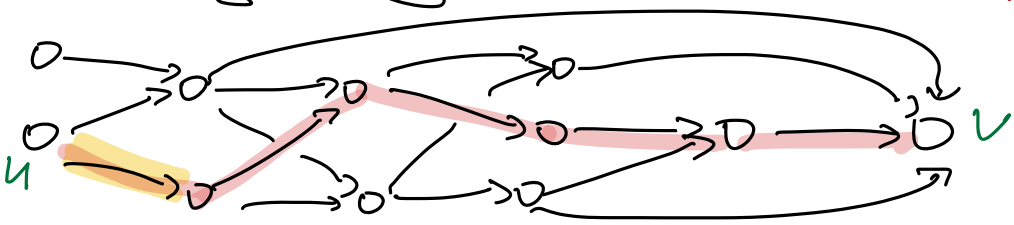
Dynamic Programming over DAGs.

① Top. sort



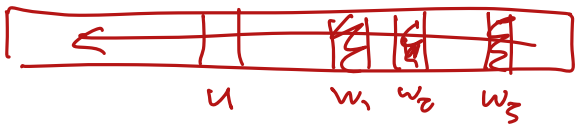
Longest Path in DAG $G=(V,E)$ Q: First edge?

$LLP(u,v)$ = length of longest path from u to v

$$LLP(u,v) = \begin{cases} 0 & \text{if } u=v \\ \max\{ w(u\to x) + LLP(x,v) \mid u\to x \in E \} & \end{cases}$$

$\max \emptyset = -\infty$

① Memoize into array indexed by top. order labeling of V



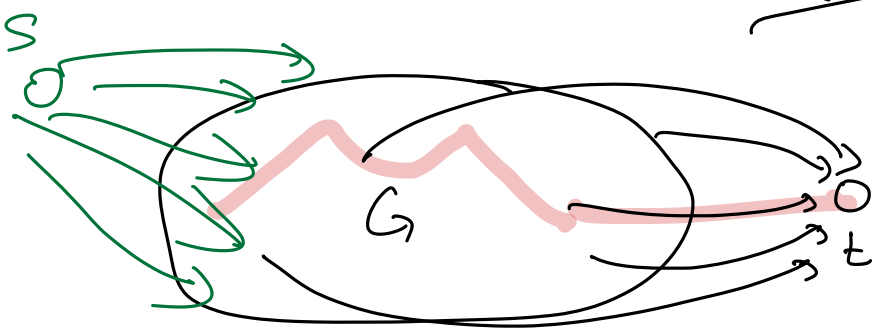
Eval in reverse topological order

② Memoize into G itself store $LLP(u,v)$ in $u.LLP$ [v is fixed]

Eval in DFS postorder

Time is $O(V+E)$ for each fixed v.

$\Rightarrow$ ~~$O(V(V+E))$ overall~~

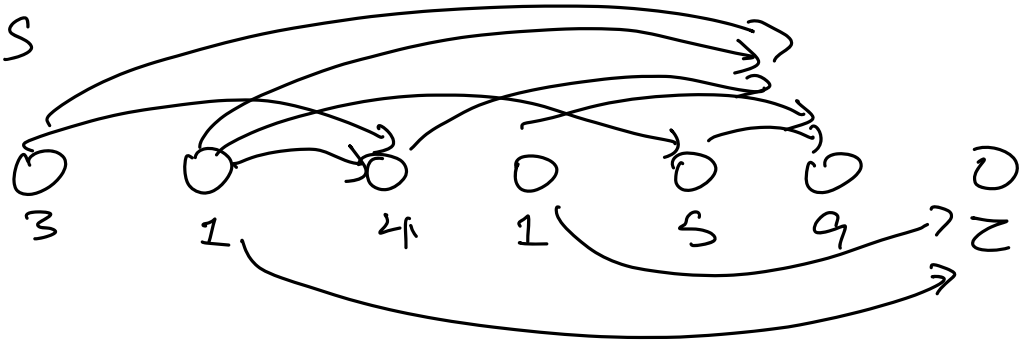


Add artificial sink

Compute $LLP(x,t)$

-1

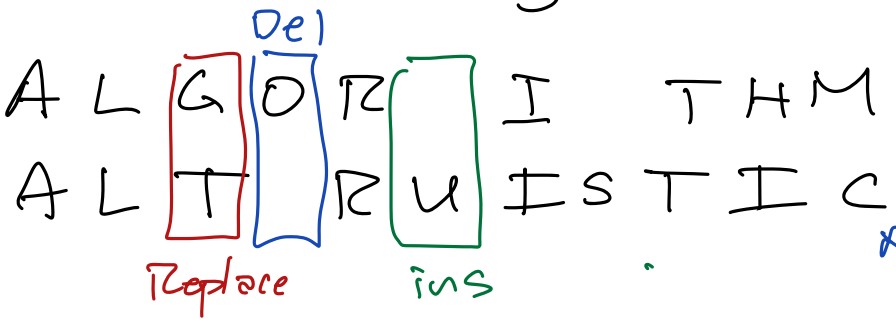
LIS



Edit Distance

BRANCHED

~~DISTANCED~~ B

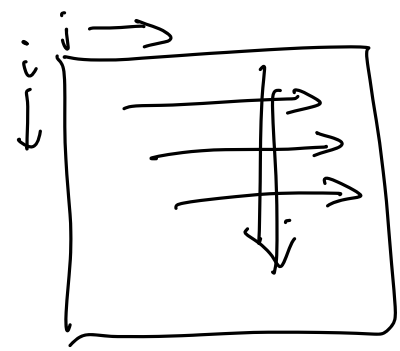


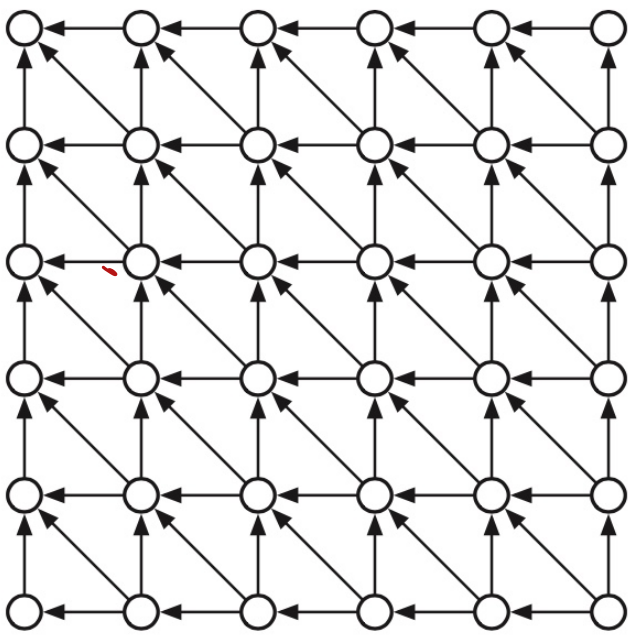
Cost = 6

#ops = #cols without $\frac{x}{x}$

Edit(i,j) = edit distance from A[1..i] to B[1..j]

$$Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} Edit(i,j-1)+1 & \leftarrow \text{ins} \\ Edit(i-1,j)+1 & \leftarrow \text{del} \\ Edit(i-1,j-1)+[A[i] \neq B[j]] & \leftarrow \text{rep} \end{cases} & \text{otherwise} \end{cases}$$



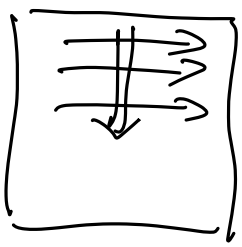


```
EditDistance(A[1..m], B[1..n]):
  for j ← 0 to n
      Edit[0, j] ← j

  for i ← 1 to m
      Edit[i, 0] ← i
      for j ← 1 to n
          ins ← Edit[i, j − 1] + 1
          del ← Edit[i − 1, j] + 1
          if A[i] = B[j]
              rep ← Edit[i − 1, j − 1]
          else
              rep ← Edit[i − 1, j − 1] + 1
          Edit[i, j] ← min {ins, del, rep}

  return Edit[m, n]
```

		A	L	G	O	R	I	T	H	M
	0	1	2	3	4	5	6	7	8	9
A	1	0	1	2	3	4	5	6	7	8
L	2	1	0	1	2	3	4	5	6	7
T	3	2	1	1	2	3	4	4	5	6
R	4	3	2	2	2	2	3	4	5	6
U	5	4	3	3	3	3	3	4	5	6
I	6	5	4	4	4	4	3	4	5	6
S	7	6	5	5	5	5	4	4	5	6
T	8	7	6	6	6	6	5	4	5	6
I	9	8	7	7	7	7	6	5	5	6
C	10	9	8	8	8	8	7	6	6	6

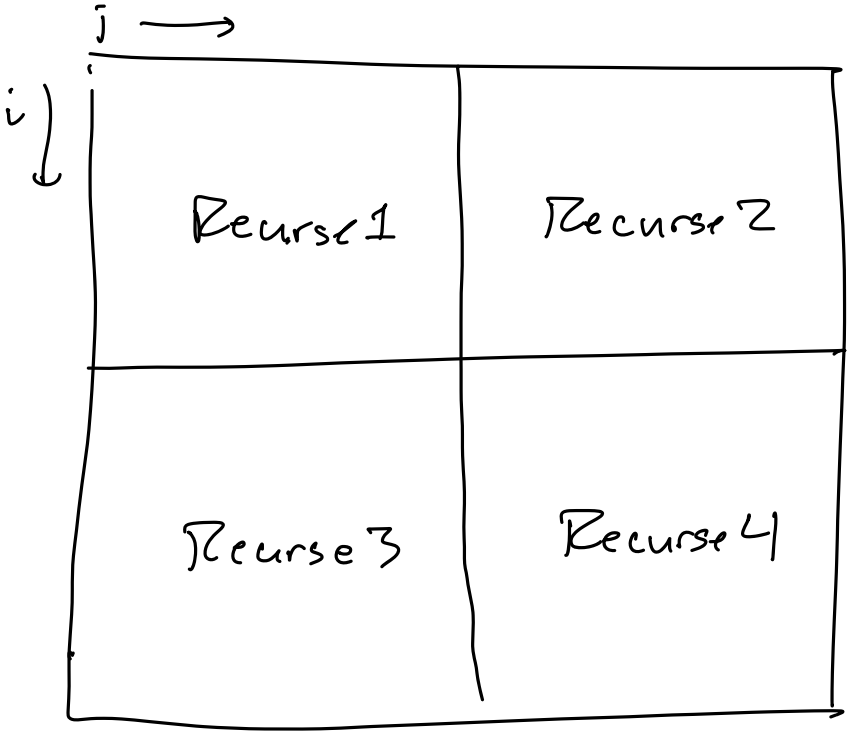


We can reconstruct opt. structure from the complete memo table

We can save space by keeping only one row of memo table

Hirschberg 197x

Choudhury Ramachandran 2005



$$T(n) = 4T\left(\frac{n}{2}\right) + O(n) \longrightarrow O(n^2)$$

$$S(n) = S\left(\frac{n}{2}\right) + O(n) \longrightarrow O(n)$$

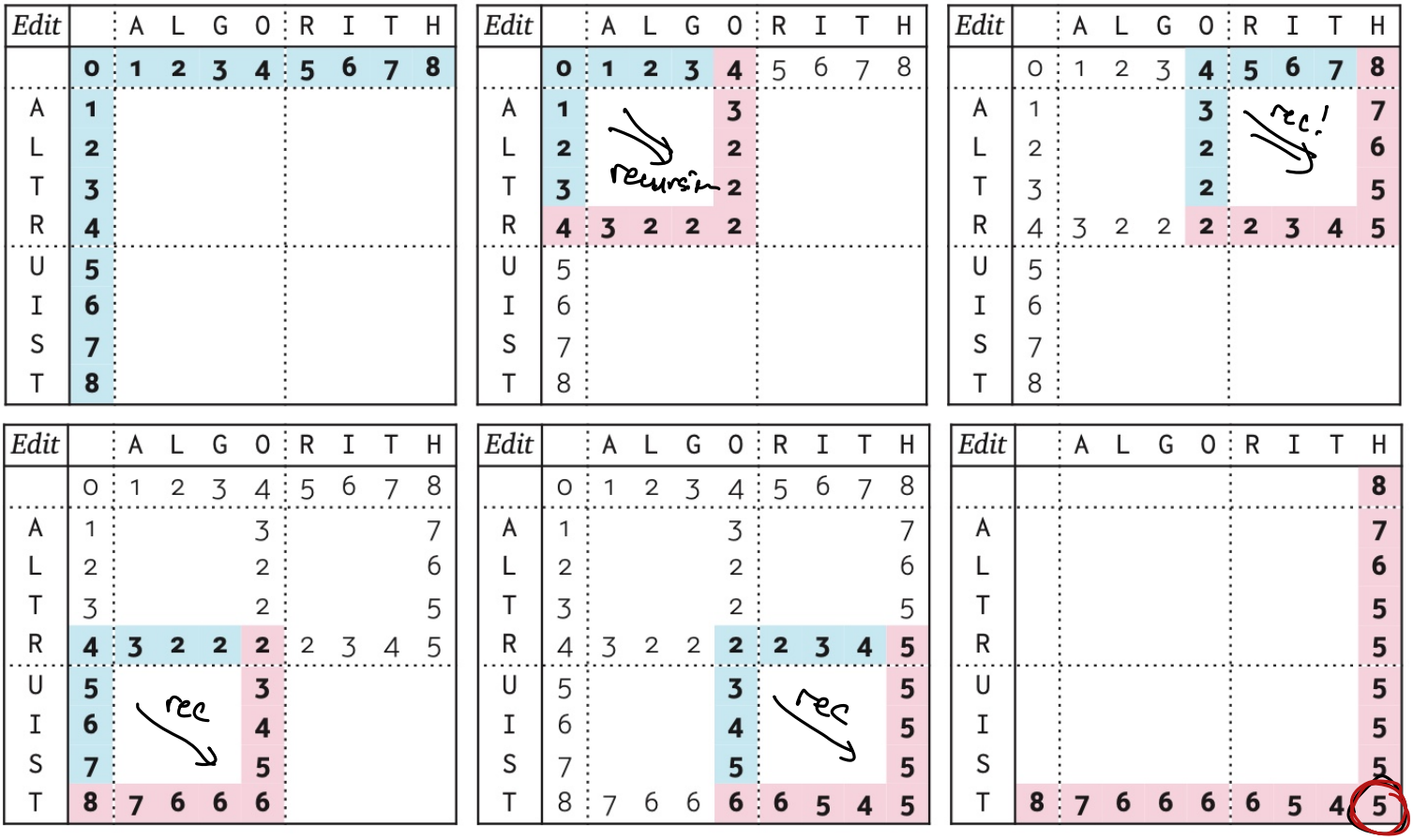


Figure D.2. Chowdhury and Ramachandran's edit distance algorithm: input, four recursive calls, and output.

```
EditBoundary(i, j, w, T[0..w], L[0..w]):
  if w = 1
      B[0] ← L[1];  R[0] ← T[1]
      compute B[1] using the edit distance recurrence
      R[1] ← B[1]
      return R[0..1], B[0..1]
  else
      T11 ← T[0..w/2];  T12 ← T[w/2..w]
      L11 ← L[0..w/2];  L21 ← L[w/2..w]
      T21, L12 ← EditBoundary(i,       j,       w/2, T11, L11)
      T22, R12 ← EditBoundary(i,       j + w/2, w/2, T12, L12)
      B21, L22 ← EditBoundary(i + w/2, j,       w/2, T21, L21)
      B22, R22 ← EditBoundary(i + w/2, j + w/2, w/2, T22, L22)
      return B21 · B22, R12 · R22      ⟨⟨Concatenation⟩⟩
```

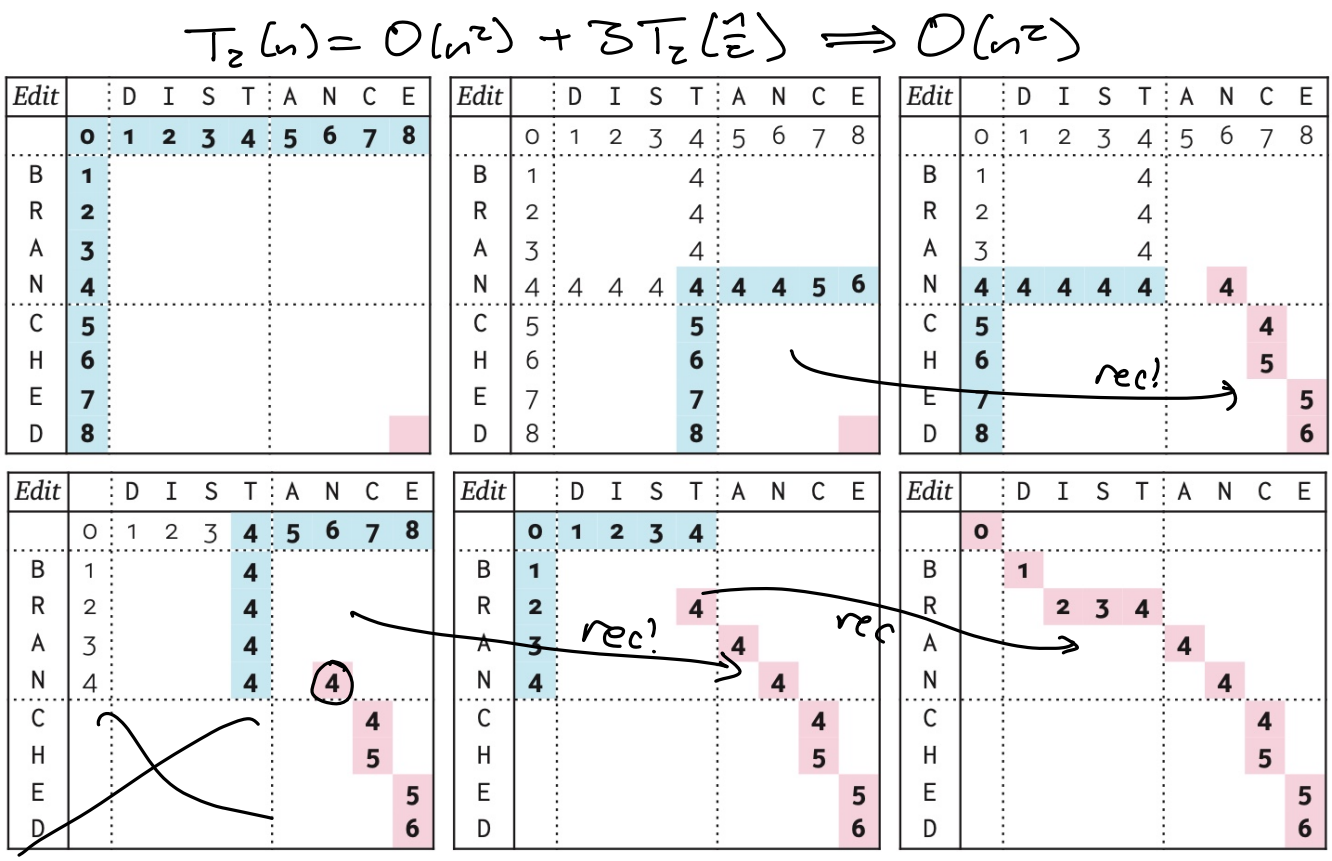


Figure D.4. Chowdhury and Ramachandran's algorithm in action: Input, EDITBOUNDARY calls in three quadrants, recursive calls in all four quadrants, and output. The bottom left recursive call is trivial.

```
EDITSEQUENCE(i, j, w, T[0..w], L[0..w], ti, tj):
  if w = 1
    ⟨⟨Base case: Brute force⟩⟩
    compute operation Z and indices si, sj using the edit distance recurrence
    return Z, si, sj
  else if (ti ≤ i or ti ≥ i + w or tj ≤ j or tj ≥ j + w or (ti < i + w and tj < j + w))
    ⟨⟨Trivial case: target indices are not on the outer block boundary⟩⟩
    return ε, ti, tj
  else
    ⟨⟨Forward phase: Set up inputs for recursive subproblems⟩⟩
    T11 ← T[0..w/2];  T12 ← T[w/2..w];
    L11 ← L[0..w/2];  L21 ← L[w/2..w]
    T21, L12 ← EDITBOUNDARY(i,       j,       w/2, T11, L11)
    T22, R12 ← EDITBOUNDARY(i,       j + w/2, w/2, T12, L12)
    B21, L22 ← EDITBOUNDARY(i + w/2, j,       w/2, T21, L21)
    ⟨⟨Backward phase: Recursively backtrack along the optimal edit path.⟩⟩
    ⟨⟨At most three of these recursive calls actually do anything.⟩⟩
    ⟨⟨Each nontrivial recursive call updates ti and tj.⟩⟩
    Z22, ti, tj ← EDITSEQUENCE(i + w/2, j + w/2, w/2, T22, L22, ti, tj)
    Z21, ti, tj ← EDITSEQUENCE(i + w/2, j,       w/2, T21, L21, ti, tj)
    Z12, ti, tj ← EDITSEQUENCE(i,       j + w/2, w/2, T12, L12, ti, tj)
    Z11, ti, tj ← EDITSEQUENCE(i,       j,       w/2, T11, L11, ti, tj)
    return Z11 · Z12 · Z21 · Z22, ti, tj
```