CSE 417 Algorithms and Complexity

Autumn 2024

Lecture 22

Longest Common Subsequence

1

1

Announcements

- Lecture plans
  - Monday: Longest Common Subsequence
  - Wednesday: Shortest Paths
  - Friday-Wednesday: Network Flow
  - After Thanksgiving: NP Completeness
-

2

2

Last week, subset sum

- Given integers $\{w_1,\ldots,w_n\}$ and an integer K
- Find a subset that is as large as possible that does not exceed K
- Opt[j, K] the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most K
- Opt[j, K] = max(Opt[j – 1, K], Opt[j – 1, K – w_j] + w_j)

```
for j = 1 to n
    for k = 1 to W
        Opt[j, k] = max(Opt[j-1, k], Opt[j-1, k-wj] + wj)
```

3

3

Two dimensional dynamic programming

Subset sum and knapsack

Opt[j, K] = max(Opt[j – 1, K], Opt[j – 1, K – w_j] + w_j)

Opt[j, K] = max(Opt[j – 1, K], Opt[j – 1, K – w_j] + v_j)

4	0																
3	0																
2	0																
1	0																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4

4

Reducing dimensions

- Computing values in the array only requires the previous row
  - Easy to reduce this to just tracking two rows
  - And sometimes can be implemented in a single row
- Space savings is significant in practice
- Reconstructing values is harder

5

5

Longest Common Subsequence

- $C=c_1\ldots c_g$ is a subsequence of $A=a_1\ldots a_m$ if C can be obtained by removing elements from A (but retaining order)
- LCS(A, B): A maximum length sequence that is a subsequence of both A and B

ocurranec

occurrence

attacggct

tacgacca

6

6

Determine the LCS of the following strings

BARTHOLEMEWSIMPSON

KRUSTYTHECLOWN

7

7

String Alignment Problem

- Align sequences with gaps

CAT TGA AT

CAGAT AGGA

- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

Note: the problem is often expressed as a minimization problem, with $\gamma_{xx} = 0$ and $\delta_x > 0$

8

8

Recursive Version

```
LCS(a1a2…am, b1b2…bn){
    if (am == bn)
        return LCS(a1a2…am-1, b1b2…bn-1) + 1;
    else
        return max(LCS(a1a2…am-1, b1b2…bn),
                   LCS(a1a2…am, b1b2…bn-1));
}
```

9

9

LCS Optimization

- $A = a_1a_2\ldots a_m$
- $B = b_1b_2\ldots b_n$
- Opt[j, k] is the length of $LCS(a_1a_2\ldots a_j, b_1b_2\ldots b_k)$

10

10

Optimization recurrence

If $a_j = b_k$, Opt[j,k] = 1 + Opt[j-1, k-1]

If $a_j \neq b_k$, Opt[j,k] = max(Opt[j-1,k], Opt[j,k-1])

11

11

Give the Optimization Recurrence for the String Alignment Problem

- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

Opt[j, k] =

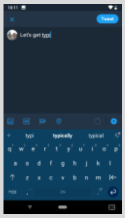
Let $a_j = x$ and $b_k = y$
Express as minimization

12

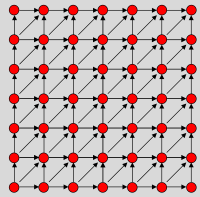
12

String edit with Typo Distance

- Find closest dictionary word to typed word
- Dist('a', 's') = 1
- Dist('a', 'u') = 6
- Capture the likelihood of mistyping characters



Dynamic Programming Computation

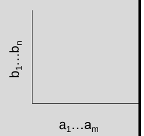


Code to compute Opt[n, m]

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++)
    if (A[ i ] == B[ j ] )
      Opt[ i,j ] = Opt[ i-1, j-1 ] + 1;
    else if (Opt[ i-1, j ] >= Opt[ i, j-1 ])
       Opt[ i, j ] := Opt[ i-1, j ];
    else
       Opt[ i, j ] := Opt[ i, j-1];
```


Storing the path information

```
A[1..m],  B[1..n]
for i := 1 to m     Opt[i, 0] := 0;
for j := 1 to n     Opt[0,j] := 0;
Opt[0,0] := 0;
for i := 1 to m
          for j := 1 to n
                    if A[i] = B[j]  {  Opt[i,j] := 1 + Opt[i-1,j-1];  Best[i,j] := Diag; }
                    else if Opt[i-1, j] >= Opt[i, j-1]
                              {  Opt[i, j] := Opt[i-1, j], Best[i,j] := Left; }
                    else      {  Opt[i, j] := Opt[i, j-1], Best[i,j] := Down; }
```



Reconstructing Path from Distances

LCS Arguments
211031321102033212120000321302
100222010121310130323233121011

0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	2	2	2	2	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
1	2	2	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5
1	2	2	2	2	2	2	3	3	4	4	4	4	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6
1	2	2	2	2	2	2	3	3	4	4	4	5	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6
1	2	2	3	3	3	3	3	3	4	5	5	5	5	5	5	6	6	6	7	7	7	7	7	7	7	7	7	7	7
1	2	2	3	3	3	3	4	4	4	5	6	6	6	6	6	6	6	6	7	7	7	7	7	8	8	8	8	8	8
1	2	2	3	3	3	3	4	4	5	5	6	6	7	7	7	7	7	7	7	7	7	7	7	8	8	9	9	9	9
1	2	3	3	3	3	4	4	5	5	5	6	6	7	8	8	8	8	8	8	8	8	8	8	8	8	9	10	10	10
1	2	3	4	4	4	4	4	5	5	6	6	6	7	8	8	8	8	8	9	9	9	9	9	9	9	9	10	10	10
1	2	3	4	4	4	5	5	5	5	6	6	6	7	8	8	8	9	9	9	9	9	9	9	9	9	9	10	10	10
1	2	3	4	4	4	5	5	5	5	6	6	7	7	8	8	9	9	10	10	10	10	10	10	10	10	10	10	10	10
1	2	3	4	4	4	5	5	5	5	6	6	7	7	8	8	9	9	10	10	11	11	11	11	11	11	11	11	11	11
1	2	3	4	5	5	5	5	5	5	6	6	7	7	8	8	9	9	10	11	11	12	12	12	12	12	12	12	12	12
1	2	3	4	5	5	5	6	6	6	6	7	7	8	8	9	9	9	10	11	11	12	12	12	13	13	13	13	13	13
1	2	3	4	5	6	6	6	6	6	7	7	7	8	8	9	9	9	10	11	11	12	12	12	13	14	14	14	14	14
1	2	3	4	5	6	6	7	7	7	7	8	8	8	8	9	9	9	10	11	11	12	12	12	13	14	15	15	15	15
1	2	3	4	5	6	6	7	7	7	8	8	8	8	8	9	9	9	10	11	11	12	12	12	13	14	15	15	15	15
1	2	3	4	5	6	7	7	8	8	8	8	8	8	9	9	9	10	10	11	11	12	12	12	13	14	15	16	16	16
1	2	3	4	5	6	7	7	8	8	8	8	8	8	9	9	9	10	10	11	11	12	12	12	13	14	15	16	16	16
1	2	3	4	5	6	7	7	8	8	8	8	8	8	9	9	9	10	10	11	11	12	12	12	13	14	15	16	16	16
1	2	3	4	5	6	7	7	8	8	8	8	8	8	9	9	9	10	10	11	11	12	12	12	13	14	15	16	16	16
1	2	3	4	5	6	7	7	8	8	8	8	9	9	9	9	10	10	11	11	12	12	13	13	13	14	15	16	16	16
1	2	3	4	5	6	7	7	8	8	9	9	9	9	9	9	10	10	11	12	12	13	13	13	13	14	15	16	16	16
1	2	3	4	5	6	7	8	8	9	9	10	10	10	10	10	10	10	11	12	12	13	13	13	14	14	15	16	17	17
1	2	3	4	5	6	7	8	8	9	9	10	11	11	11	11	11	11	11	12	13	13	14	14	14	14	15	16	17	17
1	2	3	4	5	6	7	8	9	9	9	10	11	11	12	12	12	12	12	12	13	13	14	14	14	14	15	16	17	17
1	2	3	4	5	6	7	8	9	9	10	10	11	11	12	12	12	12	12	13	13	14	14	14	14	15	15	16	17	17

How good is this algorithm?

- Is it feasible to compute the LCS of two strings of length 300,000 on a standard desktop PC? Why or why not.

Implementation 1

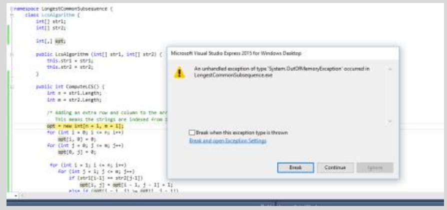
```
public int ComputeLCS() {
        int n = str1.Length;
        int m = str2.Length;

        int[,] opt = new int[n + 1, m + 1];
        for (int i = 0; i <= n; i++)
            opt[i, 0] = 0;
        for (int j = 0; j <= m; j++)
            opt[0, j] = 0;

        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                if (str1[i-1] == str2[j-1])
                    opt[i, j] = opt[i - 1, j - 1] + 1;
                else if (opt[i - 1, j] >= opt[i, j - 1])
                    opt[i, j] = opt[i - 1, j];
                else
                    opt[i, j] = opt[i, j - 1];

        return opt[n,m];
    }
```

19

19

N = 17000

Runtime should be about 5 seconds*



* Personal PC, 10 years old

Manufacturer:	Dell
Model:	Optiplex 990
Processor:	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz 3.10 GHz
Installed memory (RAM):	8.00 GB (7.88 GB usable)
System type:	64-bit Operating System, x64-based processor

20

20

Implementation 2

```
public int SpaceEfficientLCS() {
        int n = str1.Length;
        int m = str2.Length;
        int[] prevRow = new int[m + 1];
        int[] currRow = new int[m + 1];

        for (int j = 0; j <= m; j++)
            prevRow[j] = 0;

        for (int i = 1; i <= n; i++) {
            currRow[0] = 0;
            for (int j = 1; j <= m; j++) {
                if (str1[i - 1] == str2[j - 1])
                    currRow[j] = prevRow[j - 1] + 1;
                else if (prevRow[j] >= currRow[j - 1])
                    currRow[j] = prevRow[j];
                else
                    currRow[j] = currRow[j - 1];
            }
            for (int j = 1; j <= m; j++)
                prevRow[j] = currRow[j];
        }

        return currRow[m];
    }
```

21

21

N = 300000

N: 10000 Base 2 Length: 8096 Gamma: 0.8096 Runtime:00:00:01.86
N: 20000 Base 2 Length: 16231 Gamma: 0.81155 Runtime:00:00:07.45
N: 30000 Base 2 Length: 24317 Gamma: 0.8105667 Runtime:00:00:16.82
N: 40000 Base 2 Length: 32510 Gamma: 0.81275 Runtime:00:00:29.84
N: 50000 Base 2 Length: 40563 Gamma: 0.81126 Runtime:00:00:46.78
N: 60000 Base 2 Length: 48700 Gamma: 0.8116667 Runtime:00:01:08.06
N: 70000 Base 2 Length: 56824 Gamma: 0.8117715 Runtime:00:01:33.36

N: 300000 Base 2 Length: 243605 Gamma: 0.8120167 Runtime:00:28:07.32

22

22

Observations about the Algorithm

- The computation can be done in O(m+n) space if we only need one column of the Opt values or Best Values
- The computation requires O(nm) space if we store all of the string information

23

23

Computing LCS in O(nm) time and O(n+m) space

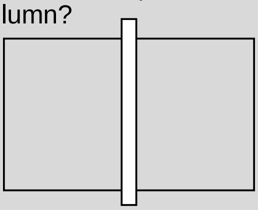
- Divide and conquer algorithm
- Recomputing values used to save space
- Section 6.7 of the text, but we will not have time to cover in detail (so you are not responsible for section 6.7)

24

24

Divide and Conquer Algorithm

- Where does the best path cross the middle column?



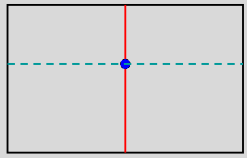
- For a fixed i, and for each j, compute the LCS that has a_i matched with b_j

25

25

Algorithm Analysis

- T(m,n) = T(m/2, j) + T(m/2, n-j) + cnm
- Solution: T(m,n) <= 2cnm



26

26