“All the News that fits is printed”

CS 5 Green Today

HMC CS Professor Develops New Search Engine

Claremont, CA: A Harvey Mudd CS Professor has developed a sophisticated new search engine called Giigle that gives entirely random responses to search queries. “It’s not very useful, but it is kinda funny,” said one dedicated user. “I put in a query for ‘pumpkin bread recipes’ and the top hit was a website on smelly socks. It made me chuckle.” The competition, however, is not laughing. “We’ve been working on this idea for over 5 years now,” said a VP for a large search engine company. “The idea was ours and we plan to sue for intellectual property infringement.” Most experts believe that such a lawsuit will not be successful. “We’ve done some research on this,” said one prominent IP attorney. “When you do a websearch on ‘search engine intellectual property’ the only thing that comes up are hits about spam donuts, how to change the air filter in your car, and a blog on cool things to make out of rotten pumpkins. As far as we can tell, there is no precedent for suing for this kind of thing.” A more likely scenario, experts agree, is that Giigle will be bought out for a large sum of money by a major competitor. “This is a big idea,” said the HMC professor. “They’ll have to offer me a three-figure sum for me to even consider it.”

New computer program predicts today’s weather: IndexError: list index out of range

News in Brief

Hoch-Shanahan to introduce Spam smoothies. Page 42

Spam smoothies plan cancelled. Page 42

“Bring back the Spam smoothies!” say students. Page 42

Spam smoothies to be reintroduced next week. Page 42

Blenders go up in flames in Hoch-Shanahan. Page 42

“No more Spam in blenders!” says Fire Marshall. Page 42

Spam pancakes make debut next Week. Page 42

Sports: CS Profs to compete in SPam Eating Week 2018 (SPEW 2018)

New computer program predicts today’s weather: IndexError: list index out of range
Last time: inferring chromosome homology based on gene homology
Last time: finding orthologs using `alignScore` and the best reciprocal hit method

<table>
<thead>
<tr>
<th>human</th>
<th>chicken</th>
<th>human</th>
<th>chicken</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>c1</td>
<td>h1</td>
<td>c1</td>
</tr>
<tr>
<td>h2</td>
<td>c2</td>
<td>h2</td>
<td>c2</td>
</tr>
<tr>
<td>h3</td>
<td>c3</td>
<td>h3</td>
<td>c3</td>
</tr>
<tr>
<td>h4</td>
<td>c4</td>
<td>h4</td>
<td>c4</td>
</tr>
<tr>
<td>h5</td>
<td>c5</td>
<td>h5</td>
<td>c5</td>
</tr>
<tr>
<td></td>
<td>c6</td>
<td></td>
<td>c6</td>
</tr>
</tbody>
</table>
Scenario 1: X and Z partially homologous
Scenario 2: X and Z not homologous

- Ancestor
  - X
    - Mammal
  - Z
    - Bird
What is an alignment anyway?

Alignment representation

S1  GCCTGG–
S2  ACC–GGA

GCCTGG → ACCTGG (change G to A)
   (keep the C)
   (keep the C)
ACCTG → ACCGG (delete the T)
   (keep the G)
   (keep the G)
ACCGG → ACCGGA (insert A)
Returning the alignment

```python
>>> alignScore("GCCTGG","ACCGGA",-4,dnamat)
11
```
```
alignScore only returns score
```

```python
>>> align("GCCTGG","ACCGGA",-4,dnamat)
[11, 'gC.CtGG-', 'aCC-GGa']
```
```
align returns score and alignment
```

```
gC.CtGG-
aCC-GGa
```
Returning the solution: example with change

```python
def change(amount, denominations):
    ''' Returns the least number of coins required to make the given amount using the list of provided denominations.'''
    if amount == 0: return 0
    elif denominations == []: return float('infinity')
    elif denominations[0] > amount: return change(amount, denominations[1:])
    else:
        useIt = 1 + change(amount - denominations[0], denominations)
        loseIt = change(amount, denominations[1:])
        return min(useIt, loseIt)

>>> change(42, [25, 21, 1])
2

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
```
def showChange(amount, denominations):
    ''' Takes an integer amount and a list of possible denominations. Returns a list containing two elements. First the minimum number of coins required, and second a list of the actual coins used in that solution. '''

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
showChange

def showChange(amount, denominations):
    ''' Takes an integer amount and a list of possible
denominations. Returns a list containing two elements. First
the minimum number of coins required, and second a list of the
actual coins used in that solution.''
    if amount == 0:
        elif denominations == []:

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
showChange

def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible
denominations. Returns a list containing two elements. First
the minimum number of coins required, and second a list of the
actual coins used in that solution.''
    if amount == 0: return [0, []]
elif denominations == []: return [float('infinity'), []]

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible
denominations. Returns a list containing two elements. First
the minimum number of coins required, and second a list of the
actual coins used in that solution.''
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount:
        >>> showChange(42, [25, 21, 1])
        [2, [21, 21]]
```python
def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible denominations. Returns a list containing two elements. First the minimum number of coins required, and second a list of the actual coins used in that solution.'''
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount: return showChange(amount, denominations[1:])

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
```
def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible denominations. Returns a list containing two elements. First the minimum number of coins required, and second a list of the actual coins used in that solution.'''
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount: return showChange(amount, denominations[1:])
    else:
        useIt = showChange( amount - denominations[0], denominations )
        loseIt = showChange( amount, denominations[1:] )

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible denominations. Returns a list containing two elements. First the minimum number of coins required, and second a list of the actual coins used in that solution.''
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount: return showChange(amount, denominations[1:])
    else:
        useIt = showChange(amount - denominations[0], denominations)
        loseIt = showChange(amount, denominations[1:]))

        if useIt[0] <= loseIt[0]:
            return useIt
        else:
            return loseIt

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
def showChange( amount, denominations ):
    ''' Takes an integer amount and a list of possible denominations. Returns a list containing two elements. First the minimum number of coins required, and second a list of the actual coins used in that solution.'''
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount: return showChange(amount, denominations[1:])
    else:
        useIt = showChange(amount - denominations[0], denominations)
        useIt = [useIt[0] +1, [denominations[0]] + useIt[1] ]

        loseIt = showChange(amount, denominations[1:])

        if useIt[0] <= loseIt[0]:
            return useIt
        else:
            return loseIt

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
change and showChange return different types

```python
>>> change(42, [25, 21, 1])
2

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
```
A common mistake...

```python
def change(amount, denominations):
    if amount == 0: return 0
    elif denominations == []: return float('infinity')
    elif denominations[0] > amount: return change(amount, denominations[1:])
    else:
        useIt = 1 + change(amount - denominations[0], denominations)
        loseIt = change(amount, denominations[1:])
        return min(useIt, loseIt)

def showChange(amount, denominations):
    if amount == 0: return [0, []]
    elif denominations == []: return [float('infinity'), []]
    elif denominations[0] > amount: return change(amount, denominations[1:])
    else:
        useIt = change(amount - denominations[0], denominations)
        useIt = [useIt[0] +1, [denominations[0]] + useIt[1] ]
        loseIt = change(amount, denominations[1:])
        if useIt[0] <= loseIt[0]:
            return useIt
        else:
            return loseIt
```

TypeError

Schmype error!
def showChange(amount, denominations):
    ''' Takes an integer amount and a list of possible
denominations. Returns a list containing two elements. First
the minimum number of coins required, and second a list of the
actual coins used in that solution.''
    if amount == 0: return [0, []]
elif denominations == []: return [float('infinity'), []]
elif denominations[0] > amount: return showChange(amount, denominations[1:])
else:
    useIt = showChange(amount - denominations[0], denominations)
    useIt = [useIt[0] +1, [denominations[0]] + useIt[1] ]
    loseIt = showChange(amount, denominations[1:])
    if useIt[0] <= loseIt[0]:
        return useIt
    else:
        return loseIt

>>> showChange(42, [25, 21, 1])
[2, [21, 21]]
def subset( target, numberList ):
    ''' Returns True if there exists a subset of numberList that adds
       up to target and returns False otherwise.''
    if target == 0: return True
    elif numberList == []: return False
    elif numberList[0] > target: return subset(target, numberList[1:])
    else:
        useIt = subset(target - numberList[0], numberList[1:])
        loseIt = subset(target, numberList[1:])
        return useIt or loseIt

>>> subset(9,[2,3,5])
False
showSubset

def showSubset( target, numberList ):
    '''Returns a care package indicating whether there is a subset of
    numberList that adds up to target.'''

>>> showSubset(9,[2,3,5])
[False, []]

>>> showSubset(10,[2,3,5])
[True, [5, 3, 2]]
def showSubset( target, numberList ):
    '''Returns a care package indicating whether there is a subset of numberList that adds up to target.'''
    if target == 0: return [True, []]
    elif numberList == []: return [False, []]
    elif numberList[0] > target: return showSubset(target, numberList[1:])
    else:
        loseIt = showSubset(target, numberList[1:])

        useIt = showSubset(target - numberList[0], numberList[1:])
        if useIt[0]:
            # don't add if False
            useIt = [useIt[0], [numberList[0]] + useIt[1]]

        if useIt[0]: return useIt
        else: return loseIt

>>> showSubset(9,[2,3,5])       >>> showSubset(10,[2,3,5])
[False, []]                        [True, [5, 3, 2]]
def LCS( S1, S2 ):
    ''' Return the length of the longest common subsequence 
of strings S1 and S2.'''
    if S1 == '' or S2 == '': return 0
    elif S1[0] == S2[0]: return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1[1:], S2)
        option2 = LCS(S1, S2[1:])
        return max(option1, option2)

>>> fancyLCS("human", "chimpanzee")
[4, 'h#man', '#h#m#an###']
def fancyLCS(S1, S2):
    """Returns a care package of the form [number, string1, string2] where number is the LCS length and string1 and string2 are pounded out versions of S1 and S2."""

    if S1 == '':
        elif S2 == '':

>>> fancyLCS("human", "chimpanzee")
[4, 'h#man', '#h#m#an###']
def fancyLCS(S1, S2):
    """Returns a care package of the form [number, string1, string2] where number is the LCS length and string1 and string2 are pounded out versions of S1 and S2."""

    if S1 == '': return [0,'',len(S2)*'#']
    elif S2 == '':

    >>> fancyLCS("human", "chimpanzee")
    [4, 'h#man', '#h#m#an###']
def fancyLCS(S1, S2):
    """Returns a care package of the form [number, string1, string2] where number is the LCS length and string1 and string2 are pounded out versions of S1 and S2."""

    if S1 == '': return [0,'',len(S2)*'#']
    elif S2 == '': return [0,len(S1)*'#','']
    elif S1[0] == S2[0]:
        else:

        >>>
        fancyLCS("human", "chimpanzee")
        [4, 'h#man', '#h#m#an###']

        In your notes...
def fancyLCS(S1, S2):
    """Returns a care package of the form [number, string1, string2] where number is the LCS length and string1 and string2 are pounded out versions of S1 and S2."""

    if S1 == '': return [0,'',len(S2)*'#']
    elif S2 == '': return [0,len(S1)*'#','']
    elif S1[0] == S2[0]:
        match=fancyLCS(S1[1:],S2[1:])
        match=[1+match[0],S1[0]+match[1],S2[0]+match[2]]
        return match
    else:
def fancyLCS(S1, S2):
    """Returns a care package of the form [number, string1, string2] where number is the LCS length and string1 and string2 are pounded out versions of S1 and S2."""

    if S1 == '': return [0, '', len(S2)*'#'
    elif S2 == '': return [0, len(S1)*'#', '"
    elif S1[0] == S2[0]:
        match = fancyLCS(S1[1:], S2[1:])
        match = [1+match[0], S1[0]+match[1], S2[0]+match[2]]
        return match
    else:
        option1 = fancyLCS(S1[1:], S2)
        option1 = [option1[0], '#'+option1[1], option1[2]]

        option2 = fancyLCS(S1, S2[1:])
        option2 = [option2[0], option2[1], '#'+option2[2]]

        if option1[0] > option2[0]:
            return option1
        else: return option2
Homework: superLCS

```python
>>> superLCS("human", "chimpanzee")
[4, '-hu-m-an---', 'ch-impanzee']

>>> superLCS("A","AT")
[1, 'A-', 'AT']

>>> superLCS("CG","G")
[1, 'CG', '-G']
```
GAC

AC

---

fancyLCS

[2, '#AC', 'AC']

---

GAC

AC

---

superLCS

[2, 'GAC', '-AC']
Homework: align

```python
>>> align("GCCTGG","ACCGGA",-4,dnamat)
[11, 'gCCTGG-', 'aCC-GGa']
```

gCCTGG-
aCC-GGa
Population power. Extreme throughput. $1,000 human genome.

The HiSeq X Ten is a set of ten ultra-high-throughput sequencers, purpose-built for large-scale human whole-genome sequencing.
UK Biobank data on 500,000 people paves way to precision medicine

Treatments tailored to individuals rely on the wisdom of crowds.
Read mapping is really an alignment problem

Reference: CGCCGAATAGAACCAGGAGCCGG- CGAGGCAGGATAAA
Read: ggCGTGAG-CGGACGAGcaat