Jan 2021 | © Guven

# Longest Common Subsequence (LCS) Algorithm

This notebook demonstrates,

1. An example implementation of LCS algorithm
2. `numpy` is used for matrices and nice indexing
3. Code comments explain how the algorithm/program works
4. `c` keeps the scores, filled with $0$ at the beginning
5. `b` keeps the backtracking and will have symbols eventually
6. `get_alignment` function examines the b matrix and pulls the actual alignment
7. The example uses `matplotlib.animation` library to single-step the algorithm iterations

---

**Note:** `jupyter notebook --to html dynamicprogramming_ep.ipynb` is the command to create the HTML from the ipynb format (includes the animations).

Note that `--to html_embed` disables the animation for some unknown reason to me.

---

```
In [1]: import numpy as np

        # Backtracking symbols
        SKIPX, SKIPY, ADDXY = '|', '-', '+'

        def LCS(X, Y):  # find the Longest Common Subsequence
            M, N = len(X), len(Y)
            c = np.full((N+1, M+1), 0, dtype=int)  # score matrix
            b = np.full((N+1, M+1), ' ', dtype=str)  # backtracking
            #
            for i in range(1, N+1):  # Fill the boundary
                c[i,0], b[i,0] = 0, SKIPX
            #
            for j in range(1,M+1):  # Fill the boundary
                c[0,j], b[0,j] = 0, SKIPY
            #
            for i in range(1, N+1):  # for every row, i y-axis
                for j in range(1, M+1):  # every column, j x-axis
                    if Y[i-1] == X[j-1]:
                        c[i,j] = c[i-1,j-1] + 1  # take X[i] and Y[j] for LCS
                        b[i,j] = ADDXY
                    elif c[i-1,j] >= c[i,j-1]:  # the equality can generate d
        ifferent solutions
                        c[i,j], b[i,j] = c[i-1,j], SKIPX
                    else:
                        c[i,j], b[i,j] = c[i,j-1], SKIPY
            #
            return c, b

        def get_alignment(b: np.array, X: str, Y: str) -> (str,str):
            # start from the lower right corner - global alignment
            (N,M) = b.shape
            s1, s2 = [], []  # s1 corresponds to X, s2 corresponds to Y
            i, j = N-1, M-1
            #
            while i>0 or j>0:  # i y-axis, j x-axis
                if b[i,j] == ADDXY:
                    s1, s2 = [X[j-1]]+s1, [Y[i-1]]+s2
                    i=i-1
                    j=j-1
                elif b[i,j] == SKIPX:
                    s1, s2 = ['-']+s1, [Y[i-1]]+s2  # '-' alignment skip
                    i=i-1
                elif b[i,j] == SKIPY:
                    s1, s2 = [X[j-1]]+s1, ['-']+s2  # '-' alignment skip
                    j=j-1
            #
            if isinstance(s1[0], int) or isinstance(s2[0], int):
                s1, s2 = ''.join([str(j) for j in s1]), ''.join([str(i) for i
        in s2])
            else:
                s1, s2 = ''.join(s1), ''.join(s2)
            #
            return s1, s2

        def get_backtracking(_b, _c):
```

```
    i, j = _c.shape[0]-1, _c.shape[1]-1
    clist = [(i+1,j+1)]
    #
    while i>=0 and j>=0:
        if _b[i,j] == ADDXY:
            i = i-1
            j = j-1
        elif _b[i,j] == SKIPY:
            j = j-1
        elif _b[i,j] == SKIPX:
            i = i-1
        else:  # avoid infinite loop
            break
        #
        clist += [(i+1,j+1)]
    #
    return clist[:-1]  # remove the last added (0,0)

def vals_dpmatrix(_X, _Y, _c, _b):
    N, M = _b.shape[0], _b.shape[1]
    cols = [' ', ' '] + list(_X)
    vals = [[y] + li.tolist() for y, li in zip([' ']+list(_Y),_b)]
    vals = [[x.replace(SKIPX,r'$\uparrow$') for x in li] for li in vals]
    vals = [[x.replace(SKIPY,r'$\leftarrow$') for x in li] for li in vals]
    vals = [[x.replace(ADDXY,r'$\nwarrow$') for x in li] for li in vals]
    #
    for i in range(0,N):
        for j in range(1,M+1):
            vals[i][j] = r'$_'+str(_c[i,j-1])+'$' + ' ' + vals[i][j]
    #
    return cols, vals
```

In [2]:

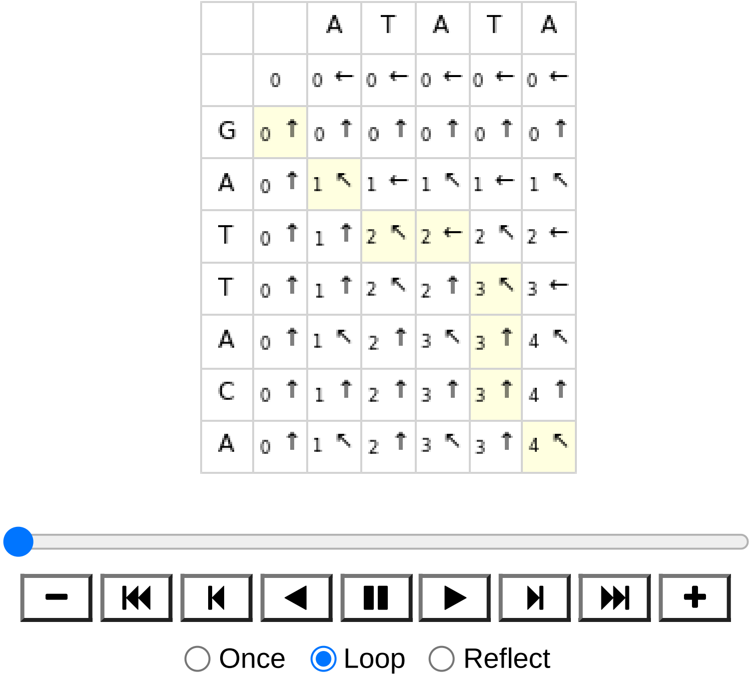
```
Y='GATTACA'; X='ATATA'  # Strings can be indexed like lists
c, b = LCS(X, Y)
#
b2 = np.empty((len(Y)+2,len(X)+2),dtype=str); b2[:] = ' '
b2[1:,1:] = b; b2[2:,0] = list(Y); b2[0,2:] = list(X)
print(b2)
print(c)
#
s1, s2 = get_alignment(b, X, Y)
print(s1)
print(s2)
```

```
[[' ' ' ' 'A' 'T' 'A' 'T' 'A']
 [' ' ' ' '-' '-' '-' '-' '-']
 ['G' '|' '|' '|' '|' '|' '|']
 ['A' '|' '+' '-' '+' '-' '+']
 ['T' '|' '|' '+' '-' '+' '-']
 ['T' '|' '|' '+' '|' '+' '-']
 ['A' '|' '+' '|' '+' '|' '+']
 ['C' '|' '|' '|' '|' '|' '|']
 ['A' '|' '+' '|' '+' '|' '+']]
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 1 1 1 1 1]
 [0 1 2 2 2 2]
 [0 1 2 2 3 3]
 [0 1 2 3 3 4]
 [0 1 2 3 3 4]
 [0 1 2 3 3 4]]
-ATAT--A
GAT-TACA
```

In [3]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

clist = get_backtracking(b, c)
cols, vals = vals_dpmatrix(X, Y, c, b)

fig, ax = plt.subplots()

def plot_dpmatrix(_ax, _cols, _vals, clist=None, vismax=0):
    _ax.clear()
    #
    Col_w, Row_h, Font_size, M, N = 0.08, 0.12, 12, len(_cols), len(_
vals)
    t = _ax.table(cellText=_vals, colLabels=_cols, colWidths=[Col_w]*
M, cellLoc='center', loc='center')
    #
    cellDict = t.get_celld()
    for i in range(0,N+1):  # i y-axis, j x-axis
        for j in range(0,M):
            if vismax != 0 and j>0 and (i-1)*(M-1)+(j) > vismax:
                cellDict[(i,j)].set_visible(False)
            cellDict[(i,j)].set_height(Row_h)
            cellDict[(i,j)].set_edgecolor('lightgray')
            if clist is not None and (i,j) in clist:
                cellDict[(i,j)].set_facecolor('lightyellow')
    #
    t.auto_set_font_size(False)
    t.set_fontsize(Font_size)
    #
    plt.tick_params(axis='x', which='both', bottom=False, top=False,
labelbottom=False)
    plt.tick_params(axis='y', which='both', right=False, left=False,
labelleft=False)
    for pos in ['right','top','bottom','left']:
        plt.gca().spines[pos].set_visible(False)
    #
    plt.close()
    #
    return t

def animate_dpmatrix(i):
    return plot_dpmatrix(ax, cols, vals, clist=clist, vismax=i)

ani_dp = animation.FuncAnimation(fig, animate_dpmatrix, interval=1000
, frames=(len(X)+1)*(len(Y)+1)+1)
HTML(ani_dp.to_jshtml())
```

Out[3]:



|  |  | A | T | A | T | A |
|---|---|---|---|---|---|---|
|  | 0 | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← |
| G | 0 ↑ | 0 ↑ | 0 ↑ | 0 ↑ | 0 ↑ | 0 ↑ |
| A | 0 ↑ | 1 ↖ | 1 ← | 1 ↖ | 1 ← | 1 ↖ |
| T | 0 ↑ | 1 ↑ | 2 ↖ | 2 ← | 2 ↖ | 2 ← |
| T | 0 ↑ | 1 ↑ | 2 ↖ | 2 ↑ | 3 ↖ | 3 ← |
| A | 0 ↑ | 1 ↖ | 2 ↑ | 3 ↖ | 3 ↑ | 4 ↖ |
| C | 0 ↑ | 1 ↑ | 2 ↑ | 3 ↑ | 3 ↑ | 4 ↑ |
| A | 0 ↑ | 1 ↖ | 2 ↑ | 3 ↖ | 3 ↑ | 4 ↖ |

○ Once ● Loop ○ Reflect

In [ ]:

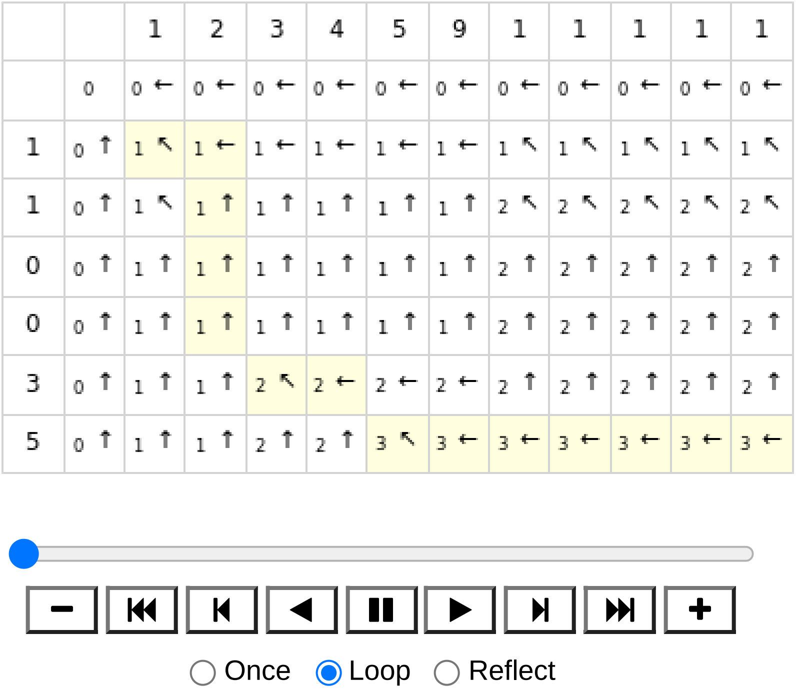
In [4]:

```
Y2='110035'
X2='12345911111'
#
c2, b2 = LCS(X2, Y2)
```

In [5]:

```
clist2 = get_backtracking(b2, c2)
cols2, vals2 = vals_dpmatrix(X2, Y2, c2, b2)

fig2, ax2 = plt.subplots()
fig2.tight_layout()
ax2.autoscale(enable=True)

def animate_dpmatrix2(i):
    return plot_dpmatrix(ax2, cols2, vals2, clist=clist2, vismax=i)

ani_dp2 = animation.FuncAnimation(fig2, animate_dpmatrix2, interval=1000, frames=(len(X2)+1)*(len(Y2)+1)+1)
HTML(ani_dp2.to_jshtml())
```

Out[5]:



|  |  | 1 | 2 | 3 | 4 | 5 | 9 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 0 | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← | 0 ← |
| 1 | 0 ↑ | 1 ↖ | 1 ← | 1 ← | 1 ← | 1 ← | 1 ← | 1 ↖ | 1 ↖ | 1 ↖ | 1 ↖ | 1 ↖ |
| 1 | 0 ↑ | 1 ↖ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 2 ↖ | 2 ↖ | 2 ↖ | 2 ↖ | 2 ↖ |
| 0 | 0 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ |
| 0 | 0 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 1 ↑ | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ |
| 3 | 0 ↑ | 1 ↑ | 1 ↑ | 2 ↖ | 2 ← | 2 ← | 2 ← | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ | 2 ↑ |
| 5 | 0 ↑ | 1 ↑ | 1 ↑ | 2 ↑ | 2 ↑ | 3 ↖ | 3 ← | 3 ← | 3 ← | 3 ← | 3 ← | 3 ← |

○ Once ● Loop ○ Reflect

## Exercises

Study the code above for Longest Common Subsequence `LCS`.

Most of the provided functions are for extracting the backtracking and actually printing the alignment, including the function `plot_dpmatrix` which creates the matrix to be animated by `animate_dpmatrix` and `HTML(ani_dp.to_jshtml())`.