Algorithms

LCS

Dynamic Programming

The dynamic programming algorithm for computing the LCS between two input texts (X, Y, with lengths m, n, respectively) is found as follows (adapted from Amaury Habrard’s slides):

Optimal Substructure:

1. For an LCS Z = <z1,…,zk>, if xm = yn, then xm = yn = zk, and Zk-1 is the LCS of Xm-1 and Yn-1
2. If xm ≠ yn, then zk ≠ xm implies Z is LCS of Xm-1 and Yn
3. If xm ≠ yn, then zk ≠ yn implies Z is LCS of Xm and Yn-1

Recursive Solution:

1. c[i,j] = 0 if i = 0 or j = 0
2. c[i,j] = c[i-1,j-1] + 1 if i,j > 0 and xi = yj
3. c[i,j] = max(c[i-1,j],c[i,j-1]) if i,j > 0 and xi ≠ yj

LCS-Length Algorithm(input: X,Y):

1. *m = X.length*
2. *n = Y.length*
3. *b[1..m,1..n], c[0..m,0..n]*
4. *for i in 1..m*
5. *c[i,0] = 0*
6. *for j in 0..n*
7. *c[0,j] = 0*
8. *for i in1..m*
9. *for j in 1..n*
10. *if xi == yj*
11. *c[i,j] = c[i-1,j-1] + 1*
12. *b[i,j] = “d” #for diagonal*
13. *else if c[i-1,j] >= c[i,j-1]*
14. *c[i,j] = c[i-1,j]*
15. *b[i,j] = “u” #for up*
16. *else*
17. *c[i,j] = c[i,j-1]*
18. *b[i,j] = “l” #for left*
19. *return c and b*

Algorithm for building LCS from b matrix *Print-LCS*(input: b, X, X.length, Y.length):

1. *if i == 0 or j == 0*
2. *return*
3. *if b[i,j] == “d”*
4. *Print-LCS(b,X,i-1,j-1)*
5. *elseif b[i,j] == “u”*
6. *Print-LCS(b,X,i-1,j)*
7. *else*
8. *Print-LCS(b,X,i,j-1)*