The Send-Ran-To-Spago Story

Spago

Website  Directions  Save

4.6 ★★★★★ 1,019 Google reviews

$$$$ · Californian restaurant

RESERVE A TABLE

Celeb chef Wolfgang Puck's buzzy, sleek bistro offers creative, locally sourced Californian cuisine.
This is Prof. Josh Hodas' Office!

Please Contribute to the "Send Ran to Spago" Fund

cans please
More grutoring on the weekend!

- Check the schedule! We now have CS 5 Black new Saturday hours and extra grutors during Sunday hours.
RNA Folding

- RNA is a string of nucleic acids A, U, C, G
- These strings fold so as to maximize matches (A-U and C-G) without any crossings (aka “pseudoknots”)

Score = 6

“Pseudo-knots” are “knot” permitted!

Homework 2, Problem 4
Evidence of a novel RNA secondary structure in the coding region of HIV-1 pol gene

Qi Wang,¹,² Ian Barr,¹,³ Feng Guo,¹,³ and Christopher Lee¹,²

Abstract

RNA secondary structures play several important roles in the human immunodeficiency virus (HIV) life cycle. To assess whether RNA secondary structure might affect the function of the HIV protease and reverse transcriptase genes, which are the main targets of anti-HIV drugs, we applied a series of different
A Phylogenetically Conserved Hairpin-Type 3’ Untranslated Region Pseudoknot Functions in Coronavirus RNA Replication

Gwyn D. Williams, Ruey-Yi Chang,† and David A. Brian *

ABSTRACT

Secondary and tertiary structures in the 3’ untranslated region (UTR) of plus-strand RNA viruses have been postulated to function as control elements in RNA replication, transcription, and translation. Here we describe a 54-nucleotide (nt) hairpin-type pseudoknot within the 288-nt 3’ UTR of the bovine coronavirus genome and show by mutational analysis of both stems that the pseudoknotted structure is required for the replication of a defective interfering RNA genome. The pseudoknot is phylogenetically conserved among coronaviruses both in location and in shape but only partially in nucleotide sequence, and evolutionary selection of pseudoknots is evident in the bovine coronavirus genome.
HW 2, Problem 4

```python
>>> getFold("CAGGCCUGUUAAC")
[6, [[1,  6], [2,  5], [3,  4], [7, 12], [8, 11], [9, 10]]]
```

- Drawn by a turtle!
Use-it-or-lose-it!

RNA = ACCCACCCUCCCUCC

Dr. Ruth Nussinov
Use-it-or-lose-it!

RNA = ACCCACCCUCCCUCC

Dr. Ruth Nussinov
Use-it-or-lose-it!

RNA = ACCCACCCUCCCUCC

Dr. Ruth Nussinov
Use-it-or-lose-it!

RNA = ACCCACCCUCCCUCC

Dr. Ruth Nussinov
Use-it-or-lose-it!

RNA = ACCCACCCUCCCUCC

Dr. Ruth Nussinov
Use-it-or-lose-it!

RNA = ACCCACCUCUCCUCC

Dr. Ruth Nussinov
Importance of the no-pseudoknots assumption

score = 1 + fold(RNA[1:8]) + fold(RNA[9:])

How are map and filter useful here?
LCS revisited...

```python
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

>>> LCS("i love spam", "look pam!")
6

>>> findLCS("i love spam", "look pam!")
[6, 'lo pam']
```
```python
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def findLCS(S1, S2):
    if S1 == "" or S2 == "": return
    elif S1[0] == S2[0]:
        gift = findLCS(S1[1:], S2[1:])
        else:
            option1 = findLCS(S1, S2[1:])
            option2 = findLCS(S1[1:], S2)
        if option1[0] > option2[0]:
            return option1
        else:
            return option2

>>> findLCS("i love spam", "look pam!")
[6, 'lo pam']
```
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def findLCS(S1, S2):
    if S1 == "" or S2 == "": return [0, ""]
    elif S1[0] == S2[0]:
        gift = findLCS(S1[1:], S2[1:])
        regift = [1 + gift[0], S1[0] + gift[1]]
        return regift
    else:
        option1 = findLCS(S1, S2[1:])
        option2 = findLCS(S1[1:], S2)
        if option1[0] > option2[0]:
            return option1
        else:
            return option2

>>> findLCS("i love spam", "look pam!")
[6, 'lo pam']
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:]):
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)
It works, but it’s slow!

LCS("spam", "qlug")

LCS("spam", "lug")  LCS("pam", "qlug")

LCS("spam", "ug")  LCS("pam", "lug")  LCS("am", "qlug")

LCS("spam", "ug")  LCS("pam", "lug")  LCS("am", "qlug")
Jumping off paper...

1 sheet of regular paper

<table>
<thead>
<tr>
<th>Folds (n)</th>
<th>Layers (2^n)</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 paper thick</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 sheets thick</td>
</tr>
<tr>
<td>... skip a few ...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>(2^7)</td>
<td>notebook</td>
</tr>
</tbody>
</table>
Jumping off paper...

1 sheet of regular paper

<table>
<thead>
<tr>
<th>Folds (n)</th>
<th>Layers ($2^n$)</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 paper thick</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 sheets thick</td>
</tr>
<tr>
<td>... skip a few ...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>$2^7$</td>
<td>notebook</td>
</tr>
<tr>
<td>13</td>
<td>$2^{13}$</td>
<td>1 meter</td>
</tr>
</tbody>
</table>
Jumping off paper...

1 sheet of regular paper

<table>
<thead>
<tr>
<th>Folds (n)</th>
<th>Layers ($2^n$)</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 paper thick</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 sheets thick</td>
</tr>
<tr>
<td>... skip a few ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$2^7$</td>
<td>notebook</td>
</tr>
<tr>
<td>13</td>
<td>$2^{13}$</td>
<td>1 meter</td>
</tr>
<tr>
<td>17</td>
<td>$2^{17}$</td>
<td>2 story house</td>
</tr>
<tr>
<td>20</td>
<td>$2^{20}$</td>
<td>1/4 Sears Tower</td>
</tr>
</tbody>
</table>
Jumping off paper...

1 sheet of regular paper

<table>
<thead>
<tr>
<th>Folds (n)</th>
<th>Layers (2^n)</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 paper thick</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 sheets thick</td>
</tr>
<tr>
<td>... skip a few ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>(2^7)</td>
<td>notebook</td>
</tr>
<tr>
<td>13</td>
<td>(2^{13})</td>
<td>1 meter</td>
</tr>
<tr>
<td>17</td>
<td>(2^{17})</td>
<td>2 story house</td>
</tr>
<tr>
<td>20</td>
<td>(2^{20})</td>
<td>1/4 Sears Tower</td>
</tr>
<tr>
<td>30</td>
<td>(2^{30})</td>
<td>Outer limit of atmosphere</td>
</tr>
</tbody>
</table>
## Jumping off paper...

1 sheet of regular paper

<table>
<thead>
<tr>
<th>Folds ($n$)</th>
<th>Layers ($2^n$)</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 paper thick</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 sheets thick</td>
</tr>
<tr>
<td>... skip a few ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$2^7$</td>
<td>notebook</td>
</tr>
<tr>
<td>13</td>
<td>$2^{13}$</td>
<td>1 meter</td>
</tr>
<tr>
<td>17</td>
<td>$2^{17}$</td>
<td>2 story house</td>
</tr>
<tr>
<td>20</td>
<td>$2^{20}$</td>
<td>1/4 Sears Tower</td>
</tr>
<tr>
<td>30</td>
<td>$2^{30}$</td>
<td>Outer limit of atmosphere</td>
</tr>
<tr>
<td>50</td>
<td>$2^{50}$</td>
<td>Distance to sun</td>
</tr>
</tbody>
</table>
$2^n$ is really bad!

- $n = 20$: 0.001 seconds
- $n = 50$: 
- $n=100$: 

That’s fast!

1 billion operations per second!
$2^n$ is really bad!

- $n = 20$: 0.001 seconds
- $n = 50$: 13 days
- $n = 100$: Ouch!

Ran-O-Matic

1 billion operations per second!
$2^n$ is really bad!

$n = 20$: 0.001 seconds

$n = 50$: 13 days

$n = 100$: $4 \times 10^{13}$ years

Aye caramba!
There’s repeated work here!

LCS("spam", "qlug")

LCS("spam", "lug")

LCS("spam", "ug")

LCS("pam", "lug")

LCS("pam", "qlug")

LCS("pam", "lug")

LCS("am", "qlug")
Dictionaries!

>>> D = {}

>>> D["spam"] = "a health food product"

>>> D[42] = "an important number"

>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]

>>> D
{
    42: 'an important number',
    'ran': ['ran@cs.hmc.edu', 'olin 1253'],
    'spam': 'a health food product'
}

>>> D[42]
'an important number'

>>> 42 in D
True

>>> 43 in D
False

>>> "ran" in D
False

>>> D = {}

"..."
Dictionaries!

```python
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
>>> D
{42: 'an important number', 'ran': ['ran@cs.hmc.edu', 'olin 1253'], 'spam': 'a health food product'}
>>> D[42]
'an important number'
>>> 42 in D
True
>>> 43 in D
False
>>> "ran" in D
False
```

the key the value
Dictionaries!

the key the value

```python
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
```
Dictionaries!

the key the value

>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
>>> D
{42: 'an important number', 'ran': ['ran@cs.hmc.edu', 'olin 1253'], 'spam': 'a health food product'}
Dictionaries!

```python
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
>>> D
{42: 'an important number', 'ran': ['ran@cs.hmc.edu', 'olin 1253'], 'spam': 'a health food product'}
>>> D[42]
'an important number'
```
Dictionaries!

the *key*  the *value*

```python
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
>>> D
{42: 'an important number', 'ran': ['ran@cs.hmc.edu', 'olin 1253'], 'spam': 'a health food product'}
>>> D[42]
'an important number'
>>> 42 in D
True
>>> 43 in D
False
>>> "abba" in D
False
```
Dictionaries!

```
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["ran"] = ["ran@cs.hmc.edu", "olin 1253"]
>>> D

{42: 'an important number', 'ran': ['ran@cs.hmc.edu', 'olin 1253'], 'spam': 'a health food product'}
```

```
>>> D[42]
'an important number'
```

```
>>> 42 in D
True
```

```
>>> 43 in D
False
```

```
>>> "abba" in D
False
```

```
>>> D["abba"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
```
A minor technicality…

```python
>>> D = {}
>>> D[42, 47] = "^)"
Bad idea! Dictionaries should map single keys to values (Python 3 doesn’t fully barf)
>>> D[ [42, 7] ] = "^)"
Barf!!! TypeError: unhashable type: 'list'
>>> D[ (42, 7) ] = "^)" # tuples are OK!
>>> D[ (42, 7) ]
"^)"
>>> (42, 7) in D
True
```
Tuples are \textit{(almost)} just like lists…

```python
>>> L = (42, 47, 5, "spam")
>>> L[3]
"spam"
```

```python
>>> L[1:]
(47, 5, "spam")
```

```python
>>> M = (1, 2, 3) + (4, 5, 6)
>>> M
(1, 2, 3, 4, 5, 6)
```

```python
>>> L[1] = "foobar"
```

\textbf{Barf!} TypeError: 'tuple' object does not support item assignment

This is the one place that lists and tuples differ!
It works, but it’s slow!

LCS("spam", "qlug")

LCS("spam", "lug")

LCS("pam", "qlug")

LCS("spam", "ug")

LCS("pam", "lug")

LCS("pam", "lug")

LCS("am", "qlug")
```python
def LCS(S1, S2):
    if S1 == "" or S2 == "":
        return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def fastLCS(S1, S2, memo):
    if (S1, S2) in memo:
        return memo[(S1, S2)]
    elif S1 == "" or S2 == "":
        return 0
    elif S1[0] == S2[0]:
        result = 1 + fastLCS(S1[1:], S2[1:], memo)
        memo[(S1, S2)] = result
        return result
    else:
        option1 = fastLCS(S1, S2[1:], memo)
        option2 = fastLCS(S1[1:], S2, memo)
        result = max(option1, option2)
        memo[(S1, S2)] = result
        return result

>>> fastLCS("sam I am", "spam sandwich", {})
```

Original version

Memo-ized version

Demo!!
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:]),
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def fastLCS(S1, S2, memo):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        result = 1 + fastLCS(S1[1:], S2[1:], memo)
        memo[(S1, S2)] = result
        return result
    else:
        option1 = fastLCS(S1, S2[1:], memo)
        option2 = fastLCS(S1[1:], S2, memo)
        result = max(option1, option2)
        memo[(S1, S2)] = result
        return result

>>> fastLCS("sam I am", "spam sandwich", {})
```python
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:])
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def fastLCS(S1, S2, memo):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        result = 1 + fastLCS(S1[1:], S2[1:], memo)
        memo[(S1, S2)] = result
        return result
    else:
        option1 = fastLCS(S1, S2[1:], memo)
        option2 = fastLCS(S1[1:], S2, memo)
        result = max(option1, option2)
        memo[(S1, S2)] = result
        return result

>>> fastLCS("sam I am", "spam sandwich", {})```
```python
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        return 1 + LCS(S1[1:], S2[1:]),
    else:
        option1 = LCS(S1, S2[1:])
        option2 = LCS(S1[1:], S2)
        return max(option1, option2)

def fastLCS(S1, S2, memo):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        result = 1 + fastLCS(S1[1:], S2[1:], memo)
        memo[(S1, S2)] = result
        return result
    else:
        option1 = fastLCS(S1, S2[1:], memo)
        option2 = fastLCS(S1[1:], S2, memo)
        result = max(option1, option2)
        memo[(S1, S2)] = result
        return result
```

```python
>>> fastLCS("sam I am", "spam sandwich", {})
```
Care packages + memoization!

```python
def fastfindLCS(S1, S2, memo):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "": return [0, ""]
    elif S1[0] == S2[0]:
        gift = fastfindLCS(S1[1:], S2[1:], memo)
        regift = [1 + gift[0], S1[0] + gift[1]]
        memo[(S1, S2)] = regift
    return regift

else:
    option1 = fastfindLCS(S1, S2[1:], memo)
    option2 = fastfindLCS(S1[1:], S2, memo)
    if option1[0] > option2[0]:
        memo[(S1, S2)] = option1
    return option1

else:
    memo[(S1, S2)] = option2
    return option2

>>> fastfindLCS("sam I am", "spam sandwich", {})
```

Demo!!
How dictionaries work (and why keys must be immutable)

```python
>>> D = {}
>>> D[ (1, 2, 3) ] = "how cool is this?"
>>> D[ (1, 2, 3) ]
'how cool is this?'
>>> D[ [1, 2, 3] ] = "and this is cool too!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: unhashable type: 'list'
```

Aka, “Barf!”
How dictionaries work (and why keys must be immutable)

```python
>>> D[ (1, 2, 3) ] = "how cool is this?"
```

![Diagram showing a hash function mapping a tuple (1, 2, 3) to a hash value 424242, which is stored in a memory location.]
How dictionaries work (and why keys must be immutable)

```python
>>> D[ (1, 2, 3) ] = "how cool is this?"
```

![Diagram showing the process of hashing a tuple key into memory]
How dictionaries work (and why keys must be immutable)

```python
>>> D[ (1, 2, 3) ] = "how cool is this?"
```

Your computer’s memory

Hash function

424242

424242

"how cool is this?"

4294967291
4294967292
4294967293
4294967294
4294967295
How dictionaries work (and why keys must be immutable)

```python
>>> D[(1, 2, 3)] = "how cool is this?"
```

Hash function

```
>>> D[(1, 2, 3)]
```

Your computer’s memory
How dictionaries work (and why keys must be immutable)

```python
>>> L = [1, 2, 3]
>>> D[L] = "how cool is this?"
```

Your computer’s memory

```python
>>> L[0] = "malice"
>>> L
["malice", 2, 3]
>>> D[L]
```

Hash function

424242

```
4294967291
4294967292
4294967293
4294967294
4294967295
```

"how cool is this?"
How dictionaries work (and why keys must be immutable)

```python
>>> L = [1, 2, 3]
>>> D[L] = "how cool is this?"
```

Hash function

```
424242
```

```
>>> L[0] = "malice"
>>> L
["malice", 2, 3]
>>> D[L]
```

Hash function

```
626364
```

Your computer’s memory

```
0
1
2
3
4
5
```

```
4294967291
4294967292
4294967293
4294967294
4294967295
```

"how cool is this?"