# DYNAMIC PROGRAMMING: EVEN MORE FUN!

David Kauchak
CS 140 – Fall 2022

1

## Admin

Assignment 6

2

## Mentor hours this week

Thursday: 6-8pm (Aidan)
Friday: 1-3pm (Emily)
Saturday: 9:30-11:30am (Millie)
Sunday: 7-9pm (Carl), 8-10pm (Alan)

3

## LC meetings

Thursday:
- 8-9pm (Emily—Edmunds upstairs, Carl—Edmunds upstairs)

Friday:
- 9-10am (Millie—Edmunds downstairs)
- 2-3pm (Jiahao—Edmunds downstairs, Aidan)
- 3-4pm(Jiahao—Edmunds downstairs)
- 4-5pm (Millie)

4

## Longest increasing subsequence

Given a sequence of numbers X = $x_1, x_2, \ldots, x_n$ find the longest increasing *subsequence* ($i_1, i_2, \ldots, i_m$), that is a subsequence where numbers in the sequence increase.

5 2 8 6 3 6 9 7

5

## Longest increasing subsequence

Given a sequence of numbers X = $x_1, x_2, \ldots, x_n$ find the longest increasing *subsequence* ($i_1, i_2, \ldots, i_m$), that is a subsequence where numbers in the sequence increase.

5 2 8 6 3 6 9 7

6

## 1a: optimal substructure

Prove: optimal solutions to the problem incorporate optimal solutions to related subproblems

5 2 8 6 3 6 9 7

$\{i_1, i_2, \ldots, i_m\}$

What would a solution to a subproblem look like?

7

## 1a: optimal substructure

Prove: optimal solutions to the problem incorporate optimal solutions to related subproblems

5 2 8 6 3 6 9 7

$\{i_1, i_2, \ldots, i_m\}$

$\{i_2, \ldots, i_m\}$ for the sequence starting at index $i_2$

8

## 1a: optimal substructure

Prove: optimal solutions to the problem incorporate optimal solutions to related subproblems

Proof by contradiction:

Assume: $\{i_1, i_2, i_3, \ldots, i_m\}$ is a solution to $x_1 \ldots x_n$ but $\{i_2, i_3, \ldots, i_m\}$ is **not** a solution to $x_{i_2} \ldots x_n$

Then some solution to $x_{i_2} \ldots x_m$ exists, $\{i'_2, i'_3, \ldots, i'_k\}$ where k $> m$.

We could create a solution $\{i_1, i'_2, i'_3, \ldots, i'_k\}$ to the original problem that is a better solution … contradiction

9

## 1b: recursive solution

5 2 8 6 3 6 9 7

Is 5 part off the LIS?

10

## 1b: recursive solution

5 2 8 6 3 6 9 7

Two options:
Either 5 is in the
LIS or it's not

11

## 1b: recursive solution

5 2 8 6 3 6 9 7

include 5

5 + LIS(8 6 3 6 9 7)

12

## 1b: recursive solution

5 2 8 6 3 6 9 7

include 5

$5 + \underbrace{\text{LIS}}(8\ 6\ 3\ 6\ 9\ 7)$

What is this function exactly?

longest increasing sequence of the numbers

longest increasing sequence of the numbers starting with 8

13

## 1b: recursive solution

5 2 8 6 3 6 9 7

include 5

$5 + \underbrace{\text{LIS}}(8\ 6\ 3\ 6\ 9\ 7)$

What is this function exactly?

~~longest increasing sequence of the numbers~~

This would allow for the option of sequences starting with 3 which are NOT valid!

14

## 1b: recursive solution

5 2 8 6 3 6 9 7

include 5

$5 + \underbrace{\text{LIS'}}(8\ 6\ 3\ 6\ 9\ 7)$

longest increasing sequence of the numbers starting with 8

Do we need to consider anything else for subsequences starting at 5?

15

## 1b: recursive solution

5 2 8 6 3 6 9 7

include 5

5 + LIS'(8 6 3 6 9 7)

5 + LIS'(6 3 6 9 7)

5 + LIS'(6 9 7)

5 + LIS'(9 7)

5 + LIS'(7)

16

## 1b: recursive solution

5 2 8 6 3 6 9 7

don't include 5

LIS(2 8 6 3 6 9 7)

Anything else?

Technically, this is fine, but now we have LIS and LIS' to worry about.

Can we rewrite LIS in terms of LIS'?

17

## 1b: recursive solution

$$LIS(X) = \max_i \{LIS'(i)\}$$

Longest increasing sequence for X is the longest increasing sequence starting at any element

And what is LIS' defined as (recursively)?

18

## 1b: recursive solution

$$LIS(X) = \max_i \{LIS'(i)\}$$

Longest increasing sequence for X is the longest increasing sequence starting at any element

$$LIS'(i) = 1 + \max_{j: i < j \leq n \text{ and } x_j > x_i} LIS'(j)$$

Longest increasing sequence starting at i

19

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j: i < j \leq n \text{ and } xj > xi} LIS'(j)$$

LIS':

5 2 8 6 3 6 9 7

20

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' :
$$\begin{array}{cccccccc} & & & & & & & 1 \\ 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \end{array}$$

21

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' :
$$\begin{array}{cccccccc} & & & & & & & 1 \\ 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \end{array}$$

22

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' :
$$\begin{array}{cccccccc} & & & & & & 1 & 1 \\ 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \end{array}$$

23

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' :
$$\begin{array}{cccccccc} & & & & & & 1 & 1 \\ 5 & 2 & 8 & 6 & 3 & 6 & 9 & 7 \end{array}$$

24

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j: i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' : 2 1 1

5 2 8 6 3 6 9 7

25

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j: i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' : 3 2 1 1

5 2 8 6 3 6 9 7

26

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j: i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' : 2 3 2 1 1

5 2 8 6 3 6 9 7

27

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j: i<j\leq n \text{ and } xj>xi} LIS'(j)$$

LIS' : 2 2 3 2 1 1

5 2 8 6 3 6 9 7

28

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\le n \text{ and } xj>xi} LIS'(j)$$

LIS': 4 2 2 3 2 1 1

5 2 8 6 3 6 9 7

29

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\le n \text{ and } xj>xi} LIS'(j)$$

LIS': 3 4 2 2 3 2 1 1

5 2 8 6 3 6 9 7

30

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\le n \text{ and } xj>xi} LIS'(j)$$

LIS': 3 4 2 2 3 2 1 1

5 2 8 6 3 6 9 7

$$LIS(X) = \max_i \{LIS'(i)\}$$

31

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\le n \text{ and } xj>xi} LIS'(j)$$

What does the data structure for storing answers look like?

32

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

1-D array: only one thing changes for recursive calls

33

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

What are the “smallest” possible subproblems?

To calculate LIS’(n), what are all the subproblems we need to calculate? This is the “table”.

How should we fill in the table?

Where will the answer be?

34

## 2: DP solution (bottom-up)

$$LIS'(i) = 1 + \max_{j:i<j\leq n \text{ and } xj>xi} LIS'(j)$$

What are the “smallest” possible subproblems?
LIS’(n) and that is well-defined for this problem

To calculate LIS’(i), what are all the subproblems we need to calculate? This is the “table”.
LIS’(1) … LIS’(n)

How should we fill in the table?
n → 1

Where will the answer be?
max(LIS’(1)…LIS’(n))

35

## 2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
    max ← 1
    for j ← i + 1 to n
        if X[j] > X[i]
            if 1 + lis[j] > max
                max ← 1 + lis[j]
    lis[i] ← max
max ← 0
for i ← 1 to n
    if lis[i] > max
        max ← lis[i]
return max
```

36

## 2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

start from the end (bottom)

37

## 2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

$$LIS'(i) = 1 + \max_{j: i < j \le n \text{ and } xj > xi} LIS'(j)$$

38

## 2: DP solution (bottom-up)

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

$$\boxed{LIS(X) = \max_i \{LIS'(i)\}}$$

39

## 3: Analysis

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

Space requirements?

Running time?

40

## 3: Analysis

```
LIS(X)
n ← LENGTH(X)
create array lis with n entries
for i ← n to 1
        max ← 1
        for j ← i + 1 to n
                if X[j] > X[i]
                        if 1 + lis[j] > max
                                max ← 1 + lis[j]
        lis[i] ← max
max ← 0
for i ← 1 to n
        if lis[i] > max
                max ← lis[i]
return max
```

Space requirements: $\Theta(n)$

Running time: $\Theta(n^2)$

41

## Another solution

Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7

LCS

2 3 5 6 6 7 8 9

42

## Another solution

Can we use LCS to solve this problem?

5 2 8 6 3 6 9 7

LCS

2 3 5 6 6 7 8 9

43

## Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$

Insertion:

ABACED → ABACCED → DABACCED

Insert 'C'

Insert 'D'

44

## Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$

Deletion:

ABACED

45

## Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$

Deletion:

ABACED → BACED

Delete 'A'

46

## Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$

Deletion:

ABACED → BACED → BACE

Delete 'A' Delete 'D'

47

## Edit distance (aka Levenshtein distance)

Edit distance between two strings is the minimum number of insertions, deletions and substitutions required to transform string $s_1$ into string $s_2$

Substitution:

ABACED → ABADED → ABADES

Sub 'D' for 'C' Sub 'S' for 'D'

48

## Edit distance examples

Edit(Kitten, Mitten) = 1

Operations:

Sub 'M' for 'K' Mitten

49

## Edit distance examples

Edit(Happy, Hilly) = 3

Operations:

Sub 'a' for 'i' Hippy
Sub 'l' for 'p' Hilpy
Sub 'l' for 'p' Hilly

50

## Edit distance examples

Edit(Banana, Car) = 5

Operations:

Delete 'B' anana
Delete 'a' nana
Delete 'n' naa
Sub 'C' for 'n' Caa
Sub 'a' for 'r' Car

51

## Edit distance examples

Edit(Simple, Apple) = 3

Operations:

Delete 'S' imple
Sub 'A' for 'i' Ample
Sub 'm' for 'p' Apple

52

## Edit distance

Why might this be useful?

53

## Is edit distance symmetric?

that is, is $\text{Edit}(s_1, s_2) = \text{Edit}(s_2, s_1)$?

Edit(Simple, Apple) =? Edit(Apple, Simple)

Why?

- sub 'i' for 'j' → sub 'j' for 'i'
- delete 'i' → insert 'i'
- insert 'i' → delete 'i'

54

## Calculating edit distance

X = A B C B D A B

↓

Y = B D C A B A

Ideas? How can we break this into subproblems?

55

## Calculating edit distance

X = A B C B D A ?

↓

Y = B D C A B ?

After all of the operations, X needs to equal Y

Start with the last two characters

56

## Calculating edit distance

X = A B C B D A ?

⬇

Y = B D C A B ?

Operations: Insert

Delete

Substitute

Assume they're different

How can we make them the same?

57

## Insert

X = A B C B D A ?

⬇

Y = B D C A B ?

How can we use insert to transform X into Y?

58

## Insert

X = A B C B D A ? ?

⬇

Y = B D C A B ?

insert the last character of Y to the end of X

59

## Insert

X = A B C B D A ? ?

⬇

Y = B D C A B ?

How does this make the problem smaller?

60

## Insert

X = A B C B D A ? ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1})$$

61

## Delete

X = A B C B D A ?

Y = B D C A B ?

How can we use delete to transform X into Y?

62

## Delete

X = A B C B D A ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m})$$

63

## Substition

X = A B C B D A ?

Y = B D C A B ?

How can we use substitution to transform X into Y?

64

## Substition

X = A B C B D A ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = 1 + Edit(X_{1...n-1}, Y_{1...m-1})$$

65

## Anything else?

X = A B C B D A ?

Y = B D C A B ?

66

## Equal

X = A B C B D A ?

Y = B D C A B ?

What if the last characters are equal?

67

## Equal

X = A B C B D A ?

Edit

Y = B D C A B ?

$$Edit(X,Y) = Edit(X_{1...n-1}, Y_{1...m-1})$$

68

## 1b: recursive solution - combining results

Insert: $Edit(X,Y) = 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1})$

Delete: $Edit(X,Y) = 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m})$

$X_n \neq Ym$
Substitute: $Edit(X,Y) = 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1})$

$X_n = Y_m$
Equal: $Edit(X,Y) = Edit(X_{1\ldots n-1}, Y_{1\ldots m-1})$

How do we decide between these?

69

## 1b: recursive solution - combining results

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

1: if they're different
0: if they're the same

70

## 2: DP solution (bottom-up)

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

What does the data structure for storing answers look like?

71

## 2: DP solution (bottom-up)

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

$Edit(X_{1\ldots i}, Y_{1\ldots j})$

$\Downarrow$

d$[i, j]$: edit distance between $X_{1\ldots i}$ and $Y_{1\ldots j}$

72

## 2: DP solution (bottom-up)

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

What are the "smallest" possible subproblems?

To calculate $d(n, m)$, what are all the subproblems we need to calculate? This is the "table".

How should we fill in the table?

Where will the answer be?

73

## 2: DP solution (bottom-up)

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

What are the "smallest" possible subproblems?
Edit(X, "") = len(X) and Edit("", Y) = len(Y)

To calculate $d(n, m)$, what are all the subproblems we need to calculate? This is the "table".
i < n and j < m

How should we fill in the table?
`i = 1…, j = 1…`

Where will the answer be?
d[n,m]

74

## 2: DP solution (bottom-up)

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
Edit(X, Y)
m ← length[X]
n ← length[Y]
for i ← 0 to m
        d[i, 0] ← i
for j ← 0 to n
        d[0, j] ← j
for i ← 1 to m
        for j ← 1 to n
                d[i, j] = min(1 + d[i − 1, j],
                              1 + d[i, j − 1],
                              DIFF(x_i, y_j) + d[i − 1, j − 1])
return d[m, n]
```

75

## 3: analysis

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
Edit(X, Y)
m ← length[X]
n ← length[Y]
for i ← 0 to m
        d[i, 0] ← i
for j ← 0 to n
        d[0, j] ← j
for i ← 1 to m
        for j ← 1 to n
                d[i, j] = min(1 + d[i − 1, j],
                              1 + d[i, j − 1],
                              DIFF(x_i, y_j) + d[i − 1, j − 1])
return d[m, n]
```

Space requirements?

Running time?

76

## 3: analysis

$$Edit(X,Y) = \min \begin{cases} 1 + Edit(X_{1\ldots n}, Y_{1\ldots m-1}) & \text{insertion} \\ 1 + Edit(X_{1\ldots n-1}, Y_{1\ldots m}) & \text{deletion} \\ Diff(x_n, y_m) + Edit(X_{1\ldots n-1}, Y_{1\ldots m-1}) & \text{equal/substitution} \end{cases}$$

```
EDIT(X, Y)
  m ← length[X]
  n ← length[Y]
  for i ← 0 to m
          d[i, 0] ← i
  for j ← 0 to n
          d[0, j] ← j
  for i ← 1 to m
          for j ← 1 to n
                  d[i, j] = min(1 + d[i − 1, j],
                                1 + d[i, j − 1],
                                DIFF(x_i, y_j) + d[i − 1, j − 1])
  return d[m, n]
```

Space requirements: $\Theta(nm)$

Running time: $\Theta(nm)$

77

## Edit distance variants

- Only include insertions and deletions
  - What does this do to substitutions?
- Include swaps, i.e. swapping two adjacent characters counts as one edit
- Weight insertion, deletion and substitution differently
- Weight **specific** character insertion, deletion and substitutions differently
- Length normalize the edit distance

78

## Skiers and Skis

Skis: 1 5 5 7 9 12 12 13

Skiers: 6 7 7 10 12

What is the optimal matching?

79