Módulo 7

Complexidade de Algoritmos

Algoritmos e Estruturas de Dados II

C++

Rone Ilídio

Tempo de execução de um programa

- Qual algoritm executará mais rápido?

```
int main(){
   SYSTEMTIME inicio,fim;
   GetSystemTime(&inicio);
   long aux = 1;
   for(long i=1; i<=10000000; i++)
      aux++;
   GetSystemTime(&fim);
   cout << "Hour: "<<inicio.wHour<<" Min: "<<inicio.wMinute<<" Second: "<<
            inicio.wSecond<<" Milliseconds: "<<inicio.wMilliseconds;
   cout << "\nHour: "<<fim.wHour<<" Min: "<<fim.wMinute<<" Second: "<<
           fim.wSecond<<" Milliseconds: "<<fim.wMilliseconds;
   getch();
}
```

```
Hour: 22 Min: 40 Second: 15 Milliseconds: 906
Hour: 22 Min: 40 Second: 15 Milliseconds: 953
```

Tempo de execução de um programa

- Qual algoritm executará mais rápido?

```
#include <conio.h>
#include <iostream.h>
#include <Windows.h>
#include <time.h>
int main(){
   SYSTEMTIME inicio,fim;
   GetSystemTime(&inicio);
   long aux = 1;
   for(long i=1; i<=100000; i++)
      for(long i=1; i<=100000; i++)
         aux++;
   GetSystemTime(&fim);
   cout << "Hour: "<<inicio.wHour<<" Min: "<<inicio.wMinute<<" Second: "<<inicio.wSecond
        <<" Milliseconds: "<<inicio.wMilliseconds;
   cout << "\nHour: "<<fim.wHour<<" Min: "<<fim.wMinute<<" Second: "<<fim.wSecond
         <<" Milliseconds: "<<fim.wMilliseconds;
   getch();
}
```

```
Hour: 22 Min: 29 Second: 32 Milliseconds: 546
Hour: 22 Min: 30 Second: 12 Milliseconds: 453
```

Tempo de execução de um programa

- Estimar o tempo → considere cada operação um ciclo

```
int main(){
    long aux = 1, n;
    cout << "Digite um numero:";
    cin >> n;
    for(long i=1; i<=n; i++)
      aux++;
    cout << "aux:"<<aux;
    getch();
}
```

2 ciclo

N *

1 ciclo

1 ciclo

2+N*1 + 1=3+N

Tempo de execução de um programa

- Estimar o tempo

```
int main(){
  long aux = 0, n;                          → 2 ciclos
  cout << "Digite um numero:";
  cin >> n;
  for(long a=1; a<=n; a++)                  → N *
     for(long b=1; b<=n; b++)               →     N *
        aux++;                              →         1 ciclo
  cout << "aux:"<<aux;                      → 1 ciclo
  getch();
}
```

$2+n*n*(1)+1=3 + n^2$

Tempo de execução de um programa

- O tempo de execução é dado em relação a uma função, denominada função de complexidade f(n), onde n é o número de entradas do algoritmo
- Para os exemplos anteriores
  - f(n) = 3 + n ou somente f(n) = n
  - f(n) = 3 + n^2 ou somente f(n) = n^2
- Analise o próximo exemplo → a partir de um vetor ordenado (crescente) exibir a posição do primeiro elemento maior que x;

Tempo de execução de um programa

```
int main(){
    int const n=100;                              1 ciclo
    int v[n];                                     1 ciclo
    //V recebe valores em ordem crescente         1 ciclo
    int x = 21;                                   1 ciclo
    for(int a=0; a<N; a++){
        if(v[a]>x){
            cout << "Posicao:"<<a;                indefinido
            break;
        }
    }
    getch();
}
```

Tempo de execução de um programa

- Qual é o tempo de execução do programa anterior?
- Definições:
  - Melhor caso: quanto o programa executa o mais rápido possível
  - Pior caso: quando demora o máximo possível para executar
  - Caso médio: média (normalmente)

Tempo de execução de um programa

- Análise do exemplo
  - Melhor caso = 4 + 1
  - Pior caso = 4 + n
  - Caso médio 4 + n /2
- Observação 1: para valores constantes, utiliza-se somente 1, ou seja:
  - Melhor caso: f(n) = 1
  - Pior caso: f(n) = n
  - Caso médio: f(n) = n /2
- Observação 2: o caso médio deve ser a média do tempo de execução para todos os valores de n

Tempo de execução de um programa

- Utilidade
  - Ajuda na escolha de algoritmos que resolvem o mesmo problema
  - Auxilia na análise da possibilidade de implementação de um algoritmo (imagine um algoritmo $f(n) = n^n$, para $n = 1000$;

Importante

- O tempo real de execução depende de vários fatores:
  - Compilador
  - SO
  - Outros programas em execução
  - Tempo de acesso aos dados na memória
  - Número de processadores utilizados
  - Entre outros
- Na análise de um algoritmo, normalmente considera-se as funções mais importantes. Ex: ordenação considera-se somente as comparações

Complexidade de Espaço

- Complexidade de tempo → exemplos anteriores
- Complexidade de espaço → quantidade de memória utilizada
- Veja exemplo

```
#include<conio.h>
#include<iostream.h>
int main(){
    int n, *vet;
    cout <<"Tamanho do vetor:";
    cin >> n;
    vet = new int[n];
    for(int i=0; i<n; i++){
        cout << "\nNumero:";
        cin >> vet[i];
    }
    for(int i=0; i<n; i++){
        cout << "\n"<< i << ": " << vet[i];
    }
    getch();
}
```

Complexidade de Memória

- No exemplo anterior, n define o tamanha da memória a ser utilizada
- Função de complexidade de espaço f(n) = n

Análise da Complexidade

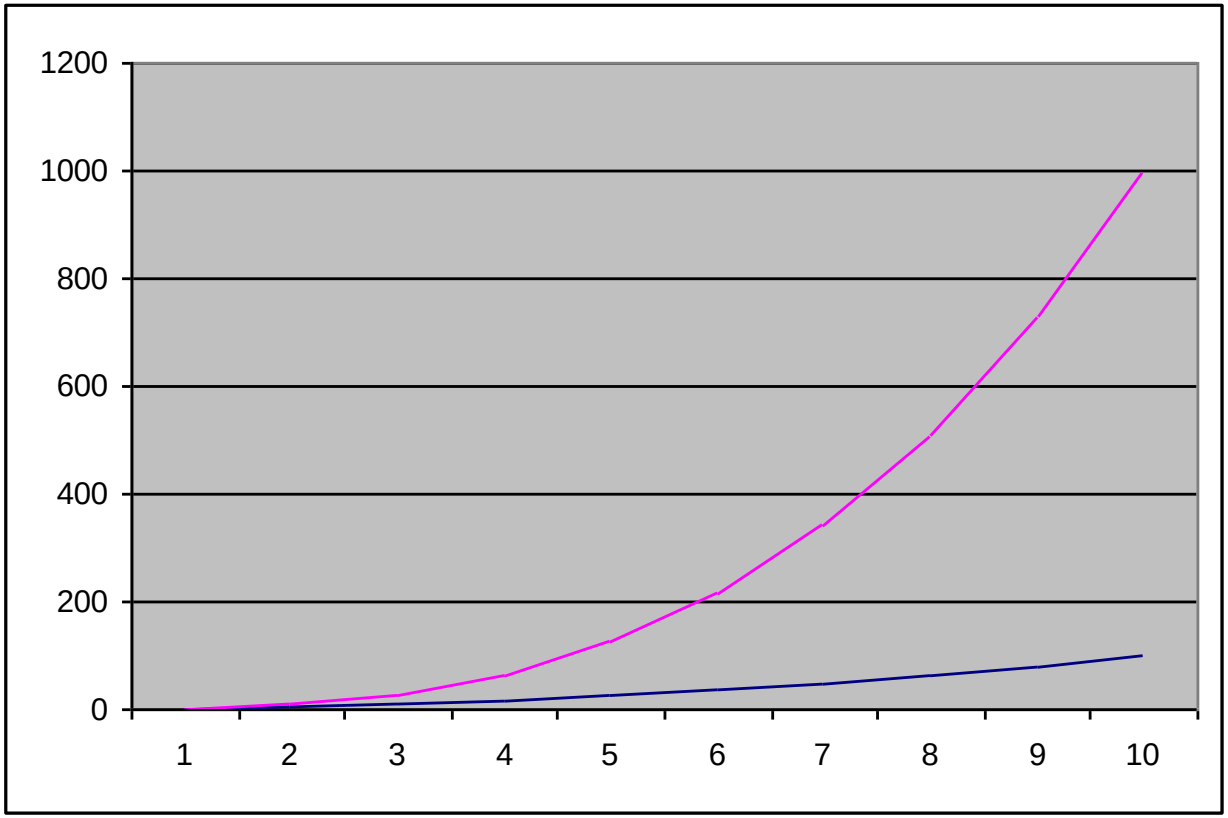
- Análise de um algoritmo particular
- Análise de um classe de algoritmos que resolvem o mesmo problema
- Algoritmo Ótimo → algoritmo que resolve um problema com a menor complexidade possível

Comportamento Assintótico de Funções

- Considere complexidade de tempo
- Considere valores grandes de n
- Comportamento assintótico de uma função é o comportamento da função para a variação de n
- Ex: sejam duas funções de complexidade $f(n) = n^2$ e $g(n) = n^3$, veja gráfico

Comportamento Assintótico da Funções

n	n*n	n*n*n
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

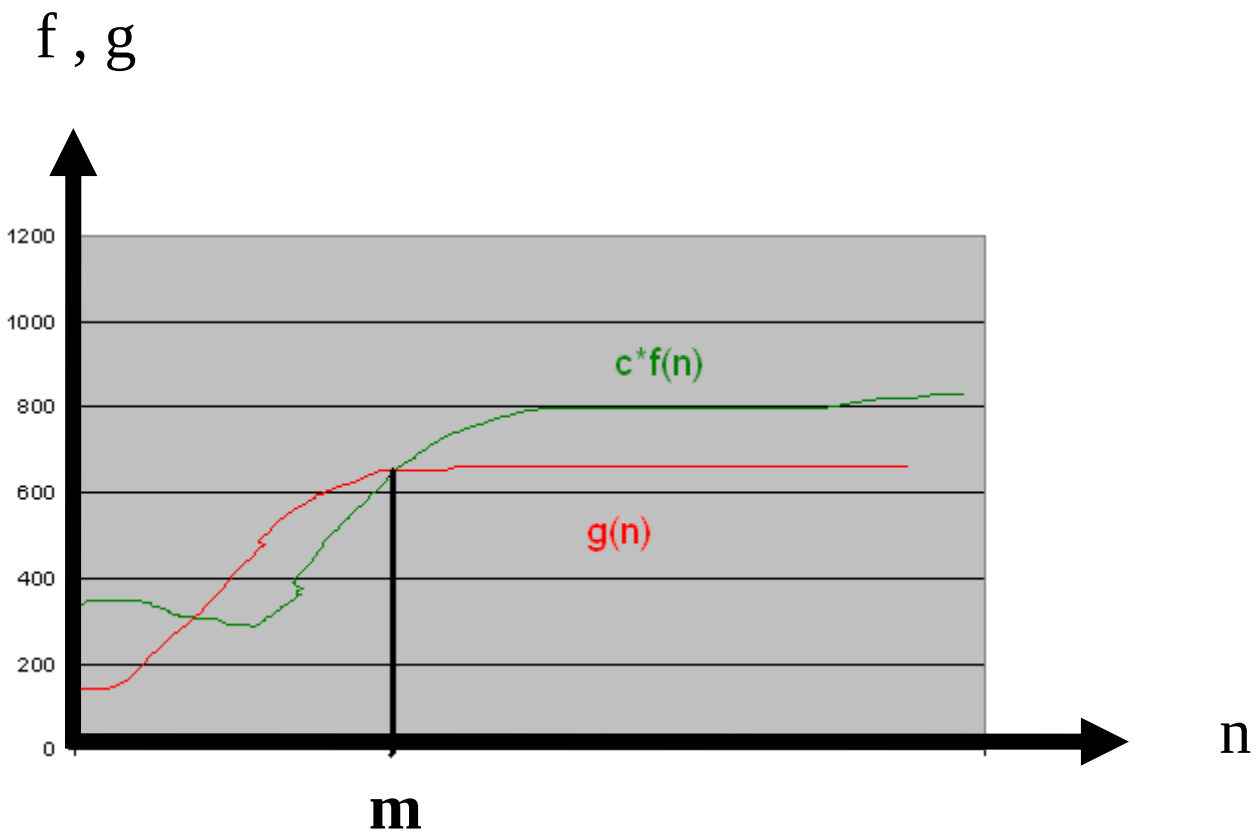


Comportamento Assintótico da Funções

- Definição

  Uma função f(n) é domina assintoticamente outra função g(n) se existem duas constantes positivas c e m tais que, para n>=m, temos | g(n) | < = c * | f(n) |

Comportamento Assintótico da Funções

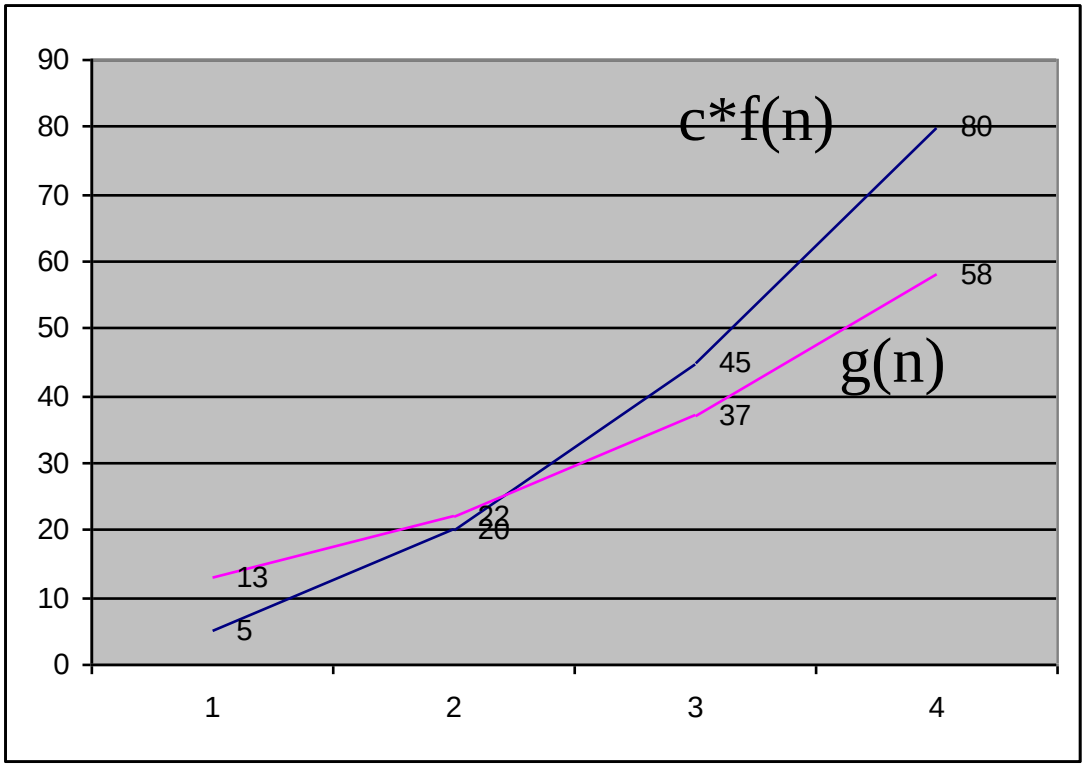


Comportamento Assintótico da Funções

- Exemplo 1: $g(n) = n^3$ e $f(n) = 3*n^2 + 10$
  - Para c = 5 e n = 3 temos o seguinte gráfico

Ajuda

n	g	f
1	5	13
2	20	22
3	45	37
4	80	58
5	125	85



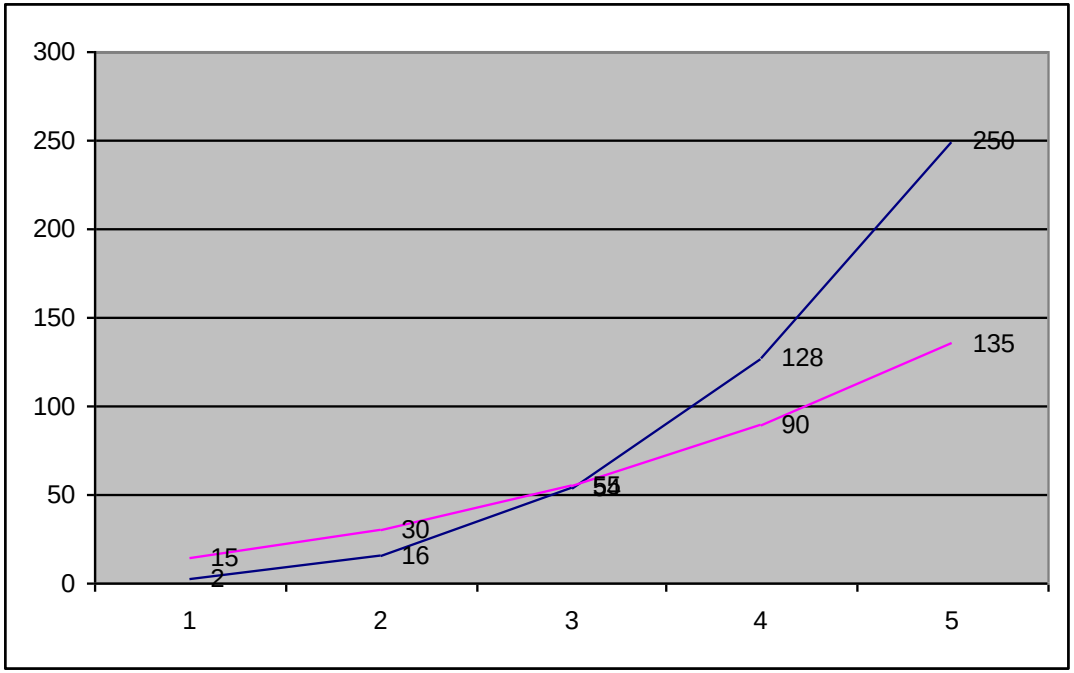
f domina g

Comportamento Assintótico da Funções

- Exemplo 2: $g(n) = n^2$ e $f(n) = 5*n^2 + 15$
  - Para c = 2 e m = 4 temos o seguinte gráfico

Ajuda

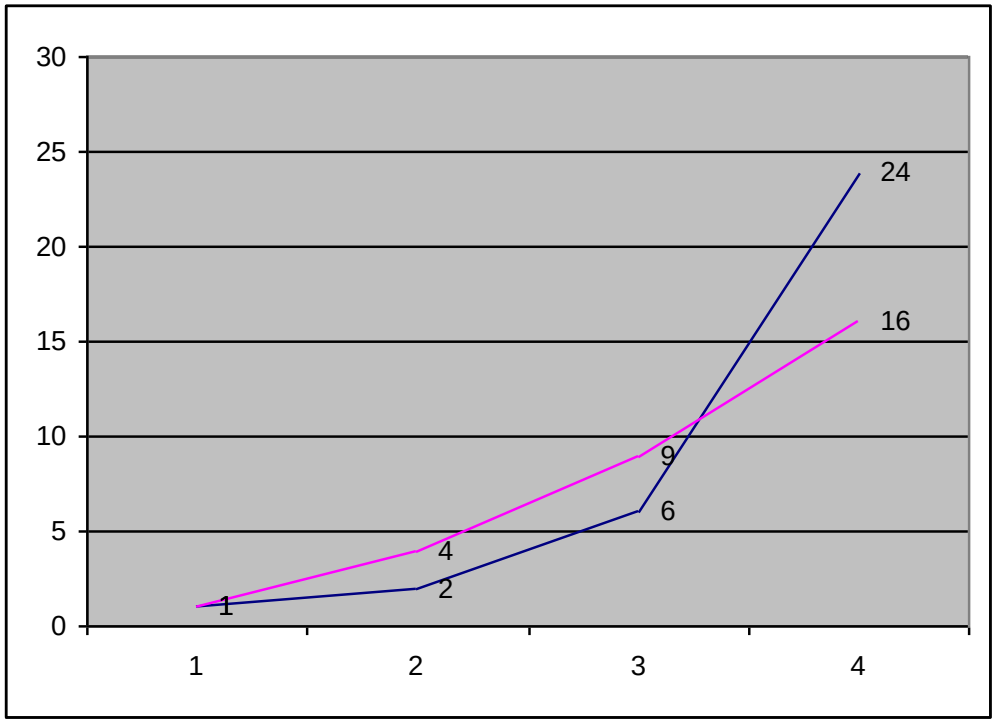
	g(n)=2 n^3	f(n)=5 n^2+15
1	2	20
2	16	35
3	54	60
4	128	95
5	250	140



g(n) e f(n) dominam uma a outra

Comportamento Assintótico da Funções

- Exemplo 3: g(n) = fatorial(n) e f(n) = n^2
  - Para c = 1, m=4 temos o seguinte gráfico



Definição da Notação O

- Uma função g(n) é O (f(n)) se f(n) domina g(n)

- Exemplo: $f(n) = (n+1)^2$, logo f(n) é O (n^2), pois temos m=1 e c=4. Ou seja, n^2 domina assintoticamente f(n)

Definição da Notação O

- Exemplo: $g(n) = 3n^3+2n^2+n$ é $O(n^3)$, pois temos $m = 1$ e $c = 6$

- Exemplo: $3n^3+2n^2+n$ é $O(n^4)$, pois temos $m = 1$ e $c = 6$

Classes de Comportamento Assintótico

- Se f é a função de complexidade para um algoritmo F, então O(f) é o comportamento assintótico ou complexidade assintótica de F
- Funções com a mesma complexidade pertencem à mesma classe de problema
- Veja as classes de problemas

Classes de Comportamento Assintótico

- $f(n) = O(1)$: complexidade constante
- $f(n) = O(\log n)$: complexidade logarítmica
- $f(n) = O(n)$: complexidade linear
- $f(n) = O(n \log n)$: não possui denominação
- $f(n) = O(n^2)$: complexidade quadrática
- $f(n) = O(n^3)$: complexidade cúbica
- $f(n) = O(2^n)$: complexidade exponencial
- $f(n) = O(n!)$: complexidade fatorial

Classes de Comportamento Assintótico

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 seg.	0,00002 seg.	0,00003 seg.	0,00004 seg.	0,00005 seg.	0,00006 seg.
n^2	0,0001 seg.	0,0004 seg.	0,0009 seg.	0,0016 seg.	0,0025 seg.	0,0036 seg.
n^3	0,001 seg.	0,008 seg.	0,027 seg.	0,64 seg.	0,125 seg.	0,216 seg.
n^5	0,1 seg.	3,2 seg.	24,3 seg.	1,7 min.	5,2 min.	13 min.
2^n	0,001 seg.	1 seg.	17,9 min.	12,7 dias	35,7 anos	366 sec.
3^n	0,059 seg.	58 min.	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Exemplos de Utilização de Análise Assintótica - Problema

1) Seja um algoritmo ***A***, de complexidade de tempo $O(n^5)$, e um algoritmo ***B***, de complexidade de tempo $O(2^n)$. Ambos resolvem o mesmo problema. Eles foram executados para um dado valor de n. O algoritmo ***A*** foi mais lento que o algoritmo ***B***. No entanto, o algoritmo ***A*** é considerado pela literatura como algoritmo ótimo para o problema. A literatura está errada? Onde se encontra o erro? A literatura está correta? Por quê? Justifique sua resposta.

Exemplos de Utilização de Análise Assintótica - Solução

Algoritmo ótimo é aquele que resolve um dado problema com menor complexidade possível.

Se verificarmos o comportamento assintótico das funções, verifica-se que n^5 é $O(2^n)$ para c=1 e m=23

Provavelmente a execução considerou n com valores pequenos, menores de 23.