Dp Lab 1

Longest common subsequence

1) String s

2) Longest Palindromic Subsequence : (s, rev(s)) *

3) Longest Bitonism Subsetting : (s, rev(s)) *

Longest common string

s: [a b c] h c [c b] a g

s1: [a b c] e k [c b] a

→ length = 3 *

Dp concepts / Problem

1) Problem : Sum

2) Backtrack :

Check path : good or bad

3) Recursion only

4) Memoization dp

5) Tabulation dp : dp2

6) Way to Recurse : dp2

Max / min : dp2

7) Iterative

B = 3

N = 4

1	2	3	4	5	6	7
3	-2	4	6	-3	-2	4

[1] → length [a] → [3]

path → [5 - 3] → prices[1]

[1, 1, 0, 3, 1, 0]

dp[i] = Max profit you can make for [i - N-1]

dp[i] = { Recurse → dp[i+1] ; dp[i+B+1] + prices[i] } Max value

dp[i] dp[i+1] dp[i+2]

Find ans: dp[0], dp[N-1] = max(0, prices[N-1])

{ 0 1 1 1 2 }

s → 0

(0,0) (N-1, M-1)

if dp[N-2, M-1] == 0 && dp[N-1, M-2] == 0) return 0

if (dp[N-2, M-1] == 0 && dp[N-1, M-2] == 0) return 1

if (dp[N-2, M-1] == 0 && dp[N-1, M-1] == 0) return 1

call 3

→ Radar!

Ans 0 1 1 1 2

dp1: [0] → 0, dp2[0] for dp & ans

→ for every dp [ ] { Now every cell value was for [ ] } →

→ match if dp [ ] equals

→ check along two diagonal

* dp[N-1][M-2]

→ 1 Blue

return 2.

n = dp[N-1][M-2]

Wild Card Matching

{ ? — Can take any character we want

* — It take as many character as match

↳ It can also be a empty character }

1) reg1 : a b c c

reg2 : a ? ? ?

2) reg1 : a b c c

reg2 : a * ?

3) { reg1 : a b

reg2 : a * b } { reg1 : a b

reg2 : a ? b }

0 1 2 3 4 5 6 7

s1 : a b c d f a c

s2 : ? b * d * c

0 1 2 3 4 5 6

s1[3] == s2[4]

→ (4, 5)

dp[i,j] = Check if we can match s1[0-i] & s2[0-j]

dp[i,j] = { if (s1[i] == s2[j] || s2[j] == '?') → { dp[i-1][j-1] }

else if (s1[i] == s2[j] && s2[j] == '*') → { false }

else { if s2[j] == * →

* → empty character → dp[i][j-1]

* → 1 character → dp[i-1][j-1]

* → 2 character → dp[i-2][j-1]

* → 3 character → dp[i-3][j-1]

* → leave → dp[i][j-1]

* → pick & staying → dp[i-1][j] }

unlimited length

→ [Recursion Code]

---

dp[i][j] = check if we can match s1[0-i] to s2[0-j]

dp[i][j] = { Base Case : s1 : ? c ; s2 : ? c

Case 1 i == -1 && j == -1 → Matched Completely

Case 2 i >= 0 && j == -1 → s1 already vanished

Case 3 i == -1 && j >= 0 → In simple, return false

Case 4 i >= 0 && j >= 0 → All remaining characters in s2[0-j] they have to '*'. Simple match

→ In simple match

s1 : N

s2 : M

if (s1[i] == s2[j] || s2[j] == '?') { dp[i-1][j-1] }

else if (s2[j] == '*') { * not picking any character: dp[i][j-1] ; * picking & staying: dp[i-1][j] }

else { dp[i][j] = false }

⇒ Final ans : dp[N-1][M-1]

s1 = "a"

s2 = "****"

s1 = "a"

s2 = "*a**"

s1 = "a"

s2 = "a?"

{ ? = ——

* = [Can replace to character] }

---

* Tabu

* Recap dp

* Solving strategy

Tabulation

{ 8 9 - 11 9 }

Tabulation

Ways to decode

1 - a

2 - b

3 - c

26 - z

1 1 3 → 1 1 + 3 → aac ; 1 1 3 → 11 + 3 → bc ; 1 1 3 → A M

if character == '0' { 0 0 } → { 0 }

{ 1 0 7 8 9 7 10 }

dp[i] = { dp[i+1] ; dp[i+2] if 1 ≤ s[i] ≤ 26 }

if i == 0 dp[i] = 1

else { dp[i+1] ; dp[i+2] if 1 ≤ s[i] ≤ 26 }

Ex: 30

→ Code

dp[N-1], dp[N-2], one zero loop can

for (int i = N-1; i >= 0; i--) {

if (s[i] == '0') { dp[i] = 0; }

else { dp[i] = dp[i+1]

// add a number by looking dp[i] & dp[i+1]

[int x = dp[i+1] - '0' * 10]

[int y = dp[i+2] - '0' * 1]

if (x * 10 + y ≤ 26) { dp[i] += dp[i+2] } } }

return dp[0]

Ex: 10 20

Then there be a '0'

{ 10 20 }