Jan 2021 | © Guven

Divide and Conquer Algorithm Examples

This notebook demonstrates,

1. Examples of Divide and Conquer Algorithms

Signal Interweaving

The problem is defined as *untangling* a superposition of two known signals.

In a common medium, such as Radio Frequency, given two sources emitting a short sequence of `0`s and `1`s over and over, and the received signal is simply an interweaving of these two emissions, with nothing extra added in. The short sequence emitted by each source is known. The problem is deciding if a received signal is the interwoven of these two sent signals, or not. The objective is to develop an efficient algorithm that takes strings `R`, `S1`, and `S2` and decides if `R` is an interleaving of `S1` and `S2`.

`S1` and `S2` are short signals and they are repeated with certain signal periods. The periods are not necessarily same or constant.

In [1]:
```
# Generate a test signal R
from random import randint

def rotate(_s):
    return _s[1:] + [_s[0]]

S1 = [1,3,2]
S2 = [0,1,3,3,1]
SIGNAL_LEN = 100

dbg_info = []

# ratio defines the amount of signal s2
def generate_signal(_n, _s1, _s2, ratio=0.4):
    global dbg_info
    r=[]
    for _ in range(_n):
        if randint(0,100)>100*ratio:
            r += [_s1[0]]
            _s1 = rotate(_s1)
            dbg_info += [0]
        else:
            r += [_s2[0]]
            _s2 = rotate(_s2)
            dbg_info += [1]
    return r


R = generate_signal(SIGNAL_LEN, S1, S2, 0.3)

print(f'{"".join(map(str,S1))}')
print(f'{"".join(map(str,S2))}')
```

```
132
01331
```

In [2]:
```
from IPython.display import Markdown

span1 = '<span style=\"font-family:Ariel; font-size:12pt; font-weigh
t:bold; \">'
span2 = '<span style=\"font-family:Courier; font-size:10pt; \">'
dbg = ''
for i in range(SIGNAL_LEN):
    dbg = dbg + (span1 if dbg_info[i]==1 else span2) + str(R[i]) +
'</span>'

display (Markdown(dbg))
```

In [3]:
```
# The algorithm finds if the signal has these two S1 and S2 interweav
ed
def interweaved(_S1, _S2, _R):
    """ DC algorithm to decide interweaved
    Args:
        A list: signal 1
        B list: signal 2
        C list: interweaved signal with repetitions of S1 and S2 in i
t
    Returns:
        boolean: True if R has S1 and S2 or False
    """
    if len(_R) == 0:  # Checked all the elements of R
        return True
    elif _S1[0]!=_R[0] and _S2[0]!=_R[0]:
        return False
    else:
        return ((_S1[0]==_R[0] and interweaved(rotate(_S1), _S2, _R[1
:])) or
                (_S2[0]==_R[0] and interweaved(_S1, rotate(_S2), _R[1
:])))
```

In [4]:
```
# Test cases
print(interweaved(S1, S2, R))

# Add S1
print(interweaved(S1, S2, R[:5]+S1+R[5:]))
print(interweaved(S1, S2, R[:6]+S1+R[6:]))

# Add S2
print(interweaved(S1, S2, R[:5]+S2+R[5:]))
print(interweaved(S1, S2, R[:7]+S2+R[7:]))
```

```
True
False
False
False
False
```

In [5]:
```
print(interweaved(S1, S2, S1+R))
print(interweaved(S1, S2, R+S1))

print(interweaved(S1, S2, S2+R))
print(interweaved(S1, S2, S2[:-1]+R))
print(interweaved(S1, S2, R+S2[1:]))

print(interweaved(S1, S2, R[:10]+[3]+R[10:]))
print(interweaved(S1, S2, R[:11]+[3]+R[11:]))
```

```
True
True
True
False
False
False
False
```

Complexity

Loose bound: $\mathrm{T}(n) = \mathrm{T}(n-1) + \mathrm{T}(n-1)$ and complexity = $\mathcal{O}(2^n)$

Question: Why above bound is loose?

Question: Is the provided algorithm a divide and conquer algorithm or rather a brute force search algorithm?

Tighter bound: Complexity = $\mathcal{O}(|\mathrm{S1}||\mathrm{S2}|n)$

Question: What if $n < |\mathrm{S1}|$ or $n < |\mathrm{S2}|$?