Conceitos Auxiliares

- É possível reduzir o detalhamento da complexidade de um algoritmo através de absorções (**conjuntivas**) e máximos assintóticos (em ordem) (**disjuntivas**).

- A absorção de uma parte da complexidade por outra acontece quando temos uma função que cresce mais rapidamente que a outra de maneira assintótica.

- Para funções **f** e **g** de **N** em R_+ dizemos que **f** é absorvida por **g** sse **f** é O(**g**).

$$(\exists\, c \in \mathbb{R}_+)\ (\exists\, n_0 \in \mathbb{N})\ (\forall\, n \geq n_0):\ f(n) \leq c \cdot g(n)$$

Princípio da absorção

- Para funções **f** e **g** de $\mathbb{N}$ em $\mathbb{R}_+$, se **f** é absorvida por **g**, sua soma pontual:

  f + **g** é Θ(g).

- Assim, como $n \,.\, logn$ é absorvida por n^2, a soma ($n \,.\, logn + n^2$) é $\Theta(n^2)$.

Conceitos Auxiliares

Para cada um dos seguintes pares de funções f e g, verifique se uma absorve a outra

$$n^2, n \log n$$

$$10^3 n^2, 2^n$$

$$2^5 n, n^2$$

$$10^n n^2, n2^n$$

Suponha que f é absorvida por g. Mostre ou dê um contra-exemplo

a) g é $O(f)$

b) f é $\Omega(g)$

Conceitos Auxiliares

Para cada um dos seguintes pares de funções f e g, verifique se uma absorve a outra

$n^2, n \log n$	**c=1, n_0 =1**
$10^3 n^2, 2^n$	**c=10^3, n_0 =4**
$2^5 n, n^2$	**c=2^5, n_0 =1**
$10^n n^2, n 2^n$	**c=1, n_0 =1**

Suponha que f é absorvida por g. Mostre ou dê um contra-exemplo

a) g é $O(f)$

b) f é $\Omega(g)$

Conceitos Auxiliares

Mostre o principio da absorção

Para funções **f** e **g** de $\mathbb{N}$ em $\mathbb{R}_+$, se **f** é absorvida por **g**, sua soma pontual:

f + **g** é Θ(g).

Conceitos Auxiliares

Mostre o principio da absorção

Para funções **f** e **g** de $\mathbb{N}$ em $\mathbb{R}_+$, se **f** é absorvida por **g**, sua soma pontual:

f + **g** é Θ(g).

Definição: $f(n) + g(n)$ é $\Theta(g(n))$ sse

$$(\exists d, d' \in R_+)(\exists n \in N)(\forall n \geq n_0)(f(n) + g(n) \geq d.g(d) \wedge f(n) + g(n) \leq d'.g(d)$$

f é absorvida por g , isto é $(\exists\, c \in \mathbb{R}_+)(\exists\, n_0 \in \mathbb{N})(\forall\, n \geq n_0)(f(n) \leq c \cdot g(n))$

Conceitos Auxiliares

Mostre o principio da absorção

Para funções **f** e **g** de $\mathbb{N}$ em $\mathbb{R}_+$, se **f** é absorvida por **g**, sua soma pontual:

f + **g** é Θ(g).

Definição: $f(n) + g(n)$ é $\Theta(g(n))$ sse

$$(\exists d, d' \in R_+)(\exists n \in N)(\forall n \geq n_0)(f(n) + g(n) \geq d.g(d) \wedge f(n) + g(n) \leq d'.g(d)$$

f é absorvida por g, isto é $(\exists\, c \in \mathbb{R}_+)(\exists\, n_0 \in \mathbb{N})(\forall\, n \geq n_0)(f(n) \leq c \cdot g(n))$

Sejam c, n_0 e $n \geq n_0$ tais constantes, então $f(n) \leq c \cdot g(n)$

$$\Rightarrow f(n) + g(n) \leq c \cdot g(n) + g(n) \leq (c+1)\, g(n)$$

Por outro lado $f(n) + g(n) \geq g(n)$. Então tome $d = 1$, $d' = c+1$ e $N = n_0$.

Conceitos Auxiliares

- Considere duas funções f e g de IN em IR_+.

- Freqüentemente, a partir de um certo ponto, uma função fica sempre maior que a outra. Portanto, se f é O(g), então Máx (f, g) também é O(g).
  Exemplo: como 2n + 3 é $O(n^2)$, então Máx $(2n + 3, n^2)$ é $O(n^2)$.

- Entretanto, em alguns casos esta dominância pode não existir (ex: quando as entradas são diferentes). Uma solução é utilizar para a ordem do máximo pontual Máx (f, g) a soma pontual, dada por

$$(f + g)(n) := f(n) + g(n).$$

- Outra sugestão é utilizar o máximo assintótico em ordem (MxAO).
  - Pior das duas funções para cada entrada pontual
  - Problema quando entradas são diferentes:
    - n e m^2 => $O(MxAO(n, m^2))$

Conceitos Auxiliares

O máximo assintótico deve ser comutativo.

(i): MxAO (f, g) = MxAO (g, f).

Devemos também requerer que as funções sejam dominadas.

(ii): f = O(MxAO (f, g)) e g = O(MxAO (f, g)).

Para ter uma cota menos folgada podemos exigir que o máximo assintótico seja a melhor cota superior.

(iii): MxAO (f, g) = O(h), sempre que f = O(h) e g = O(h).

Para tratar melhor o caso de dominância de uma das funções podemos Requerer também a seguinte propriedade.

(iv): MxAO (f, g) = g, sempre que $(\exists n_0 \in N)(\forall n \geq n_0)$: $f(n) \leq g(n)$.

Conceitos Auxiliares

Considere as propriedades (ii – as funções são dominadas) e (iii – é a melhor cota superior). Verifique quais delas são satisfeitas em cada caso abaixo

$$MxAO(n \cdot \log n \,,\, 2^n) = 2^n$$

$$MxAO(n \cdot \log n \,,\, n^2) = 2^n$$

$$MxAO(n \,,\, 2^n) = n^2$$

$$MxAO(n \,,\, n^2) = n \cdot \log n$$

$$MxAO(n \cdot \log n \,,\, n^2) = n$$

ACP - Atribuição

- Considere uma operação de atribuição $v \leftarrow e$

- A custo da atribuição é igual a

$$desemp[\, v \leftarrow e \,](d) = cálc[\, e \,](d) + transf[\, e(d) \,]$$

Sua complexidade pessimista tem as seguintes cotas

$$\text{Máx}\,(c_P\,[\, e \,],\ c_P\,[\, \leftarrow_e \,]\,) \leq c_P\,[\, v \leftarrow e \,] \leq c_P\,[\, e \,] + c_P\,[\, \leftarrow_e \,]$$

A ordem da complexidade pessimista da atribuição é

$$c_p[v \leftarrow e] = O(c_p[e] + c_p[\leftarrow_e])$$

Na maioria das vezes esta complexidade fica reduzida à avaliação da expressão **e**, logo

$$c_p[v \leftarrow e] = O(c_p[e])$$

ACP - Atribuição

- Considere o seguinte exemplo.

```
a) para variáveis inteiras
   a.1) i<-0 {inicialização}
   a.2) j<-i {tranferência}
```

A complexidade é igual a O(1)

normalmente consideramos o custo da atribuição de qualquer tipo de dado constante

```
b) para lista v de inteiros e variável inteira m
   m<-Max(v) //determinar maximo e tranferi-lo para m
```

A complexidade é igual a O(n)

```
c) para listas u,v, w
   c.1) u<-v //tranferência de elementos
```

A complexidade é igual O(n)

```
   c.2) w<-reversa(u) // inversão de lista e atribuição para w
```

A complexidade é igual n+n, i.e., O(n)

quando se atribui estruturas de dados, o custo depende do número de cópias

ACP - Atribuição

- Considere as listas u e v de inteiros e a seguinte operação de atribuição u$\leftarrow$ordene(v), onde **ordena** tem complexidade $O(n^2)$ e a atribuição tem complexidade $O(n)$. Qual é a complexidade resultante?

$$C_p[u \leftarrow ordene(v)] = O(C_p[ordene(v)] + C_p[\leftarrow_{ordena(v)}])$$

$$C_p[u \leftarrow ordene(v)] = O(n^2 + n)$$

$$C_p[u \leftarrow ordene(v)] = O(n^2)$$

Se não considerarmos a transferência de valores então, a complexidade de **v$\leftarrow$e** fica reduzida à avaliação da expressão **e**, ou seja, fica reduzida à complexidade da instrução de ordenação**.**

ACP - Seqüência

- O desempenho da seqüência S;T é a soma dos desempenhos de suas componentes

Para variáveis inteiras i e j:

$i \leftarrow 0$; $j \leftarrow i$.

Para lista v de inteiros e variável inteira m:

$m \leftarrow Max(v)$; $m \leftarrow m + 1$.

Para listas u, v e w:

$u \leftarrow v$; $w \leftarrow Reversa(v)$.

- A entrada pode mudar ao longo da seqüência….

ACP - Seqüência

- De maneira geral a execução da seqüência **S;T** tem sobre a entrada d, os esforços computacionais associados à

  Execução de **S** sobre a entrada **d;**

  Execução de **T** sobre a entrada **S(d).**

Assim o desempenho da seqüência S;T com entrada d é dado por

desemp[S ; T](d) = *desemp*[S](d) + *desemp*[T](S(d)).

ACP - Seqüência

- O desempenho da seqüência S;T é a soma dos desempenhos de suas componentes

Preservando assintoticamente o tamanho da entrada

- A complexidade pessimista da seqüência tem cotas

$$\max(c_p[S], c_p[T]) \leq c_p[S;T] \leq c_p[S] + c_p[T]$$

- A ordem da complexidade é

$$c_p[S;T] = O(c_p[S] + c_p[T])$$

ACP - Seqüência

- Dados dois algoritmos Prim(u) e Buscab(a,v), considere a seqüência: v←Prim(u) ; Buscab(a , v).
- Suponha que

  Prim(u) dá como saída a primeira metade da lista em u, com comprimento $\lfloor n / 2 \rfloor$, e tem complexidade $\Theta(n)$;

  Buscab(a , v) procura a na lista v, com complexidade $O(\log m)$, para lista v com comprimento m.

- Qual é a complexidade do algoritmo composto ?

ACP - Seqüência

$$c_P[\,v \leftarrow \mathrm{Prim}(u)] = O(\,c_P[\,\mathrm{Prim}(u)] + c_P[\,\leftarrow\,])$$

$$c_P[\,\leftarrow\,] = O(1) \text{ e } c_P[\,\mathrm{Prim}(u)] = O(n)$$

$$\Rightarrow c_P[\,v \leftarrow \mathrm{Prim}(u)] = O(n)$$

$$c_P[\,\mathrm{Buscab}](\frac{n}{2}) = O(\log\frac{n}{2}), \text{ logo}$$

$$c_P[\,v \leftarrow \mathrm{Prim}(u) + \mathrm{Busca}(a, v)] = O(n) + O(\log\frac{n}{2}) = O(n)$$

ACP - Seqüência

- A complexidade pessimista da seqüência tem cotas

$$\max(c_p[S](n), c_p[T](s(n))) \leq c_p[S;T](n) \leq c_p[S](n) + c_p[T](s(n))$$

- Onde $s(n) = \max\{tam(S(d)) | tam(d) \leq n\}$

- A ordem da complexidade é

$$c_p[S;T](n) = O(c_p[S](n) + c_p[T](s(n)))$$

ACP - Seqüência

- Considere o seguinte exemplo.

  a) para lista v de inteiros e variável inteira m

  m←Max(v); //determinar maximo e tranferi-lo para m

  m←m+1;

  A complexidade tem ordem n+1 : O(n)

  b) para listas u,v, w

  u←v //tranferencia

  w←reversa(u) // inversão de lista e atribuicao para w

  A complexidade tem ordem n+n+n: O(n)

ACP - Seqüência

- Considere o seguinte exemplo.

  v← Prim(u) ; w← Fin(u) ;

  V←Ordene(v) ; w← Ordene(w) ;

  u ←Concat(v , w)

  Suponha que

  - Prim (vista anteriormente) dá como saída a primeira metade da lista **u; e** Fin dá a segunda metade de sua entrada. Ambas tem complexidade linear

  - Ordene tem complexidade quadrática;

  - Concat(v , w) dá como saída a concatenação das listas v e w, com complexidade O(p + q), onde p e q são os tamanhos de v e w.

  Qual é a complexidade do algoritmo ?

ACP - Seqüência

- Considere o seguinte exemplo.

  v← Prim(u) ; w← Fin(u) ;

  V←Ordene(v) ; w← Ordene(w) ;

  u ←Concat(v , w)

  Suponha que

  - Prim (vista anteriormente) dá como saída a primeira metade da lista **u; e** Fin dá a segunda metade de sua entrada. Ambas tem complexidade linear

  - Ordene tem complexidade quadrática;

  - Concat(v , w) dá como saída a concatenação das listas v e w, com complexidade O(p + q), onde p e q são os tamanhos de v e w.

  Qual é a complexidade do algoritmo ?

$$n + n + \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2} + \frac{n}{2}\right) = O(n^2)$$

ACP - Condicionais

- A estrutura condicional pode assumir diversas formas, as mais usuais são :
  - se **b** então **S**
  - se **b** então **S** senão **T**

- Exemplo:

  Para variável inteira I:

  Se I=0 então I←I+1

  Avaliar a condição : complexidade $\Theta(1)$

  Executar a atribuição : complexidade $\Theta(1)$

  No pior caso, a complexidade é igual a $\Theta(1)$

ACP - Condicionais

- Exemplo:

  Para variável inteira m:

  Se m=0 então m←max(v)

  Avaliar a condição : complexidade $\Theta(1)$

  Executar a atribuição : complexidade $O(n)$

  No pior caso, a complexidade é igual a $O(n)$

ACP - Condicionais

- O desempenho desemp[se b então S](d) da estrutura condicional **se b então S**, com entrada d é dado por:

  - aval [b](d) + desemp[S](d) caso o valor de b em d seja verdadeiro;
  - aval [b](d) caso o valor de b em d seja falso.

ACP - Condicionais

- A complexidade pessimista de **se b então S** é limitada da seguinte maneira

$$c_P\,[\,b\,] \leq c_P\,[\,\underline{\text{se}}\ b\ \underline{\text{então}}\ S\,] \leq c_P\,[\,b\,] + c_P\,[\,S\,].$$

- A ordem da complexidade é $c_P\,[\,\underline{\text{se}}\ b\ \underline{\text{então}}\ S\,] = O(\,c_P\,[\,b\,] + c_P\,[\,S\,]\,).$

ACP - Condicionais

Qual é a complexidade do seguinte trecho de algoritmo ?

se Max(v)=0 então v←Ordene(v)

Considerando

c_P [Max(v) = 0] = O(n) e c_P [v ← Ordene(v)] = O(n^2);

A complexidade é dada por

c_P [<u>se</u> Max(v) = 0 <u>então</u> v ← Ordene(v)] =
= O(c_P [Max(v) = 0] + c_P [v ← Ordene(v)]).

Portanto

$$c_p[\text{se Max(v)=0 então v} \leftarrow \text{Ordena(v)}] = O(n + n^2) = O(n^2)$$

ACP - Condicionais

- Considere a condição mais geral **se b então S senão T**, os esforços para uma entrada d são iguais a
  - aval[b](d)+desempenho[S](d), caso b seja verdadeiro,
  - aval[b](d)+desempenho[T](d), caso b seja falso.

Exemplo:

Para variáveis inteiras i e j:

se $i \neq j$ então $i \leftarrow i + j$ senão $j \leftarrow i + 1$.

Esta estrutura condicional envolve:

- saber se os valores de i e j são diferentes, tem complexidade $\Theta(1)$;

- se sim, executar a atribuição $i \leftarrow i + j$, com complexidade $\Theta(1)$;

- se não, executar a atribuição $j \leftarrow i + 1$, com complexidade $\Theta(1)$.

A complexidade no pior caso é $\Theta(1)$

ACP - Condicionais

Exemplo:

Para listas u e v (de inteiros):

se **u = v** então **v ← Prim(u)** senão **u ← Ordene(v).**

Esta estrutura condicional envolve:

- determinar se as listas u e v são iguais, com complexidade O(n);
- se sim, executar a atribuição v ← Prim(u), com complexidade O(n);
- se não, executar a atribuição u ←Ordene(v), com complexidade $O(n^2)$.

Qual é a complexidade no pior caso ?

ACP - Condicionais

Exemplo:

Para listas u e v (de inteiros):

se **u = v** então **v ← Prim(u)** senão **u ← Ordene(v).**

Esta estrutura condicional envolve:

- determinar se as listas u e v são iguais, com complexidade O(n);
- se sim, executar a atribuição v ← Prim(u), com complexidade O(n);
- se não, executar a atribuição u ←Ordene(v), com complexidade $O(n^2)$.

Qual é a complexidade no pior caso ?

A complexidade no pior caso é $O(n^2)$

ACP - Condicionais

A complexidade pessimista de **se b então S senão T** tem as cotas:

Inferior (segundo o livro) $c_p[b] \leq c_p[\text{se b então S senão T fim-se}]$

Esta certo?

Superior: $c_p[\text{se b então S senão T fim-se}] \leq c_p[b] + MxAO(c_p[S], c_p[T])$

A ordem de complexidade é igual a

c_P [<u>se</u> b <u>então</u> S <u>senão</u> T] = O(c_P [b] + MxAO (c_P [S], c_P [T])

ACP - Condicionais

- Considere o exemplo

se Max(v) $\geq$ 0 **então** v $\leftarrow$ Reversa(v) **senão** v $\leftarrow$ Ordene(v).

Com $c_p[\max(v) \geq 0] = O(n)$, $c_p[v \leftarrow \text{reversa}(v)] = O(n)$, $c_p[v \leftarrow \text{ordene}(v)] = O(n^2)$.

Qual é a complexidade no pior caso ?

ACP - Condicionais

- Considere o exemplo

$\underline{\text{se}}$ Max(v) $\geq$ 0 $\underline{\text{então}}$ v $\leftarrow$ Reversa(v) $\underline{\text{senão}}$ v $\leftarrow$ Ordene(v).

Com c_p[max(v)$\geq$ 0]=O(n), c_p[v$\leftarrow$ reversa(v)]=O(n) , c_p[v$\leftarrow$ordene(v)]=O(n^2).

Qual é a complexidade no pior caso ?

c_P [$\underline{\text{se}}$ Max(v) $\geq$ 0 $\underline{\text{então}}$ v $\leftarrow$ Reversa(v) $\underline{\text{senão}}$ v $\leftarrow$ Ordene(v)] =
= O(c_P [Max(v) $\geq$ 0] + MxAO(c_P [v $\leftarrow$ Reversa(v)], c_P [v $\leftarrow$ Ordene(v)])).

MxAO (c_P [v $\leftarrow$ Reversa(v)], c_P [v $\leftarrow$ Ordene(v)]) =
= MxAO (n, n^2) = O(n^2).

Logo,

c_p[$\underline{\text{se}}$ Max(v)$\leftarrow$0 $\underline{\text{então}}$ v$\leftarrow$reversa(v) $\underline{\text{senão}}$ v$\leftarrow$ ordene(v)]=
O(n+n^2)=O(n^2).