DP-2

~~LIS (Longest increasing subsequence)~~

~~LCS (longest common subsequence)~~

Edit Distance } 2D Matrix

-> LIS length
-> longest increasing subsequence

substring

subsequence
[abcdef]
= [ace]

=> {1, 6, 3, 2, 8, 5}
=> {1, 3, 5} => 3
=> {1, 6, 8}
=> {1, 2, 8}

(i-1) (i)

=> (1, 6, 3, 2, 8, 5) 38

(1) = 1, (1, 6) => 2, (1, 6, 3) = 2
(1, 6, 3, 2) => 2 (1, 6, 3, 2, 8) => 3

16328
=> (128) = 3
1638 = 3
138

8 > 2 (1, 6, 3, 2) 8 +1

(1 9 1 8) => 2

(1) = 1 (19) = 2

(191) = 1 (1918) = 2

strictly
1122
1234
1212

_ _ _ X X X X Y
XY

_ _ X _ X
0 1 2 3 4

if (Y > X)
= ans(... X) + 1 = ()
XY

=> Optimal Substructure ✓
=> Overlapping Subproblem ✓

CSRAB

1) (i) -> (i-1)

2) lis[i] => longest increasing subsequence if i element is included

3) Recurrence lis[i] -> i-1 { if arr[i] < arr[j] max (lis[j] +1) else (1) }

4) Answer max [lis]

5) Base (1)63285

1382 max=1

| 1 | 2 | 3 | |

lis[0] = 1
for (i=1; i<n; i++)
{ ans = 1
for (j=0; j<i; j++)
{ if (arr[i] > arr[j])
ans = max(ans, arr[j]+1)
}
lis[i] = ans;
}

=> LCS

longest common subsequence (2)

=> ("aab", "azb") = ab

-> lcs("aab", "azb") 12458 (i)

-> 1 + lcs("aa", "az") 1

max (lcs("aa", "a"), lcs("a", "az")) 1

1 + lcs("a", "") 0

max (lcs("a", "a"), lcs("", "az")) 1

1) Optimal Substructure ✓
2) Overlapping Subproblem ✓

f(n) = (f(n-1) + f(n-1))

((eye), (xyf))

(ey, xyf) (eye, xy)

("abc", "aabbccd")

f(2^k) = f(2^(k-1)) + 1
f(16) = f(8) + 1
f(2) = f(1) + 1

16
8
4
2
1

=> (abxygp, acyzbp)

	""	a	b	x	y	g	p
""	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	1	1	1	1
y	0	1	1	1	2	2	2
z	0	1	1	1	2	2	2
b	0	1	2	2	2	2	2
p	0	1	2	2	2	2	3

CSRAB

1) if not equal -> (i-1, j) (i, j-1)

2) lcs(i,j) => longest common sequence for $S_1[0 \ldots i]$ & $S_2[0 \ldots j]$

3) if equal lcs(i-1)(j-1) + 1
else max (lcs(i-1)(j), lcs(i)(j-1))

4)

5) Base Case => { comparison with empty string is 0 }

max

bottom right

0 lcs (m-1) (n-1)

if (S[i] == S[j])
lcs[i][j] = 1 + lcs[i-1][j-1]

= 1 + lcs("", "")

else if (s[i] != s[j])
(abx, aby) lcs[i][j]

max ((abx, ab), (ab, aby))
max (lcs[i][j-1], lcs[i-1][j])

lcs ("a", "ab")

lcs ("a", "a")

{ Time = O(nm)
Space = O(n^2) }

tc => O(n^2) ab -> dcfe

S -> OC

=> Edit Distance

1) Replace any chars with any other character
2) insert a character
3) delete a character

S_1 => "benyam"

S_2 => "ephrem"

Situation 1 MATCH

(i m) => (j m)

=> dp[i-1][j-1]

=> (beny, ehpr)

I replace

ans ("beny" -> "ehpx") (y) will be replaced (r)

=> ans ("ben" -> "ehp") + 1
= dp[i-1][j-1] + 1

II insert

ans ("beny", "ehpr") => ans ("beny" -> "ehp")
=> dp[i-1][j] + 1

III delete

ans ("beny" -> "ehpr") deleting y
(i) (j)

= ans ("ben" -> "ehpr")
+ 1

("" -> e) ("b" -> e)

(i)	""	b	e	n	y	a	m
""	0	1	2	3	4	5	6
e	1	1	1	2	~~2~~	4	5
p	2	2	2	2	3	4	5
h	3	3	3	3	3	4	5
r	4	4	4	4	4	4	5
e	5	5	4	5	5	5	5
m	6	6	5	5	6	6	5

time => O(n^2)
Space => O(n^2)

CSRAB

1) if equal -> 1 thing to do
not equal -> 3 choices

2) dp[i][j] => minimum operations to convert A[0 ... i] to B[0 ... j]

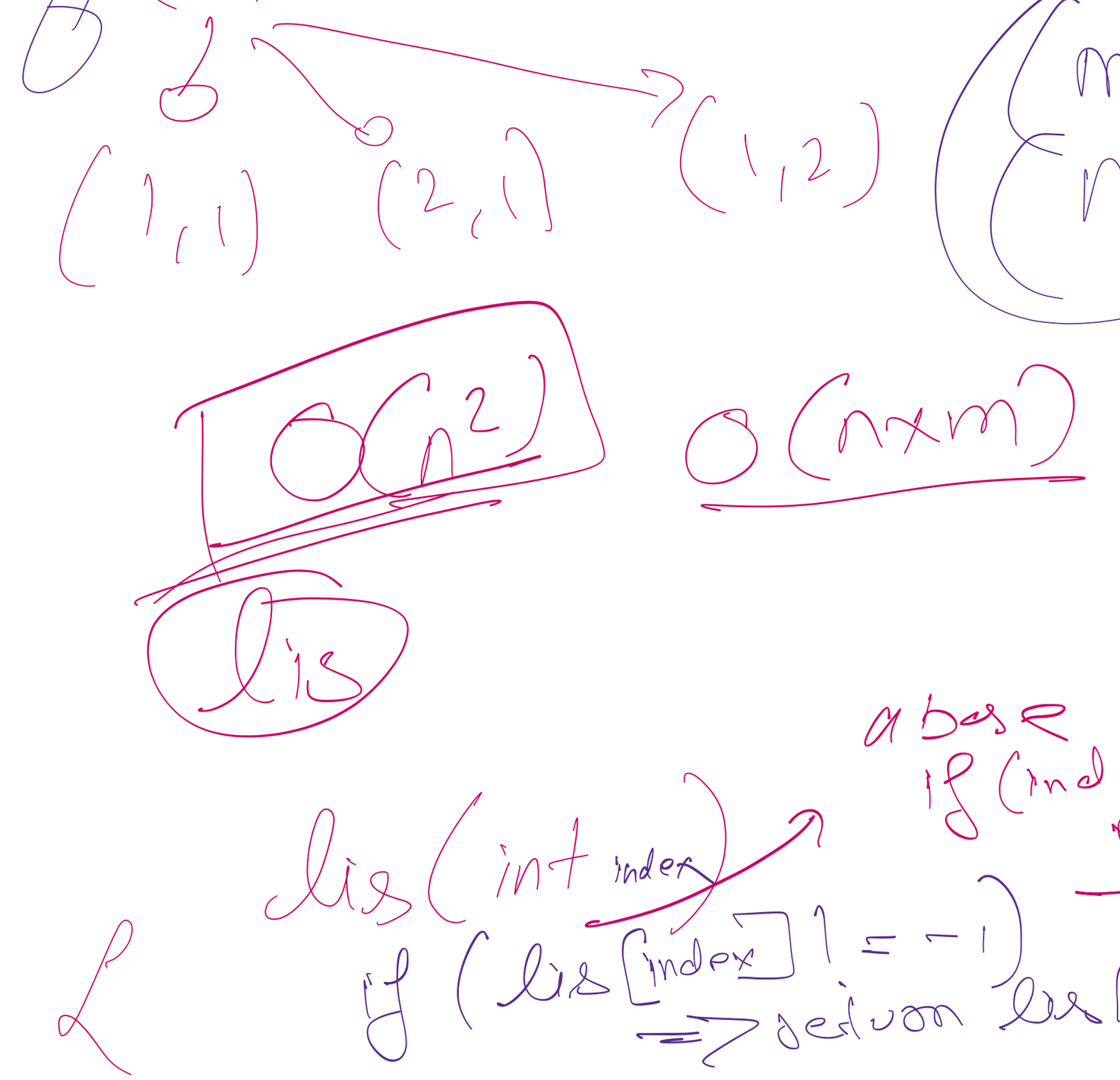
3) if A(i) = B[j]
=> dp (i-1)(j-1)
else
dp(i)(j) = min (dp(i-1, j-1) -> replace
dp(i-1, j) -> insert
dp(i, j-1) -> delete) + 1

j = string which needs converted
i => string to which we need to convert

4) Answer
dp(i-1)(j-1)

5) Base "" empty

= O(n)



m+n

(m×n)
(m+n)

O(n^2) O(n×m)

lis

lis (int index)
{ a base
if (index == 0)
return 1
if (lis[index] != -1)
=> return lis[index]
int ans = 1
for (i=0; i < index; i++)
{ if (arr[index] > arr[i])
ans = max (ans, lis(i)+1)
}
lis[index] => ans;
return lis[index];
}

=> lis

1 3 7 4 5

1 7 2 4 (5) 3

| x | y | a | b | |) => 45

b+1

(ab xy -> byzp) insert
(i) (j)

"" -> ab
ab

=> P

ans ("abxy" -> "byz") + 1

4 3 byzp

ans ("abxy" -> "byzp")

ans ("abx" -> "byzp")