Problem Set #4 has been graded
Problem Set #5 by Thursday
Randomized algorithms incorporate random, rather than deterministic, decisions.

- Commonly used in situations where no exact and/or fast algorithm is known.
- Works for algorithms that behave well on typical data, but poorly in special cases.
- Main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time.
Select Algorithm

- **Select(L, k)** finds the $k^{th}$ smallest element in $L$
- **Select(L, 1)** find the smallest element in a list:
  - Well known $O(n)$ algorithm
    
    ```python
    minv = HUGE
    for v in L:
        if (v < minv):
            minv = v
    ```

- **Select(L, len(L)/2)** finds the median...
  - How?
    - median = sorted(L)[len(L)/2] → $O(n \log n)$
- Can we find medians, or 1st quartiles in $O(n)$?
Recursive Select

- **Select**(\(L, k\)) finds the \(k^{th}\) smallest element in \(L\)
  - Select an element \(m\) from unsorted list \(L\) and partition \(L\) into two smaller lists:
    - \(L_{lo}\) - elements smaller than \(m\)
    - \(L_{hi}\) - elements larger than \(m\)
  - if \(\text{len}(L_{lo}) > k\):
    - Select(\(L_{lo}, k\))
  - else if \(k > \text{len}(L_{lo}) + 1\):
    - Select(\(L_{hi}, k - \text{len}(L_{lo}) - 1\))
  - else:
    - \(m\) is the \(k^{th}\) smallest element
Example of \textit{Select}(L,5)

Given the array: \( L = \{6, 3, 2, 8, 4, 5, 1, 7, 0, 9\} \)

- \textbf{Step 1:} Choose the first element as \( m \)

\[
L = \{6, 3, 2, 8, 4, 5, 1, 7, 0, 9\}
\]

- \textbf{Step 2:} Split into two lists

\[
L_{lo} = \{3, 2, 4, 5, 1, 0\}
\]
\[
L_{hi} = \{8, 7, 9\}
\]
Example of $\text{Select}(L,5)$ (continued)

- **Step 3** Recursively call Select on either $L_{lo}$ or $L_{hi}$ until $\text{len}(L_{lo}) = k$, then return $m$.

  \[
  \text{len}(L_{lo}) > k = 5 \rightarrow
  \text{Select}\{3, 2, 4, 5, 1, 0\}, 5
  \]
  \[m = 3 \quad L_{lo} = \{2, 1, 0\} \quad L_{hi} = \{4, 5\}\]

  \[k = 5 > \text{len}(L_{lo}) + 1 \rightarrow
  \text{Select}\{4, 5\}, 5 - 3 - 1
  \]
  \[m = 4 \quad L_{lo} = \{\} \quad L_{hi} = \{5\}\]

  \[k = 1 > \text{len}(L_{lo}) + 1 \rightarrow
  \text{return} \ 4\]
```python
def select(L, k):
    value = L[0]
    Llo = [t for t in L if t < value]
    Lhi = [t for t in L if t > value]
    below = len(Llo) + 1
    if (k < len(Llo)):
        return select(Llo, k)
    elif (k > below):
        return select(Lhi, k - below)
    else:
        return value

print(select([6, 3, 2, 8, 4, 5, 1, 7, 0, 9], 5))
```

4
Select(L,k) Performance

Runtime depends on our selection of \( m \):

- A *good* selection will split \( L \) evenly such that
  \[
  |L_{lo}| = |L_{hi}| = \frac{|L|}{2}
  \]

- The recurrence relation is:
  \[
  T(n) = T\left(\frac{n}{2}\right)
  \]

\[
  n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \ldots = 2n \rightarrow O(n)
  \]

- which is the same as the search for \( \text{min}(L) \) or \( \text{max}(L) \)
However, a poor selection will split $L$ unevenly and in the worst case, all elements will be greater or less than $m$ so that one Sublist is full and the other is empty.

- For a poor selection, the recurrence relation is:

$$T(n) = T(n - 1)$$

- In this case, the runtime is $O(n^2)$, which is worse than sorting first and selecting the $k^{th}$ value

Our dilemma: $O(n)$ or $O(n^2)$ depending on the list... or $O(n \log n)$ independent of it
Select(L,k) verdict

- Select seems risky compared to sort
- To improve Select, we need to choose $m$ to give good ‘splits’
- It can be proven that to achieve $O(n)$ running time, we don’t need a perfect splits, just reasonably good ones.
- In fact, if both subarrays are at least of size $n/4$, then running time will be $O(n)$.
- This implies that half of the choices of $m$ make good splitters.
A Randomized Approach

- To improve $\text{Select}(L,k)$, randomly select $m$.
- Since half of the elements will be good splitters, if we choose $m$ at random we will get a 50% chance that $m$ will be a good choice.
- This approach will make sure that no matter what input is received, the expected running time is small.
import random

def randomizedSelect(L, k):
    value = random.choice(L)
    Llo = [t for t in L if t < value]
    Lhi = [t for t in L if t > value]
    below = len(Llo) + 1
    if (k < len(Llo)):
        return randomizedSelect(Llo, k)
    elif (k > below):
        return randomizedSelect(Lhi, k-below)
    else:
        return value

print randomizedSelect([6, 3, 2, 8, 4, 5, 1, 7, 0, 9], 5)
%timeit randomizedSelect(range(10000), 500)

5
100 loops, best of 3: 1.97 ms per loop
RandomizedSelect($L,k$) Performance

- Worst case runtime: $O(n^2)$
- Expected runtime: $O(n)$
- Expected runtime is a good measure of the performance of randomized algorithms, often more informative than worst case runtimes.
- Worst case runtimes are rarely repeated
- RandomizedSelect($L,k$) always returns the correct answer, which offers a way to classify Randomized Algorithms.
Two Types of Randomized Algorithms

- **Las Vegas Algorithms** – always produce the correct solution (i.e. randomizedSelect)
- **Monte Carlo Algorithms** – do not always return the correct solution.
- Las Vegas Algorithms are always preferred, but they are often hard to come by.
The Motif Finding Problem

Given a list of \( t \) sequences each of length \( n \), find the “best” matching pattern of length \( l \) that appears in each of the \( t \) sequences.
A New Approach

- **Motif Finding Problem**: Given a list of $t$ sequences each of length $n$, find the “best” pattern of length $l$ that appears in each of the $t$ sequences.
- **Previously**: we solved the Motif Finding Problem using a Branch and Bound or a Greedy technique.
- **Now**: *Randomly* select possible locations and find a way to greedily change those locations until we converge to the hidden motif.
Profiles Revisited

- Let $s = (s_1, s_2, \ldots, s_t)$ be the starting positions for $l$-mers in our $t$ sequences.
- The substrings corresponding to these starting positions will form:
  - $t \times l$ alignment matrix
  - $4 \times l$ profile matrix
- Normalized counts that they represent the fraction of each base at each position

\[
\begin{array}{cccccc}
A & 0.6 & 0.0 & 0.2 & 0.0 & 0.2 \\
C & 0.0 & 0.4 & 0.8 & 0.0 & 0.0 \\
G & 0.0 & 0.0 & 0.2 & 0.8 & 0.0 \\
T & 0.0 & 0.0 & 0.0 & 1.0 & 0.2 \\
X & 0.0 & 0.0 & 0.0 & 0.0 & 0.8 \\
\end{array}
\]

$P(X|\text{profile}) = 0.6 \times 0.8 \times 0.8 \times 1.0 \times 0.6 \times 0.8 \times 0.6 \times 0.8 = 0.0885$
Scoring Strings with a Profile

- Let k-mer, \( a = a_1, a_2, a_3, \ldots a_k \)
- \( \text{Prob}(a|P) \) is defined as the probability that an k-mer \( a \) was created by the Profile \( P \).
- If \( a \) is very similar to the consensus string of \( P \) then \( \text{Prob}(a|P) \) will be high.
- If \( a \) is very different, then \( \text{Prob}(a|P) \) will be low.

\[
\text{Prob}(a|P) = \prod_{i=1}^{l} p(a_i, i)
\]
Scoring with a Profile

Given the profile: $P =$

<table>
<thead>
<tr>
<th>base</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>7/8</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
<td>1/4</td>
<td>1/8</td>
</tr>
</tbody>
</table>

The probability of the consequence string:
$\text{Prob}(aaacct|P) = \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} = 0.033646$

The probability of a different string:
$\text{Prob}(atacag|P) = \frac{1}{2} \times \frac{1}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{1}{8} \times \frac{1}{8} = 0.001602$
P-Most Probable k-mer

- Define the P-most probable k-mer from a sequence as a k-mer in that sequence which has the highest probability of being created from the profile $P$.

<table>
<thead>
<tr>
<th>base</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>7/8</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
<td>1/4</td>
<td>1/8</td>
</tr>
</tbody>
</table>

Given a sequence = CTATAACCTTACATC, find the P-most probable k-mer

Find the $\text{Prob}(a|P)$ of every possible 6-mer

- CTATAACCTTACATC
- CTATAAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
- CTATAACCTTACATC
## P-Most Probable k-mer

Compute $Prob(a|P)$ of every possible 6-mer

<table>
<thead>
<tr>
<th>String highlighted in red</th>
<th>Path</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×1/8×3/8×0×1/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAAACCTTACATC</td>
<td>1/2×7/8×0×0×1/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/2×1/8×3/8×0×1/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×7/8×3/8×0×3/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/2×7/8×3/8×5/8×3/8×7/8</td>
<td>0.0336</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/2×7/8×1/2×5/8×1/4×7/8</td>
<td>0.0299</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/2×0×1/2×0×1/4×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×0×0×0×1/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×1/8×0×0×3/8×0</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×1/8×3/8×5/8×1/8×7/8</td>
<td>0.0004</td>
</tr>
<tr>
<td>CTATAAACCTTACATC</td>
<td>1/8×7/8×1/2×0×1/4×0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **AACCT** is the P-most probable 6-mer
Dealing with Zeros

- In our toy example $\text{Prob}(a|P)$, in many cases. In practice, there will be enough sequences so that the number of elements in the profile with a frequency of zero is small.
- To avoid many entries with $\text{Prob}(a|P)$, there exist techniques to equate zero to a very small number so that one zero does not make the entire probability of a string zero (assigning a prior probability, we will not address these techniques here).
P-Most Probable k-mers in Many Sequences

- Find the P-most probable k-mer in each of the “t” sequences.

<table>
<thead>
<tr>
<th>base</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>7/8</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
<td>1/4</td>
<td>1/8</td>
</tr>
</tbody>
</table>

cataaacgttacatc
tagcgattcgactga
cagccccagaacctgg
cggtgaaccttacatc
tgcatatagctta
tgtcctgtccactcact
tccaatcccttttaca
ggtctacctttatcct
Next Time

- A consensus of consensus
- When does randomization show up?

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>g</td>
<td>t</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>g</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>4</td>
<td>g</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>g</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>t</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>8</td>
<td>t</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

A 5/8 5/8 4/8 0 0 0
C 0 0 4/8 6/8 4/8 0
T 1/8 3/8 0 0 3/8 6/8
G 2/8 0 0 2/8 1/8 2/8

ctataacgttacatc
atagcgattcgactga
cagcccagaaacctgg
cggtaaccttacatc
tgcattcaatagctta
tgtctgtccactcac
tctcaaatcctttaca
ggtctacctttatcct