Tailored off the peg

Static Polymorphism in Software Library Design

Andreas Döring

11/2007
SeqAn - Sequence Analysis

www.seqan.de
SeqAn - Sequence Analysis

- Searching in long strings
- Quite simple concepts
- Main problem: Performance!
- C++
Tailored off the Peg

Speed up trick:

different implementations
for different cases

Andreas Döring: “Tailored off the peg” (2007)
specialization Hierarchy

special — fast

general — slow

Andreas Döring: “Tailored off the peg” (2007)
Select best variant at run time depending on the data
String matching algorithms:

- Alphabet size
- Pattern length

- Shift-Or
- Horspool
- BOM
- BNDM

[Navarro, Raffinot, "Flexible Pattern Matching in Strings" (2002)]

Andreas Döring: "Tailored off the peg" (2007)
Compile Time "Tailoring"

Select best variant

at compile time

depending on data types
Given:

1. Alphabet:
   \[ \Sigma = \{A,C,G,T\} \]

2. Sequence:
   \[ \text{A C G T G C C A T G C C A T C G T G C ...} \]

3. Shape:
   \[ \square \quad \square \quad \square \]
Gapped q-Gram Hash

\[ \Sigma = \{A,C,G,T\} \]

ACGTGCCATGCCATCGTGC...

Andreas Döring: "Tailored off the peg" (2007)
Gapped q-Gram Hash

To do:

Compute hash values for all positions!

Σ = \{A,C,G,T\}

0 1 2 3

\[ \text{hash value} = 28_{10} \]
Gapped q-Gram Hash

To do:

Compute hash values for all positions!

Application:

Fast scan for similar regions in two sequences
<table>
<thead>
<tr>
<th>Generic Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positions</td>
</tr>
<tr>
<td>hash()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>(loop)</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Implementations

**Ungapped Shape**

<table>
<thead>
<tr>
<th>Length</th>
<th>q</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hash()</strong></td>
<td>O(q)</td>
<td>1 multiplication</td>
<td>1 modulo</td>
<td>1 addition</td>
</tr>
<tr>
<td><strong>hashNext(prev)</strong></td>
<td>O(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example String:**

```
ACGTGCCATGCCATCGTG...
```

**Hash Values:**

```
0 1 2 3 4 2 4 1 4
```

```
2711018510
```
### Implementations

**Hardwired Ungapped Shape**

- hash()
- hashNext(prev)

---

*Andreas Döring: “Tailored off the peg” (2007)*
Hardwired Shape

hash()
{
    (loop unrolled)
}

Andreas Döring: "Tailored off the peg" (2007)
Polymorphism

hashAll(Sequence, Shape)

hashAll(Sequence, GenericShape)
hashAll(Sequence, HardwiredShape ⟨...⟩)
hashAll(Sequence, UngappedShape)
hashAll(Sequence, HardwiredUngappedShape ⟨...⟩)
Generic Algorithm

```
hashAll(Sequence, Shape)
{
    for position=0 to Sequence.length():
        Shape.hash(position)
}
```

```
hashAll(Sequence, Shape)
{
    Shape.hash(0)
    for position=1 to Sequence.length():
        Shape.hashNext(position)
}
```
Andreas Döring: "Tailored off the peg" (2007)
Dynamic Binding

Generic

hashAll()  
hash()

Hardwired

hash()

generic.hashAll(Sequence)
{
  for position=0 to Sequence.length():
    hash(position)
}

virtual is bad!

Andreas Döring: "Tailored off the peg" (2007)
Wish List

We need:
- Polymorphism
- Delegation (Inheritance)
- Spezialization (Overloading)
- Static Binding

Solution:
C++ Templates
<table>
<thead>
<tr>
<th>Template Subclassing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generic</strong></td>
</tr>
<tr>
<td>Shape ( \langle \rangle )</td>
</tr>
<tr>
<td><strong>Hardwired</strong></td>
</tr>
<tr>
<td>Shape ( \langle \text{Hardwired} \langle \ldots \rangle \rangle )</td>
</tr>
<tr>
<td><strong>Ungapped</strong></td>
</tr>
<tr>
<td>Shape ( \langle \text{Ungapped} \langle \rangle \rangle )</td>
</tr>
<tr>
<td><strong>HardwiredUngapped</strong></td>
</tr>
<tr>
<td>Shape ( \langle \text{Ungapped} \langle q \rangle \rangle )</td>
</tr>
</tbody>
</table>

Andreas Döring: "Tailored off the peg" (2007)
hashAll(seq, Shape ⟨ ? ⟩ sh)
{
    for position=0 to seq.length():
        hash(sh, position)
}

hashAll(seq, Shape ⟨ Ungapped ⟨ ? ⟩ ⟩ sh)
{
    hash(sh, 0)
    for position=1 to seq.length():
        hashNext(sh, position)
}
Template Subclassing

Subclasses specified by template arguments

Animal

Animal \langle \rangle

Mammal

Animal \langle \text{Mammal} \langle \rangle \rangle

Rodent

Animal \langle \text{Mammal} \langle \text{Rodent} \langle \rangle \rangle \rangle

Mouse

Animal \langle \text{Mammal} \langle \text{Rodent} \langle \text{Mouse} \rangle \rangle \rangle
All animals must eat:

\[
\text{eat}(\ Animal\ ⟨\ ?\ ⟩)
\]

Rodents eat greenstuff:

\[
\text{eat}(\ Animal\ ⟨\ Mammal\ ⟨\ Rodent\ ⟨\ ?\ ⟩⟩⟩)
\]

Pandas eat bamboo:

\[
\text{eat}(\ Animal\ ⟨\ Marsupials\ ⟨\ Panda\ ⟨\ ?\ ⟩⟩⟩)
\]

\[
\text{Animal\ ⟨\ Mammal\ ⟨\ Human\ ⟩⟩}
\]
All animals must eat:

\begin{verbatim}
    eat( Animal ⟨ ? ⟩ )
\end{verbatim}

Rodents eat greenstuff:

\begin{verbatim}
    ✔ eat( Animal ⟨ Mammal ⟨ Rodent ⟨ ? ⟩ ⟩ ⟩ )
\end{verbatim}

Koala bears eat eucalyptus:

\begin{verbatim}
    eat( Animal ⟨ Marsupial ⟨ Koala ⟨ ? ⟩ ⟩ )
\end{verbatim}

Animal ⟨ Mammal ⟨ Rodent ⟨ Mouse ⟩ ⟩}
We need:
- Polymorphism  ✔
- Delegation (Inheritance)  ✔
- Spezialization (Overloading)  ✔
- Static Binding  ✔
Conclusion

Message 1:
"Good libraries offer tailor-made suits of the peg."

Message 2:
"Never put off till run time what you can do at compile time."

Message 3:
"Use C++ templates to move work to compile time."
EOT
(end of talk)