CSE525 Lec10: Dynamic Programming

Debajyoti Bera (M21)

Parenthesis of arithmetic expressions

1+ 3 x 2 x 0 + 1 x 6 + 7

Best parenthesis positions to yield the largest value.

$(1+(3\times 2))\times 0+(1\times 6)+7=13$

$((1+(3\times 2\times 0)+1)\times 6)+7=19$

$(1+3)\times 2\times (0+1)\times (6+7)=104$

Consider a backtracking search tree... what would be the possibilities?

$(1+3\times 2)$

Best value of $\text{Expr.}(E_1 \dots E_n) = \max_{j=1\dots n-1} P(1, j, n)$

$P(5, j, 6) = (E_5 + E_6)$

$(7+5+6)$

$((7+5)+6)$

3D array $(n+1)\times(n+1)\times(n+1)$

$$P(i,j,k) = \begin{cases} E_i & \text{if } k = i \\ -\infty & \text{if } j < i \text{ or } j \geq k \\ (E_i + E_k) \text{ or } (E_i \times E_k) & \text{if } k = i+1 \\ \left[\max_{t=i\dots j-1} P(i,t,j)\right] + \left[\max_{s=j+1\dots k} P(j+1,s,k)\right] & \text{if sign is } + \\ \left[\max_{t=i\dots j-1} P(i,t,j)\right] \times \left[\max_{s=j+1\dots k} P(j+1,s,k)\right] & \text{if sign is } \times \end{cases}$$

Parenthesis of arithmetic expressions

1 + 3 x 2 x 0 + 1 x 6 + 7

$$((1+(3\times 2))\times 0)+((1\times 6)+7) = 13$$

$$((1+((3\times 2)\times 0)+1)\times 6)+(7) = 19$$

$$((1+3)\times 2)\times((0+1)\times(6+7)) = 104$$

Maximum possible value = ?

Group into (...) x (...) or (...) + (...)

(1) + (3 x 2 x 0 + 1 x 6 + 7)

(1 + 3) x (2 x 0 + 1 x 6 + 7)

(1 + 3 x 2) x (0 + 1 x 6 + 7) ...

What should be problem that we recursively solve?

P(i,j,k) = maximum possible value by grouping E[i...k] as (E[i ... j]) and (E[j+1 ... k]).

P(1,3,5) = max. value from (1,3,2) & (0,1) = max. value from (1 + 3 x 2) x (0 + 1) = ?

① all cases are covered

Any optimal position can be expressed as expression over pairs

② optimal substructure prop.

② Best value for Expr E given that top-level parenthesis ($E_1 \cdots E_k$) ($E_{k+1} \cdots E_n$) must involve the best value for ($E_1 \cdots E_k$) & ($E_{k+1} \cdots E_n$)

Best value(E) = 200

Best (E′) = 150

Best (E″) = 250

Edit Distance

ALGORITHM → . . . . → ALTRUISTIC
d d d . . . . . d
i i i i
dist = 9 + 10 = 19

Dist(ALGORITHM, ALTRUISTIC) = ?

Given A[1...n] & B[1...m], compute length of smallest edit seq from A → B

ALGORITHM → ALTRUISTI → ALTRUISTIC

A	L	G	O	R		I		T	H	M
A	L		T	R	U	I	S	T	I	C
.	.	*d*	*c*	.	*i*	.	*i*	.	*c*	*c*

Cases?

dist = 6

ALGORITHM → AL·ORITHM → AL·TRITHM → AL·TRUITHM → AL·TRUISTHM → ALTRUISTIM → AL·TRUISTIC

we can always change from left → right.

ALGORITHM → ALGORITHC → ...

ALGO → ALTO

S = opt. edit seq. from A⋯M → A⋯C

Three cases for the last no-op step in:
→ S ends in C
→ S ends in i
→ S ends in d

Edit Distance

Dist(ALGORITHM, ALTRUISTIC) = ?

Seq1 : best edit seq. from ALG… → ALT… in which M → C

(1) Coverall cases

(2) optimal substructure property

Claim:- subpart of seq1 must be best edit seq. from ALG⋯TH → ALT⋯STI

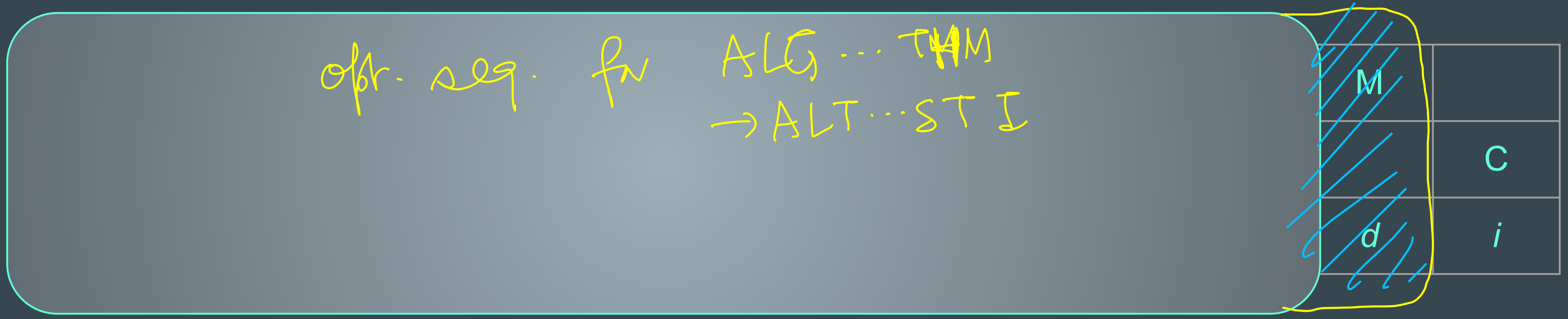
∴ subpart of seq1 is optimal for the subproblem

M
C
c

Minimal sequence of changes for the larger instance (say, with 6 changes) must include the minimal sequence of changes (5 changes) for the smaller instance. **Why?** If there is a sequence with 4 or less changes, then those changes & **M** → **C** (1 change) would yield a sequence with 5 or less changes for the larger instance.

Edit Distance

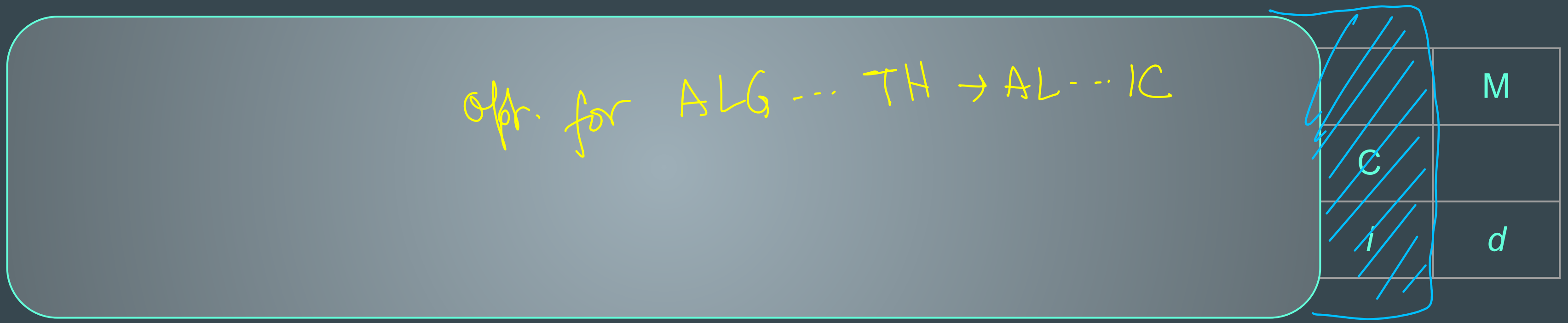
Dist(ALGORITHM, ALTRUISTIC) = ?



Minimal sequence of changes for the larger instance (say, with 7 changes) must include the minimal sequence of changes (5 changes) for the smaller instance. **Why?** If there is a sequence with 4 or less changes, then those changes & $\mathbf{M}. \to .\mathbf{C}$ (2 changes) would yield a sequence with 6 or less changes for the larger instance.

Edit Distance

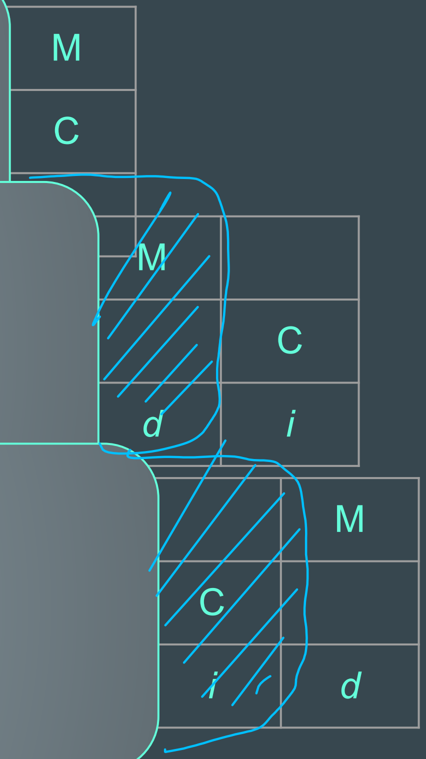
Dist(ALGORITHM, ALTRUISTIC) = ?



Minimal sequence of changes for the larger instance (say, with 7 changes) must include the minimal sequence of changes (5 changes) for the smaller instance. **Why?** If there is a sequence with 4 or less changes, then those changes & $.\,\mathbf{M} \rightarrow \mathbf{C}\,.$ (2 changes) would yield a sequence with 6 or less changes for the larger instance.

Edit Distance

Try all possibilities to convert M to C

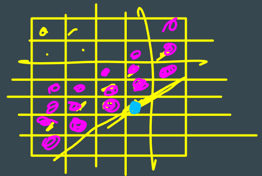


Order :- row wise
column wise

Opt. Edit Dist = Edit(|A|, |B|)

Memo : 2D array of (|A|+1) × (|B|+1)

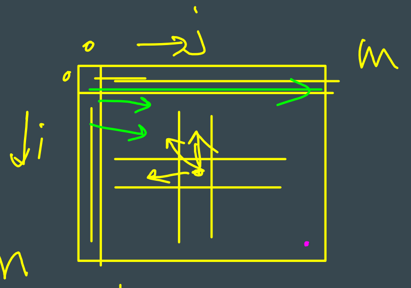
Time comp. : O(mn)



Recurrence

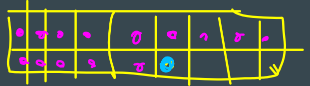
Space complexity: O(mn)
O(min(m,n))

Edit(i,j) = minimum number of edits to changes A[1...i] to B[1...j]



Edit seq :-> space complexity O(mn)

$$
Edit(i,j) = \min \begin{cases} Edit(i-1,j) + 1 & \text{delete } A[i] \\ Edit(i,j-1) + 1 & \text{insert } A[j] \\ Edit(i-1,j-1) + 1 & \text{change } A[i] \to B[j] \text{ if } A[i] \neq B[j] \\ Edit(i-1,j-1) & \text{do nothing if } A[i] = B[j] \end{cases}
$$



$$
\begin{cases} 1 & \text{if } A[i] \neq B[j] \text{ if } i=1,\ j=1 \\ 0 & \text{if } A[i] = B[j] \\ j & \text{if } i=0 \end{cases}
\qquad
\begin{cases} i & \text{if } j=0 \end{cases}
$$