My code is slow. What can I do about that?
My code is slow. What can I do about that?
About me

Primož Gabrijelčič

http://primoz.gabrijelcic.org

• programmer, MVP, writer, blogger, consultant, speaker
• Blog http://thedelphigeek.com
• Twitter @thedelphigeek
• Skype gabr42
• LinkedIn gabr42
• GitHub gabr42
• SO gabr
CompareValue for booleans

The CompareValue function is incredibly practical when you are writing comparers (functions that determine how some data structure is ordered). System.Math and System.StrUtils define a bunch of functions that can be used to compare integers, doubles, strings... There's, however, no CompareValue for booleans.

A CompareValue function compares two parameters, traditionally named `left` and `right`, and returns 0 if they are the same, -1 if `left` is smaller and 1 if `right` is smaller.

If we use the usual ordering of `false < true`, we can write the missing function as follows:

```pascal
function CompareValue(left, right: boolean): integer; overload;
begin
  if left < right then
    Result := -1
  else if left > right then
    Result := 1
  else
    Result := 0;
end;
```

Your task for today – if you choose to accept it – is: Write this function without any `if` statements.

Read more »
Books

Parallel Programming with OmniThreadLibrary

Delphi High Performance

Hands-On Design Patterns with Delphi

http://tiny.cc/pg-ppotl

http://tiny.cc/pg-dhp

http://tiny.cc/pg-dpd
PERFORMANCE
Performance

• What is performance?
• How do we “add it to the program”?  
  • There is no silver bullet!
What defines performance?

- Running “fast enough”
- Raw speed
- Responsiveness
  - Non-blocking
Improving performance

• Analyzing algorithms
• Measuring execution time
• Fixing algorithms
• Fine tuning the code
• Writing parallel code
Algorithm Complexity
Algorithm complexity

• O()
  • Tells us how algorithm slows down if data size is increased by a factor of $n$
  • $O(n)$, $O(n^2)$, $O(n \log n)$ ... 

• Time and space complexity
Frequently encountered $O()$

- $O(1)$ accessing array elements
- $O(\log n)$ searching in ordered list
- $O(n)$ linear search
- $O(n \log n)$ quick sort (average)
  introsort
- $O(n^2)$ quick sort (worst),
  naive sort (bubblesort, insertion, selection)
- $O(c^n)$ recursive Fibonacci,
  travelling salesman
## Comparing complexities

<table>
<thead>
<tr>
<th>Data size</th>
<th>O(1)</th>
<th>O(log n)</th>
<th>O(n)</th>
<th>O(n log n)</th>
<th>O(n²)</th>
<th>O(c^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Comparing complexities

<table>
<thead>
<tr>
<th>Data size</th>
<th>O(1)</th>
<th>O(log n)</th>
<th>O(n)</th>
<th>O(n log n)</th>
<th>O(n²)</th>
<th>O(c^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>43</td>
<td>100</td>
<td>512</td>
</tr>
</tbody>
</table>
## Comparing complexities

<table>
<thead>
<tr>
<th>Data size</th>
<th>O(1)</th>
<th>O(log n)</th>
<th>O(n)</th>
<th>O(n log n)</th>
<th>O(n²)</th>
<th>O(c^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>43</td>
<td>100</td>
<td>512</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>8</td>
<td>100</td>
<td>764</td>
<td>10.000</td>
<td>$10^{29}$</td>
</tr>
</tbody>
</table>
Comparing complexities

<table>
<thead>
<tr>
<th>Data size</th>
<th>O(1)</th>
<th>O(log n)</th>
<th>O(n)</th>
<th>O(n log n)</th>
<th>O(n^2)</th>
<th>O(c^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>43</td>
<td>100</td>
<td>512</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>8</td>
<td>100</td>
<td>764</td>
<td>10,000</td>
<td>10^{29}</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>9</td>
<td>300</td>
<td>2.769</td>
<td>90,000</td>
<td>10^{90}</td>
</tr>
</tbody>
</table>

DATA STRUCTURES

HTTP://BIGOCHEATSHEET.COM/
### Array

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>O(1)</td>
</tr>
<tr>
<td>Search</td>
<td>O(n)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(n)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
## Sorted array

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>O(1)</td>
</tr>
<tr>
<td>Search</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(n)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

- Bisection search
- `TArray.BinarySearch<T>`
TLList, TList<T>, TObjectList, TStringList

- Array / Sorted array
  - Faster when inserting/deleting
- TStringList.IndexOf, Find
- TList<T>.BinarySearch
TQueue<T> / TStack<T>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>O(1)</td>
</tr>
<tr>
<td>Search</td>
<td>O(n)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(1) / O(n)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

- Array-based
- Access via .List
- Clumsy Search
- Insertion may cause the storage to grow
Linked list

- Not CPU cache-friendly
- Adding order to other structures
- TGpDoublyLinkedList / GpLists.pas
  - https://github.com/gabr42/GpDelphiUnits
TDictionary<\texttt{K,V}>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>N/A</td>
</tr>
<tr>
<td>Search</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(1)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

- Relatively slow O(1)
- Increased memory consumption
  - “index” (hash) + data
- Unordered
TRedBlackTree\langle K, V \rangle [Spring4D]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Search</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>

- Worst complexity = average complexity
- Ordered
**Spring4D collections**

- List, SortedList
  - Array
- Stack, Queue
  - Array
- Dictionary
  - TDictionary
- SortedDictionary
  - TRedBlackTree
- Set
  - Dictionary
- OrderedSet
  - Dictionary + List
- MultiMap
  - Dictionary<K, List<V>>
- BidiDictionary
  - Dictionary<K,V> + Dictionary<V,K>
TIPS & TRICKS
• Accessing array element vs. accessing dictionary element
• ➔ Fine tuning (optimizing)
• Measure first!
If something is slow, do it less often

• Don’t update UI million times a second
• Caching!
TGpCache<K,V>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>N/A / O(1)</td>
</tr>
<tr>
<td>Search</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(1)</td>
</tr>
<tr>
<td>Deletion</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

• Ordered (by time)
• Accessing oldest/newest: O(1)
• Dictionary + Linked list
• TGpDoublyLinkedList / GpLists.pas  
  • https://github.com/gabr42/GpDelphiUnits
If something is slow, don’t do it

• BeginUpdate/EndUpdate
• UI virtualization
  • Virtual listbox
  • Virtual TreeView
LEARN MORE
Q&A