Universidade Federal do Ceará – Departamento de Computação

CK0019–T02A–2014.2 Avaliação Parcial 1 – Parte 1 2014-09-10

Aluno(a) (matrícula e nome):

1. (2 pontos) Nesta questão, você deve argumentar que o algoritmo abaixo é correto. Mais especificamente, você deve mostrar que ele sempre termina, e para isso você deve apresentar E PROVAR um variante do laço do algoritmo; além disso, você deve ENUNCIAR um invariante para o laço do algoritmo, e argumentar, com base nesse invariante, que o valor retornado pelo algoritmo é correto. Observação: o invariante deve ser enunciado, mas não precisa ser demonstrado.

```
algoritmo: mdc_euclides             | 01. m := mín{a,b} , M := máx{a,b}
entrada: números naturais "a" e "b" | 02. ENQUANTO m != 0
         // a >= 1, b >= 1          | 03. | r := M mod m
saída: um número natural            | 04. | M := m
       // o M.D.C. de "a" e "b"     | 05. | m := r
                                    | 06. RETORNE M.
```

==========

Resposta:

OBSERVAÇÃO: abaixo é apresentada uma demonstração completa da correção do algoritmo; para responder a questão acima, porém, os trechos que provam os invariantes não são necessários.

Mostrarei primeiramente que "0 <= m <= M" é um invariante do laço:

1) De fato, como mín{a,b} <= máx{a,b}, então m <= M logo antes do início do laço. Além disso, como a >= 1 e b >= 1, então m >= 1 > 0 também vale.

2) Supondo que 0 <= m <= M é verdade no início de uma iteração do laço, mostrarei que também é verdade ao fim da iteração em questão. De fato, observe que r < m é verdade logo após a execução da linha 3 (pois o resto de uma divisão é sempre menor que o divisor). Assim, as atribuições das linhas 3 e 4 tornam m <= M novamente verdade ao fim da iteração em questão. Além disso, o resto da divisão de números positivos é sempre não negativo, ou seja, r >= 0 vale logo após a linha 3, e portanto m >= 0 vale ao final da iteração, CQD.

Agora, mostrarei que M+m é um variante do laço do algoritmo:

1) Primeiramente, mostrarei que o valor de m+M sempre decresce com a execução de uma iteração do laço. De fato, sejam v e V os valores de m e M no início de uma iteração qualquer do laço. Nesse caso, o valor de m+M no início da iteração é v+V, ao passo que, ao final, é (V mod v)+v. Observe que v <= V, pois nós já mostramos que m <= M é um invariante do algoritmo. Logo, V mod v < v <= V, e portanto (V mod v) + v < V + v, CQD.

2) Mostrarei agora que m+M >= 0 é sempre verdade no início de cada iteração do

laço. De fato, logo antes da primeira iteração, como a >= 1 e b >= 1, então m >= 1 e M >= 1, e portanto m+M >= 2, CQD. Além disso, ao final de uma iteração qualquer temos M >= m >= 0 (pelo invariante anterior), e portanto m+M >= 0, CQD.

Observe que, como m+M é um variante do laço do algoritmo, então o laço em questão sempre termina, e portanto o algoritmo também sempre termina.

Agora, mostrarei que o algoritmo sempre retorna uma resposta correta.
Para tanto, mostrarei primeiramente que a afirmação abaixo é um invariante do laço do algoritmo:

A) Se m > 0, então MDC(m,M) = MDC(a,b); e
B) Se m = 0, então M = MDC(a,b).

1) É imediato ver que "A" vale logo antes do início do laço, pois m >= 1 e, se a <= b, então MDC(m,M) = MDC(a,b), e, se b < a, então MDC(m,M) = MDC(b,a) = MDC(a,b), CQD.

2) Suponhamos agora que a afirmação é verdade no início de uma iteração qualquer do laço do algoritmo; nós mostraremos que essa afirmação continua verdadeira ao final da iteração em questão.
De fato, são dois casos:

Caso 1: ao final da iteração, m = 0. Logo, no início da iteração, vale m <= M e M mod m = 0, ou seja, M é múltiplo de m, e portanto MDC(M,m) = m. Além disso, como MDC(m,M) = MDC(a,b) é verdade no início da iteração, então m = MDC(a,b) é verdade no início da iteração, e portanto M = MDC(a,b) é verdade ao fim da iteração em questão.

Caso 2: ao final da iteração, m > 0. Nesse caso, sejam v e V os valores de m e M no início da iteração. Logo, MDC(v,V) = MDC(a,b). Além disso, seja k = MDC(v,V), e sejam x e X os números naturais tais que V = k*X e v = k*x (logo, X mod x != 0). Observe que, ao final da iteração, temos M = v = k*x e m = V mod v = (k*X) mod (k*x) = k*(X mod x). Logo, k é múltiplo comum de M e m ao final da iteração, isto é, MDC(m,M) >= k vale ao final da iteração. Resta, então, mostrar que essa inequação vale na igualdade ao fim da iteração em questão. De fato, se MDC(m,M) > k fosse verdade ao fim da iteração, e como k é divisor de v e de V mod v, então teríamos que MDC(m,M) = k*y, para algum y > 1 tal que v = k*y*x' e V mod v = k*y*m'. Logo, k*y seria divisor tanto de v quanto de V, e, como k*y > k, então MDC(v,V) > k, contradizendo a definição de k. Logo, MDC(m,M) = k é verdade ao final da iteração em questão, e portanto MDC(m,M) = MDC(a,b) continua verdade ao fim da iteração, CQD.

Para concluir, observe que, ao fim do laço, temos m = 0, pois a condição do laço é falsa. Logo, pelo invariante acima, temos M = MDC(a,b), e portanto o valor retornado é correto, CQD.
==========

2. (2 pontos) Dado que $t(n) = 2 * t(\lfloor n/2 \rfloor) + (n+17)n + log_2(n-3)$ se $n \geq 4$, prove que $t = \Theta(n^k)$, para algum k à sua escolha. Para provar que $t = O(n^k)$, você deve usar indução; para provar que $t = \Omega(n^k)$, você pode não utilizar indução, se assim preferir. Consulte o professor caso deseje utilizar algum resultado sem demonstrá-lo.

Demonstração. Mostrando que $t = O(n^2)$, isto é, que existem $c, n_0 > 0$ tais que, $\forall n \geq n_0$, $t(n) \leq c * n^2$. De fato, sejam $n_0 = 1$ e c igual ao máximo entre 4 e $\max\{t(n) : n < 18\}$. Nós provaremos o resultado por indução em n:

- Base ($1 \leq n < 18$): Temos $t(n) \leq c \leq c * n^2$, CQD.
- H.I.: $t(m) \leq c * m^2, \forall m < n$.
- Passo ($n \geq 18$):

$$
\begin{aligned}
t(n) &= 2 * t(\lfloor n/2 \rfloor) + (n+17)n + log_2(n-3) \\
&\leq 2 * t(\lfloor n/2 \rfloor) + (n+17)n + log_2(n) && \rightarrow n-3 < n \text{ e } log_2 \text{ é crescente} \\
&\leq 2 * t(\lfloor n/2 \rfloor) + (n+17)n + n && \rightarrow log_2(n) \leq n \\
&= 2 * t(\lfloor n/2 \rfloor) + (n+18)n \\
&\leq 2 * t(\lfloor n/2 \rfloor) + (n+n)n && \rightarrow 18 \leq n \\
&= 2 * t(\lfloor n/2 \rfloor) + 2n^2 \\
&\leq 2c(\lfloor n/2 \rfloor)^2 + 2n^2 && \rightarrow \lfloor n/2 \rfloor < n \text{ e H.I.} \\
&\leq 2c(n/2)^2 + 2n^2 \\
&= 2cn^2/4 + 2n^2 \\
&= (c/2+2)n^2 \\
&\leq c * n^2, \text{ CQD.} && \rightarrow c/2 + 2 \leq c \iff c + 4 \leq 2c \iff c \geq 4
\end{aligned}
$$

Mostrando agora que $t = \Omega(n^2)$, isto é, que existem $c, n_0 > 0$ tais que, $\forall n \geq n_0$, $0 \leq c * n^2 \leq t(n)$. De fato, sejam $n_0 = 4$ e $c = 1$. Temos, para $n \geq n_0$: $t(n) = 2 * t(\lfloor n/2 \rfloor) + (n+17)n + log_2(n-3) \geq 0 + n^2 + 17n + 0 \geq 1 * n^2 \geq 0$, CQD. (Observação: aqui nós usamos o fato – implícito – de que $t(n) \geq 0$ para todo n.) □

3. (1 ponto) Mostre que, se $f = o(g)$ e $h = \omega(g)$, então $f = o(h)$.

Demonstração. Temos que mostrar que $f = o(h)$, isto é, que, para todo $c > 0$, existe $n_0 > 0$ tal que, $\forall n \geq n_0$, $0 \leq f(n) < c * h(n)$.

Como $h = \omega(g)$, então, para todo $c > 0$, existe $n_0 > 0$ tal que, $\forall n \geq n_0$, $0 \leq c * g(n) < h(n)$. Seja, então, $n' > 0$ tal que, $\forall n \geq n'$, $0 \leq g(n) < h(n)$.

Além disso, como $f = o(g)$, então, para todo $c > 0$, existe $m(c)$ tal que, $\forall n \geq m(c)$, $0 \leq f(n) < c * g(n)$.

Definamos então $l(c) = \max\{n', m(c)\}$, $\forall c > 0$. Sejam, agora, $c > 0$ e $n \geq l(c)$. Como $n \geq l(c) \geq n'$, então $g(n) < h(n)$. Além disso, como $n \geq l(c) \geq m(c)$, então $0 \leq f(n) < c * g(n)$. Logo, temos $0 \leq f(n) < c * g(n) < c * h(n)$, o que prova o resultado. □

— Boa prova! —

Universidade Federal do Ceará – Departamento de Computação

CK0019–T02A–2014.2 Avaliação Parcial 1 – Parte 2 2014-09-12

Aluno(a) (matrícula e nome):

4. (2,5 pontos) Escreva um algoritmo que particiona um vetor `A[1..n]` em três partes, `A[1..m]`, `A[m+1..M-1]` e `A[M..n]`, que devem conter respectivamente os elementos de `A` que são menores que, iguais a e maiores que um certo valor `v`. A entrada do algoritmo deve ser o vetor `A[1..n]` (`n >= 1`) e o valor `v` (que não necessariamente pertence ao vetor), e a saída deve consistir nos índices `m` e `M`. Você deve argumentar brevemente que o seu algoritmo executa em tempo $O(n)$. Dica: use uma estratégia incremental para manter os intervalos.

Resposta.

```
================================================================================
Algoritmo: partic_3
Entrada: vetor A[1..n], valor "v".
Saída: índices "m" e "M".
--------------------------------------------------------------------------------
01. m := 0 , M := 1 , f := 1
    // INV.: 0 <= m < M <= f <= n+1
    //       A[1..m]: < v , A[m+1..M-1]: = v , A[M..f-1]: > v.
02. ENQUANTO f <= n
03. | e := A[f]
04. | SE e = v
05. | | A[f] := A[M] , A[M] := e , ++M
06. | SENÃO SE e < v
07. | | A[f] := A[M], A[M] := A[m+1] , A[m+1] := e , ++m , ++M
08. | ++f
09. RETORNE (m,M).
================================================================================
```

Para verificar que o algoritmo executa em tempo $O(n)$, primeiro observe que toda iteração do laço dele leva tempo constante. Além disso, observe que $n + 1 - f$ é um variante do laço do algoritmo. Logo, como f começa valendo 1, então os valores possíveis para $n + 1 - f$ são n, $n - 1$, $n - 2$, ..., 0, ou seja, a condição do laço é testada no máximo $n + 1$ vezes. Isso significa que o laço executa no máximo n iterações, e portanto que o algoritmo executa em tempo $O(n)$, como desejado. □

5. (2,5 pontos) Dados um vetor `A[1..n]` (`n >= 1`) e um valor `v`, dizemos que `v` possui a propriedade P com relação a A se pelo menos $1 + \lfloor n/2 \rfloor$ elementos de `A` são iguais a `v`. Escreva então um algoritmo que recebe um vetor `A` como entrada e que retorna um valor `v`, se `v` possui a propriedade P, ou que retorna `"nenhum"` se não existe elemento `v` que possua a propriedade P com relação a A. Explique o porquê de o seu algoritmo ser correto, e argumente também que ele executa em tempo $O(n \log n)$ (você pode utilizar o Teorema Mestre para isso).

Dica: use divisão-e-conquista, e considere que relação existe entre (1) existir um elemento `v` que possui a propriedade P com relação a `A` e (2) existir um elemento `v` que possui a propriedade P com relação a `C` ou com relação a `A \ C`, sendo `C` e `A \ C` quaisquer coleções complementares de elementos de `A` (complementares no sentido de que os elementos de `A` que não estão em `C` estão em `A \ C`, e vice-versa).

Dica: se o seu algoritmo não executar em tempo $O(n \log n)$, você ganhará pontuação parcial na questão; atenção, porém, porque não é difícil obter um algoritmo que execute na complexidade de tempo desejada.

Resposta.

```
============================================================
Algoritmo: prop_P
Entrada: vetor A[1..n]
Saída: um elemento de "A" ou o valor "nenhum".
------------------------------------------------------------
01. SE n = 1
02. | RETORNE A[1]
03. m := piso((1+n)/2)                ...
04. r := prop_P(A[1..m])          12. r := prop_P(A[m+1..n])
05. SE r != "nenhum"              13. SE r != "nenhum"
06. | c := 0                      14. | c := 0
07. | PARA i DE 1 A n            15. | PARA i DE 1 A n
08. | | SE A[i] = r              16. | | SE A[i] = r
09. | | | ++c                    17. | | | ++c
10. | SE c >= 1 + piso(n/2)      18. | SE c >= 1 + piso(n/2)
11. | | RETORNE r                19. | | RETORNE r
    ...                           20. RETORNE "nenhum".
============================================================
```

O algoritmo acima se baseia no seguinte fato:

> Se existe um elemento v que possui a propriedade P com relação a A, então, para qualquer partição $\{C, A \setminus C\}$ de A ($C \neq \varnothing \neq A \setminus C$), v possui a propriedade P com relação a C ou v possui a propriedade P com relação a $A \setminus C$.
>
> *Demonstração.* Suponha que existe v que possui a propriedade P com relação a A, e suponha também, por absurdo, que v não possui a propriedade P com relação a C e nem com relação a $A \setminus C$. Nós exibiremos então uma contradição em cada um dos seguintes casos:
>
> 1. $|C| = 2x$ e $|A \setminus C| = 2y$: como v não possui a propriedade P com relação a C e nem com relação a $A \setminus C$, então v ocorre no máximo x vezes em C e y vezes em $A \setminus C$. Logo, v ocorre no máximo $x + y$ vezes em A. Como $n = 2x + 2y$, então v não possui a propriedade P com relação a A, um absurdo.
> 2. $|C| = 2x + 1$ e $|A \setminus C| = 2y + 1$: mesmo argumento do caso anterior, exceto que $n = 2x + 2y + 2$.
> 3. $|C| = 2x$ e $|A \setminus C| = 2y + 1$: mesmo argumento, exceto que $n = 2x + 2y + 1$.
> 4. $|C| = 2x + 1$ e $|A \setminus C| = 2y$: idem caso anterior.
>
> Logo, v necessariamente possui a propriedade P com relação a C ou com relação a $A \setminus C$, CQD. □

Continuando a argumentação da correção do algoritmo, observe que, caso A possua apenas um elemento, então esse elemento possui a propriedade P, e portanto a resposta retornada pelo algoritmo é correta. Já quando A possui mais que um elemento, o algoritmo resolve o problema recursivamente para as metades $A[1..m]$ e $A[m + 1..n]$. Pelo resultado acima, se A possui um

elemento com a propriedade P, então esse elemento também possui a propriedade com relação a pelo menos uma das metades em questão. Assim sendo, caso haja elementos que possuam a propriedade P com relação a alguma dessas metades, o algoritmo verifica se tais elementos também possuem a propriedade com relação a A; em caso afirmativo, o elemento é retornado. Se o caso anterior não ocorre, então o algoritmo retorna `"nenhum"`, corretamente.

Com relação ao tempo de execução do algoritmo, observe que a seguinte função recorrente limita superiormente o tempo levado pelo algoritmo:

$$t(n) = 2t(\lceil n/2 \rceil) + \Theta(n).$$

Aplicando o caso 2 do Teorema Mestre, concluímos que $t(n) = \Theta(n \log n)$, e portanto que o algoritmo executa em tempo $O(n \log n)$, CQD. □

— Boa prova! —