Probabilistic Approach of Algorithms Analysis and Randomized Algorithms

Ulises Tirado Zatarain [1]
(ulises.tirado@cimat.mx)

[1] Algorists Group

February, 2016

Outline

1. Introduction
   - Notation flashback
   - Definitions
2. Analysing average cases
   - How to...?
   - Example
   - For randomized algorithms

Notation Flashback

Given A an algoritm, let $n \in \mathbb{Z}^+$ be the size of it input and $T(n)$ it run time:

- Big $\mathcal{O}(\cdot)$: we say that $A \in \mathcal{O}(f(n))$ iff $\exists k \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$ such that $\forall n > n_0$ then $T(n) \leqslant kf(n)$.
    - $f(n)$ is an upper bound of $T(n)$.
- Big $\Omega(\cdot)$: we say that $A \in \Omega(f(n))$ iff $\exists k \in \mathbb{R}^+, n_0 \in \mathbb{Z}^+$ such that $\forall n > n_0$ then $T(n) \geqslant kf(n)$.
    - $f(n)$ is a lower bound of $T(n)$.
- Big $\Theta(\cdot)$: $A \in \Theta(f(n))$ iff $A \in \mathcal{O}(f(n))$ and $A \in \Omega(f(n))$.

Best, worst and average cases

- Best case: usually trivial.
- Worst case: usually trivial.
- Average case: usually **NOT** trivial.

Definitions

Randomized Algorithm

Is an algorithm that employs a degree of randomness as part of its logic. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits.

- We can use randomized algorithms to remove the worst case.
- Average case will be similar to the best case.

Example: Linear median

function `median`$(\boldsymbol{x} \in \mathbb{R}^n)$ **begin**

 Parameters: $k \in [n]$

 $p \leftarrow$ `random`(n);

 Split $\boldsymbol{x}$ into subarrays $\boldsymbol{l}$ and $\boldsymbol{g}$ by comparing each element to p-th elemnent. While we are at it, count the number K of elements going in to $\boldsymbol{l}$.

 if $K = k - 1$ **then:**

  return x_p;

 else if $K > k - 1$ **then:**

  return `median`$(\boldsymbol{l}, k)$;

 else:

  return `median`$(\boldsymbol{g}, k - K - 1)$;

 end

end

Deterministic vs Randomized

- Many times exists a big gap in between complexity of deterministic algorithms in the worst case and average case.
- The run time of many randomized algorithms can be infinite, specially if they have a bad design.
- The design of randomized algorithms isn't trivial as deterministic.
- Usually a randomized algorithms has a deterministic version.

How to...? Answer: probability

Given A an deterministic algorithm:

1. Chose a sample S and probability distribution p from which inputs are drawn, then the input $\mathbf{X} \in S$ is a random variable for the distribution.
2. For $x \in S$, let $t(x)$ be the time taken by A for input x.
3. Compute, as function of the "size" n of inputs:

$$T(n) = \mathbb{E}[t(x)] = \sum_{x \in S} [t(x)\, p(\mathbf{X} = x)]$$

which is the expected or average run time of A.

Example: quicksort

function $\texttt{quicksort}(x \in \mathbb{F}^n, l, h \in \mathbb{Z})$ **begin**

    if $l \geqslant h$ **then: return**;
    $p \leftarrow l$;
    $i \leftarrow l - 1$;
    $j \leftarrow h + 1$;
    while true do
        while $x_{i+1} < x_p$ **do** $i \leftarrow i + 1$;
        while $x_{j-1} > x_p$ **do** $j \leftarrow j - 1$;
        if $i \geqslant j$ **then: break**;
        $\texttt{swap}(x, i, j)$;
    end
    $\texttt{quicksort}(x, l, p)$;
    $\texttt{quicksort}(x, p + 1, h)$;

end

Example: quicksort

Taking a pivote and compare this with all elements:

$$c_n = n+1+\sum_{k=1}^{n}\left[\left(c_{k-1}+c_{n-k}\right)\frac{1}{n}\right]$$

$$c_n = n+1+\frac{2}{n}\sum_{k=1}^{n}c_{k-1}$$

Multiplying by n both sides:

$$nc_n = n(n+1)+2\sum_{k=1}^{n}c_{k-1} \tag{1}$$

By induction, we can do same for $n-1$:

$$(n-1)c_{n-1} = n(n-1)+2\sum_{k=1}^{n-1}c_{k-1} \tag{2}$$

Example: quicksort

Substracting equation (2) from equation (1):

$$\begin{aligned} nc_n - (n-1)c_{n-1} &= n(n+1) - n(n-1) + 2c_{n-1} \\ nc_n &= (n+1)c_{n-1} + 2n \end{aligned}$$

Dividing by $n(n+1)$:

$$\begin{aligned} \frac{c_n}{n+1} &= \frac{c_{n-1}}{n} + \frac{2}{n+1} \\ &= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\vdots \\ &= c_0 + c_1 + \sum_{k=2}^{n} \frac{2}{k+1} \end{aligned}$$

Example: quicksort

Since $c_0 = c_1 = 0$ then, we can aproximate by:

$$\begin{aligned}
\frac{c_n}{n+1} = 2\sum_{k=1}^{n-1}\frac{1}{k} &\approx 2\int_1^n \frac{1}{x}dx \\
&\approx 2\log n \\
c_n &\approx 2(n+1)\log n \\
&\approx 2(n\log n + \log n)
\end{aligned}$$

Finally, quicksort is $T(n) = \mathbb{E}[t(x)] \in \Theta(n\log n)$.

For randomized algorithms

Given R an randomized algorithm:

1. The input $\mathbf{X} \in \mathbf{I}$ is fixed, as usual, $\mathbf{I}$ is only the source space of the possible input, but the algorithm may draw (and use) random samples $Y \in \{y_1, y_2, \ldots\}$ from given sample space and probability distribution p.
2. For any $x \in \mathbf{I}$ and any $y \in S$, let $t(x,y)$ be the time taken by R on input $\mathbf{X} = x$ and $Y = y$ from S.
3. Compute, as function of the "size" n of inputs:

$$T(n) = \mathbb{E}[t(x,y)] = \max_{x \in \mathbf{I}} \sum_{y \in S} [t(x,y)\, p(Y = y | \mathbf{X} = x)]$$

since $\mathbf{X}$ and Y are independents and $\mathbf{X}$ is fixed:

$$T(n) = \mathbb{E}[t(x,y)] = \max_{x \in \mathbf{I}} \sum_{y \in S} [t(x,y)\, p(Y = y)]$$

Randomized quicksort

function $\texttt{quicksort}(x \in \mathbb{F}^n, l, h \in \mathbb{Z})$ **begin**
- **if** $l \geqslant h$ **then: return**;
- $p \leftarrow \texttt{random}(l, h)$;
- $i \leftarrow l - 1$;
- $j \leftarrow h + 1$;
- **while true do**
  - **while** $x_{i+1} < x_p$ **do** $i \leftarrow i + 1$;
  - **while** $x_{j-1} > x_p$ **do** $j \leftarrow j - 1$;
  - **if** $i \geqslant j$ **then: break**;
  - $\texttt{swap}(x, i, j)$;
- **end**
- $\texttt{quicksort}(x, l, p)$;
- $\texttt{quicksort}(x, p + 1, h)$;

end

References I

- Johan van Horebeek - Métodos Estocásticos en Computación
- Jean-Bernard Hayet - Programación Avanazada
- Robert Sedgewick - Algorithms in C++
- Wikipedia
- College of Computing - Georgia Tech
- Computer Science - Washington University