CSE 417 Algorithms and Complexity

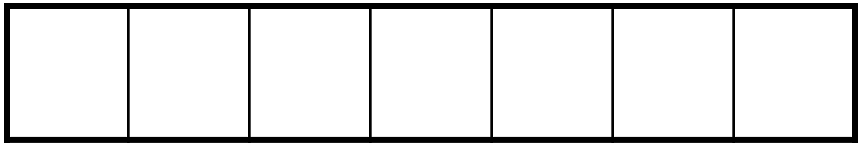
Richard Anderson

Lecture 20

Dynamic Programming

One dimensional dynamic programming: Interval scheduling

$$Opt[\ j\ ] = \max\ (Opt[\ j - 1], w_j + Opt[\ p[\ j\ ]\ ])$$



Billboard Placement

- Maximize income in placing billboards
  - $b_i = (p_i, v_i)$, v_i: value of placing billboard at position p_i
- Constraint:
  - At most one billboard every five miles
- Example
  - {(6,5), (8,6), (12, 5), (14, 1)}

Design a Dynamic Programming Algorithm for Billboard Placement

- Compute Opt[1], Opt[2], . . ., Opt[n]
- What is Opt[k]?

Input $b_1, \ldots, b_n$, where $b_i = (p_i, v_i)$, position and value of billboard i

$Opt[k] = fun(Opt[0],\ldots,Opt[k-1])$

- How is the solution determined from sub problems?

Input $b_1, \ldots, b_n$, where $b_i = (p_i, v_i)$, position and value of billboard i

Solution

```
j = 0; // j is five miles behind the current position
       // the last valid location for a billboard, if one placed at P[k]
for k := 1 to n
       while (P[ j ] < P[ k ] - 5)
              j := j + 1;
       j := j - 1;
       Opt[ k ] := Max(Opt[ k-1 ] , V[ k ] + Opt[ j ]);
```

Two dimensional dynamic programming

K-segment linear approximation

$Opt_k[\, j\,] = \min_i \{ Opt_{k-1}[\, i\,] + E_{i,j} \}$ for $0 < i < j$

4																	
3																	
2																	
1																	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Two dimensional dynamic programming

Subset sum and knapsack

$$Opt[\, j, K] = \max(Opt[\, j - 1, K], Opt[\, j - 1, K - w_j] + w_j)$$

$$Opt[\, j, K] = \max(Opt[\, j - 1, K], Opt[\, j - 1, K - w_j] + v_j)$$

4	0																
3	0																
2	0																
1	0																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Optimal line breaking

Element distinctness has been a particular focus of lower bound analysis. The first time-space tradeoff lower bounds for the problem apply to structured algorithms. Borodin et al. [13] gave a time-space tradeoff lower bound for computing ED on *comparison* branching programs of $T \in \Omega(n^{3/2}/S^{1/2})$ and, since $S \geq \log_2 n$, $T \in \Omega(n^{3/2}\sqrt{\log n}/S)$. Yao [32] improved this to a near-optimal $T \in \Omega(n^{2-\epsilon(n)}/S)$, where $\epsilon(n) = 5/(\ln n)^{1/2}$. Since these lower bounds apply to the average case for randomly ordered inputs, by Yao's lemma, they also apply to randomized comparison branching programs. These bounds also trivially apply to all frequency moments since, for $k \neq 1$, $ED(x) = n$ iff $F_k(x) = n$. This near-quadratic lower bound seemed to suggest that the complexity of ED and F_k should closely track that of sorting.

Optimal Line Breaking

- Words have length w_i, line length L
- Penalty related to white space or overflow of the line
  - Quadratic measure often used
- Pen(i, j): Penalty for putting w_i, w_{i+1},…,w_j on the same line
- Opt[k, m]: minimum penalty for ending line k with w_m

Optimal Line Breaking

Optimal score for ending line k on w_m

$$Opt_k[m] = \min_i \{ Opt_{k-1}[\ i\ ] + Pen(i+1,m)\} \text{ for } 0 < i < m$$

4																	
3																	
2																	
1																	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Longest Common Subsequence

- $C = c_1 \ldots c_g$ is a subsequence of $A = a_1 \ldots a_m$ if C can be obtained by removing elements from A (but retaining order)
- LCS(A, B): A maximum length sequence that is a subsequence of both A and B

`ocurranec`

`occurrence`

`attacggct`

`tacgacca`

Determine the LCS of the following strings

BARTHOLEMEWSIMPSON

KRUSTYTHECLOWN

String Alignment Problem

- Align sequences with gaps

CAT TGA AT

CAGAT AGGA

- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

Note: the problem is often expressed as a minimization problem,
with $\gamma_{xx} = 0$ and $\delta_x > 0$

LCS Optimization

- $A = a_1a_2\ldots a_m$
- $B = b_1b_2\ldots b_n$

- Opt[j, k] is the length of $LCS(a_1a_2\ldots a_j, b_1b_2\ldots b_k)$

Optimization recurrence

If $a_j = b_k$, Opt[j,k] = 1 + Opt[j-1, k-1]

If a_j != b_k, Opt[j,k] = max(Opt[j-1,k], Opt[j,k-1])

Give the Optimization Recurrence for the String Alignment Problem

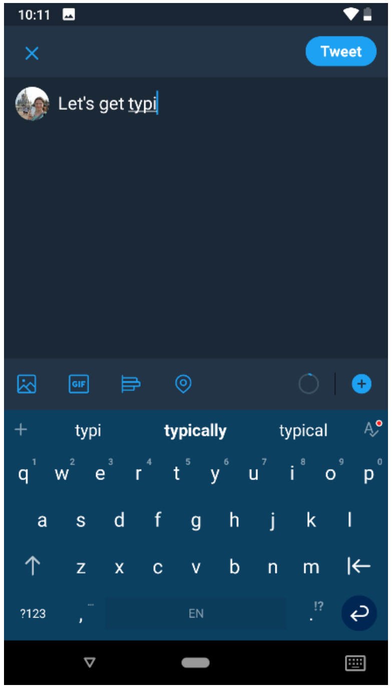
- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

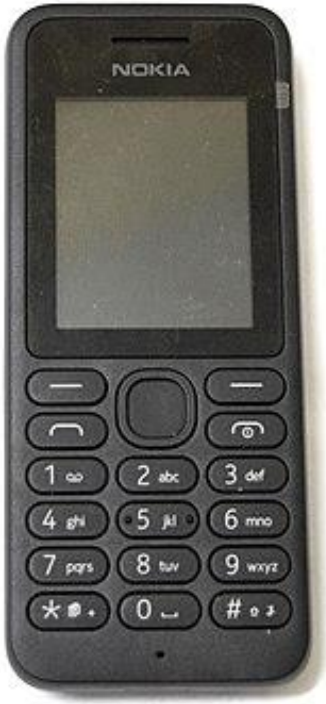
Opt[j, k] =

Let $a_j = x$ and $b_k = y$
Express as minimization

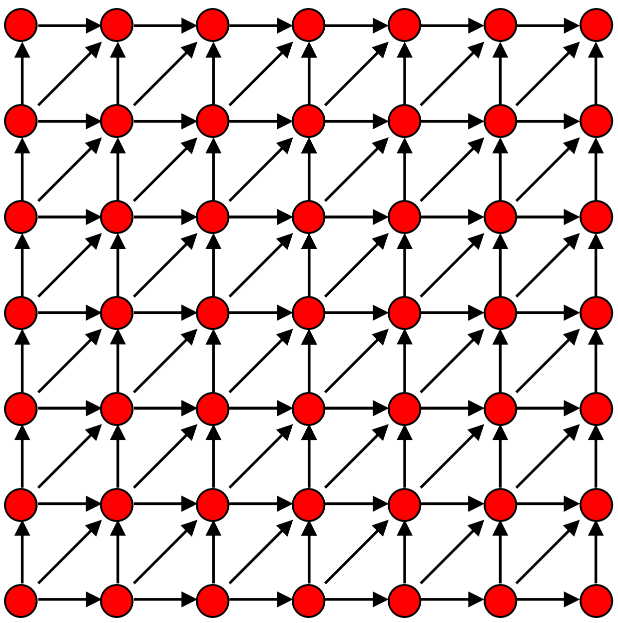
String edit with Typo Distance

- Find closest dictionary word to typed word
- Dist('a', 's') = 1
- Dist('a', 'u') = 6
- Capture the likelihood of mistyping characters
- Different distance model for T9 on basic mobile phone





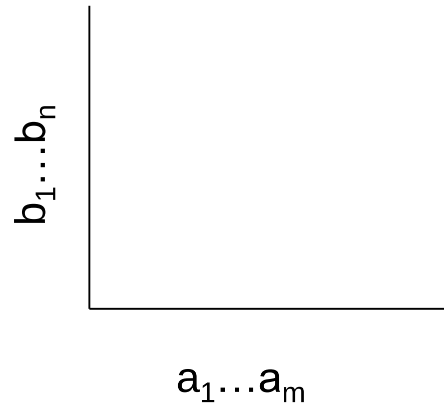
Dynamic Programming Computation



Code to compute Opt[n, m]

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++)
    if (A[ i ] == B[ j ] )
      Opt[ i,j ] = Opt[ i-1, j-1 ] + 1;
    else if (Opt[ i-1, j ] >= Opt[ i, j-1 ])
       Opt[ i, j ] := Opt[ i-1, j ];
    else
       Opt[ i, j ] := Opt[ i, j-1];
```

Storing the path information




```
A[1..m],  B[1..n]
for i := 1 to m     Opt[i, 0] := 0;
for j := 1 to n     Opt[0,j] := 0;
Opt[0,0] := 0;
for i := 1 to m
          for j := 1 to n
                    if A[i] = B[j]  {  Opt[i,j] := 1 + Opt[i-1,j-1];  Best[i,j] := Diag; }
                    else if Opt[i-1, j] >= Opt[i, j-1]
                              {  Opt[i, j] := Opt[i-1, j], Best[i,j] := Left; }
                    else      {  Opt[i, j] := Opt[i, j-1], Best[i,j] := Down; }
```

Reconstructing Path from Distances

```
LCS_Arguments
 211031321102033212120000321302
 100222010121310130323233121011
_0 _0 _0 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1
_1 _1 _1 _1 _1 _1 _1 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2
_1 _1 _1 _1 _1 _1 _1 _2 _2 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3
_1 _2 _2 _2 _2 _2 _2 _2 _3 _3 _3 _3 _3 _3 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4
_1 _2 _2 _2 _2 _2 _2 _2 _3 _3 _3 _3 _4 _4 _4 _4 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5
_1 _2 _2 _2 _2 _2 _2 _3 _3 _4 _4 _4 _4 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _6 _6 _6 _6 _6 _6
_1 _2 _2 _2 _2 _2 _2 _3 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6
_1 _2 _2 _3 _3 _3 _3 _3 _3 _4 _5 _5 _5 _5 _5 _5 _6 _6 _6 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7
_1 _2 _2 _3 _3 _3 _3 _4 _4 _4 _5 _6 _6 _6 _6 _6 _6 _6 _6 _7 _7 _7 _7 _7 _8 _8 _8 _8 _8 _8
_1 _2 _2 _3 _3 _3 _3 _4 _4 _5 _5 _6 _6 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _8 _8 _9 _9 _9 _9
_1 _2 _3 _3 _3 _3 _4 _4 _5 _5 _5 _6 _6 _7 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _9 10 10 10
_1 _2 _3 _4 _4 _4 _4 _4 _5 _5 _6 _6 _6 _7 _8 _8 _8 _8 _8 _9 _9 _9 _9 _9 _9 _9 _9 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _6 _7 _8 _8 _8 _9 _9 _9 _9 _9 _9 _9 _9 _9 _9 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 10 10 10 10 10 10 10 10 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 10 11 11 11 11 11 11 11 11 11 11
_1 _2 _3 _4 _5 _5 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 11 11 12 12 12 12 12 12 12 12 12
_1 _2 _3 _4 _5 _5 _5 _6 _6 _6 _6 _7 _7 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 13 13 13 13 13
_1 _2 _3 _4 _5 _6 _6 _6 _6 _6 _7 _7 _7 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 14 14 14 14
_1 _2 _3 _4 _5 _6 _6 _7 _7 _7 _7 _8 _8 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 15 15 15 15
_1 _2 _3 _4 _5 _6 _6 _7 _7 _7 _8 _8 _8 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 15 15 15 15
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _9 _9 _9 _9 10 10 11 11 12 12 13 13 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _9 _9 _9 _9 _9 _9 10 10 11 12 12 13 13 13 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _8 _8 _9 _9 10 10 10 10 10 10 10 11 12 12 13 13 13 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _8 _9 _9 10 11 11 11 11 11 11 11 12 13 13 14 14 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _9 _9 _9 10 11 11 12 12 12 12 12 12 13 13 14 14 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _9 _9 10 10 11 11 12 12 12 12 12 13 13 14 14 14 14 15 15 16 17 17
```

How good is this algorithm?

- Is it feasible to compute the LCS of two strings of length 300,000 on a standard desktop PC? Why or why not.