APPLICATION:
Variable Elimination
Reminders

• Homework C: Data Structures
  – Out: Mon, Nov. 26
  – Due: Mon, Dec. 3 at 11:59pm

• Quiz B: Computation; Programming & Efficiency
  – Wed, Dec. 5, in-class
  – Covers Lectures 7 – 12
APPLICATION: EXACT INFERENCE IN GRAPHICAL MODELS
EXACT INFERENCE
Exact Inference

1. Data
\[ \mathcal{D} = \{ \mathbf{x}^{(n)} \}_{n=1}^{N} \]

2. Model
\[ p(\mathbf{x} | \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C) \]

3. Objective
\[ \ell(\boldsymbol{\theta}; \mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} | \boldsymbol{\theta}) \]

5. Inference
1. Marginal Inference
\[ p(\mathbf{x}_C) = \sum_{\mathbf{x}' : \mathbf{x}'_C = \mathbf{x}_C} p(\mathbf{x}' | \boldsymbol{\theta}) \]
2. Partition Function
\[ Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C) \]
3. MAP Inference
\[ \hat{\mathbf{x}} = \arg \max_{\mathbf{x}} p(\mathbf{x} | \boldsymbol{\theta}) \]

4. Learning
\[ \boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \mathcal{D}) \]
Complexity Classes

- An algorithm runs in **polynomial time** if its runtime is a polynomial function of the input size (e.g. $O(n^k)$ for some fixed constant $k$)
- The **class P** consists of all problems that can be solved in polynomial time

- A problem for which the answer is binary (e.g. yes/no) is called a **decision problem**
- The **class NP** contains all decision problems where ‘yes’ answers can be verified (proved) in polynomial time
- A problem is **NP-Hard** if given an $O(1)$ oracle to solve it, every problem in NP can be solved in polynomial time (e.g. by reduction)
- A problem is **NP-Complete** if it belongs to both the classes NP and NP-Hard
5. Inference

Three Tasks:

1. **Marginal Inference** (#P-Hard)
   Compute marginals of variables and cliques
   
   $$p(x_i) = \sum_{x': x'_i = x_i} p(x' | \theta)$$
   
   $$p(x_C) = \sum_{x': x'_C = x_C} p(x' | \theta)$$

2. **Partition Function** (#P-Hard)
   Compute the normalization constant
   
   $$Z(\theta) = \sum_{x} \prod_{C \in C} \psi_C(x_C)$$

3. **MAP Inference** (NP-Hard)
   Compute variable assignment with highest probability
   
   $$\hat{x} = \arg\max_x p(x | \theta)$$
Suppose we took many samples from the distribution over taggings:

\[ p(x) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(x_\alpha) \]
Marginals by Sampling on Factor Graph

The marginal $p(X_i = x_i)$ gives the probability that variable $X_i$ takes value $x_i$ in a random sample.
Marginals by Sampling on Factor Graph

Estimate the marginals as:

Sample 1: 

Sample 2: 

Sample 3: 

Sample 4: 

Sample 5:

Sample 6:

<START>
Simple and general exact inference for graphical models

VARIABLE ELIMINATION
Brute Force (Naïve) Inference

For all \( i \), suppose the range of \( X_i \) is \( \{0, 1, 2\} \).
Let \( k=3 \) denote the size of the range.
The distribution factorizes as:

\[
p(\mathbf{x}) = \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \\
\quad \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

Naively, we compute the partition function as:

\[
Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} p(\mathbf{x})
\]
Brute Force (Naïve) Inference

For all $i$, suppose the range of $X_i$ is $\{0, 1, 2\}$. Let $k=3$ denote the size of the range. The distribution factorizes as:

$$p(x) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$

Naively, we compute the partition function as:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} p(x)$$

$p(x)$ can be represented as a joint probability table with $3^5$ entries:
Brute Force (Naïve) Inference

For all $i$, suppose the range of $X_i$ is \{0, 1, 2\}. Let $k=3$ denote the size of the range.

The distribution factorizes as:

$$p(x) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$

Naively, we compute the partition function as:

$$Z = \sum x_1 \sum x_2 \sum x_3 \sum x_4 \sum x_5 p(x)$$

$p(x)$ can be represented as a joint probability table with $3^5$ entries:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$p(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.019517693</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.017090249</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.014885825</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.024117638</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.000925849</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0.028112576</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.028052020</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0.004812689</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.007987737</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.028433687</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.037073469</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0.013558227</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.019479016</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.012312901</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.023439775</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.038206131</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.038996005</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0.041458783</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.044616806</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.020846989</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0.03006475</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0.048436964</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.02854376</td>
</tr>
</tbody>
</table>

Naïve computation of $Z$ requires $3^5$ additions.
Can we do better?
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $p(x)$.

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi(x_5)$$

Only $3^2$ additions are needed to marginalize out $x_5$. We denote the marginal’s table by $m_5(x_4)$.

This “factor” is a much smaller table with $3^2$ entries:

<table>
<thead>
<tr>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$p(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.019517693</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.017090249</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0.014885825</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.024117638</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.000925849</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.028112576</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.028050205</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.004812689</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.007987737</td>
</tr>
</tbody>
</table>
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $p(x)$.

\[
Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)
\]

\[
= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4)\psi_{5}(x_4)
\]

\[
m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4, x_5)\psi_5(x_5)
\]
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $p(x)$.

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \psi_{12}(x_1, x_2) \psi_{13}(x_1, x_3) \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

This “factor” is still a $3^4$ table so apply the same trick again.

$$m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$
The Variable Elimination Algorithm

Instead, capitalize on the factorization of $p(x)$.

$$Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

$$= \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) m_3(x_1, x_2)$$

$$= \sum_{x_1} m_2(x_1)$$

Naïve solution requires $3^5 = 243$ additions.
Variable elimination only requires $3 + 3^2 + 3^3 + 3^3 + 3^2 = 75$ additions.
The Variable Elimination Algorithm

The same trick can be used to compute **marginal probabilities**. Just choose the variable elimination order such that the query variables are last.

The same trick can be used to compute **marginal probabilities**. Just choose the variable elimination order such that the query variables are last.

\[
p(x_1) = \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_5(x_5)
\]

\[
= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4)
\]

\[
= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)
\]

\[
= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) m_3(x_1, x_2)
\]

\[
= \frac{1}{Z} m_2(x_1)
\]

For directed graphs, \( Z = 1 \).

For undirected graphs, if we compute each (unnormalized) value on the LHS, we can sum them to get \( Z \).
The Variable Elimination Algorithm

In a factor graph, variable elimination corresponds to replacement of a subgraph with a factor.
The Variable Elimination Algorithm

In a factor graph, variable elimination corresponds to replacement of a subgraph with a factor.
The Variable Elimination Algorithm

In a factor graph, variable elimination corresponds to replacement of a subgraph with a factor.
The Variable Elimination Algorithm

\[ Z = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5) \psi_{5}(x_5) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4) \psi_{234}(x_2, x_3, x_4) m_5(x_4) \]

\[ = \sum_{x_1} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3) \]

In a factor graph, variable elimination corresponds to replacement of a subgraph with a factor.
Variable Elimination for Marginal Inference

Algorithm 1: Variable Elimination for Marginal Inference

**Input:** the factor graph and the query variable  
**Output:** the marginal distribution for the query variable

a. Run a breadth-first-search starting at the query variable to obtain an ordering of the variable nodes
b. Reverse that ordering
c. Eliminate each variable in the reversed ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

**Input:** the variable to be eliminated  
**Output:** new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set
b. Replace the eliminated set with a new factor  
   a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set  
   b. The new factor should assign a score to each possible assignment of its neighboring variables  
   c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable
Variable Elimination for Marginal Inference

Algorithm 3: Variable Elimination for the Partition Function

**Input:** the factor graph

**Output:** the partition function

a. Run a breadth-first-search starting at an arbitrary variable to obtain an ordering of the variable nodes

b. Eliminate each variable in the ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

**Input:** the variable to be eliminated

**Output:** new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set

b. Replace the eliminated set with a new factor

   a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set

   b. The new factor should assign a score to each possible assignment of its neighboring variables

   c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable
Variable Elimination Complexity

Instead, capitalize on the factorization of \( p(x) \).

In-Class Exercise: *Fill in the blank*

Brute force, naïve, inference is \( O(\text{____}) \)  Variable elimination is \( O(\text{____}) \)

where  
\( n = \# \) of variables  
\( k = \max \# \) values a variable can take  
\( r = \# \) variables participating in largest “intermediate” table
PROFILING FOR EFFICIENCY
Software Profiling

CPU Profiler:
- Intermediate Goal: Analyze the CPU usage of a program at a fine-grained level (e.g. time spent within each function)
- End Goal: To make the program more CPU efficient by optimizing most time consuming parts of program

Memory Profiler:
- Intermediate Goal: Analyze the memory consumption of a program (e.g. how much space does a particular type of object use on the heap)
- End Goal: To make the program more memory by utilizing different data structures or data storage techniques to reduce memory load
# Software Profiling

## Deterministic CPU Profiler
- Augments the code with additional bookkeeping calls
- Provides **exact number** of times each function is called, and **exact amount** of time spent in each function
- Comes at the cost of much slower runtime

## Statistical CPU Profiler
- Leaves the code nearly unchanged, and instead **takes samples** (hundreds or more) of the stacktrace
- Provides the **proportion of samples** that landed in each function and **estimates** the total time spent in each function
- Typically yields **little to no slowdown** of the code

## Line Profiler
- Same as above for each type, but counts the **number of times each line is executed** and provides the amount of **time spent on each line**
- Increases complexity of the profiler, but provides **much more detailed analysis**
## Python Profilers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Level of Detail</th>
<th>Output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cProfile</td>
<td>deterministic</td>
<td>function-level</td>
<td>console</td>
<td>built into Python standard library; C-based implementation</td>
</tr>
<tr>
<td>profile</td>
<td>deterministic</td>
<td>function-level</td>
<td>console</td>
<td>same as cProfile, but implemented in pure Python</td>
</tr>
<tr>
<td>line_profiler</td>
<td>deterministic + statistical</td>
<td>line-level</td>
<td>console</td>
<td>C-based implementation</td>
</tr>
<tr>
<td>pprofile</td>
<td>deterministic + statistical</td>
<td>line-level</td>
<td>console</td>
<td>pure Python implementation (few users)</td>
</tr>
<tr>
<td>PyFlame by Uber</td>
<td>deterministic + statistical</td>
<td>line-level</td>
<td>flame graph</td>
<td>Linux only as of 2018</td>
</tr>
<tr>
<td>Plop by Dropbox</td>
<td>deterministic + statistical</td>
<td>line-level</td>
<td>bubble plot</td>
<td>(few users)</td>
</tr>
</tbody>
</table>
## cProfile Output

```bash
$ python -m cProfile -s cumtime lwn2pocket.py
72270 function calls (70640 primitive calls) in 4.481 seconds

Ordered by: cumulative time

<table>
<thead>
<tr>
<th>ncalls</th>
<th>Ttime</th>
<th>Pcall</th>
<th>Tcumtime</th>
<th>Pcumtime</th>
<th>filename:lineno(function)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.004</td>
<td>0.004</td>
<td>4.481</td>
<td>4.481</td>
<td>lwn2pocket.py:2(&lt;module&gt;)</td>
</tr>
<tr>
<td>1</td>
<td>0.001</td>
<td>0.001</td>
<td>4.296</td>
<td>4.296</td>
<td>lwn2pocket.py:51(main)</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>4.286</td>
<td>1.429</td>
<td>api.py:17(request)</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>4.268</td>
<td>1.423</td>
<td>sessions.py:386(request)</td>
</tr>
<tr>
<td>4/3</td>
<td>0.000</td>
<td>0.000</td>
<td>3.815</td>
<td>1.272</td>
<td>sessions.py:539(send)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>2.965</td>
<td>0.741</td>
<td>adapters.py:323(send)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>2.962</td>
<td>0.740</td>
<td>connectionpool.py:421(urllopen)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>2.961</td>
<td>0.740</td>
<td>connectionpool.py:317(_make_request)</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>2.675</td>
<td>1.338</td>
<td>api.py:98(post)</td>
</tr>
<tr>
<td>30</td>
<td>0.000</td>
<td>0.000</td>
<td>1.621</td>
<td>0.054</td>
<td>ssl.py:727(recv)</td>
</tr>
<tr>
<td>30</td>
<td>0.000</td>
<td>0.000</td>
<td>1.621</td>
<td>0.054</td>
<td>ssl.py:610(read)</td>
</tr>
<tr>
<td>30</td>
<td>1.621</td>
<td>0.054</td>
<td>1.621</td>
<td>0.054</td>
<td>{method 'read' of '_ssl._SSLSocket' objects}</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.611</td>
<td>1.611</td>
<td>api.py:50(get)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>1.572</td>
<td>0.393</td>
<td>httplib.py:1095(getresponse)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>1.572</td>
<td>0.393</td>
<td>httplib.py:446(begin)</td>
</tr>
<tr>
<td>60</td>
<td>0.000</td>
<td>0.000</td>
<td>1.571</td>
<td>0.026</td>
<td>socket.py:410(readline)</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>1.571</td>
<td>0.393</td>
<td>httplib.py:407(_read_status)</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.462</td>
<td>1.462</td>
<td>pocket.py:44(wrapped)</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.462</td>
<td>1.462</td>
<td>pocket.py:152(make_request)</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.462</td>
<td>1.462</td>
<td>pocket.py:139(_make_request)</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.459</td>
<td>1.459</td>
<td>pocket.py:134(_post_request)</td>
</tr>
</tbody>
</table>
```

Figure from https://julien.danjou.info/guide-to-python-profiling-cprofile-concrete-case-carbonara/
Pystone(1.1) time for 50000 passes = 2.48
This machine benchmarks at 20161.3 pystones/second
Wrote profile results to pystone.py.lprof
Timer unit: 1e-06 s

File: pystone.py
Function: Proc2 at line 149
Total time: 0.606656 s

<table>
<thead>
<tr>
<th>Line #</th>
<th>Hits</th>
<th>Time</th>
<th>Per Hit</th>
<th>% Time</th>
<th>Line Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>@profile</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>def Proc2(IntParIO):</td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>50000</td>
<td>82003</td>
<td>1.6</td>
<td>13.5</td>
<td>IntLoc = IntParIO + 10</td>
</tr>
<tr>
<td>152</td>
<td>50000</td>
<td>63162</td>
<td>1.3</td>
<td>10.4</td>
<td>while 1:</td>
</tr>
<tr>
<td>153</td>
<td>50000</td>
<td>69065</td>
<td>1.4</td>
<td>11.4</td>
<td>if Char1Glob == 'A':</td>
</tr>
<tr>
<td>154</td>
<td>50000</td>
<td>66354</td>
<td>1.3</td>
<td>10.9</td>
<td>IntLoc = IntLoc - 1</td>
</tr>
<tr>
<td>155</td>
<td>50000</td>
<td>67263</td>
<td>1.3</td>
<td>11.1</td>
<td>IntParIO = IntLoc - IntGlob</td>
</tr>
<tr>
<td>156</td>
<td>50000</td>
<td>65494</td>
<td>1.3</td>
<td>10.8</td>
<td>EnumLoc = Ident1</td>
</tr>
<tr>
<td>157</td>
<td>50000</td>
<td>68001</td>
<td>1.4</td>
<td>11.2</td>
<td>if EnumLoc == Ident1:</td>
</tr>
<tr>
<td>158</td>
<td>50000</td>
<td>63739</td>
<td>1.3</td>
<td>10.5</td>
<td>break</td>
</tr>
<tr>
<td>159</td>
<td>50000</td>
<td>61575</td>
<td>1.2</td>
<td>10.1</td>
<td>return IntParIO</td>
</tr>
</tbody>
</table>
PyFlame Output

Figure from https://github.com/uber/pyflame
Plop Output

Figure from https://blogs.dropbox.com/tech/2012/07/plop-low-overhead-profiling-for-python/