CS 5 and Physics Penguins Stranded in Spaceship Crash

Wellington (AP): Two HMC penguins were missing after their spaceship lost power and crashed into the southern ocean. “The CS penguin had kindly offered a ride to her friend,” blubbered a distraught professor, “and apparently he was fiddling with the flight computer just before takeoff. I don’t know what I’ll do for classroom examples now.”

With weather worsening, there is little hope for rescue. A memorial service will be held Sunday in the Hoch-Shanahan freezer.
This Week

Homework 3:
  • Reading

  • Black lab (Problem 1) is same as Gold this week

  • Problem 2: Spel Chekking
  • Problem 3: Word Break game
def doubleList1(L):
    return list(map(lambda x: 2*x, L))

def doubleList2(L):
    return [2*x for x in L]

def doubleListFiltered1(L):
    return list(map(lambda x: 2*x, filter(lambda x: x != 42, L)))

def doubleListFiltered2(L):
    return [2*x for x in L if x != 42]
>>> list(map(lambda x: "nice" if x == 42 else "blech!", [42, 7, 6, 42, 3]))
["nice", "blech!", "blech!", "nice", "blech!"]

>>> list(map(lambda x: "HM" if x == 42 else "PO" if x == 47 else x, [42, 7, 6, 47, 3]))
['HM', 7, 6, 'PO', 3]
How Efficient is LCS?

def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    else:
        if S1[0] == S2[0]:
            return 1 + LCS(S1[1:], S2[1:])
        else:
            return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))

Demo LCS, fastLCS

Two strings of length 100 nucleotides each...

>>> steps = 2**100
>>> speed = 3 * 10**9
>>> seconds = steps / speed
>>> years = seconds / (60*60*24*365.25)
>>> years
13389807845846.213

13 trillion years!
I love “am and ims”!
Dictionaries Revisited

```python
>>> D = {"spam" : "yummy!", (42, 42): "an important point"}
>>> D["spam"]
"yummy!"
>>> D[(42, 42)]
"an important point"
>>> D[ ["zaster!", "putrid", "smoke!"], ]
BARF!
```
Dictionaries Revisited

```python
>>> D = {
    "spam" : "yummy!",
    (42, 42): "an important point"
}
>>> D = {}
>>> D[ "spam" ] = "yummy!"
>>> D[ (42, 42) ] = "an important point"
>>> D[ [1, 2] ] = "but this is bad"
BARF!
>>> "spam" in D
True
>>> 42 in D
False
>>> (42, 42) in D
True
>>> D[(42, 42)]
"an important point"
```
How Dictionaries Work: Hashing

```python
>>> D
{"Julie": "chocolate", ... }
>>> x = (1, 2)
>>> D[x] = "my tuple"
>>> D
{"Julie": "chocolate", (1, 2): "my tuple"}
```

Imagine that we now changed `x[0]=42`

D["Julie"] → 5999

D[x] =
D[(1, 2)]

Memory locations

```
1
2
3  "my tuple"
5999  "chocolate"
6000
```
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1+LCS(S1[1:], S2[1:])
    else:
        return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))

memo = {} # global empty dictionary
def fastLCS(S1, S2):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "":
        answer = 0
    elif S1[0] == S2[0]:
        answer = 1+fastLCS(S1[1:], S2[1:])
    else:
        answer = max(fastLCS(S1, S2[1:]), fastLCS(S1[1:], S2))
    memo[(S1, S2)] = answer
    return answer
def change(value, coins):
    if value <= 0:
        return 0
    elif coins == []:
        return float("inf")
    loseIt = change(value, coins[1:])
    if value < coins[0]:
        return loseIt
    else:
        useIt = 1 + change(value - coins[0], coins)
        return min(useIt, loseIt)
Changing change

memo = {}       # Empty dictionary

def fastChange(value, coins):
    if (value, coins) in memo:
        return memo[(value, coins)]
    elif value <= 0:
        return 0
    elif coins == []:   # coins must be a tuple rather than a list!
        return float("inf")

    # Finish writing this!

My solution coming up…
Changing change

```python
memo = {}       # Empty dictionary

def fastChange(value, coins):
    if (value, coins) in memo:
        return memo[(value, coins)]
    elif value <= 0:
        return 0
    elif coins == []:
        return float("inf")
    else:
        loseIt = fastChange(value, coins[1:])
        if value < coins[0]:
            memo[(value, coins)] = loseIt
            return loseIt
        else:
            useIt = 1 + fastChange(value - coins[0], coins)
            best = min(useIt, loseIt)
            memo[(value, coins)] = best
            return best
```

*coins must be a tuple rather than a list!*
Beyond LCS: Edit Distance

```python
>>> ED("ATTATCG", "ACATTC")
4
```

ATTAT–CG
A–CATTC–

The lower the edit distance the better!
Beyond LCS: Edit Distance

>>> ED("ATTATCG", "ACATTC")
4

ATTAT-CG
A-CATTC-

>>> ED("spam", "scramble")
5

sp_am___
scramble

spam ->
scam ->
scram ->
scramb -> scrambl -> scramble
Beyond LCS: Edit Distance

>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
scramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '': return ???
    elif S2 == '': return ???
    elif S1[0] == S2[0]: return ???
    else:  # substitute, insert, or delete!
>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
scramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ???
    else:  # substitute, insert, or delete!
Beyond LCS: Edit Distance

```python
>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
scramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:]))
    else:  # substitute, insert, or delete!
        substitute =
        insert =
        delete =
        return min(substitute, insert, delete)
```
Beyond LCS: Edit Distance

```python
>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
scramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert =
        delete =
    return min(substitute, insert, delete)
```
Beyond LCS: Edit Distance

```python
>>> ED("spam", "scramble")
5
spam ->
scam ->
scram ->
escramb -> scrambl -> scramble

def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert = 1 + ED(S1, S2[1:])
        delete =
        return min(substitute, insert, delete)
```
Beyond LCS: Edit Distance

```python
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        substitute = 1 + ED(S1[1:], S2[1:])
        insert = 1 + ED(S1, S2[1:])
        delete = 1 + ED(S1[1:], S2)
        return min(substitute, insert, delete)
```
Mergesort

msort([42, 3, 1, 5, 27, 8, 2, 7])

msort([42, 3, 1, 5]) / msort([27, 8, 2, 7])

merge([1, 3, 5, 42], [2, 7, 8, 27])

[ ]
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ \text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7]) \]

\[ \text{merge}([1, 3, 5, 42], [2, 7, 8, 27]) \]

\[ [1, \text{✓}] \]
Mergesort

\[ msort([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ msort([42, 3, 1, 5]) \mid msort([27, 8, 2, 7]) \]

\[ merge([1, 3, 5, 42], [2, 7, 8, 27]) \]

\[ [1, 2] \]
Mergesort

\[ \text{msort([42, 3, 1, 5, 27, 8, 2, 7])} \]

\[ \text{msort([42, 3, 1, 5])} \quad \text{msort([27, 8, 2, 7])} \]

\[ \text{merge([1, 3, 5, 42], [2, 7, 8, 27])} \]

\[ [1, 2, 3] \]
Mergesort

\[
\text{msort}([42, 3, 1, 5, 27, 8, 2, 7])
\]

\[
\text{msort}([42, 3, 1, 5]) / \text{msort}([27, 8, 2, 7])
\]

\[
\text{merge}([1, 3, 5, 42], [2, 7, 8, 27])
\]

\[
[1, 2, 3, 5]
\]
Mergesort

```
msort([42, 3, 1, 5, 27, 8, 2, 7])

msort([42, 3, 1, 5])  \msort([27, 8, 2, 7])

merge([1, 3, 5, 42], [2, 7, 8, 27])

[1, 2, 3, 5, 7]
```
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ \text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7]) \]

\[ \text{merge}([1, 3, 5, 42], [2, 7, 8, 27]) \]

\[ [1, 2, 3, 5, 7, 8] \]
Mergesort

\[
\text{msort(}[42, 3, 1, 5, 27, 8, 2, 7])
\]

\[
\text{msort(}[42, 3, 1, 5]) \text{ / } \text{msort(}[27, 8, 2, 7])
\]

\[
\text{merge(}[1, 3, 5, 42], [2, 7, 8, 27])
\]

\[
[1, 2, 3, 5, 7, 8, 27]
\]
Mergesort

\[ msort([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ msort([42, 3, 1, 5]) \] \[ msort([27, 8, 2, 7]) \]

\[ \text{merge([1, 3, 5, 42], [2, 7, 8, 27])} \]

\[ [1, 2, 3, 5, 7, 8, 27, 42] \]  \textbf{Done!}
Let’s try it out - and let’s not even make \( n \) a power of 2!

\[
\begin{align*}
\text{msort([42, 3, 1, 6, 5, 2, 7])} \\
\text{msort([42, 3, 1])} & \quad \text{msort([6, 5, 2, 7])}
\end{align*}
\]
\texttt{msort([42, 3, 1, 6, 5, 2, 7])}

\texttt{msort([42, 3, 1])} \quad \texttt{msort([6, 5, 2, 7])}

\texttt{msort([42])} \quad \texttt{msort([3, 1])} \quad \texttt{msort([6, 5])} \quad \texttt{msort([2, 7])}
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

msort([42])  msort([3, 1])

msort([6, 5, 2, 7])

msort([6, 5])  msort([2, 7])

msort([6])  msort([5])

msort([2])  msort([7])

msort([2])  msort([7])
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

msort([42])

msort([3, 1])

msort([3])

msort([1])

msort([6, 5, 2, 7])

msort([6, 5])

msort([2, 7])
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

\[
\begin{array}{c}
\text{msort([42])} \\
[42] \\
\text{msort([3, 1])} \\
[1, 3] \\
\text{msort([3])} \\
[3] \\
\text{msort([1])} \\
[1]
\end{array}
\]

msort([6, 5, 2, 7])

\[
\begin{array}{c}
\text{msort([6, 5])} \\
[6, 5] \\
\text{msort([2, 7])} \\
[2, 7]
\end{array}
\]
\texttt{msort([42, 3, 1, 6, 5, 2, 7])}

\begin{align*}
\text{msort([42, 3, 1])} & \quad \text{msort([6, 5, 2, 7])} \\
\text{msort([42])} & \quad \text{msort([3, 1])} \\
\text{msort([3])} & \quad \text{msort([1])} \\
\text{msort([1])} &
\end{align*}
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

msort([42])

msort([3, 1])

msort([6, 5, 2, 7])

msort([6, 5])

msort([2, 7])

msort([1, 3, 42])

msort([2, 5, 6, 7])
\[ \text{msort([42, 3, 1, 6, 5, 2, 7])} \]

\[ \text{msort([42, 3, 1])} \]
\[ \text{msort([6, 5, 2, 7])} \]
\[ \text{msort([42])} \]
\[ \text{msort([3, 1])} \]
\[ \text{msort([6, 5])} \]
\[ \text{msort([2, 7])} \]
How “Efficient” Is Mergesort?
How big a deal is this?

The Meder-O-Matic Supercomputer:
100 billion steps/second

\[ n^2 \text{ algorithm} \quad n \log_2 n \text{ algorithm} \]

\[ n = 10^8 \quad 11.5+ \text{ days} \]
How big a deal is this?

The Meder-O-Matic Supercomputer: 100 billion steps/second

- $n^2$ algorithm: $n = 10^8$, 11.5+ days
- $n \log_2 n$ algorithm: $n = 10^8$, 0.27 seconds