Runtime Complexity

Associative Arrays
   Introduction
   Practical Example
      Sorting
      Associative Array
Structure

Runtime Complexity

Associative Arrays
  Introduction
  Practical Example
  Sorting
  Associative Array
The runtime does not entirely depend on the size of the problem, but also on the type of input.

This results in:

- **Best runtime:**
  Lowest possible runtime complexity for an input of size $n$.

- **Worst runtime:**
  Highest possible runtime complexity for an input of size $n$.

- **Average / Expected runtime:**
  The average of all runtime complexities for an input of size $n$. 
Input: Array $a$ with $n$ elements
$a[i] \in \mathbb{N}, 1 \leq a[i] \leq n, 0 \leq i < n$

Output: Updated $a$ with $n$ elements where $a[0] \neq 1$

```
if $a[0] == 1$:
    $a[0] = 2$
else:
    for $i$ in range(0, $n$):
        $a[i] = 2$
```

Best runtime: $O(1) + O(1) = O(1)$
Worst runtime: $O(1) + O(n) = O(n)$
The average runtime is determined by the average runtime for all input instances of size $n$.

Every element of $a$ can have $n$ different values:

- $a[0] == 1$ in $n^{n-1}$ instances
- $a[0] != 1$ in $n^n - n^{n-1} = n^{n-1} \cdot (n - 1)$ instances

Sum of all runtime complexities:

$$
\underbrace{n^{n-1} \cdot O(1)}_{a[0] == 1} + \underbrace{n^{n-1} \cdot (n - 1) \cdot O(n)}_{a[0] != 1}
$$

Average runtime: (normalize by number of instances)

$$\frac{n^{n-1} + n^{n-1} \cdot (n - 1) \cdot n}{n^n} = \frac{1}{n} + n - 1 \in O(n)$$
Runtime Complexity
Example 2 - Binary Addition

- **Input**: binary number $b$ with $n$ digits
- **Output**: binary number $b + 1$ with $n$ digits
- **Runtime of the algorithm** is determined by the number of bits getting changed (steps)
  1. "0" $\rightarrow$ "1"
  2. "1" $\rightarrow$ "0"

- **Best runtime**: 1 step = $O(1)$
- **Worst runtime**: $n$ steps = $O(n)$

**Table**: Binary addition

<table>
<thead>
<tr>
<th>Digits ($n$)</th>
<th>Input</th>
<th>Output</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1011100100</td>
<td>1011100101</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>1100</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>11111111</td>
<td>000000000</td>
<td>8</td>
</tr>
</tbody>
</table>
### Example 2 - Average Steps

**Table: Binary addition with $n = 1$**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{steps} = \frac{1 + 1}{2} = 1
\]

**Table: Binary addition with $n = 2$**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{steps} = \frac{1 + 2 + 1 + 2}{4} = \frac{3}{2}
\]
Runtime Complexity
Example 2 - Average Steps

Table: Binary addition with $n = 3$

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>000</td>
<td>3</td>
</tr>
</tbody>
</table>

$$\text{steps} = \frac{1 + 2 + 1 + 3 + 1 + 2 + 1 + 3}{8} = \frac{7}{4}$$

$$= 2 - \frac{1}{4} = 2 - \frac{1}{2^{n-1}}$$

⇒ Average runtime:

$$2 - \frac{1}{2^{n-1}} \in \mathcal{O}(1)$$
### Runtime Complexity

#### Example 2 - Average Steps

**Table:** Case analysis for instances of size $n$

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Instances</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>____...0</td>
<td>____...1</td>
<td>$2^{n-1}$</td>
<td>1</td>
</tr>
<tr>
<td>____...01</td>
<td>____...10</td>
<td>$2^{n-2}$</td>
<td>2</td>
</tr>
<tr>
<td>____...011</td>
<td>____...100</td>
<td>$2^{n-3}$</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>_01...1111</td>
<td>_10...0000</td>
<td>$2^1$</td>
<td>n-1</td>
</tr>
<tr>
<td>011...1111</td>
<td>100...0000</td>
<td>$2^0$</td>
<td>n</td>
</tr>
<tr>
<td>111...1111</td>
<td>000...0000</td>
<td>1</td>
<td>n</td>
</tr>
</tbody>
</table>

**Average steps:**

$$
\frac{1 \cdot 2^{n-1} + 2 \cdot 2^{n-2} + \cdots + (n - 1) \cdot 2^1 + n \cdot 2^0 + n \cdot 1}{2^{n-1} + 2^{n-2} + \cdots + 2^1 + 2^0 + 1} = \binom{n}{1} \cdot \binom{2^{n-1}}{n-1} + n
$$
Runtime Complexity

Example 2 - Average Steps

- **Denominator:**
  \[
  \left( \sum_{i=0}^{n-1} 2^i \right) + 1 = \text{geometric series} = (2^n - 1) + 1 = 2^n
  \]

- **Counter:**
  \[
  \left( \sum_{i=1}^{n} i \cdot 2^{n-i} \right) + n^{[x=2x-x]} = \left( 2 \sum_{i=1}^{n} i \cdot 2^{n-i} \right) - \left( \sum_{i=1}^{n} i \cdot 2^{n-i} \right) + n
  \]
  \[
  = 1 \cdot 2^n + 2 \cdot 2^{n-1} + 3 \cdot n^{n-2} + \cdots + (n - 1) \cdot 2^2 + n \cdot 2^1
  \]
  \[
  -1 \cdot 2^{n-1} - 2 \cdot 2^{n-2} - \cdots - (n - 2) \cdot 2^2 - (n - 1) \cdot 2^1 - n \cdot 2^0 + n
  \]
  \[
  = \underbrace{2^n + 2^{n-1} + \cdots + 2^1 + 2^0} - 2^{0} = 2^{n+1} - 2
  \]
  
  \[
  2^{n+1} - 1
  \]
Average steps:

\[
\text{steps} = \frac{n \sum_{i=1}^{n} i \cdot 2^{n-i}}{\left( \sum_{i=0}^{n-1} 2^i \right) + 1} + n = \frac{2^{n+1} - 2}{2^n} = 2 - \frac{1}{2^{n-1}}
\]

\[
\lim_{n \to \infty} \left( 2 - \frac{1}{2^{n-1}} \right) = 2 \in O(1)
\]
Runtime Complexity

**Associative Arrays**

- Introduction
- Practical Example
  - Sorting
  - Associative Array
Normal array:

\[ A = [0 \Rightarrow A_0, \ 1 \Rightarrow A_1, \ 2 \Rightarrow A_2, \ 3 \Rightarrow A_3, \ldots] \]

- Searching elements by index
- Lookup of element with index "3":
Associative Arrays

Introduction

- In practice: all major programming project require associative arrays
- In our lecture: example of countries with associated information

Associative array:

\[
A = \begin{bmatrix}
"Europa" & \Rightarrow & A_0, 
"Amerika" & \Rightarrow & A_1, 
"Asien" & \Rightarrow & A_2, 
"Afrika" & \Rightarrow & A_3, 
\ldots
\end{bmatrix}
\]

- Searching elements by key
- The keys can be of any type with unique elements
- Lookup of element with key "Afrika":
  \[A["Afrika"] = A_3\]
Structure

Runtime Complexity

Associative Arrays
  Introduction
  Practical Example
    Sorting
    Associative Array
## Associative Arrays

### Practical Example

Table: Country data query from [http://geonames.org](http://geonames.org)

<table>
<thead>
<tr>
<th>ISO</th>
<th>ISO3</th>
<th>Country</th>
<th>Continent</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>AND</td>
<td>Andorra</td>
<td>EU</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>ARE</td>
<td>United Arab Emirates</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>AFG</td>
<td>Afghanistan</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td>ATG</td>
<td>Antigua and Barbuda</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>AIA</td>
<td>Anguilla</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>AL</td>
<td>ALB</td>
<td>Albania</td>
<td>EU</td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>ARM</td>
<td>Armenia</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>AO</td>
<td>AGO</td>
<td>Angola</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td>ATA</td>
<td>Antarctica</td>
<td>AN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task: How many countries belong to each continent?
- We are interested in column 3 (Country) and 4 (Continent)
- There are two typical ways to solve this:
  - Using sorting
  - Using an associative array
Structure

Runtime Complexity

Associative Arrays

Introduction

Practical Example

Sorting

Associative Array
Idea using sorting:

- We sort the table by Continent, so that all countries from the same continent are grouped in one block.
- We count the size of the blocks.

Disadvantage:

- Runtime of $\Theta(n \log n)$
- We have to iterate the array twice (sort and then count).

Advantage:

- Easy to implement (even with simple Linux / Unix commands).
Input:
- The data is saved as tab separated text (countryInfo.txt)
- Comments begin with a hash sign #

Commands:
- **grep**: Selects a specific set of lines (filter by ...)
  
  ```bash
grep -v '^[#]' countryInfo.txt
  
  -v: not
  
  ^#: # at start of line
  ```

- **cut**: Selects specific columns of each line (tab separated)
  
  ```bash
cut -f5,9
  
  -f5,9: columns 5 and 9 (= columns 3+4 of shown Table 17)
  ```
Associative Arrays

Practical Example - Sorting With Linux / Unix Commands

Commands:

- **sort**: Sorts lines by a key
  
  `sort -t ' ' -k2,2`
  
  `-t ' ': Separator: Tab (Insert with CTRL-V TAB)`
  
  `-k2,2: Key from column 2 to 2`

- **uniq**: Finds or counts unique keys

  `uniq -c`

  `-c: count occurrences of keys`

- **head**: Displays a provided number of lines

  `head -n30`

  `-n30: Displays the first 30 lines`

- **less**: Displays the file page wise
Sort countries by continent:

grep -v '^
' countryInfo.txt | cut -f5,9 \
  | sort -t ' ' -k2,2 | less

Table: Resulting data

<table>
<thead>
<tr>
<th>Algeria</th>
<th>AF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angola</td>
<td>AF</td>
</tr>
<tr>
<td>Benin</td>
<td>AF</td>
</tr>
<tr>
<td>Botswana</td>
<td>AF</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>AF</td>
</tr>
<tr>
<td>Burundi</td>
<td>AF</td>
</tr>
<tr>
<td>Cameroon</td>
<td>AF</td>
</tr>
<tr>
<td>Cape Verde</td>
<td>AF</td>
</tr>
</tbody>
</table>

Figure: Data pipeline

```
grep
  cut
  sort
  less
```
Count countries per continent:

grep -v '^-#' countryInfo.txt | cut -f9 \n| sort | uniq -c | sort -nr

Table: Resulting data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>AF</td>
</tr>
<tr>
<td>54</td>
<td>EU</td>
</tr>
<tr>
<td>52</td>
<td>AS</td>
</tr>
<tr>
<td>42</td>
<td>NA</td>
</tr>
<tr>
<td>27</td>
<td>OC</td>
</tr>
<tr>
<td>14</td>
<td>SA</td>
</tr>
<tr>
<td>5</td>
<td>AN</td>
</tr>
</tbody>
</table>

Figure: Data pipeline

grep → cut → sort → uniq → sort
Structure

Runtime Complexity

Associative Arrays

Introduction

Practical Example

Sorting

Associative Array
Idea using associative arrays:

- Take the continent as key
- Use a counter (occurrences) or a list with all countries associated with this continent as value

Advantage:

- Runtime $\mathcal{O}(n)$, implied we can find an element in $\mathcal{O}(1)$ as in normal arrays
Python:

```
# creates a new map (called dictionary)
countries = {
    "DE" : "Deutschland",
    "EN" : "England"
}

# check if element exists
if "EN" in countries:
    print("Found %s!" % countries["EN"])

# map key "DE" to value "Germany"
countries["DE"] = "Germany"

# delete key "DE"
del countries["DE"]
```
Efficiency:

- Depends on implementation
- Two typical implementations:
  - **Hashing**: Calculates a checksum of the key used as key of a normal array
    - search: $O(1) \ldots O(n)$
    - insert: $O(1) \ldots O(n)$
    - delete: $O(1) \ldots O(n)$
  - **(Binary-)Tree**: Creates a sorted (binary) tree
    - search: $O(\log n) \ldots O(n)$
    - insert: $O(\log n) \ldots O(n)$
    - delete: $O(\log n) \ldots O(n)$
### Table: Map implementations of programming languages

<table>
<thead>
<tr>
<th></th>
<th>Hashing</th>
<th>(Binary-)Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>all dictionaries</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>java.util.HashMap</td>
<td>java.util.TreeMap</td>
</tr>
<tr>
<td>C++11/14</td>
<td>std::unordered_map</td>
<td>std::map</td>
</tr>
<tr>
<td>C++98</td>
<td>__gnu_cxx::hash_map</td>
<td>std::map</td>
</tr>
</tbody>
</table>
Further Literature

General

Introduction to Algorithms. 

[MS08] Kurt Mehlhorn and Peter Sanders. 
Algorithms and data structures, 2008. 
Further Literature

**Map - Implementations / API**

- **[Java]** Java - HashMap
  
  [https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html](https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html)

- **[Java]** Java - TreeMap
  
  [https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html](https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html)

- **[Pyt]** Python - Dictionaries (Hash table)
  
  [https://docs.python.org/3/tutorial/datastructures.html#dictionaries](https://docs.python.org/3/tutorial/datastructures.html#dictionaries)
Further Literature

Map - Implementations / API

[Cppa] C++ - hash_map
http://www.sgi.com/tech/stl/hash_map.html

[Cppb] C++ - map
http://www.sgi.com/tech/stl/Map.html