Dynamic programming (Cormen et al.)

- Optimal substructure: The optimal solution to the problem contains within it subsolutions, i.e., optimal solutions to subproblems.
- Overlapping subsolutions: The subsolutions overlap. These subsolutions are computed over and over again when computing the global optimal solution in a brute-force algorithm.
- Examples: Fibnnacci numbers,Binomial Coefficient and internet search (information retrieval): Levensthein distance.
- Take notes on Fibonaccy and Binomial Coefficient example

Basic Examples

Example 1: Fibonacci numbers (take notes on code!)

Recurrence equation:

F(0) = 0, F(1) = 1
F(n) = F(n-1) + F(n-2) [when n is greater than or equal to 2]

Example 2: Binomial Coefficients (take notes on code!)

Recurrence equation:

C(n,0) = C(n,n) = 1
C(n,k) = C(n-1,k-1) + C(n-1,k) [for n > k > 0]

Example 3: Levenshtein distance

- Correcting user queries for information retrieval applications (google search etc.)

Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of "correct words" from which the correct spellings come.
- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.
- Simple spelling correction algorithm: return the "correct" word that has the smallest distance to the misspelled word.
- Example: *informaton* → *information*
- For the list of correct words, we can use the vocabulary of all words that occur in our collection.
- Why is this problematic? (mis-spellings....)

Alternatives to using the term vocabulary

- A standard dictionary (Webster’s, OED etc.)
- An industry-specific dictionary (for specialized IR systems)
- The term vocabulary of the collection, appropriately weighted

Edit distance

- The edit distance between string s_1 and string s_2 is the *minimum* number of basic operations that convert s_1 to s_2.
- Levenshtein distance: The admissible basic operations are insert, delete, and replace
- Levenshtein distance *dog-do*: 1 (Deletion)
- Levenshtein distance *cat-cart*: 1 (Insertion)
- Levenshtein distance *cat-cut*: 1 (Replacement)
- Levenshtein distance *cat-act*: 2 (Replacements)

Levenshtein distance: Computation

		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
for i ← 0 to |s1|
do m[i, 0] = i
for j ← 0 to |s2|
do m[0, j] = j
for i ← 1 to |s1|
do for j ← 1 to |s2|
     do if s1[i] = s2[j]
          then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
          else m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

LevenshteinDistance(s_1, s_2)

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
  do if $s_1[i] = s_2[j]$
    then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
    else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
for i ← 0 to |s1|
do m[i, 0] = i
for j ← 0 to |s2|
do m[0, j] = j
for i ← 1 to |s1|
do for j ← 1 to |s2|
     do if s1[i] = s2[j]
          then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
          else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
for i ← 0 to |s1|
do m[i, 0] = i
for j ← 0 to |s2|
do m[0, j] = j
for i ← 1 to |s1|
do for j ← 1 to |s2|
      do if s1[i] = s2[j]
            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
for i ← 0 to |s1|
do m[i, 0] = i
for j ← 0 to |s2|
do m[0, j] = j
for i ← 1 to |s1|
do for j ← 1 to |s2|
    do if s1[i] = s2[j]
        then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
        else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Example

			f		a		s		t	
		0	1	1	2	2	3	3	4	4
c		1	1	2	2	3	3	4	4	5
		1	2	1	2	2	3	3	4	4
a		2	2	2	1	3	3	4	4	5
		2	3	2	3	1	2	2	3	3
t		3	3	3	3	2	2	3	2	4
		3	4	3	4	2	3	2	3	2
s		4	4	4	4	3	2	3	3	3
		4	5	4	5	3	4	2	3	3

Each cell of Levenshtein matrix

cost of getting here from my upper left neighbor (copy or replace)	cost of getting here from my upper neighbor (delete)
cost of getting here from my left neighbor (insert)	the minimum of the three possible "movements"; the cheapest way of getting here

Levenshtein distance: Example

			f		a		s		t	
		0	1	1	2	2	3	3	4	4
c		1	1	2	2	3	3	4	4	5
		1	2	1	2	2	3	3	4	4
a		2	2	2	1	3	3	4	4	5
		2	3	2	3	1	2	2	3	3
t		3	3	3	3	2	2	3	2	4
		3	4	3	4	2	3	2	3	2
s		4	4	4	4	3	2	3	3	3
		4	5	4	5	3	4	2	3	3

Invariant: transform the initial segment s[1..i] into t[1..j] using a minimum of d[i,j] operations.

This invariant holds:

It is initially true on row and column 0 because s[1..i] can be transformed into the empty string t[1..0] by dropping all i characters. Similarly, we can transform s[1..0] to t[1..j] by simply adding all j characters.If s[i] = t[j], and we can transform s[1..i-1] to t[1..j-1] in k operations, then we can do the same to s[1..i] and just leave the last character alone, giving k operations.

- Otherwise, the distance is the minimum of the three possible ways to do the transformation:

- If we can transform `s[1..i]` to `t[1..j-1]` in `k` operations, then we can simply add `t[j]` afterwards to get `t[1..j]` in `k+1` operations (insertion).
- If we can transform `s[1..i-1]` to `t[1..j]` in `k` operations, then we can remove `s[i]` and then do the same transformation, for a total of `k+1` operations (deletion).
- If we can transform `s[1..i-1]` to `t[1..j-1]` in `k` operations, then we can do the same to `s[1..i]`, and exchange the original `s[i]` for `t[j]` afterwards, for a total of `k+1` operations (substitution).

- The operations required to transform `s[1..n]` into `t[1..m]` is of course the number required to transform all of `s` into all of `t`, and so `d[n,m]` holds our result.

This proof fails to validate that the number placed in `d[i,j]` is in fact minimal; this is more difficult to show, and involves an argument by contradiction in which we assume `d[i,j]` is smaller than the minimum of the three, and use this to show one of the three is not minimal.

Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., *m* more likely to be mistyped as *n* than as *q*.
- Therefore, replacing *m* by *n* is a smaller edit distance than by *q*.
- We now require a weight matrix as input.
- Modify dynamic programming to handle weights

Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of “correct” words
- Then suggest terms in the intersection to the user.
- → exercise in a few slides

Exercise

1. Compute Levenshtein distance matrix for OSLO – SNOW
2. What are the Levenshtein editing operations that transform *cat* into *catcat*?

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1				
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 ?			
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1			
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 ?		
s	2 / 2				
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2		
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 ?	
s	2 2				
l	3 3				
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4		
		1	2	1	2	2	3	2		
s		2								
		2								
l		3								
		3								
o		4								
		4								

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	?
s		2								
		2								
l		3								
		3								
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2				
l	3 3				
o	4 4				

		s		n		o		w	
	0	1	1	2	2	3	3	4	4
o	1 1	1 2	2 1	2 2	3 2	2 3	4 2	4 3	5 3
s	2 2	1 3	2 ?						
l	3 3								
o	4 4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1			
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 ?		
l	3 3				
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3				
		2	3	1	2	2				
l		3								
		3								
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 ?	
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	4 5 / 3 3
s	2 / 2	1 2 / 3 1	2 3 / 2 2	3 3 / 3 3	
l	3 / 3				
o	4 / 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	?
l		3								
		3								
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3				
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2						
		3	4	?						
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2			
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3				
		3	4	2	3	?				
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2		
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4		
		3	4	2	3	2	3	?		
o		4								
		4								

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	
o	4 4				

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	?
o		4								
		4								

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4								
		4								

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3						
		4	5	?						

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
		3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o		**4**	4	**3**						
		4	5	**3**						

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3	3	3				
		4	5	3	4	?				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3		

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3	3	3	2	4		
		4	5	3	4	3	4	?		

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
		3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o		**4**	4	**3**	**3**	**3**	**2**	4		
		4	5	**3**	4	**3**	4	**2**		

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
		3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o		**4**	4	**3**	**3**	**3**	**2**	4	4	5
		4	5	**3**	4	**3**	4	**2**	3	?

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3	3	3	2	4	4	5
		4	5	3	4	3	4	2	3	3

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3	3	3	2	4	4	5
		4	5	3	4	3	4	2	3	3

Outline

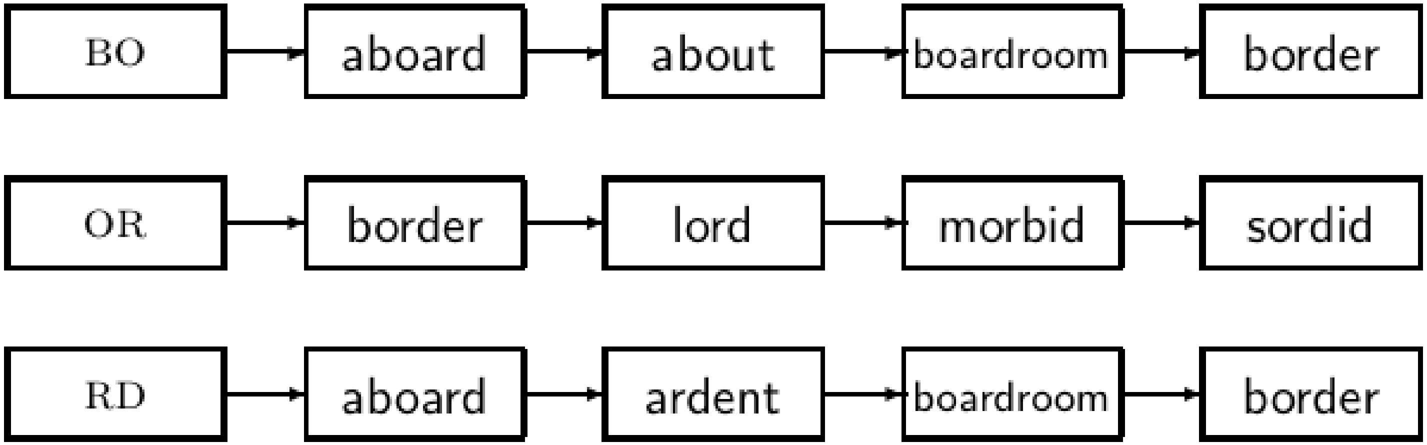
Spelling correction

- Now that we can compute edit distance: how to use it for isolated word spelling correction – this is the last slide in this section.
- *k*-gram indexes for isolated word spelling correction.
- Context-sensitive spelling correction
- General issues

k-gram indexes for spelling correction

- Enumerate all *k*-grams in the query term
- Example: bigram index, misspelled word bordroom
- Bigrams: *bo, or, rd, dr, ro, oo, om*
- Use the *k*-gram index to retrieve "correct" words that match query term *k*-grams
- Threshold by number of matching *k*-grams
- E.g., only vocabulary terms that differ by at most 3 *k*-grams

k-gram indexes for spelling correction: *bordroom*



Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: hit-based spelling correction
  - Retrieve "correct" terms close to each query term
  - *for flew form munich: flea for flew, from for form, munch for*
  - *munich*
  - Now try all possible resulting phrases as queries with one word "fixed" at a time
  - Try query *"flea form munich"*
  - Try query *"flew from munich"*
  - Try query *"flew form munch"*
  - The correct query *"flew from munich"* has the most hits.
- Suppose we have 7 alternatives for *flew*, 20 for form and 3 for *munich*, how many "corrected" phrases will we enumerate?

Context-sensitive spelling correction

- The “hit-based” algorithm we just outlined is not very efficient.
- More efficient alternative: look at “collection” of queries, not documents