Algorithm

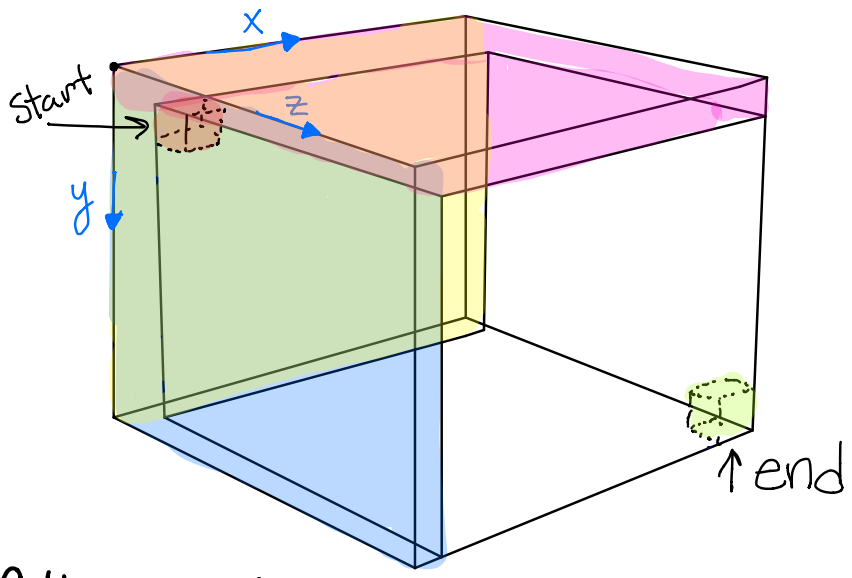
AlignString(Xs, Ys, Zs):

```
A = [len(xs)][len(Ys)][len(Zs)] # Create empty 3d array
A[:][:][0] = twoString (Xs, Ys)   # create 3 2d String alignment
A[0][:][:] = twoString ( Zs,Ys)   # arrays for each pair of x y z
A[:][0][:]= twoString ( Xs,Zs)    # insert them into the sides of A
# Build 3d matrix A
for z from 1 to len(Zs)-1:
    for Y from 1 to len(Ys)-1:
        for X from 1 to len(Xs)-1:#determine which adjacent element is min
            Cmin = min(A[X-1:X][Y-1:Y][Z-1:Z])
            Xmin, Ymin, Zmin = find( A[X-1:X][Y-1:Y][Z-1:Z]==Cmin) #index of min
            StringAlign = []                                        # adj element to
             if Xmin == X:                                          #A(X,Y,Z)
                StringAlign.append(Null)
             else:
                StringAlign.append(XS[X])

             if Ymin == Y:
                StringAlign.append(null)
             else:
                StringAlign.append(YS[Y])

             if Zmin == Z:
                StringAlign.append(null)
             else:
                String Align.append(ZS[Z])
             CurrCost = 0
             # find cost of aligning set StringAlign
              if (StringAlign[0] != StringAlign[1] ):
                        CurrCost ++
              if (String Align[1] != StringAlign[2]):
                        Currcost ++
              if (StringAlign[0] != StringAlign[2]):
                        Currcost ++
             A[X, Y, Z] = Cmin + CurrCost

# now optimal cost matrix is created
 X= len(Xs) -1
 Y= len(Ys) -1
 Z = len(Zs)-1
 String= ""
 While !( X==0 and Y==0 and Z==0):
     C = Cmin( A[X-1:X][Y-1:Y][Z-1:Z])
     Xmin, Ymin, Zmin = find(A[X-1:X][Y-1:Y][Z-1:Z]==C)
     alignmentSet= []
```

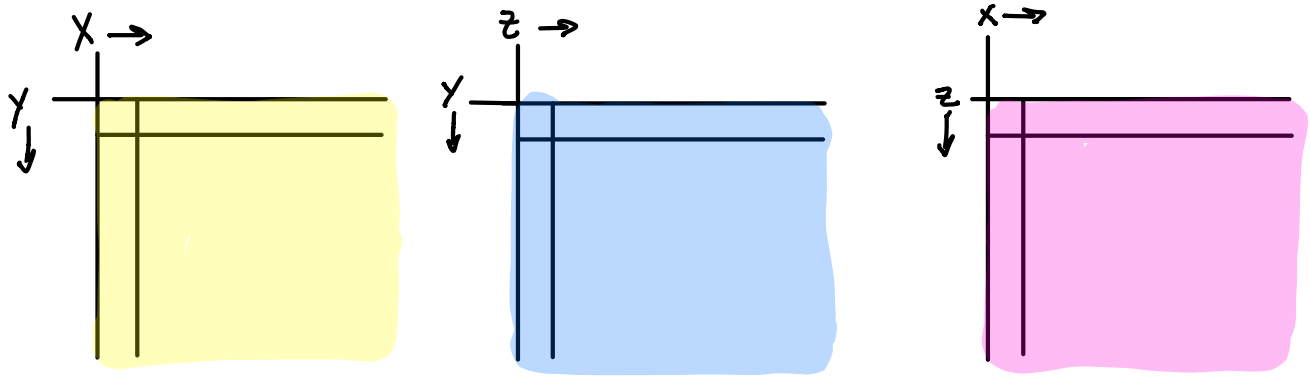
```
if (Xmin != X):
    alignmentSet.append(Xs[X])
if (Ymin != Y):
    alignmentSet.append(Ys[Y])
if (Zmin != Z):
    alignmentSet.append(Zs[Z])

if len(alignmentset == 1):   # ops delete one letter from one String
# no change to String
elif len(alignmentset == 2):
    String = String + alignmentSet[0]
elif len(alignmentSet == 3):
    if (alignmentSet[0] == alignmentset[1]):
        String = String + alignmentset[0]
    elif (alignmentSet[1] == alignmentSet[2]):
        String = String + alignmentSet[1]
    elif (alignmentset[2] == alignmentset[0]):
        String = String + alignmentSet[2]
    else: # if all 3 char dont match pick first
        String = String + alignmentset[0]

X = Xmin
Y = Ymin
Z = Zmin

Return String
```

Dynamic Programming Algorithm:

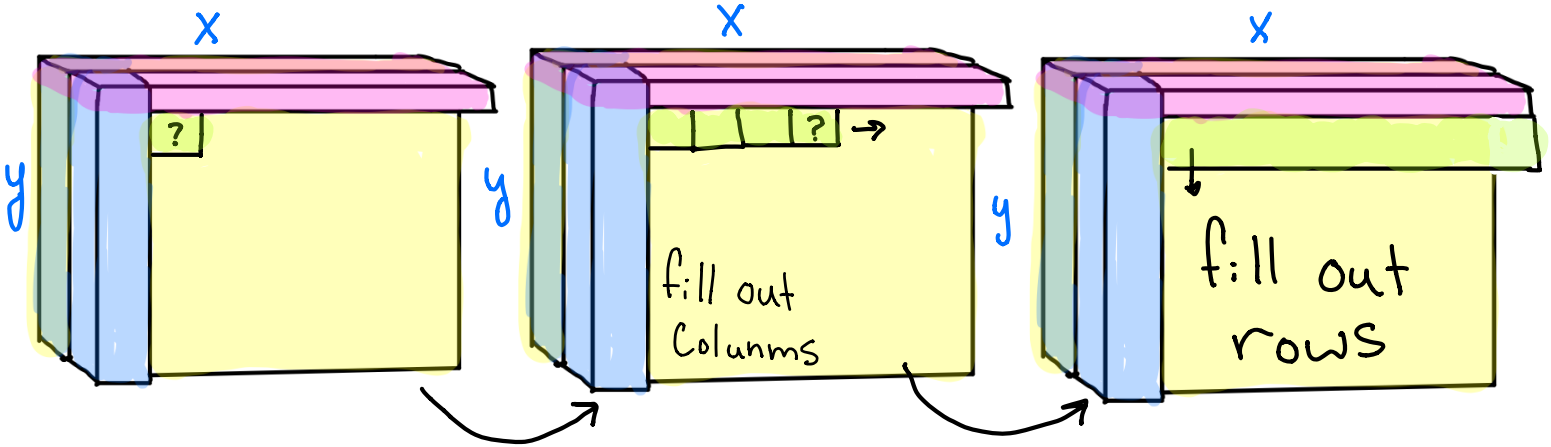
Step 1: fill out 3d matrix A



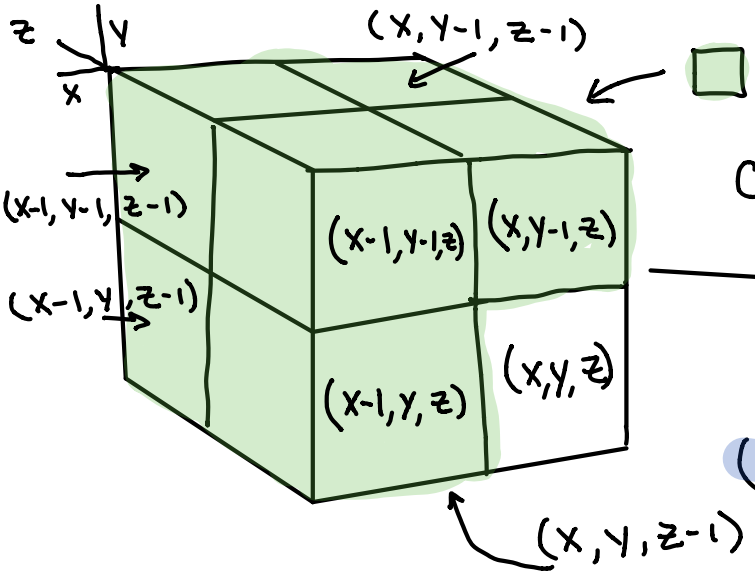
Step 1a: fill out known sides by using 2 string alignment and creating 3 2d matrices



Step 1b: fill out layer in matrix

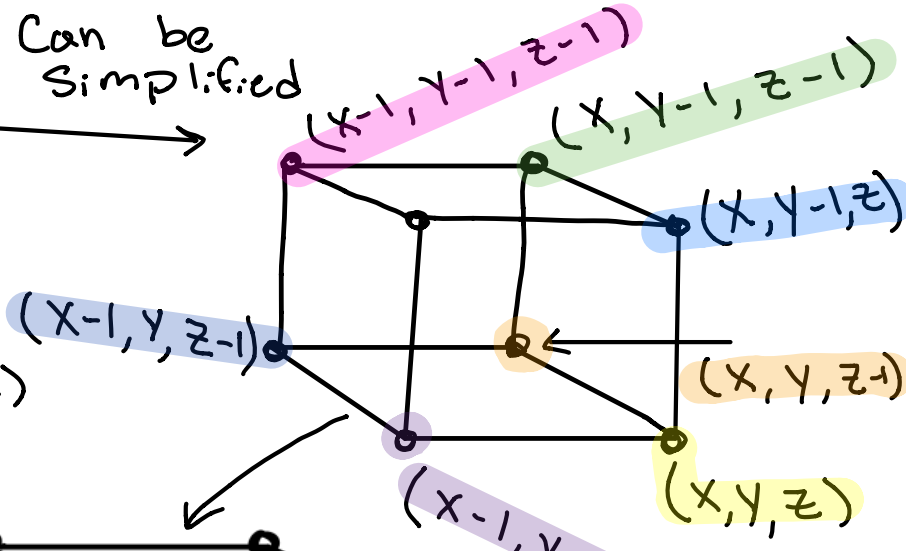


How to fill out element (x,y,z)



Known Costs

Can be simplified

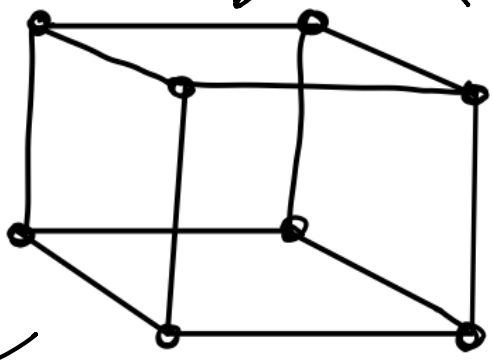


Determine Cases:

ex (x-1, y, z-1)

(x-1, y, z-1) → (x, y, z)

a b



Cases: ① $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ② $\begin{pmatrix} x \\ y \\ - \end{pmatrix}$ ③ $\begin{pmatrix} x \\ - \\ z \end{pmatrix}$ ④ $\begin{pmatrix} - \\ x \\ y \end{pmatrix}$ ⑤ $\begin{pmatrix} x \\ - \\ - \end{pmatrix}$ ⑥ $\begin{pmatrix} - \\ y \\ - \end{pmatrix}$ ⑦ $\begin{pmatrix} - \\ - \\ z \end{pmatrix}$

Case a→b:

I. already determined: y → represented by "-"

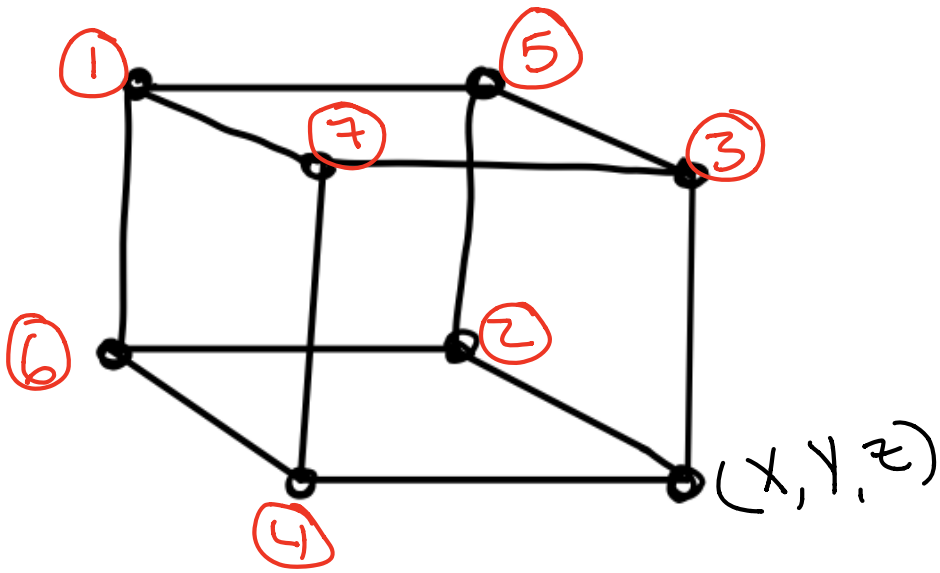
II. Changing indices / needs to be determined: x, z → represented by "x" "z"

Therefore

(x-1, y, z-1) → (x, y, z) is $\begin{pmatrix} x \\ - \\ y \end{pmatrix}$

↳ do it with all known elements adjacent to (x, y, z)

adjacent elements represented by cases



Determine A(X,Y,Z)

- Choose min out of adjacent known elemnts around it
- then based on min's case it represents determine Cost of (x,y,z)

TotalCost: A[X][Y][Z] = Min adj Cost + element Cost

element cost = determined by the min adjacent costs case

Cost by Case

Cases $\begin{pmatrix}1\\2\\3\end{pmatrix}$ $\begin{pmatrix}X\\Y\\Z\end{pmatrix}$ 1. $\begin{pmatrix}X\\Y\\-\end{pmatrix}$ 2. $\begin{pmatrix}X\\-\\Z\end{pmatrix}$ 3. $\begin{pmatrix}-\\X\\Y\end{pmatrix}$ 4. $\begin{pmatrix}X\\-\\-\end{pmatrix}$ 5. $\begin{pmatrix}-\\Y\\-\end{pmatrix}$ 6. $\begin{pmatrix}-\\-\\Z\end{pmatrix}$ 7.

Cost:

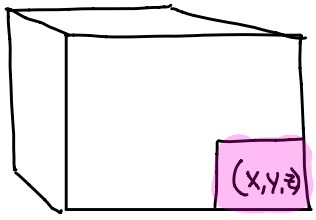
alignment	1 OP	1 C	2 OP	2 C	3 OP	3 C	4 OP	4 C	5 OP	5 C	6 OP	6 C	7 OP	7 C
1→2	*S	0/1	*S	1/0	I	1	D	1	I	1	D	1	X	0
2→3	*S	0/1	I	1	D	1	*S	1/0	X	0	I	1	D	1
3→1	*S	0/1	D	1	*S	0/1	I	1	D	1	X	0	I	1

*S if $\begin{cases} i = j & \text{then } S_{cost} = 0 \\ i \neq j & \text{then } S_{cost} = 1 \end{cases}$

Repeat 1. till full 3d matrix is filled out

2. Use filled out matrix to find Optimal alignment String

- reverse back through matrix by going to elements adj minimum after each step



(x,y z) we know the set → is aligned.

$X_0 \ldots X_x$
$Y_0 \ldots Y_y$
$Z_0 \ldots Z_z$

and optimally was aligned from element adjacent with the min cost.

if (X-1, Y-1, Z-1) is the min adjacent Cost then

$X_0 \ldots X_{x-1} \mid X_x$
$Y_0 \ldots Y_{y-1} \mid Y_y$
$Z_0 \ldots Z_{z-1} \mid Z_z$

was optimally aligned to produce set

or if (X-1, Y-1, Z) is the min adjant

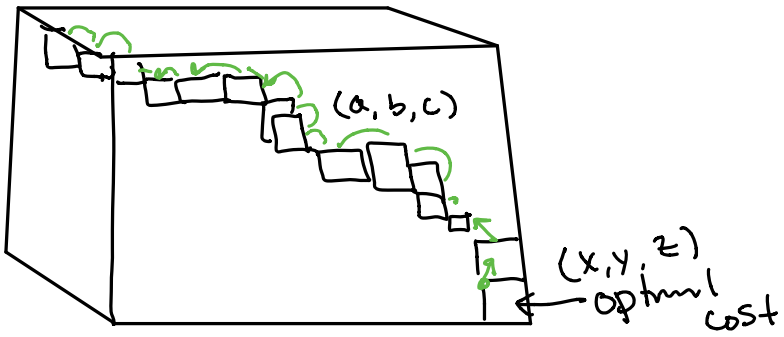
$X_0 \ldots X_{x-1} \mid X_x$
$Y_0 \ldots Y_{y-1} \mid Y_y$
$Z_0 \ldots Z_z \mid -$

or if (X, Y, Z-1) is the min adjant

$X_0 \ldots X_x \mid -$
$Y_0 \ldots Y_y \mid -$
$Z_0 \ldots Z_{z-1} \mid Z_z$

- Let "alignment set" be the set of characters that were aligned at each step

- Alignment sets can be found by finding the path in the matrix A from A[0][0][0] to A[len(X)-1][len(Y)-1][len(Z)-1] that produced the cost in A[max][max][max]



Which at step A[a][b][c] will leave us with alignment sets up to (a,b,c)

ex

$X_0 \ldots X_{a-1}$ | X_a – ... – X_x

$Y_0 \ldots Y_{b-1}$ | Y_b – ... Y_y –

$Z_0 \ldots Z_{c-1}$ | – Z_c ... Z_z –

alignment sets →

where we find resulting value of each alignment set whether it be a character or blank space, then concatonate all alignmt set's results to get final alignment string