PAA - Estudos para prova 3

Conteúdo

1 Teorema Mestre

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Figura 1: Teorma Mestre (Cormen)

□ Theorem 3.4

The solution of the recurrence relation $T(n) = aT(n/b) + cn^k$, where a and b are integer constants, $a \geq 1$, $b \geq 2$, and c and k are positive constants, is

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > b^k \\ O(n^k \log n) & \text{if } a = b^k , \\ O(n^k) & \text{if } a < b^k \end{cases}$$

Figura 2: Teorma Mestre (Manber)

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
($f(n)$ is polynomially smaller than $n^{\log_b a}$.)
Solution: $T(n) = \Theta(n^{\log_b a})$.
(Intuitively: cost is dominated by leaves.)

Case 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$.
[This formulation of Case 2 is more general than in Theorem 4.1, and it is given in Exercise 4.6-2.]
($f(n)$ is within a polylog factor of $n^{\log_b a}$, but not smaller.)
Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
(Intuitively: cost is $n^{\log_b a} \lg^k n$ at each level, and there are $\Theta(\lg n)$ levels.)
Simple case: $k = 0 \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.
($f(n)$ is polynomially greater than $n^{\log_b a}$.)
Solution: $T(n) = \Theta(f(n))$.
(Intuitively: cost is dominated by root.)

Figura 3: Teorma Mestre (Cormen, completo)

2 Indução

2.1 Avaliação de Polinômios 1

Depende da complexidade de se resolver x^n. Considerando que isso pode ser feito em tempo $\Theta(\log n)$, a relação fica $T(n) = T(n-1) + \Theta(\log n)$ cuja complexidade é $\Theta(n \log n)$.

Algorithm 1: AP1

Data: vetor de coeficientes a, grau n, ponto x a avaliar
Result: Avaliação do polinômio $a_0 + a_1 x + \cdots + a_n x^n$ no ponto x

```
1 if n == 0 then
2 |   return a[0];
3 end
4 y = AP1(a, n − 1, x);
5 return y + a[n] · x^n;
```

2.2 Avaliação de Polinômios 2

Relação $T(n) = T(n-1) + 1$, Complexidade $\Theta(n)$

Algorithm 2: AP2

Data: vetor de coeficientes a, grau n, ponto x a avaliar
Result: Avaliação do polinômio $a_0 + a_1x + \cdots + a_nx^n$ no ponto x

```
1 if n == 0 then
2 |  return (a[0], 1);
3 end
4 (y, potencia) = AP2(a, n − 1, x);
5 return (y + a[n] · x · potencia, x · potencia);
```

2.3 Avaliação de Polinômios 3

Considera que o polinômio de grau n pode ser reescrito como $a_0 + x(a_1 + a_2x + \cdots + a_nx^{n-1})$. Relação $T(n) = T(n-1) + 1$, Complexidade $\Theta(n)$

Algorithm 3: AP3

Data: vetor de coeficientes a, grau n, coeficiente inicial i e ponto x a avaliar
Result: Avaliação do polinômio $a_0 + a_1x + \cdots + a_nx^n$ no ponto x

```
1 if n == 0 then
2 |  return a[i];
3 end
4 y = AP3(a, n − 1, i + 1, x);
5 return a[i] + x · y;
```

2.4 Sub-grafo de grau mínimo K

Procedimento, num grafo de n vértices e m arestas, do cálculo do grau de um vértice e enfileiramento de vértices com grau $< k$ em $O(n+m)$. Remoção dos vértices e seus vizinhos que ficarem com grau $< k$ também $O(n+m)$.

Algorithm 4: SGM

Data: Grafo G e grau inteiro k
Result: subgrafo onde todos os vértices tem grau $\geq k$

```
1 if E(G) == ∅ then
2 |  return ∅];
3 end
4 if grau mínimo ≥ k then
5 |  return G;
6 end
7 ache v com grau < k;
8 return SGM(G − v, k);
```

2.5 Sub-vetor de soma máxima

Relação $T(n) = T(n-1) + \Theta(n)$, complexidade $O(n^2)$.

Algorithm 5: SSM

Data: vetor $v[]$ de tamanho n
Result: Soma máxima de um sub-vetor em v

```
1  if n==0 then
2  |  return 0;
3  end
4  soma = SSM(v, n − 1);
5  x = 0;
6  for i = n to 1 do
7  |  x = x + v[i];
8  |  soma = max(soma, x);
9  end
10 return soma;
```

2.6 Sub-vetor de soma máxima com Sufixo

Relação $T(n) = T(n-1) + \Theta(1)$, complexidade $\Theta(n)$.

Algorithm 6: SSM2

Data: vetor $v[]$ de tamanho n
Result: Soma máxima de um sub-vetor em v e maior sufixo

```
1  if n==0 then
2  |  return (0,0);
3  end
4  (soma, sufixo) = SSM(v, n − 1);
5  novasoma = max(soma, v[n] + sufixo);
6  novosufixo = max(0, v[n] + sufixo);
7  return (novasoma, novosufixo);
```

2.7 Celebridade em Grafos

Considera que a matriz de adjacências é dada. A indução está em que se descarta iterativamente vértices que não são celebridade, mesmo que a implementação

não seja recursiva. Complexidade $\Theta(n)$.

Algorithm 7: CELEB

```
Data: Grafo G (matriz de adjacências já montada)
Result: Índice da celebridade ou 0 se não há
1  i = 1;
2  j = 2;
3  for k = 3 to n do
4  |  if adj[i, j] == 0 then
5  |  |  j = k;
6  |  end
7  |  else
8  |  |  i = k;
9  |  end
10 end
11 if adj[i, j] == 0 then
12 |  cand = i;
13 end
14 else
15 |  cand = j;
16 end
17 for k = 1 to n do
18 |  if k ≠ cand AND (adj[i, cand] == 0 OR adj[cand, i] == 1) then
19 |  |  return 0;
20 |  end
21 end
22 return cand;
```

2.8 Centro de uma árvore

Implementação com fila e marcadores permite fazer em tempo linear. Termina quando a fila tem somente um ou dois elementos, que são a resposta. Complexidade $\Theta(n)$.

Algorithm 8: CENTRO

```
Data: Árvore T
Result: Vértice(s) do centro da árvore
1 if |V(T)| ≤ 2 then
2 |  return V(T);
3 end
4 Seja F o conjunto de folhas de T;
5 return CENTRO(T − F);
```

3 Divisão e Conquista

3.1 Maior Elemento de Vetor

Relação $T(n) = 2T(n/2) + O(1)$, complexidade $\Theta(n)$.

Algorithm 9: ME

Data: vetor $v[]$, início i e fim f
Result: maior elemento de $v[]$

```
1 if i == f then
2 |   return v[i];
3 end
4 meio = (i + f)/2;
5 maioresq = ME(v, 1, meio);
6 maiordir = ME(v, meio + 1, n);
7 return max(maioresq, maiordir);
```

3.2 Exponenciação 2

A primeira versão feita era iterativa. A primeira com paradigma recursivo era EXP2. Relação $T(e) = 2T(e/2) + O(1)$, complexidade $\Theta(e)$.

Algorithm 10: EXP2

Data: base b, expoente e
Result: b^e

```
1 if e == 0 then
2 |   return 1;
3 end
4 esq = EXP2(b, ⌊e/2⌋);
5 dir = EXP2(b, ⌈e/2⌉);
6 return esq · dir;
```

3.3 Exponenciação 3

Elimina uma chamada recursiva. Relação $T(e) = T(e/2) + O(1)$, complexidade $\Theta(\log e)$.

Algorithm 11: EXP3

Data: base b, expoente e
Result: b^e
1 **if** $e == 0$ **then**
2 | **return** 1;
3 **end**
4 esq = EXP3(b, $\lfloor e/2 \rfloor$);
5 dir = esq;
6 **if** *e é impar* **then**
7 | dir = $b \cdot$ dir;
8 **end**
9 **return** esq $\cdot$ dir;

3.4 Multiplicação números grandes em base 2

Relação $T(n) = 4T(n/2) + \Theta(n)$, complexidade $\Theta(n^2)$. Não é melhor que o método tradicional.

Algorithm 12: MUL

Data: Números grandes como vetores $x[]$, $y[]$ de n dígitos
Result: vetor $x \cdot y$
1 **if** $n == 1$ **then**
2 | **return** $x \cdot y$;
3 **end**
4 **return** $2^n \cdot \text{MUL}(X_L, Y_L, n/2) + 2^{n/2} \cdot \text{MUL}(X_L, Y_R, n/2) + 2^{n/2} \cdot \text{MUL}(X_R, Y_L, n/2) + \text{MUL}(X_R, Y_R, n/2)$;

3.5 Multiplicação números grandes em base 2 (versão melhor)

Usa o fato de que a multiplicação pode ser escrita na forma $2^n A + 2^{n/2} B + C$. Ou, sendo $D = (X_L + X_R)(Y_L + Y_R)$, na forma $2^n A + 2^{n/2}(D - A - C) + C$, o que reduz uma multiplicação. Relação $T(n) = 3T(n/2) + \Theta(n)$, complexidade

$\Theta(n^{\log_2 3}) = \Theta(n^{1.58})$.

Algorithm 13: MULT

Data: Números grandes como vetores $x[]$, $y[]$ de n dígitos
Result: vetor $x \cdot y$

```
1  if n == 1 then
2  |  return x · y;
3  end
4  A = MULT(X_L, Y_L, n/2);
5  C = MULT(X_R, Y_R, n/2);
6  D = MULT(X_L + X_R, Y_L + Y_R, n/2 + 1);
7  return 2^n · A + 2^{n/2} · (D − A − C) + C;
```

3.6 Menor distância entre par de pontos

Relação $T(n) = 2T(n/2) + O(n \log n)$, complexidade $O(n \log^2 n)$. Professor mostrou método para ordenar o vetor por x e por y uma vez só, que reduziria a complexidade para $O(n \log n)$, mas não fez a versão em algoritmo. Há algumas imprecisões no algoritmo que ele fez em sala. Tentei corrigir algumas, mas não é garantido.

Algorithm 14: PMP

Data: vetor de pontos P e tamanho n
Result: menor distância entre par de pontos

```
1  if n == 1 then
2  |  return ∞;
3  end
4  Ordenar P por coordenada x;
5  d1 = PMP(P, 1, n/2);
6  d2 = PMP(P, n/2 + 1, n);
7  d = min(d1, d2);
8  Ordenar P por coordenada y;
9  Eliminar pontos fora da faixa;
10 for i=1 to n do
11 |  for j=i+1 to i+min(11, n − i) do
12 |  |  if dist(P[i], P[j]) < d then
13 |  |  |  d = dist(P[i], P[j]);
14 |  |  end
15 |  end
16 end
17 return d;
```

3.7 Sub-vetor de soma máxima

Relação $T(n) = 2T(n/2) + O(1)$, complexidade $\Theta(n)$.

Algorithm 15: SSM

Data: indíces de início e fim **ini** e **fim**, vetor v
Result: soma dos elementos do sub-vetor de soma máxima

```
1  if ini == fim then
2  |  resp.ssm = max(0, v[ini]);
3  |  resp.pref = v[ini];
4  |  resp.suf = v[ini];
5  |  resp.total = v[ini];
6  end
7  meio = (ini + fim)/2;
8  esq = SSM(ini, meio, v);
9  dir = SSM(meio + 1, fim, v);
10 resp.ssm = max(esq.ssm, dir.ssm, esq.suf + dir.pref);
11 resp.pref = max(esq.pref, esq.total + dir.pref);
12 resp.suf = max(dir.suf, dir.total + esq.suf);
13 resp.total = esq.total + dir.total;
14 return resp;
```

4 Algoritmos Gulosos

São geralmente focados em problemas de otimização e muitas vezes não é possível pensar na forma de sub-problemas iguais, como nos demais paradigmas. O paradigma consiste em tomar seguidas decisões baseadas no que for a melhor escolha no momento, por isso seu nome.

4.1 Seleção de atividades

Complexidade $\Theta(n \log n)$, o tempo de ordenação domina o algoritmo.

Algorithm 16: SDA

Data: Vetor de atividades A
Result: S, maior lista possível de atividades não conflitantes

```
1  Ordene A por término (por b_i);
2  S = ∅;
3  termino = −∞;
4  for i=1 to n do
5  |  if A[i].a ≥ termino then
6  |  |  termino = A[i].b;
7  |  |  S = S ∪ A[i];
8  |  end
9  end
10 return S;
```

5 Programação Dinâmica

Sub-problemas com interseção, se armazena o resultado dos sub-problemas para não ter que calculá-los novamente.

5.1 N-ésimo elemento da Sequência de Fibonacci (top-down)

A linha 7 é executada $n-1$ vezes (para todos os números ≤ 2). A cada vez que ela é executada são feitas 2 chamadas. Portanto a rotina é chamada $2(n-1)+1$, contando a chamada externa. Portanto $\Theta(n)$.

Algorithm 17: FIB

Data: n número na sequência. Vetor f inicializado com valores negativos
Result: Valor do n-ésimo elemento da sequência de Fibonacci

```
1 if n ≤ 1 then
2 |  return n;
3 end
4 if f[n] ≥ 0 then
5 |  return f[n];
6 end
7 f[n] = FIB(n − 1) + FIB(n − 2);
8 return f[n];
```

5.2 N-ésimo elemento da Sequência de Fibonacci (Bottom-up)

Calcula de baixo pra cima, sem chamadas recursivas. Bem mais fácil de ver que complexidade é $\Theta(n)$.

Algorithm 18: FIB

Data: n número na sequência.
Result: Valor do n-ésimo elemento da sequência de Fibonacci

```
1 f[0] = 0;
2 f[1] = 1;
3 for i=2 to n do
4 |  f[i] = f[i − 1] + f[i − 2];
5 end
6 return f[n];
```

5.3 Problema da Mochila

A relação de recorrência é:

$$\mathrm{M}(n, C) = \begin{cases} 0, & \text{se } n = 0 \text{ ou } C = 0, \\ \mathrm{M}(n-1, C), & \text{se } p_n > C \\ \max(\mathrm{M}(n-1, C), v_n + \mathrm{M}(n-1, C-p_n), & \text{outros casos} \end{cases}$$

Essa relação daria complexidade $O(2^n)$. A implementação com programação dinâmica (abaixo) consegue uma complexidade de $O(n \cdot C)$.

Algorithm 19: M

Data: n número de objetos, C capacidade da mochila, vetor p de pesos e vetor$v$ de valores, matriz m de resultados intermediários com valores negativos
Result: maior valor possível de por na mochila

```
1 if n == 0 ou c == 0 then
2 |  return 0;
3 end
4 if m[n][C] > 0 then
5 |  return m[n][C];
6 end
7 if p[n] > C then
8 |  m[n][C] = M(n − 1, C);
9 |  return m[n][C];
10 end
11 m[n][C] = max(M(n − 1, C), v[n] + M(n − 1, C − p[n]));
12 return m[n][C];
```

5.4 Seleção de atividades com peso

Complexidade $O(n^2)$ se a busca das linhas 4 a 6 for feita com busca linear, $O(n \log n)$ com utilização de busca binária.

Algorithm 20: SAP

Data: lista de atividades A ordenada por data fim, com elemento $A_0.b = -\infty$ e $p_0 = 0$
Result: Maior peso possível de se obter com as atividades

```
1 pd[0] = 0;
2 for i=1 to n do
3 |   j = 0;
4 |   while A[j].b < A[i].a do
5 |   |   j = j + 1;
6 |   end
7 |   pd[i] = max(pd[i − 1], p[i] + pd[j]);
8 end
9 return p[n];
```

5.5 Maior sequência crescente

Complexidade $O(n^2)$, por causa dos dois loops aninhados.

Algorithm 21: MSC

Data: vetor v de tamanho n
Result: soma da maior subsequência crescente

```
1  resposta = 1;
2  for i=1 to n do
3  |   msc[i] = 1;
4  |   for j=1 to i-1 do
5  |   |   if v[j] < v[i] then
6  |   |   |   if msc[j] + 1 > msc[i] then
7  |   |   |   |   msc[i] = msc[j + 1];
8  |   |   |   end
9  |   |   end
10 |   end
11 |   resposta = max(resposta, msc[i]);
12 end
13 return resposta;
```

5.6 Maior sequência crescente v.2

Complexidade $O(n \log n)$, considerando busca binária para achar o maior j tal que $a[j] < v[i]$.

Algorithm 22: MSC2

Data: vetor v de tamanho n
Result: soma da maior subsequência crescente

```
1  resposta = 1;
2  a[0] = −∞;
3  for i=1 to n do
4  |  a[i] = ∞;
5  end
6  for i=1 to n do
7  |  ache maior j tal que a[j] < v[i];
8  |  msc[i] = j + 1;
9  |  a[j + 1] = min(a[j + 1], v[i]);
10 |  resposta = max(resposta, msc[i]);
11 end
12 return resposta;
```