Sequence Homology

M.M. Dalkilic, PhD
Monday, September 08, 2008
Class II
Indiana University, Bloomington, IN
Outline

- Sword of Damocles
- New Reading Posted on Website
- New Lab II / Homework Posted on Website
- Readings [Mount] Chap 3, [R] Chaps 3-4
- Most Important Aspect of Bioinformatics—homology search through sequence similarity
Outline (cont’d)

- Sequence Similarity
  - Most common Bioinformatics tool in use
  - One of the most misunderstood tools in use
  - Requires a great deal of background in ancillary disciplines—especially computation
Outline (cont’d)

• Computational Elements of Sequence Similarity
  o Algorithms
  o Complexity
  o Recursion & recurrence relations
  o Program strategies to reduce complexity of algorithms
    • Divide and Conquer
    • Dynamic Programming
How do we compare two “things”

Shape, Size, Color, Orientation...

Any kind of properties--best to think of chemical properties

Indel (Insertion/Deletion)

“representation of sequence of molecules”

Biochemical Example

Glutamate Neg. charged R-group

Aspartate Neg. Charged R-group

Sequence Similarity (Computation) M.M. Dalkilic, PhD
Sol Indiana University, Bloomington, IN 2008 ©
Computation

Algorithm

“process or rules for (esp. machine) calculations. The execution of an algorithm must not include any subjective decisions, nor must it require the use of intuition or creativity” [Brassard & Bratley]
Algorithm

When we set out to solve a problem, it is important to decide...on

(1) size of the instances
(2) representation
(3) time/memory considerations
Computation

Algorithm

(1) Size of the instances
(2) Representation
(3) Time/Memory considerations
Let $g : \mathbb{N} \rightarrow \mathbb{R}_{>0}$ be an arbitrary function

$$O(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_{>0} | \exists k \exists n_0 \forall n \ n \geq n_0 \rightarrow f(n) \leq kg(n)\}$$
Computation

Let \( g : \mathbb{N} \rightarrow \mathcal{R}_{>0} \) be an arbitrary function

\[
O(g(n)) = \{ f : \mathbb{N} \rightarrow \mathcal{R}_{>0} | \exists k \exists n_0 \forall n \ n \geq n_0 \rightarrow f(n) \leq kg(n) \}
\]

Prove \( n^2 \in O(n^3) \) (1)

\[
n \geq n_0 \rightarrow n^2 \leq kn^3
\]

(2)

\[
\frac{1}{n} \leq k
\]

(3)

let \( k = 2 \land n_0 = 1 \) (4)

Observe \( \forall n \ n \geq 1 \rightarrow n^2 \leq 2n^3 \)

QED
Recurrence relations—relations defined by

(1) initial point

(2) recursive (self-depreciation)

\[
\begin{align*}
    f_0 &= 0 \\
    f_1 &= 1 \\
    f_n &= f_{n-1} \times n
\end{align*}
\]

<table>
<thead>
<tr>
<th>(f(n))</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

Sequence Similitary (Computation) M.M. Dalkilic, PhD
Solv Indiana University, Bloomington, IN 2008 ©
Computation

\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_n = f_{n-1} \times n \]

#include <iostream>
using namespace std;

void main(void) {
    int f(int);
    int tf(int);
    
    cout << f(4) << endl;
    cout << tf(4) << endl;
}

int f(int n) {
    if (n <= 0) return 1;
    else return f(n-1) * n;
}

int tf(int n) {
    int t(int, int);
    return t(n,1);
}

int t(int x, int c) {
    if (x <= 0) return c;
    else return t(x-1, c*x);
}

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Though every recursive program can be written as a while

```
#include <iostream>
using namespace std;

void main(void) {
    int f(int);  
    cout << f(4) << endl;
}

int f(int n) {
    int temp(1);
    while (n>0) {
        temp*=(n--);
    }
    return temp;
}
```

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Computation

\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_n = f_{n-1} + f_{n-2} \]

```cpp
//recursive fib
int fib(int x) {
    static int cnt1(0);
    if (x < 1) {
        return 0;
    }
    if (x == 1){
        return 1;
    }
    else {
        cnt1++;
        cout << "call \\
        return fib(x-1) + fib(x-2);
    }
}
```
De Moivre proved \( f_n = \frac{1}{\sqrt{5}}(\Phi^n - (-\Phi)^n) \)

Recursive Fib \( \in O(\Phi^n) \), where \( \Phi = \frac{1+\sqrt{5}}{2} \)

\[
\begin{align*}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2}
\end{align*}
\]
Computation (Next Lecture)

Divide and Conquer gives rise to Dynamic Programming—the approach used in sequence comparison.

```cpp
// top down fib
int toptodownfib(int x) {
    static int cnt2(0);
    static int memory[9];
    if (memory[x] != 0) {
        return memory[x];
    }
    int t = x;
    if (t < 0) {
        return 0;
    }
    if (t == 1) {
        return 1;
    }
    if (t > 1) {
        cnt2++;
        cout << "call [" << cnt2 << "]" << endl;
        t = toptodownfib(x-1) + toptodownfib(x-2);
        memory[x] = t;
        return memory[x];
    }
}
```

While loop $\text{Fib} \in O(n)$, but an even better algorithm is $\text{Fib} \in O(\log(n))$. 