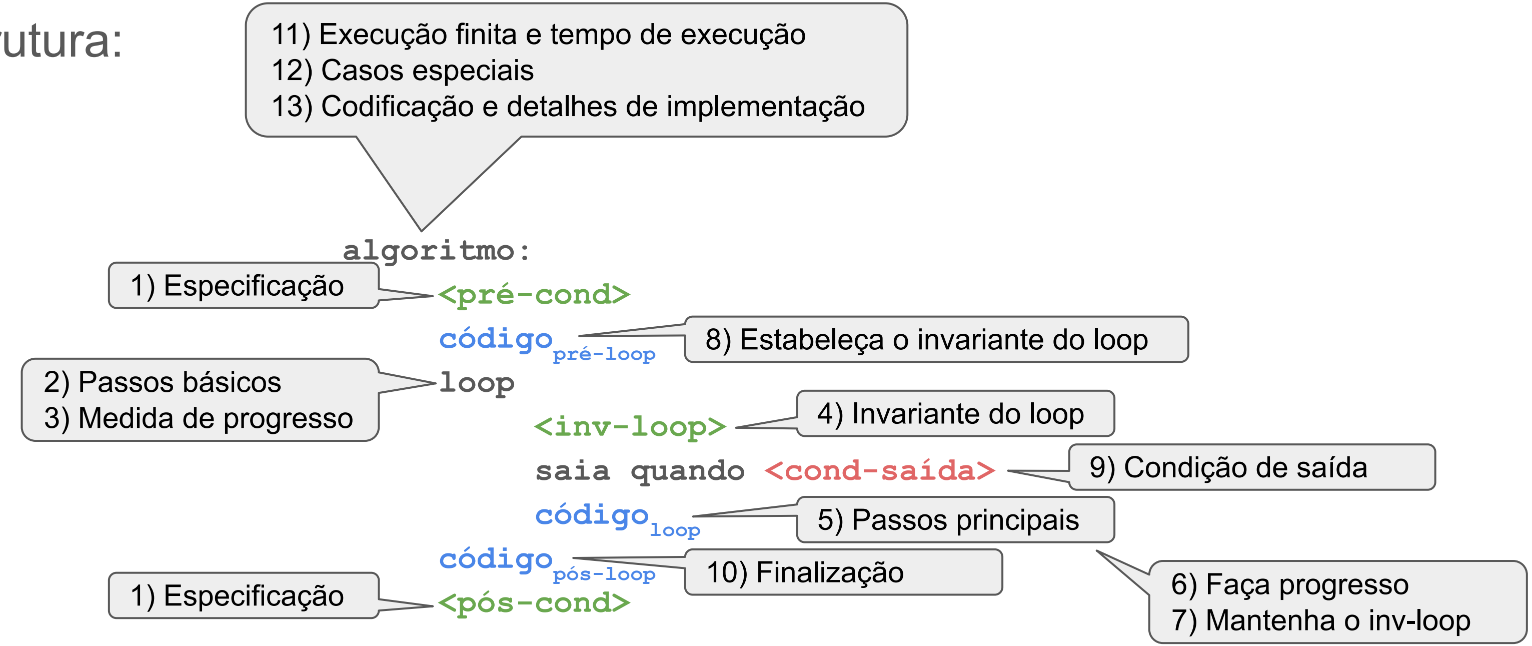
```
algoritmo:
    <pré-cond>
    código_pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código_loop
    código_pós-loop
    <pós-cond>
```

```
algoritmo:
    <pré-cond>
    código_pré-loop
    while not <cond-saída>

        código_loop
    código_pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

- Estrutura:



Passos 1-11 fornecem prova formal de corretude!

Passos para projetar um algoritmo iterativo

1. Especificação
    - Definição precisa do que deve ser resolvido
    - Pré-condições: tudo que é assumido verdadeiro sobre a entrada
    - Pós-condições: tudo que deve ser satisfeito pela saída

    - Ex. (Find-Max): Posição do maior elemento em uma lista
        - Pré-condições: A entrada é uma lista `L(1..n)` com n números.
        - Pós-condições: A saída é um índice m tal que L(m) tem valor máximo. Em caso de empate, qualquer dos índices pode ser retornado.

    - Corretude depende da especificação: `<pre-cond> & code ⇒ <pos-cond>`
    - Contrato entre o implementador e o usuário (quem chama o algoritmo)
        - Implementador assume pré-condições e garante pós-condições
        - Usuário assume pós-condições sempre que fornecer entradas válidas (pré-condições)

Passos para projetar um algoritmo iterativo

2. Passos básicos
    - Projeto preliminar indicando em linhas gerais como cada iteração avança para a solução
    - Testes algumas iterações em instâncias simples
    - Ex. (Find-Max): Dois índices: i e m. O índice i percorre a lista (um elemento por iteração), e o índice m guarda a posição do maior (qualquer deles) encontrado até então.

3. Medida de progresso
    - Função que, dado o estado atual, fornece quando progresso foi feito ou quanto ainda falta
    - Valores inteiros
    - Algoritmos deve terminar: não pode ser infinito, e cada iteração deve gerar progresso
    - Ex.: quantidade de saídas produzidas, quantidade de entradas consideradas, tamanho do espaço de busca, etc
    - Ex. (Find-Max): Quantidade de elementos considerados até então (percorridos por i).

Passos para projetar um algoritmo iterativo

4. Invariante de loop
    - Assertiva colocada no início do loop, e deve ser verdadeira em todas as iterações
    - Parte mais difícil (criatividade), mas restante geralmente decorre facilmente
    - Descrição deveria dar uma imagem visual do estado das EDs
    - Deve garantir que a computação se mantém em direção à solução
    - Significativa: quando combinada com a condição de saída e com o código pós-loop, deve garantir a pós-condição
    - Alcançável: deve ser capaz de estabelecê-la e mantê-la
    - O que gostaria que fosse verdadeiro no meio da computação? É razoável?
        - Imagine uma iteração onde o invariante é satisfeito no início
        - É possível fazer progresso na iteração mantendo o invariante?
            - Se é fraco demais, você não tem o que precisa para fazer progresso
            - Se é forte demais, você avança mas não consegue manter o invariante
    - Ex.: (Find-Max): m tem posição do maior (qualquer deles) dentre os considerado até então

Passos para projetar um algoritmo iterativo

5. Passos principais (código do loop)
    - Assuma que está em uma iteração intermediária (não necessariamente a primeira)
    - Quais passos devem ser feitos em uma única interação?
        - Necessário fazer progresso e manter o invariante
    - Ex. (Find-Max): Avance o índice i. Se L(i) > L(m), copie o valor de i para m.

6. Faça progresso
    - Mostre que cada iteração avança em pelo menos uma unidade a medida de progresso
    - Pode ser necessário reforçar a medida de progresso ou corrigir o código do loop
    - Ex. (Find-Max): Cada iteração considera um novo elemento (avanço do índice i).

Passos para projetar um algoritmo iterativo

7. Mantenha o invariante do loop
   - Prove que o invariante do loop é mantido em cada iteração

     `<inv-loop′> & not <cond-saída> & código`$_{\texttt{loop}}$ ⇒ `<inv-loop′′>`

   - Técnica de prova:
     - Assuma execução no topo do loop
     - Assuma que o invariante do loop é satisfeito
     - Assuma que a condição de saída não é satisfeita
     - Execute uma iteração do código do loop. Como isso alterou as EDs?
     - Mostre que as alterações realizadas conservam o invariante do loop
   - Usamos `′` e `′′` para diferenciar o estado antes e depois da execução
     - Ex.: matematicamente, a atribuição `x = x+2` pode ser expressa com `x′′= x′+2`

Passos para projetar um algoritmo iterativo

7. Mantenha o invariante do loop
    - Ex. (Find-Max):
        - Assuma início da iteração. Início: i = i’ e m = m’. Final: i = i’’ e m = m’’.
        - Condição de saída é falsa: existe elemento ainda não considerado.

$$\underbrace{\overbrace{L(1) \qquad \cdots \qquad}^{L(m') \text{ é o maior}} \; \overbrace{L(i'')}^{\text{iteração}}}_{L(m'')=\max(L(m'),L(i''))}$$

        - Pelo código do loop:
            - m’’ = i’’, se L(i’’) > L(m’)
            - m’’ = m’, caso contrário
            - Ou seja, L(m’’) = max(L(m’), L(i’’))

Passos para projetar um algoritmo iterativo

8. Estabeleça o invariante do loop
    - Prove que o invariante do loop vale na 1a iteração

      <pre-cond> & $\texttt{código}_{\texttt{pré-loop}}$ ⇒ <inv-loop>
    - Técnica de prova
        - Assuma que está no início da execução
        - Assuma que a entrada satisfaz as pré-condições
        - Execute o código pré-loop
        - Mostre que o invariante do loop está satisfeito
    - Ex. (Find-Max):
        - No código pré-loop fazemos i = m = 1.
        - Como apenas o 1o elemento foi considerado, m é o índice para o maior

Passos para projetar um algoritmo iterativo

9. Condição de saída
    - Expressa cumprimento da tarefa do laço
    - Será usado na prova da <pos-cond>
    - Ex. (Find-Max): O índice i já percorreu por todos os elementos de L?

```
algoritmo:
    <pré-cond>
    código_pré-loop
    loop
        <inv-loop>
        saia quando <cond-saída>
        código_loop
    código_pós-loop
    <pós-cond>
```

Passos para projetar um algoritmo iterativo

10. Finalização
    - Mostre que após encerrar o loop seremos capazes de resolver o problema
      `<inv-loop> & <cond-saída> & código` $_{\texttt{pós-loop}}$ ⇒ `<pós-cond>`
    - Técnica de prova
      - Assuma que acabou de sair do loop
      - Pode assumir que inv-loop é satisfeito, pois vale no início de toda iteração, e o teste de saída é a 1a ação da iteração
      - Assuma que cond-saída é satisfeita, pois acabou de sair do loop
      - Execute o código pós-loop
      - Mostre que a pós-condição é satisfeita
    - Ex. (Find-Max):
      - inv-loop: m contém a posição do maior (qualquer deles) dentre os considerados
      - cond-saída: todos os elementos foram considerados
      - Concluímos que m contém a posição do maior em L (qualquer deles)
      - Então para satisfazer pós-cond basta retornar m no código pós-loop

Passos para projetar um algoritmo iterativo

11. Execução finita e tempo de execução
    - Mostre que o algoritmo não fica em loop infinito
        - Prove que o loop já terá encerrado quando a medida de progresso atingir um determinado valor finito
        - Número de iterações será este progresso total dividido pelo progresso de uma iteração
    - O tempo de execução: tempo código pré-loop + tempo código pós-loop + soma dos tempos de código do loop para cada iteração
        - Expressar em notação θ
    - Ex. (Find-Max):
        - Número de iterações é o tamanho da lista n (finito)
        - código pré-loop, código loop e código pós-loop são $\theta(1)$
        - Portanto, o algoritmo é $\theta(n)$.

Passos para projetar um algoritmo iterativo

12. Casos especiais
    - Comece projetando para um caso geral, e depois acrescente casos particulares
    - Verifique se os casos que já são atendidos pelo algoritmo (sem necessidade de mais código)
    - Implemente os casos não cobertos, verificando se os casos anteriores ainda são atendidos
    - Ex. (Find-Max): Teste entradas com valores repetidos, e com tamanho $n = 0$ e $n = 1$.

Passos para projetar um algoritmo iterativo

13. Codificação e detalhes de implementação
    - Forneça o pseudocódigo e detalhes de implementação
    - Detalhes de implementação podem ser ocultados por tipos abstratos de dados
    - Deixe em aberto detalhes que não fazem diferença (flexibilidade)
    - Ex. (Find-Max):

```
algoritmo Find-Max(L)
     <pre-cond>: L é um array com n números, n > 0.
     <pos-cond>: Retorna um i t.q. L[i] é máximo.
     i = 1; m = 1
     loop
          <inv-loop>: L[m] é máximo em L[1..i].
          saia quando i >= n
          i = i + 1
          se L[i] > L[m] então m = i
     retorne m
```

Passos para projetar um algoritmo iterativo

14. Prova formal
    - Os passos 1-11 são suficientes para garantir que o algoritmo funciona p/ toda entrada válida
    - Vimos no passo 8 que inv-loop é satisfeito na 1a iteração

      <pre-cond> & código$_{\text{pré-loop}}$ ⇒ <inv-loop>
    - Vimos no passo 10 que pós-cond é satisfeita se inv-loop vale na saída do loop

      <inv-loop> & <cond-saída> & código $_{\text{pós-loop}}$ ⇒ <pós-cond>
    - Basta mostrar que inv-loop é mantido em todas as iterações (por indução)
      - Caso base: pelo passo 8, inv-loop vale na 1a iteração.

        <pre-cond> & código$_{\text{pré-loop}}$ ⇒ <inv-loop>
      - Passo indutivo: pelo passo 7, inv-loop é mantido após a execução da iteração.

        <inv-loop'> & not <cond-saída> & código $_{\text{loop}}$ ⇒ <inv-loop''>

Tipos de algoritmos iterativos

- Mais da saída
  - Medida de progresso: quantidade da saída construída
  - Invariante do loop: a saída construída até então está correta
- Mais da entrada
  - Medida de progresso: quantidade da entrada considerada
  - Invariante do loop: se a entrada já considerada fosse completa, teríamos solução completa
- Estreitando o espaço de busca
  - Medida de progresso: tamanho do espaço no qual a busca foi restringida
  - Invariante do loop: se o objeto buscado está na entrada, então está no espaço restringido
- Trabalho realizado
  - Medida de progresso: alguma forma criativa de medir o trabalho realizado
  - Ex. (bubble sort): número de pares de elementos ainda fora de ordem

Ex. (mais da saída): Ordenação por seleção

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos: repetidamente selecione menor dentre os não selecionados, e coloque no final da lista de selecionados
3. Medida de progresso: número k de elementos já selecionados
4. Invariante do loop: os k selecionados são os k menores, e estão em ordem
5. Passos principais: encontre o menor dentre os não selecionados, e mova para a última posição dos selecionados
6. Faça progresso: sim, pois o k aumenta
7. Mantenha o invariante:
    - Pelo invariante anterior, o selecionado não é menor que os selecionados anteriormente
    - Pelo código do loop, o selecionado não é maior que nenhum dentre os não selecionados
    - Então ele pode entrar na posição k+1 da lista de selecionados

Ex. (mais da saída): Ordenação por seleção

8. Estabeleça o invariante: inicialmente k = 0 (nenhum foi selecionado)
9. Condição de saída: k = n
10. Finalização:
    - Pela condição de saída todos já foram selecionados
    - Pelo invariante os selecionados estão em ordem
    - Então basta retornar a lista de selecionados
11. Execução finita e tempo de execução:
    - Depende da estratégia para localizar o menor elemento

Ex. (mais da entrada): Ordenação por inserção

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos:
   - Repetidamente leia próxima entrada e coloque em posição que mantenha os lidos em ordem
3. Medida de progresso: número k de elementos já lidos
4. Invariante do loop: os k lidos estão em ordem
5. Passos principais:
   - Leia próxima entrada e coloque em posição que mantenha os lidos em ordem
6. Faça progresso: sim, pois o k aumenta
7. Mantenha o invariante:
   - Código do loop posiciona novo elemento de modo a manter os lidos em ordem (invariante)

Ex. (mais da entrada): Ordenação por inserção

8. Estabeleça o invariante: inicialmente k = 0 (nenhum foi lido)
9. Condição de saída: k = n (todos foram lidos)
10. Finalização:
    - Pela condição de saída todos já foram lidos
    - Pelo invariante os lidos estão em ordem
    - Então basta retornar a lista de elementos lidos
11. Execução finita e tempo de execução:
    - Depende da estrutura de dados que mantém os elementos lidos

Ex. (estreitando o espaço de busca): Busca binária

1. Especificação:
   - Entrada: Lista ordenada A[1..n] e chave de busca. Elementos podem ser repetidos.
   - Saída: Índice i tal que A[i] = chave, se a chave está na lista. Mensagem, caso contrário.
2. Passos básicos:
   - Divida o espaço de busca ao meio, e continue a busca na parte que contém a chave
4. Invariante do loop:
   - Se chave está na entrada, então ocorre em pelo menos um elemento de A[i..j].
     - Caso a chave seja repetida, pode ocorrer também fora de A[i..j]
3. Medida de progresso:
   - Número de elementos em A[i..j], ou seja, j-i+1

Ex. (estreitando o espaço de busca): Busca binária

5. Passos principais:
   - Encontre elemento do meio (posição $\lfloor (i+j)/2 \rfloor$)
   - Se chave ≤ A[meio], faça j = meio (continue a busca em A[i..meio])
   - Se chave > A[meio], faça i = meio+1 (continue a busca em A[meio+1..j])
9. Condição de saída: quando j-i+1 ≤ 1 (0 ou 1 no espaço de busca).
6. Faça progresso:
   - j-i+1 diminui (j reduz ou i aumenta se não for condição de saída)
     - Poderia não aumentar quando meio = j, mas isso ocorre quando j-i+1 ≥ 2
7. Mantenha o invariante:
   - Como A está ordenado, se chave está em A[i..j],
     - estará em A[i..meio] quando chave ≤ A[meio], ou
     - estará em A[meio+1..j] quando chave > A[meio]
8. Estabelecendo o invariante: faça i = 1 e j = n (lista inteira).

Ex. (estreitando o espaço de busca): Busca binária

10. Finalização:
    - Invariante diz que chave estará em A[i..j], se estiver na entrada
    - Condição de saída diz A[i..j] tem zero ou um elemento
        - Se tem zero, concluímos pelo invariante que chave não está na entrada
        - Se tem um, resta apenas testar se é igual a chave
11. Execução finita e tempo de execução
    - Como cada iteração reduz aprox. à metade o intervalo [i..j], número de iterações $\theta(\log n)$.
    - Cada iteração é $\theta(1)$, e também código pré e pós loop.
    - Então, o algoritmo é $\theta(\log n)$.
12. Casos especiais:
    - Se chave não está na lista, iremos alcançar uma sublista vazia
13. Codificação e detalhes de implementação:
    - Podemos incluir o teste chave = A[meio], reduzindo o número de iterações
        - Na prática deixa o algoritmo mais lento

Ex. (trabalho realizado): Bubble sort

1. Especificação: rearranjar lista com n valores em ordem não decrescente
2. Passos básicos:
   - Inverter pares de elementos consecutivos que estão fora de ordem
3. Medida de progresso:
   - Involução: par de elementos fora de ordem ($1 \leq i < j \leq n$, $A[i] > A[j]$)
   - Medida: número de involuções. Ex.: [1,2,5,4,3,6] tem 3 involuções.
4. Invariante do loop: temos uma permutação dos elementos da entrada
5. Passos principais:
   - Encontrar pelo menos um par de elementos consecutivos fora de ordem, e invertê-los
6. Faça progresso: reduz pelo menos uma involução
7. Mantendo o invariante: uma inversão mantêm os elementos originais

Ex. (trabalho realizado): Bubble sort

8. Estabelecendo o invariante: lista de entrada já é uma permutação
9. Condição de saída: nenhuma involução restante (lista está ordenada)
10. Finalização:
    - Invariante: lista é uma permutação dos elementos de entrada
    - Condição de saída: lista está ordenada
    - Basta retornar a lista
11. Execução finita e tempo de execução
    - Máximo de involuções (ordem inversa): $n(n-1)/2 \in \theta(n^2)$
    - Cada iteração remove pelo menos uma involução, resultando em $O(n^2)$ iterações
    - Se o tempo para encontrar uma involução é $\theta(n)$, teremos um algoritmo $O(n^3)$
    - Será $O(n^2)$ se acrescentarmos no invariante que os k maiores estão nas últimas k posições