Upcoming Contests

• TopCoder Marathon Match 49
  (currently running – ends Feb. 18th)
TopCoder Open 2009

- Algorithms qualifying rounds:
  - Tuesday, February 24\textsuperscript{th} @ 7:00 AM
  - Saturday, February 28\textsuperscript{th} @ 12:00 PM
  - Wednesday, March 4\textsuperscript{th} @ 9:00 PM
TopCoder Open 2009

- Marathon Matches:
  - Wednesday, February 25th @ 12:00 PM
  - Wednesday, March 11th @ 12:00 PM
  - Wednesday, March 25th @ 12:00 PM
TopCoder Open 2009

- Other contests as well
  - Visit http://www.topcoder.com/tco09 for more information
<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tues</th>
<th>Wed</th>
<th>Thurs</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Marathon Match</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td><strong>Marathon Match</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>28</td>
</tr>
</tbody>
</table>

All times are Eastern Time unless stated otherwise. Stay informed with our [Upcoming Contests page](#) or subscribe to the RSS feed.
<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tues</th>
<th>Wed</th>
<th>Thurs</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Marathon Round 1</strong> 12:00 PM</td>
<td><strong>Algo Round 1</strong> 12:00 AM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Marathon Round 2</strong> 12:00 AM</td>
<td><strong>Algo Round 2</strong> 12:00 AM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Registration for All Tracks Closes</strong> 8:00 AM</td>
<td><strong>Algo Round 3</strong> 12:00 PM</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td><strong>SBM 431</strong> 11:00 AM</td>
<td><strong>Marathon Round 3</strong> 12:00 AM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Events Calendar (March)*
Dynamic Programming

Programming Puzzles and Competitions
CIS 4900 / 5920
Spring 2009
Lecture Outline

- Order Notation
- Solution to Text Segmentation
- Dynamic Programming
- Text Segmentation w/ Memoization
- Practice
Algorithmic Complexity and Order Notation

- We would like to be able to describe how long a program takes to run.
Algorithmic Complexity and Order Notation

- We would like to be able to describe how long a program takes to run
- We should express the runtime in terms of the input size
Algorithmic Complexity and Order Notation

- Absolute time measurements are not helpful, as computers get faster all the time

- Also, runtime is clearly dependent on the input (and input size)
Algorithmic Complexity and Order Notation

- We define $O(*)$ as follows

$$f(n) = O(g(n)) \iff \exists c, n_0 [n > n_0 \rightarrow f(n) < cg(n)]$$
Text Segmentation

• Some languages are written without spaces between the words
Text Segmentation

• Difficult for search engines to decipher the meaning of a search
Text Segmentation

• Difficult for search engines to decipher the meaning of a search

• Difficult to return relevant results
Text Segmentation

• Given a string without spaces, what is the best segmentation of the string?
Text Segmentation

- Examples:
  “upordown” → “up or down”
Text Segmentation

• Examples:
  “upordown” → “up or down”
  “upsidedown” → “upside down”
Text Segmentation

• Examples:
  “upordown” → “up or down”
  “upsidedown” → “upside down”
  “haveaniceday” → “have a nice day”
Text Segmentation

• Ambiguities are possible, e.g.:
  “theyouthevent” → ?
Text Segmentation

• At least three ways to segment “theyouthevent” into valid words:

1) “they out he vent”
2) “the you the vent”
3) “the youth event”
Text Segmentation

• At least three ways to segment “theyouthevent” into valid words:

1) “they out he vent”
2) “the you the vent”
3) “the youth event” (most likely)
Text Segmentation: Some ambiguities

- whorepresents.com
- therapistfinder.com
- speedofart.com
- expertsexchange.com
- penisland.com

These examples are borrowed from Peter Norvig.
Text Segmentation

• How can this be done?
Text Segmentation: Solution

• Recursion:

\[ P_{\text{max}}(y) = \max_i P_0(y[0:i]) \times P_{\text{max}}(y[i:n]); \]
\[ P_{\text{max}}(\text{""}) = 1; \]

where \( n = \text{length}(y) \)
Text Segmentation: Solution

• Recursion:

\[ P_{\text{max}}(y) = \max_i P_0(y[0:i]) \times P_{\text{max}}(y[i:n]) ; \]
\[ P_{\text{max}}(\"\"") = 1; \]

“base case”

where \( n = \text{length}(y) \)
Text Segmentation: Solution

\[ P_{\text{max}}(\text{"theyouthevent"}) = \max(\\]
\[ P_0(\text{"t"}) \times P_{\text{max}}(\text{"heyoutevent"}), \]
\[ P_0(\text{"th"}) \times P_{\text{max}}(\text{"eyoutevent"}), \]
\[ \ldots \]
\[ P_0(\text{"theyouthevent"}) \times P_{\text{max}}(\"\"") \]
double get_Pmax(const string &s)
{
    if(s.length() == 0)
        return get_P0(s);

    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_P0(s.substr(0, i - 0)) * get_Pmax(s.substr(i));

        if(temp > max)
            max = temp;
    }

    return max;
}
double get_Pmax(const string &s) {
    if(s.length() == 0)
        return get_P0(s);

    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_P0(s.substr(0, i - 1)) * get_Pmax(s.substr(i));

        if(temp > max)
            max = temp;
    }

    return max;
}
double get_Pmax(const string &s) {
    if(s.length() == 0)
        return get_P0(s);
    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_PO(s.substr(0, i - 0)) * get_Pmax(s.substr(i));
        if(temp > max)
            max = temp;
    }
    return max;
}
double get_Pmax(const string &s) {
    if(s.length() == 0)
        return get_P0(s);

    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_P0(s.substr(0, i - 0)) * get_Pmax(s.substr(i));
        if(temp > max)
            max = temp;
    }

    return max;
}
double get_Pmax(const string &s) {
    if(s.length() == 0)
        return get_P0(s);
    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_P0(s.substr(0, i - 0)) * get_Pmax(s.substr(i));
        if(temp > max)
            max = temp;
    }
    return max;
}

\[ P_{\text{max}}(y) = \max_{i} P_{0}(y[0:i]) \times P_{\text{max}}(y[i:n]) \]
Text Segmentation: Solution

- Recursion:

\[ P_{\text{max}}(y) = \max_{i} P_{0}(y[0:i]) \times P_{\text{max}}(y[i:n]); \]
\[ P_{\text{max}}(\"\") = 1; \]

where \( n = \text{length}(y) \)
Text Segmentation: Solution

• Note that this algorithm is (extremely) inefficient
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$

- $P_{\text{max}}(\text{"ent"})$
  - $P_{0}(\text{"en"}) \times P_{\text{max}}(\text{"t"})$
  - $P_{0}(\text{"ent"}) \times P_{\text{max}}(\text{""})$
- $P_{0}(\text{"e"}) \times P_{\text{max}}(\text{"nt"})$
  - $P_{0}(\text{"n"}) \times P_{\text{max}}(\text{"t"})$
  - $P_{0}(\text{"nt"}) \times P_{\text{max}}(\text{""})$
  - $P_{0}(\text{"t"}) \times P_{\text{max}}(\text{""})$
- $P_{0}(\text{"n"}) \times P_{\text{max}}(\text{"t"})$
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$
Fibonacci Numbers

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ for $n > 1$
Fibonacci Numbers

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ for $n > 1$

- 0, 1, 1, 2, 3, 5, 8, 13, ...
Fibonacci Numbers

• How would we write code for this?
Fibonacci Numbers

- How would we write code for this?

- \( F_0 = 0 \)
- \( F_1 = 1 \)
- \( F_n = F_{n-1} + F_{n-2} \) for \( n > 1 \)
Fibonacci Numbers

```c
int fibonacci(int n) {
    if(n == 0 || n == 1) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```
Fibonacci Numbers: Computing $F(5)$
Dynamic Programming

• Same algorithm as before, but compute each value only once
Dynamic Programming

- Same algorithm as before, but compute each value only once

- This *significantly* reduces the runtime
Dynamic Programming

• Can be done in one of two ways
  - iteration
  - recursion w/ memoization
Dynamic Programming

• Can be done in one of two ways
  - iteration (bottom-up)
  - recursion w/ memoization (top-down)
Dynamic Programming

- Can be done in one of two ways
  - iteration (bottom-up)
  - recursion w/ memoization (top-down)

- We will focus on the second of these (for now)
Dynamic Programming: Recursion w/ Memoization

- "Memoize"* previously computed function evaluations

*the word "memoize" stems from the word "memo"; it is not a misspelling of the word "memorize"
Dynamic Programming: Recursion w/ Memoization

- "Memoize"* previously computed function evaluations

- There is no reason to run the same computation twice

*the word “memoize” stems from the word “memo”; it is not a misspelling of the word “memorize"
Fibonacci Numbers: F(5) revisited
Fibonacci Numbers: F(5) revisited

- F(5)
- F(4)
- F(3)
- F(2)
- F(1)
- F(0)

Red = re-computations
Fibonacci Numbers: F(5) revisited
Dynamic Programming: Recursion w/ Memoization

• Implementation
  - keep a map from argument values to return values:
    $lookup[arguments] = \text{return value};$
Dynamic Programming: Recursion w/ Memoization

• Why does a lookup table help?
Dynamic Programming: Recursion w/ Memoization

• Why does a lookup table help?

• Avoids re-computations by saving previously computed values
Fibonacci Numbers
w/ Dynamic Programming

```cpp
int fibonacci(int n) {
    static map<int, int> memo;

    if(memo.find(n) != memo.end())
        return memo[n];

    if(n == 0 || n == 1)
        memo[n] = n;
    else
        memo[n] = fibonacci(n-1) + fibonacci(n-2);

    return memo[n];
}
```
Fibonacci Numbers w/ Dynamic Programming

```c
int fibonacci(int n) {
    static map<int, int> memo;
    if(memo.find(n) != memo.end())
        return memo[n];
    if(n == 0 || n == 1)
        memo[n] = n;
    else
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
    return memo[n];
}
```
Fibonacci Numbers: Computing $F(5)$
Fibonacci Numbers: Computing $F(5)$

Table Lookups
Fibonacci Numbers:
Computing $F(5)$
Fibonacci Numbers: F(5) w/ memoization

F(5) → F(4) → F(3) → F(2) → F(1) → F(0)

Table Lookups
Fibonacci Numbers: 
F(5) w/ memoization

F(5)
  /   \
F(4)  F(3)
   /    /
F(3)  F(2)
   /    /
F(2)  F(1)
   /    /
F(1)  F(0)

F(1)  F(0)
Text Segmentation w/ Dynamic Programming

double get_Pmax(const string &s)
{
    static map<string, double> probabilities;

    if(probabilities.find(s) != probabilities.end())
        return probabilities[s];

    if(s.length() == 0)
        return get_P0(s);

    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
            get_P0(s.substr(0, i - 0)) * get_Pmax(s.substr(i));

        if(temp > max)
            max = temp;
    }

    probabilities[s] = max;
    return max;
}
double get_Pmax(const string &s) {
    static map<string, double> probabilities;
    if(probabilities.find(s) != probabilities.end())
        return probabilities[s];
    if(s.length() == 0)
        return get_P0(s);
    double max = 0.0;
    for(int i = 1; i <= s.length(); ++i) {
        double temp =
                    get_P0(s.substr(0, i - 0)) * get_Pmax(s.substr(i));
        if(temp > max)
            max = temp;
    }
    probabilities[s] = max;
    return max;
}
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$

- $P_0(\text{"e"}) \times P_{\text{max}}(\text{"nt"})$
- $P_0(\text{"n"}) \times P_{\text{max}}(\text{"t"})$
- $P_0(\text{"t"}) \times P_{\text{max}}(\text{""})$

Table Lookup
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$

- $P_0(\text{"e"}) \times P_{\text{max}}(\text{"nt"})$
- $P_0(\text{"n"}) \times P_{\text{max}}(\text{"t"})$
- $P_0(\text{"t"}) \times P_{\text{max}}(\text{""})$

Table Lookup

- $P_0(\text{"en"}) \times P_{\text{max}}(\text{"t"})$
- $P_0(\text{"ent"}) \times P_{\text{max}}(\text{""})$
- $P_0(\text{"t"}) \times P_{\text{max}}(\text{""})$
Text Segmentation: Computing $P_{\max}("ent")$

- $P_0("e") \times P_{\max}("nt")$
- $P_0("n") \times P_{\max}("t")$
- $P_0("t") \times P_{\max}("\"\")$
- $P_0("en") \times P_{\max}("t")$
- $P_0("nt") \times P_{\max}("\")$
- $P_0("ent") \times P_{\max}("\")$

Table Lookup
Text Segmentation: Computing $P_{\text{max}}(\text{"ent"})$

$P_{\text{max}}(\text{"ent"})$

- $P_0(\text{"e"}) \times P_{\text{max}}(\text{"nt"})$
  - $P_0(\text{"n"}) \times P_{\text{max}}(\text{"t"})$
    - $P_0(\text{"t"}) \times P_{\text{max}}(\text{""})$
  - $P_0(\text{"nt"}) \times P_{\text{max}}(\text{""})$
- $P_0(\text{"en"}) \times P_{\text{max}}(\text{"t"})$
- $P_0(\text{"ent"}) \times P_{\text{max}}(\text{""})$
Dynamic Programming through Memoization

• Any questions?
Options

• TopCoder...
  - Marathon Match 49
  - SRM 418 – Division II

• Solve text segmentation using dynamic programming