Informática
PUC-Rio

Linguagens Formais e Autômatos (LFA)

Aula de 25/11/2013

Complexidade (Custo) de Tempo/Espaço, Decidibilidade

Complexidade (Custo) de Tempo/Espaço em MT's

Complexidade de TEMPO (CT):

CT é o **número máximo de transições processadas por uma computação de MT** quando iniciada por uma cadeia de comprimento **n**, independentemente de a cadeia ser aceita ou não.

CT é expresso como uma função de n. Como se trata do número MÁXIMO de transições, trata-se sempre do **pior caso**. Nem todas as cadeias de mesmo comprimento requerem o mesmo número de transições para serem decididas..

Complexidade de ESPAÇO (CE):

CE é o **número** que expressa a **quantidade MÁXIMA de células** de MT utilizadas na computação de cadeias de entrada de comprimento **n para ler/gravar INFORMAÇÃO DE PROCESSAMENTO** (i.e. não são contabilizadas as células da fita utilizadas apenas para a entrada), independentemente de a cadeia ser aceita ou não. Trata-se também do **pior caso** e CE é expresso como uma função de n.

A complexidade de Tempo/Espaço na prática

Em termos práticos:

- Complexidade de tempo tem a ver com uso de CPU
  - Número de instruções elementares que um algoritmo tem de executar até terminar. A medida de complexidade de tempo é dada em função do tamanho ('n', por convenção) da entrada.
- Complexidade de espaço tem a ver com uso de Memória (RAM)
  - Número de objetos (de informação) elementares que um algoritmo tem de guardar para executar completamente. A medida de complexidade de espaço é dada em função do tamanho ('n', por convenção) da entrada.

Ambas as definições envolvem certas "idealizações". Por exemplo, não se trata de *tempo de relógio* e também ao falar de objetos elementares de informação, não se consideram as *diferenças de tamanho* que as representações internas destes objetos podem ter entre si.

Análise de Complexidade de Algoritmos

- A complexidade de tempo dos algoritmos é um ponto fundamental para a computação. Sendo "n" o tamanho da entrada, os algoritmos podem, por exemplo, exibir complexidade nas seguintes ordens (representadas por "O"):
  - Constante : $O(1)$
  - Logarítimica : $O(\ln(n))$
  - Linear : $O(n)$
  - Quadrática : $O(n^2)$
  - Cúbica : $O(n^3)$
  - Polinomial : $O(n^p)$ $p \in N$
  - Exponencial : $O(\exp(n))$

The following table describes integer sorting algorithms and other sorting algorithms that are not comparison sorts. As such, they are not limited by a $\Omega(n \log n)$ lower bound. Complexities below are in terms of n, the number of items to be sorted, k, the size of each key, and d, the digit size used by the implementation. Many of them are based on the assumption that the key size is large enough that all entries have unique key values, and hence that $n << 2^k$, where << means "much less than."

Non-comparison sorts

http://en.wikipedia.org/wiki/Sorting_algorithm

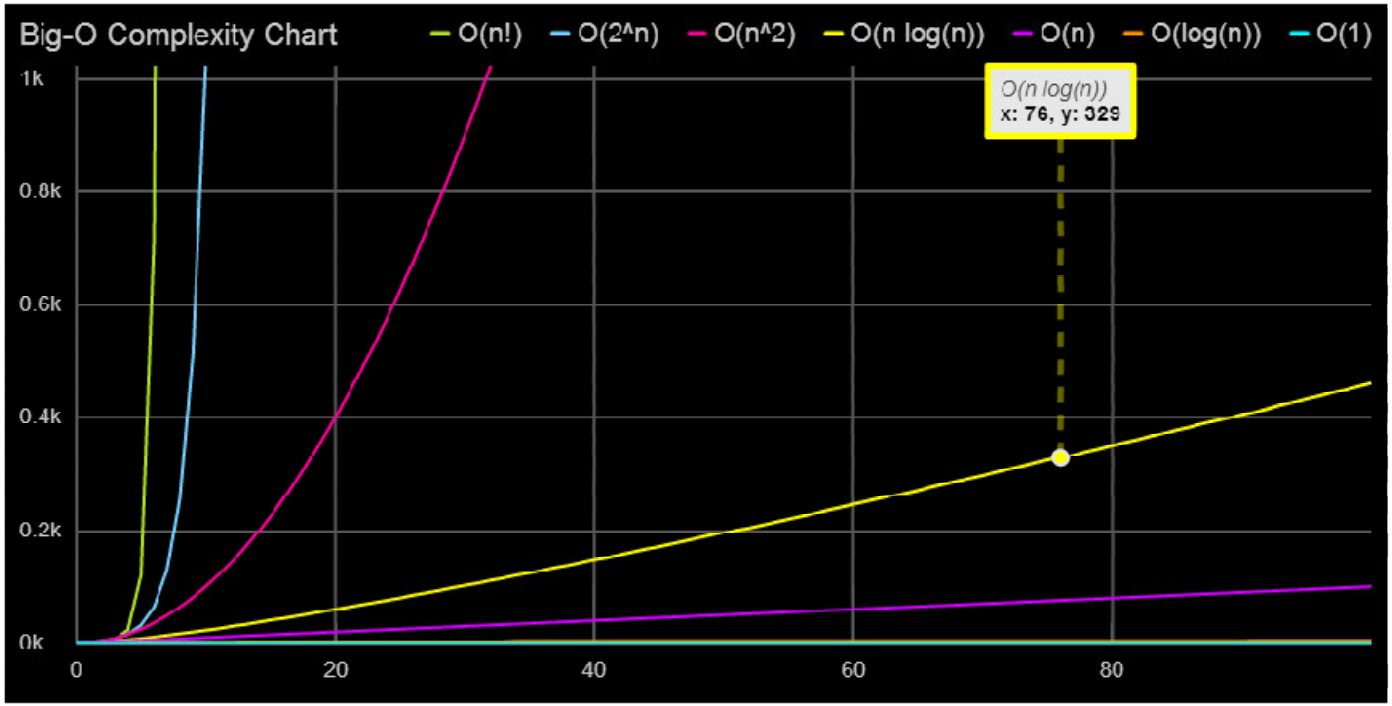
Name	Best	Average	Worst	Memory	Stable	$n << 2^k$	Notes
Pigeonhole sort	—	$n + 2^k$	$n + 2^k$	2^k	Yes	Yes	
Bucket sort (uniform keys)	—	$n + k$	$n^2 \cdot k$	$n \cdot k$	Yes	No	Assumes uniform distribution of elements from the domain in the array.[6]
Bucket sort (integer keys)	—	$n + r$	$n + r$	$n + r$	Yes	Yes	r is the range of numbers to be sorted. If r = $\mathcal{O}(n)$ then Avg RT = $\mathcal{O}(n)$[7]
Counting sort	—	$n + r$	$n + r$	$n + r$	Yes	Yes	r is the range of numbers to be sorted. If r = $\mathcal{O}(n)$ then Avg RT = $\mathcal{O}(n)$[6]
LSD Radix Sort	—	$n \cdot \frac{k}{d}$	$n \cdot \frac{k}{d}$	n	Yes	No	[6][7]
MSD Radix Sort	—	$n \cdot \frac{k}{d}$	$n \cdot \frac{k}{d}$	$n + \frac{k}{d} \cdot 2^d$	Yes	No	Stable version uses an external array of size n to hold all of the bins
MSD Radix Sort	—	$n \cdot \frac{k}{d}$	$n \cdot \frac{k}{d}$	$\frac{k}{d} \cdot 2^d$	No	No	In-Place. k / d recursion levels, 2^d for count array
Spreadsort	—	$n \cdot \frac{k}{d}$	$n \cdot \left(\frac{k}{s} + d\right)$	$\frac{k}{d} \cdot 2^d$	No	No	Asymptotics are based on the assumption that n << 2^k, but the algorithm does not require this.

The following table describes some sorting algorithms that are impractical for real-life use due to extremely poor performance or a requirement for specialized hardware.

Name	Best	Average	Worst	Memory	Stable	Comparison	Other notes
Bead sort	—	N/A	N/A	—	N/A	No	Requires specialized hardware
Simple							Count is number of flips.

Big-O Complexity Chart

This interactive chart, created by our friends over at MeteorCharts, shows the number of operations (y axis) required to obtain a result as the number of elements (x axis) increase. O(n!) is the worst complexity which requires 720 operations for just 6 elements, while O(1) is the best complexity, which only requires a constant number of operations for any number of elements.



http://bigocheatsheet.com/

Contributors

Decidibilidade

Ref: J.L.M. Rangel (1999) – Apostila de Linguagens Formais e Autômatos. Online em http://www.inf.puc-rio.br/~inf1626/Apostila/LFA-LivroRangel.pdf (Capítulo 8, seção 8.5)

"Para qualquer conjunto X **não recursivo**, a pergunta $x \in X$? não admite solução através de um algoritmo que aceite x como entrada e responda SIM ou NÃO corretamente à pergunta."

Mas, do ponto de vista prático, a diferença entre um conjunto não recursivamente enumerável, e um conjunto recursivamente enumerável que não é recursivo pode ser considerada pequena. Com efeito, suponha que um conjunto L é recursivamente enumerável mas não recursivo, e que dispomos de uma mT M que reconhece L, mas, não pára quando sua entrada não pertence a L. Suponha que, com entrada x, M foi executada por, digamos, mil passos, e que não parou. Nada podemos responder à pergunta "$x \in L$?". Será que M vai parar nos próximos mil passos?

O Problema da Parada

"M_i para com entrada x_j?"

Esquema geral da prova, por contradição, de que não existe uma MT que possa "decidir" esta pergunta para qualquer outra M_i com entrada x_j.

[1] Suponha que **EXISTE** um programa que computa

```
para?(programa, entrada) = Sim/Não
```

O Problema da Parada

"M_i para com entrada x_j?"

Esquema geral da prova, por contradição, de que não existe uma MT que possa "decidir" esta pergunta para qualquer outra M_i com entrada x_j.

[1] Suponha que **EXISTE** um programa que computa

```
para?(programa, entrada) = Sim/Não
```

[2] Agora suponha que Fulano escreveu um programa de uma linha

```
x(prog) = se para?(prog,prog)=Não retorna "F",
          se não entra em "loop".
```

O Problema da Parada

"M_i para com entrada x_j?"

Esquema geral da prova, por contradição, de que não existe uma MT que possa "decidir" esta pergunta para qualquer outra M_i com entrada x_j.

[1] Suponha que **EXISTE** um programa que computa

```
para?(programa, entrada) = Sim/Não
```

[2] Agora suponha que Fulano escreveu um programa de uma linha

```
x(prog) = se para?(prog,prog)=Não retorna "F",
          se não entra em "loop".
```

[3] Qual o resultado de `x(x)`?

O Problema da Parada

"M_i para com entrada x_j?"

Esquema geral da prova, por contradição, de que não existe uma MT que possa "decidir" esta pergunta para qualquer outra M_i com entrada x_j.

[1] Suponha que **EXISTE** um programa que computa

```
para?(programa, entrada) = Sim/Não
```

[2] Agora suponha que Fulano escreveu um programa de uma linha

```
x(prog) = se para?(prog,prog)=Não retorna "F",
          se não entra em "loop".
```

Contradição

[3] Qual o resultado de `x(x)`? Se x(x) para, entra em loop; se entra em loop, para.

O problema da parada (Youtube)

Conclusões Importantes até Aqui

Não há como decidir se um programa qualquer, com uma entrada qualquer, vai parar ou não.

A computação teórica oferece desafios práticos; problemas que são decidíveis podem ser impossivelmente "caros" para computadores reais.

- Desafios de software: Otimização
- Desafios de hardware: Produzir processadores mais rápidos

Os problemas de otimização de *tempo* são mais críticos do que os de otimização de espaço. (Até porque se fossem resolvidos, acesso a memória adicional/estendida deixaria de ser problema para um programa ter desempenho aceitável.)

Para entrega do trabalho

Forma:

- Email para a professora com:
  - Todos os programas, completos, em um único arquivo *.zip ou *.rar
  - Base de teste usada
  - Arquivo ASCII com instruções ou comentários (se necessário)

Prazo:

- 23:59 de 27/11/2013

Penalidade de até ⅓ dos pontos:

- Arquivos corrompidos
- Entregas com atraso de até 12 horas

Não serão aceitos arquivos de reposição ou entregas de trabalho depois de decorrids mais de 12 horas do prazo de entrega.