Estrutura de Dados II (ED2)

Aula 07 – Análise de Algoritmos (parte 2)

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (UFES)

(Material baseado nos slides do Professor Eduardo Zambon)

Teoria de Algoritmos

Teoria de algoritmos (complexidade)

Objetivos:

- Estabelecer a “dificuldade” de um problema.
- Desenvolver algoritmos “ótimos”.

Método:

- Suprimir detalhes na análise: analisar “com uma variação de um fator constante”.
- Eliminar variabilidade da entrada: focar no pior caso.

Limite superior (*upper bound*): garantia de desempenho do algoritmo para qualquer entrada.
Limite inferior (*lower bound*): prova de que nenhum algoritmo pode fazer melhor.
Algoritmo ótimo: *lower bound* = *upper bound* (variando de um valor constante).

Notações usuais na teoria de algoritmos

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ ⋮	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ ⋮	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$ ⋮	develop lower bounds

Teoria de algoritmos – Exemplo 1

Objetivos:

- Estabelecer a "dificuldade" de um problema e desenvolver algoritmos "ótimos".
- Ex.: 1-SUM = *"Existe um* 0 *no array?"*

Upper bound: Um algoritmo específico.

- Ex.: Algoritmo força-bruta para 1-SUM: testar todas as posições do *array*.
- Tempo de execução máximo: $O(N)$.

Lower bound: Provar que não é possível fazer melhor.

- Ex.: É necessário examinar todas as N posições. (Qualquer uma pode conter um 0.)
- Tempo de execução mínimo: $\Omega(N)$.

Algoritmo ótimo:

- *Lower bound* é igual a *upper bound* (salvo uma constante).
- Ex.: algoritmo força-bruta para 1-SUM é ótimo. Tempo de execução é $\Theta(N)$.

Teoria de algoritmos – Exemplo 2

Objetivos:

- Estabelecer a “dificuldade” de um problema e desenvolver algoritmos “ótimos”.
- Ex.: 3-SUM.

Upper bound: Um algoritmo específico.

- Ex.: Algoritmo força-bruta para 3-SUM.
- Tempo de execução máximo: $O(N^3)$.

Mas já vimos um algoritmo melhor!

Teoria de algoritmos – Exemplo 2

Objetivos:

- Estabelecer a “dificuldade” de um problema e desenvolver algoritmos “ótimos”.
- Ex.: 3-SUM.

Upper bound: Um algoritmo específico.

- Ex.: Algoritmo melhorado para 3-SUM.
- Tempo de execução máximo: $O(N^2 \log N)$.

Lower bound: Provar que não é possível fazer melhor.

- Ex.: É necessário examinar todas as N posições para resolver 3-SUM.
- Tempo de execução mínimo: $\Omega(N)$.

Problemas em aberto:

- Algoritmo ótimo para 3-SUM?
- Algoritmo sub-quadrático para 3-SUM?
- *Lower bound* quadrático para 3-SUM?

Metodologia de projeto de algoritmos

Início:

- Desenvolva um algoritmo. (E analise-o.)
- Prove um *lower bound*.

Há uma diferença?

- Diminua o *upper bound*: desenvolva um novo algoritmo.
- Suba o *lower bound*. (Mais difícil.)

1970: Década de ouro do projeto de algoritmos.

- *Upper bounds* para vários problemas importantes foram melhorados.
- Muitos algoritmos ótimos descobertos.

Alerta:

- Pessimismo demais focar só no pior caso?
- É necessário mais do que “com uma variação de um fator constante” para se prever o desempenho.

Notações usuais na teoria de algoritmos

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10\,N^2$	$10\,N^2$ $10\,N^2 + 22\,N \log N$ $10\,N^2 + 2\,N + 37$	provide approximate model
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2}\,N^2$ $10\,N^2$ $5\,N^2 + 22\,N \log N + 3N$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10\,N^2$ $100\,N$ $22\,N \log N + 3\,N$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2}\,N^2$ N^5 $N^3 + 22\,N \log N + 3\,N$	develop lower bounds

Erro comum: Interpretar *big-Oh* como um modelo aproximado.
Este curso: foco em modelos aproximados. Uso da notação $\sim$.

Memória

Consumo típico de memória de tipos em C

```c
typedef struct {
  int i;
  char c;    // Padding of 3 bytes.
  double d;
} Data;

int main() {
  printf("Byte size:\n");
  printf("CHAR   : %2lu\n", sizeof(char));
  printf("SHORT  : %2lu\n", sizeof(short));
  printf("INT    : %2lu\n", sizeof(int));
  printf("LINT   : %2lu\n", sizeof(long int));
  printf("FLOAT  : %2lu\n", sizeof(float));
  printf("DOUBLE : %2lu\n", sizeof(double));
  printf("LDOUBLE: %2lu\n", sizeof(long double));
  Data d, *p;
  printf("DATA   : %2lu\n", sizeof(d));
  printf("DATA*  : %2lu\n", sizeof(p));
}
```

```
Byte size:
CHAR   :  1
SHORT  :  2
INT    :  4
LINT   :  8
FLOAT  :  4
DOUBLE :  8
LDOUBLE: 16
DATA   : 16
DATA*  :  8
```

Em uma plataforma x86 64 bits com `gcc 8.1`.

Exemplo de análise de consumo de memória

Q: Quanto de memória o algoritmo de *weighted quick-union* utiliza em função de N? Use a notação $\sim$ para simplificar.

```
static int *id;
static int *sz;
static int N;

void UF_init(int size) {
  N = size;
  id = malloc(N * sizeof(int));
  sz = malloc(N * sizeof(int));
  for (int i = 0; i < N; i++) {
    id[i] = i;
    sz[i] = 1;
  }
}
```

A1: $4N + 4N + 4 = 8N + 4 \sim 8N$ bytes.

A2: Use ferramentas como o `valgrind`.

C é bem econômico com a memória. Por outro lado, em Java o *object overhead* é 16 bytes.