Tópicos Avançados em Algoritmos

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

23 de janeiro de 2019

Corretude e Eficência em Competição

Corretude e eficiência de algoritmos em Competições

- Técnicas para a corretude:
  - Afirmação de Operação: Basta afirmar as operações que o algoritmo propõe.
  - Loop Invariante: *Um invariante é uma propriedade que característica do algoritmo, vale antes da primeira instância do loop, se mantém invariante ao longo da execução de um loop, e que após o término do loop tem uma função importante para provar a corretude do algoritmo*.
  - Prova por indução: *Demonstre por indução que é possível resolver o problema computacional, construa um algoritmo baseado na prova por indução*.

Afirmação da Operação

Problema Computacional:

Apresente um algoritmo que recebe um valor inteiro e informe se este valor é par, ímpar ou nulo

Algoritmo $\textsc{Par}(n)$
    se $n|2 = 0$ **então**
        se $n = 0$ **então**
            “Imprimir: NULO”
        senão
            “Imprimir: PAR”
    senão
        “Imprimir: IMPAR”

Afirmação da Operação

- Na linha 2, é verificado o resto da divisão do número por 2, se o resto é zero, o número pode ser par ou nulo.
- Na linha 3, verifica-se, então se sendo o resto 0, o número em si é 0, neste caso o número é nulo.
- Na linha 4, imprime o fato do número ser nulo
- Na linha 5, o resto é zero, mas o número não é 0, logo o número é par.
- Na linha 6, imprime o fato do número ser par.
- Na linha 7, o número verificado não tem o resto sua divisão por 2 nulo, logo o número é ímpar.
- Na linha 8, imprime o fato do número ser ímpar.

Afirmação da Operação

- Normalmente este tipo de verificação apenas atesta o óbvio, que já está escrito no próprio código.
- Usualmente evita-se o detalhe linha a linha e faz-se um resumo de toda a operação, enfatizando a ação mais importante.
- Em um algoritmo misto que possui parte de afirmação e loops (ou recursão), a parte de afirmação aparece de forma implícita na prova que se faz da iteração (ou recursão).

Loops e Indução

- Um loop pode aparecer em um algoritmo de duas formas:
  - Laço: *for*, *while*, *do*, *repeat*, ...
  - Recursão
- O fato é: ambos são loops, e ambos podem ser traduzidos, de laço para recursão e vice-e-versa.
- Tudo se reduz ao fato de que são uma descrição algorítmica de uma indução finita.

Loops e Indução

Problema Computacional

Dada uma sequência A de números, de tamanho n, identificar a posição i que um número de valor x está na sequência, ou retornar 0 caso o número não conste da sequência.

Solução

- Podemos provar por indução que sabemos resolver o problema e criar um algoritmo recursivo para o problema.
- Podemos criar um algoritmo iterativo (possui laços) e provar correto por loop invariante

De qualquer forma a solução é a mesma: *busca binária* e a prova do loop invariante é justamente a indução que usaremos para criar o algoritmo recursivo.

Loops e Indução

Provando por Indução

Para o intervalo $[ini, fim]$ da sequência A de tamanho n $(fim - ini + 1 = n)$ eu sei dizer se o valor x está presente e sua posição, ou não.

Base: $n = 0$:
Se $n = 0$ então o valor x não está presente, logo $i = 0$.
Hipótese de Indução: $k < n$:
$fim - ini + 1 = k \rightarrow$ eu consigo encontrar a posição i do valor x ou $i = 0$.

Loops e Indução

Passo: n

Seja A definido no intervalo $[ini..fim]$ com $fim - ini + 1 = n$, vamos tomar o elemento de índice $meio = \frac{ini + fim}{2}$. Se $A[meio] = x$ então $i = meio$,

senão, se $x < A[meio]$, como os números estão em ordem crescente, se x estiver presente está no intervalo $[ini, meio - 1]$, caso contrário, no intervalo $[meio + 1, fim]$. Por hipótese de indução eu tenho a resposta correta para ambos intervalos possíveis

Loops e Indução

Algoritmo

Algoritmo BUSCABINARIA(A,ini,fim,x)
  se $ini > fim$ **então retorne** 0
  senão
    $meio = (ini + fim)/2$
    se $A[meio] = x$ **então retorne** $meio$
    senão
      se $x < A[meio]$ **então**
        retorne *BuscaBinaria(A,ini,meio-1,x)*
      senão
        retorne *BuscaBinaria(A,meio+1,fim,x)*

A solução é obtida com: *BuscaBinaria(A,1,n,x)*

Loops e Indução

Traduzindo para a forma iterativa

1: **Algoritmo** BUSCABINARIA(A,1,n,x)
2: $i = 0$
3: $ini = 1$; $fim = n$
4: $meio = (ini + fim)/2$
5: **enquanto** $ini \leqslant fim$ **e** $A[meio]! = x$ **faça**
6: **se** $x < A[meio]$ **então** $fim = meio - 1$
7: **senão** $ini = meio + 1$
8: $meio = (ini + fim)/2$
9: **se** $A[meio] = x$ **então** $i = meio$
10: **retorne** i

Loops e Indução

Invariane do Loop das linhas 5 a 8

Antes da iteração k, se o valor x estiver na sequência, ele estará no intervalo $[ini..fim]$ ou ele não está em definitivo.

Prova do invariante

Iniciação: Antes da primeira iteração, $ini = 1$ e $fim = n$, se x estiver na sequência, ele estará neste intervalo (que é a sequência toda), caso contrário ele não está na sequência em definitivo.
Manutenção: Antes da iteração k (onde $ini \leqslant fim$), com certeza (H.I.) se x estiver na sequência, ele estará no intervalo $[ini..fim]$ ou ele não está na sequência, em definitivo. Na linha 5, já verificamos se x está na posição *meio* (um dos critérios de término).

Loops e Indução

Prova do invariante..continuação

Se $x \neq A[meio]$, comparamos x com o valor $A[meio]$ se x menor que $A[meio]$ e sendo a sequência crescente, x, estando na sequência, só poderá estar no intervalo $[ini, meio - 1]$ logo $fim = meio - 1$, caso contrário só poderá estar no intervalo $[meio + 1, fim]$ logo $ini = meio + 1$, de qualquer forma para antes da próxima iteração, o invariente continua válido.

Término: Um critério de parada é $A[meio] = x$, logo x está na sequência na posição $i = meio$, outro critério de parada é $ini > fim$, que representa uma sequência vazia, logo x não está na sequência.

Corretude em provas de competição

- Como o juiz verifica a corretude de um algoritmo?
  - Testes, o algoritmo deve funcionar para todas a situações previstas.
  - São oferecidas entradas válidas dentro do limite indicado dos valores de entrada.
  - São comparadas as saídas do algoritmo com a saída padrão esperada para cada entrada.
- Um algoritmo incorreto pode ser validado como correto?
  - Sim, desde que esta algoritmo produza as saídas corretas para todos os testes de entrada.
  - Se soubessemos qual a sequência de saída a ser gerada, um algoritmo que simplesmente emitisse a sequência de saída seria válido, mesmo que ele não resolva o problema.
- Os testes sofrem da falha lógica na prova de corretude dos algoritmos: *um exemplo não prova que a teoria é válida*

Corretude em provas de competição

- Qual o custo de um algoritmo incorreto?
  - 20 minutos a cada submissão incorreta (somados como penalidade ao submeter uma versão correta).
- Qual o custo de provar a corretude de um algoritmo?
  - Em competições é alta, pois você pode tentar provar correto algo que possua uma falha de lógica, logo sua falha de lógica seria uma premissa para a prova de corretude (uma falácia) sem que se aperceba disto.
- Em suma, em competições, tentamos analisar a lógica empregada como correta e não analisar a corretude do algoritmo construído, não vale o custo de tempo, comparado com o custo de uma submissão incorreta.

Eficiência

- A prova da corretude não é importante na competição (o importante é estar correto e não provar que está correto)
- A noção da assintocidade da solução é de extrema importância.
  - Muitos problemas exigem soluções eficientes para seus problemas, ou seja, soluções que precisam ser $O()$ da solução prevista.
  - Também não é necessário provar a assintocidade de um problema, mas, dada a técnica empregada na solução é possível identificar

Eficiência

Exemplo: URI 2091 - Número Solitário

O problema: Será dado a você um vetor com N números, onde todos estarão em pares. Porém um desses números acabou ficando sem par, você consegue identificar qual é esse número ?
Por exemplo, A = 1, 1, 3, 3, 5, 5, 5, o número que ficou sozinho foi o 5.
Entrada: A entrada é composta por vários casos de teste. Cada caso de teste é composto por uma linha contendo um inteiro N ($1 \leqslant N < 10^5$), seguida por N números A ($0 \leqslant A \leqslant 10^{12}$). A entrada termina quando $N = 0$ e não deve ser processada.
Saída: Para cada caso de teste imprima apenas o número que ficou sozinho. É garantido que apenas um número está sozinho.

Exemplo de Entrada:	**Exemplo de Saída:**
5	4
1 3 4 3 1	1
3	5
1 1 1	
7	
1 1 3 3 5 5 5	
0	

Eficiência

Solução:

Algoritmo SOLUCAO
  $n \leftarrow ENTRADA$
  enquanto $n \neq 0$ **faça**
    $fim \leftarrow falso$
    para $i = 1$ **até** n **faça**
      $A[i] \leftarrow ENTRADA$
    para $i = 1$ **e** $\neg fim$ **até** n **faça**
      $cont \leftarrow 0$
      para $j = 1$ **até** n **faça**
        se $A[i] = A[j]$ **então** $cont \leftarrow cont + 1$
      se $cont|2 = 1$ **então**
        $SAIDA \leftarrow A[i]$
        $fim \rightarrow verdadeiro$
    $n \leftarrow ENTRADA$

Eficiência

- Fato: Nosso código está correto, ele devolve corretamente o número solitário.
- Assintocidade do nosso código: facilmente $O(n^2)$ (Por quê? Melhor caso? Pior Caso?)
- A entrada fala em $n = 10^5$, o que acontece quando rodamos com $n = 10^5$?

```
time ./uri.2091.v1 < uri.2091.big.in
```

... ← A saída foi omitida, mas está correta

```
real 5m37.070s
user 5m35.871s
sys 0m1.193s
```

- Mais de 5 minutos para rodar! Dá para fazer melhor???

Eficiência

Solução Melhor: Entrada será ordenada

Algoritmo SOLUCAO
    $n \leftarrow ENTRADA$
    enquanto $n \neq 0$ **faça**
        $fim \leftarrow falso$
        para $i = 1$ **até** n **faça** $A[i] \leftarrow ENTRADA$
        $quicksort(A)$
        $i \leftarrow 1$
        enquanto $\neg fim$ **faça**
            se $i = n$ **ou** $A[i] \neq A[i+1]$ **então**
                $SAIDA \leftarrow A[i]$
                $fim \leftarrow verdadeiro$
            senão $i \leftarrow i + 2$
        $n \leftarrow ENTRADA$

Eficiência

- Fato: Nosso código está correto, ele devolve corretamente o número solitário.
- Assintocidade do nosso código: facilmente $O(n \log n)$ (Por quê? Melhor caso? Pior Caso? Pode dar ruim!!!)
- A entrada fala em $n = 10^5$, o que acontece quando rodamos com $n = 10^5$?

```
time ./uri.2091.v2 < uri.2091.big.in
...
```
← A saída foi omitida, mas está correta

```
real 0m0.815s
user 0m0.811s
sys 0m0.005s
```

- Uau!!! Baixamos de 5 minutos para menos de 1 s. Dá para fazer melhor??? Ainda assim temos TLE se usarmos este código.

Eficiência

- Vamos à solução ótima, e aí já começamos a responder àquela pergunta: Porque estudamos Arquitetura? Software Básico?
- Sabemos que no computador cada número possui uma representação binária.
- Ao operarmos com os números podemos fazer operações binárias, além da aritmética, uma das operações é o OU EXCLUSIVO.
    - $A \oplus A = 0$
    - $A \oplus 0 = A$
    - $A \oplus B \oplus C = A \oplus C \oplus B = B \oplus A \oplus C = \cdots$
    - A ordem das parcelas não altera o resultado.

Eficiência

Solução Ótima: Operação OU EXCLUSIVO de todos

Algoritmo SOLUCAO

$n \leftarrow ENTRADA$

enquanto $n \neq 0$ **faça**

$v \leftarrow ENTRADA$

$ultimo \leftarrow v$

para $i = 2$ **até** n **faça**

$v \leftarrow ENTRADA$

$ultimo \leftarrow ultimo \oplus v$

$SAIDA \leftarrow ultimo$

$n \leftarrow ENTRADA$

Eficiência

- Fato: Nosso código está correto, ele devolve corretamente o número solitário.
- Assintocidade do nosso código: facilmente $O(n)$ (Não tem como melhorar, pois temos de ler n números)
- A entrada fala em $n = 10^5$, o que acontece quando rodamos com $n = 10^5$?

  ```
  time ./uri.2091.v3 < uri.2091.big.in
  ```

  1 $\leftarrow$ A saída é correta, o número solitário é 1

  ```
  real 0m0.670s
  user 0m0.663s
  sys 0m0.007s
  ```

- Conseguimos melhorar, este código será aceito por ser de eficiência ótima.

Bibliotecas prontas

- Nós utilizamos na versão 2 deste código uma biblioteca pronta: `qsort(...)`
- Até sabemos implementar quicksort, mas sendo um algoritmo padrão e conhecido, por que não utilizá-lo de uma biblioteca
- Ganhamos em eficiência, não na execução, mas na escrita do código ao trabalhar com o reuso de códigos
- São vários níveis de reuso, desde reescrever um código que você já tem, até usar um código que já está pronto.
- As próximas aulas vamos trabalhar com reuso a partir de bibliotecas prontas, em especial, estruturas prontas: Fila, Pilha, ...